BERGISCHE
UNIVERSITÄT
WUPPERTAL

Optimierung von Networks-on-Chip mit Hilfe der dynamischen Rekonfiguration

Dem Fachbereich Elektrotechnik, Informationstechnik, Medientechnik
der Bergischen Universität Wuppertal
zur Erlangung des akademischen Grades eines Doktor-Ingenieurs
genehmigte Dissertation

von
M. Sc. -Ing. Alexander Logvinenko

Wuppertal, den 08. September 2014

Die Dissertation kann wie folgt zitiert werden:

urn:nbn:de:hbz:468-20150623-102353-7
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20150623-102353-7]

Kurzfassung

Moderne Prozessoren werden modular entwickelt. Dadurch muss bei Neuentwicklungen nicht alles neu geplant werden, sondern es kann auf bereits vorgefertigte Komponenten – sogenannte IP Blocks (Intellectual Property Blocks) – zurückgegriffen werden.

Um die einzelnen IPs miteinander verbinden zu können, wurde eine rekonfigurationsfähige Architektur für Network-on-Chip entwickelt: Reconfigurable Multi-Interconnection Network (RecMIN). Diese ist in der Lage, sich an die aktuelle Datenverkehrssituation des Netzes anzupassen.

Die entwickelte Architektur besteht aus einzelnen rekonfigurierbaren Zellen. Zusätzlich weist die Rekonfigurationsarchitektur einen höheren Wiederverwendungsgrad für die Hardware-Strukturen auf, z.B. bzgl. Puffer, Multiplexer. Das bedeutet, dass die Komponenten der alten Topologie auch nach der Rekonfiguration für die neue Topologie aktiv verwendet werden.

Um die Simulation der Rekonfigurationsarchitektur zu ermöglichen, wurde ein Simulationsporgramm *RecSim* implementiert.

Zum Schluss werden drei Algorithmen präsentiert, die das Verhalten des RecMIN beobachten und bei Bedarf eine Rekonfiguration durchführen. Auf diese Weise soll die Netztopologie auch bei Belastungsänderungen optimal an den Datenverkehr angepasst bleiben.

Die Ergebnisse der Simulation haben gezeigt, dass die hier vorgeschlagene rekonfigurierbare Netzwerkstruktur das Potential hat, die Leistungsfähigkeit einiger Anwendungen, insbesondere derer, die zwischen zwei oder mehreren Verkehrsprofilen wechseln, drastisch zu verbessern. So, z. B., würde diese Baustruktur die Paketverzögerungen bedeutend (bis zu Faktor sieben) reduzieren (s. Kapitel 6).

Für die Hardwarerealisierung der RecMIN-Architektur auf FPGA ist Xilinx-FPGA mit FIFO-Speicherprinzip (beim einfachen FIFO für die RecMINs mit dünnem Paketverkehr) als eine der besten Lösungen zu empfehlen. Wird aber für RecMIN ein Ringspeicher gewählt, was vor allem für einen starken Paketverkehr sinnvoll ist, so wäre die Wahl zwischen Altera- und Xilinx-FPGAs für jeden konkreten Fall mithilfe der Daten aus den Abbildungen 7.9 - 7.11 individuell zu treffen.

Der Vergleich zwischen den drei vorgeschlagenen Rekonfigurationsalgorithmen liefert folgende Schlussfolgerungen:

Der η-Algorithmus kann in Computersimulationen verwendet werden, um die von anderen Algorithmen (die keine erschöpfende Suche benutzen) erzielten Ergebnisse zu vergleichen. Jedoch ist das Integrieren des η-Algorithmus in ein SoC mit großem Hardware-Aufwand verbunden.

Der Minimal-Queues-Algorithmus (MQA) ist schneller als der η-Algorithmus und erfordert weniger Fläche auf dem Chip. Er kann mit kleinerem Hardware-Aufwand als der η-Algorithmus in SoC integriert werden. Der MQA ist dann zu empfehlen,

falls die Vorkenntnisse über die Eigenschaften des möglichen Verkehrs im Netzwerk nicht vorhanden sind.

Der Pattern-Identification-Algorithmus (PIA) ist in den meisten Fällen schneller als der η-Algorithmus und der Minimal-Queues-Algorithmus. Die Umsetzung des PIAs auf SoC ist relativ einfach. Im Falle, dass die Muster für den PIA optimal implementiert sind, ist der PIA von allen drei hier für RecMIN vorgeschlagenen Reconfigurationsalgorithmen die effizienteste Lösung.

Abstract

Modern processors are developed modularly. Therefore new developments need not be re-scheduled from the scratch; rather prefabricated components – so-called IP blocks (Intellectual Property blocks) – can be used.

In order to connect the individual IPs, a reconfigurable architecture for network-on-chip has been developed: Reconfigurable Multi-Interconnection Network (RecMIN). It is able to adapt to the current traffic situation of the network.

The developed architecture consists of single reconfigurable cells. In addition, the reconfiguration architecture has a higher degree of reuse for the hardware structures, such as regarding buffer, multiplexer. This means that the components of the previously active topology are actively used also after the reconfiguration to the new topology.

In order to enable the simulation of the reconfiguration architecture, a simulation program *RecSim* was implemented.

Finally, three algorithms are presented that observe the behavior of the RecMIN and perform a reconfiguration when needed. In this way the network topology is to remain optimally adapted to the traffic even if the load changes.

The results of the simulation showed that the proposed reconfigurable network structure has the potential to improve the performance of some applications, especially those which switch between two or more traffic profiles. For some applications RecMIN-architecture would significantly reduce paket delays (up to factor seven) (see chapter 6).

For the hardware realization of the RecMIN-architecture on FPGA a Xilinx-FPGA with FIFO-buffer concept is recommended, as one of the best solutions for the RecMin realization. However, if circular buffer is chosen for RecMIN, which mostly makes sense for high packet traffic, the choice between Altera- and Xilinx-FPGAs would have to be made individually, for each specific case, based on the data from the images 7.9 – 7.11.

The comparison between the three suggested reconfiguration algorithms provides the following conclusion:

η-algorithm can be used in computer simulations to compare the results achieved from other algorithms However, integrating the η-algorithm into a SoC is attended with hardware complexity.

Minimal-Queues-Algorithm (MQA) is faster than the η-algorithm and requires less area on the SoC. It can be integrated into SoC with less hardware complexity than η-algorithm. The MQA is a best solution in case that foreknowledge about the nerwork traffic are not available.

The Pattern-Identification-Algorithm (PIA) is faster than the η-algorithm and the Minimal-Queue-Algorithm. In case of properly implementation of the pattern for the PIA, the PIA is the most efficient of three algorithms proposed in this work.

Inhaltsverzeichnis

1 Einleitung

Im Jahre 1965 formulierte einer der Gründer von Intel das sogenannte Mooresche Gesetz. Moore stellte die Behauptung auf, dass sich die Anzahl der Transistoren eines Chips alle 24 Monate verdoppelt. Die Forschung im Bereich der Hochintegrationstechnik sorgt dafür, dass diese Aussage auch heute noch gültig ist [33]. Daraus folgt, dass immer mehr Komponenten auf einem einzelnen Chip untergebracht werden können. Die heutige Technik ermöglicht es bereits, dass vollständige Systeme mit digitalen und analogen Komponenten auf einem einzelnen Chip abgebildet werden können. Solche Systeme werden als Systems-on-Chip (SoC) bezeichnet.

Moderne Prozessoren werden modular entwickelt. Dadurch muss ein Entwickler nicht „bei null“ anfangen, sondern kann bereits auf vorgefertigte Komponenten – sogenannte Intellectual Property Blocks (IP Blocks) – zurückgreifen. Anschließend müssen nur die fehlenden Hardware-Einheiten ergänzt werden. Durch die leichte Austauschbarkeit der IPs ist es möglich diese separat weiterzuentwickeln. Dies ermöglicht ebenfalls das schnelle Ausmachen sogenannter Bottlenecks und deren Beseitigung.

Die dabei auftretende Problemstellung, wie man die einzelnen IPs miteinander verbinden kann, wurde bis vor kurzem üblicherweise durch einen oder mehrere interne Busse gelöst. Da allerdings die Anzahl der verwendeten IPs innerhalb eines Chipkerns stetig ansteigt, kann ein solcher Bus nicht mehr den benötigten Datendurchsatz gewährleisten. Eine andere Problemlösung muss also gefunden werden.

Diese Arbeit behandelt daher die Modellierung, Implementierung und Simulation eines Netzes zur schnellen Kommunikation zwischen den IPs, welches außerdem möglichst wenig Chipfläche zur Umsetzung benötigen soll. Als Schwerpunkt ist dabei die Rekonfiguration eines solchen Netzes zu betrachten.

1.1 Motivation

Die durch das Schalten der Transistoren und durch Leckströme zwischen Emitter und Gate der Feldtransistoren entstehende Wärme stellt das Hauptproblem bei der Erhöhung der Frequenz von Einkernprozessoren dar. Mit den heutigen Fertigungstechniken ist es nicht möglich diese weiter zu verringern. Der Grund dafür ist, dass die Erhöhung der Schaltfrequenz den Leistungsverbrauch und somit auch die Abwärme steigert. Spezielle thermische Chip-Designs sowie der Einsatz von aktiven

Kühltechniken reduzieren zwar das Problem [51], bieten aber keine Komplettlösung. Aus diesem Grund beendete auch Intel im Jahre 2004 die Entwicklung des Prozessorkerns Tejas mit Taktfrequenzen von 4,4 Ghz.

Dies führte in der Industrie zu einem Umdenken: Anstatt die Taktfrequenz eines einzelnen Kerns weiter zu steigern, wird auf Mehrkernsysteme gesetzt [15], [38]. Diese Chips besitzen mehrere parallel arbeitende Prozessorkerne (Multicore-Prozessoren). Dies hat den Vorteil, dass die Anzahl der auf dem Chip eingesetzten Kerne vergrößert werden kann, ohne dabei wesentliche Kühlungsprobleme zu verursachen. State of the Art ist dabei momentan die von Intel geführte Entwicklung des Teraflop Research Chip mit 80 Kernen [4].

Den eigentlichen Rechenvorgang durch die Verwendung von mehreren Prozessorkernen zu beschleunigen ist allerdings nicht neu. Schon seit Jahren versucht man Parallelcomputer mit mehreren Rechnereinheiten herzustellen. Die Durchbrüche in der Hochintegration der letzten Zeit ermöglichen nun „nur“ das Umsetzen dieser Idee auf einem einzelnen Chip. Dies bietet gegenüber dem Einsatz von mehreren Prozessoren in Parallelrechner mehrere Vorteile. Dazu zählen beispielsweise die Möglichkeit die Verbindungsleitungsanzahl wesentlich zu erhöhen und die Verbindungslänge zu verringern. Dies führt wiederum dazu, dass die benötigte Leistungsaufnahme geringer ist, während die Übertragungsraten steigen.

Wie bei allen parallelen Systemen tritt auch die Kommunikation zwischen diesen einzelnen Kernen als ein entscheidender Faktor für die Gesamteffizienz des Systems auf. Können die benötigten Daten zwischen den einzelnen Kernen nicht schnell genug übertragen werden, ist die Auslastung der Kerne nicht optimal. Dadurch entsteht ein Geschwindigkeitsverlust für das gesamte System.

Diese äußerst wichtige Verbindungsstruktur sollte demnach folgende Kriterien erfüllen:

- Die Paket-Verzögerung (Delay) muss möglichst gering sein. Die Paket-Verzögerung ist die Zeit, die benötigt ist um ein Informationspaket vom Sender zum Empfänger zu übertragen. Für getaktetes Network-on-Chip kann Paket-Verzögerung als die Anzahl der Taktzyklen, die ein Paket braucht um durch das Netzwerk durchzugehen, definiert werden.

- Der Durchsatz (Throughput) von der Verbindungsstruktur muss maximal sein. Für Network-on-Chip spiegelt der Durchsatz die Wahrscheinlichkeit des Ankommens eines Pakets an einem bestimmten Netzwerk-Ausgang pro Taktzyklus wieder.

- Die von der Struktur benötigte Chipfläche muss so gering wie möglich sein.

- Die Struktur muss flexibel, und leicht zu erweitern sein, um das Hinzufügen neuer (oder anderer) Komponenten zu ermöglichen.

Wie bereits erwähnt, wurde dies in der Vergangenheit durch einen Bus realisiert, welcher aber den heutigen Anforderungen bezüglich des Datendurchsatzes nicht mehr gerecht wird. Erste mögliche Lösung wäre der Einsatz einer Punkt-zu-Punkt-Kommunikation. Leider steigt bei dieser Variante der Verbindungsaufwand exponentiell mit der Anzahl der Kerne bzw. IPs. Ebenso ist es nur schwer möglich ein solches System zu erweitern, da dabei bereits vorhandene Elemente in der Regel angepasst werden müssen. Die Lösung dieser Probleme wird durch ein Network-on-Chip (NoC) präsentiert.

Wenn die zur Verfügung stehenden Hardwareressourcen optimal genutzt und eine unsymmetrische Belastung vermieden werden soll, muss die Anpassung des Netzwerkes an die aktuelle Situation als ein bedeutender Faktor anerkannt werden. Dieser Punkt soll auch als Schwerpunkt dieser Arbeit betrachtet werden.

1.2 Beitrag und Einordnung

Die im vorigen Abschnitt erwähnten Punkte stellen die Motivation zur Lösung der Aufgabenstellung dieser Arbeit dar. Somit ist das Ziel eine rekonfigurationsfähige Architektur für Network-on-Chip zu entwickeln, welche in der Lage ist, sich an die „Datenverkehrssituation“ des Netzes anzupassen. D.h. sollten in den Netzwerk Bottlenecks entstehen, so muss die Netzwerk-Architektur in die Lage sein, durch die Rekonfiguration seine Topologie auf die Art und Weise zu ändern, dass nach Möglichkeit die Datenpaketeverzögerung erniedrigt wird, und gleichzeitig die Durchsatzkapazität des Netzwerkes so hoch wie möglich gehalten wird.

Die gesuchte Architektur muss aus einzelnen reconfigurierbaren Zellen bestehen (ähnlich wie es bei FPGA-Architekturen[1] der Fall ist). Dies soll dem Entwickler die Möglichkeit geben, die vorgeschlagene Architektur für Netzwerke mit beliebiger Sender- /Empfänger-Anzahl zu nutzen. Zusätzlich muss die Rekonfigurationsarchitektur einen höheren Wiederverwendungsgrad für die Hardware-Strukturen (Puffer, Multiplexer) aufweisen. D. h., die Komponenten der einen Topologie müssen auch nach der Rekonfiguration für die andere Topologie aktiv verwendet werden. Es muss auch möglich sein, das Netzwerk während der Rekonfiguration vollständig oder teilweise im Betrieb zu halten. (Alternativ wäre das Netzwerk komplett von den Datenpaketen zu lehren und erst danach den Rekonfigurationsprozess zu starten).

Um die Simulation und Bewertung der Rekonfigurationsarchitektur zu ermöglichen, muss weiterhin ein Simulationsporgramm implementiert werden. Das Programm soll nicht nur in der Lage sein die Netzwerk-Architektur simulieren zu können, sondern auch mithilfe von Statistiken das Verhalten des Netzwerkes unter verschiedenen Auslastungen beurteilen zu können.

[1] Field Programmable Gate Array (FPGA)

Zum Schluss müssen die Algorithmen entwickelt werden, die das Verhalten des Netzwerkes beobachten und bei Bedarf eine Rekonfiguration durchführen. Auf diese Art und Weise soll die Netzwerk-Topologie, auch bei Belastungsänderung, optimal an den Datenpaket-Verkehr angepasst bleiben.

1.3 Aufbau

Die Arbeit ist folgendermaßen aufgebaut:

An diese Einleitung schließt sich Kapitel 2 an, in dem Aufbau, Funktionsweise sowie Ablauf der Pakettransfers in einem typischen NoC kurz zusammengefasst werden. Es wird auf die Unterschiede zwischen einem Network-on-Chip und einem Inter-Chip-Network (z.B Internet oder LAN) genau eingegangen.

Kapitel 3 beschreibt die im Rahmen dieser Arbeit entwickelten Rekonfigurations-Techniken für mehrstufige NoC-Verbindungsnetzwerke. Um die Technicken für NoC anzuwenden, muss mit Multistage Interconnection Network (MIN) mit Banyan-Eigenschaften gearbeitet werden. Die MINs werden kurz vorgestellt, danach werden die Anpassungstechniken detailliert erläutert, ihre Eigenschaften werden diskutiert. Am Ende des Kapitels wird geklärt, wie die Puffer des Netzwerks zu teilen sind, damit das Netzwerk nach der Rekonfiguration möglichst hohen Paket-Durchsatz hat. Abschließend wird das optimale Puffer-Längen-Verhältnis zwischen der ersten und der zweiten MIN-Stufe gesucht.

Kapitel 4 stellt ein analytisches Modell für eine alleinstehende Router-Komponente mit Input-Buffering und einer unendlichen Puffergröße vor. Im Verlauf des Kapitels wird dieses Modell für ein Router mit endlichem Puffer weiterentwickelt. Anschließend wird ein Versuch unternommen, das Modell auf das komplette MIN zu erweitern, der allerdings an Komplexitätsgrenzen stößt: die resultierende Markov-Kette kann analytisch nicht gelöst werden, da jede MIN-Stufe die Markov-Kette um eine zusätzliche Dimension erweitert.

Nachdem gezeigt worden ist, dass ein analytisches Modell für die MIN-Topologie zu einer komplexen mehrdimensionale Markov-Kette führt, wird in Kapitel 5 der Simulator *RecSim* präsentiert. Mit *RecSim* ist es möglich, die Eigenschaften der verschiedenen MIN-Typologien zu erforschen, die Bottlenecks (Flaschenhälse) zu diagnostizieren und die Rekonfigurations-Algorithmen zu testen. Außerdem ermöglicht *RecSim* eine automatische Generierung der unsymmetrischen MINs beliebiger Größe und Komplexität, was die Stimulation der NoCs wesentlich erleichtert.

In Kapitel 6 wird die Rekonfigurations-Architektur RecMIN vorgestellt. RecMIN basiert auf Rekonfigurations-Techniken aus Kapitel 3, und wird mit Hilfe des in Kapitel 5 beschriebenen Simulators *RecSim* getestet. Bei der Entwicklung der RecMIN-Architektur, wird auf die in der Motivation beschriebenen Kriterien geachtet: modularer Aufbau, Möglichkeit der dynamischen Rekonfiguration während

des Betriebs etc.

Kapitel 7 zeigt eine Untersuchung, für die zukünftige Realisierung der RecMIN auf FPGAs. Es wird untersucht, welcher Puffer-Speicher-Typ (LIFO, FIFO, Ringspeicher) für das RecMIN am besten geeignet ist. Außerdem werden zwei FPGA-Produkte der momentan auf dem Markt dominierenden FPGA-Hersteller (Xilinx und Alreta) verglichen.

Schließlich stellt Kapitel 8 drei Algorithmen vor, die zur Beobachtung und dynamischer Anpassung des RecMIN benutzt werden können: den η-Algorithmus, den Minimal-Queues-Algorithmus und den Pattern-Identification-Algorithmus. Die Effizienz sowie Vor- und Nachteile des jeweiligen Algorithmus werden diskutiert.

Das letzte Kapitel fasst die gewonnenen Ergebnisse noch ein Mal zusammen und bietet einen Überblick über noch offene Probleme.

2 NoC im Vergleich zu Off-Chip Netzwerken

NoCs unterscheiden sich hauptsächlich in ihren Beschränkungen und ihrer Synchronisierung von Off-Chip Netzwerken. Normalerweise sind Ressourcen-Einschränkungen On-Chip strenger als Off-Chip. Außerdem ist der Speicherplatz On-Chip teurer, weil Datenspeicher mehr Platz einnimmt und so den Chip vergrößert. Darüber hinaus sind auch mathematische Berechnungen On-Chip kostspieliger, da sie zusätzliche IP-Einheiten auf dem Chip erfordern. Folglich sollte das Zwischenspeichern in NoCs begrenzt und die Berechnungen einfach gehalten werden.

On-Chip Leitungen sind zudem generell sehr viel kürzer als Off-Chip Leitungen, was eine viel dichtere Synchronisierung als Off-Chip ermöglicht. Diese Synchronisierung erlaubt eine Reduzierung des für die Zwischenspeicherung benötigten Platzes im Router, da die Kommunikation mit einer geringeren Granularität durchgeführt werden kann.

Treten Staus im Netzwerk auf, so gehen in den meisten Off-Chip Netzwerken Daten verloren. Nicht so in On-Chip Netzwerken. Würden die Daten auch in On-Chip Netzwerken verloren gehen, müsste die Kommunikation gesichert werden, was aus Kostengründen nicht praktikabel wäre. Da in On-Chip Netzwerke allerdings keine Daten verloren gehen, stellen sie eine sehr viel günstigere Lösung dar. Die Kehrseite bei der Vermeidung von Datenverlust ist, dass die Wahrscheinlichkeit eines Staus zunimmt. Die Architektur der Off-Chip Netzwerke hat in der Regel eine unregelmäßige Struktur, die Pufferzyklen einführt. In diesen Fällen kann ein Paketverlust notwendig sein, um einen Stau zu vermeiden. In NoCs können Staus durch die Verwendung von regulär strukturierten Architekturen verhindert werden.

In Off-Chip Modellen gibt es häufig reihenfolgevertauschende Datenübertragungen, weil verschiedene Pakete aus derselben Quelle mit demselben Ziel auf unterschiedlichen Wegen durch das Netzwerk folgen. On-Chip können Daten gezwungen werden, zwischen einer Quelle und einem Ziel immer denselben Weg im Netzwerk zu nehmen. Dies garantiert, dass die Daten in der Reihenfolge, in der sie gesendet wurden, auch zum Ziel gelangen. Auf diese Weise wird die Nutzung des Puffers im Vergleich zu den Off-Chip Netzwerken, bei denen eine Sendung gemäß der Reihenfolge garantiert werden soll, reduziert.

2.1 Network-on-Chip

Das NoC besteht aus zwei verschiedenen Elementen, wie in Abbildung 2.1 zu sehen ist: Netzwerk-Schnittstellen (NI) und Router (R). Die Netzwerk-Schnittstellen werden als Schnittstellen zwischen den IP-Blöcken und dem NoC verwendet. Sie wandeln die Informationen aus den IP-Blöcken in Informationen, die das NoC versteht (z.B Pakete), um. Ebenso werden die Informationen vom NoC in Informationen, die die IP-Blöcke verstehen, konvertiert (z.B OCP, AXI, DTL, welche drei Protokolle höheren Levels für die IPs von drei verschiedenen Firmen sind). Die Aufgabe der Router ist es, Daten von einer Netzwerk-Schnittstelle zu einer anderen zu transportieren.

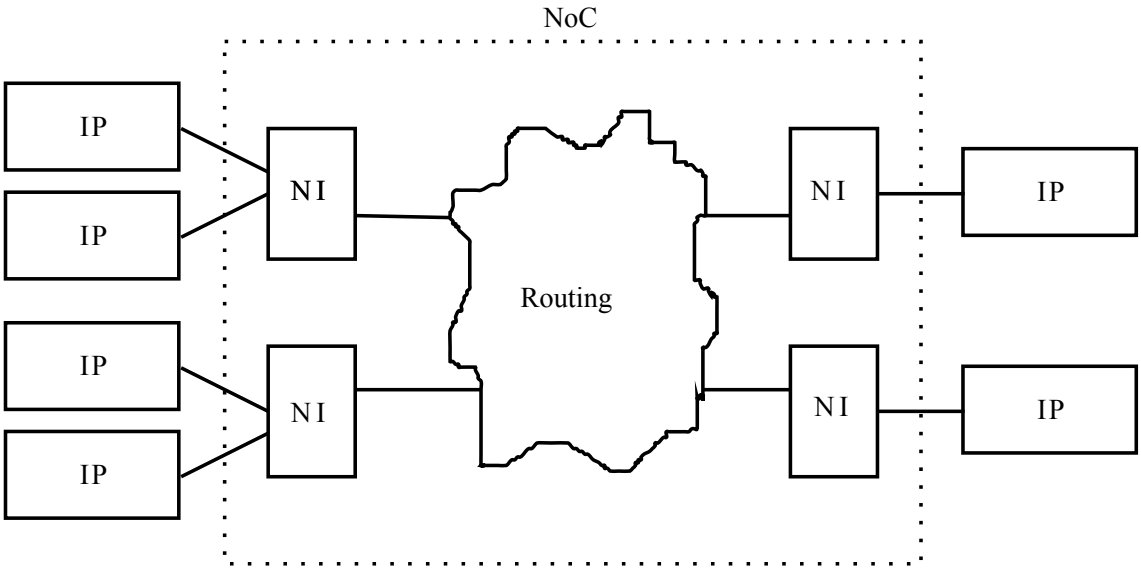

Abbildung 2.1: Das Verschalten des NoCs

Die IPs müssen keine konkrete Information über die Umsetzung der Kommunikation besitzen – es reicht, wenn klar ist, dass die Verbindung gewährleistet ist. Im allgemeinen ist das Routing durch die Router einer Verbindung gesteuert. Das bedeutet, dass die gesamte Kommunikation zwischen zwei IP-Blöcken immer durch denselben Weg von Routern gehen wird. Dieser Weg von Routern wird zu einer Modellzeit festgelegt. Bei derzeitigen Modellen dient die kürzeste Verbindung zwischen den zwei miteinander korrespondierenden Netzwerk-Schnittstellen als ebenjener Weg von Routern.

Netzwerk-Schnittstellen werden durch bidirektionale Verknüpfungen mit IP-Blöcken und Routern verbunden. Die Datenübertragung über mehrere Verbindungen tritt gleichzeitig auf, weil On-Chip Netze in der Regel mit einem Takt synchronisiert werden.

Die Kommunikationsleistung zwischen den IP-Kernen durch NoCs hängt sowohl von der Anzahl der Router im NoC, als auch von der Architektur des Netzwerks, z.B der Art und Weise in der die Router miteinander verbunden sind, ab.

Die Topologien können beispielsweise vollständig verbundene Netzwerke, Tori, Bäume, Verbindungsnetze, etc. sein (siehe Abbildung 2.2). In letzter Zeit gibt es immer mehr Arbeiten aus dem akademischen Sektor, die sich mit rekonfigurierbaren Architekturen für NoCs beschäftigen. Die Rekonfiguration von NoCs ist auch Gegenstand dieser Arbeit.

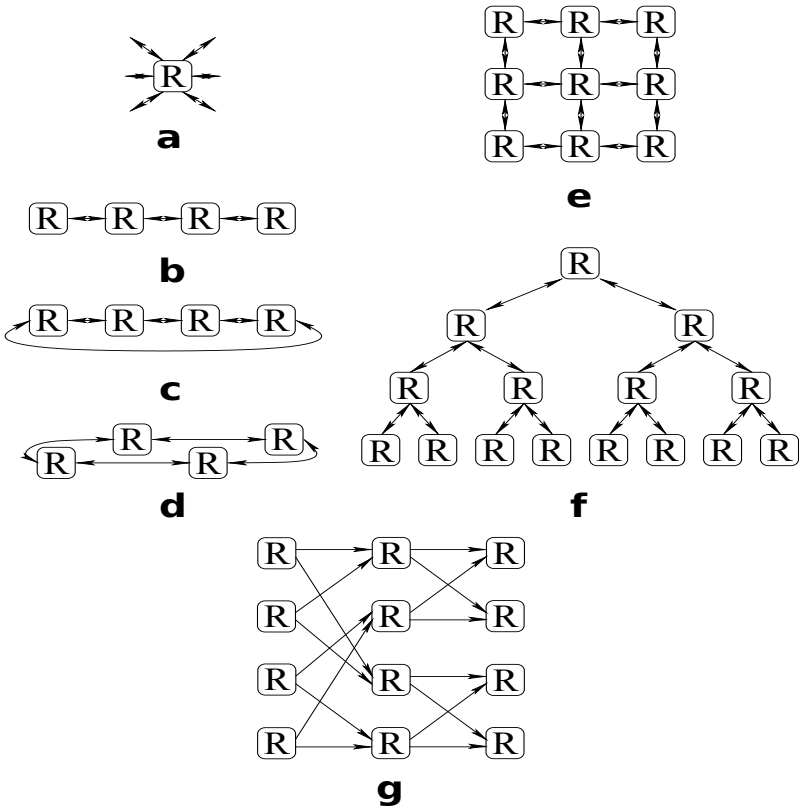

Abbildung 2.2: a) Vollständig verbundenes Netzwerk b) Linear array c) Torus d) Ring e) Mesh f) Binäres Baum g) Mehrstufig

Ein weiterer wichtiger Punkt in der Kommunikation ist die Zuordnung der IP-Adressen der NIs. IPs, die häufig miteinander kommunizieren müssen, sollten physisch nahe beieinander liegen, während IPs, die kaum interagieren, weiter voneinander entfernt sein können. Die Zuordnung von IP-Blöcken zu Netzwerkschnittstellen kann durchaus einen großen Einfluss auf die Effizienz des NoCs haben, aber das übersteigt den Rahmen dieser Arbeit. In den folgenden Abschnitten werden einige wichtige Merkmale und Eigenschaften der NoCs diskutiert.

2.2 Paketbasierter Verkehr

Die Informationen, die die IPs zueinander senden, durchlaufen das NoC in Paketen. Die Paketenlänge kann variieren, aber die meisten NoCs arbeiten mit einer konstanten Paketlänge. Die Nachricht ist im Normalfall folgermaßen aufgebaut:

- Ein Befehl (z.B lesen oder schreiben)
- Adresse (z.B die Adresse, wo die Information gelesen oder geschrieben werden soll)
- Nutzlast, also die eigentlichen Daten

Sobald diese Information das NoC erreicht, schneidet die Netzwerkschnittstelle die Nachricht in Stücke (Flits). Dabei enthält der „erste Schnitt“ (Flit), auch Paket-Header genannt, das Ziel, den Weg, dem es durch das NoC folgen muss, sowie die Art der Daten, die die Nachricht enthält (Abbildung 2.3).

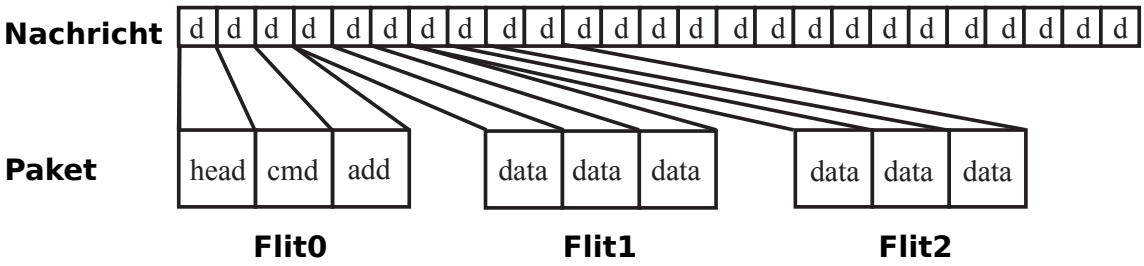

Abbildung 2.3: Message, Paket und Flit

Das Netz hat einen internen Takt, den Netztakt, während dem die Pakete, wenn möglich, von einem Puffer zum Puffer des nächsten Routers vermittelt werden. Dies geschieht wie folgt: Der Kopf eines Pakets, der die Zieladresse enthält, wird ausgewertet, um den Zielausgang des momentanen Schaltelements zu ermitteln. Dieser Ausgang ist dann erreichbar, wenn das Paket im anderen Schaltelementeingang nicht denselben Zielausgang hat oder, falls beide denselben Zielausgang haben, wenn das betrachtete Paket den resultierenden Wettbewerb gewinnt. Der Sieger kann über Prioritäten, durch abwechselndes Bedienen (round robin), zufällig oder durch andere Methoden bestimmt werden. Wenn der Zielausgang erreichbar ist und im nachfolgenden Puffer ein Platz frei ist, dann wird das Paket in den Puffer des nachfolgenden Routers vermittelt. Im anderen Fall wird das Paket blockiert. Es versucht dann in den nächsten Taktzyklen den nachfolgenden Router zu erreichen bis es Erfolg hat.

In jedem Taktzyklus werden alle im Netz vorhandenen Pakete zu dem nächsten auf dem Routing-Path stehenden Router weitergeleitet, sofern sie nicht blockiert sind. Dies wird für jedes Paket solange wiederholt, bis es den gewünschten Netzausgang

erreicht hat. Ist ein Paket ins NoC eingetreten, so kann es nicht gelöscht werden, d.h. ein Packet kann unendlich lange im Puffer gespeichert werden. Allgemein gilt: Ist das Paket ins Netzwerk eingetreten, so kann es nicht gelöscht werden, bis es irgendwann den Ausgang erscheit hat. Diese Strategie wird "backpressure"genannt.

2.3 Vermittlungsart des Pakettransfers

Es gibt drei Arten der Vermittlung [47].: Store-and-Forward, virtuelles Cut-Through und Wormhole Routing

- Bei Store-and-Forward wird ein gesamtes Paket in der Warteschlange eines Routers gespeichert, bevor es zum nächsten Router gesendet wird. Dies erfordert ausreichenden Pufferspeicher für ein gesamtes Paket an jedem Router und das Paket kommt an jedem Router verspätet an, da gewartet werden muss, bis das gesamte Paket eingetroffen ist. Diese Weise der Vermittlungsart erfordert daher große Puffer und hat große Verzögerung zur Folge.

- Beim virtuellen Cut-Through wird ein Paket weitergeleitet, sobald genug Speicherplatz für ein ganzes Paket am nächsten Speicher zur Verfügung steht. Diese Methode nimmt folglich weniger Zeit in Anspruch, erfordert jedoch ebenfalls großen Puffereinsatz.

- Beim Wormhole-Routing werden Pakete in Flits zerteilt. Jedes Flit enthält die Menge der Daten, die über einen Link in einem Zeittakt weitergeleitet werden kann. Bei dieser Art wird ein Flit zum nächsten Router geleitet, sobald dieser genügend Speicherplatz zu Verfügung hat. Sobald der erste Flit eines Pakets über einen bestimmten Ausgangs-Port gesendet wurde, bleibt dieser Port für alle Flits dieses Pakets reserviert. Ein Paket kann daher durchaus über mehrere Router verteilt werden. Da Flits anstelle von ganzen Paketen gespeichert werden, benötigt das Wormhole-Routing den geringsten Puffer-Platz.

2.4 Router

Ein Router ist ein Struktur, die ein Paket von einem Link zum nächsten leitet. Die Pakete kommen bei Links, die mit den Eingangs-Ports der Router verbunden sind, an und gehen über Links, die mit den Ausgangs-Ports der Router verbunden sind. Wenn mehrere Pakete dasselbe Ziel haben, kann nur eines weitergeleitet werden, wobei der Router das Paket auswählt. Pakete, die nicht weitergeleitet werden können, müssen gespeichert werden und werden versuchen, ihr Ziel während des nächsten

Zeittakts zu erreichen. Eine schematische Darstellung eines Routers ist in Abbildung 2.4 aufgeführt.

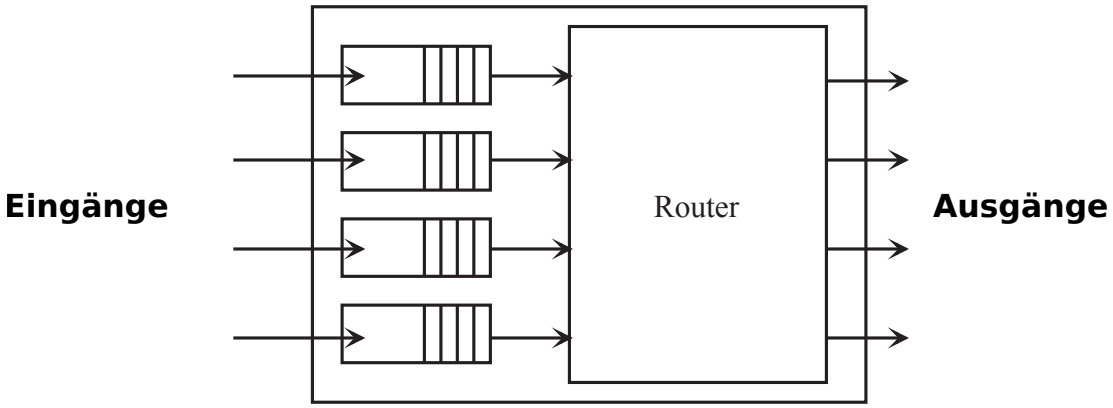

Abbildung 2.4: Router mit 4×4 Ein-/Ausgänge

Wird der erste Flit eines Pakets von einem beliebigen Eingang zu einem beliebigen Ausgang geschaltet, so müssen ihm die übrigen Flits desselben Pakets unverzüglich folgen, was bedeutet, dass es zu keiner Vermischung mit Flits anderer Pakete kommen kann. Der Grund dafür ist, dass das erste Flit den Paket-Header mit den Routing-Informationen enthält. Würde dem ersten Flit des Pakets ein Flit eines anderen Pakets folgen, würden die Routing-Informationen verloren gehen und somit würden die übrigen Flits ihre Zieladresse nicht kennen.

Im Folgenden dieser Arbeit wird davon ausgegangen, dass die Anzahl c der Eingangs- und Ausgangs-Ports der verwendeten Router gleich sind. Ein Router mit c Eingangs-Ports und c Ausgangs-Ports wird $c \times c$ Router genannt. Um die Beschreibung der verschiedenen Router zu vereinfachen, wird zudem davon ausgegangen, dass alle Pakete aus einem Flit bestehen, was bedeutet, dass ein Paket genau einen Zeittakt benötigt um weitergeleitet zu werden.

2.5 Vorgehensweise von Warteschlangen

In diesem Abschnitt werden vier verschiedene Pufferstrategien diskutiert, nämlich kombinierte Eingangs-Ausgangs-Warteschlangen, Ausgangs-Warteschlangen, Eingangs-Warteschlangen und Warteschlangen eines virtuellen Ausgangs.

1. Eingangs-Warteschlangen (Input Queueing = IQ): Bei Eingangs-Warteschlangen gibt es eine FIFO Warteschlange an jedem Eingang des Routers. Während jedem Zeitfenster ist der Head of Line (HOL)-Flit in jeder Warteschlange berechtigt, zu seinem bevorzugten Ausgang geschaltet zu werden. Es muss allerdings eine Übereinstimmung zwischen den Eingängen und

Ausgängen gefunden werden, sodass kein Ausgang zu mehr als einem Eingang zugeordnet wird. Eingangs-Warteschlangen nutzen c FIFO Warteschlangen und die Schalterstruktur ist ein $c \times c$ Router, wie in Abbildung 2.5 a) zu sehen ist. Der Vorteil der IQ ist die günstige Umsetzung in SoC. Der Nachteil wiederum ist, dass der Head of Line-Flit einiger Warteschlangen nicht zum gewünschten Ausgang in einem bestimmten Zeitfenster gelangen kann, da es den Wettstreit mit einem anderen Head of Line-Flit verloren hat. Daraufhin müssen auch die anderen Flits dieser Warteschlange warten, obwohl deren Ausgang in diesem Zeitfenster möglicherweise nicht genutzt wird. Dieses Problem wird Head of Line Blockade (HOLB) genannt. In [22] wird deutlich, dass dies bei großen c und hoher Eingangsrate zu nur 59% Nutzung der Maximalnutzung der Router führt.

2. Ausgangs-Warteschlangen (Output Queueing = OQ): Ausgangs-Router haben eine Beschleunigung s in Vergleich zu IQ; jedes Mal wird ein Zeitfenster in s Phasen unterteilt, wobei s zwischen 1 und c liegt. In jeder Phase werden Pakete von Eingängen zu Ausgängen geschaltet. Dabei gilt die Einschränkung, dass in jeder Phase nur ein Paket von einem Eingangs-Port und nur ein Paket von einem Ausgangs-Port geschaltet werden kann. Das bedeutet, dass jeder Eingangs- und Ausgangs-Port nur einmal pro Phase benutzt werden kann. Bis zu s Pakete werden daher pro Port pro Zeitfenster geschaltet, während immer nur ein Paket pro Zeitfenster über einen Link weitergeleitet werden kann, sodass der Router s-mal so schnell wie der mit ihm verbundene äußere Link arbeiten muss (Abbildung 2.5 c)).

