Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Jan Jasper Salathé

FPGA basiertes HW/SW Codesign eines Frameworks für TSN Switches

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jan Jasper Salathé

FPGA basiertes HW/SW Codesign eines Frameworks für TSN Switches

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Franz Korf
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Eingereicht am: 05. Februar 2016

Jan Jasper Salathé

Titel dieser Masterarbeit
FPGA basiertes HW/SW Codesign eines Frameworks für TSN Switches

Stichworte
TSN, Time Sensitive Networking, RT-Ethernet, Real-Time Ethernet, AFDX, TTE, Timetriggered-Ethernet, AVB, Fahrzeugnetze, FPGA, HW/SW-Codesign, NetFPGA, Prototyping

Kurzzusammenfassung
Die aktuelle Entwicklung neuer Anwendungen wie Advanced Driver Assistance Systems im Automobil zeigen, dass zukünftige Fahrzeugnetze einen steigenden Bedarf an Bandbreite aufweisen werden, während Fahrzeugnetze ein weiteres Spektrum an Echtzeitanforderungen erfüllen müssen. Da gegenwärtig verwendete Technologien in absehbarer Zeit diesen Anforderungen nicht mehr gerecht werden, stellt Real-Time Ethernet einen geeigneten Nachfolger dar. Das weite Spektrum an Echtzeitanforderungen erfordert die Kombination mehrerer Konzepte für Real-Time Ethernet. Time Sensitive Networking (TSN) stellt eines dieser Konzepte dar. Diese Arbeit befasst sich mit der Entwicklung eines Frameworks für TSN Switches. Ziel ist es, in Zukunft über ein Framework zur flexiblen Kombination mehrerer Konzepte für Real-Time Ethernet zu einem Prototypen zu verfügen und daran zukünftige Fahrzeugnetze zu evaluieren. Das Framework wurde auf Basis des NetFPGA 1G Boards umgesetzt.

Title of the master thesis
FPGA based HW/SW Codesign of a Framework for TSN Switches

Keywords
TSN, Time Sensitive Networking, Real-Time Ethernet, RT-Ethernet, AFDX, TT-Ethernet, AVB, in-car networks, FPGA, HW/SW-Codsign, NetFPGA, prototyping

Abstract
New applications like Advanced Driver Assistance Systems show an increasing demand for bandwidth in future in-car networks, as real-time requirements become more diverse. While current technology will fail to fulfill these demands in foreseeable future, Real-Time Ethernet will satisfy them. The increasing diversity of demands for real-time communication requires the combination of different concepts for Reat-Time Ethernet. Time Sensitive Networking (TSN) is one of them. In this thesis, a framework for prototypes of TSN Switches is developed. This framework allows the flexible combination of multiple Real-Time Ethernet concepts to build and evaluate prototypes of future in-car networks. It is based on the NetFPGA 1G Board.

Inhaltsverzeichnis

1 Einleitung **1**

2 Ist-Analyse **4**
2.1 IEEE 802.3 Ethernet . . . 5
2.2 Computersysteme mit Echtzeitanforderungen . . . 6
2.3 RT-Ethernet Konzepte . . . 8
2.4 Verwandte Arbeiten . . . 17
2.5 Zielsetzung dieser Arbeit . . . 23

3 Konzept **26**
3.1 Pre Processing . . . 29
3.2 Input Arbiter . . . 29
3.3 Filtering Database . . . 30
3.4 Output Port Multiplexer . . . 31
3.5 Paket Buffering . . . 31
3.6 Post Processing . . . 33
3.7 Buffer Management . . . 34
3.8 Eingebettetes Prozessorsystem . . . 34

4 Umsetzung **36**
4.1 Taktgenerierung . . . 36
4.2 Paketempfang und -versand . . . 48
4.3 Paketforwarding . . . 57
4.4 Paketpufferung . . . 64
4.5 Puffermanagement . . . 80
4.6 Eingebettetes Prozessorsystem . . . 88
4.7 Software . . . 95

5 Qualitätssicherung **102**
5.1 Vorgehen zur Sicherung der Qualität . . . 102
5.2 Durchgeführte Maßnahmen an der Komponente Packet Analyzer . . . 104

6 Evaluation **108**
6.1 Messaufbau . . . 108
6.2 Paketlatenzen . . . 113
6.3 Einfluss auf die TTE Zeitsynchronisation . . . 119

6.4 Scheduling von TT Nachrichten 121
6.5 Minimale Latenz von TT Nachrichten 123

7 Zusammenfassung **125**

Literatur **128**

Abbildungsverzeichnis

2.1 Queues eines ausgehenden Ports bei IEEE 802.1Q mit 4 Paketklassen 9
2.2 Queues eines ausgehenden Ports bei IEEE 802.1Q mit 4 Paketklassen, davon 2 AVB Paketklassen 10
2.3 Exemplarischer Verlauf des Credits bei einem CBS mit den aktuell gesendeten Paketen 11
2.4 Vermittlung eines TT-Pakets mit interferierendem BE-Paket 13
2.5 Queues, TSAs, Gates und Arbiter eines ausgehenden Ports bei TSN mit Scheduled Traffic 14
2.6 Beispielhafte Übermittlung von Cyclic Queueing (CQ) Traffic mit interferierenden BE-Paketen 15
2.7 Beispielhaftes UBS Netzwerk mit den Streams S_1 und S_2[50] 16
2.8 Queues der UBS Paketklasse für drei eingehend und ausgehende Ports[53] . . 17
2.9 Komponenten auf dem NetFPGA 1G Board 20
2.10 Übersicht über die NetThreads Prozessorkerne [35] 21
2.11 Übersicht über die Network Code Switch Prozessorkerne [4] 22

3.1 Übersicht über die entwickelte Architektur der RT-Bridge 27
3.2 Komponenten der Referenzumsetzung einer Ethernet Bridge im NetFPGA (Reference Bridge Projekt) 27

4.1 Signale des RGMII Interfaces 37
4.2 Paketbeginn bei 1000 $\mathrm{Mbit/s}$ RGMII Signalisierung 38
4.3 Setup- und Hold-Zeiten für RGMII Signalisierung 38
4.4 Paketbeginn bei 100 $\mathrm{Mbit/s}$ und 10 $\mathrm{Mbit/s}$ RGMII Signalisierung 39
4.5 Ressourcen zum Bereitstellen von Taktsignalen auf Virtex II Prio 50 FPGA des Unternehmens Xilinx (nach [29, 65]) 40
4.6 Ursprüngliche Komponenten zur Taktgenerierung und Anbindung des RGMII 42
4.7 Angepasste Komponenten zur Taktgenerierung und Anbindung des RGMII . . 45
4.8 Verhalten des Zählers zum Generieren von *txc* 46
4.9 Taktquellen auf dem NetFPGA Board 47
4.10 Signale des im NetFPGA verwendeten Interfaces zur Übermittlung von Paketen 49
4.11 Beispielhafte Paketübermittlung mit dem im NetFPGA verwendeten Interface zur Paketübermittlung 49
4.12 Beispielhafte Paketübermittlung mit dem in der RT-Bridge verwendeten Interface zur Paketübermittlung (*data* Signal enthält Index des Paketworts) . . . 50
4.13 Mealy-Zustandsautomat der Buffer Sink 52

4.14 Mealy-Zustandsautomat der Buffer Source . . . 54
4.15 Mealy-Zustandsautomat des VLAN Taggers . . . 54
4.16 Berechnung des aktualisierten *transparentClock* Felds in CTRL Exec . . . 56
4.17 In CTRL Exec verwendeter Zustandsautomat . . . 57
4.18 Umsetzung des RR-Selectors . . . 63
4.19 Interfaces der Komponenten im Packet Buffering Modul . . . 65
4.20 Beispiel der Paketauswahl im Packet Buffering Modul . . . 67
4.21 Mealy-Zustandsautomat der FIFO Queue . . . 68
4.22 Rechenwerk des Queue Arbiters . . . 78
4.23 Mealy-Zustandsautomat zur Steuerung des Rechenwerks im Queue Arbiter . . . 79
4.24 Interface des Kontrollpfads des Buffer Managements . . . 81
4.25 Beispielhafte Zugriffe auf den Kontrollpfad des Buffer Managements . . . 82
4.26 Interface des Datenpfads des Buffer Managements . . . 83
4.27 Beispielhafte Zugriffe auf den Datenpfad des Buffer Managements . . . 83
4.28 Interface von je einem lesenden und schreibenden Endpunkt des SRAM Arbiters 84
4.29 Beispielhafte Zugriffe auf den SRAM-Arbiter . . . 85
4.30 Verarbeitungspipeline des Buffer Management Kontrollpfads . . . 86
4.31 Verarbeitungspipeline des Datenpfades des Buffer Managements . . . 87
4.32 Signale des Interfaces der Register Bridge . . . 90
4.33 Beispielhafte Zugriffe über das Interface der Register Bridge . . . 91
4.34 Signale des Interfaces zur Konfiguration der FDB . . . 91
4.35 Beispielhafte Schreibzugriffe auf die FDB . . . 92
4.36 Eingebettetes Prozessorsystem der RT-Bridge . . . 93
4.37 Zu korrigierende Offsets bei der Zeitsynchronisation . . . 99

5.1 Verarbeitung eines durch die Testbench für das Modul Pre Processing generierten Paketes . . . 105
5.2 Verarbeitung eines durch die Testbench für den User Data Path generierten Paketes . . . 107

6.1 Zur Evaluation verwendeter Messaufbau . . . 109
6.2 Validierung von *rx_dv* zur Messung von Paketeingängen . . . 112
6.3 Validierung von *tx_en* zur Messung von Paketausgängen . . . 112
6.4 Messung der Korrekturwerte für die Zeitstempeleinheiten bei eingehenden Paketen . . . 114
6.5 Messung der Korrekturwerte für die Zeitstempeleinheiten bei ausgehenden Paketen . . . 114
6.6 Bei der Latenzmessung von Paketen durchlaufene Komponenten und gemessene Latenzen . . . 115
6.7 Messung zur Bewertung des Einflusses auf die TTE Zeitsynchronisation ohne RT-Bridge . . . 120
6.8 Messung zur Bewertung des Einflusses auf die TTE Zeitsynchronisation mit RT-Bridge . . . 121

6.9 Messung der Präzision des Schedulings von TT Nachrichten 122
6.10 Messung des Einflusses von BE Nachrichten auf TT Nachrichten 123
6.11 Messung der minimalen Latenz bei der Weiterleitung von von TT Nachrichten 124

Tabellenverzeichnis

2.1 Struktur eines Ethernet Pakets . . . 5
2.2 Struktur eines Ethernet Pakets mit VLAN Tag . . . 9
2.3 Aufbau der PC-Frames von TTE [62] . . . 13

3.1 Aus IEEE 802.1Q [14, Kapitel 8.6.8] hervorgehendes Interface der TSAs . . . 31
3.2 Signale des Interfaces der TSAs . . . 32
3.3 Signale des Interfaces der Queues . . . 32

4.1 Quadranten der RGMII IO-Signale . . . 43
4.2 Felder des innerhalb der RT-Bridge verwendeten Paketheaders . . . 50
4.3 Felder der Paketversandevents (Beschreibungen siehe Tabelle 4.2) . . . 51
4.4 Felder der PC-Frames in Ethernet Paketen mit und ohne VLAN Tag [61] . . . 55
4.5 Belegung der Lookup Table VLAN Table . . . 61
4.6 Belegung der Lookup Table First Level Table . . . 61
4.7 Belegung der Lookup Table Second Level Table . . . 61
4.8 Belegung der Lookup Table Autolearning Table . . . 62
4.9 Aktionen in den Pipelinestufen des Kontrollpfads des Buffer Managements bei den unterschiedlichen Operationen . . . 86
4.10 Memory Map des eingebetteten Prozessorsystems . . . 96

6.1 Im Messaufbau verwendete Konfiguration der FDB . . . 109
6.2 Im Messaufbau verwendeter Schedule der TT Pakete in der RT-Bridge . . . 109
6.3 Über den Debugheader aus der RT-Bridge herausgeführte und für die Messungen verwendete Signale . . . 110
6.4 Für Evaluationsmessungen verwendeter Schedule in Netzwerkknoten . . . 111
6.5 Messergebnisse der Validierung von *rx_dv* und *tx_en* zur Messung von Paketein- und Paketausgängen . . . 113
6.6 In den Komponenten der RT-Bridge anfallende Latenzen . . . 116
6.7 Zur Abschätzung der Latenzen in der RT-Bridge verwendete Parameter . . . 116
6.8 Messergebnisse der Paketlatenzen für die Paketklasse AVB von PHY zu PHY . . . 118
6.9 Messergebnisse der Paketlatenzen für die Paketklasse AVB zwischen den Zeitstempeleinheiten . . . 118

1 Einleitung

Fahrerassistenzsysteme bieten in Zukunft das Potential die Zahl der Verkehrstoten durch eine Unterstützung des Fahrers zu reduzieren. In Fahrzeugen werden Fahrerassistenzsysteme (Advanced Driver Assistance Systems, ADAS) mittlerweile nicht mehr nur im gehobenen Preissegment integriert, sondern allgemein in immer mehr Fahrzeugen. So müssen Neuwagen in den USA ab Anfang 2018 mit Systemen zur besseren Einsehbarkeit des Bereiches hinter dem Fahrzeug ausgerüstet sein. Auch in Europa werden ADAS immer verbreiteter. In Zukunft soll die Integration von ADAS zu einer besseren Bewertung im Euro-NCAP (New Car Assessment Program) führen[41]. Folgende Systeme werden Beispielsweise schon in Fahrzeuge integriert oder wird es in Zukunft geben[6]:

- Fahrspurassistenzsysteme
- Notbremssysteme mit Erkennung von Fußgängeren oder drohenden Auffahrunfällen
- Abstandsregeltempomaten
- Verkehrszeichenerkennung
- Einparkhilfen

Die genannten ADAS benötigen Sensoren wie RADAR (Radio Detection And Ranging), Kameras und LIDAR (Light Detection And Ranging)[7]. Für kamerabasierte Einparkhilfen ist zumindest eine Auflösung von 1280 x 720 Pixeln bei 15 bis 30 $\mathrm{Bilder/s}$ nötig[6]. Entsprechend hoch ist die Bandbreite der zu vermittelnden Daten. Die aktuell in Fahrzeugen verbreiteten Netzwerktechnologien wie CAN (Controller Area Network) und FlexRay sind nicht in der Lage, entsprechende Bandbreiten zu übertragen. Somit werden aktuell unter anderem auf LVDS (Low Voltage Differential Signaling) basierende Punkt-zu-Punkt Verbindungen verwendet.

Die Einführung neuer elektrischer Systeme ist ein Trend, der seit längerem zu beobachten ist. Der Trend begann in den 70er Jahren mit Tempomaten, in den 80ern folgten Antiblockiersysteme und ist bis heute mit den genannten ADAS nachzuvollziehen. Dieser Trend führt dazu, dass in Fahrzeugen des oberen Preissegmentes aktuell bis zu 140 Steuergeräte (Electronical Control Units, ECU) verbaut sind. Um die einzelnen Komponenten miteinander zu verbinden, kommt ein in seiner Komplexität und seinem Umfang wachsender Kabelbaum zum Einsatz. So erreicht der Kabelbaum aktuell durchschnittlich ein Gewicht von 100 kg[43]. Die steigende

Komplexität führt außerdem dazu, dass zur Optimierung des Kabelbaums bezüglich Gewicht und Leitungslänge zunehmend softwarebasiertes, automatisches Routing eine Rolle spielt[37].

Mit IEEE 802.3 Ethernet [18] steht eine auf dem Markt etablierte Technologie zur Verfügung, die die Bandbreite für die kommenden ADAS bereitstellen kann. Im Gegensatz zu Bussystemen wie CAN und FlexRay lässt sich bei Ethernet die Bandbreite nach Bedarf skalieren. So sind physikalische Schnittstellen (Physical Layer, PHY) mit $10\ \mathrm{Mbit/s}$ bis $40\ \mathrm{Gbit/s}$ standardisiert, $400\ \mathrm{Gbit/s}$ PHYs werden in IEEE 802.3bs standardisiert [21]. Weiterhin wird mit IEEE 802.3bp [19] die erforderliche Anzahl von Leitungspaaren für Gigabit Ethernet reduziert und die Resistenz gegenüber externen Störquellen verbessert. Die Special Interest Group OPEN (One-Pair Ether-Net) Alliance [42] beschäftigt sich ebenfalls mit der Reduzierung der Leitungspaare für Ethernet auf ein Leitungspaar. Für Systeme im Automobil bestehen teilweise harte Anforderungen an die Echtzeitfähigkeit der Kommunikationsstrecken. Die zu übertragenden Daten müssen innerhalb einer gegebenen Latenz garantiert verlustfrei übertragen werden und es muss eine definierte Übertragungsqualität gewährleistet werden (Quality of Service, QoS). Kommt es beispielsweise bei der Übertragung von Kommunikationsdaten eines Notbremssystemes zu Verlusten, so führt dies zu unvorhersehbarem Verhalten des Fahrzeuges, im schlimmsten Fall zu Todesfällen. Zwar wurden Ethernet mit IEEE 802.1Q [14] um grundlegende QoS Eigenschaften erweitert, jedoch können Pakete bei Überlastsituationen unkontrolliert verzögert und verworfen werden. Daher erfüllt Ethernet diese Echtzeitanforderungen nicht.

Um die Echtzeitfähigkeit von Ethernet (Realtime-Ethernet, RT-Ethernet) zu erreichen existieren unterschiedliche Konzepte, wie Time-Triggered Ethernet (TT-Ethernet) und Audio-/Video-Bridging (AVB). Diese sind bereits kommerziell verfügbar während sich mit Time Sensitive Networking (TSN) der Nachfolger von AVB noch in der Entwicklung befindet. Hinzu kommt, dass es gerade im Automobil Anwendungen mit unterschiedlichen Anforderungen an die Echtzeitfähigkeit ihrer Kommunikation gibt. So beträgt die maximale Latenz für Kontrollfunktionen $100\ \mu\mathrm{s}$ [60], während für kamerabasierte Einparkhilfen maximale Latenzen von $33\ \mathrm{ms}$ ausreichend sind. Dies spricht dafür, dass in zukünftigen, auf Ethernet basierenden Fahrzeugnetzen eine Kombination mehrerer Konzepte zum Einsatz kommt. In [60] wird die Kombination mehrerer Konzepte mit AVB durch Simulationsmodelle evaluiert, in [38] wird eine Kombination von TT-Ethernet und AVB ebenfalls anhand einer Simulation evaluiert. Da allerdings keinerlei Lösungen zur Erstellung von Prototypen zur Erprobung von neuen Konzepten und der Kombination mehrerer Konzepte existieren, beschäftigt sich diese Arbeit mit einem Framework zur Umsetzung solcher Prototypen. Mit dem umgesetzten Framework für Realtime-Bridges (RT-Bridge) soll folgendes erreicht werden:

- Umsetzung von Ethernet-Bridges mit mehreren Konzepten für Echtzeitfähigkeit
- Minimaler Aufwand bei der Umsetzung neuer Konzepte
- Klare und intuitive Interfaces zur Umsetzung neuer Konzepte
- Unterstützung von möglichst vielen Konzepten
- Deterministisches Verhalten bezüglich Jitter und Latenz

Das Framework wird FPGA-basiert umgesetzt. Zum Einsatz kommt die NetFPGA Plattform. Sie verfügt über eine Virtex II Pro der Firma Xilinx, SRAM, DRAM sowie vier Gigabit Ethernet Schnittstellen. Auf der NetFPGA Plattform wurden mehrere im Sourcecode verfügbare Projekte umgesetzt, unter anderem Ethernet Bridges und IPv4 (Internet Protocol Version 4) Router.

Im Folgenden Kapitel 2 wird die aktuelle Situation diskutiert, es wird auf verwandte Arbeiten eingegangen und die Ziele dieser Arbeit werden erläutert. Im Kapitel 3 wird das Konzept des Frameworks erläutert und in Kapitel 4 seine Realisierung beschrieben. In Kapitel 5 werden die ergriffenen Maßnahmen zur Qualitätssicherung dargestellt und das Framework in Kapitel 6 evaluiert. In Kapitel 7 wird diese Arbeit zusammengefasst und ein Ausblick auf zukünftige Herausforderungen gegeben.

2 Ist-Analyse

In Kapitel 1 wurden zukünftige und aktuelle Anwendungen im Automobil skizziert und grundlegend hergeleitet, warum Ethernet für zukünftige Anwendungen im Automobil wichtig ist. Im Folgenden wird tiefgehender auf die Motivation für Ethernet im Automobil eingegangen und in Abschnitt 2.1 die Funktionsweise von Ethernet erläutert. Danach werden in Abschnitt 2.2 auf Computersysteme mit Echtzeitanforderungen und in Abschnitt 2.3 Konzepte für echtzeitfähiges Ethernet dargestellt. In Abschnitt 2.4 wird auf verfügbare Optionen zum Erstellen von Realtime-Bridges eingegangen und in Abschnitt 2.5 die Ziele dieser Arbeit definiert.

Zur Realisierung aktueller Anwendungen im Automobil wird eine Reihe von Bussystemen in Fahrzeuge integriert. Die einzelnen Busse werden über Gateways verbunden[54]. Zum Einsatz kommen Bussysteme wie CAN (Controler Area Network) [3], FlexRay [46], LIN (Local Interconnect Network) [36], MOST (Media Oriented Systems Transport) [39] und der Signalisierungsstandard LVDS (Low Voltage Differential Signaling). Bei Feldbussystemen wie CAN, LIN und FlexRay teilen sich alle am Bus angeschlossenen Knoten eine maximale Datenrate von $10\ \mathrm{Mbit/s}$ (FlexRay $<10\ \mathrm{Mbit/s}$, CAN und LIN: $< 1\ \mathrm{Mbit/s}$). MOST und LVDS verfügen zwar über mehr Bandbreite, mit LVDS lassen sich aber nur Punkt-zu-Punkt Verbindungen umsetzen und für MOST sind nur Ring-Topologien vorgesehen.

Wie in Kapitel 1 erläutert müssen die eingesetzten Bussysteme beispielsweise für Fahrerassistenzsysteme oder Regelsysteme Echtzeitanforderungen erfüllen. Die genauen Anforderungen an Anwendungen und Kommunikation mit Echtzeitanforderungen werden in Abschnitt 2.2 beschrieben. Werden die Echtzeitanforderungen nicht erfüllt, kann dies die funktionale Sicherheit des Fahrzeuges beeinträchtigen. Gleichzeitig werden für Fahrerassistenzsysteme mehrere Kameras verwendet, wobei die Kameras Bilddaten mit einer hohen Bandbreite übertragen. Für Applikationen mit geringer Bandbreite werden aktuell Bussysteme mit einer gemeinsamen Kollisionsdomäne (CAN, LIN und FlexRay) verwendet. Die Echtzeitfähigkeit wird durch eine umfangreiche Planung und Analyse der Kommunikationsströme über die Bussysteme erreicht. Für Datenströme mit hoher Datenrate werden serielle Punkt-zu-Punkt Verbindungen (LVDS) verwendet. Da hier lediglich ein Signal vermittelt wird, werden die Echtzeitanforderungen erfüllt. Dementsprechend können aktuell verwendete Bussysteme zwar die

Byte	-8 bis -2	-1	0 bis 5	6 bis 11	12 bis 13	14 bis n-1	n bis n+4
Feld	Präambel	SFD	Zieladresse	Quelladresse	Type	Nutzlast	FCS
Wert	55 55 55 55 55 55 55	D5	xx xx xx xx xx xx	xx xx xx xx xx xx	xx xx	xx ...	xx xx xx xx

Tabelle 2.1: Struktur eines Ethernet Pakets

Anforderungen an Bandbreite und Echtzeitfähigkeit erfüllen, es entstehen aber komplizierte und inhomogene Fahrzeugnetze. Dies führt trotz der verwendeten softwarebasierten automatischen Routingalgorithmen für Kabelbäume zu hohem Entwicklungsaufwand [37].

Das in IEEE 802.3 [18] standardisierte Ethernet stellt eine Alternative zu den genannten Bussystemen dar. Es stellt eine flexible Strukturierung der Topologien, Übertragungsraten zwischen $10\ \mathrm{Mbit/s}$ und $40\ \mathrm{Gbit/s}$, getrennte Kollisionsdomänen (also kein geteiltes Medium) sowie unterschiedliche physikalische Medien wie Kupferadern oder Glasfasern zur Verfügung. Gleichzeitig wird Ethernet in vielen Anwendungsbereichen bereits verwendet, wodurch Hardware kostengünstig verfügbar ist und bei der Entwicklung von Applikationen auf umfangreiche Erfahrungen zurückgegriffen werden kann. Da Ethernet jedoch über keine Echtzeitfähigkeiten verfügt, existieren diesbezüglich mehrere Erweiterungskonzepte. Für um Echtzeitfähigkeiten erweitertes Ethernet wird der Terminus Realtime-Ethernet (RT-Ethernet) verwendet. RT-Ethernet erlaubt es, die Signale diverser Anwendungen mit unterschiedlichen Anforderungen an die Echtzeitfähigkeit und Bandbreite über ein Netzwerk zu vermitteln. Im folgenden Abschnitt 2.1 wird die Funktionsweise von Ethernet erläutert.

2.1 IEEE 802.3 Ethernet

IEEE 802.3 Ethernet [18] erlaubt eine paketorientierte Datenvermittlung. Es besteht mit PHY (Physical Layer) und MAC (Medium Access Control) eine klare Trennung zwischen Übertragungs- (OSI-Modell, Schicht 1 [64]) und Sicherungsschicht (OSI-Modell, Schicht 2). Ursprünglich war für die Übertragungsschicht CSMA/CD (Carrier Sense Multiple Access Collision Detection) vorgesehen, also eine gemeinsame Kollisionsdomäne und Half Duplex. Allerdings spielt CSMA/CD mittlerweile keine Rolle mehr, zwischen Netzwerkknoten bestehen meist Full Duplex Punkt-zu-Punkt Verbindungen ohne Kollisionsdomänen. Sollen mehr als zwei Endgeräte mit Punkt-zu-Punkt Verbindungen verbunden werden, werden Bridges (auch: Switch) zur gezielten Vermittlung der Pakete an den Empfänger verwendet. Wie in Tabelle 2.1 dargestellt, enthalten die Pakete jeweils eine $48\ \mathrm{bit}$ breite Ziel- und Quelladresse sowie ein $16\ \mathrm{bit}$ breites Type-Feld zur Identifizierung des nächsten verwendeten Protokolls. Weiterhin markiert eine $64\ \mathrm{bit}$ Präambel den Paketanfang, eine $32\ \mathrm{bit}$ breite FCS (Frame Check Sequence) dient

der Fehlererkennung. Zwischen zwei Paketen muss mit der Interframe Gap (IFG) eine Pause mit einer minimalen Länge von 12 B bei Fast Ethernet (100 Mbit/s Datenrate) sowie 8 B bei Gigabit Ethernet (1000 Mbit/s Datenrate) eingehalten werden. Bridges entscheiden anhand der Zieladresse, auf welchen Ports Pakete weitergeleitet werden. Über einen Port erreichbare Endgeräte werden anhand der Quelladressen gelernt.

2.2 Computersysteme mit Echtzeitanforderungen

Im Folgenden wird auf unter Echtzeitanforderungen stehende Computersysteme und die sich daraus ergebenden Anforderungen an die Computersysteme und deren Kommunikationsnetze eingegangen.

Echtzeitfähige Computersysteme werden nach [31] wie folgt definiert:

> A real-time computer system is a computer system where the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced.

Ein echtzeitfähiges Computersystem ist also ein Computersystem, bei dem die Korrektheit des Verhaltens nicht nur vom Ergebnis sondern auch vom Zeitpunkt der Fertigstellung des Ergebnisses abhängt. Echtzeitfähige Computersysteme sind dabei nur ein Teil des Gesamtsystemes und beeinflussen oft physikalische Prozesse.

Die beeinflussten Prozesse erfordern dabei die Reaktion des echtzeitfähigen Computersystems auf ein zur Arrival Time eingetroffenes Ereignis bis zu einer definierten Deadline. Erfolgt die Reaktion nicht bis zum Zeitpunkt der Deadline, so wird die Deadline verletzt. Ein Beispiel hierfür sind die Bremssysteme eines Fahrzeuges. Wird vom Fahrer die Bremse betätigt und die Bremsung nicht rechtzeitig durch das System ausgelöst, droht ein Unfall. Bei der genannten Deadline zur Auslösung der Bremsung handelt es sich um eine harte Deadline, da eine Verletzung der Deadline zu einer kritischen Fehlfunktion des Gesamtsystems führt. Eine Deadline ist firm (auch: fest), wenn ihre Verletzung zu einer tolerablen Fehlfunktion des Gesamtsystems führt und das Ergebnis nach der Deadline nicht mehr zu verwendet ist. Bei einer weich (auch: soft) Deadline ist das Ergebnis bei einer Verletzung der Deadline noch zu verwenden. Kann ein Computersystem beispielsweise beim Abspielen von Audiodaten die Audiodaten nicht rechtzeitig bereitstellen, wird die Wiedergabe unterbrochen. Dies wird zwar vom Anwender als störend empfunden, führt aber nicht zu einem relevanten Schaden. Werden die Audiodaten während der Unterbrechung verworfen, handelt es sich um eine firme Deadline. Werden sie nach der Unterbrechung ohne Verwerfung von Daten abgespielt, handelt es sich um eine

weiche Deadline. Entsprechend der Klassifizierung der Deadlines unterliegen Echtzeitsysteme harten, firmen oder weichen Echtzeitanforderungen.[31]

Die Reaktion eines Computersystems auf ein Ereignis beinhaltet eine über einen Zeitraum andauernde Verarbeitung. Die Verarbeitung beginnt zur Start Time und endet zur Finish Time. Aus der Differenz der Start und Finish Time ergibt sich die Verarbeitungszeit. Weiterhin werden Ereignisse in periodisch und aperiodisch auftretende Ereignisse unterteilt. Periodisch auftretende Ereignisse treten mit einer definierten Periode auf, wie beispielsweise mit einer definierten Frequenz ausgelesene Messwerte. Andererseits sind Periode und Arrival Time von aperiodisch auftretenden Ereignissen unbekannt, wie beispielsweise Nutzereingaben beim Schreiben eines Textes auf einer Tastatur.[31, 63]

Verarbeitet ein Computersystem aperiodisch auftretende Ereignisse, geschieht die Verarbeitung Event Triggered (ET). Durch die maximale Rechenleistung des Computersystems ergibt sich dabei eine maximale zu verarbeitende Ereignisrate. Übersteigt die Ereignisrate die Rechenleistung des Computersystems, werden die Deadlines verletzt. Periodisch auftretende Ereignisse treten einerseits periodisch zu definierten Zeitpunkten auf. Dabei kann Start und Finish Time im Voraus geplant werden und somit ein Einhalten der Deadlines garantiert werden. Die Verarbeitung der Ereignisse geschieht somit Time Triggered (TT). Andererseits kann lediglich die Periode des Auftretens der Ereignisse bekannt sein. Somit ist die Arrival Time nicht im Voraus bekannt, Start und Finish Time ergeben sich zur Laufzeit. Um das Einhalten der Deadlines zu garantieren, muss eine maximale Ereignisrate beziehungsweise minimale Periode der Ereignisse sowie eine maximale Verarbeitungszeit garantiert werden. Die Verarbeitung der Ereignisse geschieht somit Rate Constrained (RC).[31]

Wie in Kapitel 1 beschrieben, bestehen Automobile aus einer Vielzahl von Computersystemen, die über ein Kommunikationsnetz verbunden sind. Da zur Durchführung unter Echtzeitanforderungen stehender Aufgaben gegebenenfalls das Kommunikationsnetz verwendet wird, muss es aus Sicht einer unter Echtzeitanforderungen stehende Applikation die folgenden Anforderungen erfüllen:

- Für die Latenz einer Nachricht muss ein Maximum garantiert werden.
- Nachrichten werden genau einmal gesendet, also weder verworfen oder noch mehrfach zugestellt.
- Die Reihenfolge von Nachrichten wird nicht geändert.
- Fehler müssen erkannt werden.
- Abhängig vom Anwendungsfall ist eine transparente redundante Auslegung erforderlich.
- Die Nachrichtenrate muss zur Vermeidung von Überlastung begrenzt werden.

Da die konkreten Anforderungen an das Kommunikationsnetz abhängig von der Aufgabe variieren, werden unterschiedliche Servicequalitäten (Quality of Service, QoS) oder Nachrichtenklassen benötigt. Teile der Anforderungen können auch durch Softwarekomponenten des Computersystems übernommen werden, beispielsweise die Wiederherstellung der Reihenfolge eingehender Nachrichten. In dieser Arbeit werden Konzepte zur Fehlererkennung und für Redundanz nicht weiter betrachtet, daher wird in den folgenden Abschnitten und Kapitel nicht weiter auf diese eingegangen. Im folgenden Abschnitt wird auf Konzepte für echtzeitfähiges Ethernet eingegangen.

2.3 RT-Ethernet Konzepte

Wie am Anfang dieses Kapitels erläutert, existieren unterschiedliche Konzepte für RT-Ethernet. Die folgenden RT-Ethernet Konzepte werden erläutert:

- Virtual Bridged Local Area Networks (VLAN), spezifiziert in IEEE 802.1Q [14], beschrieben in Abschnitt 2.3.1.
- Audio-/Video-Bridging (AVB), spezifiziert in IEEE 802.1BA und IEEE 802.1Q [13, 14], beschrieben in Abschnitt 2.3.2.
- Avionics Full-Duplex Switched Ethernet Network (AFDX), spezifiziert in ARINC 664 Part 7 [1], beschrieben in Abschnitt 2.3.3.
- Time-Triggered Ethernet (TT-Ethernet, TTE), spezifiziert in AS6802 [61], beschrieben in Abschnitt 2.3.4.
- Time Sensitive Networking (TSN), spezifiziert in IEEE 802.1Qbv und IEEE 802.1Q [24, 14], beschrieben in Abschnitt 2.3.5.

2.3.1 IEEE 802.1Q VLAN

Der Standard IEEE 802.1Q [14] erweitert IEEE 802.3 Ethernet um Virtuelle Netzwerke (Virtual Bridged Local Area Networks, VLAN) und Prioritäten. Zu den in Abschnitt 2.1 beschriebenen Feldern kommen die in Tabelle 2.2 dargestellten hinzu. Das Type-Feld des Ethernet Headers enthält 0x8100 (TPID, Tag Protocol Identifier), darauf folgen 3 bit Priorität (PCP, Piority Code Point), 1 bit Drop Eligible Indicator (DEI) und 12 bit VID (VLAN Identifier). Diese Felder ergeben zusammen 16 bit TCI (Tag Control Information). Danach folgt ein 16 bit breites Type-Feld zur Identifizierung des nächsten verwendeten Protokolls. Da es, wie in Abbildung 2.1 dargestellt, für jede ausgehende Ethernet Schnittstelle (Port) sowohl bei Bridges als auch bei Endgeräten separate Puffer für bis zu 8 Prioritäten gibt, kann im Regelfall für Nachrichten

Byte	-8 – -2	-1	0 – 5	6 – 11	12 – 13	14 – 15	16 – 17	18 – n-1	n – n+4
Feld	Präambel	SFD	Zieladresse	Quelladresse	TPID	TCI	Type	Nutzlast	FCS
Wert	55 55 55 55 55 55 55 55	D5	xx xx xx xx xx xx	xx xx xx xx xx xx	81 00	xxxx	xxxx	xx …	xxxxxxxx

TPID				
Byte	14			15
Bit	7 – 5	4	3 – 0	7 – 0
Feld	PCP	DEI	VID	

Tabelle 2.2: Struktur eines Ethernet Pakets mit VLAN Tag

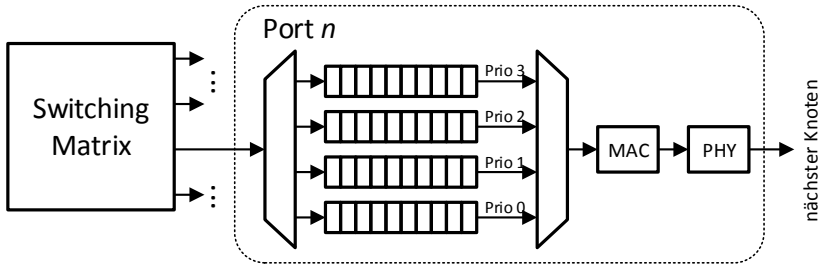


Abbildung 2.1: Queues eines ausgehenden Ports bei IEEE 802.1Q mit 4 Paketklassen

mit hoher Priorität bei definiertem Sendeverhalten aller Endgeräte ein echtzeitfähiges Kommunikationsverhalten mit einer Obergrenze für Latenzen erreicht werden.

Um eine Obergrenze für Latenzen zu erreichen, muss das gesamte Netzwerk umfangreich geplant werden. So muss zum Erreichen geringer Latenzen unter anderem darauf geachtet werden, dass an keiner Stelle des Netzes mehrere Pakete gleichzeitig auf dem selben Port gesendet werden sollen. Weiterhin verschlechtern direkt aufeinander folgende Pakete (Bursts) mit hoher Priorität die maximale Latenz von Paketen mit geringerer Latenz massiv. Eine so erreichte Garantie für Echtzeitfähigkeit kann allerdings durch ein einzelnes Endgerät mit nicht planmäßigem Sendeverhalten verhindert werden. So können nicht planmäßig gesendete Pakete mit hoher Priorität geplante Pakete unkontrolliert verzögern. Paketverluste werden generell nicht erkannt, somit müssen betreffende Fehlerfälle durch die Anwendung erkannt werden.

2.3.2 IEEE 802.1BA AVB

Der Standard IEEE 802.1BA [13] für Audio-/Video-Bridging (AVB) beschreibt eine Erweiterung des IEEE 802.1Q Standards zum garantierten Erreichen von Latenzobergrenzen. AVB zielte ursprünglich auf Anwendungen ab, die für das Vermitteln von Audio- und Videodatenströmen garantierte Latenzen und ein genaue gemeinsame Zeitbasis benötigen, ist aber nicht darauf beschränkt. Ein Beispiel ist die Vermittlung der Tonsignale bei einem Live-Konzert von den Instrumenten und Mikrofonen über Effektgeräte und ein Mischpult zu den Lautsprechern.

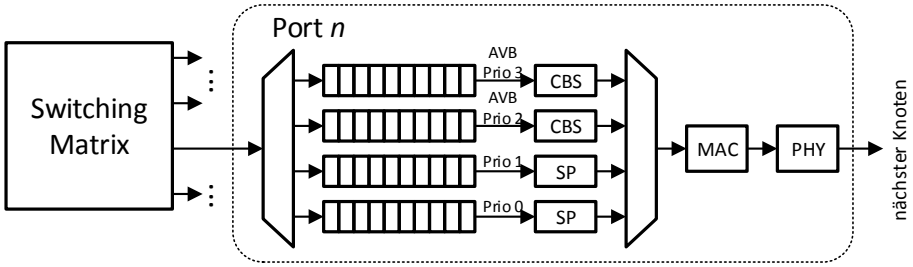


Abbildung 2.2: Queues eines ausgehenden Ports bei IEEE 802.1Q mit 4 Paketklassen, davon 2 AVB Paketklassen

Zur Vermeidung von Auslöschungen der Schallwellen durch Interferenzen sollten die Lautsprecher das Tonsignal ohne Phasenverschiebung ausspielen. Dies kann nur geschehen, wenn alle Tonquellen und -senken über eine präzise gemeinsame Zeitbasis verfügen. Diese wird bei AVB durch eine in IEEE 802.1AS [12] definierte Untermenge von IEEE 1588 [28] erreicht. Weiterhin sollte eine Verzögerung der Tonsignale durch den Hörer nicht wahrnehmbar sein und die Tondaten sollten bis zum synchronen Abspielen alle Lautsprecher erreicht haben. Daher muss eine maximale Latenz garantiert werden.

Mit AVB lassen sich Multicast-Datenströme mit definierter Datenrate von einem Sender zu mehreren Empfängern vermitteln. Bandbreiten werden bei Bedarf mit dem Stream Reservation Protocol (SRP, auch MSRP, siehe IEEE 802.1Qat [16]) reserviert und stehen dann mit garantierter Latenz zur Verfügung. Kann die Latenz eines Datenstroms an einem Punkt der Übertragungsstrecke nicht garantiert werden, wird dies als Fehler durch das SRP gemeldet. Dies kann entweder bei der Reservierung durch Belegung der Bandbreite durch reservierte Datenströme oder bei einer Änderung der Bandbreite geschehen, beispielsweise bei WLAN. Für AVB sind mehrere Paketklassen vorgesehen, meist werden die Klassen A und B mit garantierten Latenzen von 2 ms und 50 ms über 7 Bridges genannt. An jedem ausgehenden Port ist, wie Abbildung 2.2 zeigt, für jede Paketklasse ein Transmission Selection Algorithm (TSA) vorgesehen. Für die AVB Paketklassen wird der Credit Based Shaper Algorithm (CBS, siehe IEEE 802.1Qav[17]) verwendet. Er sorgt für das Einhalten der reservierten Bandbreite. Für die übrigen Paketklassen ist der Strict Priority Algorithm (SP) vorgesehen. Er unterscheidet sich im Verhalten nicht vom ursprünglichen IEEE 802.1Q VLAN.

Jeder CBS hält einen *credit* vor. Ist der *credit* positiv oder 0, darf die entsprechende Paketklasse am zugeordneten Port senden. Beim Senden eines Paketes der entsprechenden Paketklasse wird der *credit* um *sendSlope* dekrementiert. Ist der *credit* negativ oder wird ein Paket der Paketklasse verzögert, wird *credit* um *idleSlope* inkrementiert. Wenn der *credit* positiv ist und keine Pakete der Paketklasse sendebereit sind, wird der *credit* auf 0 gesetzt. In Abbildung

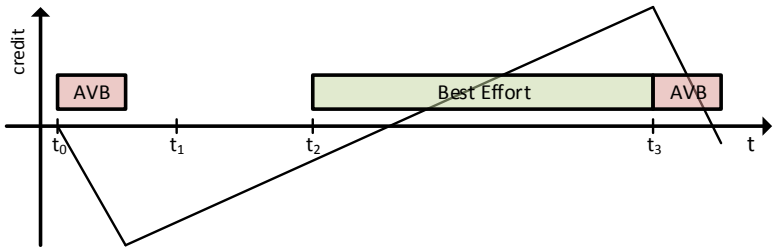


Abbildung 2.3: Exemplarischer Verlauf des Credits bei einem CBS mit den aktuell gesendeten Paketen

2.3 ist der Verlauf des *credits* beim Senden mehrerer Pakete exemplarisch dargestellt. Anfangs ist der *credit* 0, das bei t_0 vorhandene AVB-Paket wird versendet und der *credit* fällt. Ab t_1 wartet ein AVB-Paket auf einen positiven *credit*. Kurz vor Erreichen eines positiven *credits* wird bei t_2 ein Paket mit geringerer Priorität begonnen. Das wartende AVB-Paket kann somit trotz positivem *credits* erst mit dem Ende des angefangenen Pakets bei t_3 gesendet werden. Da das an t_2 begonnene Paket bis zu $1530\ \mathrm{B}$ (inklusive Ethernet Feldern, mit Präambel) lang sein kann, ergibt sich hierdurch eine Verzögerung von bis zu $122,4\ \mu\mathrm{s}$ bei $100\ \mathrm{Mbit/s}$ Übertragungsrate. Zusammen mit der Zeit zum Versenden des AVB-Paketes von bis zu $122,4\ \mu\mathrm{s}$ ergibt sich die maximale Latenz beim Versenden eines AVB-Paketes über einen Port. Fällt diese bei 8 ausgehenden Ports (1 Sender und 7 Bridges) an, ergibt sich mit etwa $2\ \mathrm{ms}$ die maximale Latenz von Paketklasse A.

Das beschriebene Verhalten der CBS resultiert darin, dass der Abstand zwischen zwei aufeinander folgenden Paketen der selben AVB-Klasse maximiert wird, während die reservierte Bandbreite noch eingehalten werden kann. Dementsprechend werden Bursts aufgelöst, die maximale Latenz von Paketklassen mit geringerer Priorität wird minimal größer. Weiterhin wird die genutzte Bandbreite der einzelnen AVB-Klassen effektiv auf die reservierte Bandbreite limitiert, die nicht genutzte Bandbreite steht anderen Paketklassen zur Verfügung. Die in [60] angegebene zu garantierende Latenz von Kontrollfunktionen im Automobil von $100\ \mu\mathrm{s}$ über 5 Bridges wird allerdings durch AVB nicht erreicht.

2.3.3 ARINC 664 AFDX

Avionics Full-Duplex Switched Ethernet Network (AFDX) ist im ARINC 664 Standard Part 7 [1] spezifiziert und für die Anwendung in sicherheitskritischen Systemen der Luftfahrt vorgesehen. Dabei wird Ethernet um das Konzept der statisch konfigurierten virtuellen Links erweitert. Über einen virtuellen Link werden Pakete mit einer limitierten Paketrate vermittelt. Die einem virtuellen Link zugeordneten Pakete werden durch zugewiesene Zieladressen

identifiziert. Um ein Überlasten von Endgeräten durch zu hohe Paketraten zu verhindern, wird bei AFDX für virtuelle Links die Paketrate durch Traffic Shaper limitiert. Somit wird die Rate der Events für Anwendungen, wie in Abschnitt 2.2 beschrieben, für RC-Anwendungen begrenzt. Die Traffic Shaper sorgen dafür, dass nach einem Paket ein Mindestzeitraum vor dem Versand des nächsten Paketes gewartet wird (Bandwidth Allocation Gap, BAG). Durch die maximale Paketgröße wird entsprechend die Bandbreite begrenzt. Wie allerdings in [49] beschrieben beträgt bei AFDX der Jitter allein bis zu 500 µs. Dementsprechend genügt AFDX nicht den Anforderungen von Kontrollfunktionen im Automobil.

2.3.4 AS 6802 TT-Ethernet

Time Triggered Ethernet (TT-Ethernet, TTE) wurde an der TU-Wien entwickelt, wird von der Firma TTTech vertrieben und wurde in AS 6802 [61, 62] standardisiert. Es unterstützt die Paketklassen Time Triggered (TT), Rate Constrained (RC) und Best Efford (BE). Die TT-Paketklasse ist für Anwendung in sicherheitskritischen Systemen vorgesehen und erreicht geringere garantierte Latenzen und Jitter als AVB und AFDX. So können Latenzen von < 100 µs garantiert werden [55]. Diese Latenzen können allerdings nur für eine im Voraus bekannte Menge von TT-Paketen garantiert werden. Für jedes TT-Paket muss der Pfad durch das Netzwerk, die Paketlänge, die Sendeperiode sowie der Sendezeitpunkt an jeder Bridge relativ zu einem global synchronisierten Zyklus bekannt sein. Es muss also ein Schedule mit genannten Informationen zu den zu vermittelten TT-Paketen im Voraus erstellt und in die beteiligten Netzwerkkomponenten eingespielt werden.

Somit ist für jeden ausgehenden und eingehenden Port im Netzwerk im Voraus bekannt, wann welche TT-Pakete eintreffen oder versandt werden müssen. Die TT-Pakete werden anhand von eindeutig zugewiesenen Zieladressen (Critical Traffic ID) identifiziert und an den ausgehenden Ports sind dementsprechend Zeitfenster zum Versenden reserviert. In den reservierten Zeitfenstern werden also keine anderen Pakete versandt. Vor dem Versenden anderer Paketklassen wird geprüft, ob das Paket vor dem Beginn des nächsten reservierten Zeitfensters beendet werden kann. Beim Empfang der TT-Pakete wird weiterhin geprüft, ob der Empfangszeitpunkt und die Paketlänge den Erwartungen entspricht. Ist dies nicht der Fall, werden sie verworfen. Für jedes TT-Paket ist ein eigener Puffer alloziert, weshalb TT-Pakete nicht auf Grund von nicht zur Verfügung stehenden Speicherressourcen verworfen werden können. Das Vermitteln der TT-Pakete zu definierten, zyklisch auftretenden Zeitpunkten entspricht dem Time Division Multiple Access (TDMA) Verfahren. So kann garantiert werden, dass vor jedem Ausführen einer in Abschnitt 2.2 beschriebenen TT-Anwendung die erforderlichen Pakete vorliegen und danach versandt werden können. Liegen die erforderlichen

Feld	Zieladresse	Quelladresse	Type	Integration Cycle	Membership New
Bit	0 – 47	48 – 95	96 – 111	112 – 143	144 – 175

Feld	Reserved	Sync Priority	Sync Domain	Type	Reserved	Transparent Clock
Bit	176 – 207	208 – 215	216 – 223	224 – 227	228 – 271	272 –335

Tabelle 2.3: Aufbau der PC-Frames von TTE [62]

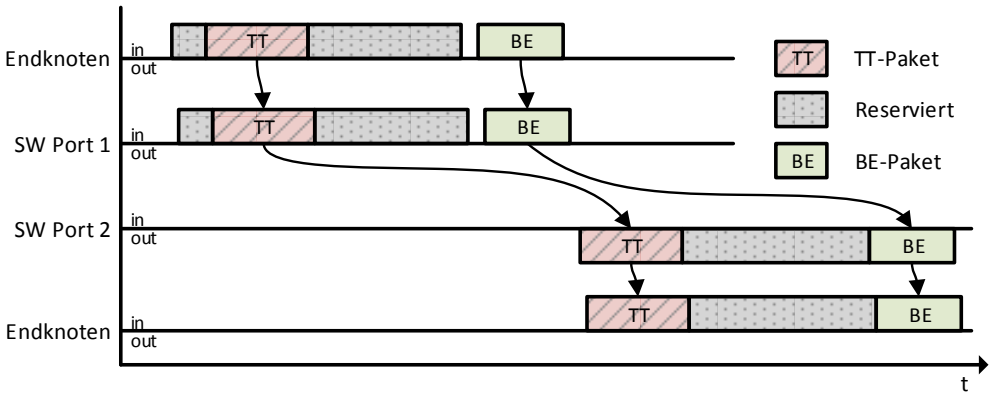


Abbildung 2.4: Vermittlung eines TT-Pakets mit interferierendem BE-Paket

Pakete nicht wie erwartet vor, liegt ein Fehlerfall vor und die entsprechende Fehlerbehandlung kann durchgeführt werden. Bei der Paketklasse RC handelt es sich um AFDX, bei BE um herkömmliches Ethernet.

