DIPLOMARBEIT

Systemdesign und Entwicklung einer Netzwerkkarte zur Uhrensynchronisation

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs unter der Leitung von

o. Univ. Prof. Dipl. Ing. Dr. techn. Dietmar Dietrich
und
Univ. Ass. Dipl. Ing. Dr. techn. Thilo Sauter
und
Mag. Dipl. Ing. Georg Gaderer
als verantwortlich mitwirkendem Assistenten am
Institut für Computertechnik
Institutsnummer: 384

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von

Reinhard Exel
Matr.Nr. 0025899
Wolfshof 6, 3571 Gars am Kamp

Wien, 9. Mai 2007 _____

Kurzfassung

Die Synchronisation der Uhren in jedem Netzwerkknoten ist von elementarer Bedeutung für Echtzeitanwendungen in einem verteilten System. In jüngster Zeit rückt die Verwendung von Ethernet zur Verbindung der Netzwerkknoten vor allem in der Automatisierungstechnik in den Vordergrund, da hierdurch ein einheitliches Netzwerk bis zur Feld-Ebene ermöglicht wird. Der IEEE 1588-Standard beschreibt ein Verfahren zur hochgenauen Uhrensynchronisation in paketbasierten Netzen. Mithilfe von Hardwareunterstützung zum Ziehen und Speichern der Sende- und Empfangszeiten von Synchronisationspaketen ist es möglich, die Genauigkeit der Uhrensynchronisation auf bis zu 100 ns zu verbessern. Diese Arbeit befasst sich mit der Erstellung eines Netzwerkadapters, welcher es handelsüblichen Computern mit PCI-Bus ermöglicht, ihre Uhren gemäß dem IEEE 1588-Standard zu synchronisieren. Es zeigte sich, dass kommerziell verfügbare Funktionsblöcke zur Erstellung der Netzwerkkarte nicht besser geeignet sind als frei verfügbare.

Abstract

The synchronisation of the clocks in each network node is an essential interest for real-time applications in a distributed system. Recently the use of Ethernet for the connection of network nodes has come to the fore, particularly in automation engineering because this permits a common network down to field level. The IEEE 1588 standard describes a method for highly accurate clock synchronization in packet based networks. With the assistance of hardware support for retrieving and storing the point in time, when sending or receiving a synchronization packet, it is possible to improve the accuracy of clock synchronization up to around 100 ns. This thesis addresses the creation of a network interface controller, which facilitates commercial computers with a PCI-Bus to synchronize their clocks according to the IEEE 1588 standard. It showed that commercial available function blocks were not better suited for the creation of the network adapter than free ones.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Für Netzwerke in der industriellen Automatisierungstechnik und Prozesssteuerung ist ein definiertes Zeitverhalten ein wichtiger Schlüsselpunkt für die Funktion und das Verhalten eines Netzes. Um dieses strenge Echtzeitverhalten erfüllen zu können, wurden in Vergangenheit zahlreiche Feldbussysteme entwickelt, die auf einem eigenen Protokoll basieren und eine eigene Verkabelung benötigen [15].

In jüngster Zeit besteht ein vermehrtes Interesse am Einsatz von Ethernet bis hin zur Feld-Ebene. Die Verwendung von Ethernet ist begründet durch den Wunsch nach einer einheitlichen Netzwerkfunktionalität und der damit ermöglichten vertikalen Integration in der Automatisierungstechnik. Damit einhergehend ist eine Kostenreduktion durch den Einsatz von Standard Hardware und Verkabelung. Ethernet wurde jedoch ursprünglich nicht für Automatisierungsaufgaben und deren enge zeitliche Grenzen entwickelt. Es basiert nicht wie viele Feldbussysteme auf einem Zeitschlitzverfahren, bei dem jeder Knoten eine vordefinierte Zeitspanne zum Austausch von Nachrichten im Netzwerk erhält, sondern auf dem zufälligen Buszugriffsverfahren CSMA/CD. Um Ethernet ein deterministisches Verhalten zu verleihen, gibt es mehrere Ansätze. Manche Feldbushersteller verwenden Ethernet-Hardware mit einem proprietären Protokoll in der Meinung, dass herkömmliches Ethernet für Echtzeitanwendungen ungeeignet sei. Das macht diese Feldbusse nur kostengünstiger, ermöglicht jedoch kein homogenes Netzwerk, da keine Interoperabilität zu gewöhnlichem Ethernet gegeben ist [12][27].

Dem Ethernet den gewünschten Determinismus des Zeitverhaltens zu verleihen versucht der IEEE (Institute of Electrical and Electronics Engineers) 1588-Standard, der mit dem Precision Time Protocol (PTP) ein Verfahren zur hochgenauen Uhrensynchronisation in paketbasierten Netzwerken beschreibt. Die Synchronisation der Uhren in den Netzwerkknoten wird durch den Versand spezieller Synchronisationspakete erreicht. Jeder Knoten muss in der Lage sein, diese Pakete zu erkennen und gemäß dem PTP-Protokoll zu verarbeiten. Um eine genaue Synchronisation aller Netzwerkknoten zu erreichen, muss der Empfang und Versand der Synchronisationspakete möglichst exakt bestimmt werden, was nur durch die Implementierung einer speziellen zusätzlichen Hardware möglich ist [15].

Ein Eingriff in die Hardware gewöhnlicher COTS (commercial off-the-shelf)-Produkte ist jedoch nicht möglich, da die Funktionen der Netzwerkkarte durch das Maskenlayout des Netzwerk-ICs fix vorgegeben ist und nicht verändert werden kann. Ziel dieser Arbeit ist es einen Netzwerkadapter auf PCI-Basis zu entwickeln, der neben der gewöhnlichen Funktion einer Netzwerkkarte die Fähigkeit hat, hochgenaue Uhrensynchronisation gemäß dem PTP-Protokoll durchzuführen.

1.1 Motivation

Das Gebiet der Uhrensynchronisation in paketbasierten Netzen ist ein relativ neues. Netzwerktechnologien in Anwendungsgebieten, wo ein zeitlicher Determinismus gefordert war, wie z. B. in der industriellen Automatisierungstechnik, basierten bisher stets auf einem Verfahren, das den Zugriff auf einen gemeinsamen Bus derart streng geregelt hat, dass niemals zwei Busteilnehmer gleichzeitig den Bus für sich beanspruchen konnten. Beispiele hierfür sind TDM (Time Division Multiplex)-Systeme, Zeitmultiplexsysteme, bei denen jeder Busteilnehmer in exakt periodischen Abständen den Bus für kurze Zeit erhält. Paketbasierende Netze wie Ethernet sind von der Architektur für zeitkritische Anwendungen eher ungeeignet, da es aufgrund des Zugriffsverfahrens CSMA/CD und den dadurch möglicherweise entstehenden Kollisionen zu einer theoretisch unbeschränkten Verzögerung beim Zugriff auf das Medium kommen kann.

Der allgemeine Trend von leitungsvermittelnden Netzen zu paketvermittelnden Netzen findet nun auch in der Industrieautomatisierung Anklang. Der IEEE 1588-Standard schafft den Spagat, Ethernet einem mit klassischen TDM-Netzen vergleichbaren zeitlichen Determinismus zu verschaffen, ohne jedoch auf die Vorzüge von TCP/UDP verzichten zu müssen. Für volle Interoperabilität ist gesorgt, da IEEE 1588 den Protokollstack selbst nicht verändert, sondern nur eine hardwarenahe Logik zur Zeitstempelung einfügt, welche von der Software aus bedient werden kann. Softwarelösungen zur Uhrensynchronisation wie das Network Time Protocol verfolgen ein ähnliches Ziel, erreichen jedoch eine wesentlich gröbere Synchronisation als hardwareunterstützte Verfahren. Zahlenmäßig ausgedrückt liegen zwischen hardwareunterstützten Verfahren und reinen Softwareverfahren etwa 4 Zehnerpotenzen [9]. All dies und die Tatsache, dass IEEE 1588 als Standard im Jahr 2002 verabschiedet wurde, macht Ethernet für Automatisierungsanwendungen interessanter denn je.

1.2 Aufbau der Arbeit

Diese Diplomarbeit ist in einer strukturierten Weise aufgebaut:

Kapitel 2 beschreibt die Aufgabenstellung, das Ziel dieser Arbeit sowie technische Anforderungen an das zu erstellende System.

Kapitel 3 gibt einen Überblick über das breite Feld der Uhrensynchronisation und beschreibt die Kennwerte von Uhrensynchronisationsalgorithmen. Zudem wird auf den IEEE 1588-Standard sowie die Syn1588-Technologie eingegangen und die Motivation dargelegt, Zeitstempel mit Hardwareunterstützung zu erstellen .

Kapitel 4 analysiert die Funktion eines Netzwerkadapters und trennt diese in Funktionsblöcke auf. Es werden zwei verschiedene Implementierungskonzepte erstellt und verglichen. Für diese beiden Lösungen wird anhand von objektiven Kriterien eine Bewertung durchgeführt und eine Auswahl für das bessere Konzept getroffen.

In Kapitel 5 werden die Technologien der Funktionsblöcke beschreiben sowie die eigentliche Implementierung des in Kapitel 4 ausgewählten Konzepts abgehandelt.

Kapitel 6 widmet sich der während der gesamten Projektdauer durchgeführten Verifikationsarbeit und zeigt die Resultate der Synthese.

Das letzte Kapitel fasst die Ergebnisse dieser Arbeit zusammen und gibt Hinweise auf mögliche Weiterentwicklungen und Anpassungen.

2 Aufgabenstellung

Im Folgenden werden der grundsätzliche Funktionsumfang und die Anforderungen und Rahmenbedingungen beschrieben. Da dieser Abschnitt viele technische Abkürzungen enthält, deren genaue Erklärung an dieser Stelle den Text schlecht lesbar machen würde, sei auf das Glossar am Ende der Arbeit verwiesen.

2.1 Ziel der Arbeit

Im Rahmen dieser Arbeit soll ein Ethernet Netzwerkadapter erstellt werden, der über die Fähigkeit verfügt, hochgenaue Uhrensynchronisation durchzuführen. Als Verfahren zur Synchronisation soll die Syn1588-Technologie zum Einsatz kommen, welche gegenüber dem IEEE 1588-Standard in wichtigen Punkten verbessert und erweitert wurde. Die Syn1588-Technologie wurde an der TU-Wien in Zusammenarbeit mehrerer Institute sowie der Akademie der Wissenschaften entwickelt und ermöglicht eine Uhrensynchronisation mit einer Genauigkeit im Bereich von 1 ns. Dieser Netzwerkadapter soll in Form einer Hardwarebeschreibungssprache wie Verilog oder VHDL entwickelt werden, wobei für die Anbindung an das Medium ein Standard Physical Layer Chip verwendet werden soll. Das Design soll klar strukturiert sein und günstigerweise aus bereits getesteten Funktionskomponenten, so genannten IP-Cores aufgebaut sein.

Die Verifikation des erstellten Designs soll durch eine Simulation erfolgen. Für diesen Zweck ist eine Testbench zu erstellen.

2.2 Anforderungen

Folgende technische Anforderungen werden an das Design gestellt:

- Die PCI-Bridge muss im Master/Slave-Modus betrieben werden, um es der Netzwerkkarte zu ermöglichen, selbst Transaktionen am PCI-Bus starten zu können.
- Die PCI-Bridge ist mit einer Breite des Adress/Datenbusses von 32 Bit zu verwenden bei einer maximalen Taktfrequenz von 33 MHz.
- Der verwendete Ethernet MAC muss jeweils den Betriebsmodus mit 10 MBit/s als auch mit 100 MBit/s unterstützen und hat dem IEEE 802.3 Standard zu entsprechen.
- Dem Ethernet MAC muss es möglich sein, selbst Transaktionen am PCI-Bus auslösen zu können.

- Sowohl Ethernet MAC als auch der Syn1588-Core müssen Interrupts am PCI-Bus auslösen können.
- Die Register des MACs und des Syn1588-Core müssen über den PCI-Bus gelesen und beschrieben werden können.
- Für die PCI-Bridge und den den Ethernet MAC sollen fertige getestete Komponenten verwendet werden.
- Die Wahl der Verbindungsstruktur zum Zusammenschluss der Komponenten ist frei.
- Sollte ein anderes Interface zur chip-internen Verbindung als AHB gewählt werden, so ist eine Umsetzung auf ein AHB-Interface für den Syn1588-Core erforderlich.
- Die Entwicklung hat in der Hardwarebeschreibungssprache VHDL zu erfolgen.
- Für Überprüfung der Funktion ist eine Testbench zu erstellen.
- Das erstellte Design soll in einer Hardware getestet werden.

3 Uhrensynchronisation

Dieses Kapitel gibt einen Einblick in das Thema Uhren und deren Synchronisation und legt die Gründe dar, warum für genaue Uhrensynchronisation eine Unterstützung durch spezielle Uhrensynchronisationshardware unabdingbar ist. Weiters soll das seit mehreren Jahren betriebene Forschungsprojekt Syn1588 vorgestellt werden, welches eine hochgenaue fehlertolerante Uhrensynchronisation in verteilten Systemen ermöglicht.

3.1 Grundlagen der Zeitmessung und Uhrensynchronisation

Ein verteiltes System besteht allgemein aus einer Menge von Netzwerkknoten, die über einen Kommunikationskanal Nachrichten austauschen. Das Vorhandensein einer gemeinsamen Zeitbasis ist für den Betrieb eines verteilten Systems vorteilhaft, für viele verteilte Algorithmen ist es eine Voraussetzung für deren korrekte Funktionsweise. Ein globaler einheitlicher Zeitbegriff ermöglicht es, Abläufe koordiniert auszulösen oder zu erfassen. Algorithmen, die den Abgleich von divergierend laufenden Uhren in einem Netzwerk durchführen, werden *Uhrensynchronisationsalgorithmen* genannt.

Der heutige Zeitmaßstab, die Sekunde, ist über die Anzahl von Schwingungsperioden einer elektromagnetischen Welle definiert, die durch ein Cäsium 133-Atom emittiert wird. Beim Übergang zwischen zwei Hyperfeinstrukturniveaus strahlt Cäsium eine elektromagnetische Welle aus, deren Perioden gezählt werden. 9192631770 Perioden entsprechen einer Sekunde. Weltweit über 50 Labors betreiben rund 300 Atomuhren und teilen in regelmäßigen Abständen die Anzahl der Übergänge ihrer Uhren dem BIPM (Bureau international des poids et mesures) mit, woraus durch Mittelung die internationale Atomzeit TAI (Temps Atomique International) gebildet wird. TAI als Zeitmaßstab gibt die Anzahl der Sekunden an, die seit dem 1. Januar 1958 vergangen sind [5] [24].

Vom technischen Standpunkt ist TAI makellos, jedoch entsprechen die 86400 Sekunden eines Tages nicht genau einem mittleren Sonnentag. Die Gründe hierfür liegen in der sich minimal ändernden Rotationsgeschwindigkeit der Erde und der damit verbundenen Verlängerung des Tages. Das BIPM (früher BIH) löste dieses Problem durch Einführung von Schaltsekunden, und führte damit die universelle koordinierte Zeit, UTC (Coordinated Universal Time), ein. Die koordinierte Zeit UTC stellt die Grundlage der Zeitmessung im täglichen Leben dar, wird aber in gleichem Maße für die technische Zeitmessung wie z. B. in der Telekommunikation oder Astronomie verwendet. Die Dauer einer Sekunde ist bei UTC und TAI exakt gleich, jedoch werden zwischen TAI und UTC bei Bedarf Schaltsekunden eingefügt.

Die meisten UTC-Tage haben somit 86400 Sekunden, sodass jede Minute 60 Sekunden hat. An gewissen Tagen werden jedoch Schaltsekunden am Ende des Tages eingefügt oder entfernt, sodass der Tag dann 86399 oder 86401 Sekunden hat. Insgesamt wurden bis zum heutigen Zeitpunkt 33 positive Schaltsekunden gegenüber TAI eingefügt. Eine negative Schaltsekunde ist zwar nach UTC prinzipiell möglich, aufgrund der abnehmenden Rotationsgeschwindigkeit der Erde aber eher unwahrscheinlich [24].

Elektronische Geräte besitzen Uhren in Form einer Schaltung, die auf den Schwingungseigenschaften eines Quarz-Kristalls basieren. Wird an einen Quarz-Kristall über eine Oszillatorschaltung eine Spannung angelegt, so schwingt dieser mit einer definierten Frequenz, welche vom Typ des Quarzes abhängig ist. Der Ausgang des Oszillators ist an einen Zähler gekoppelt, welcher bei jeder Schwingung des Quarzes seinen Wert um eins inkrementiert. Der Zählerstand zeigt die Anzahl der Schwingungen des Quarzes, woraus die Zeit der Uhr ermittelt werden kann. In weiterer Folge soll das diskrete Fortschreiten einer Uhr vernachlässigt werden und von einer kontinuierlich laufenden Uhr ausgegangen werden.

3.1.1 Abweichung nichtidealer Uhren

Im Idealfall würden Knoten mit einer idealen Uhr für immer über dieselbe Zeit verfügen, wenn ihre Uhren nur einmal synchronisiert worden wären. Reale Uhren weisen jedoch immer eine kleine Abweichung (Drift) auf, sodass die Schwingungsfrequenz etwas von der nominellen Frequenz differiert. Dies bedeutet, dass die gemeinsame Zeitbasis mit dem Fortschreiten der Zeit immer stärker differiert. Diese Unterschiede werden als Uhr-Asymmetrie bezeichnet [24]. Die Konsequenz davon ist, dass anhand von den Werten der unterschiedlich laufenden Uhren keine Aussage getroffen werden kann, welches von zwei Ereignissen vorher und welches nachher stattgefunden hat. Diese Abweichungen liegen in der Größenordnung von 10^{-5} Sekunden je Sekunde, sodass eine Abweichung von der realen Uhr von knapp 1 Sekunde je Tag zustande kommt. Zudem ist die Höhe der Driftrate nicht konstant und ändert sich durch äußere Einflüsse wie Temperatur oder Alterung.

Aus historischer Sicht ist die Driftrate heutiger Quarze von 10^{-5} Sekunden je Sekunde keine besondere Ingenieursleistung. Bereits im 18. Jh. stellte der englische Uhrmacher John Harrison Uhren mit ähnlicher Genauigkeit her, um in der Schifffahrt die exakte Bestimmung des Längengrades zu ermöglichen. Gemäß der königlichen Ausschreibung durfte eine Uhr bis zu 2 Sekunden je Tag von der realen Zeit abweichen. Tatsächlich konnten seine Uhren diese obere und untere Schranke stark unterbieten, wobei seine Uhr *H.4* über eine 6 wöchige Reise nach Jamaika nur 5,1 s gegenüber der realen Zeit verlor [2].

In der Literatur wird die Abweichung wie folgt definiert: Es sei eine Anzahl von Uhren p gegeben, wobei jede Uhr zur realen Zeit t die Zeit $C_p(t)$ hat. Idealerweise sollte $C_p(t) = t$ gelten d.h. die Zeitableitung von $C_p(t)$ sollte gleich 1 sein. Für eine schnelle Uhr gilt daher $dC/dt > 1$ und für eine langsame Uhr $dC/dt < 1$. Die Abweichung vom einer idealen Uhr kann man mit der Konstante ρ beschreiben, welche als Formel in Gleichung 3.1 und grafisch in Abbildung 3.1 dargestellt ist [24].

$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho \tag{3.1}$$

Ist die Abweichung innerhalb der gesetzten Schranke ρ, so arbeitet die Uhr innerhalb ihrer Spezifikation. Weichen zwei Uhren in entgegengesetzter Richtung von der realen Zeit ab, so beträgt die größtmögliche Abweichung nach einer Zeit Δt seit der Synchronisation $2\rho\Delta t$. Anders ausgedrückt bedeutet das, wenn garantiert sein muss, dass zwei Uhren nicht um mehr als Δt von einander abweichen dürfen, so müssen diese alle $\Delta t / 2\rho$ Sekunden synchronisiert werden.

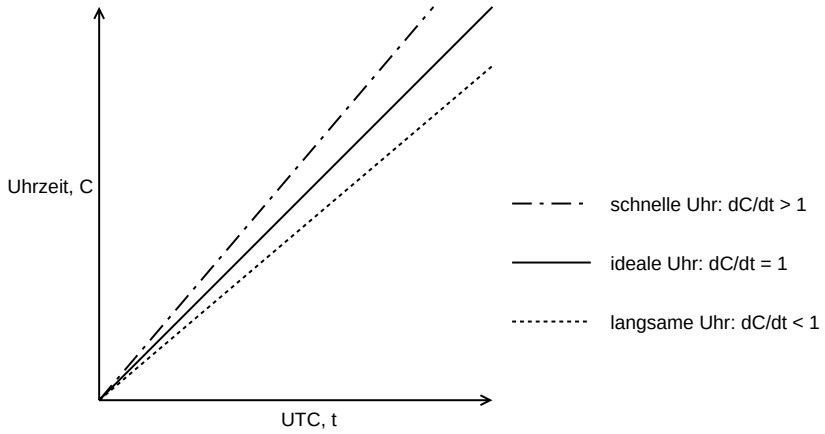

Abbildung 3.1: Driftrate von Uhren

Algorithmen zur Uhrensynchronisation haben eine Reihe von Schwierigkeiten zu bewältigen: Neben der Driftrate der Uhr, kommt es zu unterschiedlichen Verzögerungen bei der Übertragung zwischen den Knoten bedingt durch die Netzauslastung und dadurch, dass verschiedene Nachrichten je nach der Topologie des Netzes über unterschiedliche Netzwerkpfade zugestellt werden können. Eine weitere Herausforderung ist es fehlerhafte Uhren im Netzwerk eindeutig erkennen zu können.

3.1.2 Genauigkeit und Präzision von Uhrensynchronisationsalgorithmen

Die Qualität einer Synchronisationsmethode kann anhand der Präzision (*Clock Precision*) und der Genauigkeit (*Clock Accuracy*) beurteilt werden. Die Präzision π beschreibt die maximale Abweichung zweier Uhren p und q eines Uhrenensembles zu jedem Zeitpunkt t. Diese *interne* Uhrensynchronisation erlaubt es verteilte Ereignisse nach deren Auftreten zu sortieren, ein Bezug zur UTC-Zeit besteht im Allgemeinen jedoch nicht. Formel 3.2 beschreibt dies.

$$\left|C_p(t) - C_q(t)\right| \leq \pi \tag{3.2}$$

Die Qualität der *externen* Synchronisation zu einer Referenzzeit lässt sich durch die Genauigkeit α beschreiben. Sie gibt eine Schranke an, wie weit jede Uhr p zu jedem Zeitpunkt von der externen

Referenzzeit t (z. B. UTC) abweichen darf:

$$\left|C_p(t) - t\right| \leq \alpha_p \tag{3.3}$$

Die externe Synchronisation wird vor allem dann benötigt, wenn die Daten mehrerer synchronisierter Netzwerke gemeinsam verarbeitet werden sollen, da dies einen gemeinsamen Bezugspunkt erfordert [1].

Die Präzision π kann eines Uhrensynchronisationsalgorithmus für den *worst-case* wie folgt modelliert werden [13]:

$$\pi = c_1\epsilon + c_2 P\rho + c_3 G + c_4 u + c_5 G_s \tag{3.4}$$

c_1 bis c_n sind Gewichtsfaktoren, die abhängig vom verwendeten Algorithmus sind.

- ϵ gibt die Unsicherheit der Übertragungszeit beim Auslesen der Uhrzeit an.
- $P\rho$ beschreibt die Driftrate der Uhr zwischen den Synchronisationsperioden, wobei P die Periode angibt und ρ die Driftrate des Oszillators beschreibt.
- G bezeichnet die Granularität beim Auslesen der Uhr. Da die Uhrzeit in einer realen Uhr nicht kontinuierlich fortschreitet, sondern von einem Takt abhängig ist, kann eine Korrektur auch nicht beliebig fein erfolgen.
- u beschreibt die Unsicherheit beim Nachstellrate der Uhr infolge der Granularität.
- G_s bezeichnet die Granularität beim Schreiben der Uhr aufgrund einer Anpassung.

Eng verbunden mit der Genauigkeit und Präzision eines Uhrensynchronisationsalgorithmus ist die Festlegung, ob die Uhrensynchronisation rein in Software erfolgt oder mittels Hardwareunterstützung realisiert wird. Softwarebasierte Algorithmen verwenden standardisierte Netzwerkkomponenten um mittels Nachrichten in einem paketbasierten Netz eine Synchronisation durchzuführen. Typisch hierfür ist z. B. das *Network Time Protocol* (NTP), welches vor allem im Internet zum Einsatz kommt [17]. Hardwarebasierte Algorithmen verwenden eine spezielle Hardware in jedem Knoten und erreichen dadurch eine genauere und bessere Synchronisation.

Die erreichbare Genauigkeit lässt sich anhand des OSI (Open Systems Interconnection)- Referenzmodells veranschaulichen. Das OSI-Modell trennt ein Netzwerksystem in sieben Schichten auf, wobei jede Schicht nur mit ihrer darunter liegenden Schicht (ausgenommen der Bitübertragungsschicht) interagiert und für die darüber liegende Schicht gewisse Dienste und Services anbietet. Abbildung 3.2 zeigt die sieben Schichten des OSI-Modells und die Kommunikation eines Senders mit einem Empfänger.

Uhrensynchronisationsalgorithmen, die einen reinen Softwareansatz verfolgen wie z. B. das NTP-Protokoll verschicken die NTP-Synchronisationspakete über den verbindungslosen Dienst UDP (User Datagramm Protocol) über das Netz. Dieses Datagramm wird in den darunter liegenden Schichten verarbeitet, indem an das Datagramm zusätzliche Informationen an dieses vorne und hinten angehängt werden; bildlich wird die Information in jeder Schicht neu verpackt. Beim Empfänger wird die Information in derselben Schicht wieder entpackt, sodass man von horizontalem

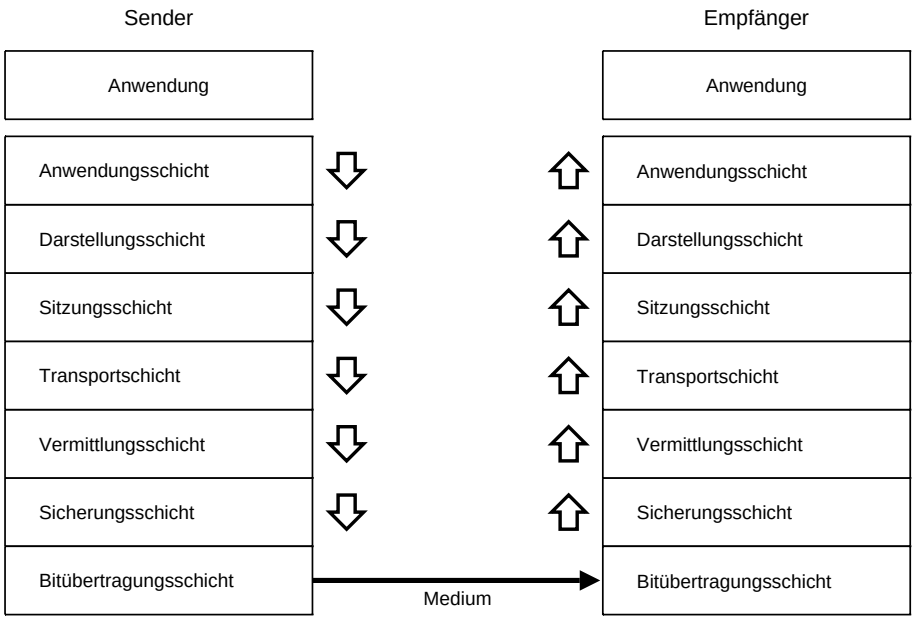

Abbildung 3.2: OSI-Referenzmodell

Datenfluss sprechen kann. Das Paket selbst durchläuft die Schichten jedoch vertikal und erfährt in jeder Schicht eine gewisse Verzögerung.

Wenn nun ein Client beim Zeitserver die aktuelle Uhrzeit anfordert, so wird der Server möglichst rasch antworten. Dennoch kommt es neben der rein physikalischen Übertragungszeit bedingt durch die Ausbreitungsgeschwindigkeit des Signals zu weiteren Verzögerungen. Im Allgemeinen kann ein Netzwerk aus beliebig viele Standardnetzwerkkomponenten wie Hubs, Switches oder Router bestehen, die nach dem Store & Forward-Prinzip arbeiten und in Abhängigkeit des Verkehrs im Netzwerk ihre Pakete mit unterschiedlicher Verzögerung weiterleiten können. In CSMA/CD-Systemen kann es zu nicht berechenbaren Verzögerungen kommen, sodass Synchronisationspakete verspätet ankommen können oder möglicherweise auch ganz verschwinden. CSMA/CD-Systeme sind aus diesem Grund für genaue Synchronisationsaufgaben im Allgemeinen eher ungeeignet. Die Endpunkt-zu-Endpunkt-Signallaufzeit weist aus den genannten Gründen einen Fehler durch eine nicht abschätzbare Zeit (Jitter) auf.

Auch in den Endpunkten selbst, den Netzwerkknoten, kommt es durch die Abarbeitung des Protokollstacks, welcher sich in die verschiedenen Schichten im OSI-Modell aufgliedern lässt, zu einer weiteren Verzögerung. Diese Verzögerung ist jedoch nicht deterministisch und abhängig von der Resourcenauslastung des Netzwerkknotens, sodass eine Abschätzung praktisch unmöglich ist. Die durch die Abwicklung der Uhrensynchronisation in Software hervorgerufene Unsicherheit bewegt sich im Bereich von rund 1 ms.

Der Algorithmus von *Cristian* [6] schlägt eine Messung der Verzögerung zwischen dem Absetzen der Zeitanforderung und dem Erhalt des Synchronisationspakets vom Zeitserver vor. Eine weitere Verbesserung ergibt sich durch das Aufzeichnen einer Messreihe und Bildung eines Mittelwerts,

wobei Messwerte, die einen gewissen Schwellenwert überschreiten, nicht zur Berechnung herangezogen werden. All dies bedingt, dass vom Absetzen eines Datagramms bis zum tatsächlichen Empfang eine für das System nicht einschätzbare Zeit vergeht, die die Genauigkeit maßgeblich einschränkt.

Hardwarebasierte Algorithmen verwenden im Gegensatz dazu eine spezielle Hardware in jedem Knoten, was eine genauere und bessere Synchronisation erlaubt. Beispiele hierfür sind die Verwendung von GPS-Empfängern. Ein weiterer Ansatz eines hardwarebasierten Algorithmus ist die Verwendung eines dedizierten Netzwerks zur Uhrensynchronisation, wobei alle Knoten im Netzwerk mittels eines PLL (Phase Locked Loop) ähnlichen Schemas auf bis zu 1 ns genau synchronisiert werden können. Die Verwendung eines dedizierten Netzes erlaubt jedoch nur den lokalen Einsatz in einem Rechnernetzwerk [23].

In einem paketbasierten Netzwerk lässt sich die Unsicherheit der Durchlaufzeit des Protokollstacks verbessern, indem man spezielle Hardware verwendet, die über eine eigene Uhr verfügt und das Eintreffen der Synchronisationspakete möglich nahe der Bitübertragungsschicht misst. Man spricht in diesem Zusammenhang von *Hardware Timestamping*. Darunter ist zu verstehen, dass das Eintreffen eines Synchronisationspakets an einer möglichst niedrigen OSI-Schicht aufgezeichnet wird z. B. durch Scannen der Schnittstelle zwischen der Bitübertragungsschicht und der Sicherungsschicht. Sobald der Beginn eines Frames erkannt wird, wird der Zeitstempel in einem Register abgelegt. Beim Versand eines Synchronisationspakets kann das Frame verändert und die aktuelle Uhrzeit in das Frame eingetragen werden bzw. wird die tatsächliche Sendezeit durch ein Follow Up-Paket übertragen. Mithilfe dieses Mechanismus ist es möglich, dass die unterschiedlichen Durchlaufzeiten der höheren Protokollschichten in den Knoten selbst die Uhrensynchronisation nicht beeinflussen.

Da aber auch die Standardnetzwerkkomponenten wie Hubs, Switches und Router die Synchronisationspakete für eine ungewisse Zeit verzögern, muss auch in diesen Komponenten die Verweildauer gemessen werden. Dies kann ebenfalls mit dem System der Zeitstempelung beim Senden und Empfangen jedes Synchronisationspakets realisiert sein und somit die Verweildauer exakt bestimmt werden kann. Die Genauigkeit unterschiedlicher Systeme ist grob in Abbildung 3.3 vgl. [4] dargestellt.

3.2 Systeme zur Uhrensynchronisation

Dieser Abschnitt stellt unterschiedliche Systeme zur Uhrensynchronisation vor und zeigt deren Stärken und Schwächen auf.

3.2.1 Network Time Protocol

Das Network Time Protocol (NTP) basiert auf einem hierarchischen System, dessen Ebenen *Stratum* genannt werden. Alle Zeitserver auf der Ebene Statum 1 sind mit einer genauen Uhr in der Ebene Stratum 0 verbunden. Die Zeitserver von Stratum 2 beziehen deren Zeit von den Servern aus

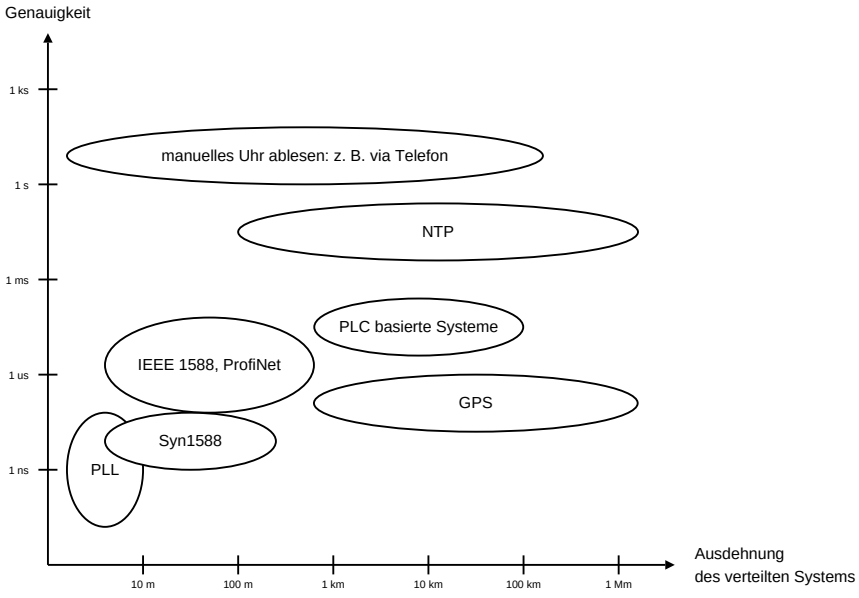

Abbildung 3.3: Systeme zur Uhrensynchronisation

Stratum 1 bzw. synchronisieren ihre Uhr mit anderen Servern auf der gleichen Ebene. Der Pfad der Synchronisation zieht sich somit von der untersten Ebene Stratum n bis zu Stratum 0, welches aus Atomuhren oder GPS-Empfängern besteht. Die günstige Wahl der Anzahl der Ebenen ist somit der Hauptparameter für die Qualität von NTP. Der hierarchische Aufbau dieses Protokolls ist in Abbildung 3.4 dargestellt. Das NTP-Protokoll kommt von Stratum 1 bis Stratum n zur Anwendung. Die Uhren in Stratum 0 sind direkt z. B. über eine serielle Schnittstelle mit den Rechnern in Stratum 1 verbunden.

NTP-Clients senden und empfangen NTP-Nachrichten über die OSI Schicht 4, das UDP-Protokoll. NTP misst die Antwortzeit des Servers, welche sich aus der Übertragungszeit und der Abarbeitungszeit des Protokollschichten zusammensetzt. Mithilfe der gefilterten Antwortzeit und der Information des Zeitservers wird die Uhr synchronisiert. Die erreichte Genauigkeit beträgt rund 20 ms, in lokalen Netzen 1 ms. [25]

3.2.2 Global Positioning System

Ein satellitengestütztes Navigationssystem wie das Global Positioning System (GPS) basiert auf Satelliten, die stets ihre Position und die genaue Zeit ihrer internen Uhr ausstrahlen. Anhand der unterschiedlichen Signallaufzeit und der bekannten Position der Satelliten, lässt sich die Position des GPS-Empfängers auf der Erde bestimmen. Es ist daher jedes Gerät mit einem GPS-Empfänger auszurüsten, und somit kann die genaue Uhrzeit zur Verfügung gestellt werden. Als Nachteil sind die Kosten des GPS-Empfängers zu sehen als auch die Notwendigkeit einer nahezu direkten Sichtverbindung zu den Satelliten. Aufgrund des verwendeten Frequenzbereich um 1572,42 MHz (L1-Band)

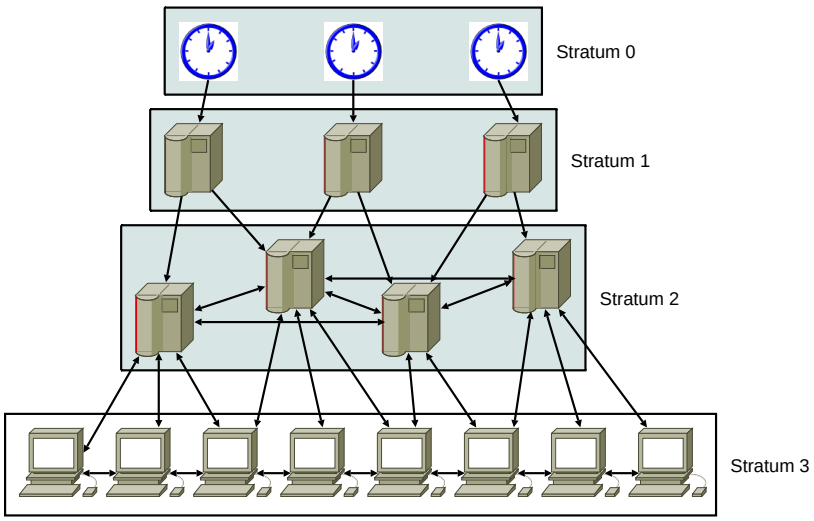

Abbildung 3.4: Hierarchischer Aufbau eines NTP-Systems

und der geringen Sendeleistung der Satelliten ist eine quasi-optische Sicht zum Satelliten erforderlich. Die erreichte Genauigkeit liegt bei etwa 100 ns.