3. Vielfache Warteschlangen (Multiple Queueing = MQ): Beim MQ befinden sich die Warteschlangen an der Eingangsseite des Routers und es gibt c Warteschlangen an jedem Ausgang (eine für jeden Eingang). Daher gibt es c^2 Warteschlangen, eine für jedes Eingangs-/Ausgangs-Paar. Bei der Schalterstruktur handelt es sich nun im einen $c \times c^2$ Router, bei dem $c + c^2$ Leitunden genutzt werden, wie in Abbildung 2.5 b) zu sehen ist. Am Ende eines jeden dieser Drähte befindet sich eine FIFO Warteschlange. Ein Flit, der an Eingang i ankommt und der zu Ausgang j gelangen soll, geht unmittelbar zu Warteschlange $Q(i, j)$. Zwei Flits von unterschiedlichen Eingängen, die zur selben Zeit zum selben Ausgang gelangen sollen, treten verschiedenen Warteschlangen bei. Folglich gibt es keinen Wettstreit im Datenverkehr von Eingangs- zu Ausgangs-Warteschlangen. Ein einfacher Zufalls- oder Ringverteilungsalgorithmus entscheidet dann an jedem Ausgang, von welcher Warteschlange (Wahl aus c Warteschlangen) ein Paket gesendet wird. Leere Warteschlangen werden bei diesem Algorithmus übersprungen. MQ hat den besten Datendurchsatz, da alle Flits direkt zur passenden Warteschlange

an der Ausgangsseite gehen und an jedem Ausgang j wird stets ein Flit weitergeleitet, sofern irgendeine der Warteschlangen $Q(i,j)$ $(i = 1, 2, ..., c)$ nicht entleert ist. Head of Line-Blockaden können bei Ausgangs-Warteschlangen nicht auftreten. Das Problem bei MQ ist der Gebrauch von c^2 Warteschlangen und $c + c^2$ Leitungen. Insbesondere die Komplexität der Verdrahtung würde viel Platz auf dem Chip einnehmen, was teuer ist.

4. Virtuelle Ausgangs-Warteschlangen (Virtual Output Queueing VOQ): Bei virtuellen Ausgangs-Warteschlangen werden alle c Eingangs-Warteschlangen in d getrennte Warteschlangen unterteilt, so dass jede Warteschlange ausschließlich Pakete derselben Herkunft und mit demselben Ziel speichert werden, wie in Abbildung 2.5 d) deutlich wird. In der Praxis ist der tatsächliche Gebrauch von d physisch getrennten Warteschlangen nicht immer notwendig; es ist auch möglich, noch einen Puffer per Eingang zu benutzen. In diesem Fall wird jedoch die Paket-Reihenfolge zerstört. Virtuelle Ausgangs-Warteschlangen verursachen daher hauptsächlich ein Wechsel in der Reihenfolge, in der das Paket eine Warteschlange verlässt; anstelle von FIFO wird eine dynamischere und kompliziertere Reihenfolge angewendet.

In den meisten Applikationen werden Eingangs-Warteschlangen-Router mit virtuellen Ausgangs-Warteschlangen kombiniert. In der Tat bezieht sich der Ausdruck IQ-Router in der Literatur häufig auf einen Router mit Warteschlangen an den Eingängen, egal ob er mit virtuellen Ausgangs-Warteschlangen verbunden ist oder nicht.

In NoCs ist der physische Platz, der für den Router benötigt wird, der dominierende Kostenfaktor des Netzwerks. OQ und Router mit kombinierten Eingang-Ausgangs-Warteschlangen benötigen viele Puffer und sind folglich zu teuer [39]. VOQ kann mit einem einzigen Puffer realisiert werden, macht dann jedoch den Gebrauch eines RAM (Random Access Memory), anstelle einer FIFO Warteschlange, nötig [39]. Für NoC ist der Hauptunterschied zwischen RAM und FIFO der, dass im RAM Pakete aus jeder Position im Speicher entfernt werden können, während in einer FIFO-Warteschlange nur das erste Paket entfernt werden kann. RAMs nehmen im Allgemeinen eine große Fläche ein [48], was sie ebenfalls teuer macht.

IQ-Router haben nur wenige Warteschlangen und es sind günstige FIFO-Warteschlangen [48]. IQ-Router sind daher sehr viel günstiger als die fortgeschritteneren Arten von Routern. (Obwohl einige Netzwerke die fortgeschritteneren Router benutzen, werden IQ-Router in den meisten NoCs in der Literatur [35] trotz ihrer schlechteren Leistung verwendet).

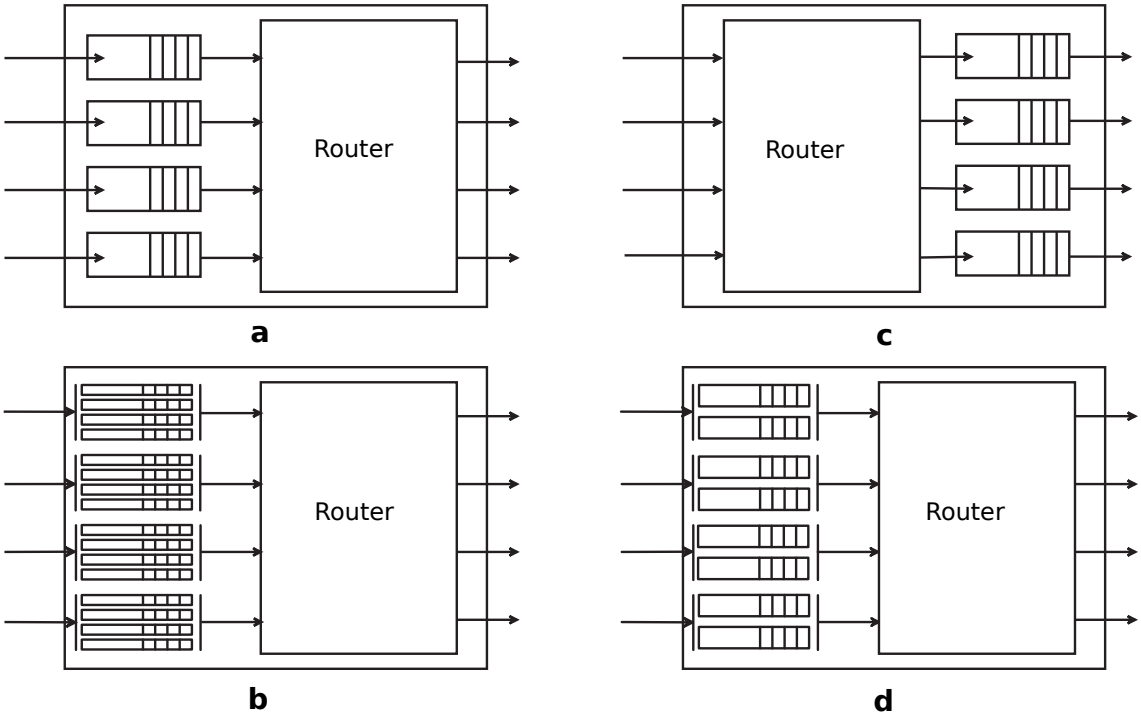

Abbildung 2.5: Pufferstrategien: a) Input Queueing, b) Multiple Queueing, c) Output Queueing d) Virtual Output Queueing

2.6 Struktur und die Funktionsweise des Multistage-Interconnection Network

Im Rahmen dieser Arbeit wird die Rekonfigurationsarchitektur Reconfigurable Multistage-Interconnection Network (RecMIN) auf der Basis von Multistage-Interconnection Network (MIN) entwickelt, so wird in diesem Kapitel die Struktur und die Funktionsweise des MINs näher dargelegt.

Das MIN-Netzwerk besteht aus:

- Router: Dynamisch wechselnde Verbindungen zwischen Schalteingängen und -ausgängen. Eingänge und Ausgänge sind entsprechend des benötigten Ausgangs der Nachricht verbunden. Falls mehrere Eingänge die für denselben Ausgang bestimmten Pakete enthalten, wird eins der Pakete zufällig ausgewählt.
- Puffer: Speicher von Nachrichten. Es ist notwendig, dass jeder Schalteingang des Routers mit einem Puffer verbunden ist. Normalerweise arbeitet MIN mit Eingangs-Warteschlangen (s. Kapitel 2.5).
- Quellen: Erzeuger von Verkehr, der dem Netzwerk angeboten wird. Kann ein

Paket das Netzwerk aufgrund der Überlastung nicht betreten, wird es von der Quelle verworfen. Alle Pakete, die das Netzwerk betreten können, werden ihre Ziele irgendwann dank des Backpressure-Mechanismus (s. Kapitel 2.2) erreichen.

- Ziele: Ausgänge des Netzwerks.
- Leitungen: Verbindung der Quellen mit ihren Router-Puffern.

Die Pakete werden entlang des Routings-Weges von Quellen zu Zielen geleitet (der paketbasierte Verkehr ist in Kapitel 2.2 beschrieben). Es wird angenommen, dass die Größe eines Flits der Größe eines Phit (physical unit) entspricht. Ein Phit repräsentiert die Größe einer Informationseinheit, die ein Netzwerk innerhalb eines Taktzyklus im Netzwerk von einem Puffer zum Folgenden weiterleiten kann. Um die Komplexität des Modells begrenzt zu halten, wird angenommen, dass alle Netzwerkbestandteile vollkommen synchron, von einem globalen internen Takt angetrieben, arbeiten.

Die Struktur eines gepufferten MINs ist in Abbildung 2.6 dargestellt.

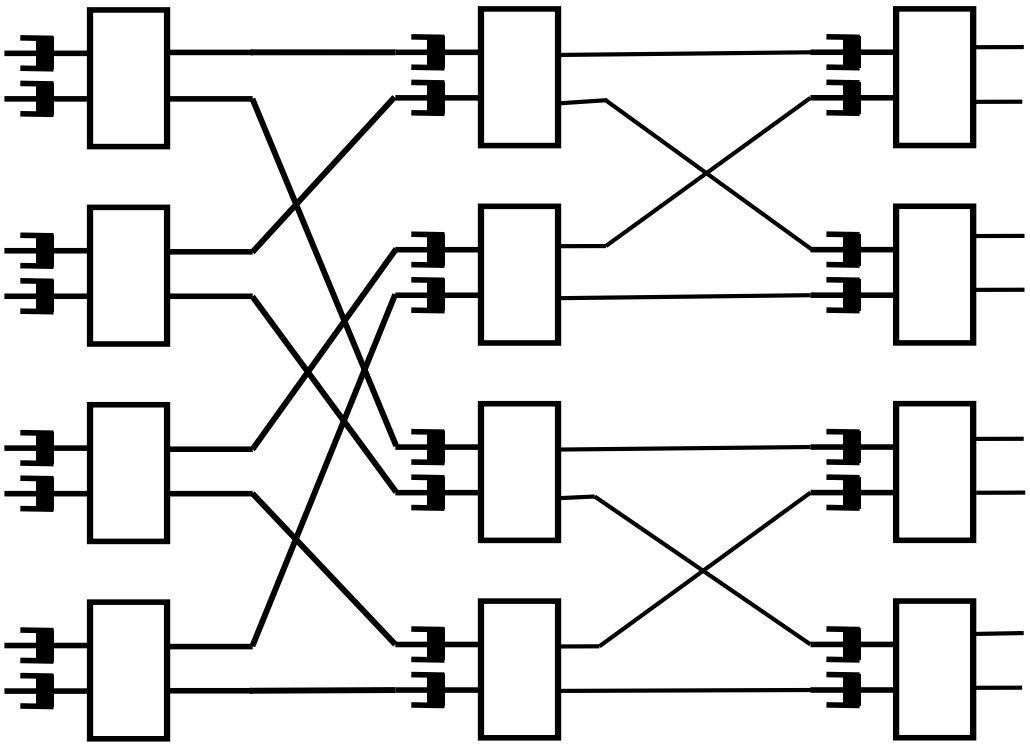

Abbildung 2.6: Multistage-Interconnection Network (MIN)

Das Netz in Abbildung 2.6 besteht aus 8 Eingängen auf der linken Seite und 8 Ausgängen auf der rechten Seite. Dazwischen befinden sich drei Netzstufen, die aus

2×2 Router bestehen. Vor jedem Router-Eingang befindet sich ein Puffer. Da diese Puffer in jeder Netzstufe vorhanden sind, d.h., über das Netz verteilt sind, spricht man von einem intern gepufferten Netz.

Die Schaltelemente der einzelnen Stufen sind über Zwischenleitungen so miteinander verbunden, dass von jedem Netzeingang aus jeder Netzausgang erreicht werden kann. Außerdem besitzen die manche MIN-Netzwerke die s.g. Banyan-Eigenschaft[1]. Solche Netze werden als Banyan-Netze bezeichnet und haben bei N Ein- bzw. Ausgängen und bei $c \times c$ Routerelementen genau $\log_c N$ Stufen. Eine mathematische Berechnungsvorschrift, nach der bestimmt werden kann, mit welchem Eingang $\ell_{i,k+1}$ der Stufe $k+1$ der Ausgang $\ell_{o,k}$ der Stufe k verbunden werden muss, ist in [47] zu finden.

Bei jedem MIN mit Banyan-Eigenschaft kann in jedem der Router lokal entschieden werden, welchen Ausgang eine ankommende Nachricht nehmen muss, damit sie den gewünschten Netzwerkausgang erreicht. Diese lokal in den Routern durchführbare Entscheidung ist ein großer Vorteil des MINs gegenüber anderen Netzwerktopologien. Es ist keine Steuereinheit nötig, die den gesamten Netzwerkzustand kennen muss, um Nachrichten durch das Netz zu leiten. Außerdem kann ein Teil des solchen Netzwerks rekonfiguriert werden, ohne das die restlichen Netzwerk-Komponenten die Umleitung der Pakete vorführen müssen.

2.7 Network on Chip und Rekonfiguration

NoC-Strukturen basieren generell auf einem Universal-Modell, das in der Lage ist, ein breites Spektrum an Applikationen mit vielseitigem Nachrichtenaustausch und Berechnungsprofilen zu unterstützen. Allerdings wird die Leistung von Applikationen mit unregelmäßigen Datenverkehrsprofilen verbessert, wenn spezialisierte Netzwerk-Topologien eingesetzt werden. Heutzutage existieren zwei Möglichkeiten um dieses Problem zu lösen: Rekonfiguration und Umleitung (Re-Routing). Die Idee der Rekonfiguration von NoCs ist weitgehend unerforscht, da die Möglichkeit der Rekonfiguration von Netzwerk-Topologien zusätzliche Hardware-Strukturen benötigt. Die gängige Alternative, die häufig angewendet wird, um das Problem der Netzwerküberlastung aufgrund von Ineffizienz zu lösen, ist die Anwendung eines komplexen Algorithmus, der den Datenfluss im NoC umleitet. Die Komplexität eines solchen Algorithmus wächst normalerweise exponentiell mit der Größe des Netzwerks. Deshalb konzentrieren sich einige Arbeiten der akademischen Gemeinschaft auf die Möglichkeit, NoC-Rekonfiguration als Alternative zu den Umleitungsalgorithmen anzuwenden. Tutsch und Lüdtke [31], [30], [28] und Al Faruque [1] [2] schlagen beispielsweise eine Änderung der Richtungen des Datenfluss vor, um das NoC

[1] Bei Netzen mit der Banyan-Eigenschaft (wie im in Abbildung 2.6 dargestellten Netz) existiert genau ein Weg von jedem Eingang zu jedem Ausgang

für spezielle Verkehrsprofile zu optimieren.

Das Thema dieser Arbeit ist es, eine neue Rekonfigurations-Topologie für NoC zu finden: Dies würde es den NoCs ermöglichen, sich nach den Bedürfnissen des Datenverkehrs zu rekonfigurieren. Auf diese Weise würde die Effizienz des Netzwerks steigen, ohne dass die Regeln des Paket-Routings in dem gesamten NoC, global, geändert werden müssten.

3 Rekonfigurationstechniken für Mehrstufige NoC-Verbindungsnetzwerke

Bei dem Entwurf des NoCs wird meistens nur die mittlere Netzwerkbelastung eingerechnet. Eine mögliche Konsequenz daraus könnte sein, dass manchmal ein Teil des Netzwerks überlastet ist. Das hat zur Folge, dass die Datenpakete mit einer beachtlichen Verzögerung transferiert werden. Währenddessen wird der andere Teil "unterbelastet". In diesem Kapitel stellen wir eine Technik vor, die dazu benutzt werden kann, die Topologie des NoCs zu rekonfigurieren, als eine Reaktion auf die Änderung in der Netzwerkbelastung. Das Ziel der Rekonfiguration ist es, die Datenverzögerung und den Paket-Durchsatz zu optimieren, ohne die zur Verfügung stehende Chipflache zu überschreiten.

3.1 Einleitung

Die sich auf einem Chip befindlichen Netzwerke werden normalerweise als starre Kommunikationsstrukturen angesehen. Es gibt viele Publikationen, die die Netzwerkstruktur [45], [47], [34], [42], [12], [11], die Tiefe der Puffer im Router [45], das Routing von Daten [12], [37] und [29] usw. untersuchen. Nun beschäftigen sich mehr und mehr Autoren mit der Möglichkeit der NoC-Rekonfiguration. Die Rekonfiguration des Netzwerks ermöglicht es, auf die Belastungsänderungen mit der Modifizierung der Netzwerkstruktur zu reagieren und damit die Datenverzögerung auf ein Minimum zu beschränken. Nehmen wir Tutsch [31], [30], [28] oder Al Faruque [1] [2] als Beispiel. In diesen Artikeln schlagen die Autoren vor, die Verbindungen zwischen den Routern [31] oder ihre Ausrichtungen [2] zu verändern. (Dabei bleibt die Anzahl sowohl der Netzwerk-Knoten als auch der Parameter der Router immer gleich).

Eine andere Rekonfiguration-Philosophie bietet ein Algorithmus, der in [7] vorgestellt wird. Abhängig von der Belastung im NoC schlagen die Autoren vor, die Struktur des Netzwerks komplett zu verändern. Unglücklicherweise gibt es mit dieser Lösung für NoC jedoch zwei Probleme:

- der Algorithmus lässt die Tatsache, dass die Chipfläche begrenzt ist, außer Acht. Daher sind in der Realität nicht alle Rekonfigurationsmöglichkeiten zulässig.
- die Rekonfiguration des Netzwerks findet nur global und nicht lokal statt. Nachteil ist dabei, dass die globale Rekonfiguration mehr Zeit beansprucht als die lokale.
- Im Bezug auf NoC wird eine globale Rekonfiguration mit Algorithmus [7] dazu führen, dass einige flow control digits (Flits) nicht zu ihren Ziel-Knoten geliefert werden können. Da im NoC die in das Netzwerk hereingekommenen Pakete nicht gelöscht werden können, blockieren solche Flits die Kommunikation im gesamten Netzwerk.

In diesem Kapitel präsentieren wir für die NoC spezifizierte Rekonfigurationstechnik, mithilfe derer NoC auf die Änderung der Paket-Belastung reagieren kann. Die im Rahmen dieser Arbeit entwickelte Rekonfigurationstechnik verfolgt den Zweck mit einer lokalen Netzwerk-Anpassung die Netzwerk-Ressourcen an den Paketverkehr so anzupassen, dass sich sowohl der Durchsatz, als auch die Paketverzögerung im Netzwerk verbessern. Um eine lokale Änderung zu ermöglichen verwenden wir eine bestimme Struktur, MIN-Struktur (s. Kapitel 2.6) aus der Baseline-Klasse [30], [47], bestehend aus symmetrischen Routern mit jeweils 2^k Ein- und Ausgängen (k= 1,2...) und Input Buffering. Im Folgenden werden wir diese Netzwerkart als 2^k-MIN bezeichnen.

Ein Beispiel eines Netzwerks mit diesen Beschaffenheiten ist in Abbildung 3.1 angegeben. Wie es später gezeigt wird, kann das Netzwerk mit solcher Topologie

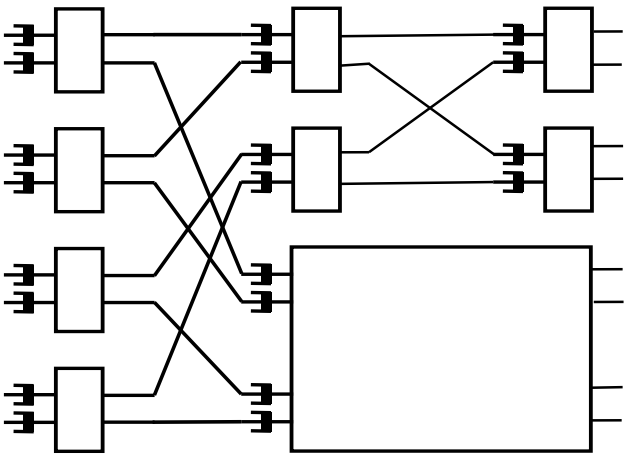

Abbildung 3.1: 2^k-MIN mit 8 Ein- und Ausgängen.

in einige Segmente unterteilt werden, und daher ist es möglich, jedes von diesen Segmenten lokal und getrennt von den anderen zu optimieren.

Um die Effizienz des Netzwerks zu messen, werden wir den Durchsatz des Netzwerks, die Paketverzögerung und die zur Verfügung stehenden Fläche auf dem Chip vor und nach der Rekonfiguration benutzen. Andere Netzwerkeigenschaften wie etwa der Energieverbrauch werden hier bewusst ausgelassen.

3.2 Algorithmus des Netzwerkzerfalls und Synthese

Als Crossbar aufgebaute Router mit c-Eingängen und c-Ausgängen bestehen aus c^2-Kreuzpunkten (Abbildung 3.2) . Ein solcher Router ist zwar skalierbar, aber die Komplexität und insbesondere der Bereich auf dem Chip, der von dem Router in Anspruch genommen wird, steigen quadratisch (entsprechend der Anzahl der Kreuzpunkte). So benötigt, z.B, ein 32×32 Router-Element mit 1024 Kreuzpunkten 4 Mal größere Chipfläche als 16×16 Router mit 256 Kreuzpunkten.

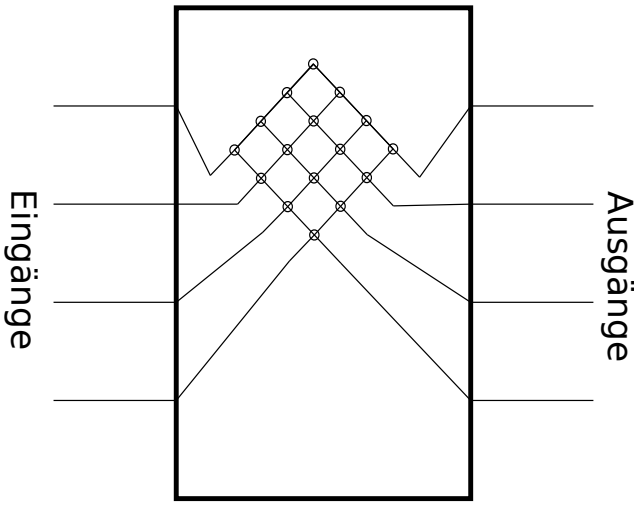

Abbildung 3.2: 4×4 Router mit 16 Kreuzpunkten.

In der Praxis steht normalerweise nicht genug Chipfläche zur Verfügung um komplettes NoC als einen einzelnen großen Router zu realisieren. Um diesen Nachteil zu vermeiden benutzt man MINs, Meshes oder andere Netzwerk-Topologien. Das Verwenden von MINs anstatt eines einzigen Routers bedeutet allerdings, dass der Durchmesser des Netzwerks[1] nicht mehr eins gleicht. Als Folge zeigt ein MIN höhere Datenverzögerungszeiten und einen niedrigen Datendurchsatz als einen einzigen Router. So muss, auf einer Seite, auf den Flächenverbrauch bezogen, die optimale

[1] Durchmesser des Netzwerks ist die maximale direkte Entfernung zwischen zwei Knoten.

MIN-Topologie ausschließlich aus 2×2 Routern bestehen, auf der anderen Seite, um den besten Durchsatz zu erreichen, soll man das MIN durch einen einzelnen Router ersetzen.

Aus der oben aufgestellten Überlegung folgt: Für NoC wäre es optimal, ein MIN zu realisieren, das aus Router-Elementen von verschiedener Größe gebaut ist: Große Router in den Teilen des Netzwerks, die hochbelastet sind - um Datendurchsatz zu maximieren; kleine Router - in unter niedriger Belastung stehenden Teilen des Netzwerks, um den Chipflächen-Verbrauch auszugleichen. Da die Belastung der NoC dynamisch ist, kann es unter Umständen passieren, dass die überlasteten Netzwerkbereiche zu "unterbelasteten" werden und umgekehrt, also muss sich auch die Netzwerk-Topologie entsprechend der neuen Last anpassen können.

Um eine solche Anpassung zu ermöglichen werden einige Rekonfigurationstechniken vorgestellt[2]. Die Rekonfigurationstechniken basieren auf zwei Router-Umwandlungen (Zerfällen; Englisch: decays), mithilfe deren einen Router (single stage MIN) in ein zweistufiges 2^k-MIN rekonfiguriert wird. Eine umgekehrte Rekonfiguration, von zweistufigen 2^k-MIN zu einem Router, wird entsprechend Synthese genannt. Des Weiteren wird gezeigt, dass es möglich ist, angefangen von einem $N \times N$ Router durch die bestimmte Anwendung von Zerfall/Synthese jedes mögliche $N \times N$ 2^k-MIN zu erreichen.

Es werden zwei Zerfälle definiert:

1. **Alpha+ Zerfall**
2. **Alpha- Zerfall**

Der **Alpha+ Zerfall** transformiert das Router-Element, bestehend aus $c \times c$ Kreuzungspunkten, in ein zweistufiges MIN, wobei die erste Stufe aus $c/2$ Routern besteht, wobei jeder Router nur 2×2 groß ist. Die zweite MIN-Stufe besteht aus zwei gleichen $c/2 \times c/2$ Routern (s. Abbildung 3.3b). Der **Alpha- Zerfall** gleicht dem **Alpha+ Zerfall**, aber in diesem Fall besteht die erste Stufe aus zwei $c/2 \times c/2$ Routern und die zweite Stufe besteht aus $c/2$- Routern, jeweils die 2×2 groß (s. Abbildung 3.3c).

Die Anzahl der Kreuzungspunkte sowohl nach **Alpha+ Zerfall** als auch nach **Alpha- Zerfall** wird immer reduziert (Ausnahme ist $c = 4$, wo sie unverändert bleibt:

$$c^2 \geq 2 \cdot \left(\frac{c}{2}\right)^2 + \frac{c}{2} \cdot 2^2 \quad \text{für } c \geq 4 \tag{3.1}$$

$$c^2 \geq \frac{c^2}{2} + 2 \cdot c \quad \text{für } c \geq 4 \tag{3.2}$$

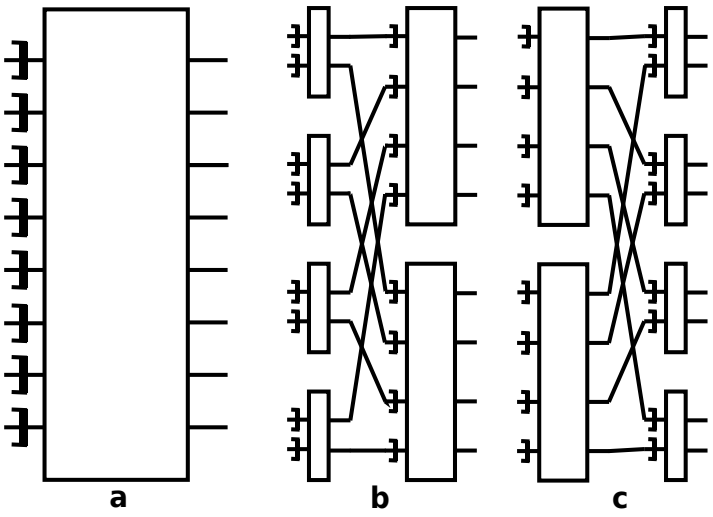

Abbildung 3.3: **Alpha+ Zerfall** und **Alpha- Zerfall** für $c = 8$.

Daher benötigt das daraus resultierende MIN $\frac{2 \cdot c}{c+4}$ mal weniger Kreuzpunkte, was entsprechend zu $\frac{2 \cdot c}{c+4}$ Chipflächen-Ersparnis führt.

Die minimale Verzögerung der Daten, die durch das **Alpha+** / **Alpha-**-Zerfall resultierende Netzwerk passieren müssen, vergrößert sich zu zwei Takten wegen der zwei Stufen, aus denen das Netzwerk besteht. Allerdings benötigt das Netzwerk weniger Chipfläche als ein $c \times c$- Router.

Ein anderer wichtiger Aspekt des **Alpha+** / **Alpha-**-Zerfalls ist, dass der Paketverkehr im Netzwerk (an der Stelle, die aus zwei Router-Elementen besteht) in zwei fast voneinander unabhängige Teile gespalten wird.

Demnach, falls der Paketverkehr in einem von zwei Routern niedrig ist, ist es sinnvoll das Router-Element durch ein Bus-Element zu ersetzten. Die dabei freigesetzten Ressourcen können z.B für die zusätzlichen Puffer-Speicher in anderen NoC-Bereichen benutzt werden. Es ist ebenfalls von Vorteil, die freigesetzten Hardware-Ressourcen dazu zu verwenden, eine **Alpha+** / **Alpha-**-Synthese in einem überlasteten NoC-Bereich durchzuführen (s. nächstes Unterkapitel).

Wie oben erwähnt, wird der umgekehrte Prozess des Zerfalls als Synthese definiert. So kann das Netzwerk mit $N = 16$, bestehend aus 2×2 Routern, mit einer **Alpha+**-Synthese zu einem MIN überführt werden, das in Abbildung 3.4 zu sehen ist. Entsprechend zeigt die Abbildung 3.5 ein MIN, das als Ergebnis aus einer **Alpha-**–Synthese resultiert.

[2]Diese Techniken werden weiter als Grundlage für die rekonfigurierbare Architektur RecMIN benutzt.

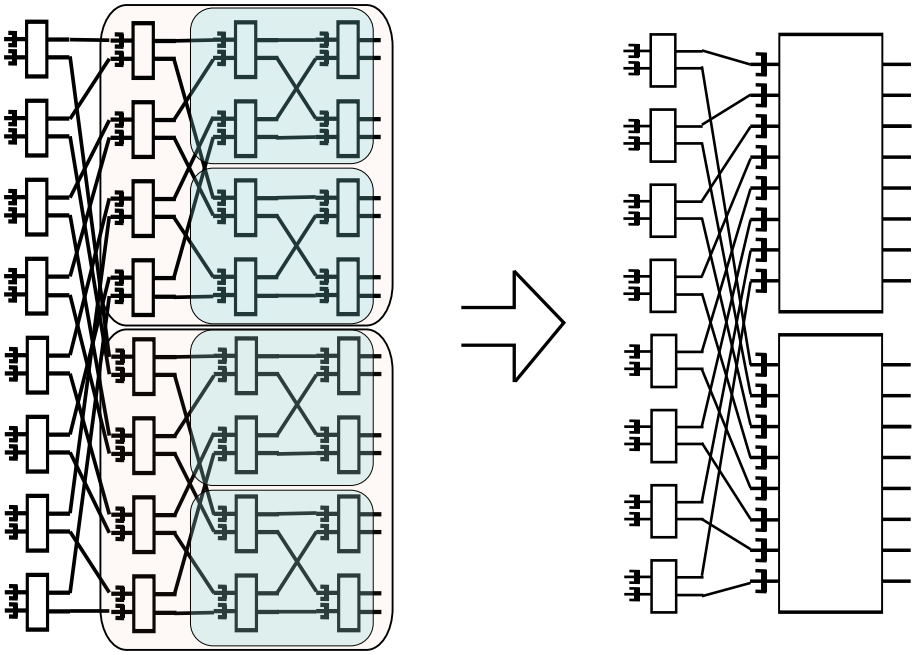

Abbildung 3.4: **Alpha**+-Synthese für $N = 16$

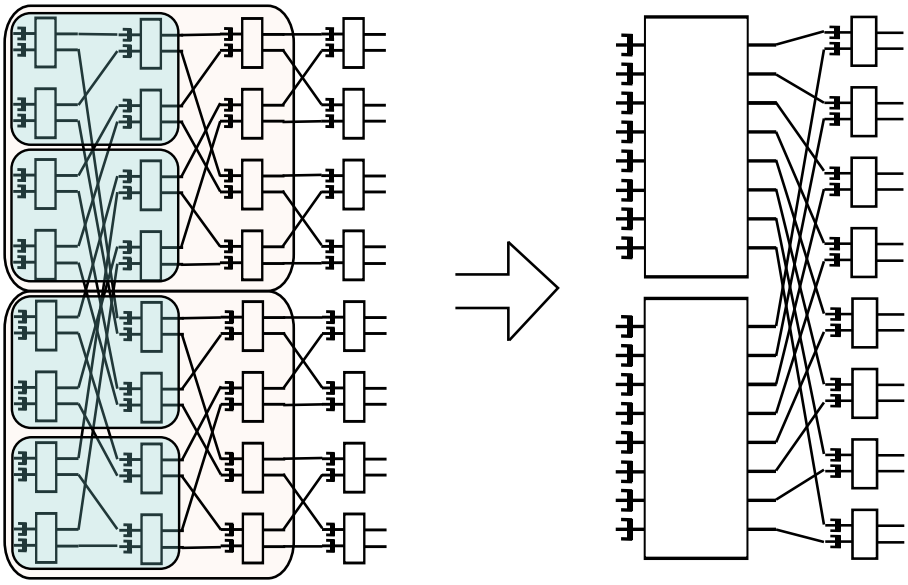

Abbildung 3.5: **Alpha-**-Synthese für $N = 16$

Genau wie der Zerfall, kann die Synthese nicht nur im Ganzen im NoC, sondern auch in einzelnen Segmenten angewendet werden. Ein durch Synthese entstandenes

Netzwerk benötigt normalerweise mehr Chipflache als das ursprüngliche Netzwerk vor der Synthese. Aber wegen der Reduktion der MIN-Stufen wird die minimale Paketverzögerung dementsprechend auch niedriger.

3.3 Diskussion des Alpha+ und Alpha- Zerfalls

Eine der wichtigsten Fragen in Bezug auf **Alpha**+ und **Alpha-**-Zerfall/Synthese ist, ob es möglich ist jedes beliebige $N \times N$ 2^k-MIN als Folge von **Alpha**+ und **Alpha-**-Zerfall/Synthese (angefangen mit einem $N \times N$ Router) darzustellen. Wenn ja, ist es möglich nur mit einer der beiden Zerfalls/Synthese-Techniken (z.B ausschließlich mit **Alpha**+) ein beliebiges $N \times N$ 2^k-MIN zu erreichen? Wenn nein, welche Untermenge der $N \times N$ 2^k-MINs können auf solche Art und Weise dargestellt werden?

Lemma: Es ist möglich, angefangen mit einem einzigen $N \times N$ Router, ein beliebiges 2^k-MIN mit N Ein- und Ausgängen als eine Folge von nur **Alpha**+ oder nur **Alpha-** Zerfall/Synthese zu errechnen.

Beweis: Ein beliebiger $c \times c$ Router kann nach ld(c)-1 **Alpha**+/**Alpha-**-Zerfällen zu einem MIN, bestehend ausschließlich aus 2×2 Router-Elementen, transformiert werden. Ebenso ist es möglich ein beliebiges $N \times N$ 2^k-MIN in ein MIN, bestehend ausschließlich aus 2×2 Router-Elementen zu transformieren. Dafür muss man **Alpha**+/**Alpha-**-Zerfälle (es macht keinen Unterschied, ob mit **Alpha**+ oder **Alpha-**-Zerfall gearbeitet wird) iterativ auf jeden $c \times c$ Router im vorgegebenen Netzwerk anwenden, das mehr als 2 Ein- und Ausgänge hat. Infolge dessen muss der umgekehrte Weg auch möglich sein, durch die Reihenfolge der Synthese-Schritte von einem MIN aus 2×2 Router zu einen beliebigen $N \times N$ 2^k-MIN zu gelangen. Da man, wie schon oben erwähnt, von einem beliebigen $N \times N$ Router nach ld(N)-1-Zerfälle zu einem MIN aus 2×2 Router gelangen kann, so ist es möglich, angefangen bei einem einzigen $N \times N$ Router, ein beliebiges 2^k-MIN mit N Ein- und Ausgängen als eine Folge von nur **Alpha**+ oder nur **Alpha-** Zerfall/Synthese zu erreichen□.

Ein weiterer Punkt, der im Bezug auf **Alpha**+/**Alpha-**-Zerfälle diskutiert werden muss: Welches von diesen zwei Zerfällen zu einem effizienteren Netzwerk (mit niedrigster Paketverzögerung sowie höchst möglichem Paket-Durchsatz) führt. Noch ein wichtiger Untersuchungspunkt: Wie groß sollen die Eingangspuffer der Router in einem aus dem Zerfall resultierenden MIN sein, d. h., sollen die Puffer in der ersten und zweiten MIN-Stufe gleichgroß sein oder sollen die Pufferlängen im bestimmten Verhältnis zueinander stehen?

Anhand der Simulation[3] wurde eine Analyse der **Alpha**+ und **Alpha-** Netzwerke gemacht. Alle Simulationen sind unter folgenden Bedingungen gemacht worden:

[3] Alle Simulationen werden mithilfe des im Rahmen dieser Arbeit entwickelten Simulation-Tools *RecSim* gemacht; die genaue Beschreibung von *RecSim* befindet sich in Kapitel 5.

- die Anzahl der Flits in jedem Paket ist konstant - jedes Paket besteht aus einem Flit
- der Paketverkehr im NoC ist Unicast und gleichverteilt
- die Kollisionen in den Routern werden mit dem Random-Verfahren gelöst
- die Pakete werden ohne Priorität verschickt

Die Summe der Speicherplätze der Eingangspuffer vor und nach dem Zerfall muss gleich bleiben. d. h., $m_0 = m_1 + m_2$, wobei m_0 - die Tiefe des Puffers pro Eingang im Router ist, der rekonfiguriert werden soll, m_1 - die Tiefe des Puffers pro Eingang in der ersten MIN-Stufe und m_2 - die Tiefe des Puffers pro Eingang in der zweiten MIN-Stufe sind. Es wird also nach dem Verhältnis zwischen m_1 und m_2 gesucht, bei dem das nach dem Zerfall entstandene MIN die beste Effizienz zeigt. Dafür sind 8×8, 16×16 und 32×32 Router mit Puffertiefe $m_0 = 2, 4, 6, 8, 16$ untersucht worden. Alle möglichen Verhältnisse zwischen m_1 und m_2 wurden sowohl für **Alpha+** als auch für **Alpha-** Zerfall untersucht; bei den Simulationen entspricht das Konfidenzintervall 95% und der relative Fehler ist auf 0,5% gesetzt worden.

3.4 Ergebnisse der Rekonfiguration

Die Abbildung 3.6 zeigt die Simulationsergebnisse für zwei MINs, die aus einem 8×8 Routern mit Puffergröße von $m_0 = 6$ pro Eingang entstanden sind. Die in Abbildung 3.7 dargestellten Ergebnisse ähneln den Ergebnissen für alle untersuchten Router-Elemente mit unterschiedlichen Puffergrößen. Um die nach dem Zerfall entstandenen MINs besser miteinander vergleichen zu können, sind die Simulationsergebnisse in drei Gruppen unterteilt: **Alpha+**-Zerfall (Abbildungen 3.6a, 3.6b, 3.7a, 3.7b), **Alpha-** Zerfall mit $m_1 > m_2$ (Abbildungen 3.6c, 3.6d, 3.7c, 3.7d) und **Alpha-** Zerfall mit $m_1 \leq m_2$ (Abbildungen 3.6e, 3.6f, 3.7e, 3.7f). Anschließend werden die besten Verteilungen aus diesen drei Gruppen getrennt vorgestellt um sie besser vergleichen zu können (Abbildungen 3.6g, 3.6h, 3.7g, 3.7h).

