RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

Christian Gutmann

Implementation einer Gigabit-Ethernet-Schnittstelle zum Betrieb eines Künstlichen Neuronalen Netzwerkes

Diplomarbeit

HD-KIP 07-08

KIRCHHOFF-INSTITUT FÜR PHYSIK

Fakultät für Physik und Astronomie
Ruprecht-Karls-Universität Heidelberg

Diplomarbeit
im Studiengang Physik
vorgelegt von
Christian Gutmann
aus Müllheim
2007

Implementation einer Gigabit-Ethernet-Schnittstelle zum Betrieb eines Künstlichen Neuronalen Netzwerkes

Die Diplomarbeit wurde von
Christian Gutmann ausgeführt am
Kirchhoff-Institut für Physik
unter der Betreuung von
Herrn Prof. Dr. Karlheinz Meier

Implementation einer Gigabit-Ethernet-Schnittstelle zum Betrieb eines Künstlichen Neuronalen Netzwerkes

Diese Arbeit beschreibt die Implementation und den Test einer Gigabit-Ethernet-Schnittstelle. Die Schnittstelle findet Verwendung bei der Übertragung von Trainingsdaten und zur Steuerung einer Plattform für künstliche neuronale Netze. Ausgehend von einer Platine mit Phy (Physikalische Netzwerkschicht) für Gigabit-Ethernet sowie einem Ethernet-Core für bis zu 100-Mbit/s wurde ein Gigabit-Ethernet-MAC (Sicherungsschicht des Netzwerkes) realisiert. Gleichzeitig wurden die Fähigkeiten des Cores verbessert und um Automatismen zum einfacheren Betrieb erweitert. Der Gigabit-Ethernet-MAC wurde in programmierbarer Logik in den Hardware-Beschreibungssprachen Verilog HDL und VHDL erstellt. Simulationen haben die erfolgreiche Implementation von Gigabit-Ethernet nachgewiesen. Erfolgreiche Kommunikationstests mit einer handelsüblichen Gigabit-Ethernet-Netzwerkkarte dienen als Beleg für die Einhaltung des Standards. Der fertige Gigabit-Ethernet-MAC kommt zusammen mit dem Phy auf zwei verschiedenen Plattformen für künstliche neuronale Netze zum Einsatz. Die Verwendung des Gigabit-Ethernet-MACs ist jedoch nicht auf die momentane Plattform beschränkt, sondern er lässt sich überall einsetzen, wo programmierbare Logik zur Verfügung steht und Gigabit-Ethernet-Kommunikation benötigt wird.

Implementation of a Gigabit Ethernet Interface to operate an Artificial Neural Network

This thesis describes the implementation and test of a Gigabit Ethernet interface. It is used to transmit training data and to operate a platform for artificial neural networks. A board with a Phy (physical network layer) for Gigabit Ethernet and an Ethernet core for 100 Mbit/s have been enhanced in this thesis: a Gigabit Ethernet MAC (media access control) has been implemented, the features of the core have been improved and have been extended with automatisms for easier handling. The Gigabit Ethernet MAC has been implemented in programmable logic using the Hardware Description Languages Verilog HDL and VHDL. Successful implementation of Gigabit Ethernet has been verified in simulation. Communication tests with a customary Gigabit Ethernet network interface card has been successfully used as a reference to check the compliance of the standards. The final version of the Gigabit Ethernet MAC is used together with the Phy on two different artificial neural network platforms. The application of the Gigabit Ethernet MAC is not limited to the presented platform but is suitable for any system where programmable logic is available and Gigabit Ethernet communication is desired.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung und Motivation

Das Gehirn ist eine der komplexesten Strukturen, die bekannt sind. Es besteht aus einem engen Geflecht aus Neuronen und Synapsen. Diese Komplexität macht es zu einem der leistungsfähigsten informationsverabeitenden Systeme. Um das Gehirn oder allgemeiner *Neuronale Netze* zu verstehen, wird die Funktionsweise von Neuronen und Synapsen nachgebildet.

Ein Beispiel für die Forschung an Neuronalen Netzen ist das FACETS-Projekt [10], welches von der EU unterstützt wird. Es ist ein Forschungsprojekt mit dem Ziel, biologisch inspirierte Informationssysteme auf der Grundlage des Gehirns zu untersuchen. Das Projekt ist in drei Gebiete aufgeteilt: Biologische Experimente, Computer-basierte Modelle und *Künstliche Neuronale Netze* in Hardware. Diese Künstlichen Neuronalen Netze sind Forschungsgegenstand der *Electronic Vision(s)*-Arbeitsgruppe von Prof. Meier in Heidelberg.

Die Electronic Vision(s)-Gruppe hat zu diesem Zweck mehrere *ANN*[1] *ASICs*[2] entwickelt, die jeweils die Funktionsweise von mehreren hundert einzelnen Neuronen nachahmen. Ein ANN ASIC wird dabei auf dem sogenannten Netzwerkmodul *Nathan* [12] betrieben. Um große Netze mit mehreren tausend Neuronen untersuchen zu können, wurde ein vernetztes System aus Nathan-Modulen aufgebaut. Die einzelnen Module sind dabei über eine *Plattform für parallel arbeitende Neuronale Netze* [11], mittels einer *Backplane*, verbunden.

Dieses System muss für Experimente mit den künstlichen neuronalen Netzen mit großen Datenmengen versorgt werden. Zur Auswertung der Ergebnisse müssen wiederum viele Daten aus dem System ausgelesen werden. Zur Bewältigung des hohen Datenaufkommens bietet sich *Ethernet* an. Diese Netzwerktechnologie in der Variante *Gigabit-Ethernet* bewerkstelligt die Aufgabe der schnellen Datenübertragung in Verbindung mit einem gegen Fehler gesicherten Übertragungsweg. Eine Gigabit-Ethernet-Schnittstelle besteht aus zwei Komponenten: Der Hardware zur Verbindung mit dem Übertragungsmedium und der Steuerlogik.

In der vorliegenden Arbeit wird die Realisierung einer Gigabit-Ethernet-Schnittstelle präsentiert. Grundlage bilden ein erworbenes Hardware-Modul sowie ein *IP*[3]*-Core*. Letzter liegt in Form einer Hardware-Beschreibung zur Implementation auf einem FPGA[4] vor. Es wird beschrieben, wie der IP-Core, der zunächst nur den langsameren 100-Mbit/s-Standard unterstüzt hat, auf den Gigabit-Ethernet-Betrieb erweitert wird. Der veränderte IP-Core wird simuliert und anschließend zusammen mit dem Hardware-Modul in das System integriert und kann sowohl auf einem Nathan-Modul als auch auf der Backplane eingesetzt werden. Die Zuverlässigkeit der Schnittstelle wird schließlich durch Datenübertragungen zwischen PC und System getestet.

[1] **A**rtificial **N**eural **N**etwork (Künstliches Neuronales Netzwerk)
[2] **A**pplication **S**pecific **I**ntegrated **C**ircuit (Anwendungsspezifische Integrierte Schaltung)
[3] **I**ntellectual **P**roperty
[4] **F**ield **P**rogrammable **G**ate **A**rray

Gliederung der Arbeit

Die vorliegende Arbeit ist wie folgt aufgebaut: Direkt im Anschluss an dieses Kapitel werden die Grundlagen zum Verständnis der Arbeit erläutert (Kapitel 2). Anschließend folgt eine Beschreibung des Systems, auf dem die Gigabit-Ethernet-Schnittstelle implementiert wurde (Kapitel 3). Nach einer Präsentation der Schwierigkeiten, die in dieser Arbeit gelöst wurden (Kapitel 4), folgt eine Darstellung der beiden Ausgangskomponenten, d.h. des Hardware-Moduls und des IP-Cores für FPGA (Kapitel 5 und 6). Die Erweiterung des IP-Cores auf eine höhere Geschwindigkeit wird im Anschluss behandelt (Kapitel 7). Nachfolgend sind die zum Betrieb notwendigen Implementationen wie die Integration in das bestehende FPGA-Design und die Erstellung der Software für den Zugriff auf die Schnittstelle vom PC aus beschrieben (Kapitel 8 und 9). Die Inbetriebnahme, die dabei aufgetretenen Schwierigkeiten und die Ergebnisse von Tests mit Gigabit-Ethernet sind in Kapitel 10 beschrieben. Abschließend wird die Arbeit und deren erzielte Resultate noch einmal zusammengefasst dargestellt und ein Ausblick über zukünftige Entwicklungsmöglichkeiten gegeben (Kapitel 11).

2 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen behandelt, die zum Verständnis der Arbeit nötig sind. Zunächst werden Netzwerke allgemein erläutert (Abschnitt 2.1), danach erfolgt eine Beschreibung des OSI[1]-Referenzmodells (Abschnitt 2.2) und des Ethernets (Abschnitt 2.3). Schließlich wird noch auf FPGAs eingegangen (Abschnitt 2.4), denn darauf wird ein Teil der Ethernet-Schnittstelle implementiert werden.

2.1 Netzwerke

Ein Netzwerk wird in *Knoten* und *Verbindungsleitungen* unterteilt. Bei den Knoten unterscheidet man zwischen solchen, die das Netzwerk benutzen (*Host*) und denjenigen, die der Weiterleitung dienen (*Switch*). Dies ist in Abbildung 2.1 dargestellt. Der gesamte Aufbau und die Struktur eines Netzwerkes werden als *Netzwerkarchitektur* bezeichnet [26]. Um dessen Komplexität überschaubarer zu machen, hat man verschiedene Abstraktionsebenen eingeführt, die man als *Schichten* (*Layer*) bezeichnet. Diese sind hierarchisch strukturiert. In jeder Schicht existieren *Dienste*, die bestimmte Aufgaben erfüllen. Sie nutzen dazu die von den niedrigeren Schichten bereitgestellten Dienste. Die Dienste werden auch als *Protokolle* bezeichnet, von denen verschiedene in einer Schicht vorhanden sein können.

Die Protokolle besitzen nun zwei unterschiedliche Schnittstellen, eine *Dienst-Schnittstelle* und eine *Partner-Schnittstelle* (*Peer Interface*). Die Dienst-Schnittstelle stellt die Verbindung zu den anderen Schichten desselben Hosts dar, die dieses Protokoll benutzen, während die Partner-Schnittstelle die Verbindung zur gleichen Schicht auf einem anderen Host ist (vgl. Abbildung 2.2). Dabei ist zu betonen, dass das für die Protokolle in jeder Schicht gilt, wobei in höheren Schichten die scheinbare Verbindung zum Partner über die niedrigeren Schichten stattfindet. Die Vorteile einer solchen modularen Betrachtungsweise bestehen darin, dass die Implementation der Protokolle nicht fest vorgegeben ist, sondern beliebig verändert werden

[1] **O**pen **S**ystems **I**nterconnection

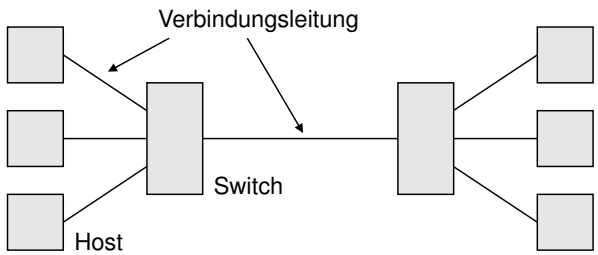


Abbildung 2.1: Beispiel einer Netzwerkarchitektur

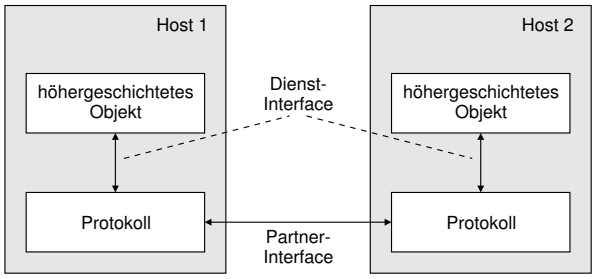


Abbildung 2.2: Schnittstellen der Protokolle einer Netzwerk-Schicht: Dienst- und Partner-Schnittstelle (Quelle: [26])

kann, solange die Spezifikationen der Schnittstellen eingehalten werden. Außerdem ist dadurch das Hinzufügen von weiteren Protokollen leicht möglich. Darüber hinaus arbeiten die einzelnen Protokolle unabhängig voneinander, was bei der Datenübertragung zwischen zwei Hosts deutlich wird: Beim Versenden werden die Daten der höheren Schichten durch ein tieferliegendes Protokoll gekapselt, indem es ihnen eigene Informationen hinzufügt; die eigentlichen *Nutzdaten* bleiben dabei unbeachtet. Im Empfänger entfernen tieferliegende Protokolle diese Zusatzinformationen wieder und reichen die Daten über das Dienst-Interface nach oben weiter. Für die höheren Schichten scheint es so, als hätten sie direkt über das Partner-Interface miteinander kommuniziert, obwohl die Daten vom Dienst-Interface übermittelt wurden.

2.2 OSI-Referenzmodell

Ein häufig verwendetes Modell zur formalen Beschreibung einer Netzwerkarchitektur ist das *OSI-Referenzmodell* [40]. Es unterteilt die Funktionalität eines Netzwerkes in sieben Schichten und spezifiziert deren Aufgaben. Diese Aufgaben werden durch ein oder mehrere Protokolle implementiert. Da es sich bei dem Modell um keinen strikten Standard handelt und in der Anwendung meist ein abgewandeltes Modell eingesetzt wird, werde ich im Folgenden nur die wichtigsten beschreiben und die Schichten fünf und sechs nicht erläutern [36]. In Abbildung 2.3 ist der Vollständigkeit halber das komplette OSI-Referenzmodell dargestellt.

Die Schichten des OSI-Modells von unten nach oben sind:

1. Die *Bitübertragungsschicht* (*Physical Layer*) überträgt einzelne *Bits* und ist direkt mit dem physikalischen Übertragungsmedium verbunden. Die Realisierung dieser Schicht heißt *Phy* und ist Bestandteil jeder Netzwerk-Karte. Der Phy ist ein Transceiver, d.h. er dient dem Empfangen und Senden von Daten. Dazu übersetzt er die Daten-Signale in Signale für das entsprechende Medium, weshalb für jedes Medium ein anderer Phy zum Einsatz kommt. Die Signale müssen speziell kodiert sein, um mehrere Anforderungen zu erfüllen. So muss z.B. die Taktfrequenz aus dem übertragenen Signal gewonnen werden. Wie dies technisch realisiert wird, wird in Abschnitt 2.3.1 erläutert.
2. Die *Sicherungsschicht* (*Data Link Layer*) ist für die Kommunikation zwischen zwei direkt verbundenen Hosts zuständig und kontrolliert den Zugriff auf das Medium. Das ist die Aufgabe des sogenannten *MAC*[2]-Protokolls. Der MAC reguliert den Zugriff auf das

[2]Media Access Control

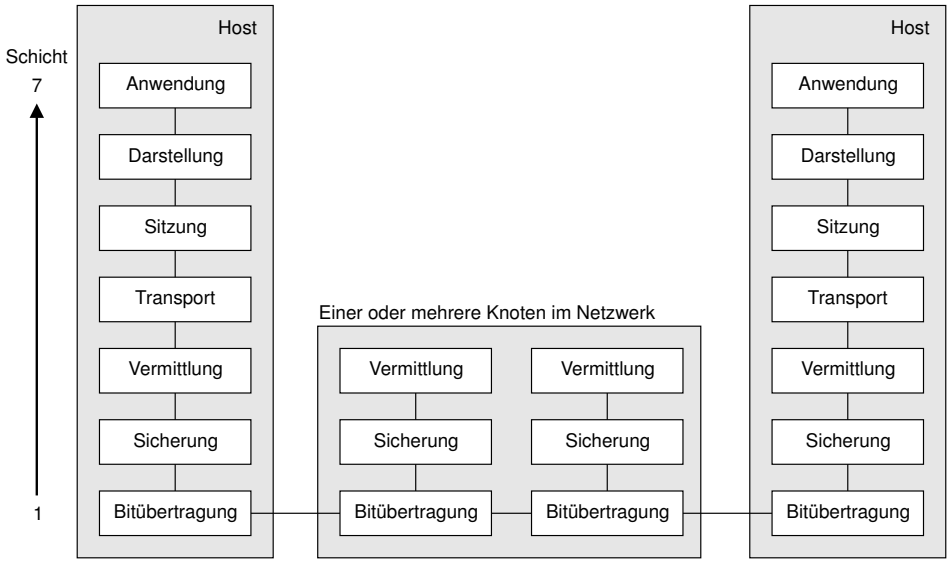


Abbildung 2.3: OSI-Referenzmodell mit einer Netzwerk-Verbindung über mehrere Knoten (Quelle: [26])

Übertragungsmedium und sichert die Übertragung dadurch, dass er aufgetretene Fehler detektiert. Höhere Schichten erhalten so nur verifizierte Daten, fehlerhafte werden verworfen. Dies erreicht er, indem er die Daten aus der nächsthöheren Schicht in sogenannte *Frames* kapselt (vgl. Abschnitt 2.3.3). Außerdem fasst er die in Schicht 1 vom Phy empfangenen Bits zu Frames zusammen und reicht deren Nutzdaten als Paket nach oben weiter.

3. Die *Vermittlungsschicht* (*Network Layer*) steuert die Weiterleitung der Daten über ggf. notwendige Knoten zwischen den Endpunkten einer Datenübertragung (*Routing*).
4. Die *Transportschicht* (*Transport Layer*) und die höheren Schichten sind nur in den Endpunkten einer Verbindung vorhanden. Sie dient als Schnittstelle zwischen den darunterliegenden verbindungs- und den darüberliegenden anwendungsorientierten Schichten

   ⋮
7. Die *Anwendungsschicht* (*Application Layer*) ist die oberste Schicht und stellt Dienste für Anwendungen zur Verfügung. Anwendungen selbst sind nicht Teil des OSI-Modells, sondern liegen darüber.

2.3 Ethernet

Das *Ethernet* ist die am weitesten verbreitete Netzwerktechnologie. Die Standards hierfür wurden von dem *IEEE*[3] unter der Bezeichnung 802.3 [15] festgelegt. Ethernet ist auf den beiden untersten Schichten des OSI-Referenzmodells angesiedelt, wobei der MAC in der oberen Schicht die vom Phy in der unteren Schicht bereitgestellte Funktionalität benutzt. Ganz

[3] Institute of Electrical and Electronics Engineers

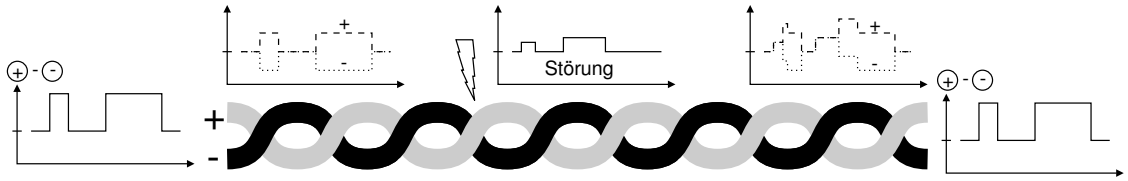


Abbildung 2.4: Adernpaar eines Twisted-Pair-Kabels mit Störung

allgemein sind unter IEEE 802 verschiedene Normen für lokale Netze (*LAN*[4]) zusammengefasst, deren wichtigste Vertreter Ethernet (802.3), Token-Ring (802.5), WLAN (802.11) und Bluetooth(802.15) sind. In IEEE 802.3 enthalten sind unter anderem das Frame-Format, die verschiedenen Übertragungsmedien und -geschwindigkeiten sowie die Kodierung der Daten durch den Phy.

2.3.1 Bitübertragungsschicht – Phy

Wie in Abschnitt 2.2 schon erwähnt wurde, existieren mehrere mögliche Übertragungsmedien. Für Ethernet sind dies *Monomode-* und *Multimode-Glasfaserkabel* sowie *Koaxial-*, *Twinax-* und *Twisted-Pair-Kupferkabel*. Letzteres besteht aus acht Kabeladern, wovon jeweils zwei miteinander verdrillt sind und über die ein Signal jeweils *differentiell* übertragen wird. Dies reduziert die Abstrahlung auf andere Adern und verringert die Störanfälligkeit (vgl. Abbildung 2.4). Außerdem gibt es noch verschiedene Übertragungsgeschwindigkeiten: 10 Mbit/s, 100 Mbit/s (*Fast-Ethernet*), 1000 Mbit/s (*Gigabit-Ethernet*) und 10 Gbit/s (*10-Gigabit-Ethernet*). Für alle verwendeteten Kombinationen von Medium und Geschwindigkeit existiert eine eindeutige Bezeichnung der Art *10Base2* (das früher weit verbreitete 10-Mbit/s-Ethernet über Koaxial-Kabel) oder *100Base-TX* (der heutige Standard der Heimanwendung von Fast-Ethernet über *Cat-5-Kabel*). Ein Cat-5-Kabel ist ein Twisted-Pair-Kupferkabel, das speziell für 100Mbit/s-Ethernet ausgelegt ist, jedoch auch für Gigabit-Ethernet verwendet werden kann. Die Kombinationen sind zum Teil noch einmal in der IEEE 802.3 zusammengefasst und spezifiziert, z.B. 802.3u für 100Base-T und 802.3z für 1000Base-X (Gigabit-Ethernet über Glasfaser- und Twinax-Kabel). Tabelle 2.1 zeigt eine Übersicht über die wichtigsten Übertragungsarten. Für jede Betriebsart existiert zusätzlich noch die Unterscheidung zwischen *Voll-* und *Halbduplex-Modus*. Im Halbduplex-Modus ist die Netzwerk-Karte zu einem bestimmten Zeitpunkt nur in der Lage, entweder zu Senden oder zu Empfangen. Diese Einschränkung ist im Vollduplex-Modus aufgehoben, dort ist gleichzeitiges Senden und Empfangen möglich.

Zu jeder der genannten der genannten Übertragungsarten werden bestimmte Anforderungen an die Signalübertragung gestellt, die durch den Phy erfüllt werden müssen. Aufgrund der hier dargestellten Menge von Varianten werde ich im Folgenden nur die Technik des von mir verwendeten *1000Base-T* beschreiben, also Gigabit-Ethernet über Cat-5-Kabel, das in IEEE 802.3ab spezifiziert ist [15]. Damit Gigabit-Ethernet in dieser Variante funktioniert, müssen verschiedene technische Probleme wie Dämpfung und Übersprechen der Signale gelöst werden. Hinzu kommen noch EMV[5]-Richtlinien, die eingehalten werden müssen. Ein wichtiger

[4] Local Area Network
[5] Elektromagnetische Verträglichkeit

Geschwindigkeit (Mbit/s)	IEEE-Standard	Jahr	Bezeichnung	Medium
1000	802.3ab	1999	1000Base-T	Twisted-Pair-Kabel
	802.3z	1998	1000Base-SX	Glasfaser
			1000Base-LX	Glasfaser
			1000Base-CX	Twinax-Kabel
100	802.3u	1995	100Base-T	Twisted-Pair-Kabel
			100Base-FX	Glasfaser
10	802.3j	1993	10Base-F	Glasfaser
	802.3i	1990	10Base-T	Twisted-Pair-Kabel
	802.3a	1988	10Base2	Koaxial-Kabel

Tabelle 2.1: Auflistung einiger Ethernet-Übertragungsarten

Schritt, um überhaupt ein Cat-5-Kabel nutzen zu können, ist dabei die Reduktion der Daten-Übertragungsrate auf 125 Mbit/s, denn das ist die Geschwindigkeit, für die dieses Kabel spezifiziert ist. Dazu verzichtet man auf die sonst bei Gigabit-Ethernet übliche *8B/10B*-Kodierung, d.h. 8 Bits an Daten werden mit 10 Bits auf den Verbindungsleitungen kodiert, weil sich dadurch die effektive Übertragungsrate zusätzlich auf 1250 Mbit/s erhöht. Die 1000 Mbit/s verteilt man nun auf die vier Adernpaare des Cat-5-Kabels und erreicht damit eine Übertragungsrate von 250 Mbit/s. Diese lässt sich nochmals halbieren, indem man je zwei Daten-Bits zu einem *Symbol* zusammenfasst und mittels einer 5-Level-Puls-Amplituden-Modulation, *4D-PAM5* genannt, in einem Takt überträgt [29]. So erzielt man 125 Mbit/s bzw. 8 Bits pro 125-MHz-Takt.

Die 4D-PAM5-Kodierung, die im Phy stattfindet, liefert $5^4 (= 625)$ Symbole für $2^8 (= 256)$ mögliche Datenwörter. Mit Hilfe der überzähligen Symbole kann man redundante Codegruppen bilden, um den Signalabstand zu erhöhen und damit die Störanfälligkeit zu reduzieren. Die Auswahl der jeweils zu verwendenden Codegruppe erfolgt über ein festes Schema, das *Trellis-Verfahren*. Das *Scrambling-Verfahren* sorgt dann dafür, dass das Frequenzspektrum der Übertragung gleichmäßig verteilt ist, was die elektromagnetische Abtrahlung verringert. Dies wird erreicht, indem der Datenstrom mit einem zufälligen Datenwort verknüpft wird. Außerdem verhindert es zu lange Folgen von Nullen oder Einsen, wodurch die Taktrückgewinnung durch den Phy des Empfängers gestört würde.

Um trotz der Nutzung aller vier Adernpaare Daten im Vollduplex-Modus empfangen zu können, müssen die gesendeten Daten aus der Überlagerung von empfangenen und gesendeten Daten herausgefiltert werden. Das *Echo-Cancellation-Verfahren* verwendet dafür *Hybride*, die nach dem Schema einer Brückenschaltung aufgebaut sind, um Störungen (*Echo*) wie die Sende-Daten aus dem Signal zu entfernen. Für den Vollduplex-Betrieb bedarf es zusätzlich noch einer Möglichkeit, den Datenstrom bei Problemen auf Seiten eines Hosts reduzieren bzw. stoppen zu können. Man spricht dabei von *Fluss-Kontrolle* (*Flow Control*). Es wird ein spezielles *Pausen-Frame* versendet (vgl. Abschnitt 2.3.3), um dem Gegenüber mitzuteilen, dass für bestimmte Zeit keine weiteren Frames angenommen werden können. Daraufhin unterbricht dieser das Senden.

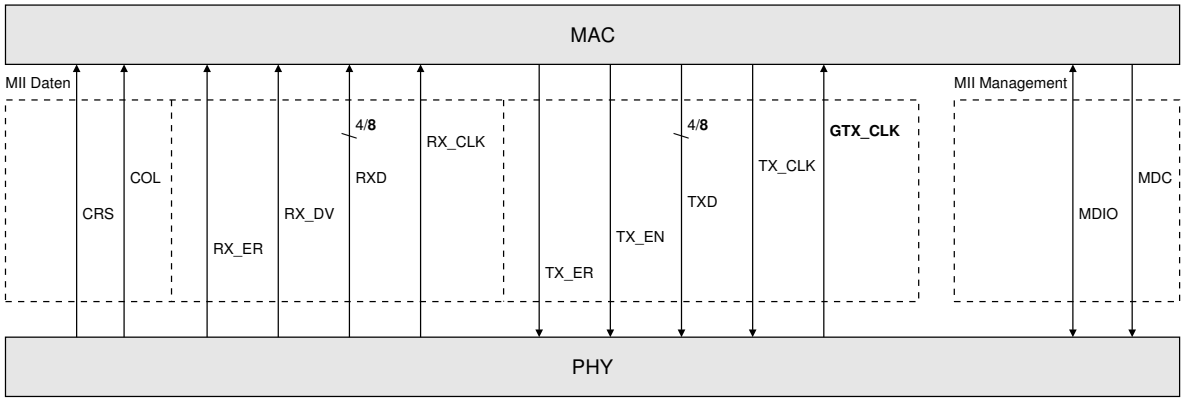


Abbildung 2.5: Signale der MII und GMII Schnittstelle (TX_CLK wird bei Gigabit-Ethernet durch GTX_CLK ersetzt)

2.3.2 Verbindung zwischen Phy und MAC – MII[6]

Bei Ethernet sind Phy und MAC über das sogenannte *Media Independent Interface* verbunden, das in IEEE 802.3u spezifiziert ist. Es wird bei 10-Mbit/s-Ethernet und 100-Mbit/s-Ethernet verwendet und trägt seinen Namen, weil es unabhängig vom verwendeten Medientyp dasselbe Dienst-Interface zwischen Layer 1 und Layer 2 des OSI-Referenzmodells bereitstellt (vgl. Abschnitt 2.2).

Diese Schnittstelle ist in zwei Bereiche gegliedert: das *MII Management-Interface* und das *MII Daten-Interface* (vgl. Abbildung 2.5). Ersteres dient zur Steuerung des Phys über dessen Status- und Kontroll-Register und besteht aus einer Takt-Leitung zum Phy (*MDC*, Management Data Clock) sowie einer bidirektionalen seriellen Daten-Leitung (*MDIO*, Management Data Input/Output). Über diese können die Register beschrieben und gelesen werden.

Das Daten-Interface lässt sich wiederum in drei Bereiche unterteilen: Sende-, Empfangs- und Statussignale. Um zu Senden, benötigt der Phy einen Takt (*TX_CLK*, Transmit Clock), der je nach Übertragungsrate unterschiedlich schnell ist (2,5 Mhz bei 10 Mbit/s, 25 MHz bei 100 Mbit/s, 125 MHz bei 1000 Mbit/s). Mit diesem werden die vier parallelen Sende-Datenleitungen (*TXD0* bis *TXD3*, Transmit Data) und das Signal, das die Gültigkeit der Daten angibt (*TX_EN*, Transmit Enable), getaktet. Ein weiteres Signal (*TX_ER*, Transmit Error) veranlasst den Phy dazu, ungültige Signale zu übertragen. Auf Empfangsseite übermittelt der Phy den Takt (*RX_CLK*, Receive Clock), den er aus den empfangenen Daten rekonstruiert hat, an den MAC. Damit werden die vier Empfangs-Datenleitungen (*RXD0* bis *RXD3*, Receive Data) und das Signal (*RX_ER*, Receive Error), das Empfangsfehler angibt, getaktet. Synchron zu den Daten wird deren Gültigkeit übertragen (*RX_DV*, Receive Data Valid). Die Schnittstelle besitzt noch zwei weitere Signalleitungen zur Statusübertragung. Ein Signal zeigt im Halbduplex-Modus dem MAC die Kollision von Daten auf dem Kabel an (*COL*, Collision Detect), d.h. dass gleichzeitig Daten gesendet und empfangen werden. Im Vollduplex-Modus ist dieses Signal immer `0`. Ein Zweites signalisiert eine stattfindende Übertragung (*CRS*, Carrier Sense).

[6] Media Independent Interface

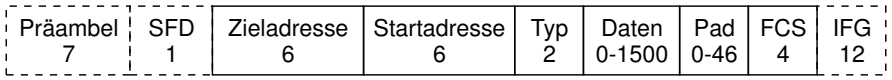


Abbildung 2.6: Ethernet-Frame mit Byte-Längen der Felder bei IEEE 802.3 (Länge des Frames 64 – 1518 Bytes)

Für Gigabit-Ethernet heißt die Schnittstelle *Gigabit MII* (*GMII*) und ist in IEEE 802.3z spezifiziert. Sie unterscheidet sich im Wesentlichen dadurch von dem MII, dass jeweils acht statt vier Bits an Daten übertragen werden. Außerdem ist die Takt-Leitung TX_CLK durch *GTX_CLK* (Gigabit Transmit Clock), die eine höhere Frequenz besitzt (125 Mhz), ersetzt worden. Es existiert eine Variante dieses Interfaces, das *Reduced GMII* (*RGMII*). Bei ihr werden, um die Anzahl an Pins zu verringern, verschiedene Signale miteinander verknüpft und die Daten werden sowohl zur steigenden als auch zur fallenden Flanke des Taktes mit vier Bits parallel übertragen.

2.3.3 Sicherungsschicht – MAC

Im Folgenden wird die Kapselung der Daten in Frames durch den Ethernet-MAC beschrieben, um die Datenübertragung zu sichern (vgl. Abschnitt 2.2). In Abbildung 2.6 ist ein Frame, wie er bei Ethernet verwendet wird, mit den zusätzlich zur Übertragung notwendigen Feldern abgebildet [28]. Im Einzelnen sind dies folgende:

Präambel: Das Bit-Muster `10101010` wird wiederholt, damit das Taktsignal des Empfängers sich mit dem des Senders synchronisieren kann.

Start Frame Delimiter (SFD): Dieses Feld entspricht der Präambel, mit Ausnahme des bei der Übertragung letzten Bits, das ebenfalls `1` ist. Mit diesem Byte wird der Anfang des Frames gekennzeichnet.

Zieladresse: Die Empfänger-Netzwerkkarte wird über diese Adresse identifiziert.

Startadresse: Dies ist die Adresse der Absender-Netzwerkkarte.

Typ: Dieses Feld gibt das darüberliegende Protokoll an, dem die Daten angehören [16]. Es existiert auch eine Frame-Variante, in der stattdessen die Anzahl der Daten-Bytes eingetragen ist. Das ist unproblematisch, da diese auf 1500 Bytes beschränkt ist und alle Typenbezeichnungen einen höheren Wert besitzen.

Daten: Hier befinden sich die Nutzdaten der höheren Schicht. Da die Maximallänge eines Frames 1518 Bytes beträgt, können darin höchstens 1500 Bytes an Daten enthalten sein.

Pad: Jedes Frame muss eine Mindestlänge von 64 Bytes besitzen, damit Kollisionen von mehreren Frames auf dem Medium erkannt werden können. Aus diesem Grund werden im Falle einer geringeren Anzahl von Daten bis zu 46 Bytes an Nullen aufgefüllt.

