Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Jens-Peter Eltze

Entwicklung eines DSP-basierten Stereokamerasystems zur Positionsbestimmung eines bewegten Objekts in Echtzeit

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

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

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

Jens-Peter Eltze

Entwicklung eines DSP-basierten Stereokamerasystems zur Positionsbestimmung eines bewegten Objekts in Echtzeit

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

Betreuender Prüfer: Prof. Dr.-Ing. Hans Peter Kölzer
Zweitgutachter: Prof. Dr.-Ing. Stephan Hußmann

Abgegeben am 4. März 2010

Jens-Peter Eltze

Thema der Masterarbeit

Entwicklung eines DSP-basierten Stereokamerasystems zur Positionsbestimmung eines bewegten Objekts in Echtzeit

Stichworte

Stereovision, Bildverarbeitung, Binarisierung, Morphologische Operationen, Disparität, DSP, FPGA, EDMA, DSP/BIOS, TMS320C6416T

Kurzzusammenfassung

Ziel dieser Arbeit ist die Entwicklung eines DSP-basierten Stereokamerasystems zur Positionsbestimmung eines bewegten Objekts in Echtzeit. Über zwei achsparallel zueinander angeordnete Kameras, die zeitlich synchron operieren, werden Bilder aufgenommen. Mittels Bildverarbeitungsalgorithmen wie Binarisierung, Opening und Closing wird das gesuchte Objekt vom Hintergrund separiert. Anschließend wird anhand der horizontalen Verschiebung (Disparität) der Massenschwerpunkte und intrinsischen Systemparametern (Abstand, Brennweite) die Position des Objekts im Raum berechnet.

Jens-Peter Eltze

Title of the paper

Development of a DSP-based stereocamerasystem for position determination of a moving object in realtime

Keywords

Stereo vision, Image processing, binarisation, morphological operations, disparity, DSP, FPGA, EDMA, DSP/BIOS, TMS320C6416T

Abstract

Goal of this thesis was the development of a DSP-based stereocamerasystem for position determination of a movin object in realtime. Images are taken via two parallel aligned cameras, which are time synchronized. By using image processing algorithms like binarisation, opening and closing the searched object will be separated from the background of the image. In the end the position of the object in space will be calculated by the horizontal displacement (disparity) of the center of mass of the objects and the intrinsic parameters of the System (distance, focal distance).

Abkürzungsverzeichnis

ADDK	Add Signed 16-Bit Constant to Register
AOE	asynchronous output enable
ARE	asynchronous read enable
AWE	asynchronous write enable
B	Branch Using a Register
BDEC	Branch and Decrement
BITC4	Bit Count, Packed 8-Bit
CE	chip enable
CMPEQ4	Compare for Equality, Packed 8-Bit
CMPGTU4	Compare for Greater Than, Unsigned, Packed 8-Bit
CPU	Central Processing Unit
CS	Chip Select
DSK	DSP Starter Kit
DSP	Digital Signal Processor
EDMA	Enhanced Direkt Memory Acess
EMIF	External Memory Interface
EVA	Eingabe Verarbeitung Ausgabe
FF	Flipflop
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HDL	Hardware Description Language
HSV	Hue Saturation Value
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IP	Internet Protocol
IP-Core	Intellectual Property
LDDW	Load Doubleword From Memory With a 5-Bit Unsigned Constant Offset or Register Offset

LDW	Load Word From Memory With a 5-Bit Unsigned Constant Offset or Register Offset
LwIP	Leight weight Internet Protocol
MAC	Media Access Control
MPY	Multiply Signed 16 LSB x Signed 16 LSB
MV	Move From Register to Register
MVKH	Move 16-Bit Constant Into Upper Bits of Register
MVKL	Move Signed Constant Into Register and Sign Extend
OE	output enable
PACKH4	Pack Four High Bytes Into Four 8-Bit Halfwords
PACKL4	Pack Four Low Bytes Into Four 8-Bit Halfwords
PLL	Phasenregelschleife (engl. Phase-locked loop)
RAM	Random Access Memory
RE	read enable
RGB	Red Green Blue
SDRAM	Synchronous Dynamic Random Access Memory
SHL	Arithmetic Shift Left
SHRMB	Shift Right and Merge Byte
SHRU	Logical Shift Right
STDW	Store Doubleword to Memory With a 5-Bit Unsigned Constant Offset or Register Offset
STW	Store Word to Memory With a 5-Bit Unsigned Constant Offset or Register Offset
SUB4	Subtract Without Saturation, Four 8-Bit Pairs for Four 8-Bit Results
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WE	write enable
XPND4	Expand Bits to Packed 8-Bit Masks
ZERO	Zero a Register

Inhaltsverzeichnis

Tabellenverzeichnis

Bildverzeichnis

1 Einleitung

Die Natur wird oft als Vorbild für technische Entwicklungen verwendet. Es wird versucht durch Nachbildung der Natur ein technisches Problem zu lösen. Betrachtet man autonome Systeme wie Roboter, so wird der Mensch mit seinen Eigenschaften als Grundlage verwendet.

Eine Maschine oder ein Roboter funktioniert prinzipiell wie der Mensch nach dem EVA-Prinzip (Eingabe, Verarbeitung, Ausgabe). Beim Menschen geschieht die Eingabe über die ihm gegebenen Sinne, wie die akustische und visuelle Wahrnehmung über Ohren und Augen. Die Verarbeitung und Interpretation übernimmt unser Gehirn und steuert darüber z.B. unsere Stimmbänder und Gliedmaßen. So können wir Menschen mit unserer Umwelt interagieren.

Ein Roboter hingegen benötigt für die akustische und visuelle Wahrnehmung Sensoren wie Mikrofone und Kameras. Verarbeitet werden diese Daten durch Prozessoren und die Ausübung der Reaktion erfolgt über Aktoren wie Motoren oder ähnliches. Die Intelligenz des Roboters, die in der Verarbeitung der Daten und der damit resultierenden Reaktion auf ein Ergebnis liegt, muss für jeden Einzelfall entwickelt werden.

Wird uns ein Ball zugeworfen, so können wir ihn fangen, da wir das Objekt vom Hintergrund unterscheiden und die Entfernung zu unseren Händen schätzen können. Ein Roboter kann dies ebenfalls, sofern man es ihm beibringt. Dazu muss dieser aber in die Lage versetzt werden, den Ball zu sehen und die Entfernung zu schätzen.

1.1 Motivation

Im Rahmen des Forschungsprojekts ELEKTRA[1] an der HAW-Hamburg werden Bildverarbeitungsalgorithmen für verschiedene Roboter eingesetzt, damit diese autonom arbeiten. Ein späteres Ziel ist es, dass zwei Roboter autonom gegeneinander Tennis spielen. Dazu ist es nötig, dass diese den Ball erkennen und dessen Entfernung wissen, um ihn fangen zu können.

1.2 Aufgabenstellung

Ziel dieser Arbeit ist die Entwicklung eines DSP-basierten Stereokamerasystem zur Positionsbestimmung eines Objekts im Raum in Echtzeit.

Durch zwei horizontal und parallel zueinander angeordneten Kameras werden Bilder aufgenommen. Aus den Bildern wird das Objekt, in diesem Fall ein Ball mit einer signifikanten Farbe, extrahiert und der Mittelpunkt bestimmt. Die horizontale Verschiebung der Mittelpunkte gibt dann die Möglichkeit, Rückschlüsse auf die Entfernung des Objekts von den Kameras zu ziehen.

Als Hardware kommen zwei identische Hardwareplattformen zum Einsatz. Dabei handelt es sich um DSK6416 Entwicklungsboards von Spectrum Digital mit einem Fixpoint-Signalprozessor von Texas Instruments und einer Adapterplatine. Die Adapterplatine wurde von der Firma Bitec Ltd entwickelt und stellt ein FPGA, eine Netzwerkschnittstelle, sowie eine Schnittstelle für eine Kamera zur Verfügung. Zur Bildaufnahme werden zwei 5 Megapixel Kameras der Firma Omnivision verwendet.

Die Aufgabenverteilung auf die einzelnen Prozessoren sieht wie folgt aus. Einem der beiden Prozessoren stehen die Bilder beider Kameras zu Verfügung, um das Objekt zu erkennen und die Entfernung zu ermitteln. Der andere Prozessor ist für die Übertragung des Bildes einer Kamera, sowie der Ergebnisse der Entfernungsberechnung, über das Netzwerk an einen PC zur Visualisierung zuständig.

Die Kameras werden in dieser Arbeit mit einer Auflösung von 640x480 Bildpunkten und einer Bildwiederholungsrate von 30 Hz betrieben. Damit das System in Echtzeit arbeitet, muss die komplette Bildverarbeitung und Entfernungsberechnung für ein Stereopaarbild innerhalb von ca. 33 ms abgeschlossen sein.

1.3 Verwendete Hardware und Software

In dieser Arbeit kommt die folgende Hardware zum Einsatz:

- 2 Spectrum Digital DSK6416 Entwicklungsboards
- 2 Bitec DSKEye Boards mit Omnivision OV5620 Kameramodul

Unter Verwendung der Software:

- The MathWorks™ - MATLAB Version 7.8.0.347 (R2009a) vom 12.02.2009
- Mentor Graphics - ModelSim Altera Starter Edition 6.4 Revision: 2008.08 vom 22.10.2008
- Altera - Quartus II Version 9.0sp2 Build 235 SJ Web Edition vom 17.06.2009
- Texas Instruments - Code Composer Studio Version 3.3.38.2
  (Integrated Development: 5.98.0.219, BIOS: 5.31.02, Code Generation Tools: v6.0.8)

1.4 Gliederung

Das folgende Kapitel befasst sich mit den Grundlagen der digitalen Bildverarbeitung, die für diese Arbeit relevant sind. Die verwendeten Algorithmen werden in der Theorie und anhand von Beispielen erläutert.

Kapitel 3 beschreibt die eingesetzte Hardware. Aufgrund der Komplexität der Hardware gibt dieses Kapitel eine Übersicht über die verwendeten Komponenten, die zur Lösung der Aufgabe benötigt werden.

Nach der Theorie und der Beschreibung der Hardware folgt in den Kapiteln 4 und 5 die Hard- und Softwarekonfiguration, die die Voraussetzung für die eigentliche Bildverarbeitung darstellt.

In Kapitel 6 wird die Implementierung der Bildverarbeitungsalgorithmen dargestellt. Hier findet die Bildverarbeitung zur Erkennung des Objekts und der Ermittlung der Entfernung anhand der Algorithmen aus Kapitel 2 statt.

Die erzielten Ergebnisse der Masterarbeit werden in Kapitel 7 diskutiert.

Abschließend folgt ein Überblick über mögliche Weiterentwicklungen in Kapitel 8 und eine Zusammenfassung in Kapitel 9.

2 Bildverarbeitung

Dieses Kapitel befasst sich mit den in dieser Arbeit verwendeten Algorithmen der Bildverarbeitung. Angefangen mit der Darstellung von verschiedenen Farbräumen und der daraus resultierenden Farbklassifikation. Anschließend folgen einige morphologische Operationen und Algorithmen zur Merkmalsextraktion, sowie die Positionsbestimmung im Raum. Das in diesem Kapitel verwendete Beispielbild (Bild 2.1), sowie das zugehörige Skript zur Simulation mit MATLAB ist auf dem beiliegenden Datenträger zu finden (siehe Anhang A).

2.1 Farbräume

In Bild 2.1 ist ein mit einer Digitalkamera aufgenommenes Bild zu sehen. Das, was wir als Farbe wahrnehmen, ist das reflektierte Licht der zu erkennenden Objekte mit einer bestimmten spektralen Zusammensetzung. Um diese Farben voneinander unterscheiden zu können, müssen diese klassifiziert werden.

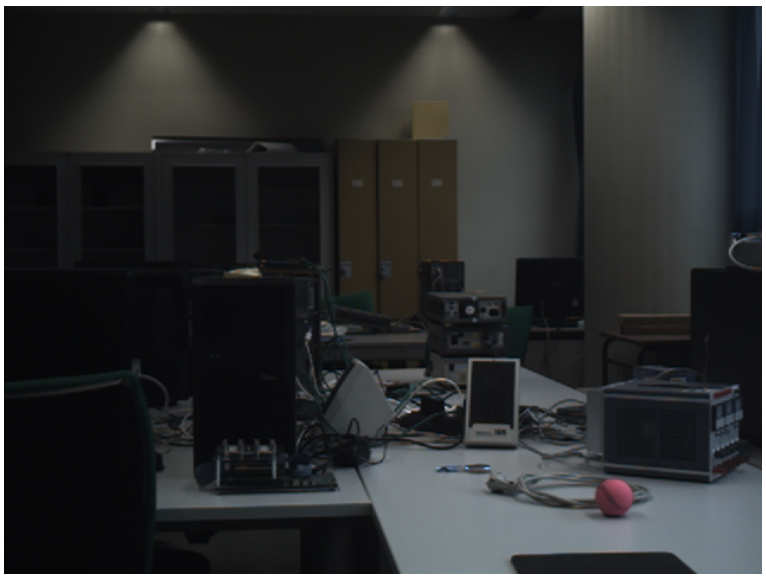
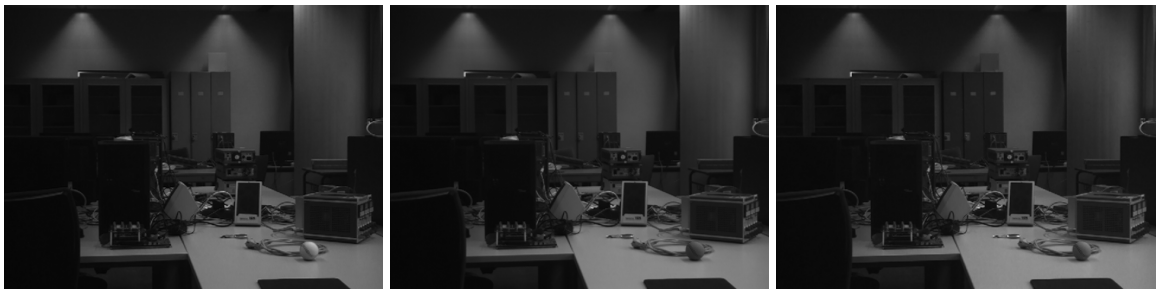


Bild 2.1: Farbbild

Ein Farbraum stellt meist eine dreidimensionale Darstellung der Farbe dar. Dabei wird jeder Farbe ein Zahlenwert zugeordnet, über den zwischen verschiedenen Farben unterschieden werden kann. Anhand einer solchen Klassifizierung können bestimmte Farben aus einem Bild extrahiert werden.
In diesen Fall soll der Ball unten rechts im Bild vom Hintergrund separiert werden. Dazu werden in den folgenden Unterkapiteln zwei Farbräume und deren Eignung dargestellt.

2.1.1 RGB-Farbraum

Beim RGB-Farbraum handelt es sich um einen additiven Farbraum. Die Farbewahrnehmung wird durch das Mischen der drei Grundfarben rot, grün und blau nachgebildet. Zerlegt man das Bild 2.1 in die drei Grundfarben, so ergeben sich die drei in Bild 2.2 gezeigten Farbanteile.



(a) Rot (b) Grün (c) Blau

Bild 2.2: RGB-Farbanteile

Der Ball ist im linken Bild deutlich heller als in den anderen beiden Bildern. In Bild 2.1 ist der Ball deutlich als Pink zu erkennen. Da Pink und Rot recht ähnlich sind, ist klar, dass der Rotanteil recht groß im Vergleich zum Grün- und Blauanteil sein muss. Zwischen dem Grünanteil (Bild 2.2(b)) und dem Blauanteil (Bild 2.2(c)) lassen sich nur geringfügige Unterschiede im Farbanteil feststellen. Vergleicht man den jeweiligen Farbanteil des gesuchten Objekts mit dem Hintergrund so, lässt sich dieser nicht sehr deutlich davon unterscheiden.

Berücksichtigt man alle drei Farbanteile, so wäre es sicherlich möglich, den Ball aus dem Bild zu extrahieren. Jedoch ist dies nur ein einzelnes Bild und keine fortlaufende Sequenz von Bildern die von einer Kamera auf einem sich bewegenden Roboter aufgenommen werden. In dem Fall kann es durchaus vorkommen, dass sich die Helligkeit ändert und damit dieser Farbraum für eine Separierung des Objekts vom Hintergrund eher ungeeignet ist.

2.1.2 HSV-Farbraum

Der HSV-Farbraum hingegen ist kein additiver Farbraum. Die Farbe ist ebenfalls durch drei Eigenschaften klassifiziert, dabei handelt es sich aber nicht um Farbanteile. Dieser Farbraum wird häufig bevorzugt, da er die Farbwahrnehmung des Menschen besser nachempfindet. Die Farbe wird anhand der drei Eigenschaften Farbton (engl. hue), Farbsättigung (engl. saturation) und Hellwert (engl. value) dargestellt. Dieser Farbraum wird in ähnlichen Definitionen auch als HSI-, HSL- oder HSB-Farbraum bezeichnet.

Die Komponenten des HSV-Farbraums werden meist auf zwei Arten dargestellt. Entweder als Zylinder, oder wie in Bild 2.3 als Kegel. Die Helligkeit bzw. der Hellwert wird durch die Höhe im Bereich von 0 bis 1 dargestellt. Die Sättigung entspricht dem Radius der Kegelunterseite mit dem selben Wertebereich wie die Helligkeit. Der Farbton hingegen wird durch einen Winkel von 0° bis 360° abgebildet.

Die Berechnung der Komponenten des HSV-Farbraums bzw. die Umrechnung aus dem RGB-Farbraum wird hier nicht erläutert. Dies folgt im weiteren Verlauf in Kapitel 6.3.

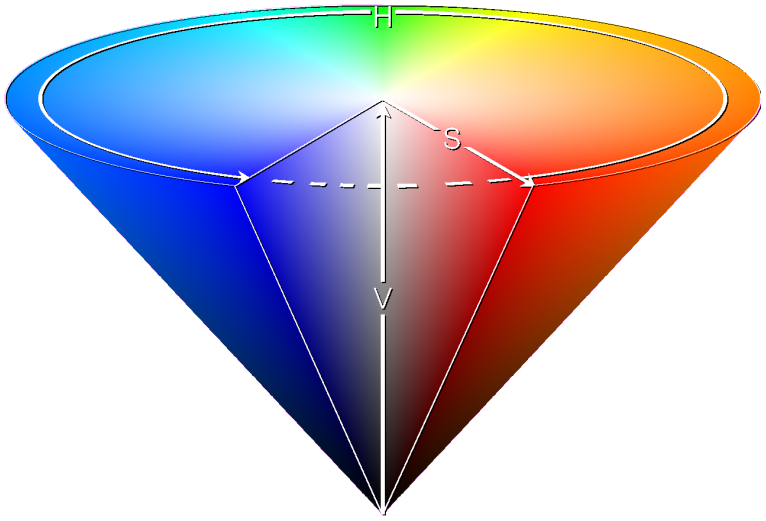


Bild 2.3: HSV-Farbraum als Kegel[2]

In der digitalen Bildverarbeitung werden einzelne Pixel eines Bildes meist als Byte gespeichert. Dadurch ist ein Wertebereich von 0 bis 255 gegeben. Im Falle des Farbtons muss dieser von 0° bis 360° auf 0 bis 255 abgebildet werden. Dies muss bei der Berechnung der Komponenten berücksichtigt werden.

Betrachtet man den Farbton bei einer bestimmten Sättigung und Helligkeit, so ergibt sich der in Bild 2.4 gezeigte Farbtonverlauf. In den Randbereichen des Farbton für 0° und 360° ist die Ähnlichkeit sehr groß, vom Zahlenwert her jedoch überhaupt nicht. Dieser Umstand ergibt einen besonderen Vorteil, der im folgenden Beispiel gezeigt wird.

Bild 2.4: Farbtonskala[2]

Konvertiert man Bild 2.1 mit Hilfe der MATLAB-Funktion *rgb2hsv* in den HSV-Farbraum, so ergeben sich die einzelnen Anteile wie Farbton, Sättigung und Hellwert wie in Bild 2.5 dargestellt.

(a) Farbton (b) Sättigung (c) Helligkeit

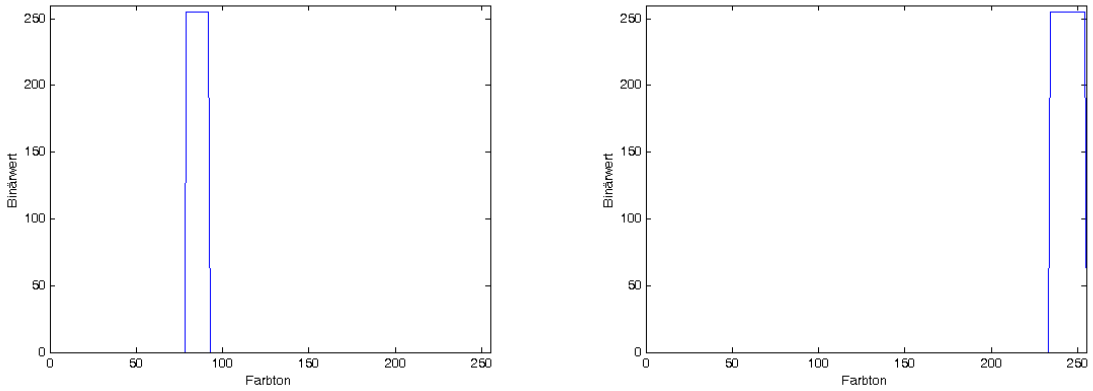
Bild 2.5: HSV-Komponenten

Auf der linken Seite ist der Farbton dargestellt (Bild 2.5(a)). Die Sättigung in Bild 2.5(b) und die Helligkeit in Bild 2.5(c) auf der rechten Seite sind jeweils als Grauwerte dargestellt. Je heller der Punkt, desto höher der jeweilige Wert. Die Helligkeit alleine betrachtet stellt ein Schwarzweißbild dar. Betrachtet man die Sättigung, so ist der Ball vom Hintergrund schon zu unterscheiden, es wird aber keine Aussage über die eigentliche Farbe getroffen.

In der Farbtonkomponente lässt sich der Ball sehr deutlich erkennen. Jedoch sind auch andere Teile des Bildes mit einem ähnlichen Farbton vorhanden. Dies sieht im ersten Moment nicht so gut aus, betrachtet man aber die Zahlenwerte des Farbtons, so hat der Ball sehr hohe Werte und die übrigen Bildpunkte mit ähnlichem Farbton eher niedrige Werte.

2.2 Farbklassifikation

Ziel ist es, alle Bildpunkte eines bestimmten Farbtons zu extrahieren. Eine Binarisierung mit einem Schwellwert ist dafür nicht ausreichend. Will man z.B. ein Objekt mit einen Farbton im Bereich von 110° bis 130° nach Bild 2.4 vom Hintergrund separieren, so müssen alle Bildpunkte, die nicht in diesem Bereich liegen, entfernt werden. Bild 2.6(a) zeigt die benötigte Übertragungsfunktion. Allen Bildpunkten, die innerhalb des Bereiches liegen, wird im Ergebnisbild der Grauwert 255 zugewiesen, die übrigen Bildpunkte werden auf 0 gesetzt.



(a) Fall 1 (b) Fall 2

Bild 2.6: Übertragungsfunktion

Für den hier vorliegenden Fall nach Bild 2.1 bzw. Bild 2.5 liegen die durch mehrere Bilder ermittelten optimalen Schwellwerte im Bereich von 325° und 360°, wie in Bild 2.6(b) dargestellt. Die Winkel für den Farbton wurden dabei bereits nach der Formel 2.1 auf den Wertebereich von 0 bis 255 umgerechnet.

$$\text{Grauwert} = 255 \cdot \frac{Winkel}{360°} \tag{2.1}$$

Die in Bild 2.6(b) gezeigte Übertragungsfunktion besitzt die Schwellwerte gemäß den Formeln 2.2 und 2.3.

$$\text{unterer Schwellwert} = 255 \cdot \frac{320°}{360°} = 230 \tag{2.2}$$

$$\text{oberer Schwellwert} = 255 \cdot \frac{360°}{360°} = 255 \tag{2.3}$$

Wendet man die Übertragungsfunktion mit den Schwellwerten 230 und 255 auf die Farbtonkomponente nach Bild 2.5 an, so ergibt sich Bild 2.7.

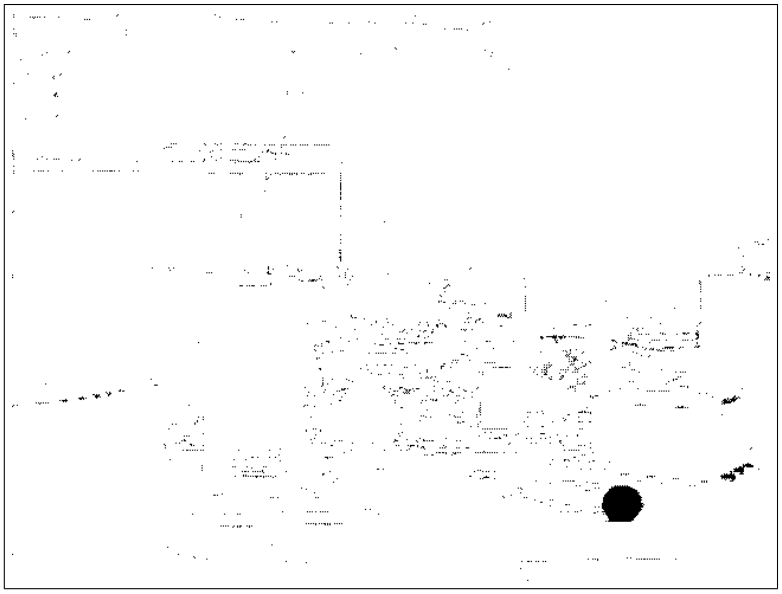


Bild 2.7: Ergebnis der Farbklassifikation

Das gesuchte Objekt ist deutlich zu erkennen, jedoch sind auch noch andere Bildpunkte vorhanden, die nicht zu dem Objekt gehören. Diese Störungen müssen noch beseitigt werden, um die Position des Objekts erkennen zu können. Im folgenden Unterkapitel werden die morphologischen Operationen vorgestellt, die für die Beseitigung dieser Störungen eingesetzt werden.

2.3 Morphologische Bildverarbeitung

Dieses Kapitel befasst sich mit der morphologischen Bildverarbeitung. Die große Stärke der Morphologie liegt in der Vielzahl der Möglichkeiten mit nur wenigen Grundoperationen. Mit geringem Aufwand sind sehr gute Ergebnisse für das Suchen von Mustern, Kantenfilterung, Verdünnungs/Verdickungsoperationen und vielen anderen Operationen möglich.

Ausgehend vom Ergebnis der Farbklassifikation aus Kapitel 2.2 folgt die weitere Vorgehensweise zur Extraktion des gesuchten Objekts vom Hintergrund anhand der verwendeten morphologischen Operationen. Hauptsächlich bestehen alle morphologischen Operationen aus den zwei Grundoperationen Erosion und Dilatation mit einem so genannten Strukturelement. Die beiden Grundoperationen und die sich damit ergebenden Möglichkeiten werden in den folgenden Unterkapiteln erläutert.

2.3.1 Erosion

Der Name Erosion stammt von dem lateinischen Begriff erodere und bedeutet übersetzt abnagen. Zur Ausführung der Operation wird ein Strukturelement herangezogen. Dabei kann es sich um ein beliebiges Element mit einer beliebigen Anzahl von Bildpunkten handeln. Dieses Strukturelement muss für den jeweiligen Anwendungszweck angepasst werden. Dadurch entstehen eine Vielzahl von Möglichkeiten. Bild 2.8 zeigt ein Beispiel für eine Erosion mit einem 3x3 Strukturelement, bei dem alle Bildpunkte gesetzt sind und der Ursprung in der Mitte liegt.

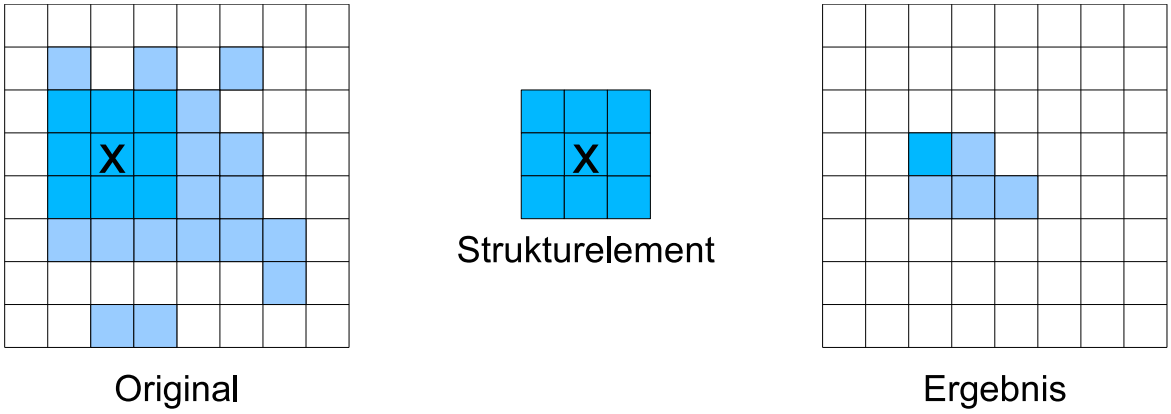


Bild 2.8: Beispiel für die Erosion mit einem 3x3 Strukturelement

Das Bild wird Pixel für Pixel mit dem Strukturelement bearbeitet. Passt das Strukturelement mit allen Bildpunkten auf das Original auf der linken Seite, so wird der Ursprung des Strukturelements im Ergebnisbild gesetzt. Zur Verdeutlichung ist das Vorgehen für einen Bildpunkt hervorgehoben. Für die Randbereiche des Bildes ist es dem Anwender freigestellt, wie diese aussehen. In den meisten Fällen nimmt man die Randpunkte als nicht gesetzt an und für den Fall, dass im Ergebnisbild ein Bildpunkt außerhalb des Bildes gesetzt werden sollte, wird dieser ignoriert.

Zusammenfassend ist zu sagen, dass mit Hilfe der Erosion Bildpunkte abgetragen werden. Mit einem entsprechenden Strukturelement kann dieses Abtragen so gesteuert werden, dass bei dem in Bild 2.8 vorliegenden Beispiel z.B. nur die Fransen des Objekts abgetragen werden. Mit dem gewählten Strukturelement werden aber weitaus mehr Bildpunkte entfernt als vielleicht erwünscht sind. Man müsste geeignete Strukturelemente finden, mit denen dies nicht geschieht. Dabei kann es passieren, dass das Bild mehrfach einer Erosion mit verschiedenen Strukturelementen unterzogen werden muss. Dies stellt natürlich einen entsprechend höheren Rechenaufwand dar. Als Alternative kann anschließend auch eine Dilatation durchgeführt werden.

2.3.2 Dilatation

Der Begriff Dilatation entstand, wie die Erosion, aus dem lateinischen Begriff dilatare und bedeutet soviel wie ausdehnen oder erweitern. Auch für diese Operation wird ein Strukturelement benötigt, welches die Erweiterung der Objekte im Binärbild steuert. Bild 2.9 zeigt ein Beispiel für eine Dilatation, ausgehend von dem Ergebnis der Erosion aus Bild 2.8 mit dem selben 3x3 Strukturelement.

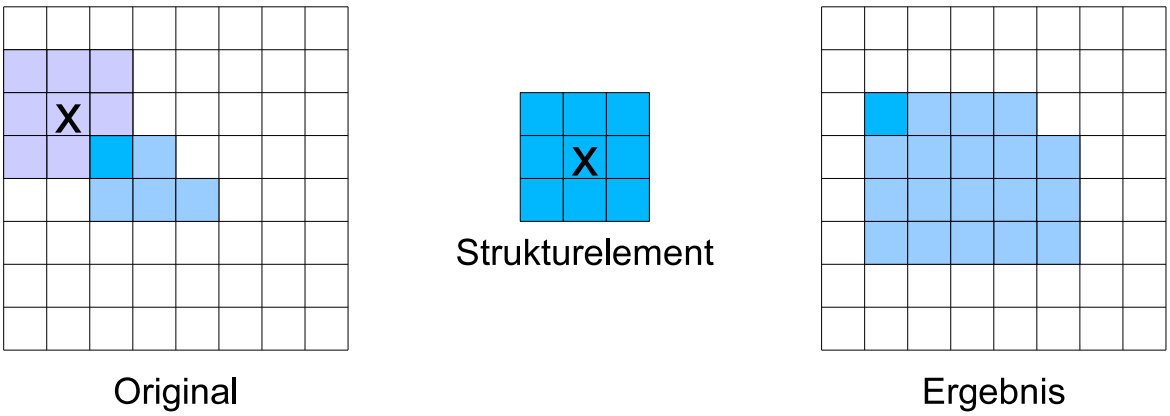


Bild 2.9: Beispiel für die Dilatation mit einem 3x3 Strukturelement

Wenn mindestens ein Bildpunkt des Originalbildes mit dem Strukturelement übereinstimmt, wird der Ursprung des Strukturelements im Ergebnisbild gesetzt. Dadurch wird das Objekt ausgedehnt. Vergleicht man nun das Ausgangsbild der Erosion (Bild 2.8) mit dem Ergebnis der Dilatation, so ist das Objekt erhalten geblieben und lediglich die Fransen sind abgetragen worden.

Der durch die Erosion aufgetretene Flächenverlust wurde für dieses Bild und Strukturelement vollständig kompensiert. Durch die Kombination von Erosion und Dilatation ist es also möglich, Störpixel aus Objekten bzw. aus Bildern zu entfernen. Die Anwendung einer Erosion gefolgt von einer Dilatation mit dem selben Strukturelement bezeichnet man in der Bildverarbeitung als Opening.

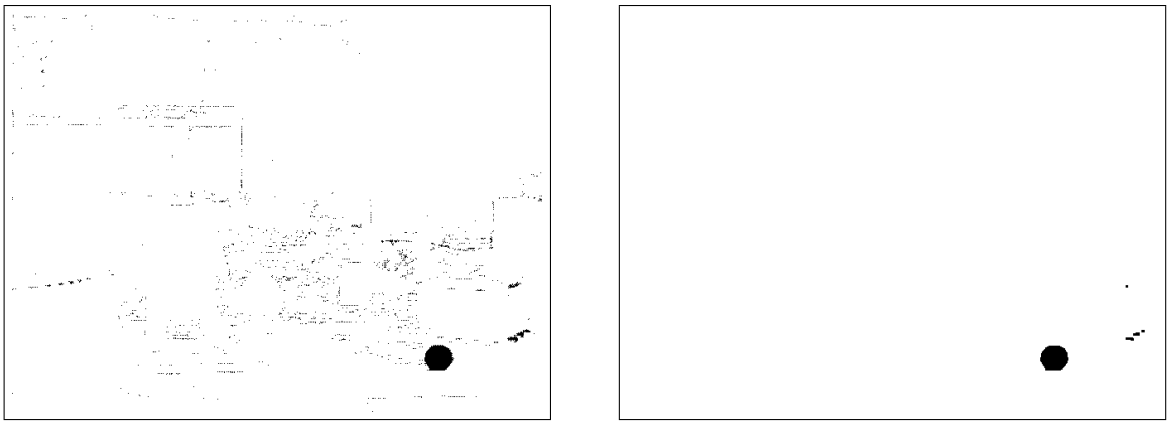
Das resultierende Ergebnis durch Anwendung der Dilatation auf das bereits bearbeitete Kamerabild (Bild 2.10(b)) ist im folgenden Abschnitt im Zusammenhang mit der Erläuterung des Openings zu finden.

2.3.3 Opening

Opening bezeichnet ein morphologisches Verfahren um Bildpunkte, die nicht zu einem gesuchten Objekt gehören zu entfernen. Auf das vorliegende Binärbild wird zunächst die Erosion und anschließend die Dilatation angewendet. Dabei wird für beide Operationen das selbe Strukturelement genutzt. Für das Strukturelement gilt die Bedingung, dass es die Form und mindestens die Größe der zu entfernenden Bildanteile besitzt. Objekte, die größer als das verwendete Strukturelement sind, werden durch die Erosion zwar verkleinert, aber durch die anschließende Dilatation nahezu rekonstruiert.

Vergleicht man das Eingangsbild (Bild 2.8) mit dem Ergebnis der Dilatation (Bild 2.9) so ist das Objekt originalgetreu erhalten geblieben und nur die Störpixel wurden entfernt. Bei dem hier verwendeten Strukturelement werden also alle Objekte, die aus mindestens 3x3 Bildpunkten bestehen erhalten und die übrigen entfernt.

