Mareike Stöppler

Implementierung einer Ethernetschnittstelle auf einem FPGA

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Henning Hasemann
Zweitgutachter : Prof. Dr.–Ing. Ulfert Meiners

Abgegeben am 11. Januar 2010

Mareike Stöppler

Thema der Diplomarbeit

Implementierung einer Ethernetschnittstelle auf einem FPGA

Stichworte

FPGA, Ethernet, VHDL

Kurzzusammenfassung

Diese Arbeit umfasst die Implementierung einer Ethernetschnittelle auf einem FPGA unter Verwendung des Entwicklerboards Spartan-3E der Firma XILINX, Inc.. Die Implementierung erfolgt im Rahmen des OPERA-Experimentes zum experimentellen Nachweis der Neutrinooszillation, an welchem unter anderem die Universitäten Hamburg und Rostock beteiligt sind. Vorgestellt werden die Grundlagen der Netzwerkübertragung, der Ethernetstandards und des Aufbaus eines FPGAs. Für die konkrete Umsetzung wird ein Quelltext mittels VHDL zur Konfiguration des FPGAs entwickelt. Es wird die Simulation des erstellten digitalen Schaltkreises und das Testen der Sendung eines Ethernetframes dargestellt und erläutert.

Mareike Stöppler

Title of the paper

Implementation of an ethernetinterface on a FPGA

Keywords

Ethernet, FPGA, VHDL

Abstract

This report examines the construction of an ethernetinterface on a FPGA using the developementboard Spartan-3E of the company XILINX, Inc.. This task is part of the OPERA-Experiment for the experimental evidence of the phenomenon of neutrino oscillations, with cooperation of the universities of Hamburg and Rostock. The basics of networktransmission, of ethernetstandards and FPGAs are shown. For the implementation there will be developed a source code using VHDL to configure the FPGA. The simulation of a constructed digital circuit and the test of transmitted ethernetframe will be displayed and explained.

Inhaltsverzeichnis

1 Einleitung

Field Programmable Gate Arrays[1] werden nicht nur im Bereich der digitalen Audioverarbeitung und Echtzeitverarbeitung eingesetzt, sondern sollen auch verstärkt im Bereich der Teilchenphysik zum Einsatz kommen. Als Beispiel zur Integration von FPGAs in der Teilchenphysik wurde hier das OPERA[2]- Experiment, an welchem die Universität Hamburg beteiligt ist, gewählt. Allgemein geht es in dem OPERA- Experiment bezüglich der Elektrotechnik um das Messen und Auswerten von sehr geringen Spannungspulsen aus Driftröhren[3]. Um diese Spannungspulse messen und speichern zu können, ist jede Driftröhre an einen Verstärker angeschlossen, welcher das Signal aufbereitet und verstärkt an eine Ausleseelektronik weiterleitet. Diese Ausleseelektronik besteht aus dem von der Universität Hamburg und der Universität Rostock entwickelten TDC[4]-Board, welches für die Driftzeitmessung und das Speichern der Messdaten in einem FIFO[5] zuständig ist, sowie aus einem Mezzanine[6]- Board, welches das Senden der Daten an einen Server und das Versehen eines Zeitstempels übernimmt. Der Aufbau und die Aufgaben dieser Boards werden in dem Kapitel 2.1 näher erläutert. Diese Messelektronik soll im Ganzen in weiteren Experimenten zur Neutrinooszillation durch ein neu entwickeltes Board ersetzt werden, wobei der Hauptbestandteil der Ausleseelektronik aus FPGAs bestehen soll. Das bedeutet, das Setzen eines Triggersignals, das Messen und Auswerten der Spannungspulse, das Versehen eines Zeitstempels und das Senden der Messdaten an einen Server sollen mittels FPGAs umgesetzt werden. Diese Bausteine wurden aufgrund ihrer einfachen Re- Konfiguration und des geringen Kostenaufwandes gewählt. In dieser Arbeit wird, als Teil des Gesamtprojektes, das Senden von Daten an einen Zielrechner mit integrierter Netzwerkkarte über eine, im FPGA zu konstruierende, Ethernetschnittstelle realisiert. Selbst festgelegte und in einem BlockRAM[7] gespeicherte Daten werden zu einem Ethernetframe[8] aufbereitet und über einen externen Baustein auf ein Übertragungsmedium gegeben.

Um einen etwas genaueren Einblick in das Experiment und den OPERA- Detektor bekommen zu können, was für dieses hier dargestellte Projekt jedoch nicht grundlegend ist, befindet sich im Anhang A eine kurze Erläuterung zur Neutrinooszillation, mit welcher sich das Experiment befasst, und zum Aufbau des Detektors.

---

[1] FPGA (engl.): im Arbeitsbereich programmierbare aus Logikgattern bestehende Matrizen
[2] Oscillation Project with Emulsion- tRacking Apparatus (engl.): Projekt zur Untersuchung von Neutrinooszillation mit einem lichtempfindlichen Apparat
[3] Driftröhren sind zylindrische Elektroden zur Bestimmung der Flugbahn von Elementarteilchen (Neutrinos)
[4] Time to digital converter (engl.): Zeit- Digital- Umsetzer- Diese TDCs sind Messkreise, welche kurze Zeitintervalle messen und in digitaler Form zur Ausgabe bringen
[5] First In First Out (engl.): als erstes rein als erstes raus – bezeichnet eine Art der Datenspeicherung und bedeutet, dass die ersten gespeicherten Daten auch als erstes dem Speicher wieder entnommen werden
[6] abgeleitet von mezzanino (ital.): Zwischengeschoss
[7] Block- Random Access Memory (engl.): Block mit Speicherbausteinen mit wahlfreiem Zugriff
[8] Ethernetframe (engl.): Ethernetrahmen

2 Projektdarstellung

In diesem Kapitel wird beschrieben wie das derzeit vorhandene System zur Signalverstärkung und zum Auslesen dieser Signale über das TDC- Board und das Mezzanine-Board funktionieren. Außerdem werden das genutzte Entwicklerboard, sowie die nötige Software zur Konfiguration des FPGAs vorgestellt.

2.1 Derzeitig genutzte Ausleseelektronik

Die schon in der Einleitung erwähnte Elektronik zum Messen der Driftzeiten und Aufbereitung dieser Messdaten, besteht aus dem TDC- Board mit aufgesetztem Mezzanine-Board. In Abbildung 2-1 ist die derzeit bei dem OPERA- Experiment genutzte Ausleseelektronik abgebildet.

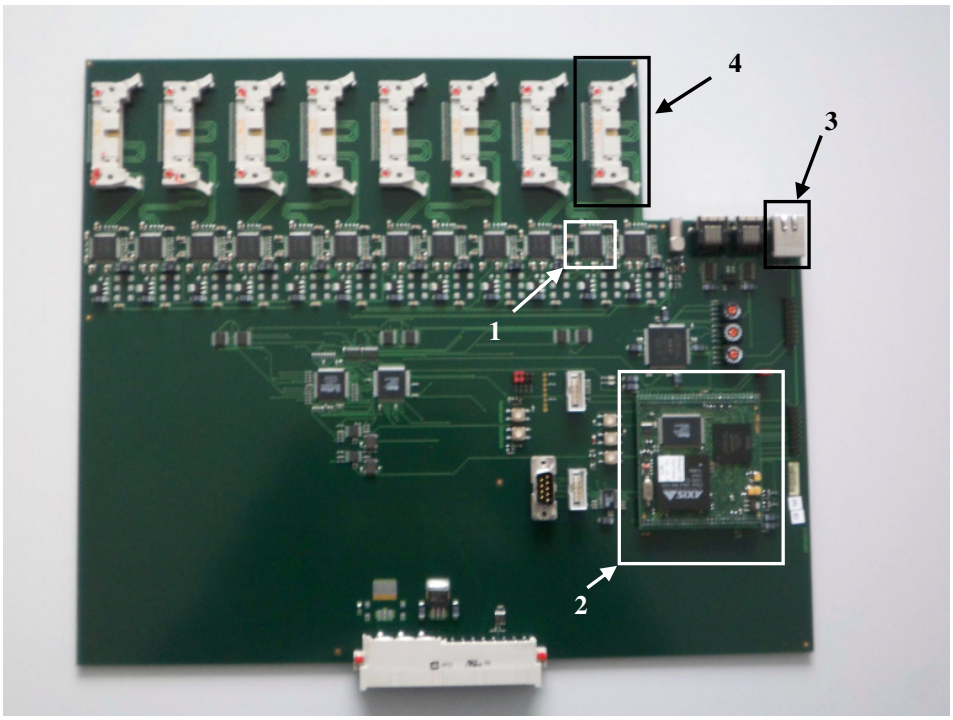


Abbildung 2-1: Derzeitig im OPERA- Experiment genutzte Ausleseelektronik

Unter ‚1' ist einer der zwölf TDC- Bausteine dargestellt, welche diesem Board auch seinen Namen geben. Die genutzten TDCs sind in Zusammenarbeit von OPERA und der Firma MSC entwickelte Multihit[9]- TDCs. Die kleinsten Messschritte dieser Messkreise betragen 0,49 ns. Nach Messung eines Impulses in einer Driftröhre, welche über 32-polige Twisted Pair[10]- Flachbandkabel am TDC- Board angeschlossen sind, werden die Daten im TDC aufbereitet und mittels eines 16 Bit breiten Datenbusses zum Mezzanine- Board gesandt. Unter ‚4' ist eine Anschlussbuchse für die Flachbandkabel von den

[9] Multihit- TDC (engl.): Mehrfachereignis- TDC – zum Messen mehrerer Ereignisse dicht hintereinander
[10] Twisted Pair (engl.): paarweise verdrillt

Driftröhren aus kommend dargestellt. ‚2' markiert das Mezzanine- Board, welches in den Hauptkomponenten aus einem ETRAX 100LX– Chip der Firma AXIS, einem FPGA der Firma Altera und einem FIFO- Speicherbaustein der Firma IDT besteht. Der ETRAX 100LX- Chip ist ein MCM[11] und besteht aus einer RISC[12] CPU mit 100 MHz Systemtakt und integriertem Betriebssystem, sowie einem 4 MByte großen Schnellspeicher, einem 16 MByte großen SDRAM[13] und einem Ethernet- Transceiver[14]. Die Hauptaufgabe des FPGAs ist die Kontrolle des Datenbusses, das bedeutet, er liest die digitalen Daten der TDCs ein und speichert diese in der richtigen Reihenfolge in den FIFO- Speicherbaustein. Die RISC- CPU liest die Daten aus dem FIFO aus und bereitet sie zum Senden über das integrierte Ethernetinterface auf. Die 8P8C- Modular-Buchse, in Abbildung 2-1 durch ‚3' markiert und allgemein auch als Ethernetbuchse bezeichnet, setzt die digitalen Signale durch interne Impulstransformatoren in analoge Signale um. Diese Signale werden dann über ein achtpoliges Twisted Pair- Kabel an eine zentrale Einheit, einen Server, gesandt und können über das Internet durch den Anwender abgerufen und analysiert werden. Da mit einem TDC- Board und integriertem Mezzanine-Board nur 96 Driftröhren ausgelesen werden können, werden für den OPERA- Detektor, welcher 9504 Driftröhren umfasst, 99 dieser Boards benötigt [1].

[11] Multi- Chip- Modul (engl.): Mehrfach- Baustein- Modul – mehrere Bausteine sind in einem Gehäuse integriert
[12] Reduced Instruction Set Computing (engl.): Rechnen mit reduziertem Befehlssatz
[13] Synchronous Dynamic Random Access Memory (engl.): synchron dynamischer Speicherbaustein mit wahlfreiem Zugriff
[14] Transceiver (engl.): Übermittler

2.2 *Projektspezifisch genutzte Hard- und Software*

Unter folgenden Punkten sind die in diesem Projekt genutzte Hard- und Software aufgeführt und näher erläutert, um dem Leser die Möglichkeit zu geben sich einen Überblick verschaffen zu können.

2.2.1 Xilinx Spartan- 3E

Zur Entwicklung eines neuen Auslesesystems, bei welchem der Hauptbaustein ein FPGA sein soll, wird das Entwicklerboard Spartan-3E der Firma Xilinx genutzt. Dieses ist in Abbildung 2-2 dargestellt.

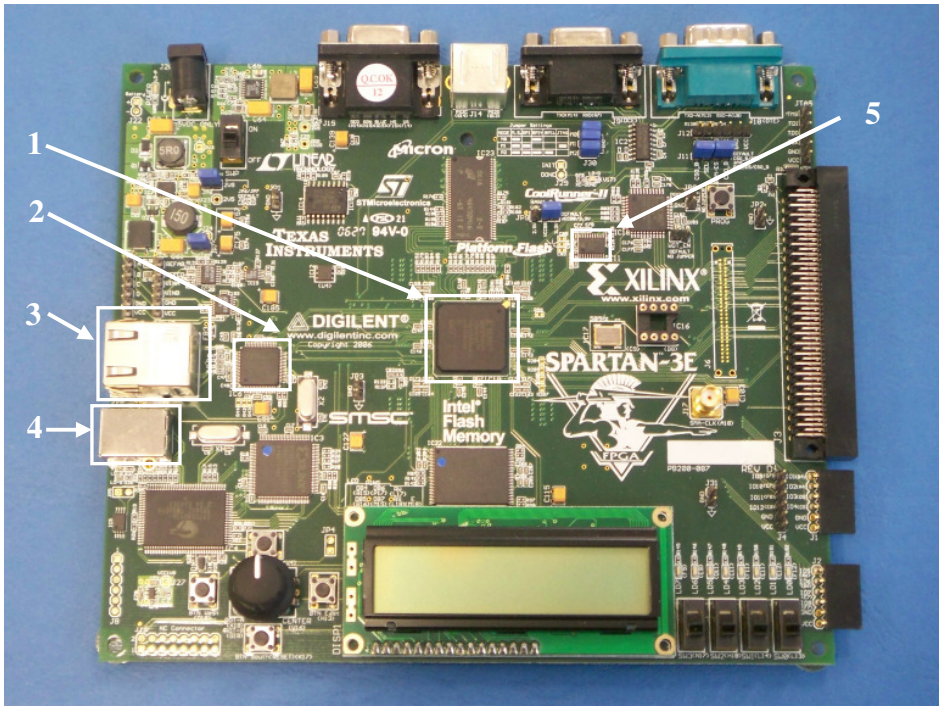


Abbildung 2-2: Entwicklerboard Spartan-3E der Firma Xilinx

Das Herzstück ist hier der FPGA XC3S500E, welcher in der Abbildung 2-2 durch ‚1' markiert ist. Der genauere Aufbau dieses Bausteins wird in dem Kapitel 5 „Grundlagen FPGA" erläutert. Das Board bietet die Möglichkeit unterschiedliche Schnittstellen, wie USB[15], Ethernet und VGA[16] sowie die Ausgabe über ein LC- Display[17] oder Leuchtdioden, zu konfigurieren und zu testen. Die nötige Hardware ist bereits implementiert und kann durch Verbinden von Leiterbahnen, was durch so genannte Jumper durchführbar ist, angesprochen werden. Die jeweiligen Jumpereinstellungen sind im Anwenderhandbuch „UG230" der Firma Xilinx beschrieben. Mittels eines USB- JTAG[18] Interfaces

[15] Universal serial bus (engl.): universeller serieller Bus
[16] Video Graphics Array (engl.): Video- Grafik- Feld – ist ein Computergrafik- Standard
[17] Liquid Crystal Display (engl.): Flüssigkristallbildschirm
[18] Joint Test Action Group, umgangssprachlicher Name für den IEEE Standard 1532-2002

kann der FPGA und auch ein so genanntes Platform Flash PROM[19] konfiguriert werden. Die USB- Buchse ist in der Abbildung 2-2 durch die ‚4' gekennzeichnet. Das Platform Flash PROM, in Entwicklerboardabbildung durch die ‚5' markiert, ist ein auf dem Board integrierter Speicherbaustein in welchem das erstellte Bitfile[20] gespeichert werden kann. Der FPGA kann also direkt oder über das Platform Flash PROM konfiguriert werden. Der Vorteil der Konfiguration über den Speicherbaustein ist eine mögliche schnelle Wiederherstellung des Systems. Bei Unterbrechung der Spannungsversorgung, beispielsweise, geht die Konfiguration des FPGAs verloren und muss erneuert werden. Dies kann durch das immer noch im Flash PROM vorhandene Bitfile durchgeführt werden. Ebenfalls sind eine Ethernetbuchse, in der Abbildung 2-2 durch die ‚3' markiert, und ein so genannter PHY[21], in der Abbildung mit ‚2' markiert, auf dem Entwicklerboard integriert. Da der PHY für eine Ethernetschnittstellenimplementierung nötig und sein Aufbau komplex ist, wird dieser im Abschnitt 2.2.2 näher erläutert. Auf alle weiteren Bausteine dieses Entwicklerboards wird hier nicht näher eingegangen, da diese für die vorliegende Arbeit nicht relevant sind.

[19] Platform Flash Programmable Read Only Memory (engl.) programmierbarer schnell ladbarer nur lesbarer Speicherbaustein
[20] Bitfile (engl.): Bitdatei - Datei mit Informationen zur Schaltungsverknüpfung
[21] Ethernet Physical Layer Transceiver (engl.): physikalischer Sendeempfänger auf Ethernetebene

2.2.2 PHY

Als PHY wird ein Chip bezeichnet, welcher die über Ethernet zu sendenden Signale moduliert und auf das genutzte Übertragungsmedium gibt. Der in dieser Arbeit genutzte Chip „LAN83C185“ wurde von der Firma SMSC entwickelt und ist auf dem genutzten Entwicklerboard integriert. In Abbildung 2-2 ist der Baustein durch die ‚2’ markiert. Der LAN83C185 übernimmt für die IEEE[22] Standards 802.3i (10BaseT) und 802.3u (100BaseTX) die nötigen Signalkodierungen– und dekodierungen um Daten senden oder empfangen zu können. In Abbildung 2-3 ist eine schematische Darstellung des Datenpfades zur Datenübertragung mittels des Ethernetstandards 100Base- TX über den genutzten PHY abgebildet.

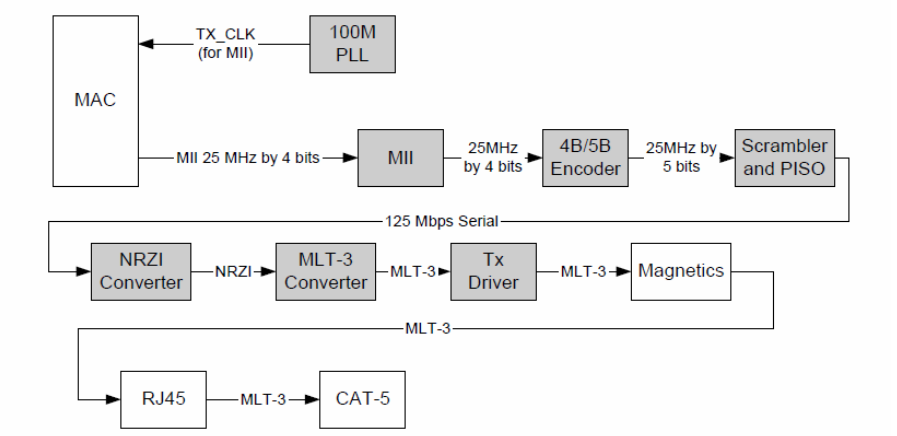


Abbildung 2-3: 100Base- TX Datenübertragungspfad [2]

Die dunkel hinterlegten Abschnitte sind digitale, durch den PHY umgesetzte, ethernet-spezifische Elemente. Die Blöcke 4B/5B- Encoder[23], Scrambling[24] und PISO[25], NRZI[26] und MLT3- Converter[27] gehören zur Signalkodierung und werden im Kapitel 4.3 näher erläutert. Durch den 100M PLL[28]- Block wird mit Hilfe eines externen veränderbaren Quarzes ein Takt von 125MHz generiert, welcher zum Antreiben der nach dem 4B/5B-Encoder- Block folgenden Logik nötig ist. Der MII[29]- Block ist das Interface zwischen Ethernetcontroller und Signalkodierung. Erst wenn Daten eingehen, soll auch eine Signalkodierung stattfinden. Der TX Driver, auch als Transmit Driver[30] bezeichnet, gene-

[22] Institute of Electrical and Electronics Engineers (engl.): Institut für Ingenieure der Elektrotechnik – dies ist ein Berufsverband aus Ingenieuren und Informatikern bestehend, welche informationstechnische Verfahren standardisieren
[23] 4- Bit zu 5- Bit- Kodierung
[24] Scrambling (engl.): Verwürfelung
[25] Parallel In Serial Out (engl.): Parallel rein seriell raus
[26] Non Return to Zero Invert (engl.): keine Rückkehr zur Null invertiert
[27] Multilevel Transmission -3 (engl.): Übertragung auf mehreren Ebenen
[28] Phase Locked Loop (engl.): Phasenregelschleife
[29] Media Independent Interface (engl.): Mediumunabhängige Schnittstelle
[30] Transmit Driver (engl.): Treiber zur Signalübertragung

riert zu dem Eingangssignal ein komplementäres Signal und sendet diese beiden Signale weiter.

Die hell hinterlegten Abschnitte sind nicht im PHY integrierte, sondern externe Steuerungen oder Bauelemente. Der hier als MAC [31] bezeichnete Block stellt den Ethernetcontroller dar, welcher als Eingang das Taktsignal des externen Quarzes, nicht mit dem im 100M PLL- Block generiertem Taktsignal zu verwechseln, besitzt. Der Ausgang dieses Blocks besteht aus einem Datenbus, an welchen die zu übertragenden Daten angelegt werden, einem Signal zur Steuerung des MII- Blocks und einem Signal zur Meldung von Fehlern bei der Datenübertragung. Der Block Magnetics[32] ist ein externer Baustein und ist für die Transformation digitaler in analoge Signale zuständig. Dieses Bauelement ist meist in den Ethernetbuchsen integriert. Diese Buchse, sowie das CAT5- Kabel, welches hier als Übertragungsmedium genutzt wird, werden im Kapitel 4.3.1 näher erläutert.

2.2.3 ISE Webpack

Das ISE[33] Webpack, ist eine von der Firma Xilinx entwickelte Softwareumgebung zur Konfiguration von FPGAs. Mittels der Hardwarebeschreibungssprachen VHDL und Verilog, welche in den Kapiteln 5.2.1 und 5.2.2 näher erläutert werden, kann ein Quelltext erstellt werden. Dieser beschreibt den zu integrierenden digitalen Schaltkreis und wird durch ISE Webpack in ein so genanntes Bitfile umgesetzt, durch welches über das USB- JTAG- Interface die Konfiguration des FPGAs vorgenommen wird.

In Abbildung 2-4 ist ein so genannter Designablaufplan dargestellt, der zeigt in welchen Schritten aus einem Quelltext ein Bitfile generiert wird. Da die Anschlüsse eines FPGAs bis zu einem gewissen Grad von dem Programmierer als Eingangs- oder Ausgangsanschluss festgelegt werden können, ist bei der Konfiguration auch die Prüfung der Anschlussbelegung nötig. In dem Vorgang, welcher hier als Pin-Planning[34] bezeichnet wird, ist es möglich die Anschlussbelegung zu prüfen und manuell zu ändern. Darauf folgt die Implementierung des Schaltkreises und Anschlussplans auf einem gespeicherten, nicht realen FPGA, um Analysen durchführen zu können. Durch die Analysen werden Informationen über die FPGA spezifischen Elemente aufgezeigt. Beispielsweise erhält der Programmierer die Anzahl genutzter Anschlüsse, sowie die Nutzung vorhandene Gatter und eine Zeitangabe für den kritischen Signalpfad. Dieser kritische Pfad beschreibt den zeitintensivsten Weg, welchen ein Signal von einem Eingang bis zu einem Ausgang durch den digitalen Schaltungskreis gehen muss. Nach dieser

[31] Media Access Control (engl.):Medienzugriffskontrolle
[32] Magnetics (engl.): Impulstransformator
[33] Integrated Software Environment (engl.): Integrierte Software Umgebung
[34] Anschlussplanung

Ergebnisanalyse, können der Schaltkreis oder die Anschlussbelegung optimiert, oder das Bitfile erstellt und geladen werden.

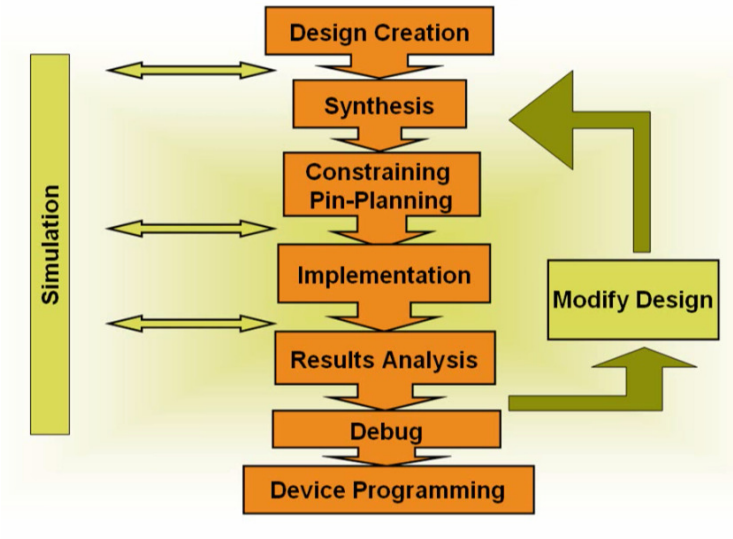


Abbildung 2-4: Designablaufplan zur Konfiguration eines FPGAs

2.2.4 Sonstige Software

Die Synthetisierung des Quelltextes und das Erstellen eines Bitfiles werden durch ISE Webpack 11 durchgeführt. Bevor allerdings die Konfigurierung stattfinden kann, sollten Simulationen des Schaltkreises durchgeführt werden. Hierzu können ISE Webpack oder das von der Firma Mentor Graphics Corporation entwickelte Programm ModelSim genutzt werden. In dieser Arbeit werden Schaltungen mit dem Programm ModelSim PE Student Edition 6.5b simuliert, worauf im Abschnitt 6.2 näher eingegangen wird.

Zum Testen des Sendens eines Ethernetframes wird das Programm Wireshark genutzt. Hiermit können Daten, welche über die Netzwerkkarte eingespeist werden, zur Anzeige gebracht werden. Dieses Programm ist im Internet kostenlos erhältlich [3] und läuft unter den gängigen Betriebssystemen wie Windows oder Linux. Eine nähere Erläuterung zu diesem Programm befindet sich im Kapitel 8.

Um überprüfen zu können ob das Programm Wireshark auch reine Ethernetframes zur Anzeige bringt, wird zur Generierung eines Ethernetframes das Programm PakETH, welches im Internet [4] erhältlich ist, genutzt. Durch Angabe der Ziel- und Quelladresse der entsprechenden Stationen kann ein Ethernetframe generiert und auf das Übertragungsmedium gegeben werden.

3 Grundlagen der Netzwerkübertragung

In diesem Kapitel werden die Grundlagen über die Verbindung mehrerer Rechnerstationen, also eines Netzwerks dargelegt. Eingegangen wird hierbei auf die verschiedenen Formen der Verbindungen, die so genannten Netzwerktopologien, das standardisierte Kommunikationsverbindungsmodell, welches auch als OSI- Referenzmodell bezeichnet wird, sowie auf einige Protokolle, welche die Art der Übertragung der Daten festlegen. Ziel dieses Kapitels ist es mögliche Arten der Datenübertragung und gewisse Risiken bei Signalübertragungen kennen zu lernen.

3.1 Netzwerktopologien

Rechnerstationen, welche Sender und Empfänger sein können, müssen zum Zweck eines Datenaustausches miteinander über feste Verdrahtung, Funk- oder optische Verbindung miteinander verknüpft werden. Die Form dieser Verknüpfung wird Netzwerktopologie genannt und kann durch fünf verschiedene Modelle umgesetzt werden:

- Die **Punkt-zu-Punkt-Verbindung** ist die am leichtesten umzusetzende Form. Jede Station ist mit jeder anderen direkt verbunden. Somit müssen keine Stationsnamen oder Stationsadressen zum filtern des richtigen Empfängers gesandt werden, da anhand der Auswahl der Verbindung festgelegt ist, welche Station die Daten zugesandt bekommt. Ein weiterer Vorteil ist die Schnelligkeit mit der Daten übertragen werden können, diese hängt hier nur von Art und Länge des Übertragungsmediums ab. Zusätzlich fällt nicht die gesamte Kommunikation bei einer Unterbrechung einer Verbindung zusammen, sondern nur die der zwei dort angeschlossenen Stationen.

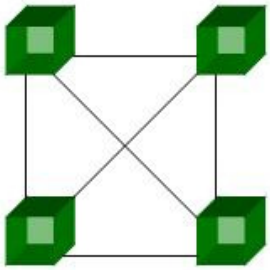


Abbildung 3-1: Punkt-zu-Punkt-Verbindung

- Die **Bus-Topologie** verwendet entweder nur ein Übertragungsmedium an welchem mehrere Stationen angeschlossen werden können, oder Stationen, die untereinander verbunden sind. Dies ist in Abbildung 3-2 dargestellt. Bei der ersten möglichen Bustopologie müssen an den Enden des Busses Abschlusswiderstände installiert werden. Bei einem offenen Leitungsende würden die ankommenden elektrischen Signale reflektiert und es würde zu einer Überlagerung dieser

kommen, was eine nachhaltige Störung der gesendeten Daten zur Folge hat. Diese Bus-Topologie ist kostengünstig installierbar, da nur eine Hauptleitung verlegt werden muss und an jeder Stelle eine Station angeschlossen werden kann. Somit ist das Netz leicht erweiterbar. Die Kommunikation ist allerdings etwas komplizierter als bei der Punkt-zu-Punkt-Verbindung, da die Stationen über nur ein Übertragungsmedium Daten austauschen. Daher muss jede Station eine eindeutige Adresse besitzen, um filtern zu können für welchen Empfänger die Daten bestimmt sind, und es dürfen nicht mehrere Stationen gleichzeitig senden. Um dies zu koordinieren, kann eine zentrale Einheit zur Steuerung eingesetzt werden. Diese Steuerungseinheit wird Bus-Arbiter genannt und kann von jeder Station aus über eine externe Leitung angesprochen werden. Das heißt, hat ein Sender Daten zu verschicken, muss eine Anfrage an den Bus-Arbiter gesendet werden. Sobald dieser das Versenden erlaubt, also das Übertragungsmedium frei ist, kann der Sender die Daten übertragen. Eine andere Möglichkeit der Kollisionsvermeidung ist das Carrier Sense Multiple Access/ Collision Detection[35]-Verfahren. Hier überprüft jede Station selbständig ob Daten gesendet werden können, indem die Hauptleitung „abgehorcht“ wird. Der Sender muss mit dem Verschicken seiner Daten warten bis die Leitung frei ist. Das CSMAD/CD-Verfahren wird in Kapitel 4.4 näher erläutert.

Ein Nachteil bei nur einer verlegten Hauptleitung ist die eingeschränkte oder sogar ganz unterbrochene Kommunikation unter den Stationen, wenn das Übertragungsmedium beschädigt ist. Liegt zum Beispiel ein Kabelbruch vor, wird so das Netz in zwei Teilnetze gegliedert und es können nur noch die Stationen des jeweiligen Teilnetzes Daten austauschen. Außerdem würde es zu Reflexionen der Signale am „gebrochenen“ Leitungsende kommen, da hier kein Abschlusswiderstand angeschlossen ist.

