Masterarbeit

Friedrich Groß

Eine skalierbare Hardwarearchitektur zur Echtzeit-Erweiterungen von Standard-Ethernet-Controllern

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Friedrich Groß

Eine skalierbare Hardwarearchitektur zur Echtzeit-Erweiterungen von Standard-Ethernet-Controllern

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Abgegeben am 02.01.2015

Friedrich Groß

Thema der Masterarbeit

Eine skalierbare Hardwarearchitektur zur Echtzeit-Erweiterungen von Standard-Ethernet-Controllern

Stichworte

Time-Triggered Ethernet, Hardware- Software Co-Design, Co-Prozessor, Echtzeit

Kurzzusammenfassung

In Automobil werden immer mehr Fahrerassistenzsysteme verwendet, die hohe zeitliche Anforderungen an Netzwerke stellen und dabei gleichzeitig große Datenmengen übertragen müssen. Die heutzutage eingesetzten BUS-Systeme kommen momentan an ihre Grenzen. Aus diesem Grund ist die Automobilindustrie auf der Suche nach einem neuen BUS-System, dass die steigenden Anforderungen erfüllt. Das zukünftige BUS-System wird höchstwahrscheinlich ein echtzeitfähiges Ethernet sein, da Ethernet in seiner Bandbreite skalierbar ist. Time-Triggered Ethernet ist eine Echtzeiterweiterung des Standard Ethernet. Die Umsetzung dieses Protokolls in Software erfordert einen erheblichen Entwicklungsaufwand und viel Rechenleistung. Diese Arbeit zeigt ein skalierbares Hardware- Software Co-Design für einen Time-Triggered Ethernet Controller, das die benötigte Rechenleisung reduziert. Es wird eine Prototyp-Implementierung auf einem FPGA gezeigt. Dabei wird für jedes Modul gezeigt, welche Auswirkungen es hat, dieses in Hardware oder in Software zu partitionieren. Mit dem vorgestellten Konzept ist es möglich, für jede Anwendung eine kostenoptimierte Hardware-Software Partitionierung zu finden.

Title of the paper

A scalable hardware architecture for real-time extensions of standard Ethernet controllers

Keywords

Time-Triggered Ethernet, Hardware- Software Co-Design, coprocessor, real-time

Abstract

In an automobile more and more driver assistance systems are used, that have high remporal requirements and need to transfer large amounts of data. The bus systems currently used in cars are come to their limits. For this reason, the automotive industry search for a new bus system that satisfy the demands of the new automotive application. The future bus system will most likely be a real-time Ethernet, because the bandwidth of Ethernet is scalable. Time-Triggered Ethernet is a real-time extension of standard Ethernet. The implementation of this protocol in software requires a considerable development effort and a lot of processing power. This work presents a scalable hardware software co-design for a Time-Triggered Ethernet controller, witch reduce the required processing power. A prototype implementation on a FPGA is shown. It is shown for each module what effect it has this partitioning in hardware or in software. With the presented concept it is possible to finde a cost-optimized hardware- software partitioning for each application.

Inhaltsverzeichnis

Kapitel 1

Einführung

Die Komplexität elektronischer Komponenten in modernen Fahrzeugen ist seit der Erfindung des Automobils stetig gestiegen. Zu Beginn wurden die Steuergeräte Punkt zu Punkt miteinander verbunden. Musste ein Steuergerät A mit einem Steuergerät B kommunizieren, so wurde eine Leitung zwischen A und B verlegt. Mit steigender Komplexität entwickelten sich riesige Kabelbäume im Fahrzeug, die einen erheblichen Anteil am Gewicht und Kosten des Fahrzeugs einnahmen. Um dieses Problem zu lösen, wurden BUS-Systeme eingesetzt. Mit BUS-Systemen können mehrere Steuergeräte auf einer Leitung kommunizieren. Dadurch wurde die Anzahl der Leitungen im Fahrzeug erheblich reduziert, was auch zu geringeren Kosten und einem geringerem Gewicht führte.

In den vergangenen Jahren sind immer mehr Fahrerassistenzsysteme im Fahrzeug zum Einsatz gekommen, die die Anforderung haben, große Datenmengen zu kommunizieren, die teilweise in Echtzeit übertragen werden müssen. Dazu gehören Bild- und Laserdaten. So müssen z. B. die Bilddaten der Rückfahrkamera mit geringer Latenz auf dem Bildschirm im Cockpit angezeigt werden. Außer den eben beschriebenen Anforderungen an das BUS-System, steigt auch die Komplexität der Netzwerke im Auto. Abbildung 1.1 zeigt den Wachstum der durchschnittlichen Anzahl der Knotenpunkte (Steuergeräte) im Fahrzeug. Heutzutage haben moderne Fahrzeuge über 70 Steuergeräte (vgl. Saad, 2003)[S. 03] (Abbildung 1.2 stellt dies schematisch dar), die mit über 2500 Nachrichtentypen miteinander kommunizieren (vgl. Schäuffele und Zurawka, 2013; Marscholik und Subke, 2007; Navet u. a., 2005).

Aktuell werden im Fahrzeug bis zu vier verschiedene Arten von BUS-Systemen verwendet. Diese kommen momentan an ihre Grenzen und werden in Zukunft den steigenden Anforderungen nicht gerecht werden. Aus diesem Grund ist die Automobilindustrie auf der Suche nach einem neuen Kommunikationsmedium, dass langfristig den steigenden Anforderungen gerecht wird. Ein Umstieg auf ein echtzeitfähiges Ethernet ist sehr wahrscheinlich, da Ethernet in der Bandbreite skalierbar ist und Standard Ethernet in anderen Gebieten, wie z. B.

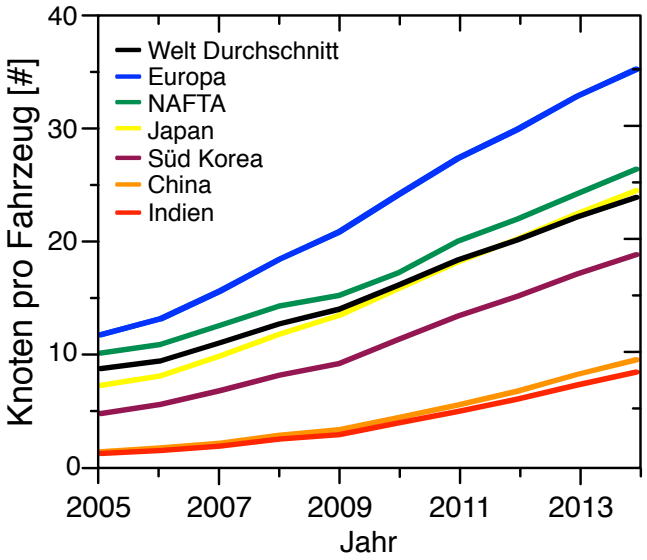

Abbildung 1.1: Durchschnittliche Anzahl Kommunikationsknoten pro Fahrzeug — Internationaler Vergleich (Quelle: Bruckmeier, 2010)

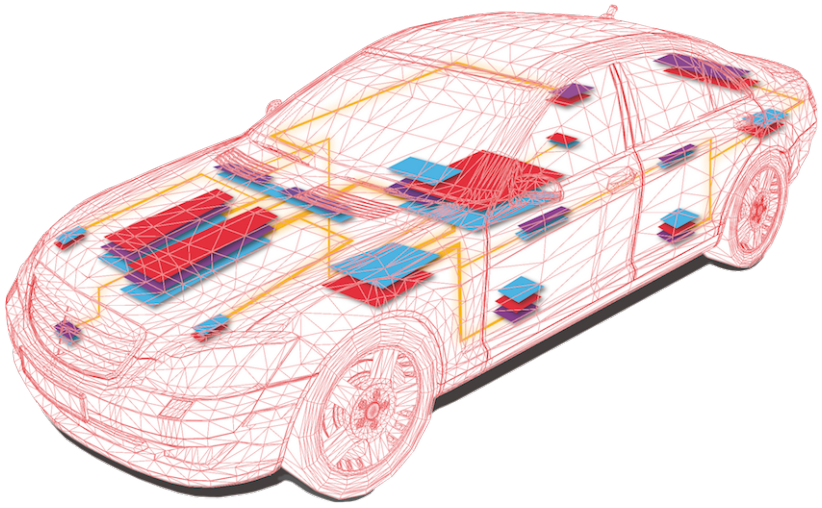

Abbildung 1.2: Elektronische Komponenten und deren Vernetzung in einem modernen Oberklassefahrzeug (Quelle: CoRE-Arbeitsgruppe)

dem Internet, bereits verbreitet ist.

Es gibt mittlerweile viele Protokolle mit verschiedenen Ansätzen, die Ethernet echtzeitfähig machen. Time-Triggered Ethernet gilt dabei als ein Kandidat für den Einsatz im Automobil.

In verschiedenen Arbeiten (vgl. Steinbach u. a., 2012; Alderisi u. a., 2012) wurde in Simulationen gezeigt, dass unter realistischen Bedingungen der Einsatz möglich ist.

In der Arbeit *A Real-time Ethernet Prototype Platform for Automotive Applications* (vgl. Müller u. a., 2011) wurde eine Softwareimplementierung des Time-Triggered Ethernet Protokolls AS6802 (vgl. SAE AS-2D Committee, 2011) auf einem Mikrocontroller gezeigt. Ein Ergebnis dieser Arbeit ist, dass bei einer hohen Netzwerklast das Time-Triggered Ethernet Protokoll alleine bis zu 90% der dort verwendeten CPU benötigt. Aus diesem Grund kann eine rechenaufwendige Anwendung nur bei einer geringen Netzwerklast auf dem Mikrocontroller ausgeführt werden. Eine geringe Netzwerklast kann jedoch nie garantiert werden, da Ethernet ein geteiltes Kommunikationsmedium ist. So könnte beispielsweise ein defektes Gerät in einen Zustand gelangen, in dem es permanent Nachrichten an alle Netzwerkteilnehmer versendet und somit die Ausführung von Anwendungen behindert.

In dieser Arbeit geht es darum, einen Co-Prozessor zu entwickeln, der die CPU bei der Protokollausführung unterstützt und so für eine geringere Belastung der CPU sorgt.

1.1 Ziel der Arbeit

Ziel der Arbeit ist es ein skalierbares Hardware- Software Co-Design zu entwickeln, dass Time-Triggered Ethernet Kommunikation ermöglicht. Um dies zu Ermöglichen muss die Protokollausführung in mehrere Module aufgeteilt werden, so dass je Modul einzeln entschieden werden kann, ob dies in Hardware oder Software partitioniert wird. Skalierbar bedeute hier, dass die Module je nach Anforderungen in Software, in Hardware oder gar nicht partitioniert werden. Mit der Skalierbarkeit sollen dabei je nach Anforderungen die optimale Modul-Partitionierung gefunden werden können, so, dass die Kosten für Hardware optimiert werden. Bei Anwendungen die viel CPU-Ressourcen benötigen ist ein Hardware-Software Co-Design unverzichtbar, da sich in vergangenen Arbeiten gezeigt hat, dass die Protokollausführung selbst viele CPU-Ressourcen benötigt (vgl. Müller u. a., 2011).

Zur Implementierung wurde das Time-Triggered Ethernet AS6802 Protokoll verwendet, da Aufgrund vorheriger Arbeiten Expertise zu dem Protokoll vorhanden ist und Hardware, dass dieses Protokoll unterstützt, zur Bearbeitung dieser Arbeit bereitgestellt wurde. Die hier entwickelten Konzepte lassen sich auf andere Time-Triggere Ethernet Protokolle überführen, wie z. B. PROFINET IRT (vgl. PROFIBUS & PROFINET International) oder der kommende IEEE 802.1Qbv Standard (vgl. Institute of Electrical and Electronics Engineers, 2013).

1.2 Struktur der Arbeit

In Kapitel 2 auf der nächsten Seite werden Grundlagen beschrieben, die zum besseren Verständnis dieser Arbeit beitragen sollen.
In Kapitel 3 auf Seite 13 wird die Problemstellung beschrieben, die sich bei der Entwicklung eines Hardware- Software Co-Designs für echtzeitfähige Controller ergibt.
In Kapitel 4 auf Seite 18 werden verwandte Arbeiten und die in dieser Arbeit übernommenen Konzepte gezeigt.
In Kapitel 5 auf Seite 26 wird ein Konzept und eine Realisierung für eine vollständige Hardwareimplementierung des Time-Triggered Ethernet Protokolls vorgestellt.
Im Kapitel 6 auf Seite 57 werden Anforderungskriteren an ein Hardware- Software Co-Design gezeigt. Anschließend werden für jedes Modul die Auswirkungen von einer Hardware- und Softwarepartitionierung gezeigt. Anhand von realistischen Partitionierungsbeispielen wird gezeigt, wie für bestimmte Anwendungsfälle die kostenoptimierte Partitionierung gefunden werden kann.
In Kapitel 7 auf Seite 71 wird anhand der Jitter als Qualitätsmerkmal beschrieben. Es wird für jedes einzelne Modul der Jitter errechnet und an Beispielen gezeigt, wie der Jitter bezogen auf die globale Uhr ermittelt werden kann.
In Kapitel 8 auf Seite 84 werden die Ergebnisse zusammengefasst und ein Ausblick auf zukünftige Arbeiten gegeben.

Kapitel 2

Grundlagen

In diesem Kapitel werden Grundlagen vermittelt, die dem besseren Verständnis dieser Arbeit dienen sollen.

2.1 Time-Triggered Ethernet AS6802

In diesem Abschnitt wird das Time-Triggered Ethernet (TTEthernet) Protokoll beschrieben, dass im dem Standard AS6802 spezifiziert ist (vgl. SAE AS-2D Committee, 2011). Dieses Protokoll wurde von der in Zusammenarbeit von TTTech (vgl. TTTech Computertechnik AG) und Honeywell (vgl. Honeywell International). Es basiert auf den Forschungsergebnissen der TU Wien (vgl. Kopetz u. a., 2005). Zunächst wird ein Überblick über das Protokoll verschafft. Anschließend wird der Synchronisations-Client detailliert erläutert.

2.1.1 Überblick

Time-Triggered Ethernet (TTEthernet) ist eine Echtzeiterweiterung für des Standard Ethernet. Es ermöglicht u. a. das zeitgesteuerte Kommunizieren innerhalb eines Netzwerks, wobei für die zeitgesteuerten Nachrichten eine deterministische Paketlaufzeit durch das Netzwerk garantiert wird. Um dieses Verhalten zu ermöglichen, werden spezielle TTEthernet-Switches zwischen Netzwerkteilnehmern benötigt. Das TTEthernet-Protokoll unterstützt drei Nachrichtenklassen, die im Folgenden beschrieben sind:

Time-Triggered-Traffic (TT): Dies sind zeitgesteuerte Nachrichten mit einem deterministischem Zeitverhalten (konstante Paketlaufzeit mit niedrigem Jitter[1]). Zeitgesteuerte Nachrichten werden in einem offline konfigurierten Zeitplan zyklisch versendet. Diese Nachrichtenklasse wird von den Netzwerkgeräten mit höchster Priorität behandelt.

[1] Mit Jitter (deutsch: Fluktuation/Schwankung) wird in der Nachrichtentechnik die Varianz der Laufzeit von Datenpaketen bezeichnet

Rate-Constrained-Traffic (RC): Diese Nachrichten werden für weniger zeitkritische Echtzeitkommunikation verwendet. Die Echtzeitfähigkeit wird durch eine vorher festgelegte garantierte Bandbreite realisiert. Diese Nachrichtenklasse entspricht dem AFDX-Protokoll-Standard ARINC-644 (vgl. Aeronautical Radio Incorporated, 2002).

Best-Effort-Traffic (BE): Diese Nachrichtenklasse entspricht dem Standard Ethernet. Nachrichten dieser Klasse werden mit niedrigster Priorität behandelt.

Um das zeitgesteuerte Kommunizieren zu ermöglichen, ist eine globale Uhr notwendig, auf die alle Netzwerkteilnehmer synchronisiert werden müssen. Der Ablauf der Synchronisation wird nun anhand der Abbildung 2.1 beschrieben. Während der Synchronisation wird mit *protocoll control frames* (PCF) kommuniziert. Dieses Frame hat ein Feld, dass sich $pcf\_transparent\_clock$ nennt, in das wie folgt Uhrzeiten eingetragen werden. Die Synchronisation wird in zwei Schritten vollzogen. Im ersten Schritt (linke Abbildung) senden die Synchronisations-Master (SM) mit einem PCF ihre aktuelle Uhrzeit, die im $pcf\_transparent\_clock$-Feld steht, an den Compression-Master (CM). Alle unterwegs liegenden Switches addieren die durch sich selbst verursachte Verzögerung auf das $pcf\_transparent\_clock$-Feld auf. Nachdem der CM alle PCFs empfangen hat, rechnet dieser die durchschnittliche Zeit aller empfangenen PCFs aus. Das Ergebnis ist die neue globale Zeit.

Nun kommt der zweite Schritt (rechte Abbildung). Der CM sendet die neue globale Zeit an alle Sychronisations-Master und alle Synchronisations-Clients. Auch hier addieren alle auf dem Weg liegenden Switches, die durch sich selbst verursachte Verzögerung auf das $pcf\_transparent\_clock$-Feld auf. Nach dem Empfang des PCF stellen alle SMs unc SCs ihre Uhr um.

2.1.2 Synchronisations-Client

Abbildung 2.2 zeigt den zeitlichen Verlauf der Synchronisation. Dieser Zeitverlauf gilt, wenn alle SM und SC direkt an einen CM angeschlossen sind.

Zum Zeitpunkt $sm\_dispatch\_pit$ leiten die SMs die Synchronisation ein, indem eine Synchronisationsnachricht (PCF) verschickt wird. Die SCs erwarten das PCF zum Zeitpunkt $smc\_scheduled\_pit$. Diese Zeit wird mit Formel 2.1 berechnet.

$$smc\_scheduled\_pit = 2 * max\_transmission\_delay + compression\_master\_delay \quad (2.1)$$

Der Wert $max\_transmission\_delay$ ist die maximale Verzögerung aller SMs, um das PCF zu versenden. Es beinhaltet die statische und dynamische Verzögerungen, die beim Senden entsteht und die Verzögerungen, die auf der Sendeleitung entsteht. Der Wert wird mit dem

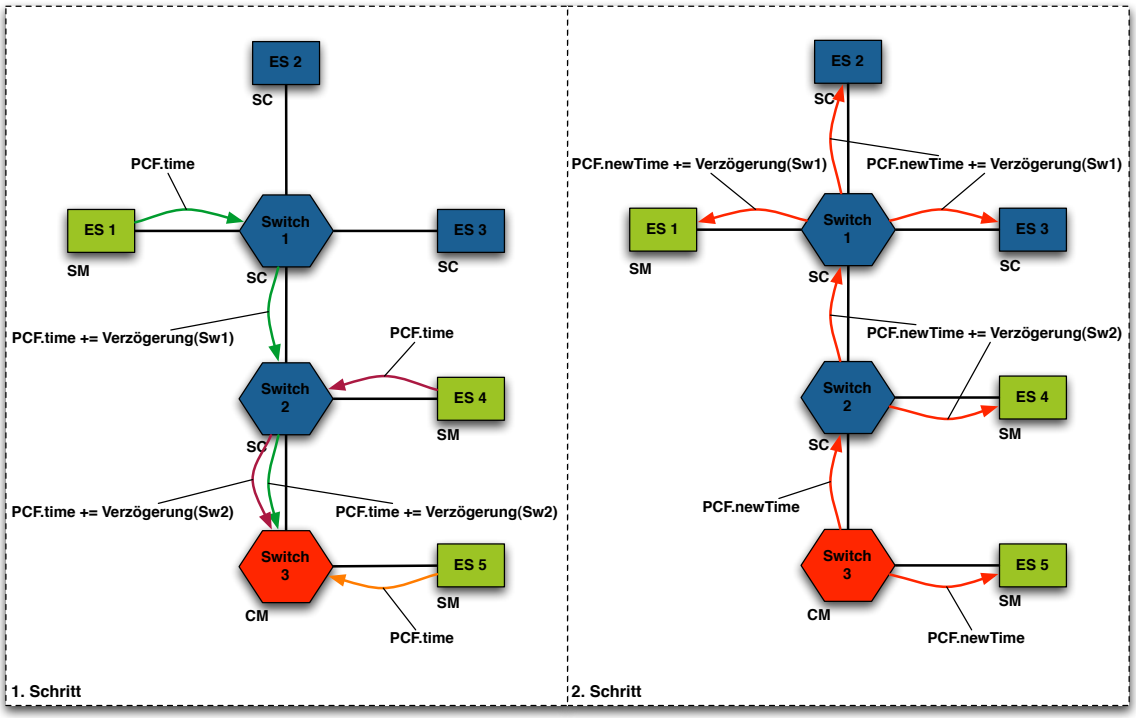

Abbildung 2.1: Beispiel: Synchronisation im TTEthernet (Quelle: Bartols, 2010)

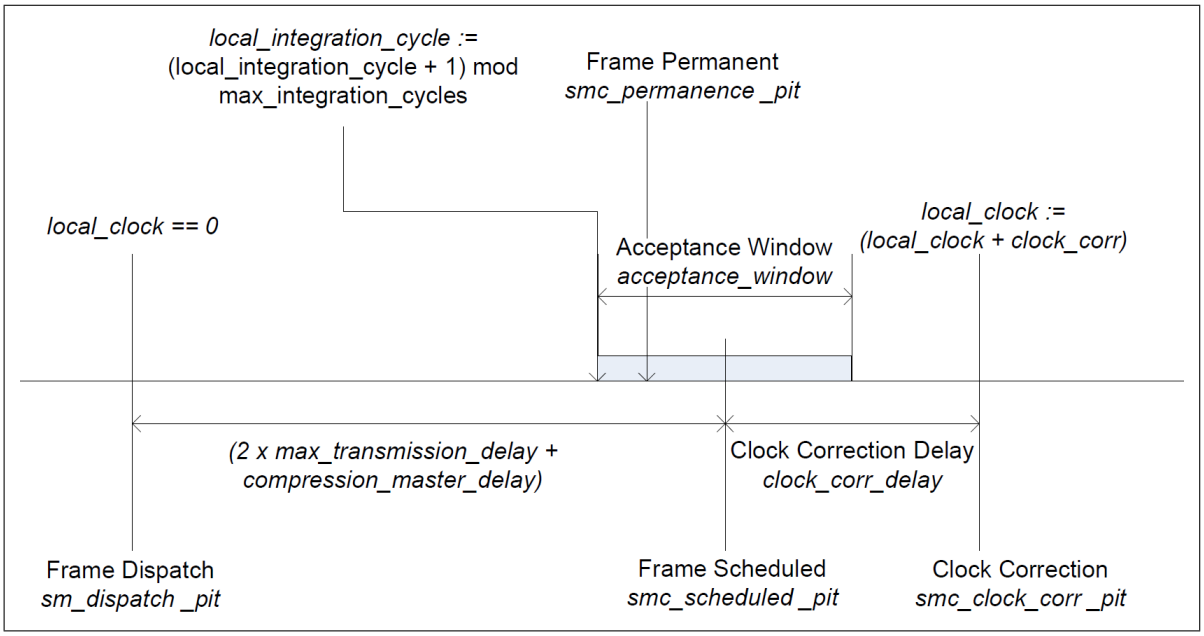

Abbildung 2.2: AS6802 Zeitverlauf (Quelle: SAE AS-2D Committee, 2011, S.44)

Wert 2 multipliziert, da das PCF in diesem Fall zwei Sendewege hat. Einmal von den SMs zum CM und vom CM an die SCs und SMs. Der $compression\_master\_delay$ ist die Verzögerung, die der CM verursacht.

Um den Zeitpunkt $smc\_scheduled\_pit$ wird mittig ein Zeitfenster definiert, dass die Breite $acceptance\_window$ hat. Der Wert $acceptance\_window$ wird mit Formel 2.2 berechnet

$$acceptance\_window = 2 * precision \tag{2.2}$$

Der Wert $precision$ ist die worst-case Abweichung von zwei beliebigen Synchronisierten Uhren im Netzwerk (vgl. SAE AS-2D Committee, 2011, S.87).

Nur innerhalb des Zeitfensters $acceptance\_window$ empfangene PCFs werden für den Synchronisationsvorgang verwendet. Außerhalb und innerhalb des Zeitfensters empfangene PCFs werden für eine Clique-Detection verwendet. Das Thema Clique-Detection wird in dieser Arbeit nicht behandelt, ist aber in (SAE AS-2D Committee, 2011, S.49) beschrieben.

Die Offset-Korrektur findet zum Zeitpunkt $smc\_clock\_corr\_pit$ statt. Dieser Zeitpunkt wird mit Formel 2.3 berechnet.

$$smc\_clock\_corr\_pit = smc\_scheduled\_pit + clock\_corr\_delay \tag{2.3}$$

Für die Zeitspanne $clock\_corr\_delay$ muss Bedingung 2.4 gelten.

$$clock\_corr\_delay > acceptance\_window \tag{2.4}$$

So wird sichergestellt, dass nach einer Offset-Korrektur die lokale Uhr nicht wieder in das Zeitfensters $acceptance\_window$ fällt.

Abbildung 2.3 zeigt den Aufbau eines PCF (protocol control frame). Dies ist der Payload-Teil eines Ethernetpakets. Im Folgenden werden die Felder beschrieben:

Integration Cycle (pcf_integration_cycle): Nummer des Zyklus, in dem die PCF versendet wurde. Jeder Teilnehmer pflegt einen Zähler für den Time-Triggered Ethernet Zyklus. Dieser wird von jedem Teilnehmer vor Beginn des des Zeitfensters $acceptance\_window$ hochgezählt (siehe Abbildung 2.2). Beim Empfang eines PCF wird überprüft, ob der lokale Zähler dem Wert in dem PCF entspricht. Falls nicht, wird das PCF verworfen. Dies stellt sicher, dass fehlgeleitete bzw. zu spät ankommende PCFs nicht zur Synchronisierung verwendet werden.

Membership New (pcf_membership_new): An diesem Feld kann erkannt werden, wie viele SMs an der Synchronisation beteiligt sind. Jedem SM wird eine Bit-Position in

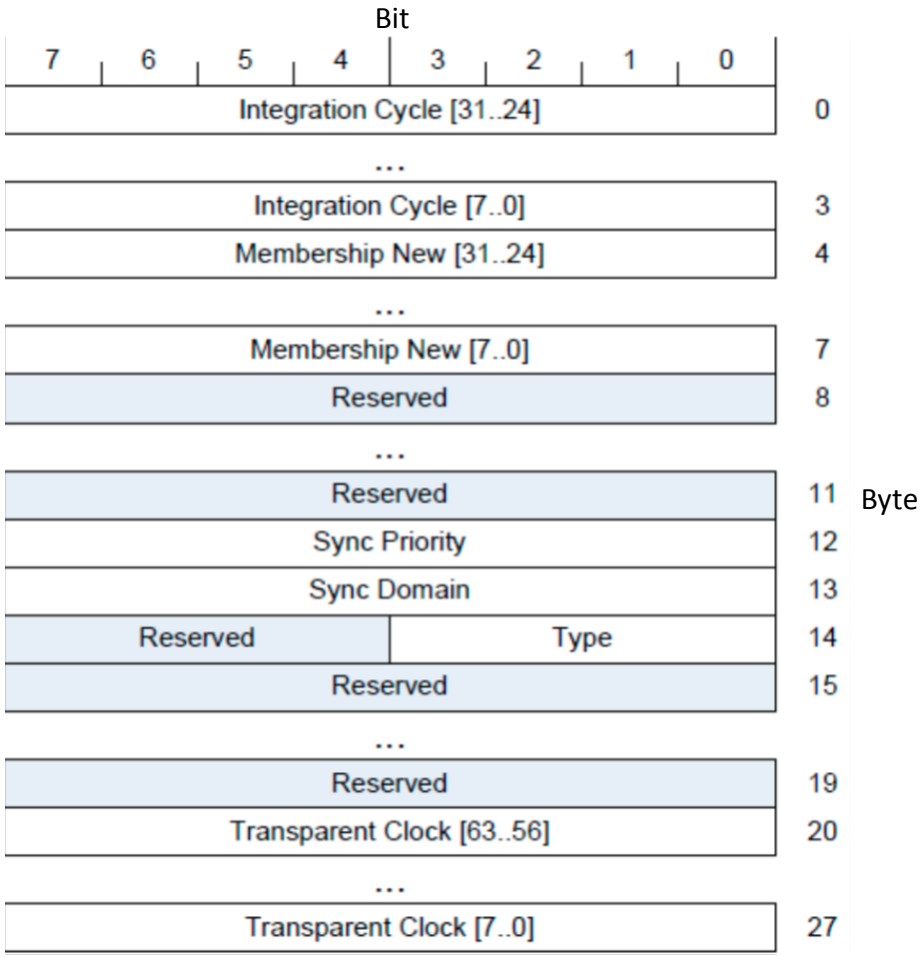

Abbildung 2.3: Aufbau PCF (protocol control frame) (Quelle: SAE AS-2D Committee, 2011, S.30)

diesem Feld zugewiesen. Der SM setzt dieses eine Bit auf 1, bevor das PCF an den CM gesendet wird. Der CM sammelt alle PCFs und erstellt ein neues. Darin sind in dem Feld $pcf\_membership\_new$ alle Bits von den SMs auf 1 gesetzt, die am Synchronisationsvorgang beteiligt sind.

Sync Priority (pcf_sync_priority): Jeder Netzwerkteilnehmer hat lokal den Parameter $local\_sync\_priority$ fest definiert. Jeder Teilnehmer akzeptiert nur PCFs, bei denen dieses Feld diesem Parameter entspricht. Dieser Parameter wird beim Clustern von Netzwerken verwendet. Dies wird in dieser Arbeit nicht behandelt, kann aber in (SAE AS-2D Committee, 2011, Kapitel 10) nachgelesen werden.

Sync Domain (pcf_sync_domain): Jeder Netzwerkteilnehmer hat lokal den Parameter $local\_sync\_domain$ fest definiert. Jeder Teilnehmer akzeptiert nur PCFs, bei dem dieses Feld diesem Parameter entspricht. Dieser Parameter wird beim Clustern von Netzwerken verwendet. Dies wird in dieser Arbeit nicht behandelt, kann aber nachgelesen werden in (SAE AS-2D Committee, 2011, Kapitel 10).

Type (pcf_type): Dieser Wert gibt den Typ der PCF Nachricht an. Es gibt drei verschiedene Typen. Für den SC ist nur der Typ *Integration Frame (IN)* relevant, daher wird hier nicht näher auf die anderen Typen eingegangen. Ein SC akzeptiert nur PCFs, die den Type IN haben.

Transparent Clock (pcf_transparent_clock): In diesem Feld steht der aufsummierte Delay des PCF vom Zeitpunkt 0 des Time-Triggered Ethernet Zykluses. Der SM schreibt hier den Delay, den er bis zum Absenden gebraucht hat. Der CM addiert auf diesen Wert den Delay, der durch den CM entstanden ist + den Delay, der auf der Empfangsleitung entstanden ist. Der SC addiert den Delay, der zwischen der Deserialisierung des ersten Bits und dem Zeitpunkt, an dem ein Eingangszeitstempel gesetzt wird + den Delay, der auf der Empfangsleitung entstanden ist. Ziel ist es, dass bei jedem Teilnehmer zum Zeitpunkt der Deserialisierung die korrekte globale Uhrzeit in diesem Feld steht.

Beim Empfang werden folgende Zwischenergebnisse ausgerechnet, die später verwendet werden. Der Wert $receive\_pit$ ist der Eingangszeitstempel.

$$permanence\_delay = max\_transmission\_delay - pcf\_transparent\_clock \tag{2.5}$$

$$permanence\_pit = receive\_pit + permanence\_delay \tag{2.6}$$

Laut Spezifikation kann ein Synchronisations-Client über mehrere Ethernetports an das Netzwerk angeschlossen werden. Auch sind laut Spezifikation Situationen möglich, in denen ein SC mehrere PCFs innerhalb des Zeitfensters $acceptance\_window$ empfangen kann. Dies entsteht insbesondere, wenn das Netzwerk aus Fehlertoleranzgründen redundant ausgelegt wird. Aus dem Grund werden in der Spezifikation Regelungen getroffen, wie mit mehreren PCFs umgegangen wird. Für jeden Channel (Ethernetport) wird das beste PCF verwendet ($smc\_best\_pcf\_channel$). Dies ist wie folgt definiert:

$$smc\_best\_pcf_{channel} = \max_{smc\_permanence\_pit} \left(\max_{pcf\_membership\_new} \left(PCF_{channel}^{in-schedule} \right) \right) \tag{2.7}$$

$PCF_{channel}^{in-schedule}$: Menge aller PCFs, die innerhalb des Zeitfensters $aceptance\_window$ eingetroffen sind

max pcf_membership_new: Wähle das PCF mit den meisten 1-Bits im Feld $pcf\_membership\_new$

max permanence_pit: Wähle das PCF, bei dem der ausgerechnete Wert $permanence\_pit$ am höchsten ist. Also das PCF, das zuerst ankam.