Bezogen auf das Ergebnis der Farbklassifikation aus Bild 2.7 bzw. 2.10(a), unter Verwendung des 3x3 Operators aus dem vorangegangenen Beispiel, ergibt die Durchführung einer Erosion und anschließender Dilatation das in Bild 2.10(b) gezeigte Bild.



(a) Vorher (Bild 2.7) (b) Nachher

Bild 2.10: Ergebnis des Openings

Ein Großteil der Störungen bzw. Bildpunkte des selben Farbtons, die nicht zum gesuchten Objekt gehören, sind entfernt. Wie der Zufall so will, besitzt das Objekt im hier vorliegenden Bild eine geschlossene Fläche. Bei einer fortlaufenden Sequenz von Bildern kann es durch Rauschen durchaus vorkommen, dass das Objekt keine geschlossene Fläche mehr besitzt und einige Bildpunkte im Objekt sozusagen fehlen. Um dies zu kompensieren kommt das Gegenstück zum Opening, das Closing, zum Einsatz.

2.3.4 Closing

Das Closing wird wie das Opening ausgeführt, wobei die Reihenfolge der Erosion und Dilatation getauscht ist. Als erster Bearbeitungsschritt wird eine Dilatation durchgeführt. Das Ergebnis ist in Bild 2.11 zu sehen.

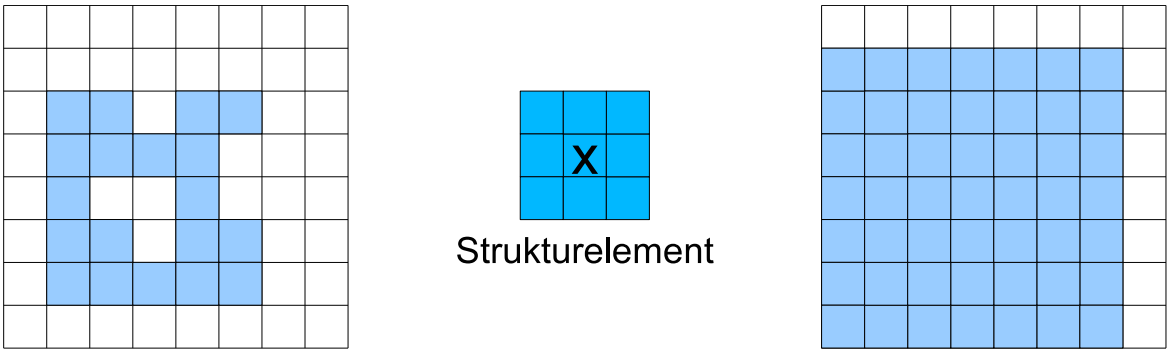


Bild 2.11: Dilatation mit einem 3x3 Strukturelement

Anschließend wird die Erosion mit dem selben Strukturelement auf das Ergebnis der Dilatation angewendet. Das Ergebnis in Bild 2.12 zeigt das Ergebnis des Closings. Die Löcher im Objekt sind geschlossen und die fehlenden Bildpunkte an der oberen und rechten Kante des Objekts werden ebenfalls hinzugefügt.

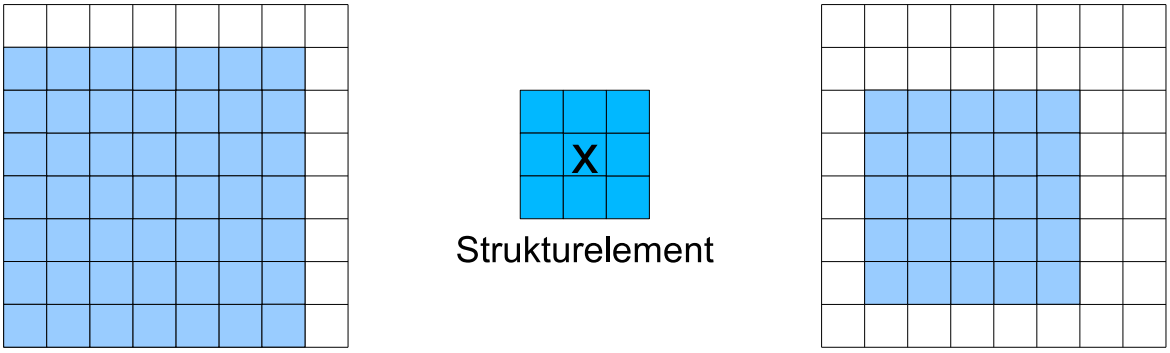


Bild 2.12: Erosion mit einem 3x3 Strukturelement

Eine Anwendung des Closings auf das Ergebnis der Erosion aus Bild 2.10(b) bringt in diesem Fall keine Veränderung, da das Objekt bereits eine geschlossene Fläche hat. Es handelt sich hierbei jedoch nur um einen Einzelfall und keine Bildsequenz, daher muss damit gerechnet werden, dass durch Rauschen Lücken im gesuchten Objekt auftreten.

2.4 Merkmalsextraktion

Dieses Kapitel befasst sich mit der Methode zur Berechnung charakteristischer Eigenschaften von Objekten in Binärbildern. Diese Methoden werden verwendet, um das im Bild enthaltene Objekt zu identifizieren. Es muss schließlich geklärt werden, ob das Objekt dem gesuchten entspricht.

2.4.1 Flächeninhalt

Der Flächeninhalt eines Objekts entspricht der Summe aller Bildpunkte des Objekts. Die Fläche wird dementsprechend in der Maßeinheit Bildpunkt angegeben. Je nach Kodierung des Binärbildes kann es nötig sein, dass das Ergebnis durch den Wert, durch den ein gesetzter Bildpunkt repräsentiert wird, geteilt werden muss. Sofern ein gesetzter Bildpunkt den Wert 1 hat, ist dies nicht erforderlich. Die allgemeine Formel zur Berechnung des Flächeninhaltes ist in Gleichung 2.4 dargestellt. Das verwendete Binärbild ist hier mit B bezeichnet und der Grauwert, durch den die gesetzten Bildpunkte repräsentiert werden mit G. Die Dimension des Bildes wird durch die Parameter L und R beschrieben.

$$A = \frac{1}{G} \sum_{y=0}^{R-1} \sum_{x=0}^{L-1} B(x, y) \tag{2.4}$$

Der Flächeninhalt des Objekts aus Bild 2.12 ist im Bild 2.13 dargestellt. Nach Gleichung 2.4 ergibt sich ein Flächeninhalt von 25 Bildpunkten mit L,R = 8 und G = 1.

	1	2	3	4	5		
	6	7	8	9	10		
	11	12	13	14	15		
	16	17	18	19	20		
	21	22	23	24	25		

Bild 2.13: Objektfläche

Da der Flächeninhalt allerdings abhängig von der Entfernung des Objekts zur Kamera ist, ist dieses Merkmal alleine betrachtet nicht sehr aussagefähig.

2.4.2 Umfang

Der Umfang eines Objekts in einem Binärbild verhält sich ähnlich wie der Flächeninhalt. Er ist abhängig von der Entfernung des gesuchten Objekts zur Kamera. Die Berechnung erfolgt, indem für jeden Bildpunkt des Objekts überprüft wird, ob die vier oder acht Nachbarbildpunkte ebenfalls zum Objekt gehören. Es werden dann nur die Bildpunkte aufsummiert, bei denen diese Bedingung nicht zutrifft.

Die vierer Nachbarschaft bezeichnet diejenigen Bildpunkte, die sich in Richtung der vier Himmelsrichtungen um den zu untersuchenden Punkt gruppieren. Bei der achter Nachbarschaft werden auch die diagonalen Bildpunkte mit berücksichtigt.

In Bild 2.14 ist der Umfang des Objekts dunkler dargestellt. Zählt man die Bildpunkte zusammen, so erhält man einen Umfang von 16 Bildpunkten. Bei einer Kantenlänge von 5 Bildpunkten würde man eigentlich einen Umfang von 20 Bildpunkten erwarten. Diese Abweichung kommt durch die Quantisierung zustande, die durch die Bildpunkte hervorgerufen wird. Bei Objekten mit einer deutlich größeren Anzahl an Bildpunkten wird dieser Quantisierungsfehler jedoch deutlich kleiner. Angenommen das Objekt hat einen Kantenlänge von 100 Bildpunkten, dann Entspricht der Umfang 396 Bildpunkten, was in etwa den erwarteten 400 Bildpunkten entspricht. Die Genauigkeit ist demnach proportional zu der Objektgröße bzw. Auflösung der Kamera.

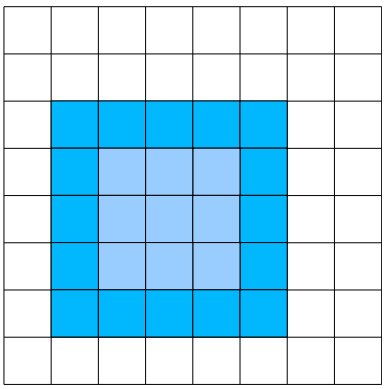


Bild 2.14: Objektumfang

Anhand des Flächeninhaltes nach Kapitel 2.4.1 und des Umfangs können Aussagen über die Größe des Objekt gemacht werden. Aber wie bereits erwähnt, sind diese beiden Eigenschaften von der Entfernung des Objekts abhängig und daher für sich betrachtet nicht sehr aussagekräftig. Kombiniert man diese Eigenschaften miteinander, so besteht die Möglichkeit, ein entfernungs- und damit größenunabhängiges Merkmal des Objekts zu bestimmen. Dieses Merkmal wird als Kompaktheit bezeichnet und im folgenden Unterkapitel erläutert.

2.4.3 Kompaktheit

Die Kompaktheit gibt eine Aussage über die Form des Objekts. Zur Berechnung der Kompaktheit nach Gleichung 2.5 wird der Umfang und der Flächeninhalt benötigt. Den niedrigsten Wert für die Kompaktheit beschreibt einen Kreis, da dort das Verhältnis von Umfang zu Flächeninhalt am kleinsten ist.

$$V = \frac{U^2}{A} \tag{2.5}$$

Setzt man die allgemeinen Formeln für die Berechnung des Umfangs und des Flächeninhaltes einen Kreises in Gleichung 2.5 ein, so erhält man den Wert 4π unabhängig vom Radius des Kreises. Über die Kompaktheit kann man also eine Aussage über die Form des Objekts treffen. Ist die Kompaktheit im Bereich von 4π, so ist das Objekt kreisförmig. Aufgrund dieses Merkmales kann entschieden werden, ob es sich bei einem Objekt um den gesuchten Ball handelt oder nicht. Je weiter die Objektform von einem Kreis abweicht, desto Größer ist Kompaktheit.

Angenommen, in einem Binärbild ist ein Quadrat mit einer Kantenlänge von 100 Bildpunkten enthalten. Dann ist der Umfang 396 Bildpunkte und der Flächeninhalt entspricht 10000 Bildpunkten. Nach Gleichung 2.5 ist die Kompaktheit für dieses Objekt 15,68. Ein ideales Quadrat hätte den Wert 16. Dieser ist aufgrund der Quantisierung durch die einzelnen Bildpunkte jedoch nur schwer zu erreichen.

2.4.4 Massenschwerpunkt

Um die Koordinaten der Position des gesuchten Objekts zu beschreiben, eignet sich der Mittelpunkt am besten. Dieser lässt sich sehr leicht ermitteln, indem man die Ausdehnung aller gesetzten Pixel im Binärbild (Bild 2.10(b)) bestimmt. Das Problem, das dabei auftritt, ist in Bild 2.15 dargestellt.

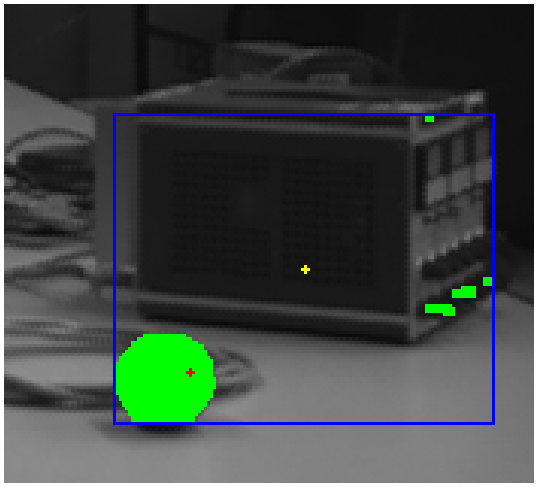


Bild 2.15: Massenschwerpunkt

Es handelt sich hierbei um einen Ausschnitt aus Bild 2.1 aus Kapitel 2.1. Die grün markierten Bildpunkte stellen dabei alle Bildpunkte dar, die nach der Durchführung des Closing übrig geblieben sind. Die maximale Ausdehnung über alle gesetzten Bildpunkte ist durch das blaue Rechteck markiert. Errechnet man anhand der Ausdehnung dieses Rechtecks den Mittelpunkt, so landet man an der Position des gelben Kreuzes. Dabei handelt es sich um den Mittelpunkt über alle gesetzten Bildpunkte des Binärbildes nach Abschluss des Closings.

Wie zu erkennen ist, liegt dieser aufgrund einiger falsch erkannter Bildpunkte außerhalb des gesuchten Objekts. Um diesen Fehler zu kompensieren wird anstatt des Mittelpunktes der Massenschwerpunkt mit den Gleichungen 2.6 und 2.7 bestimmt. Es werden alle x-Koordinaten und alle y-Koordinaten einzeln aufsummiert und durch den Flächeninhalt geteilt. Man erhält so den Mittelwert der x-Koordinaten und y-Koordinaten über alle gesetzten Bildpunkte.

$$x_m = \frac{1}{A} \sum_i x_i \tag{2.6}$$

$$y_m = \frac{1}{A} \sum_i y_i \tag{2.7}$$

Der Massenschwerpunkt ist im Bild 2.15 durch das rote Kreuz dargestellt. Trotz der breits angesprochenen fehlerhaft erkannten Bildpunkte liegt der Massenschwerpunkt immer noch im Objekt. Dank dieser Methode steht also eine robuste Möglichkeit zur Bestimmung des Mittelpunktes zur Verfügung, sofern die Anzahl der Bildpunkte des gesuchten Objekts deutlich größer ist, als die Anzahl der fehlerhaft erkannten Bildpunkte.

2.5 Stereovision

Eine Kamera liefert ein zweidimensionales Bild einer dreidimensionalen Szene. Dabei geht jegliche Information über die Entfernung verloren. In Bild 2.16 ist das Problem dargestellt. Der Punkt P ist im Raum durch seine Koordinaten X, Y und Z definiert. Dieser Punkt wird über die Optik der Kamera auf die Sensorfläche im Abstand f (Brennweite) abgebildet. Der Punkt p ist die Projektion des Punktes P, wobei dieser nur noch durch die Koordinaten x und y repräsentiert wird. Die Information der Entfernung Z ist nicht mehr vorhanden.

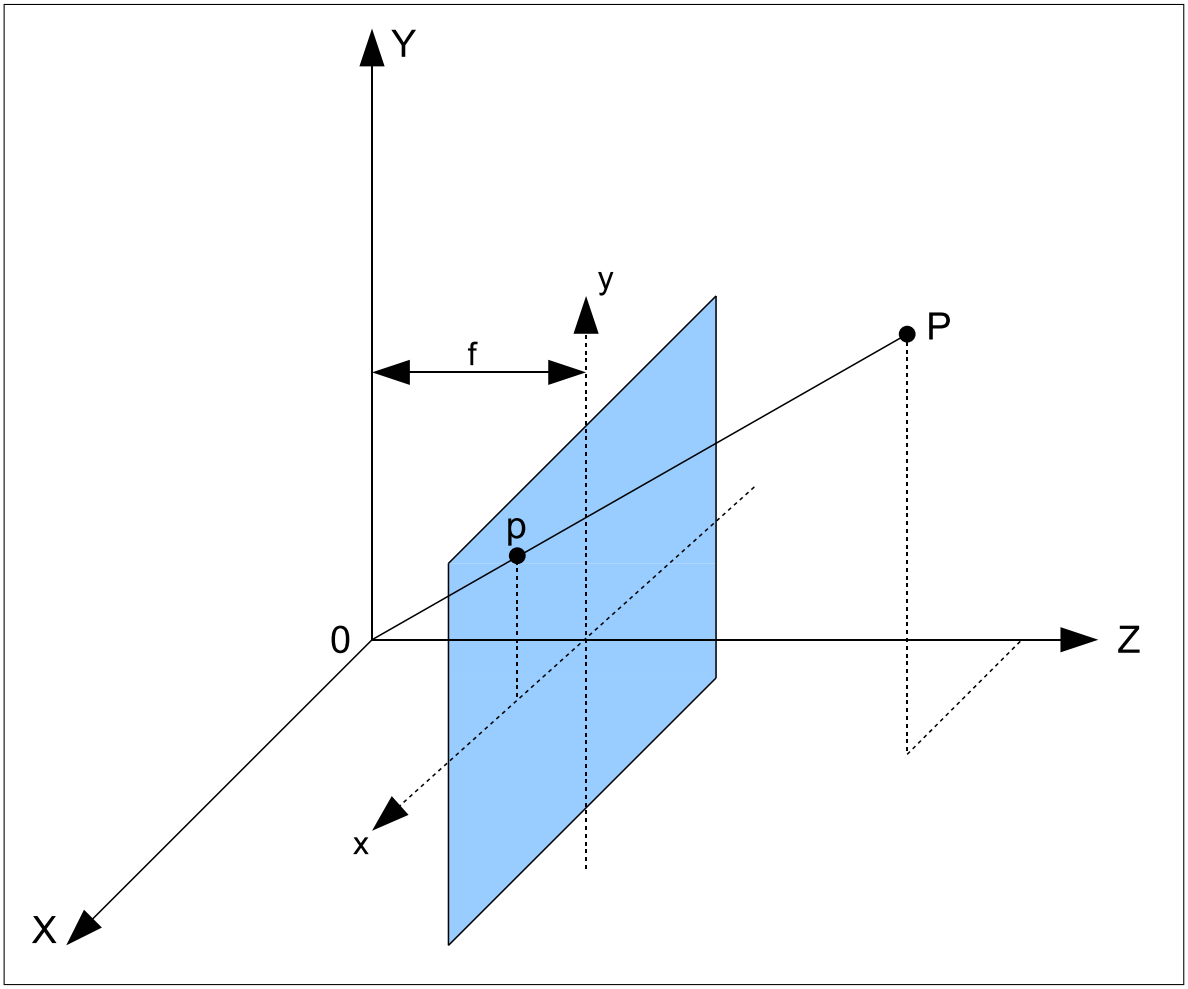
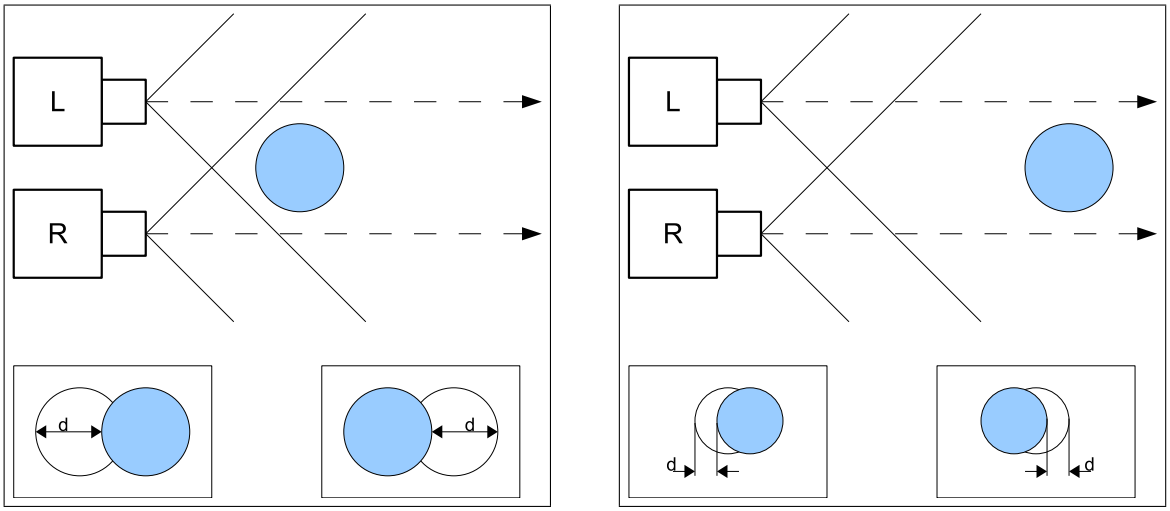


Bild 2.16: Kameraprojektion

Durch den Einsatz von zwei achsparallelen Kameras kann diese verloren gegangene Entfernung wiederhergestellt werden. Durch den Abstand der Kameras zueinander ist das Objekt in den jeweiligen Bildern verschoben. Diese Verschiebung wird auch als Disparität bezeichnet. In Bild 2.17 ist die Disparität exemplarisch dargestellt. Mit zunehmender Entfernung sinkt die Disparität. Demnach ist der Abstand umgekehrt proportional zur Disparirät.



(a) Nah (b) Fern

Bild 2.17: Disparität

Zur Berechnung der Entfernung Z anhand der Disparität d, ist es nötig einige Systemparameter zu kennen. Der Abstand b der Kameras zueinander, die Brennweite f der Optik, sowie der Pixelabstand muss bekannt sein. Wenn diese Parameter vorhanden sind, kann durch Anwendung des Strahlensatzes die Gleichung 2.8 aufgestellt werden.

$$\frac{Z}{f} = \frac{b}{d} \tag{2.8}$$

Aus der Stahlensatzgleichung ergibt sich durch entsprechende Umformungen die Gleichung 2.9 für die Entfernung.

$$Z = \frac{b \cdot f}{d} \tag{2.9}$$

Anhand der ermittelten Massenschwerpunkte des gesuchten Objekts kann nun durch den Versatz in horizontaler Richtung die Disparität berechnet werden. Damit die Entfernung Z in Metern angegeben werden kann, muss die Disparität zuvor von der Einheit Bildpunkte in Meter umgerechnet werden. Dafür wird die Angabe der Pixelgröße bzw. des Pixelabstandes benötigt.

Anhand der berechneten Entfernung Z können nun die noch fehlenden beiden Koordinaten für eine Ortsbestimmung im Raum berechnet werden. Bild 2.18 zeigt die Projektion in Y-Richtung. Ein Objekt, welches sich vertikal im Abstand Y' von der optischen Achse einer Kamera befindet, ist im Bild y Bildpunkte vertikal von der Bildmitte entfernt. Anhand des Massenschwerpunktes des Objekts im Bild einer Kamera kann die Verschiebung in Y-Richtung vom Mittelpunkt durch Subtraktion von der halben Bildhöhe ermittelt werden.

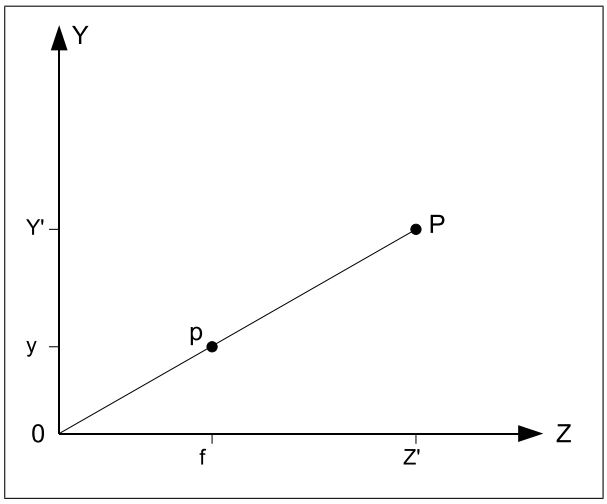


Bild 2.18: Kameraprojektion in Y-Richtung

Anhang der Zeichnung kann mit Hilfe des Strahlensatzes die Gleichung 2.10 aufgestellt werden.

$$\frac{Y'}{Z'} = \frac{y}{f} \tag{2.10}$$

Die einzige Unbekannte in dieser Gleichung ist die gesuchte Y-Koordinate. Nach entsprechender Umformung ergibt sich für die Y-Koordinate die Gleichung 2.11 und analog dazu für die X-Koordinate die Gleichung 2.12.

$$Y = \frac{Z \cdot y}{f} \tag{2.11}$$

$$X = \frac{Z \cdot x}{f} \tag{2.12}$$

Die ermittelten Koordinaten x und y von einer Kamera müssen ebenfalls wie die Disparität von der Einheit Bildpunkte in Meter umgerechnet werden, damit die Einheiten der Gleichungen übereinstimmen.

Die Implementierung der Berechnung und der verwendeten Parameter werden im Kapitel 6.6 erläutert.

3 Hardware

Nachdem die Theorie zur Bildverarbeitung geklärt ist, befasst sich dieses Kapitel mit der Hardware, die für die Umsetzung der Aufgabe verwendet wird. Angefangen mit der DSP-Plattform und deren Peripherie bis hin zur Adapterplatine DSKEye. Aufgrund der Komplexität der Komponenten werden nur die für diese Arbeit wesentlichen Aspekte und Eigenschaften erläutert.

3.1 DSK6416 Board

Die verwendete DSP-Plattform ist in Bild 3.1 gezeigt. Es handelt sich um ein Entwicklungsboard der Firma Spectrum Digital. Als Digitaler Signal Prozessor (DSP) kommt darauf ein TSM320C6416T der Firma Texas Instruments zum Einsatz. Darüber hinaus verfügt dieses Board über weitere Hardware, wie ein CPLD, SDRAM und FLASH Speicher, LEDs, Schalter und Erweiterungsanschlüsse.

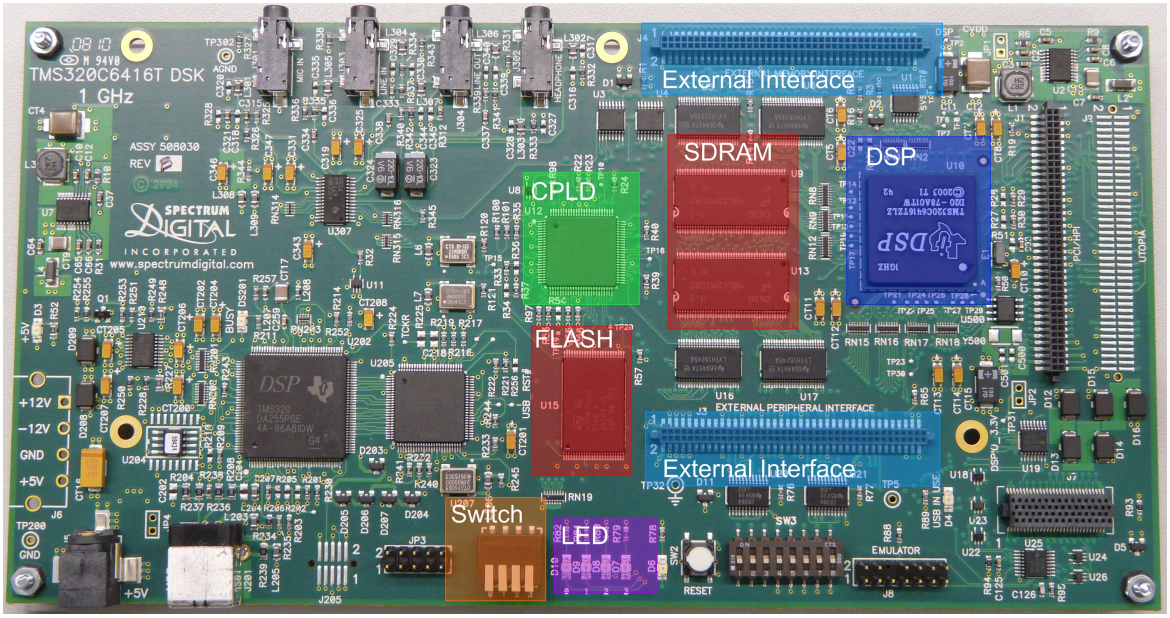


Bild 3.1: DSK6416 Board

3.1.1 Digitaler Signal Prozessor

Der auf dem Entwicklungsboard befindliche DSP ist ein TMS320C6416T, der mit einem maximalen Systemtakt von 1 GHz betrieben wird. Bild 3.2 zeigt den internen Aufbau des Prozessors. Der Prozessor verfügt über jeweils einen 16 KB großen Level 1 Cache für Instruktionen und Daten. Diese sind wiederum mit dem 1024 KB großen Level 2 Speicher verbunden. Dieser interne Speicher wird sowohl für das Programm als auch die Daten genutzt. Außerdem kann ein Teil dieses Speichers als Level 2 Cache verwendet werden. Die vorhandene Peripherie ist über den Speichercontroller (Enhanced DMA Kontroller) mit dem Level 2 Speicher verbunden. Die genaue Funktionsweise des EDMA Kontrollers ist in Kapitel 3.1.4 beschrieben.

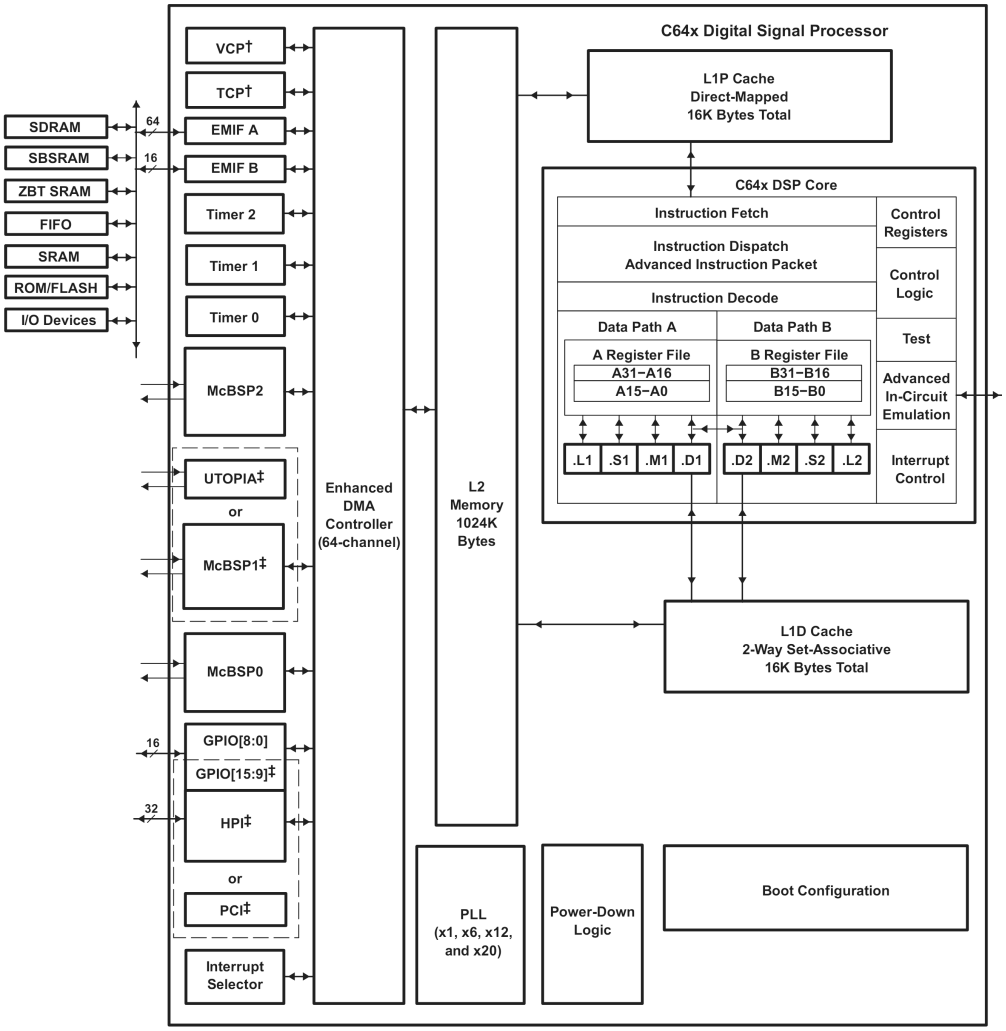


Bild 3.2: Internet Aufbau des DSP[3]

Bei dem Prozessor handelt es sich um eine C64x CPU mit einer „VelociTI.2“ Architektur. In Bild 3.3 ist der Aufbau der CPU dargestellt. Bei den gelb eingefärbten Objekten handelt es sich um Erweiterungen im Vergleich zur vorherigen „VelociTI Advanced Very-Long-Instruction-Word“ Architektur. Die Instruktionen für den Prozessor sind dabei in 256 Bit breite Instruktionspakete gepackt. Der Prozessor kann diese Pakete in einem Takt laden (Fetch Packet, FP). Darüber hinaus können diese Instruktionspakete auch parallel ausgeführt werden (Execute Packet, EP). Dafür muss in den einzelnen 32 Bit Instruktionen jeweils ein entsprechendes Bit gesetzt werden.

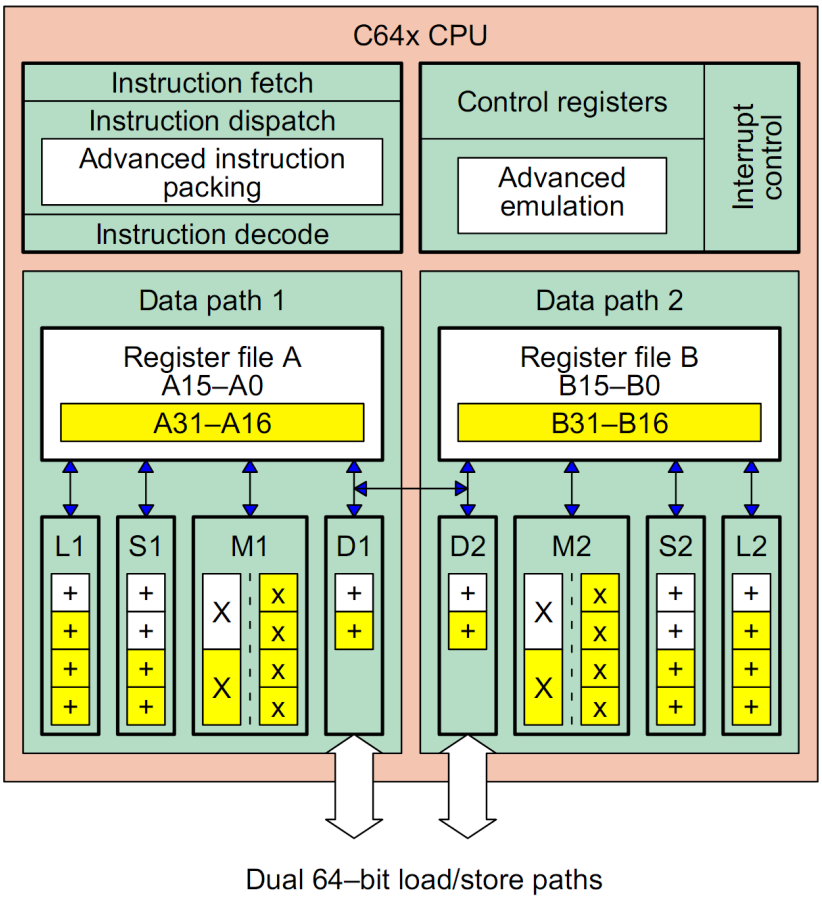


Bild 3.3: Aufbau der C64x CPU[4]

Die CPU hat insgesamt acht Hardwareeinheiten, die auf zwei unabhängige Datenpfade aufgeteilt sind. In jedem Datenpfad ist ein Registerfile mit 32 Registern enthalten, die jeweils 32 Bit breit sind. Die Instruktionen werden in den vier Hardwareeinheiten L, S, M und D ausgeführt und die Ergebnisse in die Register geschrieben. Diese vier verschiedenen Einheiten sind zur Ausführung unterschiedlicher Instruktionen vorhanden, wobei es auch Überschneidungen beim Funktionsumfang gibt.

Die L-Einheit ist für logische und arithmetische Operationen wie Exklusiv-Oder, Und, Addition, Subtraktion usw. konzipiert. Darüber hinaus kann diese auch Vergleichsoperationen durchführen, um nur einige Beispiele zu nennen.

Die S-Einheit ist der L-Einheit recht ähnlich. Sie verfügt teilweise über die selben, aber auch über andere logische und arithmetische Operationen. Außerdem kann diese Einheit auch noch weitere Vergleichsoperationen ausführen. Darüber hinaus hat die S-Einheit noch eine besondere Funktion. Die S-Einheit ist als einzige in der Lage, Sprungoperationen auszuführen, die für Funktionsaufrufe benötigt werden.