Um allen beteiligten Netzwerkknoten eine globale Zeitbasis zur Verfügung zu stellen, werden durch einen oder mehrere Synchronization Master Protocol Control Frames (PC-Frame) versandt. Der Aufbau der PC-Frames ist in Tabelle 2.3 dargestellt. Die PC-Frames werden zu definierten Zeitpunkten im Schedule in der Paketklasse RC versandt. Durch die vermittelnden Netzwerkknoten wird im PC-Frame im Feld *transparentClock* die bei der Vermittlung anfallende Zeit akkumuliert. Da die sich synchronisierenden Synchronization Clients den Empfangszeitpunkt anhand des bekannten Sendezeitpunktes und der akkumulierten Vermittlungszeit ermitteln können, können sie ihre lokale Uhr auf die des Synchronization Masters synchronisieren.

Die beispielhafte Vermittlung von TT-Nachrichten von einem Endknoten über eine Bridge zu einem anderen Endknoten ist in Abbildung 2.4 dargestellt. Der an Port 1 der Bridge angeschlossene Endknoten überträgt im für TT-Pakete reservierten Zeitfenster ein TT-Paket an die Bridge. Nach dem Ende des reservierten Zeitfensters wird ein BE-Paket vermittelt. Beide Pakete sollen über Port 2 an den zweiten Endknoten vermittelt werden. Nachdem das BE-

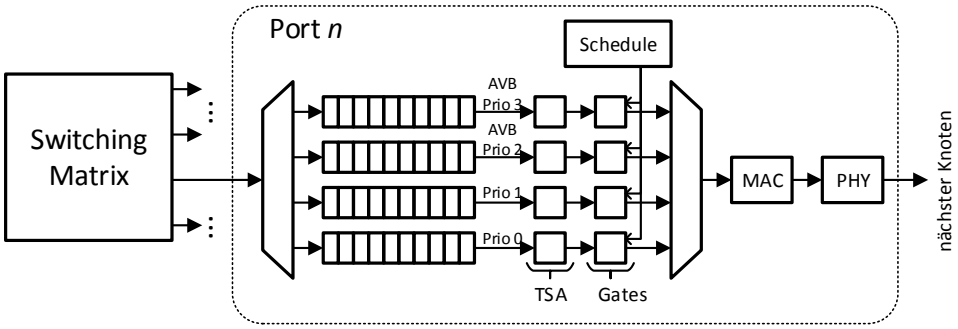


Abbildung 2.5: Queues, TSAs, Gates und Arbiter eines ausgehenden Ports bei TSN mit Scheduled Traffic

Paket vollständig von der Bridge empfangen wurde, ist der Port 2 zwar frei, da das BE-Paket aber nicht vor Beginn des für das TT-Paket reservierten Zeitfensters übermittelt werden kann wird erst das TT-Paket übermittelt. Nach dem Ende des reservierten Zeitfensters wird das BE-Paket übertragen.

2.3.5 TSN

Time Sensitive Networking (TSN) wird aktuell als Nachfolger von AVB standardisiert. Um über TSN Kontrollfunktionen mit Bedarf nach einer garantierten Latenz von unter 100 µs über 5 Bridges zu betreiben [60], wird TSN Konzepte für Schedule Traffic, Redundanz und Unterbrechen von Paketen sowie neue TSA beinhalten. Zur Reduzierung der Latenz sind mehrere Konzepte vorgesehen[57]. Im Folgenden werden die unterschiedlichen Konzepte erläutert.

Scheduled Traffic wird mit Time Aware Gates im IEEE 802.1Qbv Draft Standard spezifiziert[24]. Scheduled Traffic ähnelt der TT-Paketklasse bei TT-Ethernet. Allerdings können in einem Zeitfenster mehrere Nachrichten vorkommen und die Pakete werden in FIFO-Queues gepuffert. So gibt es zusätzlich zu den TSAs ein Time Aware Gate für jede Priorität an jedem Port. Time Aware Gates sind entweder offen oder geschlossen. Ist ein Gate offen, darf die zugehörige Priorität Pakete versenden. Vor dem Senden eines neuen Paketes muss geprüft werden, ob es vor dem Schließen des Gates beendet werden kann. Für jeden Port gibt es einen Schedule, dem entsprechend die Gates angesteuert werden. Dies ist in Abbildung 2.5 dargestellt.

Cyclic Queueing beziehungsweise der Peristaltic Shaper werden aktuell im IEEE 802.1Qch Draft Standard [26] spezifiziert. Da noch keine Versionen dieses Drafts verfügbar sind, sind die folgenden Informationen Vortragsfolien der TSN Arbeitsgruppen entnommen [58, 59]. Der Peristaltic Shaper stellt eine simple Alternative zu Scheduled Traffic dar. Er erlaubt determinis-

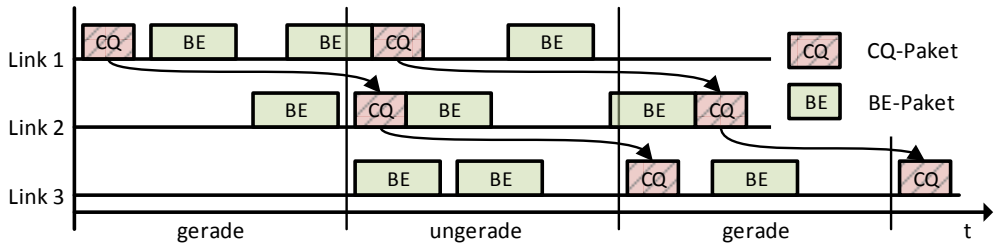


Abbildung 2.6: Beispielhafte Übermittlung von Cyclic Queueing (CQ) Traffic mit interferierenden BE-Paketen

tische Latenzen und schließt Paketverlust durch Anstauung von Paketen aus. Die Pakete werden in sich abwechselnden, geraden und ungeraden Zyklen versandt. Pakete die in ungeraden Zyklen empfangen wurden, werden in die FIFO-Queue für gerade Zyklen abgelegt und umgekehrt. Die Pakete in der Queue für gerade Zyklen werden in den geraden Zyklen versandt, ungerade dementsprechend. Der Zyklus muss dabei mindestens so lang wie ein interferierendes Paket mit maximaler Länge sein. Dementsprechend beträgt bei $100\ \mathrm{Mbit/s}$ Bandbreite der minimale Zyklus $125\ \mu\mathrm{s}$ und $12,5\ \mu\mathrm{s}$ bei $1\ \mathrm{Gbit/s}$ Bandbreite. Wenn Pakete unterbrochen werden können, dann reduziert sich die minimale Zykluslänge entsprechend. Die Latenz pro Hop entspricht maximal zwei Zyklen. Der benötigte Speicher für die Paketqueues ergibt sich entsprechend aus der Bandbreite des Links und der Zykluslänge. In Abbildung 2.6 ist beispielhaft die Übermittlung von zwei Nachrichten mit Hilfe von Cyclic Queueing dargestellt. Dabei interferieren mehrere Best Effort Pakete mit ihnen. Zu erkennen ist, dass die BE-Pakete die als Cyclic Queueing Traffic weitergeleiteten Pakete überholen und gegebenenfalls verzögern. Allerdings beträgt die maximale, zusätzliche Verzögerung über den gesamten Pfad maximal eine Zykluslänge.

Weiterhin steht der Burst Limiting Shaper (BLS) [10, 9, 11] für Paketklassen mit hoher Priorität zur Diskussion. Er lässt Bursts mit einer definierten Maximallänge zu. Hierfür wird, ähnlich wie bei AVB, ein *credit* geführt. Beim Senden wird er inkrementiert und sonst dekrementiert, solange er > 0 ist. Erreicht der *credit* einen Maximalwert, wird die Priorität der Paketklasse herabgesetzt. Beim Erreichen einer Untergrenze wird die ursprüngliche Priorität wieder hergestellt. Dementsprechend werden im Gegensatz zum CBS von AVB Bursts mit begrenzter Anzahl von Paketen nicht verzögert, der negative Einfluss auf Paketklassen geringerer Priorität wird aber begrenzt. Beispielsweise entstehen bei Steuergeräten zur Anlagensteuerung, wie in [10, 11] beschrieben, periodisch kurze Bursts, da periodisch alle Sensoren einlesen, die Daten verarbeiten und Anweisungen an die Aktoren herausgeben werden. Im Gegensatz zum CBS kann eine geringe Latenz bei entsprechend kurzen Bursts garantiert werden und im Ver-

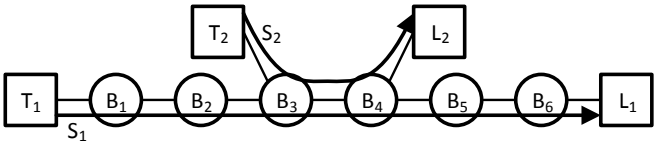


Abbildung 2.7: Beispielhaftes UBS Netzwerk mit den Streams S_1 und S_2[50]

gleich zum TAS müssen keine Schedules erstellt und konfiguriert werden, sondern lediglich die maximale Bandbreite und Burstlänge konfiguriert werden.

Auch mit dem Urgency Based Scheduler (UBS) [50, 53, 51, 52] sollen minimale Latenzen erreicht werden und dabei wie der BLS weniger Konfigurationsaufwand als Scheduled Traffic aufweisen. So muss für Streams nur die Periode und maximale Paketlänge angegeben werden. Dementsprechend muss kein konkreter Schedule entwickelt werden. Das Konzept des UBS basiert darauf, dass Pakete mit einem langen Pfad, also vielen Hops, bevorzugt und somit früher weitergeleitet werden. Dies soll gerade bei Netzwerken mit linienförmiger Topologie, wie in Abbildung 2.7 dargestellt, zu geringeren maximalen Latenzen führen. Um Pakete mit langem Pfad früher weiterzuleiten, wird bei der Auswahl des zu sendenden Paketes auf einem Port beim UBS das dringlichste Paket ausgewählt. Das dringlichste Paket ist das Paket, welches seiner maximalen Latenz (oder auch: Deadline) am nächsten ist. So sind beispielsweise in Abbildung 2.7 bei Bridge B_3 Pakete des Streams S_1 denen von Stream S_2 vorzuziehen, da bei gleicher Maximallatenz für Pakete von S_1 weniger Zeit bis zum Erreichen der maximalen Latenz verbleibt. Da an einem Port eingehende Pakete bereits vom vorherigen Netzwerkknoten nach Dringlichkeit geordnet wurden, wird pro ausgehendem Port lediglich eine FIFO pro eingehendem Port benötigt (siehe Abbildung 2.8). Um das dringlichste Paket auszuwählen, muss aus den jeweils erstem Paket jeder FIFO das dringlichste ausgesucht werden, da nachfolgende Pakete weniger dringlich sind.

Weiterhin soll die Unterbrechung von Paketen Frame Preemption zur Reduzierung des Guard Windows bei Scheduled Traffic und der Latenz von hohen Prioritäten generell führen. Frame Preemption wird in den Standards IEEE 802.1Qbu und IEEE 802.3br spezifiziert [23, 20]. Weiterhin sollen durch Input Gating (IEEE 802.1Qci [27]) bei eingehenden Paketen das Einhalten des Schedules geprüft werden. Der Standard IEEE 802.1CB (Seamless Redundancy [22]) definiert Redundanzkonzepte zum Ausschließen von Paketverlusten und durch IEEE 802.1Qcc [25] werden die neuen Konzepte mit dem SRP konfigurierbar.

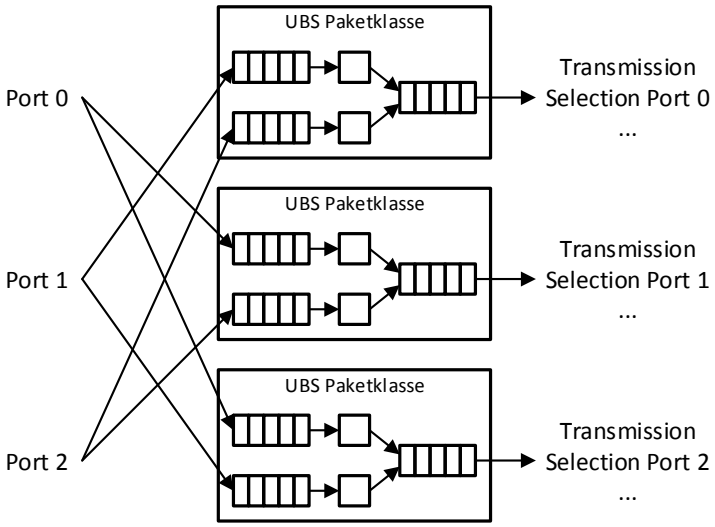


Abbildung 2.8: Queues der UBS Paketklasse für drei eingehend und ausgehende Ports[53]

2.4 Verwandte Arbeiten

Wie aus den vorherigen Abschnitte hervorgeht, benötigen unter anderen zukünftige Anwendungen im Automobil Fahrzeugnetze, die Echtzeitanforderungen erfüllen. Wie in den Kapiteln 1 und 2.2 erläutert, unterliegen unterschiedliche Anwendungen unterschiedlichen Echtzeitanforderungen, die gegebenenfalls mehrere Konzepte zur echtzeitfähigen Kommunikation erfordern. Dementsprechend wird davon ausgegangen, dass in zukünftigen Fahrzeugnetzen mehrere Konzepte zur Echtzeitkommunikation zum Einsatz kommen. Kombinationen von mehreren Konzepten für RT-Ethernet lassen sich bereits durch das Simulationsframework Core4Inet[1] für Omnet++[2] simulieren. Der nächste Schritt auf dem Weg der Produktentwicklung ist die Evaluation anhand von Prototypen. Daraus ergibt sich der Bedarf, zukünftige Fahrzeugnetze mit einer Kombination mehrerer Konzepte zur Echtzeitkommunikation anhand von Prototypen zu evaluieren. Mit der Realisierung von Prototypen für Bridges solcher Fahrzeugnetze befasst sich diese Arbeit. Im Folgenden wird auf verfügbare Plattformen zur Erstellung solcher Prototypen eingegangen.

Aktuell sind Bridges mit Unterstützung für Konzepte zur Echtzeitkommunikation kommerziell verfügbar. So bietet das Unternehmen Extreme Networks beispielsweise Bridges mit Unterstützung für AVB an[3]. Weiterhin stellt das Unternehmen TTTech Bridges mit Unterstüt-

[1] http://core4inet.realmv6.org/
[2] http://www.omnetpp.org/
[3] http://www.extremenetworks.com/audio-video-bridging

zung von TTE, AVB, TSN Scheduled Traffic und AFDX her[4]. Allgemein setzen die verfügbaren Bridges zwar ein oder mehrere Konzepte um, allerdings keine noch in der Spezifikation befindliche Konzepte. Insbesondere können neue Protokolle nicht integriert werden.

Ethernet Bridges lassen sich auch auf Basis von Prozessorsystemen mit Software umsetzten. Beispielsweise kann der Linux Kernel so konfiguriert werden, dass mehrere Ethernet-Karten zu einer Bridge verbunden werden[5]. Jedoch sind mit softwarebasierten Lösungen die erreichbaren Paketraten und Bandbreiten begrenzt. Netzwerkkarten verfügen mittlerweile über hardwarebasierte Zeitstempeleinheiten für die exakte Bestimmung des Empfangszeitpunktes von Paketen. Diese werden in der Arbeit [45] für eine Zeitsynchronisation mit Hilfe der von TTE bereitgestellten Synchronisationsnachrichten verwendet. Allerdings kommt es dabei zu einem maximalen Jitter von 3 µs. Weiterhin kann der Sendezeitpunkt von Paketen nicht mit der nötigen Präzision festgelegt werden. Spezialisierte Prozessorsysteme wie der AM5K2E0x Sitara[6] des Unternehmens Texas Instruments verfügen über spezialisierte Koprozessoren zur Umsetzung von Bridges. Sie bieten allerdings nicht die nötige Flexibilität, um Konzepte wie Scheduled Traffic umzusetzen.

2.4.1 NetFPGA

Zum garantierten Erreichen von hohen Paketraten und Bandbreiten bei der benötigten Präzision beim Sendezeitpunkt von Paketen bieten sich spezialisierte Hardwareplattformen an. Durch die Verwendung von FPGAs lassen sich die unterschiedlichen Konzepte zur Echtzeitkommunikation flexibel umsetzen. Entsprechende Plattformen mit mehreren an einen FPGA angeschlossenen Ethernet Ports sind verfügbar. So stehen mit den NetFPGA Plattformen[7] einerseits Plattformen mit unterschiedlicher Ausstattung zur Verfügung. Andererseits wurde bereits eine Reihe von Forschungsprojekten auf Basis der NetFPGAs umgesetzt[8]. Weiterhin stehen die umgesetzten Projekte meist unter einer freien Lizenz und sind somit im Quellcode frei verfügbar.

Zum Zeitpunkt des Beginns dieser Arbeit (Anfang 2013) waren zwei NetFPGA Plattformen verfügbar. Der NetFPGA 1G mit einem Virtex II Pro 50 FPGA, 4 PHYs für RJ45 mit 10/100/1000 MBit/s, 4,5 MB SRAM und 64 MB DDR2 DRAM wird als PCI-Karte in einen Host PC eingebaut.

[4] https://www.tttech.com/products/automotive/automotive-ethernet/switches/de-switch-hermes/
[5] http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge
[6] http://www.ti.com/lsds/ti/processors/sitara/arm_cortex-a15/am5k2ex/overview.page
[7] http://netfpga.org
[8] http://netfpga.org/site/#/systems/4netfpga-1g/applications/

Der NetFPGA 10G mit einem Virtex 5 TX240T FPGA, 4 PHYs für SFP mit 10 GBit/s, 27 MB Quad Data Rate Static Random Access Memory (QDRII SRAM) sowie 288 MB Reduced Latency Random Access Memory (RLDRAM II) wird als PCI-Express-Karte in einen Host-PC eingebaut, kann aber auch alleinstehend betrieben werden. Da davon ausgegangen wird, dass die NetFPGA 1G Plattform ausreichend Ressourcen zur Verfügung stellt und für diese Plattform mehr umgesetzte Projekte verfügbar sind, wird sie für diese Arbeit verwendet. Abgesehen von den NetFPGA Plattformen gibt es noch weitere Plattformen, keine dieser Plattformen verfügt aber über eine ähnliche Anzahl von im Quellcode verfügbaren Projekten.

Im Folgenden wird ein Überblick über die Architektur der NetFPGA 1G Plattform gegeben. Auf der NetFPGA 1G PCI-Karte sind, wie in Abbildung 2.9 dargestellt, vier PHYs mit Unterstützung für Ethernet, Fast Ethernet und Gigabit Ethernet (10/100/1000 MBit/s) verfügbar. Weiterhin sind als Speicher zwei SRAM Chips mit jeweils 36 Bit breiten Datenleitungen und zusammen 4,5 MB Kapazität sowie ein DDR2 RAM Chip mit 32 Bit breiten Datenleitungen und somit 64 breiter Anbindung an den NetFPGA angebunden. Der NetFPGA wird über die PCI-Schnittstelle der Karte durch den zweiten FPGA (CPCI FPGA) vom Host-PC konfiguriert. Über die PCI-Schnittstelle kann weiterhin durch den Host-PC auf Register innerhalb des NetFPGAs, den SRAM sowie das MDIO Interface der PHYs zugegriffen werden.

Beim Erstellen neuer Projekte wird auf eine Reihe von Modulen aus einer Bibliothek zurückgegriffen. Die Bibliothek beinhaltet Module zum Zugriff auf die auf dem Board vorhandene Hardware sowie Module mit oftmals benötigter Funktionalität. Somit beschränkt sich das Erstellen neuer Projekte auf die Umsetzung der Kernfunktionalität im User Data Path. Die Bibliotheken beinhalten weiterhin eine Toolchain aus Makefiles zum Erstellen der FPGA-Konfiguration mit Hilfe der Xilinx Tools sowie Software und Kernel-Treiber für den Zugriff auf die NetFPGA Plattform. Auf Basis der NetFPGA 1G Plattform wurden eine Reihe von Projekten umgesetzt. Diese wurden auf Verwendbarkeit für diese Arbeit untersucht. Im Folgenden werden thematisch in Frage kommende Projekte betrachtet.

2.4.2 NetThreads

Ziel des NetThreads Projektes[9] war unter anderem softwarebasierte Deep Packet Inspection auf dem NetFPGA. Das Projekt wurde in mehreren Schritten weiterentwickelt. NetThreads basiert auf zwei vierfach Multithreaded MIPS Prozessorkernen die, wie in Abbildung 2.10 dargestellt, auf den FPGA-Ressourcen instantiiert werden [35]. Sie werden mit 125 MHz betrieben und die Synchronisation zwischen den Ausführungspfaden findet über Hardware-Mutexe statt. Eingehende und ausgehende Pakete werden durch die Hardware im hierfür vorgesehe-

[9] https://github.com/NetFPGA/netfpga/wiki/NetThreads

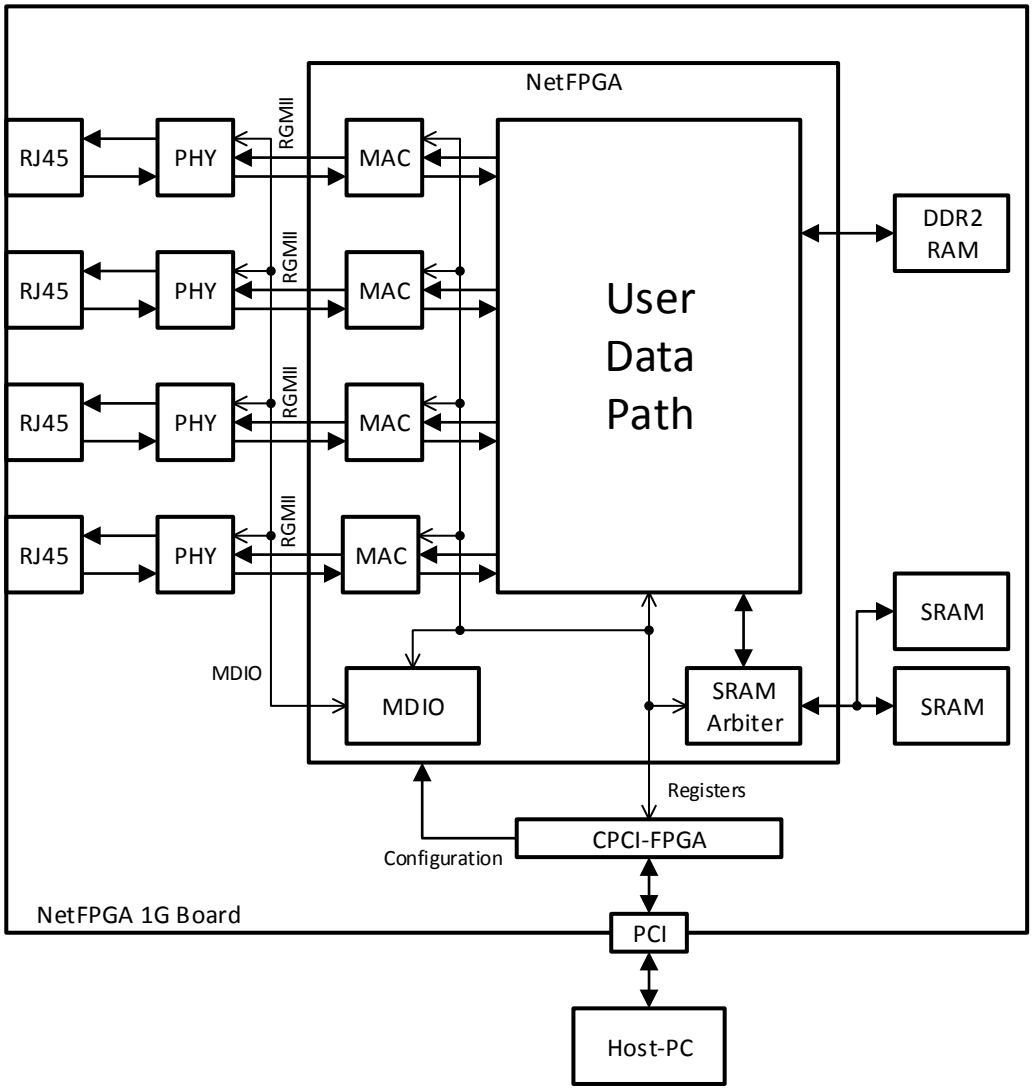


Abbildung 2.9: Komponenten auf dem NetFPGA 1G Board

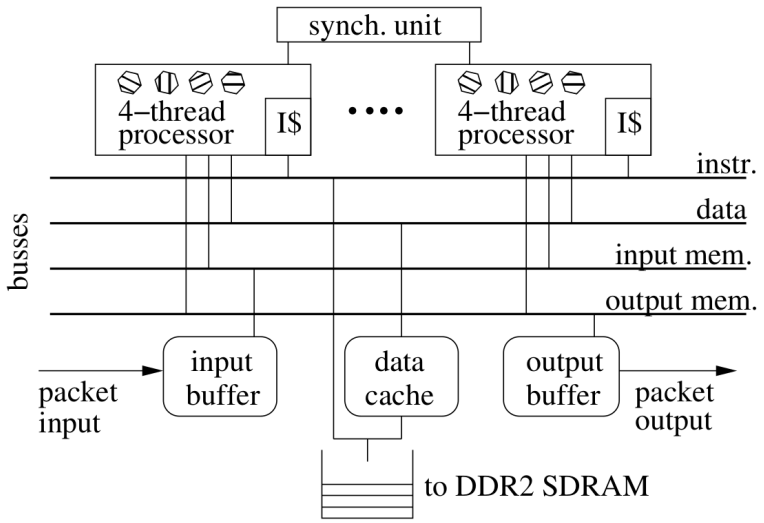


Abbildung 2.10: Übersicht über die NetThreads Prozessorkerne [35]

nen Speicher abgelegt beziehungsweise gelesen. Die Prozessoren können auf die gespeicherten Pakete, gemeinsamen Datenspeicher und eigenen Instruktionsspeicher zugreifen.

Die Weiterentwicklungen des Projekts NetThreadsRE[10] und NetTM[11] erweitern die Prozessorkerne unter anderem mit Synchronisationskonzepten über transaktionellen Speicher [34, 33]. Bei einer Bridge mit vier $100\ \mathrm{Mbit/s}$ Ethernet Ports müssen bis zu $5,95 * 10^5\ \mathrm{Pakete/s}$ verarbeitet werden. Dies ergibt sich aus einer minimalen Paketgröße von $84\ \mathrm{B} = 672\ \mathrm{bit}$ ($48\ \mathrm{B}$ minimale Nutzdaten, $12\ \mathrm{B}$ Ethernet Header, $4\ \mathrm{B}$ Frame Check Sum, $8\ \mathrm{B}$ Präambel, $12\ \mathrm{B}$ Inter Frame Gap). Aus [35] geht jedoch hervor, dass die maximal zu erreichende Paketrate $1,24 * 10^5\ \mathrm{Pakete/s}$ bei einem Prozessorkern beträgt. Weiterhin ist aktuell (Stand 27.12.2015) der Sourcecode für dieses Projekt nicht verfügbar. Daher wird das NetThreads Projekt im Rahmen dieser Arbeit nicht verwendet.

2.4.3 Network Code Switch

Ziel des Network Code Switch Projektes[12] war die Umsetzung von auf TDMA basierender, echtzeitfähiger Kommunikation über Ethernet [4]. Jedem Port ist zur Umsetzung der definierten Schedules, wie in Abbildung 2.11 dargestellt, ein ASIP (Application-Specific Instruction Set Processor) zugeordnet. Die einzelnen ASIPs kommunizieren über einen gemeinsamen Speicherbereich miteinander. Für die ASIPs wurde eine spezialisierte Maschinensprache mit

[10] https://github.com/NetFPGA/netfpga/wiki/NetThreadsRE
[11] https://github.com/NetFPGA/netfpga/wiki/NetTM
[12] https://github.com/NetFPGA/netfpga/wiki/RealTimeSwitch

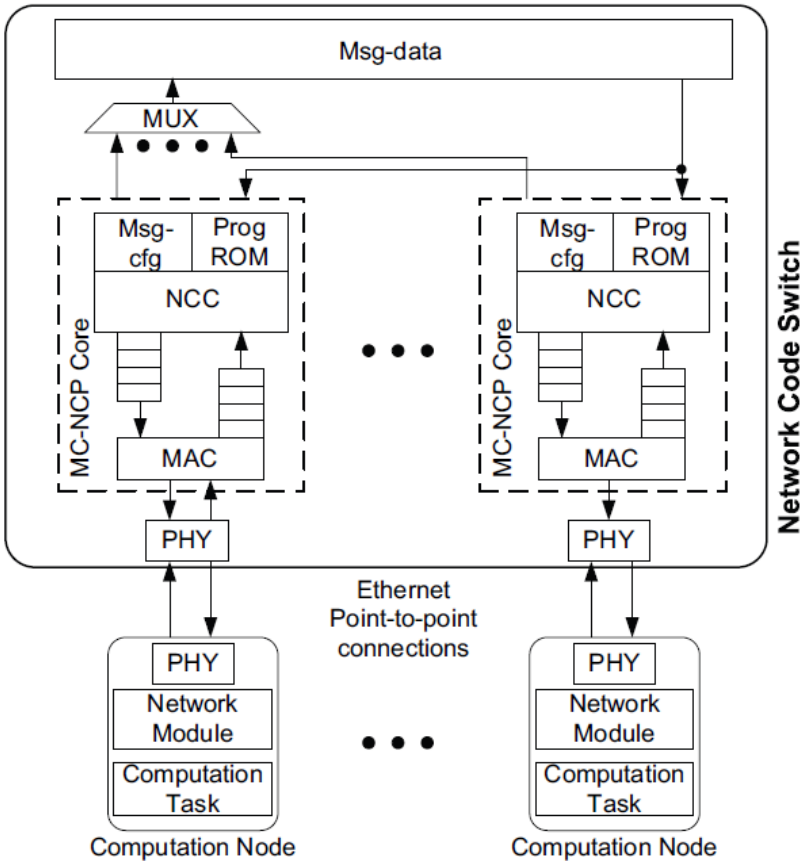


Abbildung 2.11: Übersicht über die Network Code Switch Prozessorkerne [4]

Befehlen zum Kontrollfluss, zur Datenmanipulation sowie zum Senden und Empfangen von Paketen definiert. Spezielle Instruktionen erlauben das Ausführen von Befehlen zu definierten Zeitpunkten. Wie in [4] angegeben, kann eine Paketrate von 100 $\mathrm{Mbit/s}$ bei Paketen mit minimaler Größe erreicht werden. Um die Nutzbarkeit des Network Code Switches Projektes für diese Arbeit zu bewerten, wurde es in die bestehenden Projektstrukturen eingegliedert und synthetisiert. Dabei ergab sich eine weitgehende Ausnutzung der FPGA Ressourcen durch den Network Code Switch. So wurden bereits 73 % der Slices und 58 % des BlockRAMs verwendet, wobei jedem ASIP lediglich ein Instruktionsspeicher von 256 $\mathrm{Wörtern}$ zugeordnet wurde. Weiterhin konnte durch die Toolchain kein Routing der Signale ermittelt werden, womit eine angestrebte Taktfrequenz von 125 MHz erreicht werden konnte. Da davon ausgegangen werden musste, dass die spezialisierte Maschinensprache zur Umsetzung weiterer Konzepte für Echtzeitkommunikation über Ethernet erweitert werden muss und hierfür die vorhandenen Ressourcen auf dem FPGA nicht ausreichend sind, wurde das Network Code Switch im Rahmen dieser Arbeit nicht verwendet.

2.5 Zielsetzung dieser Arbeit

Aus den vorherigen Abschnitten geht hervor, dass die in Kapitel 1 grob abgesteckten Ziele nicht mit verfügbaren Produkten und Projekten erreicht werden können. Lediglich das NetFPGA Projekt kann als Grundlage zum Erstellen eines Frameworks zum Evaluieren von Netzen mit mehreren Konzepten zur echtzeitfähigen Kommunikation über Ethernet verwendet werden. Dementsprechend befassen sich die folgenden Kapitel mit der Umsetzung eines entsprechenden Frameworks. Im Folgenden werden in Abschnitt 2.5.1 die gesetzten Ziele im Einzelnen diskutiert und in Abschnitt 2.5.2 die daraus resultierenden Anforderungen abgeleitet.

2.5.1 Ziele

Das Ziel dieser Arbeit ist ein Framework zum Erstellen von RTE Bridges mit Unterstützung für mehrere Konzepte zur echtzeitfähigen Kommunikation über Ethernet. Hiermit sollen bereits im Rahmen von Simulationen evaluierte Kombinationen von mehreren Konzepten zur echtzeitfähigen Kommunikation über Ethernet [38] als Prototypen in Zukunft umgesetzt und evaluiert werden. Da die Umsetzung von Prototypen in Hardware generell mit einem großen Aufwand verbunden ist, ist primäres Ziel dieser Arbeit den Aufwand bei der Umsetzung von aktuellen und zukünftigen Konzepten für RTE signifikant zu reduzieren. Idealerweise besteht somit die Umsetzung eines Konzeptes für RTE lediglich aus wenigen Komponenten, die in ei-

nem Umfeld mit klaren und intuitiven Konzepten und Interfaces eingebunden werden. Demgegenüber steht, dass mit den bestehenden Interfaces möglichst flexibel noch unbekannte Konzepte für RTE umzusetzen sind. Somit müssen die verwendeten Interfaces und dahinter stehenden Konzepte gleichzeitig möglichst intuitiv und leicht verständlich sein und eine hohe Flexibilität aufweisen. Weiterhin sollte, um mehrere Konzepte flexibel zu kombinieren, das Kombinieren mehrerer Konzepte nicht die zu einer Änderung der Implementierung von einzelnen Konzepten führen.

Das Framework soll den Bedarf von Prototypen für Fahrzeugnetze erfüllen, sollte aber auch in anderen Umfeldern einzusetzen sein. Damit mit Prototypen, die mit dem Framework umgesetzt wurden, aussagekräftige Evaluationen durchgeführt werden können, ist eine entsprechende Qualität des Frameworks zu gewährleisten. Um dies zu erreichen, sind Maßnahmen zur Qualitätssicherung zu ergreifen. Weiterhin ist das Erreichen der gesetzten Ziele und Anforderungen anhand von Messungen und Tests zu verifizieren. Ziel dieser Arbeit ist weiterhin die Erstellung eine Prototyps mit Unterstützung von TTE beziehungsweise TSN Scheduled Traffic und AVB, wie in [38] anhand von Simulationen evaluiert.

2.5.2 Anforderungen

Damit die Prototypen von Fahrzeugnetzen umgesetzt werden können, müssen die folgenden grundlegenden Anforderungen erfüllt werden:

- Deterministisches Verhalten bezüglich Latenz und Jitter im gesamten Paketpfad.
- Latenz von maximal 20 µs [60] bei minimaler Paketlänge.
- Jitter wesentlich kleiner als Latenz.
- Paket- und Datenrate nur durch Übertragungsrate der PHYs beschränkt.
- Reihenfolge der Nachrichten in einer Nachrichtenklasse ändert sich nur, wenn vom RTE Konzept vorgesehen.

Da mehrere Konzepte realisiert werden sollen, müssen für jeden Port mehrere Paketklassen mit eigenen Paketpuffern umgesetzt werden können. Jeder Paketklasse ist dabei eine Priorität und ein RTE-Konzept zuzuordnen. Wie in Abschnitt 2.3 erläutert, muss dabei die Arbitrierung der Paketklasse auf Prioritäten basieren. Dabei sollten RTE-Konzepte einen Paketversand in der Zukunft anmelden können. Bei der Zeitbasis für einen zukünftigen Paketversand sollte es sich dabei um eine global synchronisierte Zeitbasis handeln.

Um eine global synchronisierte Zeitbasis bereitzustellen, ist weiterhin eine hochpräzise Zeitsynchronisation umzusetzen. Um die Kompatibilität mit bestehenden Prototypen zu wahren, ist hierfür der in TTE verwendete Mechanismus zu realisieren. Die Umsetzung einer

hochpräzisen Zeitsynchronisation erfordert das Zeitstempeln von eingehenden und ausgehenden Paketen mit der aktuellen, global synchronisierten Zeitbasis in der Hardware. Weiterhin müssen bei den von TTE verwendeten Nachrichten zur Zeitsynchronisation Felder mit der akkumulierten Weiterleitungslatenz aktualisiert werden. Dementsprechend muss der Pfad zum Weiterleiten von Paketen eine Aktualisierung der entsprechenden Felder anhand der erstellten Zeitstempel unterstützen.

Ein Großteil der betrachteten Konzepte für RTE ordnet die unter Echtzeitbedingungen zu vermittelnden Pakete anhand der Zieladresse zu. Dementsprechend müssen die Pakete anhand ihrer Zieladresse mehreren Ausgangsports und einer Paketklasse zugeordnet werden. Weiterhin können Pakete von AVB und TSN Streams über VLAN Tags verfügen. Die VLAN Tags müssen entsprechend verarbeitet, hinzugefügt und entfernt werden.

AVB setzt die Umsetzung von Konfigurationsprotokollen wie MSRP voraus. Komplexe Konfigurationsprotokolle wie MSRP unterliegen im Gegensatz zur Vermittlung von Paketen unter Echtzeitbedingungen keinen Echtzeitanforderungen. Weiterhin würde eine Umsetzung dieser Protokolle in Hardware in einem großen Entwicklungsaufwand und vielen belegten Hardwareressourcen resultieren. Selbiges gilt für die Umsetzung von Protokollen für hochpräzise Zeitsynchronisation, sofern Konzepte bestehen, um die Präzision zu gewährleisten. Daher sollten entsprechende Protokolle in Software umgesetzt werden können.

Um Konfigurations- und Synchronisationsprotokolle in Software umzusetzen, ist ein eingebettetes Prozessorsystem zu integrieren. Das eingebettete Prozessorsystem sollte über Konfigurationsschnittstellen zur Konfiguration der Hardwarekomponenten verfügen. Weiterhin sollte es mit einer Schnittstelle zum Senden und Empfangen von Pakten auf allen Ports ausgestattet sein. Zur Umsetzung von Zeitsynchronisation und zur Bereitstellung einer global synchronisierten Zeitbasis sollte es über einen entsprechenden Timer verfügen. Im folgenden Kapitel wird nun ein Konzept zur Erfüllung der gestellten Ziele und Anforderungen erarbeitet.

3 Konzept

Am Ende des vorherigen Kapitels wurden die zu erreichenden Ziele und Anforderungen an das Framework gesetzt. In diesem Kapitel wird nun ein Konzept zum Erreichen der gesetzten Ziele und Anforderungen erarbeitet. Daraus wird die in Abbildung 3.1 dargestellte Architektur abgeleitet. Im Folgenden wird nun ein Überblick über die entwickelte Architektur gegeben und danach auf die einzelnen Komponenten und die dahinter stehenden Konzepte eingegangen.

Die umgesetzte Architektur ist an die Umsetzung der im NetFPGA Projekt enthaltenen Referenzumsetzung einer Ethernet Bridge (Reference Switch Projekt) angelehnt. Die Komponenten des Reference Switch Projektes sind in Abbildung 3.2 dargestellt. Da davon ausgegangen wurde, dass die in Abschnitt 2.4 beschriebenen, auf Prozessorsystemen basierenden Konzepte nur mit sehr hohem Entwicklungsaufwand die beschriebenen Anforderungen erfüllen, werden die RTE Konzepte vollständig in Hardware umgesetzt. Lediglich zur Konfiguration der Hardwarekomponenten wird ein eingebettetes Prozessorsystem verwendet. Die Komponenten der umgesetzten Architektur lassen sich in fünf Module zusammenfassen. Nachdem die eingehenden Pakete durch PHY und MAC empfangen wurden, werden sie an das Pre Processing Modul weitergeleitet. Die Komponenten im Pre Processing Modul extrahieren die benötigten Informationen aus den Paketen, versehen sie mit einem Eingangszeitstempel und legen sie im Pufferspeicher ab.

Die Pakete der eingehenden Ports werden dann durch den Input Arbiter in einen Paketstrom serialisiert und an die Filtering Database weitergeleitet. In der Filtering Database wird anhand von Regeln entschieden, an welche Ports die Pakete weitergeleitet und welcher Paketklasse sie zugeordnet werden. Durch den Output Port Multiplexer werden die Pakete an die Paket Buffering Module der ausgewählten ausgehenden Ports geleitet. Im Paket Buffering Modul werden die Pakete bis zum Versand gepuffert. Es beinhaltet die Implementierung der unterschiedlichen RTE Konzepte sowie Komponenten zur Verarbeitung der Pakete. Soll ein Paket versandt werden, so wird es an das Post Processing Modul weitergeleitet. Hier werden die Paketdaten aus dem Pufferspeicher ausgelesen, die nötigen Header generiert und die Pakete werden Ausgangszeitstempeln versehen. Bei Bedarf kann das *transparentClock* von PC-Frames anhand der Zeitstempel aktualisiert und ein Event bezüglich des versendeten Pa-

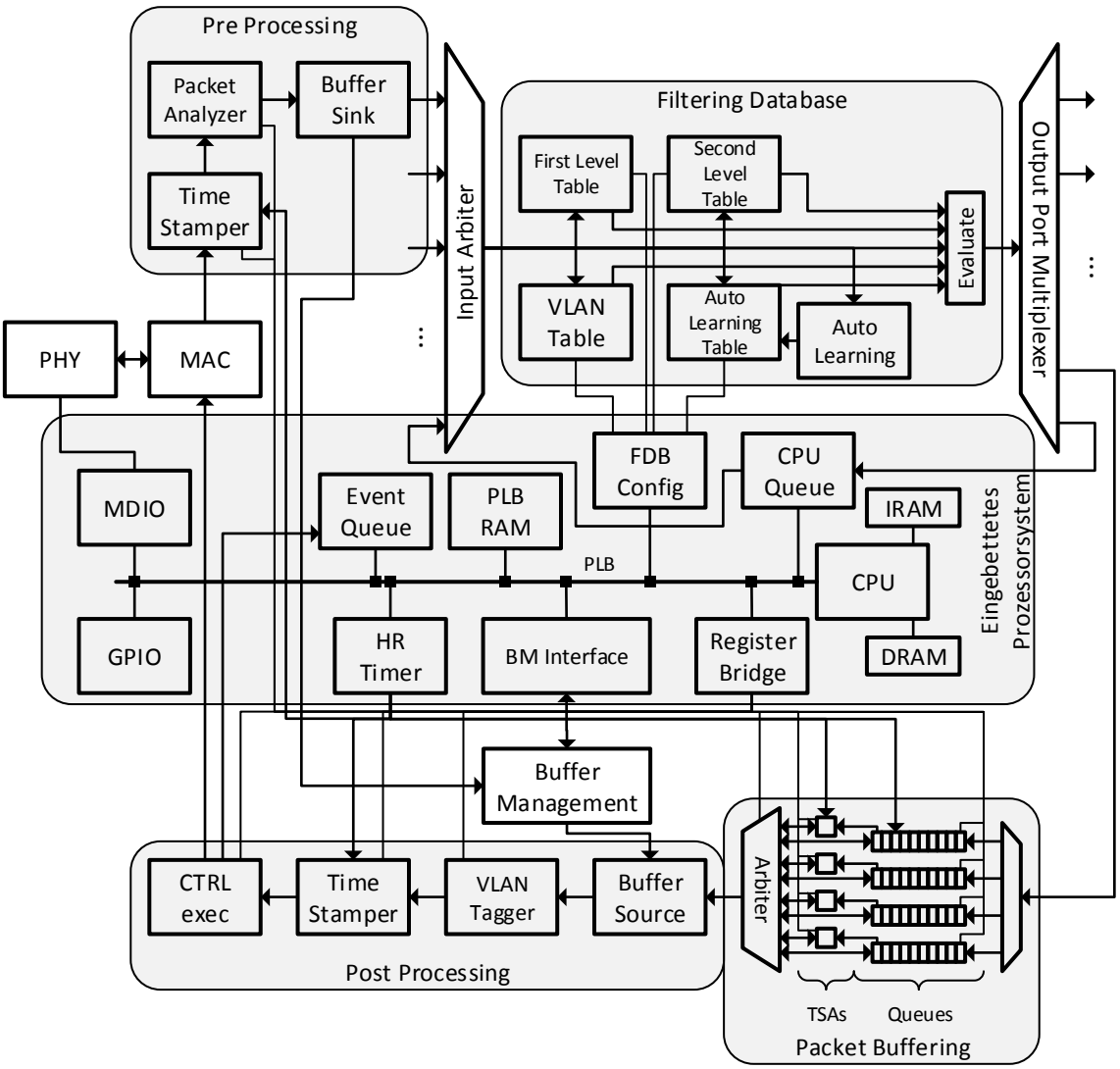


Abbildung 3.1: Übersicht über die entwickelte Architektur der RT-Bridge

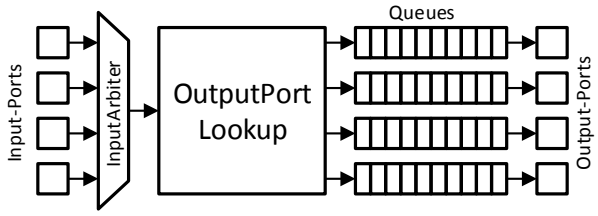


Abbildung 3.2: Komponenten der Referenzumsetzung einer Ethernet Bridge im NetFPGA (Reference Bridge Projekt)

ketes an das eingebettete Prozessorsystem übergeben werden. Zu sendende Pakete werden vom Post Processing Modul über MAC und PHY versandt. Das eingebettete Prozessorsystem verfügt über Konfigurationsinterfaces zu allen Komponenten, stellt über einen Timer eine global synchronisierte Zeitbasis bereit, erhält bei Bedarf Events beim Versand von Paketen und kann über eine Anbindung an den Paket Forwarding Pfad Pakete senden und empfangen.

Zwar ist die umgesetzte Architektur an das Reference Switch Projekt angelehnt, allerdings unterscheidet sie sich, wie im Folgenden beschrieben, wesentlich. Im Reference Switch Projekt werden die Paketdaten der Pakete vollständig durch den Forwarding Pfad der Bridge geleitet. Hierfür werden die Pakete aller Ports serialisiert und durch die, der Filtering Database entsprechenden, Output Port Lookup Komponente zum Puffern an die Ausgangsqueues weitergeleitet, bevor sie versandt werden. Ein eingehendes Paket muss bei der Serialisierung gegebenenfalls auf das Serialisieren von bis zu drei anderen Paketen mit einer maximalen Länge von $1518\ \mathrm{B}$ warten. Für die $64\ \mathrm{bit}$ breite Verarbeitungspipeline ergibt sich hier bei der Betriebsfrequenz $f = 125\ \mathrm{MHz}$ und der Paketlänge $l = 1518\ \mathrm{B}$ für jedes vorher serialisierte Paket die Verzögerung d:

$$
\begin{aligned}
d &= \frac{l}{64\ \mathrm{bit} * f} \\
&= \frac{1518\ \mathrm{B}}{64\ \mathrm{bit} * 125\ \mathrm{MHz}} \\
&= \frac{1518\ \mathrm{B}}{8\ \mathrm{B} * 125 * 10^6\ \mathrm{s}^{-1}} \\
d &= 1,518\ \mu\mathrm{s} \\
3 * d &= 4,554\ \mu\mathrm{s}
\end{aligned}
$$

Somit beträgt der maximale Jitter durch die Serialisierung der vollständigen Pakete $4,554\ \mu\mathrm{s}$, was bereits $25\ \%$ der als Ziel gesetzten Latenz von maximal $20\ \mu\mathrm{s}$ entspricht. Um diesen zu hohen Jitter zu vermeiden, werden die Nutzdaten der Pakete in einem hierfür vorgesehenen Pufferspeicher abgelegt. Alle im Rahmen des Paket Forwarding Pfades benötigten Informationen werden im Pre Processing Modul aus den Pakete extrahiert und zu einem intern genutzten Header zusammengefasst. Die intern genutzten Paketheader haben eine fixe Länge von $10\ \mathrm{Wörtern}$ mit je $32\ \mathrm{bit}$ und werden an Stelle der Pakete durch die Verarbeitungspipeline verarbeitet. Da die intern verwendeten Paketheader wesentlich kürzer als die durch sie repräsentierten Pakete sind, ergeben sich selbst bei maximaler Paketrate auf allen Ports Pausen zwischen den Paketheadern. Dementsprechend wird auf eine Flusskontrolle verzichtet, lediglich eine Leitung zum Signalisieren des ersten Wortes eines Paketheaders wird verwen-

det. Im Post Processing Modul werden die Pakete anhand der internen Paketheader und den im Pufferspeicher abgelegten Paketdaten zusammengesetzt. Das bis jetzt noch nicht genannte Buffer Management Modul verwaltet den Pufferspeicher und abstrahiert den Zugriff auf die einzelnen Paketpuffer. Im Folgenden wird die Funktionalität der einzelnen Komponenten in den Modulen beschrieben.

3.1 Pre Processing

Der Time Stamper schreibt die Ankunftszeit des Paketes in den internen Paketheader. Die Ankunftszeit ergibt sich aus dem Wert der aktuellen Zeitbasis korrigiert um einen über ein Register konfigurierbaren Wert. Bei den Time Stampern für das Pre Processing und Post Processing handelt es sich um die selbe Komponente, die lediglich anders parametrisiert ist.

Der Paket Analyzer extrahiert die Zieladresse, Quelladresse und VID aus dem Paket und generiert eine für den Port eindeutige Paket ID. Diese Daten werden im internen Paketheader zusammen mit dem Portindex abgelegt. Die an diesem Port für Pakete ohne VLAN Tag zu verwendete VID wird über ein Register konfiguriert. Die Buffer Sink allokiert einen Paketpuffer und legt die Paketdaten im Pufferspeicher ab. Im internen Paketheader hinterlegt die Buffer Sink eine ID für den verwendeten Paketpuffer.

3.2 Input Arbiter

Der Input Arbiter nimmt interne Paketheader von den den Ports zugeordneten Pre-Processing Modulen sowie dem eingebetteten Prozessorsystem entgegen. Die eingehenden Paketheader werden serialisiert an die Filtering Database weitergegeben. Da Paketheader von mehreren Ports an den Input Arbiter weitergeleitet werden können, werden die Paketheader vor der Serialisierung in einem Puffer abgelegt. Damit eine maximale Wartezeit für alle Ports garantiert werden kann, wird der nächste weiterzuleitende Paketheader nach dem Round-Robin Verfahren [56, S. 152] ausgewählt. Die von den Ports an den Input Arbiter übermittelte maximale Paketrate kann theoretisch die maximal zu verarbeitende Paketrate des Input Arbiters übersteigen. Im Folgenden wird gezeigt, dass dies auf Grund der maximalen Paketrate der Ethernet Ports nicht möglich ist.