Jeder Channel (Ethernetport) wählt mit der eben genannten Formel den besten PCF aus. Nun wird mit Formel 2.8 der Offset-Korrekturwert berechnet.

$$clock\_corr = median/average_{channels}(smc\_best\_pcf.permanence\_pit - smc\_scheduled\_pit) \tag{2.8}$$

Für jeden Channel wird die Differenz des errechneten Wertes $permanence\_pit$ und des konfigurierten Wertes $smc\_scheduled\_pit$ berechnet. Aus diesen Werten kann dann entweder der Durchschnitt oder der Median genommen werden. Zum Zeitpunkt $smc\_clock\_corr\_pit$ kann die lokale Uhr $local\_clock$ dann um den errechneten Wert $clock\_corr$ korrigiert werden.

2.2 LocalLink Interface

Das LocalLink Interface (vgl. Xilinx Inc., 2005) ist eine schnelle unidirektionale 32/64-Bit 1-zu-1 Verbindung. Dieser Verbindung ist darauf ausgelegt, Pakete mit einem Header und einem Footer zu transportieren. Der Kontrollmechanismus ist einfach gehalten. Es gibt viele verschiedene Varianten, die in (vgl. Xilinx Inc., 2005) nachgelesen werden können.

In Abbildung 2.4 wird die Flusskontrolle des Protokolls dargestellt. Im Folgenden sind die einzelnen Kontrollsignale beschrieben:

CLK Taktung des Busses

SOF_N (start of frame) signalisiert den Beginn eines Frames

SOP_N (start of payload) signalisiert den Beginn des Payloads

EOP_N (end of payload) signalisiert das Ende des Payloads

EOF_N (end of frame) signalisiert das Ende eines Frames

SRC_RDY_N (source ready) signalisiert die Sendebereitschaft eines Senders

DST_RDY_N (destination ready) signalisiert die Empfangsbereitschaft eines Empfängers

DATA Datenteil des Busses, 32 oder 64 Bit breit

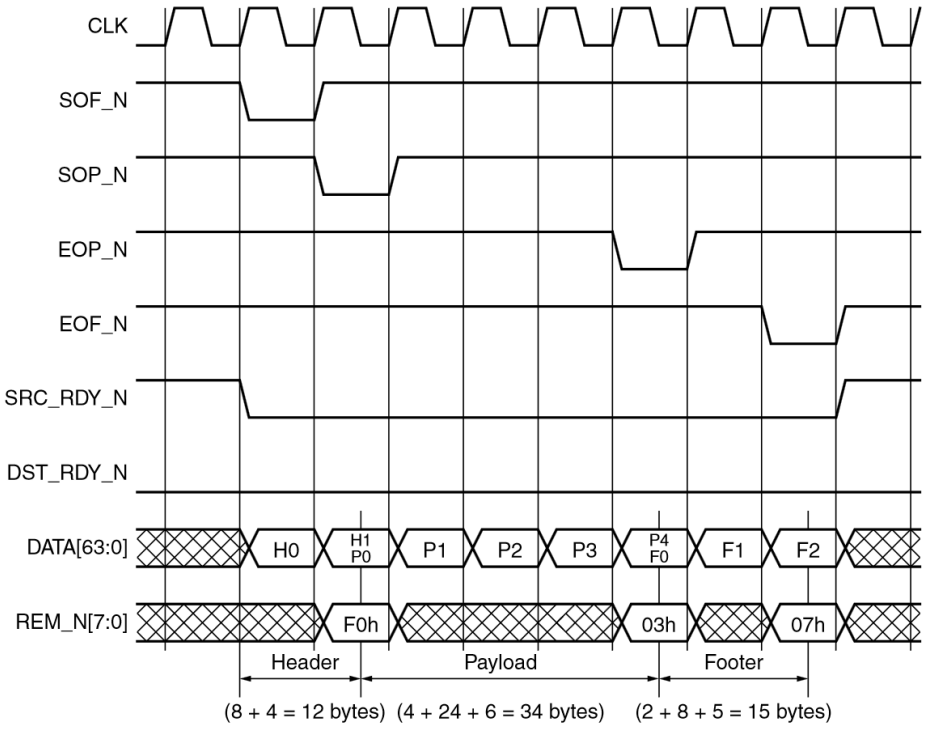

Abbildung 2.4: LocalLink Interface Flow Control (Quelle: Xilinx Inc., 2005)

REM_N (reminder) beim Wechsel von Header zu Payload, Payload zu Footer und Footer zu nicht definierten Daten, beinhaltet dieses Signal die Anzahl der Bytes, die dem vorherigen Abschnitt zugeordnet werden müssen

Kapitel 3

Problemstellung

In diesem Kapitel wird die Problemstellung erläutert. Wie in der Einleitung erwähnt, existiert eine Softwareimplementierung (vgl. Müller u. a., 2011) für das Time-Triggered Ethernet Protokoll AS6802 (vgl. SAE AS-2D Committee, 2011).

Die Softwarelösung in dieser Arbeit wurde auf dem für Netzwerkprotokolle optimierten Mikrocontroller Hilscher netX500 (vgl. Lipfert, 2008) entwickelt. Dieser Mikrocontroller hat einen ARM9 Prozessor, der mit 200 MHz getaktet ist. Bei einer hohen Netzwerklast verbraucht die Abarbeitung des Netzwerkprotokolls 90% der CPU-Zeit. Eine niedrige Netzwerklast kann nie garantiert werden, da beispielsweise defekte Geräte in einen Zustand verfallen können, in dem sie permanent Nachrichten versenden. Aus diesem Grund muss davon ausgegangen werden, dass für die Anwendung nur 10% der CPU-Zeit übrig bleibt, was für viele Anwendungen zu wenig ist.

Der Hauptgrund für diese hohe CPU-Belastung ist, dass Speicherplatz für eintreffende Pakete stets freigehalten werden muss. Wird dies nicht gemacht, werden eintreffende Pakete bei zu wenig Speicherplatz von der Hardware verworfen. Der Verlust von Echtzeitnachrichten würde zu einer Fehlfunktion führen und ist deswegen nicht erwünscht.

Ein weiteres Problem bei einer Softwarelösung ist, dass der Sendevorgang per Software angestoßen wird. Das bedeutet, dass die Ausführung einer Anwendung zum Sendezeitpunkt einer Time-Triggered Nachricht unterbrochen werden muss, um den Sendevorgang in Software anzustoßen. Die könnte dazu führen, dass zeitkritische Berechnungen verzögert werden. Außerdem kostet dieser Kontextwechsel extra CPU-Zeit.

Ziel dieser Arbeit ist es, einen Co-Prozessor zu entwickeln, der die CPU bei der Abarbeitung des Protokolls deutlich entlastet und somit mehr CPU-Ressourcen für eine Anwendung zur

Verfügung stehen.

Nicht in jedem Endteilnehmer werden alle Funktionen des Protokolls benötigt. Es wird Teilnehmer geben, die Time-Triggered Nachrichten nur versenden, aber nicht empfangen und umgekehrt. Oder ein Teilnehmer ist nicht für den Empfang der Nachrichtenklasse Rate-Constrained (siehe Grundlagen) bestimmt. In diesem Fall sinkt die CPU-Belastung bei einer reinen Software-Implementierung erheblich (Grund wird in Abschnitt 3.2 erläutert). Es kann auch sein, dass unterschiedliche Qualitätsanforderungen an die Teilnehmer gestellt werden. Beispielsweise könnten einem Teilnehmer nur kleine Zeitfenster für das Versenden von Time-Triggered-Nachrichten gewährt werden, was dazu führen würde, dass ein Sende-Modul in Hardware implementiert werden müsste, da man hier ein niedrigeren Jitter hat.

Aus diesem Grund macht es Sinn, ein skalierbares Hardware- Software Co-Design zu entwickeln, dass je nach Anforderungen nur bestimmte Teilaufgaben in Hardware auslagert. Bei jedem Teilnehmer eine volle Hardwareimplementierung zu verwenden, würde die Kosten für die Hardware bei den meisten Teilnehmern unnötig erhöhen. Ziel ist es, einen Baukasten zu entwickeln, aus dem je nach Anforderung bestimmte Module aus Hardware zusammengesetzt und so für jedes Steuergerät die optimale Hardware- Software Partitionierung bestimmt werden kann.

3.1 Hoher CPU-Bedarf bei Verwendung aller Nachrichtenklassen

In diesem Abschnitt wird erklärt, wie der hohe CPU-Bedarf zustande kommt - speziell für den Fall, dass alle drei Nachrichtenklassen für den Empfang vorgesehen sind.

Ein Mikrocontroller hat für den Empfang von Ethernetpaketen typischerweise nur einen Empfangspuffer, in dem mehrere Pakete abgelegt werden können. Dieser Empfangspuffer wird von einer Ethernet-MAC gesteuert. Die CPU erhält nach dem Empfang ein Interruptsignal und ließt nach einer gewissen Zeit Daten aus dem Paket aus. Wenn die Daten nicht mehr benötigt werden, gibt die CPU diesen Speicher frei und die Ethernet-MAC kann an diese Position ein neues Paket ablegen.

Wenn alle Speicherbänke voll sind, verwirft die Ethernet-MAC neu empfangene Pakete. In zeitkritischen Abschnitten schalten Anwendungen die Interrupts aus, um diese ohne Unterbrechung berechnen zu können. Wenn in dieser Zeit Best-Effort Pakete einlaufen und den Speicher füllen, werden die danach eintreffenden Nachrichten verworfen. Die verworfenen Nachrichten können auch Echtzeitnachrichten sein. Da diese nicht verworfen werden dür-

fen, ergibt sich die Anforderung an die Software, dass Interrupts für den Ethernetempfang nie ausgeschaltet werden dürfen und empfangene Nachrichten schnell verarbeitet werden müssen, um den Speicher schnell wieder freizugeben. Nur so kann sichergestellt werden, dass keine empfangene Echtzeitnachricht verworfen wird.

Bei einem 100Mbit Netzwerk muss davon ausgegangen werden, dass alle 6,72µs eine Nachricht eintreffen kann. Dies ist die minimale Zykluszeit von Ethernetpaketen mit minimaler Größe. Die Software muss also in der Lage sein, Speicherplätze in diesem Zyklus wieder freizugeben. Dies ist der Grund, warum bei einer hohen Netzwerklast 90% der CPU benötigt werden.

3.2 Reduzierter CPU-Bedarf beim Weglassen der RC-Nachrichtenklasse

Als nächstes soll der Fall betrachtet werden, bei dem ein Mikrocontroller nur Time-Triggered und Best-Effort Nachrichten empfangen soll. In diesem Fall sind wesentlich geringere CPU-Ressourcen notwendig.

In diesem Fall gibt es nur Echtzeitnachrichten, die zeitgesteuert ankommen. Die weggelassenen Rate-Constrained Nachrichten sind event-basierte Echtzeitnachrichten. Im AS6802 Protokoll werden Time-Triggered Nachrichten zur Designzeit offline konfiguriert. Der Zeitpunkt des Eintreffens ist also bekannt. In diesem Fall reicht es, wenn die Software die Speicherplätze kurz vor dem Eintreffen einer Time-Triggered Nachricht freigibt. So kann die Software ihre zeitkritischen Berechnungen zu Zeitpunkten planen, in denen keine Time-Triggered Nachrichten empfangen oder versendet werden und für diese Zeit die Interrupts ausschalten.

Eintreffende Best-Effort Nachrichten können dann außerhalb zeitkritischer Berechnungen verarbeitet werden. Ein Teil dieser Nachrichten könnte während der zeitkritischen Berechnung verloren gehen, was für diese Nachrichtenklasse zulässig ist.

3.3 Versenden von Time-Triggered Nachrichten

Bei einer Softwareimplementierung muss das Versenden von Nachrichten über die Software angestoßen werden. Dies gilt auch für Time-Triggered Nachrichten. Erstellt eine Anwendung eine Time-Triggered Nachricht, wird diese zunächst in dem Sendepuffer gespeichert. Ist der Sendezeitpunkt erreicht, muss die aktuelle Berechnung unterbrochen und das Senden der Nachricht angestoßen werden. Diese Unterbrechungen kosten weitere CPU-Zeit. Mit einem

Co-Prozessor kann eine Lösung entwickeln werden, bei der die Software die zu versenden Pakete asynchron in den Sendepuffer ablegt und der Co-Prozessor den Sendevorgang zum konfigurierten Sendezeitpunkt anstößt.

Die eben beschriebene Softwarelösung (ohne Co-Prozessor) setzt voraus, dass es einen Schedulingmechanismus in Software gibt. In der Arbeit (vgl. Müller u. a., 2011) wurde so ein Mechanismus entwickelt, ohne ein Betriebssystem zu verwenden.

In der Automobielindustrie wird auf Steuergeräten oft das Betriebssystem AUTOSAR (vgl. AUTOSAR Development Cooperation) verwendet. Dieses Betriebssystem bietet in der aktuellen Version keinen Schedulingmechanismus, mit dem Aktionen zeitgesteuert ausgeführt werden können. Eine Softwareimplementierung für das Versenden von Time-Triggered Nachrichten wäre somit nur schwer zu realisieren.

Eins der aktuell verwendeten Bus-Systeme im Automobil ist FlexRay (vgl. FlexRay Consortium, 2005). Mit dem Betriebssystem AUTOSAR ist es möglich, zeitgesteuerte Nachrichten über FlexRay zu versenden. Dies wird mithilfe eines Co-Prozessors realisiert, in dem die Software asynchron die Time-Triggered Nachrichten in einen Sendepuffer ablegt und der Co-Prozessor diese zum richtigen Zeitpunkt absendet. Es hat sich also auch schon in anderen Bereichen gezeigt, dass die Verwendung von Co-Prozessoren sinnvoll ist.

3.4 Reduzierung des Migrationsaufwands

Mit einer vollständigen Hardwareimplementierung ist es möglich, den Migrationsaufwand bestehender Anwendungen, die in Automobil-Steuergeräten laufen, zu reduzieren. Beispielsweise wäre es wünschenswert bei der Migration von FlexRay nach Time-Triggered Ethernet einfach nur den Co-Prozessor auszutauschen.

Bei einer vollständigen Hardwareimplementierung sind die erwarteten Anpassungen an die Software am geringsten. Hier könnte das Ziel verfolgt werden, das Time-Triggered Ethernet Protokoll transparent für die Software im Co-Prozessor zu betreiben.

3.5 Optimierung der Energiebilanz

Im Automobil existieren einige Fahrerassistenzsysteme die in Echtzeit kommunizieren müssen, aber die meiste Zeit nicht benötigt werden. Beispielsweise werden die Einparkautomatik und die Rückfahrkamera nur während des Einparkvorgangs benötigt. Um Energie zu sparen, versucht man solche Steuergeräte so oft und lange wie möglich im Sleep-Modus zu belas-

sen. In diesem Modus werden die meisten Teile des Steuergeräts nicht mit Strom versorgt. Um Time-Triggered kommunizieren zu können, müssen die Uhren der Steuergeräte stets synchron sein. Im AS6802 Protokoll bedeutet dies, dass die CPU einmal pro Zyklus den Sleep-Modus beenden, ein empfangenes Synchronisationspaket verarbeiten und die Uhr neu stellen müsste.

Ist das Synchronisationsmodul in Hardware als Co-Prozessor implementiert, könnte dieser die Synchronisation übernehmen und die CPU müsste nicht alle paar Millisekunden aus dem Sleep-Modus geweckt werden. Dies könnte dazu beitragen, Energie zu sparen bzw. den Mehrverbrauch des Co-Prozessors zu kompensieren.

Kapitel 4

Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten aufgeführt und die daraus gewonnen Information, die in diese Arbeit verwendet werden, beschrieben.

4.1 TTEthernet Hardwareimplementierung

In der Dissertation *Design of an FPGA-Based Time-Triggered Ethernet System* (vgl. Steinhammer, 2006) wurde ein FPGA-basierte Hardwareimplementierung eines Time-Triggered Ethernet Systems vorgestellt. Daraus resultierte die Veröffentlichung *Hardware Implementation of Time-Triggered Ethernet Controller* (vgl. Steinhammer und Ademaj, 2007), in der ein Hardwareimplementierung eines Controllers gezeigt wird. Teile der Konzepte wurden von einer Hardwareimplementierung (vgl. Stöger, 1997) des Protokoll TTP/C (vgl. Elmenreich und Ipp, 2003) übernommen. Im Folgenden werden die dort definierten Module beschrieben.

4.1.1 Synchronisation

Der Synchronisationsalgorithmus wurde in dieser Hardwareimplementierung nicht mit realisiert. Es wird der Software eine Schnittstelle geboten, um eine Offset-Korrektur vornehmen zu können. Außerdem wird ein Micro/Macrotick-Timer bereitgestellt, mit dem es möglich ist, die Geschwindigkeit der lokalen Uhr an die globale Uhr anzupassen. Die Einstellung hierfür muss ebenfalls in Software vorgenommen werden. Wie dies in Software geschieht, wird in der Arbeit nicht beschrieben. Die Funktion des Micro/Macrotick-Timers wird im Folgenden beschrieben:

Microtick und Macrotick sind zwei verschiedene Timer. Die Frequenz des Microtick-Timers wird durch einen Quartz bestimmt. Der Microtick-Timer wird mit einem Defaultwert beschrieben, der von der Software zur Geschwindigkeitskorrektur verwendet wird. Mit jedem Takt wird der Wert im Microtick-Timer dekrementiert. Erreicht der Microtick-Timer den Wert

0, wird der Macrotick-Timer inkrementiert, womit die Macrotick-Frequenz bestimmt wird. Der Microtick wird anschließend mit dem Defaultwert neu beschrieben. Alle Hardware und Softwaremodule verwenden den Macrotick-Timer als lokale Uhr. Durch das verändern des Defaultwertes für den Microtick-Timer wird die Geschwindigkeit des Macrotick-Timers geregelt

Bei allen eintreffenden Ethernet-Paketen wird direkt beim Empfang ein Zeitstempel vom Macrotick-Timer genommen und dieser an die Header-Daten des Ethernet-Pakets angehängt. So kann der Synchronisationsalgorithmus, asynchron anhand des Zeitstempels und der in den Synchronisationsnachrichten enthaltenen Zeit, den Defaultwert des Microticks neu berechnen.

4.1.2 Scheduler

Das Scheduler-Modul ist für das planmäßige Versenden von TT-Nachrichten zuständig. Der Scheduler beinhaltet einen sogenannten MEDL-Speicher (message descriptor list memory). Dieser Speicher ist eine verkettete Liste, die für jede Time-Triggered Nachricht im Zyklus einen Eintrag hat. Jeder Eintrag enthält folgende Informationen:

- Absendezeitpunkt der Time-Triggered Nachricht
- Nachrichtenheader (Ethernetheader)
- Pointer auf den Datenteil, der in einem Dual-Ported-Memory liegt.

Der MEDL-Speicher ist aufsteigend nach dem Absendezeitpunkt sortiert. Da dieser Speicher eine Linked-List ist, können nachträglich Nachrichten hinzugefügt werden, die dann ohne Kopieraufwand in die Liste einsortiert werden können. Für die richtige Sortierung muss der Softwareteil sorgen, der auf einem Mikrocontroller läuft. Dieser ist in der verwandten Arbeit nicht beschrieben.

Ein separates Sendemodul überwacht diesen Speicher und sendet jeweils die Nachricht, dessen Absendezeitpunkt gleich den Wert des Macrotick-Timers ist. Die MEDL-Liste wird dabei zyklisch in sortierter Reihenfolge abgearbeitet. Abbildung 4.1 zeigt dies schematisch.

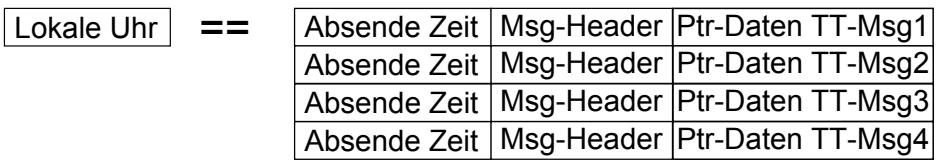

Abbildung 4.1: Ablauf Scheduler und Sendemodul

4.1.3 Bufferpool

In dieser Arbeit wird der Datenteil einer Nachricht im Dual-Ported-Memory gespeichert. Jeder MEDL-Eintrag hat einen Pointer zum Datenteil, der beim Senden immer wieder ausgelesen wird. Eine temporale Firewall verhindert, dass der Mikrocontroller und die Hardwaremodule gleichzeitig auf die gleichen Datenteile zugreifen. Diese temporale Firewall wird als Umkehrung des im Modul *Schedule* festgelegten Zeitplans realisiert und ist in dieser Arbeit nicht näher beschrieben.

4.1.4 Best-Effort

Dieses Modul ist für das Versenden von Nachrichten der Nachrichtenklasse Best-Effort zuständig. Das Best-Effort Modul wurde in dieser Arbeit wie folgt realisiert. Steht eine eventbasierte Nachricht an, so wird anhand der aktuellen Zeit und dem Zeitpunkt der nächsten zu sendenden TT-Nachricht durch eine Subtraktion ausgerechnet, wie viel Zeit bis zur nächsten TT-Nachricht bleibt. Anhand der Serialisierungszeit (bestimmt durch die Bandbreite des Ethernet) und der Paketgröße kann dann die Sendezeit für die anstehende eventbasierte Nachricht ausgerechnet werden. Ist die Sendezeit kleiner als die Zeit bis zur nächsten TT-Nachricht, so wird die eventbasierte Nachricht verschickt. Wenn nicht, wird so lange gewartet, bis eine ausreichend große Lücke da ist. Hierbei wird das Interframe-Gap (12 bytes), dass laut Ethernet-Spezifikation zwischen jeder Nachricht liegen muss, mitberücksichtigt.

4.1.5 Fazit

Diese Arbeit liefert für meine Arbeit wertvolle Informationen - zugleich ist es die einzige Hardwareimplementierung die veröffentlicht wurde. Das Konzept zum Versenden von BE-Nachrichten zwischen den TT-Nachrichten wird in meiner Arbeit weitestgehend übernommen. Der Scheduler wird in einer stärker abgewandelten Art in dieser Arbeit verwendet.

4.2 TTEthernet Softwareimplementierung

In der Arbeit *A Real-time Ethernet Prototype Platform for Automotive Applications* (vgl. Müller u. a., 2011) wurde das TTEthernet Protokoll auf dem Mikrocontrollerboard *Hilscher NXHX500-ETM* (vgl. Lipfert, 2008) realisiert. Es wurde dabei kein Betriebssystem verwendet. Dieses Board ist mit einer ARM926EJ CPU ausgestattet, die mit 200MHz getaktet wird. Alle anderen Komponenten werden mit 100MHz getaktet. Das Bord verfügt über 8MB SDRAM, 16MB Flashspeicher und diverse Schnittstellen wie CAN, USB, RS232 und 2x Ethernet. Das Besondere an diesem Board ist, dass die Ethernetschnittstellen jeweils eine Zeitstempeleinheit besitzen. Diese zeichnen die Ankunftszeit eines Ethernetframes auf 40ns

genau. Eine weitere Besonderheit ist, dass ein Timer existiert, der in seiner Geschwindigkeit eingestellt werden kann. Im Folgenden werden die dort definierten Module beschrieben.

4.2.1 Synchronisation

Der Synchronisationsalgorithmus ist eine Eigenlösung und wurde als Regelung implementiert. Diese weicht von der AS6802 Spezifikation (vgl. SAE AS-2D Committee, 2011) ab und wird hier deswegen nicht weiter erläutert. Jedoch ist der Timer, mit dem die Zeit geregelt wird, für meine Arbeit interessant.
Für die Regelung wurde ein auf dem Mikrocontrollerboard befindlicher Hardware-Timer verwendet. Dieser wird als Festkommazahl organisiert. Der Timer besteht aus einem 32-Bit Register und einem 24-Bit Register. Die 32 Bit stehen vor dem Komma, die 24 Bit nach dem Komma. Ein weiteres Register, das ADDUP genannt wird, bestimmt den Wert, der in jedem Takt auf die Festkommazahl addiert wird. Im Startzustand steht im ADDUP-Register der Wert 10.0 (Dezimal). Soll die Geschwindigkeit des Timers bspw. um 1% erhöht werden, so wird der Wert10.1 in das ADDUP-Register geschrieben.

4.2.2 Scheduler

Das Scheduler-Modul wird für das Versenden von TT-Nachrichten verwendet. Für das Scheduling wird in der Startup-Phase eine Liste mit den Time-Triggered Nachrichten und deren Absendezeit festgelegt. Diese Liste wird vom Scheduling-Modul nach Absendezeit sortiert. Für die erste Nachricht wird ein Event an den Synchronisations-Timer gebunden. Dieser Mechanismus wird von der Mikrocontrollerarchitektur bereitgestellt. Dabei wird die Eventzeit mit der Zeit des Festkomma-Timers verglichen. Wenn diese gleich ist, wird ein Interrupt ausgelöst. In der ISR wird dann die Nachricht verschickt. In der Implementierung wird ein Timer verwendet, der nicht den gleichen Zyklus hat, wie der Time-Triggered Ethernet Zyklus. Daher muss in der eben genannten ISR noch der Versendezeitpunkt für die nächste TT Nachricht berechnet werden. Dazu wird die Zeitdifferenz der aktuellen Uhrzeit und Absendezeit der nächsten TT-Nachricht berechnet und auf die aktuelle Uhrzeit addiert. Dieser Wert ergibt den Timerwert, auf den das nächste Timer-Interrupt-Event gebunden wird.

4.2.3 Frame-Dropping

Das Frame-Dropping-Modul ist für das Aufräumen des Empfangspuffers zuständig. Es wird bei jeder empfangenen Nachricht die Priorität überprüft. Übersteigt der Speicherfüllstand ein bestimmtes Level, so werden nieder priorisierte Nachrichten verworfen. Bei jeder empfangenen Nachricht muss hier die CPU aktiv werden. Bei einer hohen Netzwerklast braucht dieses Modul die meiste CPU-Zeit.

4.2.4 Best Effort

Das Versenden von Best Effort Nachrichten wurde wie folgt realisiert. Nach jedem Sendevorgang von Time-Triggered Nachrichten wird geprüft, ob in der Zeit bis zur nächsten Time-Triggered Nachricht ein Fullsize-Frame (1540 Byte) verschickt werden kann. Ist dies der Fall, so wird eine Best-Effort Nachricht verschickt. Wenn nicht, wird beim Versenden der nächsten Time-Triggered Nachricht erneut geprüft.

4.2.5 Fazit

Diese Arbeit zeigt, welche Module besonders viel CPU-Ressourcen brauchen. Dies zeigt meiner Arbeit, bei welchen Modulen die Hardwareimplementierung die CPU am meisten entlasten kann. Außerdem wird das Konzept des Festkomma-Timers in meiner Arbeit für die Anpassung der Geschwindigkeit der lokalen Uhr verwendet.

4.3 PROFINET IRT Hardwareimplementierung

Profinet IRT (vgl. PROFIBUS & PROFINET International) ist ein Echtzeit Ethernet Protokoll, welches ähnlich dem TTEthernet zeitgesteuert funktioniert. In diesem Protokoll ist eine künstlich geschaffene Grenze von 250 Microsekunden für die kürzeste Zykluszeit festgeschrieben. Dies ist aus der Anforderung entstanden, dass ein Fullsize Ethernetframe, zusätzlich zum Protokolloverhead, innerhalb eines Zykluses übertragen werden kann.

Es gibt jedoch Anwendungsfälle, wie Motion-Control Systeme, bei denen diese Zykluszeit zu lang ist und bei denen keine Full-Size Frames übertragen werden. In diesen Fällen wäre es nützlich, vom Protokoll abzuweichen und eine kürzere Zykluszeit im Netzwerk zu verwenden. Die Arbeit *Optimising PROFINET IRT for Fast Cycle Times* der ZHAW Zürich (vgl. Gunzinger u. a., 2010) beschreibt eine FPGA Implementierung des PROFINET IRT Protokolls, mit der Zykluszeiten von 31,25 Microsekunden möglich sind.

Das PROFINET IRT Protokoll wird mit dem *Precision Time Protokoll* synchronisiert, welches im IEEE1588 Standard (vgl. Institute of Electrical and Electronics Engineers, 2002) definiert ist. Um eine hohe Synchronisationsgenauigkeit zu erhalten, muss in erster Linie die Zeitstempeleinheit genau arbeiten. Aus diesem Grund wurde in dieser Arbeit ein Hardwarebaustein entwickelt, der direkt nach dem Erkennen des *Start of Frame delimiter* (das Byte direkt nach der Präambel), einen Zeitstempel aufzeichnet. Die Zeit wird von einem Timer aufgezeichnet, dessen Geschwindigkeit sich von einem Synchronisationsalgorithmus einstellen lässt. Der Synchronisationsalgorithmus wird in einem *Offset/Drift Control Block* ausgeführt. Dieser korrigiert durch eine Offset-Korrektur und eine Geschwindigkeitsan-

passung die lokale Uhr. Die Funktionalität des *Offset/Drift Control Block* ist nicht näher beschrieben.

Die Eingangszeitstempel werden auch dazu verwendet, die Pünktlichkeit von eingehenden Paketen festzustellen. Kommt eine Echtzeitnachricht zu spät an, so wird diese von dem Hardwaremodul verworfen.

Der komplette PROFINET IRT Block mit zwei MAC-Einheiten, einer Zeitstempeleinheit und einer Hardware Zeitsynchronisation wurde auf einem Altera Cyclon III Board synthetisiert. Auf diesem Board stehen 120.000 *logic elements* (LE) zur Verfügung. Verbraucht wurden davon weniger als 25000. Der Synchronisationsmechanismus verwendet davon 9000 LE's und die Zeitstempeleinheit 2960 LE's.

Das gesamte Modul erreicht beim Versenden von Nachrichten einen Jitter von 6ns und braucht 1,4 Microsekunden, um ein Paket weiterzuleiten (Cut-Through Technologie). Die Synchronisationsgenauigkeit beträgt <50ns. Der Jitter beim Versenden von Nachrichten wurde hier kleiner angegeben, als die Synchronisationsgenauigkeit. Deswegen ist hier davon auszugehen, dass sich der angegebene Jitter nur auf die lokale Uhr bezieht.

4.3.1 Fazit

In dieser Arbeit wird die Hardwareimplementierung des PROFINET IRT Protokolls beschrieben. Die Synchronisierung hat ähnliche Anforderungen bezüglich des Timers, wie beim Time-Triggered Ethernet. Andere Teile des Protokolls unterscheiden sich jedoch deutlicher und wurden hier nicht weiter beleuchtet. Der Aufbau der Zeitstempeleinheit wird in dieser Arbeit am besten beschrieben und wird für meiner Arbeit eine Hilfe sein.

4.4 IEEE 1588 Hardwareimplementierung

In der Arbeit *High precision clock synchronization according to IEEE 1588 implementation and performance issues* (vgl Weibel, 2005) werden Anwendungsbereiche des IEEE 1588 Protokolls vorgestellt, das Protokoll an sich erklärt und eine Hardwareimplementierung gezeigt. Dieser Abschnitt konzentriert sich auf die Hardwareimplementierung.
In dieser Arbeit wird erklärt, dass die Synchronisationsgenauigkeit direkt von der Genauigkeit der Ankunftszeitstempel abhängt. Es werden verschiedene Wege gezeigt Zeitstempel aufzuzeichnen, wobei sich herausstellt, dass die höchste Genauigkeit erreicht wird, wenn die Zeitstempel an der MII-Schnittstelle (Media indipendent Interface) aufgezeichnet wird - der Schnittstelle zwischen OSI-Layer 1 und 2. In der Arbeit wird ein *IEEE 1588 Evaluation Kit for*

Ordinary Clocks vorgestellt, dass ein spezielles Timing analyzer Board enthält (siehe Abbildung 4.2). Dieses Board enthält zwei Zeitstempeleinheiten und eine einstellbare Uhr (Offset und Drift). Mit den zwei Zeitstempeleinheiten ist es möglich den physikalischen Empfangs- und Sendezeitpunkt von Paketen aufzuzeichnen. Die Kommunikation mit dem Rechner erfolgt über eine PC/104 Schnittstelle. So ist es möglich bei einem Rechner, der eine einfache Netzwerkkarte eingebaut hat, genaue Empfangs- und Sendezeitpunkte aufzuzeichnen und diese dann für das IEEE 1588 Synchronisationsprotokoll zu verwenden.