Die Abbildungen 3.6a, 3.6b zeigen den Paket-Durchsatz in Abhängigkeit von der Netzwerkbelastung und den Paket-Durchsatz für die MINs, die aus einem 8×8 Routern mit $m_0 = 6$ mithilfe des **Alpha+** Zerfalls entstanden sind. Die Kurven für die verschiedenen Puffer-Verteilungen zwischen zwei MIN-Stufen sind präsentiert worden.

Das gleiche gilt für die Abbildungen 3.7a, 3.7b mit dem Unterschied, dass hier der 32×32 Router mit Puffertiefe $m_0 = 16$ gespalten wird.

So kann man feststellen, dass sich in der ersten Gruppe, bestehend aus allen möglichen Puffer-Verteilungen (ausgenommenen $m_1 = 1$), die Verteilung $m_1 = 2$

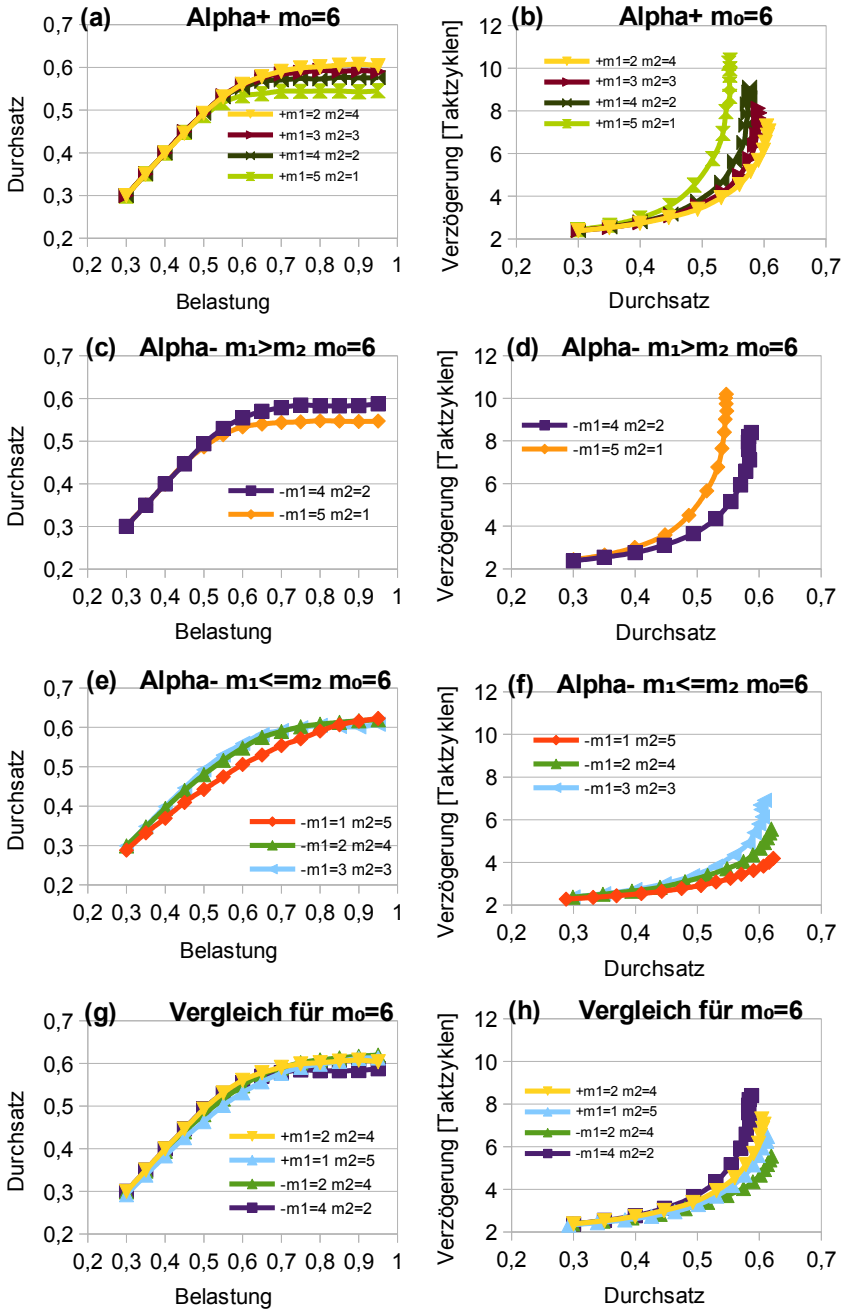

Abbildung 3.6: Rekonfiguration des 8×8 Router-Elementes mit einer Puffertiefe $m_0 = 6$ zu einem MIN.

und $m_2 = m_0 - 2 = 4$ in Abbildungen 3.6a 3.6b (entsprechend die Verteilung $m_1 = 2$ und $m_2 = m_0 - 2 = 14$ in Abbildungen 3.7a 3.7b) jeweils den höchsten Durchsatz

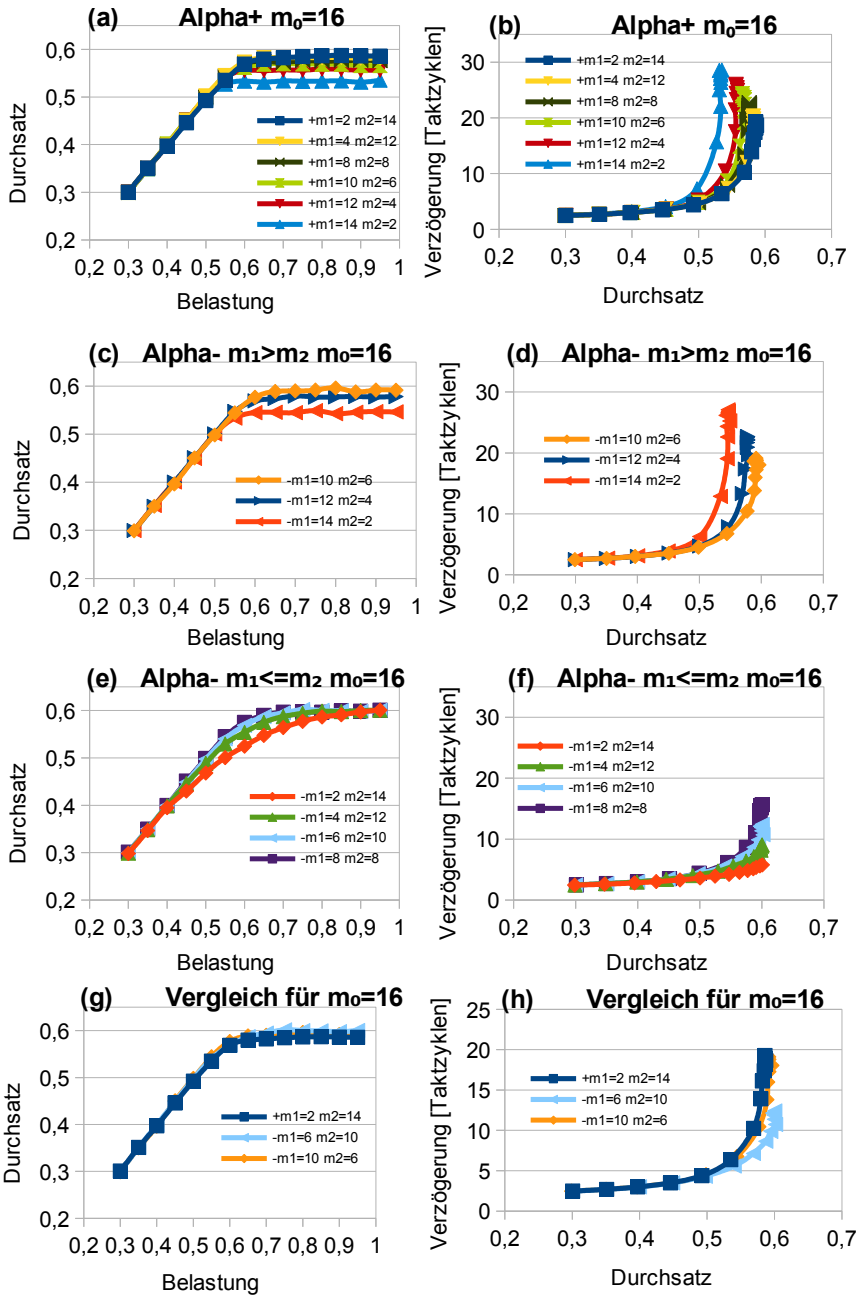

Abbildung 3.7: Rekonfiguration des 32×32 Router-Elementes mit einer Puffertiefe $m_0 = 16$ zu einem MIN.

und die niedrigste Verzögerung im Vergleich zu allen anderen Puffer-Verteilungen innerhalb dieser Gruppe zeigt. Es kann ebenfalls gesehen werden, dass je größer der

Puffer m_1 und je kleiner der Puffer m_2 ist, desto schlechter ist der Paket-Durchsatz und desto größer die Verzögerung.

Die zweite Gruppe, die auf Abbildungen 3.6c, 3.6d, 3.7c, 3.7d dargestellt wird, zeigt das Verhalten der Netzwerke, die aus dem **Alpha-** Zerfall des gleichen Routers resultieren, wobei die Puffer-Verteilung zwischen m_1 und m_2 die Bedingung $m_1 > m_2$ erfühlen muss. Zum Beispiel besteht diese Gruppe für den 8×8 Router mit $m_0 = 6$ aus zwei mithilfe des **Alpha-** Zerfalls entstandenen Netzwerken: Mit $m_1 = 4$, $m_2 = 2$ und $m_1 = 5$, $m_2 = 1$. Diese Gruppe zeigt das gleiche Verhalten wie das der ersten Gruppe: Ein Netzwerk mit der größten m_1 Puffer-Länge (für $m_1 = 4$ und $m_2 = 2$ s. Abbildungen 3.6c, 3.6d, und für $m_1 = 10$ und $m_2 = 6$ s. Abbildungen 3.7c, 3.7d) zeigt die beste Effizienz, sowohl für den Durchsatz als auch für die Verzögerung.

Die dritte Gruppe auf Abbildungen 3.6e, 3.6f, 3.7e, 3.7f der Netzwerke dar, die aus dem **Alpha-**-Zerfall erstanden sind, wobei die Puffer-Größe $m_1 \leq m_2$ ist. Also zu dieser dritten Gruppe gehören drei Verteilungen: $m_1 = 1$ und $m_2 = 5$, $m_1 = 2$ und $m_2 = 4$, und $m_1 = 3$ und $m_2 = 3$. Die Netzwerk-Eigenschaften in dieser Gruppe unterscheiden sich von den Netzwerk-Eigenschaften in den ersten beiden Gruppen. Hier zeigt keine der betrachteten Verteilungen eindeutig die besten Ergebnisse sowohl für den Durchsatz als auch für die Verzögerung. Allerdings kann eine empirische Regel für alle Netzwerke aus dieser Gruppe formuliert werden: Je höher der maximale Durchsatz im MIN, desto niedriger die Paketverzögerung. Aus diesem Grund kann kein eindeutiger "Sieger" in der dritten Gruppe bestimmt werden. Stattdessen muss ein Netzwerk gefunden werden, das einen guten Kompromiss zwischen Durchsatz und Verzögerung anbietet. Zum Beispiel kann bei den Abbildungen 3.6e, 3.6f das Netzwerk mit $m_1 = 2$ und $m_2 = 4$, und die Abbildungen 3.7e, 3.7f $m_1 = 6$ und $m_2 = 10$ als ein solcher Kompromiss gesehen werden.

In den letzten zwei Grafiken (Abbildungen 3.6g, 3.6h und Abbildungen 3.7g, 3.7h) werden die effizientesten MINs aus drei Gruppen miteinander verglichen. (Zusätzlich wird auch das MIN, das als Ergebnis von **Alpha+**-Zerfall mit $m_1 = 1$, and $m_2 = m_0 - 1$ Puffer-Verteilung entsteht, in den Vergleich einbezogen). Es kann gesehen werden, dass das Netzwerk aus der dritten Gruppe die niedrigste Paketverzögerung zeigt: **Alpha-**-Zerfall mit $m_1 \leq m_2$. Der Durchsatz in diesem Netzwerk ist auch fast der höchste von allen.

So zeigt das MIN, das aus dem **Alpha-** resultiert und die Puffer-Verteilung $m_1 = 2$ und $m_2 = 4$ hat, das effizienteste Verhalten nach der Rekonfiguration des 8×8 Router-Elementes mit einer Puffertiefe $m_0 = 6$ (für den 32×32 Router).

Die Simulationsergebnisse nach der Spaltung der anderen Routers sehen ähnlich aus. Insofern zeigten die Simulationen, dass die effizientesten 2^k-MIN als Ergebnis des **Alpha-**-Zerfalls mit der Puffer-Verteilung von $1 : 2$ (zwischen m_1 und m_2) entstehen.

Die mögliche Erklärung zu gewonnenen Ergebnissen lautet wie folgt: Die zweite Stufe des MINs kann als ein Hauptnetzwerk betrachtet werden, und die erste Stufe

des MINs kann als "Filter" vor dem Hauptnetzwerk gesehen werden. Um zum Hauptnetzwerk zu gelangen, soll also der Paketverkehr zuerst durch den Filter durchgehen.

Ist der Paket-Durchsatz des Filters viel größer als der Durchsatz des Hauptnetzwerkes, so wird das Hauptnetzwerk mit dichtem Paketverkehr belastet. Das Hauptnetzwerk wird überlastet und blockiert den Paketverkehr, was sowohl zu Reduzierung des Paket-Durchsatzes als auch zum Wachstum der Paketverzögerung an den MIN-Ausgängen führt. Ist aber der Durchsatz des Filters kleiner als der Durchsatz des Hauptnetzwerkes, so werden nur wenige Pakete zum Hauptnetzwerk geleitet (was zu Reduzierung des Durchsatzes im MIN führt). Allerdings entstehen wenige Flaschenhälse im Hauptnetzwerk, so werden die Pakete schneller zu ihren Zielen geleitet (Paketverzögerung reduziert sich).

Aus der oben beschriebenen Überlegung folgt: Es lohnt sich also in erster Linie die Effizienz der zweiten MIN-Stufe zu erhöhen, da es sowohl den Durchsatz als auch die Verzögerung im gesamten MIN verbessert. Die Erhöhung der Effizienz der ersten Stufe erhöht meistens nur den Paket-Durchsatz, aber nicht die Paketverzögerung.

Aus der Untersuchungen von Boot [36] folgt, dass von allen symmetrischen $c \times c$ Router der Router mit zwei Ein- /Ausgängen den besten Durchsatz wie die Verzögerung zeigt. Deshalb ist es vorteilhaft die zweite MIN-Stufe aus 2×2 Router zu realisieren (also **Alpha--**Zerfall zu benutzen). Die Untersuchungen von Tutsch [47], zeigen, dass je tiefer der Eingangspuffer der Router, desto höher ist der Paket-Durchsatz in MIN. Aufgrund dieser Behauptung kann man die Puffertiefe als einen zusätzlichen Parameter sehen, mithilfe dessen die Effizienz des MINs beeinflusst werden kann. Auch hier ist vorteilhaft zunächst die Effizienz der zweiten MIN-Stufe zu erhöhen. Allerdings benutzt man alle vorhandenen Puffer-Ressourcen nur für die zweite MIN-Stufe, so muss man mit der starken Reduzierung des Paket-Durchsatzes rechnen. Um es zu vermeiden, müssen die für gesamten MIN vorhandenen Puffer-Ressourcen im Verhältnis 1:2 zwischen erster und zweiter Stufe verteilt werden.

Dieser Überlegung entsprechend, muss das effizienteste 2^k-MIN als Ergebnis des **Alpha--**Zerfalls mit der Puffer-Verteilung 1:2 entstehen, was auch durch die Simulation bestätigt wird.

4 Netzwerkberechnung

Die Möglichkeit zwischen zwei verschiedenen Topologien umschalten zu können basiert auf mehreren Informationen. Zum einen, welche Eigenschaften das resultierende Netzwerk im Vergleich zum ursprünglichen Netzwerk haben wird. Zum anderen, wie sich die Paketverzögerung und der maximale Durchsatz ändern. Diese Kenntnisse über die Netzwerkeigenschaften können entweder analytisch (z.B mithilfe der Markov-Kette, Network Calculus oder Petri-Netzen) oder nichtanalytisch (Computersimulationen) gewonnen werden.

4.1 Stand der Forschung

Die Arbeiten, die mit den klassischen Warteschlangen-Modellen das Netzwerk zu beschreiben versuchen sind vor allem die Arbeiten von Boot [36] und Beekhuizen [6]. Boot stellt unter anderem ein Modell für ein symmetrisch belasteten $c \times c$ Router mit Eingangs-Warteschlangen dar. Die Tiefe der Puffer wird dabei als unendlich angenommen. Das Modell besitzt die Markov-Eigenschaft (Gedächtnislosigkeit) [24] und kann so durch die Markov-Kette beschrieben werden. Das von Boot vorgeschlagene Modell ist eine Weiterentwicklung des Modells von Karol, Hluchyj and Morgan [22], wobei anstatt die Parameter der geometrischen Verteilung für die Servicezeit-Funktion anzupassen, wird die Approximation durch ein Polynom zweiten Grades benutzt.

Beekhuizen entwickelt dieses Konzept weiter [6]. Er schlägt sowohl ein analytisches Modell für einen unsymmetrisch belastete Router als auch das Approximationsmodell für die Baum-Topologie mit dem Backpressure-Mechanismus (s. Kapitel 2.2) vor. Allerdings ist das vorgeschlagene Modell nicht für alle Netzwerktopologien anwendbar.

Network Calculus (Deutsch: Netzwerk Kalkül) ist eine vergleichsweise neue Systemtheorie für deterministische Warteschlangen. Die Basis-Arbeiten stammen von Cruz [8] [9] [10] und sind in der 1990-er Jahren vorgestellt. Die Idee des Network Calculus basiert darauf, dass die deterministischen Dienstgütegarantien durch Verkehrsregulierung, Scheduling und Zugangskontrolle gegeben werden können. Die Anwendungsgebiete des Network Calculus sind die drahtlose Sensor-Netzwerke [47], Internet-Servicequalität und geschaltete Ethernets. Es gibt auch Arbeiten, in denen versucht wird, diese Methode auch für die NoC-Berechnung zu verwenden, unter

anderem [21], [18], [17]. Bakhouya, Suboh, Gaber, El-Ghazawi, Niar schlagen eine Methode vor, die mit Network Calculus die NoC 2D-Mesh-, WK-recursive- und Spidergon-Topologie beschreibt [5]. Leider zeigen die Ergebnisse eine deutliche Abweichung vom Simulationsergebnis - bis zu 5% bei Paket-Durchsatz und bis zu 28% bei Paketverzögerung. Der vor allem große relative Fehler bei der Paketverzögerung macht das Benutzen der heute vorhandenen Modelle des Network Calculus für NoCs nur als grobe Parameterabschätzung sinnvoll.

Jenq [19] zeigt eine weitere Methode, Paketverzögerung und den Durchsatz für eine symmetrische synchrone MIN-Topologie, bestehend aus 2×2 Router mit einer symmetrischen Belastung, durch das Iterationsverfahren zu lösen. Dabei nutzt sie die Symmetrie aus, um die Netzstufen des MINs zu reduzieren und betrachtet nur noch einen einzelnen Ein- bzw. Ausgang eines 2×2 Schaltelementes. Bei diesem Verfahren sind die Puffer des Netzwerks genau 1 Flit groß. Diese Methode wurde durch Yoon [50] auf die Puffer mit beliebiger Länge erweitert. Ferner erweiterten und verbesserten Arbeiten von Shaikh [41] und Theimer [44] die Methode von Jenq durch Ergänzung seines Konzepts auf $c \times c$ Router-Elemente, mit Puffer beliebiger Größe. Die in diesen Publikationen vorgeschlagenen Verbesserungen der Methode führen dazu, dass das Iterationsmodell allgemeiner und akkurater wird: Die relative Abweichung von dem durch die Simulation gewonnenen Durchsatz liegt unter 5%. Bezüglich der Paketverzögerung, die wesentlich empfindlicher gegenüber Vernachlässigungen von Abhängigkeiten als der Durchsatz ist, liegt die relative Abweichung bei 10% bis 30%. Außer großen Abweichungen für die Paketverzögerung ist eine weitere Schwachstelle der Iterationsmethode deren begrenzte Anwendbarkeit: Ist das Netzwerk oder dessen Belastung unsymmetrisch, so lässt sich die Iterationsmethode nicht anwenden.

Tutsch [46] schlägt vor, die MIN-Topologie anhand des von ihm vorgestellten Petri-Netz-Modells zu approximieren. Dabei werden unter anderem zusätzliche Erweiterungen des Modells, wie die Möglichkeit der Mehrfachsendung, vorgenommen. Zwar zeigt das Petri-Netz-Modell gegenüber dem Iterationsmodell [46] eine erhebliche Verbesserung, dennoch wird die Berechnungsdauer für ein solches Modell vom Autor als "sehr lang "eingeschätzt. So dauert laut seinen Angaben die Berechnung eines solchen Modells auf einem 1,2 GHz One-Core-Processor Computer für 64×64 MIN mehr als zwei Wochen.

Für das Berechnen unsymmetrischer MIN-Topologien mit Back-Pressure-Übertragung existiert nach bestem Wissen des Autors momentan kein analytisches Modell. Alle Forschungsergebnisse zu diesem Thema werden ausschließlich mit Simulationen gewonnen. Jede Forschungsgruppe schreibt entweder seinen eigenen Simulator oder nutzt eine von dritter Person entwickelte Software. Zu den bekanntesten Software-Simulatoren für die Netzwerke gehören von allem Booksim [20] und OPNET [13]. Für die Real Time NoCs wird beispielsweise RTNoC [32] verwendet. Li, Ling, und Hu [25] bekommen ihre Ergebnisse für den Energieverbrauch in NoC

durch ein in SystemC geschriebenes Simulations-Tool MSNS (MPI-Style NoC Simulator). Allgemein lässt sich sagen: Während man vor 10-15 Jahren unbedingt ein analytisches oder vereinfachte Modell des Systems haben wollte, geben sich heutzutage die meisten Forschungsgruppen mit der vollständigen Simulation des Systems zufrieden. Dies liegt unter anderem daran, dass in der Vergangenheit wenige Rechnerressourcen für die Simulation zur Verfügung standen. Dieses Problem besteht heutzutage nicht mehr.

4.2 Das analytische Modell für einen einzelnen Router mit unendlichen Eingangs-Warteschlangen und begrenzer Pufferlänge

Das analytische Modell für einen einzelnen Router mit unendlicher Pufferlänge (Eingangs-Warteschlangen) wurde, wie oben erwähnt, in Arbeiten von Boot [36] dargestellt. Es wurde gezeigt, dass ein solcher Router als ein M/D/1 System dargestellt werden kann. Als eine Weiterentwicklung kann jetzt ein Modell für ein ähnliches Router-Element mit endlicher Pufferlänge entwickelt werden. Die Markov-Kette für ein solches Element ist in der Abbildung 4.1 vorgestellt.

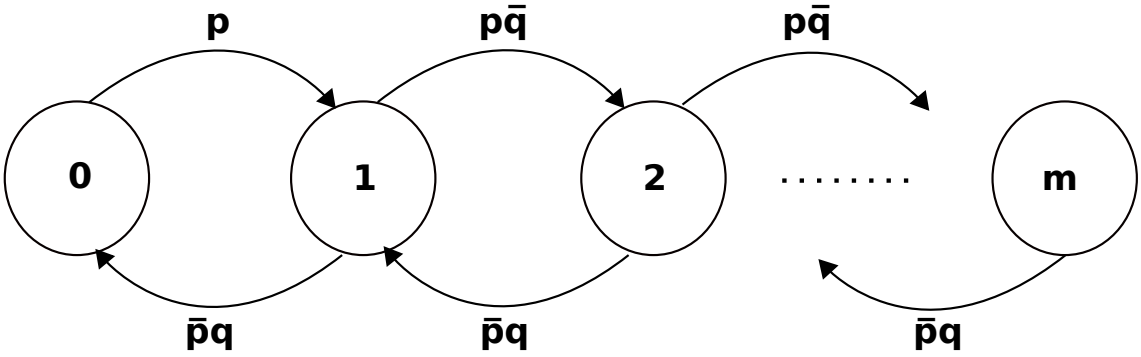

Abbildung 4.1: Markov-Kette für ein Router-Element mit endlicher Pufferlänge m

Es wird davon ausgegangen, dass der Router symmetrisch aufgebaut (alle Puffer sind gleichgroß und haben m Speicherplätze), der Paketverkehr symmetrisch ist und einer geometrischen Verteilung unterliegt, die Pakete nur aus einem Flit bestehen und keine Priorität besitzen. So könnte als Zustand der Markov-Kette die Pufferbelegung in einem der Eingangspuffer benutzt werden. Der größtmögliche Zustand der Markov-Kette ist m – der Puffer ist voll, alle hineinkommenden Pakete werden

abgewiesen. Der Kleinstmögliche hingegen ist 0 – der Puffer ist leer. Es gibt zwei Ereignisse, die zum Zustandswechsel in der Markov-Kette führen können: Ein neues Paket will in den Puffer kommen (dieses Ereignis kommt mit Wahrscheinlichkeit p vor) und ein Paket verlässt den Puffer - (dieses Ereignis kommt mit Wahrscheinlichkeit q vor).

Da die in der Abbildung 4.1 vorgestellte Markov-Kette ein klassisches M/D/1/m System beschreibt, können die Zustandswahrscheinlichkeiten π_n nach Kleinrock [24] (π_n ist die Wahrscheinlichkeit, dass Puffer mit n Phits[1] gefüllt ist) folgendermaßen berechnet werden:

$$\pi_n = \begin{cases} \left(\frac{p(1-q)}{q(1-p)}\right)^n \cdot \frac{\pi_0}{1-q} & \text{for } 1 \leqslant n \leqslant m \\ \pi_0 & \text{for } n = 1 \\ 0 & \text{for } n > m \end{cases} \tag{4.1}$$

wobei

$$1 = \pi_0 + \frac{\pi_0}{1-q} \cdot \sum_{n=1}^{m} \left(\frac{p(1-q)}{q(1-p)}\right)^n \tag{4.2}$$

was sich zu

$$\pi_0 = \frac{1}{1 + \frac{1}{1-q} \sum_{n=1}^{m} \left(\frac{p(1-q)}{q(1-p)}\right)^n} \tag{4.3}$$

unformen lässt. Die mittlere Länge $E(L)$ der Paketschlange in Puffer und auch die Paketverzögerung ist somit

$$E(L) = \sum_{n=0}^{m} n \cdot \pi_n \tag{4.4}$$

Da die Eingänge des Router-Elements mit einem geometrisch verteilten Paketverkehr belastet werden (nach der Vorbedingung), ist p eine Wahrscheinlichkeit, dass das Quellen-Element[2] ein Paket in einem bestimmten Takt generieren

[1] s. Kapitel 2.6.
[2] s. Kapitel 2.6.

wird. Die Wahrscheinlichkeit q entspricht der Wahrscheinlichkeit, dass ein Paket im Eingangspuffer einen HOL (Head of Line)-Wettbewerb gewinnt. Diese Wahrscheinlichkeiten, als Funktion von p und maximalen Durchsatz des Routers (T_{sat}) sind von den Pufferlängen unabhängig und können somit aus dem Modell für unendlichen Puffer-Router von Boot [36][3] übernommen werden:

$$q(p) = \begin{cases} \frac{1}{T_{sat}} \left(\frac{3N-1}{2N} - \frac{1}{T_{sat}} \right) p^2 - \frac{N-1}{2N} p + 1 & \text{for } 0 \leqslant p \leqslant T_{sat} \\ T_{sat} & \text{for } > T_{sat} \end{cases} \tag{4.5}$$

Die Werte für q für Router mit 2, 4, 8 Ein-/Ausgängen sind in der Tabelle 4.1 berechnet.

p	$c = 2$	$c = 4$	$c = 8$
0,1	0,973889	0,960573	0,953346
0,2	0,945556	0,917291	0,900883
0,3	0,915000	0,870154	0,842611
0,4	0,882222	0,819163	0,778530
0,5	0,847222	0,764317	0,708641
0,6	0,810000	0,705617	0,632943
0,7	0,770556	0,665242	0,618390
0,8	0,750000	0,665242	0,618390
0,9	0,750000	0,665242	0,618390
1,0	0,750000	0,665242	0,618390

Tabelle 4.1: Werte für q für verschiedenen c und p

Der Vergleich von den analytisch berechneten und anhand der Simulation gewonnenen Werten für Paketverzögerung und Durchsatz an dem Beispiel des 4x4 Routers zeigt die Abbildung 4.2.

4.3 Ein Versuch der Modellerweiterung auf die symmetrischen-MIN Strukturen

Nachdem ein Modell für das analytische Berechnen eines einzelnen Router entwickelt ist, kann man versuchen eine ähnliche Markov-Kette für ein zweistufiges MIN zu

[3] Seite 33, Formel (2.36). dabei handelt es sich um eine Näherungsformel für die diskrete Funktion $q(N)$, die mithilfe des Polynoms zweiten Grades gemacht ist, siehe [36] Die Werte für T_{sat} können ebenfalls aus der selben Thesis Seite 27 Tabelle 2.2 entnommen werden.

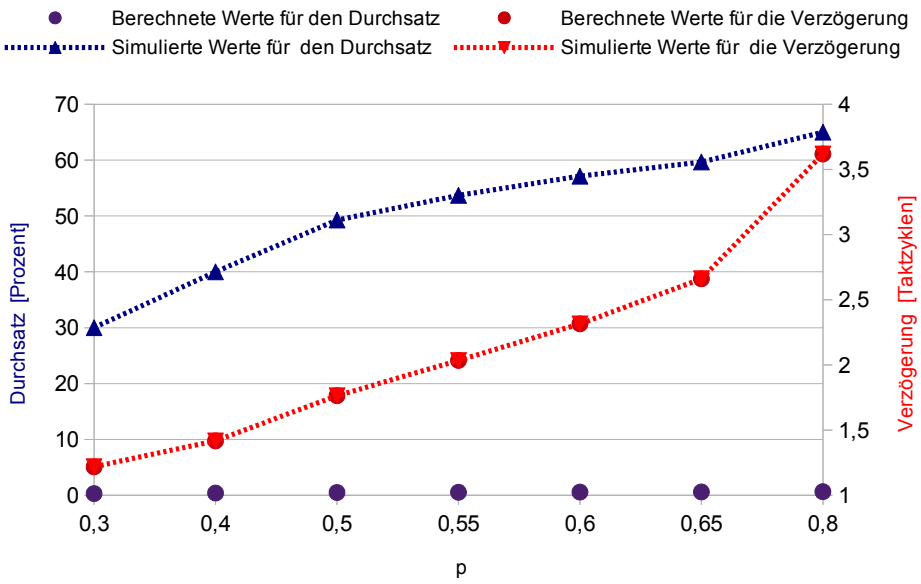

Abbildung 4.2: Vergleich zwischen simulierten und berechneten Werte für 4×4 Router mit $m = 3$

entwickeln, mit der Grenzbedingung, dass das untersuchte MIN vollkommen symmetrisch ist und aus den gleichen Router-Elementen besteht. Die Abbildung 4.3 stellt eine Markov-Kette für ein solches Netz vor.

Da es sich auch hier um ein symmetrisches System handelt, ist es ausreichend auch für dieses Modell nur einen einzigen Eingangspuffer pro MIN-Stufe zu betrachten. So hat jeder Zustand der Markov-Kette zwei Indizes: Der erste ist die Puffer-Belegung in der ersten MIN-Stufe (Puffer sind gleichgroß und haben m Speicherplätze), der zweite – die Puffer-Belegung in der zweiten MIN-Stufe (Puffer sind ebenfals gleichgroß und haben m' Speicherplätze). Zu der schon oben erwähnten Ereignis-Wahrscheinlichkeit p kommt noch hinzu q_0 - die Wahrscheinlichkeit, dass ein Paket die HOL in der ersten Stufe gewinnt und ein zusätzliches Ereignis q_1 – die Wahrscheinlichkeit, dass ein Paket die HOL in der zweiten Stufe des MINs gewinnt. Leider ist die hier vorgestellte Markov-Kette schwierig zu lösen, aus folgenden Gründen:

- Der Parameter q_1 ist unbekannt. Außerdem weist q_1 keine geometrische Verteilung auf, da der in der zweiten MIN-Stufe hineinkommende Paketverkehr schon nicht geometrisch verteilt ist. Also kann die q_1-Wahrscheinlichkeit für die Markov-Kette im besten Fall mithilfe einer anderen geometrisch verteilten Funktion lediglich angenährt werden.

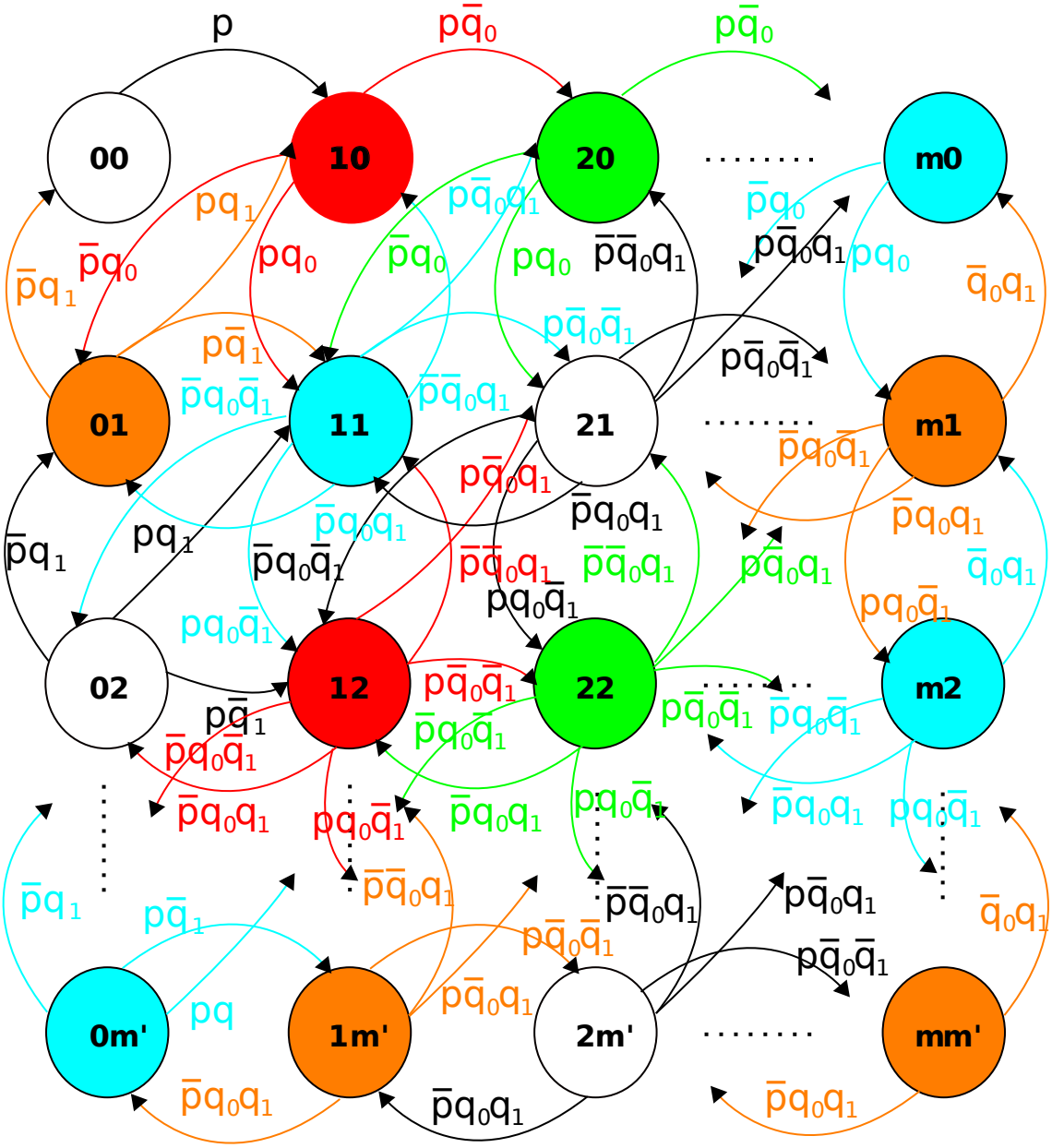

Abbildung 4.3: Markov-Kette für zwei MIN-Stufen

- Obwohl die Kette eine bestimmte Struktur aufweist, wird diese Struktur nicht in der Literatur behandelt, somit muss dafür eine spezielle analytische Lösung gefunden werden.

Wird aber nicht ein zweistufiges, sondern ein X-Stufiges MIN-Netzwerk behandelt, so muss man mit einer $X-$dimensionalen Markov-Kette rechnen, mit zusätzlichen

X Parametern (von q_0 bis q_x). Ein solches Netzwerk ist mit den zurzeit bekannten mathematischen Verfahren analytisch nicht zu lösen.

Wenn im Rahmen dieses Modells noch ein unsymmetrisches MIN mit einem unsymmetrischen Paketverkehr betrachtet werden soll oder die Paket-Prioritäten berücksichtigt werden müssen, so stößt das analytische Modell an seine Grenzen.

Um mit dieser Problematik umzugehen wird im Rahmen dieser Arbeit nicht mit den analytischen Modellen gearbeitet, sondern das spezielle Simulations-Tool *RecSim* entwickelt, das im nächsten Kapitel vorgestellt wird.

5 Simulator-Tool *RecSim* zur Untersuchung verschiedener Rekonfigurations-Architekturen

RecSim ist ein im Rahmen dieser Arbeit entwickelter Simulator, der es Entwicklern ermöglicht, Network-on-Chip zu testen. Wie andere Simulatoren liefert *RecSim* Ergebnisse für die Effizienz einer bestimmten Netzwerktopologie unter Belastung. Im Gegensatz zu anderen Simulatoren prüft der präsentierte Simulator die Wirksamkeit von Rekonfigurationsalgorithmen und ermöglicht somit die Wahl der besten Strategie zur Rekonfiguration eines Netzes. Die Ergebnisse der Messungen stehen entweder in Form eines Diagramms oder als deskriptive Statistik zur Verfügung.

5.1 Einleitung

RecSim ermöglicht es, sowohl Netzwerktopologien zu analysieren, als auch die Wirksamkeit von Algorithmen zur Optimierung zu prüfen. Diese Algorithmen sind für die Rekonfiguration der Netzwerktopologie und somit für die Verbesserung der Leistungsfähigkeit verantwortlich. Zusätzlich bietet *RecSim* die Möglichkeit unterschiedliche Verbindungsnetzwerk-Architekturen zu untersuchen. Obwohl *RecSim* dazu entworfen ist, Network-on-Chip auszuwerten, kann er zur Unterstützung von Netzwerken von Supercomputern und Router-Netzen erweitert werden. *RecSim* ist in der Lage, eine Vielzahl von Verbindungsnetzwerken, die aus atomaren Komponenten (Router, Busse, Puffer, Quellen und Ziele) aufgebaut wurden, zu simulieren.

RecSim implementiert eine Modellierungsmethode, die auf der Zusammensetzung von einfachen Baukästen (atomare Komponenten) basiert, um die gewünschte Netzwerktopologie darzustellen. Angesichts der vorgeschlagenen Methode können Entwickler den Einfluss jedes architektonischen Parameters (z.B Puffergröße und Netzlast) separat untersuchen.

Im vorgeschlagenen Simulator kann die Netzwerktopologie manuell oder automatisch festgelegt werden. Im manuellen Modus entwerfen und verbinden Entwickler manuell die atomaren Bestandteile für das Netzwerk. Der Modus wurde entworfen, um irreguläre Netzwerke zu gestalten. Der Hauptnachteil dieses Modus ist die Zeit, die benötigt wird, um den Entwurf großer Netzwerke auszuführen.

Im automatischen Modus bestimmen Entwickler Topologien, indem alle Parameter in Bezug auf das Netzwerk eingestellt werden (z.B Quellenanzahl und Zielknoten, Routing, Puffergröße) und die Topologie automatisch erstellt wird. Des weiteren kann ein Netzwerk auch durch Beschreibung generiert werden.