An einem Port können bei einer Datenrate des Ports $d = 1\ \mathrm{Gbit/s}$ maximal $\frac{d}{l_{min}} \approx 1,488 * 10^6\ \mathrm{Pakete/s}$ eintreffen. Bei vier Ports ergeben sich maximal etwa $5,952 * 10^6\ \mathrm{Pakete/s}$, die durch das eingebettete Prozessorsystem generierte Paketrate ist kontrollierbar. Die Minimale Paketlänge $l_{min} = l_{IFG} + l_{Präambel} + l_{Header} + l_{Data} + l_{FCS} = 84\ \mathrm{B}$

ergibt sich aus $l_{IFG} = 12\,\mathrm{B}$, $l_{Präambel} = 8\,\mathrm{B}$, $l_{Header} = 14\,\mathrm{B}$, $l_{Daten} = 46\,\mathrm{B}$ sowie $l_{FCS} = 4\,\mathrm{B}$. Bei einer Länge der Paketheader von 10 Wörtern und 125 MHz können bis zu $1,25 * 10^7$ $\mathrm{Paketheader/s}$ verarbeitet werden. Da die maximale Paketrate wesentlich über der maximal auftretenden Paketrate liegt können keine Paketverluste auftreten. Auch wenn die Verarbeitungsfrequenz des Input Arbiters auf $62,5$ MHz halbiert wird, können keine Paketverluste auftreten.

3.3 Filtering Database

Wie in Kapitel 2.5 erläutert, müssen die eingehenden Pakete anhand ihrer Zieladresse mehreren Ausgangsports zugeordnet werden. Weiterhin ist den Paketen eine Paketklasse zuzuordnen und zu beachten, welches VLAN an welchem Port aktiv ist. Das Ergebnis der Operation der Filtering Database (FDB) wird im internen Paketheader in den entsprechenden Feldern abgelegt. Dabei handelt es sich um die zentrale Funktionalität einer Ethernet Bridge, da hierdurch mehrere Ethernet Segmente zusammengefügt werden. Pakete werden nicht an alle angeschlossenen Segmente und somit Endstationen weitergeleitet, sondern nur an die, für die das Paket bestimmt ist. Dies führt zu einer geringeren Auslastung der Segmente sowie Endstationen und trägt somit zu einer besseren Skalierbarkeit von Ethernet bei. Im IEEE Standard 802.1Q [14, Kapitel 8.8] wird hierfür das Konzept der FDB eingeführt. Sie beinhaltet Regeln, anhand derer entschieden wird, an welche Ports ein Paket weitergeleitet wird. Im einfachsten Fall beinhalten diese Regeln lediglich die Zieladresse, auf die sie zutreffen, sowie das Zielsegment. Die Regeln werden in diesem Fall automatisch anhand der Quelladressen der Pakete, die von einem Segment empfangen wurden, gelernt.

Im IEEE Standard 802.1Q sind sechs unterschiedliche Regeln vorgesehen, es wird zwischen statischen und dynamischen Regeln unterschieden. Bei statischen Regeln handelt es sich um manuell konfigurierte Regeln. Dynamische Regeln sind das Resultat aus dem Betrieb unterschiedlicher Protokolle oder dem automatischen Lernen von Adressen. Jede Regel enthält eine Beschreibung der Pakete, auf die sie zutrifft, sowie eine PortMap. In der PortMap ist für jeden Port definiert, ob zutreffende Pakete gefiltert (Filter) oder weitergeleitet (Forward) werden sollen. Da statische den dynamischen Regeln bevorzugt werden, kann bei statischen Regeln vermerkt werden, dass auf eventuell vorhandene dynamische Regeln zurückgegriffen werden soll. Weiterhin bestehen Regeln, die die Zugehörigkeit von Ports zu VLANs definieren und welches VLAN für Pakete ohne VLAN Tag (untagged) an einem Port verwendet wird. Wildcard Regeln beinhalten keine konkreten Zieladressen, sondern gelten für eine Menge von Zieladressen. Sie gelten entweder für Gruppenadressen, auf die keine andere Regel zu-

Signal	Beschreibung
notEmpty	Zugeordnete Queue beinhaltet mindestens ein Paket.
transmit	Zugeordnete Paketklasse sendet aktuell.
transmitAllowed	Zugeordnete Paketklasse ist sendebereit.

Tabelle 3.1: Aus IEEE 802.1Q [14, Kapitel 8.6.8] hervorgehendes Interface der TSAs

trifft (all unregistered Group), für alle Gruppenadressen oder alle individuellen Adressen (all Group/Individual).

Wie ersichtlich, ist das Interface der FDB klar definiert. Spezifikationen von AVB und TSN verwenden das Interface der FDB. Mit den verfügbaren Regeln lassen sich die virtuellen Links von AFDX und TTE realisieren. Wird eine in Hardware umgesetzte und auf Seiten der Software das spezifizierte Interface verwendende FDB realisiert, wird die nötige Flexibilität erreicht. Weiterhin werden die Pakete durch die FDB einer Paketklasse zugeordnet und die in der CTRL Exec auszuführenden Operationen festgelegt.

3.4 Output Port Multiplexer

Der Output Port Multiplexer leitet die Paketheader anhand der Ergebnisse der FDB an die Paket Buffering Module der jeweiligen Ports weiter. Dabei muss dafür gesorgt werden, dass festgehalten wird, an wie vielen Stellen eine Referenz auf den Paketpuffer gehalten wird. Das Konzept zum Verwalten der Referenzen auf die Paketpuffer wird im Abschnitt 3.7 bezüglich des Buffer Managements beschrieben. Der Output Port Multiplexer gibt an das Buffer Management die Anzahl der neuen Referenzierungen weiter.

3.5 Paket Buffering

Im Paket Buffering Modul werden einerseits die Pakete vor dem Versand gepuffert und andererseits die RTE Konzepte umgesetzt. Die Umsetzung eines RTE Konzepts ergibt sich pro Paketklasse aus zwei Komponenten: einer Queue zum Puffern der Pakete sowie einem Transmission Selection Algorithm (TSA) zur Steuerung des Paketversandes. Die TSA Komponente ist an den im IEEE 802.1Q Standard [14, Kapitel 8.6.8] beschriebenen TSA angelehnt. Das daraus hervorgehende Interface beinhaltet die in Tabelle 3.1 gelisteten Signale. Anhand dieses Interfaces ist für die TSAs allerdings nicht ersichtlich, ob aktuell eine andere Paketklasse sendet oder zurzeit keine Pakete gesendet werden. Weiterhin kann ein Beginn des nächsten

Signal	Beschreibung
not_empty	Zugeordnete Queue beinhaltet mindestens ein Paket.
transmit	Zugeordnete Paketklasse sendet aktuell.
busy	Zugeordneter Port sendet aktuell.
send_request	Zugeordnete Paketklasse ist sendebereit.
next_frame	Beginn des nächsten sendebereiten Pakets.

Tabelle 3.2: Signale des Interfaces der TSAs

Signal	Beschreibung
not_empty	Queue beinhaltet mindestens ein Paket.
next_len	Länge des nächsten Pakets.
transmit	Nächstes Paket senden.

Tabelle 3.3: Signale des Interfaces der Queues

Paketes in der Zukunft nicht signalisiert werden. Dementsprechend wurde das Interface, wie in Tabelle 3.2 gelistet, angepasst.

Die Queues melden, ob ein Paket sendebereit ist, und geben dessen Länge an. Die Queue wird durch ein entsprechendes Signal zum Übermitteln des aktuell sendebereiten Paketes aufgefordert. Dies führt zu dem in Tabelle 3.3 beschriebenen Interface. Den Queues wird durch ein Signal vom Buffer Management signalisiert, wenn Pakete mangels Pufferspeichers verworfen werden müssen. Damit Pakete durch die Queues verworfen werden können, können sie weiterhin dem Buffer Management die Freigabe eines Paketpuffers signalisieren.

Zudem steht den TSAs und Queues die global synchronisierte Zeitbasis zur Verfügung. Sie können über acht Register durch das eingebettete Prozessorsystem konfiguriert werden. Die Pakete werden durch den Queue Multiplexer an die Queue für die entsprechende Paketklasse übergeben.

Durch den Queue Arbiter wird bestimmt, welche Paketklasse das nächste Paket senden darf. Der Queue Arbiter wählt das Paket aus, das als nächstes sendebereit ist und vor allen angemeldeten zukünftigen Paketen mit höherer Priorität zu Ende ist. Melden zwei Paketklassen ein Paket zum selben Zeitpunkt an, wird das Paket mit der höheren Priorität ausgewählt. Der Beginn der Pakete wird direkt durch die TSAs gemeldet, das Ende eines Paketes ergibt sich aus dem Beginn, seiner Länge und der aktuellen Bandbreite. Die aktuelle Bandbreite wird in Form der beim Senden benötigten Zeit für ein Byte in einem Register konfiguriert. Die Priorität der Paketklassen ergibt sich implizit aus dem genutzten Anschluss am Queue Arbiter. Der kleinste Index verfügt über die höchste Priorität. Der Queue Arbiter beginnt erst mit der

Auswahl des nächsten Paketes, wenn der errechnete Endzeitpunkt des zuvor ausgewählten Paketes erreicht ist.

Im Folgenden werden Beispiele für die Umsetzung der RTE Konzepte CBS und TSN Scheduled Traffic gegeben. Soll CBS umgesetzt werden, wird als Queue eine FIFO verwendet. Trifft ein Paketheader bei der Queue ein, wird er in die FIFO geschrieben. Das Signal *not_empty* ergibt sich durch Negieren des *empty* Ausgangs der FIFO. Ist die FIFO nicht leer, wird der erste Paketheader aus der FIFO gelesen, in einem Schieberegister gepuffert und die Paketlänge extrahiert. Die Länge des Paketes auf der Leitung ergibt sich durch Addieren der Länge von IFG, FCS und Header und wird an das Signal *next_len* angelegt. Wird eine positive Flanke am *transmit* Eingang der Queue registriert, wird der gepufferte Paketheader ausgegeben und gegebenenfalls der nächste aus der FIFO gelesen. Der TSA verfügt über Register zum Konfigurieren des Send und Idle Slopes. In einem Register wird der aktuelle *credit* gehalten und entsprechend der im Standard definierten Regeln aktualisiert. Das Signal *send_request* ergibt sich aus $not\_empty \wedge (credit >= 0)$. Als Beginn des nächsten sendebereiten Paketes wird immer die aktuelle Zeit weitergegeben.

Soll TSN Scheduled Traffic umgesetzt werden, wird wie bei CBS eine FIFO als Queue verwendet. Bei Bedarf kann aber auch eine andere Queue implementiert werden, beispielsweise mit eigenen Pufferslots für jede RTE Nachricht. Dem TSA wird über die Register durch das eingebettete Prozessorsystem laufend der Schedule zugespielt. Er beinhaltet für jedes Sendefenster den Start- und Endzeitpunkt. Wenn ein Scheduleeintrag vorliegt, wird der Startzeitpunkt an *next_frame* angelegt und *send_request* gesetzt. Der nächste Scheduleeintrag wird gelesen, wenn der Endzeitpunkt erreicht wurde.

3.6 Post Processing

Durch die Buffer Source des Post Processing Moduls wird der Ethernet Header mit Ziel und Quelladresse generiert und die Paketdaten werden aus dem Paketpuffer gelesen. Der VLAN Tagger fügt dem Paket bei Bedarf einen VLAN Tag hinzu. Hierzu wird die an diesem Port für Pakete ohne VLAN Tag zu verwendende VID in einem Register konfiguriert. Der Timestamper entspricht weitgehend dem Timestamper im Pre Processing Modul, die Zeitstempel werden lediglich in einem anderen Feld des internen Paketheaders abgelegt.

Die CTRL Exec Komponente führt bei Bedarf zwei mögliche Operationen aus. Einerseits kann das *transparentClock* Feld der PC-Frames anhand der im internen Paketheader abgelegten Eingangs- und Ausgangszeitstempel aktualisiert werden. Da zum Aktualisieren des *transparentClock* Felds der Propagation Delay bekannt sein muss, kann dieser über ein Register

konfiguriert werden. Dieses Register kann weiterhin zum Korrigieren des gemessenen Dynamic Delays verwendet werden. Außerdem kann für ein ausgehendes Paket ein Event mit beiden Zeitstempeln, dem Index des eingehenden Ports sowie der eindeutigen Paket ID an das eingebettete Prozessorsystem geleitet werden. Die Aktivierung der Operationen wird durch ein Feld im internen Paketheader gesteuert. Das Feld wird durch die FDB gesetzt und ist in entsprechenden Regeln vermerkt.

3.7 Buffer Management

Wie bereits erläutert, sorgt das Ablegen der Pakete in einem Pufferspeicher für einen geringeren Jitter. Der zu verwendende Speicher wird in Pufferslots mit gleicher Größe aufgeteilt. Jeder Pufferslot wird durch eine ID eindeutig identifiziert. Diese ID wird im internen Paketheader abgelegt. Pufferslots müssen beim Empfang allokiert und beim Senden wieder freigegeben werden. Dementsprechend muss verwaltet werden, ob Pufferslots frei oder belegt sind. Durch den Output Port Multiplexer können Pakete aber auch an mehrere Ports weitergeleitet werden (Multicast/Broadcast). Pakete könnten dann zwar in weitere Pufferslots kopiert werden, dies würde aber zu einer höheren benötigten Bandbreite am Pufferspeicher und zusätzlichem Jitter durch das Kopieren führen. Durch die Verwendung von zählenden Referenzen wird dies verhindert [30, S. 413 f]. Das Buffer Management sorgt einerseits für die Verwaltung der Pufferslots mit Hilfe der zählenden Referenzen. Andererseits abstrahiert es den Zugriff auf die Pufferslots.

Das Buffer Management sorgt weiterhin für die Arbitrierung der Zugriffe von einer konfigurierbaren Anzahl von Endpunkten. Um auf einen Mangel an Pufferslots zu reagieren, wird die Anzahl der freien Pufferslots mit mehreren über Register zu konfigurierenden Thresholds verglichen. Jeder Paketklasse ist ein Threshold zugewiesen. Liegt die Anzahl der freien Pufferslots unter dem Threshold, so müssen die der Paketklasse zugeordneten Queues Pakete verwerfen und somit Pufferslots freigeben.

3.8 Eingebettetes Prozessorsystem

Da auf dem Virtex II Pro 50 FPGA zwei PowerPC 405 Prozessorkerne vorhanden sind, wird einer der Prozessoren für das eingebettete Prozessorsystem verwendet. Daten- und Instruktionsspeicher werden auf dem FPGA instantiiert. Das eingebettete Prozessorsystem kann über an den Processor Local Bus (PLB) angeschlossene Registerinterfaces auf die Komponenten der RT-Bridge zugreifen. Über FDB Config wird die Filtering Database konfiguriert, in Event

Queue werden Events der CTRL Exec Komponente gepuffert, MDIO erlaubt die Konfiguration der PHYs und das Buffer Management Interface mappt einerseits den Pufferspeicher in den Adressbereich des Prozessorssystems und erlaubt andererseits das Allozieren und Freigeben der Pufferslots. Der Timer wird zur Bereitstellung einer global synchronisierten Zeitbasis verwendet und stellt diese den Komponenten der RT-Bridge bereit. Die CPU Queue erlaubt das Senden und Empfangen von Paketen. Für die übrigen Komponenten der RT-Bridge stellt die Register Bridge ein vereinfachtes Interface für Register bereit. Weiterhin verfügt das Prozessorsystem mit den Config und Stdio FIFOs eine Schnittstelle zur Kommunikation mit dem Host-PC. Interrupts im Prozessorsystem können durch den Host-PC, die Event Queues, die CPU Queue sowie den Timer ausgelöst werden.

4 Umsetzung

Im vorherigen Kapitel 3 wurde ein Konzept zu Erfüllung der in Kapitel 2.5 gestellten Anforderungen und Ziele entwickelt. In diesem Kapitel wird nun auf die Umsetzung des Konzeptes auf Basis der NetFPGA 1G Plattform eingegangen. Die folgenden Abschnitte befassen sich jeweils mit der Umsetzung einer Komponentengruppe. Dabei wird erst auf die wesentlichen verwendeten Interfaces eingegangen und danach auf die Umsetzung. In Abschnitt 4.1 wird die Anpassung der verwendeten RGMII Schnittstellen sowie die Taktgenerierung dargestellt. In Abschnitt 4.2 wird auf die Umsetzung der Pre und Post Processing Komponenten eingegangen. Abschnitt 4.3 setzt sich mit der Serialisierung der von mehreren Ports kommenden Pakete, der Umsetzung der Filtering Database sowie der Übergabe der Pakete an die durch die FDB ermittelten Ports auseinander. Abschnitt 4.4 geht auf die Umsetzung der, die RTE Konzepte beinhaltenden Paket Buffering Komponenten ein, Abschnitt 4.5 behandelt die Umsetzung des Puffermanagements. Der Abschnitt 4.6 befasst sich mit dem umgesetzten eingebetteten Prozessorsystem sowie Abschnitt 4.7 mit der darauf ausgeführten Software. Auf die für die Umsetzung verwendete Toolchain und auf die an ihr durchgeführten Anpassungen wurde im Rahmen der Ausarbeitung zu Projekt 1 [48] eingegangen.

4.1 Taktgenerierung

Die für die Evaluation dieser Arbeit zur Verfügung stehenden Endgeräte unterstützen mit Fast Ethernet eine maximale Datenrate von $100\ \mathrm{Mbit/s}$. Bei den verfügbaren Bridges mit Unterstützung für TTE trifft dies ebenfalls zu. Damit die RT-Bridge mit diesen Netzwerkknoten evaluiert werden kann, muss sie also Fast Ethernet unterstützen. Die auf dem NetFPGA Board vorhandenen PHYs können durch entsprechende Konfiguration umgestellt werden. Allerdings muss auch die Signalisierung umgestellt werden. Da die Umstellung der Signalisierung eine Änderung der Taktrate sowie der Verwendung der Signale beinhaltet, müssen die hier beschriebenen Änderungen durchgeführt werden. Weiterhin muss auf Grund der hohen Ressourcenauslastung auf dem FPGA der Kerntakt von $125\ \mathrm{MHz}$ auf $62,5\ \mathrm{MHz}$ reduziert werden. Im Folgenden Abschnitt 4.1.1 wird erst das zu Grunde liegende Reduced Gigabit Media Independent Interface (RGMII) beschrieben und die sich daraus ergebenden Änderungen

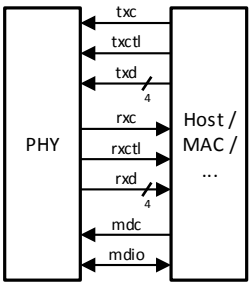


Abbildung 4.1: Signale des RGMII Interfaces

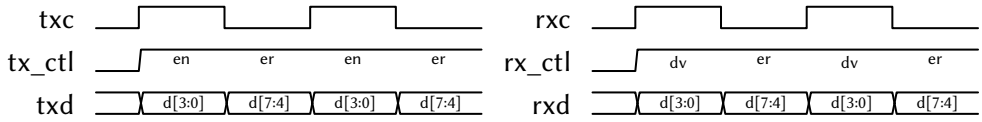
erläutert. In Abschnitt 4.1.2 werden die auf den Viretex II Pro FPGA zur Verfügung stehenden Komponenten zur Taktgenerierung und Verteilung beschrieben. In Abschnitt 4.1.3 wird auf die durchgeführten Anpassungen der MACs und der zugehörigen Komponenten zur Taktgenerierung eingegangen. Im letzten Abschnitt 4.1.4 werden die Änderungen zur Umstellung des Kerntaktes erläutert.

4.1.1 Reduced Gigabit Media Independent Interface

Die PHYs auf dem NetFPGA Board sind über das RGMII angebunden. Die auf dem RGMII verwendete Signalisierung ist an das im IEEE Standard 802.3 [18] spezifizierten GMII (Gigabit Media Independent Interface) angelehnt und wurde durch die Unternehmen Broadcom, HP und Marvell gemeinsam festgelegt[44]. Im Gegensatz zum GMII wird beim RGMII zur Reduzierung der Anzahl der Signale eine Signalisierung mit Double Data Rate (DDR), also mit Signalflanken sowohl zur steigenden als auch zur fallenden Taktflanke verwendet. Im Folgenden wird erst auf die verwendeten Signale und dann auf die Signalisierung bei den unterschiedlichen Datenraten eingegangen.

Wie aus Abbildung 4.1 hervorgeht, werden zur Übermittlung von ausgehenden Nachrichten vom MAC zum PHY die Leitungen *txc*, *txctl* sowie *txd[4]* verwendet. Zur Übermittlung von eingehenden Nachrichten vom PHY zum MAC werden die Leitungen *rxc*, *rxctl* sowie *rxd[4]* verwendet. Weiterhin steht mit den Leitungen *mdc* und *mdio* das MDIO Interface zur Konfiguration des PHYs bereit. Auf die Signalisierung des MDIO Interfaces wird hier nicht weiter eingegangen, sie ist in Kapitel 22.2.4 Des IEEE 802.3 Standards [18] spezifiziert. Alle für die ausgehenden Nachrichten verwendeten Signale werden vom MAC getrieben, die für die eingehenden Signale vom PHY.

Die Signalisierung für eingehende und ausgehende Nachrichten ist gleich, lediglich die Bezeichnung der Signale unterscheidet sich. Die Taktleitungen *txc* und *rxc* werden bei $1000\ \mathrm{Mbit/s}$ Datenrate mit $125\ \mathrm{MHz}$ betrieben. Pakete beginnen immer zur steigenden Taktflanke. Über



(a) Ausgehendes Paket (b) Eingehendes Paket

Abbildung 4.2: Paketbeginn bei 1000 Mbit/s RGMII Signalisierung

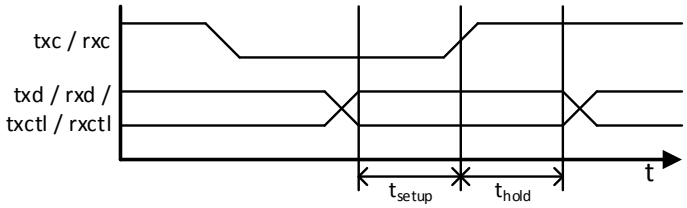


Abbildung 4.3: Setup- und Hold-Zeiten für RGMII Signalisierung

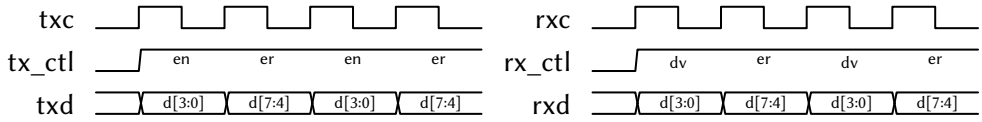
die Datenleitungen wird pro Takt ein Byte übertragen. Zur steigenden Taktflanke werden die ersten vier Bits, zur fallenden Taktflanke die letzten vier Bits übertragen. Auf der Kontrollleitung liegt zur steigenden Taktflanke das Signal *tx_en* beziehungsweise *rx_dv* an. Zur fallenden Taktflanke liegen die Signale *tx_err* und *rx_err* an. Die Signale *tx_en* und *rx_dv* zeigen die Validität der Paketdaten an, *tx_er* und *rx_er* ergeben sich aus:

$$tx\_er = tx\_en \veebar gmii\_tx\_er$$
$$rx\_er = rx\_dv \veebar gmii\_rx\_er$$

Die Signale *gmii_rx_er* und *gmii_tx_er* dienen zum Anzeigen und Propagieren von Fehlern beim Paketempfang. Der Beginn eines Paketes ist beispielhaft in Abbildung 4.2 dargestellt.

Sollten allerdings die Signale wie abgebildet beim Empfänger ankommen, würde dies zu einer Verletzung der Setup- und Hold-Zeiten der die Signale aufnehmenden Flipflops führen. Das Taktsignal ist laut [44] durch entsprechende Leitungslängen um 1 ns bis 2,6 ns zu verzögern. Die Setup- und Hold-Zeiten ergeben sich wie in Abbildung 4.3 dargestellt. Für die Signale *txd* und *tx_ctl* müssen t_{setup} und t_{hold} 1,2 ns bis 2 ns betragen, für *rxd* und *rx_ctl* 1 ns bis 2 ns.

Sollen über RGMII Daten mit 100 Mbit/s oder 10 Mbit/s Datenrate übertragen werden, werden die Taktleitungen entsprechend dem Media Independent Interface (MII) mit 25 MHz beziehungsweise 2,5 MHz betrieben. Die Signale werden nicht mit DDR signalisiert. Zu den steigenden Taktflanken werden an die Datenleitungen *txd* und *rxd* jeweils vier Bit pro Takt angelegt, an die Leitungen *tx_ctl* und *rx_ctl* wird jeweils abwechselnd *tx_en* und *rx_dv* sowie



(a) Ausgehendes Paket (b) Eingehendes Paket

Abbildung 4.4: Paketbeginn bei 100 Mbit/s und 10 Mbit/s RGMII Signalisierung

tx_er und *rx_er* angelegt. Pakete beginnen mit einer positiven Flanke des *tx_ctl* beziehungsweise *rx_ctl* Signals. Der Beginn eines Paketes ist beispielhaft in Abbildung 4.4 dargestellt. Die Setup- und Hold-Zeiten sind für die Datenraten 100 Mbit/s oder 10 Mbit/s nicht angegeben, müssen aber eingehalten werden.

4.1.2 Taktgenerierung auf Virtex II Pro FPGAs

Auf dem verwendeten Virtex II Pro 50 FPGA sind eine Reihe von Komponenten zur Bereitstellung von Taktsignalen verfügbar. Sie sind in den Dokumenten DS083[29] und UG012[65] des Unternehmens Xilinx dokumentiert. Die Ressourcen zum Bereitstellen von Taktsignalen auf dem FPGA sind in Abbildung 4.5 dargestellt. Taktsignale werden über einen von 16 speziellen IO-Pads (GCLK PAD) über Input-Buffer (IBUFG) in den FPGA geleitet. Acht der, auf Taktsignale spezialisierten, GCLK Pads sind am oberen und acht sind am unteren Ende des FPGAs verfügbar. Durch acht Digital Clock Manager (DCM) können Taktsignale manipuliert und aufbereitet werden. Vier DCM sind am oberen sowie weitere vier sind am unteren Ende des FPGAs verfügbar. Jeder DCM kann ein Taktsignal um 0° (*CLK0*), 90° (*CLK90*), 180° (*CLK180*) und 270° (*CLK270*) phasenverschoben wieder ausgeben. Das angelegte Taktsignal kann verdoppelt (*CLK2X*, *CLK2X180*) oder durch einen Wert der Form 1,5, 2, 2,5, … 15, 16 geteilt (*CLKDV*) wieder ausgegeben werden. Weiterhin kann das angelegte Taktsignal mit einer Ganzzahl multipliziert und durch eine Ganzzahl geteilt (*CLKFX*, *CLKFX180*) wieder ausgegeben werden. Mit Hilfe des an den *CLKFB* Eingang zurückgeführten verteilten Taktsignals gleicht der DCM die Phasen des verteilten und eingehenden Taktsignals an.

Die durch die DCM aufbereiteten Taktsignale werden über 16 BUFGMUX an Verteilbäume weitergegeben. Taktsignale können auch direkt vom IBUFG an BUFGMUX weitergeleitet werden. Jeder BUFGMUX beinhaltet einen globalen Taktpuffer und einen Multiplexer zum Umschalten des verwendeten Taktsignals. Jeweils acht BUFGMUX sind am oberen und unteren Ende des FPGAs verfügbar. Je zwei nebeneinander liegende BUFGMUX können aus insgesamt zwei Taktquellen umschalten. Weiterhin werden die BUFGMUX zu sich je auf dem oberen und unteren Ende gegenüberliegenden Paaren gruppiert.

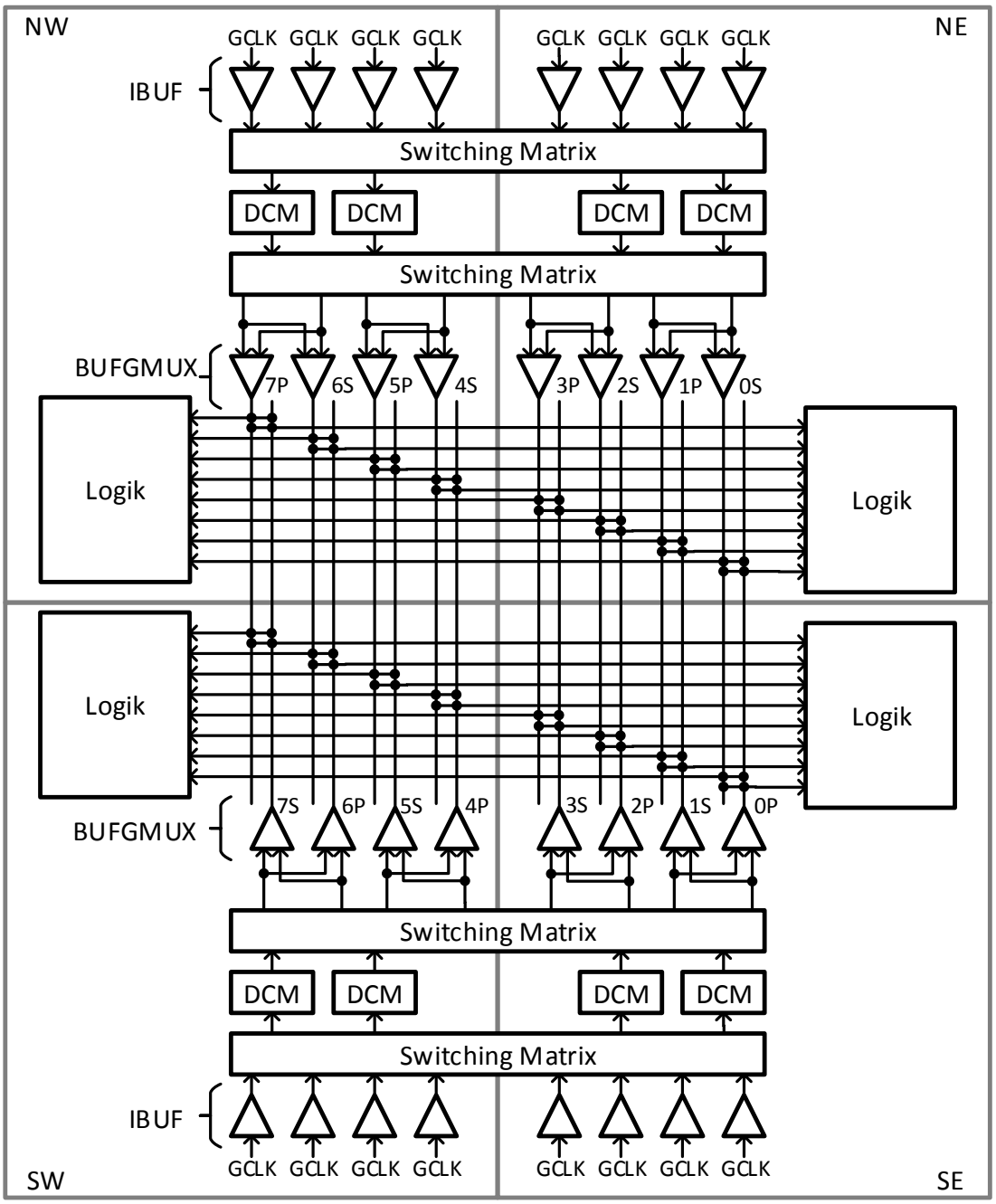


Abbildung 4.5: Ressourcen zum Bereitstellen von Taktsignalen auf Virtex II Prio 50 FPGA des Unternehmens Xilinx (nach [29, 65])

Die Virtex II Pro FPGA sind in die Quadranten NW, NE, SE sowie SW unterteilt. In jedem Quadranten stehen acht Taktverteilbäume zur Verfügung. Die Verteilbäume verteilen das Taktsignal mit gleichmäßiger Verzögerung an die instantiierte Logik. Jeder Verteilbaum ist einem Paar von BUFGMUX zugeordnet und kann wahlweise von einem der beiden das Taktsignal erhalten. Taktsignale können über normale IO-Pads aus dem FPGA heraus geführt werden, solange das Taktsignal im Quadranten des IO-Pads verfügbar ist.

4.1.3 Anpassung der Taktgenerierung und der RGMII Schnittstellen

Im Folgenden wird daraus eine Anpassung der bestehenden Komponenten erarbeitet. Hierzu wird erst der Ausgangszustand und danach werden die Anpassungen beschrieben.

In der ursprünglichen Umsetzung wurden die RGMII Signale in, dem IEEE 802.3 Standard [18] entsprechende, GMII Signale umgewandelt. In Abbildung 4.6 sind die zur Taktgenerierung und zur Anbindung und Wandlung der RGMII Signale verwendeten Komponenten dargestellt. Aus Tabelle 4.1 wird weiterhin ersichtlich, in welchem Quadranten die für das RGMII verwendeten IO-Signale auf dem FPGA lokalisiert sind. Somit verwendet die implementierte MAC lediglich GMII Signale, wobei diese den Takt der zugehörigen RGMII Signale verwendeten. Innerhalb der MACs findet eine Taktsynchronisation zum im NetFPGA verwendeten Kerntakt statt. Dementsprechend wird in jeder MAC der Kerntakt sowie jeweils ein Takt für eingehende und ausgehende GMII Signale verwendet. Da der Takt für die eingehenden RGMII Signale von den einzelnen PHYs vorgegeben wird und die Taktsignale unabhängig voneinander sind, wird für jedes RGMII und die dazugehörige MAC das entsprechende Taktsignal separat aufbereitet und verteilt. Die eingehenden RGMII Signale werden jeweils zu den steigenden und fallenden Taktflanken in Registern übernommen, über zwei weitere Register auf die steigenden Taktflanken synchronisiert und daraus die GMII Signale gebildet.

Für alle ausgehenden RGMII Signale wird die vom Kerntakt unabhängige Taktquelle *gtx_clk* verwendet. Die von der MAC kommenden GMII Singale werden in die für RGMII benötigten Signale gewandelt und vor der Umwandlung in DDR Signale in einem Register gehalten. Danach werden die zur fallenden Taktflanke auszugebenden RGMII Signale durch ein mit dem negierten *gtx_clk* Taktsignal betriebenen Register um einen halben Takt verzögert. Durch die für die Ausgabe von DDR Signalen zu verwendenden FDDRRSE Komponenten zur Ausgabe von DDR Signalen werden die RGMII Signale über die IO-Pads aus dem FPGA herausgeleitet. Für das Taktsignal *txc* wird ein um 90° phasenverschobenes Taktsignal verwendet und ebenfalls über FDDRRSE ausgegeben. Somit ist *txc* um 2 ns verzögert und die Setup- und Hold-Zeiten des PHYs werden eingehalten. Dementsprechend werden für die RGMII Interfaces 5 DCM sowie 6 BUFGMUX mit angeschlossenen Verteilernetzen verwendet. Für den Betrieb

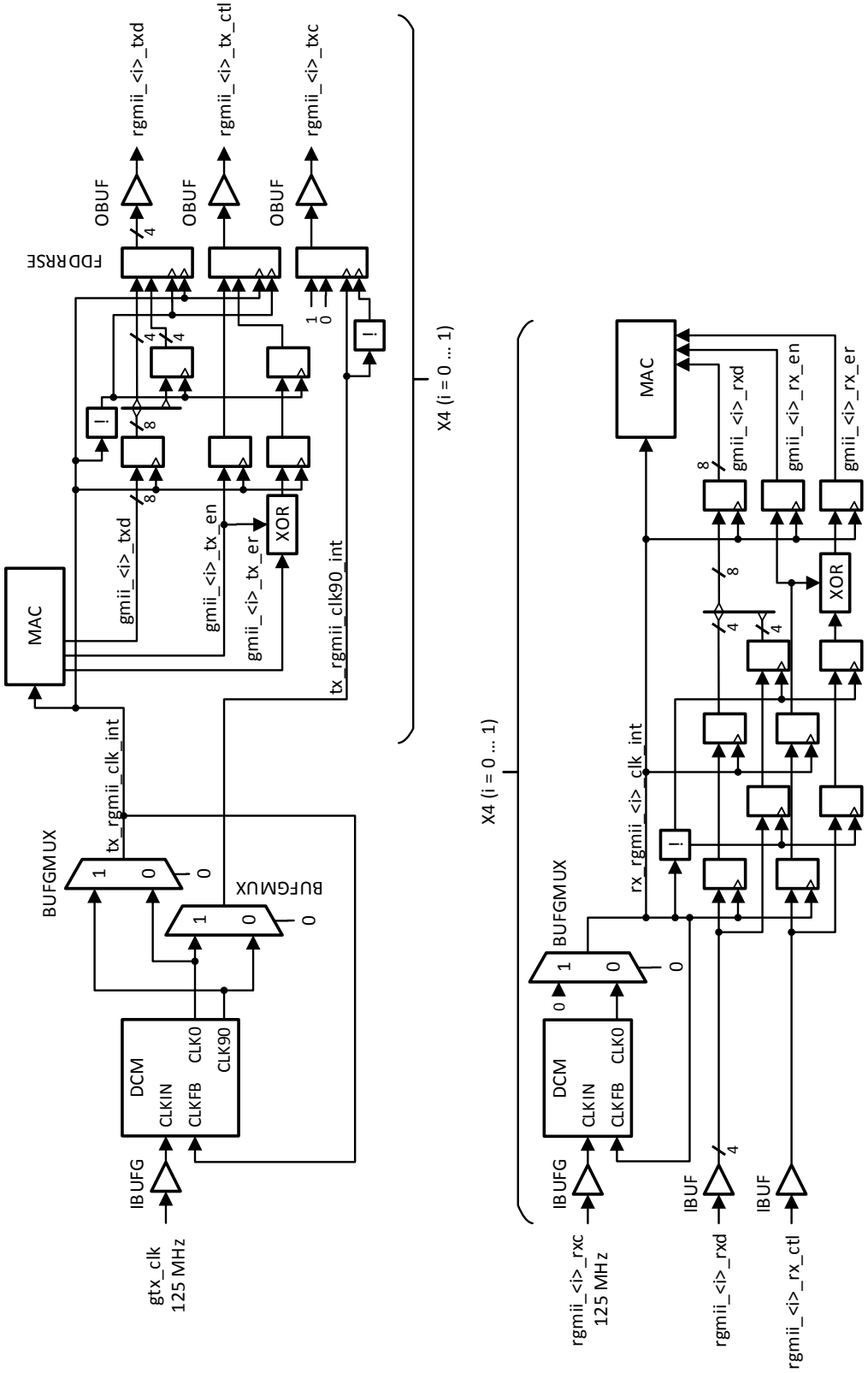


Abbildung 4.6: Ursprüngliche Komponenten zur Taktgenerierung und Anbindung des RGMII

Signal	Quadrant	GCLK	Signal	Qdr.
gtx_clk	NE	GKLK2S	rgmii_0_txc	SW
rgmii_0_rxc	NW	GKLK4S	rgmii_0_tx_ctl	SW
rgmii_0_rx_ctl	SW		rgmii_0_txd[0]	SW
rgmii_0_rxd[0]	SW		rgmii_0_txd[1]	SW
rgmii_0_rxd[1]	SW		rgmii_0_txd[2]	SW
rgmii_0_rxd[2]	SW		rgmii_0_txd[3]	SW
rgmii_0_rxd[3]	SW		rgmii_1_txc	SW
rgmii_1_rxc	NW	GKLK6S	rgmii_1_tx_ctl	SW
rgmii_1_rx_ctl	SW		rgmii_1_txd[0]	SW
rgmii_1_rxd[0]	SW		rgmii_1_txd[1]	SW
rgmii_1_rxd[1]	SW		rgmii_1_txd[2]	SW
rgmii_1_rxd[2]	SW		rgmii_1_txd[3]	SW
rgmii_1_rxd[3]	SW		rgmii_2_txc	SW
rgmii_2_rxc	SW	GKLK5S	rgmii_2_tx_ctl	SW
rgmii_2_rx_ctl	SW		rgmii_2_txd[0]	SW
rgmii_2_rxd[0]	SW		rgmii_2_txd[1]	SW
rgmii_2_rxd[1]	SW		rgmii_2_txd[2]	SW
rgmii_2_rxd[2]	SW		rgmii_2_txd[3]	SW
rgmii_2_rxd[3]	SW		rgmii_3_txc	SW
rgmii_3_rxc	SW	GKLK7S	rgmii_3_tx_ctl	SW
rgmii_3_rx_ctl	SE		rgmii_3_txd[0]	SW
rgmii_3_rxd[0]	SW		rgmii_3_txd[1]	SW
rgmii_3_rxd[1]	SW		rgmii_3_txd[2]	SW
rgmii_3_rxd[2]	SW		rgmii_3_txd[3]	SW
rgmii_3_rxd[3]	SW			

Tabelle 4.1: Quadranten der RGMII IO-Signale

des NetFPGA wird weiterhin ein DCM und ein BUFGMUX mit angeschlossenem Verteilernetz für den Kerntakt sowie ein BUFGMUX mit Verteilernetz für das für die PCI-Schnittstelle verwendete Taktsignal instantiiert. Somit werden insgesamt 6 DCM sowie 8 BUFGMUX mit angeschlossenem Verteilernetz verwendet. Die ursprüngliche Umsetzung erlaubt ausschließlich eine Übertragungsrate von $1000\ \mathrm{Mbit/s}$.

Der verwendete PHY unterstützt mit Autonegotiation das dynamische Aushandeln der auf dem Link verwendeten Übertragungsrate. Üblicherweise wird diese Funktionalität aktiviert, die angeschlossene MAC muss die verwendeten Taktsignale und die Signalisierung entsprechend der ausgehandelten Übertragungsrate umschalten. Für die RGMII Signale von PHY zum MAC würde dies lediglich zu geringen Änderungen führen, weitere Ressourcen zur Taktverarbeitung müssten nicht in Anspruch genommen werden. Allerdings müssten für die RGMII Signale vom MAC zum PHY entweder die Taktraten $25\ \mathrm{MHz}$ und $2,5\ \mathrm{MHz}$ generiert werden oder die Logik der MAC mit entsprechend gesteuerten Enables erweitert werden und somit zum Betrieb der MAC als bezüglich des RGMII übertaktete Komponente umgestaltet werden. Die Einführung von Enables in der MAC kommt auf Grund des umfangreichen Aufwandes bei der Anpassung der MAC nicht in Frage. Die weiteren Taktraten $25\ \mathrm{MHz}$ und $2,5\ \mathrm{MHz}$ könnten zwar mit Hilfe eines DCM generiert werden und für jede MAC individuell über BUFGMUX ausgewählt werden. Allerdings würden so im Quadranten SW allein zur Versorgung der Logik der MACs mit den RGMII Takten acht Taktverteilbäume benötigt werden. Somit könnte der Quadrant SW ausschließlich für den Teil der Logik der MACs verwendet werden, der mit den Takten des RGMII betrieben wird. Der im Kerntakt des NetFPGAs betriebene Teil der MACs müsste in andere Quadranten verlagert werden. Da somit die Ressourcen des Quadrantens SW nur zu einem sehr geringen Teil ausgelastet wären, die RT-Bridge aber mehr als $90\ \%$ der Ressourcen des FPGAs beansprucht, kann ein dynamisches Umstellen der Übertragungsrate nicht realisiert werden.

Dementsprechend wird die ursprüngliche Umsetzung so angepasst, dass auf den PHYs die Autonegotiation deaktiviert und eine Übertragungsrate von $100\ \mathrm{Mbit/s}$ fest eingestellt wird. Die zur Taktgenerierung und Anbindung des RGMII verwendeten Komponenten werden, wie in Abbildung 4.7 dargestellt, angepasst. Um möglichst große Teile der MAC ohne Anpassungen zu verwenden, werden die MACs nicht im Takt des RGMII betrieben. Durch eine Halbierung des RGMII Taktes auf $12,5\ \mathrm{MHz}$ werden die Paketdaten aus Sicht der MAC mit DDR signalisiert. Dementsprechend muss das Interface und die Umsetzung der MAC nicht angepasst werden. Der Betriebstakt für die ausgehende Seite der MACs *tx_rgmii_clkdv* wird von *gtx_clk* abgeleitet. Die RGMII Signale *txd* und *tx_ctl* vom MAC zum PHY werden mit dem

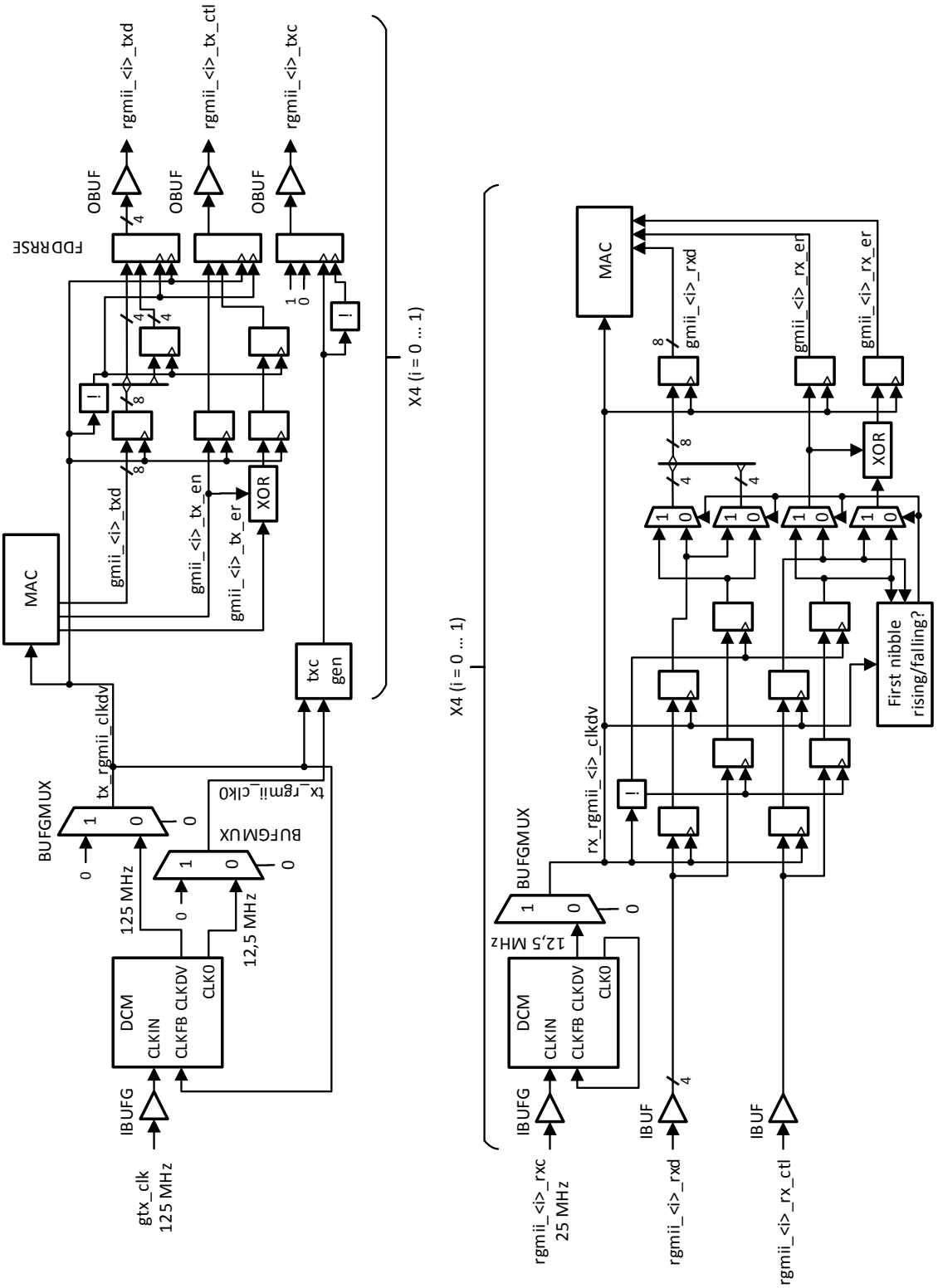


Abbildung 4.7: Angepasste Komponenten zur Taktgenerierung und Anbindung des RGMII

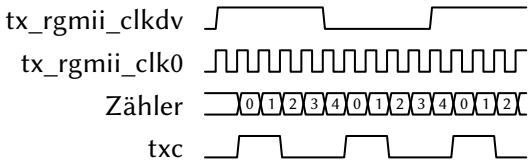


Abbildung 4.8: Verhalten des Zählers zum Generieren von *txc*

Takt *tx_rgmii_clkdv* verarbeitet und durch die bereits vorhandene Schaltung zur Ausgabe mit DDR geleitet.

Das Signal *txc* wird wie folgt erzeugt: die Ausgänge *CLKDV* und *CLKFX* der DCM werden unabhängig voneinander vom an *CLKIN* anliegenden Takt abgeleitet. Somit treten die Taktflanken von *CLKDV* und *CLKFX* zwar jeweils zu Taktflanken von *CLKIN* auf, es besteht allerdings kein weiterer Zusammenhang zwischen *CLKDV* und *CLKFX*. Dies trifft auch auf Konfigurationen der DCM zu, bei denen es sich bei *CLKFX* um ein Vielfaches von *CLKDV* handelt. Da somit die Phasen von *CLKDV* und *CLKFX* unabhängig sind, kann der für das Signal *txc* benötigte Takt nicht direkt durch den bereits verwendeten DCM erzeugt werden. Weiterhin kann die benötigte Taktquelle nicht durch einen weiteren DCM von *tx_rgmii_clkdv* abgeleitet werden, da die verfügbaren DCM anderweitig benötigt werden. So werden für die Generierung der MAC Takte fünf DCM verwendet, für den NetFPGA Kerntakt eine DCM sowie für die für das eingebettete Prozessorsystem benötigten Takte zwei DCM.