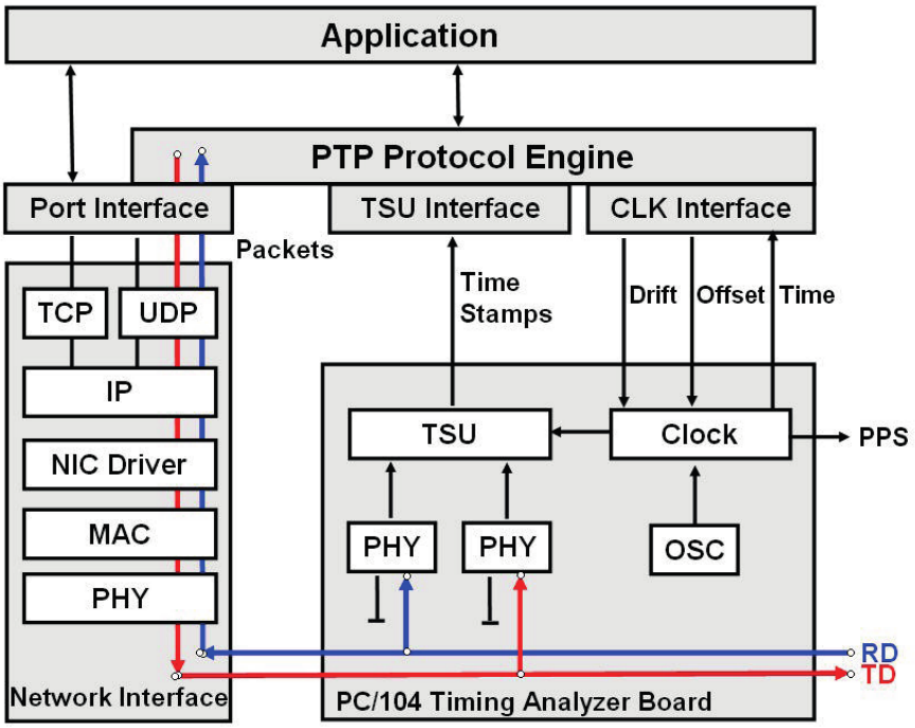

Abbildung 4.2: Aufbau - IEEE 1588 Evaluation Kit for Ordinary Clocks (Quelle: Weibel, 2005)

In diesem Konzept hat man bei eintreffenden Paketen zwei Pfade. Einen, der über die Netzwerkkarte geht und einen, der die Zeitstempel über das Timing analyzer Board an den Rechner weiter gibt. Wenn ein Pfad davon nicht die CRC Summe oder andere Ethernet Konformitäten überprüft, so können Zeitstempel mit falschen Paketen assoziiert werden. In dieser Arbeit wurde nicht verraten wie dieses Problem gelöst wurde.

4.4.1 Fazit

In meiner Arbeit ist es ebenfalls notwendig Zeitstempel aufzuzeichnen. Um dies möglichst genau zu tun, verwende ich auch die Schnittstelle MII wie in der Arbeit Weibel (2005) be-

schrieben. Da bei mir die Entwicklung nicht auf einem normalen Rechner erfolgt, sondern auf einem FPGA, kann ich relativ frei Designentscheidungen treffen und somit mir eine Verbindung zwischen Ethernet-Paketpfad und dem Zeitstempelpfad schaffen. Diese Verbindung signalisiert dem Zeitstempelpfad, ob ein Ethernetpaket Ethernetkonform ist (Checksumme, Mindestlänge, Maximallänge) und ob genug Speicherkapazität für das Paket vorhanden war. Somit kann eine korrekte Zuweisung zwischen Paket und Zeitstempel gewährleistet werden.

Kapitel 5

Konzeption und Realisierung

In diesem Kapitel werden zunächst Module identifiziert, mit denen ein Hardware- Software Co-Design für einen Time-Triggered Ethernet Controller realisiert wird. Anschließend wird ein Konzept und eine Realisierung für eine vollständige Hardwareimplementierung gezeigt. Danach wird ein Konzept zur Partitionierung vorgestellt. Dabei werden die Auswirkungen der Partitionierung in Hardware oder Software der einzelnen Module diskutiert.

5.1 Module identifizieren

In diesem Abschnitt werden Hardwaremodule identifiziert und definiert, in die das Time-Triggered Ethernet Protokoll aufgeteilt werden soll. Zunächst wird von einer vollen Hardwareimplementierung ausgegangen, so dass für jede Aufgabe, zur Verarbeitung von Time-Triggered Ethernet Paketen, ein Modul definiert wird. In Abschnitt 5.2 werden diese Module genauer spezifiziert. Abbildung 5.1 zeigt ein UML-Komponentendiagramm des Systems. Die grün markierten Module werden in diesem Abschnitt definiert. Die weiß markierten Module sind Standardmodule.

Das Protokoll benötigt für eintreffende Synchronisationspakete (protocol control frames) und Time-Triggered Pakete (TT-Frame) die Ankunftszeit. Für die Synchronisationspakete wird dies benötigt, um die neue Uhrzeit berechnen zu können. Hier fließt ein paketinterner Zeitstempel und der Empfangszeitpunkt in die Berechnung mit ein. Für Time-Triggered Pakete wird dies benötigt, um festzustellen, ob die die Nachricht rechtzeitig eingetroffen ist. Dies ist wichtig, um festzustellen, ob die Echtzeitbedingung der Nachricht erfüllt wurde. Daraus ergibt sich das Modul **Timestamping**.

Eintreffende Pakete sollen in verschiedene Puffer oder Hardwaremodule sortiert werden können, bevor diese von der Software verarbeitet werden. Dabei ist es wichtig, BE-Nachrichten und TT-Nachrichten in jeweils eigene Puffer zu sortieren. Bei einer Softwarelösung haben

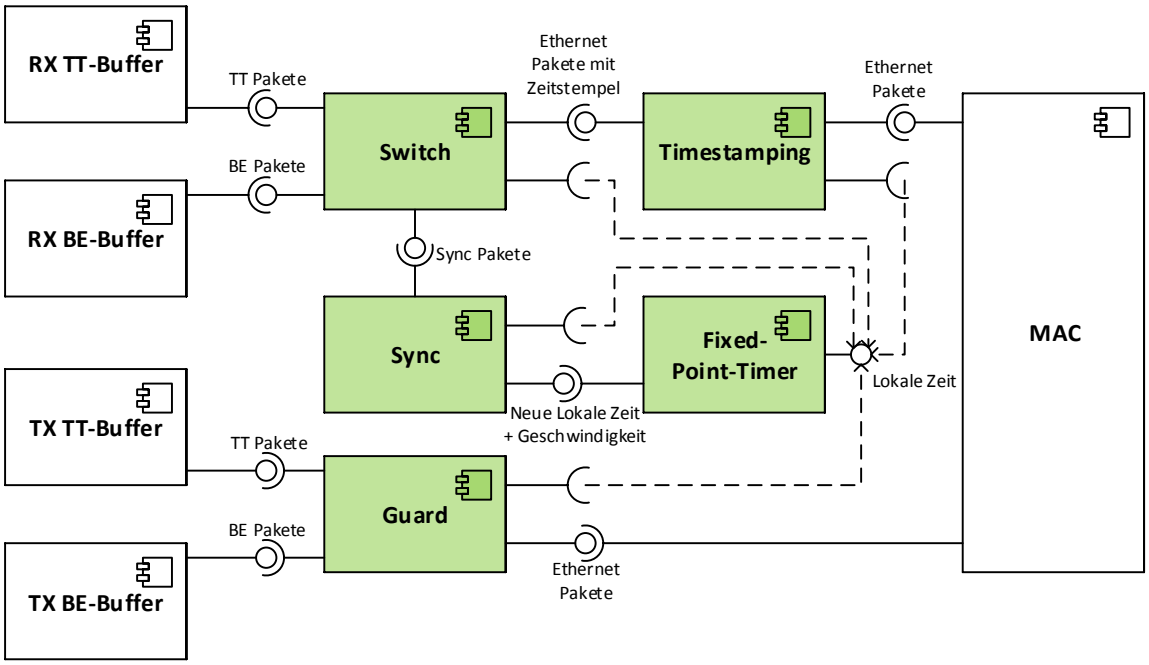

Abbildung 5.1: Komponentendiagramm Time-Triggered Ethernet Hardwareimplementierung

beide Nachrichtenklassen den gleichen Empfangspuffer. Dabei ist es möglich, dass schnell hintereinander eintreffende BE-Nachrichten den Eingangspuffer füllen und somit eintreffende TT-Nachrichten wegen zu wenig Speicher verworfen werden. Durch die Trennung der Puffer ist dies nicht mehr möglich.

Zudem ergibt sich der Vorteil, dass die verschiedenen Puffer unterschiedliche Interrupts haben können. Somit können für verschiedene Nachrichtenklassen die Interrupts zeitweise ausgeschaltet werden. Bleibt der Interrupt für die BE-Nachrichten immer eingeschaltet, kann bei ständigem Eintreffen von BE-Nachrichten die Software lahm gelegt werden, da diese dann ständig in der Interrupt-Service-Routine (ISR) arbeitet. Wenn dieser Interrupt zeitweise ausgeschaltet werden kann, kann dies nicht mehr passieren.

Ein weiterer Vorteil ist, dass bei einer Hardwareimplementierung des Synchronisationsalgorithmuses die Synchronisationsnachrichten direkt an das Synchronisations-Modul geleitet werden können, ohne dass die Synchronisationsnachrichten vorher von der Software empfangen werden. Dadurch kann eine komplett von der Software entkoppelte Zeitsynchronisation implementiert werden. Daraus ergibt sich das Modul **Switch**, das für das identifizieren von Paketen zuständig ist und diese dann den entsprechenden Puffern oder

Modul	Aufgabe
Timestamping	Setzen von Empfangszeitstempeln
Switch	Empfangspakete an richtige Empfangspuffer leiten
Sync	Uhren synchronisation
Fixed-Point-Timer	Geschwindigkeitsanpassung an globale Uhr
Guard	Asynchrones senden von TT-Nachrichten und Verhinderung von Kollisionen beim Senden

Tabelle 5.1: Zusammenfassung der Modulde

Hardwaremodulen weiterleitet.

Das Time-Triggered Ethernet Protokoll erfordert eine Uhrensynchronisation. Einmal pro Time-Triggered Ethernet Zyklus trifft eine Synchronisationsnachricht ein. Dann wird anhand dieser Nachricht und dem Eingangszeitstempel die neue lokale Uhrzeit berechnet und per Offset-Korrektur korrigiert. Daraus ergibt sich das Modul **Sync**.

Wie eben beschrieben, wird laut Spezifikation nur eine Offset-Korrektur vorgenommen. Um diese Offsetkorrektur gering zu halten und so möglichst über den gesamten Time-Triggered Ethernet Zyklus eine geringe Abweichung zur globalen Uhr zu erreichen, ist ein Timer notwendig, der in der Geschwindigkeit einstellbar ist. Wie schon in Abschnitt 4.2 beschrieben, kann dies mit einem Festkomma-Timer geregelt werden. Daraus ergibt sich das Modul **Fixed-Point-Timer**.

Pakete der Nachrichtenklasse TT müssen zeitgesteuert versendet werden. Damit dies asynchron zur CPU geschehen kann, ist ein Hardwaremodul notwendig, das Nachrichten aus dem Sendepuffer nimmt und diese zum geplanten Zeitpunkt versendet. Ein Vorteil dabei wäre, dass die Software nun zum Erstellzeitpunkt der Nachricht, diese in den Puffer legen kann und zum Versendezeitpunkt nicht nochmal durch ein Interrupt geweckt werden müsste, um den Sendevorgang anzustoßen. Ein weiterer Vorteil ist, dass ein Hardwaremodul mit einem wesentlich kleineren Jitter den Sendevorgang anstoßen kann. Eine weitere Aufgabe, die das Modul umsetzt, ist das kollisionsfreie Versenden von BE-Nachrichten zwischen den TT-Nachrichten. Daraus ergibt sich das Modul **Guard**, dass den Sendevorgang steuert.

Tabelle 5.1 stellt die definierten Module zusammengefasst dar.

5.2 Hardwareimplementierung

In diesem Abschnitt wird eine vollständige Hardwareimplementierung des Time-Triggered Ethernet Protokolls gezeigt. Zunächst werden Standard Ethernet Implementierungen vorgestellt und dabei gezeigt, dass diese hier vorgestellte Implementierung eine Erweiterung einer Standard Ethernet Implementierung ist. Dies zeigt, dass diese Implementierung auch auf andere Systeme portiert werden kann. Im nächsten Abschnitt wird dann ein Konzept zum skalierbaren Hardware- Software Co-Design gezeigt, was mit dieser Hardwareimplementierung möglich ist.

5.2.1 Standard Ethernet Kommunikation

In diesem Abschnitt werden zwei Wege gezeigt, mit denen Standard Ethernet Kommunikation in Controllern realisiert wird.

Abbildung 5.2 zeigt den Aufbau einer einfachen Ethernet Kommunikation. Dieser Aufbau braucht die wenigsten Hardwareressourcen und wird deswegen in günstigen Controllern verwendet.

Das PHY-Modul ist die OSI-Layer 1 Implementierung des Ethernet. Hier ist die Ethernetleitung indirekt angeschlossen. Das PHY-Modul übersetzt die Spannungen auf der Leitung in echte Einsen und Nullen. Es erkennt den Anfang eines Paketes und teilt dies der nächsten OSI-Layer-Schicht mit. Es hat noch viele andere Funktionen, die aber für diese Arbeit nicht relevant sind. Das PHY-Modul schickt die empfangenen Daten an das MAC-Modul. Dies ist die OSI-Layer 2 Implementierung des Ethernet. Hier wird das Ethernetpaket erkannt und auf Ethernetkonformität geprüft. Dabei wird geprüft, ob die CRC-Checksumme gültig ist und die Minimal- und Maximallänge eingehalten wird. Ethernet Light MAC's haben meistens je nur einen Speicher für die Empfangs- und Sendeseite. In jedem Speicher wird nur ein Ethernetpaket abgelegt. Wenn ein Paket empfangen wurde, werden alle weiteren empfangenen Nachrichten verworfen, bis der Speicher wieder frei ist. Wenn ein Paket von der MAC empfangen wurde, löst diese ein Interrupt aus. Anschließend holt die Software auf der CPU das Paket von der MAC ab und gibt den Speicher frei. Erst dann kann das nächste Paket empfangen werden. Diese Implementierung kann nur bei unkritischen Systemen verwendet werden, da bei einem hohen Datenaufkommen viele Pakete verloren gehen werden. Diese Variante ist schlecht erweiterbar.

Abbildung 5.3 zeigt einen Standard Ethernet Aufbau mit einer High-End MAC. Mit diesem Aufbau ist es möglich, auch hohe Datanaufkommen zu bearbeiten. Diese Variante verbraucht die meisten Hardwareressourcen und ist deswegen nur in teuren Controllern

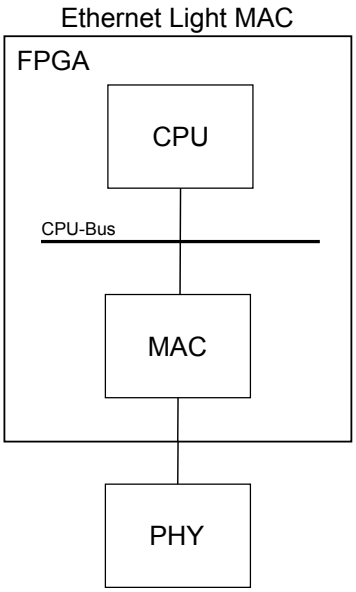

Abbildung 5.2: Standard Ethernet Kommunikation mit einer Light-MAC

vorhanden.

Zur Beschreibung wird hier das MAC-Modul *LogiCORE IP XPS LL TEMAC* (vgl. Xilinx Inc., 2010) von Xilinx verwendet. Andere Hersteller haben MAC-Module, die in ähnlicher Weise arbeiten, wie z. B. das MAC-Modul *Triple-Speed Ethernet MegaCore* (vgl. Altera, 2013) von Altera. Die MAC hat auf der Empfangs- und Sendeseite jeweils einen FIFO-Speicher. Nachdem ein Paket vollständig empfangen wurde, wird es auf Ethernetkonformität geprüft. Anschließend wird es über eine 1-zu-1 Verbindung zum nächsten Modul gesendet. Hier gibt es nun zwei Möglichkeiten, die in der Abbildung 5.3 farblich unterschiedlich dargestellt sind.

Zunächst wird der rot dargestellte Pfad beschrieben. Hier wird das Paket von einem weiteren internen Speicher von der 1-zu-1 Verbindung aufgenommen. Der Speicher in dem Beispiel auf dem Bild ist ein FIFO-Speicher. Nachdem dieser das Paket vollständig empfangen hat, löst er einen Interrupt aus und die Software auf der CPU kann das Paket auslesen. Die Senderichtung funktioniert analog.

Bei dem blau markierten Pfad sendet die MAC über die 1-zu-1 Verbindung die Pakete an einen Memory Controller. Dieser sorgt dafür, dass die Pakete auf einem externen RAM abgelegt werden. Hier wird nach dem vollständigen Empfang ein Interrupt ausgelöst und die Software auf der CPU kann das Paket auslesen.

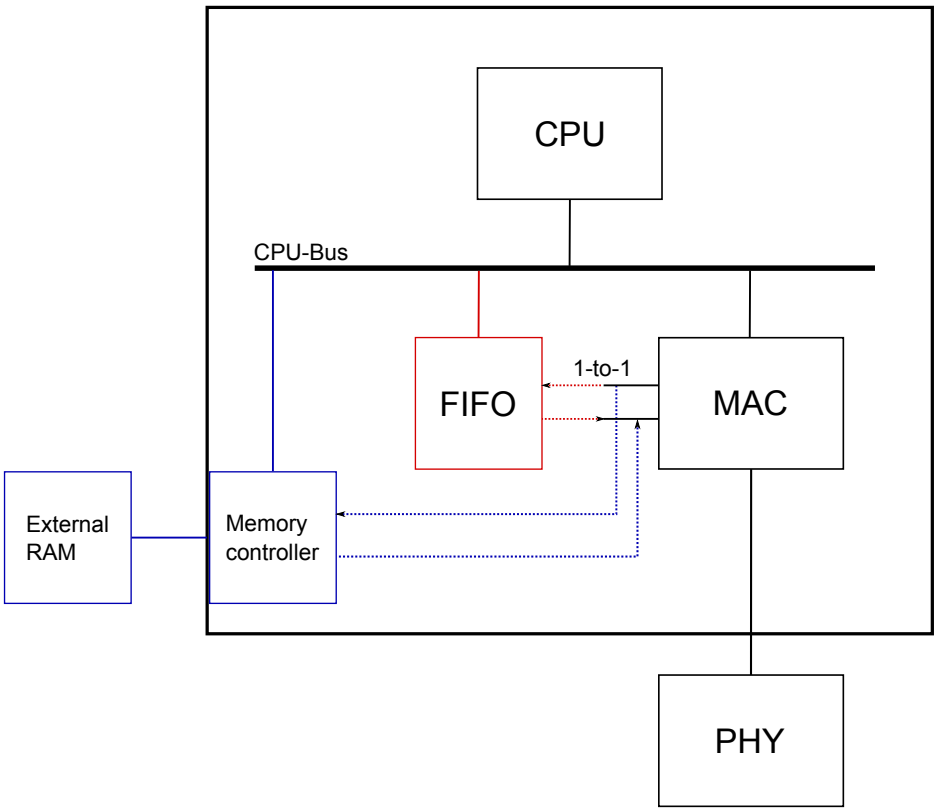

Abbildung 5.3: Standard Ethernet Kommunikation mit einer High-End MAC

Die Lösung mit dem externen RAM ist für größere Datenmengen sehr gut geeignet, da hier größere RAM-Module angeschlossen werden können. Bei der Lösung mit dem internen FIFO-Modul ist oft nur wenig Speicher vorhanden.

Der Ansatz der Time-Triggered Ethernet Hardwareimplementierung ist es, die MAC aus der zuletzt genannten Standard Ethernet Variante zu verwenden und ein Time-Triggered Ethernet Modul an die 1-zu-1 Verbindung anzuschließen. Dies ist in Abbildung 5.4 skizziert. Die Time-Triggered Ethernet Erweiterung kommuniziert über die 1-zu-1 Verbindung mit der MAC und über den CPU-Bus mit der CPU. In den folgenden Abschnitten wird die Implementierung der Erweiterung beschrieben.

5.2.2 Architektur Überblick

Dieser Abschnitt zeigt einen Überblick über die Time-Triggered Ethernet Hardwarearchitektur, die in Abbildung 5.5 dargestellt ist. In den folgenden Abschnitten werden die Module detailliert beschrieben.

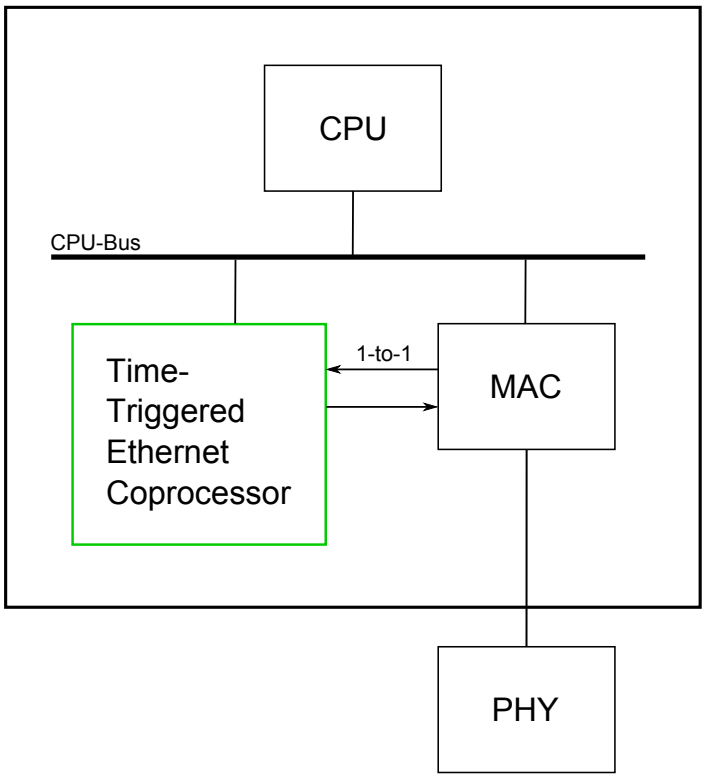

Abbildung 5.4: Skizzierte Time-Triggered Ethernet Erweiterung mit einer Standard Ethernet MAC

Peripherie

Diese Hardwarearchitektur wurde auf dem ML605 FPGA-Board (vgl. Xilinx Inc., 2012b) von Xilinx entwickelt. Die blau dargestellten Module auf der Abbildung 5.5 sind Standardmodule. Als PHY-Modul wurde das Modul 88E1111 (vgl. Marvell Semiconductor, Inc, 2013) verwendet. Das PHY-Modul ist nicht innerhalb des FPGA's, sondern ein externer Chip, der die Ethernetsignale über eine MII/GMII Schnittstelle ausgibt. Als MAC-Modul wurde das Modul LogiCORE IP XPS LL TEMAC (vgl. Xilinx Inc., 2010) von Xilinx verwendet. Diese bietet die Standard Ethernet Funktionalität mit einer 1-zu-1 Schnittstelle, wie es in Abschnitt 5.2.1 beschrieben ist. Der Microblaze Softcore Prozessor (vgl. Xilinx Inc., 2012a) wurde als CPU verwendet. Der PLB (processor local bus) (vgl. IBM Systems and Technology Group, 2012) von IBM wurde als CPU-Bus verwendet.

Im Folgenden wird die Funktionalität der Module beschrieben.

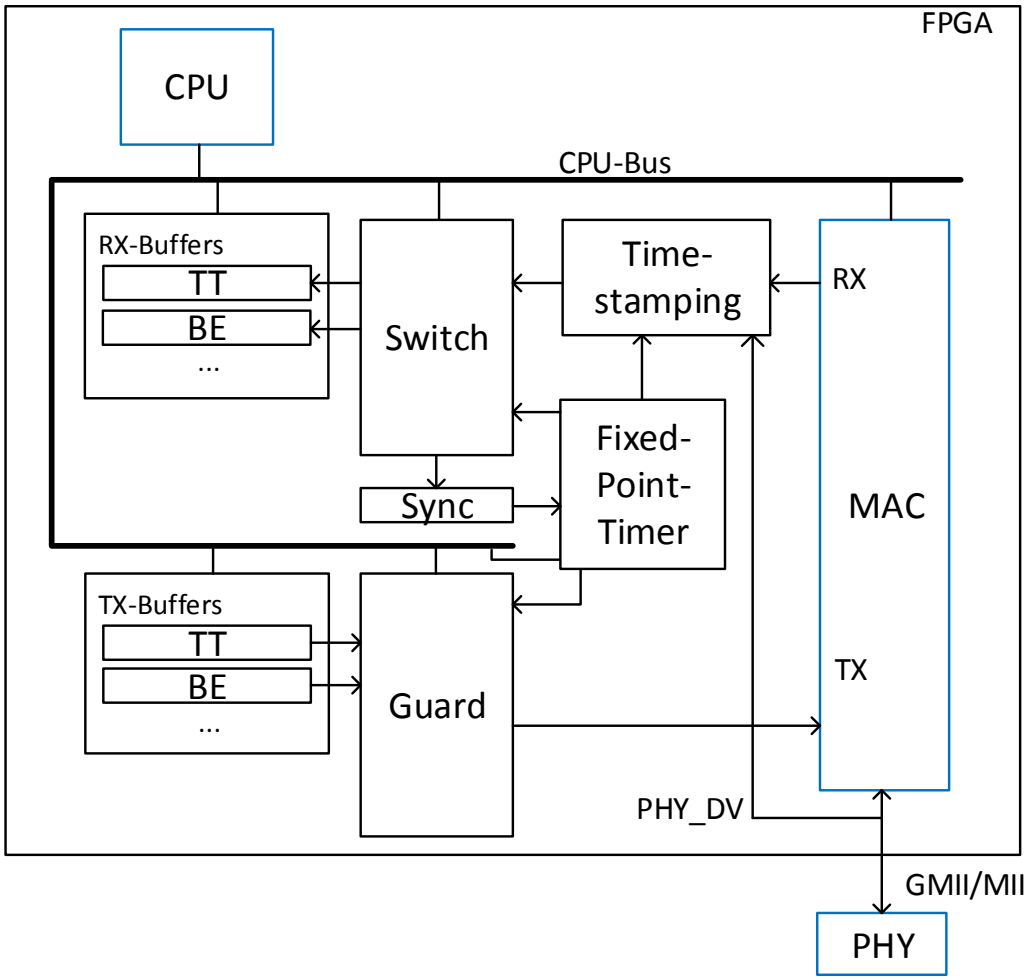

Abbildung 5.5: Time-Triggered Ethernet Hardwarearchitektur
Blau dargestellte Module sind Standardmodule
Schwarz dargestellte Module sind Teil dieser Arbeit

5.2.3 Timestamping

Die Zeitstempeleinheit dient dazu, die physikalische Ankunftszeit eines Ethernetpakets zu messen. Diese Zeit wird in der AS6802 Spezifikation (SAE AS-2D Committee, 2011) als *receive_pit (receive point in time)* bezeichnet und ist definiert als: Ankunft des ersten Bits nach dem SOF (Start of Frame). Die Genauigkeit der Zeitsynchronisation ist abhängig von dieser gemessenen Zeit. Anhand des Zeitpunkts *receive_pit* und des Zeitpunkts *transparent_clock* (siehe Grundlagen Abschnitt 2.1), das im PCF-Paket enthalten ist, wird die Korrektur der Uhr vorgenommen. Ungenauigkeiten des Zeitstempels wirken sich also direkt auf die Genauig-

keit der Uhr aus. Im Abbildung 5.6 zeigt die Untermodule in die das Timestamping-Modul aufgeteilt wurde, diese Untermodule werden in im Folgenden beschrieben.

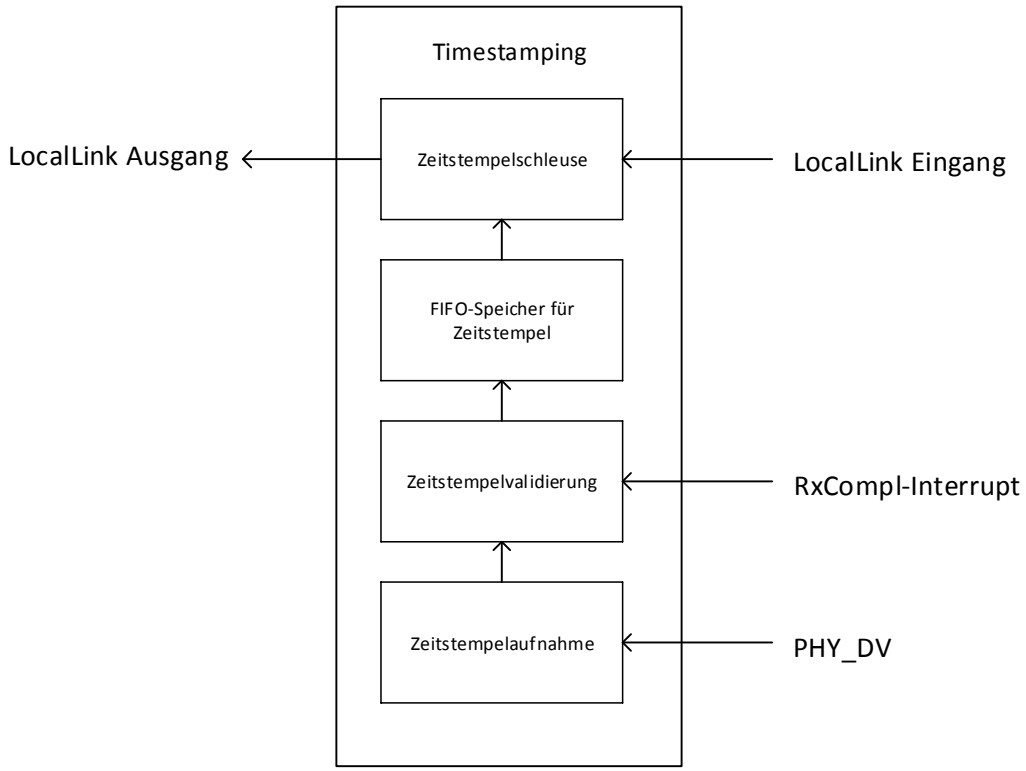

Abbildung 5.6: Untermodule des Timestamping-Moduls

5.2.3.1 Zeitstempelaufnahme

In der Konzeption wurde darauf geachtet, eine möglichst genaue Zeitstempeleinheit zu konstruieren. Dazu wurde das *PHY_DV* Signal aus der *MII/GMII* Schnittstelle verwendet. Dies ist eine standardisierte Schnittstelle, die zur Verbindung von OSI-Layer 1 und 2 verwendet wird. Die *MII* Schnittstelle wird für 10Mbit und 100Mbit Verbindungen verwendet und die *GMII* Schnittstelle wird für 1GBit Verbindungen verwendet. Das *PHY_DV* Signal ist in beiden Schnittstellen vorhanden. Das *PHY_DV* Signal signalisiert dem OSI-Layer 2, dass eine Ethernet-Präambel erkannt wurde und nun gültige Daten folgen. Der OSI-Layer 1 wird im Folgenden *PHY* und der OSI-Layer 2 *MAC* genannt.

Auf dem verwendeten FPGA-Board ist die PHY ein Chip außerhalb des FPGAs und die MAC ein Modul im FPGA. PHY und MAC sind über die FPGA-Pins miteinander verbunden. Um nun den Pegelwechsel des *PHY_DV*-Signals zu messen, wurde innerhalb des FPGA's das

Signal auf einen weiteren Pin, der mit Messspitzen erreichbar ist, verlegt. Alle nun folgenden Messungen wurden im 100MBit Betrieb gemessen.

Die Messung des Signals ergab, dass die Flanke des *PHY_DV*-Signals 390ns vor dem ersten Bit der Ziel-Mac-Adresse ansteigt. Ein Jitter konnte nicht erkannt werden. Aufgrund der Auflösung des Timers, der zur Aufzeichnung verwendet wird, muss jedoch von einem Jitter von 20ns ausgegangen werden. Das Messergebnis ist in Abbildung 5.7 dargestellt.

