Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Sebastian Kuhrt

Ein Time-Triggered Ethernet basiertes Rückfahrkamerasystem – vom Entwurf bis zur Integration in einen Fahrzeugdemonstrator

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sebastian Kuhrt

Ein Time-Triggered Ethernet basiertes Rückfahrkamerasystem – vom Entwurf bis zur Integration in einen Fahrzeugdemonstrator

Bachelorarbeit eingereicht im Rahmen Bachelorprüfung

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

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

Abgegeben am 22. Mai 2013

Sebastian Kuhrt

Thema der Arbeit
Ein Time-Triggered Ethernet basiertes Rückfahrkamerasystem -
vom Entwurf bis zur Integration in einen Fahrzeugdemonstrator

Stichworte
Rückfahrkamera, Ethernet, TTEthernet, Mikrocontroller, Infotainmentsystem, Echtzeit

Kurzzusammenfassung
In modernen Automobilen befinden sich mehr als 60 elektronische Steuergeräte, verbunden über verschiedene Bussysteme. Echtzeit- Ethernet ist ein geeigneter Kandidat, der die Möglichkeit bietet, die heute eingesetzten Bussysteme zu ersetzen.
Ein Fahrzeugdemonstrator zeigt, wie Systeme mit harten Echtzeitanforderungen, zusammen mit Anwendungen, die eine hohe Bandbreite erfordern, über dasselbe physikalische Ethernet Netzwerk kommunizieren.
In dieser Arbeit wird ein Rückfahrkamerasystem für einen Fahrzeugdemonstrator entwickelt. Für die Realisierung wird geeignete Hardware ausgesucht und es werden die dazugehörigen Anwendungen implementiert. Mit den Anwendungen wird das Kamerabild als kritischer Ethernet-Nachrichtenverkehr zwischen Kamera und Infotainmentsystem übertragen.
Sebastian Kuhrt

Title of the paper
A Time-Triggered Ethernet based rear-view camera system – from design to integration into an in-car network prototype

Keywords
Rear-view-camera, Ethernet, TTEthernet, Microcontroller, Infotainmentsystem, Real-time

Abstract
Modern automobiles consist of more than 60 electronic control units interconnected via different bus systems. Real-time Ethernet is a suitable candidate that offers the possibility to replace the state-of-the-art bus systems. An in-car network prototype demonstrates how systems with hard real-time requirements communicate together with bandwidth demanding applications via the same physical Ethernet network.
In this paper a rear-view camera system for the in-car network prototype is developed.
For the realization, suitable hardware is selected and dedicated applications are implemented. In this application, the camera images are transmitted as time-critical Ethernet messages between the camera and the infotainment system.

Inhaltsverzeichnis

1 Einleitung und Motivation

In heutigen Automobilen gibt es eine größere Anzahl an Steuergeräten, Sensoren und Assistenzsystemen mit unterschiedlichen Anforderungen an das Kommunikationssystem. Typische Anforderungen liegen in der Datenübertragungsrate, der möglichen Botschaftslänge, der Anzahl der anschließbaren Knoten im deterministischen Übertragungsverhalten, sowie bei Zuverlässigkeits- oder Sicherheitsanforderungen (vgl. [21]). Die SAE (Society of Automotive Engineers) hat eine grobe Klassifizierung der Bussysteme nach Bandbreite und Anwendungsbereich vorgenommen. In der Tabelle 1.1 sind die verschiedenen Bussysteme und deren Klasseneinteilung zu sehen. Heute muss diese Einteilung jedoch noch um sicherheitsrelevante Merkmale erweitert werden, damit die oben genannten Anforderungen erfüllt werden (vgl. [39]). Dazu zählen Bussysteme wie FlexRay und Time-Triggered CAN (TTCAN), die „hohen" Echtzeitanforderungen genügen. Wie zu sehen ist, wird in heutigen Automobilen eine Vielzahl an unterschiedlichen Bussystemen eingesetzt.

SAE-Klasse	Merkmale	Typisches Bussystem
A	Vernetzung von Aktoren und Sensoren Geringe Datenraten (ca. 10 kBit/s) Geringe Ausprägung von Fehlererkennungs- und -behebungsmechanismen	LIN-Bus
B	Vernetzung von Steuergeräten (z. B. Komfortbereich) Mittlere Datenraten (ca. 125 kBit/s) Komplexe Mechanismen zur Fehlererkennung und -behebung	CAN-Bus („Low Speed")
C	Vernetzung von Steuergeräten mit „einfachen" Echtzeitanwendungen (z. B. Antriebsstrang) Hohe Datenraten (bis zu 1 MBit/s)	CAN-Bus („High Speed")
D	Multimedia-Anwendungen Sehr hohe Datenraten (bis zu 10 MBit/s) und Botschaftslängen	MOST-Bus

Tabelle 1.1: SAE- Klassen für Bussysteme (Quelle: [28])

Das Ethernet (IEEE 802.3) bietet die Möglichkeit, all diese Systeme bzw. Anwendungen über ein Netzwerk kommunizieren zu lassen. Ethernet ist jedoch in seiner Grundform nicht fähig, die genannten unterschiedlichen Anforderungen zu erfüllen, dies ist erst durch eine Erweiterung möglich. Eine Erweiterung des Standards Ethernet (IEEE 802.3) ist das TTEthernet (Time- Triggered Ethenet). TTEthernet ist von der Firma TTTech Computertechnik AG [37] entwickelt worden und wurde im November 2011 durch die SAE standardisiert (SAE AS6802 – Time-Triggered Ethernet [31]). Diese Technologie bietet deterministische, synchrone und staufreie Echtzeitkommunikation über eine fehlertolerante, selbststabilisierende Synchronisationsstrategie (vgl. [31]).
Ziel dieser Arbeit ist die Konzeption und Integration einer Rückfahrkamera in ein bestehendes TTEthernet Netzwerk. Die Rückfahrkamera soll es dem Fahrer erleichtern, den Überblick über das Verkehrsgeschehen beim Rückwärtsfahren zu behalten. Gefahrensituationen von plötzlich auftauchenden Personen oder Objekten, sollen mit Hilfe der Kameraperspektive vom Fahrer besser erkannt werden. Dabei müssen die Kameradaten in Echtzeit, bis zur Anzeige beim Fahrer übertragen werden. Dies ist ein Einsatzgebiet für das TTEthernet. Die Arbeit findet im Rahmen der CoRE [7] (Communication over Real-Time Ethernet) Gruppe der HAW- Hamburg statt. Zur Demonstration eines echtzeitfähigen Ethernets ist über mehre Abschlussarbeiten, ein Demonstrator entwickelt worden, der ein TTEthernet Netzwerk im Automotive Kontext realistisch abbildet. Kern des Netzwerks ist ein Steer-by-Wire[1] Lenksystem für ein Automobil (vgl. [33]).

[1]Steer-by-Wire: Der Lenkbefehl wird elektrisch vom Steuergerät im Lenkrad, zur elektronischen Kontrolleinheit für die Ränder übertragen. Es besteht keine mechanische Verbindung.

2 Grundlagen

In folgenden Abschnitten wird beschrieben, wie das Ethernet erweitert wird, damit ein echtzeitfähiges Kommunikationsmedium entsteht, sodass die heutigen, im Automobil eingesetzten Bussysteme ersetzt werden können. Dazu muss der Begriff Echtzeit- bzw. Echtzeitsysteme geklärt werden. Weiterhin wird ein Überblick über den Demonstrator gegeben, damit der Kontext dieser Arbeit verstanden wird. Zwar bietet das Ethernet eine hohe Bandbreite (Demonstrator- TTEthernet 100 MBit/s), trotzdem darf die Kameraplattform das Rückfahrkamerabild nicht ohne eine Datenreduktion übertragen. Ein unkomprimiertes Kamerabild würde die Kapazität des Netzwerks schnell überlasten. Deswegen wird ein Überblick über die Grundlagen von Kodierverfahren gegeben. Das Kamerabild soll in Echtzeit zur Anzeige auf dem Infotainmentsystem- im Fahrgastraum übertragen werden, zudem wird das Video mit einem gängigen Kodierverfahren kodiert- dies erfordert spezielle Hardware die im letzten Abschnitt kurz vorgestellt wird.

2.1 Echtzeitsysteme

Wird ein Programm auf einem Rechner ausgeführt, so können während der Berechnung viele unvorhersehbare Ereignisse eintreten, ein Festplattenzugriff, die Kommunikation über das Netzwerk dauert länger und die Ausführung der Berechnung selbst verzöger sich, weil im Hintergrund der Virusscanner läuft. Der Anwender muss geduldig auf sein Ergebnis warten, welches dennoch korrekt auf dem Bildschirm ausgegeben wird. In diesem Fall spricht man nicht von einem Echtzeitsystem. Von einem Echtzeitsystem spricht man, wenn zu der logischen Korrektheit noch die zeitliche Korrektheit dazukommt (vgl. [3]). Anhand von Zeitbedingungen wird das Echtzeitsystem in drei Klassen eingeteilt.

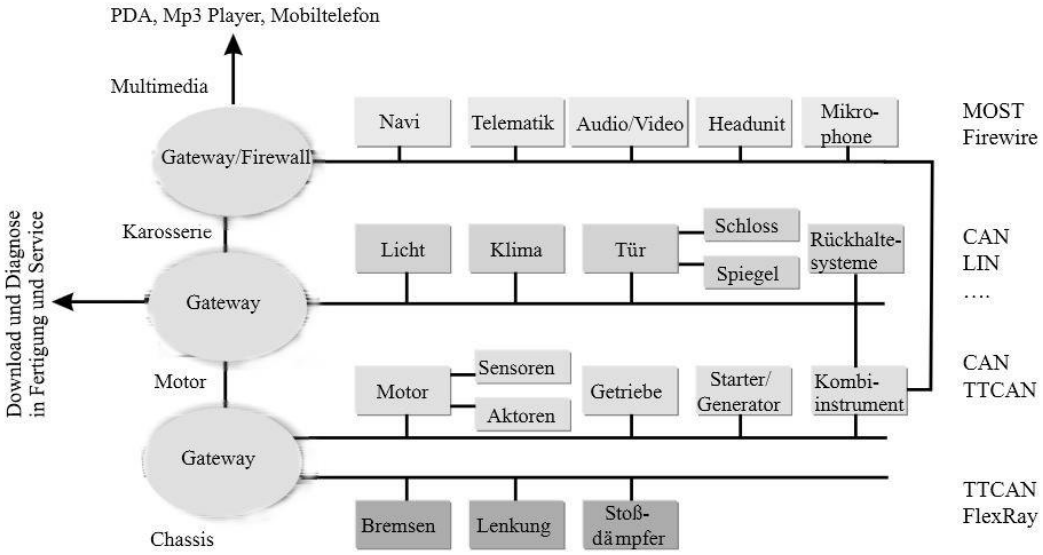
- Wird in einem **harten Echtzeitsystem** eine Zeitschranke (Deadline) verpasst, so hat dies katastrophale Folgen auf das System und kann im schlimmsten Fall zu einer Katastrophe führen. Muss ein System mindestens eine harte Deadline erfüllen, spricht man auch von einem „safety-critical System“. Ein typisches Beispiel ist der Airbagcontroller, hält dieser seine harte Deadline nicht ein, führt dies unweigerlich zu einer Katastrophe.
- Bei einem **festen Echtzeitsystem** ist die Lieferung eines Ergebnisses nach der Deadline unbrauchbar, dies führt jedoch nicht zu Schäden am System oder zu katastrophalen Ereignissen.

- Von einem **weichen Echtzeitsystem** spricht man, wenn auch ein nach der Deadline geliefertes Ergebnis brauchbar ist.

2.2 Echtzeitsysteme im Automobil

In heutigen Automobilen finden sich viele solcher Systeme (20 – 80 vgl. [19]), man spricht bei einem Auto auch von einem großen verteilten Echtzeitsystem, diese elektrischen Steuergeräte müssen miteinander verbunden werden, parallel dazu wird immer mehr Informations- und Unterhaltungselektronik in einem Auto verbaut. Hinzu kommen die verschiedenen Bussysteme, die untereinander Nachrichten austauschen. Der Austausch der Nachrichten findet über die in Abbildung 2.1 dargestellten Gateways[2] statt. Damit ein Echtzeitsystem seine Deadlines einhalten kann, müssen die Nachrichten aus seiner Umgebung (z.B. ein Signal eines anderen Steuergerätes) schließlich auch mit vorhersagbarer Übertragungszeit (Latenz) und mit geringen Schwankungen (Jitter) beim Echtzeitsystem eintreffen. Das heißt also, dass das verteilte Echtzeitsystem stark abhängig von seinem Kommunikationsmedium ist.

Abbildung 2.1: Bussysteme im Automobil u



nd deren Vernetzung (Quelle: [20])

2.3 Echtzeitfähiges Ethernet

Wie bereits erwähnt, ist das Ethernet in seiner Grundform nicht Echtzeitkommunikationsfähig. Werden mehrere Teilnehmer im Netzwerk zusammen geschlossen, ist es nicht garantiert, wann (Latenz) oder ob eine Nachricht beim Empfänger ankommt. Würde ein Teilnehmer im Netzwerk z.B. ein Steuergerät für einen Airbag im Auto sein, welches

[2]Ein Gateway übersetzt und übermittelt Nachrichten von einem Netz in ein anderes. Vor allem findet eine Übersetzung der Kommunikationsprotokolle statt. Ein Gateway ist somit eine Art Protokollkonverter (vgl. www.itwissen.info).

Nachrichten von einem anderen Teilnehmer empfangen soll, wie einem Aufprallsensor, so könnte es sein, dass diese Nachricht viel zu spät oder gar nicht beim Steuergerät des Airbags ankommt. Teilen sich mehrere Systeme- im allgemeinen Knoten genannt, ein Kommunikationsmedium, so muss es Zugriffsverfahren geben, die den Medienzugriff regeln. Ein Zugriffsverfahren ist das kontrollierte- zeitgesteuerte Time Devision Multiple Access- Verfahren (TDMA)- vgl. [17]. Durch das TDMA- Verfahren wird ein deterministischer Medienzugriff realisiert. Jedem Knoten wird ein fester Sendezeitpunkt zugeteilt. Es muss einen festen Ablaufplan für das gesamte Netzwerk geben. Dieser Ablaufplan muss statisch, also bevor das Netzwerk in Betrieb genommen wird feststehen. Damit alle Knoten ihren Sendezeitpunkt einhalten können, müssen die internen Uhren der Knoten synchron laufen (vgl. [27]). Ein Protokoll, welches dieses Verfahren nutzt, ist das TTP (Time-Triggered-Protocol) oder das bereits erwähnte FlexRay Protokoll. Netzwerkweit gibt es eine statisch definierte Zugriffstabelle, die bei jedem Netzwerknoten hinterlegt wird (vgl. [17]). Das TTP wurde gemeinsam von der Technischen Universität Wien und der TTTech Computertechnik AG entwickelt und wird z.B. in Eisenbahnsystemen und Spezialfahrzeugen eingesetzt (vgl. [39]. Das TTEthernet (Netzwerk des Demonstrators) ist eine Weiterentwicklung des TTP.

2.4 TTEthernet

Das TTEthernet (SAE AS6802) ist hart Echtzeitfähig- und erfüllt die in der Einleitung genannten Anforderungen an Kommunikationssystem im Automobil. Der Zusammenschluss mehrere Knoten wird durch spezielle Switches realisiert (Switched- Ethernet). Diese müssen das TTEthernet- Protokoll implementieren, damit sie die Nachrichten der Knoten zum richtigen Zeitpunkt an andere Knoten weiterleiten können und ein deterministischer Medienzugriff realisiert wird.

Für das TDMA Verfahren besitzt jeder Knoten eine sog. Schedulingtabelle. In dieser wird ein periodischer Zyklus definiert. Dieser legt fest, wann eine Nachricht gesendet oder empfangen werden darf. Diese Konfiguration findet statisch- vor Inbetriebnahme des Netzwerkes statt. In Abbildung 2.2 ist ein beispielhaftes TTEthernet zu sehen. Es gibt Sende- und Empfangsknoten, diese besitzen periodische Zyklen (2ms – 6ms) in denen sie Nachrichten (TT, RC- BE) senden (Sender 1 und 2) oder Nachrichten empfangen (Receiver). Eine genauere Beschreibung der Nachrichten erfolgt im nächsten Abschnitt. Durch dieses Protokoll, aufbauend auf dem Switched Ethernet und der statischen Konfiguration des Netzwerkes ist die Übertragungsverzögerung der Nachrichten gering- dies führt zu einer geringen Latenz und minimalem Jitter.

Wie bereits im oberen Abschnitt erwähnt, erfordert das TDMA Verfahren synchrone interne Uhren der Knoten, damit alle Knoten die konfigurierten Zeitschlitze (Sende und Empfangszeitpunkte für Nachrichten) einhalten. Dafür definiert das TTEthernet- Protokoll ein Synchronisationsmechanismus. Ein oder mehrere sog. Synchronisations- Master starten die Synchronisierung, indem sie ihre Uhrzeit an einen Compression- Master verschicken. Dieser berechnet die globale Zeit und schickt diese an alle Knoten (Synchronisation- Clients) des Netzwerks zurück. Ob ein Knoten Synchronisations- Client oder eine andere Funktion übernimmt, wird konfiguriert.

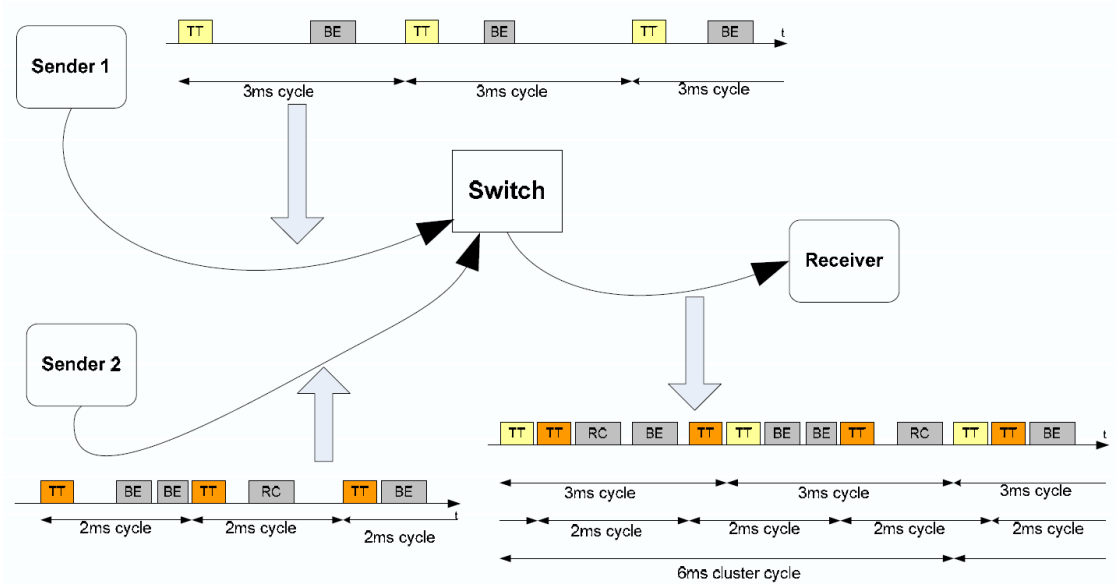


Abbildung 2.2: Ein beispielhaftes TTEthernet- Netzwerk (Quelle: [31])

2.4.1 Nachrichtenklassen

TTEthernet definiert drei Nachrichtenklassen, diese sind in Abbildung 2.2 zu erkennen: Time-Triggered (TT), Rate-Constrained(RC) und Best-Effort (BE). Diesen Klassen wird eine Priorität zugeordnet. TT/RC- Nachrichten sind echtzeitfähig und werden auch als Critical-Traffic bezeichnet.

- **Time-Triggered (TT) Nachricht- höchste Priorität**
  Die Sende und Empfangszeitpunkte werden im Schedule definiert. Es wird sichergestellt bzw. garantiert, dass eine TT- Nachricht zu den definierten Zeitpunkten bei den Knoten gesendet bzw. empfangen werden.
- **Rate-Constrained (RC) Nachricht- mittlere Priorität**
  Dieser Nachrichtenklasse werden keine bestimmten Sendezeitpunkte per Schedule zugeordnet. Es wird jedoch garantiert, dass eine gewisse Bandbreite der Nachricht bereitgestellt wird. Dies wird mit in Abbildung 2.3 gezeigten Bandwidth Allocation Gap (BAG) erreicht.

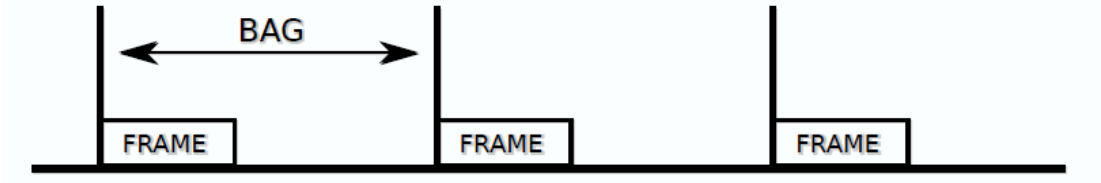


Abbildung 2.3: Bandwidth Allocation Gap (Quelle: [1])

Diese definiert die minimale/maximale Länge einer Nachricht und den Zeitabstand, wann die nächste Nachricht gesendet werden darf [6]. Der TTEthernet Switch verwirft eine RC- Nachricht, wenn der Sendeabstand nicht eingehalten wird. Diese Nachrichtenklasse ist kompatibel zu dem ARINC 664 Part 7 Standard (vgl. [1]) aus der Luftfahrtsbranche bzw. Avionik- auch AFDX genannt. Für RC Nachrichten muss der Teilnehmer nicht im Netzwerk synchronisiert sein, höher priorisierte TT Nachrichten haben immer Vorrang. Deshalb ist kein exakter Sende und Empfangszeitpunkte der Nachricht vorhersagbar. Trotzdem ist für RC- Nachrichten die Übertragung garantiert und es kann eine maximale Latenz (Worst- case delay) angegeben werden (vgl. [6]).

- **Best Effort (BE) Nachricht- niedrigste Priorität**
  TTEthernet baut auf dem Standard Ethernet (IEEE 802.3) auf. TTEthernet bietet die Möglichkeit, auch Knoten in das Netzwerk zu integrieren, die auch auf dem Standard Ethernet aufbauen. So ist es möglich, parallel ein TCP/IP Netzwerk zu betreiben. BE- Ethernet Nachrichten werden vom Switch erkannt und nur dann weitergeleitet, wenn keine TT oder RC- Nachricht ansteht. Damit ist keine Aussagen über die maximale Latenz einer BE- Nachricht möglich. Außerdem ist es nicht garantiert, dass eine BE Nachricht überhaupt übertragen wird.

2.4.2 Indentifizierung von Echtzeitnachrichten

Um Nachrichten von BE oder Critical-Traffic (TT oder RC) zu unterscheiden, muss in der Sicherungsschicht (Layer 2- OSI-7 Schichtenmodell) der Ethernet- Frameheader untersucht werden. Dieser ist in Abbildung 2.4 zu sehen. Ob es sich um Critical-Traffic (CT) handelt, wird anhand der 6 Byte langen Ziel- MAC- Adresse festgestellt (TT- Adresse). In den ersten 4 Bytes steht der Critical Traffic Marker. Anhand des Markers wird eine zeitkritische Nachricht erkannt. Jede zeitkritische Nachricht (TT und RC) besitzt außerdem eine eindeutige Critical Traffic ID, um sie richtig zuordnen zu können. Diese steht in den letzten 2 Bytes der TT-Adresse.