Die M-Einheit kann ausschließlich Multiplikationen durchführen. Dabei gibt es aber auch eine Vielzahl von Möglichkeiten. Durch die neue Architektur ist die CPU in der Lage gepackte Daten zu verarbeiten. Daher kann die M-Einheit anstatt einer 32x32 Bit auch zwei 16x16 Bit oder sogar vier 8x8 Bit Multiplikationen mit einer Instruktion durchführen.

Zum Schluss wäre da noch die D-Einheit, die als einzige auf den Speicher zugreifen kann. Darüber hinaus kann diese Einheit ebenfalls ein paar wenige logische Operationen. Der Zugriff auf den Speicher kann wahlweise in beide Richtungen mit 8, 16, 32 oder 64 Bit erfolgen.

Bild 3.4 zeigt einen Ausschnitt der internen Verschaltung zwischen Registerfile und Hardwareeinheiten. Dabei ist zu erkennen, dass die beiden Datenpfade über zwei sogenannte Crosspath miteinander verbunden sind. Dadurch kann z.B. die M-Einheit im Datenpfad A einen Operanden aus dem Registerfile B und umgekehrt verwenden.

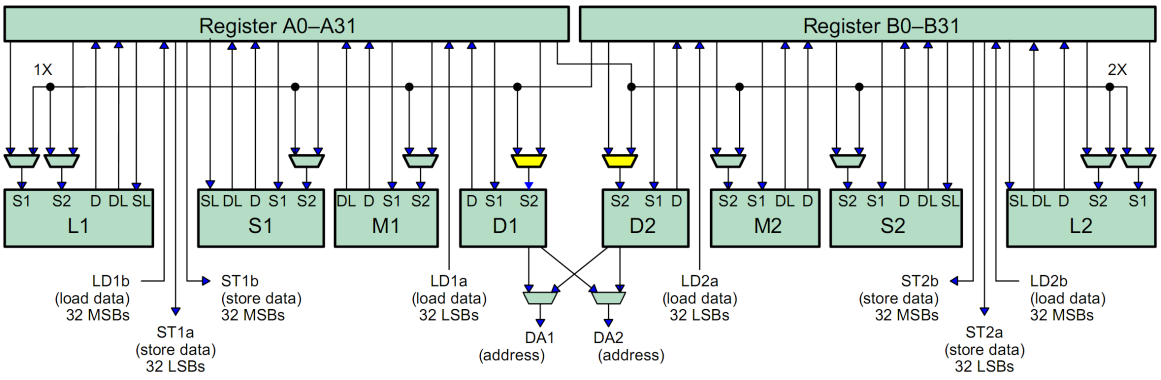


Bild 3.4: Interner Aufbau des C64x Datenpfads[4]

Da manche Instruktionen mehr als einen Takt für die Ausführung benötigen, bedarf das Pipelining besonderer Aufmerksamkeit. Genaueres dazu ist in Kapitel 6 zu finden.

3.1.2 DSP/BIOS

Das von Texas Instruments für diese DSP Familie entwickelte DSP/BIOS stellt einen Echtzeit-Kernel zur Verfügung, um komplexe Programmabläufe zu realisieren. Die immer weiter steigenden Anforderungen an Programme verlangen eine einfache Schnittstelle zur Hardware, um die Entwicklung deutlich zu vereinfachen. Dazu stellt das DSP/BIOS eine Reihe von Funktionen und Schnittstellen zur Verfügung, die in ähnlicher Weise auch auf anderen Plattformen vorhanden sind. Dadurch bekommen Entwickler die Möglichkeit, hardwareunabhängig Programme zu entwickeln.

Der größte Vorteil dieses DSP/BIOS ist die Fähigkeit mehrere Tasks parallel auszuführen, wobei immer nur ein Task zur Zeit den Prozessor belegt. Die aktuelle Zuteilung des Prozessors wird dabei in definierten periodischen Zeitabständen durch den Scheduler des Betriebssystems überprüft. Jedem Task wird bei der Erstellung eine von 15 Prioritätsstufen zugewiesen, die die Reihenfolge der Abarbeitung festlegt.

Bild 3.5 zeigt die verschiedenen Zustände, die ein Task einnehmen kann. Befinden sich z.B. zwei Tasks mit unterschiedlicher Priorität im Zustand *TSK_READY*, so wird der Task mit der höheren Priorität als erstes in den Zustand *TSK_RUNNING* versetzt und ausgeführt. Nachdem die Ausführung abgeschlossen ist, wird anschließend der Task mit der niedrigeren Priorität ausgeführt.

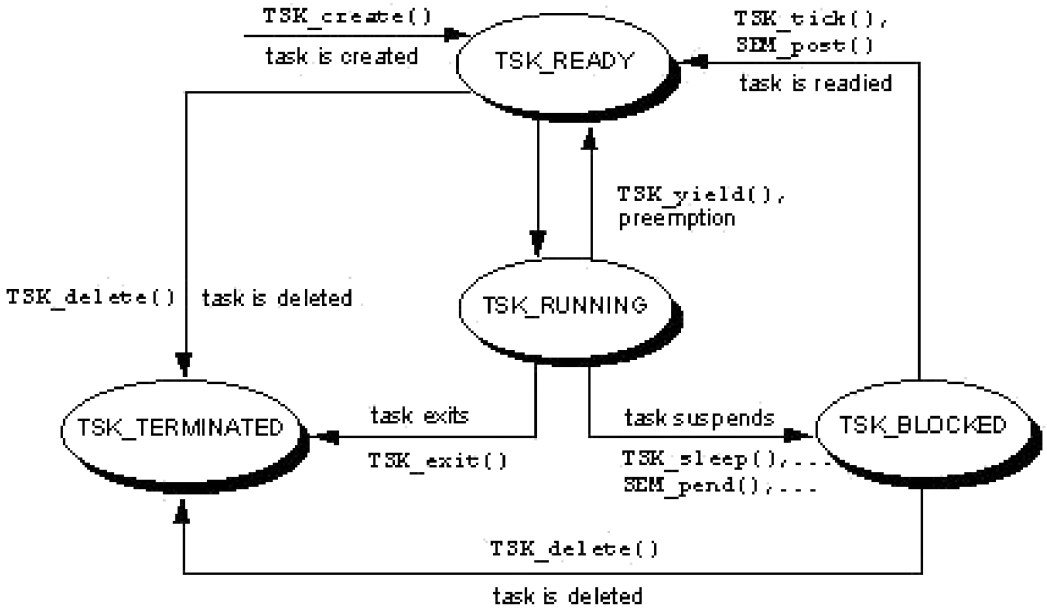


Bild 3.5: DSP/Bios Task Zustände[5]

Ein wichtiges Werkzeug im Zusammenhang mit Mehrprozessumgebungen sind Semaphoren. Diese sind zur Steuerung bzw. Synchronisation verschiedener Tasks untereinander zuständig. Wenn ein Task durch Aufruf der Funktion *SEM_pend()* auf eine Semaphore wartet, so befindet sich dieser im Zustand *TSK_BLOCKED* und wird nicht mehr ausgeführt. Erst wenn die Sempahore durch einen anderen Task gesetzt wird oder eine Zeitüberschreitung eintritt, wechselt der Task wieder in den Zustand *TSK_READY*.

Darüber hinaus gibt es noch Software-Interrupts, die den regulären Programmablauf unterbrechen können. Unter anderem werden diese Ereignisse von Treibern für Peripherie verwendet, wenn z.B. neue Daten empfangen wurden. Diese Ereignisse sind ebenfalls in 15 Prioriätsstufen eingeteilt. Im Vergleich zur Taskpriorität ist diese aber höher angesiedelt. Ein Software-Interrupt mit der niedrigsten Priorität ist einem Task mit der höchsten Priorität übergeordnet.

Als letzten Punkt wären dann noch die Hardware-Interrupts zu nennen, deren Priorität noch oberhalb der Software-Interrupts liegt. Der Prozessor verfügt über insgesamt 16 Interrupts, die im Zusammenhang mit dem Interruptcontroller erläutert werden. Bei Hardware- und Software-Interrupts gibt es keine Zustände wie bei den Tasks. Tritt ein Interrupt auf, wird die entsprechende Interruptroutine ohne Unterbrechung ausgeführt. Sollten während dieser Ausführungszeit weitere Interrupts auftreten, so werden diese anschließend in der Reihenfolge ihrer Priorität ausgeführt.

3.1.3 Interruptcontroller

Der Interruptcontroller ist für die Verarbeitung der Hardware-Interrupts zuständig. Tritt ein Interrupt auf, so wird der Prozessor dazu veranlasst, die zugehörige Interruptroutine auszuführen. In Bild 3.6 sind alle möglichen Interruptquellen angegeben. Insgesamt verfügt der Prozessor über 16 Hardware-Interrupts und 30 Interruptquellen.

Die ersten 4 Interrupts sind dabei fest definiert und können nicht geändert werden. Die anderen Interrupts können mit anderen Ereignissen verknüpft werden. Über die beiden Register *MUXH* und *MUXL* können die Hardware-Interrupts 4 bis 15 mit den 30 übrigen Interrruptquellen verbunden werden. Durch die veränderbare Zuordnung der Interruptquellen besteht die Möglichkeit, die Prioritäten verschiedener Ereignisse den Bedürfnissen des Programms anzupassen.

Bei den externen Interrupts muss dazu noch die Flanke, bei der ein Interrupt ausgelöst wird, eingestellt werden. Dies wird bei der DSP/BIOS Konfiguration berücksichtigt. Außerdem ist zu beachten, dass das Signal am externen Eingang für mindestens 4 Takte nach einem Flankenwechsel stabil anliegt, damit der Schaltvorgang richtig erkannt wird[3].

CPU INTERRUPT NUMBER	INTERRUPT SELECTOR CONTROL REGISTER	SELECTOR VALUE (BINARY)	INTERRUPT EVENT	INTERRUPT SOURCE
INT_00†	–	–	RESET	
INT_01†	–	–	NMI	
INT_02†	–	–	Reserved	Reserved. Do not use.
INT_03†	–	–	Reserved	Reserved. Do not use.
INT_04‡	MUXL[4:0]	00100	GPINT4/EXT_INT4	GPIO interrupt 4/External interrupt pin 4
INT_05‡	MUXL[9:5]	00101	GPINT5/EXT_INT5	GPIO interrupt 5/External interrupt pin 5
INT_06‡	MUXL[14:10]	00110	GPINT6/EXT_INT6	GPIO interrupt 6/External interrupt pin 6
INT_07‡	MUXL[20:16]	00111	GPINT7/EXT_INT7	GPIO interrupt 7/External interrupt pin 7
INT_08‡	MUXL[25:21]	01000	EDMA_INT	EDMA channel (0 through 63) interrupt
INT_09‡	MUXL[30:26]	01001	EMU_DTDMA	EMU DTDMA
INT_10‡	MUXH[4:0]	00011	SD_INTA	EMIFA SDRAM timer interrupt
INT_11‡	MUXH[9:5]	01010	EMU_RTDXRX	EMU real-time data exchange (RTDX) receive
INT_12‡	MUXH[14:10]	01011	EMU_RTDXTX	EMU RTDX transmit
INT_13‡	MUXH[20:16]	00000	DSP_INT	HPI/PCI-to-DSP interrupt (PCI supported on C6415T and C6416T)
INT_14‡	MUXH[25:21]	00001	TINT0	Timer 0 interrupt
INT_15‡	MUXH[30:26]	00010	TINT1	Timer 1 interrupt
–	–	01100	XINT0	McBSP0 transmit interrupt
–	–	01101	RINT0	McBSP0 receive interrupt
–	–	01110	XINT1	McBSP1 transmit interrupt
–	–	01111	RINT1	McBSP1 receive interrupt
–	–	10000	GPINT0	GPIO interrupt 0
–	–	10001	XINT2	McBSP2 transmit interrupt
–	–	10010	RINT2	McBSP2 receive interrupt
–	–	10011	TINT2	Timer 2 interrupt
–	–	10100	SD_INTB	EMIFB SDRAM timer interrupt
–	–	10101	Reserved	Reserved. Do not use.
–	–	10110	Reserved	Reserved. Do not use.
–	–	10111	UINT	UTOPIA interrupt (C6415T/C6416T only)
–	–	11000 – 11101	Reserved	Reserved. Do not use.
–	–	11110	VCPINT	VCP interrupt (C6416T only)
–	–	11111	TCPINT	TCP interrupt (C6416T only)

Bild 3.6: Interrupttabelle[3]

In dieser Arbeit kommen unter Anderem die Interrupts 4 bis 6 für die Kameras, Interrupt 8 für EDMA und die beiden Interrupts XINT2 und RINT2 für McBSP2 zum Einsatz. Die letzten beiden Interrupts müssen entsprechend mit noch nicht verwendeten Hardware-Interrupts verbunden werden.

3.1.4 Enhanced Direct Memory Access

Beim Enhanced Direct Memory Access (EDMA) Kontroller handelt es sich um eine Komponente, die im DSP dafür sorgt, dass der Prozessor entlastet wird. Neben der steigenden Komplexität der Aufgaben, die ein DSP auszuführen hat, steigen meistens auch die zu verarbeitenden Datenmengen. Der EDMA Kontroller ist dafür zuständig, die Bereitstellung der Daten zu vereinfachen.

Ohne den EDMA Kontroller müsste der Prozessor jeden Datentransfer von der Peripherie selbst ausführen. Die damit verbundene CPU-Zeit kann unter Umständen dazu führen, dass der Prozessor gar keine Zeit mehr hat, um die eigentlichen Berechnungen auszuführen. An dieser Stelle kommt der EDMA Kontroller zur Hilfe.

Der EDMA Kontroller im hier verwendeten DSP hat insgesamt 64 voneinander unabhängige Kanäle. Bei dieser Arbeit wird der EDMA Kontroller unter Anderem für den Transfer der Bilddaten der Kameras in den Speicher verwendet. Von der Adapterplatine werden die Bilder der Kameras zeilenweise übertragen. Dem EDMA Kontroller wird durch ein Interruptsignal das Vorhandensein einer neuen Bildzeile mitgeteilt und dieser beginnt dann über das External Memory Interface (EMIF) die Daten zu lesen und in den Speicher zu kopieren. Der EDMA Kontroller wird so konfiguriert, dass er dem Prozessor durch den EDMA Interrupt (INT08, siehe Bild 3.6) mitgeteilt wird, wenn ein neues Bild komplett in den Speicher geschrieben wurde. Erst jetzt muss die CPU aktiv werden und kann die Daten verarbeiten. Durch diese Entlastung steht deutlich mehr CPU-Zeit für die eigentliche Datenverarbeitung zur Verfügung.

Bei der Konfiguration werden Angaben wie Quell- und Zieladresse, sowie die Datenmenge benötigt. Diese Daten werden in ein sogenanntes „Parameter RAM“ für den entsprechenden EDMA Kanal eingetragen. Der Transfer der Daten kann je nach Anwendung wahlweise über einen Interrupt oder manuell durch die CPU gestartet werden.

Der EDMA Kontroller verfügt außerdem über die Möglichkeit fortlaufende Datentransfers zu verarbeiten. Das kommt u.A. bei Kameras vor, die mit einer bestimmten Bildwiederholungsrate arbeiten. Dazu können in dem „Parameter RAM“ mehrere Konfigurationen miteinander verkettet werden. Im Fall der Kamera, die immer fortlaufend die selbe Datenmenge liefert, kann die Konfiguration mit sich selbst verkettet werden. Dadurch wird die Konfiguration des EDMA Kontrollers für diesen Kanal automatisch neu geladen und muss nicht nach jedem Bild durch den Prozessor aktualisiert werden.

Wie die Konfiguration im Detail von statten geht und was dabei zu beachten ist, wird im Rahmen der Softwarekonfiguration in Kapitel 5.2.5 erläutert.

3.1.5 External Memory Interface

Das External Memory Interface ist eine parallele Schnittstelle des Prozessors. Der DSP besitzt insgesamt zwei externe parallele Schnittstellen EMIFA und EMIFB. Das EMIFA hat 64 und das EMIFB 8 Datenleitungen. An dem zweiten Interface sind bei dem verwendeten Board ein FLASH Speicher und ein CPLD für die Ansteuerung der LEDs und zum Auslesen der Schalter angeschlossen. An dem EMIFA ist das 16 MB große SDRAM mit allen 64 Datenleitungen angeschlossen. Darüber hinaus sind 32 der 64 Datenleitungen über den Erweiterungsstecker erreichbar (siehe [6]).

Bild 3.7 zeigt die Adresstabelle für das DSK Board. Jedes EMIF verfügt über 4 separate Freigabesignale, um die einzelnen Adressbereiche auszuwählen. Über den Adressbereich von EMIFA mit den Freigabesignalen CE2 und CE3 wird die Verbindung zur Adapterplatine hergestellt.

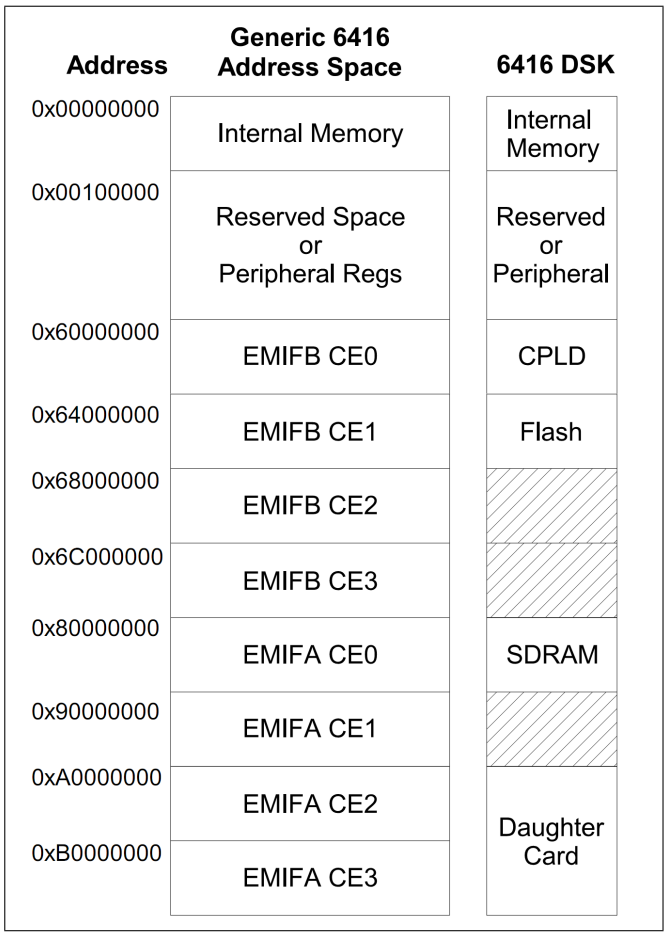


Bild 3.7: Adresstabelle[6]

Grundsätzlich wird das EMIF bei diesem Prozessor mit einer Taktfrequenz von 125 MHz betrieben. Dabei gibt es zwei verschiedene Übertragungsmodi. Zum Einen können die Daten taktsynchron und zum Anderen asynchron übertragen werden.

Für die Verwendung von handelsüblichem Speicher wie SDRAM gibt es eine gesonderte Konfiguration, bei der das EMIF die gesamte Kommunikation automatisch anhand der Speicherparameter realisiert (siehe [7]).

Wie die verwendeten Steuersignale und Signalformen für das Lesen und Schreiben über das EMIF aussehen, wird im Kapitel 5.2.2 im Zusammenhang mit der Konfiguration erläutert.

3.1.6 SDRAM

Auf dem DSK Board kommen zwei MICRON MT48LC2M32B2 TG-6 SDRAM ICs zum Einsatz. Jedes IC hat eine Speicherkapazität von 8 MB und einen 32 Bit breiten Datenbus. Die Speicherbausteine sind so an das EMIF angeschlossen, dass es einen 16 MB großen Speicher mit einem 64 Bit breiten Datenbus ergibt (siehe [6]). Betrieben wird der Speicher mit dem EMIF Takt von 125 MHz. Die für SDRAM üblichen Steuersignale für die Zeilen- und Spaltenauswahl werden ebenfalls durch das EMIF zur Verfügung gestellt. Der Speicher kann daher ohne große Probleme direkt an den Prozessor angeschlossen werden, so dass dieser direkt auf die Adressen des Speichers zugreifen kann.

Da es sich bei SDRAM um flüchtigen Speicher handelt, muss der Speicherinhalt von Zeit zu Zeit aufgefrischt werden. Um diese Aufgabe muss sich der Prozessor jedoch nicht kümmern, da dies ebenfalls vom EMIF übernommen wird. Sollte der Fall eintreten, dass der Speicher gerade aufgefrischt wird während der Prozessor darauf zugreifen möchte, muss der Prozessor warten. Bei sehr schnellen Anwendungen ist daher der schnellere interne Speicher vorzuziehen, sofern dessen Speicherkapazität ausreicht.

Im Rahmen dieser Arbeit reicht der interne Speicher für die Zwischenspeicherung der Bilder nicht aus, aber die Bandbreite des externen SDRAM ist ausreichend, um das Echtzeitverhalten sicher zu stellen.

3.1.7 Multichannel Buffered Serial Port

Der Multichannel Buffered Serial Port (McBSP) ist eine serielle Schnittstelle. Dieser Peripheriebaustein ist sehr umfangreich um eine Vielzahl von verschiedenen seriellen Übertragungsformaten nachzubilden. Unter Anderem kann diese Schnittstelle als Serial Peripheral Interface (SPI) oder als Inter-IC Sound (I2S) verwendet werden.

In dieser Arbeit wird der McBSP zur Kommunikation zwischen zwei DSPs eingesetzt und daher in der Standardkonfiguration verwendet. Bild 3.8 zeigt die Signalform der einzelnen Signale. Für jede Übertragungsrichtung gibt es drei Signale. Das Taktsignal (CLKX, CLKR) dient zur Synchronisation der Daten (DX, DR). Die Signale FSR und FSX sind zur Rahmensynchronisation. Mit Beginn einer neuen Übertragung eines 32 Bit langen Datenwortes wird dieses Signal gesetzt, um dem Empfänger den Anfang einer neuen Übertragung zu signalisieren.

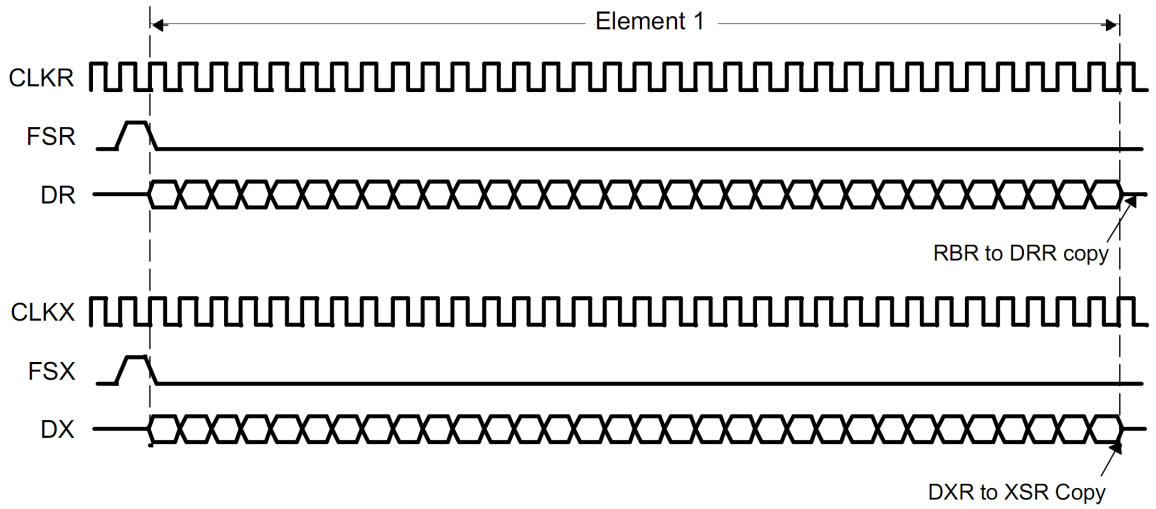


Bild 3.8: McBSP Signalübertragung[8]

Mit dem Empfang der Rahmensynchronisation werden die Daten auf der Datenleitung mit jedem Übertragungstakt in ein Schieberegister geschrieben. Sobald 32 Bit empfangen wurden, wird der Inhalt des Schieberegisters (RBR) in das Datenregister (DRR) kopiert und die nächste Übertragung kann beginnen.

Anhand der Konfiguration kann beim Empfangen und/oder Senden je ein Interrupt aktiviert werden, der dem Prozessor das Abschließen der Übertragung bzw. des Datenempfangs signalisiert. Sollten größere Datenmengen als einzelne Datenworte übertragen werden, so besteht die Möglichkeit, den EDMA Kontroller für die Verwaltung der ankommenden bzw. abgehenden Daten zu verwenden, um den Prozessor zu entlasten.

3.2 DSKEye Adapterplatine

Dieses Kapitel befasst sich mit der in Bild 3.9 dargestellten DSKEye Adapterplatine von der Firma Bitec Ltd. Dieses Board wird in Verbindung mit einem Kameramodul von Omnivision vertrieben. Wichtigstes Bauteil auf dieser Platine ist ein Altera Cyclone II FPGA, über das die Verbindung zwischen der Kamera und dem DSP hergestellt wird. Außerdem ist noch eine Gigabit-Ethernet Schnittstelle mit den dafür benötigten ICs implementiert.

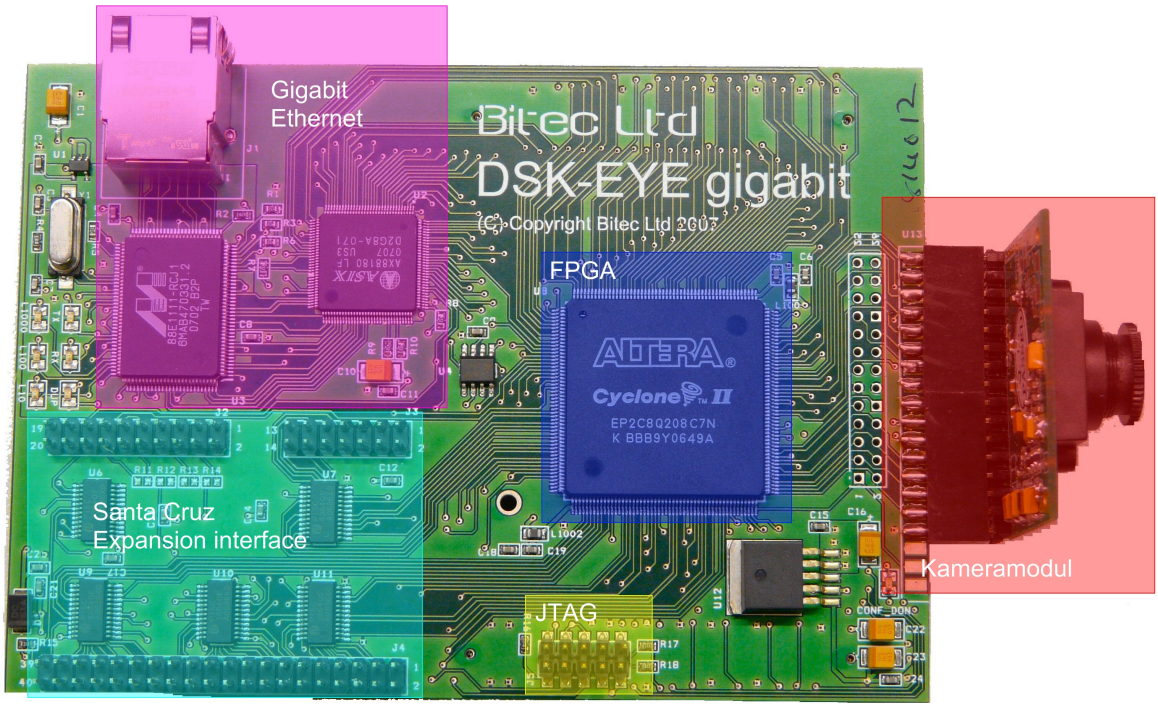


Bild 3.9: DSKEye Board

Im Zusammenhang mit dieser Platine vertreibt der Hersteller Beispielprojekte für verschiedene Anwendungen mit dem DSK Board. Dabei ist allerdings zu beachten, dass die Beispielprojekte für ein DSK6713 Board mit einem Floating-Point DSP, der mit 225 MHz betrieben wird, entwickelt wurden. Die Anschlüsse des DSK6713 sind zu dem hier verwendeten DSK6416 kompatibel, aber der EMIF-Takt ist verschieden. Aus diesem Grund muss das Beispieldesign für das FPGA angepasst werden.

Darüber hinaus handelt es sich bei dem mitgelieferten FPGA-Design lediglich um ein grobes Beispiel, das teilweise funktionsfähig ist. Daher wird die gegebene Vorlage als Ideensammlung für die Entwicklung betrachtet.

Bild 3.10 zeigt das Blockdiagram der DSKEye Platine. Das FPGA ist direkt mit der Kamera verbunden und kann die Bilddaten empfangen. Die Konfiguration der Kamera erfolgt über eine separate I2C Schnittstelle, die im FPGA durch entsprechende Hardwaremodule realisiert werden muss. Über die Anschlüsse auf der Unterseite der Platine erhält das FPGA eine Verbindung über das EMIF mit dem DSP.

Außerdem ist noch ein McBSP mit dem FPGA verbunden. Bei der Verwendung der McBSPs ist aber Vorsicht geboten, da die Bezeichnungen im Schaltplan des DSK Boards und der DSKEye Platine inkonsistent sind. Der McBSP2 vom DSK Board wird auf der DSKEye Platine als McBSP1 bezeichnet. Der McBSP0 ist nicht vollständig an das FPGA angeschlossen und kann daher gar nicht verwendet werden.

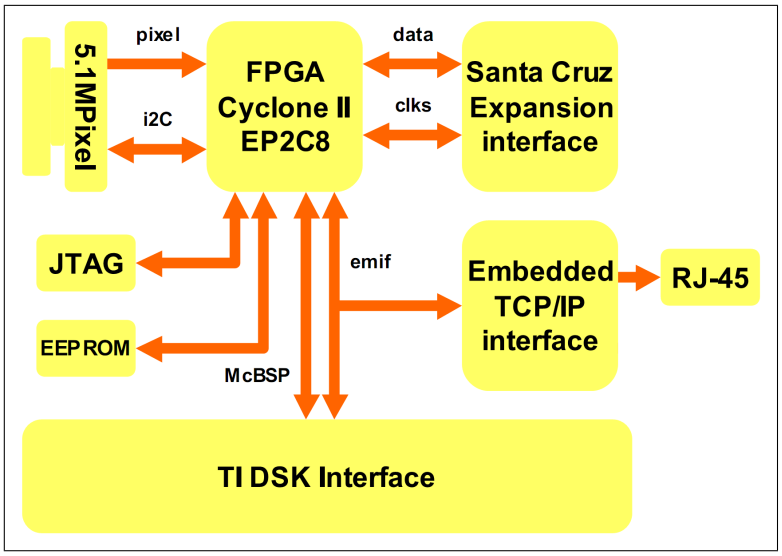


Bild 3.10: DSKEye Blockdiagram[9]

Darüber hinaus ist das Layout der Platine nicht für einen synchronen Datentransfer mit 125 MHz über das EMIF geeignet, da die Leiterbahnführung dafür nicht ausgelegt ist. Dadurch wird das FPGA-Design aufwendiger und der Datendurchsatz geringer bzw. die Busbelegung größer. Das Ethernet Interface ist parallel zum FPGA am EMIF angeschlossen, die Unterscheidung zwischen FPGA und Ethernet wird dabei über die beiden Freigabesignale CE2 und CE3 des EMIF realisiert (siehe Kapitel 3.1.5).

Das Santa Cruz Expansion Interface wird zur Verbindung zwischen den beiden DSKEye Platinen verwendet. Darüber kann das Bild der zweiten Kamera an den DSP zur anschließenden Bearbeitung übertragen werden.

3.2.1 Kamera

Bei den Kameras handelt es sich um zwei 5.1 Megapixel CMOS Farbkameras der Firma Omnivision (siehe Bild 3.11). In der vom Hersteller beigelegten Dokumentation ist die Konfiguration der Kamera anhand von Beispielprogrammen für verschiedene Auflösungen erläutert. Dabei fällt auf, dass ein Großteil der einzustellenden Parameter nicht erläutert wird. Aufgrund dieser fehlenden Informationen ist es leider nicht möglich, einige Parameter zu ändern, da diese wiederum von anderen nicht dokumentierten Parametern abhängen können.

Im Großen und Ganzen ist die Kamera mit einer Auflösung von 640x480 Bildpunkten und einer Bildwiederholungsrate von ca. 30 Hz verwendbar, da für diesen Fall eine undokumentierte aber funktionierende Beispielkonfiguration vorliegt.

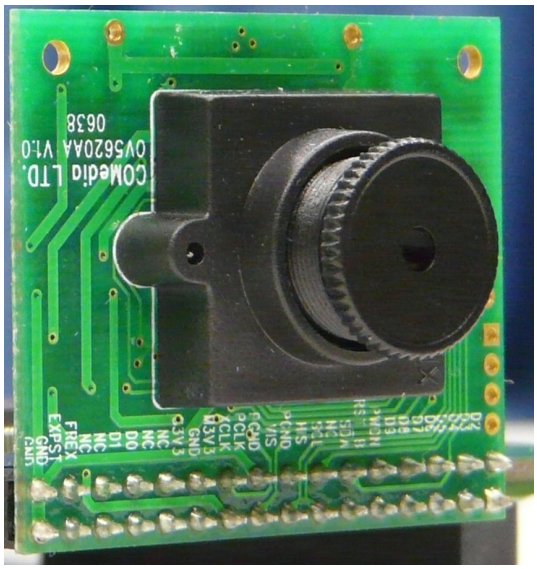


Bild 3.11: Omnivision OV5620 Kameramodul

Die Optik der Kamera hat eine Brennweite von 8.2 mm mit integriertem Infrarot Filter (650 nm) und eine maximale Auflösung von 2592x1944 Bildpunkten. Bei einer Pixelgröße von 2.2 μm x 2.2 μm ergibt das eine Sensorgröße von 5.808 mm x 4.294 mm.

Das Kameramodul besitzt einen 10 Bit breiten Datenport zur Übertragung der einzelnen Bildpunkte. Dazu gehören ebenfalls zwei Kontrollsignale (siehe Kapitel 3.2.1.2) und ein I2C Interface zur Konfiguration der Kamera, das in Kapitel 3.2.1.3 erläutert wird.

3.2.1.1 Sensor

Damit ein Farbbild aufgenommen werden kann, ist auf dem Sensor der Kamera ein spezieller Farbfilter angebracht. Jeder Pixel repräsentiert dadurch nur einen Farbanteil des RGB-Farbraums. Die Aufteilung über die Sensorfläche ist in Bild 3.12 dargestellt. Diese besondere Anordnung wird auch als Bayer Mosaik bezeichnet.

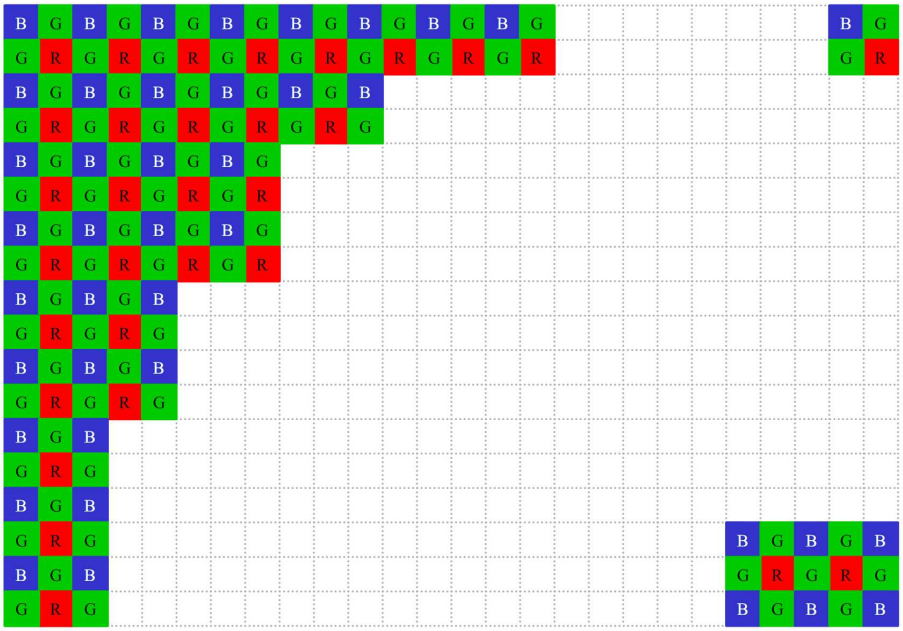


Bild 3.12: Bayer Mosaik[9]

Zu erkennen ist außerdem, dass die Anzahl der grünen Pixel doppelt so groß ist, wie die der blauen oder roten Pixel. Dieser Aufbau beruht auf der Tatsache, dass das menschliche Auge im grünen Farbanteil die größte Empfindlichkeit besitzt.