Das Signal *txc* muss außerdem die Setup- und Hold-Zeiten des RGMII Interfaces einhalten. Die steigenden Taktflanken müssen somit mindestens $2\ \mathrm{ns}$ nach den Taktflanken von *tx_rgmii_clkdv* auftreten. Dementsprechend wird *txc* durch einen Zähler mit Komparatoren generiert. Der Zähler wird mit $125\ \mathrm{MHz}$ betrieben. Wird eine Taktflanke von *tx_rgmii_clkdv* detektiert, wird der Zähler im darauf folgenden Takt auf 0 gesetzt und an *txc* ein high Pegel angelegt. Beträgt der Wert des Zählers 2, wird an *txc* ein low Pegel angelegt. Daraus ergibt sich das in Abbildung 4.8 dokumentierte Verhalten und ein Duty Cycle von $40\ \%$.

Der Betriebstakt für die eingehende Seite der MACs *rx_gmii_<i>_clkdv* wird direkt vom RGMII Signal *rxc* abgeleitet. Hierzu wird *rxc* durch einen DCM halbiert. Die RGMII Signale *rxd* und *rx_ctl* werden durch die bereits vorhandenen Schaltungen in den FPGA geleitet. Da Pakete zu steigenden Flanken von *rxc* beginnen und somit auch zu fallenden Flanken von *rx_gmii_<i>_clkdv* beginnen können, musste die Verarbeitung der RGMII Signale, wie in Abbildung 4.7 dargestellt, angepasst werden. Dabei wird erkannt, ob ein Paket zu einer fallenden Flanke von *rx_gmii_<i>_clkdv* begonnen hat und die RGMII Signale werden gegebenenfalls anders interpretiert.

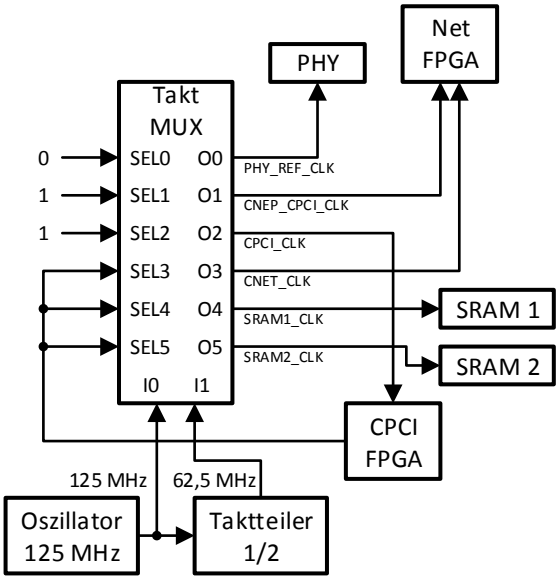


Abbildung 4.9: Taktquellen auf dem NetFPGA Board

4.1.4 Anpassung der Kerntaktgenerierung

Auf Grund der hohen Auslastung der FPGA Ressourcen kann durch die verwendete Toolchain kein Routing der Signale mehr erreicht werden, das einen Takt von 125 MHz erlaubt. Weiterhin wird die Bandbreite der Ports, wie im vorherigen Abschnitt erläutert, reduziert. Da die zu verarbeitende maximale Datenrate ein Zehntel beträgt, kann bei einer Reduzierung des Kerntaktes weiterhin die volle Bandbreite aller Ports verarbeitet werden. Dementsprechend wird der Kerntakt des NetFPGA auf $62,5$ MHz halbiert. Im Folgenden werden erst die an der Generierung des Kerntaktes beteiligten Komponenten erläutert. Danach werden die sich daraus ergebenden Änderungen beschrieben.

An der Generierung der auf dem NetFPGA verwendeten Taktsignalen sind die in Abbildung 4.9 dargestellten Komponenten beteiligt. Wie ersichtlich wird ein 125 MHz Taktsignal durch einen Oszillator generiert. Das generierte Taktsignal wird durch einen Taktteiler auf $62,5$ MHz halbiert. Durch einen Multiplexer kann für sechs Ausgänge zwischen den beiden verfügbaren Taktraten selektiert werden. Die Taktquellen für den zur Anbindung an die PCI Schnittstelle verwendeten CPCI FPGA, die Schnittstelle zum CPCI FPGA auf dem NetFPGA sowie den PHYs sind anhand der Verdrahtung festgelegt. Der im NetFPGA verwendete Kerntakt sowie der Betriebstakt des SRAM kann über ein Register im CPCI FPGA über die PCI-Schnittstelle konfiguriert werden.

Dementsprechend muss der Kerntakt des NetFPGA vor dem Laden der Konfiguration über die PCI-Schnittstelle umgeschaltet werden. Der Kerntakt kann nicht mit Hilfe eines DCM halbiert werden, da die Schnittstelle zum SRAM taktsynchron betrieben wird und die Ansteuerung des SRAM somit nicht mehr korrekt erfolgen würde. Dementsprechend wurden die Änderung des Kerntaktes in einem angepassten User Constraint File (UCF) eingetragen.

4.2 Paketempfang und -versand

Nachdem im vorherigen Abschnitt auf die Erzeugung der benötigten Taktsignale sowie die Anpassung der RGMII Schnittstellen eingegangen wurde, setzt sich dieser Abschnitt mit der Umsetzung der Module Pre und Post Processing auseinander. In Abschnitt 4.2.1 wird auf die verwendeten Interfaces eingegangen und in Abschnitt 4.2.2 die Umsetzung des Pre Processing Moduls beschrieben. Auf die Umsetzung des Post Processing Moduls wird in Abschnitt 4.2.3 eingegangen.

4.2.1 Interfaces

Für die Umsetzung der Module zum Paketempfang und -versand müssen die Pakete von der bestehenden NetFPGA Infrastruktur entgegengenommen und an diese übergeben werden. Dementsprechend wird hier das vom NetFPGA Projekt verwendete Interface zum Übermitteln der Pakete beschrieben. Weiterhin wird das Interface zum Übermitteln der internen Paketheader in sowie das Interface zum Übermitteln von Events zum Paketversand an das eingebettete Prozessorsystem beschrieben. Zudem wird in den hier beschriebenen Modulen das in Abschnitt 4.5 erläuterte Interface zum Buffer Management verwendet.

Das in der bestehenden NetFPGA Infrastruktur verwendete Interface zur Übermittlung von Paketen unterstützt eine beidseitige Flusskontrolle. Das Interface verfügt über eine konfigurierbare Breite des Datenpfades, im NetFPGA wird eine Breite von 32 bit oder 64 bit verwendet. Eine Verbindung zwischen einer Paketquelle und einer Paketsenke ist in Abbildung 4.10 mit 64 bit Breite des Datenpfades dargestellt, eine beispielhafte Paketübermittlung ist in Abbildung 4.11 dargestellt. Das Signal *rdy* zeigt an, ob der Empfänger zum nächsten Takt ein Wort des Paketes aufnehmen kann. Durch das Signal *rdy* kann die Paketsenke so den Paketfluss steuern. Das Signal *wr* zeigt an, ob die Signale *ctrl* und *data* Paketdaten enthalten. Die Paketquelle kann entsprechend durch das Signal *wr* den Paketfluss steuern. Mit dem Signal *data* werden die Daten der Pakete übermittelt.

Mit dem Signal *ctrl* wird signalisiert, ob es sich um ein Headerwort mit zusätzlichen Informationen, Paketdaten oder das letzte Wort eines Paketes handelt. Es verfügt für jedes Byte

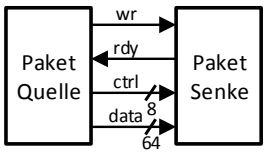


Abbildung 4.10: Signale des im NetFPGA verwendeten Interfaces zur Übermittlung von Paketen

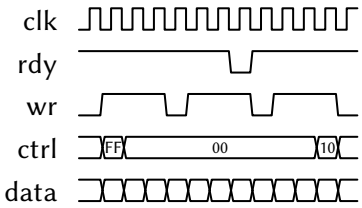


Abbildung 4.11: Beispielhafte Paketübermittlung mit dem im NetFPGA verwendeten Interface zur Paketübermittlung

des *data* Signals über ein Bit. Sind im ersten Wort eines Paketes Bits des *ctrl* Signals gesetzt, handelt es sich um ein Headerwort. Im Headerwort ist beispielsweise die Länge des folgenden Paketes abgelegt. Sind alle Headerworte übermittelt, beginnen die Paketdaten. Während der Übermittlung der Paketdaten sind keine Bits des *ctrl* Signals gesetzt. Lediglich im letzten Wort der Paketdaten wird durch das Setzen eines Bits im *ctrl* Signal das Ende des Paketes signalisiert. Dabei wird gleichzeitig signalisiert, wie viele Byte des letzten Worts valide Paketdaten enthalten. Jedes Bit des *ctrl* Signals wird dabei einem Byte des Datenwortes zugeordnet, für das letzte valide Byte des Wortes wird das entsprechende Bit im *ctrl* Signal gesetzt.

Im in Abbildung 4.11 dargestellten Beispiel ist die Paketsenke anfangs empfangsbereit. Dann übermittelt die Paketquelle mit dem Headerwort der eingehenden MAC den Beginn des Paketes. Danach werden die Paketdaten begonnen und die Paketquelle setzt für einen Takt die Übertragung aus. Nach Wiederaufnahme der Paketübermittlung wird die Übermittlung der Paketdaten durch die Paketsenke für einen Takt unterbrochen, das letzte Wort des Paketes beinhaltet drei Byte. Somit besteht das übermittelte Paket aus einem Headerwort und 8 Worten mit Paketdaten. Das Ethernetpaket hat somit eine Länge von 59 B, davon 56 B in 7 vollständig genutzten Wörtern und 3 B im letzten Wort.

Da innerhalb der RT-Bridge Pakete ohne Unterbrechungen weitergeleitet werden sollen, verfügt das Interface zum Vermitteln der intern verwendeten Paketheader über keinerlei Flusskontrolle. Dementsprechend sind die dieses Interface verwendenden Komponenten entweder in der Lage, die maximal durch dieses Interface zu übermittelnde Paketrate zu verarbeiten, oder es muss entsprechend gezeigt werden, dass sie die tatsächlich auftretende Paketrate ver-

Wort Idx	Bits	Feld	Beschreibung
0, 1	31–0, 31–16	src	Quelladresse des Paketes
1, 2	15–0, 31–0	dst	Zieladresse des Paketes
3	31–20	vid	VLAN ID des Paketes
3	19–17	in_port	Index des Eingangsports
3	16–1	packet_id	Für Port eindeutiger Paketidentifier
3	0	fdb_overwr	FDB wird ignoriert, wenn gesetzt
4	31–27	out_port	Ports, an die Paket vermittelt wird
4	26–25	ctrl	In CTRL-Exec auszuführende Operationen
4	24–14	len	abgelegte Daten im Paketpuffer in Byte
4	13–3	buff_id	ID des Paketpuffers
5	31–29	queue_id	ID der Paketklasse
6, 7	31–0, 31–0	in_tstmp	Eingehender Zeitstempel
8, 9	31–0, 31–0	out_tstmp	Ausgehender Zeitstempel

Tabelle 4.2: Felder des innerhalb der RT-Bridge verwendeten Paketheaders

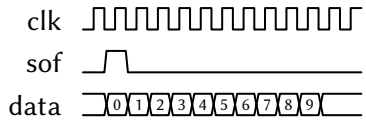


Abbildung 4.12: Beispielhafte Paketübermittlung mit dem in der RT-Bridge verwendeten Interface zur Paketübermittlung (*data* Signal enthält Index des Paketworts)

arbeiten können. Daraus resultiert einerseits, dass für die betroffenen Komponenten entsprechende Analysen durchgeführt werden müssen. Da andererseits ein deterministisches Verhalten der RT-Bridge in Kapitel 2.5 gefordert wird, sind diese ohnehin durchzuführen. Weiterhin führt ein Verzicht auf die Flusskontrolle zu einer geringeren Komplexität bei der Verwendung dieses Interfaces. Es verfügt über die Signale *sof* und *data*. Beide Signale werden durch die Paketquelle getrieben. Das Signal *sof* markiert das erste, auf dem 32 bit breiten Signal *data* übermittelte Wort. Ein Paket wird immer durch 10 Worte repräsentiert, in den einzelnen Worten sind die in Tabelle 4.2 gelisteten Felder enthalten. Daraus ergibt sich die in Abbildung 4.12 dargestellte beispielhafte Übermittlung eines Paketes. Für den Zugriff auf die Felder der Paketheader wurden die Komponenten Header Update und Header Extract umgesetzt.

Das Interface zum Übermitteln der Paketversandevents an das eingebettete Prozessorsystem besteht aus dem Kontrollsignal *wr* sowie der 32 bit breiten Datenleitung *data*. Sie werden beide durch die Quelle der Events getrieben. Ein Event besteht aus fünf, über das Signal *data* übermittelte Wörter. Während der Übermittlung der Wörter ist das Signal *wr* gesetzt, die Wörter werden ohne Unterbrechung übermittelt. Die Wörter enthalten die in Tabelle 4.3 gelisteten

Word Idx	Bits	Feld
0	31–29	in_port
0	28–13	packet_id
1, 2	31–0, 31–0	in_tstmp
3, 4	31–0, 31–0	out_tstmp

Tabelle 4.3: Felder der Paketversandevents (Beschreibungen siehe Tabelle 4.2)

Felder. Da keine Flusskontrolle umgesetzt wurde, muss das eingebettete Prozessorsystem vor der Verarbeitung durch die Software mehrere Events puffern.

4.2.2 Pre Processing

Im Modul Pre Processing werden, wie im Kapitel 3 beschrieben, die eingehenden Pakete analysiert und die benötigten Informationen extrahiert. Weiterhin werden die Paketdaten im Pufferspeicher abgelegt und die Eingangszeitstempel erstellt. Im Folgenden werden die einzelnen Komponenten beschrieben.

Die Time Stamper Komponente erstellt die Eingangszeitstempel anhand der global synchronisierten Zeitbasis. Die global synchronisierte Zeitbasis liegt als $64\ \mathrm{bit}$ breites Signal vor und wird durch ein $32\ \mathrm{bit}$ Korrekturregister um einen fixen Wert korrigiert. Das Korrekturregister wird als vorzeichenbehafteter Wert betrachtet und kann über die Register Bridge durch das eingebettete Prozessorsystem konfiguriert werden. Der Zeitstempel wird zur steigenden Flanke eines Triggers in einem Register gehalten und bei der Ankunft des nächsten Paketheaders in das entsprechende Feld geschrieben. Weiterhin wird die Time Stamper Komponente auch zum Erstellen des Ausgangszeitstempels verwendet. Über entsprechende Parameter kann zum Zeitpunkt der Synthese konfiguriert werden, ob Eingangs- oder Ausgangszeitstempel erstellt werden sollen.

Mit Hilfe der Komponente Packet Analyzer werden die benötigten Felder aus dem Paket extrahiert. Sie besteht aus einem Zähler, der die Worte des Paketes. Bei den entsprechenden Zählerständen im Paket werden die dazugehörigen Felder aus dem Ethernet Header extrahiert. Weiterhin wird ermittelt, ob das Paket über einen VLAN Tag verfügt und gegebenenfalls die VLAN ID extrahiert. Verfügt das Paket über keinen VLAN Tag, wird die in einem Register konfigurierte VLAN ID für Pakete ohne VLAN Tag verwendet. Die extrahierten Felder werden in den internen Paketheader geschrieben. Die für Pakete ohne VLAN Tag zu verwendende VLAN ID wird durch das eingebettete Prozessorsystem über die Register Bridge konfiguriert.

Die Komponente Buffer Sink sorgt für das Ablegen der Paketdaten im Paketpuffer. Hierfür verfügt sie über einen Mealy-Zustandsautomaten mit den folgenden Zuständen:

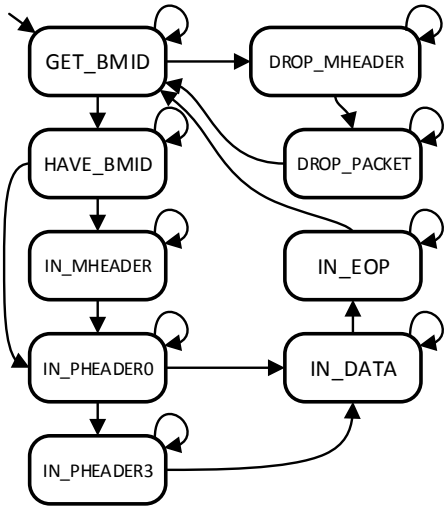


Abbildung 4.13: Mealy-Zustandsautomat der Buffer Sink

- **GET_BMID:** Dieser Zustand wird nach dem Reset aktiviert und fordert eine Buffer ID vom Puffermanagement an.
- **HAVE_BMID:** Buffer ID erhalten, warte auf den Beginn des nächsten Pakets.
- **IN_MHEADER:** Wurde der Beginn eines Paketes erkannt, wird hier aus dem NetFPGA Paketheader die Länge des Paketes extrahiert, Verlassen löst den Trigger für den Eingangszeitstempel aus.
- **IN_PHEADER0, IN_PHEADER3:** Diese Zustände überspringen den Paketheader, gegebenenfalls mit VLAN Tag.
- **IN_PDATA:** Dieser Zustand legt die Paketdaten inklusive des Type Felds im Paketpuffer ab.
- **IN_EOP:** Beim Verlassen dieses Zustands wird der Beginn des internen Paketheaders ausgelöst, warten bis alle Paketdaten gespeichert sind.
- **DROP_MHEADER, DROP_PACKET:** Hat die Buffer Sink keine Buffer ID erhalten, erhält aber ein Paket, wird in diesen Zuständen das Paket verworfen.

Nicht aufgeführt sind hier Zustände, die für die konfigurierbare Unterstützung eines 32 bit breiten Paketpfades verwendet werden. Der Zustandsautomat verfügt über die in Abbildung 4.13 dargestellten Zustandsübergänge. Die Paketdaten werden in einer FIFO gepuffert, bevor sie im Paketpuffer abgelegt werden.

4.2.3 Post Processing

Im Modul Post Processing werden, wie in Kapitel 3 beschrieben, durch die Komponenten Buffer Source die zu sendenden Paketdaten aus dem Pufferspeicher gelesen und die benötigten Paketheader generiert. Durch die Komponente VLAN Tagger wird gegebenenfalls ein VLAN Tag hinzugefügt und die Ausgangszeitstempel werden durch den Time Stamper generiert. Bei dem Time Stamper handelt es sich um den in Abschnitt 4.2.2 beschriebenen Time Stamper mit entsprechender Parametrisierung. Dementsprechend wird er hier nicht weiter erläutert. Gegebenenfalls wird durch die Komponente CTLR Exec das *transparentClock* Feld der PC-Frames aktualisiert und Events über den Paketversand an das eingebettete Prozessorsystem übermittelt. Im Folgenden wird die Umsetzung der einzelnen Komponenten beschrieben.

Um die Paketdaten aus dem Pufferspeicher zu lesen und den Ethernet Header zu generieren, werden durch die Buffer Source die Felder *buff_id*, *len*, *src* und *dst* aus dem internen Paketheader extrahiert. Danach werden die Paketdaten aus dem Pufferspeicher ausgelesen und bis zur Weiterleitung in einer FIFO gepuffert. Kern der Buffer Sink ist ein Mealy-Zustandsautomat mit folgenden Zuständen:

- **IDLE:** In diesem Zustand wird auf das nächste Paket gewartet.
- **EXTRACTING:** In diesem Zustand sind die ersten Wörter eines internen Paketheaders eingetroffen, die benötigten Felder werden extrahiert; Verlassen löst Trigger für Time Stamper aus.
- **ETH_HDR2:** Der Ethernet Header wird ausgegeben.
- **DATA:** In diesem Zustand werden die Paketdaten ausgegeben.
- **FREEID:** Alle Paketdaten wurden ausgegeben, die Buffer ID wird freigegeben.

Nicht aufgeführt sind hier Zustände, die für die konfigurierbare Unterstützung eines 32 bit breiten Paketpfades verwendet werden. Der Zustandsautomat verfügt über die in Abbildung 4.14 dargestellten Zustandsübergänge. Durch die Buffer Source wird der Trigger für den Timestamper beim Verlassen des Zustandes EXTRACTING betätigt.

Die Komponente VLAN Tagger extrahiert das Feld *vid* aus dem internen Paketheader und vergleicht die VLAN ID des Paketes mit der für diesen Port für Pakete ohne VLAN Tag zu verwendenden VLAN ID. Die an diesem Port für Pakete ohne VLAN Tag zu verwendende VLAN ID wird über die Register Bridge durch das eingebettete Prozessorsystem in einem Register konfiguriert. Unterscheidet sich das Feld *vid* von der konfigurierten VLAN ID, wird ein VLAN Tag angefügt. Wird ein 32 bit langer VLAN Tag in einen 64 bit breiten Paketdatenstrom eingefügt, müssen die Paketdaten um ein halbes Wort verzögert werden. Somit ergibt sich das als nächstes auszugebende Wort des Paketstroms aus der zweiten Hälfte des zuletzt eingegan-

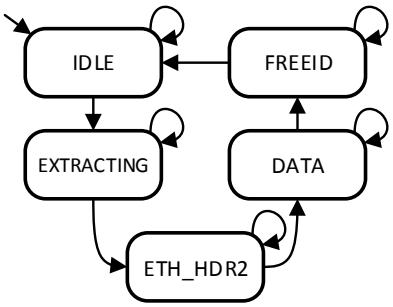


Abbildung 4.14: Mealy-Zustandsautomat der Buffer Source

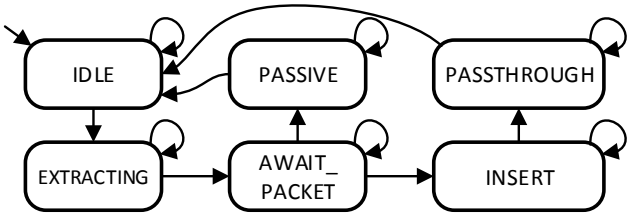


Abbildung 4.15: Mealy-Zustandsautomat des VLAN Taggers

genen Wortes und der ersten Hälfte des aktuell eingehenden Wortes. Dies wurde im VLAN Tagger entsprechend umgesetzt. Ein Mealy-Zustandsautomat mit den folgenden Zuständen steuert das Einfügen des VLAN Tags:

- **IDLE:** Warten auf die Ankunft eines internen Paketheaders.
- **EXTRACTING:** Warten auf die Extraktion des Feldes *vid*.
- **AWAIT_PACKET:** Es wird auf den Beginn des Paketes gewartet.
- **PASSIVE:** Wenn kein VLAN Tag eingefügt wird, wird hier auf das Paketende gewartet.
- **INSERT:** Wenn ein VLAN Tag eingefügt wird, wird er in diesem Zustand eingefügt.
- **PASSTHROUGH:** Wenn ein VLAN Tag eingefügt wird, werden in diesem Zustand die übrigen Paketdaten weitergeleitet.

Nicht aufgeführt sind hier Zustände, die für die konfigurierbare Unterstützung eines 32 bit breiten Paketpfades verwendet werden. Der Zustandsautomat mit den beschriebenen Zuständen verfügt über die in Abbildung 4.15 abgebildeten Zustandsübergänge.

In der Komponente CTRL Exec wird abhängig vom Feld *ctrl* des internen Paketheaders das *transparentClock* Feld der PC-Frames aktualisiert und ein Paketversandevent an das eingebettete Prozessorsystem übergeben. Dementsprechend werden aus dem internen Paketheader die benötigten Felder *ctrl*, *in_port*, *packet_id*, *in_tstmp* sowie *out_tstmp* extrahiert. Für jede der auszuführenden Aktionen ist im *ctrl* Feld ein Bit vorhanden. Ist Bit 0 gesetzt, wird das

Wort Idx	Position der Felder ohne VLAN Tag							
	Byte 1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**
0	Zieladresse						Quelladresse	
1	Quelladresse				Ethernet Type		Integration Cycle	
2	Integration Cycle		Membership New				reserviert	
3	reserviert		Sync Prio	Sync Dom.	Type	reserviert		
4	reserviert		Transparent Clock					
5	Transparent Clock		ungenutzt					
	Position der Felder mit VLAN Tag							
0	Zieladresse						Quelladresse	
1	Quelladresse				0x8100		VLAN Tag	
2	Ethernet Type		Integration Cycle				Membership New	
3	Membership New		reserviert				Sync Prio	Sync Dom.
4	Type	reserviert					Transparent Clock	
5	Transparent Clock						ungenutzt	

Tabelle 4.4: Felder der PC-Frames in Ethernet Paketen mit und ohne VLAN Tag [61]

transparentClock Feld aktualisiert, ist Bit 1 gesetzt, wird für das übermittelte Paket ein Paketversandevent an das eingebettete Prozessorsystem übergeben. Für die Generierung des Paketversandevents werden die extrahierten Felder über das in Abschnitt 4.2.1 definierte Interface ausgegeben.

Um eine Synchronisation mit hoher Präzision zu ermöglichen, ist laut [61] die Latenz beim Weiterleiten der PC-Frames zu messen und das *transparentClock* Feld der PC-Frames wie folgt zu aktualisieren:

$$\begin{aligned} pcf\_transparent\_clock_{new} = {} & pcf\_transparent\_clock_{old} \\ & + dynamic\_relay\_delay \\ & + static\_relay\_delay \\ & + wire\_delay \end{aligned}$$

Der bisherige Wert des *transparentClock* Felds wird aus dem Paket extrahiert. Der $dynamic\_relay\_delay$ ergibt sich aus der Differenz zwischen eingehendem und ausgehendem Zeitstempel und $static\_relay\_delay$ wird zusammen mit $wire\_delay$ über ein Register konfiguriert. Da, wie sich aus Tabelle 4.4 ergibt, das *transparentClock* Feld abhängig von der Existenz eines VLAN Tags über unterschiedliche Teile von zwei Wörtern der Paketdaten verteilt, ist müssen mehrere Wörter des Paketes gepuffert werden. Weiterhin benötigt die

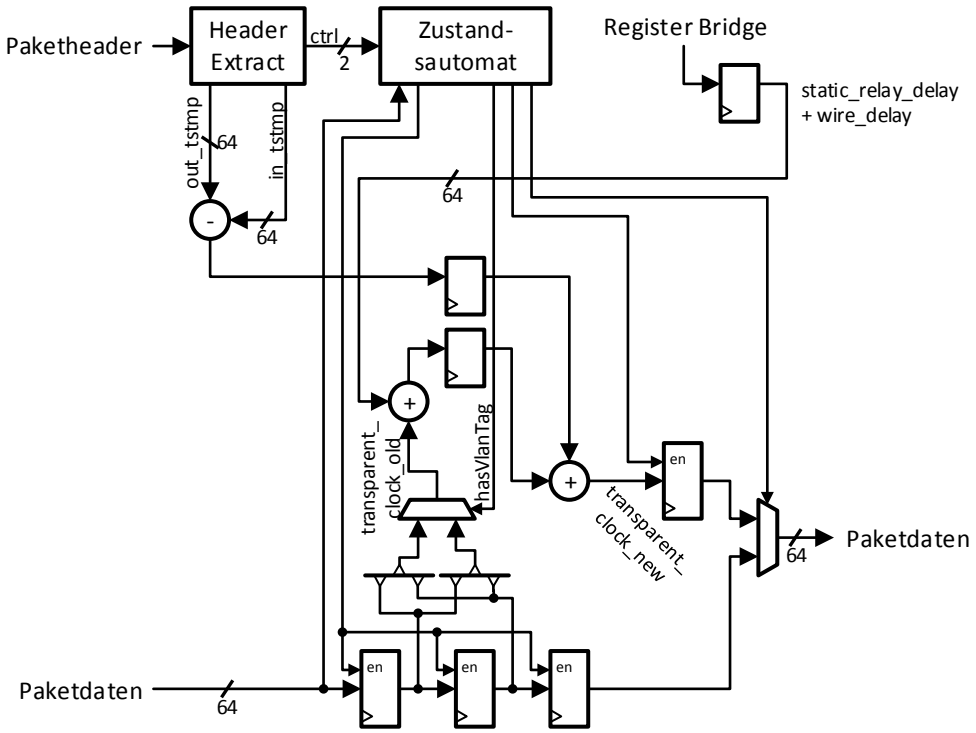


Abbildung 4.16: Berechnung des aktualisierten *transparentClock* Felds in CTRL Exec

Logik zum Berechnen des neuen *transparentClock* Felds eine Laufzeit von mehr als einem Takt und das neu berechnete Feld muss in das Paket zurückgeschrieben werden. Dementsprechend werden die Wörter des Paketes in mehreren Registern gehalten, es ergibt sich die in 4.16 dargestellte Schaltung. Zur Berechnung des neuen *transparentClock* Felds wird, wie dargestellt, ein Balanced-Adder-Tree verwendet. Die Steuerung der Berechnung wird durch einen Mealy-Zustandsautomaten durchgeführt. Dem Zustandsautomaten steht der die Paketworte zählende Zähler *t_cnt* zur Verfügung. Anhand des Zählers werden die korrekten Worte des PC-Frames abgepasst. Er verfügt über die folgenden Zustände:

- **IDLE:** Warten auf den Beginn des nächsten internen Paketheaders.
- **AWAIT_PACKET:** Warten auf den Beginn der Paketdaten.
- **PROCESS_PACKET:** Warten auf das Ende der Paketdaten, abhängig von *p_cnt* wird die Berechnung des *transparentClock* Felds duchgeführt.
- **PACKET_END1, PACKET_END2, PACKET_END3:** In diesen Zuständen werden die in den Registern verbleibenden Paketdaten weitergegeben.

Der Zustandsautomat verfügt über die in Abbildung 4.17 dargestellten Zustandsübergänge.

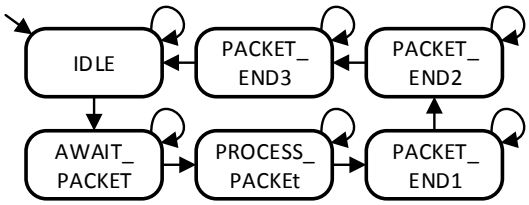


Abbildung 4.17: In CTRL Exec verwendeter Zustandsautomat

4.3 Paketforwarding

Im vorherigen Abschnitt wurde die Umsetzung der Module zum Empfangen und Versenden der Pakete beschrieben. Folgend wird die Umsetzung der Komponente Input Arbiter zur Serialisierung der eingehenden Pakete von mehreren Ports sowie der in Kapitel 3.3 angesprochenen FDB erläutert. Weiterhin wird die Umsetzung der Komponente Output Port Multiplexer zur Weiterleitung der Pakete an die, durch die FDB ermittelten, Ports beschrieben. Zur Umsetzung der FDB in Hardware wurde das in Abschnitt4.3.1 beschriebene Konzept entwickelt. Es erlaubt die Konfiguration aller möglichen Regeln über das eingebettete Prozessorsystem. Da für das entwickelte Konzept allerdings nicht ausreichend Ressourcen auf dem FPGA zur Verfügung stehen, wurde lediglich die in Abschnitt 4.3.2 beschriebene Umsetzung durchgeführt. Sollte sich durch folgende Arbeiten eine Reduzierung der auf dem FPGA genutzten Ressourcen ergeben, kann das entwickelte Konzept vollständig umgesetzt werden. Der Input Arbiter wird in Abschnitt 4.3.3 beschrieben und der Output Port Multiplexer in Abschnitt 4.3.5.

4.3.1 Konzept zur Umsetzung der FDB in Hardware

Wie im IEEE 802.1Q Standard [14] und in Kapitel 3.3 erläutert, werden die Pakete in einer Bridge anhand von Regeln an die Ausgangsports weitergeleitet. Jede Regel beinhaltet eine Beschreibung, auf welche Pakete sie angewandt werden soll, sowie eine *PortMap* mit den für einzelne Ports durchzuführenden Aktionen. Die Regeln lassen sich in statische und dynamische Regeln unterteilen. Statische Regeln sind Resultat manueller Konfigurationen und dynamische Regeln resultieren aus der Operation von Protokollen wie dem Multiple MAC Registration Protocol (MMRP) und dem Multiple VLAN Registration Protocol (MVRP). Statische Regeln sind generell dynamischen Regeln vorzuziehen. Entspricht ein Paket der Beschreibung einer statischen Regel, dürfen dynamische Regeln nur angewandt werden, wenn dies in der statischen Regel explizit erlaubt ist. Dementsprechend wird in der *PortMap* bei allen Regeln für jeden Port angegeben ob Pakete gefiltert (*filter*) oder weitergeleitet (*forward*) werden. Bei

statischen Regeln kann weiterhin für jeden Port angegeben werden, ob Pakete anhand dynamischer Regeln verarbeitet werden sollen.

Es sind sowohl statische als auch dynamische Regeln zum Aufspannen eines VLANs vorgesehen. Zur Beschreibung der Pakete wird die VID angegeben. Die *PortMap* beschreibt, welche Ports zu einem VLAN gehören. Die Beschreibung der Pakete beinhaltet lediglich die auf sie zutreffende VID. Für statische VLAN Regeln wird weiterhin angegeben, an welchen Ports Pakete ohne VLAN Tags zum VLAN der Regel gehören.

Außerdem sind sowohl statische als auch dynamische Regeln zum Weiterleiten von Paketen mit definierten Zieladressen und VIDs vorgesehen. Eine VID wird nicht angegeben, wenn die Regel auf alle VLANs zutreffen soll. Abgesehen von konkreten Zieladressen können die Regeln auch für alle Gruppenadressen (*all group*), alle individuellen Adressen (*all individual*) sowie alle Gruppenadressen auf, die keine andere Regel zutrifft (*all unregistered group*), gelten.

Die Weiterleitung von AVB Streams wird über dynamische Regeln spezifiziert. Sie enthalten die Zieladresse und VID der zum Stream gehörenden Pakete sowie eine *PortMap*. Das automatische Lernen von über einen Port erreichbaren Netzwerkknoten resultiert im Anlegen dynamischer Regeln. Die *PortMap* kann dabei nur einen Port, an den weitergeleitet werden soll, enthalten.

Die beschriebenen Regeln lassen sich auf die folgenden Informationen reduzieren:

- Welche VLANs sind an einem Port aktiv? Welches VLAN wird an einem Port ohne VLAN Tags vermittelt?
- Was soll mit Paketen in einem VLAN geschehen, auf die Regeln mit *all group*, *all individual* und *all unregistered group* Wildcards zutreffen?
- An welche Ports soll ein Paket mit gegebener Zieladresse und VID weitergeleitet werden?
- Handelt es sich bei einer Regel um eine statische oder dynamische Regel?

Dabei ist zu beachten, dass die Komponente Packet Analyzer im Modul Pre Processing und die Komponente VLAN Tagger im Modul Post Processing bereits die Funktionalität zur Zuordnung von Paketen ohne VLAN Tag zu einem definierten VLAN umsetzen. Aus den in Kapitel 2.5 definierten Anforderungen ergibt sich weiterhin, dass die Pakete mit Hilfe der Regeln einer Paketklasse zugeordnet werden müssen. Dementsprechend sind die genannten, reduzierten Informationen um Informationen zur Zuordnung der Pakete zu einer Paketklasse zu erweitern.

Um die reduzierten Regeln in Hardware umzusetzen, stehen mehrere Optionen zur Verfügung. Einerseits könnten die Regeln in RAM abgelegt werden und für jedes zu verarbeitendes

Paket über alle eingetragenen Regeln iteriert werden. Dementsprechend würde pro Takt eine Regel verarbeitet. Um bei 10 Wörtern langen internen Paketheadern direkt aufeinander folgende Pakete zu verarbeiten, könnten also maximal 10 Regeln eingetragen werden. Zum Zeitpunkt der Konzipierung der FDB wurde von einer Datenrate von 1000 Mbit/s pro Port und einem Kerntakt von 125 MHz ausgegangen. Daraus ergibt sich bei der in Kapitel 3.2 hergeleiteten maximalen Paketrate von maximal 5952384 Pakete/s eine Verarbeitungszeit von maximal 20 Takten pro Paket. Dementsprechend könnten maximal 20 Regeln angewandt werden. Da Pakete von allen Ports (vier PHY Ports, ein Port des eingebetteten Prozessorsystems) gleichzeitig eingehen können und somit maximal fünf Pakete direkt hintereinander an der FDB eintreffen können, müssten die Paketheader gepuffert werden. Wie aus dem folgenden Abschnitt 4.3.3 hervorgeht, muss ein Paket im Input Arbiter maximal auf das Vermitteln von vier anderen Paketen warten. Daraus ergibt sich am Input Arbiter eine Verzögerung von minimal 0 Takten und maximal 40 Takten und somit ein Jitter von 40 Takten, also 320 ns. Folgen die Pakete an der FDB direkt aufeinander, müssten sie zusätzlich um 10 Takte pro Paket verzögert werden. Somit würde sich der Jitter durch diese Lösung auf 640 ns verdoppeln.

Andererseits könnten die auf ein Paket zutreffenden Regeln mit Hilfe von Content Addressable Memory (CAM) ermittelt werden. Das auf dem FPGA verfügbare CAM erlaubt für jeden Eintrag das Ablegen einer Maske und eines Wertes. Bei Abfragen wird die Adresse des ersten mit der Eingabe übereinstimmenden Eintrags unter Berücksichtigung der Maske zurückgegeben. Zur Umsetzung der Regeln würde jeweils ein CAM- und ein RAM-Baustein zu einer Lookup Table zusammengefasst werden. Der CAM-Baustein würde die Adresse der ersten antreffenden Regel ermitteln und der RAM-Baustein an der entsprechenden Speicherstelle die für die Regel geltenden Forwardinginformationen enthalten. Dabei würde sich eine Verarbeitungszeit von drei Takten ergeben, jeweils ein Takt für die Anfragen an CAM und RAM sowie die Verarbeitung des Ergebnisses der Abfrage. Dementsprechend würde die Verarbeitung eines internen Paketheaders weniger Takte als dessen Übermittlung beanspruchen. Die Paketrate wird somit lediglich durch das in Abschnitt 4.2.1 beschriebene Interface zur Übermittlung der internen Paketheader auf $1,25 * 10^7$ Pakete/s beschränkt. Da die Verarbeitungszeit der Pakete konstant wäre, würde die FDB auch keinen Jitter verursachen. Die maximale Anzahl der so anwendbaren Regeln wird durch die auf dem FPGA verfügbaren Ressourcen begrenzt und wirkt sich im Gegensatz zur im vorherigen Absatz beschriebenen Lösung nicht auf die maximale Paketrate der FDB aus.

Da die Vorteile bei der Verwendung vom CAM überwiegen, wird die FDB mit Hilfe von CAM umgesetzt. Das Konzept zur Umsetzung der Regeln wird im Folgenden erläutert. Zur Umsetzung der Regeln werden vier Lookup Tables verwendet. Die Lookup Table VLAN Table

wird zur Umsetzung von VLANs verwendet. Die VID wird als Eingabe verwendet, im RAM werden Flags zum Anzeigen der zum VLAN gehörenden Ports sowie Flags für die im VLAN geltenden Regeln mit *all group*, *all individual* und *all unregistered group* Wildcards abgelegt.

In den Lookup Tables First Level Table und Second Level Table werden die statischen und dynamischen Regeln mit gegebenen Zieladressen abgelegt. Als Eingabe für das CAM dienen Zieladresse und VID der Pakete. Über die Maske der Einträge kann gegebenenfalls VID maskiert werden, somit gilt eine Regel für alle Pakete einer bestimmten Zieladresse ohne Beachtung der VID. Die Zieladresse kann ebenso vollständig oder in Teilen maskiert werden. Im RAM wird die für die Pakete zu verwendende Paketklasse sowie für jeden Port ein Flag abgelegt. In der Second Level Table zeigt das Flag an, ob Pakete bei Verwendung dieser Regel auf dem zugehörigen Port gefiltert oder weitergeleitet werden sollen. Die Flags der First Level Table können zusätzlich für jeden Port anzeigen, dass auf dynamische Regeln zurückgegriffen werden soll. Indem statische Regeln in der First Level Table und dynamische Regeln in der Second Level Table abgelegt werden, werden statische Regeln dynamischen Regeln vorgezogen. Wie erläutert, wird durch das auf dem FPGA verfügbare CAM jeweils die Adresse des ersten der Eingabe entsprechenden Eintrags ausgegeben. Eine höhere Priorisierung der statischen Regeln lässt sich jedoch nicht durch eine entsprechende Sortierung umsetzen, da eine statische Regel für ein Paket auf einem Port gelten kann, während sie für einen anderen Port die Verwendung der dynamischen Regeln vorschreibt. In diesem Fall muss eine statische und eine dynamische Regel angewandt werden. Dementsprechend werden die Regeln in der First und Second Level Table abgelegt.

Würde das automatische Lernen der Adressen der über einen Port zu erreichenden Netzwerkknoten durch das eingebettete Prozessorsystem durchgeführt, müssten alle Pakete, auf die keine Regel zutrifft, an das eingebettete Prozessorsystem weitergeleitet werden. Da dabei im schlechtesten Fall die maximale Paketrate aller Ports verarbeitet werden muss, würde dies zu einer hohen Auslastung des Prozessorsystems führen. Dementsprechend wird das automatische Lernen von Adressen in Hardware umgesetzt. Die automatisch gelernten Adressen werden in der Lookup Table Autolearning Table abgelegt. Im CAM wird die Adresse abgelegt und im RAM der Index des zugehörigen Ports. Das automatische Lernen wird durch die Komponente Autolearning durchgeführt. Hierzu wird für jedes Paket, auf das keine Regel zutrifft, geprüft, ob für die Quelladresse bereits ein Eintrag in der Autolearning Table angelegt ist. Existiert kein Eintrag in der Autolearining Table, wird die Quelladresse mit dem zugehörigen eingehenden Port in der Autolearning Table abgelegt.

Für die VLAN Table wird die in Tabelle 4.5 angegebene Belegung verwendet. Die verwendete Belegung der First Level Table ist in Tabelle 4.6 angegeben, die der Second Level Table in

Feld	Ort	Breite	Belegung
VID	CAM	12 bit	VLAN ID
Port Flags	RAM	5 bit	je 1 bit: 0 filter, 1 forward
AllIndividual Flags	RAM	5 bit	je 1 bit: 0 filter, 1 forward
AllGroup Flags	RAM	5 bit	je 1 bit: 0 filter, 1 forward
AllUnregisteredGroup Flags	RAM	5 bit	je 1 bit: 0 filter, 1 forward

Tabelle 4.5: Belegung der Lookup Table VLAN Table

Feld	Ort	Breite	Belegung
VID	CAM	12 bit	VLAN ID
MAC	CAM	48 bit	Zieladresse
Port Flags	RAM	10 bit	je 2 bit: 0 filter, 1 forward, 2 next
QueueID	RAM	3 bit	Paketklasse
CTRL	RAM	2 bit	Kommandos für CTRL Exec

Tabelle 4.6: Belegung der Lookup Table First Level Table

Tabelle 4.7. Die Autolearning Table verwendet die in Tabelle 4.8 angegebene Belegung. Somit können alle Regeln abgebildet werden. Trifft ein interner Paketheader in der FDB ein, werden die benötigten Informationen extrahiert und an alle Lookup Tables eine Anfrage mit den entsprechenden Daten abgesetzt. Die Ergebnisse der Anfragen werden an die Komponente Evaluation weitergegeben. Die Komponente Evaluation ermittelt daraus, an welche Ports das Paket weitergegeben werden soll, und aktualisiert entsprechend die Felder *out_port*, *queue_id* und *ctrl* im internen Paketheader, sofern nicht das Flag in Feld *fdb_overwr* gesetzt ist.

4.3.2 Umsetzung der FDB

Die einzelnen Tabellen werden mit der Komponente Lookup Table instantiiert. Durch entsprechende Parameter kann die Breite der Einträge CAM und RAM an den Bedarf der einzelnen Komponenten angepasst werden. Weiterhin ist die Anzahl der Einträge in der Lookup Table

Feld	Ort	Breite	Belegung
VID	CAM	12 bit	VLAN ID
MAC	CAM	48 bit	Zieladresse
Port Flags	RAM	10 bit	je 1 bit: 0 filter, 1 forward
QueueID	RAM	3 bit	Paketklasse
CTRL	RAM	2 bit	Kommandos für CTRL Exec

Tabelle 4.7: Belegung der Lookup Table Second Level Table

Feld	Ort	Breite	Belegung
MAC	CAM	48 bit	Zieladresse
Port	RAM	3 bit	Ausgangsport

Tabelle 4.8: Belegung der Lookup Table Autolearning Table

zu konfigurieren. Durch eine entsprechende Belegung der Einträge sind die Regeln der einzelnen Tabellen umzusetzen. Da für den grundlegenden Betrieb zur Evaluation der RT-Bridge im Rahmen dieser Arbeit keine vollständige Umsetzung der Filtering Database benötigt wird, wurde die Komponente Evaluation in reduziertem Umfang umgesetzt. Dabei wird lediglich auf die Ergebnisse der First Level Table zurückgegriffen. Die *PortMap* verfügt über die Belegungen *filter* und *forward*. In der *PortMap* wird für jeden Port jeweils das höherwertige Bit verwendet, für die Belegung *forward* wird das Bit gesetzt.

4.3.3 Input Arbiter

Wie in Kapitel 3.2 beschrieben, nimmt der Input Arbiter interne Paketheader sowohl von den Pre Processing Modulen als auch vom eingebetteten Prozessorsystem entgegen und gibt sie nacheinander an die FDB weiter. In Kapitel 3.2 wurde weiterhin gezeigt, dass es bei maximaler Paketrate von allen Ports zu keinen Paketverlusten kommen kann, wenn für jeden Port mindestens ein Paket gepuffert werden kann und der nächste, an die FDB zu übermittelnde interne Paketheader nach dem Round-Robin Verfahren ausgewählt wird. Die internen Paketheader werden über das in Abschnitt 4.2.1 beschriebene Interface an den Input Arbiter übergeben und vom Input Arbiter an die FDB weitergeleitet. Die Wörter der eingehenden Paketheader werden für jeden Port in einer FIFO gepuffert.

Das Signal *empty* der FIFOs wird gesetzt, wenn die FIFO leer ist. Durch negieren des Signals *empty* ergibt sich, ob die FIFO mindestens ein Wort enthält und ein Paketheader weiterzuleiten ist. Anhand aller negierten *empty* der FIFOs wählt die im nächsten Abschnitt 4.3.4 beschriebene Komponente RR-Selector den nächsten weiterzuleitenden Paketheader aus. Der ausgewählte Paketheader wird aus der FIFO extrahiert und an die FDB geleitet.

4.3.4 RR-Selector

Der RR-Selector wird sowohl im Input Arbiter als auch im Buffer Management verwendet. Im Folgenden werden erst die an den RR-Selector gestellten Anforderungen und danach seine Umsetzung beschrieben. Im Input Arbiter wählt der RR-Selector den nächsten an die FDB zu leitenden Paketheader, im Buffer Management die als nächstes zu verarbeitende Anfrage aus.

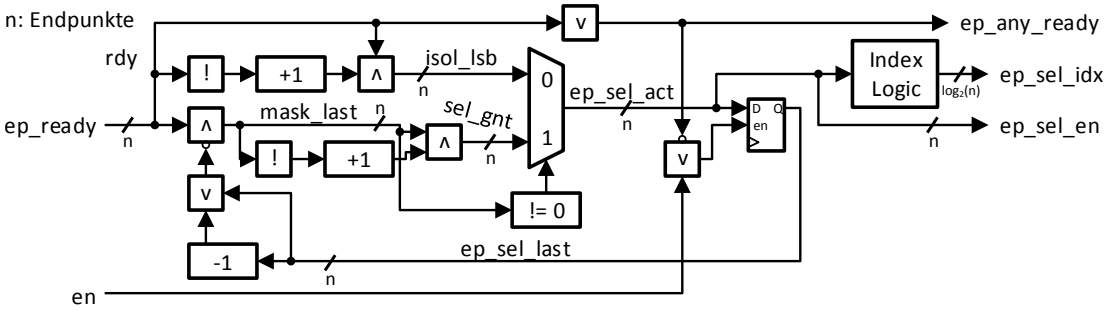


Abbildung 4.18: Umsetzung des RR-Selectors

An den RR-Selector meldet somit eine Reihe von Endpunkten, ob sie bereit sind, eine Anfrage oder ein Paket abzusetzen. Dabei müssen alle Endpunkte regelmäßig ausgewählt werden, da sonst nicht, wie in Kapitel 3.2 beschrieben, Paketverluste ausgeschlossen werden können.

Wie in Kapitel 3.2 beschrieben erfüllt das in [56, S. 152] beschriebene Round-Robin Verfahren diese Anforderungen. Um weiterhin Latenzen durch den Input Arbiter zu minimieren sowie die Verarbeitung einer Anfrage zu jedem Takt im Buffer Management zu garantieren, muss zu jedem Takt ein neuer Endpunkt ausgewählt werden. Sind beispielsweise von den vier Endpunkten 1 bis 4 die Endpunkte 2 und 4 bereit und der Endpunkt 2 wurde als letztes ausgewählt, muss direkt Endpunkt 4 ausgewählt werden. Eine Auswahl des nächsten Endpunktes durch einen zu jedem Takt inkrementierten Zähler erfüllt diese Anforderung nicht, da der Endpunkt 3 des Beispiels nicht übersprungen werden würde.

Die Umsetzung des RR-Selectors ist in Abbildung 4.18 dargestellt. Über das Signal *ep_ready* melden die Endpunkte ihre Bereitschaft und über *en* wird gemeldet, ob die Anfrage des zuletzt ausgewählten Endpunktes verarbeitet wurde. Mit dem Signal *ep_any_ready* zeigt der RR-Selector an, ob aktuell ein Endpunkt bereit ist. Wenn ein Endpunkt bereit ist, wird der ausgewählte Endpunkt durch *ep_sel_idx* als Index und durch *ep_sel_en* als Flag angezeigt.

Um den RR-Selectors mit 26 Endpunkten im Buffer Management mit der geplanten Kernfrequenz zu betreiben musste die Umsetzung auf geringe Signallaufzeiten optimiert werden. Zur Umsetzung von Ripple-Carry Addierern verfügt der verwendete FPGA über besonders schnelle Leitungen zwischen nebeneinander liegenden Logikzellen. Sie werden als Carry Logic Chain [29, Module 2 S. 35] bezeichnet und leiten bei Addierern das im kritischen Pfad liegende Carry Signal mit besonders geringen Laufzeiten weiter. Dementsprechend basiert der in [32] beschriebene und für die Umsetzung des RR-Selectors verwendete Round-Robin Arbiter auf Addierern und Subtrahierern. Dabei werden bei 26 Endpunkten Signallaufzeiten von weniger als 8 ns erreicht.