Außerdem ist *RecSim* in der Lage, die dynamische Rekonfiguration von Verbindungsnetzen abzubilden. Dynamische Rekonfiguration von Netzwerken ist durch die Änderungen von Netzwerk-Hardware während des Betriebs gekennzeichnet. Dies schließt die Netzwerktopologie, die Puffer- und Router-Positionen und -größen, als auch Routing ein. Solch dynamisch rekonfigurierbare Netzwerke können zunächst mit dynamisch und rekonfigurierbaren Hardware-Komponenten, wie FPGAs (field-programmable gate arrays), realisiert werden.

RecSim ist modular aufgebaut, so dass jede atomare Komponente des Netzwerks in einer separaten Klasse implementiert ist. Das Programm ist in der Programmiersprache C++ geschrieben. Dadurch können die NoC-Entwickler *RecSim* mit weiteren benutzerdefinierten Komponenten erweitern, oder die Eigenschaften der vordefinierten Komponenten je nach Bedarf ändern.

Aufgrund der beteiligten stochastischen Ereignisse (z.B zufälliger Datenverkehr), müssen Konfidenzintervall und geschätzte Verlässlichkeit der Ergebnisse während der Simulation quantifiziert werden um eine vorgegebene Genauigkeit zu erreichen.*RecSim* bietet diese statistischen Ansätze als vor-implementiertes Paket.

RecSim überwacht die Simulation und kalkuliert die Netzwerkeigenschaften (Paket-Durchsatz, -Verzögerung etc.). Zudem liefert es die statistischen Ergebnisse mit Konfidenzintervall und geschätzter Genauigkeit.

RecSim unterstützt außerdem "steady-state" (stationäre) und "terminating" (transiente) Simulationen. Des weiteren bietet der Simulator Pseudozufallszahlengeneratoren mit sehr langen Zyklen.

5.2 Komponentenbasiertes Verbindungsnetz in *RecSim*

Der Ausdruck "komponentenbasiertes Verbindungsnetz" bezieht sich auf Netzwerke, die aus einer beliebigen Kombination der Komponenten (Quelle, Ziel, Router, Busse und Puffer) bestehen. Mit diesen atomaren Komponenten kann eine große Vielfalt von paketvermittelten Verbindungsnetz-Architekturen modelliert werden und mit *RecSim* untersucht werden. Um die Komplexität des Modells begrenzt zu halten, wird angenommen, dass alle Netzwerkbestandteile vollkommen synchron, von einem globalen, internen Takt angetrieben, arbeiten. Diese Annahme ist sowohl für Network-on-Chip, die intern getaktet sind, als auch für eine große Anzahl an Applikationen, z.B NetzwerksRouter, durchaus realistisch. Dementsprechend werden

Zeit- und Leistungsergebnisse in *RecSim* in Taktzyklen gemessen.

Verbindungsnetz-Komponenten, die mit einem Netzwerk verbunden und mit *RecSim* simuliert werden können, sind:

- Router: diese Komponenten dienen der Realisierung dynamisch wechselnder Verbindungen zwischen Router-Eingängen und -Ausgängen. Eingänge und Ausgänge sind entsprechend des benötigten Ausgangs der Pakete verbunden. Falls mehrere Eingänge Pakete, die für denselben Ausgang bestimmt sind, enthalten, wird eine der Pakete zufällig ausgewählt. *RecSim*-Router werden verzögerungslos betrieben: Der Transport eines Flit durch einen Router findet unverzüglich statt.

- Busse: Komponente, die den Routern gleichen. Im Gegensatz zu Routern kann allerdings nur ein Flit per Taktzyklus von einem beliebigen Router-Eingang zum Router-Ausgang geschaltet werden.

- Puffer: Speicher von Nachrichten. Es ist notwendig, dass jeder Router-Eingang mit einem Puffer verbunden ist. *RecSim* arbeitet mit single-queued Eingangspuffern. Die Länge der Warteschlange eines jeden Puffers kann individuell festgelegt werden. Ausgangspufferung oder Shared- Pufferung werden nicht unterstützt.

- Quellen: Erzeuger von Verkehr, der dem Netzwerk angeboten wird. Verschiedene Unicast-Zielverkehrsmuster können erzeugt werden. Kann ein Paket das Netzwerk aufgrund von Überlastung nicht betreten, wird es von der Quelle verworfen. Alle Pakete, die das Netzwerk betreten können, werden ihre Ziele irgendwann dank des Backpressure-Mechanismus erreichen.

- Ziele: Darstellung der Ausgänge des Netzwerks. Sie sind für das Löschen von Paketen von den Ausgängen, sobald diese ankommen, verantwortlich.

- Leitungen: Verbindung von Quellen mit ihren Router-Puffern. Leitungen, wie auch Router und Busse, arbeiten verzögerungslos.

Zusätzlich zu diesen Komponenten bietet *RecSim* auch Analysatoren zur Leistungsmessung. Analysatoren können mit Puffern, Quellen und Zielen verbunden werden, um Messpunkte für die gewünschten Werte zu definieren. Ein Analysator kann mit diversen Komponenten verbunden werden, was zum arithmetischen Mittelwert von allen verbundenen Komponentenergebnissen führen wird. Je nach Art der Komponente werden mehrere Leistungsmessungen unterstützt (Paket-Durchsatz, -Verzögerung etc.).

RecSim unterstützt zwei Beobachtungsmodi der Wertmessungen - "ausschließlich statistische" und "allgemeine Messung". Im "ausschließlich statistischen" Modus speichert die Software nicht alle gemessenen Werte, sondern zeichnet nur die statistischen Parameter auf (wie beispielsweise den Mittelwert der Verteilung, das Konfidenzintervall und die geschätzte Genauigkeit, etc). Konfidenzintervalle werden durch eine Spektralanalyse der Proben [16] ermittelt. Die geschätzte Genauigkeit eines Simulationsergebnisses wird als relative Hälfte des entsprechenden Konfidenzintervall definiert. Mit diesen Informationen ist es möglich, das Vertrauen in eine Annäherung durch die Bereitstellung eines Konfidenzintervalls auszudrücken. Informell gesagt, bedeutet ein Konfidenzintervall von 95% beispielsweise, dass die Wahrscheinlichkeit, dass der reale Wert der Messung außerhalb des kalkulierten Konfidenzintervalls liegt, 5% entspricht.

Allerdings kann es in einigen Fällen irreführend sein, Schlüsse aus Simulationsergebnissen, die nur Mittelwerte enthalten, zu ziehen. Der Mittelwert enthält keine Informationen über Extremwerte und die Verteilung der betroffenen Messung. Zur Betrachtung der vollständigen Verteilung der Messung wird der "allgemeine Messungsmodus" eingeführt. In diesem Modus werden alle Werte, ausgehend von der Einschwingphase, in einem Vektor gespeichert.

Bei der Simulation des Netzwerks unterscheidet *RecSim* zwischen Übergangsphase und stationärer Phase. Die anfängliche Übergangsphase der Simulation kann die Ergebnisse im ausschließlich statistische stationärer Phase, insbesondere das Konfidenzintervall, verzerren. *RecSim* verwendet den Schruben-Test [40] um die Ende der Übergangsphase zu ermitteln, und damit vorübergehende Einflüsse auf die Leistungsmessung zu beseitigen.

Der Simulator leitet Pakete entlang des Routing-Weges weiter und simuliert das Verhalten beteiligter Netzwerk-Komponenten (Unterkapitel 5.4 beschreibt diese Funktionsweisen). Es wird angenommen, dass die Größe eines Flits der Größe eines Phit (physical unit) entspricht. Ein Phit repräsentiert die Größe einer Informationseinheit, die ein Netzwerk innerhalb eines Taktzyklus im Netzwerk von einem Puffer zum folgenden weiterleiten kann.

5.3 Generierung von 2^k-MIN

RecSim ist für die Erforschung von Rekonfigurationstechniken in Netzwerken entwickelt worden, wodurch die Paketverzögerung durch die Anpassung von Netzwerktopologien für verschiedene Verkehrsmuster reduziert wird. Im Weiteren wird die Funktionsweise des *RecSim*s am Beispiel der 2^k-MINs (s. Kapitel 3) demonstriert. Eine wichtige Eigenschaft des 2^k-MINs ist, dass ein solches Netzwerk in mehrere unabhängige Segmente unterteilt werden kann (Abbildung 5.1). Anschließend ist es möglich jedes dieser Segmenten lokal zu optimieren.

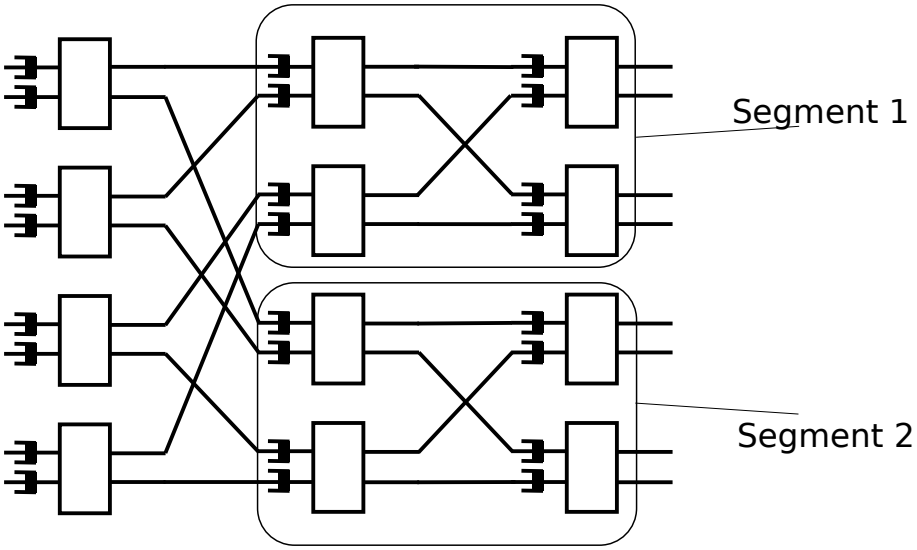

Abbildung 5.1: 2^k-MIN.

Der Durchsatz des Netzwerks, die Paketverzögerung und die belegte Fläche auf dem Chip wurden vor und nach Durchführung der Rekonfiguration analysiert. Andere Messungen, wie beispielsweise Energieverbrauch, liegen nicht im Rahmen dieser Arbeit.

Die Topologie muss vor Beginn der Netzwerksimulation beschrieben werden. Die Topologie mit einem Generationsschema oder einer Generationsregel, anstelle einer manuellen Definition der Topologie, zu beschreiben, ist zu bevozugen. Tutsch [47] leitet eine Formel für automatische Router-Verbindungen für Modified Data Manipulator-Topologien mit N Eingängen und N Ausgängen ab. Unglücklicherweise funktionieren die Beschreibungsformeln, die für die Netzwerkgenerierung verwendet werden können, nur für reguläre Topologien. Die 2^k-MINs Topologie ist allerdings nicht regulär. Diese Topologie verwendet Router, die unterschiedlich groß sind und an verschiedenen Stellen platziert sind. Demnach ist es generell schwierig, diese Topologie automatisch mit irgendeiner Parameterformel zu generieren.

Um den Mechanismus der automatischen Generierung eines 2^k-MIN zu ermöglichen, wurden zwei Techniken entwickelt. Die Generierung beginnt mit einem einzigen $c \times c$ Router. Die erste Technik heißt **Alpha+**/**Alpha-**-Zerfall/Synthese und ist in dem Kapitel 3 vorgeschlagen worden. Die zweite Technik, die in *RecSim* implementiert wird, wird $D[X]/\ S[\text{-}]$-Zerfall/Synthese genannt. Beide Techniken spalten einen einzigen $c \times c$ Router in ein MIN, das aus zwei Stufen besteht. Während einer **Alpha+**/**Alpha-**-Zerfalls besteht einer der sich ergebenden MIN-Abschnitte aus 2×2 Routern. Im Gegensatz zu dieser ersten Technik kann einen Router beim $D[X]/\ S[\text{-}]$-Zerfall in zwei benutzerdefinierte, ungleiche MIN-Stufen geteilt werden.

Die Größe der Router für jeden Abschnitt muss jedoch c Eingängen und Ausgängen für $c = 2^{2*k}$ (k=1,2,..) gleichen.

Ein Beispiel für diese neue Topologie-Generierungstechnik für 2^k-MIN ist in Abbildung 5.2 dargestellt.

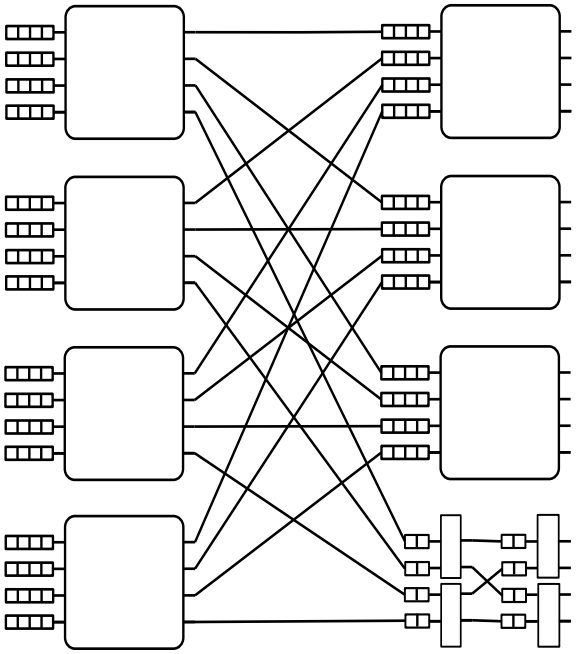

Abbildung 5.2: Topologie-Generierungstechnik für 2^k-MIN

Es wird mit einem 16×16 Router mit Puffer-Tiefe 8 an jedem Eingang begonnen und dieser wird in MIN in zwei Abschnitte zerlegt. Sowohl der erste Abschnitt, als auch der zweite bestehen aus 4×4 Routern. Die Pufferverteilung zwischen den Abschnitten ist im Verhältnis 4:4 (siehe Abbildung 5.3). Nun wird der untere rechte der 4×4 Router geteilt und erreicht die gewünschte Topologie (Pufferverteilung 2:2).

Um die Darstellung für die hier vorgeschlagene Methode zur Netzwerk-Generierung zu vereinfachen, wird die folgende Syntax vorgeschlagen:
$D[X](Sw,\ m_1)$- der Zerfall wird auf den Router Nummer Sw angewendet (die Nummerierung geht spaltenweise von oben nach unten); das resultierende MIN beinhaltet zwei Stufen, wobei die zweite Stufe aus einem Router mit X Eingängen und Ausgängen besteht, während die Größe seiner Eingangspuffer m_1 ist. Die erste Stufe besteht aus X Routern. Jeder einzelne von ihnen hat c/X Eingänge und Ausgänge und seine Puffer haben jeweils die Größe $m_0 - m_1$ (m_0 ist die Größe des Eingangspuffers im ursprünglichen Router Sw, und c ist die Anzahl seiner Eingänge).
Durch Verwendung dieser neuen Darstellung kann nun die Topologie des Netzwerks

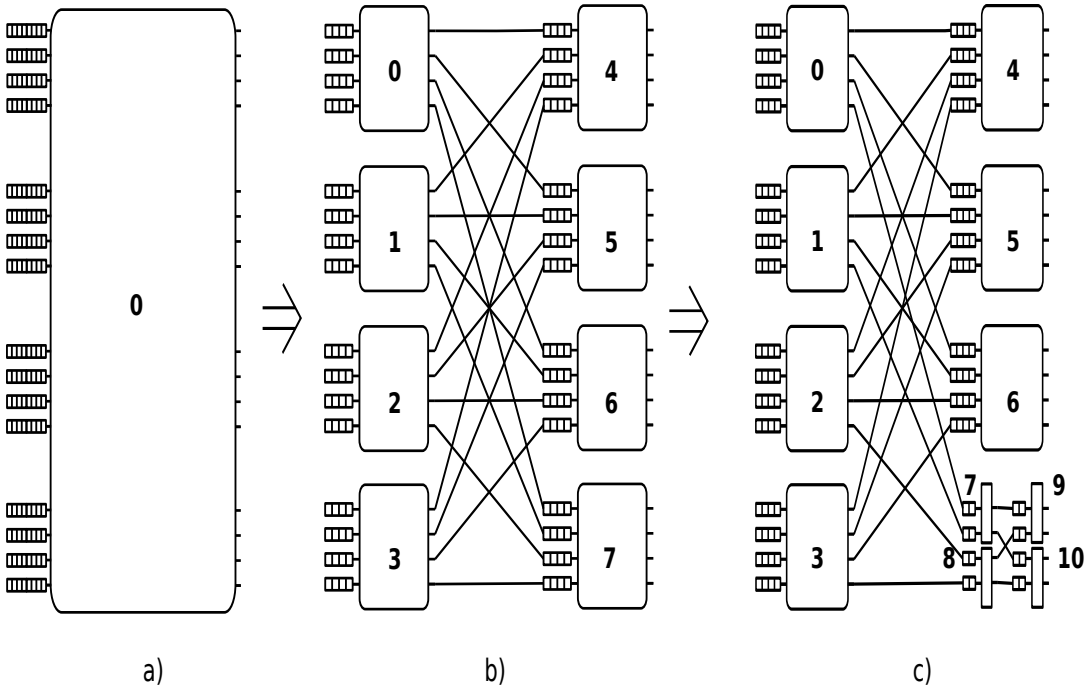

Abbildung 5.3: Topologie-Generierungstechnik mit $D[X]$-Zerfall

auf der Abbildung 5.2 wie folgt beschrieben werden:

$$D[4](0,4) \rightarrow D[2](7,2) \tag{5.1}$$

Nach der Folge von $D[X]$ Zerfälle, kann die endgültige MIN-Topologie-Fläche geschätzt werden, indem die Anzahl die Kreuzungspunkte aller Router-Komponenten verwendet werden. Ein einziger Router mit c Eingängen und Ausgängen setzt sich aus c^2 Kreuzungspunkten [27] zusammen. Es wurde mit einem Netzwerk, das aus einem $N \times N$ Router bestellt und daher die Größe von N^2 Kreuzungspunkten hat, begonnen. Nach der Folge von $D[X]$ Zerfälle, wird die Anzahl der Kreuzungspunkten reduziert (mit Ausnahme von $c = 4$, bei dem es unverändert bleibt), wegen

$$\begin{aligned} c^2 &\geq X \cdot \left(\frac{c}{X}\right)^2 + \frac{c}{X} \cdot X^2 \\ c &\geq \frac{c}{X} + X \qquad \text{für } c > X, c \geq 4 \quad \text{und} \quad c, X \in 2^k, k = 1, 2, 3... \end{aligned} \tag{5.2}$$

Wird die Chip-Flächen-Reduzierung für $D[X]$-Zerfall als Funktion $R(D[X])$ definiert, so kann die Chip-Flächen-Schätzung des endgültigen MIN wie gefolgt

geschrieben werden:

$$A_{chip} = N^2 - R(D[X])_1 - R(D[X])_2 ... - R(D[X])_n \tag{5.3}$$

wobei $R(D[X])_n$ die Kreuzungspunkt-Reduzierung nach n-ten Zerfall repräsentiert.

Bei dem Vergleich zwischen $D[X]$-Zerfall und im Kapitel 4 beschriebenen **Alpha+**/**Alpha-**-Zerfällen, wird deutlich, dass **Alpha+**/**Alpha-**-Zerfälle jeweils mit $D[2]$ und $D[c/2]$ -Zerfällen korrespondieren.

Der Umkehrprozess zum Zerfall wird Synthese genannt. Im Gegensatz zum Zerfall, wird die Synthese nicht ausschließlich auf einen einzigen Router angewendet, sondern auf ein Netzwerksegment, welches als aus zwei Stufen bestehende 2^k-MIN beschrieben werden kann. Die Darstellung der Synthese in der vorgeschlagenen Syntax ist folgendermaßen:

S[-](Sw) - die Synthese wird auf ein 2^k-MIN angewendet, das aus zwei Stufen besteht, wobei die Router Nummer Sw die des oberen, rechten Routers dieses MIN ist. Nach der Synthese wird das MIN durch einen einzigen Router ersetzt. Die Eingangspuffer-Größe dieses neuen Routers ist die Summe der Größen der ursprünglichen Puffer (Beispiel auf der Abbildung 5.4).

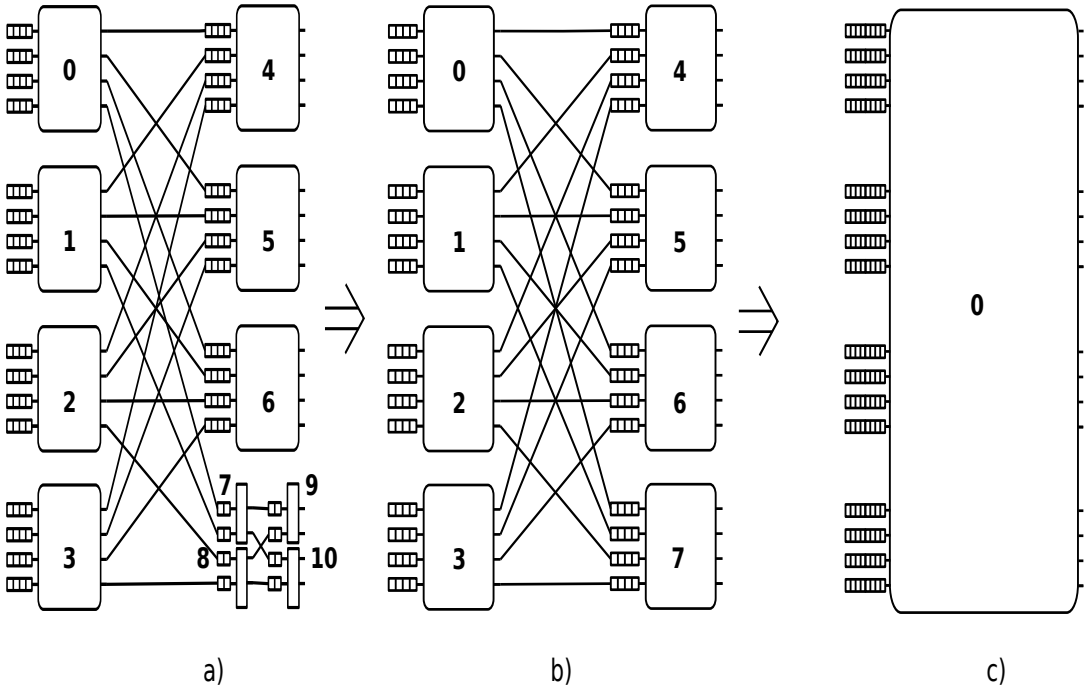

Abbildung 5.4: die Synthese S[-](9) $\rightarrow$ S[-](4)

5.4 Simulation und Rekonfiguration

Dank der heutigen technischen Fortschritte im Bereich rekonfigurierbarer Hardware ist die Verwendung von dynamisch rekonfigurierbaren Netzwerken auf der Router-Ebene oder einer höheren Netzwerkebene möglich. Das bedeutet beispielsweise, dass die internen "switching fabrics" der Router in einem Cluster-System rekonfigurierbar sind. In einem SoC kann das gesamte NoC reorganisiert werden. Viele Parameter der Verbindungsnetz-Architektur können rekonfiguriert werden: Pufferspeicher, Pufferstrategien, Router-Größe etc. Es ist also durchaus möglich, Algorithmen zur Rekonfiguration von 2^k-MIN-Netzwerktopologien in Hardware zu implementieren.

Jede Simulation beginnt mit einem leeren Netzwerk und generiert Verkehr entsprechend den gegebenen Verkehrsverteilungen eines Zufallsgenerators. Der Simulator leitet ein Paket entlang des Routing-Weges weiter und simuliert das Verhalten der beteiligten Netzwerk-Komponenten. *RecSim* gehört[1] zur Klasse der diskreten zeitdiskreten Simulatoren. Es gibt drei Arten von Komponenten im Netzwerk: Synchrone Elemente (z.B Puffer, Quellen, Ziele), asynchrone Elemente (z.B Router, Busse, Leitungen) und Messungen. Jedes Element wird einmal pro Taktzyklus aktiviert. *RecSim* simuliert den gleichzeitigen Betrieb der synchronen Elemente wie folgt: Bei jedem Takt aktualisiert *RecSim* zuerst asynchrone Komponenten, anschließend synchrone Komponenten und zum Schluss die Einheiten zur Messung (Analysatoren). Diese Methode wird üblicherweise für parallel arbeitende Systeme verwendet und ermöglicht eine Reduzierung der Simulationszeit.

Erreicht ein Paket eine Komponente, an die ein Analysator angeschlossen ist, wird die Messung (der Art der Beobachtung entsprechend) aktualisiert. Diese Aktualisierungen von Beobachtungsergebnissen können als Stichprobe von einem Zufallsexperiment gesehen werden. Normalerweise ist es wünschenswert, dass der Mittelwert aller zufälligen Experimente in der stationäre Phase der Messung gewonnen wird. Deshalb wendet *RecSim* den Schruben Test [40] auf Simulationsintervalle an, um vorübergehende Einflüsse auf Messungen zu verhindern. Sind diese Einflüsse klein genug, so nimmt *RecSim* an, dass das Netzwerk einen stabilen Zustand erreicht hat und fängt an, Ergebnisse für die Messungen zu sammeln.

Es werden Konfidenzintervalle kalkuliert, um zu entscheiden, ob die Anzahl der gesammelten Stichproben, die bis jetzt gesammelten Ergebnisse der Messung, ausreichen. Die berechneten Konfidenzintervalle können entweder als Messergebnisse ausgegeben werden, um die Netzwerkleistung aufzuzeigen oder als Trigger für Algorithmen zur Rekonfiguration verwendet werden.

Zu einem bestimmten Zeitpunkt ändert sich der Verkehr im Netzwerk und bewirkt die Erfüllung der Trigger-Regel. Der Rekonfigurationsprozess beginnt. Der Rekonfigurationsprozess hängt von der gewählten Netzwerktopologie und vom getesteten

[1] aufgrund des synchronen Betriebs aller Komponenten im Netzwerk

Rekonfigurationsalgorithmus ab. Es werden 2^k-MINs erforscht und mit $D[X]$- und $S[\text{-}]$-Prozessen rekonfiguriert. Es können allerdings auch andere Topologien getestet werden.

RecSim kontrolliert das Netzwerk während des Rekonfigurationsprozesses, überprüft, ob die Randbedingungen während der Rekonfigurationsphase eingehalten werden, also beispielsweise ob die genutzte Chip-Fläche nicht übertroffen wird oder die Paketverzögerung den vorgegebenen Wert nicht überschreitet. Des weiteren beobachtet *RecSim* die Länge der Rekonfigurationsphase, ihre Komplexität und die Effizienz der ausgeführten Änderungen im Netzwerkverkehr.

Der Ausgang von *RecSim* besteht aus zwei Dateien. Die erste Datei beschreibt die Topologie des Netzwerks in DOT-Sprache. Die DOT-Sprache ist ein geläufiges Format zur Beschreibung eines gerichteten Graphen [43], welcher automatisch in eine graphische Form synthetisiert werden kann. Graph Visualization Software - Graphviz [14] wird benutzt, um die Beschreibung von der DOT-Sprache zu synthetisieren, um die Netzwerktopologie-Darstellung zu erhalten.

Die graphische Ausgabe der Netzwerktopologie ist immer dann von Vorteil, wenn die Rekonfiguration stattfindet. Auf diese Weise kann der Entwickler die Korrektheit jeder Rekonfiguration leicht überprüfen.

Die zweite Ausgangsdatei enthält die Statistik der Netzwerk-Charakteristika, der vordefinierten Messungen (z.B Paketverzögerung, Quellen-Durchsatz, Puffer-Warteschlange etc.) Durch die Analyse dieser Datei sind Entwickler in der Lage, sowohl Engpässe im Netzwerk zu erkennen, als auch die Ergebnisse von Vergleichen der verschiedenen Topologien zu erhalten.

Im Rahmen dieser Arbeit wurde der Rekonfigurationstrigger als Anwachsen der Paketverzögerung an den Netzwerk-Zielen definiert. Wenn die Änderung der Verzögerung an allen Netzwerk-Zielen den Schwellengrenzwert überschreitet, versucht der Algorithmus $D[X]$- und $S[\text{-}]$-Prozesse anzuwenden, um so die Netzwerktopologie dem Verkehr anzupassen. Kann eine neue, bessere Topologie gefunden werden, sucht *RecSim* nach der kürzesten Sequenz der $D[X]$- und $S[\text{-}]$-Schritten, die zu einer solchen Topologie führen, und erfüllt diese Schritte.

Bevor der $D[X]$-Zerfall allerdings begonnen werden kann, muss eine Rekonfigurationsvorbereitung durchgeführt werden (Vorbereitungsphase). Während der Vorbereitungsphase wird die Größe aller Puffer, die geteilt werden, auf m_1 begrenzt. Somit ist m_1 die Größe des neuen Puffers in der ersten MIN-Stufe, die nach dem Teilen entsteht. Der Pufferabschnitt, der zum Puffer in der zweiten MIN-Phase rekonfiguriert werden wird, muss komplett unbesetzt sein (andernfalls könnten Pakete übrig bleiben, die nach der Teilung nicht zu ihrem Ziel geleitet werden können). Daher kann der $D[X]$-Zerfall nicht angewendet werden, bis dass der Teilungspuffer nicht wenigstens m_2 unbesetzte Pufferplätze frei hat, wobei m_2 die Größe des neuen Puffers in der zweiten MIN-Stufe nach dem Zerfall ist. Die Vorbereitungsphase der Rekonfiguration dauert so viele Takte, wie für das Räumen der

benötigten Puffer-Plätze in allen Puffern, die geteilt werden, benötigt wird.

Nachdem die Rekonfiguration beendet wurde, simuliert *RecSim* das Netzwerk, bis der abschließende Simulationszustand erreicht ist. Beispielsweise kann solch ein Zustand der stationäre Zustand der Paketverzögerung für die Netzwerktopologie oder einfach der Ablauf von n beliebigen Taktzyklen sein.

5.5 Ergebnisse

Diese Fallstudie präsentiert den Simulator *RecSim* mittels eines Netzwerks, welches 16 Eingänge und 16 Ausgänge (siehe Abbildung 5.5) und eine Puffergröße von 8 an jedem Router-Eingang hat. Die verfügbare Fläche auf dem Chip ist auf 170 Kreuzungspunkte begrenzt. Tabelle 5.1 zeigt das Verkehrsprofil im Netzwerk.

Quelle	$\mathbf{P}_{trans}(\mathbf{Q}_i)$	$\mathbf{P}_{rec}(\mathbf{T}_0)...\mathbf{P}_{rec}(\mathbf{T}_{15})$
Q_0,Q_1	0,95	0,95/16
Q_2...Q_{15}	0,1	0,1/16

Tabelle 5.1: Verkehrsprofil im Netzwerk während der 10 000 ersten Takte

In Tabelle 5.1 ist Q_i die Bezeichnung für jede Quelle i; P_{trans} (Q_i) ist die Wahrscheinlichkeit, dass die Quelle i ein Paket per Taktzeit-Einheit sendet; P_{rec} (T_x) ist die Wahrscheinlichkeit, dass der Zielknoten x (T_x) ein Paket von der Quelle i (Q_i) per Taktzeit-Einheit erhält.

Der Leser sollte bemerken, dass die Aktivität fast aller Quellen niedrig ist (P_{trans} =0,1). Allerdings sind die Wahrscheinlichkeiten $P_{trans}(Q_0)$ und $P_{trans}(Q_1)$ nahe zu Eins per Taktzeit-Einheit (P_{trans} =0,95).

Sobald ein Paket generiert ist, wird seine Zieladresse gleichzeitig erstellt. Anschließend wird das Paket zum adressierten Ziel gesendet. Die Wahrscheinlichkeit, dass ein bestimmtes Ziel adressiert wird, ist gleichgroß für jedes Ziel, gemäß $P_{trans}(T_0)$=$P_{trans}(T_1)$=...=$P_{trans}(T_{15})$.

Nach 10 000 Takten wird der Netzwerkverkehr geändert. Das neue Verkehrsprofil ist in Tabelle 5.2 zu sehen.

Quelle	$\mathbf{P}_{trans}(\mathbf{Q}_i)$	$\mathbf{P}_{rec}(\mathbf{T}_0)$, $\mathbf{P}_{rec}(\mathbf{T}_1)$	$\mathbf{P}_{rec}(\mathbf{T}_2)...\mathbf{P}_{rec}(\mathbf{T}_{15})$
Q_0...Q_{15}	3/16	0,8/16	0,1/16

Tabelle 5.2: Verkehrsprofil in das Netzwerk nach der 10 000 ersten Tackte

Man kann sehen, dass die neue Sende-Wahrscheinlichkeit für alle Quellen (P_{trans} = 3/16) gleich ist. Es treten aber Hotspots an zwei Zielen T_0 und T_1 auf. Nach den Verkehrsänderungen ist das Netzwerk überlastet. Um das Problem der Überlastung

aufzuzeigen, wird die Netzwerktopologie nicht unverzüglich rekonfiguriert, sondern für 10 000 Takte unverändert gelassen. Danach beginnt *RecSim* die Rekonfiguration. Nach der Analyse verschiedener Topologien wählt *RecSim* die Topologie, die in Abbildung 5.6 zu sehen ist.

Allerdings gibt es keine Möglichkeit, von Topologie A, der Ausgangstopologie, gezeigt in Abbildung 5.5, zu Topologie C, gezeigt in Abbildung 5.6, direkt in einem einzigen Schritt zu

rekonfigurieren. Um dies zu erreichen, gibt es zwei theoretische Alternativen: (I) den Router 16×16 zu synthetisieren und ihn dann mit $D[8]$-Zerfall zu teilen oder (II) beide übriggebliebenen 8×8 Router mit $D[4]$ zu teilen und diese dann mit den Routern X_2 bis X_9 zu synthetisieren. Die erste Option ist nicht möglich, da der 16×16 Router die Fläche von $16^2 = 256$ Kreuzungspunkten belegen würde, diese jedoch auf 170 Kreuzungspunkte begrenzt ist (nach der Vorbedingung). Daher bleibt für *RecSim* nur die zweite Option: Topologie A (Abbildung 5.5) zu Topologie B (Abbildung 5.7) und danach zu Topologie C (Abbildung 5.6) zu rekonfigurieren. Die folgende $D[X]$ Bezeichnung repräsentiert diese von Topologie A ausgehende Rekonfiguration:

Die erste Rekonfiguration (beide Zerfälle können parallel ausgeführt werden) entspricht:

$$\begin{matrix} D[4](X_1, 4) \\ D[4](X_0, 4) \end{matrix} \tag{5.4}$$

Die neue MIN-Struktur ist Topologie B. Hier wird ebenfalls der nächste Rekonfigurationsschritt zu Topologie C nicht unverzüglich ausgeführt, sondern erst nach 10 000 Taktzyklen, um die Netzwerkeigenschaften von Topologie B zu zeigen. Anschließend wird der zweite Rekonfigurationsprozess durchgeführt (beide Zerfälle können parallel ausgeführt werden)

$$\begin{matrix} S[\text{-}](X_0) \\ S[\text{-}](X_4) \end{matrix} \tag{5.5}$$

Nachdem die letzte Rekonfiguration vervollständigt wurde, wird das Netzwerkverhalten für die nächsten 30 000 Takte protokolliert und analysiert. Die Abbildungen 5.8 a)-d) zeigen die Ergebnisse. Um die Darstellung der Ergebnisse im Rahmen zu halten, wird der Verlauf von nur vier Komponenten-Messungen vorgestellt: Die Warteschlangenlänge in Puffer B_0 und B_8 (Abbildung 5.8 a), b)) und die Paketverzögerung der Ziele T_0 und T_{15} (Abbildung 5.8 c), d)).

In der ersten Phase (die ersten 10 000 Takte) sind die Puffer B_0, B_1 überladen (Abbildung 5.8 a)), das übrige Netzwerk ist jedoch aufgrund der symmetrischen

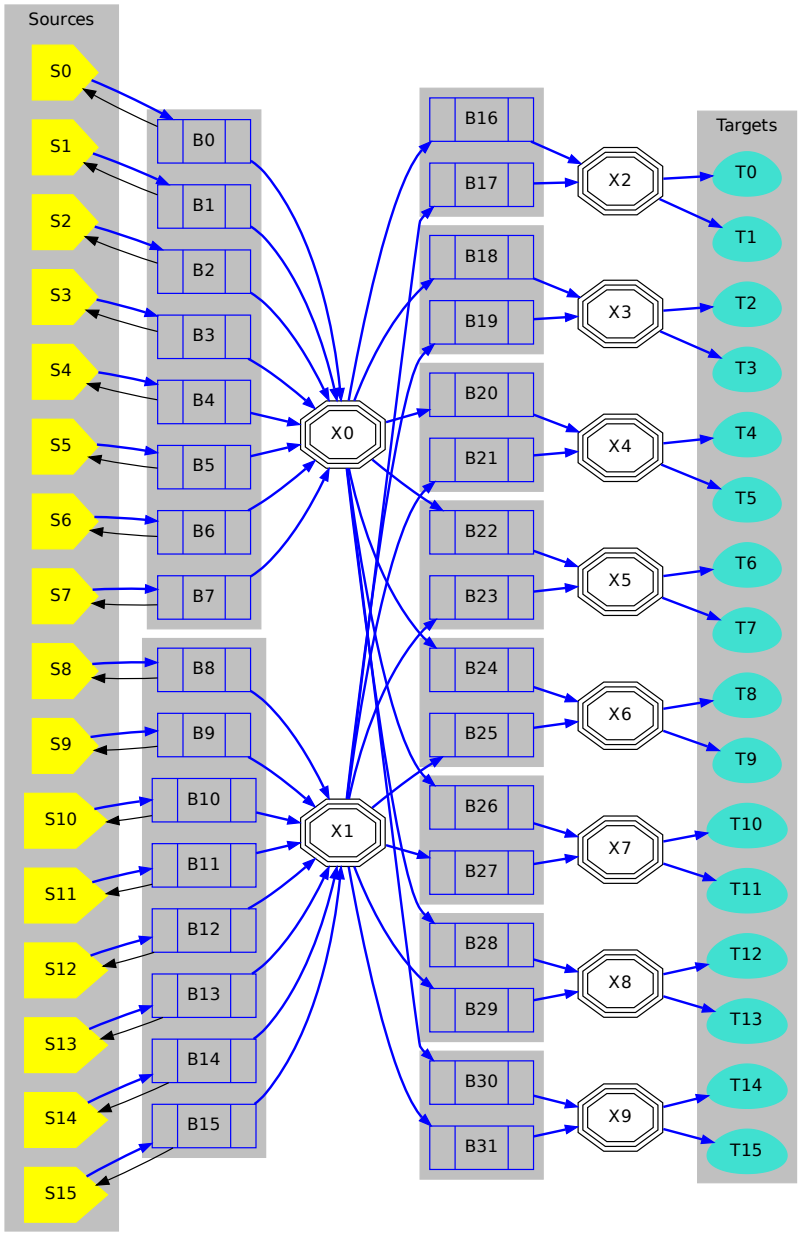

Abbildung 5.5: Netzwerktopologie A (generiert von *RecSim*).