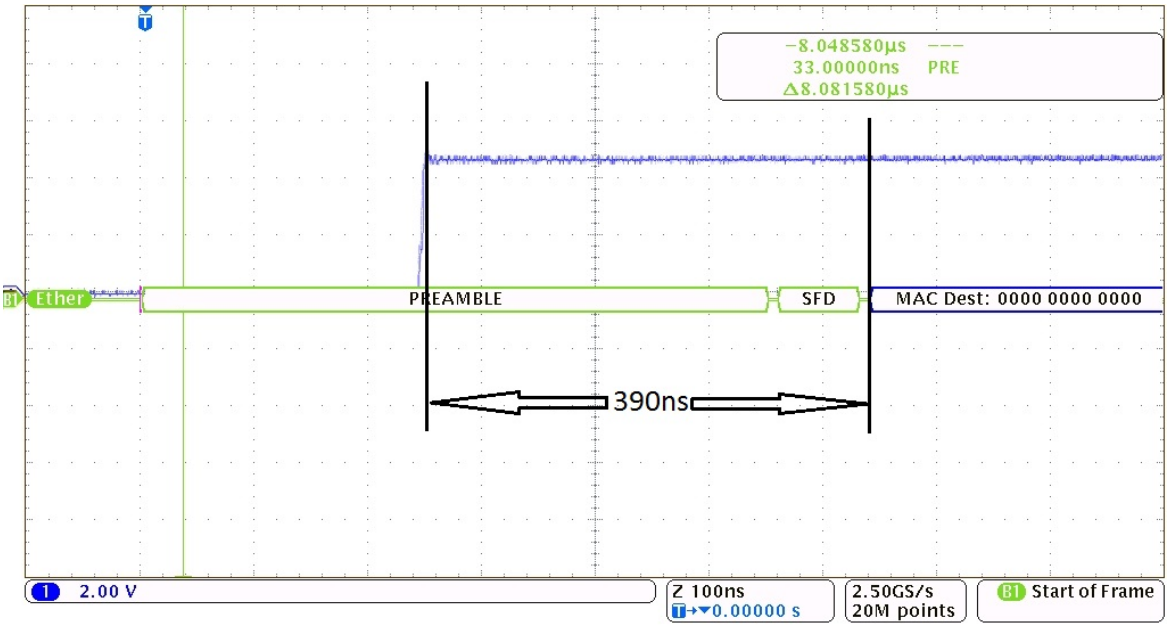

Abbildung 5.7: PHY_DV Signal (blau) steigende Flanke bei der Erkennung eines Ethernetpakets

Um sicherzugehen, ob das *PHY_DV*-Signal auch bei direkt hintereinander eintreffenden Ethernetpaketen verwendet werden kann, muss gemessen werden, ob das *PHY_DV*-Signal vor dem Eintreffen des nächsten Pakets wieder fällt. Eine Messung ergab, dass das Signal 246ns nach Paketende fällt (siehe Abbildung 5.8). Dieses Ergebnis wurde mit verschiedenen Paketgrößen gemessen. Der Inter-Frame-Gap (minimale Zeit zwischen zwei Paketen) beträgt bei einem 100Mbit Netzwerk 960ns. Somit ist dieses Signal auch bei einem minimalem Paketabstand verwendbar.

5.2.3.2 Zeitstempelvalidierung

Durch den in dieser Arbeit verwendeten Aufbau entstehen zwei parallele Pfade, die vor der MAC abzweigen und nach der MAC zusammengeführt werden. Da die MAC intern einen

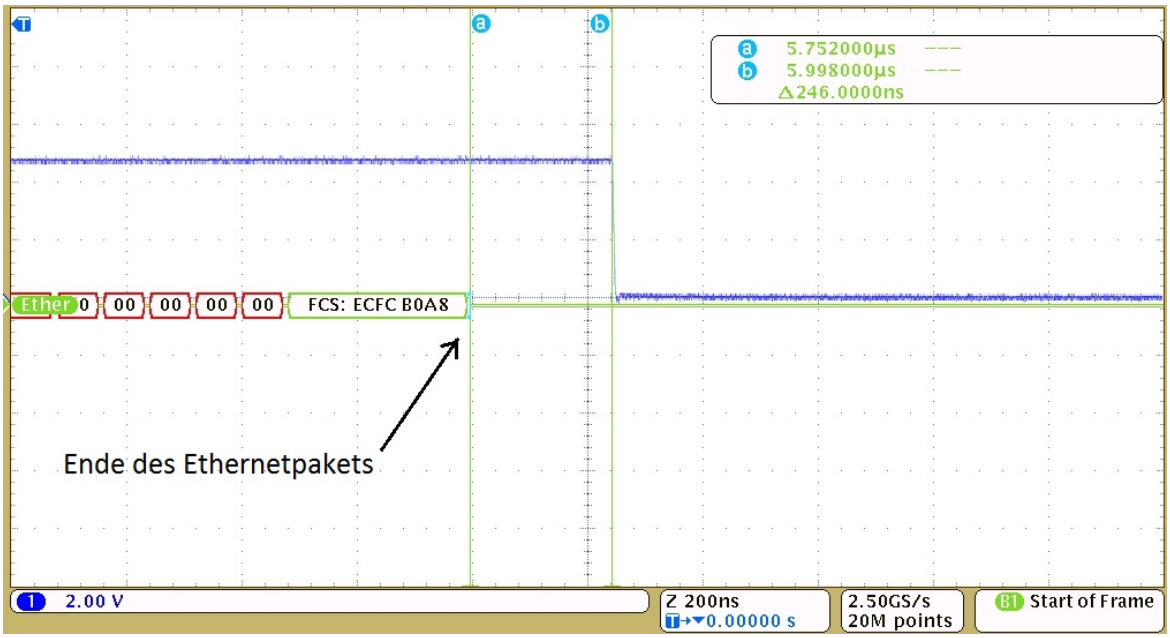

Abbildung 5.8: PHY_DV Signal (blau) fallende Flanke nach der Serialisierung eines Ethernetpakets

FIFO-Speicher hat, in dem mehrere Ethernetpakete sein können, muss das Timestamping-Modul auch einen FIFO-Speicher haben und Zeitstempel puffern, bis das dazugehörige Ethernetpaket die MAC verlässt. Dies birgt die Gefahr, dass Ethernetpakete und Zeitstempel am Ende des Pfades mit einer falschen Zuordnung wieder zusammengeführt werden. Eine Ursache für eine falsche Zusammenführung ist, wenn ein Ethernetpaket nicht Ethernetkonform ist, was bedeutet, dass es zu kurz oder zu lang ist oder der Inter-Frame-Gap nicht eingehalten wurde. Eine zweite Ursache ist, dass die Checksummenüberprüfung fehlschlägt. Eine dritte Ursache ist, dass nicht genug Speicherplatz in der MAC-FIFO vorhanden ist und das empfangene Paket verworfen wird. Diese Kriterien werden innerhalb der MAC überprüft. Die Zeitstempel werden vor der Überprüfung aufgezeichnet. Um eine richtige Zuordnung von Zeitstempel und Ethernetframe sicherzustellen, muss sichergestellt werden, dass der vor dem MAC-Modul aufgezeichnete Zeitstempel ebenfalls verworfen wird, wenn eine der genannten Fälle eintrifft. Wird dies nicht sichergestellt, ist es möglich, dass Zeitstempel den falschen Ethernetpaketen zugeordnet werden. Wie dies realisiert wurde, wird im Folgenden beschrieben.

Die verwendete MAC hat für das Senden und Empfangen jeweils einen FIFO-Speicher, deren Größe sich zwischen 2KB und 32KB einstellen lässt (vor der Synthetisierung). Die MAC bietet einen Interrupt *RxCmplt (Receive Complete)*, der ein erfolgreich empfangenes Paket signalisiert. „Erfolgreich empfangen“ bedeutet in diesem Fall, dass ein Paket zur Ab-

holung bereit steht. Dies garantiert, dass das Paket ethernetkonform ist, die Checksumme erfolgreich überprüft wurde und genug Speicherplatz vorhanden war.

Um diesen Interrupt als Zeitstempelvalidierer verwenden zu können, muss überprüft werden, ob dieser Interrupt bei schnell hintereinander eintreffenden Paketen für jede Nachricht einzeln auslöst. Dazu muss zunächst der Interrupt schnell genug zurückgesetzt werden. Dazu wurde die MAC soweit verändert, dass der Interrupt 100ns nach Auslösung automatisch zurückgesetzt wird. Dazu wurde einfach ein Timer in die MAC integriert, der mit Auslösung des Interrupts angestoßen wird und bei Erreichen eines Zählwerts (nach 100ns) das Interrupt-Reset Register so beschreibt, dass der Interrupt gelöscht wird. Somit entsteht am Interruptausgang ein 100ns Impuls nachdem ein Paket erfolgreich empfangen wurde. Der einzige Grund, warum hier 100ns gewartet wird, ist, damit dieses Signal mit einem Oszilloskop gemessen bzw. getriggert werden kann. Theoretisch kann dieses Signal sofort zurückgesetzt werden, so dass nur ein Impulse über einen Takt entsteht.

Es wurde weiterhin überprüft, ob dieser Interrupt bei zwei direkt aufeinander folgenden Nachrichten mit minimalem Abstand zueinander auch zweimal auslöst. Dazu wurde die Zeit, zwischen Eintreffen des letzten Bits eines Pakets und Auslösung des Interrupts gemessen. Das Ergebnis wird in Abbildung 5.9 dargestellt Das Ergebnis ist 1,76µs. Der Interrupt löst also 1,76µs nach vollständigem Eintreffen eines Ethernetpakets aus. In einem 100MBit Netzwerk beträgt der minimale Zyklus von Paketen 6,72µs. Somit kann für jeden Ethernetframe ein Interrupt ausgelöst werden. Allerdings löst der Interrupt nicht innerhalb des Inter-Frame-Gap aus, was bedeutet, dass ein zweiter Zeitstempel vor der Validierung des ersten Zeitstempels entstehen kann. Um dieses Problem zu lösen, wurde ein Konzept entwickelt, dass zwei unvalidierte Zeitstempel vorhält, bevor diese in den FIFO-Speicher des Timestamping-Moduls gelegt werden. Dieses Konzept wird im Folgenden anhand der Abbildung 5.10 beschrieben.

- Das Paket Pkt1 trifft ein und der Ankunftszeitstempel TS-Pkt1 wird in Register TS1 gespeichert.
- Nach Ablauf von 900ns wird der Inhalt des Registers TS1 (TS-Pkt1) in das Register TS2 verschoben. Warum hier 900ns gewählt wurden, wird im nächsten Durchlauf klar.
- Nun kann ein zweites Paket Pkt2 eintreffen und dessen Ankunftszeitstempel TS-Pkt2 in Register TS1 gespeichert werden.
- 1760ns nach Paketende von PKt1 löst der *RxCmplt*-Interrupt aus, was den ersten Zeitstempel TS-Pkt1 validiert. Dies führt dazu, dass der Zeitstempel TS-Pkt1 aus dem

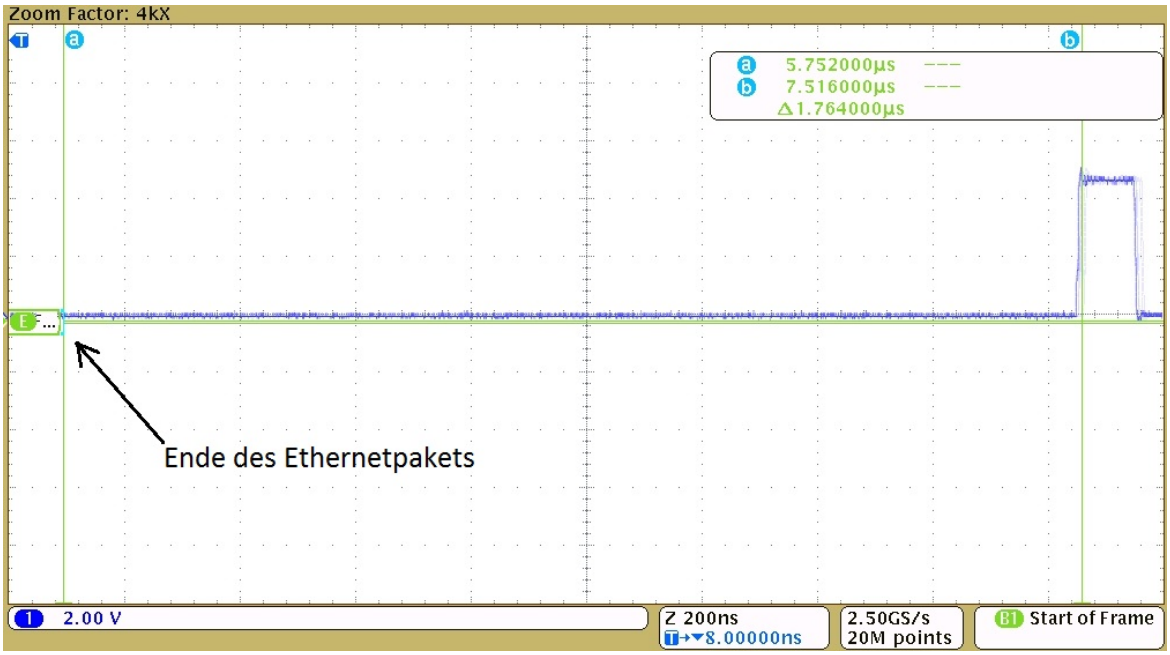

Abbildung 5.9: Messung RxCompl-Interrupt steigende Flanke nach Ethernetpaket-Ende

Register TS2 in den FIFO-Speicher verschoben wird. Damit der Zeitstempel vom zweiten Paket Pkt2 den Zeitstempel vom ersten Paket im Register TS2 nicht überschreibt, muss ein gewisse Zeit gewartet werden. Diese Zeit lässt sich errechnen aus dem Interrupt Delay (1760ns) und dem Inter-Frame-Gap (960ns) und beträgt 1760ns - 960ns = 800ns. Um Ungenauigkeiten im Inter-Frame-Gap entgegen zu kommen, wurden 900ns gewählt.

- 900ns nach Beginn von Pkt2 wird der Wert von TS1 (TS-Pkt2) nach TS2 verschoben.

Durch diesen Mechanismus ist sichergestellt, dass ein neuer Ankunftszeitstempel einen unvalidierten vorherigen Ankunftszeitstempel nicht überschreibt. Somit konnte nachgewiesen werden, dass der *RxCmplt*-Interrupt das gewünschte Verhalten aufweist, dass für die Validierung eines Zeitstempels benötigt wird. Aus Zeitgründen wurde eine Zeitstempelvalidierung mit nur einem Validierungsregister (TS1) implementiert und nicht mit zwei, wie in diesem Konzept beschrieben.

5.2.3.3 Zeitstempelschleuse

Das Zeitstempelschleuse-Modul (Abbildung 5.11) sorgt dafür, dass die aufgezeichneten Zeitstempel im LokalLink-Kanal eingeschleust werden. Dieses Modul ist an dem LocalLink

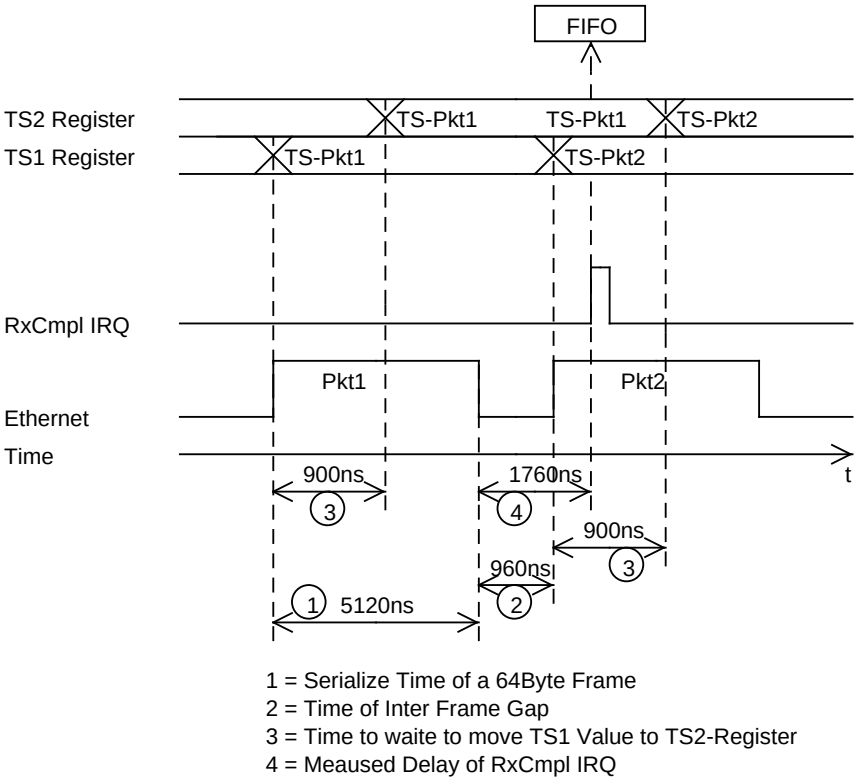

Abbildung 5.10: Zeitverlauf Validierung

Ausgang der MAC angeschlossen und bietet selbst einen eigenen LocalLink Ausgang, an dem das Switch-Modul angeschlossen werden kann. Die Zeitstempel werden der Zeitstempel-FIFO entnommen. In Abbildung 5.11 wird dies schematisch dargestellt.

Das Einschleusen eines Zeitstempels kann aufgrund der Flusskontrollmechanismen des LocalLink-Interface relativ einfach erfolgen (Siehe Grundlagen Kapitel 2.2). Sobald der Sender mit dem *SOP_n*-Signal den Beginn des Payload signalisiert, signalisiert die Zeitstempelschleuse der MAC mit dem *DST_RDY_n*-Signal, dass die Übertragung pausiert werden soll. Dies wird für 6 Takte vorgenommen. In den ersten vier Takten wird gewartet, bis ein Zeitstempel aus der Zeitstempel-FIFO entnommen wurde. Diese Wartezeit ist notwendig, weil der *RxCmplt*-Interrupt gleichzeitig mit dem *SOP_n*-Signal aktiviert werden. Der Zeitstempel wird aber erst mit dem *RxCmplt*-Interrupt in die Zeitstempel-FIFO gespeichert. Das heißt, zum Zeitpunkt des *SOP_n*-Signal der Zeitstempel noch nicht mal in der FIFO ist. Es muss also gewartet werden, bis diese in die FIFO gespeichert wird und dort auf der anderen Sei-

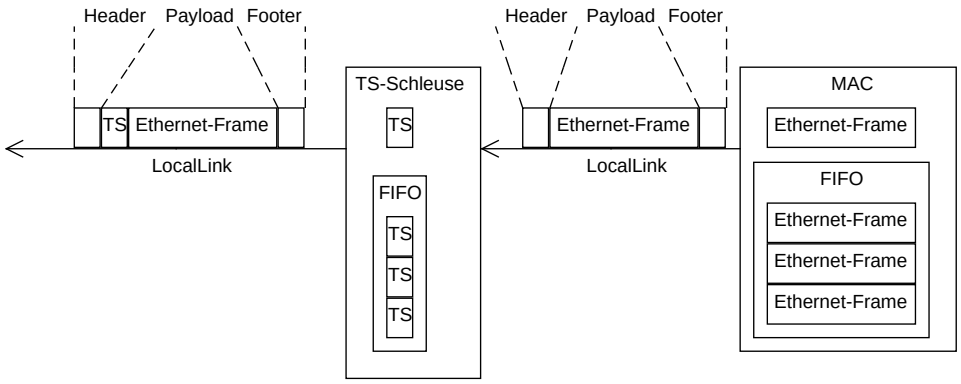

Abbildung 5.11: Schematische Darstelung der Zeitstempelschleuse

te (bei der Zeitstempelschleuse) wieder rausgeholt wird. Dies dauert 4 Takte. Am LocalLink Ausgang der Zeitstempelschleuse wird innerhalb der nächsten zwei Takte der Zeitstempel angelegt. Das *SRC_RDY_n*-Signal am Ausgang wird zu dem Zeitpunkt aktiviert, so dass der am Ausgang anliegende Empfänger nichts von der Unterbrechung merkt. Somit wird die Größe des Payload um 2 x 32 Bit verlängert, worin der Zeitstempel eingefügt wird.

5.2.3.4 Hardwareressourcen

Das Timestamping-Modul braucht 72 Register und 109 LUTs[2]. Hier wurde der FIFO-Speicher nicht mit eingerechnet.

5.2.4 Switch

Das Switch-Modul soll dazu dienen, den Typ der empfangenen Pakete zu identifizieren und an die entsprechende Ziel-Hardwaremodule weiter zu leiten. Die Bestimmung des Ziels kann anhand folgender Kriterien vorgenommen werden:

Ethernet-Type Durch den Inhalt des Ethernet-Type Feldes können beispielsweise Synchronisationsnachrichten (PCF) eindeutig identifiziert werden. Für PCF Nachrichten aus dem AS6802 Protokoll wurde bei der IEEE der Ethernet-Type 891d registriert (vgl. IEEE Registration Authority, 2014). Wird dieser Wert bei einem Ethernetpaket erkannt, kann die Switch diese Nachricht an das Sync-Modul weiterleiten, ohne andere Kriterien zu prüfen.

[2]Lookup-Tabelle. Eine LUT kann, je nach Anzahl der verfügbaren Eingänge, jede beliebige n-stellige Binärfunktion realisieren. Die Programmierung der gewünschten Funktion erfolgt durch die Hinterlegung der definierenden Wahrheitstabelle in den SRAM-Zellen der LUT

CT-ID Eine CT-ID (Critical Traffic ID) wird einer konkreten Nachricht im Time-Triggered Ethernet Zyklus zugeordnet und befindet sich im Ethernet-Header im Feld der Destination-Mac-Adresse. Aus einer CT-ID kann auch die Größe der Nachricht abgeleitet werden, da diese zum Designzeitunkt des Netzwerks offline konfiguriert wird. Wenn eine Nachricht mit einer bestimmten CT-ID erkannt wurde, kann diese in den entsprechenden Puffer weitergeleitet werden.

Ankunftszeitstempel Im Switch-Modul kann bei den Echtzeitnachrichten anhand des Ankunftszeitstempels überprüft werden, ob diese rechtzeitig eingegangen sind und somit die Echtzeitbedingung erfüllt haben. Liegt der Ankunftszeitstempel außerhalb eines Zeitfensters, das für eine CT-ID festgelegt wurde, kann diese im Switch-Modul als ungültig betrachtet und verworfen werden, ohne dass die Software durch diese Nachricht belastet wird.

Speicherfüllstand Für jede CT-ID wird eine maximale Paketgröße festgelegt. Somit ist mit dem Auslesen der CT-ID die maximale Größe der Nachricht bekannt, bevor die Nachricht vollständige empfangen wurde. Nun kann der Speicherfüllstand der Zielhardware überprüft und festgestellt werden, ob diese Nachricht in die Zielhardware passt. Der in dieser Implementierung verwendete FIFO-Speicher bietet eine Schnittstelle an, mit dem der Füllstand abgefragt werden kann. Es könnte auch eine andere Strategie verfolgt werden, bei der die Zielhardware dazu getriggert wird, die älteste Nachricht zu löschen, um Platz für das neue Paket zu schaffen.

In Abbildung 5.12 wird der Aufbau des Moduls dargestellt. Auf der rechten Seite im Bild sieht man den LocalLink-Eingang. Über diesen Eingang werden die Ethernetpakete in das Modul übertragen. Die Daten werden in ein Schieberegister, das 6 Plätze hat, gespeichert. Das Schieberegister nimmt dabei den Datenteil und den Kontrollteil des LocalLink-Bus auf. In einem weiteren Feld wird ein Zielschlüssel gespeichert. Dieser ist zu Beginn unbekannt. Nachdem alle Schieberegister gefüllt sind, beginnt das Modul *Caluclate Destination* anhand der oben beschriebenen Kriterien den Zielschlüssel zu bestimmen. Dabei wird das *1st*-Signal ausgelöst, das dafür sorgt, dass in allen Schieberegistern das Feld mit dem Zielschlüssel (*regx-dest*) beschrieben wird. Im nächsten Takt ist das *1st*-Signal nicht mehr aktiv und es wird nur das erste Zielschlüsselfeld (*reg0-dest*) von dem Modul *Caluclate Destination* beschrieben. Alle anderen Zielschlüsselfelder erhalten ihren Wert vom Vorgänger-Register. Im gleichen Takt erhält der Demultiplexer, der links in der Abbildung zu sehen ist, die ersten Daten und weiß anhand des Zielschlüssels, wohin diese weitergeleitet werden sollen. Die Ausgänge des Demultiplexers sind vollwertige LocalLink-Ausgänge mit Daten- und Kontrollteil.

Das Schieberegister im Switch-Modul ist so konzipiert, dass es immer taktet (shiftet). Dadurch muss sichergestellt werden, dass in den Zielmodulen entweder immer genug Platz für

die Daten vorhanden ist oder das Zielmodul muss dem Switch-Modul mitteilen, dass nicht genug Platz für dieses Paket vorhanden ist, damit das Switch-Modul das Paket verwerfen kann. Ein Vorteil dieses Konzeptes ist es, dass die Verzögerung konstant 6 Takte beträgt. In diesem Fall 60ns. Time-Triggered Nachrichten können nicht wegen gefüllter Puffer auf unbestimmte Zeit verzögert werden. Außerdem ist das Switch-Modul den Zielmodulen gegenüber transparent. Entscheidet man sich, das Switch-Modul nicht zu verwenden, beispielsweise weil das Steuergerät keine Time-Triggered Nachrichten empfangen, sondern nur senden muss, so kann dies aus der Hardware einfach entfernt werden. Der Eingangspuffer kann dann direkt an das Timestamping-Modul oder MAC-Modul angeschlossen werden.

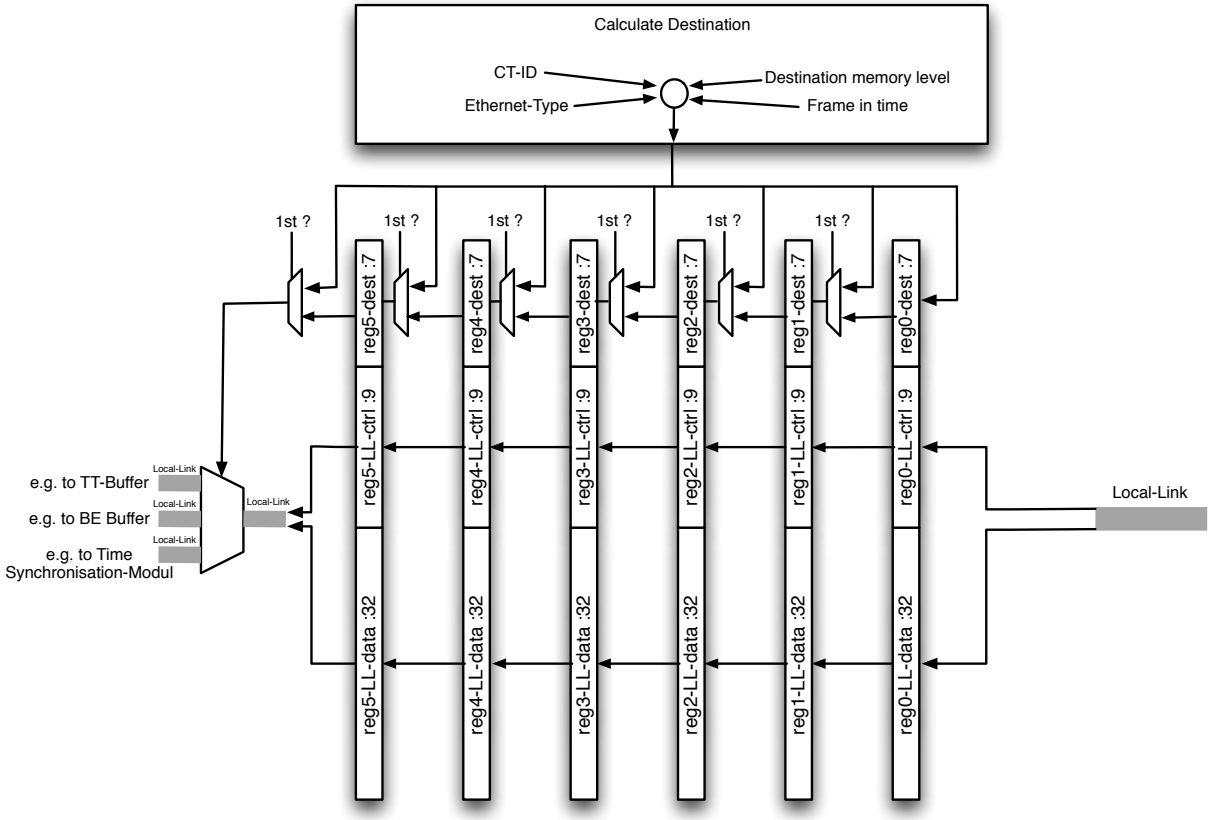

Abbildung 5.12: Schematischer Aufbau des Switch-Modul

5.2.4.1 Hardwareressourcen

Eine Switch-Modul mit zwei Ausgangskanälen braucht 179 Register und 100 LUTs.

5.2.5 Sync-Modul

Im Folgenden werden zwei Implementierungen des Sync-Moduls beschrieben. Die erste Variante entspricht dem Synchronisations-Client, der in der AS6802 Spezifikation beschrieben ist (vgl. SAE AS-2D Committee, 2011). Die zweite Variante ist eine Light-Implementierung, bei der davon ausgegangen wird, dass nur ein Synchronisationspaket pro Time-Triggered Ethernet Zyklus eintrifft. Beide Varianten wurden implementiert und durch eine Simulation verifiziert. Die Light-Implementierung wurde synthetisiert und in einem Testaufbau erprobt.

5.2.5.1 Sync-Modul AS6802

Die AS6802 Implementierung wurde nach Vorgabe der Spezifikation (vgl. SAE AS-2D Committee, 2011) implementiert (siehe dazu Abschnitt 2.1 auf Seite 5 aus dem Kapitel Grundlagen). In dieser Spezifikation ist das Verhalten von Synchronisations-Master, Compression-Master und Synchronisations-Client (SC) beschrieben. In dieser Arbeit wurde der SC implementiert.

Laut Spezifikation kann ein SC über mehrere Ethernetports verfügen, über die Synchronisationsnachrichten (PCF) eintreffen können. Aus dem Grund wurde das Sync-Modul in zwei Teile aufgeteilt. Abbildung 5.13 zeigt den Aufbau mit zwei Ethernetports. Das Untermodul *tte_sync_channel* empfängt die PCF innerhalb des Zeitfensters $acceptance\_window$, ermittelt den besten PCF (siehe Abschnitt 2.1), berechnet den Wert *permanence_pit* und stellt diesen dem Untermodul *tte_sync* bereit. Es werden noch weitere Werte bereitgestellt, die später beschrieben werden. Das Modul *tte_sync* berechnet aus dem bereitgestellten Werten aller Ethernetports den Offset-Korrekturwert und korrigiert die Zeit im Fixed-Point-Timer-Modul.

Untermodul tte_sync_channel

Das *tte_sync_channel*-Modul muss softwareseitig mit folgenden Parametern konfiguriert werden:

local_acceptance_window_begin Startzeitpunkt des Zeitfensters *acceptance_window* in der Einheit 20ns.

local_acceptance_window_end Endzeitpunkt des Zeitfensters *acceptance_window* in der Einheit 20ns.

max_transmission_deleay Maximale Verzögerung aller SM, um ein PCF zu versenden (siehe Abschnitt 2.1), in der Einheit 20ns.