3.2.1.2 Schnittstelle

Die Schnittstelle der Kamera zur Übertragung der Bildinformationen ist ein paralleles Interface. Die analogen Bildinformationen des Sensors werden über den in der Kamera integrierten 10 Bit A/D-Umsetzer digitalisiert und ausgegeben. Zusätzlich wird noch ein Taktsignal, sowie zwei Signale zur Synchronisation bereitgestellt (siehe Bild 3.13). Die Signale werden im Datenblatt [10] als PCLK (Pixeltakt), VSYNC (vertikale Synchronisation) und HREF (horizontale Referenz) bezeichnet.

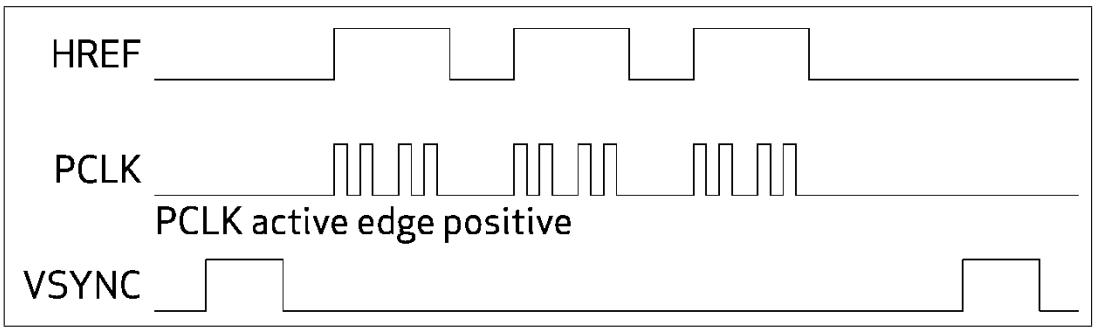


Bild 3.13: Kamerainterface[10]

Die Übertragung eines Bildes beginnt mit der fallenden Flanke des VSYNC Signals. Die Bildpunkte der einzelnen Zeilen werden anschließend synchron zum Pixeltakt bei gesetztem HREF Signal ausgegeben. Die Bildwiederholungsrate der Kamera entspricht dann der Periodendauer des VSYNC Signals. Bei einer Auflösung von 640x480 Bildpunkten ist das HREF Signal für 640 Pixeltakte gesetzt und durchläuft 480 Perioden während einer Bildübertragung. Mit der steigenden Flanke der VSYNC Signals ist die Ausgabe des Bildes abgeschlossen.

3.2.1.3 I2C

Das I2C kompatible Interface der Kamera wird, wie bereits erwähnt, nur zur Konfiguration verwendet. Über die beiden Signale SCL und SDA können Parameter in die jeweiligen Register der Kamera geschrieben werden. Ein Kommando besteht dabei immer aus drei Byte. Angefangen mit einem Schreibkommando, gefolgt von der Registeradresse und dem zu speichernden Parameter.

Im Ruhezustand sind beide Leitungen auf High-Pegel (Bild 3.14). Zu Beginn einer Übertragung wird die Datenleitung vom Sender auf Low-Pegel gesetzt und die anderen Teilnehmer bereiten sich auf neue Daten vor. Die Daten werden anschließend bitweise mit dem Taktsignal übertragen. Am Ende wird noch ein Acknowledge-Bit gesendet um die Übertragung abzuschließen.

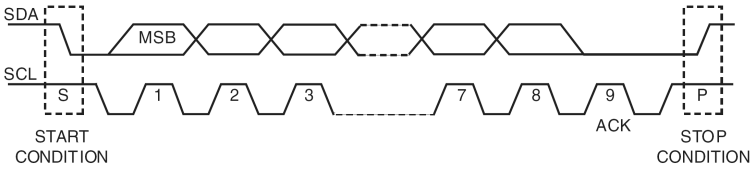


Bild 3.14: I2C Datenübertragung

3.2.2 FPGA

Ein Field Programmable Gate Array (FPGA) ist ein programmierbarer Integrierter Schaltkreis (IC). Die ersten FPGAs bestanden nur aus einem Feld von konfigurierbaren logischen Einheiten (LE), wodurch die Namensgebung zustande kam. Heutzutage sind FPGAs sehr komplexe ICs, die neben den LEs noch weitere Komponenten beinhalten können. Das auf der DSKEye Platine befindliche FPGA ist ein Cyclone II vom Typ EP2C8Q208 C7N von Altera. Die wichtigsten Ausstattungsmerkmale sind in Tabelle 3.1 gezeigt.

Tabelle 3.1: EP2C8Q208 Ausstattung[11]

logische Einheiten (LE)	8256
M4K RAM Blocks	36
RAM Bits insgesamt	165888
Eingebettete Multiplizierer	18
Phase-locked loop	2
Ein- und Ausgänge	138

Neben den 8256 logischen Einheiten sind 36 M4K Ram Blocks und 18 Hardwaremultiplizierer vorhanden, die für die Implementierung verschiedenster Aufgaben einsetzbar sind. Außerdem verfügt dieses FPGA über zwei Phasenregelschleifen (PLL), die zur Generierung verschiedener Takte herangezogen werden können. Welche und wie viele der vorhandenen Ressourcen genutzt werden, ist in Kapitel 4.3 zu finden.

3.2.3 Gigabit Ethernet

Die Gigabit Ethernet Schnittstelle auf der DSKEye Platine basiert auf dem ASIX 88180 IC in Verbindung mit dem Marvel M88E111 IC. Das ASIX 88180 IC ist über die Steckerleisten an der Unterseite der Platine direkt mit dem EMIF des DSP verbunden. Die Übertragung muss dabei asynchron erfolgen und benötigt 5 Takte für einen Schreib- bzw. Lesezugriff bei einem EMIF Takt von 125 MHz. Der effektive Takt liegt also nur bei 25 MHz. Im Bezug auf die Busbreite von 32 Bit ergibt dies bei permanenter Busbelegung eine Bandbreite von lediglich 0.8 Gigabit/s.

Da das EMIF auch noch für andere Peripherien verwendet wird und durch die Netzwerkprotokolle zusätzliches Datenaufkommen entsteht, ist die mit dem Namen der Schnittstelle gegebene Übertragungsrate bei weitem nicht zu erreichen.

3.2.3.1 Ethernet

Beim Ethernet werden die Daten in Paketen (Frames) übertragen. Bild 3.15 zeigt den Aufbau eines solchen Paketes bei der Verwendung des Transmission Control Protokolls (TCP). Den Anfang macht der Ethernet Header, in dem die Quell- und Zielhardwareadresse enthalten ist. Diese Media-Access-Control (MAC) Adressen sind 48 Bit lang. Dadurch kann eine Netzwerkkarte innerhalb eines Abschnittes des Netzwerkes eindeutig identifiziert werden. Die ersten 24 Bit beschreiben eine von der IEEE vergebene Herstellerkennung, die übrigen 24 Bit stehen dem Hersteller zur freien Verfügung. Abschließend ist in dem Header noch ein 16 Bit breites Feld enthalten, dass angibt, welches Protokoll für die nächst höhere Schicht verwendet wird. Beim Einsatz des Internet Protokolls (IP) steht in diesem Feld der Wert $0x0800$.

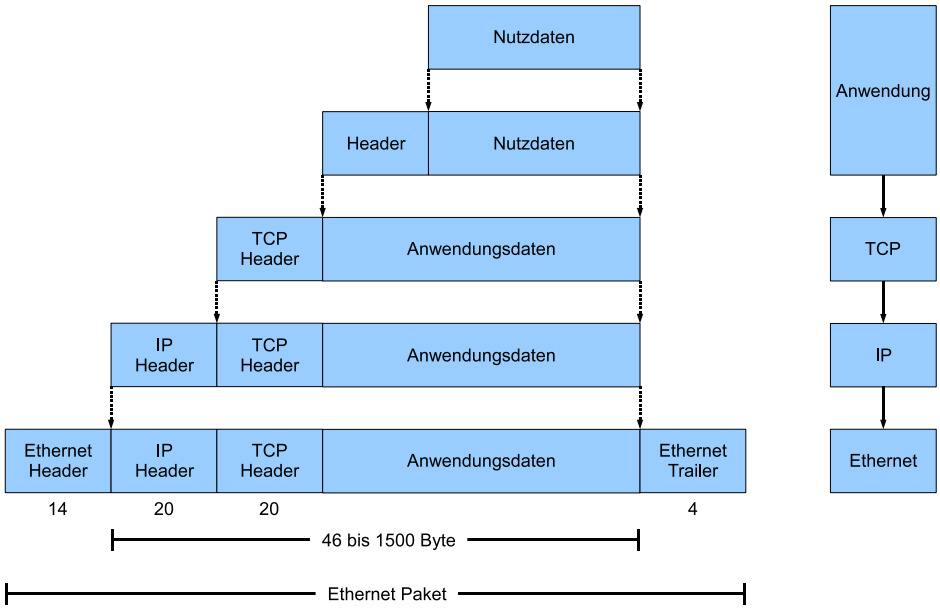


Bild 3.15: Aufbau eines Ethernet Paketes

Am Ende des Paketes wird noch eine 32 Bit Prüfsumme angehängt. Der Empfänger berechnet aus dem empfangenen Paket ebenfalls die Prüfsumme und vergleicht diese mit der gesendeten. Stimmen die Prüfsummen nicht überein, geht der Empfänger von einer fehlerhaften Übertragung aus und verwirft die Daten.

Ein Paket kann bis zu 1500 Byte an Nutzdaten enthalten, wobei dies durch die Protokolle der höheren Ebenen weiter eingeschränkt wird. Der TCP und IP Header benötigen jeweils mindestens 20 Byte, was die Menge an Nutzdaten auf 1460 Byte reduziert.

3.2.3.2 Internet Protokoll

Das Internet Protokoll (IP) ist für die Vermittlung zuständig. Die Pakete werden anhand der im Header (Bild 3.16) eingetragenen Ziel IP-Adresse zugestellt. Das Address Resolution Protokoll (ARP) ermöglicht anhand der IP-Adresse, die zugehörige MAC-Adresse des Ziels zuzuweisen und in den Ethernet Header einzutragen.

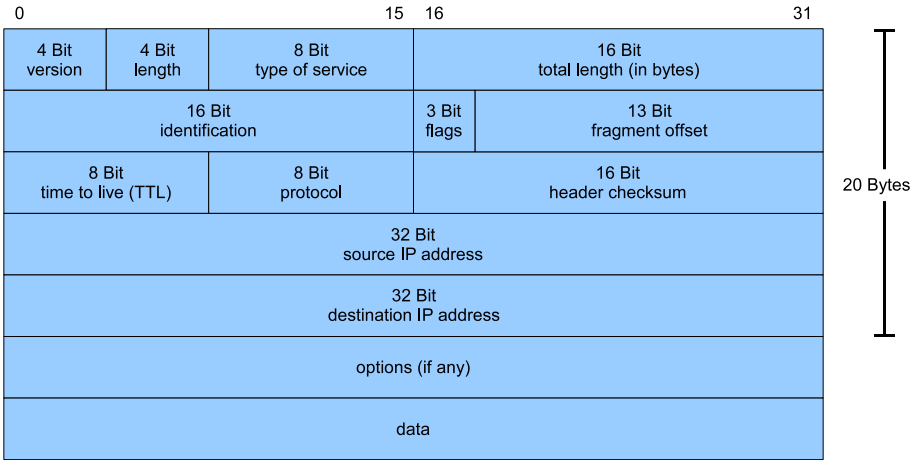


Bild 3.16: Aufbau des IPv4-Header[12]

Am Anfang des Headers steht die Versionsnummer, hierbei wird zwischen den Versionen 4 und 6 unterschieden. Die beiden Versionen unterscheiden sich im Wesentlichen durch die Tatsache, dass bei Version 6 die IP-Adresse auf 128 Bit erweitert wurde. Diese Entwicklung ist nötig, da die Zahl der an das Internet angeschlossener Rechner stetig steigt. Dennoch wird heutzutage meistens Version 4 verwendet. Die Länge des Headers wird in die nächsten 4 Bit eingetragen und zwar als Vielfaches von 32 Bit, gefolgt von der Priorität der Nachricht (type of service).

Die Länge des Paketes setzt sich aus der Länge des Headers, den eventuellen Optionen und den Daten zusammen. Bei 16 Bit ergibt das eine maximale Paketlänge von 65535 Byte. Da dies die Größe eines Ethernet-Paketes überschreitet, werden die IP-Pakete aufgeteilt (fragmentiert). Die nötigen Informationen, um die Daten nach der Übertragung in der richtigen Reihenfolge zu rekonstruieren, befinden sich in der zweiten Zeile des IP-Headers.

Die Lebensdauer (time to live,TTL) eines IP-Paketes bezeichnet die Anzahl der zulässigen Zwischenstellen (Router) bis zum Zielrechner, jede Zwischenstelle dekrementiert diese Zahl. Ist die TTL Null und das Ziel noch nicht erreicht, werden die Daten verworfen und eine Fehlernachricht wird an den Absender zurückgeschickt.

Im Protokollfeld steht, wie beim Ethernet Header, das Protokoll der nächst höheren Ebene. Die Prüfsumme ergibt sich nach dem selben Algorithmus wie beim Ethernet Header, jedoch ohne Berücksichtigung der Optionen und Nutzdaten. Abschließend sind die Ziel- und Quell IP-Adressen zu finden, gefolgt von eventuellen Optionen und den Nutzdaten.

3.2.3.3 Transmission Control Protokoll

Das Transmission Control Protokoll (TCP) ist, wie bereits erwähnt, für die eigentliche Datenverbindung zuständig. Die Übertragung mittels TCP geschieht Verbindungsorientiert. Dadurch ist sichergestellt, dass keine Daten verloren gehen und diese auch in der richtigen Reihenfolge beim Empfängerprogramm ankommen. Damit die Daten das richtige Client- bzw. Server-Programm auf dem Zielrechner erreichen, wird in den Header die Portnummer beider Seiten eingetragen. Ein Server-Programm reagiert auf TCP-Verbindungen auf einem Port. Sofern ein an diesen Port adressiertes Paket eintrifft, wird es von dem Programm verarbeitet. Über die im Header eingetragene Portnummer des Absenders können Daten zurück geschickt werden (siehe Bild 3.17).

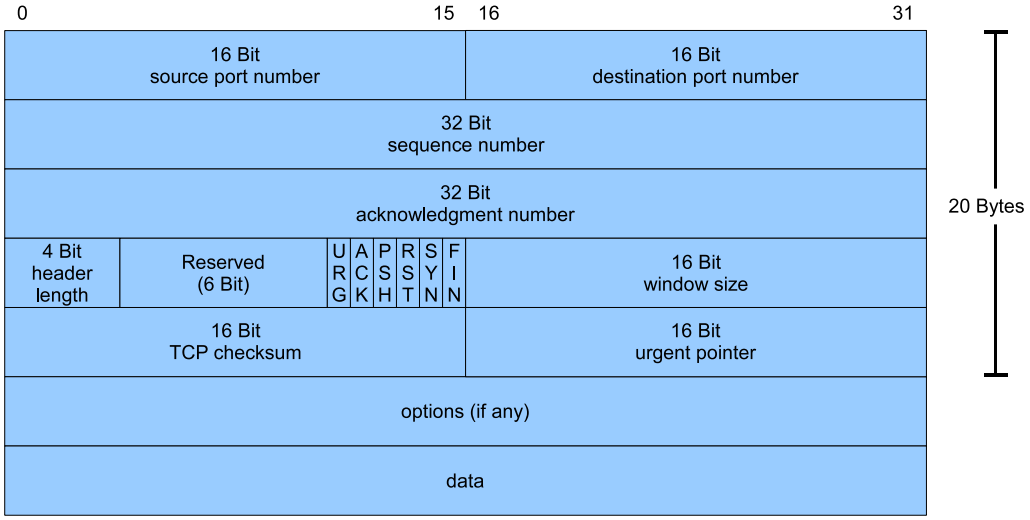


Bild 3.17: Aufbau eines TCP-Header[13]

Die Headerlänge gibt die Startadresse der Nutzdaten innerhalb des TCP-Pakets an, auch hier wird wie beim IP-Header die Länge in 32 Bit Wörtern angeben. Die darauf folgenden Flags (URG, ACK, PSH, RST, SYN, FIN) sind zur Steuerung der Verbindung. Mit jedem Paket gibt der Sender an, wie viel Speicher noch in seinem Empfangspuffer vorhanden ist (window size). Sollte dieser Wert kleiner als das nächste zu sendende Paket sein, wird die Übertragung unterbrochen und gewartet, bis der Puffer wieder genügend Kapazität hat. Dies signalisiert der Empfänger durch das Senden eines Paketes ohne Nutzdaten, aber mit

aktualisierter Puffergröße. Zur Berechnung der Prüfsumme werden neben dem Header auch die Nutzdaten berücksichtigt um Übertragungsfehler zu erkennen.

Die Flags kennzeichnen bestimmte Zustände für die Kommunikation und die Weiterverarbeitung der Daten.

- **URG**
  Ist das Urgent-Flag (dringend) gesetzt, werden die Daten, auf die der Urgent-Pointer zeigt, sofort verarbeitet. Dieses Flag kann verwendet werden, um eine Anwendung im Empfänger abzubrechen. In der Praxis wird dieses Flag aber eher selten verwendet.
- **ACK**
  In Verbindung mit der Acknowledgment-Nummer dient das Acknowledgment-Flag zur Bestätigung von TCP-Paketen. In Verbindung mit dem SYN-Flag wird es für das Drei-Wege-Handshake verwendet.
- **PSH**
  Bei gesetztem Push-Flag werden die Daten ohne Verzögerung, am internen Puffer vorbei, an die Anwendung weitergeleitet.
- **RST**
  Das Reset-Flag dient zum Abbrechen einer Verbindung.
- **SYN**
  Pakete mit gesetztem Synchronize-Flag bauen eine Verbidung auf. Der Server antwortet entweder SYN+ACK, wenn er bereit ist, sonst mit RST. Die anfänglichen Sequenznummern werden hiermit synchronisiert.
- **FIN**
  Das Finish-Flag dient zum Beenden einer Verbindung. Danach werden keine weiteren Daten mehr übertragen, bis die nächste Verbindung aufgebaut wird.

Eine TCP-Verbindung wird mit einem Drei-Wege-Handshake Verfahren aufgebaut (siehe Bild 3.18). Dazu sendet der Client im ersten Schritt ein TCP-Paket mit gesetztem SYN-Flag und beliebiger Sequenznummer an den Server (SYN-Paket). Wenn der betreffende Port geschlossen ist, antwortet der Server mit einem Paket, bei dem das RST-Flag gesetzt ist und die Verbindung kann nicht aufgebaut werden. Ist der Port geöffnet, so antwortet der Server im zweiten Schritt auf diese Verbindungsanfrage mit einem Paket, bei dem SYN- und ACK-FLAG gesetzt sind. Die Acknowledge-Nummer entspricht der vorherigen Sequenznummer um eins erhöht (siehe Bild 3.18) und die Sequenznummer wird vom Server vergeben. Im dritten und letzten Schritt des Handshake Verfahrens antwortet der Client auf das SYN+ACK Paket mit einem ACK-Paket. Die empfangene Sequenznummer wird zur Bestätigung an den Server zurück geschickt und als Acknowledgenummer wird die um eins erhöhte Sequenznummer verwendet. Damit ist die Verbindung aufgebaut und es können in beiden Richtungen Daten übertragen werden.

Bei der Datenübertragung wird jedes empfangene Paket mit einem ACK-Paket bestätigt. Dabei beinhaltet die Acknowledgenummer die Summe der Sequenznummer des empfangenen Paketes und der aktuellen Paketgröße. Fehlt diese Bestätigung, überträgt der Client das Paket erneut, bis eine Bestätigung eintrifft oder ein Timeout auslöst.

Bei den in der Hard- und Software verwendeten Puffern kann es vorkommen, dass die Pakete in unterschiedlicher Reihenfolge übertragen werden. Anhand der Sequenznummer können diese durch die Hardware oder den IP-Stack wieder in die richtige Reihenfolge gebracht werden.

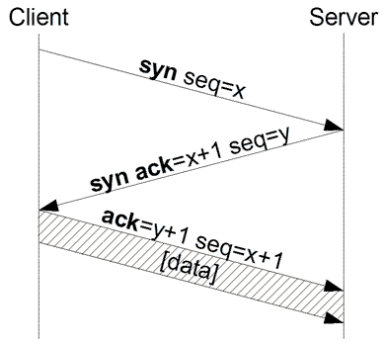


Bild 3.18: TCP Verbindungsaufbau[14]

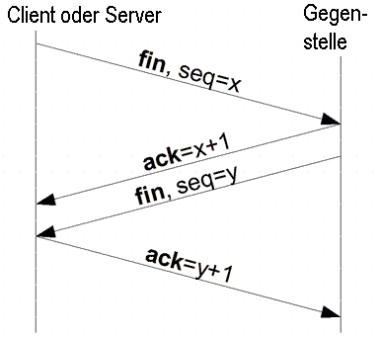


Bild 3.19: TCP Verbindungsabbau[14]

Der Abbau der TCP-Verbindung (siehe Bild 3.19) kann von beiden Seiten erfolgen. Dazu sendet einer der Teilnehmer ein Paket mit gesetztem FIN-Flag. Die Gegenstelle bestätigt dies mit einem ACK-Paket und sendet ebenfalls ein FIN-Paket um die Verbindung zu beenden. Abschließend wird das FIN-Paket der Gegenstelle bestätigt und die Verbindung ist beendet. Zur Sicherheit wird bei einigen Implementierungen noch ein RST-Paket hinterher gesendet um die Verbindung zurückzusetzen.

3.2.3.4 User Datagram Protokoll

Das User Datagram Protokoll ist im Gegensatz zu TCP keine feste Verbindung. Es werden lediglich einzelne Datenpakete ohne jegliche Kontrolle übertragen. Für die Beseitigung von Übertragungsfehlern und eventuell nötige Handshake-Verfahren ist der Anwender selbst verantwortlich. Über die Portnummern werden wie bei TCP die Anwenderprogramme im Server und Client identifiziert. Die Portnummern sind dabei unabhängig von den Protokollen, d.h., es kann sowohl eine UDP-, als auch eine TCP-Verbindung zur selben Zeit über den selben Port erfolgen.

Auch bei diesem Protokoll wird die Summe aus der Länge des Headers und der Nutzdaten in den Header eingetragen. Die Prüfsumme dient auch hier zur Fehlererkennung. Wenn Übertragungsfehler auftreten und die Prüfsumme demnach falsch ist, wird das Paket ohne Fehlermeldung verworfen.

Dafür ist der Header (Bild 3.20) nur 8 Byte lang und die Menge an Nutzdaten ist dementsprechend größer. Die Pakete werden nicht durch den Empfänger bestätigt, was zu Datenverlusten führen kann, wenn der Empfangspuffer zu voll ist. Dennoch ist dieses Protokoll zur Übertragung von großen Datenmengen geeignet, da der Overhead durch die nicht verbingunsorientierte Kommunikation deutlich geringer ist (vgl. Bild 3.17 und 3.20).

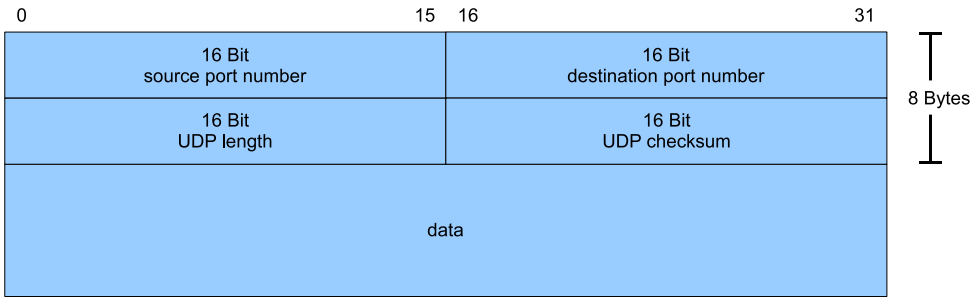


Bild 3.20: Aufbau des UDP-Header[15]

4 Hardwarekonfiguration

In diesem Kapitel wird die Entwicklung der Hardware dargestellt. Als Grundlage dafür werden zunächst die Anforderungen in Kapitel 4.1 dargestellt, die sich anhand der Aufgabenstellung ergeben. Anschließend folgt das Konzept zur Realisierung der Hardware des Stereokamerasystems in Kapitel 4.2. Nachdem die Anforderungen und das Konzept erläutert wurden, folgt die Beschreibung der einzelnen Hardwaremodule, die im FPGA implementiert werden.

4.1 Anforderungen

Nach der Aufgabenstellung aus Kapitel 1.2 werden zwei Kameras mit einer Auflösung von 640x480 Bildpunkten und einer Bildwiederholungsrate von ca. 30 Hz verwendet. Das dadurch entstehende Datenaufkommen beläuft sich auf 73.728 MBit/s für eine Kamera (siehe Gleichung 4.1) und 147.456 MBit/s für zwei Kameras.

$$640 \cdot 480\,\text{Pixel} \cdot 30\,\frac{\text{Bilder}}{s} \cdot 8\,\frac{\text{Bit}}{\text{Pixel}} = 73.728\,\text{MBit/s} \tag{4.1}$$

Das EMIF zwischen DSP und FPGA verfügt, wie in Kapitel 3.1.5 bereits dargestellt wurde, über einen 32 Bit breiten Datenbus und wird mit einem Takt von 125 MHz betrieben. Der Transfer kann aufgrund der Leiterbahnführung leider nicht synchron erfolgen. Anhand des Beispielprojekts der Herstellerfirma wurden durch erste Tests Parameter für den störungsfreien asynchronen Betrieb ermittelt. Dabei hat sich herausgestellt, dass für jeden Lese- oder Schreibzugriff fünf Takte benötigt werden. Daraus resultiert eine Bandbreite von 800 MBit/s bei maximaler Auslastung (siehe Gleichung 4.2).

$$32\,\text{Bit} \cdot 125\,\text{MHz} \cdot \frac{1}{5\ \text{Takte/Zugriff}} = 800\,\text{MBit/s} \tag{4.2}$$

Vergleicht man die Bandbreite des EMIF mit dem Datenaufkommen von zwei Kameras unter Vernachlässigung der Latenzzeiten, so liegt die Auslastung des EMIF bei ca. 18.4 %. Es stehen also noch ausreichend Kapazitäten für andere Peripherien wie die Ethernet Schnittstelle zur Verfügung.

4.2 Stereokamerasystem

Dieses Kapitel befasst sich mit dem Konzept zur Realisierung der Hardware. Es stehen zwei DSK6416 Entwicklungsboards, sowie zwei DSKEye Platinen mit Kamera zur Verfügung. Bild 4.1 gibt eine Übersicht über den prinzipiellen Aufbau. Ein Board bestehend aus DSK6416 und DSKEye Platine ist für die Bildverarbeitung zuständig und muss daher beide Bilder zur Verfügung haben. Der andere DSP kann lediglich auf eine Kamera für die Visualisierung sowie die Ergebnisse der Bildverarbeitung zugreifen.

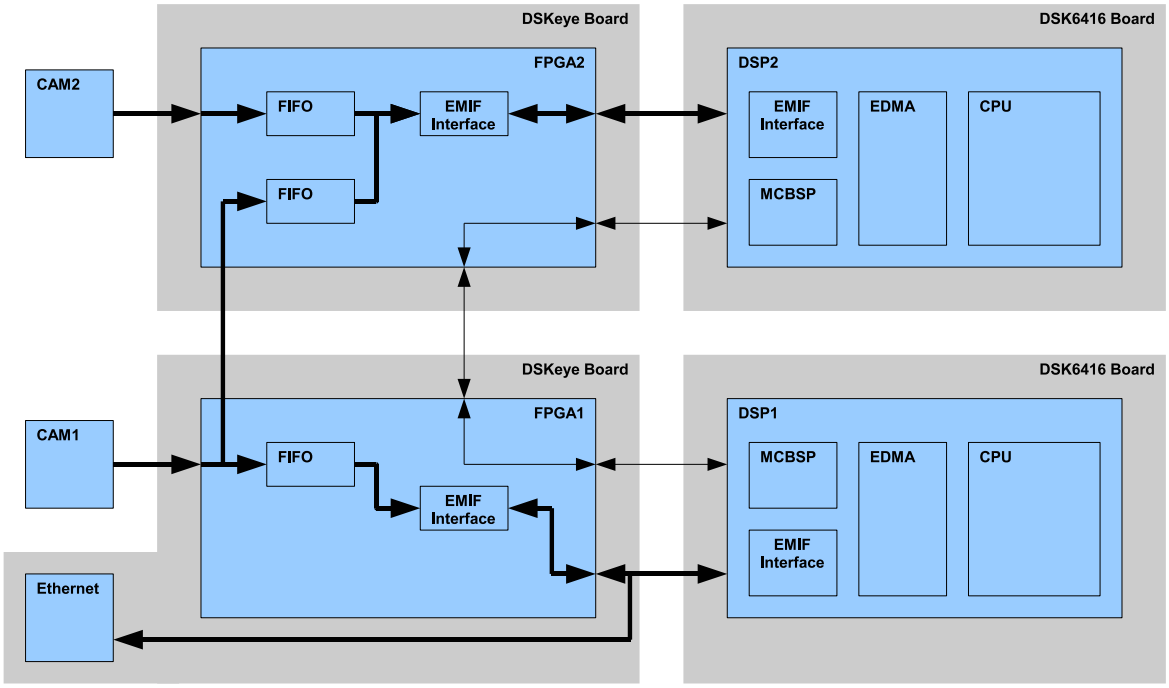


Bild 4.1: Überischt

Die einzelnen Bildpunkte der Kameras werden in einem FIFO (First In, First Out) gespeichert und sobald eine Bildzeile von 640 Pixeln darin enthalten ist, wird dies dem DSP durch einen Interrupt mitgeteilt. Der DSP liest dann mit Hilfe des EDMA Kontrollers die Daten aus dem FIFO aus und kopiert diese in das externe SDRAM.

Nachdem ein Bild von jeder Kamera in dem oberen DSP in Bild 4.1 angekommen ist, startet die Bildverarbeitung. Der andere DSP hat parallel dazu das Bild einer Kamera empfangen und trägt die über den McBSP erhaltenen Ergebnisse der Bildverarbeitung in das Kamerabild ein. Abschließend erfolgt die Übertragung des Ergebnisses über die Ethernet Schnittstelle an einen PC, auf dem ein zugehöriges MATLAB-Skript ausgeführt wird.

4.3 FPGA

Nachdem die Anforderungen und das Konzept soweit geklärt sind, folgt die Implementierung der benötigten Hardwaremodule im FPGA. Da für diese Arbeit zwei FPGAs verwendet werden und ein Großteil der entwickelten Module in beiden zum Einsatz kommen werden die beiden Plattformen unterschiedlich bezeichnet. Die im oberen Teil von Bild 4.1 dargestellte Kombination aus DSK6416 und DSKEye Platine wird im Folgenden mit dem Zusatz „Stereo“ bezeichnet, da diese Plattform Zugriff auf die Daten beider Kameras besitzt. Die andere Plattform hat die Kontrolle über die Ethernet Schnittstelle und wird daher naheliegender weise mit dem Zusatz „Ethernet“ bezeichnet. Diese Bezeichnungen werden auch für die noch folgenden Kapitel beibehalten.

Zur Veranschaulichung werden die einzelnen Hardwaremodule nicht anhand des VHDL-Codes, sondern überwiegend durch Blockdiagramme dargestellt. Die Farbgebung ist dabei für alle in diesem Kapitel enthaltenen Blockdiagramme gleich. Alle Komponenten und Prozesse, die getaktet arbeiten, werden hellbau dargestellt. Die übrigen, rein kombinatorischen Elemente, werden dunkler dargestellt. Bei manchen Blockdiagrammen ist eine separate Legende angegeben, die über die Bedeutung besonderer Signale Aufschluss gibt. Konstanten beginnen mit „C_“ und Signale, die nur innerhalb einer Komponente verwendet werden, beginnen mit dem Zusatz „S_“. Instanzen werden mit „INST_“ und Prozesse mit „P_“ bezeichnet.

Bei dem Beispielprojekt der Firma Bitec ist ein FIFO zum Zwischenspeichern der Pixel der Kameras implementiert. Sobald eine Bildzeile im FIFO enthalten ist, wird beim DSP ein Interrupt ausgelöst und die Daten werden über das EMIF ausgelesen. Das Prinzip wird in dieser Arbeit übernommen, die Quelltexte müssen jedoch größtenteils neu entwickelt werden, damit die Funktionalität für die zweite Kamera implementiert werden kann. Darüber hinaus sind die vorliegenden Quelltexte in ihrer undokumentierten und intransparenten Form nur schwer nachvollziehbar.

Ausgehend von den Anforderungen und dem Beispielprojekt der Firma Bitec wird also ein FIFO zum Zwischenspeichern der Bildinformation der Kamera benötigt. Außerdem muss zum Transfer der Daten über das EMIF des DSP eine entsprechende Schnittstelle entworfen werden. Die darüber hinaus notwendigen Komponenten zur Konfiguration und Kontrolle der ordnungsgemäßen Funktion der Kameras werden ebenfalls im laufe dieses Kapitels erläutert.

Im Folgenden wird zunächst die physikalische Verbindung zwischen den beiden Plattformen dargestellt, bevor die beiden FPGAs beschrieben werden. Anschließend folgt die Erläuterung der einzelnen Teilkomponenten, die in beiden Projekten zum Einsatz kommen.

4.3.1 Platinenverbindung

Die Verbindung zwischen den beiden Plattformen wird über den Santa Cruz Erweiterungsanschluss der DSKEye Platinen hergestellt. Dabei werden die Platinen über die Anschlüsse J2 und J4 miteinander verbunden (siehe [16]). In Bild 4.2 sind alle relevanten Signale mit Quelle und Ziel dargestellt.

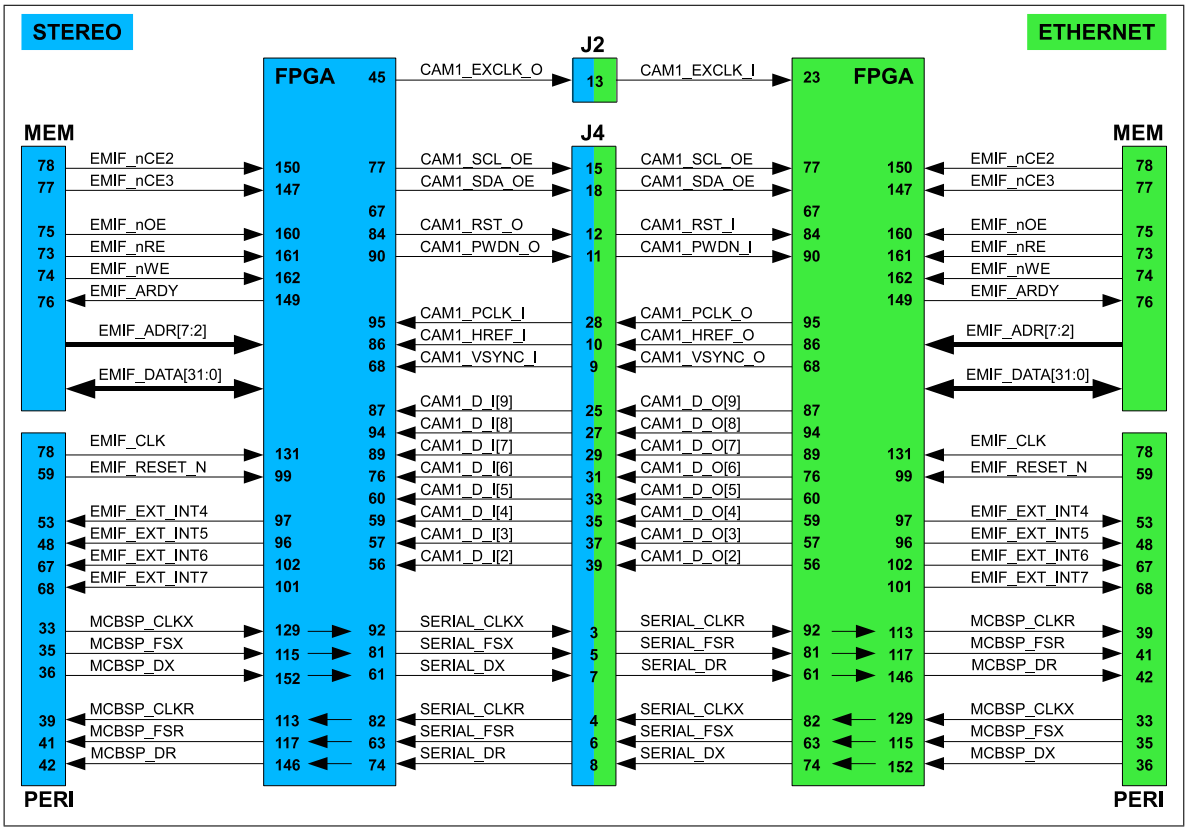


Bild 4.2: Platinenverbingung

Die Stereo-Plattform stellt den Master dar und die Ethernet-Plattform den Slave. Dementsprechend gehen die Signale zur Steuerung (Takt, Reset, PowerDown) und Konfiguration (I2C) der Kamera von der Stereo-Plattform aus. Die Datenleitungen der Kamera auf der Ethernet-Plattform stehen der Stereo-Plattform zur Verfügung um Kontrolle über die Kamera zu erlangen.