Adressenverteilung an den Ausgängen von Router X_0 nicht überladen. Dennoch lässt sich die größtmögliche mittlere Belastung an jedem X_0 Ausgang mit $mean_{max}$=(0, 1∗

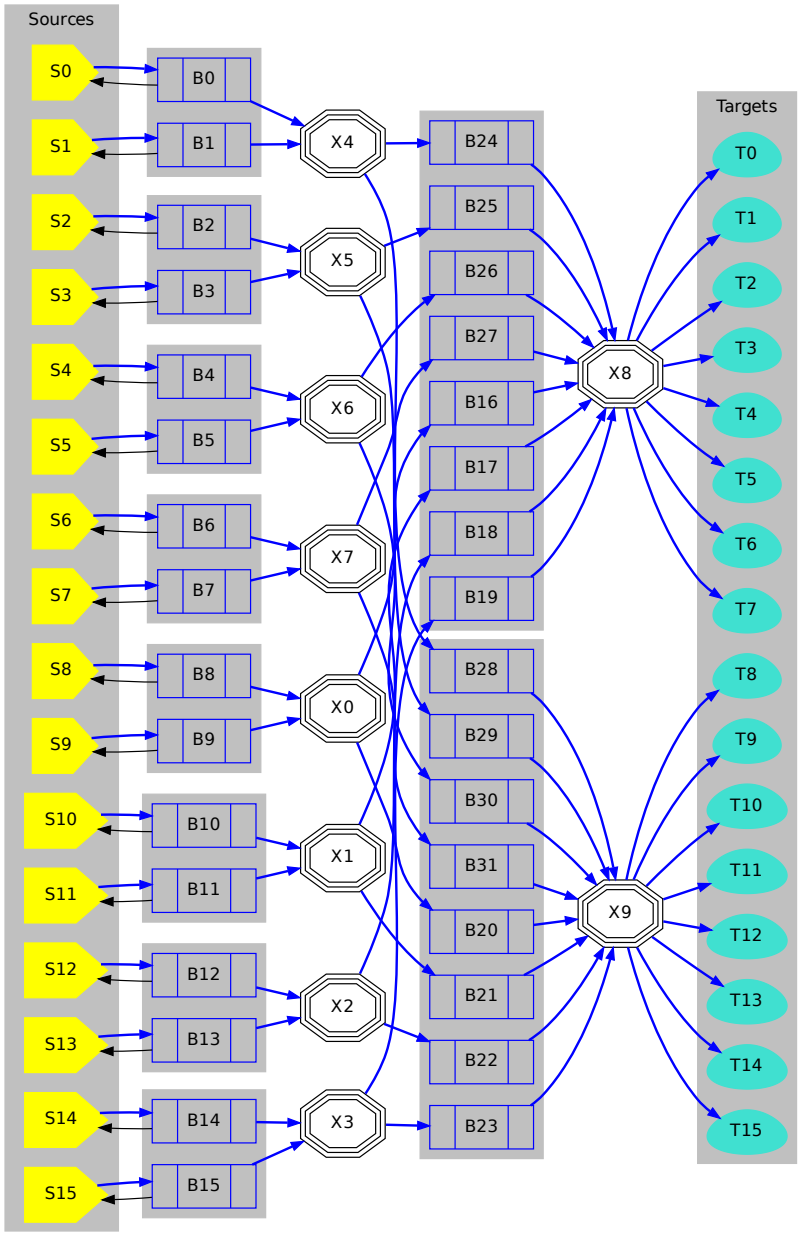

Abbildung 5.6: MIN-Topologie C (generiert von *RecSim*).

$6 + 0,95 * 2)/8 = 0,3125$ schätzen. Die Verzögerung der Pakete an allen Zielen $T_0 ... T_{15}$ ist identisch. Die Simulation mit *RecSim* schätzt es auf 5,53 Takte (das

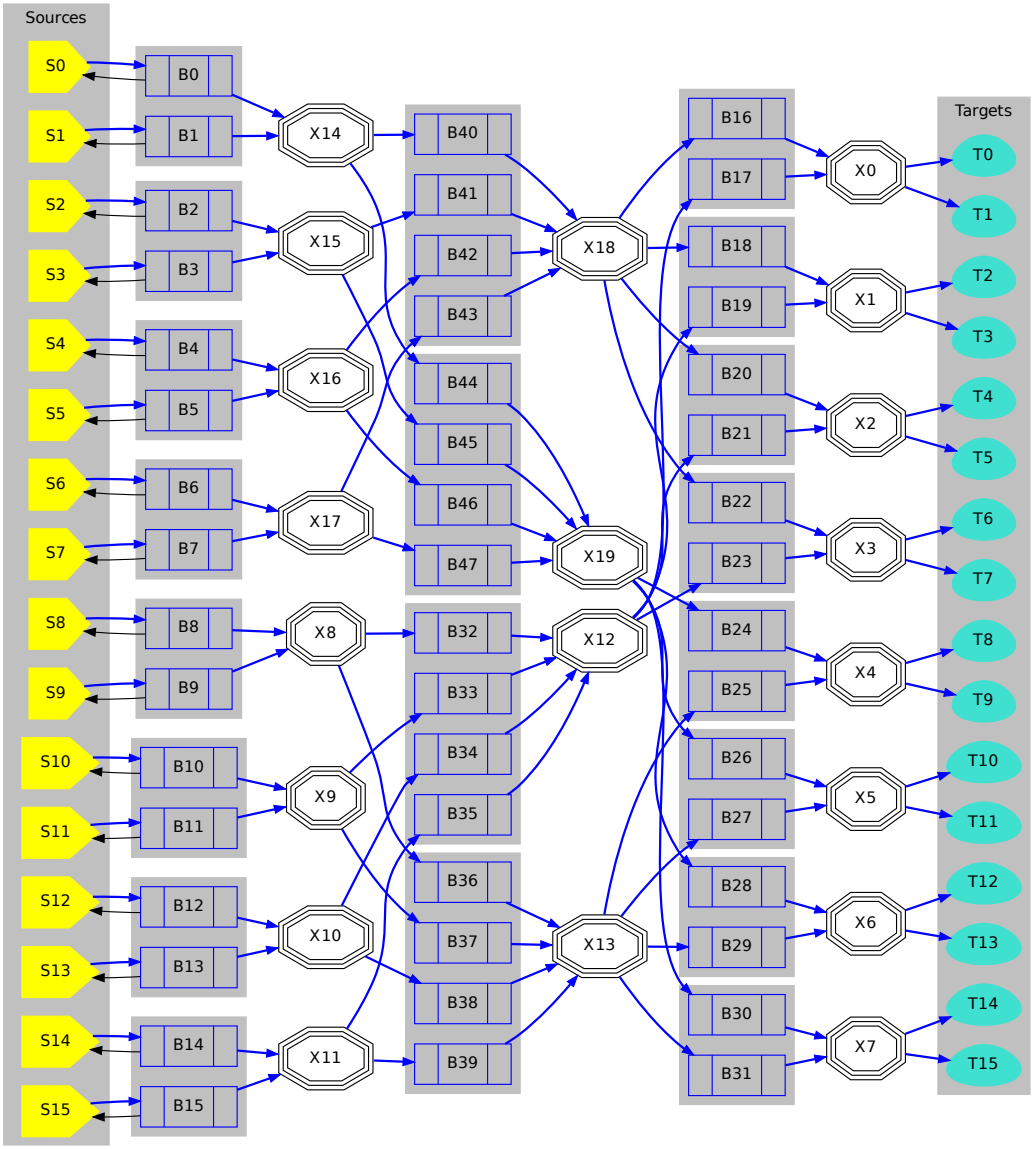

Abbildung 5.7: MIN-Topologie B (generiert von *RecSim*).

Konfidenzlevel ist 0,95 und die relative Messabweichung ist 0,04).

In der zweiten Phase (zwischen 10 000 und 20 000 Takten) überlastet die Verkehrsänderung, wie oben aufgeführt, das gesamte Netzwerk. Wegen des Hotspot-Verkehrs und des Rückdrucks laufen die Puffer im gesamten Netzwerk voll (siehe Abbildung 5.8 a), b)). Die Pakete bleiben vor den Routern stecken und die Verzögerung

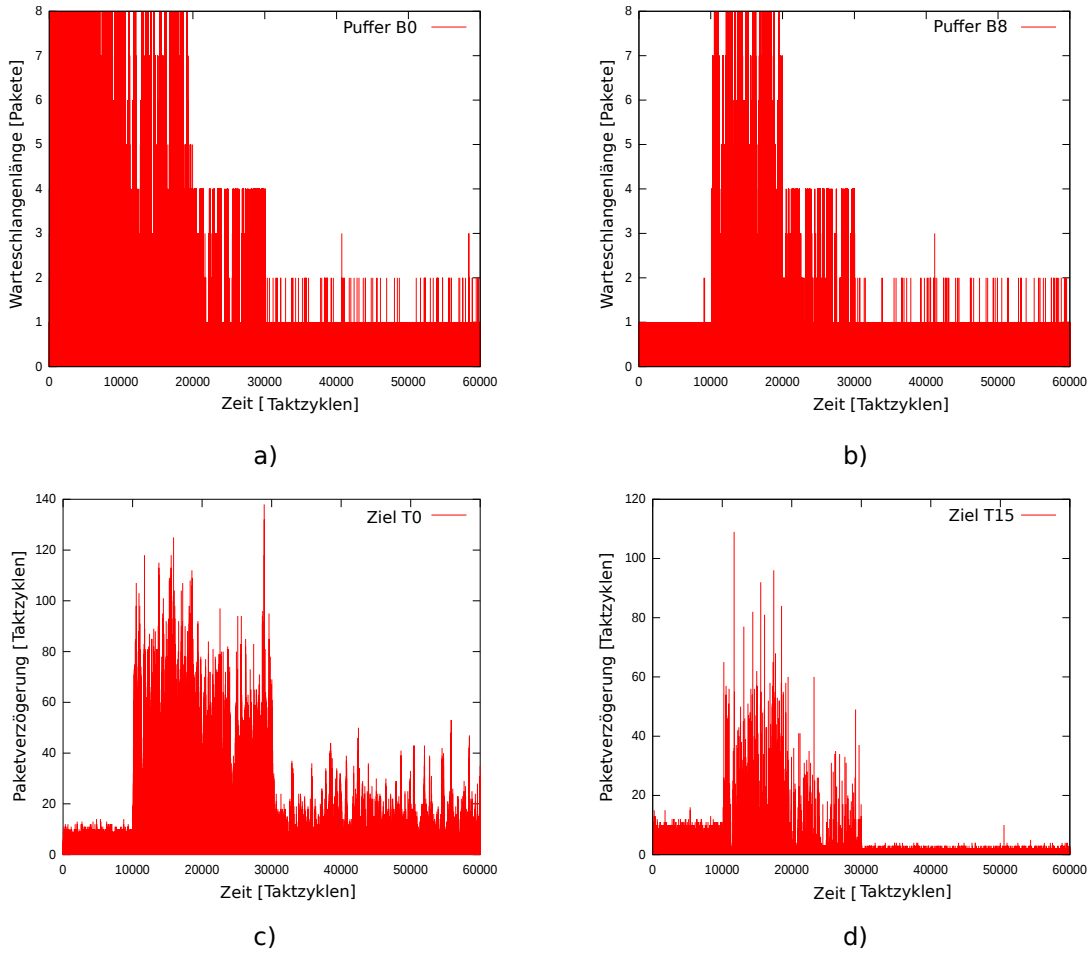

Abbildung 5.8: a) Die Warteschlangengröße in Puffer B_0. b) Die Warteschlangengröße in Puffer B_8. c) Die Paketverzögerung für das Ziel T_0. d) Die Paketverzögerung für das Ziel T_{15}.

der Pakete vergrößert sich nicht nur für die Hotspot-Ziele (z.B für T_0, siehe Abbildung 5.8 c)), sondern auch für alle anderen (non-hot-spot) Ziele (z.B für T_{15}, siehe Abbildung 5.8 d)). Dies macht Rekonfiguration notwendig.

Nach dem ersten Rekonfigurationsschritt ändert sich die Netzwerktopologie zu Topologie B. Der Abschnitt zwischen 20 000 und 30 000 Takten in Abbildung 5.8 zeigt die Änderung der Eigenschaften im Netzwerk. Die Paketverzögerung sinkt, das Netzwerk ist hingegen immer noch überladen: Die Puffer B_0 und B_{15} sind dutzendfach voll (Abbildung 5.8 a), b)). Die neue Größe der Puffer B_0...B_{15} nach der Rekonfiguration ist 4, da sie in die Puffer B_{32}...B_{47} aufgespalten wurde. Darüber hinaus ist überaus bemerkenswert, dass Vorbereitungsphase der Rekonfiguration (die er-

sten 50 Takte[2] der Rekonfigurationsphase) keine nennenswerte Auswirkungen auf die Puffer-Warteschlangen und die Paketverzögerungen zeigt.

Die letzte Phase der Simulation (ab 30 000 Takten) zeigt die Überlegenheit der Netzwerktopologie C (Abbildung 5.6) gegenüber den Topologien A und B im Hinblick auf den Hotspot-Verkehr. Die Puffer sind nicht mehr überlastet (Abbildung 5.8 a), b)). Folglich können die Pakete die Ziele schneller, als in den Topologien A und B erreichen, wie in Abbildung 5.8 c) deutlich wird.

[2]hier auf die Abbildung auf Grund der Skalierung nicht zu erkennen, aber im Originaldatensatz abzulesen

6 Rekonfigurierbare Netzwerk-Architektur RecMIN

Im folgenden Kapitel wird die im Rahmen dieser Arbeit entwickelte rekonfigurierbare Netzwerk-Architektur RecMIN vorgestellt.

6.1 Allgemein

In diesem Kapitel wird eine neue Architektur zur Rekonfiguration vorgestellt: RecMIN (Reconfigurable Multistage-Interconnection Network), welche den NoCs eine individuelle, an die Bedürfnisse des Datenverkehrs angepasste Rekonfiguration ermöglicht. Auf diese Weise lässt sich die maximale Leistung des Netzwerks steigern und die Verzögerung des Datenpakets minimieren ohne die Richtlinien des Paket-Routings global ändern zu müssen. Das RecMIN ist baukastenartig strukturiert: Das gesamte NoC besteht aus RecCells und kann problemlos für Netzwerke unterschiedlicher Größen erweitert werden. Die Skalierbarkeit des RecMIN kann daher auf dieselbe Art ausgeführt werden, wie es für das MIN gemacht wird. Des Weiteren ist das RecMIN dynamisch rekonfigurierbar. Dynamische Rekonfiguration von Netzwerken ist durch Hardware-Änderungen des Netzwerks während des Betriebs gekennzeichnet. Das schließt die Architektur des Netzwerks, die Puffer, Standort- und Größenänderungen, sowie Routing ein.

6.2 Rekonfiguration

Das Hauptproblem der NoCs im Vergleich zu der Verbindung aller Ein- und Ausgänge ist die Gefahr von Engpässen bei bestimmten Strukturen des Datenverkehrs. Die Bauweise des RecMINs, die in diesem Kapitel vorgestellt wird, würde das Problem der Engpässe in zwei von drei möglichen Fällen lösen. Der Vorschlag ist, das MIN nicht wie gewöhnlich aus 2×2 Routern herzustellen, sondern aus bestimmten Rekonfigurationshalbzellen – RecHC (Reconfiguration Half Cell). Die Bauweise ist in Abbildung 6.1 aufgeführt.

RecHC hat 8 Eingänge und 8 Ausgänge, vor jedem Ausgang befindet sich ein Puffer. Jede Halbzelle kann in einem von zwei möglichen Modi genutzt werden:

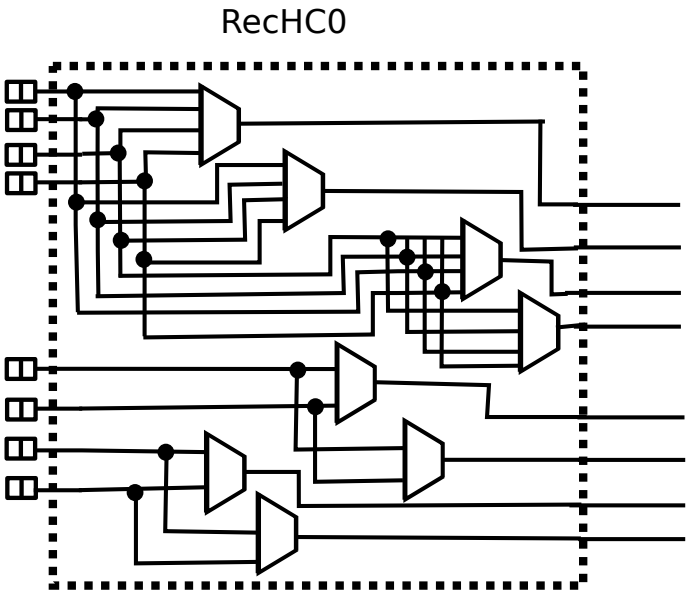

Abbildung 6.1: Die Architektur von Rekonfigurationshalbzelle - RecHC

- im ersten Modus (A) besteht das RecHC aus vier voneinander unabhängigen 2×2 Routern
- im zweiten Modus (B) gibt es hingegen nur einen 4×4 Router im oberen Teil der Zelle und vier einfache Leitungen im unteren Teil

Wenn eine RecHC von Modus A zu B wechselt (Abbildung 6.2), dann werden

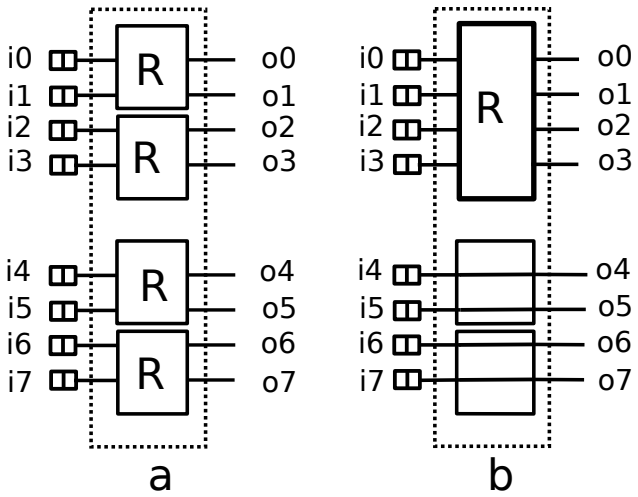

Abbildung 6.2: RecHC in zwei Modi. a: Modus A, b: Modus B

Datenpakete, die im oberen Abschnitt der Halbzelle ankommen, richtig weitergeleitet. Allerdings können im unteren Teil der RecHC Probleme auftreten, da im Modus B keine Umleitung stattfindet und die Datenpakete direkt weitergeleitet werden (Abbildung 6.2). Beispielsweise haben einige Datenpakete, die sich im Puffer von Eingang i4 befinden und an Ausgang o5 adressiert sind, nach dem Wechsel von Modus A zu B keine Möglichkeit, ihr Ziel zu erreichen. Daher ist die Verwendung von zwei Halbzellen zu bevorzugen.

Werden die zwei RecHCs zusammengefügt (die zweite Zelle in der Horizontalen gespiegelt), so formen sie eine Rekonfigurationszelle – RecCell (Abbildung 6.3). Wenn beide RecHCs, aus denen die RecCell gebildet wird, auf Modus A eingestellt werden, führt diese Konstruktion zu zwei unabhängigen MINs (Abbildung 6.3a) mit jeweils 4×4 Eingängen-Ausgängen und je 2×2 Routern. Werden die beiden RecHCs auf Modus B eingestellt, so entstehen zwei unabhängige 4×4 Router (Abbildung 6.3b). Die anderen zwei Kombinationen (AB und BA) sind bedeutungslos und finden daher keine weitere Verwendung. Folglich hat eine vollständige Zelle zwei mögliche Rekonfigurationen: gefaltet (BB) und entfaltet (AA).

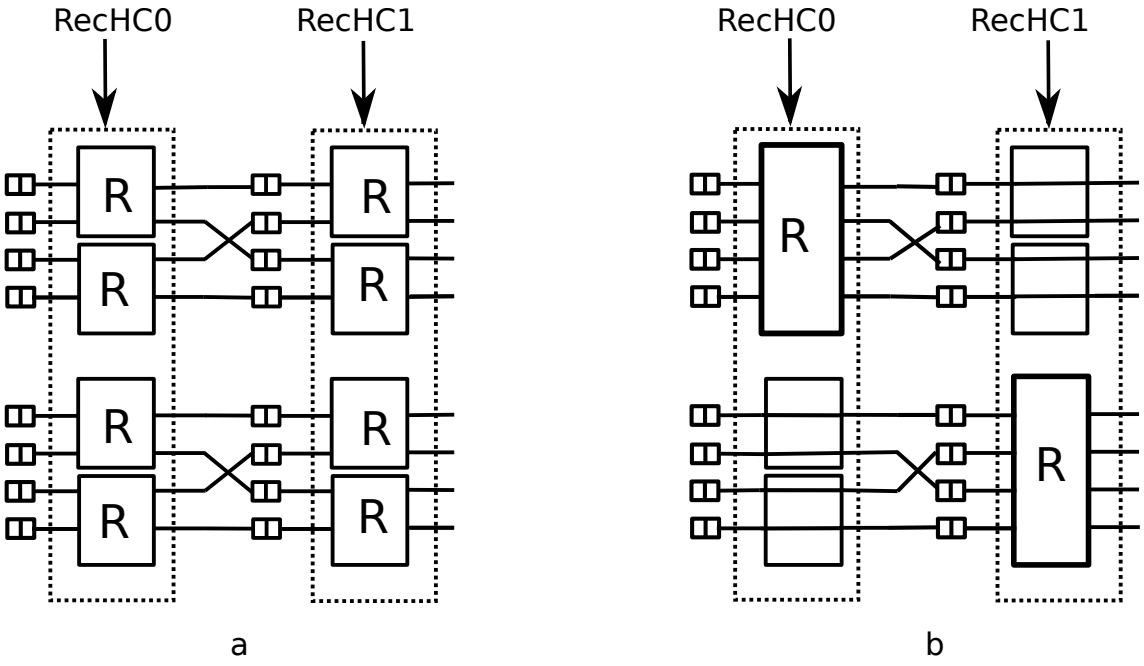

Abbildung 6.3: RecCell in zwei Modi. a: gefaltet (BB), b: entfaltet (AA)

Mithilfe von RecCells lässt sich ein MIN erstellen. Die so entstandene Struktur wird RecMIN genannt. Ist die Anzahl der 2×2 Router durch 16 teilbar, so kann das gesamte Netzwerk aus Rekonfigurationszellen konstruiert werden. Andernfalls ist der Gebrauch von nicht-rekonfigurierbaren Router (z.B 2×2 Router) notwendig

um die Rekonfigurationszellen zu verbinden. Demzufolge kann RecMIN mit einer beliebigen Anzahl an 2×2 Routern versehen werden.

RecMIN mit 16 Ein- und Ausgängen dient im weiteren als Beispiel für die RecMIN-Bauweise (Abbildung 6.4). Das RecMIN kann aus vier RecCells gebaut werden: C0, C1, C2 und C3.

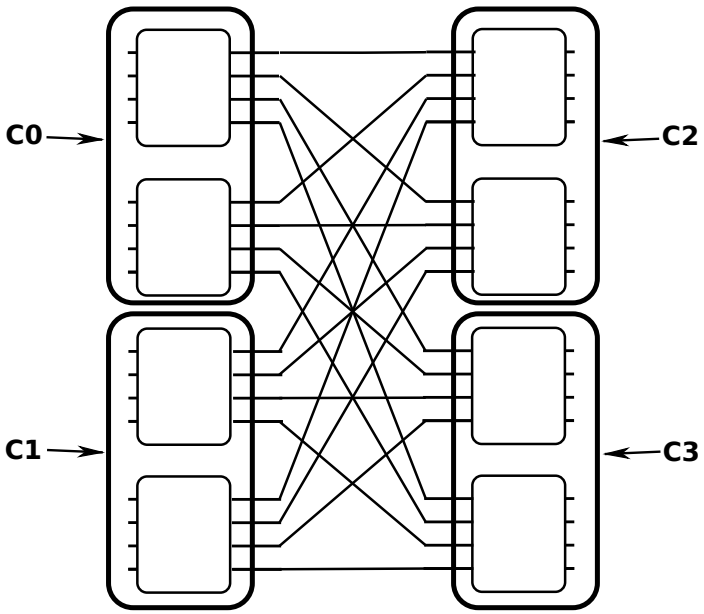

Abbildung 6.4: RecMIN mit 16 Ein- und Ausgänge

Vergleicht man diese Bauweise mit der eines nicht-rekonfigurierbaren MIN mit 2×2 Routern, lässt sich feststellen, dass die mit gestrichelter Linie markierte Router-Verbindung nach Abbildung 6.5 (zwischen dem ersten und dem zweiten Abschnitt des 2×2 Routers) rekonfiguriert werden kann. Wenn also der Datenverkehr an diesen Stellen Engpässe verursacht, kann das NoC seine Struktur der Verkehrsmenge entsprechend rekonfigurieren. Auf diese Weise wird der Datendurchsatz des Netzwerks gesteigert und die Verzögerung der Datenpakete verringert.

Verursacht der Datenverkehr jedoch in einen der nicht-rekonfigurierbaren Leitungen einen Datenstau, so hilft die Rekonfiguration leider nicht. Allerdings kennt der Designer des NoCs den Gebrauch, für den es vorgesehen ist, und ist so normalerweise in der Lage, die für Datenstau anfällige Leitung so in der Rekonfigurationszelle anzuordnen, dass dieser vermieden wird.

Andersherum hätte ein 4×4 Router einen geringeren Durchsatz als ein 2×2 Router [36]. Daher werden bei symmetrischer Überlastung (mehr als 0,63 Flits pro Taktzyklen, sehe [36]) im 2×2 Router automatisch NoC-Datenstaus entstehen. In diesem Fall ist eine Rekonfiguration in den ungefalteten Modus nötig.

Ein weiterer Nachteil dieser Struktur ist, dass nun zwei unabhängige Router des NoCs miteinander verbunden sind. Falls sich einer der 4×4 Router der RecCell "entscheidet", den Modus zu wechseln (von gefaltet zu ungefaltete und andersrum), muss kontrolliert werden, ob der andere Router derselben RecCell dem Modus-Wechsel "zustimmen" wird.

6.3 Ergebnisse

Das RecMIN mit 16 Ein- und Ausgängen (Abbildung 6.5) wurde mithilfe des Simulators

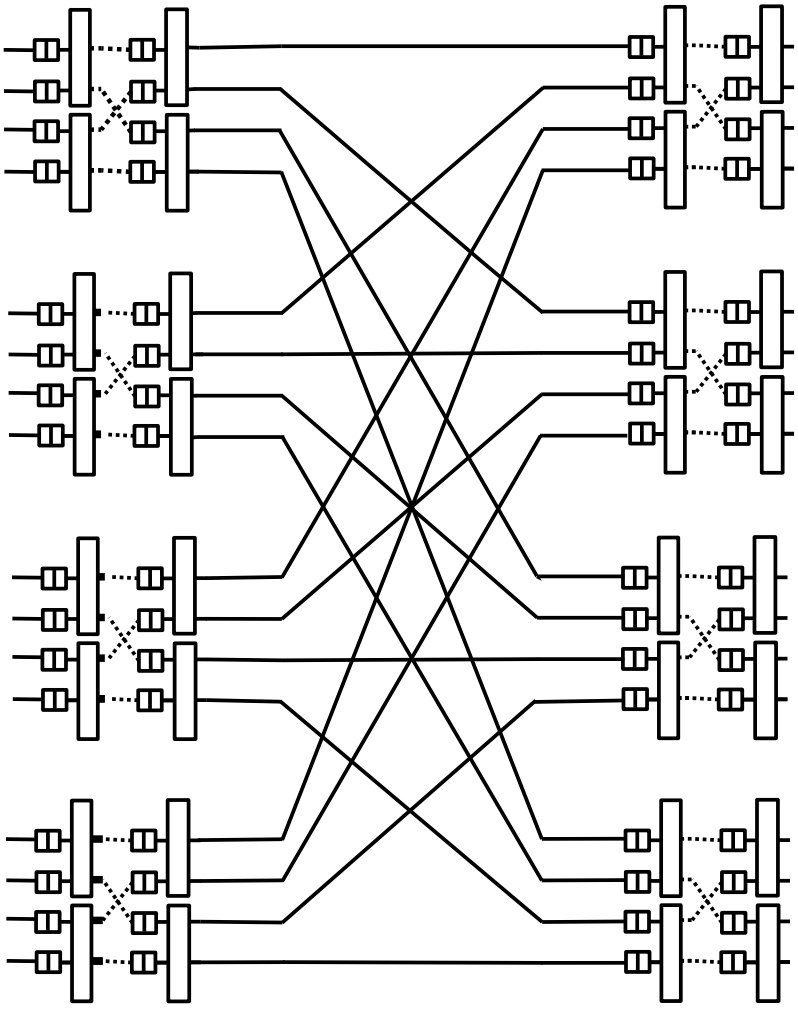

Abbildung 6.5: RecMIN mit 16 Ein- und Ausgänge

RecSim simuliert und seine Effizienz mit zwei unterschiedlichen Verkehrsprofile

getestet (Simulationsparameter: Puffergröße 16 Phits für jeden Puffer, Konfliktlösungsalgorithmus für jeden Router ist Round Robin, jedes Paket besteht aus einem Flit, ein Flit entspricht der Größe eines Phit). Dies verdeutlicht das Potential der Rekonfigurationsstruktur RecMIN, die Effizienz des NoC zu verbessern. Die Verkehrsprofile durch das Netzwerk wurden periodisch gewechselt; Verkehrsprofil 1 (Tabelle 6.1) und Verkehrsprofile 2 (Tabelle 6.2) als Beispiele sich ändernder Verkehrsprofile.

Quelle	$\mathbf{P}_{trans}(\mathbf{Q}_i)$	$\mathbf{P}_{rec}(\mathbf{T}_{10})$	$\mathbf{P}_{rec}(\mathbf{T}_{11})$	$\mathbf{P}_{rec}(\mathbf{T}_{12})$	$\mathbf{P}_{rec}(\mathbf{T}_{rst})$
Q_0, Q_1	0,65	0,65/16	0,65/16	0,65/16	0,65/16
Q_2, Q_3	0,65	0,65/16	0,65/16	0,65/16	0,65/16
Q_4 - Q_7	0,65	0,65/16	0,65/16	0,65/16	0,65/16
Q_8 - Q_{15}	0,65	0,65/16	0,65/16	0,65/16	0,65/16

Tabelle 6.1: Verkehrsprofil 1 (0 - 5 000 und 15 001 - 30 000 Taktzyklen)

Quelle	$\mathbf{P}_{trans}(\mathbf{Q}_i)$	$\mathbf{P}_{rec}(\mathbf{T}_{10})$	$\mathbf{P}_{rec}(\mathbf{T}_{11})$	$\mathbf{P}_{rec}(\mathbf{T}_{12})$	$\mathbf{P}_{rec}(\mathbf{T}_{rst})$
Q_0, Q_1	0,4875	0,2/16	0,3	0,2/16	0,2/16
Q_2, Q_3	0,3875	0,2/16	0,2/16	0,2	0,2/16
Q_4 - Q_7	0,55	0,1	0,2/16	0,2/16	0,2/16
Q_8 - Q_{15}	0,2	0,2/16	0,2/16	0,2/16	0,2/16

Tabelle 6.2: Verkehrsprofil 2 (5 001 - 15 000 Taktzyklen)

In den Tabellen 6.1 und 6.2 ist Q_i die Bezeichnung für jede Quelle i; P_{trans} (Q_i) stellt die Wahrscheinlichkeit, dass die Quelle i ein Paket pro Taktzyklus sendet, dar; P_{rec} (T_x) stellt die Wahrscheinlichkeit, dass der Ausgang x Ziel des Pakets von der Quelle i (Q_i) ist. P_{rec} (T_{rst}) stellt die Wahrscheinlichkeiten P_{rec} (T_x) für alle restlichen Quellen.

Abbildung 6.6 und Abbildung 6.7 zeigen die Auswirkungen der Rekonfiguration, wie die gemessene Paketverzögerung in Ausgang 0 und Ausgang 11 des NoC. Während der ersten 5000 Takte wurde Verkehrsprofil 1 durch das Netzwerk gesendet. Anschließend schaltete der Verkehr auf Verkehrsprofil 2 um. Das führte unverzüglich zu einer Überlastung des Netzwerks, was wiederum in einer viel größeren Verspätung des Datenpakets resultierte. Es wurden weitere 5000 Takte in diesem Zustand ausgeführt, welche zeigten, dass die Überlastung zu einer dauernden Verzögerung führt, bis

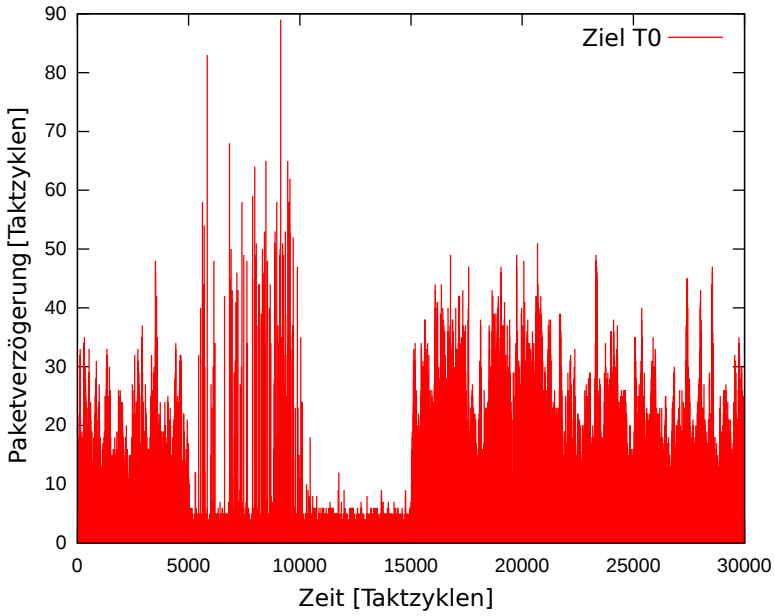

Abbildung 6.6: Die gemessene Paketverzögerung in Ausgang 0

dass das Netzwerk rekonfiguriert wurde. Nachdem dies demonstriert wurde, wurde das Netzwerk rekonfiguriert und damit die Verzögerung der Datenpakete reduziert.

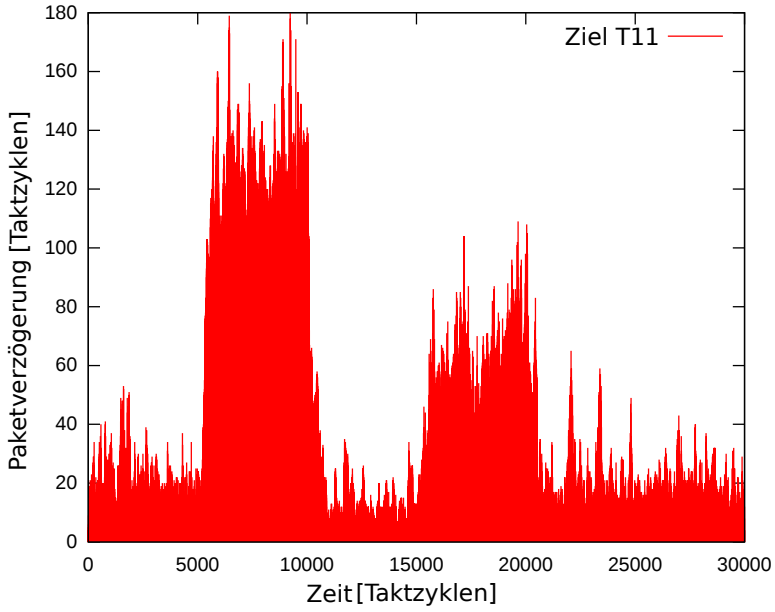

Abbildung 6.7: Die gemessene Paketverzögerung am Ausgang 11

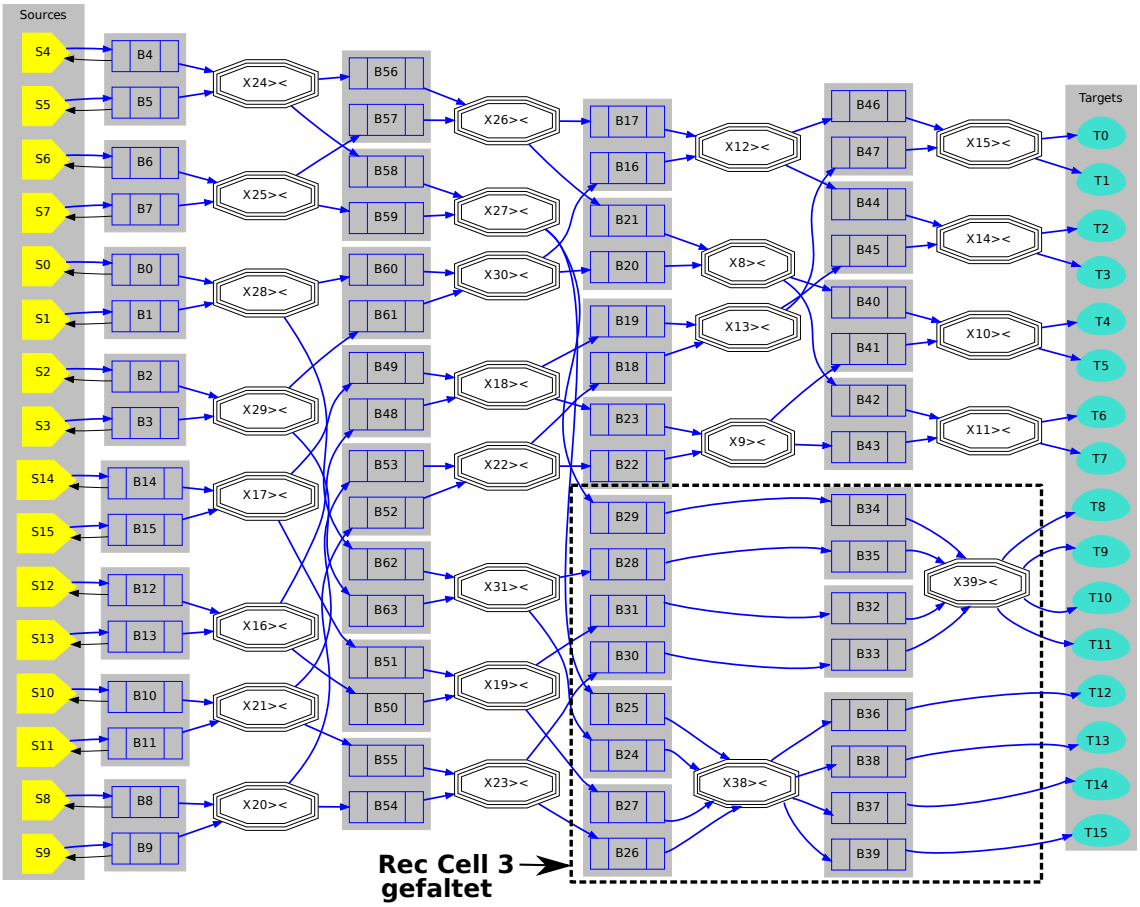

Abbildung 6.8: RecMIN mit 16 Ein- und Ausgängen. Ausgabe des Simulators *RecSim*

Im weiteren Verlauf des Experiments wurden weitere 5000 Takte mit dem effizienten, rekonfigurierten Netzwerk, das Verkehrsprofil 2 verarbeitete, durchgeführt. Abermals ändert sich der Verkehr wieder auf Verkehrsprofil 1. Dies führt nochmals dazu, dass die Verzögerung der Datenpakete beträchtlich anstieg. Nach 5000 Takten wurde das Netzwerk rekonfiguriert und dementsprechend sank die Verzögerung der Datenpakete.

Abbildung 6.8 veranschaulicht die RecMIN-Struktur als graphischen Ausgabe der *RecSim*-Software. Die Simulation zeigt, dass die RecCell 3 im entfalteten Modus operieren muss (da der 2x2 Router einen besseren Datenfluss als der 4x4 Router erlaubt), um Datenstaus für Verkehrsprofil 1 zu vermeiden. Für Verkehrsprofil 2 muss die RecCell 3 im gefalteten Modus operieren (andernfalls lässt der Verkehr einen Datenstau im Zugang zu Puffer B32 entstehen). Die anderen RecCells benötigen keine Rekonfiguration während der Simulation.

Die Ergebnisse der Simulation haben gezeigt, dass die hier vorgeschlagene rekonfigurierbare Netzwerkstruktur das Potential hat, die Leistungsfähigkeit einiger Anwendungen, insbesondere derer, die zwischen zwei oder mehreren Verkehrsprofilen wechseln, drastisch zu verbessern. Außerdem reduziert diese Architektur Paketverzögerungen bedeutend.