Die Erweiterung des Netzes um neue Stationen ist leicht möglich, da an jeder beliebigen Stelle der Leitung ein T-Stück[36] eingesetzt und eine Station angeschlossen werden kann. Der Nachteil liegt hier in der Unterbrechung des Netzes bei der Installation. Der Datenbus kann erst wieder in Betrieb genommen werden, sobald die Erweiterung abgeschlossen ist.

Die zweite mögliche Bustopologie besteht zwar aus mehreren Übertragungsmedien, dennoch ist sie kostengünstiger zu installieren als die Punkt- zu- Punkt-Verbindung. Außerdem gibt es hier keine offenen Leitungsenden, daher kann auch auf Abschlusswiderstände verzichtet werden. Dagegen werden aber für die mittleren Stationen immer zwei Anschlüsse für die Leitungen benötigt. Ein

[35] CSMA/CD (engl.): Trägerabfrage bei Mehrfachzugriff / Kollisionserkennung
[36] Verbindungsglied mit einer Abzweigung

Nachteil bei einer solchen Installation ist, dass der Bus nicht nur durch defekte Leitungen, sondern auch durch eine defekte Station unterbrochen werden kann.

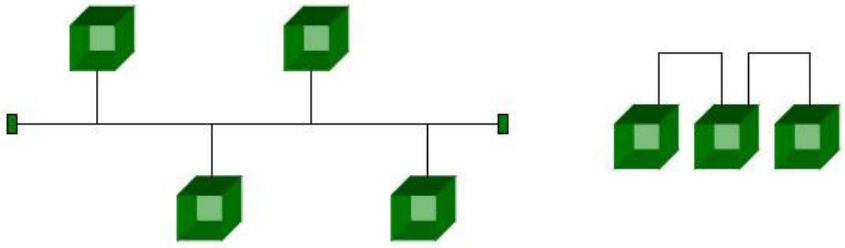


Abbildung 3-2: Bus-Topologien

- Die **Ring-Topologie** besteht aus mehreren Stationen die jeweils mit der nächsten Station verbunden sind. Dadurch dass die letzte Station mit der ersten verknüpft wird entsteht ein Ring mit einer Datenhauptleitung über die von jedem Sender aus jede Station erreicht werden kann. Dieses Netzmodell ist allerdings sehr fehleranfällig, da bei Unterbrechung einer Verbindung die gesamte Kommunikation gestört wird. Der Vorteil liegt in der vereinfachten Fehlerbehebung. Da die Daten immer nur in eine Richtung auf das Übertragungsmedium gegeben werden, kann die Station, welche keine Daten mehr über den Ring empfängt, schnell ausfindig gemacht werden. Zur Erweiterung des Netzes ist die Unterbrechung dieses nötig, was wieder zu einer eingeschränkten Kommunikation führt.

  Um die Einschränkung der Kommunikation bei Unterbrechung des Netzes zu vermeiden werden auch Ring-Topologien mit zwei Ringen installiert. Dadurch können leicht neue Stationen hinzugefügt und bei einer Störung des Übertragungsmediums kann trotzdem kommuniziert werden. Dieses Verfahren ist allerdings kostenintensiver als die einfache Ring-Topologie.

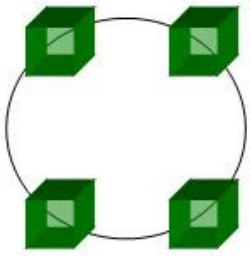


Abbildung 3-3: Ring-Topologie (einfacher Ring)

- Die **Stern-Topologie** ist ein Netzwerkmodell mit einer zentralen Verteilereinheit, an der jede Station direkt angeschlossen werden kann. Der Installations- und Planungsaufwand ist gering, da die Kommunikation bei einer Netzwerkerweiterung nicht unterbrochen werden muss. Der Kostenaufwand ist höher als bei anderen Topologien, da von jeder Station aus ein Übertragungsmedium zur Verteilereinheit verlegt werden muss. Die Verteilereinheit kann ein Hub[37] oder ein Switch[38] sein, welche folgend näher erläutert werden.

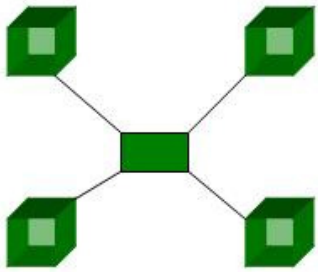


Abbildung 3-4: Stern-Topologie

Bei allen Topologien ist es beim Verlegen des Übertragungsmediums über mehrere Meter nötig die gesandten Signale zu verstärken. Jeder Kabeltyp besitzt einen anderen Wellenwiderstand, durch den die elektrischen Signale über Strecken hin abgeschwächt werden. Auf Kabeltypen, ihren Wellenwiderstand, die Ausbreitungsgeschwindigkeit und maximale Netzwerkausbreitung wird für die Ethernetstandards im Kapitel 4.2 näher eingegangen. Folgend werden nur die gängigen Signalverstärker und Verteiler, die in einem Netzwerk nötig sein können, vorgestellt.

In der schon vorgestellten Bus-Topologie ist es bei einer Netzausbreitung von einigen hundert Metern nötig die gesandten elektrischen Signale zu verstärken, da diese sonst durch den kabelspezifischen Wellenwiderstand so abgeschwächt werden können, dass beim Empfänger kein lesbares Signal ankommt. Diese Signalverstärkung wird von einem Signalgenerator, dem so genannten **Repeater**[39] übernommen. Dieser hat zwei Anschlüsse, welche bidirektional als Ein- oder Ausgang betrieben werden. Ein Repeater bereitet das Eingangssignal auf und sendet es dann über seinen Ausgang weiter, er hat keine regulierende Funktion in einem Netzwerk und ist transparent. Ein guter Nebeneffekt beim Anschluss eines Repeaters in einer Bus-Topologie ist die Teilung des Netzes in zwei voneinander unabhängige Teilnetze. Alle Stationen eines Teilnetzes können beim Ausfall eines anderen Teilnetzes miteinander kommunizieren. Besitzt ein Repeater mehr als zwei Anschlüsse wird er als **Multiport**[40]**-Repeater** oder auch **Hub** bezeichnet. Ein Hub ist ein Netzwerkknotenpunkt, durch den die Stern-Topologie oder der Zusam-

[37] hub (engl.): Nabe, Knotenpunkt
[38] switch (engl.): Schaltgerät, Weiche
[39] repeater (engl.): Wiederholer
[40] multiport (engl.): Mehrkanal

menschluss verschiedener Netzwerkmodelle realisiert werden kann. Die ankommenden Signale werden wie bei einem Repeater nur elektronisch aufbereitet und dann an alle Stationen weitergeleitet. Die Stationen müssen selbst die für sie bestimmten Daten filtern. Beim Einsatz eines **Switches**, auch als intelligenter Hub bezeichnet, werden durch diesen die Daten nicht nur aufbereitet, sondern auch gefiltert. Die ankommenden Signale werden nur an den bestimmten Empfänger weitergesandt. Mittels einer in einer Tabelle gespeicherten eindeutigen Adresse und Anschlussbelegung für jede Station kann der Switch den Datenfluss kontrollieren.

3.2 OSI-Referenzmodell

Das OSI-Referenzmodel[41] wurde von der ISO[42] zwischen 1977 und 1984 entwickelt und ist seit 1984 als Standardnetzwerkmodell für die Kommunikation innerhalb eines Netzwerks festgelegt [5]. Das Modell ist in sieben aufeinander aufbauende Schichten, welche auch als Layer bezeichnet werden, unterteilt. Jede Schicht beschreibt einen Dienst, der bei der Übertragung von Daten in einem Netzwerk ausgeführt und eingehalten werden muss. Sender und Empfänger sollten im Einklang miteinander stehen, was die Art der Übertragung von Daten betrifft. Dies bedeutet, es muss beispielsweise festgelegt werden, in welcher Reihenfolge die Signale übertragen werden, oder ob sie vor dem Senden codiert wurden. Der gesamte Ablauf einer Datenübertragung in einem Netzwerk ist mittels der sieben Schichten im OSI-Referenzmodell festgelegt. Folgend wird das Modell mit seinen Schichten graphisch dargestellt und anschließend erläutert.

OSI-Referenzmodell		
OSI-Schicht		Einheit
7	Anwendungsschicht (Application Layer)	Daten
6	Darstellungsschicht (Presentation Layer)	Daten
5	Sitzungsschicht (Session Layer)	Daten
4	Transportschicht (Transport Layer)	Segmente
3	Vermittlungsschicht (Network Layer)	Pakete
2	Sicherungsschicht (Data Link Layer)	Rahmen (Frames)
1	Bitübertragungsschicht (Physical Layer)	Bits

Abbildung 3-5: OSI-Referenzmodell

- Die **Bitübertragungsschicht** wird auch als Physical Layer bezeichnet und ist die erste Schicht im OSI-Referenzmodell. Hier wird die physikalische und elektrotechnische Datenübertragung festgelegt. Es wird beschrieben, in welcher physikalischen Form das Signal über welches Übertragungsmedium gesendet wird. Der Aufbau und die Übertragung eines Bits werden festgelegt, daher wird die Einheit in dieser Schicht als Bits bezeichnet. Des Weiteren werden die physikalischen Anschlüsse und die Art der verwendeten Verbindungselemente definiert.
- In der **Sicherungsschicht** wird der zu übertragende Datenstrom in Abschnitte unterteilt, welche mehrere Bytes umfassen. Diesen Abschnitten, oder auch Da-

[41] Open- Systems- Interconnection (engl.): Systemunabhängiges Kommunikations- Referenzmodell
[42] International Organization for Standardization (engl.): Internationale Organisation für Standardisierung

tenpakete genannt, wird jeweils eine Prüfsumme angehängt, um ermitteln zu können, ob ein gesendetes Paket korrekt übermittelt wurde. Die Prüfsumme wird anhand der Datenbits errechnet. So werden hier nicht nur Nutzbits, also Bits welche die zu übertragende Nachricht darstellen, sondern auch redundante Daten gesandt. Aus diesem Grund wird in der zweiten Schicht des OSI-Referenzmodells nicht von Bits sondern von Frames[43] als Einheit gesprochen. Die Sicherungsschicht wird auch als Data Link Layer bezeichnet und ist für die fehlerfreie Übertragung zuständig. Dies wird durch Quittier- und Wiederholmechanismen realisiert. Das bedeutet, hat ein Empfänger einen Frame erhalten, schickt dieser ein Signal an den Sender, quittiert also den Empfang. Erhält ein Sender nach der Übertragung keine Quittierung wiederholt dieser das Senden des Frames. In der Praxis wird die Sicherungsschicht in zwei Unterschichten, so genannte Sub Layer, gegliedert. Der Sub Layer, der an die Bitübertragungsschicht grenzt, wird von dem MAC[44] ausgeführt, welcher den Zugriff auf das Übertragungsmedium regelt. Der übergeordnete Sublayer[45] grenzt an die Vermittlungsschicht, wird vom LLC[46] ausgeführt und ist für die Datensicherung auf der Verbindungsebene zuständig. Das MAC und das LLC werden in dem Kapitel Netzwerkprotokolle näher erläutert.

Abbildung 3-6: Teilung der Sicherungsschicht

- Die **Vermittlungsschicht**, auch Netzwerkschicht oder Network Layer genannt, dient der Vermittlung zwischen Sender und Empfänger. Die dritte Schicht des OSI-Referenzmodells ist für die Wahl der Datenwege, das Routing[47], zuständig. Hierzu werden Datenpakete mit Ziel- und Quelladresse versehen, welche Aufschluss darüber geben, in welchem Netzwerk sich die Quelle und das Ziel befinden. Außerdem werden falsche Datenpaketreihenfolgen korrigiert sowie Duplikate erkannt und beseitigt.

- Bei der **Transportschicht** handelt es sich um einen Vermittler zwischen den ankommenden Datenpaketen und den Anwenderprogrammen der Stationen. Die Daten werden hier in Datensegmente unterteilt, damit diese durch die Vermitt-

[43] Frames (engl.): Rahmen
[44] Media Access Control (engl.): Medienzugriffskontrolle
[45] Sublayer (engl.): Teilschicht
[46] Logical Link Control (engl.): logische Verbindungskontrolle
[47] Routing (engl.): rangieren

lungsschicht weitergeleitet werden können. Auf der anderen Seite werden die Datensegmente in der richtigen Reihenfolge wieder zu einem gesamten Datenblock zusammengesetzt und an die Sitzungsschicht übergeben.

- Die **Sitzungsschicht**, auch Kommunikationssteuerungsschicht genannt, steuert den Aufbau, die Durchführung und das Beenden einer Verbindung. Bei einem Zusammenbruch einer Verbindung, welche auch als Session bezeichnet wird, ist die Sitzungsschicht dafür zuständig, eine neue Verbindung aufzubauen.

- Die **Darstellungsschicht** sorgt durch spezielle Dienste für die Aufarbeitung der Daten in ein bestimmtes Format. Hier wird festgelegt wie die Daten dem Anwender präsentiert werden.

- Die **Anwendungsschicht** ist die Verbindung zwischen dem Anwender und den netzwerkspezifischen Prozessen. Wie in den Schichten fünf und sechs werden auch hier ganze Datenblöcke und nicht Segmente oder einzelne Bits bearbeitet.

Das OSI-Referenzmodell wird bei einer Datenübertragung zweimal durchlaufen. Der erste Durchlauf wird auf der Sender-Seite vollzogen. Die zu übertragenden Datenblöcke müssen in Segmente, Frames und schließlich in einen Bitstrom gewandelt werden. Währenddessen muss der Empfänger im Netzwerk lokalisiert und eine Verbindung zu ihm aufgebaut werden. Sind die Signale beim Empfänger eingetroffen, muss eine Empfangsbestätigung an den Sender übermittelt und die Daten nach Eingang aller Datenpakete wieder zu einem Datenblock zusammengesetzt und an das richtige Anwenderprogramm übergeben werden.

3.3 Netzwerkprotokolle

Um Daten von einer Station zu einer anderen in einem Netzwerk übertragen zu können, müssen diese dieselben Verfahren zum Senden und Empfangen der Daten verwenden. Sie müssen auf einer Ebene miteinander kommunizieren. In einem eigenen lokalen Netzwerk ist dies leicht zu realisieren, da selbst festgelegt werden kann, wie Nachrichten übertragen, segmentiert und kodiert werden. Sind aber Sender und Empfänger in unterschiedlich strukturierten Netzen, welche nur über einen Knoten miteinander verbunden sind, stationiert und durch verschiedene Rechnersysteme betrieben, ist die Kommunikation deutlich schwieriger. Damit aber auch hier Daten übertragen werden können, gibt es allgemeine Regeln wie die Dienste in jeder Schicht des OSI-Referenzmodells zu arbeiten haben und in welcher Form die Daten den angrenzenden Schichten zur Verfügung gestellt werden müssen. Diese Regeln sind in so genannten Netzwerkprotokollen festgehalten. Es gibt für jede Schicht des OSI-Referenzmodells zur Verfügung stehende Protokolle, einige decken nur eine Schicht, andere auch zwei Schichten ab. Folgend werden einige gängige Protokolle erläutert, um einen Überblick

zu erlangen. Die Erläuterung wird in den obersten Schichten des OSI-Refrenzmodells beginnen, wobei das Hauptaugenmerk auf die Bitübertragungsschicht und die Sicherungsschicht gelegt wird, da hier das Ethernet-Protokoll angesiedelt ist, welches im Kapitel 4 näher erläutert wird.

Die meisten Protokolle, welche in der obersten Schicht des OSI-Referenzmodells angesiedelt sind, umfassen alle Dienste der Anwendungs-, Darstellungs- und Sitzungsschicht. Folgende Abbildung veranschaulicht die gängigsten Protokolle der obersten drei Schichten [6].

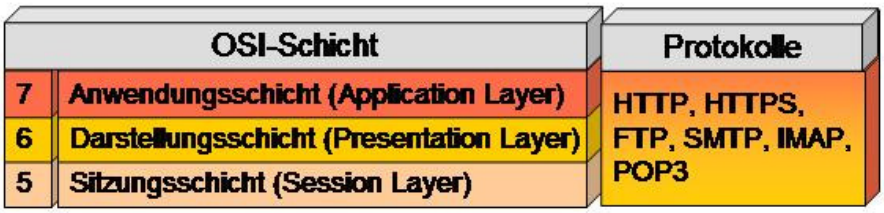


Abbildung 3-7: Protokolle der Schichten 5-7

- Das **HTTP**[48] ist das Standard Transfer Protokoll und hardware- und betriebssystemunabhängig. Der Dienst des HTTP wird von einem Anwenderprogramm, meist einem Webbrowser[49], angesprochen. Durch die Eingabe einer URL[50] im Webbrowser wird der Dienst des HTTP ausgelöst, welcher anhand der Eingabe eine Anfrage an eine Station sendet, welche die gesuchten Daten enthält, und diese bittet, die Daten zurückzusenden. Konnte eine Verbindung aufgebaut werden und hat diese Station die Daten gespeichert, werden diese an das anfragende System gesandt und im Webbrowser dargestellt und die Verbindung wieder getrennt. Die Daten werden meist über mehrere Netzwerkknoten oder Rechner gesandt, da Sender und Empfänger oftmals nicht in einem lokalen sondern überregionalen Netzwerk stehen. Beim Hypertext Transfer Protocol können die gesendeten Nachrichten von jedem durchlaufenden Knotenpunkt und Computer gelesen werden. Um dies zu verhindern, wurde eine erweitete Version des Protokolls, das HTTPS[51], entwickelt, über welches die Daten verschlüsselt gesandt werden. [6]

[48] Hypertext Transfer Protocol (engl.): Hypertext Übertragungsprotokoll
[49] Anwenderprogramm zum Suchen von Daten im globalen Netzwerk
[50] Uniform Ressource Locator (engl.): einheitlicher Quellenanzeiger - lokalisiert Datenspeicher in Netzwerken
[51] Hypertext Transfer Protocol Secure (engl.): sicheres Hypertext Übertragungsprotokoll

- Ein weiteres, die Schichten fünf bis sieben abdeckendes, Protokoll ist das **FTP**[52]. Dieses stellt die Regelung dar, eine Datei über ein Netzwerk senden zu können und Verzeichnisse auf einer anderen Station zu verwalten.

- Das **SMTP**[53] ist für das Senden und Einspeisen von Daten in Form von E-Mails zuständig.

- Die Pendants zum SMTP, sind das **IMAP**[54] und das **POP3**[55]. Diese Protokolle sind für das Abholen von E-Mails zuständig. Durch das POP3 ist der Anwender in der Lage eine Verbindung zur Station mit den gespeicherten E-Mails, dem E-Mail-Server, aufzubauen und dort die elektronische Post abzurufen und zu löschen. Das IMAP ist hingegen etwas komfortabler, hiermit können E-Mails abgerufen, gelöscht und verwaltet werden.

Folgende Abbildung stellt die nächste OSI-Schicht, die Transportschicht, und die gängigen Protokolle, welche die nötigen Dienste anbieten, dar.

Abbildung 3-8: Protokolle der Schicht 4

- Das **UDP**[56] übernimmt die Übermittlung der Daten an das entsprechende Anwenderprogramm auf dem Zielrechner. Dieses Protokoll arbeitet verbindungslos, was zur Folge hat, dass nicht gesichert ist ob überhaupt alle Datenpakete oder ob diese in der richtigen Reihenfolge beim Empfänger angekommen sind. Daraus resultiert aber auch ein Vorteil, denn durch das fehlende Hand-Shake-Verfahren[57] ist die Übertragung deutlich schneller.

- Das **TCP**[58] ist ein verbindungsorientiertes und somit zuverlässiges Transportprotokoll. TCP baut im Gegensatz zu UDP eine feste Verbindung, sozusagen einen Kanal, zwischen den kommunizierenden Stationen auf. Hier kommt auch das Hand-Shake-Verfahren zum Einsatz. Das bedeutet, ein Sender übermittelt an

[52] File Transfer Protocol (engl.): Dateiübertragungsprotokoll
[53] Simple Mail Transfer Protocol (engl.): einfaches Mailübertragungsprotokoll
[54] Internet Message Access Protocol (engl.): Internet Nachrichtenzugriffsprotokoll
[55] Post Office Protocol 3 (engl.): Postfach- Protokoll
[56] User Datagram Protocol (engl.): Anwender Datagramm Protokoll
[57] Verfahren in der Datenübertragung, bei dem nach dem Empfang der Daten ein Quittierungssignal an den Sender übermittelt wird
[58] Transmission Control Protocol (engl.): Übermittlungskontrollprotokoll

den Empfänger ein Signal, welches anzeigt, dass er sendewillig ist. Der Empfänger schickt eine Antwort, dass er empfangsbereit ist. Nachdem die Antwort des Empfängers angekommen ist, beginnt die eigentliche Datenübertragung, welche nach Beendigung quittiert wird. Ist ein Datenpaket nicht übertragen worden, also ist keine Quittierung des Empfängers angekommen, wird das Datenpaket erneut verschickt. Dieses Verfahren macht die Kommunikation sicherer, aber langsamer als UDP.

- Das **SCTP**[59] vereint die Vorteile von UDP und TCP, Schnelligkeit und Sicherheit. Wie TCP arbeitet auch SCTP mit dem Hand-Shake-Verfahren, das heißt jedes gesandte Datenpaket wird quittiert, falls dies nicht geschieht wird der Versandt neu gestartet. Ebenso wird ein so genannter Kommunikationstunnel aufgebaut, über den sicher kommuniziert werden kann. Trotz dieser Sicherheit ist der Transfer mit dem SCTP schnell, da nicht nur ein Kommunikationstunnel aufgebaut wird, sondern mehrere, über die zur selben Zeit Daten versandt werden können.

Die zweite Schicht des OSI-Referenzmodells wird, wie schon erwähnt, nochmals in zwei Schichten unterteilt. Folgend werden die Protokolle für die zwei Sub Layer dargestellt und erläutert.

Abbildung 3-9: Protokolle der Schichten 2a und 2b

- Das **Logic Link Control** ist in der oberen Hälfte der Sicherungsschicht angesiedelt. Das **LLC** Protokoll wurde durch das IEEE unter 802.2 standardisiert und ermöglicht höheren Protokollen, sich eine physische Verbindung zu teilen.

- Die untere Schicht des zweiten Layers im OSI-Referenzmodell wird durch das **Media Access Control** Protokoll beschrieben und ist für die Kollisionsvermeidung der Daten auf dem Übertragungsmedium zuständig. Da in einem Netz mehrere Stationen gleichzeitig das Senden von Daten anweisen können, kann es auf der Leitung zu Kollisionen und dadurch zu Störungen und Datenverlust kommen. Um dies zu vermeiden gibt es zwei Verfahren, die **kontrollierte** und die **konkurrierende Kollisionsvermeidung**. Bei der kontrollierten Kollisions-

[59] Stream Control Transmission Protocol (engl.): Datenstrom- Übermittlungskontrollprotokoll

vermeidung wird ein zusätzlicher Kommunikationskanal hergestellt, über welchen eine Station anmelden kann, dass sie Daten übertragen möchte. Das Senderecht wird dieser zugeteilt und für alle anderen gesperrt. Die konkurrierende Kollisionsvermeidung lässt zwar gleichzeitiges Senden zu, aber sobald dies geschieht wird die Datenübertragung aller beteiligten Stationen abgebrochen. Nach einer zufälligen Wartezeit, für jede Station unterschiedlich, wird der Sendeversuch erneut ausgeführt.

Die erste Schicht im OSI-Referenzmodell ist die Bitübertragungsschicht, welche das Umsetzen eines Informationsbits in ein elektrisches Signal und das Übertragungsmedium beschreibt. Folgend sind die Schicht und deren zugehörigen gängigen Protokolle dargestellt.

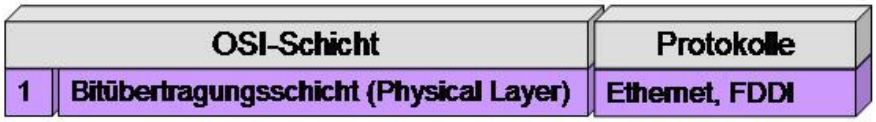


Abbildung 3-10: Protokolle der Schicht 1

- Das **FDDI**[60] besitzt als Übertragungsmedium das Glasfaserkabel und arbeitet mit einer Datenübertragungsrate von 100, 155 und 1000 MBit/s. Als Netztopologie wird die Ringstruktur in einfacher oder doppelter Ausführung gewählt. Die Reichweite eines Netzes kann bis zu 200 Kilometer betragen, wobei bis zu 1000 Stationen angeschlossen werden können.

Das **Ethernet**, welches die derzeit größte Bedeutung durch Schnelligkeit und geringe Kosten besitzt, wird im nächsten Kapitel eingehend erläutert. Daher wird an dieser Stelle nicht näher darauf eingegangen.

[60] Fiber Distributed Data Interface (engl.): Dezentralisierte Datenschnittstelle über Glasfaser

4 Grundlagen Ethernet

In diesem Kapitel soll das Protokoll Ethernet näher erläutert werden. Es wird beschrieben, wie das Ethernet entstand, welche Ethernetstandards es gibt, was in dem Reglement festgehalten ist und welche zukünftigen Entwicklungen es geben soll.

4.1 Entwicklung des Ethernets

Der Name Ethernet setzt sich zusammen aus ‚Ether', englisch für Äther, und ‚net', englisch für Netz. Historischen Annahmen nach war Äther das Medium in dem sich Wellen, so auch Funkwellen, ausbreiten. Die Existenz solch eines Mediums konnte allerdings nicht nachgewiesen werden [7].

Anfang der siebziger Jahre wurde durch Robert Metcalfe, damals Angestellter der Firma Xerox Corporation, die erste Form des Ethernets mit einer Geschwindigkeit von 2,94 MBit/s entwickelt. Abgeleitet wurde dieses Protokoll ursprünglich von dem auf Funkübertragung basierenden ALOHAnet, welches die Inseln um Hawaii mit einem Zentralrechner in Honolulu miteinander verband. Dieses Netz hatte zwei Kanäle zur Verfügung, wobei ein Kanal zum Übertragen der Daten und der andere zum Senden von Quittierungssignalen eingesetzt wurden. Sendeten zwei Stationen gleichzeitig kam es zur Kollision, worauf dann kein Quittierungssignal gesendet wurde. Nach einer zufällig gewählten Zeitspanne versuchten die Stationen unabhängig voneinander erneut zu senden, wie bei der im vorangegangen Kapitel schon beschriebenen konkurrierenden Kollisionsvermeidung. Bei höherem Kommunikationsaufkommen, führte dieses Verfahren zwangsläufig zu vielen Datenkollisionen, woraufhin Robert Metcalfe das Carrier Sense Multiple Access/Collision Detect– Verfahren entwickelte. Das CSMA/CD- Verfahren wird in dem Kapitel 4.4 näher erläutert. Robert Metcalfe verließ 1979 die Firma Xerox Corporation und gründete das Unternehmen 3Com. Ein Konsortium, bestehend aus den Firmen, DEC, Intel und Xerox, auch DIX-Gruppe genannt, entwickelte mit Robert Metcalfe das Ethernet für eine Datenrate von 10 MBit/s mit dem Ziel Ethernet zum Standard für Datenübertragungen in lokalen Netzen zu machen. 1980 wurde die firmenspezifische Ethernet-Version der DIX-Gruppe von den Normungsinstitut IEEE aufgegriffen und in der Arbeitsgruppe 802.3 weiterentwickelt und zum anerkannten Standard ernannt. Durch ständige Weiterentwicklung hat sich Ethernet überall auf der Welt durchsetzen können. Im folgenden Unterkapitel Ethernetstandards ist eine Tabelle aufgeführt mit einigen entwickelten Ethernetstandards, woraus auch die geschichtliche Entwicklung ersichtlich wird [8].

4.2 Ethernetstandards

Seit 1980 wurde das Ethernet immer weiterentwickelt. Die Datenübertragungsrate reicht von 10 MBit/s bis 10 GBit/s und es gibt Übertragungsmedien vom Koaxialkabel[61] bis zum Glasfaserkabel. Nicht alle Technologien konnten sich durchsetzen. Folgend wird die Namenszusammensetzung der Ethernetstandards näher erläutert. Außerdem sind vier Tabellen mit Standards für die Übertragungsraten 10, 100, 1000 MBit/s und 10 GBits/s abgebildet.

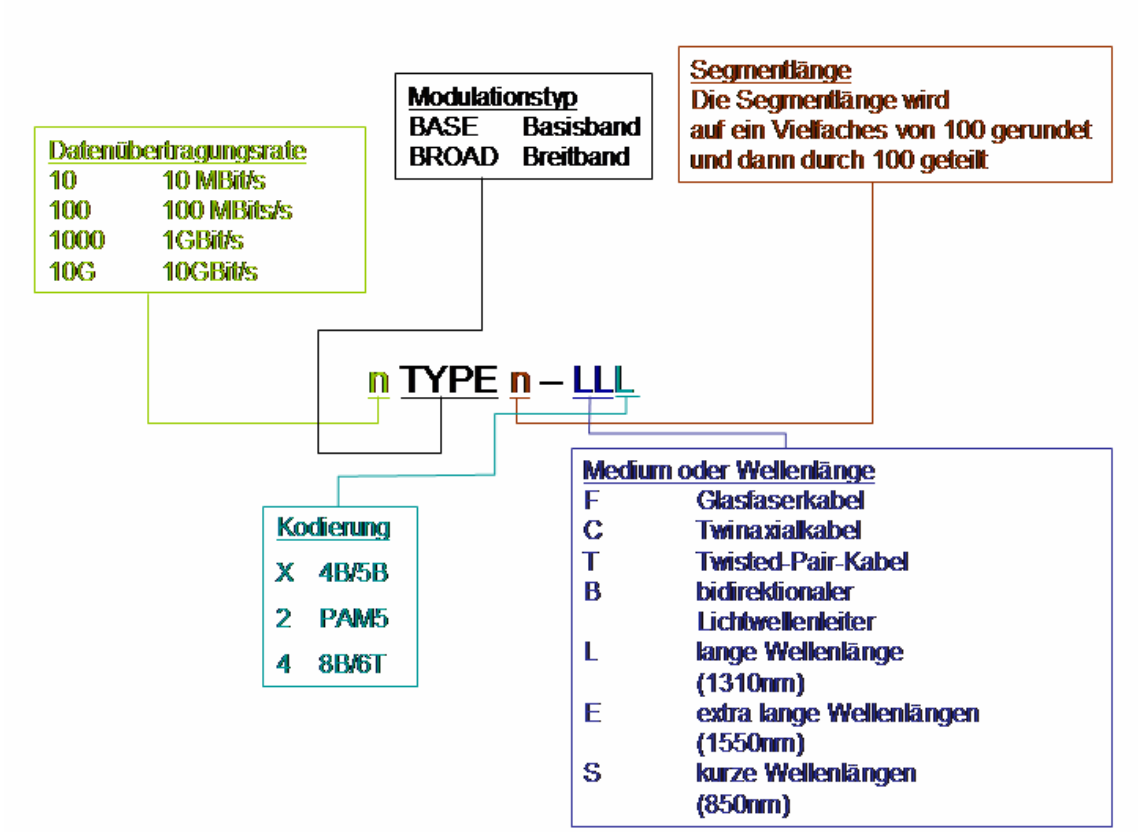


Abbildung 4-1: Namenszusammensetzung der Ethernetstandards

In Abbildung 4-1 ist die Namenszusammensetzung der Ethernetstandards dargestellt. Anhand der Ethernetbezeichnung ist ersichtlich mit welcher Übertragungsrate, Modulation und über welches Übertragungsmedium die Daten in dem bestimmten Netwerk übertragen werden. Außerdem wird die Segmentlänge, die beschreibt welchen Abstand die Stationen maximal voneinander im Netz haben dürfen, angegeben.