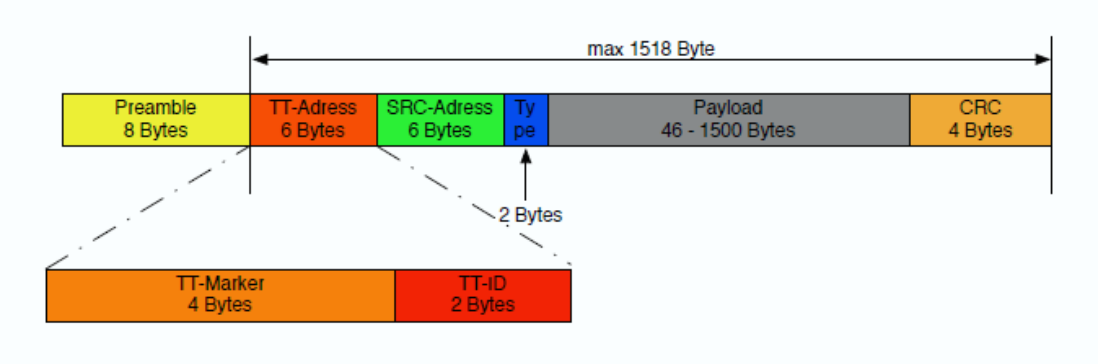


Abbildung 2.4: Aufbau eines Ethernet- Frames (IEEE 802.3) (Quelle: [24])

2.5 Demonstrator

Im Demonstrator Netzwerk werden unterschiedliche Anwendungen im Automotive Kontext gezeigt, die verschiedene Echtzeit- Anforderungen an das Kommunikationsmedium stellen. Wie bereits erwähnt, wird dies in heutigen Automobilen über verschiedene Bussyteme realisiert. Durch den Demonstrator wird gezeigt, wie zeitkritische Anwendungen mit harten Echtzeitanforderungen (Steer-Bye-Wire Lösung), zusammen mit bandbreitenintensiven Multimedia- Anwendungen (Streamen von Videos), über ein echtzeitfähiges Netzwerk - dem TTEthernet kommunizieren. Das TTethernet Protokoll wurde auf einem Mikrocontroller im Rahmen einer Bachelorarbeit implementiert (vgl. TTE- Protokollstack für eingebettete Systeme [24]).

Da echte Steuergeräte eingesetzt werden und diese größtenteils das CAN- Protokoll sprechen, müssen diese Nachrichten erst übersetzt werden, dazu wurde im Rahmen einer Bachelorarbeit eine Bridge entwickelt die Nachrichten von CAN auf TTEthernet übersetzt (vgl. [18]). Grundlage der Bridge, ist der eben erwähnte TTE- Protokollstack. Es befinden sich im Demonstrator Netzwerk auch Systeme mit „einfachen“ Echtzeitanforderungen - dafür wurde ein Scheinwerfersystem mit dem TTEthernet verbunden, welches mit RC-Nachrichten gesteuert wird. Ein weiterer Bestandteil des Demonstrators ist das Infotainmentsystem[3]. Über das Infotainmentsystem lassen sich die Parameter der Scheinwerfer und der Steer-By-Wire Lenkung verändern, außerdem visualisiert es den aktuellen Zustand des Scheinwerfers und der Steer-By-Wire Lenkung.

Auf dem Display des Infotainmentsystems kann ein Video- oder Kamerabild angezeigt werden. Der Video- und Kamerastream ist als BE- Traffic realisiert. In dieser Arbeit wird die Kamera entfernt und ein neues Kamerasystem mit Echtzeitanforderungen realisiert.

Die Vorteile werden im Verlauf der Arbeit erläutert. Die Abbildung 2.5 gibt einen Überblick über das Demonstrator- TTEthernet. Das Rad bzw. der Controller überträgt Erschütterungen (Force Feedback) an das Lenkrad. Zwischen den zwei Switches ist der Hauptstrang, das sog. Backbone des Netzwerkes, hier bündelt sich der Netzwerk- Traffic. Die Videoanwendung simuliert eine Multimediaanwendung und streamt ein Kamerabild - dies könnte ein embedded PC sein, der in der Kopfstütze des Fahrzeugs integriert ist.

[3] Ein Infotainmentsystem im Automobil stellt dem Fahrer verschiedene Funktionen im Bereich Navigation, Fahrerassistenz, Kommunikation und Multimedia zur Verfügung (vgl. www.wikipedia.de).

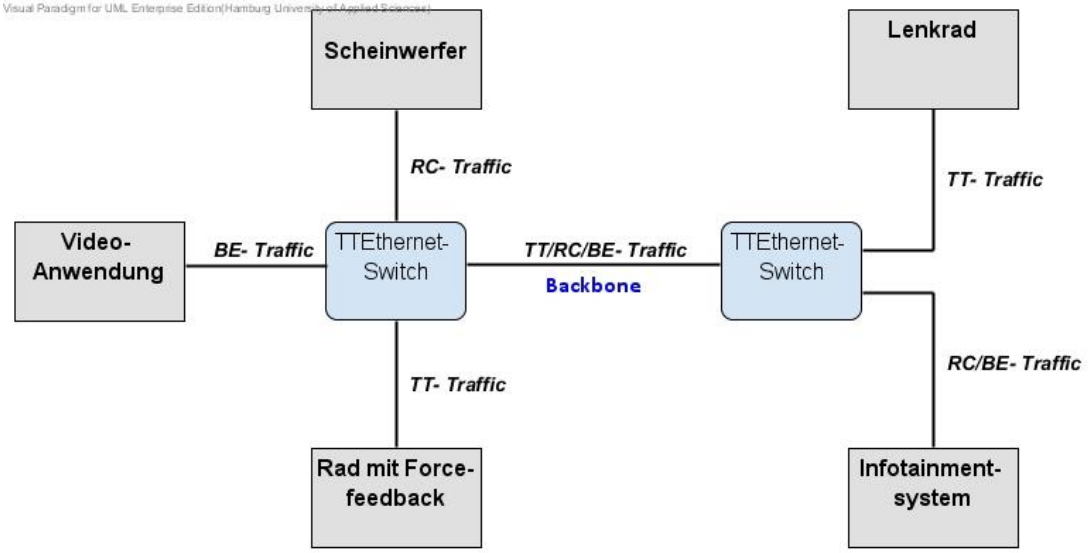


Abbildung 2.5: Demonstrator Netzwerk

2.6 Kodierverfahren- Kompression

Liefert eine Kamera unkomprimierte Bilder (z.B. im RGB-Format[4]), entsteht bei dessen Übertragung über das Netzwerk eine hohe Datenrate. Ein Bild besteht aus mehreren Pixeln, die in Zeilen und Spalten angeordnet sind. Ein Pixel braucht 24Bit zur Übertragung- 8 Bit für jeweils die Farben Rot, Grün und Blau. Die Auflösung eines Pixels kann auch höher oder niedriger als 24 Bit sein. Bei einer Bildgröße von 640 Spalten * 480 Zeilen wären das 7.372.800 Bit. Insgesamt liefert die Kamera bei einer Ausgabe von 25 Bildern pro Sekunde (fps) 184.320.000 Bit/s (184,32 Mbit/s). Eine Übertragung der Bilddaten über ein Netzwerk, welches auf 100 Mbit/s begrenzt ist, wäre somit ohne eine Komprimierung des Videobildes nicht möglich. In unkomprimierter Form enthalten die Bilder viel mehr Daten, als das menschliche Auge verarbeiten kann. Somit können diese Daten verworfen werden, ohne dass es dem menschlichen Betrachter auffällt. Bei einer Bildfolge kommt es außerdem häufig vor, dass sich nicht das komplette Bild ändert, sondern nur Ausschnitte. Als Beispiel sei eine Person genannt, die sich durch das Bild bewegt, der Hintergrund bleibt bei dieser Bildfolge gleich. Es gibt Kompressionsverfahren die Redundanzen in Bildern ausnutzen. Im folgenden Abschnitt sollen die Grundlagen von Kompressionsverfahren vorgestellt werden, diese werden in verschiedenen Codecs[5] benutzt.

2.6.1 Unterabtastung und Farbraum

Das menschliche Auge nimmt Helligkeitsunterschiede besser wahr, als Farbunterschiede (vgl. [35]), hier setzt auch das erste Kompressionsverfahren an. Liefert eine Kamera Bilder im RGB- Format, müssen diese erst in das YCbCr- Format umgerechnet werden. Das YCbCr-

[4]Durch das additive Mischen dreier Grundfarben (Rot, Grün und Blau) wird die Farbwahrnehmung des Menschen nachgebildet (vgl. www.wikipedia.org).

[5]Codec steht für Coder und Decoder oder auch für Kompression und Dekompression von einem Datenstrom (vgl. www.itwissen.info).

Format besteht aus dem Grundhelligkeitsanteil (Luminanz, Y) und dem Farbanteil (Chrominanz- Cb und Cr).
Die Umwandlung ist linear und ist nicht verlustbehaftet. Für eine Kompression wird nun der Farbanteil reduziert, der Helligkeitsanteil bleibt gleich. Das Menschliche Auge ist für grünes Licht besonders empfindlich, der Grünanteil steckt in der Grundhelligkeit Y. Dadurch kann die Chrominanz in geringerer Auflösung gespeichert werden, ohne dass die Bildqualität subjektiv für den menschlichen Betrachter abnimmt.
Dieser Prozess wird auch Unterraumabtastung bzw. Chroma-Subsampling genannt [vgl. [25]). Beim Chroma 4:2:0 Subsampling wird das Bild über 4 Pixel, horizontal und vertikal abgetastet, dabei wird nur ein Chrominanz- Wert für diese Pixel gespeichert. Die restlichen 3 Chrominanz- Werte werden verworfen. In Abbildung 2.6 ist dieser Zusammenhang sichtbar. Durch diesen Prozess wird eine Datenreduktion von 50% erreicht (vgl. [29].

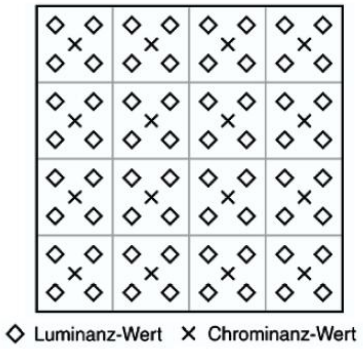


Abbildung 2.6: Chroma Subsampling 4:2:0 (Quelle:[29])

2.6.2 Makroblock und Diskrete Kosinustransformation (DCT)

Codecs wie MPEG-4 bzw. h.264 oder auch JPEG teilen das Bild außerdem noch in die sog. Makroblocks ein. Diese können unterschiedliche Größen haben, wie z.B., 16x16, 8x8 oder 4x4 Pixel. Je nach Anforderung an die Bildqualität, steht jedem Makroblock eine gewisse Menge an Speicher zur Verfügung. Je weniger Speicher einem Makroblock zur Verfügung steht, desto schlechter wird die Bildqualität und die Makroblocks werden auf dem Bild sichtbar. Der jetzige Farbraum (YCbCr), aus vorherigem Abschnitt, liefert nur Informationen über Helligkeit und Farbanteil in einer rechteckigen Fläche (Ortsbereich), aus diesen Informationen lassen sich keine Informationen über wichtige (komplexe) oder unwichtige (einfache) Bilddetails ableiten. Diskrete Kosinustransformation (DCT) wandelt die Makroblocks aus dem Ortsbereich in einen Frequenzbereich um und bewertet dadurch den Makroblock hinsichtlich der Komplexität (vgl. [11]). Liegen die Daten im Frequenzbereich vor, können Aussagen über die Wichtigkeit des Bildausschnittes getroffen werden. Hohe Frequenzen zeugen von komplexen feinen Strukturen, niedrige dagegen von homogenen Flächen (z.B. blauer Himmel) (vgl. [29]). Die Umwandlung vom Ortsbereich in den Frequenzbereich ist theoretisch linear, d.h. eine Umwandlung kann ohne Verlust rückgängig

gemacht werden (IDCT- Inverse Diskrete Kosinustransformation). In der Praxis lässt sich dies aber nur annähernd genau berechnen, sodass auch dieser Prozess verlustbehaftet ist (vgl. [29]). Durch die DCT entstehen aus einem 8x8 Makroblock die sog. DCT- Koeffizienten, welche in Abbildung 2.7 zu sehen sind.
Die DCT- Koeffizienten werden zwischen einem DC- Koeffizienten und 63 AC Koeffizienten unterschieden. Der DC- Koeffizient, auch Gleichanteil genannt, bestimmt den mittleren Farbton eines Makroblocks. Seine Frequenz ist 0 Hz in beiden Bildachsen.
Die 63 AC- Koeffizienten, auch Wechselanteil genannt, haben eine Frequenz größer als 0 Hz. Die Frequenzen nehmen in horizontaler und vertikaler Richtung zu. Da ein Bild hauptsächlich aus homogenen Flächen und weniger aus komplexen Flächen- wie scharfen Kanten besteht, gibt es weniger DCT- Koeffizienten, die hohe Frequenzen repräsentieren (vgl. [34] und [29]).

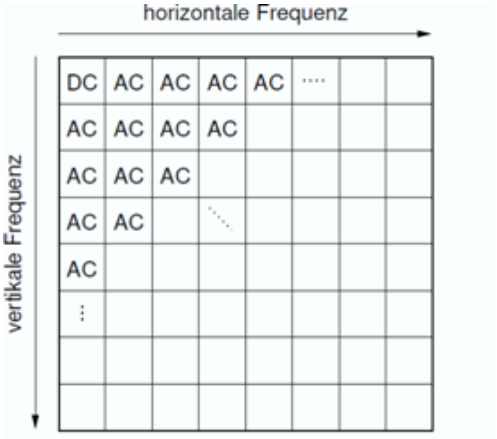


Abbildung 2.7: DCT Koeffizienten für einen 8x8 Makroblock (Quelle: [34])

2.6.3 Quantisierung und Kodierung der DCT- Koeffizienten

Durch die diskrete Kosinustransformation ist noch keine Datenreduktion erreicht worden. Der Bildbereich ist lediglich nach Komplexität bewertet. Ab hier setzt die Quantisierung ein. Die zuvor 64 generierten DCT Koeffizienten werden nun durch die sogenannte Quantisierungstabelle dividiert und als Ganzzahl gerundet. Die Quantisierungstabelle ist so aufgebaut, dass sie die menschliche Wahrnehmung berücksichtigt, über Testreihen wird diese ermittelt (vgl. [34]). Wird nun die DCT Koeffiziententabelle bzw. Matrix mit der Quantisierungsmatrix dividiert, wird der Wertebereich verkleinert, viele DCT Koeffizienten werden zu 0. Dieser Prozess ist verlustbehaftet, d.h. man kann durch Umkehrung nicht die Ausgangsqualität des Bildes erreichen. Bei einer starken Quantisierung können Blockartefakte entstehen, werden alle AC- Koeffizienten zu null, nimmt der Makroblock den Farbton des DC- Koeffizienten an. Die Qualitätseinstellungen für einen Codec modifizieren meistens diese Quantisierungstabellen (vgl. [29]).

- Die DC- Koeffizienten werden mit einem prädiktiven (auf Vorhersagen beruhenden) Verfahren codiert. Es wird davon ausgegangen, dass benachbarte Pixel einen

geringen Unterschied ihrer DC- Werte aufweisen (Korrelation). Anstatt nun jeden DC- Koeffizienten einzeln zu speichern, wird jeder DC- Wert aus seinem Vorgänger vorhergesagt. Dafür werden alle DC- Werte des Bildes durchlaufen und die Differenz (Prädiktionsfehler) zwischen Vorhersage und wahrem DC- Wert gespeichert.
Anschließend werden die Prädiktionsfehler durch Huffmann- Kodierung entropiecodiert. Das bedeutet, dass ein häufig vorkommender Prädiktionsfehler-Wert mit einem kurzen Codewort codiert wird und ein weniger häufig vorkommender Prädiktionsfehler- Wert mit einem langen Codewort. Abgespeichert oder übertragen werden dann nur noch die Codewörter. Dies setzt natürlich voraus, dass der Kodierer (Encoder) und Dekodierer (Decoder) die gleichen Huffman- Codetabellen verwenden, um den richtigen DC- Koeffizienten wiederherzustellen (vgl. [34] und [29]).

- Die AC- Koeffizienten werden durch die sog. Zick-Zack- Sortierung aufgereiht, durch die Quantisierung sind viele AC- Koeffizienten zu 0 geworden. Durch diese Aufreihung stehen die hochfrequenten AC- Koeffizienten hinten. Dabei entstehen lange Ketten von Nullen. Diese Eigenschaft wird durch sog. Lauflängencodierung ausgenutzt. Dabei wird die Anzahl der Nullen und die Anzahl der Bits, die für die Kodierung des nachfolgenden AC- Koeffizienten benötigt werden als sog. Symbol codiert. Danach folgt der AC- Koeffizient als Wert. Nach diesem Muster werden alle AC- Koeffizienten codiert. Wie bei der Kodierung der DC- Koeffizienten werden die Symbolfolgen durch Huffmann- Kodierung entropiecodiert. Häufig auftretende Symbolfolgen werden kurze Codewörter zugeordnet und weniger häufig vorkommende Symbolfolgen lange Codewörter (vgl. [34] und [29])

Bei der Kodierung der DCT- Koeffizienten handelt es sich um Redundanzreduktion, d.h. man erzielt eine Datenkompression, welche nicht verlustbehaftet ist.
Verlustbehaftete Kompression wird Irrelevanzreduktion genannt, diese setzen eine DCT voraus, auch das Chroma- Subsampling 4:2:0 ist eine verlustbehaftete Kompression.
Die beschriebenen Verfahren sind nur ein kleiner Teil, der in heutigen Codecs angewandten Verfahren, dennoch bauen die in dieser Arbeit verwendeten Codecs darauf auf. Bis jetzt wurde nur ein Einzelbild betrachtet und kodiert, da es sich bei der Rückfahrkamera um Bildfolgen handelt, werden noch weitere Verfahren eingesetzt. Die im nächsten Abschnitt vorgestellten Verfahren beziehen sich auf Bildfolgen, mit diesen Verfahren wird der größte Kompressionsfaktor erzielt- zudem sind diese Verfahren nicht verlustbehaftet- es sind Redundanzreduktionen.

2.6.4 Differenzenkodierung

Differenzenkodierung baut darauf auf, dass sich bei aufeinanderfolgenden Bildern kaum etwas ändert. Es werden nur die Änderungen zum vorherigen Bild übertragen. Am Anfang einer Videoanwendung muss zuerst ein Vollbild übertragen werden, ein sogenanntes I(Intra

Coded Picture)- Frame. Dieses wird unter anderem nach den vorherigen beschriebenen Kodierverfahren kodiert. Das nächste Bild wird nicht als Vollbild übertragen, sondern als sogenannter P(Predictive Codec Picture)- Frame. Wie in Abbildung 2.8 zu sehen ist, werden nur die Änderungen zum vorherigen I- Frame als Differenzbild gespeichert. Dabei kann der Vorgänger selbstverständlich auch ein anderer P- Frame sein. Ein B(Bidirectional Codec Picture)- Frame ist Abhängig von vorausgegangenen und nachfolgenden I- oder P- Frames. Nachfolgende Bilder können bei einer Live- Videoübertragung jedoch nicht genutzt werden.

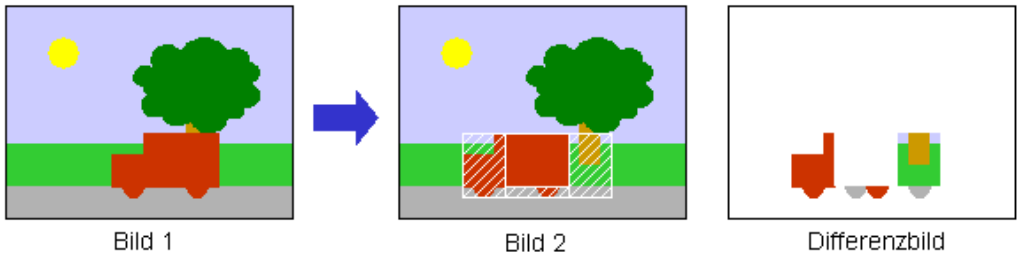


Abbildung 2.8: Differenzenkodierung (Quelle: [2])

2.6.5 Bewegungskompensation (Motion Compensation)

In der vorherigen Abbildung 2.8, mit einfacher Differenzkodierung, bewegt sich nur ein Objekt auf dem Bildschirm. Hier gibt es noch ein anderes Verfahren, welches die Bewegung im Bild erkennt. Der Algorithmus sucht nach wiederkehrenden Makroblöcken in beiden Bildern und errechnet daraus Bewegungsvektoren. Anstatt die komplette Differenz zu übertragen, wird der Bildbereich mittels des Bewegungsvektors verschoben und danach die Differenz kodiert. Dieser Vorgang ist in Abbildung 2.9 dargestellt. Dadurch wird die zu übertragende Datenmenge nochmals reduziert

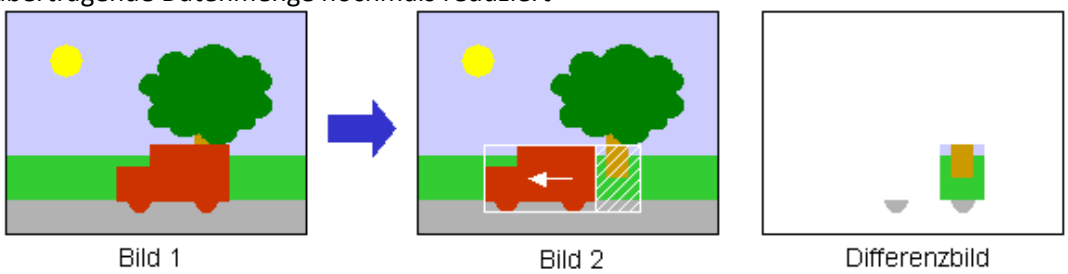


Abbildung 2.9: Bewegungskompensation (Quelle: [2])

2.7 Digitaler Signal Prozessor (DSP)

Ein Einsatzgebiet von DSPs ist der Automotive Bereich. Im Antiblockiersystem, in der Motorregelung oder in der aktiven Fahrzeugfederung werden sie eingesetzt. Hauptaufgabe ist die Verbreitung von analogen Signalen, die z.B. von Temperatur-, Druck-, Kraft- oder Geschwindigkeitssensoren übermittelt werden.
Über einen Analog Digital Wandler werden die Signale digitalisiert und können dann weiterverarbeitet werden- z.B. in einem Regelalgorithmus. Im Gegensatz zum Mikrocontroller ist der DSP mit speziellen Befehlen und Hardware ausgestattet, um für die Signalverarbeitung wichtige Algorithmen schnell zu verarbeiten. Ein weiteres Einsatzgebiet ist der Multimediabereich.
Im vorherigen Abschnitt wurden die Grundlagen der Videokompression vorgestellt, die auch als Teil der digitalen Bildbearbeitung bezeichnet werden können. So ist ein typischer DSP- Algorithmus die diskrete Kosinustransformation (DCT), eine der wichtigsten Transformationen in der Bilddaten- Kompression. Ein Digitaler Signalprozessor führt arithmetische Operationen in einem Taktzyklus aus, dies ist durch seine spezielle Architektur möglich. Die Operanden werden aus separaten Speicherbereichen geladen und der Datenspeicher ist vom Programmspeicher getrennt (Harvard Architektur).
Typische Operationen der digitalen Signalverarbeitung bestehen aus kombinierten Multiplikations- und Additionsoperationen, wie die MAC (Multiply- Accumulate) Operation: c = c + (a*b). Diese Berechnung erfolgt in einem Taktzyklus. Außerdem kann ein DSP auch mehrere Operationswerke besitzen und dadurch mehrere Berechnungen parallel ausführen (vgl. [5]).

3 Analyse und Konzept

Im folgenden Kapitel wird auf Video- Codecs eingegangen und deren Datenreduktion. Darauf aufbauend, wird diskutiert, welche Nachrichtenklasse für die Übertragung der Videodaten sinnvoll ist. Zudem wird analysiert, wie sich die Kamera- bzw. die Kameraplattform in das TTEthernet integrieren lässt. Anhand dieser Informationen wird durch Auswahlkriterien die dazugehörige Hardware ausgesucht. Zum Schluss werden die Software Erweiterungen und Neuentwicklungen untersucht und konzipiert, um das Rückfahrkamerasystem zu realisieren.

3.1 Video- Codecs und Datenreduktion

Wie bereits in den Grundlagen erläutert, wäre das Übertragungsmedium von 100 Mbit/s Bandbreite mit der Übertragung von nur einem unkomprimierten Videosignal bereits ausgelastet. Je nach Kodierverfahren, ist eine Datenreduktion, um den Faktor 1:5 bis 1:200 möglich (vgl. [11]). Abbildung 3.1 gibt eine Übersicht über die verbreiteten Standards der Videokodierung.