3.2.3 Precision Time Protocol

Der im Jahr 2002 verabschiedete IEEE 1588-Standard [14] definiert mit dem Precision Time Protocol (PTP) ein Verfahren um räumlich verteilte Echtzeituhren in einem paketbasierten Netzwerk zu synchronisieren. Es ist nicht an eine gewisse Netzwerktechnologie gebunden, sondern beschreibt ein Protokoll zur Uhrensynchronisation, welches sowohl in Software aber auch in Hardware implementiert sein kann. Das zugrunde liegende Netzwerk kann Ethernet sein, im Allgemeinen aber jedes paketbasierte Netz.

Das PTP-Protokoll definiert im Wesentlichen Folgendes:

- die Auslegung und das Verhalten der Uhr
- ein Verfahren zur Segmentierung des PTP-Netzwerks
- ein nachrichtenbasiertes Protokoll zur Synchronisation von vernetzten Uhren
- einen Best-Master-Clock-Algorithmus zur Bestimmung und Kontrolle der Master-Uhr
- ein PTP-Management-Protokoll

Die Synchronisation läuft nach einem Master/Slave-Prinzip ab. Wenn alle Busteilnehmer angeschlossen sind, wird im ersten Schritt der beste Master anhand des *Best-Master-Clock*-Algorithmus ausgewählt. Jede PTP-Uhr muss ihre Eigenschaften allen Busteilnehmern in einem spezifizierten Format offen legen, sodass die Teilnehmer entscheiden können, ob diese als Master- oder Slave-Clock arbeiten wollen. Diese Entscheidung wird nicht durch Wahlalgorithmen verhandelt und es können auch jederzeit weitere Busteilnehmer beitreten. Befinden sich Switches oder Router innerhalb des Netzes, so kann der kann die Übertragungszeit zu stark variieren, sodass PTP so genannte

boundary clocks definiert, um dieses Problem zu umgehen. Diese boundary clocks verfügen über mehr als einen PTP-Port und stellen ihre Uhr für beide Seiten des Netzes zur Verfügung. Somit können Netzwerkelemente umgehen werden. Die Verwendung von boundary clocks ist für eine Synchronisation über mehrere Subnetze hinweg sogar verpflichtend. Master, die für mehrere Subnetze die Funktion des Masters übernehmen, werden *Grand-Master* genannt.

Der Master schickt zuerst ein multicast Sync-Paket an die Slaves, welches die lokale Uhrzeit enthält. Die tatsächliche Sendezeit wird, sofern eine Hardwarezeitstempelung verwendet wird, nahe an der physikalischen Schicht aufgezeichnet und an den Softwaretreiber übermittelt. Der Master sendet daraufhin ein Follow Up-Paket, welches den Zeitstempel des vorangegangen Sync-Pakets enthält. Der Empfänger kann durch die Empfangszeit des Sync-Pakets und dem Inhalt des Follow Up-Pakets die Zeitdifferenz zwischen seiner Uhr und der Master-Uhr bestimmen. Dies inkludiert jedoch auch die Leitungsverzögerung zwischen Master und Slave. Um diese Zeit zu bestimmen, versendet der Slave ein Delay Request-Paket an den Master. Dieser antwortet dem Slave durch ein Delay Response-Paket, in welchem die lokale Empfangszeit des Delay Request-Pakets enthalten ist. Ausgehend von einer symmetrischen Verzögerung in beide Richtungen, lässt sich so die Leitungsverzögerung bestimmen. Abbildung 3.5 zeigt das Synchronisationsschema.

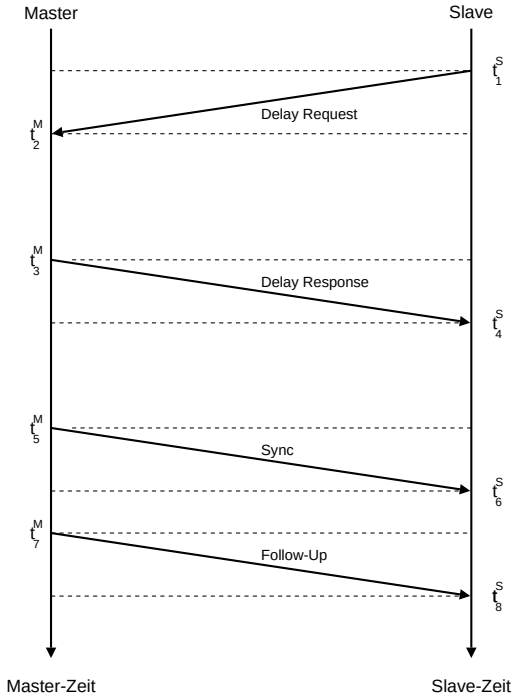

Abbildung 3.5: Versand von Synchronisationspaketen und Delaymessung gemäß PTP

Das Ausmessen der Laufzeitverzögerung erfolgt in Intervallen von 1 bis 64 s. Der Synchronisationsvorgang durch Versenden der Sync-Pakete erfolgt alle 2 bis 8 s, weil die Driftrate der lokalen Uhren im Allgemeinen einen größeren Einfluss je Zeitintervall verursacht als die sich nur langsam ändernde Leitungsverzögerung. Die erreichte Genauigkeit einer Uhrensynchronisation hängt vor allem davon ab, ob PTP unter Verwendung von Hardware-Timestamping implementiert ist. In diesem

Fall beträgt die erreichte Genauigkeit rund 100 ns. Der IEEE-1588 Standard kann in vielen Bereichen eingesetzt werden, findet aber vor allem eine Anwendung in der Automatisierungstechnik, wo eine durchgängige Kommunikationslösung auf Basis von Ethernet vorhandene echtzeitfähige Feldbussysteme ersetzen soll [15].

3.2.4 Syn1588-Technologie

Die an der TU-Wien gemeinsam mit der Akademie der Wissenschaften entwickelte Syn1588 (Synchronised IEEE 1588)-Technologie beschreibt ein nicht ausschließlich auf Ethernet beschränktes System, welches in der Lage ist, in einem verteilten Netzwerk Uhren mit einer Genauigkeit bis zu 1 ns zu synchronisieren unter Verwendung des Precision Time Protocols. Im Vergleich zum IEEE1588-Standard verfügt Syn1588 über mehrere Verbesserungen im Hinblick auf Genauigkeit und Fehlertoleranz.

Syn1588 folgt im Gegensatz zum Master/Slave-orientierten IEEE1588-Protokoll über einen demokratischen Ansatz. Es gibt keinen ausgezeichneten Master, der die Slaves mit der Uhrzeit versorgt, sondern einen Austausch von Synchronisationspaketen von jedem Knoten mit jedem Knoten. Dieses Paradigma erhöht die Fehlertoleranz, sodass aus $2N + 1$ Uhren N fehlerhafte Knoten erkannt werden können. Im Gegenzug steigt aber im Vergleich zu einem zentralen Ansatz die Anzahl der Nachrichten bei einer großen Anzahl von Knoten n stark an: In einem zentralen System verschickt der Master $n - 1$ Nachrichten je Synchronisationsrunde. Bei einem demokratischen System verschickt jedoch jeder Knoten $n - 1$ Nachrichten je Runde, sodass die Effizienz im Vergleich zu einem zentralen Ansatz mit $\frac{1}{n}$ sinkt [21].

Fällt in einem zentralen System ein Master aus, so werden alle Uhren in ihrer Präzision eingeschränkt, da bis zur Wahl eines neuen Masters alle Uhren voneinander weg driften. In einem demokratischen System wie Syn1588 kommt es zu keinen Einschränkungen, was die Präzision der Uhrzeit zwischen den Knoten selbst betrifft. Die interne Uhrensynchronisation bleibt in jeden Fall erhalten. Die externe Uhrensynchronisation, welche sich auf die UTC-Zeit bezieht, kann eventuell durch den Ausfall eines Knotens mit GPS-Empfänger gefährdet sein. Im schlimmsten Fall können sich alle Uhren gemeinsam vom externen Zeitnormal wegbewegen, untereinander bleiben die Uhren jedoch hochgenau synchronisiert.

Der Synchronisationsvorgang läuft rundenbasiert ab:

- Jeder Knoten sendet jedem Knoten seine Uhrzeit gemeinsam mit einem Genauigkeitsintervall.
- Es folgt die Wartephase, in der die Nachrichten aller Knoten gesammelt werden.
- Die Berechnung der neuen Uhrzeit und des Genauigkeitsintervalls erfolgt anhand einer fehlertoleranten Schnittfunktion.
- Die Re-Synchronisation erfolgt anhand der berechneten Zeit.
- Die restliche Zeit läuft die Uhr frei.

Syn1588 verwendet eine Hardware-basierte Zeitstempelung. Da es bei einer Software-basierten Zeitstempelung zwangsweise durch nicht bekannte Verzögerungen des Protokollstacks zu nicht deterministischen Verzögerungen kommt, muss die Zeiterfassung nahe an der Bitübertragungsschicht erfolgen. Die Zeitstempelung erfolgt bei Syn1588 am Interface zwischen Bitübertragungsschicht und der Sicherungsschicht, welches im Fall von 100 MBit/s Ethernet das Media Independent Interface (MII) ist. Die Daten des MII werden durch eine spezielle Schaltung geleitet, welche die Frames nach dem Start Frame Delimiters (SFD) durchsucht. Für jedes gesendete und empfangene Synchronisationspaket wird die genaue Sende- bzw. Empfangszeit in einem Register abgelegt.

Die Uhr eines Syn1588-Knotens baut nicht auf einer Zählerstruktur auf, bei der der Zähler bei jeder steigenden Taktflanke um eins vorrückt, sondern auf einer *Adder Based Clock* (ABC). Die ABC basiert auf einem 96 Bit breiten Zähler, der durch ein hochauflösendes Register mit einem programmierbaren Wert erhöht werden kann. Dies erlaubt es sowohl die Oszillatorfrequenz in einem weiten Bereich wählen zu können, aber auch die Uhr langsamer und schneller laufen zu lassen. Die ABC ermöglicht es daher, neben dem einfachen Setzen der Uhrzeit (*state synchronization*), die lokale Uhr stetig an die Soll-Uhrzeit heranzuführen, indem die Geschwindigkeit der Uhr verändert wird. Dies wird als *rate synchronization* bezeichnet.

Syn1588 verfolgt den Ansatz der intervallbasierten Synchronisation, welche so genannte Genauigkeitsintervalle definiert [21]. Neben der Haupt-ABC kommen noch zwei weitere Adder Based Clocks zum Einsatz, jedoch mit einer Breite von 64 Bit. Diese beiden ABC legen ein oberes und unteres Intervall fest zwischen denen sich die Haupt-ABC bewegen darf. Die Verwendung der Genauigkeitsintervalle in Verbindung mit einer *rate synchronization* ist in Abbildung 3.6 vgl. [9] dargestellt.

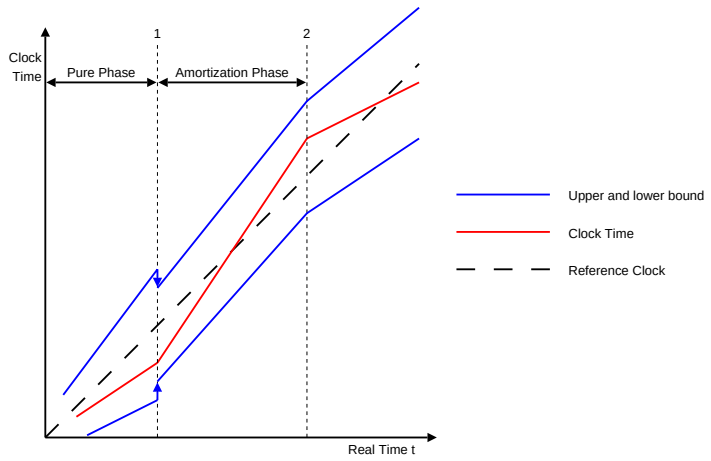

Abbildung 3.6: Intervallbasierte Synchronisation und *rate synchronization*

In der *Pure Phase* läuft die Uhr frei. Zum Zeitpunkt 1 wird die Uhr und das Genauigkeitsintervall aufgrund einer neuen Berechnung angepasst. Die Uhr selbst wird jedoch nicht auf den neuen Wert gesetzt wie dies bei einer *state synchronization* der Fall wäre, sondern es wird die Geschwindigkeit der Uhr angepasst um der Referenzzeit zu folgen. Man nennt dieses Anpassungsphase *Armortisierungsphase*. Zum Zeitpunkt 2 geht die Uhr wieder in die *Pure Phase* über. Dieses kontinuierliche

Nachziehen der Uhrzeit ist nur durch den Einsatz der ABC möglich, wobei eine vergleichbare progressive Heranführung der Uhrzeit an die Referenzzeit sonst nur durch teure OCXOs (Oven Controlled Crystal Oscillators) möglich ist [9].

Synchronisationspakete, welche auch CSP (Clock Synchronisation Packet) genannt werden, werden in periodischen Abständen zur Synchronisation verschickt. Jedes dieser Pakete beinhaltet im Wesentlichen folgende Zeiten:

- den Aussendezeitpunkt des Quellknotens
- die Verweildauer des Pakets in allen Verbindungsknoten
- den Empfangszeitpunkt im Zielknoten

Der zu erwartende Empfangszeitpunkt errechnet sich daher aus der Summe von Aussendezeitpunkt, der gesamten Verweildauer in den Verbindungsknoten sowie der physikalischen Übertragungszeit. Die Abweichung der Uhren ergibt sich somit aus der Differenz zwischen dem berechneten und dem tatsächlichen Empfangszeitpunkt. Anhand dieser Differenz können die Uhren durch Änderung ihrer Geschwindigkeit nachgestellt werden.

Dies bedeutet aber, dass für jedes Synchronisationspaket die Verweildauer in den Verbindungsknoten (z. B. Switches) bestimmt werden muss. Es bietet sich daher an, das Verfahren der Zeitstempelung an der MII-Schnittstelle auch für den Switch zu verwenden, um somit die Verweildauer Δt ausmessen zu können. Da Δt auch sofort in das CSP eingetragen werden soll, müssen die MII-Signale an geeigneter Stelle manipuliert werden, und auch die Prüfsumme am Ende jedes Frames korrigiert werden.

4 Auswahl von IP-Cores

Dieses Kapitel legt das Konzept einer Netzwerkkarte zur hochgenauen Uhrensynchronisation dar und vergleicht zwei unterschiedliche Implementierungsmöglichkeiten.

Für die Entwicklung komplexer Systeme für Mikrochips erweist es sich als günstig auf dem Konzept von IP-Cores (IP= Intellectual Property) aufzubauen. IP-Cores sind fertige und getestete Funktionseinheiten, die ohne deren interne Kenntnis zu einem Gesamtsystem zusammengeschlossen werden können. Voraussetzung hierfür ist jedoch ein einheitliches Interface.

4.1 Grundkonzept einer Netzwerkkarte

Eine Netzwerkkarte, Netzwerkadapter oder NIC (Network Interface Controller) bietet ein Interface zur Anbindung eines Computers an ein Netzwerk. Eine Netzwerkkarte stellt somit geeignete Funktionen und Dienste zur Verfügung um mit anderen Computern in einem Netzwerk Daten austauschen zu können. Nach außen hin verfügt eine Netzwerkkarte über zwei Schnittstellen: Einerseits über eine Schnittstelle zum Bussystem des Computers, anderseits über eine physikalische Anbindung an ein Netzwerk.

Die Funktion eines Ethernet-Netzwerkadapters für den PCI (Periphical Component Interconnect)-Bus kann in mehrere Funktionsblöcke unterteilt werden:

- PCI-Interface
- Ethernet MAC (Media Access Controller)
- PHY (Physical Layer Chip)
- Verbindungsstruktur

Das PCI-Interface erlaubt die Kommunikation der NIC zum Computer-internen-Bussystem, welches – im Fall von PCI – auf einem gemeinsam genutzten Datenbus (shared bus) aufbaut. Der PCI-Bus gilt als das Standard Bussystem für Einsteckkarten in den meisten heute gebräuchlichen Computern. Seinen Ursprung hatte PCI zu Beginn der 90er Jahre bei der Firma Intel, die mit dem PCI-Bus eine Standardmethode zur Verbindung von Mikrochips auf Boards entwickelten. Etwas später wurde die *PCI Special Interest Group* (PCI SIG) gegründet und PCI als Industriestandard verabschiedet. Gegenüber der ersten Spezifikation (Version 1.0) aus dem Jahr 1992 wurde der Standard in der Zwischenzeit um zahlreiche Features erweitert und auch die Datenbreite des gemeinsamen Adress-/Datenbusses von ursprünglich 32 auf 64 Bit erhöht und der Takt von ursprünglich 33 MHz

auf 66 MHz gesteigert. Die am meisten in Computern gebräuchlichste Variante des PCI-Busses verfügt über eine Datenbreite von 32 Bit und wird mit 33 Mhz getaktet. Daraus ergibt sich ein Bruttodatendurchsatz von bis zu 133 MB/s. [22]

Der Ethernet MAC sorgt zusammen mit dem PHY für eine korrekte Übertragung der Daten über das Medium. Der MAC regelt hauptsächlich den Zugriff auf das Medium, versendet und empfängt Datenrahmen (Frames) und überprüft diese anhand der Frame Check Sum (FCS). Der Datenaustausch zwischen MAC und PHY erfolgt bei der Ethernet-Variante 100Base-TX über das Media Independent Interface (MII). Wie die englische Bezeichnung bereits andeutet, handelt es sich hierbei um ein medienunabhängiges Interface d.h. eine Anpassung einer Netzwerkkarte an ein anderes Medium bedarf nur einer Änderung des PHYs. Das MII-Interface wird mit 25 MHz für eine Übertragungsrate von 100 MBit/s bzw. 2,5 MHz für 10 MBit/s getaktet. Der Datentransfer erfolgt über jeweils 4 Bit breite Datenleitungen in Empfangs- und Senderichtung sowie zusätzlichen Takt- und Signalisierungsleitungen.

Der PHY führt eine Codierung der über MII erhaltenen Daten durch und überträgt diese mittels Media Dependent Interface (MDI) auf das Medium. Die Codierung selbst ist abhängig vom Betriebsmodus (z. B. 100Base-TX) und im Ethernet-Standard spezifiziert. Zur Konfiguration und zum Auslesen von Statusinformationen enthält das MII auch eine serielle Schnittstelle, die dem MAC einen von den Datenleitungen unabhängigen Zugriff auf den PHY ermöglicht. Die Übertragung der Daten erfolgt bei Ethernet generell im Basisband, wobei als Übertragungsmedium entweder Twisted Pair Kabel oder Lichtwellenleiter in Verwendung sind.

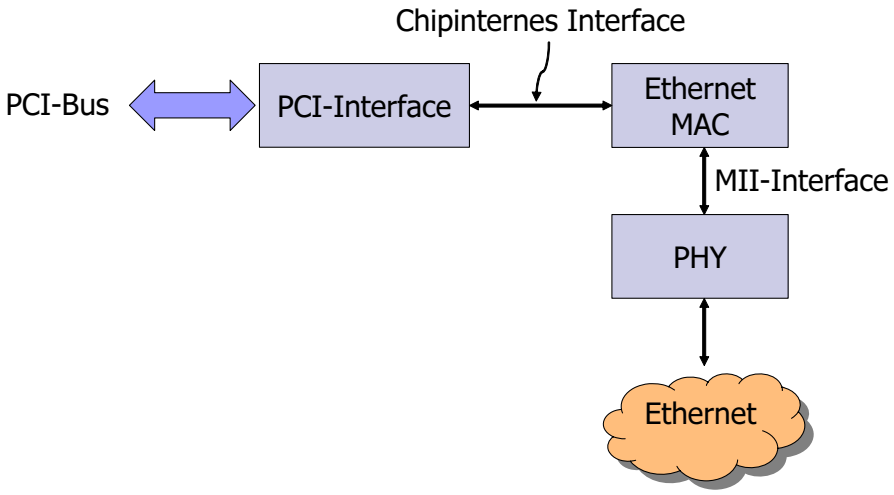

Abbildung 4.1: Aufbau einer Netzwerkkarte

Während das Interface zwischen Ethernet MAC und PHY durch das MII festgelegt ist, können zur Verbindung des Ethernet MACs mit dem PCI-Interface beliebige Verbindungsstrukturen eingesetzt werden. Es können sowohl standardisierte als nicht standardisierte Verbindungsstrukturen zum Einsatz kommen. Standardisierte Schnittstellen haben den Vorteil, dass aufgrund der höheren Verbreitung oftmals fertige Funktionsblöcke mit der benötigten Funktionalität zur Verfügung stehen. Die Integration mit bereits vorhandenem Code wird aufgrund der einheitlichen Schnittstelle erleich-

tert. Für gewisse Schnittstellen existieren auch fertige Generatoren, wodurch das fehlerträchtige Erstellen der Verbindungslogik (*glue logic*) entfällt. Dieses Grundmodell eines Netzwerkadapters ist grafisch in Abbildung 4.1 dargestellt.

Die Hardware der meisten heutigen Netzwerkkarten basiert aus Kostengründen auf einer Single-Chip-Lösung, bei welcher das Bus-Interface und der MAC in einem Chip integriert sind. Ergänzt wird das Layout durch den PHY und weiteren Komponenten wie Übertrager und Buchsen.

Prinzipiell kann man mit einer gewöhnlichen COTS (commercial off-the-shelf)-Netzwerkkarte eine softwarebasierte Uhrensynchronisation durchführen z. B. nach dem NTP-Protokoll. In einer derartigen Konfiguration kommt es durch den Einsatz eines Multitasking-Betriebssystems, eines von mehreren Busteilnehmern geteilten PCI-Bus und eines MACs mit Pufferspeicher unvermeidlich zu einer nicht deterministischen Verzögerung bei der Abwicklung des Protokollstacks beim Empfang und Senden eines Synchronisationspakets. Um die geforderte Genauigkeit erreichen zu können, wird der Mechanismus der Hardwarezeitstempelung benötigt, welcher in Form des Syn1588-Cores implementiert wird. Dieser IP-Core ermöglicht es mithilfe der Syn1588-Technologie eine Uhrensynchronisation mit einer Genauigkeit von bis zu 1 ns durchzuführen.

Die Integration des Syn1588-Core in das Konzept einer Standardnetzwerkkarte betrifft primär die Anbindung eines Uhrensynchronisations-Cores an die Verbindungslogik. Während zwei IP-Cores einfach mittels einer Punkt-zu-Punkt-Verbindung zusammen geschaltet werden können, wird bei den gängigen chipinternen Verbindungssystemen eine Verbindungslogik benötigt. Denn meist basieren diese On-Chip-Bussysteme aus Performancegründen nicht auf gemeinsamen sondern auf dedizierten Bussen. Dedizierte Busse haben den Vorteil, dass auf Tristate-Treiber verzichtet werden kann, welche einerseits nicht in allen Technologien verfügbar sind und andererseits hohe Kapazitäten mit sich bringen, die die erreichbare Taktfrequenz einschränken. Diese dedizierten Busse erlauben jedoch nur zwei Busteilnehmer. Sollen mehrere IP-Cores miteinander verbunden werden, so werden die Daten- und Adressleitungen jedes IP-Cores über eine zentrale Verbindungslogik miteinander verbunden. Sie steuert den Zugriff aller Busteilnehmer und kann in Form einer Multiplexerstruktur, auf Basis eines Kreuzschienenverteilers oder mittels einer ähnlichen Struktur realisiert sein. Der Syn1588-Core verfügt über ein AMBA-AHB-Interface, sodass beim Einsatz eines anderen Interfaces als AHB eine Umsetzung mittels einer Bridge auf AHB notwendig ist.

Neben der Anbindung des Syn1588-Cores an das chipinterne Interface muss noch die Timestampunit integriert werden. Diese besteht aus einem MII-Scanner, der die MII-Schnittstelle zwischen dem MAC und dem PHY abhört und anhand des Start Frame Limiters (SFDs) den Beginn von Frames erkennen kann. Sobald das Eintreffen oder der Versand oder Empfang eines Frames erkannt wird, wird dies dem Uhrensynchronisationsmodul signalisiert, sodass der Zeitstempel exakt bei der Übergabe der Daten an den PHY erstellt werden kann. Dieser Zeitstempel wird zusammen mit einer ID im Speicher des Syn1588-Cores abgelegt. Abbildung 4.2 zeigt die Erweiterung einer Netzwerkkarte zur Uhrensynchronisation.

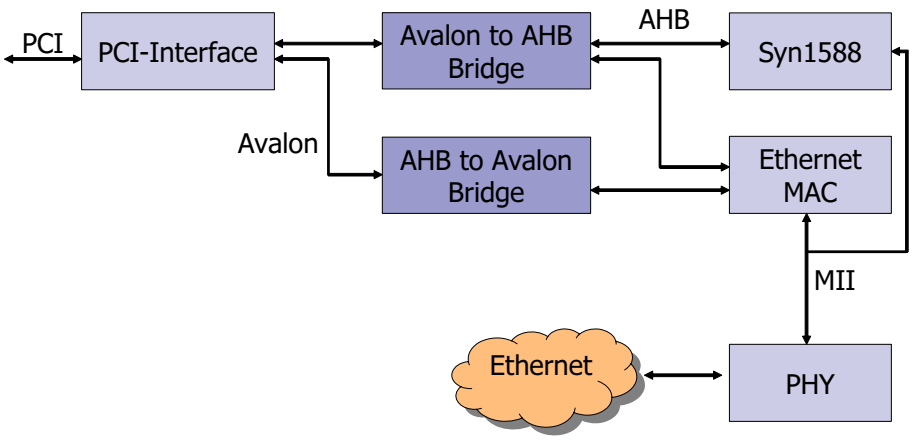

Abbildung 4.2: Aufbau einer Netzwerkkarte zur Uhrensynchronisation

4.2 NIC basierend auf IP-Cores der Firma Altera

Grundlage dieser Lösung sind die von der Chip-Firma Altera (www.altera.com) verfügbaren IP-Cores. So bietet Altera sowohl einen PCI-Core als auch einen Ethernet-MAC für ihre SOPC (system on a programmable chip)-Entwicklungsumgebung Quartus II V5.0 an. Der im Quartus II enthaltende SOPC-Builder ermöglicht es rasch Systeme zu generieren bestehend aus Prozessoren, Speicher und anderer Peripherie und automatisiert die Integration der einzelnen Hardwarekomponenten zu einem System. Die IP-Cores werden mittels eines graphischen Interfaces verbunden und parametriert. Nach abgeschlossener Konfiguration wird mittels eines Perl-Skripts die Verbindungslogik generiert sowie eine Testbench, die als Ausgangspunkt der Simulation dient. Sowohl die Ausgabe von VHDL- als auch Verilog-Dateien wird unterstützt.

Altera verwendet das proprietäre Avalon-Interface zusammen mit der Avalon Switching Fabric zur Verbindung der IP-Cores. Avalon ist ein memory mapped-Interface d.h. der Zugriff auf das Avalon-Interface der Komponenten erfolgt durch einen Schreib- oder Lesevorgang im Adressbereich der Komponente. Die verwendeten Komponenten müssen eine unterschiedliche Basisadresse aufweisen und die Speicherbereiche dürfen sich nicht überdecken.

Alle Komponenten müssen ein Avalon- oder AHB-Interface besitzen, damit diese mit Hilfe des SOPC-Builders zu einem Gesamtsystem zusammengefügt werden können. Die Unterstützung des AHB-Interfaces erfolgt über Bridges. Eine Umsetzung des Avalon-Interfaces auf AHB erfolgt durch die *AHB to Avalon Bridge*, eine Umsetzung von AHB auf Avalon durch die *Avalon to AHB Bridge*.

Die *AHB to Avalon Bridge* verfügt über einen AHB Slave Port sowie einem Avalon Master Port, auf dem das umgesetzte Avalon Signal zur Verbindung mit *Alteras Switching Fabric* bereitsteht. Vergleichbar besitzt die *Avalon to AHB Bridge* einen Avalon Slave Port, auf dem das Avalon-Signal entgegengenommen wird, in ein AHB-Signal umgesetzt wird und über den AHB Master Port ausgegeben wird, und somit für die Verbindung an den Slave-Port einer AHB-Komponente zur Verfügung steht. Ein wesentlicher Unterschied besteht jedoch: So können an eine Avalon to AHB Bridge

mehrere AHB-Slaves angeschlossen werden, an der AHB to Avalon Bridge jedoch nur jeweils ein AHB-Master.

Für die Avalon to AHB Bridge erstellt der SOPC-Builder die Decoderstruktur, welche die Auswahl des korrekten AHB-Slaves übernimmt. Der Dekoder kennt die zu jedem Slave zugeordneten Adressbereiche. Wird nun über die Bridge auf einen der Slaves zugegriffen, so setzt der Decoder das Selektionssignal HSEL für denjenigen Slave, auf dessen Adressbereich zugegriffen wird. Dieses Signal wird auch für den erstellten Lesemultiplexer benötigt, um die Lesedaten des betreffenden Slaves auf HRDATA zu legen.

Bei der AHB to Avalon Bridge wird keine vergleichbare Struktur erstellt. Um mehrere AHB-Master an einer Bridge betreiben zu können, müsste neben den Multiplexerstrukturen für Adress-, Kontroll- und Schreibdaten ein Arbiter zum Einsatz kommen. Dies würde jedoch bedeuten, dass für die Bridge selbst arbitriert werden müsste und an der Avalon Switching Fabric die Bridge selbst nochmal mit allen anderen Avalon-Cores arbitrieren müsste. Altera umgeht dies, indem es je AHB-Master eine eigene AHB to Avalon Bridge verwendet und übergibt damit die Aufgabe der Arbitrierung an den ohnehin vorhandenen Avalon-Arbiter. Abbildung 4.3 zeigt die beiden Bridges.

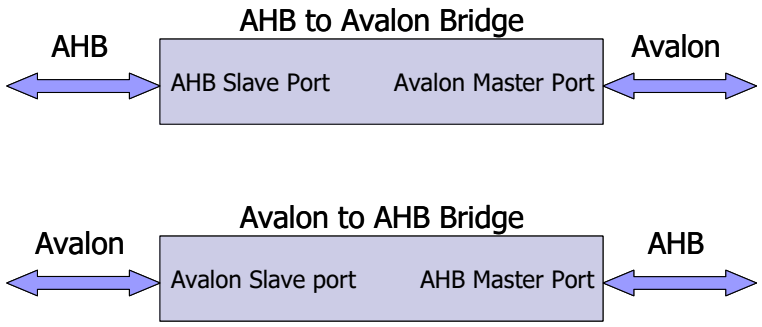

Abbildung 4.3: AHB to Avalon Bridge und Avalon to AHB Bridge

4.2.1 Ethernet MAC

In dieser Implementierung soll der von Altera angebotene Ethernet MAC 1.3.0 wird als Ethernet MAC eingesetzt werden. Der MAC verfügt über ein AHB-Interface, sodass die genannten Bridges verwendet werden müssen. Sowohl ein AHB Master- als auch ein AHB Slave-Interface sind für den MAC notwendig. Das AHB Master-Interface ermöglicht es dem MAC eine Übertragung der über Ethernet empfangenen Daten zum PCI-Interface zu initiieren, während das AHB Slave-Interface für vom PCI-Interface initiierte Übertragungen verwendet wird. Der Syn1588-Core verwendet nur ein AHB Slave-Interface, da er selbst keine Transfers mit anderen AHB-Busteilnehmern startet, jedoch vom Treiber über den PCI-Bus konfiguriert und ausgelesen werden muss. Abbildung 4.4 zeigt den Aufbau einer NIC mittels des Altera SOPC-Builders.

Die *Avalon to AHB Bridge* kann sowohl vom Syn1588-Core als auch vom Ethernet MAC genutzt werden. Die Unterscheidung, welche Komponente von einem Zugriff betroffen ist, erfolgt anhand des

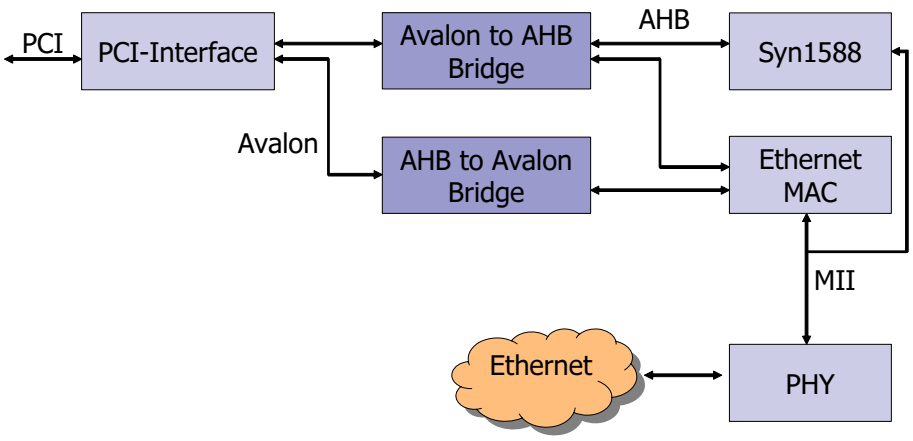

Abbildung 4.4: Modularer Aufbau einer NIC mit Altera Komponenten

Adressbereichs. Der Ethernet MAC benötigt einen 10 Bit-breiten Adressbereich (d. h. 1024 Adressen), dessen Basisadresse nur an einem Vielfachen von 0x400 (dezimal 1024) gesetzt werden kann. Diese Zuteilung kann beliebig geschehen. Es erweist sich jedoch als vorteilhaft die Basisadressen der Komponenten nahe zueinander festzusetzen, damit die *PCI bar size* klein bleibt. Als Basisadresse wurde daher 0x400 gewählt.

Der Ethernet MAC ermöglicht die Verwendung eines RMON (Remote Monitoring)-Blocks. Dieser Block besteht aus zwei RAM Blöcken zu je 64 mal 32 Bits. Diese beiden Blöcke werden verwendet, um mittels Zähler eine Statistik für den Netzwerkverkehr zu führen. So werden darin z. B. diverse Frame-Fehler oder Verzögerungen geloggt. Ein Block wird zum Lesen freigegeben, während der andere Block zum Zählen verwendet wird. Der Wechsel zwischen den beiden Speicherbänken erfolgt durch die Software und kann erst erfolgen, wenn die Abarbeitung des aktuellen Frames abgeschlossen ist. Dies stellt sicher, dass die Statistik eines Frames nicht auf zwei Bänke aufgeteilt wird. Da diese Funktion vor allem der nicht verwendeten *Remote Administration* dient, wurde diese Funktion deaktiviert. Tabelle 4.1 zeigt die Parametrierung des Ethernet MACs.

Tabelle 4.1: Parametrierung des Altera AHB Ethernet MACs

Einstellung	Wert	Kurzbeschreibung
RMON	Off	Die Statistikfunktionen sind deaktiviert.
AHB Base Address	0x400	Die Basisadresse wurde auf 0x400 gesetzt.

4.2.2 Syn1588-Core

Der Syn1588-Core verwendet die gleiche Avalon to AHB Bridge wie der MAC und benötigt einen 9 Bit-breiten Adressbereich (d.h. 512 Adressen) um insgesamt bis zu 128 interne Register verwalten zu können. Durch die Ausrichtung des Adressbusses auf Doppelwörter kommt den beiden nieder-

wertigsten Bits keine Bedeutung zu, sodass der nötige Adressraum viermal so groß wie die Anzahl der vorhandenen Register sein muss. Die Basisadresse wurde auf 0x000 gelegt.