Frame Check Sequence (FCS): Am Ende jeden Frames steht die CRC[7]-32-Prüfsumme, anhand derer der Empfänger Fehler bei der Übertragung feststellen kann [36]. Die CRC-

[7] Cyclic Redundancy Check

Zieladresse oder 01:80:c2:00:00:01	Startadresse	Typ 8808	Opcode 0001	Pausen Zähler	reserviert	FCS
6	6	2	2	2	42	4

Abbildung 2.7: Pausen-Frame mit Byte-Längen der Felder (Länge: 64 Bytes)

32-Prüfsumme beruht auf der Polynomdivision mit

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1\,.$$

Inter Frame Gap (IFG): Zwischen den einzelnen Frames muss eine Pause von der Länge liegen, die eine Übertragung von 96 Bits bei der entsprechenden Geschwindigkeit benötigen würde. In dieser Zeit kann sich der Empfänger auf das nächste Frame vorbereiten.

Die ersten beiden Felder sowie das letzte werden, wie in Abbildung 2.6 durch die gestrichelten Felder angedeutet, nicht dem Ethernet-Frame zugerechnet, sie werden jedoch für eine Datenübertragung mit benötigt. Die Felder Zieladresse, Startadresse und Typ werden auch unter dem Begriff *Header* zusammengefasst. Es ist noch wichtig zu erwähnen, dass alle Felder in der Byte-Folge ihrer Notation übertragen werden, innerhalb eines Bytes wird jedoch mit dem niedrigstwertigen Bit (*LSB*[8]) begonnen. Einzige Ausnahme stellt die FCS dar, dort wird das höchstwertige Bit (*MSB*[9]) zuerst gesendet [15].

Die Adressen im Ethernet-Frame werden als *MAC-Adressen* bezeichnet. Sie bestehen aus einem 24-Bit-Präfix, der dem Hersteller der Netzwerkkarte zugeordnet ist, die restlichen Bits der MAC-Adresse vergibt der Hersteller. Diese Zuordnung ist weltweit eindeutig [17]. Die Notation erfolgt in Byte-Blöcken in hexadezimaler Schreibweise, getrennt durch Doppelpunkte, wie z.B. `00:15:E9:F4:F0:D2`. Neben sogenannten *Unicast*-Adressen, die je an einen einzelnen Host adressiert sind, gibt es noch spezielle MAC-Adressen, um Daten an eine Gruppe von Netzwerk-Karten im lokalen Netz zu senden. Diese werden durch Setzen des LSBs des ersten Bytes gekennzeichnet und heißen *Multicast*-Adressen. Eine nur aus Einsen bestehende Adresse (`FF:FF:FF:FF:FF:FF`) ist eine *Broadcast*-Adresse und ist an alle angeschlossenen Netzwerk-Karten adressiert. Zusätzlich existieren auch noch reservierte Adressen. Dazu gehört z.B. `01:80:c2:00:00:01`, die als Zieladresse bei Pausen-Frames verwendet wird (vgl. Abschnitt 2.3.1).

Es unterscheidet sich von einem normalen Frame dadurch, dass es die reservierte Zieladresse besitzt, eine für Kontroll-Frames spezifische Typen-Bezeichnung (`8808`) und danach den *Opcode* `0001`. Der Wert der darauffolgenden zwei Bytes gibt die Länge der Pause in Vielfachen einer 64-Byte-Übertragung an. Der Rest des Frames entspricht einem normalen Frame, das auf die Mindestlänge von 64 Bytes aufgefüllt wurde. Statt der reservierten Zieladresse besteht auch die Möglichkeit, die Unicast-Adresse des Empfängers anzugeben.

Eine Netzwerk-Karte empfängt immer alle Frames, die über das mit ihr verbundene Medium übertragen werden. Dann prüft sie, um welche Art von Zieladresse es sich handelt und leitet die Daten nur dann an höhere Schichten weiter, wenn diese an sie adressiert sind. Das ist der normale Betriebsmodus einer Netzwerk-Karte. Sie kann jedoch auch in den *Promiscuous Mode*

[8] **L**east **S**ignificant **B**it
[9] **M**ost **S**ignificant **B**it

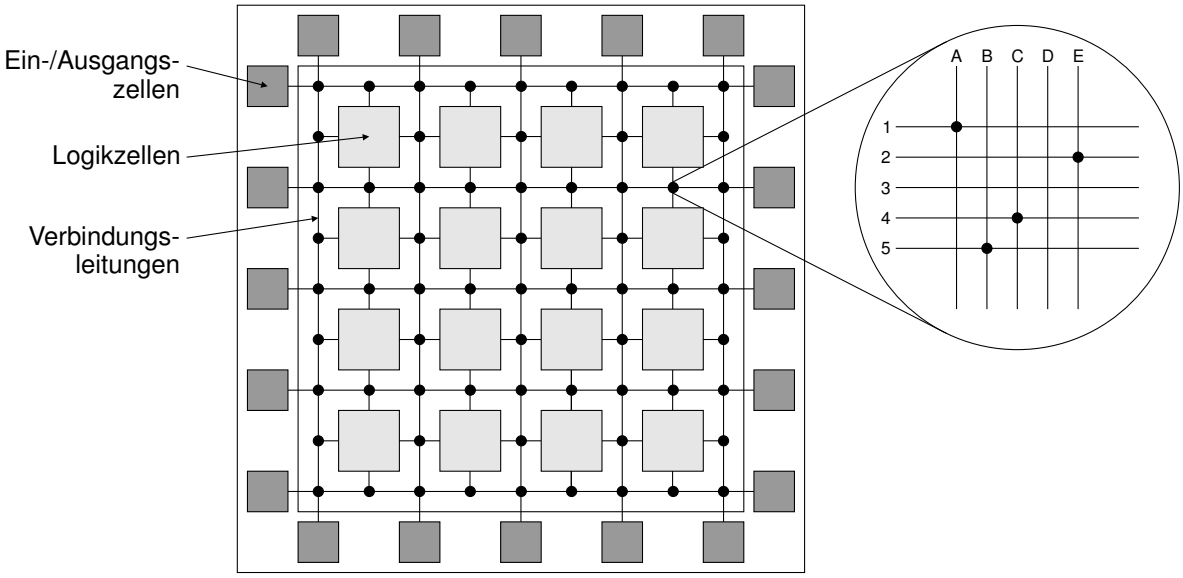


Abbildung 2.8: Schematischer Aufbau eines FPGAs. Durch die Konfiguration wurden Verbindungen der Leitungen A1, B5, C4 und E2 hergestellt (vergrößerter Ausschnitt)

versetzt werden, dann übermittelt sie alle Daten ungeachtet der Zieladresse an die höheren Schichten.

2.4 FPGA

Ein FPGA ist im Gegensatz zu einem ASIC ein programmierbarer Halbleiter, weshalb er sich jederzeit veränderten Anforderungen anpassen und neu konfigurieren lässt. Nachfolgend wird seine Grundstruktur und der Weg bis zu seiner Konfiguration, der sogenannte *Design Flow*, erklärt.

2.4.1 Aufbau

In Abbildung 2.8 ist der allgemeine Aufbau eines FPGAs schematisch dargestellt. Er besteht allgemein aus identisch aufgebauten Logikzellen, die in einer Matrix angeordnet sind. Kombinatorische und synchrone Logik lässt sich dadurch auf den FPGA abbilden, dass vorhandene Leitungen zwischen und innerhalb der Logikzellen eine nahezu beliebige Verschaltung zulassen. Zusätzlich zu diesen Leitungen existieren noch besondere globale Leitungen, die vor allem für die Verteilung von Takt und Reset eingesetzt werden. Die Anbindung des FPGAs nach außen erfolgt über spezielle Ein-/Ausgangszellen, die mit den Pins des FPGAs verbunden sind. Die genaue Struktur eines FPGAs sowie die für die einzelnen Schritte verwendeten Programme hängen vom Hersteller und FPGA-Typ ab, weshalb in Abschnitt 3.4 der verwendete FPGA gesondert betrachtet wird.

2.4.2 Design Flow

Damit ein FPGA eingesetzt werden kann, müssen die entsprechenden Verbindungen und die Komponenten im FPGA konfiguriert werden. Vom Entwurf eines Designs bis zur Konfiguration sind die in den nächsten Absätzen beschriebenen Schritte notwendig [37].

2.4.2.1 Beschreibung in HDL[10]

Als erstes wird die gewünschte Funktionalität (Logik) mit Hilfe einer *HDL* beschrieben. Die am häufigsten verwendeten sind *Verilog HDL* und *VHDL*[11]. Der Unterschied zur Softwareprogrammierung besteht darin, dass zeitliche Abläufe und Gleichzeitigkeiten in der Sprache festgelegt werden können. Das bedeutet, dass die einzelnen Bestandteile der Beschreibung, die *Module* und *Prozesse*, parallel abgearbeitet werden und nicht wie bei Software nacheinander. Diese Beschreibung kann auf unterschiedlichen Abstraktionsebenen geschehen. Eine hohe Abstraktionsebene stellt die reine *Verhaltensbeschreibung* dar, die das Verhalten des Moduls widerspiegelt, jedoch nicht unbedingt auf die Hardware des FPGAs abbildbar ist. Einen niedrigen Abstraktionsgrad bildet die Beschreibung auf *Gatter-Ebene*, dabei werden nur elementare Einheiten wie Logikgatter und Flip-Flops verwendet. Als Zwischenstufe existiert das *Register-Transfer-Level*, bei dem das Verhalten durch Datenaustausch zwischen Registern in diskreten Zeitschritten dargestellt wird. Meist wird die Beschreibung der Hardware auf dieser Ebene durchgeführt und die verwendeten Programme übersetzen dies auf Gatter-Level, das sich dann auf die Struktur des FPGAs umsetzen lässt. Zusätzlich zu den selbst geschriebenen Modulen besteht die Möglichkeit, einfache Elemente aus einer vorhandenen Bibliothek einzubinden oder vorgefertigte Module mit einem für den verwendeten FPGA spezifischen Programm zu konfigurieren und einzubinden. Durch Parameter werden Elemente wie Fifos oder Speicherzellen den Anforderungen entsprechend angepasst.

2.4.2.2 Simulation

Die Verhaltensbeschreibung wird hauptsächlich eingesetzt, um ein Modul zu prüfen. Dazu schreibt man eine sogenannte *Testbench*, die die Umgebung nachbildet, in der das Modul eingesetzt wird. Die Testbench und das zu testende Modul werden dann in einen *Simulator* geladen, der die Signale für jeden einzelnen Taktschritte berechnet und das Verhalten der Signale mit ihrem zeitlichen Ablauf darstellen kann. Bei kleineren Programmteilen kann man darin Fehler erkennen, während es bei komplexeren notwendig ist, die Testbench zu automatisieren, so dass möglichst viele Fehlerquellen automatisch erkannt und entsprechende Meldungen ausgegeben werden. Die Simulation erlaubt es, die Hardware-Beschreibung in ihrer Funktionsweise zu testen, die genauen zeitlichen Abläufe bleiben aber unberücksichtigt.

[10] **H**ardware **D**escription **L**anguage
[11] **V**ery High Speed Integrated Circuit **HDL**

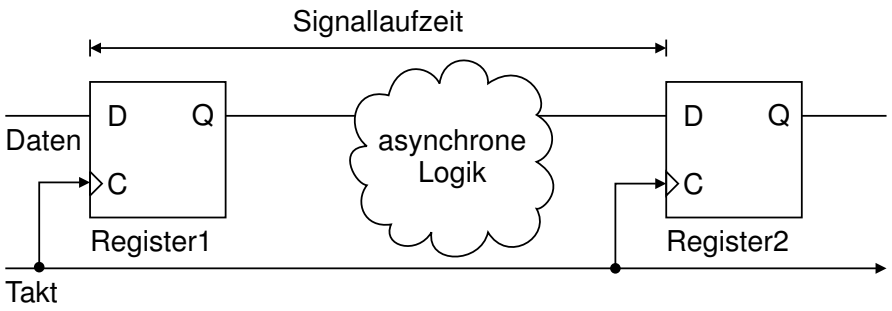


Abbildung 2.9: Signallaufzeit zwischen den Registern

2.4.2.3 Synthese und Mapping

Der nächste Schritt nach Programmierung und erfolgreicher erster Verifikation ist die *Synthese*. Ein spezielles Programm untersucht die einzelnen Teile der funktionellen Beschreibung und versucht, diese mit Standardelementen wie Addierer, Register, Zähler usw. darzustellen und entsprechend der Programmierung logisch zu verknüpfen. Dabei wird eine sogenannte Netzliste erzeugt. Dieser Schritt ist größtenteils unabhängig davon, welcher FPGA verwendet wird oder ob ein ASIC hergestellt wird. Durch ein weiteres Programm werden im nächsten Schritt die gefundenen Elemente auf die Strukturen abgebildet, die auf dem verwendeten FPGA vorhandenen sind (*Mapping*). Das Ergebnis ist eine physikalische Repräsentation des Designs durch Komponenten des FPGAs. Dadurch ist bekannt, wie viele von welchen Elementen des FPGAs benötigt werden.

2.4.2.4 Placing und Routing

Nun folgt die Platzierung der Elemente auf der entsprechenden Struktur (*Placing*) und deren elektrische Verschaltung (*Routing*). In diesem Schritt muss das zuständige Programm neben den Vorgaben bezüglich der verwendeten Elemente auch noch einige Randbedingungen (*Constraints*) einhalten. Zu allererst ist natürlich der Platz und die Anzahl der von einem Typ vorhandenen Elemente begrenzt. Zusätzlich müssen auch die Taktfrequenz und die Laufzeiten zwischen einzelnen Registern sowie die vorgegebene Position einiger Komponenten auf dem FPGA berücksichtigt werden. Die Constraints können vom Benutzer vorgegeben, aber auch teilweise vom verwendeten Programm selbst erstellt werden.

Ein wichtiger Begriff, der in diesem Zusammenhang häufig auftaucht, ist der des *kritischen Pfades*. Damit wird der Weg von einem Register zum nächsten bezeichnet, der die längste Laufzeit besitzt (vgl. Abbildung 2.9). Die Zeitdauer, die das Signal von `Register1` zu `Register2` über die eventuell dazwischenliegende asynchrone Logik benötigt, stellt eine Untergrenze für die Periodendauer des Takts und somit eine Obergrenze für die Taktfrequenz dar. Die gewünschte Laufzeit kann über Constraints für ganze Gruppen von Registern festgelegt werden. Sie ist oft auch ein Grund für das Scheitern des Place-and-Route-Vorganges. Durch eine sogenannte *Timing-Analyse* kann überprüft werden, ob die zeitlichen Anforderungen der Verbindungen eingehalten werden und an welchen Leitungen im Fehlerfall Probleme auftreten.

2.4.2.5 Bitstream-Generierung

Sind alle bisherigen Schritte erfolgreich gewesen, schreibt ein weiteres Programm die für die Konfiguration benötigten Informationen in ein spezielles Format, den *Bitstream*, und speichert diesen als Datei. Sie enthält die vollständige Funktionalität des FPGAs, also welche Elemente des FPGAs verwendet werden und wie diese untereinander verbunden sind, und muss zu dessen Verwendung in diesen geladen werden. Man spricht in diesem Zusammenhang von SRAM-basierten FPGAs, da die einzelnen Bits, die durch den Bitstream gesetzt werden und die die Verschaltung im FPGA kontrollieren, durch Abtrennen von der Stromversorgung gelöscht werden. Um den FPGA zu benutzen, ist es dann notwendig, ihn erneut mit dem Bitstream zu konfigurieren.

3 Beschreibung des vorhandenen Systems

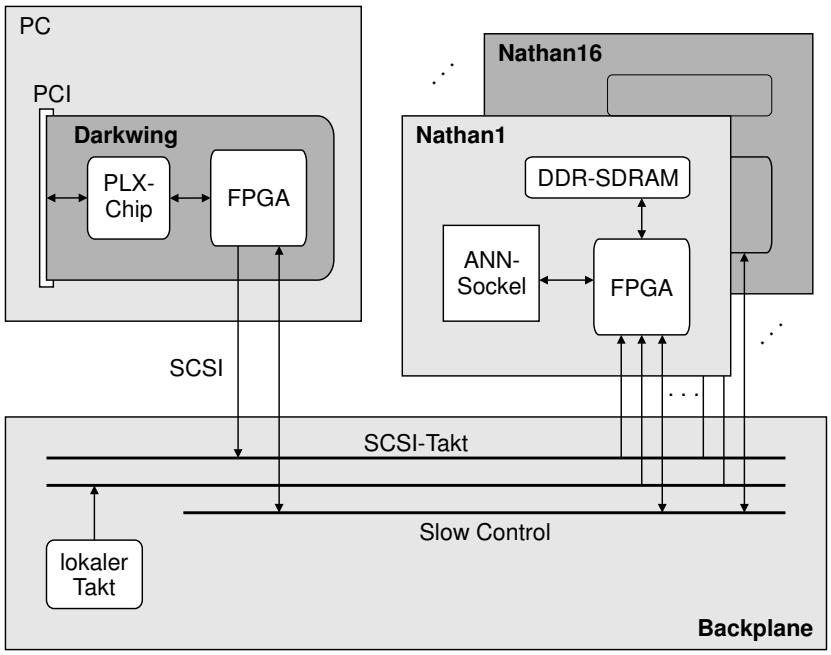
Das System, für das Gigabit-Ethernet-Schnittstelle implementiert wurde, ist eine *Plattform für parallel arbeitende Neuronale Netze* [11]. Abbildung 3.1(a) stellt das System schematisch dar und Abbildung 3.1(b) zeigt eine Fotografie des Aufbaus. Die wichtigste Komponente des Systems ist das Netzwerkmodul Nathan [12]. Das System beinhaltet insgesamt 16 Nathan-Module, die jeweils einen ANN ASIC besitzen. Die identischen Module sind in der Backplane eingesteckt und zum Teil über diese miteinander verbunden. Die Backplane ist wiederum an einen PC angeschlossen, der das System steuert.

Die einzelnen Elemente, aus denen das System besteht, und deren Zusammenhang werden in den nächsten Abschnitten erläutert. Dabei wird zunächst auf den PC (Abschnitt 3.1) und anschließend auf die Backplane (Abschnitt 3.2) sowie das Nathan-Modul(Abschnitt 3.3) eingegangen. Zuletzt wird der im System verwendete FPGA der Firma *Xilinx* beschrieben (Abschnitt 3.4).

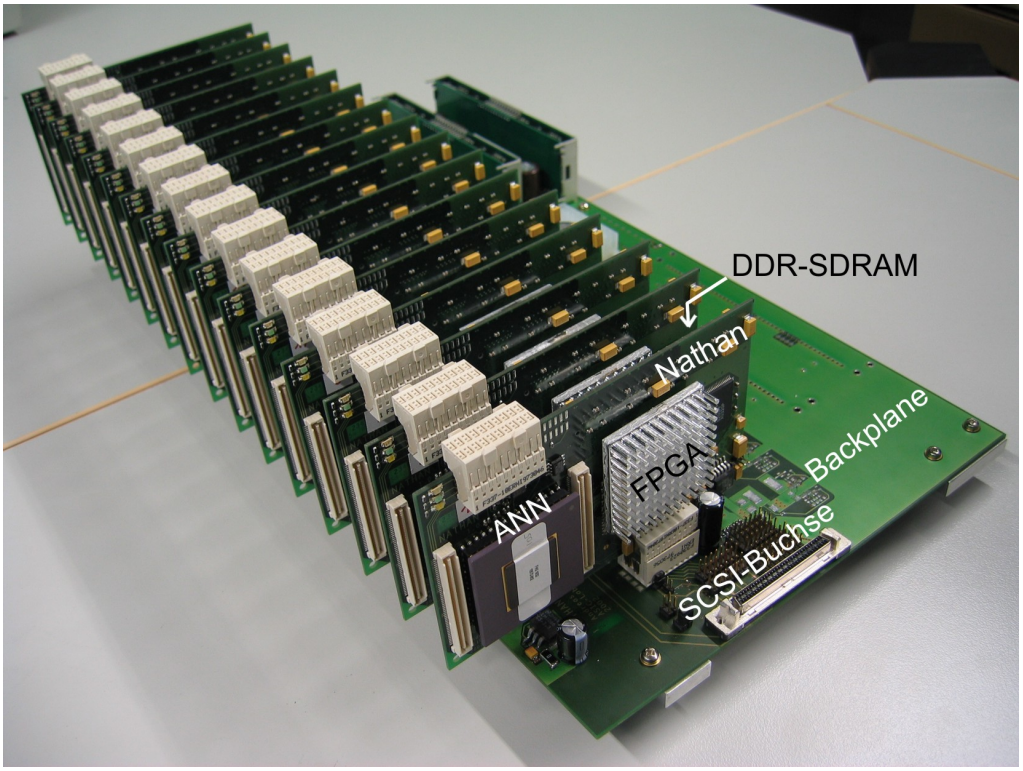
3.1 PC – Darkwing-Platine und Slow-Control

Der PC stellt die Schnittstelle zwischen System und Benutzer dar. Vom PC aus werden Test- und Trainingsdaten für das Neuronale Netzwerk an das System übertragen und die Ergebnisse ausgelesen. Dies geschieht über eine in den PC eingesteckte *PCI*[1]-Karte, die *Darkwing-Platine*. Sie besitzt einen PLX-Chip zur Kommunikation mit der PCI-Schnittstelle und einen FPGA, der über ein *SCSI*[2]-Kabel mit der Backplane verbunden ist. Die Steuerung des Systems erfolgt über die *Slow-Control*, welche von der Darkwing-Platine aus Daten seriell zur Backplane sendet und empfängt [27] [32]. Die Befehle werden in speziellen Frame übertragen und mit dem sogenannten *Handshake-Verfahren* quittiert [36]. Dies dient allgemein zur Sicherung von Verbindungen, denn ein Frame wird nur dann als erfolgreich versendet betrachtet, wenn innerhalb einer bestimmten Zeit eine Empfangsbestätigung (*Acknowledgment*) des Empfängers beim Sender ankommt. Ansonsten gilt das Frame als fehlerhaft bzw. nicht übertragen. Die Slow-Control ist im FPGA mit den zu steuernden Modulen verbunden. Zusätzlich werden über das SCSI-Kabel noch ein *Takt* (*Clock*) und mehrere Signale zur Konfiguration der FPGAs auf den Nathan-Modulen übertragen. Der Takt dient der Synchronisation der Daten und gibt die Arbeitsgeschwindigkeit der FPGAs vor.

[1] **P**eripheral **C**omponent **I**nterconnect
[2] **S**mall **C**omputer **S**ystem **I**nterface



(a) Schematische Darstellung der Systemkomponenten



(b) Foto einer voll bestückten Backplane mit 16 Nathan-Modulen (Quelle: Andreas Grübl)

Abbildung 3.1: Aufbau des Systems

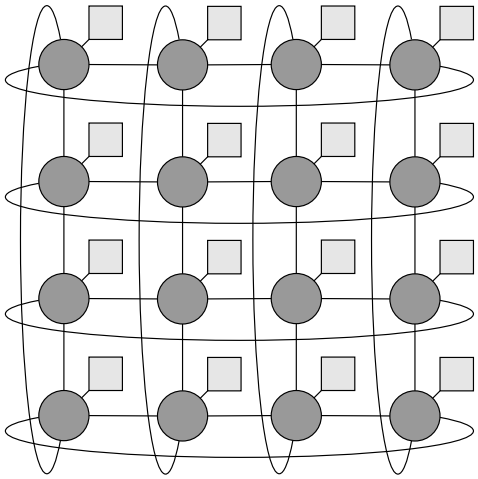


Abbildung 3.2: Aus den 16 Nathan-Modulen (dunkelgrau) gebildetes Netzwerk mit den zugehörigen FPGAs (hellgrau)

3.2 Backplane

Auf der Backplane befindet sich ein Quarz-Oszillator, der einen zusätzlichen Takt erzeugt. Das System kann deshalb wahlweise mit dem externen Takt über die SCSI-Verbindung oder aber mit dem Takt der Backplane betrieben werden. Die Slow-Control konfiguriert über eine serielle Kette alle Nathan-Module. Nach der Konfiguration wird die Leitung zur Implementation eines mit bis zu 100 MHz getakteten Netzwerks zwischen den Nathan-Modulen verwendet, das nach dem Prinzip eines Token-Rings funktioniert [14]. In einer zweiten Version der Backplane *Balu*, die Anfang 2007 aufgebaut und in Betrieb genommen wurde, ist die physikalische Busstruktur ersetzt worden. Es befindet sich nun ein FPGA auf der Backplane, von dem aus es einzelne Verbindungen zu jedem der Nathan-Module gibt. Dieser FPGA ist identisch mit dem auf den Nathan-Modulen (vgl. Abschnitt 3.4). Außerdem existiert auf Balu ein weiterer Steckplatz für Zusatzmodule.

Die parallel arbeitenden Nathan-Module können zusätzlich noch über die Backplane miteinander Daten austauschen, allerdings ist jedes mit nur jeweils vier anderen Modulen über Datenleitungen mit 312,5 MByte/s fest verbunden. Sie bilden zusammen einen 2D-Torus, wie er in Abbildung 3.2 dargestellt ist. Man erkennt in der Abbildung, dass selbst im schlechtesten Fall die Verbindung zwischen zwei beliebigen Nathan-Modulen über höchstens drei Zwischenstationen führt.

3.3 Nathan-Modul

Die 16 Nathan-Module besitzen jeweils einen Xilinx FPGA (vgl. Abschnitt 3.4) mit acht integrierten *MGTs*[3] und einem integriertem IBM *PowerPC* 405 Prozessor [38]. Vier dieser MGTs führen zur Backplane und bilden den im vorangegangenen Abschnitt erwähnten 2D-Torus.

[3] **M**ulti-**G**igabit **T**ransceiver

Rocket IO Transceiver Blocks	PowerPC Processor Blocks	Logic Cells[7]	CLB (1 = 4 slices = max 128 bits)	
			Slices	Max Distr RAM (Kb)
8	1	11.088	4.928	154

18 X 18 Bit Multiplier Blocks	Block SelectRAM+		DCMs	Maximum User I/O Pads
	18 Kb Blocks	Max Block RAM (Kb)		
44	44	792	4	396

Tabelle 3.1: Kenndaten des XC2VP7 (Quelle: [38])

Die vier anderen MGTs führen zu einem Stecker auf der Oberseite des Nathan-Moduls. Von dort lassen sie sich beliebig mit Kabeln zu weiteren Nathan-Modulen auf derselben oder einer anderen Backplane verbinden. Der PowerPC bietet die Möglichkeit, mit *Linux* betrieben zu werden, was den Einsatz der vorhandenen Linux-basierten Software auf den Nathan-Modulen zulässt [35].

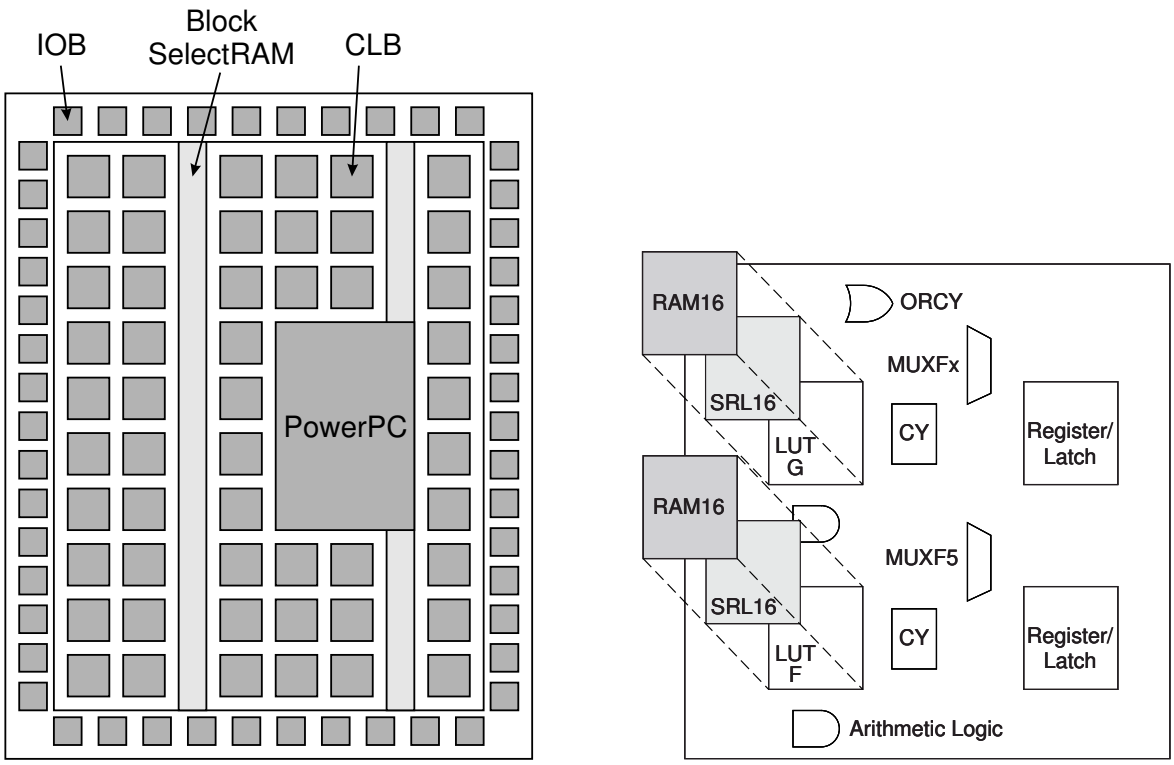
Zusätzlich sind auf dem Nathan-Modul Speicher-Module vorhanden, die jeweils über ein 64-Bit-Interface direkt mit dem FPGA verbunden sind: zwei *SRAM*[4]-Module und ein *DDR-SDRAM*[5]-Modul. Die beiden SRAM-Module bieten einen Datenzugriff in 2 Takten (*ZBT*[6]), während das SDRAM als Massenspeicher für bis zu 1 GByte zur Verfügung steht. Über die MGT-Verbindungen gibt es zusätzlich die Möglichkeit, die SDRAM-Module der einzelnen Nathans zu einer gemeinsamen virtuellen *Shared-Memory*-Struktur zusammenzufügen [11]. Eine weitere wichtige Komponente des Nathan-Moduls ist der ANN-Sockel. Dies ist der Steckplatz des eigentlichen ANN ASIC, er kann aber auch für andere passende Module verwendet werden.

3.4 Xilinx FPGA

Das System ist Dank der auf den Nathan-Modulen befindlichen FPGAs sehr flexibel einsetzbar. Dies zeigt sich unter anderem daran, dass der FPGA, obwohl ursprünglich zur Steuerung des ANN ASICs konzipiert, auch für den Einsatz von Ethernet verwendet werden kann (vgl. Abschnitt 5.2). Der FPGA auf Balu erlaubt eine variable Verwendung der Nathan-Module und ermöglicht die Ansteuerung von zusätzlichen Komponenten auf der Backplane wie z.B. dem Modul für die Ethernet-Schnittstelle (vgl. Abschnitt 5.3).

Bei den FPGAs auf den Nathan-Modulen sowie auf Balu handelt es sich jeweils um einen Xilinx Virtex-II Pro FPGA (Typ: XC2VP7). Dessen wichtigste Merkmale sind in Tabelle 3.1 zusammengefasst. Für zusätzliche Informationen wird auf [38] und [39] verwiesen. Abbildung 3.3(a)

[4] **S**tatic **R**andom **A**ccess **M**emory
[5] **D**ouble **D**ata **R**ate **S**ynchronous **D**ynamic **R**andom **A**ccess **M**emory
[6] **Z**ero **B**us **T**urnaround
[7] Logic Cell ≈ (1) 4-input LUT + (1)FF + Carry Logic



(a) Darstellung der einzelnen Bestandteile

(b) Slice innerhalb der CLBs (Quelle: [38])

Abbildung 3.3: Schematischer Aufbau des Xilinx FPGAs

stellt den schematischen Aufbau des FPGAs dar, der Übersicht wegen wurde dabei auf die Verbindungsleitungen verzichtet.

Die Logikzellen des Xilinx FPGAs werden CLBs[8] genannt. Diese sind beim verwendeten FPGA aus vier gleichen Elementen aufgebaut, den *Slices* [38]. Ein Slice wiederum besitzt, wie in Abbildung 3.3(b) gezeigt, zwei 4x4-Matrizen, die je nach Konfiguration als 16 Bit des verteilten Speichers (Distributed SelectRAM), als 16-Bit-*Schieberegister* oder als 4-input LUT[9] dienen. Außerdem enthält ein Slice Multiplexer und Register. Die Ein-/Ausgangszellen des FPGAs werden mit IOB[10] bezeichnet. Mittels sogenannter *Makros* lassen sich diese Zellen als Eingangspuffer ($IBUF$[11]) oder Ausgangspuffer ($OBUF$[11]) für Signale konfigurieren. Außerdem besitzt der FPGA noch weitere Elemente für spezielle Aufgaben. Zu erwähnen sind dabei die Speicherzellen (Block SelectRAM) und die schon in Abschnitt 3.3 genannten MGTs sowie der PowerPC. Für die Verteilung eines Taktes auf dem FPGA gibt es die schon angesprochenen globalen Leitungen, die je von speziellen Eingangspuffern ($BUFG$[11]) getrieben werden. Dies sorgt für eine gleiche Phasenlage des Taktes an allen CLBs. Die Takte wiederum gelangen in den FPGA über bestimmte IOBs, die in HDL durch $IBUFGs$[11] repräsentiert werden. Zwischen IBUFG und BUFG kann sich zusätzlich noch ein DCM[12] befinden. Diese Komponente

[8] **C**onfigurable **L**ogic **B**locks
[9] **L**ook-**U**p-**T**able
[10] **I**nput/**O**utput **B**lock
[11] (**I**nput/**O**utput) **BUF**fer (**G**lobal)
[12] **D**igital **C**lock **M**anager

ermöglicht es, die Frequenz des Taktes am Eingang des DCMs um einstellbare Faktoren zu vervielfachen oder zu reduzieren. Die veränderten Frequenzen liegen an verschiedenen Ausgängen des DCMs an. Gleichzeitig lässt sich die Phase zwischen Eingang und Ausgängen gegeneinander verschieben.