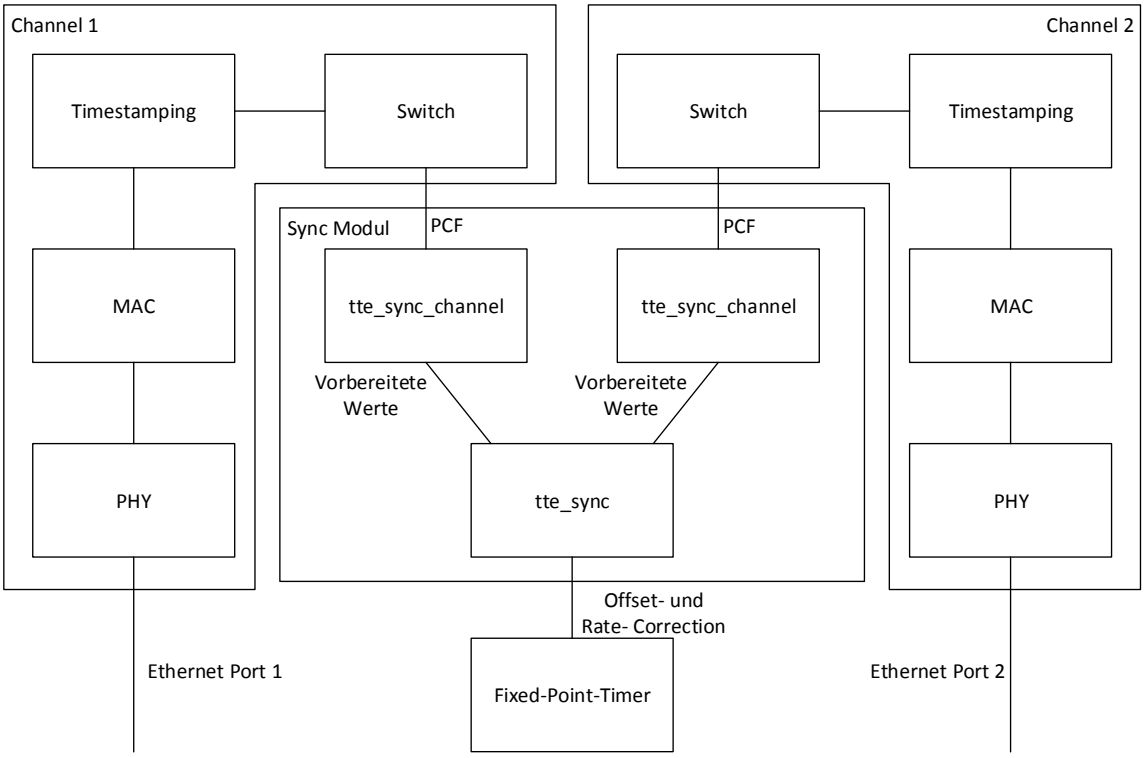

Abbildung 5.13: Aufbau: Synchronisation über zwei Ethernetports
Es werden nur Module und Verbindungen dargestellt, die an der Synchronisation beteiligt sind.

receive_delay Verzögerung zwischen dem Eintreffen des PCF und dem Zeitpunkt, an dem ein Zeitstempel aufgezeichnet wurde. Einheit 20ns.

local_sync_priority Sync-Priorität. Lokal fest konfigurierter Wert. Es werden nur PCF verarbeitet, die zu dieser Sync-Priorität gehören.

local_sync_domain Sync-Domain. Lokal fest konfigurierter Wert. Es werden nur PCF verarbeitet, die zu dieser Sync-Domain gehören.

Folgende variable Daten erhält das Modul vom Modul *tte_sync*:

local_integration_cycle Zyklusnummer, in der sich das Sync-Modul befindet. Es werden nur PCF akzeptiert, die zu diesem Zyklus gehören.

state_integrate Signal, das angibt, ob sich das Modul *tte_sync* in dem Zustand *state_integrate* befindet (siehe nächsten Abschnitt).

Folgende Daten werden von dem Modul *tte_sync_channel* übermittelt:

best_membership_new_num_out Anzahl 1-Bits im PCF Feld $pcf\_membership\_new$ des besten PCF.

best_permanence_pit_out Errechnetes Zwischenergebnis (siehe Grundlagen) des besten PCF.

pcf_integration_cycle_out Wert aus dem Feld *pcf_integration_cycle* des PCF.

Das *tte_sync_channel*-Modul ist mit dem Switch-Modul über die LocalLink-Schnittstelle verbunden. Diese überträgt 4 Byte pro Takt und ist mit 100MHz getaktet. Das *tte_sync_channel*-Modul erkennt durch das SOF-Signal der LokalLink-Schnittstelle, wenn ein PCF eintrifft und startet einen Zähler. Abbildung 2.3 auf Seite 9 zeigt den Aufbau eines PCF. Das PCF wird nicht zwischengespeichert, sondern direkt aus der Empfangsschnittstelle geparst. Wenn die Übertragung beginnt, wird ein Zähler *dword_counter* gestartet, der von 0 an mit jedem Takt hochzählt. Im Folgenden wird dieser Zählwert Takt genannt. Vor dem PCF wird der Eingangszeitstempel übertragen.

Für das Bereitstellen der Ergebnisse braucht das *tte_sync_channel*-Modul 63 Takte - gezählt von der Übertragung der ersten vier Bytes des PCF bis zur Bereitstellung des Ergebnises $best\_permanence\_pit\_out$. Im Folgenden wird dieser Vorgang beschrieben.

In den ersten 11 Takten wird das PCF und der Eingangszeitstempel über die LocalLink-Schnittstelle eingelesen (Felder siehe Abbildung 2.3 auf Seite 9). Zwei Felder werden direkt beim Einlesen umgerechnet.

In dem Feld $pcf\_membership\_new$ entspricht die Anzahl der Bits, die auf 1 gesetzt sind, der Anzahl der Synchronisations-Master, die an der Synchronisation beteiligt sind. An das Modul *tte_sync* wird direkt die Anzahl der 1-Bits über den Ausgang $best\_membership\_new\_num\_out$ übergeben. Dieser Wert wird in Takt 7 innerhalb eines Taktes mit folgender VHDL-Funktion berechnet:

```
1  function count_ones(s : std_logic_vector) return integer is
2          variable temp : natural := 0;
3  begin
4          for i in s'range loop
5                  if s(i) = '1' then
6                          temp := temp + 1;
7                  end if;
8          end loop;
9          return temp;
10 end function count_ones;
```

Diese Funktion (vgl. Rushton, 2011, Kapitel 11.2.4) erhält einen std_logic_vector, errechnet die Anzahl 1-Bits und gibt diesen Wert als $integer$ im selben Takt zurück. Wie dies in Hardware aussieht, wird dem Synthetisierer überlassen.

Das Feld *pcf_transparent_clock* hat die Einheit 1ns. Die lokale Uhr hat die Einheit 20ns. Alle Berechnungen werden in der Einheit der lokalen Uhr vorgenommen. Daher muss das Feld $pcf\_transparent\_clock$ auf diese Einheit umgerechnet werden. Dazu wurde ein Dividierer-Modul von (Miller, 2009) verwendet, mit dem das Feld $pcf\_transparent\_clock$ durch 20 geteilt wird. Das Dividierer-Modul wird in Takt 12 angestoßen. In Takt 61 steht das Ergebnis bereit.

Nachdem das Feld $pcf\_transparent\_clock$ in der Einheit 20ns errechnet wurde, wird im Takt 61 mit der Formel 2.5 auf Seite 10 der Wert $permanence\_delay$ berechnet. In Takt 62 wird der Wert $permanence\_pit$ mit der Formel 2.6 auf Seite 10 berechnet. In Takt 63 wird geprüft, ob das Paket zur richtigen Sync-Domain gehört, zur richtigen Sync-Priority gehört und ob es innerhalb des Zeitfensters $acceptance\_window$ eingetroffen ist. Anschließend wird mit der Formel 2.7 auf Seite 10 geprüft, ob es das bisher beste PCF ist, dass innerhalb des Zeitfensters eingetroffen ist. Falls ja, werden die Register $best\_membership\_new\_num$ und $best\_permanence\_pit$ beschrieben. Diese Werte werden von dem Modul *tte_sync* verwendet, das im Folgenden beschrieben ist.

Untermodul tte_sync

Das *tte_sync*-Untermodul wurde so implementiert, dass es mit nur einem Ethernetport kommuniziert. Es wurde bei der Implementierung darauf vorbereitet, dass eine Erweiterung auf mehrere Ethernetports mit wenig Aufwand möglich ist. Das *tte_sync*-Untermodul wurde als Automat nach Vorgabe der Spezifikation AS6802 implementiert (vgl. SAE AS-2D Committee, 2011). Dieser Automat ist in Abbildung 5.14 beschrieben. Zunächst werden die Schnittstellen des Moduls beschrieben und anschließend der Automat.

Im Folgenden werden Parameter beschrieben, die softwareseitig konfiguriert werden müssen.

sc_integrate_to_sync_thrld: Minimale Anzahl 1-Bits im $membership\_new$-Feld, um vom Zustand *sc_integrate* zum Zustand *sc_sync* zu gelangen.

sc_sync_threshold_sync: Minimale Anzahl 1-Bits im $membership\_new$-Feld, um nicht von dem Zustand *sc_sync* zurück in den Zustand *sc_integrate* zu verfallen.

sc_stable_threshold_sync: Minimale Anzahl 1-Bits im $membership\_new$-Feld, um im Zustand *sc_stable* keinen instabilen Zyklus zu zählen. Ein instabiler Zyklus ist ein

Zyklus, bei dem die Anzahl 1-Bits im $membership\_new$-Feld kleiner dem Wert $sc\_stable\_threshold\_sync$ ist.

smc_scheduled_receive_pit: Geplante Ankunftszeit des PCF. Einheit 20ns.

smc_sync_eval_pit: Zeitpunkt, bei dem die Ergebnisse des Moduls *tte_sync_channel* verarbeitet werden.

smc_clock_corr_pit: Zeitpunkt, bei dem die Offset-Korrektur durchgeführt wird.

num_stable_cycles: Anzahl aufeinander folgender stabiler Zyklen, um von dem Zustand *sc_sync* zum Zustand *sc_stable* wechseln zu können. Ein stabiler Zyklus ist ein Zyklus, bei dem die Anzahl 1-Bits im $membership\_new$-Feld mindestens dem Wert $sc\_sync\_threshold\_sync$ entspricht.

num_unstable_cycles: Anzahl aufeinander folgender instabiler Zyklen. Wird diese Anzahl erreicht, wird von dem Zustand *sc_stable* in den Zustand *sc_sync* gewechselt.

sc_sync_to_stable_enabled: Mit diesem Parameter gibt die Software dem Automaten die Erlaubnis, in den Zustand *sc_stable* zu wechseln.

Im Folgenden wird die Schnittstelle zum Modul Fixed-Point-Timer beschrieben.

local_clock: Eingang der lokalen Uhr. Einheit 20ns.

local_clock_setter: Ausgang der neu berechneten Uhrzeit nach einer Offset-Korrektur.

local_clock_set_en: Ausgang Enable-Bit um dem Fixed-Point-Timer mitzuteilen, dass die lokale Uhr auf den Wert $local\_clock\_setter$ gesetzt werden soll.

Die Schnittstelle zwischen *tte_sync* und *tte_sync_channel* wurde im vorigen Abschnitt schon beschrieben.

Automat

Abbildung 5.14 zeigt den Zustandsautomaten mit dem das Modul *tte_sync* realisiert wurde. Dieser entspricht dem Automaten der in der AS6802 Spezifikation für den Synchronisations-Client beschrieben wird. In dem hier beschriebenen Automaten wurde die Clique-Detection-Funktionalität weggelassen. Im Folgenden wird dieser beschrieben.

SC_START Dies ist der Startzustand. In diesem Zustand soll die Software das Sync-Modul konfigurieren. Wenn dies abgeschlossen ist, setzt die Software das Signal $sync\_enable$ auf 1, was einen Zustandsübergang zum Zustand *SC_INTEGRATE* auslöst.

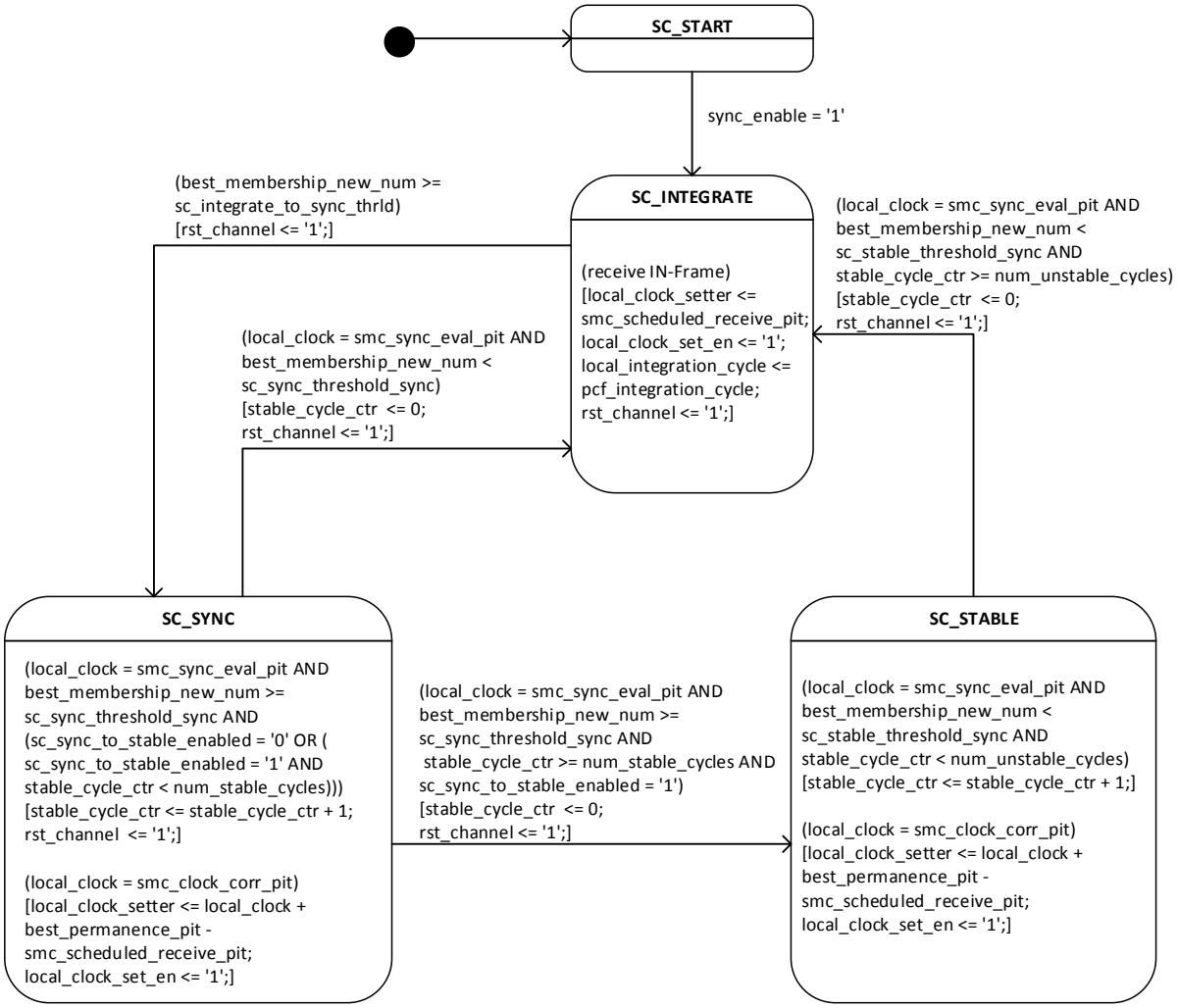

Abbildung 5.14: Zustandsautomat des Moduls *tte_sync* (vgl. SAE AS-2D Committee, 2011, S.65)
In runden Klammern werden Bedingungen dargestellt. Werden diese erfüllt, werden die darauf folgenden, in eckigen klammern dargestellten, Aktionen ausgeführt

SC_INTEGRATE In diesem Zustand wird gewartet, bis das erste PCF vom Typ Integration-Frame (IN) eintrifft. Wenn eins eintrifft, wird die lokale Uhr auf den Wert $smc\_scheduled\_receive\_pit$ gesetzt und der lokale Zähler für den Integrationszyklus auf den Wert $pcf\_integration\_cycle$. Wenn im PCF-Feld $pcf\_membership\_new$ genug Bits auf 1 gesetzt sind, wird in den Zustand *SC_SYNC* gewechselt.

SC_SYNC In diesem Zustand wird eine Offset-Korrektur durchgeführt. Dabei werden stabile Zyklen gezählt. Erreicht die Anzahl der stabilen Zyklen den Wert num_stable_cycles, wird in den Zustand *SC_STABLE* gewechselt. Ist die Anzahl 1-Bits im Feld $pcf\_membership\_new$ kleiner dem Wert $sc\_sync\_threshold\_sync$, so wird in den Zustand *SC_INTEGRATE* gewechselt. Per Software kann zusätzlich mit dem Signal $sc\_sync\_to\_stable\_enable$ entschieden werden, ob in den *SC_STABLE* gewechselt werden darf.

SC_STABLE In diesem Zustand gilt die Synchronisation als stabil. Es wird eine Offset-Korrektur vorgenommen. Es werden instabile Zyklen gezählt. Übersteigt die Anzahl instabiler Zyklen den Wert $num\_unstable\_cycles$, so wird in den Zustand *SC_INTEGRATE* gewechselt.

In der AS6802 Implementierung wurde kein Rate-Correction (Geschwindigkeitskorrektur) implementiert.

Hardwareressourcen

Die AS6802-Implementierung des Sync-Moduls braucht 736 Register und 1100 LUTs.

5.2.5.2 Sync-Modul Light-Implementierung

Die Light-Implementierung des Sync-Moduls unterscheidet sich von der AS6802-Implementierung dadurch, dass jedes empfangene Synchronisationspaket zu einem Synchronisationsvorgang führt. Es werden keine Zeitfenster für den Empfang von PCFs beachtet. In dieser Implementierung wurde eine Rate-Correction implementiert. Es gibt nur einen Parameter, der softwareseitig konfiguriert werden muss:

integration_cycle_duration Dauer eines Time-Triggered Ethernet Zykluses in der Einheit 20ns.

Im Folgenden wird die Schnittstelle zum Fixed-Point-Timer-Modul beschrieben:

local_clock: Eingang der lokalen Uhr. Einheit 20ns.

local_clock_setter: Ausgang der neu berechneten Uhrzeit nach einer Offset-Korrektur.

local_clock_set_en: Ausgang Enable-Bit, um dem Fixed-Point-Timer mitzuteilen, dass die lokale Uhr auf den Wert $local\_clock\_setter$ gesetzt werden soll.

local_addup Eingang des aktuellen ADDUP-Wertes des Fixed-Point-Timers.

local_addup_setter Ausgang des neu berechneten ADDUP-Wertes.

local_addup_set_en Ausgang Enable-Bit, um dem Fixed-Point-Timer mitzuteilen, dass der neue ADDUP-Wert $local\_addup\_setter$ übernommen werden soll

Über die LocalLink-Schnittstelle erhält das Sync-Modul die PCFs. Es werden nur der Eingangszeitstempel $receive\_pit$ und das Feld $pcf\_transparent\_clock$ von dieser Schnittstelle geparst. Wie bei der AS6802-Implementierung wird ein lokaler Timer während des Empfangs gestartet, der je Takt von 0 aufwärts zählt. Nach 12 Takten sind die eben beschriebenen Felder eingelesen. Auch hier muss das $pcf\_transparent\_clock$-Feld durch 20 geteilt werden, um es auf die Einheit der lokalen Uhr zu bringen. Das Ergebnis der Division steht im Takt 63 bereit. In diesem Takt wird die Offset-Korrektur vorgenommen.

Anhand des Wertes der Offset-Korrektur und der Zykluslänge wird der neue ADDUP Wert berechnet, der die Geschwindigkeit der lokalen Uhr verändert. Dies wird mit Formel 5.1 berechnet.

$$ADDUP = ADDUP + \frac{offset\_correction\_value}{integration\_cycle\_duration} \tag{5.1}$$

Die Division in dieser Formel gibt den Faktor an, um den der aktuelle ADDUP-Wert verändert werden soll. Wie der Name „Faktor“ schon sagt, müsste dieser eigentlich mit dem ADDUP-Wert multipliziert werden, wie in Formel 5.2 mathematisch korrekt dargestellt.

$$ADDUP = ADDUP + ADDUP * \frac{offset\_correction\_value}{integration\_cycle\_duration} \tag{5.2}$$

In diesem Fall führt eine Addition (im Vergleich zu einer Multiplikation) zu einem sehr geringen Fehler, da der ADDUP-Wert immer nahezu den Wert 1,0 hat. Bei Quarzen kann man davon ausgehen, dass die Geschwindigkeitsabweichung im kleinen einstelligen Prozentbereich liegt. Weicht die Geschwindigkeit beispielsweise um ein Prozent nach unten ab, so hat der ADDUP einen Wert von $1,01_{10}$. Wird nun im nächsten Durchlauf nochmal um ein Prozent korrigiert, liefert Formel 5.1 den Wert 1,02 und die Formel 5.2 den Wert 1,0201. Die Abweichung im Ergebnis (100ppm) entspricht der Schwankung eines Quarzes bei hohen Temperaturschwankungen. Durch das Weglassen der Multiplikation können viele Hardwareressourcen gespart werden. Daher wird dieser geringe Fehler in Kauf genommen.

Die Division aus Formel 5.1 wird in Takt 63 angestoßen. Im Takt 97 steht das Ergebnis bereit. In diesem Takt wird dann die Geschwindigkeitskorrektur, wie eben beschrieben, vorgenommen.

Hardwareressourcen

Die Light-Implementierung braucht 364 Register und 564 LUTs.

5.2.6 Fixed-Point-Timer-Modul

Das Fixed-Point-Timer-Modul ist ein Timer, bei dem die Geschwindigkeit eingestellt werden kann. Das Modul funktioniert wie eine Festkommazahl. Der Timer zählt zyklisch von 0 bis zum eingestellten Zeitpunkt *integration_cycle_duration* (So wird die Time-Triggered Ethernet Zyklusdauer in der AS6802 Spezifikation genannt).

Abbildung 5.15 zeigt den Aufbau des internen Timers. In dieser Implementierung werden 33 Bit (63 bis 31) vor dem Komma verwendet und 31 Bit (30 bis 0) nach dem Komma. Die obersten 32 Bit (63 bis 32) werden von den anderen Modulen als Zeitgeber verwendet. Bit 31, dass vor dem Komma ist, wird von den anderen Modulen nicht verwendet, da hier Sprünge möglich sind. Wie diese entstehen wird im Folgenden beschrieben.

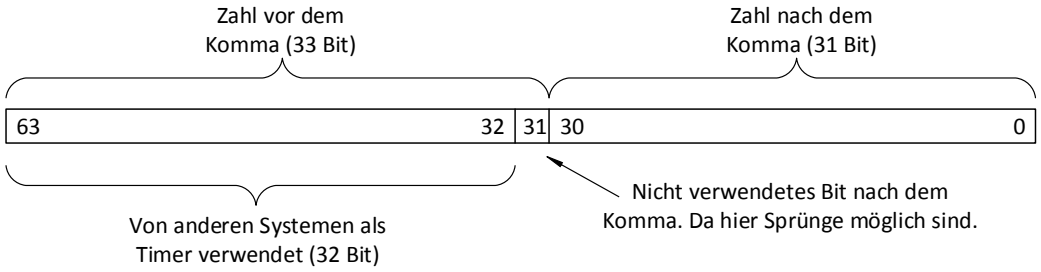

Abbildung 5.15: Fixed-Point-Timer Modul: Aufbau des internen Timers

Dem Modul wird über den Eingang *local_clock_addup_in* der Wert übermittelt, der pro Takt auf den Timer addiert werden soll. Das Signal *local_clock_addup_in* hat ein Bit vor dem Komma und 31 Bit nach dem Komma. Im Startzustand ist dieser Wert auf 1.0 gesetzt (0x80000000). Nur das oberste Bit ist auf 1 gesetzt, alle anderen Bits auf 0. Im Startzustand kann kein Übersprung passieren. Abbildung 5.16 zeigt ein Beispiel, bei dem der Timer 25% schneller gegenüber dem Startzustand läuft. Pro Takt wird der Wert 1.25 addiert (1.01 Binär). In diesem Beispiel wird alle vier Takte ein Wert übersprungen. So ist es möglich, den Timer schneller laufen zu lassen.

Für andere Module, die diesen Timer als Zeitgeber verwenden, können übersprungene Werte ein Problem darstellen. Würde ein anderes Modul zum Zeitpunkt 4 eine Aktion planen und warten bis dieser Timer die Zahl 4 liefert, so würde die Aktion nicht ausgeführt werden, da die Zahl 4 in diesem Beispiel übersprungen wurde. Die anderen Module könnten das Problem umgehen, indem sie „größer 3“ und „kleiner 6“ abfragen. Jedoch verbrauchen die Operatoren „kleiner“ und „größer“ viel Hardware, da diese oft als Volladdierer synthetisiert werden. Damit andere Module nicht mit „größer“ und „kleiner“ vergleichen müssen, sondern direkt auf Gleichheit vergleichen können, wird das Bit 31 von den anderen Modulen nicht

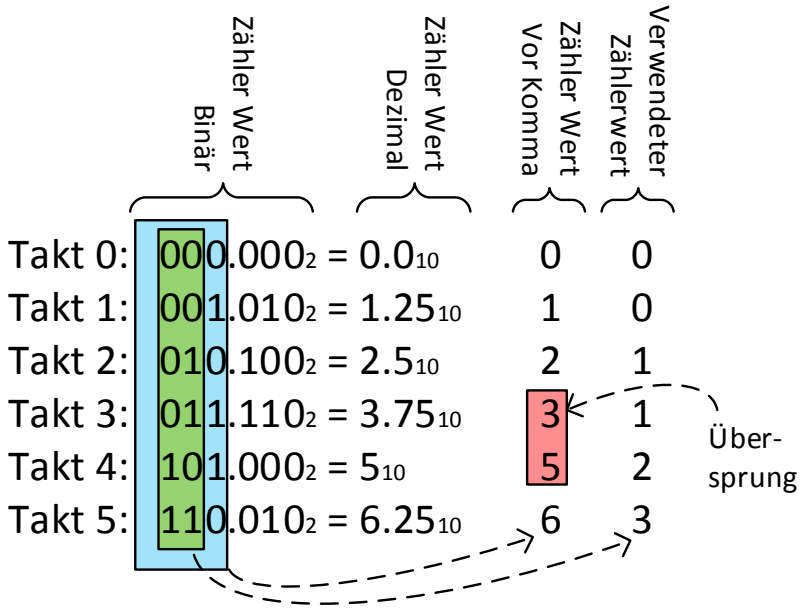

Abbildung 5.16: Beispiel Übersprung mit einem 25% schnellere Timer

verwendet. Da die Geschwindigkeit des Timers maximal kleiner 2.0 sein kann, wird es nur in Bit 31 einen Übersprung geben können. Die von anderen Modulen verwendeten Bits (63 bis 32) werden nie einen Übersprung haben und somit können andere Module ihre Zeiten immer auf Gleichheit prüfen. Durch diesen Mechanismus sparen andere Module viele Hardwareressourcen.

5.2.6.1 Hardwareressourcen

Das Fixed-Point-Timer-Modul braucht 64 Register und 160 LUTs.

5.2.7 Guard-Modul

Das Guard-Modul steuert das Senden von Time-Triggered Nachrichten und Best-Effort Nachrichten. Es sorgt dafür, dass Time-Triggered Nachrichten zum konfigurierten Zeitpunkt versendet werden und dass es keine Kollisionen zwischen Time-Triggered und Best-Effort Nachrichten gibt.

Das Guard-Modul wird durch einen Automaten gesteuert, der in Abbildung 5.17 dargestellt ist. Die Funktionalität wird im Folgenden beschrieben.

Folgender Parameter muss vor der Synthese in VHDL konfiguriert werden:

NUM_TT_PORTS Dieser Parameter gibt an, wie viele Eingänge für Puffer für die Nachrichtenklasse Time-Triggered erstellt werden sollen. Ein zusätzlicher Eingang für einen Puffer der Nachrichtenklasse BE wird immer erstellt.

Folgende Parameter müssen softwareseitig vor dem Start je Time-Triggered Eingangspuffer konfiguriert werden. Das x im Parameternamen ist ein Platzhalter für eine Eingangsportnummer. Diese Nummer beginnt bei 0 und endet bei NUM_TT_PORTS - 1.

tt_port_x_enable Gibt an, ob der Eingang x für den Puffer von Time-Triggered Nachrichten aktiv verwendet wird.

tt_port_x_begin Sendezeitpunkt der Time-Triggered Nachrichten aus dem Eingang x in der Einheit 20ns

tt_port_x_end Für diesen Parameter muss die Serialisierungszeit auf der Ethernetleitung der TT-Nachricht berechnet werden. Diese berechnete Zeit addiert mit dem Parameter tt_port_x_begin ergibt den Parameter tt_port_x_end.

Die Ports müssen aufsteigend sortiert nach dem Parameter tt_port_x_begin konfiguriert werden.

Folgende Parameter müssen unabhängig von den Time-Triggered Eingängen softwareseitig konfiguriert werden.

tt_cycle_duration Time-Triggerd Ethernet Zykluslänge in der Einheit 20ns.

tt_ifg Ethernet Inter-Frame-Gap (minimale Zeit zwischen den Ethernetpaketen). Einheit 20ns.

Im Folgenden werden die Zustände des Automaten aus Abbildung 5.17 beschrieben.

state_wait_for_cycle_start Dies ist der Startzustand. In diesen Zustand gerät der Automat beim Einschalten und nach einem Reset. In diesem Zustand wird auf den Anfang eines Zykluses gewartet. Wurde einer erkannt, wird in den Zustand *state_find_next_enabled_port* gewechselt.

state_find_next_enabled_port In diesem Zustand wird der nächste aktive Port ermittelt. Mit einem aktiven Port ist hier Port gemeint, bei dem der Parameter *tt_port_x_enable* auf 1 gesetzt ist. Ist Beispielsweise der Parameter *NUM_TT_PORTS* (vor der Synthese) auf 4 gesetzt, aber von der Anwendung werden nur zwei Ports benötigt, so werden die nicht verwendeten Ports übersprungen. Im ersten Durchlauf dieses Zustandes wird der erste Port als aktiver Port gefunden. Beim nächsten Durchlauf dieses Zustandes wird der nachfolgende Port als aktiver Port gewählt. Nachdem ein aktiver Port ermittelt

wurde, wird in den Zustand *state_wait* gewechselt. Wird kein aktiver Port gefunden, z. B. weil die Anwendung keine TT-Nachrichten versenden muss, wird in den Zustand *state_pure_be* gewechselt.

state_wait In diesem Zustand wird gewartet, bis entweder die Absendezeit des ermittelten aktiven Ports der lokalen Uhr (tt_timer) entspricht oder bis eine BE-Nachricht ansteht, deren Ethernet-Serialisierungszeit kleiner ist, als die Zeit bis zur Absendezeit des ermittelten aktiven Ports. Wenn die Absendezeit der TT-Nachricht erreicht ist, wird in den Zustand *state_send_tt_msg* gewechselt. Wenn eine versendbare BE-Nachricht bereitsteht, wird in den Zustand *state_send_be_msg* gewechselt. Der verwendete BE-Puffer ist ein Standard-Modul von Xilinx mit der Bezeichnung *LogiCORE IP XPS LL FIFO* (vgl. Xilinx Inc., 2005). Zur Übertragung wird das LocalLink Interface verwendet, dass im Abschnitt 2.2 auf Seite 11 beschrieben wird. Das LocalLink Protokoll hat einen Header. In diesen Header schreibt das *LogiCORE IP XPS LL FIFO* die Größe des zu übertragenen Paketes rein. Wenn eine BE-Nachricht bereit steht, ließt das Guard-Modul den Header aus. Dadurch weiß das Guard-Modul vor der Übertragung die Größe der BE-Nachricht und kann damit die Ethernet-Serialisierungszeit ausrechnen. Mit der ausgerechneten Zeit kann dann überprüft werden, ob die BE-Nachricht noch vor der nächsten TT-Nachricht versendet werden kann. Mit dieser Technik werden Kollisionen verhindert.

state_send_tt_msg In diesem Zustand wird die TT-Nachricht an das MAC Modul übertragen. Nach der Übertragung wird in den Zustand *state_block_all_after_tt* gewechselt.

state_block_all_after_tt In diesem Zustand wird gewartet bis der Zeitpunkt *tt_port_0_end* eintrifft. Die Differenz zwischen *tt_port_0_end* und *tt_port_0_begin* ist die Serialisierungszeit des Ethernetpakets. Diese Wartezeit ist notwendig, weil die Bandbreite zwischen Guard-Modul und MAC-Modul (3,2GBit/s) größer ist als die Ethernetbandbreite (in der Konfiguration 100MBit/s). Würde an dieser Stelle nicht gewartet werden, so gäbe es zwei unerwünschte Effekte. Würden mehrere Pakete im BE-Puffer anstehen, so würden diese alle direkt hintereinander an das MAC-Modul übertragen werden. In dem Fall könnte der FIFO-Speicher überfüllt werden und Pakete könnten verloren gehen. Der zweite unerwünschte Effekt ist, dass durch diese Wartezeit sichergestellt wird, dass TT-Nachrichten nach der Übertragung ins MAC-Modul dort nicht auf Grund eines gefüllten FIFO-Speichers warten müssten, bis diese versendet werden. Dies stellt sicher, dass hierdurch kein Jitter beim Versenden von TT-Nachrichten entsteht. Erreicht die lokale Zeit (tt_timer) den Wert block_end, wird in den Zustand *state_find_next_enabled_port* gewechselt.

state_send_be_msg In diesem Zustand wird eine BE-Nachricht aus dem BE-Puffer zum MAC-Modul übertragen. Nach der vollständigen Übertragung wird in den Zustand *state_block_all_after_be* gewechselt.

state_block_all_after_be In diesem Zustand wird gewartet, bis der Zeitpunkt *block_end* eintrifft. Das ist der Zeitpunkt, bei dem die Ethernet-Serialisierung abgeschlossen sein müsste. Der Grund für diese Wartezeit ist der Gleiche, wie schon im Zustand *state_block_all_after_tt* beschrieben. Ist der Zeitpunkt *block_end* erreicht, wird in den Zustand *state_find_next_enabled_port* gewechselt.

state_pure_be In diesen Zustand gelangt der Automat, wenn kein aktiver Port für TT-Nachrichten ermittelt werden konnte. Die Anwendung, also keine TT-Nachrichten übertragen muss. In diesem Zustand wird der BE-Puffer direkt mit dem MAC-Modul verbunden, so dass permanent BE-Nachrichten übertragen werden. Nur durch ein Resetsignal kann der Automat aus diesem Zustand in den Startzustand gebracht werden.