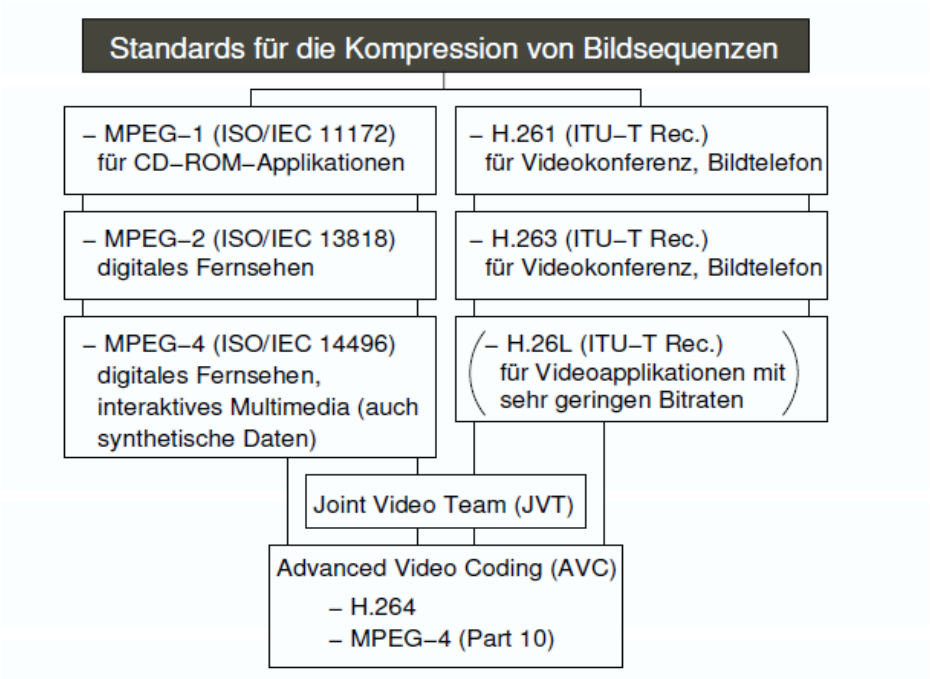


Abbildung 3.1: Standards für Videokodierung (Quelle: [11])

H.264 ist für Videostreaming- Anwendungen und Anwendungen mit hohen Qualitätsansprüchen geeignet. Im Gegensatz zu MPEG-2 und H.263 ist die Einsparung an Bitrate ca. 40% oder mehr (vgl. [11]). Im Jahre 2001 arbeiteten die ITU und die MPEG-Gruppe im Rahmen des Joint Video Team (JVT) zusammen und es entstand der H.264/MPEG-4 (Part 10) oder auch AVC genannter Codec (vgl. [29]). Alle diese Codecs geben eine gewisse Bandbreite für den Videostream an, z.B. 4 MBit/s. Wird der Videostream genauer betrachtet, erkennt man, dass die maximale Datenrate innerhalb dieser Übertragungszeit sehr viel höher sein kann. Durch die zyklisch auftretenden I- Bilder (vgl. 2.6.4) wird ein komplettes Einzelbild übertragen. Wenn das Übertragungsmedium eine zu geringe Bandbreite besitzt und das I- Frame nicht bereits in einen Puffer geladen wurde, ruckelt die Videowiedergabe oder die Videowiedergabe bleibt stehen, bis genügend Daten zwischengepuffert wurden und das I- Bild angezeigt werden kann. Um dieses Problem zu kompensieren, gibt es noch weitere Techniken. Dabei wird die Qualität des Videos in solchen Fällen reduziert, sodass die Bandbreite für die Übertragung ausreicht, auf Kosten der Qualität. Das Resultat ist eine angepasste konstante Bitrate. Dieses Konzept nennt man Constant Bitrate (CBR). Wird gleichbleibende Qualität verlangt, kommt die Variable Bitrate (VBR) zum Einsatz (vgl. [11]). Je nachdem wie komplex die Szene ist, erhöht sich auch die Datenrate. Durch diese dynamisch Anpassung der Datenrate und eine nicht ausreichende Bandbreite, kann es wie bereits erwähnt zu Bildaussetzern kommen. Abbildung 3.2 zeigt, wie bei der VBR die Datenrate bei Szenen/Sequenzen mit viel Bewegung ansteigt, dies ist bei der CBR nicht der Fall.

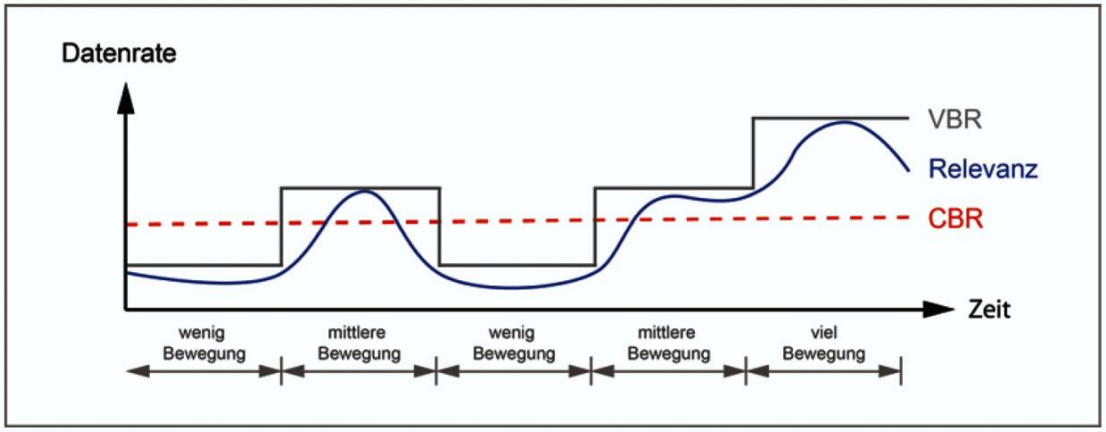


Abbildung 3.2: Zusammenhang zwischen komplexen Szenen und Bitrate (Quelle: [11])

Im Kontext einer Rückfahrkamera, die ein Live- Bild liefert, muss demnach das Kommunikationsmedium so viel Bandbreite zur Verfügung stellen, dass I- Frames, als „Worst Case Szenario“ möglichst kaum verzögert werden. Das Puffern- bzw. eine zeitversetze Wiedergabe des Kamerabildes ist keine Option, da es im Kontext einer Rückfahrkamera nicht zu Bildaussetzern oder Verzögerungen kommen darf. Die Konzeption der Bandbreite ist ein entscheidender Faktor, ein Encodier mit CBR ist von Vorteil, um Bildaussetzer zu minimieren.

3.2 Wahl der Nachrichtenklasse

Um die Kameradaten über das Netzwerk zu transportieren, stehen grundsätzlich drei verschiedene Nachrichtenklassen zur Verfügung (TT, RC und BE vgl. 2.4.1). Im folgenden Abschnitt soll analysiert werden, welche dieser Nachrichtenklassen für die Rückfahrkamera sinnvoll ist.

Für die Qualität von Video- Übertragungssystemen gibt es zwei negative Effekte, die direkt mit dem Kommunikationsmedium zusammenhängen. Zum einen das Auseinanderreißen des Bildes (Slicing) und das Ruckeln (Jerkiness). Durch die Wahl der Nachrichtenklassen lässt sich zumindest das Ruckeln beheben, denn dieses entsteht durch verspätete Pakete (engl. Packet Discard) oder durch Paketverlust (engl. Packet Loss).
Slicing entsteht bei Übertragungsfehlern und äußert sich im allg. durch den Verlust von Bildbereichen, decodiert der Decoder gerade einen fehlerhaften Bildbereich (Slice, typischerweise 4 Makroblocks), treten dort Artefakte[6] auf (vgl. [25]). Der Slicing Effekt (Übertragungsfehler) kann nicht durch die Wahl der Nachrichtenklasse behoben werden. Jerkiness (ausgelöst durch Paketverlust/Verzögerung), macht sich durch ruckartige Bewegungsartefakte bemerkbar, das Video wird nicht mehr flüssig dargestellt. Durch die Wahl der Nachrichtenklasse lässt sich der negative Jerkiness Effekt minimieren.
Wie bereits kennengelernt, ist das Backbone des Demonstrators ein Switched Ethernet mit dem TTTech- Protokoll. Wird für den Videostrom die BE- Nachrichtenklasse gewählt, so kann es passieren, dass ein Paketverlust auftritt oder Pakete verzögert werden, da RC und TT Nachrichten immer Vorrang vor BE Nachrichten haben. Paketverlust tritt dann auf, wenn die Kameraplattform mehr Pakete sendet, als das Switch weiterleiten kann. Ist der Buffer des Switches voll, werden neue ankommende Pakete verworfen.
Jeder Sendezeitpunkt und Empfangszeitpunkt einer TT Nachricht wird klar definiert, dabei ist es garantiert, dass diese Nachricht übertragen wird. Dieser Nachrichtentyp ist somit prinzipiell für die Videoübertragung geeignet und verhindert Paketverluste. Der Nachteil ist jedoch, dass für jede definierte TT Nachricht die nutzbare Bandbreite reduziert wird. Werden keine Videodaten übertragen oder die Live- Bild- Szene ist von geringerer Komplexität bzw. produziert kaum Bandbreite, so ist das Übertragungsmedium trotzdem durch die definierten TT Nachrichten für die Videoübertragung eine gewisse Zeit reserviert. Ein Switch muss einen definierten Zeitpunkt (Acceptance Window) warten, bis er eine andere Nachricht verschicken kann.
Als letzte zu untersuchende Nachrichtenklasse bleibt RC. Mit RC Nachrichten kann dem Videostream eine Bandbreite garantiert werden, außerdem ist auch hier die Übertragung der Nachricht garantiert. Von Vorteil ist auch die Tatsache, dass die Videoplattform nicht mit dem Netzwerk synchronisiert werden muss, dies eröffnet verschiedene Möglichkeiten, die Kameraplattform in den Demonstrator zu integrieren.
Bei weniger komplexen Szenen, mit einer geringeren Anzahl von RC Nachrichten, wird das Übertragungsmedium nicht belegt. Andere RC oder BE Nachrichten können übertragen werden. Wie bereits in den Grundlagen erwähnt (vgl. 2.4.1), muss der Mindest-sendeabstand zwischen zwei Nachrichten eingehalten werden, da sonst der Switch die

[6] Eine sichtbare, unerwünschte Anzeige in digitalen Bildern, die nicht von den Ausgangsdaten des Bildes hervorgerufen werden (vgl. www.wikipedia.de).

Nachricht verwirft. Bezogen auf den Videostream darf der BAG Value (der Sendeabstand) nicht zu groß sein, da sonst die Pakete zu verzögert beim Empfänger ankommen. Dies verursacht dann ggf. wieder negative Jerkiness Effekte.
Nach der näherer Betrachtung der verschiedenen Nachrichtenklassen und dessen Vor- und Nachteilen wird der RC- Nachrichttyp gewählt.

3.3 Analyse der Verbindungsmöglichkeiten der Kameraplattform zum TTEthernet

Im folgenden Abschnitt wird analysiert, welche Möglichkeiten es gibt, die Kameraplattform in den Demonstrator zu integrieren. Möglichkeiten sind die direkte Anbindung mit einem TTEthernet Netzwerkkarten- Treiber, einem Mikrocontroller als Bindeglied (Bridge) zum TTEthernet oder einem RC- konformen Ansatz. Es werden die Vor- und Nachteile der jeweiligen Möglichkeiten aufgezeigt.

3.3.1 Kameraplattform mit direktem Zugang zum TTEthernet

Auf der Kameraplattform läuft ein Linux Betriebssystem mit einem TTE (TTEthernet)-Protokollstack. Dadurch ist es möglich, die Kameraplattform direkt an ein Switch im Netzwerk anzuschließen und TTEthernet konforme Nachrichten zu senden und zu empfangen (alle Nachrichtenklassen). Die Firma TTTech bietet ein TTE- Protokollstack (Kernel- Modul) an, zusammen mit einer Beispielanwendung, welche die Arbeitsweise des Moduls veranschaulicht. Der TTE- Protokollstack läuft aber nur auf spezieller Hardware, genauergenommen mit Netzwerkkarten- Chipsätzen der Firma Intel (e1000e), Realtek (r8169) und Attansic (ar8132). Außerdem, wird die Linux- Kernel Version 2.6.24 mit Real-Time Patch benötigt. Abbildung 3.4 zeigt den Aufbau der Software und den TTE-Protokollstack bzw. den TT- Protokollayer.

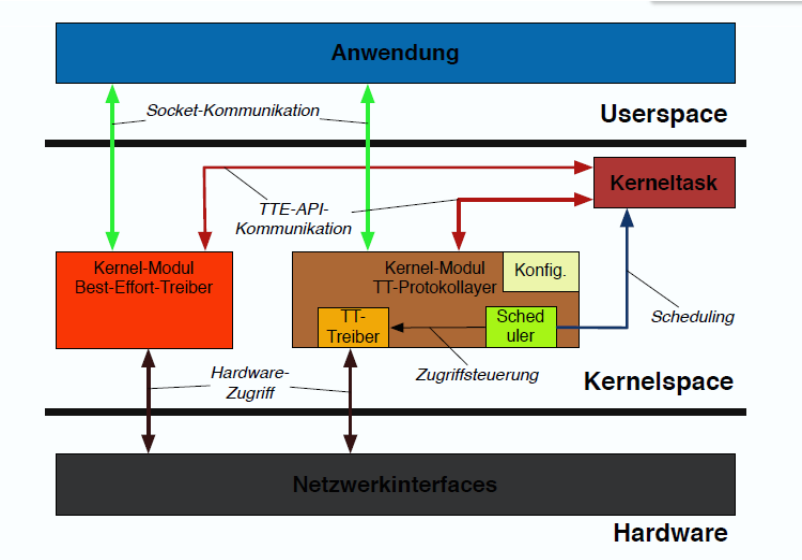


Abbildung 3.3: Überblick TTTech Software- Struktur (Quelle: [3])

Wie zu sehen ist, kann eine User- Space Anwendung über das Linux- Socket- API mit dem TT- Protokolllayer kommunizieren, die send()- und receive()- Funktionen wurde als blockierender Aufruf implementiert (vgl. [3]).
Auf das TTE- API (vgl. [38]) kann nur im Kerne- Space zugegriffen werden. Über das TTE- API können z.B. Ereignisse an eintreffende Nachrichten gebunden werden (Callback- Funktion), weiterhin bietet das API Zugriff auf die TT/BE/RC Buffer, in denen die jeweiligen Nachrichten gespeichert sind. Da die API nur im Kernel- Space verfügbar ist, kann eine Anwendung, die diese nutzen will, nur gegen den Kernel selbst gelinkt werden. Dies hat zur Folge, dass keine Bibliotheken aus dem User- Space eingebunden werden können (vgl. [10]), wie z.B. die Bibliothek eines Codecs. In einer Konfigurationsdatei wird unter anderem der Ablaufplan (engl. Schedule) für die TT- Nachrichten und die jeweiligen Buffer konfiguriert. Zurzeit wird im Rahmen der CoRE- Projektgruppe ein neuer TTE- konformer Protokollstack, für eine neuere Linux- Kernel Version 3.x entwickelt. Auch dieser wird nur mit einer speziellen Netzwerkkarte nutzbar sein. Da die Kameraplattform unabhängig von einer speziellen Netzwerkkarte und Kernelversion bleiben soll, ist diese Art der Anbindung nicht geeignet.

3.3.2 Verwendung des TTE- Protokollstack für eingebettete Systeme

In einer vorherigen Bachelorarbeit im Rahmen des CoRE- Projekts, wurde die TTEthernet Spezifikation (vgl. [32], heute standardisiert als SAE AS6802 [31]) und das TTE- API auf einer Mikrocontroller- Plattform realisiert (vgl. [24]). Der Mikrocontroller- basierende TTE- Protokollstack kann dadurch leicht in das Demonstrator- TTEthernet integriert werden. Bei der verwendeten Hardware handelt sich um ein NXHX500- ETM Evaluations Board (vgl. [22]) der Firma Hilscher. Kernstück des Boards ist ein netX 500 System- On- Chip Design mit

einem ARM 926EJ-S GPP (200 MHz). Das Board verfügt über zwei 100 MBit/s Ethernet-Schnittstellen (siehe Abbildung 3.4). Eine Schnittstelle wird für die Kommunikation zum TTEthernet verwendet, die andere ist frei verfügbar.
Es bietet sich nun an, über die verfügbare Ethernet- Schnittstelle eine Kameraplattform anzuschließen. Die Kommunikation mit dem TTEthernet übernimmt der in Abbildung 3.5 dargestellte TTE- Protokollstack.
Das Sync. Modul ist für die Zeitsynchronisation im TTEthernet zuständig. Wie in den Grundlagen kennengelernt (vgl. 2.4), gibt es verschiedene Rollen im Synchronisationsprozess (Client und Master), diese werden mit dem Rx- und Tx Sub- Modul realisiert. Für das Senden von TT- Nachrichten oder zeitgesteuerte Tasks ist das Scheduler- Modul zuständig. Wie bei der Anbindungsmöglichkeit aus vorherigem Abschnitt, findet die Konfiguration für den Schedule und die Konfiguration der TT- und RC- Buffer über eine Konfigurationsdatei statt. Dem Anwendungsprogrammierer steht das TTEthernet API zur Verfügung.
Durch diese Wahl der Anbindung, ist eine freie Wahl der Hardware- und Software Architektur der Kameraplattform möglich. Einzige Voraussetzung ist ein eine Ethernet-Schnittstelle.

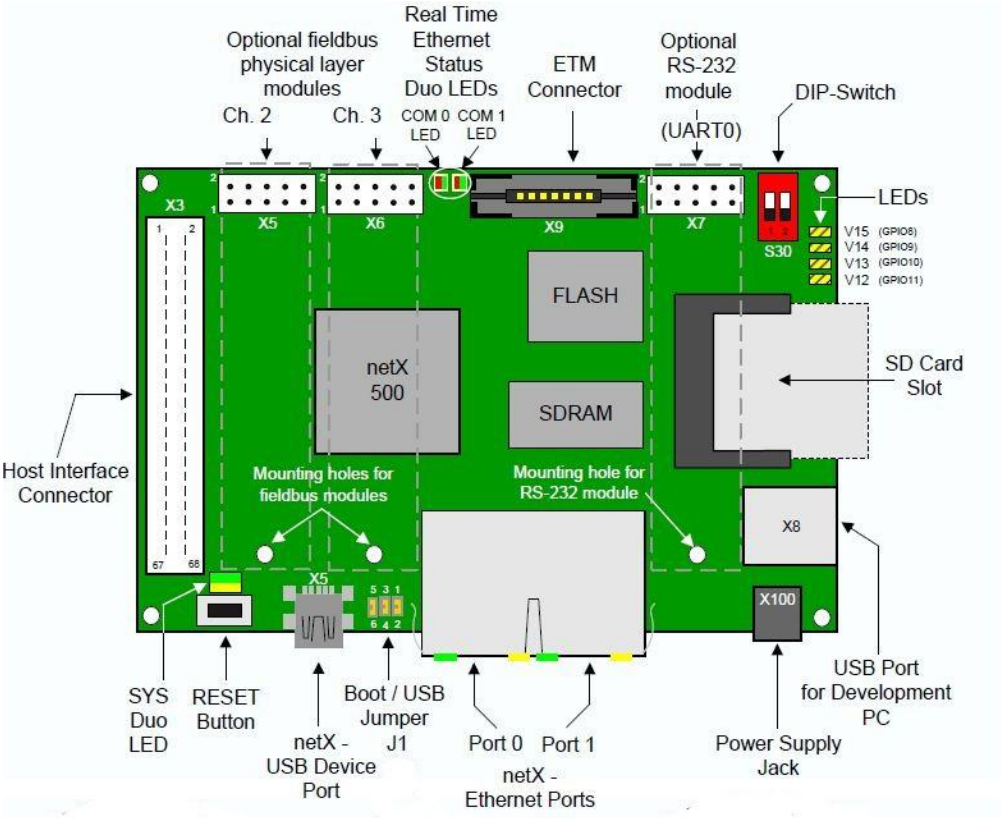


Abbildung 3.4: Entwicklungsboard NXHX500- ETM (Quelle: [22])

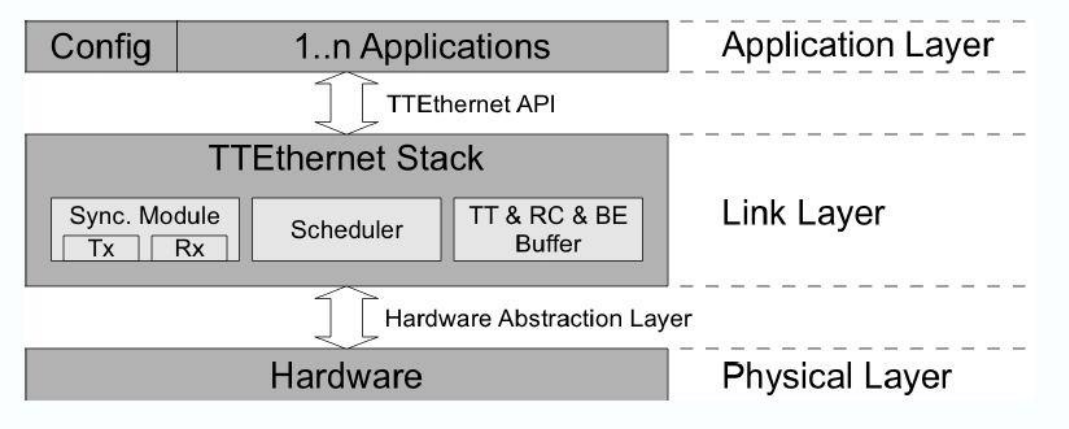


Abbildung 3.5: Mikrocontroller basierender TTE- Protokollstack (Quelle: [24])

3.3.3 RC- konforme Anbindung an das TTEthernet

Wie in den Grundlagen kennengerlernt (vgl. 2.4.1), ist für die Übertragung von RC-Nachrichten keine Zeitsynchronisation mit dem TTEthernet nötig. Dies bietet die Möglichkeit, eine beliebige Kameraplattform, ohne TTE- Protokollstack zu realisieren. Damit ist die Kameraplattform, wie die Anbindungsmöglichkeit aus vorherigem Abschnitt, unabhängig von einem bestimmten Betriebssystem und einer bestimmten Netzwerkkarte. Es muss lediglich sichergestellt sein, dass die zu sendenden RC- Nachrichten den BAG- Value einhalten, damit die Switche im TTEthernet die RC- Nachrichten nicht verwerfen. Der Nachteil liegt in der fehlenden Unterstützung, TT- Nachrichten zu senden.

3.3.4 Auswahl für die Anbindung der Kameraplattform

Nach der Vorstellung der möglichen Anbindungsvarianten einer Kameraplattform mit dem TTEthernet, wird in dieser Arbeit der Ansatz unter der Verwendung des Mikrocontroller basierenden TTE- Protokollstacks gewählt. Mit diesem Ansatz kann eine Kameraplattform frei gewählt werden. Das Senden und Empfangen von TT, RC und BE- Nachrichten ist möglich. Ferner kann die Kameraplattform für künftige Weiterentwicklungen als Advanced-Driver- Assistance- System (ADAS) dienen. Um eine Anwendung zu nennen: Aufgenommene Bilder der Kamera werden mit Imageprocessing auf Gefahrensituationen, wie plötzlich auftretende Personen analysiert und die Kameraplattform kann über den Mikrocontroller basierten TTE- Protokollstack eine zeitkritische Nachricht (TT oder RC) an ein anderes Steuergerät im Automobil senden.

3.4 Auswahl der Hardware für die Kameraplattform

An die Kameraplattform wird eine Reihe an Anforderungen gestellt. In Tabelle 3.1 sind diese aufgelistet. Danach werden einige Hardware- Architekturen kurz vorgestellt und Anhand der Anforderungen eine Plattform gewählt.

#	Anforderung
1	Die Plattform muss mindestens eine Ethernet- Schnittstelle zur Kommunikation bereitstellen
2	Mindestens eine der folgenden Anschlussmöglichkeiten/Schnittstellen für eine Kamera unterstützen: Analog- oder Digitales Video Interface, FireWire, USB 2.0 Host/OTG
3	Für embedded- Bereich im Kontext Automotive geeignet
4	Hardware unterstütztes Video- Encoding, oder genug Leistung für die Encodierung eines VGA (640*480 Pixel)- Live- Bildes
5	Entwicklung mit lizenzfreier Linux- Distribution möglich
6	Möglichkeit für weiterführende Projekte im Bereich Driver Assistance Systems (ADAS)
7	Display Anschluss für Debugging- Aufgaben

Tabelle 3.1: Anforderungen an die Kameraplattform

In die nähere Auswahl kommen Hardwareplattformen- Evaluationsboards der Firma Analog Devices, Texas Instruments und Freescale. Kern der Plattformen ist ein Mikroprozessor (MCU) oder ein Digitaler Signal Prozessor (DSP), sowie Mischformen.