Der MII-Scanner braucht hier vorläufig nicht beachtet zu werden. Dieses externe Modul muss nach der Erstellung des Systems an die Trigger-Leitungen des Syn1588-Cores verbunden werden sowie an die MII-Schnittstelle des Ethernet MACs. Dem MII-Scanner hört lediglich das MII-Interface ab, führt jedoch keine Änderung der MII-Daten durch, sodass seine Eingänge mit den Ausgangstreibern des PHYs und MACs verbunden werden müssen.

Beim Zugriff der PCI-Bridge auf den Syn1588-Core bzw. dem Ethernet MAC ergibt sich folgendes Adresszugriffsschema:

- 0x00000000 - 0x000001FF: Syn1588
- 0x00000400 - 0x000007FF: Ethernet MAC

Beim Zugriff des Ethernet MACs auf die PCI-Bridge erfolgt mit folgenden Adressen:

- 0x00000000 - 0x000FFFFF: PCI-Bridge

4.2.3 PCI-Interface

Die Implementierung des PCI-Interfaces erfolgt mithilfe des PCI-Compilers (Version 4.0.0), welcher ebenfalls für den Design Flow mittels SOPC-Builder geeignet ist. Der PCI-Compiler kann durch die Festlegung zahlreicher Parameter konfiguriert werden, um den geeigneten Betriebsmodus festzulegen, aber auch im Hinblick auf eine Optimierung der benötigten Chipfläche. Die Wahl der Parameter des PCI-Interfaces erfolgte nach dem Gesichtspunkt ein möglichst einfaches System zu erhalten, bei dem die mögliche Adressumsetzung deaktiviert ist. Die Adressumsetzung erlaubt es, gewisse Adressbereiche des PCI-Busses auf andere Adressbereiche des Avalon-Interfaces umzusetzen. Dies geschieht durch den Austausch einer gewissen Anzahl der höherwertigen Bits der Adresse gegen jene, die in der Umsetzungstabelle eingetragen sind.

Die grundsätzliche Frage besteht im Betriebsmodus: Wenn der PCI-Core nicht im Host-Modus z. B. für eine PCI-Backplane eingesetzt werden soll, so muss festgelegt werden, ob das PCI-Gerät selbst Transfers initialisieren können muss oder nicht. Im Allgemeinen benötigt ein Netzwerkadapter diese Funktionalität, um die empfangenen Daten via DMA (Direct Memory Access) in den Arbeitsspeicher schaffen zu können. Weitere Einstellungen betreffen die Performance in Bezug auf die Tiefe der eingesetzten FIFO-Speicher. Die Festlegung der Übertragungsfrequenz des PCI-Bussystem sowie die Bitbreite des Adress-/Datenbusses ist abhängig in welche Backplane oder in welches Motherboard der Netzwerkadapter eingesteckt wird.

Da der PCI-Bus gewöhnlich (die Spezifikation erlaubt jede Frequenz bis zur Maximalfrequenz) mit 33 oder 66 MHz betrieben wird, weitere IP-Cores jedoch oftmals mit höheren Frequenzen betrieben werden können, besteht die Möglichkeit für das Avalon-Interface eine eigene Clock-Domain zu verwenden. Jedes PCI-Gerät verfügt über mindestens ein Basisadressregister (BAR), welches in geeigneter Form konfiguriert werden muss. Vereinfacht gesprochen wird mit dem BAR die Größe des geforderten Adressbereichs festgelegt, die mögliche Adressumsetzung konfiguriert sowie

Tabelle 4.2: Parametrierung des PCI-Interfaces

Einstellung	Wert	Kurzbeschreibung
PCI Device Mode	PCI Master/Target Peripheral	Dies erlaubt sowohl dem Avalon- als auch dem PCI-Bus Zugriffe zu initiieren.
PCI Target Performance	Single-Cycle Transfer only	Dies ist die einfachste und resourcenschonenste Übertragungsart, jedoch nicht die performanteste. Es sind nur Einzel- aber keine Blocktransfers möglich.
PCI Master Performance	Burst Transfers with Single Pending Reads	Jede Lesetransaktion muss vor dem Beginn der nächsten Lesetransaktion abgeschlossen sein.
PCI Bus Speed	33 MHz	Mit dieser Frequenz wird der PCI-Bus getaktet.
PCI Data Width	32 bit	Die Datenbreite des Busses ist 32 Bit.
Clock Domains	Shared PCI and Avalon Clocks	Der PCI-Bus und das Avalon-Interface verwenden einen gemeinsamen Takt.
PCI Bus Arbiter	Arbiter External to Device	Die Busarbitrierung wird durch das Chipset des Computers durchgeführt.
PCI Bar Type	32 bit Prefetchable Memory	Der vom BAR beschriebene Bereich wird als *prefetchable memory* eingebunden.
PCI Bar Size	Auto	Die Größe der PCI Bar wird durch den PCI-Compiler angepasst.
Avalon Base Address	Auto	Basisadresse wird durch den PCI-Compiler angepasst.
Address Translation Table Configuration	Fixed Translation Table	Die PCI-Avalon Adressumsetzung wird beim Compilieren festgesetzt.
Number of Address Pages	1	Es soll nur eine Tabelle zur Adressumsetzung verwendet werden.
Size of Address Pages	1 MByte - 20 bit	Der Adressbereich soll 1 MByte groß sein.
Avalon Base Address	0x00000000	Die Avalon-Basisadresse soll 0 sein.
PCI Base Address	0x00000000	Die PCI-Basisadresse soll 0 sein. Damit ist die Adressumsetzung praktisch inaktiv.
Address Type	32-bit Memory	Zugriffe auf den PCI-Core werden als *memory read* oder *memory write* durchgeführt.

der Typ des Adressbereichs festgelegt (memory oder I/O). Tabelle 4.2 zeigt die Parametrierung des PCI-Interfaces.

4.3 NIC basierend auf IP-Cores der Plattform OpenCores

Grundlage dieser Implementierung sind die IP-Cores von OpenCores. Im Internet befindet sich unter der Adresse http://www.opencores.org eine Plattform für Hardwareentwickler, die ihre IP-Cores zur Verwendung in FPGAs (Field Programmable Gate Array) zur freien Verfügung anbieten. Alle IP-Cores sind im vollen Umfang als Sourcecode entweder in der Hardwarebeschreibungssprache VHDL oder Verilog verfügbar.

Eine PCI-Bridge bietet ein Interface zwischen dem PCI-Bus und einem chipinternen Interface. Als PCI-Bridge dient der von Miha Dolenc und Tadej Markovic in Verilog codierte IP-Core. Dieser IP-Core verwendet – wie die meisten Cores von OpenCores – das WISHBONE-Interface zur Verbindung mit anderen IP-Cores. Als Ethernet MAC soll hier der IP-Core von Igor Mohor verwendet werden, welcher ebenfalls über ein WISHBONE-Interface verfügt. Für eine einfache Netzwerkkarte könnte man diese beiden Cores direkt miteinander verbinden, was in der WISHBONE-Terminologie Point-to-Point-Interconnection genannt wird. Dies bedeutet, dass das WISHBONE-Master-Interface des PCI-Cores mit dem WISHBONE-Slave-Interface des Ethernet MACs verbunden werden muss und umgekehrt, sodass jedes Master-Interface einem Slave-Interface gegenübersteht. Fügt man nun einen weiteren IP-Core wie den Syn1588-Core hinzu, so muss zwischen den Cores eine Verbindungslogik (Interconnection Logic) eingefügt werden, die den Zugriff regelt.

Da der Syn1588-Core außerdem über ein AHB-Interface verfügt, muss das Interface des Cores entweder auf ein WISHBONE-Interface portiert werden, oder aber das AHB-Interface wird mittels einer WISHBONE-AHB-Bridge auf WISHBONE umgesetzt. Solch eine Umsetzung kostet aufgrund der unterschiedlichen Syntax der Schnittstellen stets ein oder mehrere Wartezyklen, was jedoch bei Cores mit niedrigen nötigen Durchsatzraten wie dem Syn1588-Core nicht besonders ins Gewicht fällt. Abbildung 4.5 zeigt den Aufbau mit OpenCores Komponenten.

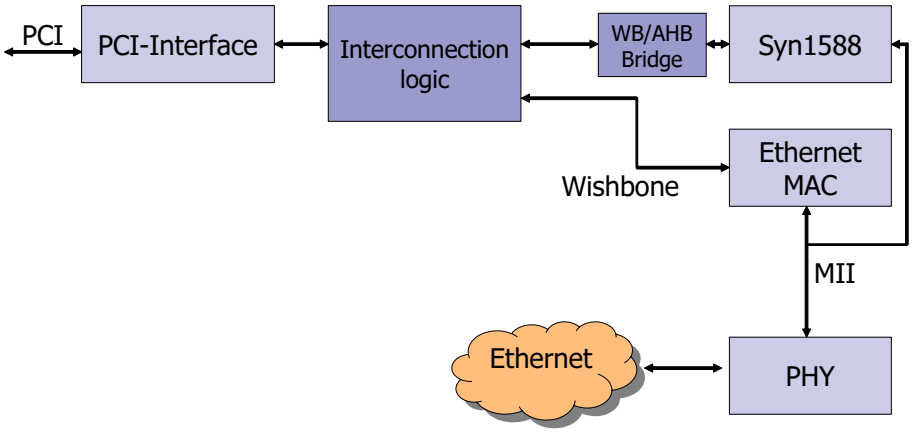

Abbildung 4.5: Modularer Aufbau einer NIC mit OpenCores Komponenten

Im Gegensatz zum graphischen Benutzerinterface der Altera SOPC-Entwicklungsumgebung, die die Integration einer IP-Cores automatisiert, gibt es für das WISHBONE-Interface keine derartige Software. Das Zusammenfügen der einzelnen Cores ist selbst in der Hardwarebeschreibungssprache VHDL oder Verilog durchzuführen. Als standardisiertes Interface sind aber auch Werkzeuge zur Unterstützung der Erstellung der Verbindungslogik verfügbar.

Die Erstellung einer Testbench ist dem Programmierer überlassen. Es besteht die Möglichkeit die bei den IP-Cores beiliegenden Testbenches derart zu verändern, dass man diese für das aus mehreren IP-Core erstellte Design verwenden kann. Das bedingt jedoch eine gute Kenntnis des Cores selbst sowie entsprechenden Testbenches. Der Aufbau der Testbenches erfolgt nicht nach einem vorgegebenen Schema, sodass je nach Autor die Testbenches mit den zahlreichen Steuersignalen einen komplexeren und schlecht dokumentierten Aufbau haben können als der IP-Core selbst. Dies trifft in dieser Form auf die Testbench des PCI-Core zu, welche es dank des monolithischen Charakters schwierig macht, diese in Kombination mit anderen IP-Cores als Ausgangspunkt für eine Testbench zu verwenden.

4.4 Vergleich der Implementierungen

In diesem Abschnitt werden die beiden Implementierungsmöglichkeiten verglichen im Hinblick auf folgende Ziele:

- Kosten
- Chipfläche
- Portierbarkeit und Adaptierbarkeit
- Funktion

4.4.1 Kosten

Ein wichtiges Kriterium sind die Kosten, die durch die Entwicklung und den Betrieb entstehen. Entwicklungskosten entstehen durch die Umsetzung eines Konzepts in Hard- und Software und dem anschließenden Test. Diese Kosten sind maßgeblich davon abhängig, welche Entwicklungswerkzeuge zur Erstellung und zum Test zur Verfügung stehen.

Die Firma Altera stellt für deren FPGAs die Entwicklungssoftware Quartus II zur Verfügung. Diese Software inkludiert zur raschen Erstellung von Systemen aus verschiedenen IP-Cores zwei mögliche Tool-Chains:

- SOPC-Builder
- MegaCore Design Flow

Grundsätzlich erlaubt der SOPC-Builder eine geringe Freiheit in der Wahl der Parameter, da nur die wichtigsten Parameter über eine grafische Oberfläche konfiguriert werden können. Eine Implementierung mit dem MegaCore Design Flow erlaubt im Gegensatz dazu mehr Freiheiten, erfordert jedoch eine fundierte Kenntnis der einzelnen MegaCore-Funktionen. Glaubt man den Versprechungen von Alteras Handbüchern zum Thema SOPC-Builder, so lässt sich ein lauffähiges System bestehend aus zahlreichen IP-Cores innerhalb von Minuten generieren. Was die Erstellung selbst betrifft, ist diese Aussage richtig. Jedoch ist für den Betrieb eines System die genaue Kenntnis jeder Komponente nötig, sodass die Zeit- und Kostenersparnis nur in der fertigen Implementierung der Verbindungslogik zu sehen ist.

Für die WISHBONE-Verbindungsstruktur sind auch Generatoren zur Erstellung der Verbindungslogik verfügbar. Vergleichbar mit der einheitlichen Lösung in Form des SOPC Builders sind diese dennoch nicht. Es wird z. B. keine Syntaxprüfung der Eingaben durchgeführt (ob sich z. B. die Adressbereiche zweier IP-Cores überschneiden) oder aber, ob diese Konfiguration des WISHBONE-Interfaces für den betreffenden IP-Core überhaupt geeignet ist. Das WISHBONE-Interface ist in dieser Hinsicht etwas zu universell: Wenn z. B. ein IP-Core die Generierung des Error-Signal unterstützt, der andere IP-Core jedoch nicht, so kann es zu Problemen mit der Kommunikation kommen, im schlimmsten Fall zu einer Verklemmung (Deadlock). Dies bedingt einen insgesamt höheren Entwicklungsaufwand für ein System auf Basis der IP-Cores von OpenCores.

Neben den einmaligen Entwicklungskosten fallen auch Lizenzkosten für die Nutzung der Entwicklungssoftware und der IP-Cores an. Zur Lizenzierung stehen bei Altera unterschiedliche Lizenzmodelle zur Verfügung: Einerseits auf die Hardware eines Computers gebundene Lizenzen (anhand eines Dongles oder der MAC-Adresse der Netzwerkkarte) sowie Netzwerklizenzen, die von jedem Computer im Netzwerk genutzt werden können. Neben der technischen Realisierung des Lizenzierungsverfahrens, werden Lizenzen mit stark unterschiedlichem Umfang angeboten. Manche Lizenzen erlauben es nur das Design zu testen, jedoch nicht zu synthetisieren, andere wiederum sind auf eine gewisse FPGA-Familie beschränkt. Für die Verwendung in Hardware müssen jedenfalls Lizenzen für die Entwicklungssoftware Quartus II als auch für alle eingesetzten IP-Cores vorhanden sein.

Die Simulation eines System auf Basis von beliebigen IP-Cores ist in der Regel möglich, jedoch wird für die Synthese eine zu bezahlende Lizenz benötigt. Die Preise für die Lizenzen der Software Quartus II und die verfügbaren IP-Cores sind nicht offen gelegt. Es kann daher kaum eine Abschätzung über die anfallenden Kosten getroffen werden.

Die Frage nach der Höhe der Lizenzkosten lässt sich für die Entwicklungsplattform OpenCores einfach beantworten: Es gibt keine Lizenzkosten, da OpenCores auf alle Urheberrechte und Patente verzichtet. Es wird jedoch in den Lizenzbedingungen darauf hingewiesen, dass Dritte Patente oder sonstige Rechte besitzen könnten, was zu Rechtsansprüchen gegenüber dem Lizenznehmer führen könnte.

4.4.2 Chipfläche und maximale Taktfrequenzen

Die Chipfläche und die erreichbaren Taktfrequenzen geben einen Aufschluss darüber, wie effizient ein Design implementiert ist. Grundsätzlich ist eine möglichst kleine Chipfläche bei gleichzeitig möglichst hohen Taktfrequenzen anzustreben. Im Allgemeinen ist beides gleichzeitig nicht erreichbar und man muss einen günstigen Trade-Off finden. Alle Ergebnisse wurden unter den gleichen Einstellungen unter Quartus II erzeugt, wobei das FPGA Cyclone EP1C20F324C6 von Altera zum Vergleich herangezogen wurde. Dieses FPGA wurde gewählt, da die vorhandene Lizenz des Altera Ethernet MACs nicht für alle FPGA-Familien gültig ist, und dieses Cyclone-FPGA über eine ausreichende Menge von logischen Elementen verfügt, um auch den Syn1588-Core inkludieren zu können. Für den Vergleich wurde der Syn1588-Core ausgeschlossen, da dieser in beiden Implementierungen ohnehin die gleiche Fläche einnimmt (rund 9100 Logikelemente sowie 4000 Register und 8 KByte Speicher).

Tabelle 4.3: Bedarf an logischen Elementen sowie maximale Taktfrequenzen

Zellentyp / Taktfrequenz	Altera	OpenCores
Logic Cells	11547	6429
LC Registers	3537	3618
Memory Bits	55232	9216
LUT-Only LCs	8010	2811
Register Only LCs	1573	2397
LUT/Register LCs	1964	1221
Carry Chain LCs	1261	576
PCI_CLK	49,06 MHz	130,48 MHz
internal BUS_CLK	49,06 MHz	33,42 MHz
MII_RX_CLK	135,54 MHz	90,24 MHz
MII_TX_CLK	75,64 MHz	63,62 Mhz

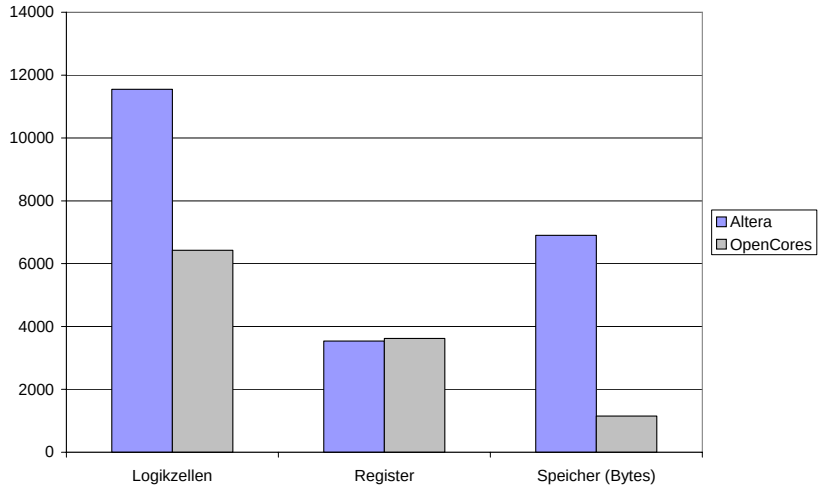

Abbildung 4.6: Bedarf an logischen Elementen in einem Cyclone EP1C20F324C6-FPGA

Wie aus der Tabelle 4.3 ersichtlich ist, benötigt eine Ethernetkarte auf Basis der PCI-Bridge und des

Ethernet MACs von OpenCores deutlich weniger Chipfläche. Etwas anschaulicher ist dies in Abbildung 4.6 dargestellt, wobei hier nicht auf die interne Struktur mit LUT (Look-Up Tables) und kombinierten Strukturen eingegangen wird, da dies nur für die Cyclone-Familie gültig ist, den Bedarf an logischen Elementen in anderen FPGAs aber nicht widerspiegelt. Die Größe des Speichers wurde zwecks besserer Darstellbarkeit auf Bytes umgerechnet.

Der große Unterschied beim PCI-Takt beider Lösungen ergibt sich aus der Tatsache, dass in dieser Altera-Sythese der PCI-Takt auch für das interne Avalon-Interface verwendet wird. Der längste logische Pfad, welcher durch den Syn1588-Core begründet ist, zieht sich somit über die Avalon-AHB-Bridge und die Avalon Switching Fabric durch bis zum PCI-Takt. Würde man zwei getrennte Taktdomainen verwenden würde diese Limitierung nicht bestehen. Vergleichbar sind somit nur die Taktfrequenzen des internen Busses. Die MII-Interfaces werden gemäß Ethernet-Standard mit maximal 25 MHz getaktet, sodass die erreichbaren Taktfrequenzen bei weitem ausreichend sind.

4.4.3 Portierbarkeit und Adaptierbarkeit

Die IP-Cores von Altera sind für eine Auswahl deren eigener FPGA-Familien optimiert und liegen nur in vorkompilierter Form vor; man spricht von Firm-Cores. IP-Cores können als Soft-Cores, Firm-Cores und Hard-Cores vorliegen. Soft-Cores sind Implementierungen, die im Quellcode einer Hardwarebeschreibungssprache vorliegen und somit komplett einsehbar sind. Sofern es die Lizenzbedingungen erlauben, ist es möglich, Soft-Cores an die Bedürfnisse des Projekts anzupassen. Firm-Cores liegen in Binärform vor. Es kann sich hierbei um eine Netzlistenlistendatei oder eine vorkompilierte Bibliotheksdatei handeln, die mit den Entwurfswerkzeugen parametriert und optimiert werden können. Hard-Cores sind die am wenigsten flexiblen Implementierungen. Hierbei ist auch die Anordnung der Transistoren am Chip festgelegt, sodass Hard-Cores z.B. in Form von Masken für die Chip-Herstellung vorliegen können. Sie belegen daher eine definierte Fläche und bieten keine Flexibilität.

Sowohl der PCI- als auch der Ethernet-Core der Firma Altera liegen als Firm-Core vor. Durch die in den Firm-Cores enthaltenen Optimierungen für die FPGAs der Cyclone- und Stratix-Familien ist eine Portierbarkeit auf andere als die genannten FPGAs nicht möglich. Eine Adaptierung des Designs ist aufgrund des Vorliegens als binäre Bibliothek ebenfalls nicht möglich. Ein Eingriff ist nur an den Schnittstellen zwischen den Cores möglich.

Die PCI-Bridge und der Ethernet MAC der Plattform OpenCores liegen hingegen als Soft-Core vor. Im Gegensatz zu den Altera-Produkten ist der komplette Quellcode, welcher in Verilog vorliegt, einsehbar und adaptierbar. Auf eine spezielle Optimierung für eine gewisse FPGA-Familie wurde verzichtet. Dies erlaubt dem Entwickler die volle Kontrolle über den Sourcecode.

Adaptierbarkeit ist somit nur für die IP-Cores von OpenCores gegeben. Ähnlich sieht es bei der Portierbarkeit aus. Die IP-Cores der Firma Altera sind nur unter verschiedenen Altera FPGA-Familien portierbar, nicht jedoch auf FPGAs von Fremdherstellern. Die Wiederverwendbarkeit des Designs ist somit nur auf die eigene FPGA-Familie beschränkt. An dieser Stelle sollte man aber auch den

Vorteil darin ansprechen: So sind die IP-Cores auf der jeweiligen Hardware intensiv getestet und funktionieren weitgehend einwandfrei.

4.4.4 Funktion

Da es mithilfe des SOPC-Builders einfach ist, ein System zusammen mit einer Testbench zu erstellen, wurde dies durchgeführt und die grundsätzliche Funktionalität überprüft. Damit es gemeint, dass Lese- und Schreibzugriffe auf gewisse Register durchgeführt wurden, um somit zu testen, ob die IP-Cores korrekt angesprochen werden können. Ein Funktionstest auf Basis der IP-Cores von OpenCores wurde an dieser Stelle aus Komplexitätsgründen nicht durchgeführt.

Der SOPC-Builder generiert eine Testbench, die das erstellte PCI-Interface initialisiert und stellt dem Benutzer eine Reihe von Tasks zur Verfügung mit denen man über die PCI-Bridge einen Zugriff auf das Design ausführen kann. Die master transactor (mstr_tranx.vhd)-Datei steuert die Funktion eines PCI-Masters, der der PCI-Bridge direkt vorgeschaltet ist. Die Aspekte der Busarbitrierung und Konkurrenz mit weiteren PCI-Mastern wird nicht simuliert, sodass über den PCI-Master initiierte Transaktionen sofort an die PCI-Bridge weitergereicht werden.

Man kann somit mit den zusätzlich in der master transactor-Datei eingetragenen PCI-Transaktionen die PCI-Bridge selbst über Konfigurationszugriffe parametrieren als auch über Speicher- oder I/O-Zugriffe auf die IP-Cores zugreifen. Für den Zugriff auf einen gewissen IP-Core am Avalon-Bus muss der Avalon-Master den Adressbereich und die Bedeutung der Register im IP-Core kennen. Die Adresse am Avalon-Bus muss aber nicht zwangsweise der Adresse am PCI-Bus entsprechen. Denn die PCI-Bridge führt eine Adressumsetzung zwischen den Adressen am PCI-Bus und Avalon-Bus durch. Der Sinn dahinter ist klar: PCI-Adressbereiche können nur einmal vergeben werden, sodass PCI-Karten, die denselben Adressbereich für sich beanspruchen, nicht betrieben werden könnten. In dieser Simulation soll jedoch aus Gründen der besseren Überschaubarkeit die Adressumsetzung deaktiviert sein.

Wird nun eine Schreibtransaktion mittels des mem_wr_32()-Befehls über den PCI-Bus gesendet, so wird diese Transaktion an das Avalon-Interface weitergereicht. D. h. die Adresse wird auf das Avalon-Signal ADDRESS gelegt, die zu schreibenden Daten auf WRITEDATA und die Steuerdaten werden geeignet gesetzt. Die Anbindung der PCI-Bridge an die Verbindungslogik, die Avalon Switching Fabric, erfolgt je nach Konfiguration mit ein bis vier Avalon ports. Betrifft die Transaktion einen IP-Core hinter einer Avalon-AHB-Bridge (z.B. den Ethernet MAC), so wird die Avalon-AHB-Bridge von der Verbindungslogik mittels des Setzens des CHIPSELECT-Signals selektiert, wenn die betreffende Adresse im Bereich der Bridge liegt. Die Avalon-AHB-Bridge nimmt somit am Avalon-Slave-Interface das Kommando und die Daten entgegen und gibt es umgesetzt auf ein AHB-Signal am AHB-Master-Interface der Bridge aus.

Gemäß AMBA-AHB-Spezifikation [3] muss jedoch ein AHB-Master, bevor er den Bus treiben darf, beim Arbiter des AHB-Systems den Zugriff für den Bus anfordern. Dies geschieht durch das Setzen des HBUSREQ-Signals. Sobald der Bus frei ist und der AHB-Arbiter dem AHB-Master den Bus

zuteilen kann, setzt der Arbiter das HGRANT-Signal. Den Zugriff auf den Bus erhält der AHB-Master, wenn sowohl HGRANT als auch Statussignal HREADY bei der steigenden Taktflanke des AHB-Taktes HCLK gesetzt sind. Abbildung 4.7 zeigt die Verbindung von zwei AHB-Slaves über die Avalon-AHB-Bridge an die Avalon Switching Fabric.

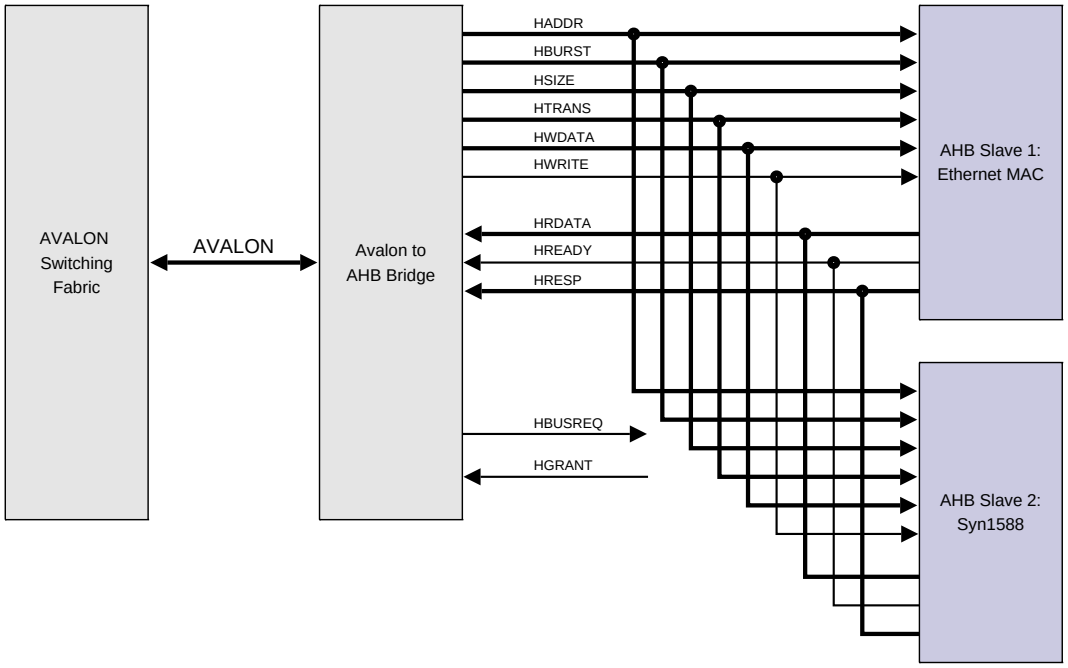

Abbildung 4.7: Verbindung der Avalon Switching Fabric mit zwei AHB-Slaves

Die Implementierung der Avalon to AHB-Bridge scheint diesbezüglich jedoch fehlerhaft zu sein. Sobald die Bridge durch Setzen des Avalon-Signals CHIPSELECT ausgewählt wird, setzt die durch den SOPC-Builder erstellte Logik ebenfalls AHB-seitig HBUSREQ. Da in dem an die Avalon-AHB-Bridge angeschlossenen AHB-Subsystem ohnehin nur ein Master vorhanden ist (die Bridge selbst), ist das Buszuteilungssignal HGRANT korrekterweise auf 1 gesetzt, sodass jeder Buszugriff sofort genehmigt wird.

Unglücklicherweise scheint jedoch die logische Verknüpfung, die das Statussignal HREADY treibt, nicht korrekt zu sein. Denn sobald das HBUSREQ-Signal auf 1 gesetzt wird, geht HREADY in den nicht definierten Zustand *U* über. VHDL verwendet eine 9-wertige Logik, wobei der logische Zustand *U* einen nicht initialisierten Zustand kennzeichnet. Gemäß der logischen Verknüpfungstabelle ergibt die Verknüpfung eines beliebigen Signals mit einem Signal, das den Zustand *U* trägt, ebenfalls den Zustand *U*. Es ist daher anzunehmen, dass HREADY durch ein nicht initialisiertes Signal gebildet wird. Verfolgt man die Generierung des HREADY-Signals zurück so stellt man fest, dass dies tatsächlich der Fall ist, und an der mangelnden Initialisierung von ahb_ethernet_mac_0_s1_hready_-from_sa_master bedingt ist.

Da somit HGRANT und HREADY nie an einer steigenden Taktflanke von HCLK gleich 1 sind, erhält der AHB-Master keinen Buszugriff und wartet für immer. Die Avalon-AHB-Bridge bzw. deren Ansteuerung ist somit fehlerhaft und kann nicht verwendet werden. Es wurden weitere Versu-

che durchgeführt die Anzahl der AHB-Slaves auf einen zu reduzieren bzw. die Konfiguration der IP-Cores zu ändern, doch dies brachte immer wieder ein gleich fehlerhaftes Ergebnis. Als Korrekturmaßnahme könnte man natürlich die Erstellung des HREADY Signals selbst implementieren. Dies ist jedoch insofern schwierig, da die Bridges nur in Form einer Netzlistendatei vorliegen, was Änderungen nahezu unmöglich macht. Alternativ könnte man allen AHB-Cores einen selbst entwickelten Avalon-AHB-Wrapper vorsetzen, sodass IP-Cores mit AHB-Interface nach außen hin ein Avalon-Interface erhalten.

Der Firma Altera dürfte bewusst gewesen sein, dass in Verbindung mit den Bridges gewisse Probleme auftreten können. Obwohl diese in Quartus II Version 5.0 enthalten sind, wurde der Support eingestellt mit dem Hinweis, man möge auf deren proprietäres Bussystem Avalon umsteigen. Schriftliche Anfragen beim technischen Support konnten nicht beantwortet werden, da laut Altera der Support gänzlich eingestellt worden sei. Dies bedeutet natürlich, dass man alle bis dahin entwickelten IP-Cores auf Basis des AHB-Busses mit einem neuen Interface für Avalon ausstatten muss, um diese weiterhin in deren Entwicklungsumgebung verwenden zu können. Mit der Version 5.1 von Quartus II verschwanden schließlich auch die beiden Bridges, sodass man ohnehin nur mehr Cores mit Avalon-Interface verwenden kann bzw. auf einen selbst entwickelten Wrapper zurückgreifen muss.

4.4.5 Auswahl

Nach der Beschreibung der Vor- und Nachteile der beiden möglichen Implementierungen, soll hier eine endgültige Auswahl getroffen werden. Auf die Bildung einer Zielfunktion wurde verzichtet, da eine objektive Bestimmung der Gewichtsfaktoren kaum möglich gewesen wäre. Eine Zusammenfassung der Eigenschaften beider Lösungen wird in Tabelle 4.4 gezeigt.

Tabelle 4.4: Eigenschaften der unterschiedlichen Implementierungsmöglichkeiten

Kriterium	Altera	OpenCores
Kosten: Entwicklung	gering, gute Entwicklungstools	mittel, eigene Anpassungen nötig
Kosten: Lizenzen	Kosten für Entwicklungstools und verwendete IP-Cores	keine
Chipfläche	mittel	klein
Wiederverwendbarkeit	nur für Altera Hardware wiederverwendbar	universell wiederverwendbar
Adaptierbarkeit	nein (Firm-Core)	ja (Soft-Core)
Support	verfügbar	nicht verfügbar
Mängel	fehlerhafte Implementierung des AHB-Dekoders	keine offensichtlichen Mängel

Beide Lösungen haben ihre Stärken und Schwächen. Für die Lösung mit Altera IP-Cores spricht die ausgereifte Entwicklungsumgebung und der hohe Automatisierungsgrad bei der Erstellung des Systems samt Testbench. Jedoch ist die mangelnde Funktion der benötigten Bridges ein schwerwiegender Mangel, der nur durch Erstellung einer eigenen Bridge umgangen werden kann. Für die

IP-Cores von OpenCores sprechen die geringen Kosten und die volle Transparenz des Sourcecodes. Der eingeschränkte Support und die nötigen Anpassungsarbeiten erfordern bei der Erstellung jedoch mehr Einarbeitungszeit. Insgesamt erscheint die OpenCores-Lösung aufgrund der Mängel bei Altera als die Bessere.

5 Implementierung

Dieses Kapitel beschreibt die Implementierung einer Netzwerkkarte mithilfe von der frei erhältlichen *OpenCores* IP-Cores. Im Internet befindet sich unter der Adresse http://www.opencores.org eine Plattform für Hardwareentwickler, die ihre IP-Cores zur Verwendung in FPGAs zur freien Verfügung anbieten. Alle IP-Cores sind in vollem Umfang als Sourcecode entweder in der Hardwarebeschreibungssprache VHDL oder Verilog verfügbar.

An dieser Stelle sollen sowohl die Grundlagen des PCI-Busses als auch des WISHBONE-Interfaces vorgestellt werden. Die Kenntnis dieser Protokolle ist eine Voraussetzung um das Zusammenwirken der IP-Cores erfassen zu können. Es soll hier auch angemerkt werden, dass nur die relevanten Mechanismen der sehr umfangreichen Spezifikationen erwähnt werden. Für detaillierte Information sowie Erweiterungen des PCI-Busses sei auf [19] und [22] verwiesen. Details zu WISHBONE sind unter [11] verfügbar.

5.1 Grundsätzliche Funktionalität des Peripheral Component Interconnect-Busses

Der PCI (Peripheral Component Interconnect)-Bus bezeichnet einen Datenbus mit bidirektionalen Adress- und Datenleitungen. Dieser Bus wird vor allem in Computern verwendet, um Komponenten des Mainboards oder Peripheriekomponenten mit dem Prozessor- und Speicherinterface zu verbinden. Die Übertragung erfolgt synchron zu einem Takt, der im ursprünglichen Standard bis zu 33 MHz sein durfte, was eine Übertragung alle 30 ns erlaubte. Dieser erste Standard aus 1992 (Revision 1.0) wurde durch mehrere Nachfolgestandards ergänzt, wobei dadurch die mögliche Taktrate auf 66 MHz angehoben wurde und auch die Breite des Adress- und Datenbusses von ursprünglich 32 Bit auf 64 Bit angehoben wurde. Dennoch verfügt die Mehrzahl der zur Zeit verwendeten Computer über einen 32 Bit-breiten PCI-Bus, welcher mit 33 MHz getaktet wird.

Im Gegensatz zu früheren Technologien wie dem ISA-Bus oder VESA Local-Bus benötigen PCI-Geräte keinerlei Jumper. Die Konfiguration der PCI-Geräte erfolgt beim Systemstart mithilfe der in jedem Gerät enthaltenen Konfigurationsregistern. Diese Register erlauben es der PCI-Komponente sich zu identifizieren, d. h. um welche Type es sich handelt (Videocontroller, Ethernet, SCSI usw.) sowie welcher Hersteller das Gerät gebaut hat. Weitere Register erlauben die Konfiguration der Speicheradressen, I/O-Adressen, Interrupt Levels usw.