In diesem Abschnitt wurde die Funktionsweise des Guard-Moduls beschrieben. In Abschnitt 7.6 auf Seite 77 werden die Ergebnisse zu diesem Modul beschrieben.
Hardwareressourcen Ein Guard-Modul mit 4 Time-Triggered und einem Best-Effort Eingang braucht 85 Register und 596 LUTs.

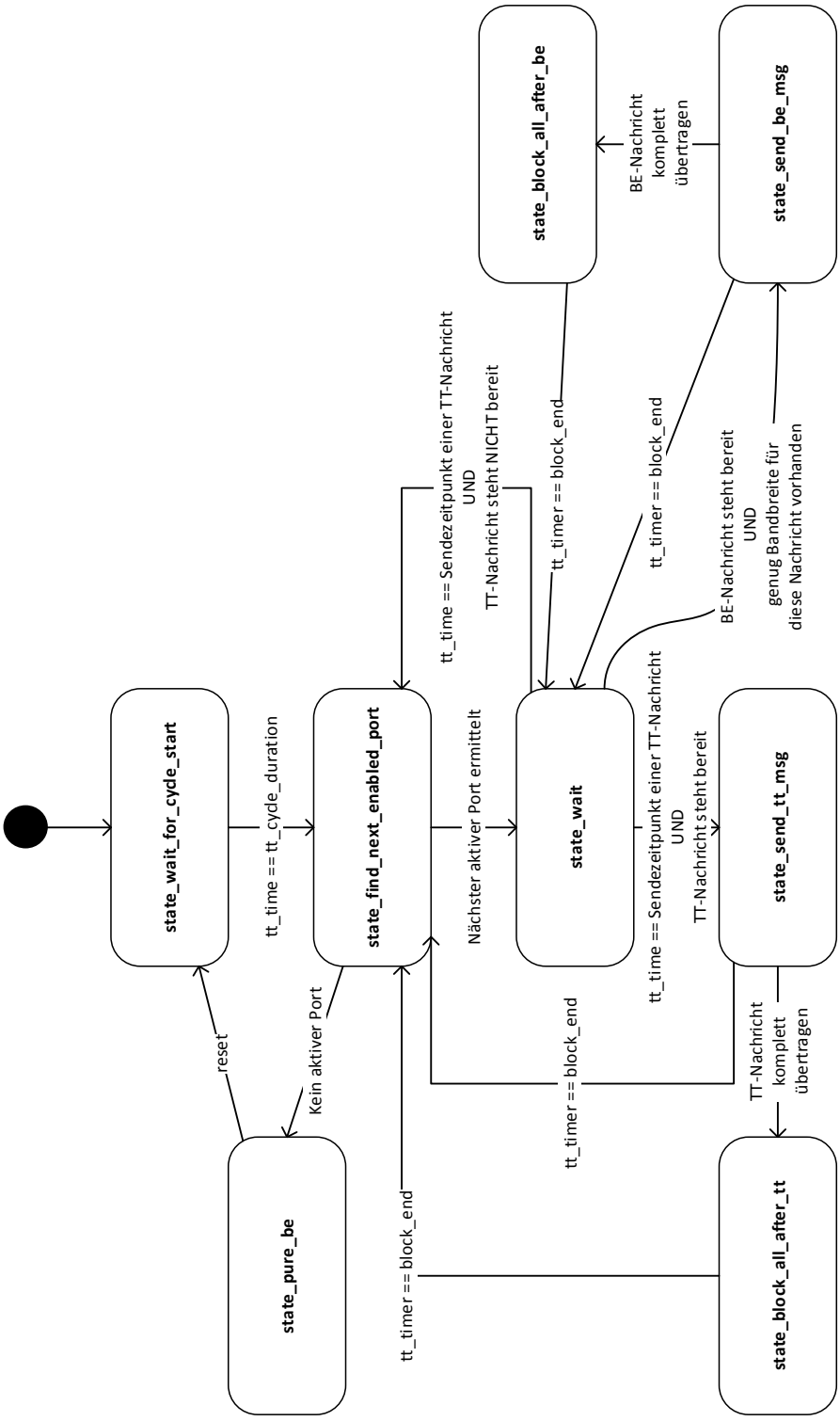

Abbildung 5.17: Guard-Modul Automat schematisch

Kapitel 6

Partitionierung

6.1 Anforderungskriterien

In diesem Abschnitt werden Anforderungskriterien definiert, die beim Hardware- Software Co-Design von Echtzeitsystemen bei der Partitionierung berücksichtigt werden müssen.

Erlaubter Jitter In diesem System bezieht sich der Jitter auf das Versenden von Time-Triggered Nachrichten in Bezug zur globalen Uhr.

Verfügbare CPU-Ressourcen Je höher der Rechenaufwand einer Anwendung ist, umso geringer sind die verfügbaren CPU-Ressourcen, die für das Time-Triggered Ethernet Protokoll übrig bleiben.

Verfügbare Chipfläche Entspricht den Kosten.

Benötigte Funktionalität Nicht in jeder Anwendung wird die komplette Funktionalität benötigt. Eine Aufteilung des Protokolls in mehrere Module soll ermöglichen, nur die nötigste Funktionalität zu implementieren.

Energie Stromverbrauch einer Partitionierung.

Im Folgenden wird auf die Anforderungskriterien genauer eingegangen.

Erlaubter Jitter

Diese Anforderung legt fest, wie stark ein System schwanken darf. Muss eine Aktion zum Zeitpunkt x erfolgen, dann beschreibt diese Anforderung, um wie viel Zeit die Aktion vor und nach dem Zeitpunkt x maximal ausgeführt werden darf. In diesem System geht es dabei um das Versenden von Time-Triggered Nachrichten. Diese müssen zu einem bestimmten

Zeitpunkt versendet werden. Für jede Nachricht wird dabei im Netzwerk ein Zeitfenster definiert, in dem der Teilnehmer diese Nachricht absenden darf. Die Größe des Zeitfensters wird durch die Größe der Nachricht und den Jitter des Senders bestimmt. Dabei versucht man, ein möglichst kleines Zeitfenster im Verhältnis der Nachricht zu erreichen, weil große Zeitfenster im Verhältnis zur Nachricht die reservierte Zeit im Netzwerk erhöhen und somit für diese Zeit keine weiteren Nachrichten über diesen Link versendet werden dürfen. Um die Bandbreite zu reduzieren muss die Partitionierung hardwarelastig erfolgen. Kann man sich hohe Belastung der Bandbreite erlauben, so kann softwarelastig partitioniert werden und somit Chipfläche gespart werden.

Verfügbare CPU-Ressourcen

Über die verfügbaren CPU-Ressourcen entscheidet die Anwendung, die auf der CPU ausgeführt wird. Ist der Rechenaufwand der Anwendung hoch, bleiben wenig CPU-Ressourcen für die Protokollausführung übrig und es muss hardwarelastig partitioniert werden. Bei geringem Rechenaufwand können mehr Module in Software partitioniert werden, um Chipfläche zu sparen.

Verfügbare Chipfläche (Kosten)

Um ein Gerät so günstig wie möglich zu realisieren, muss dieser Punkt bei der Suche nach dem optimalen Partitionierungsprofil mit berücksichtigt werden. So kann es sinnvoll sein, die Echtzeitanforderungen auf das Nötigste zu reduzieren, um möglichst viel in Software partitionieren zu können und so an Chipfläche zu sparen. Im Endefekt entscheiden die Echtzeitanforderungen und die Auslastung der CPU durch die Anwendung, wie die Kosten ausfallen werden und somit, ob mehr Chipfläche oder eine bessere CPU verwendet wird.

Benötigte Funktionalität

Um so wenig wie möglich CPU- und Hardwareressourcen zu verbrauchen, macht es in vielen Anwendungen Sinn, Funktionalität wegzulassen. Beispielsweise kann es sein, dass eine Anwendung nicht alle Nachrichtenklassen benötigt. Die entsprechenden Module müssen in diesem Fall nicht mit in die Partitionierung aufgenommen werden.

Energie

Ein geringer Stromverbrauch ist in den heutigen Fahrzeugen sehr wichtig. Dieser führt zu einer geringeren Belastung der Lichtmaschiene und somit auch zu einem geringeren Treibstoffverbrauch. Mikrocontroller bieten heute gute Möglichkeiten Strom zu sparen, indem Teile des Mikrocontrollers in einen Sleep-Modus gesetzt werden können. Diese Teile können

dann, wenn sie benötigt werden, geweckt werden und nach der Benutzung wieder in den Sleep-Modus gesetzt werden. Mehr Hardware führt oft dazu, dass mehr Strom verbraucht wird. Aus diesem Grund muss bei der Partitionierung auch auf den Energieverbrauch geachtet werden.

6.2 Partitionierung

In diesem Abschnitt werden die Auswirkungen der Hardware oder Software Partitionierung für die im Abschnitt 5.1 definierten Module beschrieben. Dabei wird beschrieben, wie sich die Partitionierung auf die Anforderungskriterien auswirkt.

6.2.1 Timestamping-Modul

Die Aufzeichnung des Zeitstempels zwischen Ethernet PHY (OSI-Layer 1) und Ethernet MAC (OSI-Layer 2) erzeugt den kleinsten Jitter (vgl. Weibel, 2005). Bei der in Kapitel 5 beschriebenen Implementierung beträgt der Jitter 20ns. Dies ermöglicht eine sehr genaue Offset-Korrektur bei der Zeitsynchronisation. Eine hohe Synchronisationsgenauigkeit wirkt sich positiv auf den Jitter für das Versenden von Time-Triggerd Nachrichten aus.

Würden die Zeitstempel in dem System, dass in Kapitel 5 beschrieben ist, in Software aufgezeichnet werden, würde die Synchronisierung um 5µs ungenauer werden. Dies wirkt sich direkt auf den Jitter beim Senden von TT- Nachrichten aus. Dies wurde auf folgender Grundlage berechnet.

- Die Ethernet MAC hat einen FIFO Eingangspuffer mit der Größe 2048 Byte.
- Ein Synchronisationsframe (PCF) ist 60 Byte groß.
- Nach dem Empfang eines PC-Frames können also noch 2048 - 60 = 1988 Byte in der FIFO vor dem PCF liegen, die zuerst übertragen werden.
- Der Datenbus ist 32 Bit breit. Also werden pro Takt 4 Byte übertragen.
- Es kann somit bis zu 497 Takte dauern, bis das PC-Frame übertragen wird.
- Die Frequenz beträgt 100Mhz. Also liegen zwischen jedem Takt 10ns.
- 497 Takte * 10ns = 4970ns $\approx$ 5µs.

Erst wenn das PCF in den nächsten Puffer geladen wurde, wird ein Interrupt erzeugt. Hier entstehen weitere Jitter, die stark von den Cache Einstellungen der CPU abhängig sind.

Besteht die Anforderung, einen niedrigen Jitter zu haben, so reicht es nicht nur, dieses Modul in Hardware zu implementieren. Bei einem System ohne Fixed-Point-Timer konnte ein Jitter,

bezogen auf die globale Uhr, gemessen werden, der ein Prozent von der Time-Triggered Ethernet Zykluszeit betrug. In diesem Fall war eine Zykluszeit von 5ms eingestellt. Der gemessene Jitter Betrug 50µs. Dieser konnte allein auf das Fehlen des Fixed-Point-Timers zurückgeführt werden. Mehr Details dazu werden in den nächsten Abschnitten beschrieben.

In Zusammenarbeit mit dem Switch-Modul ist es möglich, zu spät eintreffende TT-Nachrichten zu verwerfen, ohne dass die CPU davon etwas mitbekommt. Das Switch-Modul ließt dann den Eingangszeitstempel aus und kann dann anhand der CT-ID und des für die CT-ID konfigurierten Zeitfensters ermitteln, ob diese Nachricht zu einem gültigen Zeitpunkt angekommen ist. Bei einer Fehlfunktion eines anderen Netzwerkteilnehmers, der in einen Fehlerzustand gerät, bei dem permanent TT-Nachrichten versendet werden, kann diese Funktionalität die Eingangspuffer für die TT-Nachrichten davor schützen, überzulaufen und somit auch verhindern, dass die Software durch ständige Interrupts lahm gelegt wird.

Ergebnis Timestamping-Modul

Das Timestamping-Modul nimmt 5,3% der LUTs und 6,3% der Register der gesamten Time-Triggered Ethernet Implementierung ein und reduziert in diesem Design den Jitter beim Senden der Time-Triggered Nachrichten um mindestens 5µs.

6.2.2 Sync-Modul

Nach der AS6802 Spezifikation muss die Offset-Korrektur der Uhr innerhalb eines Time-Triggered Ethernet Zykluses vorgenommen werden. Die einzige Bedingung für den Zeitpunkt der Offset-Korrektur ist in Formel 2.4 auf Seite 8 beschrieben. Dieser Time-Triggered Ethernet Zyklus ist nicht fest definiert, beträgt aber in den bisher aufgebauten Systemen mehrere Millisekunden.

Der Korrekturwert wird u. a. anhand des Eingangszeitstempels des Synchronisationspakets berechnet. Wenn die Korrektur spät erfolgt, wird die diese nicht exakt vorgenommen, da die lokale und die globale Uhr wieder auseinander gelaufen sind (siehe Abbildung 6.1). Bei einem Versuchsaufbau mit einem Mikrocontroller-Board und einem FPGA-Board wurde bei den internen Timern eine Geschwindigkeitsdifferenz von einem Prozent gemessen. Der Synchronisationsalgorithmus musste in diesem Versuch die Uhr des Synchronisations-Clients um ein Prozent der Zykluszeit korrigieren. Bei einer Zykluszeit von 5ms entspricht dies einer Offsetkorrektur von 50µs. Diese Abweichung wird jedoch zum Eingangszeitpunkt des Synchronisationspakts gemessen, was zu Beginn des Time-Triggered Ethernet Zykluses ist, da zu Beginn die Synchronisation eingeleitet wird. Wird nun in Software korrigiert, wird die Korrektur nicht direkt beim Empfang vorgenommen. Wird nun im Extremfall am Ende der Zykluszeit korrigiert, hat man eine weitere Ungenauigkeit hinzubekommen. Abbildung 6.1

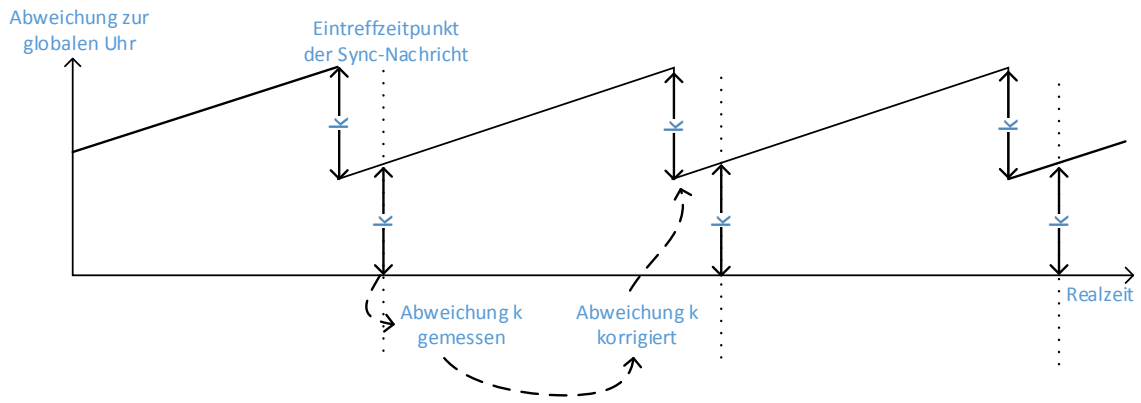

Abbildung 6.1: Erhöhter Fehler bei später Offset-Korrektur

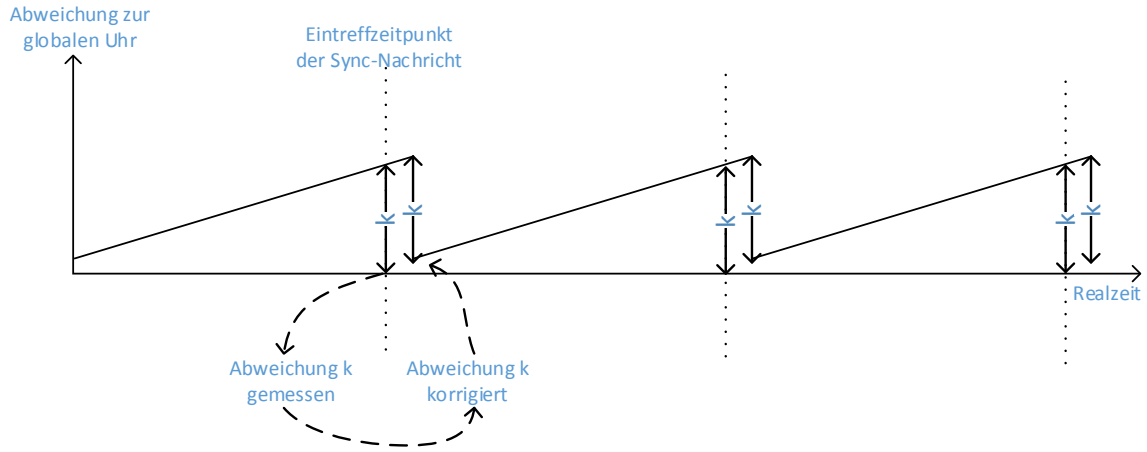

Abbildung 6.2: Geringer Fehler bei früher Offset-Korrektur

soll diesen Extremfall verdeutlichen. Dies führt bei dem genannten Versuch im schlechtesten Fall zu einer Gesamtungenauigkeit von 2% der Zykluszeit, in diesem Beispiel also 100µs entspricht. Mit einer Hardwareimplementierung könnte immer direkt nach dem Empfang korrigiert und so sichergestellt werden, dass eine bestimmte maximale Ungenauigkeit nicht überschritten wird. Abbildung 6.2 soll dies verdeutlichen. In dem eben beschriebenen Versuch wären dies 50µs. Diese Ungenauigkeit wirkt sich direkt auf den Jitter beim versenden von Time-Triggered Nachrichten aus.

Die eben beschriebenen Korrekturwerte entstehen nur, wenn der Synchronisations-Client keinen in der Geschwindigkeit anpassbaren Timer hat. Solch ein Timer reduziert den Korrekturwert und somit den Jitter erheblich. Im nächsten Abschnitt wird solch ein Timer

beschrieben.

Wenn das Timestamping-Modul nicht in Hardware implementiert wird, kann das Sync-Modul nur in Software implementiert werden, da die Eingangszeitstempel der Synchronisationsnachrichten vorher in Hardware aufgezeichnet werden müssen.

Ein weiterer Vorteil einer Hardwareimplementierung ist die Einsparung von Energie, die sich in bestimmten Fällen ergibt. Nicht alle Steuergeräte müssen, während ein Automobil in Betrieb ist, durchgehen eingeschaltet sein. Beispielsweise die Einparkautomatik oder eine Rückfahrkamera werden nur während des Einparkvorgangs benötigt. Auch gibt es Steuergeräte, die in größeren Zykluszeiten Berechnungen durchführen müssen. In diesen Fällen kann die CPU in einen Sleep-Modus versetzt werden, um Energie zu sparen. Ist die Synchronisation nun in Software implementiert, muss das Steuergerät in jedem Time-Triggered Ethernet Zyklus aus dem Sleep-Modus geweckt werden, um die Synchronisation durchzuführen. Bei einer Hardwareimplementierung, die die Synchronisation vollständig übernimmt, ist dies nicht mehr notwendig. So muss nur noch ein kleiner Teil der Hardware mit Strom versorgt werden.

Ergebnisse Sync-Modul

Die AS6802 Implementierung des Sync-Modul nimmt 53,3% der LUTs und 64,8% der Register einer gesamten Timer-Triggered Ethernet Hardwareimplementierung ein. Die Light Implementierung nimmt 36,9% der LUTs und 47,7% der Register einer gesamten Timer-Triggered Ethernet Hardwareimplementierung ein. Der Jitter wird durch eine schnellere Offset-Korrektur reduziert. Bei Steuergeräten die einen Großteil der Zeit in den Sleep-Modus gebracht werden können, verbessert die Hardwareimplementierung die Energiebilanz.

6.2.3 Fixed-Point-Timer

Der Fixed-Point-Timer ist ein Timer, bei dem die Geschwindigkeit eingestellt werden kann. Dessen Funktion ist in Abschnitt 5.2.6 auf Seite 51 beschrieben. Eine Implementierung des Timers in Software ist nicht möglich.

Wie in dem Abschnitt zuvor erwähnt, kann die Verwendung eines Fixed-Point-Timers den Offset-Korrekturwert des Sync-Moduls erheblich reduzieren. Die Schwankung eines Quarzes liegt je nach Qualität zwischen 0.001% und 0.01%. Die Abweichung zweier Quarze zueinander kann jedoch im einstelligen Prozentbereich liegen. Im optimalen Fall kann der Offset-Korrekturwert auf die Schwankung der Quarze optimiert werden. Damit sorgt dieses Modul auch dafür, dass die Uhren über den gesamten Time-Triggered Ethernet Zyklus hin-

weg annähernd die gleiche Geschwindigkeit haben. Dies wirkt sich direkt auf den Jitter beim versenden von Time-Triggered Ethernet Nachrichten aus.

Ergebnisse Fixed-Point-Timer-Modul

Das Fixed-Point-Timer-Modul nimmt 7,7% der LUT's und 5,6% der Register einer gesamten Timer-Triggered Ethernet Hardwareimplementierung ein. Es reduziert den Jitter erheblich und kann bei Steuergeräten, die nicht ständig in Betrieb sein müssen, die Vorteile eine Hardwareimplementierung des Sync-Moduls aufheben, da der Offset-Korrekturwert stark verkleinert wird. Das Sync-Modul nimmt im Vergleich zu diesem Modul wesentlich mehr Hardware ein.

6.2.4 Switch

Eine Hardwareimplementierung dieses Moduls erkennt die Nachrichtenklasse der eintreffenden Ethernetpakete und leitet diese an separate Puffer weiter. Mit dieser Funktionalität wird die CPU am meisten entlastet, da die Software jetzt nicht permanent für das Aufräumen des Speichers zur Verfügung stehen muss. Eintreffende BE-Pakete können nicht mehr dafür sorgen, dass der Speicher für RC- und TT-Pakete überfüllt wird und auf Grund der Überfüllung RC- oder TT-Pakete verloren gehen könnten. Es können durch zu viele BE-Pakete nur noch weitere BE-Pakete verloren gehen.

Wie in einigen Abschnitten schon erläutert, wurde ohne solch ein Hardwaresupport die meisten CPU-Ressourcen verbraucht. In der Software Implementierung von (vgl. Müller u. a., 2011) wurde eine CPU Belastung von 90% gemessen. Dies gilt jedoch nicht in jedem Fall. Beim Weglassen der Nachrichtenklasse RC auf der Empfangsseite kann die CPU Belastung einer reinen Softwareimplementierung reduziert werden. Näheres dazu wird in Abschnitt 3.2 auf Seite 15 beschrieben. In diesem Fall werden die Vorteile einer Hardwareimplementierung zum Teil aufgehoben.

Ergebnisse Switch-Modul

Das Switch-Modul nimmt 4,8% der LUT's und 15,8% der Register einer gesamten Timer-Triggered Ethernet Hardwareimplementierung ein. Ein weiterer Hardwarebedarf entsteht durch die Verwendung mehrerer Eingangspuffer. Dies wurde bei der Berechnung nicht mit berücksichtigt. Das Modul entlastet die CPU am stärksten.

6.2.5 Guard

Die Hardwareimplementierung dieses Moduls stößt den Sendevorgang von TT-Nachrichten an und stellt dabei sicher, dass die verschiedenen Nachrichtenklassen nicht miteinander kollidieren. Dies führt dazu, dass die Software die Datenpakete nun asynchron in den Sendepuffer ablegen kann und zum Sendezeitpunkt nicht nochmal aktiv werden muss. Dies ist insbesondere bei der Verwendung von bestimmten Betriebssystemen wie, beispielsweise AUTOSAR wichtig. AUTOSAR bietet in der aktuellen Version keinen Schedulingmechanismus um Tasks zeitgesteuert auszuführen.

Mit einer Hardwareimplementierung dieses Moduls kann die Software alle Nachrichtenklassen beim Versenden gleich behandeln, was bedeutet, dass diese asynchron in die Sendepuffer abgelegt werden können. Dies reduziert den Migrationsaufwand bestehender Anwendungen.

Ein weiterer Vorteil ist eine höhere Genauigkeit beim Senden. Ein Hardwaremodul kann einen Sendevorgang nahezu ohne Jitter anstoßen. Bei der Implementierung, die im Kapitel 5 beschrieben ist, beträgt der Jitter bezogen auf die lokale Uhr 40ns. In der Softwareimplementierung (vgl. Müller u. a., 2011) beträgt der Jitter bezogen auf die lokale Uhr 350ns.

Ergebnisse Guard-Modul

Das Guard-Modul nimmt 28,9% der LUT's und 7,5% der Register einer gesamten Timer-Triggered Ethernet Hardwareimplementierung ein. Ein weiterer Hardwarebedarf entsteht durch die Verwendung mehrerer Sendepuffer. Dies wurde bei der Berechnung nicht mit berücksichtigt. Das Modul ermöglicht der Software, TT-Pakete asynchron in den Sendepuffer abzulegen und reduziert den Jitter beim versenden von TT-Nachrichten.

6.2.6 Zusammenfassung der Einzelbetrachung

In der Tabelle 6.1 werden die Ergebnisse zusammengefasst. Es wird zu jedem Modul dargestellt, welche Anforderungskriterien durch eine Partitionierung in Hardware verbessert oder verschlechtert werden. Dies kann jedoch nicht allgemein gesehen werden, da es einige Spezialfälle gibt, die in den vorangegangenen Abschnitten beschrieben sind.

	Time-stamping	Sync	Fixed-Point-Timer	Switch	Guard
Jitter	+ + +	+	+ + +		+ +
Verfügbare Chipfläche	-	- - -	-	- -	- -
Verfügbare CPU-Ressourcen	+	+		+ + +	+
Energiebilanz	-	+ + +	-	-	

Tabelle 6.1: Auswirkungen der Partitionierung in Hardware auf die Anforderungskriterien. Subjektiv dargestellt.
Die Anzahl der Plus-Zeichen soll die positive Auswirkung verdeutlichen, die Minus-Zeichen die negative Auswirkung

6.2.7 Abhängigkeiten von Modulen

In diesem Abschnitt werden Abhängigkeiten zwischen den Modulen beschrieben. Zunächst werden die technisch unmöglichen Partitionierungen beschrieben und anschließend die wenig sinnvollen Partitionierungen.

6.2.7.1 Technisch unmögliche Partitionierungen

Sync und Timestamping

Das Sync-Modul nimmt Synchronisationspakete auf, die einen Eingangszeitstempel haben. Ist das Sync-Modul in Hardware partitioniert, erhält das Hardwaremodul direkt die Synchronisationspakete, ohne dass diese vorher per Software bearbeitet werden. Das Timestamping-Modul muss also bei einer Hardwareimplementierung des Sync-Moduls auch in Hardware implementiert werden, da nur hier die Chance besteht, einen Eingangszeitstempel dem Synchronisationspaket zu geben.

Aus (Hardwareimplementierung Sync) folgt (Hardwareimplementierung Timestamping).

Sync und Switch

Wie eben beschrieben, erhält eine Hardwareimplementierung des Sync-Moduls direkt die Synchronisationspakete, ohne dass diese vorher von der Software empfangen werden. Es muss also ein Hardwaremodul geben, dass die Synchronisationspakete aus dem Datenverkehr filtert und an das Sync-Modul weiterleitet. Hierfür wird das Switch-Modul verwendet. Daraus folgt:

Aus (Hardwareimplementierung Sync) folgt (Hardwareimplementierung Switch).

6.2.7.2 Wenig sinnvolle Partitionierung

Timestamping und Fixed-Point-Timer

Die Partitionierung des Timestamping-Moduls in Hardware reduziert, in der in dieser Arbeit vorgestellten Hardwareimplementierung, den Jitter um 5µs. Wird das Ziel verfolgt, eine Partitionierung zu finden, bei der der Jitter im kleinen einstelligen Mikrosekundenbereich liegen soll, macht es keinen Sinn, das Fixed-Point-Timer Modul nicht mit in Hardware zu implementieren, da das Fehlen des Fixed-Point-Timers einen Jitter im zweistelligen Mikrosekundenbereich verursacht. Umgekehrt verhält es sich ähnlich. Hier macht die alleinige Hardwareimplementierung des Fixed-Point-Timer-Moduls nur in dem speziellen Fall Sinn, wenn der Jitter im Bereich von 5µs liegen soll, also nicht im zweistelligen Mikrosekundenbereich. Daraus folgt:

Ist ein Jitter im kleinen einstelligen Mikrosekundenbereich gefordert, so müssen das Timestamping-Modul und das Fixed-Point-Timer-Modul in Hardware partitioniert werden.

6.2.8 Partitionierungsbeispiele

In diesem Abschnitt werden sinnvolle Partitionierungsbeispiele gezeigt.

6.2.8.1 Synchronisation in Software

Abbildung 6.3 zeigt eine Partitionierung bei der das Sync-Modul in Software implementiert wurde und alle anderen Module in Hardware. Zwischen dem Zeitpunkt, an dem der Eingangszeitstempel aufgezeichnet wurde und der Offset-Korrektur, vergeht bei einer Software-Implementierung mehr Zeit, als bei einer Hardwareimplementierung. In dieser Zeit kann die lokale Uhr eine weitere Abweichung zur globalen Uhr entwickeln. Diese Mehrabweichung kann dann bei der Offset-Korrektur nicht mehr berücksichtigt werden und führt dazu, dass die Uhr etwas ungenauer synchronisiert wird. Durch die Hardwareimplementierung des Fixed-Point-Timers kann diese Abweichung sehr gering gehalten werden, da dieser die Geschwindigkeit der lokalen Uhr an die Geschwindigkeit der globalen Uhr angleicht.

Bei dieser Partitionierung kann die Energiebilanz nicht verbessert werden, da nun die CPU nicht über längere Zeiträume in den Sleep-Modus gesetzt werden kann. Die Synchronisation findet in Software statt. Daher muss die CPU diese einmal pro Time-Triggered Ethernet Zyklus ausführen.