[61] zweipoliges Kabel mit konzentrischem Aufbau

Bezeichnung	IEEE-Norm	Jahr der Standardisierung
10Base5	802.3	1980
10Broad36	802.3a	1985
10Base2	802.3b	1988
10BaseT	802.3i	1990
10BaseFL	802.3j	1993
10BaseFB	802.3j	1993
10BaseFP	802.3j	1993

Tabelle 4-1: Ethernetstandards mit 10 MBit/s Übertragunsrate [10]

In der vorangegangenen Tabelle 4-1 sind die Ethernetstandards mit einer Übertragungsrate von 10 MBit/s gelistet. Diese Standards, zwischen 1980 und 1993 entwickelt, haben als Übertragungsmedium, Koaxialkabel, Twisted Pair- Kabel oder Glasfaserkabel. Nur die ersten drei Entwicklungen, 1980 bis 1988, besitzen eine Bustopologie und senden die Daten über das Koaxialkabel, welches allgemein nicht mehr für den Aufbau von Rechnernetzwerken genutzt wird. Abgelöst wurde dieses Medium durch das Twisted Pair- Kabel, was heute das am weitesten verbreitete Medium zur Datenübertragung in der kabelgebundenen Rechnerwelt darstellt. In Abbildung 4-2 sind die Kabelarten Koaxial, Twisted Pair und Glasfaser abgebildet.

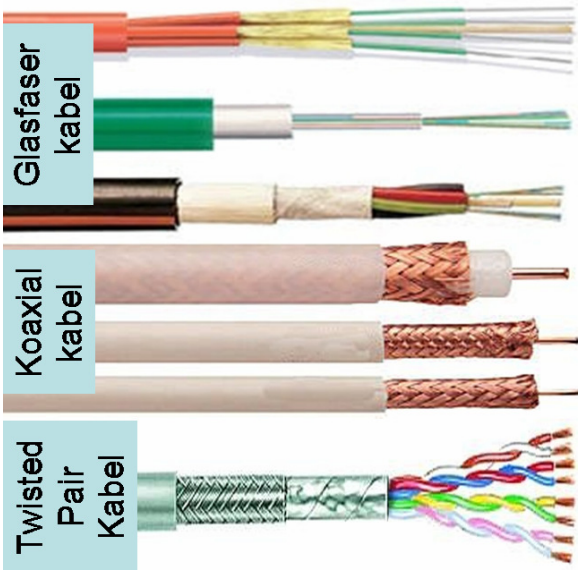


Abbildung 4-2: Beispiel der Übertragungsmedien für Ethernet [9]

In Tabelle 4-2 sind die gängigsten Ethernetstandards mit einer Übertragungsrate von 100 MBit/s dargestellt, welche auch als Fast-Ethernet bezeichnet werden. Auch hier ist das am weitesten verbreitete Übertragungsmedium das Twisted Pair Kabel. Der Standard 100Base- TX ist das meist eingesetzte Kommunikationsprotokoll der letzten Jahre und in vielen Rechnersystemen standardmäßig integriert, daher wird im nächsten Kapitel näher auf diesen Standard eingegangen.

Bezeichnung	IEEE-Norm	Jahr der Standardisierung
100Base-TX	802.3u	1995
100Base-FX	802.3u	1995
100Base-T2	802.3x&y	1997
100Base-T4	802.3x&y	1997

Tabelle 4-2: Ethernetstandards mit 100 MBit/s Übertragungsrate [10]

Tabelle 4-3 zeigt die Ethernetstandards für eine Übertragungsrate von 1GBit/s, welche in den Jahren 1998 und 1999 von der IEEE standardisiert wurden.

Bezeichnung	IEEE-Norm	Jahr der Standardisierung
1000Base-SX	802.3z	1998
1000Base-LX	802.3z	1998
1000Base-CX	802.3z	1998
1000Base-T	802.3ab	1999

Tabelle 4-3: Ethernetstandards mit 1000 MBit/s Übertragungsrate [10]

Der erste Erfolg eine Datenübertragungsrate von 10 GBit/s standardisieren zu können wurde im Jahr 2002 erreicht. Lichtwellenleiter mit kurzen, langen und extra langen (850nm, 1310nm, 1550nm) Wellenlängen werden hier als Übertragungsmedium eingesetzt. Der Durchbruch, diese hohe Übertragungsrate auch über Twisted Pair Kabel, welche weltweit im Einsatz sind, erreichen zu können, gelang erst im Jahre 2006.

Bezeichnung	IEEE-Norm	Jahr der Standardisierung
10GBase-SR	802.3ae	2002
10GBase-SW	802.3ae	2002
10GBase-LR	802.3ae	2002
10GBase-LW	802.3ae	2002
10GBase-ER	802.3ae	2002
10GBase-EW	802.3ae	2002
10GBase-LX4	802.3ae	2002
10GBase-CX4	802.3ak	2004
10GBase-T	802.3an	2006
10GBase-LRM	802.3aq	2006

Tabelle 4-4: Ethernetstandards mit 10 GBit/s Übertragungsrate [10]

Auch 40 GBit/s als Zwischenstufe zur 100 GBit/s Übertragungsrate ist in der Entwicklung vorangeschritten, aber derzeit nicht standardisiert.

4.3 Ethernetstandard 100Base-TX

Anhand des Ethernetstandards 100Base-TX, entwickelt im Jahre 1995, wird in diesem Kapitel die Kommunikation per Ethernet näher erläutert. Was beinhaltet, welches Übertragungsmedium und welche Steckverbindungen bei dieser Art der Kommunikation genutzt, sowie wie die zu sendenden Bits aufbereitet und codiert werden müssen.

4.3.1 Übertragungsmedium

Der 100Base-TX Standard verwendet als Übertragungsmedium U/UTP [62] - oder S/UTP[63]-Kabel. Die Kabel, ungeschirmt wie geschirmt, bestehen aus vier verdrillten Adernpaaren, wobei 100Base-TX nur ein Adernpaar zum Senden und ein Adernpaar zum Empfangen nutzt. Für die Industrie gibt es daher ein Twisted Pair Kabel mit nur zwei verdrillten Adernpaaren, das so genannte Industrie Twisted Pair- Kabel. In Abbildung 4-3 sind ein U/UTP-Kabel und ein geschirmtes ITP-Kabel dargestellt.

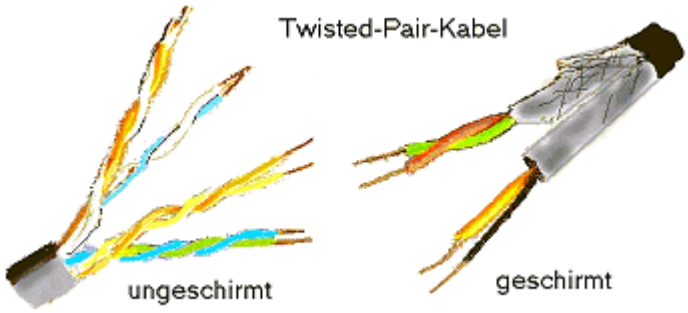


Abbildung 4-3: ungeschirmtes und geschirmtes Twisted Pair Kabel [11]

[62] Unshielded/Unshielded Twisted Pair Kabel (engl.): ungeschirmtes/ungeschirmtes paarweise verdrilltes Kabel
[63] Screened /Unshielded Twisted Pair Kabel (engl.): geschirmtes/ungeschirmtes paarweise verdrilltes Kabel

Die folgenden Abbildungen 4-4 und 4-5 zeigen den Querschnitt eines U/UTP- Kabels und eines S/UTP- Kabels. Zu sehen sind hier je vier verdrillte Adernpaare, geschirmt und ungeschirmt. Die Leitungen bestehen aus Kupfer und der Schirm aus einer metallischen Folie, einer metallisierten Kunststofffolie oder einem Drahtgeflecht, welche die elektromagnetische Verträglichkeit und Abhörsicherheit begünstigen. Außerdem können so Wechselwirkungen mit anderen Geräten vermindert werden.

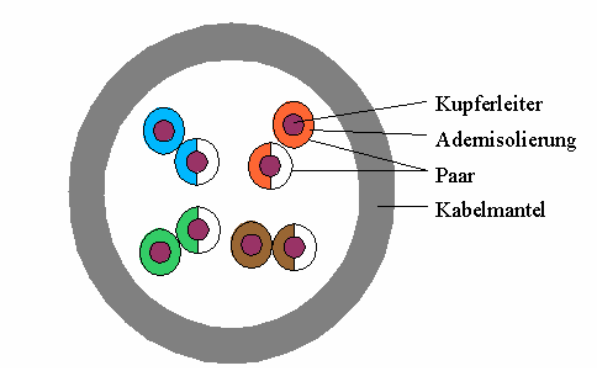


Abbildung 4-4: Aufbau eines U/UTP-Kabels [11]

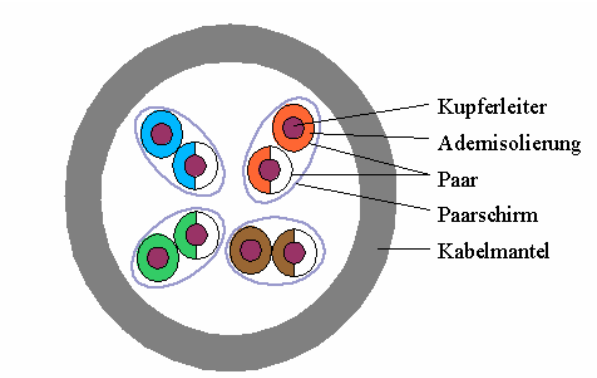


Abbildung 4-5: Aufbau eines S/UTP-Kabels [11]

Die Verbindung zwischen Ethernetsystemen wird mittels Modularbuchse und –stecker der Form 8P8C, welches umgangssprachlich fälschlicher Weise als RJ-45 bezeichnet wird, realisiert. In der Abbildung 4-6 sind 8P8C-Modularbuchse und -stecker mit entsprechender Anschlussnummerierung dargestellt.

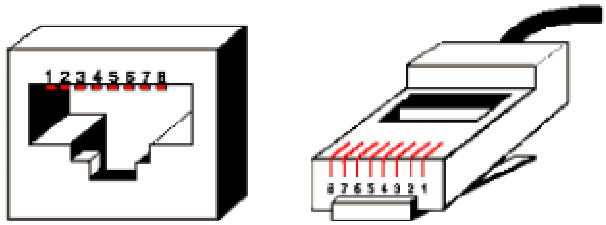


Abbildung 4-6: 8P8C- Modularbuchse und -stecker [11]

In Abbildung 4-7 ist die schematische Darstellung einer 8P8C- Modularbuchse und die Anschlussbelegung der Adernpaare eines Twisted Pair Kabels abgebildet. 100Base-TX nutzt das Adernpaar zwei als Datenausgang, wobei Anschluss eins für das positive Ausgangssignal (TX+) und zwei für das negative (TX-) genutzt werden. Der positive Eingang (RX+) belegt den Anschluss drei und das negative Eingangssignal (RX-) wird über den Anschluss sechs empfangen. Aus dieser Pin[64]-Belegung ergibt sich die zu verwendende Kabelart. Sollen zwei Ethernetschnittstellen direkt miteinander verbunden werden, muss ein gekreuztes Kabel, ein so genanntes Cross- Over- Kabel verwendet werden, so dass die Ausgangssignale der sendenden Station zu den Eingangssignalen der empfangenen Station werden. Die Leitung des Anschlusses an Pin eins (TX+) der sendenden Station muss am anderen Leitungsende den Pin drei für das positive Eingangssignal (RX+) der empfangenen Station belegen. Für die negativen Signale gilt dasselbe.

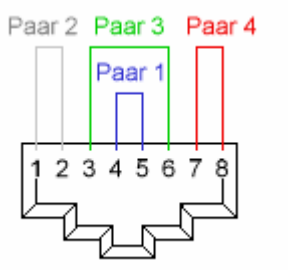


Abbildung 4-7: Anschlussbelegung eines Twisted Pair Kabels [11]

[64] Pin (engl.): Anschluss

Wie schon erwähnt, werden zur Datenübertragung, beim Senden wie auch beim Empfangen, je zwei analoge Signale gesandt. Das so genannte positive Signal, mit einem ‚+' gekennzeichnet, ist das eigentliche Signal. Das negative, mit einem ‚-' gekennzeichnete, Signal, ist das dazu komplementäre Signal. Durch dieses Verfahren können elektromagnetische Abstrahlungen reduziert werden und man erhält zwei am Empfänger ankommende Nutzsignale.

((+ Nutzsignal) + (+ Störsignal) – ((- Nutzsignal) + (Störsignal)) = 2 x Nutzsignal [5]

Da in einem Netzwerk meist nicht nur zwei Stationen integriert sein sollen, wird beim 100Base-TX Standard ein Hub zur Verbindung mehrerer Ethernetschnittstellen verwendet. Dieser ist für die Aufbereitung, das heißt Verstärkung und Entzerrung der elektrischen Signale, und Weiterleitung an alle Stationen zuständig. Für diese Vernetzung sind U/UTP- Kabel nötig, die an beiden Enden dieselbe Anschlussbelegung haben. U/UTP-Kabel unterscheiden sich nicht nur in der Steckeranschlussbelegung, sondern auch im Wellenwiderstand, der maximalen Betriebsfrequenz und Dämpfung. Aus diesem Grunde wurden die verschiedenen Kabel durch den amerikanischen Unternehmerverband EIA[65] in sieben Kategorien eingeteilt. Derzeit sind nur noch die Kabel ab Kategorie fünf, welche der Klasse D der europäischen Normierung entspricht, für die Vernetzung von Rechnersystemen von Interesse. Kabel der Kategorie fünf (Cat 5) haben eine maximale Betriebsfrequenz von 100MHz und sind abwärtskompatibel. Das heißt, auch Netzwerke mit einer Übertragungsrate von 10 MBit/s können mit Cat 5- Kabeln verbunden werden. Die Dämpfung bei der maximalen Betriebsfrequenz (100 MHz) beträgt 22dB [5] und der Segmentabstand beträgt wie beim 10Base-T Standard 100 Meter. Der Wellenwiderstand beträgt wie auch bei allen anderen Kategorien 100 Ohm±15%.

4.3.2 Kodierungsverfahren

Nach Festlegung des Übertragungsmediums muss auch eine Standardisierung der Datenübertragungsform vollzogen werden. Da Nachrichten in der Digitaltechnik nur aus logischen Einsen und Nullen bestehen, wurde hier festgelegt, dass eine logische ‚1' dem High- Spannungspegel von 5 Volt entspricht und eine logische ‚0' einem Low- Spannungspegel von 0 Volt. Das bedeutet, besteht eine Nachricht aus ‚11110000' wird über vier Takte ein High-Signal und über vier Takte ein Low-Signal gesandt. Dieses Verfahren lässt allerdings keine Taktrückgewinnung zu, was für die Synchronisation von Sender und Empfänger aber nötig ist. Um keine weitere Leitung für die Taktsynchronisation legen zu müssen, werden die Datenbits vor dem Senden mit dem Kodierungsverfahren **4Bit-to-5Bit**, kurz **4B/5B**, kodiert. Das bedeutet, es wird ein Nibble[66] , also vier Datenbits, in fünf Signalbits umgewandelt. In Tabelle 4-5 ist die Kodierung der Datennibble in den 5-Bit-Code und dessen Bedeutung und Namen aufgelistet. Mit einem Nibble sind 16 (2^4) mögliche Bitkombinationen darstellbar, mit fünf Bits sind es 32 (2^5)

[65] Electronic Industries Alliance (engl.): Elektronikindustrieverband
[66] Nibble (engl.): halbes Byte

Kombinationen. Daher könnten 16 Kombinationen, die nicht für die Datenbitkodierung benötigt werden, als Statusanzeigen genutzt werden. Insgesamt werden sechs Zustände durch Bitkombinationen dargestellt. Der Idle[67]- Zustand wird beim 100Base-TX Standard während einer Sendepause, also zwischen dem Senden mehrerer Datenpaket, gesandt und zur Synchronisation zwischen den Stationen genutzt. Außerdem gibt es zur Signalisierung des Starts und des Endes einer Datenpaketsendung je zwei Bitkombinationen. Der letzte noch genutzte Bit-Code wird als Transmit Error bezeichnet und zeigt an, dass eine Signalstörung aufgetreten ist. Die übrigen Bitkombinationen sind so genannte Violation Combinations[68]. Tritt ein Violation Combination auf, ist dies ein Garant für eine fehlerhafte Sendung. Durch die 4B/5B Kodierung werden zwar redundante Daten erzeugt, dafür können aber zusätzlich Statusinformationen gesandt werden. Der eigentliche Zweck dieser Kodierung sind die Möglichkeit einer Taktrückgewinnung und die Vermeidung eines Baselinewanderns. Das Wandern einer Baseline[69] bedeutet eine Verschiebung des Arbeitspunktes, auf welchen der Empfänger zur Erkennung der gesandten Signalpegel eingestellt ist. Beim Senden einer Reihe gleicher Pegel, entsteht ein Gleichspannungsanteil im Datenstrom, dadurch kann es zur Baselinewanderung kommen, was zu einer Nichterkennung unterschiedlicher Pegel führen würde. Um das zu verhindern ist das Kodieren mittels 4B/5B notwendig, denn durch diese Kodierung sind ständige Pegelwechsel garantiert. Auch eine Taktrückgewinnung kann durch diese Kodierung vollzogen werden, da bei jeder gesandten Bitfolge ein Pegelwechsel stattfindet. Eine Taktrückgewinnung ist für eine Synchronisation zwischen Sender und Empfänger notwendig, damit beide Stationen im selben Takt arbeiten und so die Pegelerkennung auf der Empfängerseite funktionieren kann.

Codetyp	Datenbits (4 Bits)	Signalbits (5 Bits)	Name
Daten 0	0000	11110	0
Daten 1	0001	01001	1
Daten 2	0010	10100	2
Daten 3	0011	10101	3
Daten 4	0100	01010	4
Daten 5	0101	01011	5
Daten 6	0110	01110	6
Daten 7	0111	01111	7
Daten 8	1000	10010	8
Daten 9	1001	10011	9
Daten A	1010	10110	A
Daten B	1011	10111	B
Daten C	1100	11010	C

[67] Idle (engl.): Leerlauf
[68] Violation Combination (engl.): verbotene Kombination
[69] Baseline (engl.): Grundlinie

Daten D	1101	11011	D
Daten E	1110	11100	E
Daten F	1111	11101	F
Idle	undefiniert	11111	I
Start of Stream (Teil 1)	0101	11000	J
Start of Stream (Teil 2)	0101	10001	K
End of Stream (Teil 1)	undefiniert	01101	T
End of Stream (Teil)2	undefiniert	00111	R
Transmit Error	undefiniert	00100	H

Tabelle 4-5: 4B/5B- Kodierung [6]

Da durch die 4B/5B Kodierung pro Nibble ein Bit mehr übertragen werden muss, beträgt bei einer Datenübertragungsrate von 100 MBit/s die Signalübertragungsrate 125 MBit/s. Durch MLT-3[70] kann die Übertragungsfrequenz auf 31,25 MBit/s gesenkt werden. Hierbei handelt es sich um eine Signalübertragung mit den drei Zuständen ‚+1', ‚0' und ‚-1'. Bei jeder logischen ‚1' die übertragen werden soll, wird der Signalpegel um eine Stufe je nach Anfangspegel herauf oder herabgesetzt.

Bei einer Übertragungsrate von 10 MBit/s wurde eine einfachere Kodierung des Nutzsignals zur Taktrückgewinnung, die Manchesterkodierung, genutzt. Hierbei wird die Taktinformation in der Mitte einer Bitperiode übertragen. Das bedeutet, in der ersten Hälfte einer Bitperiode wird der komplementäre Wert und in der zweiten Hälfte der eigentliche Nutzwert auf die Leitung gegeben. Dadurch ist sichergestellt, dass ausreichend Taktinformationen innerhalb des Bitstroms übertragen werden. In Abbildung 4-8 sind die Machesterkodierung und die MLT-3-Kodierung in Beziehung zu den Datenbits dargestellt.

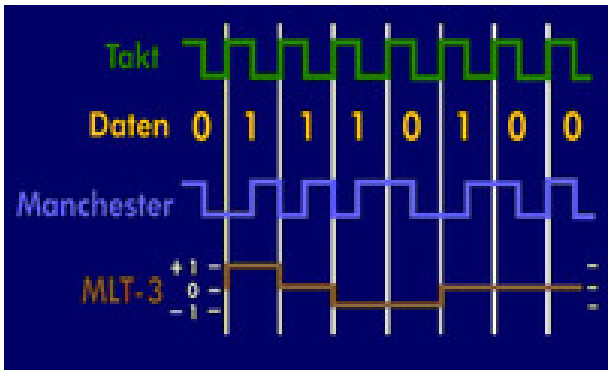


Abbildung 4-8: Darstellung der Manchester und MLT-3 Codierung [12]

[70] Multilevel Transmission 3 (engl.): Übertragung auf mehreren Ebenen

Eine weitere Verschlüsselungstechnik, welche bei der Übertragung mittels 100BaseTX genutzt wird, ist das Scrambling[71]. Der genutzte Algorithmus ist Sender und Empfänger bekannt und wird durch eine logische Oder- Verknüpfung[72] zwischen Datenstrom und einem 11 Bit langem Code realisiert. Durch diese Datenverwürfelung ist sichergestellt, dass sich die in einem lang anhaltenden Idle- Zustand ständig wiederholenden Symbole nicht auf die Signalübertragungsqualität auswirken. Zusätzlich zur 4B/5B- und MLT-3-Kodierung wird die NRZI- Kodierung vorgenommen. Bei diesem Verfahren wird nur bei einem High-Pegel des Datenbits ein Pegelwechsel vorgenommen. Das bedeutet, werden viele High- Pegel hintereinander übertragen, wird bei jedem Bit ein Signalpegelwechsel vorgenommen. Dieses Verfahren wurde auch schon bei dem Ethernet- Standard 10BaseT genutzt und konnte daher ohne Probleme in den 100BaseTX- Standard als zusätzliche Vorsichtsmaßnahme integriert werden.

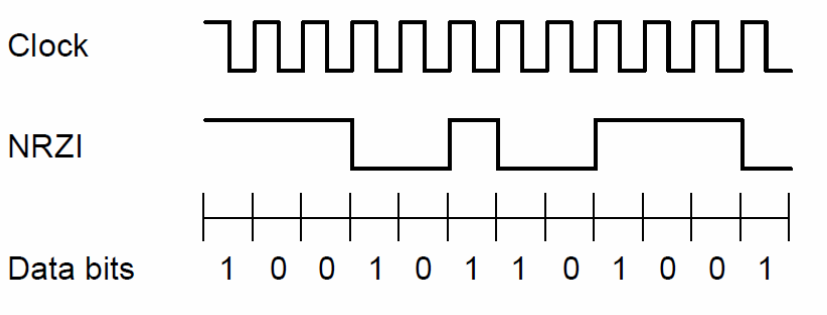


Abbildung 4-9: NRZI- Kodierung [13]

Die Funktion Auto- Negotiation [73] findet beim 100Base-TX- Standard, sowie beim 10Base-T- Standard Verwendung und dient dem Auslesen der verfügbaren Übertragungsmodi der Gegenstation. Das bedeutet, sind beide angeschlossenen Ethernet- Controller in der Lage Daten mit einer Übertragungsrate von 100MBit/s zu verarbeiten, wird dieser Modus zur Übertragung gewählt. Außerdem können die Übertragungsmodi Halb- oder Vollduplex gewählt werden. Dieser Austausch der Modi zwischen den Stationen findet beim 10Base-T- Standard über die so genannten NLPs[74] und beim 100Base-T-Standard über die Idle- Signale statt, welche in Übertragungsruhephasen auf das Übertragungsmedium gegeben werden, um eine mögliche Verbindungsunterbrechung erkennen zu können. Wenn beispielsweise eine Netzwerkkarte mit beiden Übertragungsgeschwindigkeitsmodi ein NLP empfängt, wird automatisch eine Übertragungsgeschwindigkeit von 10 MBit/s eingestellt, so dass eine Kommunikation möglich ist.

[71] Scrambling (engl.): Verwürfelung
[72] liegt an einem der Dateneingänge ein High an, wird auch der Ausgang auf High gesetzt
[73] Auto- Negotiation (engl.): automatische Übertragung
[74] Normal Link Puls (engl.): normale Verbindungspulse

4.4 Kollisionsvermeidung unter Ethernet

Bei dem 100Base-TX Standard werden die Stationen in einer Sterntopologie über einen Hub vernetzt. Dieser Hub ist nur für die Aufbereitung und Weiterleitung der Signale an alle Stationen zuständig, daher entspricht die Sterntopologie logisch einer Bustopologie. Da es bei gleichzeitigem Senden mehrerer Stationen auf dem Übertragungsmedium zu Kollisionen der Signale und dadurch zum Verlust der Daten kommt, wird zur Kollisionsvermeidung das **CSMA/CD- Verfahren**, eingesetzt. Frei übersetzt bedeutet dieser Name, dass mehreren Stationen gleichberechtigter Zugriff auf das Übertragungsmedium gewährt wird (Multiple Access), diese aber vor dem Zugriff das Medium auf anderweitige Benutzung prüfen (Carrier Sense). Außerdem wird bei Kollisionserkennung ein Signal auf das Medium gegeben, durch das alle Stationen das Senden einstellen (Collision Detection). In Abbildung 4-10 ist der Vorgang vom gleichzeitigen Senden bis zum Stoppen des Sendevorgangs schematisch dargestellt [6].

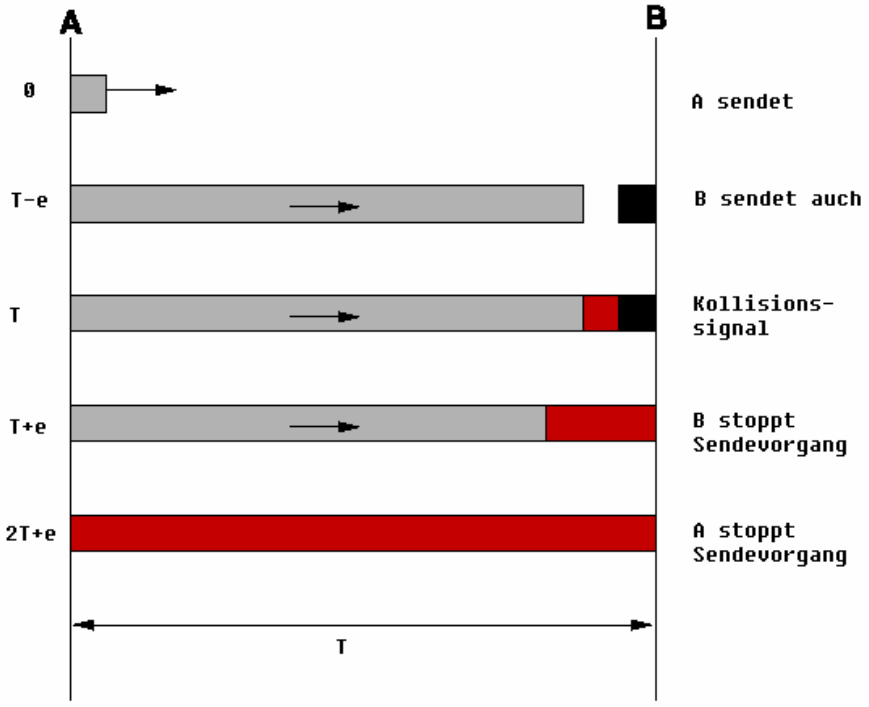


Abbildung 4-10: Schematische Darstellung einer Kollisionserkennung [11]

Das Prüfen auf schon bestehende Signalübertragungen auf dem Übertragungsmedium minimiert das Risiko der Kollision, kann sie aber dennoch nicht verhindern. Denn mehrere Stationen können das Medium zur selben Zeit für frei befinden und dann gleichzeitig senden. Erkannt wird eine Kollision, wenn eine sendende Station während der Datenübertragung auch Daten empfängt. Da nur eine Station zur selben Zeit senden darf, kann kein gleichzeitiger Empfang möglich sein. Tritt dieser Fall ein, überträgt die Station ein Kollisionssignal, das so genanntes Jam[75] Signal. Dieses Signal besteht aus 32

[75] Jam (engl.): Blockierung

aufeinander folgenden alternierenden Bits, beginnend mit einem High Pegel. Diese Kollisionserkennung kann nur angewandt werden wenn der Netzbetrieb im Halbduplexmodus läuft, das heißt gleichzeitiges Senden und Empfangen von Daten ist nicht möglich. Im Vollduplexmodus, also gleichzeitiges Senden und Empfangen ist gestattet, wird kein Kollisionserkennungsverfahren eingesetzt.

4.5 Ethernetheader

In einem Netzwerk reicht es nicht aus die Datenbits zu kodieren und in einer bestimmten Form auf das Übertragungsmedium zu geben. Die Daten müssen, wie beim realen Senden von Dingen, in Pakte von minimaler und maximaler Größe eingeteilt werden. Die Adressen des Empfängers und Senders sowie eine Garantie der Unversehrtheit des Paktes sind nötig. All diese Zusatzinformationen werden zusammen mit den Daten in einem so genannten Header[76] verschickt. In Abbildung 4-11 ist der Ethernetheader 802.3 dargestellt. Dieser beinhaltet eine Startsequenz, den Ethernetframe und den Inter Frame Gap[77], welche folgend näher erläutert werden. Der Ethernetframe besteht aus Ziel- und Quelladresse, dem Typfeld beziehungsweise Längenfeld, der zu übertragenden Nachricht und einem Prüffeld.

<table>
<tr><td>Bitfolge
1010101010..</td><td>Bitfolge
10101011</td><td colspan="5">Ethernet - Frame min. 64 Byte max. 1518 Byte</td><td>Inter
Frame
Gap
9,6µs</td></tr>
<tr><td>Preamble</td><td>SFD</td><td>6 Byte
Dest.-
Addr</td><td>6 Byte
Source-
Addr</td><td>2 Byte
Type/Length</td><td>min 46 Bytes max 1500
Bytes
Daten</td><td>4
Byte
FCS</td></tr>
</table>

Abbildung 4-11: Ethernet 802.3 Header [11]

- Die **Preamble** besteht aus einer alternierenden Folge von High- und Low-Pegel beginnend mit einem High-Pegel. Diese umfasst sieben Bytes und dient der Synchronisation, also dem Taktabgleich, zwischen Sender und Empfänger.

- Das **SFD**[78] zeigt den Beginn des Ethernetframes an. Er besteht aus der ein Byte langen Kombination ‚10101011'.

- Die **Destination- Address** ist die Adresse der empfangenen Station. Jedes System mit einer Netzwerkanbindung hat seine eindeutige Adresse, die so genannte **MAC-Adresse**. Sie besteht immer aus einer Kombination von sechs Byte und wird zur besseren Lesbarkeit byteweise in hexadezimaler Schreibweise angegeben.

[76] Header (engl.): Dateikopf
[77] Inter Frame Gap (engl.): Abstand zwischen Rahmen
[78] Start Frame Delimiter (engl.): Rahmenstartbegrenzer

- Die **Source- Address** ist die Quelladresse, welche die MAC-Adresse des Senders beschreibt.

- Das Headerfeld **Type/Length** besteht aus zwei Byte und gibt den Typ des übergeordneten Protokolls oder die Anzahl der zu übertragenden Datenbytes an. In Tabelle 4-6 ist ein Ausschnitt der möglichen Typfeldangaben dargestellt. In der ersten Zeile ist die mögliche Längenangabe des Datenfeldes eingetragen. Die Länge muss mindestens 46 und darf höchstens 1500 Byte betragen.