Der PCI-Standard definiert sowohl 5 V als auch 3,3 V Spannungslevels zur Übertragung, wobei 5 V Versorgungsspannung nur auf älteren Karten noch verwendet wird. Um zu verhindern, dass

Einsteckkarten in ein System mit nicht unterstützter Versorgungsspannung gesteckt werden können, wird ein *Keying*-Schema verwendet, welches aufgrund einer nicht vorhandenen Ausfräsung an der Steckverbindung der Einsteckkarte das Einsetzen der Karte in den PCI-Steckplatz mechanisch nicht zulässt. Der PCI-Bus verwendet keine TTL-Treiber zur Datenübertragung wie z. B. der ISA-Bus, sondern speziell für PCI spezifizierte Treiber. Im Gegensatz zum *Wave Switching*, bei dem die Leitungen mit der vollen Spannung getrieben werden, und das Ende jeder Leitung mit Terminatoren abgeschlossen ist, um Reflexionen zu verhindern, verwendet PCI *Reflected-Wave Switching*. Die PCI-Ausgangstreiber legen an die Leitung den halben Nennwert der Spannung an, sodass durch Überlagerung mit der am Leitungsende reflektierten Welle die Nennspannung anliegt.

Initiator, Target und Agents Das PCI-Protokoll unterscheidet zwischen zwei Teilnehmern bei einem PCI-Transfer. Der *Initiator* oder Bus Master ist das Gerät, das den Transfer beginnt. Das *Target* oder der Slave ist das vom Initiator adressierte Gerät. Sowohl Initiator als auch Target werden als *Agent* bezeichnet.

5.1.1 PCI-Signale

Die Signale des PCI-Interfaces sind zu Gruppen zusammengefasst. Die Systemleitungen (Tabelle 5.1) versorgen das PCI-Gerät mit dem Takt und dem Resetsignal. Die Schnittstellensignale steuern die Datenübertragung (Tabelle 5.3). Die Adress-, Daten- und Kommandoleitungen (Tabelle 5.2) tragen die Adressen, Daten und Buskommandos bzw. die *Byte Enable*-Daten. Der PCI-Bus verwendet zur Übertragung der Adressen und Daten ein Zeitmultiplexverfahren. Im Zusammenhang mit Datenbussystemen spricht man von einem *multiplexten Bus* dann, wenn Adressen und Daten über gemeinsame Leitungen zeitversetzt übertragen werden. In der Adressphase werden die Leitungen zur Übertragung der Adresse verwendet, in den folgenden Datenphasen zur Übertragung der Nutzdaten. Dies reduziert die Anzahl der nötigen Pins und damit auch die Kosten von PCI-Komponenten. Eine typische 32 Bit PCI-Steckkarte kommt daher mit rund 50 Pins aus. [22]

Die Fehlermeldungsleitungen (Tabelle 5.4) zeigen im Falle einer gescheiteren Transaktion den Fehler an. Bus Master verfügen außerdem über Arbitrationsleitungen mit denen der Buszugriff mittels eines Arbiters gesteuert wird.

Optional können die Signale für die 64-Bit Erweiterung, ein JTAG-Interface, Interruptleitungen, ein *LOCK*-Signal zum Sperren des Busses sowie ein *CLOCKRUN*-Signal implementiert werden.

Manche Signale des PCI-Busses sind *low aktiv*; diese werden durch das Suffix # am Ende ihres Signalnamens gekennzeichnet.

Die PCI-Leitungen verwenden unterschiedliche Signaltypen:

- **IN** bezeichnet ein gewöhnliches Eingangssignal.
- **OUT** bezeichnet ein gewöhnliches Ausgangssignal.
- **T/S** bezeichnet ein bidirektionales, tri-state Eingangs- und Ausgangssignal.

Tabelle 5.1: PCI-Systemleitungen

Signalname	Typ	Bitbreite	Kurzbeschreibung
CLK	IN	1	Das PCI-Systemtaktsignal bildet eine Referenz für alle Buszyklen. Alle PCI-Signale (außer Reset- und Interrupt-Signale) werden bei der steigenden Taktflanke gesampelt. Die Taktfrequenz kann zum Energiesparen auch gesenkt bzw. komplett abgeschaltet werden.
RST#	IN	1	Das asynchrone Resetsignal versetzt ein PCI-Interface in einen definierten Anfangszustand.

Tabelle 5.2: PCI-Adress- und Datenleitungen

Signalname	Typ	Bitbreite	Kurzbeschreibung
AD()	T/S	32	Der bidirektionale Adress- und Datenbus wird zur Übertragung der Daten und Adressen verwendet.
C/BE#	T/S	4	Das Bus Command and Bytes Enable-Signal erfüllt mehrere Funktionen: Während der Adressphase zeigt dieses Signal die Art des Buscommandos an, während der Datenphase legt es fest welche Bytes von AD gültige Werte enthalten müssen.
PAR	T/S	1	Das gerade Parity-Signal wird über AD und C/BE# gebildet. Es dient dazu, Fehler bei der Übertragung zu erkennen.

Tabelle 5.3: PCI-Schnittstellenleitungen

Signalname	Typ	Bitbreite	Kurzbeschreibung
FRAME#	S/T/S	1	Das Cycle Frame-Signal zeigt Start und Ende einer Bustransaktion an und ist während einer Transaktion stets gesetzt.
IRDY#	S/T/S	1	Das Initiator Ready-Signal wird vom Initiator getrieben und zeigt an, dass bei Schreibzyklen gültige Daten auf AD anliegen; bei Lesezyklen, dass der Initiator bereit zum Lesen ist.
TRDY#	S/T/S	1	Das Target Ready-Signal wird vom Target getrieben und zeigt an, dass bei Lesezyklen gültige Daten auf AD anliegen; bei Schreibzyklen, dass das Target bereit zum Empfang der Daten ist.
STOP#	S/T/S	1	Mit dem Stop-Signal kann das Target Fehler an den Initiator melden.
DEVSEL#	S/T/S	1	Mit dem Device Select-Signal zeigt das adressierte Target dem Initiator an, dass es für die aktuelle Bustransaktion das Target ist.
IDSEL#	IN	1	Das Initialisation Device Select-Signal wird zur direkten Auswahl des Targets verwendet. Dies dient der Vergabe der Basisadresse beim Systemstart.

Tabelle 5.4: PCI-Fehlermeldungsleitungen

Signalname	Typ	Bitbreite	Kurzbeschreibung
PERR#	S/T/S	1	Das Parity Error-Signal wird vom Daten empfangenden Gerät gesetzt zur Meldung von Datenparitätsfehlern.
SERR#	O/D	1	Das Parity Error-Signal wird vom Daten empfangenden Gerät gesetzt zur Meldung von Adressparitätsfehlern.

- **S/T/S** bezeichnet ein bidirektionales, tri-state Signal, welches nur von einem Busteilnehmer getrieben werden darf. Jeder Agent muss dieses Signal einen Taktzyklus auf high setzen, bevor es hochohmig geschaltet wird. Ein Pullup-Widerstand ist notwendig um diesen inaktiven Zustand halten zu können.
- **O/D** bezeichnet ein open drain-Signal. Dieses darf von mehreren Geräten gleichzeitig getrieben werden. Für den inaktiven Zustand ist ein Pullup-Widerstand notwendig.

5.1.2 Adressierung

Alle Transfers auf dem PCI-Bus werden über das PCI CLK-Signal synchronisiert. Die Frequenz dieses Signal beträgt zwischen 0 und 33 MHz. Jede Bustransaktion beginnt mit einer Adressphase, gefolgt von einer oder mehren Datenphasen. Die Adressphase dauert immer einen Taktzyklus, die Anzahl der Datenphasen ist von der Anzahl der Burst-Transfers abhängig. I/O-Transaktionen, welche auf Register eines PCI-Targets zugreifen, haben meist nur eine Datenphase. Speichertransfers, die große zusammenhängende Datenblöcke transferieren, verfügen typischerweise über eine große Anzahl von Datenphasen. Der Nachteil der gemeinsamen Nutzung von AD für Adressen und Daten hinsichtlich des Durchsatzes fällt vor allem bei Transaktionen mit einer großen Anzahl von Datenphasen kaum ins Gewicht.

Zur Verdeutlichung des PCI-Protokolls sei hier ein Beispiel einer Schreibtransaktion aufgeführt [vgl. [22] S. 139ff.], welche in Abbildung 5.1 dargestellt ist. Nach der Buszuteilung durch den Arbiter setzt der Initiator FRAME# um die Transaktion zu beginnen, wobei dieses Signal bis zum Ende der Transaktion anliegen wird. Gleichzeitig treibt er AD mit einer Adresse und legt an C/BE# ein Buskommando. An den Signalen IRDY#, TRDY#, DEVSEL# und PAR wird ein so genannter Turn-Around-Zyklus eingelegt, in welchem diese Signale vom neuen Initiator nicht getrieben werden. Dieser Zyklus ist notwendig, da es sonst in dieser Phase zu einer Überlagerung am Bus kommen könnte, da mehrere Initiatoren den Bus gleichzeitig treiben könnten bedingt durch das notwendige Hochsetzen des Signals vom vorigen Initiator. In Abbildung 5.1 ist dies durch einen rückgerichteten Doppelpfeil dargestellt.

Sofern es sich um eine Schreibtransaktion handelt, ist keine Übergabe von AD mittels eines Turn-Around-Zyklus an das Target erforderlich und der Initiator kann vor der Taktflanke 3 die Daten an AD legen und über C/BE# die Byte Enable-Daten übertragen. Die Byte Enable-Information gibt an, welche Bits der an AD anliegenden Daten für das Target gültig sind. Um dem Target mitzuteilen, dass sich gültige Daten an AD befinden, setzt der Initiator IRDY#. Das Target setzt DEVSEL# um

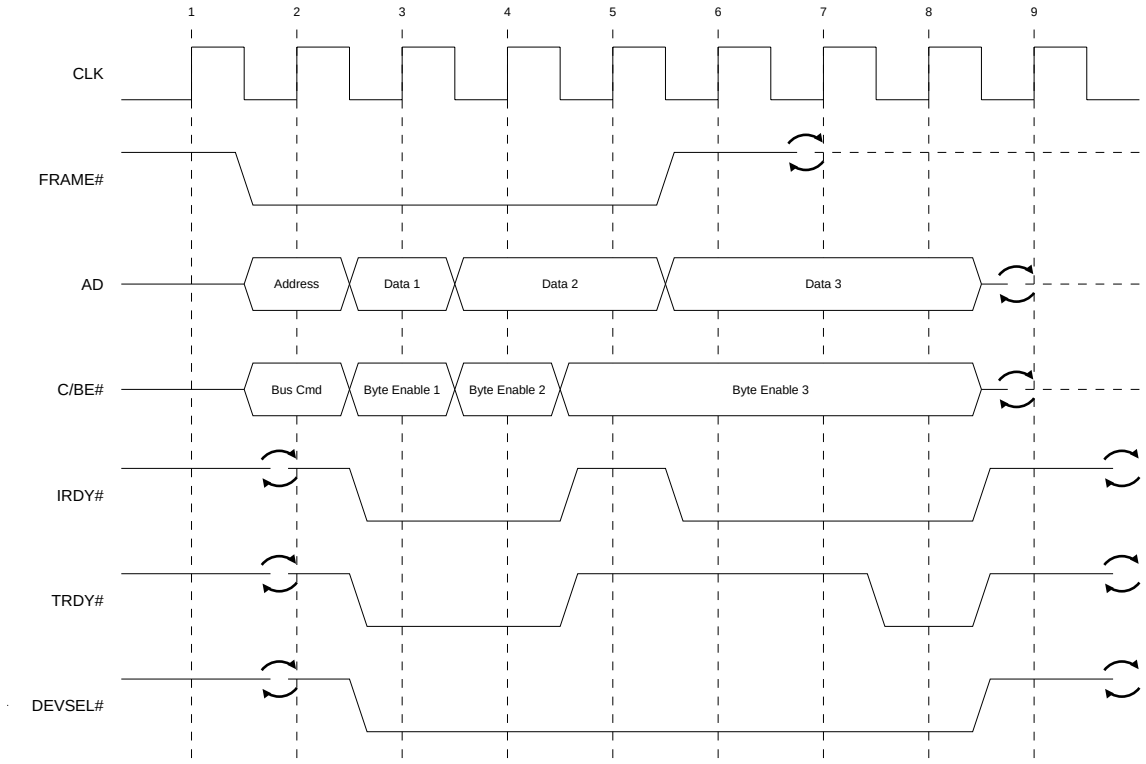

Abbildung 5.1: Beispiel eines PCI-Schreibtransfers mit drei Datenphasen

das Buskommando zu bestätigen und kann nun TRDY# setzen, um die Bereitschaft des Datenempfangs zu signalisieren. An der Taktflanke 3 wird die erste Datenphase erfolgreich abgeschlossen, da sowohl Target als auch Initiator ihre Bereitschaft durch TRDY# und IRDY# anzeigen. Für die Taktflanke 4 gilt selbiges.

Bei der Taktflanke 5 sind sowohl IRDY# als auch TRDY# gelöscht, was die Agents grundsätzlich dann machen, wenn sie einen Wartezyklus einlegen möchten z. B. weil sie für die Verarbeitung der Daten mehr Zeit benötigen. Vor der Taktflanke 6 ist der Initiator wieder bereit und legt *Data 3* an AD und löscht FRAME#, da es sich um die letzte Datenphase handelt. Das Target ist jedoch nicht bereit und hat TRDY# gelöscht. Schließlich ist das Target an Taktflanke 8 wieder bereit und übernimmt die Daten. Mit dem Abschluss des letzten Zyklus kann der Bus an einen anderen Initiator übergeben werden.

Auch wenn der Initiator, wie an Taktflanke 5, einen Wartezyklus einlegt, so muss er dennoch den Bus weiterhin treiben. Was die Daten an AD betrifft, so steht es ihm frei, AD mit einem beliebigen Wert zu treiben, solange IRDY# gelöscht ist. C/BE# muss er aber in jedem Fall mit der Byte Enable-Information des nächsten Taktzyklus treiben. Vor Taktflanke 6 hat der Initiator das FRAME#-Signal gelöscht, um anderen Initiatoren und dem Target das Ende der Übertragung zu signalisieren. Es ist aber seine Aufgabe dieses Signal vor der Übergabe noch einen Taktzyklus lang zu löschen und anschließend hochohmig zu schalten; dies gilt auch für IRDY#, TRDY# und DEVSEL#.

Ein Lesetransfer läuft relativ ähnlich ab. Der Initiator legt bei der Taktflanke 2 anstatt eines Schreib-

kommandos ein Lesekommando an C/BE#. Da der AD-Bus aber von nun an vom Target getrieben werden soll, muss an Taktflanke 3 ein Turn-Around-Zyklus folgen, welcher dadurch zustande kommt, dass das Target TRDY# löscht. Die weiteren Schritte sind vergleichbar mit einer Schreibtransaktion, wobei eben das Target die Leitungen AD und PAR statt dem Initiator treibt.

5.1.3 Busarbitrierung

Wenn zu einer gewissen Zeit ein oder mehrere PCI-Bus-Master einen Transfer über den Bus durchführen möchten, so muss sichergestellt sein, dass stets nur ein Bus Master Zugriff auf den PCI-Bus erhält. Würden mehr als ein Bus Master den Bus treiben, so würde sich das Signal überlagern und die Übertragung wäre gestört. Um dies zu verhindern, verfügt jeder Bus Master über ein REQ#-Signal, mit dem er dem Arbiter anzeigen kann, dass er den Bus beansprucht, sowie über ein GNT#-Signal, mit dem der Arbiter dem Bus-Master den Bus zuteilen kann. Die Arbitrierung erfolgt *hidden* (verdeckt). Dies bedeutet, dass der Arbitrierungsvorgang auch durchgeführt werden kann, während der Bus durch einen anderen Bus-Master beansprucht wird. Erhält ein PCI-Master die Erlaubnis zum Buszugriff durch Setzen des GNT#-Signals durch den Arbiter, so muss der PCI-Master noch warten bis der aktuelle Initiator seine Transaktion abgeschlossen hat. Dies ist erkennbar durch das Negieren von FRAME#. Die Arbitrierung verbraucht daher selbst keine Taktzyklen.

Abbildung 5.2 zeigt vier PCI-Geräte gemeinsam mit einem Arbiter. Üblicherweise ist der Arbiter nicht eine separate Komponente, sondern im PCI-Chipset integriert.

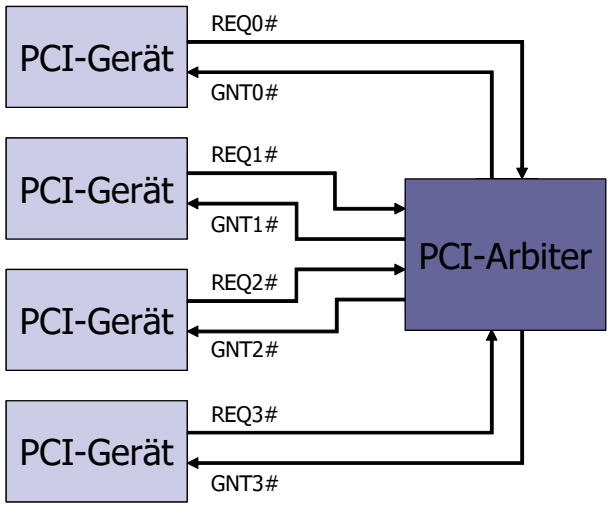

Abbildung 5.2: PCI Busarbitrierung

Die PCI-Spezifikation[19] gibt lediglich den Mechanismus der Arbitrierung vor, nicht jedoch, welches PCI-Gerät den Zugriff auf den Bus bekommen soll, wenn mehrere Bus-Master zugleich den Zugriff beanspruchen. Sie legt lediglich fest, dass das Prinzip von *Fairness* angewendet werden soll. Der Bus Arbiter muss daher den Bus intelligent managen, damit es nicht zu Starvation (Verhungern), Deadlock (Verklemmung) oder übermäßiger Beanspruchung des Busses durch ein PCI-Gerät

kommt. Die Monopolisierung des Busses durch ein PCI-Gerät wird u. a. durch den PCI Latency Timer verhindert. Der Latency Timer erlaubt es, eine gewisse Anzahl von Zyklen festzulegen innerhalb der ein PCI-Gerät seine Transaktion abschließen muss, wenn ein anderer Bus Master den Bus beansprucht. Große Werte (128) verbessern zwar die Performance, verschlechtern jedoch das Echtzeitverhalten.

Eine weitere Aufgabe des Arbiters ist es, ein *Bus Parking*-Schema zu implementieren. Unter Bus Parking versteht man, dass der PCI-Bus einem definierten PCI-Master zugeteilt wird, wenn kein anderer PCI-Master den Bus beansprucht. Dieser vordefinierte PCI-Master erhält also den Bus mittels eines GNT#-Signals durch den Arbiter ohne selbst mittels REQ# den Bus angefordert zu haben. Der Arbiter parkt also den Bus auf einem Master. Der Grund dafür ist, dass nicht getriebene PCI-Leitungen dazu führen, dass die CMOS-Eingangstreiber aller PCI-Geräte zu schwingen beginnen und dadurch einen hohen Strom verbrauchen. Der Master, auf dem der Bus geparkt ist, muss aus diesem Grund AD, C/BE# und PAR# treiben. Falls der Arbiter kein Bus Parking implementiert hat, so muss er selbst dafür sorgen, dass die genannten Signale einen definierten Wert ausweisen, sobald der Bus von keinem Master mehr getrieben wird.

5.1.4 Buskommandos

Sobald ein PCI-Master den Zugriff auf den PCI-Bus durch den Arbiter erhalten hat, kann dieser Transaktionen durchführen. Jede Transaktion beginnt mit einer Adressphase, bei welcher der PCI-Master anzeigt um welche Art von Transaktion es sich handelt. Dies wird durch das C/BE#-Signal angezeigt. Diese Bedeutung erhält das C/BE#-Signal nur in der Adressphase, während es in den folgenden Datenphasen angibt, welche Bytes des 32 Bit-Datenwortes von AD() gültig sind. Tabelle 5.5 zeigt eine Übersicht der 16 möglichen Buskommandos.

Tabelle 5.5: PCI-Kommandos

C/BE3#	C/BE2#	C/BE1#	C/BE0#	Buskommando
0	0	0	0	Interrupt Acknowledge
0	0	0	1	Special Cycle
0	0	1	0	I/O Read
0	0	1	1	I/O Write
0	1	0	0	Reserviert
0	1	0	1	Reserviert
0	1	1	0	Memory Read
0	1	1	1	Memory Write
1	0	0	0	Reserviert
1	0	0	1	Reserviert
1	0	1	0	Configuration Read
1	0	1	1	Configuration Write
1	1	0	0	Memory Read Multiple
1	1	0	1	Dual-Access Cycle
1	1	1	0	Memory Read Line
1	1	1	1	Memory Write and Invalidate

Interrupt Acknowledge Dieser Befehl wird verwendet, um den Interrupt-Request zu bestätigen und um den Interruptvektor vom System Interruptcontroller zu lesen. Die Adresse ist bei diesem Befehl implizit vorgegeben und braucht nicht übermittelt zu werden.

Special Cycle Ein PCI-Master verwendet diesen Befehl, um einen Broadcast von Nachrichten zu mehreren Targets durchzuführen. Die Implementierung dieses Befehls ist optional. Definiert sind drei Nachrichtentypen: *Shutdown*, *Halt* und *x86 specific message*. Jedes Target muss selbst entscheiden, wie es die Nachrichten interpretiert.

I/O Read und I/O Write Die I/O-Lese- und Schreibbefehle werden zum Zugriff auf Adressbereiche verwendet, die als I/O-Adressraum konfiguriert sind.

Memory Read und Memory Write Der Memory Read- und Memory Write-Befehl wird zum Zugriff auf Systemspeicher verwendet, welcher als *non-prefetchable Memory* eingebunden ist. Als prefetchable gilt ein Speicher, wenn sich der Speicherinhalt durch vorausschauende Lesetransaktionen nicht ändert. Vereinfacht gesprochen versteht man unter *prefetchable Memory* einen Speicher, bei dem Zugriffe nicht verbrauchend sind. D. h. eine mehrmalige Lese- oder Schreiboperation (mit selbem Datum) ändert seinen Inhalt nicht. Für I/O-Transaktionen gilt dies im Allgemeinen jedoch nicht, sodass dieser Speicher als *non-prefetchable Memory* eingebunden werden muss. Es sind sowohl Einzelübertragungen als auch Blockübertragungen möglich.

Configuration Read und Configuration Write Diese Befehle werden verwendet, um auf die internen Konfigurationsregister der PCI-Bridge zugreifen zu können. Damit kann z. B. die Basisadresse und die Images der PCI-Bridge angepasst werden.

Memory Read Line und Memory Read Multiple Diese Befehle werden für den Lesezugriff auf *prefetchable Memory* verwendet. Der Memory Read Line-Befehl erlaubt das Lesen mehrerer Doppelwörter, jedoch maximal einer Cache Line. Der Memory Read Multiple-Befehl erlaubt es auch über Cache Line-Grenzen hinweg zu lesen.

Dual-Access Cycle Der Dual-Access Cycle-Befehl ermöglicht es einem 32 Bit PCI-Bussystem einen 64 Bit-breiten Adressbereich zu adressieren, indem zwei 32 Bit-Adressen zu einer 64 Bit-breiten Adresse zusammengesetzt werden.

Memory Write and Invalidate Der Memory Write and Invalidate-Befehl ist gleich dem Memory Write-Befehl, wobei hierbei garantiert ist, dass eine ganze Cache Line geschrieben wird.

5.1.5 Gerätekonfiguration

Bevor ein PCI-Gerät adressiert werden kann, muss es über eine Basisadresse verfügen. Zum Systemstart wird jedem PCI-Gerät eine Basisadresse vom System zugeteilt. Zur Ermittlung der Basisadresse wird gemäß PCI-Spezifikation wie folgt vorgegangen: Das Gerät legt die Größe des angeforderten Speicherbereichs im Basisadressregister fest, indem es eine gewisse Anzahl der niederwertigsten Bits auf 0 setzt und diese nicht beschreibbar macht. Wenn nun ein Gerät z. B. ein 1 MByte großen Adressraum beansprucht, so setzt es die untersten 20 Bits fix auf 0.

Während des Systemstarts wird mittels eines Configuration Write-Befehls das Basisadressregister mit lauter Einsen beschrieben und anschließend via Configuration Read ausgelesen. Da nun die unteren 20 Bits nicht beschreibbar sind und fix auf 0 gesetzt sind, kann das System erkennen wieviel Speicher das Gerät in Anspruch nehmen möchte. Das System vergibt daraufhin eine Basisadresse durch erneutes Schreiben in das Basisadressregister. Jedes PCI-Gerät darf bis zu 6 Basisadressen verwenden, wobei der reservierte Bereich je Basisadresse zwischen 4 KByte und 2 GByte sein kann.

Während dieser Initialisierungsphase kann ein PCI-Gerät jedoch noch nicht gewöhnlich adressiert werden. Um ein PCI-Target anzusprechen verfügt jedes über eine eigene IDSEL-Leitung. Ist IDSEL gemeinsam mit FRAME# gesetzt, so können über C/BE# Konfigurationstransaktionen angefordert werden, um das Gerät zu konfigurieren.

5.2 WISHBONE-Architektur

Die WISHBONE System-On-Chip Architektur ist ein universelles Interface für Halbleiter IP-Cores. Ein einheitliches chipinternes Interface soll es ermöglichen, die Integration der einzelnen IP-Cores zu vereinfachen und die Wiederverwendbarkeit zu erhöhen. WISHBONE versteht sich als universelles Interface. Es definiert die Mechanismen des Datenaustauschs zwischen den einzelnen Modulen, ohne jedoch die Funktion der einzelnen Module einzuschränken. WISHBONE kann aufgrund der Einfachheit und der Flexibilität sowohl für eine SoC (System-on-Chip)-Architektur zur Verbindung mehrerer IP-Cores herangezogen werden, aber auch als Off-Chip Interconnection zur Verbindung von Einsteckkarten über eine Backplane.

5.2.1 Vergleichbare Interfaces

Der Halbleiterhersteller Altera (www.altera.com) bietet unter dem Namen *Avalon SoC*-Bus ein proprietäres Interface an, das jedoch nur für deren eigene Produkte verwendbar ist. Etwas umfangreicher und weitergehend ist der AMBA (Advanced Microcontroller Bus Architecture)-Bus von der Firma ARM [3]. Vergleichbar mit dem AMBA-Bus bietet auch IBM ihr Interface namens CoreConnect an (http://www.ibm.com/chips/products/coreconnect/). Sowohl der AMBA als auch CoreConnect definieren nicht nur ein einziges Protokoll, sondern – im Gegensatz zu WISHBONE – jeweils drei Protokolle für unterschiedliche Anwendungsbereiche und Geschwindigkeiten. AMBA

definiert für niedrige Leistungen das APB-Protokoll (Advanced Peripheral Bus), für mittlere Leistungen das ASB-Protokoll (Advanced System Bus) und für hohe Leistungen das AHB-Protokoll (Advanced High-performance Bus). Vergleichbar dazu sind die CoreConnect-Protokolle DCR (Device Control Register) für niedrige Performance, OPB (On-Chip Peripheral Bus) für mittlere Leistungen und PLB (Processor Local Bus) für hohe Leistungen. Kommen mehrere unterschiedliche Protokolle einer Protokollgruppe zum Einsatz, so erfolgt die Verbindung mittels Bridges, da sich die unterschiedlichen Leistungsstufen dieser Protokolle auch in der Syntax unterscheiden.

5.2.2 Interface Spezifikation

WISHBONE wird spezifiziert durch Regeln (*Rules*), Empfehlungen (*Recommendations*), Vorschlägen (*Suggestions*), Erlaubnissen (*Permissions*) und Beobachtungen (*Observations*). Regeln bilden das Rahmenwerk und garantieren eine Kompatibilität zwischen den Interfaces. Diese haben verpflichtenden Charakter und müssen stets eingehalten werden. Empfehlungen sind nicht verpflichtend, dienen dem Anwender aber als Hinweis, um das WISHBONE-Interface möglichst günstig im Hinblick auf Performance und Stabilität zu entwickeln. Vorschläge beinhalten Hinweise, wobei der Entwickler selbst entscheiden muss, ob dieser Hinweis für ihn von Relevanz ist. Erlaubnisse zeigen unter zahlreichen Lösungen einen gangbaren Weg auf. Beobachtungen beinhalten keinerlei Hinweise, zeigen jedoch Zusammenhänge und Implikationen mit anderen Regeln auf.

WISHBONE verwendet ein Master/Slave-Modell. Dies bedeutet, dass stets Master-Interfaces einen Buszyklus beginnen und Slave-Interfaces vom Master adressiert werden. Die Verbindung der einzelnen Interfaces erfolgt über eine Verbindungslogik, welche nach WISHBONE-Nomenklatur *INTERCON* heißt. Die Spezifikation legt nicht fest, welche Art von Verbindungslogik verwendet werden soll, man spricht in der WISHBONE-Nomenklatur von *variable interconnection*. Die Kommunikation zwischen Master und Slave kann durch eine Punkt-zu-Punkt-Verbindung (einem System aus einem Master und einem Slave), einer Datenflussverbindung (einer Kette mehrerer hintereinander geschalteter Interfaces), einen gemeinsamen Bus oder auch durch einen Kreuzschienenverteiler realisiert sein. Damit unterscheidet sich WISHBONE auch wesentlich von herkömmlichen Verbindungsschemas wie PCI, ISA-Bus usw., welche die Topologie und das Verbindungsschema fest vorgeben, sodass diese als *static interconnection* bezeichnet werden können. WISHBONE definiert daher keinen WISHBONE-Bus, sondern eine Verbindungsstruktur, die in der WISHBONE-Spezifikation als Anlehnung an das Telefonnetz als Wolke dargestellt wird. Wie die Abbildung 5.3 [vgl. [11]] gezeigt wird, können außerhalb der INTERCON-Struktur auch Master und Slaves direkt miteinander verbunden werden, was noch zusätzliche Flexibilität bringt.

5.2.3 WISHBONE-Signale

Dieser Abschnitt beschreibt die einzelnen Signale, die in der WISHBONE SoC-Architektur verwendet werden. Manche dieser Signale sind optional und nicht in jedem Interface vorhanden.

Die Spezifikation sieht vor, stets den Typ des Signals (Eingang oder Ausgang) mittels eines Suffix festzulegen. So handelt es sich bei einem Signal, dass mit _ I endet um ein Eingangssignal, ein Signal

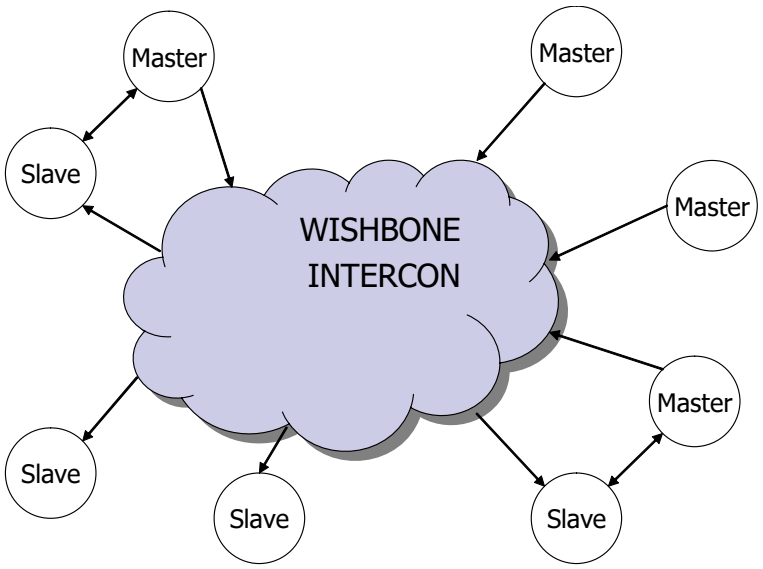

Abbildung 5.3: WISHBONE Verbindungsstruktur

das mit _O endet um ein Ausgangssignal. WISHBONE verwendet für die Interfaces nur Signale, die entweder Eingangs- oder Ausgangssignale sind, aber keine bidirektionalen Signale. Alle Signale haben eine *active high*-Logik.

Die Signale können zu verschiedenen Gruppen zusammengefasst werden. Die in Tabelle 5.6 gezeigten WISHBONE SYSCON-Signale werden durch das SYSCON-Modul eingeprägt und beinhalten das Takt- sowie das Resetsignal. Alle WISHBONE-Interfaces innerhalb einer Taktdomain werden mit denselben SYSCON-Signalen versorgt. Die zweite Kategorie sind die von Master und Slave gemeinsam verwendeten Signale. Diese beinhalten im Wesentlichen ein bis zu 64 Bit breites Datensignal in beide Richtungen neben den Tag-Signalen. Tags sind benutzerdefinierte Signale, die dazu verwendet werden können, um Signalen, auf die sie sich beziehen, eine weitere Bedeutung zu geben. Ein typisches Beispiel hierfür ist z. B. die Implementierung von Parity-Bits, um Übertragungsfehler erkennen zu können. Tabelle 5.7 zeigt diese Signale.

Die Signale des Masters und des Slaves sind von Typ (Eingang, Ausgang) genau entgegengesetzt, sodass z. B. ein ACK_O stets auf ein ACK_I trifft. Diese Signale beinhalten einen bis zu 64 Bit-breiten Adressbus sowie verschiedene Steuersignale, die die Datenübermittlung (das *Handshaking)* steuern. Das SEL-Signal legt dabei fest, wo sich während eines Lese- oder Schreibzyklus gültige Daten auf dem Datenbus DAT befinden müssen. Die Breite des SEL-Signals ist abhängig von der Granularität des Ports. Für eine 8 Bit-Granularität beträgt die Breite von SEL je 1 Bit je 8 Bit von DAT. So korrespondiert jedes Bit von SEL mit einem Byte von DAT beginnend mit dem niederwertigsten BIT (LSB). Tabelle 5.8 zeigt die Signale eines WISHBONE-Masters und Tabelle 5.9 die des Slaves.

Tabelle 5.6: WISHBONE SYSCON-Signale

Signalname	In/Out	Bitbreite	Kurzbeschreibung
CLK_O	Out	1	Das WISHBONE-Taktsignal regelt den Ablauf des synchronen Interfaces. Alle Signale werden bei der steigenden Taktflanke gesampelt.
RST_O	Out	1	Das Resetsignal setzt alle WISHBONE-Interfaces in einen definierten Anfangszustand. Das Resetsignal ist synchron zum Takt, sodass sich der Anfangszustand der WISHBONE-Interfaces an der steigenden Taktflanke von CLK nach dem Negieren des Resetsignals einstellen muss.

Tabelle 5.7: Gemeinsame WISHBONE-Signale

Signalname	In/Out	Bitbreite	Kurzbeschreibung
CLK_I	In	1	Das Takteingangssignal koordiniert die internen Aufgaben der Interfaces. Alle WISHBONE Ausgangssignale müssen vor der steigenden Flanke von CLK_I stabil sein und bei der steigenden Flanke einen gültigen Wert aufweisen.
DAT_I()	In	bis zu 64	Das Dateneingangsfeld wird benutzt die eigentlichen Binärdaten zu übertragen.
DAT_O()	Out	bis zu 64	Das Datenausgangsfeld wird benutzt die eigentlichen Binärdaten zu übertragen.
RST_I	In	1	Der Reset-Eingang erzwingt einen Neustart des WISHBONE Interfaces, indem das Signal das WISHBONE-Interface auf den Anfangszustand setzt. Dieses Signal kann auch dazu verwendet werden, IP-Cores zu resetten.
TGD_I()	In	beliebig	Das Tag-Eingangsfeld wird benutzt um den im DAT_I-Feld übertragenen Daten eine besondere Bedeutung zu geben.
TGD_O()	Out	beliebig	Das Tag-Ausgangsfeld wird benutzt um den im DAT_O-Feld übertragenen Daten eine besondere Bedeutung zu geben.

Tabelle 5.8: WISHBONE Master-Signale

Signalname	In/Out	Bitbreite	Kurzbeschreibung
ACK_I	In	1	Das Acknowledgment-Signal zeigt an, sobald es gesetzt ist, dass ein Buszyklus normal abgeschlossen wurde.
ADR_O()	Out	bis zu 64	Das Adressfeld wird verwendet um Adressen zu übertragen.
CYC_O	Out	1	Das Cycle Output-Signal zeigt an, sobald es gesetzt ist, dass gerade ein gültiger Buszyklus durchgeführt wird. Es bleibt bei einem Blocktransfer vom ersten bis zum letzten Datentransfer gesetzt.
ERR_I	In	1	Das Fehlereingangssignal zeigt einen unerwarteten Zyklusabbruch an.
LOCK_O	Out	1	Das Lock-Signal zeigt an, dass der Buszyklus nicht unterbrechbar ist. Dieses Signal wird gesetzt um alleinigen Zugriff zu fordern.
RTY_I	In	1	Das Retry-Signal zeigt an, dass das Interface nicht bereit ist Daten zu empfangen oder zu senden und, dass der Zyklus wiederholt werden soll.
SEL_O()	Out	bis zu 8 (siehe Text)	Das Ausgangsselektionsfeld zeigt während eines Lesezyklus an, wo sich gültige Daten auf DAT_I befinden.
STB_O	Out	1	Das Strobe-Signal zeigt einen gültigen Transfer an. Der Slave setzt als Antwort entweder ACK_I, ERR_I oder RTY_I.
TGA_O()	Out	beliebig	Das Address Tag-Ausgangsfeld beinhaltet zusätzliche Informationen zum Adressfeld ADR_O.
TGC_O()	Out	beliebig	Das Cycle Tag-Ausgangsfeld beinhaltet zusätzliche Informationen zum Cycle Output-Signal.
WE_O	Out	1	Das Write Enable Output-Signal zeigt an, ob es sich bei dem aktuellen Buszyklus um einen Lese- oder Schreibzyklus handelt. Für Schreibzyklen wird WE_O gesetzt.