Die AS6802 Implementierung des Sync-Moduls verbraucht 53,3% der LUTs und 64,8% der Register einer gesamten Time-Triggered Ethernet Hardwareimplementierung. Wenn dieses

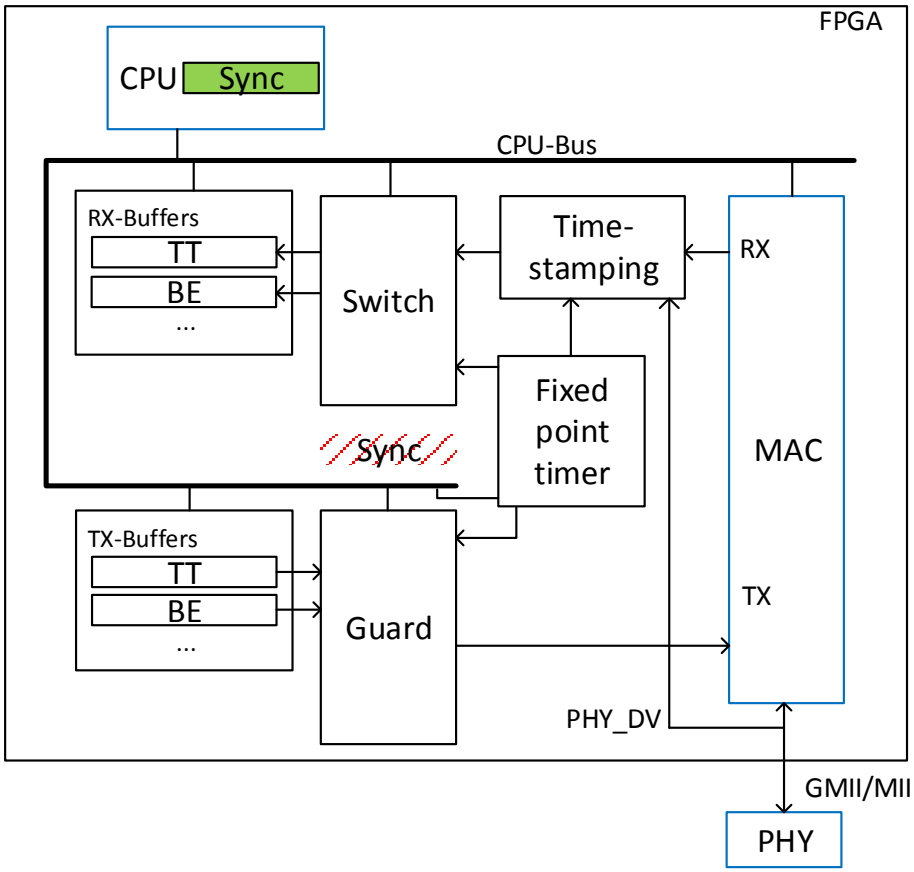

Abbildung 6.3: Partitionierungsbeispiel bei dem in Software Synchronisiert wird. *Das Sync-Modul wurde aus der Hardware entfernt (rot durchgestrichen) und in Software verlagert (grün hinterlegt).*

Modul in Software implementiert wird, werden also rund 60% der Hardwareressourcen eingespart. Bei Steuergeräten, die auf Grund der Anwendung sowieso nicht in den Sleep-Modus versetzt werden können und eine kleine Ungenauigkeit in der Synchronisierung hingenommen werden kann, wird diese Partitionierung die meisten Vorteile bringen.

6.2.8.2 Echtzeitnachrichten senden, aber nicht empfangen

Einige Steuergeräte haben Anwendungen die Echtzeitnachrichten versenden, aber nicht empfangen. Beispielsweise sendet das Steuergerät einer Rückfahrkamera Bilddaten mit Echtzeitnachrichten an den Bildschirm im Fahrzeugcockpit. Die Rückfahrkamera selber muss keine Echtzeitnachrichten empfangen. In diesem Fall kann das Switch-Modul und die getrennten Empfangspuffer eingespart werden und nur ein Empfangspuffer für Best-Effort

Nachrichten eingefügt werden.

Abbildung 6.4 zeigt eine Partitionierung bei der das Switch-Modul nicht implementiert ist. Wie in Abschnitt 6.2.7.1 auf Seite 65 beschrieben, muss dann das Sync-Modul in Software partitioniert werden, da eine Sync-Hardwareimplementierung an dem Switch-Modul angeschlossen werde muss. Die getrennten RX-Buffer wurden entfernt und durch eine RX-FIFO ersetzt. Als RX-FIFO kann dann z. B. das von Xilinx bereitgestellte Modul *LogiCORE IP XPS LL FIFO* (vgl. Xilinx Inc., 2011) verwendet werden.

Das Switch-Modul verbraucht 4,8% der LUTs und 15,8% der Register einer gesamten Time-Triggered Ethernet Hardwareimplementierung. Das Sync-Modul verbraucht 53,3% der LUTs und 64,8% der Register. Wenn ein Steuergerät keine Echtzeitnachrichten empfängt, eine geringe Ungenauigkeit in der Synchronisierung und eine verschlechterte Energiebilanz in Kauf genommen werden kann, so können mit dieser Partitionierung 58,1% der LUTs und 80,6% der Register eingespart werden. Aufgrund der weggefallen getrennten Puffer werden noch mehr Hardwareressourcen eingespart. Da der Hardwareverbrauch der Speicher in der gesamten Arbeit nicht berücksichtigt wird, wird dies hier auch nicht mit eingerechnet.

6.2.8.3 Echtzeitnachrichten empfangen, aber nicht senden

Einige Steuergeräte haben Anwendungen die Echtzeitnachrichten empfangen, aber nicht senden. Beispielsweise empfängt das Steuergerät, das Bilddaten der Rückfahrkamera im Cockpit anzeigt, die Bilddaten mit Echtzeitnachrichten, muss aber selbst keine Echtzeitnachrichten versenden. In diesem Fall kann das Guard-Modul und die getrennten Sendepuffer eingespart werden und nur ein Sendepuffer für Best-Effort Nachrichten eingefügt werden.

Abbildung 6.5 zeigt eine Partitionierung, bei der das Guard-Modul nicht implementiert ist. Die getrennten TX-Buffer wurden entfernt und durch eine TX-FIFO ersetzt. Hierfür kann dann z. B. das von Xilinx bereitgestellte Modul *LogiCORE IP XPS LL FIFO* (vgl. Xilinx Inc., 2011) verwendet werden.

Das Guard-Modul verbraucht 28,9% der LUTs und 7,5% der Register einer gesamten Time-Triggered Ethernet Hardwareimplementierung. Wenn ein Steuergerät keine Echtzeitnachrichten sendet, kann diese Hardware eingespart werden. Aufgrund der weggefallenen getrennten Puffer werden noch mehr Hardwareressourcen eingespart. Da der Hardwareverbrauch der Speicher in der gesamten Arbeit nicht berücksichtigt wird, wird dies hier auch nicht mit eingerechnet.

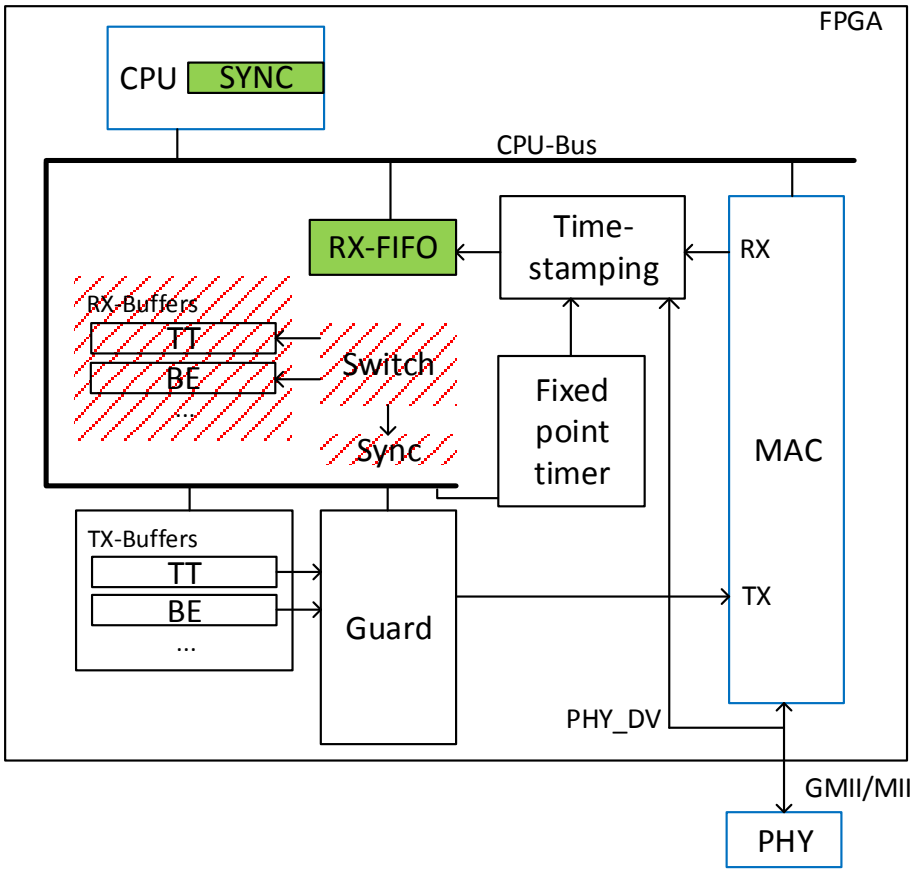

Abbildung 6.4: Partitionierungsbeispiel für Geräte die Echtzeitnachrichten Senden, aber nicht Empfangen.
Das Sync-, Switch- und RX-Buffers-Modul wurde aus der Hardware entfernt (rot durchgestrichen). Das Sync-Modul wurde in Software implementiert und ein einfacher RX-FIFO in Hardware hinzugefügt (beides grün hinterlegt).

Anmerkung

Diese Partitionierung kann auch verwendet werden, um Echtzeitnachrichten zu versenden. Es muss mit einem höheren Jitter gerechnet werden, da in Software viele Faktoren, wie z. B. CPU-Caches, zu einem hohen Jitter führen. Auch kann dann in Software nicht mehr asynchron gesendet werden. Die Software muss außerdem aufpassen, dass Best-Effort und TT-Nachrichten nicht kollidieren. Daher empfiehlt sich diese Partitionierung nicht, wenn Echtzeitnachrichten versendet werden sollen.

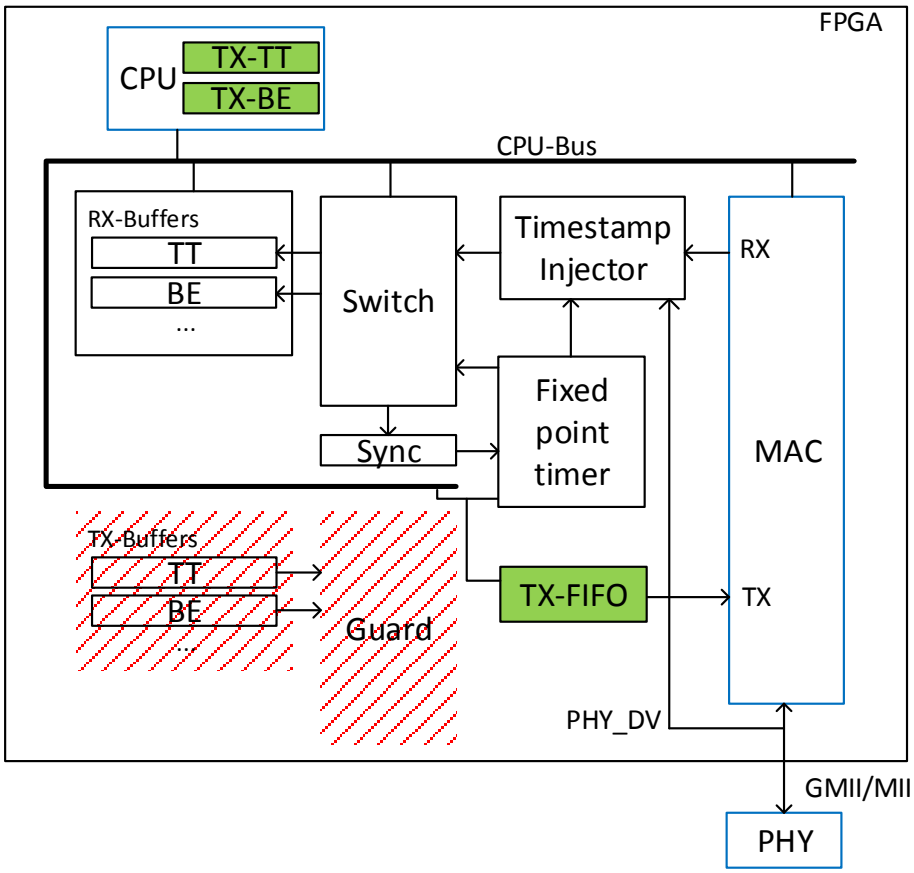

Abbildung 6.5: Partitionierungsbeispiel für Geräte die Echtzeitnachrichten Empfangen, aber nicht Senden
Das Guard-Modul und die TX-Buffer wurden aus der Hardware entfernt (rot dargestellt). Das Modul TX-FIFO, dass als Sendepuffer dient, wurde hinzugefügt (grün dargestellt).

Kapitel 7

Qualität und Evaluation

In diesem Kapitel wird die Qualität bezogen auf den Jitter beschrieben, errechnet und mit Messungen evaluiert. Der Jitter gibt die Schwankung eines Systems an. Bei der Hardwareimplementierung und bei den Hardware- Software Partitionierungen kann dies mehrere Bedeutungen haben. Diese werden im Folgenden beschrieben.

Jitter bezogen auf ein Endgerät in Bezug zur lokalen Uhr: Hierbei bezieht man die Schwankung eines Systems, bezogen auf die lokale Uhr. Wenn das Sendesteuerungsmodul Guard in Hardware implementiert ist, kann die Aussage getroffen werden, dass beim Senden der Time-Triggered Nachrichten ein geringer Jitter entsteht. In der Guard-Implementierung, die in Abschnitt 5.2.7 beschrieben ist, sind dies 40ns. Diese Aussage ist jedoch nicht viel wert, wenn der Rest des Systems so partitioniert ist, dass die lokale Uhr mit einer hohen Ungenauigkeit auf die globale Uhr synchronisiert wird. In diesem Fall ist der Jitter beim Senden von Time-Triggered Nachrichten bezogen auf die globale Uhr trotzdem groß, obwohl der Jitter beim Senden der Nachrichten bezogen auf die lokale Uhr sehr klein ist. Da andere Netzwerkteilnehmer nur die globale Uhr kennen, und nicht die lokale Uhr der anderen Teilnehmer, muss in kommunizierenden Systemen immer der Jitter bezogen auf die globale Uhr betrachtet werden.

Betrachtet man den Jitter bezogen auf die globale Uhr, so ist dieser von allen Modulen abhängig, die in dieser Arbeit beschrieben wurden. Alle implementierten Module haben entweder mit der Synchronisation, dem weiterleiten von Synchronisationsnachrichten oder dem Versenden von Nachrichten zu tun. Der gesamte Jitter setzt sich aus folgenden Teilen Zusammen:

Verzögerung von Synchronisationsnachrichten Mit dieser Verzögerung ist die Zeit zwischen dem Aufzeichnen eines Zeitstempels und der Durchführung der Offset-Korrektur gemeint. Diese Verzögerung erhöht den Jitter, da innerhalb dieser Zeit die lokale und globale Uhr aufgrund der unterschiedlichen Geschwindigkeiten einen Feh-

ler erzeugen. Die Zeit der Verzögerung ist nicht der Wert, um den der Jitter erhöht wird. Allgemein kann dies mit der Gleichung 7.1 ausgedrückt werden. Wenn die Geschwindigkeiten der Uhren beispielsweise um ein Prozent auseinander liegen ($local\_global\_ungenauigkeit$) und die Synchronisationsnachricht um 50µs verzögert wird $t\_verzoegerung$, dann beträgt der Jitter der hieraus erzeugt wird 500ns ($jitter\_verzoegerung$).

$$jitter\_verzoegerung = t\_verzoegerung * local\_global\_ungenauigkeit \quad (7.1)$$

Jitter beim Aufzeichnen der Ankunftszeitstempel Dieser Jitter wirkt sich direkt auf den Gesamtjitter aus, da der Ankunftszeitstempel zur Berechnung der Offset-Korrektur verwendet wird.

Jitter beim versenden von Time-Triggered Nachrichten Dieser Jitter wirkt sich auch direkt auf den Gesamtjitter aus.

Warum die ersten beiden genannten Punkte zum Jitter beitragen, wird in Abschnitt 6.2.2 auf Seite 60 detaillierter beschrieben. Im Folgenden wird der Jitter, den die einzelnen Module erzeugen, beschrieben.

7.1 MAC-Modul (Empfangsseite)

Variable Verzögerung von Synchronisationsnachrichten

Das MAC-Modul wurde nicht in dieser Arbeit entwickelt. Es wurde dafür die von Xilinx bereitgestellte *LogiCORE IP XPS LL TEMAC* (vgl. Xilinx Inc., 2010) verwendet. Das MAC-Modul hat einen 2048 Byte großen internen FIFO-Speicher. Je nach Füllstand des Speichers, werden eintreffende Synchronisationsnachrichten verzögert. Ein Synchronisationspaket ist 60 Byte groß. Nach Eintreffen eines Synchronisations Frames können also 0 bis 1988 Byte (2048 - 60) im FIFO-Speicher liegen, die vor dem Synchronisationspaket übertragen werden. Das implementierte System taktet mit 100 Mhz (alle 10ns ein Takt). Pro Takt werden 4 Byte an das nächste Modul übertragen. Wenn 1988 Byte vor der Synchronisationsnachricht liegen, braucht es also 497 Takte (1988 Byte / 4 Byte pro Takt), bis die Nachricht übertragen wird. Das ergibt eine Verzögerung von 4970ns. Wenn der FIFO-Speicher nicht gefüllt ist, entsteht durch den FIFO-Speicher keine Verzögerung.

Somit ergibt sich für das MAC-Modul eine maximale Verzögerung von Synchronisationsnachrichten von 4970 ns.

Konstante Verzögerung von Synchronisationspaketen

Im Folgenden wird von einer Ethernetbandbreite von 100 Mbit/s ausgegangen. Die konstante Verzögerung eines Synchronisationpakets setzt sich aus drei Teilen zusammen.

- Deserialisierungszeit beim Empfang
- Verweildauer im MAC-Modul
- Versenden des Pakets an ein anders Modul über die LocalLink Schnittstelle.

Die Deserialisierungszeit beim Empfang und die Verweildauer wurde mit einem Oszilloskop gemessen. Abbildung 7.1 zeigt den Messaufbau. Das SOF-Signal (start of frame) wird von der LocalLink Schnittstelle verwendet, um den Beginn eines Frames zu signalisieren (siehe Abschnitt 2.2 auf Seite 11. Dieses Signal wurde an einen Pin des FPGA's ausgeführt und als Messpunkt verwendet. Als zweiter Messpunkt wurde direkt die Ethernetleitung verwendet. Die gemessene Zeitdifferenz beträgt 6880ns und hat einen Jitter von 80ns (Maximalwert 6960ns). Bei der Messung wurde sichergestellt, dass nur Synchronisationspakete von dem MAC-Modul empfangen werden. So konnte sichergestellt werden, dass beim Eintreffen von Synchronisationsnachrichten der FIFO-Speicher leer war und dadurch nicht noch ein weiterer Jitter hinzukam. Zur der gemessenen Verzögerung kommt noch die Versendezeit auf der LocalLink Schnittstelle hinzu. Diese Schnittstelle wird mit 100MHz getaktet. Pro Takt werden 4 Byte versendet. Jedem Paket wird noch ein 4 Byte großer LocalLink-Header angehängt. Bei der Serialisierung müssen also 60 Byte des Ethernetpakets und 4 Byte des LocalLink Headers berücksichtigt werden. (64 Byte) / (4 Byte pro Takt) = 16 Takte. Bei einem Takt von 100MHz ergibt dies eine Verzögerung von 170ns.

Somit ergibt sich für das MAC-Modul eine konstante Verzögerung von
$6960ns + 170ns = 7130ns$.
Die konstante und variable Verzögerung von Synchronisationspaketen ergeben zusammen eine maximal mögliche Verzögerung von
$7130ns + 4970ns = 12100ns$

7.2 Timestamping-Modul

7.2.1 Genauigkeit der Ankunftszeitstempel

Das Untermodul *Zeitstempelaufnahme* ist für das Aufzeichnen der Ankunftszeit zuständig. Wie schon in Abschnitt 5.2.3 auf Seite 33 beschrieben wurde, wird der Zeitstempel anhand des PHY_DV Signals der MII Schnittstelle vorgenommen. Bei diesem Signal konnte kein Jitter gemessen werden, jedoch muss aufgrund der Auflösung des Fixed-Point-Timers ein

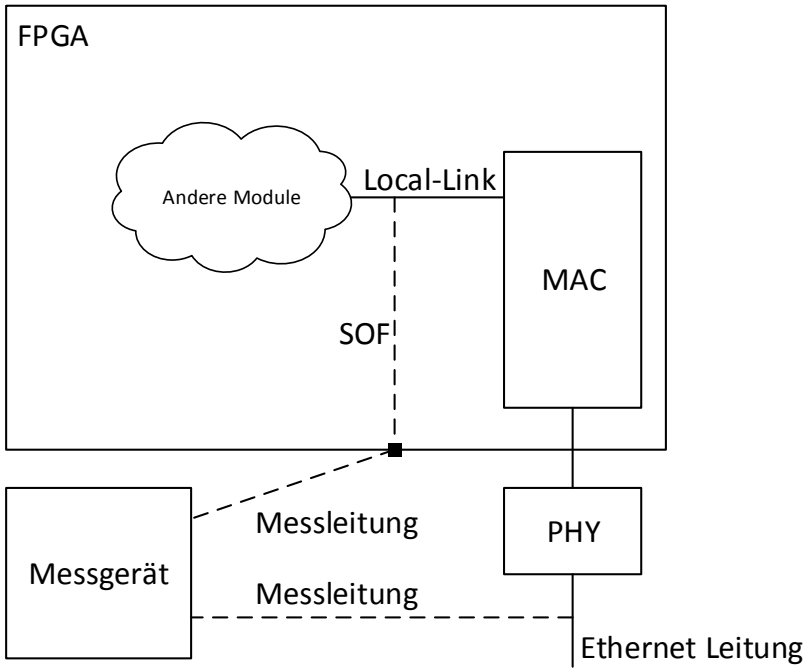

Abbildung 7.1: Messaufbau zur Messung der Latenz im MAC-Modul
Gestrichelt dargestellte Linien sind Leitungen, die zur Messung verlegt wurden.

Jitter von 20ns angenommen werden. Das PHY_DV Signal wechselt von 0 auf 1 390ns vor der Deserialisierung des ersten Bits der Ziel-MAC-Adresse. Theoretisch müsste diese Verzögerung mit der Gleichung 7.1 berücksichtigt werden. Das Ergebnis ist allerdings kleiner als die Auflösung des Fixed-Point-Timers und kann deswegen vernachlässigt werden.

Damit ergibt sich für die Genauigkeit der Ankunftszeitstempel ein Wert von 20ns.

7.2.2 Konstante Verzögerung von Synchronisationspaketen

Das MAC-Modul sendet empfangene Ethernetpakete über das LocalLink Interface an das Switch-Modul. Damit zu diesem Zeitpunkt Zeitstempel in den Ethernetpaketen enthalten sind, wurde das Untermodul *Zeitstempelschleuse* zwischen dem MAC-Modul und dem Switch-Modul platziert, welches die Zeitstempel „on the fly“ in den Payload-Teil des Local-Link Protokolls einfügt. Wie schon in Abschnitt 5.2.3 auf Seite 33 beschrieben, verursacht dieses Modul eine konstante Verzögerung von sechs Takten. Dies entspricht bei einem Takt

von 100MHz einer konstanten Verzögerung von 60ns. Dieses Modul erzeugt keinen Jitter.

Daraus ergibt sich eine konstante Verzögerung der Synchronisationspakete von 60ns.

7.3 Switch-Modul

Das Switch-Modul dient dazu, den Typ der Ethernetpakete zu erkennen und diese dann in die jeweiligen Puffer oder Module weiterzuleiten. Wie schon in Abschnitt 5.2.4 auf Seite 40 beschrieben, hat dieses Modul ein Schieberegister, das permanent taktet. Das Ethernetpaket wird durch das Schieberegister verzögert und erkennt dabei den Typ des Ethernetpakets. Da es permanent taktet und es keinen Zustand gibt, bei dem das Schieberegister anhält, gibt es keinen Jitter. Das Schieberegister verzögert das Ethernetpaket um 6 Takte. Dies entspricht bei einem Takt von 100MHz einer konstanten Verzögerung von 60ns. Weitere Einflüsse auf den Jitter bezogen auf die globale Uhr hat dieses Modul nicht.

7.4 Sync-Modul

In dieser Arbeit wurden zwei Implementierungen des Sync-Moduls gezeigt. Diese haben einen unterschiedlichen Jitter bezogen auf die globale Uhr. Zunächst wird der Jitter der Light Implementierung gezeigt.

7.4.1 Light Implementierung

Bei der Light Implementierung wird kein Zeitfenster definiert, in dem auf Synchronisationsframes reagiert werden soll. Es wird immer auf alle Synchronisationsframes reagiert. Die Offset-Korrektur wird direkt nach dem diese berechnet wurde vorgenommen.

Im Abschnitt 5.2.5.2 auf Seite 49 wird beschrieben, dass zwischen zwischen dem Eintreffen des PCF und der Offset-Korrektur 64 Takte vergehen. Bei einem 100MHz Takt ergibt dies eine Verzögerung von 640ns.

7.4.2 AS6802 Implementierung

Bei der AS6802 Implementierung (siehe Abschnitt 5.2.5.1 auf Seite 43) wird zu einem festgelegten Zeitpunkt $smc\_clock\_corr\_pit$ die Offset-Korrektur vorgenommen. Diese Implementierung schafft es, ohne Jitter exakt zu diesem Zeitpunkt die Korrektur vorzunehmen. Dieser Zeitpunkt wird mit der Formel 2.3 auf Seite 8 berechnet und ist abhängig von der Ungenauigkeit der Uhren im Netzwerk. Nimmt man eine Zykluszeit von 5ms an und eine

Ungenauigkeit von einem Prozent, so ergibt sich, dass dieser Zeitpunkt 50µs von dem erwarteten Eintreffzeitpunkt $smc\_scheduled\_pit$ des PCF entfernt ist. Da in dieser Implementierung diese Zeit gewartet werden muss, spielt die Verzögerung der vorangegangen Module kein Rolle, da die Verzögerungen in Summe kleiner als diese Wartezeit ist.

Somit ergibt sich für dieses Modul eine konstante Verzögerung von 50µs, bei einer Zykluszeit von 5ms und einer Geschwindigkeitsabweichung von einem Prozent.

7.5 Fixed-Point-Timer Modul

Das Fixed-Point-Timer Modul wird vom Sync-Modul konfiguriert. Das Sync-Modul berechnet anhand der Offset-Korrektur den neuen Wert, der je Takt auf den Fixed-Point-Timer addiert wird. Zur Berechnung des Jitters, der durch den Fixed-Point-Timer entsteht, muss die Auflösung des Timers und die Schwankung von Quarzen betrachtet werden.

In dem Artikel *Fundamentals of Quartz Oscillators* (vgl. Hewlett Packard) wird die Funktionsweise und das Verhalten von Quarzen beschrieben. Die Schwankung von Quarzen ist von vielen Bedingungen abhängig. Die Bedingungen, die die Schwankung am meisten beeinträchtigen, sind die Schwankung der Umgebungsthemperatur, Schwankung der Spannungsversorgung und das Alter des Quarzes. Bei Temperaturen von $-50°$C bis $+50°$C schwankt ein durchschnittlicher Quarz um 100ppm. Das entspricht einer Schwankung von 0,01%. Diese Schwankung wird für die zukünftigen Berechnungen angenommen.

Angenommen die lokale Uhr und die globale Uhr haben zu Beginn eines Time-Triggered Ethernetzykluses die gleiche Uhrzeit und die gleiche Frequenz und die Time-Triggered Ethernet Zykluszeit beträgt 5ms. Dann kann die Abweichung am Ende des Zykluses maximal 500ns (0,01% von 5ms) betragen. Dieser Wert entspricht dem Jitter bei einer Zykluszeit von 5ms und wird für zukünftige Berechnungen angenommen. Bei einer Temperaturschwankung von $5°$C um den Temperaturbereich von $25°$C schwankt ein Quarz nur noch um 25ppm. Dies entspricht einem Jitter von 125ns. Jedoch sind Temperaturen von $-50°$C bis $+50°$C in einem Automobil realistischer. Aus dem Grund wird hier von einem Jitter von 500ns ausgegangen.

Wie im Abschnitt 5.2.6 auf Seite 51 beschrieben, hat der Timer die Einheit 20ns und 31 Bit nach dem Komma. Daraus ergibt sich eine Auflösung von $2^{-31} * 20ns = 9,31 * 10^{-9}ns$. Diese Auflösung ist feiner, als die durch die Schwankung der Quarze ergebene Ungenauigkeit. Mit diesem Timer kann also die physikalisch maximale Genauigkeit eingestellt werden.

Daraus ergibt sich für den Fixed-Point-Timer ein direkter Jitter von 500ns, wenn die Geschwindigkeitskorrektur vorgenommen wurde.

7.6 Guard-Modul und MAC-Modul (Sendeseite)

Das Guard-Modul triggert den Sendevorgang der Ethernetpakete. Für die Time-Triggered Nachrichten wird für jede CT-ID ein Sendezeitpunkt angegeben. Das Guard-Modul sorgt dann dafür, dass genau zu diesem Zeitpunkt die Daten aus dem richten Sendepuffer in das MAC-Modul übertragen werden. Dies geschieht ohne Jitter. Durch einen Mechanismus, der in Abschnitt 5.2.7 auf Seite 52 beschrieben ist, sorgt das Guard-Modul dafür, dass jedes Ethernetpaket nach der Übertragung zum MAC-Modul direkt gesendet werden kann, ohne dass im FIFO-Speicher noch andere Ethernetpakete sein können. Dies geschieht, indem die Bandbreite zwischen Guard-Modul und MAC-Modul künstlich der Ethernetbandbreite angepasst wird.

Der Jitter beim Sendevorgang wurde mit dem Messaufbau bestimmt, der in Abbildung 7.2 skizziert ist. Es wurde ein System aufgebaut, dass eine Zykluszeit von 500µs hat. In diesem Zyklus wurde eine Time-Triggered Nachricht geplant. Auf der Softwareseite wurde ein Programm geschrieben, dass dafür sorgt, dass immer eine Nachricht zum Senden bereit steht. Am Ethernetausgang wurde ein Oszilloskop angeschlossen, das Ethernetsignale decodieren und erkennen kann. Zusätzlich wurde das SOF-Signal (start of frame) der Local-Link Schnittstelle an einen FPGA-Pin gelegt und am selben Oszilloskop gemessen. Da das Guard-Modul nicht jittert, wurde hier davon ausgegangen, dass das SOF-Signal auch nicht jittert, da es vom Guard-Modul gesteuert wird. Bei dieser Messung erhält man nicht nur den Jitter, sondern auch die Latenz die das MAC-Modul erzeugt.

Die gemessene Latenz ist abhängig von der Paketgröße. Bei einem 60Byte Paket wurde eine mittlere Latenz von 500ns gemessen. Bei einem 1514Byte großem Paket wurde eine mittlere Latenz von 4140 gemessen. Die Latenz jittert bei beiden Paketgrößen um 40ns.

Die Abweichungen in der Latenz ergeben sich durch die unterschiedliche Übertragungszeit der verschieden großen Pakete vom Guard-Modul zum MAC-sModul. Die LocalLink Schnittstelle überträgt pro Takt 4Byte. Der Takt beträgt 100MHz. Dies ergibt eine Bandbreite von 3,2GBit/s. Für 60Byte werden 15 Takte benötigt, also 150ns. Für 1514Byte werden 379 Takte benötigt, also 3790ns. Die interne Latenz im MAC-Modul beträgt beim 60Byte Paket 500ns (gemessen) - 150ns (gerechnet) = 350ns. Beim 1514Byte Paket 4140ns (gemessen) - 3790ns (gerechnet) = 350ns. Die interne Latenz des MAC-Modul ist also konstant. Die Schwankung in der Latenz, bezogen auf die unterschiedlichen Paketgrößen, muss nicht in

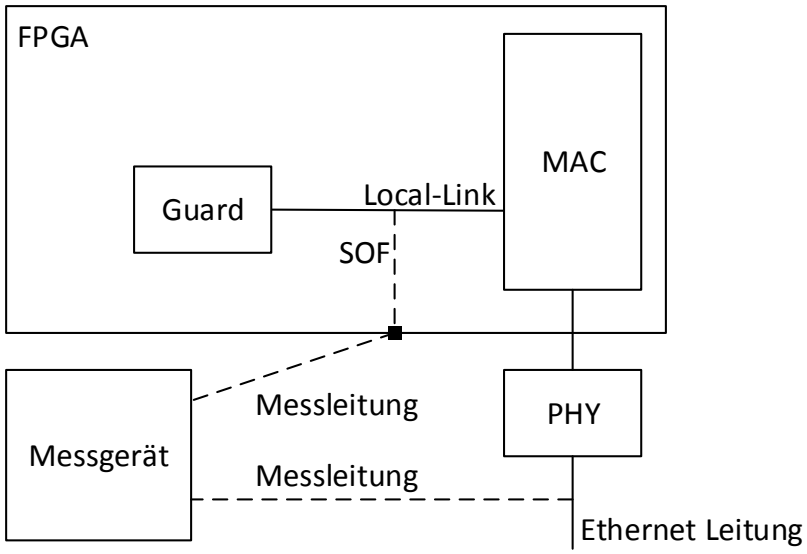

Abbildung 7.2: Messaufbau zur Messung des Jitters beim Senden
Gestrichelt dargestellte Linien sind Leitungen, die zur Messung verlegt wurden.

den Jitter eingerechnet werden, da die Paketgröße von Time-Triggered Nachrichten bekannt ist und offline konfiguriert wird. Bei der Berechnung der Konfiguration der Absendezeitpunkte im Guard-Modul kann dies mit berücksichtigt werden. Als Ergebnis hat das Guard-Modul einen Jitter bezogen auf die lokale Uhr von 40ns.