Freescale MPC5604E EVB:

Ein Design speziell auf den Automotive Bereich ausgelegt. Kern des Designs ist ein 32 Bit, 64MHz PowerPC Mikroprozessor (Qorivva e200 zen0h). Wie in Abbildung 3.6 zu erkennen ist, steht neben dem Hardware M-JPEG Video- Encoder (mit Kamera Sensor Interface) eine Ethernet- Schnittstelle zur Verfügung. Für dieses Board gibt es kein Betriebssystem. Mit diesem Board lässt sich eine rein Mikrocontroller programmierte Kameraplattform realisieren. Für weitere Informationen, vgl. [9]. Kostenpunkt 150 $.

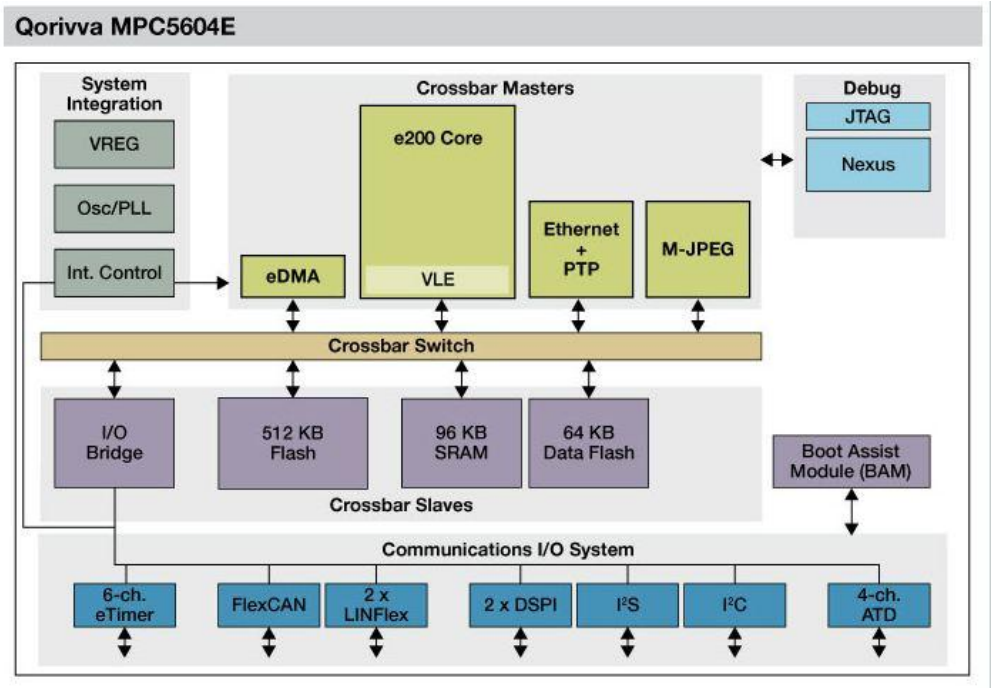


Abbildung 3.6: Blockdiagramm des Freescale Qorivva MPC5604E (Quelle: [9])

Freescale i.MX53QSB Quick Start Board mit i.MX536 Multimedia Application Prozessor:

Wie in der Abbildung 3.7 (nächste Seite) zu erkennen ist, wird bei diesem Design ein ARM Cortex- A8 800 MHz Mikroprozessor verwendet, eine für die Kameraplattform wichtige Video Processing Unit (VPU) und ein Kamera Interface ist außerdem vorhanden.
Die VPU encodiert bis zu 4 Videosignale simultan. Viele aktuelle Codecs, die in Abbildung 3.1 zu sehen sind, werden unterstützt (MPEG2, MPEG4, H.264). Dieses Board ist außerdem dazu geeignet, eine Realisierung nach Anbindungsmöglichkeit 3.3.1 zu ermöglichen, da die Netzwerkkarte eine Time Stamp Unit (TSU) besitzt, die für die Entwicklung eines TTE-Protokollstacks essentiell ist. Außerdem ist die Hardware IEEE 1588 kompatibel. Dies ist ein Protokoll für die Zeitsynchronisation in einem Netzwerk. Für weitere Informationen vgl. [9]. Als Linux- Distribution steht unter anderem Ubuntu zur Verfügung. Kostenpunk 149$.

Analog Devices BF537 EZ-KIT Lite:

Bei diesem Design (Abbildung 3.8) wird ein Blackfin 600 MHz Prozessor verwendet (Digitale Signal Prozessor). Es steht eine Ethernet- Schnittstelle zur Verfügung. Für den Anschluss einer Kamera muss eine Erweiterungskarte mit dem Board verbunden werden (A-V EZ-EXTENDER). Neben einem digitalen Kamera- Sensor Interface bietet die Erweiterung auch einen Anschluss von analogen Kameras. Zudem kann ein Display angeschlossen werden. Dadurch wird das Gesamtsystem teurer als die anderen Evaluationsboards. Eine lizenzfreie Linux Distribution ist verfügbar (uCLinux). Kostenpunkt 250 $ für das Board und 230 $ für die Erweiterungskarte.

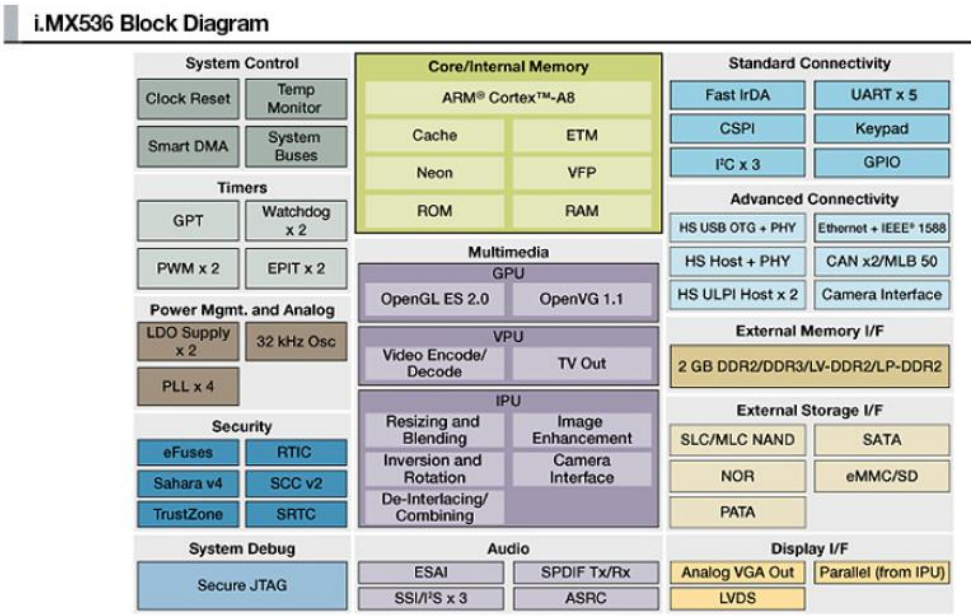


Abbildung 3.7: Blockdiagramm des Freescale i.MX536 (Quelle: [9])

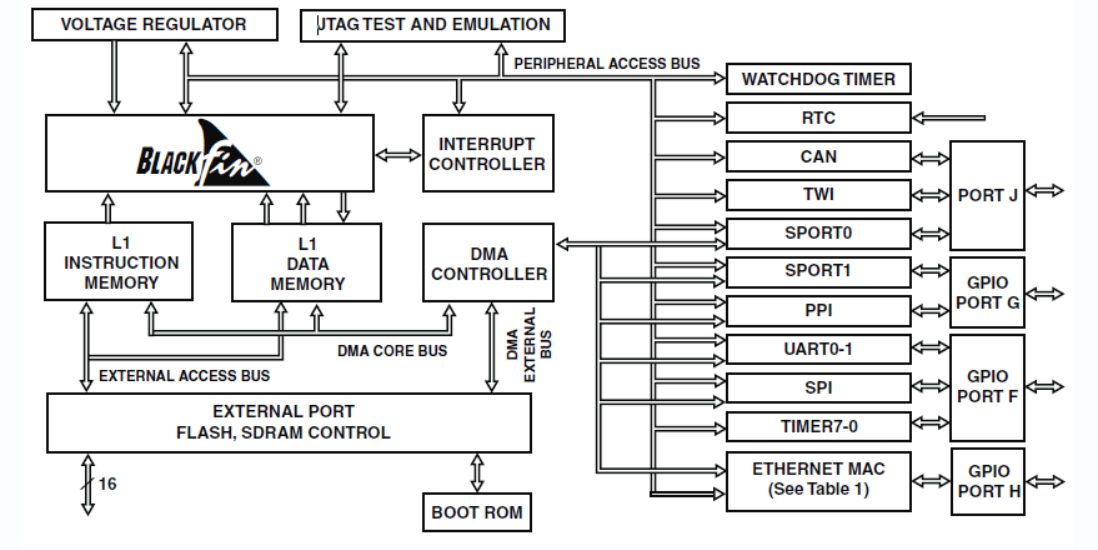


Abbildung 3.8: Blackfin Embedded Processors ADSP-BF537 (Quelle: [30])

Texas Instrumens Digital Media Processor DM3730:

In Abbildung 3.9 ist das Blockschaltbild des DM3730 (vgl. [15])zu sehen. In diesem Design wird ein ARM Cortex A8 1GHz Mikroprozessor, zusammen mit einer von Texas Instruments (TI) entwickelten Digitalen Signal Prozessor TMS 320DM64x+ verwendet. Für den Kameraanschluss bietet das Board ein Kamera Sensor Interface und USB 2.0. Wie im Blockschaltbild (Abbildung 3.10) zu erkennen ist, bietet der DM3730 keine Ethernet-

Schnittstelle an. Das Evaluationsboard von TI (TMDSEVM3730) bietet dafür eine Wireless LAN Erweiterungskarte an, welche für die Kameraplattform aber nicht geeignet ist.
Zudem hat der DM3730 einen Grafik- Beschleuniger für 2D (OpenVG1.0)- und 3D (OpenGLES 1.1) Grafik API's, sodass diese Plattform auch als Infotainmentsystem eingesetzt werden kann. Ein Anschluss für analoge oder digitale Displays ist ebenfalls vorhanden. Der Kostenpunkt für das Evaluationsboard von TI liegt bei 1495 $.
Eine Besonderheit ist, dass Texas Instruments das Hardware Design frei gegeben hat und es kostenlos heruntergeladen werden kann. Dadurch ist es möglich, das Board selbst herzustellen. Daraufhin ist das „Community Board" BeagleBoard entstanden, in seiner letzten Version, das BeagleBoard-xM. Produziert wird es von der Firma Circuitco. Dieses Board verfügt zusätzlich über eine Ethernet- Schnittstelle, die über den USB 2.0 Port realisiert wurde.
Dadurch ist dieses Board für den Einsatz als Kameraplattform geeignet. Die Kosten liegen statt 1500 $ bei 149 $. Freie Linux- Distributionen sind außerdem erhältlich (Angstrom, Ubuntu), zudem andere Betriebssysteme wie Android, Windows oder QNX. Da die erste Version des Beagleboards bereits im Jahre 2008 vorgestellt wurde, hat sich rund um das Board eine große Community gebildet. Nachfolger des Boards ist das Pandaboard und das BeagleBone, welche aber eine andere Hardware Architektur besitzen.

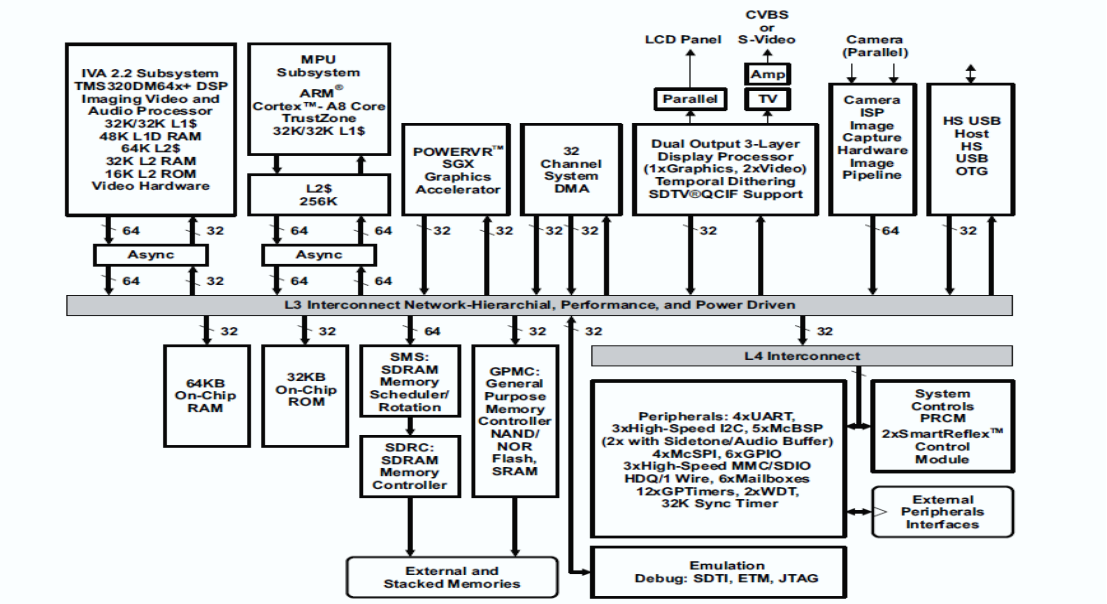


Abbildung 3.9: Blockdiagramm TI Digitaler Media Prozessor DM3730 (Quelle: [15])

3.4.1 Vergleich der vorgestellten Kameraplattformen

Anbieter	Board	1	2	3	4	5	6	7
Circuitco	BeagleBoard-xM	x	xx	x	xx	xx	xx	x
Freescale	i.MX53QSB	xx	x	x	xx	x	x	x
Freescale	MPC5604E EVB	x	x	xx	x			
Analog Devices	BF537 EZ-KIT	x	x	x	x	x		x

Tabelle 3.2: Erfüllte Anforderungen der Kameraplattform im Vergleich

Ein Vergleich der Hardware nach den in Tabelle 3.1 gestellten Anforderungen:
Ist eine Anforderung erfüllt, wird diese markiert (x), trifft eine Anforderung besonders zu, wird diese doppelt markiert (xx). Tabelle 3.2 zeigt, dass sowohl das BeagleBoard-xM, als auch das IMX53QSB alle Anforderungen erfüllt. Beim Vergleich der Blockdiagramme (Abbildung 3.7 und Abbildung 3.9), fällt auch eine Ähnlichkeit der zur Verfügung stehenden Funktionen auf.
Die Wahl fällt auf das in Abbildung 3.10 dargestellte Beagleboard-xM, da dieses eine größere Flexibilität für künftige Projekte in der CoRE- Gruppe bietet und eine große Community hinter sich hat. Außerdem besitzt das Freescale i.MX53QSB Board- bzw. der i.MX536 Prozessor eine Video Processing Unit (VPU), die auf Encoding spezialisiert ist. Aus dem Reference Manual des i.MX536 ist nicht zu entnehmen, welche Hardware sich hinter der VPU verbirgt. Dadurch ist die Entwicklung weiterer Imageprocessing- Aufgaben eingeschränkt. Bei dem Beagleboard-xM ist klar dokumentiert, dass es sich um dem TMS320C64+ Digitalen Signal Processor handelt. Dadurch ist eine flexiblere Entwicklung möglich, im Gegensatz zum Entwicklungsboard von Freescale.

Abbildung 3.10: Das „Community Board“ Beagleboard- xM (Quelle: [4])

3.4.2 Kamera und Objektiv:

Als Kamera wurde ein Kamera- Board der Firma Leopard Inc. ausgesucht. Dabei handelt es sich um das LI-5M03CS Modul (5 Mega Pixel). Kern des Moduls ist ein Aptina 1/2.5- inch CMOS Sensor (MT9P031- vgl. [12]). Das Modul kann über einen Kameraheader Anschluss mit dem Beagleboard- xM verbunden werden. Aptina bietet darüber hinaus, Treiber für den Linux- Kernel 2.6.x an. Die Verarbeitung eines 5MP Formats (2592 Horizontal- Pixel * 1944 Vertikal- Pixel), lässt eine Framerate von 14 Bildern pro Sekunde zu, dies ist zwar für eine flüssige Bilddarstellung ausreichend, jedoch kann der Encoder dieses Format nicht in Echtzeit verarbeiten.
Das Ausgabeformat des Sensors muss über das in Abbildung 3.11 dargestellte 4- fach Binning geändert werden. Dabei werden benachbarte Pixel (jeweils 2 Horizontal und Vertikal) zusammengefasst, dies führt zu einer Auflösung von 640*480 Pixel und entspricht den Anforderungen an die Kameraplattform- die Bilder können in Echtzeit kodiert werden. Darüber hinaus wird durch dieses Verfahren die Bildqualität verbessert, da eine höhere Lichtempfindlichkeit pro Pixel erreicht wird. Mit dem Einsatz dieses Verfahrens, erreicht das Kamera- Modul bzw. der Sensor eine Framerate von bis zu 53 Bildern pro Sekunde (vgl. [12]). Weiterhin ist für dieses Modul ein sog. „Fischaugenobjektiv“- ein Weitwinkelobjektiv ausgesucht worden. Dabei handelt es sich um das MPL1.55 1/2- inch CS von der Firma Arecont Vision. Durch die extrem kurze Brennweite[7] von 1.55 mm erreicht dieses Objektiv einen Bildausschnitt von ~180^{o} Grad. Dadurch entsteht je nach Qualität des Objektivs eine starke Verzeichnung[8] des Bildes.
Ein Objektiv kann auch so konstruiert werden, dass die Verzeichnung minimiert wird. Andernfalls muss die Verzeichnung mit mathematischer Berechnung heraus gerechnet werden. Dies wird in dieser Arbeit aber nicht behandelt.

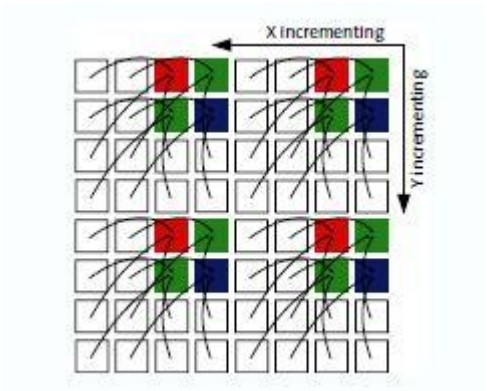


Abbildung 3.11: 4- fach Binning (Quelle: [12])

[7]Abstand zwischen dem optischen Mittelpunkt eines Objektivs und dem Brennpunkt, kleine Brennweiten stehen für weitwinklige Objektive (vgl. www.vision-doctor.de)
[8]Ein geometrischer Abbildungsfehler, der zu einer lokalen Veränderung des Abbildungsmaßstabes führt. Dieser nimmt von der Bildmitte zum Bildrand zu, bei Weitwinkelobjektives besonders ausgeprägt (vgl. www.wikipedia.de)

Nachdem die Hardware für die Kameraplattform ausgesucht ist und verschiedene Anbindungsmöglichkeiten diskutiert wurden, ergibt sich der in Abbildung 3.12 dargestellte Aufbau des Kamerasystems. Das Beagleboard-xM dient als Kameraplattform und encodiert das Kamerabild. Der Mikrocontroller- basierende TTEthernet- Protokollstack ist der Zugang zum TTEthernet und wird mit dem Beagleboard über die freie, zweite Ethernet-Schnittstelle verbunden.

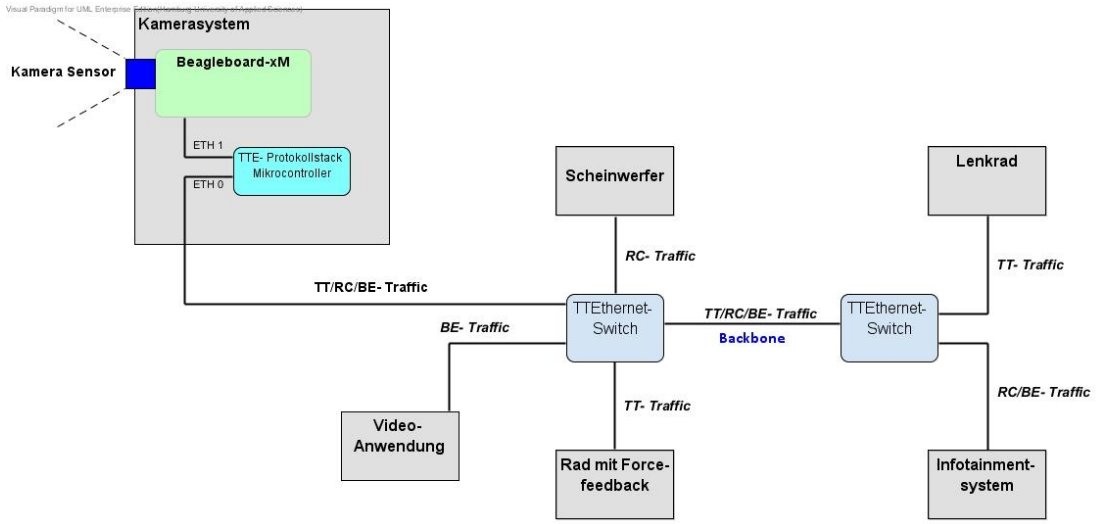


Abbildung 3.12: Übersicht Kamerasystem und Anbindung an das TTEthernet

3.5 Generelle Überlegungen und Anforderungen:

Im Infotainmentsystem soll das Rückfahrkamerabild angezeigt werden. Dafür muss das Videobild bzw. der Datenstrom vom Kamerasystem zum Infotainmentsystem übertagen werden. Dabei ist es möglich, die anfallenden Daten direkt mit einem Ethernet- Frame zu übertragen. Dies kann unter Linux mit dem sog. Raw- Socket realisiert werden. Ein Raw-Socket arbeitet jedoch höchstens auf der Internetschicht (vgl. [40]) des in Abbildung 3.13 dargestellten TCP/IP- Referenzmodells. Dadurch ist die Nutzung von anderen Anwendungen bzw. Netzwerkprotokollen, die auf höhere Schichten des TCP/IP- Referenzmodells angewiesen sind, nicht möglich.

Eine Anforderung an das Kamerasystem ist es, dass dieses im TTEthernet- Netzwerk erreichbar ist. Serverdienste oder Anwendungen müssen alle Netzwerkprotokolle benutzen können. Per TCP soll eine Client- Server Kommunikation zwischen Infotainment und Kamerasystem möglich sein, dies ist mit Raw- Sockets nicht möglich, da ein TCP- Socket auf das TCP- Protokoll angewiesen ist, welches der Transportschicht angehört. Netzwerkprotokolle wie FTP sollen zum Uploaden von Programmcode- bzw. neuen Anwendungen auf die Kameraplattform genutzt werden. Per SSH soll das „einloggen“ auf der Kameraplattform möglich sein.

Diese Protokolle liegen in der Anwendungsschicht und sind auf alle darunterliegenden Schichten angewiesen. Deswegen wird zum einen auf die Kommunikation mit Raw- Sockets verzichtet und stattdessen mit TCP- Sockets gearbeitet. Der Mikrocontroller basierte TTE-Protokollstack muss außerdem transparent implementiert werden. Er darf keine Auswirkungen auf die Erreichbarkeit der Kameraplattform im TTEthernet haben. Wie bereits erwähnt, soll die Auflösung der Kamera bei 640 Breite * 480 Höhe liegen, die Bildrate soll für eine flüssige Darstellung bei mindestens 14 Bilder pro Sekunde liegen. Der Kamerastream soll über das Infotainmentsystem steuerbar sein und als Critical- Traffic übertragen werden. Die Encodierung soll mittels dem h.264 Codec realisiert werden, damit möglichst wenig Bandbreite belegt wird.