Typfeld		Protokoll
Hex	Dezimal	
0x002E-0x05DC	46-1500	Ethernet- Längenangabe
0X0600	1536	XEROX IDP
0x0800	2048	IP Internet Protocol (IPv4)
0x0806	2054	Address Resolution Protocol (ARP)
0x8035	32821	Reverse Address Resolution Protocol (RARP)
0x809B	32923	AppleTalk (EtherTalk)
0x8137	33079	Novell IPX (alt)
0x8138	33080	Novell
0x86DD	34525	Internet Protocol, Version 6 (IPv6)

Tabelle 4-6: Ausschnitt aus Ethernettypangabe [6]

- Das Datenfeld, in welchem die eigentliche Nachricht steht, hat eine Mindestgröße von 46 Byte und eine maximale Größe von 1500 Byte. Da es unterschiedliche Ethernetframeformate gibt, kann die Größe des Datenfeldes variieren. Festgelegt ist aber die Länge des Ethernetframes, diese muss zwischen 64 und 1518 Bytes liegen. Ist die zu sendende Nachricht kleiner als 46 Byte, wird das Datenfeld mit ‚0'en aufgefüllt.

- Um eine Prüfung der korrekten Datenübermittlung vornehmen zu können, wird dem Datenfeld eine Prüfsumme, die **FCS**[79] angehängt. Diese besteht aus vier Byte und wird durch einen Algorithmus, welcher auf Polynomdivision beruht, generiert. Im folgenden Kapitel 4.6 wird der Algorithmus näher erläutert.

- Das **Inter Frame Gap**, ist als „Erholungszeit" für den Empfänger gedacht. Der Inter Frame Gap beträgt 96 Bitzeiten, was bei Übertragung mit 10 MBit/s einer zeitlichen Dauer von 9,6 µs und äquivalent 0,96 µs bei einer Übertragungsrate von 100 Mbit/s. Diese Sendepause wird zwischen der Sendung zweier Frames durchgeführt [5].

[79] Frame Check Sequence (engl.): Rahmenprüfsequenz

4.6 Zyklische Redundanzprüfung

Bei der Übertragung von Daten über Netzwerke werden seriell oder parallel einzelne Bits über meist mehrere Meter Kabel oder Funk übertragen. Da Störungen einer Funkverbindung oder eines Kabels auftreten können, ist es möglich dass die übertragenen Bits verfälscht werden. Somit entspräche die am Empfänger angekommene Nachricht nicht mehr dem Original. Um solche inkorrekten Daten filtern und sogar korrigieren zu können, wird bei Netzwerkübertragungen sowie beim Kopieren von CDs, DVDs und Festplatten das Verfahren der zyklischen Redundanzprüfung angewandt. Dieses Verfahren beruht auf der Polynomdivision und kann mit geringem Aufwand in Software sowie Hardware realisiert werden. Durch die Berechnung einer Prüfsumme aus den zu übertragenden Nutzdaten sind die Daten der Prüfsumme bereits in der Nachricht enthalten und somit zwar redundant, jedoch zur Feststellung ob die Daten korrekt übermittelt wurden von hohem Wert. Jedes berechnete Codewort ergibt bei einer zyklischen Verschiebung seiner Stellen wieder ein Codewort, damit ist die Codierung zyklisch. Folgend wird dieses Verfahren mit dem gängigeren Namen CRC[80]-Verfahren bezeichnet.

Da die Berechnung der Prüfsumme, auch Codewort genannt, auf der Polynomdivision beruht, ist es nötig alle Daten als dyadische Polynome darzustellen. Das bedeutet jedes Bit wird je nach Stellenwert einem X zugewiesen.

Die aus einem Byte bestehende zu übertragende Nachricht

$$N = 01010010$$

wird mit

$$N(X) = \sum_{k} b * X^{k} \quad \text{mit} \quad \begin{matrix} k \in \subseteq 0, \infty \supseteq \\ b \in \subseteq 0,1 \supseteq \end{matrix} \tag{1}$$

zu

$$N(X) = X^{6} + X^{4} + X \; . \tag{2}$$

Der Schlüssel, hier als Generatorpolynom G(X) bezeichnet, ist der selbst festgelegte Divisor der Polynomdivision und muss dem Sender sowie dem Empfänger bekannt sein. Nach dem IEEE Standard 802.3 wird bei der Übertragung mittels Ethernet der so genannte CRC-32-Schlüssel verwendet.

[80] cyclic redundancy check (engl.): zyklische Redundanzprüfung

Folgend das CRC-32-Generatorpolynom:

$$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1 \quad (3)$$

Nach der Polynomdivision des Nachrichtpolynoms durch das Generatorpolynom bleibt ein nicht dividierbarer Rest R(X), welcher die Prüfsumme darstellt. Dieser wird der zu übertragenden Nachricht angehängt und dem Empfänger mit übermittelt. Anhand des, dem Empfänger bekannten, Generatorpolynoms, der empfangenen Nachricht und des Restpolynoms kann nach Durchführung einer erneuten Polynomdivision ermittelt werden, ob die Daten korrekt übertragen wurden. Das bedeutet, die Nachricht mit angehängtem Rest wird durch das Generatorpolynom geteilt, bleibt kein Rest wurden die Nutzdaten korrekt übermittelt. Andernfalls liegt ein Fehler vor und das Datenpaket wird verworfen und kann neu angefordert werden [5].

Um eine Polynomdivision durchführen zu können muss das Nachrichtpolynom mit der höchsten Potenz des Generatorpolynoms multipliziert werden:

$$N(X)' = N(X) * X^{32} = (X^6 + X^4 + X) * X^{32} = X^{38} + X^{36} + X^{33} \quad (4)$$

Durchführung der Polynomdivision:

$$\frac{N(X)'}{G(X)} = D(X) + R(X) \quad (5)$$

Durch Einsetzen von (3) und (4) in (5)

$$\frac{X^{38} + X^{36} + X^{33}}{X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1} =$$

$$X^6 + X^4 + X + 1 \quad \text{Rest} \quad X^{30} + X^{29} + X^{28} + X^{24} + X^{20} + X^{18} + X^{16} + X^{15} + X^{12} + X^6 + X^5 + X^3 + 1$$

ergibt sich:

$$D(X) = X^6 + X^4 + X + 1 \quad (6)$$

$$R(X) = X^{30} + X^{29} + X^{28} + X^{24} + X^{20} + X^{18} + X^{16} + X^{15} + X^{12} + X^6 + X^5 + X^3 + 1 \quad (7)$$

Die zu übermittelnde Nachricht N(X)'' soll nun aus dem erweiterten Nachrichtpolynom N(X)' und dem angehängtem Restpolynom R(X) bestehen.

Durch einsetzen von (4) und (7) ergibt sich

$$N(X)'' = N(X)' + R(X) = \tag{8}$$

$$(X^{38} + X^{36} + X^{33}) + (X^{30} + X^{29} + X^{28} + X^{24} + X^{20} + X^{18} + X^{16} + X^{15} + X^{12} + X^{6} + X^{5} + X^{3} + 1)$$

Um im Empfänger überprüfen zu können wird das Polynom N(X)'' durch das Generatorpolynom G(X) dividiert.

Durch Einsetzen von (8) und (3) ist das Resultat (6):

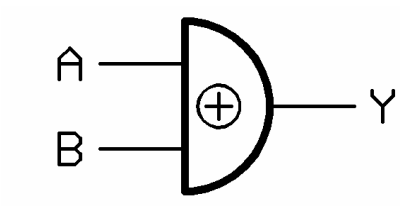
$$N(X)'' / G(X) = \tag{9}$$

$$\frac{X^{38} + X^{36} + X^{33} + X^{30} + X^{29} + X^{28} + X^{24} + X^{20} + X^{18} + X^{16} + X^{15} + X^{12} + X^{6} + X^{5} + X^{3} + 1}{X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1}$$

$$= X^{6} + X^{4} + X + 1$$

Anhand dieses Beispiels ist zu sehen, dass sich die übertragene Nachricht ohne ein Restpolynom dividieren lässt und die Division in (9) wieder zu dem ganzzahligen Ergebnis wie in (6) führt. Die Nachricht wurde somit korrekt übermittelt.

Das Verfahren der Polynomdivision wird in der Digitaltechnik mittels Schieberegister und der so genannten Exklusiv- Oder- Verknüpfung, kurz XOR, realisiert. XOR bedeutet, dass bei einem High- Pegel an nur einem der Eingänge, auch die am Ausgang anliegende Leitung einen High- Pegel besitzt. Das heißt, liegt entweder an Eingang A oder an Eingang B ein High- Pegel an, so wird auch der Ausgang auf einen High- Pegel gesetzt. Sind beide Eingänge mit einem High- Pegel beschaltet wird der Ausgang auf einen Low- Pegel gesetzt. In Abbildung 4-12 ist die schematische Darstellung einer XOR-Verknüpfung und der zugehörigen Wahrheitstabelle abgebildet. Die Wahrheitstabelle gibt an, was bei den Eingangskombinationen am Ausgang anliegt. Die ‚0' entspricht einem Low- Pegel und die '1' einem High- Pegel.



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Abbildung 4-12: Schematische Darstellung einer XOR- Verknüpfung mit Wahrheitstabelle

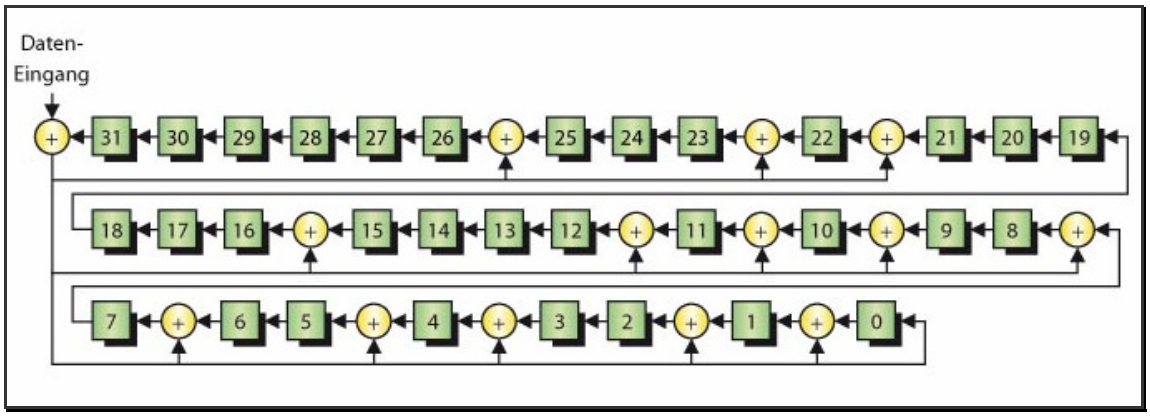


Abbildung 4-13: Schematische Darstellung des Schieberegisters zur CRC- 32 Berechnung [14]

5 Grundlagen FPGA

In diesem Kapitel wird auf den Aufbau von FPGAs eingegangen und kurz dargestellt welche Hardwarebeschreibungssprachen im Allgemeinen zur Verfügung stehen.

5.1 Aufbau eines FPGAs

Field Programmable Gate Arrays sind widerkonfigurierbare Siliziumchips, welche aus logischen Komponenten bestehen, die durch steuerbare Schalter verbunden und in einer Matrix angeordnet sind. Diese logischen Komponenten erfüllen die Funktion grundlegender Logik, wie Konjunktion, Disjunktion, Peirce-Funktion und Negation. Durch das Verknüpfen dieser Komponenten, auch Logikblöcke genannt, kann eine digitale Schaltung mit komplexer Logik erstellt werden. Die Vorteile von FPGAs sind änderbare Schaltkreise und schnelle Schaltzeiten. Das bedeutet die entwickelten Schaltkreise sind nicht wie auf einer Platine fest verdrahtet, sondern durch steuerbare Schalter stets neu konfigurierbar und daher leicht veränderbar.

Um den Aufbau eines vielfach verwendeten FPGAs näher erläutern zu können, ist in Abbildung 5-1 die Architektur des Bausteins XC3S500E der Firma Xilinx schematisch dargestellt.

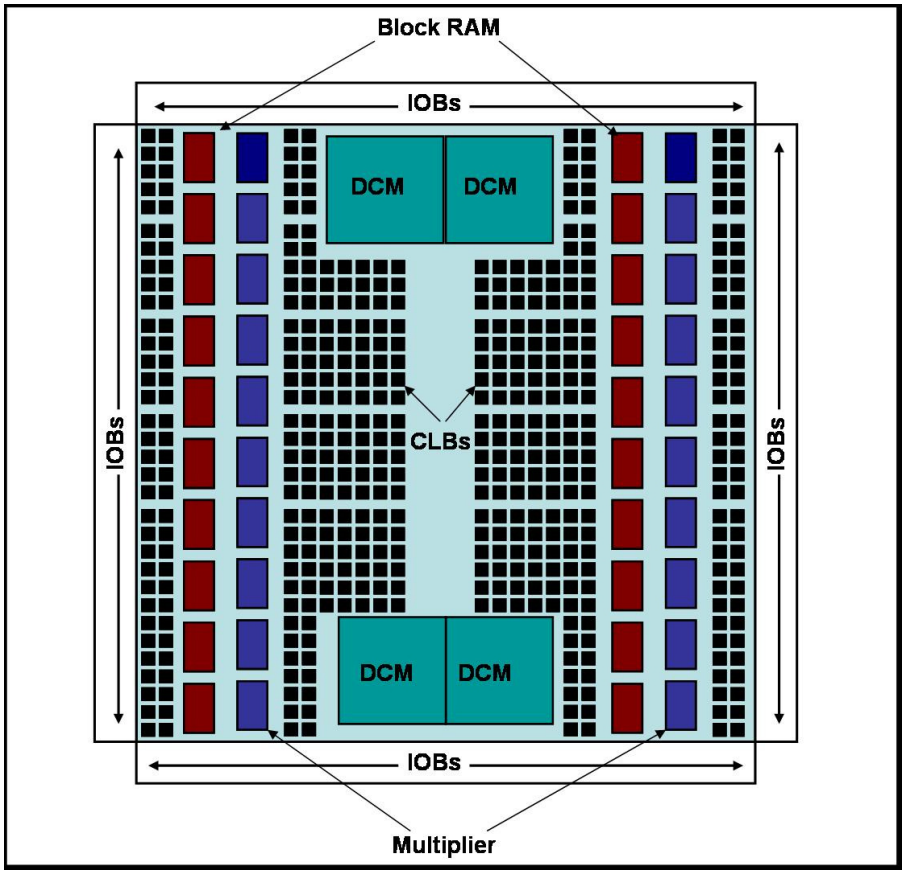


Abbildung 5-1: Schematische Darstellung eines FPGAs

Die CLBs[81] bestehen aus je vier Slices[82]. In Abbildung 5-2 ist die Anordnung der Slices innerhalb eines CLBs schematisch dargestellt.

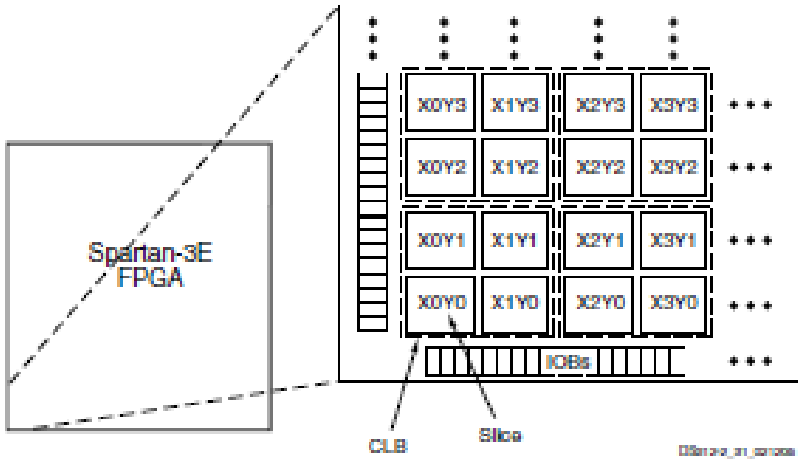


Abbildung 5-2: Schematische Darstellung eines Configurable Logic Block [15]

Jedes Slice besteht aus **LUTs**[83], welche als Speicher für vorausberechnete Ausgangssignale dienen. Außerdem enthält jedes Slice zwei Speicherelemente, welche als Flip-Flops[84] oder Latches[85] genutzt werden können. In Abbildung 5-3 ist ein vereinfacht dargestelltes Slice mit seinen integrierten LUTs, welche je vier Eingänge besitzen, den Carry- und Steuerungslogikeinheiten und zwei Speicherelementen, welche hier als FlipFlops dargestellt sind, abgebildet.

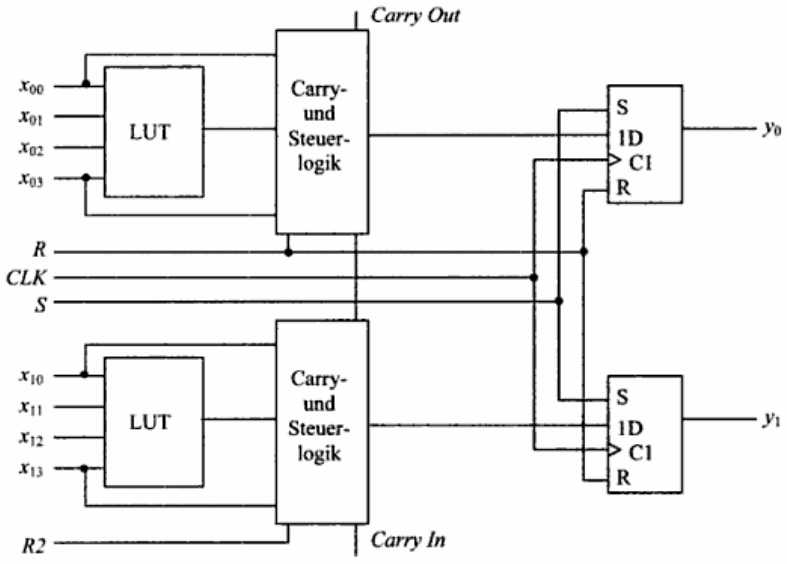


Abbildung 5-3: Vereinfacht dargestelltes Slice [16]

[81] Configurable Logic Block (engl.) konfigurierbare Logikblöcke
[82] Slice (engl.): Teil
[83] Look Up Table (engl.): Nachschlagewerk
[84] taktgesteuerte 1- Bit- Datenspeicher
[85] zustandsgesteuerte 1- Bit- Datenspeicher

Der hier beschriebene FPGA umfasst 1164 CLBs und entsprechend 9312 takt- oder zustandsgesteuerte Speicherelemente, sowie 9312 LUTs. In Abbildung 5-1 sind außerdem die **Multiplizierer** dargestellt. Dieser FPGA- Baustein enthält zwei Blöcke mit je zehn Multiplizierern, welche jeweils zwei binäre Zahlen mit einer Größe von maximal 18 Bits multiplizieren können. Direkt neben den Multiplizierern befinden sich die **Block RAM**s[86]. Der FPGA umfasst zwei Reihen mit je zehn Block RAMs. Jeder Block RAM umfasst 18.000 Bit statischen Speicher. Statischer Speicher bedeutet, dass beim Abschalten der Versorgungsspannung die gespeicherten Daten verloren gehen. Wie in Abbildung 5-1 dargestellt, ist der Speicher in zwei Blöcke unterteilt. Einer befindet sich in der linken und der andere in der rechten Hälfte des FPGAs. Beide Blöcke besitzen je einen Zugangsport (Port A und Port B), durch den die Speicherblöcke unabhängig voneinander ausgelesen oder beschrieben werden können. Der Speicher wird taktabhängig bearbeitet, daher besitzt jeder Block seinen eigenen Taktgenerator. Die **DCM**[87] **s**ind für die Taktsteuerung zuständig. Ein DCM besteht aus vier Einheiten:

- Die **DLL**[88] generieren Taktausgangssignale mit einer Null-Verzögerung zum Takteingangssignal, welches durch einen externen Quarz erzeugt wird. Außerdem können durch den Verzögerungsregelkreis unterschiedliche Phasenverzögerungen im Einklang mit der Phasen- Verschiebungs- Einheit, die folgend erläutert wird, durch Verzögerungselemente erzeugt werden. Die Phasenverschiebungen können 90°, 180° und 270° betragen. Zudem ist noch eine Verdopplung der Frequenz bei gleicher Phase oder 180° Phasenverschiebung möglich.

- Das **Phase Shift**[89] kontrolliert die Phasengleichheit des Takteingangssignals und des Taktausgangssignals. Durch eine vom Benutzer festgelegte Variable kann mittels des Phase Shiftings die Phase des Takteingangssignals von 0° bis +270° verschoben werden.

- Durch den **Digital Frequency Synthesizer**[90] ist es mittels zweier vom Benutzer festzulegenden Variablen möglich, eine Taktfrequenz von 5 bis 333 MHz aus einem gegebenen Taktsignal generieren zu lassen. Dieser Frequenzgenerator hat zwei mit gewünschter Frequenz generierte Taktausgangssignale, eins mit der gleichen Phase wie das Takteingangssignal und eines um 180° phasenverschoben.

[86] Random Access Memory (engl.): Speicher mit wahlfreiem Zugriff
[87] Digital Clock Manager (engl.): Digitaltaktsteuerer
[88] Delay- Locked Loop (engl.): Verzögerungsregelkreis
[89] Phase Shift (engl.): Phasen-Verschiebungs-Einheit
[90] Digital Frequency Synthesizer (engl.): digitaler Frequenzgenerator

- Der aktuelle Status des Digital Clock Managers wird durch die insgesamt acht Ausgangssignale des **Status Logic** dargestellt. Ein acht Bit breites Bussignal zeigt die Status der Aktivitäten der Phasen- Verschiebungs- Einheit und des Verzögerungsregelkreises an. Ein Ausgangssignal, welches als LOCKED bezeichnet wird, zeigt an, ob die Phase des Taktausgangssignals des DCMs und des Takteingangssignals übereinstimmen.

In Abbildung 5-1 sind außerdem an jeder Seite des Bausteins so genannte IOBs[91] dargestellt und bezeichnen die Anschlussmöglichkeiten des FPGAs, welche je als Signaleingang, Signalausgang oder Tri- State- Ausgang[92] genutzt werden können. Als Tri- State-Ausgang werden Ausgänge mit drei Zuständen bezeichnet, welche einen High- Pegel einen Low- Pegel ausgeben und auch hochohmig beschaltet werden können. Eine hochohmige Beschaltung kann als nicht verbunden betrachtet werden, da der Ausgang keinen Spannungswert annimmt. Der hier vorgestellt Baustein besitzt 232 dem Benutzer zur Verfügung stehenden Anschlüsse [15].

[91] Input- Output- Block (engl.): Eingangs- Ausgangs- Block
[92] Ausgang mit drei Zuständen

5.2 Hardwarebeschreibungssprache

FPGAs werden mittels so genannter HDLs[93] konfiguriert, von welchen es zwei weit verbreitete gibt.

5.2.1 VHDL

In Europa ist VHDL[94] die meistverbreitete Hardwarebeschreibungssprache, welche eine Erweiterung der Sprache VHSIC[95] des amerikanischen Verteidigungsministeriums, auf wessen auch der Name beruht, ist. 1987 wurde VHDL als IEEE-1076-Standard verabschiedet, welcher 1993 durch den IEEE-1164-Standard abgelöst wurde. Durch die Standardisierung dieser Beschreibungssprache sind mittels VHDL entworfene Schaltungen portierbar und bausteinunabhängig. Auf einem von der Firma Xilinx entwickelten FPGA kann ein durch VHDL entwickelter Schaltkreis genauso umgesetzt werden wie auf einem Baustein der Firma Altera. Durch einen VHDL- Quelltext können, über die entsprechende Software, die in einem FPGA integrierten Logikblöcke miteinander verknüpft werden. Die digitale Schaltung wird also mittels eines Routenplans in Hardware umgesetzt. VHDL ist eine sehr wortreiche Sprache und in Anlehnung an die Programmiersprache ADA entwickelt worden [17].

5.2.2 Verilog

Die zweite eher in den USA und Asien verbreitete Hardwarebeschreibungssprache ist die Verilog[96] HDL, welche Mitte der 80er Jahre ursprünglich von der Firma Gateway Design Automation als Simulationssprache entwickelt wurde. Verilog-95 wurde von der IEEE im Jahre 1995 als Standard *IEEE-1364-1995* verabschiedet, welcher dann im Jahr 2001 aufgrund einiger Mängel von dem Standard *IEEE-1364-2001* abgelöst wurde. Verilog wurde in Anlehnung an die Programmiersprache C entwickelt [17].

[93] Hardware Description Language (engl.): Hardwarebeschreibungssprache
[94] Very High Speed Integrated Circuit Hardware Description Language (engl.): Hardwarebeschreibungssprache für sehr schnelle integrierte Schaltkreise
[95] Very High Speed Integrated Circuit (engl.): sehr schnelle integrierte Schaltkreise
[96] Verify logic (engl.): Logik verifizieren

6 Problemdarstellung

Die Umsetzung eines Ethernetcontrollers auf einem FPGA beinhaltet folgende Probleme:

- Auseinandersetzung mit den Spezifikationen des genutzten FPGAs und der Entwicklungsumgebung, was unter anderem die Anzahl der nutzbaren FlipFlops und Slices betrifft
- Programmiermöglichkeiten und Spezifikationen des vorhandenen PHY- Bausteins eruieren
- Zusammenspiel zwischen FPGA und PHY- Baustein extrahieren, das bedeutet, welche Programmiermöglichkeiten sind gegeben
- Wie kann der erstellte digitale Schaltkreis getestet und in welchem Umfang sollte dieser simuliert werden

6.1 Konfigurationsplanung

In diesem Kapitel wird die Planung der Hardwarekonfiguration erläutert. Es wird dargestellt, welche Aspekte die digitale Schaltung zur Umsetzung der Ethernetschnittstelle auf dem FPGA erfüllen muss. Um einen Überblick zu bekommen wird außerdem erläutert, in welche Komponenten der zu erstellende Schaltkreis gegliedert wird.

6.1.1 Rahmenbedingungen der Konfigurationsanforderung

Wie schon in Kapitel 2.2.2 erwähnt, wird zur 4B/5B- Kodierung, das Scrambling, sowie für die NRZI- und MLT-3- Kodierungen der auf dem Entwicklerboard integrierte PHY-Baustein „LAN83C185“ der Firma SMSC genutzt. Das Entwickeln einer Ethernetschnittstelle zum Senden eines Ethernetframes ist also an die Logik des genutzten Bausteins gebunden. In Abbildung 6-1 ist eine schematische Übersicht des Bausteins abgebildet, in welcher seine Ein- und Ausgangssignale sowie seine Logikblöcke dargestellt sind. In diesem Projekt sind die Signaleingänge zur Datenübertragung, also die Transmit- Signale, sowie das zugehörige Taktausgangssignal einzubinden. Die Einstellung der verschiedenen Übertragungsmodi und einstellbare Adresse des PHY- Bausteins, sowie das Reset- Signal sind auf dem genutzten Entwicklerboard nicht mit dem FPGA verbunden, wodurch kein direkter Einfluss auf diese Signal genommen werden kann. Modi und Adresse können über das Management- Signal und ein selbst zu generierendes Management- Taktsignal eingestellt und abgefragt werden, was jedoch zum einfachen Senden eines Ethernetframes nicht nötig ist und daher in dieser Arbeit nicht berücksichtigt werden muss. Ebenfalls müssen die Signale zum Empfangen eines Ethernetframes hier nicht berücksichtigt werden, da nur Daten übertragen und nicht empfan-

gen werden sollen. Die zwei Ausgangssignale zur Trägerabfrage und Kollisionserkennung sollen mit den vorhandenen Leuchtdioden verbunden werden um visuell eine Kollision erkennen zu können. Außerdem soll bei Kollisionserkennung das Senden des Ethernetframes unterbrochen und neu gestartet werden.

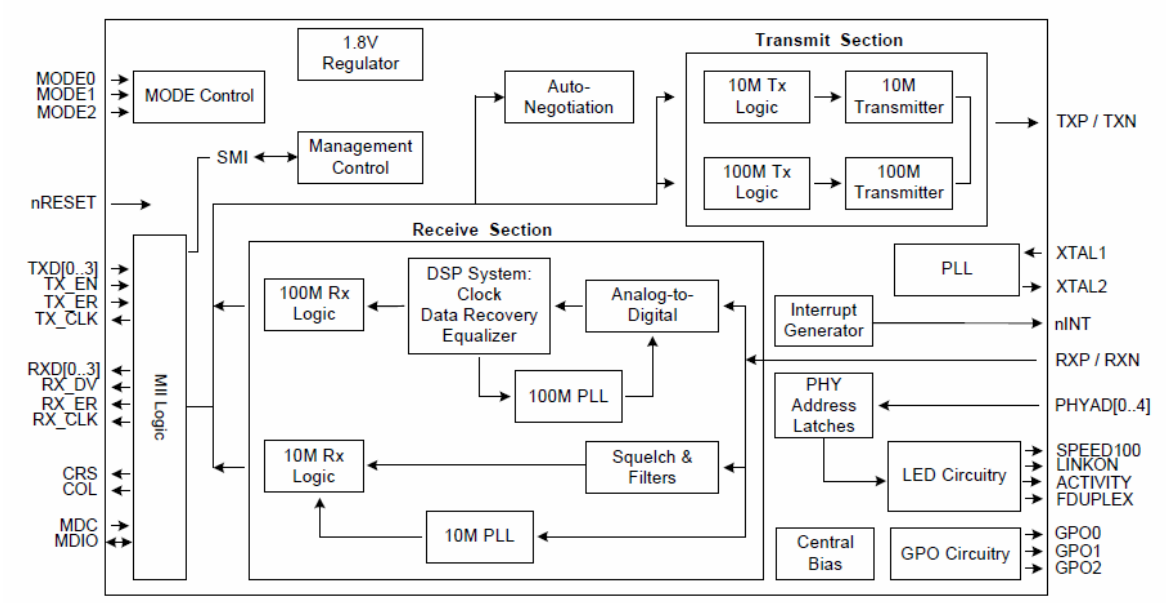


Abbildung 6-1: Übersicht des PHY- Bausteins LAN83C185 [2]

6.1.2 Konfigurationskomponenten

Eine Unterteilung des zu erstellenden Schaltkreises in Komponenten ist nötig, um den Überblick behalten zu können. Kleinere Module können leichter synthetisiert und simuliert werden, außerdem wird die Fehlersuche enorm erleichtert.

Die erste Komponente soll das Senden eines zuvor festgelegten und nicht variablen Ethernetframes an den PHY- Baustein ermöglichen. Hierzu müssen alle zur Kommunikation mit dem PHY- Baustein nötigen Signale in die Komponente mit aufgenommen werden. Außerdem muss eine zeitkritische Analyse durchgeführt werden, um gewährleisten zu können, dass der PHY- Baustein die vom FPGA ankommenden Daten verarbeiten und auf das Übertragungsmedium geben kann.

Der nächste wichtige Punkt dieses Moduls ist das Eruieren der Reihenfolge der zu übertragenden Nibble, damit der Ethernetempfänger die Nutzdaten filtern und wieder zusammensetzen kann.

Für das Senden von Daten sind in diesem Modul folgende Signale nach außen beziehungsweise in den FPGA zu führen:

- **TXD[0..3]:** Das vier Bit breite Bussignal, TransmitData, dient der Übertragung der Datennibble vom FPGA zum PHY- Baustein
- **TX_EN:** Das TransmitEnable Signal wird auf einen High- Pegel gesetzt sobald Daten am TransmitData Bus anliegen
- **TX_CLK:** Das zur Datenübertragung genutzte Taktsignal TransmitClk hat ein Tastverhältnis von 50 % und eine Frequenz von 2,5 MHz bei einer 10 MBit/s und 25 MHz bei einer 100 MBit/s Übertragungsrate. Dieses Taktsignal wird durch einen externen Quarz erzeugt.