7 Untersuchung der Hardwarerealisierung der Pufferkomponenten für RecMIN

Jede NoC-Architektur einschließlich RecMIN kann als eine Menge der Speicherelemente (Puffer), die miteinander mittels der Routerelemente verbunden sind, dargestellt werden. Dieses Kapitel untersucht die mögliche Realisierung der Puffer in der Hardware, für RecMIN basierend auf FPGA, sowie die Einschränkungen, die eine solche Realisierung mitbringt (als Parameter sind Taktrate und die Fläche auf dem FPGA gewählt). Die Ergebnisse dieser Untersuchung sind nicht nur zur Umsetzung der RecMIN von Bedeutung, sondern auch für die Implementierung anderer NoC-Architekturen auf FPGA.

7.1 Einführung

In diesem Kapitel wird beschrieben, wie verschiedene Speicherprinzipien in der rekonfigurierbaren Hardware, in diesem Fall FPGAs, umgesetzt werden. Dabei werden die Prinzipien FIFO, Ringspeicher und LIFO benutzt. Diese wurden dafür in der Hardwarebeschreibungssprache VHDL entworfen und für zwei spezifische FPGA-Familien synthetisiert.

Für die Ausführung der RecMIN ist es erforderlich, hauptsächlich zwei Netzwerk-Komponenten in Hardware zu implementieren: den Puffer für den Zwischenspeicher der Pakete und den Router für das Weiterleiten der Pakete. Dieses Kapitel beschäftigt sich mit der Ausarbeitung des Puffers, wobei die Ausführung des Routers als Teil der weiteren Forschung angesehen wird.

Ziel dieser Ausarbeitung ist der Effizienzvergleich der Methoden FIFO, Ringspeicher und LIFO, indem bei unterschiedlichen Speichergrößen anhand folgender Kriterien untersucht wird:

- Fläche: Da Umsetzung eines System-on-Chips häufig zunächst auf FPGAs geschieht, spielt der durch den Speicherblock verursachte Flächenverbrauch in Form von Grundzellen (Logikzellen), aus denen das FPGA besteht, eine entscheidende Rolle. Vor allem ist beim Einsatz von vielen Komponenten auf dem IC der noch zur Verfügung stehende Platz ausschlaggebend.

- Geschwindigkeit: Die maximale mögliche Taktfrequenz, bei der das Gesamtsystem noch zuverlässig arbeitet, ist ein entscheidender Faktor, der die Verarbeitungsgeschwindigkeit abschätzen lässt.

Die Untersuchung erfolgt dabei hauptsächlich anhand der im FPGA eingebetteten Speicherblöcke, welche neben den eigentlichen Logikzellen von nahezu allen FPGA-Herstellern implementiert werden und gerade für den Zweck der Erzeugung größerer Speicherbereiche zur Verfügung stehen.

7.2 Entwurf der Entity

Alle benutzten Speicherarchitekturen wurden in der Hardwarebeschreibungssprache VHDL entworfen. Dafür war es notwendig zunächst eine sogenannte "Entity" mit einer dazugehörigen Schnittstellenliste (den Ports) zu entwerfen, welche Ein- und Ausgangsleitungen festlegt. Für alle benutzten Speichertechniken wurde der in Abbildung 7.1 gezeigte Aufbau festgelegt, wobei die Eingangssignale auf der linken und die Ausgangssignale auf der rechten Seite zu finden sind.

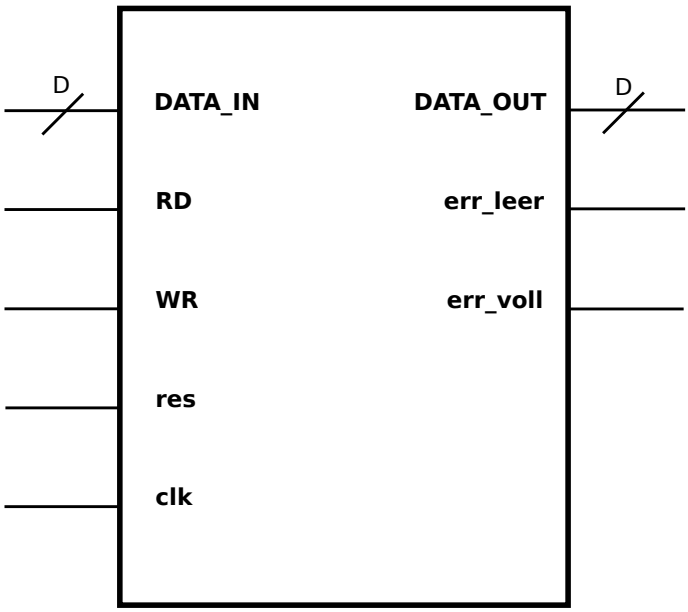

Abbildung 7.1: Beschaltung der Speichermodule

Es wurden zwei Befehlsleitungen für den Lese- und Schreibbefehl `RD` und `WR` benutzt; ein Dateneingang und -ausgang `DATA_IN` und `DATA_OUT`, welche entsprechend der Breite des abzulegenden Datenworts (Wortbreite D) beschaltet sind, sowie zwei Signalleitungen (Fehlerleitungen) `err_leer` und `err_voll` festgelegt, um

anzuzeigen, ob der Speicher voll oder leer ist. Alle Leitungen wurden dabei mit dem im IEEE-Standard 1164 festgelegten Datentyp *STD_LOGIC* bewerkstelligt.

Die eigentliche Verhaltensbeschreibung des Speichers wurde als getakteter Prozess entworfen, die Zustandsänderungen erfolgen bei steigender Taktflanke. Somit ist ein Takteingang `clk` nötig. Außerdem ist ein Reset-Signal `res` implementiert, welches den Speicher auf den Ausgangszustand zurücksetzt. Die Flags `err_leer` und `err_voll` zeigen, ob der Speicher leer oder voll ist. Diese Information ist für andere IPs vorgesehen, die den Speicher benutzen.

Das grundlegende Verhalten wurde dabei wie folgt festgelegt: In jedem Taktzyklus (bei positiver Taktflanke) erfolgt neben einem Abfragen der Reset-Leitung eine Abfrage der Befehlsleitungen (`RD` und `WR`). Je nach anliegendem Befehlssignal wird ein entsprechendes Verhalten ausgelöst. Bei einem Schreibbefehl erfolgt eine Speicherung des an `DATA_IN` anliegenden Datenwortes am nächsten freien Speicherplatz. Ist der Speicher voll, erfolgt keine Speicherung. Bei einem Lesebefehl wird, je nach verwendetem Speicherprinzip, das nächste Datenwort ausgegeben. Ist der Speicher leer, erfolgt keine Ausgabe bzw. `DATA_OUT` bleibt unverändert. Die beiden Fehlerleitungen werden dabei direkt mit dem auslösenden Ereignis gesetzt. Das heißt, wird das letzte Datenwort ausgegeben, wird das `err_leer`-Signal noch im selben Taktzyklus auf 1 gesetzt. Umgekehrt erfolgt bei einer Belegung des letzten freien Speicherplatzes im selben Taktzyklus ein Setzen des err_voll-Signals. Sobald also mindestens ein Speicherplatz frei ist, ist `err_voll` = 0; sobald mindestens ein Speicherplatz belegt ist, ist `err_leer` = 0. (Diese Verhaltensbeschreibung setzt bei allen Speichervarianten eine Tiefe des Speichers von mindestens 2 Speicherplätzen voraus).

Je nach verwendetem Speicherprinzip muss zusätzlich mindestens ein Register für eine Zählvariable definiert sein, die den Füllstand des Speichers anzeigt. Bei jedem Speicher- oder Lesevorgang wird die Zählvariable inkrementiert bzw. dekrementiert. Die Zählvariable kann einen beliebigen Wert zwischen 0 und m (Anzahl der Speicherplätze) annehmen. Für das LIFO- und das einfache (kein Ringspeicher) FIFO-Prinzip reicht die Zählvariable aus, welche auf den nächsten freien Speicherplatz zeigt. Beim FIFO-Ringspeicher muss mit zwei Variablen gearbeitet werden, die das jeweilige als nächst zu lesende oder zu schreibende Datenwort anzeigt. Dementsprechend müssen zwei Register implementiert werden.

7.3 Die Verwendeten FPGA-Architekturen

Die Speicherimplementierung war auf der Basis der Spartan-3E-FPGA-Familie (XC3S500E) von Xilinx und Cyclone II-Familie (EP2C15) von Altera realisiert worden. Die Spartan-Familie kann als Low-Cost-Variante der teureren Virtex-Architektur gesehen werden und stellt somit eine Basis in der Xilinx-Produktpalette

dar.

Der verwendete XC3S500E verfügt über:

- 10476 Logikzellen, aufgeteilt in 1164 Basisblöcke (CLBs) bzw. 4656 Slices
- 232 Ein/Ausgangsleitungen
- 73 KBit Distributed RAM
- 360 KBit Block-RAM
- Herstellungsprozess (CMOS) in 90 nm

sowie über weitere, für diese Untersuchung nicht relevante Merkmale (z.B Multiplizierer, Taktdomänen).

Um eine vergleichbare FPGA-Technologie eines anderen Herstellers zu verwenden wurde die von Altera gefertigte Cyclone II-Serie, konkret in Form eines EP2C15-FPGAs, gewählt. Dieser verfügt über:

- 14448 Logikzellen
- je nach Variante bis zu 315 belegbare Ein/Ausgangsleitungen
- 240 KBit M4K-Speicher
- Herstellungsprozess (CMOS) in 90 nm

Beide Architekturen benutzen Logikzellen in Form einer Lookup-Tabelle mit 4 Eingängen (4-LUT) und verfügen somit prinzipiell über einen vergleichbaren Grundaufbau. Die beiden gehören zu einer Generation der FPGAs, die damit erreichbaren Taktfrequenzen für die Speicher sollten von derselben Größenordnung sein.

Für die Implementierung wurden die ISE Design Suite (Xilinx) und die Quartus II Design Software (Altera) benutzt.

7.4 Organisation der Basiszellen bei EP2C15- und XC3S500E-FPGAs

Der Aufbau und die Organisation der Basiszellen müssen bei den verwendeten FPGA-Familien unterschiedlich sein, damit der Flächenverbrauch nach der Programmsynthese verglichen werden kann. Grundlegend sind beide Familien, wie alle FPGAs, in Logikzellen aufgeteilt. Die Gruppierung in Basisblöcke folgt dabei aber unterschiedlichen Mustern.

Bei Xilinx wird von CLBs (Cell Logic Block) und Slices gesprochen. Ein CLB besteht aus vier Slices; ein Slice besteht aus zwei Logikzellen und somit aus zwei Lookup-Tabellen und zwei Flip-Flops. Jeder CLB verfügt dabei über zwei Typen von Slices: Auf der linken Seite jeweils zwei sogenannten SLICEM-Komponenten, welche neben Logik auch die zusätzlichen Speicherfunktionen ausführen sowie auf der rechten Seite jeweils zwei sogenannten SLICEL-Komponenten, deren Aufgabe es ist, ausschließlich Logik zu realisieren und damit Standard-Logikzellen darzustellen [49]. Einen typischen Aufbau eines CLBs zeigt Abbildung 7.2.

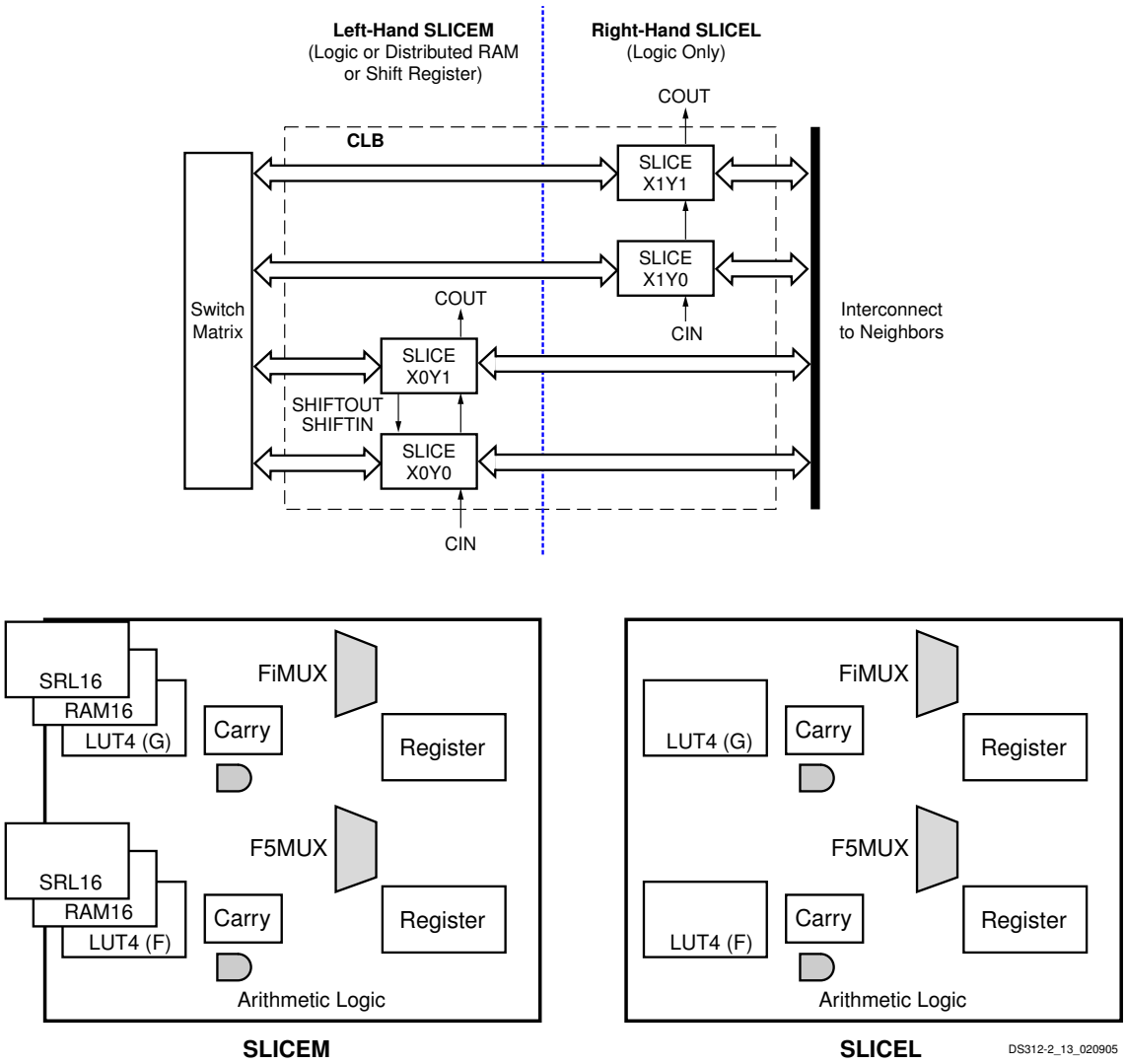

Abbildung 7.2: Aufbau der Xilinx-CLB [49]

Xilinx gibt dabei an, dass ein Slice mit der zusätzliche Funktionalität in Form von

Multiplexer und Übertragfunktionen durchschnittlich den 2,25 Standard-Logikzellen entspricht [49].

Bei Altera erfolgt die Gruppierung in Logic Array Blocks, kurz LAB. Ein LAB besteht aus 16 Logikelementen, welche bis auf einige Zusatzfunktionen (z.B Übertragfunktion) mit den erwähnten Standard-Logikzellen vergleichbar sind.

Entscheidend für die Vergleichbarkeit im Flächenverbrauch ist die Ausgabe der absorbierten, also für die implementierte Schaltung benötigten Logikzellen und Basisblöcke. Bei der Quartus-Software (Altera) wird stets die Anzahl der benötigten Logikzellen angegeben, bei ISE (Xilinx) - die der benötigten Slices. (Zusätzlich haben beide Programme die Ausgabe der benutzten Lookup-Tabellen und Register).

7.5 Eingebettete Speicher

Neben den Registern in den Logikzellen der FPGAs implementieren beide FPGA-Familien Speicher in Form von dedizierten, eingebetteten Speicherblöcken, welche auf dem FPGA gleichmäßig verteilt sind. Bei Xilinx ist dies der sogenannte Block-RAM (BRAM), bei Altera wird beim verwendeten FPGA von M4K-Memory gesprochen.

Block-RAM ist als ein kompakter Speicherblock zu verstehen, der über eine Dual-Port-Fähigkeit verfügt. Im verwendeten FPGA sind 20 Blöcke mit jeweils 18 KBit Speicherkapazität vorhanden [49]. Im Allgemeinen setzen Block-RAMs einen synchronen Betrieb voraus [23]. Der von Altera auf dem ausgewählten FPGA eingesetzte M4K-RAM ist prinzipiell mit BRAM vergleichbar. Allerdings kann M4K-Speicher auch asynchron betrieben werden. (Jeder M4K-Block besteht aus 4 KBit sowie Paritätsinformationen (512 Byte) [3]).

Neben den eingebetteten Speicherblöcken besteht noch eine andere Möglichkeit den Speicher zur Verfügung zu stellen ohne auf jeweils einzelne Register in den Logikzellen zurückzugreifen. Beide gewählte FPGAs basieren auf der SRAM-Technik, d.h. die Lookup-Tabellen sind prinzipiell als Speicher zu betrachten. Allerdings implementiert nur Xilinx diese Speichertechnik in Form sogenannter Distributed RAM. Bei den ausgewählten sowie den meisten anderen Xilinx-FPGAs ist es deshalb möglich die LUT in den SLICEM-Komponenten (jede zweite Logikzelle als einen 16x1 Bit breiten Speicher) aufzufassen. Im Gegensatz zu BRAM lässt sich Distributed RAM auch asynchron auslesen. Ist die Leseadresse getaktet, kann sowohl BRAM als auch Distributed RAM erzeugt werden. Wird die Leseadresse hingegen asynchron angelegt, so muss Distributed RAM verwendet werden [49].

Eine weitere Möglichkeit ist die Anwendung der Lookup-Tabellen als Schieberegister. Diese Möglichkeit besteht sowohl bei Xilinx als auch bei Altera. Für eine konkrete Pufferspeicherumsetzung kommt aber nur Xilinx infrage, da es nur hier

möglich ist mit einem Offset zu arbeiten[1]. Bei Altera kann das Ausgeben der Werte nur am Ende der Schieberichtung erfolgen [3]. So müsste also bei der Umsetzung des FIFO-Prinzips der Speicher zunächst vollständig gefüllt werden, bevor er wieder ausgelesen werden kann. Eine Umsetzung ist somit nicht praktikabel.

7.6 Ergebnisgewinnung

Die Anzahl der abzulegenden Speicherzellen (einer Speicherzelle entspricht einem Datenwort) wird schrittweise von 2 auf 50 Zellen gesteigert. Die Breite des Datenwortes einer einzigen Speicherzelle wird schrittweise von 2 auf 64 Bit erhöht.

Ein Grundproblem bei der Simulation von größeren Bitbreiten auf FPGAs ist das automatische Platzieren der synthesierten Schaltung, da jedem Bit ein Ein- / Ausgangsport (bzw. Ein- / Ausgangspin) zugewiesen wird. Somit werden beispielsweise bei 64-Bit Datenwortbreite schon 128 Leitungen bzw. Pins benötigt[2]. Die verwendeten FPGAs begrenzen somit die verfügbare Bitbreite, die erwähnten 64 Bit waren dabei das Maximum, welches bei beiden Chips noch funktionierte[3].

Bei beiden Softwareumgebungen und den externen Syntheseprogrammen wurde, wenn nicht anders erwähnt, mit den Standardoptionen gearbeitet. Um die verschiedenen Speichertypen im Xilinx-FPGA zu erzwingen wurde die Syntheseoption *RAMStyle* in den *HDL Options* der Optionen des XST-Syntheseprogramms benutzt. Dort kann ein Speichertyp forciert werden.

Die Angabe der Fläche in den Ergebnissen erfolgt anhand der jeweiligen Basiszellen. Bei Xilinx sind dies Slices und bei Altera Logic Elements, welche mit Standard-Logikzellen prinzipiell vergleichbar sind.

Die Angabe der maximalen Taktfrequenz ergibt sich bei der Xilinx-XST-Synthese aus dem Synthesis-Report. Bei Altera wird diese mit F_{max} angegebene Zeit durch den standardmäßig nicht aktivierten Early Timing Estimate durchgeführt.

Allgemein handelt es sich sowohl bei Xilinx als auch bei Altera bei der Synthese und beim Place-and-Route-Vorgang um ein heuristisches Verfahren. Somit zeigen die Ergebnisse eine gewisse zufällige Komponente.

7.7 Leistungsaufnahme

Die vom SoC verbrauchte Leistung besteht aus: Statischer und dynamischer. Statische Verlustleistung hängt von der Größe der angelegten Betriebsspannung, der

[1] es ist also möglich, ein konkrets Register innerhalb der Schieberegister-Kette auszugeben [49] .

[2] zuzüglich der restlichen Signal- und Befehlsleitungen.

[3] Prinzipiell wäre eine Logiksynthese möglich, bei der mehr Leitungen synthetisiert werden als vorhanden sind. Jedoch verweigerten beide Entwicklungsumgebungen und die eingebundene Logiksynthese-Software diese Lösung.

Ausgangskapazität der Schaltung und der Umgebungstemperatur ab. Dynamische Verlustleistung verursacht das Taktnetz sowie die Schaltprozesse im Speicher selbst.

Letztere ist die entscheidende Größe, die eine Aussage über das Verhalten der Schaltung bei steigender Speichergröße – und somit Komplexität – möglich macht. Die Abschätzung ist dabei generell mit einer hohen Ungenauigkeit verbunden[4].

Der XilinxXPower-Analyser und der Altera PowerPlay Power-Analyser sind in den jeweiligen Entwicklungsumgebungen für eine Leistungsabschätzung zuständig. Das prinzipielle Vorgehen sieht vor, zunächst allgemeine Parameter wie Umgebungstemperatur, Betriebsspannung und vor allem den Takt festzulegen und anschließend anhand der generierten Netzliste nach der Logiksynthese eine Abschätzung der Schaltvorgänge der einzelnen Elemente zu machen. Dafür ist es unumgänglich, die Schalthäufigkeiten (engl. activity rates oder toggle rates) festzulegen. Dies kann zunächst allgemein geschehen, indem festgelegt wird, mit welcher Rate, z.B die Flip-Flops oder die Ein- und Ausgangsleitungen durchschnittlich schalten.

Die Ergebnisse sind dabei laut Aussage der Hersteller umso zuverlässiger, je genauer die vorgegebene Simulation des Schaltvorgangs das tatsächliche Verhalten der Schaltung widerspiegelt. Das konkrete Vorgehen zur Simulation sah vor, ein möglichst identisches Verhalten bei unterschiedlichen Speichergrößen vorzugeben. Bei Altera bestand die Simulation des Schaltvorgangs aus:

- einstellen der Taktfrequenz auf 50 MHz (20 ns)
- simulationszeit 5 µs (250 Taktzyklen)
- Den Speicher zunächst zu 50% füllen und anschließend bis zum Simulationsende gleichzeitig zu beschreiben und zu lesen

Die eingehenden Datenwörter wurden dabei zufällig generiert, sodass in jedem Takt ein zufälliges Datenwort am Eingangsport anliegt. Bei Xilinx war das Vorgehen ähnlich, allerdings musste hier die Taktfrequenz auf 12,5 MHz (80 ns) festgelegt werden, da höhere Frequenzen fehlerhafte Simulationen erzeugten und die maximale Simulationszeit auf 12 µs beschränkt wurde (150 Taktzyklen). Somit ist durch diese Simulation eine Aussage über die dynamische Verlustleistung möglich, welche je nach Hersteller eine mehr oder weniger genaue Abbildung der tatsächlichen Leistungsaufnahme darstellt. Da versucht wurde, die Bedingungen in der Simulation möglichst identisch zu lassen und weil davon ausgegangen werden kann, dass sich die Berechnungsmethoden in den Programmen von ändernder Speichergröße nicht unterscheiden, ist somit zumindest eine Aussage über die Tendenz möglich. Die in den Ergebnissen dargestellten Leistungsaufnahmen präsentieren ausschließlich den dynamischen Teil der Gesamt-Leistungsaufnahme.

[4]Es ist nicht bekannt, wie oft ein bestimmter Puffer gelesen und geschrieben wird. Dies hängt stark sowohl von RecMIN als auch von dem Paketverkehr im Netz ab.

7.8 Speichererkennung

Als grundlegendes Problem bei der Synthese der Speicherprinzipien haben sich die Erkennung (engl. inference) der Speicherblöcke und die damit verbundene Verwendung der eingebetteten Speicher herausgestellt. Das Indizieren eines Speicherbereichs erfolgt in VHDL in der Form $mem(pt_w)$, wobei pt_w in der benutzten Implementierung den Speicherpointer, also die Zählvariable darstellt. Aufgrund dessen ergab sich das Problem, dass das Hinzufügen eines Offsets, also einer konstanten Zahl zum Index, wie z.B $mem(pt_w - 1)$, eine Erkennung oft nicht mehr möglich mht. Beim Implementieren des LIFO-Speichers ist es somit problematisch, nur eine Zählvariable (Register) zu benutzen[5]. Um dieses Problem zu umgehen, wurde eine Variante mit zwei Zählvariablen implementiert, bei der die erste immer noch auf den nächsten freien Speicherbereich zeigt, die zweite aber automatisch immer um eins dekrementiert ist, um das Offset-Problem zu umgehen. Folglich kann hier, wie auch beim Ringspeicher, auch von einem Lese- und Scheibpointer (Zeiger) gesprochen werden, da der nun hinzugekommene Zeiger ausschließlich für die Leseoperation benutzt wird. Außerdem war die Erkennung des LIFO-Speichers als Dual-Port-Speicher nicht möglich (s. Kapitel 7.10.1).

7.9 IP-Core

IP-Cores (intellectual property core) bezeichnen zunächst grundlegend einen wiederverwendbaren Teil eines Chipdesigns und somit einen fertigen Baustein, welcher in ein eigenes, möglicherweise schon bestehendes Design eingefügt werden kann. Der Umfang kann dabei von einfachster arithmetischer Funktion bis hin zu komplexen Mikroprozessor-Kernen reichen. In der Halbleiterindustrie bezeichnet er vor allem das geistige Eigentum des Schaltungs-Entwicklers, welches z.B durch Lizensierung genutzt werden kann. Allerdings wird der Begriff auch für Komponente benutzt, die als fertiges Modul in rekonfigurierbarer Hardware, vor allem FPGAs, eingebunden werden kann.

Die eingesetzten Software-Umgebungen von Altera und Xilinx, aber auch die anderer FPGA-Hersteller bieten zu diesem Zweck sogenannte Komponentengeneratoren an, welche vor allem häufig benutzte Komponenten in optimierter, an die Zielhardware angepasster Form generieren können. Die damit erzeugten Komponenten werden prinzipiell als Makros bezeichnet [49], bei Altera werden diese auch Megafunctions genannt. Diese Komponenten sind dann als Black-Box generiert, d.h., sie werden nur über ihre Schnittstellen und ihr Verhalten definiert [3]. Über den tat-

[5] Beim Speicher ist der Offset unnötig, da pt_w immer auf den nächsten freien Speicherblock zeigt. Im Gegensatz dazu muss beim Auslesen mit dem Offset gearbeitet werden, da sich der zuletzt belegte Speicherplatz bei $pt_w - 1$ befindet.

sächlichen Aufbau erfährt der Anwender zunächst nichts. Die generierbaren Komponenten können dabei meist mit den verschiedensten optionalen Eigenschaften und Parametern versehen werden, wie z.B zusätzlichen Statussignalen oder speziellem Verhalten im Fehlerfall. Diese Makros sind damit schon vorsynthetisiert und müssen nur noch mit den Ein- und Ausgangsleitungen bzw. der Peripherie oder anderen Modulen verbunden werden.

Unter verschiedenen kostenlosen IP-Blöcken, die für FPGAs synthetisiert werden können, findet sich in beiden Umgebungen (Altera und Xilinx) auch der FIFO-Speicher. Also lassen sich Puffer als ein FIFO-Modul generieren. Es kann dabei entschieden werden, ob der für das Modul benötigte Speicher in Distributed-RAM oder in Block-RAM generiert werden soll. Die minimal einstellbare Anzahl der Speicherplätze beträgt dabei 16. Mithilfe des benutzen Speichers kann somit ein Vergleich mit dem Ringspeicher-Prinzip gemacht werden. Es wird daher anhand einiger wenigen Speichergrößen untersucht, in welchem Maße sich der generierte vom selbst erzeugten Speicher unterscheidet.

7.10 Vergleiche und Ergebnisse

7.10.1 LIFO Speicher

Der größte Nachteil des LIFO-Speichers in RecMIN besteht darin, dass die Flit-Reihenfolge während der Übertragung zerstört wird. Allerdings kann es durchaus sein, dass die Nutzung des LIFO-Speichers andere Vorzüge mit sich bringt, wie z.B höhere Geschwindigkeit oder geringeren Flächenverbrauch.

Der Pseudocode für den implementierten LIFO steht im Anhang.

Die Tatsache, dass die automatische Erkennung des Speicher-Makros durch die Synthesesoftware oft fehlgeschlagen hat, stellt ein erhebliches Problem des LIFO-Speichers dar. So, wie oben beschrieben (s. Kapitel 7.8), musste der LIFO-Speicher mit zwei Zählervariablen implementiert werden. Außerdem war das Indizieren eines Dual-Port-fähiges LIFO-Speicherbereichs mit keiner der verwendeten Software möglich, aus folgendem Grund: Beim Dual-Port-Zugriff muss immer aus demselben Speicherplatz in einem Takt gleichzeitig gelesen und geschrieben werden, was offensichtlich die Erkennung des Speicher-Makros verhindert. Die Ergebnisse für den LIFO (ohne Dual-Port) sind in den Abbildungen 7.3, 7.4[6], 7.5[7] präsentiert.

Allgemein kann man sagen, dass der LIFO-Speicher wegen der Vertauschung der Flit-Reihenfolge eine ungeeignete Variante für die Puffer in RecMIN ist. Offensichtlich ist LIFO schlecht indizierbar, besonders im Falle einer Dual-Port-

[6] bei der Verzögerung handelt es sich um "minimum period"

[7] Der Unterschied ist dadurch entstanden, weil Altera und Xilinx zwei unterschidlichen Algorithmen für Energieverbrauch-Schätzung verwenden.

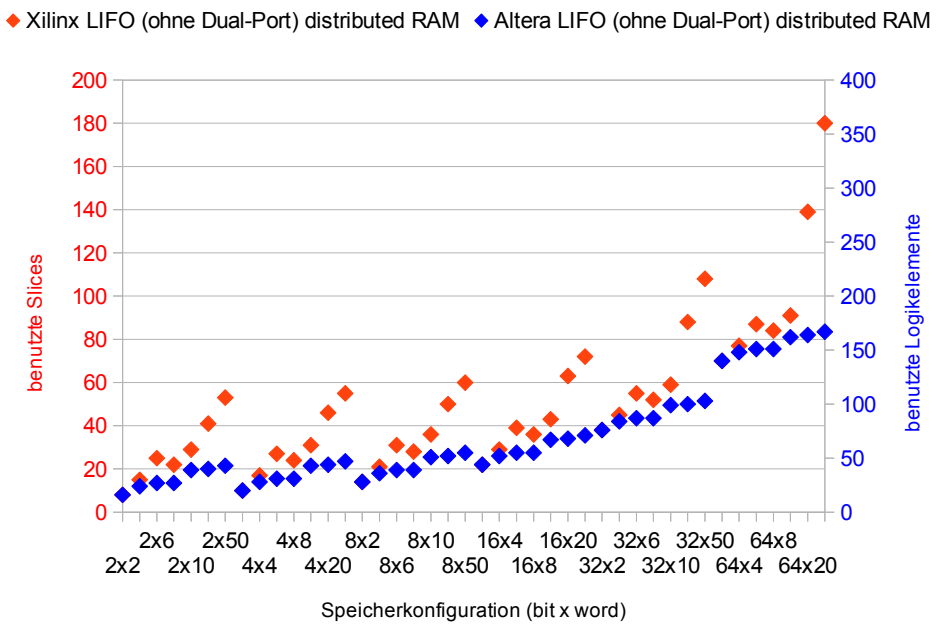

Abbildung 7.3: LIFO. Vergleich nach Flächenverbrauch

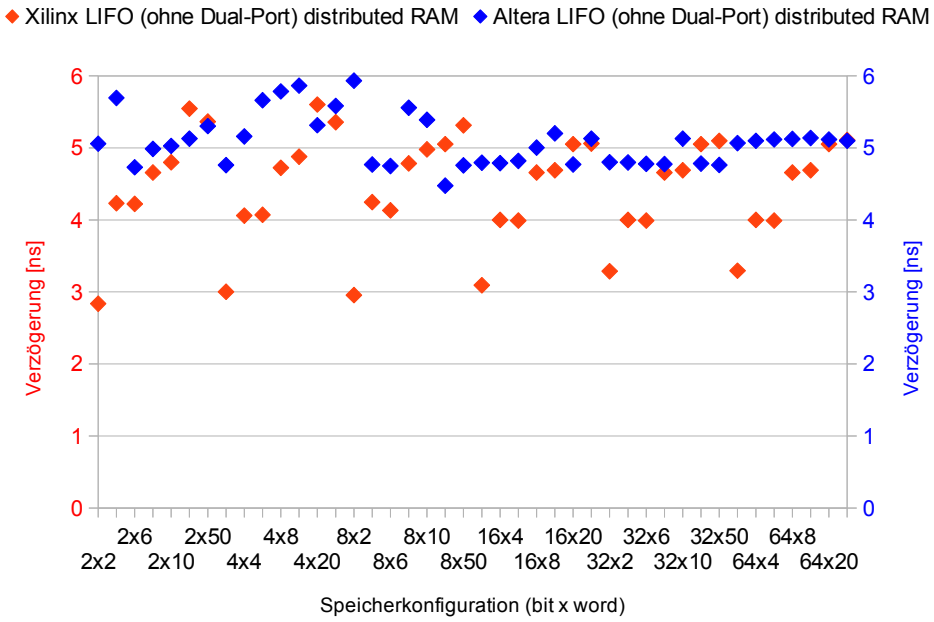

Abbildung 7.4: LIFO. Vergleich nach Verzögerung

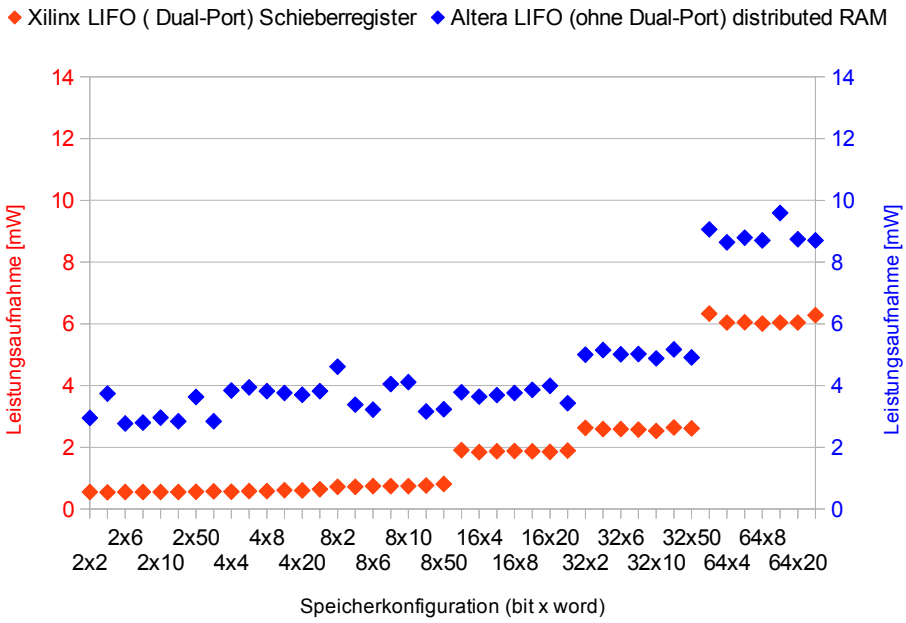

Abbildung 7.5: LIFO. Vergleich nach Energieverbrauch

Zugriff-Implementierung. Der Vergleich mit den anderen Speicherstrukturen (FIFO /Ringspeicher) zeigt eindeutig, dass LIFO keineswegs höhere Geschwindigkeit oder geringeren Flächenverbrauch denen gegenüber aufweist (s. Kapitel 7.10.2, 7.10.3). So wird die Tatsache, dass LIFO die Flit-Reihenfolge während der Übertragung zerstört nicht mit anderen positiven Effekten kompensiert. Folglich wird LIFO als nicht relevante Lösung für RecMIN erkannt und kommt somit bei den Speicher-Vergleichen nicht in Betracht.

7.10.2 FIFO Speicher

FIFO-Speicher mit Dual-Port-Betrieb kann nur in Xilix FPGA- Architektur implementiert werden, da nur bei Xilinx FPGA die Benutzung von Lookup-Tabellen als dynamische Schieberegister-Blöcke möglich ist[8]. Das Benutzen der dynamischen Schieberegister-Blöcke bringen noch weitere Pluspunkte mit sich : Da die SLICEM Cells über die ganze FPGA-Fläche gleichmäßig verteilt sind, wird die Anwendung dieser Cells als Puffer für RecMIN zu Verkürzungen der Verbindungsleitungen zwischen Puffer und Router führen. Außerdem bleibt beim Einsetzen des FIFO-Speichers die Reihenfolge der Flits im Paket vorhanden, was einen weiteren Vorteil für IP-Cores darstellt.

[8] in SLICEM Cells

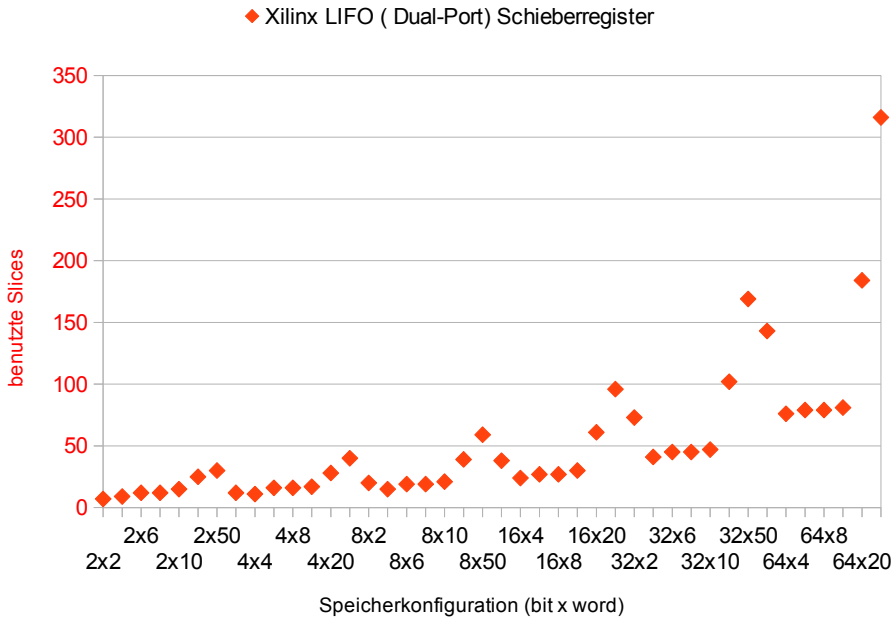

Abbildung 7.6: FIFO. Vergleich nach Flächenverbrauch

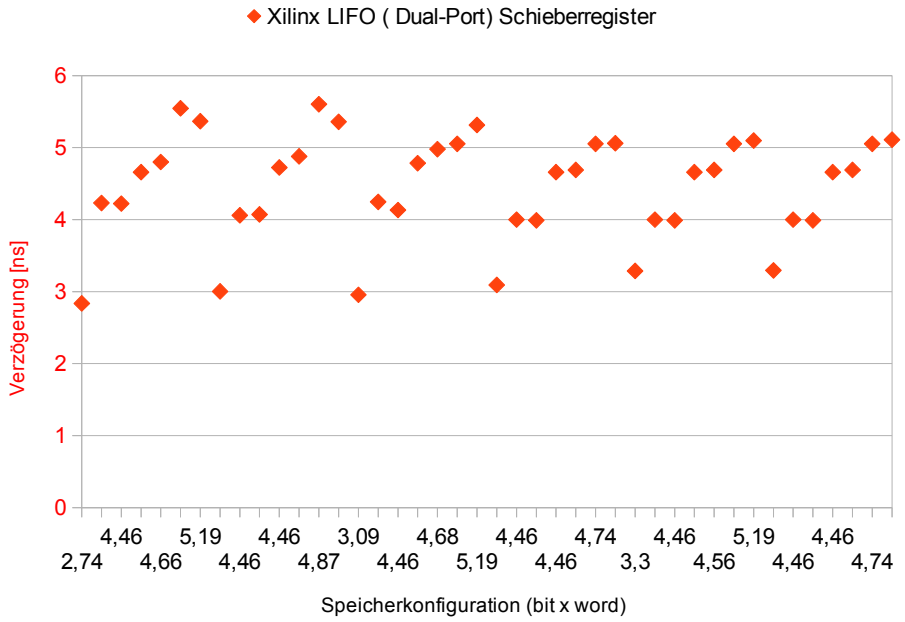

Abbildung 7.7: FIFO. Vergleich nach Verzögerung

Der Pseudocode für den implementierten FIFO steht im Anhang.
Die Ergebnisse der Untersuchung der Anwendung des FIFO-Speichers auf Xilinx FPGA sind in den Abbildungen 7.6 - 7.8 vorgestellt. Es ist von Vorteil die FIFO-Strukturen auf dem Xilinx-FPGA zu realisieren, da nur auf der Basis dieser Technologie die Möglichkeit gegeben, ist die Lookup-Tabellen der SLICEM-Zellen als Schieberegister zu nutzen.