4.3.5 Output Port Multiplexer

Nachdem durch die FDB die ausgehenden Ports für die Pakete ermittelt wurden, werden die Pakete durch den Output Port Multiplexer an die den jeweiligen Ports zugeordneten Packet Buffering Module sowie an das eingebettete Prozessorsystem übergeben. Dabei müssen, wie in Kapitel 3.4 erläutert, die Referenzzähler der durch die Pakete verwendeten Paketpuffer aktualisiert werden. Dementsprechend werden aus den internen Paketheadern die Felder *out_port* sowie *buff_id* extrahiert und der Paketheader in Schieberegistern gepuffert. Nach dem Extrahieren der benötigten Felder werden die Headerworte an alle ausgehenden Ports weitergeleitet, durch ein logisches „Und“zwischen *sof* Signal und den Flags des *out_port* Feldes werden die ausgehenden Ports selektiert.

Zum Aktualisieren des Referenzzählers werden die gesetzten Bits des *out_port* Feldes gezählt und für die bereits allozierte Referenz um den Wert 1 dekrementiert. Der daraus resultierende signed Wert stellt somit die durchzuführende Korrektur des Referenzzählers dar. Ist der Wert positiv, wird der Referenzzähler im Buffer Management inkrementiert und der Wert dekrementiert, bis er 0 ist. Ist der Wert negativ, werden Referenzen freigegeben und somit der Referenzzähler im Buffer Management dekrementiert und der Wert inkrementiert, bis er 0 ist. Das für den Zugriff auf das Buffer Management verwendete Interface wird in Abschnitt 4.5.1 beschrieben.

4.4 Paketpufferung

Im Folgenden wird die Umsetzung des in Kapitel 3.5 beschriebenen Konzepts des Packet Buffering Moduls erläutert. Hierfür wird in den folgenden Absätzen erst ein Überblick über die Komponenten des Packet Buffering Moduls gegeben. Danach wird in Abschnitt 4.4.1 auf die verwendeten Interfaces und in den darauf folgenden Abschnitten auf die Umsetzung der einzelnen Komponenten eingegangen.

Das Packet Buffering Modul beherbergt für jede Paketklasse eine Kombination aus einer Queue und TSA. In den in Abschnitt 4.4.3 beschriebenen Queues werden die internen Paketheader der zu puffernden Pakete abgelegt. Durch die in Abschnitt 4.4.4 beschriebenen TSAs wird der Sendezeitpunkt für Pakete der zugeordneten Paketklasse bestimmt. Aus den durch die TSAs angemeldeten Sendezeitpunkte wird durch den in Abschnitt 4.4.5 beschriebenen Queue Arbiter die aktive Paketklasse ausgewählt. Der in Abschnitt 4.4.2 beschriebene Queue Multiplexer leitet die eingehenden Paketheader an die durch die FDB bestimmte Queue weiter. Im letzten Abschnitt 4.4.6 wird auf die im Rahmen dieser Arbeit verwendeten Paketklassen eingegangen.

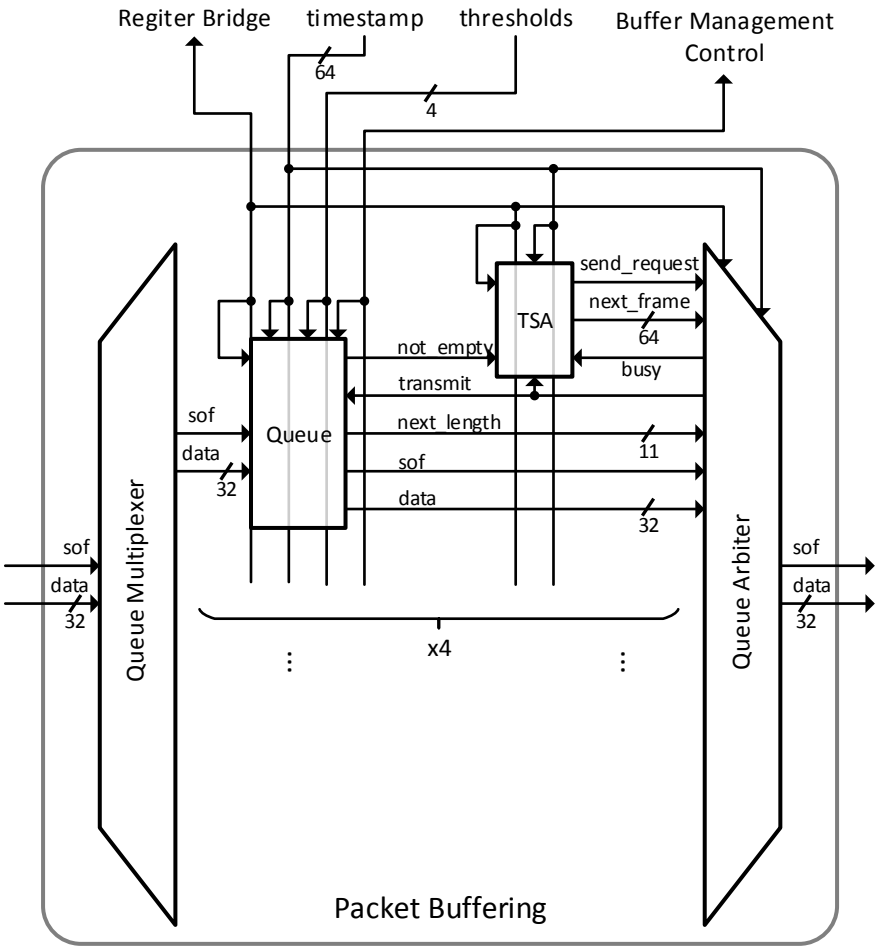


Abbildung 4.19: Interfaces der Komponenten im Packet Buffering Modul

4.4.1 Interfaces

Die Interfaces der Komponenten des Packet Buffering Moduls sind in Abbildung 4.19 dargestellt. Das aus den Signalen *sof* und *data* bestehende Interface zum internen Vermitteln der Paketheader wurde in Abschnitt 4.2.1 beschrieben. Das Interface zum internen Vermitteln der Paketheader wird für die eingehenden und ausgehenden Paketheader am Queue Multiplexer, dem Queue Arbiter sowie den Queues verwendet. Die Interfaces zur Register Bridge und zum Buffer Management werden in den Abschnitten 4.5.1 und 4.6.1 beschrieben. Die verbleibenden Interfaces werden nun beschrieben, danach folgt ein Beispiel.

Alle Signale der folgenden Interfaces verfügen zu jeder steigenden Taktflanke des Kerntaktes über gültige Werte. Das Signal *timestamp* wird durch das eingebettete Prozessorsystem bereitgestellt und stellt allen Komponenten auf der RT-Bridge eine global synchronisierte

Zeitbasis zur Verfügung. Es ist als 64 bit breite Festkommazahl ohne Vorzeichen mit der Einheit Nanosekunde zu interpretieren. Das Komma befindet sich zwischen Bit 15 und 16, somit handelt es sich bei den 16 Least Significant Bit um die Nachkommastellen.

Das Interface zwischen Queue und TSA besteht aus dem Signal *not_empty*. Ist es gesetzt, verfügt die Queue über ein zu sendendes Paket. Weiterhin meldet die Queue über das Signal *next_len* die Länge des Paketes an den Queue Arbiter. Das Signal *next_len* wird als 11 bit Integer ohne Vorzeichen betrachtet und ergibt sich aus dem Paketheaderfeld *len* und sämtlichen auf der Leitung auftretenden zusätzlichen Headern. Die Berechnung von *next_len* ist in Kapitel 3.5 beschrieben.

Der TSA meldet über das Signal *send_request* dem Queue Arbiter, dass ein Paket seiner Paketklasse bereit zum Senden ist. Ist *send_request* gesetzt, ist ein Paket sendebereit und die Signale *next_len* der Queue und *next_frame* des TSA müssen Informationen über das zu sendende Paket beinhalten. *next_frame* beinhaltet den Sendezeitpunkt des zu sendenden Paketes und wird wie das Signal *timestamp* interpretiert.

Der Queue Arbiter zeigt den Queues und TSAs über das Signal *transmit* an, wenn ein Paket der zugehörigen Paketklasse gesendet wird. Ist *transmit* gesetzt, wird ein Paket übermittelt, Queues müssen zur positiven Flanke von *transmit* den Paketheader ausgeben. Das Signal *busy* wird bei der Übermittlung eines Paketes auf dem Port durch den Queue Arbiter gesetzt und zeigt somit allen TSAs an, ob auf dem zugehörigen ausgehenden Port aktuell gesendet wird.

In Abbildung 4.20 ist ein beispielhafter Paketversand dargestellt. Das Beispiel beinhaltet die zwei Paketklassen 0 und 1, die Paketklasse 0 hat die höchste Priorität. Signale mit Indizes sind einer Paketklasse entsprechend dem Index zugeordnet, bei *sof_in* und *data_in* handelt es sich um die Eingänge des Queue Multiplexers. Die Latenz des Queue Multiplexers wird hier vernachlässigt, die Signale werden bei dem eintreffenden Paket direkt an die Queue der Paketklasse 0 geleitet. Bei den *queue_sof[]* und *queue_data[]* handelt es sich um das ausgehende Interface zur Weiterleitung der Paketheader der Queues. Wurde eine Paketklasse arbitriert und sendet sie den Paketheader, wird dieser durch den Queue Arbiter verzögert auf *sof_out* und *data_out* ausgegeben. Zu Beginn wird ein Paket der Klasse 1 gesendet. Während das Paket gesendet wird, erhält die Klasse 0 zugeordnete Queue ein Paket. Nach dem Eintreffen des Paketes setzt die Queue das Signal *not_empty* und der TSA darauf das Signal *send_request* und fordert somit das Senden eines Paketes an. Nach dem Ende des Pakets der Paketklasse 1 findet der Arbitrierungsprozess statt und das Paket der Klasse 0 wird ausgewählt und gesendet.

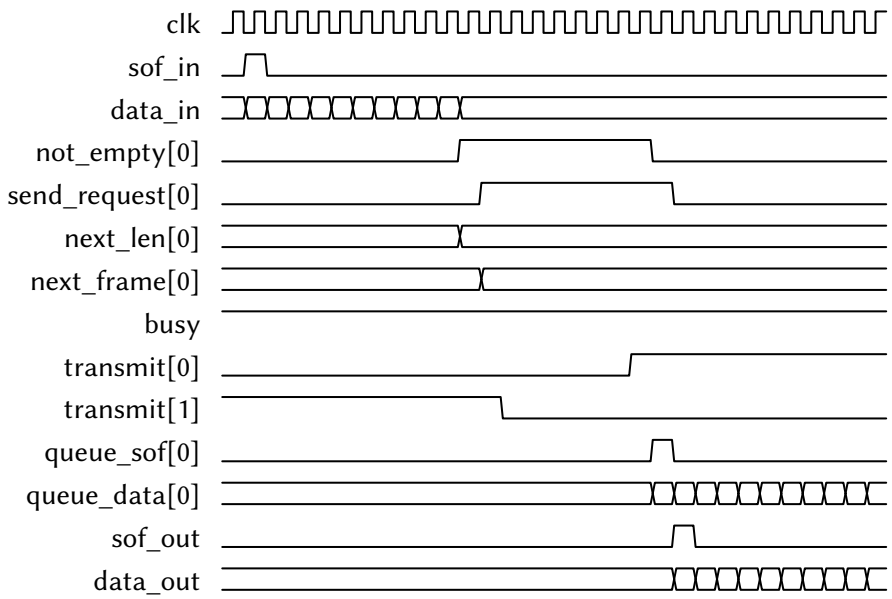


Abbildung 4.20: Beispiel der Paketauswahl im Packet Buffering Modul

4.4.2 Queue Multiplexer

Wie beschrieben, nimmt der Queue Multiplexer die Pakete vom Output Port Multiplexer entgegen und leitet sie anhand des Feldes *queue_id* an die für die Paketklasse zuständige Queue weiter. Dementsprechend wird das Feld *queue_id* aus dem Paketheader extrahiert und der Paketheader in einer FIFO gepuffert. Wurde das benötigte Feld extrahiert, wird der Paketheader aus der FIFO entnommen und an die entsprechende Queue weitergeleitet.

4.4.3 Queues

Zur Umsetzung der RT-Ethernet Konzepte TTE, AFDX und AVB müssen den jeweiligen Anforderungen entsprechende Queues umgesetzt werden. Weiterhin wird für Pakete ohne Anforderungen an die Echtzeitfähigkeit eine Queue benötigt. Für AFDX und AVB besteht lediglich die Anforderung, dass die Reihenfolge von Paketen nicht verändert wird. Somit werden die Pakete in der Komponente FIFO Queue in einer FIFO gepuffert. Die FIFO Queue wird für die RT-Ethernet Konzepte AFDX und AVB sowie Pakete ohne Echtzeitanforderungen verwendet. Im Folgenden wird erst die Umsetzung der FIFO Queue und danach die Umsetzung der TT Queue für TTE beschrieben.

Bei der FIFO Queue werden die eingehenden Worte der Paketheader in einer 32 bit breiten und 256 Wörter tiefen FIFO abgelegt. Ein Mealy-Zustandsautomat steuert die Entnahme und

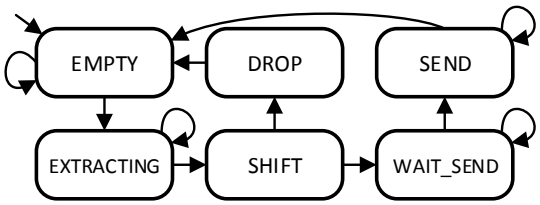


Abbildung 4.21: Mealy-Zustandsautomat der FIFO Queue

Verarbeitung der Pakete. Er verfügt über die in Abbildung 4.21 dargestellten Zustandsübergänge zwischen den folgenden Zuständen:

- **EMPTY:** Die FIFO ist leer. Übergang in Zustand **EXTRACT**, wenn sich ein Wort in der FIFO befindet.
- **EXTRACT:** Lesen eines Paketheaders aus der FIFO und extrahieren der Felder *buff_id* und *len*. Der Paketheader wird in einem Schieberegister gepuffert.
- **SHIFT:** Letztes Wort des Paketheaders in Schieberegister ablegen.
- **WAIT_SEND:** Das Signal *not_empty* ist in diesem Zustand gesetzt. Wechseln in den Zustand **SEND**, wenn eine positive Flanke des Signals *transmit* erkannt wird, oder in den Zustand **DROP**, wenn die FIFO voll ist oder das Buffer Management über das *threshold* Signal das Verwerfen von Paketen anfordert.
- **SEND:** Paketheader wird aus dem Schieberegister ausgegeben.
- **DROP:** Paketpuffer wird freigegeben und das Paket somit verworfen.

Wie ersichtlich, wird das sich im Schieberegister befindliche und somit älteste Paket im Zustand **DROP** verworfen. Dies geschieht einerseits, wenn das Buffer Management meldet, dass die Anzahl der freien Paketpuffer einen Schwellwert erreicht hat und Paketpuffer freigegeben werden müssen. Andererseits werden Pakete verworfen, wenn ein neues Paket in der FIFO Queue eintrifft und nicht mehr gepuffert werden könnte und durch das einfache Verwerfen des eingehenden Paketheaders die Referenz auf den Paketpuffer nicht freigegeben werden würde.

Das Signal *next_len* ergibt sich wie in Kapitel 3.5 beschrieben wie folgt:

$$\begin{aligned} \textit{next\_len} &= \textit{len} + l_{IFG} + l_{Preambel} + l_{Header} + l_{VLANTag} + l_{FCS} \\ \textit{next\_len} &= \textit{len} + 12\ \mathrm{B} + 8\ \mathrm{B} + 12\ \mathrm{B} + 4\ \mathrm{B} + 4\ \mathrm{B} \\ \textit{next\_len} &= \textit{len} + 40\ \mathrm{B} \end{aligned}$$

Dabei wird nicht beachtet, ob ein VLAN Tag gesendet wird. Daraus resultiert gegebenenfalls eine zu lange Arbitrierung der ausgehenden Leitung für das zu sendende Paket durch den Queue Arbiter. Da dies in keinem Fall zur unkontrollierten Verzögerung nachfolgender Pakete führen kann, hat dies keinen Einfluss auf die Echtzeitfähigkeit. Allerdings wird, abhängig von der Paketgröße, die verfügbare Bandbreite nicht vollständig ausgenutzt. Durch den Verzicht auf die Berücksichtigung der VID muss in den FIFO Queues nicht die für Pakete ohne VLAN Tags verwendete VID konfiguriert werden.

Zur Umsetzung des RT-Ethernet Konzeptes TTE muss die TT Queue für jede zu vermittelnde Nachricht ein festen Speicherplatz bereitstellen. Die eingehenden Paketheader müssen anhand ihrer Zieladresse in dem für sie vorgesehenen Speicherplatz abgelegt werden. Weiterhin ist bei der Ankunft der Nachricht das Empfangszeitfenster zu prüfen und die Nachrichten müssen anhand eines zu konfigurierenden Schedules versandt werden. Weiterhin ist über die angegebene Paketlänge *next_len* die Länge des durch den Queue Arbiter zu reservierenden Zeitfensters zum Paketversand entsprechend dem Schedule zu beeinflussen.

Die Paketheader werden im Header RAM abgelegt. Der Header RAM ist 32 bit breit und kann 16 Paketheader aufnehmen. Er ist in Slots zu je 10 Wörtern aufgeteilt, ein Slot nimmt einen Paketheader auf. Auf den Header RAM kann gleichzeitig über zwei Ports schreibend und lesend auf unterschiedliche Adressen zugegriffen werden. Bei den Header RAM Slot Valid Flags handelt es sich ebenfalls um einen Dual Ported RAM, allerdings 1 bit breit. Die Zellen zeigen an, ob sich in einem Slot ein Paketheader befindet.

Zum Ablegen des Schedules für die eingehenden Pakete wird die in Abschnitt 4.3 beschriebene Komponente Lookup Table genutzt. Als Eingabe für das CAM wird die Zieladresse verwendet. Das RAM beinhaltet Anfangs- und Endzeitstempel des aktuellen Zeitfensters, die erwartete Paketlänge, die Ports, auf denen das Paket empfangen werden darf, sowie die Startadresse zum Ablegen des Paketheaders im Header RAM. Die Zeitstempel werden entsprechend dem Signal *timestamp* interpretiert. Durch das Ablegen des zum aktuellen Zeitpunkt gültigen Empfangszeitfensters in Form von Zeitstempeln kann auf Logik zum Errechnen des zur Ankunftszeit eines Paketes gültigen Zeitfensters verzichtet werden. Allerdings müssen die Zeitstempel durch das eingebettete Prozessorsystem laufend aktuell gehalten werden.

Beim Empfang eines Paketheaders werden somit erst die Felder *dst_addr*, *in_port*, *buff_id*, *len* und *in_tstmp* extrahiert und der Paketheader in einem Schieberegister gepuffert. Mit der extrahierten Zieladresse wird eine Anfrage an die Lookup Table abgesetzt. Der zurückgegebene Eintrag wird mit den extrahierten Feldern abgeglichen. Entspricht das eingegangene Paket den Erwartungen, wird es an der angegebenen Startadresse im Header RAM abgelegt und das betreffende Header RAM Slot Valid Flag gesetzt. Entspricht der eingegangene Paketheader

nicht den Erwartungen oder die Lookup Table enthielt für die angegebene Zieladresse keinen Eintrag, wird der Paketpuffer des Paketes freigegeben und der Paketheader verworfen.

Zum Ablegen des Schedules für ausgehende Pakete wird eine FIFO verwendet. Sie kann bis zu 16 Einträge aufnehmen. Ein Eintrag enthält die Startadresse des zu sendenden Paketes im Header RAM, die Adresse des Header RAM Slot Valid Flags, die Länge des durch den Queue Arbiter zu reservierenden Sendefensters in Byte sowie einen Zeitstempel, an dem das Paket spätestens gesendet worden sein muss. Der Beginn des Sendefenster wird durch den zugehörigen TSA festgelegt.

Die TT Queue meldet Sendebereitschaft über *not_empty*, wenn in der FIFO Einträge vorhanden sind. Das Signal *next_len* wird auf die, im aktuell zu verarbeitenden Eintrag gesetzte Paketlänge gesetzt. Sind *transmit* und das betreffende Header RAM Slot Valid Flag gesetzt, wird der Paketheader aus der entsprechenden Stelle im Header RAM gelesen und ausgegeben, das Header RAM Slot Valid Flag gelöscht und der nächste Eintrag aus der FIFO gelesen. Sobald der späteste Sendezeitpunkt in der Vergangenheit liegt, also *timestamp* numerisch größer als der Zeitstempel ist, und das zu sendende Paket nicht eingetroffen ist, wird der aktuelle Eintrag verworfen und der nächste aus der FIFO gelesen. Wie ersichtlich, müssen die Einträge der FIFO kontinuierlich über das Register Interface der TT Queue durch das eingebettete Prozessorsystem zugespielt werden. Somit können Pakete zu definierten Zeitpunkten gesendet und mit Hilfe des Queue Arbiters die Länge des Sendefensters beeinflusst werden. Werden die reservierten Sende- und Empfangsfenster nicht genutzt, wird dies entsprechend behandelt.

4.4.4 Transmission Selection Algorithms

Wie in Kapitel 3.5 beschrieben, muss für jedes RT-Ethernet Konzept ein TSA umgesetzt werden. Der Strict Priority TSA setzt eine Arbitrierung anhand von Prioritäten um und wird für Pakete ohne Echtzeitanforderungen verwendet. Mit einer Kombination mehrerer Strict Priority TSA lassen sich die unterschiedlichen Prioritäten von IEEE 802.1Q VLANs abbilden. Der AFDX TSA setzt einen AFDX Traffic Shaper um, der CBS TSA das Credit Based Traffic Shaping von AVB. Der TT TSA führt in Kombination mit der TT Queue den Versand von Paketen anhand eines definierten Schedules aus. Im Folgenden werden die einzelnen TSAs beschrieben.

Der Strict Priority TSA meldet ein zu sendendes Paket über *send_request* an, wenn das Signal *not_empty* gesetzt ist. Als Sendezeitpunkt übermittelt er die aktuelle Zeit, *timestamp* wird also direkt an *next_frame* angelegt. Der Strict Priority TSA verfügt über keinerlei, über die Register Bridge zu konfigurierende Register.

Der AFDX TSA verfügt über ein Register, in dem die Bandwidth Allocation Gap (BAG) über die Register Bridge konfiguriert wird. Das Register zur Konfiguration der BAG ist $27\ \mathrm{bit}$ breit, das LSB (Least Significant Bit) hat eine Wertigkeit von $8\ \mathrm{ns}$. Somit beträgt die maximal zu konfigurierende BAG $(2^{27} - 1) * 8\ \mathrm{ns} > 1\ \mathrm{s}$ bei einer Genauigkeit von $8\ \mathrm{ns}$. Zum Zeitpunkt der Umsetzung des AFDX TSA lag die Periode der verwendeten Taktrate bei $8\ \mathrm{ns}$, eine höhere Genauigkeit des BAG Wertes würde also auf Grund der verwendeten Taktrate nicht zu einer höheren Genauigkeit bei der Umsetzung der BAG führen. Mit der Reduzierung der Kerntaktrate auf $62,5\ \mathrm{MHz}$ und eine Periode von $16\ \mathrm{ns}$ wurde die Genauigkeit des AFDX TSA nicht angepasst, da durch eine entsprechende Reduzierung der Genauigkeit des konfigurierten BAG Wertes zu keiner signifikanten Reduktion der verwendeten Ressourcen geführt hätte.

Wird ein Paket der dem TSA zugeordneten Paketklasse versandt, also eine positive Flanke am *transmit* Signal detektiert, wird der aktuelle Zeitstempel *timestamp* mit dem Wert des BAG Registers addiert und im Register *bag_end* abgelegt. Daraus ergibt sich der Zeitpunkt, an dem das nächste Paket versandt werden darf. Ein zu versendendes Paket wird über *send_request* angemeldet, wenn *not_empty* gesetzt ist. Ist der in *bag_end* gespeicherte Wert größer als der aktuelle Zeitstempel in *timestamp*, wird an *next_frame bag_end* angelegt, ansonsten *timestamp*. Somit wird die für AFDX benötigte BAG zwischen den Paketen eines virtuellen Links umgesetzt.

Der CBS TSA verfügt jeweils über ein $51\ \mathrm{bit}$ breites Register zur Konfiguration von Send und Idle Slope über die Register Bridge. Die Slopes ergeben sich aus der bei der Reservierung über das SRP angegebenen Paketlänge und der Paketrate. Der Idle Slope entspricht der für die AVB Paketklasse reservierten Bandbreite, Send Slope ergibt sich wie folgt:

$$\text{Send Slope} = \text{Idle Slope} - b_{Port}$$

Bei b_{Port} handelt es sich um die am Port verfügbare Bandbreite. Dementsprechend ist der Send Slope laut [15, Kapitel 8.6.8.2] negativ. Im Konfigurationsregister wird der Send Slope als positive Zahl ohne Vorzeichen gehalten, dementsprechend ist der ermittelte Send Slope mit -1 zu multiplizieren.

Der IEEE 802.1Q Standard [15] sieht für die Slopes die Einheit $\mathrm{bit/s}$ vor. Da die Paketlänge in Byte angegeben wird und es sich bei der Paketrate um einen ganzzahligen Wert mit der Einheit $\mathrm{Pakete/s}$ handelt, beträgt die Wertigkeit des LSB für die Angabe der Slopes in den Konfigurationsregistern $8\ \mathrm{bit/s} = 1\ \mathrm{B/s}$. Soll für die Paketklasse eine Bandbreite von $1\ \mathrm{Mbit/s}$ bei $100\ \mathrm{Mbit/s}$ Übertragungsrate des Ports konfiguriert werden, ergibt sich ein Idle Slope von $1\ \mathrm{Mbit/s}$ und ein Send Slope von $99\ \mathrm{Mbit/s}$. Dementsprechend ist im Idle Slo-

pe Register ein Wert von $\frac{1\ \mathrm{Mbit/s}}{8\ \mathrm{bit/s}} = 1,25 * 10^5$ und im Send Slope Register ein Wert von $\frac{99\ \mathrm{Mbit/s}}{8\ \mathrm{bit/s}} = 1,2375 * 10^7$ zu konfigurieren.

Der aktuelle Credit wird im Register *credit* gehalten und mit dem Kerntakt vom $125\ \mathrm{MHz}$ aktualisiert. Im IEEE 802.1Q Standard ist für den Credit die Einheit bit festgelegt. Wie erläutert, beträgt die Wertigkeit des LSB der Slopes $8\ \mathrm{bit/s}$. Das Register *credit* wird mit einem Takt von $62,5\ \mathrm{MHz}$ aktualisiert. Somit beträgt die Wertigkeit des LSB des *credit* Registers $\frac{8\ \mathrm{bit/s}}{62{,}5\ \mathrm{MHz}} = \frac{8\ \mathrm{bit{*}s^{-1}}}{6{,}25{*}10^7\ \mathrm{s^{-1}}} = 1,28 * 10^{-7}\ \mathrm{bit}$. Im Folgenden wird der durch das *credit* Register abzudeckende Wertebereich hergeleitet.

Um bei einer Umstellung des Kerntaktes auf $125\ \mathrm{MHz}$ und der Datenrate der Ports auf $1000\ \mathrm{Mbit/s}$ keine Anpassungen am CBS TSA durchführen zu müssen, wird er entsprechend ausgelegt. Bei einem Kerntakt von $125\ \mathrm{MHz}$ beträgt die Wertigkeit des LSB des *credit* Registers $\frac{8\ \mathrm{bit/s}}{125\ \mathrm{MHz}} = 6,4 * 10^{-8}\ \mathrm{bit}$. Der minimal zu erreichende Credit ergibt sich, wenn ein Paket mit maximaler Länge bei einem anfänglichen Credit von $0\ \mathrm{bit}$ und einem minimalen Send Slope gesendet wird. Der Send Slope erreicht sein Minimum beim kleinsten möglichen Idle Slope. Der kleinste mögliche Idle Slope ergibt sich, wenn ein Stream mit der minimalen Paketrate von $8000\ \mathrm{Pakete/s}$ und einer minimalen Paketlänge reserviert wird. Die minimale Länge der Nutzdaten $l_{Payload}$ eines Paketes beträgt $46\ \mathrm{B}$. Die minimale Paketlänge l_{min} auf der Leitung ergibt sich aus der Länge der IFG l_{IFG}, der Länge des Headers inklusive Type Feld l_{Header} sowie der Länge der FCS l_{FCS}wie folgt:

$$
\begin{aligned}
l_{min} =& l_{IFG} + l_{Preambel} + l_{Header} + l_{Payload} + l_{FCS} \\
l_{min} =& l_{IFG} + 8\ \mathrm{B} + 14\ \mathrm{B} + 46\ \mathrm{B} + 4\ \mathrm{B} = l_{IFG} + 72\ \mathrm{B}
\end{aligned}
$$

Zu Beachten ist, dass ein Paket minimaler Länge kein VLAN Tag enthält und das die IFG bei $100\ \mathrm{Mbit/s}$ Datenrate des Ports $12\ \mathrm{B}$ und bei $1000\ \mathrm{Mbit/s}$ $8\ \mathrm{B}$ beträgt. Somit ergibt sich:

$$
\begin{aligned}
l_{min\ 100\ \mathrm{Mbit/s}} =& 84\ \mathrm{B} \\
l_{min\ 1000\ \mathrm{Mbit/s}} =& 80\ \mathrm{B}
\end{aligned}
$$

Daraus ergibt sich die minimal zu reservierende Bandbreite b_{min} sowie der kleinste Send Slope für 100 Mbit/s und 1000 Mbit/s Datenrate des Ports:

$$
\begin{aligned}
b_{min} &= l_{min} * 8000 \text{ Pakete/s} \\
b_{min\ 100\ \text{Mbit/s}} &= 84\ \text{B} * 8000\ \text{Pakete/s} = 5,376\ \text{Mbit/s} \\
b_{min\ 1000\ \text{Mbit/s}} &= 80\ \text{B} * 8000\ \text{Pakete/s} = 5,12\ \text{Mbit/s} \\
\text{Send Slope}_{min} &= b_{Port} - \text{Idle Slope} = b_{Port} - b_{min} \\
\text{Send Slope}_{min\ 100\ \text{Mbit/s}} &= 100\ \text{Mbit/s} - 5,376\ \text{Mbit/s} = -94,624\ \text{Mbit/s} \\
\text{Send Slope}_{min\ 1000\ \text{Mbit/s}} &= 1000\ \text{Mbit/s} - 5,12\ \text{Mbit/s} = -994,88\ \text{Mbit/s}
\end{aligned}
$$

Wird nun ein Paket mit maximaler Länge l_{max} versendet, ergibt der Credit c_{min} direkt nach dem Senden aus dem Send Slope und der verstrichenen Zeit zum Senden des Paketes t_{send}. Die Zeit zum Senden eines Paketes ergibt sich aus der Paketlänge und der Datenrate des Ports. Die maximale Länge der Nutzdaten eines Paketes entsprechen 1500 B, zusätzlich zu den in l_{min} berücksichtigten Feldern muss die Länge des VLAN Tags $l_{VLANTag}$ berücksichtigt werden. Somit gilt:

$$
\begin{aligned}
l_{max} &= l_{IFG} + l_{Preambel} + l_{Header} + l_{VLANTag} + l_{Payload} + l_{FCS} = l_{IFG} + 1530\ \text{B} \\
t_{send} &= l_{max} / b_{port} \\
c_{min} &= t_{send} * \text{Send Slope}_{min} = \frac{l_{IFG} + 1530\ \text{B}}{b_{port}} * \text{Send Slope}_{min} \\
c_{min\ 100\ \text{Mbit/s}} &= \frac{12\ \text{B} + 1530\ \text{B}}{100\ \text{Mbit/s}} * -94,624\ \text{Mbit/s} = -11673\ \text{bit} \\
c_{min\ 1000\ \text{Mbit/s}} &= \frac{8\ \text{B} + 1530\ \text{B}}{1000\ \text{Mbit/s}} * -994,88\ \text{Mbit/s} = -12241\ \text{bit}
\end{aligned}
$$

Bei der gegebenen reservierten Bandbreite entspräche das Senden eines Paketes mit maximaler Länge zwar nicht den Spezifikationen von AVB, es handelt sich dabei allerdings um ein zu behandelndes Fehlverhalten. Unter Berücksichtigung der Wertigkeit des LSBs ergibt sich ein minimaler Wert von $\frac{-12241\ \text{bit}}{6,4*10^{-8}\ \text{bit}} \approx -1,913 * 10^{11}$ für das *credit* Register.

Der maximal zu erreichenden Credit ergibt sich aus der maximalen Dauer $t_{blocked}$ für die Pakete der AVB Klasse, die durch eine Paketklasse mit höherer Priorität verzögert werden, sowie einem maximalen Idle Slope. In IEEE 802.1Q wird empfohlen, dass maximal 75 % der verfügbaren Bandbreite für AVB Streams verwendet wird. Da es sich um eine Empfehlung handelt, wird für die folgenden Berechnungen die verfügbare Bandbreite als Idle Slope angenommen. In der RT-Bridge verfügt lediglich die Paketklasse TTE über eine höhere Priorität

und kann somit AVB Pakete verzögern. Das Maximum von $t_{blocked}$ durch TTE ergibt sich aus dem konkreten Schedule der TT Nachrichten. Für die folgenden Berechnungen wird davon ausgegangen, dass der Schedule in keinem Fall eine größere Verzögerung als $50\ \mathrm{ms}$ verursacht. Von einer größeren Verzögerung wird nicht ausgegangen, da $50\ \mathrm{ms}$ bereits die in den Kapiteln 1 und 2.5 genannten Latenzen wesentlich übersteigt. Der maximale Credit c_{max} ergibt sich durch die Verzögerung $t_{blocked}$ und den mit der Datenrate des Ports konfigurierten Idle Slope wie folgt:

$$
\begin{aligned}
c_{max} &= t_{blocked} * \text{Idle Slope}_{max} \\
c_{max} &= t_{blocked} * b_{Port} \\
c_{max\ 100\ \mathrm{Mbit/s}} &= 50\ \mathrm{ms} * 100\ \mathrm{Mbit/s} = 5 * 10^6\ \mathrm{bit} \\
c_{max\ 1000\ \mathrm{Mbit/s}} &= 50\ \mathrm{ms} * 1000\ \mathrm{Mbit/s} = 5 * 10^7\ \mathrm{bit}
\end{aligned}
$$

Unter Berücksichtigung der Wertigkeit des LSBs ergibt sich ein maximaler Wert von $\frac{5*10^7\ \mathrm{bit}}{6{,}4*10^{-8}\ \mathrm{bit}} \approx 7,813 * 10^{14}$ für das *credit* Register. Dar hergeleitete, durch das *credit* Register abzudeckende Wertebereich kann durch eine $51\ \mathrm{bit}$ breite, mit dem Zweier-Komplement vorzeichenbehaftete Ganzzahl abgedeckt werden. Der Wertebereich beträgt somit $\pm 1,126 * 10^{15}$ und unter Berücksichtigung der Wertigkeit des LSB $\pm 7,206 * 10^7\ \mathrm{bit}$.

Nachdem die verwendete Breite des *credit* Registers hergeleitet wurde, wird nun dessen Aktualisierung beschrieben. Wird durch die Paketklasse des TSAs aktuell ein Paket gesendet (*transmit* ist gesetzt), wird *credit* um den konfigurierten Send Slope dekrementiert. Wird ein Paket einer anderen Paketklasse (*transmit* nicht gesetzt und *busy* gesetzt) gesendet oder ist *credit* negativ, wird *credit* um Idle Slope inkrementiert. Für die Rechenoperationen zur Aktualisierung von *credit* wird zur Verhinderung von Überläufen eine Sättigung umgesetzt. Ist das Resultat der Aktualisierung von *credit* positiv, die Paketklasse des TSAs sendet nicht und die zugeordnete Queue enthält keine Paketheader (*transmit* und *not_empty* nicht gesetzt), wird *credit* zurückgesetzt. Somit wird Credit, wie im IEEE 802.1Q Standard und Kapitel 2.3.2 beschrieben, aktualisiert.

Das Signal *send_request* wird gesetzt, wenn *not_empty* gesetzt ist und *credit* $>= 0$ gilt. An *next_frame* wird der aktuelle Zeitstempel *timestamp* angelegt. Beim Schreiben der Slope Register über die Register Bridge ist zu beachten, dass die Slope Werte auf zwei Register aufgeteilt sind. Damit zu jedem Zeitpunkt des Schreibvorgangs dem CBS TSA korrekt Slope Werte vorliegen, müssen erst die MSB (Most Significant Bit) und in einem zweiten Zugriff die LSB geschrieben werden. Der Schreibzugriff auf die MSB wird in einem internen Register bis zum Schreibzugriff auf die LSB gepuffert. Erfolgt der Schreibzugriff auf die LSB ohne einen

vorherigen Schreibzugriff auf die MSB, wird der betreffende Slope nicht aktualisiert. Somit erfüllt der CBS TSA die im IEEE 802.1Q Standard beschriebenen Spezifikationen.

In der RT-Bridge setzt der TT TSA in Kombinationen mit der TT Queue das Konzept TT-Ethernet um. Durch die TT Queue werden die Empfangszeitfenster der eingehenden Nachrichten geprüft sowie zum im Schedule definierten Zeitpunkt das zu sendende Paket bereitgestellt. Die TT Queue gibt weiterhin die Länge des Sendezeitfensters an den Queue Arbiter weiter, der dies umsetzt. Somit ist durch den TT TSA lediglich der Beginn des Sendezeitfensters an den Queue Arbiter zu melden. Die Einträge des Schedules werden laufend durch das eingebettete Prozessorsystem in der Schedule FIFO abgelegt und durch den TT TSA ausgeführt. Ein Schedule Eintrag beinhaltet den Startzeitpunkt des Sendefensters im Format des *timestamp* Signals. Befindet sich in der Schedule FIFO mindestens ein Eintrag, wird der als nächstes zu lesende Eintrag verarbeitet. Der TT TSA meldet entsprechend über *send_request* Sendebereitschaft und an *next_frame* wird der im Eintrag angegebene Beginn des Sendezeitfensters angelegt. Der aktuell verarbeitete Eintrag gilt als verarbeitet, wenn an *transmit* eine positive Flanke detektiert wurde. In diesem Fall wird der nächste Eintrag von der Schedule FIFO angefragt. Somit kann ein Schedule für TT-Ethernet vollständig umgesetzt werden.

4.4.5 Queue Arbiter

Wie bereits erläutert, wählt der Queue Arbiter aus allen an einem Port Sendebereitschaft meldenden Paketklassen die als nächstes ein Paket sendende Paketklasse aus. Dabei ist jeder Paketklasse eine Priorität zugewiesen und sie meldet bei Sendebereitschaft den geforderten Sendezeitpunkt und die Länge des nächsten Paketes. Es wird das Paket ausgewählt, dessen Sendezeitpunkt am frühesten ist und das vor den geplanten Sendezeitpunkten aller gemeldeten Pakete mit höherer Priorität zu Ende ist. Wenn eine Paketklasse ausgewählt wurde, werden *transmit* und *busy* entsprechend gesetzt.

Der Queue Arbiter beginnt erst nach dem angemeldeten Paketende mit der Auswahl des nächsten zu sendenden Paketes. Somit werden die Pakete entsprechend der ausgehenden Bandbreite des Ports an das Post Processing Modul weitergegeben. Ein Anstauen von Paketen beim Senden wird verhindert, dementsprechend kann dies als Quelle für nicht deterministische Latenzen ausgeschlossen werden. Die Sendezeitpunkte werden entsprechend dem Signal *timestamp* interpretiert. Somit ist von zwei Sendezeitpunkten der numerisch kleinere Sendezeitpunkt der frühere.

Um sicherzustellen, dass eine gewählte Paketklasse das zu sendende Paket vor den gemeldetem Sendezeitpunkten aller Paketklassen mit höherer Priorität beenden kann, muss aus gemeldeten Startzeitpunkt t_{start} und Paketlänge l der Zeitpunkt des Paketendes t_{end} errechnet

werden. Der Zeitpunkt des Paketendes ergibt sich aus der aktuellen Datenrate b_{Port} des Ports beziehungsweise der sich daraus ergebenden Zeit t_{byte} zur Übertragung eines Bytes wie folgt:

$$t_{byte} = \frac{1}{b_{Port}} * 8 \text{ bit/B}$$
$$t_{end} = t_{start} + l * t_{byte}$$

Für die sendebereiten Paketklassen wird t_{start} über *next_frame* gemeldet und l über *next_len*, t_{byte} wird über ein Register durch das eingebettete Prozessorsystem konfiguriert und steht als *byte_time* zur Verfügung. Der berechnete Zeitpunkt t_{end} wird im Queue Arbiter im Signal *frame_end* gehalten. Der ermittelte Endzeitpunkt des Paketes wird auch zum Abwarten des Endes des ausgewählten Paketes vor der Auswahl des nächsten Paketes verwendet.

Wie ersichtlich, muss für die Auswahl des nächsten zu sendenden Paketes der Endzeitpunkt für alle sendebereiten Pakete errechnet werden. Für die durchzuführende Multiplikation können auf dem FPGA vorhandene Multiplikatoren verwendet werden. Die auf dem FPGA verfügbaren Multiplikatoren sind direkt auf dem Chip umgesetzt und Multiplizieren zwei 18 bit breite Faktoren innerhalb von $4,08$ ns [29, Module 3 S. 36, 65, S. 40] (Speed Grade 7). Da für jede Paketklasse an jedem Port jeweils der Endzeitpunkt für das nächste Paket berechnet werden muss, würde dies bei vier Ports und vier Paketklassen zur Verwendung von 16 Multiplizierern führen. Werden die Endzeitpunkte für alle Paketklassen eines Ports sequentiell mit einem Multiplizierer berechnet, lassen sich die durch den Queue Arbiter verwendeten Ressourcen signifikant reduzieren. Durch eine entsprechende Auslegung des Rechenwerks zur Berechnung von t_{end} fällt so lediglich eine zusätzliche Latenz von drei Takten bei vier Paketklassen an. Diese Latenz tritt konstant beim Senden jedes Paketes auf und kann somit zum Einhalten der definierten Schedules kompensiert werden. Da die Latenz konstant ist, führt dies zu keinem zusätzlichen Jitter beim Senden von Paketen.

Zur Auswahl und zum Senden des nächsten Paketes wird wie folgt vorgegangen:

1. Speichere alle durch TSAs und Queues an den Queue Arbiter gemeldeten Werte in Registern.
2. Wenn keine Paketklasse ein Paket anmeldet dann gehe zu 1.
3. Initialisiere die Register für das aktuell ausgewählte Paket mit $next\_idx = 0$, $next\_start =$ Maximalwert, $next\_end = 0$, $have\_frame = 0$. Initialisiere das Register für die aktuell zu betrachtende Paketklasse mit $idx = 0$.
4. Berechne den Endzeitpunkt für die Paketklasse idx.
5. Wenn $frame\_end < next\_start$ und *send_request* gesetzt ist dann aktualisiere die Register für das aktuell ausgewählte Paket mit dem aktuell betrachteten.

6. $idx = idx + 1$.
7. Wenn $idx <$ *Anzahl der Paketklassen*, dann gehe zu 4.
8. Wenn *next_start* in der Zukunft ($timestamp < next\_start$) dann gehe zu 1.
9. Signalisiere angeschlossenen TSAs und Queues, wer sendet. Schalte den Demultiplexer für die Weiterleitung des internen Paketheaders entsprechend.
10. Warte, bis *next_end* in der Vergangenheit ist ($next\_end < timestamp$).
11. Versenden des Paketes beendet, gehe zu 1.

Die Rechenoperationen werden im Rahmen des im Folgenden beschriebenen Rechenwerkes durchgeführt. Das Rechenwerk des Queue Arbiters besteht, abgesehen von den Komponenten zum Berechnen von *frame_end*, auch aus Registern zum Speichern der Informationen über das aktuell ausgewählte Paket und zum Vergleichen der Zeitstempel. Das umgesetzte Rechenwerk ist in Abbildung 4.22 dargestellt. Wie ersichtlich, wird zur Berechnung von *frame_end* ein Addierer und ein Multiplizierer verwendet und für sämtliche Vergleichsoperationen ein Komparator. Weiter kann der Ressourcenbedarf des Queue Arbiters nicht reduziert werden. Um die Laufzeiten des Rechenwerks im Rahmen des Kerntaktes zu halten, werden die Zwischenergebnisse der Recheneinheiten in Registern gespeichert.

In Abbildung 4.22 ist an den als Festkommazahl interpretierten Signalen das Format in der Form Q<Vorkommabits>.<Nachkommabits> angemerkt. Das Format des Signals *byte_time* ergibt sich aus benötigtem Maximalwert und Genauigkeit der Berechnung der Paketlänge. Bei einer Datenrate von $100\ \mathrm{Mbit/s}$ dauert die Übertragung eines Bytes $80\ \mathrm{ns}$. Ein Betrieb mit $10\ \mathrm{Mbit/s}$ ist nicht vorgesehen, somit beträgt der maximal zu konfigurierende Wert für *byte_time* $80\ \mathrm{ns}$. Der Sendezeitpunkt eines Paketes kann bei $1000\ \mathrm{Mbit/s}$ Datenrate auf Grund der Synchronisation zwischen den unterschiedlichen Taktraten in den Kernkomponenten und den MACs maximal mit einer Genauigkeit von $8\ \mathrm{ns}$ definiert werden. Dementsprechend sollte der maximal auftretende Fehler bei der Berechnung der Übertragungszeit eines Paketes t_{err} geringer als $8\ \mathrm{ns}$ sein.

Der in die Berechnung der Paketlänge durch die Genauigkeit byte_time$_{prec}$ von *byte_time* induzierte Fehler fließt durch die Multiplikation der Paketlänge *next_len* mit *byte_time* in *frame_time* und *frame_end* ein. Der in die Berechnung von *frame_time* und *frame_end* induzierte Fehler beträgt t_{err} und sollte $8\ \mathrm{ns}$ nicht übersteigen. Die gemeldete Paketlänge ist nicht fehlerbehaftet, der Fehler des gemeldeten Sendezeitpunktes des Paketes *next_frame* wird nicht

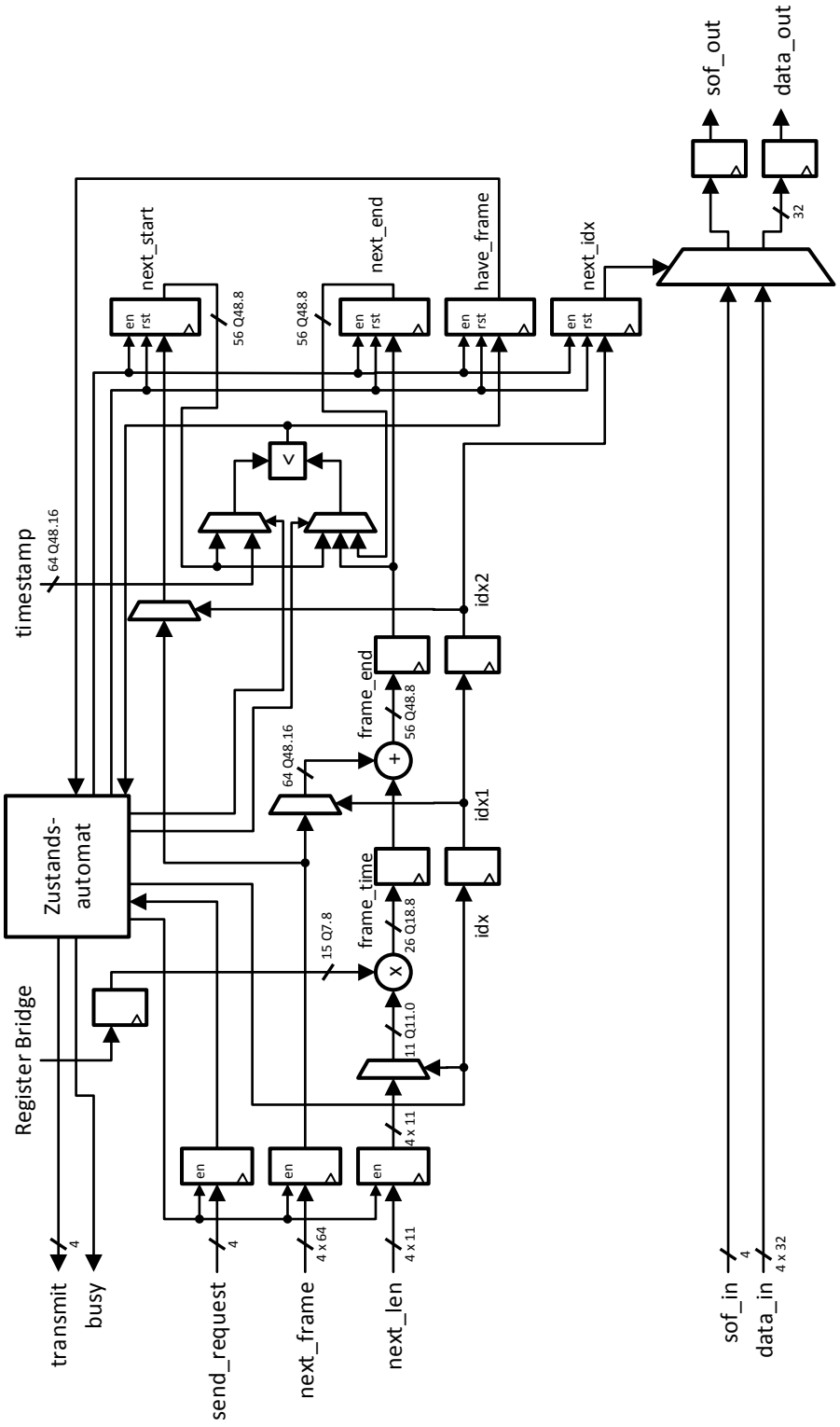


Abbildung 4.22: Rechenwerk des Queue Arbiters

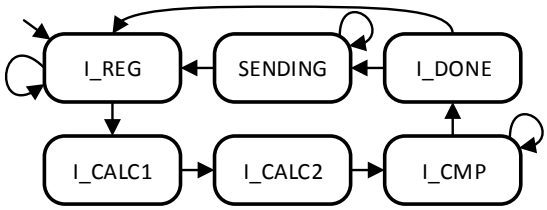


Abbildung 4.23: Mealy-Zustandsautomat zur Steuerung des Rechenwerks im Queue Arbiter

betrachtet. Somit fallen die Fehler bei der Berechnung von *frame_end* wie folgt an, woraus sich die benötigte Genauigkeit byte_time$_{prec}$ ergibt:

$$\begin{aligned}
\text{frame_end} + t_{err} &= \text{next_len} * (\text{byte_time} + \text{byte_time}_{prec}) + \text{next_frame} \\
\text{byte_time}_{prec} &= \frac{t_{err}}{\text{next_len}} \\
\text{byte_time}_{prec} &= \frac{8\ \text{ns}}{1538\ \text{B}} = 5,2\ \text{ps/B} \\
5,2\ \text{ps/B} &\geq 2^{-8}\ \text{ns/B} = 3,9\ \text{ps/B}
\end{aligned}$$

Somit kann mit dem Format Q7.8 für *byte_time* der geforderte Wertebereich mit ausreichender Genauigkeit abgedeckt werden. Für die daraus resultierenden Signale *frame_time* und *frame_end* ergeben sich die Formate aus den durchgeführten Operationen, wobei *next_frame* auf Q48.8 reduziert wird, da somit die geforderte Genauigkeit erreicht werden kann.