Für den Design-Flow stellt Xilinx eine Reihe von Programmen zur Verfügung (vgl. Abschnitt 2.4.2). Mit dem *CORE Generator* lassen sich die vorgefertigten Module konfigurieren, die Synthese des Designs geschieht mit *XST*[13] und das Mapping mit *MAP*. Das Programm *PAR*[14] übernimmt das Placing und Routing und schließlich wird mit *BitGen* der Bitstream erzeugt. Außerdem wird *ModelSim* für die Durchführung von Simulationen unterstützt.

[13] **X**ilinx **S**ynthesis **T**echnology
[14] **P**lace-**A**nd-**R**oute

4 Problemstellung und Lösungskonzept

Nachdem in den vorangegangenen Kapiteln die grundlegenden Begriffe erläutert (Kapitel 2) und die wichtigsten Bestandteile des Systems dargestellt wurden (Kapitel 3), wird in diesem Kapitel die Problemstellung genauer beschrieben, mit der sich diese Arbeit befasst. Außerdem wird ein entsprechender Lösungsansatz präsentiert und ein Überblick über den Stand zu Beginn der Arbeit gegeben.

Beim Einsatz des Systems in seiner jetzigen Form treten mehrere Schwierigkeiten auf. Das erste Problem stellt die Übertragungsgeschwindigkeit zwischen PC und System dar. Der ANN ASIC wird über den ANN-Sockel der Nathan-Moduls gesteuert (vgl. Abschnitt 3.3). Dieser beherbergt das künstliche neuronale Netzwerk, um das es in der Forschung der Electronic Vision(s) Arbeitsgruppe geht. Mittlerweile existieren zwei Chips, die ein neuronales Netzwerk implementieren: *HAGEN*[1] [31] und *Spikey*[2] [30]. Auf dem Erstgenannten befinden sich 256 Neuronen und 32768 Synapsen. Der aktuelle ANN ASIC Spikey bildet 384 Neuronen und 98304 Synapsen nach. Bei damit durchgeführten Experimenten wird unter anderem das Feuerverhalten der Neuronen, d.h. die Frage wann ein *Puls* erzeugt wird, untersucht. Aus diesem Grund wäre es wünschenswert, wenn die Pulse von möglichst vielen Neuronen ausgewertet werden könnten.

Das ist aufgrund der großen Datenmenge nicht für alle möglich, wie folgende Rechnung zeigt: Ein biologisches Neuron feuert im Durchschnitt mit 10 Hz. Da die Implementation in Hardware eine Beschleunigung gegenüber der biologischen Zeitskala von etwa 10^5 bewirkt, resultiert daraus eine Feuerrate von 1 MHz für ein Neuron auf dem ANN ASIC. Je drei Pulsereignisse werden am Interface in 64 Bits übermittelt, woraus sich eine Datenrate von 21 Mbit/s ergibt. Für das gesamte System aus 16 Nathan-Modulen mit 16 ANN ASICs und je 384 Neuronen ergibt sich eine durchschnittliche Datenrate von 130 Gbit/s. Für die Datenübertragung zwischen PC und Backplane wird die Slow-Control benutzt (vgl. Abschnitt 3.1). Diese überträgt bei 80 bis 100 MHz seriell, also 80 bis 100 Mbit/s. Dass sie in erster Linie zur Steuerung des Systems ausgelegt ist, erkennt man an der Struktur der Frames. Nur 39% der übertragenen Bits sind Daten, der Rest beinhaltet die Adressierung und weitere Bits zur korrekten Frameerkennung. Diese Umstände verdeutlichen das erste Problem: Die Geschwindigkeit der Datenübertragung stellt einen stark limitierenden Faktor für die Leistungsfähigkeit des Systems und dessen weiteren Ausbau dar. Weniger als ein Promille der erzeugten Pulse können bisher gleichzeitig analysiert werden.

Ein weiteres Problem stellt die Art der Steuerung des Systems dar. Das geschieht über die Darkwing-Platine (vgl. Abschnitt 3.1) und über eigens dafür geschriebene Software. Da, wie in Kapitel 1 beschrieben, die Electronic Vision(s)-Gruppe an dem Projekt FACETS beteiligt ist,

[1] **H**eidelberg **A**nalo**G** **E**volvable **N**etwork
[2] Spiking Neural Network Chip

besteht ein Ziel ihrer Arbeit auch darin, dass andere daran beteiligte Arbeitsgruppen das System einsetzen können. Deren bisherigen Software-Simulationen von neuronalen Netzen sollen durch den Einsatz des Backplane-Systems ergänzt werden. Bei der Darkwing-Platine handelt es sich aber um einen extra für die Ansteuerung des neuronalen Netzwerk-Systems entwickeltes Board. Dieses ist nur in begrenzter Stückzahl vorhanden und müsste jedem, der das System einsetzen will, bereitgestellt werden. Außerdem wird spezielle Software zur Ansteuerung der Darkwing-Platine benötigt, die dann die Slow-Control-Befehle zur Backplane überträgt. Diese muss an den jeweiligen PC angepasst werden. Dadurch wird die Verwendung des neuronalen Netzwerkes durch andere Arbeitsgruppen erheblich erschwert.

Damit das System aus Backplane und Nathan-Modulen für weitere Entwicklungen des ANN ASICs in Zukunft leistungsfähig genutzt und auch von anderen Arbeitsgruppen des FACETS-Projektes für ihre Forschung verwendet werden kann, ist es dringend notwendig, die Verbindung zwischen PC und System hinsichtlich der beiden dargestellten Probleme zu überarbeiten. Dies war die Aufgabe, die sich der vorliegenden Arbeit gestellt hat.

Ein Ziel ist es eine möglichst schnelle Auslese der Daten der ANN ASICs zu erreichen. Zusätzlich ist beabsichtigt, über dieselbe Verbindung die Backplane und die Nathan-Module zu steuern, um damit langfristig die SCSI-Anbindung zu ersetzen. Dadurch wird eine höhere Portabilität zu den anderen Partnern des FACETS-Projektes erreicht.

Zur Lösung der genannten Probleme bietet sich aufgrund dieser Ziele Gigabit-Ethernet an. Gigabit-Ethernet überträgt Daten mit einer Rate von 1000 Mbit/s, außerdem beinhaltet ein Frame maximaler Länge unter 2 % an Header-Informationen, der Rest sind Daten. Im Hinblick auf die Weitergabe des Systems an andere Gruppen besitzt Ethernet den Vorteil, dass es sich um einen sehr weit verbreiteten Standard handelt. Ein handelsüblicher PC lässt sich kostengünstig und ohne Probleme auf Gigabit-Ethernet erweitern, d.h. nahezu jeder PC kann zur Steuerung der Backplane und der Nathan-Module eingesetzt werden.

Um eine Gigabit-Ethernet-Schnittstelle zu implementieren, ist es notwendig, die beiden Schichten des OSI-Referenzmodells, die Ethernet benutzt, zu realisieren (vgl. Abschnitt 2.2 und 2.3). Dazu wurde eine Platine mit Phy der Firma *devboards* gekauft [6], die im weiteren Text als *Phy-Modul* bezeichnet wird. Dieses stellt Schicht 1 des OSI-Referenzmodells dar. Weiter wurde für Schicht 2 auf einen frei verfügbaren IP-Core des Internet-Projektes *OpenCores* verwendet [24]. Dieser liegt als HDL-Beschreibung in Verilog HDL vor. Bei dem IP-Core handelt es sich um einen MAC für 10- und 100-Mbit/s-Ethernet.

Das Phy-Modul soll auf den entsprechenden Steckplätzen auf dem Nathan-Modul bzw. auf der Balu-Backplane verwendet werden. Für den MAC ist vorgesehen, ihn auf dem damit verbundenen FPGA einzusetzen. Zu ersterem lagen bei Beginn der Arbeit schon Erfahrungen vor, denn Sebastian Millner hat mit den beiden Komponenten (Phy-Modul und MAC) erste Übertragungen im Rahmen einer Miniforschungsarbeit durchgeführt [20]. Diese dienten als Grundlage, auf die aufgebaut werden konnte.

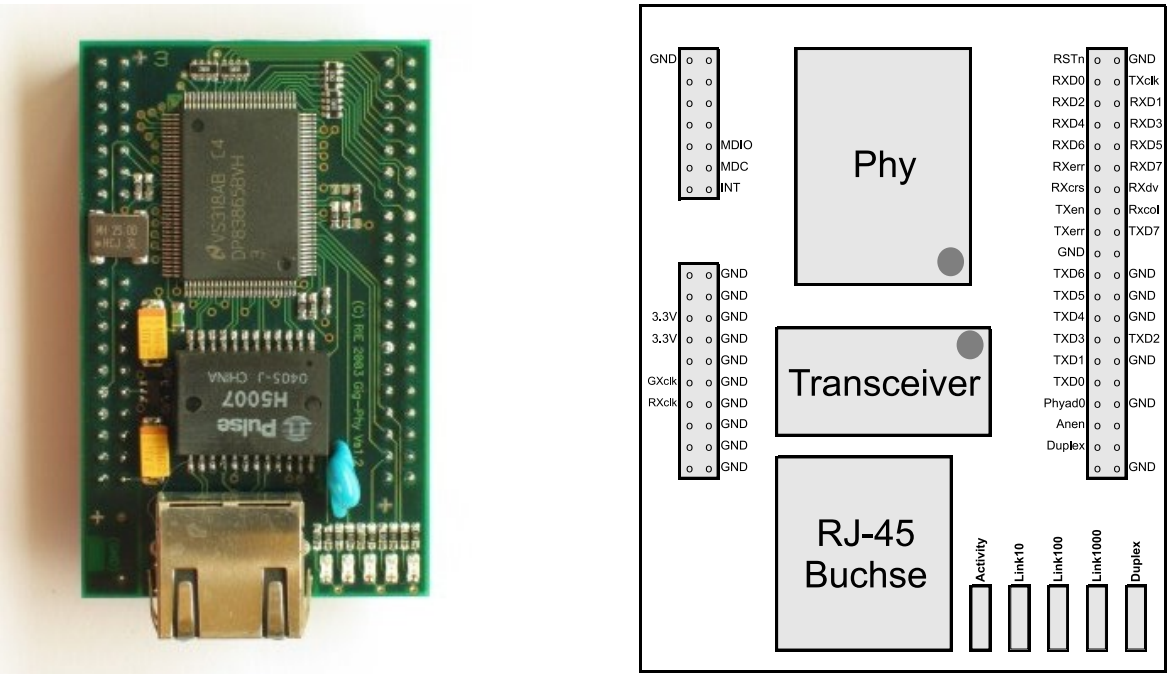
In Kapitel 5 wird das Phy-Modul beschrieben und dessen Einsatzgebiete, Nathan-Modul und Balu-Backplane, dargestellt. Kapitel 6 enthält detaillierte Erläuterungen zum Ethernet-IP-Core.

5 Ethernet-Hardware

Die Ethernet-Hardware ist sowohl für den Einsatz auf dem Nathan-Modul als auch für den auf der Backplane geeignet. In Abschnitt 5.1 werden die wichtigsten Funktionen des Phy-Moduls beschrieben und anschließend in den Abschnitten 5.2 und 5.3 die Verwendung mit der jeweiligen Hardware erläutert.

5.1 Phy-Modul

Das Phy-Modul stellt den physikalische Verbindung zum Übertragungsmedium her. Die erworbene Platine trägt die Bezeichnung *DBGIG1 Gigabit Ethernet Phy Module* [6] und ist in Abbildung 5.1 dargestellt. Sie wurde vom Hersteller für die Montage auf *Altera NIOS Development Boards* konzipiert und besitzt deshalb die von Altera spezifizierten Anschlüsse für einen *Santa-Cruz-Sockel* [18]. Das Modul verfügt über einen Phy, der zu den entsprechenden Standards für 10, 100 und 1000 Mbit/s-Ethernet über Twisted-Pair-Kabel kompatibel ist [23]. Für



(a) DBGIG1 (Quelle: devboards.de)

(b) Aufsicht (Quelle: [6])

Abbildung 5.1: Phy-Modul

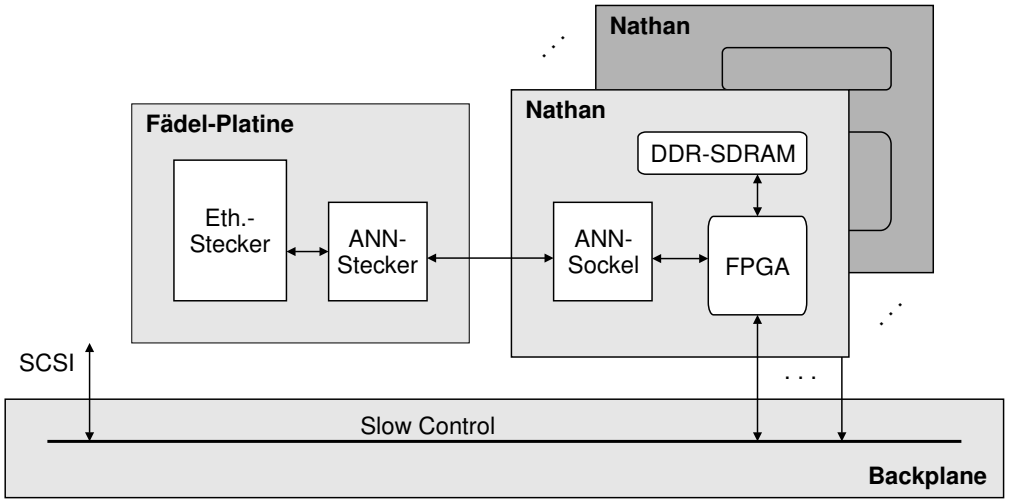


Abbildung 5.2: Verwendung der Phy-Moduls auf dem Nathan-Modul

den Anschluss des Kabels an die Platine und für die Datenübertragung sind ein *magnetischer Transceiver* sowie eine *RJ-45-Buchse* vorhanden und mit dem Phy verbunden. Im Transceiver ist auch der Hybrid für das Echo-Cancellation-Verfahren untergebracht (vgl. Abschnitt 5.1). Zum Senden von Daten werden je nach Betriebsmodus ein 2,5-MHz-, 25-MHz- oder ein 125-MHz-Takt verwendet, beim Empfangen rekonstruiert der Phy den Takt aus dem Datenstrom. Für das Senden mit 10 und 100 Mbit/s besitzt das Phy-Modul einen *25-MHz-Oszillator*, der den benötigten Takt erzeugt, für den Betrieb im 1000 Mbit/s-Modus ist ein 125-MHz-Takt von außen notwendig. Neben dem Oszillator sind auf dem Phy-Modul fünf *LEDs*[1] zur Statusanzeige untergebracht. Zusätzlich zu den Takten wird die Platine noch mit einer Spannung von 3,3 V versorgt.

Zum Anschluss an den MAC der höheren Netzwerk-Schicht besitzt der Phy ein GMII, das auch zur Verwendung als RGMII oder als MII konfiguriert werden kann (vgl. Abschnitt 2.3.2). Es wird über die Pins des Phy-Moduls entsprechend verbunden. Mit Hilfe des *MII Management-Interfaces* sind Zugriffe auf die Steuer- und Status-Register des Phys synchron zur MDC möglich. Die MDC darf mit höchstens 2,5 MHz betrieben werden. Die einzelnen Register sowie deren Bedeutung sind in [23] aufgeführt. Gleichzeitig stellt das *MII Daten-Interface* die Daten-Verbindung des Phys mit der Host-Platine her. In Abbildung 5.1(b) ist das Modul schematisch mit Pin-Belegung und den Bedeutungen der Status-LEDs dargestellt. Abbildung 5.1(a) zeigt zum Vergleich eine Fotografie des Phy-Moduls.

5.2 Anwendungskonzept der Ethernet-Hardware auf dem Nathan-Modul

Wie in Abschnitt 3.3 erwähnt, steht auf dem Nathan-Modul ein Steckplatz zur Verfügung, der ANN-Sockel. Dieser besitzt genügend Pins für das GMII und das Management Interface, die Anschlüsse sind aber nicht kompatibel. Aus diesem Grund hat Sebastian Millner, der im

[1] **L**ight **E**mitting **D**iode

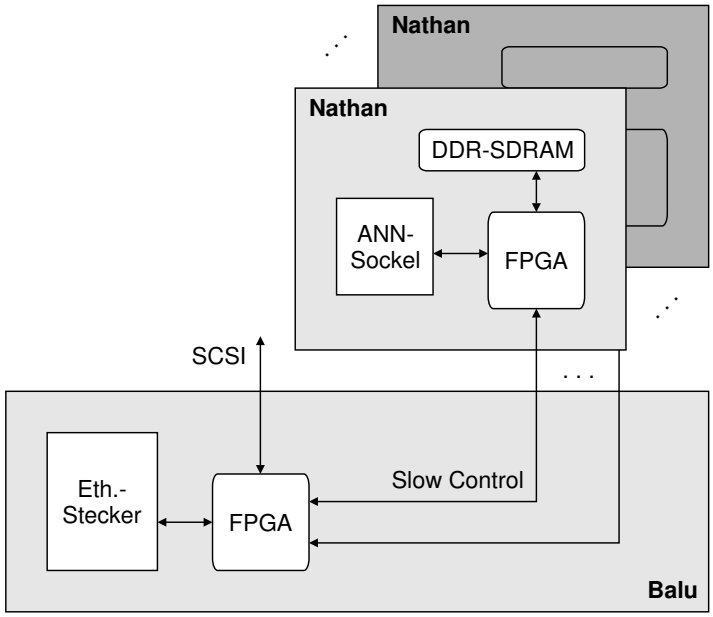


Abbildung 5.3: Verwendung der Phy-Moduls auf der zweiten Backplane (Balu)

Rahmen seiner Miniforschungsarbeit bei Electronic Vision(s) die erste Inbetriebnahme des Ethernets mit 10 und 100 Mbit/s durchgeführt hat, eine Lochraster-Platine mit den beiden entsprechenden Anschlüssen versehen und die Pins mit Fädeldraht verbunden [20]. Diese Methode, wie sie in Abbildung 5.2 dargestellt wird, hat bei 10 und 100 Mbit/s gut funktioniert, bei Gigabit-Ethernet jedoch traten Probleme mit den Signalen, vor allem den Takten auf. In Abschnitt 10.3 werde ich näher darauf eingehen und Verbesserungen vorstellen.

Der MAC des Ethernets wird im FPGA des Nathan-Moduls untergebracht. Die Steuerung des MAC erfolgt durch die Anbindung an die Slow-Control, die über die Backplane mit dem FPGA verbunden ist [27] [32]. Ein Vorteil, den dieser Einsatzort mit sich bringt, ist die direkte Anbindung des Ethernet-MACs an das DDR-SDRAM auf dem Nathan-Modul. Jedoch stehen so dem System nur noch 15 Nathan-Module für ANN-ASICs zur Verfügung.

5.3 Anwendungskonzept der Ethernet-Hardware auf der Balu-Backplane

Ende 2006 wurde von Dan Husmann de Oliveira ein Nachfolgemodell der Backplane entwickelt, Balu. Sie besitzt neben den in Abschnitt 3.2 und Abschnitt 3.4 beschriebenen Änderungen wie den Einzelverbindungen der Nathan-Module mit dem vorhandenen FPGA außerdem einen eigenen Steckplatz für das Phy-Modul, der direkt mit dem FPGA verbunden ist (vgl. Abbildung 5.3). Der Steckplatz ist kompatibel zu den Santa-Cruz-Steckerleisten, d.h. eine zusätzliche Platine zur Verbindung ist überflüssig. Neben dem Vorteil der besseren physikalischen Anbindung können wieder alle Nathan-Module mit ANN ASICs bestückt werden. Der MAC wird im FPGA von Balu implementiert. Seine Steuerung erfolgt ebenfalls über die Slow-Control, die direkt über den SCSI-Anschluss vom PC kommt. Die Steuerung der Backplane

ohne SCSI-Anschluss über Ethernet ist nicht möglich, da das entsprechende Modul erst noch implementiert wird. In Abschnitt 7.11 stelle ich das Konzept vor, wie dies auszusehen hat. Auf dem FPGA von Balu steht mehr Platz zur Verfügung als auf den Nathan-Modulen, da bisher lediglich der Ethernet-MAC darauf implementiert wird. Von Nachteil ist, dass zur Speicherung von Daten nur auf den im FPGA vorhandenen Speicher (SelectRAM) zugegriffen werden kann (vgl. Abschnitt 3.4). Um das DDR-SDRAM auf den Nathan-Modulen zu verwenden, müssen die Daten erst über die Slow-Control an das entsprechende Nathan-Modul gesendet werden.

6 Übersicht über den verwendeten Ethernet-Core

In den nachfolgenden Abschnitten wird der IP-Core von OpenCores beschrieben. Obwohl dieser ein MAC für 10-Mbit/s- und 100-Mbit/s-Ethernet ist (vgl. Abschnitt 2.3.3), wird er im weiteren Text als *Ethernet-Core* bezeichnet. Das dient der Unterscheidung zwischen dem erworbenen IP-Core und dem an verschiedenen Stellen veränderten und erweiterten MAC für Gigabit-Ethernet, *Ethernet-MAC* genannt (vgl. Kapitel 7). Wie der Ethernet-Core mit den anderen Teilen des Systems in Verbindung steht, zeigt Abbildung 6.1 Die näheren Erläuterungen dazu erfolgen in Abschnitt 6.2.1.

Der Ethernet-Core besteht aus einer Reihe von Modulen, deren Hierarchie in Abbildung 6.2 dargestellt ist [21]. Zusätzlich zeigt die Abbildung die Unterteilung des Ethernet-Cores in Bereiche mit unterschiedlicher Taktung, wie sie in Abschnitt 6.3 beschrieben wird. Nach einem Überblick über die Arbeitsweise des Ethernet-Cores (Abschnitt 6.1) erfolgt eine Erläuterung der einzelnen Module mit ihren Unter-Modulen (Abschnitt 6.2). Diese sind in der Hardware-Beschreibungssprache Verilog HDL geschrieben und jeweils mit ihrem Modul-Namen aufgeführt.

6.1 Funktionsweise des Ethernet-Cores

Der Ethernet-Core hat in seiner Funktion als MAC mehrere Aufgaben zu erfüllen, die sich auf die einzelnen Module verteilen. Diese sind nachfolgend beschrieben.

6.1.1 Empfangen von Frames

Das Modul `eth_rxethmac` (Abschnitt 6.2.6) ist für das Empfangen von Frames zuständig. Deshalb ist es über das MII mit dem Phy verbunden. Neben dem Empfang der Daten erfüllt

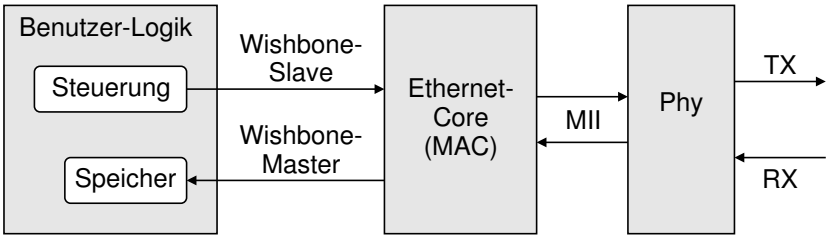


Abbildung 6.1: Anbindung des Ethernet-Cores ans System

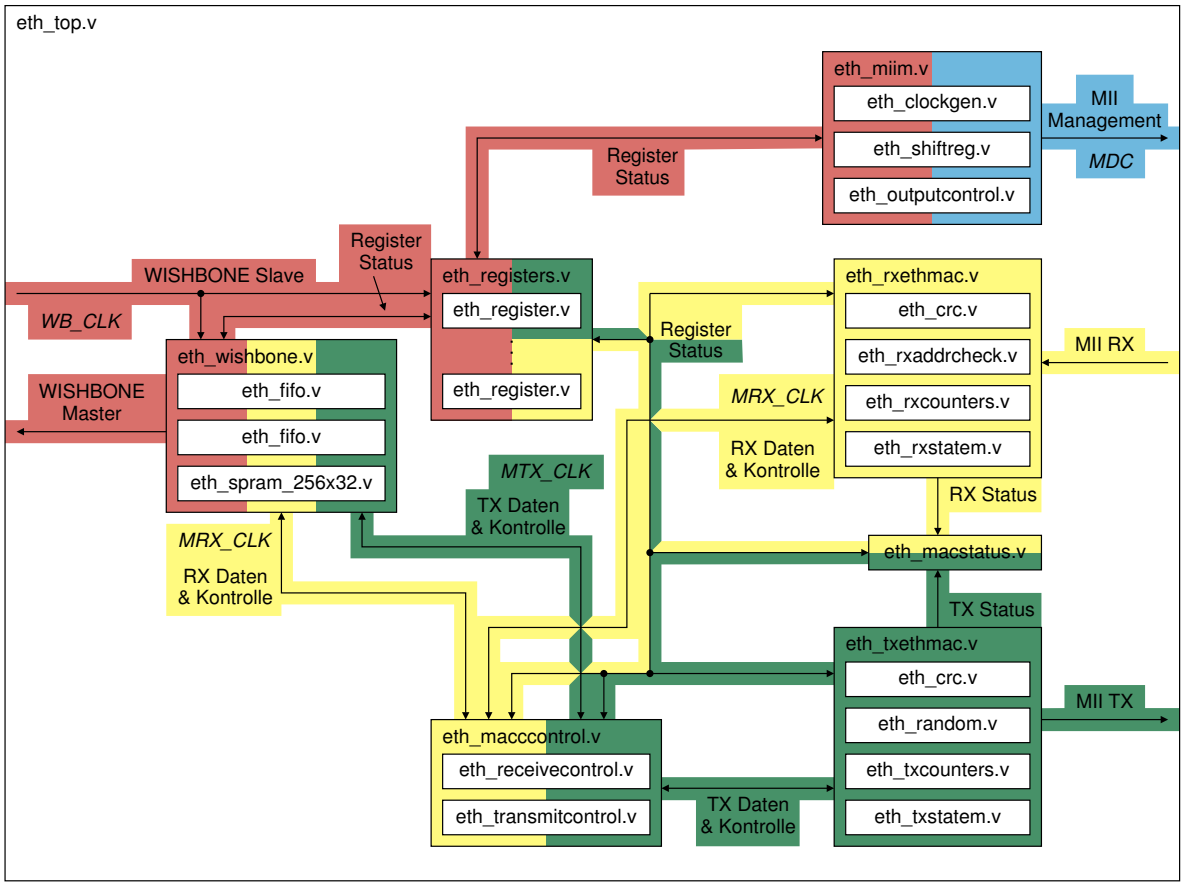


Abbildung 6.2: Alle Module des Ethernet-Cores und deren logische Verbindungen (Quelle:[21]). Die unterschiedlichen Farben stellen dabei die verschiedenen Takt-Domänen dar (MDC, WB_CLK, MTX_CLK und MRX_CLK).

es die damit zusammenhängenden Kontrollfunktionen. So wird z.B. die CRC-32-Prüfsumme und die Zieladresse jedes Frames überprüft.

6.1.2 Versenden von Frames

Alle das Versenden von Frames betreffenden Aufgaben werden durch `eth_txethmac` (Abschnitt 6.2.7) implementiert. Es sendet über das MII die Daten zum Phy. Zusätzlich wird hier die CRC-32-Prüfsumme zur Fehlererkennung erzeugt und die Behandlung von Kollisionen im Halbduplex-Betrieb durchgeführt.

6.1.3 Überwachung der laufenden Übertragungen

Das Modul `eth_macstatus` (Abschnitt 6.2.3) überwacht den Zustand von stattfindenden Daten-Übertragungen zum Phy. Dazu ist es mit den beiden Modulen `eth_rxethmac` und `eth_txethmac` verbunden und sammelt Informationen über ankommende und abgehende Frames.

6.1.4 Fluss-Kontrolle

In `eth_maccontrol` (Abschnitt 6.2.2) ist die Flusskontrolle untergebracht (vgl. Abschnitt 2.3.3). Hier werden die empfangenen Frames auf Pausen-Frames hin untersucht und das Senden von Frames gegebenenfalls unterbrochen. Außerdem werden auch Pausen-Frames für den Versand zusammengestellt.

6.1.5 Steuerung des Phys

Durch das Modul `eth_miim` (Abschnitt 6.2.4) wird über das MII Management Interface mit dem Phy kommuniziert. Dessen Register können hiermit gelesen und beschrieben werden.

6.1.6 Zugriff auf den Daten-Speicher

Das Modul `eth_wishbone` (Abschnitt 6.2.8) ist über das WISHBONE-Master-Interface mit einem Daten-Speicher verbunden, um von dort Daten zu lesen und zu schreiben. Da Senden ohne genügend Daten bzw. Empfangen ohne freien Speicher nicht möglich ist, reguliert das Modul auch die entsprechenden Vorgänge.

6.1.7 Steuerung durch den Benutzer

Das Modul `eth_registers` (Abschnitt 6.2.5) dient der Steuerung des Ethernet-Cores durch weitere Benutzer-Logik. Das geschieht durch Schreiben von Registerwerten über das WISHBONE-Slave-Interface. Gleichzeitig erlaubt das Lesen der Registerwerte eine Statuskontrolle des Ethernet-Cores.

6.2 Detaillierte Beschreibung der Module des Ethernet-Cores

Die nachfolgende Beschreibung der einzelnen Module soll deren wichtigste Funktionen beschreiben. Außerdem wird ein Überblick über die wichtigsten dazu verwendeten Signale gegeben, sofern diese im Kapitel 7 von Bedeutung sind.

6.2.1 eth_top

Das oberste Modul in der Hierarchie des Ethernet-Cores heißt `eth_top`. Es dient der Verschaltung der gesamten Untermodule, die nachfolgend beschrieben werden (Abschnitt 6.2.2 bis 6.2.9). Das letzte der beschriebenen Module, `eth_defines`, beinhaltet jedoch nur einige vom Ethernet-Core benötigten Definitionswerte. `eth_top` beinhaltet außerdem noch folgende Schnittstellen: Ein MII zum Phy (vgl. Abschnitt 2.3.2) sowie zwei WISHBONE-Schnittstellen [25]. Bei Letztgenannten handelt es sich um standardisierte Schnittstellen, die die Verbindung von verschiedenen IP-Cores erleichtern sollen. Es gibt zwei Arten von ihnen: Master und Slave. Der verwendete Ethernet-Core besitzt jeweils eines jeder Sorte: Über das WISHBONE-Master-Interface kontrolliert der Ethernet-Core den Zugriff auf den damit verbundenen Datenspeicher,

das WISHBONE-Slave-Interface dient der Steuerung des Ethernet-Cores durch vom Benutzer erstellte Logik. Zusätzlich findet über die am WISHBONE-Slave-Interface angelegte Adresse eine Selektion statt, auf welches der beiden Module, `eth_registers` (Abschnitt 6.2.5) oder `eth_wishbone` (Abschnitt 6.2.8), zugegriffen werden soll. Eine weitere Funktion des Moduls `eth_top` ist der *Core-Loopback*. Damit lassen sich zu Testzwecken die MII-Ausgänge zum Phy direkt mit den MII-Eingängen des Ethernet-Cores kurzschließen. Dadurch ist der Phy vom Ethernet-Core getrennt. `eth_top` besitzt außerdem noch Synchronistationslogik für einige Signale, die asynchron sind bzw. eine Takt-Grenze überschreiten. Auf die Synchronisation allgemein wird in Abschnitt 6.3 näher eingegangen.

6.2.2 eth_maccontrol

In diesem Modul ist die Flusskontrolle (vgl. Abschnitt 2.3.1) des Ethernets implementiert. Wie in Abschnitt 2.3.3 erklärt ist, werden Pausen-Frames vom Empfänger eingesetzt, um die Sendegeschwindigkeit des Gegenübers zu reduzieren. Dafür durchlaufen alle Daten- und zugehörigen Statussignale (`TxData`, `TxStartFrm`, `TxEndFrm` usw.) dieses Modul bevor sie gesendet werden. Ein Multiplexer entscheidet, ob die normalen Frame-Daten oder die Daten eines Pausen-Frames aus `eth_transmitcontrol` weitergegeben werden. Ein Pausen-Frame wird dabei erst weitergeleitet, wenn gerade kein Frame gesendet wird. Für das Senden und den Empfang von Pause-Frames sind dabei die beiden Untermodule `eth_receivecontrol` und `eth_transmitcontrol` zuständig.

6.2.2.1 eth_receivecontrol

In `eth_receivecontrol` gelangen die Daten eines empfangenen Frames inklusive Statussignale (`RxData`, ...) von `eth_rxethmac` (Abschnitt 6.2.6). Hier wird überprüft, ob die Daten ein Pausen-Frame darstellen, d.h. ob die einzelnen Felder dem in Abbildung 2.7 gezeigten Frame entsprechen. Ist dies der Fall, liest das Modul daraus die Dauer der Pause aus und unterbricht das Senden von Frames für die angegebene Zeitdauer. Stattfindende Übertragungen werden dabei nicht gestoppt, sondern die Pause nach deren Ende ausgeführt. Die Pausendauer wird gespeichert (`PauseTimer`) und dem Modul `eth_macccontrol` mitgeteilt, dass bis auf weiteres keine Frames mehr gesendet werden dürfen. Ein einzelner Zeitschritt von 64 Bytes Länge wird im Ethernet-Core als *Slot* bezeichnet (vgl. Abschnitt 2.3.1). In `eth_receivecontrol` sorgen zwei Zähler für die Einhaltung der richtigen Pausenzeit: `SlotTimer` zählt die Dauer eines Slots, danach wird `PauseTimer` um eins reduziert. Erreicht letzterer Zähler null, wird die Pause beendet.