Abbildung 3.13: TCP/IP- Referenzmodell (vgl. [23])

3.6 Software Überlegungen und Konzept:

Im folgenden Abschnitt wird analysiert und diskutiert, welche Software Erweiterungen und Neuentwicklungen realisiert werden müssen, damit das encodierte Live- Videobild der Kameraplattform im Infotainmentsystem angezeigt werden kann und die zuvor genannten generellen Anforderungen erfüllt werden. Zuerst werden die notwendigen Erweiterungen für den TTE- Protokollstack, dann für das Infotainmentsystem und zuletzt für die Neuentwicklung der Kameraplattform festgestellt.

3.6.1 Erweiterung auf dem Mikrokontroller basierenden TTE-Protokollstack

Das Senden und Empfangen von Nachrichten über die freie Ethernet- Schnittstelle muss implementiert werden. Für das Empfangen muss eine Interrupt Service Routine geschrieben werden, die Nachrichten von der Kameraplattform annimmt und über das TTEthernet API in den dafür konfigurierten Output- Buffer kopiert. Trifft eine Nachricht über das TTEthernet ein, wird diese in einen Input- Buffer geschrieben. An einen Input-Buffer wird über das TTEthernet API eine Callback- Funktion angemeldet, die beim Eintreffen einer Nachricht ausgeführt wird. Diese Callback- Funktion muss implementiert werden. Weiterhin muss der Schedule für die RC/TT- Nachrichten in einer Konfigurationsdatei vorgenommen werden. Die Implementierung ähnelt demnach der Funktion einer Bridge. Weitere Funktionen müssen nicht implementiert werden. Nachrichten, die von der Kameraplattform kommen, werden angenommen, TTEthernet-konform verpackt und je nach Schedule weitergeleitet. Nachrichten vom Infotainmentsystem werden über die Callback- Funktion 1:1 weitergeleitet- dadurch wird Transparenz erreicht.
Beim TTEthernet- konformen Verpacken der Nachrichten, wird der Header des ankommenden Ethernet- Frames von der Kameraplattform geändert (vgl. Abbildung 2.4). Somit entspricht der Ethernet- Frameheader nicht mehr 1:1 dem Original. Soll eine TCP-Verbindung zwischen dem Infotainmentsystem und dem Kamerasystem aufgebaut werden, schlägt dies fehl. Die Netzzugangschicht bzw. die Netzwerkkarte des Infotainmentsystems wird das Ethernet- Frame nicht annehmen bzw. an höhere Protokollschichten weiterleiten, da die MAC- Adresse nicht mit seiner eigenen übereinstimmt. Deswegen muss die Netzwerkkarte des Infotainmentsystems im sog. promiscuous- Mode betrieben werden. Damit nimmt die Netzwerkkarte jedes Ethernet- Frame an, unabhängig der Destination MAC- Adresse.

3.6.2 Erweiterung Infotainmentsystem

Abbildung 3.14 zeigt eine Klassenübersicht über die existierenden Klassen. Programmiert wurde das Infotainmentsystem von der CoRE- Gruppe, in C++ mit dem QT Framework, dieses ist besonders für die Erstellung von grafischen Benutzeroberflächen (GUI) bekannt. Es folgt eine kurze Beschreibung der Klassen.

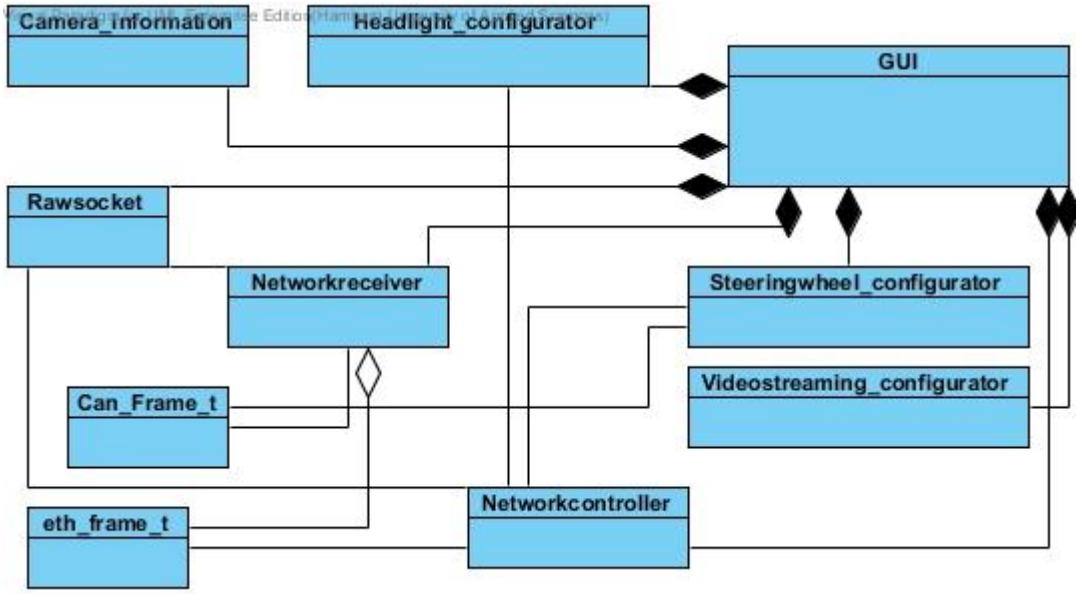


Abbildung 3.14: Klassenübersicht InfotainmentsystemCamera_Information:

Realisierung der aktuellen Rückfahrkamera. Empfängt einen BE- Nachrichtenstream von einer Webcam und stellt das Bild in der grafischen Benutzeroberfläche dar.

Headlight_configurator:
Über diese Klasse wird die Lichtsteuerung realisiert. Soll z.B. das Warnlicht des Demonstrators angeschaltet werden, wird die entsprechende Nachricht als Ethernet- Frame verpackt und an den Networkcontroller zum Versenden weitergegeben.

Steeringwheel_configurator:
Über diese Klasse können Parameter der Steer- By- Wire Lenkung geändert werden. Um ein Beispiel zu nennen, der Force- Feedback- Effekt soll stärker werden, dazu wird eine CAN- Nachricht erstellt. Diese CAN Nachricht- wird in das Datenfeld eines Ethernet- Frames verpackt und über den Networkcontroller verschickt.

Videostreaming_configurator:
Streamt ein Video als BE- Traffic an einen Laptop im Netzwerk und zeigt dieses Video selbst im Fenster der grafischen Oberfläche (GUI) an.

Rawsocket:
Erstellt einen Rawsocket (auf Protokolllayer- 3 des TCP/IP- Referenzmodells vgl. Abbildung 3.14) zum Senden und Empfangen von Ethernet- Frames.

Can_Frame_t:
Datenklasse für einen CAN- Frame

Eth_Frame_t:
Datenklasse für einen Ethernet- Frame

Networkcontroller:
Sendet einen Ethernet- Frame über einen Rawsocket

Networkreceiver:
Empfängt Ethernet- Frames über einen Rawsocket und informiert das GUI, dass Statusänderung der Steer- By- Wire- Lenkung oder der Headlights stattgefunden haben. Dadurch kann das GUI aktualisiert werden.

Geplante Erweiterungen:

Wie in Abbildung 3.15 zu sehen, wird das Infotainmentsystem um folgende Klassen erweitert.

GPlayer:
Die Camera_informationsklasse wird entfernt und eine neue GPlayer- Klasse wird erstellt. Diese neue Klasse ist für den Empfang, das decodieren und der Anzeige des Videostreams auf der GUI zuständig.

Tcp_communication:
Um eine Client- Server Kommunikation zwischen der Kameraplattform und dem Infotainmentsystem zu realisieren, wird das Infotainmentsystem um eine weitere Klasse erweitert. Mit dieser Klasse kann eine TCP Verbindung mit der Kameraplattform hergestellt werden. Funktionen zum Verbinden und zur Steuerung der Kameraplattform sollen implementiert werden.

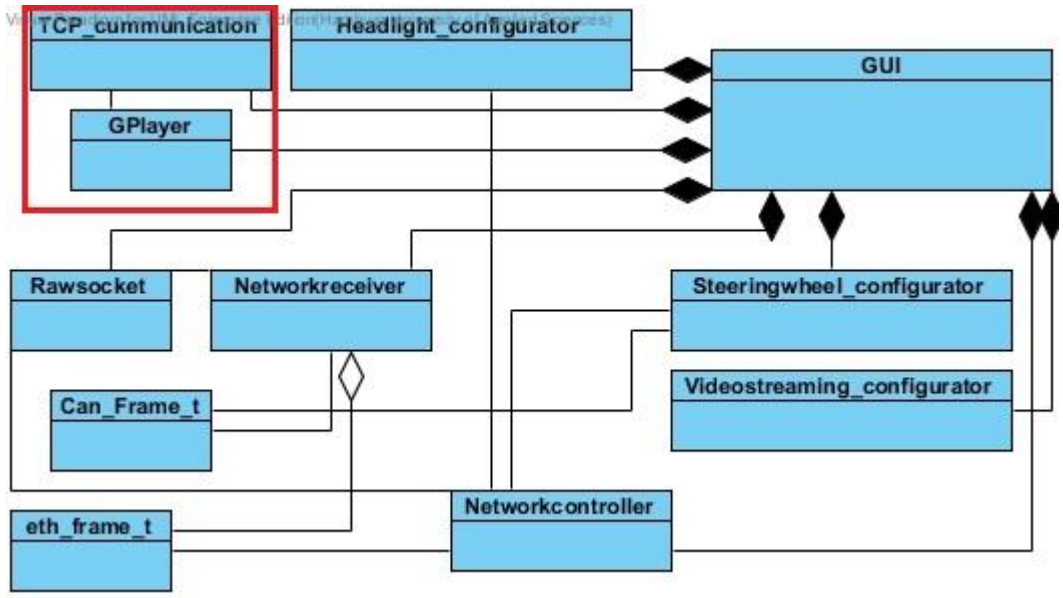


Abbildung 3.15: erweiterte Klassenübersicht Infotainmentsystem

Diskussion Client- Server Kommunikation per TCP:

Das Infotainmentsystem besitzt zwei Netzwerk- Schnittstellen. Für die Kommunikation mit dem TTEthernet, ist das Interface 0 zuständig. Mit Hilfe von Rawsockets können über dieses Interface RC- Nachrichten über die in Abbildung 3.15 dargestellte Netzwerkcontroller-Klasse verschickt werden. Die geplante Kommunikation mit der Kameraplattform findet nicht über diese Klasse statt. Da, wie bereits erwähnt, höhere Schichten des Netzwerkprotokolls bzw. des TCP/IP- Referenzmodells für die TCP Kommunikation und sonstige auf TCP/UPD aufbauende Anwendungen gebraucht werden. Dieses führt dazu, dass keine bidirektionale TTEthernet- Verbindung zwischen Infotainmentsystem und Kamerasystem, wie in Abbildung 3.16 dargestellt besteht.

Der Nachrichtenverkehr vom Kamerasystem zum Infotainmentsystem erfolgt per RC- bzw. TT- Nachrichten. Der Nachrichtenverkehr vom Infotainmentsystem kann demnach nur als BE- Verkehr behandelt werden, somit ist die Übertragung der Nachricht nicht garantiert. Soll die Rückfahrkamera über die GUI des Infotainmentsystems gestartet werden, kann diese BE- Nachricht im „Worst Case Szenario“ verloren gehen.

Durch den BE- Nachrichtenverkehr kann zwar eine Nachricht verloren gehen, dies wird aber durch die TCP- Verbindung erkannt. Das Infotainmentsystem schickt im Falle eines Nachrichtenverlusts solange eine neue Nachricht, bis der Empfang von der Kameraplattform bestätigt wird. Dadurch wird der Kamerastream gestartet, jedoch durch die Übertragung einer oder mehreren, weiteren Nachrichten verzögert. Beachtet man die menschlichen Reaktionszeiten, sollte die Verzögerung jedoch nicht bemerkbar sein. Es sei denn, der Backbone ist über einen längeren Zeitraum zu 100% von RC, oder TT Nachrichten ausgelastet, sodass keine BE- Nachrichten übertragen werden können.

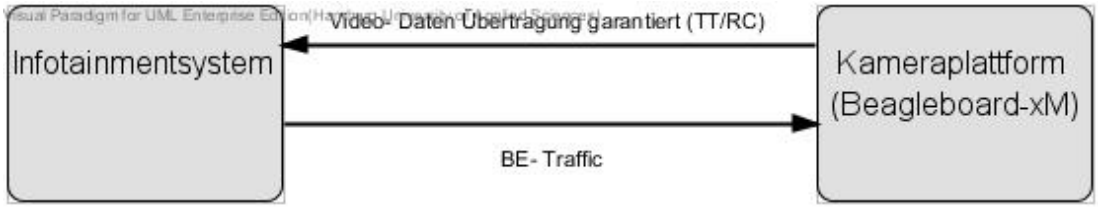


Abbildung 3.16: Nachrichtenverkehrs zwischen Kamerasystem und Infotainmentsystem

3.6.3 Neuentwicklung Kameraplattform

Hauptaufgabe der Kameraplattform ist das Encodieren des Kamerabildes und die Übertragung der Kameradaten über das Netzwerk (streaming). Zur Kommunikation mit dem Infotainmentsystem muss ein TCP- Server entwickelt werden, der eine Verbindung des Infotainmentsystems annehmen kann. Anfragen wie das Starten oder Stoppen der Kamera werden verarbeitet und stoßen das Encodieren und Streaming der Videodaten an. Abbildung 3.17 zeigt eine Übersicht der zu entwickelnden Klassen.
Die Klasse TCP- Server, soll die Kommunikation mit dem Infotainmentsystem realisieren. Aufgabe der Klasse Gst- Server ist das Parametrisieren der Kamera, dass Encodieren des Videosignals (h264 Codec) und dessen Streaming über das Netzwerk- zum Infotainmentsystem. Zur Übertragung des Videostreams wird das verbindungslose Transportprotokoll UDP benutzt.
Da die Kameradaten vom Kameraboard zum Infotainmentsystem als RC- Nachrichten übertragen werden und das TCP- Protokoll mehr Overhead durch Flusskontrolle, Fehlerkontrolle oder durch die erneute Übertragung wegen eines beschädigten oder verlorengegangenes Paketes besitzt (vgl. [36]), ist diese Übertragungsart nicht notwendig und eher ungeeignet für das Streaming. Im Kontext eines Decoders ist eine zu spät eintreffende Nachricht nicht mehr relevant, da schon das nächste Bild dekodiert werden muss. Die Deadline wurde durch die zu spät eintreffende Nachricht nicht eingehalten und das Paket wird verworfen. Im Gegensatz zu einer TCP- Verbindung wird vor der Datenübertragung auch keine Verbindung aufgebaut. Bekommt die Gst- Server- Klasse den Auftrag den Stream zu starteten, so wird sofort mit dem Streaming begonnen.
Eine Einschränkung gibt es jedoch, der Gst- Server muss die MAC- Adresse der Infotainmentanlage kennen, andernfalls muss zuvor durch ein ARP- Request die Zuordnung der IP-Adresse zur MAC- Adresse stattfinden. Die Zuordnung der IP-Adresse zu einer MAC-Adresse wird in einer Tabelle bzw. Konfigurationsdatei abgespeichert, diese kann editiert werden (vgl. [36]).
Da aber vor dem Start des Streamings ein Verbindungsaufbau durch die TCP-Kommunikation stattgefunden hat, sollte diese Zuordnung der IP- Adressen und MAC-Adressen bereits stattgefunden haben. Weiterhin besteht die Möglichkeit, dass der Tabelleneintrag nach einiger Zeit (Timeout) gelöscht wird. Dieser Mechanismus sorgt dafür, dass wenn sich zu einer IP- Adresse die MAC Adresse ändert (z.B. durch den Austausch der Netzwerkkarte, oder der Benutzung eines anderen Interfaces) eine neue Zuordnung

stattfinden kann. Durch die TCP- Verbindung und dem Streaming sollte dieser Timeout nicht stattfinden. Dies bleibt es noch bei der Realisierung zu untersuchen.

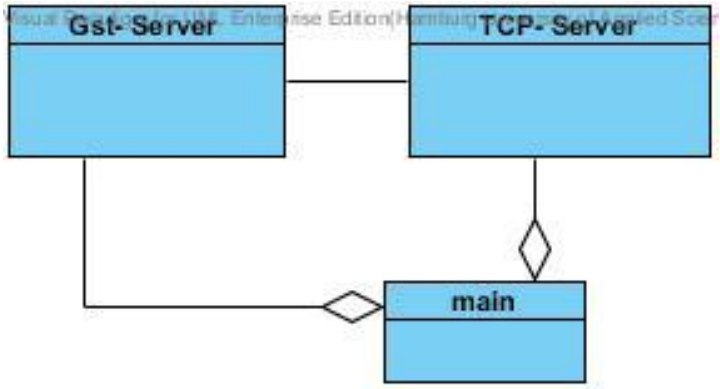


Abbildung 3.17: Klassenübersicht Kameraplattform

4 Realisierung und Implementierung

Im folgenden Kapitel wird die Umsetzung des Entwurfs aus dem vorherigen Kapitel vorgestellt. Es werden die Erweiterungen und Neuentwicklungen im Mikrocontroller basierten TTE- Protokollstack, des Infotainmentsystems und der Kameraplattform vorgestellt. Damit die Kameraplattform im Netzwerk erreichbar ist, muss außerdem eine Konfiguration der TTEthernet- Switche erfolgen, dies wird im letzten Abschnitt vorgestellt.

4.1 Mikrocontroller basierter TTE- Protokollstack

Für die Realisierung steht zum einen das TTEthernet- API [38] zur Verfügung. Über dieses Interface wird auf die erstellten Buffer (TT, RC- oder BE) zugegriffen (vgl. Abbildung 3.5 aus vorherigem Kapitel), desweiteren wird über dieses Interface eine Callback- Funktion an einen bestimmten Buffer angemeldet. Dieser Mechanismus sorgt dafür, dass bei einer eintreffenden Nachricht, die angemeldete Funktion aufgerufen wird. Zudem steht das API des Hardware Abstraction Layers (HAL) zur Verfügung (vgl. [8]). Über dieses Interface stehen unter anderen Funktionen zum Senden und Empfangen von Ethernet- Frames zur Verfügung. In der Konfigurationsdatei wird das Schedule für die RC- Nachrichten- bzw. dem BAG- Value, sowie die Konfiguration der Buffer vorgenommen. Im folgenden Abschnitt werden alle Erweiterungen vorgestellt, damit der Mikorcontroller als „Bridge“ im TTEthernet arbeitet. Programmiersprache ist C.

4.1.1 Interrupt Service Routine

Wird ein sog. Interrupt Request (IRQ) auf der Ethernet- Schnittstelle (Port1) ausgelöst, wird die Interrupt Service Routine (ISR) *ethernet_IRQ_Handler_Port1* aufgerufen. Dieses Ereignis tritt bei einem Nachrichtenempfang von der Kameraplattform auf. Über einen Interrupt Vector Controller wird diesem IRQ eine Priorität zugeordnet. Dabei ist die Priorität so gewählt, dass diese niedriger als das Senden einer TT- Nachricht ist und höher als eine Callback- Task. Damit kann eine BE- Nachricht vom Infotainmentsystem nicht den Empfang einer Nachricht von der Kameraplattform unterbrechen. Nachdem der IRQ ausgelöst wurde, wird in der aufgerufenen Interrupt Service Routine (ISR) überprüft, ob der IRQ am Ethernet- Port1 anliegt und bestätigt. Danach wird der in Abbildung 4.1 dargestellte Ablauf in der ISR durchlaufen. Im Fehlerfall wird über die RS232 Schnittstelle eine Fehlermeldung

ausgegeben (Terminal- Ausgabe). Die Terminal- Funktionen sind bereits implementiert und dienen als Debugging Ausgaben. Über einfache Ausgaben hinaus, ist auch das Auslesen von definierten Speicherabbildern oder gesamten Ethernet- Frames möglich (vgl. [24]).

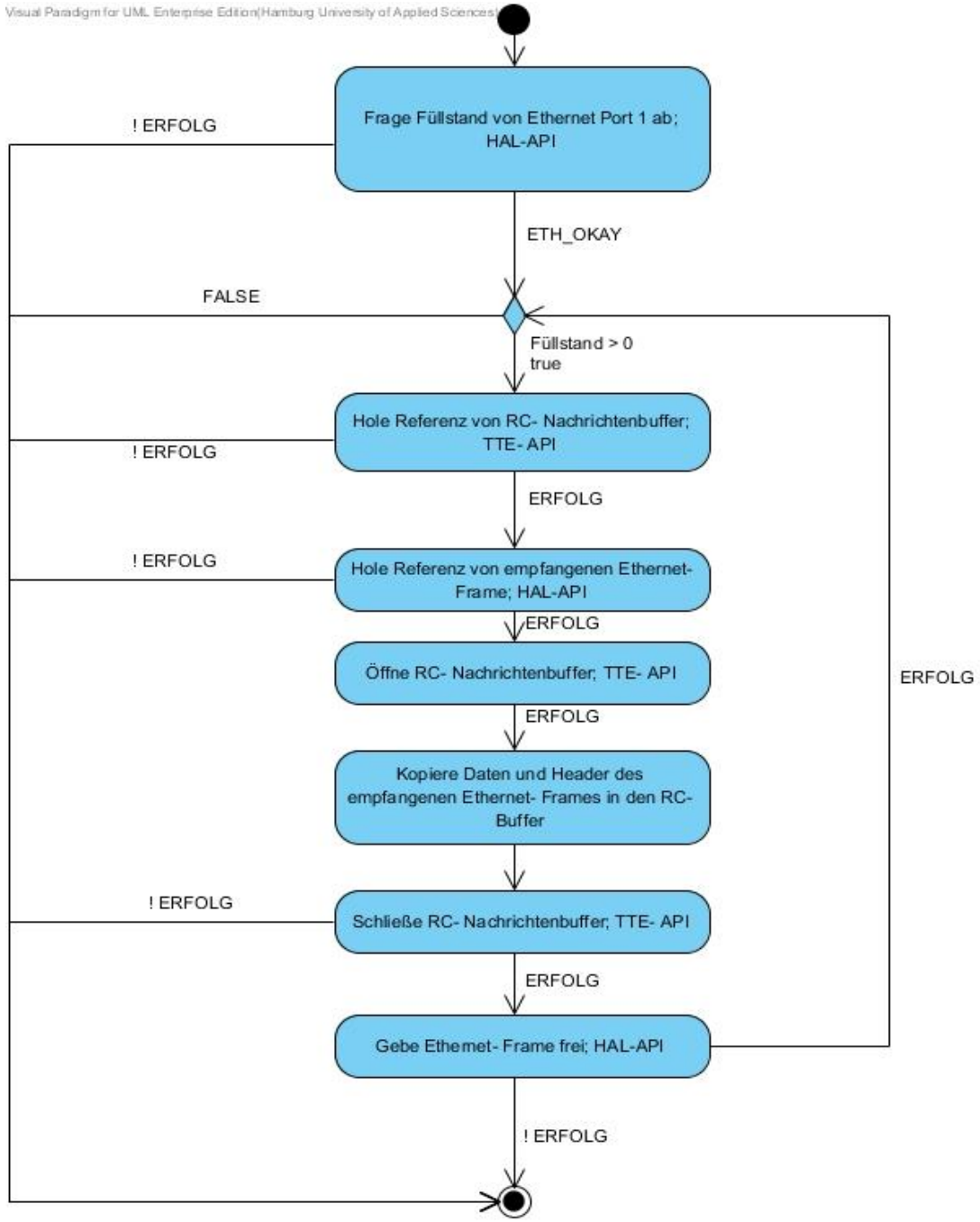


Abbildung 4.1: ISR beim Empfang einer Nachricht von der Kameraplattform

4.1.2 Callback- Funktion

Für BE- Nachrichten, die vom TTEthernet empfangen werden, wird beim Systemstart ein Buffer zur Verfügung gestellt, dieser muss daher nicht in der Konfigurationsdatei konfiguriert werden. Jedem Nachrichtenbuffer (auch RC- oder TT- Nachrichten) kann über die TTE- API eine Callback- Funktion angemeldet werden. Dies geschieht mit folgender Funktion, zur Vereinfachung der Darstellung wurde auf die Datentypen verzichtet:

Return value	Name	Parameter 1	Parameter 2	Parameter 3	Parameter 4
Error code	tte_set_buf_var	buffer*	var_id	var_size	Value*

Tabelle 4.1: Funktion zum Anmelden einer Callback- Funktion an einen Buffer

Bei der Initialisierung des Systems wird in Tabelle 4.1 dargestellte Funktion aufgerufen. Über die TTE- API Funktion *tte_get_bg_input_buf* wird die Referenz vom BE- Buffer (Port0) abgerufen, diese ist der erste Parameter der Funktion. Beim zweiten Parameter wird die Variable Identifikationsnummer übergeben, damit wird die Aufgabe der Funktion spezifiziert. In diesem Fall soll ein „receive Callback“ realisiert werden. Über diese Funktion lässt sich unter anderem auch ein „transmit Callback“ realisieren. Beim dritten Parameter wird die Größe eines Funktionspointers übergeben und beim vierten die Adresse der Callback- Funktion. Damit ist die Initialisierung der Callback- Funktion abgeschlossen. Trifft eine Nachricht vom TTEthernet, bzw. vom Infotainmentsystem ein, wird die in Abbildung 4.2 dargestellte Callback- Funktion aufgerufen.