Das beschriebene Rechenwerk wird von einem Mealy-Zustandsautomat mit den folgenden Zuständen gesteuert:

- **I_REG:** Eingehende Signale in Registern puffern. Übergang nach **I_CALC1**, wenn ein Bit in *send_request* gesetzt ist.
- **I_CALC1, I_CALC2:** Warten bis Ergebnisse durch das Rechenwerk ausgegeben werden.
- **I_CMP:** Vergleiche Startzeitpunkt des ausgewählte Paketes mit aktuell berechnetem Endzeitpunkt, gegebenenfalls neues Paket auswählen.
- **I_DONE:** Nächstes zu sendendes Paket ausgewählt. Prüfe, ob Sendezeitpunkt in der Zukunft liegt.
- **SENDING:** Veranlasse Senden des ausgewählten Paketes.

Der Zustandsautomat verfügt über die in Abbildung 4.23 dargestellten Zustandsübergänge.

4.4.6 Paketklassen

Im Rahmen dieser Arbeit werden Queues und TSAs für vier Paketklassen instantiiert. Über die höchste Priorität 0 verfügt TT-Ethernet, damit der konfigurierte Schedule korrekt ausgeführt werden kann. Es wird eine AVB Klasse mit Priorität 1 instantiiert sowie ein AFDX Virtual Link mit Priorität 2 für die Zeitsynchronisation von TT-Ethernet. Pakete ohne Echtzeitanforderungen werden über einen Strict Priority TSA mit der geringsten Priorität 3 vermittelt.

4.5 Puffermanagement

Wie in Kapitel 3.7 beschrieben, sind durch das Buffer Management Modul die Pufferslots zu verwalten und der Zugriff auf sie zu abstrahieren. Da eine zu konfigurierende Anzahl von Endpunkten Anfragen an das Buffer Management übergeben, ist für eine Arbitrierung der Anfragen ohne das Verhungern eines Endpunktes zu sorgen (siehe: Starvation in [56, S. 459]). Bei den zu verarbeitenden Anfragen handelt es sich entweder um Zugriffe auf Pufferslots oder um die Verwaltung betreffende Kontrolloperationen. Da beim Zugriff auf die Pufferslots nicht geprüft wird, ob ein Slot alloziert ist, wird das Buffer Management in unabhängige Pfade zur Verarbeitung von Kontroll- und Datenanfragen aufgeteilt. Die Umsetzung des Kontrollpfades wird in Abschnitt 4.5.3 beschrieben, die Umsetzung des Datenpfades in Abschnitt 4.5.4.

Der Kontrollpfad übernimmt die Verwaltung der Pufferslots mit Hilfe von zählenden Referenzen. Zur Verwaltung der zählenden Referenzen stehen die folgenden Operationen bereit:

- **ReqID:** Fordere einen neuen Pufferslot an.
- **IncID:** Inkrementiere den Referenzzähler des Pufferslots mit gegebener ID.
- **FreeID:** Dekrementiere den Referenzzähler des Pufferslots mit gegebener ID. Wenn der Referenzzähler 0 erreicht, ist der Pufferslot wieder frei.

Weiterhin ist durch den Kontrollpfad die Anzahl der noch freien Pufferslots zu überwachen. Unterschreitet die Menge der freien Pufferslots einen konfigurierten Schwellwert, meldet das Buffer Management an die Queues, dass Pakete verworfen und somit Pufferslots freigegeben werden müssen. Für jede Paketklasse wird ein Schwellwert konfiguriert.

Bei Anfragen an den Datenpfads handelt es sich um Lese- oder Schreiboperationen auf einen Pufferslot mit gegebener ID und einer Adresse relativ zum Beginn des Pufferslots. Um eine Fragmentierung zu vermeiden, ist der Speicher in Pufferslots mit einer festen Größe von 2048 B unterteilt. Dementsprechend ergibt sich bei Lese- und Schreiboperationen die Speicheradresse durch Konkatenieren der Puffer-ID und der Adresse relativ zum Beginn des Puf-

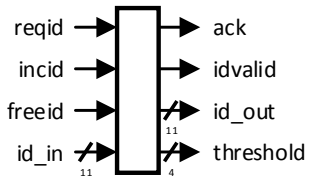


Abbildung 4.24: Interface des Kontrollpfads des Buffer Managements

ferslots. Kontroll- und Datenpfad verfügen über die in Abschnitt 4.5.1 beschriebenen Interfaces.

Die Paketdaten werden im auf dem NetFPGA Board verfügbaren SRAM abgelegt. Es handelt sich um zwei Chips des Unternehmens Micron, die jeweils mit einem $36\ \mathrm{bit}$ breiten Datenbus und einem gemeinsamen, $19\ \mathrm{bit}$ breiten Adressbus angebunden sind. Das SRAM wird, wie in Abschnitt 4.1.4 beschrieben, mit dem im Kern verwendeten Takt von $62,5\ \mathrm{MHz}$ betrieben und verarbeitet in jedem Takt einen unabhängigen Zugriff[5]. Da der Datenpfad für die Paketdaten $64\ \mathrm{bit}$ breit ist, werden $64\ \mathrm{bit}$ des Datenbusses zum SRAM verwendet. Von den verfügbaren $4,5\ \mathrm{MiB}$ werden also $4\ \mathrm{MiB}$ des SRAM verwendet. Somit ist die Bandbreite mit $4\ \mathrm{Gbit/s}$ für vier mit $100\ \mathrm{Mbit/s}$ bei Full Duplex betriebene Ports mit einer Datenrate von insgesamt $800\ \mathrm{Mbit/s}$ ausreichend.

Der Zugriff auf den SRAM wird im NetFPGA Projekt durch den SRAM Arbiter abstrahiert. Er stellt insgesamt vier Endpunkt bereit, zwei Endpunkte wickeln Schreibzugriffe und zwei Endpunkte Lesezugriffe ab. Über jeweils einen Endpunkt zum Schreiben und Lesen wird der Speicher des SRAM dem Host-PC über die PCI-Schnittstelle bereitgestellt. Die übrigen Endpunkte werden für das Buffer Management verwendet. Das Interface des SRAM Arbiters wird in Abschnitt 4.5.2 beschrieben.

4.5.1 Interfaces des Buffer Managements

Im Folgenden werden die Interfaces zum Zugriff auf das Buffer Management beschrieben und anhand von Beispielen verdeutlicht. Wie bereits im vorherigen Abschnitt erläutert, sind die Interfaces des Buffer Managements in einen Kontroll- und einen Datenpfad unterteilt. Es wird erst das Interface des Kontroll- und danach das des Datenpfads beschrieben.

Wie aus Abbildung 4.24 hervorgeht, ist für jede der im vorherigen Abschnitt beschriebenen Operation des Kontrollpfads ein Signal in deren Interface vorgesehen. Jedem Endpunkt steht das dargestellte Interface zur Verfügung. Für die Operation **ReqID** wird *reqid* gesetzt, *incid* zum Inkrementieren eines Referenzzählers (Operation **IncID**) sowie *freeid* zum Freigeben einer Referenz (Operation **FreeID**). Für die Operationen **IncID** und **FreeID** wird in *id_in* die ID

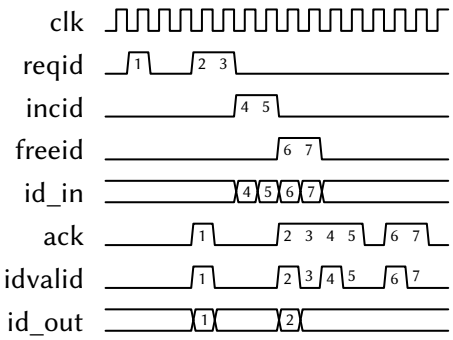


Abbildung 4.25: Beispielhafte Zugriffe auf den Kontrollpfad des Buffer Managements

des betreffenden Paketslots übermittelt. Würden mehrere Signalleitungen gesetzt, würde dies dem Absetzen mehrerer Operationen gleichkommen. Dies ist jedoch nicht vorgesehen. Über das Signal *ack* wird bestätigt, dass eine Operation ausgeführt wurde und das Ergebnis anliegt. Das Signal *idvalid* zeigt bei der Operation **ReqID** an, ob ein Paketpuffer allokiert werden konnte, und bei den Operationen **IncID** und **FreeID**, ob die übermittelte ID für die Anfrage gültig ist. Wurde bei der Operation **ReqID** erfolgreich ein Paketslot allokiert, wird die ID am Signal *id_out* ausgegeben. Wird der 4 MiB große SRAM in Pufferslots mit zu je 2 KiB unterteilt, so sind 2048 Pufferslots verfügbar. Die Pufferslots lassen sich über eine 11 bit breite ID adressieren. Die einzelnen Bits des Signals *threshold* werden entsprechend der konfigurierten Thresholds gesetzt.

Zum Absetzen einer Operation werden die entsprechenden Signale für einen Takt gesetzt. Nach mehreren Takten wird das Ergebnis der Operation für einen Takt ausgegeben. In Abbildung 4.25 ist eine beispielhafte Sequenz von Operationen dargestellt. Die Ziffern an den Signalen ordnen die angelegten Signalwerte einer Operation zu. Bei den abgesetzten Operationen 1, 2 und 3 handelt es sich um **ReqID** Operationen, bei 4 und 5 um **IncID** Operationen und bei 6 und 7 um **FreeID** Operationen. Die abgesetzten Operationen 3, 5 und 7 waren nicht erfolgreich, die verbleibenden waren erfolgreich.

Über den Datenpfad des Buffer Managements werden Lese- und Schreibzugriffe auf die Pufferslots abgewickelt. Das Interface des Datenpfads verfügt für jeden Endpunkt über die in Abbildung 4.26 dargestellten Signale und ist teilweise an das Interface des SRAM Arbiters angelehnt. Um einen Lesezugriff abzusetzen, wird das Signal *read* gesetzt, an *id* die ID des Pufferslots und an *idx* die Adresse relativ zum Beginn des Pufferslots angelegt. Sobald das Signal *ack* durch das Buffer Management gesetzt wird, ist der Lesezugriff erfolgreich abgesetzt worden und wird durch das Buffer Management verarbeitet. Nach mehreren Takten wird durch das Buffer Management an *data_out* das gelesene Datenwort angelegt und *dout_vld* gesetzt.

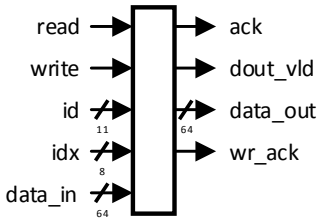


Abbildung 4.26: Interface des Datenpfads des Buffer Managements

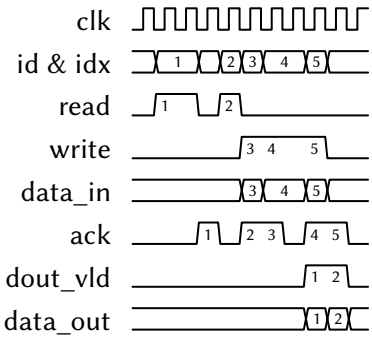


Abbildung 4.27: Beispielhafte Zugriffe auf den Datenpfad des Buffer Managements

Um einen Schreibzugriff anzusetzen, wird das Signal *write* gesetzt, an *id* die ID des Pufferslots, an *idx* die Adresse relativ zum Beginn des Pufferslots sowie das zu schreibende Datenwort an *data_in* angelegt. Wird das Signal *ack* durch das Buffer Management gesetzt, wurde der Schreibzugriff erfolgreich abgesetzt und wird verarbeitet. Entsprechend wird *write* zurückgesetzt. Lese- und Schreibzugriffe auf den Pufferspeicher werden lediglich als Zugriffe auf vollständige 64 bit breite Wörter durchgeführt. Zugriffe werden zur Wortgröße alligned durchgeführt. Dementsprechend werden die 2 KiB eines Pufferslots in 256 Wörter unterteilt. Zur Adressierung aller Wörter eines Pufferslots wird die 8 bit breite Adressleitung *idx* verwendet.

In Abbildung 4.27 sind beispielhafte Zugriffe auf den Speicher durch einen Endpunkt dargestellt. Bei den Zugriffen 1 und 2 handelt es sich um lesende Zugriffe. Zugriff 1 wird nach einem Takt Verzögerung angenommen, Zugriff 2 ohne Verzögerung. Zwischen den Zugriffen 1 und 2 befindet sich eine Pause ohne abgesetzte Zugriffe. Die gelesenen Daten werden direkt hintereinander zurückgegeben und werden durch *dout_vld* signalisiert. Die schreibenden Zugriffe 3, 4 und 5 folgen direkt aufeinander, Zugriff 3 folgt direkt auf die lesenden Zugriffe. Zugriff 4 wird um einen Takt verzögert, die Zugriffe 3 und 5 werden direkt angenommen.

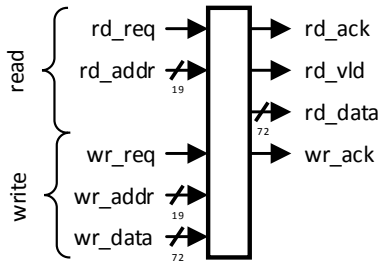


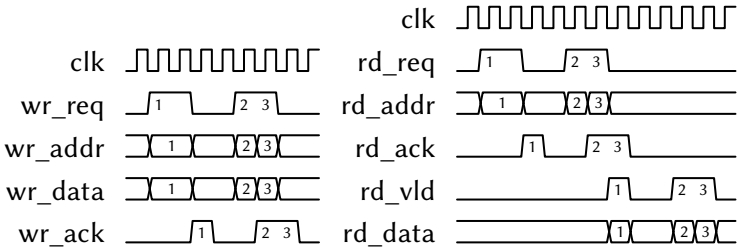
Abbildung 4.28: Interface von je einem lesenden und schreibenden Endpunkt des SRAM Arbiters

4.5.2 Interfaces des SRAM Arbiters

Im Folgenden wird das für den Zugriff auf das SRAM genutzte Interface des SRAM Arbiters beschrieben und anhand von Beispielen verdeutlicht. Durch den SRAM Arbiter werden vom Host-PC und dem Buffer Management abgesetzte Zugriffe auf den SRAM arbitriert. Weiterhin werden durch den SRAM Arbiter die von den SRAM Chips geforderten Details zur Ansteuerung umgesetzt. Für die Zugriffe des Buffer Managements stehen die in Abbildung 4.28 dargestellten Signale zur Verfügung. Die Signale mit dem Präfix *rd_* werden für lesende Zugriffe, Signale mit dem Präfix *wr_* für schreibende Zugriffe verwendet.

Um einen lesenden oder schreibenden Zugriff durchzuführen, wird am entsprechenden Endpunkt das Signal *req* gesetzt und an *addr* die Adresse angelegt. Bei schreibenden Zugriffen werden zusätzlich die zu schreibenden Daten an *data* angelegt. Nach mehreren Takten bestätigt der SRAM Arbiter mit *ack*, dass der Zugriff ausgeführt wird. Dabei muss im selben Takt, in dem *ack* gesetzt wird, der Zugriff zurückgenommen werden oder der nächste Zugriff angelegt werden. Bei Lesezugriffen wird nach vier Takten nach dem Setzen des *ack* Signals das *vld* Signal gesetzt und die gelesenen Daten an *data* angelegt.

In den Abbildungen 4.29 ist jeweils eine beispielhafte Sequenz von lesenden (b) und schreibenden (a) Zugriffen dargestellt. In Abbildung 4.29 (a) sind drei Schreibzugriffe dargestellt. Der erste einzelne Zugriff 1 wird um eine Takt verzögert angenommen. Die beiden folgenden Zugriffe 2 und 3 finden ohne Verzögerung direkt hintereinander statt. In Abbildung 4.29 (b) sind analog zu den Schreib- drei Lesezugriffe dargestellt. Auch hier wird erst ein um einen Takt verzögerter Lesezugriff und danach zwei direkt aufeinander folgende Zugriffe abgesetzt. Wie zu erkennen, werden die gelesenen Worte nach mehreren Takte ausgegeben und dies durch *vld* angezeigt.



(a) Schreibzugriffe (b) Lesezugriffe

Abbildung 4.29: Beispielhafte Zugriffe auf den SRAM-Arbiter

4.5.3 Umsetzung des Kontrollpfads des Buffer Managements

Im Folgenden wird die Umsetzung des Kontrollpfads des Buffer Managements beschrieben. Die Referenzzähler werden im RAM Ref Cnt mit 2048 Wörtern mit jeweils 4 bit Breite abgelegt. Das RAM Ref Cnt verfügt über einen Port zum Lesen und einen Port zum Schreiben. Zur Adressierung wird die ID des betreffenden Pufferslots verwendet. Wird ein Wort im selben Takt gelesen und geschrieben, wird der zu schreibende Wert zurückgegeben. Die IDs freier Pufferslots werden in der FIFO ID Pool gehalten. Sie verfügt über 2048 Einträge mit jeweils 11 bit Breite und wird nach einem Reset durch einen Zustandsautomaten mit einem Zähler initialisiert.

Operationen werden durch die Request Acceptor Logic von jedem Endknoten entgegengenommen und die Art der Operation in 2 bit kodiert zusammen mit der übergebenen ID in der dem Endknoten zugeordneten Req FIFO abgelegt. In jeder Req FIFO können bis zu 16 Anfragen abgelegt werden. Durch den in Abschnitt 4.3.4 beschriebenen RR-Selector wird die als nächstes zu bearbeitende Anfrage ausgewählt und das Lesen aus der betreffenden Req FIFO veranlasst. Die Operationen werden nach der Auswahl durch den RR-Arbiter durch die in Abbildung 4.30 dargestellte, dreistufige Verarbeitungspipeline innerhalb von drei Takten verarbeitet. In den einzelnen Stufen der Verarbeitungspipeline werden die in Tabelle 4.9 beschriebenen Aktionen für die unterschiedlichen Operationen durchgeführt.

4.5.4 Umsetzung des Datenpfads des Buffer Managements

Im Folgenden wird die Umsetzung des Datenpfades des Buffer Managements beschrieben, sie ist in Abbildung 4.31 dargestellt. Entsprechend dem Kontrollpfad werden die Anfragen der Endpunkte durch die Request Acceptor Logic entgegengenommen und in den Endpunkten RA FIFOs (Request Acceptor FIFO) abgelegt. Die Request Acceptor Logic generiert somit die

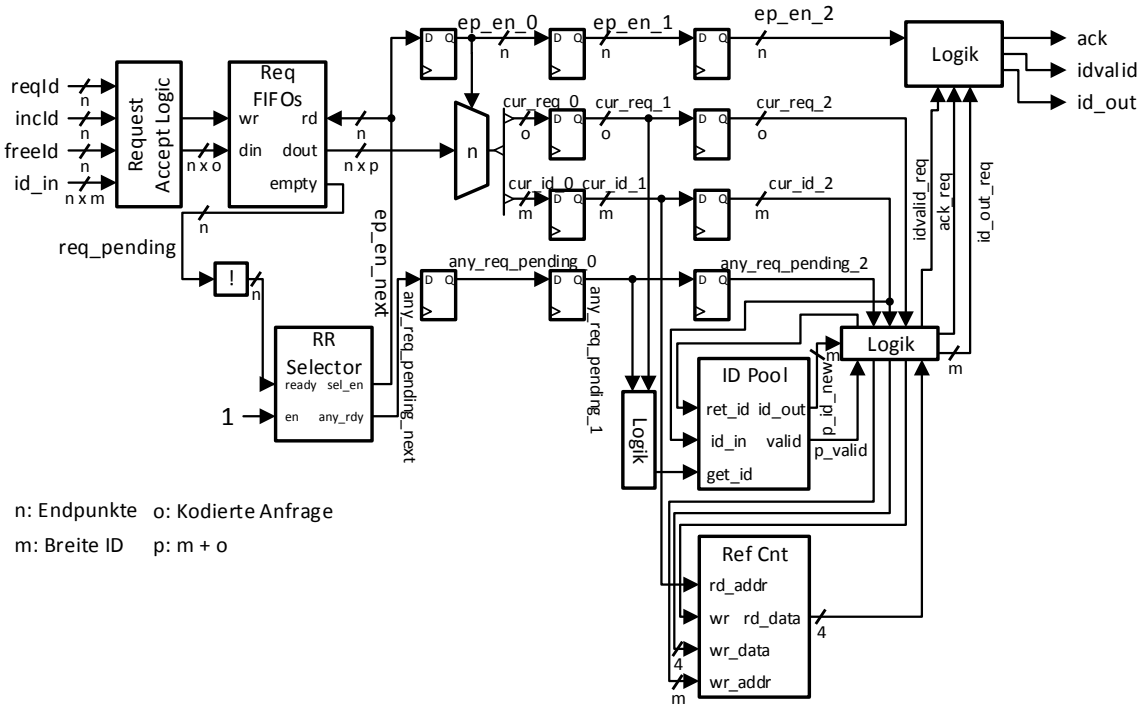


Abbildung 4.30: Verarbeitungspipeline des Buffer Management Kontrollpfads

Stufe	Operation		
	ReqID	**IncID**	**FreeID**
0	Demultiplexing des Requests von der ausgewählten Req FIFO		
1	ID von ID-Pool anfordern	Referenz-Zähler aus Referenzspeicher anfordern	
2	Referenz-Zähler auf 1 setzen	Referenz-Zähler inkrementieren	Referenz-Zähler dekrementieren, wenn Zähler 0, dann frei gewordene ID in ID-Pool ablegen
	Referenz-Zähler speichern, *ack* und *valid* Signale setzen		

Tabelle 4.9: Aktionen in den Pipelinestufen des Kontrollpfads des Buffer Managements bei den unterschiedlichen Operationen

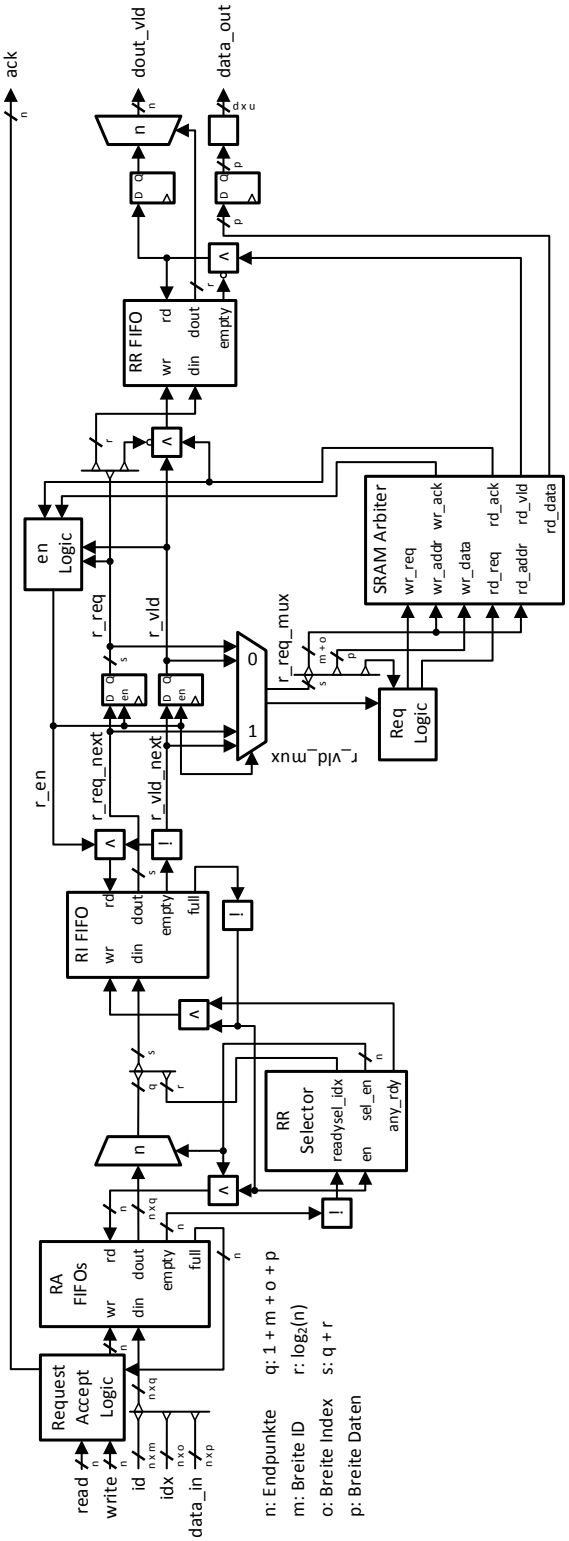


Abbildung 4.31: Verarbeitungspipeline des Datenpfades des Buffer Managements

ack Signale. In den RA FIFOs wird ein read/write Flag zusammen mit den über die Signale *id*, *idx* und *data_in* übergebenen Werten abgelegt. Jede RA FIFO kann bis zu 4 Anfragen puffern. Durch eine Instanz des in Abschnitt 4.3.4 beschriebenen RR-Selectors wird die nächste zu verarbeitende Anfrage ausgewählt, aus der entsprechenden RA FIFO gelesen und zusammen mit dem Index des Endpunktes in der RI FIFO (Request In FIFO) abgelegt. Die RI FIFO kann bis zu 4 Anfragen puffern.

Die Anfragen werden aus der RI FIFO gelesen und an die daran angeschlossene Verarbeitungspipeline übergeben. Die Verarbeitungspipeline übernimmt die Ansteuerung des SRAM Arbiters. Da die Anfragen durch den SRAM Arbiter verzögert werden können, verfügt die Verarbeitungspipeline über eine Flusskontrolle. Da das Ergebnis von Lesezugriffen erst mehrere Takte nach Absetzen der Anfrage durch den SRAM Arbiter zurückgegeben wird, werden für die aktuell bearbeiteten Lesezugriffe die Indizes der Endpunkte in der RR FIFO (Read Request FIFO) abgelegt. In der RR FIFO werden bis zu 16 Lesezugriffe gepuffert. Meldet der SRAM Arbiter das Ergebnis eines Lesezugriffes, wird aus der RR FIFO der Index des betreffenden Endpunktes gelesen und das gelesene Wort an den Endpunkt weitergeleitet.

4.6 Eingebettetes Prozessorsystem

Wie in den Kapiteln 2.5 und 3.8 erläutert, wird in die RT-Bridge ein eingebettetes Prozessorsystem integriert. Es dient zur Konfiguration der Komponenten der RT-Bridge. So sind in vielen Umsetzungen von Komponenten zu konfigurierende Register vorgesehen. Weiterhin sind die PHYs für den Betrieb mit $100\ \mathrm{Mbit/s}$ zu konfigurieren und der Zustand der Ports ist zu überwachen. Das Prozessorsystem soll weiterhin für das Konzept AVB das SRP umsetzen sowie die Zeitsynchronisation auf Basis der PC-Frames von TT-Ethernet durchführen.

Daraus resultiert, dass das Prozessorsystem über Schnittstellen zum Senden und Empfangen von Paketen verfügen muss. Weiterhin wird ein spezialisierter Timer zum Bereitstellen der global synchronisierten Zeitbasis benötigt. Um die erwähnten Protokolle effizient umzusetzen, müssen außerdem Interrupts zu definierten Zeitpunkten auf Basis der synchronisierten Zeitbasis im Prozessor ausgelöst werden können. Weitere Quellen für Interrupts stellen das Empfangen von Paketen sowie das Auslösen von Events durch die Instanzen der CTRL Exec Komponente dar.

Als Prozessor für das Prozessorsystem wird einer der auf dem Chip des FPGA in Hardware vorhandenen PowerPC Prozessorkerne verwendet. Das eingebettete Prozessorsystem wird im Rahmen dieser Arbeit mit Hilfe des im Embedded Development Kit (EDK) enthaltenen Xilinx Platform Studio (XPS) des Unternehmens Xilinx erstellt. Hierfür wurde die Toolchain des

EDK, wie im Rahmen der Ausarbeitung zu Projekt 1 [48] beschrieben, in die Toolchain des NetFPGA Projektes eingebunden. Der PowerPC Prozessor verfügt über jeweils einen Bus zur Anbindung von On-Chip Memory (OCM) an die Daten- und die Instruktionsseite des Prozessors. Peripheriekomponenten lassen sich über den Processor Local Bus (PLB) an den Prozessor anbinden. Um die Komponenten der RT-Bridge über den PLB an den Prozessor anzubinden, wird eine Reihe von Peripheriekomponenten umgesetzt. Die umgesetzten Peripheriekomponenten sowie die daraus resultierende Memory Map des Prozessors werden in Abschnitt 4.6.2 beschrieben. Auf die verwendeten Interfaces wird in Abschnitt 4.6.1 eingegangen.

4.6.1 Interfaces

Im Folgenden wird eine Übersicht über die durch das eingebettete Prozessorsystem verwendeten Interfaces zu den Komponenten der RT-Bridge gegeben. Zur Konfiguration der PHYs wird das in Kapitel 4.1.1 beschriebene MDIO Interface verwendet. Zum Senden und Empfangen von Paketen wird das in Kapitel 4.2.1 beschriebene interne Interface zum Vermitteln von Paketheadern verwendet. Events der CTRL Exec Komponenten werden durch das in Kapitel 4.2.1 beschriebene Interface zum Übermitteln von Paketversandevents an das Prozessorsystem übermittelt.

Für den Zugriff auf den Daten- und Kontrollpfad des Buffer Managements werden die in Kapitel 4.5.1 beschriebenen Interfaces verwendet. Das Interface zur Konfiguration der Thresholdwerte im Buffer Management wird im Folgenden beschrieben. Die Signale des Interfaces zur Konfiguration der Thresholdwerte im Buffer Management verfügen über den Präfix *bm_c_thr_*. Das Interface besteht aus den Signalen *wr*, *idx*, *rd_val* sowie *wr_val*. Über *idx* wird der Thresholdwert ausgewählt, ist *wr* gesetzt, wird er mit *wr_val* zur nächsten positiven Taktflanke aktualisiert. An *rd_val* wird der aktuell ausgewählte Thresholdwert ausgegeben. Dementsprechend wird ein Thresholdwert innerhalb eines Taktes gelesen oder geschrieben.

Die Register Bridge dient zur Anbindung aller Komponenten, in denen lediglich wenige Register verwendet werden. Dementsprechend kann die Anzahl der an den PLB anzubindenden Peripheriekomponenten gering gehalten werden. Weiterhin kann durch die Wahl eines, im Gegensatz zum Interface des PLB, einfachen Interfaces die in den Komponenten der RT-Bridge zum Umsetzen von Registern benötigte Logik minimal gehalten werden.

Jede Komponente, die über die Register Bridge zu konfigurierende Register instantiiert, stellt einen an die Register Bridge angeschlossenen Endpunkt dar. Um die Anzahl der aus dem eingebetteten Prozessorsystem herauszuführenden Signale gering zu halten, werden lediglich die in Abbildung 4.32 dargestellten Signale herausgeführt und durch die Komponente Register Bridge an die Endpunkte in der RT-Bridge verteilt. Im Gegensatz zum Interface zur

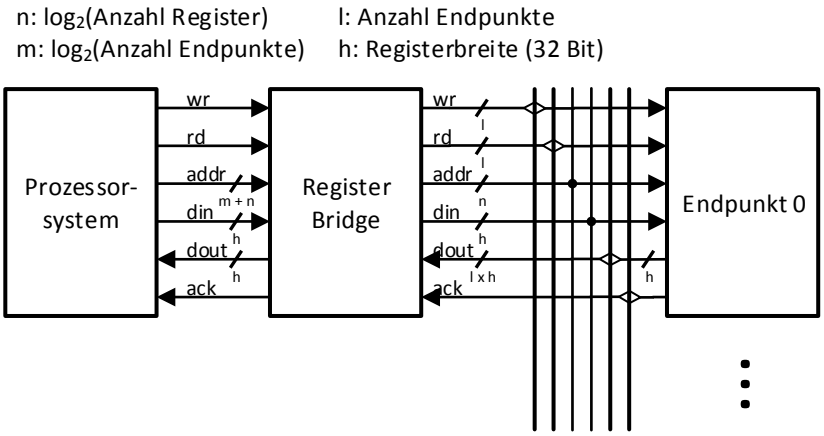


Abbildung 4.32: Signale des Interfaces der Register Bridge

Konfiguration der Thresholdwerte im Buffer Management beschränkt es sich nicht auf einen Takt lange Registerzugriffe. Die Dauer der Registerzugriffe wird durch die Implementierung der Register in den Endpunkten bestimmt.

Wie in Abbildung 4.32 dargestellt, verfügt sowohl das Interface vom eingebetteten Prozessorsystem zur Register Bridge Komponente als auch das Interface von der Register Bridge zu den Endpunkten über die folgenden Signale:

- *rd*, *wr* zeigen einen Lese- oder Schreibzugriff an. Das eingebettete Prozessorsystem setzt diese so lange, bis der Zugriff durch die Endpunkt mit Setzen des Signals *ack* quittiert wird.
- *addr* beinhaltet die Adresse des Registers, auf das zugegriffen werden soll. Die MSBs von *addr* zwischen dem eingebetteten Prozessorsystem und der Register Bridge dienen der Adressierung des Endpunktes.
- *din* beinhaltet den in das Register zu schreibenden Wert.
- *dout* beinhaltet den aus dem Register gelesenen Wert.
- *ack* wird durch den Endpunkt gesetzt, wenn der Zugriff zum Ende des Taktes abgeschlossen ist.

Jeder Endpunkt verfügt maximal über 8 Register, dementsprechend ist *addr* zwischen Register Bridge und den Endpunkten 3 bit breit. Über das Interface werden 32 bit breite Zugriffe durchgeführt, *din* und *dout* verfügen über eine entsprechende Breite. Die in der RT-Bridge instantiierte Komponente Register Bridge beinhaltet durch die MSBs von *addr* gesteuerte Einheiten zum Adressieren der Endpunkte und zum Demultiplexen der Signal von den Endpunkten.

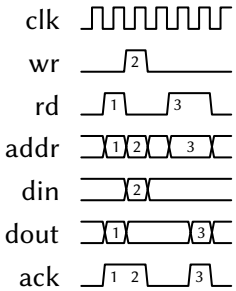


Abbildung 4.33: Beispielhafte Zugriffe über das Interface der Register Bridge

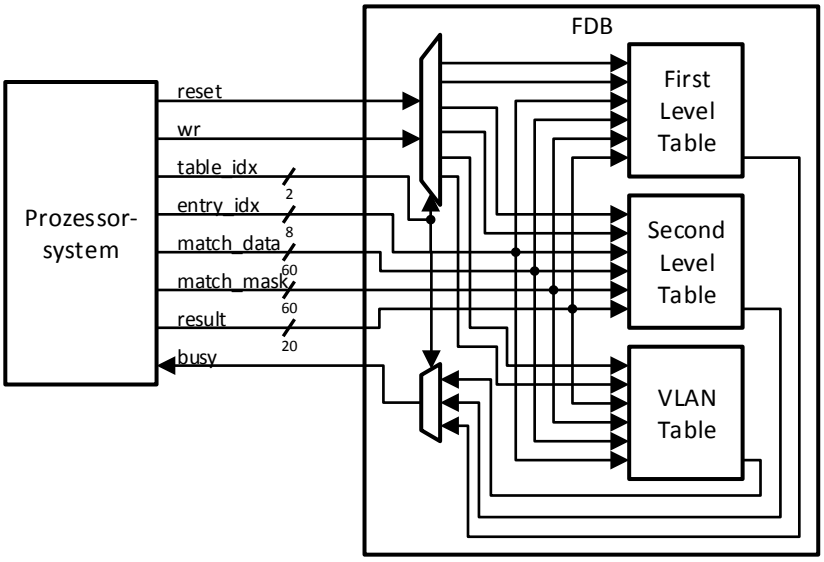


Abbildung 4.34: Signale des Interfaces zur Konfiguration der FDB

Im Folgenden wird das in Abbildung 4.33 dargestellte Beispiel einer Sequenz von Zugriffen über die Register Bridge beschrieben. Bei den Zugriffen bei Ziffer 1 und 2 handelt es sich um jeweils einen Takt lange Lese- und Schreibzugriffe. Darauf folgt nach einem ungenutzten Takt bei 3 ein durch den Endpunkt auf zwei Takte gestreckter Lesezugriff.

Die Konfiguration der Filtering Database durch das eingebettete Prozessorsystem wird über das im Folgenden beschriebene Interface durchgeführt. Es verfügt über die in Abbildung 4.34 dargestellten Signale. Um einen Eintrag einer der Lookup Tables First Level Table, Second Level Table oder VLAN Table der FDB zu aktualisieren, werden die Signale wie folgt verwendet: An das Signal *table_idx* wird der Index der Tabelle, auf die der Zugriff stattfinden soll, angelegt. An *entry_idx* wird der Index des zu schreibenden Eintrags in der Lookup Table angelegt. Der in der Lookup Table abzulegende Eintrag wird an *match_data*, *match_mask* und *result*

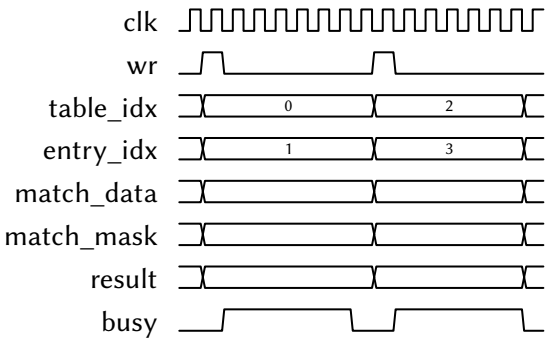


Abbildung 4.35: Beispielhafte Schreibzugriffe auf die FDB

angelegt. Sind die Einträge einer Lookup Table nicht so breit wie die durch dieses Interface bereitgestellten Daten, werden die LSB verwendet. Um den Schreibzugriff zu beginnen, wird *wr* für einen Takt gesetzt. Während der Schreibzugriff verarbeitet wird, setzt die Lookup Table das Signal *busy*. Weiterhin kann der Inhalt einer Lookup Table durch Auswahl der Lookup Table durch *table_idx* und setzen von *reset* zurückgesetzt werden. Durch das Signal *table_idx* werden dementsprechend die Signale *wr*, *busy* und *reset* der Lookup Tables selektiert.

Im Folgenden wird die in Abbildung 4.35 dargestellte beispielhafte Sequenz von Schreibzugriffen auf die FDB beschrieben. Bei dem ersten Zugriff wird auf den Eintrag mit dem Index 1 in der First Level Table geschrieben. Der nächste Zugriff wird erst nach Ende des ersten Zugriffs mit Zurücknehmen von *busy* begonnen. Dabei handelt es sich um einen Zugriff auf den Eintrag mit dem Index 3 in der VLAN Table.

4.6.2 Peripheriekomponenten des eingebetteten Prozessorsystems

Im Folgenden werden die einzelnen Peripheriekomponenten und das sich daraus ergebende eingebettet Prozessorsystem beschrieben. Im Rahmen dieses Abschnittes wird nicht auf die Registerinterfaces der einzelnen Peripheriekomponenten eingegangen. In der Ordnerstruktur jeder umgesetzten Peripheriekomponente ist in der Datei „functional description.txt“eine Beschreibung des Registerinterfaces enthalten. Am Ende dieses Abschnittes wird weiterhin die definierte Memory Map dargestellt. Das Prozessorsystem ist, wie in Abbildung 4.36 dargestellt, mit Peripheriekomponenten und Speicher ausgestattet. Der PowerPC Prozessorkern wird mit 250 MHz betrieben, der an den OCM Schnittstellen angeschlossene Speicher mit 125 MHz. Alle Peripheriekomponenten werden an den mit 62, 5 MHz betriebenen PLB angeschlossen. Sämtliche Takte werden von Kerntakt vom 62, 5 MHz mit Hilfe von DCM abgeleitet, die an

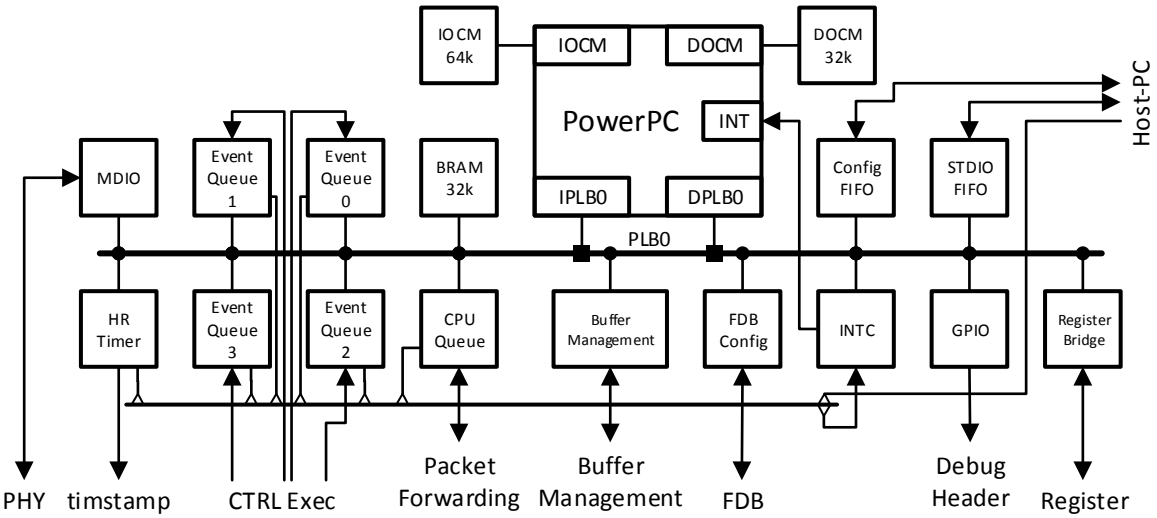


Abbildung 4.36: Eingebettetes Prozessorsystem der RT-Bridge

den PLB angeschlossenen Peripheriekomponenten arbeiten taktsynchron mit den Komponenten der RT-Bridge.

An die OCM Schnittstellen werden 64 KiB Instruktionsspeicher und 32 KiB Datenspeicher angeschlossen. An den PLB sind weitere 32 KiB geteilter Daten- und Instruktionsspeicher angeschlossen. Im OCM Instruktionsspeicher wird die durch den Prozessor auszuführende Firmware abgelegt. Die compilierte Firmware wird als Binary (ELF-File) in das durch die Toolchain generierte Konfigurationsfile (BIT-File) für den FPGA geladen. Somit werden die Speicher des Prozessors während der Konfiguration des FPGAs initialisiert. Um bei der Entwicklung der Firmware eine Schnittstelle zum Debugging zur Verfügung zu haben, können STDIO Ausgaben des Prozessorsystems über eine FIFO an den Host-PC übergeben werden. Weiterhin kann der Host-PC über ein Registerinterface die STDIO FIFO lesen, schreiben sowie Resets und Interrupts auf dem Prozessorsystem auslösen. Die STDIO FIFO besteht aus jeweils einer 32 bit breiten und 512 Wörter tiefen FIFO vom Prozessorsystem zum Host-PC und vom Host-PC zum Prozessorsystem. Das Prozessorsystem ist über die weitere Config FIFO an den Host-PC angebunden. Die Config FIFO ist zum Einspielen von Konfigurationsdaten in die RT-Bridge vorgesehen.

Aus den Bibliotheken des XPS wird der XPS Interrupt Controller in der Version 1.00.a sowie die XPS General Purpose Input/Output (GPIO) Schnittstelle in der Version 1.00.a verwendet. Sämtliche Interruptleitungen des Prozessorsystems werden durch den Interrupt Controller gebündelt und an den Prozessor weitergeleitet. Die GPIO Schnittstelle wird für Debugging- und Evaluationszwecke verwendet.

Über die Peripheriekomponente Register Bridge befinden sich sämtliche Register der an sie angeschlossenen Endpunkte im direkt adressierbaren Speicherbereich des Prozessors. Die Register sind ab dem Offset der Register Bridge Peripheriekomponente an zu $4\,\mathrm{B}$ aligned Speicheradressen verfügbar. Die Adresse relativ zur Offset-Adresse der Register Bridge Peripheriekomponente eines Registers r des Endpunktes e ergibt sich aus $(r + e * 8) * 4$.

Eingehende Paketheader werden in der CPU Queue in einer FIFO abgelegt. In der FIFO können bis zu 51 interne Paketheader gepuffert werden. Trifft ein Paketheader ein, wird er in der FIFO abgelegt und ein Interrupt wird ausgelöst. Wird durch das Prozessorsystem das Auslesen des Paketheaders aus der FIFO angestoßen, wird der Paketheader aus der FIFO gelesen und die Felder werden in Registern abgelegt. Die einzelnen Felder des Paketheaders können entsprechend durch das Prozessorsystems aus den Registern ausgelesen werden. Soll ein Paket an die RT-Bridge übergeben werden, werden durch das Prozessorsystem die einzelnen Felder in Registern konfiguriert und die Ausgabe des internen Paketheaders angestoßen. Daraufhin generiert die CPU Queue aus den konfigurierten Feldern einen Paketheader und leitet ihn an den Input Port Multiplexer weiter. Weiterhin wird für ausgehende Pakete nach dem Generieren eines Paketes die für einen Port eindeutige PaketID inkrementiert.

Die durch die CTRL Exec Komponenten ausgelösten Events werden jeweils in einer Instanz der Event Queue abgelegt. Wie die CPU Queue enthält eine Instanz der Event Queue eine FIFO zum Puffern der Events. In der FIFO können bis zu 102 Events gepuffert werden. Trifft ein neues Event ein, wird ein Interrupt ausgelöst und das Prozessorsystem stößt das Auslesen des Events aus der FIFO an. Die Felder des Events werden wie bei der CPU Queue in Registern abgelegt und können dann vom Prozessorsystem ausgewertet werden. Wie erläutert, wird eine Event Queue Instanz einer Instanz der CTRL Exec Komponente eines Ports zugeordnet. Somit verfügt das Prozessorsystem über vier Instanzen der Event Queue.

Zum Schreiben von Einträgen in die Lookup Tables der FDB werden die Daten des zu schreibenden Eintrags in Register der FDB Config Peripheriekomponente geschrieben und in weiteren Registern die Index des zu schreibenden Eintrags und der Lookup Table konfiguriert. Über ein Kontrollregister wird der Schreibvorgang angestoßen und der Status des *busy* Signals über ein Statusregister geprüft.

Die Buffer Management Peripheriekomponente stellt zwei Speicherbereiche zur Verfügung. Die Pufferspeicher werden vollständig als Speicherbereich im Adressbereich des Prozessors für direkte Zugriffe zur Verfügung gestellt. Die Startadresse eines Paketslots mit der ID i relativ zum Offset des Data Speicherbereichs des Buffer Managements ergibt sich dabei aus $i * 2048$. Der zweite Speicherbereich beinhaltet Register zum Zugriff auf den Kontrollpfad des

Buffer Managements sowie zur Konfiguration der Thresholdwerte. Weiterhin ist die Anzahl der aktuell freien Pufferslots in einem Register verfügbar.

Die Komponente High Resolution Timer (HR Timer) dient zum Bereitstellen einer global synchronisierten Zeitbasis. Es handelt sich um einen 64 bit breiten Zähler, der in jedem Takt um einen konfigurierten Wert inkrementiert wird. Der Wert des Zählers entspricht dem in Kapitel 4.4.1 beschriebenen Format und wird somit als Festkommazahl der Einheit Nanosekunde mit 16 Nachkommabit interpretiert. Somit wird der Oszillator für die Generierung des Kerntaktes als Zeitbasis verwendet. Während des Betriebes kommt es zu Unregelmäßigkeiten des Schwingverhaltens des Oszillators. Die Unregelmäßigkeiten werden mit Hilfe der PC-Frames gemessen und durch kontinuierliche Anpassung des Inkrements des Zählers ausgeglichen. Zum Auslösen von Interrupts zu definierten Zeitpunkten verfügt der HR Timer weiterhin über ein *match* Register. Überschreitet der Wert des Zählers den im *match* Register konfigurierten Wert, wird ein Interrupt ausgelöst. Weiterhin wird bei Überläufen des Zählers ein Interrupt ausgelöst.

Über die Peripheriekomponente MDIO lassen sich Zugriffe auf die Register der PHYs über das MDIO Interface absetzen. Die Felder der MDIO Zugriffe werden durch den Prozessor in Registern geschrieben und der Zugriff über das *control* Register ausgelöst. Aus allen Peripheriekomponenten des eingebetteten Prozessorsystems ergibt sich aus Sicht des Prozessors die in Tabelle 4.10 definierte Memory Map.

4.7 Software

Im Folgenden wird die auf dem eingebetteten Prozessorsystem betriebene Software beschrieben. Da diese Software integraler Bestandteil der RT-Bridge ist, wird sie im Rahmen dieser Arbeit als Firmware bezeichnet. Für Tests einzelner Hardwaremodule wurden eigene Testfirmwares entwickelt, auf diese wird unter anderem in Kapitel 5 eingegangen. In diesem Kapitel werden die grundlegend zum Betrieb der RT-Bridge benötigten Softwaremodule beschrieben. Ziel ist es, die in Kapitel 6 beschriebene Evaluation durchzuführen.

Im Folgenden wird erst auf die benötigte Funktionalität eingegangen. Dann wird die Verwendung des Speichers durch die Firmware beschrieben und danach die wesentlichen Softwaremodule. Die folgende Funktionalität wird zur Evaluation benötigt:

- Konfiguration aller Komponenten der RT-Bridge.
- Zeitsynchronisation anhand von PC-Frames.
- Zuspielen eines TT Schedules an TT TSAs und Queues.