Tabelle 5.9: WISHBONE Slave-Signale

Signalname	In/Out	Bitbreite	Kurzbeschreibung
ACK_O	Out	1	Das Acknowledgment-Signal zeigt an, sobald es gesetzt ist, dass ein Buszyklus normal abgeschlossen wurde.
ADR_I()	In	bis zu 64	Im Adressfeld wird vom Master die Adresse des Transaktion übermittelt
CYC_I	In	1	Das Cycle Input-Signal zeigt einen gültigen Buszyklus an.
ERR_O	Out	1	Das Error Out-Signal zeigt dem Master einen unerwarteten Zyklusabbruch an.
LOCK_I	In	1	Das Lock-Signal zeigt an, dass der aktuelle Buszyklus nicht unterbrechbar ist. Ein Master erhält somit alleinigen Zugriff auf einen Slave.
RTY_O	Out	1	Das Retry-Signal zeigt an, dass das Interface nicht bereit ist Daten zu empfangen oder zu senden und, dass der Zyklus wiederholt werden soll.
SEL_I()	In	bis zu 8 (siehe Text)	Das Ausgangsselektionsfeld zeigt während eines Schreibzyklus an, wo sich gültige Daten auf DAT_I befinden bzw. während eines Lesezyklus, welche Daten von DAT_O gültig sein müssen.
STB_I	In	1	Das Strobe-Signal zeigt dem Slave an, dass dieser ausgewählt wurde. Der Slave setzt als Antwort auf STB_I entweder ACK_O, ERR_O oder RTY_O.
TGA_I()	In	beliebig	Das Address Tag-Eingangsfeld beinhaltet zusätzliche Informationen zum Adressfeld ADR_I.
TGC_I()	In	beliebig	Das Cycle Tag-Eingangsfeld beinhaltet zusätzliche Informationen zum Cycle Input-Signal.
WE_I	In	1	Das Write Enable Input-Signal zeigt an, ob der aktuelle Buszyklus ein Schreib- oder Lesezyklus ist. Für Schreibzyklen ist WE_I gesetzt.

5.2.4 Das Handshaking-Protokoll

Die einfachste Art der Verbindung eines Masters mit einem Slave stellt eine Punkt-zu-Punkt-Verbindung dar. Wie in Abbildung 5.4[vgl. [11]] gezeigt, ist ein Master mit einem Slave direkt verbunden, wobei vom SYSCON-Modul der Takt angelegt wird bzw. das Resetsignal generiert wird. Alle Signale werden im Folgenden aus der Sicht des Masters beschrieben, die Signalnamen am Slave-Interface können anhand der Verbindung zwischen dem Master und dem Slave bestimmt werden.

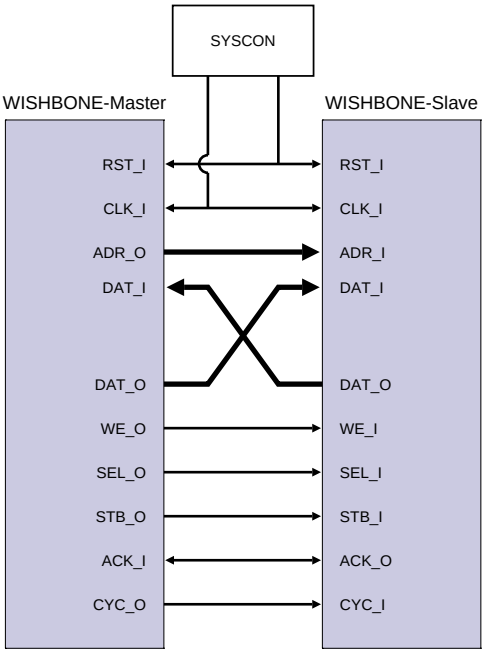

Abbildung 5.4: Punkt-zu-Punkt-Verbindung

Der Master kann eine der drei grundlegenden Transfers am WISHBONE-Interface initiieren:

- SINGLE READ oder SINGLE WRITE: Dieser Buszyklus dient der Übertragung eines einzelnen Datums.
- BLOCK READ oder BLOCK WRITE: Dieser Buszyklus dient der Übertragung eines Datenblocks, wobei der Master den Beginn und das Ende durch Setzen bzw. Negieren von CYC_O anzeigt.
- READ MODIFY WRITE (RMW): Dieser Buszyklus erlaubt das Lesen, Verändern und Zurückschreiben eines Speicherbereichs. Dieser Buszyklus wird atomar durchgeführt, sodass kein anderer Master den Zugriff unterbrechen kann.

Wie werden nun die Daten auf dem WISHBONE-Interface übertragen? Ein einfacher Lese- und Schreibtransfer ist in Abbildung 5.5 dargestellt. Der Master beginnt die Übertragung, indem er die Adresse an ADR_O sowie die zur Adresse gehörenden Tags an TGA_O legt. Zugleich legt der Master über das Schreibsignal WE_O fest, dass dies ein Lesetransfer ist, sowie über SEL_O welcher Teil der Daten an DAT_I gültig sein muss. Mittels CYC_O und STB_O teilt der Master dem Slave mit, dass er einen Transfer beginnen möchte.

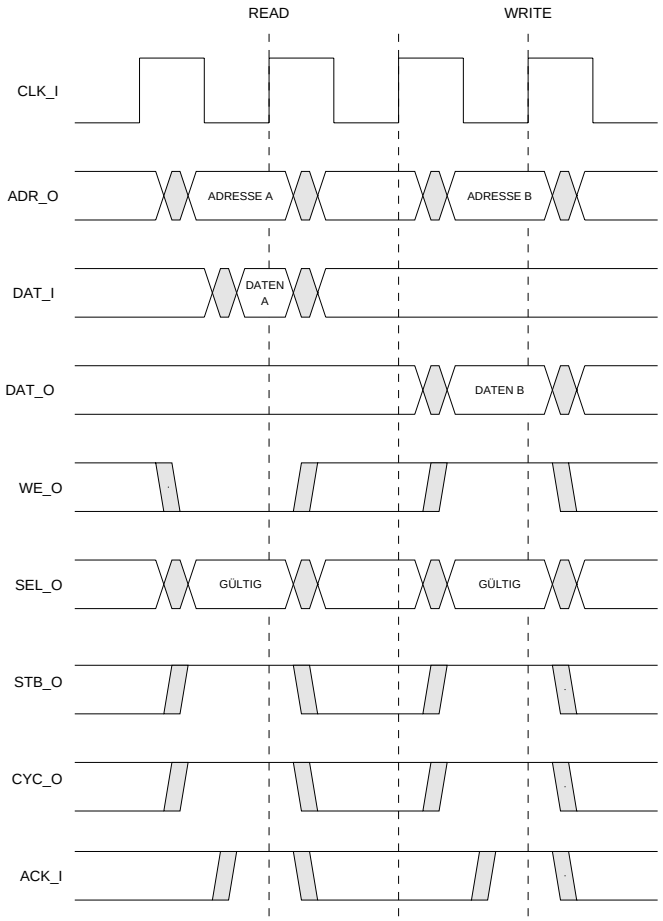

Abbildung 5.5: WISHBONE: Single Read und Single Write

Der Slave erkennt durch STB_O und CYC_O den Beginn eines Transfers. Er indiziert seinen Speicher mit der Adresse *A* und legt die aus dem Speicher entnommenen Daten an DAT_I. Dass die Daten an DAT_I gültig sind, signalisiert der Slave dem Master durch das Setzen des Acknowledgment-Signals ACK_I. Bei der steigenden Taktflanke von CLK, welche im Impulsdiagramm mit *READ* eingezeichnet ist, übernimmt der Master die Daten von DAT_I. Der nächste Datentransfer kann sofort wieder starten. Dieser wurde im Diagramm offen gelassen.

Im dritten Taktzyklus (*WRITE*) ist ein Schreibtransfer dargestellt. Dieser gleicht dem Lesetransfer weitgehend, wobei der Master zum Schreiben WE_I setzt und die zu schreibenden Daten an den Datenbus DAT_O legt. Der Slave übernimmt die Schreiboperation und quittiert den Erhalt durch ACK_I.

Im Idealfall kann je Taktzyklus ein komplettes Datenwort mit bis zu 64 Bit (Breite des Datenbusses) übertragen werden. Bei Lesetransfers bleibt dem Slave aber nur wenig Zeit die Daten an den Datenbus zu legen. Er muss in einem Taktzyklus die Adresse in seinem Speicher indizieren und im selben Taktzyklus den Speicherinhalt an den Datenbus legen. Bei großen Systemen kann das zu Problemen mit der Signallaufzeit führen. Wenn sichergestellt ist, dass ein Slave stets ohne Wartezyklus antworten kann, so muss das ACK_I-Signal (sowie ERR_I und RTY_I) asynchron generiert werden. Möchte ein Slave Wartezyklen einfügen, so hat er nur die Generierung des Acknowledgment-Signals um ein oder mehrere Takte zu verzögern z. B. durch die Verwendung eines Registers.

Die Blocktransfers basieren auf den Einzeltransfers. Der Master setzt über die gesamte Dauer des Transfers das Cycle Output-Signal CYC_O. Die einzelnen Übertragungen, welche *Phasen* genannt werden, werden durch STB_O gestartet. Bei dieser Form der Übertragung können sowohl Master als auch Slaves Wartezyklen einfügen. Der Slave fügt – wie bei einem einzelnen Transfer – Wartezyklen durch das verzögerte Anlegen von ACK_I ein. Der Master kann durch das Löschen von STB_O den Beginn der nächsten Phase verzögern. Es findet solange kein Transfer statt, solange der Master nicht STB_O wieder setzt. Wird eine Verbindungsstruktur mit einem Arbiter zur Verbindung mehrerer WISHBONE-Interfaces verwendet, so dient CYC_O zur Anforderung eines Slaves beim Arbiter. Wenn sichergestellt sein muss, dass ein Blocktransfer in einem Stück abgeschlossen werden muss, so kann der Master mittels LOCK_O verhindern, dass der Arbiter ihm den Slave entzieht für den Fall, dass ein anderer Master denselben Slave für sich beansprucht.

Der RMW (Read Modify Write)-Zyklus ist ein Blocktransfer mit zwei Phasen, wobei die erste Phase eine Lesephase und die zweite eine Schreibphase ist. Durch Verwendung der LOCK_O-Signals wird dieser Transfer atomar durchgeführt, wodurch der RMW-Zyklus zur Synchronisation mittels Semaphoren eingesetzt werden kann.

5.3 PCI-Bridge

Als PCI-Bridge dient der von Miha Dolenc und Tadej Markovic in Verilog beschriebene IP-Core, welcher als zweite Schnittstelle (neben PCI) ein WISHBONE-Interface bietet. Diese Bridge erfüllt

PCI-seitig die *PCI Specification 2.2* und WISHBONE-seitig die *WISHBONE SoC Interconnection architecture Specification Rev. B*, wobei die Datenbreite des WISHBONE-Interfaces auf 32 Bit fix festgelegt ist. [8]

5.3.1 Architektur der PCI-Bridge

Die PCI-Bridge besteht aus zwei unabhängigen Einheiten:

- WISHBONE Slave Unit
- PCI Target Unit

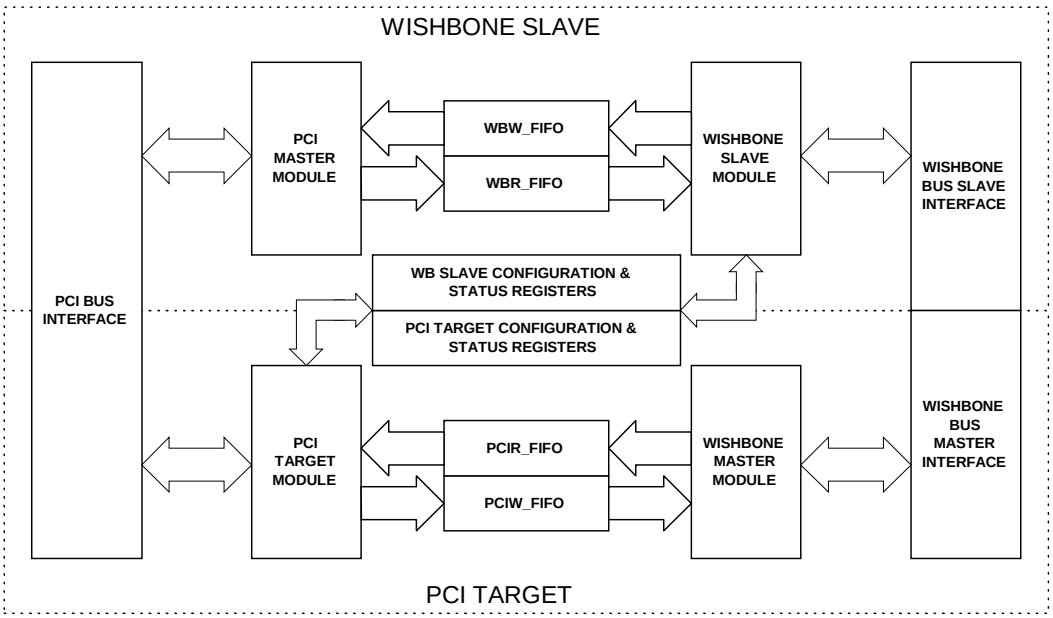

Abbildung 5.6: PCI-Bridge-Architektur

WISHBONE Slave Unit

Die WISHBONE Slave Unit dient als Slave auf der WISHBONE-Seite und kann Transaktionen am PCI-Bus als PCI-Master starten. Andere WISHBONE-Busteilnehmer können daher über das WISHBONE Slave-Interface der PCI-Bridge auf den PCI-Bus zugreifen. Für diesen Zweck stehen ein bis fünf Images zur Verfügung. Jedes Image besteht aus mehreren Registern, die den Zugriff regeln:

- Base Address Register
- Address Mask Register
- Translation Address Register
- Image Control Register

Die ersten drei Register erlauben es zwischen WISHBONE und PCI-Bus eine Adressumsetzung durchführen zu können. Mithilfe der Adressumsetzung (*Address Translation*) können WISHBONE-Adressen eindeutig auf andere Adressen des PCI-Adressraums abgebildet werden. Mithilfe des Address Mask Registers wird festgelegt, wie viele der höherwertigen Bits ausgeblendet werden und durch die Adresse, welche im Translation Address Register gespeichert ist, ersetzt werden.

Jedes dieser Images kann aus Sicht des PCI-Busses sowohl in den I/O-Bereich als auch in den Memory-Bereich eingeblendet werden. Schreibtransaktionen auf den PCI-Bus werden als *posted-writes* gehandhabt. Das bedeutet, dass Schreibtransaktionen zwischengepuffert werden und WISHBONE-seitig nicht gewartet werden muss bis die Schreibtransaktion abgeschlossen ist. Lesetransaktionen werden als *delayed-reads* durchgeführt, sodass das lesende WISHBONE-Gerät auf den Abschluss der Transaktion warten muss. Es besteht auch die Möglichkeit die Lesetransaktionen als pre-fetched Reads durchzuführen. Dabei werden die Daten im Voraus anfordert, um diese rascher bereitstellen zu können. Dies bedingt jedoch, dass der betreffende Speicherbereich als *pre-fetchable* markiert ist, was nur unter gewissen Vorraussetzungen gilt. Für die Zwischenspeicherung der zu schreibenden Daten dient WBW_FIFO, für die zu lesenden Daten WBR_FIFO. Die obere Hälfte von Abbildung 5.6[vgl. [8]] zeigt die WISHBONE slave unit.

PCI Target Unit

Die PCI Target Unit kann von anderen PCI-Geräten verwendet werden, um Zugriff auf Geräte zu erhalten, die an das WISHBONE-Interface angeschlossen sind. PCI-seitig erscheint die PCI Target Unit als Slave und WISHBONE-seitig als Master. Vergleichbar mit der WISHBONE Slave Unit können auch hier ein bis sechs Images konfiguriert werden. Die Images selbst sind das Gegenstück zu denen der WISHBONE Slave Unit. Auch hier kann eine Adressumsetzung durchgeführt werden sowie das Image entweder in den I/O-Bereich oder in den Memory-Bereich eingebunden werden.

Zur Zwischenspeicherung werden die beiden FIFO-Speicher PCIR_FIFO und PCIW_FIFO verwendet. Die untere Hälfte von Abbildung 5.6 zeigt die PCI target unit.

Adressumsetzung

Sowohl die WISHBONE Slave Unit als auch die PCI Target Unit verwenden ein Adressumsetzungsschema. Abbildung 5.7 zeigt die Adressumsetzungslogik. Die AND und OR-Blöcke sind bitorientiert zu verstehen.

Das Address Mask Register definiert die Größe eines Images und damit wie viele der höherwertigen Bits ersetzt werden sollen. Die Maske muss im Bit 31 (welches das MSB ist) eine 1 enthalten und darf bis zum Bit 11 durchgehend mit 1 besetzt sein; der restliche Teil der Maske ist mit 0 besetzt. Die Länge dieser Kette beginnend mit dem MSB kann vom Benutzer gewählt werden und definiert die Größe des Images. Daraus ergibt sich die kleinste Größe eines Images zu 4 KByte, die maximale Größe zu 2 GByte.

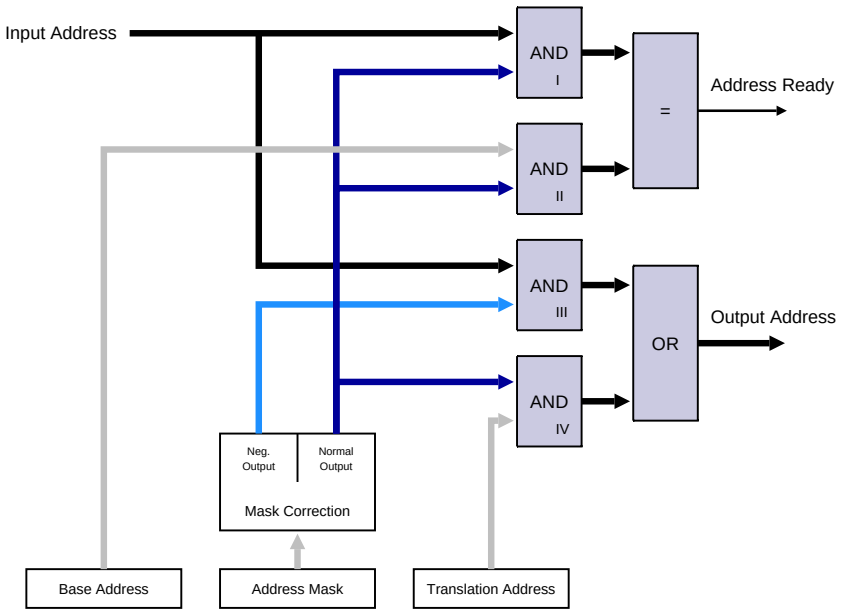

Abbildung 5.7: Adressumsetzung

Die Adresse ergibt sich somit aus den niederwertigen Bits der Eingangsadresse und den höherwertigen Bits aus dem Address Translation Register (siehe AND-Block III und IV). Der Adress-Decoder jedes Images muss feststellen können, ob die Eingangsadresse in den Adressbereich des Images fällt. Dazu wird mittels der AND-Blöcke I und II die Adresse mit der Base Address des Images verglichen und bei Gleichheit das Address Ready-Signal gesetzt.

5.3.2 Konfiguration und Parametrierung

Um die PCI-Bridge in einem PCI-System verwenden zu können müssen zu Beginn gewisse Grundeinstellungen getroffen, die fix in der Hardware einzutragen sind. Neben der Festlegung der Tiefe der verschiedenen FIFO-Speicher und anderer Einstellungen, die zur Synthese benötigt werden, muss der so genannte Type0-Header festgelegt werden. Dieser erstreckt sich über die ersten 64 Doppelwörter im Configuration Space der PCI-Bridge, wobei die ersten 16 Doppelwörter in der PCI-Spezifikation eine vordefinierte Bedeutung haben.

Alle PCI-Geräte bis auf PCI-to-PCI-Bridges verfügen über einen Type0-Header, in welchem die grundlegendsten Einstellungen der PCI-Bridge eingetragen sind. Diese Einstellungen werden beim Start eines PCI-Systems benötigt, um die Geräte identifizieren zu können und diese korrekt konfigurieren zu können. Der Zugriff auf diesen Konfigurationsbereich erfolgt durch spezielle Type0-Transaktionen, welche von der Host-Bridge beim Start des Systems generiert werden. Für die Initierung dieser Type0-Transaktionen stehen wiederum zwei unterschiedliche Mechanismen für x86-Architekturen zur Verfügung, auf die hier nicht näher eingegangen werden soll. Dieser Type0-Header kann bei der Implementierung des PCI-Interfaces als Bridge nur vom PCI-Bus aus geschrieben werden und kann WISHBONE-seitig nur ausgelesen werden, im Gegensatz zu den übrigen

Registern. Folgende Register müssen definiert sein:

- VendorID: Dieses Register gibt den Hersteller des Geräts an und ist in der Hardware fixiert. Die Zuordnung der numerischen VendorID zu einem Hersteller wird von der PCI SIG überwacht.
- DeviceID: Dieser Wert gibt den Typ des Geräts an und wird vom Hersteller vergeben.
- Class Code: Dass Klassenregister gibt an, um welches Gerät es sich handelt und ist für sich in Subklassen aufgeteilt. Z. B. 0x020000 für einen Netzwerkadapter mit der Subklasse Ethernet.
- Revision ID: Diese ID wird vom Hersteller vergeben und gibt die Revision an.
- Header Type: Der Header Type gibt an, ob sich nach der PCI-Bridge ein Gerät mit ein oder mehreren Funktionen befindet bzw. ob die PCI-Bridge als PCI-to-PCI-Bridge verwendet wird.

Ein minimales PCI-Gerät muss außerdem noch mindestens das *Status Register* sowie das *Command Register* des Type0-Headers implementiert haben. Abgesehen vom Type0-Header verfügt die PCI-Bridge über weitere Register, die die Parameter der Images festlegen und die Generierung der Interrupts steuern. Der komplette Konfigurationsbereich umfasst 128 Doppelwörter. Die für den Type0-Header nötigen Einstellungen sowie weitere grundlegende Parameter für die OpenCores PCI-Bridge werden in Tabelle 5.10 beschrieben. Diese Einstellungen sind in der Datei pci_user_-constansts.v gespeichert.

5.4 Ethernet MAC

Zur Implementierung des Ethernet MACs wurde der Ethernet MAC von Igor Mohor [18] verwendet, welcher frei auf der Internetplattform *OpenCores* in Verilog verfügbar ist. Dieser MAC beherrscht die notwendigen Betreibsmodi mit 10 und 100 MBit/s. Er verfügt zur Verbindung mit anderen IP-Cores über ein WISHBONE Master/Slave-Interface sowie über ein MII (Media Independent Interface) zur Anbindung an einem PHY (Physical Layer Device).

Im OSI-ISO-Referenzmodell führt der MAC die Aufgaben der Schicht 2a (Media Access Control) aus. Die Anbindung an den Physical Layer erfolgt über den Reconcilation Layer und die MII-Schnittstelle. Das MII ist ein medienunabhängiges Interface, sodass bei der Änderung des Medium nur der PHY getauscht werden muss [20]. Das MII überträgt die Daten zwischen PHY und MAC mittels eines 4 Bit-breiten Busses, wobei TXD zur Übertragung vom MAC zum PHY verwendet wird und RXD zum Empfangen vom PHY. Die Dateneinheit des MII beträgt 4 Bit, die als Nibble bezeichnet werden. Bytes werden zur Übertragung über das MII in zwei Nibbles aufgeteilt, wobei das erste Nibble zur Übertragung der 4 niederwertigsten Bit verwendet wird, das zweite Nibble zur Übertragung der 4 höherwertigsten Bits. Das MII besteht aus den in Tabelle 5.11 beschriebenen Signalen. Alle Signale werden aus der Sicht des MACs bezeichnet.

Der MAC wird vom PHY von zwei Taktsignalen versorgt: Dem Taktsignal TX_CLK zum Versenden der Daten und RX_CLK zum Empfangen der Daten. Die Taktrate ist vom Übertragungsmodus abhängig und beträgt für den 100 MBit/s-Modus 25 MHz, für den 10 MBit/s-Modus 2,5 Mhz. Dies

Tabelle 5.10: Konstanten für die OpenCores PCI-Bridge

Parameter	Wert	Kurzbeschreibung
GUEST	defined	GUEST legt fest, dass die Bridge für ein PCI-Gerät und nicht für eine Backplane verwendet werden soll.
WBW_ADDR_LENGTH	3	Die WISHBONE Write Address Length legt die Tiefe des Schreibbuffers für WISHBONE fest.
WBR_ADDR_LENGTH	6	Die WISHBONE Read Address Length legt die Tiefe des Lesebuffers für WISHBONE fest.
PCIW_ADDR_LENGTH	3	Die PCI Write Address Length legt die Tiefe des Schreibbuffers für PCI fest.
PCIR_ADDR_LENGTH	6	Die PCI Read Address Length legt die Tiefe des Lesebuffers für PCI fest.
NO_CNF_IMAGE	defined	Auf die Implementierung des von WISHBONE beschreibbaren Konfigurationsimages wird verzichtet.
PCI_BA0_MEM_IO	0b0	Das letzte Bit legt fest, dass das PCI Image 0 als memory mapped implementiert werden soll. Die Basisadresse wird beim Systemstart festgelegt.
WB_NUM_OF_DEC_ADDR_LINES	1	Dies legt fest, wie viele Bits der PCI-Adresse dekodiert werden sollen. Für 1 Bit ergibt sich ein WISHBONE-Adressraum vom 2 GByte.
WB_DECODE_FAST	defined	Decode Fast erlaubt es der PCI-Bridge mit nur einem Wartezyklus auf den WISHBONE-Bus zuzugreifen. Maximal sind drei Wartezyklen programmierbar.
WB_CONFIGURATION_BASE	0x00000	Diese Einstellung definiert die Basisadresse des Konfigurationsbereichs beim Zugriff über WISHBONE.
PCI33	defined	Dies gibt den PCI-Betriebsmodus an und legt die maximale spezifizierte Taktfrequenz fest.
WB_RTY_CNT_MAX	0xFF	Dieser begrenzt die maximale Anzahl von Retries der Zustandsmaschine des WISHBONE-Interfaces bevor ein Error signalisiert wird.
HEADER_VENDOR_ID	0x1172	Die Hersteller-ID 0x1172 zeigt ein Altera-Gerät an.
HEADER_DEVICE_ID	0x0001	Die Geräte-ID wurde frei gewählt.
HEADER_REVISION_ID	0x01	Die Revisions-ID wurde ebenfalls frei gewählt.

Tabelle 5.11: MII-Signale

Signalname	In/Out	Bitbreite	Kurzbeschreibung
RX_CLK	In	1	Empfangstaktsignal
TX_CLK	In	1	Sendetaktsignal
TXD	Out	4	Sendedaten
RXD	In	4	Empfangsdaten
TXD	Out	4	Sendedaten
RX_DV	In	1	Daten an RXD sind gültig
RX_ERR	In	1	Empfangsfehler
TX_EN	Out	1	Daten an TXD sind gültig
TX_ERR	In	1	kennzeichnet Übertragungsfehler
RX_ERR	In	1	kennzeichnet Empfangsfehler
COL	In	1	Kollisionssignal
CRS	Out	1	Carrier Sense-Signal
MDC	Out	1	Taktsignal für serielle Übertragung zum PHY
MDIO	In/Out	1	serielles Datensignal zum PHY

ergibt sich aus der Übertragung von jeweils einem Nibble je Taktzyklus. Zusätzlich stehen mehrere Signale zur Übertragung des Status des PHY sowie zur Signalisierung von Fehlern zur Verfügung.

5.4.1 Aufgaben des Physical Layer Devices

Der PHY führt mit den Nibbles eine 4B/5B-Kodierung durch d. h. jede der 16 möglichen Bitkombinationen eines Nibbles werden mittels einer Tabelle in ein 5 Bit langes Symbol umgesetzt. Dadurch entstehen zusätzliche 16 Symbole, die zum Teil zur Signalisierung verwendet werden bzw. als verbotene Symbole (Violation Symbols) gekennzeichnet sind. Tritt ein verbotenes Symbol beim Empfang auf, so deutet dies auf einen Übertragungsfehler hin. Diese 4B/5B-Kodiertabelle ist so gewählt, dass bei einer beliebigen Folge aufeinanderfolgender Nibbles, die Folge einer Null-Sequenz limitiert ist. Damit ist gemeint, dass nach der Serialisierung der 5B Symbole niemals mehr als vier Nullen in Folge auftreten können.

Die Ethernet-Variante 10Base-T basiert auf einer NRZI (Non Return to Zero/Inverted)-Kodierung, bei der eine logische 1 als Pegelwechsel übertragen wird und eine logische 0 durch einen fehlenden Wechsel übertragen wird. Um eine sichere Übertragung zu garantieren, muss der Pegel ausreichend oft wechseln, um die Taktinformation aus dem Datenstrom gewinnen zu können und den Gleichanteil des Signals zu unterdrücken. Daher kodiert man die Nibbles mit einer 4B/5B-Kodierung bevor diese nach dem NRZI-Verfahren kodiert übertragen werden. Mit einem Overhead in der Höhe von 25% erreicht man neben der Sicherstellung der Taktgewinnung und der Unterdrückung des Gleichanteils eine zusätzliche Sicherung der Daten.

Bei der am weitesten verbreiteten Variante von Ethernet, 100Base-TX, wird anstatt der NRZI-Kodierung das Multi Level Threshold 3 (MLT-3)-Verfahren verwendet, welches drei Stufen zur Codierung der Daten vorsieht: +1, 0 und -1. Diese mehrstufige Kodierung erlaubt es, die nötige Übertragungsfrequenz zu verringern, sodass aufgrund des reduzierten Spektrums günstigere Kabel eingesetzt

werden können. Neben dem MLT-3-Verfahren wird noch mittels eines rekursiven digitalen Filters ein Scrambling durchgeführt. Man versteht darunter eine Verwürfelung der Daten, die das Frequenzspektrum glättet und sich somit ebenfalls günstig auf die Übertragung und elektromagnetische Abstrahlung auswirkt. Jedes Bit des gesendeten und empfangenen Datenstroms wird mit folgender Funktion verschlüsselt:

$$X[n] = X[n-1] + X[n-9]$$

Das Filter besteht somit aus einem 11 Bit breiten rückgekoppelten Schieberegister, sodass Codewiederholungen erst nach 2048 Bit auftreten können. Da sowohl Sender als auch Empfänger die Verschlüsselungsfunktion kennen, können die Bits wieder dekodiert werden. Im Gegensatz zu Ethernet mit 10 Mbit/s verfügt Ethernet mit 100 MBit/s über die Möglichkeit Idle-Symbole zu versenden. Ein Idle-Symbol ist ein 5B-Symbol mit fünf Einsen, welches als zusätzliches Symbol im 4B/5B-Kodierungsverfahren festgelegt ist. Diese Symbole werden in Übertragungspausen versendet und erlauben es, eine Link-Erkennung durchführen zu können, und bieten Ethernet-Geräten die Möglichkeit, sich gegenseitig anhand der Idle-Symbole zu synchronisieren. [20]

Aufgrund der unterschiedlichen Kodierung besitzen PHYs zwei komplette Transceiver, einen für 10BaseT und einen für 100BaseTX, wobei je nach Betriebsmodus (10 oder 100 MBit/s) einer der beiden Transceiver aktiv ist. Dies gewährleistet Kompatibilität zu älteren 10BaseT-Geräten.

5.4.2 Aufbau von Ethernet-Frames

Ethernet gruppiert zusammenhängende Bytes zu sogenannten Frames (Datenrahmen). Der MAC packt Nutzdaten zur seriellen Übertragung über das Medium in einen Rahmen (Frame) ein. Ein Frame besteht aus der Präamble, der Zieladresse, der Quelladresse, dem Längen/Typenfeld, den eigentlichen Nutzdaten sowie ein Prüfsumme, der Frame Check Sum (FCS). Die Festlegung des Aufbau der Frames erfolgte durch die DIX-Gruppe, jedoch veränderte die IEEE für IEEE 802.3 später den Aufbau geringfügig. [10] [20] Abbildung 5.8 zeigt das Format eines Ethernet-Frames.

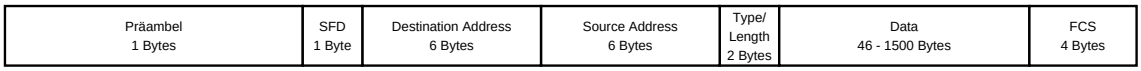

Abbildung 5.8: Aufbau eines Ethernetframes

Die Präambel dient der Synchronisation des Empfänges und soll einen Verlust der ersten Bits der Nutzdaten verhindern und steht am Beginn jedes Frames. Gemäß Ethernet-Standard besteht sie aus 8 Bytes, deren Bits zwischen 1 auf 0 wechseln. Durch diese achtmalige 10101010 Bitkombinationen kann der Empfänger seinen Takt einstellen und es wird sichergestellt, dass der Empfänger das Datensignal zum richtigen Zeitpunkt abtastet. Das letzte Bit der Präambel ist ebenfalls auf 1 gesetzt und zeigt den Start des Frames an. Im IEEE 802.3 Standard wird das gesamte letzte Byte daher Start-of-Frame-Delimiter (SFD) genannt, wobei der SFD im IEEE 802.3 Standard nicht mehr zur Präambel zählt, sodass die Präambel nur 7 Bytes lang ist. Die Präambel des Ethernet-Standards ist daher gleich der Präambel plus dem SFD in der IEEE 802.3-Spezifizierung.

Darauf folgen das Destination Address Field sowie das Source Address Field, welche beide 6 Byte lang sind. Das Destination Address Field enthält die MAC Adresse auf welche das Frame adressiert ist; das Source Address Field enthält die MAC-Adresse, welche das Frame lokal ausgesendet hat. Anschließend an die Adressen folgt entweder das Type Field (bei Ethernet) oder das Length Field (bei IEEE 802.3) sowie die eigentlichen Nutzdaten, welche mindestens aus 46, maximal aus 1500 Bytes bestehen können. Die Mindestlänge ist bedingt durch das Zugriffsverfahren CSMA/CD, welches von einem Frame mit mindestens 64 Bytes ausgeht (inkl. der Header). Wird diese minimale Größe eines Frames nicht erreicht, so werden an die Daten 0x00 Füllbytes angehängt, sodass das Datenfeld 46 Bytes lang ist. Das Längenfeld enthält jedoch weiterhin die Länge der Nutzdaten exklusive der Füllbytes. Die maximale Größe ist beschränkt durch Typ/Längenfeld, das für die Angabe der Länge nur Werte bis 1500 aufweisen darf. Ist in diesem Feld ein Wert ab 1536 (0x600) eingetragen, so wird dies nach IEEE 802.3 als Protokolltyp interpretiert, bis zu 1500 jedoch als Längenangabe für das Datenfeld.

Abschließend folgt die 4 Byte breite Frame Check Sum (FCS). Die FCS wird mithilfe eines CRC (Cyclic Redundancy Check)-Algorithmus berechnet und wird verwendet, um Übertragungsfehler erkennen zu können. Das Generatorpolynom zur Berechnung der FCS lautet wie folgt [10]:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Die Berechnung der FCS erstreckt sich über die Zieladresse, die Quelladresse, das Typ-/Lengenfeld und die Daten. Der Sender hängt die FCS an jedes Frame. Der Empfänger berechnet ebenfalls die FCS und vergleicht die empfangene FCS mit der berechneten. Stimmen beide Werte nicht überein, so ist von einer fehlerhaften Übertragung auszugehen, und der MAC verwirft das Frame. Neben einer ungültigen FCS kann ein Frame auch dann verworfen werden, wenn die Länge des Datenpakets nicht ein Vielfaches von 8 Bit aufweist bzw. wenn die Länge der Daten eines 802.3-Frames nicht mit der im Längenfeld gespeicherten Länge übereinstimmt.