Die Schnittstelle zwischen DSP und FPGA ist auf beiden Plattformen identisch. Lediglich die serielle Kommunikation über McBSP ist auf Seiten der FPGAs zwischen Sende- und Empfangssignalen gedreht.

4.3.2 Ethernet-Plattform

Bild 4.3 zeigt die oberste Ebene des FPGA Entwurfs für die Ethernet Plattform. Diese besteht im Wesentlichen aus drei Teilen. Die Komponente CAM2EMIF ist für die Zwischenspeicherung der Bilddaten der Kamera in einem FIFO zuständig. Die nachgeschaltete Komponente EMIF_INTERFACE ermöglicht dem DSP, die Bildpunkte aus dem FIFO der anderen Komponente über das EMIF auszulesen. Darüber hinaus ist noch ein Statusregister implementiert, über das der DSP durch entsprechende Schreibzugriffe die Reset und PowerDown Signale der Kamera steuern kann. Diese Signale werden mit denen vom Master logisch verknüpft, damit beide Plattformen darauf Zugriff haben. Dies ist für die Synchronisation der beiden DSP nötig und wird in Kapitel 5.3.2 erläutert.

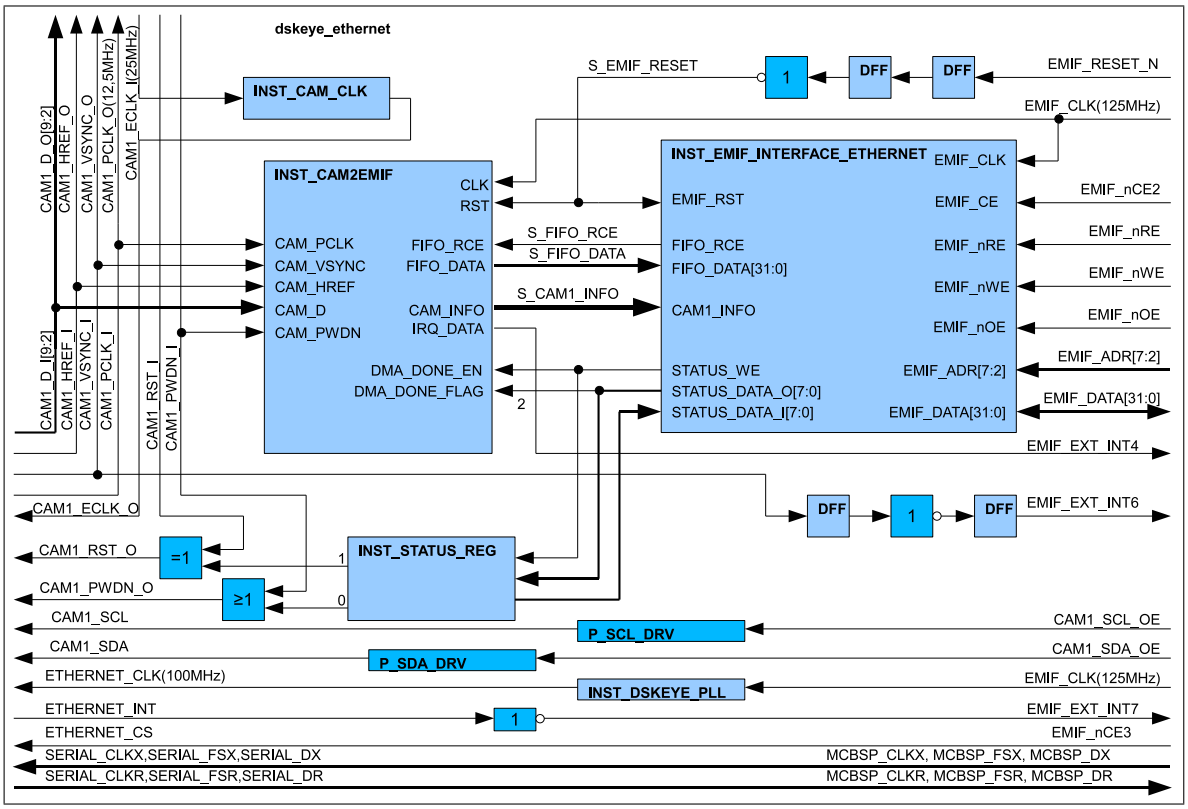


Bild 4.3: Top Entity - dskeye_ethernet.vhd

Außerdem ist ein Bit des Statusregisters mit der CAM2EMIF Komponente verbunden. Dies dient als Handshakesignal zwischen DSP und FIFO, damit bei einem eventuellen Datenverlust der EDMA Kontroller neu synchronisiert werden kann. Eine genauere Erklärung dazu erfolgt im Kapitel 5.2.5.

Die Instanz CAM_CLK enthält lediglich eine Phasenregelschleife (PLL), die die Frequenz des Taktsignals für die Kamera halbiert. Dies ist notwendig, da durch eine schlechte Signalintegrität bei der Übertragung der Takt verdoppelt wird.

Das I2C Interface zur Konfiguration der Kameras wird an die Ethernet Plattform nur unidirektional übertragen. Da die Kamera aber ein bidirektionales Interface hat, wurden die Tristate-Treiber durch die Prozesse P_SCL_DRV und P_SDA_DRV hinzugefügt.

Der Ablauf sieht wie folgt aus: Nachdem die Kamera durch das I2C Interface von der Stereo-Plattform konfiguriert wurde, nimmt die CAM2EMIF Komponente die Daten der Kameras entgegen. Sobald eine Bildzeile von 640 Pixeln im FIFO enthalten ist, wird das Signal EMIF_EXT_INT4 gesetzt. Der EDMA Kontroller wird dann die Daten über die Komponente EMIF_INTERFACE aus dem FIFO auslesen. Nachdem der DSP ein komplettes Bild empfangen hat, wird das Synchronistationssignal DMA_DONE_FLAG gesetzt und das nächste Bild kann übertragen werden. Außerdem ist das VSYNC Signal, was den Anfang eines neuen Bildes signalisiert, invertiert an den EMIF_EXT_INT6 angeschlossen. Dieser Interrupt wird mit Beginn eines neuen Bildes ausgelöst.

Die Signale für das Ethernet Interface (Takt, ChipSelect, Interrupt) und den McBSP (Takt, Sync, Daten) werden nur durch das FPGA durchverbunden. Das benötigte Taktsignal von 100 MHz für das Ethernet Interface wird dabei durch eine PLL aus dem 125 MHz EMIF Takt abgeleitet.

Eine detailierte Beschreibung der einzelnen Instanzen und Prozesse erfolgt nach der Erklärung der Top-Entity der Stereo-Plattform, da dort die selben Instanzen verwendet wurden.

4.3.3 Stereo-Plattform

Der FPGA Entwurf für die Stereo-Plattform setzt sich im Wesentlichen aus den selben Komponenten wie bei der Ethernet-Plattform zusammen. Für jede Kamera ist eine Instanz des CAM2EMIF implementiert (siehe Bild 4.4). Das EMIF_INTERFACE unterscheidet sich von der Ethernet-Plattform lediglich in der Tatsache, dass diese Instanz ein Interface zur I2C Komponente besitzt und für zwei Kameras ausgelegt ist. Auch bei diesem Projekt ist das VSYNC Signal der ersten Kamera invertiert mit dem Interrupt EMIF_EXT_INT6 verbunden. Aus Gründen der Übersicht entfällt es aber in diesem Blockdiagramm.

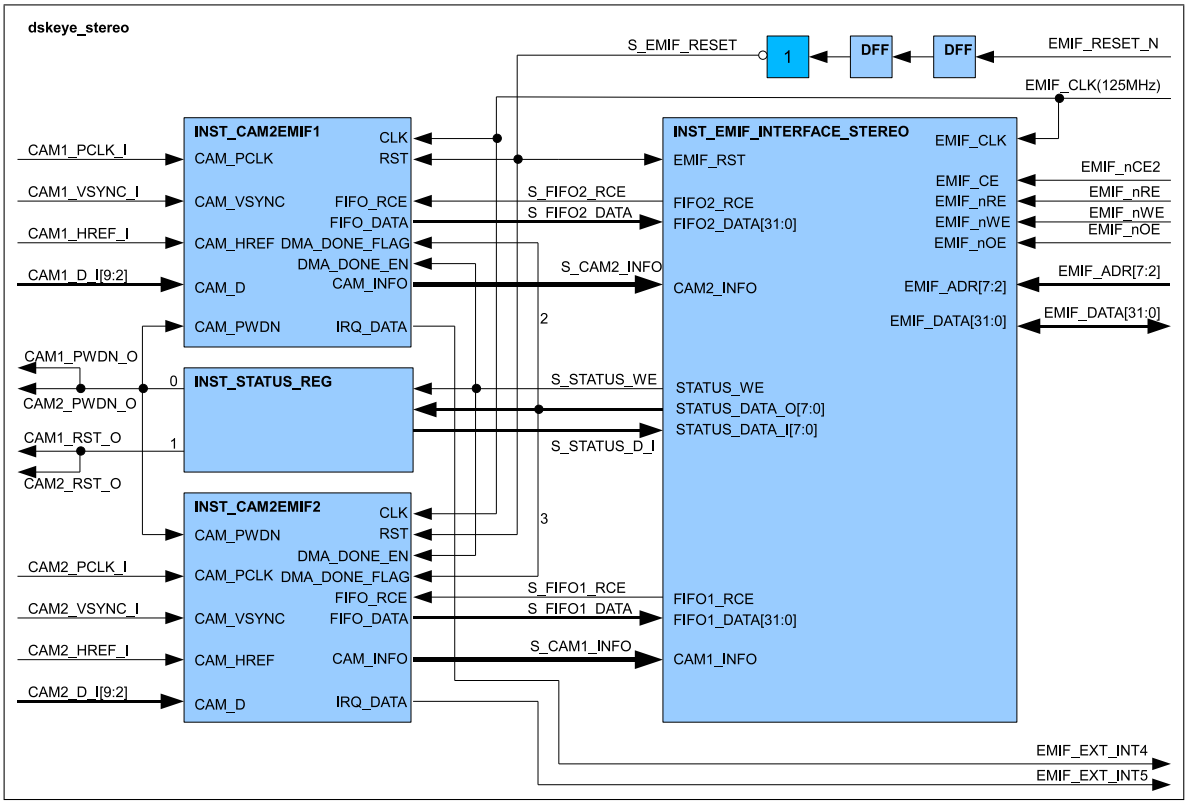


Bild 4.4: Top Entity - dskeye_stereo.vhd

Der zweite Teil der Top-Entity der Stereo-Plattform ist in Bild 4.5 dargestellt. Dabei handelt es sich um das I2C Interface zur Konfiguration der Kameras, sowie den Signalen für das Ethernet Interface, einer PLL und den McBSP. Über die EMIF_INTERACE Komponente kann auf die I2C Komponente zugegriffen werden um Daten über die I2C Schnittstelle zu senden. Die hier gezeigte I2C Komponente ist aus dem Beispielprojekt der Firma Bitec übernommen. Es handelt sich um ein OpenSource Projekt und kann daher frei verwendet werden.

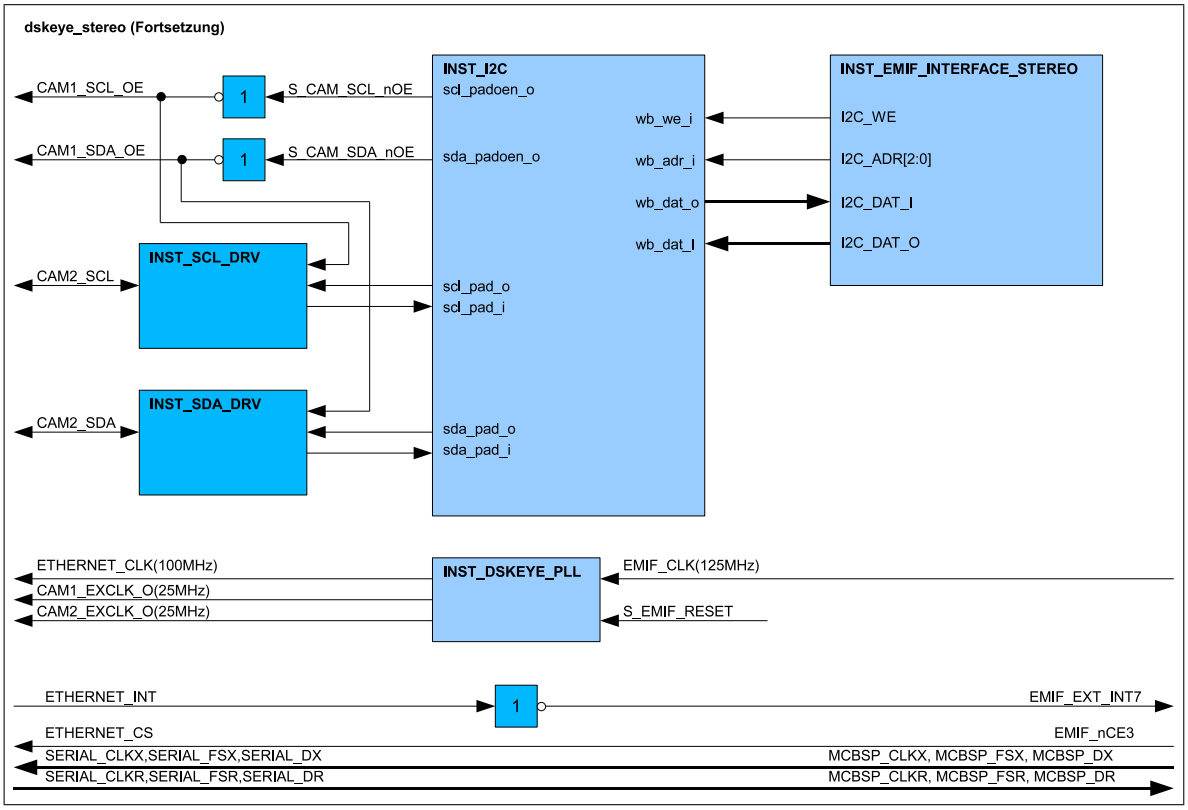


Bild 4.5: Top Entity - dskeye_stereo.vhd (Fortsetzung)

Lediglich die Tristate Treiber für die bidirektionale Kommmunikation wurden aus der Komponente extrahiert, um die zweite Kamera unidirektional anschließen zu können. Dies ist nötig, da eine bidirektionale Verbindung zur zweiten Kamera über die beiden FPGAs nicht möglich ist.

Die PLL in diesem Projekt ist für die Generierung der Takte von 25 MHz für die beiden Kameras und dem 100 MHz Takt für das Ethernet Interface zuständig. Die darüber hinaus benötigten Signale für das Ethernet Interface und die serielle Kommunikation über McBSP sind analog zur Ethernet-Plattform implementiert, auch wenn das Ethernet Interface bei dieser Plattform vorerst nicht zum Einsatz kommt.

In den folgenden Unterkapiteln werden die einzelnen Komponenten im Detail erläutert, die sowohl bei der Ethernet- als auch bei der Stereo-Plattform implementiert sind. Das Zusammenspiel zwischen den spezifischen Hardwarefunktionen und mit der auf dem DSP ablaufenden Software wird anschließend im Zusammenhang mit der Softwarekonfiguration in Kapitel 5 näher erläutert.

4.3.4 Schnittstelle zwischen Kamera und EMIF

Die Komponente CAM2EMIF unterteilt sich in zwei Bereiche, die aus jeweils zwei Komponenten bestehen. Bild 4.6 zeigt die enthaltenen Instanzen und ihre Verbindung untereinander.

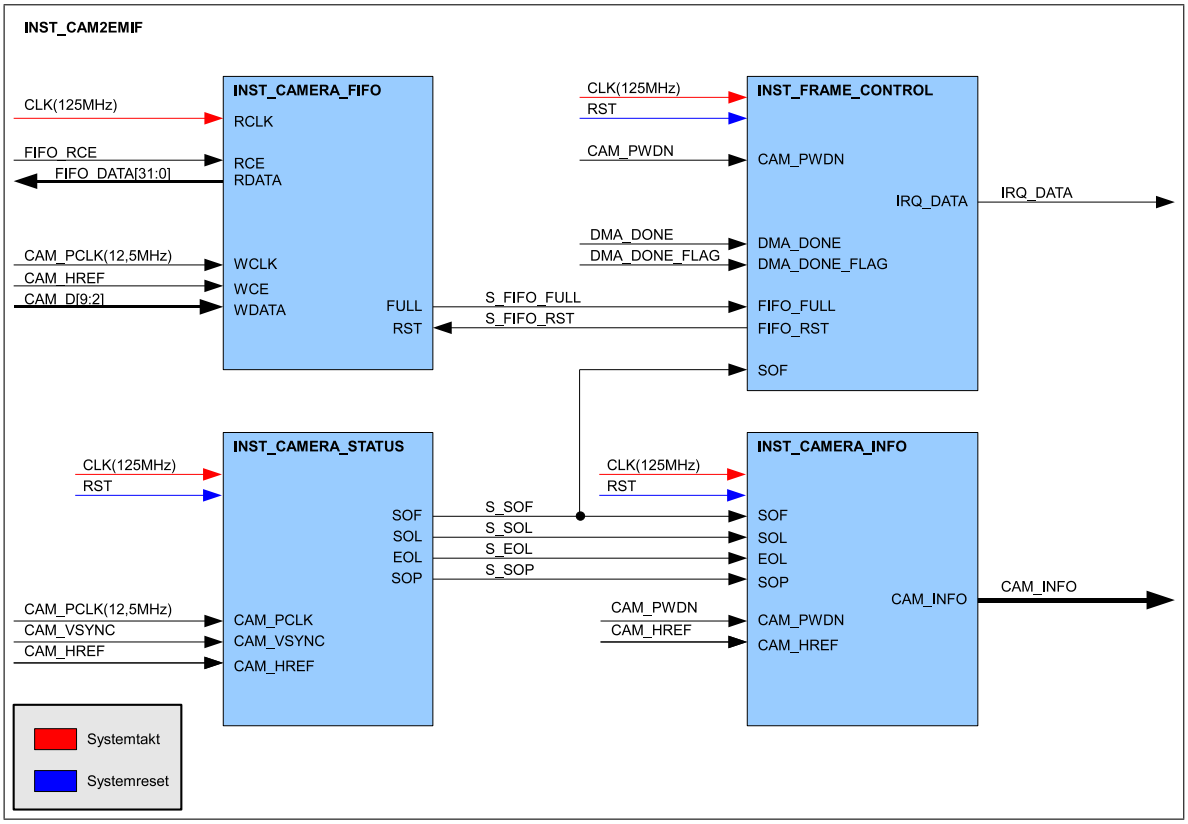


Bild 4.6: Blockdiagramm zu cam2emif.vhd

Die CAMERA_FIFO Komponente empfängt die Bilddaten (CAM_D) der Kamera taktsynchron zum Pixeltakt (CAM_PCLK). Als Freigabesignal wird das von der Kamera gelieferte Signal (CAM_HREF) verwendet. Dieses Signal ist, wie bereits erwähnt, nur aktiv, wenn Bildpunkte übertragen werden. Über das Freigabesignal (FIFO_RCE) können die Bilddaten aus dem FIFO innerhalb der Komponente taktsynchron zum EMIF Takt ausgelesen werden. Zusätzlich stehen zwei Signale (FULL und RST) für die Komponente FRAME_CONTROL zur Verfügung.

Die Komponente FRAME_CONTROL ermittelt aus den Steuersignalen (CAM_PWDN, SOF) und dem Status des FIFOs den Zustand für das Interruptsignal, das mit dem DSP verbunden ist, sowie den Zustand des FIFO Reset Signals. Dabei werden die Synchronisationssignale (DMA_DONE und DMA_DONE_FLAG) mit berücksichtigt.

Der zweite Teil dieser Instanz ist lediglich zum Bereitstellen von Informationen über die empfangenen Bilder verantwortlich. In der CAMERA_STATUS Komponente werden die Eingangssignale auf Flankenwechsel hin überprüft und die Ausgangssignale entsprechend gesetzt. Die Übertragung eines neuen Bildes beginnt mit der fallenden Flanke des Signals CAM_VSYNC und wird durch aktivieren des SOF Signals (Start Of Frame) signalisiert. Die Signale SOL (Start Of Line) und EOL (End Of Line) werden bei der steigenden bzw. fallenden Flanke des CAM_HREF Signals gesetzt. Abschließend wird das Signal SOP (Start Of Pixel) bei der steigenden Flanke des Kamerataktes gesetzt.

Mit diesen vier Signalen und CAM_HREF kann die Auflösung des Bildes und die Bildwiederholungsrate durch die CAMERA_INFO Komponente bestimmt und vom DSP mit der Konfiguration der Kamera abgeglichen werden. So ist sichergestellt, dass die Kamera innerhalb der vorgegeben Parameter arbeitet.

4.3.5 Bildzeilenspeicher (FIFO)

Die Komponente CAMERA_FIFO (siehe Bild 4.7) besteht aus einem FWFT-FIFO (First Word Fall Through-FIFO) und einem Prozess zur Generierung des FULL Signals. Das FIFO wurde durch den in der Entwicklungsumgebung Quartus II enthaltenen Altera MegaWizzard Plug-In Manager erstellt. Es dient in erste Linie zum Zwischenspeichern der Bildpunkte. Darüber hinaus stellt es eine Verbindung zwischen den Taktdomänen der Kamera und des DSPs her, da diese mit unterschiedlichen Frequenzen arbeiten.

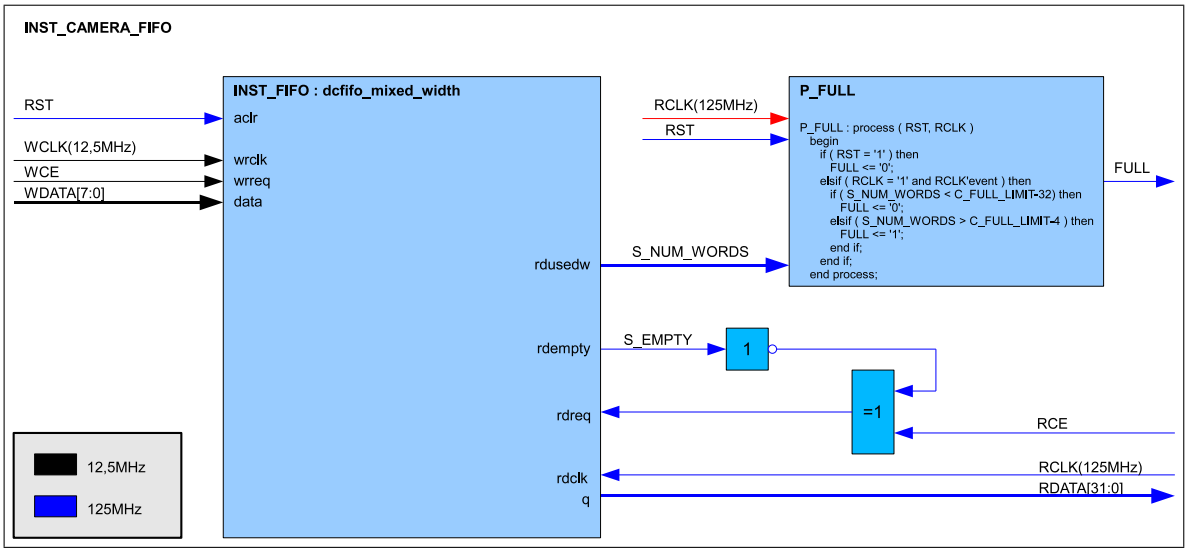


Bild 4.7: Blockdiagramm zu camera_fifo.vhd

In diesem Projekt wird ein FIFO mit einer Größe von 16384 Bit verwendet. Es können also maximal 2048 Pixel im FIFO zwischengespeichert werden. Diese Größe hat sich als geeignet ergeben, um eine stabile Bildübertragung zu gewährleisten. Über den Wizzard wurde das FIFO so konfiguriert, dass der Dateneingang 8 Bit und der Datenausgang 32 Bit breit ist, damit die Busbreiten mit der der Kamera und der des DSP übereinstimmen.

Neben dem Signal zum Auslesen der Daten aus dem FIFO steht noch ein Statussignal zur Verfügung. Dies enthält die Anzahl der momentan im FIFO vorhandenen 32 Bit Worte (S_NUM_WORDS). Der nachgeschaltete Prozess P_FULL ist für die Signalisierung zuständig, dass eine Bildzeile im FIFO enthalten ist. Sobald mehr als 156 Worte mit einer Breite von 32 Bit bzw. 624 Bildpunkten im FIFO vorhanden sind, wird das Signal FULL gesetzt, um die Latenzzeiten des EMIF und EDMA-Kontrollers auszugleichen. Werden die Daten aus dem FIFO ausgelesen und der Füllstand sinkt unter 512 Bildpunkte ab, wird das Signal zurückgesetzt.

Zur Sicherheit ist das Freigabesignal zum Lesen (RCE) mit dem Signal S_EMPTY verknüpft. Dadurch ist sichergestellt, dass nur dann aus dem FIFO gelesen wird, wenn Daten vorhanden sind. Beim Schreibzugriff ist diese Sicherheitsschaltung nicht eingebaut, da durch die Größe des FIFOs und dem Bandbreitenvorteil des EMIF die Wahrscheinlichkeit eines Überlaufens nur sehr gering ist. Davon abgesehen wird das FIFO mit Beginn eines neuen Bildes zurückgesetzt (siehe Kapitel 4.3.6).

4.3.6 Datensynchronisation

Die FRAME_CONTROL Komponente ist für die Synchronisation der Daten zwischen DSP und FIFO zuständig. Dafür sind zwei Signale (IRQ_DATA und FIFO_RST) nötig. Bild 4.8 zeigt den Aufbau der Komponente.

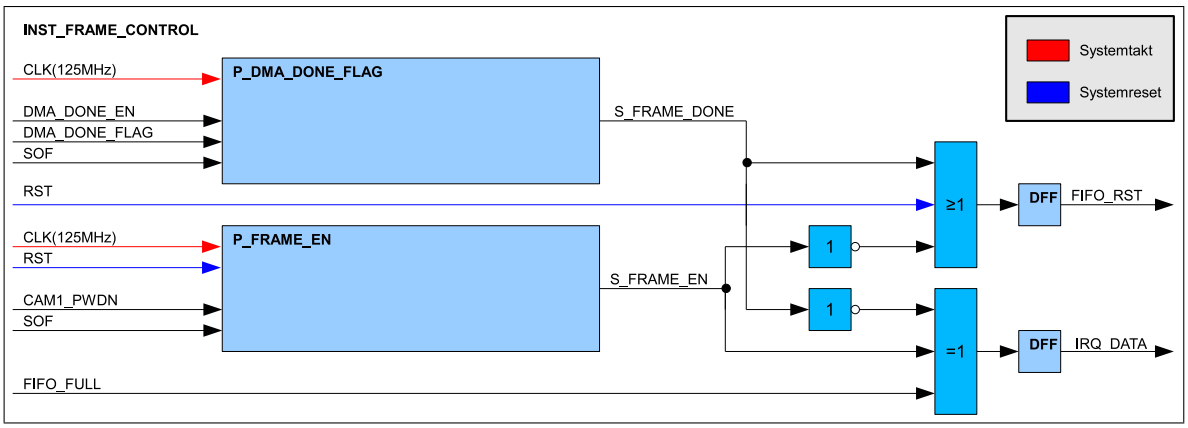


Bild 4.8: Blockdiagramm zu frame_control.vhd

Listing 4.1 zeigt den Quelltext für den Prozess P_FRAME_EN. Das Ausgangssignal wird gesetzt, sobald die Kamera nicht mehr im Power Down Modus ist und die erste fallende Flanke des VSYNC Signals empfangen wurde. Sollte die Kamera durch setzen des Power Down Bit gestoppt werden, so wird das Signal S_FRAME_EN gelöscht.

```
P_FRAME_EN : process ( RST, CLK )
begin
  if ( RST = '1' ) then
     S_FRAME_EN <= '0';
  elsif ( CLK = '1' and CLK'event ) then
     if ( CAM_PWDN = '1' ) then
        S_FRAME_EN <= '0';
     elsif ( SOF = '1' ) then -- falling edge of VSYNC
        S_FRAME_EN <= '1';
     end if;
  end if;
end process P_FRAME_EN;
```

Listing 4.1: P_FRAME_EN (frame_control.vhd)

Wenn der EDMA Kontroller ein komplettes Bild empfangen hat, wird in der zugehörigen Interruptroutine ein entsprechendes Bit gesetzt, um dem FPGA das Bildende zu signalisieren. Wird im Statusregister Bit 2 oder 3 gesetzt, je nachdem welche Kamera gemeint ist, so wird das Signal S_FRAME_DONE gesetzt (siehe Listing 4.2 und Bild 4.5). Erst wenn die Übertragung eines neuen Bildes durch die Kamera beginnt, wird dieses Signal zurückgesetzt.

```
P_DMA_DONE_FLAG : process ( CLK )
begin
  if ( CLK = '1' and CLK'event ) then
     if ( DMA_DONE_EN = '1' and DMA_DONE_FLAG = '1' ) then
        S_FRAME_DONE <= '1';
     elsif ( SOF = '1' ) then
        S_FRAME_DONE <= '0';
     end if;
  end if;
end process P_DMA_DONE_FLAG;
```

Listing 4.2: P_DMA_DONE_FLAG (frame_control.vhd)

Wie in Bild 4.8 und Listing 4.3 zu erkennen ist, werden aus den Signalen S_FRAME_EN und S_FRAME_DONE über logische Operationen die Signale FIFO_RST und IRQ_DATA gebildet. In Kapitel 5.2.5 wird das Zusammenspiel mit dem Prozessor noch einmal verdeutlicht.

```
FIFO_RST <= RST or ( not ( S_FRAME_EN ) ) or S_FRAME_DONE;

IRQ_DATA <= FIFO_FULL and S_FRAME_EN and ( not ( S_FRAME_DONE ) );
```

Listing 4.3: FIFO_RST und IRQ_DATA (frame_control.vhd)

4.3.7 Kamerastatus

Die CAMERA_STATUS Komponente ist ebenfalls Teil der übergeordneten CAM2EMIF Komponente. Die Steuersignale des Datenbusses der Kamera werden hier auf Flankenwechsel hin analysiert, um Informationen für das übertragene Bild zu erhalten. Bild 4.9 zeigt den Aufbau der Komponente. Um zum Beispiel die Anzahl der Bilder mit einem Zähler zu ermitteln, ist es nötig, den Anfang eines neuen Bildes zu erkennen. Wenn sich der Zustand des VSYNC Signals der Kamera von 1 auf 0 ändert, wird das SOF (Start Of Frame) Signal für einen Systemtakt gesetzt. Durch Zählen der Pulse des SOF Signals kann die Anzahl der Bilder bestimmt werden.

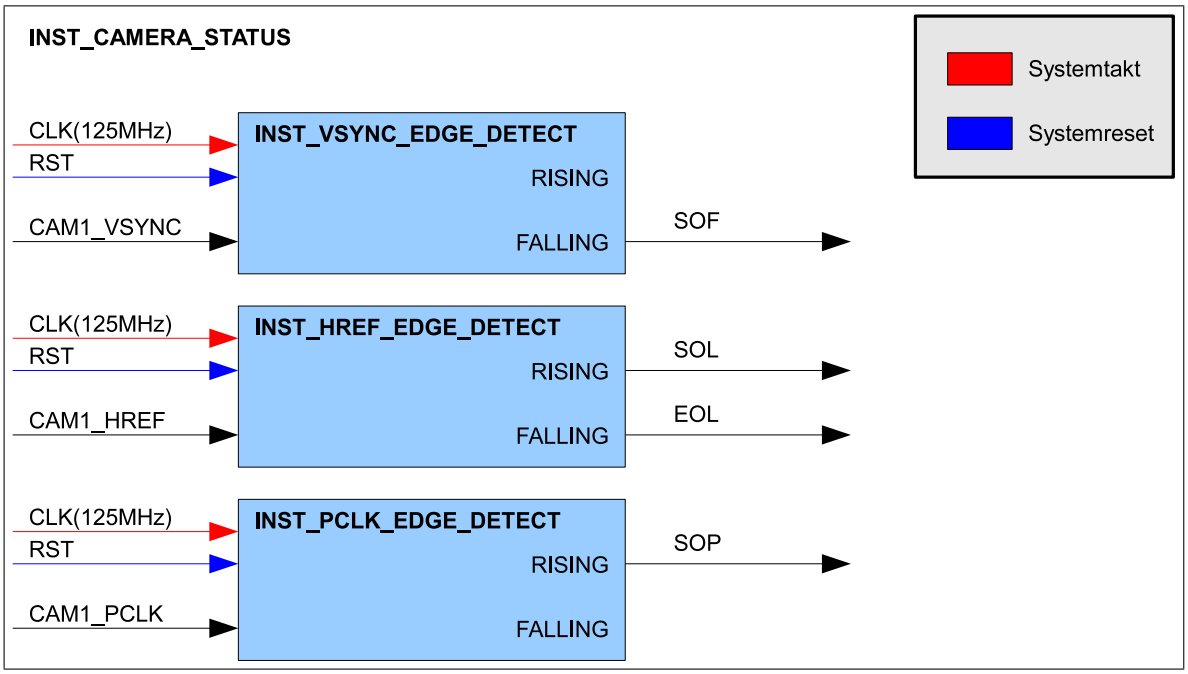


Bild 4.9: Blockdiagramm zu camera_status.vhd

Für die Anzahl der Zeilen in einem Bild wird das SOL (Start Of Line) benötigt. Dieses Signal wird mit der steigenden Flanke des HREF Signals der Kamera für einen Systemtakt gesetzt. Bei der fallenden Flanke des HREF Signals wird entsprechend das EOL (End Of Line) gesetzt. In Verbindung mit dem SOP (Start Of Pixel) Signal, welches bei jeder steigenden Flanke des Pixeltaktes (PCLK) gesetzt wird, kann die Anzahl der Bildpunkte pro Zeile ermittelt werden. Der genaue Aufbau der drei in Bild 4.9 gezeigten Instanzen zur Flankenerkennung wird im Folgenden Unterkapitel erläutert.

4.3.8 Flankenerkennung

Die in Bild 4.10 gezeigte Flankenerkennung besteht aus drei Teilen. Der erste Prozess P_SYNC beinhaltet eine Kette aus FlipFlops (FF). Die Anzahl der in Reihe geschalteten FFs richtet sich dabei nach einer globalen Konstanten, die mindestens den Wert 2 hat. Sollte der Fall auftreten, dass das Taktsignal und das Eingangssignal zum selben Zeitpunkt einen Flankenwechsel haben, so wird der Ausgang des FFs schwingen. Man spricht dabei von metastabilen Zuständen. Um diese zu vermeiden, wird ein weiteres FF dahinter geschaltet, da sich das erste FF in der Regel nach einer Taktperiode beruhigt.

