Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Kevin Bolls

System on Chip Entwicklung einer FPGA basierten Echtzeitverarbeitung von Radar-Informationen unter Verwendung eines in Hardware realisierten UDP/IP-Stacks

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Hamburg University of Applied Sciences
Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Kevin Bolls

System on Chip Entwicklung einer FPGA basierten Echtzeitverarbeitung von Radar-Informationen unter Verwendung eines in Hardware realisierten UDP/IP-Stacks

Masterarbeit eingereicht im Rahmen der Masterprüfung
im gemeinsamen Studiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Jürgen Reichardt
Zweitgutachter : Prof. Dr.-Ing. Stephan Hußmann

Abgegeben am 15. Februar 2012

Kevin Bolls

Thema der Masterthesis

System on Chip Entwicklung einer FPGA basierten Echtzeitverarbeitung von Radar-Informationen unter Verwendung eines in Hardware realisierten UDP/IP-Stacks

Stichworte

Navigations-Radar, System on Chip, Embedded System, Echtzeit, MicroBlaze, DAC, Ethernet, Multicast, Hardware Ethernet-Stack, Spartan-6 FPGA, VHDL, C, C#

Kurzzusammenfassung

Radar-Transceiver der neuesten Generation übertragen empfangene Reflexionen und notwendige Synchronisationspulse nicht mehr mittels analoger Signale zu den Sichtgeräten, sondern digitalisieren diese unter Einhaltung harter Echtzeitanforderungen. Auf diese Weise kann eine Übertragung der empfangenen Reflexionen über Ethernet statt finden. Dadurch entstehen jedoch Kompatibilitätsprobleme zwischen Radar-Transceivern, die digitale Informationen bereitstellen und Sichtgeräten, die analoge Signale benötigen. In dieser Arbeit sind Voruntersuchungen, Konzeptionierung, Entwicklung und Test eines Prototypen auf Basis eines FPGAs durchgeführt worden. Mittels eines in Hardware realisierten Ethernet-Stacks werden die Multicast-Nachrichten eines Radar-Transceivers über Ethernet empfangen und unter Einhaltung harter Echtzeitanforderungen das ursprüngliche, analoge Signalverhalten des Radar-Transceivers rekonstruiert. Sichtgeräte die analoge Signale benötigen können hierdurch wieder an das digitale System angebunden werden.

Kevin Bolls

Title of the paper

System on Chip development of a FPGA based real time processing of radar information under use of an UDP/IP stack realised in hardware

Keywords

Navigations-Radar, System on Chip, Embedded System, Realtime, MicroBlaze, DAC, Ethernet, Multicast, Hardware Ethernet-Stack, Spartan-6 FPGA, VHDL, C, C#

Abstract

During the past generations Radar-Transceivers provided the received echo information and control signals in analogue form. The latest generations of radar transceivers digitize received reflections and process them under hard real-time requirements, before they are sent via an Ethernet. For this reason, problems arise in compatibility. Radar indicators, that need analog signals, are not supported anymore. The goal of this master thesis is the development of an interface to analogize digital information received from the radar-transceiver and to reproduce the same behavior with hard real-time requirements like the original analog signals before digitization. For that reason this thesis deals with preliminary studies, conceptual design, development and testing of a prototype based on a FPGA. To meet the above requirements among others a hardware Ethernet-Stack is developed.

Danksagung

An dieser Stelle möchte ich mich bei meinem betreuenden Prüfer, Herrn Prof. Dr. rer. nat. Jürgen Reichardt, für seine intensive Betreuung und die vielen Anregungen danken. Ebenso danke ich Herrn Prof. Dr.-Ing. Stephan Hußmann für sein Interesse an dieser Arbeit und die Übernahme der Aufgabe des Zweitgutachters.

Bei Herrn Wolfgang Voß möchte ich mich für seine organisatorische Unterstützung während der Master-Arbeit bedanken. Herrn Hubert Hajek danke ich, dass er sich als unbeteiligter Lektor zur Verfügung gestellt hat. Ein weiterer Dank geht an meine weiteren Kollegen der Entwicklung, die für fachliche Diskussionen zu systemspezifischen Themen immer zur Verfügung standen.

Ebenso möchte ich an dieser Stelle ganz herzlich meiner Familie für deren Unterstützungen während der Masterarbeit und des Studiums danken.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACP Azimuth Clock Pulse

ARP Azimuth Reference Pulse

ARP Adress Resolution Protocol

AXI Advanced eXtensible Interface Bus

BMM Block RAM Memory Map

BRAM Block RAM

CLB Configurable Logic Block

CSMA/CD Carrier Sense Multiple Access/Collision Detection

DAC Digital Analog Converter

DDR Double Data Rate

DMA Direct Memory Access

FF FlipFlop

FIFO First-In First-Out Speicher

FLASH Digitaler Speicherchip

FPGA Field Programmable Gate Array

FMC FPGA Mezzanine Card

FSL Fast Simplex Link

FSM Finite State Machine

GMII Gigabit Media Independent Interface

GPIO General Purpose Input/Output

GUI Graphical User Interface

ICAP Internal Configuration Access Port

ICMP Internet Control Message Protocol

IGF Interframe Gap

IGMP Internet Group Management Protocol

IOB Input-Output-Block

ISA Instruction Set Architecture

ISR Interrupt Service Routine

IP Internet Protocol

IP-Core Intellectual property core

IPERF Tool zur Messung von UDP-/TCP-Durchsatz von Netzwerken

LUT Look-Up-Table

MAC Media Access Control

MCS Intel MCS-86 Hexadecimal Object

MUX Multiplexer

MII Media Independent Interface

NRE-Costs Non-recurring engineering costs

OSI-Modell Open Systems Interconnection Reference Model

PLB Processor Local Bus

PRF Pulse Repetition Rate

PRT Pulse Repetition Time

Radar Radio Detection and Ranging

RAM Random Access Memory

ROM Read Only Memory

RPM Revolution per minute

SFD Start-Frame-Delimiter

SPI Serial Peripheral Interface

SDMA Soft Direct Memory Acces

SGDMA Scatter-Gather Direct Memory Access

TCP Transmission Control Protocol

UDP User Datagram Protocol

Bildverzeichnis

Tabellenverzeichnis

KAPITEL 1

Einleitung

In den letzten drei Jahrzehnten hat sich Ethernet zu einem der weit verbreitetsten Kommunikationsstandards entwickelt. Während es anfänglich ausschließlich im Büroumfeld zur Kommunikation zwischen Computern eingesetzt wurde, erlangt Ethernet inzwischen einen immer größer werdenden Einzug im industriellen Umfeld. Es verdrängt damit klassische Feldbusse und vereinheitlicht die Kommunikation von der Feldebene bis in die Leitebene. Durch die Anforderungen, die ein industrieller Einsatz stellt, kann Ethernet inzwischen auch unter ungünstigen Umwelteinflüssen eingesetzt werden. Hierzu gehören Temperaturschwankungen, starke Erschütterungen (Schock), dauerhafte Vibrationsbelastungen, elektromagnetische Störungen und korrosive Einflüsse durch Feuchtigkeit und salzhaltige Luft. Umwelteinflüsse, wie sie auch auf Schiffen vorzufinden sind. Im Zuge des Strebens nach einer höher werdenden Effizienz von Schiffen erhält auch hier die Vereinheitlichung der Kommunikation durch ein "Industrial Ethernet" zunehmende Wichtigkeit. Ethernet wird dabei nicht nur in der Schiffsautomation eingesetzt, sondern auch bei der Steuerung und Navigation. Die verschiedensten Systeme auf einem Schiff entwickeln sich damit vom Analogen zum Digitalen und vom Bus zum Ethernet.
Aufgrund höher werdender Sicherheitsstandards und alternder Systeme ist ein Austausch der Navigations-Anlage etwa alle 10 Jahre [38] notwendig. Dies führt unweigerlich zu Kompatibilitätsproblemen zwischen der analogen und der digitalen Welt. Zu einem dieser Probleme gehören die Radar-Anlagen eines Schiffes. Während neue Radar-Transceiver Informationen über empfangene Echos digitalisieren und aufbereitet zu den Sichtgeräten übertragen, gibt es weiterhin Sichtgeräte, die analoge Signale benötigen, um eine Rundumsicht zu generieren.

1.1. Ziel der Arbeit

Das Ziel dieser Arbeit besteht daher darin, einen Prototypen zu entwickeln, welcher die Schnittstelle zwischen einem Radar-Transceiver, der digital die Radar-Informationen bereitstellt und einem Sichtgerät, das analoge Signale benötigt, bildet. Durch die Digitalisierung besteht dabei die Problematik, dass das im Analogen wichtige Zeitverhalten nicht mehr vorhanden ist. Die Entfernung der Ziele wird nicht mehr über die Zeit zwischen Senden eines elektromagnetischen Pulses und Empfangen einer Reflexion definiert - sondern über die Entfernung pro Sample. Die Richtung, aus der die Echos empfangen wurden, wird nicht mehr über analoge, winkelstabile Synchronisationspulse ermittelt, sondern über digitale Zahlenwerte definiert. Um wieder analoge Signale zu generieren, muss genau dieses Zeitverhalten der Signale rekonstruiert werden. Die vor der Digitalisierung einzuhaltenden harten Echtzeitbedingungen gelten auch wieder bei der Analogisierung. Diese ist so genau durchzuführen, dass ein Sichtgerät keinen Unterschied zu analog arbeitenden Radar-Transceivern feststellen kann.

Es ist damit ein eingebettetes System zu entwickeln, welches auf der einen Seite eine performante Anbindung an ein Ethernet aufweist und auf der anderen Seite analoge Radar-Signale reproduzieren kann. In allen Betriebsmodi des Radar-Transceivers und den sich dadurch ergebenden unterschiedlichen Auslastungen der Elektronik muss die Signalqualität den Anforderungen genügen. Kosteneffizienz ist bei der Entwicklung genauso zu beachten, wie Zuverlässigkeit und Fehlerfreiheit, die es durch Tests zu bestätigen gilt.
Das Bild 1.1 stellt die Integration dieses Prototypen im Navigations-System schematisch dar. Im oberen Teil des Bildes ist ein System dargestellt, bei dem die Übertragung der Daten digital erfolgt. Im unteren Teil des Bildes erfolgt hingegen die Übertragung der empfangenen Reflexionen analog. Der zu entwickelnde Umsetzer verbindet beide Systeme miteinander.
Tabelle 1.1 listet alle funktionalen und nicht funktionalen Anforderungen an das zu entwickelnde System auf.

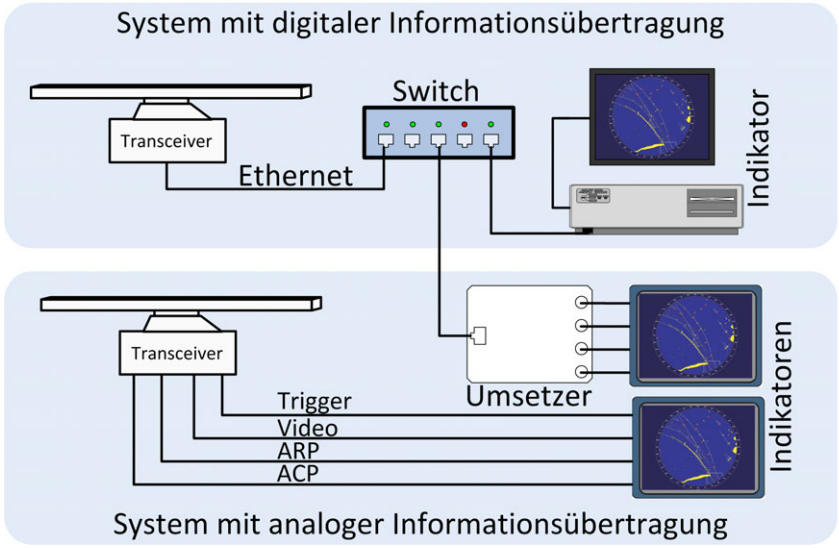


Bild 1.1.: Navigations-System mit zu entwickelndem Radar-Signal-Umsetzer

1.2. Gliederung der Arbeit

Bevor die schriftliche Ausarbeitung dieser Masterarbeit beginnt, werden in Kapitel 2 verschiedene Grundlagen erläutert, die zum Verständnis dieser Arbeit notwendig sind. Auf spezielle Thematiken wird zum besseren Verständnis in den Kapiteln direkt eingegangen.
In Kapitel 3 werden zu Beginn Untersuchungen durchgeführt, die verschiedene Realisierungen zur Anbindung eines FPGAs an ein Ethernet betrachten. Auf Basis dieser Ergebnisse findet in Kapitel 4 die Konzeptionierung des zu implementierenden Designs statt. Dabei werden unterschiedliche Realisierungsmöglichkeiten verschiedener Problemstellungen gegenübergestellt und unter Beachtung der Anforderungen an das Design aus Abschnitt 1.1 bewertet und ausgewählt. Kapitel 5 beschäftigt sich anschließend mit dem ersten Teil der Implementierung - der Implementierung des Hardware-Designs. Dabei werden die Schwerpunkte auf die Ethernet-Anbindung, die Radar-Video-Verarbeitung und die Schnittstelle zwischen Hard- und Software gelegt. In Kapitel 6 folgt der zweite Teil der Implementierung - die der Software. Schwerpunkte werden hier bei der Erzeugung der Radar-Pulse und der Updatefähigkeit des FPGAs über Ethernet liegen. Im Anschluss wird in Kapitel 7 auf Systemtests eingegangen und im Speziellen auf

Untersuchungen der Signalqualität der erzeugten Radar-Signale sowie die Dimensionierung des Video-Buffers vertieft. Ebenso erfolgt die Untersuchung der Hardware-Software-Schnittstelle. Kapitel 8 resümiert die Arbeit und gibt einen Ausblick auf eine mögliche Fortsetzung dieser Arbeit.

	Funktionale Anforderungen	Nicht funktionale Anforderungen
Ethernet-Anbindung	Unterstützung der Protokolle UDP, IP, ARP, IGMP und ICMP	Senden und Empfangen mit mindestens 50 MBit/s
	Dynamische Konfiguration des Ethernet-Stacks zur Laufzeit	Uneingeschränkte Unterstützung einer FullDuplex-Übertragung
	Senden und Empfangen von Unicast-UDP/IP-Nachrichten	Verwaltung der Mitgliedschaft in bis zu 3 Multicast-Gruppen unter Verwendung des IGMPv2 Protokolls
	Flexible Auslegung des Ethernet-Stacks um eine Verwendung für zukünftige Projekte zu ermöglichen	Fehlerfreies Senden- und Empfangen von Ethernet-Paketen
	Empfang von Multicast-UDP/IP-Nachrichten	
Radar	Ausgabe des Radarvideos als analoges Signal	Verarbeiten einer Pulsfolgefrequenz zwischen 400 Hz und 1500 Hz
	Ausgabe der Synchronisationspulse ACP und ARP	Verarbeiten von 20 - 80 Antennenumdrehungen pro Minute
	Generierung von ARP-Pulsen wahlweise als Headmarker oder Northmarker	Variabel konfigurierbare Ausgabe von 72 bis 8192 ACP-Pulsen pro Umdrehung
	Ausgabe eines analogen (Pre-)Trigger-Puls	Ausgabe von einem ARP-Puls pro Umdrehung
	Rekonstruktion des ursprünglichen Zeitverhaltens der Radarsignale, wie es im Radar-Transceiver vorlag	Jittern der Pulse darf bei der Darstellung der Reflexionen durch das Sichtgerät keinen wahrnehmbaren Winkelfehler verursachen
Allgemeines	Entwicklung als Protyp auf Basis von Evaluationsboards	
	Updatefähigkeit der FPGA-Konfiguration über Ethernet	
	Konfiguration und Steuerung der Elektronik ausschließlich über Ethernet	
	Entwicklung einer Lösung unter Beachtung des Kostenfaktors	
	Realisierung auf FPGA der Spartan-Familie von Xilinx	

Tabelle 1.1.: Funktionale und nicht funktionale Anforderungen an den Prototypen

KAPITEL 2

Grundlagen

Das folgende Kapitel beschäftigt sich mit verschiedenen theoretischen Hintergründen, die zum Verständnis dieser Arbeit notwendig sind. Abschnitt 2.1 geht hierfür auf den Aufbau von FPGAs ein, bevor sich Abschnitt 2.2 mit Grundlagen zum Thema Ethernet beschäftigt. In Abschnitt 2.3 wird abschließend auf das Funktionsprinzip von Radaranlagen eingegangen.

2.1. FPGAs

Im FPGA (engl. Field Programmable Gate Array) können logische Schaltungen mit Hilfe von Logikelementen wie z. B. FlipFlops (FF), Look-Up-Tabellen (LUT), Multiplexer (MUX), Distributed RAM und Schiebe-Registern konfiguriert werden. Mehrere dieser Elemente werden zu einem elementaren Logik-Block, dem sog. “Slice”, zusammengefasst.
Um den Ressourcenverbrauch verschiedener Designs im Vergleich bewerten zu können, werden in erster Linie entweder die Anzahl benötigter Look-Up-Tabellen (LUTs) und Register/Flip-Flops (FF) oder die Anzahl an benötigten Slices angegeben. Dabei ist zu beachten, dass i. d. R. nicht alle Logikelemente in einem Slice genutzt werden, sodass die Angabe der benötigten Slices keine konkrete Aussage über die Anzahl an benötigten Logikelementen macht.
Ein direkter Vergleich zwischen FPGAs verschiedener Familien über die jeweilige Anzahl ist nicht zulässig. Der interne Aufbau unterscheidet sich oft so grundsätzlich, dass die Anzahl an benötigten Slices bzw. LUTs und Registern deutlich variieren kann. Bei einem Spartan-3 beinhaltet ein Slice 2 LUTs und 2 FlipFlops [51], ein Virtex-5 hat 4 LUTs und 4 FlipFlops [52] und ein Spartan-6 bereits 4 LUTs und 8 FlipFlops [53]. Diese Logikelemente können sich zudem unterscheiden, so haben etwa die LUTs vom Virtex-5 und dem Spartan-6 sechs Eingänge, während die Spartan-3 LUT vier Eingänge besitzen. Innerhalb einer FPGA-Familie ist die Anzahl an LUTs und Registern in jedem Slice gleich. Es gibt jedoch verschiedene Slice-Typen innerhalb eines FPGAs, die eine unterschiedliche Anzahl zusätzlicher Logikelemente beinhalten.
Slices werden in sog. “Configurable Logic Blocks” (CLB) zusammengefasst. Beim Spartan-3 sind vier Slices in einem CLB, beim Spartan-6 und Virtex-5 hingegen zwei Slices. CLBs werden dann über Schaltmatrizen an die Routing-Ressourcen angebunden, um eine Verbindung z. B. mit anderen CLBs herstellen zu können. Neben CLBs gibt es weitere Blöcke. Hierzu zählen u. a. IOBs (Input-Output-Block), Digital-Clock-Manager-Blöcke und Block-RAM-Blöcke.

2.2. Ethernet

Im folgenden Abschnitt werden die wichtigsten Grundlagen des Ethernet und der Kommunikation über Ethernet erläutert.

2.2.1. Aufbau eines Ethernet-Interfaces

Die Kommunikations-Abläufe in einem Netzwerk können in verschiedenen Schichten abstrahiert werden. Diese Schichten lassen sich durch das OSI-Schichtenmodell [18] beschreiben, wie dem Bild 2.1 entnommen werden kann.

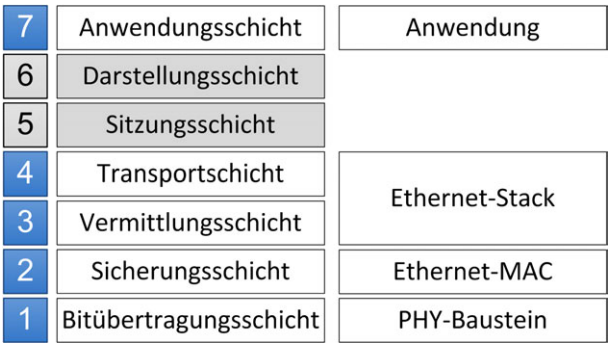


Bild 2.1.: OSI-Schichtenmodell eines Ethernet-Interfaces

1. OSI-Schicht (Bitübertragungsschicht)

Auf der untersten, ersten OSI-Schicht, kommt der sog. PHY-Baustein zum Einsatz. Dieser stellt das physikalische Interface zwischen dem Übertragungsmedium und der zweiten OSI-Schicht dar. PHY-Bausteine besitzen hierzu zwei Interfaces. Auf der Seite des Übertragungsmediums kommt das medienabhängige Interface zum Einsatz (MDI, engl. “Media Dependend Interface”). Dieses variiert in Abhängigkeit des verwendeten Mediums. Am gebräuchlichsten sind hier Twisted-Pair- oder Glasfaser-Leitungen. Zum OSI-Layer 2 hin kommt das medienunabhängige Interface (MII, engl. “Media Independend Interface”) zum Einsatz. Es sind bei diesem Interface verschiedene Schnittstellen-Standards definiert, mit denen ein PHY-Baustein angebunden werden kann. Die weit verbreitetsten Standards sind das gleichnamige MII-Interface und das GMII-Interface (siehe Bild 2.2). Das MII-Interface besteht je Richtung aus vier Datenleitungen (RXD und TXD) sowie einem Enable- (RXEN und TXEN) und einem Error-Signal (RXERR und TXERR). Das Enable-Signal gibt an, wann Daten auf den Datenleitungen liegen, das Error-Signal hingegen teilt einen Abbruch des Übertragungsvorganges mit. Ebenfalls stehen zwei Taktleitungen (RXCLK und TXCLK) zur Verfügung. Für beide Taktleitungen werden die Takte vom PHY-Baustein generiert. Der zweiten OSI-Schicht wird so mitgeteilt, wann die Daten in Sende-Richtung stabil anzuliegen haben bzw. in Empfangsrichtung stabil anliegen. Das MII-Interface wird für Übertragungen von 10 MBit/s und 100 MBit/s (Fast-Ethernet) verwendet, sodass mit einem Takt von 2.5 MHz bzw. 25 MHz gearbeitet wird. Soll eine Anbindung an ein Gigabit-Ethernet erfolgen, um 1 GBit/s übertragen zu können, muss das GMII-Interface genutzt werden. Dieses verwendet die Signalleitungen des MII-Interface und ergänzt diese um

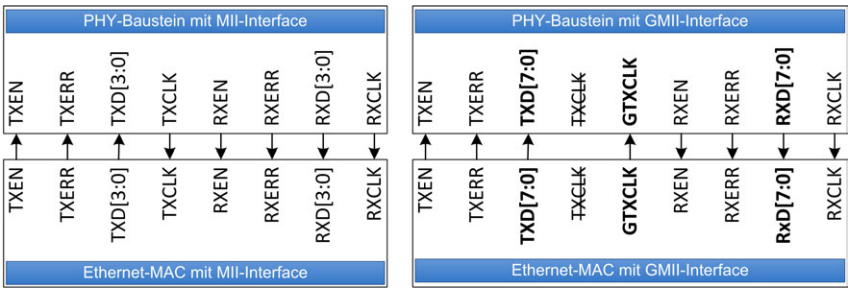


Bild 2.2.: Darstellung des MII- und GMII-Interface zwischen Ethernet-MAC und PHY-Baustein

weitere Signalleitungen. Damit werden anstelle von vier nun acht Datenleitungen für jede Richtung verwendet (siehe Bild 2.2). Die Taktfrequenz erhöht sich von 25 MHz auf 125 MHz. Dies führt dazu, dass der Sende-Takt nicht mehr vom PHY-Baustein selbst generiert wird. Auf einer separaten, zusätzlichen Leitung ist daher dem PHY-Baustein der Sende-Takt zur Verfügung zu stellen. Auf diesen Takt müssen die zu sendenden Daten synchronisiert sein. Die Sendetakt-Leitung des MII-Interface bleibt beim GMII-Interface hingegen unbenutzt.

2. OSI-Schicht (Sicherungsschicht)

In der zweiten OSI-Schicht befindet sich der sogenannte Ethernet-MAC (Media Access Control). Dieser ist immer in Hardware realisiert und kann damit, im Gegensatz zum PHY-Baustein, auch in einem FPGA implementiert werden. Die Verarbeitung in Hardware ist notwendig, da die Aufgaben des Ethernet-MAC zeitkritisch sind und eine zu hohe Rechenleistung bei einer Realisierung in Software benötigen würden.
Zu den Aufgaben zählt u. a. die Zugriffssteuerung auf das Übertragungsmedium. Bei einer Übertragung von 10/100 MBit/s besteht die Möglichkeit einer Kommunikation mittels Half-Duplex-Übertragung. Der Zugriff auf das Medium muss hier mittels des CSMA/CD (Carrier Sense Multiple Access/Collision Detection) Verfahren geschehen, um Daten-Kollisionen zu erkennen und einen erneuten Sende-Versuch starten zu können. Bei Full-Duplex-Übertragungen besteht die einzige Möglichkeit der Zugriffssteuerung im sog. “Flow-Control”. Bei dieser Flusssteuerung kann ein MAC-Control-Frame an eine hierfür reservierte Multicast-Gruppe gesendet werden, um dem Absender mitzuteilen keine Nachrichten für eine anzugebende Zeit zu senden. Eine weitere Aufgabe des Ethernet-MAC liegt in der Generierung und Auswertung eines Teils des MAC-Frames. Von besonderer Relevanz ist hier die enthaltene CRC-Summe, die über die gesamte Nachricht zur Fehlererkennung gebildet bzw. überprüft werden muss. Durch die Verarbeitung in Hardware werden die nachfolgenden Schichten in der hierfür notwendigen Rechenleistung entlastet. Ferner sorgt der Ethernet-MAC dafür, dass Präambel-Bits mit Start-Frame-Delimiter (SFD) vor der Nachricht zur Taktsynchronisation und eine Inter-Frame-Gap (IFG) nach der Nachricht zum Definieren eines Mindestabstands zwischen den Nachrichten beim Senden generiert wird.
Da, sofern das Routing der Switches diese nicht unterbindet, jeder Empfänger im Netzwerk alle Nachrichten empfängt, ist es notwendig, dass der Ethernet-MAC eine Filterung anhand der Empfänger-MAC-Adresse durchführt. Diese folgt direkt nach dem SFD. Die nachfolgenden Schichten, die oft in Software realisiert sind, können so deutlich entlastet werden.

3. OSI-Schicht (Vermittlungsschicht)

Die dritte OSI-Schicht wird, neben der vierten OSI-Schicht, durch den Ethernet-Stack repräsentiert. Der Ethernet-Stack ist dafür zuständig, die Kommunikation im Netzwerk auf Basis spezifizierter Netzwerk-Protokolle durchzuführen. In der dritten Schicht ist die Verarbeitung von Protokollen angeordnet, die für das korrekte Routing von Daten innerhalb eines Netzwerkes benötigt werden. Das wichtigste Protokoll hierbei ist das Internet-Protokoll (IP). Es ordnet jeder Nachricht eine IP-Adresse zu. Mittels der IP-Adresse besteht die Möglichkeit das Nachrichten über verschiedene Netzwerke hinweg zum richtigen Subnetz geroutet werden können. Innerhalb eines Subnetzes kann die Zustellung der Nachricht dann nicht mehr direkt über die IP-Adresse erfolgen, sondern muss über die physikalische Adresse eines Teilnehmers, der MAC-Adresse, geschehen. Damit Switches oder andere Teilnehmer im Subnetz eine Zuordnung zwischen IP-Adresse und MAC-Adresse herstellen können, ist das sog. ARP-Protokoll (Adress Resolution Protocol) notwendig. Dies erlaubt Anfragen an alle Teilnehmer, wer die in der Anfrage anzugebende IP-Adresse besitzt. Der zugehörige Teilnehmer antwortet mit einer ARP-Reply-Nachricht, welche die MAC-Adresse enthält. Auf diese Weise können Switches oder auch andere Netzwerk-Teilnehmer ARP-Tabellen anlegen, in denen die IP- den MAC-Adressen zugeordnet werden.

4. OSI-Schicht (Transportschicht)

In der vierten Schicht wird die Verarbeitung von Protokollen implementiert, die unabhängig vom Routing zwischen den Teilnehmern arbeiten. Zu diesen Protokollen zählt etwa das verbindungsorientierte TCP-Protokoll (Transmission Control Protocol) und das verbindungslose UDP-Protokoll (User Datagram Protocol). Diese Protokolle erlauben es, dass ein Netzwerkteilnehmer die empfangene Nachricht einer Applikation zuordnen kann, indem Port-Nummern verwendet werden.

7. OSI-Schicht (Anwendungsschicht)

Die OSI-Schicht 5 und 6 werden in dem hier benötigten Ethernet-Interface nicht benötigt. Diesen Schichten wären Protokolle wie das HTTP- (Hypertext Transfer Protocol) und FTP-Protokoll (File Transfer Protocol) zuzuordnen. Der OSI-Schicht 7 sind die Anwendungsprogramme enthalten.

Dem Anhang A.1 ist der konkrete Aufbau der Protokolle zu entnehmen, die für die hier vorliegende Anwendung benötigt werden.

2.2.2. Unicast, Broadcast und Multicast

Die klassische Kommunikation zwischen Teilnehmern ist die Unicast-Übertragung. In diesem Fall beinhalten die Ethernet-Pakete die MAC-Adresse des Empfängers. Soll die gleiche Nachricht an mehrere Teilnehmer versendet werden, so müsste diese bei einer Unicast-Übertragung an jeden Empfänger separat gesendet werden. Dies erhöht den notwendigen Datendurchsatz und belastet den Absender. Aus diesem Grund können Nachrichten auch per Broadcast übertragen

werden. Die MAC-Adresse ist hier immer FF-FF-FF-FF-FF-FF. Jeder Teilnehmer muss, sofern er diese MAC-Adresse nicht im Ethernet-MAC filtert, diese Nachrichten im Ethernet-Stack verarbeiten. Der Sender wird hierdurch entlastet, alle anderen Teilnehmer jedoch belastet, da diese Nachrichten i. d. R. immer durch den Ethernet-Stack und damit oft in Software verarbeitet und ggf. gefiltert werden müssen. Die Übertragung von Nachrichten mittels Multicast vereint die Vorteile von Uni- und Broadcast-Übertragungen. Jeder Teilnehmer kann sich hier in Gruppen anmelden. Diese Gruppen werden durch die Ethernet-Switches verwaltet, die daraufhin das Routing von Nachrichten, die an die Gruppe gesendet wurden, durchführt. Damit erhalten auf der einen Seite nur die Teilnehmer die Nachricht, die auch an dieser interessiert sind, auf der anderen Seite muss der Sender die Nachricht nur einmal versenden.

2.3. Radar

Radar steht abkürzend für "Radio Detection and Ranging" und ist Synonym für Erkennungs- und Ortungsverfahren, sowie Ortungsgeräte auf Basis elektromagnetischer Wellen. Unter den verschiedenen Radar-Prinzipien wird zur Navigation auf Schiffen das sogenannte Pulsradar als Rundsichtradar eingesetzt. Radar-Transceiver senden hier mit einer Pulsfolgefrequenz (PRF) von bis zu wenigen Kilohertz mit einer Leistung von 70 bis 74 dBm Primärsignale in Form gebündelter, vertikal polarisierter, elektromagnetischer Wellen aus. Aus der Messung der Laufzeit zwischen dem gesendeten Primärsignal und dem Empfang eines Sekundärsignals ist eine Bestimmung der Entfernung möglich. Als Sekundärsignal wird die an einem Objekt reflektierte elektromagnetische Energie (auch Echo) bezeichnet. Die reflektierte Energie hängt dabei linear von der Querschnittsfläche des Ziels und unter Berücksichtigung der hin- und rücklaufenden Welle biquadratisch von der Entfernung des Ziels ab. Reflektierte, elektromagnetische Wellen können bis zu einer Leistung von etwa -100 dBm von den Radar-Transceivern detektiert werden. Alle Reflexionen, die innerhalb eines auf einen Sendepuls folgenden Zeitfensters empfangen werden, werden im Folgenden als Video(-Sweep) bezeichnet.Der Beginn jedes Video-Sweeps wird durch einen Trigger-Puls repräsentiert (siehe Bild 2.3), der sich aus dem Sende-Puls ableitet und damit für die Entfernungsbestimmung von entscheidender Bedeutung ist.
Durch Drehen der Radarantenne bei wiederholtem Senden elektromagnetischer Pulse kann aus einer Vielzahl von Video-Sweeps eine 360°-Rundumsicht erzeugt werden. Damit ein Sichtgerät dies kann, ist zusätzlich jedem Sweep ein Winkel zuzuordnen, aus dem die Echos empfangen wurden. Für eine Bestimmung der Richtung kann z. B. ein optisch gesteuerter Impulsgeber am drehenden Teil der Antenne befestigt werden, der im Betrieb zwei Arten von winkelstabile Pulsen erzeugt. Mehrfach pro Umlauf generierte, winkelstabile Pulse werden im Folgenden als "Azimuth Clock Pulse" (ACP) bezeichnet (siehe Bild 2.3). Diese ACP-Pulse erlauben die Bestimmung der Winkeldifferenzen zwischen Video-Sweeps.
Zeigt die Antenne in Nordrichtung oder in eine auf das Schiff bezogene Richtung wird einmal pro Umlauf der "Azimuth Reference Pulse" (ARP[1]) generiert. Der ARP-Puls wird daher auch als Head- oder Northmarker bezeichnet. Nur so können die ACP-Pulse und damit die Radar-Sweeps auch absoluten Winkeln zugeordnet werden.
Bei analogen Radar-Transceivern werden diese vier Einzelsignale (Trigger, Video, ACP und

[1] Die Abkürzung ARP ist in dieser Arbeit doppelt belegt. Sie wird zum einen für den "Azimuth Reference Pulse" und zum anderen für das "Adress Resolution Protocol" verwendet.

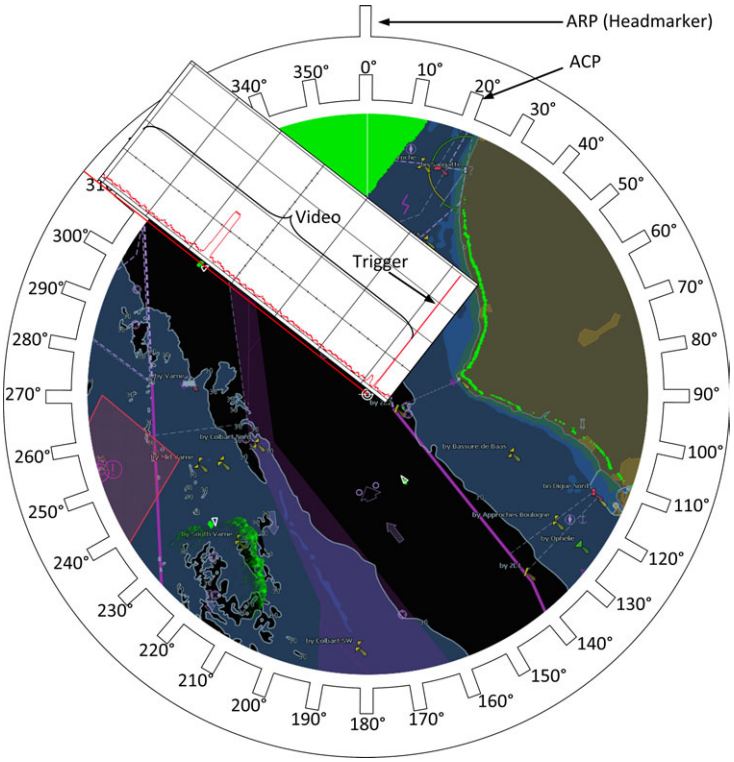


Bild 2.3.: Rundumsicht eines Radar-Sichtgerätes mit den dargestellten Radar-Signalen Video, Trigger, ACP und ARP

ARP) den Sichtgeräten in analoger Form zur Verfügung gestellt, die daraus eine Rundumsicht generieren. Bei digitalen Anlagen werden diese analogen Signale ganz, oder teilweise, im Radar-Transceiver digitalisiert und damit digital zu den Sichtgeräten übertragen.
Die erreichbare Winkelauflösung bei der Abtastung der Umgebung ist definiert durch die Richtcharakteristik der Antenne. Je schmaler die Hauptkeule ist, desto größer ist die erreichbare Auflösung bei der Abtastung der Umgebung. Die Breite der Hauptkeule liegt bei Radar-Antennen etwa im Bereich von $1°$.

KAPITEL 3

Voruntersuchungen

Für eine optimale Konzeptionierung sind Voruntersuchungen notwendig, die verschiedene Lösungsmöglichkeiten zur Realisierung der Aufgabenstellung untersuchen. Die Leistungsfähigkeit, der Ressourcenverbrauch, aber auch die Beurteilung möglicher Auswirkungen der Systemkomponenten untereinander kann so vorab bewertet werden, sodass die Konzeptionierung eines optimalen Systems ermöglicht werden kann. Das folgende Kapitel beschäftigt sich hierzu mit der Untersuchung von Möglichkeiten, FPGAs an ein "Industrial Ethernet" anzubinden. Die Anbindung an das Ethernet wird, aufgrund der Verarbeitung größerer Datenmengen, als die Aufgabe angesehen, die am meisten Rechenleistung benötigt und damit die größten Auswirkungen auf die Realisierbarkeit und Einhaltung der Anforderungen haben wird.

3.1. Übertragungsraten

Neben der erreichbaren Übertragungsrate stellt die hierfür notwendige Rechenleistung eines der wichtigsten Kriterien zur Bewertung einer Ethernet-Anbindung dar. Für eine richtige Bewertung dieses Kriteriums sind Vorbemerkungen notwendig.
Jede Übertragung von Nutzdaten in einem Ethernet-Netzwerk basiert auf dem IP-Protokoll und wahlweise dem TCP- oder UDP-Protokoll. Bei dem verbindungsorientiert ausgelegtem TCP-Protokoll werden Handshake-Verfahren genutzt, um fehlerhafte oder fehlende Nachrichten zu erkennen und ein erneutes Senden beim Absender anfragen zu können. Anders als beim verbindungslosen UDP-Protokoll entsteht damit eine zusätzliche Fehlersicherheit und bei größeren Netzwerken die Garantie, das die Reihenfolge der Datenpakete wiederhergestellt werden könnte, wenn Datenpakete aufgrund von Laufzeitunterschieden vertauscht werden. Der so deutlich höhere Protokoll-Overhead führt zwar auch zu einer geringeren Übertragungsrate der effektiven Nutzdaten gegenüber dem theoretischen Maximum, bei eingebetteten Systemen entscheidender ist jedoch der zusätzliche Kommunikationsaufwand und damit die zusätzlich benötigte Rechenleistung. Ferner führt die Nutzung des TCP-Protokolls auch dazu, dass ein größerer Buffer für die Nachrichten benötigt wird, damit bei erneuter Anfrage nach einem Datenpaket die Reihenfolge der Pakete wiederhergestellt werden kann. Gegenüber UDP-Nachrichten, die auf ein Handshake-Verfahren verzichten, weisen TCP-Nachrichten damit eine geringere Echtzeitfähigkeit auf.
In Netzwerken mit (gemanagten) Switches können Datenkollisionen nicht auftreten und Datenverluste praktisch ausgeschlossen werden, solange die verfügbare Bandbreite nicht überschritten wird. Nachrichten, die mittels des UDP-Protokoll versendet werden, können im "Industrial Ethernet" daher genauso zuverlässig übertragen werden, wie es auch bei der Verwendung des

TCP-Protokolls erreicht werden kann. UDP-basierte Übertragungen sind daher aufgrund des geringeren Protokoll-Overhead und der niedrigeren Rechenleistung die ein Netzwerk-Teilnehmer aufbringen muss vorzuziehen.
Die maximal erreichbare Übertragungsrate hängt bei einer UDP-basierten Übertragung damit maßgeblich von der Größe der einzelnen Datenpakete und in diesem Zusammenhang auch von der Leistungsfähigkeit des Systems ab. Je größer ein Datenpaket ist, desto größer ist die maximale Übertragungsgeschwindigkeit von Nutzdaten, durch den relativ geringer werdenden Anteil des Protokoll-Headers. Für in Software realisierte Ethernet-Stacks bedeuten größere Nachrichten aber auch eine geringere Anzahl an zu verarbeitenden Ethernet-Paketen und damit eine geringere absolute Rechenzeit zur Generierung und Interpretierung der Protokolle bei unverändertem Datendurchsatz. Nach dem IEEE 802.3 Standard, welcher das Ethernet standardisiert, ist die MTU-Größe[1] auf 1518 Byte beschränkt. Unter Berücksichtigung des Protokoll-Overheads durch den MAC-Frame, dem des IP- und UDP-Protokolls, sowie dem Start-Frame-Delimiter (SFD), die Präambel und die Inter-Frame-Gap (IFG) kann bei einer Übertragungskapazität von 100 MBit/s, insgesamt 95.7 MBit/s an Nutzdaten übertragen werden. Werden hingegen nur noch 64 Byte an UDP-Nutzdaten pro Ethernet-Paket übertragen, sinkt die maximal mögliche Übertragungsrate der Nutzdaten auf knapp über 49.2 MBit/s ab.
Es besteht nun die Möglichkeit Jumbo-Frames zu nutzen. Als Jumbo-Frames gelten alle Ethernet-Pakete, die eine MTU-Größe größer als 1518 Byte aufweisen. Die tatsächliche Übertragungskapazität der Nutzdaten kann dadurch zwar gesteigert werden, der eigentliche Vorteil liegt aber darin, dass die Interrupt-Rate durch eine noch geringer werdende Anzahl an Datenpaketen abnimmt und damit noch weniger Protokoll-Header zu verarbeiten sind. Dadurch kann deutlich Rechenleistung eingespart werden. Gerade im Bereich eingebetteter Systeme ist dies oft unerlässlich, um höhere Übertragungsraten erreichen zu können, wie u. a. [35] zeigt. Da Jumbo-Frames bei Übertragungen mit 10/100 MBit/s sowie 1 GBit/s nicht standardisiert [43] sind, kann eine Verarbeitung von Jumbo-Frames durch alle Netzwerkteilnehmer, sowie die eingesetzten Switches/Router nicht garantiert werden.
Während etwa im Multimedia-Bereich die Leistungsfähigkeit durch Jumbo-Frames gesteigert werden kann, werden bei “Industrial Ethernet”-Netzwerken stattdessen möglichst kleine Datenpakete verwendet [9]. Dies wirkt sich zwar nachteilig auf die maximale Übertragungsgeschwindigkeit und die zur Übertragung benötigte Rechenperformance der einzelnen Netzwerk-Teilnehmer aus, steigert aber die Echtzeitfähigkeit[2] des Netzwerkes, denn Ethernet-Pakete können nicht unterbrochen werden. Je länger demnach ein Datenpaket ist, desto größere Verzögerungen entstehen für andere Datenpakete im Netzwerk. Wird ein eingebettetes System eingesetzt, bei dem die Abarbeitung eines Ethernet-Paketes nicht durch andere Aufgaben unterbrochen werden kann, führen längere Ethernet-Pakete auch hier zu einer Abnahme der erreichbaren Echtzeitfähigkeit des eingebetteten Systems.
Jumbo-Frames werden daher nicht zur Übertragung von Radar-Videoinformationen verwendet, wodurch diese Möglichkeit der Performancesteigerung nicht genutzt werden kann. Eine Untersuchung der möglichen Ethernet-Interfaces unter Verwendung von Jumbo-Frames wird daher nicht durchgeführt.

[1] MTU steht für “Maximum Transmission Unit” und beschreibt die maximale Paketgröße eines Protokolls in der Vermittlungsschicht des OSI-Modells, und damit die Größe des IP-Protokolls, ohne das eine Fragmentierung der Nachricht notwendig ist. Der MAC-Frame wird damit hier nicht zugezählt.

[2] Als Echtzeit kann hier verstanden werden, dass die Übertragung einer Nachricht in einem definierten Zeitfenster sichergestellt werden kann.

3.2. Lösungsansätze

Ein oft vorhandene Gemeinsamkeit von performanten eingebetteten Systemen, ist die optimale Nutzung der Synergie von Hardware und Software. Eine Realisierung von Aufgaben in Software und damit auf Prozessoren ist prädestiniert für die Verarbeitung komplexer Aufgabenstellungen, zu denen etwa mathematische Berechnungen, sowie Steuer- und Überwachungsaufgaben zählen. In Hardware realisierte Abarbeitungen sind hingegen prädestiniert für performancekritische Aufgabenstellungen. Hierzu zählen Aufgaben mit Anforderungen an die Verarbeitung hoher Datendurchsätze, an die besonders schnelle Berechnung von Ergebnissen, an die Nebenläufigkeit oder aber auch an die deterministische Abarbeitung der Aufgaben und damit auch Anforderungen an die echtzeitfähige Abarbeitung einer Aufgabe.
Aus diesem Grund sollen hier Lösungsansätze untersucht werden, die alle drei möglichen Abstufungen abdecken. Dies sind vollständig in Software realisierte Lösungen des Ethernet-Stacks, in Software realisierte Ethernet-Stacks mit der Nutzung von Hardware-Beschleunigern, und vollständig in Hardware realisierte Ethernet-Stacks.
Der für ein Ethernet-Interface ebenfalls notwendige Ethernet-MAC muss immer in Hardware realisiert werden.

3.3. Untersuchung von in Software realisierten Lösungen

Eine Realisierung in Software erlaubt eine sehr flexible Lösung. Software-Stacks, wie der lwIP-Stack [15], können an den gewünschten Anwendungsfall, etwa im Umfang der zu unterstützenden Protokolle oder plattformspezifische Gegebenheiten, angepasst werden. Die in Software realisierte Protokollverarbeitung erlaubt einen abstrahierten Zugriff durch eine Software-Applikation mittels Sockets, sodass diese weitestgehend unabhängig von dem zu verwendenden Ethernet-Stack aufgebaut sein kann. Dies gilt etwa, wenn die Schnittstellen POSIX-Konform[3] sind. Dies wirkt sich positiv auf die Portierbarkeit der Applikations-Ebene aus.
Es muss sich vergegenwärtigt werden, dass Prozessorsysteme, wie der MicroBlaze[4], hochoptimierte Systeme mit etwa mehrstufigen Pipeline-Systemen, speziellen Recheneinheiten (z. B. FloatingPoint-Unit und Hardware-Multiplizieren) u. v. m. sind. Ein optimales Zusammenwirken der einzelnen Aufgaben und eine optimale Ausnutzung der Möglichkeiten die ein Prozessor bietet, kann bei der Verarbeitung von Ethernet-Paketen nur sehr schwer erreicht werden und ist ohne Hardware-Unterstützung kaum möglich. Durch unterschiedliche Abarbeitungsgeschwindigkeiten entstehen Wartezeiten, sodass Rechenzeit verloren geht. In [81] sind Untersuchungen für den frei verfügbaren lwIP-Ethernet-Stack angestellt worden, die eine Verteilung der benötigten Rechenleistungen für die einzelnen Aufgaben beim Empfang von Audio-Paketen über Ethernet untersucht. Der dort eingesetzt Prozessor benötigt rund 75 % für die gesamte Verarbeitung. Hiervon lediglich 12.5 % für die Applikation. Jeweils 43.75 % werden vom Ethernet-Stack

[3] POSIX (engl. Portable Operating System Interface) ist ein für Unix entwickeltes standardisierte API zwischen Applikation und dem Betriebssystem.
[4] Der MicroBlaze ist ein 32 Bit RISC-Mikrocontroller, der als konfigurierbarer Soft-Core auf FPGAs der Firma Xilinx eingesetzt werden kann.

und vom NIC-Treiber[5] verwendet. Der größte Anteil an Rechenperformance geht bei Software-basierten Lösungen bei folgenden Verarbeitungsschritten verloren:

Ethernet-MAC-Treiber stellen den Datenaustausch zwischen Ethernet-Stack und Ethernet-MAC her. Performanceverluste können hier zum einen durch die Kopiervorgänge zwischen Ethernet-MAC und Ethernet-Stack auftreten, zum anderen durch zu hohe Interrupt-Raten, die der Ethernet-MAC an den Ethernet-Stack gibt, damit dieser die Daten interpretiert. Häufige Kontext-Wechsel benötigen Rechenzeit.
Eine optimale Hardwareunterstützung dieses Vorganges stellt der Transport von Daten mittels SGDMA (Scatter Gather Direct Memory Access) dar. Dieses Verfahren zeichnet sich gegenüber herkömmlichen DMAs dadurch aus, dass Protokoll- und Nutzdaten eines Ethernet-Paketes getrennt in verschiedene Speicherbereiche geschrieben werden können, ohne dass Prozessorleistung benötigt wird. Beim lwIP-Stack sind in diesem Fall keine Kopiervoränge mehr notwendig [39], die Rechenzeit benötigen. Es wird hierzu mit zwei "pbufs"[6] gearbeitet, einer für die Header-Daten und einer für die Applikations-Daten.

Checksummenberechnungen sind in Protokollen wie TCP, IP und UDP[7] notwendig. Checksummenberechnungen sind nicht mit der CRC-Prüfsummenberechnung im MAC-Frame, die durch den Ethernet-MAC und damit in Hardware berechnet wird, zu verwechseln. Die Berechnung und Überprüfung der Checksumme verbraucht bei den erwähnten Untersuchungen nach [81] rund 24.9 % der vom lwIP-Ethernet-Stack benötigten Rechenleistung. Dies kann vermieden werden, wenn eine Auslagerung der Checksummenberechnung und -überprüfung in Hardware erfolgt (Checksum-Offloading).

Kopiervorgänge finden nicht nur zwischen Ethernet-MAC und Ethernet-Stack statt, sondern sind ggf. auch bei der Protokoll-Verarbeitung im Ethernet-Stack selbst notwendig. Eine Performancesteigerung auf Softwareseite ist durch Minimierung der Anzahl an Kopiervorgängen möglich. Eine Hardware-Unterstützung kann durch Einsatz eines (SG-)DMA oder durch Minimierung der Speicherzugriffszeiten die Performance erhöhen. Wird der lwIP-Stack durch den MicroBlaze auf einem FPGA verarbeitet, so besteht i. d. R. die Notwendigkeit einer Auslagerung in externen Speicher, der prinzipiell langsamer ist als lokaler BlockRAM-Speicher. Eine Verbesserung der Zugriffzeit kann entweder erreicht werden, indem für besonders Performance-kritische Zugriffe die Verwendung des lokalen BlockRAMs stattfindet, der in der Zugriffsgeschwindigkeit einem Level-1-Cache gleichkommt oder aber indem Speicher-Cache-Systeme eingesetzt werden.

Ist die Möglichkeit einer Hardware-Unterstützung nicht gegeben, oder diese bereits voll ausgenutzt, können bei Software-basierten Systemen weitere Leistungssteigerungen nur durch leistungsstärkere Prozessoren, höhere Taktfrequenzen bei der Verarbeitung und kürzeren Speicherzugriffszeiten erreicht werden.

[5] Der NIC-Treiber (Network Interface Card) ist in erster Linie verantwortlich für den Transport empfangener und zu sendender Daten zwischen Ethernet-MAC und Ethernet-Stack.

[6] Ein "pbuf" ist eine Struktur, die im lwIP-Stack ein Datenpaket repräsentiert.

[7] Nach dem RFC 768 Standard besteht optional die Möglichkeit auf Checksummenberechnung/-überprüfungen im UDP-Header zu verzichten. In diesem Fall ist das Checksummenfeld mit Nullen zu füllen.

3.3.1. Untersuchungskonzept

Untersucht werden sollen zwei verschiedene, softwarebasierte Varianten einer Ethernet-Anbindung. Jede Variante für sich wird bei verschiedenen Konfigurationen untersucht, die von keiner Hardwareunterstützung bis hin zur vollen, bei dem jeweiligen Design möglichen, Hardwareunterstützung reichen.
Beide Ausgangssysteme basieren auf der Verwendung des MicroBlaze-Mikrocontrollers (Version 8.00.b) der auf einem Virtex-5 (Typ: xc5vfx70t-1ff1136) FPGA implementiert wird und der die von Xilinx angepasste Version des unter BSD-Lizenz verfügbaren lwIP-Stacks [15] in der Version 2.1.0 verarbeitet. Der lwIP-Stack wird als geeignet angesehen, da dieser die in Abschnitt 1.1 geforderten Protokolle IP, UDP, ARP, IGMP[8] und ICMP unterstützt und von Xilinx auf die Nutzung mit dem MicroBlaze und den von Xilinx angebotenen Ethernet-MACs angepasst wurde. Im letzteren Punkt unterscheiden sich die Ausgangssysteme. Es wird der lwIP-Stack einmal in Verbindung mit dem Ethernet-MAC "XPS LL TEMAC" [58] und einmal mit dem "XPS Ethernet Lite" [59] untersucht. Beide Ethernet-MACs verwenden unterschiedliche Möglichkeiten der Anbindung an den Ethernet-Stack und unterscheiden sich deutlich in den Möglichkeiten, eine Hardware-Unterstützung zu nutzen.
Um die relative Prozessorauslastung bei verschiedenen Sende- und Empfangsgeschwindigkeiten bestimmen zu können, wird eine Software-Applikation entwickelt, deren Ausgangsbasis ein von Xilinx zur Verfügung gestelltes Beispiel [49] zur Bestimmung maximaler Übertragungsgeschwindigkeiten nach dem von Iperf[9] [28] vorgegebene Messverfahren ist. Da nicht die maximal zu erreichende Übertragungsgeschwindigkeit im Vordergrund steht, sondern die benötigte Rechenleistung bei der maximal zu verarbeitenden Übertragungsgeschwindigkeit, wird das zur Verfügung stehende Beispiel angepasst: Um verschiedene Übertragungsraten im Sende-Betrieb zu ermöglichen, wird die Sende-Routine nicht in der main-Schleife bearbeitet, wie es im Beispiel der Fall ist, sondern mittels Timer aufgerufen, sodass die Sendegeschwindigkeit steuerbar wird. Um die Prozessor-Auslastung zu messen, werden die Durchläufe in der main-Schleife gezählt und mittels eines weiteren Timer in äquidistanten Abständen über die RS232-Schnittstelle des Evaluationboards ausgegeben. Durch Vergleich der main-Schleifendurchläufe zwischen Last- und Ruhe-Betrieb kann die relative Prozessorauslastung bei der jeweiligen Übertragungsgeschwindigkeit bestimmt werden. Anders als beim vorgegebenen Beispiel wird jede fehlende/-fehlerhafte Nachricht ausgegeben. Eine Übertragungsgeschwindigkeit wird als nicht erreichbar angesehen, wenn bei der jeweiligen Geschwindigkeit ein einziger Fehler festgestellt wird.
Alle im Folgenden durchgeführten Messungen basieren auf der Übertragung von UDP/IP-Nachrichten mit 1200 Byte Nutzdaten.
Die in Abschnitt 3.3.2 und 3.3.3 erstellten Projekte sind dem Anhang A.3.3 zu entnehmen. Der Anhang A.3.1 enthält eine Dokumentation der Untersuchungen mit zusätzlichen Informationen zu den einzelnen Projekten.

[8] IGMP wird nur bei Anpassungen der Xilinx-Version des lwIP-Stacks unterstützt.

[9] Iperf ist ein Tool zur Messung von UDP-/TCP-Datendurchsätzen in einem Ethernet-Netzwerk. Das Messverfahren wird durch eine mit jeder Nachricht inkrementierten 32-Bit-Zahl am Anfang der Nachricht realisiert. Der jeweilige Empfänger kann dadurch überprüfen, ob bei der jeweiligen Übertragungsrate Nachrichten verloren gegangen sind und damit ob die Übertragungsgeschwindigkeit von dem Netzwerk-Teilnehmer verarbeitet werden kann.

3.3.2. Untersuchung mit dem Ethernet-MAC “XPS LL TEMAC”

Der “XPS LL TEMAC” ist ein von Xilinx entwickelter IP-Core, der als Soft-IP-Core auf verschiedenen FPGAs zum Einsatz kommen kann und auf einigen FPGAs auch als Hard-IP-Core[10] realisiert ist, wie es auf dem hier eingesetzten Virtex-5-FPGA der Fall ist. Der Ethernet-Core kann mittels SDMA-Engine über eine LocalLink-Verbindung (siehe Abschnitt 4.2.4) direkt zwischen RAM und Ethernet-MAC Daten austauschen. Die DMA-Engine ist hierzu im MPMC (Multi-Port Memory Controller) Speicher-Controller von Xilinx integriert. Der MPMC kann nur in Verbindung mit externem RAM eingesetzt werden. Optional besteht die Möglichkeit, die Berechnung und Überprüfung der Checksummen durch den “XPS LL TEMAC” in Hardware durchführen zu lassen (Checksum-Offloading). Der von Xilinx zur Verfügung gestellte lwIP-Netzwerkadapter unterstützt bei der Verwendung des “XPS LL TEMAC” bereits das Checksum-Offloading für TCP/IP Nachrichten, jedoch nicht für UDP-Nachrichten. In [35] wurde das Checksum-Offloading für den Sende-Betrieb implementiert, jedoch nicht für den Empfangs-Betrieb. Da hier die Empfangsgeschwindigkeit ebenfalls von Interesse ist, wurde auch die Möglichkeit des UDP-Checksum-Offloading für den Empfangs-Betrieb implementiert. Dem Anhang A.3.3 sind die hierfür zu verändernden Dateien zu entnehmen.

Ergebnisse für die Empfangsrichtung

Dem Bild 3.1 können die Untersuchungsergebnisse für die Empfangsrichtung entnommen werden. Feststellbar sind deutliche Performance-Verluste durch die Notwendigkeit externen Speicher zu benutzen. Durch den Einsatz von Cache wird es erst möglich, die geforderten 50 MBit/s

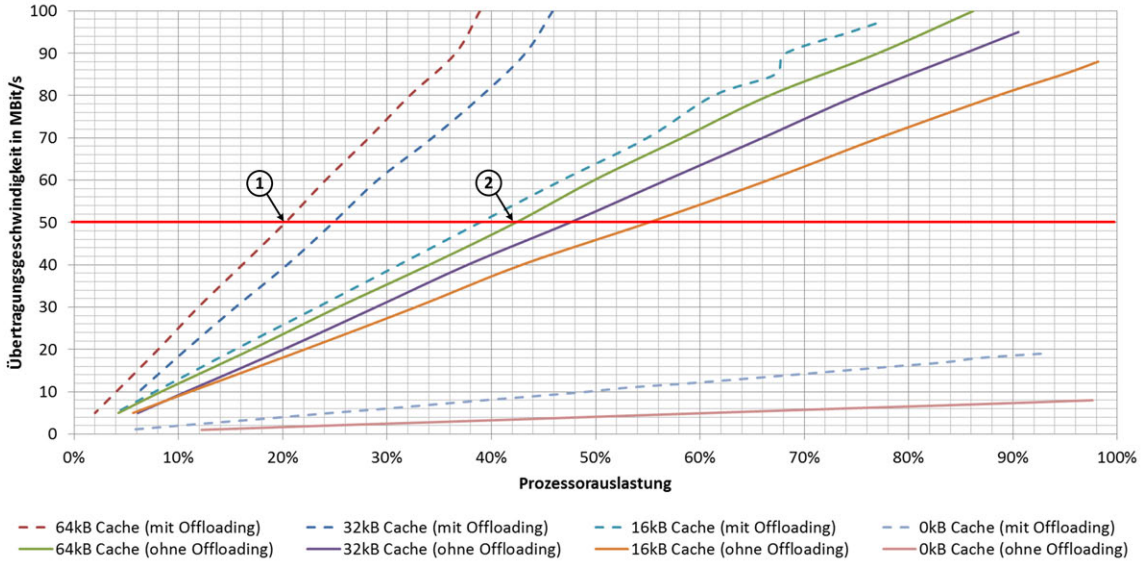


Bild 3.1.: Übertragungsgeschwindigkeit in Empfangsrichtung bei verschiedenen Prozessorauslastungen, verschiedenen Cache-Konfigurationen und optionalem Checksum-Offloading bei einem Systemtakt von 125 MHz und 1200 Byte Nutzdaten pro Ethernet-Paket

[10] Ein Hard-IP-Core ist ein fertig layouteter Block im FPGA-Chip. Hard-IP-Cores, wie der PowerPC auch einer ist, haben i. d. R. eine höhere Performance, da diese ohne Berücksichtigung FPGA spezifischer Gegebenheiten, hochoptimiert auf dem Die aufgebracht werden können.