6.2.2.2 eth_transmitcontrol

Das an den Core über die WISHBONE-Master-Schnittstelle angeschlossene Modul kann den Core auch zum Senden eines Pausen-Frames veranlassen. Die Anfrage danach wird in ein Core-Register (`eth_registers`) zusammen mit der Pausendauer geschrieben (vgl. Abschnitt 6.2.5). `eth_transmitcontrol` erzeugt, wenn gerade kein anderes Frame gesendet wird, anhand dieses Wertes die Daten- und Statussignale (`ControlData TxCtrlStartFrm`, `TxCtrlEndFrm` usw.), die über den Multiplexer in `eth_maccontrol` dann weiter übertragen werden. Dazu besitzt

das Modul einen Byte-Zähler `ByteCnt`, mit dem die einzelnen Bytes des Pausenframes zum richtigen Zeitpunkt auf das Signal `ControlData` geschrieben werden.

6.2.3 eth_macstatus

Dieses Modul sammelt zahlreiche Informationen über den Status eines Sende- bzw. Empfangsvorgangs. Am Ende jeden Frames wird der Status an das `eth_wishbone`-Modul (Abschnitt 6.2.8) übertragen und dort gespeichert. Die Informationen über die empfangenen Daten stammen von dem Modul `eth_rx_ethmac` (Abschnitt 6.2.6) und beinhalten die Framelänge, die aktuelle Richtigkeit der CRC-32-Prüfsumme und die Daten- und Statussignale, wie sie vom Phy übermittelt werden. In `eth_macstatus` wird überprüft, ob die Framelänge einen gültigen Wert aufweist (`ReceivedLengthOK`). Die Richtigkeit der Prüfsumme wird am Ende des Frames zur weiteren Übermittlung zwischengespeichert. Die Phy-Signale werden ausgewertet, um zu überprüfen, ob ein überzähliges Nibble[1] oder ein ungültiges Symbol übertragen wurde. Ersteres kann allerdings wegen der 4-Bit-Datenbreite nur bei 10- und 100-Mbit/s-Ethernet vorkommen. Aus `eth_txethmac` (Abschnitt 6.2.7) stammen Informationen über aufgetretene Kollisionen und damit zusammenhängende Verzögerungenen beim Senden von Frames.

6.2.4 eth_miim

Dieses Modul realisiert den Management-Teil des MII zum Phy (vgl. Abschnitt 2.3.2). Es wird durch mehrere Core-Register gesteuert (Abschnitt 6.2.5). Die Lese- und Schreibbefehle, die Adresse und die zu sendenden sowie die erhaltenen Daten werden aus Core-Registern gelesen bzw. dorthin geschrieben. Das Modul selbst liefert vor allem Statusinformationen an die Core-Register und steuert die für die Kommunikation mit dem Phy zuständigen Untermodule.

6.2.4.1 eth_clockgen

Das Managment-Interface des MII wird mit einem langsameren Takt als das Daten-Interface betrieben (vgl. Tabelle 6.1 auf Seite 39). In `eth_clockgen` wird dieser Takt (MDC) mittels eines Zählers aus dem Takt der WB_CLK erzeugt, wobei das Verhältnis zwischen beiden über ein Core-Register eingestellt wird.

6.2.4.2 eth_outputcontrol

Das Modul `eth_outputcontrol` erhält die zum Phy zu sendenden Daten vom `eth-shiftreg`-Modul und synchronisiert diese mit einem Freigabesignal, dass die bidirektionale Datenleitung MDIO in die entsprechende Richtung schaltet.

[1] 1 Nibble = 4 Bits

6.2.4.3 eth_shiftreg

Den Hauptbestandteil dieses Moduls stellt ein Schieberegister mit jeweils seriellem und parallelem Ein- und Ausgang dar. Dessen Aufgabe ist es, bei Lesezugriffen auf Phy-Register aus der Phy-Adresse, der Adresse des Phy-Registers und den Daten ein Management-Frame zusammenzustellen und zu serialisieren. Dieses wird dann zum Phy gesendet. Bei Schreibzugriffen wird ebenfalls ein Frame mit Phy-Adresse und der Adresse des Phy-Registers zum Phy übertragen, der Datenteil des Frames wird jedoch vom Phy geschrieben. Die Daten werden dann von `eth_shiftreg` wieder zu einem 16-Bit-Datenwort parallelisiert, um abgespeichert werden zu können. Den Aufbau der beiden Frame-Typen ist in [15] erläutert.

6.2.5 eth_registers

Dieses Modul enthält alle Core-Register des gesamten Ethernet-Cores. Die einzelnen Register dienen seiner Steuerung und enthalten Informationen über seinen Status. In Anhang C sind alle Core-Register einzeln aufgeführt. Durch das WISHBONE-Slave-Interface kann über die entsprechende Adressierung auf sie zugegriffen werden. Um dies zu erreichen ist `eth_registers` mit den zu steuernden Modulen des Ethernet-Cores über Signale direkt verbunden. Die Signale steuern das betreffende Modul bzw. liefern Statusinformationen über dieses.

6.2.5.1 eth_register

Jedes Core-Register wird durch eine Instanz von `eth_register` erzeugt. Bei der Instanzierung eines Registers in `eth_registers` wird zusätzlich noch dessen Datenbreite und der Initialisierungswert über einen Parameter angegeben.

6.2.6 eth_rxethmac

Dieses Modul empfängt Daten vom MII Daten-Interface. Aus den 4-Bit-Daten vom Phy (`MRxD`) sowie dem synchron dazu anliegenden Gültigkeitssignal (`MRxDV`) werden die internen Signale `RxData`, `RxStartFrm`, `RxEndFrm` und `RxValid` generiert, die zu `eth_maccontrol` weitergeleitet werden. Diese Signale enthalten die Daten, den Anfang und das Ende eines Frames sowie die Gültigkeit der Daten, wobei hier schon die Präambel und der SFD entfernt wurden. Dazu werden die ankommenden Nibble mit den Hexadezimalwerten `5` und `d` an das Modul `eth_rxstatem` gemeldet. Außerdem wird überprüft, ob es sich bei dem ankommenden Frame um ein Broadcast- oder Multicast-Frame handelt (vgl. Abschnitt 2.3.3). Dies wird dem Modul `eth_addrcheck` mitgeteilt. Zur Überprüfung der CRC-32-Prüfsumme muss die Bit-Reihenfolge der Daten umgekehrt werden, da die Prüfsumme beginnend mit dem MSB erzeugt wird (vgl. Abschnitt 2.3.3).

6.2.6.1 eth_rxstatem

Der *endliche Zustandsautomat* (*FSM*[2]) in diesem Modul steuert alle Vorgänge im gesamten Modul `eth_rxethmac`. Gleichzeitig gibt er an, in welchem Zustand sich das Empfangsmodul befindet: Es wird unterschieden zwischen dem Leerlauf (`StateIdle`), dem Verwerfen von Daten (`StateDrop`), der Präambel (`StatePreamble`), dem SFD (`StateSFD`) sowie dem unterem (`StateData[0]`) und dem oberem Daten-Nibble (`StateData[1]`) eines Bytes.

6.2.6.2 eth_crc

Dieses Modul überprüft die Richtigkeit der CRC-32-Prüfsumme, um Fehler bei der Datenübertragung zu detektieren. Dazu wird die Prüfsumme zu einem bestimmten Zeitpunkt initialisiert und während des Datenempfangs anhand der Daten aus `eth_rxethmac` fortwährend neu berechnet. Die Berechnung selbst erfolgt mit einer Reihe von `XOR`-Verknüpfungen zwischen den einzelnen Bits der Prüfsumme und den Daten. Wurde ein Frame korrekt übertragen, so beträgt die CRC-32-Prüfsumme des Frames inkl. FCS (vgl. Abschnitt 2.3.3) immer `c704dd7b`, ein davon abweichender Wert wird als Übertragungsfehler detektiert.

6.2.6.3 eth_rxaddrcheck

Um nur die Frames weiterzureichen, die für den Host bestimmt sind, vergleicht dieses Modul die Zieladresse des Frames mit der in den Register MAC_ADDR0 und MAC_ADDR1 eingetragenen MAC-Adresse des Ethernet-Cores. Je nach Einstellungen in den Registern werden normalerweise Broadcast-Frames und die korrekten Unicast-Frames zugelassen (vgl. Abschnitt 2.3.3). Es besteht auch die Möglichkeit, den Promiscuous-Betrieb zu aktivieren, d.h. alle ankommenden Frames werden an die nächsthöhere Schicht weitergeleitet. Mit einer anderen Einstellung werden nur Unicast-Adressen akzeptiert. Sollte die Ziel-Adresse keiner gültigen Adressierung des Ethernet-Cores unter Berücksichtigung der Einstellungen entsprechen, so wird der Empfang des Frames abgebrochen.

6.2.6.4 eth_rxcounters

Dieses Modul enthält die beiden Zähler, die den Empfang von Frames betreffen. Ein Zähler ermittelt die Länge des empfangenen Frames, die dann in den Buffer-Deskriptoren gespeichert wird (vgl. Abschnitt 6.2.8). Der Andere zählt die Dauer des IFGs (vgl. Abschnitt 2.3.3), um einen genügend großen Abstand zwischen zwei Frames zu verifizieren, in dem sich der Empfänger auf den nächsten Frame vorbereiten kann.

[2] **F**inite **S**tate **M**achine

6.2.7 eth_txethmac

Dieses Modul ist zuständig für das Senden von Daten über das MII Daten-Interface. Es erhält die Daten (`TxData`) sowie den Beginn und das Ende eines Frames (`TxStartFrm` und `TxEndFrm`) vom Modul `eth_maccontrol` (vgl. Abschnitt 6.2.2). Neben den Daten werden auch noch die Präambel, der SFD und die FCS durch einen endlichen Zustandsautomat (FSM) zur Erzeugung der Signale für den Phy (`MTxD`, `MTxEn` und `MTxErr`) verwendet. Bei der FCS wird, wie auch in `eth_txethmac`, die Datenreihenfolge umgekehrt (vgl. Abschnitt 6.2.6). `eth_txethmac` gibt ständig Rückmeldung über den Sende-Status (`TxUsedData`, `TxDone`, `TxAbort` usw.), da während des Vorgangs kontinuierlich Daten erforderlich sind.

6.2.7.1 eth_txstatem

Die in `eth_txstatem` implementierte steuert den gesamten Vorgang des Moduls `eth_txethmac` zum Versenden von Frames. Sie unterscheidet dabei folgende Zustände: den Leerlauf (`StateIdle`), die Präambel (`StatePreamble`), das untere Nibble (`StateData[0]`), das obere Nibble (`StateData[1]`), das PAD (`StatePAD`), die FCS (`StateFCS`), den IFG (`StateIPG`), Stau (`StateJam` und `StateJam_q`), die Verzögerung von Frames (`StateBackOff`) und das Verschieben von Frames (`StateDefer`).

6.2.7.2 eth_crc

Dieses Modul ist identisch mit demjenigen aus dem `eth_rxethmac`-Modul (vgl. Abschnitt 6.2.6). Die CRC-32-Prüfsumme wird während des Sendens aus den Daten berechnet und dann direkt im Anschluss an das Frame angehängt.

6.2.7.3 eth_random

Für den Fall einer Kollision im Halbduplex-Betrieb wird das erneute Versenden von Frames um eine gewisse Zeit verzögert. Um erneute Kollisionen zu Vermeiden,hat die Zeitdauer einen zufälligen Wert. Dieser wird in `eth_random` mittels eines rückgekoppelten Schieberegisters berechnet.

6.2.7.4 eth_txcounters

Hier wird ähnlich wie in dem `eth_rxcounters`-Modul die Länge des Frames mitgezählt. Dabei existieren zwei Zähler: einer für die Anzahl an Nibbles (`NibCnt`) und einer für die Byte-Anzahl (`ByteCnt`), die gesendet werden. Spezielle Zählerstände werden mit einem zusätzlichen Signal gemeldet (`NibCntEq`...).

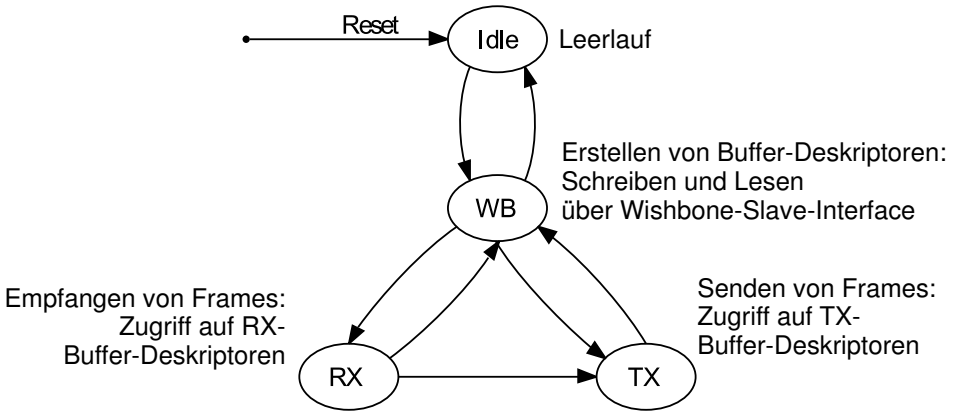


Abbildung 6.3: FSM für den Zugriff auf den Buffer-Deskriptor-Speicher. Im Normalbetrieb erfolgt ein ständiger Wechsel zwischen den Zuständen.

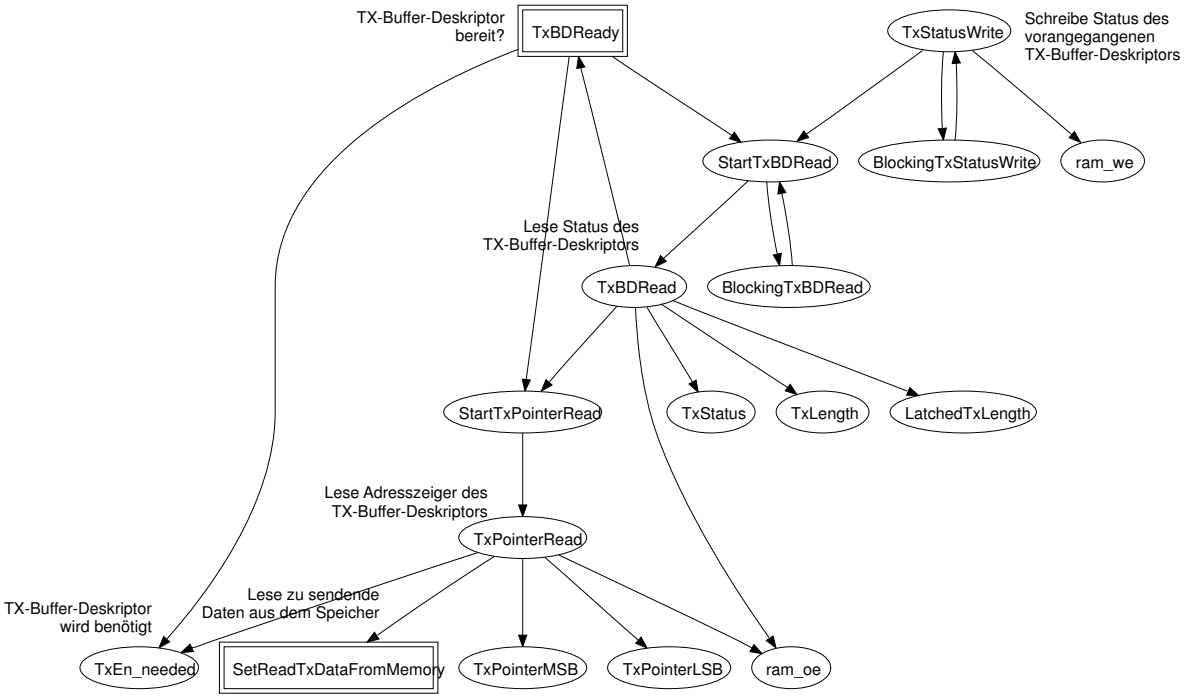
6.2.8 eth_wishbone

Das größte Modul des Ethernet-Cores bildet `eth_wishbone`. Es steuert das Senden und Empfangen von Frames mit Hilfe der sogenannten *Buffer-Deskriptoren*. Außerdem werden die Daten zwischen Daten-Speicher und Phy beim Senden und Empfangen in jeweils einem Fifo zwischengespeichert. Nachfolgend wird erklärt, was die Buffer-Deskriptoren sind, wie auf sie zugegriffen wird und wie sie abgearbeitet werden. Anschließend wird die Ansteuerung der Fifos und deren Rolle bei Sende- und Empfangsvorgängen erläutert.

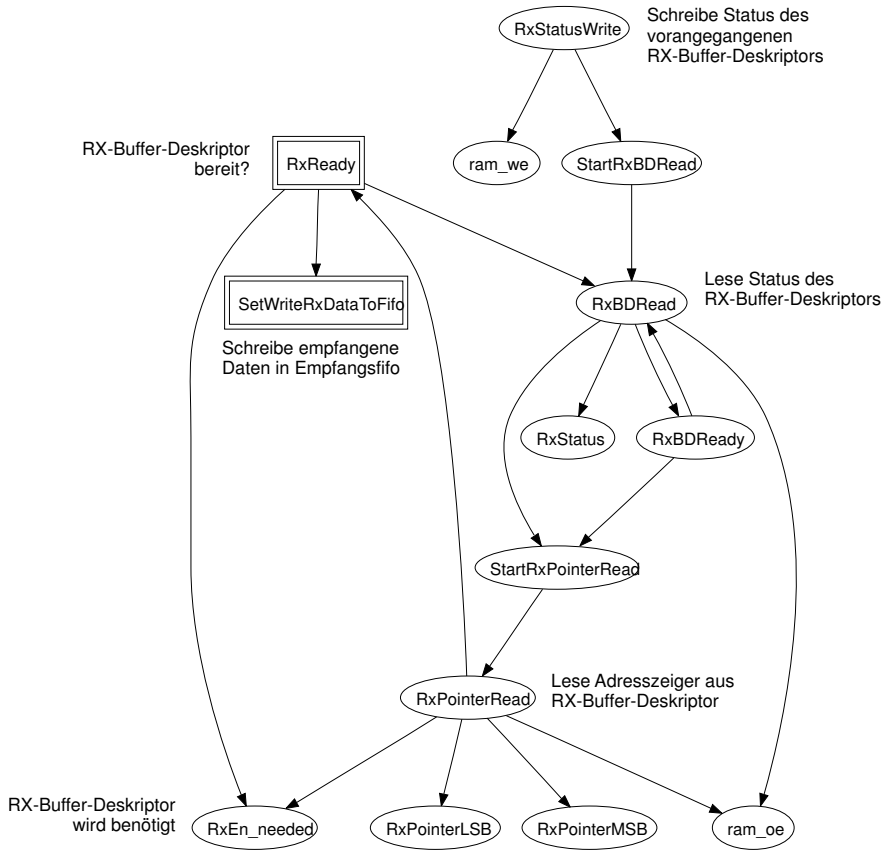
Ein Buffer-Deskriptor besteht aus zwei 32-Bit-Datenwörtern und beinhaltet die komplette Information zu einem Frame [22]. Die ersten vier Bytes beinhalten verschiedene Statusinformationen und unterscheiden sich je nachdem, ob der Buffer-Deskriptor zu einem Frame gehört, das gesendet wird (TX-Buffer-Deskriptor), oder zu einem, das empfangen wird (RX-Buffer-Deskriptor). Die nächsten vier Bytes des entsprechenden Buffer-Deskriptors speichern einen Zeiger auf die Adresse im RAM, von wo aus die Daten gelesen werden bzw. wo sie hingespeichert werden.

Die TX-Buffer-Deskriptoren beinhalten die Anzahl an Datenbytes, die gesendet werden sollen, die Information, ob der Deskriptor zum Senden bereit ist und weitere Informationen über das zu sendende Frame. Nach dem Senden werden darin zusätzlich noch Statuswerte gespeichert, z.B. ob es zu Kollisionen gekommen ist. Bei den Rx-Buffer-Deskriptoren wird eingetragen, ob der Buffer-Deskriptor leer ist. Wurde ein Frame empfangen, so wird dessen Länge und ob es zu Übertragungsfehlern (ungültige Symbole, CRC-Fehler usw.) gekommen ist, im zugehörigen RX-Buffer-Deskriptoren gespeichert.

Über ein Core-Register kann festgelegt werden, wie viele Plätze TX- und RX-Buffer-Deskriptoren im gemeinsamen Speicher zur Verfügung stehen. Die beiden Typen von Buffer-Deskriptoren bilden jeweils für sich einen Ring: Beginnend mit dem Buffer-Deskriptor mit der niedersten Adresse werden diese nacheinander abgearbeitet und deaktiviert. Nach dem letzten geht es wieder mit dem ersten weiter, falls dieser in der Zwischenzeit neu beschrieben wurde. Ist ein Buffer-Deskriptor nicht bereit, gesendet zu werden oder zu empfangen, so wird der entsprechende Vorgang unterbrochen, bis sich sein Status geändert hat.



(a) Zugriff auf einen TX-Buffer-Deskriptor während des TX-Zustandes



(b) Zugriff auf einen RX-Buffer-Deskriptor während des RX-Zustandes

Abbildung 6.4: Steuer-Signale für Zugriffe auf die Buffer-Deskriptoren

Die Buffer-Deskriptoren lassen sich über einen bestimmten Adressbereich des WISHBONE-Slave-Interface beschreiben und auslesen. Auch die für Senden und Empfangen zuständigen Teile des `eth_wishbone`-Moduls greifen auf die Buffer-Deskriptoren zu. Aus diesem Grund regelt dort die in Abbildung 6.3 gezeigte FSM den Zugriff. Sie besteht aus den vier Zuständen `Idle`, `WB`, `RX` und `TX`. Jeder Zustand ist genau zwei Takte aktiv. Im Normalfall wechselt die FSM zwischen `Idle` und `WB`. Während des `WB`-Zustandes erhält das WISHBONE-Slave-Interface die Möglichkeit eines Zugriffs auf den Speicher der Buffer-Deskriptoren. Für das vollständige Lesen bzw. Schreiben eines Buffer-Deskriptors sind dabei wegen der 32-Bit-Busbreite zwei Zugriffe nötig.

Ein Frame wird nur gesendet, wenn ein TX-Buffer-Deskriptor bereit ist. Ist keiner verfügbar, muss auf den Speicher zugegriffen und der nächste ausgelesen werden. Ein TX-Buffer-Deskriptor, der bereit ist, wird durch `TxBDReady` angezeigt. Ist dies nicht der Fall, signalisiert `TxEn_needed` den erforderlichen Zugriff auf den Buffer-Deskriptor-Speicher. Der Zusammenhang zwischen den genannten und den folgenden Signalen wird der Übersichtlichkeit wegen in Abbildung 6.4(a) dargestellt. Ein Zugriff ist notwendig, wenn gerade kein Frame gesendet wird und direkt nach dem Initialisieren des Ethernet-Cores. Die FSM wechselt daraufhin vom `WB`-Zustand in den `TX`-Zustand. Wurde zuvor schon ein Frame gesendet, wird dessen Status in den alten Buffer-Deskriptor geschrieben (`TxStatusWrite`). Nach einem erneuten Zustandswechsel wird der Status des nächsten TX-Buffer-Deskriptors für das darauffolgende Frame gelesen (`TxBDRead`). Der zum Frame gehörende Status sowie seine Länge werden gespeichert (`TxStatus` und `TxLength`). Signalisiert der Status Bereitschaft, so wird im darauffolgenden `TX`-Zustand der Zeiger ausgelesen (`TxPointerRead`) und gespeichert (`TxPointerMSB` und `TxPointerLSB`). Damit endet der Zugriff auf den Buffer-Deskriptor-Speicher, `TxEn_needed` wird deaktiviert. Die zu sendenden Daten werden jetzt aus dem RAM gelesen (`SetReadTxDataFromMemory`) und im Anschluss daran das Frame gesendet.

Das Empfangen von Frames und der Zugriff auf einen RX-Buffer-Deskriptor verlaufen ähnlich wie im TX-Fall. Die ausschlaggebenden Signale sind in Abbildung 6.4(b) aufgeführt. Ein Frame kann nur empfangen werden, wenn ein Buffer-Deskriptor dazu bereit ist. Das wird durch `RxReady` angezeigt. Ist kein RX-Buffer-Deskriptor bereit, wird auch hier ein Zugriff auf den Speicher zum Lesen des nächsten notwendig. Durch `RxEn_needed` wechselt der Zustand der FSM von `WB` nach `RX`. Das geschieht auch, wenn sowohl `RxEn_needed` als auch `TxEn_needed` aktiv sind. Die FSM springt dann jedoch nach dem `RX-Zustand` nicht zurück in den `WB`-, sondern in den `TX`-Zustand. Nach einem empfangenen Frame wird dessen Status und seine Länge in den vorherigen RX-Buffer-Deskriptor geschrieben (`RxStatusWrite`) und nach einem Zustandswechsel der Status des folgenden Buffer-Deskriptors gelesen (`RxBDRead`). Ist der Buffer-Deskriptor bereit, erfolgt nach einem weiteren Wechsel des Zustandes der FSM das Auslesen des Zeigers. Dann wird `RxEn_needed` deaktiviert und der Ethernet-Core ist in der Lage, Frames zu empfangen und zwischenzuspeichern (`SetWriteRxDataToFifo`). Frames, die ankommen während `RxReady` nicht aktiv ist, werden verworfen.

Die Zwischenspeicherung in `eth_wishbone` geschieht in zwei Fifos. Sie speichern die zu sendenden Daten aus dem RAM (`tx_fifo`) bzw. die empfangenen Daten bevor sie ins RAM geschrieben werden (`rx_fifo`). Die beiden identischen Fifos besitzen je einen 32 Bit breiten Dateneingang und Datenausgang. Sie können bis zu 16 Datenwörter speichern und verfügen über mehrere Füllstandssignale (`empty`, `almost_empty`, `full` und `almost_full`) und einen Füllstandszähler (`cnt`). Beide Fifos sind über das WISHBONE-Master-Interface mit dem RAM

verbunden. Ob TX- oder RX-Fifo die Schnittstelle benutzt, wird wiederum durch eine FSM geregelt.

Das Senden von Frames geschieht folgendermaßen: Ist ein TX-Buffer-Deskriptor bereit, wird das Signal `SetReadTxDataFromMemory` gesetzt. Dadurch wird die FSM dazu veranlasst, von der durch den Zeiger angegebenen Adresse im RAM die Menge an Daten zu lesen, die durch `TxLength` vorgegeben wird. Dazu sind mehrere Lese-Zugriffe auf das RAM notwendig. Anhand des Füllstandszählers wird entschieden, ob ein Einzelzugriff oder mehrere direkt aufeinanderfolgende Zugriffe (*Burst*) erfolgen. Erreicht der Füllstand eine gewisse Schwelle oder sind alle Daten gelesen, beginnt die Übertragung der Daten. Die 32-Bit-Datenworte werden auf 8 Bit verteilt (`TxData`) und zusammen mit den Statussignalen `TxStartFrm` und `TxEndFrm` zum `eth_maccontrol`-Modul übertragen (vgl. Abschnitt 6.2.2). Falls der Fifo vor Ende das Frames leerläuft, wird ein Fehler signalisiert (`TxUnderRun`). Nach dem Senden erfolgt ein Zugriff auf den zugehörigen TX-Buffer-Deskriptor, wobei der Status des gesendeten Frames in diesen geschrieben wird (`TxStatusWrite`). Direkt im Anschluss wird der nächste TX-Buffer-Deskritpor gelesen.

Empfangene Frames werden nur bei aktivem Signal `SetWriteRxDataToFifo` in das `rx_fifo` geschrieben, nachdem die Daten zuvor zu 32 Bits zusammengefasst wurden. Sobald Daten darin enthalten sind, werden diese, durch die zuständige FSM gesteuert, in den Daten-Speicher geschrieben. Dabei wird die im Buffer-Deskriptor gespeicherte Adresse verwendet. Sollte dieser Fifo überlaufen, wird das durch `RxOverrun` gemeldet. Nach Ende des Frames werden dessen Status-Informationen im RX-Buffer-Deskriptor gespeichert und der nächste Empfang durch Lesen des folgenden Buffer-Deskriptors vorbereitet.

In `eth_wishbone` werden außerdem die Signale, die von `eth_maccontrol` kommen, synchronisiert, da sie eine Taktgrenze überschreiten (vgl. Abschnitt 6.3). Daneben werden noch einige Interrupt-Signale erzeugt, die über Register ausgelesen werden können.

6.2.8.1 eth_spram_256x32

Dieses Modul bildet den Speicher für die Buffer-Deskriptoren. Es benutzt dafür mehrere im FPGA vorhandene Block SelectRAM-Zellen (vgl. Abschnitt 3.4), die über das Modul `RAMB4_S8` instantiiert werden. Insgesamt können 128 Buffer-Deskriptoren gespeichert werden.

6.2.8.2 eth_fifo

Dieses Modul implementiert die in `eth_wishbone` verwendeten Fifos. Dazu erlaubt es die Auswahl zwischen verschiedenen FPGA-abhängigen Speichertypen. Der in dieser Arbeit verwendete FPGA benutzt `xilinx_dist_ram_16x32`.

xilinx_dist_ram_16x32
Der Speicher eines Fifos ist, ähnlich wie der Speicher der Buffer-Deskriptoren, aus mehreren Speicherzellen zusammengesetzt. Hier werden viele einzelne Distributed SelectRAM-Einheiten (vgl. Abschnitt 3.4) verbunden. Diese werden jeweils durch das Modul `RAM16X1D` instantiiert.

Takt	10 Mbit/s	100 Mbit/s	1000 Mbit/s	Aufgabe
MDC	$\leq$ 2,5 MHz	$\leq$ 2,5 MHz	$\leq$ 2,5 MHz	MII Management
WB_CLK	78 MHz	78 MHz	125 Mhz	Steuerung des Ethernet-Cores
(G)TX_CLK	2,5 MHz	25 MHz	125 MHz	MII, Versenden von Daten
RX_CLK	2,5 MHz	25 MHz	125 MHz	MII, Empfangen von Daten

Tabelle 6.1: Übersicht über die verschiedenen Takte und deren Frequenz (TX_CLK und RX_CLK sind nur nominell gleich, während bei Gigabit-Ethernet WB_CLK und GTX_CLK identisch sind.)

6.2.9 eth_defines

Dieses Modul enthält alle Definitionen für den Ethernet-Core. Es beinhaltet vor allem die Adressen der Core-Register (vgl. Anhang C) sowie einige Parameterwerte für die Daten-Busbreiten. Auch die Auswahl des FPGAs wird hier getroffen.

6.3 Takt-Domänen des Ethernet-Cores

Nach der Beschreibung der einzelnen Module wird in diesem Abschnitt, deren Taktung erläutert. Der Ethernet-Core lässt sich in einzelne Bereiche unterteilen, die unterschiedlich getaktet werden (*Takt-Domänen*). Das wird in Abbildung 6.2 auf Seite 28 mit Hilfe der Farben angedeutet, zusätzlich gibt Tabelle 6.1 die Frequenzen der verschiedenen Takte an.

Der kleinste der Bereiche verwendet die MDC zur Kommunikation über das MII Management-Interface mit dem Phy (vgl. Abschnitt 2.3.2). Dieser Takt existiert nur im `eth_miim`-Modul, in dem der Takt erzeugt wird (vgl. Abschnitt 6.2.4).

Zur Steuerung des Ethernet-Cores wird die WB_CLK verwendet, die er über die WISHBONE-Schnittstelle erhält. Dieser Takt stellt den System-Takt für den Ethernet-Core dar, denn er wird für alle Module verwendet, die nicht speziell für Senden oder Empfangen zuständig sind: Das sind `eth_miim`, `eth_registers` und `eth_wishbone` (vgl. Abschnitt 6.2.4, 6.2.5 und 6.2.8). Im ersten Modul wird damit der Datenpfad zu den Core-Registern und die zugehörigen Signale getaktet. Alle Registerzugriffe in `eth_registers` sind zu diesem Takt synchron, also sowohl die internen Zugriffe durch andere Module des Ethernet-Core als auch die externen über die WISHBONE-Slave-Schnittstelle. In `eth_wishbone` taktet WB_CLK auch die externen Zugriffe auf die Buffer-Deskriptoren, die ebenfalls über diese Schnittstelle erfolgen. Außerdem sind die internen Lese- und Schreibzugriffe auf die Buffer-Deskriptoren, die beim Senden und Empfangen von Frames auftreten, dazu synchron. Auch die Taktung der Fifos und der Zugriff auf sie, um Daten über das WISHBONE-Master-Interface ins RAM zu übertragen bzw. von dort zu lesen, erfolgt mit der WB_CLK.