Peripherie	Bus	Startadresse	Länge
PLB BlockRAM	PLB0	00000000_{16}	8000_{16}
DOCM BlockRAM	DOCM	$40C08000_{16}$	8000_{16}
XPS Interrupt Controller	PLB0	80000000_{16}	10000_{16}
XPS GPIO	PLB0	84000000_{16}	10000_{16}
FDB Config	PLB0	86000000_{16}	10000_{16}
STDIO FIFO	PLB0	88000000_{16}	10000_{16}
Config FIFO	PLB0	$8A000000_{16}$	10000_{16}
Register Bridge	PLB0	$8C000000_{16}$	10000_{16}
Buffer Management Kontrollpfad	PLB0	$8E000000_{16}$	10000_{16}
Buffer Management Datenpfad	PLB0	90000000_{16}	400000_{16}
CPU Queue	PLB0	92000000_{16}	10000_{16}
Event Queue Port 0	PLB0	94000000_{16}	10000_{16}
Event Queue Port 1	PLB0	96000000_{16}	10000_{16}
Event Queue Port 2	PLB0	98000000_{16}	10000_{16}
Event Queue Port 3	PLB0	$9A000000_{16}$	10000_{16}
MDIO	PLB0	$9C000000_{16}$	10000_{16}
HR Timer	PLB0	$9E000000_{16}$	10000_{16}
IOCM BlockRAM	IOCM	$FFFF0000_{16}$	10000_{16}

Tabelle 4.10: Memory Map des eingebetteten Prozessorsystems

Die ausführbaren Sektionen der Firmware werden im, am IOCM Interface angeschlossenen RAM abgelegt. Im langsameren, an den PLB angeschlossenen RAM werden der Exception Vector des PowerPC sowie der Heap abgelegt. Die übrigen Sektionen befinden sich im, am DOCM Interface angeschlossenen RAM. Das Instruktionscache wird für den am IOCM Interface und PLB angeschlossenen RAM aktiviert, das Datencache für den am DOCM Interface und PLB angeschlossenen RAM.

Low-Level Treibermodule stellen Makros für den Zugriff auf die Register der Peripherie bereit. High-Level Treibermodule beinhalten für die Peripherie der RT-Bridge Funktionen zum Zugriff auf die Funktionalität der einzelnen Peripheriekomponenten des Prozessorsystems. Die Treibermodule wurden implementiert, auf sie wird hier allerdings nicht weiter eingegangen. Im Folgenden wird auf das Timer Modul in Kapitel 4.7.1, die Umsetzung eines Clients zur Zeitsynchronisation in Kapitel 4.7.2 sowie das Modul zur laufenden Versorgung der TT TSAs und Queues mit dem Schedule der TT Nachrichten in Kapitel 4.7.3 eingegangen. In zukünftigen Arbeiten ist eine Umsetzung der für AVB benötigten Protokolle, das Laden der Konfiguration über den Host-PC in die RT-Bridge sowie die Umsetzung weiterer Protokolle oder Funktionalitäten denkbar.

4.7.1 Timer

Das auf dem HR Timer basierende Timer Modul erlaubt es, mehrere softwarebasierte Timer zu verwenden. Beim Einrichten eines Timers wird entweder absolut oder relativ zur aktuellen Zeit ein Zeitpunkt angegeben, an dem die mitgegebene Callbackfunktion aufgerufen werden soll. Weiterhin kann eine Periode, mit der der Timer periodisch neu gestartet werden soll, übergeben werden.

Intern werden die registrierten Timer in einer Liste gehalten. In das Match Register des HR Timers wird immer der absolute Zeitpunkt des als nächstes auslösenden Timers geschrieben. Wird durch den HR Timer ein Match-Interrupt ausgelöst, wird das Callback des entsprechenden Timers aufgerufen. Handelt es sich bei dem ausgelösten Timer um einen einmalig aufzurufenden Timer, wird er aus der Liste der registrierten Timer entfernt. Soll er periodisch aufgerufen werden, wird der nächste Zeitpunkt des Auslösens berechnet. Danach wird der Zeitpunkt des als nächstes auszulösenden Timers in das Match Register des HR Timers geschrieben. Nach dem Aufrufen der Callabackfunktionen ist weiterhin zu prüfen, ob ein Timer verpasst wurde. Ist dies der Fall, wird die Callabackfunktion nachträglich ausgeführt.

Zur vollständigen Abstraktion des HR Timers stellt das Timer Modul Funktionen zur Anpassung des Increment Registers bereit und erlaubt das Registrieren von bei Änderungen des Increment Registers aufgerufenen Callbackfunktionen. Um bei der Zeitsynchronisation initial den bestehenden Offset zu korrigieren, verfügt das Timer Modul über Funktionen zur Offsetkorrektur. Der aktuelle Offset wird in einer Variable gehalten. Wird auf die Register des HR Timers zugegriffen, müssen die Registerwerte um den aktuellen Offset korrigiert werden. Wird der Offset aktualisiert, muss berücksichtigt werden, dass registrierte Timer sich nach der Korrektur gegebenenfalls in der Vergangenheit befinden. Ist dies der Fall, werden deren Callbackfunktionen nachträglich aufgerufen. Dementsprechend verhalten sich die Timer gegenüber von Korrekturen des Offsets für nicht beteiligte Softwarekomponenten transparent. Jedoch sind Zeitstempel, die durch Komponenten der RT-Bridge generiert wurden, um den aktuellen Offset zu korrigieren.

4.7.2 Sync Client

Das Modul Sync Client setzt einen Client zur Synchronisation anhand von PC-Frames um. Es ist an die in [40] beschriebene Umsetzung eines Cients zur Synchrinisation angelehnt. Im Folgenden wird erst die Funktionsweise der Zeitsynchronisation auf Basis von PC-Frames erläutert und danach die Umsetzung beschrieben.

Bei TTE werden die PC-Frames zu definierten Zeitpunkten im Schedule der zu vermittelnden TT Nachrichten durch den Synchronization Master versendet. Somit ist allen, an TTE beteiligten Netzwerkknoten der Zeitpunkt sowie die Periode des Sendens der PC-Frames bekannt. Die PC-Frames werden über einen virtuellen Link von AFDX vermittelt. Die Dauer der Übertragung ist somit nicht deterministisch im Voraus bekannt und wird von den an der Vermittlung beteiligten Netzwerkknoten im Feld *transparentClock* akkumuliert. Also ergibt sich der Empfangszeitpunkt des PC-Frames am Sync Client in der Zeitbasis des Sync Masters aus dem bekannten Sendezeitpunkt und dem in Feld *transparentClock* akkumulierten Wert.

Durch den Sync Client wird beim Empfang der PC-Frames mit der lokal verfügbaren Uhr ein Zeitstempel erstellt. Anhand von zwei aufeinander folgenden PC-Frames kann die lokal gemessene Periode mit der vorgegebenen Periode der PC-Frames verglichen werden und die Ungenauigkeit der Taktquelle ausgeglichen werden. Weiterhin kann der Offset der Uhren des Sync Masters und Clients anhand der lokal gemessenen und vorgegebenen Empfangszeitpunkte der PC-Frames korrigiert werden.

Die Korrektur des Increment Registers ergibt sich wie folgt aus der Differenz der vorgegebenen und gemessenen Periode. Die gemessene Periode p_m weicht um den Fehler p_e von der vorgegebenen Periode p_t des Sync Masters ab. Somit gilt $p_t = p_m + p_e$. Die Anzahl der während einer Periode vergangenen Takte c_i ergibt sich aus dem alten Increment i_o und der gemessenen Periode p_m. Der alte Increment ist mit dem Fehler i_e behaftet, der neue Increment i_n ist zu berechnen. Zwischen den Perioden, den vergangenen Takten und den Increments besteht folgender Zusammenhang:

$$
\begin{aligned}
p_t &= i_n * c_i \\
p_m &= i_o * c_i \\
p_e &= i_e * c_i
\end{aligned}
$$

Daher gilt auch:

$$
\begin{aligned}
\frac{i_n}{c_i} &= \frac{i_o}{c_i} + \frac{i_e}{c_i} \\
i_n &= i_o + i_e \\
i_e &= i_n - i_o = \frac{p_t}{c_i} - i_o \\
c_i &= \frac{p_m}{i_o}
\end{aligned}
$$

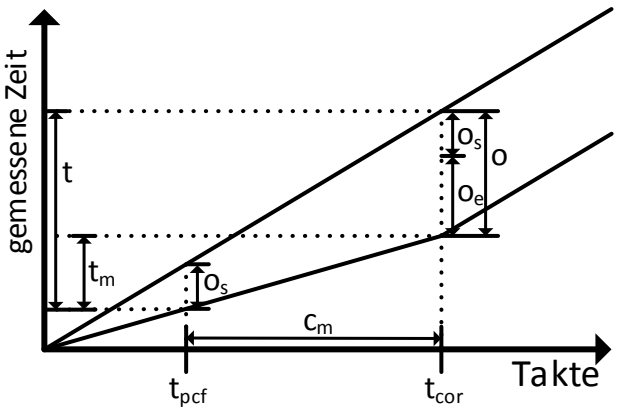


Abbildung 4.37: Zu korrigierende Offsets bei der Zeitsynchronisation

Als Ausgangsgrößen der Berechnung sind p_m, p_t sowie i_o bekannt. Durch Ersetzen des Fehlers i_e ergibt sich somit der korrigierte Increment:

$$
\begin{aligned}
i_n &= i_o + i_e \\
i_n &= i_o + \frac{p_t}{c_i} - i_o \\
i_n &= i_o + \frac{p_t}{p_m / i_o} - i_o \\
i_n &= \frac{p_t * i_o}{p_m}
\end{aligned}
$$

Der so errechnete neue Increment wird jeweils nach dem Erhalt eines neuen PC-Frames errechnet und an das Timer Modul übergeben.

Der aktuelle Offset der lokalen Uhr ergibt sich zur Ankunftszeit des PC-Frames aus der Differenz der errechneten und gemessenen Ankunftszeit des PC-Frames. Bis der Increment korrigiert werden kann, vergeht jedoch auf Grund der durchzuführenden Berechnungen Zeit. Während dieser Zeit ändert sich der Offset mit dem Fehler des Increments i_e. Dies ist bei der Korrektur des Offsets zu berücksichtigen. In Abbildung 4.37 ist der Vorgang der Korrektur des Increments dargestellt. Es wird die gemessene Zeit (Y-Achse) in Abhängigkeit von den verstrichenen Takten (X-Achse) in Form einer Ist- und einer Soll-Funktion dargestellt.

Zum Zeitpunkt t_{pcf} trifft das PC-Frame ein und der Offset o_s ist bekannt. Über die Zeit t_m bis zur Korrektur des Increments bei t_{cor} ergibt sich über c_m Takte mit dem Fehler des Increments i_e das zusätzliche Offset o_e. Wird zu t_{cor} ein Zeitstempel genommen, kann t_m ermittelt werden. Der zu korrigierende Offset o ergibt sich somit aus o_s und o_e. Mit einem

korrekten Increment wäre während t_m die Zeit t verstrichen. Dementsprechend ergibt sich folgender Zusammenhang:

$$o = o_s + o_e$$
$$t = t_m + o_s$$
$$t_m = i_o * c_m$$

Während t verstreicht dabei die selbe Zahl von Takten, somit muss gelten:

$$t = i_n * c_m$$

Aus den bekannten Variablen lässt sich o_e herleiten:

$$o_e = t - t_m$$
$$o_e = i_n * c_m - t_m$$
$$o_e = \frac{i_n * t_m}{i_o} - t_m$$
$$o_e = t_m * (\frac{i_n}{i_o} - 1)$$

Das errechnete Offset wird dann entsprechend im Timer korrigiert.

Die hergeleiteten Berechnungen werden periodisch durch einen registrierten Timer durchgeführt. Mit einer direkten Korrektur des Offsets ist ein Zeitsprung verbunden. Somit wird durch die Abarbeitung der ausstehenden Timer Events gegebenenfalls eine Flut von Events generiert, was zu vermeiden ist. Dementsprechend wird der Offset lediglich zum initialen Erreichen eines synchronisierten Zustandes durch einen Zeitsprung korrigiert. Fällt der gemessene Offset unter 16 µs, wird die Synchronisation als stabil betrachtet. Offset und Periodenlänge werden dann nur noch über den Increment des HR Timers korrigiert.

4.7.3 TT Schedule

Das Modul TT Schedule wird zur Konfiguration und zur laufenden Versorgung der TT TSAs und Queues mit dem Schedule der TT Nachrichten verwendet. Der TT Schedule wird über eine Struktur konfiguriert. Sie enthält eine Beschreibung jeder zu vermittelnden TT-Nachricht. Der definierte Schedule wiederholt sich mit der für die PC-Frames im Sync Client definierten Periode. Für jede Nachricht werden die folgenden Informationen abgelegt:

- Zieladresse

- Paketlänge
- Quell- und Zielport
- Sende- und Empfangszeitfenster als Offset zum Beginn der verwendeten Periode

Während der initialen Konfiguration der RT-Bridge wird aus dem abgelegten Schedule für die Packet Buffering Komponente jedes Ports eine Liste mit Scheduleeinträgen für eingehende und ausgehende Pakete erstellt. Das TT Schedule Modul registriert einen periodischen Timer, um die in den Listen abgelegten Scheduleinformationen laufend in die TT TSAs und Queues einzuspielen.

5 Qualitätssicherung

Im Rahmen dieses Kapitels werden die durchgeführten Maßnahmen zur Sicherung der Qualität der entwickelten Komponenten erläutert. Hierzu wird erst das Vorgehen bei der Umsetzung der Komponenten in Abschnitt 5.1 beschrieben und danach werden in Abschnitt 5.2 beispielhaft die durchgeführten Maßnahmen bei der Umsetzung einer Komponente dargestellt.

5.1 Vorgehen zur Sicherung der Qualität

Laut Hoffmann in [8] lässt sich die Qualität von Software anhand unterschiedlicher Merkmale bewerten. Die Qualität einer Software lässt sich beispielsweise anhand der folgenden Merkmalen beurteilen:

- Funktionalität: Erfüllt die Software die gewünschte Funktionalität? Sind Fehler vorhanden?
- Laufzeit: Erfüllt die Software in einem vorgegebenen Zeitraum die zu erledigenden Aufgaben?
- Zuverlässigkeit: Wie oft treten Fehler auf?
- Wartbarkeit: Wie groß ist der Aufwand, wenn Korrekturen oder Änderungen vorgenommen werden müssen?
- Testbarkeit: Wie gut können Fehler erkannt werden?

Im Rahmen dieser Arbeit wurde davon ausgegangen, dass die Qualität der entwickelten Hardwarekomponenten insbesondere bei der Verwendung von Hardwarebeschreibungssprachen (HDL) wie VHDL mit ähnlichen Merkmalen zu bewerten ist. Die durchgeführten Maßnahmen zur Qualitätssicherung zielen besonders auf gute Wartbarkeit und die Erfüllung der geforderten Funktionalität durch die umgesetzten Komponenten ab. Eine gute Wartbarkeit ist für die weitere Verwendung der umgesetzten Komponenten von Wichtigkeit, das Erreichen der in Kapitel 2.5 gesetzten Ziele geht mit der Erfüllung der geforderten Funktionalität einher.

Um die Wartbarkeit der umgesetzten Komponenten sicherzustellen, wurde vor der Umsetzung der einzelnen Komponenten eine klare Definition des Funktionsumfangs und der zu ver-

wendenden Interfaces erstellt. Weiterhin sind die Komponenten im Quellcode dokumentiert. Um die Erfüllung der Funktionalität durch die umgesetzten Komponenten sicherzustellen, wurden sie Tests unterzogen. Dabei wurden die umgesetzten Komponenten jeweils zu einem möglichst frühen Zeitpunkt getestet. Das Testen einzelner Komponenten zu einem frühen Zeitpunkt in der Entwicklung weist mehrere Vorteile auf. Dadurch ist insgesamt ein geringerer Anteil des potentiell fehlerbehafteten Funktionsumfangs umgesetzt worden. Es ist zu vermuten, dass die zu testende Komponente weniger, sich gegenseitig beeinflussende Fehler enthält und das Fehlerbild entsprechend klarer ist. Wird weiterhin ein Fehler gefunden, müssen bei der Behebung des Fehlers weniger fertiggestellte Teile der Umsetzung auf die Auswirkung des Fehlers beziehungsweise dessen Fehlerbehebung getestet werden.

Die Tests werden in mehreren Stufen durchgeführt. In der ersten Stufe werden lediglich die einzelnen Komponenten getestet. Die zweite Stufe befasst sich mit dem Testen eines vollständigen Moduls. Im Rahmen der dritten Stufe werden die fertiggestellten Komponenten im vorgesehenen Rahmen des User Data Path getestet. Während die ersten drei Stufen Tests mit Hilfe des Simulationstools ModelSim beinhalten, werden die Komponenten in der vierten Stufe mit Hilfe der Hardware getestet. Das Simulationstool des Unternehmens Mentor Graphics ist auf die Simulation von in HDL beschriebenen Komponenten spezialisiert.

Die Tests der ersten Stufe beginnen nach der Fertigstellung einer einzelnen Komponente. Für die Tests der einzelnen Komponenten wurden einfache Scripte, Testbenches oder manuell generierte Stimulis verwendet. Wenn die Tests der einzelnen Komponenten erfolgreich abgeschlossen werden, wird mit den Tests der zweiten Stufe begonnen. Weitere Vorbedingung für die zweite Stufe der Tests ist, dass signifikante Teile eines Moduls fertiggestellt wurden. Zum Durchführen der zweiten Teststufe wird ein Modul in einer Testbench instantiiert. Durch diese Tests kann sichergestellt werden, dass die zusammengesetzten Komponenten als Modul korrekt funktionieren. Außerdem kann die korrekte Kommunikation der Komponenten über die verwendeten Interfaces sichergestellt werden. Verwendet eine Komponente ein Interface zu einer Komponente außerhalb des Moduls, generiert die Testbench entweder die benötigten Stimuli direkt oder instantiiert die über das entsprechende Interface angesprochenen Komponenten wie beispielsweise den SRAM-Arbiter.

Wenn die vorherigen Tests erfolgreich durchgeführt wurden, werden alle fertiggestellten Komponenten im Rahmen des User Data Path in einer Testbench instantiiert. Hierfür wurden anfangs Dummys als Ersatz für noch nicht implementiere Komponenten erstellt und verwendet. Somit kann eine korrektes Funktionieren des vollständigen Systems möglichst früh geprüft werden. Die Testbench für den User Data Path instantiiert zu diesem Zweck Teile der MAC sowie den SRAM-Arbiters mit angeschlossenem SRAM. Der angeschlossene SRAM wur-

de mit Hilfe eines in Verilog beschriebenen und durch den Hersteller Micron bereitgestellten Modells simuliert. Dementsprechend wird die vollständige Funktionalität des Gesamtsystems getestet.

Sind die simulationsbasierten Tests des im User Data Path enthaltenen Gesamtsystems abgeschlossen, wird der aktuelle Stand der entwickelten Komponenten in den NetFPGA eingespielt. Die Tests bestehen aus dem Zuspielen von Paketen mit unterschiedlichem Inhalt in den Ethernet Headern sowie unterschiedlicher Paketlänge. Da zu Beginn der Umsetzung die Pakete lediglich empfangen werden können, werden über das PCI auszulesende Zähler zur Auswertung der Verarbeitung der Pakete verwendet. Da der Host-PC über die PCI-Schnittstelle den Inhalt des SRAM auswerten kann, lassen sich die empfangenen Paketdaten entsprechend verifizieren. Mit der Integration des eingebetteten Prozessorsystems werden die Tests mit hierfür umgesetzten Testfirmwares durchgeführt. Die Verwendung des eingebetteten Prozessorsystems erlaubt das zuverlässige und präzise Generieren von Testpaketen. Dies ist beispielsweise bei Tests des Arbitrierungsverhaltens des Queue Arbiters hilfreich. Im folgenden Kapitel werden die in den einzelnen Stufen an der Komponente Packet Analyzer durchgeführten Tests beschrieben.

5.2 Durchgeführte Maßnahmen an der Komponente Packet Analyzer

Im Folgenden werden die in den einzelnen Stufen durchgeführten Tests am Beispiel des Packet Analyzers beschrieben. Wie aus den Kapiteln 3.1 und 4.2.2 hervorgeht, ist der Packet Analyzer im Modul Pre Processing für das Extrahieren der benötigten Felder aus den Paketheadern zuständig. Zu extrahieren sind Quell- und Zieladresse sowie die verwendete VID. Die extrahierten Felder sind im internen Paketheader abzulegen. Für die Tests der ersten Stufe wurden dem Packet Analyzer manuell Paketdaten zugespielt und die im internen Paketheader abgelegten Daten überprüft. Die Tests der zweiten Stufe wurden nach Fertigstellung der Komponenten Time Stamper sowie Buffer Sink begonnen. Hierfür wurde für das Modul Pre Processing eine Testbench umgesetzt. Durch die Testbench wird eine Reihe von Paketen mit unterschiedlichen Headern und Längen generiert. Der Strom der Paketdaten kann dabei beliebig manuell über die Flusskontrolle der Paketquelle unterbrochen und fortgesetzt werden. Das Ablegen der Paketdaten durch die Buffer Sink über das Buffer Management kann weiterhin manuell durch andere, an das Buffer Management abgesetzte, Anfragen beeinflusst werden. Daraus ergibt sich beispielsweise die in Abbildung 5.1 dargestellte Verarbeitung eines Paketes. Zu erkennen ist, wie während der Übermittlung des Paketes das Signal *wr_in* gesetzt wird und der Packet

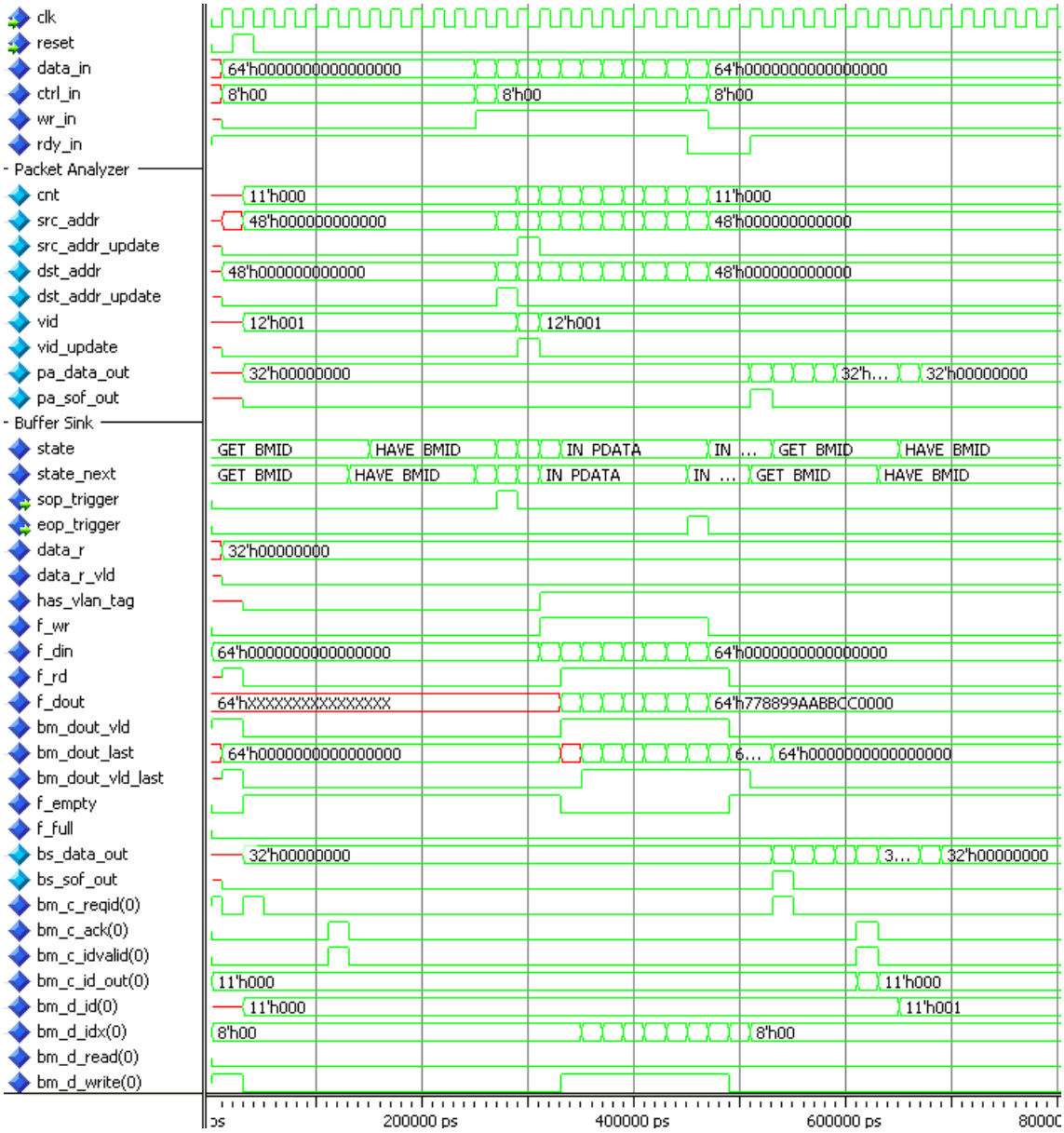


Abbildung 5.1: Verarbeitung eines durch die Testbench für das Modul Pre Processing generierten Paketes

Analyzer die extrahierten Felder durch Setzen der Signale *src_addr_update*, *dst_addr_update* und *vid_update* der Header Update Komponente zur Ablage im nächsten internen Paketheader übergibt. Das Durchlaufen des internen Paketheaders durch die einzelnen Komponenten ist an den Pulsen der *sof* Signale zu erkennen. Nach dem Durchlaufen des Packet Analyzers ist ein Puls an *pa_sof_out* zu erkennen, nach der Buffer Sink ein Puls an *bs_sof_out*. Zur Überprüfung der korrekten Funktionalität des Pre Processing Moduls wurden weitere Signale analysiert.

Für die Tests der dritten Stufe wurde für alle Komponenten eine Testbench verwendet. Sie verfügt über eine Definition von Stimuli für alle vier Ports. Die Definition der Stimuli für einen Port besteht aus einer Liste von Paketen für die jeweils der Zeitpunkt, an dem sie übermittelt werden sollen, die Paketlänge sowie die Paketdaten angegeben werden. Die definierten Paketdaten werden durch den Stimuli-Generator der Testbench an die instantiierten Teile der MAC übergeben und durch sie an den zu testenden User Data Path übergeben.

Daraus ergibt sich beispielsweise die in Abbildung 5.2 dargestellte Verarbeitung eines Paketes. Abgebildet sind lediglich die Signale des Ports mit dem Index 0. An den Signalen *rx_gmac_data(0)* und *rx_gmac_dvld(0)* ist die Übermittlung der letzten Bytes an die MAC zu erkennen, der Puls an *rx_gmac_goodframe* zeigt der MAC die erfolgreiche Validierung der FCS (Frame Check Sum) an. Am gesetzten Signal *in_wr(0)* ist zu erkennen, dass die Paketdaten von der MAC an das Pre Processing Modul übergeben werden und durch den Packet Analyzer und die Buffer Sink verarbeitet werden. Entsprechend der Pulse an den *sof* Signalen durchläuft der interne Paketheader erst den Time Stamper (*ts_sof_out*), dann den Packet Analyzer (*pa_sof_out*) und wird nach Durchlaufen der Buffer Source vom Pre Processing Modul (*sof_out*) an den Input Arbiter weitergegeben und weiter an die FDB übergeben (*fdb_sof_in*). Nach der Verarbeitung des durch Port 0 empfangenen Paketes ist zu erkennen, wie ein Paket eines anderen Ports an die FDB weitergeleitet wird. Im Verlauf der Signale mit dem Präfix *sram_* lassen sich die Zugriffe auf den SRAM erkennen. Die dargestellten Signale dienen lediglich zur Veranschaulichung des Vorgehens. Zur Überprüfung der Funktionalität der Module und Komponenten wurden weitere Signale angezeigt und analysiert.

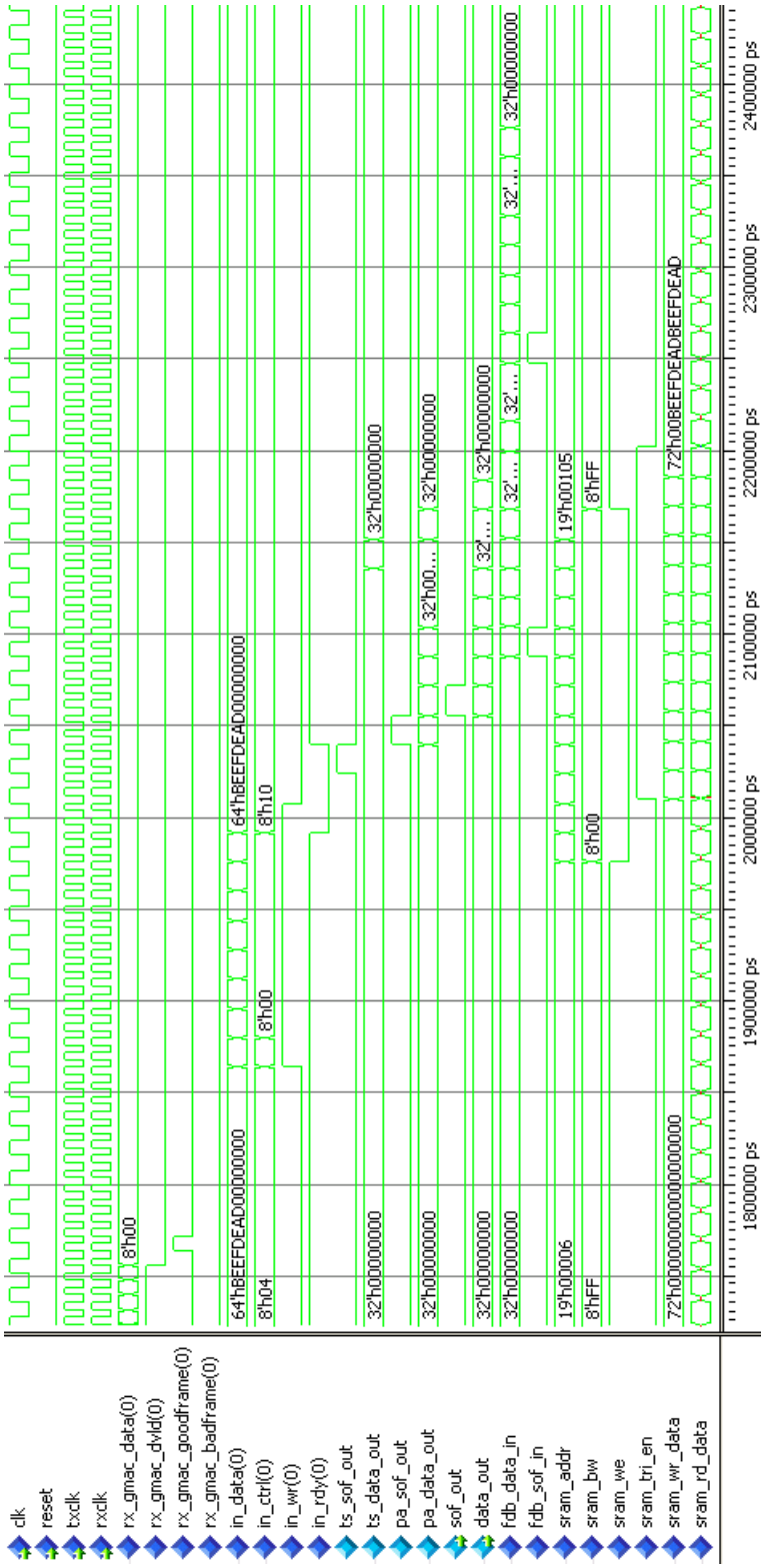


Abbildung 5.2: Verarbeitung eines durch die Testbench für den User Data Path generierten Paketes

6 Evaluation

Die grundlegende Funktionsfähigkeit der RT-Bridge wurde, wie im vorherigen Kapitel beschrieben, mit Hilfe der eingesetzten Maßnahmen zur Qualitätssicherung sichergestellt. Im Folgenden wird diese Arbeit anhand von Messungen evaluiert und somit quantitativ eingeordnet. Im Rahmen der durchgeführten Messungen wurden weiterhin Korrekturparameter zur Korrektur der Zeitstempeleinheiten und der TTE Implementierung ermittelt. Hierfür wird in Abschnitt 6.1 erst der Messaufbau und grundlegende Messungen zu dessen Verwendung beschrieben. In Abschnitt 6.1.1 werden Messungen zur Verifikation des Messaufbaus erläutert. Im Abschnitt 6.2 wird ein Modell für die zu erwartenden Latenzen beim Weiterleiten von Paketen in den Paketklassen AVB, RC und BE entwickelt und mit durchgeführten Messungen verglichen. Dementsprechend wird grundlegend gezeigt, dass das tatsächliche Verhalten der RT-Bridge den Erwartungen entspricht. Um die Korrektheit der Aktualisierung des *transparentClock* Felds in den PC-Frames sowie der Implementierung des Sync Clients der RT-Bridge zu beurteilen, wird der Einfluss der RT-Bridge auf die Präzision der Zeitsynchronisation von TTE in Abschnitt 6.3 betrachtet. Um zu zeigen, dass die Umsetzung des Konzeptes TTE korrekt funktioniert, wird in Abschnitt 6.4 die Genauigkeit des Schedulings von TT Nachrichten analysiert. Weiterhin wird in Abschnitt 6.5 die minimal erreichbare Latenz von TT Nachrichten ermittelt und somit gezeigt, dass mit der RT-Bridge die in Kapitel 2.5 gestellten Anforderungen erreicht wurden.

6.1 Messaufbau

Im Folgenden wird der für die Evaluationsmessungen verwendete Messaufbau beschrieben. Der Messaufbau besteht, wie in Abbildung 6.1 dargestellt, aus einem Sync Master, einem Sync Client, der RT-Bridge sowie einem Oszilloskop. Zum Durchführen der Messungen wurde das Oszilloskop MSO 4054B des Unternehmens Tektronix mit zwei Ethernet Taps TD0500 zum Abgreifen von Paketen direkt von der Leitung verwendet. Im Folgenden wird die auf der RT-Bridge verwendete Software und Konfiguration beschrieben und danach die verwendeten Netzwerkknoten Sync Master und Sync Client.

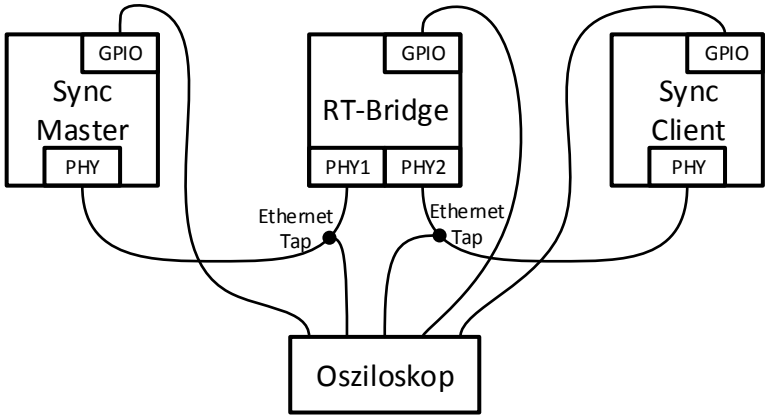


Abbildung 6.1: Zur Evaluation verwendeter Messaufbau

MAC Adresse	Ports Index	Queue	Beschreibung
03:04:05:06:0F:CB	0, 1, 2, 3, 4	AFDX	PC-Frames
03:04:05:06:00:10	2	TT	TT Pakete von Sync Master zu Sync Client
03:04:05:06:00:11	1	TT	TT Pakete von Sync Client zu Sync Master
03:04:05:06:00:20	2	SP/BE	BE Pakete von Sync Master zu Sync Client
03:04:05:06:00:31	2	AVB/CBS	Latenzmessung von Sync Master zu Sync Client
03:04:05:06:00:32	2	AFDX	Latenzmessung von Sync Master zu Sync Client
03:04:05:06:00:33	2	SP/BE	Latenzmessung von Sync Master zu Sync Client

Tabelle 6.1: Im Messaufbau verwendete Konfiguration der FDB

Die im Kapitel 4.7 beschriebenen Softwarekomponenten wurden in Rahmen dieses Messaufbaus verwendet: Komponenten für Softwaretimer, Low-Level Treiber für die Registerinterfaces, Komponenten zur Konfiguration der Hardware, einen Sync Client zur Zeitsynchronisation anhand der TTE PC-Frames, eine Komponente zur Ausführung eines TT Schedules sowie ein Treiber zur Nutzung der Stdio FIFO zur textbasierten Ausgabe von Debuginformationen über die in Xilinx Bibliotheken. Die in Tabelle 6.1 gelisteten Einträge wurden in der FDB abgelegt. Der in Tabelle 6.2 gelistete Schedule wurde für die TT Nachrichten in der RT-Bridge verwendet.

MAC Adresse	Paketlänge	Port		Empfangsfenster	Sendefenster
		in	out		
03:04:05:06:00:10	74	1	2	$3,1\ \text{ms} - 3,2\ \text{ms}$	$3,3\ \text{ms} - 3,33\ \text{ms}$
03:04:05:06:00:11	74	2	1	$3,1\ \text{ms} - 3,2\ \text{ms}$	$3,3\ \text{ms} - 3,33\ \text{ms}$

Tabelle 6.2: Im Messaufbau verwendeter Schedule der TT Pakete in der RT-Bridge

Signal	Debugheader	Beschreibung
in_wr	debug[31] → Pin 37	Am Eingang vom Pre Processing Modul Port 1 mit positiver Taktflanke einen Takt nach Erstellen des Eingangszeitstempels
sof_in	debug[30] → Pin 35	Am Eingang des Post Processing Moduls Port 2 mit positiver Taktflanke zwei Takte vor Erstellen des Ausgangszeitstempels
tx_en	debug[29] → Pin 33	RGMII Signal von Port 2, positive Flanke mit Beginn des ersten Bytes der Präambel eingehender Pakete
rx_dv	debug[28] → Pin 31	RGMII Signal von Port 1, positive Flanke mit Beginn des ersten Bytes der Präambel ausgehender Pakete
gpio	debug[27:0]	Durch Prozessorsystem kontrolliert

Tabelle 6.3: Über den Debugheader aus der RT-Bridge herausgeführte und für die Messungen verwendete Signale

Um präzise Messungen zu erlauben, wurden die in Tabelle 6.3 gelisteten Signale aus der RT-Bridge herausgeführt. Das interne Signal *in_wr* wird zum Messen der Latenz zwischen Paketempfang auf der Leitung und Erstellung des eingehenden Zeitstempels verwendet. Zum Messen der Latenz zwischen Erstellung des ausgehenden Zeitstempels und dem Senden des Paketes auf der Leitung, wird das interne Signal *sof_in* verwendet. Um den Zeitpunkt der Übergabe des Paketes vom PHY an den FPGA zu Messen, wird auf das RGMII Signal *rx_dv* von Port 1 zurückgegriffen. Die Messung des Zeitpunkts der Übergabe eines Paketes vom FPGA an den PHY basiert auf dem RGMII Signal *tx_en* des Ports 2. Durch die Verwendung der RGMII Signale können Sende- und Empfangszeitpunkte von Paketen auf der Leitung ohne Ethernet Taps gemessen werden.

Für die Netzwerkknoten Sync Master und Sync Client wurden Boards des Unternehmens Hilscher mit einem NETX500-ETM Prozessor verwendet. Auf dem Prozessorsystem der Boards wird der aus den Arbeiten von Kai Müller, Flemming Bunzel und Soeren Rumpf [40, 2, 47] hervorgehende TTE-Stack ausgeführt. Der Sync Master übernimmt dabei die Rolle des Synchronisation Masters und stellt eine Zeitbasis anhand von PC-Frames bereit. Der Sync Client übernimmt die Rolle des Clients. Der auf den Netzwerkknoten ausgeführte TTE-Stack wurde mit einer Periode von 10 ms und den Schedules in Tabelle 6.4 konfiguriert.

Mit Hilfe des für die Messungen verwendeten Oszilloskopes lassen sich statistische Auswertungen von Latenzen erstellen. Dabei kann die minimale, maximale und mittlere zeitliche Differenz zwischen den Flanken zweier Signale über eine Zeitspanne ermittelt werden. Diese Funktion ist allerdings nicht für die Ethernet Taps verfügbar, es können nur einzelne manuelle Messungen vorgenommen werden. Dementsprechend wurden die RGMII Signale *rx_dv* und

Zeitpunkt	Sync Master	Sync Client
0 µs	PC-Frame versenden	PC-Frame empfangen, Synchronisation
1000 µs	GPIO setzen	
2000 µs	GPIO freigeben	
3000 µs	TT Paket vorbereiten	
3100 µs	TT Paket senden	
3200 µs	BE Paket vorbereiten	
3280 µs	BE Paket senden	BE Paket empfangen
3300 µs	TT Paket empfangen	
3400 µs	TT Paket verarbeiten	

Tabelle 6.4: Für Evaluationsmessungen verwendeter Schedule in Netzwerkknoten

tx_en auf dem Debug Header des NetFPGA Boards herausgeführt. Beim *rx_dv* Signal tritt am Anfang eines empfangenen Paketes eine positive Flanke auf, beim *tx_en* Signal am Anfang eines gesendeten Paketes.

6.1.1 Verifikation des Messaufbaus

Mit der folgenden Messung wurde verifiziert, dass Messungen an den genannten RGMII Signalen zuverlässig sind und als Referenz bei folgenden Messungen verwendet werden können. Hierfür wurden Pakete über die RT-Bridge weitergeleitet und für die eingehenden Pakete die Latenz vom Beginn des ersten Bytes der Zieladresse im Header des Paketes auf der Leitung zur steigenden Flanke des *rx_dv* Signals gemessen. Für ausgehende Pakete wurde die Latenz von der steigenden Flanke des *tx_en* Signals zum Beginn des ersten Bytes der Zieladresse auf der Leitung gemessen. Um eine Abhängigkeit der Latenz von der Paketlänge auszuschließen, wurde diese variiert. Der Beginn des ersten Bytes der Zieladresse wurde als Referenzpunkt für alle Messungen dieser Evaluaion gewählt und entspricht dem im IEEE Standard 1588 [28] definierten Referenzpunkt.

Die in Abbildung 6.2 dargestellte Messung fand beim Empfang eines Paketes statt, die in Abbildung 6.3 dargestellte Messung beim Senden eines Paketes. Wie in Abbildung 6.2 zu erkennen ist, befindet sich beim Empfang von Paketen die steigende Flanke des *rx_dv* Signals vor dem Beginn des ersten Bytes der Zieladresse, somit ist die gemessene Latenz negativ. Dies resultiert daraus, dass das Signal *rx_dv* durch den PHY generiert wird und den Beginn der Präambel markiert. Wie aus Abbildung 6.3 hervorgeht, wird beim Senden eines Paketes erst die steigende Flanke des *tx_en* Signals detektiert und danach der Beginn des Paketes auf der Leitung. Aus den in Tabelle 6.5 gelisteten Messergebnissen ergibt sich eine Latenz von durchschnittlich -60 ns für eingehende Pakete und 1037 ns für ausgehende Pakete. Wie aus

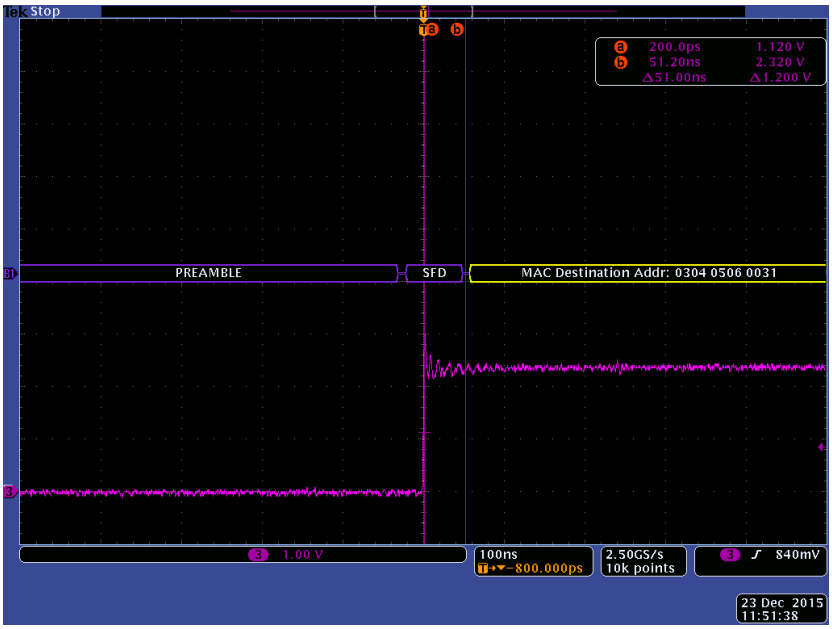


Abbildung 6.2: Validierung von *rx_dv* zur Messung von Paketeingängen (B1: Paket auf der Leitung, 3: *rx_dv*)

Abbildung 6.3: Validierung von *tx_en* zur Messung von Paketausgängen (B1; Paket auf der Leitung, 3: *tx_en*)

Paketlänge	Eingehend	Ausgehend
60 B	−51 ns	1020 ns
100 B	−43 ns	1030 ns
200 B	−83 ns	1042 ns
500 B	−47 ns	1058 ns
1000 B	−46 ns	1022 ns
1500 B	−92 ns	1052 ns

Tabelle 6.5: Messergebnisse der Validierung von *rx_dv* und *tx_en* zur Messung von Paketein- und Paketausgängen

den Messungen hervorgeht, betrug der Jitter weniger als 80 ns. Der gemessene Jitter befindet sich weiterhin im Rahmen des durch die Taktsynchronisierung der Pakete mit dem für den PHY verwendeten Takt von 12, 5 MHz zu erwartenden Jitters. Somit können die Signale *rx_dv* und *tx_en* unter Beachtung der gemessenen Latenzen zum Detektieren von Paketempfang und Paketversand verwendet werden.

Weiterhin wurden die für die Zeitstempeleinheiten benötigten Korrekturwerte ermittelt. Hierfür wurden die auf dem Debug Header der RT-Bridge herausgeführten Signale *in_wr* sowie *sof_in* mit dem Anliegen der Pakete auf der Leitung verglichen. Zwei Takte (16 ns) vor der steigenden Flanke des Signals *in_wr* wird der eingehende Zeitstempel und vier Takte (32 ns) vor der steigenden Flanke des Signals *sof_in* wird der ausgehende Zeitstempel erstellt. Die Messungen wurden an den vom Sync Master zum Sync Client vermittelten PC-Frames durchgeführt. Wie aus den in den Abbildungen 6.4 und 6.5 dargestellten Messungen hervorgeht, wurde für eingehende Pakete ein Korrekturwert von etwa 7280 ns und für ausgehende Pakete ein Korrekturwert von etwa 2270 ns ermittelt.

6.2 Paketlatenzen

Diese Messung befasst sich mit der Ermittlung der minimalen Latenzen und des Jitters beim Weiterleiten von Paketen der Paketklassen AVB, AFDX sowie BE über die RT-Bridge. Im Folgenden wird erst auf das Vorgehen bei den Messungen, danach auf die erwarteten Ergebnisse und dann auf die eingetretenen Ergebnisse eingegangen. Latenz und Jitter wurden in Abhängigkeit von Paketgröße und Paketklasse betrachtet.