(rote Linie in Bild 3.1) verarbeiten zu können. Bei Nutzung von Checksum-Offloading und maximal möglichem Einsatz von Cache-Speicher, kann bei einer Übertragungsrate von 50 MBit/s und einem Prozessor-Takt von 125 MHz die Prozessorauslastung auf bis zu 20 % (Punkt 1 in Bild 3.1) gesenkt werden. Ohne Checksum-Offloading, aber maximaler Cache-Größe, liegt die Prozessorauslastung hier bei 42 % (Punkt 2 in Bild 3.1).
Wird für ein optimiertes System die Prozessor-Taktfrequenz von 125 MHz herabgesetzt, so sind zu 100 MHz keine signifikanten Veränderungen feststellbar, bei der nächstkleineren konfigurierbaren Frequenz von 83.33 MHz steigt die Prozessorauslastung um 5 % mit Checksum-Offloading und um 10 % ohne Checksum-Offloading an. Bei 75 MHz Taktfrequenz wird eine Rechenleistung von insgesamt 27 % mit und 58 % ohne Checksum-Offloading benötigt. Die Auslastung bei verschiedenen Taktfrequenzen ist von großem Interesse, da der MicroBlaze nicht auf allen FPGAs mit einer Frequenz von 125 MHz betrieben werden kann.

Ergebnisse für die Senderichtung

Bild 3.2 zeigt die Ergebnisse für die Senderichtung. Hier existiert keine Anforderung an die zu erreichende Übertragungsgeschwindigkeit, sodass für einen besseren Vergleich auch hier 50 MBit/s (rote Linie in Bild 3.2) als Richtwert angenommen wird. Ohne Einsatz von Cache ist

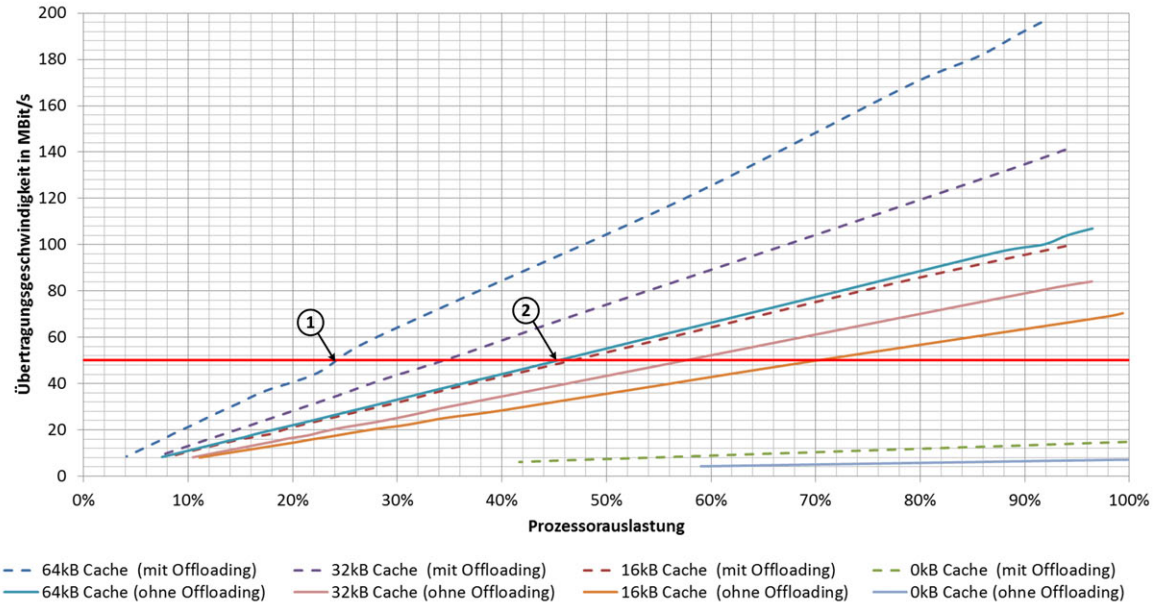


Bild 3.2.: Übertragungsgeschwindigkeit in Senderichtung bei verschiedenen Prozessorauslastungen, verschiedenen Konfigurationen und optionalem Checksum-Offloading bei einem Systemtakt von 125 MHz und 1200 Byte Nutzdaten pro Ethernet-Paket

auch hier das Erreichen der Übertragungsgeschwindigkeit nicht möglich. Die Prozessorauslastung zum Senden von Daten ist vergleichsweise höher, als beim Empfang. Durch Verwendung von Cache-Speicher lässt sich die Prozessorauslastung auf bis zu 24 % (Punkt 1 in Bild 3.2) bei einer Taktfrequenz von 125 MHz und Nutzung des Checksum-Offloading senken. Ohne Checksum-Offloading lässt sich eine minimale Auslastung von 45 % (Punkt 2 in Bild 3.2) erreichen. Auch hier ist zwischen einer Taktung von 100 MHz und 125 MHz keine signifikante Reduzierung der relativen Rechenleistung feststellbar. Bei 83.33 MHz steigt die Prozessoraus-

lastung um 5 % (mit Checksum-Offloading) bzw. um 11 % (ohne Checksum-Offloading) an. Bei 75 MHz steigt die Auslastung auf insgesamt 32 % bzw. 62 % an.

Weitere Ergebnisse

Zusammenfassend ist festzustellen, dass bei Paketgrößen von 1200 Byte im Sendebetrieb bis zu 196 MBit/s übertragen werden können. Im Empfangsbetrieb konnten bis zu 100 MBit/s nachgewiesen werden. Aufgrund der nicht erreichten maximalen Prozessorauslastung ist anzunehmen, dass die Ursache für die geringe Übertragungsgeschwindigkeit in Empfangsrichtung auf der Senderseite liegt. Wird ein, wie bei allen Messungen feststellbarer, näherungsweiser linearer Verlauf angenommen, so würde sich bei einer Prozessorauslastung von 100 % eine Übertragungsgeschwindigkeit beim Empfangen von 250 MBit/s ergeben.
Bei der Nutzung von Jumbo-Frames von 8192 Byte Länge ließe sich die Performance im Sendebetrieb nach [35] auf bis zu 990 MBit/s steigern. In Empfangsrichtung liegen keine Messergebnisse vor.

3.3.3. Untersuchung mit dem Ethernet-MAC “XPS EthernetLite”

Die Untersuchung der erreichbaren Übertragungsgeschwindigkeit und der hierfür benötigten Prozessorauslastung beim “XPS EthernetLite” geschieht auf gleiche Weise wie beim “XPS LL TEMAC”, indem die gleiche Test-Applikation genutzt wird.
Im Gegensatz zum “XPS LL TEMAC” ist der “XPS EthernetLite” ein auf 100 MBit/s beschränkter Ethernet-MAC-Core, der keine DMA-Transfers von Daten zwischen Ethernet-MAC und RAM unterstützt. In Hardware sind maximal jeweils 2 Sende- und Empfangs-Buffer realisiert, die über den PLB-Bus (weiteres siehe Abschnitt 4.2.4) ausgelesen werden können. Für FPGAs die den AXI-Bus (siehe ebenfalls Abschnitt 4.2.4) unterstützen, wäre auch eine Anbindung über diesen möglich. Bei einem Virtex-5 FPGA, wie hier verwendet, ist diese Möglichkeit nicht gegeben. Eine Nutzung von Jumbo-Frames ist nicht möglich. Mit Ausnahme der Nutzung von Cache und der Erhöhung der Taktfrequenz können damit keine Maßnahmen zur Performancesteigerung aufseiten der Hardware ergriffen werden. Aus diesen Grund konnte eine maximale Übertragungsgeschwindigkeit von bis zu 42 MBit/s im Sendebetrieb bei voller Prozessorauslastung und maximalem Cache-Speicher (64 kByte) nachgewiesen werden. Eine Performanceverbesserung zwischen 64 kByte Cache und 32 kByte Cache ist nicht mehr gegeben. Im Vergleich zu 16 kByte Cache lässt sich noch ein Performance-Gewinn von bis zu 16 % nachweisen. Dies deutet daraufhin, dass die fehlende DMA-Funktion und damit die Kopiervorgänge, sowie zu kleine Buffer, die maximale Datenübertragungsrate vorgeben.
Im Empfangsbetrieb konnte lediglich eine absolut fehlerfreie Übertragungsrate von maximal 1.9 MBit/s nachgewiesen werden. Die Prozessorauslastung ist dabei allerdings vernachlässigbar gering, sodass die Ursache nicht in der Rechen-Performance zu suchen ist. Da bereits im Sende-Betrieb die Übertragungsgeschwindigkeit nicht den Anforderungen aus Abschnitt 1.1 entspricht, ist eine Untersuchung der niedrigen Empfangsgeschwindigkeit nicht mehr durchgeführt worden.

3.3.4. Übergreifende Ergebnisse

Aus den Untersuchungen von beiden Ethernet-MACs lassen sich übergreifende Ergebnisse bei der Verwendung des lwIP-Ethernet-Stacks ableiten. Tabelle 3.1 listet die untersuchten Möglichkeiten auf, die Prozessorauslastung unter Verwendung des Ethernet-Stacks "lwIP" zu senken. Die Angaben beziehen sich dabei auf die relative Prozessorauslastung vor der jeweiligen Maßnahme zur Beschleunigung des Systems. Durch die nachgewiesene Linearität können diese Angaben auf jede Übertragungsgeschwindigkeit angewendet und verschiedene Maßnahmen miteinander kombiniert werden.
Die Tabelle 3.1 zeigt, dass der größte Performance-Gewinn durch Nutzung von 16 kByte

Beschreibung	Einsparung TX	RX
Checksum-Offloading (64 kByte Cache)	45 %	52 %
Checksum-Offloading (32 kByte Cache)	41 %	45 %
Checksum-Offloading (16 kByte Cache)	32 %	27 %
Speicherzugriff (64 kByte zu 32 kByte Cache, mit Checksum-Offloading)	27 %	28 %
Speicherzugriff (32 kByte zu 16 kByte Cache, mit Checksum-Offloading)	27 %	30 %
Speicherzugriff (16 kByte zu 0 kByte Cache, mit Checksum-Offloading)	86 %	84 %
Speicherzugriff (64 kByte zu 32 kByte Cache, ohne Checksum-Offloading)	24 %	17 %
Speicherzugriff (32 kByte zu 16 kByte Cache, ohne Checksum-Offloading)	16 %	11 %
Speicherzugriff (16 kByte zu 0 kByte Cache, ohne Checksum-Offloading)	90 %	91 %
DMA Zugriff (64 kByte Cache, ohne Checksum-Offloading)	61 %	-
Systemtakt (75 MHz zu 83.333 MHz)	12 %	10 %
Systemtakt (83.333 MHz zu 125 MHz)	17 %	17 %

Tabelle 3.1.: Reduzierung der Prozessorauslastung bei den verschiedenen Optimierungsmaßnahmen. Die Angaben beziehen sich auf die jeweilige relative Prozessorauslastung und sind näherungsweise auf alle Übertragungsgeschwindigkeiten bei Verwendung von 1200 Byte großen UDP/IP-Nachrichten anwendbar.

Cache-Speicher erreicht wird, gefolgt von der Verwendung einer DMA-Übertragung zwischen Ethernet-MAC und -Stack und dem Checksum-Offloading. Die minimale Prozessorauslastung liegt in Sende-Richtung bei etwa 50 % des Wertes der Übertragungsgeschwindigkeit[11], in Empfangs-Richtung sind dies 40 %. Jeweils bei einem Systemtakt von 125 MHz. Diese Prozessorauslastung kann nur eingespart werden, wenn auch die Protokoll-Verarbeitung in Hardware durchgeführt wird.

3.4. Untersuchung von in Hardware realisierten Lösungen

Während Ethernet-Stacks oft in Software realisiert werden, ist zunehmend auch die Realisierung von Ethernet-Stacks in Hardware interessant. Die Realisierung durch ein ASIC ist aufgrund der NRE-Kosten[12] für kleinere Serienfertigungen uninteressant. FPGAs, die genügend

[11] Beispiel: 50 MBit/s ergibt 25 % der Prozessorauslastung bei einem Systemtakt von 125 MHz
[12] NRE-Kosten steht abkürzend für "Non-recurring engineering"-Kosten und beinhaltet einmalige Entwicklungskosten des einbetteten Systems.

Logik-Ressourcen besitzen und leistungsfähig genug sind, um Ethernet-Daten zu verarbeiten, haben hingegen sinkende Unit-Kosten.
Durch die Nutzung einer in Hardware realisierten Variante können die einzelnen Funktionen und die dafür notwendigen Ressourcen optimal auf die Aufgabe der Kommunikation abgestimmt werden, sodass es zu keinem Ressourcenverlust aufgrund eines für die Kommunikation eher ineffizienten Prozessor-Systems kommt. Zeitverluste, etwa durch Kopiervorgänge, können aufgrund der erreichbaren Verarbeitungsgeschwindigkeit und der Möglichkeit einer nebenläufigigen Verarbeitung vermieden werden. Hierdurch ist eine Realisierung in Hardware geeignet, wenn härtere Echtzeitanforderungen an die Verarbeitung von über Ethernet empfangenen Daten gestellt werden und deren Verarbeitung keinen störenden Einfluss auf die Abarbeitung anderer Aufgaben haben darf.
Oft ist es jedoch sinnvoll und praktikabel die Applikationsebene in Software zu realisieren. Dies stellt keinen Widerspruch dar: Durch die vollständige Protokoll-Verarbeitung in Hardware kann, auch wenn die Daten auf Applikations-Ebene durch Software weiter verarbeitet werden, eine geringere Rechenauslastung des Prozessors, gegenüber einem in Software realisierten Ethernet-Stack, bei gleichem Datendurchsatz, erreicht werden.

3.4.1. Untersuchungskonzept

Es gibt bereits einige Veröffentlichungen [19, 26, 27], die sich mit einer Realisierung von Ethernet-Stacks in Hardware beschäftigt haben. Die Funktionsumfänge dieser Realisierung variieren stark. Keiner der Ethernet-Stacks erfüllt alle die in Kapitel 1.1 gestellten Anforderungen, da kein Ethernet-Stack die Verarbeitung von Multicast-Nachrichten und die Verwaltung von Multicast-Gruppen-Mitgliedschaften unter Verwendung des IGMP-Protokolls unterstützt. Die Entwicklung eines eigenen Ethernet-Stacks wäre in diesem Fall notwendig.
Die Untersuchung legt daher hier den Schwerpunkt auf die Ethernet-MACs. Untersucht werden sollen zwei verschiedene Ethernet-MACs in Verbindung mit dem sehr minimalistischen Ethernet-Stack “UDP/IP Core” [27] von OpenCores. Ziel der Untersuchung ist damit zum einen, die Machbarkeit festzustellen, ob die geforderte Übertragungsrate fehlerfrei und zuverlässig übertragen werden kann. Zum anderen ist festzustellen, welcher Ethernet-MAC am geeignetsten für die Anbindung an einen, in Hardware realisierten, Ethernet-Stack ist und welche Logik- und Speicher-Ressourcen hierfür benötigt werden.
Bei dem für die Voruntersuchungen verwendeten Ethernet-Stack handelt es sich um einen Ethernet-Stack, welcher mit dem Ziel entwickelt wurde, sehr wenig Logik- und Speicherressourcen zu verwenden. Dabei wurde nur das Allernötigste implementiert, um eine Kommunikation zwischen zwei Teilnehmern zu ermöglichen. Alle Protokoll-Informationen sind dabei statisch in Look-Up-Tabellen hinterlegt. Dynamisch können lediglich die Nutzdaten und die Längeninformation eingetragen werden und im Zuge dessen die IP-Checksumme. Um bei Letzterem die Ressourcen noch weiter zu minimieren, wird die Checksumme, auf Basis der statischen Daten, soweit wie möglich vorausberechnet.
Der in Hardware realisierte Ethernet-Stack soll mit folgenden Ethernet-MACs untersucht werden:

- Der Ethernet-MAC “XPS LL TEMAC” kann auch ohne MicroBlaze/PowerPC in einem HDL-Design eingebunden werden und soll daher auch hier untersucht werden.

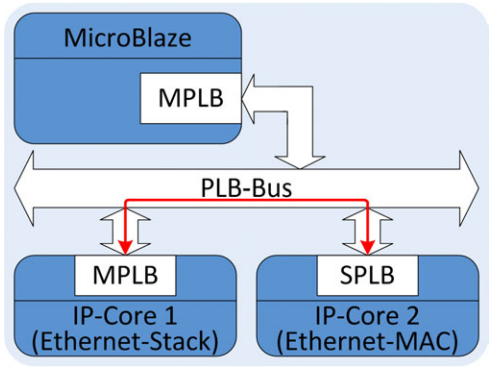


Bild 3.3.: Schematischer Aufbau für mögliche Nutzung des "XPS EthernetLite" in Verbindung mit einem in Hardware realisierten Ethernet-Stack

- Der Ethernet-MAC "XPS Ethernet Lite" steht nur als IP-Core in Verbindung mit der Nutzung des MicroBlaze/PowerPC unter Verwendung des PLB- oder AXI-Busses zur Verfügung und kann nicht als eigenständiger IP-Core in einem reinen HDL-Design eingesetzt werden.
  Es sind Untersuchungen angestellt worden, ob es machbar und praktikabel ist, über einen IP-Core mit PLB-Master-Interface auf einen IP-Core mit PLB-Slave-Interface zuzugreifen. Diese Situation würde sich bei einer Ansteuerung dieses Ethernet-MACs durch einen in Hardware realisierten Ethernet-Stack ergeben, wie das Bild 3.3 veranschaulicht. Dies konnte erfolgreich demonstriert werden. Nach [59] ist die Ansteuerung des Ethernet-MAC relativ einfach gehalten und daher der zu erwartende Steueraufwand gering. Durch den Nachteil, dass dieses Ethernet-Interface damit fest an die Nutzung eines MicroBlaze gekoppelt ist und der "XPS Ethernet Lite" nicht direkt den Empfang von Multicast-Nachrichten unterstützt, wird dieser Ethernet-MAC auf die Anbindbarkeit eines Hardware-Stacks hin nicht untersucht.
  Als Ersatz für den "XPS Ethernet Lite" wird sich für die Untersuchung des EthMAC [23] von OpenCores entschieden. Dieser unterstützt alle benötigten Funktionen. Hierzu zählt die Full-Duplex-Übertragung mit Flow-Control, die Verarbeitung von Broad- und Multicast-Nachrichten und ein Interface zur Konfiguration und Status-Abfrage des PHY-Bausteins. Unterstützt wird ausschließlich das MII-Interface, wodurch die Übertragungsrate auf 100 MBit/s begrenzt ist. Die Anforderungen werden hierdurch jedoch erfüllt. Die zusätzlich umfangreiche Dokumentation, die Fehlerfreiheit, eine bereits kommerzielle Nutzung des Ethernet-MACs durch andere Firmen und die Möglichkeit, die HDL-Beschreibung in Verilog an spezielle Gegebenheiten anpassen zu können, wurde dieser Ethernet-MAC in die Untersuchungen mit einbezogen. Im Gegensatz zum "XPS LL TEMAC" ist eine integrierte DMA-Engine enthalten. Ziel der Untersuchung ist daher auch festzustellen, inwiefern ein Ethernet-MAC mit DMA-Engine für die Anbindung an einen in Hardware realisierten Ethernet-Stack geeignet ist.

3.4.2. XPS LL TEMAC

Für eine hochperformante Anbindung des "XPS LL TEMAC" [58] an z. B. den MicroBlaze hat Xilinx die LocalLink-Verbindung entwickelt (siehe Abschnitt 4.2.4). Diese Verbindung ist ursprünglich dafür gedacht, eine Anbindung über eine SDMA-Engine an den MPMC-

Speichercontroller zu erreichen. Für den Fall, dass der TEMAC an ein Hardware-Design angebunden werden soll, stellt Xilinx für das hier verwendete Virtex-5 FPGA den “Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Wrapper” [60] zur Verfügung. Es handelt sich hierbei um ein Beispiel-Design, welches aus mehreren Wrappern aufgebaut ist, wie dem Bild 3.4 entnommen werden kann. Ein “Block Level Wrapper” instanziert den “Ethernet MAC Wrapper” und

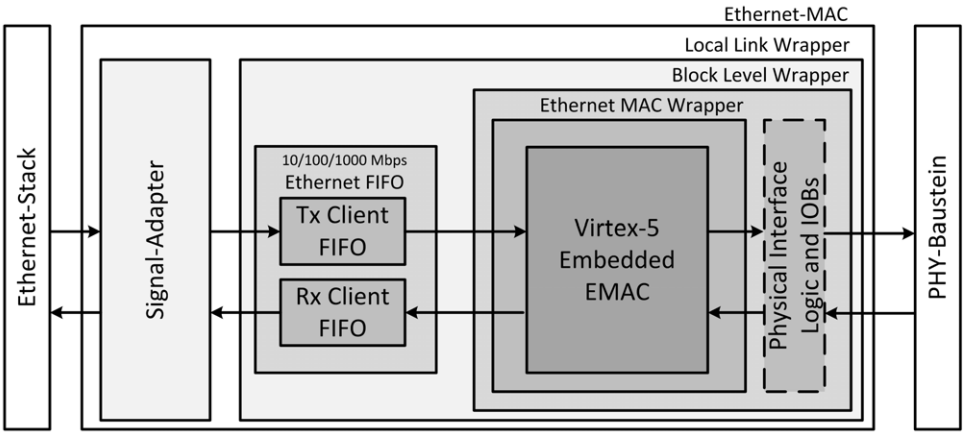


Bild 3.4.: “Virtex-5 FPGA Embedded Tri-Mode Ethernet-MAC Wrapper” [60] mit entwickeltem Wrapper zur Signalanpassung für den “UDP/IP”-Core [27] von OpenCores

bindet damit die Hard-IP-Core-Variante des “XPS LL TEMAC” in das Design ein. Ein “Local Link Wrapper” verbindet dann den LocalLink mit einem LocalLink-FiFo. Zusätzlich existiert der “Example-Design-Wrapper” der eine beispielhafte Nutzung des LokalLink-FiFo zeigt, indem auf MAC-Ebene, d. h. OSI-Layer 2, ein Echo empfangener Nachrichten durchgeführt wird, bei dem Sender- und Absender-MAC-Adresse getauscht werden. Dieser Wrapper ist so umgeschrieben worden, dass eine Kommunikation mit dem verwendeten Ethernet-Stack möglich ist und so ein Echo der Nachrichten auf Applikations-Ebene durchgeführt werden kann.
Das Beispiel-Design unterstützt die Möglichkeit einer Anbindung des Ethernet-MAC an den PHY-Baustein über verschiedene Interfaces. Neben MII und GMII werden auch weitere Interfaces unterstützt. Dadurch waren Anpassungen im Beispiel-Design notwendig, damit dieses unter Verwendung des MII-Interface genutzt werden konnte.
Für die Tests wird auf der Applikationsebene ein in Hardware realisierter Echoserver implementiert, der alle empfangenen Nachrichten unverändert wieder zurücksendet. Werden Nachrichten mittels des Iperf-Verfahren [28] gesendet, kann durch die Auswertung zurückgesendeter Nachrichten nachgewiesen werden, ob Nachrichten fehlerfrei in Sende- und Empfangsrichtung übertragen werden konnten.
Durch vollständige Parallelisierung der Empfangs- und Sendeverarbeitung konnte nachgewiesen werden, dass bei der Nutzung des MII-Interface in Sende- und Empfangsrichtung jeweils 96 MBit/s fehlerfrei übertragen werden können. Der Einsatz eines Prozessors, wie dem MicroBlaze, ist hierzu nicht notwendig.

3.4.3. EthMAC

Der EthMAC ist ein Ethernet-MAC [23] der über zwei sog. Wishbone-Busse[13] und alternative über den dem Wishbone-Bus ähnlichen Avalon-Bus angesteuert werden kann [24, 44]. Das Bild 3.5 zeigt den Aufbau des EthMAC. Der EthMAC besitzt ein Wishbone-Master-Interface, mit dem dieser empfangene Daten mithilfe einer eigenen, integrierten DMA-Engine in einen Speicher schreiben und zu sendende Daten per DMA aus einem Speicher auslesen kann. Verwaltet werden kann die DMA-Verarbeitung durch bis zu 128 Speicher-Deskriptoren je Richtung. Jeder Deskriptor speichert im Wesentlichen die Adresse des zugehörigen Speicherblockes, den Zustand des Speicherblockes und den Zustand einer ggf. enthaltenen Nachricht ab. Die Konfiguration des Ethernet-MACs und des DMA-Interfaces, sowie Statusabfragen, geschehen über einen zweiten, unabhängigen Wishbone-Bus, für den der EthMAC ein Slave-Interface besitzt. Ferner existieren verschiedene Interruptleitungen, die über Statusänderungen informieren. Hierzu zählt u. a. der Empfang neuer Nachrichten, das erfolgreiche Senden von Nachrichten und das Auftreten von Fehlern.
Um die Untersuchung des EthMAC durchführen zu können, ist für die Anbindung an den in Hardware realisierten Ethernet-Stack ein Wrapper geschrieben worden, der ebenfalls in Bild 3.5 dargestellt wird. Dieser setzt das Schnittstellenverhalten so um, dass das Verhalten äquivalent dem entwickelten/angepassten Wrapper des “XPS LL TEMAC” aus Abschnitt 3.4.2 ist.
Der Wrapper beinhaltet hierfür zwei Abstraktionsebenen (siehe Bild 3.5). Die erste Ebene wer-

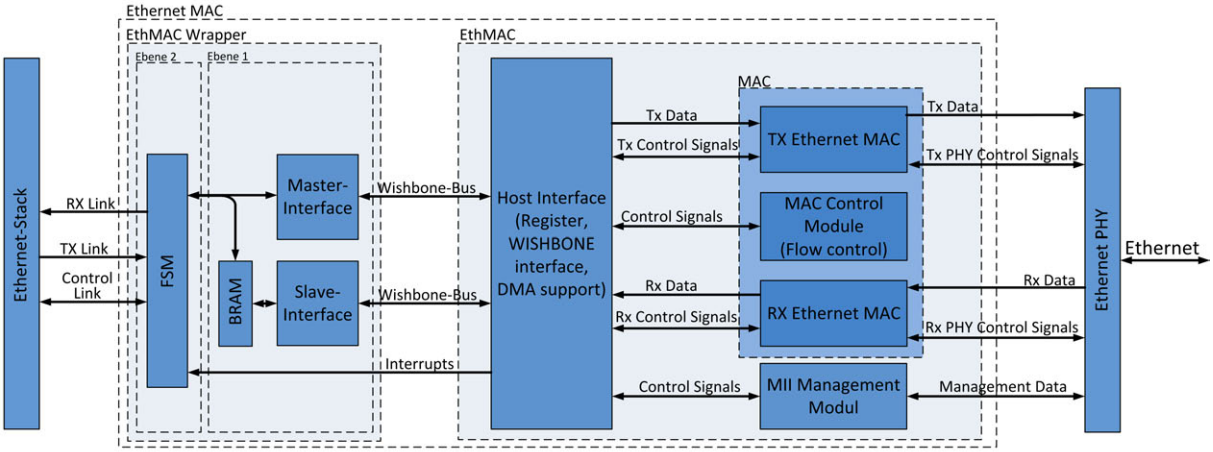


Bild 3.5.: “EthMAC” [23] von OpenCores mit entwickeltem Wrapper zur Anbindung an den “UDP/IP”-Core [27] von OpenCores

den beide Wishbone-Busse auf ein einziges Parallel-Interface umgesetzt. Hierzu ist in dieser Ebene ein Dual-Port-RAM[14] und ein hierzu passendes Wishbone-Slave-Interface implementiert, durch welches der EthMAC auf diesen Speicher lesend und schreibend zugreifen kann. Zugriffe durch die zweite Ebene über das Parallel-Interface werden für zwei Adressbereiche verschieden behandelt. Während der eine Adressbereich ebenfalls einen lesenden wie schreibenden Zugriff auf das Dual-Port-RAM ermöglicht, werden Zugriffe auf den zweiten Adressbereich auf ein Wishbone-Master-Interface umgesetzt, sodass Zugriffe auf interne Register des

[13] Der Wishbone-Bus ist ein OpenSource Bus, der u. a. in vielen Designs von OpenCores verwendet wird, um IP-Cores an CPUs anzubinden.
[14] Bei einem Dual-Port-RAM kann nebenläufig zwei Lese- und Schreibzugriffe bei verschiedenen Taktfrequenzen erfolgen.

EthMAC möglich werden.
Die zweite Ebene beinhaltet umfangreichere Zustandsautomaten, die in einem ersten Schritt nach einem Power-Up den EthMAC und die Buffer-Deskriptoren der integrierten DMA-Engine initialisieren. Danach setzen die Zustandsautomaten bidirektional die Datenverarbeitung zwischen der Schnittstelle zum Ethernet-Stack und dem Parallel-Interface zur ersten Ebene des HDL-Wrappers um:

- In Empfangsrichtung ist hierzu auf eintreffende Interrupts zu reagieren, zugehörige Zustandsabfragen durchzuführen und anschließend die empfangenen Daten aus dem richtigen Speicherblock auszulesen. Serialisiert sind diese Daten an den Ethernet-Stack zu übergeben. Nach dem Auslesen ist der Speicherblock wieder freizugeben.
- In Senderichtung müssen die Daten vom Ethernet-Stack in dem richtigen Speicherblock abgelegt werden, die Deskriptoren zum Senden konfiguriert und der nach dem Senden erfolgende Interrupt, zur Bestätigung des Versandes, quittiert werden.

Eine Minderung der Leistungsfähigkeit entsteht dadurch, dass Sende- und Empfangspfad über ein gemeinsames Wishbone-Interface geführt werden müssen und damit nicht mehr nebenläufig verarbeitet werden können. Es besteht hierdurch ein zusätzlicher Synchronisationsaufwand zwischen Sende- und Empfangspfad und die Notwendigkeit, Daten mit möglichst hoher Frequenz in der Applikation, sowie im Ethernet-Stack zu verarbeiten, damit Verzögerungszeiten in den beiden Kommunikations-Pfaden möglichst gering gehalten werden können. Ist die Verarbeitung zu langsam, kann es zu Überläufen des Buffers kommen. Dieser und weitere Fehler verursachen Interrupts, die ebenfalls zu quittieren sind und z. B. eine Zurücksetzung der Bufferdiskriptoren erfordern. Problematisch an dieser Lösung ist daher, dass verschiedenste Ereignisse in unterschiedlichsten Abfolgen zueinander auftreten können und alle Ereignisse einer Bearbeitung bedürfen. Wird die Abarbeitung nicht oder inkorrekt durchgeführt, kann dies zum Deadlock des Gesamtsystems führen.
Die Implementierung dieser Lösung hat damit Vor- und Nachteile aufgezeigt:

- Nachteile:
  Durch den Einsatz des Wishbone-Busses, der DMA-Engine und die dadurch sehr aufwendige Steuerung im notwendigen Wrapper müssen vergleichsweise viele Logik-Ressourcen eingesetzt werden. Durch die sequenzielle Verarbeitung von Sende- und Empfangsdaten sind zur Zwischenspeicherung zusätzliche Speicher-Ressourcen im Wrapper, wie auch im Ethernet-MAC selbst notwendig, auch wenn das FPGA prinzipiell in der Lage wäre, die Daten schnell genug zu verarbeiten. Hinzu kommt, dass der Nachweis über eine Deadlock-freie Verarbeitung unter allen Umständen unter diesen Bedingungen nur schwer zu erbringen ist.
- Vorteile:
  Vorteil dieser Lösung ist, dass der Ethernet-MAC selbst nicht verändert werden muss und daher keine Einarbeitung in die Funktionsweisen des Ethernet-MAC erfolgen muss.

Die Probleme der Ansteuerung konnten für den Zweck der Voruntersuchung ausreichend gelöst werden, sodass die Durchführung von Geschwindigkeitsmessungen möglich wurde. Die in der Voruntersuchung gewählte Konfiguration von einem Sende-Speicher für ein Datenpaket und einem Empfangs-Speicher für zwei Datenpakete erlaubt eine maximale, fehlerfreie Sendegeschwindigkeit von 87.63 MBit/s und eine maximale Empfangsgeschwindigkeit von 98 MBit/s. Für den Empfang ist die Nutzung von zwei Speicherblöcken unerlässlich, da die Daten nicht

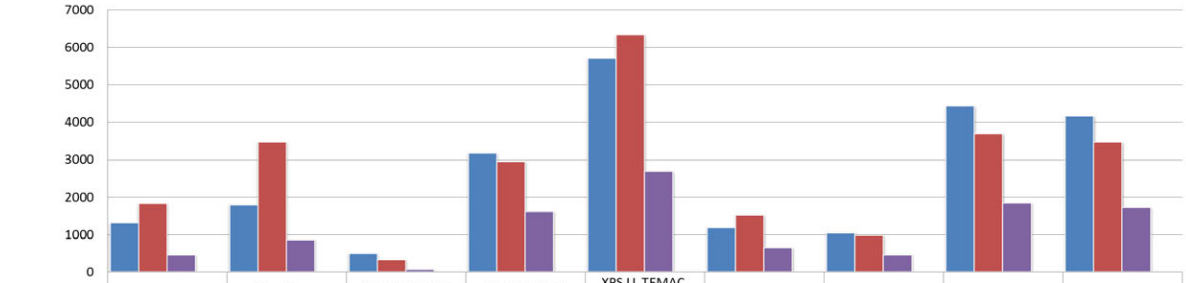
so schnell abgeholt werden können, wie ein neues Ethernet-Paket empfangen werden könnte. Beim Sendevorgang ist dies weniger entscheidend, wirkt sich aber dennoch auf die Übertragungsgeschwindigkeit aus. Die Übertragungsgeschwindigkeit könnte dadurch prinzipiell höher liegen, wenn zwei Speicherblöcke zur Verfügung stehen.
Da der Ethernet-Stack und der Ethernet-MAC hier nur sequenziell arbeiten können, entstehen Wartezeiten. Die Geschwindigkeit bei einer Full-Duplex-Kommunikation hängt daher ausschließlich davon ab, wie schnell die Daten in den Sende- und Empfangs-Speichern im Zeitmultiplex beschrieben und ausgelesen werden können. Es ergeben sich dadurch nicht eindeutig vorhersagbare Latenzen im Sende- und Empfangspfad.

3.5. Vergleich von Lösungen zur Anbindung eines FPGAs an ein Ethernet

Neben Kriterien wie die Rechenauslastung bei einer zu erreichenden Übertragungsgeschwindigkeit, dem deterministischen Verhalten der Verarbeitung oder auch die Latenz der Datenverarbeitung, stellt der Ressourcen-Verbrauch ebenfalls ein wichtiges Kriterium dar. Im Folgenden wird daher ein Vergleich der Ressourcen unterschiedlicher Realisierungsmöglichkeiten für ein Ethernet-Interface vorgenommen. In einem ersten Schritt erfolgt der Vergleich untersuchter und weiterer Ethernet-MACs. Anschließend werden Ressourcenverbräuche von existierenden Hardware-Ethernet-Stacks in Abhängigkeit des jeweiligen Funktionsumfangs verglichen. Nachfolgend erfolgt eine Zusammenstellung verschiedener Lösungsmöglichkeiten für die vollständige Implementierung eines Ethernet-Interfaces.

3.5.1. Ethernet-MACs

Bild 3.6 zeigt die Ressourcenauslastung [8] von den in Abschnitt 3.3 und 3.4 untersuchten Ethernet-MACs. Zusätzlich wurde der nicht untersuchte “Tri-Mode Ethernet-MAC” [25] von



	OpenCores EthMAC	OpenCores EthMAC mit Wrapper	XPS LL TEMAC (Hard-IP) + Wrapper	XPS LL TEMAC (Soft-IP) + Wrapper	XPS LL TEMAC (Soft-IP+ Checksum-Offloading)	OpenCores Tri-Mode MAC	ComBlock (COM_5401)	CAST MAC-1G (10/100/1000)	CAST 10/100 MAC
Slice Registers	1311	1800	507	3177	5709	1198	1055	4437,6	4164
Slice LUTs	1830	3476	326	2960	6337	1526	978	3698	3470
Slices	458	869	80	1624	2689	646,92	465	1849	1735
BRAM	4	8	4	7	36	4	2	3	3

Bild 3.6.: Ressourcen-Verbrauch verschiedener Ethernet-MAC-Cores

OpenCores, sowie drei weitere käuflich erwerbbare Ethernet-MAC-Cores [11, 12, 13] in den Ressourcenvergleich mit aufgenommen. Alle Ethernet-MACs unterstützen die Anforderung,

eine 100 MBit/s-Full-Duplex-Übertragungen zu verarbeiten. Bis auf den "Tri-Mode Ethernet-MAC" und dem Ethernet-MAC von ComBlock unterstützen alle eine konfigurierbare Filterung von Multicast-Nachrichten. Die Ethernet-MACs von CAST[15] und der EthMAC von OpenCores besitzen integrierte DMA-Engines, wodurch ein zusätzlicher Ressourcenverbrauch entsteht. Der "Tri-Mode Ethernet-MAC" von OpenCores, der Ethernet-MAC von ComBlock und der "XPS LL TEMAC" beinhalten hingegen keine DMA-Engine. Letzterer kann jedoch in Verbindung mit einer DMA-Engine betrieben werden. Wie die Untersuchungen gezeigt haben, ist die DMA-Engine für die Verwendung softwarebasierter Ethernet-Stacks zur Minderung der benötigten Rechenauslastung sehr wichtig, bei der Verwendung hardwarebasierter Ethernet-Stacks jedoch mit deutlich höherem Logikaufwand für den HDL-Wrapper und den Ethernet-MAC selbst, verbunden.
Werden die Ressourcenverbräuche betrachtet, ist im Bild 3.6 zu sehen, dass durch den in Abschnitt 3.4.3 entwickelten Wrapper für den EthMAC der Verbrauch an Logik- und Speicherressourcen deutlich ansteigt. Der Aufwand für den Wrapper beim "XPS LL TEMAC" ist hingegen deutlich geringer, wie an der Hard-IP-Core-Variante gesehen werden kann. Es zeigt sich, dass der Ressourcenverbrauch in der Soft-IP-Core-Variante des "XPS LL TEMAC" relativ hoch ist. Aufgeführt ist eine Variante ohne Unterstüzung und eine Variante mit Unterstützung des Checksum-Offloading. Die Ethernet-MACs von CAST liegen aufgrund der integrierten DMA-Engine ebenfalls relativ hoch.

3.5.2. Ethernet-Stacks

In Hardware realisierte Ethernet-Stacks, die bisher veröffentlicht wurden, sind oft auf spezielle Anwendungsfälle zugeschnitten. Sie unterscheiden sich oft deutlich im Funktionsumfang. Keiner der in Hardware realisierten verfügbaren Ethernet-Stacks unterstützt das IGMP-Protokoll um An- und Abmeldungen an Multicast-Gruppen zu verarbeiten. Dies gilt auch für die Unterstützung der Möglichkeit, UDP-Nachrichten zu verarbeiten, die nicht nur an die eigene IP-Adresse gesendet wurden. Dies ist für den Empfang von Multicast-Nachrichten jedoch erforderlich.
Ein Vergleich von Hardware-Ethernet-Stacks wird daher durchgeführt, um abschätzen zu können, wie hoch der Ressourcenverbrauch sein wird, wenn eine eigene auf die Anforderungen angepasste Ethernet-Stack-Lösung entwickelt werden müsste. Tabelle 3.2 listet hierfür die Referenzbeispiele auf, die im Folgenden vorgestellt werden:

- Der "UDP/IP Core" von Opencores [27], der auch für die Voruntersuchungen verwendet wurde, ist vor dem Hintergrund entwickelt, zu zeigen, wie gering der Ressourcenverbrauch eines Ethernet-Stack sein kann (siehe Abschnitt 3.4.1). Der Ethernet-Stack eignet sich daher nur für eine nicht dynamisch konfigurierbare Punkt-zu-Punkt-Verbindung unter Verwendung des UDP und IP Protokolls.
- Der "OpenCores 1G UDP/IP Stack" ist nach den Voruntersuchungen veröffentlicht worden und befindet sich derweil noch in der Entwicklungsphase. Unterstützt werden die Protokolle IP, UDP und ARP. Bei Letzterem ist die Verwaltung auf eine Verbindung beschränkt. Ein Alleinstellungsmerkmal dieses Ethernet-Stacks ist, dass keine Verzögerungen der Nutz-Daten zwischen Ein- und Ausgang existieren sollen.

[15] CAST Inc. bietet auch Ethernet-MACs ohne DMA-Engine an.

- In einer Veröffentlichung von A. Löfgren u. w. sind drei weitere Ethernet-Stacks vorgestellt wurden [19]. Diese nicht quelloffenen Stacks sind mit dem Ziel erstellt, eine Einschätzung zu erlauben, wie viele Ressourcen für welchen Funktionsumfang benötigt werden. Während die kleinste Variante nur die Protokolle UDP und IP, kein Full-Duplex und nur eine begrenzte Nachrichtenlänge verarbeiten kann, kann die mittlere Variante zusätzlich die Protokolle ARP und ICMP, sowie eingeschränkt eine Full-Duplex-Übertragung unterstützen. Die "Advanced"-Variante weist die höchste Flexibilität auf, unterstützt UDP, IP, ARP und ICMP bei einer Full-Duplex-Übertragung und erlaubt zudem die Anbindung an ein Gigabit-Ethernet. Ferner beinhaltet diese Lösung einen "TCP-Channel"[17]. Für die Ethernet-Stacks von A. Löfgren u. w. existieren keine Angaben für die Spartan-6-Familie. Nach Abschnitt 2.1 ist anzunehmen, dass die benötigten Slices gegenüber einem Spartan-3 etwa bei der Hälfte liegen werden, wenn diese mit einem Spartan-6 zu vergleichen sind.
- In der "Advanced" Kategorie ist auch der Stack von ComBlock (COM-5402SOFT [14]) einzusortieren, der noch über eine TOE[18] verfügt.

Ressource	OpenCores UDP/IP Core	OpenCores 1G UDP/IP Stack	Löfgren "Minimum"	Löfgren "Medium"	Löfgren "Advanced"	ComBlock (COM-5402SOFT)
Occupied Slices	142	504	512	1022	1584	807[18]
f_{max} [MHz]	131.8	157.3	90.7	60.3	105.6	145
UDP	Ja	Ja	Ja	Ja	Ja	Ja
IP	Ja	Ja	Ja	Ja	Ja	Ja
ARP	Nein	Ja (1)	Nein	Ja (4)	Ja (4)	Ja (128)
ICMP	Nein	Nein	Nein	Ja	Ja	Ja
IGMP	Nein	Nein	Nein	Nein	Nein	Nein
FullDuplex	Ja	Ja	Ja	Ja	Ja	Ja
TCP-Channel	Nein	Nein	Nein	Nein	Ja	Ja
Bezugs-FPGA	Virtex-5	Spartan-6	Spartan-3	Spartan-3	Spartan-3	Spartan-6

Tabelle 3.2.: Vergleich verschiedener hardwarebasierter Ethernet-Stacks

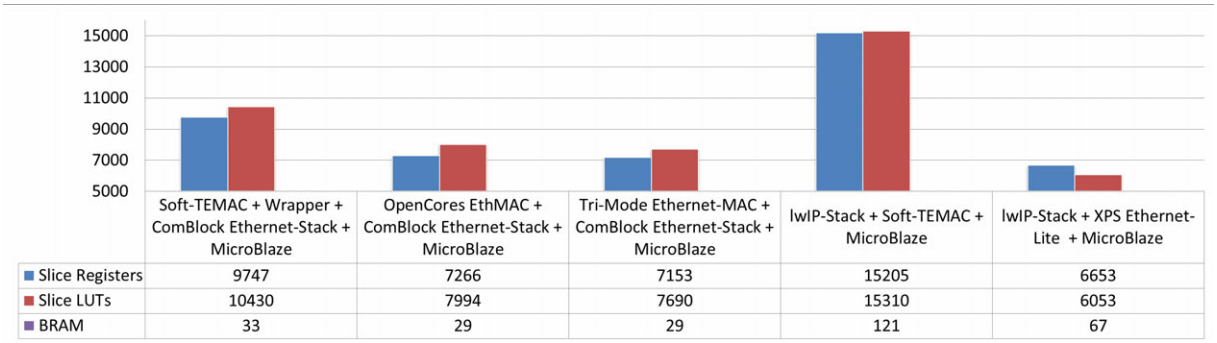
[17] Mit einem TCP-Channel wird nicht die Verarbeitung des TCP-Protokolls in Hardware verstanden, sondern die Möglichkeit TCP-Nachrichten z. B. an einen Mikrocontroller weiterzugeben, damit dieser diese verarbeiten kann.

[18] TOE: TCP Offload Engine, dient dazu die CPU von rechenintensiven Aufgaben bei der Verarbeitung der Protokolle TCP und IP zu entlasten, etwa durch ein Checksum-Offloading

[18] Nach [14] liegt die Anzahl an Slice-Register bei 1938 und die Slice-LUTs bei 2815. Die Anzahl der "Occupied Slices" wurde hieraus mit 807 geschätzt (bei 50 % Auslastung der Slice-Register und 30 % Auslastung der Slice-LUTs in einem Slice und damit wie beim "OpenCores 1G UDP/IP Stack" [26]).

3.5.3. Ethernet-Interface

Bild 3.7 zeigt eine Zusammenstellung möglicher Gesamtlösungen für die Anbindung eines FPGAs an ein Ethernet-Netzwerk. Hierbei handelt es sich um zum Teil errechnete Ressourcen-Verbräuche aus den Ressourcen-Verbräuchen der einzelnen Komponenten und nicht um die exakten Ergebnisse einer Implementierung der Designs. Da der Vergleich mittels Slice-Register und Slice-LUTs durchgeführt wird und nicht auf der Anzahl der "Occupied Slices" basiert, ist der hierbei entstehende Fehler als gering anzunehmen. Alle Werte beziehen sich dabei auf eine Implementierung unter Nutzung eines Spartan-6 FPGAs. Um einen sinnvollen Vergleich zwischen Lösungen mit hardwarebasiertem Ethernet-Stack und softwarebasiertem Ethernet-Stack zu ermöglichen, werden zu Ersterem die Ressourcenverbräuche eines MicroBlazes hinzuaddiert, sodass alle Lösungen eine Realisierung der Applikations-Ebene in Software erlauben würden.
Die ersten drei in Bild 3.7 dargestellten Lösungen basieren auf einem in Hardware realisierten



	Soft-TEMAC + Wrapper + ComBlock Ethernet-Stack + MicroBlaze	OpenCores EthMAC + ComBlock Ethernet-Stack + MicroBlaze	Tri-Mode Ethernet-MAC + ComBlock Ethernet-Stack + MicroBlaze	lwIP-Stack + Soft-TEMAC + MicroBlaze	lwIP-Stack + XPS Ethernet-Lite + MicroBlaze
Slice Registers	9747	7266	7153	15205	6653
Slice LUTs	10430	7994	7690	15310	6053
BRAM	33	29	29	121	67

Bild 3.7.: Ressourcenvergleich von Gesamtlösungen zur Anbindung eines Spartan-6 FPGAs an ein Ethernet (mit angepasster y-Achsen-Darstellung)

Ethernet-Stack. Verwendet wurde hier der Ethernet-Stack von ComBlock [14]. Nach Tabelle 3.2 stellt dieser einen ähnlichen Funktionsumfang zu einem dann zu entwickelnden Ethernet-Stack dar. Anstelle der TCP-Unterstützung wäre eine Unterstützung des IGMP-Protokolls notwendig. In der ersten Lösung wird dieser Ethernet-Stack in Verbindung mit der Soft-IP-Core-Variante des "XPS LL TEMAC" betrachtet. Die Hard-IP-Core-Variante wird nicht berücksichtigt, da in Abschnitt 1.1 eine Realisierbarkeit auf einem FPGA der Spartan-Familie gefordert ist, die diesen Ethernet-MAC nicht als Hard-IP-Core-Variante integriert haben. Die zweite Variante verwendet den in Abschnitt 3.4.2 untersuchten EthMAC[19], dessen Datendurchsatz auf 100 MBit/s beschränkt ist. Die dritte Lösung stellt die gleiche Lösung, nur mit dem "Tri-Mode Ethernet-MAC" dar, welche damit auch 1 GBit/s verarbeiten kann. Der Ressourcenverbrauch dieser Lösung ist gegenüber der vorherigen Lösung mit dem EthMAC geringer, da hier keine DMA-Engine integriert ist.
Die beiden letzten Varianten stellen eine Realisierung unter Verwendung des in Software realisierten lwIP-Stacks dar. Im Gegensatz zu der Voruntersuchung wird auch hier mit der Soft-IP-Core-Variante des "XPS LL TEMAC" gerechnet. Anders als bei der ersten Lösung in Bild 3.7 muss hier auch der Ressourcenverbrauch berücksichtigt werden, der durch die Checksum-Offloading-Funktion entsteht. Der Vollständigkeit wegen wird der Ressourcenverbrauch der untersuchten Variante mit dem Ethernet-MAC "XPS Ethernet-Lite" ebenfalls aufgelistet.

[19] In Bild 3.7 worden die Ressourcen des EthMAC ohne Wrapper verwendet, es besteht hier Potenzial für Ressourcen-Optimierung.

KAPITEL 4

Konzept

Das Kapitel “Konzept” stellt unterschiedliche Realisierungsmöglichkeiten vor, bewertet diese und entwickelt hieraus ein Konzept zur Realisierung des Prototypen. Die Konzeptionierung basiert dabei auf den in Kapitel 1.1 angegebenen funktionalen und nicht funktionalen Anforderungen und auf den Ergebnissen der Voruntersuchung in Kapitel 3.

4.1. Allgemeines

Das im Folgenden erarbeitete Konzept wird vor dem Hintergrund entwickelt, eine Lösung zu finden, die bei minimalem Ressourcenverbrauch die spezifizierten Anforderungen optimal erfüllt. Um dies zu erreichen, ist eine Partitionierung der verschiedenen Aufgaben auf in Hardware realisierte Komponenten und auf in Software realisierte Komponenten oft entscheidend. Für Hardware prädestinierte System-Komponenten sind i. d. R. Performance-kritische Komponenten. Die Realisierung dieser Komponenten führt oft zu höheren Kosten durch höhere Entwicklungszeiten und können, je nach Anwendungsfall, auch zu höheren Unit-Kosten führen, wenn ein Prozessor die Grundvoraussetzung für eine Realisierung ist. In Software lassen sich hingegen weniger Performance-kritische Komponenten realisieren. Die Realisierungskosten sind i. d. R. geringer [30].
Bei einer Radar-Anlage gibt es diverse Betriebsmodi, die unterschiedlichste Anforderungen an die benötigte Rechenleistung stellen. Größen, wie die Anzahl der zu erzeugenden Azimuth-Clock-Pulse pro Antennenumdrehung, die Drehgeschwindigkeit der Antenne, die Puls-Folge-Frequenz des Triggers und damit die Frequenz mit der Radar-Sweeps ausgegeben werden oder auch die Anzahl an DAC-Samples pro Sweep können je nach Systemkonfiguration und Betriebsmodus des Radar-Transceivers unterschiedlich hoch sein und in unterschiedlichen Kombinationen zueinander auftreten. Ein Lösungskonzept muss daher sicherstellen können, dass nicht nur genug Rechenzeit für alle Einzelaufgaben zur Verfügung steht, sondern auch die Auswirkungen der Verarbeitungen untereinander nicht derart groß werden, dass die zulässigen Toleranzen erreicht bzw. überschritten werden. Dies könnte z. B. die Zunahme des zeitlichen Jitterns der Synchronisationspulse um den idealen Ausgabezeitpunkt sein, sobald sich der Datendurchsatz von Ethernet-Nachrichten erhöht. Ist das Jittern zu hoch, kann dies zu Verzerrungen und Sprüngen von Radar-Zielen auf dem Bildschirm des Sichtgerätes führen.
Wird eine reine in Software realisierte Lösung durch Einsatz eines Mikrocontrollers (bei einer CPU) verwendet, führt dies zu einer sequenziellen Abarbeitung dieser Aufgaben. Damit ist das Risiko, dass sich die Teilaufgaben unzulässig stark aufeinander auswirken höher, als wenn eine Auslagerung einzelner Performance-kritischer Aufgaben in Hardware durchgeführt oder eine zweite CPU eingesetzt wird. Eine ideale Plattform für ein Hardware-Software-Codesign stellen

FPGAs u. a. der Firma Xilinx dar. Hier besteht die Möglichkeit, Mikrocontroller, wie den MicroBlaze, im FPGA zu integrieren und innerhalb desselben Chips eine performante Anbindung an IP-Cores (engl. intellectual property core) zu ermöglichen, die performance-kritische Abarbeitungen durchführen können. Die Prozessorauslastung kann dadurch gesenkt und das Risiko unzulässig starker Auswirkungen der einzelnen Aufgaben aufeinander minimiert werden.

4.2. Partitionierung der Aufgaben

Bei der Partitionierung von Aufgaben wird festgelegt, wie die Aufgaben realisiert werden sollen und welche Ressourcen hierfür zur Verfügung zu stellen sind.

4.2.1. Generierung der Synchronisationspulse

Synchronisationspulse sind winkelstabil zu generierende Pulse. Sichtgeräte, die diese Pulse auswerten, setzen diese Winkelstabilität voraus. Nur so kann durch Zählen der ACP-Pulse nach einem ARP-Puls und durch Messung derer Zeitabstände die exakte Richtung fehlerfrei bestimmt werden, aus der die Echos empfangen wurden. Ein zeitlicher Jitter der Synchronisationspulse oder des Triggers bedeutet damit immer einen Winkelfehler in der azimutalen Zuordnung der Echos. Ist der Jitter zu groß, springen Echos wahrnehmbar in ihrer Richtung.
Nach Abschnitt 1.1 beträgt die maximal zu verarbeitenden Antennendrehgeschwindigkeit 80 Umdrehungen pro Minute bei einer Pulsfolgefrequenz von bis zu 1500 Hz und bis zu 8192 zu generierenden ACP-Pulse. Für eine fehlerfreie Rekonstruktion darf der ACP-Puls maximal um die halbe Periodendauer jittern. Dies entspricht einem Einzelwinkelfehler von $\pm 0.022°$, wenn 8192 ACP-Pulse pro Umlauf zu generieren sind. Umgerechnet in einen zeitlichen Jitter des ACP-Puls ergibt sich bei einer Antennendrehgeschwindigkeit von 80 RPM ein maximal zulässiger Jitter von $\pm 45.78\,\mu s$. Akkumulieren sich die Winkelfehler während eines Umlaufes, so darf der maximale Winkelfehler der halben Winkelbreite der Hauptkeule der Radarantenne betragen und damit $\pm 0.5°$ nicht überschreiten.
Die Berechnung der Puls-Ausgabe-Zeitpunkte kann am geeignetsten mittels Software auf einem Mikrocontroller durchgeführt werden. Dieser ist prinzipiell in der Lage, Pulse mit einem Jitter unter $\pm 45.78\,\mu s$ zu errechnen und zu generieren. Notwendig hierfür wären drei Timer (für Trigger, ACP und ARP), die nach Ablauf der berechneten Zeiten Interrupts generieren und damit eine Pulserzeugung auslösen. Eine Realisierung in Software bringt ein hohes Maß an Skalierbarkeit mit sich: Die Anzahl an ACP-Pulsen pro Umlauf muss variabel konfigurierbar sein und der ARP muss wahlweise als Head- oder Northmarker genierbar sein. Beides soll über Ethernet von einem anderen Netzwerkteilnehmer konfiguriert werden können. Bei einer Realisierung in Software müssen dem MicroBlaze für die Berechnung und Generierung der Pulse die Headerinformationen des proprietären Video-Protokolls zur Verfügung stehen.
Eine Realisierung in Hardware ist zwar prinzipiell möglich, die notwendige Entwicklungszeit zur Implementierung der mathematischen Berechnungen ist jedoch deutlich höher anzusetzen, bei geringerer Flexibilität bei nachträglichen Anpassungen.

4.2.2. Ethernet-Anbindung

In Kapitel 3 sind verschiedenste Möglichkeiten untersucht worden, ein FPGA an ein Ethernet anzubinden. Die Ergebnisse dieser Untersuchungen sollen hier bewertet werden, sodass eine Entscheidung über die zu wählende Realisierung getroffen werden kann.
Die Untersuchungen haben gezeigt, dass alle Lösungen, mit Ausnahme der auf dem "XPS-EthernetLite" basierten Variante, die geforderte Übertragungsgeschwindigkeit erreichen können. Tabelle 4.1 stellt die wichtigsten Schlüssel-Faktoren der möglichen, untersuchten Varianten gegenüber, die im Folgenden bewertet werden.

Kriterium	Stack:	HW-Ethernet-Stack		SW-Ethernet-Stack
	MAC:	Soft-TEMAC	EthMAC 10/100	Soft-TEMAC
Ressourcenverbrauch	LUTs	10430	7994	15310
	Register	7266	6598	15205
	BRAM	33	29	121
Übertragungs-	RX	96.00 MBit/s	98.00 MBit/s	100 MBit/s
geschwindigkeit (max.)	TX	96.00 MBit/s	87.63 MBit/s	190 MBit/s
Prozessorauslastung	RX	0 %	0 %	20 %
(min.) bei 50 MBit/s	TX	0 %	0 %	24 %
Flexibilität		mittel	mittel	hoch
Realisierungsrisiko		gering	mittel	hoch
Entwicklungszeit		mittel	hoch	gering
Kosten		hoch	gering	hoch

Tabelle 4.1.: Vergleich verschiedener Ethernet-Stack-Lösungen

Ressourcenverbrauch:

Bei allen in Tabelle 4.1 aufgeführten Varianten wird der Ressourcenverbrauch des MicroBlaze berücksichtigt, der nach Abschnitt 4.2.1 für die Pulsgenerierung eingesetzt werden soll. Dabei ergibt sich der geringste, zu erwartende Ressourcenverbrauch bei einer Implementierung des Ethernet-Stacks in Hardware unter Einsatz des EthMAC. Am meisten Ressourcen benötigt die Variante mit dem in Software realisierten lwIP-Stack unter Verwendung der Soft-IP-Core-Variante des "XPS LL TEMAC". Diese Systemkonfiguration benötigt deutlich mehr Logik- und Speicherressourcen als die ebenfalls mit dem "XPS LL TEMAC" realisierte Variante unter Verwendung des Hardware-Ethernet-Stacks. Die Ursache liegt in der benötigten Checksum-Offloading-Funktion und in der Notwendigkeit externen Speicher zu nutzen. Letzteres erfordert die Nutzung des Speichercontroller MPMC mit DMA-Engine und Cache-Speicher.
Je mehr Ressourcen vom Design benötigt werden, desto höher werden die späteren Unit-Kosten sein.