Alle Empfangssignale, die vom Phy zum Ethernet-Core gesendet werden, sind zu der RX_CLK (im Ethernet-Core-Design als MRX_CLK bezeichnet) synchron (vgl. Abschnitt 2.3.2). Dieser Takt wird vom Phy aus den empfangenen Daten gebildet und wird entlang des gesamten Pfades der Empfangssignale bis hin zum Modul `eth_wishbone` verwendet. Der Weg beinhaltet die Module `eth_rxethmac`, `eth_maccontrol` und `eth_wishbone` (vgl. Abschnitt 6.2.6, 6.2.2 und

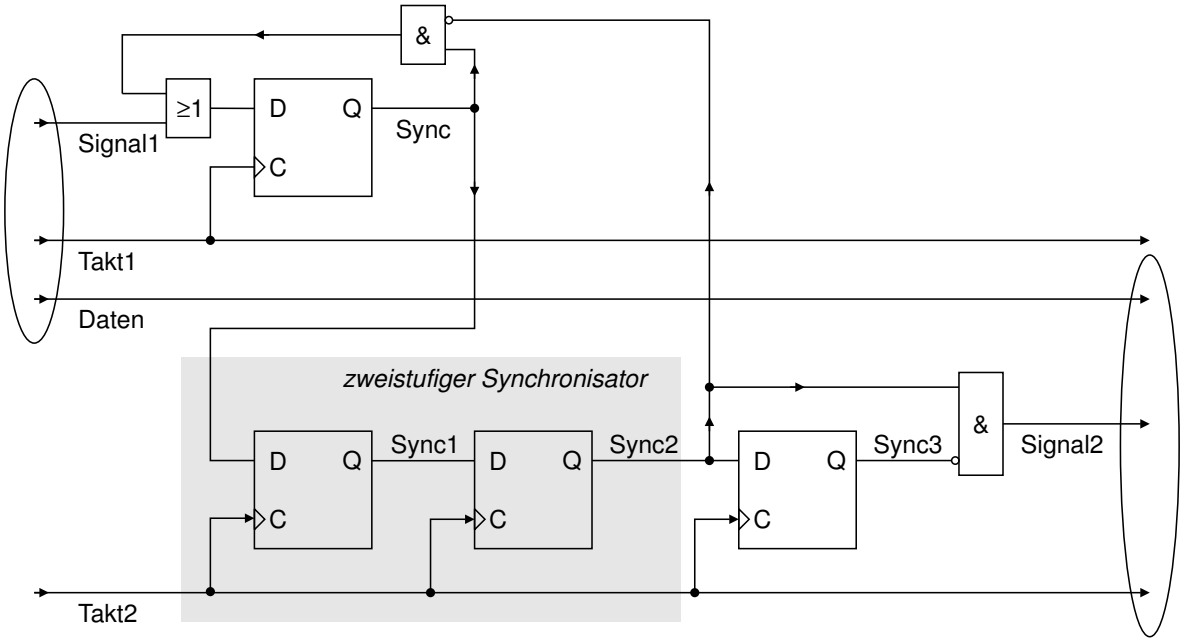


Abbildung 6.5: Synchronisation im Ethernet-Core

6.2.8). In Letzterem werden die Daten dann in den Empfangsfifo (`rx_fifo`) geschrieben, der mit der WB_CLK getaktet ist.

Normalerweise müssen alle Signale, die eine Taktgrenze überschreiten, synchronisiert werden. Ausnahme sind Signale, die mit Sicherheit zu dem Zeitpunkt, zu dem sie gelesen werden, stabil sind. Ansonsten kann dies zu metastabilen Signalen und damit zu Fehlern in der Weiterverarbeitung führen [19]. Die Daten selbst werden nicht synchronisiert, sondern nur die entsprechenden Statussignale über einen Synchronisator [5]. Dieser wird auch an anderen Stellen zur Überquerung von Taktgrenzen verwendet und ist deshalb in Abbildung 6.5 dargestellt. Man erkennt, dass die Daten, die wie `Signal1` synchron zu `Takt1` sind, mindestens für zwei Periodenlängen von `Takt2` stabil am Fifo anliegen, bevor sie über `Signal2` synchron zu `Takt2` in den Fifo geschrieben werden.

Außerhalb des Datenpfades werden die den Empfang betreffenden Statussignale ebenfalls mit RX_CLK getaktet. Dies ist der Fall in den Modulen `eth_macstatus` und `eth_registers` (vgl. Abschnitt 6.2.3 und 6.2.5). Da die Core-Register mit der WB_CLK getaktet sind, wird dort das gleiche Prinzip zur Synchronisation verwendet. Dabei entspricht das Statussignal `Signal1` und das synchronisierte `Signal2` wird in das Core-Register geschrieben. Ein Datensignal existiert in diesem Fall nicht.

Damit Daten versendet werden können, liefert der Phy die TX_CLK (im Ethernet-Core-Design MTX_CLK genannt). Mit ihr werden die Signale synchronisiert, die der Ethernet-Core zum Phy sendet. Für diese Takt-Domäne gilt das gleiche wie für den Empfangsteil. Der Datenpfad beinhaltet die Module `eth_txethmac`, `eth_maccontrol` und `eth_wishbone`, wobei in Letzterem der Pfad am Sendefifo beginnt (vgl. Abschnitt 6.2.7, 6.2.2 und 6.2.8). Auch hier werden die entsprechenden Statussignale mit der TX_CLK getaktet und gegebenenfalls über

Taktgrenzen hinweg synchronisiert. Das betrifft wie bei RX_CLK das `eth_macstatus`- und das `eth_registers`-Modul.

7 Erweiterung des Ethernet-MACs auf 1-Gbit/s-Ethernet

In diesem Kapitel werden die Anpassungen beschrieben, die in der Arbeit geleistet wurden, um den Ethernet-Core mit 1 Gbit/s betreiben zu können. Dazu gehört das Hinzufügen von neuen Modulen, um die Funktionalität zu erweitern. Um zwischen dem verwendeten IP-Core, der ein MAC für 10- und 100-Mbit/s-Ethernet ist, und dem erweiterten MAC für Gigabit-Ethernet unterscheiden zu können, wird wie im vorigen Kapitel ersterer mit Ethernet-Core und letzterer mit Ethernet-MAC bezeichnet.

Die Veränderungen am Ethernet-Core lassen sich sich in zwei Bereiche unterteilen: Die generellen Modifikationen für den Einsatz des Ethernet-Cores und die für den Betrieb mit Gigabit-Ethernet notwendigen Änderungen. Die generellen Modifikationen wurden bei 100-Mbit/s-Ethernet durchgeführt und werden in Abschnitt 7.1 bis 7.6 beschrieben. Es wurde ein neues oberstes Modul erstellt, der Ethernet-MAC. Darin wurden die Schnittstellen des Ethernet-Cores an die angeschlossenen Module angepasst. Um das Senden und Empfangen von Frames zu erleichtern, wurde ein Modul implementiert, das die dafür nötigen Buffer-Deskriptoren automatisch generiert. Außerdem wurde Logik zum Ethernet-Core hinzugefügt, die den Frames beim Senden die richtige Adressierung anfügt und beim Empfangen diese aus den Frames entfernt.

Die Änderungen, die speziell für Gigabit-Ethernet durchgeführt wurden, werden in Abschnitt 7.7 bis 7.10 dargestellt. Dazu musste die Breite der Datenbusse erhöht werden. Weiter erforderten die höheren Geschwindigkeiten bei Gigabit-Ethernet eine schnellere Erzeugung der Buffer-Deskriptoren sowie einen schnelleren Lese-Zugriff auf das DDR-SDRAM. In Abschnitt 7.11 wird anschließend das Konzept vorgestellt, wie der noch nicht implementierten Logik zur Steuerung des Systems Befehle von Ethernet-Frames auswertet. Damit verbunden ist die Behandlung von Frames mit falscher CRC-32-Prüfsumme durch den Ethernet-MAC.

Im Gegensatz zu den Ethernet-Core-Modulen sind die erstellten Module in VHDL geschrieben. Der Grund dafür ist, dass diese Sprache in den anderen Modulen der Arbeitsgruppe verwendet wird. Die Verilog HDL-Module des Ethernet-Cores können jedoch ohne Probleme in die VHDL-Module eingebunden werden.

7.1 Oberstes Modul des Ethernet-MACs (n_ethernet)

Für den Einsatz des Ethernet-Cores war es zunächst notwendig, dessen Schnittstellen an das vorhandene System anzupassen. Dies geschah durch zusätzliche Module außerhalb des ursprünglichen Ethernet-Cores (vgl. Abbildung 7.1). Das oberste davon ist das Modul `n_ethernet`. Es stellt die höchste Hierarchie-Stufe des Ethernet-MACs dar und beinhaltet die

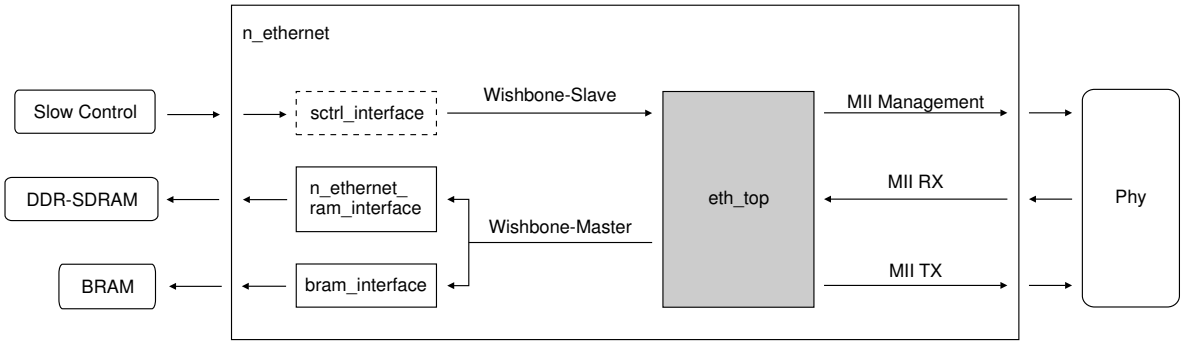


Abbildung 7.1: Ethernet-MAC mit dem darin enthaltenen Ethernet-Core

verschiedenen Schnittstellen und den Ethernet-Core mit dem Modul `eth_top`. Die Schnittstellen erlauben Verbindungen zu Slow-Control, DDR-SDRAM, SelectRAM des FPGAs und GMII zum Phy (vgl. Abschnitt 5.2 und 5.3). `n_ethernet` wurde in einer ersten Version von Sebastian Millner im Rahmen seiner Miniforschungsarbeit für 10- und 100-Mbit/s-Ethernet erstellt [20]. Da in `n_ethernet` zwei verschiedene Module (`n_ethernet_ram_interface` und `bram_interface`) an das WISHBONE-Master-Interface angeschlossen werden können, besteht die Möglichkeit über das Setzen einer Konstante zwischen der Verwendung des DDR-SDRAMs und des SelectRAMs zu wählen.

7.2 Schnittstelle zur SlowControl (n_ethernet)

Die Slow-Control wird zur Steuerung des Ethernet-MACs eingesetzt (vgl. Abschnitt 5.2 und 5.3). Wie in Abschnitt 6.2.8 erläutert wurde, besitzt der Ethernet-Core dafür das WISHBONE-Slave-Interface. Da die Slow-Control nicht direkt mit dem WISHBONE-Interface kompatibel ist, musste ein Modul geschrieben werden, das die beiden miteinander verbindet. Sebastian Millner hat auch dieses erstellt [20]. Statt in einem eigenen Modul ist die Schnittstelle im obersten Modul des Ethernet-MACs (`n_ethernet`) integriert und besteht aus einer FSM mit den Zuständen `idle` und `wait_ack`. Die FSM sorgt für die richtigen Handshake-Signale auf Seiten des Ethernet-Cores und der Slow-Control (vgl. Abschnitt 3.1). Die Schnittstelle konnte für die Arbeit fast unverändert übernommen werden.

7.3 SDRAM- bzw. SelectRAM-Schnittstelle (n_ethernet_ram_interface und bram_interface)

Neben der Slow-Control-Schnittstelle werden in `n_ethernet` zwei Module eingebunden, die mit der WISHBONE-Master-Schnittstelle des Ethernet-Cores verbunden sind. Nur auf dem Nathan-Modul steht sowohl DDR-SDRAM als auch das SelectRAM des FPGAs zur Verfügung, d.h. nur dort ist es möglich, zwischen den beiden Speichern zu wählen (vgl. Abschnitt 5.2). Auf der Backplane hingegen muss das SelectRAM zur Datenspeicherung genutzt werden (vgl. Abschnitt 5.3).

DDR-SDRAM
Das erste Modul (`n_ethernet_ram_interface`) stellt die Schnittstelle zu dem von Tillmann Schmitz in seiner Doktorarbeit implementierten RAM-Modul (`manager_nat_sdram`) dar, das das DDR-SDRAM benutzt [32]. Diese Schnittstelle wurde in einer ersten Version für den Einsatz mit 10- und 100-Mbit/s-Ethernet ebenfalls von Sebastian Millner erstellt [20]. Hier steuert eine FSM die DDR-SDRAM-Zugriffe des WISHBONE-Master-Interfaces mittels der Zustände `idle`, `ready`, `ready_suspend`, `read_request`, `read` und `write`. Die Komplexität der FSM ergibt sich aus den Anforderungen, die das RAM-Modul stellt. Der Datenaustausch erfolgt über Handshake-Signale (vgl. Abschnitt 3.1), die den folgenden Bedingungen unterliegen: Die Adressierung der RAM-Daten erfolgt in 64-Bit-Schritten, was der Breite der Datenleitung zum RAM entspricht. Da die Datenleitung im Ethernet-MAC jedoch nur 32 Bits breit ist, muss die FSM Zugriffe auf benachbarte RAM-Adressen zusammenfassen. Darüber hinaus können Lesezugriffe auf das RAM nur auf gerade Adressen erfolgen und es werden 2 x 64 Bits in zwei aufeinanderfolgenden Takten als Antwort zurückgeliefert. Schreibzugriffe müssen paarweise beginnend mit einer geraden Adresse auftreten, d.h. es müssen 2 x 64 Bits auf benachbarte Adressen geschrieben werden [32]. Auch diese beiden Bedingungen werden durch die FSM eingehalten, indem sie die empfangenen Daten bis 128 Bits in einem Register zwischenspeichert. Die FSM wurde im Rahmen der Arbeit so verändert, dass die letzten 32 Bits von empfangenen Frames nicht mit abgespeichert werden, sonst würde zusätzlich zu den Daten auch noch die CRC-32-Prüfsumme in das DDR-SDRAM geschrieben.

Bei 10- und 100-Mbit/s-Ethernet konnte das `n_ethernet_ram_interface`-Modul bis auf die Änderung wegen der Prüfsumme unverändert verwendet werden, bei 1 Gbit/s jedoch erwies es sich in seiner vorliegenden Form als zu langsam. Die Modifikationen für Gigabit-Ethernet werden in Abschnitt 7.9 beschrieben.

SelectRAM
Das Modul `bram_interface`, das in dieser Arbeit erstellt wurde, verbindet den Handshake des WISHBONE-Master-Interfaces mit dem Modul `bram_ethernet_dp` (vgl. 3.1). Dieses verwendet Block SelectRAM des FPGAs und wurde mit dem CORE Generator-Programm von Xilinx erzeugt (vgl. Abschnitt 3.4). Dadurch lässt sich die Größe des Speichers problemlos den Anforderungen anpassen. Der Speicher besitzt neben der Schnittstelle für den Ethernet-MAC eine zweite Schnittstelle, die für den Zugriff durch die Slow-Control vorgesehen ist. Das Modul `bram_ethernet_dp` ist aus dem Grund nicht in den Ethernet-MAC integriert, da es unabhängig von ihm von der Slow-Control genutzt werden kann.

7.4 Automatische Generierung der Buffer-Deskriptoren (eth_registers und eth_wishbone)

Durch die bisher genannten Zusatzmodule war es möglich, Frames in Verbindung mit dem System bei 100 Mbit/s zu versenden und zu empfangen. Jedoch mussten dazu die Buffer-Deskriptoren für jeden Frame einzeln beschrieben werden. Da das auf Dauer und vor allem bei größeren Datenmengen nicht praktikabel ist, sollten die Buffer-Deskriptoren automatisch erzeugt werden. In einer ersten Variante hat der auf dem FPGA vorhandene PowerPC diese Aufgabe übernommen und mittels eines C-Programms die Buffer-Deskriptoren überwacht. Ist

ein Buffer-Deskriptor ausgeführt worden, hat das Programm einen neuen mit den entsprechenden Werten beschrieben. Schon die Simulation hat aber gezeigt, dass die Geschwindigkeit selbst schon bei 100 Mbit/s viel zu gering ist: Die Generierung eines Buffer-Deskriptors mit dem Power-PC dauerte etwa 3,8 μs (300 Takte bei 78 MHz). Der Inter Frame Gap von Ethernet hingegen beträgt weniger als 1 μs (24 Takte bei 25 MHz). Diese Zeit steht dem Ethernet-Core zur Verfügung, um sich auf den Empfang des nächsten Frames vorzubereiten.

Aus diesem Grund wurde der Ethernet-Core so verändert, dass die Buffer-Deskriptoren automatisch erzeugt werden. Das geschieht in zwei Schritten: Zuerst werden die Werte für den nachfolgenden Buffer-Deskriptor erzeugt. Diese werden dann an die entsprechende Stelle im Buffer-Deskriptor-Speicher geschrieben und gleich darauf ausgelesen, um ein weiteres Frame senden bzw. zu empfangen zu können. Sieben neue Core-Register in `eth_registers` dienen dabei der Steuerung: TX_RAM_L, TX_RAM_H, TX_BD_DEF, RX_RAM_L, RX_RAM_H, RX_BD_DEF und BD_INT. Über die Register RAM_H und RAM_L werden die obere und untere Grenze des Speicherbereichs angegeben, der gesendet werden soll (TX_...) bzw. in den Daten gespeichert werden können (RX_...). Die beiden BD_DEF-Register geben jeweils die Standardwerte an, mit denen die Status-Felder der Buffer-Deskriptoren beschrieben werden. Das letzte Register BD_INT signalisiert mit den beiden niederwertigsten Bits, wenn der vorgegebene Speicherbereich aufgebraucht ist (TX: Bit[1], RX: Bit[0]). Wird dann kein neuer Speicherbereich angegeben oder der alte nicht zum Überschreiben freigegeben, stoppt je nach Bit das Senden oder Empfangen von Frames. Die Freigabe erfolgt durch Zurücksetzen des entsprechenden Bits in BD_INT auf Null. Die im Modul `eth_registers` stehenden Registerwerte werden in das `eth_wishbone` geführt, wo die Erzeugung der Buffer-Deskriptoren stattfindet.

Generierung der Buffer-Deskriptor-Werte
Die automatische Generierung eines TX-Buffer-Deskriptors funktioniert folgendermaßen: Sobald das Signal `TxEn_needed` in `eth_wishbone` einen benötigten Zugriff auf die TX-Buffer-Deskriptoren anzeigt und der Schreibzugriff auf den vorangegangenen Deskriptor abgeschlossen ist, wird das Signal `gen_txbd` auf `1` gesetzt (vgl. Abschnitt 6.2.8). Gleichzeitig werden die WISHBONE-Eingänge der Schreibsignale der FSM über einen Multiplexer auf die Signale umgeleitet, die die automatisch generierten Werte beinhalten. Im darauf folgenden Takt werden Status (`tx_st`) und Adresse (`tx_adr`) des neuen Buffer-Deskriptors erstellt. Der Status wird dabei aus dem Registerwert von TX_BD_DEF übernommen und die Adresse, beginnend mit der Speicher-Untergrenze TX_RAM_L, um die Framelänge hochgezählt. Dabei ist zu beachten, dass der Speicherzugriff eine 16-Byte-Granularität besitzt (entspricht zwei Speicheradressen mit je 64 Bits). Wird die Speicher-Obergrenze TX_RAM_H erreicht, so wird Bit 1 des BD_INT-Registers auf `1` gesetzt und die Generierung der TX-Buffer-Deskriptoren abgebrochen. Ist die Adresse hingegen im gültigen Bereich, werden die erzeugten Status- und Adresswerte mit den nächsten WISHBONE-Zugriffen der FSM in den Buffer-Deskriptor geschrieben. Nach dem Schreib-Zugriff wird das Steuersignal `gen_txbd` wieder auf `0` zurückgesetzt und die Umleitung aufgehoben. Dadurch kann wieder über das WISHBONE-Slave-Interface auf die Buffer-Deskriptoren zugegriffen und diese bei Bedarf ausgelesen werden. Der beschriebene Vorgang gilt auch für die RX-Buffer-Deskriptoren, nur dass dort die neue Adresse anhand der Länge des zuvor empfangenen Frames berechnet wird.

Schreiben und Auslesen der Buffer-Deskriptoren
Abbildung 7.2 zeigt, wie der Zugriff auf den Speicher der Buffer-Deskriptoren genau abläuft.

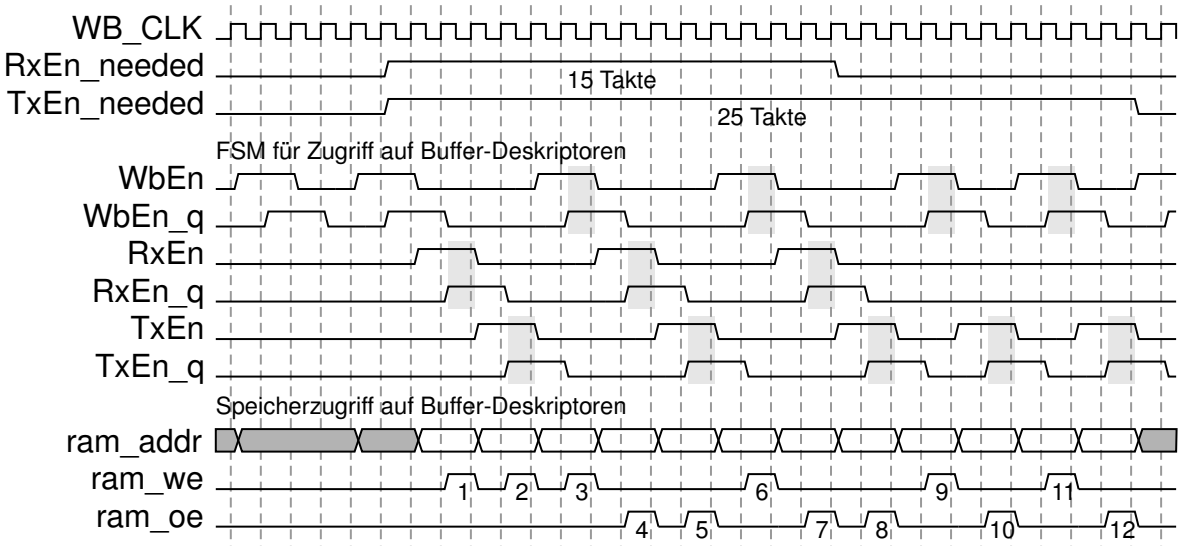


Abbildung 7.2: Zugriff auf die Buffer-Deskriptoren mit der FSM des Ethernet-Cores

Im oberen Teil sind der jeweilige Zustand der FSM (...`En_q`) und die Signale, die einen Zustandswechsel einleiten (...`En`) dargestellt. Im unteren Teil lässt sich erkennen, ob es sich um einen Lese- (`ram_oe`[1]) oder um einen Schreib-Zugriff (`ram_we`[2]) auf den Speicher der Buffer-Deskriptoren handelt. Außerdem lässt sich daran ablesen, wie lange dies im schlechtesten Fall, bei gleichzeitiger Anfrage beider Buffer-Deskriptoren, dauert. Nur zu bestimmten Zeitpunkten ist eine Änderung der Signale `TxEn_needed` und `RxEn_needed`, die den Bedarf eines Zugriffs auf die Buffer-Deskriptoren anzeigen, auf `1` möglich, nämlich genau dann, wenn `WbEn 1` und `WbEn_q` noch `0` ist. Die Zugriffe auf die Buffer-Deskriptoren in den einzelnen Zuständen der FSM sind hellgrau schattiert. Auch sie finden nur in den Takten statt, in denen sowohl ...`En` als auch ...`En_q` für den jeweiligen Zustand `1` sind. Auf jeden Zugriff erfolgt ein Zustandswechsel.

Der Ablauf des Speicher-Zugriffs ist wie folgt: Wird Zugriffsbedarf sowohl von Sende- als auch von Empfangsseite angemeldet (`TxEn_needed = 1` und `RxEn_needed = 1`), so wechselt die FSM nach dem WB-Status in den RX-Status, da dieser gegenüber dem TX-Status priorisiert ist. Zuerst wird der Status des gerade empfangenen Frames in den entsprechenden Buffer-Deskriptor eingetragen (`1`), dann wird in den TX-Zustand gewechselt, wo das gleiche mit dem Status des gerade gesendeten Frames geschieht (`2`). Im WB-Zustand wird dann der Status des nächsten RX-Buffer-Deskriptors geschrieben (`3`) und gleich darauf im RX-Zustand gelesen (`4`). Der nächste WB-Zugriff schreibt dann die Adresse des RX-Buffer-Deskriptors (`6`). Mit dem Lesen dieser Adresse im RX-Zustand endet die Erzeugung des RX-Buffer-Deskriptors (`7`) und `RxEn_needed` wird auf `0` gesetzt. Die Lese-Zugriffe im TX-Zustand (`5` und `8`) beziehen sich auf den Status des TX-Bufferdeskriptors, jedoch wird erst, nachdem er im WB-Zustand geschrieben wurde (`9`), ein gültiger Status gelesen (`10`). Die nächsten beiden Zugriffe schreiben die Adresse des Tx-Buffer-Deskriptors (`11`) und lesen sie anschließend (`12`). Damit kann `TxEn_needed` wieder auf `0` gesetzt werden und der gesamte Vorgang ist abgeschlossen.

Die Erzeugung und das Auslesen beider Buffer-Deskriptoren dauert insgesamt 0,32 μs (25

[1] **ram o**utput **e**nabled
[2] **ram w**rite **e**nabled

Takte bei 78 MHz) und – was für den Vergleich mit dem IFG (0,96 μs bei 100 Mbit/s) wichtiger ist – die eines RX-Buffer-Deskriptoren nur 0,19 μs (15 Takte bei 78 MHz). Hinzu kommen noch bis zu 0,05 μs (4 Takte bei 78 MHz), da die Signale `RxEn_needed` und `TxEn_needed` nur zu bestimmten Takten gesetzt werden. Außerdem entsteht durch die Synchronisation des Frame-Endes mit der WB_CLK eine weitere Verzögerung. Insgesamt ist die Zeitspanne kleiner als der IFG, d.h. es ist bei 10- und 100-Mbit/s-Ethernet problemlos möglich, Pakete mit minimalem Abstand zu empfangen bzw. zu senden. Der Ethernet-MAC bereitet sich ausreichend schnell auf weitere Frames vor.

Bei Gigabit-Ethernet hingegen ist das Erzeugen der Buffer-Deskriptoren wegen der geänderten Taktverhältnisse nicht mehr schnell genug, weshalb zusätzlich noch die FSM umgestellt wurde: Der IFG beträgt dort 96 ns (12 Takte bei 125 MHz) im Vergleich zur Dauer der Generierung eines RX-Buffer-Deskriptoren von 152 ns (19 Takte Bei 125 MHz). Die Veränderung der FSM wird in Abschnitt 7.8 genau erläutert.

Zusätzlich zur automatischen Generierung der Buffer-Deskriptoren wurde der Speicher für die Buffer-Deskriptoren durch ein mit dem CORE Generator erzeugtes Modul ersetzt. Das war notwendig, weil der ursprüngliche Speicher Elemente eines anderen Typs von FPGA verwendet hat, was zu einer ineffektiven Speicherausnutzung geführt hat.

7.5 Anfügen des Ethernet-Headers beim Senden (eth_registers und eth_txethmac)

Ein Ethernet-Frame besteht aus verschiedenen Feldern (vgl. Abschnitt 2.3.3). Um größere zusammenhängende Datenmengen aus dem Speicher verschicken zu können, sollte der Ethernet-MAC diese selbstständig auf mehrere Frames aufteilen. Neben den Daten müssen dazu noch die Felder des Headers, also Ziel-, Startadresse und Typ, sowie die CRC-32-Prüfsumme ausgefüllt werden. Bisher musste man, um korrekte Frames zu versenden, die entsprechenden Header-Daten zusammen mit den Daten abspeichern und so die Daten schon beim Schreiben in den Speicher auf verschiedene Frames aufteilen. Da der Ethernet-Core die CRC-32-Prüfsumme anhängt (vgl. Abschnitt 6.2.7), wurde er dahingehend verändert, dass auch die Felder des Frame-Headers von ihm geschrieben werden. Zwei neue Core-Register in `eth_registers` waren notwendig, um neben der Core-eigenen MAC-Adresse auch die MAC-Adresse des Ziels und den Typ des Frames zu speichern (MAC_DEST0 und MAC_DEST1). Diese Informationen werden über die beiden Signalleitungen `r_MacDest` (aus MAC_DEST1[15:0] und MAC_DEST0[31:0]) und `r_EthType` (aus MAC_DEST1[31:16]) in das `eth_txethmac`-Modul geleitet. Die Absender-Adresse kann direkt aus den Registern MAC_ADDR0 und MAC_ADDR1 übernommen werden. Um den Header den Daten voranzustellen, wird in `eth_txethmac` ein neu hinzugefügtes Schieberegister, über das die Daten gelenkt werden, durch das Signal `TxStartFrm` parallel mit den entsprechenden Werten des Headers initialisiert. Sobald normalerweise die Daten gesendet würden, verschiebt das aktive Signal `shift_en` die Daten des Schieberegisters alle zwei Takte, so dass am seriellen Ausgang zuerst die Header-Werte anliegen und gesendet werden. Das Ende des Frames, signalisiert durch `TxEndFrm`, wird durch ein weiteres Schieberegister verzögert, so dass das Frame-Ende mit dem Abschluss der Daten übereinstimmt. Damit der Schiebevorgang genügend lange andauert, wird ein weiteres

Signal, `not_finished`, benötigt. Es signalisiert, wann das letzte Daten-Byte das Schieberegister verlässt, weshalb es im Modul `eth_transmitcontrol` auch das vorzeitige Absenden von Pausen-Frames verhindert.

7.6 Entfernen des Ethernet-Headers beim Empfangen (eth_rxethmac)

Wenn Frames empfangen werden, verzögert ein Schieberegister in `eth_rxethmac` das `RxStartFrm`-Signal um 28 bzw. bei Gigabit-Ethernet um 14 Takte. Dadurch bleibt der Header auf dem weiteren Datenpfad, speziell bei der Speicherung in das RX-Fifo im `wishbone`-Modul, unberücksichtigt. Die Adressen-Überprüfung findet unabhängig davon in dem Modul `eth_rxaddrcheck` statt. Die CRC-32-Prüfsumme wird später im `n_ethernet_ram_interface` entfernt (vgl. Abschnitt 7.3), weshalb im Modul `eth_wishbone` die Länge des empfangenen Frames, die im Buffer-Deskriptor abgespeichert wird, um die Header- und FCS-Länge reduziert werden muss (vgl. Abschnitt 6.2.8).

7.7 Umstellung auf 8 Bit-Datenbreite zum Phy (eth_maccontrol, eth_txethmac und eth_rxethmac)

Nach den allgemeinen Anpassungen des Ethernet-MACs, werden in den kommenden Abschnitten die Änderungen dargestellt, die speziell für den Gigabit-Ethernet-Betrieb nötig waren. Die Veränderungen sind notwendig wegen der Unterschiede zwischen MII und GMII, d.h. der Verbindung zwischen Phy und MAC. Das sind die verdoppelte Busbreite sowie die schnellere Taktung mit 125 MHz statt 25 MHz.

Das GMII benötigt beim Senden von Daten 8 Bit pro Takt bzw. liefert diese beim Empfangen. Aus diesem Grund ist ein erster Schritt beim Umstieg auf Gigabit-Ethernet die Verbreiterung der Datenleitungen gewesen. Diese betreffen im wesentlichen den Datenpfad von den MII-Datenports zu den Fifos im `eth_wishbone`-Modul. Im Folgenden werden die zusätzlichen Änderungen am Sende- und Empfangsteil des Ethernet-MACs beschrieben.

Die zu sendenden Daten werden byteweise aus dem TX-Fifo ausgelesen und an das Modul `eth_maccontrol` übergeben. An der Datenbreite musste deshalb dort nichts geändert werden, aber der Zähler `ByteCnt` im Untermodul `eth_transmitcontrol`, der die Daten des Signals `MuxedCtrlData` steuert, hat dies im 10- und 100-Mbit/s-Modus nur jeden zweiten Takt getan. Das wurde durch Halbierung der Zählerwerte korrigiert.

Das Schieberegister, in `eth_txethmac` für die Erstellung des Frame-Headers zuständig, ist durch `shift_en` nur jeden zweiten Takt aktiv (vgl. Abschnitt 7.5); durch Entfernen des Signals aus den Abfragen erhält man die erforderlichen 8 Bit pro Takt. Die Berechnung der CRC-32-Prüfsumme für 8 Bit breite Daten wurde dadurch erreicht, dass das zuständige Untermodul `eth_crc` ein neues Modul einbindet (`crc32_d`), in dem durch eine Konstante die Datenbreite auf 4 oder 8 Bit eingestellt werden kann. Dieses Modul wiederum verwendet zur Berechnung der Prüfsumme Funktionen, die mit Hilfe des *CRC Tools* von Easics erstellt wurden (`CRC32_D4` und `CRC32_D8`) [8]. Damit lassen sich CRC-Funktionen für unterschiedliche

Polynome und Datenbreiten in VHDL oder Verilog generieren und in entsprechende Module einbinden. Um das korrekte Senden der Daten bei Gigabit-Ethernet zu erreichen, mussten zudem die beiden Untermodule `eth_txcounters` und `eth_txstatem` angepasst werden. In dem Erstgenannten wurden die Signale für das Erreichen der Nibble 7 und 15 ersetzt durch die Signale für Nibble 6 (=Byte 3) und Nibble 14 (=Byte 7), da die ungeraden Nibble-Werte wegen der 8-Bit-Busbreite nicht mehr vorkommen können. Außerdem wurde die Abfrage der minimalen Framelänge `NibbleMinFl` auf gerade Werte korrigiert. Analog zu `eth_transmitcontrol` wurde auch hier der Byte-Zähler `ByteCnt` angepasst. In dem zweiten Modul mussten die Bedingungen für die Zustandswechsel der FSM auf die veränderten Zählersignale abgestimmt werden und die Unterscheidung zwischen `StateData[1]` und `StateData[0]` aufgehoben werden. Das gleiche gilt für die FSM in `eth_txethmac`, von der die Werte für die einzelnen Felder auf den Ausgangsport `MTxD` zum Phy geschrieben werden. Diese Werte mussten ebenfalls auf 8 Bit erweitert werden. Neben `TxData` sind auch die Werte der Zustände `Jam` und `Preamble` angepasst worden. Bei letzterem ist wichtig, dass der SFD wegen der umgekehrten Bit-Reihenfolge `d5` lautet (vgl. Abschnitt 2.3.3), sonst wird der Frame-Anfang nicht richtig erkannt.