Zur Konfiguration und zum Auslesen von Statusinformationen des PHYs steht ein serielles Interface zur Verfügung, welches über die Datenleitung MDIO (Management Data Input/Output) und die Taktleitung MDC (Managment Clock Data) mit dem PHY kommuniziert. Bei jeder steigenden Taktflanke von MDC wird aus einem Schieberegister das niederwertigste Bit auf die Datenleitung MDIO geschoben und somit an den PHY übertragen. Es stehen folgende drei Kommandos zur Verfügung:

- Write Control Data: schreibt in die Statusregister des PHYs
- Read Status: liest von den Statusregistern und Kontrollregistern des PHYs
- Scan Status: kontinuierliches Auslesen der Statusregister

5.4.3 Architektur des Ethernet-Cores

Der Ethernet MAC von Igor Mohor [18] besteht aus fünf Modulen: Das Sendemodul (TX Ethernet MAC) sendet die Daten an den PHY. Das Empfangsmodul (RX Ethernet MAC) übernimmt die Da-

ten vom PHY. Das Kontrollmodul (MAC Control Module) wird zur Steuerung der Sende- und Empfangsmodule benötigt. Diese drei Module können, wie in Abbildung 5.9 gezeigt, zum MAC-Modul zusammengefasst werden. Zur Konfiguration und zum Auslesen des Status des PHY ist außerdem ein MII Managementmodul vorhanden. Ergänzt wird der Ethernet-Core durch ein WISHBONE Host-Interface. Der MAC verfügt nach außen hin über ein WISHBONE Master/Slave-Interface zur Kommunikation mit anderen WISHBONE-Komponenten sowie dem MII-Interface zur Anbindung an einen PHY.

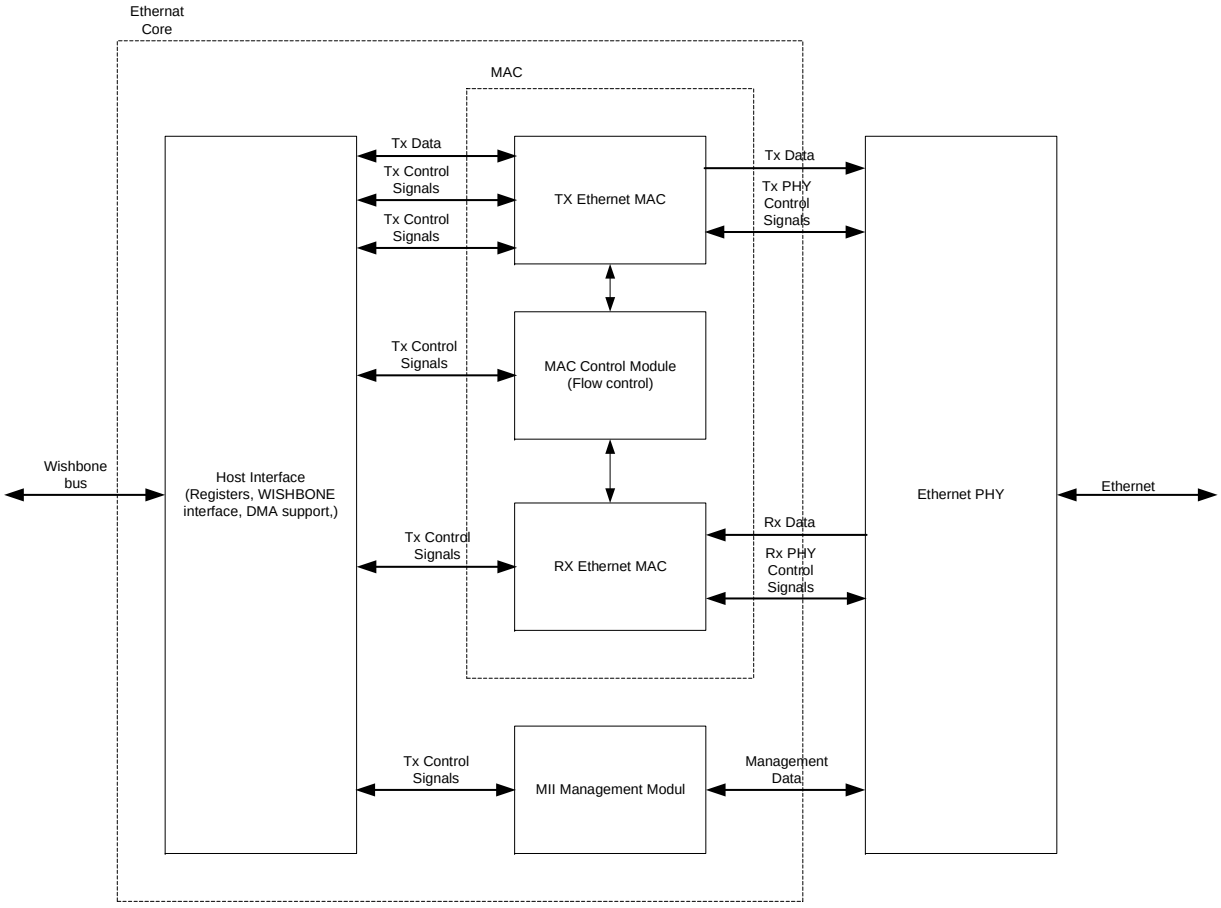

Abbildung 5.9: Aufbau des Ethernet MACs

Die Konfiguration und Verwendung des MACs erfolgt durch 21 Register zu je 32 Bit, welche im Adressbereich des MACs die Adressen von 0x00-0x53 einnehmen. Mithilfe dieser Register können grundlegende Einstellungen getroffen werden wie z. B. die Wahl des Betriebsmodus (Duplexmodus, Übertragungsrate, MAC-Adresse, Interruptgenerierung usw..), es können aber auch Statusinformationen ausgelesen werden, sowie Aktionen wie z. B. der Versand eines Frames initialisiert werden.

Bufferdeskriptoren

Ein wichtiges Konzept beim Empfang und Versand von Frames sind Bufferdeskriptoren (BDs). Man unterscheidet Transmit Descriptors (TxD), welche zum Versand vom Frames verwendet werden und Receive Descriptors (RxD), welche zum Empfang von Frames verwendet werden. Jeder Deskriptor ist 64 Bit groß, wobei die ersten 32 Bit eine Länge und Statusinformation speichern und die letzten 32 Bit einen Zeiger auf die Daten speichern. Abbildung 5.10 zeigt den grundsätzlichen Aufbau eines Deskriptors, wobei die Statusflags von TxDs und RxDs unterschiedlich aufgebaut sind.

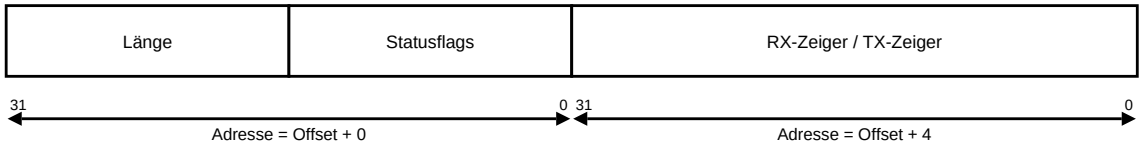

Abbildung 5.10: Bufferdeskriptoren des Ethernet MACs

Die Bufferdeskriptoren werden im Adressbereich von 0x400 bis 0x7FF gespeichert, sodass bis zu 128 BDs verwendet werden können. Die Anzahl von TxDs und RxDs ist über TX_BD_NUM parametrierbar, welches die Anzahl der Schreibdeskriptoren festlegt, sodass sich die Anzahl der Lesedeskriptoren als Differenz zur Gesamtzahl von 128 ergibt. Bei gleicher Aufteilung zu je 64 Deskriptoren (TxD, RxD) hält der Speicherbereich von 0x400-0x5FF die Schreibdeskriptoren und 0x600-0x7FF die Lesedeskriptoren.

Wie erfolgt nun der Versand und Empfang von Daten? Zum Versenden eines Frames müssen die Daten des Frames im Speicher abgelegt werden und der TxD für diese Daten konfiguriert werden. Es muss also die korrekte Länge, die gewünschten Übertragungsflags gesetzt werden sowie der Zeiger auf den Beginn der Daten gesetzt werden. Zum Start des Frame-Versands wird das TXEN-Bit (Register 0, Bit 1) gesetzt. Sobald dies geschehen ist, beginnt der MAC von der im BD abgelegten Adresse zu lesen und füllt seinen FIFO-Speicher. Wenn dieser voll ist, beginnt die Übertragung. Nach der Übertragung wird der Übertragungsstatus in den zugehörigen BD geschrieben und gegebenenfalls ein Interrupt generiert. Sollte im aktuellen BD das WR bit (Bit 13 des ersten Doppelworts des BDs) gesetzt sein, so erfolgt ein Umbruch des Deskriptors und der nächste Deskriptor wird geladen und ausgeführt.

Ähnlich ist die Situation für den Empfang von Frames. Der Empfangsdeskriptor für das zu empfangende Frame muss gesetzt sein und als leer markiert sein. Zudem muss das Empfangsmodul durch Setzen des RECEN Bits (Register 0, Bit 0) eingeschaltet sein. Der MAC empfängt das Frame und sichert es in seinem Speicher und generiert gegebenenfalls einen Interrupt. Der Status beim Empfang wird in den betreffenden BD geschrieben und anschließend wird der nächste BD geladen. Sofern dieser BD ebenfalls als leer markiert markiert ist, kann das nächste Frame empfangen werden.

5.5 Verbindungslogik

Die WISHBONE Interconnection Architecture definiert zwei Typen von Interfaces: Master- und Slave-Interfaces. Master-Interfaces können eine Verbindung zu einem Slave-Interface aufbauen und somit Daten austauschen. Slave-Interfaces können nur dann Daten austauschen, wenn diese durch einen Master adressiert werden, aber sie können selbst keinen Kommunikationspfad aufbauen. Die Verbindung aller Master und Slaves wird über die Verbindungslogik (glue-logic) namens *INTERCON* hergestellt. Die Wahl der Art der Verbindungslogik überlässt WISHBONE dem Systemdesigner. Gemäß Spezifikation [11] gibt es vier vordefinierte Typen, die in den Abbildungen 5.11, 5.12, 5.13 und 5.14 gezeigt werden.

- Point-to-point: Die Punkt-zu-Punkt-Verbindung ist die einfachste Form der Verbindung eines Master- und eines Slave-Interfaces. Hierbei werden die Ausgangssignale des Masters mit den korrespondierenden Eingängen des Slaves verbunden und umgekehrt. Diese Struktur ist nur für die Verbindung eines Master- und eines Slave-Interfaces geeignet.
- Data flow: Die Datenflußverbindung wird eingesetzt, wenn Daten sequentiell bearbeitet werden. Die Daten werden von jedem IP-Core nach der Bearbeitung in den nachgeschalteten weitergegeben, sodass eine Art Kette entsteht. Jeder IP-Core benötigt daher sowohl ein Master- als auch ein Slave-Interface.
- Shared bus: Ein gemeinsamer Bus ermöglicht es zwei oder mehrere Master mit einem oder mehreren Slaves zu verbinden. Ein Arbiter ist nötig, um den Zugriff auf den Bus zu regeln, wenn mehrere Master gleichzeitig den Bus für sich beanspruchen. Ein dafür übliches Arbitrierungsschema ist Round-Robin. Die Implementierung eines gemeinsamen Busses ist entweder durch Multiplexer oder tri-state-Busse möglich, wobei die Entscheidung welche Implementierung günstiger ist vor allem von der verwendeten Hardware abhängt. PCI verwendet vergleichsweise ebenfalls einen gemeinsamen Bus, wobei hier tri-state-Signale zur Anwendung kommen.

  Der Vorteil des gemeinsamen Busses ist vor allem die Kompaktheit der Verbindungslogik; es werden generell relativ wenig logische Elemente benötigt. Jedoch ist durch die Verwendung eines gemeinsamen Busses die mögliche Geschwindigkeit im Vergleich zu einem Kreuzschienenverteiler gering.
- Crossbar switch: Der Kreuzschienenverteiler ermöglicht es mehrere Verbindungspaare zwischen Master und Slave gleichzeitig aufzubauen d. h. es können gleichzeitig mehrere Master mit mehreren Slaves verbunden sein. Dieses Konzept macht daher erst ab mindestens zwei Master und zwei Slaves Sinn, da ein Kreuzschienenverteiler mit nur einem Kanal keinen Vorteil zum gemeinsamen Bus hat. Jeder Master fordert beim Arbiter einen Kanal zu einem Slave an. Sobald die Verbindung aufgebaut ist, läuft über diesen Kanal die Übertragung unabhängig davon, wie die anderen WISHBONE-Interfaces miteinander kommunizieren. Der Arbiter sorgt neben der Zuteilung der Slaves zu den Mastern dafür, dass es zu keiner Monopolisierung von Slaves kommt, wenn zwei Master den Zugriff auf einen gewissen Slave beanspruchen. Abbildung 5.14 zeigt strichliert zwei mögliche Verbindungen.

Dem Vorteil eines hohen Datendurchsatzes und einer geringen Latenzzeit steht der Nachteil des Bedarfs an logischen Elementen und Verbindungsstrukturen am Chip gegenüber (im Vergleich zum gemeinsamen Bus).

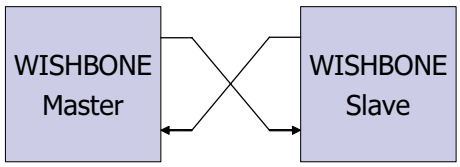

Abbildung 5.11: Point-to-point Interconnection

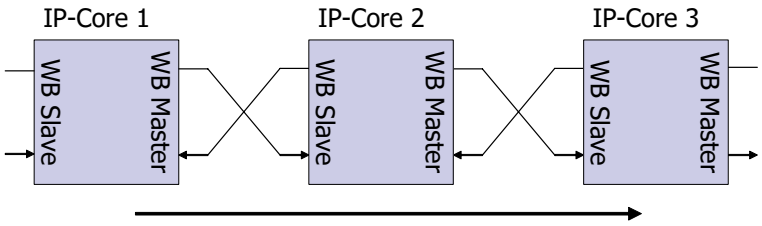

Abbildung 5.12: Dataflow Interconnection

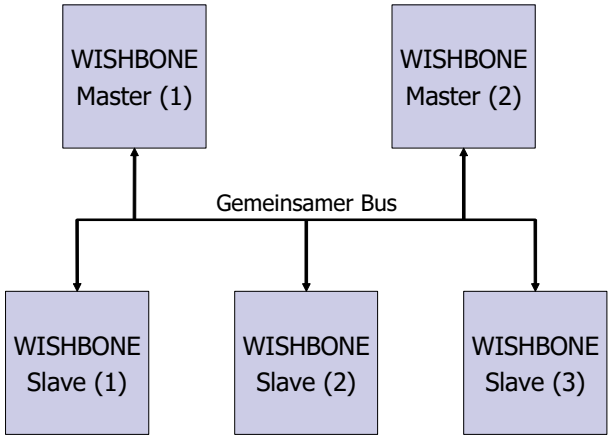

Abbildung 5.13: Shared Bus Interconnection

Die Wahl der Verbindungslogik hängt daher von der Anzahl der Master und Slaves, der Chipfläche und dem nötigen Datendurchsatz ab. Das Design der Netzwerkkarte verfügt über zwei Master-Interfaces (PCI-Master und Ethernet-Master) sowie drei Slave-Interfaces (PCI-Slave, Ethernet-Slave und Syn1588-Slave). Eine Punkt-zu-Punkt-Verbindung scheidet aufgrund der Anzahl der Busteilnehmer generell aus, aber auch eine Datenflußverbindung erscheint als wenig geeignet. Hierfür müssten alle Daten zwischen der PCI-Bridge und dem Ethernet-MAC durch den Syn1588-Core durchgeschleift werden, was vor allem eine Verzögerung zur Folge hätte. Als geeignet erscheint die Verbindung mittels eines gemeinsamen Busses sowie der Verbindung mittels eines Kreuzschienenverteilers.

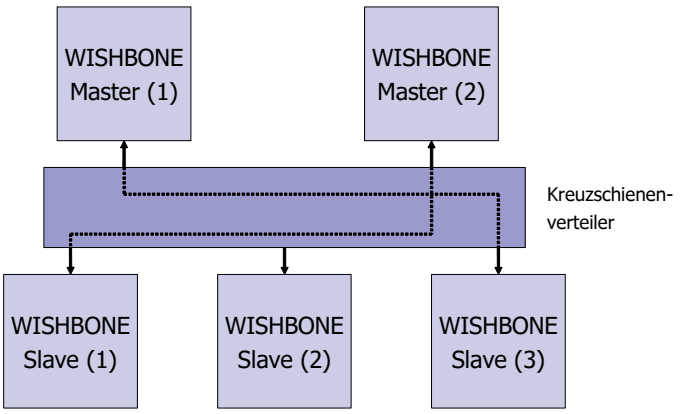

Abbildung 5.14: Crossbar Interconnection

Es wurde ein Kreuzschienenverteiler zur Verbindung gewählt, da dieser den optimalen Datendurchsatz bietet und bei der Einbindung von lediglich drei Mastern und zwei Slaves nur geringfügig mehr Transistoren beansprucht als ein gemeinsamer Bus.

5.5.1 WISHBONE-Builder

Das Erstellen einer Verbindungslogik zwischen IP-Cores ist eine Aufgabe, die jeder Entwickler durchzuführen hat, der mehrere IP-Cores verwendet. Diese relativ fehleranfällige Arbeit kann mithilfe von Softwarewerkzeugen wesentlich vereinfacht werden, da nun nicht mehr die einzelnen Signale miteinander verschaltet werden müssen, sondern getestete Software diese Aufgabe übernimmt, die mit geeigneten Parametern eingestellt werden muss.

Für die WISHBONE-Architektur gibt es zur Erstellung derartiger Verbindungsstrukturen ein Perl-Programm namens *WISHBONE Builder* von Michael Unneback [26]. Dieses Tool ermöglicht die Erstellung einer Verbindungslogik sowohl von Typ *Gemeinsamer Bus* als auch vom Typ *Kreuzschienenverteiler*. Es sind zahlreiche Anpassungsmöglichkeiten vorgesehen, sodass sowohl Adaptionen an die verwendete Hardware durchgeführt werden können (z. B. Anpassungen an FPGAs der Firmen XILINX und Altera), als auch an die Art des WISHBONE-Interfaces (z. B. Breite des Busses, Verwendung von Tags, Unterstützung von speziellen Steuersignalen usw.).

Zur weiteren Einsparung von Leitungen werden WISHBONE-Master in drei Arten von Typen unterteilt:

- Read Only: Der Master liest nur. Dies erspart die Verbindung von WE_O und DAT_O.
- Write Only: Der Master schreibt nur. Die Implementierung von DAT_I ist nicht nötig.
- Read/Write: Der Master schreibt und liest. Alle Signale sind vorhanden.

Sowohl der WISHBONE-Master der PCI-Bridge als auch der des Ethernet MACs müssen Schreib- und Lesebuszyklen durchführen können, sodass diese Optimierung nicht zur Anwendung kommen kann.

Prioritäten

Der Arbiter basiert auf dem Round-Robin-Schema, welches mithilfe von Prioritäten beeinflusst werden kann. Bei einer gegebenen Anzahl von Mastern m und einer Anzahl von Slave n wird jedem Master M_m gegenüber jedem Slave S_n eine Priorität p_{nm} zugeordnet, welche eine garantierte Mindestanzahl von Buszyklen festlegt, bevor der Master vom Slave durch den Arbiter getrennt werden kann. Nicht definierte Prioritäten werden auf eins gesetzt. Jeder Master erhält damit, einen konkreten Slave betreffend, eine durch die Priorität definierte Anzahl von Buszyklen aus der Gesamtzahl der Prioritätsbuszyklen aller Master. Vereinfacht gesprochen bedeutet dies, dass der Master über die Priorität festlegt, wie viele Zeitscheiben im Round-Robin-Schema er beansprucht. Ein Master mit Priorität 4 bekommt die vierfache Übertragungszeit zugeteilt, als ein Master mit Priorität 1. Eine Priorität 0 bedeutet, dass keine Zuteilung erfolgt. Die Anzahl der gleich großen Zeitscheiben ergibt sich aus der Summe der Prioritäten aller Master. Der Master erhält bei maximaler Konkurrenz mit anderen Mastern daher mindestens η_{nm} der Zeit eines Slaves.

$$\eta_{nm} = \frac{p_{nm}}{\sum_{i=1}^{i=m} p_{ni}} \tag{5.1}$$

Master, die nicht den Zugriff auf einen gewissen Slave beanspruchen, werden im Round-Robin-Schema übersprungen. Dadurch ist η_{nm} auch bei einer großen Anzahl an Mastern wesentlich höher als im worst case-Fall. Abbildung 5.12 zeigt die Vergabe der Prioritäten.

Tabelle 5.12: Prioritätenvergabe für Kreuzschienenverteiler

	PCI Slave	Ethernet Slave	Syn1588 Slave
PCI Master	0	1	1
Ethernet Master	1	0	0

Adressen

Die Unterscheidung, ob nun die PCI-Bridge auf den Ethernet MAC oder den Syn1588-Core zugreift, erfolgt anhand der Adresse. Jedes Slave-Interface benötigt einen definierten Adressbereich, der gegenüber anderen Slave überlappungsfrei ist. Damit ist gemeint, dass keine Adresse existieren darf, unter welcher mehrere Slaves angesprochen werden könnten. Grundsätzlich steht es dem Entwickler frei seine Basisadresse zu wählen, OpenCores definiert jedoch eine nicht verbindliche Zuordnung von Adressbereichen zu gewissen Funktionsgruppen. Damit wird ein Vorschlag gemacht welcher Adressbereich z. B. für einen IDE-Controller oder einen Audio-Controller verwendet werden sollte. Abbildung 5.13 zeigt die betreffenden Adressbereiche.

Der Syn1588-Core benötigt einen 10 Bit-breiten Adressbereich, sodass als Basisadresse 0x9201000 gewählt wurde und die Größe auf 0x400 festgesetzt wurde. Die Basisadresse des Ethernet MACs wurde auf 0x9200000 gesetzt, die Größe auf 0x1000 entsprechend den 12 Bits der Breite des Adressbus-

Tabelle 5.13: Definierte Adressbereiche für WISHBONE

Startadresse	Endadresse	Größe (MByte)	Typ
0x9200000	0x93FFFFF	16	Ethernet-Controller
0x8000000	0x8FFFFFF	256	PCI-Controller

ses. Für die PCI-Bridge wurde der komplette Bereich mit der Basisadresse 0x8000000 und einer Größe von 0x1000000 reserviert.

Möchte nun ein WISHBONE-Master auf einen Slave zugreifen, so legt er eine Adresse im Adressbereich des Slaves an seinen Adressausgang ADR_O und setzt CYC_O, um den Slave anzufordern. Der Arbiter in der Verbindungslogik erkennt anhand der Adresse, welcher Slave verbunden werden soll. Sofern der gewünschte Slave keinen aufrechten Buszyklus mit einem anderen Master durchführt, wird der Master im nächsten Taktzyklus verbunden. Wird der adressierte Slave jedoch momentan von einem anderen Master verwendet, so trennt der Arbiter den letzten Master und verbindet ihn mit dem aktuellen Master, jedoch unter Einhaltung der Mindestanzahl von Buszyklen, welche durch die Prioritäten festgelegt sind.

5.6 Syn1588-Core

Die PCI-Bridge zusammen mit dem Ethernet MAC bildet eine gewöhnliche Netzwerkkarte. Der Syn1588-IP-Core ergänzt das Design mit der wesentlichen Fähigkeit hochgenaue Uhrensynchronisation durchführen zu können. Für eine korrekte Kommunikation mit der Verbindungslogik wird hier ebenfalls ein WISHBONE-Interface benötigt. In der vorliegenden Form verfügt der Syn1588-Core jedoch über ein AHB-Interface, welches zu WISHBONE nicht kompatibel ist. Es gibt daher zwei Möglichkeiten eine Verbindung mit den WISHBONE-Komponenten herzustellen:

- Änderung des AHB-Interfaces auf ein WISHBONE-Interface
- Umwandlung der AHB-Signale in WISHBONE-Signale

Eine Änderung des Interfaces bedingt, dass der Quellcode des Hardwaredesigns, zumindest jedoch das AHB-Interfaces in einer Form vorliegt, die eine Änderung ermöglicht. Für IP-Cores vom Fremdherstellern liegt in der Regel kein Quellcode vor, sodass eine interne Änderung nicht möglich ist. Da der Syn1588-Core als Sourcecode in VHDL vorliegt, ist eine Änderung der Schnittstelle prinzipiell möglich. Die Implementierung eines WISHBONE-Interfaces bietet den Vorteil, dass direkt – sprich ohne Umweg über einen Umsetzer – auf den IP-Core zugegriffen werden kann.

Eine Umwandlung der AHB-Signale in WISHBONE-Signale kann bei jedem IP-Core durchgeführt werden, weil dafür kein Eingriff in den IP-Core selbst nötig ist. Die AHB-Signale werden mit einem so genannten Wrapper bzw. einer Bridge verbunden, welche die Umsetzung von AHB auf WISHBONE durchführt, sodass ein IP-Core mit AHB-Interface durch Verwendung der Bridge mit WISHBONE-kompatiblen Signalen adressiert werden kann. Eine Änderung des IP-Cores z. B. durch Implementierung von Zusatzfeatures in einer neueren Version führt zu keiner Änderung der Bridge, solange sich das Interface nicht ändert. Es ist davon auszugehen, dass das AHB-Interface dem

Standard konform bleibt, jedoch muss eine derartige Bridge alle Busmodi sowohl von WISHBONE als auch AHB korrekt umsetzen können, damit man eine derartige Bridge allgemein mit jedem WISHBONE- und jedem AHB-Gerät verwenden kann.

Die Verwendung einer Bridge erscheint in diesem Fall als günstiger. Denn nur so ist sichergestellt, dass für neuere Versionen des Syn1588-Core nicht stets wieder eine Portierungsarbeit erledigt werden muss. Zudem kommt hinzu, dass viele Signale des Syn1588-Cores die Bezeichnung für den AHB-Bus tragen. Man müsste also entweder in allen Modulen, welche über AHB adressiert werden können, die Namen der Signale ändern oder Signale an das WISHBONE-Interface anbinden, welche intern fälschlicherweise noch immer den Namen der ehemaligen AHB-Signale tragen. Jede Form der Umsetzung zwischen zwei Interface-Standards verbraucht, wenn diese nicht ausschließlich in Form vom kombinatorischer Logik möglich ist, ein oder mehrere Taktzyklen. Dies erhöht die Zugriffszeit und senkt die mögliche Übertragungsgeschwindigkeit. Der Syn1588-Core hat jedoch hinsichtlich Geschwindigkeit keine hohen Anforderungen, da die Datenübertragung neben der Konfiguration zum Auslesen und Schreiben von Zeitstempeln dient, welche nur einen geringen Prozentsatz an der Gesamtkommunikation der Netzwerkkarte benötigt.

5.6.1 Aufbau des Syn1588-Cores

Der Syn1588-Core besteht aus mehreren Modulen, die es ermöglichen eine hochgenaue Uhrensynchronisation durchzuführen [16]:

- AHB-Interface (ahbsifc)
- Event Trigger and Timestamp Unit (etrig)
- Adder Based Clock Core (abccore)
- Adder Based Clock Control Unit (abcctrl)
- Accurency Counter (accn)
- Dual Port and FIFO memory (dpram, tsfifo)

Die Architektur des Cores ist allgemein für beliebige paketbasierte Netze einsetzbar, wobei die Generierung von Zeitstempeln von außen durch das Anlegen verschiedener Eingangssignale initialisiert werden kann. Zu diesem Zweck ist je nach Netzwerk eine angepasste Logik notwendig, die nahe an der Bitübertragungsschicht den Beginn eines Pakets erkennt und an den Syn1588-Core signalisiert. Für Ethernet mit 10 und 100 MBit/s kommt ein MII-Scanner zum Einsatz, welcher das MII-Interface zwischen MAC und PHY abhört und den Beginn eines Frames anhand des Start-of-Frame-Delimiters erkennt. Eine weitere Aufgabe des MII-Scanners ist es Synchronisationspakete anhand ihres Aufbaus zu erkennen und das Auftreten solcher Frames an den Syn1588-Core getrennt zu melden.

Der Syn1588-Core verfügt über mehrere Eingangssignale, die den Vorgang der Zeitstempelung bei deren steigender Taktflanke auslösen können. Die Signale framercv_i (frame receive) und framesnd_i (frame send) werden durch den MII-Scanner beim Empfang und Versenden eines Ethernet-Frames gesetzt und teilen dem Syn1588-Core mit, dass ein Zeitstempel für ein gewöhnliches Frame

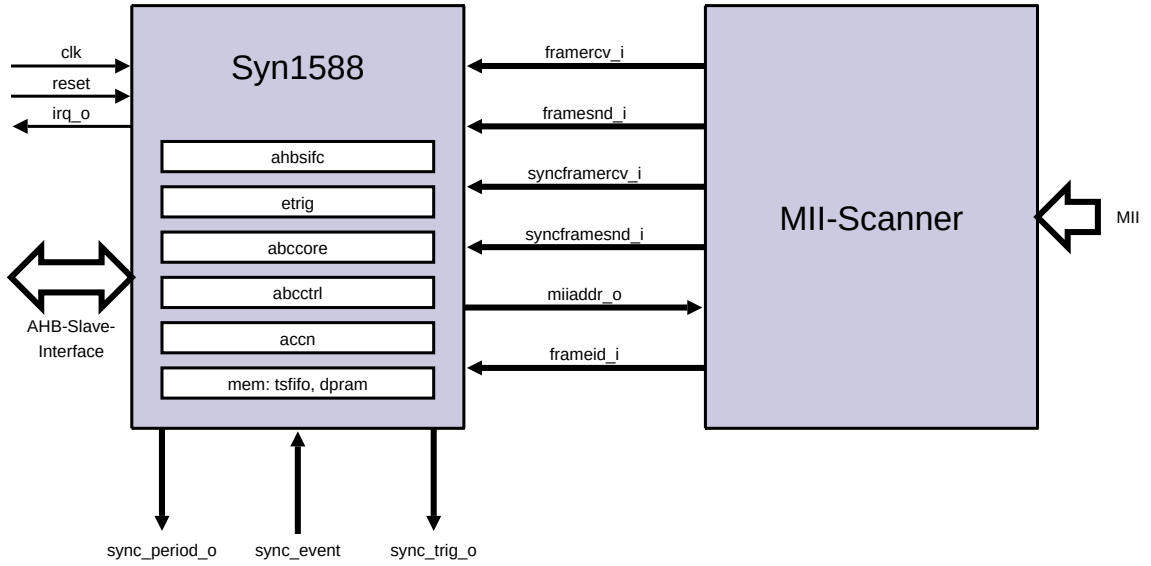

Abbildung 5.15: Aufbau des Syn1588-Core in Verbindung mit dem MII-Scanner

generiert werden soll. Eine ähnliche Aufgabe kommt den Signalen syncframer_i (sync packet receive) und synframes_i (sync packet send) zu. Diese werden dann gesetzt, wenn ein Synchronisationspaket empfangen oder gesendet wurde. Daneben verfügt der Syn1588-Core über das sync_event-Signal mit dem auch für externe Signale ein Zeitstempel generiert werden sowie das sync_trig_o-Signal mit dem externe Events getriggert werden können. Sync_period ist der Ausgang des Periodentimers und mittels irq_o können – je nach Konfiguration – Interrupts z. B. beim Auftreten eines Synchronisationspakets generiert werden. Abbildung 5.15 zeigt den Syn1588-Core in Verbindung mit dem MII-Scanner.

Bei der Übermittlung von Ethernet-Frames kann es zu Übertragungsfehlern kommen. Die Zeitstempelung findet zu Beginn jedes Frames anhand des SFDs statt, sodass auch Zeitstempel für fehlerhafte Frames generiert werden können. Ob nun ein Frame tatsächlich korrekt übertragen wurde, kann nämlich erst nach dem vollständigen Empfang des Frames sowie Überprüfung der Frame Check Sum festgestellt werden. Fehlerhafte Frames werden in der Regel vom MAC verworfen, sodass ein Zeitstempel für ein nicht mehr vorhandenes Frame vorhanden wäre. Dies würde die Zuordnung der Zeitstempel zu den korrespondierenden Frames stören. Man umgeht dieses Problem durch die Verwendung von FrameIDs, die gemeinsam mit dem Zeitstempel abgespeichert werden und dem Treiber eine korrekte Zuordung des Frames zu seinem Zeitstempel ermöglichen. Der MII-Scanner vergibt für jedes Frame eine ID und überträgt diese über das frameid_i-Signal an den Syn1588-Core.

5.6.2 Funktion des AMBA-AHB-Busses

Der Syn1588-Core verfügt über ein AHB-Interface, welches auf WISHBONE umgesetzt werden soll. An dieser Stelle soll eine kurze Einführung in das AHB-Bussystem gegeben werden. Der Chipher-

steller ARM spezifiziert drei Bustypen in der AMBA (Advanced Microcontroller Bus Architecture)-Spezifikation [3] für unterschiedliche Anforderungsprofile, wobei hier der Bustyp AHB (Advanced High-performance Bus) betrachtet werden soll.

Die Spezifikation verwendet eine eigene Terminologie zur Beschreibung der Bustransaktionen. So wird eine Anzahl zusammenhängender Bustransaktionen *Burst* genannt. Ein Burst besteht aus einer oder mehreren Einzeltransaktionen, welche *Beats* genannt werden. Die Beats einer Burst-Operation hängen durch ihre Adresse zusammen. Die Adressen aufeinander folgender Beats werden entweder mit einer konstanten Zahl erhöht oder können auch umbrechen (*wrapping*). Unter umbrechen versteht man, dass die Adressen nach Überschreiten der Obergrenze wieder an die Untergrenze gesetzt werden, sodass ein wiederholtes Beschreiben oder Lesen eines Adressbereichs erreicht wird. Innerhalb eines Bursts kann jedoch nur ein Umbruch stattfinden. Ob und wann der Umbruch stattfindet wird durch das Signal hburst und die Anzahl der Beats festgelegt.

Das AHB-Bussystem besteht – wie auch die WISHBONE-Architektur – aus dem Adressbus, dem Datenbus sowie einer Anzahl von Steuersignalen, die für das Handshaking zwischen Master und Slave sowie zur Arbitrierung dienen. AHB verwendet wie WISHBONE zwei Datenbusse; einen für die zu schreibenden und einen für die zu lesenden Daten. Alle Signale beginnen mit den Präfix H, was ihre Zuordnung zum AHB-Bus kennzeichnet. Generell sind alle Signale *active high*, es sei denn diese sind durch ein *n* gekennzeichnet, was auf ein active low-Signale hinweist (z. B. HRESETn). Einige Signale verfügen über eine direkte Verbindung eines Masters bzw. Slaves mit dem Arbiter oder Decoder, um z. B. die Arbitrierung zu ermöglichen. Diese Signale enden mit dem Suffix *x*, wobei dieses Zeichen repräsentiv für eine Zahl oder Zeichenkette steht. Tabelle 5.14 listet alle AHB-Signale auf.

Ein typisches AHB-System verfügt über folgende Busteilnehmer:

- AHB-Master: Der Master beginnt die Bustransaktion.
- AHB-Slave: Der Slave antwortet auf eine vom Master gestartete Transaktion.
- AHB-Arbiter: Der Arbiter verwaltet den Bus.
- AHB-Decoder: Der Decoder wird verwendet, um die Adresse jedes Transfers zu dekodieren.

Das Verbindungsschema des AHB-Busses basiert auf einer Multiplexerstruktur, sodass keine Tri-State-Signale zum Einsatz kommen. Eine typische Anordnung mehrerer AHB-Master und AHB-Slaves zeigt Abbildung 5.16 [vgl. [3]] . Neben den Mastern und den Slaves wird ein Arbiter verwendet, der den Multiplexer der Daten- und Kontrollsignale sowie den Multiplexer der zu schreibenden Daten steuert. Der Multiplexer der zu lesenden Daten wird von einem Decoder gesteuert.