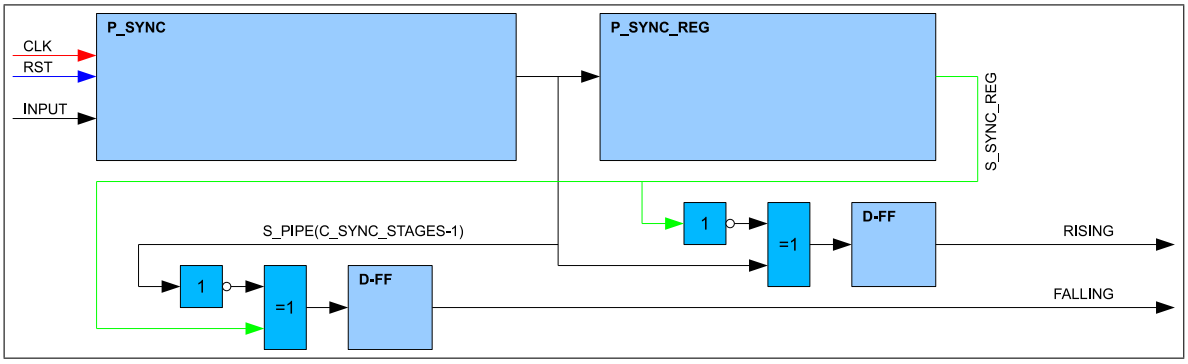


Bild 4.10: Blockdiagramm zu edge_detect.vhd

Nachdem die Eingangssignale auf den Systemtakt synchronisiert sind, wird das Signal im zweiten Prozess P_SYNC_REG um einen weiteren Takt verzögert. Dadurch kann im dritten Teil ein Flankenwechsel detektiert werden. Wenn das synchronisierte Eingangssignal 0 und das um einen Takt verzögerte Signal 1 ist, dann ist eine fallende Flanke beim Eingangssignal aufgetreten. Das Signal FALLING wird dann entsprechend im nächsten Systemtakt gesetzt. Die Ausgangssynchronisation stellt zwar eine weitere Verzögerung um einen Takt dar, die hier aber nicht ins Gewicht fällt, da die Eingangssignale eine deutlich niedrigere Frequenz im Vergleich zum Systemtakt haben. Der Vorteil ist allerdings, dass der kombinatorische Pfad, je nachdem, was für eine Hardware dahinter geschaltet ist, kürzer wird.

4.3.9 Bildinformationen

Anhand der Ausgangssignale der in Kapitel 4.3.7 gezeigten CAMERA_STATUS Komponente können in der CAMERA_INFO Komponente (Bild 4.11) die Bildinformationen ermittelt werden. Das SOF (Start Of Frame) Signal wird mit Beginn eines neuen Bildes gesetzt. Zu diesem Zeitpunkt wird ein Zähler (IMG_CNT) erhöht. Anhand des Zählerstandes kann jederzeit die Anzahl der Bilder seit dem Systemstart ermittelt werden. Mit einer Stopuhr im DSP kann dadurch die aktuelle Bildwiederholungsrate ermittelt werden.

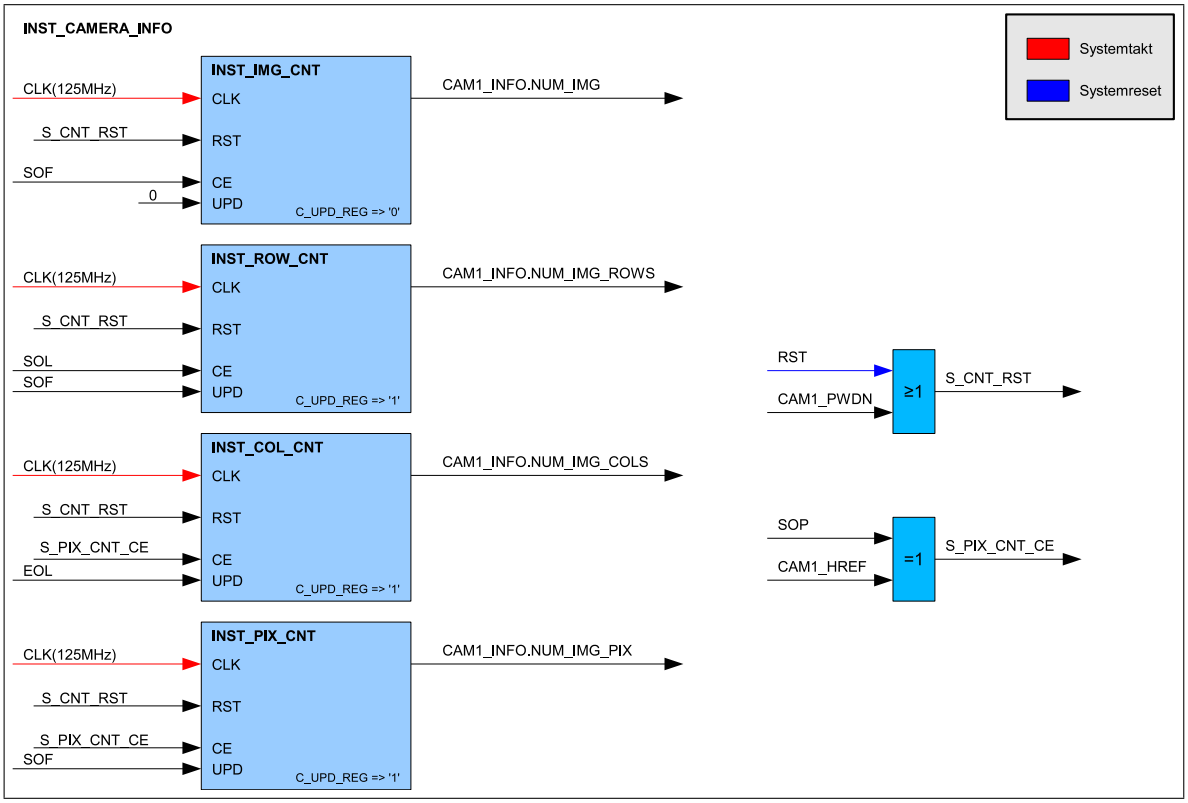


Bild 4.11: Blockdiagramm zu camera_info.vhd

Die übrigen drei Zähler haben eine Besonderheit. Es gibt ein zustätzliches Ausgangsregister, dass über das Eingangssignal UPD aktualisiert wird. Mit dem UPD Signal wird gleichzeitig der Zähler zurückgesetzt. Der Zeilenzähler (ROW_CNT) wird immer dann erhöht, wenn das SOL (StartOfLine) Signal gesetzt ist. Sobald das SOF (StartOfFrame) Signal gesetzt ist, liegt das Ergebnis am Ausgang an und der Zähler fängt wieder bei Null an. Der Zähler für die Bildspalten und Pixel pro Bild ist analog dazu aufgebaut.

Im Zusammenhang mit der vorgeschalteten CAMERA_STATUS Komponente aus Kapitel 4.3.7 stehen nun folgende Informationen über die übertragenen Bilder zur Verfügung. Zum einen ist die aktuelle Bildnummer vorhanden, zum anderen steht die Anzahl der Zeilen und Spalten, sowie die Anzahl der Pixel im Bild in einem entsprechenden Register bereit.

4.3.10 Schnittstelle zum DSP (EMIF)

Nun geht es darum, die bereits zwischengespeicherten Bildpunkte und Bildinformationen dem DSP bereitzustellen. Aus den Anforderungen in Kapitel 4.1 ist bereits ersichtlich geworden, das die Schnittstelle zum EMIF asynchron erfolgen muss und fünf Takte pro Zugriff benötigt.

Bei dieser asynchronen Übertragung werden die Freigabesignale zum Lesen (nRE) und Schreiben (nWE) jeweils für vier Takte auf Masse gelegt und anschließend wieder auf Versorgungspsannung. Das ChipSelect Signal bleibt dabei insgesamt fünf Takte aktiv. Eine detaillierte Darstellung der Signalverläufe ist im Zusammenhang mit der EMIF Konfiguration des DSP in Kapitel 5.2.2 zu finden.

Bild 4.12 zeigt den Aufbau des Gegenstücks zum EMIF des DSP. In diesem Fall handelt es sich um die Variante der Stereo-Plattform. Die Ethernet-Plattform ist vom Prinzip her genauso aufgebaut, lediglich die Signale für das I2C Interface und die zweite Kamera fehlen, der Rest ist gleich. Auch die Adressen, mit denen die einzelnen Register angesprochen werden, sind auf beiden Plattformen identisch.

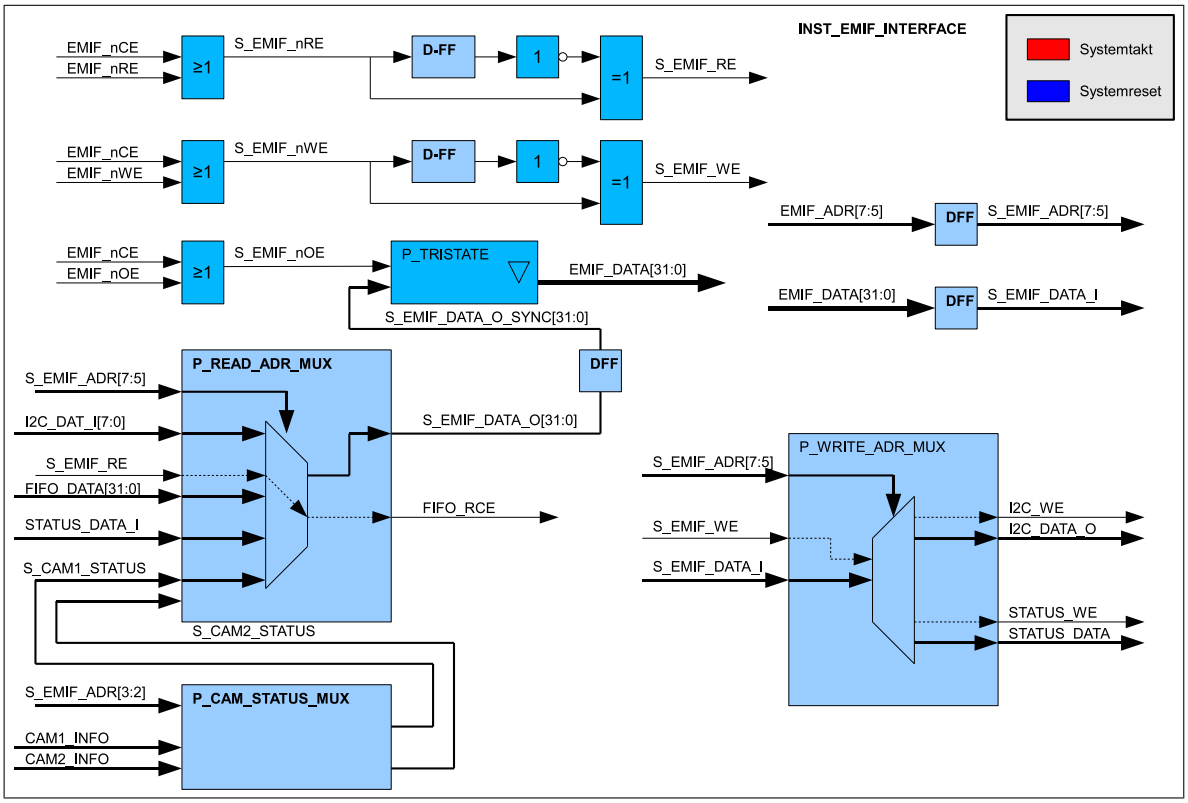


Bild 4.12: Blockdiagramm zu emif_inteface_stereo.vhd

Die Steuersignale nRE (not Read Enable), nWE (not Write Enable) und nOE (not Output Enable) können aufgrund des asynchronen Interface nicht direkt verwendet werden. Zunächst werden alle diese Signale mit dem Freigabesignal nCE (not Chip Enable) über eine OR-Verknüpfung verbunden. Da die Signale alle Low-aktiv sind, muss auf jeden Fall das nCE Signal aktiv sein, damit auf die Signale reagiert wird.

Nach der OR-Verknüpfung folgt eine Flankenerkennung für einen Pegelwechsel von eins nach null bei den Signalen für das Lesen und Schreiben. Das Freigabesignal für die Ausgänge (nOE) wird nach der Verschaltung mit dem nCE Signal direkt als Steuersignal für die Tristate-Treiber der Datenausgänge verwendet. Da das Businterface bidirektional ist, dürfen die Datenleitungen nur bei aktivem nOE Signal als Ausgang verwendet werden. Ansonsten entsteht unter Umständen ein Kurzschluss und die Halbleiter könnten beschädigt werden.

Bei einem Schreib- bzw. Lesezugriff wird in der hier verwendeten Konfiguration bei aktivem nCE Signal das entsprechende Signal (nWE oder nRE) für vier Takte auf Low-Pegel gesetzt und anschließend für einen Takt auf High-Pegel gehalten. Durch die eben beschriebene Schaltung wird direkt nach dem Erkennen der steigenden Flanke (Takt 5) das Signal S_EMIF_WE bzw. S_EMIF_RE für einen Takt gesetzt.

Der Schreibzugriff in ein bestimmtes Register erfolgt, indem durch den Adressmultiplexer das entsprechende Register anhand der anliegenden Adresse ausgewählt wird. Gleichzeitig wird das S_EMIF_WE Signal an das Register weitergeleitet, in das die Daten übernommen werden.

Zum Auslesen eines Registers bzw. des FIFOs ist ebenfalls ein Adressmultiplexer vorhanden. Die Auswahl erfolgt auch hier über die anliegende Adresse. Parallel zum S_EMIF_RE Signal muss das nOE Signal aktiv sein, damit das Ausgangssignal des Multiplexers auch am DSP anliegt. Das S_EMIF_RE Signal ist nur für das FIFO wichtig, da das gesetzte Signal dem FIFO als Lesebestätigung dient. Nachdem die Bestätigung beim FIFO eingetroffen ist, wird direkt das nächste Datenwort am Ausgang des FIFOs bereitgestellt und es kann erneut gelesen werden, bis das FIFO leer ist.

4.3.11 I2C

Bei der I2C Komponente handelt es sich um ein Open Source Projekt, welches aus dem Beispielprojekt der Firma Bitec übernommen wurde. Die vorhandene Schnittstelle wurde lediglich durch minimale Änderungen an die EMIF_INTERFACE Komponente angepasst.

4.3.12 Simulation

Um die Funktionalität der entworfenen Komponenten zu testen, wurde eine Testbench zur Simulation geschrieben. Die Testbench simuliert die Datenschnittstelle der Kameras und das EMIF des DSP. Eine Darstellung der Simulationsergebnisse erfolgt aufgrund der Komplexität des Entwurfs an dieser Stelle nicht. Alle Dateien, die für die Simulation benötigt werden, sind auf dem beiliegenden Datenträger enthalten (Anhang A).

4.3.13 Hardwareaufwand

Nachdem die Beschreibung der Hardware abgeschlossen ist, kann mit der Umsetzung begonnen werden. Eine Zusammenfassung der verwendeten Hardwareressourcen der jeweiligen Plattformen ist in Tabelle 4.1 zu finden. Bei der Stereo-Plattform werden mehr als doppelt so viele logische Einheiten wie bei der Ethernet-Plattform benötigt. Dies ist auch nicht weiter verwunderlich, da die CAM2EMIF Komponente doppelt vorhanden ist. Außerdem ist in der Stereo-Plattform eine I2C Komponente enthalten und die Schnittstelle zum EMIF ist durch die zweite Kamera ebenfalls aufwändiger.

Tabelle 4.1: Hardwareaufwand

	Stereo-Plattform	Ethernet-Plattform
logische Einheiten (LE)	954 (12%)	392 (5%)
Ein- und Ausgänge	98 (71%)	97 (70%)
Eingebettete Multiplizierer	0 (0%)	0 (0%)
Speicher Bits	32768 (20%)	16384 (10%)
Phase-locked loop	1 (50%)	2 (100%)
maximale Taktfrequenz (EMIF Takt)	128.24 MHz	131.68 MHz

Die Anzahl der Ein- und Ausgänge unterscheidet sich lediglich um ein einziges Signal. Dabei handelt es sich um den externen Interrupt, der auf der Stereo-Plattform für die zweite Kamera verwendet wird. Beim Speicherbedarf ist zu erkennen, dass auf der Ethernet-Plattform 16384 Bit verwendet werden, dies entspricht der Größe des FIFO. Auf der Stereo-Plattform ist der Bedarf durch die zweite Kamera entsprechend doppelt so groß. Auch wenn noch deutlich mehr Speicher im FPGA vorhanden ist, so können die FIFOs nicht weiter vergrößert werden ohne die maximale Taktfrequenz zu reduzieren.

Bei der hier gewählten Speicherkapazität der FIFOs handelt es sich um die maximale Größe, bei der die maximal zulässige Taktfrequenz oberhalb des verwendeten EMIF Taktes von 125 MHz liegt.

5 Softwarekonfiguration

In diesem Kapitel werden die Voraussetzungen für die Programmierung der Bildverarbeitungsalgorithmen aus Kapitel 2 geschaffen. Ausgehend von der Hardwarekonfiguration aus Kapitel 4 werden die grundlegenden Funktionen und Tasks dargestellt, die nötig sind, um die Kommunikation zwischen den einzelnen Komponenten herzustellen. Zunächst wird die Initialisierung der Kameras anhand der verwendeten Parameter erläutert. Anschließend folgt die Konfiguration des DSPs. Zuletzt wird der Programmablauf der implementierten Tasks beschrieben.

Bis auf einen kleinen Anteil an Funktionen und Tasks wird auf beiden Plattformen der selbe Programmcode ausgeführt. Die Unterscheidung zwischen den Plattformen erfolgt dabei durch eine Konstante, die in den Projekteinstellungen definiert ist. An der Bezeichnung der beiden DSPs ändert sich nichts im Vergleich zum vorangegangenen Kapitel.

5.1 Kamera

Für die Konfiguration der Kamera wurden die Parameter aus dem Bespielprojekt der Firma Bitec übernommen. Die Kamera arbeitet bei dieser Konfiguration mit einer Auflösung von 640x480 Bildpunkten und einer Bildwiederholungsrate von 31.25 Hz. Es werden insgesamt 109 Parameter über das im FPGA implementierte I2C Interface an die Kameras übertragen. Von diesen Parametern sind jedoch nur ca. 30 im Datenblatt dokumentiert. Die Funktionen, die sich hinter den übrigen Einstellungen verbergen, werden nicht erwähnt. Anscheinend hängen einige Parameter auch voneinander ab, so dass eine Änderung eines bekannten Parameters keine Funktionsänderung der Kamera nach sich zog.

Aufgrund der mangelnden Dokumentation können nur einige wenige Einstellungen geändert werden. Zum einen wurde der automatische Weißabgleich deaktiviert und der Pixeltakt wurde so eingestellt, dass dieser nur bei gesetztem HREF Signal aktiv ist. Ansonsten wurden die Einstellungen wie im Beispielprojekt belassen.

Eine Begründung für die Abschaltung des Weißabgleiches ist im Zusammenhang mit der Implementierung der Farbklassifikation in Kapitel 6.3 zu finden.

5.2 DSP Peripherie Konfiguration

Dieses Kapitel beschäftigt sich mit der Konfiguration der verwendeten Peripherie zur Kommunikation mit dem FPGA und zwischen den DSPs. Angefangen wird mit den grundlegenden Einstellungen für das DSP/BIOS, über die Konfiguration des EMIF bis hin zum Auslesen der Bilddaten aus dem FIFO mit Hilfe des EDMA Kontrollers.

5.2.1 DSP/BIOS

Als Grundlage für die Konfiguration des DSP/BIOS wurden die Einstellungen aus einem Beispielprojekt der Firma Bitec übernommen und angepasst. Zunächst wird über die graphische Benutzeroberflöche die Interruptpolarität für die externen Interrupts 4 bis 7 auf die steigende Flanke eingestellt. Außerdem wird die Ticksize, mit der der Scheduler die Tasks zuteilt, auf 100 μs festgelegt. Als letzter Punkt wird die Größe des Stacks für die einzelnen Tasks auf 4096 Byte eingestellt und der 256 KB große L2 Cache aktiviert.

5.2.2 EMIF

Das EMIF arbeitet wie bereits erwähnt mit einem Takt von 125 MHz. Ebenfalls bekannt ist, dass für eine fehlerfreie Kommunikation fünf Takte für einen Schreib- oder Lesezugriff benötigt werden. Ein Datentransfer über das EMIF besteht bei asynchronem Interface immer aus den drei Phasen Setup, Strobe und Hold. Die Konfiguration erfolgt dabei für jeden Adressbereich unabhängig voneinander. Das FIFO und die übrige im FPGA implementierte Peripherie ist an CE2 angeschlossen. Die folgende Konfiguration bezieht sich daher nur auf diesen Adressbereich. Die Einstellungen für die Kommunikation mit dem Ethernet Interface, welches an CE3 angeschlossen ist, wird unverändert aus dem Beispielprojekt übernommen.

Zur Kommunikation mit dem FPGA wird das EMIF mit einer 0-4-1 Konfiguration verwendet. In diesem Fall ist das EMIF so eingestellt, dass bei jedem Zugriff kein Takt für Setup, vier Takte für Strobe und ein Takt für die Hold Phase verwendet werden. Zur Verdeutlichung ist in Bild 5.1 der Signalverlauf beispielhaft für einen Schreibzugriff über das EMIF mit einer 2-3-1 Konfiguration dargestellt.

Der Lese- und Schreibzugriff läuft nahezu identisch ab. Lediglich das ARE (asynchronous read enable) bzw. nRE Signal wird beim Lesezugriff aktiv anstatt des AWE (write enable) Signals. Außerdem wird das AOE (output enable) zum Umschalten der Signalrichtung parallel zum ARE aktiv, da die Setup-Zeit auf Null gesetzt ist. Die Bezeichnung der Signale ist hier anders, da damit Bezug auf den asynchronen Betrieb genommen wird.

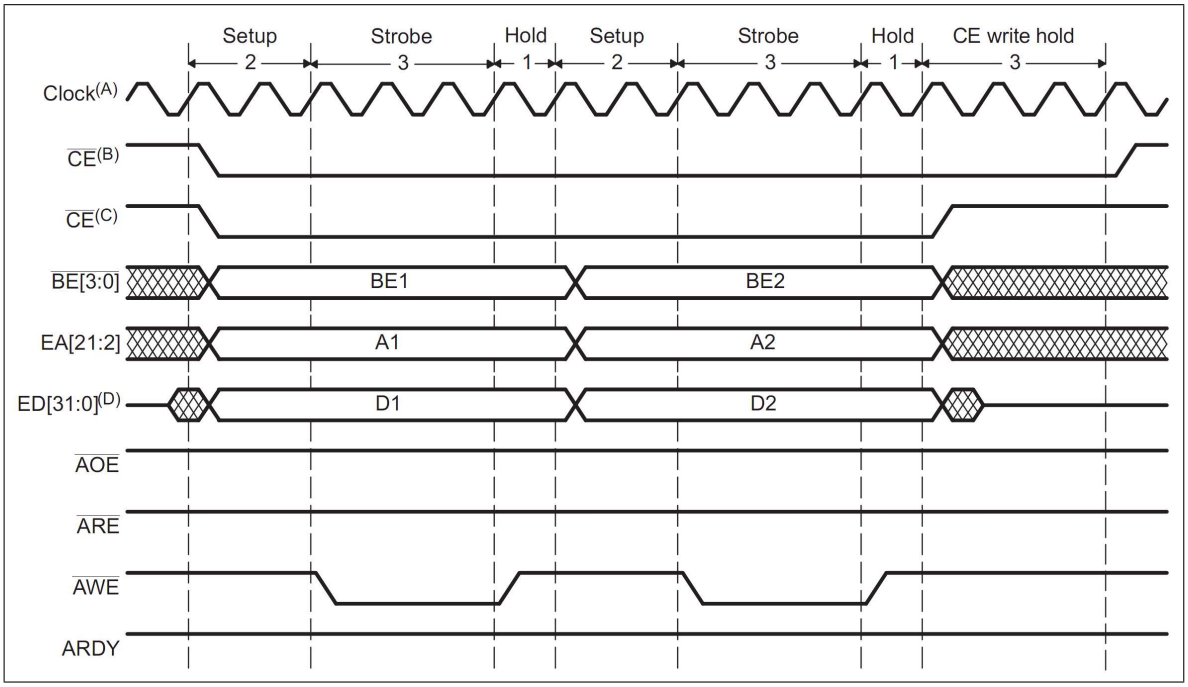


Bild 5.1: Asynchroner Schreibzugriff über das EMIF[7]

Die Schnittstelle im FPGA ist so aufgebaut, dass die anliegenden Daten mit der steigenden Flanke des AWE Signals übernommen werden. Die Vorlaufzeit von vier Takten ist nötig, damit die Daten stabil am FPGA anliegen. Beim Lesezugriff wird mit der steigenden Flanke des ARE Signals dem FIFO das Auslesen bestätigt und das nächste Datenwort wird bereitgestellt.

Im Beispielprojekt der Firma Bitec wurde eine 8-8-1 Konfiguration verwendet. Durch diese Optimierung auf eine 0-4-1 Konfiguration steigt der maximale Datendurchsatz von ca. 235 MBit/s auf 800 MBit/s, was deutlich mehr Flexibilität für die Umsetzung bringt.

5.2.3 DSP Kommunikation (McBSP)

Die Kommunikation zwischen den beiden DSPs erfolgt seriell über den McBSP2. Dabei wird die Schnittstelle für zwei Aufgaben verwendet. Zum einen synchronisieren sich die beiden DSPs darüber und zum anderen werden die durch die Bildverarbeitung ermittelten Ergebnisse von der Stereo-Plattform an die Ethernet-Plattform übertragen. Bei dieser Ergebnisübermittlung wird allerdings der EDMA Kontroller zur Hilfe genommen um den Prozessor zu entlasten (siehe Kapitel 5.2.5).

Damit die Kommunikation zwischen den beiden DSPs über McBSP funktioniert, müssen die Schnittstellen auf beiden DSPs identisch konfiguriert werden. Diese Aufgabe wird durch die Funktion *MCBSP2_init()* erledigt.

Die Übertragungsfrequenz wird auf 1 MHz und die Datenwortgröße auf 32 Bit eingestellt. Außerdem werden die Richtungen für die Signale festgelegt. Die Signale CLKX, FSX und DX sind für das Senden von Daten zuständig und daher als Ausgang konfiguriert. Die Signale CLKR, FSR und DR werden entsprechend als Eingang konfiguriert.

Der zugehörige Interrupt für den Datenempfang wird so eingestellt, dass dieser beim Eintreffen eines neuen Datenwortes auslöst. Da dieser Interrupt (RINT2) außerhalb der 16 möglichen Interrupts (siehe Bild 3.6) liegt, wird dieser auf den Interrupt für den nicht verwendeten McBSP0 gelegt.

Nachdem die Initialisierung der Schnittstelle auf beiden Plattformen abgeschlossen ist, können mit den folgenden entworfenen Funktionen Daten zwischen den beiden DSPs übertragen werden. Um Daten an den jeweils anderen DSP zu verschicken, wird die Funktion *MCBSP2_poll_write(Uint32 value)* mit dem zu übertragenden 32 Bit Wert als Übergabeparameter aufgerufen. Bevor der eigentliche Transfer beginnt, wird zunächst durch Polling abgefragt, ob die Schnittstelle bereit ist zum Senden. Anschließend werden die Daten übertragen.

Zum Empfangen von Datenwörtern wird ähnlich vorgegangen, jedoch sind hierfür zwei verschiedene Funktionen programmiert worden. Bevor Daten aus dem Empfangsregister des McBSP2 ausgelesen werden können, muss überprüft werden, ob dieses bereit ist. Dazu kann entweder die Funktion *MCBSP2_poll_read()* oder *MCBSP2_nb_read()* aufgerufen werden.

Die erste Funktion wartet solange, bis Daten empfangen wurden, dies kann unter Umständen dazu führen, dass die Funktion das Programm blockiert und endlos wartet, sofern keine Daten empfangen werden. Die zweite Funktion stellt dazu eine Alternative dar. Es wird geprüft, ob ein neues Datenwort vorhanden ist und wenn dem nicht der Fall ist, gibt die Funktion den Wert null zurück. Dadurch kann es nicht passieren, dass das Programm blockiert.

Damit diese Schnittstelle zur Synchronisation der beiden Plattformen verwendet werden kann, ist eine Interruptroutine und eine weitere Funktion nötig. Sobald ein neues Datenwort empfangen wurde, wird die Interruptroutine *MCBSP2_receive_isr()* aufgerufen und das Datenwort aus dem Empfangsregister in eine globale Variable gespeichert. Das Eintreffen neuer Daten wird durch anschließendes Setzen einer Semaphore signalisiert.

In Verbindung mit der Interruptroutine wird zur Verarbeitung der empfangenen Daten die Funktion *MCBSP2_wait_for_ack(Uint32 message, Uint32 timeout)* verwendet. Bild 5.2 zeigt den Aufbau der Funktion in Form eines Struktogrammes. Diese Funktion ist Hauptbestandteil des Synchronisationstasks aus Kapitel 5.3.2.

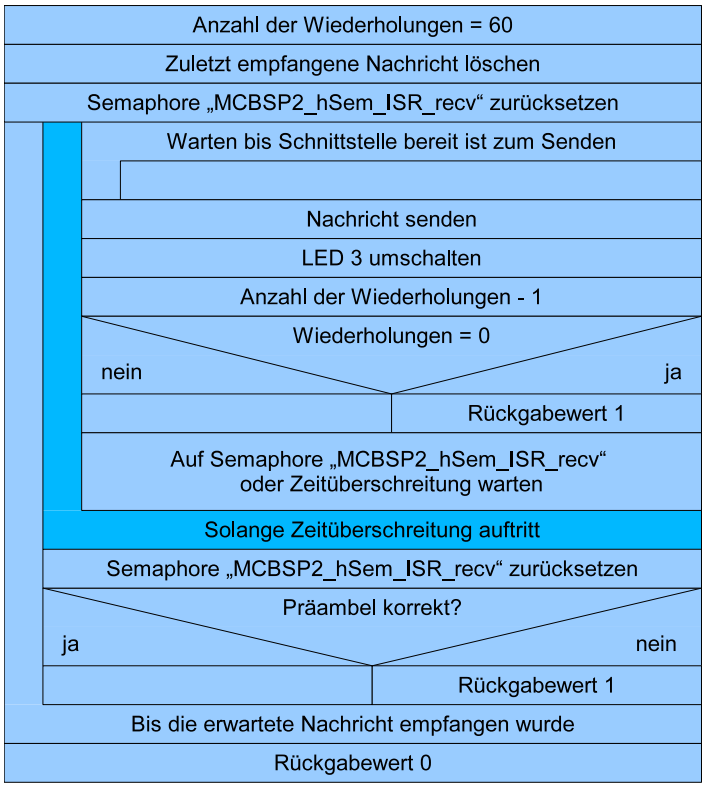


Bild 5.2: Struktogramm - MCBSP2_wait_for_ack()

Der Ablauf der Funktion sieht wie folgt aus: Zunächst wird die zuletzt erhaltene Nachricht durch Überschreiben des Inhaltes der globalen Variablen gelöscht und die Semaphore zurückgesetzt. Anschließend wird darauf gewartet, dass die Schnittstelle bereit ist zum Senden und die übergebene Nachricht wird übertragen. Mit jeder Übertragung der Nachricht wird die Anzahl der Wiederholungen verringert. Ist innerhalb dieser Wiederholungen keine Antwort eingetroffen wird die Funktion beendet, um das übergeordnete Programm nicht zu blockieren.

Nach dem Senden der Nachricht wird auf die Semaphore gewartet, die durch die Interruptroutine beim Empfang neuer Daten gesetzt wird. Sollte die Semaphore nicht innerhalb der durch einen Übergabeparameter festgelegten Zeit gesetzt werden, so wird die Nachricht erneut gesendet.

Wird die Semaphore innerhalb der definierten Zeitüberschreitung gesetzt, so wird diese zurückgesetzt und die Präambel der Nachricht wird überprüft. Bei der Präambel handelt es sich um die oberen 16 Bit der 32 Bit Nachricht. Alle zugelassenen Nachrichten sind in Form von Konstanten definiert und zwar so, dass die oberen 16 Bit bei allen Nachrichten identisch sind. Sollte die Präambel nicht korrekt sein, so wird diese Funktion mit einem Fehler beendet.

Ist die Präambel allerdings korrekt, wird der Rest der Nachricht dahingehend überprüft, ob er mit der gesendeten Nachricht übereinstimmt. Wenn dies der Fall ist wird die Funktion erfolgreich beendet, ansonsten wird die ursprüngliche Nachricht erneut gesendet und der beschriebene Ablauf beginnt von vorn.

Zusammenfassend ist zu sagen, dass diese Funktion eine Nachricht an den anderen DSP übermittelt und die selbe Nachricht als Antwort erwartet. Diese Funktion wird nur unter der Bedingung erfolgreich durchlaufen, dass sie auf beiden DSPs mit der gleichen Nachricht aufgerufen wird. Außerdem ist durch die Definition einer Zeitüberschreitung und der Begrenzung der Wiederholungen gewährleistet, dass diese Funktion den Prozessor nicht blockieren kann, falls der Andere gerade nicht antwortet.

5.2.4 Light weight IP Stack

Bei dem LwIP Stack handelt es sich um ein Open Source Softwareprojekt, dass eine standardisierte Schnittstelle für die Kommunikation über Ethernet bereitstellt. Der von der Firma Bitec für die verwendete Hardware angepasste LwIP Stack aus dem Beispielprojekt wurde für diese Arbeit übernommen und in einigen Teilen angepasst.

Die Initialisierung erfolgt über die Funktion *lwIP_NetStart()*. Bisher wurde die IP-Adresse durch diese Funktion fest vergeben. Sie wurde so angepasst, dass die IP-Adresse als Parameter übergeben wird. Dadurch ist es möglich, die gewünschte IP-Adresse jederzeit zu ändern ohne den LwIP Stack jedes mal neu zu compilieren.

Außerdem ist der Interrupt des Ethernet IC auf den externen Interrupt 5 konfiguriert. In dieser Arbeit wurde der Interrupt allerdings hardwareseitig mit dem externen Interrupt 7 verbunden. Daher muss der LwIP-Stack an mehreren Stellen angepasst werden. Die Veränderungen im Quelltext sind durch den Kommentar *INT7* gekennzeichnet und befinden sich in den Dateien *lwIP_NetStart.c*, *lwIP_NetStart.asm*, *ax88180if.h* und *ax88180if.asm*.

5.2.5 EDMA

Nachdem die Kommunikation über McBSP zwischen den DSPs geklärt ist, folgt in diesem Unterkapitel die Konfiguration des EDMA Kontrollers. Dieser wird zum einen für die Übertragung der Ergebnisse der Bildverarbeitung über McBSP und zum anderen für den Transfer der Bilddaten aus dem FIFO des FPGA über das EMIF eingesetzt.

Zwischen den DSPs erfolgt der Datenaustausch nach der Synchronisation in Form von Paketen. Von der Stereo-Plattform werden Pakete mit einer Größe von 16 Datenwörtern mit jeweils 32 Bit an die Ethernet-Plattform übertragen. Darin enthalten sind die Ergebnisse der Bildverarbeitung. In der anderen Richtung wird ein Paket mit vier Elementen versendet. Dabei handelt es sich um die Schwellwerte für die Farbklassifikation.

Würde man diese Daten ohne EDMA Kontroller übertragen, so müsste der Prozessor die Datenwörter einzeln auslesen und dazwischen jeweils den Status des Empfangsregisters prüfen. Verwendet man den zugehörigen Interrupt, so würde dieser nach dem Versenden eines Datenwortes ausgelöst. Bei einem Datenpaket von 16 Worten würde die Interruptroutine 16 mal aufgerufen werden. In beiden Fällen müsste der Prozessor dafür unnötig Zeit aufwenden.

Als Grundlage für die Konfiguration wird die Beispielkonfiguration aus der Dokumentation von Texas Instruments verwendet (siehe [17]). Der EDMA Kontroller wird so konfiguriert, dass die Übertragung des Paketes mit einem kurzen Funktionsaufruf durch den Prozessor gestartet wird. Der Prozessor hat anschließend mit der Übertragung nichts mehr zu tun. Für den Datenempfang sieht die Konfiguration so aus, dass der Prozessor nur einen Interrupt beim Erhalt eines neuen Paketes erhält und nicht bei jedem Datenwort.

Die Übertragung der Daten erfolgt durch Aufruf der Funktion *MCBSP2_edma_xmit()* mit einem Zeiger auf die zu übertragenden Daten als Übergabeparameter. Auf der anderen Plattform werden die Daten durch den EDMA Kontroller in den Speicher kopiert und nach Erhalt eines Datenpaketes wird ein Interrupt ausgelöst. Die Interruptroutine ist dann dafür zuständig einen neuen Speicherbereich für das folgende Datenpaket bereitzustellen. Dabei wird eine Dreifachpufferung verwendet, damit keine Daten verloren gehen (siehe Kapitel 5.2.6). Außerdem wird eine Semaphore erhöht. Die Funktion *MCBSP2_getData()* gibt daraufhin einen Zeiger auf das zuletzt empfangene Datenpaket zurück.