Prozessorauslastung/Realisierungsrisiko:

Neben dem geringeren Ressourcenverbrauch haben die Lösungen mit in Hardware rea-

lisiertem Ethernet-Stack den Vorteil, dass keine Rechenzeit durch die Verarbeitung der Aufgaben des Ethernet-Stacks entsteht. Dies wirkt sich positiv auf das Realisierungsrisiko aus. Eine Realisierung in Software hat hingegen ein höheres Realisierungsrisiko: Interrupt-Routinen können auf dem MicroBlaze nicht durch Interrupt-Requests höherer Priorität unterbrochen werden[1]. Dies kann aufgrund der Anforderungen an einen maximal erlaubten Jitter der Radar-Pulse problematisch werden, die ebenfalls interruptbasiert berechnet und generiert werden sollen. Ein Jitter der Pulse ergibt sich demnach für den Worst-Case-Fall aus der Rechenzeit zur Verarbeitung eines Ethernet-Paketes mit der darauffolgenden Interpretation und Verarbeitung des proprietären Video-Protokolls, sowie der Abarbeitungszeit der Interrupt-Handler von bis zu zwei der drei Pulse, bevor der Interrupt-Handler des dritten Pulses abgearbeitet werden kann. Diese Gesamtverzögerung darf nach Abschnitt 4.2.1 eine Zeit von $45.78\,\mu s$ nicht überschreiten. Es steht zu erwarten, dass der größte nicht unterbrechbare Verarbeitungsblock bei der Verarbeitung der Ethernet-Pakete entsteht. Ob die Gesamtverzögerung gering genug ist, müsste vorab genauer untersucht werden.

Die maximale Taktfrequenz des MicroBlaze beim Spartan-3-FPGA beträgt 75 MHz und bei einem Spartan-6 FPGA 83.3 MHz bei Verwendung des PLB-Bus bzw. 100 MHz bei Verwendung des AXI-Bus. Es ist zu beachten, dass diese Taktfrequenzen absolute Maxima darstellen. Der "Base System Builder"[2] warnt explizit, dass ggf. Timing-Constraints nicht eingehalten werden können, wenn der dem MicroBlaze zur Verfügung gestellte lokale BlockRAM für die Taktfrequenz zu groß ist. Aus diesem Grund wird eine Taktfrequenz von 75 MHz angesetzt. Die Voruntersuchungen haben gezeigt, dass bei einer Software-basierten Realisierung des Ethernet-Interfaces die Prozessorauslastung bei dieser Taktfrequenz auf bis zu 27 % gesenkt werden kann. Damit verbleiben 73 % der Rechenleistung für die Berechnung und Ausgabe der Pulse, die Interpretation des Video-Protokolls und eine Übertragung der Video-Samples zurück in die Hardware. Letzteres kann dabei mittels DMA-Engine realisiert werden.

Ein zusätzlich zu berücksichtigender Faktor in der Rechenauslastung ist die Belastung des Ethernet-Stacks durch die Filterung von Nachrichten, die nicht vom Ethernet-MAC vorgefiltert worden sind. Hierzu zählen etwa alle mittels Broadcast übertragenen Nachrichten.

Entwicklungsaufwand/Kosten:

Neben dem sich ergebenden möglichen Realisierungsrisiko bei einer Verarbeitung der Video-Daten in Software, sind die NRE-Kosten zu berücksichtigen. Der Entwicklungsaufwand für eine Realisierung in Software ist zwar geringer anzusetzen, bei Einsatz des "XPS LL TEMAC" sind jedoch die Kosten für eine Produktions-Lizenz zu berücksichtigen[3]. Bei zusätzlichem Entwicklungsaufwand besteht die Möglichkeit, den lwIP-Ethernet-Stack auch an frei verfügbare Ethernet-MACs anzubinden. Die NRE-Kosten würden dann deutlich niedriger liegen.

[1] Ein geschachtelter Aufruf von Interrupt-Handler ist nicht automatisch möglich, da der MicroBlaze die für einen Rücksprung notwendige Position im Programm-Code nicht im Stack, sondern in einem Register ablegt. Geschachtelte Interruptaufrufe müssen daher verhindert werden.

[2] Mit Hilfe des "Base-System-Builder" kann die Konfiguration des MicroBlaze erfolgen.

[3] Zur Verfügung stünden verschiedene Lizenz-Modelle, u. a. mit der Möglichkeit ein auf 100 MBit/s beschränktes Design zu wählen.

Vor dem Hintergrund des Risikos, die Echtzeit-Anforderungen bei einer Nutzung eines in Software realisierten Ethernet-Stacks nicht erfüllen zu können und den sich ergebenden zusätzlichen NRE- und Unit-Kosten wird eine Realisierung des Ethernet-Stacks in Hardware vorgezogen. Die Verwendung eines zweiten MicroBlaze kommt aufgrund des zusätzlichen Ressourcenverbrauches nicht in Betracht.
Bei den Voruntersuchungen hat sich gezeigt, dass die Realisierung eines Wrappers für den EthMAC ein suboptimales Design bzgl. der Performance und des Ressourcenverbrauchs ergibt. Ursache hierfür ist die DMA-Engine, die im EthMAC integriert ist. Für die Implementierung soll diese DMA-Engine entfernt werden, wodurch sich eine einfachere Zugriffssteuerung und Konfiguration des Ethernet-MAC ergeben wird. Es wird erwartet, dass der Ressourcenverbrauch dadurch unterhalb des ebenfalls frei verfügbaren "Tri-Mode Ethernet-MAC" sinken wird und dennoch alle Anforderungen erfüllt werden. Der EthMAC wird daher für eine Realisierung vorgezogen.

4.2.3. Datenverarbeitung

Durch die Entscheidung, den Ethernet-Stack in Hardware zu realisieren, besteht die Möglichkeit, die Leistungsfähigkeit des Gesamtsystems zu steigern, indem die von den Radar-Transceiver kommenden Ethernet-Pakete in Hardware vorinterpretiert werden. Die Sample-Werte eines Radar-Sweeps, die einen hohen Datendurchsatz erreichen, deren Weiterverarbeitung jedoch eine vergleichsweise geringen Komplexitätsgrad aufweist, können so in Hardware weiter verarbeitet werden. Hingegen können die Header-Informationen des Radar-Sweep, die nur einen geringen Anteil des Gesamt-Datendurchsatzes ausmachen, für eine weitere Verarbeitung an den MicroBlaze übergeben werden. Der MicroBlaze benötigt diese Informationen zur Pulsgenerierung.
Dadurch kann auf der einen Seite die benötigte Rechenleistung des MicroBlaze gesenkt werden, auf der anderen Seite verringert sich der Bedarf Buffer für die Speicherung größerer Datenmengen einzusetzen. Die Empfangs- und Ausgabeperiode von Videosweeps muss so nur durch einen einzigen Buffer zeitlich entkoppelt werden. Würden die Daten über den MicroBlaze verarbeitet werden, müsste vermutlich mehr Speicher für die Zwischenspeicherung zur Verfügung stehen, da aufgrund zusätzlicher Kopiervorgänge die Latenz zwischen Empfang und Ausgabe zunehmen wird. Ein Speicherplatz müsste während der vollständigen Verarbeitung zur Verfügung stehen.
Die Reduzierung des Verbrauchs an Speicher-Ressourcen ist von besonderer Wichtigkeit. Ist dieser gering genug, kann auf eine Nutzung von externem RAM verzichtet werden. Die Voruntersuchungen in Abschnitt 3.3.4 haben gezeigt, welche negativen Auswirkungen die Nutzung von externem RAM auf die Performance des MicroBlazes hat. Die höchste Leistungsfähigkeit kann nur erzielt werden, wenn auf externen Speicher vollständig verzichtet und ausschließlich im FPGA integrierter BlockRAM eingesetzt wird [10]. Die Notwendigkeit Cache-Speicher einzusetzen entfällt in diesem Fall.
Aufgrund der Geschwindigkeit eines Hardware-Designs sind Buffer zur zeitlichen Entkoppelung zwischen Ethernet-MAC und Ethernet-Stack nicht notwendig und daher in der Entwicklung des Ethernet-Stacks zu vermeiden. Die Optimierung des EthMAC ist hierauf ebenfalls auszulegen.
Bild 4.1 zeigt die Partitionierung der Datenflüsse und die notwendigen Taktdomänen nach diesem Konzept: Auf der Hardware-Applikations-Ebene wird ein demultiplexen empfangener

UDP/IP-Nachrichten vorgenommen. Multicast-Nachrichten, die die Video-Informationen enthalten, werden an einen in Hardware realisierten Interpreter für das proprietäre Video-Protokoll gegeben. Der Video-Interpreter legt die Informationen in zwei verschiedenen Buffern ab. Der

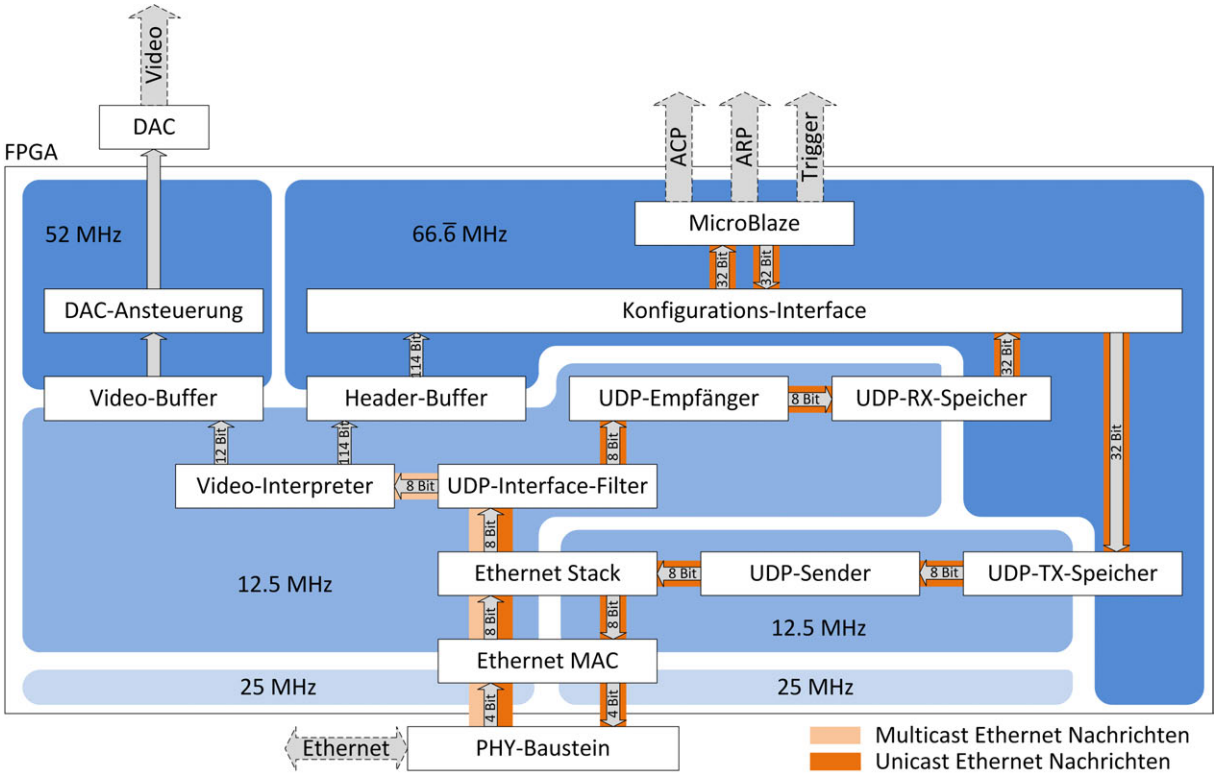


Bild 4.1.: Konzeptioneller Entwurf der Datenflüsse im Hardware-Design

erste Buffer beinhaltet ausschließlich die Sample-Werte und wird von der DAC-Ansteuerung ausgelesen, die ebenfalls in Hardware realisiert ist (Video-Buffer in Bild 4.1). Im zweiten Buffer werden die Informationen über den jeweiligen Radar-Sweep selbst abgelegt (Header-Buffer in Bild 4.1). Dadurch kann der MicroBlaze diese Informationen selbst abfragen und zur Pulsgenerierung, sowie zur Steuerung der DAC-Ansteuerung, verwenden. Wann das Auslesen und damit die Ausgabe des Videos durch die DAC-Ansteuerung stattfindet, wird durch den MicroBlaze auf Basis des zu generierenden Trigger-Pulses vorgegeben.
Unicast-Nachrichten, die z. B. Steuerinformationen beinhalten, werden in einem in Hardware realisierten Emfangs-Buffer gespeichert, sodass der MicroBlaze die Daten hieraus abfragen kann. Ferner wird die Möglichkeit implementiert, dass der MicroBlaze Daten in einem Sende-Buffer ablegen kann, um diese über Ethernet unter Verwendung des UDP/IP-Protokolls zu versenden.
Es wird angenommen, dass Speicher für die Zwischenspeicherung von mindestens 16 Radar-Sweeps notwendig sind und die MicroBlaze-Software 32 kByte Speicher für die Speicherung und Ausführung der Software benötigt. Für das Senden und Empfangen von Ethernet-Nachrichten über den MicroBlaze wird je Richtung ein Buffer von 4 Ethernet-Nachrichten angesetzt. Damit ergibt sich ein Minimumverbrauch von 42 BlockRAM-Blöcken, bei 18 kBit je Block, welches insgesamt 94.5 kByte an BlockRAM-Speicher entspricht.

4.2.4. Hardware-Software-Interface

Durch die Partitionierung der Aufgaben in Hard- und Software besteht die Notwendigkeit, eine Verbindung zwischen beiden Ebenen herzustellen. Am wenigsten Ressourcen würde eine Schnittstelle benötigen, wenn diese alle Anforderungen erfüllt und damit nur eine Schnittstelle implementiert werden muss. Die Anforderungen sind:

- Lesende und Schreibende Zugriffe auf einzelne Register des Hardware-Designs. Der MicroBlaze kann so Initialisierungen, Steuerungen und Statusabfragen durchführen.
- Die bidirektionale Übertragung von Datenblöcken mit bis zur Größe einer UDP-Nachricht (maximal 1478 Byte), um die Übertragung von Steuerinformationen zu ermöglichen.

Die Anforderung an die Geschwindigkeit der Schnittstelle lässt sich messbar machen, indem die Abarbeitungszeiten der Interrupthandler so kurz sein müssen, dass der maximal zulässige Jitter der Radarpulse nach Abschnitt 4.2.1 nicht überschritten wird.
Für die Anbindung des MicroBlaze an die Hardware unterstützt der MicroBlaze verschiedene Möglichkeiten:

- PLB-Bus:
  Hierbei handelt es sich um den "IBM Processor Local Bus", der zur Anbindung verschiedener Peripherien an den MicroBlaze genutzt wird. Es besteht die Möglichkeit das sich bis zu 16 Slave- und 16 Master-Teilnehmer einen Bus teilen. Der Buszugriff wird dabei von einem Bus-Arbiter geregelt, der innerhalb von drei Takten einen prioritätsgesteuerten Zugriff aushandelt. Die Anbindung der Peripherie kann entweder direkt mittels PLB-Bus erfolgen oder über ein Shared-Memory. Letzteres wäre als Dual-Port-BlockRAM realisiert. Über den "XPS Central DMA Controller" besteht die Möglichkeit, auch DMA Transfers durchzuführen. In diesem Fall kann der PLB-Bus jedoch nicht zur Ansteuerung anderer Peripherien, wie Timer, genutzt werden, wodurch der Einsatz eines zweiten PLB-Bus notwendig wird [55].
- AXI:
  AXI ist ein von ARM[4] entwickeltes und zur AMBA (Advanced Microcontroller Bus Architecture) gehörendes "Advanced eXtensible Interface". Es beschreibt damit keinen Bus, sondern ein Interface. Xilinx unterstüzt die Anbindung von IP-Cores mit AXI-Interface erst bei den neueren FPGAs, d. h. ab FPGAs der Spartan-6 und Virtex-6 Familie. Zur Verfügung steht ein Interface für Memory-Mapped-Zugriffe und ein Interface für Streaming-Kommunikationen. Während bei Ersterem Burst-Zugriffe auf 256 Daten-Transfer-Zyklen beschränkt sind, gibt es bei Letzterem keine Beschränkung. Ferner existiert eine AXI-Lite Version, welche Memory-Mapped-Zugriffe mit geringerem Aufwand, aber auch mit geringerem Datendurchsatz, erlaubt.
  Ein über die AXI-Interfaces ansteuerbarer Bus stellt eine performantere Alternative zum PLB-Bus dar, mit dem Vorteil, dass auch anderes FPGA-Hersteller, wie Altera, diese Interfaces unterstützen [7]. Damit kann zum einen auf einen größeren Umfang an IP-Cores zurückgegriffen werden und zum anderen IP-Cores entwickelt werden, die kompatibel zu FPGAs verschiedener Hersteller sind. Der auf dem AXI-Interface basierende Bus wird

[4] Firmen-Abkürzung für das Unternehmen "ARM Limited", welche "Intellectual Property Cores" im Bereich der Mikroprozessoren anbietet.

bei Xilinx den PLB-Bus zunehmend ersetzen.
Der Nachteil des AXI-Busses besteht darin, dass keine Kompatibilität zu älteren FPGAs von Xilinx besteht.

- LocalLink:
  Der LocalLink ist kein Bus im eigentlichen Sinne, sondern eine hochperformante, synchrone Punkt-Zu-Punkt-Verbindung. Der LocalLink wurde ursprünglich von Xilinx für die Anbindung des Ethernet-MAC "XPS LL TEMAC" [58] entwickelt, kann aber auch durch Customize-IP-Cores genutzt werden. Mittels des LocalLink ist eine Anbindung über den Soft-DMA-Controller (SDMA) und dem MPMC (Multiport Memory Controller) möglich, sodass Daten per DMA in ein externes RAM geschrieben oder aus diesem gelesen werden können. Beim "XPS LL TEMAC" wird der LocalLink von Xilinx für die Datenübertragung mit hoher Übertragungsrate verwendet, nicht aber für Registerzugriffe. Xilinx selbst nutzt für Registerzugriffe beim "XPS LL TEMAC" den PLB-Bus bzw. alternativ den AXI-Bus.
  Die Anbindung mittels LocalLink wäre für die Übertragung der Ethernet-Information unter Einsatz der DMA-Engine ideal, da diese Verbindung sehr performant ist. Nachteil ist, dass die Nutzung nur in Verbindung mit dem MPMC und damit unter Einsatz von externen Speicher möglich ist. Die Nutzung von externem Speicher sollte jedoch vermieden werden, solange keine absolute Notwendigkeit besteht.

- FSL:
  Beim "Fast Simplex Link" handelt es sich um eine unidirektionale, optional FiFo-basierte, Punkt-zu-Punkt-Verbindung. Gedacht ist die FSL-Verbindung zur Anbindung von Hardware-Beschleunigern an den Prozessor, sodass Berechnungen in Hardware ausgelagert werden können, um sie dort schneller ausführen zu können. Die Ansteuerung von bis zu 16 FSL-Verbindungen geschieht direkt über get- und put-Befehle der MicroBlaze ISA (Instruction Set Architecture). Einzelne Schreib- und Lesezugriffe können so vollständig innerhalb von 2 Takten abgearbeitet werden [16, 40, 54].
  Die FSL-Verbindung stellt damit eine flexibel gehaltene Möglichkeit dar, Hardware-IP-Core an den MicroBlaze anzubinden. Es können nicht nur einzelne Werte, sondern auch beliebig viele Daten als Datenstream übertragen werden. Um das Beschreiben bzw. Auslesen von verschiedenen Registern und Speicherbereichen zu ermöglichen, müssen hier jedoch Zugriffs-Interfaces auf der Software- und Hardware-Seite entwickelt werden, die nach einem zu definierenden Kommunikationsprotokoll Memory-Mapped-Zugriffe ermöglichen. Die Möglichkeit eines DMA-Zugriffs existiert hier nicht, ein Burst-Mode[5] kann hingegen implementiert werden.

Für die Realisierung wird sich gegen den AXI-Bus, aufgrund fehlender Kompatibilität zu Spartan-3 und Virtex-5 FPGAs[6], entschieden. Ebenfalls der LocalLink ist nicht geeignet, da dieser nur in Verbindung mit externem Speicher genutzt werden kann und seine Leistungsfähigkeit nur bei der Übertragung größerer Datenmengen ausgenutzt werden kann. Bei Einsatz des PLB-Busses müssen zwei separate Busse implementiert werden, wenn eine DMA-Übertragung

[5] Die Übertragung von Daten im Burst-Modus beschleunigt Lese- und Schreibvorgänge, indem Datenpakete zusammengefasst übertragen werden und damit nur eine einmalige Initialisierung der Übertragung am Anfang stattfinden muss und nicht für jedes Datenpaket erneut.

[6] Eine Kompatibilität zu Virtex-5 FPGAs muss sichergestellt sein, damit das Ethernet-Interface in Projekten der "HAW Hamburg" genutzt werden kann.

gewünscht ist. Größere Datenmengen können dadurch kopiert werden, ohne dass Rechenleistung dafür benötigt wird. Bei Einzelzugriffen wird der PLB-Bus jedoch hinter der Zugriffsgeschwindigkeit der FSL-Verbindung zurückbleiben. Die FSL-Verbindung wird zwar größere Datenmengen schneller transportieren können als der PLB-Bus, für den Kopiervorgang selbst kann jedoch kein DMA eingesetzt werden, wodurch Rechenzeit benötigt wird.
Unter Abwägung der Vor- und Nachteile wurde sich für die FSL-Verbindung entschieden, da Einzelzugriffe sehr hochfrequent auftreten und hier der Fast-Simplex-Link im Vorteil liegt. Die Übertragung größerer Datenmengen wird eher die Ausnahme sein. Steuerkommandos über Ethernet sind i. d. R. wenige Byte groß. Eine DMA-Verarbeitung wird hier - wenn überhaupt - nur mit einem minimalen Gewinn an Rechenzeit verbunden sein. Der zusätzliche Ressourcenverbrauch und der zusätzliche Aufwand, die DMA-Verarbeitung zu verwalten, wird sich kaum rentieren. Durch die schnellen Zugriffe wird der erreichbare Datendurchsatz höher erwartet als beim PLB-Bus. Muss die Kompatibilität zu FPGAs der Spartan-3 bzw. Virtex-5 Familie nicht mehr sichergestellt sein, bestünde die Möglichkeit ein Signal-Mapping auf eine AXI-Stream-Verbindung durchzuführen. Der Fast-Simplex-Link und die AXI-Streaming-Verbindung haben nach [79] ähnliche Signale.

4.3. Testkonzept

Für die Entwicklung eines Produktes bzw. hier eines Prototypen sind Tests während und nach der Entwicklung unerlässlich. Für die Durchführung einer testgetriebenen Entwicklung gibt es zahlreiche Literaturen, die allerdings die testgetriebene Entwicklung vor dem Hintergrund der objektorientierten Software-Programmierung betrachten. Dennoch lassen sich einzelne Aspekte auch auf die Implementierung eines Hardware-Designs übertragen.
Ein interessanter Ansatz der testgetriebenen Entwicklung ist die inkrementelle Entwicklung. Ein Projekt wird hier in Teilaufgaben zerlegt, wie die Erstellung eines Ethernet-Stacks. Teilaufgaben werden weiter in sog. Mikroschritte unterteilt. Ein solcher Schritt kann z. B. die Implementierung der Verarbeitung des IP-Protokoll-Headers in Empfangsrichtung sein. Jeder dieser Mikroschritte wird für sich genommen getestet. Die Programmierung findet damit im Mikroiterationen statt: Als Erstes werden Tests geschrieben für den in dieser Iteration zu erstellenden Code, danach erfolgt die Beschreibung der Hardware, bis diese den Test erfüllt. Anschließend findet das Refactoring statt, bei dem die Hardware-Beschreibung “aufgeräumt” wird, um diese verständlicher zu machen und um ggf. einzelne Funktionsblöcke zu abstrahieren. Bei diesem “evolutionären Entwurf” wird der Fokus auf eine durchgehende Designverbesserung gelegt und weniger auf das Festhalten an einem detailliert erstellen Anfangsentwurf. Dies schließt konzeptionelle Vorgedanken nicht aus [20].
Die Kombination der Aspekte: Unit-Tests, Refactoring und fortlaufende Integration in einem iterativen Entwicklungsprozess scheint sehr geeignet für die Entwicklung eines HDL-Designs zu sein und soll daher bei dieser Entwicklung zum Einsatz kommen.

4.3.1. Unit-Tests

Unit-Tests werden hier mittels VHDL-Testbenches realisiert. Um die Praktikabilität dieser Tests und damit die Effizienz der Entwicklung zu steigern, werden nicht separate Testbenches für

jede Unit entwickelt, sondern eine einzige Testbench. Diese Testbench stellt jedoch Testvektoren auf der niedrigsten Abstraktionsebene des HDL-Designs zur Verfügung. Im konkreten Fall ist dies die zeitgleiche Simulation des MII-Interface zwischen Ethernet-MAC und PHY-Baustein und die Simulation des FSL-Interfaces zwischen HDL-Design und MicroBlaze. Im Zuge einer bevorzugten Bottom-Up-Entwicklung, bei der Systemkomponenten mit niedrigem Abstraktionsniveau vor den Systemkomponenten mit einem hohen Abstraktionsniveau entwickelt werden, ergeben sich automatisch weitere Unit- und Integrationstests. Während etwa der Ethernet-MAC als Unit direkt durch die Testbench getestet werden kann, kann für die Entwicklung des Ethernet-Stack der Ethernet-MAC als Teil der Testbench angesehen werden, der dann die Testvektoren zur Verfügung stellt. Zeitgleich erfolgen damit auch Integrationstests, da die einzelnen Units immer direkt in Interaktion mit anderen Units getestet werden.
Die konkrete Implementierung der Testbenches wird in Abschnitt 5.1.4 erläutert.

4.3.2. System-Tests

Durch die Bottom-Up-Entwicklung können teilweise bereits während der Entwicklung erste Systemtests durchgeführt werden. Dieses Tests ermöglichen, das System auf funktionale wie auch nicht funktionale Anforderungen hin zu testen. Da hier die Möglichkeit besteht, gegen die Spezifikation zu testen, können diese Tests als Black-Box-Tests verstanden werden. Es ist unerheblich, wie die konkrete Realisierung aussieht, die spezifizierten Anforderungen müssen erfüllt werden.
Um Systemtests zu ermöglichen, die in Kapitel 7 genauer erläutert werden, besteht die Notwendigkeit, das entwickelte Design auf einem Prototypen zu implementieren. Der Aufbau des Prototypen soll dabei soweit wie möglich durch ein oder mehrere Evaluationboards realisiert werden. Bild 4.2 zeigt die hierzu notwendigen Funktionseinheiten. Relevante Bauteile, wie das

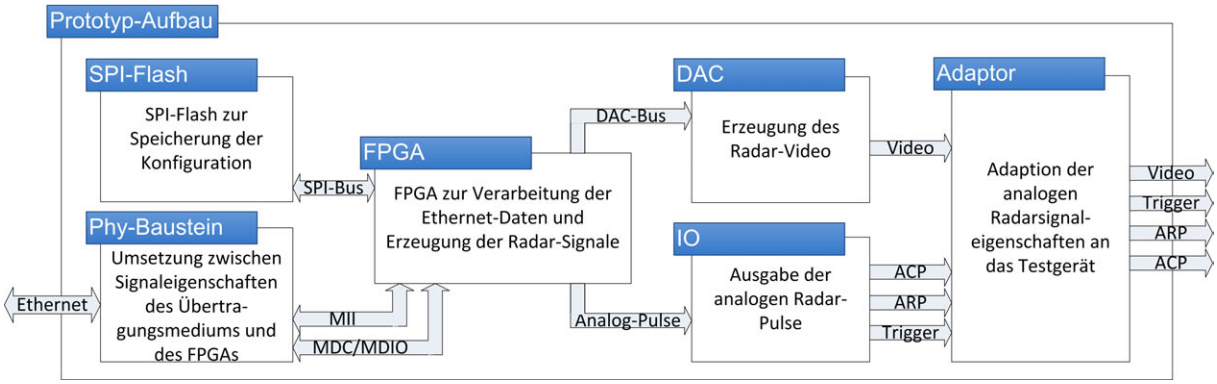


Bild 4.2.: Funktionsblöcke des Prototypen-Aufbaus

FPGA und der DAC sollen in ihrem Funktionsumfang, sowie dem Ein- und Ausgangsverhalten, Bauteilen entsprechen, wie sie auch bei einer Serienproduktion zum Einsatz kommen können.
Unter Berücksichtigung der Speicherabschätzung in Abschnitt 4.2.3 von 42 BlockRAM-Blöcken und der Anzahl benötigter Logik-Ressourcen nach Abschnitt 3.5 für die Anbindung eines FPGAs mittels Ethernet und der Bereitstellung eines MicroBlaze ist nach [51, 53] mindestens eine Spartan-6 der Derivate XC6SLX25 notwendig. Weitere Anforderungen, die sich an ein Evaluationboard mit diesem FPGA stellen sind:

- Das Board ist mit einem Ethernet-PHY-Baustein ausgestattet, welcher wahlweise mit MII- oder GMII-Interface an das FPGA angebunden ist.
- Das Board ist mit einem Digital-Analog-Umsetzer (DAC) bestückt, welcher mindestens 52 MSample/s bei einer Auflösung von 12 Bit erreichen kann. Ist ein zusätzliches Evaluationboard notwendig, muss die Schnittstelle zwischen FPGA und DAC mindestens für Übertragungsraten von 52 MBit/s pro Leitung ausgelegt sein (624 MBit/s insgesamt).
- Für die Ausgabe der Pulse Trigger, ACP und ARP müssen mindestens 3 Pins des FPGA zugänglich sein.

Unter Beachtung dieser Anforderungen ist das FPGA-Evaluationboard SP605 mit einem Spartan-6 XC6SLX45T ausgewählt worden. Das FPGA hat mit insgesamt 6822 Slices und 116 BlockRAMs mehr als doppelt so viel Ressourcen zur Verfügung, wie nach der Schätzung benötigt werden. Dies senkt das Risiko einer Nicht-Realisierbarkeit. Für die Ausgabe der Analog-Pulse besitzt das SP605 GPIO-Pins, die genutzt werden können. Dieses Board, wie auch alle anderen FPGA-Evaluationboards, besitzt hingegen keinen DAC mit den gestellten Anforderungen. Eine Nutzung des RAMDACs des "DVI Transmitter"-Chip, wie er auch auf dem SP605 vorhanden ist, kommt nicht in Betracht. Dieser erreicht zwar die gewünschte Samplerate, aber nicht die Bitauflösung von 12 Bit. Da der Prototyp möglichst dicht an einer späteren Produktrealisierung aufgebaut werden soll, soll ein DAC als klassisches Bauelement zum Einsatz kommen. Nicht zuletzt auch, um sich das Wissen über die Ansteuerung eines solchen DACs zu erarbeiten.
Im Gegensatz zu den meisten anderen Evaluationboards besitzt das SP605 einen sog. FMC-Stecker (FPGA Mezzanine Card). Hierbei handelt es sich um eine nach ANSI[7]-Standard definierte Möglichkeit, Evaluationboards anderer Hersteller an das FPGA-Evaluationboard anzubinden. Der FMC-Stecker ist für Übertragung hochfrequenter Signale bis 10 GBit/s ausgelegt, sodass die Anbindung eines DAC-Evaluationboards über diese Verbindung möglich ist.
Für diese Schnittstellen liefern daher verschiedene Hersteller DAC-Evaluationboards. Hersteller wie "Analog Device" und "Texas Instruments" stellen für diese Boards zusätzliche Adapter zur Verfügung. Diese sind notwendig, da die DAC-Boards selbst proprietäre Stecker für die herstellereigenen "Data-Pattern"-Generatoren nutzen.
Neben der Berücksichtigung technischer Anforderungen, wie Sample-Rate, Bit-Auflösung und die Möglichkeit der Anbindbarkeit unter Verwendung des FMC-Steckers, musste auch die Lieferbarkeit in die Entscheidung einfließen. Für einen Prototyp ist dies akzeptierbar, da die Ansteuerung von DACs auf ähnliche Weise erfolgen. Aus diesem Grund wurde sich für den DAC AD9716 von Analog Device [1] entschieden. Hierbei handelt es sich um einen 12-Bit DAC mit zwei Kanälen mit jeweils maximal 125 MSample/s.
Für einen Systemtest im Gesamtsystem sind die, durch den Prototypen generierten, analogen Signale auf ein Radar-Sichtgerät zu geben, sodass eine qualitative Beurteilung der Aufbereitung des Radarbildes durchgeführt werden kann. Bild 4.3 zeigt den sich ergebenden Aufbau für die Systemtests im Detail: Während die Radar-Informationen von einem Radar-Transceiver zur Verfügung gestellt werden, wird die Steuerung der Elektronik über einen PC durchgeführt. Unter Verwendung dieser Daten berechnet und generiert die Elektronik die analogen Radar-Signale. Die Radar-Pulse Trigger, ARP und ACP werden direkt über Pins des FPGAs auf eine entwickelte Adapter-Platine gegeben. Das Radar-Video wird hingegen durch das DAC-Evaluationboard [2] generiert, welches über den FMC-Stecker und der FMC-Adapter-Platine an

[7] ANSI: American National Standards Institute

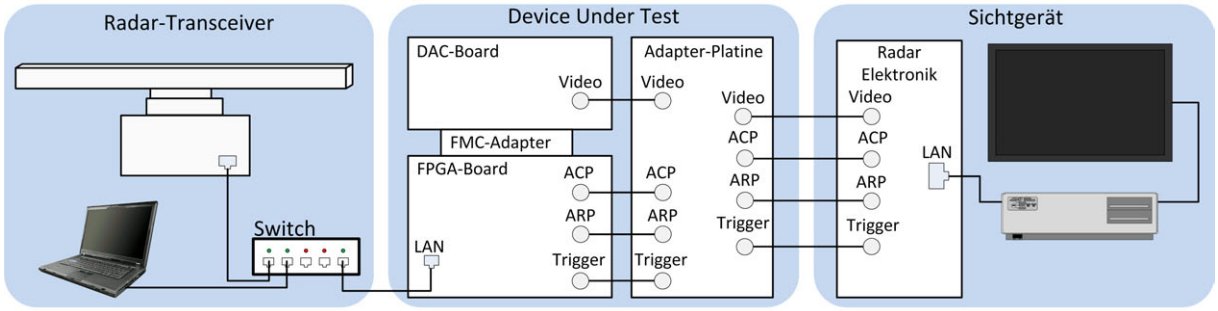


Bild 4.3.: Aufbau für Systemtest des Prototypen

das FPGA-Evaluationboard angebunden ist. Auch das Video wird auf die entwickelte Adapter-Platine gegeben. Die Adapter-Platine bereitet die Signale in ihren analogen Signaleigenschaften so auf, dass diese von einem Sichtgerät verarbeitet und als Radar-Bild dargestellt werden können. Aus Gründen der Praktikabilität wird hierfür ein Sichtgerät der neuesten Generation eingesetzt. Dies macht es erforderlich, dass die erzeugten, analogen Radar-Signale wieder digitalisiert werden. Dies wird durch eine Elektronik ermöglicht, die gewöhnlich diese Aufgabe in einem Radar-Transceiver übernimmt.
Dem Anhang A.3.1 kann die Entwicklungs-Dokumentation entnommen werden, in der alle notwendigen Änderungen am DAC-Evaluationboard sowie Entwurf und Layout der entwickelten Adapter-Platine im Detail dokumentiert sind.

KAPITEL 5

Hardware Implementierung

Im folgenden Kapitel wird auf den ersten Bestandteil der Implementierung eingegangen - der Hardware-Implementierung. Die Implementierung setzt dabei das in Kapitel 4 erläuterte Konzept um.
Der Aufbau des Kapitels behandelt in Abschnitt 5.1 vorerst übergeordnete Aspekte, bevor in Abschnitt 5.2 auf die Implementierung des MicroBlaze und in Abschnitt 5.3 auf dessen Anbindung an die Hardware eingegangen wird. Abschnitt 5.4 und 5.5 beschäftigen sich anschließend mit der Realisierung des Ethernet-MACs und -Stacks. Abschnitt 5.6 behandelt die Verarbeitung der Radar-Informationen vom Ethernet bis hin zur Ausgabe des Radar-Videos.

5.1. Allgemeines

Dieser Abschnitt wird vorab auf einzelne Aspekte der Hardware-Implementierung eingehen, auf deren Grundlage die Implementierung durchgeführt ist. Hierzu zählt das Timing in Abschnitt 5.1.1, der Codierungs-Stil in Abschnitt 5.1.2 und Untersuchungen zur Durchführung eines zeiteffizienten Design-Flows in Abschnitt 5.1.3.

5.1.1. Timing

Timing-Constraints in FPGA-Designs definieren minimale zeitliche Anforderungen an ein Design, welche für eine korrekte Funktion einzuhalten sind. Unter Berücksichtigung dieser Constraints führen die Implementierungs-Tools die Implementierung des Designs durch, indem sie nach einer Lösung suchen, die alle gestellten Anforderungen an das Timing erfüllt. Auf Over-Constraints[1] sollte verzichtet werden, sie verlängern unnötig den Design-Flow und können unter Umständen zu einer Verschlechterung des Gesamtdesigns führen [67, S.10]. Das von den Tools angesetzte Zeitverhalten der Logik basiert dabei auf den vom Hersteller garantierten Zeiten unter allen spezifizierten Bedingungen. Damit wird auch berücksichtigt, dass das Timing-Verhalten aufgrund von Schwankungen in der FPGA-Produktion, unterschiedlichen Chiptemperaturen, Versorgungsspannung u. v. m. variieren kann.
Vor dem Hintergrund des internen Aufbaus von FPGAs (siehe Abschnitt 2.1), kann sich das Routing eines FPGAs mit jedem Design-Flow ändern. Eine Überprüfung des Timings des Implementierungs-Ergebnisses ist daher prinzipiell nach jedem Design-Flow notwendig. Timing-Constraints ermöglichen hierfür eine schnelle statische Timing-Analyse, mithilfe der

[1] Over-Constraints sind Timing-Constraints die eine höhere Anforderungen an das Timing eines Designs stellen, als für die eigentliche Aufgabe notwendig wäre.

alle durch Timing-Constraint beschriebenen Pfade auf die Einhaltung der Constraints hin überprüft werden können. Unter der Annahme, dass das erforderte Zeitverhalten ausreichend durch Timing-Constraints definiert ist, kann so auf Timing-Simulationen verzichtet und die Überprüfung auf eine statische Timing-Analyse beschränkt werden [31]. Gerade bei größeren Projekten ist dies unabdingbar. Je größer ein Projekt ist, desto mehr Rechenleistung und -zeit wird für die Timing-Simulation benötigt. In [31] wird angegeben, dass die Rechenzeit zur Durchführung einer Timing-Simulation im Bereich von 10 Stunden liegt, wenn für die Verhaltenssimulation eine Rechenzeit von 30 Minuten benötigt wird.

Es stehen eine Vielzahl verwendbarer Timing-Constraints zur Verfügung. In dem hier realisierten Design finden die drei wichtigsten Constraints eine Anwendung:

Period-Constraints: Angewendet auf ein Takt-Netz werden alle kombinatorischen Pfade mit ihren Logik- und Routing-Verzögerungen, der Setup- und Hold-Zeiten, des Clock-Skew und der Takt-Phase, sowie des Tastgrades, berücksichtigt. Es kann so sichergestellt werden, dass die Ergebnisse der kombinatorischen Pfade innerhalb einer Taktperiode stabil anliegen [67, S. 46].

Offset-Constraints: Definieren das Timing-Verhältnis zwischen einem Takt-Pad und dazugehörigen Datenein- oder -ausgangs-Pads. Bild 5.1 zeigt dies für die Ausgabe von Daten. Mit z. B. einer “OFFSET OUT AFTER”-Angabe kann angegeben werden, wieviel Zeit nach steigender/fallender Takt-Flanke vergehen darf, bis die Daten stabil am Ausgangs-Pin anliegen müssen und wie lange diese anzuliegen haben. Berücksichtigt wird die Verzögerung im Taktpfad, die “Clock-To-Output”-Zeit der Register und die Laufzeit vom Register zum IO-Pin [67, S. 51ff].

From:To-Constraints: Sind für “Multi-Cycle”-Pfade, d. h. kombinatorische Pfade bei denen die Laufzeit zwei oder mehr Takte betragen darf, geeignet. Durch Period-Constraints alleine (One-Cycle-Constraints) könnte es bei diesen Pfaden zu einer, ggf. unbegründeten, Nichteinhaltung des Timings kommen. Dies kann zu Performance-Verlusten im Design führen, wenn die Implementierungstools versuchen, das Timing zu erreichen.

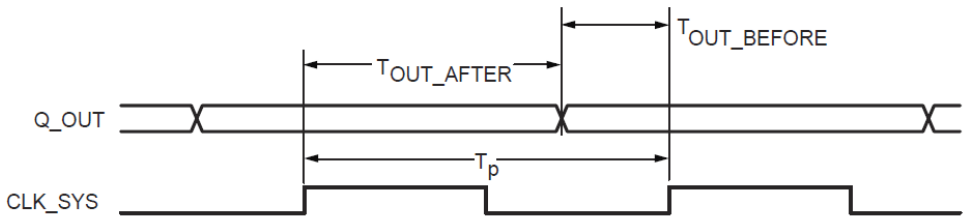


Bild 5.1.: Beispiel für Offset Out Constraints [67, S. 51]

Das Timing kann ebenfalls durch Anpassung von Synthese- und Implementierungs-Optionen beeinflusst werden. So führt z. B. eine so konfigurierbare automatische Register-Duplikation dazu, dass an kritischen Stellen der FAN-Out reduziert werden kann. Ein hoher FAN-Out senkt die maximal mögliche Taktrate. Mittels der “SmartXplorer”-Funktion, die im “ISE Design Navigator” zur Verfügung steht, besteht die Möglichkeit verschiedene Synthese- und Implementierungsstrategien für das Design automatisiert anzuwenden und die Ergebnisse zu vergleichen. Ein Vergleich zeigt neben der benötigten Rechenzeit und den benötigten Ressourcen den minimalsten “Slack” des Designs. Der Slack gibt an, um wieviel schneller der kritischste Pfad,

bezogen auf die gewählten Timing-Constraints, ist. Ist der Slack größer Null, so ist das Design besser, ist der Wert kleiner Null, so werden nicht alle Timing-Constraints erfüllt. Anstatt der Verwendung der “SmartXplorer”-Funktion, kann auch “PlanAhead” genutzt werden. Dieses Tool erlaubt die Erstellung mehrerer paralleler Synthese- und Implementierungs-Flows unter Verwendung verschiedener Optionen.

5.1.2. Codierungs-Stil

Automaten

Es gibt verschiedene Möglichkeiten Automaten zu realisieren. Um Zustandsautomaten zur Verarbeitung von Protokollen zu erstellen, stellten sich Moore-Automaten als ungeeignet herraus. So müsste z. B. für die Interpretation eines Protokolls jedes Datenfeld durch einen Zustand beschrieben werden. Die Aufgabe der FSM bestünde dann u. a. darin separierte Logik zu steuern, welche die Verarbeitung der Daten vornimmt. Hierzu zählt z. B. das Speichern von Datenfeldern, um diese anderen Zustandsautomaten zur Verfügung stellen zu können. Es entsteht dadurch ein relativ großer “VHDL-Overhead” zur Beschreibung des Verhaltens. Die Beschreibung wird dadurch unübersichtlich und fehleranfälliger.
Es ist sich daher entschieden worden, Zustandsautomaten als Mealy-Automaten zu realisieren. Diese sind flexibler und erlauben eine geringere Anzahl an Zuständen. Durch die gewählte Beschreibung dieser Automaten in der Ein-Prozess-Darstellung ergibt sich automatisch eine Ausgangssignalsynchronisation, wodurch sich das Eingangssignalverhalten nicht direkt auf die nachfolgende Logik auswirken kann. Ein Fehlverhalten der nachfolgenden Logik durch Glitches im Eingangssignalverhalten kann so vermieden werden. Darüber hinaus werden so Latches, kombinatorische Schleifen und lange kombinatorische Pfade automatisch vermieden. Letzteres kann zu einer Steigerung der maximal möglichen Taktfrequenz führen. Bei einer Verarbeitung von Protokollen können die Daten bei dieser Realisierung direkt durch den Zustandsautomaten verarbeitet werden. So kann z. B. Aufgrund der Ausgangssignalsynchronisation eine Speicherung der Datenfelder direkt durch die FSM erfolgen, da alle Zuweisungen durch Flip-Flops realisiert werden. Die Protokoll-Verarbeitung lässt sich dadurch übersichtlicher und damit einfacher nachvollziehbar erstellen. Die Fehleranfälligkeit wird so minimiert.
Ein Nachteil, der sich durch die Ausgangssignalsynchronisation ergibt, ist der relativ hohe Verbrauch von Register-Ressourcen. Dies ist akzeptierbar, da diese Ressource i .d. R. in einem sehr hohen Umfang auf den FPGAs zur Verfügung steht.
Nach [80] wird eine Beschreibung von Automaten dieser Art durch die Synthese-Tools unterstützt.

Resets

Xilinx rät davon ab, globale Resets zu verwenden [62, 63]. Es wird garantiert, dass alle Register eines FPGAs von Xilinx nach einem Power-Up logisch null sind. Da die FPGAs hier SRAM basiert sind, ist eine Signalinitialisierung mit Power-Up-Werten zulässig. Globale Resets werden daher nicht zur Initialisierung benötigt, sie wirken sich sogar negativ auf die Performance, den Ressourcenverbrauch und die Leistungsaufnahme des Designs aus.

Aus diesem Grund wird in der Design-Implementierung auf globale Resets verzichtet. Ausgenommen hiervon ist der Reset des externen PHY-Bausteins und einer damit einhergehenden Notwendigkeit die Initialisierung des PHY-Bausteins nach einem Reset durchzuführen. Der Reset wird zeitlich erst nach einem Power-Up zurückgenommen.

5.1.3. Untersuchungen zur Beschleunigung des Design-Flows

Das hier implementierte Design benötigt für einen normalen Design-Flow eine Implementierungszeit von 23 Minuten, sodass die Notwendigkeit gesehen wurde, verschiedene Möglichkeiten des Design-Flows zu untersuchen:

Synthese- und Implementierungs-Tools im Command-Line-Mode:

Die erste untersuchte Möglichkeit ist der Verzicht auf GUI-basierte Tools. Nach [36] führt dies zur Einsparung von RAM und CPU-Performance und damit zu einer Minimierung der benötigten Rechenzeit. Xilinx bietet hierfür verschiede Möglichkeiten an: Neben dem Tool "XFlow" [73, S. 287 ff.], bei dem alle Schritte des Design-Flows auf Command-Line-Ebene durch ein einziges Tool durchgeführt werden, können auch die von Xilinx bei GUI-basierten Programmen im Hintergrund eingesetzen Tools XST, NGDBuild, MAP, PAR und BitGen separiert im Command-Line-Mode aufgerufen werden. Ferner existieren Möglichkeiten, mittels TCL-Skipt[2] den Design-Flow entweder mit "xtclsh"[3] [73, S.335 ff.], ebenfalls im Command-Line-Mode, durchzuführen oder den TCL-Mode von PlanAhead zu verwenden.

Der größte Zeitgewinn wurde bei der Nutzung der einzelnen Xilinx-Tools (XST, NGDBulid, u. s. w.) im Command-Line-Mode erwartet. Signifikate Verbesserungen konnten jedoch nicht eindeutig nachgewiesen werden. Es steht zu erwarten, dass die Nutzung von Command-Line-Tools nur sinnvoll ist, wenn eine Durchführung des Design-Flows auf leistungsstärkeren Rechnern ohne GUI-Unterstützung erfolgen soll. Dem Anhang A.3.6 kann das erstellte Batch-Skript entnommen werden, welches einen Design-Flow zu Testzwecken durchgeführt hat.

Inkrementeller Design-Flow:

Eine weitere Möglichkeit den Design-Flow zu beschleunigen besteht im inkrementellen Design-Flow unter Verwendung von Partitionen. Wird eine Komponente als Partition deklariert, so kann den Synthese- und Implementierungs-Tools mitgeteilt werden, in welchem Umfang Synthese- und/oder Platzierung- und Routing-Ergebnisse für die jeweilige Partition für zukünftige Design-Flows übernommen werden sollen. Dadurch kann neben der Einsparung an Rechenzeit auch die Reproduzierbarkeit von Designs erhöht und auf eine erneute Timing-Verifikation der Partitionen verzichtet werden.

Xilinx empfiehlt [57] die Verwendung von Partitionen für Funktionsblöcke wie DSP-Module, EDK-Systeme (z. B. der MicroBlaze), für zeitkritische IP-Cores, Logik die zusammenhängend platziert und Module die nicht mehr verändert werden sollen. Eine Partitionierung sollte nur dort stattfinden, wo sie wirklich notwendig ist, da diese immer die

[2] TCL steht abkürzend für "Tool Command Language". Ein TCL-Skript beschreibt den exakten Ablauf eines Design-Flows mit allen gewünschten Optionen. Diese Skripte sind daher geeignet, um sicherzustellen, dass ein Design-Flow immer exakt gleich durchgeführt wird.

[3] Als "xtclsh" wird die Xilinx Tcl Shell bezeichnet, die das Ausführen eines TCL-Skriptes von der Command-Line-Ebene aus ermöglicht.

Freiheiten der Design-Flow-Tools einschränken und damit auch negative Effekte verursachen können, etwa einen erhöhten Ressourcenverbrauch oder ein schlechteres Timing des Gesamtdesigns.
Mittels des "ISE Project Navigator" lassen sich Partitionen durch Erstellung einer "xpartition.pxml"-Datei im Root-Verzeichnis des Projektes und durch manuelles Auslagern der zu verwendenden Synthese/Implementierungs-Ergebnisse erstellen [68]. Das Tool "PlanAhead" liefert hierfür eine GUI-Oberfläche zur einfacheren Verwaltung von Partitionen. Dem Anhang A.3.5 kann ein erstelltes Tutorial entnommen werden, indem die wichtigsten Schritte zur Erstellung eines Projektes in PlanAhead und zur Durchführung eines inkrementellen DesignFlows erläutert werden.

"SmartGuide"-Funktion:
Anstelle eigene Partitionen einzurichten, wird vom "ISE Design Navigator" auch die sog. "SmartGuide"-Funktion angeboten. Auch diese nutzt die Ergebnisse einer früheren Implementierung. Bei einer Implementierung werden jeweils nur die Bereiche neu implementiert, die sich verändert haben. Die "Smart-Guide"-Funktion sollte erst am Ende der Entwicklungsphase eingesetzt werden, wenn nur noch kleine Änderungen vorgenommen werden und die Zeitanforderungen bereits erfüllt werden. Im Gegensatz zum zuvor beschriebenen Vorgehen kann hier kein Einfluss auf den Umfang der Wiederverwendung von Implementierungs-Ergebnissen genommen werden. In einem Test konnte so die Laufzeit um 17.7 % reduziert werden.

Neben der Möglichkeit einer Partitionierung des Designs mittels GUI und einer dadurch nachweislich schnelleren Durchführung des Design-Flows, bietet PlanAhead eine ausführlichere grafische Aufbereitung u. a. der Ressourcenverbräuche und Ergebnisse der statischen Timing-Analyse an. Im Gegensatz zu den Text-Reports des ISE Design Navigators ist damit eine subjektiv bessere und effektivere Analyse des Designs möglich. Aus diesem Grund wird im folgenden Projekt die Synthese und Implementierung mit PlanAhead durchgeführt.

5.1.4. Testen der Hardware-Implementierung

Nach der Konzeption in Abschnitt 4.3 soll die Hardware unter dem Gesichtspunkt der testgetriebenen Entwicklung implementiert werden. Hierzu gehören zum einen die Unit-Tests, zum anderen die System-Tests. Da System-Tests das Zusammenwirken aller Systemkomponenten testen und damit auch die Software-Implementierung in Kapitel 6 betreffen, werden die System-Tests erst in Kapitel 7 behandelt. Im Folgenden wird auf die Realisierung der für die Entwicklung des HDL-Design notwendigen Unit-Tests eingegangen. Das Bild 5.2 zeigt hierfür den Aufbau des HDL-Design mit den verschiedenen Systemblöcken und den entwickelten Testbenches.

Ethernet

Zur Unterstützung der testgetriebenen Entwicklung des Ethernet-MACs, Ethernet-Stacks und die hiervon abhängigen Applikations-Komponenten, ist eine Testbench vor der eigentlichen Implementierung entwickelt worden, die das MII-Interface zwischen externem PHY-Baustein und dem Ethernet-MAC simuliert (siehe Bild 5.2 im unteren Bereich). Diese Testbench ist auf Basis einer von Xilinx zur Verfügung gestellten Testbench entwickelt [60], die zum Testen

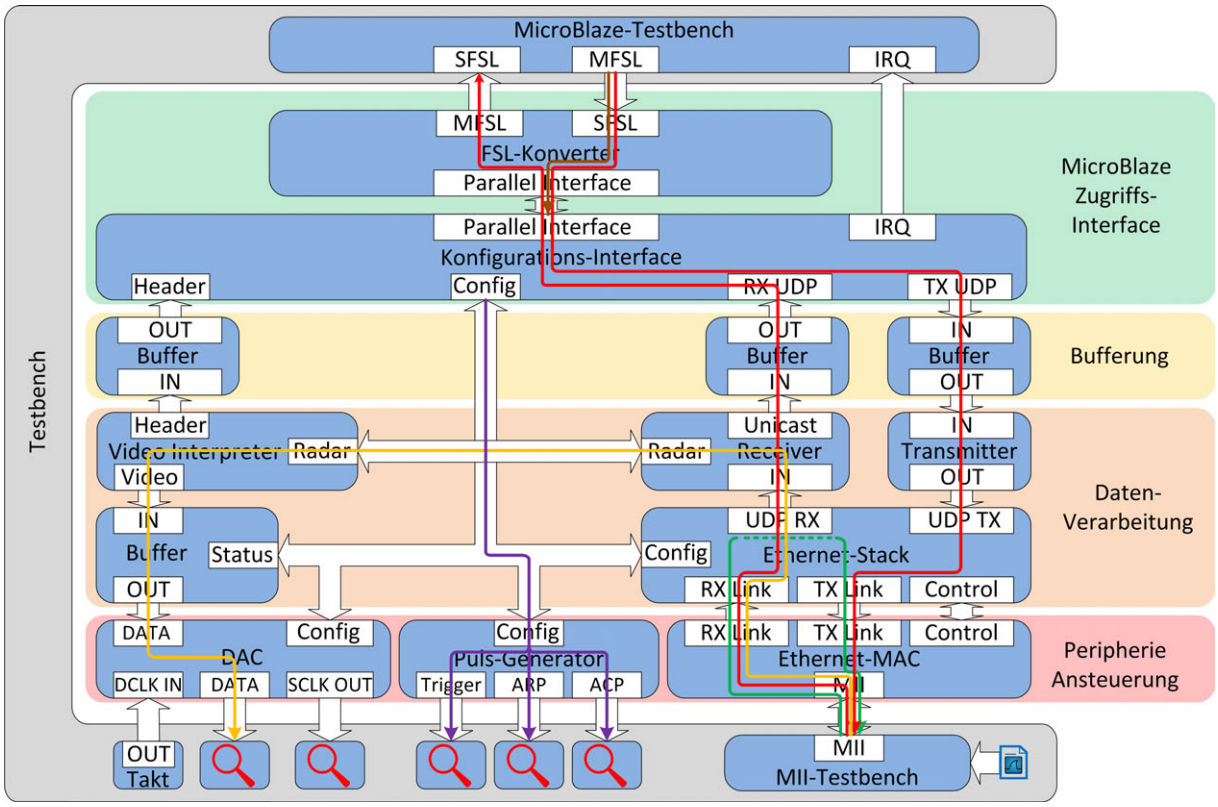


Bild 5.2.: Aufbau des HDL-Designs anhand der einzelnen Systemblöcke (einschließlich der entwickelten Testbench)

des Ethernet-MAC “XPS LL TEMAC” gedacht ist. Diese Ausgangs-Testbench ist in der Lage, statisch in einem Array hinterlegte Ethernet-Nachrichten über die MII-Schnittstelle zu verschicken. Präambel, Start-Frame-Delimiter, sowie die CRC-Summe werden zur Simulationszeit dynamisch generiert und den statischen Daten hinzugefügt.
Um eine größere Bandbreite an Unit-Tests durchführen zu können, wie etwa das Testen einzelner Protokoll-Interpreter-Module des Ethernet-Stacks oder aber auch Komponenten der Applikationsebene, deren Testvektoren die Signale des Ethernet-Stacks sind oder auf diese zurückgeführt werden können, besteht der Anspruch, eine deutlich flexiblere und teils automatisierte Testbench einzusetzen. Um diese Vielzahl an möglichen Unit-Tests zu ermöglichen, ist es notwendig, dass die Testbench verschiedenste Nachrichten, basierend auf unterschiedlichen Protokollen, simulieren kann. Dadurch können nicht nur die einzelnen Protokoll-Verarbeitungen im Ethernet-Stack (grüner Pfad in Bild 5.2), sondern z. B. auch die Verarbeitung der Video-Nachrichten von den Radar-Transceivern getestet werden. Mit Letzterem ließen sich Unit-Test in der Applikationsebene vom Video-Interpreter (siehe Abschnitt 5.6.2), über den Video-Buffer (siehe Abschnitt 5.6.1) bis hin zur DAC-Ansteuerung (siehe Abschnitt 5.6.3) durchführen (gelber Pfad in Bild 5.2).
Hierzu wird die existierende Testbench dahingehend erweitert, dass .txt-Dateien automatisiert eingelesen werden können, die aus dem Netzwerk-Analysator Wireshark [47] exportierte Daten beinhalten. Die Daten werden dann in der Testbench automatisch um die Präambel, den SFD und die CRC-Summe ergänzt. Diese sind nicht Bestandteil der durch Wireshark exportierbaren Daten. Damit wurde die Möglichkeit geschaffen, jeden beliebigen realen Netzwerkverkehr im beliebigen Umfang simulativ zu reproduzieren.
Die Testbench muss hierzu eine durch Wireshark exportierte Datei einlesen und deren Inhalt

interpretieren können. Letzteres ist im Wesentlichen das Erkennen von Anfang und Ende der einzelnen Nachrichten, das Erkennen der ASCII-Zeichen, die einer Nachricht zuzuordnen sind und die Konvertierung dieser Daten von ASCII-Hex-Werte in Binärwerte. Letzteres ist mittels VHDL-Funktionen realisiert. Bei der Realisierung ist bewusst darauf verzichtet worden, auch das ursprüngliche reale Timing zwischen zwei Ethernet-Paketen simulativ zu reproduzieren. Stattdessen wird eine anpassbare, statisch definierte Pause zwischen jeder Nachricht durchgeführt. Würde das Original-Timing simuliert werde, kann dies mitunter zu sehr langen, dann aber notwendigen, Simulationszeiten führen.
Für Nachrichten, die vom zu implementierenden Design an den PHY-Baustein gesendet werden, ist ein Prozess implementiert, der die Datenkonsistenz dieser Nachrichten überprüfen kann. Hierzu müssen die gesendeten Nachrichten in den ersten 32-Bit eine nach dem Iperf-Verfahren mit jeder Nachricht zu inkrementierende Zahl beinhalten. Diese Zahl wird von der Testbench ausgelesen und überprüft. Das Ergebnis wird im Command-Line-Fenster in Text-Form dargestellt. Auf diese Weise kann automatisiert überprüft werden, ob die Nachricht prinzipiell richtig ist oder ob es zum Datenverlust kam.
Um die Testbench besser der Realität anzunähern, weisen die verschiedenen Takte in der Testbench ein Pseudo-Zufalls-Jittern auf, welches nach [21] implementiert wurde.
Dem Anhang A.3.5 ist ein Tutorial zu entnehmen, welches das Exportieren von Daten aus Wireshark für die Nutzung durch diese Testbench beschreibt.