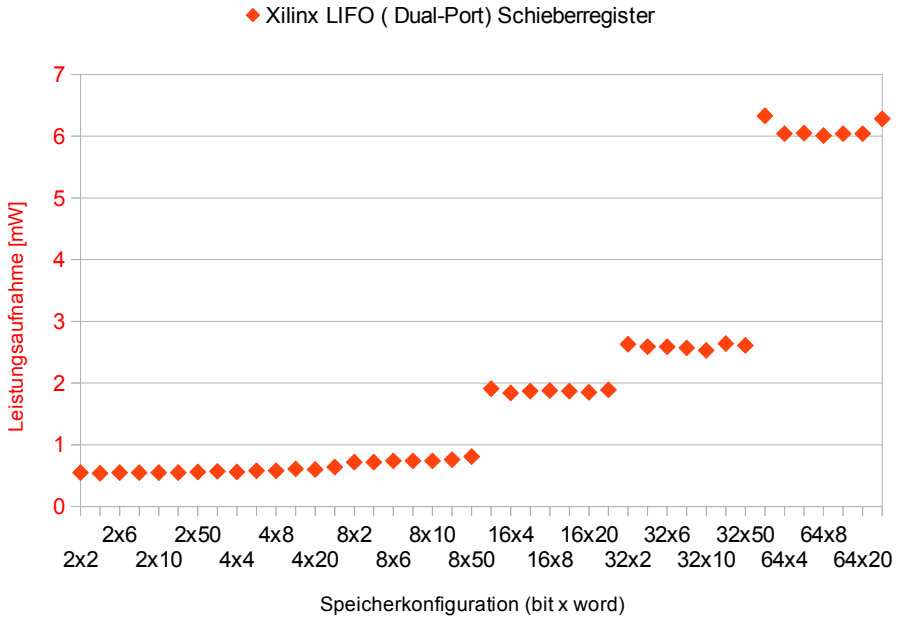

Abbildung 7.8: FIFO. Vergleich nach Energieverbrauch

7.10.3 Ringspeicher

Wird das RecMIN mit einem dichten Paketverkehr belastet, so kann die Verwendung von einfachen FIFO-Speichern zur ineffizienten Energienutzung führen. Die Ursache dafür ist das Schieben von allen Speicher-Registern, das nach jedem Auslesen des FIFO-Speichers ausgeführt werden muss (s. Algorithmus in Kapitel 7.10.2). Mit dem Einsetzen der Ringspeicher kann das Problem des ständigen Register-Schiebens gelöst werden. Allerdings muss man bei jedem Ringspeicher mit zwei Zeiger, Lese- und Schreibepointer arbeiten (für einfaches FIFO reicht ein Schreibepointer vollkommen aus). Für eine zusätzliche Pointer-Variable wird auch ein zusätzliches Speicher-Register von zwei oder mehr Byte (abhängig von der Speichergröße) erforderlich.

Der Pseudocode für den implementierten Ringspeicher steht im Anhang.

Die Ergebnisse für Ringspeicher sind auf den Abbildungen 7.9 - 7.11 zu sehen.

Da bei den Ringspeichern kein Register-Schieben nötig ist, kann man sowohl bei Altera als auch bei Xilinx einige LUTs als Speicher-Register verwenden. Wie man

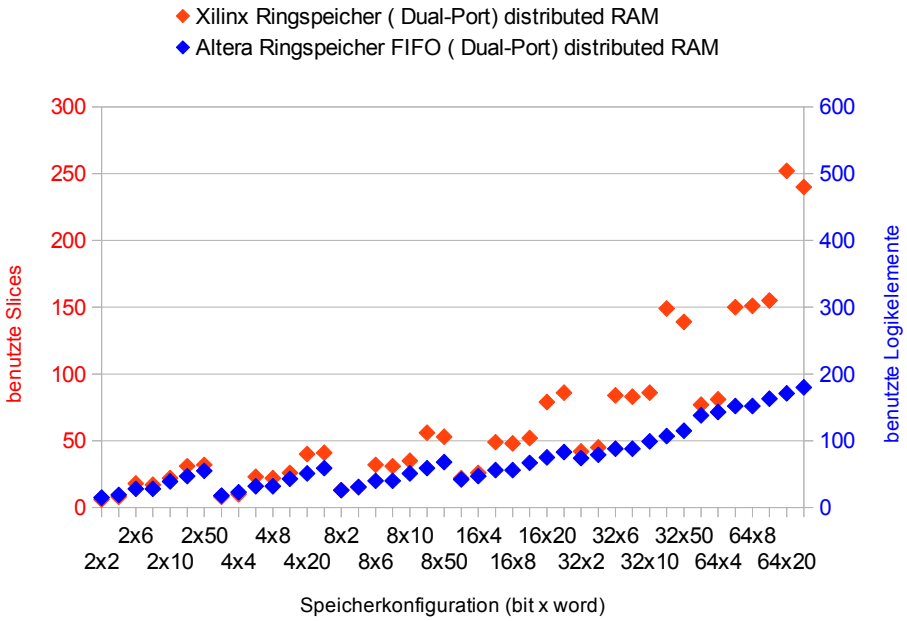

Abbildung 7.9: Ringspeicher. Vergleich nach Flächenverbrauch

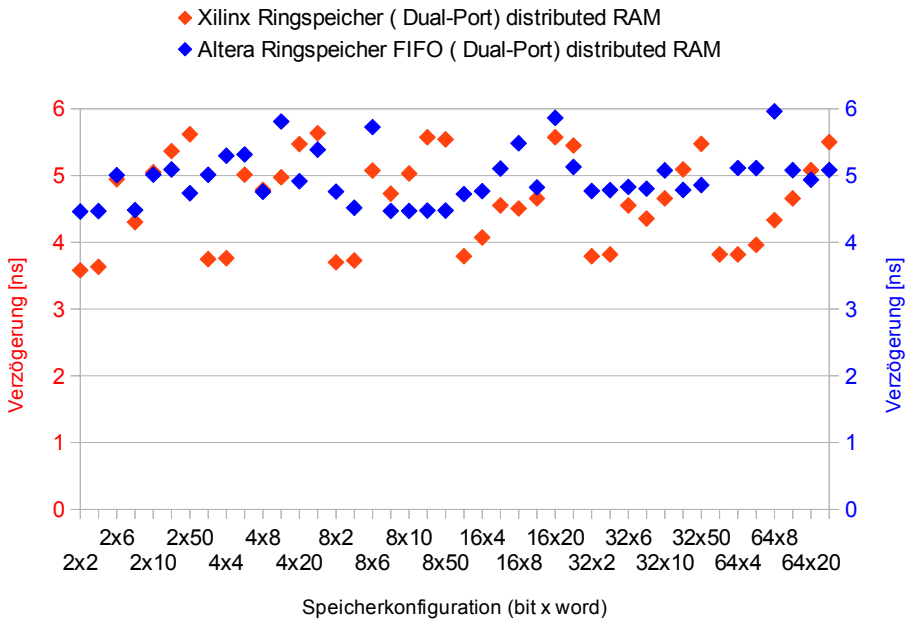

Abbildung 7.10: Ringspeicher. Vergleich nach Verzögerung

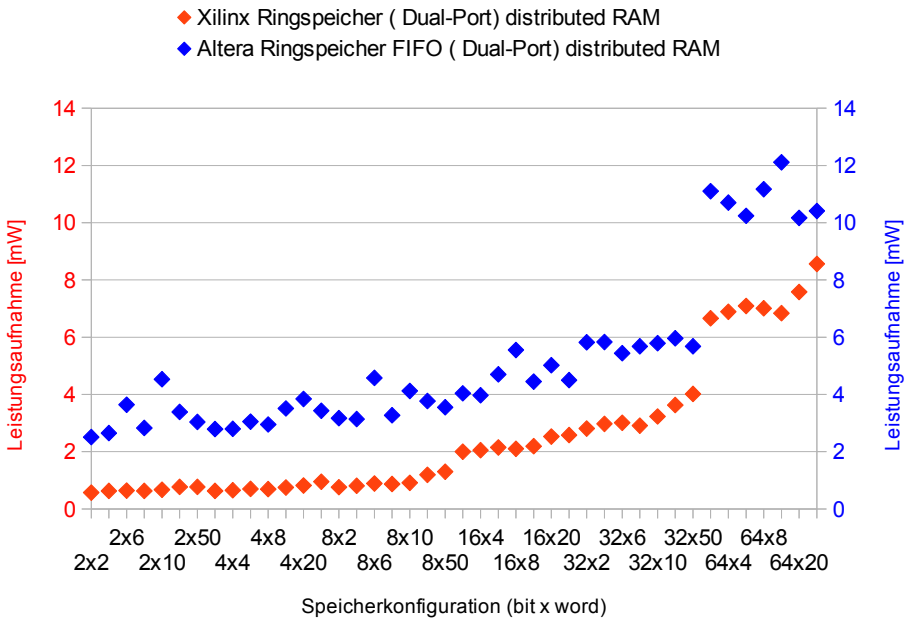

Abbildung 7.11: Ringspeicher. Vergleich nach Energieverbrauch

aus der Grafiken erkennen kann, lässt es sich nicht eindeutig sagen, welche FPGA-Architektur - Altera oder Xilinx - für die RecMIN-Realisierung besser geeignet ist: z.B falls die Verzögerung entscheidend ist, so scheint für die Puffer mit kleiner Wortlänge und mit Puffer-Tiefe kleiner als 32 die Xilinx FPGA-Variante vorteilhaft (bei den Puffern mit Tiefe 8 und der Wortlänge 2 und 4 Bits ist die Verzögerung bei Xilinx-FPGA 3,71 ns und 3,72 ns, wobei sie bei Altera 4,7 und 4,8 ns beträgt, s. Abbildung 7.10), für die Speicher mit den längeren Worten ist es aber umgekehrt: Die Nutzung der Altera zeigt eine bessere Verzögerung (bei den Puffern mit Tiefe 8 und der Wortlänge 50 Bits ist die Verzögerung bei Xilinx-FPGA 5,5 ns und 3,72 ns, bei Altera hingegen beträgt sie 4,4 ns, s. Abbildund 7.10). Ist aber Energieverbrauch entscheidend, so bis zu Puffer-Tiefe 64 ist die Altera-FPGA besser geeignet.

7.10.4 Fazit

Am Ende dieser Untersuchung kann man ein Xilinx-FPGA mit FIFO-Speicherprinzip (beim einfachen FIFO für die RecMINs mit wenig Paketverkehr) als eine der besten Lösungen für die RecMIN-Realisierung empfehlen. Wird aber für RecMIN ein FIFO mit Ringspeicher gewählt, was vor allem für einen starken Paketverkehr sinnvoll ist, so wäre die Wahl zwischen Altera- und Xilinx-FPGAs für jeden konkreten Fall mit Hilfe der Daten aus die Abbildungen 7.9 - 7.11 individual

zu treffen.

Allerdings muss man für die spätere RecMIN-Realisierung anmerken, dass der Unterschied bei den möglichen Speicher-Realisierungen für RecMIN in Altera- und Xilinx-FPGA meistens im Bereich der 10-20% liegt. Dieser Unterschied ist unter Umständen nicht ausschlaggebend, da es bei dieser Speicheruntersuchung um ein Vormodellieren und eine vorläufige Untersuchung handelt (so wird hier z.B kein FPGA-Platzierungsvorgang (Place-and-Route) hineingezogen). Werden die Speicher-Komponenten erst beim vollständig implementierten RecMIN auf dem Chip platziert, so kann hier mit gewisser Wahrscheinlichkeit behauptet werden, dass der Unterschied von 10-20% zwischen den verschiedenen Puffer-Realisierungen nicht von Bedeutung ist.

8 Rekonfigurationsalgorithmen für die RecMIN-Architektur

Damit die im Kapitel 6 vorgestellte RecMIN-Architektur sich erfolgreich an die Netzwerkbelastung anpassen kann, müssen Rekonfigurationsalgorithmen implementiert werden. In diesem Kapitel präsentieren wir drei solche Algorithmen: den η-Algorithmus, den Minimal-Queues-Algorithmus und den Pattern-Identification-Algorithmus. Die Algorithmen werden miteinander verglichen, ihre Effizienz wird erörtert.

8.1 η-Funktion

Die Effizienz des Netzes unter einer asymmetrischen Belastung wird mithilfe der drei Hauptparameter charakterisiert: Der Durchsatz der Netzwerkein- und -ausgänge sowie der Verzögerung. Jeder von diesen Leistungsparametern wird separat für jeden Ein- und Ausgang gemessen. Damit nur ein einziger Wert für die Effizienz des Netzes verwendet werden kann, wird in dieser Arbeit eine Effizienz-Funktion η eingeführt:

$$\eta = \sum_{i=0}^{N-1} (\varsigma_i * C_{\varsigma_i} + \tau_i * C_{\tau_i} + \delta_i * C_{\delta_i}) \tag{8.1}$$

Wobei N der Anzahl von Ein- und Ausgängen im NoC entspricht und die Konstanten C_{ς_i}, C_{τ_i}, C_{δ_i} die Gewichtungen der Prioritäten für Belastung, Durchsatz und Verzögerung sind. Beim Setzen dieser Koeffizienten spezifiziert der Netzwerkentwickler, wie wichtig die entsprechenden Leistundsparameter für das jeweilige Vorhaben sind. Jede der Gewichtungen der Prioritäten C_{ς_i}, C_{τ_i}, C_{δ_i} kann auch negativ sein, algemein soll gelten: Je leistungsstärker das Netz, desto höhe der Wert für η.

Beispielsweise kann für ein bestimmtes Netzwerk der Durchsatz der Quellen und der Ziele nicht von großer Bedeutung sein. Im Gegensatz dazu ist es im Beispiel, durchaus wichtig, dass die Verzögerung so niedrig wie möglich gehalten wird. Besonders wichtig soll außerdem die Verzögerung an den Zielen T_4 und T_5 sein. In diesem Fall können die Konstanten C_{ς_i}, C_{τ_i}, C_{δ_i} wie gefolgt definiert werden:

$$
\begin{aligned}
C_{\varsigma_i} = C_{\tau_i} = 0 \quad &[\text{Taktzyklen/Flit}] \quad \text{für } i \in \{0, .., N-1\} \\
C_{\delta_i} = -1 \quad &[\text{Taktzyklen}]^{-1} \quad \text{für } i \in \{0, .., N-1\}/\{4, 5\} \\
C_{\delta_4} = C_{\delta_5} = -2 \quad &[\text{Taktzyklen}]^{-1} \quad \text{für } i \in \{4, 5\} \; .
\end{aligned} \tag{8.2}
$$

Die Parameter ς_i, τ_i, δ_i (ς_i - Belastung: die Wahrscheinlichkeit, dass die Quelle i ein Paket in einem Takt generieren wird; τ_i - Durchsatz: die Wahrscheinlichkeit, dass Ziel i ein Paket in einem Takt bekommt; δ_i - Verzögerung: Anzahl der Taktzyklen, die ein Paket braucht bis es Ziel i erreicht) sind von der Netzwerk-Topologie abhängig und müssen mittels der Simulation berechnet werden. Es werden unterschiedliche Rekonfigurationsmöglichkeiten simuliert. Der Entwickler bekommt die Möglichkeit die Effizienz der Topologien miteinander zu vergleichen und so die beste Rekonfiguration für den bestimmten Verkehrsmuster zu finden. In der Tabelle 8.1 ist ein Beispiel für einen Verkehrsmuster (RecMIN mit 16 Ein- und Ausgängen und 4 RecCells) gegeben (Abbildung 6.5). Die Gewichtungen der Prioritäten sind entsprechend zu (8.2) gewählt.

Quelle	$\mathbf{P}_{trans}(\mathbf{Q}_i)$	$\mathbf{P}_{rec}(\mathbf{T}_{10})$	$\mathbf{P}_{rec}(\mathbf{T}_{11})$	$\mathbf{P}_{rec}(\mathbf{T}_{12})$	$\mathbf{P}_{rec}(\mathbf{T}_{rst})$
Q_0, Q_1	0,4875	0,2/16	0,3	0,2/16	0,2/16
Q_2, Q_3	0,3875	0,2/16	0,2/16	0,2	0,2/16
Q_4 - Q_7	0,55	0,1	0,2/16	0,2/16	0,2/16
Q_8 - Q_{15}	0,2	0,2/16	0,2/16	0,2/16	0,2/16

Tabelle 8.1: Verkehrsmuster im RecMIN

In der Tabelle 8.1 bezeichnet Q_i die i-te Quelle (i-ten Quelle); P_{trans} (Q_i) ist die Wahrscheinlichkeit, dass Quelle i ein Paket in einem Takt generieren wird (entspricht ς_i); P_{rec} (T_x) ist die Wahrscheinlichkeit, dass Ziel x Ziel für ein Paket von einer Quelle i (Q_i) ist. P_{rec} (T_{rst}) stellt die Wahrscheinlichkeiten P_{rec} (T_x) für alle restlichen Quellen.

Die Simulationsergebnisse sind in Abbildung 8.1 dargestellt. Die Simulationsparameter für alle Simulationen, die in diesem Kapitel präsentiert werden, lauten: Puffergröße 16 Flit für jeden Puffer, zufällige HOL-Konfliktlösung, alle Pakete sind ein Flit groß, ein Flit ist gleich ein Phit. Abbildung 8.1 präsentiert die Auswertung von verschiedenen RecMIN-Rekonfigurationen, die aus allen möglichen Schaltungs-Modi der RecCells resultieren. Da das benutzte Netzwerk aus 4 RecCells besteht, existieren für ein solches RecMIN $2^4 = 16$ mögliche Topologien[1]. Wie es in Fig. 8.1 vorgestellt ist, hat die η-Funktion die höchstmöglichen Werte für die Topologien mit

[1]jede RecCell hat zwei Schaltungs-Modi (A und B), so wie es im Kapitel 6 beschrieben ist.

der ungeraden Nummer (1, 3, 5, usw.) [2]. Dementsprechend muss für die optimale Kommunikation zwischen den NoC-Knoten für den Verkehrsmuster aus der Tabelle 8.1 das RecMIN in der Topologie Nummer 1,3,5,7,9,11,13, oder 15 rekonfiguriert werden.

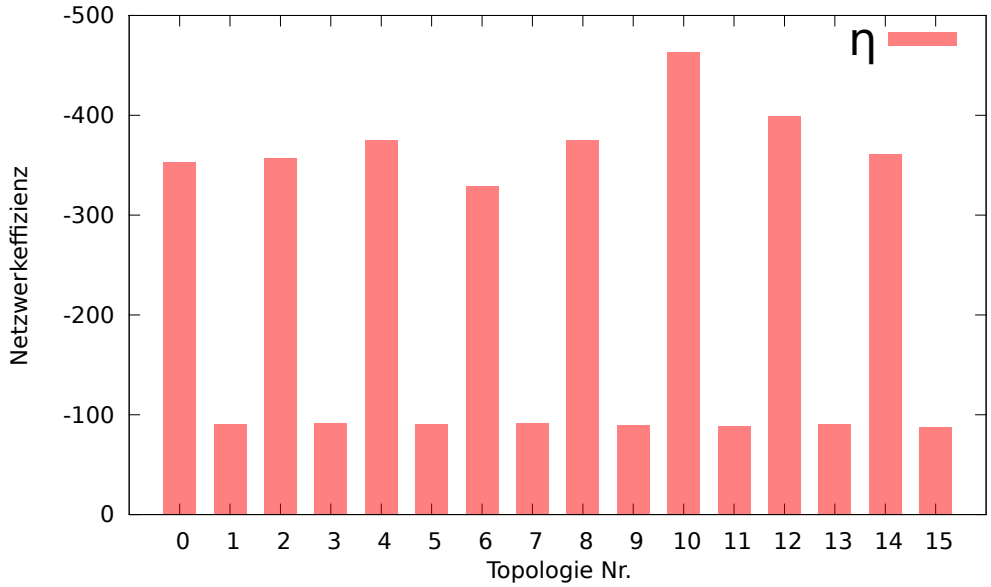

Abbildung 8.1: η-Funktion für 16×16 RecMIN unter Verkehrsmuster-Belastung aus der Tabelle 8.1

8.2 Allgemeine Anforderung an die Rekonfigurationsalgorithmen

Die Aufgaben, die der Rekonfigurationsalgorithmus erfüllen soll, können in drei Schritte eingeteilt werden:

- Beobachtung des Triggers: Es werden die Ereignisse bzw. die Ereignissequenz verfolgt, deren Auftreten das Starten eines Rekonfigurationsprozesses zu Folge hat.
- Das Suchen nach einer alternativen Struktur: das Finden einer Topologie, die den Flaschenhals ersetzen sollte

[2] Erklärung: bei dem Verkehrsmuster aus der Tabelle 8.1 entsteht ein Flaschenhals in RecMIN falls RecCell CO nicht gefaltet wird. Die Topologien mit der geraden Nummer sind die Topologien, bei denen es der Fall ist.

- Der unmittelbare Rekonfigurationsprozess.

Jeder dieser Schritte soll einen hohen Zeitaufwand vermeiden und möglichst einfache mathematische Berechnungen verwenden[3]. Ein weiterer wichtiger Punkt ist die Frage der Stabilität. Es soll die Situation vermieden werden, in der der Algorithmus ständig das Netzwerk zu optimieren versucht. Andernfalls führt der Algorithmus zu einem endlosen Rekonfigurationsprozess, was die normale Funktion des Netzwerks erheblich stört. Zum Beispiel, kann ein solches Problem auftreten, wenn der Rekonfigurationsalgorithmus nicht in der Lage ist, die eindeutig beste Netzwerktopologie zu finden. So weist, nach der Netzwerk-Rekonfiguration, die neue Topologie immer noch einen oder mehrere Flaschenhälse auf. Dieser Zustand leitet das erneute Triggern des Algorithmus ein, was wiederum einen neuen Rekonfigurationsprozess anstößt. Daher sollte nach einer Netzwerk-Rekonfiguration, auch wenn die Netzwerk-Effizienz niedrig geblieben ist, kein neuer Rekonfigurationsprozess gestartet werden, bis sich der Netzwerk-Verkehrsmuster ändert. Daraufhin soll, außer den Triggern für den Rekonfigurationsprozess, ebenfalls ein Beobachtungsverfahren implementiert werden, mithilfe dessen die Änderungen im Verkehrsmuster festgestellt werden können.

Um das Problem der Instabilität zu lösen, wird in dieser Arbeit vorgeschlagen eine Aussage über die Verkehrsmusteränderungen zu treffen. Eine solche Aussage soll aus den Änderungen in den Puffern in NoC gemacht werden: Angenommen, jedem Verkehrsmuster in NoC entspricht ein Puffer-Füllungs-Vektor $\vec{\theta}$. Dann kann man die Verkehrsmusteränderung mithilfe des $\vec{\Delta\theta}$ nachverfolgen. Dabei ist der Vektor $\vec{\Delta\theta}$ als Differenz zwischen zwei vorausberechnete $\vec{\theta}$-Vektoren definiert:

$$\vec{\Delta\theta} = \vec{\theta_2} - \vec{\theta_1} = \begin{pmatrix} \theta_{0,2} \\ \theta_{1,2} \\ \vdots \\ \theta_{N-1,2} \end{pmatrix} - \begin{pmatrix} \theta_{0,1} \\ \theta_{1,1} \\ \vdots \\ \theta_{N-1,1} \end{pmatrix} \tag{8.3}$$

wobei $\theta_{j,i}$ die Länge der Warteschlange in Puffer j ist, die vom Verkehrsmuster i verursacht worden ist. Als Erweiterung dieser Technik können die Paare (Verkehrsmuster-Vektor $\leftrightarrow$ die bestmögliche Topologie) im internen Speicher des SoCs gespeichert werden. In dem Fall kann der Rekonfigurationsalgorithmus das RecMIN unmittelbar in optimale Topologie überführen, falls das Verkehrsmuster sich wiederholen sollte.

[3]Die Umsetzung von komplexen Berechnungen in Hardware erfordert teure Chipfläche.

8.3 Der η-Algorithmus

Der η - Algorithmus verwendet die η-Funktion zur Bewertung und Verbesserung der Wirksamkeit des Netzes. Der Algorithmus nimmt die Mittelwerte der Belastung, des Durchsatzes, der Verzögerung nach 1000 Takten über. Die Anzahl der Takte kann vom Netzwerk-Designer geändert werden. Anhand dieser Werte berechnet der Algorithmus den Betrag der η-Funktion. Falls der berechnete Betrag von η sich unter dem vordefinierten Schwellwert befindet, startet der Algorithmus den Rekonfigurationsprozess.

Die Suche nach einer optimierten Netzwerkstruktur bedeutet ein typisches globales Optimierungsproblem: Das Auffinden einer approximierten Lösung einer Funktion mit einer hohen Komplexität. Der vorgestellte η-Algorithmus nutzt die erschöpfende Suche; es werden alle möglichen Rekonfigurationen ausprobiert, um die optimale NoC-Topolgie zu finden. Obwohl die erschöpfende Suche wohl als das langsamste Suchverfahren gilt, ist es ein sinnvoller Ausweg für kleine Netzwerke[4]. Nachdem alle möglichen Topologien für das RecMIN ausprobiert worden sind, wählt der Algorithmus die Topologie mit dem maximalen η - Wert.

Der η - Algorithmus, in Pseudo-Code geschrieben, ist unten dargestellt:

```
INPUT:  RecMIN, traffic;
OUTPUT: RecMIN_topology;

best_calculated_η := calculate η;
BEGIN
 IF η < η_threshold THEN
    IF no reconfiguration is running THEN
       FOR i := 0 TO i < all_possible_reconfigurations - 1 DO
             simulate topology i;
             calculate η;
            IF calculated η > best_calculated_η THEN
               best_calculated_η := calculated_η;
               best_topology := simulated_topology;
            END IF;
       END FOR;
    END IF;
 END IF;
```

[4] So z. B. für 16×16 RecMIN, in dem nur $2^4 = 16$ Rekonfigurationen möglich sind. Allerdings für Netzwerke mit der höheren Anzahl von RecCells empfiehlt der Autor ein anderes heuristisches Optimierungsverfahren wie Simulated Annealing, genetische Algorithmen oder andere heuristische Algorithmen zu nutzen.

```
 RETURN best_topology;
END;
```

8.3.1 Vorteile und Nachteile des η-Algorithmus

Der Hauptvorteil des η-Algorithmus ist die Möglichkeit, die optimale Netzwerktopologie für jeden Verkehr zu finden. Abbildung 8.2 zeigt eine Analyse der sechs vordefinierte NoC-Verkehrsmuster (im 16 ×16 RecMIN aus vier RecCells) mit dem η-Algorithmus. Diese sechs Verkehrsmuster sind so gewählt worden, dass jedes von ihnen einen Flaschenhals in verschiedenen Bereichen des Netzwerks verursacht. Den entstandenen Flaschenhals kann man aber mithilfe der Rekonfiguration eines oder mehreren RecCells beseitigen. Die gewählten Prioritätsgewichte sind $C_{\varsigma_i} = C_{\tau_i} = 50$ [Taktzyklen/Flit] und $C_{\delta_i} = -1$ [Taktzyklen]$^{-1}$ für alle $i \in \{0, .., N-1\}$. Für jeden Verkehrsmuster zeigt Abbildung 8.2 zwei Werte: Den Minimal- und Maximalwert von η, die durch Rekonfiguration des Netzwerks mit dem η-Algorithmus erreicht werden können.

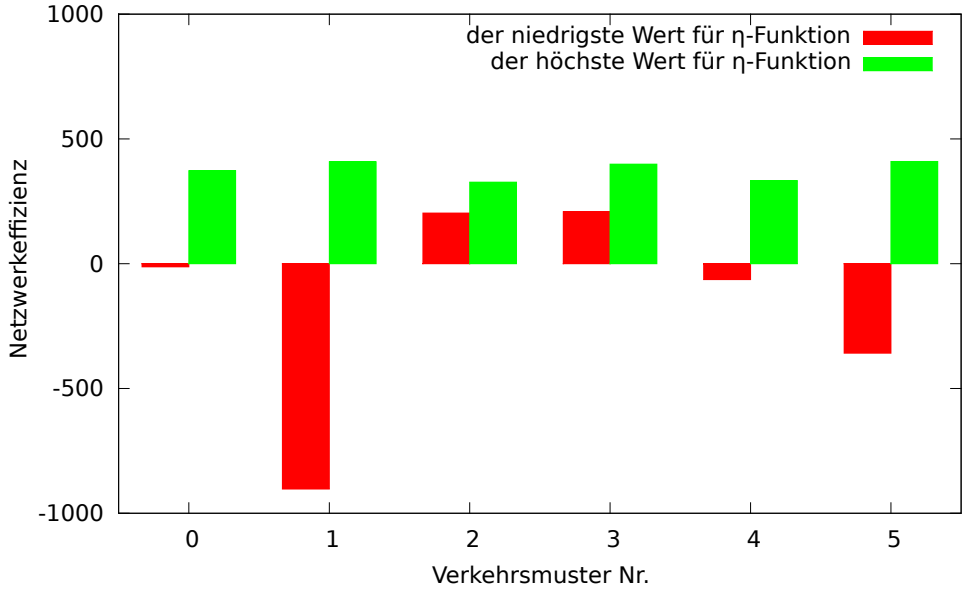

Abbildung 8.2: η-Funktionen für unterschiedliche Verkehrsmuster in 16×16 RecMIN

Es zeigt sich, dass es für einige Verkehrsmuster (z.B. Verkehrsmuster Nr. 1 und Nr. 5) durch Rekonfiguration möglich ist, eine beachtliche Verbesserung der Netzwerkleistung zu erreichen. Auf der anderen Seite gibt es einige Verkehrsmuster (z.B. Verkehrsmuster Nr. 2 und 3), für die Rekonfiguration leider nur zu irrele-

vanten Verbesserungen führt. Für solche Verkehrsmuster ist die Verwendung des η-Algorithmus mit erschöpfender Suche nicht empfehlenswert.

Die zusätzlichen Nachteile des η-Algorithmus sind:

1. Der Algorithmus erfordert eine ständige Führung von Statistiken für den Durchsatz und die Verzögerung der Quellen und Ziele in NoC. Dafür müssen zusätzliche anspruchsvolle IP-Cores (zuständig für die Erhebung statistischer Daten) in den Netzwerkschnittstellen integriert werden. Dies erhöht die Chipfläche, die vom Netzwerk belegt wird, dementsprechend steigen der SoC-Produktionskosten.

2. Der Algorithmus befasst sich mit einem großen Suchraum, falls große NoCs mit mehreren RecCells optimiert werden müssen. Dieses Problem kann z.B. durch Verwendung von Simulated Annealing gelöst werden. In dem Fall liefert der η-Algorithmus jedoch keine garantierte optimale Lösung.

Allgemein lässt sich sagen, dass der η-Algorithmus aufgrund der o. g. Nachteile nicht optimal für den Einsatz in SoC geeignet ist. Dafür kann aber der η-Algorithmus sehr gut in der Simulation verwendet werden. So kann der Netzwerk-Entwickler die Wirksamkeit anderen Rekonfigurationsalgorithmen auswerten, indem ihre Ergebnisse mit den Resultaten des η-Algorithmus verglichen werden.

8.4 Der Minimal-Queues-Algorithmus

Die Analyse der verschiedenen NoCs zeigt, dass in der Regel, je effektiver das Netzwerk ist, desto kürzer sind die Warteschlangen in den Netzwerkpuffern. Sollte ein Engpass im NoC entstehen, so verursacht er das Anwachsen der Warteschlangen in den Puffern in bestimmten Netzwerkabschnitten. Anschließend führt dieser Effekt im Allgemeinen zu einer Erhöhung der Länge der Pufferwarteschlangen im gesamten Netzwerk.

Auf dem oben beschriebenem Effekt basiert die Idee des Minimal-Queues-Algorithmus (MQA): Reagieren auf das Anwachsen der Längen der Pufferwarteschlangen im Netzwerk und Minimieren dieser mittels Rekonfigurationen.

Die Beobachtung der Länge der Pufferwarteschlangen in einem realen SoC ist wesentlich einfacher als die Bearbeitung der Statistiken (von Durchsatz und Verzögerung), die an den Quellen und Zielen gewonnen werden. Somit ist der MQA besser für die Implementierung in SoC geeignet, als es beim η-Algorithmus der Fall ist.

Die Auslösebedingung für den MQA ist, dass die Gesamtzahl der Pakete in den Netzwerkpuffern den vordefinierten Schwellenwert[5] übersteigt. Danach führt der

[5]Der Schwellenwert wird vom Entwickler festgelegt.

MQA $k_s tep$ Rekonfigurationsschritte durch. In jedem Rekonfigurationsschritt wird durch Umschalten des Modus einer einzigen RecCell versucht, die Längen der Pufferwarteschlangen im gesamten NoC deutlich zu verkürzen. Der MQA beginnt an der RecCell mit den längsten Pufferwarteschlangen. Die Zahl $k_s tep$ wird durch den Entwickler festgelegt. Empfohlen ist es aber, den Parameter $k_s tep$ gleich der Hälfte der Anzahl von RecCells im Netzwerk zu setzen. Zum Beispiel falls ein Netzwerk aus vier RecCells besteht, so soll $k_s tep = 2$ sein. Der Rekonfigurationsprozess an sich erfordert weder das Anhalten des NoC-Betriebs noch die komplette NoC-Säuberung der in RecMIN vorhandenden Pakete. Die verwendete Technik ist in [26] beschrieben.

Der MQA, in Pseudo-Code geschrieben, ist unten angegeben:

```
INPUT:  RecMIN, traffic;
OUTPUT: RecMIN_topology;

best_calculated_buffer_sum :=  calculate(buffer_sum);
BEGIN
 IF buffer_sum > buffer_sum_threshold THEN
    IF no reconfiguration is running THEN
       FOR i := 0 TO i < k_step - 1 DO
        list_of_tried_cells := {};
        FOR each RecCell DO
            switching_cell :=  search for
                  RecCell with the highest  buffer_sum_in_cell;
            IF switching_cell ∉ list_of_tried_cells THEN
             switch RecCell mode  (switching_cell);
             simulate topology;
             calculate(buffer_sum);
            #if the buffer queues do not decrease
            IF NOT (calculated buffer_sum <<
                           best_calculated_buffer_sum) THEN
                step back to previous topology;
                add  switching_cell  to  list_of_tried_cells;
            END IF;
       END FOR;
       END FOR;
    END IF;
 END IF;
 RETURN actual_topology;
END;
```

8.4.1 Vorteile und Nachteile des MQA

Wie bereits erwähnt, ist der MQA besser geeignet für SoC als der η-Algorithmus, weil er Informationen von Puffer-Aktivitäten statt Statistiken von Durchsatz und Paketverzögerungswerten verwendet. Darüber hinaus verzichtet MQA auf die erschöpfende Suche: Im Worst Case führt der MQA $k_{step} * n_{cells\_in\_MIN}$ Rekonfigurationsschritte durch ($n_{cells\_in\_MIN}$ die Anzahl RecCells in MIN).

Der Hauptnachteil des MQA ist, dass die optimale Lösung nicht mit Sicherheit gefunden wird. MQA ist ein empirischer Algorithmus. Abbildung 8.3 stellt den Vergleich der Netzwerkleistung zwischen dem η-Algotithmus und dem MQA dar. Bemerkenswert ist, dass nur in einem von sechs Fällen der MQA ein lokales Optimum anstatt des globalen gefunden hat (Verkehrsmuster 5).

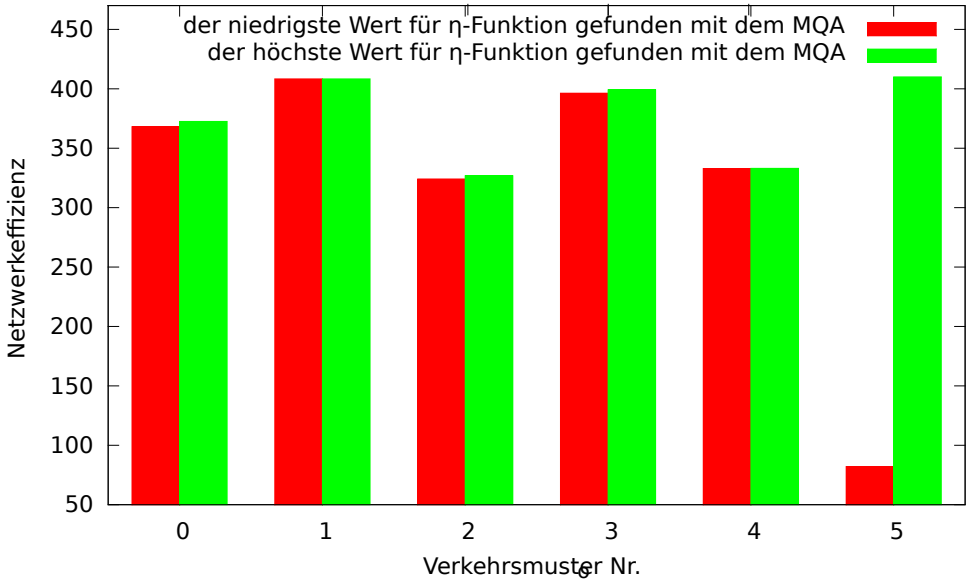

Abbildung 8.3: Vergleich zwischen dem MQA und dem η-Algorithmus für unterschiedliche Verkehrsmuster in 16×16 RecMIN

8.5 Der Pattern-Identification-Algorithmus

Im Allgemeinen kann das Auftreten eines Engpasses in der RecCell durch die Belegung der Puffer identifiziert werden. D. h., wenn ein Engpass eintritt, so kann er durch das Muster der Pufferwarteschlängen im RecCell diagnostiziert werden. Es lassen sich mehrere solche Muster definieren, je nachdem welche Leitung des RecCell überlastet ist. Dementsprechend, wird eines dieser Muster während des Betriebs

des NoCs erkannt, so soll das RecCell rekonfiguriert werden.

Aus der o. g. Überlegung entstand ein Mustererkennungsalgorithmus (Pattern-Identification-Algorithm - PIA). Er überwacht das Auftreten der Engpässe durch die Erkennung des Pufferbelegungsmusters. Die Überwachung findet gleichzeitig bei allen RecCell statt. Wird in einem der RecCells ein Engpass erkannt, so führt der PIA die erforderliche Rekonfiguration durch. Bei einer Erkennung von mehr als einem Engpass-Muster in verschiedenen RecMIN-Stufen wird das RecCell der hinteren Stufe vor RecCell der vorderen Stufe rekonfiguriert[6].