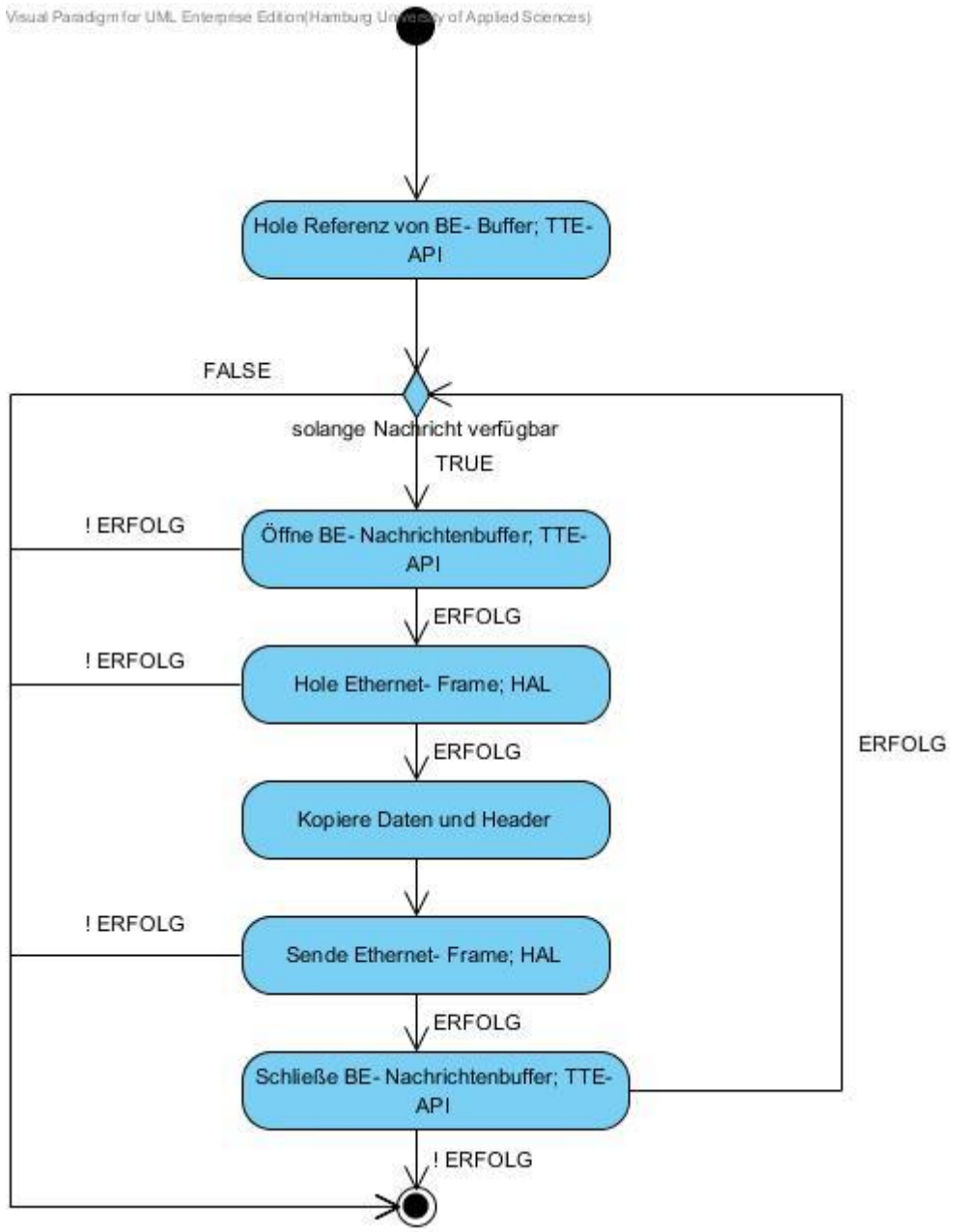


Abbildung 4.2: Ablauf Callback- Funktion beim Empfang einer Nachricht vom TTEthernet

4.1.3 Bufferkonfiguration

Der TTE- Protokollstack soll synchron im TTEthernet eingebunden werden. Dazu muss er Synchronisationsnachrichten empfangen. Dazu wird ein Empfangs- Buffer definiert. Der TTE- Protokollstack unterstützt Double- Buffer und Queue Buffer, es können aber auch eigene Buffertypen implementiert werden. Dies ist für diese Anwendung jedoch nicht nötig. Für den Empfang der Synchronisationsnachrichten wird ein Double- Buffer verwendet. Trifft eine neue Synchronisationsnachricht ein, wird die alte überschrieben. Double- Buffer sind performant, da zwei Kopien einer Nachricht verwaltet werden. Lesende Zugriffe arbeiten auf einer Kopie, sobald eine neue Nachricht in den Speicher geschrieben wurde, wird die Lesekopie durch diese ersetzt. Dadurch greifen Lese- und Schreibzugriffe nicht ständig auf demselben Speicherbereich zu.
Weiterhin wird ein Sende- Buffer definiert. In diesen Buffer werden die Nachrichten von der Kameraplattform gespeichert. Nachrichten müssen gepuffert werden, dazu wird der Sende-Buffer als Queue- definiert. Wird die Kapazität des Buffers beim Schreiben überschritten, wird die älteste Nachricht, auf der kein Lesezugriff stattfindet überschrieben (vgl. [24]). Bei dem Queue Buffer handelt es sich um eine FIFO Datenstuktur. Wird ein Element (Nachricht) gelesen, so wird dieses „konsumiert“ und der Speicherbereich freigegeben.
Es bietet sich an, die Größe auf 20 Frames zu begrenzen, dies entspricht der Kapazität einer Ethernet- Schnittstelle (vgl. [24]). Dieses führt jedoch dazu, dass ein Nachrichtenverlust durch eine Überlastung des Buffers nicht erkannt wird, da wie bereits erwähnt, die älteste Nachricht überschrieben wird. Darum wird eine Kapazität > 20 gewählt, dadurch wird ein Bufferüberlauf in der ISR erkannt. Der Ethernet- Controller meldet einen Error-Code, dass keine weitere Nachricht verarbeitet werden kann. Diese Situation muss zwingend verhindert werden. Zum einen durch die Wahl des Schedule, hier wird festgelegt, wie schnell die Nachrichten konsumiert werden und zum anderen durch die Kameraplattform. Diese darf nicht mehr Nachrichten senden, als der Schedule verarbeiten kann.
Bei jeder Buffer- Konfiguration wird Typ (Queue), Länge eines Buffers, die sog. Message- ID (auch Critical ID genannt), Destination MAC- Adresse (TT- Adresse) festgelegt. Der Nachrichtenverkehr von der Kameraplattform wird über eine neue Message- ID festgelegt (0x450). Ob diese Message als TT- oder RC- Nachricht behandelt wird entscheidet das TTEthernet Switch. In der Konfiguration der Switche (vgl. 4.4) wird definiert, ob die Nachricht- ID als RC- oder TT- Nachricht behandelt wird.
Die Größe für das Datenfeld einer Nachricht wird auf das Maximum von 1500 Bytes gesetzt. Damit kann jede Nachricht von der Kameraplattform weitergeleitet werden.

4.1.4 Schedule

Der Mikrokontroller basierte TTE- Protokollstack besitzt keinen Mechanismus für die Behandlung von RC- Nachrichten, diese werden wie TT- Nachrichten im Schedule definiert. Als Basiskonfiguration wird ein kleinstmögliches Schedule gewählt. Der Zyklus wird auf 5ms eingestellt, dies entspricht der aktuellen Zykluseinstellung im gesamten TTEthernet des Demonstrators. Somit wird ein Schedule für 5ms in einer Tabelle eingetragen. Für die Übertragung eines maximalen Ethernet- Frames (1518 Bytes- IEEE 802.3) in einem 100Mbit/s Netzwerk braucht der Controller 121,44 µs. Die ersten 100 µs werden außerdem für den Empfang von einer Synchronisationsnachricht freigehalten. Weitere Einträge liegen im 125µs Abstand, sodass ein maximaler Ethernet- Frame übertragen werden kann. Damit sind 39 Schedule- Einträge für die Datenübertragung von der Kameraplattform zum Infotainmentsystem eingetragen. Daraus ergibt sich, dass unabhängig der Größe des Ethernet- Frames 39 Pakete in 5ms übertragen werden.

4.2 Realisierung Kameraplattform

Im folgenden Abschnitt wird die Umsetzung der Kameraplattform beschrieben und die Probleme die dabei entstanden sind. Zuerst werden die Vorbereitungen für die Programmierung vorgestellt, welches Betriebssystem eingesetzt wird und welche Bibliothek bzw. Frameworks eingesetzt werden. Danach folgen Probleme mit der Kamera und der TCP-Verbindung. Anschließend wird die Implementierung detailliert betrachtet.

4.2.1 Vorbereitung

Als Betriebssystem steht unter anderem die Linux Distribution Angstrom zur Verfügung. Angstrom wurde speziell für eingebettete Systeme konzipiert, läuft auf Geräten mit 4MB Flashspeicher, sowie Smartphones, Tablets und diversen anderen Plattformen. Damit Angstrom auf der Kameraplattform läuft, wird eine microSD- Karte benötigt mit zwei Partitionen. Eine boot- Partition und eine für das Linux root- Filesystem. Images für die boot Partition (FAT) und das root Filesystem (EXT3) können heruntergeladen werden. Eine Crosscompile- Toolchain steht außerdem zur Verfügung. Angstrom selbst wurde mit dem build Framework für eingebettete Linux- Systeme erstellt (OpenEmbedded). Außerdem ist es möglich, über ein Online- Builder ein angepasstes Angstrom Image zu erstellen. Angstrom ist Paket basiert und bietet ein Paket- Managementsystem an. Dadurch ist eine leichte Erweiterung des Systems und Entwicklung möglich. Für diese Arbeit wurde über den Online Builder ein Angstrom Image erstellt. Die Linux Kernel Version ist 2.6.32. Die Programmiersprache ist C++.

Folgende Bibliotheken werden auf der Kameraplattform und der Crosscompile- Toolchain für die Programmierung der Anwendung benötigt, diese werden über das Paket-Managementsystem installiert:

- Gstreamer
  Gstreamer ist ein Multimedia- Framework, geschrieben in C und bietet Unterstützung für Audio und Video Anwendungen. Durch die Plugin Architektur ist das Framework erweiterbar.
- Boost
  Eine C++ Bibliothek unter anderem mit einem Socket API, basierend auf Linux Barkeley Sockets
- Glib-2.0
  Für Gstreamer notwendig, das API bietet unter anderem Funktionen für objektorientierte Programmierung in C

Eine Besonderheit des Gstreamer- Frameworks ist die Pipeline Architektur.
In Abbildung 4.3 ist die GstServer- Pipeline zu sehen, mehrere Elemente werden über src- und sink- Pads verbunden. Ein Element empfängt über seinen sink- Pad Daten und generiert Daten an seinem source- Pad. Die Elemente haben spezielle Funktionen. Das Video- src Element liest Daten der Videokamera ein und ist mit dem Encoder- Element verlinkt. Der Encoder encodiert das Videosignal h.264 konform, welches über das UDP- sink Element in das Netzwerk gestreamt wird.

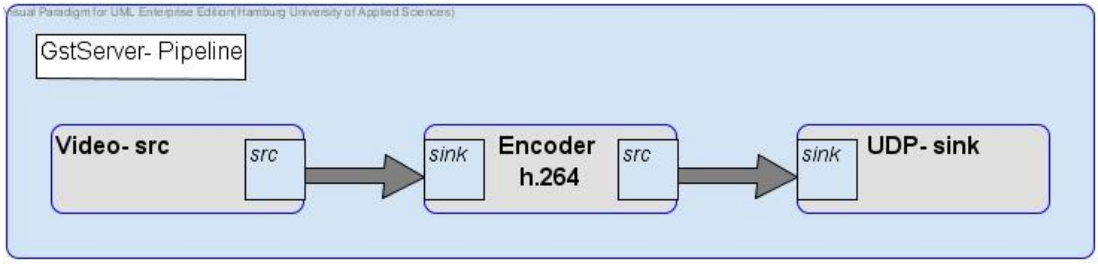


Abbildung 4.3: Pipeline der GstServer- Anwendung

Wie bereits erwähnt, ist das Gstreamer- Framework durch Plugins erweiterbar. Diese können dynamisch zur Laufzeit geladen werden. Texas Instruments bietet ein Plugin an, welches Hardware unterstütztes encodieren ermöglicht. In Abbildung 3.10 ist diese Hardware als IVA2.2 Subsystem (vgl. [13]) gekennzeichnet. Das Subsystem beinhaltet den Digitalen Signal Prozessor und weitere Video- Hardware- Beschleuniger.
Das GStreamer Plugin läuft wie in Abbildung 4.4 zu erkennen, auf dem ARM Prozessor. Der Codec- Server, auf der DSP Seite, dieser stellt Funktionen für das Encodieren bereit. Per Remote Procedure Call (RPC- Stub und Skeleton) greift das GStreamer Plugin auf diese zu. Damit die Daten zwischen dem ARM Prozessor und der DSP übertragen werden und der RPC realisiert werden kann, wird die DSP/BIOS Link Software benötigt (vgl. [14]). Für die

Interprozesskommunikation, wie dem RPC, stellt diese Software ein generisches API zur Verfügung (VISA- Video, Image, Speech and Audio), welches die physikalische Verbindung zwischen MCU (ARM Prozessor) und dem DSP abstrahiert. Zusätzlich bietet die DSP/BIOS Link Software einen Gerätetreiber für das IVA2.2 Subsystem an. Zur weiteren Kommunikation zwischen ARM Prozessor und DSP wird Shared Memory und Message Passing eingesetzt. Die Größe und Anfangsadresse des Shared Memorys wird in einer Konfigurationsdatei gespeichert, die auf der boot Partition liegt.

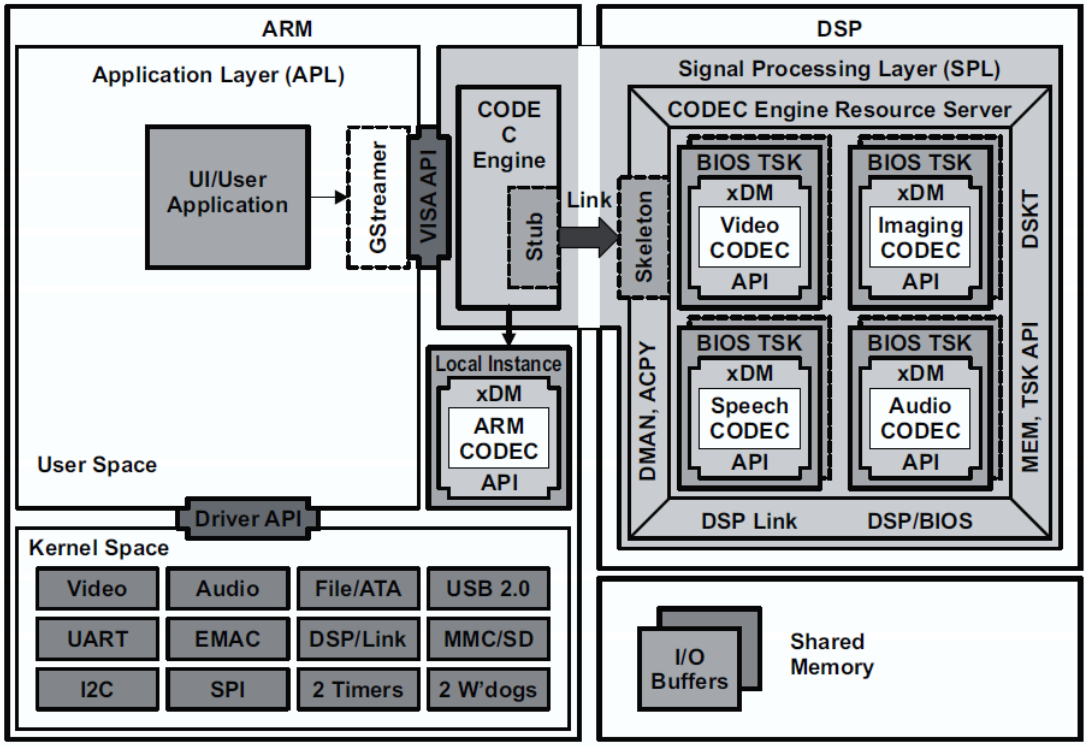


Abbildung 4.4: Software Struktur GStreamer- Digitaler Media Prozessor (Quelle: [16])

Kamera:

Die Kameraplattform besitzt einen digitalen und analogen Monitorausgang, zum Testen des Kamerabildes wird die MPlayer Anwendung genutzt. Es hat sich herausgestellt, dass das Kamerabild nicht korrekt ausgegeben wird. Das Format kann nicht korrekt, wie gewünscht, auf 640*480 Pixel eingestellt werden, der Treiber stammt von Aptina, für die Kernelversion 2.6.32. Der Treiber für den MT9P031 Kamerasensor wurde von einem Entwicklerteam überarbeitet und in den Mainline- Linux- Kernel ab der Version 3.2 übernommen. Die Treiberstruktur hat sich dabei grundsätzlich verändert und einige V4l2[9] API Funktionen sind nicht mehr implementiert. Darum kann der Treiber nicht mehr in der GStreamer Pipeline eingesetzt werden und es wird mit der Treiberversion für den 2.6.32 Kernel gearbeitet. Da der Treiber weiterentwickelt wird, ist in Zukunft mit der Implementierung weiterer V4l2 API

[9]Ein applications programming interface für video capture devices, integriert in den Mainline Linux Kernel.

Funktionen zu rechnen. Für die weitere Entwicklung wird daher nicht der komplette Bildbereich zur Verfügung stehen.

TCP- Verbindung:

Nachdem die Realisierung der Erweiterung für den Mikrokontroller basierten TTE-Protokollstacks abgeschlossen ist, hat sich beim Testen gezeigt, dass keine TCP- Verbindung aufgebaut werden kann. Die Manipulation des Ethernet- Frame- Header wird im TCP-Protokoll erkannt, dadurch wird die Nachricht nicht an die Anwendungsschicht bzw. die Anwendung weitergereicht. Damit ist keine Anwendung verfügbar, die eine TCP Verbindung nutzt. Die Stelle, an der das TCP- Protokoll den Ethernet- Header überprüft konnte nicht gefunden werden, im TCP- Protokoll findet sich lediglich die Überprüfung des IP- Headers, dieser wird aber nicht manipuliert. Der „Fehler“ ist deshalb nicht nachvollziehbar und kann ihm Rahmen dieser Arbeit nicht mehr gelöst werden. Die Lösung ist die Verwendung eines weiteren TTE- Protokollstacks vor dem Infotainmentsystem, der die ankommenden Nachrichten wieder an den originalen Ethernet- Header der Kameraplattform anpasst. Der Empfangen der Ethernet- Frames über Raw- Sockets und die anschließende Manipulation des Ethernet- Headers ist auch möglich, wenn der Raw- Socket die Protokollfunktionen der Transportschicht des TCP/IP- Referenzmodells implementieren (TCP/UDP- Protokoll).
Für die Kommunikation wird stattdessen UDP verwendet und ein eigener Algorithmus implementiert, der einen Paketverlust erkennt.

ARP- Request:

Es hat sich gezeigt, dass trotz ständigem UDP Datenverkehr ein erneuter ARP Request, stattfindet.

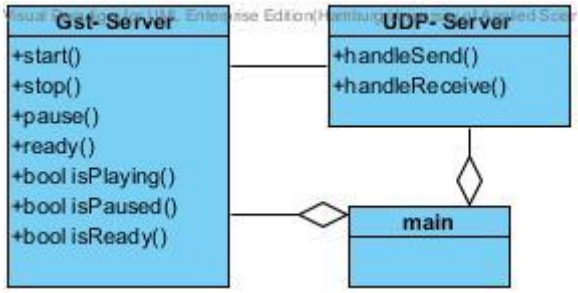


Abbildung 4.5: Aktualisierte Klassen der Kameraplattform- Anwendung

4.2.2 Implementierung GstServer Klasse

In Abbildung 4.5 ist die aktualisierte Klassenübersicht der Kameraplattform- Anwendung zu sehen. Kern der Anwendung ist die in Abbildung 4.3 dargestellte Pipeline.
Eine leere Pipeline und die jeweiligen Elemente werden erstellt. Alle Elemente werden dynamisch zur Laufzeit geladen. Die Kamera ist v4l2 kompatibel und wird als v4l2src Element genutzt. Wie bereits erwähnt, wird das Texas Instruments GStreamer Plugin als Encoder- Element eingesetzt und über das sink- Element als UDP Stream über das TTEthernet zum Infotainment gestreamt.

Alle Elemente besitzen Zustände (States):

- NULL
  Default State, keine Ressourcen werden allokiert, wird ein Element in diesen Zustand gesetzt, so werden alle allokierten Ressourcen freigegeben.
- READY
  Ressourcen werden für das Element allokiert, der Datenstream wird nicht gestartet, ein bereites laufender Datenstream wird unterbrochen und resettet.
- PAUSED
  Läuft ein Datenstream wird er nicht aktiv vom Element bearbeitet (Kein Clock Signal aktiv)
- PLAYING
  Bearbeitung des Datenstreams (Clock Signal aktiv)

Die Pipeline selbst kann diese Zustände annehmen, alle Elemente in der Pipeline nehmen den globalen Zustand der Pipeline an. Eine Pipeline im Zustand NULL allokiert keine Ressourcen mehr und muss vor einem Neustart erst in den Zustand READY sein.
Über sog. Capabilities wird der Media Typ (Format) des Datenstreams zwischen zwei Elementen beschrieben oder welchen Media Typ ein Pad unterstützt. Für die Gst- Server Pipeline wird ein weiteres Filter- Element eingesetzt (Abbildung 4.6), dieses soll das Format der Kamera limitieren, da die Kamera mehrere Formate unterstützt. Dieses Filter- Element setzt die Capabilities der Kamera auf eine Auflösung von 640 Breite * 480 Höhe, Framerate auf 14 und das Pixel- Format auf UYVY (4:2:2 Sub- Sampling), zwei Pixel teilen sich dieselbe Cb/Cr Komponente, dies entspricht effektive 16 Bit per Pixel oder 66% der Rohdaten.