Auf der Empfangsseite sind analoge Änderungen vorgenommen worden. Neben der Verbreiterung der Ports von `MRxD`, `RxData` und `DataCrc` in `eth_rxethmac` ist die Erkennung des SFD dem neuen Wert `d5` angepasst worden. In `eth_rxstatem` und `eth_macstatus` wird außerdem auch nicht mehr zwischen den Zuständen `StateData[0]` und `StateData[1]` unterschieden. Zusätzlich wurde `SlotTimer` in `eth_receivecontrol` so verändert, dass der Zähler jeden Takt zählt und damit wieder die richtige Pausenlänge liefert.

7.8 Umstellung der FSM zur automatischen Generierung der Buffer-Deskriptoren (eth_wishbone)

Neben den zuvor genannten Änderungen war es auch notwendig, Teile des Ethernet-MACs zu verändern, da die für Gigabit-Ethernet erforderlichen Geschwindigkeiten und Datenraten sonst nicht erreicht werden können. Dies betrifft den Zugriff auf das DDR-SDRAM (vgl. Abschnitt 7.3) und die Erzeugung der Buffer-Deskriptoren (vgl. Abschnitt 7.4). Der geänderte Zugriff auf das DDR-SDRAM wird allerdings erst in Abschnitt 7.9 erläutert.

Die Generierung und das Auslesen der Buffer-Deskriptoren sollte höchstens der Dauer des IFGs entsprechen, der bei Gigabit-Ethernet 12 Takte beträgt. Dauert es länger, besteht die Möglichkeit, dass Frames verworfen werden, weil der Ethernet-MAC sich nicht schnell genug auf den Empfang vorbereiten kann. Wie in Abschnitt 7.4 gezeigt wurde, dauert die Erstellung eines Buffer-Deskriptors im ungünstigsten Fall mindestens 15 Takte, wobei noch mehrere Takte zwischen Frame-Ende und dem Signal `...En_needed` sowie für die Synchronisation hinzukommen. Da sowohl RX_CLK als auch WB_CLK bei Gigabit-Ethernet mit 125 MHz getaktet sind, bedeutet das, dass der IFG nicht mehr ausreicht, um den Ethernet-MAC auf das nächste ankommende Frame vorzubereiten.

Aufbau der veränderten FSM
Die zur Generierung und zum Auslesen erforderliche Zeit wird durch die feste Abfolge der Zustandswechsel und die Priorisierung des WB-Zustandes bestimmt. Die FSM, die den Zugriff

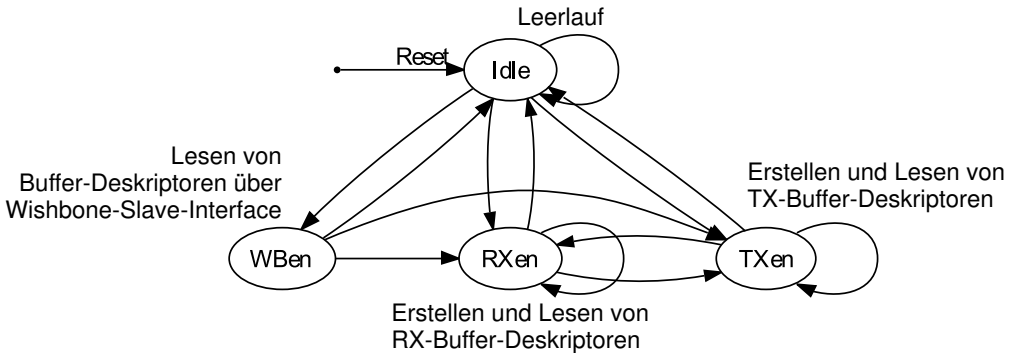


Abbildung 7.3: veränderte FSM für den Zugriff auf den Buffer-Deskriptor-Speicher (vgl. Abbildung 6.3 auf Seite 35)

auf den Speicher der Buffer-Deskriptoren regelt, wurde deshalb gemäß Abbildung 7.3 verändert. Neben der Umbenennung der Zustände wurden die Aufgaben der einzelnen Zustände und deren Abfolge neu verteilt. Ein wesentlicher Unterschied zur ursprünglichen FSM des Ethernet-Cores besteht darin, dass die Zustände nach einem Zugriff nicht zwangsläufig wechseln. Ohne eine Zugriffsanfrage bleibt die FSM im `Idle`-Zustand. Der `WBen`-Zustand für den Zugriff über das WISHBONE-Slave-Interface besitzt keine Sonderrolle mehr, er ist sogar den beiden anderen Zugriffen untergeordnet. Das liegt daran, dass er nur noch für Lese-Zugriffe über das WISHBONE-Slave-Interface auf die Buffer-Deskriptoren genutzt wird, beschreiben lassen sie sich nicht mehr über die Schnittstelle. Die Schreibzugriffe werden je nach Typ des Buffer-Deskriptors in dem entsprechenden Zustand durchgeführt. Außerdem dauert ein `WBen`-Zugriff nur noch einen Takt, danach wechselt er je nach Anfrage in den `Idle`-, den `RXen`- oder den `TXen`-Zustand. Bei den letzten beiden ist neu, dass die FSM solange in dem jeweiligen Zustand verbleibt, bis alle für einen Buffer-Deskriptor erforderlichen Zugriffe durchgeführt wurden. Welcher Zugriff innnerhalb des Zustandes erfolgt, wird durch zwei Zähler bestimmt (`strx_cnt` und `sttx_cnt`). Da der `RXen`-Zustand Priorität vor dem `TXen`-Zustand besitzt, wird bei gleichzeitigem Bedarf an Buffer-Deskriptoren zuerst der RX-Buffer-Deskriptor fertiggestellt und dann der TX-Buffer-Deskriptor abgearbeitet. Sollte die FSM sich gerade im `TXen`-Zustand befinden, wenn ein RX-Buffer-Deskriptor benötigt wird, so besteht die Möglichkeit, in den `RXen`-Zustand zu wechseln. Der Zustandswechsel findet allerdings nur zwischen dem Schreiben und dem Lesen des TX-Buffer-Deskriptors statt, erfolgt die Anfrage später, wird der TX-Buffer-Deskriptor erst fertiggestellt und anschließend in den `RXen`-Zustand gewechselt. Diese Abfolge stellt somit den ungünstigsten Fall für den Vergleich zwischen IFG und der Dauer zur Erzeugung eines RX-Buffer-Deskriptors dar.

Schreiben und Auslesen der Buffer-Deskriptoren

Abbildung 7.4 zeigt dieses Szenario. Der obere Teil zeigt die FSM und der untere die Art des Zugriffs. Die Lese- und Schreibzugriffe auf den Speicher sind dabei hellgrau unterlegt. Die Anfrage eines Buffer-Deskriptors wird durch `tx_` und `rx_access` signalisiert. Das sind die Verknüpfungen zwischen dem schon vorhandenen `TxEn_` bzw. `RxEn_needed` und `r_BDInt`. Letzteres gibt an, ob der Speicherbereich aufgebraucht ist und deshalb kein Buffer-Deskriptor erzeugt werden kann. Die FSM wechselt in den Zustand `TXen` und startet den Zähler `sttx_cnt`. Zuerst erfolgen nacheinander die Schreib-Zugriffe (1-3): das Schreiben des Status des vorangegangenen Frames (`sttx_cnt = 1`) sowie des Status (`sttx_cnt = 2`) und des Zeigers

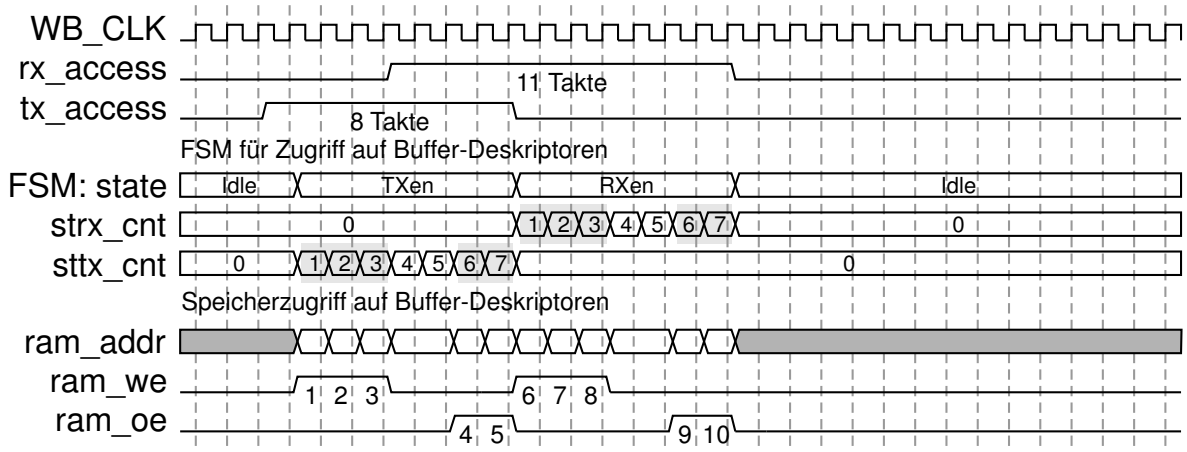


Abbildung 7.4: Zugriff auf die Buffer-Deskriptoren mit der veränderten FSM

des neuen Buffer-Deskriptors (`sttx_cnt = 3`). Ist zu diesem Zeitpunkt `rx_access = 1`, so wird in den `RXen`-Zustand gewechselt. Ansonsten findet noch einmal eine Überprüfung des Wertes von `r_BDInt[1]` statt (`sttx_cnt = 4`) und das Lesen des Buffer-Deskriptors wird gestartet (`sttx_cnt = 5`). Anschließend wird der Status (`sttx_cnt = 6`) und der Zeiger (`sttx_cnt = 7`) gelesen (4-5). Die FSM geht direkt in den `RXen`-Zustand. Der Ablauf ist genau der gleiche mit den entsprechenden `strx_cnt`-Werten. Zwischen den Schreib- (6-8) und Lese-Zugriffen (9-10) kann allerdings kein Zustandswechsel stattfinden.

Die Dauer für die Erstellung eines RX-Buffer-Deskriptors ist im hier aufgeführten ungünstigsten Fall 88 ns (11 Takte bei 125 MHz) und liegt damit noch unter dem Wert des IFGs von 96 ns (12 Takte bei 125 MHz). Selbst bei minimalem Frame-Abstand wird kein Frame verworfen, da sich der Ethernet-MAC ausreichend schnell darauf vorbereiten kann. Wichtig ist außerdem, dass das Auslösen der `...En_needed`-Signale jederzeit und nicht nur zu bestimmten Zeitpunkten erfolgen kann, wodurch eine weitere Verzögerung vermieden wird. Die Unsicherheit wegen der Synchronisation des Frame-Endes bleibt allerdings, sie beträgt im schlechtesten Fall 2 Takte. Sollte es deswegen zu Übertragungsproblemen kommen, lässt die FSM noch ein wenig Spielraum zur Optimierung. Für die Simulation hat allerdings gezeigt, dass bei minimalem IFG keine Frame verworfen werden (vgl. Abschnitt 10.3).

7.9 Beschleunigter Zugriff auf das DDR-SDRAM (n_ethernet_ram_interface und eth_wishbone)

Auch die Schnittstelle zum DDR-SDRAM erfüllt die Geschwindigkeitsansprüche von Gigabit Ethernet nicht. Durch den Handshake, der in dem Modul `n_ethernet_ram_interface` verwendet wird, um aus dem DDR-SDRAM zu lesen, können die Daten bei Gigabit-Ethernet nicht schnell genug von dort in den Sende-Fifo transportiert werden (vgl. Abschnitt 7.3). Schreibzugriffe sind nicht betroffen, da diese schon beim Absetzen des Schreibbefehls quittiert werden. Es spielt dann keine Rolle, ob eine Verzögerung auftritt. Ein einzelner Lesezugriff auf das DDR-SDRAM hingegen dauert im Durchschnitt 22 Takte und liefert dann 128 Bits an

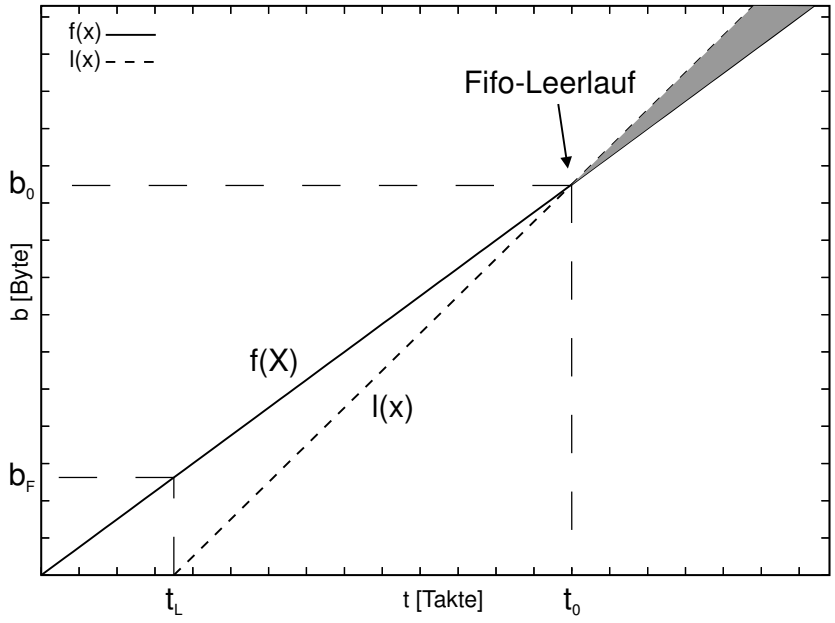


Abbildung 7.5: Füllstand des TX-Fifos

Daten. Dies entspricht weniger als 6 Bits pro Takt, Gigabit-Ethernet hingegen benötigt aber jeden Takt 8 Bit, weshalb der in `eth_wishbone` bisher verwendete TX-Fifo leerlaufen würde.

Eine Möglichkeit wäre, den TX-Fifo zu vergrößern und eine dadurch bedingte Latenz in Kauf zu nehmen. Das ist jedoch nicht praktikabel, wie folgende Rechnung zeigt: Sowohl das Befüllen des Fifos von Seiten des DDR-SDRAMs als auch dessen Leeren durch das Senden der Daten können je durch eine Gerade beschrieben werden. Für das Befüllen gilt:

$$f(t) = \frac{3\,\text{Byte}}{4\,\text{Takte}}\,t\,, \tag{7.1}$$

für das Leeren:

$$l(t) = 1\,\frac{\text{Byte}}{\text{Takt}}\,t + C(t_L)\,.$$

Dabei geben $f(t)$ und $l(t)$ die Datenmenge in Byte zu einem bestimmten Takt t an. Die Konstante C hängt dabei von der Latenz t_L zwischen dem Beginn des Schreibens von Daten in den TX-Fifo und dem Beginn des Lesens der Daten aus dem Fifo ab. Beide Geraden sind in autoreffig:geraden dargestellt.

Der Schnittpunkt der beiden gibt den Zeitpunkt t_0 an, wenn der TX-Fifo leer ist und keine weiteren Daten mehr daraus gelesen werden können. Dadurch ist die maximale Datenmenge b_0 festgelegt, die ein Frame ohne Leerlauf des Fifos enthalten darf:

$$b_0 = f(t_0) = \frac{3\,\text{Byte}}{4\,\text{Takte}}\,t_0\,. \tag{7.2}$$

Mit Hilfe der Punktsteigungsformel und dem Schnittpunkt $(t_0\,, b_0)$ ergibt sich für $l(t)$:

$$l(t) = 1\,\frac{\text{Byte}}{\text{Takt}}\,(t - t_0) + b_0 = 1\,\frac{\text{Byte}}{\text{Takt}}\,t - \frac{1}{3}\,b_0\,. \tag{7.3}$$

Dabei wurde t_0 mit Hilfe von Gleichung 7.2 durch b_0 ausgedrückt. Die Nullstelle von Gleichung 7.3 gibt die minimale Latenzzeit t_L je nach Datenmenge des Frames an:

$$t_L = \frac{1\,\text{Takt}}{3\,\text{Byte}}\, b_0\,. \tag{7.4}$$

Aus Gleichung 7.1 und 7.4 erhält man die mindestens benötigte Fifo-Größe:

$$b_F = \frac{1}{4}\, b_0\,. \tag{7.5}$$

Mit den Gleichungen 7.4 und 7.5 lassen sich die Anforderungen an den TX-Fifo überprüfen. Mit dem vorhandenen Fifo von 32 Bytes können nur Frames bis 128 Byte versendet werden bzw. für die Maxmimallänge von 1500 Bytes müsste der Fifo mindestens 375 Bytes groß sein. Gleichzeitig würde noch eine Latenz von 500 Takten auftreten.

Eine andere Möglichkeit das Problem des TX-Fifo-Leerlaufs zu beheben, besteht darin, die Schnittstelle zum DDR-SDRAM zu beschleunigen. Das war durch mehrere Änderungen am Modul `n_ethernet_ram_interface` möglich. Eine Verbreiterung der Verbindung zwischen diesem Modul und dem `eth_wishbone`-Modul von 32 auf 64 Bits reduziert die Zeit, die es dauert, die Daten von der DDR-SDRAM-Schnittstelle in den TX-Fifo im `eth_wishbone`-Modul zu schreiben. Damit verbunden war eine Änderung der Fifos in `eth_wishbone`, wie sie in Abschnitt 7.10 beschrieben wird. Um die Daten mit einer höheren Rate aus dem DDR-SDRAM auszulesen, bietet das Steuermodul für das DDR-SDRAM (`manager_nat_sdram`) die Möglichkeit, mehrere Daten-Anfragen zu senden, bevor von der ersten eine Rückmeldung kommt. Dazu speichert das `manager_nat_sdram`-Modul die Anfragen in einem Fifo zwischen und verarbeitet sie dann mittels Pipelining. Das bewirkt, dass die Wartezeit auf die ersten Daten zwar dieselbe Zeit beträgt, aber die darauffolgenden Anfragen unmittelbar abgearbeitet werden. Bei Schreib-Zugriffen geschieht das gleiche, dies ist aber für den Benutzer nicht erkennbar. Deshalb musste auch nur der Lese-Zugriff modifiziert werden.

Das geänderte `n_ethernet_ram_interface`-Modul nutzt dies statt des Handshakes aus (vgl. Abschnitt 3.1), indem es den von dem Fifo übermittelten Status abfragt und solange Daten-Anfragen an das DDR-SDRAM sendet, wie dies erlaubt ist. Da das Signal `m_wb_ack_i` jetzt nur noch die Anfrage bestätigt, nach der dann gleich eine weitere folgen kann, war ein neues Signal (`m_dv_i`) nötig, um die vom DDR-SDRAM gelesenen Daten anzuzeigen und in den TX-Fifo zu schreiben. In `eth_wishbone` ist außerdem zusätzlich ein weiterer Zähler (`TxLengthFifo`) eingeführt worden, der nicht wie der bisherige die Zahl der Anfragen, sondern die der empfangenen Daten zählt. Beide werden mit der Anzahl der für ein Frame notwendigen Speicherzugriffe gestartet. Die Datenübertragung an den Phy beginnt dann, wenn keine Daten vom DDR-SDRAM mehr erforderlich sind (`TxLengthFifo` = 0) oder der TX-Fifo voll ist. Durch die beiden Änderungen wird die frühere FSM in `n_ethernet_ram_interface` überflüssig und das Modul vereinfacht sich stark.

7.10 Asynchrone Fifos (eth_wishbone)

Die beiden identischen Fifos in `eth_wishbone` wurden durch zwei unterschiedliche asynchrone Fifos ersetzt, die mit dem CORE Generator-Programm erstellt wurden. Sie besitzen unterschiedliche Daten-Breiten an den Ein- und Ausgängen. Auf der Sendeseite ist dies

`asyn_fifo_64_8` und auf der Empfangsseite `asyn_fifo_32_64`. Beide Fifos besitzen jetzt einen 64 Bit breiten Port zum WISHBONE-Master-Interface. Dadurch wird die zusätzliche Logik zum Aufteilen und Zusammenfügen der unterschiedlichen Port-Breiten zwischen DDR-SDRAM und dem WISHBONE-Master-Interface überflüssig. Außerdem erspart es einen Teil der Synchronisationslogik zwischen Fifos und Phy, da direkt mit den entsprechenden Takten TX_CLK und RX_CLK auf diese Seite der Fifos zugegriffen wird. Ein letzter Grund ist die durch das Ersetzen erlangte Flexibilität, da sich die Fifos jetzt ohne Probleme in ihrer Größe an den Bedarf anpassen lassen.

7.11 Konzept zur Fehlerbehandlung und Steuerung des Systems über Ethernet

Die Fehlererkennung bei empfangenen Frames erfolgt durch den Ethernet-MAC. Die Daten werden jedoch im Fall eines erkannten Fehlers trotzdem in den DAten-Speicher geschrieben. Dabei wird an anderer Stelle im MAC gespeichert, dass ein Fehler vorliegt. Die noch nicht implementierte Logik zur Steuerung des Systems über Gigabit-Ethernet muss diesen Fehlerspeicher auswerten.

Steuerung des Systems über Ethernet
Um das System über Ethernet steuern zu können, muss es die im empfangenen Ethernet-Frame enthaltenen Befehle bearbeiten können. Bislang werden die Daten der empfangenen Frames nur in den Speicher geschrieben. Das Konzept, das in Zusammenarbeit mit Sebastian Millner entwickelt wurde, sieht vor, diesen Speicher als Schnittstelle von Ethernet-MAC zur Auswertungslogik zu nutzen. Die Logik stellt nach dem OSI-Referenzmodell für den MAC ein höheres Protokoll dar. Sie untersucht die im Daten-Speicher enthaltenen Werte auf Befehle und Daten und verarbeitet diese entsprechend.

Für den Ethernet-MAC besteht hier die Möglichkeit, Frames mit falscher CRC-32-Prüfsumme zu verwerfen und so nur gesicherte Daten weiterzugeben. Dies gelingt wie folgt: Neben dem Daten-Speicher existiert ein weiterer Speicher, der die Information enthält, auf welche Speicherbereiche die Auswertungslogik zugreifen darf. Die Freigabe eines Bereiches erfolgt durch den Ethernet-MAC nachdem die CRC-32-Prüfsumme eines Frames bestätigt wurde. Auf diese Weise wird die Auswertung falscher Daten verhindert. Diese Methode stellt einen elegante Art dar, eine zusätzliche Latenz beim Schreiben der Daten in den Speicher zu vermeiden. Ohne sie müsste der Ethernet-MAC die Daten eines Frames solange zurückhalten, bis deren Richtigkeit überprüft werden konnte. Erst dann würden die Daten gespeichert werden. Das würde zu einer Latenz von der Länge des empfangenen Frames führen, schlimmstenfalls der maximalen Läange eines Frames, also 1518 Takte. Die Daten eines Ethernet-Frames stellen für die Auswertungslogik ein durch das Frame gekapseltes Paket dar. Fehlerhafte Frame-Daten bedeuten somit gleichzeitig einen Paket-Verlust für die höhere Schicht. Paket-Verluste werden im vorliegenden Konzept ähnlich dem Sliding-Window-Verfahren bei TCP detektiert und behandelt [26].

8 Integration des Ethernet-MACs ins System

Nach den Erweiterungen, die aus dem Ethernet-Core den Ethernet-MAC für 1-Gbit/s-Ethernet machen, wurde dieser in die Logik der FPGAs auf den Zielplattformen eingefügt. Dies geschah durch Einbindung des Ethernet-MACs in das jeweils oberste Modul des FPGAs, das alle Module beinhaltet und speziell für die verwendete Plattform vorhanden ist. Zum einen ist das `nathan_top` auf den Nathan-Modulen (Abschnitt 8.1), zum anderen das Modul `balu_top` auf Balu, der zweiten Version der Backplane (Abschnitt 8.2). Diese Module sorgen für die Verbindungen mit den Pins des FPGAs bzw. den anderen Modulen, indem die Ports des Ethernet-MACs über FPGA-interne Signale an diese angeschlossen werden. Wichtig dabei war auch die Erzeugung der richtigen Takte für die verschiedenen Module aus den vorhandenen Takten.

8.1 Einbindung in das Nathan-Modul (nathan_top und nathan_clk_reset)

Auf dem Nathan-Modul wird das Phy-Modul über den ANN-Sockel betrieben (vgl. Abschnitt 5.2). Dazu müssen im Nathan-Modul die Ports des GMII mit den zugehörigen FPGA-Anschlüssen, die über Pins mit dem ANN-Sockel verbunden sind, zusammengeschaltet werden. Weiter ist der Ethernet-MAC im FPGA mit den Modulen der Slow-Control (`sctrl_main`) und des DDR-SDRAMs (`manager_nat_sdram`) verbunden. Die nötigen Takt-Signale stammen aus dem von Stefan Philipp erstellten Modul `nathan_clk_reset`, in dem zentral alle Takt- und Reset-Signale kontrolliert werden. Für 10- und 100-Mbit/s-Ethernet konnte die bestehende Verschaltung in `nathan_top` übernommen werden, so wie sie von Sebastian Millner erstellt wurde. Bei Gigabit-Ethernet musste schon aufgrund der veränderten Taktung (vgl. Abschnitt 2.3.2) das Modul `nathan_clk_reset` umgeschrieben werden. Außerdem verwendet der Ethernet-MAC bei dieser Übertragungsrate andere Pins des ANN-Sockels, da die bei den geringeren Übertragunsraten benutzten Pins durch nebeneinander verlaufende Leitungen zu Übersprechen zwischen den betroffenen Signalen geführt hat. Das hatte zur Folge, dass auch `nathan_top` für diese Arbeit geändert wurde.

Das Resultat der Änderungen an `nathan_clk_reset` ist in Abbildung 8.1 dargestellt. Es zeigt wie die verschiedenen Takte für das Nathan-Modul erzeugt werden. Das FPGA-Design auf dem Nathan-Modul verfügt insgesamt über vier mögliche Takteingänge mit unterschiedlichen Frequenzen. Die EXT_CLK1 ist der Takt, der auf der Darkwing-Platine erzeugt und über das SCSI-Kabel auf die Backplane übertragen wird, ihr Takt beträgt maximal 100 MHz. Die EXT_CLK2 verwendet den Takt des lokalen Oszillators auf der Backplane und besitzt eine Frequenz von 156,25 MHz. Die beiden letzten Takte sind ANNA_CLK1 und ANNA_CLK2.

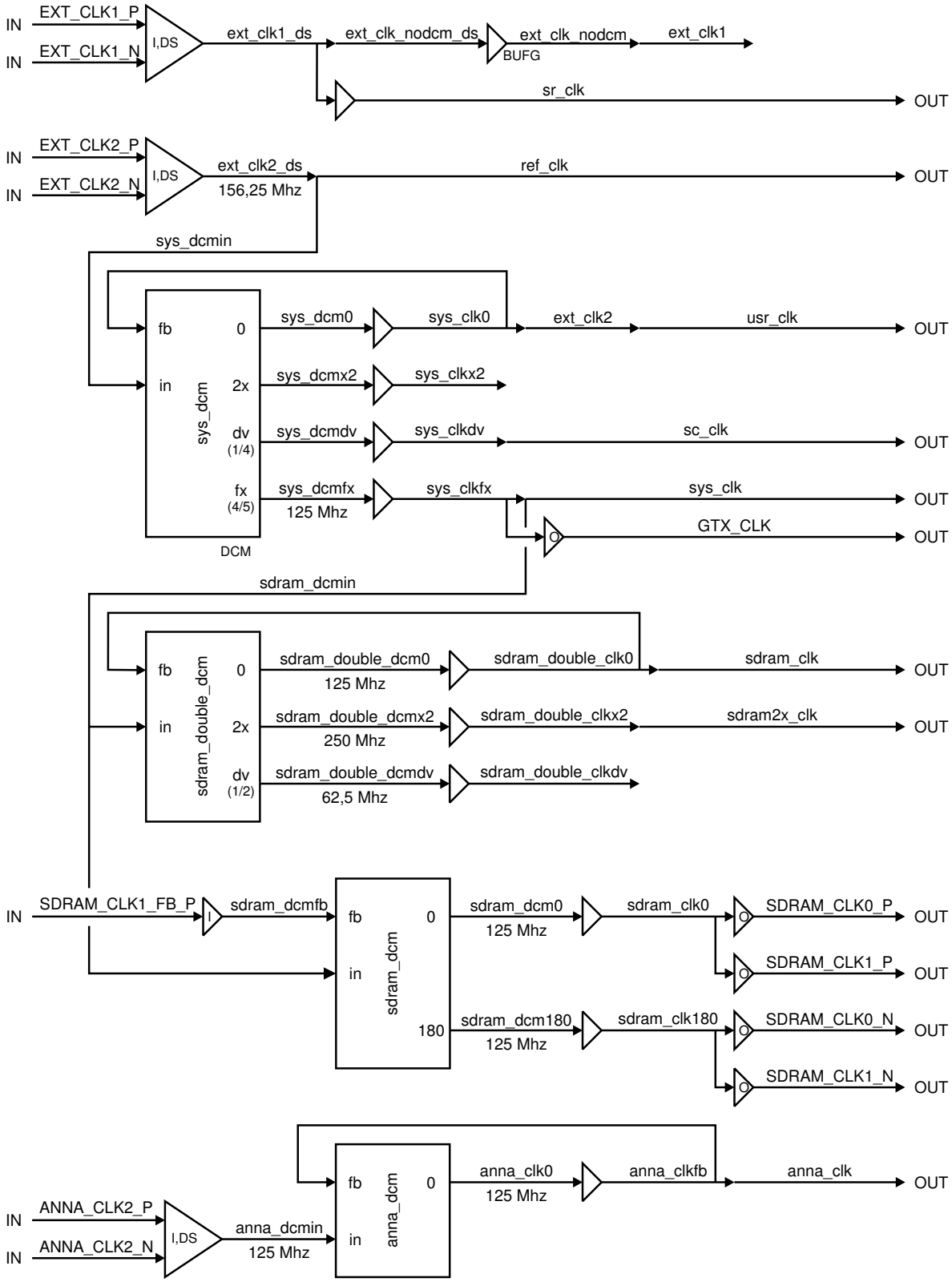


Abbildung 8.1: Übersicht über die Erzeugung der Takte auf dem Nathan-Modul

Dabei handelt es sich um zwei bidirektionale Verbindungen zum ANN-Sockel, d.h. der FPGA kann entweder einen Takt dorthin übertragen oder auf demselben Weg den externen Takt von dort empfangen.

Bei 10- und 100-Mbit/s-Ethernet wurde mit einem DCM die Frequenz der EXT_CLK2 für WB_CLK halbiert (≈ 78 MHz) und die beiden Takte TX_CLK und RX_CLK stammten vom Phy (25 MHz) (vgl. Abschnitt 3.4). Für Gigabit-Ethernet sind die Takte des Ethernet-MACs komplett verändert. Sowohl GTX_CLK als auch RX_CLK betragen jetzt 125 Mhz, wobei letzterer durch den Phy rekonstruiert und zum FPGA übertragen wird (vgl. Abschnitt 6.3). Für GTX_CLK benötigt der Phy den Takt vom FPGA. Derselbe Takt wird auch für WB_CLK verwendet, um den übertragungsunabhängigen Teil des Ethernet-MACs zu beschleunigen.

Der Takt der EXT_CLK2 wird auf der Backplane und dem Nathan-Modul differentiell übertragen. Ein DCM (`sys_dcm`) reduziert diese EXT_CLK2 dann auf 4/5 ihrer Frequenz (125 MHz). Über einen BUFG, einen globalen Eingangspuffer, gelangt diese Frequenz auf eine globale Taktleitung, von wo aus sie als `sys_clk` zur Taktung des Ethernet-MACs verwendet wird. Von der globalen Leitung aus wird der Takt gleichzeitig über einen Ausgangspuffer (OBUF) auf einen bestimmten Pin des FPGAs gegeben, der mit der GTX_CLK zum Phy auf dem Nathan-Modul verbunden ist.

Die gleiche Frequenz wird auch zur Taktung des RAM-Moduls im Design (`manager_nat_sdram`) und zum Betrieb des DDR-SDRAMs gebraucht. Diese und die für `manager_nat_sdram` gleichzeitig benötigte doppelte Frequenz werden mit dem `sdram_double_dcm` aus der `sys_clk` erzeugt und über je einen BUFG auf den globalen Takt-Leitungen `sdram_clk` und `sdram2x_clk` bereitgestellt.