Wie werden Master und Slave miteinander verbunden? Bevor der Master einen Zugriff auf den Bus erhält, muss er beim Arbiter den Zugriff auf den Bus anfordern, indem er das HBUSREQ-Signal setzt. Sobald alle notwendigen Bedingungen erfüllt sind, und der aktuelle Burst (sofern gerade einer ausgeführt wird) abgeschlossen ist, erhält der Master vom Arbiter durch Setzen von HGRANT den Zugriff auf den Bus. Der *Address and Control Mux* sowie der *Write Data Mux* werden vom Arbiter so konfiguriert, dass die Signale des arbitrierten Masters zu den Slaves durchgeschaltet werden. Sobald der letzte Burst eines anderen Masters abgeschlossen ist, was der Master dadurch erkennt,

Tabelle 5.14: Signale des AHB-Bussystems

Signalname	Quelle	Bitbreite	Kurzbeschreibung
HCLK	Taktgenerator	1	Bustaktsignal
HRESETn	Resetcontroller	1	Resetsignal
HADDR	Master	32	Adressbus
HTRANS	Master	2	Übertragungstyp: IDLE, BUSY, NONSEQ, SEQ
HWRITE	Master	1	Schreibsignal: HIGH=schreiben, LOW=lesen
HSIZE	Master	3	Datenmenge je Beat: 8, 16, 32, 64, 128, 256, 512 oder 1024 Bit
HBURST	Master	3	Anzahl der Beats eines Burst sowie Festlegung, ob umbrechend oder nicht umbrechend: SINGLE, INCR, WRAP4, INCR4, WRAP8, INCR8, WRAP16 oder INCR16
HPROT	Master	4	Zusätzliche Information zu einem Burst
HSELx	Decoder	1	Slave-Auswahl
HRDATA	Slave	32	Lesedaten
HWDATA	Master	32	Schreibdaten
HREADY	Slave	1	Bestätigung des Beats
HRESP	Slave	2	Statusantwort des Slaves an den Master: OKAY, ERROR, RETRY, SPLIT
HBUSREQx	Master	1	Anforderung des Busses beim Arbiter
HLOCKx	Master	1	Anforderung eines exklusiven Buszugriffs beim Arbiter
HGRANTx	Arbiter	1	Buszuteilung an einen Master
HMASTER	Arbiter	4	Nummer des aktuellen Masters an den Slave zur Fortsetzung nach einem SPLIT-Transfer
HMASTLOCK	Arbiter	1	Anzeige eines exklusiven Buszugriffs
HSPLITx	Slave	16	Anforderung, welcher Master nach einem SPLIT die Transaktion abschließen soll

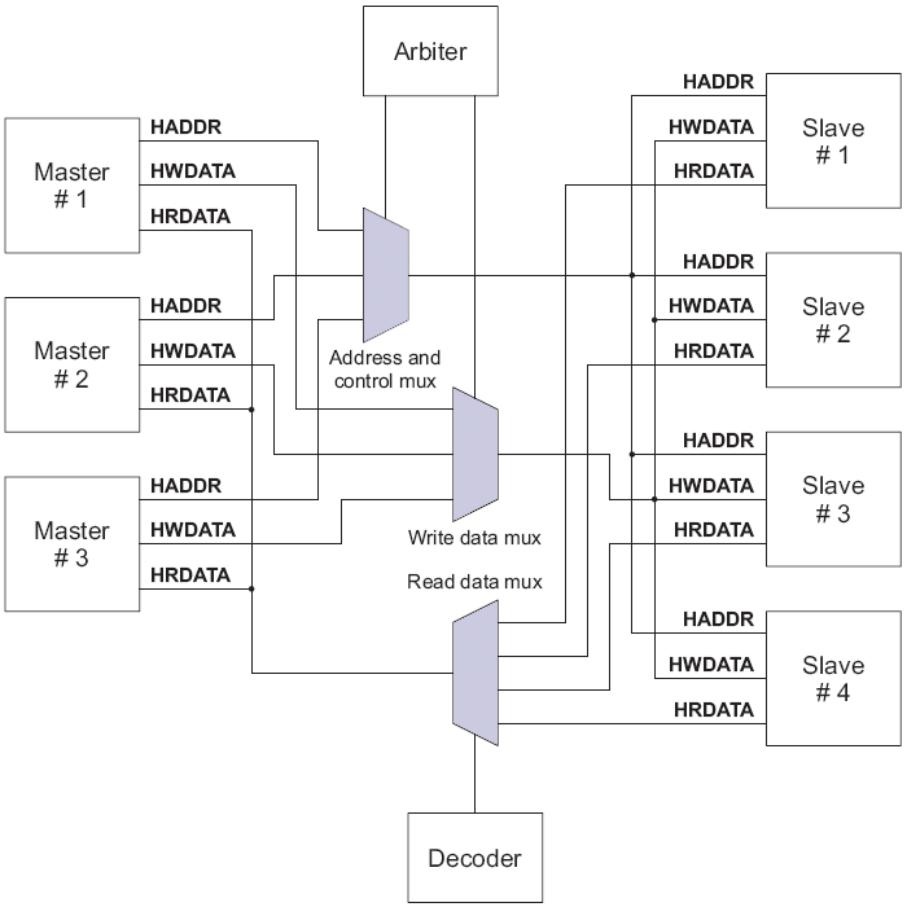

Abbildung 5.16: Verbindungsstruktur eines AHB-Systems

dass der zuletzt ausgewählte Slave den letzten Beat mit HREADY=1 quittiert hat, legt Master seine Steuer- und Adressdaten an den Bus. Der Decoder wählt zugleich anhand der angelegten Adresse den entsprechenden Slave aus, indem er dessen HSEL-Signals setzt. Im folgenden Takt werden zu der bereits übertragenen Adresse die dazugehören Daten übertragen. Für den Fall, dass eine Leseoperation durchgeführt wurde, legt der Slave die gelesenen Daten an HRDATA, für den Fall, dass der Master eine Schreiboperation gestartet hat, legt der Master die zu schreibenden Daten an HWDATA. Gleichzeitig legt der Master die Adress- und Steuerdaten für den nächsten Beat auf den Bus, wodurch eine Überlappung entsteht, das *Adress-Pipelining*. Das Überlappen von Adressen und Daten ermöglicht es dem Slave ausreichend Zeit zu lassen um Leseoperationen zu beantworten. Dieses System erlaubt höhere Taktfrequenzen als ein nicht verschränktes Bussystem, da der längste logische Pfad verkürzt wird und damit die Signallaufzeit sinkt.

Während eines Transfers kann der Slave mit HREADY und HRESP dem Master seinen Status übermitteln. Mit HREADY kann der Slave einen Beat bestätigen oder aber Wartezyklen einfügen. Mittels HRESP kann er Fehler während eines Transfers melden, eine Wiederholung erzwingen sowie in Verbindung mit HREADY den erfolgreichen Abschluss eines Transfers anzeigen. HRESP definiert folgende Zustände:

- OKAY: Dies kennzeichnet, dass die Übertragung fehlerfrei vor sich geht. Ist HREADY gleich eins, während HRESP gleich OKAY ist, so zeigt das dem Master an, dass der Transfer erfolgreich abgeschlossen wurde.
- ERROR: Dieser Zustand zeigt an, dass die Übertragung nicht erfolgreich abgeschlossen wurde.
- RETRY and SPLIT: Dieser Zustand zeigt an, dass die Übertragung nicht erfolgreich beendet wurde, und weist den Master an, eine Wiederholung der Übertragung zu versuchen.

Ein wichtiges Steuersignal des Masters ist HTRANS. Mit diesem Signal kann der Master Bursts starten, Wartezyklen einfügen sowie das Ende eines Bursts anzeigen. HTRANS verfügt über folgende Zustände:

- IDLE: Der IDLE-Zustand zeigt an, dass momentan kein Transfer stattfindet.
- BUSY: Durch Setzen von BUSY kann der Master Wartezyklen innerhalb eines Bursts einfügen.
- NONSEQ: Der Master setzt NONSEQ, um den ersten Transfer eines Bursts anzuzeigen. Adress- und Kontrollsignale haben keinen Zusammenhang mit dem letzten Transfer.
- SEQ: Weitere Transfers nach NONSEQ sind sequentiell. Die anliegenden Daten gehören zu der im letzten Takt angelegten Adresse.

Die Verwendung dieses Signals wird in Abbildung 5.17 gezeigt. Der Master startet den Burst durch Setzen des HTRANS-Signals auf NONSEQ. Dies zeigt an, dass die anliegenden Adress- und Kontrollsignale in keinem Zusammenhang mit vorher gesendeten Signalen stehen, weil die im letzten Takt gesendeten Signale zum Burst eines anderen Masters gehört haben, bzw. während des IDLE-Zustands nicht gültig waren. Zeitgleich mit NONSEQ legt der Master die Adresse A an den Adressbus. Im nächsten Takt erfolgt die Datenübertragung, welche abhängig ist, ob es sich um eine

Lese- oder Schreibtransaktion handelt. Bei einer Lesetransaktion legt der Slave die Daten an HRDATA, welche der Master bei der steigenden Flanke von HCLK einliest. Bei einer Schreibtransaktion erhält der Slave die Daten über HWDATA. Jeder Beat wird durch den Slave bestätigt.

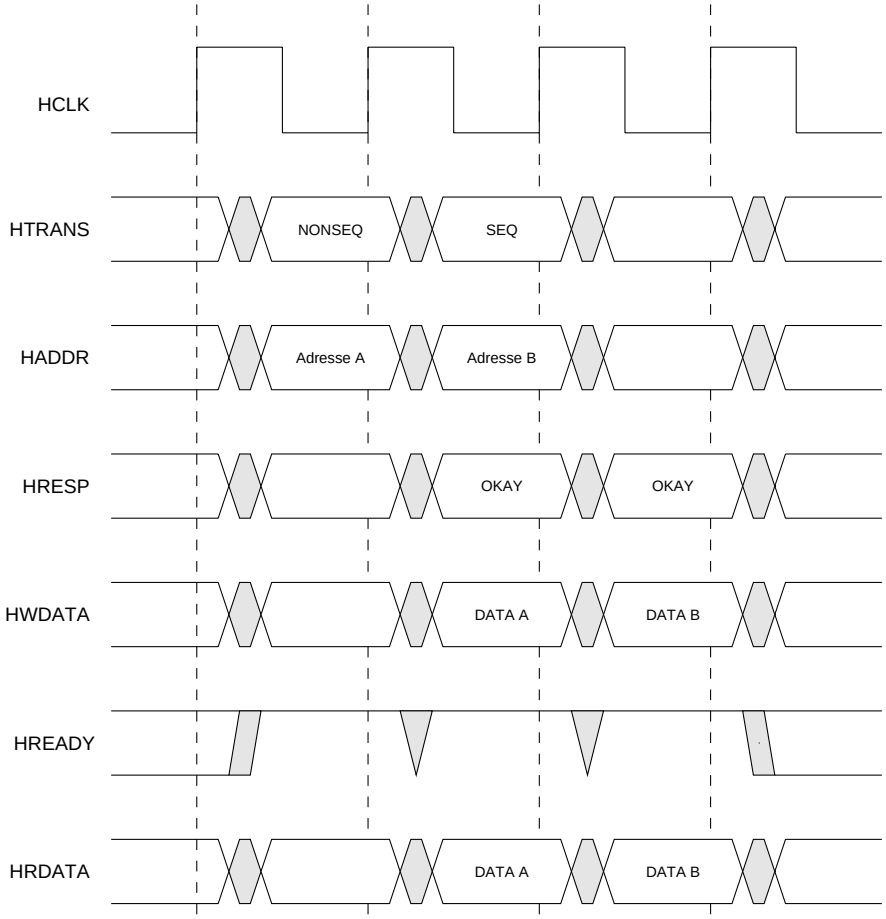

Abbildung 5.17: AHB Burst

Besteht der Burst nur aus einem einzelnen Beat, so setzt der Master HTRANS auf IDLE, um den Burst zu beenden, oder NONSEQ, um den aktuellen Burst zu beenden und einen weiteren Burst zu beginnen. Ein Burst, der aus mehreren Beats besteht, wird nach dem ersten Takt mit NONSEQ durch SEQ in den folgenden Takten fortgesetzt. Dies zeigt an, dass es *Address Pipelining* verwendet wird, also die Lese- und Schreibdaten den Steuer- und Adressdaten mit der Verzögerung um einen Taktzyklus angelegt werden. Abbildung 5.17 zeigt einen Burst, der aus mindestens zwei Beats besteht.

Sowohl Master als auch Slave können Wartezyklen einfügen, wenn diese z. B. nicht in Lage sind, die Daten mit ausreichender Geschwindigkeit zu verarbeiten. Der Master kann HTRANS statt SEQ auf BUSY setzen und somit die Übertragung verzögern. Einen Beat mit einer Verzögerung zu beginnen, ist nicht möglich (BUSY statt NONSEQ). Sobald der Master wieder in der Lage ist einen weiteren Beat zu verarbeiten, kann er HTRANS wieder auf SEQ setzen und dadurch die Übertragung fortsetzen. Der Slave quittiert den Erhalt von BUSY durch das Setzen von HREADY und muss die

aktuellen Adress- und Steuerdaten verwerfen und die im darauf folgenden Takt angelegten Daten an HWDATA ignorieren.

Auch der Slave kann Wartezyklen einfügen, wenn er die Daten in der vorhandenen Zeit nicht verarbeiten kann. Der Slave kann den Beat dadurch verlängern indem er dem Master den Beat nicht sofort durch HREADY quittiert. Das drosselt die Übertragungsgeschwindigkeit. Die Verspätung bei der Annahme der Daten verzögert auch die Adress- und Steuerdaten des nächsten Beats.

Jedem AHB-Slave sind gewisse Adressbereiche zugeordnet, wobei die Mindestgröße 1 KByte beträgt. Der Decoder dekodiert daher die niederwertigsten 10 Bit der Adresse nicht. Fällt die vom Master angelegte Adresse in den Adressbereich eines gewissen Slaves, so wählt der Decoder über HSEL den zum Adressbereich gehörigen Slave aus.

5.6.3 Wishbone-AHB-Bridge

Die WISHBONE-AHB-Bridge führt die Umsetzung des WISHBONE-Interfaces auf AHB durch, sodass AHB-Slaves mit der WISHBONE-Architektur verbunden werden können. Die Verwendung einer Bridge ist in Abbildung 5.18 dargestellt. Die Funktionsweise ist einfach: Ein WISHBONE-Master greift über die Verbindungsstruktur auf das WISHBONE-Slave-Interfaces der Bridge zu. Diese verändert die Steuerdaten in einer geeigneten Art und Weise und gibt die Lese- oder Schreiboperation an das AHB-Master-Interface weiter, welches direkt mit dem AHB-Slave verbunden ist. Die Antwort des AHB-Slaves wird wiederum auf eine dem WISHBONE-Standard entsprechende Form umgesetzt.

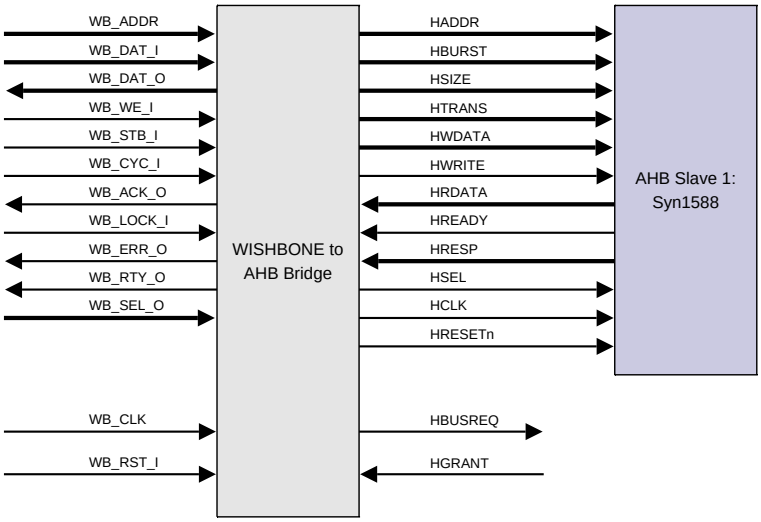

Abbildung 5.18: WISHBONE-AHB-Bridge mit einem AHB-Slave

Schränkt man die Bridge hinsichtlich der Anzahl der AHB-Slaves auf eines einen Slave ein, so entfallen der Lesedatenmultiplexer, der Steuerdatenmultiplexer sowie der Adressdecoder. Arbiter und

Steuerdatenmultiplexer entfallen ohnehin bzw. sind degeneriert, wenn nur ein Master im AHB-Bussystem vorhanden ist. Damit ergibt sich eine einfache direkte Verbindung wie bei der Point-to-Point-Interconnection zweier WISHBONE-Interfaces (vgl. Abb. 5.11).

Da das Konzept derartiger Verbindungsstrukturen in den Grundzügen sehr ähnlich ist, lassen sich viele der Signale direkt verbinden bzw. durch kombinatorische Logik erzeugen. Tabelle 5.15 zeigt die Zuordnung der Signale, wobei auch die Richtung mittels Pfeil angedeutet wird. Ein einfacher Pfeil weist ein Signal zu, ein Doppelpfeil einen Bus. Die Zuordnung von SEL_I und HSIZE erfolgt anhand einer Tabelle. Geht man von einer Granularität von 8 Bit und einer Breite von 32 Bit von DAT_I und DAT_O aus, so ist SEL_I 4 Bit breit (1 Bit je Byte des Datenbusses). Jedes Bit von SEL_I zeigt an, welches Bytes des Datenbusses gültig sind. Der Wert des AHB-Signals HSIZE bezeichnet hingegen wie viele Bytes des Datenbusses gültig sind, beginnend bei niederwertigsten Bit. HSIZE enthält also die Länge der Kette von Einsen von SEL_I. Leicht unterschiedlich ist ebenfalls das Reset-Signal, welches in AHB-Systemen low-aktiv ist. Außerdem ist der Reset-Vorgang – im Gegensatz zu WISHBONE – asynchron zum Takt.

Tabelle 5.15: Kombinatorisch verbundene Signale der WISHBONE-AHB-Bridge

WISHBONE Signal	Richtung	AMBA AHB-Signal
CLK	$\rightarrow$	HCLK
not(RST_I)	$\rightarrow$	HRESETn
ADR_I	$\Rightarrow$	HADDR
WE_I	$\rightarrow$	HWRITE
SEL_I	$\rightarrow$	HSIZE (siehe Text)
DAT_O	$\Rightarrow$	HWDATA when WE_I='1'
DAT_I	$\Leftarrow$	HRDATA when WE_I='0'
CYC_I	$\rightarrow$	HSELx

Der wesentliche Unterschied zwischen WISHBONE und AMBA-AHB besteht im *Address Pipelining* d. h. Adressen und Daten sind im AHB-Bussystem stets um einen Takt versetzt. Der Start jedes Bursts verbraucht einen Taktzyklus, sodass der Overhead bei einer Einzelübertragung mit nur einem Beat recht groß ist. Eine Einzeltransaktion erfordert daher mindestens zwei Taktzyklen. Im Gegenzug verdoppelt das Pipelining die mögliche Verarbeitungszeit für den Slave, sodass im Allgemeinen der AHB-Bus höhere Taktraten ermöglicht.

Das Problem der WISHBONE-AHB-Bridge besteht darin, dass das Address Pipelining nicht verwendet werden kann, und damit im Extremfall die Performance auf die Hälfte schrumpft. Beginnt die Bridge z. B. einen Lesetransfer, so wird im ersten Takt die Adresse an den Bus gelegt, und – sofern der Slave keine Wartezyklen einfügt – im zweiten Takt die Daten vom Datenbus gelesen. In diesem zweiten Takt könnte bereits die Adresse des nächsten Transfers auf den AHB-Adressbus gelegt werden. Diese Adresse ist jedoch nicht bekannt, da diese erst zum Beginn des nächsten Transfers am WISHBONE-Interface anliegt. Ohne eine Kenntnis der zukünftigen Adresse können Lesetransfers nur als Burst mit einem Beat umgesetzt werden.

Für Schreibtransfers gilt im Allgemeinen das Gleiche. Geht man jedoch davon aus, dass jede Schreibtransaktion erfolgreich abgeschlossen werden kann, so könnte man die Schreibtransfers in einem

FIFO-Speicher zwischenspeichern und zeitversetzt, jedoch mit voller Datenrate an den AHB-Bus übertragen. Diese Technik wird *posted write* genannt. Da das AHB-Interface des Syn1588-Cores aber ohnehin keine Burst-Transfers unterstützt, wurde auf die Implementierung dieses Features verzichtet.

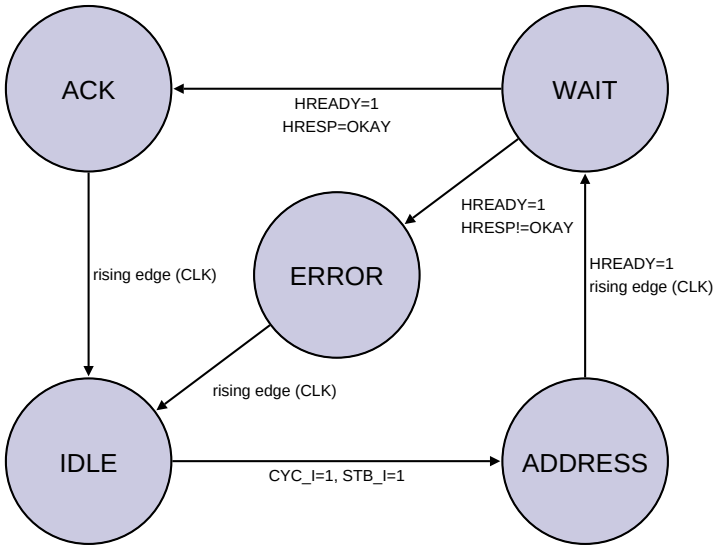

Abbildung 5.19: Finite-State-Machine der WISHBONE-AHB-Bridge

Die Steuersignale werden mithilfe einer Finite-State-Machine (FSM), wie sie in Abbildung 5.19 vereinfacht gezeigt wird, umgesetzt. Es erscheint nicht als günstig, den Übergang von einem Zustand in den nächsten nur an steigenden Taktflanken durchzuführen, da die speichernde Wirkung eines Latches die Umsetzung nur noch weiter verzögern würde. Im Speziellen lässt sich sonst das Acknowledgment-Signal ACK_O erst einen Taktzyklus verspätet setzen, da die steigende Taktflanke von HREADY erst im kommenden Takt übernommen werden könnte. Dieser Aufbau muss allerdings wohl überlegt sein, damit es zu keiner *Race-Condition* kommen kann, bei der der Zustand der FSM davon abhängen würde, welches Signal zuerst an der Bridge eintrifft.

Der Grundzustand ist IDLE. Dieser Zustand wird stets dann eingenommen, wenn entweder ein Resetvorgang durchgeführt wurde (d. h. RST_I bei einer steigenden Taktflanke von CLK gesetzt ist) oder wenn die Bridge inaktiv ist, weil sie von keinem WISHBONE-Master adressiert wird. Für die Inaktivität gilt generell, dass CYC_I=0 ist.

Da kein Address Pipeling verwendet werden kann, können sowohl die Adressen als auch die zu schreibenden Daten (HWDATA) immer am AHB-Bus anliegen ohne die Wirkungsweise des AHB-Busses zu beeinträchtigen. Es widerspricht nicht der AHB-Spezifikation die Signale permanent anzulegen, sie müssen nur zum vereinbarten Zeitpunkt gültig sein. Der Vorgang der Zustandsmaschine wird für einen Lesetransfer auch im Impulsdiagramm 5.20 dargestellt, wobei die WISHBONE-Signale mit dem Präfix WB gekennzeichnet sind und auch die AHB-Signalnamen nach Tabelle 5.15 eingetragen sind.

Der Beginn eines Transfers wird durch Setzen des Cycle Output- und des Strobe-Signals des ver-

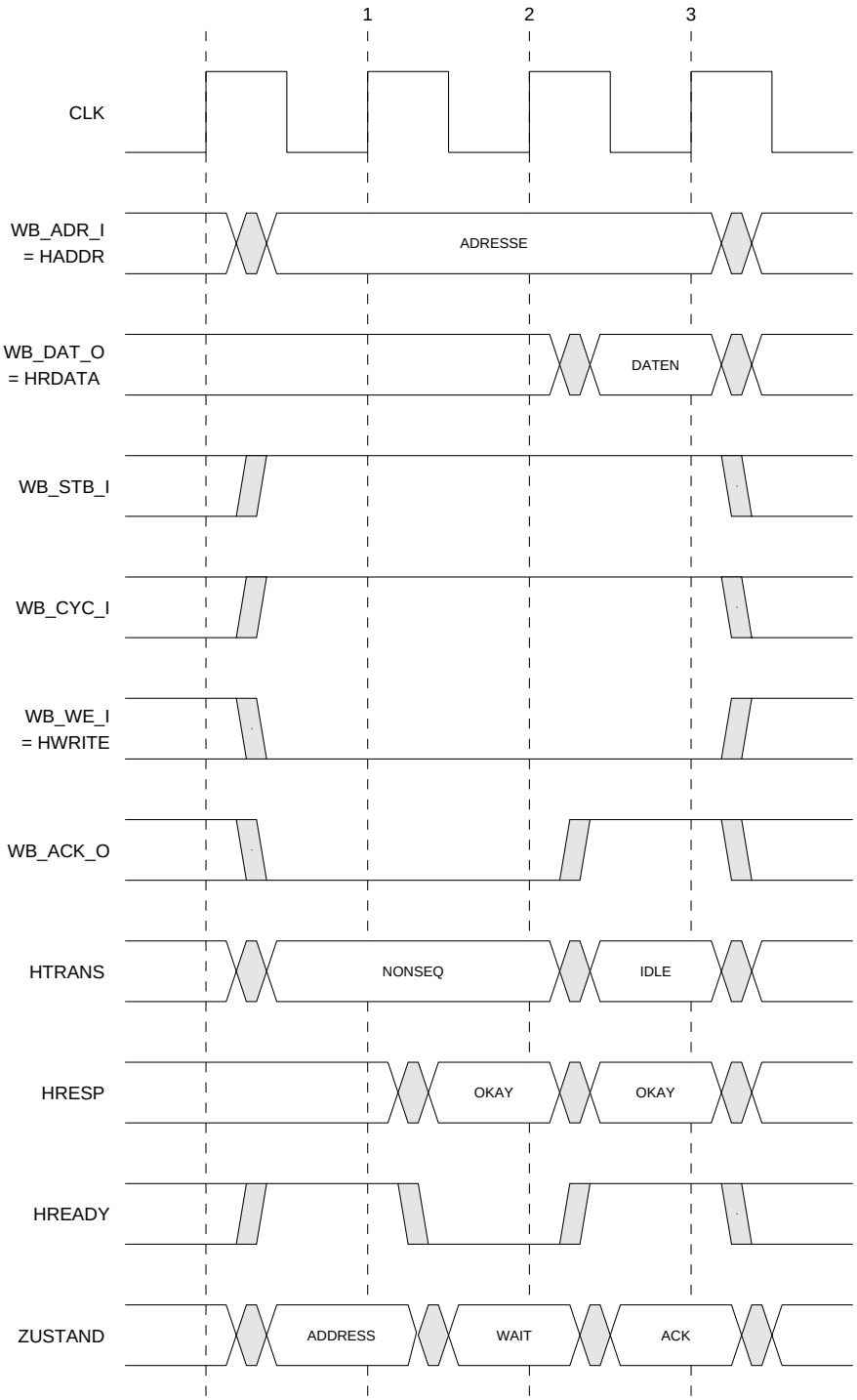

Abbildung 5.20: Impulsdiagramm der WISHBONE-AHB-Bridge

bundenen WISHBONE-Masters initiiert, wodurch die Bridge vom IDLE-Zustand in den ADDRESS-Zustand übergeht. In diesem Zustand müssen die Adresse und die Steuerdaten an den AHB-Bus gelegt werden, wobei die Adresse aus den genannten Gründen stets anliegt. Durch das Setzen von HTRANS auf NONSEQ zeigt die Bridge dem AHB-Slave den Beginn eines Bursts an.

Der Slave holt bei der Taktflanke 1 in Abbildung 5.20 die Adresse vom Adressbus und erkennt anhand des Schreibsignals HWRITE, dass dies eine Leseoperation ist. Dies bestätigt der Slave durch das Setzen von HRESP auf OKAY. Kann der AHB-Slave die zu lesenden Daten bei der Taktflanke 2 bereitstellen, so muss er die Gültigkeit der Daten am Datenbus durch HREADY=1 anzeigen. Wenn nicht, hat er die Möglichkeit eine beliebige Anzahl von Wartezyklen einzufügen. Abbildung 5.20 zeigt, dass der AHB-Slave einen Wartezyklus einfügt. Dieser WAIT-Zustand zum Einfügen von Wartezyklen kann ggf. durchlaufen werden.

Nun wird solange gewartet bis HREADY an einer steigenden Taktflanke von CLK wieder gleich 1 ist, was bei Taktflanke 3 der Fall ist. Der Slave hat die Daten an den Datenbus gelegt und zugleich mittels Response-Code HRESP=OKAY in Verbindung mit HREADY=1 die Korrektheit der Daten bestätigt. Die Bridge erzeugt am WISHBONE-Slave-Interface ebenfalls das Acknowledgment-Signal ACK_O für die Dauer eines Taktes. Sollte der AHB-Slave nicht mit OKAY antworten, wird anstatt des Acknowledgment-Signals das Error-Signal ERR_O generiert. Der AHB-Slave könnte mittels HRESP neben OKAY und ERROR auch mit RETRY oder SPLIT antworten, jedoch ist dies nur für einen Burst mit mehreren Beats spezifiziert, sodass in diesem Fall ebenfalls ein Error erzeugt wird.

Der Aufbau der Zustandsmaschine des AHB-Interfaces im Syn1588-Core bedingt, dass stets mindestens ein Wartezyklus eingefügt wird, maximal jedoch drei. Damit verbraucht ein Einzeltransfer mindestens drei Taktzyklen im besten Fall und fünf Taktzyklen im ungünstigsten Fall.

5.7 Gesamtsystem

Nach der Beschreibung der IP-Cores und deren Verbindungslogik soll hier auf weitere notwendige Logik eingegangen werden.

Die PCI-Bridge ist intern ohne Tristate-Signale aufgebaut, da diese nicht in allen FPGAs effizient implementiert werden können. Das PCI-Bussystem basiert jedoch auf mehrstufigen, teils bidirektionalen Signalen, die – bei Verzicht auf interne Tristate-Signale – externe Bustreiber, sogenannte Buffer, erfordern. Jedes Bit eines bidirektionales Tristate-Signals erfordert aus diesem Grund intern drei Signale wie z. B. das Adress/Datensignal AD():

- Ausgangssignal: AD_out
- Eingangssignal: AD_in
- Output Enable-Steuersignal für den Bustreiber: AD_OE

In Abhängigkeit vom Steuersignal AD_OE kann das Ausgangssignal AD_out an den Adress/Datenbus gelegt werden oder der Adress/Datenbus wird nicht getrieben. Ob nun der Treiber im high- oder low-Zustand aktiv ist, hängt vom verwendeten Bustreiber ab, welcher anhand der eingesetzten

Hardware gewählt werden muss. Aus diesem Grund ist auch die Generierung dieses Enable-Singals für die PCI-Bridge parametrierbar. Hier wurde der vitalbufif0-Buffer aus der VITAL-Bibliothek gewählt. Es handelt sich dabei um einen nicht invertierenden Buffer mit low-aktivem Steuersignal.

Neben der Verbindungslogik wird noch ein einfaches Oder-Gatter für die Verbindung der Interruptleitungen des Ethernet MACs sowie des Syn1588-Cores mit der PCI-Bridge benötigt. Sobald einer der beiden IP-Cores einen Interrupt auslöst (z. B. nach Empfang eines Ethernet-Frames), generiert die PCI-Bridge einen Interrupt am PCI-Bus. Dieser Interrupt veranlasst das Betriebssystem einen Treiberaufruf durchzuführen. Da der Treiber den Verursacher des Interrupts nicht kennt, so hat er die Statusregister aller IP-Cores auszulesen um somit die Quelle des Interrupts zu finden. Abbildung 5.21 zeigt das Gesamtsystem, wobei zusätzliche Signale wie z. B. die Takt- und Resetsignale nicht eingezeichnet sind.

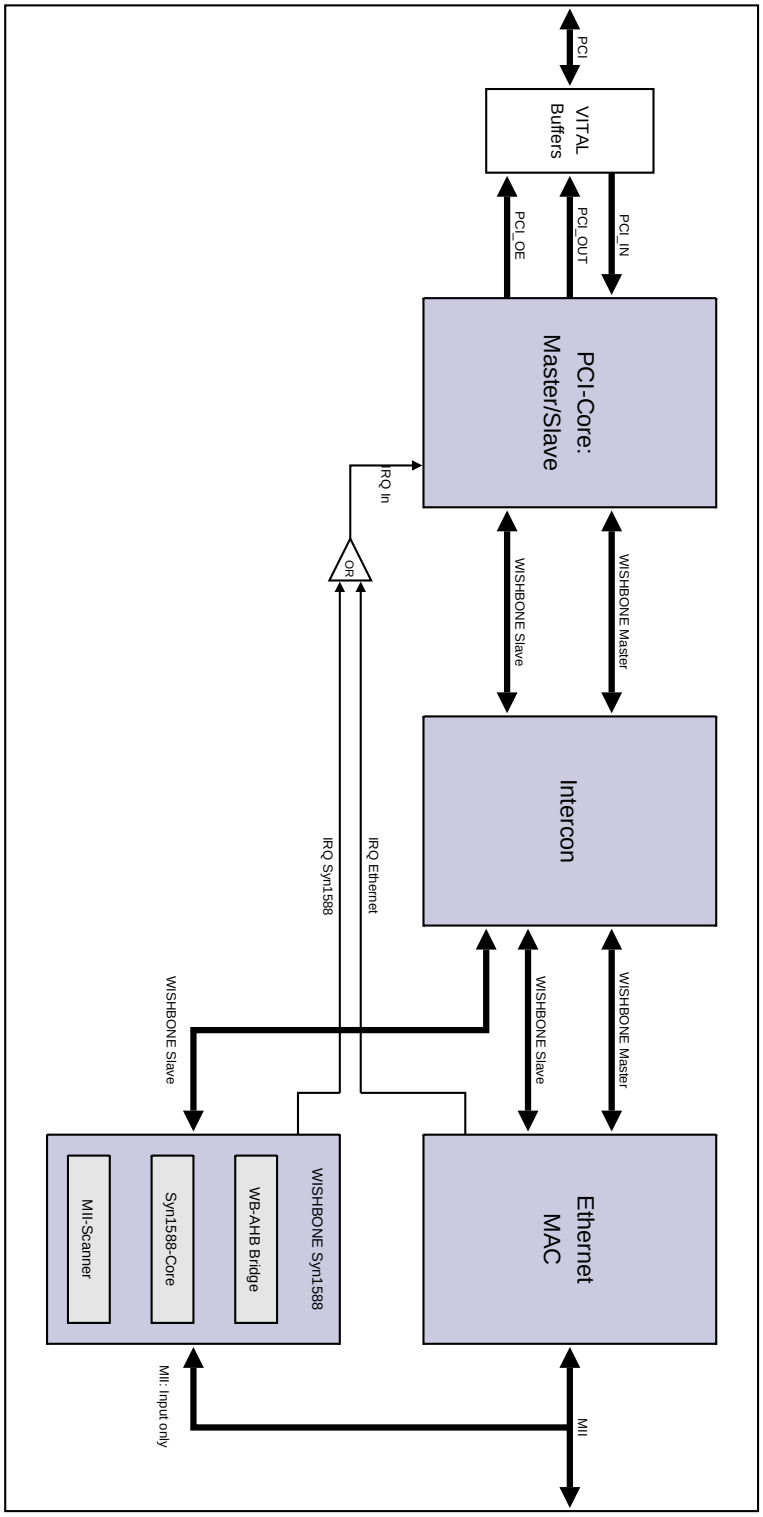

Abbildung 5.21: Aufbau der PCI-Netzwerkkarte

6 Verifikation und Synthese

Um das erstellte System testen zu können, wurde eine Testbench erstellt. Eine Testbench ist eine Simulationsumgebung, die es ermöglicht ein Design auf Korrektheit zu überprüfen. Das zu untersuchende Sytem wird mit einer gewissen Eingangskombination, der sogannten Stimuli, sowie dem Taktsignal angeregt, wobei die Ausgänge aber auch die internen Signale und Zustände beobachtet werden können. Da eine vollständige Verifikation aller möglichen Eingangssignalkombinationen und innerer Zustände in der Praxis aufgrund der vorhandenen Komplexität nicht möglich ist, beschränkt man sich auf die Verifikation von ausgesuchten Testfällen. Zur Nachbildung der Hardware auf einem Computer kommt eine Simulationssoftware wie z. B. ModelSim zum Einsatz. Bei der Simulation eines Systems wird das zu testende Gerät (device under test - DUT) von der Testbench angeregt und das Verhalten beobachtet und mit dem erwarteten Verhalten verglichen.

6.1 Aufbau des Testsystems

Da das erstellte System wegen der großen Komplexität nur schwierig in einem Stück zu verifizieren ist, wurden zwei Testbeches mit unterschiedlichen DUTs erstellt. Die Komplexität ist bedingt durch unterschiedliche Bussysteme wie PCI, WISHBONE, AMBA-AHB und schließlich Ethernet, die jedes für sich eine komplexe Zustandsmaschine bilden. Jedes Bussystem für sich gehorcht einer großen Anzahl von zeitlichen und funktionalen Regeln; die Aneinanderkettung jedoch macht das System äußerst komplex. Praktisch müsste man für eine alles umfassende Testbench alle möglichen Kombinationen abdecken, was diese unüberschaubar komplex und fehleranfällig machen würde. So ist es nahezu unmöglich anhand des Verlaufs eines Ausgangssignals die dazugehörige Transaktion am PCI-Bus exakt zu lokalisieren, da es alleine im PCI-System zu unterschiedlichen Verzögerungen z. B. aufgrund von posted writes oder delayed reads kommen kann.