FSL-Verbindung / MicroBlaze

Um alle Schnittstellen des HDL-Designs simulativ abzudecken, besteht neben den Testvektoren für die MII-Schnittstelle auch die Notwendigkeit die FSL-Schnittstelle zwischen MicroBlaze und HDL-Design zu simulieren. Neben der Simulation des Signalverhaltens nach der FSL-Spezifikation [50] für lesende und schreibende Zugriffe muss die Testbench auch das in der Entwicklungsdokumentation im Anhang A.3.1 definierte Kommunikations-Protokoll bei Zugriffen einhalten, welches gemäß der Konzeptionierung in Abschnitt 4.2.4 notwendig ist um Memory-Mapped-Zugriffe zu ermöglichen. Auf diese Weise kann nicht nur die FSL-Verbindung an sich simuliert werden, sondern auch das Verhalten des MicroBlazes, welcher über diese Schnittstelle auf das Design zugreift. Um dieses Verhalten zu simulieren, sind zwei Aktions-Ketten mit Steueranweisungen in der Testbench realisiert.

- Die erste Kette von Aktionen (brauner Pfad in Bild 5.2) wird direkt nach dem Reset abgearbeitet und führt zu einer Initialisierung der Register im Konfigurations-Interface des HDL-Designs.
- Die zweite Aktions-Kette wird jedes Mal abgearbeitet, wenn beim Empfang einer Unicast-UDP-Nachricht an den konfigurierten UDP-Port ein Interrupt ausgelöst wird. In dieser Kette wird als Erstes der Interrupt durch die Testbench zurückgesetzt, um weitere Nachrichtenempfänge detektieren zu können. Daraufhin folgt das Auslesen der empfangenen Nachricht. Diese Nachricht wird automatisiert überprüft, sofern es sich um eine Nachricht handelt, die in den ersten 32 Bit nach dem Iperf-Verfahren einen Identifier beinhaltet. Über eine Text-Ausgabe wird automatisch mitgeteilt, ob es zu einem Verlust einer Nachricht gekommen ist oder nicht. Anschließend wird dieser Iperf-Identifier zusammen mit weiteren statisch im Array der Aktions-Kette hinterlegten Daten in den

Sende-Buffer geschrieben und ein Sendevorgang ausgelöst. Diese zweite Aktions-Kette erlaubt damit das Testen der Systemblöcke entlang des roten Pfades in Bild 5.2.

Die Anweisungen für beide Aktionsketten sind in C-ähnlichen Strukturen unter Verwendung von VHDL-Records statisch abgelegt. Mithilfe der Laufvariablen von Schleifen können so die jeweiligen Informationen aus dem Array ausgelesen und die Hardware-Software-Schnittstelle entsprechend angesteuert werden.
Durch die zweite Aktionskette schließt sich ein Kreis: Werden durch die MII-Testbench Iperf-Nachrichten verschickt, werden diese durch den Ethernet-MAC und Ethernet-Stack verarbeitet und bei fehlerfreier Funktion in den UDP-RX-Ringbuffer abgelegt. Die FSL-Testbench holt automatisch diese Werte über die FSL-Verbindung ab - wofür das in Hardware realsierte FSL-Interface fehlerfrei funktionieren muss - und analysiert den Iperf-Identifier mit entsprechender anschließender Text-Ausgabe des Ergebnisses. Der Identifier wird zurück in den UDP-TX-Ringbuffer geschrieben und über den Ethernet-Stack und -MAC verschickt. Die im vorherigen Abschnitt implementierte MII-Testbench kann den IPERF-Identifier ebenfalls auslesen und überprüfen. Melden beide Testbench-Teile keine Fehler, konnte das für diesen Fall fehlerfreie Zusammenwirken verschiedener Komponenten in Sende- und Empfangsrichtung nachgewiesen werden.
Zu beachten ist, dass durch diesen automatisierten Teil zwar ein Test mit einer sehr langen Kausalkette durchgeführt wird, eine vollständige Testabdeckung jedoch bei weitem nicht realisiert ist. Die Implementierung weiterer automatisierter Auswertungen von Tests ist zwar prinzipiell möglich, verspricht bei einem Projekt dieser Größenordnung gegenüber einer manuellen Auswertung aber keinen zeitlichen Vorteil.

5.2. MicroBlaze

Die Konzeptionierung in Abschnitt 4.2.1 sieht den Einsatz eines MicroBlaze vor. Die Aufgabe dieses konfigurierbaren 32-Bit RISC Mikrocontrollers besteht primär in der Generierung der Pulse Trigger, ARP und ACP, sekundär in der Konfiguration und Steuerung des HDL-Designs. Letzteres soll dabei auch durch den Empfang von Steuerinformationen über das in Hardware implementierte Ethernet-Interface initiiert werden können.
Für die Einbindung des MicroBlaze in ein Projekt existieren zwei verschiedene Varianten:

- Der MicroBlaze wird als übergeordnete Einheit interpretiert, der verschiedene IP-Cores untergeordnet werden können. In diesem Fall geht die Entwicklung vom “Xilinx Platform Studio” (EDK) aus, über welches der MicroBlaze konfiguriert wird. Einzelne IP-Cores können direkt im EDK oder unter Einsatz des “ISE Design Navigator” entwickelt werden. Zu einem solchen IP-Core könnte etwa das Ethernet-Interface als Ganzes zählen.
- Der MicroBlaze wird als untergeordnete Komponente interpretiert, indem dieser als HDL-Komponente in ein HDL-Design eingebunden wird. Die Design-Entwicklung geht in diesem Fall vom “ISE Design Navigator” aus. IP-Cores müssen dann nicht mehr zwangsläufig dem MicroBlaze untergeordnet werden, auch wenn diese Möglichkeit weiterhin besteht. Das EDK wird so nur bei der Konfiguration des MicroBlaze oder bei der Änderung eines untergeordneten IP-Cores benötigt.

In verschiedenen Veröffentlichungen, wie [74, 75], wird auf Design-Prinzipien zur VHDL-Codierung eingegangen, die zur Erstellung performanter HDL-Designs, und zur Verbesserung der Reproduzierbarkeit von Synthese- und Implementierungsergebnissen führen sollen. Eine der dort genannten Empfehlungen ist der hierarchische Entwurf eines HDL-Designs. Die Synthese- und Implementierungstools werden hierdurch unterstützt, indem diese zusammengehörende Logik so besser optimieren und implementieren können.
Ein besserer hierachischer Aufbau kann durch die zweite Variante erzielt werden. Dem MicroBlaze sind so keine IP-Cores untergeordnet, die einen sehr hohen Ressourcenverbrauch aufweisen. Dies wäre bei der ersten Lösung, nach Abschätzung der Ressourcenverbräuche in den Voruntersuchungen, hingegen der Fall.
Der in der hierarchischen Strukturierung sehr flach gehaltene MicroBlaze wird hierdurch hierarchisch vollständig separiert. Er kann so besser als eine abgeschlossene Einheit interpretiert und als diese optimiert und implementiert werden. Da dieser während der Entwicklung nur einmal konfiguriert werden muss und keine sich fortlaufend ändernden IP-Cores dem MicroBlaze untergeordnet sind, verändert sich dieser Hierarchie-Zweig kaum. Es ergibt sich damit die Möglichkeit diesen als Partition in einem inkrementellen Design-Flow zu definieren. Diese Möglichkeit hätte bei der ersten Variante so nicht bestanden. Ein inkrementeller Design-Flow mit ausschließlicher Partitionierung des MicroBlaze reduziert die Rechenzeit für einen Design-Flow in dem hier erstellten Projekt um etwa 24 %.

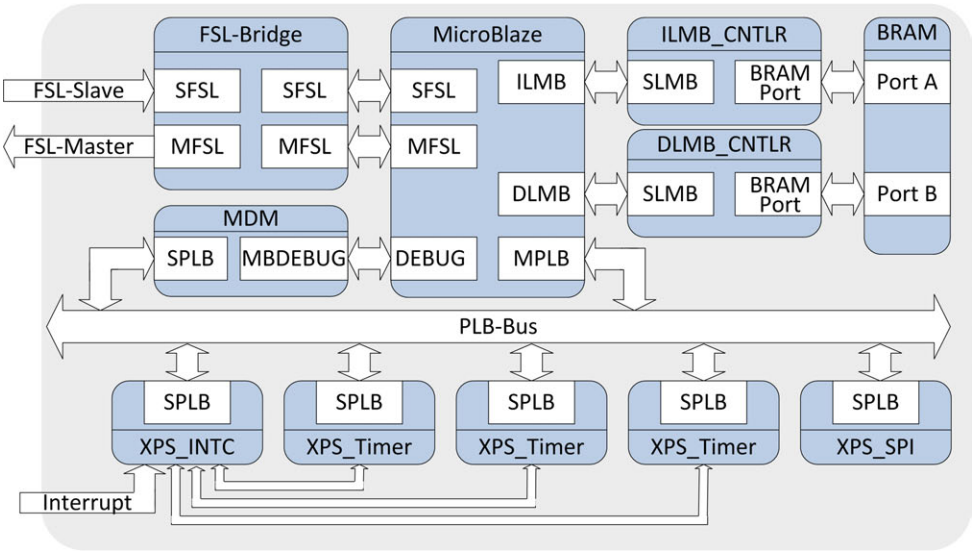


Bild 5.3.: Interner Aufbau der MicroBlaze-Komponente

Die Konfiguration des MicroBlaze kann an die speziellen Anforderungen des eingebetteten Systems angepasst werden. Dem Bild 5.3 können die konfigurierten Peripherie-Elemente entnommen werden. Durch die ausschließliche Nutzung von lokalem BlockRAM sind Cache-Speicher nicht notwendig. Der BlockRAM ist am MicroBlaze über zwei “Local Memory Busse” (LMB) angebunden. Über den “Processor Local Bus“ (PLB) wird ein XPS Serial Peripheral Interface (xps_spi) und zusätzlich drei Timer (xps_timer) angebunden. Über das SPI-Interface kann der externe SPI-Flash angesteuert werden, um zur Laufzeit Zugriff auf das Konfigurations-Image des FPGAs zu erhalten (weiteres siehe Abschnitt 6.4). Die Timer mit zugehörigem Interrupt-Handler werden für die Bestimmung und Generierung der Ausgabezeitpunkte der

Radar-Pulse benötigt. Jedes Timer-Modul beinhaltet per Default zwei Timer, jedoch mit gemeinsamer Interrupt-Leitung. Dieser zweite Timer wird benötigt, wenn z. B. eine Pulsweiten-Modulation durchgeführt werden soll. Bei dem hier vorliegenden Anwendungsfall ist die Nutzung des zweiten Timers ungeeignet, da bei jedem Interrupt-Request geprüft werden müsste, welcher der Timer den Interrupt ausgelöst hat. Der zweite Timer wird daher wegkonfiguriert, um Ressourcen einzusparen. Dennoch benötigen die Timer im Mittel 28 % der vom MiroBlaze benötigten Logik-Ressourcen. Zur Verarbeitung der Timer-Interrupts und des vom HDL-Design generierbaren Interrupts wird der "XPS_INTC" Interrupt Controller konfiguriert, der ebenfalls über den PLB-Bus angesteuert wird. Neben einem Taktgenerator, um aus dem externen 200 MHz Takt des SP605-Evaluationboard einen mit 66.666 MHz gewählten Systemtakt zu generieren, wird noch ein MicroBlaze-Debug-Modul (MDM) konfiguriert, um die Entwicklung zu unterstützen. Alternativ zum PLB-Bus könnte auch der AXI-Bus gewählt werden. Der PLB-Bus wird für dieses Design gewählt, um die Kompatibilität zu FPGAs der Familien Spartan-3 und Virtex-5 zu bewahren. Performance-Verluste werden hierdurch nicht erwartet, da über diesen Bus keine höheren Datendurchsätze zu verarbeiten sind.
Dem MicroBlaze selbst wird ein hardwarebasierter Multiplizierer und ein Dividierer hinzukonfiguriert, um die Berechnung der Pulse beschleunigen zu können. Berechnungen mit Fließkommazahlen finden nicht statt, sodass die Floating-Point-Unit (FPU) nicht konfiguriert wird. Ferner ist der Barrel-Shifter hinzukonfiguriert, um Schiebe-Operationen zu beschleunigen.

5.3. Hardware-Software-Interface

Aufgabe des Hardware-Software-Interface ist es, dem MircoBlaze die Möglichkeit zu geben, einen schnellen Zugriff auf das HDL-Design durchführen zu können. Das Interface ist aus zwei Komponenten aufgebaut, wie das Bild 5.2 zeigt. Die erste Komponente stellt einen FSL-Konverter dar, der die Kommunikation der beiden Fast-Simplex-Links, vom MicroBlaze kommend, auf ein Parallel-Interface umsetzt. Die zweite Komponente ist das Konfigurations-Interface, welches dann abstrahiert von FSL-spezifischen Zugriffsverfahren arbeiten kann. Die Aufgabe des Konfigurations-Interfaces besteht darin, einen einheitlichen Zugriff auf das HDL-Design zu ermöglichen.
Durch die Separierung der beiden Komponenten besteht einfacher die Möglichkeit, das HDL-Design über eine andere Schnittstelle anzusprechen, wenn sich die Anforderungen an das Design ändern. Nachfolgend wird auf die konkrete Implementierung des FSL-Konverters und die des Konfigurations-Interface eingegangen.

5.3.1. FSL-Konverter

Um kurze Einzel-Zugriffszeiten und dennoch die Verarbeitung höherer Datendurchsätze zu ermöglichen, ist in der Konzeption in Abschnitt 4.2.4 entschieden worden, zwei unidirektionale Fast-Simplex-Links einzusetzen. Durch die Design-Entscheidung, den MicroBlaze in der Design-Hierachie zu separieren, besteht die Notwendigkeit, dass der Fast-Simplex-Link nicht auf einen "internen", dem MicroBlaze untergeordneten IP-Core geführt werden muss, sondern aus Sicht des MicroBlaze auf nach außen führende Ports zu geben ist. Hierfür ist ein IP-Core

implementiert worden, der das HDL-Design aus Sicht des MicroBlaze repräsentiert, indem dieser beide Fast-Simplex-Links direkt auf die Ausgangspins führt (siehe Bild 5.3).
Auf der Seite des Hardware-Designs ist damit ein Master- und ein Slave-Interface notwendig, welches die FSL-Streaming-Verbindung so umsetzt, dass ein Zugriff auf Register und Speicherinhalte möglich wird. Die Ansteuerung durch den MicroBlaze geschieht dabei durch ein die Kommunikation initiierendes 32-Bit Daten-Wort. Dieses enthält Informationen über Zugriffsadresse, Zugriffsart und Anzahl der zu übertragenden Daten-Worte. Der Entwicklungsdokumentation im Anhang A.3.1 kann der konkrete Aufbau dieses Daten-Worts entnommen werden.
In Bild 5.4 ist der Zustandsautomat dargestellt, der im Hardware-Design die Kommunikation

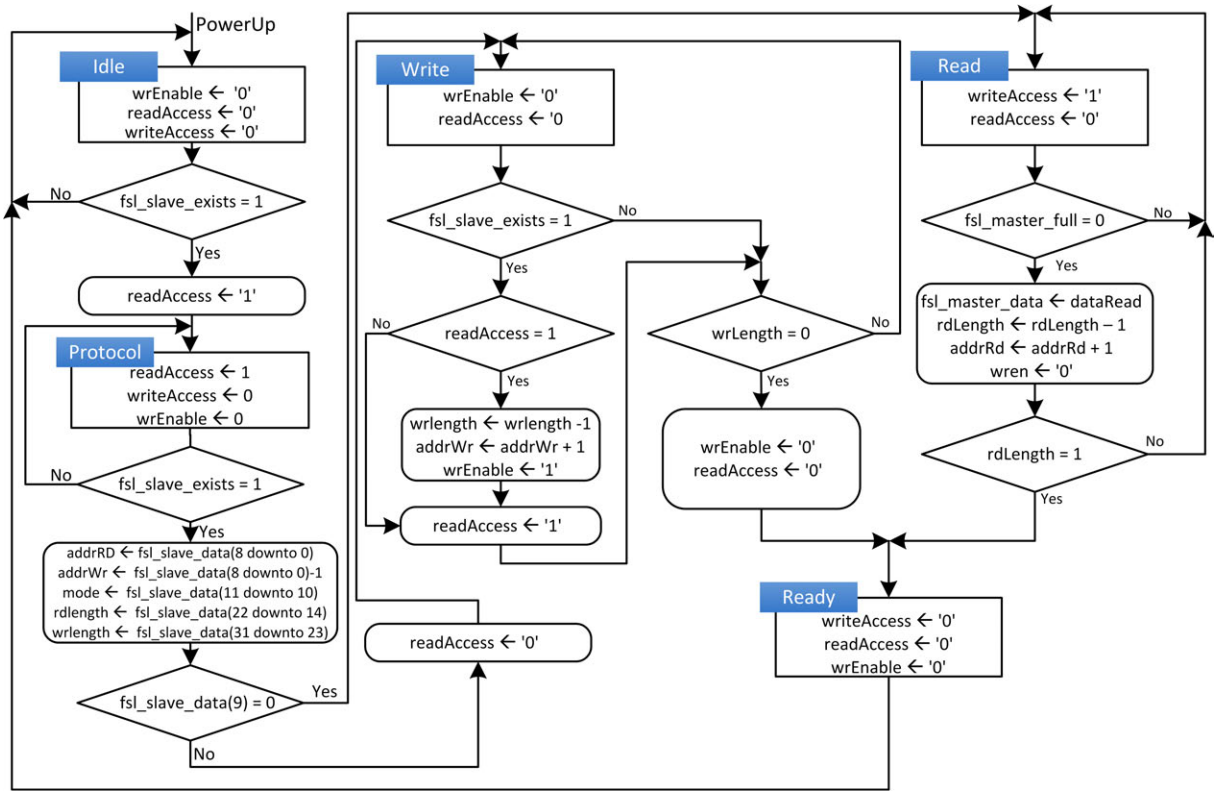


Bild 5.4.: ASM-Diagramm für FSM zur Umsetzung von zwei unidirektionalen Fast-Simplex-Link auf ein Parallel-Interface

mit den beiden Fast-Simplex-Links, die Interpretation des Protokolls und die notwendige Umsetzung auf ein Parallel-Interface steuert und der im Folgenden erläutert wird:

“Protocol”-Zustand:

Sobald der MicroBlaze das Signal “fsl_slave_exists” auf High zieht, liegen Daten vom MicroBlaze zur Abholung bereit und der MicroBlaze wechselt vom “Idle”-Zustand in den “Protocol”-Zustand und setzt das Signal “readAccess”. Dieses Signal wird kombinatorisch UND-verknüpft mit “fsl_slave_exists” und auf auf das Signal “fsl_slave_read” geführt. Dem MicroBlaze wird so mitgeteilt, dass die Daten gelesen wurden. Die kombinatorische UND-Verknüpfung ist notwendig, da nach der FSL-Spezifikation [50] “fsl_slave_read” nicht erst um einen Takt verzögert zurückgenommen werden darf. Nach dem ersten empfangenen Datenwort nimmt der Zustandsautomat das “fsl_slave_read” zurück. Die FSM hat damit die ersten 32-Bit gelesen, in denen die Art der nachfolgenden Kommunikation angegeben ist. Solange “fsl_slave_read” zurückgesetzt bleibt, wird der

MicroBlaze, bzw. der eingesetzten FSL-FiFo, keine weiteren, möglicherweise bereits im FiFo vorliegenden, Daten übergeben. Abhängig von diesen ersten 32-Bit der Kommunikation, wechselt die FSM in den Zustand “Write”, wenn der MicroBlaze Daten schreiben will oder in den Zustand “Read”, wenn dieser Daten lesen möchte. Informationen wie die zu lesende/schreibende Anzahl an 32-Bit-Werten und die Startadresse werden zwischengespeichert.

“Write”-Zustand:

Im “Write”-Zustand wird mit jedem Takt abgefragt, ob weitere Daten zum Lesen zur Verfügung stehen. Sobald dies der Fall ist, wird der Read-Access wieder gesetzt und mit der nächsten Taktflanke die ersten Daten gelesen. Solange der “ReadAccess” gesetzt bleibt, werden mit jedem Takt die Daten vom MicroBlaze zum Hardware-Design übertragen. Während der Datenübertragung wird das Enable-Bit des Parallel-Interface für Schreibzugriffe gesetzt und die Adresse an das Konfigurations-Interface übergeben. Die Adresse wird hierzu von der im Header angegebenen Startadresse ausgehend mit jedem Datenwort inkrementiert. Die Datenleitungen der FSL-Verbindung können wegen der Nutzung des Enable-Bits direkt auf das Parallel-Interface geführt werden.
Indem die übertragende Anzahl an 32-Bit-Werten mit jedem Datenwort dekrementiert wird, erfolgt der Wechsel in den “Ready”-Zustand, sobald alle Daten gelesen wurden. Kann der MicroBlaze die Daten nicht schnell genug nachliefern, wird dieser das “fsl_slave_exists”-Signal zurücksetzen, sodass die State-Machine auf die nächsten Daten wartet.

“Read”-Zustand:

Wird nach dem “Protocol”-Zustand in den “Read”-Zustand gewechselt, wird das “writeAccess”-Signal gesetzt. Dieses wird UND-verknüpft mit der Negation des “fsl_master_full”, vom MicroBlaze kommend, auf das “fsl_master_write”-Signal gegeben, welches dem MicroBlaze mitteilt, dass Daten anliegen. Das “fsl_master_write”-Signal wird damit sofort zurückgenommen, wenn der MicroBlaze, bzw. bei dieser Implementierung der zwischengeschaltete FSL-FiFo, die Daten nicht mehr annehmen kann. Solange Daten übertragen werden können, wird die Lese-Adresse beginnend ab der übergebenen Startadresse inkrementiert und die Anzahl an zu übertragenden 32-Bit-Werten dekrementiert. Die Daten werden hier nicht direkt, sondern über Register geführt, an den MicroBlaze übergeben. Sind alle Daten übertragen worden, wird über den Ready-Zustand zurück in den Idle-Zustand gewechselt.

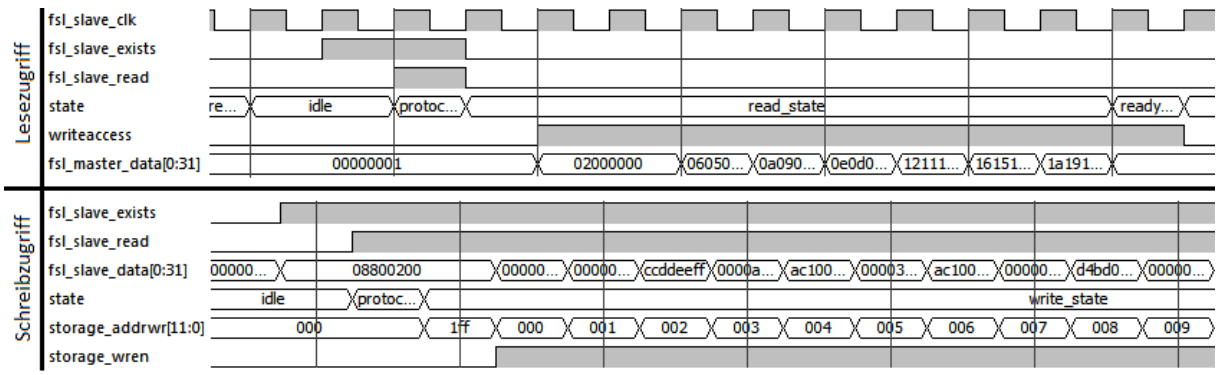


Bild 5.5.: Lese-/Schreibzugriff von der Software- auf die Hardware-Ebene

Gemäß der Verhaltenssimulation (siehe Bild 5.5) ist, von Seiten der Hardware, ein Einzelzugriff innerhalb von 4 Takten vollständig durchführbar, sofern der MicroBlaze dies schnell genug verarbeiten kann. Durch eine zusätzliche Latenz aufgrund der eingesetzten FSL-FIFOs werden Lese-Vorgänge mehr Takte benötigen, da hier auf die Reaktion des HDL-Designs gewartet werden muss.
Ist mehr als ein Datenwort zu übertragen, erhöht sich die Zugriffszeit mit jedem 32-Bit-Datenwort um einen weiteren Takt.

5.3.2. Konfigurations-Interface

Beim Konfigurations-Interface handelt es sich um eine HDL-Komponente die einen einheitlichen, abstrahierten Zugriff des MicroBlaze auf das HDL-Design ermöglicht. Das Konfigurations-Interface selbst wird dabei über ein Parallel-Interface angesprochen. Anhand des adressierten Speicherbereiches wird auscodiert ob Lese-/Schreibzugriffe auf Konfigurations-Register, Lesezugriffe auf den selektierten UDP-Empfangs-Speicherblock oder Schreibzugriffe auf den selektierten UDP-Sende-Speicherblock erfolgen sollen. Die Adressauscodierung geschieht dabei kombinatorisch.
Das Konfigurations-Interface verwaltet damit alle notwendigen Daten, um das HDL-Design zu steuern, (Status-)Informationen abzufragen und Zugriffe auf die in Bild 5.2 gezeigten Buffer durchzuführen. Damit werden folgende Bereiche abgedeckt:

Ethernet-Stack: Konfiguration und Verwaltung aller vier UDP-Interfaces (ein Unicast- und drei Multicast-Interfaces), Initiierung von An- und Abmeldungen an Multicast-Gruppen, Abfrage von MAC-Adressen beliebiger Netzwerk-Teilnehmer, Rx-Flow-Control u. v. m..

UDP-Nachrichten: Abfrage empfangener UDP/IP-Nachrichten die per Unicast an den konfigurierten UDP-Port gesendete wurden und Übergabe von Nachrichten zum Senden. Des Weiteren können Statusabfragen und Steuerungen der Ring-Buffer erfolgen.

Interrupt: Verwaltung und Generierung eines Sammelinterrupts mit Bereitstellung eines Status-Registers zum Auslesen der ursprünglichen Interruptquelle. Interrupts werden generiert, wenn eine neue UDP-Unicast-Nachricht oder ein ARP-Reply eintrifft.

Radar-Verarbeitung: Durch Zugriff auf den Videoheader-Ring-Buffer können Header-Informationen zu den Video-Sweeps ausgelesen werden, damit der MicroBlaze die Ausgabezeitpunkte der Pulse errechnen kann. Ebenso kann die Ansteuerung und Konfiguration der Pulsgeneratoren sowie die der DAC-Ansteuerung vorgenommen werden. Die Zustände der Ring-Buffer sind auch hier abfragbar und die Buffers selbst steuerbar.

Reboot: Ansteuerung der ICAP-Ansteuerungs-FSM zur Durchführung eines Warm-Reboot des FPGAs.

Auf die einzelnen Funktionen wird in den folgenden Abschnitten detaillierter eingegangen.

5.4. Ethernet-MAC

Ausgangsbasis für die Implementierung des Ethernet-MAC stellt der EthMAC 10/100 von OpenCores [23] dar. Die Voruntersuchungen haben gezeigt, dass durch Implementierung eines HDL-Wrappers dieser Ethernet-MAC auch in Verbindung mit einem in Hardware realisierten Ethernet-Stack eingesetzt werden kann. Aufgrund der im EthMAC integrierten DMA-Engine und der Verwendung der Wishbone-Busse ergibt sich durch die Anpassung an einen in Hardware realisierten Ethernet-Stack ein suboptimales Design. Der Ressourcenverbrauch und die Leistungsfähigkeit des Designs könnten bei Verzicht auf die DMA-Engine und die beiden Wishbone-Busse verbessert werden.
Abschnitt 5.4.1 geht daher auf die durchgeführten Optimierungen und das sich hieraus ergebende neue Schnittstellenverhalten des EthMAC ein. Abschnitt 5.4.2 thematisiert die Notwendigkeit der Konfiguration des PHY-Bausteines. Abschließend wird in Abschnitt 5.4.3 auf zwei weitere Ethernet-MACs eingegangen, für die zu Testzwecken ein VHDL-Wrapper zur Anbindung an den Ethernet-Stack entwickelt wurde.

5.4.1. Implementierung des EthMAC

Ziel der Implementierung ist zum einen, die Minimierung der Logik-Ressourcen und der Verzicht auf Speicher-Ressourcen, zum anderen, eine Anbindung zu ermöglichen, die absolut deterministisch ist, Full-Duplex-Übertragungen zulässt und keine unnötig hohe Taktfrequenz des Ethernet-Stacks erfordert.

Optimierung des Ethernet-MAC

Zur Performance-Steigerung sind folgende Veränderungen vorgenommen worden:

- Der Wishbone-Bus zur Konfiguration und Steuerung der DMA-Engine und der internen Register des Ethernet-MAC wurde entfernt und durch ein performanteres Parallel-Interface ersetzt (siehe Bild 5.6, vgl. Bild 3.5). Dies reduziert die Logikressourcen zur Beschreibung der Wishbone-Interfaces auf der Seite des Wrappers sowie auf der Seite des Ethernet-MAC. Zugriffe werden zudem beschleunigt, da Verzögerungen durch den Bus vermieden werden. Beim Wishbone-Bus entsteht etwa eine Verzögerung durch ein angewendetes Handshake-Verfahren, bei dem jeder Zugriffs-Zyklus des Master-Teilnehmer durch den Slave-Teilnehmer zu bestätigen ist [22].
- Der zweite Wishbone-Bus und damit auch die vollständige DMA-Engine sind entfernt worden. Dies hat zur Folge, dass weder im Ethernet-MAC, noch im Wrapper eine Zwischenspeicherung von Daten notwendig ist. Der Verbrauch an Logik- und Speicher-Ressourcen minimiert sich damit erheblich. Die aufwendige Steuerung der DMA-Deskriptoren im Wrapper, wie auch im Ethernet-MAC, entfällt. Anstelle des zweiten Wishbone-Interfaces ist ein Interface zur Übergabe der Daten implementiert worden, welches im folgenden Abschnitt erläutert wird.

- Nach Abschnitt 5.1.2 kann bei FPGAs prinzipiell auf ein Power-On-Reset verzichtet werden. Beim Ethernet-MAC sind daher alle Power-On-Resets, mit Ausnahme der Resets, die das MIIM-Modul und dessen Ansteuerung betreffen, entfernt und durch Signalinitialisierungen ersetzt worden. Die Ausnahme wurde eingeführt, da der externe PHY-Baustein ein Reset-Signal benötigt. Durch einen hieraus abgeleiteten Reset für das MIIM-Modul wird sichergestellt, dass die Initialisierung erst nach dem Reset des PHY-Baustein erfolgt. Die Anzahl an benötigten Logik-Slices konnte ausschließlich durch diese Maßnahme um 11.8 % reduziert werden.
- Gegenüber der Vorversion ist der Wrapper deutlich verkleinert worden (siehe Bild 5.6). Ein Zustandsautomat übernimmt hier in der Startphase die Konfiguration des EthMAC und PHY-Baustein (siehe Abschnitt 5.4.2) nach einem Power-Up. Es wurde hier mit VHDL-Records gearbeitet, sodass mit Hilfe von Signalinitialisierungen getrennt von der eigentlichen FSM die Werte vorgegeben werden, die dann durch die FSM zu konfigurieren sind. Dies erhöht die Übersichtlichkeit wesentlich. Ein weiterer Zustandsautomat verarbeitet die Sende-Anfragen durch den Ethernet-Stack.

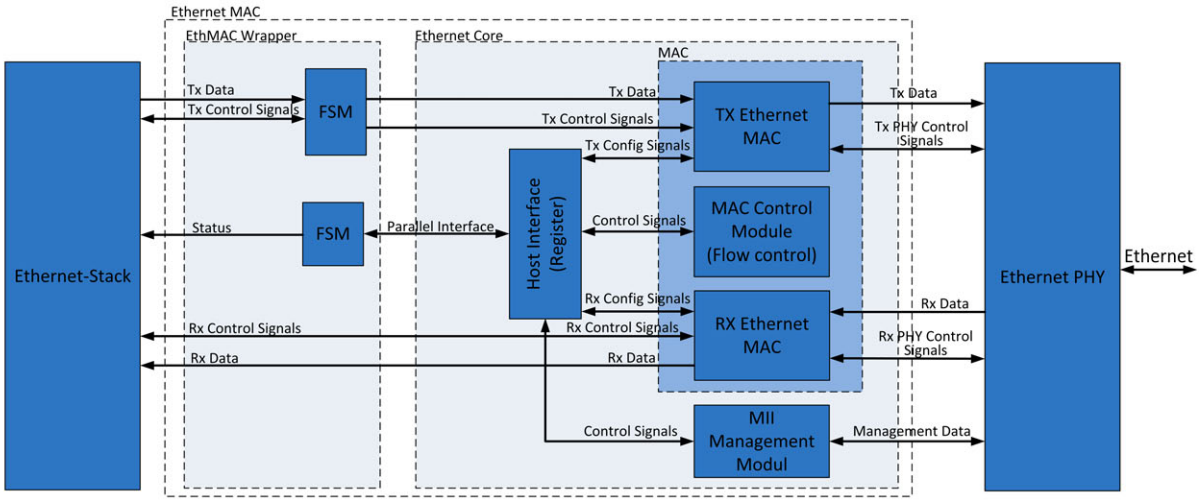


Bild 5.6.: Optimierte Version des Ethernet-MAC EthMAC von OpenCores

Durch diese Optimierungen konnten die benötigten Logik-Ressourcen um insgesamt 70 % im Vergleich zu der Version der Voruntersuchungen reduziert werden. Gegenüber dem “Tri-Mode Ethernet-MAC” [25] werden 60 % weniger Logik-Ressourcen verbraucht und gegenüber der Soft-IP-Variante des “XPS LL TEMAC” rund 84 % weniger Logik-Ressourcen benötigt. Durch die nicht mehr notwendige Zwischenspeicherung von Daten werden keine Speicher-Ressourcen und damit keine BlockRAM-Blöcke benötigt. Dies wirkt sich auch auf die Verzögerung der Daten aus. Bei Ethernet-MACs, welche Nachrichten zwischenspeichern beträgt diese mindestens die Übertragungszeit eines Ethernet-Paketes. Bei z. B. einem 1518 Byte großen Ethernet-Paket und Verwendung des MII-Interface für Fast-Ethernet wären dies mindestens 3036 Takte. Durch die Optimierung beträgt die Latenz zwischen Ein- und Ausgang des Ethernet-MAC nun immer 3 Takte, bezogen auf die Taktfrequenz von 25 MHz entspricht dies 120 ns. Dies ist unabhängig von der Länge der Ethernet-Nachricht. Durch die vollständige Trennung des Sende- und Empfangspfades ist nun auch eine echte Full-Duplex-Übertragung möglich, d. h. in beide Richtungen kann zeitgleich die Übertragung von Daten mit maximaler Geschwindigkeit erfolgen, ohne das erhöhte Taktfrequenzen in den höheren OSI-Schichten benötigt werden. Trotz der

Änderung werden weiterhin, neben FullDuplex, auch Flow-Control, die Filterung von MAC-Adressen, die Verarbeitung von Broad- und Multicast-Nachrichten und die Ansteuerung des MII-Management-Interfaces unterstützt.

Schnittstellenverhalten

Die Schnittstellen zwischen Ethernet-MAC und Ethernet-Stack sind universell definiert worden. In Empfangs-Richtung (siehe Bild 5.7 im oberen Teil) wird durch den Ethernet-MAC ein Start-Of-Frame (SOF) zeitgleich mit dem ersten Datenbyte gesetzt. Alle folgenden Datenbytes liegen bei der fallenden Flanke des "Dataphase"-Signals stabil an. Das Ende wird durch ein End-Of-Frame (EOF) Flag während des letzten Datenbytes markiert.
In Sende-Richtung kann über ein Request-Signal das Senden durch den Ethernet-Stack einge-

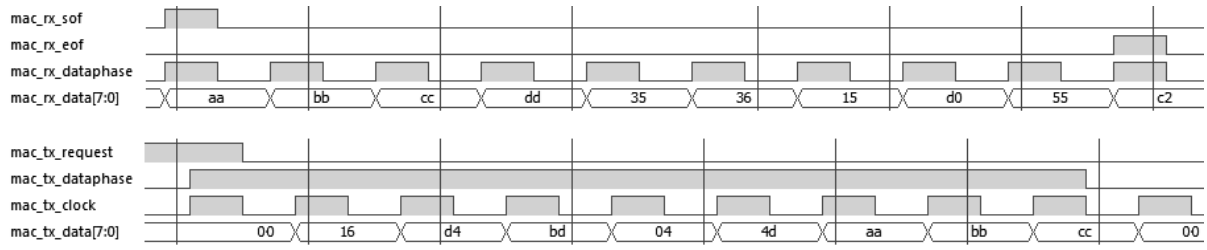


Bild 5.7.: Verkürztes Signaldiagramm der Signale der Ethernet-MAC-Schnittstelle in Empfangsrichtung (oberer Bildbereich) und Senderichtung (unter Bildbereich)

leitet werden, woraufhin der Ethernet-MAC mit der Generierung der Präambel und des Start-Frame-Delimiters beginnt (siehe Bild 5.7 im unteren Teil). Sobald die Daten vom Ethernet-Stack benötigt werden, wird das "Dataphase"-Signal gesetzt. Nach Setzen dieses Signals müssen bei den fallenden Flanken, des dem Ethernet-MAC zur Verfügung zu stellenden Taktsignals die Daten stabil anliegen. Das "Dataphase"-Signal wird automatisch zurückgenommen, wenn die Anzahl an Bytes, die parallel zum Setzen des Request-Signals übergeben wurde, erreicht ist.
Bei dem hier verwendeten MII-Interface werden die Daten mit einem Takt von 25 MHz übertragen. Die Breite des Datenbusses liegt hierbei bei 4 Bit. Der Ethernet-MAC setzt dieses Busbreite zwischen vier und acht Bit um und halbiert bzw. verdoppelt dabei die Taktfrequenz. In Empfangsrichtung ist dies problematisch, da keine Zwischenspeicherung der Nachrichten im Ethernet-MAC stattfindet und der Beginn einer Nachricht durch den anderen Kommunikationsteilnehmer bestimmt wird. Die Daten können dadurch bei steigender oder fallender Flanke des in der Frequenz halbierten Taktes stabil anliegen, wenn dieser Takt starr generiert wird. Bild 5.8 veranschaulicht dies grafisch. Anders als in Sende-Richtung ist es daher notwendig, dass sich das Taktsignal in der Phase auf das Paket synchronisiert. Für die Verarbeitung empfangener Daten muss daher das "Dataphase"-Signal verwendet werden, welches vom Ethernet-MAC generiert wird. Beim GMII-Interface kann hingegen der Empfangstakt gleich dem Dataphase-Signal gesetzt sein, da hier keine Bitbreitenumsetzung stattfinden muss.

5.4.2. Konfiguration des PHY-Baustein

Der EthMAC beinhaltet ein Modul zur Ansteuerung des PHY-Bausteins über das "Management Data Input/Output" Interface (kurz: MDIO, auch MIIM für "Media Independent Interface Ma-

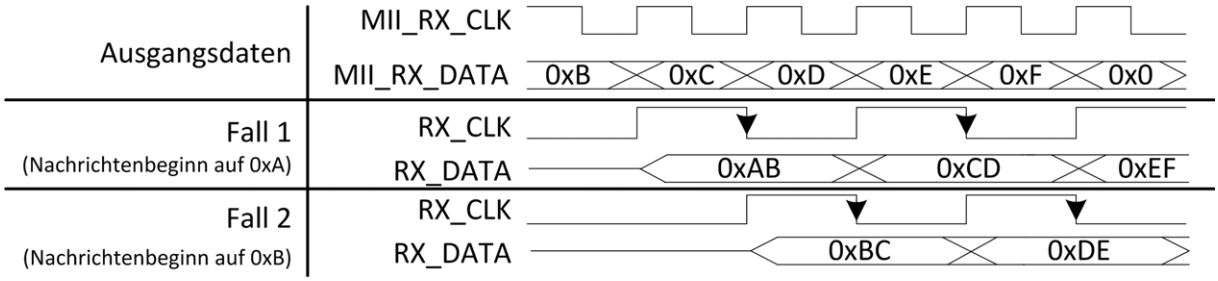


Bild 5.8.: Schematische Darstellung der Taktsynchronisation des Ethernet-MAC auf empfangene Daten

nagement"). Das Interface ermöglicht damit, auf interne Konfigurations- und Status-Register des PHY-Bausteins lesend und schreibend zuzugreifen. Dies ist notwendig, da der PHY-Baustein nach Abschnitt 2.2.1 den Layer 1 des Ethernet repräsentiert. Auf dieser Ebene muss bereits zwischen den Teilnehmern eine Kommunikation auf Basis des "Auto-Negotiation"-Protokolls stattfinden, mit dessen Hilfe u. a. die Übertragungsrate und, bei 10/100-MBit/s Übertragungen, auch der Übertragungsmodus (Half- oder Full-Duplex) zwischen den Teilnehmern ausgehandelt wird. Jeder Teilnehmer kann dabei eine oder mehrere Übertragungsraten bzw. Betriebsmodi anbieten, sodass sich auf die höchste, gemeinsame Übertragungsrate geeinigt wird. Dies ist notwendig, da in Abhängigkeit des Betriebsmodus die Art des Buszugriffes und auch die zu nutzende Schnittstelle zwischen Phy-Baustein und Ethernet-MAC zu wählen ist. Während bei 10/100 MBit/s-Übertragungen das MII-Interface zu nutzen ist, muss bei Gigabit-Ethernet auf das GMII-Interface umgeschaltet werden.
Die Konfigurations-Schnittstelle des Phy-Bausteins ist standardisiert, der Aufbau der internen Register des PHY-Baustein hingegen nicht. Der MIIM-Modul des Ethernet-MAC ermöglicht daher nur eine Abstraktion der Konfigrations-Schnittstelle zwischen PHY-Baustein und Ethernet-MAC, sodass die herstellerspezifische Konfiguration der internen Register des PHY-Baustein unabhängig vom Schnittstellenverhalten durchgeführt werden kann.
Die in Abschnitt 5.4.1 im Wrapper implementierte FSM konfiguriert so nicht nur den Ethernet-MAC, sondern auch den PHY-Baustein "Marvell 88E1111" [76] auf eine 100 MBit/s Full-Duplex-Übertragung. Darüber hinaus prüft die FSM periodisch, ob das Netzwerkkabel noch angeschlossen ist bzw. die physikalische Verbindung zum Teilnehmer am anderen Ende des Netzwerk-Kabels besteht. Dieser Status ist über das Konfigurations-Interface und damit durch den MicroBlaze auslesbar. In der Startphase nutzt der MicroBlaze diese Information, indem sich dieser erst initialisiert und mit der Abarbeitung beginnt, wenn eine physikalische Verbindung zum Netzwerk besteht.

5.4.3. Anbindbarkeit anderer Ethernet-MACs

Der in Abschnitt 5.5 entwickelte Ethernet-Stack benötigt die in Abschnitt 5.4.1 beschriebenen Signal-Schnittstellen. Prinzipiell lassen sich auch andere Ethernet-MACs einsetzen, die dieses Signalverhalten aufweisen oder sich bei abweichendem Verhalten durch einen HDL-Wrapper auf dieses Signalverhalten anpassen lassen. Die Nutzung eines anderen Ethernet-MACs kann für zukünftige Projekte von Interesse sein, deren Anforderungen vom Ethernet-Stack, jedoch nicht vom Ethernet-MAC erfüllt werden. Hierzu zählt etwa die Möglichkeit, Ressourcen auf

den Virtex-5-FPGAs einzusparen, indem der dort als Hard-IP-Core verfügbare "XPS LL TEMAC" genutzt wird oder, wenn die Anforderung besteht, einen Ethernet-MAC einzusetzen, der das GMII-Interface und damit eine Anbindung an ein Gigabit-Ethernet unterstützt.
Um einen Nachweis darüber führen zu können, dass der Ethernet-Stack auch für Gigabit-Übertragungen genutzt werden kann, sind zwei Wrapper für den "Tri-Mode Ethernet-MAC" [25] von OpenCores entwickelt worden. Einer für die Nutzung des MII-Interface und einer für die Nutzung des GMII-Interface:
Bei dem "Tri-Mode Ethernet-MAC" [25] handelt es sich um einen Ethernet-MAC für die Datenübertragungsraten 10 MBit/s, 100 MBit/s und 1000 MBit/s. Der Ethernet-MAC beinhaltet interne FiFo-Speicher, die durch den HDL-Wrapper mittels 32 Bit-breitem Bus abgefragt bzw. beschrieben werden können.

- Beim Senden besteht daher die Hauptaufgabe einer notwendigen FSM im HDL-Wrapper darin, Sende-Request vom Ethernet-Stack entgegenzunehmen, die Daten vom Ethernet-Stack anzufordern, diese von 8-Bit-Breite auf 32-Bit-Breite umzusetzen und im FiFo-Speicher unter Einhaltung der spezifizierten Zugriffsverhalten [25] abzulegen.
- In Empfangsrichtung teilt der Ethernet-MAC mit, wann Daten im FiFo-Speicher bereitliegen. Eine FSM erkennt dies, ruft die Daten ab und setzt diese auf 8-Bit-Breite um, bevor sie unter Einhaltung der Signalanforderungen nach Abschnitt 5.4.1 an den Ethernet-Stack übergeben werden.

Der Zugriff auf die FiFo-Speicher des Ethernet-MAC erfolgt so, dass der Ethernet-Stack einen Datendurchsatz von 8 Bit pro Takt bei einem 125 MHz Takt verarbeiten muss, wodurch der vom Ethernet-MAC verarbeitbare Datendurchsatz von 1 GBit/s unidirektional und 2 GBit/s bidirektional auch durch den Ethernet-Stack verarbeitet werden kann.
Für die testgetriebene Entwicklung des Wrappers ist eine zweite Testbench, auf Basis der in Abschnitt 5.1.4 entwickelten Testbench, entwickelt worden. Diese unterstützt anstelle des MII- das GMII-Interface. Auch hier können weiterhin Wireshark-Dateien als Datenquelle genutzt werden.

Um auch einen Nachweis der Kompatibilität zum "XPS LL TEMAC" erbringen zu können ist der Wrapper aus den Voruntersuchungen in Abschnitt 3.4.2 angepasst worden. Projekt-Subsets mit Verwendung dieser HDL-Wrappers liegen dem Anhang A.3.2 bei[4].

5.5. Ethernet-Stack

Die funktionalen Anforderungen nach Abschnitt 1.1 an einen zu entwickelnden Ethernet-Stack sind die Unterstützung des UDP- und IP-Protokolls für Datenkommunikationen, das ARP-Protokoll (Adress Resolution Protocol) zur Auflösung der physikalischen MAC-Adressen und damit einer Unterstützung von Unicast-Kommunikationen, sowie das IGMP-Protokoll (Internet Group Management Protocol) zur An- und Abmeldung an/von Multicast-Gruppen, wodurch der Empfang von Multicast-Nachrichten ermöglicht wird. Ferner ist das ICMP-Protokoll (Internet Control Message Protocol) zur Unterstützung der Echo-Funktion zu implementieren, mit der

[4]Das Beispiel-Projekt für den "XPS LL TEMAC" basiert auf einer Zwischenversion des entwickelten Ethernet-Stacks. Eine Inbetriebnahme mit der Endversion des Ethernet-Stacks konnte aus Zeitgründen und dem eingeschränkten Zugriff auf ein Evaluationboard mit Virtex-5-FPGA nicht mehr erfolgreich durchgeführt werden.

die Erreichbarkeit der Elektronik überprüft werden kann.
Der Aufbau des Ethernet-Stacks ist dabei so durchzuführen, dass eine Unterstützung weiterer Protokolle nachträglich implementiert bzw. eine bestehende Unterstützung entfernt werden kann. Der Ethernet-Stack muss dabei dynamisch zur Laufzeit konfigurierbar sein und mindestens zwei UDP/IP-Ethernet-Interfaces zeitgleich verarbeiten können. Zum einen ein Interface für Unicast-Nachrichten in Sende- und Empfangsrichtung zur Steuerung der Elektronik über Ethernet und zum Anderen ein Interface zum Empfang von Multicast-Nachrichten, welches für den Empfang von Video-Informationen des Radar-Transceivers notwendig ist.
Nichtfunktionale Anforderungen bestehen in der absoluten Zuverlässigkeit und Fehlerfreiheit bei einer zu verarbeitenden Datenübertragungsrate von mindestens 50 MBit/s in Sende- und Empfangsrichtung bei voller Unterstützung des FullDuplex-Modus. Die geforderte Übertragungsgeschwindigkeit ist hierbei auf das Gesamtsystem bezogen. Durch die Realisierung des Ethernet-Stacks in Hardware wird die Übertragungsrate ausschließlich durch den Datendurchsatz pro Takt und der maximal möglichen Taktrate definiert und kann daher prinzipiell höher liegen. Während der Durchsatz auf 8 Bit pro Takt festgelegt wird, da die kleinste Feldgröße in Ethernet-Protokollen ein Byte beträgt, ist eine maximale Taktfrequenz von 125 MHz als Ziel zu setzen. Gemäß Abschnitt 2.2.1 kann der Ethernet-Stack so auch in anderen Projekten für die Anbindung an ein Gigabit-Ethernet eingesetzt werden. Hierzu müsste dann z. B. der, über den in Abschnitt 5.4.3 entwickelte Wrapper, anbindbare “Tri-Mode Ethernet-MAC” genutzt werden.
Ziel ist es, dass der Ressourcenverbrauch im Bereich des Ethernet-Stack von ComBlock liegt. Dessen Ressourcenverbrauch wurde in Abschnitt 3.5 für die Abschätzung des Gesamt-Ressourcenverbrauchs des Ethernet-Interface verwendet und war damit für die Entscheidung einer Realisierung in Hardware mit verantwortlich.

5.5.1. Stack-Aufbau

Der Aufbau des Ethernet-Stacks geschieht nach dem in [33] empfohlen Modell zur Strukturierung digitialer Systeme, indem eine Aufteilung in Daten- und Steuerpfaden vorgenommen wird. Der Ethernet-Stack ist hierzu in drei Haupt-Komponenten unterteilt, die dem Bild 5.9 zu entnehmen sind. Während zwei Komponenten die beiden Datenpfade für das Senden und das

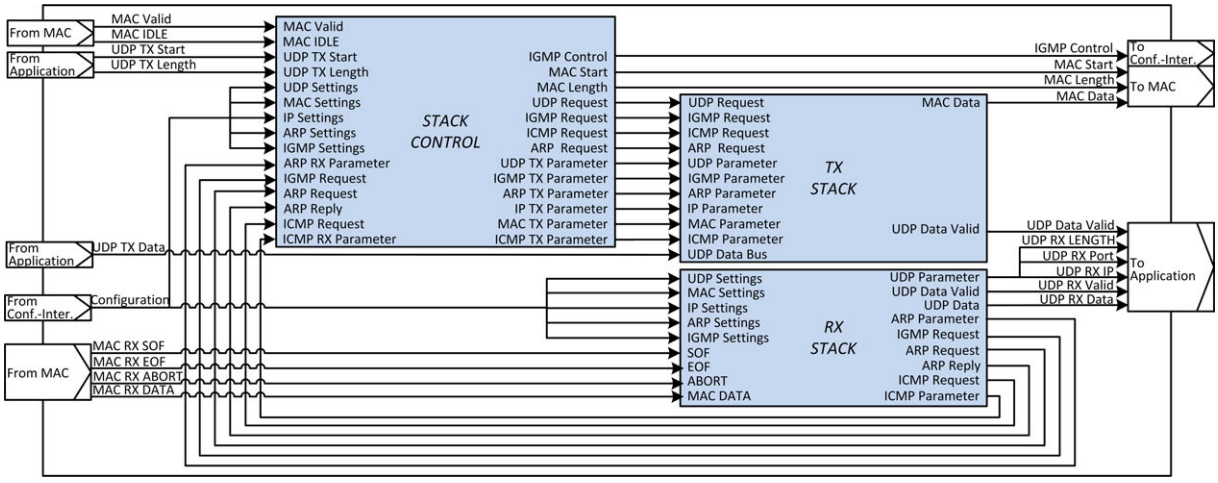


Bild 5.9.: Top-Layer des Ethernet-Stacks mit Rx- und Tx-Daten- und Steuerpfad

Empfangen beinhalten, befindet sich in der dritten Komponente der Steuerpfad. In der hierarchischen Strukturierung unterteilt sich der Steuerpfad in zwei Bereiche:

- Der erste Bereich ist übergeordnet und regelt die Zugriffssteuerung auf den Ethernet-Stack in Senderichtung. Damit verarbeitet dieser neben Sende-Anfragen durch die Applikationsebene auch die Sende-Anfragen als Resultat von Protokoll-Interpretationen im Empfangspfad und ist für einen ggf. notwendigen Informationsaustausch zwischen den Datenpfaden zuständig.
- Der zweite Bereich des Steuerpfades steuert den Protokoll-Aufbau der zu sendenden Ethernet-Pakete.

Eine Steuerung des Empfangspfades wird aus Gründen der zu erreichenden Geschwindigkeit nicht mittels FSM durchgeführt, sondern durch kombinatorische Logik gesteuert. Statusinformationen werden hingegen auch hier an den übergeordneten Steuerpfad übergeben.

Sende-Datenpfad

Im Prinzip besitzen alle Protokolle den gleichen Grundaufbau: Sie beginnen mit einem Header-Bereich und enden mit einem Datenbereich. In diesem Datenbereich können entweder andere Protokolle integriert, wie z. B. beim IP-Protokoll das IGMP-Protokoll oder Nutzdaten abgelegt werden, wie etwa beim UDP-Protokoll. Protokolle wie das ARP-Protokoll beinhalten hingegen keinen Datenbereich.

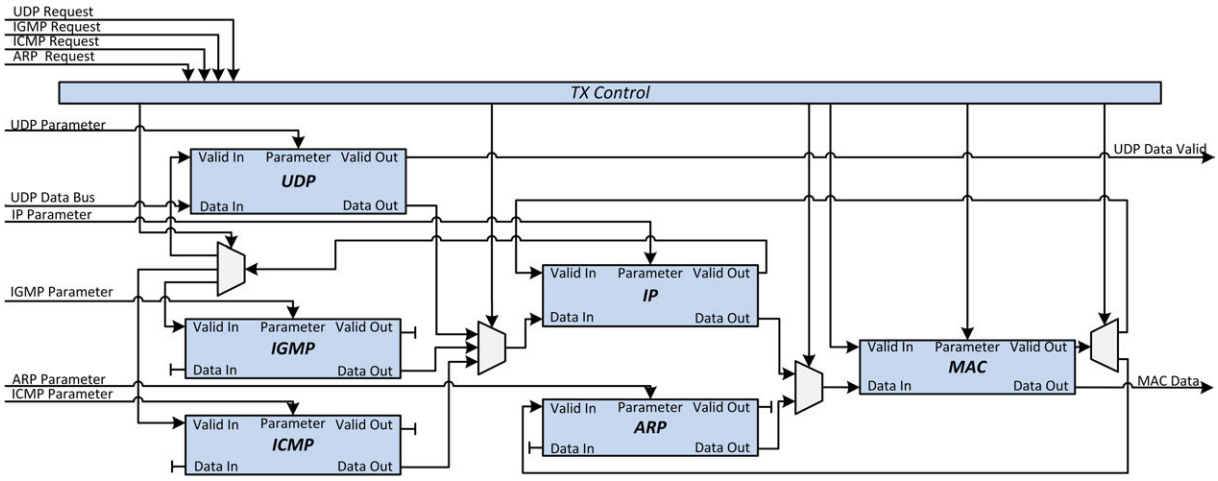


Bild 5.10.: Schematische Darstellung der Sende-Einheit für den Protokollaufbau im Ethernet-Stack

Jedes Ethernet-Datenpaket setzt sich, neben dem MAC-Frame damit aus mindestens einem weiteren Protokoll zusammen. Ethernet-Pakete, die z. B. das UDP, ICMP oder IGMP Protokoll beinhalten, benötigen alle auch das IP-Protokoll und den MAC-Frame. Die Erzeugung eines Datenpaketes findet daher geeigneterweise durch die Kaskadierung einzelner Module

statt. Jedes Modul für sich genommen, ist für die Generierung eines einzigen Protokolls zuständig. Dies hat den Vorteil, dass verschiedene Kaskadierungen vorgenommen werden können, ohne dass eine unnötige Duplizierung der Logik und damit eine Erhöhung des Ressourcenverbrauches entsteht. Dies kann als eine Art des Ressource-Sharing gesehen werden [33, S. 199]. Bild 5.10 zeigt die hierzu notwendige Anordnung dieser Module. Jedes Modul besitzt einen “Valid In”-Eingang über den ein anderes Modul, über dessen “Valid-Out”-Ausgang die Generierung des Protokolls (Header- und Datenbereich) anfordern kann. Die Daten werden über den “Data Out”-Ausgang auf den “Data In”-Eingang des anfragenden Moduls gegeben. Die korrekte Kaskadierung der Module findet über die in Bild 5.10 gezeigte Steuereinheit statt. Im Bild ist dies durch (De-)Multiplexer schematisch dargestellt.

Dem Bild 5.11 kann beispielhaft der Aufbau des Moduls zur Generierung des UDP-Protokolls entnommen werden. Alle Module sind in ähnlicher Weise aufgebaut:

- Idle-Zustand:
  Nach einem Power-Up oder nach durchgeführter Generierung des Protokolls befindet sich die FSM im Idle-Zustand. Sobald der “VALID_IN”-Eingang gesetzt wird, wird der Protokoll-Zustand gewechselt.
- Protocol-Zustand:
  Im Protocol-Zustand findet die Generierung des Protokolls mittels Multiplexer statt, der von einem Counter angesteuert wird, sodass nacheinander verschiedene Protokoll-Informationen auf den Ausgang gelegt werden. Einen Takt, bevor in den “Data”-Zustand gewechselt wird, wird das “VALID_OUT”-Signal gesetzt, welches in diesem Fall die Applikations-Ebene auffordert, Daten zum Versenden bereitzustellen.
- Data-Zustand:
  Im “Data”-Zustand werden die Eingangsdaten, in diesem Fall die Nutzdaten von der Applikationsebene, auf den Ausgang des Moduls gelegt.

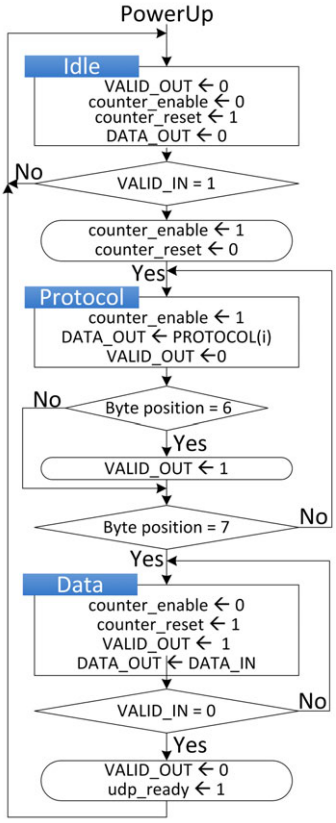


Bild 5.11.: ASM-Diagramm für FSM zur Generierung von UDP-Protokollen

Alle generierten Daten werden über FlipFlops auf den Ausgang geführt, sodass jedes Modul eine Latenz von einem Takt aufweist. Dies verkürzt die kombinatorischen Pfade und erhöht die maximal mögliche Taktfrequenz. In Sendepfad weist der Ethernet-Stack beim Versenden von UDP/IP-Nachrichten eine Latenz von drei Takten auf.
Die für den Protokoll-Header benötigten Informationen werden je nach Art entweder von der Stack-Steuerung zur Verfügung gestellt (z. B. Angaben zur Länge, IP- und MAC-Adressen, Portnummern etc.) oder die Daten werden intern im Modul generiert, wie etwa Checksummen.