Bei der EDMA Konfiguration für den Datenempfang der Kameras wird ähnlich vorgegangen. Sobald ein FIFO im FPGA den nötigen Füllstand von 640 Pixeln erreicht hat, wird ein Interrupt ausgelöst. Im Vergleich zur McBSP EDMA Konfiguration können in diesem Fall 160 Datenwörter mit jeweils 32 Bit ausgelesen werden. Daher sieht die Konfiguration etwas

anders aus. Der Transfer wird so eingestellt, dass bei jedem Interrupt 160 Datenwörter ausgelesen werden, was genau einer Bildzeile entspricht. Sobald 480 Zeilen durch den EDMA Kontroller aus dem FIFO ausgelesen wurden, wird der EDMA Interrupt ausgelöst.

In der zugehörigen Interruptroutine wird ebenfalls eine Dreifachpufferung und eine Semaphore zur Synchronisation mit dem Prozessor verwendet. Dabei kam es hin und wieder vor, dass der EDMA Kontroller einen Interrupt nicht erkannt hat und das Bild im Speicher verschoben ist. In dem Fall ist eine Bildverarbeitung unmöglich. Da der EDMA Kontroller dieses Problem nicht erkennen kann, sind die folgenden Bilder ebenfalls im Speicher verschoben. Aus diesem Grund wird innerhalb der Interruptroutine durch Setzten des jeweiligen Bits im Statusregister das FIFO zurückgesetzt (siehe Kapitel 4.3.6).

Nachdem die Interruptroutine beendet ist, wartet der EDMA Kontroller auf das nächste Bild und das Reset Signal des FIFO ist solange aktiv bis die Kamera ein neues Bild überträgt (fallende Flanke beim VSYNC Signal). Dadurch ist sichergestellt, dass die Bilddaten der Kameras über das FIFO und den EDMA Kontroller vollständig im Speicher des DSP ankommen.

Durch den Einsatz des EDMA Kontrollers verringert sich das Interruptaufkommen durch die Kameras von 480 auf einen pro empfangenem Bild und Kamera. Darüber hinaus muss der Prozessor die Daten auch nicht aus dem FIFO auslesen, was ebenfalls eine erhebliche Entlastung darstellt.

Abgesehen von den recht kurzen EDMA Interruptroutinen steht dem Prozessor die übrige Zeit zwischen zwei Bildern für die Abarbeitung des Bildverarbeitungsalgorithmus zur Verfügung. Ohne den Einsatz des EDMA Kontrollers wäre ein Betrieb in Echzeit wahrscheinlich gar nicht erst möglich, da die CPU permanent mit dem Datentransfer beschäftigt wäre.

5.2.6 Dreifachpufferung

Verwendet man eine Doppelpufferung, bei der der Prozessor die Daten aus einem Puffer bearbeitet während der EDMA Kontroller neue Daten in einen anderen Puffer schreibt, so kann es zu Datenverlusten kommen. Ein Datenverlust tritt immer dann auf, wenn der Prozessor zur Verarbeitung der Daten mehr Zeit benötigt als der EDMA Kontroller zum Bereitstellen der Daten. Um dieses Problem zu beheben, wird eine Dreifachpufferung verwendet. Dabei wird so vorgegangen, dass drei Puffer für die Speicherung der Daten bereitstehen.

In Bild 5.3 sind die drei möglichen Fälle dargestellt. Im ersten Fall verarbeitet der Prozessor die Daten aus dem ersten Puffer während der EDMA Kontroller die ankommenden Daten abwechselnd in den zweiten und dritten Puffer speichert. Sobald der Prozessor ein neues Datenpaket oder Bild verarbeiten möchte, werden die Puffer, wie im zweiten Fall zu sehen ist, getauscht. Jetzt verwendet der Prozessor die Daten aus dem zweiten Puffer und der EDMA Kontroller wechselt zwischen den anderen beiden Speicherbereichen hin und her.

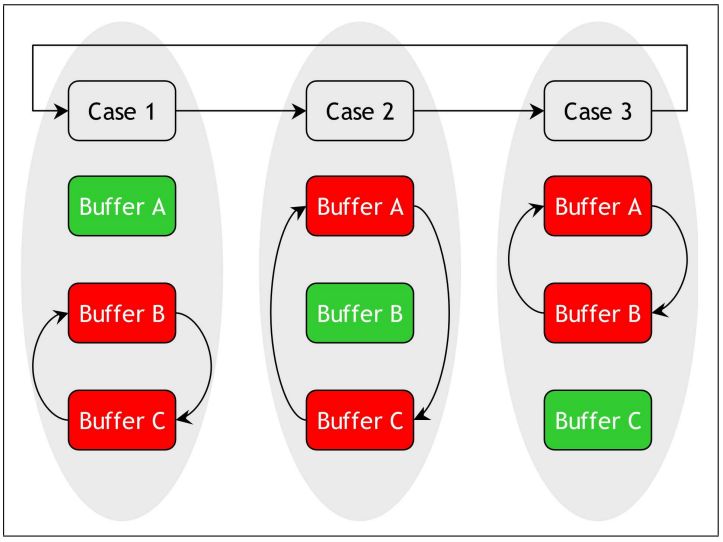


Bild 5.3: Dreifachpufferung[9]

Dadurch ist sichergestellt, dass kein Datenverlust auftritt wenn der EDMA Kontroller zwischen den Puffern wechselt, unabhängig davon wie lange der Prozessor für die Verarbeitung der Daten benötigt. Da die Algorithmen laut Aufgabenstellung in Echtzeit ablaufen sollen, ist diese Dreifachpufferung eigentlich nicht nötig, dennoch ist dieser Aufbau während der Entwicklung sehr hilfreich um die Funktion der Algorithmen zu testen. Im späteren Betrieb entsteht dadurch bis auf den höheren Speicherbedarf kein Nachteil.

5.3 Programmablauf

In diesem Kapitel wird der Programmablauf auf den beiden DSPs beschrieben. Der Ablauf wird dabei anhand von Struktogrammen erläutert. Die Programmteile, die auf beiden Plattformen in identischer Form ablaufen, sind blau gekennzeichnet. Die Funktionen, die nur auf der Stereo-Plattform ausgeführt werden, sind grau dargestellt. Spezifische Funktionen der Ethernet-Plattform sind gelb dargestellt.

Der Programmablauf ist in Bild 5.4 zu erkennen. Als erstes wird die Initialisierung durchgeführt. Dabei wird die Peripherie des DSP konfiguriert. Darüber hinaus werden die benötigten Semaphoren zur Synchronisation der Tasks, sowie die Tasks selber erstellt. Auf beiden Plattformen werden zwei Tasks ausgeführt. Der erste Task ist für die Synchronisation der beiden Plattformen zuständig, der zweite für die Bildverarbeitung. Auf der Ethernet-Plattform wird ein dritter Task erstellt, der die Ethernetverbindung verwaltet.

Nach der Initialisierung arbeitet zunächst nur der Synchronisationstask, während der Bildverarbeitungstask auf eine Semaphore wartet. Sobald die Synchronisation der DSPs abgeschlossen ist, wird die Semaphore, auf die der Bildverarbeitungstask wartet, gesetzt. Die Bildverarbeitung wird anschließend ausgeführt. Der Synchronsiationstask wird in der Zwischenzeit durch eine weitere Semaphore blockiert. Sollte eine der Kameras nicht mehr reagieren, oder die Paketübermittlung zwischen den DSPs fehlerhaft sein, wird der Bildverarbeitungstask den Synchronisationstask wieder freigeben und sich selbst blockieren.

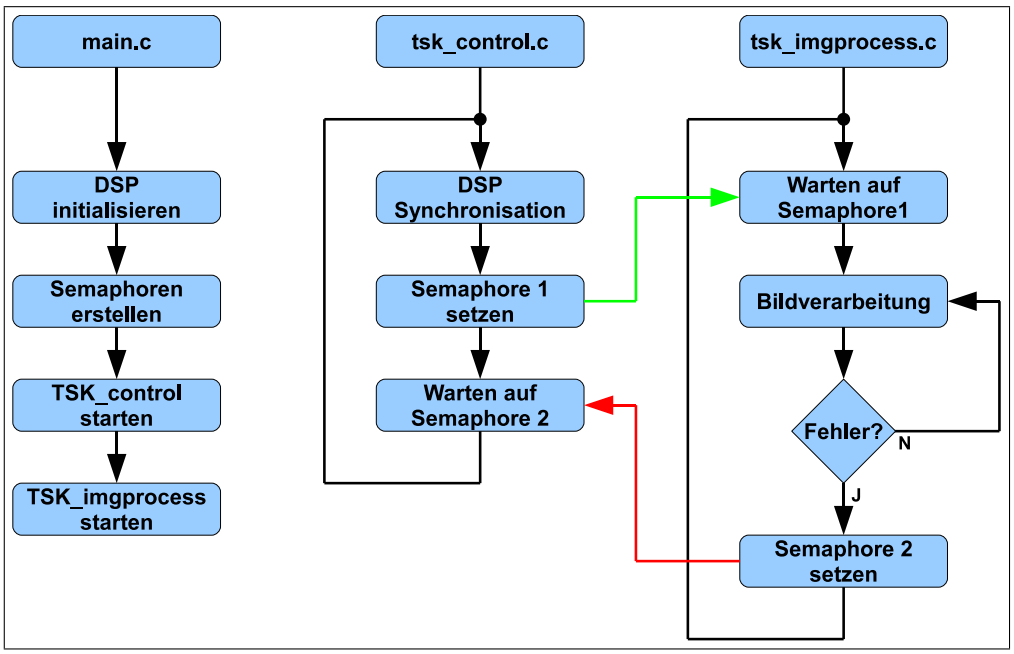


Bild 5.4: Programmablauf

5.3.1 Initialisierung

Das in Bild 5.5 dargestellte Struktogramm zeigt den Ablauf der Initialisierung. Als erstes wird die Systembibliothek des DSP initialisiert und das EMIF konfiguriert, anschließend werden die Kameras zurückgesetzt. Die LEDs und DIP-Schalter werden in dieser Arbeit zu Testzwecken verwendet und daher ebenfalls initialisiert.

Die Funktion *I2C_init()* wird nur auf der Stereo-Plattform ausgeführt, da nur auf dieser Plattform die zugehörige Hardware vorhanden ist. Die anschließende Konfiguration des McBSP2 und EDMA für die Kameras wird wieder auf beiden Plattformen in identischer Form durchgeführt. Außerdem wird der Interrupt des EDMA Kontrollers mit dem Interrupthandler des DSP/BIOS verbunden.

Nachdem die Konfiguration abgeschlossen ist, werden ein paar Daten über McBSP2 versendet, damit der Inhalt des Empfangsregisters bei dem anderen DSP überschrieben wird.

Sollte beim Anlegen der benötigten Semaphoren ein Fehler auftreten, wird der DSP angehalten. Anderenfalls werden im Anschluss daran die Tasks gestartet. Der Task für die Ethernetverbindung wird dabei nur auf der Ethernet-Plattform erstellt.

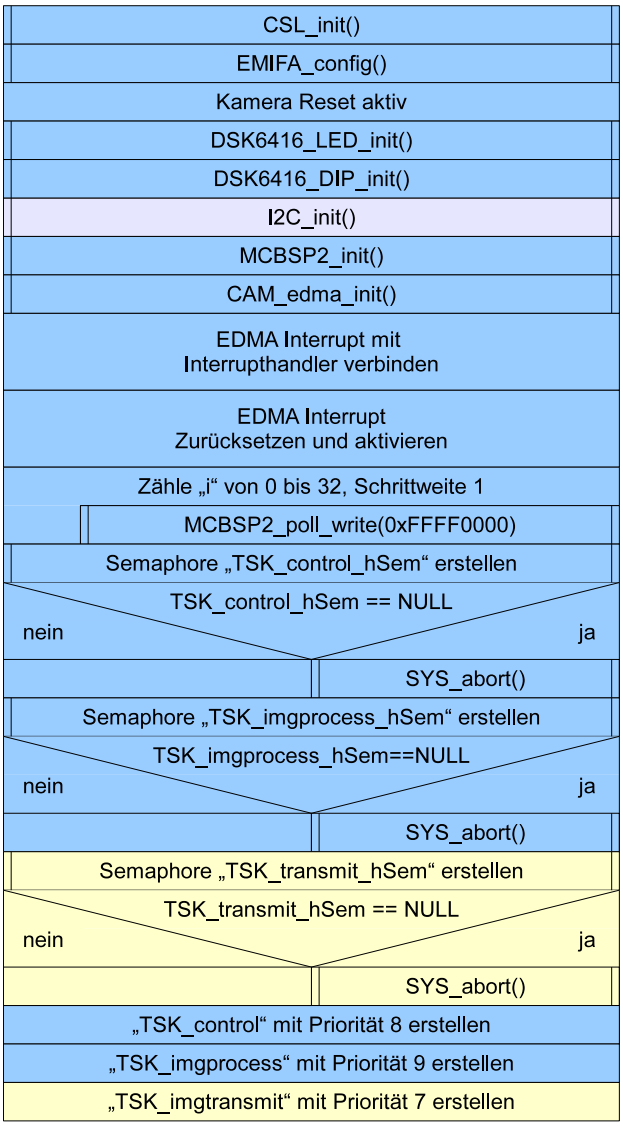


Bild 5.5: Initialisierung (main.c)

Der Bildverarbeitungstask erledigt die Hauptaufgabe des Systems und wird mit der höchsten Priorität ausgeführt. Sollte der Synchronisationstask zufällig auch noch arbeiten, so wird dieser die Bildverarbeitung automatisch durch Abschalten der Kameras stoppen. Dies führt beim Bildverarbeitungstask zu einem Fehler und der Task blockiert sich selbst.

5.3.2 Synchronisationstask

In diesem Abschnitt folgt die Erläuterung des Synchronisationstask, der für die Synchronisation der Plattformen zuständig ist. Da dieser Task etwas komplexer ist, fällt das Struktogramm entsprechend aufwendig aus. In Bild 5.6 ist der erste Teil des Task dargestellt.

Als erstes wird der Name des Task ausgegeben und die LED3 eingeschaltet. Dies dient lediglich dazu, dass man erkennen kann ob der Task arbeitet. Die Synchronisation ist wie ein Zustandsautomat aufgebaut, bei dem die einzelnen Zustände nacheinander durchlaufen werden. In jedem Zustand bitten die DSPs den jeweils anderen darum in den Folgezustand zu wechseln. Dies wird durch Aufruf der Funktion *MCBSP2_wait_for_ack()* realisiert, indem der Folgezustand als Nachricht übergeben wird. Erst nachdem eine Antwort erfolgt ist, wird die eigentliche Aufgabe des Zustandes erledigt. So ist sichergestellt, dass sich beide DSPs im selben Zustand befinden.

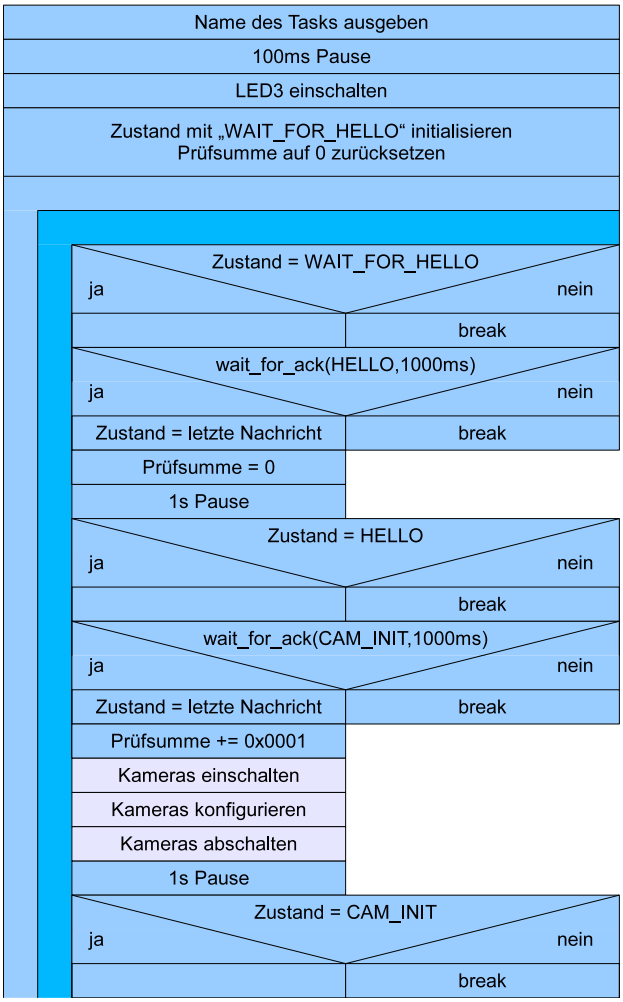


Bild 5.6: Synchronisationstask - Teil 1

Sollte bei dem Frage- und Antwortspiel ein Fehler auftreten, so wird in den Anfangszustand zurückgekehrt und die Synchronisation wird erneut begonnen. Im ersten Zustand (WAIT_FOR_HELLO) wird auf ein Lebenszeichen des anderen DSP gewartet. Da dieser Task auf beiden Plattformen ausgeführt wird, sollten nach dem Start beide DSPs auf die Nachricht „HELLO“ warten und eine Antwort mit dem gleichen Inhalt erhalten. Wenn das soweit geklappt hat, wird eine Prüfsumme initialisiert und der Folgezustand wird eingenommen.

Nachdem sich die beiden DSPs begrüßt haben, wird im folgenden Zustand (HELLO) auf das Startsignal (CAM_INIT) für die Konfiguration der Kamera gewartet. Anschließend erfolgt die Konfiguration über die I2C Schnittstelle im Stereo-FPGA.

Die Kameras befinden sich im Anschluß an die Konfiguration im Power Down Modus, so dass keine Bilddaten gesendet werden. Im Folgezustand (siehe Bild 5.7) können dann ohne Probleme die EDMA Kanäle für den Transfer der Bilddaten aktiviert werden. Anschließend folgt im nächsten Zustand die Aktivierung der Kameras. Ab diesem Zeitpunkt liefern die Kameras kontinuierlich Bilddaten und der EDMA Kontroller übernimmt die Daten aus dem FIFO in den Speicher des DSPs.

Da die Kameras jetzt arbeiten, werden zunächst die aktuellen Bildnummern aus den entsprechenden Registern im FPGA ausgelesen. Anschließend wird eine definierte Zeit gewartet und die Bildnummern werden erneut ausgelesen. Anhand der Bildnummern im Zusammenhang mit der verstrichenen Zeit wird die Bildwiederholungsrate errechnet. Zusätzlich werden die übrigen Register mit Statusinformationen wie Bildzeilen, -spalten und Pixel pro Bild ausgelesen.

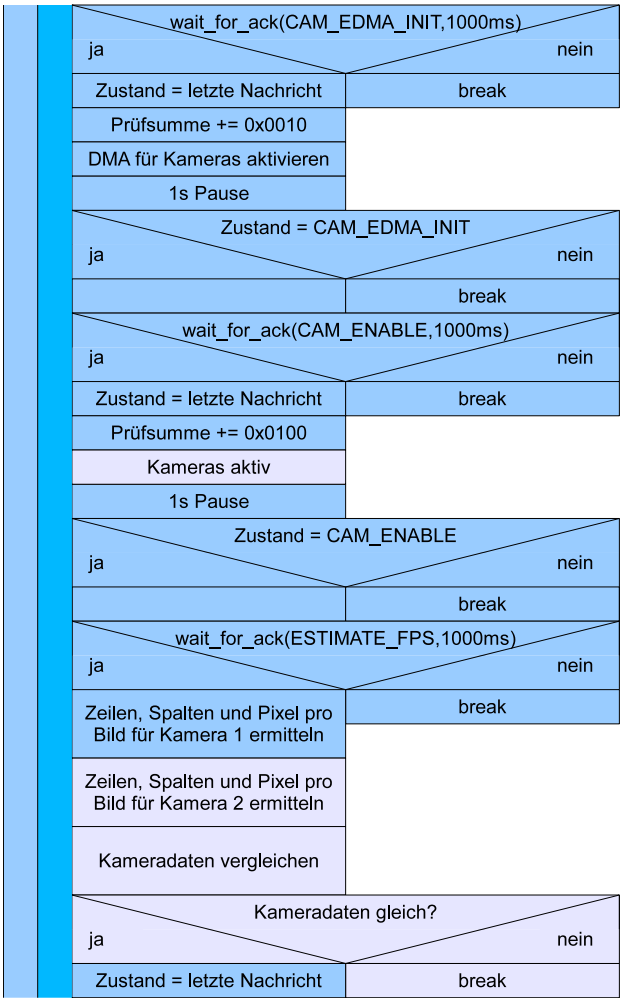


Bild 5.7: Synchronisationstask - Teil 2

Auf der Stereo-Plattform werden die Daten der beiden Kameras miteinander verglichen und auf Übereinstimmung geprüft. Sofern die beiden Kameras die selben Bildformate liefern, ist die Synchronisation so gut wie abgeschlossen und der Bildverarbeitungstask kann im Prinzip freigegeben werden.

Zuvor müssen jedoch noch ein paar Kleinigkeiten erledigt werden. In Bild 5.8 ist der letzte Teil des Synchronisationstasks dargestellt. Zunächst wird noch die Prüfsumme überprüft, die in jedem vorangegangenen Zustand verändert wurde. Stimmt die Prüfsumme mit dem erwarteten Wert überein, so wurde die Abfolge der Zustände eingehalten. Im Fehlerfall wird die Synchronisation neu gestartet.

Anschließend wird der McBSP2 auf EDMA Transfer umgestellt, damit der Datentransfer paketorientiert über McBSP2 zwischen den DSPs abläuft. Da der Task gleich in den Ruhezustand übergeht, wird die LED3 ausgeschaltet. Der Zustand der LED3 wurde im Verlauf der Synchronisation bei jedem Versenden einer Nachricht durch die Funktion *MCBSP2_wait_for_ack()* verändert. Dadurch kann überprüft werden, ob der Prozessor noch aktiv ist oder nicht.

Die Synchronisation ist damit abgeschlossen und die Semaphore, auf die der Bildverarbeitungstask wartet, wird gesetzt. Anschließend wird auf eine Semaphore gewartet, die durch den Bildverarbeitungstask im Fehlerfall gesetzt wird. Dadurch ist dieser Task solange blockiert, bis beim Bildverarbeitungstask ein Fehler auftritt.

Sollte beim Bildverarbeitungstask ein Fehler auftreten und die Semaphore gesetzt werden, wird zunächst der EDMA Transfer über McBSP2 abgeschaltet und die Übertragung erfolgt wieder in 32 Bit Datenworten. Anschließend beginnt die Synchronisation mit dem ersten Zustand von vorne.

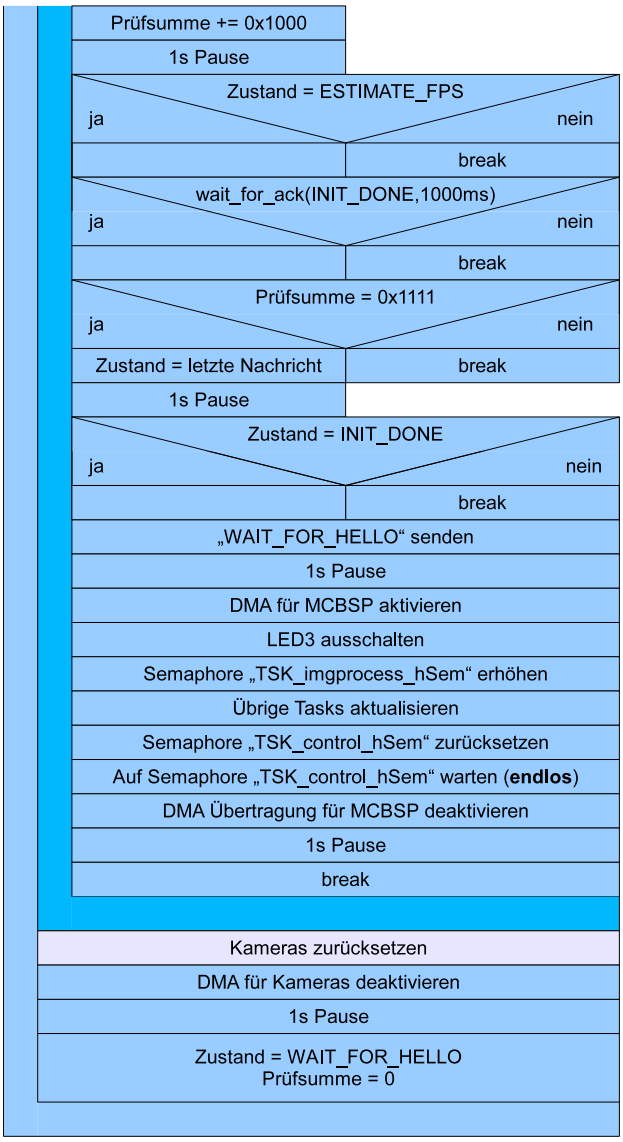


Bild 5.8: Synchronisationstask - Teil 3

Nachdem die Synchronisation geklärt ist, folgt nun die Erläuterung zum Bildverarbeitungstask im nächsten Unterkapitel.

5.3.3 Bildverarbeitungstask

Das in Bild 5.9 dargestellte Struktogramm zeigt den Ablauf des Bildverarbeitungstasks. Dieser Task stellt die Voraussetzung für die Implementierung des eigentlichen Bildverarbeitungsalgorithmus zur Verfügung. Zu Beginn wird, wie bereits in Kapitel 5.3 erwähnt, auf eine Semaphore gewartet, die durch den Synchronisationstask gesetzt wird. Nachdem die Freigabe erfolgt ist, wird in einer Endlosschleife die Bildverarbeitung durchführt.

Im ersten Schritt wird ein Zeiger auf das aktuelle Bild durch Aufruf der Funktion *CAM1_getFrame()* geholt. Auf der Stereo-Plattform wird dies zusätzlich für die zweite Kamera durchgeführt. Anschließend werden je nach Plattform entweder die Funktion *img_process_stereo()* oder *img_process_ethernet()* ausgeführt. Innerhalb dieser Funktion findet die eigentliche Bildverarbeitung statt, die im Verlauf von Kapitel 6 erläutert wird.

Nach Durchführung der Bildverarbeitung werden die Informationen zwischen den beiden Plattformen über McBSP ausgetauscht. Während die Bildverarbeitung und die anschließende Ergebnisübertragung durchgeführt wird, ist die LED2 eingeschaltet. Anhand der Einschaltdauer kann die Ausführungszeit gemessen werden, um eine Aussage über die Echtzeitfähigkeit zu treffen.

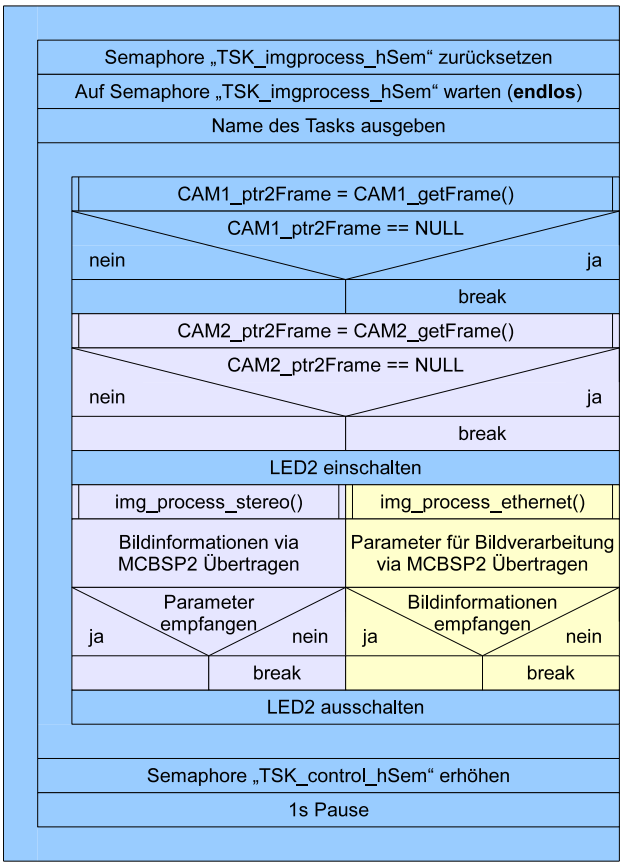


Bild 5.9: Bildverarbeitungstask

Die Ausführungszeit muss unterhalb des Kehrwerts der Bildwiederholungsrate liegen, damit das System in Echtzeit arbeitet. Bei der gewählten Bildwiederholungsrate von 31.25 Hz ergibt sich eine maximale Einschaltdauer für LED2 von 32 ms. Der in Kapitel 2 vorgestellte Bildverarbeitungsalgorithmus muss demnach innerhalb von 32 ms abgearbeitet werden, damit das System in Echtzeit arbeitet.

5.3.4 Ethernetverbindung

Der Task für die Verbindungsbereitstellung (TSK_img_transmit) wird nur auf der Ethernet-Plattform ausgeführt. Zu Beginn erfolgt die Initialisierung des lwIP-Stacks durch Aufruf der Funktion *lwIP_NetStart()* mit der IP-Adresse als Übergabeparameter. Die Ethernet-Plattform wird in dieser Arbeit mit der IP-Adresse 192.168.11.33 verwendet. Anschließend wird über das TCP der Port mit der Nummer 33333 geöffnet und auf eine Verbindung gewartet. Der effektive Datendurchsatz bei TCP ist im Vergleich zu UDP zwar geringer, aber dafür besteht eine zuverlässige Verbindung, was in diesem Fall wichtiger ist.

Sobald eine Verbindung hergestellt wurde, wird die IP-Adresse und die Portnummer des Client als Statusinformation ausgegeben. Anschließend wird eine globale Variable gesetzt, anhand deren Zustand innerhalb der Funktion *img_process_ethernet()* entschieden wird, ob die Ergebnisse über die Ethernet Schnittstelle übertragen werden oder nicht. Wenn die Verbindung beendet wird, wird auf einen erneuten Verbindungsaufbau gewartet.

Der Task ist so aufgebaut, dass immer nur eine Verbindung zur Zeit möglich ist. Mehrere Verbindungen zur selben Zeit sind nicht möglich, da dafür mehr CPU-Zeit benötigt wird, die nicht vorhanden ist.

5.3.5 Programmausgabe

Nachdem die Softwarekonfiguration abgeschlossen ist, folgt in diesem Abschnitt die Darstellung der Programmausgabe der beiden Plattformen und die Informationen, die sich hinter den LEDs verbergen. Listing 5.1 zeigt die Ausgabe der Stereo-Plattform. Wie zu erkennen ist, wird als erstes der Bildverarbeitungstask (TSK_imgprocess) gestartet und anschließend der Synchronisationstask. Dann wartet dieser auf eine Antwort durch die Ethernet-Plattform. Nachdem die erwartete Antwort angekommen ist, folgen die einzelnen Schritte des Synchronisationstasks aus Kapitel 5.3.2.

```
TSK_imgprocess() waiting for sync.
TSK_control()
Waiting for answer...
Camera Init
Camera EDMA enable
Cameras enabled.
Estimating FPS (4 seconds)
Camera 1 is running at 31 Frames per Second
Framesize is 640x480 Pixel (307200 Byte)
Camera 2 is running at 31 Frames per Second
Framesize is 640x480 Pixel (307200 Byte)
Init done.
TSK_control() suspended.
TSK_imgprocess() running.
```

Listing 5.1: Programmausgabe Stereo-Plattform

Bei der Ethernet-Plattform sieht die Ausgabe ähnlich aus (siehe Listing 5.2). Da auf dieser Plattform zusätzlich noch der Task für die Verbindung über Ethernet ausgeführt wird, kommen ein paar Statusmeldungen hinzu. Die IP-Adresse und die Portnummer, über die eine Verbindung hergestellt werden kann, wird ausgegeben und bei einem Verbindungsaufbau werden die IP-Adresse und die Portnummer des Client angezeigt. Wird die Verbindung beendet, so wird die Ausgabe über die IP-Adresse und Portnummer des Client mit dem Zusatz „closed.“ erweitert.

```
TSK_imgprocess() waiting for sync.
TSK_control()
TSK_imgtransmit() started.
Waiting for answer...
IP:Port -> 192.168.11.33:33333
Waiting for connection.
Waiting for Camera Init
Camera EDMA enable
Cameras enabled.
Estimating FPS (4 seconds)
Camera 1 is running at 31 Frames per Second
Framesize is 640x480 Pixel (307200 Byte)
Init done.
TSK_control() suspended.

TSK_imgprocess() running.
Connection from 192.168.11.22:3154 closed.
Waiting for connection.
```

Listing 5.2: Programmausgabe Ethernet-Plattform

Die vier LEDs auf den DSK-Boards geben Aufschluss über verschiedene Ereignisse auf der jeweiligen Plattform. Die ersten beiden LEDs (LED0 + 1) werden während der Ausführung der EDMA Interruptroutinen der Kameras jeweils umgeschaltet. Anhand der Frequenz, mit der die LEDs blinken, kann die Bildwiederholungsrate überprüft werden. Bei der Ethernet-Plattform hat LED1 keine Funktion, da diese nur Zugriff auf die Daten einer Kamera besitzt.

LED2 zeigt an, dass die Bildverarbeitung durchgeführt wird. Anhand der Einschaltdauer dieser LED kann die benötigte Zeit für die Bildverarbeitung ermittelt werden. Als letztes wäre da noch LED3 zu nennen, die mit Beginn der Synchronisation gesetzt wird und bei jeder Nachrichtenübermittlung ihren Zustand ändert. Dadurch kann überprüft werden, ob der Synchronisationstask noch arbeitet.

Während der Bildverarbeitungstask ausgeführt wird läuft die Datenübertragung über McBSP paketorientiert. In dieser Zeit wird LED3 immer dann umgeschaltet, wenn der EDMA Interrupt für das McBSP ausgelöst wird. Darüber kann ermittelt werden ob und in welchen Abständen Datenpakete auf der jeweiligen Plattform empfangen werden.

6 Implementierung

Dieses Kapitel befasst sich mit der Implementierung der Bildverarbeitungsalgorithmen, die in Kapitel 2 vorgestellt wurden. Zunächst werden die einzelnen Schritte der Bildverarbeitung für die jeweilige Plattform anhand der beiden Funktionen *img_process_stereo()* und *img_process_ethernet()* erläutert. Anschließend folgt eine detaillierte Beschreibung der Umsetzung der Algorithmen.

6.1 img_process_stereo()

Der Ablauf der Funktion *img_process_stereo()* ist in Bild 6.1 dargestellt. Bei den einzelnen Bearbeitungsschritten werden jeweils die Bilder beider Kameras separat voneinander bearbeitet.

Im ersten Schritt wird die Farbklassifikation aus Kapitel 2.2 angewendet. Anschließend folgt das Opening durch Anwendung einer Erosion gefolgt von einer Dilatation (siehe Kapitel 2.3.1 und 2.3.2). Nach dem Opening wird ein Closing durchgeführt, um eventuell auftretende Lücken im Objekt zu schließen.

Nach der Aufbereitung der Bilddaten werden die Merkmale wie Massenschwerpunkt, Flächeninhalt usw. berechnet. Zuletzt wird aus den berechneten Merkmalen und den Systemparametern, wie Kameraabstand und Brennweite der Optik, die Position des ermittelten Objekts bestimmt.

Die Übergabe der Ergebnisse an den Bildverarbeitungstask aus Kapitel 5.3.3 erfolgt über globale Variablen.

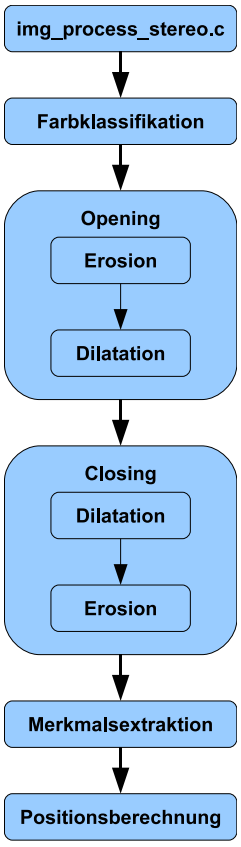


Bild 6.1: img_proc_stereo()

6.2 img_process_ethernet()

Die Bildverarbeitung auf der Ethernet-Plattform läuft zunächst in ähnlicher Form wie bei der Stereo-Plattform ab. In Bild 6.2 ist der Aufbau der Funktion dargestellt. Zunächst wird die Farbklassifikation, das Opening und das Closing für das Bild einer Kamera in gleicher Weise wie bei der Stereo-Plattform ausgeführt.