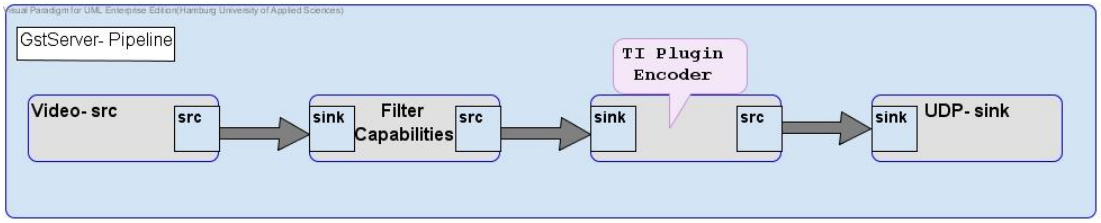


Abbildung 4.6: Erweiterte GstServer Pipeline

Weiterhin müssen die Elemente noch konfiguriert werden. Dafür werden die sog Properties des Elements gesetzt, nicht gesetzte Properties nehmen einen Default- Wert an.
Die wichtigsten Properties vom GStreamer- TIVidenc1 Plugin:

- bitRate
  Setzt die Video- Encoder Bitrate
- frameRate
  Setzt die Framerate für den Videostream zum nächsten Element
- rateControlPreset
  Einstellungen: No rate control, constant bit rate (CBR), variable bit rate (VBR)
- encodingPreset
  Einstellungen: Codec default, high speed, high quality

Die wichtigsten Properties von UDP- sink:

- clients
  IP- Adresse und Port der Clients
- sockfd
  Socket file descriptor, im default Wert- wird ein neuer Socket allokiert

Abbildung 4.7 zeigt ein Ablaufdiagramm der Initialisierungsphase der Gst- Server Klasse, nach der Initialisierungsphase wird die Pipeline über die in Abbildung 3.4 Methoden gesteuert:

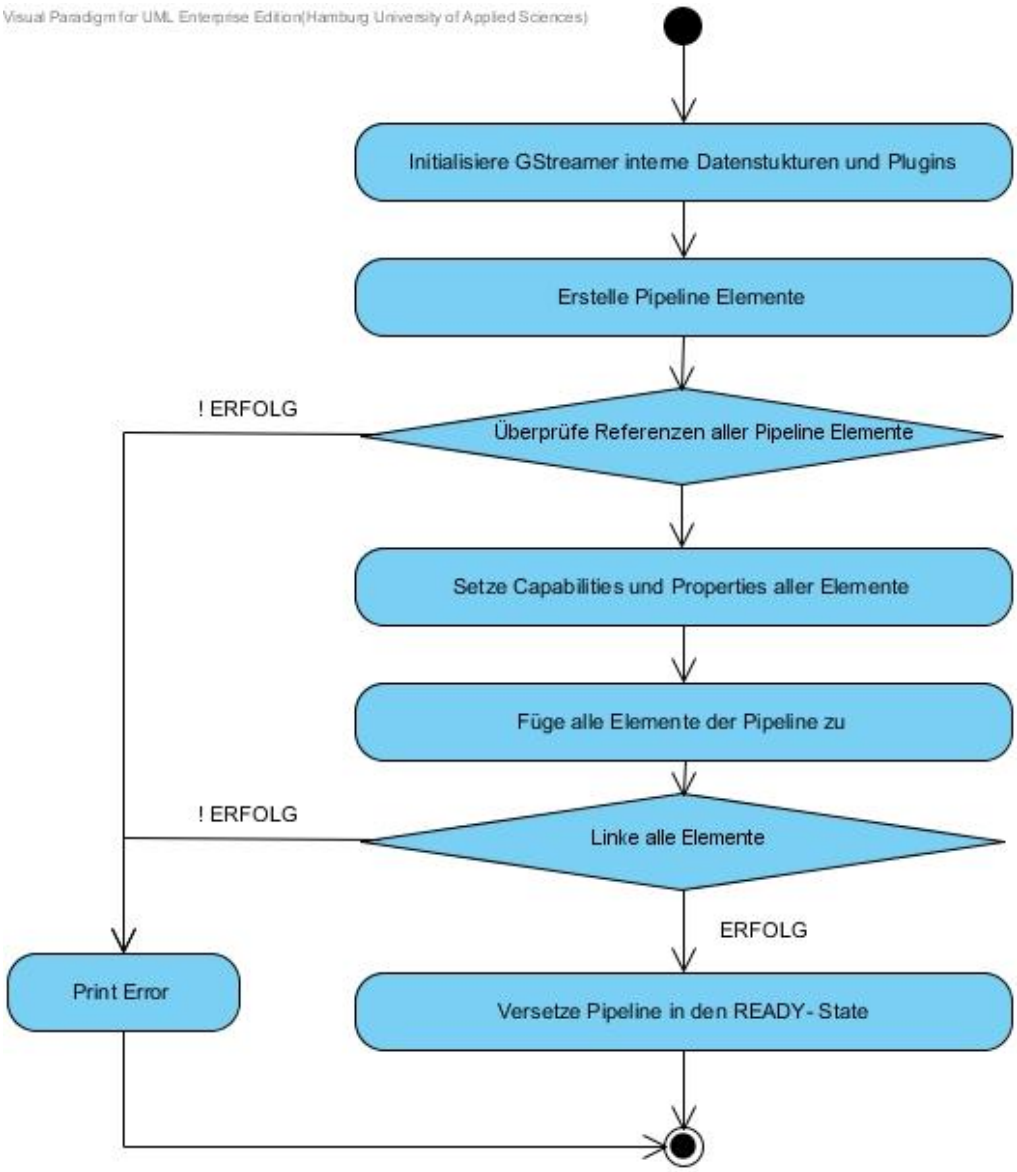


Abbildung 4.7: Initialisierung der Gst- Server Pipeline

4.2.3 Implementierung UDP-Server Klasse

Für die Kommunikation zwischen Infotainmentsystem und Kameraplattform wird ein UDP Server erstellt. Dabei handelt es sich um einen asynchronen (nicht blockierende send() und receive() Aufrufe) UDP Server, der zwei Methoden implementiert, wie in Abbildung 4.5 zu erkennen ist. Wie in Abbildung 4.8 abgebildet, nimmt der Server eine UDP Nachricht (start/stop) vom Infotainmentsystem an und startet bzw. stoppt die Gst- Server Pipeline. Dann schickt der UDP- Server eine Bestätigungs- Nachricht zurück, indem er die gesendete

Anfrage zurückschickt. Das Infotainmentsystem kann mit der Anzeige des Kamerabildes beginnen.

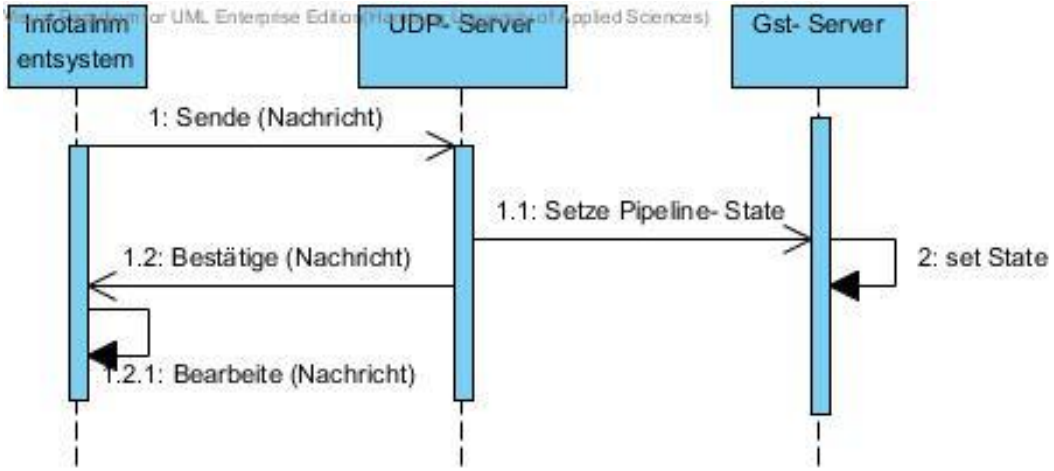


Abbildung 4.8: Nachrichtenaustausch zwischen Infotainment und Kameraplattform

Grundlage des UDP- Servers bildet die Boost Bibliotheksklasse io_service. Ein io_service Objekt dient als logisches Interface zum Betriebssystem vgl. Abbildung 4.9 . Um Input und Output Operationen durchzuführen wird ein I/O Objekt erstellt, dies ist der UDP- Socket. Zur Erstellung eines UDP- Sockets wird dem Konstruktor das io_service Objekt und ein UDP-Endpoint übergeben. Ein UDP Endpoint bestimmt die IP- Adresse und den Port des UDP-Server Sockets. Die in Abbildung 4.5 dargestellten Methoden des Servers sind die Handle-Methoden, sobald eine send() oder receive() Operation abgeschlossen ist, wird die jeweilige handle()- Methode vom io_service Objekt aufgerufen. In der letzten Phase wird auf dem UDPSocket Objekt die async_receive Methode aufgerufen. Damit ist der UDP- Server bereit für den Nachrichtenempfang.
In Abbildung 4.10 ist die handleReceive/Send() Methode abgebildet. Wie zu erkennen ist, wird nach jedem Nachrichtenempfang, erneut die asyn_receive Methode aufgerufen, damit der Server eine erneute Nachricht empfangen kann. Falls eine Nachricht nicht gesendet werden konnte, wird die Pipeline gestoppt und der State NULL gesetzt, das Infotainmentsystem muss eine erneute Anfrage senden, der UDP- Server muss auch in diesem Fall wieder zum Nachrichtenempfang bereit sein und die async_receive Methode aufrufen.

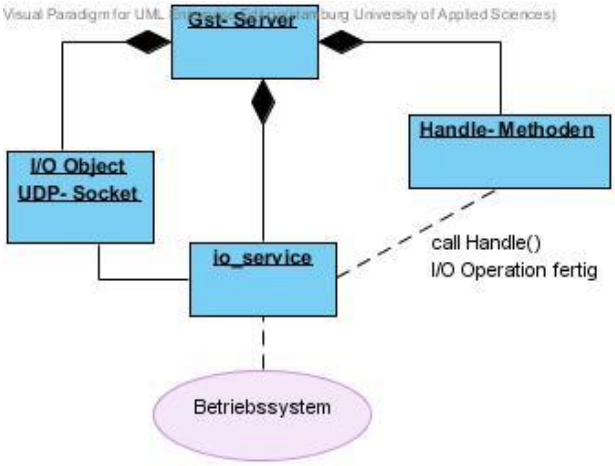


Abbildung 4.9 Übersicht Boost asynchrone I/O (vgl. [26])

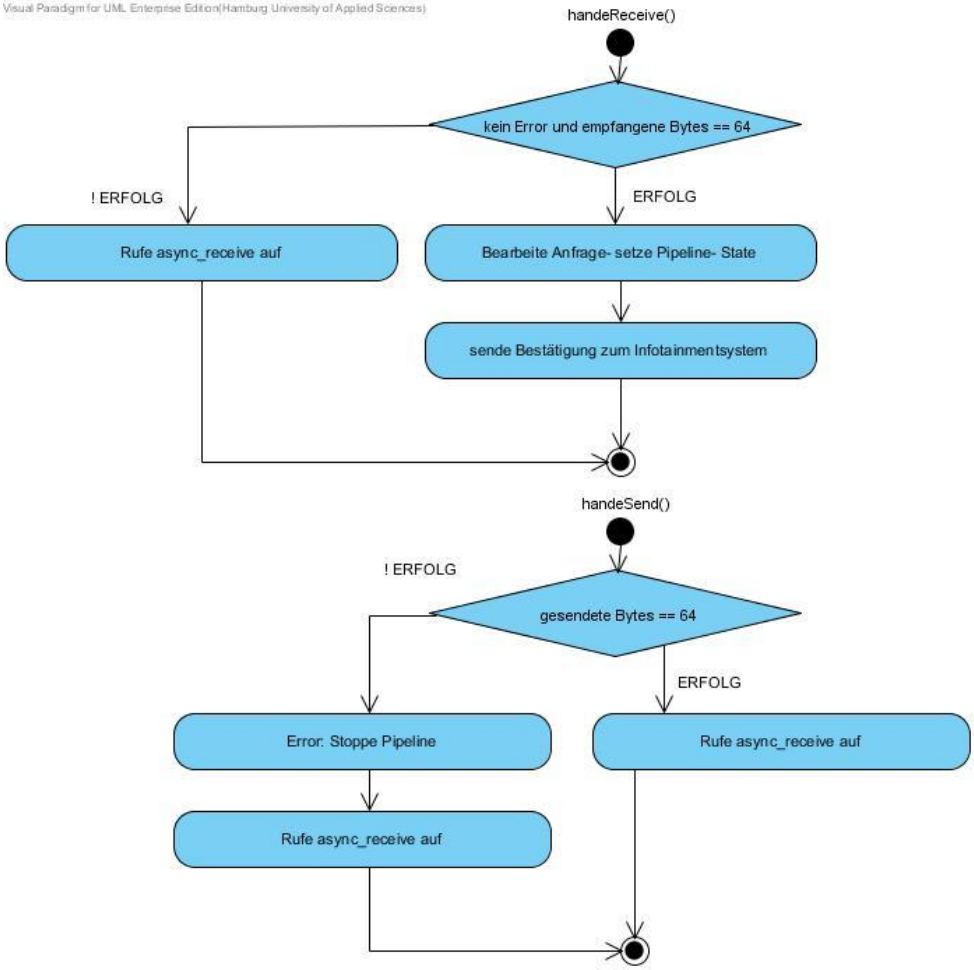


Abbildung 4.10: Die Gst- Server Methoden handleReceive() und handleSend()

4.3 Realisierung Infotainmentsystem

In Abbildung 4.11 ist ein aktualisierter Ausschnitt der Klassen für das Infotainmentsystem zu sehen. Die Kommunikation findet mittels UDP Nachrichten über die UDP_communication Klasse statt. Für die Realisierung der Gplayer- Klasse wird, wie auf der Kameraplattform das GStreamer- Framework verwendet. Folgende Bibliotheken müssen zusätzlich installiert werden:

- GStreamer und Glib-2.0- bereits bekannt aus vorherigem Abschnitt.
- Gstinterfaces
  Diese Bibliothek wird benötigt, damit der Videostream auf dem GUI angezeigt wird.

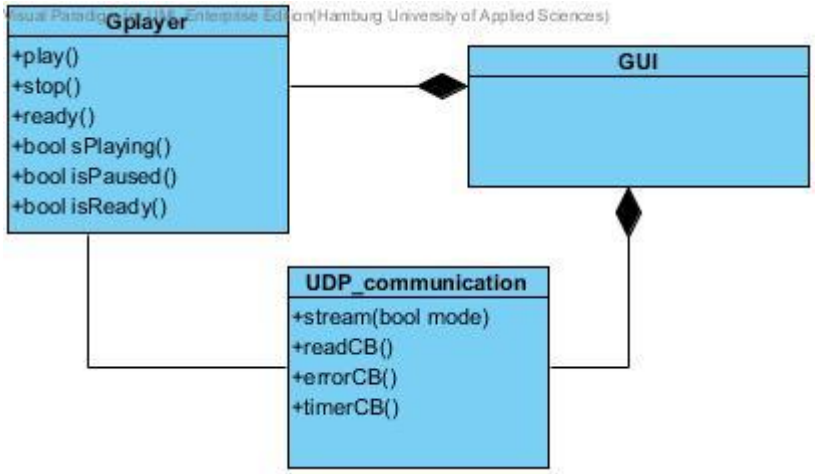


Abbildung 4.11: Realisierte Klassen für die Erweiterung des Infotainmentsystems

Die Gplayer Klasse ist für die Initialisierung der Pipeline zuständig. Über die in Abbildung 4.11 dargestellten Funktionen werden die Pipeline- States geändert und abgefragt. Ein zentrales Feature vom QT- Framework sind die sog Signale und Slots (vgl. Abbildung 4.12). Diese werden für die Kommunikation zwischen Objekten genutzt. Für die GUI Programmierung ist es wichtig, dass wenn sich der Zustand eines Objektes ändert ein anderes Informiert wird. Um ein Beispiel zu nennen, die Klasse Networkreceiver (vgl. Abbildung 3.15 aus vorherigem Kapitel) besitzt mehrere Signale (Blinken links, blinken rechts und Modus).
Ist eine CAN- Nachricht mit der Information blinken links eingetroffen, wird dieses Signal durch ein emit(blinken links)- Befehl angestoßen. Die Klasse GUI besitzt mehrere Slots. Unter anderem einen Slot zum Setzen des linken oder rechten Blinklichts in der grafischen Benutzeroberfläche. Über die *connect()* Methoden wird das Signal der Networkreceiver-Klasse mit dem Slot der GUI- Klasse verbunden. Dadurch fängt die Anzeige für den linken Blinker an zu blinken.
Der Slot ist eine normale C++ Methode und muss implementiert werden. Die Signale werden nur in der Header- Datei- bzw. der Klassen- Deklaration bekannt gegeben. Signale können Übergabeparameter besitzen, aber niemals einen Rückgabewert. Signale müssen

nicht definiert werden. Bei den Signalen und Slots handelt es sich um C++ Extensionen, diese können nicht von einem C++ Compiler übersetzt werden. QT benutzt deshalb ein Meta- Object Compiler (moc). Dieser liest die C++ Header Datei, wird eine oder werden mehrere Klassen Deklararationen mit einem Q_OBJECT Macro gefunden, erstellt der moc C++ Source Code für diese Klassen. Der erstellte Meta- Object Code wird dann von einem C++ Compiler übersetzt. Dieser Feature wird auch bei der UDP_communication Klasse eingesetzt. Es wird ein UDP- Socket erstellt. Dieser besitzt unter anderem das Signal readyRead(). Empfängt der Socket eine Nachricht, wird dieses Signal ausgelöst, dadurch wird ein nicht blockierender Read- Befehl auf dem UDP Socket realisiert.

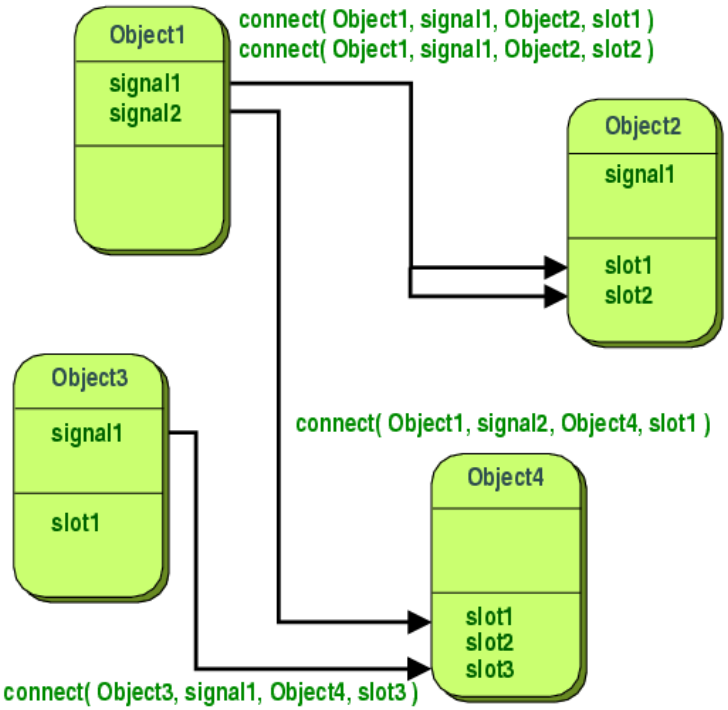


Abbildung 4.12: Signale und Slots, ein Feature von dem QT- Framework (Quelle:[26])

4.3.1 Implementierung Gplayer Klasse

Der Aufbau der Gplayer Pipeline ist dem Aufbau der Gst-Server Pipeline ähnlich. Damit das Bild auf der GUI angezeigt wird, muss dem sink Element mittgeteilt werden, auf welchem Fenster der Videostream ausgegeben wird. Jedes Fenster der Infotainmentsystems besitzt eine eindeutige ID. Die ID des Fensters für die Rückfahrkamera wird abgefragt und mit der Methode gst_x_overlay_set() wird dem sink- Element mittgeteilt, auf welchem Fenster die Ausgabe des Videobildes stattfinden soll. Andernfalls stellt das GStreamer- Framework ein eigenes Fenster zur Anzeige bereit. In Abbildung 4.13 ist die Gplayer- Pipeline zu sehen. Das src- Element ist eine UDP- src. Dieses Element nimmt den UDP- Stream der Kameraplattform entgegen. Durch das h264- Decoder Element wird der Videostream dekodiert und zuletzt mit dem xvideosink- auf dem Bildschrim ausgegeben. Die

Initialisierung erfolgt nach dem gleichen Muster wie in Abbildung 4.7 bei der GstServer-Pipeline und wird nicht erneut dargestellt. Wie bereits erwähnt, dienen die Methoden der Klasse zur Steuerung der Pipeline- States.

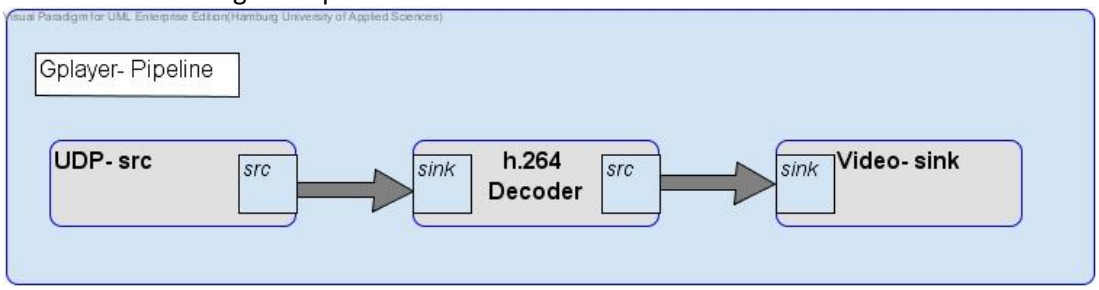


Abbildung 4.13: Gplayer- Pipeline

4.3.2 Implementierung UDP-communication Klasse

Es wird ein UDP Socket erstellt und IP-Adresse/Port gebunden. Da die Nachrichtenübertragung nicht garantiert ist, wird versucht, die Nachricht mehrmals zu übertragen. Wenn die Nachricht (Start oder Stop) nicht innerhalb 1 Sekunde übertragen ist, wird eine Fehlermeldung ausgegeben. Dazu wird ein Timer gestartet, der periodisch alle 200ms ausgelöst.

Per Timeout() Signal wird die Methode timerCB() aufgerufen. Diese sendet die Start bzw. Stop Nachricht an die Kameraplattform, dabei wird ein Counter erhöht. Trifft eine UDP Nachricht ein, wird die Methode readCB() aufgerufen, der Timer gestoppt und der Counter zurückgesetzt. Mit Hilfe der Signale wird ein schleifenfreier und nicht blockierender Zustand in der UDP_communication Klasse erreicht. Die Signale und Slots werden wie in der Abbildung 4.12 mit der connect() Methode gebunden.