Ein weiterer DCM (`sdram_dcm`) generiert das Taktsignal für das DDR-SDRAM aus der `sys_clk`. Da die entsprechenden Taktleitungen auf dem Nathan-Modul differentiell sind, wird sowohl der ursprüngliche Eingangstakt des DCMs als auch der um 180° phasenverschobenene Takt erst von einem BUFG auf globale Takt-Leitungen gegeben (`sdram_clk0` und `sdram_clk180`), um dann über je einen OBUF als differentielles Signal zum DDR-SDRAM übertragen zu werden. Um die Last an den Takt-Leitungen zu reduzieren, wird eine Hälfte des DDR-SDRAMs über zusätzliche Leitungen mit demselben Takt versorgt. Eines dieser Takt-Signale wird auf den Feedback-Eingang des DCMs gegeben. Durch diese Rückkopplung werden Phasenverschiebungen der Takte zwischen DCM-Eingang und DDR-SDRAM-Eingang ausgeglichen.

Die rekonstruierte RX_CLK, die ebenfalls als differentielles Signal am FPGA ankommt, wird auf den `anna_dcm` geführt. Dieser wird dazu benutzt, den Takt in seiner Phase zu verschieben, um mögliche Unterschiede in der Länge der Signalleitungen von Daten und Takt aufzuheben. Die Verschiebung lässt sich während des Betriebs der Backplane mit Hilfe von Slow-Control-Befehlen einstellen. Der resultierende Takt steht dem Ethernet-MAC dann auf einer globalen Takt-Leitung unter der Bezeichnung `anna_clk` zur Verfügung.

In `nathan_top` wurden die bisher mit dem Ethernet-MAC verbunden Signale gegen die jetzt verwendeten ausgetauscht. Den aktuellen Stand gibt Anhang A wieder.

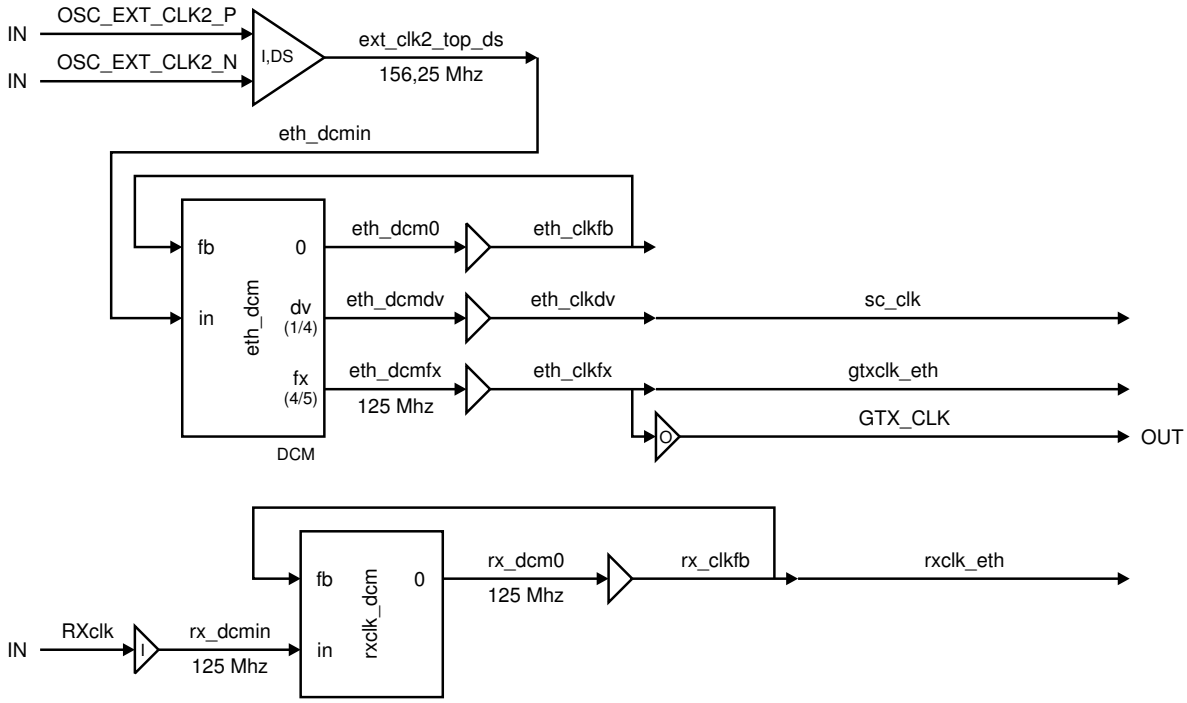


Abbildung 8.2: Takt-Erzeugung auf der Balu-Backplane

8.2 Einbindung in Balu-Backplane (balu_top)

Bei Verwendung des Phy-Moduls auf Balu stehen im Vergleich zum Nathan-Modul folgende Takte zur Verfügung: EXT_CLK1, EXT_CLK2 und RX_CLK. Letzterer stammt von dem speziell für das Phy-Modul vorgesehenen Steckplatz auf der Backplane. Für die Frequenzen der in `balu_top` und im Ethernet-MAC auf der Backplane vorhandenen Takte gilt das gleiche wie auf dem Nathan-Modul: Die EXT_CLK1 ($\leq$ 100 MHz) wird über das SCSI-Kabel empfangen, die EXT_CLK2 (156,25 Mhz) wird lokal auf der Backplane erzeugt und die anderen Takte (WB_CLK, GTX_CLK und RX_CLK) entsprechen ihren Signalen auf dem Nathan-Modul. Bisher wurde die neue Version der Backplane zunächst benutzt, um die Funktionalität der alten nachzubilden, damit die vorhandene Ansteuerungssoftware nicht verändert werden muss. Aus diesem Grund war auch keine gesonderte Kontrolle über die Takte erforderlich. Für die Verwendung des Ethernet-MACs auf der Backplane ist dies aber notwendig. Deshalb wurde die entsprechende Funktionalität in das Modul `balu_top` integriert. In Abbildung 8.2 ist dargestellt, wie dort die verschiedenen Takte erzeugt werden.

Im Gegensatz zu `nathan_clk_reset` werden nur zwei der DCMs benutzt, da auf der Backplane kein DDR-SDRAM zur Verfügung steht. Der `eth_dcm` entspricht `sys_dcm`: Aus dem differentiellen Takt der EXT_CLK2 (OSC_EXT_CLK2_P und OSC_EXT_CLK2_N) werden der Takt für den Ethernet-MAC (`gtxclk_eth`) und die GTX_CLK für den Phy erzeugt. Aus der RX_CLK vom Phy wird durch den `rxclk_dcm` der entsprechende, eventuell phasenverschobene Takt des Ethernet-MACs mit der Bezeichnung `rxclk_eth`.

9 PC-Software

Dieses Kapitel beschreibt die für die Ansteuerung des Systems und dessen Test erstellte Software. Die Backplane und damit auch der darauf befindliche FPGA ist über das SCSI-Kabel mit der Darkwing-Platine auf einem PC verbunden, der das gesamte System steuert und überwacht (vgl. Kapitel 3). Durch die Implementation der Gigabit-Ethernet-Schnittstelle steht dem System eine weitere Verbindungsmöglichkeit zur Verfügung. Das Phy-Modul dieser Schnittstelle ist mit der Gigabit-Ethernet-Netzwerkkarte eines weiteren PCs verbunden, mit dem Daten ausgestauscht werden sollen. Dabei kann es sich auch um denselben PC wie bei der SCSI-Anbindung handeln. Dieser Aufbau wird in Abbildung 9.1 gezeigt, wobei die beiden PCs zur Unterscheidung ihrer Funktionalität mit *Steuer-PC* und *Daten-PC* bezeichnet werden.

Um nun den Ethernet-MAC verwenden zu können, wurden mehrere Programme in der Programmiersprache C erstellt. Sie lassen sich je nach Einsatzgebiet unterteilen. Zu den Programmen gehört jeweils noch eine Bibliothek bestehend aus Header-Datei und einer Datei mit der Implementation der darin enthaltenen Funktionen. Diese enthalten die wichtigsten Werte und Funktionen, um die Bibliothek auch in anderen Programmen verwenden zu können. Im Einzelnen sind das auf dem Steuer-PC `eth_reg.c`, `eth_corelib.h` und `eth_corelib.c`, die über SCSI-Kabel auf den Ethernet-MAC und Phy zugreifen, sowie `eth_pc.c`, `eth_pclib.h` und `eth_pclib.c`, die es ermöglichen, am Daten-PC Frames zu senden und zu empfangen. Zum Testen der Implementation existiert zusätzlich noch das Programm `eth_test.c`, das ebenfalls die oben genannten Bibliotheken verwendet.

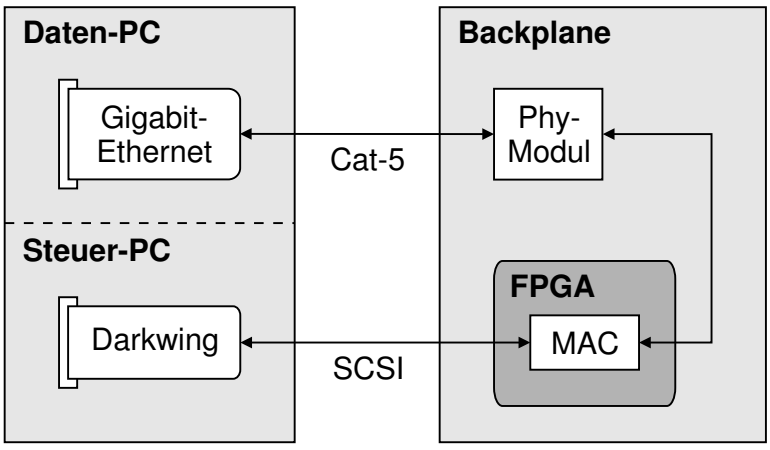


Abbildung 9.1: Systemaufbau mit Gigabit-Ethernet-Verbindung (bei den beiden PCs kann es sich, durch die gestrichelte Linie angedeutet, auch um denselben handeln)

9.1 Bibliothek für MAC und Phy (eth_corelib.h)

Die Bibliothek `eth_corelib.h` enthält die Definitionen der wichtigsten Werte und die Deklaration der Funktionen, die benötigt werden, um den Core bzw. den Phy anzusteuern. Darin enthalten sind zuerst einige allgemeine Werte wie die Geschwindigkeit der WB_CLK (CORE_CLK), die Anzahl der Core-Register (MAXREGCORE) und die Anzahl der Phy-Register (MAXREGPHY). Weiter sind die Bezeichnungen der Core-Register, wie sie in Anhang C aufgeführt sind, über eine `#define`-Anweisung mit ihrer Speicher-Adresse verknüpft. Auf dieselbe Art sind auch häufig auftretende Bit-Werte der einzelnen Core-Register mit ihrer entsprechenden Bedeutung hinterlegt, vor allem von den Buffer-Deskriptoren, dem MODER-Register und den Registern, die dem Zugriff auf den Phy dienen (MIIMODER, MIICOMMAND und MIISTATUS). Dies erlaubt eine einfache Abfrage bzw. ein einfaches Setzen dieser Werte mit Hilfe von Bit-Operatoren. Zusätzlich ist ein Feld REGC deklariert, in dem die Namen der Core-Register an der Position der zugehörigen Adresse eingetragen sind, um die Ausgabe der Namen statt der Adressen zu ermöglichen. Für den Teil, der den Phy betrifft, gilt das gleiche. Zum einen ist die Standardadresse des verwendeten Phy-Moduls abgespeichert (PHY), zum anderen existieren auch für die Phy-Register die Name-Adresse- und Adresse-Name-Zuweisungen (REGP) wie bei den Core-Registern. Für die Register BMCR[1] und LINK_AN[2] sind zudem noch einige Bit-Werte hier eingetragen.

Außer den Werten sind noch folgende Funktionen deklariert, die in `eth_corelib.c` definiert werden: `CORE_Read`, `CORE_Write`, `PHY_Read`, `PHY_Write` und `ETH_Init`.

9.2 Implementation der MAC- und Phy-Funktionen (eth_corelib.c)

In dieser C-Datei sind die in Abschnitt 9.1 genannten Felder REGC und REGP und die im Abschnitt zuvor genannten Funktionen implementiert. Die Funktionen dienen dem Beschreiben und dem Auslesen der Register von Ethernet-MAC und Phy.

ETH_Init
Mit `ETH_Init` wird der Ethernet-MAC initialisiert. Dazu überprüft die Funktion zuerst, ob der im Quellcode fest eingestellte Teiler den Takt MDC auf eine Frequenz $\leq$ 2,5 MHz einstellt. Ist das der Fall, wird dieser Wert mit `CORE_Write` in das Core-Register MIIMODER geschrieben, ansonsten wird eine Fehlermeldung ausgegeben. Danach ermittelt die Funktion die Adresse des Phy, indem sie mit einer `while`-Schleife die 32 möglichen Adressen durchläuft und ein für den Phy unveränderliches Register auf einen bestimmten Wert überprüft. Wird kein Phy gefunden, beträgt der Rückgabewert der Funktion `-1`, ansonsten die Adresse des Phys.

CORE_Write und **CORE_Read**
Diese beiden Funktionen geben die ihnen übergebenen Variablen direkt an die entsprechenden Slow-Control-Funktionen weiter (`SCTRL_Write` und `SCTRL_Read`). Die Rückgabewerte sind ebenfalls zu diesen Funktionen identisch.

[1] **B**asic **M**ode **C**ontrol **R**egister
[2] **LINK** and **A**uto-**N**egotiation status register

PHY_Write und **PHY_Read**
Die Zugriffe auf die Phy-Register erfolgen durch das Beschreiben mehrerer Core-Register. Diese werden durch die Funktionen `PHY_Write` und `PHY_Read` gekapselt. Die Syntax der beiden Funktionen entspricht dem von `CORE_Write` und `CORE_Read`. `PHY_Write` erzeugt aus einem Aufruf mehrere Slow-Control-Befehle, die in das Core-Register MIIADDRESS die Adresse von Phy und Phy-Register, in MIITX_DATA die zu übertragenden Daten schreiben und schließlich in MIICOMMAND das Bit für einen Schreibbefehl setzen. Danach wird das Register MIISTATUS überwacht, um das Ende des Schreibzugriffes festzustellen. Ein Lesezugriff mit `PHY_Read` verläuft ganz ähnlich. Nach der Adressierung wird durch MIICOMMAND der Lesevorgang gestartet. MII_STATUS zeigt dessen Abschluss an, nachdem die empfangenen Daten aus MIIRX_DATA gelesen werden können.

9.3 Lesen und Schreiben der MAC- und Phy-Register (eth_reg.c)

Das Programm dient dazu vom Steuer-PC aus, die Gigabit-Ethernet-Schnittstelle zu kontrollieren. Es ermöglicht den einfachen Zugriff auf die Register des Ethernet-MACs und des Phys. Dafür wird die Bibliothek `eth_corelib.h` mit den darin zur Verfügung gestellten Funktionen verwendet. Beim Programmaufruf muss angegeben werden, auf welchem Nathan-Modul und an welchem Slow-Control-Modul sich der Ethernet-MAC befindet, denn das ist je nach Design und Systemaufbau verschieden. Gleichzeitig trifft man die Auswahl zwischen den Registern des Ethernet-MACs und des Phys und übergibt im Falle eines Schreibzugriffes Adresse und Daten. Bei einem Lesezugriff wird nur die Adresse benötigt, wobei für MAC und Phy die Möglichkeit besteht, alle Register auf einmal auszulesen.

9.4 Bibliothek für PC (eth_pclib.h)

Die Bibliothek eth_pclib.h definiert einige Konstanten wie die MAC-Adresse des Ethernet-Cores und den verwendeten Ethernet-Typ sowie die maximale zu versendende Datenmenge pro Frame `MAX_DATA_SIZE`. Außerdem enthält sie die Deklarationen der Funktionen `ETH_Open`, `ETH_Close`, `ETH_Filter`, `ETH_Recv` und `ETH_Send`.

9.5 Implementation der Funktionen für PC (eth_pclib.c)

Die in `eth_pclib.h` implementierten Funktionen werden zum Senden und Empfangen von Frames verwendet. Die ersten vier Funktionen (`ETH_Open`, `ETH_Close`, `ETH_Filter` und `ETH_Recv`) dienen dabei dem Empfangen von Frames. Die Funktionen verwenden die Bibliothek `pcap.h` [3] [4]. Zusätzlich wird die Funktion `recv_packet` dazu verwendet, das empfangene Frame auszugeben. Da sich die erstellten Funktionen bei den Tests, von PC zu PC Daten zu übertragen, als nicht ausreichend leistungsfähig herausstellten, wurde daraufhin zum Aufzeichnen der Pakete das frei verfügbare Programm `Ethereal` [9] verwendet. Aus diesem Grund sind die oben genannten Funktionen der Vollständigkeit wegen nur kurz erläutert. Für den Einsatz mit

dem Backplane-System muss das Aufzeichnen der Daten allerdings noch verbessert werden, da es bei großen Datenmengen zu einem signifikanten Verlust von Frames kommt [33]. Die Funktion `ETH_Send` zum Senden von Frames verwendet Funktionen der Bibliothek `libnet.h` [1] [34].

ETH_Open
Diese Funktion verwendet `pcap_open_live` um eine Netzwerkkarte zum Aufzeichnen von Frames zu öffnen. Das Programm versetzt dazu die Netzwerkschnittstelle in den Promiscuous-Modus, in dem alle Frames an die nächsthöhere Schicht weitergeleitet werden.

ETH_Close
Am Ende der Aufzeichnung wird mit dieser Funktion die geöffnete Netzwerkkarte durch `pcap_close` wieder geschlossen.

ETH_Filter
Diese Funktion erlaubt es, einen Filter zu definieren, so dass nicht alle Frames, die an der Netzwerkschnittstelle ankommen, aufgezeichnet werden. Dazu wird die Syntax einer an `pcap_compile` übergebenen Variablen auf einen Filter hin überprüft und dieser gegebenenfalls erstellt. Mit `pcap_setfilter` wird der Filter der Netzwerkkarte zugewiesen.

ETH_Recv
Zum Aufzeichnen der empfangenen Frames wird die Funktion `pcap_loop` verwendet. Diese ruft in einer Schleife nach jedem Frame die mittels Funktionszeiger übergebene Funktion auf, die das empfangene Frame behandelt.

recv_packet
Diese erstellte Funktion gibt Informationen über das erhaltene Frame auf dem Bildschirm aus, d.h. die einen Frame-Zähler und die Länge des Frames. Bei Ausgabe des Paketinhaltes kam es zu durch die Verzögerung der Ausgabe zu Paketverlusten. Das war der Grund für den Wechsel auf `Ethereal`.

ETH_Send
Diese Funktion verteilt eine angegebene Datei auf die benötigte Anzahl von Frames und versendet die Daten zusammen mit Header-Informationen an eine angegebene MAC-Adresse. Die CRC-32-Prüfsumme wird automatisch von der Netzwerkkarte des PCs hinzugefügt. Da nicht nur Ethernet-Frames mit `libnet` versendet werden können, muss zuerst der Übertragungstyp und die verwendete Netzwerkschnittstelle mit `libnet_init` festgelegt werden. Gleichzeitig wird ein Zeiger auf eine Variable vom Typ `libnet_t` erzeugt und initialisiert, die zur Verwendung der weiteren Funktionen benötigt wird. Der Typ für Ethernet lautet `LIBNET_LINK`. Mit `libnet_open_link` wird eine Verbindung zur angegebenen Netzwerkschnittstelle für die Datenübertragung geöffnet. `libnet_get_hwaddr` ermittelt die MAC-Adresse der Netzwerkkarte, die als Startadresse in das Frame eingetragen wird.

Mittels einer Schleife wird die angegebene Datei in Frames aufgeteilt und versendet. Dazu werden so viele Bytes, wie `MAX_DATA_SIZE` zulässt, aus der Datei gelesen und durch `libnet_build_ethernet` mit der Ziel- und Startadresse sowie dem Typ zu einem Frame zusammengesetzt. `libnet_write` versendet dieses Frame dann über die Schnittstelle.

Sind alle Daten versendet, schließt `libnet_close_link` die Verbindung zur Netzwerkkarte und `libnet_destroy` entfernt den erzeugten Zeiger.

9.6 Senden und Empfangen von Ethernet-Frames (eth_pc.c)

Das Programm eth_pc.c vollzieht die Datenübertragung zwischen PC-Netzwerkkarte und Gigabit-Ethernet-Schnittstelle. Es benutzt die in Abschnitt 9.4 beschriebene Bibliothek, um Dateien vom PC an eine angegebene MAC-Adresse zu senden und schließlich in das SDRAM auf dem Nathan-Modul bzw. dem BRAM auf der Balu-Backplane zu speichern. Die vom PC empfangenen Daten werden auf dem Bildschirm auszugeben. Dadurch können Daten zwischen PC und System mübetragen werden. Zusätzlich kann für das Empfangen von Frames ein Filter gesetzt werden. Außerdem kann eine Datei mit zufälligem Inhalt bis zu einer Größe von 20 MByte zur Übertragung erzeugt werden.

9.7 Testen des Ethernet-MACs (eth_test.c)

Das Programm `eth_test.c` dient dazu, die Register des Ethernet-MACs und des Phys mit bestimmten Werten zu beschreiben und dadurch die Datenübertragung zwischen Netzwerkkarte des PCs und Gigabit-Ethernet-Schnittstelle zu starten. Es verwendet die Funktionen aus `eth_corelib`, um die Register von MAC und von Phy zu beschreiben. Je nach Plattform, Nathan-Modul oder Balu-Backplane, unterscheiden sich die zu schreibenden Werte. Gleichzeitig besteht die Möglichkeit, über eine Textdatei die Slow-Control-Befehle für die Simulation zu erstellen. Dieses Merkmal wird von den Bibliotheken der Slow-Control bereitgestellt.

10 Tests und Ergebnisse

In den folgenden Abschnitten werden die weiteren Schritte von der HDL-Beschreibung zur Durchführung von Datenübertragungen mit Gigabit-Ethernet dargestellt. Diese sind die Simulation des Ethernet-MACs mittels Testbench (Abschnitt 10.1) und dann die Umwandlung der HDL-Beschreibung in einen Bitstream (Abschnitt 10.2), mit dem der FPGA auf dem Nathan-Modul bzw. Balu konfiguriert werden kann (vgl. Abschnitt 2.4.2). Weiter ist die Inbetriebnahme (Abschnitt 10.3) der Ethernet-Schnittstelle beschrieben. Die damit erzielten Resultate sind an den entsprechenden Stellen in Abschnitt 10.1 und 10.3 aufgeführt.

10.1 Simulation des Designs

Für die Simulation des Ethernet-MACs wurden zwei Testbenches erstellt, die die Umgebung des Moduls `n_ethernet` nachbilden und zur Simulation mit dem Simulator ModelSim genutzt werden: Die *Ethernet-Testbench* und die *Balu-Testbench*. Die Erste simuliert lediglich den Ethernet-MAC mit den entsprechenden Stimuli von außen. Die zweite Textbench simuliert die gesamte Balu-Backplane und ist in Zusammenarbeit mit Dan Husmann de Oliveira und Stefan Philipp entstanden.

Ethernet-Testbench
Mit dieser Testbench wird alleine die Funktionsweise des Ethernet-MACs getestet. Die Umgebung, in die der Ethernet-MAC eingebettet ist, ist nur für die vom Ethernet-MAC verwendeten Komponenten nachgebildet. Mit Hilfe von Konstanten kann festgelegt werden, was genau darin simuliert wird. So besteht die Möglichkeit, das DDR-SDRAM auf dem Nathan-Modul oder ein FPGA-SelectRAM als Datenspeicher zu simulieren, die beide in Form einer Verhaltensbeschreibung vorliegen. Zusätzlich lässt sich noch als Datenquelle ein Zufallsgenerator statt einer der zuvor genannten Daten-Speicher an die entsprechende Schnittstelle anschließen. Für die Nutzung des DDR-SDRAM wird die korrekte Adressierung der Daten beim Schreiben und Lesen überprüft. Die Zugriffe auf die Schnittstelle zum verwendeten Daten-Speicher werden in einem Fifo gespeichert. Gesteuert wird der Ethernet-MAC über die Slow-Control-Schnittstelle. Die Befehle dazu werden in der Testbench erzeugt und an die Schnittstelle weitergegeben. Die Testbench beinhaltet auch ein vereinfachtes Modell eines Phys. Dieser empfängt Frames vom Ethernet-MAC, speichert sie und sendet sie dann wieder zum Ethernet-MAC zurück. Durch die vorherige Protokollierung der Daten an der Schnittstelle zum Daten-Speicher können die empfangenen Daten mit den gesendeten verglichen werden. Jeder Vergleich fehlerhafter Daten führt zu einer entsprechenden Ausgabe im Simulator und wird in eine Protokoll-Datei geschrieben. Zusätzlich zu den normalen Frames können auch Pausen-Frames mit zufälligem Abstand versendet werden. Dabei wird überprüft, ob das Versenden des Pausen-Frames durchgeführt wurde und ob dieses auch beim Empfang detektiert wurde.

Paketgröße [Byte]	Datenmenge [Byte]	Anzahl Pakete	Anzahl Pausen-Frames
64	1 M	16384	267
128	1 M	8192	132
240	1048800	4370	-
384	1048704	2731	-
496	1049040	2115	-
640	1048960	1639	-
752	1049040	1395	-
896	1049216	1171	-
1008	1049328	1041	-
1152	1049472	911	-
1264	1049120	830	-
1280	1049600	820	-
1488	1049040	705	-
1520	0	0	-

Tabelle 10.1: Simulationsergebnisse

Ergebnisse mit der Ethernet-Testbench
Die DDR-SDRAM-Schnittstelle wurde durch Tests mit Zugriffstests auf korrekte Adressierung und Datenweitergabe überprüft und verifiziert. Die weitere Funktionalität der Ethernet-Testbench wurde in die Balu-Testbench integriert. Die Ergebnisse der entsprechenden Tests sind dort zu finden.

Balu-Testbench
Diese Testbench simuliert die gesamte Balu-Backplane und damit das komplette real existierende System, um vor allem die korrekte Ansteuerung des Ethernet-MACs zu verifizieren. Die Backplane wird wie das System in Hardware über die Slow-Control gesteuert. Dabei kann die Darkwing-Platine mitsimuliert werden oder nur die Stimuli, die über das SCSI-Kabel zur Backplane gelangen. Die für den Test des Gigabit-Ethernets relevanten Komponenten sind dabei die Slow-Control, das Modell des Phys und das SelectRAM des Backplane FPGAs. Letzteres kann wie schon in der Ethernet-Testbench durch einen Zufallsgenerator ersetzt werden. Die Tests entsprechen denen der Ethernet-Testbench.

Ergebnisse mit der Balu-Testbench
Tabelle 10.1 zeigt die Ergebnisse der Simulation der Gigabit-Ethernet-Schnittstelle. Für die angegebenen Paketgrößen wurde jeweils ein Speicherbereich von 1 MByte zum Versenden bereitgestellt. Die genaue Anzahl der Bytes gibt die Spalte Datenmenge an. Die Frames wurden vom Phy zurück an den Ethernet-MAC gesendet und dort an der Schnittstelle zum Speicher mit den gesendeten Daten verglichen. Die Anzahl der versendeten Frames ist in Spalte Pakete zu finden. Der Abstand der Frames betrug den minimalen IFG von 12 Takten. Bei den Paketgrößen von 64 und 128 Byte wurden zugleich auch noch in zufälligem Abstand Pausen-Frames versendet und wieder vom Ethernet-MAC empfangen (Spalte Pausen-Frames), diese wurden vom MAC richtig erkannt. Einzig die übertragene Datenmenge liegt fast immer über dem Wert von 1 Mbyte (=1048576). Der Grund dafür ist die Beschreibung der Buffer-Deskriptoren mit Adresswerten des Speichers, wobei die Speicherobergrenze zu spät erkannt wird.

Es kam zu keinen Übertragungsfehlern, auch wurde die zu großen Frames mit 1520 Bytes Inhalt korrekt verworfen. Die Generierung der Buffer-Deskriptoren erfolgt ebenfalls schnell genug, da keine Frames verworfen wurden. Der Zugriff auf die Register des Ethernet-MACs sowie das Starten von Sende- und Empfangsvorgängen wurde im Rahmen der genannten Tests mit überprüft. Auch bei der Ansteuerung des Ethernet-MAC über die Slow-Control traten keine Fehler auf.

10.2 Bitstream-Generierung zur Konfiguration des FPGAs

Um die FPGAs auf der Balu-Backplane und dem Nathan-Modul mit dem Ethernet-Design zu konfigurieren, musste die HDL-Beschreibung des FPGAs der entsprechenden Plattform mit dem darin enthaltenen Ethernet-MAC in einen Bitstream umgewandelt werden (vgl. Abschnitt 2.4.2). Die Generierung des Bitstreams ist speziell auf die verwendete Plattform angepasst, auf der der Ethernet-MAC eingesetzt werden soll. Sie unterscheidet sich somit für das Nathan-Modul bzw. die Balu-Backplane. Zusätzlich zur HDL-Beschreibung müssen die Constraints, d.h. die Randbedingungen, die dabei zu beachten sind, angegeben werden. Für den Ethernet-MAC sind das die Angaben, auf welchen Pins und Leitungen welcher Takt verwendet wird (vgl. Abschnitt 2.4.2.4). Dazu wird das entsprechende Leitungsnetz mit einem Namen versehen und die maximale Laufzeit, also die Periodendauer des Taktes, zwischen den Registern dieses Netzes festgelegt. Bei Gigabit-Ethernet sind das 8 ns für WB_CLK und GTX_CLK, die aus derselben Quelle stammen, und ebenfalls 8 ns für die RX_CLK, die vom Phy kommt. Das entspricht der Frequenz der Takte von 125 MHz.

Bitstream-Generierung für das Nathan-Modul
Die Bitstream-Generierung für das Nathan-Modul ist mehrfach nach korrekter Angabe der Constraints nicht erfolgreich gewesen. Der Place-and-Route-Vorgang, in dem die einzelnen Elemente des Design auf den FPGA abgebildet werden, konnte die Zeitvorgaben nicht einhalten (vgl. Abschnitt 2.4.2.4). Der Grund dafür liegt an der Tatsache, dass auf dem Nathan-Modul zusätzlich noch andere Module, wie z.B. das zur Ansteuerung des DDR-SDRAMs, untergebracht sind. Dadurch reduzieren sich die Möglichkeiten der Platzierung und die Laufzeit zwischen den Registern nimmt zu. Mit Hilfe von Timing-Analysen ließen sich die problematischen Stellen ausfindig machen. So war in einem Fall zwischen der Generierung der Länge des empfangenen Frames und der Speicherung der Frame-Länge ein kritischer Pfad, d.h. zwischen dem Register am Anfang und am Ende der Strecke lag zuviel asynchrone Logik, um die maximale Periodendauer einzuhalten. Nach Überprüfung anhand der HDL-Beschreibung und der Simulation konnte in diesen Pfad ein Register eingefügt werden, was zur Einhaltung der Zeitvorgaben geführt hat. Der Bitstream wurde daraufhin erfolgreich erzeugt.

Bitstream-Generierung für Balu
Im Gegensatz zum Nathan-Modul war die Umwandlung der HDL-Beschreibung für den Balu-FPGA in einen Bitstream ohne Probleme möglich. Dies liegt daran, dass der FPGA bislang nur die Ansteuerung der Nathan-Module zu bewältigen hatte. Der Ethernet-MAC war somit das einzige größere Modul, das auf dem FPGA untergebracht werden musste. Es wird in Zukunft mindestens noch die Implementation der Steuerlogik, mit der das System durch Ethernet kontrolliert werden soll, hinzukommen.

10.3 Inbetriebnahme von Gigabit-Ethernet und Ergebnisse

Vor den Tests mit dem Phy-Modul musste die Gigabit-Ethernet-Fähigkeit des verwendeten PCs getestet werden. Dazu wurden mit Unterstützung von Martin Trefzer die beiden erworbenen Gigabit-Ethernet-Netzwerkkarten in je einen PC mit Linux-Betriebssystem eingebaut und die entsprechenden Treiber installiert. Mit Hilfe des erstellten Programms `eth_pc` wurden Daten von einem PC zum anderen übertragen (Abschnitt 9.6), wo sie dann mit `Ethereal` [9] aufgezeichnet wurden. Das gelang zu Anfang nicht. Nach einem Umstieg auf eine andere Betriebssytem-Version mit aktuellem Kernel (*Ubuntu 6.06* [2]) funktionierten die Übertragungen ohne Probleme. Die beiden genannten Programme wurden auch für die späteren Tests der Datenübertragung verwendet.

Inbetriebnahme auf dem Nathan-Modul
Die Tests zur Funktionsfähigkeit der Gigabit-Ethernet-Schnittstelle verliefen in mehreren Schritten. Zuerst wurde der Zugriff auf die Register von MAC und Phy sowie im Anschluss der im Phy integrierte Selbsttest durchgeführt [23]. Zuletzt wurde versucht, Daten mit Gigabit-Ethernet zu übertragen.

Zuerst erfolgte die Konfiguration des Nathan-FPGAs. Danach war der Zugriff auf die Register des Ethernet-MACs über die Slow-Control, die ihn steuert, möglich, ein Auslesen der Phy-Register jedoch nicht. Bei Messungen mit dem Oszilloskop zeigte sich ein starkes Übersprechen der 125 MHz schnellen GTX_CLK zum Phy auf die 2,5 MHz schnelle MDC, die dem Zugriff auf die Phy-Register dient. Das Übersprechen lag am parallelen Verlauf der Leitungen auf der Fädel-Platine vom ANN-Sockel zum Phy-Modul sowie auf dem Nathan-Modul zwischen FPGA und ANN-Sockel (vgl. Abschnitt 5.2). Die Veränderung der Lage der Fädeldrähte zueinander und die Verwendung anderer Pins auf dem ANN-Sockel reduzierte die gegenseitige Störung der Leitungen, so dass Zugriffe auf die Phy-Register möglich wurden. Eine genaue Auflistung der verwendeten Pins des ANN-Sockels und des Santa-Cruz-Steckers des Phy-Moduls befindet sich in Anhang A.