Zusätzlich sollte ein asynchrones Signal zum Rücksetzen der Signalwerte, das so genannte Reset- Signal, integriert werden.

- **Reset:** Signal zum asynchronen Rücksetzen der Signalwerte

Die nächste Komponente beinhaltet die Generierung einer Prüfsumme mittels Schieberegister. Nach der Simulation der beiden Komponenten sollten diese zusammengeführt, wieder simuliert und in Hardware getestet werden.

Das nächste Modul sollte eine Möglichkeit schaffen, die vom Testaufbau kommenden Daten speichern und automatisiert in einen Ethernetframe integrieren zu können, um diese dann über die Schnittstelle zu senden. Als Speicher sollte ein BlockRAM dienen, welcher zu Testzwecken mit selbst festgelegten Daten initialisiert wird.

Zum Abschluss sollte das Kontrollsignal zur Erkennung einer Kollision, ein Signal zur Anzeige bei Betätigen des Reset- Signals, sowie ein Signal zur Anzeige bei neuer Framesendung, zur Visualisierung mit Leuchtdioden verknüpft werden. Beim Einsatz von Tastern ist ebenfalls eine Entprellung dieser nötig, um das Auslösen mehrerer High-Signale nach nur einem Tastendruck zu vermeiden.

- **COL:** Signal zur Kollisionserkennung aus dem PHY- Baustein

6.2 Simulation und Testbench

Die Simulation ist bei der Konfiguration von FPGAs sehr wichtig, da der Programmierer sich in kurzer Zeit einen Überblick aller Signalverläufe über einen selbst bestimmten Simulationszeitraum verschaffen und dadurch fehlerhafte Verläufe schnell erkennen kann. Aus diesen Gründen wird das Programm ModelSim PE Student Edition 6.5 der Firma MentorGraphics zur Simulation hinzugezogen, um das Verhalten der Eingangs-, Ausgangs- und internen Signale in bestimmten Zuständen prüfen zu können. Für eine Simulation kann jedes Eingangssignal per Hand gesetzt werden oder es wird eine so genannte Testbench genutzt. Diese mit der Beschreibungssprache VHDL erstellte Datei beinhaltet die Zuordnung aller im Design vorhandenen Signale und die Stimulierung der Eingangssignale. Das bedeutet für ein Taktsignal müssen nötige Informationen, wie Frequenz und Dauer des High- und Low- Pegels angegeben werden. Für ein Reset- Signal beispielsweise wird ein genauer Zeitpunkt angegeben an dem der Reset aktiv werden soll. Nach Ausführung der Testbench können alle Signale in binärer, dezimaler oder hexadezimaler Schreibweise und deren Verlauf angezeigt und auf Richtigkeit überprüft werden. In Kapitel 7 wird ein solches Simulationsergebnis überprüft und näher erläutert. Außerdem befinden sich im Anhang D der Inhalt einer Testbench und im Anhang E ein Teil des zugehörigen Simulationsergebnisses.

7 Realisierung

In diesem Kapitel wird die Umsetzung der Konfigurationsplanung beschrieben. Es wird dargestellt, welche zeitlichen Aspekte bei der Konstruktion der Signalabläufe beachtet werden müssen. Außerdem werden Ausschnitte der durchgeführten Simulation näher erläutert.

Die Erstellung der ersten Komponente, welche das Senden eines konstanten Frames über das Twisted- Pair- Kabel beinhaltet, ist der wichtigste Abschnitt. Ist diese Komponente erfolgreich in Hardware getestet worden, können die folgenden Module darauf aufbauen.

Nach Integration aller nötigen Signale, welche in Kapitel 6.1.2 aufgeführt sind, wird ein Zustandsdiagramm angefertigt, durch welches visualisiert werden kann wann sich Signale in welcher Form ändern sollen. In der Programmierung, zum Beispiel mittels der Sprache C, wird diese Form der Visualisierung durch Programmablaufpläne oder auch Struktogramme realisiert. Nachdem die Erstellung der Komponente abgeschlossen ist, werden die Kompilierung und anschließend eine Simulation mittels Testbench in ModelSim durchgeführt.

Bei der Konfiguration des Schaltkreises zum Senden eines festgelegten, gespeicherten Ethernetframes sind zwei Aspekte zu beachten. Zum einen die korrekte Reihenfolge in der die Nibble am Ausgang des FPGAs und entsprechend am Eingang des PHY- Bausteins anliegen müssen. Zum zweiten ist der richtige Zeitpunkt des Anliegens wichtig. In Abbildung 7-1 ist das Timing zur Datenübertragung dargestellt, welches durch den genutzten PHY- Baustein vorgeschrieben ist. Das Signal TX_CLK stellt das externe Taktsignal des PHY- Bausteins dar, welches einer Frequenz von 2,5 MHz für eine Übertragungsrate von 10 MBit/s oder entsprechend 25MHz bei einer Übertragungsrate von 100MBit/s beträgt. Das Bussignal TXD[3:0] stellt das zu übertragende Nibble dar und TX_EN ist das Enable[97]- Signal, welches anzeigt, dass Daten am Bus anliegen. Der Abbildung zufolge muss das Nibble die Zeit T5.1, welches laut Datenblatt [2] einer Zeit von 12 ns entspricht, vor der nächsten steigenden Flanke des Taktsignals am Bus anliegen, außerdem muss das Enable- Signal einen High- Pegel besitzen. Die in der Abbildung dargestellte Zeitangabe T5.2 beschreibt die Zeit welche das Datennibble nach der steigenden Flanke des Taktsignals anliegen muss. In dem Datenblatt ist für diese Zeit 0 ns angegeben, was bedeutet, dass die Daten direkt bei der steigenden Flanke an den PHY- Baustein übergeben werden. Daraus ergibt sich die Möglichkeit die zu übertragenden Datennibble immer bei steigender Taktflanke an den Datenbus legen zu können, ohne die zeitliche Vorgabe zu verletzen. Außerdem ist es möglich das Enable- Signal,

[97] to enable (engl.): aktivieren

für das Anzeigen vorhandener Daten zu Beginn der Framesendung zu aktivieren und erst nach dem Senden wieder auf einen Low- Pegel zu setzen.

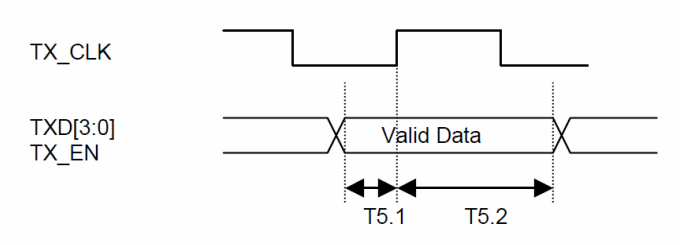


Abbildung 7-1: Timing zur Datenübertragung an den PHY [2]

Zum Senden von Daten mittels Ethernetprotokoll ist es nötig die Daten wie in Kapitel 4.5 beschrieben in einen Ethernetheader zu integrieren und alle zugehörigen Daten nibbleweise an den Datenausgang des FPGAs zu legen. Um dies realisieren zu können werden für jedes Headerfeld ein oder mehrere Zustände definiert, die nacheinander durchlaufen werden und bestimmte Signaländerungen auslösen. Diese Zustände und zugehörigen Signaländerungen sind in einem Zustandsdiagramm im Anhang B dargestellt. Die enthaltenen Zustände werden folgend kurz erläutert.

- **idle:** Dieser Zustand ist der Start- Zustand, welcher nach einem erfolgten Reset, einer Kollision oder dem Senden eines Ethernetframes eingenommen wird. Hier werden die Signale auf ihre Anfangswerte, auch defaults genannt, gesetzt.

- **sendPreamble:** Dieser Zustand sorgt, mittels eines Zählers für das Senden der Präambel.

- **sendSFD:** In diesem Zustand wird das Senden des zweiten Nibbles des Start Frame Delimiters durchgeführt.

- **sendDestMac:** Das Senden der Ziel- MAC- Adresse wird in zwölf Zustände (sendDestMacAddr_1 bis sendDestMacAddr_12) unterteilt, wobei die ungeraden Zustände das Senden der unteren vier Bit des zugehörigen Bytes und die geraden Zustände das Senden der oberen vier Bit übernehmen. Außerdem wird in den ungeraden Zuständen der Prüfsummengenerator aktiviert und das gesamte zugehörige Byte an diesen gesandt.

- **sendSourceMac:** Wie bei den Zuständen sendDestMac wird auch dieser in zwölf Zustände gegliedert und die Bytes der konstant gespeicherten MAC- Adresse an den Prüfsummengenerator gesandt.

- **sendLength:** Dieser Zustand ist in acht Teilzustände gegliedert um die vier Byte des Type/Length- Feldes, welche die Summe der Datenbytes und des LLC- Feldes enthalten, zu senden.

- **sendLLC:** Der Zustand sendLLC ist ebenfalls in acht Teilzustände gegliedert um die viert Byte, welche einen konstanten Wert besitzen, des LLC- Feldes senden zu können.

- **sendData:** Die Zustände sendData_1 und sendData_2 übernehmen das nibbleweise Senden der Nutzdaten. Mittels eines Zählers, welche hier als Grundeinstellung von 0 bis 49 läuft, können 50 Datenbytes übermittelt werden. sendData_1 sendet das niederwertige Nibble und aktiviert den Prüfsummengenerator und sendData_2 sendet das höherwertige Nibble des zugehörigen Bytes. Nachdem die 50 Datenbytes übertragen wurden kann der nächste Zustand eingenommen werden.

- **sendCRC32:** Dieser Zustand wurde in acht Teilzustände gegliedert, da die Prüfsumme aus 32 Bit besteht und die Daten dieser, wie in den anderen Zuständen auch, nibbleweise an den PHY- Baustein gesandt werden.

- **sendFrameIsSent:** Nachdem alle in einem Ethernetheader nötigen Daten gesandt wurden, wird in dem Schluss- Zustand das Transmit Enable- Signal wieder auf 0 gesetzt, so dass der PHY- Baustein keine weiteren, an dem Transmit Datenbus anliegenden Signale verarbeitet. Um keine Zufallssignale an den zuständigen Ausgängen des FPGAs zu erhalten werden diese auf den Low- Pegel gesetzt.

- **sendIFG:** Der letzte zu durchlaufende Zustand realisiert mittels eines Zählers das Senden des Inter Frame Gap. Das bedeutet, über 24 Taktzyklen wird der Datenbus auf den Low- Pegel gesetzt, so dass der Empfänger Zeit hat die ankommenden Daten zu verarbeiten bevor ein neuer Frame gesandt wird. Der Folgezustand ist der Anfangszustand idle.

Die Umsetzung dieses Zustandsdiagramms mittels VHDL ist in der Datei ethernetSend.vhd, welche sich im Anhang F.2 befindet, dargestellt.

Um die Zustandsumsetzung zu verdeutlichen ist in Abbildung 7-2 ein Ausschnitt des Simulationsergebnisses, welches durch die im Anhang D enthaltene Testbench generiert wurde, dargestellt. Der Ausschnitt zeigt den Zeitabschnitt von 0 bis 2460 ns, in welchem deutlich wird, dass erst nach einem inaktivem Reset und einem High- Pegel des newFrame- Signals das Senden von Daten an den PHY- Baustein gestartet wird. Die Zustandsanzeige „nextState“ gibt den Folgezustand an, welcher bei der nächsten fallen-

den Taktflanke eingenommen wird. Die in den jeweiligen Zuständen vorgenommenen Signaländerungen werden um einen Takt zur Anzeige verzögert durchgeführt.

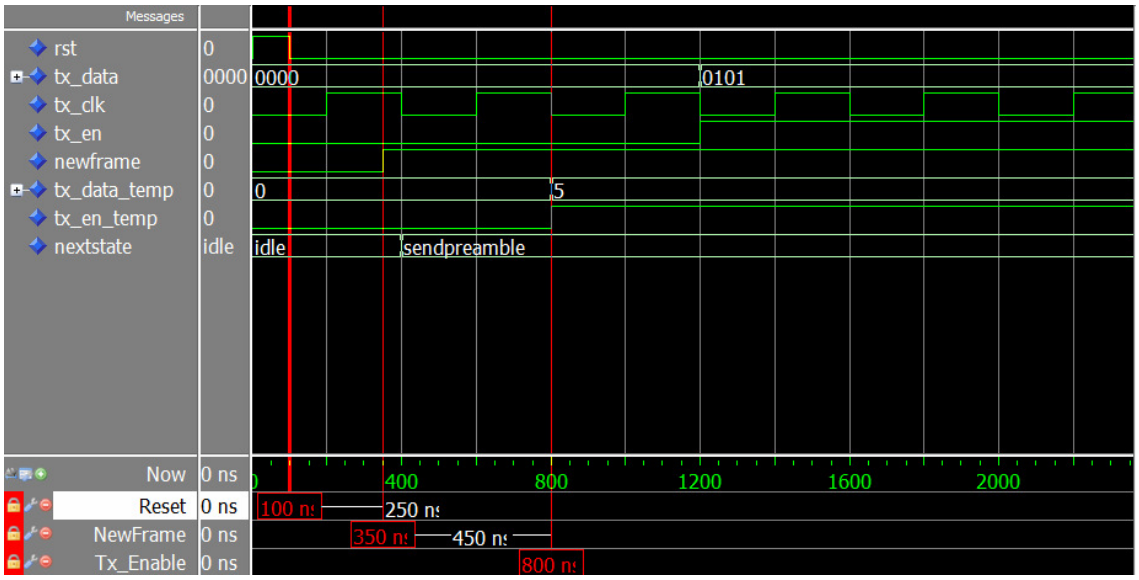


Abbildung 7-2: Ausschnitt des Simulationsergebnisses zur Darstellung der ersten Zustände

Die erste Markierung bei 100 ns zeigt die fallende Flanke des Reset- Signals an, nach welcher der Schaltkreis erst aktiviert wird. Die zweite Markierung zeigt die steigende Flanke des newFrame- Signals an, welche das Senden eines Ethernetframes auslöst. Der Zustand idle wird so lange beibehalten bis der Reset den Low- Pegel und das newFrame- Signal einen High- Pegel einnimmt. Bei der nächsten fallenden Flanke des Taktsignals „tx_clk“ wird der Folgezustand, hier sendPreamble, angekündigt. Einen Taktzyklus später ist der Folgezustand eingenommen und das temporäre Ausgangssignal „tx_data_temp“ wird auf den entsprechenden Wert gesetzt. Die Markierung bei 800 ns zeigt diesen Vorgang an. Einen Taktzyklus später wird das temporäre Signal, hier in hexadezimaler Schreibweise angegeben, an den Datenbus „tx_data“, welcher direkt an den PHY- Baustein gekoppelt ist, weitergeleitet. Dieser Daten- Bus besteht aus vier Bit und wird in binärer Schreibweise angegeben um das Senden des eigentlichen Bitstreams zu verdeutlichen. Das Taktsignal wird wie das Reset- Signal und das Signal newFrame durch die im Anhang D enthaltene Testbench generiert und beträgt eine Frequenz von 2,5 MHz. Weitere Ausschnitte der Simulation befinden sich im Anhang E.

Die Umsetzung der Prüfsummengenerierung wird mittels eines Schieberegisters, welches in Abbildung 4-13 dargestellt ist, und logischen XOR- Gattern realisiert. Die mit in die Prüfsummenberechnung einzubeziehenden Daten, wie Ziel- und Quell- MAC- Adresse, Längenfeld und Nutzdaten, werden, wie zuvor erwähnt durch die entsprechenden Zustände, byteweise in den Prüfsummengenerator geladen. Hier werden die Daten mit Hilfe einer Signalschleife und der XOR- Gatter innerhalb eines Taktzyklus kodiert und zwischengespeichert. Sobald alle mit einzubeziehenden Daten kodiert sind, kann die 32

Bit lange Prüfsumme über einen Bit- Vektor an den Sendekontrollprozess übergeben und dann nibbleweise an den PHY- Baustein gesandt werden.
Abbildung 7-3 zeigt einen Ausschnitt der Simulation zur Darstellung der Prüfsummengenerierung. Die erste Markierung bei 7200 ns zeigt die steigende Flanke des Signals „newFrameCRC“, welches den Prüfsummengenerator einmalig aktiviert. Diese Aktivierung erfolgt in dem Zustand „sendSFD“, dessen Folgezustand „sendDestMac_1“ ist. Da in dem Zustand „sendDestMac_1“ die ersten zur Prüfsummengenerierung nötigen Daten an den PHY- Baustein und an den Prüfsummengenerator gesandt werden, muss die Aktivierung des Generators einen Taktzyklus vor Sendung der Daten vorgenommen werden. Damit ist sichergestellt, dass die entsprechenden Signale keine zufällig angenommenen Pegel besitzen. Im Zustand „sendDestMac_1“ wird bei fallender Taktflanke das erste zu kodierende Byte, hier ‚00’, über den acht Bit breiten Datenbus „crc32ByteIn“ in das Schieberegister geladen. Sobald das Signal Enable- Signal „crc32_en“ einen High- Pegel besitzt wird der temporäre 32 Bit breite Vektor „crc32DataTemp“ generiert und bei der nächsten steigenden Taktflanke negiert an den Ausgangsvektor „crc32DataOut“ gegeben. Mit der Sendung der Datenbytes wird in dem Zustand „sendDestMac_1“ zusätzlich das Signal „crc32_en“ gesetzt um die Prüfsummengenerierung nur bei jedem zweiten Taktzyklus zu aktivieren. Diese Vorgehensweise wurde gewählt, damit die Komponenten mit einem einheitlichen Takt arbeiten können, denn der Prüfsummengenerator benötigt ein Byte als Eingangssignal, der PHY- Baustein nur ein halbes Byte. Die Beschreibung des Schaltkreises zur Prüfsummengenerierung befindet sich in der Datei crc32_gen.vhd, welche im Anhang F.3 dargestellt ist.

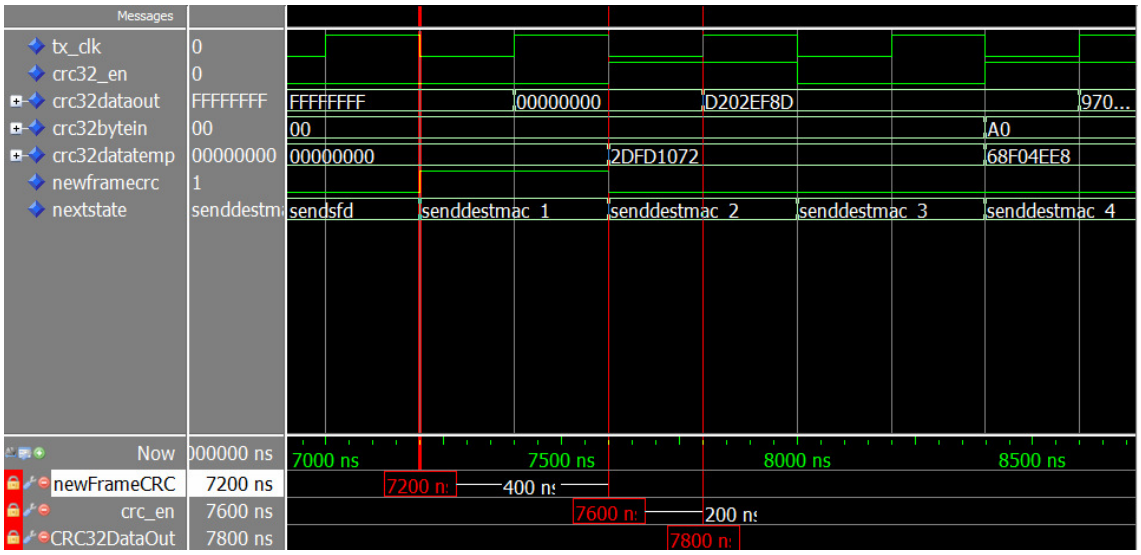


Abbildung 7-3: Ausschnitt des Simulationsergebnisses zur Darstellung der Prüfsummengenerierung

Nach erfolgreichem Test der Module in Hardware, welcher in Kapitel 8 näher erläutert wird, kann die nächste Komponente, welche das Auslesen von Daten aus einem BlockRAM beinhaltet, realisiert werden.

Die Umsetzung eines BlockRAMs, mittels VHDL, auf einem FPGA besteht aus der Deklaration einer Matrix. Hier wurde eine Größe von 256 Speicherelementen, welche je acht Bit umfassen, angelegt. Zu Testzwecken wurden die ersten 50 dieser Speicherelemente mit Zahlen von ‚01' bis ‚50' initialisiert. In Abbildung 7-4 ist ein Ausschnitt der Simulation zur Verdeutlichung der für den BlockRAM spezifischen Signale dargestellt. Die unterste Zeile in dieser Grafik zeigt die initialisierten Speicherzellen. Die Speicherzellen können über ihre jeweilige Adresse über einen acht Bit breiten Vektor direkt angesprochen werden. Dieser Vektor „ramAddr" besteht nicht, wie die übrigen Signale, aus Bit- Signalen, sondern aus so genannten standard_logic- Signalen. Diese Signale können nicht nur die Werte ‚0' und ‚1' für Low- und High- Pegel annehmen, sondern unter anderem auch den Wert ‚X', welcher für den undefinierten Zustand steht. Die erste Markierung bei 21600 ns zeigt das Ende des undefinierten Zustands für den Signalvektor „ramAddr" an. Da der BlockRAM an dieser Stelle sein Enable- Signal über „ram_en" erhält, wird dem Adressvektor sein default- Wert ‚00000000' zugewiesen und ist jetzt grün anstatt rot gekennzeichnet. Um solche undefinierten Signalzustände zu vermeiden, werden in dieser Arbeit so weit es möglich ist Bit- Signale, mit nur zwei Zustandsmöglichkeiten eingesetzt. Da allerdings die Speicherzellen direkt über ihren Index angesprochen werden, muss die Adressierung über standard- logic- Signale realisiert werden. Durch den letzten Zustand „sendLLC_8" vor Beginn des Lesens und Sendens der Nutzdaten wird das enable- Signale für den BlockRAM gesetzt und die erste Adresse der Speicherzelle übergeben. Die Daten, welche unter diesem Speicherort stehen werden bei der nächsten steigenden Taktflanke an den Ausgang gelegt und das erste Datennibble kann über den Zustand „sendData_1" bei fallender Taktflanke an den PHY- Baustein weitergeleitet werden. Zeitgleich wird ein Zähler inkrementiert welcher als Adresse der nächsten Speicherzelle dient und im Zustand „sendData_2" übergeben wird. Die Markierung bei 22400 ns zeigt das Anlegen des niederwertigen Datennibbles des ersten zu übertragenden gespeicherten Datenbytes. Die Markierung bei 22800 ns zeigt die Übertragung des höherwertigen Datennibbles des entsprechenden Bytes.

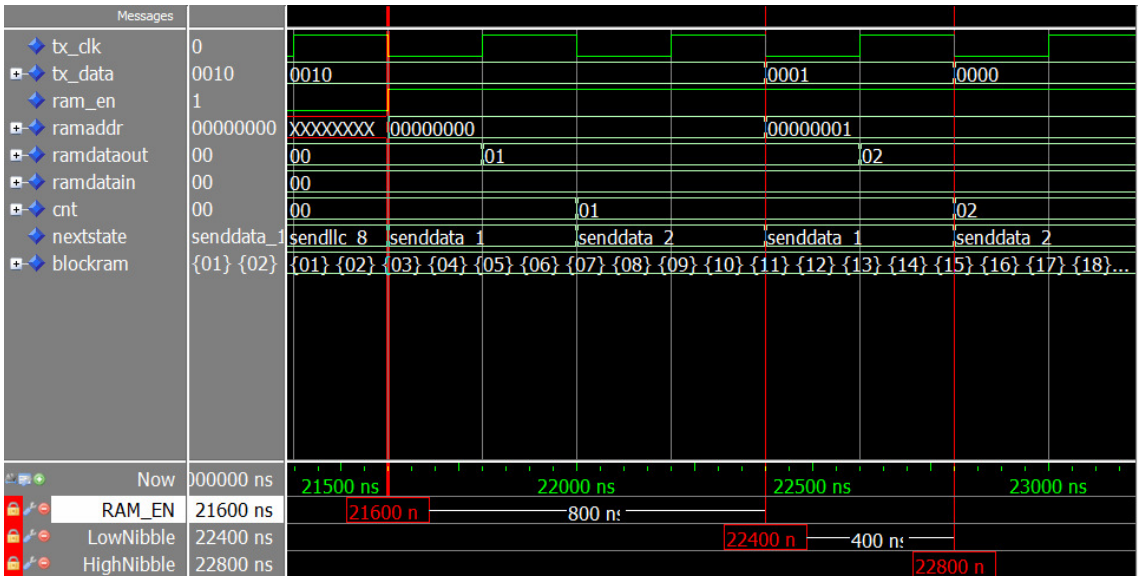


Abbildung 7-4: Ausschnitt des Simulationsergebnisses zur Darstellung der Block- RAM- Aktivitäten

Auch das Auslesen und Übertragen der im BlockRAM gespeicherten Daten wird, wie das Senden des EthernetFrames und die Prüfsummengenerierung, durch das Taktsignal „tx_clk“ in der Simulation mit einer Frequenz von 2,5 MHz gesteuert. Einzig zum Schreiben von Daten in die Speicherzellen wurde das Taktsignal „clk50“ mit einer Frequenz von 50MHz integriert. Der Prozess zum Beschreiben des BlockRAMs wurde zwar integriert ist jedoch für diese Arbeit nicht relevant, daher wird auf diesen Teil nur kurz in der Zusammenfassung eingegangen.

Der Quelltext zum BlockRAM befindet sich in der Datei blockRAM.vhd und ist im Anhang F.4 dargestellt.

Die zuvor vorgestellten Komponenten „ethernetSend“, „crc32_gen“ und „blockRAM“ müssen in einer so genannten Top- Level[98]- Architektur miteinander verknüpft werden, damit diese untereinander die entsprechenden Signale austauschen können. In dieser Top- Level- Architektur sind die eigentlichen Ein- und Ausgangssignale definiert, welche in oder aus den FPGA geleitetet, und außerhalb des Bausteins mit Tastern, Leuchtdioden und dem PHY- Baustein verknüpft werden. Um diese Signalweiterleitung zu visualisieren ist im Anhang C ein so genanntes Blockdiagramm dargestellt. In diesem Blockdiagramm sind die einzelnen, eingebundenen Komponenten und deren Signalbahnen abgebildet. Der zugehörige Quelltext ist in der Datei Top_level.vhd enthalten und im Anhang F.1 dargestellt.

Ebenfalls ist in der Top- Level- Architektur das Entprellen des Tasters zum Auslösen der Sendung eines Ethernetframes enthalten. Dies ist nötig um das Senden des Frames nur einmalig auszulösen. Da die Taster, welche auf dem Entwicklerboard integriert sind,

[98] top- level (engl.): höchste Ebene

durch hardwarebedingte Pegelschwankungen oder lang andauernder Betätigung mehrere High- Pegel auslösen, kommt es zur Sendung mehrerer Ethernetframes. Die Sendung eines Frames benötigt bei einer Übertragungsrate mit 100MBit/s cirka 0,7 µs, das Drücken eines Tasters nimmt sehr viel mehr Zeit in Anspruch, wodurch das Signal zur Sendung eines Frames auch nach der ersten Sendung noch einen High- Pegel besitzt und dadurch eine neue Sendung auslöst. Mittels eines Zählers, welcher bei jedem Takt und vorhandenem High- Pegel inkrementiert wird, kann durch das Zuweisen eines High-Pegels an das Signal „newFrame“ bei zuvor festgelegtem Zählerstand das Auslösen der Sendung mehrerer Frames vermindert werden.

8 Test

In diesem Kapitel wird der Test der Ethernetschnittstelle dargestellt. Außerdem werden Probleme des Systemtests und Testergebnisse näher erläutert.

Jede im Kapitel 6.1.2 beschriebene Komponente wurde, nach der mittels ModelSim durchgeführten Simulation, auf der Hardware getestet. Um solch einen Test realisieren zu können, muss die Beschreibung des Schaltkreises mittels der Software ISE Webpack von Xilinx synthetisiert werden. Dieses Programm generiert das Bitfile, welches über die USB- Schnittstelle an den FPGA gesandt und zur Konfiguration dieses führt. Um prüfen zu können ob die Konfiguration vollzogen wurde, wurden Testsignale nach außen geführt und mit Leuchtdioden, welche auf dem Entwicklerboard integriert sind, verknüpft. Zum einen soll die LED 6 ein High- Signal empfangen, also leuchten, sobald das durch einen Taster betätigte RESET- Signal aktiviert wurde. Zum anderen soll die LED 7 leuchten, sobald die Übertragung eines Ethernetframes beginnt. Die Leuchtdiode LED 5 zeigt an ob der BlockRAM derzeit beschrieben wird und LED 4 zeigt die Aktivitäten des Kollisionerkennungssignals an. Um testen zu können ob ein Ethernetframe auf dem Zielrechner empfangen wurde, ist es nötig die integrierte Netzwerkkarte auf Aktivitäten abzufragen. Dieses Abfragen der Netzwerkkarte ist durch das Programm Wireshark möglich. Durch die in diesem Programm wählbare promiscous Einstellung ist es möglich alle Daten, welche über das angeschlossene Übertragungsmedium empfangen werden, anzeigen zu lassen. Promicous bedeutet dass auch empfangene Ethernetframes angezeigt werden, die nicht die MAC- Adresse des Zielrechners als Zieladresse eingetragen haben. Das heißt, falls die Ziel- MAC- Adresse nicht korrekt übertragen wird, kommt es dennoch zur Anzeige des Frames auf dem Zielrechner. Die über das Übertragungsmedium empfangenen Daten werden in dem Programm Wireshark in hexadezimaler Schreibweise angezeigt. Da die zu sendenden Daten durch den selbst erstellten Ethernetcontroller ebenfalls in hexadezimaler Schreibweise an den Transmit- Bus übergeben werden, kann leicht überprüft werden, ob die Daten korrekt übermittelt wurden. In Abbildung 8.1 ist der Testaufbau schematisch dargestellt. Zu erkennen sind hier das zur FPGA- Konfiguration nötige Programm ISE Webpack mit USB- Schnittstelle und das zum Auslesen der Netzwerkkarte nötige Programm Wireshark mit entsprechendem Übertragungsmedium. Außerdem sind die erwähnten Leuchtdioden und einer der Taster dargestellt.