7.7 Gesamtjitter bezogen auf die globale Uhr

In diesem Abschnitt werden die Jitter der einzelnen Module zu einem Jitter bezogen auf die globale Uhr zusammengerechnet.

Der Jitter bezogen auf die globale Uhr ist davon abhängig, welche Implementierung für das Sync-Modul verwendet wird. Bei der AS6802 Implementierung spielt die Verzögerung auf der Empfangsseite beim MAC-Modul, Timestamping-Modul und Switch-Modul keine Rolle, da das Sync-Modul sowieso bis zu einem konfigurierten Zeitpunkt wartet, in dem die Offset-Korrektur vorgenommen wird. Die Verwendung eines Fixed-Point-Timers reduziert die Jitter, die durch die Verzögerung und deren Jitter beim Empfang von Synchronisationsnachrichten entstehen auf einen geringen Wert. Um die Unterschiede in der Sync-Modul Implementierung und der Verwendung des Fixed-Point-Timer-Moduls zu verdeutlichen, werden nun an

den folgenden Beispielen der Jitter berechnet. Bei der folgenden Berechnung werden folgende Werte angenommen.

- Ethernetbandbreite 100MBit/s
- Time-Triggered Ethernet Zyklus von 5ms
- Abweichung der Quarze um ein Prozent

7.7.1 Jitter mit AS6802 Sync ohne Fixed-Point-Timer

Wenn der Fixed-Point-Timer nicht verwendet wird, weicht die lokale Uhr nach einem Zyklus von 5ms von der globalen Uhr um 50µs ($jitter\_lokal\_zu\_global$) ab. Dieser Wert entspräche dem Jitter, wenn die Offset-Korrektur ohne Verzögerung stattfinden würde.

Bei der AS6802 Implementierung beträgt die Verzögerung zwischen Eintreffen des Synchronisationspakets und der Offset-Korrektur 50µs. Um den Jitter, der durch diese Verzögerung entsteht zu ermitteln, muss die Abweichung der Uhren berechnet werden, die innerhalb dieser Zeit einstehen kann. Da die Uhren um ein Prozent abweichen, beträgt hier der Jitter 500ns ($jitter\_verzoegerung$) (siehe Gleichung 7.1. Abbildung 6.2 und 6.1 auf Seite 61 verdeutlichen diesen Effekt

Zusätzlich zu diesem Jitter kommt noch die Ungenauigkeit der Zeitstempeleinheit von 20ns ($jitter\_timestamping$) hinzu, der Jitter beim Senden von Time-Triggered Nachrichten von 40ns ($jitter\_sending$) und die Schwankung des Quarzes von 500ns ($jitter\_quarz$).

In diesem Beispiel beträgt der Jitter bezogen auf die globale Uhr:

$$jitter\_lokal\_zu\_global + jitter\_verzoegerung + jitter\_timestamping + jitter\_sending + jitter\_quarz = jitter\_gesamt$$

$$50s + 500ns + 60ns + 40ns + 500ns = 51,1\mu s.$$

7.7.2 Jitter mit Light Sync ohne Fixed-Point-Timer

In diesem Beispiel weichen die Uhren ebenfalls um 50µs ($jitter\_lokal\_zu\_global$) nach einem Zyklus voneinander ab. Dieser Wert entspräche dem Jitter, wenn die Offset-Korrektur ohne Verzögerung stattfinden würde.

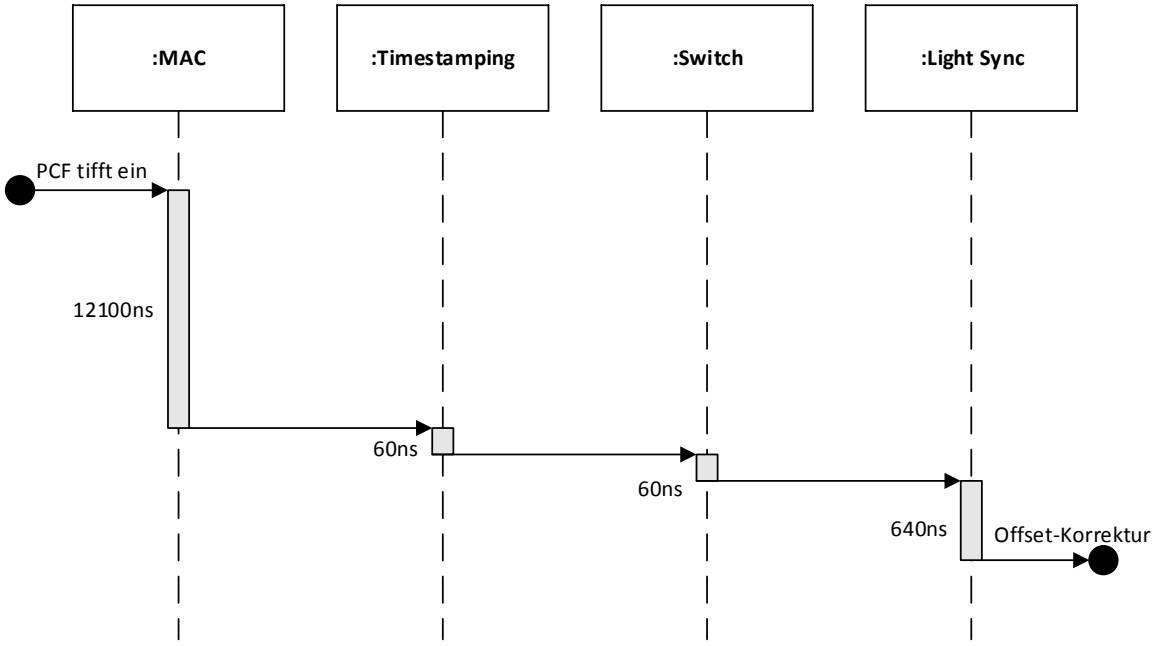

Abbildung 7.3: Sequenzdiagramm, dass die Verzögerung von PCFs der einzelnen Module darstellt

Bei der Sync-Modul Light-Implementierung wird die Offset-Korrektur, direkt nach dem das Sync-Modul das Synchronisationspaket erhalten hat, eingeleitet. Hier muss also die Verzögerung der vorherigen Module mit berücksichtigt werden. Diese wird in Abbildung 7.3 als Sequenzdiagramm dargestellt.

- MAC-Modul: Maximale Verzögerung 12100ns (v_mac)
- Timestamping-Modul: Konstante Verzögerung 60ns ($v\_timestamping$)
- Switch-Modul: Konstante Verzögerung 60ns (v_switch)
- Light-Sync-Modul: Konstante 640ns (v_sync)
- Gesamte Verzögerung: $v\_mac + v\_timestamping + 60nsv\_switch + v\_sync = 12860ns$

Die gesamte Verzögerung beträgt maximal 12860ns. Die lokale Uhr weicht in dieser Zeit um ein Prozent von der globalen Uhr ab. Daraus ergibt sich ein Jitter in Bezug zur globalen Uhr von 128ns ($jitter\_verzoegerung$) (siehe Formel 7.1).

Zusätzlich zu diesem Jitter kommt noch die Ungenauigkeit der Zeitstempeleinheit von 20ns ($jitter\_timestamping$) hinzu, der Jitter beim Senden von Time-Triggered Nachrichten von 40ns ($jitter\_sending$) und die Schwankung des Quarzes von 500ns ($jitter\_quarz$).

In diesem Beispiel beträgt der Jitter bezogen auf die globale Uhr

$$jitter\_lokal\_zu\_global + jitter\_verzoegerung + jitter\_timestamping + jitter\_sending + jitter\_quarz = jitter\_gesamt$$

$$50\mu s + 128ns + 20ns + 40ns + 500ns = 50,688\mu s.$$

7.7.3 Jitter mit Fixed-Point-Timer

Wie schon in Abschnitt 7.5 beschrieben, ist es mit dem Fixed-Point-Timer Modul möglich, die Geschwindigkeit der lokalen Uhr genau einzustellen. Dadurch wird die Offset-Korrektur stark reduziert. Wenn die Quarze vom Synchronisations-Master und Synchronisations-Client in der Geschwindigkeit um ein Prozent abweichen, so kann dies zum größten Teil durch den Fixed-Point-Timer ausgeglichen werden. Die Verzögerung der Synchronisationsnachrichten spielt nur noch vernachlässigbar in den Jitter ein, da die Uhr über den gesamten Zyklus hinweg nahezu die Geschwindigkeit der globalen Uhr hat. Im optimalen Fall besteht der Jitter dann nur noch aus der Schwankung des Quarzes, der Genauigkeit der Eingangszeitstempel und des Jitters beim Senden von Time-Triggered Nachrichten. Mit den genannten angenommen Werten ergibt dies einen Jitter von 500ns ($jitter\_quarz$) + 60ns ($jitter\_timestamping$) + 40ns ($jitter\_sending$) = 600ns.

Abbildung 7.4 zeigt einen Aufbau, mit dem ein Jitter wie folgt ermittelt wurde. Ein Synchronisations-Master sendet in einem Zyklus von 5ms Synchronisationsnachrichten. Das Switch-Modul wurde so konfiguriert, dass Synchronisationsnachrichten an das Sync-Modul und an einen Eingangspuffer dupliziert weitergeleitet werden, so dass die Software auch die Synchronisationsnachrichten erhält. Als Sync-Modul wurde die Light Implementierung verwendet. Der Fixed-Point-Timer war eingeschaltet.

Auf der CPU läuft eine Anwendung, die die Synchronisationsnachrichten aus dem Empfangspuffer ausliest und die Differenz zwischen dem Eingangszeitstempel und dem Feld *pcf_transparent_clock* berechnet. Wie schon in Abschnitt 2.1 beschrieben, enthält dieses Feld die Uhrzeit, die der Synchronisations-Client zum Zeitpunkt des Empfangs haben sollte. Die Differenz dieser Werte ergibt die Offset-Korrektur. Der Wert der Offset-Korrektur wurde aufgezeichnet. Das Maximum lag bei 500ns. Um nun den gesamten Jitter zu erhalten, muss zu diesem Wert noch der Jitter, der beim Senden von Time-Triggered Nachrichten entsteht,

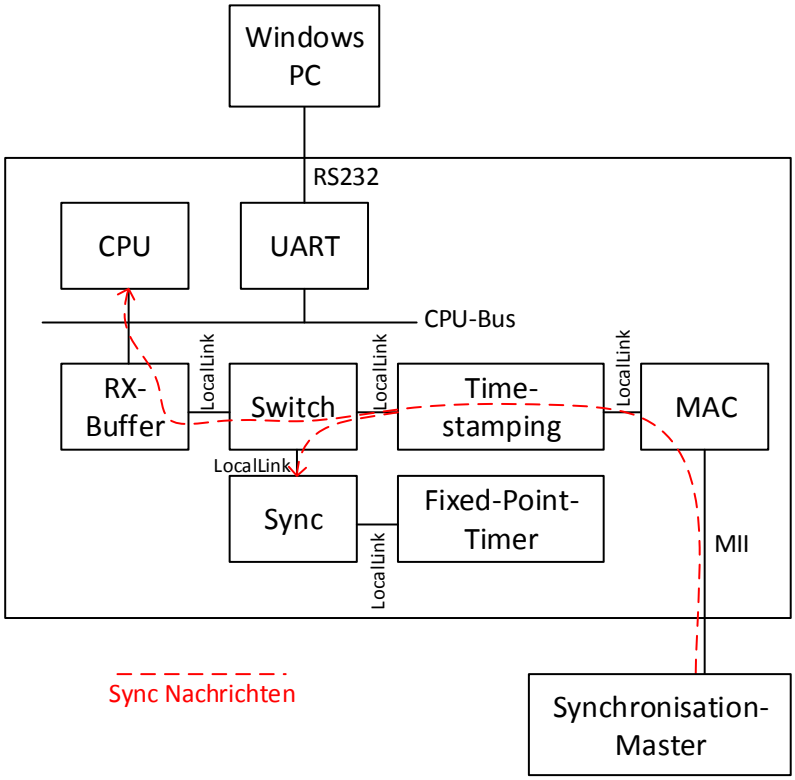

Abbildung 7.4: Messaufbau zur Messung des gesamten Jitters bezogen zur globalen Uhr

addiert werden und man erhält einen gesamten Jitter bezogen auf die globale Uhr von 540ns. Dieser Wert konnte durch folgende optimale Bedingungen erreicht werden.

- Nahezu konstante Raumtemperatur
- Netzwerkteilnehmer waren nur ein SM und ein SC (an dem gemessen wurde) ohne CM. D. h. es wurde keine Durchschnittszeit als globale Uhr verwendet.
- Der Sync-Master hatte einen Jitter von nur 300ns.

300ns Jitter von den gemessenen 500ns stammen also vom Synchronisations-Master.

7.8 Zusammenfassung

In diesem Abschnitt wurden Auswirkungen der Module auf den Jitter bezogen auf die globale Uhr beschrieben. Dabei hat sich deutlich gezeigt, dass das Fixed-Point-Timer-Modul am

stärksten dazu beiträgt, den Jitter zu reduzieren. Dies gilt jedoch nur, wenn in den Netzwerkteilnehmern Quarze verwendet werden, die stärker von einander abweichen. Davon kann man aber ausgehen, da gut kalibrierte Quarze aus Kostengründen oft nur in Messgeräten verwendet werden. In den für die Beispielrechnungen verwendeten Teilnehmern wurde eine Abweichung von einem Prozent gemessen, dass in dieser Arbeit als hohe Abweichung bewertet wird.

Bei einer vollen Hardwareimplementierung konnte unter optimalen Bedingungen ein Jitter von 540ns gemessen werden. Dies wurde bei einem Zyklus von 5ms und einem Synchronisationsmaster der um 300ns pro Zyklus schwankt, gemessen.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung der Ziele und Ergebnisse

In dieser Arbeit wurde ein skalierbares Hardware- Software Co-Design vorgestellt, dass Timer-Triggered Ethernet Kommunikation ermöglicht. Es konnte gezeigt werden, dass die CPU durch einen Co-Prozessor stark entlastet wird und dass ein Co-Prozessor den Jitter bezogen auf die globale Uhr deutlich reduzieren kann. Wenn alle Module in Hardware partitioniert sind, ist es sogar möglich, die Protokollausführung nahezu transparent neben einer Softwareanwendung zu betreiben. Dies reduziert den Migrationsaufwand der Software und ermöglicht somit einen leichteren Umstieg auf ein Time-Triggered Ethernet Protokoll.

Das entwickelte Timestamping-Modul ermöglicht es, auf 20ns genau den Ankunftszeitpunkt von Ethernetpaketen aufzuzeichnen. Dadurch ist eine höhere Synchronisationsgenauigkeit möglich, was zu einem niedrigeren Jitter führt.
Das entwickelte Switch-Modul klassifiziert Ethernetframes und leitet diese an die dazugehörigen Empfangspuffer weiter. Durch dieses Modul wird die CPU am stärksten entlastet, da die CPU nun Interrupts ausschalten kann und nicht bei jedem empfangenen Paket die Ausführung der Anwendung unterbrechen muss. Das Sync-Modul ermöglicht es Steuergeräten, die nicht ständig benötigt werden, über einen längeren Zeitraum im Sleep-Modus zu verweilen und somit Energie zu sparen. Das entwickelte Guard-Modul ermöglicht der Anwendung generierte Time-Triggered Nachrichten asynchron in einen Sendepuffer abzulegen, ohne zum Absendezeitpunkt nochmal aktiv werden zu müssen. Außerdem reduziert das Guard-Modul, durch das hardwaregesteuerte Senden den Jitter und verhindert Kollisionen mit Best-Effort Nachrichten.

8.2 Ausblick auf zukünftige Arbeiten

In dieser Arbeit wurde zur Speicherung von Ethernetpaketen das Modul *LogiCORE IP XPS LL FIFO* (vgl. Xilinx Inc., 2011) verwendet. Dies ist ein FIFO-Speicher der in Hardware synthetisiert wird. Dies ist kostenintensiv, ermöglicht aber eine schnelle Entwicklung von Modulen. Konzeptionell werden sich die hier entwickelten Module von der Art der Speicherung der Ethernetpakete nicht stark unterscheiden. Da in dieser Arbeit nicht der optimale Weg zur Speicherung von Ethernetpaketen gefunden werden sollte, wurde hier dieser Weg gewählt. Um Hardwarekosten zu reduzieren, sollte in zukünftigen Arbeiten ein Memory-Controller entwickelt werden, der die empfangenen Ethernetpakete vom Switch-Modul in einen externen RAM überträgt und die zu sendenden Pakete vom RAM zum Guard-Modul überträgt.

Das Time-Triggered Ethernet Protokoll AS6802 spezifiziert drei Nachrichtenklassen, die zur Kommunikation verwendet werden können. In dieser Arbeit wurden bei der Konzeption und Implementierung nur die Nachrichtenklassen Time-Triggered und Best-Effort berücksichtigt, da hier die meisten Schnittmengen mit anderen Time-Triggered Ethernet Protokollen bestehen. Die Nachrichtenklasse Rate-Constrained wurde in dieser Arbeit nicht beachtet. Soll in zukünftigen Arbeiten die volle AS6802 Funktionalität verfügbar sein, so muss bei der Konzeption dies berücksichtigt werden. Das Guard-Modul hat einen Mechanismus, der für diese Nachrichtenklasse wiederverwertet werden kann. In diesem Modul wird die Bandbreite zum MAC-Modul künstlich auf die Ethernetbandbreite reduziert. Dieser Drosselmechanismus kann in veränderter Form bei der Erweiterung des Konzepts behilflich sein.

Literaturverzeichnis

[Aeronautical Radio Incorporated 2002] AERONAUTICAL RADIO INCORPORATED: Aircraft Data Network / ARINC. Annapolis, Maryland, 2002 (664). – Standard. – URL http://www.aviation-ia.com/cf/store/catalog_detail.cfm?item_id=668. – Zugriffsdatum: 2014-12

[Alderisi u. a. 2012] ALDERISI, Giuliana ; CALTABIANO, Alfio ; VASTA, Giancarlo ; IANNIZZOTTO, Giancarlo ; STEINBACH, Till ; BELLO, Lucia L.: Simulative Assessments of IEEE 802.1 Ethernet AVB and Time-Triggered Ethernet for Advanced Driver Assistance Systems and In-Car Infotainment. In: *2012 IEEE Vehicular Networking Conference (VNC),*. Piscataway, NJ, USA : IEEE Press, November 2012, S. 187–194. – URL http://inet.cpt.haw-hamburg.de/papers/acvis-saeat-12.pdf. – Zugriffsdatum: 2014-12. – ISBN 978-1-4673-4996-3

[Altera 2013] ALTERA: *Triple-Speed Ethernet MegaCore Function - User Guide*. Altera. Dezember 2013. – URL http://www.altera.com/literature/ug/ug_ethernet.pdf. – Zugriffsdatum: 2014-06

[AUTOSAR Development Cooperation] AUTOSAR DEVELOPMENT COOPERATION: *AUTomotive Open System ARchitecture*. – URL http://www.autosar.org

[Bartols 2010] BARTOLS, Florian: *Leistungsmessung von Time-Triggered Ethernet Komponenten unter harten Echtzeitbedingungen mithilfe modifizierter Linux-Treiber*. Hamburg, HAW Hamburg, Bachelorthesis, Juli 2010. – URL http://core.informatik.haw-hamburg.de/images/publications/theses/bachelorarbeit_florian_bartols.pdf. – Zugriffsdatum: 2014-12. – Bachelorthesis

[Bruckmeier 2010] BRUCKMEIER, Robert: *Ethernet for Automotive Applications.* Vortrag. Juni 2010. – URL http://www.freescale.com/files/ftf_2010/Americas/WBNR_FTF10_AUT_F0558.pdf. – Zugriffsdatum: 2010-12-10

[CoRE-Arbeitsgruppe] CoRE-Arbeitsgruppe: *Communication over Real-time Ethernet*. – URL http://core.informatik.haw-hamburg.de. – Zugriffsdatum: 2014-03-24

[Elmenreich und Ipp 2003] Elmenreich, Wilfried ; Ipp, Richard: Introduction to TTP/C and TTP/A. In: *Proceedings of the Workshop on Time-Triggered and Real-Time Communication Systems*, URL https://mobile.aau.at/~welmenre/papers/2003/rr-55-2003.pdf. – Zugriffsdatum: 2014-12, Dezember 2003, S. 1–9. – eingeladen; Vortrag: Workshop on Time-Triggered and Real-Time Communication Systems, Manno, Switzerland; 2003-12-02

[FlexRay Consortium 2005] FlexRay Consortium: FlexRay Communications System Protocol Specification / FlexRay Consortium. Stuttgart, Dezember 2005 (2.1 Revision A). – Specification

[Gunzinger u. a. 2010] Gunzinger, David ; Kuenzle, Cyrill ; Schwarz, Andreas ; Doran, Hans D. ; Weber, Karl: Optimising PROFINET IRT for fast cycle times: A proof of concept. In: *Factory Communication Systems (WFCS), 2010 8th IEEE International Workshop on* IEEE (Veranst.), URL ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5548637. – Zugriffsdatum: 2014-12, 2010, S. 35–42

[Hewlett Packard] Hewlett Packard: *Fundamentals of Quartz Oscillators*. Electronic Counters Series. – URL http://cp.literature.agilent.com/litweb/pdf/5965-7662E.pdf. – Zugriffsdatum: 1997

[Honeywell International] Honeywell International: . – URL http://www.honeywell.com

[IBM Systems and Technology Group 2012] IBM Systems and Technology Group: *Processor Local Bus - Architecture Specification*. IBM Systems and Technology Group. November 2012. – URL https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/B507AB32B6ACC15B852577F50064848E/$file/PLB6Spec_2012NOV15_v101_NotConNotP6Plus_ext_pub.pdf. – Zugriffsdatum: 2014-06

[IEEE Registration Authority 2014] IEEE Registration Authority: *Ethertype Field Public Listing*. IEEE Registration Authority. Dezember 2014. – URL http://standards.ieee.org/develop/regauth/ethertype/eth.txt. – Zugriffsdatum: 2014-12

[Institute of Electrical and Electronics Engineers 2002] Institute of Electrical and Electronics Engineers: IEEE Standard for a Precision Clock Synchronization Protocol

for Networked Measurement and Control Systems / IEEE. 2002 (IEEE Std. 1588). – Standard. – ISBN 0-7381-3369-8

[Institute of Electrical and Electronics Engineers 2013] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS: 802.1Qbv - Bridges and Bridged Networks - Amendment: Enhancements for Scheduled Traffic / IEEE. URL http://www.ieee802.org/1/pages/802.1bv.html. – Zugriffsdatum: 2014-12, Dezember 2013 (P802.1Qbv/D1.0). – Draft Standard

[Kopetz u. a. 2005] KOPETZ, Hermann ; ADEMAJ, Astrit ; GRILLINGER, Petr ; STEINHAMMER, Klaus: The time-triggered Ethernet (TTE) design. In: *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005.*, URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1420949. – Zugriffsdatum: 2014-12, Mai 2005, S. 22–33

[Lipfert 2008] LIPFERT, Jan: *Technical Data Reference Guide - netX500/100*. Hilscher GmbH. Dezember 2008. – URL http://www.hilscher.com/fileadmin/cms_upload/de/Resources/pdf/netX500-100_Technical_Data_Reference_Guide_13.pdf. – Zugriffsdatum: 2014-12

[Marscholik und Subke 2007] MARSCHOLIK, Christoph ; SUBKE, Peter: *Datenkommunikation im Automobil: Grundlagen, Bussysteme, Protokolle und Anwendungen*. Heidelberg : Hüthig, 2007 (Hüthig Praxis). – ISBN 3-7785-2969-2

[Marvell Semiconductor, Inc 2013] MARVELL SEMICONDUCTOR, INC: *88E1111 Product Brief - Integrated 10/100/1000 Ultra Gigabit Ethernet Transceiver*. Marvell Semiconductor, Inc. Oktober 2013. – URL http://www.marvell.com/transceivers/assets/Marvell-Alaska-Ultra-88E1111-GbE.pdf. – Zugriffsdatum: 2014-06

[Miller 2009] MILLER, Lothar: *Division in VHDL*. Online Artikel. 2009. – URL http://www.lothar-miller.de/s9y/archives/29-Division-in-VHDL.html. – Zugriffsdatum: 2014

[Müller u. a. 2011] MÜLLER, Kai ; STEINBACH, Till ; KORF, Franz ; SCHMIDT, Thomas C.: A Real-time Ethernet Prototype Platform for Automotive Applications. In: *2011 IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. Piscataway, New Jersey : IEEE Press, September 2011, S. 221–225. – URL http://core.informatik.haw-hamburg.de/images/publications/papers/msks-reppa-11a.pdf. – Zugriffsdatum: 2014-12. – ISBN 978-1-4577-0233-4

[Navet u. a. 2005] NAVET, Nicolas ; SONG, Yeqiong ; SIMONOT-LION, Françoise ; WILWERT, Cédric: Trends in Automotive Communication Systems. In: *Proceedings of the IEEE* 93 (2005), Juni, Nr. 6, S. 1204–1223. – URL http://ieeexplore.ieee.org/iel5/5/30937/01435746.pdf. – Zugriffsdatum: 2014-12. – ISSN 0018-9219

[PROFIBUS & PROFINET International] PROFIBUS & PROFINET INTERNATIONAL: *Profinet*. – URL http://www.profibus.com/technology/profinet. – Zugriffsdatum: 2011-02-07

[Rushton 2011] RUSHTON, A.: *VHDL for Logic Synthesis*. Wiley, 2011. – URL https://books.google.de/books?id=lC8HKr0e2nwC. – ISBN 9780470977972

[Saad 2003] SAAD, Alexandre: Das Automobil als Anwendungsgebiet der Informatik - ein Auto ohne Informatik, geht das? In: HUBWIESER, Peter (Hrsg.): *INFOS* Bd. 32, GI, 2003, S. 37–40. – URL http://dblp.uni-trier.de/db/conf/schule/schule2003.html#Saad03. – ISBN 3-88579-361-X

[SAE AS-2D Committee 2011] SAE AS-2D COMMITTEE: *Time-Triggered Ethernet AS6802*. SAE Aerospace. November 2011. – URL http://standards.sae.org/as6802/

[Schäuffele und Zurawka 2013] SCHÄUFFELE, Jörg ; ZURAWKA, Thomas: *Automotive Software Engineering*. Wiesbaden : Vieweg und Teubner, 2013. – ISBN 978-3-8348-2469-1

[Steinbach u. a. 2012] STEINBACH, Till ; LIM, Hyung-Taek ; KORF, Franz ; SCHMIDT, Thomas C. ; HERRSCHER, Daniel ; WOLISZ, Adam: Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802). In: *2012 IEEE Vehicular Technology Conference (VTC Fall)*. Piscataway, New Jersey : IEEE Press, September 2012. – ISSN 1090-3038

[Steinhammer 2006] STEINHAMMER, Klaus: *Design of an FPGA-Based Time-Triggered Ethernet System*. Treitlstr. 3/3/182-1, 1040 Vienna, Austria, Technische Universität Wien, Institut für Technische Informatik, Dissertation, 2006. – URL http://www.vmars.tuwien.ac.at/php/pserver/extern/docdetail.php?DID=2124&viewmode=thesis. – Zugriffsdatum: 27.12.2014

[Steinhammer und Ademaj 2007] STEINHAMMER, Klaus ; ADEMAJ, Astrit: Hardware Implementation of the Time-Triggered Ethernet Controller. In: *IFIP Advances in Information and Communication Technology*, 2007, S. 325–338

[Stöger 1997] STÖGER, Georg: *Hardware-Software-Co-Design of a Time-Triggered Protocol*. Wien, Technischen Universit Wien, Dissertation, Juni 1997

[TTTech Computertechnik AG] TTTECH COMPUTERTECHNIK AG: . – URL http://www.tttech.com. – Zugriffsdatum: 2014-03-24

[Weibel 2005] WEIBEL, Hans: High Precision Clock Synchronization according to IEEE 1588 Implementation and Performance Issues. In: *Embedded World Conference 2005*. Design & Elektronik, WEKA-Fachzeitschriften-Verlag, Februar 2005, S. 981–989

[Xilinx Inc. 2005] XILINX INC.: *LocalLink Interface Specification*. Xilinx Inc. Juli 2005. – URL http://www.xilinx.com/aurora/aurora_member/sp006.pdf. – Zugriffsdatum: 2014-06

[Xilinx Inc. 2010] XILINX INC.: *LogiCORE IP XPS LL TEMAC*. Xilinx Inc. Dezember 2010. – URL http://www.xilinx.com/support/documentation/ip_documentation/xps_ll_temac.pdf. – Zugriffsdatum: 2014-06

[Xilinx Inc. 2011] XILINX INC.: *LogiCORE IP XPS LL FIFO*. Xilinx Inc. Dezember 2011. – URL http://www.xilinx.com/support/documentation/ip_documentation/xps_ll_fifo.pdf. – Zugriffsdatum: 2014-12-23

[Xilinx Inc. 2012a] XILINX INC.: *MicroBlaze Processor Reference Guide*. Xilinx Inc. April 2012. – URL http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/mb_ref_guide.pdf. – Zugriffsdatum: 2014-06

[Xilinx Inc. 2012b] XILINX INC.: *ML605 Hardware User Guide*. Xilinx Inc. Oktober 2012. – URL http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf. – Zugriffsdatum: 2014-12-23

Abkürzungsverzeichnis

AFDX	Avionics Full-Duplex Switched Ethernet
ARINC	Aeronautical Radio Incorporated
AS	Aerospace Standard
BE	Best-Effort
BUS	Binary Unit System
CAN	Controller Area Network
CLK	Clock Signal
CM	Compression Master
CoRE	Communication over Real-time Ethernet
CPU	Central Processing Unit
CRC	cyclic redundancy check
CT-ID	critical traffic identifier
DST	destination
EOF	end of frame
EOP	end of payload
FIFO	first in first out
FPGA	Field Programmable Gate Array
GMII	Gigabit Media Independent Interface
IEEE	Institute of Electrical and Electronics Engineers
IN	Integration Frame
IP	Internetprotokoll
ISR	interrupt service routine

LE	logic elements
LUT	look up table
MAC	Media-Access-Control
MEDL	massage description list
MII	Media Independent Interface
NAFTA	North American Free Trade Agreement
NIC	Network Interface Card
OSI-Modell	Open Systems Interconnection Model
PCF	protocol control frame
PDF	Portable Document Format
PHY	Physical Layer
Pkt	Paket
PLB	Processor Local Bus
RAM	Random-Access Memory
RC	Rate-constrained
RDY	rady Signal
REM	reminder Signal
RX	receive (Empfang)
SAE	Society of Automotive Engineers
SC	Synchronisations-Client
SDRAM	Synchronous Dynamic Random Access Memory
SM	Synchronisations-Master
SOF	start of frame - Signal
SOP	start of payload - signal
SRC	source - Signal
TCP	Transmission Control Protocol
TEMAC	Tripple Speed Ethernet Media-Access-Control
TS	Timestamp
TSU	Timestamping-Unit
TTE	Time-Triggered Ethernet
TTEthernet	Time-Triggered Ethernet
TX	transceive

UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
UML	Unified Modeling Language
USB	Universal Serial Bus
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Abbildungsverzeichnis

Inhalt der beigelegten CD

- **Digitale Version dieser Arbeit** im PDF-Format.
  - /MA_Friedrich_Gross.pdf
- **Sourcecode der einzelnen Module:**
  - /module_einzeln/...
- **Simulationsprojekte:** Für jedes Modul ein Projektordner. Diese können mit Xilinx ISE Project Navigator geöffnet werden. Innerhalb dieses Programms kann dann die Simulation mit ISim gestratet werden.
  - /simulation/...
- **Xilinx EDK-Projekt:** Projektordner der mit Xilinx Platform Studio geöffnet werden kann. Beiliegende Installationsanleitung beachteten.
  - /EDK-Projekt/...

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 02.01.2015 Friedrich Groß