Empfangs-Datenpfad

Im Bild 5.12 ist der Empfangs-Datenpfad des Ethernet-Stacks zu sehen. Das Prinzip ist hier ähnlich der Sende-Einheit, nur dass keine Steuerung von außen existiert. Die Module sind da-

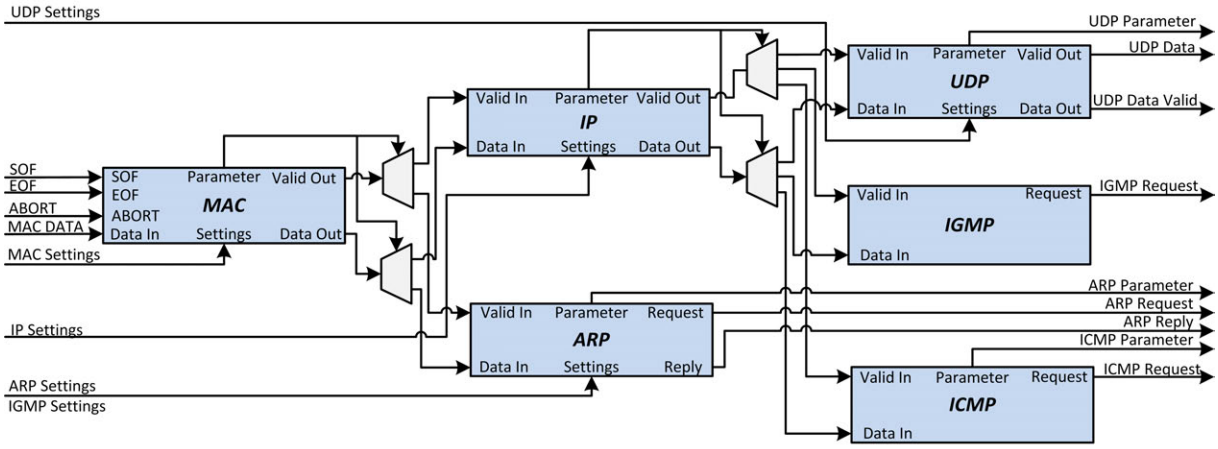


Bild 5.12.: Empfangs-Datenpfad für Protokollinterpretation im Ethernet-Stack

her so aufgebaut, dass sie dynamisch ermitteln, in welcher Reihenfolge diese zur Interpretation der Nachricht kaskadiert werden müssen. Dies ist durch Informationen im MAC-Frame und im IP-Header möglich, die Auskunft über das nachfolgende Protokoll geben. Im Falle des MAC-Frames liegt diese Information in den letzten beiden Bytes des Headers, wodurch innerhalb eines Taktes die Kaskadierung des nachfolgenden Protokoll-Moduls erfolgen muss. Die Kaskadierung erfolgt daher nicht getaktet, sondern kombinatorisch.
Das Modul, welches sich als Letztes in der Kaskade befindet, gibt entweder eine Meldung über den Empfang einer Nachricht an die Steuereinheit des Ethernet-Stacks und stellt ggf. für die Weiterverarbeitung spezifische Zusatzinformationen bereit oder gibt, wenn es sich um das UDP-Empfangs-Modul handelt, die Nutzdaten direkt an die Applikationsebene weiter.
Jedes Modul wird durch das Konfigurations-Interface in der Applikationsebene mit grundlegenden Informationen zu den vier Netzwerk-Interfaces versorgt. Hierzu zählen die eigene MAC- und IP-Adresse, allen drei Multicast-Gruppen-Adressen und die UDP-Port-Nummern. Mit diesen Informationen können die einzelnen Module die Filterung von Nachrichten durchführen. Um die Übersichtlichkeit zu erhöhen, ist ein “Package” erstellt, indem u. a. VHDL-Records definiert sind. Dadurch besteht die Möglichkeit, alle Informationen zu einem einzigen Signal zusammenzufassen. Da diese Informationen dauerhaft anliegen und nur bei Bedarf durch das Konfigurations-Interface angepasst werden müssen, gibt es keine kritischen Anforderungen an das Zeitverhalten zur Verteilung dieser Informationen.
Dem Bild 5.13 ist beispielhaft die Kaskade zur Interpretation von empfangenen UDP/IP-Nachrichten zu entnehmen. Die Module sind dabei alle nach einem einheitlichen Schema und ähnlich wie die Module im Empfangspfad aufgebaut:

- “Idle”-Zustand:
  Jedes Modul befindet sich nach einem Power-UP und nach der jeweiligen durchgeführten Interpretation eines Ethernet-Paketes im “Idle”-Zustand. Sobald das Valid-In durch das dem Modul vorgelagerte Modul gesetzt wird, wird in einen Zustand zur Interpretation des Protokoll-Headers gewechselt.

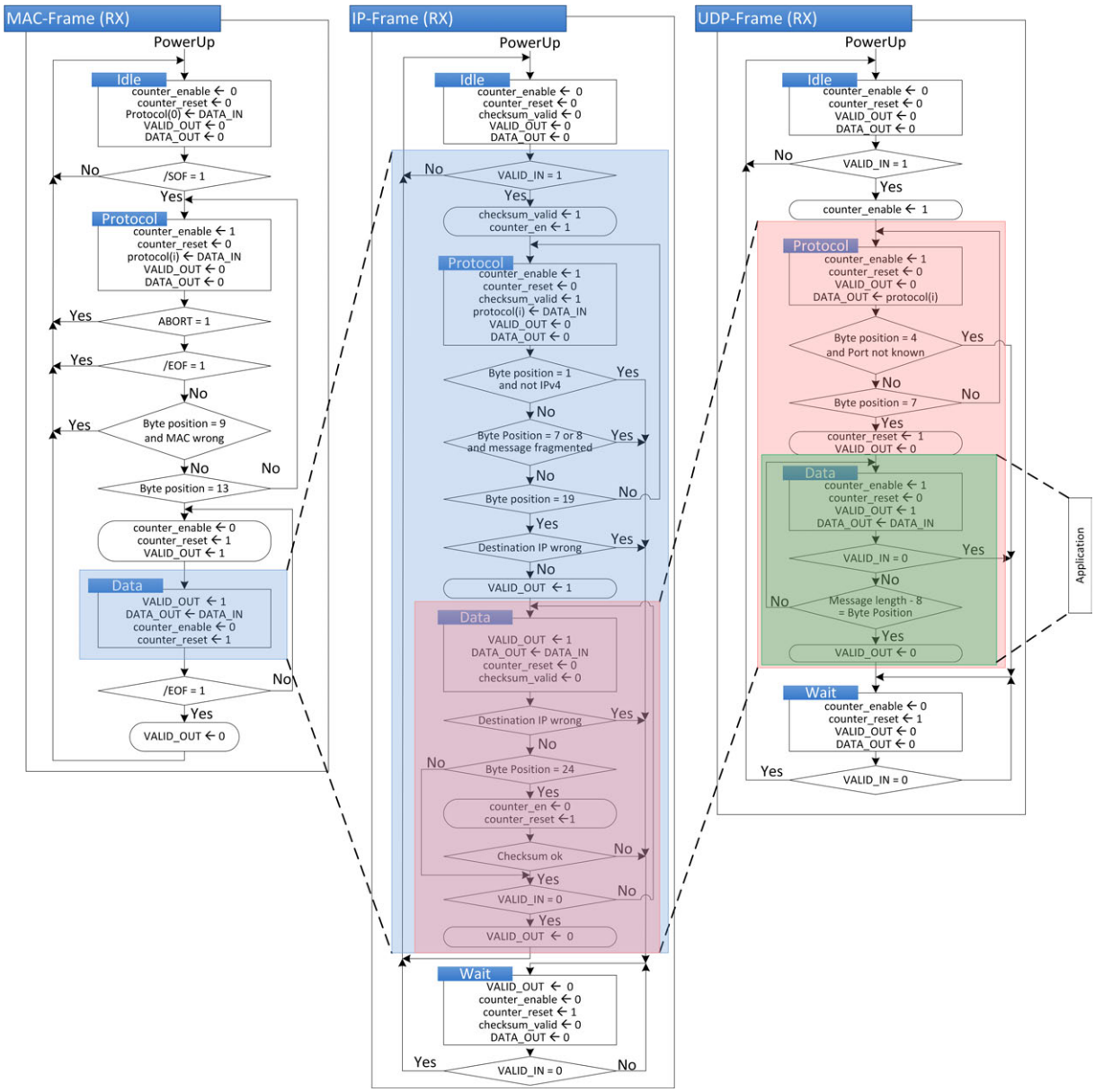


Bild 5.13.: Darstellung der Modul-Kaskadierung zur Interpretation von UDP-IP-Nachrichten im Empfangspfad des Ethernet-Stacks

- “Protocol”-Zustand:
  Im “Protocol”-Zustand werden mithilfe eines Demultiplexers relevante Informationen in Abhängigkeit ihrer Byteposition im Protokoll in FlipFlops abgelegt, die eine Speicherung der Informationen bis zur Verarbeitung der nächsten Nachricht sicherstellen. So gespeicherte Informationen können z. B. der Steuereinheit zur Verfügung gestellt werden.
  Anhand der Headerinformationen werden zudem protokollspezifische Überprüfungen vorgenommen, die zu einer Filterung der Nachricht führen können. Filterungen finden statt, wenn Fehler (z. B. Checksummenfehler) erkannt werden oder wenn die Nachricht an eine, im System nicht konfigurierte MAC- oder IP-Adresse oder UDP-Port gerichtet ist. Nur wenn keine Filterung stattfand, wird in den “Data”-Zustand gewechselt.

- “Data”-Zustand:
  In diesem Zustand wird, analog zu den Modulen im Sendepfad, der Eingang auf den

Ausgang geschaltet. Der Zustand wird erst verlassen, wenn die Datengültigkeit von der vorherigen Stufe zurückgenommen wird.

- "Wait"-Zustand:
  Alle Module, die dem MAC-Modul nachgelagert sind, besitzen einen "Wait"-Zustand, in den gewechselt wird, sobald ein Fehler auftrat oder eine Filterung stattfindet. Der "Wait"-Zustand ist notwendig, damit die FSM vom Idle-Zustand nicht direkt wieder in den "Header"-Zustand wechseln kann, wenn das vorgelagerte Empfangsmodul das Valid-Signal noch nicht zurückgenommen hat.
  Im "Wait"-Zustand wird das Valid-Signal zur Angabe der Datengültigkeit für das nachfolgende Modul zurückgenommen. Das nachfolgende Modul erkennt dies, entweder indem ein Fehler bei der eigenen Protokoll-Interpretation festgestellt wird, da keine Daten mehr geliefert werden oder indem das Valid-Signal abgefragt wird. In beiden Fällen wechselt der Zustandsautomat direkt oder über den "Wait"-Zustand zurück in den "Idle"-Zustand. Es kann damit sichergestellt werden, dass es zu keinen undefinierten Zuständen oder Deadlocks kommen kann, wenn der Ethernet-MAC am Ende der Nachricht ein End-Of-Frame Signal setzt und damit selbst Deadlockfrei ist.

Stack-Steuerung

Den Datenpfaden übergeordnet befindet sich eine Steuer-Einheit, die in erster Linie Zugriffe auf die Sende-Einheit steuert. Die Aufgabe der Steuereinheit besteht darin, Sende-Anforderungen entgegen zu nehmen und diese prioritätenbasiert abzuarbeiten. Neben der Applikationsebene, die Sende-Anfragen für UDP-Nachrichten, ARP-Requests und IGMP-Membership-Querys stellen kann, kann auch der Empfangspfad Sendeanfragen generieren:

- Wird ein ARP-Request empfangen, wird automatisch ein Sende-Request für ein ARP-Reply gestellt.
- Wird ein IGMP-Membership-Query empfangen, wird eine Sende-Request zum Verschicken von IGMP-Membership-Reports für jede Multicast-Gruppe eingeleitet.
- Auf den Empfang eines ICMP-Echo-Requests wird mit einem Sende-Request für ein ICMP-Echo-Replys reagiert.

Der Tabelle 5.1 können die gewählten Prioritäten einzelner Sende-Request entnommen werden, die die Abarbeitungsreihenfolge vorgeben, wenn diese zeitgleich anliegen. UDP-Nachrichten erhalten dabei die höchste Priorität, um das Senden von Information mit der kleinstmöglichen Latenz durchführen zu können. Die Antwort auf ARP-Request erhält die zweithöchste Priorität, da andere Teilnehmer diese Antwort benötigen, um der Elektronik Unicast-Nachrichten zuzuschicken. Die übrige Reihenfolge der Sendepriorität ist hingegen unkritischer und daher beliebig zu wählen. Sendeanfragen, die aufgrund des Empfangs eines IGMP-Membership-Querys, ARP-Request oder ICMP-Request generiert werden, werden von der Stack-Steuerung gespeichert und automatisch nach der Abarbeitung gelöscht. Gespeichert wird jeweils die letzte Sende-Anfrage. Dies ist ausreichend, da die Wiederholfrequenz dieser Anfragen vergleichsweise niedrig ist.
Bild 5.14 zeigt am Beispiel des Versendens von UDP/IP-Nachrichten ein UML-Sequenzdiagramm für den Ablauf eines Sendevorganges mit den notwendigen Interaktionen zwischen den einzelnen Komponenten: Die Applikation muss eine Sende-Anfrage an

Priorität	Anforderung
1	Anforderung zum Senden einer UDP-Nachricht
2	Anforderung zum Antworten auf eine ARP-Request eines anderen Teilnehmers
3	Anforderung zum Senden eines ARP-Request an einen anderen Teilnehmer
4	Anforderung zum Senden eines IGMP-Membership Reports
5	Anforderung zum Senden eines ICMP-Reply

Tabelle 5.1.: Abarbeitungs-Prioritäten von Sende-Anforderungen

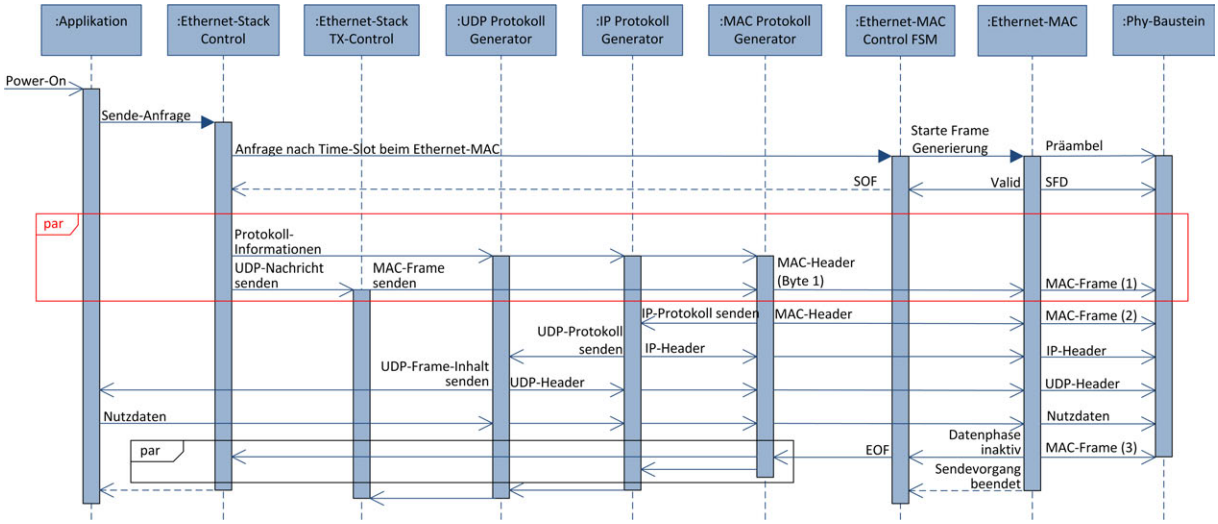


Bild 5.14.: UML-Sequenzdiagramm für das Senden von UDP/IP-Nachrichten

den Ethernet-Stack übergeben. Wenn der Sendepfad des Ethernet-Stacks nicht mehr mit der Durchführung eines anderen Sendevorgangs beschäftigt ist, wird die Stack-Steuerung an den Ethernet-MAC eine Sende-Anfrage stellen. Sobald dieser für einen neuen Sende-Vorgang bereit ist, beginnt der Ethernet-MAC mit dem Verschicken der Präambel und des Start-Frame-Delimiters. Wenn die Daten vom Ethernet-Stack benötigt werden, wird eine Rückmeldung an die anfragende Steuereinheit gegeben. Diese gibt daraufhin an die zweite Steuereinheit die Anweisung eine UDP/IP-Nachricht zu generieren. Die zweite Steuereinheit kaskadiert daraufhin die einzelnen, notwendigen Protokoll-Generator-Module und teilt dem Generator zur Erzeugung des MAC-Frames mit, diesen auszugeben. Der Ethernet-Stack beginnt damit automatisch die Abfolge des MAC-Frames, IP-Headers und UDP-Headers zu generieren, bevor der Applikationsebene mitgeteilt wird, die Nutzdaten dem Ethernet-Stack zu übergeben. Nach der Übertragung der Nutzdaten beendet der Ethernet-Stack die Abarbeitung und ist prinzipiell in der Lage, eine neue Nachricht zu generieren. Der Ethernet-MAC beendet hingegen seine Abarbeitung verzögert, da dieser noch die CRC-Summe übertragen muss und erst nach der Inter-Frame-Gap (IFG, siehe Abschnitt 2.2.1) wieder Daten übertragen darf.

5.5.2. Protokoll-Verarbeitung

In den folgenden Abschnitten wird auf die Besonderheiten der Protokoll-Implementierungen der Protokolle ARP, ICMP und IGMP eingegangen. Auf die Protokolle UDP, IP und den MAC-

Frame wird hier nicht weiter eingegangen, diese sind als Beispiele bereits in Abschnitt 5.5.1 thematisiert worden.

Adress Resolution Protocol (ARP)

Alle per Unicast gesendeten Nachrichten in einem Netzwerk sind an einen bestimmten Teilnehmer gerichtet. Um das Routing von Ethernet-Paketen innerhalb eines Subnetzes durchführen zu können, müssen die IP-Adressen den physikalischen MAC-Adressen zugeordnet werden. Hierfür ist das "Adress Resolution Protocol" (ARP) notwendig. Jeder Netzwerkteilnehmer kann mit einem per Broadcast verschickten ARP-Request die MAC-Adresse eines anderen Netzwerk-Teilnehmers erfragen. Ein ARP-Request ist mit einem ARP-Reply von diesem Netzwerk-Teilnehmer zu beantworten. Sollen Unicast-Verbindungen zu mehreren Teilnehmern unterhalten werden, so muss jeder Sender für sich eine ARP-Tabelle anlegen. In dieser Tabelle werden Zuordnungen zwischen IP- und MAC-Adressen abgelegt. Mithilfe des ARP-Protokolls muss die Applikations-Ebene damit keine Kenntnis über die physikalische Adresse des Empfängers haben.
Folgende Funktionen sind im Ethernet-Stack implementiert:

- Anfrage der MAC-Adresse eines anderen Netzwerk-Teilnehmers:
  Die hier durchgeführte Implementierung des Ethernet-Stacks sieht vor, eine Unicast-Verbindung zu einem Teilnehmer zu unterhalten, der die Steuerung der Elektronik übernimmt. Damit entfällt die Notwendigkeit mehrere Ziel-MAC-Adressen, und damit eine ARP-Tabelle, zu verwalten. Um die Flexibilität des Ethernet-Stacks zu erhöhen, wird das Design so ausgelegt, dass eine dynamische Steuerung für das Senden von ARP-Request möglich ist. Über das Konfigurations-Interface lässt sich durch den verwendeten MicroBlaze ARP-Requests für das Erfragen der MAC-Adresse zu beliebige IP-Adressen auslösen.
  Wird eine ARP-Reply-Nachricht empfangen, die die eigene IP-Adresse beinhaltet, so speichert die Stack-Steuerung die enthaltene MAC-Adresse in Registern des Konfigurations-Interfaces der Applikations-Ebene ab und löst dabei einen Interrupt an den MicroBlaze aus, um diesen zu informieren. In Software kann dadurch eine ARP-Tabelle mit beliebig vielen Teilnehmern erstellt werden. Soll eine Nachricht an einen dieser Teilnehmer gesendet werden, muss lediglich IP- und MAC-Adresse in die zuständigen Register des Konfigurations-Interface eingetragen werden. Damit ist das Design prinzipiell in der Lage, beliebig viele Unicast-Verbindungen zu unterhalten.
- Bereitstellen der eigenen MAC-Adresse für andere Netzwerk-Teilnehmer:
  Neben der Möglichkeit, ARP-Requests zu senden und die folgenden ARP-Replys auszuwerten, kann auch der Empfang von ARP-Requests anderer Netzwerk-Teilnehmer verarbeitet und automatisch ein ARP-Reply versendet werden. Nach Abschnitt 5.5.1 kann sich die Antwort verzögern, wenn der Sende-Pfad nicht frei ist oder eine höher priorisierte Sende-Anfrage vorliegt.

Internet Group Management Protokoll (IGMP)

In dem hier entwickelten Ethernet-Stack wird das, zur An- und Abmeldung von Multicast-Gruppen-Mitgliedschaften notwendige, IGMP-Protokoll in der Version 2 einschließlich der Ver-

waltung von bis zu drei Mitgliedschaften implementiert.
Die Multicast-Gruppen selbst werden durch die Ethernet-Switches eines Netzwerkes verwaltet, da diese das Routing der an eine Multicast-Gruppe gesendeten UDP/IP-Nachrichten durchführen müssen. Für die Verwaltung der Mitgliedschaften sind folgende Funktionen implementiert:

- Um die Listen über Gruppen-Teilnehmer aktuell zu halten, wird ein IGMP-Snooping durchgeführt, bei dem die Switches periodisch durch Membership-Querys bei allen Teilnehmern im Netzwerk anfragen, an welchen Mitgliedschaften diese interessiert sind. Die Teilnehmer haben hierauf innerhalb eines im IGMP-Protokoll vorgegebenen Zeitfensters zu antworten. Der Ethernet-Stack tut dies, indem dieser automatisch Membership-Reports für alle aktivierten Gruppen sendet, sobald ein Query eintrifft. Die Multicast-Gruppen-Adressen, sowie ihr Status, werden durch Register des Konfigurations-Interface definiert und sind dadurch vom MicroBlaze zur Laufzeit veränderbar.
- Unabhängig vom IGMP-Snooping-Intervall können asynchron An- und Abmeldungen durchgeführt werden, indem Membership-Reports mit den spezifischen Angaben versendet werden. Auch dies ist über das Konfigurations-Interface steuerbar.

Anders als bei den anderen Protokollen erfordert die Verarbeitung des IGMP-Protokolls in Sende- wie auch in Empfangsrichtung eine zusätzliche Flexibilität der MAC- und IP-Module. Membership-Querys werden von den Switches an die Multicast-Gruppe 224.0.0.1 und der daraus ableitbaren MAC-Adresse (01:00:5E:00:00:01) gesendet (siehe hierzu Anhang A.1.5). Multicast-Nachrichten werden nicht an die IP-Adresse der Elektronik, sondern an die Multicast-Gruppenadresse. Die Anmeldung und Bestätigung von Mitgliedschaften ist durch den Ethernet-Stack an die Adresse der betreffenden Gruppe zu schicken. Abmeldungen hingegen müssen an die Multicast-Gruppe 224.0.0.2 geschickt werden (MAC-Adresse 01:00:5E:00:00:02). Dadurch ergibt sich die Notwendigkeit in Sende- und Empfangsrichtung Nachrichten zu verarbeiten, die an verschiedene IP- und MAC-Adressen gerichtet sind. Dies ist etwa bei der Filterung von Nachrichten zu berücksichtigen.
Zudem muss beim Senden von IGMP-Nachrichten die Time-To-Live[5]-Information (TTL) im IP-Header auf eins reduziert und das im IP-Header optional spezifizierte Optionsfeld genutzt werden. Im Optionsfeld wird dem Switch mitgeteilt, dass die Auswertung der Nachricht durch das Switch selbst zu erfolgen hat und keine Weiterleitung der Nachricht durchzuführen ist. Dieses würde, aufgrund der angegebenen TTL, zur Zerstörung der Nachricht führen.
Bild 5.15 zeigt die Abläufe der Stack-Steuerung um IGMP-Membership-Reports zur An-/Abmeldungen bzw. Bestätigung von bis zu drei Multicast-Gruppen-Mitgliedschaften verschicken zu können. Initiiert wird der Vorgang entweder durch den MicroBlaze bzw. das Konfigurations-Interface oder aber durch den Empfang eines Membership-Querys. Der im Bild 5.15 angegebene Dummy-Zustand ist stellvertretend für alle anderen Zustände in der Stack-Steuerung, die nicht für die Multicast-Verarbeitung relevant sind. Dieser Zustand wird nur verlassen, wenn keine höher priorisierte Sende-Anfrage zu verarbeiten ist. Stufenweise erfolgt dann die Abfrage der Statusregister der drei Multicast-Gruppen im Konfigurations-Interface. Es kommt zum Senden eines Membership-Reports, wenn die Gruppe aktiv ist und diese nach

[5]Die Time-To-Live (TTL) wird mit jedem “Hop” (engl. Hopser) dekrementiert. Als ein “Hop” wird der Weg von einem Netzwerkknoten zum nächsten bezeichnet. Die TTL wird damit bei jeder Weiterleitung durch ein Ethernet-Switch/-Router dekrementiert. Sobald die TTL null ergibt, erfolgt keine Weiterleitung. Auf diese Weise ist sichergestellt, das eine Nachricht nicht beliebig lange ein Netzwerk belastet, wenn diese nicht zugestellt werden kann.

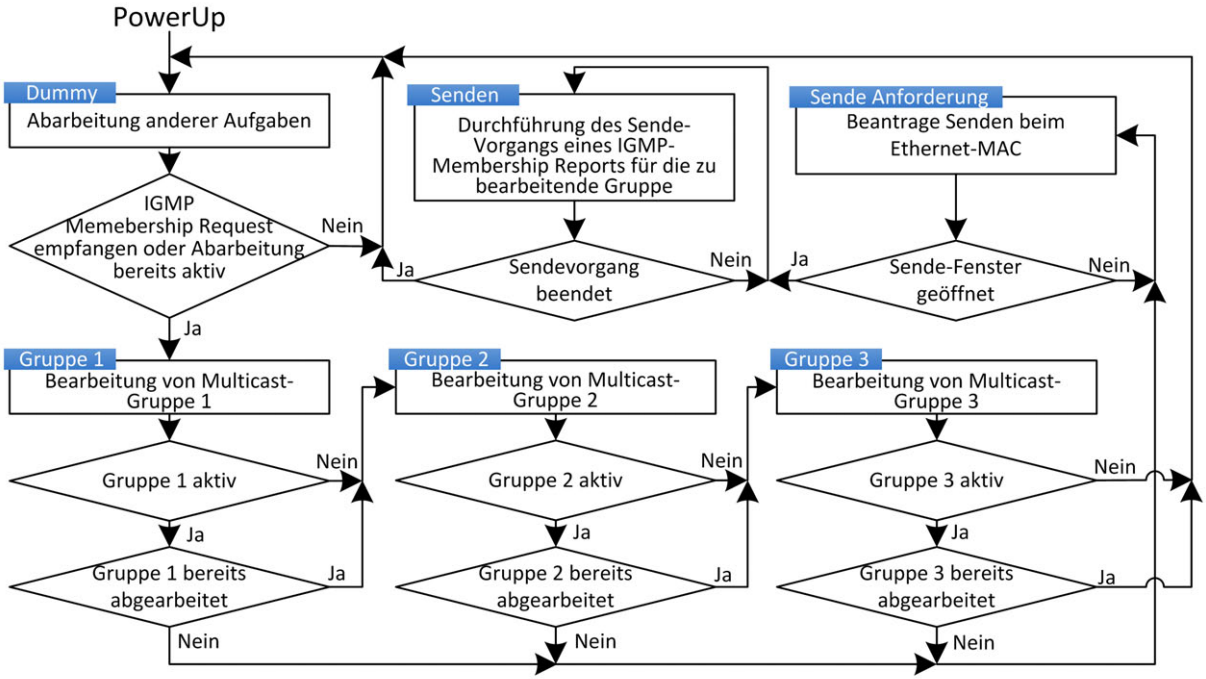


Bild 5.15.: Vereinfachte Darstellung der Stack-Steuerung zum Senden von IGMP-Membership-Reports zur An- und Abmeldung bzw. Bestätigung an/von Multicast-Gruppen

dem aktuellen Sende-Request noch nicht bearbeitet wurde. Pro Durchlauf wird maximal ein Sendevorgang ausgeführt und anschließend zurück in den Dummy-Zustand gewechselt. Dadurch besteht die Möglichkeit, dass in der Zwischenzeit eingetroffene Sende-Anfragen höherer Priorität in der Abarbeitung dazwischen geschoben werden können. Die Abarbeitung eines Sende-Request kann sich damit aus bis zu drei Durchläufen ergeben. Durch zusätzliche Angaben im Konfigurations-Interface können neben Anmeldungen/Bestätigungen von Gruppenmitgliedschaften auch Abmeldungen über diesen Zustands-Automaten durchgeführt werden.

Internet Control Message Protocol (ICMP)

Das ICMP-Protokoll wird dazu genutzt, Informations- und Fehlermeldungen zwischen Netzwerk-Teilnehmern auszutauschen. Eine der mit diesem Protokoll realisierbaren Funktionen ist die Echo-Funktion, allgemein auch als Ping-Funktion bekannt. Wesentliche Datenfelder des Protokolls sind ein Typ-Feld, welches angibt, dass es sich um ein Echo-Request/-Reply handelt, ein Checksummen-Feld und ein Daten-Feld für optionale Daten. Bei Echo-Request/-Reply Nachrichten setzen sich diese optionalen Daten aus Identifier, Sequenz-Nummer und 32 Byte zufälliger Daten zusammen. Ein Echo-Reply muss die gleichen optionalen Daten beinhalten, wie der zugehörige Echo-Request. Nur so kann der anfragende Netzwerkteilnehmer dem Echo-Reply den Echo-Request zuordnen.
Das Vorgehen zur Verarbeitung des Echo-Request ist ähnlich zur Verarbeitung der ARP-Request implementiert. Ein Unterschied besteht in der Übergabe der Protokoll-Informationen des Echo-Requests an das Sende-Modul zur Generierung des Echo-Replys. Würde dies durch Schiebe-Register geschehen, würde die ICMP-Funktion rund 36 % der Gesamtressourcen des Ethernet-Stacks in Anspruch nehmen, da die zu übergebene Datenmenge vergleichsweise hoch ist. Daher wird hier ein RAM-Speicher als Zwischenspeicher eingesetzt. Der Verbrauch an

Logik-Ressourcen senkt sich damit erheblich.
Die Berechnung der Checksumme im Sende-Modul ist problematisch, da das Ergebnis der Checksummenberechnung über alle 36 Byte des Protokolls bereits ab Byteposition drei anliegen muss. Zur Generierung des MAC- und IP-Frame werden jedoch nur 34 Takte benötigt, sodass das Ergebnis 36 Takte nach einer Sendeanfrage vorliegen muss. Das Starten der Checksummenberechnung vor dem eigentlichen Sendebeginn oder die Durchführung der Checksummenberechnung bei höherer Taktung wird, auch in Hinblick auf den Ressourcenverbrauch, als suboptimal angesehen. Bei näherer Betrachtung der Checksummenberechnung ergibt sich eine Möglichkeit, die Checksumme über alle 36 Byte innerhalb eines Taktes bei zugleich geringem Ressourcenverbrauch durchführen zu können:
Die Checksummenberechnung erfolgt, indem der Header in 16-Bit große Blöcke unterteilt wird und das 16-Bit Einerkomplement der Einerkomplement-Summe aller Blöcke gebildet wird. Durch die Zusammenfassung von jeweils 16-Bit bedeutet dies bei der Berechnung der Checksumme eines Echo-Requests (nicht Echo-Reply!) die Addition des ersten 16-Bit-Block und damit des Typ-Feldes, mit dem Checksummenfeld, welches für die Berechnung der Checksumme mit Nullen gefüllt wird. Es kann damit bei dieser Addition zu keinem Carry-Out kommen. Damit ist es zulässig, die Checksumme des Echo-Replys aus der Checksumme des Echo-Request zu berechnen, indem der erste 16-Bit-Block des Echo-Request vom ersten 16-Bit-Block des Echo-Reply subtrahiert wird, sodass in diesem Fall also eine Addition um 0x0800 stattfindet[6]. Dies geschieht vor dem Hintergrund, dass sich ein Echo-Reply von einem Echo-Request nur im ersten 16-Bit-Block und damit auch in der Checksumme unterscheidet. Alle anderen Angaben müssen identisch sein. Der Ressourcenverbrauch für die Checksummenberechnung ist daher sehr gering. Es ist lediglich eine Addition von zwei 16-Bit-Werten und die Übergabe der empfangenen Checksumme an das Sende-Modul notwendig.

5.5.3. Designergebnis

In Tabelle 5.2 sind die Ressourcenverbräuche der einzelnen Protokoll-Module des Ethernet-Stacks zu entnehmen. Die meisten Ressourcen werden durch das IP-Sende-Modul benötigt. Die Anzahl an LUTs ist hoch, da die Checksumme aus Vergleichsweise vielen Operanden berechnet werden muss. Im Gegensatz zur Checksummen-Überprüfung des IP-Empfangs-Moduls wird hier kein Ressource-Sharing angewendet. Dies liegt darin begründet, dass die Checksumme an Byteposition 11 und 12 einzufügen ist, diese jedoch über 20 Byte, bzw. 24 Byte bei Nutzung des Optionfeldes, zu bilden ist. Je nach Intensität des Ressource-Sharing kann dies dazu führen, dass das Ergebnis zu spät anliegt. Die hohe Anzahl an Registern ist im Zuge von Optimierungen entstanden, um die für Gigabit-Ethernet notwendige maximale Taktfrequenz zu erhalten, indem die Additionen bei der Checksummenberechnung nicht kombinatorisch, sondern getaktet durchgeführt wird. Nach [74] wird empfohlen keine Addierer-Baumstruktur, sondern eine Addierer-Kettenstruktur zu verwenden. Diese benötigt zwar mehr FlipFlops, soll jedoch performanter sein und vorhersagbarere Routing-Ergebnisse liefern.
Ebenfalls ressourcenintensiv ist die Verarbeitung des ARP-Protokoll in Empfangsrichtung. Dies liegt darin begründet, dass eine hohe Anzahl an Registern notwendig ist, um der Stack-Steuerung und damit dem Sendepfad die IP- und MAC-Adresse des anfragenden Netzwerk-Teilnehmers zu übergeben, die im ARP-Request enthalten sind. Diese Informationen sind zur

[6] Eine Echo-Request beginnt im ICMP-Protokoll mit 0x0800, ein Echo-Reply mit 0x0000.

Generierung des ARP-Reply notwendig.

Ressource	Empfangsrichtung						Senderichtung						
	MAC	UDP	IP	ARP	IGMP	ICMP	MAC	UDP	IP	ARP	IGMP	ICMP	Control
LUT	37	37	127	62	4	35	80	38	217	9	89	57	374
Register	46	73	162	207	10	55	124	26	247	17	85	32	423

Tabelle 5.2.: Ressourcen-Verbrauch des Ethernet-Stacks nach Modulen

Ziel der Implementierung des Ethernet-Interfaces war die Unterstützung eines Taktes von 125 MHz in den Datenpfaden. Ethernet-MAC und -Stack müssen hierfür immer in Verbindung betrachtet werden. Die maximale Taktfrequenz des Gesamtdesigns ist dabei wenig aussagekräftig, da mitunter Komponenten im Design enthalten sind, die mit geringerer Frequenz getaktet werden und ggf. auch nur können. Zur Bewertung des Designs ist daher der Slack entscheidender. Der minimale Slack kann in der statischen Timinganalyse für jede Timing-Constraint ermittelt werden, sodass eine separierte Aussage über die maximale Taktfrequenz jener Logik gemacht werden kann, die mit ihrer Geschwindigkeit die Höhe des Datendurchsatzes bestimmt. Bei der Verwendung des “Tri-Mode Ethernet-MAC” unter Nutzung des in Abschnitt 5.4.3 entwickelten Wrappers wird bei den relevanten Pfaden bei einem Spartan-6 FPGA eine maximale Taktfrequenz von 113.8 MHz nach der vollständigen Implementierung nachgewiesen. Für die Anbindung an ein Gigabit-Ethernet ist diese zu gering. Die Timing-Analyse zeigt, dass der kritische Pfad im Ethernet-Stack zwischen dem Setzen des “Dataphase”-Signals durch den Ethernet-MAC und der Übergabe des ersten Datenbytes vom Ethernet-Stack an den Ethernet-MAC liegt. Dieser Pfad ist für das erste Datenbyte kombinatorisch, um die Daten schnell genug zur Verfügung stellen zu können. Das UML-Sequenzdiagram in Bild 5.14 zeigt dies beim oberen “Combined Fragment” (rot gekennzeichnet). Würde das Setzen des “Datenphase”-Signals einen Takt früher geschehen, kann durch Register der kombinatorische Pfad verkürzt und die Taktfrequenz so auf 126.07 MHz angehoben werden. Der Ethernet-Stack ist dann für die Anbindung an ein Gigabit-Ethernet ausgelegt. Für den “Tri-Mode Ethernet-MAC” ist dies einfach realisierbar, da dieser mit FIFOs arbeitet. Die Schnittstellenkompatibilität zum “EthMAC” geht damit jedoch verloren, sodass diese Anpassung nicht implementiert wird[7].

5.6. Video-Verarbeitung

Für die Verarbeitung des Radar-Video-Sweep ist eine Kette von mehreren Verarbeitungsstufen notwendig. Diese Kette beginnt direkt nach dem Ethernet-Stack, welcher empfangene und bereits gefilterte UDP-IP-Nachrichten an die Applikations-Ebene übergibt. Dabei übergibt der Ethernet-Stack nicht nur Nachrichten einer Unicast-Verbindung, sondern auch Nachrichten die

[7] Auf einem Virtex-5 FPGA kann eine Taktfrequenz größer 125 MHz auch ohne diese Änderung erreicht werden. Grund sind Technologie-bedingte Unterschiede zwischen Spartan- und Virtex-FPGAs.

an eine von bis zu drei Multicast-Gruppen geschickt worden sind. Es findet daher gemäß der Konzeptionierung in Abschnitt 4.2.3 ein Demultiplexen auf Applikations-Ebene statt:

- Wird die empfangene Ethernet-Nachricht als eine Nachricht erkannt, die an die erste, konfigurierbare Multicast-Gruppe gerichtet wurde, so wird diese Nachricht an den Video-Interpreter übergeben. Durch die Filterung im Ethernet-Stack handelt es sich ausschließlich um Ethernet-Pakete, die von dem zu verarbeitenden Radar-Transceiver stammen und Video-Informationen enthalten.
- Handelt es sich um eine Unicast-Nachrichten, welche an einen definierten Port der Elektronik gesendet wurde, wird hingegen die Nachricht über Buffer dem MicroBlaze zur Verfügung gestellt.

In Abschnitten 5.6.1 wird als Erstes die Speicherung der Daten thematisiert, bevor anschließend in Abschnitt 5.6.2 auf den Interpreter zur Interpretation der Video-Nachrichten eingegangen wird. Abschließend in Abschnitt 5.6.3 wird die DAC-Ansteuerung erläutert.

5.6.1. Video-Buffer

Für eine zeitliche Entkoppelung der Empfangs- und Ausgabeperiode ist eine Zwischenspeicherung der Radar-Sweeps notwendig. Um diese zu realisieren, werden zwei voneinander getrennte Ring-Speicher implementiert, die es erlauben, mehrere Sweeps zeitgleich zu speichern. Der Video-Interpreter gilt hier als der Erzeuger von Daten für beide Speicher. In einem der Speicher werden die Informationen über den Sweep abgelegt, sodass der MicroBlaze über Informationen, wie etwa der Richtungs- und Zeitangabe des Sweeps, informiert werden kann. In dem zweiten Speicher werden die Sample-Werte für die DAC-Ansteuerung gespeichert.
Um mehrere Sweeps zeitgleich abzuspeichern, wird der Sample-Speicher in einem Ring aus

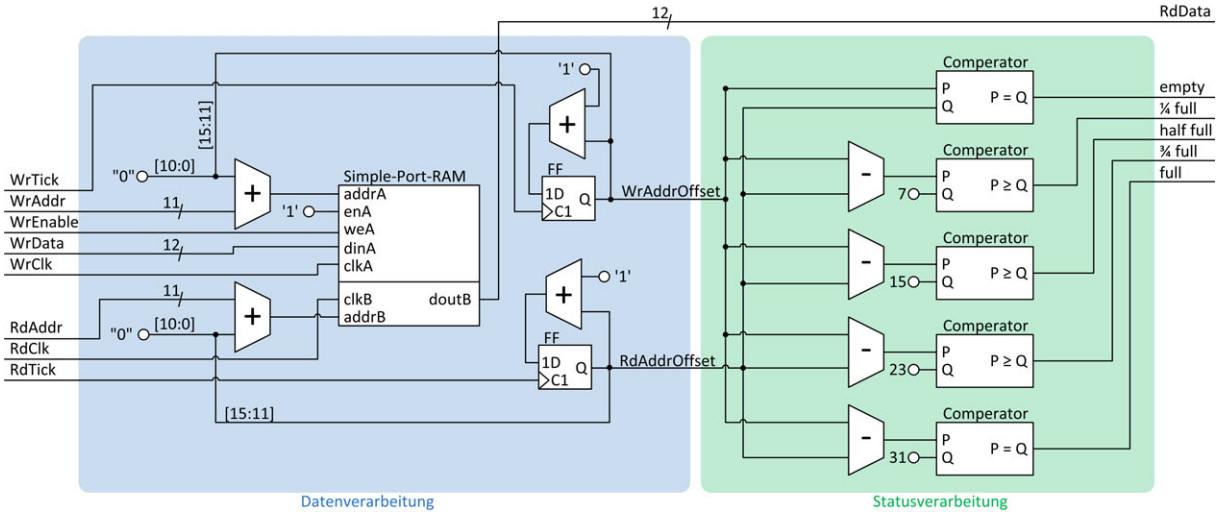


Bild 5.16.: Ring-Buffer aus Speicherblöcken zur Speicherung 32 Video-Sweeps

Speicherblöcken organisiert, indem die Beschreibung der Logik gemäß Bild 5.16 erfolgt. Für den Erzeuger und den Verbraucher verhält sich der Ring-Speicher daher ähnlich einem normalen Speicher, auf den durch Angabe der Speicheradresse zugegriffen werden kann. Zusätzlich existieren jedoch zwei weitere Takteingänge, mit denen bei aktiver Flanke der nächste Speicherblock getrennt für das Lesen und Schreiben angefordert werden kann. Nach dem letzten

Speicherblock erfolgt automatisch das Zurückspringen auf den ersten Speicherblock. Durch die Verwendung eines Simple-Dual-Port-RAM kann auf den Speicher zeitgleich und unter Verwendung verschiedener Taktfrequenzen zugegriffen werden. Der Füllstand des Ring-Buffers kann über das Konfigurations-Interface vom MicroBlaze jederzeit ausgelesen werden. Unterteilt wird die Angabe des Füllstandes in Viertel-Schritte. Der Status wird dabei aus der Differenz zwischen dem aktuell selektierten Schreib- und Lese-Block-Identifier ermittelt.
Der Ring-Speicher für die Sweep-Informationen kann einfacher gehalten werden. Anstelle der Speicherblöcke steht für jeden speicherbaren Sweep ein Vektor zu Verfügung, indem die verschiedenen Informationen an unterschiedlichen Bitpositionen abgelegt werden können. Auch hier ist ein Simple-Dual-Port-RAM die Grundlage des Speichers, sodass auch hier das Lesen und Schreiben zeitgleich und mit verschiedenen Geschwindigkeiten stattfinden kann.
Die Anforderungssignale beider Ring-Buffer, um einen neuen Speicherblock für den Lese-/Schreibzugriff anzufordern, sind miteinander verbunden. Dadurch ist sichergestellt, dass die Headerinformationen immer zu den Sample-Werten passen und beide Speicher den gleichen Füllstand aufweisen. Einen neuen Speicher(-block) zum Beschreiben kann nur der Video-Interpreter und ein neuen Speicher(-block) zum Lesen nur der MicroBlaze anfordern.

5.6.2. Video-Interpretation

Die Aufgabe des Video-Interpreters besteht in der Interpretation des proprietären Protokolls, in welchem die Sample-Werte des Radar-Sweeps sowie zusätzliche Informationen enthalten sind. Durch die begrenzte Datenlänge von Ethernet-Paketen kann es, je nach Betriebsmodus des Radar-Transceivers, dazu kommen, dass ein Video-Sweep über eine variable Anzahl an Ethernet-Pakete verteilt empfangen wird. Der Interpreter ist daher in der Lage, dies zu erkennen und den Anfang jedes Sweeps auszumachen. Ferner führt der Video-Interpreter Fehler- und Plausibilitäts-Überprüfungen durch. Auf diese Weise kann festgestellt werden, wenn Video-Sweeps fehlerhaft oder unvollständig sind. In diesem Fall wird der komplette Sweep verworfen und auf den nächsten Sweep gewartet. Es wird hierfür kein neuer Speicher(-block) angefordert, sodass durch den MicroBlaze und der DAC-Ansteuerung nur Sweeps weiterverarbeitet werden, die die Überprüfung bestanden haben. Aufgrund des Interpreter-Designs kann sichergestellt werden, dass keine Deadlocks aufgrund fehlerhafter und fehlenden Nachrichten entstehen können.
Im Zuge der Interpretation werden Sweep-Informationen für den MicroBlaze aufbereitet und in einem der konzeptionierten Ring-Speichern abgelegt. Im anderen Ring-Speicher werden hingegen die Video-Samples hinterlegt, um eine Verarbeitung durch die DAC-Ansteuerung zu ermöglichen.

5.6.3. DAC-Ansteuerung

Im folgenden wird die DAC-Ansteuerung erläutert. Anfänglich wird die Anpassung des DAC-Evaluationboards erläutert, bevor auf die HDL-Implementierung und dessen Timing eingegangen wird.

Anpassung des DAC-Evaluationboards

Der in Abschnitt 4.3.2 ausgewählte Digital-Analog-Umsetzer besitzt zwei integrierte DAC-Kanäle, welche jeweils mit einer Sample-Frequenz von bis zu 125 MSample/s betrieben werden können. Hierfür ist das DAC mit zwei verschiedenen Takten zu versorgen. Der erste Takt ist der Sample-Takt und gibt damit die Zeitpunkte zur Ausgabe der Analog-Werte an. Der zweite Takt ist der Daten-Takt und gibt dem DAC den Zeitpunkt vor, wann die Daten stabil anliegen und einzulesen sind. Die Übergabe der Daten geschieht mittels eines 12-Bit breiten Parallel-Interface, auf welches die Daten im DDR-Verfahren (Double Data Rate) gegeben werden. Auf diese Weise müssen z. B. alle für den ersten Kanal bestimmten Daten bei steigender Flanke, alle für den zweiten Kanal bestimmten Daten hingegen bei fallender Flanke stabil anliegen.
Angaben über spezielle Modi des DAC und das Format der Sample-Daten können über interne Register mittels einer SPI-Schnittstelle konfiguriert werden. Diese Schnittstelle ist allerdings nicht über den FMC-Stecker nutzbar, welcher die Verbindung zwischen DAC- und FPGA-Evaluationboard herstellt. Es bedürfte daher einer zweiten Verbindung zwischen den Boards. Für den hier gegebenen Anwendungsfall besteht jedoch keine Notwendigkeit, die Einstellungen zur Laufzeit zu verändern. Aus diesem Grund wird das DAC im sog. Pin-Mode betrieben. Dies bedeutet, dass relevante Einstellungen durch externe Beschaltung des DAC erfolgen. Durch entsprechende Anpassung der Bestückung ist dies auch beim Evaluationboard möglich.
Da das Evaluationboard für den Betrieb mit dem “Data Pattern Generator” von Analog Devices vorgesehen ist, welcher mittels des proprietären DPG2-Steckers angebunden werden kann, wird nur der Daten-Takt nicht aber der Sample-Takt über den Stecker geführt. Letzterer ist separat z. B. von einem Taktgenerator einzuspeisen. Für die Anbindung an das FPGA-Evaluationboard ist daher die Bestückung so angepasst, dass die Quelle für den Daten- und Sample-Takt die gleiche ist. Dies ist gemäß Datenblatt [1, S. 42] zulässig.
Das DAC-Evaluationboard erlaubt unterschiedliche Möglichkeiten, das Differenzsignal des DAC zu nutzen. Neben einer direkten Nutzung des DAC (entkoppelt über einen Transformator) und der Option einen Quadraturmodulator anzusteuern, besteht die Möglichkeit das Differenzsignal auf verschiedene Operationsverstärker zu geben. Das Evaluationboard sieht hier unterschiedliche Schaltungsmöglichkeiten vor, die durch Veränderung der Bestückung auf die eigene Anwendung angepasst werden können. Die hier gewählte Beschaltung sieht eine Ausgabe über einen “Single-Ended-Buffered”-Ausgang vor. Durch die so fehlende Differenzsignalverarbeitung verschlechtert sich das Hochfrequenzverhalten bzgl. des Einflusses von Störungen. Für den hier vorliegenden Anwendungsfall ist eine Weiterverarbeitung als einfache Signalleitung ausreichend und notwendig.
Bild 5.17 kann die notwendige Schaltung entnommen werden. Die Differenzströme werden hier

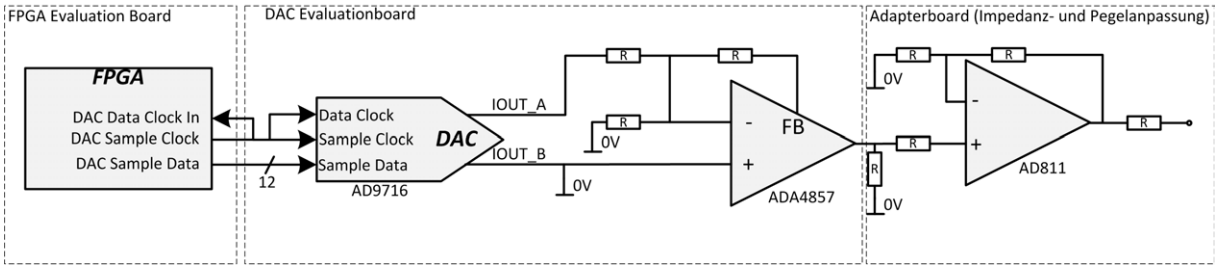


Bild 5.17.: Analoge Video-Verarbeitung durch das DAC-Evaluationboard und die entwickelte Adapterplatine

auf den Operationsverstärker gegeben, sodass dieser aus den Differenzströmen eine Spannung generiert. Zu Verfügung stehen hier zwei unterschiedliche Operations-Verstärker. Verwendet wird der ADA4857 [3]. Gegenüber dem ebenfalls zur Verfügung stehenden ADA4841 [4] weist dieser eine geringere Dämpfung bei höheren Frequenzen auf. Durch die Beschaltung des Plus-Einganges des ersten Operationsverstärkers mit Massepotenzial wird erreicht, dass die ausgegebene Spannung nur positiv ist. Das so ausgegebene Signal wird auf die Adapterplatine gegeben, die eine Pegelanpassung und Impedanzwandlung mittels eines zweiten Operationsverstärkers vom Typ AD811 [5] vornimmt, sodass die Signaleigenschaften für den Anschluss an ein Radar-Sichtgerät ausgelegt sind.

HDL-Implementierung

Die Anforderung an die HDL-Implementierung ist, einen Kanal des DAC unter Berücksichtigung des Parallelbus-Zugriffs im Double-Data-Rate-Verfahren (DDR) anzusteuern. Die Sample-Rate variiert je nach der eingestellten Reichweite der Radarantenne und wird zusätzlich innerhalb eines Sweeps mehrfach geändert. Die Ursache hierfür liegt im abnehmenden Informationsgehalt der reflektierten Echos mit zunehmender Entfernung.
Dem Bild 5.18 können die Takterzeugung (hellgrün), der Datenpfad (hellblau) und der Steuer-

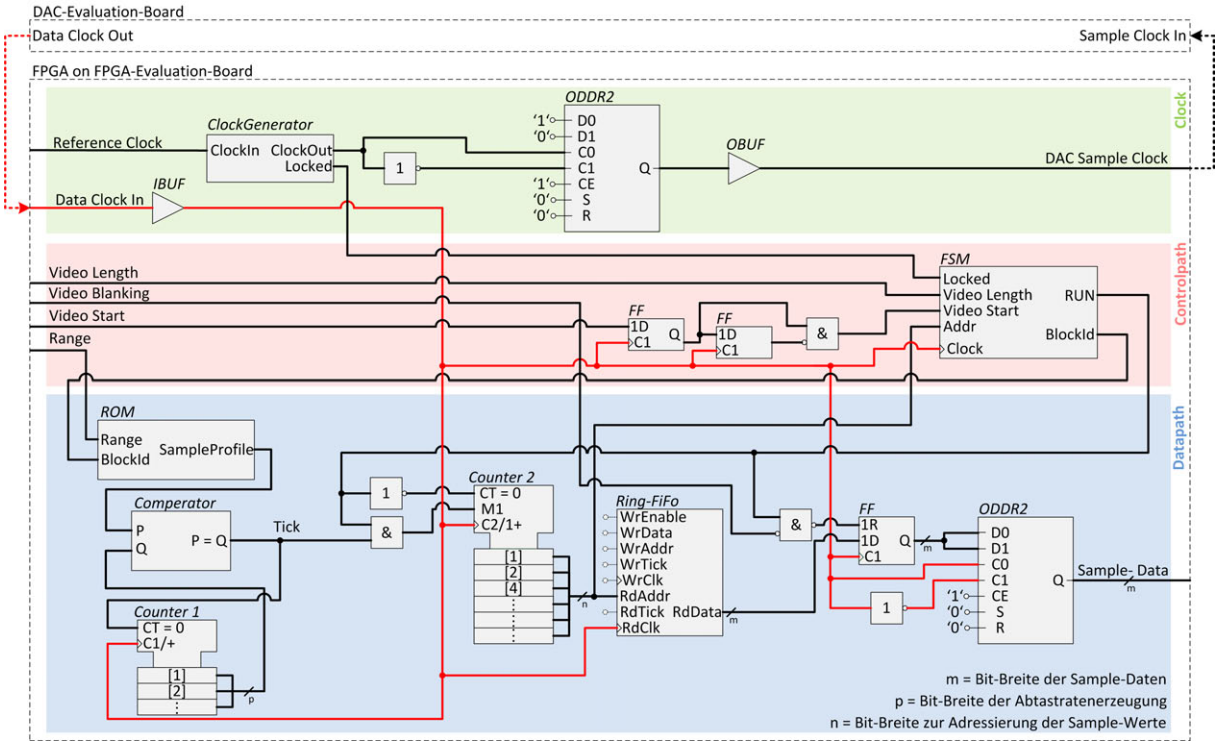


Bild 5.18.: DAC-Ansteuerungslogik zur Ausgabe von Radar-Videos unter Verwendung des AD9716

pfad (hellrot) entnommen werden, wie sie für die Ansteuerung des DACs implementiert wurden. Da das Evaluationboard keine externe Taktquelle besitzt, die die erforderliche Basis-Frequenz erzeugt, findet die Takterzeugung mittels eines PLL-basierten Frequenz Synthesizers statt. Dieser zeichnet sich durch den kleinst möglichen Taktjitter[8], welcher über eine FPGA interne Taktgenerierung erzeugt werden kann. Xilinx empfiehlt ODDR2-Register [65, 66] einzusetzen, um

[8]Nach [77] liegt der Taktjitter bei der gewählten Frequenz bei 263.25 ps Pk-to-Pk.

den Takt an einen Ausgangspin zu führen. Nur so kann eines der 16 zur Verfügung stehenden globalen Taktnetzwerke des Spartan-6 verwendet werden. Die Taktung von Daten- und Steuerpfad wird einheitlich durch den nach Bild 5.18 vom DAC-Evaluationboard zurückgegebenen Sample-Takt durchgeführt. Die Zurückführung des identischen Taktes über das DAC-Board wurde durchgeführt, da es nur so erreicht werden konnte, dass die Implementierungstools ein weiteres globales Taktnetzwerk für die Taktung der DAC-Ansteuerungs-Logik verwenden. Eine zeitgleiche Nutzung des durch den Frequenz Synthesizer erzeugten Taktes für interne Logik und das Herausführen des Taktes auf einen externen Pin unter Verwendung der globalen Taktnetzwerke konnte anders nicht erreicht werden.
Kommt vom MicroBlaze die Anforderung, durch Setzen des Triggers den nächsten Video-Sweep auszugeben, setzt der Zustandsautomat das RUN-Bit. Zusätzlich gibt der MicroBlaze das zu wählende Sample-Rate-Profil in Abhängigkeit der Reichweite vor und ob ein "Blanking" des Videos durchzuführen ist, d. h. die Ausgabe der Abtastwerte unterdrückt werden soll, sodass nur der Trigger generiert wird. Dies wird für die Sektorblanking-Funktion[9] benötigt, da in diesem Fall alte, nicht überschriebene, Sample-Werte im Buffer liegen, die sonst ausgegeben werden würden.
Die Verarbeitung im Datenpfad sieht vor, dass aus einem Read-Only-Memory (ROM) unter Angabe der Reichweite und der aktuell zu bearbeitenden Sektion innerhalb eines Sweeps, ein Zählerwert geladen wird, der die Anzahl an Takten definiert über die sich die Ausgabe der Sample-Werte nicht verändert. Auf diese Weise wird die Abtastrate nicht auf physikalischer, sondern auf digitaler Ebene definiert. Sobald das nächste Sample auszugeben ist, gibt ein Komparator das Signal zur Inkrementierung der Speicheradresse. Es erfolgt automatisch das Auslesen aus dem in Abschnitt 5.6.1 beschriebenen Ring-Speicher. Im Bild 5.18 wird dabei nicht das Einschreiben von Daten in den Ring-Speicher dargestellt. Über ein FlipFlop (zur Verbesserung des Timings, siehe hierzu den folgenden Abschnitt) erfolgt die Ausgabe der Daten durch 12 ODDR2-Register. Diese erlauben die Ausgabe der Daten im Double-Data-Rate-Format. Die Ausgabe wird beim "Blanking" unterdrückt.

Timing

Für die Realisierung der DAC-Ansteuerung sind verschiedene Ansätze der Implementierung des Designs verfolgt worden. Dies wurde notwendig, da das Timing anfänglich nicht reproduzierbar eingehalten werden konnte. Eine Nichteinhaltung der Timing-Constraints kann feststellbare Auswirkungen auf die Signalqualität haben. Bild 5.19 zeigt beispielhaft die Ausgabe eines Test-Video-Sweeps bei Einhaltung der Timing-Constraints (unterer Bildabschnitt) und bei einer Nichteinhaltung der Timing-Constraints (oberer Bildabschnitt).
Die im vorherigen Abschnitt vorgestellte Realisierung ist damit das Ergebnis verschiedener durchgeführter Design-Realisierungen. In ersten Realisierungen wurde u. a. mittels zusätzlicher Logik eine konfigurierbare Taktteilung des vom Takt-Generators gelieferten Grundtaktes vorgenommen, um die geforderten Sample-Frequenzen physikalisch zu unterstützen. FPGAs der Spartan-6 Familie besitzen spezielle Routing-Ressourcen für Takte, die aufgrund der Taktteilung, nicht von den Implementierungs-Tools genutzt werden konnten. Durch die im vorherigen Abschnitt beschriebene Implementierung ist dies hingegen möglich.
Um eine statische Timing-Analyse zu ermöglichen und den Implementierungs-Tools das zu

[9] Beim Sektorblanking sendet der Radar-Transceiver keine elektromagnetische Energie in einem festgelegten Winkel-Sektor.