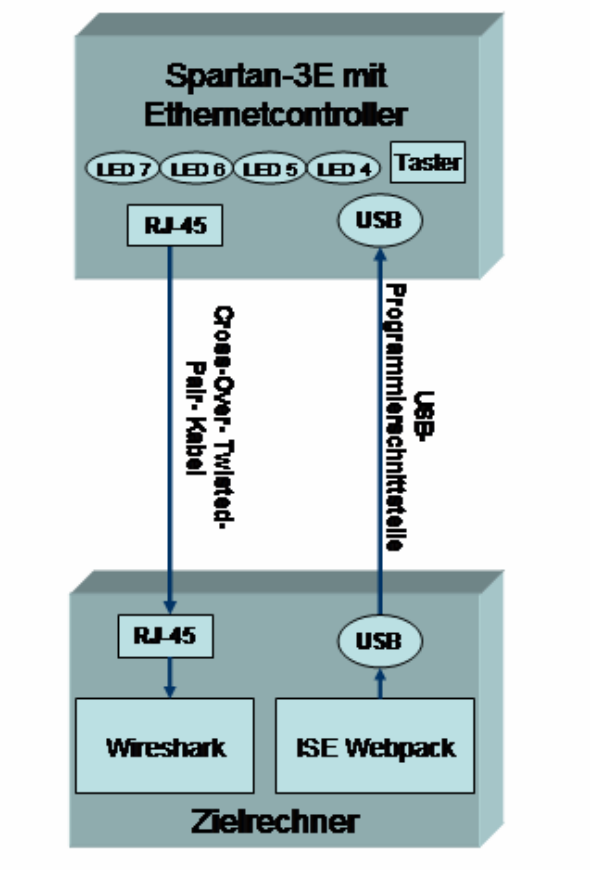


Abbildung 8-1: Schematische Darstellung des Testaufbaus

Nach dem Aufbau der Hardware kann die Generierung des Bitfiles durch ISE Webpack durchgeführt werden, zudem ist die Einbindung der Quelltexte und der UCF[99] nötig. Die UCF ist eine Datei, in welcher die Verknüpfung der Ein- und Ausgänge des FPGAs mit den entsprechenden Ein- und Ausgängen anderer, auf dem Entwicklerboard integrierter, Bauelemente beschrieben wird. Über die vorhandene USB- Programmierschnittstelle wird dann der FPGA konfiguriert. Ist das Cross- Over- Twisted- Pair- Kabel bereits angeschlossen, können schon vor der Aktivierung des digitalen Schaltkreises Datenübertragungen in Wireshark festgestellt werden. Diese Sendungen von Frames werden durch die Netzwerkkarte des Zielrechners ausgelöst. Bei den meisten Netzwerkkarten wird bei Anschluss eines Netzwerkkabels ein so genannter Broadcast[100] gesandt, welcher zur Datenübertragung an alle Teilnehmer im Netzwerk genutzt wird. Dieses Verfahren wird beispielsweise zum Senden von ARP[101]- Frames herangezogen, um jeder am Netz angeschlossenen Station die eigene MAC- Adresse bekannt zu geben. Nach Senden dieses Broadcasts und einigen, hier nicht näher erläuterten, Selbsttests der Netzwerkkarte des Zielrechners, wird durch Drücken des entsprechenden Tasters auf dem Entwicklerboard die Sendung eines Ethernetframes vom FPGA aus aktiviert. Nach kurzer Zeit erscheint auf dem Bildschirm des Zielrechners eine neue Zeile in dem Pro-

[99] User Constraints File (engl.): Anwender Grenzdatei
[100] broadcast (engl.): Sammelaufruf
[101] Address Resolution Protocol (engl.) : Adressauflösungsprotokoll

gramm Wireshark. In Abbildung 8-2 ist ein solch gesendeter Ethernetframe mit seinen dekodierten Daten dargestellt.

Die Markierung ‚1' zeigt eine Zeile, welche dem, vom FPGA aus gesandten und am Zielrechner empfangenen, Ethernetframe entspricht. Hier sind die Nummer des empfangenen Frames, Ankunftszeit, Quell- und Zieladresse, Protokolltyp und Protokollinformationen dargestellt. Die Markierung ‚2' zeigt weitere Informationen zu dem empfangenen Frame an. Hierbei werden die genaue Ankunftszeit des Frames mit Datum und Uhrzeit, sowie Differenzzeiten zu anderen Frames, die Framelänge in Byte und welche Protokolle im Frame enthalten sind angegeben. Die Länge des Frames beträgt hier 68 Byte, welche sich aus Länge der Ziel- und Quelladresse mit je 6 Byte, dem Längenfeld mit 2 Byte, dem LLC- Feld mit 4 Byte und dem Datenfeld mit 50 Byte zusammensetzt. Die im Frame enthaltenen Protokolle werden hier als „eth" für Ethernet, „LLC" für die LLC- Datenfeldangabe und „data" für das Datenfeld angegeben.

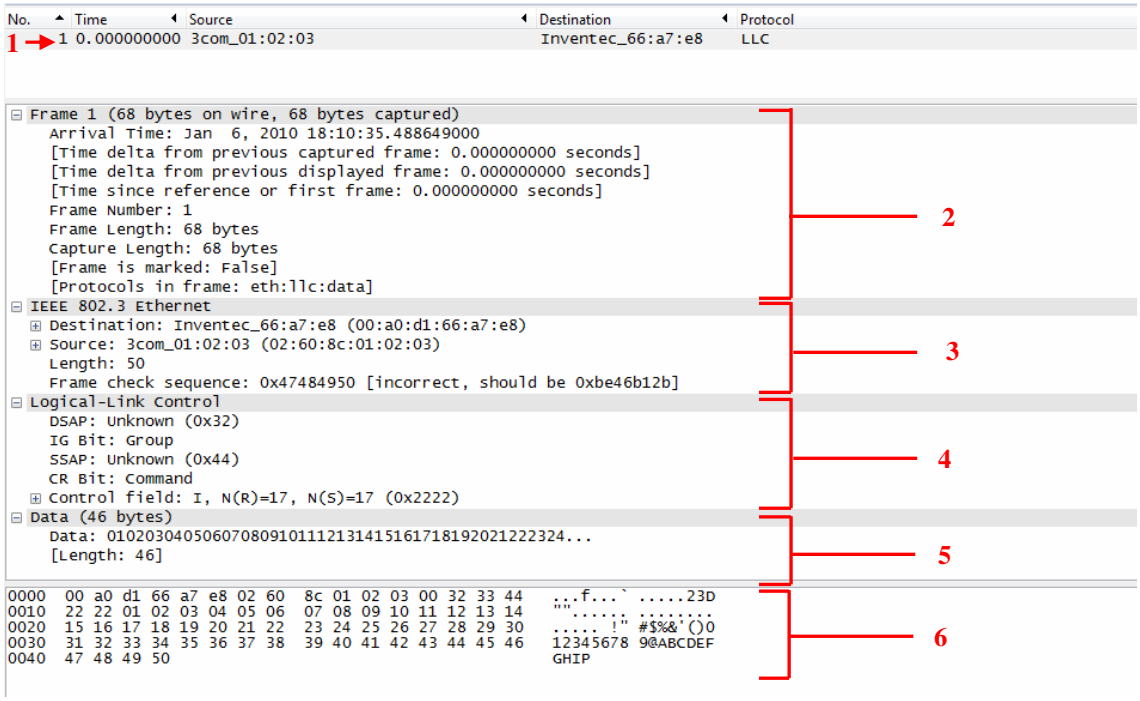


Abbildung 8-2: Screenshot zur Darstellung eines mit Wireshark empfangenen Ethernetframes

Die Markierung ‚3' zeigt an, dass es sich bei dem empfangenen Frame um den Ethernetheader IEEE 802.3 handelt, hier werden die Zieladresse und Quelladresse detailliert angezeigt. Nicht nur die eigentliche aus hexadezimalen Zahlen bestehende MAC- Adresse wird angegeben, sondern auch der Name des Herstellers der Netzwerkkomponente, welcher sich hinter den höheren drei Byte der MAC- Adresse verbirgt. Um leichter erkennen zu können, ob der vom FPGA gesandte Frame auf dem Zielrechner angezeigt wird, wurde in dieser Arbeit eine MAC- Adresse des Herstellers 3com gewählt und die Zahlen ‚01', ‚02' und ‚03' angehängt. Außerdem wird die Prüfsumme angezeigt, welche hier allerdings als inkorrekt angegeben wird. Diese Anzeige wird auf eine Fehlin-

terpretation des Programms Wireshark zurückgeführt. Durch die Analyse anderer Protokollframes, wie der vom Zielrechner selbst gesandte Broadcast, der Frame von einem anderen PC als Sendestation oder eines mittels des Programms PakETH erstellten Frames, wurde festgestellt, dass die angezeigte Prüfsumme überall als inkorrekt angegeben wird.
Die Markierung ‚4' zeigt Informationen zum LLC- Feld an. Das LLC- Feld wurde in dieser Arbeit nachträglich eingefügt und mit festen Werten belegt, da das Programm Wireshark ansonsten die ersten vier Byte des Datenfeldes als LLC- Feld interpretierte und dies eine falsche Längenangabe sowie verkürzte Nutzdatenangabe zur Folge hatte. Das LLC- Feld besteht aus einem Byte für den DSAP, einem Byte für den SSAP und zwei Byte, die als Kontrollfeld dienen. Da dieses Feld zur Angabe von höher angelegten Protokollen dient, welche in dieser Arbeit jedoch nicht genutzt werden, wird hier nicht näher auf die Bedeutung eingegangen. Die Markierung ‚5' in Abbildung 8-2 zeigt das übertragene Datenfeld und die Länge des Feldes in Byte an. An dieser Stelle soll noch mal auf die inkorrekte Prüfsumme hingewiesen werden, die hier aus den hexadezimalen Daten „ 0x47484950" besteht. Diese vier Byte wurden als Nutzdaten gesandt, durch das Programm Wireshark jedoch als Prüfsumme interpretiert, woraufhin diese als inkorrekt angezeigt wird. Auch dieses Verhalten wird auf Wireshark zurückgeführt. Getestet wurde dies durch das Senden eines Ethernetframes mit einer um vier Byte erhöhten Längenangabe, damit Wireshark auch die letzten vier Byte des Datenfeldes als Nutzdaten und nicht als Prüfsumme einliest. Dieser Test hatte jedoch eine Fehlermeldung unter Wireshark zur Folge, in welcher angezeigt wurde dass die im Längenfeld angegebene Anzahl der Datenbytes zu hoch sei. Außerdem wurde das Senden eines Ethernetframes mit falscher Prüfsumme vorgenommen. Dieser Test ergab dass kein empfangener Frame unter Wireshark angezeigt wurde. Daraus lässt sich schließen, dass die gesandte Prüfsumme korrekt ist und nur die Interpretation der Daten unter Wireshark Fehler aufweisen.

Die Markierung ‚6' zeigt die Daten des dekodierten Ethernetframes, welche in folgender Tabelle noch mal aufgeschlüsselt dargestellt sind, um die Bedeutung der Daten besser nachvollziehen zu können.

Daten	**Bedeutung**	**Headerfeld**
00 a0 d1 66 a7 e8	Ziel- MAC- Adresse	DestMACAddr
02 60 8c 01 02 03	Quell- MAC- Adresse	SourceMACAddr
00 32	50 Byte	Type/Length- Field
33 44 22 22	feste Werte	LLC- Field
01 02 03 04 05 06 07 08 09 10 11 12 13 14 1 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50	Nutzdaten Unter Wireshark werden die letzten 4 Byte der Nutzdaten als Prüfsumme interpretiert.	Datenfeld Frame Check Sequence

Tabelle 8-1: Dekodierte Daten des Ethernetframes und ihre Bedeutung

9 Analyse der genutzten Ressourcen des FPGAs

In diesem Kapitel wird beschrieben welche Ressourcen des genutzten FPGAs durch die Implementierung des Schaltkreises benötigt werden. Außerdem wird eine zeitliche Analyse angegeben.

In Tabelle 9-1 ist ein Ausschnitt des so genannten Summary Reports[102], welcher durch ISE Webpack bei der Synthetisierung des Quelltextes erstellt wird, dargestellt. Die erste Spalte zeigt die Ressource, die zweite Spalte wie viele dieser Ressourcen genutzt werden. Die Spalte „Available“ gibt an wie viele der Elemente auf dem Baustein vorhanden sind. Die letzte Spalte gibt eine prozentuale Auskunft der genutzten Ressourcen.Eine der höchsten Auslastung, mit 8%, besteht bei den BUFGMUXs, welche spezielle Treiber zur Taktsteuerung sind. Diese werden eingesetzt um in jedem Block dasselbe Taktsignal zu erhalten. In dieser Arbeit sind zwei Taktsignale „tx_clk“, welches in allen Komponenten genutzt wird, und „clk50“, welches nur zum Beschreiben des BlockRAMs eingesetzt wird, konfiguriert.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	218	9,312	2%
Number of 4 input LUTs	380	9,312	4%
Number of occupied Slices	264	4,656	5%
Number of Slices containing only related logic	264	264	100%
Number of Slices containing unrelated logic	0	264	0%
Total Number of 4 input LUTs	416	9,312	4%
Number used as logic	338		
Number used as a route-thru	36		
Number used for 32x1 RAMs	42		
Number of bonded IOBs	19	232	8%
Number of BUFGMUXs	2	24	8%
Average Fanout of Non-Clock Nets	3.78		

Tabelle 9-1: Ausschnitt des Summary Reports zur Darstellung der genutzten Ressourcen

[102] Summary Report (engl.): Übersichtsreport

Die Ressource „bonded IOBs“ besitzt ebenfalls eine Auslastung von 8% und stellt die genutzten verdrahteten Ein- und Ausgänge des FPGAs dar. Diese könnte sogar verringert werden, indem die nur zu Testzwecken gedachten LED- Ausgaben entfernt werden. Die Auslastung der vorhandenen Slices und LUTs sind in diesem Projekt sehr gering, was die Möglichkeit verschafft das Projekt um einiges erweitern zu können ohne einen größeren Baustein nutzen zu müssen.

In folgendem Textfeld ist ein Ausschnitt des Timing Summary Reports dargestellt, welcher die maximal zu erreichende Frequenz bezüglich des Taktsignals „tx_clk“ aufzeigt. Diese maximale Frequenz ergibt sich aus der längsten in dem Design vorhandenen Laufzeit eines Signals zwischen einer steigenden und fallenden Taktflanke. Dieser Signalpfad, welchen das Signal durchläuft, wird auch als „kritischer Pfad“ bezeichnet. Hier liegt er zwischen den Blöcken „ethernetSend“ und „blockRAM“ und hat eine Laufzeit von 5,267 ns. Daraus ergibt sich eine Taktperiode von 10,534 ns was einer Frequenz von 94,931 MHz entspricht. Das bedeutet, die in dieser Arbeit entwickelte Architektur könnte mit dieser Frequenz betrieben werden.

```
Timing Summary:
---------------
All values displayed in nanoseconds (ns)
=========================================================
Timing constraint: Default period analysis for Clock
'tx_clk'
Clock period: 10.534ns (frequency: 94.931MHz)
Total number of paths / destination ports: 3950 / 226
---------------------------------------------------------
Delay: 5.267ns (Levels of Logic = 4)
Source: init_ethernetSend/ramAddr_0_1 (FF)
Destination: init_blockRAM/ramDataOut_0 (FF)
Source Clock: tx_clk falling
Destination Clock: tx_clk rising
```

10 Erweiterungsmöglichkeiten

In diesem Kapitel werden Möglichkeiten zur Erweiterung der in diesem Projekt entwickelten Ethernetschnittstelle erläutert.

Eine mögliche Erweiterung dieses Projekts wäre die Übertragung der Daten per W-LAN, also eine Übertragung über ein lokales Netzwerk ohne feste Verdrahtung. Der Vorteil wäre hierbei dass die Kosten für die Twisted- Pair- Kabel und die nötigen Hubs entfallen, sowie die unabhängige Standortwahl des Zielrechners. Der Zielrechner könnte in einem anderen Raum als in dem des Testaufbaus stehen. Für diese Erweiterung wäre allerdings für jede Ethernetschnittstelle ein W- LAN- Adapter notwendig, welcher die aus dem PHY- Baustein kommenden Daten in Funksignale wandelt und so über das Übertragungsmedium Luft senden kann. Außerdem ist das Senden der richtigen Ziel-MAC- Adresse von großer Wichtigkeit, da, anders als bei fest verdrahteter Übertragung, die Daten bei einem anderen Rechnersystem oder auch gar keinem ankommen können.

Eine weitere Erweiterungsmöglichkeit wäre Power over Ethernet. Das bedeutet es werden nicht zur Datenübertragung genutzte Anschlüsse des Twisted- Pair- Kabels zur Übertragung einer Spannung genutzt. Somit könnte die Versorgungsspannung, welche der FPGA benötigt, über das vorhandene Kabel anliegen. Dadurch würden die Anschlüsse und Verkabelung zur Spannungsversorgung entfallen. Diese Erweiterung wäre natürlich nur bei einer festen Verdrahtung der Ethernetschnittstelle mit dem Zielrechner oder Hub möglich, würde also die vorangegangene W- LAN- Erweiterung ausschließen.

Eine Re- Konfiguration des FPGAs über die Ethernetschnittstelle könnte eine weitere Möglichkeit der Erweiterung sein. Bei derzeitiger FPGA- Konfiguration ist es beispielsweise nötig, die MAC- Adresse des Zielrechners als Konstante in den Schaltkreis zu integrieren. Sobald eine andere Netzwerkkarte zum Empfang der Daten eingesetzt werden soll, muss der Systembetrieb unterbrochen und der FPGA neu konfiguriert werden. Um diese Unterbrechung des Systems bei dem Einsatz neuer Zielrechner oder anderer Komponentenerweiterungen minimieren zu können, könnte eine automatisierte Re- Konfiguration mittels eines Platform Flash PROM, welches im Abschnitt 2.2.1 kurz vorgestellt wurde und ebenso auf dem Entwicklerboard integriert ist, vorgenommen werden. Dazu müsste die Ethernetschnittstelle um eine Komponente zum Empfangen von Daten über Ethernet erweitert werden. In diesem Projekt ist nur das Senden von Daten vorgesehen. Diese Empfänger müssten den empfangenen Ethernetframe entpacken und die Datenpakte in die korrekte Reihenfolge in das Platform Flash PROM schreiben. Nach der Speicherung müsste ein Signal den Betrieb des FPGAs kurzzeitig unterbrechen, damit eine Re-Konfiguration durch das Flash PROM möglich ist.

Die Einbindung höherer Protokolle, wie IP, UDP und TCP wäre ebenso eine Möglichkeit der Erweiterung. Dadurch wären auch höhere Schichten des OSI- Referenzmodells abgedeckt und die Daten können auf dem Zielrechner leichter ausgewertet werden, da diese nicht nur mit dem Programm Wireshark, sondern auch mit Internet- Browser oder ähnlichem ausgelesen werden könnten. Zur Einbindung von IP und UDP wäre nur eine Erweiterung des Ethernetheaders in den entsprechenden VHDL- Dateien nötig.

Ebenso wäre die Einbindung des ARP- Protokolls von großem Vorteil. Derzeit ist die MAC- Adresse des Zielrechners als Konstante im FPGA gespeichert und die zu sendenden Daten werden nur an die bestimmte Station gesandt. Mit Hilfe des ARP- Protokolls könnte über die Ethernetschnittstelle die Zieladresse einer anderen Netzwerkkarte ermittelt und gespeichert und dadurch die Daten an diese übertragen werden. Um diese Umsetzung realisieren zu können, müsste der Ethernetheader, wie bei den zuvor genannten Protokollen, erweitert und ein zusätzlicher Speicher angelegt werden. Dieser Speicher würde dem Anlegen einer ARP- Tabelle, in welche Zieladressen abgelegt werden können, dienen. Außerdem müsste eine Komponente zum Empfangen von Daten implementiert werden um die Adressen aller Stationen empfangen zu können.

11 Zusammenfassung

Die Aufgabe dieser Arbeit bestand in der Implementierung einer Ethernetschnittstelle auf einem FPGA, um eine Möglichkeit zu schaffen, die auf diesem Baustein gespeicherten Daten über ein Netzwerk senden zu können. Um die Zusammenhänge der Datensendung über Netzwerke zu verdeutlichen, wurden Netzwerktopologien und die Funktion von Netzwerkprotokollen, in Beziehung zum OSI- Referenzmodell dargestellt. Des Weiteren wurde das Ethernetprotokoll anhand des Standards 802.3u vorgestellt, durch welchen der Aufbau der zu sendenden Daten klar wurde. Zur Implementierung war die Darstellung der Funktionen des FPGAs und des PHY- Bausteins nötig, um einen entsprechenden Quelltext zur Konfiguration des FPGAs erstellen zu können. Die zur Simulation, zum erstellen des Bitfiles und zum Test benötigte Software wurde vorgestellt, um die praktische Umsetzung nachvollziehen zu können. Das Senden von, in einem FPGA gespeicherten, Daten wurde erfolgreich getestet und die Ergebnisse erläutert.
Damit wurde in dieser Arbeit eine Ethernetschnittstelle erfolgreich auf einem FPGA implementiert. Der hier entwickelte digitale Schaltkreis dient als Grundgerüst zur Weiterentwicklung, um die Ausleseelektronik des OPERA- Experiments durch FPGAs zu ersetzen. Die Vorteile des Einsatzes von FPGAs sind die Flexibilität bei der Entwicklung von digitalen Schaltkreisen durch mögliche Re-Konfiguration und ein geringer Kostenaufwand.

Im Folgenden sollen noch kurz einige Dinge bezüglich des entwickelten Schaltkreises erläutert werden, welche bisher nicht vorgestellt wurden, aber von allgemeinem Interesse sein könnten.

- Das Speichern von Daten in den BlockRAM wurde nur integriert um eine Weiterentwicklung des Schaltkreises zu erleichtern und daher nicht getestet. Dieser Test ist entfallen, da es keine leicht umzusetzende Möglichkeit gab, externe Daten an einen Eingang des FPGAs zu legen.

- Die in der Top- Level- Architektur enthaltene Entprellung des Tasters, durch welchen das Senden eines Ethernetframes ausgelöst wird, kann bei Einbindung eines kurz andauernden Signals entfallen. Das Senden des kleinsten Ethernetframes dauert bei einer Taktfrequenz von 25 MHz 6640 ns. Das Signal zum Auslösen einer Datensendung sollte daher höchstens einen andauernden High-Pegel von 6µs besitzen.

- Um das gesamte Simulationsergebnis erhalten zu können, müssen alle vorhandenen VHDL- Dateien, inklusive der Testbench, in ein simulationsfähiges Programm, wie ModelSim, eingebunden werden. Das gesamte Simulationsergebnis ist in dieser Arbeit leider nicht vorhanden, da sich dieses mindestens über einen Zeitraum von 7000 ns erstreckt und nicht durch Abbildungen darstellbar ist.

Anhang

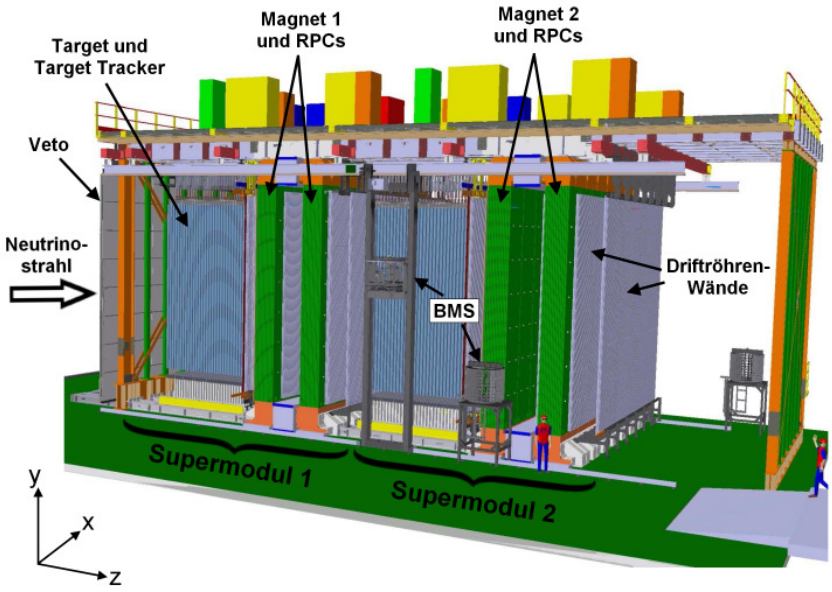
A Neutrinooszillation und Aufbau des OPERA- Detektors

Im Folgenden wird der Aufbau des OPERA- Detektors vereinfacht dargestellt und erläutert. Das OPERA- Experiment soll die Oszillation von Myon- Neutrinos zu Tauon-Neutrinos nachweisen, indem Tauon- Neutrinos in einem rein myonischen Strahl aufgezeigt werden. Neutrinos sind elektrisch neutrale Elementarteilchen, welche unter anderem in großer Zahl bei Kernfusionsprozessen der Sonne entstehen. Diese Elementarteilchen werden in drei Unterkategorien gegliedert: Elektron- Neutrinos, Myon- Neutrinos und Tauon- Neutrinos. Die Vermutung der Oszillation von Neutrinos untereinander entstand durch das in den sechziger Jahren erkannte solare Neutrinodefizit. Auf dieses Defizit wurden Wissenschaftler durch das Homestake- Experiment aufmerksam. In dem Homestake- Experiment wurde der solare Neutrinostrom mittels eines Elektron- Neutrinodetektors, welcher in der Homestake- Goldmine in South Dakota/USA in 1400 Metern Tiefe errichtet wurde, untersucht. Der gemessene Neutrinofluss entsprach aber nicht dem durch das Standard- Sonnen- Modell (SSM) berechneten erwarteten Fluss. Mögliche Erklärungen für das Defizit waren unter anderem ein inkorrektes SSM oder eine Oszillation der Elektron- Neutrinos zu anderen Neutrinos, welche nicht durch den Elektron- Neutrinodetektor nachgewiesen werden können. Im Jahre 1999 wurde das SNO[103]- Experiment in Betrieb genommen, durch welches über einen Zeitraum von sieben Jahren nachgewiesen wurde, dass die Summe der bekannten Elektron- Neutrinos, Myon- Neutrinos und des im Jahr 2000 nachgewiesenen Tauon- Neutrinos mit dem durch das SSM errechneten Neutrinofluss von der Sonne aus kommend übereinstimmt. Diese Erkenntnis führte zu einer weiteren Bestätigung des Standard- Sonnen- Modells und, verstärkt durch weitere Experimente, zu der Annahme, dass die „gesandten“ Elektron- Neutrinos durch Oszillation zu Myon- Neutrinos und Tauon- Neutrinos oszillieren. Genauso könnten Myon- Neutrinos zu Tauon- Neutrinos oder Elektron- Neutrinos oder umgekehrt oszillieren. Im Allgemeinen werden diese Vorgänge als Neutrinooszillation bezeichnet. Wie schon erwähnt, geht es in dem OPERA- Experiment um den Nachweis der Neutrinooszillation von Myon- Neutrinos zu Tauon- Neutrinos, indem das Vorkommen eines Tauon- Neutrinos in einem Myon- Neutrinostrahl aufgezeigt wird. Hierzu wurde ein Detektor am LNGS[104] aufgebaut, welches eine Entfernung von 732 km zu dem Teilchenbeschleuniger, welcher einen hochenergetischen Myon- Neutrinostrahl sendet, am CERN[105] hat. Der OPERA- Detektor besteht aus einem Veto[106], welches im Querschnitt den gesamten Detektor überdeckt, um Teilchenreaktionen außerhalb des Detektors ausschließen zu können, da diese für das Experiment nicht von Interesse sind. Der Myonen- Neutrinostrahl trifft als erstes auf das Veto und dringt dann in eines der

[103] Sudbury Neutrino Observatory in der Nähe von Sudbury in Kanada
[104] Laboratori Nazionali del Gran Sasso (ital.): Nationales Versuchslabor am Gran Sasso
[105] Conseil Européen pour la Recherche Nucléaire (frz.) : Europäische Organisation für Kernforschung
[106] Detektorkomponente bestehend aus zwei Lagen Glas- Widerstandsplattenkammern

beiden Targets[107], welche aus so genannten Bricks[108] bestehen, ein. Die Bricks enthalten schichtweise Blei und Fotoemulsion. Trifft ein Tauon- Neutrino auf eine Bleischicht reagiert dieses und es entsteht neben anderen Teilchen auch ein Tauon[109]. Das Tauon wiederum zerfällt nach kurzer Lebensdauer und es kann mit etwa zwanzigprozentiger Wahrscheinlichkeit neben anderen Teilchen ein Myon[110] entstehen. In der genannten Fotoemulsion hinterlässt das Tauon eine sichtbare aber sehr kurze Spur. Da diese sichtbare Spur eine Länge von weniger als einem Millimeter besitzt und die geschichteten Bricks im Querschnitt einer Fläche von über 100.000 Quadratmetern entsprechen, ist das Auffinden solch einer Spur nahezu unmöglich. Aus diesem Grunde werden unter anderem die Spuren der aus den zerfallenen Tauonen entstandenen Myonen verfolgt, um Hinweise auf den Ort in der Fotoemulsion und die Art des Zerfalls des Tauons zu bekommen. Diese Spurverfolgung wird durch die so genannten Target Tracker[111] realisiert, welche aus Plastikszintillatorstreifen bestehen und durch die beim Durchgang geladener Teilchen eine Spur ermittelt werden kann. Zusätzlich müssen die Ladung und der Impuls des aufgezeigten Myons ermittelt werden, um unter anderem Informationen zum Zerfall des Tauons zu erhalten. Bei einem Zerfall eines Tauons wäre ein mögliches Myon negativ geladen. Die Ladung der Myonen kann durch eine Spurablenkung dieser, mittels zweier unterschiedlich gepolter hintereinander stehender Magneten, nachgewiesen werden, da die unterschiedlichen Myonen in unterschiedliche Richtungen von ihrer Spur abgelenkt werden. Diese Myonen- Spur kann durch so genannte Driftröhren rekonstruiert werden. Die im OPERA- Detektor genutzten Driftröhren sind 7,9 Meter lange mit einem Gasgemisch gefüllte Aluminiumröhren, in deren Mitte je ein Draht gespannt ist. Zwischen Röhrenwand und Draht wird eine Spannung angelegt, sodass der Draht als Anode und die Röhrenwand als Kathode fungieren. Durchquert ein elektrisch geladenes Teilchen, wie das Myon, solch eine Röhre, wird an dem gespannten Draht ein Spannungspuls gemessen. Die Driftröhren sind so angeordnet, dass sich durch Zeitangabe des gemessenen Spannungspulses und Angabe der Nummer der insgesamt 9504 Röhren eine Myonen- Spur zurückverfolgen lässt. Durch aufzeigen dieser Spur kann dann Ladung, Ort und Impuls des Myons bestimmt werden [18].