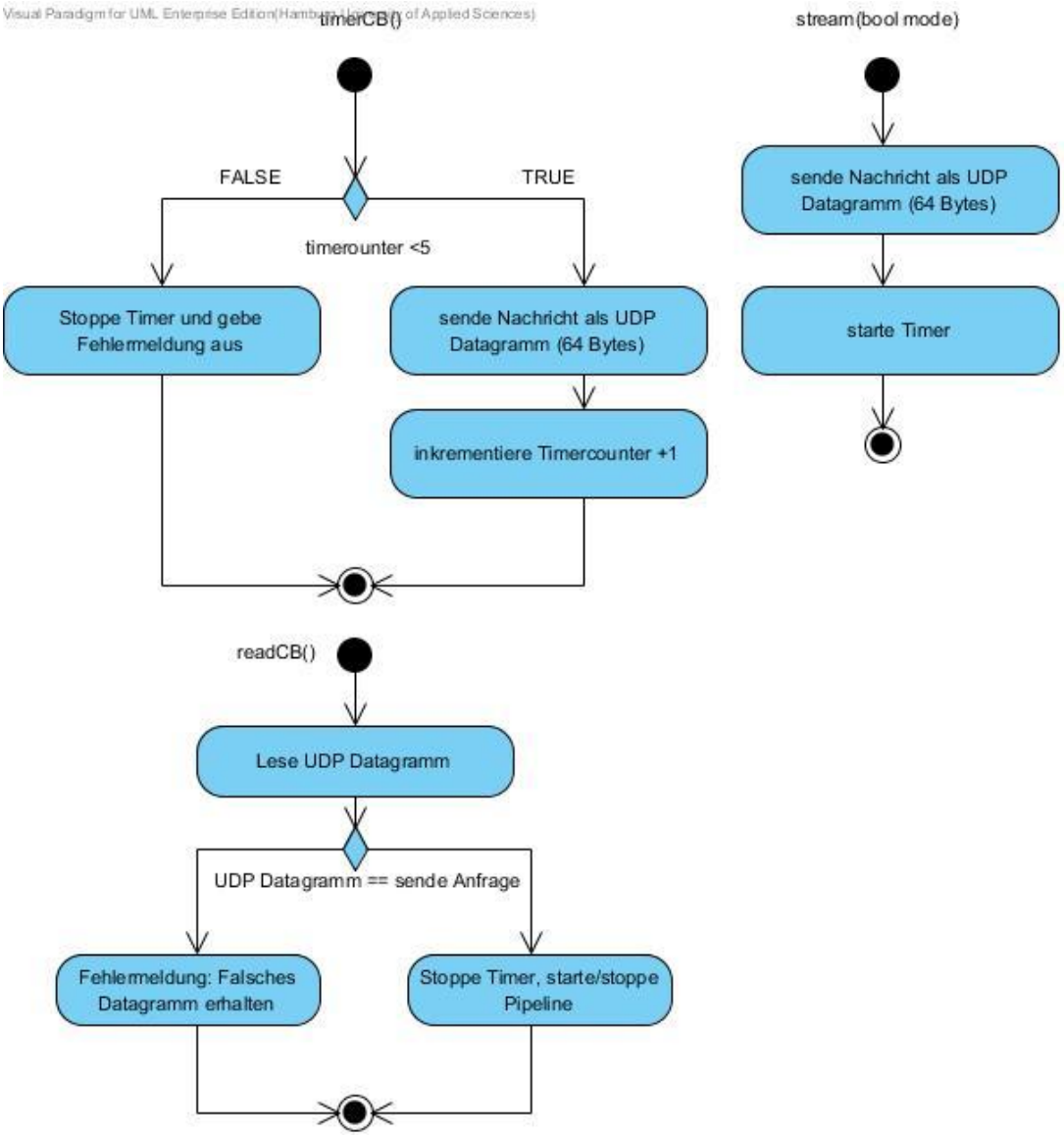


Abbildung 4.14: Ablauf der Methoden Udp_communication Klasse

4.4 Konfiguration TTEthernet Switches

Für die Beschreibung des TTEthernet Netzwerks stellt TTTech ein eigenes XML- basiertes Format zur Verfügung. Mit einer speziellen Toolchain wird das Netzwerk geplant und die Konfiguration auf die Switches hochgeladen.
Für die Kameraplattform wird ein neues Gerät eingetragen. Eigenschaften des Geräts sind unter anderem das Schedule der Nachrichten und der Nachrichten- Typ. Als Basiseinstellung wird ein RC- Buffer eingetragen (Critical ID 0x450). Der BAG- Value wird auf 1µs gesetzt und die Ethernet- Framegröße auf 1518 Bytes. Damit werden alle Ethernet-Frames von der Kameraplattform durchgelassen. Durch den Mikrocontroller basierten TTE-Protokollstack wird jedoch höchstens alle 125µs eine Nachricht am Switch empfangen.
Es müssen neue, sog. Virtual- Links eingetragen werden. Diese bilden eine uni- direktionale logische Verbindung zwischen Infotainment- und Kamerasystem. Es werden drei neue virtuelle Verbindungen erstellt, siehe Abbildung 4.15. Virtual Links werden nur für den kritischen Datenverkehr eingetragen (TT- RC- Nachrichten). Für die Verbindung vom Infotainmentsystem zum Kamerasystem wird kein Virtual- Link definiert- da es sich um keinen kritischen Datenverkehr handelt.

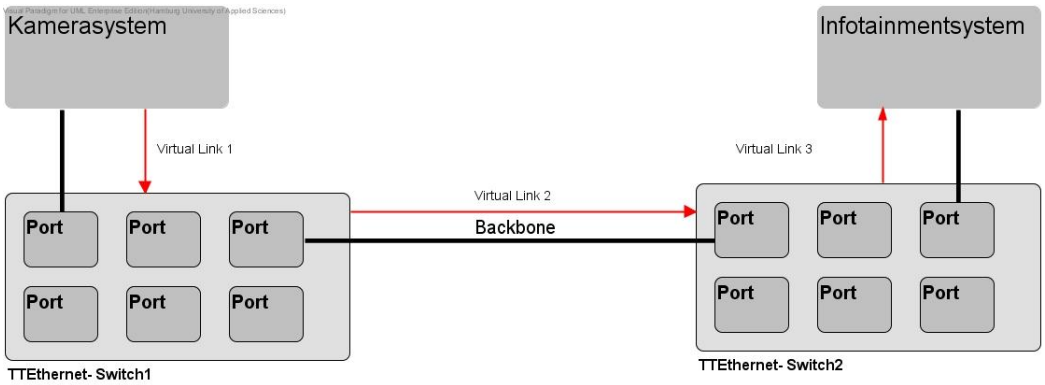


Abbildung 4.15: Übersicht neuer Virtual- Links im TTEthernet

5 Test und Ergebnisse

Im folgenden Abschnitt wird die Funktion des Kamerasystems getestet und überprüft, ob die gestellten Anforderungen erfüllt werden. Ein erster Funktionstest wird mit einem Videotestbild durchgeführt, welches mit Hilfe von GStreamer erstellt wird. Über das *Videotestsrc*- Element (GStreamer Pipeline), können verschiedene Videotestbilder erzeugt werden, in Abbildung 5.1 ist das verwendete Testpattern zu sehen. Der Aufbau des Testnetzwerkes entspricht dem Teilaufbau des Demonstrator- Netzwerkes, dies wird in Abbildung 5.2 gezeigt. Die Kameraplattform soll synchron im Netzwerk laufen, dafür befindet sich ein Compression- Master (vgl. Grundlagen 2.4) im TTEthernet Netzwerk- der Synchronisationsnachrichten verschickt.

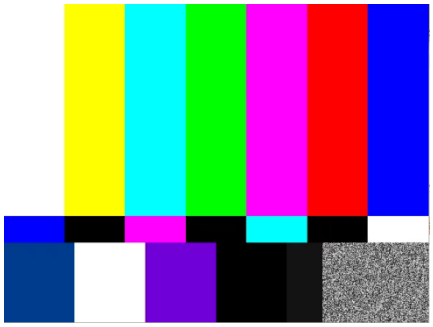


Abbildung 5.1: GStreamer Testpattern für die Videoübertragung

Das Testbild wird mit 30 Bildern pro Sekunde, bei einer Auflösung von 640*480 Pixel und dem UYVY(4:2:2 Subsampling) Format erzeugt, damit wird eine konstante Datenrate von 2,6 MBit/s erzeugt. Nicht ankommende Frames machen sich mit einer fehlerhaften Darstellung des Bildes und der nicht erreichten Datenrate von 2,6 MBit/s bemerkbar. Mit dem Netzwerkanalyse- Tool Wireshark wird der Datenverkehr gemessen. Dieser Funktionstest wird erfolgreich bestanden, es wird kein Fehler in der Darstellung entdeckt und die Datenrate ist konstant bei 2,6 MBit/s.

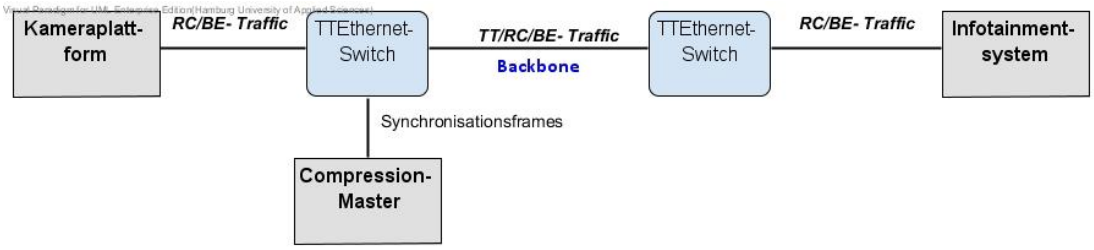


Abbildung 5.2: Testumgebung für Kamerasystem

Zur weiteren Validierung des Kamerasystems wird ein Live- Videobild übertragen. Die Encoder- Einstellungen sind:

- Video- Encoder bit rate = 2Mbit
- Format UYVY (4:2:2), Framerate 14, 640*480 Pixel
- Kein rate- Control
- Codec: h264
- Alle anderen Eigenschaften nehmen Default- Werte an (nicht wichtig für Anwendung)

Die Videoübertragung des Live- Bildes funktioniert korrekt, es sind keine Artefakte auf der Anzeige des Infotainmentsystems zu erkennen. Es werden Szenen mit mehr oder weniger Bewegung simuliert. Wie in der Abbildung 5.3 zu erkennen ist, wird zyklisch ca. alle 2 Sekunden ein I- Frame (Einzelbild/Vollbild) übertragen. Bei Bewegungen im Bild beträgt die gemessene Übertragungsrate 0,7 MBit/s. Herrscht kaum Bewegung, liegt diese bei 0,1 MBit/s - der produzierte Traffic resultiert größtenteils aus den übertragenden I- Frames. In den Encoder- Einstellungen kann man diese nicht unterdrücken. Die ARM MCU und DSP Auslastung liegt bei ca. 25% (schwankt je nach Bewegung 25% ist das Maximum). Die Encoding- Latenz liegt bei < 50ms, dieser Wert kann nicht gemessen werden- und wird der Dokumentation des Codecs entnommen (DSP C64x+ Encoder).

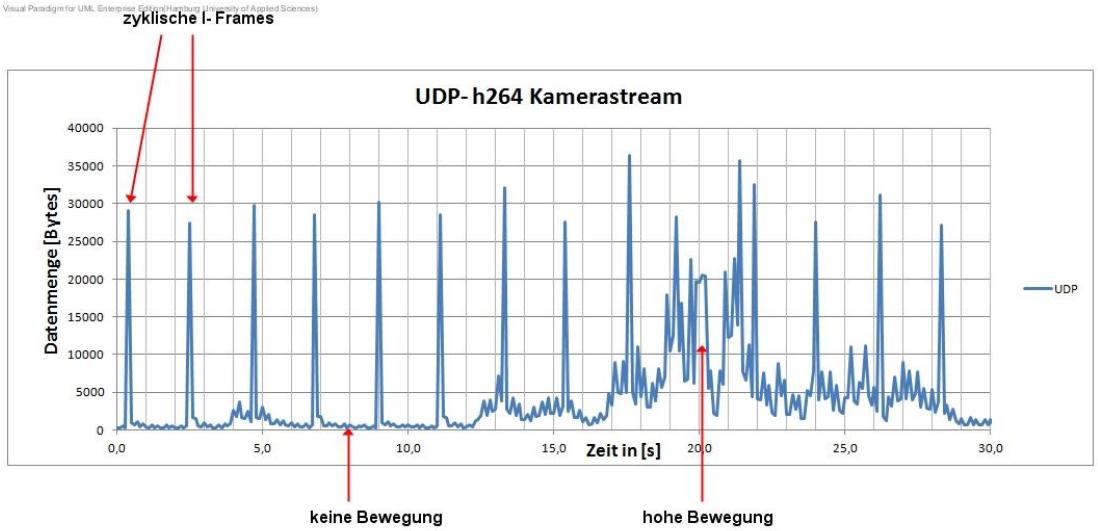


Abbildung 5.3: Von der Kameraplattform encodiertes (h264) Live Videobild

Bei einer Leistungsmessung wird die Performance vom Mikrocontroller basierten TTE-Protokollstack mit den Erweiterungen für diese Arbeit gemessen. Der zu-grundeliegende Schedule hat sich nicht geändert, es werden bis zu 39 Frames in 5ms verarbeitet.
Über einen Paket/Traffic- Generator (Ostinato) wird Netzwerk- Traffic erzeugt, der Bilddaten Häufungen (eng. burst) simulieren soll. Der Paket- Generator sendet mit 100 MBit/s. Der Aufbau für diesen Test ist in Abbildung 5.4 zu erkennen. Mit dem Netzwerkanalyse- Tool Wireshark wird gemessen, ob alle gesendeten Ethernet- Frames (Pakete) erfolgreich übertragen werden.

Abbildung 5.4: Performance- Testaufbau

Es wird mit minimaler (64 Byte) und maximaler (1518 Byte) Ethernet- Framelänge gemessen. Dabei hat es sich gezeigt, dass mit der jetzigen Implementierung höchstens 22 Ethernet- Frames mit maximaler Länge als „burst“ übertragen werden können, bevor ein Paketverlust auftritt. Werden minimale Ethernet- Frames übertragen, liegt das Maximum an übertragbaren Ethernet- Frames, bei 19.

Fälle bei denen eine Überlastung auftritt:

- Der Output- Buffer zum TTEthernet ist voll
  Dieser ist auf 20 Frames begrenzt, durch den Schedule von 39 Nachrichten in 5ms (96 MBit/s), tritt dieser Fall nur ein, wenn minimale Frames als burst übertragen werden
- System- Buffer ist voll (Out of Memory)

Dies hat zur Folge, dass Übertragungen vom Infotainmentsystem, die einen burst größer als 22 Frames zu Paketverlust führen (maximale Ethernet- Framelänge). Damit darf ein Vollbild die Größe von 33.000 Byte nicht überschreiten.
Die Encoder- Einstellungen Constant- Bitrate (CBR) und Variable Bitrate (VBR) führen dazu, dass dieses Limit überschritten wird. Bei der Einstellung für (CBR) wird beispielweise ein Burst von 41 Ethernet- Frames erzeugt. Mit der jetzigen Implementierung ist die verlustfreie Videoübertragung nur ohne rate- control möglich.

Generelle Anforderungen:

- Das Kamerabild wird auf dem Infotainmentsystem angezeigt
  erfüllt
- Das Kamerabild wird über das Infotainmentsystem gestartet und gestoppt.
  erfüllt
- Mikrocontroller transparent implementiert
  Nachrichtenverlust bei zu hohem burst, keine TCP- Verbindung möglich
  nicht erfüllt
- TCP/IP- Referenzmodell- alle Schichten sollen nutzbar sein
  Es kann kein Dienst genutzt werden der auf TCP- Sockets aufbaut, nur Dienste die UDP als Transportprotokoll nutzen sind verfügbar.
  nicht erfüllt- resultierend aus vorheriger Anforderung
- H264 Video- Encoding
  Das Encodieren läuft auf dem IVA 2.2 Subsystem (DSP + Video- Beschleuniger), die ARM MCU und die DSP sind kaum ausgelastet (25%).
  erfüllt
- Der Kamerastream wird als zeitkritische Nachricht (Critical Traffic) übertragen
  erfüllt

6 Fazit und Ausblick

In der Arbeit ist ein Time-Triggered basiertes Rückfahrkamerasystem realisiert worden. Es hat sich jedoch gezeigt, dass die Nachrichtenklasse TT in Kombination mit Videokompression eine Bandbreiten- Verschwendung hervorruft. Diese Nachrichtenklasse macht nur Sinn, wenn Einzelbilder übertragen werden. Für die Nachrichtenklasse TT, muss die Kameraplattform mit einem Echtzeit- Betriebssystem ausgestattet sein, damit die Daten zu den definierten Sendezeitpunkten bereitstehen. Für eine Videokompression nach h.264 sind RC- Nachrichten die geeignete Wahl. Es hat sich gezeigt, dass unter der Verwendung des TTE- Protokollstack die Übertragungsrate limitiert ist.
Durch eine Anpassung der Implementierung muss eine Optimierung der Performance erreicht werden, dies ist im Rahmen dieser Arbeit nicht mehr möglich gewesen. Alternativ kann dieser „Flaschenhals“ direkt auf der Kameraplattform behoben werden, indem ein eigenes GStreamer Plugin (UDP sink- Element mit Zeitverzögerung) entworfen wird, welches Nachrichten RC- konform sendet.
Die Kameraplattform bietet weitere Möglichkeiten und Ressourcen, z.B. kann durch die verwendete Hardware ein kompaktes Infotainmentsystem entwickelt werden. Die Plattform besitzt Grafik- Beschleuniger und einen Display Prozessor (vgl. Abbildung 3.9). Mit dieser Plattform lassen sich außerdem viele Fahrerassistenzaufgaben realisieren, wie z.B. Kollisionserkennung oder eine Einparkhilfe. Will der Fahrer einparken, so wird das Kamerabild analysiert und dem Fahrer visuell im Infotainmentsystem mittgeteilt, ob sein Einparkmanöver erfolgreich ist oder wie es korrigiert werden muss. Erkennt der Algorithmus eine nahende Kollision, so kann eine zeitkritische Nachricht an das Bremssystem übertragen werden und ein Bremsvorgang wird ausgelöst.

Abbildungsverzeichnis

Tabellenverzeichnis

7 Literaturverzeichnis

[1] Aeronautical Radio Incorporated. Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network. Standard ARINC Report 664P7-1, ARINC, 2009.

[2] Lehrbuch Verlag Internet Akademie. Kostenlose Kurse Differenzenkodierung.

[3] Florian Bartols. Leistungsmessung von Time-Triggered Ethernet Komponenten unter harten Echtzeitbedingungen mithilfe modifizierter Linux-Treiber. Bachelorthesis, HAW Hamburg, Hamburg, July 2010. Bachelorthesis.

[4] beagleboard.org. Beagleboard-xm system reference manual. – URL http://beagleboard.org/static/BBxMSRM_latest.pdf - Zugriffsdatum: 2012-08-12

[5] Helmut Bähring. Digitale signalprozessoren. In *eXamen.press*, pages 46–55–. Springer Berlin Heidelberg, 2010. - URL http://dx.doi.org/10.1007/978-3-642-12292-7_3 - ISSN 978-3-642-12291-0

[6] Hussein Charara, Jean-Luc Scharbarg, Jérôme Ermont, and Christian Fraboul. Methods for bounding end-to-end delays on an AFDX network. In *18th Euromicro Conference on Real-Time Systems*, 2006. – URL http://dx.doi.org/10.1109/ECRTS.2006.15

[7] CoRE RG. Communication over Real-time Ethernet. – URL http://core.informatik.haw-hamburg.de

[8] Hilscher Gesellschaft für Systemautomation mbH. Hal ethernet mac netx . – URL www.hilscher.com

[9] Freescale. Freescale semiconductor. – URL http://www.freescale.com

[10] Tilo Gockel, Rüdiger Dillmann, and Joachim Schröder. Gerätereiber und kernelmodule. In *Embedded Linux*, X.systems.press, pages 231–262. Springer Berlin Heidelberg, 2009. – URL http://dx.doi.org/10.1007/978-3-540-78620-7_11 - ISBN 978-3-540-78619-1

[11] Pierre Hansch and Christian Rentschler. Encoding. In *X.media.press*, pages 205–238–. Springer Berlin Heidelberg, 2012. – URL http://dx.doi.org/10.1007/978-3-642-13993-2_9 - ISSN 978-3-642-13992-5

[12] Aptina Imaging. Mt9p031datasheet. – URL http://www.aptina.com/products/image_sensors/mt9p031i12stc/ - Zugriffsdatum: 2012-07-22

[13] Texas Instruments. Am/dm37x multimedia device silicon revision 1.xtechnical reference manual. – URL http://www.ti.com/lit/ug/sprugn4r/sprugn4r.pdf - Zugriffsdatum: 2012-06-3

[14] Texas Instruments. Dsp/bios link overview. – URL http://processors.wiki.ti.com/index.php/Category:DSPLink – Zugriffsdatum: 2012-07-08

[15] Texas Instruments. Texas instruments inc. semiconductors. – URL http://www.ti.com

[16] Texas Instruments. Buibuild gstreamer application report spraaq9, 1 2008. – URL http://www.ti.com/lit/an/spraaq9/spraaq9.pdf - Zugriffsdatum: 2012-7-20

[17] Jürgen Jasperneite. Echtzeitkommunikation. In Juliane T. Benra and Wolfgang A. Halang, editors, *Software-Entwicklung für Echtzeitsysteme*, pages 97–128–. Springer Berlin Heidelberg, 2009. – URL http://dx.doi.org/10.1007/978-3-642-01596-0_5 - ISSN 978-3-642-01595-3

[18] Jan Kamieth. Entwurf einer Mikrocontroller basierten Bridge zur Kopplung von CAN Bussen über Time-Triggered Realtime Ethernet. Bachelorthesis, August 2011.

[19] D. Kraft. Kommunikationsnetze. In Hans-Jürgen Gevatter and Ulrich Grünhaupt, editors, *VDI-Buch*, pages 525–527–. Springer Berlin Heidelberg, 2006. – URL http://dx.doi.org/10.1007/3-540-29980-7_41

[20] Lothar Krank, Ansgar Meroth, Andreas Streit, and Boris Tolg. Netzwerke im fahrzeug. In *Infotainmentsysteme im Kraftfahrzeug*, pages 147–207–. Vieweg, 2008. – URL http://dx.doi.org/10.1007/978-3-8348-9430-4_5

[21] N. Lawrenz, W. und Obermöller. *CAN Controller Area Network: Grundlagen, Design, Anwendungen, Testtechnik*. Number ISBN 978-3-8007-3332-3. VDE-Verlag, 5. neu bearbeitete auflage edition, 2011.

[22] Jan Lipfert. Technical Data Reference Guide - netX500/100. Hilscher GmbH, December 2008. – URL http://www.hilscher.com

[23] Christoph Meinel and Harald Sack. Die grundlage des internets: Tcp/ip-referenzmodell. In *Internetworking*, X.media.press, pages 31–67. Springer Berlin Heidelberg, 2012. – URL http://dx.doi.org/10.1007/978-3-540-92940-6_2

[24] Kai Müller. Time-triggered ethernet für eingebettete systeme: Design, umsetzung und validierung einer echtzeitfähigen netzwerkstack-architektur. Bachelorthesis, August 2011.

[25] Sebastian Möller. Qualität von video-Übertragungssystemen. In *Quality Engineering*, pages 97–121–. Springer Berlin Heidelberg, 2010. – URL http://dx.doi.org/10.1007/978-3-642-11548-6_6

[26] Digia Qt Project Nokia. Qt meta object compiler. – URL http://qt-project.org/doc/qt-4.8/moc.html - Zugriffsdatum: 2012-10-25

[27] Konrad Reif. Grundlagen der vernetzung. In Konrad Reif, editor, *Bosch Autoelektrik und Autoelektronik*, pages 70–81–. Vieweg+Teubner, 2011. – URL http://dx.doi.org/10.1007/978-3-8348-9902-6_2

[28] Konrad Reif. Bussysteme. In *Automobilelektronik*, pages 1–34–. Vieweg+Teubner Verlag, 2012. – URL http://dx.doi.org/10.1007/978-3-8348-8658-3_1

[29] Jan Schulz. Kompressionsverfahren für video und audio. In *Kompendium Medieninformatik*, X.media.press, pages 1–82. Springer Berlin Heidelberg, 2006. – URL http://dx.doi.org/10.1007/3-540-30226-3_1

[30] Analog Device Inc. Semiconductor. Analog device inc. – URL http://www.analog.com/en/index.html

[31] Society of Automotive Engineers - AS-2D Time Triggered Systems and Architecture Committee. Time-Triggered Ethernet AS6802. SAE Aerospace, November 2011. – URL http://standards.sae.org/as6802/ - Standard

[32] Wilfried Steiner. TTEthernet Specification. TTTech Computertechnik AG, November 2008. – URL http://www.tttech.com

[33] Vitalij Stepanov. Mikrocontroller und CAN-basierte verteilte Regelung einer Steer-by-Wire Lenkung mit harten Echtzeitanforderungen. Bachelorthesis, August 2011.

[34] Tilo Strutz. Standards zur einzelbildkompression (jpeg). In *Bilddatenkompression*, pages 192–260–. Vieweg+Teubner, 2009.

[35] Tilo Strutz. Wahrnehmung und farbe. In *Bilddatenkompression*, pages 173–191–. Vieweg+Teubner, 2009. – URL http://dx.doi.org/10.1007/978-3-8348-9986-6_8 - ISSN 978-3-8348-0472-3

[36] Andrew S. Tanenbaum. *Computernetzwerke*. Pearson Studium, 4., überarb. a. edition, 2003. – ISBN 3827370469

[37] TTTech Computertechnik AG. – URL http://www.tttech.com

[38] TTTech Computertechnik AG. TTEthernet Application Programming Interface. TTTech Computertechnik AG, December 2008.

[39] Konrad Wallentowitz, Henning und Reif. Bordnetz und vernetzung. In Henning Wallentowitz and Konrad Reif, editors, *Handbuch Kraftfahrzeugelektronik*, pages 179–314–. Vieweg, 2006. – URL http://dx.doi.org/10.1007/978-3-8348-9121-1_4 - ISSN: 978-3-528-03971-4

[40] Markus Zahn. Grundlagen der socket-programmierung. In *Unix-Netzwerkprogramminerung mit Threads, Sockets und SSL*, X.systems.press, pages 147–234. Springer Berlin Heidelberg, 2006. – URL http://dx.doi.org/10.1007/3-540-38302-6_4 - ISSN 978-3-540-00299-4

Versicherung über Selbstständigkeit

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

_____

Hamburg, 22. Mai 2013 Sebastian Kuhrt