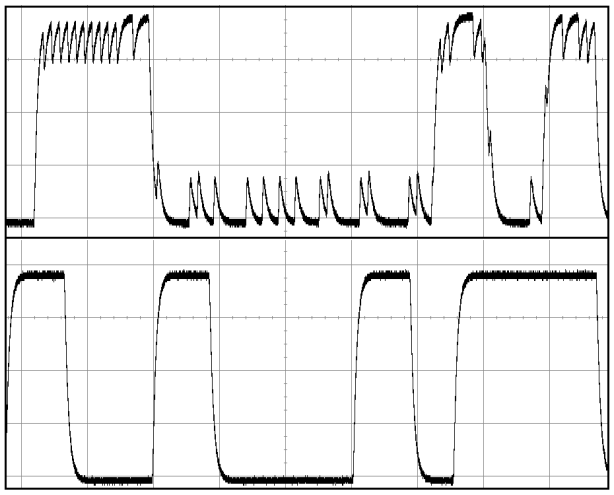


Bild 5.19.: Auswirkung eines schlechten Timings auf DAC-Ausgabe (Oben: Nichteinhaltung der Timing-Constraints; Unten: Einhaltung der Timing-Constraints)

erreichende Timing mitzuteilen, sind Timing-Constraints definiert worden. Hierbei handelt es sich zum einen um Period-Constraints zur Beschreibung der minimalen Taktfrequenzen für den Sample-Takt und den Daten-Takt, zum anderen werden Offset-Out-Constraints eingesetzt. Diese beschreiben in diesem Fall, wann die Sample-Daten vor der aktiven Flanke anliegen müssen und wie lange eine Gültigkeit dieser Daten sichergestellt sein muss. Das Datenblatt gibt hier Minimumwerte vor [1, S. 7]. Da die Ausgabe im Double-Data-Rate-Verfahren (DDR) geschieht, muss ein Offset-Constraint für die steigende und fallende Flanke definiert werden.

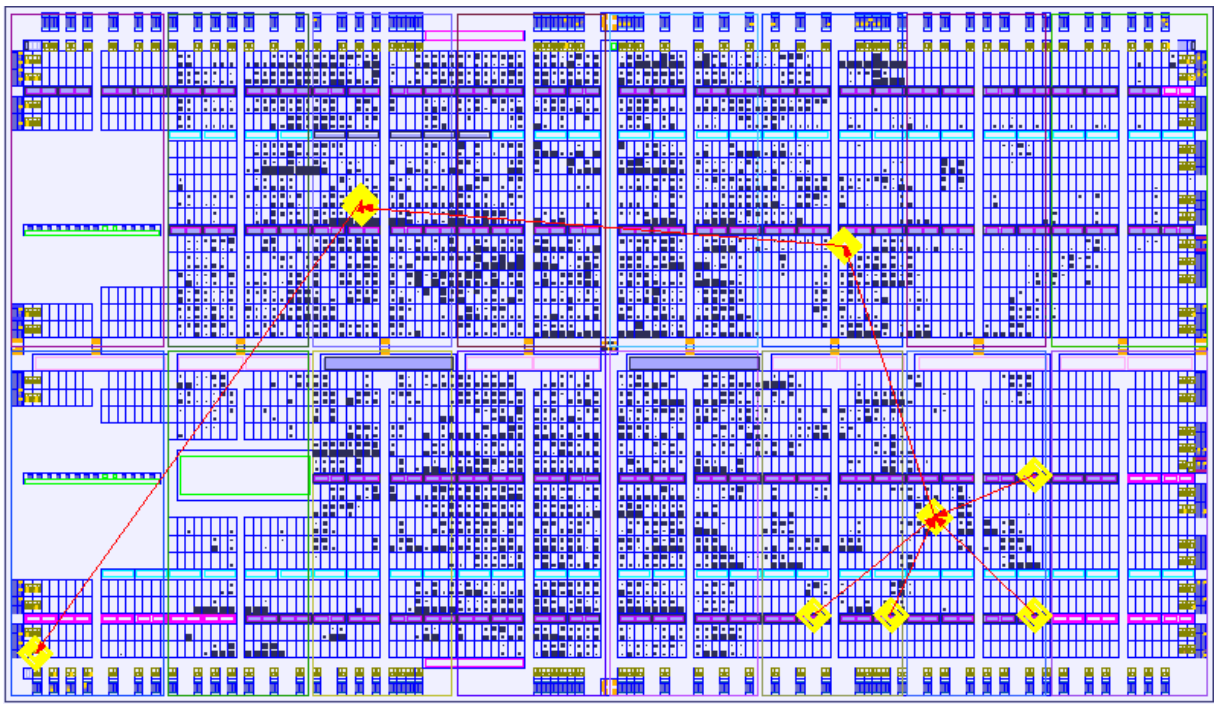


Bild 5.20.: Routing einer Datenleitung vom BlockRAM zum ODDR2-Register (Erstellt mittels PlanAhead)

Durch die statische Timing-Analyse zeigte sich, dass der Datenpfad zwischen BlockRAM und ODDR2-Register das notwendige Timing nur knapp und mitunter nicht zuverlässig reproduzierbar einhalten kann. Die Ursache ist dabei weniger in der HDL-Beschreibung und damit in Logik-bedingten Verzögerungen zu finden, sondern vielmehr in den Routing-Verzögerungen.
Bild 5.20 zeigt das Routing eines Pfades zwischen den BlockRAM-Blöcken (unten rechts im Bild) und dem ODDR2 Register (unten links im Bild), nahe des Ausgang-Pins. Die roten Pfeile stellen das grobe Routing dar. Das Problem hierbei ist, dass 38 % der zur Verfügung stehenden BlockRAM Blöcke für den Video-Buffer eingesetzt werden. Die Gesamtauslastung der BlockRAM-Blöcke liegt sogar bei 89 %. Es liegt daher nahe, dass die Pfade sich mitunter über große Bereiche des FPGAs erstrecken müssen und dies zulasten des Timings durch Routing-Delays geht. Um das Timing zu verbessern, gibt es nun zwei Lösungsmöglichkeiten. Die erste Möglichkeit ist die Verwendung von Physical Constraints, die vorschreiben, in welchen Bereichen des FPGA die Logik zur Ansteuerung des DAC inkl. der BlockRAM-Blöcke zu platzieren ist. Die zweite Möglichkeit ist, dem Implementierungs-Tool durch HDL-Design-Anpassungen Hilfestellung zu geben. Ersteres könnte zwar das Timing der DAC-Ansteuerung verbessern, wird sich aber negativ auf das Timing anderer Komponenten, etwa vom MicroBlaze, auswirken. Dieser hat ebenfalls einen hohen Bedarf an BlockRAM-Ressourcen. Daher ist Letzteres durchgeführt worden. Das Ergebnis hierzu ist im Bild 5.20 bereits zu sehen: Durch Hinzufügen einer Kaskade von FlipFlops in den Datenpfad wird den Implementierungstools die Möglichkeit gegeben, diese frei über das FPGA zu verteilen und dadurch den zuvor kritischen Pfad zu verkürzen. Die Latenz nimmt hierdurch zu. Durch die Konfigurierbarkeit der Zeitdifferenz zwischen Triggerflanke und Begin der Videoausgabe ist dies jedoch nicht kritisch. Ohne FlipFlops würde es zu einer direkten Verbindung zwischen BlockRAM und ODDR2-Register kommen.

KAPITEL 6

Software Implementierung

Im folgenden Kapitel wird auf die Software-Implementierung des MicroBlaze eingegangen. Abschnitt 6.1 beschäftigt sich hierzu mit übergeordneten Design-Entscheidungen, bevor in Abschnitt 6.2 die Anbindung an die Hardware und im Zuge dessen die Verarbeitung von Ethernet-Nachrichten thematisiert wird. Abschnitt 6.3 erläutert die Erzeugung der verschiedenen Pulse, bevor abschließend in Abschnitt 6.4 auf die Implementierung einer Funktion zum Updaten des FPGA-Konfigurations-Image über Ethernet eingegangen wird.

6.1. Allgemeines

Ziel der Design-Realisierung der MicroBlaze-Software ist in erster Linie die Berechnung und Ausgabe des Trigger-Pulses, des Azimuth-Clock-Pulses (ACP) und des Azimuth-Reference-Pulses (ARP). In zweiter Linie soll der MicroBlaze die Konfiguration und Steuerung der Hardware übernehmen.
Die Abarbeitungen der Aufgaben findet ausschließlich mittels Interruptroutinen statt und nicht durch Polling in der main-Schleife. Letzteres würde dazu führen, dass die Abarbeitungsrate in Abhängigkeit der Prozessorauslastung variieren kann und nicht direkt vorhersagbar ist. Für die Berechnung der Pulsausgabezeitpunkte wäre dies unerwünscht. Neben der Generierung und Berechnung der Analog-Pulse wird auch der Empfang von UDP-Nachrichten einschließlich der notwendigen Interpretation und ggf. Reaktion interruptbasiert abgearbeitet.
Unter Verwendung des “XPS_INTC” [70] Interrupt-Controllers können durch den MicroBlaze bis zu 32 Interrupt-Quellen verarbeitet und ein prioritätengesteuerter Aufruf der zugehörigen Interrupt-Service-Routinen (ISR) durchgeführt werden. Dabei können Interrupt-Handler nicht durch den Interrupt-Controller unterbrochen werden, wenn ein höher priorisierter Interrupt-Request während der Abarbeitung eines Interrupt-Handlers gestellt wird (siehe Abschnitt 4.2.2). Dies führt daher auch für den am höchsten priorisierten Interrupt zu nicht exakt vorhersagbaren Interrupt-Latenzen (Prioritätsinversion). Damit entsteht jedoch auch der Vorteil, dass alle Funktionsaufrufe in einer ISR praktisch atomar[1] durchgeführt werden, ohne dass gesonderte Maßnahmen, wie die Deaktivierung der Interrupts, notwendig sind. Dies müsste sonst bei nicht “Interrupt-Sicheren”-Funktionen geschehen.
Ein weiterer im Design zu berücksichtigender Punkt ist, dass der Interrupt-Controller nur einen Interrupt-Request pro Interrupteingang speichert. Treten mehrere Interrupt-Requests einer einzigen Interrupt-Quelle auf, ohne dass es zu einer Abarbeitung dieses Interrupts kommt, wird nur der letzte Interrupt-Request gespeichert, sodass Interrupts verloren gehen.

[1] Atomare Aufrufe sind nicht nicht durch eine andere Abarbeitung unterbrochen werden. Klassische atomare Aufrufe werden meist innerhalb eines Taktzyklus verarbeitet.

6.2. Hardware-Software-Interface

Die in Abschnitt 4.2.4 konzeptionierte Hardware-Software-Schnittstelle basiert auf Fast-Simplex-Links. In Abschnitt 5.3.2 ist hierzu bereits das notwendige Interface auf Seiten der Hardware implementiert worden. Auf Seiten der Software sind drei Zugriffsfunktionen realisiert, die einen Zugriff durchführen können. Neben der eigentlichen Datenübertragung besteht die Aufgabe dieser Funktionen darin, das Datenwort zur Initiierung der Kommunikation[2] zu generieren und zu übertragen:

- Zwei der Zugriffsfunktionen erlauben das Lesen bzw. Beschreiben ausschließlich von Hardware-Registern. Die notwendige Flexibilität ist hier minimal, sodass Rechenzeit eingespart werden kann.
- Die dritte Zugriffsfunktion ist flexibler ausgelegt. Diese kann Lese-/Schreibzugriffe auf die Hardware-Register und auf die beiden UDP-Buffer durchführen. Wahlweise können die Daten im Burst-Mode übertragen werden, bei dem mehrere Datenwörter bei nur einer Kommunikations-Initialisierung übertragen werden. Im Gegensatz zu den ersteren Funktionen entsteht durch die Flexibilität ein höherer Rechenaufwand. Um die Anzahl an Kopiervorgängen zu minimieren, ist die Datenübergabe mittels Pointer realisiert.

Die eigentlichen Zugriffe auf die FSL-Verbindung wird mittels put- und get-Assemblerbefehlen der MicroBlaze-ISA (Instruction Set Architecture) durchgeführt, die mittels Makros im C-Code eingebunden sind. Diese Assemblerbefehle können, jeder für sich, atomar ausgeführt werden. Die Zugriffsfunktion selbst ist hingegen nicht atomar. Diese beinhalten mindestens zwei Zugriffe auf die Hardware: Ein Zugriff zur Übertragung des Headers und mindestens ein weiterer Zugriff für die Nutzdaten. Für eine fehlerfreie Ausführung der Funktion muss daher eine Unterbrechung ausgeschlossen werden können, sofern bei der Unterbrechung selbst auch Zugriffe auf das HDL-Design über diese Schnittstelle durchgeführt werden sollen. Dies liegt darin begründet, dass sobald die Kommunikation initialisiert ist, diese auch vollständig durchgeführt werden muss. Da Interrupts nicht unterbrochen werden können und keine Zugriffe in der main-Schleife erfolgen, ist dies sichergestellt.

Das Senden von Ethernet-Nachrichten ist prinzipiell von jeder Routine aus durchführbar. Anders das Empfangen, hier müssen die Nachrichten zentralisiert über eine ISR abgeholt und interpretiert werden. Für den Austausch der Daten ist ein spezifisches Protokoll implementiert, welches neben den Nutzdaten Angaben über die Art des Inhaltes macht. Zum Versenden von Nachrichten ist deshalb eine zusätzliche Funktion implementiert, welche den Sendevorgang abstrahiert. Neben der Generierung des Protokolls wird auf Basis der FSL-Zugriffsfunktionen eine Übertragung der Daten durchgeführt und das Senden durch den Ethernet-Stack eingeleitet. Für die Übertragung von Ethernet-Nachrichten wird die dritte Funktion, die den Burst-Mode und den Zugriff auf die UDP-Buffer unterstützt, eingesetzt. Diese erreicht die notwendige Flexibilität, wird hierfür jedoch auch mehr Rechenzeit benötigen.
Die Verarbeitung empfangener Ethernet-Pakete wird durch einen Interrupt-Request eingeleitet, der an den MicroBlaze gestellt wird. Der MicroBlaze prüft im zugehörigen Interrupt-Handler, ob es sich um den Empfang einer UDP-Nachricht handelt oder um ein ARP-Reply, welches aufgrund eines ARP-Requests empfangen wird. Bei Ersterem wird ein Interpreter aufgerufen, der

[2] Details zum Aufbau des Datenwortes sind der Entwicklungs-Dokumentation in Anhang A.3.1 zu entnehmen.

die Nachricht aus dem Speicher abruft, interpretiert und eine hiervon abhängige Steuer-Aktion ausführt. Hierzu zählt z. B. die An-/Abmeldung an Multicast-Gruppen, die Konfiguration der Pulsgenerierung oder die Verarbeitung eines Updates.
Nach jeder Abarbeitung wird geprüft, ob weitere Nachrichten im Ring-Buffer enthalten sind. Erst wenn alle Nachrichten interpretiert und verarbeitet wurden, wird die Interrupt-Service-Routine wieder verlassen. Der Interrupt-Request kann damit die geringste Priorität im System bekommen, da es nicht kritisch ist, wenn Interrupts verloren gehen.
Im Zuge der Initialisierung des MicroBlaze wird ein ARP-Request gesendet, um die MAC-Adresse des Teilnehmers zu erfragen, mit dem zur Steuerung der Elektronik eine Unicast-Verbindung unterhalten werden soll. Die Aufgabe des Interrupt-Handlers ist es, beim Empfang eines ARP-Reply, den Ethernet-Stack auf die Kommunikation mit der so empfangenen MAC-Adresse zu konfigurieren.

6.3. Puls-Erzeugung

In Abschnitt 2.3 sind die grundlegenden analogen Signale einer Radar-Anlage des hier vorliegenden Typs erläutert. Diese Signale werden durch den Radar-Transceiver selbst unter Einhaltung harter Echtzeitanforderungen digitalisiert. Die Entfernung eines Echos ist dann nicht mehr durch die Zeit zwischen Senden und Empfangen elektromagnetischer Energie gegeben, sondern durch die Samples pro nautischer Meile definiert. Auch der azimutale Winkel wird nicht mehr durch die Zeitmessung zwischen winkelstabilen Pulsen ermittelt, sondern als Digitalwert weiter verarbeitet. Es ist damit nicht mehr von Bedeutung, zu welchem Zeitpunkt, und damit auch in welchem zeitlichen Abstand zueinander, die Sweeps aufgenommen wurden, wodurch weichere Echtzeitanforderungen an das Sichtgerät gestellt werden können.
Die Aufgabe der Software besteht darin, die Ausgabezeitpunkte für die analogen Pulse wieder unter Einhaltung harter Echtzeitanforderungen zu rekonstruieren und zu generieren. Durch die in den Ethernet-Paketen abgelegten Zeitstempel wird vorgegeben, welchen zeitlichen Abstand die Trigger-Pulse zueinander aufweisen müssen. Durch die digitalen Winkelinformationen können zudem den Zeitdifferenzen die Winkeldifferenzen zugeordnet werden. Die Pulse ACP und ARP sind dann so zu generieren, dass ein nachfolgendes, analog arbeitendes Sichtgerät durch Zählen der Pulse und messen der Zeitabstände zwischen den Pulsen dem Trigger-Puls einen Winkel zuordnen kann, der dem digitalen Winkelwert entspricht. Die Trigger-Pulse können dabei weder als winkel- noch als zeitstabil angenommen werden. Die Pulse ACP und ARP sind hingegen winkelstabil zu generieren.
Durch Bildung der Zeit- und Winkeldifferenz zwischen zwei Sweeps kann die Restzeit zu einem beliebigen Winkel berechnet werden. Da die Winkelgeschwindigkeit der Antennen, z. B. aufgrund von Unterschieden in der Windlast, schwanken kann, ist die Bestimmung der Pulsausgabe nur für einen, dem aktuellen Winkel nahen Winkel, näherungsweise richtig. Es sind dabei nicht nur Schwankungen der Winkeldrehgeschwindigkeit bei der Pulsgenerierung zu berücksichtigen, sondern automatisch auch die Pulsgenerierung auf unterschiedliche Antennendrehgeschwindigkeiten anzupassen.
Ein Jitter der Ausgabezeitpunkte der Pulse führt nach Abschnitt 4.2.1 zu Winkelfehlern, da die ACP- und ARP-Pulse von den Sichtgeräten weiterhin als winkelstabil angenommen werden. Ein zeitliches Jittern des Triggers führt ebenfalls zu Fehlern in der Bestimmung des azimutalen Winkels, da ein Sichtgerät in diesem Fall annimmt, dass der Sendeimpuls des Radar-Transceivers

früher oder später generiert und damit in eine andere Richtung gesendet wurde. Ein Jitter der Ausgabeperiode kann prinzipiell durch folgende Einflussfaktoren hervorgerufen werden:

- Durch höher priorisierte Aufgaben oder durch nicht unterbrechbare Abarbeitungen kann es zu einer Verzögerung im Aufruf der ISR und damit zu einer verzögerten Ausgabe des Pulses kommen.
- Bei falscher Konfiguration des Timers können sich Verzögerungen zwischen Interrupt-Request und Rekonfiguration des Timers auf die Timer-Periode auswirken und damit einen Winkelfehler verursachen.
- Bei der mathematischen Berechnung der Ausgabezeitpunkte kann es zu Berechnungsfehlern kommen, wenn mit Festkommazahlen gerechnet wird.

Die Priorisierung der Interrupts erfolgt unter Berücksichtigung der Auswirkungen eines Jitters der Pulse auf den Gesamtwinkelfehler:

- Die höchste Priorität ist damit der Erzeugung des Azimuth-Reference-Pulses zuzuordnen. Dieser kommt einmal pro Umlauf bei einer definierten Antennenausrichtung. Der hier entstehende Winkelfehler wirkt sich damit für eine vollständige Antennenumdrehung aus.
- Die zweithöchste Priorität erhält der Timer-Interrupt-Request zur Ausgabe des ACP-Puls. Dessen Wiederholfrequenz liegt i. d. R. niedriger als die Pulsfolgefrequenz des Triggers. Ein Winkelfehler beim ACP wird sich daher auf mehrere Trigger-Pulse auswirken.
- Die dritthöchste Priorität erhält der Interrupt-Request zur Erzeugung von Trigger-Pulsen.

Die hier festgelegten Prioritäten bestimmen nur die Abarbeitungsreihenfolge, wenn mehrere Interrupts zeitgleich, bzw. während der Abarbeitung eines anderen Interrupt-Handlers, generiert werden. Eine Unterbrechung der Abarbeitung von Interrupts, um höher priorisierte Interrupts zu bearbeiten, ist nicht möglich.

6.3.1. Berechnung der Trigger-Ausgabe

Die Berechnung der Ausgabeperiode des Triggers stellt die einfachste Pulsberechnung dar. In den Header-Informationen des proprietären Protokolls befindet sich ein Zeitstempel. Dieser ist gemäß Abschnitt 5.6.2 durch den MicroBlaze auslesbar. Durch die Bildung der Differenz des Zeitstempels des vergangenen Sweep und des aktuellen Sweep kann die Periode ermittelt werden, mit der der Radar-Transceiver ursprünglich elektromagnetische Pulse ausgesendet hat. Diese Periode ist zu rekonstruieren indem ein Timer auf diese Periode konfiguriert wird (siehe Bild 6.1). Dieser Timer löst nach Ablauf der jeweiligen Periode ein Interrupt-Request aus. Der

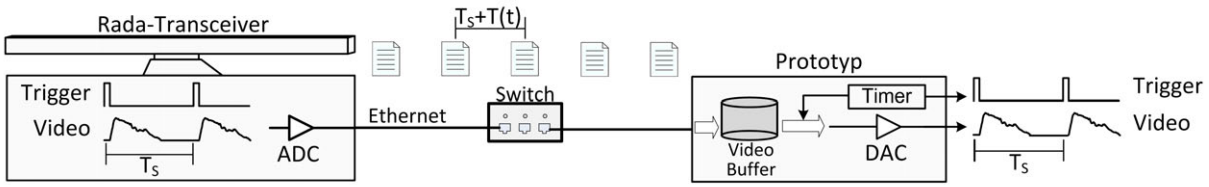


Bild 6.1.: Schematische Darstellung der Trigger- und Videoverarbeitung

zugehörige Interrupt-Handler generiert den zu rekonstruierenden Trigger-Impuls, indem er über

die FSL-Verbindung ein Bit in einem der Register des Konfigurations-Interface setzt. Durch das Setzen des Triggers wird durch die Hardware auch automatisch die Ausgabe des Videos eingeleitet. Der zeitliche Abstand zwischen Trigger-Flanke und Beginn der Videoausgabe kann dabei durch den MicroBlaze konfiguriert werden. Der Trigger-Timer gibt damit vor, wann und wie schnell der Video-Buffer entleert werden soll.
Durch die Rekonstruktion der Tiggerperiode wird der Trigger-Interrupt-Handler im Mittel immer mit der gleichen Frequenz aufgerufen, wie neue Radar-Informationen und damit auch neue Grundlagen für die Berechnung der Pulse ACP und ARP zur Verfügung stehen. Aus diesem Grund finden im Trigger-Handler auch die Berechnungen der Ausgabezeitpunkte der anderen Pulse statt.
Die Zeit, die zwischen Interrupt-Request und Ausführung der ISR vergeht, kann nicht als konstant angenommen werden, da diese durch die Abarbeitung der anderen Interrupts verzögert werden kann. Es ist daher wichtig, dass die Konfiguration des Trigger-Timers auf eine Weise geschieht, dass es zu keinen Auswirkungen auf die mittlere Ausgabeperiode des Triggers kommen kann. Der Buffer würde sonst über-/unterlaufen. Der Timer ist daher so konfiguriert, dass ein automatisches Laden des Reload-Registers nach Ablauf des Timers durchgeführt wird. Der Trigger-Handler aktualisiert dann nur das Reload-Register, setzt dabei aber nicht das "XTC_CSR_LOAD_MASK"-Bit im Control-Status-Register des Timers mit dem ein manuelles Laden des Timer-Registers mit dem Wert des Reload-Register ausgelöst werden kann.
In der Startphase wird das Reload-Register des Trigger-Timers mit einem Default-Wert geladen, sodass dieser periodisch versucht, Daten aus dem Ring-Buffer zu lesen. Sobald hier plausible Inhalte vorliegen, wird sich auf die angegebene Periodendauer synchronisiert. Dabei ist es Zufall, wie hoch der Füllstand des Video-Buffers in diesem Moment ist. Deshalb ist der MicroBlaze in der Lage, den Füllstand des Video-Buffers auszulesen. Ist der Füllstand in der Startphase kritisch bzgl. der Wahrscheinlichkeit eines Über-/Unterlaufes wird die Periodendauer mit einem Offset von $\pm 1\,\mu s$ beaufschlagt, sodass der Füllstand des Buffers langsam aus dem kritischen Bereich herausgeführt und die Wahrscheinlichkeit eines Über- oder Unterlaufes damit deutlich gesenkt wird, wie in Abschnitt 7.3 gezeigt werden wird. Dieser Offset wird nur in der Startphase erzeugt und liegt deutlich niedriger als der erlaubte Jitter. Es benötigt mehrere Tausend Sweeps, um den Füllstand des Buffers um einen einzigen Sweep zu ändern. Die Korrektur wird daher als akzeptierbar angesehen.

6.3.2. Berechnung der ACP-Ausgabe

Die Berechnung der Ausgabezeitpunkte des ACP-Pulses kann durch Gleichung 6.1 beschrieben werden. Die Zeitdifferenz zwischen zwei Trigger-Pulsen t_{dif} wird aus dem Zeitstempel des aktuellen Sweeps und dem des vergangenen Sweeps errechnet, ebenso die Winkeldifferenz zwischen zwei Sweeps w_{dif}. Für beide Differenzen wird automatisch ein Überlauf der Operanden erkannt und herausgerechnet. Der Winkel w_{min} stellt die Winkeldifferenz zwischen zwei zu generierenden ACP-Pulsen dar, die auf Basis einer, zur Laufzeit änderbaren, Anzahl zu generierender Pulse berechnet wird.

$$t_{ACP} = \frac{t_{dif}}{w_{dif}} \cdot w_{min} \tag{6.1}$$

Der Vorteil dieser Gleichung ist zum einen die geringe Anzahl an Rechenoperationen, zum anderen wird, selbst wenn ein Sweep fehlen sollte, die Zeit t_{ACP} richtig berechnet. Der Fehler

würde sich herauskürzen, da Zeit- und Winkeldifferenz näherungsweise linear zueinander ansteigen. Problematisch ist, dass die Operanden von Gleichung 6.1 Festkommazahlen sind. Bei Divisionen hat dies zur Folge, dass bei einer nachfolgenden Rechenoperation Nachkommastellen nicht mehr berücksichtigt werden und damit ein Fehler im Ergebnis entstehen kann. Ist die Anzahl der zu generierenden Pulse gering, ist w_{min} groß, sodass ein Fehler in der Division von t_{dif} mit w_{dif} die größten Auswirkungen haben kann, da dieser Fehler mit w_{min} multipliziert wird. Dreht die Antenne zudem sehr langsam, nimmt der Quotient aus t_{dif} und w_{dif} zu, sodass auch ein Genauigkeitsfehler in w_{min} maximale Auswirkungen auf das Ergebnis bekommt. Die Ungenauigkeit in w_{min} entsteht, wenn $2^{16}(\hat{=}360°)$ dividiert durch die Anzahl zu generierenden Pulse keine natürliche Zahl ergibt. Im Anhang A.2.1 ist hergeleitet, wie hoch dieser Fehler maximal werden kann.
Wird keine Skalierung der Operanden vorgenommen, so liegt der maximale Fehler pro Berechnung bei $t_f < 919.59\,\mu s$, wenn die Operanden im spezifizierten Bereich liegen. Dabei hat t_{dif} die Auflösung von einer Mikrosekunde und die Winkelangabe wird entsprechend mit einer Auflösung von $1/65536$ angegeben. Der maximale Fehler ist nach Abschnitt 4.2.1 nicht akzeptierbar. Die Anforderungen können nur erfüllt werden, wenn t_{dif} und w_{min} skaliert werden.

Eine Skalierung birgt ein zunehmendes Risiko eines Überlaufes aufgrund begrenzter Bitbreiten. Um dies für eine Skalierung zu bestimmen, sind Berechnungen angestellt worden, in welchen Wertebereichen t_{dif}, w_{dif} und w_{min} liegen dürfen, damit es zu keinem Überlauf bei der jeweiligen Skalierung kommt. Die Herleitung ist dem Anhang A.2.2 zu entnehmen.
Die Berechnungen haben gezeigt, dass wenn t_{dif} und w_{min} um den Faktor 1000 skaliert wer-

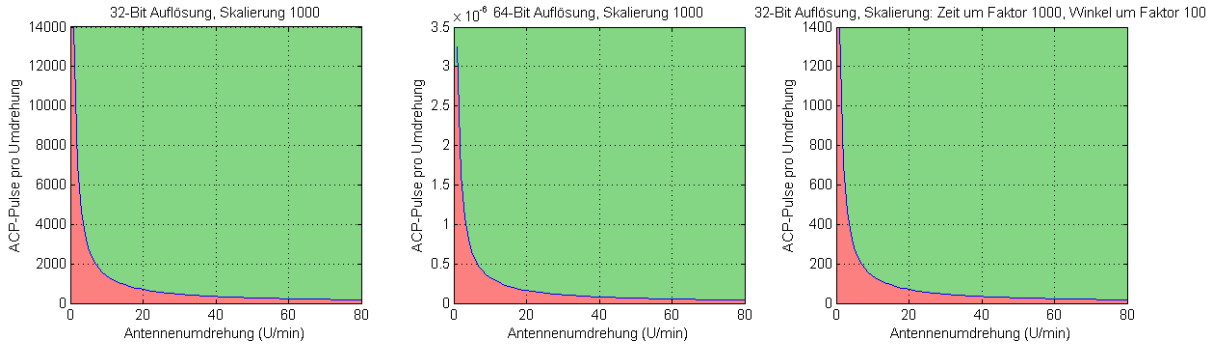


Bild 6.2.: Überlauf bei der Berechnung des ACP in Abhängigkeit der Antennendrehgeschwindigkeit, der Anzahl an ACP-Pulsen pro Umdrehung sowie des Skalierungsfaktors und genutzter Bitbreite

den, ein maximaler Einzelfehler von $t_f < 919.59\,ns$ entsteht, wenn die Anzahl zu generierender Pulse und die Antennendrehgeschwindigkeit gering ist. Dieser Einzelfehler kann sich prinzipiell mit jedem Puls innerhalb eines Umlaufes akkumulieren. Dieser Gesamtfehler wird am größten, wenn viele Pulse zu generieren sind und dabei die Antenne die maximale Drehgeschwindigkeit aufweist. Der Gesamtfehler bleibt mit $t_{fges} < 440.59\,\mu s$ ($\hat{=}0.076°$) unter den spezifizierten Grenzen sodass die Anforderungen erfüllt werden. Bild 6.2 zeigt im linken Diagramm in Abhängigkeit der Antennenumdrehungen pro Minute die Anzahl an ACP-Pulsen pro Umlauf, die konfiguriert werden dürfen, damit es zu keinem Überlauf bei dieser Skalierung kommt. Der grüne Bereich stellt dabei den erlaubten Wertebereich dar, der rote Bereich den Wertebereich, indem es zum Überlauf kommt. Dem Bild zu entnehmen ist, dass bei dieser Bitauflösung die

Skalierung zu hoch ist, sodass die Anforderungen an die notwendigen Zahlenbereiche nicht erfüllt werden können. Bei einer minimalen Antennen-Umdrehung von 20 RPM kann die minimal geforderte Anzahl von 72 ACP-Pulsen nicht erreicht werden. Im mittleren Diagramm des Bildes 6.2 wird daher, bei gleicher Skalierung, die Bitauflösung auf 64 Bit erhöht, wodurch die Anforderungen deutlich erfüllt werden können. Durch die höhere Bitauflösung steigt die benötigte Rechenzeit zur Bearbeitung der Interrupt-Service-Routine von maximal $18.66\,\mu s$ bei 32 Bit auf maximal $48.87\,\mu s$ bei 64 Bit an. Die nicht unterbrechbare Abarbeitung der ISR übersteigt damit den zulässigen Gesamtjitter von $45.78\,\mu s$. Eine Erhöhung der Bitbreite ist damit nicht realisierbar. Die Skalierung des Winkels w_{min} wird daher auf den Faktor 100 gesenkt. Eine überlauffreie Verarbeitung bei einer Auflösung von 32 Bit ist dann sichergestellt, wie dem rechten Diagram in Bild 6.2 entnommen werden kann. Es gelten hier gemäß der Herleitung im Anhang A.2.2 die Grenzen, die durch Gleichung 6.2 und 6.3 angegeben werden. Die Größen t_{dif}, w_{dif} und w_{min} sind hierzu in allgemeingültigere Größen umgerechnet: U steht für die Anzahl der Antennenumdrehung pro Minute, P für die Anzahl an ACP-Pulsen pro Umdrehung und F für die Pulsfolgefrequenz des Triggers in Hertz.

$$10.45 \leq \log_2 U + log_2 P \tag{6.2}$$

$$F \geq 0.233\,Hz \tag{6.3}$$

Die minimale Frequenz hängt ausschließlich von der Bitbreite des Produktes aus der Zeitdifferenz t_{dif} zwischen zwei Trigger-Pulsen und dem Skalierungsfaktor ab. Dies liegt darin begründet, dass die Pulsfolge-Frequenz indirekt auch in der Winkeldifferenz enthalten ist. Aufgrund der Division wirkt sich die Bitbreite so nicht auf die Bitbreite des errechneten Quotienten aus. Durch die geringere Skalierung steigt der Einzelfehler auf $t_f < 1.331\,\mu s$ an. Im ungünstigsten Fall kann der Gesamtwinkelfehler auf bis zu $w_f < 0.4875°$ ansteigen. Die Anforderungen werden damit jedoch eingehalten.

6.3.3. Berechnung der ARP-Ausgabe

Die Berechnung der ARP-Ausgabe ist ähnlich der ACP-Ausgabe. Anders als in Gleichung 6.1 wird hier anstelle der Winkeldifferenz w_{min} zwischen zwei auszugebenden Pulsen der Restwinkel w_{rest} bis zur ARP-Ausgabe eingesetzt. Da der ARP-Puls nicht nur beim Nulldurchgang des Referenzwinkels generiert werden soll, sondern auch bei einem beliebigen anderen Winkel, muss eine Berechnung für zwei Winkelsektoren verschieden durchgeführt werden. Je nachdem, ob der Sweep-Winkel im Bereich zwischen Ausgabewinkel und Referenzwinkel liegt oder außerhalb dieses Bereiches. Unabhängig hiervon wird für die Berechnung keine Division benötigt, wodurch kein Rundungsfehler in w_{rest} entstehen können. Einzig der Quotient aus t_{dif}/w_{dif} kann Rundungsfehler verursachen.
Die Konfiguration des Timers und damit die Berechnung der Restzeit erfolgt erst kurz vor der eigentlichen Ausgabe. Nur so kann der richtige Ausgabezeitpunkt bestmöglichst errechnet werden, da je kleiner der Restwinkel w_{rest} ist, desto eher können Schwankungen in der Drehgeschwindigkeit ausgeschlossen werden. Zudem gilt, je kleiner w_{rest} ist, desto weniger wirken sich Rundungsfehler aus. Bei einer spezifizierten, kleinsten Pulsfolgefrequenz von 400 Hz kann mit absoluter Sicherheit davon ausgegangen werden, dass in einem Zeitfenster von 10 ms vor

der Pulsausgabe mindestens ein Berechnungszyklus stattfindet. Dies entspricht bei der kleinsten, spezifizierten Antennendrehgeschwindigkeit von 20 Umdrehungen pro Minute einem Winkel von $1.2°$. Der sich ergebende Berechnungsfehler[3] liegt so bei bis zu $218.45\,\mu s$, wenn keine Skalierung vorgenommen wird. Durch eine Skalierung mit dem Faktor 1000 reduziert sich dieser Fehler um den gleichen Faktor und liegt damit in einen akzeptierbaren Bereich. Eine Berechnung mit 32 Bit breiten Zwischenergebnissen ist bei $t_{dif} \leq 10\,ms$ ohne Überlauf möglich.

6.4. Update des FPGA-Konfigurations-Image

Im folgenden Abschnitt wird auf die Realisierung der Updatefunktion des FPGA-Konfigurations-Image über Ethernet eingegangen, sowie dessen Absicherung über eine "Fallback Multiboot"-Funktion.

6.4.1. Allgemeines

Die meisten heute im Einsatz befindlichen FPGAs sind SRAM-basiert. SRAM ist nahezu beliebig wiederbeschreibbar, gehört jedoch zu den flüchtigen Speichern. Das FPGA verliert damit seine Konfiguration bei Verlust der Versorgungsspannung, sodass eine erneute Rekonfiguration stattfinden muss. Spartan-6-FPGAs unterstützen die normale Rekonfiguration. (Dynamisch) Partielle Rekonfigurationen werden von FPGAs der Spartan-Familie nicht unterstützt. Die Konfiguration selbst kann in verschiedenen Modi stattfinden:

- Master-Mode: Das FGPA steuert für die Rekonfiguration selbst den angeschlossenen Konfigurationsspeicher an.
- Peripheral Mode: Die Konfiguration wird von der Peripherie, etwa einem Mikrocontroller übertragen.
- Slave-Mode: Es kommt zum Einsatz von Speicherbausteinen mit integrierter Logik, die ein FPGA oder auch mehrere FPGAs konfigurieren können.

Alle diese Rekonfigurations-Modi werden vom Spartan-6 unterstützt.

6.4.2. MultiBoot

Bei einer Rekonfiguration im Master-Mode besteht u. a. bei FPGAs der Spartan-6 Familie die Möglichkeit eine sog. MultiBoot-Funktionalität zu nutzen. Diese erlaubt zur Laufzeit, dass die User-Logik im FPGA eine Rekonfiguration des FPGAs auslöst. Dabei besteht die Möglichkeit eine Warm-Boot-Adresse anzugeben, die die Start-Adresse des zu ladenden Rekonfigurations-Image in einem externen Speicher angibt. Hierdurch entsteht z. B. die Möglichkeit zur Laufzeit die Konfiguration des FPGAs anzupassen, indem ein neues, anderes Rekonfigurations-Image geladen wird. Die Steuerung der MultiBoot-Funktion erfolgt über den "Internal Configuration

[3] Ein Winkel von $1.2°$ entspricht 218.45 Winkeleinheiten, wenn gilt $360° \triangleq 65536$.

Access Port" (ICAP). Zur Ansteuerung der ICAP-Schnittstelle gibt es zwei Möglichkeiten: Entweder es wird eine ICAP-Primitive [71, S. 128f.] genutzt, um eine Ansteuerung auf Hardware-Ebene mittels FSM durchzuführen, wie es in diesem Design gemacht wird, oder es wird der Xilinx "XPS-HWICAP"-Core dem MicroBlaze hinzukonfiguriert, sodass eine Ansteuerung über Software möglich ist.
Durch die Möglichkeit, eine Rekonfiguration zur Laufzeit auszulösen, kann eine Update-Funktion implementiert werden. Während des Betriebs kann so eine neue Konfiguration von einem PC an die Elektronik über Ethernet übertragen werden. Der MicroBlaze nimmt dieses Image an und speichert es im SPI-Flash ab. Über die ICAP-Schnittstelle kann dann eine Rekonfiguration des FPGAs mit dem neuen Image durchgeführt werden, wodurch das Update aktiv wird.
Dieses Vorgehen zur Durchführung eines Updates ist jedoch nicht absolut sicher gegen Fehler. Kommt es, z. B. während eines Updates zu einem Stromausfall, ist das Rekonfigurations-Image im externen Speicher fehlerhaft, sodass das FPGA bei einem Power-Up nicht mehr konfiguriert werden kann. Die Erreichbarkeit der Elektronik über Ethernet ist damit nicht mehr gegeben. Im Feldeinsatz müsste dies zu einem Austausch der Elektronik führen. Um diesen Fall abzusichern, gibt es das sog. "Fallback MultiBoot". Es ist hier neben dem eigentlichen MultiBoot-Image ein zweites Konfigurations-Image (von Xilinx als "Golden-Image" bezeichnet) im SPI-Flash abgelegt. Während eines Updates wird dieses Image nicht geändert, sodass im Falle eines fehlerhaften MultiBoot-Image auf dieses zurückgegriffen werden kann. Damit ist eine weitere Verfügbarkeit der Elektronik gewährleistet und ein erneuter Update-Versuch möglich.

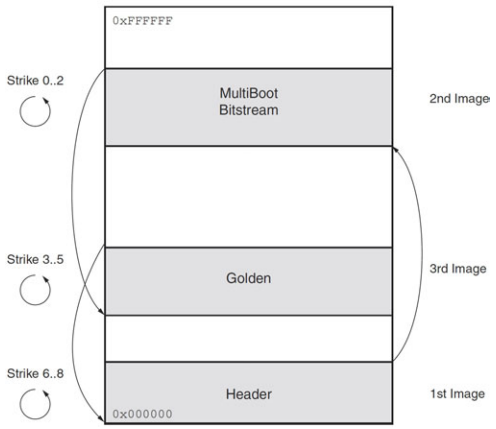


Bild 6.3.: Funktionsweise des Fallback-MultiBoot [64, S. 124]

Bild 6.3 zeigt hierfür das genaue Verfahren: Nach einem Power-Up wird immer erst ein sog. Header-Image beginnend ab Adresse 0x000000 aus dem Speicher geladen. Diese Konfiguration lädt in die vier "General Configuration"-Register des FPGAs die Startadresse des Multiboot-Image und die des Golden-Image und gibt mittels Opcodes an, ob das Laden der Images über SPI oder BPI[4] durchgeführt werden soll. Anschließend wird ein IPROG-Befehl ausgeführt, welche eine Rekonfiguration des FPGA veranlasst. Es wird dann versucht, das MultiBoot-Image zu laden. War dies dreimal nacheinander nicht erfolgreich, wird automatisch versucht, das Golden-Image zu laden. Auch hier gibt es bis zu drei Versuche. Als Fehler beim Laden der Konfiguration werden Watchdog-Timeouts, CRC-Fehler und IDCODE-Fehler erkannt. Watchdog-Timeouts entstehen etwa, wenn das SYNC-Wort am Anfang des Konfigurations-Image fehlt. CRC-Fehler entstehen bei Bit-Fehlern im Image selbst und IDCODE-Fehler treten auf, wenn das Image nicht mit dem FPGA-Typ kompatibel ist.

[4] BPI (engl. Byte Peripheral Interface) ist im Gegensatz zum SPI kein serielles, sondern ein paralleles Konfigurationsinterface über welches die Nutzung von parallelen NOR-Flashes möglich ist.

Erstellung der FPGA-Konfigurationen für das Fallback Multiboot

Die Beschreibung des Inhaltes des externen Flashspeichers erfolgt mittels einer .mcs-Datei. Diese basiert auf dem "Intel MCS-86 Hexadecimal Object"-Datei-Format, allgemein auch als Intel-Hex-Format [45] bekannt. D. h. .mcs-Dateien beinhalten neben dem eigentlichen Image Angaben zu den Speicheradressen, wo die Daten im Speicher abzulegen sind, und zur Sicherung der Daten Checksummen. .mcs-Dateien können direkt aus .bit-Dateien mittels des Tools "promgen" von Xilinx unter Angabe des gewünschten Speichermappings generiert werden.
Für die Beschreibung des gesamten Speichers muss die .mcs-Datei drei Images enthalten. Die Erstellung des ersten Image, dem Header-Image, kann auf zwei verschiedene Arten erfolgen:

- Die erste Variante ist die Erstellung eines Hex-File mittels Editor (Beispiel siehe [61]). Eine so erstellte Hex-Datei kann mittels "promgen" äquivalent zur .bit-Datei in eine .mcs-Datei gewandelt werden.
- Die zweite Variante ist die Erstellung des Header-Image mittels des Tools "bitgen". Hierfür gibt es spezielle Optionen, die beim Aufruf zur Erstellung des Golden-Image anzugeben sind, sodass das Anfügen eines Header-Image an das Golden-Image automatisch durchgeführt werden kann. Im Wesentlichen handelt es sich hierbei um die Angabe der Startadresse des Multiboot-Image und des Golden-Image. Wegen eines Fehlers im Programm "bitgen" der Version 13.2 ist den Adressen noch der Opcode anzufügen, der in diesem Fall den Zugriff auf den Speicher über den SPI-Bus angibt. Mit der nachfolgenden Version 13.3 soll dieser Fehler behoben sein [78]. Bei der Erzeugung des Bitstream des MultiBoot-Image dürfen diese Optionen dann nicht gesetzt sein.

Unabhängig davon, ob es sich um das Golden-Image zusammen mit dem Header-Image handelt oder um das Multiboot-Image, muss die Option beim Tool "bitgen" aktiviert sein, dass ein Reset beim Auftreten eines Fehlers erfolgen soll. Nur so kann ein erneuter Versuch unternommen werden, eine Konfiguration zu laden, wenn ein Fehler aufgetreten ist.
Für diese Arbeit wurden Batch-Skripte erstellt, die die Ergebnisse des "Place and Route"-Prozess nutzen, um auf Command-Line-Ebene .bit-Dateien für das Golden-Image, einschließlich Header-Image und für das MultiBoot-Image zu erstellen. Weitere Batch-Skripte ergänzen die .bit-Dateien um die Informationen aus der .elf-Datei (beinhaltet die MicroBlaze-Software) und der BlockRAM-Memory-MAP-Datei (.bmm). Nur so kann die Software des MicroBlaze ohne MicroBlaze-Debug-Modul (MDM) automatisch geladen werden. Über weitere erstellte Batch-Skripte besteht dann die Möglichkeit .mcs-Dateien zur Beschreibung des Flashs zu generieren. Neben der Erstellung eines vollständigen Image können die Skripte auch nur das Multiboot-Image genieren, wie es für ein Update benötigt würde. Alle Batch-Skripte können dem Anhang A.3.6 entnommen werden.

Entwicklung der Update-Funktion

Neben Steuerinformationen können auch Update-Informationen an den MicroBlaze über Ethernet geschickt werden. Der MicroBlaze kann dann die so empfangen Daten im externen SPI-Flash über den SPI-Bus ablegen, indem der "XPS Serial Peripheral Interface (SPI)"-Core [72] von Xilinx genutzt wird. Zur Absicherung dieser Kommunikation ist ein Handshake-Verfahren implementiert, welches den Empfang aller Nachrichten quittiert. Auf diese Weise kann die

Übertragung des Speicher-Image abgesichert und Fehler vor einer Rekonfiguration erkannt werden.
Auf PC-Seite ist ein C#-Programm entwickelt wurden, das eine beliebige .mcs-Datei einlesen und interpretieren kann. Die einzelnen Bytes des Image werden unter Berücksichtigung des in der Datei angegebenen Speichermappings in einem Byte-Array zwischengespeichert, bevor dieses über Ethernet zum MicroBlaze übertragen wird.
Damit der MicroBlaze auf den externen Flash zugreifen kann, sind Treiber-Funktionen entwickelt worden. Diese abstrahieren die Schreib-, Lese- und Löschbefehle für den auf dem Xilinx Evaluationboard SP605 eingesetzten SPI-Flash "Winbond W25Q64". Die Zugriffe erfolgen dabei über die im Datenblatt des Flash [46] spezifizierten Befehls-Sequenzen. Durch die, bezogen auf die Prozessorgeschwindigkeit, relativ geringe Schreib- und Löschgeschwindigkeit beinhalten die Funktionen Abfragen des Flash-Statusregisters. Mittels Polling dieses Statusregisters werden die einzelnen Routinen erst verlassen, wenn der Flash die Aufgabe vollständig abgearbeitet hat. Die damit einhergehende Verschwendung der Rechenzeit ist im Zuge eines Updates akzeptierbar. Eine ordnungsgemäße parallele Abarbeitung anderer Aufgaben ist hier nicht sicherzustellen. Die Lese- und Schreib-Funktionen sind jeweils auf das Lesen und Schreiben von 1024 Byte und damit 4 Pages ausgelegt. Diese Größe ist gewählt wurden, da Schreibzugriffe Page-weise erfolgen müssen. Diese sind zudem erst möglich, wenn das Flash in einen Schreib-Modus gesetzt wurde. Mit 1024 Byte ist damit auch als die Anzahl an Bytes definiert, die pro Ethernet-Nachricht dem MicroBlaze übermittelt werden. Nach erfolgtem Schreibvorgang werden die Daten über Ethernet zurückgeschickt. Dies erlaubt zum einen eine Fehlerüberprüfung, zum anderen wird die Übertragungsrate automatisch auf die Schreibgeschwindigkeit des Flashs angepasst, indem ein neues Datenpaket erst gesendet wird, wenn der MicroBlaze die Verarbeitung des letzten Datenpaketes bestätigt hat. Wird so ein Fehler erkannt, kann direkt ein erneuter Update-Versuch gestartet werden, da das FPGA noch nicht rekonfiguriert wurde. Die C#-Software erkennt automatisch, wenn keine Bestätigung von der Elektronik zurückgekommen ist. Dies kann z. B. der Fall sein, wenn es zu dem besagten Spannungsausfall kommt. Sobald das FPGA sich in diesem Fall mit dem Golden-Image rekonfiguriert hat und wieder über Ethernet erreichbar ist, setzt das C#-Tool das Update automatisch, an der jeweiligen Speicherstelle fort. Ein erneutes Starten des Updates ist nicht notwendig.
Nach erfolgreicher Durchführung des Updates ist eine Rekonfiguration des FPGAs zu veranlassen, indem ein IPROG-Befehl an die ICAP-Schnittstelle gegeben wird. Hierzu ist eine FSM nach [61, Beispiel-Projekt] im HDL-Design implementiert, die dies auf Hardware-Ebene unter Verwendung des ICAP-Primitive durchführt. Gestartet wird die ICAP-Ansteuerung, indem ein Bit in einem der Konfigurations-Register des Konfigurations-Interface durch den MicroBlaze gesetzt wird. Der MicroBlaze selbst setzt dieses Bit bei Empfang einer entsprechenden Ethernet-Nachricht, die am Ende eines Updates durch das C#-Programm gesendet werden kann. Um sicherstellen zu können, dass das Update erfolgreich war und nicht das Golden-Image geladen werden musste, gibt es zwei Möglichkeiten:

- Die erste und aufwendigste Möglichkeit ist das Auslesen des "Boot History Status Register" (BOOTSTS) über die ICAP-Schnittstelle. Dieses Register beinhaltet Informationen u. a. darüber, ob ein "Fallback" stattgefunden hat und welcher Fehler vorlag.
- Die zweite Möglichkeit ist, über einen Befehl per Ethernet den Revisions-Stand des Designs abzufragen. Auf diese Weise ließe sich ein Golden-Image kenntlich machen.

Letztere Variante ist in diesem Fall zweckmäßiger. Die Abfrage des Revisions-Standes ist auch im Zuge der Durchführung eines Updates von Interesse.
Kommt es zu einem Rückgriff auf das Golden-Image, besteht nicht mehr die Möglichkeit das Multiboot-Image über die ICAP-Schnittstelle neu zu laden, da das "Boot History Status Register" nicht durch User-Logik gelöscht werden kann und dadurch das FPGA weiterhin von einem defekten Image ausgeht [64]. Dies ist aber gerade gewünscht, wenn ein erneutes Update durchgeführt wurde und das Multiboot-Image als fehlerfrei angenommen werden kann. Nach [64] besteht die einzige Möglichkeit das BOOTSTS-Register zu löschen darin, den "PROGRAM_B" Pin zu toggeln oder aber die Versorgungsspannung des FPGAs zu unterbrechen. Letzteres ist im Feldeinsatz ungeeignet, da ein Zugang zur Elektronik notwendig wird. Der "PROGRAM_B" Pin ist ein FPGA-Pin, welcher nicht durch User-Logik direkt angesteuert werden kann. Bei einer möglichen Produktentwicklung kann dieses Problem umgangen werden, indem ein anderer IO-Pin auf den "PROGRAM_B"-Pin durch externe Beschaltung zurückgeführt wird. Diese Verbindung ist über einen Pull-Up-Widerstand gegen 2.5 V zu ziehen. Der verwendete IO-Pin ist dann mit dem Three-State-Ausgangs-Buffers OBUFT zu treiben. Im Normalzustand wird dieser hochohmig geschaltet. Über ein durch den MicroBlaze ansteuerbares Konfigurations-Register kann der Buffer den Ausgang gegen Masse ziehen und damit ein Reset auslösen. Bei dieser Art von Reset wird die Multiboot-History gelöscht, sodass das Multiboot-Image wieder geladen werden kann. Beim SP605-Evaluationboard und damit beim Prototypen, ist dieser Schritt manuell durch Betätigen eines hierfür vorgesehenen Schalters durchzuführen.

KAPITEL 7

Test und Verifikation

Während bereits in Kapitel 5.1.4 auf die Entwicklung der Unit-Tests zur testgetriebenen Entwicklung des Hardware-Designs eingegangen wurde, erfolgt in diesem Kapitel die Durchführung von System-Tests und die Analyse deren Ergebnisse. Auch wenn dieses Kapitel nach der Implementierung angeordnet ist, sind die System-Tests bereits vorher definiert gewesen: Die Elektronik ist unter dem Gesichtspunkt entwickelt, in einem bereits bestehenden Navigations-System eingesetzt zu werden. Daraus ergeben sich auf der Seite der Ethernet-Schnittstelle, wie auch auf der Seite des Analog-Interfaces ein exakt einzuhaltendes, spezifiziertes Schnittstellen-Verhalten. Überprüft werden kann dieses Verhalten zum einen, indem der Prototyp als Bestandteil dieses realen Systems eingesetzt und das Verhalten analysiert wird, zum anderen können hierüber hinaus spezielle Tests entwickelt werden, um einzelne Eigenschaften genauer untersuchen zu können.
Um diese speziellen Systemtests durchführen zu können, wird im ersten Teil des Kapitels, in Abschnitt 7.1, auf die Entwicklung einer PC-basierten Testsoftware eingegangen. Daraufhin wird in Abschnitt 7.2 die Signalqualität der erzeugten Radar-Pulse untersucht und vor dem Hintergrund der Anforderungen bewertet. Auf Basis dieser Erkenntnisse und weiteren Messungen findet die Dimensionierung des Video-Buffers in Abschnitt 7.3 statt. Abschnitt 7.4 untersucht die Leistungsfähigkeit der HW-SW-Schnittstelle, bevor in Abschnitt 7.5 eine Bewertung der Ressourcenverbräuche erfolgt und das realisierte Ethernet-Interface mit denen aus der Voruntersuchung verglichen wird.

7.1. Testsoftware

Um die Elektronik in ihrem Verhalten im Ziel-System zu überprüfen und verschiedene Tests durchführen zu können, ist eine Testsoftware[1] in der Programmiersprache C# entwickelt worden. Diese Software umfasst im Wesentlichen folgende Funktionen:

Geschwindigkeitstest: Die Iperf-Tools erweisen sich für Tests zur Analyse bei fehlerhaften Datenübertragungen als ungeeignet. Der hier implementierte Test ist kompatibel zum Testverfahren nach Iperf, kann jedoch zusätzlich ausgeben, welcher Iperf-Identifier die Nachricht aufweist, bei der eine fehlerhafte oder fehlende Nachricht festgestellt wurde. Mit dieser Information kann eine Analyse der Fehlerursache auf Basis des Netzwerk-Analysators Wireshark [47] durchgeführt werden, wenn dieser parallel die Daten aufgezeichnet hat. Der Test ist in der Lage, mit verschiedenen Übertragungsgeschwindig-

[1] Die Entwicklung fand unter Verwendung der Entwicklungs-Software "Microsoft Visual Studio 2010 Premium" statt.

keiten Ethernet-Pakete zu senden bzw. diese zu empfangen. Über eine integrierte Regelung besteht die Möglichkeit, die Übertragungsgeschwindigkeit vorzugeben, sodass diese automatisiert mittels IST-SOLL-Vergleich eingeregelt wird. Durch eine Diagramm-Darstellung kann die Anzahl der Fehler parallel zur Übertragungsgeschwindigkeit und PC-Prozessorauslastung grafisch aufbereitet dargestellt werden.

Radar: Über die Test-Software können radarspezifische Steuerungen durchgeführt und die Verarbeitung der Radar-Signale überwacht werden. Zu möglichen Steuerungen zählt die Auswahl des zu verarbeitenden Radar-Transceivers, die Auswahl ob der ARP-Puls als Head- oder Northmarker generiert werden soll, die Konfiguration der Pulsbreiten u. v. m.. Überwacht werden kann hingegen z. B. ob Radar-Sweeps verloren gehen oder die Pulsausgabezeitpunkte korrekt berechnet werden.

Konfiguration: Ebenfalls implementiert ist die Möglichkeit, Lese- und Schreib-Zugriffe auf alle Register des Konfigurations-Interface des HDL-Designs durchzuführen und anzeigen zu lassen. Auf diese Weise lassen sich neben Veränderungen der Konfiguration des Ethernet-Interfaces (IP-Adressen, MAC-Adressen, Port-Nummern), u. a. auch Funktionen überprüfen wie das An- und Abmelden an Multicast-Gruppen oder das Senden von ARP-Requests an andere Teilnehmer, mit anschließendem Auslesen der daraufhin empfangen MAC-Adresse.

Wireshark: Implementiert sind Funktionen, die aus Wireshark exportierten Netzwerkverkehr eines Radar-Transceivers analysieren. Auf diese Weise konnte automatisiert die im proprietären Protokoll enthaltenen Zeitstempel extrahiert werden, um diese für weitere Analysen auswerten zu können (siehe hierzu Abschnitt 7.3).

FPGA-Updater: Siehe hierzu Abschnitt 6.4.2

Radar-Simulator: Der Radar-Simulator ist zunächst als Zusatzfunktion der Testsoftware entwickelt worden. In einem zweiten Schritt fand zusätzlich eine Implementierung des Simulators ohne Verwendung einer GUI auf Command-Line-Ebene statt.
Unter Einhaltung des proprietären Video-Protokolls ist der Simulator in der Lage, Radar-Video-Pakete an eine Multicast-Gruppe zu verschicken. Die hierfür notwendigen Informationen, wie die Sample-Werte, die Winkelinformationen und die notwendigen Zeitstempel werden synthetisch erzeugt. Mithilfe dieser Informationen kann die entwickelte Elektronik Video-Sweeps und Synchronisationspulse generieren, sodass ein Sichtgerät die Reflexionen darstellen kann.
Grundlage für das periodische Senden der Video-Pakete stellt die Timer-Komponente des .NET-Frameworks als Zeitgeber dar. Die Anforderung an diesen Zeitgeber ist es, äquidistante Elapsed-Ereignisse mit einer Periode von 2 ms in der Anwendung auszulösen, die einen Sendevorgang eines vollständigen Sweeps über Ethernet einleiten.
Im Vergleich zu einem echten Radar-Transceiver weist der Simulator, aufgrund nicht ausreichender Echtzeitfähigkeit, eine deutlich höhere Streuung in der Sendeperiode auf. Um die hohe Streuung der Empfangsperiode auf der Seite der Elektronik verarbeiten zu können, ist der Video-Buffer aus Abschnitt 5.6.1 für die Tests mit dem Radar-Simulator größer dimensioniert worden, als es für echte Radar-Transceiver notwendig wäre, wie Abschnitt 7.3 zeigen wird. Die Ausgabeperiode kann so auch hier zeitlich von der Empfangsperiode entkoppelt werden, ohne dass es zu Über-/Unterläufen des Buffers kommt. Die Größe des Buffers ist mit 32 zu speichernden Sweeps empirisch bestimmt worden.