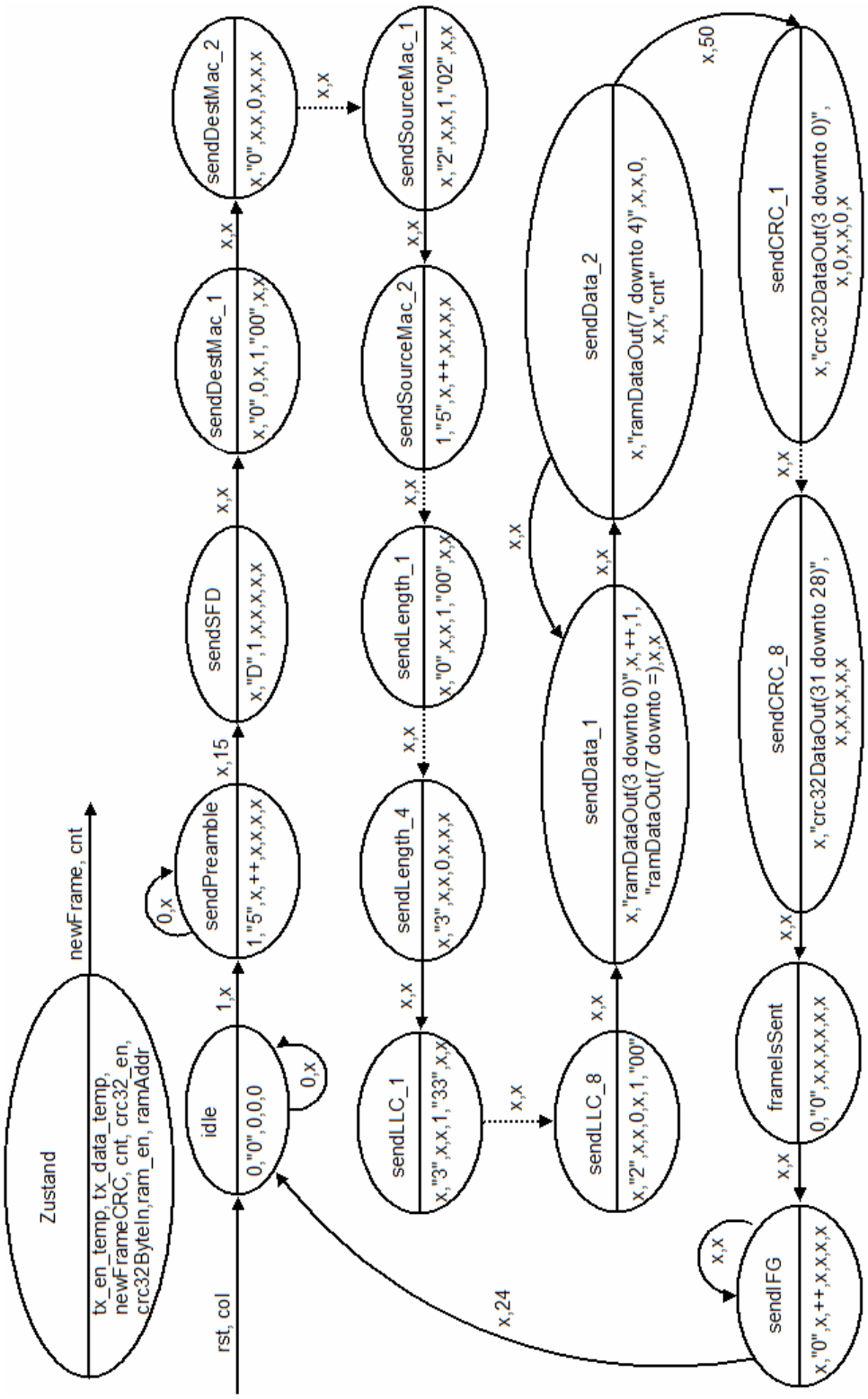
[107] target (engl.): Ziel
[108] brick (engl.): Ziegel
[109] schwerstes Teilchen der Leptonen, welche zur Klasse der Elementarteilchen gehören
[110] Teilchen der Leptonen, welche zur Klasse der Elementarteilchen gehören
[111] target tracker (engl.): Zielfinder



Schematische Darstellung des OPERA- Detektors [18]

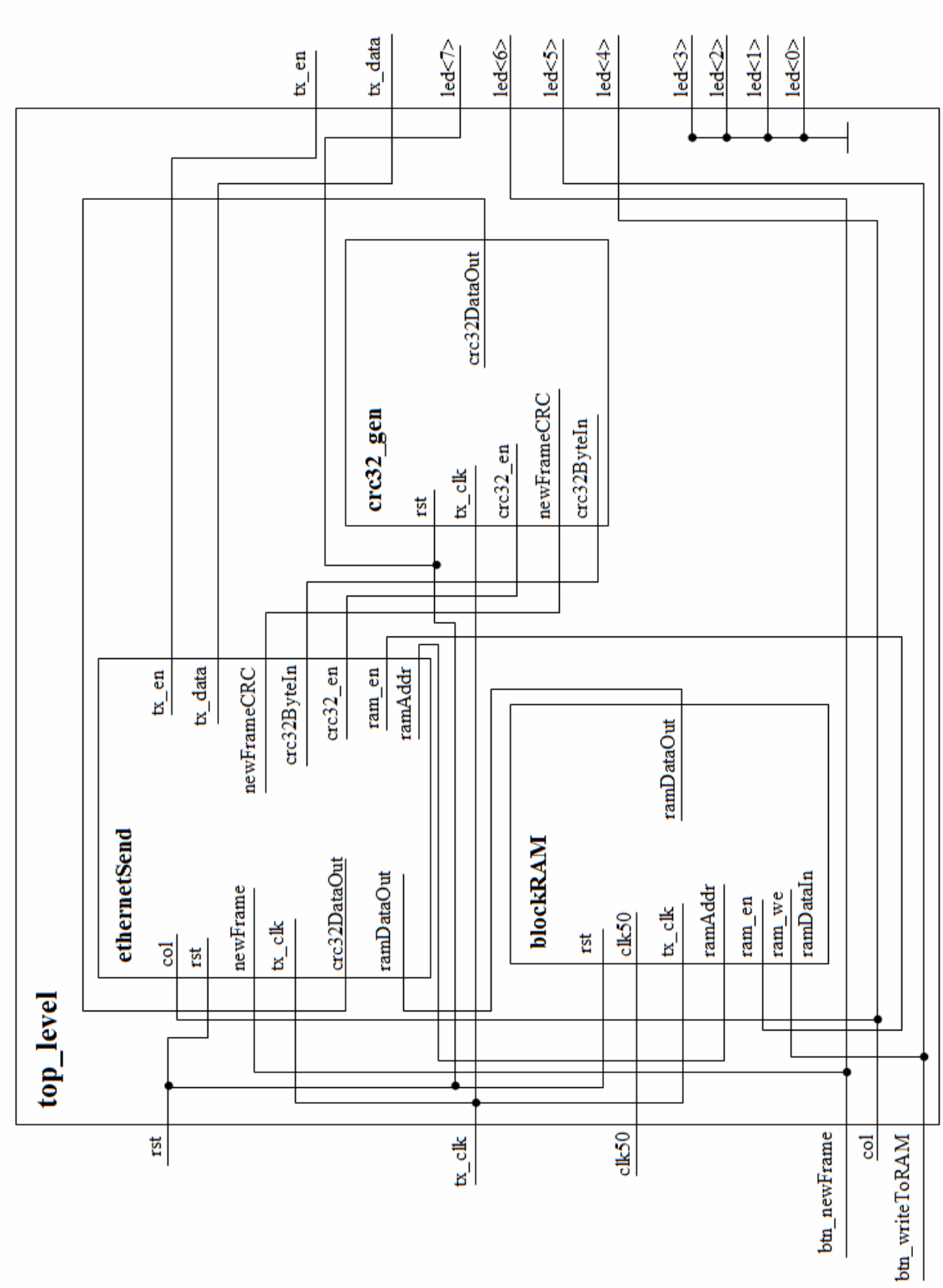
B Zustandsdiagramm

Folgend ist das Zustandsdiagramm zur FSM der Architektur „ethernetSend“ abgebildet. Der Darstellung wegen wurden einige Zustände zusammengefasst, was durch einen punktierten Pfeil zum Folgezustand gekennzeichnet ist.



C Blockdiagramm

Folgende Grafik zeigt das Blockdiagramm zur top_level- Architektur



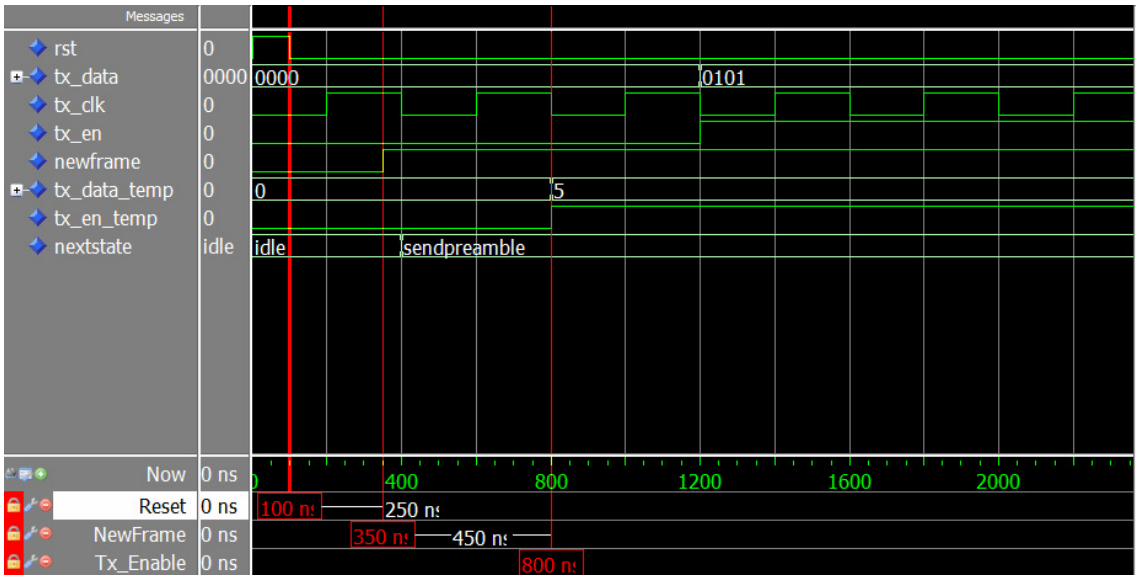
D Testbench

```
------------------------------------------------------------

--Autor: Mareike Stoeppler

--Datum: November 2009

--Datei: top_level_tb.vhd

--Beschreibung: Diese Datei ist die Architektur zur Gene-
rierung eines Taktsignals und zum Setzen der Eingangssig-
nale RESET und newFrame und wird verwendet um den Pro-
grammcode simulieren zu können.

------------------------------------------------------------


--Einfügen der Standardbibliotheken

LIBRARY IEEE  ;

USE IEEE.STD_LOGIC_1164.ALL  ;

USE IEEE.STD_LOGIC_arith.ALL  ;

USE IEEE.STD_LOGIC_unsigned.ALL  ;


ENTITY top_level_tb  IS

END ;


ARCHITECTURE top_level_tb_arch OF top_level_tb IS

--Deklaration aller in der top_level- Architektur vorhan-
den Signale

  SIGNAL btn_newframe   :     BIT;

  SIGNAL led            :     BIT_VECTOR (7 DOWNTO 0);

  SIGNAL rst            :     BIT;

  SIGNAL btn_writetoram :     BIT;

  SIGNAL tx_data        :     BIT_VECTOR (3 DOWNTO 0);

  SIGNAL tx_clk         :     BIT;

  SIGNAL tx_en          :     BIT;

  SIGNAL clk50          :     BIT;

  SIGNAL col            :     BIT;


--Einbinden der Komponente top_level


  COMPONENT top_level

    PORT (

      btn_newframe      :     IN BIT;

      led               :     OUT BIT_VECTOR (7 DOWNTO 0);

      rst               :     IN BIT ;

      btn_writetoram    :     IN BIT ;
```

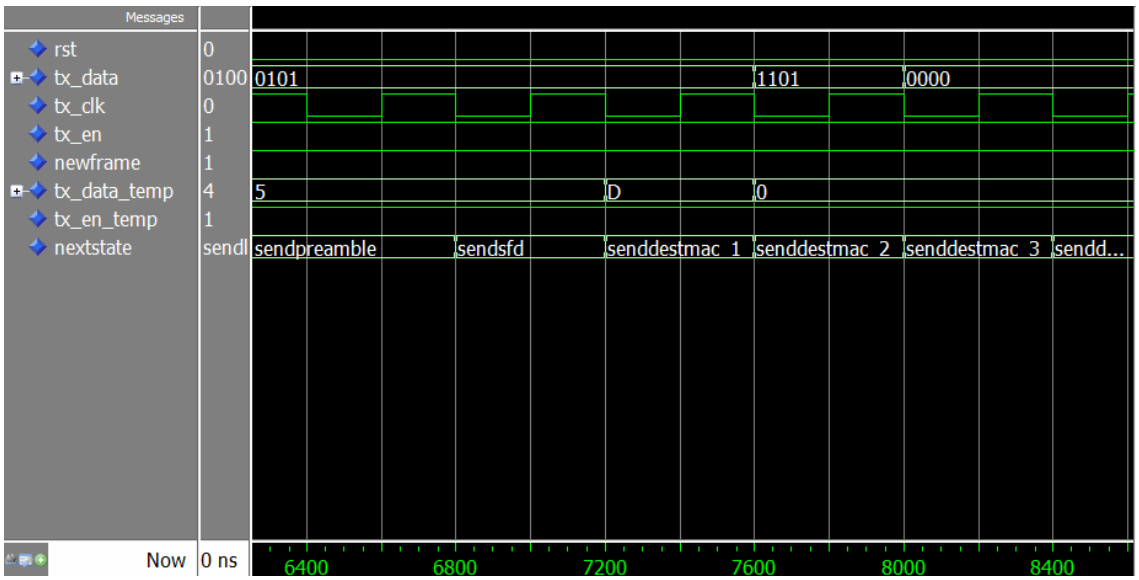
```
      tx_data           :     OUT BIT_VECTOR (3 DOWNTO 0) ;
      col               :     IN BIT;
      tx_clk            :     IN BIT ;
      tx_en             :     OUT BIT ;
      clk50             :     IN BIT
      );
  END COMPONENT ;

BEGIN

--setzen des Taktsignals, hier wird ein Taktsignal mit 2,5
MHz simuliert, das heißt eine Taktperiode hält 400 ns an
  tx_clk <= NOT tx_clk  AFTER 200 ns;
--das Reset- Signal wird nach 300 ns deaktiviert
  rst <= '1', '0' AFTER 300 ns;
--das Senden eine neuen Frames soll nach 350 ns ausgelöst
werden
  btn_newframe <= '0', '1' AFTER 350 ns;

--Das zu testende Modul (DUT: Device under test) ist der
Block top-level und wird hier eingebunden und die zugehö-
rigen Signale werden übergeben
  DUT  : top_level
    PORT MAP (
      btn_newframe   => btn_newframe  ,
      led   => led  ,
      rst   => rst  ,
      btn_writetoram   => btn_writetoram  ,
      tx_data   => tx_data  ,
      col =>  col,
      tx_clk   => tx_clk  ,
      tx_en   => tx_en  ,
      clk50   => clk50   ) ;
END ;
```

E Simulationsergebnis

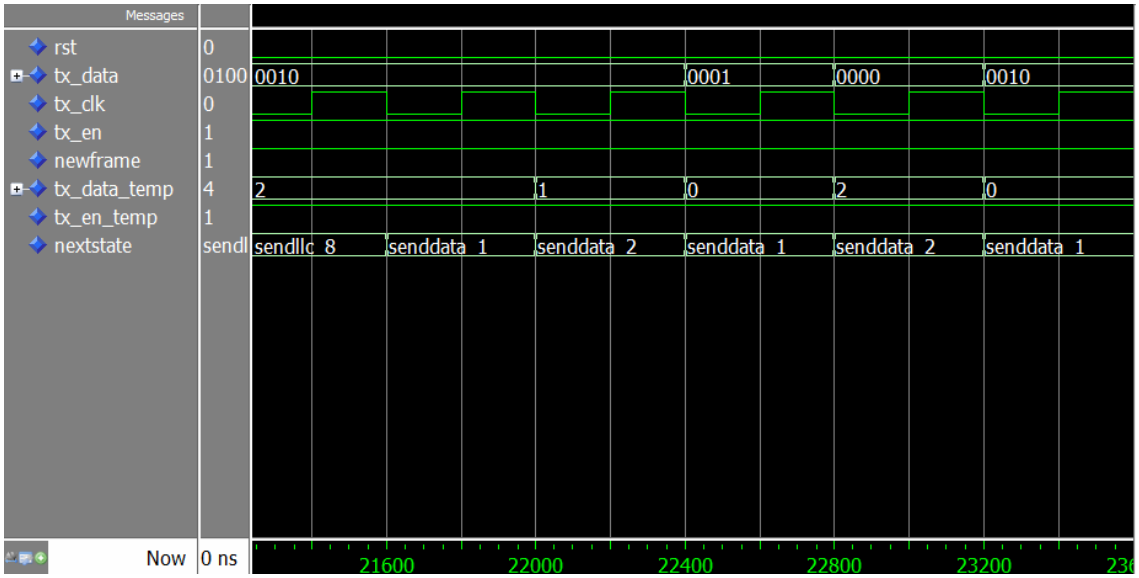
Ausschnitt des Simulationsergebnisses zur Darstellung der Zustände “idle“ und „sendpreamble“ von 0 bis 2280 ns.



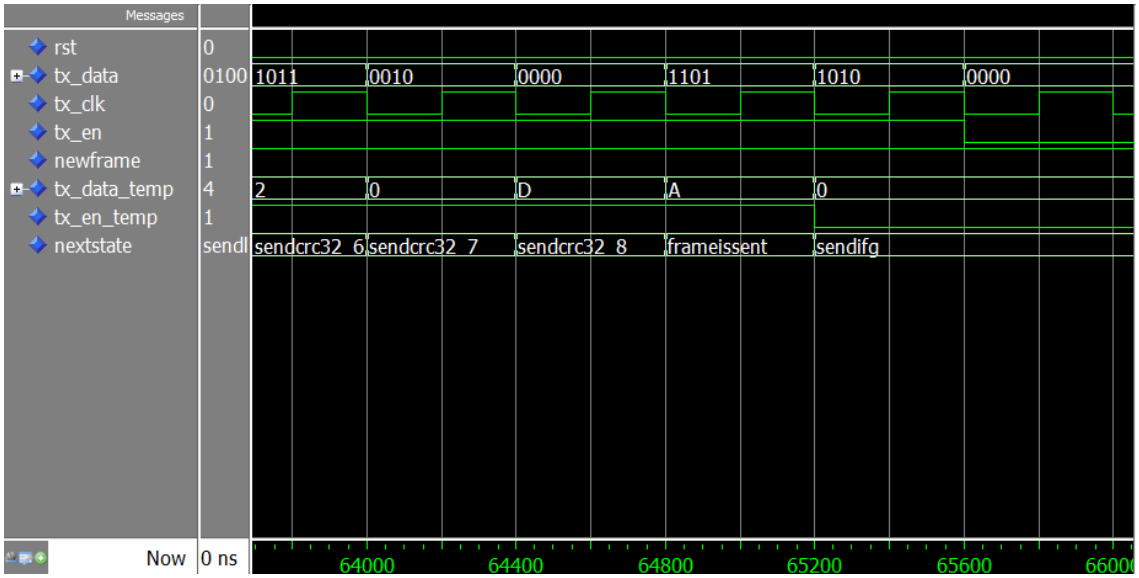
Ausschnitt des Simulationsergebnisses zur Darstellung der Zustände “sfd“ für das Senden des Start Frame Delimiters und „senddestmac“ für die Übertragung der Ziel- MAC-Adresse. Der zeitliche Abschnitt liegt zwischen 6340 und 8480 ns.



Ausschnitt des Simulationsergebnisses zur Darstellung eines Teils der Zustände zum Senden des LLC- Feldes und der ersten Datenbytes.



Ausschnitt des Simulationsergebnisses zur Darstellung der Zustände „sendcrc“, “frameissent” und “sendifg”.



F Quellcodes

F.1 Top_level.vhd

```
----------------------------------------------------------------------
--Autor: Mareike Stoeppler
--Datum: November 2009
--Datei: top_level.vhd
--Beschreibung: Diese Datei ist die Toplevelarchitektur und verknüpft
die untergeordneten Architekturen der Dateien, crc32_gen.vhd, ether-
netSend.vhd und blockRAM.vhd miteinander.
----------------------------------------------------------------------
--einbinden der Standardbibliotheken
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

--Ein- und Ausgangsdefinitionen der Top- Level- Architektur
ENTITY top_level IS
    PORT (
--Definition des Onboard-50MHz-Takteingangs, wird derzeit nur zum Be-
schreiben des RAMS benötigt
           clk50            : IN  BIT;
----------------------------------------------------------------------
--Definition der Aus- und Eingänge zum LAN-Baustein, tx_clk ist der
Taktausgang vom PHY- Baustein, tx_data ist das zu übertragende Nibble
und tx_en gibt dem PHY- Baustein an, dass Daten am Datenbus anliegen.
Das Signal col zeigt eine durch den PHY-Baustein erkannte Kollision
auf dem Übertragungsmedium an.
           tx_clk           : IN   BIT;
           tx_data          : OUT  BIT_VECTOR (3 DOWNTO 0);
           tx_en            : OUT  BIT;
           col              : IN BIT;
----------------------------------------------------------------------
--die beiden btn-Signale sind Eingänge, welche mit je einem Taster
verbunden werden
--btn_newFrame aktiviert die Sendung eines neuen Frames, btn_writeTo
RAM aktiviert den Schreibmodus desBlockRAMS
           btn_newFrame     : IN  BIT;
           btn_writeToRAM   : IN  BIT;
----------------------------------------------------------------------
--der Reset dient zum Neustart des Schaltkreises und wird mit einem
Taster verbunden
           rst              : IN  BIT;
----------------------------------------------------------------------
--der Signalbus led wird mit den vorhandenen 8 LEDs verbunden, um se-
hen zu können, ob die Aktivierung zum Senden eines neuen Frames oder
des RESETs Wirkung haben
           led         : OUT  BIT_VECTOR (7 DOWNTO 0)
            );
END top_level;


architecture Behavioral OF top_level IS

--Einbinden der Komponente ethernetSend dessen Architektur sich in der
Datei ethernetSend.vhd befindet, welche für das Generieren des
EthenetFrames zuständig ist. Zum Einbinden der Komponente müssen hier
alle Ein- und Ausgänge der einzubindenden Architektur definiert werden.
```

```
Ein- und Ausgänge der einzubindenden Architektur definiert werden.
Diese Ports werden in der Datei ethernetSend näher erläutert.

      COMPONENT ethernetSend
      PORT(
      newFrame          : IN  BIT;
            rst         : IN  BIT;
-------------------------------------------------------------------
      tx_clk            : IN  BIT;
      tx_en             : OUT BIT;
      tx_data           : OUT BIT_VECTOR (3 downto 0);
      col : IN BIT;
-------------------------------------------------------------------
      crc32DataOut      : IN  BIT_VECTOR (31 DOWNTO 0);
      crc32ByteIn       : OUT BIT_VECTOR (7 DOWNTO 0);
      crc32_en          : OUT BIT;
      newFrameCRC       : OUT BIT;
-------------------------------------------------------------------
            ramDataOut  : IN  BIT_VECTOR (7 DOWNTO 0);
            ram_en      : OUT BIT;
            ramAddr     : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
      );
      END COMPONENT;

--Einbinden der Komponente crc32_gen, welche zur Prüfsummengenerierung
--benötigt wird und sich in der Datei crc32_gen.vhd befindet
      COMPONENT crc32_gen
      PORT(
            tx_clk            :     IN BIT;
            rst               :     IN BIT;
-------------------------------------------------------------
            crc32_en          :     IN BIT;
            newFrameCRC :     IN BIT;
            crc32ByteIn :     IN BIT_VECTOR (7 DOWNTO 0);
            crc32DataOut:     OUT BIT_VECTOR (31 DOWNTO 0)
            );
      END COMPONENT;

--Einbinden der Komponente blockRAM, welche das Beschreiben und Ausle-
sen des BlockRAMs übernimmt und in der Datei blockRAM.vhd beschrieben
ist
      COMPONENT blockRAM
    PORT (
            rst : IN BIT;
            clk50       : IN  BIT;
            tx_clk      : IN  BIT;
            ramAddr     : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
            ram_en      : IN  BIT;
            ram_we      : IN  BIT;
            ramDataIn   : IN  BIT_VECTOR (7 DOWNTO 0);
            ramDataOut  : OUT  BIT_VECTOR (7 DOWNTO 0)
       );
      END COMPONENT;

--Definition interner Signale, welche benötigt werden, um die Ports
der einzelnen Komponenten miteinander verknüpfen zu können
SIGNAL newFrame         : BIT;
-------------------------------------------------------------------
SIGNAL crc32DataOut     : BIT_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
SIGNAL crc32_en         : BIT;
SIGNAL newFrameCRC      : BIT;
```

```
SIGNAL crc32ByteIn      : BIT_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
-------------------------------------------------------------------
SIGNAL ramAddr                : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL ram_en                 : BIT;
SIGNAL ram_we                 : BIT;
SIGNAL ramDataOut       : BIT_VECTOR (7 DOWNTO 0);
SIGNAL ramDataIn        : BIT_VECTOR (7 DOWNTO 0);
-------------------------------------------------------------------
--internes Signal um eine Entprellung des Taster zur Sendung eines
neuen Frames realisieren zu können
SIGNAL cnt_button: STD_LOGIC_VECTOR (31 DOWNTO 0):= x"00000000";

--Start zur Beschreibung der Top-level-Komponente
BEGIN

--die LEDs 5 und 6 sollen leuchten, sobald der Taster für das Reset-
Signal oder das writeEnable für den RAM aktiviert wurden, außerdem
wird das Signal sofort intern weitergeleitet. Die LEDs 0-4 sind der
Vollständigkeit wegen deklariert, werden hier aber nicht weiter ge-
nutzt und daher auf '0' gesetzt

LED(6) <= rst;
LED(5) <= btn_writeToRAM;
ram_we <= btn_writeToRAM;
LED(4) <= col;
LED(3 DOWNTO 0) <= "0000";


--newFrame <= btn_NewFrame;-- wird nur bei Simulation benötigt

--Taktgesteuerter Prozess zum Entprellen des Tasters für das Senden
eines neuen Frames

debounceButton: PROCESS(tx_clk)
BEGIN
----Bei steigender Taktflanke und gedrücktem Taster wird ein Zähler
inkrementiert. Erst bei einer Übersteigung des angegebenen Integerwer-
tes wird eine LED und das Signal zum Senden eines neuen Ethernetframes
auf '1' und nach einem Taktzyklus wieder auf Low gesetzt. Ist der In-
tegerwert nicht überschritten bleiben die Signale auf Low

      IF tx_clk'EVENT AND tx_clk ='1' then
            IF btn_newFrame ='1' THEN
                  cnt_button<= cnt_button+1;
            END IF;
--CONV_INTEGER dient der Konvertierung eines bit_vectors in einen In-
tegerwert
            IF (CONV_INTEGER(cnt_button)> 4967000) THEN
                  LED(7)<= '1';
                  newFrame <= '1';
                  cnt_button <= (OTHERS => '0');
            ELSE
                  LED(7) <= '0';
                  newFrame <='0';
            END IF;--Ende der if-Anweisung
      END IF;
END PROCESS debounceButton;--ende des taktgesteuerten Prozesses

--Verknüpfung der Signalein- und ausgänge der einzelnen Komponenten.
Die links stehenden Signale sind die in der jeweiligen Architektur
festgelegten Ein- und Ausgänge. Die rechts stehenden Signale sind die,
```

```
welche übergeben werden. In dieser Arbeit sind der Übersicht wegen die
Signalnamen in jeder Komponente gleich bleibend, so kann in einem
Blockschaltbild der Weg eines Signals leichter nachvollzogen werden

      init_crc32_gen: crc32_gen PORT MAP(
            tx_clk                  => tx_clk,
            rst                     => rst,
-----------------------------------------------------------------------
            crc32_en                => crc32_en,
            newFrameCRC       => newFrameCRC,
            crc32ByteIn       => crc32ByteIn,
            crc32DataOut      => crc32DataOut
      );

      init_ethernetSend: ethernetSend PORT MAP(
      newFrame          => newFrame,
            rst         => rst,
-----------------------------------------------------------------------
            tx_clk      => tx_clk,
            tx_en       => tx_en,
            tx_data     => tx_data,
            col         =>  col,
-----------------------------------------------------------------------
            crc32DataOut      => crc32DataOut,
            crc32ByteIn       => crc32ByteIn,
            crc32_en          => crc32_en,
            newFrameCRC       => newFrameCRC,
-----------------------------------------------------------------------
            ramDataOut        => ramDataOut,
            ram_en                  => ram_en,
            ramAddr                 => ramAddr
      );

      init_blockRAM: blockRAM PORT MAP(

            rst         =>  rst,
            clk50       => clk50,
            tx_clk      => tx_clk,
            ramAddr     => ramAddr,
            ram_en      => ram_en,
            ram_we      => ram_we,
            ramDataIn   => ramDataIn,
            ramDataOut  => ramDataOut
      );

END Behavioral;--Ende der Architekturbeschreibung
```