Der PIA, in Pseudo-Code geschrieben, ist unten angegeben:

```
INPUT:  RecMIN, traffic;
OUTPUT: RecMIN_topology;

BEGIN
 FOR i := stage_number -1 DOWNTO 0 DO
   FOR each stage in RecMIN beginning from stage[i]  DO
     FOR each RecCell in this stage DO
      IF no reconfiguration is running THEN
        IF  is one of the patterns found  THEN
             reconfigure the cell according to the found pattern
        END IF;
      END IF;
     END FOR;
   END FOR;
 END FOR;

 RETURN actual_topology;
END;
```

8.5.1 Vorteile und Nachteile des PIAs

Ein wichtiger Vorteil des PIAs ist, dass er nicht alle Topologien traversieren muss, um eine bessere zu erkennen; es gelingt ihm eine solche direkt zu lokalisieren. Ebenfalls führt der PIA eine Rekonfiguration ausschließlich nur dann durch, wenn die Netzwerkeffizienz deutlich verbessert werden kann. Im Worst Case würde der PIA $2^X * n_{cells\_in\_stage}$ Rekonfigurationsschritte durchführen müssen (wobei X die Anzahl der Stufen im RecMIN und $n_{cells\_in\_stage}$ die Anzahl RecCells in jeder MIN-Stufe ist). Die Simulationen zeigen aber, dass der PIA effizienter als der MQA ist, was die

[6]Dies Verfahren basiert sich auf empirischen Erkenntnissen

Anzahl der Rekonfigurationsschritte angeht, da ein Engpass meistens in ein einzigen Rekonfigurationsschritt beseitigt wird.

Weiterhin nutzt der Mustererkennungsalgorithmus, so wie MQA, ausschließlich die Pufferfüllungen als Triggerinformation. Dies macht die Implementierung des PIAs im SoC verhältnismäßig einfach.

Ein weiterer Vorteil des PIAs ist die Fähigkeit die Rekonfigurationsschritte für RecCells der gleichen MIN-Stufe gleichzeitig durchzuführen und somit die Geschwindigkeit des kompletten Rekonfigurationsprozesses zu erhöhen.

Der Nachteil des PIAs hingegen ist, dass die zusätzlichen Speicherregister auf den Chip geladen werden müssen, um die Liste der zu erkennenden Muster zu speichern. Ein weiterer Nachteil ist, dass das Implementieren der zu "weichen" Erkennungsmustern, zur Instabilität des PIAs führen kann: So wird der PIA stets ein Muster in den RecMINs wiedererkennen, was einen endlosen Rekonfigurationsprozess zur Folge haben wird.

Abbildung 8.4 stellt den Vergleich der Netzwerkleistung zwischen dem η-Algotithmus und dem PIA dar. Werden die Erkennungsmustern sachgemäß gut implementiert, so ermöglicht der PIA in alle sechs Fällen das globale Optimum zu finden.

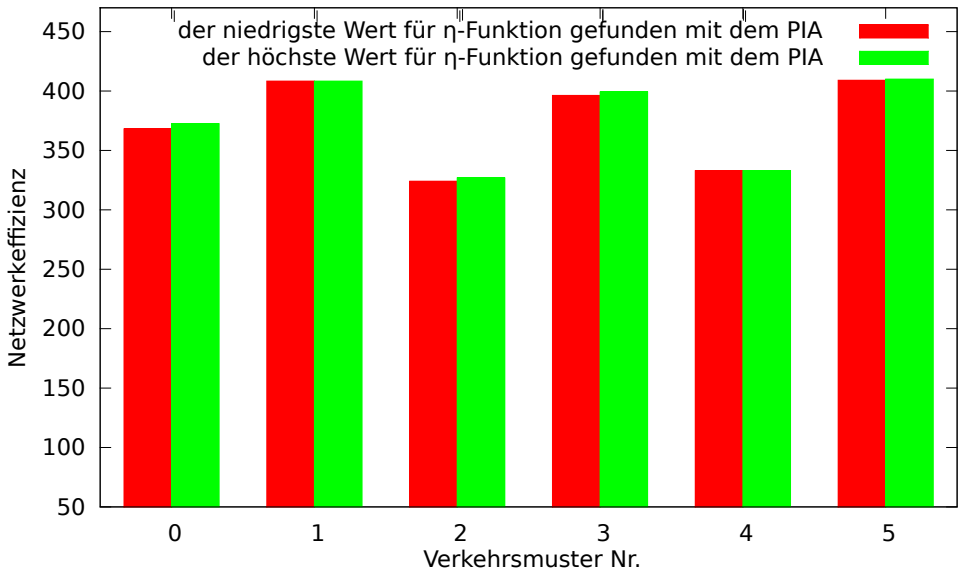

Abbildung 8.4: Vergleich zwischen dem PIA und dem η-Algorithmus für unterschiedliche Verkehrsmuster in 16×16 RecMIN

8.6 Vergleich und Erläuterung der vorgeschlagenen Rekonfigurationsalgorithmen

Der Vergleich zwischen den drei vorgestellten Algorithmen wird in der Tabelle 8.2 vorgestellt.

	η-Algorithmus	**MQA**	**PIA**
Suchverfahren	erschöpfende Suche	Step-Back	Divide&Conquer
Zeitkomplexität			
Worst-case	$\mathcal{O}\ (c^n)$	$\mathcal{O}\ (c * n)$	$\mathcal{O}\ (1)$
Best-case	$\mathcal{O}\ (c^n)$	$\mathcal{O}\ (1)$	$\mathcal{O}\ (1)$
Zusatzelemente[7]	nötig	unnötig	unnötig
Lösungstyp	optimal	suboptimal	suboptimal

Tabelle 8.2: Der Vergleich zwischen den Algorithmen

Ein Vergleich zwischen den drei vorgeschlagenen Algorithmen liefert folgende Schlussfolgerungen:

1. Es ist empfehlenswert den η-Algorithmus ausschließlich als Referenz zu verwenden [8]: Die damit erzielten Ergebnisse sind die optimale Lösung. Diese kann verwendet werden, um die von anderen Algorithmen (die keine erschöpfende Suche benutzen) erzielten Ergebnisse zu vergleichen. Um den SoC η-Algorithmus, wie oben beschrieben, einsetzen zu können, ist es notwendig, die zusätzlichen Elemente im NoC für Durchsatz- und Verzögerungsmessung zu integrieren. Jedoch kann der η-Algorithmus in Computersimulationen während eines frühen Stadiums der Entwicklung eines RecMIN effektiv angewendet werden. Mithilfe des η-Algorithmus können also potenzielle Möglichkeiten zur Optimierung des Verkehrs in RecMIN im Bezug auf spezifische Arten von Lasten erkundet werden.

2. Der Minimal-Queues-Algorithmus (MQA) ist schneller als der η-Algorithmus und erfordert weniger Ressourcen (Fläche auf dem SoC). Er kann leicht in die Hardware integriert werden. Obwohl MQA gegen PIA in der Geschwindigkeit verliert, ist es bei MQA-Realisierung nicht erforderlich, die Pufferbelegungsmuster (die sich für verschiedene Lastfälle unterscheiden können) zu identifizieren, und somit erscheint eine vorläufige Analyse der Belastung des Netzwerks nicht notwendig. Das Implementieren des MQAs ist empfehlenswert, falls die Vorkenntnisse über die Eigenschaften des möglichen Verkehrs im Netzwerk nicht vorhanden sind.

[7] für Durchsatz- und Verzögerungsmessung
[8] eine Realisierung des η-Algorithmus in SoC is mit großen Hardwareaufwand verbunden

3. Obwohl der Pattern-Identification-Algorithmus (PIA) einige Probleme mit der Geschwindigkeit im Worst Case hat, ist PIA dennoch im Regelfall immer noch schneller als alle oben beschriebenen Algorithmen. Die Umsetzung des PIAs auf SoC ist relativ einfach. Im Falle, dass die Muster für den PIA optimal implementiert sind, ist der PIA von allen drei hier für RecMIN vorgeschlagenen Reconfigurationsalgorithmen die effizienteste Lösung.

9 Fazit und Ausblick

9.1 Zusammenfassung

In dieser Arbeit ist eine Rekonfigurations-Topologie RecMIN entwickelt worden. RecMIN besteht aus einzelnen rekonfigurierbaren Zellen RecCells. Die RecCells sind so gebaut, dass die Komponenten der alten Topologie (Puffer und Multiplexer) auch nach der Rekonfiguration für die neue Topologie aktiv verwendet werden.

Für die Simulation und Evaluation wurde eine speziell dafür implementierte Software-Umgebung *RecSim* benutzt. Die Ergebnisse der Simulation haben gezeigt, dass RecMIN das Potential hat, die Leistungsfähigkeit einiger Anwendungen, insbesondere derer, die zwischen zwei oder mehreren Verkehrsprofilen wechseln, drastisch zu verbessern. So, z. B., würde mit RecMIN die Paketverzögerungen einiger Anwendungen, bis zu Faktor sieben reduzieret.

Zusätzlich wurden drei Rekonfiguration-Algorithmen für die Selbstanpassung des RecMINs vorgestellt:

- Der η-Algorithmus. Dieser Algorithmus kann, in erster Linie, in Simulationen verwendet werden. Jedoch ist das Integrieren des η-Algorithmus in ein SoC mit großem Hardware-Aufwand verbunden.

- Der Minimal-Queues-Algorithmus (MQA). Der MQA kann mit kleinerem Hardware-Aufwand als der η-Algorithmus in SoC integriert werden. Der MQA ist dann zu empfehlen, falls die Vorkenntnisse über die Eigenschaften des möglichen Verkehrs im Netzwerk nicht vorhanden sind.

- Der Pattern-Identification-Algorithmus (PIA) ist in meisten Fällen schneller als der η-Algorithmus und der Minimal-Queues-Algorithmus. Die Umsetzung des PIAs auf SoC ist relativ einfach. Im Falle, dass die Muster für den PIA optimal implementiert sind, ist der PIA von allen drei hier für RecMIN vorgeschlagenen Reconfigurationsalgorithmen die effizienteste Lösung.

- Der Pattern-Identification-Algorithmus (PIA). Der PIA ist in meisten Fällen schneller als der η-Algorithmus und der MQA. Die Umsetzung des PIAs auf SoC ist relativ einfach. Im Falle, dass die Muster für den PIA optimal implementiert sind, ist der PIA von allen drei hier für RecMIN vorgeschlagenen Reconfigurationsalgorithmen die effizienteste Lösung.

Der erste Schritt von der Simulation zur Hardware-Realisierung wurde gemacht, indem verschiedene FPGA-Architekturen untersucht worden sind. Der Schwerpunkt dieser Untersuchung war die beste Architektur bzgl. Flächenverbrauch und Verzögerung zur Puffer-Realisierung in RecMIN zu finden. Die Untersuchung zeigt, dass für Hardwarerealisierung der RecMIN-Architektur auf FPGA ist Xilinx-FPGA mit FIFO-Speicherprinzip als eine der besten Lösungen zu empfehlen. Wird aber für RecMIN ein Ringspeicher gewählt, so wäre die Wahl zwischen Altera- und Xilinx-FPGAs für jeden konkreten Fall mithilfe der Daten aus den Abbildungen 7.9 - 7.11 individuell zu treffen.

9.2 Allgemeiner Ausblick

Eine weitere Aufgabe zur Fortsetzung dieser Arbeit wäre die komplette Realisierung des RecMINs auf FPGAs und später in ASIC-Technologie. Ein weiterer wichtiger Punkt ist das Testen von RecMIN mit den Paketverkehrs-Modellen, die von real existierenden Prozessorkernen in Realität produziert werden. Allerdings ist es schwierig, an diese Paketverkehrs-Modelle zu kommen, da sie ein Teil von internen Informationen der großen Unternehmen (so wie Intel und AMD) sind; es ist also eine Partnerschaft mit diesen Unternehmen erforderlich.

Ein Pluspunkt kann auch die mögliche Erweiterung des RecMINs sein; z. B., die Implementierung der Übertragung von Paketen, die aus mehr als einem Flit bestehen. Entsprechend müssen dafür weitere Switching-Techniken, so wie Wormhole-Switching und Cut-Through-Switching implementiert werden. Es müssen ebenso Simulationen durchgeführt werden, die die RecMIN-Eigenschaften untersuchen, falls in NoC Prioritäten oder Multicast-Möglichkeiten benutzt werden.

Ein weiteres interessantes Thema ist die genauere Analyse der RecMIN auf die Energieeffizienz, die im Rahmen dieser Arbeit nicht gemacht worden war. Da der Energieverbrauch für die modernen Systeme in der heutigen Zeit von erheblicher Bedeutung ist (es wird mehr und mehr über die Green-Technology gesprochen), kann der Energieverbrauch der RecMIN, im Vergleich zu anderen NoC-Topologien, ein interessantes Untersuchungsthema sein.

9.3 Topologievorschlag

Es ist auch denkbar, dieselbe Rekonfigurations-Topologie, in der ein 4×4 Router zu vier 2×2 Router rekonfiguriert wird, auch für die anderen NoC-Topologien zu benutzen. So kann z. B. eine solche Rekonfiguration für Router in Mesh-Topologien, die 4×4 Ein- und Ausgänge haben, benutzt werden. Eine interessante Weiterentwicklung ist eine "Waben"-Topologie, die aus RecMIN-Zahlen gebaut werden kann (Abbil-

dung 9.1). Die Modellierung und Analyse dieser Architektur ist mit der im Rahmen dieser Arbeit entwickelten Software-Umgebung *RecSim* leicht zu realisieren. Die entwickelten Rekonfigurations-Algorithmen können ebenfalls für solche Architekturen verwendet werden.

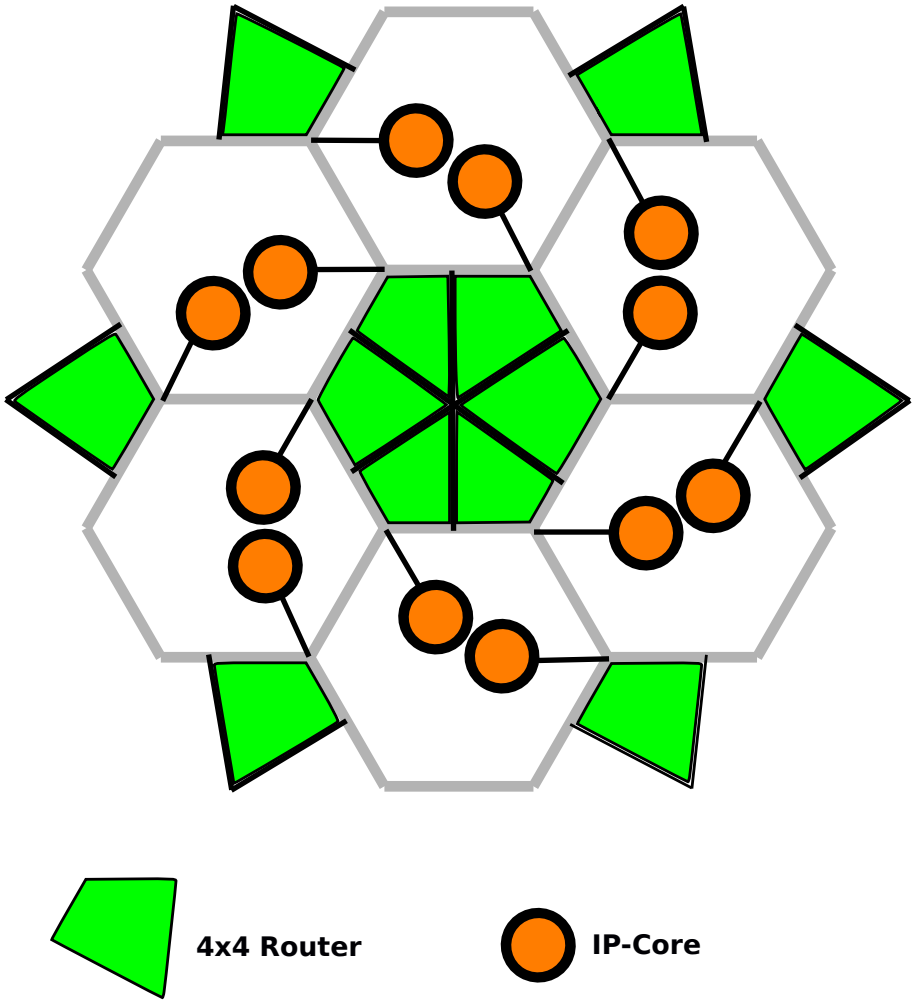

Abbildung 9.1: "Waben"-Topologie aus 4×4 Router

A Pseudocode für den implementierten Puffer-Speicher

Der Pseudocode für den implementierten LIFO aus Kapitel 7.10.1 lautet wie folgt:

```
IF reset THEN
 schreibpointer:=0;
 lesepointer:=0;
 memory_voll:=0;
 memory_leer:=1;
ELSE
 BY RISING EDGE BEGIN

  IF read request THEN
    IF memory is not empty THEN
      IF memory_voll=1 THEN
        memory_voll:=0;
      END IF
       give out memory[lesepointer];
      IF lesepointer=0 THEN
        memory_leer := 1;
        schreibpointer:=lesepointer;
      ELSE
        decrement schreibpointer;
        decrement lesepointer;
      END IF;
    END IF;

    IF write request THEN
      IF memory is not full THEN
         write in memory[schreibpointer];
        IF memory_leer=1 THEN
```

```
         memory_leer:=0;
         increment schreibpointer;
        ELSE
         increment lesepointer;
         IF lesepointer=ELEMENTE-1 THEN
          memory_voll := 0;
          increment schreibpointer;
         ELSE
          memory_voll := 1;
         END IF;
        END IF;
    END IF;

    IF write request AND read request THEN
      IF memory_leer=0 AND memory_voll=0 THEN
       give out memory[lesepointer];
       write in memory[lesepointer];
      END IF;
      IF memory_leer=1 AND memory_voll=0 THEN
       memory_leer:=0;
        write in memory[schreibpointer];
       increment schreibpointer;
      END IF;
      IF memory_leer=0 AND memory_voll=1 THEN
        give out memory[schreibpointer];
        give out memory[schreibpointer];
      END IF;

 END RISING EDGE;
```

Der Pseudocode für den implementierten FIFO aus Kapitel 7.10.2 lautet wie folgt:

```
IF reset THEN
 schreibpointer:=0;
 memory_voll:=0;
 memory_leer:=1;
ELSE
 BY RISING EDGE BEGIN
```

```
IF read request THEN
  IF memory is not empty THEN
    IF memory_voll=1 THEN
      memory_voll:=0;
    END IF
     give out memory[0];
     shift down all memory register
    IF schreibpointer=1 THEN
      memory_leer := 1;
      schreibpointer:=0;
    ELSE
      decrement schreibpointer;

    END IF;
  END IF;

  IF write request THEN
    IF memory is not full THEN
       write in memory[schreibpointer];
      IF memory_leer=1 THEN
       memory_leer:=0;
      ELSE
       IF schreibpointer=ELEMENTE-1 THEN
        memory_voll := 0;
       ELSE
        memory_voll := 1;
       END IF;
       increment schreibpointer;
      END IF;
  END IF;

  IF write request AND read request THEN
    IF memory_leer=0 AND memory_voll=0 THEN
     give out memory[0];
     shift down all memory register
     write in memory[schreibpointer-1];
    END IF;
    IF memory_leer=1 AND memory_voll=0 THEN
     memory_leer:=0;
      write in memory[schreibpointer];
     increment schreibpointer;
```

```
      END IF;
      IF memory_leer=0 AND memory_voll=1 THEN
        give out memory[0];
        shift down all memory register
       decrement schreibpointer;
      END IF;

 END RISING EDGE;
```

Der Pseudocode für einen implementierten Ringspeicher aus Kapitel 7.10.3 lautet wie folgt:

```
IF reset THEN
 schreibpointer:=0;
 lesepointer:=0;
 memory_voll:=0;
 memory_leer:=1;
ELSE
 BY RISING EDGE BEGIN

  IF read request THEN
    IF memory is not empty THEN
      IF memory_voll=1 THEN
        memory_voll:=0;
      END IF
       give out memory[lesepointer];
      IF lesepointer<ELEMENTE-1 THEN
         increment lesepointer;
         IF lesepointer=schreibpointer THEN
          memory_leer:=1;
         ELSE
          memory_leer:=0;
         END IF
      ELSE
         IF schreibpointer=0 THEN
          memory_leer:=1;
         ELSE
          memory_leer:=0;
         END IF
```

```
    lesepointer:=0
  END IF;
END IF;

IF write request THEN
  IF memory is not full THEN
     write in memory[schreibpointer];
  IF schreibpointer<ELEMENTE-1 THEN
     increment schreibpointer;
     IF lesepointer=schreibpointer THEN
      memory_voll:=1;
     ELSE
      memory_voll:=0;
     END IF
  ELSE
     IF lesepointer=0 THEN
      memory_voll:=1;
     ELSE
      memory_voll:=0;
     END IF
    schreibpointer:=0
  END IF;
END IF;
IF write request AND read request THEN
   write in memory[schreibpointer];
  IF memory_leer=0 THEN
    give out memory[lesepointer];
   IF lesepointer<ELEMENTE-1 THEN
     increment lesepointer;
   ELSE
     lesepointer=0;
   END IF;
  ELSE
     memory_leer=0;
  END IF
  IF schreibpointer<ELEMENTE-1 THEN
     increment schreibpointer;
  ELSE
     schreibpointer:=0;
  END IF;
END IF;
```

```
END RISING EDGE;
```

B Kurze Bedienungsanleitung in RecSim

Um die RecSim-Simulation zu starten, muss man, wie in Kapitel 5 beschrieben ist, drei folgende Schritte machen:

- Die Elemente des Netzwerks initialisieren
- Verbindungen zwischen den Elementen definieren
- Simulation starten

Hier wird ein kleines Beispiel gegeben, wie man das in der Abbildung B.1 dargestellte Netzwerk initialisieren und starten kann

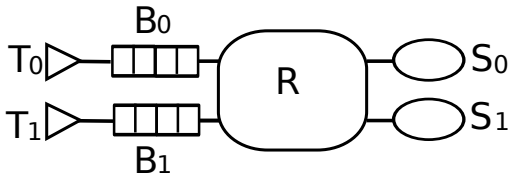

Abbildung B.1: Ein 2-MIN

1. Initialisieren des Zeigers des Typs Network

   ```
   Network* net;
   ```

2. Initialisieren der Generatoren

   a) Verkehr für den Generator definieren

   ```
   std::vector<double> trafik1;

   trafik1.push_back(0.6/2);
   trafik1.push_back(0.8/2);
   ```

Die Werte im Vektor `trafik1` sind die Wahrscheinlichkeiten, mit denen ein Paket für das Ziel mit Nummer x in einem Takt erzeugt wird. In unserem Beispiel wird das Paket für das Ziel T_0 mit die Wahrscheinlichkeit $0,6/2 = 0,3$ erzeugt, und mit der Wahrscheinlichkeit $0,8/2 = 0,4$ erzeugt. Die Wahrscheinlichkeit, mit der ein Paket in einem Takt bei der Quelle T_0 erzeugt wird, ist dann $0,4 + 0,3 = 0,7$.

b) Initialisieren der Zufallsgenerator für die Paketeinerzeigen:

```
Mtwister m_prng = new Mtwister();
```

c) Initialisieren Paketen-Generator für der Quellen

```
UnicastGenerator generator =
        (new UnicastGenerator(m_prng,trafik1));
```

3. Initialisieren der Paketüberwachung, die zu Kontrolle der Pakete im Netzwerk dient

```
PacketDispatcher packetDisp = new PacketDispatcher();
```

4. Erzeugen der zwei Quellelementen S_0 und S_1

```
SourceBuffer S0 = new SourceBuffer(0, generator, packetDisp);
SourceBuffer S1 = new SourceBuffer(1, generator, packetDisp);
```

5. Zwei Puffer mit der Tiefe 4 erstellen

```
Buffer B0 = new Buffer(2, 4);
Buffer B1 = new Buffer(3, 4);
```

6. Zwei Zielen erstellen:

```
TargetBuffer T0 = new TargetBuffer(4, generator, packetDisp);
TargetBuffer T1 = new TargetBuffer(5, generator, packetDisp);
```

7. Netzwerkverbindungen definieren

a) Die Quellen mit dem Puffer verbinden

```
Connector C0 = new Connector();
Connector C1 = new Connector();
Connector C2 = new Connector();
Connector C3 = new Connector();

C0->connect(S0->getPin(0),B0->getPin(0));
C1->connect(S0->getPin(1),B0->getPin(1));

C2->connect(S1->getPin(0),B1->getPin(0));
C3->connect(S1->getPin(1),B1->getPin(1));
```

b) Die Router erzeugen und verbinden:

```
Router R=new Router();

//Konstruiere Kontainer fuer Inputs
std::vector<Pin*>* inp=new std::vector<Pin*>();
//Konstruiere Kontainer fuer  Outputs
std::vector<Pin*>* outp=new std::vector<Pin*>();


//RouterEingaenge (mit Buffer verbunden)
 inp->push_back(B0->getPin(2)); //SignalLeitung
 inp->push_back(B0->getPin(3)); //SteuerLeitung
//RouterAusgaenge (mit Targets verbunden)
 outp->push_back(T0->getPin(0)); //SignalLeitung
 outp->push_back(T0->getPin(1)); //SteuerLeitung

//RouterEingaenge (mit Buffer verbunden)
 inp->push_back(B1->getPin(2)); //SignalLeitung
 inp->push_back(B1->getPin(3)); //SteuerLeitung
//RouterAusgaenge (mit Targets verbunden)
 outp->push_back(T1->getPin(0)); //SignalLeitung
 outp->push_back(T1->getPin(1)); //SteuerLeitung
```

c) das Routing für den Router definieren

```
std::vector<std::vector<int> >* routing =
    new std::vector<std::vector<int> >
    (2,std::vector<int>(1,0));

for (int i=0; i<2;i++)
    //Alle Pakete mit der adresse X,
```

```
//werden zu der Augang X gesendet
routing->at(i).at(0) = i;
```

d) Die Routing-Tabelle in den Router einbinden

```
R->connect(inp, outp,routing,m_prng);
```

e) Messpunkte definieren, hier werden als Beispiel drei Messpunkte mit dem Konfidenzintervall von 95% definiert. Es werden zwei weitere Messpunkte definiert, einer für die Messung des Durchsatzes und der andere für der Messung der Warteschlangengröße.

```
MeasureBufferThroughput  M1= new MeasureBufferThroughput(0.95));
MeasureBufferThroughput  M2 =new MeasureBufferQueue(0.95));
```

8. Alle Komponenten in die Komponenten-Liste eingeben und ins Netzwerk hinzufügen

```
net->m_components.push_back(S0);
net->m_components.push_back(S0);
net->m_components.push_back(B0);
net->m_components.push_back(B1);
net->m_components.push_back(T0);
net->m_components.push_back(T1);

net->m_asynchronComponents.push_back(C0);
net->m_asynchronComponents.push_back(C1);
net->m_asynchronComponents.push_back(C2);
net->m_asynchronComponents.push_back(C3);
net->m_asynchronComponents.push_back(R);

net->m_measurements.push_back(M1);
net->m_measurements.push_back(M2);
```

9. Simulieren

```
net->compute();
```

Die Simulationsergebnisse werden in der Konsole ausgegeben. Jedoch kann man die Ausgabe leicht in eine Datei mit dem Linux-Befehl „<“ umleiten.

Will man die Topologie des Netzwerks als Ausgabe bekommen, so muss man die Funktion `writeFile()` aufrufen. Die Topologie wird in Graphviz-Format gespeichert (sehe Kapitel 5).

10. Das Netzwerk löschen, um Speicher zu befreien.

```
delete net;
```

Alternativ kann man die komplette Topologie mit Hilfe des automatischen Netzwerkgenerierung-Algorithmus, so wie in Kapitel 5 beschrieben ist, erstellen, und dann die Stimulation starten.

Folgende Schritte müssen gemacht werden, um die Topologie, die in Bild B.2 dargestellt ist, zu simulieren.

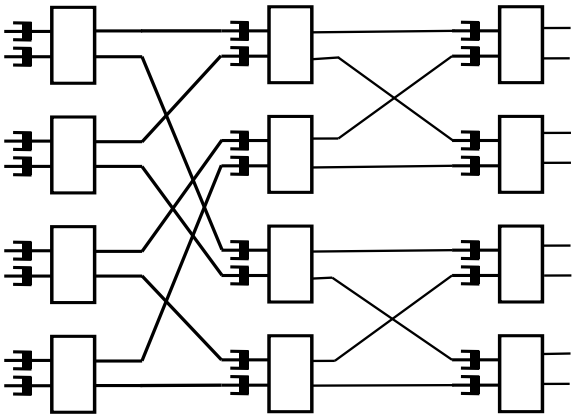

Abbildung B.2: Ein 8-MIN

1. Ein Netzwerk mit acht Quellen, acht Senken, und einem 8×8 Router-Element erstellen. Das kann man leicht mit dem folgenden Code machen

```
Network* net;
int anzahlDerQuellen=8;
int pufferTiefe=12;
int packetenGenerierungsWahrscheinlichkeit=0.1;

net=new Network(anzahlDerQuellen, pufferTiefe,
        packetenGenerierungsWahrscheinlichkeit);
```

2. Die Rekonfiguration der Router-Elemente starten

```
int pufferTeilungsTiefe=4;
int routerEingaengeInZweiteStuffe=4;

net->m_asynchronComponents[2*anzahlDerQuellen]->
       startSplitRouter(pufferTeilungsTiefe);

net->splitRouterNumberIn2Stages
   (2*anzahlDerQuellen, routerEingaengeInZweiteStuffe);



pufferTeilungsTiefe=4;
routerEingaengeInZweiteStuffe=2;


for (int i=16; i>=15; i--)

    net->m_asynchronComponents[i]->
         startSplitRouter(pufferTeilungsTiefe);

    net->splitRouterNumberIn2Stages
         (i, routerEingaengeInZweiteStuffe);


//end for
```

3. Simulation

```
net->compute();
```

4. Das Netzwerk löschen, um Speicher zu befreien.

```
delete net;
```

C Abkürzungsverzeichnis

BRAM Block Random Access Memory

CLB Cell Logic Block (Basisblock)

FIFO-Speicher First-in-First-out Speicher

FPGA Field Programmable Gate Array

HOL Head of Line

HOLB Head of Line Blockade

IP Blocks Intellectual Property Blocks

IQ Input Queueing

ISE Integrated Synthesis Environmen

LAB Logic Array Block

LIFO-Speicher Last-in-First-out Speicher

LUT Look-up Table

MIN Multistage Interconnection Network

MQ Multiple Queueing

MQA Der Minimal-Queues-Algorithmus

NoC Network-on-Chip

OQ Output Queueing

PIA Der Pattern-Identification-Algorithmus

RAM Random Access Memory

RecCell Reconfiguration Cell

RecHC Reconfiguration Half Cell

RecMIN Reconfigurable Multistage-Interconnection Network

SoC Systems-on-Chip

VOQ Virtuelle Ausgangs-Warteschlangen

XST Xilinx Synthesis Tool

D Literaturverzeichnis

[1] M.A. Al Faruque, T. Ebi, and J. Henkel. ROAdNoC: Runtime observability for an adaptive network on chip architecture. In *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on*, Nov. 2008.

[2] M.A. Al Faruque, T. Ebi, and J. Henkel. Configurable links for runtime adaptive on-chip communication. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, Apr. 2009.

[3] Altera. *Cyclone II Device Handbook, Volume 1.* USA, Feb. 2007.

[4] Research at Intel. Intel's teraflops research chip. *Intel Corporatin*, 2007.

[5] M. Bakhouya, S. Suboh, J. Gaber, and T. El-Ghazawi. Analytical performance comparison of 2d mesh, wk-recursive, and spidergon nocs. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, 2010.

[6] Paul Beekhuizen. *Performance Analysis of Networks on Chips.* PhD thesis, Technische Universiteit Eindhoven, Netherlands, 2009.

[7] L. Benini and D. Bertozzi. Network-on-chip architectures and design methods. *Computers and Digital Techniques, IEE Proceedings* -, 152(2), März 2005.

[8] Rene L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1), Jan. 1991.

[9] Rene L. Cruz. A calculus for network delay, part II: Network analysis. *IEEE Transactions on Information Theory*, 37(1), Jan. 1991.

[10] R.L. Cruz. Quality of service guarantees in virtual circuit switched networks. *Selected Areas in Communications, IEEE Journal on*, 13(6), Aug. 1995.

[11] Carsten Gremzow Daniel Lüdtke and Dietmar Tutsch. An application-optimized network on chip platform. *12. GI/ITG/GMM-Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, Berlin*, 12, 2009.

[12] Giovanni De Micheli and Luca Benini. *Networks on Chips*. Morgan Kaufmann Publishers, 2006.

[13] Ge Fen, Wu Ning, and Wang Qi. Simulation and performance evaluation for network on chip design using opnet. In *TENCON 2007 - 2007 IEEE Region 10 Conference*, Nov. 2007.

[14] C. Fox and D. Wilson. Visualization in interrogator using graphviz. In *Information Assurance Workshop, 2006 IEEE*, Jun. 2006.

[15] David Geer. Chip makers turn to multicore processors. *Computer*, 38(5), 2005.

[16] P. Heidelberger and P.D. Welch. A spectral method for confidence interval generation and run length control in simulations. *Communications of the ACM*, 24(4), 1981.

[17] F. Jafari, A. Jantsch, and Zhonghai Lu. Output process of variable bit-rate flows in on-chip networks based on aggregate scheduling. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, 2011.

[18] F. Jafari, A. Jantsch, and Zhonghai Lu. Worst-case delay analysis of variable bit-rate flows in network-on-chip with aggregate scheduling. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, 2012.

[19] Yih-Chyun Jenq. Performance analysis of a packet switch based on single–buffered banyan network. *IEEE Journal on Selected Areas in Communications*, SAC–1(6), Dec. 1983.

[20] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. J. Dally. *BookSim 2.0 User's Guide*. USA, 2010.

[21] Chen Jin-Wen, Tang Liang, Xi Hong-sheng, and Zhu Jin. A stochastic network calculus based approach for on-chip networks. In *Control Conference (CCC), 2011 30th Chinese*, 2011.

[22] M.J. Karol, M.G. Hluchyj, and S.P. Morgan. Input versus output queueing on a space-division packet switch. *Communications, IEEE Transactions on*, 35(12), 1987.

[23] Frank Kesel and Ruben Bartholomä. *Entwurf von Digitalen Schaltungen und Systemen mit HDLs und FPGAs*. Oldenbourg Wissenschaftsverlag, Oldenbourg, 2 edition, 2009.

[24] Leonard Kleinrock. *Queueing Systems, Volume 1: Theory*. Wiley Interscience, New York, 1975.

[25] Zhongqi Li, Xiang Ling, and Jianhao Hu. Msns: A top-down mpi-style hierarchical simulation framework for network-on-chip. In *Communications and Mobile Computing, 2009. CMC '09. WRI International Conference on*, volume 2, Jan. 2009.

[26] A. Logvinenko, C. Gremzow, and D. Tutsch. Recmin: A reconfiguration architecture for network on chip. In *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on*, 2013.

[27] A. Logvinenko and D. Tutsch. A reconfiguration technique for area-efficient network-on-chip topologies. In *Performance Evaluation of Computer Telecommunication Systems (SPECTS), 2011 International Symposium on*, Jun. 2011.

[28] Daniel Lüdtke and Dietmar Tutsch. Lossless static vs. dynamic reconfiguration of interconnection networks in parallel and distributed computer systems. In *Proceedings of the 2007 Summer Computer Simulation Conference (SCSC'07); San Diego*. SCS, Jun. 2007.

[29] Daniel Lüdtke and Dietmar Tutsch. Chip multiprocessor traffic models providing consistent multicast and spatial distributions. *Transactions of the SCS*, 84, 2008.

[30] Daniel Lüdtke and Dietmar Tutsch. The modeling power of CINSim: Performance evaluation of interconnection networks. *Computer Networks*, 53(8), 2009.

[31] Daniel Lüdtke, Dietmar Tutsch, Arvid Walter, and Günter Hommel. Improved performance of bidirectional multistage interconnection networks by reconfiguration. SCS, Apr. 2005.

[32] Mingsong Lv, Ying Guo, Nan Guan, and Qingxu Deng. Rtnoc: A simulation tool for real-time communication scheduling on networks-on-chips. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 4, Dec. 2008.

[33] C.A. Mack. Fifty years of moore's law. *Semiconductor Manufacturing, IEEE Transactions on*, 24(2), 2011.

[34] R. Manevich, I. Walter, I. Cidon, and A. Kolodny. Best of both worlds: A bus enhanced noc (benoc). In *Electrical and Electronics Engineers in Israel (IEEEI), 2010 IEEE 26th Convention of*, Nov. 2010.

[35] F. Moraes, N. Calazans, L. Mello, A. Möller, , and L. Ost. HERMES:an infrastructure for low area overhead packet-switching networks on chip. *the VLSI Journal*, 38(1), 2004.

[36] Boot N. Throughput and delay analysis for a single router in networks on chip. Master's thesis, Technische Universiteit Eindhoven, Netherlands, 2005.

[37] A. Patooghy and S.G. Miremadi. Xyx: A power and performance efficient fault-tolerant routing algorithm for network on chip. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, Feb. 2009.

[38] R. M. Ramanathan. Intel multicore processors: Making the move to quad core and beyond. *White Paper, Intel Corporatin*, 2006.

[39] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip. *Computers and Digital Techniques, IEE Proceedings* -, 150(5), Sep. 2003.

[40] L.W. Schruben, H. Singh, and L. Tierney. Optimal test for initialization bias in simulation output. *Operations Research*, 31(6), 1983.

[41] Salman Z. Shaikh, Mischa Schwartz, and Ted H. Szymanski. Analysis, control and design of crossbar and banyan based broadband packet switches for integrated traffic. In *IEEE International Conference On Communications ICC'90 Including Supercomm Technical Sessions. SUPERCOMM ICC'90 Conference Record; Atlanta*, volume 2, New York, Apr. 1990.

[42] Z. Song, Guangsheng Ma, and D. Song. Hierarchical star: An optimal noc topology for high-performance soc design. In *Computer and Computational Sciences, 2008. IMSCCS '08. International Multisymposiums on*, Okt. 2008.

[43] Diomidis Spinellis. Drawing tools. *Software, IEEE*, 26(3), Mai 2009.

[44] Thomas H. Theimer, Erwin P. Rathgeb, and Manfred N. Huber. Performance analysis of buffered banyan networks. *IEEE Transactions on Communications*, 39(2), Feb. 1991.

[45] D. Tutsch and M. Malek. Comparison of networks-on-chip topologies for multicore systems considering multicast and local traffic. In *2nd International Conference on Simulation Tools and Techniques, Rome, Italy, März 2009.*, März 2009.

[46] Dietmar Tutsch. *Verfahren zur Leistungsbewertung von gepufferten mehrstufigen Verbindungsnetzwerken*. Dissertation, Technische Universität Berlin, 1998.

[47] Dietmar Tutsch. *Performance Analysis of Network Architectures*. Springer Verlag, Berlin, 1 edition, 2006.

[48] P. Wielage, E.J. Marinissen, M. Altheimer, and C. Wouters. Design and dft of a high-speed area-efficient embedded asynchronous fifo. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, Apr. 2007.

[49] Xilinx. *Spartan 3E Family Data Sheet*. USA, Aug. 2009.

[50] Hyunsoo Yoon, Kyungsook Y. Lee, and Ming T. Liu. Performance analysis of multibuffered packet–switching networks in multiprocessor systems. *IEEE Transactions on Computers*, 39(3), 1990.

[51] Yue Zhang, C.R. King, J. Zaveri, Yoon Jo Kim, V. Sahu, Y. Joshi, and M.S. Bakir. Coupled electrical and thermal 3d ic centric microfluidic heat sink design and technology. In *Electronic Components and Technology Conference (ECTC), 2011 IEEE 61st*, 2011.