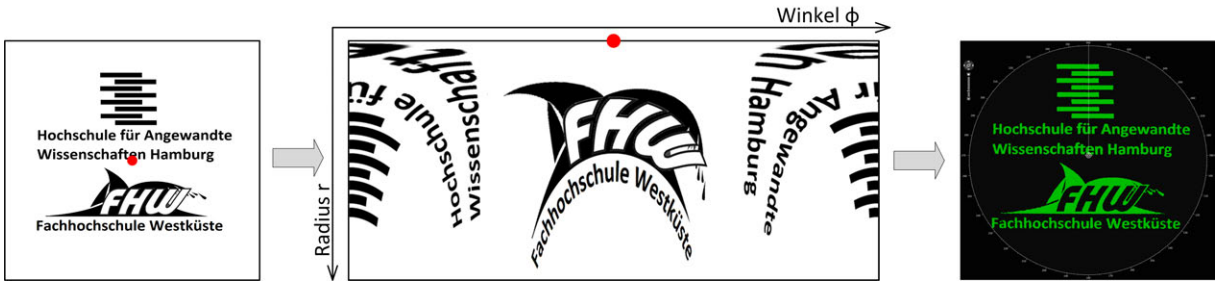


Bild 7.1.: Radarbilderzeugung mittels Simulator: Eine Bild-Datei (links) wird in Polarkoordinaten umgerechnet (mitte) und über Ethernet an den Prototypen geschickt. Ein am Prototypen angeschlossenes Sichtgerät liefert so fehlerfrei ein Radar-Bild (rechts)

Als Quelle für die zu verschickenden Sample-Werte der Echos sind zwei Möglichkeiten implementiert. Bei der ersten Möglichkeit kann ein Ring mit variablem Radius und variabler Breite als Echo an die Elektronik versendet werden. Die zweite Möglichkeit ist das Einlesen einer Bitmap-Datei. Nachdem die Bilddatei einer Koordinatentransformation in Polarkoordinaten unterzogen wurde, können schwarze Farben im Bild als Reflexionen über das Ethernet verschickt werden, weiße Farben stellen hingegen keine Reflexion dar. Dem Bild 7.1 kann dies für ein Testmuster entnommen werden. Links im Bild das Ausgangsbild, in der Mitte das transformierte Bild und rechts das Radar-Bild wie es durch die entwickelte Elektronik erzeugt und vom Sichtgerät dargestellt wird.
Mittels des Simulators ließen sich folgende Untersuchungen des Systems vornehmen:

- Zu Beginn konnten einzelne Video-Sweeps auf Benutzereingabe hin übertragen werden, sodass die korrekte Interpretation von Video-Informationen und die korrekte Funktion der Buffer auf dem FPGA verifiziert werden konnte.
- Die von den zur Verfügung stehenden echten Radar-Transceivern empfangenen Reflexionen sind aufgrund des Standortes der Anlagen und der damit einhergehenden Vielzahl an Echos sehr ausgeprägt, sodass eine qualitative Beurteilung der DAC-Ausgabe nur bedingt möglich ist. Eine zusätzliche Beurteilung eines synthetischen Videos, mit exakt definierten Reflexionen, ist daher hilfreich. Bild 5.19 zeigt ein Beispiel für ein so erkennbares Timing-Problem bei der DAC-Ansteuerung.
- Durch den Simulator ließen sich von echten Radar-Transceivern während der Entwicklung nicht unterstützte Protokoll-Versionen testen.

Im Anhang A.3.1 befindet sich eine detailliertere Erläuterung über die verschiedenen Funktionen des Test-Programmes. Im Anhang A.3.4 sind zudem die Sourcen und Setup-Dateien des Test-Programmes hinterlegt.

7.2. Untersuchung der Radar-Pulse

Im folgenden wird die Berechnung und Generierung der Radar-Pulse untersucht. In Abschnitt 7.2.1 werden die Abarbeitungszeiten der hierfür notwendigen Interrupt-Handler und die sich daraus ergebende Prozessorauslastung ermittelt. In Abschnitt 7.2.2 wird daraufhin der zeitliche Jitter dieser Pulse auf die Anforderungen hin untersucht.

7.2.1. Prozessorauslastung

Um eine Abschätzung der Prozessorauslastung zu erreichen, ist ein Profiling für die Interrupt-Routinen durchzuführen. Von Interesse sind hier die pulserzeugenden Routinen, weniger die Routine zur Ethernet-Verarbeitung, da diese im Normal-Betrieb nur sporadisch Steuerinformationen mit geringem Datendurchsatz verarbeitet. Für die Durchführung eines Code-Profiling ist die Interrupt-basierte Verarbeitung hinderlich. Das Software-Development-Kit von Xilinx bietet die Möglichkeit, ein Code-Profiling durchzuführen. Hierzu ist ein Timer zu konfigurieren, der mittels periodischer Interrupts den aktuellen Wert des Programm-Zählers[2] speichert, sodass später eine Analyse mittels des GNU-Profiler "gproof" durchgeführt werden kann[3]. Da Interrupts nicht unterbrochen werden können, kann auf diese Weise ein Profiling von Interrupt-Routinen nicht durchgeführt werden. Um nun doch ein Profiling zu erlauben, wird ein freilaufender Timer konfiguriert. Auf diese Weise kann z. B. am Anfang und Ende einer Interrupt-Routine ein Zeitstempel in einem Array abgelegt werden. Mittels des MicroBlaze-Debug-Moduls (MDM) lassen sich Array-Inhalte in eine Tabellenkalkulation übertragen, sodass eine Auswertung erfolgen kann. Da hier absolute Laufzeiten gemessen werden sollen, muss die benötigte Rechenzeit zur Speicherung der Timerwerte abgezogen werden.
Die Messungen ergaben für die Abarbeitung der Trigger-Handler eine Rechenzeit von $18.66\,\mu s$, für den ACP-Handler $11.93\,\mu s$ und für den ARP-Handler $2.31\,\mu s$. Damit ergibt sich bei einer maximal nach Abschnitt 1.1 zu verarbeitenden Pulsfolgefrequenz von 1500 Hz und 8192 zu erzeugenden ACP-Pulsen pro Umlauf bei 80 Umdrehungen pro Minute eine Rechenauslastung von $15.9\,\%$ bezogen auf einen Systemtakt von 66.666 MHz. Interruptlatenzen sind nach [69] im Worst-Case-Fall mit 6 Takten angegeben[4]. Bei einer Interruptrate von maximal 12.4 kHz ergibt sich dadurch lediglich eine Auslastung von etwa $0.11\,\%$.

7.2.2. Jittern der Radar-Pulse

Durch Jittern der Radar-Pulse kommt es zu Winkelfehlern, da der Trigger-Puls und damit die empfangenen Echos einem falschen Winkel zugeordnet werden. Zur Bestimmung des Jitters werden die Zeiten eines freilaufenden Timers beim tatsächlichen Aufruf des Interrupt-Handlers abgespeichert und ausgewertet. Die Messungen müssen bei der Verarbeitung des synthetischen Videos stattfinden. Damit kann es zu keiner temporären Veränderung der Periodendauer kommen, die nicht durch die Verarbeitung der Elektronik bedingt ist. Nur ungewollte Abweichungen werden so sichtbar.

ACP-Puls:

Bild 7.2 gibt die Wahrscheinlichkeit verschiedener Periodendauern beim ACP-Puls an. Der Soll-Wert[5] liegt bei $4915.2\,\mu s$ (im Bild 7.2 rot eingezeichnet) und wird damit sehr gut erreicht. Das arithmetische Mittel des Ist-Wertes liegt dabei 55 ns (3.67 Takte) über dem

[2] Der Programm-Zähler (engl. program counter, PC) speichert die Speicheradresse des auszuführenden Befehls ab.

[3] Dem Anhang A.3.5 ist ein erstelltes Tutorial zu entnehmen, welches die notwendigen Konfigurationen beschreibt.

[4] Eine Unterbrechung einer in Hardware durchgeführten Division, die eine Latenz von bis zu 28 Takten zur Folge haben kann, kann ausgeschlossen werden, da keine Divisionen in der main-while-Schleife stattfinden.

[5] Ergibt sich aus dem synthetischen Video des Simulators mit einer festen Winkeldifferenz von 54 Einheiten, einer Periode von 2025 μs und 500 zu generierenden ACP-Pulsen.

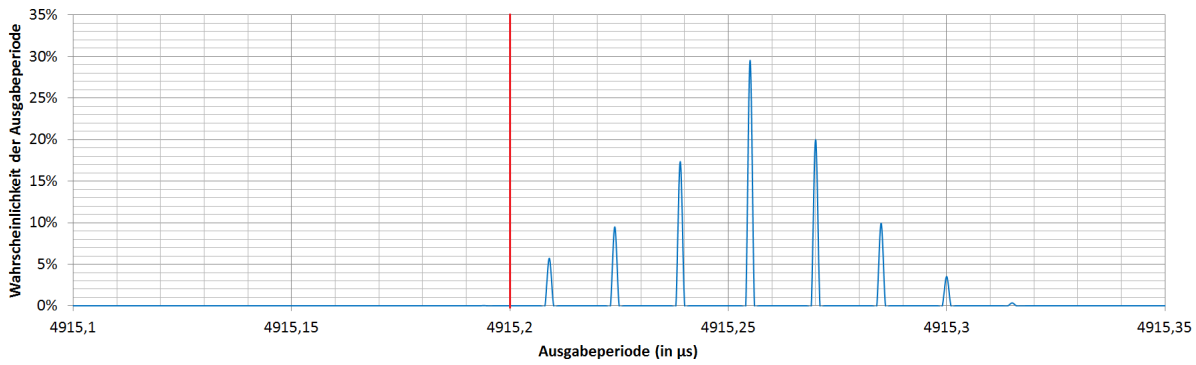


Bild 7.2.: Wahrscheinlichkeitsverteilung der Ausgabeperiode des Azimuth-Clock-Puls bei 500 Pulsen pro Umdrehung (in rot das arithmetische Mittel)

Soll-Wert und damit nach Abschnitt 6.3.2 im Bereich der Toleranz aufgrund von Rundungsfehlern. Die im Bild zu sehende Streuung lässt sich auch beim Interrupt-Handler des Trigger-Timers feststellen, wo keine Rundungsfehler auftreten können. Der Abstand zwischen den Peaks ist dabei gleich der Systemtaktperiode. Die Ursache liegt darin, dass die Interrupt-Latenz nach [69] bis zu 6 Takte betragen kann. Die tatsächliche Verzögerung hängt jedoch davon ab, welcher Befehl sich zum Zeitpunkt des Interrupt-Request in der Ausführungsphase der Befehlspipeline des MicroBlazes befindet.
Die in Bild 7.2 dargestellten Messwerte decken lediglich 96.1 % der aufgenommen Werte ab. In 3.9 % der Fälle überlappt sich die Bearbeitungszeit mit der Bearbeitung einer anderen Interrupt-Routine. Die festgestellten Abweichungen liegen im Bereich der Abarbeitungszeit, die Trigger- und ARP-Interrupthandler in Summe benötigen. Dies liegt darin begründet, dass der ARP-Interrupt-Handler die höchste Priorität besitzt. Wird während eines ACP-Interrupt-Requests der am niedrigsten priorisierteste Trigger-Interrupthandler ausgeführt, so wird dieser und der ARP-Handler ausgeführt, bevor es zu einer Abarbeitung des ACP-Handlers kommt. Die Streuung kann damit bei bis zu $\pm 20.97\,\mu s$ liegen.

ARP-Puls:
Die Untersuchung des ARP muss, aufgrund seiner Wiederholrate, über eine deutlich längere Messzeit erfasst werden. Es kann auch hier festgestellt werden, dass es durch die nicht unterbrechbaren Interrupts zum Jittern des Ausgabezeitpunktes um das arithmetische Mittel kommt. Eine Aussage über die Genauigkeit des arithmetischen Mittels kann nicht gemacht werden. Die Ausgabeperiode liegt $310\,\mu s$ über dem erwarteten Wert. Da die Periode konstant ist, kann die Ursache nur in der Erzeugung des synthetischen Videos im Simulator liegen. Das Jittern des ARP-Pulses liegt maximal bei der Ausführzeit des Trigger-Handlers. Dies ist bedingt durch dessen höchste Priorität.

Trigger-Puls:
Am Beispiel des ACP- und ARP-Pulses konnte gezeigt werden, dass es durch nicht unterbrechbaren Interrupt-Handler zu Auswirkungen auf die jeweilige Periodendauer kommt. Beim Trigger konnten diese Auswirkungen ebenfalls festgestellt werden, wenn auch nicht so deutlich. Dies liegt darin begründet, dass die Trigger-Frequenz bei den Messungen um das 2.5-fach höher lag, als die ACP-Frequenz und die Ausführzeit des ACP- und ARP-Pulses geringer ist als die des Trigger-Handlers. Die Häufigkeit einer Überlappung ist damit aufgrund der Wahrscheinlichkeit geringer, als in den zuvor durchgeführten Messungen für den ACP- und ARP-Puls. Die maximale möglichen Schwankungen setzen

sich aus der Summe der Ausführzeit der ACP- und ARP-Interrupt-Routine zusammen, welche höher priorisiert sind, als der Trigger-Interrupt. Es kann damit ein Jitter von bis zu $\pm 14.24\,\mu s$ entstehen.

Da der ARP-Puls als Referenz für einen vollständigen Umlauf gilt und zusätzlich der ACP-Puls für die Bestimmung des Winkels beim Auftreten des Trigger-Pulses verwendet wird, setzt sich der Gesamtfehler aus der Summe der einzelnen Zeitfehlern zusammen. Der entstehende Zeitfehler kann damit bei bis zu $\pm 20.97\,\mu s$ (Summe der Abarbeitungszeiten der Trigger- und ARP-Interrupt-Handler) liegen. Die Grenze von $\pm 45.78\,\mu s$ aus Abschnitt 4.2.1 wird damit eingehalten.

7.3. Untersuchung der Pufferung digitaler Radarsignale

Der Video-Buffer aus Abschnitt 5.6.1 dient dazu, eine zeitliche Entkopplung zwischen Ausgabe- und Empfangsperiode von Radar-Sweeps durchzuführen, damit die zu rekonstruierende Trigger-Periode des Radar-Transceivers nicht von der sich ergebenden Streuung der Empfangsperiode der Ethernet-Pakete abhängt. Zur geeigneten Dimensionierung des Ring-Buffers muss daher nicht nur bekannt sein, wie groß ein einzelner Radar-Sweep ist, sondern auch wie viele Radar-Sweeps selbst zu puffern sind, damit die zeitliche Entkoppelung und damit fehlerfreie Rekonstruktion des ursprünglichen Echtzeitverhalten möglich ist. Die Anzahl an Speicherplätzen ist so zu dimensionieren, dass ein Über- oder Unterlauf des Buffers aufgrund der Wahrscheinlichkeit ausgeschlossen werden kann. Um hierüber eine Aussage treffen zu können, muss eine stochastische Betrachtung des Befüllungs- und Entleerungs-Prozesses des Buffers stattfinden. Vor dem Hintergrund, dass das arithmetische Mittel der Befüllungs- und Entleerungsperiode identisch ist, kann ein Über- oder Unterlauf nur durch die Streuung der Perioden entstehen. Es können folgende Einflussfaktoren auf den Füllstand des Buffers festgestellt werden:

- Durch die Übertragung der Video-Sweeps über Ethernet passieren die Daten verschiedene Buffer, etwa im Radar-Transceiver selbst oder in den Ethernet-Switches. Eine Zwischenspeicherung hat damit zur Folge, dass die Daten temporär schneller oder langsamer von dem Prototypen empfangen werden. Eine zusätzliche Streuung dieser Empfangsperiode durch die Verarbeitung im hier implementierten Design kann ausgeschlossen werden, da keine Zwischenspeicherung vor dem Video-Buffer stattfindet.
- Die Entnahme der Video-Daten aus dem Buffer wird durch den MicroBlaze getriggert. Durch den in Abschnitt 7.2.2 nachgewiesenen Jitter des Trigger-Pulses kommt es zu einer ungewollten Streuung der Entnahmeperiode. Eine Korrelation der ungewollten Streuung der Entnahmeperiode mit der Befüllungsperiode ist aufgrund der Verarbeitung der Ethernet-Pakete in Hardware auszuschließen. Die ungewollte Streuung der Entnahmeperiode ist nicht zu verwechseln mit der zu reproduzierenden Streuung der Trigger-Periode des Radar-Transceivers, um das ursprüngliche Zeitverhalten zu rekonstruieren.
- Es ist anzunehmen, dass die Streuung der zu rekonstruierenden Trigger-Periode des Radar-Transceivers mit der Streuung der Empfangsperiode von Ethernet-Paketen zum Teil korreliert. Je größer die Korrelation ist, desto kleiner kann der Buffer aufgrund der Rekonstruktion dieser Periode dimensioniert werden.

7.3.1. Stochastische Beschreibung der Datenverarbeitung

Um eine stochastische Beschreibung durchführen zu können, sind verschiedene Messungen notwendig:

Empfangsperiode/Befüllungsperiode:
Zur Untersuchung der Streuung einer Empfangsperiode von Ethernet-Paketen und damit der Befüllungsperiode des Buffers wird der Netzwerkverkehr eines realen Systems mittels dem Netzwerk-Analysator Wireshark [47] aufgezeichnet. Dieser Analysator gibt dabei zu jeder Nachricht Informationen über den Empfangszeitpunkt[6] an. Um die Wahrscheinlichkeitsdichte der Empfangsperioden zu bestimmen, können die Zeitstempel durch die Export-Funktion von Wireshark exportiert werden. Unter Verwendung von insgesamt 6000 Messwerten wird die Häufigkeit der Zeitabstände zwischen den Sweeps ermittelt und dadurch empirisch die Wahrscheinlichkeitsdichte berechnet. Die sich so ergebene

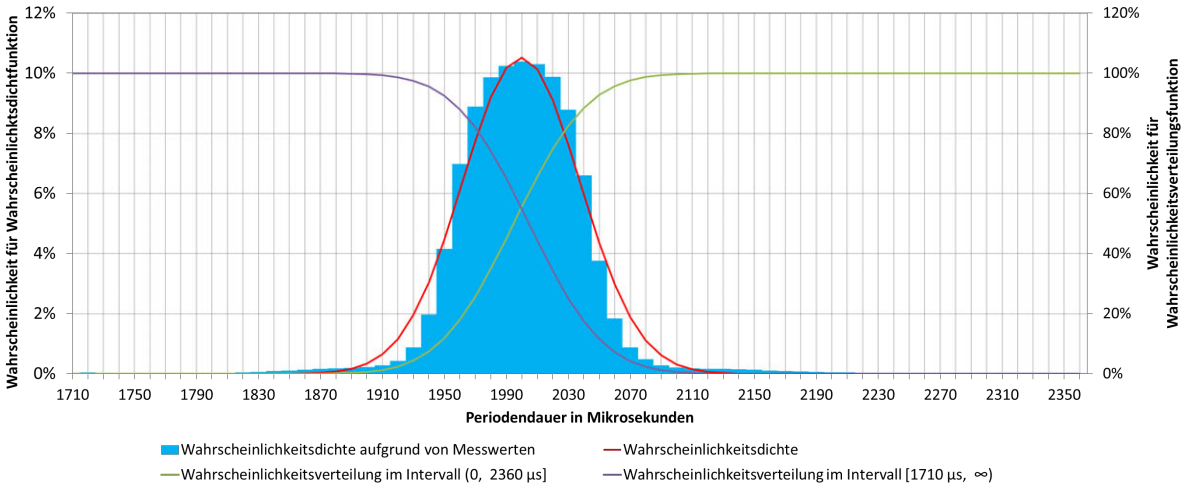


Bild 7.3.: Wahrscheinlichkeitsdichte der Empfangsperiode von Ethernet-Video-Daten (blau), ermittelte mathematische Beschreibung der Wahrscheinlichkeitsdichte (rot), Wahrscheinlichkeitsverteilungsfunktionen (grün, violett)

Wahrscheinlichkeitsdichte (blau in Bild 7.3) kann als Normal-Verteilung betrachtet werden. Durch die Maximum-Likelihood-Schätzung [42] lässt sich damit für eine Messreihe nach Gleichung 7.1[7] die Standardabweichung mit $\sigma_W = 37.924\,\mu s$ in der Periodendauer schätzen. Im konkreten Fall schwankt die Empfangsfrequenz damit in $68.3\,\%$ der Fälle (einfache Standardabweichung) um maximal $\pm 9.5\,Hz$ bei einem arithmetischen Mittel von $500\,Hz$.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{7.1}$$

Unter Kenntnis der Standardabweichung und des arithmetischen Mittels kann nach Gleichung 7.2 [29] die Wahrscheinlichkeitsdichtefunktion beschrieben werden (rot in Bild

[6] Nach [48] wird dieser Zeitstempel vom Betriebssystem generiert, sodass es prinzipiell zu einer Korrelation der so ermittelten Streuung der Empfangsperiode und z. B. der Prozessorauslastung des Test-PC kommen kann. Stochastisch betrachtet kann die Varianz dadurch nur zunehmen und nicht abnehmen, sodass die Messung das Ergebnis höchstens dahin gehend beeinflusst, dass die benötigte Buffergröße zunimmt.

[7] In Gleichung 7.1 ist n die Anzahl der Messwert, $\bar{x}$ der arithmetische Mittelwert und x_i der i-te Messwert.

7.3). Δt_i ist dabei die i-te Periodendauer zwischen zwei Zeitstempeln und $\overline{\Delta t}$ der sich aus allen Periodendauern ergebende arithmetische Mittelwert.

$$F_i = \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot e^{-\frac{(\Delta t_i - \overline{\Delta t})^2}{2 \cdot \sigma^2}} \tag{7.2}$$

Es zeigt sich eine gute Übereinstimmung der Wahrscheinlichkeitsdichte, die sich auf Basis der geschätzten Standardabweichung ergibt.

Sendeperiode

Die zu rekonstruierende Streuung der ursprünglichen Sendeperiode des Radar-Transceivers in der Ausgabeperiode ist ebenfalls bei einer stochastischen Beschreibung zu berücksichtigen. Weniger, weil sie zu einem Über-/Unterlauf führen könnte, sondern vielmehr weil die Möglichkeit besteht, dass die Periode der Zeitstempel, mit der elektromagnetische Pulse ursprünglich gesendet wurden, mit der Empfangsperiode der Ethernet-Pakete, d. h. mit der Befüllungsperiode, korreliert. Da die Elektronik diese Streuung bei der Entnahme aus dem Buffer rekonstruiert, mindert dies wahrscheinlichkeitstheoretisch die Auswirkungen der Streuung der Empfangsperiode auf den Füllstand des Buffers.

Für eine Auswertung der Zeitstempel im proprietären Video-Protokoll ist eine Funktionserweiterung des in Abschnitt 7.1 entwickelten C#-Programms vorgenommen worden, mit dem die Zeitstempel aus einer aus Wireshark exportierten Datei extrahiert und so aufbereitet werden können, dass eine Auswertung möglich wird. Nach Gleichung 7.1 ergibt sich für die durch die Zeitstempel gegebene Periode eine Standardabweichung von $\sigma_P = 7.863\,\mu s$. Da diese Daten auf der gleichen Messung beruhen wie für die zuvor ermittelten Empfangsperioden, ist eine Ermittlung der Korrelation zwischen beiden Perioden zulässig. Gleichung 7.3 erlaubt hierfür die Bestimmung des empirischen Korrelationskoeffizienten nach Bravis und Pearson [6], welche die beste Schätzung des wahren Korrelationskoeffizienten auf Basis von Messwerten zulassen soll. Dabei stellt x_i die i-te Empfangsperiode dar und y_i die i-te Periode, die durch die Zeitstempel gegeben ist. Entsprechend sind $\overline{x}$ und $\overline{y}$ die jeweiligen arithmetischen Mittelwerte.

$$r_{xy} = \frac{Cov(x,y)}{\sqrt{Var(x)} \cdot \sqrt{Var(y)}} = \frac{\sum_{i=1}^{n} \left((x_i - \overline{x}) \cdot (y_i - \overline{y})\right)}{\sqrt{\sum_{i=1}^{n} (x_i - \overline{x})^2 \cdot \sum_{i=1}^{n} (y_i - \overline{y})^2}} \quad mit -1 \leq r_{xy} \leq 1 \tag{7.3}$$

Nach Gleichung 7.3 ergibt sich damit ein Korrelationskoeffizient von $r_{xy} = 0.20011$. Während ein Korrelationskoeffizient bei 1 bzw. -1 einen linearen Zusammenhang zwischen beiden Perioden darstellen würden, ergibt ein Korrelationskoeffizient von 0 keinen linearen Zusammenhang. In diesem Fall liegt damit ein schwacher, gleichgerichteter Zusammenhang zwischen der Empfangsperiode und der zu rekonstruierenden Trigger-Periode vor. Nach dem Zähler von Gleichung 7.3 liegt die Kovarianz dieser beiden Perioden bei $Cov(x,y) = 231.028 \cdot 10^{-15}$.

Ausgabeperiode/Entnahmeperiode:

Neben der zu rekonstruierenden Streuung kommt es bei der Ausgabe zu einer weiteren Streuung. Es ergab sich in den Untersuchungen in Abschnitt 7.2.2 näherungsweise eine Normalverteilung, wenn die Auswirkungen der Systemtaktperiode nicht berücksichtigt werden. Die ermittelte Standardabweichung liegt hier bei $\sigma_T = 480.65\,ns$. Im Vergleich

zu den beiden anderen Standardabweichungen ($\sigma_W = 37.924\,\mu s$ und $\sigma_P = 7.863\,\mu s$) ist diese sehr gering. Wie erwähnt, kann angenommen werden, dass diese Streuung nicht mit der Empfangsperiode korreliert.

Zur Beschreibung des Gesamtsystems wird die Methodik des Gauß'schen Fehlerfortpflanzungsgesetzes angewendet. Die Varianz einer Funktion $f(x_i, y_i, z_i)$ dreier Periodendauern x_i, y_i und z_i kann durch Gleichung 7.6 [37] bestimmt werden, vorausgesetzt $f(x_i, y_i, z_i)$ ist linear oder kann um $f(\overline{x}, \overline{y}, \overline{z})$ linear angenähert werden. In diesem Fall wird $f(x_i, y_i, z_i) = (\overline{z} - z_i) + (y_i - x_i)$ gesetzt, mit x_i als die i-te Füll-Periode, y_i die i-te zu rekonstruierende Trigger-Periode und z_i der i-te Entnahmeperiode mit ungewollter Streuung. Damit beschreibt $f(x_i, y_i, z_i)$ die Abweichungen vom idealen Mittelwert und damit die Größe, die entscheidend für einen Über- oder Unterlauf des Buffers ist. Berücksichtigt wird in den folgenden Gleichungen bereits die nachgewiesene Korrelation zwischen der Perioden x_i und der Periode y_i.

$$var_{ges} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\delta f}{\delta x}(x_i - \overline{x}) + \frac{\delta f}{\delta y}(y_i - \overline{y}) + \frac{\delta f}{\delta z}(z_i - \overline{z})\right)^2 \tag{7.4}$$

$$\overset{N:=3}{=} \left(\frac{\delta f}{\delta x}\right)^2 \cdot \sigma_x^2 + \left(\frac{\delta f}{\delta y}\right)^2 \cdot \sigma_y^2 + 2 \cdot \frac{\delta f}{\delta x} \cdot \frac{\delta f}{\delta y} \cdot cov(x,y) + \left(\frac{\delta f}{\delta z}\right)^2 \cdot \sigma_z^2 \tag{7.5}$$

$$= \sigma_x^2 + \sigma_y^2 - 2 \cdot cov(x,y) + \sigma_z^2 \tag{7.6}$$

Es folgt eine Varianz von $var_{ges} = 1.381 \cdot 10^{-9}$ und damit eine Standardabweichung von $\sigma_{ges} = 37.156\,\mu s$. Bedingt durch die bestimmte, schwach gleichgerichtete Korrelation der zu rekonstruierenden Streuung mit der Empfangsperiode liegt die, für den Füllstand entscheidende, Standardabweichung minimal niedriger als die Standardabweichung der Empfangsperiode σ_W und deutlich niedriger als die Standardabweichung, die sich ohne Korrelation ergeben würde.

7.3.2. Beweis für die maximale Über-/Unterlauf-Wahrscheinlichkeit des Buffers

Ziel ist nicht, die Größe des Buffers selbst zu bestimmen, um eine bestimmte Wahrscheinlichkeit eines Über- oder Unterlaufs zu erhalten, sondern einen Beweis darüber zu führen, wie groß die größte Wahrscheinlichkeit für einen Über- oder Unterlauf bei verschiedenen Füllständen ist. In Bild 7.3 ist hierzu ebenfalls die Verteilungsfunktion für das Intervall $[0\,\mu s, 2360\,\mu s]$ und das Intervall $[1710\,\mu s, \infty)$ dargestellt, wovon das erstere Intervall durch die Gleichung 7.7 beschrieben werden kann. Das zweite Intervall hat aufgrund der Normalverteilung einen zum Mittelwert spiegelsymmetrischen Verlauf.

$$S_n = \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot \int_0^n e^{-\frac{(\Delta t_n - \overline{\Delta t})^2}{2 \cdot \sigma^2}} d\,t \tag{7.7}$$

S_n stellt damit die Wahrscheinlichkeit für eine Empfangsperiode dar, dass diese die Periodendauer Δt_n oder aber eine Periodendauer mit größerer Streuung aufweist. Dabei ist $\overline{\Delta t_n}$ das arithmetische Mittel.
Neben dieser Wahrscheinlichkeit ist von Bedeutung, wie viele Ereignisse mit dieser gleichen oder größeren Streuung maximal eintreffen müssen, damit der Buffer überläuft. Dies wird durch

Gleichung 7.8 beschrieben. $T_{Buffer,n}$ stellt die bei der jeweiligen Streuung, und damit tatsächlichen Periodendauer, die vom Buffer speicherbare Zeit dar. Dabei gibt N die Anzahl an freien Rest-Speicherplätze an. V_n gibt die Anzahl an notwendigen Ereignissen mit einer Periodendauer Δt_n an, damit es zu einem Über-/Unterlauf kommt.

$$V_n = \left| \frac{T_{Buffer,n}}{\Delta t_{drift,n}} \right| = \left| \frac{\overline{\Delta t} - \Delta t_n \cdot N}{\Delta t_{drift,n}} \right| = \left| \frac{\overline{\Delta t} - \Delta t_n \cdot N}{\overline{\Delta t} - \Delta t_n} \right| \tag{7.8}$$

Mit $W_n = S_n^{V_n}$ kann dann die Wahrscheinlichkeit errechnet werden, dass die Periode V_n-mal nacheinander weiter oder gleichweit vom arithmetischen Mittel abweicht als Δt_n und es damit garantiert zu einem Überlauf kommt. Bild 7.4 zeigt die sich hierzu ergebenden Wahrscheinlichkeitswerte, wenn noch drei freie Speicherplätze vorhanden sind. Dargestellt ist jeweils der Exponent zur Basis 10 der Wahrscheinlichkeit. Zu sehen ist (rot im Bild 7.4), je kleiner die Ab-

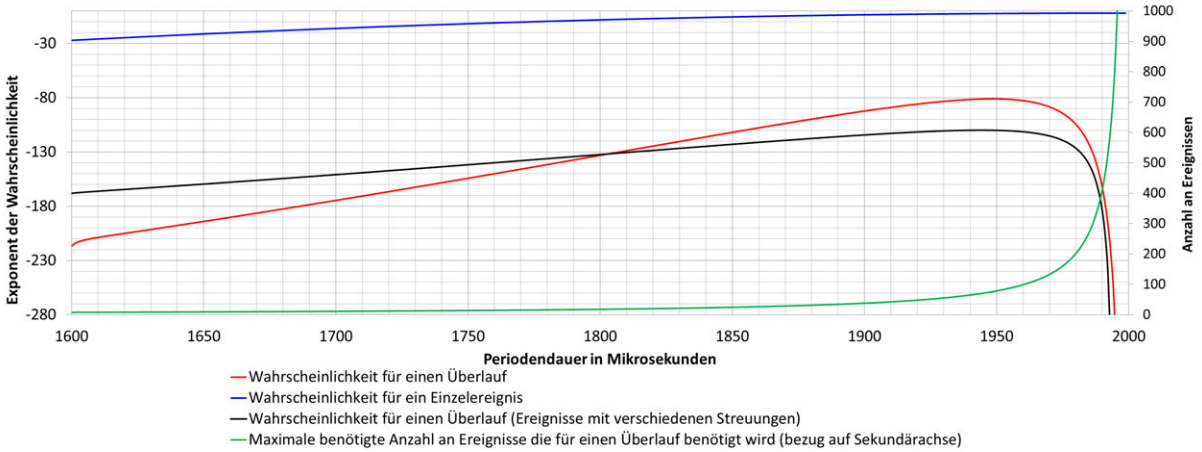


Bild 7.4.: Wahrscheinlichkeit eines Überlaufes in Abhängigkeit der Empfangsperiodendauer bei drei freien Restspeicherplätzen

weichung vom arithmetischen Mittelwert (hier $\overline{\Delta t} = 2000\,\mu s$), desto unwahrscheinlicher ist ein Über-/Unterlauf, denn die Wahrscheinlichkeit eines Einzelereignis (blau im Bild 7.4) ist zwar hoch, die Anzahl der für einen Überlauf benötigten Ereignisse (grün im Bild 7.4) aber auch. Je größer die Streuung wird, desto wahrscheinlicher wird ein Über-/Unterlauf, da die Anzahl an benötigten Ereignissen vorerst stärker abfällt, als die Wahrscheinlichkeit eines Einzelereignisses. Es wird ein Maximum in dem Punkt erreicht, in dem die Anzahl an benötigten Ereignissen nicht mehr stark genug abfällt, um die niedriger werdende Wahrscheinlichkeit für die Einzelereignisse zu kompensieren.
Es ist damit ersichtlich, dass eine Reihe von Ereignissen, die zu einem Überlauf führen kann, unabhängig von ihrer Kombination nicht wahrscheinlicher sein kann, als die in Bild 7.4 dargestellte höchste Wahrscheinlichkeit. Im Bild 7.4 ist dies exemplarisch dargestellt (schwarzer Verlauf): Würde eine Periode von $1800\,\mu s$ etwa 17-mal nacheinander auftreten, käme es zu einem Überlauf, wenn der Buffer noch drei Sweeps speichern kann. Im Bild dargestellt ist der Fall, dass nur 10 Ereignisse diese Periode aufweisen. Die restlichen, für einen Überlauf benötigten Ereignisse, haben die über die x-Achse aufgetragene Periode. Würden weniger als 10 Ereignisse die Periode von $1800\,\mu s$ aufweisen, nähert sich der Verlauf zunehmend der roten Linie an. Haben mehr als 10 Ereignisse diese Periode, nähert sich der Verlauf einer waagerechten, bei unverändertem Schnittpunkt mit der roten Linie, an. Dies zeigt, dass die Steigung der Wahrscheinlichkeit zu größeren Streuungen hin nie positiv und damit nie wahrscheinlicher als

das Maximum der roten Kurve werden kann.
In der Start-Phase der Elektronik kann nicht vorhergesagt werden, welchen Füllstand der Buffer

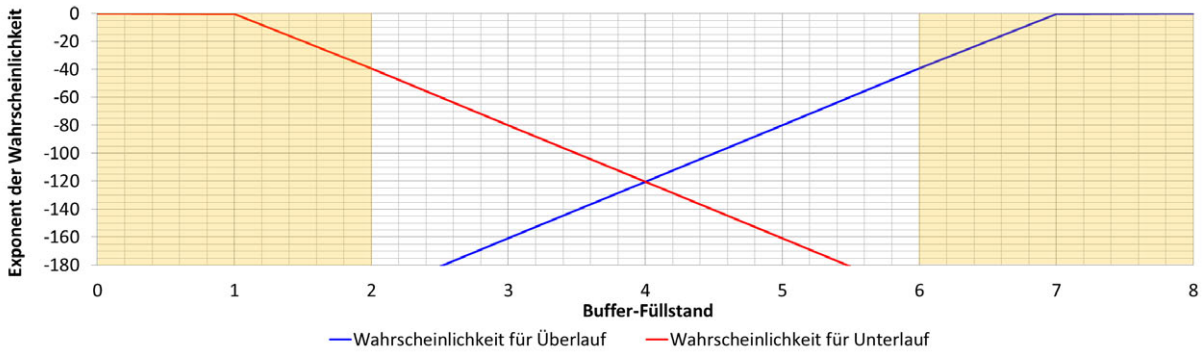


Bild 7.5.: Über-/Unterlaufwahrscheinlichkeit in Abhängigkeit des Buffer-Füllstandes

aufweist, wenn sich der MicroBlaze auf die zu rekonstruierende Periodendauer synchronisiert hat. Bild 7.5 zeigt hierfür die Wahrscheinlichkeit eines Über-/Unterlaufs in Abhängigkeit des Füllstandes für einen Buffer zur Speicherung von acht Sweeps. Jeweils bezogen auf die Abweichung vom arithmetischen Mittel, die die höchste Über-/Unterlauf-Wahrscheinlichkeit aufweist. Auch hier ist der Exponent der Wahrscheinlichkeit zur Basis 10 auf der y-Achse aufgetragen. Liegt der Füllstand in der Start-Phase in dem in Bild 7.5 orange gekennzeichneten Bereichen, führt der MicroBlaze eine temporäre Korrektur der Entnahme-Periodendauer um $\pm 1\,\mu s$ durch und korrigiert damit den Füllstand. Die Wahrscheinlichkeit eines Über-/Unterlaufes reduziert sich dadurch auf $P = 1 \cdot 10^{-41}$. Im ungünstigsten Betriebsmodus des Radar-Transceivers wird es so alle $2.11 \cdot 10^{29}$ Jahre zu einem Über-/Unterlauf kommen. Der Buffer könnte zwar noch kleiner gewählt werden, es ist dann jedoch kaum noch möglich den Füllstand des Buffers so zu bestimmen, dass die beschriebenen Korrekturen in der Startphase sinnvoll durchgeführt werden können. Der Buffer ist daher für die Speicherung von 8 Sweeps auszulegen.
Die hier durchgeführte Dimensionierung gilt für die Verwendung des Prototypen bei echten Radar-Transceivern. In der Testphase ist die Nutzung des in Abschnitt 7.1 entwickelten Radar-Simulators notwendig. Dieser weist eine höhere Streuung in der Sendeperiode auf, als die eines echten Radar-Transceivers. Die Wahrscheinlichkeiten der Perioden, entsprechen dabei keiner Normalverteilung. Eine mathematisch begründete Dimensionierung des Buffers für den Radar-Simulator ist daher nicht möglich sodass die Buffergröße des Prototypen empirisch ermittelt wurde. Der Buffer ist daher beim Prototypen auf 32 speicherbare Sweeps ausgelegt.

7.4. Untersuchung der Datendurchsätze

Im folgenden Abschnitt wird die Performance bei der Verarbeitung von Ethernet-Paketen mittels Hardware-Applikation (Abschnitt 7.4.1) und mittels Software-Applikation untersucht. Letzteres wird durch die Hardware-Software-Schnittstelle bestimmt. In Abschnitt 7.4.2 wird daher, neben der Zugriffsgeschwindigkeit auf einzelne Register, auch die Übertragung größerer Datenmengen untersucht.

7.4.1. Datendurchsätze des realisierten Ethernet-Interfaces

Um die Übertragungsgeschwindigkeit von Ethernet-Nachrichten auf Hardware-Ebene zu prüfen, sind hierfür temporär VHDL-Komponenten implementiert worden:

- Für den Sende-Test handelt es sich um eine Komponente, die kontinuierlich den Füllstand des Sende-Buffers abfragt und in diesem Nachrichten ablegt, sofern dieser nicht voll ist. Dadurch wird eine maximal mögliche Anzahl an Sende-Vorgängen ausgelöst. Als Nachrichteninhalt wird nach dem Iperf-Messverfahren in den ersten 32-Bit eine mit jeder Nachricht inkrementierte Zahl hinterlegt. Der Empfänger, in diesem Fall ein PC, kann die tatsächliche Sendegeschwindigkeit und Fehlerfreiheit überprüfen.
- Für den Empfangstest vergleicht die Empfangs-Komponente in der Applikations-Ebene den 32-Bit Iperf-Identifier in der Nachricht mit dem Identifier der zuletzt empfangenen Nachricht. Wird eine fehlende oder fehlerhafte Nachricht festgestellt, so wird ein Signal getoggelt, welches an einen mittels Oszilloskop messbaren Ausgangspin des FPGAs geführt wird. Damit lässt sich auch hier die Fehlerfreiheit sicher überprüfen.

Nachgewiesen werden konnte auf diese Weise eine Sendegeschwindigkeit von 94.9 MBit/s und eine Empfangsgeschwindigkeit von 94.8 MBit/s. Bei der Verarbeitung in Hardware konnte auch das zeitgleiche Senden und Empfangen von Daten mit den genannten Geschwindigkeiten nachgewiesen werden. Es ergibt sich damit ein verarbeitbarer Durchsatz von 189.7 MBit/s im Full-Duplex Betrieb. Die Messungen sind mit 1200 Byte Nutzdaten pro Ethernet-Paket durchgeführt worden.
In Abschnitt 5.4.3 ist ein HDL-Wrapper für den “Tri-Mode Ethernet-MAC” implementiert worden, mit dem eine Anbindung an ein Gigabit-Ethernet möglich ist. Geschwindigkeitstests zeigten, dass unidirektional Daten mit bis zu 291 MBit/s bei 1000 Byte großen Datenpaketen übertragen werden können. Im Fullduplex-Betrieb ließen sich bis zu 480 MBit/s, ebenfalls bei 1000 Byte großen Nachrichten, nachweisen. Ein deutlich schwankender Datendurchsatz, eine Verschlechterung des Datendurchsatzes zu kürzeren und längeren Datenpaketen hin, sowie die absolute Fehlerfreiheit beim Übertragungstest deuten darauf hin, dass die maximale Übertragungsgeschwindigkeit vom Test-PC begrenzt wird und nicht von der Elektronik selbst. Die HDL-Verhaltenssimulation bestätigt dies. Das theoretische Maximum läge bei 1 GBit/s pro Richtung und damit einem maximalen Datendurchsatz von 2 GBit/s bei zeitgleichem Senden und Empfangen.

7.4.2. Hardware-Software-Interface

In Abschnitt 4.2.4 sind im Rahmen der Konzeptionierung verschiedene Möglichkeiten bewertet worden, um den MicroBlaze an die Hardware anzubinden. An dieser Stelle soll nun die tatsächlich erreichbare Performance der Hardware-Software-Schnittstelle genauer untersucht werden. Wie bereits in Abschnitt 7.2.1 angewendet, wird auch hier ein manuelles Profiling durchgeführt, um die Dauer der jeweiligen Abarbeitungen zu bestimmen.

Einzelzugriffe

Zur Konfiguration und Steuerung der Hardware sind Zugriffe auf einzelne Register (32 Bit) notwendig. Nach der Verhaltenssimulation in Abschnitt 5.3.1 benötigt ein Einzelzugriff vier Takte. Nach Abschnitt 6.2 sind drei verschiedene Zugriffsfunktionen realisiert. Zwei Zugriffsfunktionen können Lese- bzw. Schreibzugriffe auf einzelne Register durchführen, eine weitere realisierte Funktion kann hierrüber hinaus auch auf die UDP-Buffer zugreifen und unterstützt dabei auch den implementierten Burst-Mode.
Die ersten beiden Funktionen werden innerhalb von 21 Takten bei einem Schreib-Zugriff und 22 Takten bei einem Lese-Zugriff abgearbeitet. Enthalten sind hierbei auch die Zeiten, die für den Aufruf und das Verlassen der Funktion benötigt werden. Ohne Funktionsaufruf konnte eine Zugriffszeit in Schreib-Richtung von vier Takten und in Lese-Richtung von 11 Takten nachgewiesen werden. Der Zeitunterschied entsteht in erster Linie durch den eingesetzten FSL-FiFo. Anders als bei Lesezugriffen ist bei Schreibzugriffen keine Reaktion durch die Hardware notwendig, wodurch keine Wartezeit entsteht. Beim Lese-Zugriff muss hingegen auf eine Reaktion gewartet werden. Die Reaktionszeit setzt sich aus der Verzögerung auf Grund des eingesetzen FSL-FiFo und der Reaktionszeit der FSL-Ansteuerung in der Hardware zusammen.
Prinzipiell ließen sich auch Einzelzugriffe mit der dritten Funktion durchführen, die flexibler ausgelegt ist. Die Messungen haben gezeigt, dass eine Nutzung dieser Funktion nur bei der Übertragung größerer Datenpakete sinnvoll ist. Es wird hier zusätzliche Rechenzeit für die vollständig dynamisch gehaltene Generierung des, die Kommunikation initiierenden, Datenwortes und für die eingesetzte Schleife zur Übertragung einer variablen Anzahl an Datenwörtern, benötigt. Wird keine Optimierung durch den C-Compiler durchgeführt, werden rund 125 Takte für einen Einzelzugriff benötigt. Bei maximaler Code-Optimierung durch den C-Compiler ließ sich die Einzelzugriffszeit auf 51 Takte reduzieren.

Übertragung größerer Datenmengen

Beim Versenden bzw. Empfangen von Daten über Ethernet sind größere Datenblöcke durch die FSL-Verbindung zu verarbeiten. Die Anzahl an zu übertragenden Bytes ist dabei nicht statisch gegeben. Es muss daher zur Übertragung der Daten die flexibel gehaltene Zugriffsfunktion verwendet werden. Hierbei entsteht ein, im vorherigen Abschnitt genannter, Overhead, der zusätzliche Rechenzeit benötigt. Die maximal verarbeitbare Datenmenge wird daher entweder durch die Hardware-Implementierung begrenzt (siehe Abschnitt 7.4.1) oder durch die Rechenleistung des Prozessors bzw. der Art der Übergabe der Daten zwischen Hard- und Software.
Tabelle 7.1 zeigt die Messergebnisse des Profiling beim Senden und Empfangen von Ethernet-Nachrichten durch den MicroBlaze. Die im Test erreichten maximalen Übertragungsgeschwindigkeiten liegen sehr nahe am durch die Hardware gegebenen Maximum. In der Tabelle 7.1 dargestellt sind ebenfalls die benötigten Prozessorauslastungen. Es zeigt sich auch hier ein deutlicher Unterschied zwischen keiner Optimierung und maximaler Optimierung des C-Codes durch den Software-Compiler.
Aufgrund der geringen Rechenauslastung bei den unidirektionalen Übertragungstests, ist auch das “zeitgleiche” Senden und Empfangen durch den MicroBlaze von der Rechenperformance her verarbeitbar. Übertragungstests haben gezeigt, dass der MicroBlaze fehlerfrei empfangene Nachrichten mit gleicher Geschwindigkeit wieder versenden kann. Eine Full-Duplex-Übertragungsgeschwindigkeit von 188.5 MBit/s konnte fehlerfrei nachgewiesen werden. Diese

	Beschreibung	Abarbeitungszeit		Takte		Geschwindig.		Auslastung	
		Unoptimiert	Optimiert	Unoptimiert	Optimiert	Unoptimiert	Optimiert	Unoptimiert	Optimiert
TX	Gesamt-Aufwand	126.6 μs	44.27 μs	8440	2951	72.8 MBit/s	94.1 MBit/s	97.48 %	43.6 %
	Protokoll-Aufwand	13.1 μs	7.11 μs	873	474				
	Kopier-Aufwand	113.5 μs	37.16 μs	7567	2477				
RX	Gesamt-Aufwand	120.8 μs	37.62 μs	8055	2508	76.0 MBit/s	94.7 MBit/s	95.50 %	37.62 %
	Protokoll-Aufwand	16.0 μs	6.00 μs	1069	400				
	Kopier-Aufwand	104.8 μs	31.62 μs	6986	2108				

Tabelle 7.1.: Performance-Analyse der Ethernet-Schnittstelle bei der Übertragung von 1200 Byte

entspricht fast der Übertragungsgeschwindigkeit, die auch in Hardware erreicht werden konnte. Das theoretische Maximum der FSL-Verbindung unter Verwendung der implementierten, dynamischen Zugriffsfunktion liegt nach den Ergebnissen in Tabelle 7.1 bei 206.8 MBit/s, wenn der MicroBlaze Daten schreibt, und 243.4 MBit/s wenn dieser Daten liest. Jeweils auf ein Systemtakt von 66.666 MHz bezogen.
Werden diese Ergebnisse mit der Prozessorauslastung eines in Software realisierten Ethernet-Stacks verglichen, ergibt sich eine deutlich niedrigere Prozessorauslastung: In den Voruntersuchungen in Abschnitt 3.3 ist der in Software realisierte Ethernet-Stack lwIP untersucht worden. In Verbindung mit dem Ethernet-MAC "XPS LL TEMAC" bei maximaler Nutzung des Cache-Speicher und Checksum-Offloading ergibt sich bei diesem eine fast doppelt so hohe Prozessorauslastung[8] bei der, von beiden Realisierungen erreichbaren maximalen Übertragungsgeschwindigkeit.
Es ist deutlich geworden, dass durch die dynamische Software-Zugriffsfunktion sehr viel Rechenleistung verloren geht. Ein Test zeigt, dass sich die Anzahl an benötigten Takten zur Verarbeitung einer 1200 Byte großen Nachricht um mehr als 60.3 % reduzieren lässt, wenn auf die Flexibilität verzichtet werden könnte. In diesem Fall wurde die in der Sende-Funktion verwendete Schleife "ausgerollt". Die Gesamtprozessorauslastung bei maximaler, unidirektionaler Übertragungsgeschwindigkeit würde auf 20.51 % sinken, bezogen auf einen Systemtakt von 66.666 MHz. Um auch diese Rechenleistung einzusparen, müsste der Einsatz einer DMA erfolgen.

7.5. Untersuchung der Ressourcenauslastung

Die Ressourcenauslastung des FPGA-Designs ist für die einzelnen Design-Bereiche in Tabelle 7.2 dargestellt. Bezogen wird der Verbrauch auf ein Spartan-6 FPGA. Die für die Auswahl einer

[8] Die Prozessorauslastung bei Verwendung des in Hardware realisierten Ethernet-Stacks ist auf einen Systemtakt von 125 MHz umgerechnet wurden. Diese ist in den Voruntersuchungen mit dem Virtex-5 FPGA genutzt worden. Es ergibt sich bei dieser Taktfrequenz eine Prozessorauslastung von 23.25 % beim Senden und 20.06 % beim Empfangen, wenn die maximal erreichbare Übertragungsgeschwindigkeit zu verarbeiten ist.

Derivate entscheidende Ressourcen-Größe stellt der BlockRAM dar und weniger die Logik-Slices. Im Prototypen-Design liegt die Anzahl an benötigtem BlockRAM bei 107 Blöcken und damit 59 Blöcke über den Minimalanforderungen an eine Lösung, wie sie für ein späteres Produkt notwendig wäre. Dies liegt darin begründet, dass der in Abschnitt 7.1 entwickelte Radar-Simulator, der für System-Tests benötigt wurde, eine höhere Streuung aufweist, als ein echter Radar-Transceiver. Diese ungewollte Streuung musste durch den Buffer von der Ausgabeperiode entkoppelt werden können. Dies ist bei einer Produktrealisierung nicht mehr notwendig. Da der MicroBlaze Steuerinformationen in Form von Ethernet-Nachrichten mit nur geringem Datendurchsatz verarbeiten können muss, ist auch eine Reduzierung des Ethernet-Buffers auf zwei Speicherblöcke je Richtung akzeptierbar. Es ist daher vertretbar, die benötigten Ressourcen von 107 auf 48 BlockRAM-Blöcke zu verringern. Nach [53] ergibt sich damit die Möglichkeit anstelle der Derivate XC6SLX45T, eingesetzt auf dem SP605-Evaluationboard, die kleinere Derivate XC6SLX25 einzusetzen[9]. Dies entspräche einer Kostenersparnis von 43 % gegenüber dem XC6SLX45 [17]. Prinzipiell ist eine Realisierung auch auf einem FPGA der Spartan-3 Familie möglich[10](XC3S2000), der jedoch keinen Preisvorteil hat.

Ressource	Ethernet-MAC	Ethernet-Stack	MicroBlaze	Radar-Applikation	Ethernet-Applikation	Daten-Manager	Sonstiges	Gesamt (Synthese)	Gesamt (Implementierung)
LUT	1095	1166	4017	492	939	930	636	9275	7111
Register	754	1507	3349	377	65	1147	243	7442	6886
BlockRAM	0	0	35	11 (48)	2 (24)	0	0	48 (107)	48 (107)

Tabelle 7.2.: Ressourcen-Verbrauch nach Verwendungsbereich nach der Synthese

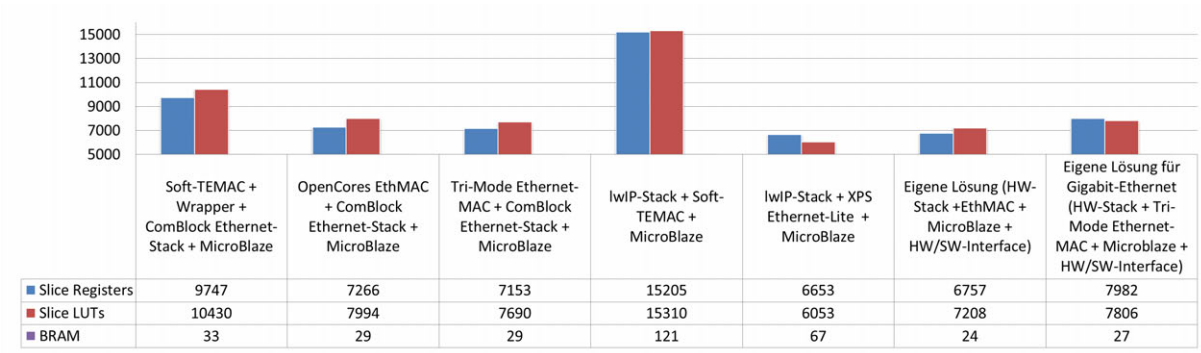


Bild 7.6.: Vergleich verschiedener Ethernet-Anbindung mit eigener Lösung

Im Zuge der Voruntersuchungen (siehe Abschnitt 3.5) sind verschiedene Realisierungsmöglichkeiten einer Ethernet-Anbindung aufgezeigt und deren Ressourcenverbräuche gegenübergestellt worden. Diese Realisierungen sind in Bild 7.6 (verwendete y-Achsenskalierung beachten) zusammen mit der für diese Arbeit implementierten Lösung dargestellt. Diese Lösung basiert auf

[9] Die Auslastung der Slice-Register liegt dann bei 22.9 %, die der Slice-LUTs bei 60.6 %. Die BlockRAM-Blöcke sind zu 92 % genutzt.

[10] Auf einem FPGA der Spartan-3 Familie werden 7762 Slices (8034 FlipFlops und 12069 LUTs) benötigt.

den Projektbestandteilen Ethernet-MAC, Ethernet-Stack, Konfigurations-Interface und MicroBlaze.
In der Konzeptionierung galt der Ethernet-MAC "EthMAC" zusammen mit dem Ethernet-Stack von ComBlock als Referenz für den Ressourcenverbrauch (zweite Lösung von links). Es zeigt sich, dass der Ressourcenverbrauch der eigenen Lösung sehr gut mit dieser Referenz übereinstimmt. Es werden 6 - 7 % weniger Logik-Ressourcen benötigt. Das entwickelte Ethernet-Interface benötigt damit den geringsten Ressourcenverbrauch aller aufgeführter Lösungen, die eine Übertragungsgeschwindigkeit größer 50 MBit/s zulassen. Gegenüber einer Lösung, die keine Entwicklung des Ethernet-Stack, Ethernet-MAC und der Anbindung zwischen Hard- und Software benötigt, wie es bei der Verwendung des Xilinx "XPS LL TEMAC" mit dem lwIp-Stack der Fall ist (vierte Lösung in Bild 7.6), können 55.6 % an Slice-Registern, 53 % an Slice-LUTs und 80 % der BlockRAM-Ressourcen eingespart werden.
Die ebenfalls realisierte Lösung, bei der der entwickelte Ethernet-Stack mit dem "Tri-Mode Ethernet-MAC" zur Anbindung an ein Gigabit-Ethernet genutzt wird, kann zwar höhere Übertragungsraten verarbeiten, benötigt jedoch auch mehr Ressourcen (siehe sechste Lösung), sodass diese Lösung nicht für den Prototypen verwendet wird.
Nach Tabelle 7.2 werden etwa 43 % - 45 % der Logik-Ressourcen und 73.3 % der Speicherressourcen für den MicroBlaze benötigt. Durch die gegebene Flexibilität des Ethernet-Interface ließe sich das Ethernet-Interface auch ohne MicroBlaze einsetzen. In diesem Fall kann ein Einsatz etwa auf dem zweitkleinsten Xilinx Spartan-3E [51] Derivat (xc3s250e[11]) erfolgen.

[11] Die Auslastung der Slice-Register liegt dann bei 43 %, die der Slice-LUTs bei 67 %.

KAPITEL 8

Zusammenfassung und Ausblick

8.1. Zusammenfassung

Ziel dieser Masterarbeit war der Entwurf und die Entwicklung eines Hardware-Software-Codesigns als Prototyp. Dessen Aufgabe ist es, digitalisierte Informationen über empfangene Echos von einem Radar-Transceiver wieder zu analogisieren. Zum Empfang dieser digitalen Informationen ist eine performante Anbindung des Designs an ein Ethernet-Netzwerk notwendig, sodass UDP/IP-Nachrichten mit mindestens 50 MBit/s bei einer Paketgröße von 1200 Byte empfangen werden können. Der Empfang von Multicast-Nachrichten, einschließlich der Verwaltung der Multicast-Gruppen-Mitgliedschaften, muss hierzu unterstützt werden. Die Besonderheit in der Generierung der analogen Signale liegt in der Rekonstruktion des ursprünglichen Zeitverhaltens des Radar-Transceivers. Die harten Echtzeitanforderungen müssen wieder hergestellt werden, da das Zeitverhalten durch die digitale Verarbeitung der Echo-Informationen verloren gegangen ist.
Um die sich so ergebenden Anforderungen nach Abschnitt 1.1 erfüllen zu können, sind zu Beginn der Arbeit in Kapitel 3 Voruntersuchungen durchgeführt worden. Die Anbindung des FPGAs an ein Ethernet wurde hier als die Ressourcen- und Performance-intensivste Systemkomponente angesehen. Dadurch sind hier die meisten Auswirkungen auf die Realisierbarkeit zu erwarten gewesen. Dies liegt nicht nur an der benötigten Rechenleistung für die Ethernet-Anbindung selbst, sondern auch an den Auswirkungen die die Abarbeitung auf die Generierung der Radar-Pulse hat. Letztere müssen innerhalb definierter Zeitschranken abgearbeitet werden. Die untersuchten Möglichkeiten reichen von der Nutzung von reinen Software-basierten Ethernet-Stacks, über Software-basierte Ethernet-Stacks unter Verwendung von Hardware-Beschleunigern bis hin zu vollständig in Hardware realisierten Ethernet-Stacks.
Auf Grundlage dieser Ergebnisse ist in der nachfolgenden Konzeptionierung in Kapitel 4 entschieden worden, einen Ethernet-Stack in Hardware zu realisieren. Dies erlaubt die Verarbeitung großer Datenmengen, wie hier die Abtastwerte des Radar-Echos, vollständig in Hardware zu realisieren. Eine komplexere Verarbeitung dieser Informationen ist nicht notwendig. Hingegen können etwa Header-Informationen der einzelnen Radar-Sweeps, die eine komplexere Auswertung und Weiterverarbeitung bedürfen, jedoch einen geringeren Datenumfang aufweisen, in Software durch den MicroBlaze verarbeitet werden. Das Echtzeitverhalten wird so nicht durch die Verarbeitung von Daten mit hohem Datendurchsatz beeinflusst.
Nach der Konzeptionierung ist mit der Implementierung des HDL-Designs in Kapitel 5 begonnen worden. Die Schwerpunkte in der Entwicklung lagen hier zum einen beim Ethernet-Interface mit der Entwicklung des Ethernet-Stacks und der Optimierung des Ethernet-MACs, zum anderen in der Video-Verarbeitung, wozu die Interpretation, die Zwischenspeicherung bis hin zur DAC-Ansteuerung gehören. Probleme, die hierbei entstanden, lagen vor allem in der

Einhaltung des notwendigen Timings und darin, die Reproduzierbarkeit von Synthese- und Implementierungsergebnissen sicherzustellen.
Während und nach der Implementierung des HDL-Designs ist die Software für den MicroBlaze entwickelt worden, welches Kapitel 6 thematisiert hat. Der Schwerpunkt lag hier vor allem in der Berechnung der Radar-Pulse in einer Weise, dass die Anforderungen an die Signaleigenschaften in allen Betriebsmodi des Radar-Transceivers erfüllt werden können. Ein weiterer Schwerpunkt lag in der Steuerung des HDL-Designs, einschließlich Empfang und Interpretation von Steuer-Nachrichten über Ethernet. Diese werden in Hardware von den Video-Paketen separiert und dem MicroBlaze zur Verfügung gestellt. Auf diese Weise ließ sich auch eine Update-Funktion implementieren, mit der ein neues FPGA-Konfigurations-Image über Ethernet in den externen SPI-Flash geladen und das FPGA rekonfiguriert werden kann, ohne dass ein direkter Zugang zu der Elektronik bestehen muss. Abgesichert wird das Update unter Verwendung der "Fallback Multiboot"-Funktion. Die größten Probleme lagen in der Generierung der Radar-Pulse und der Einhaltung der Anforderungen an die Signaleigenschaften.
Im abschließenden Kapitel 7 ist die Verifikation des Designs durchgeführt worden, um die Einhaltung der spezifizierten Anforderungen zu überprüfen. Im Fokus stand hier die Überprüfung der Radar-Pulse und dass diese korrekt und unter Einhaltung der maximal zulässigen Jitter reproduziert werden. Auf Basis dieser und weiterer Messungen ist dann die Dimensionierung des Video-Buffers mittels stochastischer Betrachtung durchgeführt worden. Die zeitliche Entkoppelung zwischen der Empfangs- und Ausgabeperiode kann so garantiert werden, ohne dass es zu Über- oder Unterläufen des Buffers kommen kann. Die korrekte Reproduzierung des Zeitverhaltens des Radar-Transceivers ist so fehlerfrei sichergestellt.