Um Pakete flexibel zu senden, wurde der Sync Master im Messaufbau durch einen PC ersetzt. Da der verwendete PC keine PC-Frames sendet und die Synchronisation für die Messungen nicht benötigt werden, wurde der Sync Client in der RT-Bridge deaktiviert. Die Paketlatenz wurde anhand der herausgeführten Signale *rx_dv* und *tx_en* gemessen und um die

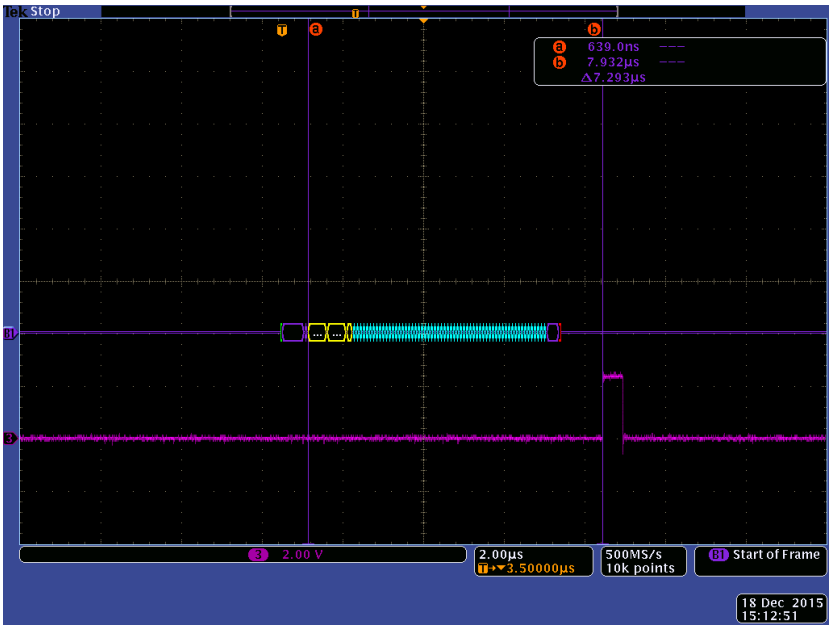


Abbildung 6.4: Messung der Korrekturwerte für die Zeitstempeleinheiten bei eingehenden Paketen (B1: eingehendes Paket auf Leitung, 3: *in_wr*)

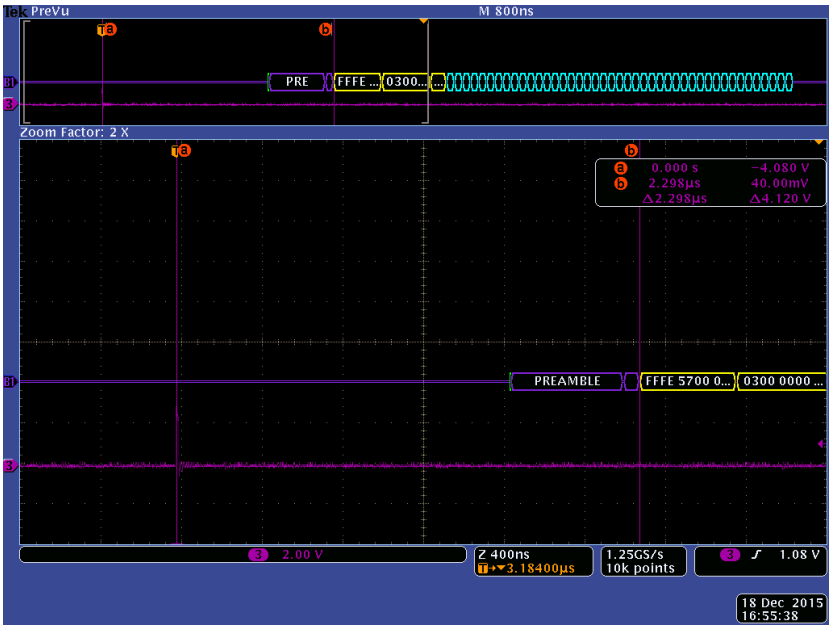


Abbildung 6.5: Messung der Korrekturwerte für die Zeitstempeleinheiten bei ausgehenden Paketen (3: *sof_in*, B1: ausgehendes Paket auf Leitung)

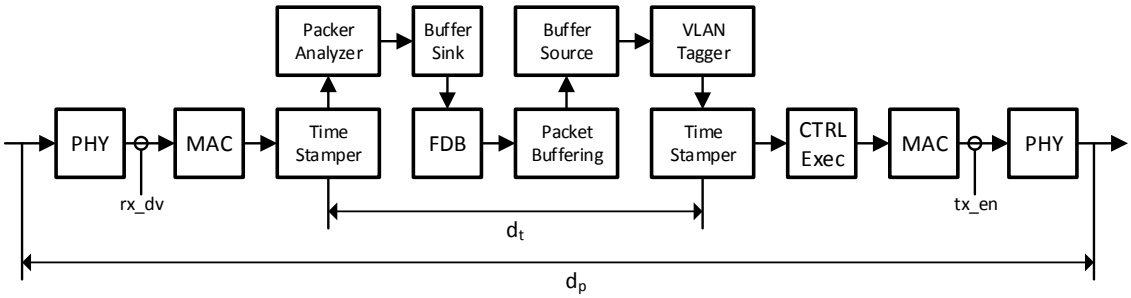


Abbildung 6.6: Bei der Latenzmessung von Paketen durchlaufene Komponenten und gemessene Latenzen

in Tabelle 6.5 ermittelten Werte korrigiert. Dabei wurde die Latenz von 100 Paketen mit der Delaymessung des Oszilloskopes statistisch erfasst. Um die korrekte Funktionsweise der Zeitstempeleinheiten weiter zu belegen, wurde mit Hilfe der durch die Zeitstempeleinheiten erstellten Zeitstempel die Latenz der Pakete beim Durchlaufen der RT-Bridge erfasst. Durch die RT-Bridge wurden somit die Latenzen derselben Pakete durch die in der RT-Bridge vorhandenen Komponenten gemessen. Die in der RT-Bridge erfassten Latenzen wurden ebenfalls statistisch ausgewertet. Die durchgeführten Messungen sind in Abbildung 6.6 zusammen mit den von den Paketen durchlaufenen Komponenten dargestellt. Die Messung des Oszilloskopes beinhaltet mit d_p sämtliche Komponenten inklusive PHYs, die Messung durch die Zeitstempeleinheiten d_t lediglich alle Komponenten zwischen den Time Stampern.

Im Folgenden werden die bei den durchgeführten Latenzmessungen erwarteten Latenzen abgeschätzt. Die Latenzen beim Durchlaufen der einzelnen Komponenten wurden ermittelt und sind in Tabelle 6.6 gelistet. Bei der Ermittlung der Latenzen wurde die Anzahl der benötigten Takte in den Komponenten festgestellt. Durch Multiplikation mit der Periodenlänge des verwendeten Taktes c_{mac} oder c_{core} ergibt sich daraus die Latenz. Wie ersichtlich handelt es sich bei einem Großteil der Komponenten um feste Latenzen, da die enthaltene Logik entsprechend ausgelegt wurde. Die eingesetzten TSAs führen zu keinen zusätzlichen Latenzen, da für den AFDX TSA eine entsprechend große BAG und für den CBS TSA entsprechende Slopes konfiguriert wurden. Dementsprechend sind die Latenzen der betrachteten Paketklassen gleich. Da im eingehenden Teil der MAC das Ende des Paketes abgewartet werden muss, fällt eine Latenz in Abhängigkeit von der Paketlänge an. In der Buffer Sink werden die Paketdaten im Pufferspeicher abgelegt und der interne Paketheader erst nach vollständigem Speichern des Paketes ausgegeben. Dementsprechend ergibt sich nochmals eine von der Paketlänge abhängige Latenz. Diese Latenzen ergeben sich in der Buffer Source und beim ausgehenden Teil der MAC nicht, da jeweils direkt mit der Weitergabe des Paketes begonnen wird. Die an den PHYs

Komponente	**Latenz**
Synchronisation RGMII Signale	$3 * c_{mac}$
MAC eingehend	$(l + 9) * c_{mac}$
Buffer Sink	$(\lceil \frac{l}{8} \rceil + 3) * c_{core}$
Time Stamper eingehend	$1 * c_{core}$
Packet Analyzer	$1 * c_{core}$
Input Arbiter	$2 * c_{core}$
FDB	$7 * c_{core}$
Output Port Multiplexer	$5 * c_{core}$
Queue Multiplexer	$8 * c_{core}$
FIFO Queue	$13 * c_{core}$
Queue Arbiter	$11 * c_{core}$
Buffer Source	$16 * c_{core}$
CTRL Exec	$3 * c_{core}$
MAC ausgehend	$11 * c_{mac} + 20 * c_{core}$
Synchronisation RGMII Signale	$1 * c_{mac}$

Tabelle 6.6: In den Komponenten der RT-Bridge anfallende Latenzen

Parameter	**Wert**	**Beschreibung**
l		Paketlänge in Byte, ohne Präambel, SFD und FCS.
c_{core}	16 ns	Taktlänge des Kerntaktes.
c_{mac}	80 ns	Taktlänge des Betriebstaktes der MACs.
d_{PHYin}	−60 ns	Gemessene eingehende Latenz über PHY.
d_{PHYout}	1037 ns	Gemessene ausgehende Latenz über PHY.
d_{TSin}	7280 ns	Korrekturwert für eingehende Zeitstempel.
d_{TSout}	2270 ns	Korrekturwert für ausgehende Zeitstempel.

Tabelle 6.7: Zur Abschätzung der Latenzen in der RT-Bridge verwendete Parameter

anfallenden Latenzen d_{PHYin} und d_{PHYout} wurden im Kapitel 6.1.1 ermittelt. Die Latenzen werden weiterhin von den in Tabelle 6.7 gelisteten Parametern beeinflusst.

Die erwarteten Paketlatenzen d_p und d_t ergeben sich somit durch Summieren der in den durchlaufenen Komponenten anfallenden Latenzen. Die durchlaufenen Komponenten sind in Abbildung 6.6 dargestellt, die jeweiligen Latenzen sind den Tabellen 6.6 und 6.7 zu entnehmen. Dabei ist zu beachten, dass der eingehende Zeitstempel mit Erreichen der eingehenden Paketdaten in der Buffer Sink erstellt wird. Der ausgehende Zeitstempel wird 4 Takte nach Ankunft des ausgehenden internen Paketheaders in der Buffer Sink erstellt. Somit ergibt sich:

$$d_p = d_{PHYin} + (l + 24) * c_{mac} + (\lceil \frac{l}{8} \rceil + 90) * c_{core} + d_{PHYout}$$
$$d_t = d_{TSin} + (\lceil \frac{l}{8} \rceil + 55) * c_{core} + d_{TSout}$$

Wie aus den aufgestellten Formeln für d_p und d_t hervorgeht, wird durch die Latenzmessungen der Zeitstempeleinheiten die in den MACs anfallende und von der Paketgröße anhängige Latenz nicht erfasst. Da die Zeitstempeleinheiten keine von der Paketlänge abhängige Korrektur erlauben, kann die Korrektur lediglich auf eine Paketlänge abgestimmt werden. Entsprechend müssen Zeitstempel für die Sende- und Empfangszeitfenster von TT Nachrichten bei Bedarf nach entsprechender Genauigkeit anhand der gegebenen Formeln in Software korrigiert werden. Für die Aktualisierung des *transparrentClock* Felds der PC-Frames kann eine Korrektur durch Software nicht stattfinden, da sie komplett in Hardware umgesetzt ist. Da lediglich PC-Frames mit einer festen Länge verwendet werden, kann die statische Korrektur der Zeitstempeleinheiten hierauf abgestimmt werden. Die Korrekturwerte wurden anhand der am Ende dieses Abschnittes durchgeführten Messung ermittelt.

Das Ergebnis der durchgeführten Messungen ist in den Tabellen 6.8 und 6.9 zusammen mit den erwarteten Latenzen sowie der Abweichung zwischen erwarteter und eingetretener Latenz gelistet. Hierbei handelt es sich lediglich um die Ergebnisse der Messung der Paketklasse AVB. Die Latenzen der Paketklassen AFDX und BE wurden ebenfalls gemessen und ausgewertet. Da sie sich aber nicht signifikant von den Ergebnissen der Messung der Paketklasse AVB unterscheiden, wurden sie hier nicht angefügt.

Wie ersichtlich beträgt der Jitter beim Weiterleiten von Paketen maximal 100 ns und ist nicht abhängig von Paketklasse und Paketlänge. Dieser liegt etwa in der Größenordnung des bei der Synchronisation der Paketdaten zwischen dem Kerntakt und dem Takt der MACs anfallenden Jitters. Der Jitter beim Messen der Paketlatenz mit Hilfe der Zeitstempeleinheiten beträgt maximal 16 ns, also einen Takt innerhalb des User Data Path. Auch dies spricht für die Synchronisation der Paketdaten zwischen den unterschiedlichen Takten als Quelle des Jitters.

Länge	Messung				$\mathbf{d_p}$	Abweichung des Mittels
	Min	Mittel	Max	Jitter		
60 B	9,456 µs	9,511 µs	9,538 µs	82 ns	9,265 µs	0,246 µs
100 B	12,66 µs	12,70 µs	12,74 µs	80 ns	12,55 µs	0,15 µs
200 B	20,74 µs	20,79 µs	20,82 µs	80 ns	20,74 µs	0,05 µs
500 B	45,36 µs	45,40 µs	45,44 µs	80 ns	45,35 µs	0,05 µs
1000 B	86,34 µs	86,39 µs	86,43 µs	90 ns	86,34 µs	0,05 µs
1500 B	127,4 µs	127,4 µs	127,5 µs	100 ns	127,3 µs	0,1 µs

Tabelle 6.8: Messergebnisse der Paketlatenzen für die Paketklasse AVB von PHY zu PHY

Länge	Messung			$\mathbf{d_t}$	Abweichung des Mittels
	Min	Mittel	Max		
60 B	10,734 µs	10,734 µs	10,750 µs	10,558 µs	0,176 µs
100 B	10,734 µs	10,734 µs	10,750 µs	10,638 µs	0,096 µs
200 B	10,814 µs	10,814 µs	10,814 µs	10,830 µs	0,016 µs
500 B	11,422 µs	11,422 µs	11,422 µs	11,438 µs	0,016 µs
1000 B	12,414 µs	12,414 µs	12,414 µs	12,430 µs	0,016 µs
1500 B	13,422 µs	13,422 µs	13,422 µs	13,438 µs	0,016 µs

Tabelle 6.9: Messergebnisse der Paketlatenzen für die Paketklasse AVB zwischen den Zeitstempeleinheiten

Weiterhin konnte gezeigt werden, dass die errechneten Latenzen weitgehend korrekt sind. Lediglich bei Paketen der Länge 60 B sowie 100 B traten unerwartet hohe Abweichungen von über 100 ns auf. Dass ein Großteil dieser Abweichungen bei den Messungen mit Hilfe der Zeitstempeleinheiten auftreten, deutet auf eine Quelle der Abweichungen innerhalb des User Data Paths hin. Da die Messungen einen minimalen Jitter aufweisen, muss es sich um einen in jedem Fall auftretende Abweichung handeln, die also nicht zufällig eintritt. Da die Latenzen der Komponenten Time Stamper, Packet Analyzer, Input Arbiter, Filtering Database, Output Port Multiplexer, Queue Multiplexer, FIFO Queue und Queue Arbtier sich trivial ergeben und mehrfach überprüft wurden, können diese Komponenten als Quelle für die Abweichung ausgeschlossen werden. Weiterhin ergibt sich, wie in den vorherigen Abschnitten erläutert, durch die TSA keine Latenz. Die ausgehenden Zeitstempel werden exakt 4 Takte nach Eintreffen des internen Paketheaders in der Buffer Source erstellt. Somit kann die Buffer Source als Quelle für die Abweichungen ausgeschlossen werden. Als mögliche Quelle für die Abweichungen verbleibt lediglich die Buffer Sink. Das aufgestellte Modell der in der Buffer Sink auftretenden Latenzen wurde jedoch bereits anhand von Simulationen durch ModelSim verifiziert. Zu beachten ist weiterhin, dass es sich bei der gemessenen Abweichung lediglich um weniger als 3 % der anfallenden Latenz handelt. Um ein korrektes und vollständiges Modell der in der RT-Bridge anfallenden Latenzen zu erhalten, sollten in zukünftigen Arbeiten jedoch weitere Analysen durchgeführt werden.

Da die Komponenten Buffer Sink und Buffer Source beim Ablegen und Auslesen der Paketdaten über das Buffer Management in diesem Szenario nicht mit den anderen Ports konkurrieren, handelt es sich weiterhin bei den gemessenen Latenzen und Jitter um Minimalwerte. Sobald mehrere Komponenten um den Zugriff auf die Paketdaten konkurrieren, entstehen höhere Latenzen und Jitter. Diese werden durch die Zeitstempeleinheiten erfasst. Um maximale Latenzen und Jitter zu erreichen, müsste somit auf allen Ports gleichzeitig ein Paket gesendet und empfangen werden. Da sich dies mit den verfügbaren Komponenten in einem Messaufbau nicht zuverlässig reproduzierbar umsetzen lässt, ist dieser in den hier aufgeführten Messungen nicht enthalten.

6.3 Einfluss auf die TTE Zeitsynchronisation

Zur Umsetzung von Konzepten wie TT und TSN Scheduled Traffic müssen alle beteiligten Netzwerkknoten auf eine gemeinsame Zeitbasis synchronisiert werden. Im Rahmen der RT-Bridge wird das in Kapitel 2.3.4 beschriebene Synchronisationsverfahren von TTE verwendet. Zur präzisen Synchronisation aller Netzwerkknoten muss in den für die Synchronisation

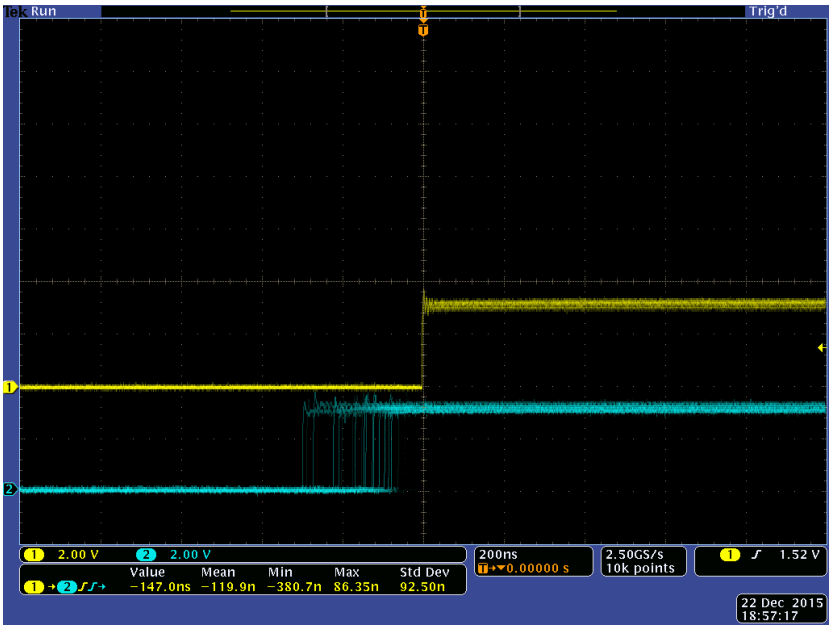


Abbildung 6.7: Messung zur Bewertung des Einflusses auf die TTE Zeitsynchronisation ohne RT-Bridge (1: GPIO Sync Master, 2: GPIO Sync Client)

verwendeten PC-Frames die seit dem Versand vergangene Zeit erfasst werden. Wie in den Kapiteln 3 und 4 beschrieben, werden hierfür in der RT-Bridge die umgesetzten Zeitstempeleinheiten verwendet. Weiterhin muss die RT-Bridge anhand der PC-Frames synchronisiert werden. Im Folgenden werden die durchgeführten Messungen zur Abschätzung des Einflusses der RT-Bridge auf die Synchronisation zwischen angeschlossenen Netzwerkknoten sowie zur Bewertung des Synchronisationsclients der RT-Bridge dargestellt.

Für die durchzuführenden Messungen wird der beschriebene Messaufbau verwendet. Zur Bestimmung der Qualität der Synchronisation wird auf Sync Master, Sync Client sowie der RT-Bridge zum selben Zeitpunkt im Schedule ein GPIO gesetzt. Da eine Fehlerhafte in der Synchronisation in einer entsprechend falschen Ausführung des Schedules resultiert, kann somit die Qualität der Synchronisation bewertet werden. Die GPIOs werden, wie in Tabelle 6.4 beschrieben, gesetzt, auf der RT-Bridge geschieht dies zum selben Zeitpunkt. Mit Hilfe des Oszilloskopes wurde eine Delaymessung vom GPIO des Sync Master zum Sync Client sowie vom Sync Master zur RT-Bridge mit einer statistischen Auswertung über 500 gemessene Perioden durchgeführt. Um den Einfluss der RT-Bridge auf die Zeitsynchronisation zu bewerten, wurden Sync Master und Sync Client einerseits direkt und andererseits über die RT-Bridge verbunden.

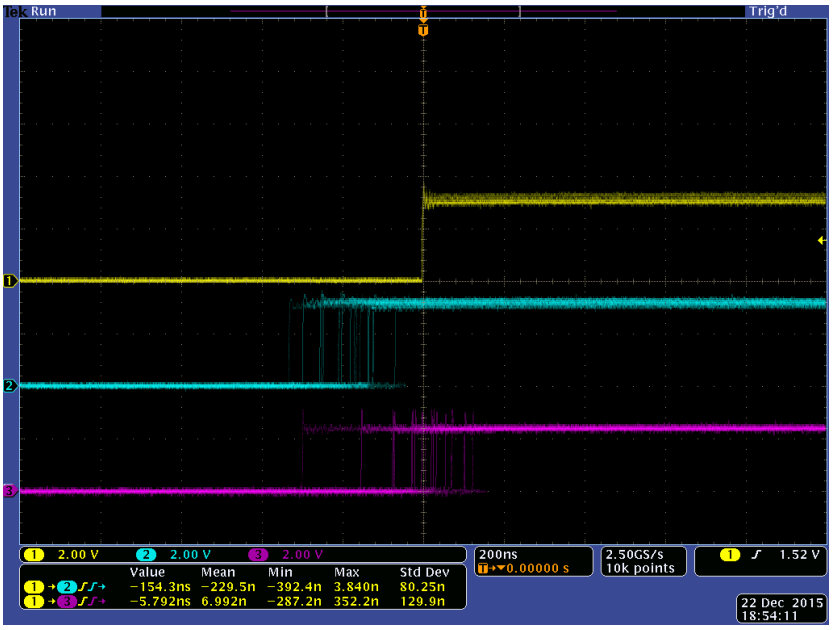


Abbildung 6.8: Messung zur Bewertung des Einflusses auf die TTE Zeitsynchronisation mit RT-Bridge (1: GPIO Sync Master, 2: GPIO Sync Client, 3: GPIO RT-Bridge)

Aus der in Abbildung 6.7 dargestellten Messung ergibt sich für die Synchronisation ohne RT-Bridge ein Jitter von 467 ns und ein Offset von 86 ns. Aus der in Abbildung 6.8 dargestellten Messung mit eingefügter RT-Bridge ergibt sich ein Jitter von 396 ns sowie ein Offset von 230 ns. Somit wird durch die RT-Bridge die Synchronisation nicht verschlechtert, lediglich ein zusätzlicher Offset von 144 ns ergibt sich. Der Offset lässt sich durch eine entsprechend optimierte Konfiguration der RT-Bridge korrigieren. Weiterhin geht aus Abbildung 6.8 hervor, dass die Synchronisation der RT-Bridge mit einem Jitter von 639 ns sowie einem Offset von 130 ns schlechter als die Synchronisation des TTE-Stacks ist. Da es sich hierbei allerdings um einen Bruchteil der in Kapitel 2.5 genannten Anforderung an die Präzision beim Paketversand handelt, wurde der umgesetzte Sync Client nicht weiter optimiert.

6.4 Scheduling von TT Nachrichten

Im Folgenden wird die Präzision der RT-Bridge beim Scheduling von TT Nachrichten anhand von Messungen ermittelt. Hierfür wird der beschriebene Messaufbau verwendet. Die Messung findet anhand der im Schedule eingetragenen, vom Sync Master zum Sync Client vermittelten TT Nachrichten statt. Zur Erfassung des Offsets wurde am Sync Master zum geplanten Sendezeitpunkt der TT Nachricht von der RT-Bridge zum Sync Client ein GPIO gesetzt und

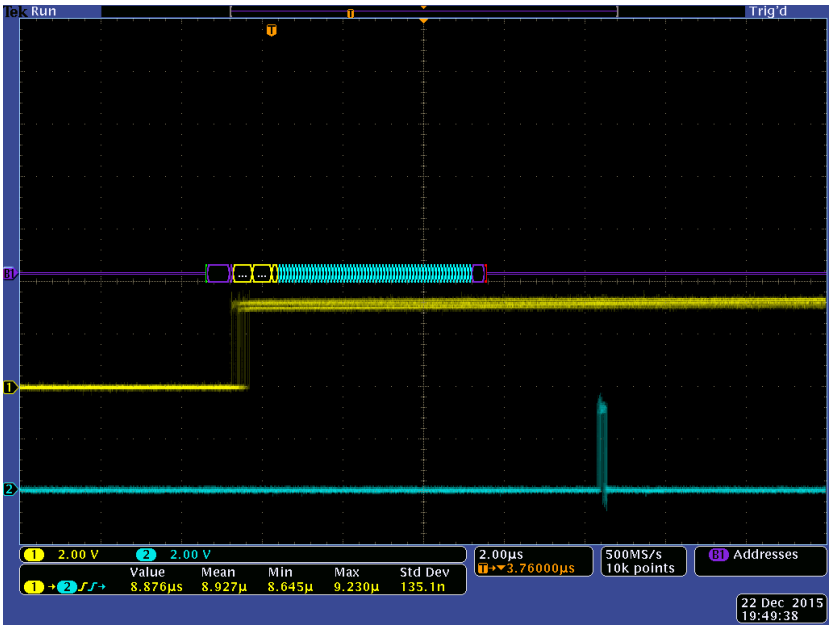


Abbildung 6.9: Messung der Präzision des Schedulings von TT Nachrichten (1: GPIO gesetzt zum geplanten Sendezeitpunkt, B1: TT Nachricht auf der Leitung, 2: GPIO gesetzt bei Ankunft beim Empfänger)

der Versand der TT Nachricht von der RT-Bridge zum Sync Client mit einem Ethernet Tap erfasst. Wie aus Abbildung 6.9 ersichtlich, betrug der Offset weniger als 400 ns. Um den Jitter mit Hilfe einer Messung des Delays und entsprechender statistischer Auswertung zu messen, wurde im TTE-Stack des Sync Clients beim Paketempfang ein GPIO gesetzt. Da der GPIO zum Beginn der den Paketempfang behandelnden ISR (Interrupt Service Routine) gesetzt wurde, ist von einem geringen Jitter auszugehen. Aus der in Abbildung 6.9 dargestellten Messung geht dementsprechend ein Jitter von 585 ns hervor. Somit werden die in Kapitel 2.5 gestellten Anforderungen erfüllt. Die Größenordnung des Jitters entspricht etwa dem Jitter der Synchronisation. Daher lässt sich bei Bedarf durch eine Optimierung der Synchronisation der RT-Bridge der Jitter weiter reduzieren.

Weiterhin ist zum Erreichen eines zuverlässigen Schedulings der Einfluss von Paketen mit geringerer Priorität auszuschließen. Um den maximalen Einfluss eines Paketes mit geringerer Priorität zu erreichen, wurde ein BE Paket durch den Sync Master so versandt, dass es entweder direkt vor oder nach der TT-Nachricht von der RT-Bridge zum Sync Client vermittelt wird. Zur Messung des entstehenden Jitters wurde der als Referenz dienende GPIO des Sync Masters sowie die RGMII Leitung *tx_en* des ausgehenden Ports an der RT-Bridge verwendet. Die Delaymessung des Oszilloskopes konnte nicht verwendet werden, da die Messung von

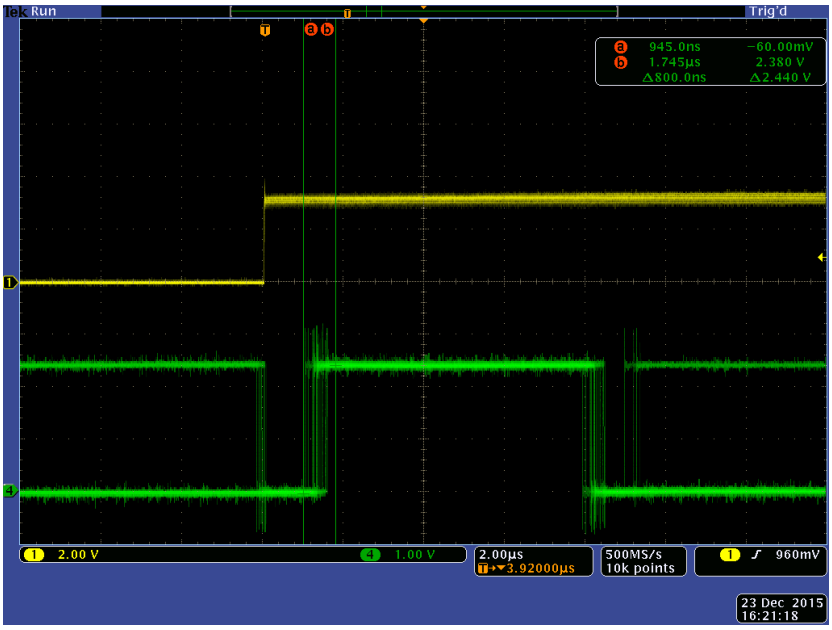


Abbildung 6.10: Messung des Einflusses von BE Nachrichten auf TT Nachrichten (1: GPIO gesetzt zum geplanten Sendezeitpunkt, 4: *tx_en*)

der Flanke des ersten Signals entweder zur ersten oder letzten Flanke des Zweiten Signals durchgeführt wird. Wie aus der Messung in Abbildung 6.10 hervorgeht, beträgt der so hervorgerufene Jitter wesentlich weniger als 800 ns. Somit kann der Jitter der TT Nachrichten durch Nachrichten mit geringerer Priorität kaum erhöht werden.

6.5 Minimale Latenz von TT Nachrichten

Um die mit der RT-Bridge minimale erreichbare Latenz bei der Weiterleitung von TT Nachrichten zu ermitteln, wurde der in Abschnitt 6.1 beschriebene Messaufbau wie folgt angepasst. Der Beginn des Zeitfenster zum Senden der TT Nachrichten wurde auf 5 µs nach den Beginn des Empfangszeitfensters angepasst. Dementsprechend werden die eingehenden Nachrichten mit einer minimalen Verzögerung weitergeleitet. Entsprechend präzise muss der Sender der Nachrichten den Zeitpunkt des Versandes veranlassen. Die Latenz wurde analog zu Abschnitt 6.2 anhand der RGMII Signale *rx_dv* und *tx_en* gemessen und statistisch über 100 Pakete ausgewertet. Weiterhin wurde am Sync Master 300 µs nach dem Senden der TT Nachricht ein GPIO gesetzt. Dieser GPIO wurde zum Triggern des Oszilloskopes verwendet. Daraus ergibt sich die in Abbildung 6.11 abgebildete Messung. Nach der Korrektur der gemessenen Latenz um die Latenzen der PHYs ergibt sich eine garantierte minimale Latenz von 10, 7 µs bei ei-

Abbildung 6.11: Messung der minimalen Latenz bei der Weiterleitung von von TT Nachrichten (1: GPIO Sync Master, 3: *rx_dv*, 4: *tx_en*)

nem Jitter von 130 µs. Somit liegt die durch die RT-Bridge minimal erreichbare Latenz bei der Weiterleitung von TT Nachrichten wesentlich unter dem in Kapitel 2.5 geforderten Ziel von 20 µs.

7 Zusammenfassung

Ziel dieser Arbeit war es, ein Framework für Prototypen von TSN Switches zu entwickeln. Dafür wurde das Konzept einer Hardwarearchitektur entwickelt und auf Basis der NetFPGA 1G Plattform umgesetzt. Mit dem umgesetzten RT-Bridge Framework wurde der Prototyp eines TSN Switches mit Unterstützung für mehrere RT-Ethernet Konzepte implementiert. Um zu ermitteln, ob die anfangs gesetzten Ziele erreicht wurden, wurde der Prototyp des TSN Switches Evaluationsmessungen unterzogen.

Zu Beginn dieser Arbeit wurde die Motivation für die Verwendung von Ethernet als Fahrzeugnetz im Automobil aufgezeigt. Aus den Anforderungen von Anwendungen im Automobil ergibt sich der Bedarf nach einem echtzeitfähigen Fahrzeugnetz. Die Anforderungen von unter Echtzeitbedingungen stehenden Anwendungen an die verwendeten Netzwerke wurden ermittelt. Dabei stellen insbesondere neue Anwendungen wie ADAS Anforderungen an das Fahrzeugnetz, die von den heute verwendeten Technologien nicht mehr erfüllt werden können. Da Ethernet nicht Echtzeitfähigkeit ist, wurden Konzepte zur Erweiterung von Ethernet um Echtzeitfähigkeit betrachtet. Die betrachteten Konzepte für RT-Ethernet weisen unterschiedliche Eigenschaften auf. Daher sind die einzelnen Konzepte jeweils für Anwendungen mit bestimmten Anforderungen an die Echtzeitfähigkeit optional. Da zukünftige Fahrzeuge viele unterschiedliche Anwendungen mit unterschiedlichen Anforderungen an die Echtzeitfähigkeit des Fahrzeugnetzes enthalten werden, ergibt sich die Notwendigkeit der Kombination mehrerer Konzepte für RT-Ethernet. Auf dem Markt sind aktuell keine Bridges verfügbar, die das flexible Kombinieren mehrerer Konzepte für RT-Ethernet erlauben. Auch betrachtete, im Sourcecode frei verfügbare Projekte auf Basis der NetFPGA Plattform erfüllen diese Anforderung nicht. Dementsprechend lassen sich mit dem in dieser Arbeit umgesetzten RT-Bridge Framework mehrere Konzepte für RT-Ethernet flexibel zu einer Bridge kombinieren. Um das Erreichen der für diese Arbeit gesetzten Ziele anhand von Evaluationsmessungen zu bewerten, wurden die Ziele und Anforderungen klar definiert.

Danach wurde ein Konzept einer Hardwarearchitektur zur Erfüllung der gesetzten Ziele und Anforderungen entwickelt. Sie erlaubt die Vermittlung von Ethernet-Paketen mit geringen Latenzen und Jitter. Heutige und zukünftige Konzepte für RT-Ethernet lassen sich flexibel

und mit geringem Entwicklungsaufwand als Prototyp einer Bridge umsetzten. So wurde der CBS TSA in 162 Zeilen VHDL-Code, der TSA für einen virtuellen AFDX Link in 140 Zeilen umgesetzt (jeweils inklusive Kommentar). Die entwickelte Hardwarearchitektur erlaubt dabei insbesondere das Kombinieren mehrerer Konzepte für RT-Ethernet. Der geringe Umfang der TSAs zeigt, dass die Komplexität bei der Umsetzung von Konzepten für RT-Ethernet, trotz der hohen Gesamtkomplexität der übrigen Komponenten der RT-Bridge, gering ist. Da für zukünftige Nutzer die Umsetzung der realisierten Bestandteile ausbleibt, reduziert sich für ihn die Komplexität bei der Realisierung von neuen RT-Ethernet Konzepten. Somit wurde eine signifikante Hürde bei der Umsetzung von Prototypen für Bridges in RT-Ethernet Netzwerken genommen.

Um die Qualität der in dieser Arbeit umgesetzten Komponenten zu sichern, wurde ein definiertes Vorgehen mit Tests in mehreren Stufen verwendet. An die Entwicklung von Komponenten schließen sich dabei direkt Tests mit einem Simulationstool an. Wenn ein definierter Teil eines Moduls fertiggestellt wird, folgen Tests des Moduls sowie Tests mit allen bisher umgesetzten Komponenten. Sind die mit dem Simulationstool durchgeführten Tests erfolgreich, werden die umgesetzten Komponenten Tests in der Hardware des NetFPGA Boards unterzogen. Mit diesem Vorgehen wurde die korrekte Funktion der umgesetzten Komponenten gewährleistet. Weiterhin wurden die Ergebnisse dieser Arbeit anhand von Messungen evaluiert und somit quantitativ eingeordnet. Hierfür wurde erst die Korrektheit des verwendeten Messaufbaus anhand von Messungen verifiziert und Modelle für die erwarteten Messergebnisse erarbeitet. Die darauf folgenden Evaluationsmessungen ergaben, dass die gesetzten Ziele und Anforderungen erreicht wurden. Allerdings ergab sich auch, dass bezüglich des Modells der in der RT-Bridge anfallenden Latenzen in Zukunft für weitere Arbeiten ein Potential zur Optimierung besteht.

Bisherige Arbeiten im Rahmen der CoRE-Gruppe (Communication over Real-Time Ethernet) haben sich unter anderem mit der simulationsbasierten Evaluation zukünftigen Fahrzeugnetze befasst. Dabei wurden auch Kombinationen mehrerer Konzepte für RT-Ethernet evaluiert. Für bisherige, in Hardware umgesetzte Prototypen wurden Switches des Unternehmens TTTech verwendet. Das Resultat dieser im Rahmen der CoRE-Gruppe durchgeführten Arbeit ist ein Schritt in Richtung der Evaluation von Kombinationen mehrerer RT-Ethernet Konzepte anhand von Prototypen in Hardware.

Weitere Arbeiten könnten …

… sich mit der Umsetzung der AVB Konfigurationsprotokolle auf Basis von MRP (MMRP, MVRP, MSRP) beschäftigen. Somit wäre die RT-Bridge AVB mit standardkonformen AVB-Bridges kompatibel.

- ... die Konfiguration der Schedules über Konfigurationsfiles erarbeiten. Die Konfiguration würde über den Host-PC in die RT-Bridge eingespielt. Diese Arbeiten könnten sich auch mit zukünftigen Vorgehensweisen und Toolchains zur Entwicklung von Fahrzeugnetzen auseinandersetzen.
- ... die RT-Bridge auf eine neuere Version der NetFPGA Boards portieren, z.B. das NetFPGA CML Board. Somit stünden umfangreichere Ressourcen für weitere Arbeiten zur Verfügung.
- ... umfangreichere und präzisere Modelle zur Abschätzung der in der RT-Bridge zu erwartenden Latenzen und Jitter aufstellen.
- ... ein Konzept zur automatischen Generierung des Moduls Packet Buffering umsetzen. Einzelne TSAs und Queues der Konzepte werden aktuell manuell im Modul Packet Buffering instantiiert und zusammengeführt. Die TSAs und Queues ließen sich mit einer Domain Specific Language (DSL) noch flexibler und mit weniger Entwicklungsaufwand kombinieren.

Literatur

[1] *Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network.* Aeronautical Radio, Incorporated, 2009.

[2] Flemming Bunzel. „Hardware-Abstraktion eines Open Source Echtzeit Ethernet Stacks: Entwurf, Umsetzung und Evaluation“. Bachelorthesis. Hamburg: Hochschule für Angewandte Wissenschaften Hamburg, 2013. URL: http://core-researchgroup.de/bib/eigene/b-haose-13.pdf.

[3] *CAN Specification Version 2.0.* Bosch, 1991. URL: http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf.

[4] G. Carvajal und S. Fischmeister. „A TDMA Ethernet Switch for Dynamic Real-Time Communication“. In: *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on.* 2010, S. 119–126. DOI: 10.1109/FCCM.2010.27.

[5] Cypress Semiconductor Corporation. *18-Mbit (512 K × 36/1 M × 18) Pipelined SRAM with NoBL Architecture.* Revision R. Online Quelle, abgerufen am 13.01.2015. Cypress Semiconductor Corporation. Sep. 2014. URL: http://www.cypress.com/?mpn=CY7C1370D-167AXC.

[6] S. Dabral u. a. „Trends in camera based Automotive Driver Assistance Systems (ADAS)“. In: *Circuits and Systems (MWSCAS), 2014 IEEE 57th International Midwest Symposium on.* 2014, S. 1110–1115. DOI: 10.1109/MWSCAS.2014.6908613.

[7] A. Diarra und A. Zimmermann. „System design issues for future in-vehicle Ethernet-based time- and safety-critical networks“. In: *Systems Conference (SysCon), 2015 9th Annual IEEE International.* 2015, S. 61–68. DOI: 10.1109/SYSCON.2015.7116730.

[8] Dirk W. Hoffmann. *Software-Qualität.* Springer-Verlag Berlin Heidelberg, 2008. DOI: 10.1007/978-3-540-76323-9.

[9] Franz-Josef Götz. *Alternative Shaper for Scheduled Traffic in Time Sensitive Networks*. IEEE 802.1 TSN TG Meeting – Vancouver, Abgerufen am 07.12.2015. Jan. 2013. URL: http://www.ieee802.org/1/files/public/docs2013/new-goetz-TSN-4-Industrial-Networks-20130115-v1.pdf.

[10] Franz-Josef Götz. *Traffic Shaper for Control Data Traffic (CDT) @ Industry*. IEEE 802 AVB Meeting – San Diego, Abgerufen am 07.12.2015. Juli 2012. URL: http://ieee802.org/1/files/public/docs2012/new-goetz-CtrDataScheduler-0712-v1.pdf.

[11] Franz-Josef Goetz. *Guaranteed Latency for Control-Data-Traffic in Time Sensitive Networks*. IEEE 802.1 TSN TG Meeting York – England, Abgerufen am 05.01.2016. Sep. 2013. URL: http://www.ieee802.org/1/files/public/docs2013/bv-goetz-TSN-GuaranteedLatency4CDT-20130904-v1.pdf.

[12] *IEEE 802.1AS Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*. IEEE, März 2011. DOI: 10.1109/IEEESTD.2011.5741898.

[13] *IEEE 802.1BA Standard for Local and metropolitan area networks–Audio Video Bridging (AVB) Systems*. IEEE, Sep. 2011. DOI: 10.1109/IEEESTD.2011.6032690.

[14] *IEEE 802.1Q Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks*. IEEE, Aug. 2011. DOI: 10.1109/IEEESTD.2011.6009146.

[15] *IEEE 802.1Q Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks*. IEEE, Aug. 2011. DOI: 10.1109/IEEESTD.2011.6009146.

[16] *IEEE 802.1Qat Amendment 14: Stream Reservation Protocol (SRP) - 802.1Qat*. IEEE, Mai 2010.

[17] *IEEE 802.1Qav Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams - 8.2.1Qav*. IEEE, Dez. 2010.

[18] *IEEE 802.3 Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*. IEEE, 2008.

[19] *IEEE 802.3bp/D2.1: Draft Standard for Ethernet Amendment Physical Layer Specifications and Management Parameters for 1 Gb/s Operation over Fewer than Three Twisted Pair Copper Cable*. IEEE, Nov. 2015.

[20] *IEEE 802.3br/D2.3: Draft Standard for Ethernet Amendment Specification and Management Parameters for Interspersing Express Traffic*. IEEE, Nov. 2015.

[21] *IEEE 802.3bs/D1.0: Draft Standard for Ethernet Amendment: Media Access Control Parameters, Physical Layers and Management Parameters for 400 Gb/s Operation*. IEEE, Nov. 2015.

[22] *IEEE P802.1CB/D1.0: Draft Standard for Local and Metropolitan Area Networks – Seamless Redundancy*. Jan. 2015.

[23] *IEEE P802.1Qbu/D3.1: Draft Standard for Local and Metropolitan Area Networks – Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks – Amendment: Frame Preemption*. Version 3.1. Okt. 2015.

[24] *IEEE P802.1Qbv/D3.1: Draft Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment: Enhancements for Scheduled Traffic*. Sep. 2015.

[25] *IEEE P802.1Qcc/D0.3: Draft Standard for Local and metropolitan area networks – Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements*. Dez. 2014.

[26] *IEEE P802.1Qch/D0.0: Draft Standard for Local and metropolitan area networks – Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment: Cyclic Queuing and Forwarding*. Aktuell (02.12.2015) kein Draft verfügbar. Dez. 2015.

[27] *IEEE P802.1Qch/D0.0: Draft Standard for Local and metropolitan area networks – Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment: Per-Stream Filtering and Policing*. Nov. 2015.

[28] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. 2008. DOI: 10.1109/IEEESTD.2008.4579760.

[29] Xilinx Inc. *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet*. v5.0. DS083. 2011.

[30] Donald Ervin Knuth. *The art of computer programming – Fundamental algorithms*. 3. Aufl. Bd. 1. Addison Wesley Longman, 1997. ISBN: 0-201-89683-4.

[31] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. 2. Aufl. Springer, 2011.

[32] Benjamin Krill. *Round-Robin Arbiter*. Online Quelle, Abgerufen am 07.04.2015. 2013. URL: http://www.krll.de/en/portfolio/round-robin-arbiter/.

[33] Martin Labrecque und J. Gregory Steffan. „NetTM: Faster and Easier Synchronization for Soft Multicores via Transactional Memory“. In: *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. FPGA '11. Monterey, CA, USA: ACM, 2011, S. 29–32. ISBN: 978-1-4503-0554-9. DOI: 10.1145/1950413.1950422. URL: http://doi.acm.org/10.1145/1950413.1950422.

[34] Martin Labrecque und J. Gregory Steffan. „The Case for Hardware Transactional Memory in Software Packet Processing“. In: *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ANCS '10. La Jolla, California: ACM, 2010, 37:1–37:11. ISBN: 978-1-4503-0379-8. DOI: 10.1145/1872007.1872053. URL: http://doi.acm.org/10.1145/1872007.1872053.

[35] Martin Labrecque u. a. „NetThreads: Programming NetFPGA with Threaded Software“. In: *NetFPGA Developers Workshop*. Onlinequelle, abgerufen am 27.12.2015. 2009. URL: http://www.eecg.toronto.edu/~steffan/papers/labrecque_netfpga09.pdf.

[36] LIN Konsortium. *Local Interconnect Network*. Online, abgerufen am 13.02.2013. URL: http://www.lin-subbus.org/.

[37] Chung-Wei Lin u. a. „Efficient Wire Routing and Wire Sizing for Weight Minimization of Automotive Systems“. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 34.11 (2015), S. 1730–1741. ISSN: 0278-0070. DOI: 10.1109/TCAD.2015.2448680.

[38] Philipp Meyer u. a. „Extending IEEE 802.1 AVB with Time-triggered Scheduling: A Simulation Study of the Coexistence of Synchronous and Asynchronous Traffic“. In: 2013.

[39] *MOST Specification 3.0 V2*. MOST Coorperation, Juli 2010.

[40] Kai Kurt Müller. „Time-Triggered Ethernet für eingebettete Systeme: Design, Umsetzung und Validierung einer echtzeitfähigen Netzwerkstack-Architektur“. Bachelorthesis. Hamburg: Hochschule für Angewandte Wissenschaften Hamburg, 2011. URL: http://core-researchgroup.de/bib/eigene/m-ttees-11.pdf.

[41] Euro NCAP. *Euro NCAP 2020 Roadmap*. Techn. Ber. Online Quelle, abgerufen am 12.11.15. 2015. URL: http://euroncap.blob.core.windows.net/media/16472/euro-ncap-2020-roadmap-rev1-march-2015.pdf.

[42] *OPEN Alliance Special Interest Group*. URL: http://www.opensig.org.

[43] Kathy Pretz. „Fewer Wires, Lighter Cars“. In: *The Institute* (2013). Onlinequelle, abgerufen am 08.11.2015. URL: http://theinstitute.ieee.org/benefits/standards/fewer-wires-lighter-cars.

[44] *Reduced Gigabit Media Independent Interface (RGMII)*. Abgerufen am 05.10.2015. Apr. 2002. URL: http://www.hp.com/rnd/pdfs/RGMIIv2_0_final_hp.pdf.

[45] Johannes Reidl. „Optimierung der Zeitpräzision eines Linux Ethernet Treibers auf Basis einer PTP Uhr“. Bachelorthesis. Hamburg: Hochschule für Angewandte Wissenschaften Hamburg, 2013. URL: http://core-researchgroup.de/bib/eigene/r-ozlet-13.pdf.

[46] *Road vehicles – FlexRay communications system – Part 1: General information and use case definition*. ISO 17458-1:2013. Internationale Organisation für Normung, 2013.

[47] Soeren Rumpf. „Ein ressourcenschonender Stack zur Unterstützung von AVB und Time Triggered Ethernet Verkehr“. Bachelorthesis. Hamburg: Hochschule für Angewandte Wissenschaften Hamburg, 2014. URL: http://core-researchgroup.de/bib/eigene/r-rsuat-14.pdf.

[48] Jan Jasper Salathé. *Master Projekt 1: Anforderungsanalyse für RT Ethernet Bridges*. Techn. Ber. Hochschule für Angewandte Wissenschaften Hamburg, Okt. 2014.

[49] Stefan Schneele und Fabien Geyer. „Comparison of IEEE AVB and AFDX“. In: *Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st*. 2012, 7A1–1–7A1–9. DOI: 10.1109/DASC.2012.6382405.

[50] Johannes Specht. *Urgency based Scheduler – Considerations for Low Latency Reserved Streams*. IEEE 802.1 TSN TG, Abgerufen am 07.12.2015. März 2013. URL: http://www.ieee802.org/1/files/public/docs2013/new-tsn-specht-urgency-based-scheduler-20130320.pdf.

[51] Johannes Specht und Soheil Samii. *Urgency Based Scheduler – comparison to other traffic classes + discussion on next steps*. IEEE 802 September 2015 Interim Meeting – San Jose, Abgerufen am 07.12.2015. Sep. 2015. URL: http://www.ieee802.org/1/files/public/docs2015/new-tsn-specht-ubs-comparison-and-steps-0915-v01.pdf.

[52] Johannes Specht und Soheil Samii. *Urgency Based Scheduler – IEEE 802.1 Motion Preparation*. IEEE 802 November 2015 Plenary Meeting – Dallas, Abgerufen am 07.12.2015. Nov. 2015. URL: http://www.ieee802.org/1/files/public/docs2015/new-ubs-specht-csd-par-motion-supporting-slides-v01.pdf.

[53] Johannes Specht und Soheil Samii. *Urgency Based Scheduler Scalability – How many Queues?!* IEEE 802.1 Time Sensitive Networks, Abgerufen am 07.12.2015. Mai 2015. URL: http://www.ieee802.org/1/files/public/docs2015/new-tsn-specht-ubs-queues-0521-v0.pdf.

[54] Till Steinbach u. a. „Beware of the Hidden! How Cross-traffic Affects Quality Assurances of Competing Real-time Ethernet Standards for In-Car Communication“. In: *2015 IEEE Conference on Local Computer Networks (LCN)*. Okt. 2015, S. 268–276. ISBN: 978-1-4673-6770-7.

[55] Till Steinbach u. a. „Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)“. In: (Sep. 2012). DOI: 10.1109/VTCFall.2012.6398932.

[56] Andrew S. Tanenbaum. *Modern Operating Systems (3. ed.)* Pearson Education, 2009. ISBN: 978-0-13-813459-4.

[57] Michael Johas Teener. *A Time-Sensitive Networking Primer: Putting It All Together*. Präsentation auf ISPCS'15, abgerufen am 30.11.2015. Okt. 2015. URL: https://drive.google.com/file/d/0B6Xurc4m_PVsZ1lzWWoxS0pTNVE/view?usp=sharing.

[58] Michael Johas Teener. *Back to the future: using TAS and preemption for deterministic distributed delays*. IEEE 802.1 Time-Sensitive Networking TG, Abgerufen am 07.12.2015. Nov. 2012. URL: http://www.ieee802.org/1/files/public/docs2012/new-avb-mjt-back-to-the-future-1112-v01.pdf.

[59] Michael Johas Teener. *Peristaltic Shaper: updates, multiple speeds*. IEEE 802.1 Time-Sensitive Networking TG, Abgerufen am 07.12.2015. Jan. 2014. URL: http://www.ieee802.org/1/files/public/docs2014/new-tsn-mjt-peristaltic-shaper-0114.pdf.

[60] S. Thangamuthu u. a. „Analysis of Ethernet-switch traffic shapers for in-vehicle networking applications“. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*. 2015, S. 55–60.

[61] *Time-Triggered Ethernet*. Society of Automotive Engineers, 2011. URL: http://standards.sae.org/as6802/.

[62] *TTEthernet Specification*. TTTech, 2008.

[63] Heinz Wörn und Uwe Brinkschulte. *Echtzeitprogrammierung*. German. Springer Berlin Heidelberg, 2005, S. 317–342. ISBN: 978-3-540-20588-3. DOI: 10.1007/3-540-27416-2_5.

[64] *X.200 : Information technology – Open Systems Interconnection – Basic Reference Model: The basic model*. ISO, Juli 1994. URL: http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=2820.

[65] Xilinx Inc. *Virtex-II Pro and Virtex-II Pro X FPGA User Guide*. v4.2. 2007.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 05. Februar 2016 Jan Jasper Salathé