Da es in der Praxis ohnehin nicht viel Sinn hat, einen bereits verifizierten IP-Core selbst nochmals in einem deutlich komplexeren System erneut zu verifizieren, wurden zwei Testbenches erstellt, wobei die Namen nach dem Protokoll der Stimulation gewählt wurden:

- PCI-Testbench: Testbench des kompletten Systems
- WISHBONE-Testbench: Testbench ohne PCI-Bridge, anstatt dessen mit WISHBONE Behavioural Master/Slave

Die PCI-Testbench umspannt das komplette System, wobei mit dieser nur die grundsätzliche Funktion überprüft werden kann. Es soll überprüft werden, ob alle an die Verbindungslogik angeschlossenen IP-Cores korrekt über den PCI-Bus adressiert werden können d. h. dass Lese- und Schreib-

transaktionen korrekt abgeschlossen werden. Die WISHBONE-Testbench ist hinsichtlich des PCI-Cores vereinfacht: Der PCI-Core wird durch ein generisches WISHBONE Master/Slave-Interface ersetzt, sodass Lese- und Schreibtransfers nicht mehr von der PCI-Seite aus gestartet werden müssen, sondern innerhalb des PCI-Cores an einem WISHBONE-Interface.

Die erstellten Testbenches wurden mit der Simulationssoftware ModelSim 6.0 getestet, wobei die Erstellung und Anpassung im VHDL- oder Verilog-Quellcode mithilfe des externen Editors Ultraedit durchgeführt wurden.

6.1.1 PCI-Testbench

Die PCI-Testbench basiert auf der Testbench für den PCI-Core vom Miha Dolenc und Tadej Markovic (vgl. [7]). Diese Testbench wurde dahingehend erweitert und verändert, sodass neben dem Funktionstest des PCI-Cores auch der Datentransfer zwischen im PCI-Bussystem vorhandenen Geräten und dem Syn1588-Core sowie dem Ethernet MAC getestet werden kann. Abbildung 6.1 zeigt die Struktur der Testbench.

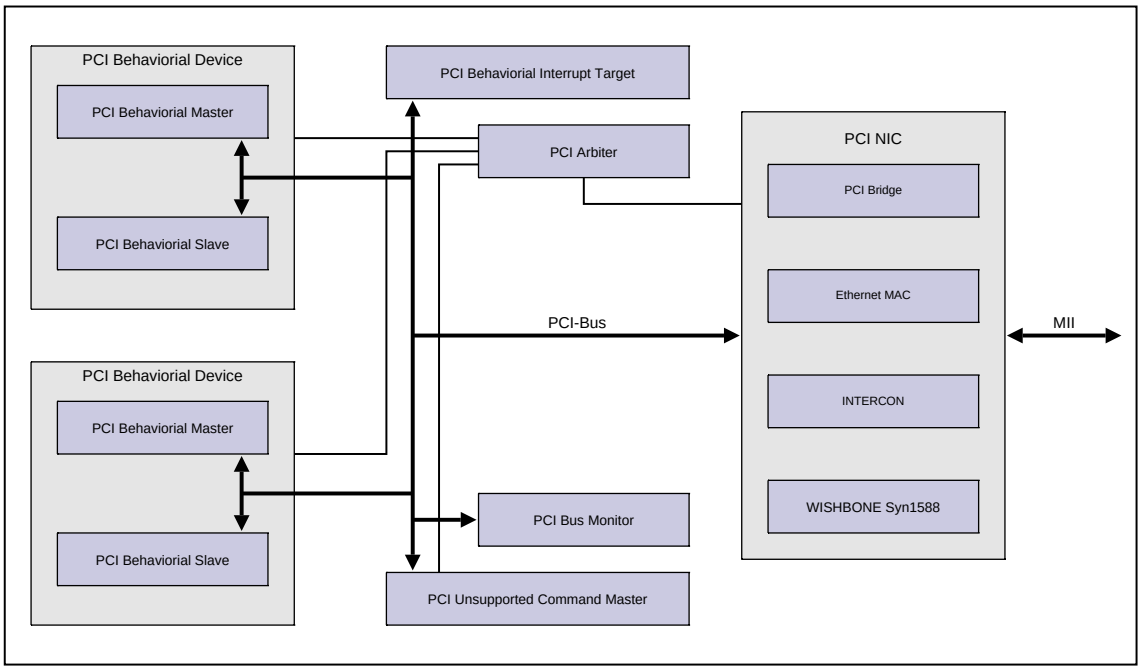

Abbildung 6.1: Aufbau der PCI-Testbench

Die beiden PCI Behavioral Devices simulieren das Vorhandensein zweier weiterer PCI-Geräte neben der PCI-NIC. Als behavioural bezeichnet werden Funktionsblöcke, die zur Simulation verwendet werden können, aber nicht synthesierbar sind. Dies bedeutet, dass diese Funktionsblöcke nur zur formalen Verifikation verwendet werden können, jedoch nicht in Hardware umsetzbar sind.

Jedes PCI Behavioral Device wird im Guest-Modus betrieben und besteht, sofern es sich bei dem PCI-Gerät nicht nur um eine reines Target handelt, aus einem Initiator (Master)- und einem Target

(Slave)-Interface. Der PCI Behavioral Master erhält seine Stimuli von der Testbench und kann PCI-Buszyklen generieren. Er initiiert daher in Abhängigkeit der in der Testbench ausgeführten tasks verschiedene Schreib- und Lesezyklen. Außerdem arbitriert der Master vor dem Zugriff auf den Bus über den PCI Arbiter. Der PCI Behaviorial Slave nimmt Kommandos des PCI-Busses entgegen. Er verfügt über ein 256 Byte großes SRAM, um bei Lesekommandos ein korrektes Datum zurückgeben zu können, und um mittels Schreibkommandos den Speicher beschreiben zu können.

Das PCI Behavioral Interrupt Target dient lediglich zur Antwort auf das Interrupt Acknowledge-Buskommando und trägt den Interruptvektor. Der PCI Arbiter regelt über ein Arbitrierungsschema den Zugriff auf den PCI-Bus. Der PCI Bus Monitor überwacht den PCI-Bus und versucht Fehler im Protokoll zu erkennen. Er überwacht auch die Enable-Signale der Bustreiber aller PCI-Geräte, um auch erkennen zu können, wenn mehrere Geräte den Bus gleichzeitig zu treiben versuchen, auch wenn diese den Bus mit dem gleichen logischen Wert treiben. Der PCI Unsupported Command Master kann Buszyklen initiieren, die nicht der PCI-Spezifikation entsprechen. Diese fehlerhaften Buszyklen können verwendet werden um das korrekte Handling der Targets zu überprüfen, wenn Fehler am PCI-Bus auftreten. Dies kann z. B. zur Überprüfung der Implementierung des Parity-Checks verwendet werden.

Initialisierung

Bevor auf eine PCI-Bridge zugriffen werden kann, welche im Guest-Modus implementiert ist, müssen eine Reihe von Initialisierungen durchgeführt werden. So wird beim Start eines PCI-Systems der PCI-Bus vom Host-System nach allen vorhanden Geräten abgesucht und anschließend durch configuration writes parametriert. Für diesen Zweck müssen folgende Operationen durchgeführt werden:

1. Konfiguration aller PCI-Basisadressregister
2. Aktivierung der Master und Einstellung der Speicherbereiche der PCI-Slave
3. Konfiguration anderer Type0 PCI-Register
4. Einstellung der restlichen Konfigurationregister über Schreib- und Lesezugriffe auf das erste PCI-Image
5. Zugriff auf alle WISHBONE-Interfaces und Aktivierung der WISHBONE-Master

Die ersten drei Punkte werden in der Regel vom PCI-Host-System durchgeführt, während ab Schritt vier ein Zugriff über einen Treiber möglich ist. In einer Testbench muss der ganze Konfigurationsvorgang durch Aufruf der geeigneten Tasks durchgeführt. Da die Tasks sehr allgemein gehalten sind, um jede nur erdenkliche Funktion des PCI-Busses korrekt wiedergeben zu können, weißt die als Basis herangezogene Testbench eine derartige Komplexität auf, dass man die sehr umfangreiche PCI-Spezifikation komplett bis ins Detail kennen muss, um die teils schlecht dokumentierten Funktionen zuordnen zu können.

Aus diesem Grund wurde an der Initialisierung nichts verändert, sondern nur der Zugriff auf die WISHBONE-Geräte hinter der PCI-Bridge getestet.

6.1.2 WISHBONE-Testbench

Die WISHBONE-Testbench besteht aus dem in der Implementierung beschriebenen System, wobei der PCI-Core durch einen WISHBONE Behavioural Master und Slave ersetzt wurde. Dies erlaubt es, das erstellte DUT unter Umgehung des PCI-Systems zu testen d. h. als ob die am PCI-Bus vom Master initiieren Transfers umgesetzt und direkt durchgeleitet würden. Im Allgemeinen kommt es beim PCI-Bus zu einer gewissen Verzögerung bis ein Schreib- oder Lesebefehl an das WISHBONE-Interface weitergeleitet wird.

Jeder PCI-Master muss zu Beginn eines Transfers beim Arbiter erst den Bus anfordern, wobei das Verfahren der Arbitrierung in der Spezifikation nicht festgelegt ist, und außerdem von der Konfiguration der Host-Bridge abhängt (wie z. B. dem Latency Timer). Die PCI-Bridge puffert wiederum je nach konfigurierter Puffertiefe Transfers, selbst wenn diese von unterschiedlichen Mastern initiiert wurden. Somit fällt selbst die Zuordnung, welcher Transfer von welchem PCI-Master gestartet wurde, im Impulsdiagramm nicht leicht.

Abhilfe schafft die Verwendung eines WISHBONE-Masters und -Slaves, welche die PCI-Bridge ersetzen. Implementiert sind die beiden Busteilnehmer als behavioural, sodass gegenüber einer synthetisierbaren Version wesentlich mehr Freiheit bei der Erstellung der notwendigen Signale gegeben ist. Abbildung 6.2 zeigt den Aufbau der WISHBONE-Testbench.

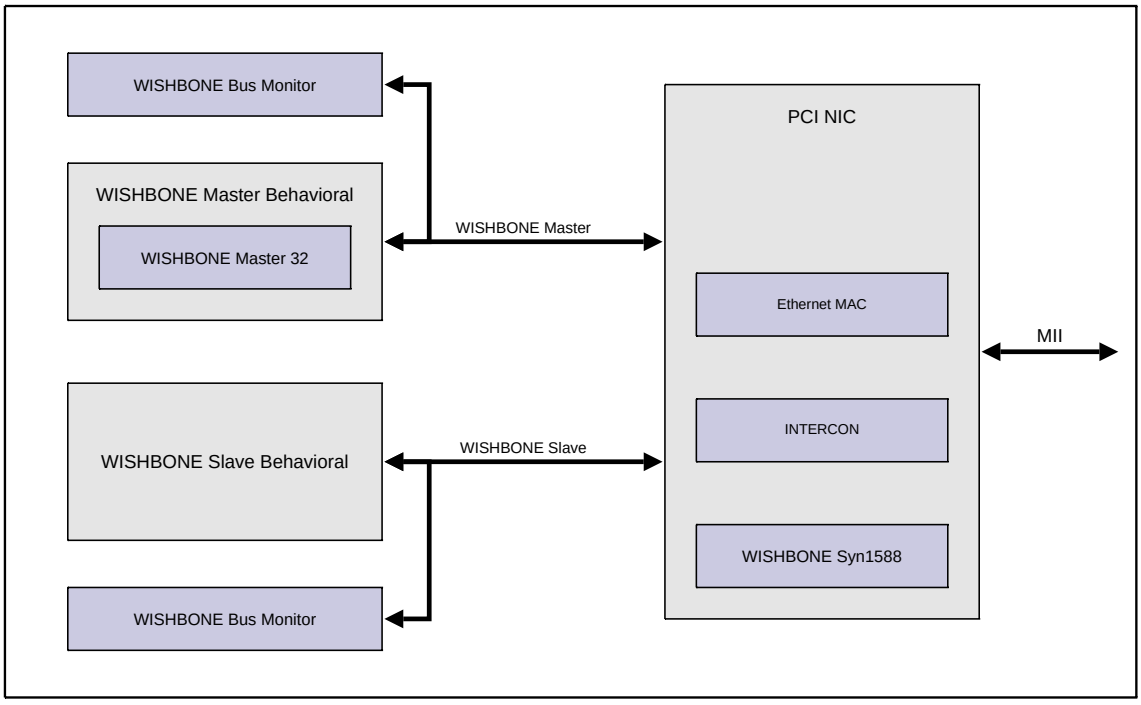

Abbildung 6.2: Aufbau der WISHBONE-Testbench

Der WISHBONE Master Behavioural sowie der WISHBONE Slave Behavioural bilden zusammen ein Master- und Slave WISHBONE-Interface. Mithilfe des WISHBONE Master Behavioural-Moduls können Einzel- und Blocktransfers von der Testbench aus über die dazugehörenden Tasks gestartet

werden. Dieses Modul beinhaltet ein Submodul, um die korrekten WISHBONE-Buszyklen zu erstellen. Der WISHBONE Slave Behavioural antwortet auf aus der PCI-NIC gestartete Transfers. Dieses Modul beinhaltet – wie auch der PCI Behavioural Slave – einen SRAM-Speicher. Das WISHBONE-Interface wurde außerdem um jeweils einen Bus Monitor für sowohl den Master als auch den Slave ergänzt, um Protokollfehler am WISHBONE-Interface erkennen zu können. Der WISHBONE Bus Monitor sowie das Master und Slave Behaviorial Interface wurden von OpenCores bezogen und für diese Testbench angepasst. Der Autor dieser Komponenten ist Blue Beaver.

Soll nun ein Transfer vom WISHBONE Master Behavioural gestartet werden, so muss der gewünschte Task gestartet werden. Es können verschiedene Parameter wie z. B. die Anzahl der Wiederholungen im Fehlerfall angegeben werden. Im Gegensatz zu PCI ist die Anzahl der verschiedenen Transfers aufgrund der WISHBONE-Architektur auf drei grundsätzliche Tasks in jeweils zwei Transferrichtungen beschränkt:

- wb_single_read() und wb_single_write()
- wb_block_write() und wb_block_read()
- wb_RMW_read() und wb_RMW_write()

Der WISHBONE Slave Behavioural kann selbst keine Transfers initiieren, sein Verhalten als Slave kann aber durch folgende Tasks definiert werden:

- cycle_response(): Mithilfe dieses Tasks ist es möglich das Zeitverhalten festzulegen. Es kann die Anzahl der Wartezyklen, die der Slave einfügen soll, festgelegt werden sowie die maximale Anzahl an Übertragungsversuchen. Außerdem kann definiert werden, dass der Slave trotz eines korrekten Buszyklus mit Retry oder Error antwortet, um so das Fehlerverhalten des Masters zu testen.
- wr_mem(): Dieser Task ermöglicht es der Testbench das SRAM des Slaves mit Daten zu füllen.
- rd_mem(): Dieser Task ermöglicht es der Testbench das SRAM des Slaves auszulesen.

6.2 Synthese

Nach den erfolgreichen Simulationsergebnissen mithilfe der Software ModelSim 6.0 wurde die Synthese mit Quartus II gestartet. Beim Synthesevorgang wird aus den in den Hardwarebeschreibungssprachen Verilog und VHDL codierten Modulen eine Netzliste ausgebaut. Diese Netzliste setzt sich aus Gattern, Registern, Flip-Flops und anderen grundlegenden Logikelementen zusammen, die vom Compiler in einer geeigneten Art und Weise miteinander verbunden sind, um die beschriebene Hardware umzusetzen.

Die Überprüfung des VHDL- und Verilog-Codes erfolgt nach strengeren Kriterien wie bei der Simulation. So mussten kleinere Änderungen im Code vorgenommen werden, welche zur Verifikation zuerst in ModelSim simuliert werden mussten, um eventuelle Fehler und Nebenwirkungen im Vorfeld lokalisieren zu können.

Um sicher zu stellen, dass das Design unabhängig vom zugrunde liegenden FPGA synthetisiert werden kann, wurde die Synthese für folgende zwei FPGAs von der Firma Altera durchgeführt durchgeführt:

- Stratix II EP2S30F672C5
- Cyclone EP1C20F324C6

Das Stratix II-FPGA findet sich auch auf dem Switch-Board, welches von der der österreichischen Akademie der Wissenschaften zum Test zur Verfügung gestellt wurde. Das Cyclone-FPGA wurde aus Vergleichsgründen herangezogen, da eben dieses in Kapitel 3 zum Vergleich mit Alteras IP-Cores herangezogen wurde. Tabelle 6.1 zeigt einen Vergleich der benötigen Logikzellen, Register und Speicherzellen für beide FPGAs, wobei die Synthese für beide FPGAs in Quartus II V5.0 durchgeführt wurde. Da an dieser Stelle keine Vergabe der Pins durchgeführt wurde, kann der Ressourcenbedarf nach der korrekten Platzierung geringfügig variieren.

Tabelle 6.1: Vergleich des Ressourcenbedarfs

FPGA	Modul	Logikzellen	Register	Speicher (Bits)
Stratix II	Gesamt	7396	6644	18832
	Ethernet MAC	1864	1332	9216
	Intercon	47	0	0
	PCI-Bridge	1291	1296	1424
	Syn1588	4113	4016	8192
Cyclone	Gesamt	15531	7634	17952
	Ethernet MAC	2713	1332	9216
	Intercon	15	0	0
	PCI-Bridge	3663	2286	544
	Syn1588	9102	4016	8192

Auf dem älteren Cyclone-FPGA werden mehr als doppelt so viele Logikzellen benötigt als auf dem wesentlich neueren Stratix II-FPGA. Das Design belegt im Cylone-FPGA damit 59% der logischen Elemente, im Stratix II hingegen nur 39%. Die Ausnutzung des vorhandenen Speichers ist in beiden Fällen recht gering: 6% beim Cyclone und 1% beim Stratix II. Anschaulich sind diese Zusammenhänge in Abbildung 6.3 dargestellt.

Zur Überprüfung der maximal erreichbaren Taktfrequenzen wurde eine Timing Analysis durchgeführt. Die Ergebnisse werden in Tabelle 6.2 gezeigt. Wie nicht anders zu erwarten war, erlaubt das Stratix II-FPGA gegenüber dem älteren low-cost Cyclone-FPGA etwas höhere Taktfrequenzen, obwohl das erstellte System weniger als die Hälfte der Logikzellen gegenüber dem Cyclone-FPGA benötigt. Dies ist vorallem auf die verbesserte Architektur des Stratix II-FPGAs zurückzuführen.

Tabelle 6.2: Erreichbare Taktfrequenzen

Taktsignal	Cyclone	Stratix II
PCI_CLK	130.48 MHz	168.95 MHz
WB_CLK	33.42 MHz	39.06 MHz
MII_RX_CLK	90.24 MHz	96.07 MHz
MII_TX_CLK	63.62 MHz	83.09 MHz

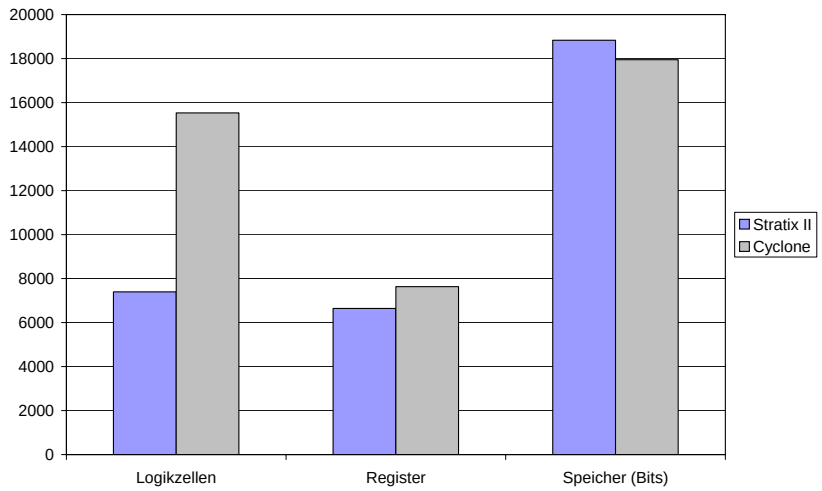

Abbildung 6.3: Vergleich der benötigten Chipfläche

6.3 Platzierung und Verdrahtung

Um die aus der Synthese gewonnene Netzliste tatsächlich in das FPGA einspielen zu können, muss die Logik verdrahtet und platziert werden. Dazu müssen die Ausgangssignale der Top Level-Entity mit den entsprechenden Pins der Hardware verbunden werden. Der Fitter und Assembler von Quartus erstellen aus der Netzliste die Logik für den gewählten Typ des FPGAs aus speziellen Grundelementen, plazieren diese im FPGA und sorgen für eine korrekte Verdrahtung. Sobald dies feststeht, können mittels des Timing Analyzers Abschätzungen über die maximale Taktfrequenz gemacht werden.

Quartus hat nach Durchlauf aller Schritte eine Programmierdatei erstellt, welche über ein serielles Interface in den FPGA gespielt werden kann. Treten Fehler auf, so muss der gesamte Entwicklungsablauf von vorne gestartet werden. Also beginnend von der Simulation in ModelSim, um formale Fehler zu finden, bis hin zur Synthese und dem Place and Route.

7 Schlussbetrachtung

In diesem Abschnitt sollen die Ergebnisse dieser Arbeit zusammengefasst werden und mögliche Ansätze für eine Weiterentwicklung dargelegt werden.

7.1 Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde die Funktion einer Netzzwerkkarte für Uhrensynchronisation analysiert. Die Funktionalität lässt sich in vier Module aufteilen: Der PCI-Bridge zur Anbindung an das Bussystem des Computers, dem Ethernet MAC zur Versand und Empfang von Frames, dem Uhrensynchronisationsmodul Syn1588 sowie einer Verbindungslogik.

Es wurde eine Auswahl von geeigneten IP-Cores getroffen. Zu diesem Zweck wurde eine Implementierung mittels von Altera zur Verfügung gestellten IP-Cores untersucht und mit einer Lösung von IP-Cores von der Internetplattform OpenCores verglichen. Hinsichtlich der Kriterien Chipfläche, Portierbarkeit, Adaptierbarkeit und Funktion erwies sich schließlich die OpenCores-Lösung als die bessere. Bei Altera mangelte es primär an der Funktion, da die Arbitrierung an der AHB-Seite der Avalon AHB-Bridge fehlerhaft war.

Im weiteren Verlauf der Arbeit wurden die im Design vorkommenden Busprotokolle wie PCI, WISHBONE, MII, AHB analysiert um mithilfe dieses Wissens die einzelnen Funktionsblöcke korrekt parametrieren und richtig verbinden zu können. Zur chipinternen Verbindung wurde die WISHBONE-Struktur ausgewählt, da diese ein einfaches universelles Interface ist, welches sich gut zur Verbindung von IP-Cores eignet, und da sowohl der PCI-Core als auch der MAC bereits über dieses Interface verfügten. Es wurde eine WISHBONE-AHB-Bridge entworfen, die es ermöglicht IP-Cores mit AHB-Slave-Interfaces an die WISHBONE-Verbindungsstruktur anzuschließen. Diese Bridge wurde für den Uhrensynchronisation-Core Syn1588 benötigt, da dieser nativ über ein AHB-Interface verfügt.

Zur Verbindung aller WISHBONE-Interfaces wurden verschiedene Möglichkeiten der Zusammenschaltung untersucht und schließlich ein System basierend auf einem Kreuzschienenverteiler mit einstellbaren Prioritäten implementiert. Während der gesamten Implementierungsphase wurde das entwickelte System mit ModelSim verifiziert und schließlich in dem von der österreichischen Akademie der Wissenschaften zur Verfügung gestellten Switch-Board getestet.

Eine Implementierung auf Basis der von Altera verfügbaren IP-Cores war anfänglich sehr verlockend, da die ausgereiften Entwicklungswerkzeuge wie der SOPC-Builder versprachen, ein fehlerfreies System rasch erstellen zu können. Die Parametrierung und Erstellung des System lässt sich

tatsächlich bei guter Kenntnis der Komponenten rasch erledigen. Es zeigte sich jedoch, dass das so zusammengefügte System fehlerhaft war. Der vom Compiler erstellte Code erzeugte eine fehlerhafte Arbitrierungslogik auf der AHB-Seite der Avalon-AHB-Bridge und verursachte, dass die Bridge für immer auf das HREADY-Signal zur Bestätigung des Beginns eines Transfers wartete.

Eine Anfrage beim Support von Altera verlief negativ. Die erste Antwort war offensichtlich für eine andere Supportanfrage bestimmt und auf die zweite Anfrage wurde erklärt, dass der Support für diese Bridges eingestellt worden wäre, obwohl diese im aktuellen Produkt Quartus II V5.0 enthalten waren. In Quartus II Version 5.1 wurden die Bridges dann tatsächlich entfernt.

Die Implementierung einer allgemeinen WISHBONE-AHB-Bridge in beide Richtungen erwies sich als keine einfache Lösung. Die WISHBONE-Struktur spezifiziert ein relativ einfaches Verbindungssystem, während das AHB-System vor allem für die AHB-Master einen relativ komplexen Aufbau hat. Möchte man nicht nur eine Untermenge aller möglichen AHB-Bursts unterstützen, sondern alle AHB-Bursts mit hoher Performance unter Ausnutzung des Address Pipelinings umsetzen, so wird diese Bridge sehr komplex. Aus diesem Grund wurde auf die Verwendung von performancesteigernden *posted writes* und *delayed reads*, welche für zur Ausnutzung des Address Pipelining notwendig sind, verzichtet. Auch auf den Entwurf einer AHB-WISHBONE-Bridge, welche die Anbindung eines AHB-Masters an die WISHBONE-Verbindungsstruktur ermöglicht, wurde mangels Bedarfs verzichtet, da für den Syn1588-Core nur eine WISHBONE-AHB-Bridge notwendig war.

7.2 Ausblick

Die im Rahmen dieser Arbeit entwickelte Netzwerkkarte bietet zahlreiche Möglichkeiten einer Weiterentwicklung und Anpassung.

7.2.1 Treiberentwicklung

Um den Netzwerkadapter tatsächlich in einem Betriebssystem verwenden zu können, muss ein für die Hardware angepasster Treiber erstellt werden. Der Gerätetreiber stellt gegenüber dem Betriebssystem eine definierte Schnittstelle zur Verfügung, die es dem Betriebssystem erlaubt, durch standardisierte Funktionsaufrufe auf das Gerät zugreifen zu können.

Beim Systemstart scannt das Betriebssystem den PCI-Bus und legt die Basisadressen aller Geräte fest. Die Aufgabe des Treibers ist es, die über Grundkonfiguration, welche bereits durch die Synthese festgelegt worden ist, hinausgehenden Einstellungen zu treffen. Es müssen z. B. die Einstellungen für die Adressumsetzung im WISHBONE-Image der PCI-Bridge festgelegt werden sowie der Ethernet MAC aktiviert und konfiguriert werden. Die Erstellung eines Treibers erfordert aus diesem Grund sowohl eine genaue Kenntnis der Hardware und der Bedeutung deren Register als auch eine gute Kenntnis des zugrundeliegenden Betriebssystems.

7.2.2 Mögliche Anpassungen

Das im Rahmen dieser Arbeit entwickelte System kann durch Austausch oder Erweiterung der Funktionsblöcke an neue Technologien und Standards angepasst und ergänzt werden.

Unterstützung von Gigabit Ethernet

Eine Möglichkeit der Weiterentwicklung ist es, sich nicht nur auf die Ethernet-Varianten mit 10 und 100 MBit/s zu beschränken, sondern auch Gigabit Ethernet mit 1 GBit/s zu unterstützen. Dieser Standard verwendet jedoch kein MII-Interface, sondern dessen Nachfolger das GMII (Gigabit Media Independent Interface). Dies erfordert einen Ethernet MAC, der auch für Gigabit Ethernet geeignet ist, aber auch einen geänderten MII-Scanner (besser gesagt GMII-Scanner), welcher die Frames nach dem Start-of-Frame-Delimiter durchsucht. An die Hardware werden auch höhere Anforderungen gestellt, da das GMII-Interface mit 8 Bit doppelt so breit ist wie MII und mit 125 MHz fünf Mal so hoch getaktet wird.

Anpassung an PCI-Express

Der weitverbreitete PCI-Bus wird immer mehr von seinem direkten Nachfolger PCI-Express in Computern abgelöst. Ein Feld der Anpassung wäre es, die PCI-Bridge durch eine PCI-Express-Bridge zu ersetzen, um somit Computer ohne PCI-Bus unterstützen zu können. Dies erfordert neben einem angepassten Treiber eine spezielle Hardware, mit Transceivern, die mit 1,25 GHz Taktfrequenz umgehen können und nicht zuletzt einen PCI-Express-Core. Zum heutigen Zeitpunkt ist jedoch kein lizenzfreier PCI-Express-Core verfügbar. Die eigene Erstellung eines derartigen Cores gehört sicherlich zu einer anspruchsvollen Aufgabe.

Implementierung des CPU-Cores

Ein Großteil der Abwicklung der Uhrensynchronisation des Syn1588-Cores wird vom Treiber und der Software ausgeführt. Diese Aufgaben könnten ebenso mittels einer an die WISHBONE-Verbindungsstruktur angeschlossenen CPU erledigt werden und würden damit den Hauptprozessor der Computers entlasten. Je nach Aufbau eines Multitasking-Betriebssystems ist nicht immer sicher gestellt, dass dieses auch allen Anforderungen des Treibers rechtzeitig nachkommen kann. Es erscheint daher als günstig z. B. den regelmäßigen Versand von Synchronisationspaketen einer CPU im Netzwerkadapter zu überlassen, der dies unabhängig von den im Hauptprozessor laufenden Prozessen erledigen kann.

Anwendung der Uhrensynchronisation für Wireless LAN-Netze

Eine weitere Herausforderung wäre es, die Netzwerkkarte für eine Uhrensynchronisation in einem WLAN (Wireless LAN)-Netz anzupassen. Im Gegensatz zu einer sternförmigen kabelgebundenen

Ethernettopologie mit Switches ist bei WLAN das Medium Luft stets ein gemeinsames Medium, bei dem es prinzipbedingt durch das Zugriffsverfahren CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) zu Kollisionen kommt. Zwar reservieren die einzelnen Knoten gewisse Funkzellen, um Kollisionen zu vermeiden, es lassen sich aber ähnlich wie bei hochausgelastetem Ethernet keine Garantien abgeben, wann ein bestimmtes Paket übertragen wird. Hinzu kommt, dass sich die Topologie des Netzes ständig ändern kann, wenn ein Knoten von einem Access-Point zum nächsten übergeben wird. Mittels des Systems der Hardware-Zeitstempelung ließe sich auch bei WLAN der exakte Sende- und Empfangszeitpunkt bestimmen, und damit der Grundstein für eine genaue Uhrensynchronisation in einem WLAN-Netz legen. Im Gegensatz zu den kabelgebundenen Ethernet-Varianten gibt es bei WLAN kein standardisiertes Interface, welches mit MII vergleichbar wäre. Die Erkennung des Beginns eines Synchronisationspakets ist daher wesentlich aufwändiger z. B. durch Gewinnung aus dem analogen Modulationssignal.

Literaturverzeichnis

[1] ANCEAUME, Emmanuelle ; PUAUT, Isabelle: A Taxonomy of Clock Synchronization Algorithms. In: *Publication Interne No. 1103, ISSN 1166-8687* Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France, 1997

[2] ANDREWES, W.J.H.: Finding space on Earth: The quest for longitude 1500-1800. In: *Frequency Control Symposium and Exhibition, 2000. Proceedings of the 2000 IEEE/EIA International*, 2000, S. 3–6

[3] ARM LTD.: *AMBA License and Specification*. Rev. 2.0. verfügbar unter: http://www.arm.com/products/solutions/AMBA_Spec.html (12.12.2006): ARM Ltd., 1999

[4] BIGLER, Thomas ; WINKLER, Franz ; GADERER, Georg: Entwurf eines Embedded Systems und IP-Cores zur Uhrensynchronisation. In: *Proceedings of the 2005 Austrochip Conference*, 2005, S. 119–126

[5] BIPM. *Bureau international des poids et mesures*. verfügbar unter: http://www.bipm.org (17.1.2007)

[6] CRISTIAN, Flaviu: Probabilistic Clock Synchronization. In: *Distributed Computing* 3 (1989), Nr. 3, S. 146–158

[7] DOLENC, Miha ; MARKOVIC, Tadej: *PCI IP Core Design document*. verfügbar unter: http://www.opencores.org/ (10.1.2007): OpenCores, 2002

[8] DOLENC, Miha ; MARKOVIC, Tadej: *PCI IP Core Specification*. verfügbar unter: http://www.opencores.org/ (10.1.2007): OpenCores, 2004

[9] GADERER, G. ; HOLLER, R. ; SAUTER, T. ; MUHR, H.: Extending IEEE 1588 to fault tolerant clock synchronization. In: *Factory Communication Systems, 2004. Proceedings. 2004 IEEE International Workshop on*, 2004, S. 353–357

[10] HEIN, Mathias: *Ethernet: Standards, Protokolle, Komponenten*. 1. Auflage. Thomson Publishing, 1995

[11] HERVEILLE, Richard: *Specification for the Wishbone System-on-Chip Interconnection Architure for Portable IP Cores*. verfügbar unter: http://www.opencores.org/projects.cgi/web/wishbone/wbspec_b3.pdf (10.1.2007): OpenCores, 2006

[12] HOLLER, R. ; SAUTER, T. ; KERO, N.: Embedded SynUTC and IEEE 1588 clock synchronization for industrial Ethernet. In: *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference* Bd. 1, 2003, S. 422–426 vol.1

[13] HORAUER, Martin: *Clock Synchronisation in Distributed Systems*, TU Vienna, Diss., 2004

[14] IEEE INSTRUMENTATION AND MEASUREMENT SOCIETY TC9: *1588 IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systemsn*. IEEE, 2000

[15] JENNINGS, Steven: IEEE 1588 - die Synchronisation über Ethernet: Hier ticken alle Uhren gleich. In: *IEE - Automatisierung + Datentechnik* 47. Jg, Nr. 7 (2002), S. 52 – 55

[16] LOSCHMIDT, Patrick: *IEEE 1588 Clock Core User Guide*. Version 0.10 preliminary, 2007

[17] MILLS, D.L.: Internet time synchronization: the network time protocol. In: *Communications, IEEE Transactions on* 39 (1991), Oct., Nr. 10, S. 1482–1493

[18] MOHOR, Igor: *Ethernet IP Core Specification*. Rev. 1.19. verfügbar unter: http://www.opencores.org/ (3.1.2007): opencores.org, 2002

[19] PCI SIG: *PCI Specification 2.1s*. Portland, Oregon: PCI Special Interest Group, 1995

[20] RECH, Jörg: *Ethernet: Technologien und Protokolle für die Computervernetzung*. 1. Auflage. Verlag Heinz Heise, 2002

[21] SCHMID, Ullrich ; SCHOSSMAIER, Klaus: Interval-based Clock Synchronization. In: *Journal of Real-Time Systems* Bd. 2, 1997, S. 173–228

[22] SHANLEY, Tom ; ANDERSON, Don: *PCI System Architecture*. Third Edition. Addison-Wesley Publishing Company, 1995

[23] SHIN, Kang G. ; RAMANATHAN, P.: Clock synchronization of a large multiprocessor system in the presence of malicious faults. In: *IEEE Transactions on Computers*, 1987, S. 2–12

[24] TANENBAUM, Andrew ; STEEN, Marten van: *Verteilte Systeme: Grundlagen und Paradigmen*. Pearson Education, München, 2003

[25] TROXEL, Gregory: *Time Surveying: Clock Synchronisation over Packet Networks*, Department of Electrical Engineering and Computer Science, Massachusetts Institut of Technology, Diss., 1994

[26] UNNEBACK, Michael: *WISHBONE Builder*. verfügbar unter: http://www.opencores.org/ (15.1.2007): opencores.org, 2005

[27] VONNAHME, Erik ; RÜPING, Stefan ; RÜCKERT, Ulrich: Measurements in Switched Ethernet Networks Used For Automation Systems University of Padeborn, Germany, 2000, S. 231 – 238

Glossar

Abkürzung	Beschreibung
AHB	Advanced High Performance Bus
COTS	commercial off-the-shelf
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
FCS	Frame Check Sum
FPGA	Field Programmable Gate Array
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
IP	Internet Protocol
MAC	Media Access Controller
MDI	Media Dependent Interface
MII	Media Independent Interface
PCI	Peripheral Component Interconnect
PCI SIG	Peripheral Component Interconnect Special Interest Group
PTP	Precision Time Protocol
RMON	Remote Monitoring
SFD	Start-of-Frame-Delimiter
Syn1588	Synchronised IEEE 1588
TAI	Temps Atomique International, International Atomic Time
TDMA	Time Division Multiple Access
UTC	Coordinated Universal Time
WB	WISHBONE