8.2. Ausblick

Der hier implementierte Prototyp erfüllt alle gestellten Anforderungen. Für eine Entwicklung zum Produkt sind weitere Entwicklungen notwendig. Hierzu zählt der Entwurf und das Layouten der Schaltung, deren Inbetriebnahme, bis hin zur Typprüfung und Abnahme der Elektronik bei einer Zulassungsbehörde, damit diese auf einem Schiff eingesetzt werden darf. Für die Produktentwicklung ist über weitere Funktionalitäten nachzudenken. Hierbei kann es sich etwa um eine über das FPGA steuerbare, variable Anpassung der analogen Signaleigenschaften von Trigger, Video, ACP und ACP handeln. Variable Anpassung sind u. a. bei den Pegeln, der Polarität und der Ausgangsimpedanz sinnvoll. Ebenfalls könnten zusätzliche Kommunikationsschnittstellen, etwa für einen CAN-Bus implementiert werden, sodass über ein CAN-zu-Ethernet-Umsetzer die Sichtgeräte auch einen steuernden Einfluss auf den Radar-Transceiver nehmen könnten.
Unabhängig von diesem Prototyp lässt sich das implementierte Ethernet-Interface auch in anderen Projekten bzw. Produkten einsetzen. Der Ethernet-Stack ist hierfür so entwickelt worden, dass der Funktionsumfang durch den modularen Aufbau flexibel an die jeweiligen Anforderungen angepasst werden kann, sodass kein unnötiger Ressourcenverbrauch entsteht aber auch zusätzliche Protokolle realisierbar sind. Durch die Entwicklung von VHDL-Wrappern zur Anbindung von anderen Ethernet-MACs, wie dem "Tri-Mode Ethernet-MAC" von OpenCores, konnte gezeigt werden, dass auch eine Anforderung an eine höhere Datenübertragungsrate bei Bedarf erfüllt werden kann, ohne dass ein Prozessor mit der Verarbeitung der Netzwerk-Protokolle belastet werden muss.

Literaturverzeichnis

[1] Analog Devices (Hrsg.): Datenblatt AD9716; Online im Internet: `http://www.analog.com/static/imported-files/data_sheets/AD9714_9715_9716_9717.pdf` [Abrufdatum: 01.12.2011]

[2] Analog Devices (Hrsg.): AD9716 Evaluation Board; Online im Internet: `http://www.analog.com/en/digital-to-analog-converters/da-converters/ad9716/products/EVAL-AD9716/eb.html` [Abrufdatum: 31.12.2011]

[3] Analog Devices (Hrsg.): ADA4857 Datenblatt; Online im Internet: `http://www.analog.com/static/imported-files/data_sheets/ADA4857-1_4857-2.pdf` [Abrufdatum: 07.01.2012]

[4] Analog Devices (Hrsg.): ADA4841 Datenblatt; Online im Internet: `http://www.analog.com/static/imported-files/data_sheets/ADA4841-1_4841-2.pdf` [Abrufdatum: 07.01.2012]

[5] Analog Devices (Hrsg.): AD811 Datenblatt; High Performance Video Op Amp; Online im Internet: `http://www.analog.com/static/imported-files/data_sheets/AD811.pdf` [Abrufdatum: 22.01.2012]

[6] Adamek, Thomas: Vorlesung Wahrscheinlichkeitslehre und Statistik, Kapitel 7 (Korrelation und Regression); Online im Internet: `http://www.uni-stuttgart.de/bio/adamek/numerik/statistik5.pdf` [Abrufdatum: 19.12.2011]

[7] Altera Corporation: Alterawiki - AXI Support; Online im Internet: `http://www.alterawiki.com/wiki/AXI_Support?GSA_pos=1&WT.oss_r=1&WT.oss=AXI` [Abrufdatum: 30.12.2011]

[8] Bolls, Kevin: Bestimmung der Werte nach Synthese der Projekte durch PlanAhead (v13.2) für Register und LUTs, sowie Nutzung der Quellen [11], [12], [13] und [58]. Register- und LUT-Angaben des “CAST MAC-1G” und “CAST 10/100 MAC” wurden aus den Slice-Registern unter Berücksichtigung von Abschnitt 2.1 und unter Verwendung der Sliceauslastung des Ethernet-MAC von ComBlock (COM_5401SOFT) geschätzt. Die Anzahl der Slices für “OpenCores EthMAC”, “OpenCores EthMAC mit Wrapper” und “Hard-TEMAC + Wrapper” sind unter Verwendung der Benötigten Register- und LUT-Anzahl ebenfalls geschätzt, da diese Teil eines Gesamtprojektes waren und damit keine seperaten Slice-Angabe existiert.

[9] Bormann A., Hilgenkamp I.: Industrielle Netze - Ethernet-Kommunikation für Automatisierungsanwendungen; Hütig-Verlag Heidelberg; 3. Auflage

[10] Bryan H. Flatcher (Memec): Embedded Training Program, Embedded Systems Conference San Francisco 2005, ETP-357

[11] CAST INC.: 10/100 Ethernet Media Access Controller Core; Online im Internet: http://www.cast-inc.com/ip-cores/interfaces/mac/index.html [Abrufdatum: 14.12.2011]

[12] CAST INC.: 1-Gigabit Ethernet Media Access Controller Core; MAC-1G; Online im Internet: http://www.cast-inc.com/ip-cores/interfaces/mac-1g/index.html [Abrufdatum: 14.12.2011]

[13] ComBlock: COM-5401SOFT Tri-Mode 10/100/1000 Ethernet; MAC VHDL SOURCE CODE OVERVIEW; Online im Internet: http://www.comblock.com/download/com5401soft.pdf [Abrufdatum 14.12.2011]

[14] ComBlock: CCOM-5402SOFT IP/TCP/UDP/ARP/PING STACK for GbE; Online im Internet: http://comblock.com/download/com5402soft.pdf [Abrufdatum 15.12.2011]

[15] Ethernet-Stack: lwIP - A Lightweight TCP/IP stack; Online im Internet: http://savannah.nongnu.org/projects/lwip/ [Abrufdatum: 03.11.2011]

[16] Fey, Prof. Dr. Dietmar: Paralleles Rechnen für die Bildverarbeitung auf Cluster-Rechnern und in FPGA - Algorithmenstudien anhand zellulärer Automaten; Online im Internet: http://www.vision-academy.de/fileadmin/user_upload/vision_academy/media/pdf/Workshops/27_Workshop_Parallelverarbeitung.pdf [Abrufdatum: 30.11.2011]

[17] Farnell: Einzellstückpreis für die kleinst mögliche FPGA-Derivate (XC6SLX25) liegt bei 36.35 €, der Preis für den XC6SLX45 liegt bei 63.36 € und für den XC6SLX45T (Eingesetz auf dem Evalboard) bei 93.51 €. Preise wurden am 19.01.2012 auf http://de.farnell.com/ abgefragt.

[18] Internation Standard Organisation: OSI-Modell; Online im Internet: http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip [Abrufdatum: 09.01.2012]

[19] Löfgren A., Lodesten L., Sjoholm S. and Hansson S.; "An analysis of fpga-based UDP/IP stack parallelism for embedded ethernet connectivity" in Proceedings of Norchip Conference; Oulu, Finland; 23.11.2005; Seite 94-97

[20] Link, Johannes: Softwaretests mit JUnit - Techniken der testgetriebenen Entwicklung; Dpunkt Verlag; Auflage: 2.; ISBN: 3898643255

[21] Mikrocontroller.net; Matthias G.: Generierung eines Jitters in einem Takt; Online im Internet: http://www.mikrocontroller.net/topic/186044#1805373 [Abrufdatum: 09.01.2012]

[22] OpenCores.org: WISHBONE System-on-Chip (SoC); Online im Internet: http://cdn.opencores.org/downloads/wbspec_b3.pdf [Abrufdatum: 01.12.2011]

[23] OpenCores.org: EthMAC für 10/100 MBit/s; Online im Internet: http://opencores.org/project,ethmac [Abrufdatum: 01.12.2011]

[24] OpenCores.org: Die Sourcen des EthMAC [23] beinhalten Verilog-Defines, die ein Umschalten von Wishbone-Bus auf den Avalon-Bus möglich machen.

[25] OpenCores.org: Tri-Mode Ethernet MAC; Online im Internet: `http://opencores.org/project,ethernet_tri_mode` [Abrufdatum: 14.12.2011]

[26] OpenCores.org: 1G eth UDP / IP Stack; Online im Internet: `http://opencores.org/project,udp_ip_stack` [Abrufdatum: 15.12.2011]

[27] OpenCores.org: UDP/IP core; Online im Internet: `http://opencores.org/project,udp_ip__core` [Abrufdatum: 31.12.2011]

[28] NLANR/DAST: Iperf; Online im Internet: `http://sourceforge.net/projects/iperf/` [Abrufdatum: 15.07.2011]

[29] Papula, Lothar: Mathematische Formelsammlung für Ingenieure und Naturwissenschaftler; 10. Auflage; Vieweg + Teubner Verlag

[30] Péter Arató, Sándor Juhász, Zoltán Ádám Mann, András Orbán, Dávid Papp: Hardware-software partitioning in embedded system design; Budapest University of Technology and Economics; Online im Internet: `http://rutcor.rutgers.edu/~dpapp/pub/hwsw_proc.pdf` [Abrufdatum: 01.12.2011]

[31] Programmable Logic Competence Center (PLC2): Schulungsunterlagen des Workshop "Timing Constraints"; 2010

[32] Programmable Logic Competence Center (PLC2): Schulungsmitschriften zum Workshop „Timing Constraints" von Sven Artz (ehmaliger Angestellter der Entwicklung); 2010

[33] J. Reichardt, B. Schwarz: VHDL-Synthese - Entwurf digitaler Schaltungen und Systeme; 5. Auflage; Oldenbourg-Verlag; ISBN: 978-3-486-58987-0

[34] J. Reichardt: Lehrbuch Digitaltechnik - Eine Einführung mit VHDL; 1. Auflage; Oldenbourg-Verlag; ISBN: 978-3-486-58908-5

[35] Rostock, Tom: Bachelorthesis: Entwurf und Realisierung eines FPGA basierten Software Defined Radios mit Echtzeit-Streaming-Schnittstelle zum PC, Vorabversion, August 2011

[36] Stavinov,Evgeni (SerialTek LLC): Using Xilinx Tools in Command-Line Mode - Uncover new ISE design tools and methodologies to improve your team's productivity; Xcell Journal; 1.Quartal 2011; Online im Internet: `http://outputlogic.com/xcell_using_xilinx_tools/74_xperts_04.pdf` [Abrufdatum: 01.12.2011]

[37] Steinkamp, Olaf: Vorlesungsfolien Statistik - Fehlerrechnung, Datenanalyse (PHY231-HS11), Herbstsemester 2011; Online im Internet: `http://www.physik.uzh.ch/lectures/datenanalyse/11/vorlesung/da11ffp.pdf` [Abrufdatum: 19.12.2011]

[38] Voß, Wolfang: Gespräch über Lebensdauer von Navigationsanlagen; Januar 2012

[39] Wikia: lwIP Wiki - Maximizing throughput; Online im Internet: `http://lwip.wikia.com/wiki/Maximizing_throughput` [Abrufdatum: 18.01.2012]

[40] Wikimedia Foundation: Fast Simplex Link; Online im Internet: http://en.wikipedia.org/wiki/Fast_Simplex_Link [Abrufdatum: 30.11.2011]

[41] Wikimedia Foundation: Embedded System; Online im Internet: http://de.wikipedia.org/wiki/Embedded_System [Abrufdatum: 01.12.2011]

[42] Wikimedia Foundation: Standardabweichung; Online im Internet: http://de.wikipedia.org/wiki/Standardabweichung [Abrufdatum: 09.12.2011]

[43] Wikimedia Foundation: Jumbo Frames; Online im Internet: http://de.wikipedia.org/wiki/Jumbo_Frames [Abrufdatum: 31.12.2011]

[44] Wikimedia Foundation: Wishbone-Computer-Bus; Online im Internet: http://en.wikipedia.org/wiki/Wishbone_(computer_bus) [Abrufdatum: 01.01.2012]

[45] Wikimedia Foundation: Intel Hex; Online im Internet: http://de.wikipedia.org/wiki/Intel_HEX [Abrufdatum: 01.12.2011]

[46] Winbond Electronics Corp.: 64M-bit Serial Flash Memory with uniform 4KB sectors and Dual/Quad SPI (Datenblatt); Online im Internet: http://www.winbond.com.tw/NR/rdonlyres/591A37FF-007C-4E99-956C-F7EE4A6D9A8F/0/W25Q64BV.pdf [Abrufdatum: 03.11.2011]

[47] Wireshark: Network Protocol Analyzer, Version 1.6.1; Online im Internet: http://www.wireshark.org/ [Abrufdatum: 15.07.2011]

[48] Wireshark: Time-Stamps; Online im Internet: http://www.wireshark.org/docs/wsug_html_chunked/ChAdvTimestamps.html [Abrufdatum: 09.12.2011]

[49] Xilinx Inc. (Hrsg.): XAPP1026 (v3.1): LightWeight IP (lwIP) Application Examples Veröffentlicht am 21. April 2011. Online im Internet: http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf [Stand 22.10.2011]

[50] Xilinx: LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c); 19.4.2010; Online im Internet: http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf [Abrufdatum: 01.12.2011]

[51] Xilinx: Spartan-3E Family Datasheet; DS312 (v3.8); 26.08.2009; Online im Internet: http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf [Abrufdatum: 30.11.2011]

[52] Xilinx: Virtex-5 Family Overview; DS100 (v5.0); 6.2.2009; Online im Internet: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf [Abrufdatum: 06.12.2011]

[53] Xilinx: Spartan-6 Family Overview; DS160 (v2.0); 25.10.2011; Online im Internet: http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf [Abrufdatum: 30.11.2011]

[54] Xilinx: LogiCORE IP Fast Simplex Link (FSL) V20 Bus (Datenblatt); Online im Internet: http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf [Abrufdatum: 30.11.2011]

[55] Xilinx: Processor Local Bus (PLB) v3.4 (Datenblatt,v1.02a); Online im Internet: http://www.xilinx.com/support/documentation/ip_documentation/plb_v34.pdf [Abrufdatum: 30.11.2011]

[56] Xilinx: AXI Reference Guide UG761, 7.3.2011; Online im Internet: http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf [Abrufdatum: 30.11.2011]

[57] Xilinx: Incremental Design Reuse with Partitions; Online im Internet: http://www.xilinx.com/support/documentation/application_notes/xapp918.pdf [Abrufdatum: 05.11.2011]

[58] Xilinx: LogiCORE IP XPS LL TEMAC (Datenblatt); Online im Internet: http://www.xilinx.com/support/documentation/ip_documentation/xps_ll_temac.pdf [Abrufdatum: 05.11.2011]

[59] Xilinx: LogiCORE IP XPS Ethernet Lite Media Access Controller (Datenblatt); Online im Internet: http://www.xilinx.com/support/documentation/ip_documentation/xps_ethernetlite.pdf [Abrufdatum: 01.01.2012]

[60] Xilinx: Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Wrapper v1.7; Getting Started Guide; UG340 April 19, 2010; Online im Internet: http://www.xilinx.com/support/documentation/ip_documentation/v5_emac_gsg340.pdf [Abrufdatum: 14.12.2011]

[61] Xilinx: SP605 MultiBoot Design, Dezember 2009; Online im Internet: http://www.xilinx.com/support/documentation/boards_and_kits/xtp059.pdf [Abrufdatum: 01.12.2011]

[62] Xilinx: Get Smart About Reset: Think Local, Not Global; Ken Chapman; WP272(v1.0.1); 7. März 200; Online im Internet: http://www.xilinx.com/support/documentation/white_papers/wp272.pdf [Abrufdatum: 01.12.2011]

[63] Xilinx: Get your Priorities Right - Make your Design Up to 50 % Smaller; Ken Chapman; WP275(v1.0.1), 22. Oktober 2007; Online im Internet: http://www.xilinx.com/support/documentation/white_papers/wp275.pdf [Abrufdatum: 01.12.2011]

[64] Xilinx: Spartan-6 FPGA Configuration - User Guide; UG380 (v2.3); 6.6.2011; Online im Internet: http://www.xilinx.com/support/documentation/user_guides/ug380.pdf [Abrufdatum: 01.12.2011]

[65] Xilinx: Anmerkung zum Fehler “ERROR:Place:1206“ in der ISE Design Suite 13.2

[66] Xilinx: Spartan-6 FPGA SelectIO Ressources - User Guide; UG281 (v1.4); 16.12.2010; Online im Internet: http://www.xilinx.com/support/documentation/user_guides/ug381.pdf [Abrufdatum: 05.12.2011]

[67] Xilinx: Timing Constraints - User Guide; UG612 (v 13.2); 6.6.2011; Online im Internet: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/ug612.pdf [Abrufdatum: 06.12.2011]

[68] Xilinx: Hierarchical Design Methodology Guide; UG748 (v 13.2); 6.7.2011; Online im Internet: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/Hierarchical_Design_Methodology_Guide.pdf [Abrufdatum: 06.12.2011]

[69] Xilinx: MicroBlaze Processor Reference Guide; UG081 (v5.1); 2.4.2005; Online im Internet: http://joule.bu.edu/~hazen/DataSheets/Xilinx/mb_ref_guide.pdf

[70] Xilinx: LogiCORE IP XPS Interrupt Controller; S572 (v2.01a); 19.4.2010; Online im Internet: http://www.xilinx.com/support/documentation/ip_documentation/xps_intc.pdf

[71] Xilinx: Spartan-6 Libraries Guide for HDL Designs; UG615 (v 12.4); 14.12.2010; Online im Internet: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/spartan6_hdl.pdf

[72] Xilinx: LogiCORE IP XPS Serial Peripheral Interface (SPI); DS570 (v2.02a); 22.06.2011; Online im Internet: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/spartan6_hdl.pdf

[73] Xilinx: Command Line Tools User Guide; UG628 (v 13.1); 02.04.2011; Online im Internet: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/devref.pdf [Abrufdatum: 02.01.2012]

[74] Xilinx: HDL Coding Practices to Accelerate Design Performance (Whitepaper); WP231 (v1.1); 06.01.2006; Online im Internet: http://www.xilinx.com/support/documentation/white_papers/wp231.pdf [Abrufdatum 02.01.2012];

[75] Xilinx: Maintaining Repeatable Results (Whitepaper); WP361 (v1.1); 03.03.2010; Online im Internet: http://www.xilinx.com/support/documentation/white_papers/wp361.pdf [Abrufdatum 02.01.2012];

[76] Xilinx: Application Note: Embedded Processing; Reference System: XPS LL Tri-Mode Ethernet MAC Embedded Systems for MicroBlaze and PowerPC Processors; XAPP1041 (v2.0); 24.09.2008; Online im Internet: http://www.xilinx.com/support/documentation/application_notes/xapp1041.pdf [Abrufdatum 05.01.2012];

[77] Xilinx: Clocking-Wizard im ISE "Design Navigator" 13.2; Angabe über Peak-To-Peak Jitter auf Page 5 von 6

[78] Xilinx: Forum-Beitrag "Multiboot with Spartan 6"; Online im Internet: http://forums.xilinx.com/t5/Spartan-Family-FPGAs/Multiboot-with-Spartan-6/m-p/170716#M12727 [Abrufdatum 01.12.2011];

[79] Xilinx: AXI Reference Guide; UG761 (v13.1); 7.3.2011; Online im Internet: http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf [Abrufdatum 29.01.2012];

[80] Xilinx: XST User Guide; UG627 (v11.3); 16.09.2009; Online im Internet: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/xst.pdf [Abrufdatum 10.02.2012];

[81] Yi-Mao Hsiao, Ming-Jen Chen, Kuo-Chang Huang, Yuan-Sun Chu und Chingwei Yeh: High Speed UDP/IP ASIC Design; Veröffentlichung des Dept. of Electr. Eng., Nat. Chung Cheng Univ., Chiayi, Taiwan im ISPACS (Intelligent Signal Processing and Communication Systems) 2009. Print ISBN: 978-1-4244-5015-2

ANHANG A

Anhang

A.1. Protokolle

Im folgenden Abschnitt wird auf den konkreten Aufbau von Netzwerk-Protokollen eingegangen, die im Ethernet-Stack in Abschnitt 5.5 implementiert sind.

A.1.1. MAC-Frame

Der MAC-Frame unterteilt sich in drei Bereiche. Während der erste und der letzte Teil durch den Ethernet-MAC in Hardware generiert wird, wird der mittlere Teil durch den Ethernet-Stack und damit oft in Software generiert.
Der erste Teil besteht aus Präambel (eine alternierende Bitfolge) und Start Frame Delimiter (SFD). Diese wurden früher zur Taktsynchronisation eingesetzt und sind heute nur noch aus Kompatibilitätsgründen vorhanden. Im dritten Teil des Frames befindet sich die Inter-Frame-Gap (IFG), die zuständig ist, einen minimalen Abstand zwischen Nachrichten sicherzustellen. Ebenfalls im dritten Teil wird die CRC-Summe angegeben, welche über die vollständige Nachricht gebildet wird und diese auf Bit-Fehler hin absichert. Berechnung bzw. Überprüfung der CRC-Summe findet ebenfalls in Hardware statt. Die benötigte Rechenleistung bei der Verarbeitung in Software wäre zu hoch. Bild A.1 zeigt damit den Teil des MAC-Frames, welcher in Hardware durch den Ethernet-MAC generiert wird. Nach Bild A.2 sind folgende Datenfelder

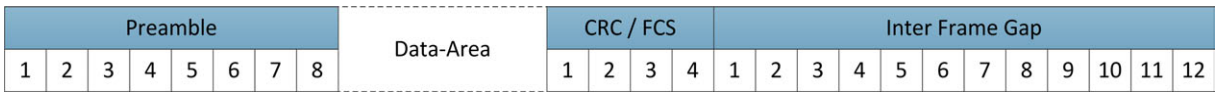


Bild A.1.: Aufbau des MAC-Frame (Bereich, der im Ethernet-MAC verarbeitet wird)

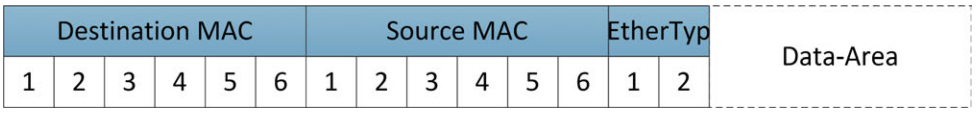


Bild A.2.: Aufbau des MAC-Frame (Bereich, der im Ethernet-Stack verarbeitet wird)

im mittleren Teil des MAC-Frames enthalten:

- Destination MAC:
  Beinhaltet die physikalische Adresse des Ziels. Jeder Netzwerk-Teilnehmer in einem Subnetz weist eine einmalige MAC-Adresse auf. Im Gegensatz zu IP-Adressen sind diese nicht veränderbar. Wird eine Unicast-Nachricht an einen Teilnehmer gesendet, so

muss zuvor die MAC-Adresse unter Verwendung des ARP-Protokolls (siehe Abschnitt A.1.4) erfragt werden. Werden Broadcast-Nachrichten versendet bzw. empfangen wird hier die MAC-Adresse FF:FF:FF:FF:FF:FF eingetragen. Bei Multicast-Adressen leitet sich die MAC-Adresse aus der Gruppen-IP der Multicast-Gruppe ab (siehe hierzu Abschnitt A.1.5).

- Source MAC:
  Beinhaltet bei jedem Senden die physikalische Adresse (MAC-Adresse) des Absenders.
- EtherTyp:
  Das MAC-Protokoll kann in seinem Datenbereich verschiedenste andere Protokolle inkludieren. In diesem Feld ist der spezifizierte Identifier des direkt nachfolgenden Protokolls anzugeben. In den meisten Fällen handelt sich um das IP-Protokoll (EtherTyp: 0x0800). Bei Protokollen, wie dem ARP-Protokoll, muss kein Routing der Nachrichten außerhalb des Subnetzes durchgeführt werden. Das IP-Protokoll ist damit nicht notwendig, sodass das ARP-Protokoll direkt im MAC-Frame inkludiert wird. Der EtherTyp ist hier 0x0806.
- Daten:
  Beinhaltet weitere Protokolle und Nutzdaten.

A.1.2. IP-Protokoll

Der Aufbau des IP-Protokolls ist dem Bild A.3 zu entnehmen. Mithilfe des IP-Protokolls erfolgt die Adressierung und damit das Routing von Ethernet-Paketen auf Netzwerk-Ebene. Spezifiziert

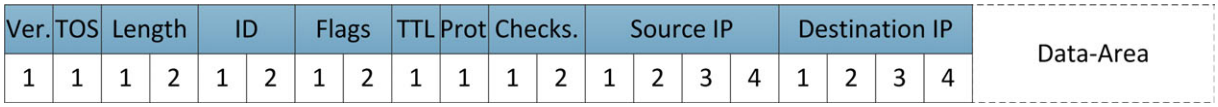


Bild A.3.: Aufbau des IP-Protokolls

sind folgende Datenfelder:

- Version:
  Gibt an, um welche Version es sich handelt. Der in Abschnitt 5.5 implementierte Ethernet-Stack unterstützt das IP-Protokoll in der Version 4, sodass hier 0x46 eingetragen sein muss, damit eine Verarbeitung korrekt durchgeführt wird. Handelt es sich um einen IP-Header mit Nutzung des zusätzlichen Options-Feldes (im Bild nicht dargestellt), wie es bei IGMP-Nachrichten benötigt wird, ist die Versionsnummer 0x45 zu verwenden.
- TOS:
  TOS steht für Type-Of-Service: Das Feld kann für die Priorisierung von IP-Datenpaketen genutzt werden.
- Length:
  Gibt die Gesamtlänge an, d. h. hier die Länge über Header- und Datenbereich.
- ID:
  Ist ein Identifier, der mit jeder Nachricht inkrementiert wird.

- Flags:
  Beinhaltet Informationen darüber, ob eine Nachricht fragmentiert wurde oder nicht und wenn dem so ist, an welcher Position das aktuelle Datenpaket einzufügen ist. Eine Fragmentierung von Nachrichten findet z. B. statt, wenn die Länge der Nachricht (theoretisch bis zu 65536 Byte) nicht vom Router bzw. dem Netzwerk unterstützt wird, d. h. die Nachrichtenlänge die sog. MTU-Größe übersteigt. Bei dem hier implementierten Design erfolgt der Einsatz in einem Netzwerk, bei dem Fragmentierungen ausgeschlossen werden können, da eine MTU-Größe von 1518 Byte pro Nachricht hier nicht überschritten wird. Ein Empfang von fragmentierten Nachrichten ist demnach nicht zu unterstützen, sodass diese Nachrichten verworfen werden.

- TTL:
  TTL bezeichnet die Time-To-Live. Die hier vom Sender angegeben Zahl wird mit jedem Hop, d. h. mit jeder Passierung eines Switch oder Router um die Zahl Eins dekrementiert. Ist die Zahl Null, erfolgt keine Weiterleitung der Nachricht. Ziel ist damit, dass Nachrichten nicht endlos im Netzwerk vorhanden sein können, wenn die Nachricht keinem Empfänger zuzuordnen ist. Bei IGMP Nachrichten wird dies genutzt, indem die TTL auf eins gesetzt wird. Es ist damit sichergestellt, dass nur der Switch, an den der Teilnehmer angebunden ist, die IGMP-Nachricht erhalten kann.

- Prot.:
  Hier wird das Protokoll in Form einer spezifizierten Nummer im nächsthöheren Layer angegeben. Hierzu zählen u. a. die realisierten Protokolle: ICMP (1), IGMP(2) und UDP(17).

- Checksumme:
  Die Checksumme wird hier nur über den IP-Header gebildet und nicht auch über den Datenbereich des IP-Protokolls. Bei der Generierung wird das Checksummen-Feld zu null gesetzt. Daraufhin wird das 16-Bit Einerkomplement der Einerkomplement-Summe aller 16-Bit-Blöcke gebildet. Das Ergebnis ist die Checksumme. Bei der Überprüfung wird die Checksumme nicht zu null gesetzt, sodass das Ergebnis der gleichen Berechnung Null ergeben muss.

- Source IP:
  Beinhaltet die IP-Adresse des Absenders

- Destination IP:
  Beinhaltet die IP-Adresse des Empfängers

- Optionsfeld:
  Dieses Feld kann optional genutzt werden, um zusätzliche Informationen zu übertragen. Diese Möglichkeit wird bei IGMP-Nachrichten genutzt. Durch die zusätzlichen Angaben wird in diesem Fall mitgeteilt, dass die Nachricht von dem Switch selbst auszuwerten ist.

A.1.3. UDP-Protokoll

Das verbindungslose UDP-Protokoll nutzt kein Handshake-Verfahren wie beim TCP-Protokoll und kann daher die Kommunikation auf einfachere Weise durchführen. Dem Bild A.4 ist das UDP-Protokoll zu entnehmen. Folgende Felder sind definiert:

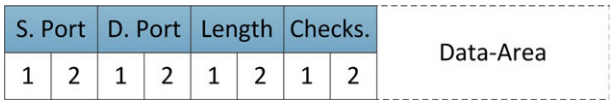


Bild A.4.: Aufbau des UDP-Protokolls

- Source Port:
  Gibt die Port-Nummer an, unter der der Absender die Nachricht gesendet hat. Anhand der Portnummer kann ein Netzwerk-Teilnehmer eine Nachricht einer Applikation zuordnen.
- Destination Port:
  Gibt die Port-Nummer der Ziel-Applikation an. Portnummern, die beim Empfänger nicht durch eine Applikation genutzt werden, werden verworfen.
- Length:
  Gibt die Gesamtlänge an, d. h. hier die Länge über Header- und Datenbereich.
- Checksumme:
  Die Berechnung und Überprüfung der Checksumme findet äquivalent zum IP-Protokoll statt. Anders als beim IP-Protokoll sieht die Standardisierung hier vor, dass das Checksummen-Feld unbenutzt bleiben darf. In diesem Fall ist als Checksumme der Wert 0x0000 einzutragen. Der Empfänger weiß in diesem Fall, dass die Checksumme nicht zu überprüfen ist. Dies spart Rechenzeit auf Seite des Senders und des Empfängers (siehe hierzu das Checksum-Offloading in Abschnitt 3.3).

A.1.4. ARP-Protokoll

Das “Address Resolution Protokoll” (ARP) ist notwendig, damit innerhalb eines Subnetzes die physikalische Adresse, d. h. die MAC-Adresse der Teilnehmer erfragt werden kann. Dies ist erforderlich, da die Adressierung in einem Subnetz nicht über die IP-Adresse erfolgt, sondern über die MAC-Adresse. Die IP-Adresse ist nur notwendig, um Nachrichten einem Netzwerk zuzuordnen und daraufhin das Routing der Nachricht zu diesem Netzwerk durchzuführen. Das ARP-Protokoll erfolgt damit direkt nach dem MAC-Frame. Das IP-Protokoll ist für das Routing hier nicht wichtig. ARP-Request und ARP-Replys werden an die Broadcast-MAC-Adresse verschickt. Bild A.5 zeigt den grundlegenden Aufbau des ARP-Protokolls. Die Datenfelder sind

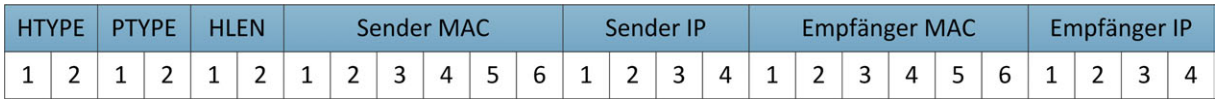


Bild A.5.: Protokoll-Aufbau des Adress-Resolution-Protokolls

folgende:

- HTYPE: Beschreibt, um welchen Typ von MAC-Adresse es sich handelt. Bei Ethernet ist der Typ immer 1.
- PTYPE: Enthält den Protokoll-Typ, für den die MAC-Adresse benötigt wird. In diesem Fall IPv4-Nachrichten, daher steht hier immer 0x0800.

- HLEN: Auch diese Feldangabe ist für den hier benötigten Anwendungsfall immer konstant. Angeben wird hier die Länge einer MAC-Adresse in Byte und damit der Wert 6.
- Sender MAC: Unabhängig davon, ob es sich um ein ARP-Request oder ein ARP-Reply handelt, muss die MAC-Adresse des Absenders der Nachricht hier eingetragen sein.
- Sender IP: Analog zur MAC-Adresse muss hier der Eintrag der IP-Adresse des Absenders erfolgen.
- Empfänger MAC: Bei einem ARP-Request ist der Inhalt nicht von Bedeutung, da hier die MAC-Adresse angefragt werden soll. Das Feld bleibt hier damit frei. Beim ARP-Reply muss die MAC-Adresse des Empfängers, d. h. von dem Teilnehmer eingetragen werden, welcher den ARP-Request gestellt hat.
- Empfänger IP: Beim ARP-Request wird hier die IP-Adresse desjenigen Teilnehmers eingetragen, für den die MAC-Adresse erfragt wird. Netzwerk-Teilnehmer können nur an diesem Eintrag überprüfen, ob auf diesen ARP-Request mit einem ARP-Reply zu antworten ist, weil das Feld die eigene IP-Adresse beinhaltet. Beim ARP-Reply muss die IP-Adresse des Teilnehmers eingetragen werden, der den ARP-Request gestellt hat.

A.1.5. IGMP-Protokoll

Das IGMP-Protokoll wird benötigt, um die Mitgliedschaften in einer Multicast-Gruppe zu verwalten. Die Verwaltung der Mitgliederliste findet dabei durch die Switches statt. Diese senden beim IGMP-Snooping in äquidistanten Abständen IGMP-Membership-Requests an alle Teilnehmer. Die Teilnehmer müssen hierauf mit einem IGMP-Membership-Report antworten, wenn diese die Mitgliedschaft in einer Gruppe behalten wollen. Asynchron hierzu kann ein Netzwerkteilnehmer IGMP-Membership-Reports zur An- und Abmeldung verschicken.
Das Verschicken und Empfangen von IGMP-Nachrichten erfordert Flexibilität in den IP-Protokoll-Modulen und im MAC-Frame. Abhängig vom IGMP-Nachrichten-Typ variiert die Ziel-MAC und Ziel-IP Adresse. Membership-Reports die zur Anmeldung an eine Multicast-Gruppe oder bei einer zyklischen Anfrage des Switches zu senden sind, müssen an die Multicast-Gruppen-Adresse gerichtet werden, an die sich angemeldet wird bzw. bei der die Mitgliedschaft zu bestätigen ist. Beim Verlassen einer Multicast-Gruppe muss die hierfür reservierte Gruppen-Adresse 224.0.0.2 verwendet werden. Membership-Querys, die ausschließlich vom Switch versendet werden, sind immer an die IP-Adresse 224.0.0.1 gerichtet. Die Filter im Ethernet-Stack und Ethernet-MAC sind hierauf zu konfigurieren. Aus den Multicast-Gruppen-Adressen leiten sich direkt die MAC-Adressen ab. Dies geschieht, indem die untersten 23 Bit der Multicast-Gruppen-Adresse in die untersten Bits der MAC-Adresse 01-00-5E-00-00-00 eingesetzt werden.
Neben der hierfür notwendigen Flexibilität des Ethernet-Stacks auf verschiedenen MAC- und IP-Adressen senden und empfangen zu können, müssen zudem im IP-Protokoll Anpassungen vorgenommen werden. Der Type-of-Service muss auf 0x00 und die Time-To-Live ist auf 0x01 gesetzt werden. Im Protokoll-Feld ist das IGMP-Protokoll mit 0x02 anzugeben. Zusätzlich ist das Optionsfeld zu nutzen, indem hier 0x94040000 eingetragen wird. Dem Router/Switch wird so mitgeteilt, dass die Nachricht vom Router/Switch selbst auszuwerten ist. Der Aufbau des IGMP-Protokolls in der Version 2 ist dem Bild A.6 zu entnehmen. Die Datenfelder sind folgende:

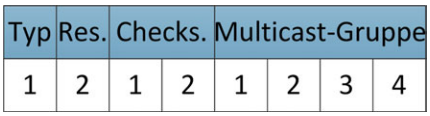


Bild A.6.: Protokoll-Aufbau des Internet-Group-Management-Protokoll

- Typ:
  Gibt den Typ der Nachricht an. Bei IGMP-Membership-Querys 0x11, bei IGMP-Membership-Reports zur Anmeldungen 0x16 und zur Abmeldung 0x17.
- Res.:
  Gibt die Response-Time an. Diese ist nur in IGMP-Membership-Querys von Bedeutung. Sie gibt an, wie lange die Antwort auf ein IGMP-Membership-Querys dauern darf. Die Auflösung ist 0.1 Sekunden. Der Switch meldet die angemeldeten Teilnehmer von der Multicast-Gruppe ab, wenn diese nicht in dem vorgegebenen Zeitfenster antworten. Eine Abmeldung bedeutet, die Nachrichten werden nicht mehr zu dem entsprechenden Teilnehmer geroutet.
- Checks.:
  Beinhaltet die Checksumme zur Absicherung des Protokolls. Diese wird analog zum IP- und UDP-Protokoll berechnet.
- Multicast Adresse:
  Beinhaltet die Multicast-Gruppenadresse, die einer IP-Adresse aus einem für Multicast-Adressen reservierten Adress-Bereich entspricht. Ist nur bei Membership-Reports und damit beim An-/Abmelden von einer Gruppe oder der Bestätigung einer Mitgliedschaft von Interesse. Bei Membership-Query wird das Feld freigelassen, wenn es sich um eine allgemeine Anfrage (General Membership Query) handelt. Handelt es sich um eine gruppenspezifische Anfrage, wird auch hier die Adresse der Gruppe vom Switch eingetragen.

A.1.6. ICMP-Protokoll

Das “Internet Control Message Protocol” wird genutzt, um Informations- und Fehlermeldungen in einem Netzwerk auszutauschen. Dieses Protokoll kann z. B. von einem Router verwendet werden, wenn dieser Pakete verwirft, weil etwa die Time-To-Live abgelaufen ist. Das Protokoll wird jedoch auch für Echo-Request und -Replys verwendet, allg. auch als Ping-Funktion bekannt. Ein Teilnehmer kann mittels Echo-Request ein Ping senden. Der Empfänger muss mit einem Echo-Reply antworten, sodass der anfragende Teilnehmer eine Information über die Erreichbarkeit und die Reaktionszeit erhält. Der Aufbau des Protokolls ist dem Bild A.7 zu entnehmen.

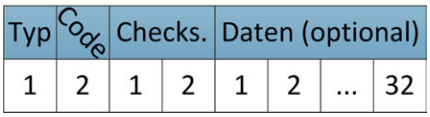


Bild A.7.: Protokoll-Aufbau des Internet-Control-Message-Protokoll

- Typ:
  Gibt an, um welche Art von Nachricht es sich handelt. Im Falle eines Echo-Request ist die “8” anzugeben, im Falle eines Echo-Replys die “0”.
- Code:
  Das Code-Feld wird für Echo-Requests und -Replys nicht benötigt. Für das Feld ist hier immer eine “0” einzutragen.
- Checksum:
  Beinhaltet die Checksumme zur Absicherung des Protokolls. Diese wird analog zum IP- und UDP-Protokoll berechnet.
- Daten (optional):
  Setzt sich im Fall von Echo-Requests und -Replys aus weiteren Datenfeldern zusammen. Hier Identifier, Sequenz-Nummer und 32 Byte zufällige Daten. Das Datenfeld muss in einem Echo-Reply identisch zum zugehörigen Echo-Request sein.

A.2. Herleitungen

Im Folgenden werden Herleitungen durchgeführt, bei denen im Hauptteil dieser Arbeit nur die Endergebnisse angegeben worden sind.

A.2.1. Bestimmung von Rundungsfehler bei der Pulsberechnung

Im Folgenden wird die Herleitung zur Berechnung des sich ergebenden Fehlers bei der Berechnung der ACP-Pulse aufgrund von Rundungsfehlern durchgeführt.
Es gilt für die Berechnung der Zeit zwischen zwei ACP-Pulsen die Anwendung der Gleichung A.1.

$$t_{ACP} = \frac{t_{dif}}{w_{dif}} \cdot w_{min} \tag{A.1}$$

Auf Basis der Anforderungen aus Abschnitt 1.1 können die Intervalle ermittelt werden, indem sich die Größen t_{dif}, w_{dif} und w_{min} befinden können. Es gilt Gleichung A.2 für die Periodendauer und Gleichung A.3 für die Winkeldifferenz zwischen zwei zu generierenden ACP-Pulsen. Dabei ist F die Pulsfolgefrequenz in Hertz und U die Zeit in Sekunden, die für eine Umdrehung pro Umlauf benötigt wird.

$$t_{dif} = \frac{1}{F} \tag{A.2}$$

$$w_{dif} = \frac{65536}{F \cdot U} \tag{A.3}$$

Nach den Anforderungen kann die Zeit, die für eine Antennenumdrehung benötigt wird, im Intervall $[0.75\,s;\ 3\,s]$ liegen. Damit ergibt sich Gleichung A.4, die den Bereich angibt, in dem der Quotient aus Gleichung A.1 liegen kann.

$$11.44 \cdot 10^{-6} \leq \frac{t_{dif}}{w_{dif}} = \frac{U}{65536} \leq 45.78 \cdot 10^{-6} \tag{A.4}$$

Für w_{min} gilt, dass sich der größte Wert bei der geringsten Anzahl zu generierender Puls pro Umlauf ergibt. Damit folgt Gleichung A.5.

$$w_{min} \leq \frac{65536}{75} = 873.813 \tag{A.5}$$

Im Folgenden wird nun der sich maximal mögliche Fehler aufgrund der Nutzung von Festkommazahlen berechnet. Dies geschieht unter verschiedenen Skalierungen der Einzelwerte, um die Genauigkeit zu erhöhen.

Keine Skalierung : Die Zeitdifferenz wird in Mikrosekunden-Auflösung zur Verfügung gestellt. Der Winkel w_{min} wird zur Laufzeit berechnet, indem $w_{min} = 65536/P$ gerechnet wird, mit P der Anzahl der gewünschten Pulse pro Antennenumdrehung. Der Winkelfehler kann daher maximal $0.\overline{9}$ Winkeleinheiten betragen. Das Ergebnis der Division kann maximal den Wert 45.776 annehmen, sodass in der Multiplikation mit dem Winkelfehler ein Zeitfehler von maximal $t_{f1} < 45.776\,\mu s$ entsteht. Ein weiterer Fehler entsteht, wenn die durchgeführte Division von t_{dif} und w_{dif} im Ergebnis Nachkommastellen aufweist. Maximal liegt der hierdurch entstehende Fehler bei $0.\overline{9}$. Multipliziert mit w_{min} entsteht ein Zeitfehler von bis zu $t_{f2} < 873.813\,\mu s$. Wird keine Skalierung durchgeführt, ergibt sich daraus ein Fehler durch die Berechnung von $t_f < 919.59\,\mu s$. Der Einzelfehler kann ein zeitliches Jittern in der Puls-Ausgabe verursachen, welcher nicht die Anforderungen erfüllt.

Skalierung von t_{dif} um Faktor 1000: Der Zwischenfehler t_{f1} steigt auf $t_{f1} < 45776\,\mu s$ an, während t_{f2} konstant beleibt. Damit folgt unter Beachtung der Rückskalierung ein Einzelfehler von $t_f < 46.65\,\mu s$. Auch hier werden die Anforderungen nicht erfüllt.

Skalierung von t_{dif} und w_{min} um Faktor 1000: Es wird nun auch die Genauigkeit von w_{min} durch Skalierung auf drei Nachkommastellen genau angeben. Der Fehler t_{f1} bleibt unverändert, der Fehler t_{f2} steigt hingegen auf $t_{f1} < 873813\,\mu s$ an. Durch Rückskalierung folgt ein Einzelfehler von $t_f < 919.59\,ns$. Bei der dieser Skalierung werden damit die Anforderungen erfüllt.

Skalierung von t_{dif} um Faktor 1000 und w_{min} um Faktor 100: Der Fehler t_{f1} fällt auf $t_{f1} < 87381.3\,\mu s$. Der Einzelfehler ergibt sich dann mit $t_f < 1.331\,\mu s$. Die Anforderungen werden auch hier erfüllt.

A.2.2. Bestimmung des zulässigen Zahlenbereiches

In Abschnitt 6.3.2 ist die Gleichung A.6 zur Bestimmung der Ausgabezeitpunkte des ACP-Pulses angegeben. Für diese Berechnung wurden Gleichungen vorgegeben, die beschreiben, in welchem Zahlenbereich sich die Anzahl der ACP-Pulse pro Umlauf, die Antennendrehgeschwindigkeit und die Pulsfolgefrequenz befinden müssen, damit es zu keinen Überläufen bei der Berechnung kommen kann. Diese Gleichungen sollen im Folgenden hergeleitet werden. Die Herleitung bezieht sich dabei auf eine Skalierung von t_{dif} mit 1000 und w_{min} mit 100, so wie sie auf dem MicroBlaze zum Einsatz kommt. Andere Skalierungen und Bitbreiten lassen sich äquivalent herleiten.

$$t_{ACP} = \frac{\frac{t_{dif}}{w_{dif}} \cdot (w_{min})}{100000} \tag{A.6}$$

In Gleichung A.6 ist t_{dif} in Mikrosekunden angegeben. Die Auflösung von w_{min} liegt bei $1/65536$.
Die Bitbreite eines Produktes ist gleich der Summe der Bitbreiten der Operanden. Hingegen ist die Bitbreite der Quotienten die Differenz der Bitbreiten der Operanden. Gleichung A.7 gibt daher die Berechnung der Bitbreite zu Gleichung A.6 an.

$$\log_2 t_{ACP} = (\log_2 t_{dif} - \log_2 w_{dif}) + log_2 w_{min} - log_2 100000 \tag{A.7}$$

Um nun anstelle von Winkel- und Zeitdifferenzen eine Beschreibung mittels allgemeingültigeren Größen wie Antennendrehgeschwindigkeit, Anzahl an ACP-Pulsen pro Antennenumdrehung und Pulsfolgefrequenz zu erlauben, werden im folgenden die Gleichungen zur Umsetzungen der Variablen und damit der Bitbreiten hergeleitet.
Der Zusammenhang zwischen der Bitbreite der Periodendauer t_{dif} (in Mikrosekunden) und der Pulsfolgefrequenz F (in Hertz) kann durch Gleichung A.8 beschrieben werden.

$$t_{dif} = \frac{1}{F} \cdot 1000000000 \Rightarrow \log_2 t_{dif} = log_2 1000000000 - log_2 F = 29.89 - log_2 F \tag{A.8}$$

Die Differenz zwischen zwei zu generierenden ACP-Pulsen wird durch w_{min} in Gleichung A.6 angegeben. Wenn $2^{16} \hat{=} 360^\circ$ entspricht und P die Anzahl an ACP-Pulsen pro Umdrehung ist, gilt Gleichung A.9.

$$w_{min} = \frac{65536}{P} \cdot 100 \Rightarrow \log_2 w_{min} = 22.644 - \log_2 P \tag{A.9}$$

Die mittlere Winkeldifferenz zwischen zwei Sweeps kann durch Gleichung A.10 beschrieben werden, wenn U die Anzahl an Antennenumdrehung pro Minute und F wieder die Pulsfolgefrequenz in Hertz ist.

$$w_{dif} = \frac{U}{60} \cdot \frac{1}{F} \cdot 65536 = \frac{U}{F} \cdot 1092.27 \Rightarrow log_2 w_{dif} = log_2 U - log_2 F + 10.093 \tag{A.10}$$

Wird Gleichung A.8, A.9 und A.10 in Gleichung A.7 eingesetzt, folgt Gleichung A.11 wenn gilt, dass $\log_2 t_{ACP} \leq 32$ ist.

$$10.448 \leq log_2 U + log_2 P \tag{A.11}$$

Die Frequenz hat sich hierbei herausgekürzt. Nicht berücksichtigt wird die Entskalierung. Diese senkt die benötigte Bitbreite des Endergebnisses wieder ab. Interessant für die Bitbreite ist daher das Zwischenergebnis vor der Entskalierung. Die minimale Frequenz ist durch die Skalierung von t_{dif} gegeben und das hieraus entstehende Zwischenergebnis. Die Grenze der Frequenz ist daher durch Gleichung A.12 gegeben.

$$\log_2 t_{dif} + \log_2 1000 \leq 32 \overset{Gl.A.8}{\Rightarrow} \log_2 F \geq 2.103 \Rightarrow F \geq 0.233\,Hz \tag{A.12}$$

A.3. CD-Anhang

Alle im Folgenden aufgeführten Anhänge sind in elektronischer Form auf einer CD abgelegt. Diese kann beim Erstprüfer Herrn Prof. Dr. rer. nat. Jürgen Reichardt eingesehen werden.

A.3.1. Dokumentationen

Im Folgenden sind verschiedene Dokumentationen aufgelistet, die im Rahmen dieser Arbeit erstellt wurden und auf dem beiliegenden Datenträger abgelegt sind.

Entwicklungs-Dokumentation:
Dokumentiert das Design. Es sind u. a. Angaben zum proprietären Ethernet-Video-Protokoll, die Schnittstelle zum DAC und die Schnittstelle zwischen MicroBlaze und Hardware-Design mit allen implementierten Registern dokumentiert.
Dateipfad auf Datenträger: dokumente/entwicklungsDokumentation.pdf

WRI Connect Tool Dokumentation:
In diesem Dokument sind die einzelnen Funktionen des Test- und Steuerprogramms aus Abschnitt 7.1 detaillierter erläutert.
Dateipfad auf Datenträger: dokumente/connectTool.pdf

Bericht zu den Voruntersuchungen:
Beinhaltet in detaillierter Form die Ergebnisse der Voruntersuchungen aus Kapitel 3.
Dateipfad auf Datenträger: dokumente/voruntersuchung.pdf

A.3.2. Projekt-Dateien

Im Folgenden sind verschiedene "Xilinx Design Navigator"-Projekte angegeben, die im Rahmen dieser Arbeit erstellt worden sind.

Hauptprojekt
Projekt beinhaltet das vollständige Projekt, wie es für die Verarbeitung der Radar-Informationen notwendig ist.
Dateipfad auf Datenträger: projekte/finalProject/

Projekt-Subsets:
Es sind verschiedene Projekt-Subsets zur Verwendung für die "HAW Hamburg" und die "FH Westküste" erstellt worden, die auf dem Datenträger abgelegt worden:

- Ethernet-Interface ohne MicroBlaze für ML507 Evaluationboard unter ISE 12.4
  Dateipfad auf Datenträger: projekte/subsets/ml507withoutMicroBlazeISE124/
- Ethernet-Interface mit MicroBlaze für ML507 Evaluationboard unter ISE 12.4
  Dateipfad auf Datenträger: projekte/subsets/ml507withMicroBlazeISE124/
- Ethernet-Interface ohne MicroBlaze für ML507 Evaluationboard unter ISE 13.2
  Dateipfad auf Datenträger: projekte/subsets/ml507withoutMicroBlazeISE132/
- Ethernet-Interface mit MicroBlaze für ML507 Evaluationboard unter ISE 13.2
  Dateipfad auf Datenträger: projekte/subsets/ml507withMicroBlazeISE132/
- Ethernet-Interface ohne MicroBlaze für SP605 Evaluationboard unter ISE 13.2
  Dateipfad auf Datenträger: projekte/subsets/sp605withoutMicroBlazeISE132/
- Ethernet-Interface mit MicroBlaze für SP605 Evaluationboard unter ISE 13.2
  Dateipfad auf Datenträger: projekte/subsets/sp605withMicroBlazeISE132/

- Ethernet-Interface ohne MicroBlaze für SP605 Evaluationboard unter ISE 13.2 (Tri-Mode Ethernet-MAC 100 MBit/s)
  Dateipfad auf Datenträger: projekte/subsets/sp605withoutMicroBlazeISE132_TriMAC100/
- Ethernet-Interface ohne MicroBlaze für SP605 Evaluationboard unter ISE 13.2 (Tri-Mode Ethernet-MAC 1000 MBit/s)
  Dateipfad auf Datenträger: projekte/subsets/sp605withoutMicroBlazeISE132_TriMAC1000/
- Ethernet-Interface Source-Code ohne Projekt
  Dateipfad auf Datenträger: projekte/subsets/otherFPGA/
- Ethernet-Interface ohne MicroBlaze für ML507 Evaluationboard unter ISE 13.2 (TEMAC, nur mit Zwischenversion des Ethernet-Stack realisiert)
  Dateipfad auf Datenträger: projekte/subsets/ml507withoutMicroBlazeISE132_TEMAC100/

A.3.3. Voruntersuchung

Im Rahmen der Voruntersuchungen zur Anbindbarkeit eines FPGAs an ein Ethernet-Netzwerk sind verschiedenste Projekte unterschiedlichster Konfigurationen erstellt worden. Alle Projekte sind mit der "ISE Design Suite" in der Version 12.4 erstellt worden. Auf dem beigefügten Datenträger sind hierzu folgende Inhalte abgelegt worden:

Projekte mit den verschiedenen Systemkonfigurationen:
Dateipfad auf Datenträger: voruntersuchung/projekte/

Rohdaten der durchgeführten Messungen:
Dateipfad auf Datenträger: voruntersuchung/rohdaten/

Checksum-Offloading
Ordner beinhaltet alle Dateien, die angepasst werden mussten, damit das Checksum-Offloading auch für UDP-Nachrichten in Sende- und Empfangsrichtung genutzt werden kann.
Dateipfad auf Datenträger: voruntersuchung/offloading/

Iperf-Tool
Tool zur Durchführung der Iperf-Tests.
Dateipfad auf Datenträger: voruntersuchung/iperf/

A.3.4. Test-Software

Die entwickelte Testsoftware ist dem CD-Anhang unter folgenden Pfadangaben zu entnehmen. Enthalten sind neben den Sourcen auch die Setup-Dateien:

Projekt-Dateien für C#-Testprogramm:
Dateipfad auf Datenträger: projekte/csharp/wrict/

Projekt-Dateien für C#-CommandLine-Simulator:
Dateipfad auf Datenträger: projekte/csharp/RadarsimulatorCommandLine/

A.3.5. Tutorials

Im Folgenden sind verschiedene Tutorials aufgelistet, die im Rahmen dieser Masterarbeit erstellt wurden. Sie beschreiben das Vorgehen bei verschiedenen Themenstellungen, die im Zuge der Entwicklung erarbeitet worden sind.

Exportieren von Netzwerk-Traffic aus Wireshark zur Nutzung mittels der Testbench
Das Tutorial bezieht sich auf die Implementierung in Abschnitt 5.1.4.
Dateipfad auf Datenträger: tutorials/WiresharkExport.pdf

Nutzung des MicroBlaze unter Verwendung von Xilinx ISE Design Navigator
Das Tutorial bezieht sich auf die Implementierung des MicroBlaze in Abschnitt 5.2. Beschrieben wird die Einbindung des MicroBlaze als Komponente in ein HDL-Design, die Konfiguration des MicroBlaze, das verwendete Verfahren, um die FSL-Verbindung aus dem MicroBlaze-Design herauszuführen und wie externe Interrupt-Leitungen konfiguriert werden.
Dateipfad auf Datenträger: tutorials/MicroBlazeimISE.pdf

Profiling
Beschreibt die Durchführung eines Software-Profiling mittels des "Software Development Kit". Das Tutorial bezieht sich auf die Untersuchungen in Abschnitt 7.2.1.
Dateipfad auf Datenträger: tutorials/Profiling.pdf

PlanAhead
Beschreibt die Erstellung eines Projektes unter PlanAhead auf Basis eines existierenden ISE-Projektes, die Durchführung eines Design-Flows, die Timing- und Ressourcen-Analyse, sowie das Einrichten und Verwalten von Partitionen, um einen inkrementellen Designflow durchführen zu können.
Dateipfad auf Datenträger: tutorials/PlanAhead.pdf

A.3.6. Batch-Skripte

Im Folgenden sind verschiedene Bach-Skripte hinterlegt, die auf Command-Line-Ebene verschiedene Tools aufrufen und damit eine schnellere und fehlerfreiere Nutzung verschiedener Tools erlauben. Die Skripte sind im Rahmen dieser Masterarbeit erstellt wurden.

Design-Flow mit ISE auf Command-Line-Ebene
Dieses Skript führt einen vollständigen Design-Flow auf Command-Line-Ebene aus. Damit dieses Skript funktioniert, muss in der Batch-Datei der "XilinxPath" richtig gesetzt sein. Das Skript selbst muss im Root-Verzeichnis des ISE-Design-Projektordners liegen. Ggf. sind die Pfade zu den .ucf-Dateien anzupassen.
Dateipfad auf Datenträger: batchskripte/DesignFlow/run_xilinx_tools.bat

Erstellung von Flash-Images für das Fallback-Multiboot
Mit Hilfe der folgenden Skripte können .mcs-Files zur Beschreibung des externen Flash unter Verwendung der "Fallback-Multiboot"-Funktion erstellt werden.
Dateipfad auf Datenträger: projekte/finalProject/multiboot/

- *step0_getter_ise.bat*
  Kopiert .ncd-, .bmm- und .elf-Datei aus den Projekt-Ordnern
- *step1_generateBitGolden.bat*
  Generiert .bit-Datei für Golden-Image einschließlich Header-Image
- *step1_generateBitMultiboot.bat*
  Generiert .bit-Datei für Multiboot-Image
- *step3_copyGolden.bat*
  Speichert generierte .bit-Datei als Golden-Image
- *step3_copyMultiboot.bat*
  Speichert generierte .bit-Datei als Multiboot-Image
- *step4_promFull.bat*
  Erstellt .mcs-Datei mit vollständigem Flash-Image
- *step4_promUpdate.bat*
  Erstellt .mcs-Datei mit Flash-Image für Update (nur Multiboot-Image)

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Ammersbek, 15. Februar 2012

Ort, Datum

Unterschrift