F.2 ethernetSend.vhd

```
----------------------------------------------------------------------
--Autor: Mareike Stoeppler
--Datum: November 2009
--Datei: ethernetSend.vhd
--Beschreibung: Diese Datei ist die Architektur zum Generieren und
Weiterleiten eines Ethernetframes
----------------------------------------------------------------------
--Einfügen der Standardbibliotheken
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

--Definition der Ein- und Ausgangssignale des ethernetSend-Blocks
ENTITY ethernetSend IS
   PORT(
--Eingangssignal zum Generieren eines neuen Frames
            newFrame    : IN  BIT;
            rst         : IN  BIT;  --Reset-Signal
----------------------------------------------------------------------
            tx_clk      : IN  BIT;  --Taktsignal des PHY-Bausteins
--Ausgangssignal zum Ankündigen neuer Daten
            tx_en       : OUT BIT;
--Datenbus welcher ein Nibble an den PHY-Baustein sendet
            tx_data     : OUT BIT_VECTOR (3 downto 0);
--Eingangssignal zur Anzeige einer Kollisionserkennung
            col : IN BIT;
----------------------------------------------------------------------
--Signaleingangsbus, enthält die im crc32_gen-Block generierte Prüf-
summe
            crc32DataOut      : IN  BIT_VECTOR (31 DOWNTO 0);
--Signalausgangsbus, sendet ein Byte an den crc32_gen-Block
            crc32ByteIn       : OUT BIT_VECTOR (7 DOWNTO 0);
--Signalausgang aktiviert die Prüfsummengenerierung
            crc32_en          : OUT BIT;
--Signalausgang aktiviert die Initialisierung des crc32_gen-Blocks
            newFrameCRC       : OUT BIT;
----------------------------------------------------------------------
--Signaleingangsbus enthält die zu sendenden Daten aus dem BlockRAM
            ramDataOut        : IN  BIT_VECTOR (7 DOWNTO 0);
--aktiviert den BlockRAM-Block
            ram_en            : OUT BIT;
--sendet die Adresse der im BlockRAM gespeicherten Daten
            ramAddr                 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
                );
END ethernetSend ;


ARCHITECTURE ethernetSend_arch OF ethernetSend IS

      --Definition der konstanten Ziel- und Quell-Mac-Adresse
      constant destMac        : BIT_VECTOR( 47 DOWNTO 0 ) :=
x"00a0d166a7e8";
   constant sourceMac  : BIT_VECTOR( 47 DOWNTO 0 ) :=  x"02608c010203";

      --Zählersignal von 255 bis 0
      SIGNAL cnt                    : STD_LOGIC_VECTOR (7 DOWNTO 0 );
      --Temporäre Signale
      SIGNAL tx_data_temp     : BIT_VECTOR(3 downto 0) :=x"0";
```

```
      SIGNAL tx_en_temp       : BIT := '0';

      --Definition der Zustände für den Zustandsautomaten
   TYPE STATETYPE IS (
      idle,
      sendPreamble,
      sendSFD,
      sendDestMac_1,
      sendDestMac_2,
      sendDestMac_3,
      sendDestMac_4,
      sendDestMac_5,
      sendDestMac_6,
      sendDestMac_7,
      sendDestMac_8,
      sendDestMac_9,
      sendDestMac_10,
      sendDestMac_11,
      sendDestMac_12,
      sendSourceMac_1,
      sendSourceMac_2,
      sendSourceMac_3,
      sendSourceMac_4,
      sendSourceMac_5,
      sendSourceMac_6,
      sendSourceMac_7,
      sendSourceMac_8,
      sendSourceMac_9,
      sendSourceMac_10,
      sendSourceMac_11,
      sendSourceMac_12,
      sendLength_1,
      sendLength_2,
      sendLength_3,
      sendLength_4,
      sendLLC_1,
      sendLLC_2,
      sendLLC_3,
      sendLLC_4,
      sendLLC_5,
      sendLLC_6,
      sendLLC_7,
      sendLLC_8,
      sendData_1,
      sendData_2,
      sendCRC32_1,
      sendCRC32_2,
      sendCRC32_3,
      sendCRC32_4,
      sendCRC32_5,
      sendCRC32_6,
      sendCRC32_7,
      sendCRC32_8,
      frameIsSent,
      sendIFG
   );

--internes Signal um im nächsten Taktzyklus in den hier gespeicherten
Zustand zu springen
SIGNAL nextState : STATETYPE := idle;
```

```
BEGIN

--Taktgesteuerter Prozess, welcher durch das Taktsignal des PHY-
Bausteins und das Reset-Signal aktiviert wird.
clockedProcess : PROCESS (tx_clk, rst, col)

   BEGIN
      IF (rst = '1') OR (col='1') THEN --bei Aktivierung des Reset-
Signals in den Idle-Zustand gehen
         nextState <= idle;
--Bei steigender Taktflanke werden die temporären Signale an die ent-
sprechenden Ausgangssignale gegeben.
      ELSIF (tx_clk'EVENT AND tx_clk = '0') THEN
                  tx_data <= tx_data_temp;
                  tx_en <= tx_en_temp;
--Zustandsautomat wird durch CASE-Anweisung realisiert. Durch die Ab-
frage des Signals nextState wird in den aktuellen Zustand gesprungen.
         CASE nextState IS
--Im idle-Zustand werden die default-Werte gesetzt und bei Aktivierung
des newFrame-Signals ist sendPreamble der Folgezustand
            WHEN idle =>

               cnt <= "00000000";
               crc32_en <= '0';
                              tx_en_temp <= '0';
               newFrameCRC <= '0';
               tx_data_temp <= (others => '0');

                              IF newFrame = '1' THEN
                                    nextState <= sendPreamble;

               ELSE
                  nextState <= idle;
               END IF;
--Im Zustand SendPreamble wird der PHY-Baustein durch das enable-
Signal aktiviert und 15 Mal die Signalfolge "1010" gesendet, welche
mit dem LSB zuerst gesandt in hexadezimaler schreibweise einer '5'
entspricht. Nach dem Senden wird der Zähler zurückgesetzt und der
sendSFD ist der Folgezustand
                        WHEN sendPreamble =>

                              tx_en_temp <='1';
                              IF (conv_integer(cnt) < 15) THEN
                                    tx_data_temp <= x"5";
                                    nextState <= sendPreamble;
                                    cnt <= cnt + 1;
                              ELSE
                                    nextState <= sendSFD;
                                    cnt <= "00000000";
                              END IF;
--Im Zustand sendSFD wird das zweite Nibble des StartFrameDelimiter
'D' gesendet und der crc32_gen-Blocks aktiviert. Der Folgezustand lau-
tet sendDestMac

                        WHEN sendSFD =>

                              tx_data_temp <= x"D";
                              nextState <= sendDestMac_1;
                              newFrameCRC <= '1';
```

```
--in den Zuständen sendDestMac_1, 3, 5, 7, 9 und 11 werden die Gene-
rierung der Prüfsumme aktiviert und die entsprechenden Bytes der Ziel-
Mac-Adresse an den crc32_gen-Block übergeben.
--Außerdem werden die niederwertigen Nibble an den PHY - Baustein ge-
sandt

            WHEN sendDestMac_1 =>

                  newFrameCRC <= '0';
                  crc32_en <= '1';
                  tx_data_temp <= destMac(43 downto 40);
                  crc32ByteIn <= destMac(47 downto 44) & destMac(43
downto 40);
                  nextState <= sendDestMac_2;
--in den Zuständen sendDestMac_2, 4, 6, 8, 10  und 12 wird die Gene-
rierung der Prüfsumme gestoppt und das jeweils höherwertige Nibble der
entsprechenden Ziel-MAC-Adresse an den PHY-Baustein gesandt

            WHEN sendDestMac_2 =>

                  crc32_en <= '0';
                  tx_data_temp <= destMac(47 downto 44);
                  nextState <= sendDestMac_3;

            WHEN sendDestMac_3 =>

                  crc32_en <= '1';
                  tx_data_temp <= destMac(35 downto 32);
                  crc32ByteIn <= destMac(39 downto 36) & destMac(35
downto 32);
                  nextState <= sendDestMac_4;

            WHEN sendDestMac_4 =>

                  crc32_en <= '0';
                  tx_data_temp <= destMac(39 downto 36);
                  nextState <= sendDestMac_5;

            WHEN sendDestMac_5 =>

                  crc32_en <= '1';
                  tx_data_temp <= destMac(27 downto 24);
                  crc32ByteIn <= destMac(31 downto 28) & destMac(27
downto 24);
                  nextState <= sendDestMac_6;

            WHEN sendDestMac_6 =>

                  crc32_en <= '0';
                  tx_data_temp <= destMac(31 downto 28);
                  nextState <= sendDestMac_7;

            WHEN sendDestMac_7 =>

                  crc32_en <= '1';
                  tx_data_temp <= destMac(19 downto 16);
                  crc32ByteIn <= destMac(23 downto 20) & destMac(19
downto 16);
                  nextState <= sendDestMac_8;

            WHEN sendDestMac_8 =>
```

```
                crc32_en <= '0';
                tx_data_temp <= destMac(23 downto 20);
                nextState <= sendDestMac_9;

            WHEN sendDestMac_9 =>

                crc32_en <= '1';
                tx_data_temp <= destMac(11 downto 8);
                crc32ByteIn <= destMac(15 downto 12) & destMac(11
downto 8);
                nextState <= sendDestMac_10;

            WHEN sendDestMac_10 =>

                crc32_en <= '0';
                tx_data_temp <= destMac(15 downto 12);
                nextState <= sendDestMac_11;

            WHEN sendDestMac_11 =>

                crc32_en <= '1';
                tx_data_temp <= destMac(3 downto 0);
                crc32ByteIn <= destMac(7 downto 4) & destMac(3 downto
0);
                nextState <= sendDestMac_12;

            WHEN sendDestMac_12 =>

                crc32_en <= '0';
                tx_data_temp <= destMac(7 downto 4);
                nextState <= sendSourceMac_1;

--In den Zuständen sendSourceMac_1 bis 12 werden die Bits der Quell-
Mac-Adresse zur Generierung an den crc32_gen-Block und an den PHY-
Baustein gesandt.

            WHEN sendSourceMac_1 =>

                crc32_en <= '1';
                tx_data_temp <= sourceMac(43 downto 40);
                crc32ByteIn <= sourceMac(47 downto 44) & sourceMac(43
downto 40);
                nextState <= sendSourceMac_2;

            WHEN sendSourceMac_2 =>

                crc32_en <= '0';
                tx_data_temp <= sourceMac( 47 downto 44);
                nextState <= sendSourceMac_3;

            WHEN sendSourceMac_3 =>

                crc32_en <= '1';
                tx_data_temp <= sourceMac(35 downto 32);
                crc32ByteIn <= sourceMac(39 downto 36) & sourceMac(35
downto 32);
                nextState <= sendSourceMac_4;

            WHEN sendSourceMac_4 =>
                crc32_en <= '0';
```

```
                tx_data_temp <= sourceMac(39 downto 36);
                nextState <= sendSourceMac_5;

            WHEN sendSourceMac_5 =>

                crc32_en <= '1';
                tx_data_temp <= sourceMac(27 downto 24);
                crc32ByteIn <= sourceMac(31 downto 28) & sourceMac(27
downto 24);
                nextState <= sendSourceMac_6;

            WHEN sendSourceMac_6 =>

                crc32_en <= '0';
                tx_data_temp <= sourceMac(31 downto 28);
                nextState <= sendSourceMac_7;

            WHEN sendSourceMac_7 =>

                crc32_en <= '1';
                tx_data_temp <= sourceMac(19 downto 16);
                crc32ByteIn <= sourceMac(23 downto 20) & sourceMac(19
downto 16);
                nextState <= sendSourceMac_8;

            WHEN sendSourceMac_8 =>

                crc32_en <= '0';
                tx_data_temp <= sourceMac(23 downto 20);
                nextState <= sendSourceMac_9;

            WHEN sendSourceMac_9 =>

                crc32_en <= '1';
                tx_data_temp <= sourceMac(11 downto 8);
                crc32ByteIn <= sourceMac(15 downto 12) & sourceMac(11
downto 8);
                nextState <= sendSourceMac_10;

            WHEN sendSourceMac_10 =>

                crc32_en <= '0';
                tx_data_temp <= sourceMac(15 downto 12);
                nextState <= sendSourceMac_11;

            WHEN sendSourceMac_11 =>

                crc32_en <= '1';
                tx_data_temp <= sourceMac(3 downto 0);
                crc32ByteIn <= sourceMac(7 downto 4) & sourceMac(3
downto 0);
                nextState <= sendSourceMac_12;

            WHEN sendSourceMac_12 =>

                crc32_en <= '0';
                tx_data_temp <= sourceMac(7 downto 4);
                nextState <= sendLength_1;

--Die Zustände sendLength_1 bis 4 senden 2 Byte, in welchen die Anzahl
der Datenbytes inklusive der 4 Bytes für das LLC-Feld enthalten sind.
```

```
            WHEN sendLength_1 =>

                  crc32_en <= '1';
                  tx_data_temp <= "0000";
                  crc32ByteIn <= "0000" & "0000";
                  nextState <= sendLength_2;

            WHEN sendLength_2 =>

                  crc32_en <= '0';
                  tx_data_temp <= "0000";
                  nextState <= sendLength_3;

            WHEN sendLength_3 =>

                  crc32_en <= '1';
                  tx_data_temp <= "0010";
                  crc32ByteIn <= "0011" & "0010";
                  nextState <= sendLength_4;

            WHEN sendLength_4 =>

                  crc32_en <= '0';
                  tx_data_temp <= "0011";
                  nextState <= SendLLC_1;
--die Zustände SendLLC_1 bis 8 senden 4 Bytes, welche dem LLC- Feld
entsprechen

            WHEN SendLLC_1 =>

                  crc32_en <= '1';
                  tx_data_temp <= "0011";
                  crc32ByteIn <= "0011" & "0011";
                  nextState <= SendLLC_2;

            WHEN SendLLC_2 =>

                  crc32_en <= '0';
                  tx_data_temp <= "0011";
                  nextState <= SendLLC_3;

            WHEN SendLLC_3 =>

                  crc32_en <= '1';
                  tx_data_temp <= "0100";
                  crc32ByteIn <= "0100" & "0100";
                  nextState <= SendLLC_4;

            WHEN SendLLC_4 =>

                  crc32_en <= '0';
                  tx_data_temp <= "0100";
                  nextState <= SendLLC_5;

            WHEN SendLLC_5 =>

                  crc32_en <= '1';
                  tx_data_temp <= "0010";
                  crc32ByteIn <= "0010" & "0010";
                  nextState <= SendLLC_6;
```

```
            WHEN SendLLC_6 =>

                  crc32_en <= '0';
                  tx_data_temp <= "0010";
                  nextState <= SendLLC_7;


            WHEN SendLLC_7 =>

                  crc32_en <= '1';
                  tx_data_temp <= "0010";
                  crc32ByteIn <= "0010" & "0010";
                  nextState <= SendLLC_8;
--Im Zustand SendLLC_8 wird außerdem der BlockRAM aktiviert und die
erste Adresse des RAMs übergeben unter der die zu sendenden Daten ge-
speichert sind, hier ist die Startadresse 0 angegeben

            WHEN SendLLC_8 =>

                  crc32_en <= '0';
                  tx_data_temp <= "0010";
                  ram_en <= '1';
                  ramAddr<= x"00";--load first RAM address
                  nextState <= sendData_1;

--der Zustand sendData_1 liest das erste Datenbyte aus dem RAM und
gibt dieses weiter an den crc32_gen-Block, außerdem wird das nieder-
wertige Nibble des 1. Datenbytes an den PHY- Baustein übergeben und
der Zähler inkrementiert
            WHEN sendData_1 =>

                  tx_data_temp <= ramDataOut(3 DOWNTO 0);
                  cnt <= cnt + 1;
                  crc32_en <= '1';
                  crc32ByteIn <= ramDataOut(7 DOWNTO 4) & ramDataOut(3
DOWNTO 0);
                  nextState <= sendData_2;

--Der Zustand sendData_2 sendet das höherwertige Nibble an den PHY-
Baustein, übergibt die Adresse unter der das nächste Datenbyte gespei-
chert ist an den BlockRAM und überprüft anhand einer Zählerabfrage ob
die Anzahl der zu sendenden Datenbytes erreicht ist. Ist dieser Zäh-
lerstand erreicht, wird der nächste Zustand zum Senden der Prüfsumme
aktiviert und der BlockRAM deaktiviert. Die Anzahl der Datenbytes ist
hier auf 50 festgelegt.
            WHEN sendData_2 =>

                  crc32_en <= '0';
                  tx_data_temp <= ramDataOut(7 DOWNTO 4);
                  IF ( CONV_INTEGER(cnt) =  50) THEN
                        cnt <= (OTHERS => '0');
                        ram_en<='0';
                        nextState <= sendCRC32_1;
                  ELSE
                        ramAddr <= cnt;
                        nextState <= sendData_1;
                  END IF;
--Die Zustände sendCRC32_1 bis 8 sind zuständig für das Senden der
Prüfsumme.
            WHEN sendCRC32_1 =>
```

```
                tx_data_temp <= crc32DataOut(3 downto 0);
                nextState <= sendCRC32_2;

            WHEN sendCRC32_2 =>

                tx_data_temp <= crc32DataOut(7 downto 4);
                nextState <= sendCRC32_3;

            WHEN sendCRC32_3 =>

                tx_data_temp <= crc32DataOut(11 downto 8);
                nextState <= sendCRC32_4;

            WHEN sendCRC32_4 =>

                tx_data_temp <= crc32DataOut(15 downto 12);
                nextState <= sendCRC32_5;

            WHEN sendCRC32_5 =>

                tx_data_temp <= crc32DataOut(19 downto 16);
                nextState <= sendCRC32_6;

            WHEN sendCRC32_6 =>

                tx_data_temp <= crc32DataOut(23 downto 20);
                nextState <= sendCRC32_7;

            WHEN sendCRC32_7 =>

                tx_data_temp <= crc32DataOut(27 downto 24);
                nextState <= sendCRC32_8;

            WHEN sendCRC32_8 =>

                tx_data_temp <= crc32DataOut(31 downto 28);
                nextState <= frameIsSent;

--Nach dem Senden der Prüfsumme, wird das Enable für den PHY- Baustein
deaktiviert und der Datenbus auf Low gesetzt.
            WHEN frameIsSent =>
                tx_en_temp <= '0';
                tx_data_temp <= x"0";
                nextState <= sendIFG;
--Zwischen dem Senden zweier Frames muss das Senden für eine Zeitspan-
ne von 0,96µs unterbrochen werden, damit der Empfänger die empfangenen
Daten verarbeiten kann. 24 Takte des Eingangstaktsignals tx_clk ent-
sprechen dieser Zeitspanne, was durch das inkrementieren und Abfragen
eines Zählers realisiert wird.
            WHEN sendIFG =>
                IF (CONV_INTEGER(cnt) <24) THEN
                    tx_data_temp <= x"0";
                    nextState <= sendIFG;
                    cnt <= cnt + 1;
                ELSE
                    nextState <= idle;
                END IF;
--Falls durch einen Übertragungsfehler ein Zustand gespeichert ist,
der hier nicht angegeben ist, kann durch diese Zustandsabfrage der
Fehler abgefangen werden und es wird automatisch der Idle- Zustand ge-
```

```
setzt, in welchem den Signalen die jeweiligen default-Werte zugewiesen
werden
            WHEN OTHERS =>

                  nextState <= idle;

         END CASE;--ende der Zustandsabfrage
      END IF;
   END PROCESS clockedProcess;
END ethernetSend_arch;
```

F.3 crc32_gen.vhd

```
--------------------------------------------------------------------------
--Autor: Mareike Stoeppler
--Datum: Dezember 2009
--Datei: crc32_gen.vhd
--Beschreibung: Diese Datei enthält die Architektur zur Generierung
der Prüfsumme. Die zu verschlüsselnden Daten werden byteweise von der
ethernetSend- Komponente gesandt und über ein Schieberegister ver-
schlüsselt. Nach dem Verschlüsseln des letzten Datenbytes wird die
Prüfsumme an ethernetSend übergeben und von dort aus an den PHY-
Baustein gesandt.
--------------------------------------------------------------------------
--einbinden der Standardbibliotheken
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity crc32_gen is
      PORT(
            tx_clk      :     IN bit;--Taktsignal des PHY- Bausteins
            rst         :     IN bit;--Reset
--------------------------------------------------------------------------
--enable des Prüfsummengenerators, welches angibt dass ein neues Byte
am Eingang bereit liegt
            crc32_en    :     IN bit;
--enable des Prüfsummengenerators, dass eine neue Prüfsumme berechnet
werden muss
            newFrameCRC :     IN bit;
--Byte breiter Eingangsvektor zum Einlesen der neu zu verschlüsselnden
Daten
            crc32ByteIn :     IN bit_vector (7 DOWNTO 0);
-- Vier Byte breiter Ausgangsvektor zur Ausgabe der Prüfsumme
            crc32DataOut:     OUT bit_vector (31 DOWNTO 0)
            );
end crc32_gen;

architecture crc32_gen_arch of crc32_gen is
      signal crc32DataInt     : bit_vector (31 downto 0);
      signal crc32DataTemp    : bit_vector(31 downto 0);
begin


--Beginn des taktgesteuerten Prozesses
takt: process (TX_CLK)
--Deklaration der Variablen, welche innerhalb des Prozesses aktuali-
siert werden
      variable crc32LastBack: bit;
      variable crc32DataTemp_var: bit_vector (31 downto 0);
begin
--Bei aktivem Reset werden die Signale auf ihren default- Wert gesetzt
      if rst = '1' then
            crc32DataOut      <= (others => '0');
            crc32DataInt      <= (others => '0');
            crc32DataTemp     <= (others => '1');
      elsif (TX_CLK'event and TX_CLK = '1') then
--Bei inaktivem Reset und Aktivierung eines neuen Frames werden die
Signale auf ihren default- Wert gesetzt
            if newFrameCRC = '1' then
                  crc32DataOut      <= (others => '0');
                  crc32DataInt      <= (others => '0');
                  crc32DataTemp     <= (others => '1');
```

```
            else
--Bei steigender Taktflanke und keiner Sendung eines neuen Frames,
wird das temporäre Signal negiert an den entsprechenden Ausgang gelegt
                  crc32DataInt <= (not crc32DataTemp);
                  crc32DataOut <= (not crc32DataTemp);
            end if;
--Bei Aktivierung des Prüfsummengenerators wird der intern gespeicher-
te Signalvektor negiert an die temporäre Variable übergeben und die
Schleife durchlaufen
            if crc32_en = '1' then
                  crc32DataTemp_var := not crc32DataInt;
                  for I in 0 to 7 loop
--Jedes Eingangsbit wird mit dem niederwertigem Bit des Schieberegis-
ters XOR verknüpft und in der Variable crc32LastBack gespeichert. Die-
ses wird wiederum an den durch das crc32-Polynom vorgesehenen Stellen
des Schieberegisters mit XOR verknüpft.
                        crc32LastBack := (crc32DataTemp_var(0) xor
crc32ByteIn(I));
                        crc32DataTemp_var(0)       :=
crc32DataTemp_var(1);
                        crc32DataTemp_var(1)       :=
crc32DataTemp_var(2);
                        crc32DataTemp_var(2)       :=
crc32DataTemp_var(3);
                        crc32DataTemp_var(3)       :=
crc32DataTemp_var(4);
                        crc32DataTemp_var(4)       :=
crc32DataTemp_var(5);
                        crc32DataTemp_var(5)       :=
(crc32DataTemp_var(6) xor crc32LastBack);
                        crc32DataTemp_var(6)       :=
crc32DataTemp_var(7);
                        crc32DataTemp_var(7)       :=
crc32DataTemp_var(8);
                        crc32DataTemp_var(8)       :=
(crc32DataTemp_var(9) xor crc32LastBack);
                        crc32DataTemp_var(9)       :=
(crc32DataTemp_var(10) xor crc32LastBack);
                        crc32DataTemp_var(10)      :=
crc32DataTemp_var(11);
                        crc32DataTemp_var(11)      :=
crc32DataTemp_var(12);
                        crc32DataTemp_var(12)      :=
crc32DataTemp_var(13);
                        crc32DataTemp_var(13)      :=
crc32DataTemp_var(14);
                        crc32DataTemp_var(14)      :=
crc32DataTemp_var(15);
                        crc32DataTemp_var(15)      :=
(crc32DataTemp_var(16) xor crc32LastBack);
                        crc32DataTemp_var(16)      :=
crc32DataTemp_var(17);
                        crc32DataTemp_var(17)      :=
crc32DataTemp_var(18);
                        crc32DataTemp_var(18)      :=
crc32DataTemp_var(19);
                        crc32DataTemp_var(19)      :=
(crc32DataTemp_var(20) xor crc32LastBack);
                        crc32DataTemp_var(20)      :=
(crc32DataTemp_var(21) xor crc32LastBack);
```

```
                        crc32DataTemp_var(21)      :=
(crc32DataTemp_var(22) xor crc32LastBack);
                        crc32DataTemp_var(22)      :=
crc32DataTemp_var(23);
                        crc32DataTemp_var(23)      :=
(crc32DataTemp_var(24) xor crc32LastBack);
                        crc32DataTemp_var(24)      :=
(crc32DataTemp_var(25) xor crc32LastBack);
                        crc32DataTemp_var(25)      :=
crc32DataTemp_var(26);
                        crc32DataTemp_var(26)      :=
(crc32DataTemp_var(27) xor crc32LastBack);
                        crc32DataTemp_var(27)      :=
(crc32DataTemp_var(28) xor crc32LastBack);
                        crc32DataTemp_var(28)      :=
crc32DataTemp_var(29);
                        crc32DataTemp_var(29)      :=
(crc32DataTemp_var(30) xor crc32LastBack);
                        crc32DataTemp_var(30)      :=
(crc32DataTemp_var(31) xor crc32LastBack);
                        crc32DataTemp_var(31)      := crc32LastBack;
                end loop;
--Nach Schleifenende muss der 32-Bit-Vektor der Variable an den tempo-
rären 32-Bit-Vektor übergeben werden, um die Bits für den nächsten
Schleifendurchlauf zwischenspeichern zu können.
                crc32DataTemp <= crc32DataTemp_var;
            end if;
      end if;
end process takt;-- Ende des taktgesteuerten Prozesses

end crc32_gen_arch;
```

F.4 blockRAM.vhd

```
----------------------------------------------------------------------
--Autor: Mareike Stoeppler
--Datum: Dezember 2009
--Datei: blockRAM.vhd
--Beschreibung: Diese Datei enthält die Architektur des BlockRAMs und
steuert das Auslesen und Beschreiben der enthaltenen Speicherzellen

----------------------------------------------------------------------

--Einbinden der Standardbibliotheken
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

--Definition der Ein- und Ausgangssignale
ENTITY blockRAM IS
    PORT (
--Das Reset- Signal um den gesamten Block zurück zusetzen
           rst         : IN  BIT;
--Das 50MHz Taktsignal dient nur zum Schreiben in das RAM, welches im
derzeitigen Projekt nicht genutzt wird, da keine Daten von außen zur
Verfügung stehen
           clk50       : IN  BIT;
--Das Taktsignal tx_clk kommt vom PHY-Baustein und wird für das takt-
gesteuerte Auslesen des RAMS benötig
           tx_clk      : IN  BIT;
----------------------------------------------------------------------
--Es stehen 2^8 Speicherzellen zur Verfügung, um diese direkt anspre-
chen zu können, wird eine Adresse durch den bytelangen Vektor angege-
ben
           ramAddr     : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
--um wahlloses Schreiben und Lesen des RAMS zu verhindern, gibt es ein
enable und ein writeEnable Signal
           ram_en      : IN  BIT;
           ram_we      : IN  BIT;
--ramDataIn stellt die Daten dar, welche gespeichert und ramDataOut
welche gelesen werden sollen. Die Datenbusse sind jeweils 8 Bit groß
          ramDataIn    : IN  BIT_VECTOR (7 DOWNTO 0);
          ramDataOut   : OUT  BIT_VECTOR (7 DOWNTO 0)
           );
END blockRAM;

ARCHITECTURE Behavioral OF blockRAM IS

--Deklaration eines RAMs mit einer Tiefe von 8 Bit und einer Länge von
256
TYPE RAM_TYPE IS ARRAY (0 to 255) of BIT_VECTOR (7 DOWNTO 0);

--Initialisierung der ersten 50 Byte des RAMs „blockRAM“ mit 01-50,
die übrigen werden mit "00" initialisiert
signal blockRAM: RAM_TYPE:=
(
X"01", X"02", X"03", X"04", X"05", X"06", X"07", X"08", X"09", X"10",
X"11", X"12", X"13", X"14", X"15", X"16", X"17", X"18", X"19", X"20",
X"21", X"22", X"23", X"24", X"25", X"26", X"27", X"28", X"29", X"30",
X"31", X"32", X"33", X"34", X"35", X"36", X"37", X"38", X"39", X"40",
X"41", X"42", X"43", X"44", X"45", X"46", X"47", X"48", X"49", X"50",
others => X"00");
```

```
BEGIN

--Um später das Beschreiben des RAMs leichter einbinden zu können,
wird hier der mit 50MHz getaktete Prozess beschrieben.

clocked50Process: PROCESS(rst, clk50)
      BEGIN
            IF rst ='1' THEN
--Bei steigender Taktflanke des 50MHz- Taktes wird abgefragt ob der
RAM zum Schreiben bereit ist und derzeit nicht aus dem RAM gelesen
wird. Wenn das der Fall ist werden die Daten, welche an ramDataIn an-
liegen, in die entsprechend durch ramAddr angegebene Speicherzelle ge-
schrieben.
            ELSIF clk50'EVENT AND clk50 = '1' THEN
                  IF ram_we = '1' AND ram_en ='0' THEN
                        blockRAM(CONV_INTEGER(ramAddr)) <= ramDataIn;
                  END IF;
            END IF;
END PROCESS clocked50Process;--Ende des 50MHz taktgesteuerten
Prozesses

--Bei steigender Taktflanke des Taktes vom PHY-Baustein kommend, wird
abgefragt ob der RAM zum Lesen bereit ist und derzeit nichts in die
Speicherzellen geschrieben wird. Ist dies der Fall, werden die Daten,
welche unter der durch ramAddr angegebenen Speicherzelle an den Vektor
ramDataOut übergeben

txClockedProcess: PROCESS (rst, tx_clk)
      BEGIN
        IF rst = '1' THEN
        ELSIF tx_clk'EVENT AND tx_clk = '1' THEN
                  IF ram_en = '1' AND ram_we ='0' THEN
                        ramDataOut <= blockRAM(CONV_INTEGER(ramAddr));
                  END IF;
            END IF;
END PROCESS txClockedProcess;--Ende des taktgesteuerten Prozesses

END Behavioral;
```

G User Constraints File

Folgend ist der Inhalt des User Constraints File dargestellt. Anhand dieser Daten können die, in der Toplevel- Architektur deklarierten, Ein- und Ausgangssignale mit den Anschlüssen der entsprechenden Bauelemente verbunden werden.

```
#Verknüpfung des Taktsignals mit dem 50MHz Taktsignals des Boards
NET "clk50"  LOC = "C9"  | IOSTANDARD = LVCMOS33;
NET "clk50" PERIOD = 20.0ns HIGH 50%;

#Verknüpfung der Signale mit entsprechenden Tastern
NET "BTN_writeToRAM" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_newFrame" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "RST" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;

#Verknüpfung der Leuchtdioden 0-7
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE =
8 ;
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE =
8 ;
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE =
8 ;
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE =
8 ;
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE =
8 ;
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE =
8 ;
NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE =
8 ;
NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE =
8 ;

#Verknüpfung mit dem PHY- Baustein
#Verknüpfung des Kollisionerkennungssignals
NET "COL" LOC = "U6" | IOSTANDARD = LVCMOS33 ;
#Verknüpfung des Taktsignals
NET "TX_CLK" LOC = "T7" | IOSTANDARD = LVCMOS33 |
CLOCK_DEDICATED_ROUTE = FALSE;
NET "TX_CLK" PERIOD = 40.0ns HIGH 50%;
#Verknüpfung des Enable- Signals
NET "TX_EN" LOC = "P15" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE
= 8 ;
#Verknüpfung des 4-Bit breiten Datenbusse
NET "TX_DATA<0>" LOC = "R11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 8 ;
NET "TX_DATA<1>" LOC = "T15" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 8 ;
NET "TX_DATA<2>" LOC = "R5" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 8 ;
NET "TX_DATA<3>" LOC = "T5" | IOSTANDARD = LVCMOS33 | SLEW = SLOW |
DRIVE = 8 ;
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Janutta, Benjamin (2009)
Dissertation – Inbetriebnahme und Funktionsnachweis des OPERA Precision Trackers insbesondere des Zeitmesssystems

[2] SMSC LAN83C185 Datasheet Revision 0.8 (06-12-08)

[3] Wirshark Media Management
http://www.wireshark.org (10.12.2009)

[4] packETH ethernet packet generator
http://packeth.sourceforge.net/ (06.01.2010)

[5] Rech, Jörg (2002)
Ethernet: Technologien und Protokolle für die Computervernetzung
Verlag Heinz Heise GmbH & Co. KG, Hannover 2002

[6] Proebster, Walter E. (1998)
Rechnernetze: Technik, Protokolle, Systeme, Anwendungen
Verlag Oldenbourg, München / Wien 1998

[7] Bayrischer Rundfunk Video: Was war der Äther?
http://www.br-online.de/br-alpha/alpha-centauri/alpha-centauri-aether-harald-lesch-ID1207830558733.xml (02.09.2009)

[8] Das Elektronik- Kompendium: Interview mit Robert Metcalfe
http://www.elektronik-kompendium.de/sites/net/0603201.htm (04.09.2009)

[9] Direct Industry – Die virtuelle Industriemesse
http://www.directindustry.de/prod/metrofunk-kabel-union/elektrisches-twisted-pair-kabel-utp-21344-380974.html (04.01.2010)

[10] TU- Ilmenau | Hauptseminar: Carrier- Netze in Deutschland
http://zack1.e-technik.tu-ilmenau.de/~webkn/Arbeiten/Hauptseminarreferat/Carrier-Netze/ (04.09.2009)

[11] Fachhochschule München | Netzmafia
Grundlagen Computernetze, Prof. Jürgen Plate
http://www.netzmafia.de/skripten/netze/ (03.09.2009)

[12] ITWissen- Das große Online- Lexikon für Informationstechnologie
http://www.itwissen.info/definition/lexikon/multi-level-transmission-MLT-MLT-Codierung.html (12.12.2009)

[13] Dr. George W. Benthien | Digital Encoding and Decoding
http://gbenthien.net/encoding.pdf (19.12.2009)

[14] elektroniknet.de | Weka Fachmedien GmbH
Fehlererkennung in FPGAs
http://www.elektroniknet.de/home/bauelemente/fachwissen/uebersicht/aktive-bauelemente/programmierbare-logikasics/fehlererkennung-in-fpgas/druckversion/ (22.12.2009)

[15] Xilinx DS312 April 18, 2008 Spartan-3E FPGA-Family: Complete Datasheet

[16] Fricke, Klaus (2008)
Digitaltechnik
5. Auflage
Verlag: Friedr. Vieweg & Sohn Verlag | GWV Fachverlag GmbH, Wiesbaden 2007

[17] Marwedel, Peter (2008)
Eingebettete Systeme
Verlag: Springer- Verlag Berlin, Heidelberg 2008

[18] Lenkeit, Jan
Diplomarbeit – Kalibrationsmessungen für das Driftröhren- Myon- Spektrometer des OPERA- Detektors