Das nächste Problem trat bei der Verbindung von Phy-Modul und der Gigabit-Ethernet-Netzwerkkarte im PC auf. Zu Beginn einer Verbindung müssen zwischen den beiden verbundenen Ethernet-Schnittstellen bestimmte Übertragungseigenschaften wie z.B. die Übertragungsgeschwindigkeit ausgehandelt werden. Das geschieht mit dem sogenannten *Auto-Negotiation-Verfahren* [13]. Erst nach dessen Ende ist eine Datenübertragung möglich. Diese schlug bei direkter Verbindung zwischen Netzwerkkarte und Phy-Modul fehl. Im Normalfall erfolgt die Auto-Negotiation innerhalb weniger Sekunden, in diesem Fall jedoch war selbst nach mehreren Minuten noch keine Verbindung hergestellt. Nach dem Einfügen eines Gigabit-Ethernet-Switches zwischen die beiden Hosts gelang die Auto-Negotiation (vgl. Abschnitt 2.1).

Das Versenden von Daten wurde auf zwei Arten getestet: Mit Hilfe des Phy-Selbsttests (*BIST*[1]), bei dem der Phy Frames generiert und überträgt [23], sowie durch das Versenden von Frames mit dem erstellten Ethernet-MAC. Bei beiden Versuchsarten wurden am PC keine Daten registriert. Die LEDs am Phy-Modul und am Switch zeigten zwar Aktivität zwischen beiden an, jedoch nicht zwischen Switch und PC. Bei den Frames des BIST lässt sich das dadurch erklären, dass diese nicht an die Netzwerkkarte des PCs adressiert waren und so bereits vom Switch nicht an diesen weitergeleitet wurden. Bei den Frames des Ethernet-MACs

[1] **B**uilt-**I**n **S**elf **T**est

haben möglicherweise Übertragungsfehler zwischen MAC und Phy zu falscher Adressierung geführt.

Der Empfang von Daten wurde ebenfalls mit dem BIST unter gleichzeitiger Verwendung des Phy-Loopbacks überprüft [23]. Dabei werden die Daten, die eigentlich über das Kabel übertragen werden sollten, vom Phy auf die GMII-Leitungen zum Ethernet-MAC zurückgekoppelt und so von diesem wie Empfangsdaten behandelt. Außerdem wurden vom PC aus Daten an das Phy-Modul gesendet. Dabei hat sich das Phy-Modul jedesmal direkt nach Beginn der Übertragung in seinen Ausgangszustand zurückgesetzt, so dass erst wieder die Auto-Negotiation durchgeführt werden musste. Trotz kurzzeitiger Aktivität am Phy wurden im Ethernet-MAC keine Daten empfangen.

Ergebnis der Tests auf dem Nathan-Modul
Es war nicht möglich, die Funktionalität der erstellten Ethernet-Schnittstelle zu testen, da sowohl beim Senden als auch beim Empfangen technische Probleme eine Zugriffsmöglichkeit auf die eventuell übertragenen Daten verhinderten. Eine mögliche Erklärung für das Verhalten des Phy-Moduls könnte die trotz der zuvor durchgeführten Verbesserungen noch immer mangelhafte Signalqualität sein, die durch Übersprechen der Signale ausgelöst wird. Diese Schwierigkeiten lassen sich mit großer Wahrscheinlichkeit nur durch Ersetzen der Fädel-Platine durch eine Platine mit besserer Signalqualität lösen. Leider konnte dem im Rahmen der Arbeit nicht mehr nachgegegangen werden.

Inbetriebnahme auf Balu
Die Inbetriebnahme der Gigabit-Ethernet-Schnittstelle auf Balu erfolgte kurz vor Abgabe dieser Arbeit. Dies lag unter anderem daran, dass die Backplane erst zu diesem Zeitpunkt einsatzfähig war. Hier wurden dieselben Schritte wie auf dem Nathan-Modul durchgeführt, um die Funktionalität zu testen.

Nach der Konfiguration des Backplane-FPGAs war es auf Anhieb möglich, die Register von Ethernet-MAC und Phy mit der dafür erstellten Software zu lesen und zu beschreiben (vgl. Abschnitt 9.3). Auch die Auto-Negotiation zwischen Gigabit-Netzwerkkarte des PCs und Phy verlief reibungslos.

Mit Hilfe des Phy-BIST ließen sich erfolgreich Frames zum PC senden. Beim Senden von Frames durch den Ethernet-MAC hingegen wurden zunächst keine angekommenen Frames am PC aufgezeichnet. Sowohl der Empfang von Frames mit BIST und Phy-Loopback blieb anfangs erfolglos, als auch der Empfang der Daten, die vom PC aus versendetet wurden.

Zur Überprüfung der zwischen Phy und MAC über das GMII ausgetauschten Daten wurde in das oberste Balu-Modul ein von Stefan Philipp erstelltes Modul zur Protokollierung der übertragenen Daten eingefügt. Nach erneut durchgeführten Versuchen zeigte sich, dass in beide Richtungen zwar Bytes übertragen wurden, diese jedoch nur beim Senden dem beabsichtigten Inhalt entsprachen. Beim Empfangen fehlten Bytes oder kamen doppelt vor. Da jeweils Aktivität auf dem Kabel angezeigt wurde, deuteten diese Fehler auf ein Synchronisationsproblem zwischen Phy und MAC hin, d.h. die Daten sind zwar am entsprechenden Sender jedoch nicht mehr am Empfänger synchron zum Takt. Bei den Datenleitungen zum Phy wurde deshalb ein Register eingefügt, das die Daten zur fallenden Flanke von GTX_CLK taktet. Das bewirkt eine Phasenverschiebung um 180° zwischen Daten und Sendetakt GTX_CLK an den Ausgängen des FPGAs zum Phy-Modul. Für den Empfang der Daten wurde ein Register eingefügt,

damit die Daten gleich an den Eingängen des Ethernet-MACs zum Empfangstakt RX_CLK synchronisiert werden.

Ergebnis der Tests auf Balu
Nach diesen Änderungen konnten erfolgreich Daten zwischen PC und Backplane ohne Fehler und mit voller Geschwindigkeit übertragen werden. Die am PC durch `Ethereal` aufgezeichneten Frames und die durch den Ethernet-MAC im SelectRAM des FPGAs gespeicherten Frames wiesen keine Fehler auf. Es muss allerdings erwähnt werden, dass aufgrund der knappen Zeit bis zur Abgabe der Diplomarbeit keine ausgiebigen Test durchgeführt werden konnten. Jedoch trat bei vorläufigen Tests mit etwa 1000 Testframes bisher kein einziger Fehler auf, so dass die Implementation unter Vorbehalt ausgiebiger Tests als erfolgreich angesehen werden kann.

11 Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, eine Gigabit-Ethernet-Schnittstelle in programmierbarer Logik zu implementieren. Diese sollte dann zur Datenübertragung und Steuerung eines Systems aus Künstlichen Neuronalen Netzen eingesetzt werden. Die Schnittstelle für Ethernet besteht im Allgemeinen aus zwei Teilen: dem Phy und dem Ethernet-MAC. Als Phy wurde eine vorgefertigte Platine erworben, die diesen beinhaltet (Phy-Modul). Der Ethernet-MAC sollte auf einem FPGA implementiert werden. Dazu wurde ein IP-Core für 10- und 100-Mbit/s verwendet und dieser dann auf 1 Gbit/s erweitert. Im Nachfolgenden soll erläutert werden, inwiefern dieses Ziel erreicht wurde (Abschnitt 11.1). Außerdem wird ein Ausblick darauf gegeben, welche weiteren Entwicklungsmöglichkeiten die erstellte Schnittstelle besitzt und wie zukünftige Einsatzmöglichkeiten aussehen (Abschnitt 11.2).

11.1 Zusammenfassung der Arbeit und Ergebnisse

Zur Implementation der Gigabit-Ethernet-Schnittstelle wurden die beiden Komponenten, Phy und MAC, zunächst in das System aus Backplane und Nathan-Modulen integriert. Zu Anfang existierte nur für das Nathan-Modul die Möglichkeit der Integration, gegen Ende der Arbeit kam noch die Balu-Backplane hinzu.

Der Hauptteil der Arbeit bestand darin, den bestehenden IP-Core auf 1 Gbit/s zu erweitern, um die gewünschte Funktionalität zu erhalten. Die einzelnen Veränderungen wurden dabei immer wieder in Simulationen überprüft. Nach Verifikation des Designs wurde es auf einem Nathan-Modul in Betrieb genommen und Tests damit durchgeführt. Diese waren jedoch nur zum Teil erfolgreich. So ließ sich nach anfänglichen Problemen zwar die Steuerung des Phy-Moduls und des Ethernet-MACs im FPGA bewerkstelligen, jedoch gelangen keine Datenübertragungen. Dabei hat sich herausgestellt, dass die Fädel-Platine, die auf dem Nathan-Modul die Verbindung zum Phy-Modul herstellt, höchstwahrscheinlich zu einer starken gegenseitigen Störung der darüber geführten Leitungen führt. Dieses Problem lässt sich aber wahrscheinlich erst durch Ersetzen der Fädel-Platine lösen. Auf der Balu-Backplane, wo das Phy-Modul direkt mit der Backplane verbunden ist, gelang es dagegen, mit Gigabit-Ethernet Daten fehlerfrei zu übertragen. Die damit erreichbaren Übertragungs- und Fehlerraten zwischen PC und System bleiben noch zu ermitteln. Erste Tests waren jedoch fehlerfrei und lassen damit auf eine erfolgreiche Implementation schließen.

Der implementierte Gigabit-Ethernet-MAC erfüllt weitestgehend die Aufgaben, die einem MAC zugeordnet werden: Er sendet Ethernet-Frames mit Adressierung, die er aus gespeicherten Daten selbst zusammenstellt. Die Daten der Frames, die er empfängt, werden in einen zugeteilten Speicherbereich geschrieben, sofern die Adressüberprüfung des Frames die korrekte Zieladresse des MACs ergibt. Ansonsten verwirft der MAC das Frame. Gleichzeitig ist er in der

Lage, auf empfangene Pausen-Frames richtig zu reagieren, d.h. für die angegebene Zeitdauer keine Frames zu versenden. Auf Anfrage versendet der MAC auch selbst Pausen-Frames. Zusammenfassend lässt sich die Gigabit-Ethernet-Schnittstelle jetzt schon zur Übertragung von Daten nutzen. Falls dabei Fehler auftreten, werden diese mittels der CRC-32-Prüfsumme durch den MAC erkannt (vgl. Abschnitt 6.1.1 und 7.11).

11.2 Ausblick auf die zukünftige Verwendung

Ausführliche Tests der Schnittstelle, die zeigen wie schnell und zuverlässig Daten zwischen PC und System ausgetauscht werden können, stehen noch aus. Selbst bei guten Ergebnissen sollte eine Weiterentwicklung angestrebt werden, denn die Möglichkeiten, die Leistung zu erhöhen, sind noch nicht ausgeschöpft. Die Leistungsfähigkeit von Gigabit-Ethernet lässt sich z.B. generell durch Vergrößerung der Frame-Größen verbessern, sogenannte *Jumbo-Frames* [7]. Außerdem ist es kein Problem, den MAC in anderen Systemen zu verwenden. Lediglich der Anschluss an das SDRAM ist momentan spezifisch für das vorliegende System mit Nathan-Modulen und Balu-Backplane. Überdies kann die Flexibilität des Gigabit-Ethernet-MACs zusätzlich erhöht werden, indem ein automatischer Betrieb mit den Geschwindigkeiten 10 Mbit/s, 100 Mbit/s und 1 Gbit/s implementiert wird. Da der Ethernet-MAC ursprünglich für die niedrigeren Geschwindigkeiten konzipiert war und jetzt mit Gigabit-Ethernet betrieben wird, müssen dazu nur wenige Änderungen durchgeführt werden.

All die genannten Maßnahmen können zu einer universell verwendbaren Technik führen, die Daten schnell und vor allem sicher zu übertragen vermag. Zusätzlich kann in Erwägung gezogen werden, ob dieser verbesserte Gigabit-Ethernet-MAC als IP-Core frei zur Verfügung gestellt wird, so wie es der ursprüngliche IP-Core für 10- und 100-Mbit/s-Ethernet war.

Schon jetzt stellt die Gigabit-Ethernet-Schnittstelle in ihrer Form einen Beitrag zur Erhöhung der Leistungsfähigkeit und der Flexibilität des Backplane-Systems dar. Es ist möglich, die Backplane ohne große Anstrengungen mit nahezu jedem PC zu nutzen, und nach Realisierung des vorgestellten Konzepts auch zu steuern. Durch die Erhöhung der Geschwindigkeit um eine Größenordnung bietet sich nun die Möglichkeit, größere Datenmengen für die Erforschung des künstlichen neuronalen Netzes auszuwerten. Denkbar ist auch, dass das System nicht mehr nur lokal genutzt wird, sondern durch entsprechende Anbindungen über das Internet gesteuert und ausgelesen werden kann. Vor der Realisierung dieser Ideen liegt allerdings noch ein interessanter Weg, den es sich lohnt zu gehen.

A Pinbelegung des ANN-Sockels

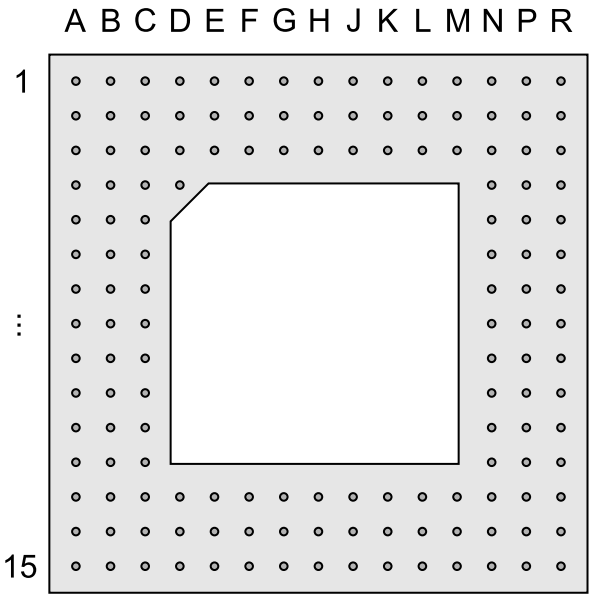


Abbildung A.1: Pinbezeichnung des ANN-Sockels

In Tabelle A.1 sind sämtliche Pins des Phy-Moduls aufgeführt. Diese sind so angeordnet, wie es die Spalte *Santa-Cruz-Stecker* angibt [18]. Der damit verbundene Pin auf dem ANN-Sockel ist in Spalte *ANN-Sockel* angegeben. Dessen Position zeigt Abbildung A.1. Die Spalte *Phy-Modul* enthält die Bezeichnung des Pins, die in der Beschreibung des Phy-Moduls verwendet wird. In der letzten Spalte *Design* steht der Name, der für den entsprechenden Pin in `nathan_top` verwendet wird. Vor dem aufgelisteten Namen befindet sich jeweils noch das Präfix `ANNA_BUS_`, vor `RXclk` steht `ANNA_CLK2_`.

Santa-Cruz-Stecker	ANN-Position	Phy-Modul	Design	Santa-Cruz-Stecker	ANN-Position	Phy-Modul	Design
1	K2	GND		1	P9	RSTn	N22
2				2	R4	GND	
3				3	J14	RXD0	N0
4				4	C4	TXclk	N18
5				5	L14	RXD2	N2
6				6	K14	RXD1	N1
7				7	F14	RXD4	N4
8				8	J13	RXD3	N3
9				9	E13	RXD6	N6
10	A8	MDIO	N15	10	E14	RXD5	N5
11				11	A5	RXerr	N17
12	A9	MDC	N14	12	D14	RXD7	N7
13				13	B6	RXcrs	N16
14	P3	INT	N29	14	A4	RXdv	N20
				15	P6	TXen	N26
				16	B3	RXcol	N19
				17	P7	TXerr	N25
				18	R13	TXD7	N13
				19	P8	GND	
				20			
1				21	N11	TXD6	N12
2	G14	GND		22	C7	GND	
3				23	R14	TXD5	N11
4	G14	GND		24	C7	GND	
5	J2	3.3V		25	F2	TXD4	N34
6	G14	GND		26	C7	GND	
7	G1	3.3V		27	F3	TXD3	N33
8	G13	GND		28	L1	TXD2	N32
9				29	M2	TXD1	N31
10	G13	GND		30	B4	GND	
11	D1	GTXclk	N35	31	K3	TXD0	N30
12	G13	GND		32			
13	A10/C9	RXclk	P/N	33	R9	PHYad0	N23
14	G13	GND		34	B4	GND	
15				35	B2	ANen	N21
16	B13	GND		36			
17				37	P4	DUPLEX	N28
18	B13	GND		38			
19				39			
20	B13	GND		40	B4	GND	

Tabelle A.1: Pinbelegung des ANN-Sockels

B Module des Ethernet-MACs

In Abbildung B.1 ist die Hierarchie der Module des Gigabit-Ethernet-MACs dargestellt. Gleichzeitig ist angegeben, ob das Modul verändert wurde oder nicht. Die Änderungen der einzelnen Module sind in der nachfolgenden Tabelle aufgeführt.

Modul	Sprache	verändert	Änderung
`asyn_fifo_32_64`	Verilog	ja	• neues Modul • Core-Generator-Modul, das RX-Fifo ersetzt
`asyn_fifo_64_8`	Verilog	ja	• neues Modul • Core-Generator-Modul, das TX-Fifo ersetzt
`bram_interface`	VHDL	ja	• neues Modul • WISHBONE-Master-Interface zum SelectRam auf dem FPGA
`bram32x256`	Verilog	ja	• neues Modul • Core-Generator-Modul, das FPGA-spezifisches Block SelectRAM verwendet
`crc32_d`	Verilog	ja	• neues Modul • ersetzt `eth_crc` • bindet die Funktionen `CRC32_D4` und `CRC32_D8` ein
`CRC32_D4`	Verilog	ja	• neue Funktion • wird von `crc32_d` zur Berechnung der CRC-32-Prüfsumme bei 4-Bit-Datenbreite eingebunden
`CRC32_D8`	Verilog	ja	• neue Funktion • wird von `crc32_d` zur Berechnung der CRC-32-Prüfsumme bei 8-Bit-Datenbreite eingebunden
`eth_clockgen`	Verilog	nein	–
`eth_crc`	Verilog	ja	• unverändert bei 100-Mbit/s-Ethernet • ersetzt durch `crc32_d` bei Gigabit-Ethernet für die Berechnung der CRC-Prüfsumme in 4- und 8-Bit-Worten
`eth_fifo`	Verilog	ja	• unverändert bei 100-Mbit/s-Ethernet • ersetzt durch `asyn_fifo_32_64` und `asyn_fifo_64_8` bei Gigabit-Ethernet
`eth_maccontrol`	Verilog	ja	• Signalweiterleitung an `eth_transmitcontrol`
`eth_macstatus`	Verilog	ja	• Register für `ReceivedLengthOK` wegen kritischem Pfad
`eth_miim`	Verilog	nein	–
`eth_outputcontrol`	Verilog	nein	–
`eth_random`	Verilog	nein	–
`eth_rxcounters`	Verilog	ja	• IFGCount auf Gigabit-Ethernet-Wert korrigiert
`eth_receivecontrol`	Verilog	ja	• PauseTimer für Gigabit-Ethernet korrigiert

Fortsetzung

Modul	Sprache	verändert	Änderung
`eth_register`	Verilog	nein	–
`eth_registers`	Verilog	ja	• Core-Register für Generierung der Buffer-Deskriptoren hinzugefügt (TX_RAM_L, TX_RAM_H, TX_BD_DEF, RX_RAM_L, RX_RAM_H, RX_BD_DEF und BD_INT) und mit `eth_wishbone` verbunden, RAM-Bereich wird in Vielfachen von 16 Bytes abgespeichert • Core-Register für Frame-Header hinzugefügt (MAC_DEST0 und MAC_DEST1) und Signale für Ziel-Adresse und Ethertyp mit `eth_txethmac` verbunden
`eth_rxaddrcheck`	Verilog	ja	• ByteCnt für Gigabit-Ethernet reduziert
`eth_rxethmac`	Verilog	ja	• 28-stufiges Schieberegister hinzugefügt, dass `RxStartFrm` um jeweils pro Takt um eins verschiebt (14-stufig bei Gigabit-Ethernet)
`eth_rxstatem`	Verilog	ja	• Unterscheidung zwischen StateData[0] und StateData[1] aufgehoben
`eth_shiftreg`	Verilog	nein	–
`eth_spram_256x32`	Verilog	ja	• durch neues Unter-Modul `bram32x256` an verwendeten FPGA angepasst
`eth_transmitcontrol`	Verilog	ja	• Signale `not_finished` und `CtrlMux` zur richtigen Pausen-Frame-Generierung mit `eth_txethmac` verbunden • Byte-Zähler zur Erzeugung von Pausen-Frames für Gigabit-Ethernet geändert
`eth_top`	Verilog	ja	• Weiterleitung von Signalen
`eth_txethmac`	Verilog	ja	• 14-stufiges Schieberegister mit seriellem Ein- und Ausgang für das Anfügen des Frame-Headers und ein zweites für die Verzögerung des `TxEndFrm`-Signals hinzugefügt, dass Daten jeden zweiten Takt (bei Gigabit-Ethernet jeden Takt) um ein Byte verschiebt • einzelne Felder der Frames an Gigabit-Ethernet angepasst: `DataCrc` und Präambel
`eth_txstatem`	Verilog	ja	• unverändert bei 100-Mbit/s-Ethernet • Unterscheidung zwischen `StateData[0]` und `StateData[1]` aufgehoben

Fortsetzung

Modul	Sprache	verändert	Änderung
`eth_wishbone`	Verilog	ja	• FSM für Generierung der Buffer-Deskriptoren hinzugefügt • Zugriff auf Buffer-Deskriptoren durch FSM beschleunigt • `eth_fifo` durch asynchrone Fifos `asyn_fifo_32_64` und `asyn_fifo_64_8` ersetzt
`n_ethernet`	VHDL	ja	• neues Modul • enthält Slow-Control-Schnittstelle zu WISHBONE-Slave-Interface
`n_ethernet_ram_interface`	VHDL	ja	• neues Modul • WISHBONE-Master-Interface zu DDR-SDRAM auf dem Nathan-Modul • modifiziert für die Entfernung der CRC-32-Prüfsumme • für Gigabit-Ethernet komplett geändert
`tx_counters`	Verilog	ja	• unverändert bei 100-Mbit/s-Ethernet • Zählerstandsabfragen `NibCntEq7` und `NibCntEq7` an Gigabit-Ethernet angepasst, sowie Minimal-Framelänge `NibbleMinFl` korrigiert • Unterscheidung zwischen `StateData[0]` und `StateData[1]` aufgehoben
`xilinx_dist_ram_16x32`	Verilog	ja	• unverändert bei 100-Mbit/s-Ethernet • fällt durch `asyn_fifo_32_64` und `asyn_fifo_64_8` bei Gigabit-Ethernet weg

Tabelle B.1: Alle Module des Ethernet-MACs mit deren Änderungen (alphabetisch sortiert)

Abbildung B.1: Hierarchie des Ethernet-MACs (Ellipse: Module des Ethernet-Cores, Kasten: neue Module des Ethernet-MACs, weiß: unverändert, grau: von mir verändert)

C Register des Ethernet-MACs

Tabelle C.1 gibt eine Übersicht über die Register des Ethernet-MACs. Die ersten bis TX_CTRL stammen aus dem Ethernet-Core und sind in [22] einzeln aufgeführt und beschrieben. Die restlichen Register habe ich hinzugefügt, diese sind in Kapitel 7 an den entsprechenden Stellen erklärt.

Name	Adresse	Byte	Beschreibung
MODER	0x00	0x00	Modus-Register
INT_SOURCE	0x01	0x04	Interrupt-Quellen-Register
INT_MASK	0x02	0x08	Interrrupt-Maske-Register
IPGT	0x03	0x0C	Back to Back Inter Packet Gap Register
IPGR1	0x04	0x10	Non Back to Back Inter Packet Gap Register 1
IPGR2	0x05	0x14	Non Back to Back Inter Packet Gap Register 2
PACKETLEN	0x06	0x18	Paket-Längen-Register
COLLCONF	0x07	0x1C	Collision and Retry Configuration Register
TX_BD_NUM	0x08	0x20	Transmit Buffer-Deskriptor Number Register
CTRLMODER	0x09	0x24	Control Module Mode Register
MIIMODER	0x0A	0x28	MII Moder Register
MIICOMMAND	0x0B	0x2C	MII Command Register
MIIADDRESS	0x0C	0x30	MII Address Register
MIITX_DATA	0x0D	0x34	MII Transmit Data
MIIRX_DATA	0x0E	0x38	MII Receive Data
MIISTATUS	0x0F	0x3C	MII Status Register
MAC_ADDR0	0x10	0x40	MAC Address Register 0
MAC_ADDR1	0x11	0x44	MAC Address Register 1
HASH0	0x12	0x48	HASH Register 0
HASH1	0x13	0x4C	HASH Register 1
TXCTRL	0x14	0x50	TX Control Register
TX_RAM_L	0x15	0x54	TX RAM Low
TX_RAM_H	0x16	0x58	TX RAM High
TX_BD_DEF	0x17	0x5C	TX Buffer-Deskriptor Default
RX_RAM_L	0x18	0x60	RX RAM Low
RX_RAM_H	0x19	0x64	RX RAM High
RX_BD_DEF	0x1A	0x68	RX Buffer-Deskriptor Default
BD_INT	0x1B	0x6C	Buffer-Deskriptor Interrupt
MAC_DEST0	0x1C	0x70	MAC Destination Address 0
MAC_DEST1	0x1D	0x74	MAC Destination Address 1

Tabelle C.1: Ethernet-Core Register

Literaturverzeichnis

[1] *libnet.* – http://libnet.sourceforge.net/

[2] *Ubuntu.* – http://www.ubuntu.com/

[3] *Manpage of PCAP – C Library Functions (3).* 2003. – http://www.tcpdump.org/pcap3_man.html

[4] Carstens, T. : *Programming with pcap.* – http://www.tcpdump.org/pcap.htm

[5] Kap. 9 In: Dally, W. J. ; Poulton, J. W.: *Digital Systems Engineering.* Cambridge University Press, 1998, S. 394ff.

[6] devboards: *DBGIG1.* 1.0.1. – http://www.devboards.de/pdf/DBGIG1.pdf

[7] Dykstra, P. : *Gigabit Ethernet Jumbo Frames.* Dezember 1999. – http://sd.wareonearth.com/~phil/jumbo.html

[8] Easics: *CRC Tool.* – http://www.easics.com/

[9] Ethereal: *Homepage.* – http://www.ethereal.com/

[10] Fast Analog Computing with Emergent Transient States (FACETS). *Homepage.* http://www.facets-project.org

[11] Fieres, J. ; Grübl, A. ; Philipp, S. ; Meier, K. ; Schemmel, J. ; Schürmann, F. : A Platform for Parallel Operation of VLSI Neural Networks. In: *Proc. of the 2004 Brain Inspired Cognitive Systems Conference (BICS2004).* University of Stirling, Scotland, UK, 2004

[12] Grübl, A. . *Eine FPGA-basierte Platform für neuronale Netze.* Diploma thesis (german), University of Heidelberg, HD-KIP-03-2. 2003

[13] Hernandez, R. : Gigabit Ethernet – Auto-Negotiation. In: *Dell Power Solutions* 1 (2001), S. 117 ff.

[14] IEEE: *IEEE 802.5: Token Ring Access Method.* 1998. – http://standards.ieee.org/getieee802/

[15] IEEE: *IEEE 802.3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method.* 2005. – http://standards.ieee.org/getieee802/

[16] IEEE: *EtherType Field listing.* Februar 2007. – http://standards.ieee.org/regauth/ethertype/eth.txt

[17] IEEE: *Organizationally Unique Identifier (OUI) listing.* Februar 2007. – http://standards.ieee.org/regauth/oui/oui.txt

[18] Kluge, K. : *Santa Cruz Connector Spec.* Rev 7. Altera Croporation, März 2004. – http://www.altera.com/

[19] Maenner, R. . *Metastable States in Asynchronous Digital Systems: Avoidable or Unavoidable*. 1988

[20] Millner, S. : *Ethernetaccess for Nathan*. – Miniforschungsarbeit

[21] Mohor, I. : *Ethernet IP Core Design Document*. Rev. 0.4, November 2002. – `http://www.opencores.org/projects.cgi/web/ethmac/overview/`

[22] Mohor, I. : *Ethernet IP Core Specification*. Rev. 1.19, November 2002. – `http://www.opencores.org/projects.cgi/web/ethmac/overview/`

[23] National Semiconductor: *DP83865 Gig PHYTER V 10/100/1000 Ethernet Physical Layer*. Oktober 2004. – `http://www.national.com/ds.cgi/DP/DP83865.pdf`

[24] OpenCores: *Homepage*. – `http://www.opencores.org/`

[25] OpenCores Organization: *WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*. Rev. B.3. September 2002. – `http://www.opencores.org/projects.cgi/web/wishbone/wishbone`

[26] Peterson, L. L. ; Davie, B. S.: *Computernetze*. dpunkt.lehrbuch, 2004

[27] Philipp, S. : *Doktorarbeit in Arbeit*, Uni Heidelberg, Diss.

[28] Rech, J. : Volldampf voraus ... – Die Technik von Gigabit Ethernet. In: *c't* 13 (1998), S. 212 ff.

[29] Rech, J. : Jetzt klappt's auch mit Kupfer – 1000Base-T: Gigabit Ethernet über Twisted-Pair-Kabel auf Kupferbasis. In: *c't* 14 (2000), S. 208 ff.

[30] Schemmel, J. ; Gruebl, A. ; Meier, K. ; Mueller, E. : Implementing Synaptic Plasticity in a VLSI Spiking Neural Network Model. In: *Proceedings of the 2006 International Joint Conference on Neural Networks (IJCNN'06)*, IEEE Press, 2006

[31] Schemmel, J. ; Hohmann, S. ; Meier, K. ; Schürmann, F. : A Mixed-Mode Analog Neural Network using Current-Steering Synapses. In: *Analog Integrated Circuits and Signal Processing* 38 (2004), Nr. 2-3, S. 233–244

[32] Schmitz, T. : *Evolution in Hardware – Eine Experimentierplattform zum parallelen Training analoger neuronaler Netzwerke*, Ruprecht-Karls-University, Heidelberg, Diss., 2005

[33] Schneider, F. ; Wallerich, J. ; Feldmann, A. ; Sommer, R. : *High Performance Packet Capture*. – `http://www.net.t-labs.tu-berlin.de/research/bpcs/`

[34] Seemann, O. : *IP Protokoll Programmierung mit libnet und libpcap*. 2004. – `http://www.oebs.net/doc/`

[35] Sinsel, A. . *Linuxportierung auf einen eingebetteten PowerPC 405 zur Steuerung eines neuronalen Netzwerkes*. Diploma thesis (german), University of Heidelberg, HD-KIP-03-14. 2001

[36] Tanenbaum, A. S.: *Computer Networks*. 4. Pearson Eduction International, 2002

[37] Xilinx, Inc.: *Development System Reference Guide*. – `http://www.xilinx.com/`

[38] Xilinx, Inc.: *Virtex-II Pro FPGA Data Sheet*. Oktober 2005. – `http://www.xilinx.com/`

[39] Xilinx, Inc.: *Virtex-II Pro FPGA User Guide*. März 2005. – `http://www.xilinx.com/`

[40] Zimmermann, H. : OSI Reference Model. In: *IEEE* 4 (1980), S. 425 ff.

Danksagung

Abschließend möchte ich mich bei allen, die zum Gelingen dieser Arbeit beigetragen haben, herzlich bedanken. Mein Dank gilt vor allem

- Herrn Prof. Dr. Karlheinz Meier für die Aufnahme in seine Arbeitsgruppe. Er hat mir die Möglichkeit gegeben, meinen Beitrag zu einem interessanten Projekt leisten zu können.
- Herrn Prof. Dr. Udo Kebschull für die Begutachtung der Arbeit.
- Dr. Johannes Schemmel für die Führung während dem Verlauf der Arbeit.
- Stefan Philipp für seine Bereitschaft, Fragen zu beantworten. Er war ein wichtige Stütze, die mir immer hilfreich zur Seite stand.
- Andreas Grübl, der bei technischen Fragen gerne seine Erfahrungen weitergegeben hat. Viel Erfolg beim Fertigstellen deiner Doktorarbeit.
- Dan Husmann de Oliveira für die Entwicklung der Balu-Backplane, ohne die ein erfolgreicher Abschluss der Arbeit nicht möglich gewesen wäre.
- Dr. Tillmann Schmitz, der immer für Fragen zur Verfügung stand.
- Dr. Martin Trefzer für die Installation der Gigabit-Ethernet-Netzwerkkarten und die Hilfe mit den entsprechenden PCs.
- allen Mitgliedern der Electronic Vision(s)-Arbeitsgruppe für die sympathische Atmosphäre. Die Zusammenarbeit mit Euch war sehr angenehm.

Abgesehen von der Unterstützung bei der Arbeit, möchte ich außerdem noch Allen meinen Dank aussprechen, auf die ich in den letzten Jahren immer bauen konnte. Dies gilt in besonderem Maße für

- meine Eltern, die mich auf meinem Weg immer unterstützt haben. Sie haben mein Studium erst ermöglicht.
- meine Freundin Bettina. Sie beschert mir immer wieder wundervolle gemeinsame Momente und hat mir durch ihre Aufopferung in den letzten Monaten viel Kraft gegeben.