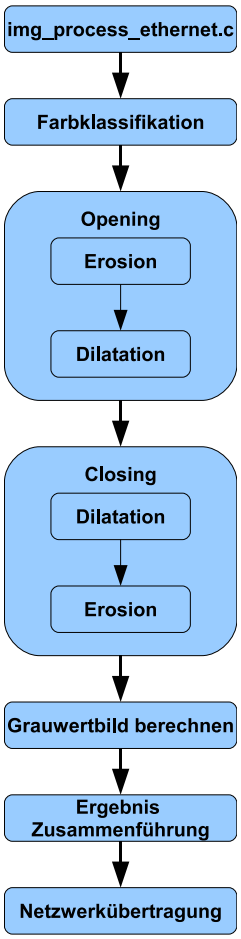


Bild 6.2: img_proc_ethernet()

Die nachfolgenden Bearbeitungsschritte unterscheiden sich allerdings von der *img_process_stereo()* Funktion. Für die Visualisierung wird aus dem Farbbild im Bayer Mosaik ein Grauwertbild berechnet. In dieses Grauwertbild werden anschließend die Ergebnisse der Bildverarbeitung eingezeichnet.

Dazu gehören unter anderem die erkannte Fläche und der Umfang, der aus dem Ergebnisbild des Closings ermittelt wird. Zusätzlich werden die beiden Massenschwerpunkte, die von der Stereo-Plattform empfangen wurden, in das Bild eingezeichnet.

Sofern eine Netzwerkverbindung zu einem PC besteht, folgt die Übertragung des bearbeiteten Grauwertbildes und der errechneten Merkmale über die Ethernet-Schnittstelle.

Die Durchführung der einzelnen Algorithmen auf der Ethernet-Plattform ist hauptsächlich zur besseren Visualisierung der Ergebnisse vorgesehen, die in Kapitel 6.10 beschrieben werden.

In den folgenden Unterkapiteln folgt die Implementierung der einzelnen Algorithmen, die für die Umsetzung der Bildverarbeitung aus Kapitel 2 eingesetzt wurden.

6.3 Farbklassifikation

In diesem Abschnitt wird die Umsetzung der Farbklassifikation nach Kapitel 2.2 erläutert. Bevor damit begonnen werden kann, muss zunächst aus dem vorliegenden Bayer Mosaik (siehe Kapitel 3.2.1.1) die Farbtonkomponente des HSV-Farbraums berechnet werden.

Die bei der Simulation in Kapitel 2.1.2 verwendete Umrechnung für die Farbtonkomponente entspricht bis auf eine Normierung der Gleichung 6.1. Für die Gleichung gilt die Bedingung, dass die einzelnen Farbkomponenten R, G und B im Intervall von 0 bis 1 sein müssen. Das Ergebnis H hat dann einen Wertebereich von 0° bis 360°. Das Maximum und Minimum berechnet sich jeweils aus den drei Komponenten Rot (R), Grün (G) und Blau (B). Sollte der Fall eintreten, dass H kleiner als Null ist, so wird 360° hinzuaddiert.

$$H := \begin{cases} 0°, & \text{falls } MAX = MIN \Leftrightarrow R = G = B \\ 60° \cdot (0 + \frac{G-B}{MAX-MIN}), & \text{falls } MAX = R \\ 60° \cdot (2 + \frac{B-R}{MAX-MIN}), & \text{falls } MAX = G \\ 60° \cdot (4 + \frac{R-G}{MAX-MIN}), & \text{falls } MAX = B \end{cases} \quad [2] \qquad (6.1)$$

Betrachtet man die einzelnen Farbwerte des gesuchten Objekts in Bild 2.1, so erkennt man, dass diese einem gewissen Schema folgen. Der rote Farbanteil ist bei allen Punkten innerhalb des Objekts maximal und der blaue Farbanteil ist größer als der grüne.

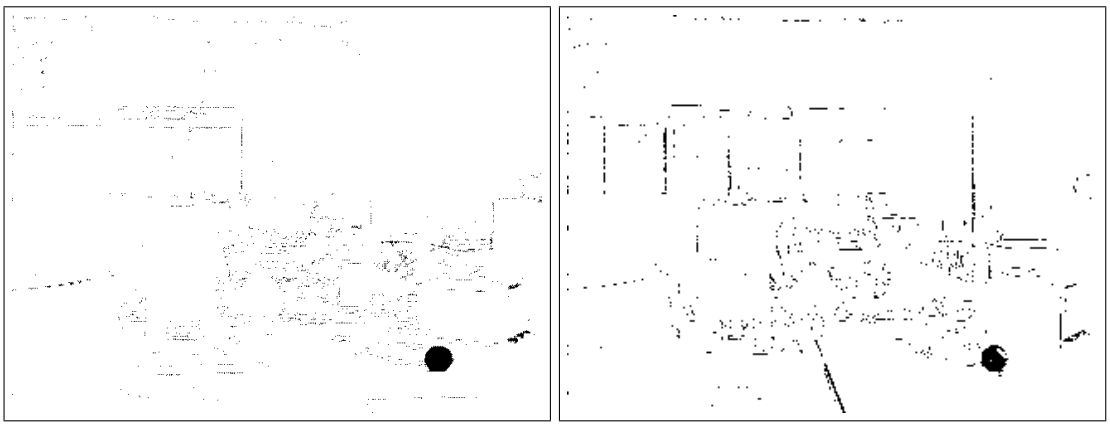
Da der rote Farbanteil bei allen Punkten im Objekt maximal ist und der grüne minimal, ergibt sich aus Gleichung 6.1 die vereinfachte Gleichung 6.2.

$$H := 60° \cdot \left(\frac{G-B}{R-G} \right), \text{falls } R > B > G \qquad (6.2)$$

Die Durchführung der Division aus Gleichung 6.2 ist auf dem verwendeten DSP recht aufwändig, da der Prozessor über keine Möglichkeit verfügt, diese in Hardware zu berechnen. Integer-Divisionen werden umständlich in Software berechnet. Lässt man diese Division und die Normierung auf einen Wertebereich von 0° bis 360° weg, ergibt sich Gleichung 6.3.

$$H = G - B, \text{falls } R > B > G \qquad (6.3)$$

Durch diese Optimierung vereinfacht sich die Berechnung auf zwei Vergleichsoperationen und eine Subtraktion. Die Unterschiede sind in Bild 6.3 mit der anschließenden Klassifikation dargestellt. Aufgrund der Optimierung sieht das Ergebnis schlechter aus, aber durch das anschließende Opening wird dieser Nachteil kompensiert.



(a) nach Gleichung 6.1 (b) nach Gleichung 6.3

Bild 6.3: Vergleich der Berechnungsmethoden (MATLAB-Skript kapitel63.m)

Wichtig bei dieser Berechnung ist, dass der Weißabgleich der Kamera abgeschaltet ist. Anderenfalls kommt es vor, dass das Objekt bei sich ändernder Beleuchtung nicht mehr in seiner Farbe, sondern in weiß dargestellt wird. Dies hängt außerdem sehr stark von dem Hintergrund ab, da der Weißabgleich über die gemittelte Bildhelligkeit von der Kamera ermittelt wird.

Eigentlich müssten für die Berechnung der Farbtonkomponente für jeden Bildpunkt alle drei Farbanteile des RGB-Farbraums vorliegen. Da die Kamera das Bild im Bayer Mosaik überträgt müsste eine Interpolation durchgeführt werden. Betrachtet man das 640x480 Bildpunkte große Bayer Mosaik als ein 320x240 großes BGGR-Farbbild, bei dem jeder Bildpunkt aus einem 2x2 Pixel großem Feld des Bayer Mosiak besteht, so kann man sich die aufwändige Interpolation ersparen. Dabei reduziert sich allerdings die Auflösung des Ergebnisses auf 320x240 Bildpunkte.

Bei dieser Vorgehensweise sind immer zwei grüne Bildpunkte vorhanden, von denen aber nur einer verwendet wird. Der doppelte grüne Anteil aus der grün-roten Zeile wird verworfen. Listing 6.1 zeigt die Umsetzung der Berechnung als C-Code. Die gewünschte Funktionalität ist gegeben, aber die gemessene Ausführungszeit liegt bei ca. 42.80 ms. Dadurch ist das Echtzeitverhalten nicht mehr gegeben, da die Kamera alle 32 ms ein neues Bild liefert.

Anhand des C-Codes erstellt der Compiler Assemblercode, wobei die Möglichkeit der parallelen Ausführung nicht besonders ausgenutzt wird, obwohl bereits Optimierungsdirektiven durch Anweisungen mittels *#pragma* für die Schleifen angegeben sind.

```
void extract_color ( Uint8 * ptr_src, Uint8 * ptr_dst,
                     Uint8 lower, Uint8 upper, Uint8 bin_value )
{
   register unsigned char r, g, b, value;
   register int x, y, idx;

   #pragma MUST_ITERATE(240)
   for ( y = 0; y < 480; y+=2)
   {
      #pragma MUST_ITERATE(320)
      for ( x = 0; x < 640; x+=2 )
      {
         r = ptr_src[(x+1)+(y+1)*640];
         g = ptr_src[(x+1)+y*640];
         b = ptr_src[x+y*640];

         idx = (x>>1)+(y>>1)*320;

         if ( r > b )
         {
            if ( b > g )
            {
               value = g-b;
               if ( (value > lower) && (value < upper) )
                  ptr_dst[idx] = bin_value;
               else
                 ptr_dst[idx] = 0;
            }
         }
      }
   }
}/* extract color */
```

Listing 6.1: Farbklassifikation - extract_color()

Die Funktionalität ist gegeben, aber die Durchführung erfolgt noch nicht in der benötigten Geschwindigkeit. Aus diesem Grund besteht nur noch die Möglichkeit, diese Funktion von Hand in Assembler zu programmieren. Der Prozessor verfügt glücklicherweise über einige Instruktionen, die speziell für die Verarbeitung von gepackten Daten ausgelegt sind. Dadurch ist es möglich, vier Operationen mit jeweils 8 Bit breiten Eingangswerten mit einer Instruktion zu bearbeiten.

Bevor allerdings mit der Assemblerprogrammierung begonnen werden kann, muss der Aufbau des Programmcodes und die benötigten Instruktionen bekannt sein. Dazu gibt es von Texas Instruments eine Dokumentation, die sämtliche Instruktionen enthält, die für den in dieser Arbeit verwendeten Prozessor verfügbar sind (siehe [18]).

Die Vorgehensweise für die Entwicklung eines Assemblerprogramms, dass die Hardware optimal ausnutzt, teilt sich in drei Abschnitte (siehe [19]). Als erstes müssen die Instruktionen ausgewählt werden, um die gewünschte Funktionalität zu gewährleisten. Diese werden mit ihren Abhängigkeiten in einem Graphen dargestellt. Anschließend werden den jeweiligen Instruktionen eine der acht verfügbaren Hardwareeinheiten zugewiesen, in der die

Instruktion ausgeführt werden kann. Im zweiten Abschnitt erfolgt der zeitliche Ablauf der Befehle für eine Berechnung. In diesem Fall wäre das genau ein Schleifendurchlauf des C-Programms. Im dritten und letzten Abschnitt wird dann die Parallelisierung der Instruktionen durchgeführt, damit alle Schleifendurchläufe ausgeführt werden.

Bild 6.4 zeigt den Abhängigskeitsgraphen der Instruktionen, der das selbe Ergebnis wie der C-Code aus Listing 6.1 ergibt. Neben den einzelnen Instruktionen ist durch eine Zahl angegeben, wie viele Takte benötigt werden, bis das Ergebnis vorliegt. Wenn keine Zahl angegeben ist, ist das Ergebnis nach einem Takt verfügbar. Darüber hinaus ist neben den Instruktionen angegeben, mit welchen Hardwareeinheiten diese ausgeführt werden.

Im ersten Takt werden durch die beiden LDDW (Load Doubleword, 64 Bit) Instruktionen jeweils acht Bildpunkte geladen. Die LDDW-Instruktion auf der linken Seite greift dabei auf die erste Zeile des Bayer Mosaik und die andere auf die zweite Zeile zu. Diese Instruktion benötigt fünf Takte bis die Daten im Register vorhanden sind. In den beiden Registerpaaren sind jetzt die ersten Bildpunkte in der Form GBGBGBGB und RGRGRGRG enthalten. Hierbei ist darauf zu achten, dass die Reihenfolge der Bildpunkte umgekehrt ist. Dies muss im weiteren Verlauf berücksichtigt werden. Beim Speicher wird die Reihenfolge erneut umgekehrt.

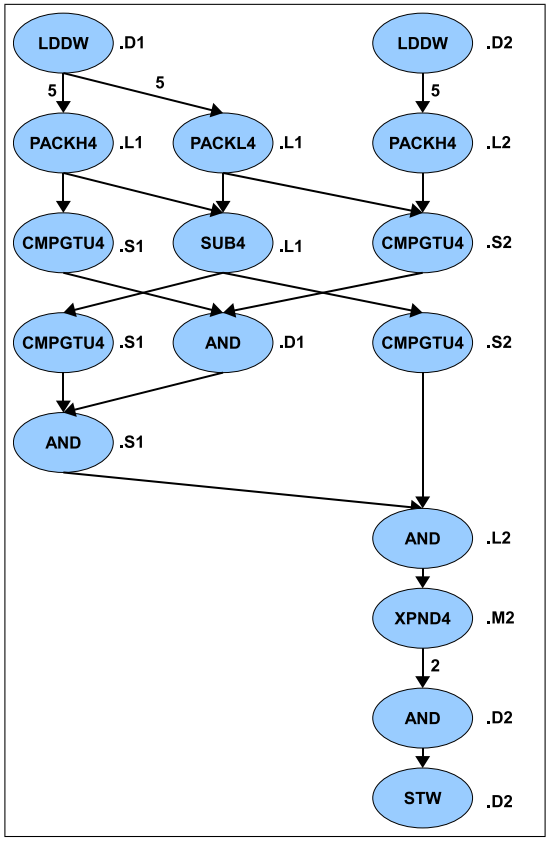


Bild 6.4: Abhängigkeitsgraph extract_color_asm()

Mit der Instruktion PACKH4 (Pack Four High Bytes Into for 8-Bit Halfwords) werden aus dem Registerpaar mit dem Inhalt GBGBGBGB nur die grünen Bildpunkte extrahiert. Die Instruktion PACKL4 entnimmt dementsprechend die blauen Bildpunkte. Auf der rechten Seite werden mit der PACKH4 Instruktion die roten Bildpunkte extrahiert. Nach der Verarbeitung dieser Instruktion sind die Farbwerte der Farbe nach in die Register sortiert (RRRR, GGGG und BBBB).

Im folgenden Takt wird die Subtraktion der grünen und blauen Bildpunkte byteweise mit der Instruktion SUB4 durchgeführt. Parallel dazu werden die beiden Vergleiche $R > B$ und $B > G$ durch die Instruktion CMPGTU4 (Compare for Greater Than, Unsigned, Packed

8-Bit) ausgeführt. Diese Instruktion arbeitet ebenfalls byteweise und das Ergebnis wird in die untersten 4 Bit des Ergebnisregisters gespeichert.

Anschließend werden die Ergebnisse der beiden Vergleiche mit einer Und-Verknüpfung kombiniert und das Ergebnis der Subtraktion wird mit dem oberen und unteren Schwellwert verglichen. Die Ergebnisse dieser Vergleichsoperationen werden dann mit den vorangegangenen verknüpft. Da das Ergebnis der Vergleichsoperationen jeweils mit 4 Bit dargestellt wird, wird dieses mit der Instruktion XPND4 (Expand Bits to Packed 8-Bit Masks) auf 32 Bit erweitert. Diese Maske wird mit dem gewünschten Ergebniswert (Variable bin_value im C-Code) verknüpft und abgespeichert.

Alle Bildpunkte, für die die Vergleichsoperationen positiv verliefen, bekommen einen Wert zugewiesen, die übrigen werden auf Null gesetzt. Mit einem Durchlauf dieser Instruktionen werden vier Ergebnisbildpunkte auf einmal berechnet.

Nachdem die Abhängigkeiten der Instruktionen geklärt sind, geht es im zweiten Schritt darum, die zeitliche Abfolge festzulegen. In Tabelle 6.1 ist der Ablauf eines Schleifendurchlaufs dargestellt. Dabei werden auch die benötigten Takte eingehalten, bis das Ergebnis der Instruktionen in den jeweiligen Registern angekommen ist. Für die Berechnung von vier Ergebnisbildpunkten werden 15 Takte benötigt.

Tabelle 6.1: Zeitablauf der Instruktionen - extract_color_asm.asm

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
D1	LDDW								AND						
L1						PACKL4	PACKH4	SUB4							
S1								CMPGTU4	CMPGTU4	AND					
M1															
D2	LDDW													AND	STW
L2						PACKH4					AND				
S2							CMPGTU4		CMPGTU4						
M2												XPND4			

Abgesehen von der LDDW Instruktion, die fünf Takte benötigt und der XPND4 Instruktion, die zwei Takte benötigt, sind alle anderen Instruktionen in einem Takt abgearbeitet.

Ausgehend von einem Schleifendurchlauf geht es im letzten Schritt darum, die Instruktionen zeitlich so anzuordnen, dass die nachfolgenden Schleifendurchläufe möglichst früh anfangen. Dazu werden die selben Instruktionen so früh wiederholt, dass keine Überlagerung mit dem vorherigen Schleifendurchlauf entsteht.

Tabelle 6.2 zeigt den daraus resultierenden Aufbau der Instruktionsabfolge für sechs Iterationen. Die einzelnen Iterationen sind dabei farblich markiert. Die zweite Iteration kann frühestens drei Takte nach dem Beginn der ersten Iteration gestartet werden, ohne dass es zu Überschneidungen bei den Hardwareeinheiten kommt. Setzt man dieses Schema für mehrere Iterationen fort, so erkennt man, dass sich die Instruktionen ab den Takten 16 bis 18 wiederholen. Dieser Programmteil wird in der Literatur auch als Kernel-Loop bezeichnet, da dieser den Kern des Programms darstellt.

Tabelle 6.2: Softwarepipelining für die Farbklassifikation - extract_color_asm.asm

Takt	Paket	Anfang der Iteration	Ende der Iteration	Label	D1	L1	S1	D2	L2	S2	M2	parallele Instruktionen
1	1			preparation					MV	MVLK		2
2	2									ADD		1
3	3	1		prolog	LDDW		MVKL	LDDW				3
4	4											
5	5											
6	6	2			LDDW			LDDW				2
7	7			outer_loop						MVKL		1
8	8					PACKL4			PACKH4			2
9	9	3			LDDW	PACKH4				CMPGTU4		3
10	10					SUB4	CMPGTU4					2
11	11				AND	PACKL4	CMPGTU4		PACKH4	CMPGTU4		5
12	12	4			LDDW	PACKH4	AND	LDDW		CMPGTU4		5
13	13					SUB4	CMPGTU4		AND	BDEC		4
14	14				AND	PACKL4	CMPGTU4		PACKH4	CMPGTU4	XPND4	6
15	15	5			LDDW	PACKH4	AND	LDDW		CMPGTU4		5
16	**16**			inner_loop		**SUB4**	**CMPGTU4**	**AND**	**AND**	**BDEC**		**5**
17	**17**		1		**AND**	**PACKL4**	**CMPGTU4**	**STW**	**PACKH4**	**CMPGTU4**	**XPND4**	**7**
18	**18**	6			**LDDW**	**PACKH4**	**AND**	**LDDW**		**CMPGTU4**		**5**
19	**16**					SUB4	CMPGTU4	AND	AND	BDEC		5
20	**17**		2		AND	PACKL4	CMPGTU4	STW	PACKH4	CMPGTU4	XPND4	7
21	**18**				LDDW	PACKH4	AND	LDDW		CMPGTU4		5
22	**16**					SUB4	CMPGTU4	AND	AND	BDEC		7
23	**17**		3		AND	PACKL4	CMPGTU4	STW	PACKH4	CMPGTU4	XPND4	5
24	**18**				LDDW	PACKH4	AND	LDDW		CMPGTU4		7
25	**16**					SUB4	CMPGTU4	AND	AND	BDEC		5
26	**17**		4		AND		CMPGTU4	STW	PACKH4	CMPGTU4	XPND4	7
27	**18**				LDDW		AND	LDDW				5
28	19			epilog		MV	BDEC	AND	AND	B		5
29	20		5					STW		ADDK	XPND4	3
30	21				*LDDW*				*LDDW*			2
31	22							AND				1
32	23		6					STW				1
33	24				*LDDW*				*LDDW*			2

Damit dieser Kern mehrfach nacheinander ausgeführt werden kann, muss an entsprechender Stelle im Quelltext ein Sprung an den Anfang des Kernel stattfinden. Die Sprünge werden durch die Instruktion BDEC (Branch and Decrement) in der S2-Einheit ausgeführt. Diese Instruktion erledigt zwei Aufgaben gleichzeitig. Im darauf folgenden Takt wird ein Zähler dekrementiert, dieser entspricht in diesem Fall der Variablen x aus dem oben gezeigten C-Code, wobei der Anfangswert anders ist. Erst fünf Takte nach der Ausführung der BDEC Instruktion wird der Sprung durchgeführt. Die erste Sprungoperation findet in Takt 13 statt. Fünf Takte später, also nach Takt 18, wird der Sprung ausgeführt und wieder die Instruktionen in Takt 16 ausgeführt.

Dieser Ablauf läuft so lange weiter, bis die Anzahl der vorgegebenen Sprünge erreicht ist. Danach folgen noch weitere Instruktionen, die die bisherigen angefangenen Iterationen abschließen. Die hier in roter Schrift dargestellten Instruktionen werden nicht mehr ausgeführt, da nach Takt 19 der Zählerstand für die Anzahl der Iterationen gleich Null ist. Diese bedingte Ausführung wird nur bei den BDEC und LDDW Instruktionen angewendet.

Die farblich nicht markierten Instruktionen in den ersten drei Takten sind dafür zuständig, dass die Adresse, auf die die LDDW Instruktion in der D2-Einheit zeigt, dem Anfang der zweiten Zeile des Bayer Mosaik Bildes entspricht. Darüber hinaus wird ein Register mit dem Wert 239 initialisiert, was der Anzahl der äußeren Iterationen enspricht. Zur Berechnung von sechs Iterationen sind drei Sprungoperationen nötig. Bei einer Bildzeile bestehend aus 640 Byte (80 LDDW Instruktionen) werden 77 Sprungoperationen benötigt. In Takt 7 wird dazu der Spaltenzähler mit 77 initialisiert.

Nachdem ein Bildzeilenpaar des Bayer Mosaiks bearbeitet wurde, werden die Adressen der beiden Zeiger mit der MV (Move) und ADDK (Add Signed 16-Bit Constant to Register) Instruktion für die Bearbeitung des folgenden Zeilenpaares vorbereitet (Takt 28 in Tabelle 6.2). Ist die Bearbeitung der ersten beiden Zeilen abgeschlossen, so wird durch die BDEC Instruktion in Takt 28 wieder zum Anfang der äußeren Schleife in Takt 7 gesprungen. Da dieser Sprung jedoch erst fünf Takte später ausgeführt wird, liegt die Sprungmarke (outer_loop) nicht direkt am Anfang der Funktion. Außerdem werden in den verbleibenden Takten bereits die ersten Ladeoperationen für die nächste Iteration durchgeführt.

Sobald das Bild vollständig bearbeitet wurde, ist der Zählerstand für die äußeren Iterationen Null und die BDEC Instruktion aus Takt 28 wird nicht mehr ausgeführt. Statt dessen wird die B (Branch) Instruktion in der S2-Einheit im selben Takt ausgeführt und ins Hauptprogramm, welches diese Funktion aufgerufen hat, zurückgesprungen.

Aufgrund der Parallelisierung ist es nötigt, die Interrupts während des Ablaufs dieser Funktion zu deaktivieren. Die Abfolge der Sprunginstruktionen ist kleiner als sechs Takte. Daher befindet sich immer mindestens eine Sprunginstruktion in der Pipeline des Prozessors, was die korrekte Ausführung einer Interruptroutine unmöglich macht. Darüber hinaus sollten die Register, die bei den einzelnen Instruktionen verwendet werden, nicht doppelt verwendet werden, da die Parallelisierung sonst nicht ordnungsgemäß funktioniert. Dies stellt jedoch kein Problem dar, da der Prozessor mit 64 Registern ausreichend zur Verfügung hat.

Durch die Parallelisierung beträgt die Ausführungszeit dieser Funktion nur noch 7.20 ms. Im Vergleich zu der in C geschriebenen Funktion, die 42.80 ms benötigt, entspricht dies einer Verbesserung um den Faktor 6 und das Echtzeitverhalten kann sichergestellt werden.

6.4 Morphologische Operationen

Nach der Farbklassifikation haben alle gesetzten Bildpunkte den Wert eins und die übrigen den Wert null. Ausgehend von diesem Bild werden nun die morphologischen Operationen durchgeführt. Zur Berechnung der Erosion bzw. Dilatation wird das Bild zunächst mit einem 3x3 Strukturelement, bei dem alle Elemente gesetzt sind, bearbeitet. Dabei wird das Bild in zwei Schritten bearbeitet. Bild 6.5 zeigt den Ablauf anhand eines Beispiels.

Im ersten Durchgang wird das Bild zeilenweise bearbeitet und die Summe über drei Bildpunkte gebildet. Das Ergebnis ist in der Mitte des Bildes dargestellt. Anschließend erfolgt im zweiten Schritt die Bearbeitung spaltenweise.

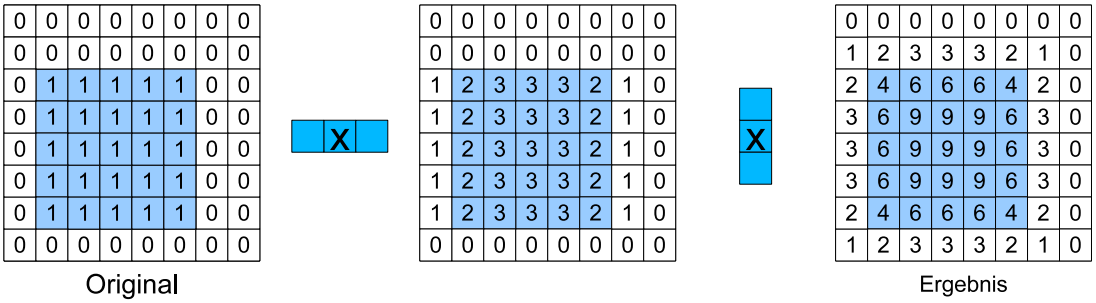


Bild 6.5: Berechnungsprinzip der Funktion für Erosion und Dilatation

An dieser Stelle wurde noch eine weitere Funktion entwickelt, die zusätzlich noch eine Maskierung anhand eines Schwellwertes durchführt. Dazu wird der Wert des Bildpunktes mit dem Schwellwert verglichen und sofern dieser größer ist, wird dem jeweiligen Bildpunkt der Wert einer Maske zugewiesen.

Bei einem Schwellwert von acht und einer Maske von eins, werden alle Bildpunkte, die den Wert neun haben übernommen und alle anderen auf null gesetzt. Dadurch ist die Funktionalität der Erosion gegeben (siehe Bild 6.6). Für die Dilatation wird der Schwellwert entsprechend auf Null gesetzt.

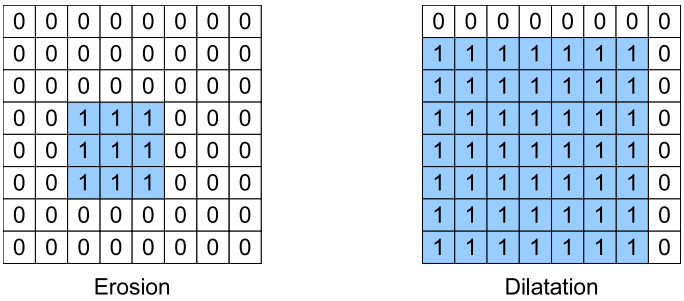


Bild 6.6: Ergebnisse der Erosion und Dilatation

In Bild 6.7 ist der Abhängigkeitsgraph für den ersten Bearbeitungschritt dargestellt. Dieser Algorithmus ist für die zeilenweise Berechnung der Summe über drei Pixel zuständig.

Als erstes werden dazu acht Bildpunkte (64 Bit) auf einmal geladen. Anschließend werden diese zweimal nacheinander um ein Byte nach rechts geschoben. Zwischen diesen Schritten werden die Inhalte der unteren Register (32 Bit) aufaddiert. Dadurch entstehen nach einer Iteration vier Summen über drei Pixel. Die vier aktuellsten Bildpunkte werden im Anschluss daran für die folgende Iteration verwendet.

Dies hat allerdings zur Folge, dass das Ergebnisbild im Vergleich zur üblichen Vorgehensweise um eine Zeile nach oben und eine Spalte nach links verschoben ist. Aufgrund der Speicheranordnung und der wortweisen Adressierung ist diese Methode aber deutlich einfacher umzusetzen.

Da das Ergebnisbild die selbe Größe wie das Ursprungsbild besitzt, müssen die letzten beiden Bildpunkte einer Zeile maskiert werden, da für diese keine Summe über drei Pixel gebildet werden kann.

Betrachtet man die Belegung der einzelnen Hardwareeinheiten, so erkennt man, dass bei diesem Algorithmus lediglich eine Hälfte des Prozessors benötigt wird. Daher können die übrigen Hardwareeinheiten parallel dazu die folgende Zeile bearbeiten.

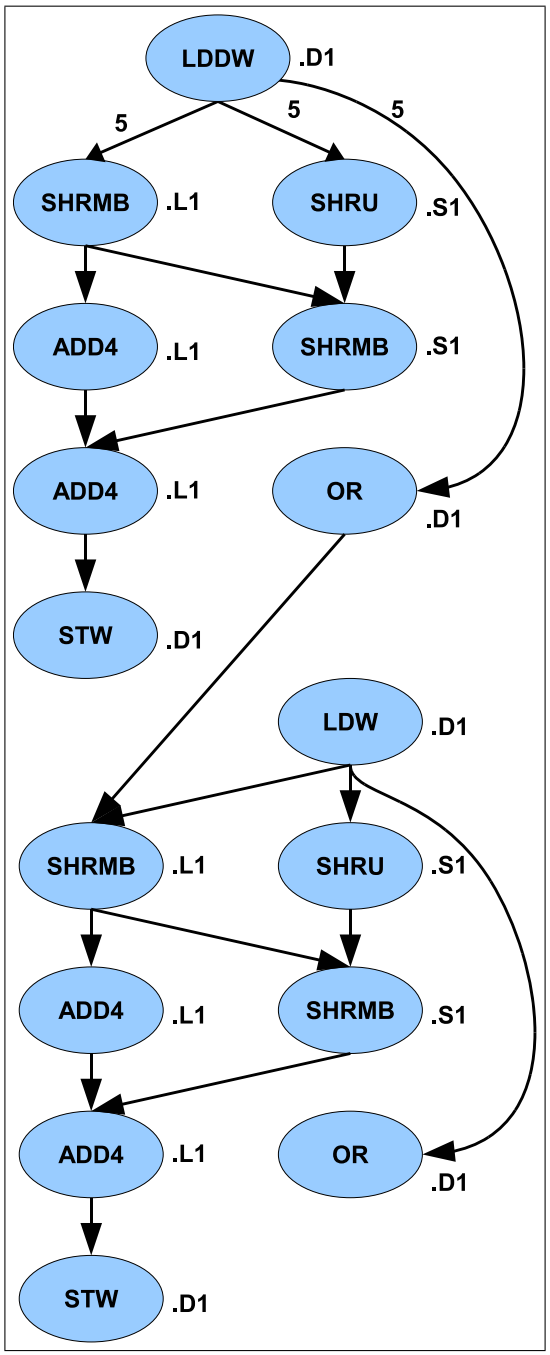


Bild 6.7: Abhängigkeitsgraph - Schritt 1

Der Algorithmus wird dementsprechend so aufgebaut, dass die eine Prozessorhälfte die ungeraden und die andere die geraden Zeilen bearbeitet.

In Tabelle 6.3 ist der Ablauf der einzelnen Instruktionen für sechs Iterationen gezeigt. Die einzelnen Iterationen sind wieder farblich gekennzeichnet. Auf beiden Seiten des Prozessors wird der selbe Programmcode ausgeführt, lediglich die Adressen, von denen die Bildpunkte geladen werden, unterscheiden sich.

Tabelle 6.3: Softwarepipelining - Schritt 1

Takt	Paket	Anfang der Iteration	Ende der Iteration	Label	D1	L1	S1	D2	L2	S2	parallele Instruktionen
1	1			preparation		MV	MVKL			ADD	3
2	2	1		prolog_row	LDDW	SHL		LDDW			3
3	3				MV	ADD	MVKH			SHRU	4
4	4					MV	SHRU		ADD4	SHRU	4
5	5	2			LDW		SUB	LDW	MV	SUB	5
6	6			outer_loop_row						BDEC	1
7	7					SHRMB	SHRU		SHRMB	SHRU	4
8	8	3			LDW	ADD4	SHRMB	LDW	ADD4	SHRMB	6
9	**9**			inner_loop_row	**OR**	**ADD4**		**OR**	**ADD4**	**BDEC**	**5**
10	**10**		1		**STW**	**SHRMB**	**SHRU**	**STW**	**SHRMB**	**SHRU**	**6**
11	**11**	4			**LDW**	**ADD4**	**SHRMB**	**LDW**	**ADD4**	**SHRMB**	**6**
12	**9**				OR	ADD4		OR	ADD4	BDEC	5
13	**10**		2		STW	SHRMB	SHRU	STW	SHRMB	SHRU	6
14	**11**	5			LDW	ADD4	SHRMB	LDW	ADD4	SHRMB	6
15	**9**				OR	ADD4		OR	ADD4	BDEC	5
16	**10**		3		STW	SHRMB	SHRU	STW	SHRMB	SHRU	6
17	**11**	6			LDW	ADD4	SHRMB	LDW	ADD4	SHRMB	6
18	**9**				OR	ADD4		OR	ADD4		4
19	**10**		4		STW	SHRMB	SHRU	STW	SHRMB	SHRU	6
20	**11**				LDW	ADD4	SHRMB	LDW	ADD4	SHRMB	6
21	**9**				OR	ADD4		OR	ADD4		4
22	**10**		5		STW	SHRMB	SHRU	STW	SHRMB	SHRU	6
23	**11**				LDW	ADD4	SHRMB	LDW	ADD4	SHRMB	6
24	12			epilog_row		ADD4	MV		ADD4	ADD	4
25	13		6			AND	BDEC	ADD	AND		3
26	14				STW		MV	STW		SHRU	3
27	15				*LDDW*	MV		*LDDW*	ADD	SUB	5
28	16				LDDW		SHRU	LDDW		MV	
29	17				LDDW		SUB	LDDW		SUB	
30	18				*LDW*			*LDW*			2

Nachdem die Pipeline des Prozessors gefüllt ist, werden fortlaufend die Instruktionen der Takte 9 bis 11 ausgeführt, bis die aktuelle Bildzeile bearbeitet ist. Anschließend werden die zuletzt angefangenen Iterationen abgeschlossen und die Adressregister werden für die nächsten Zeilen vorbereitet.

Sobald der Iterationszähler bei Null angekommen ist, wird die letzte Iteration ohne Laden neuer Bildpunkte durchgeführt. Die letzten Bildpunkte wurden mit der vorangegangenen Iteration bereits geladen. In dem Register stehen dann zwar noch irgendwelche Bildpunkte, die aber aus bereits genannten Gründen maskiert werden.

Nach der Bearbeitung eines Zeilenpaares wird am Ende direkt mit dem Laden (grau) der Bildpunkte aus den folgenden Zeilen begonnen. Die darüber hinaus in blauer Schrift dargestellten Instruktionen werden erst ausgeführt wenn das gesamte Bild bearbeitet wurde, da diese für den zweiten Durchgang (spaltenweise) verwendet werden.

Beim zweiten Durchgang wird das Bild spaltenweise bearbeitet. Die Adressierung erfolgt daher nicht linear, sondern so, dass beim nächsten Speicherzugriff die Bildpunkte aus der folgenden Zeile geladen werden. Es wird wieder die Summe über drei Bildpunkte gebildet und abgespeichert. Die einzelnen Bildpunkte werden aber in Registern vorgehalten, dass für die nächste Iteration einfach nur die ersten Bildpunkte von der vorherigen Summe abgezogen und die neuen hinzu addiert werden (siehe Listing 6.2).

```
diffH = H1 + H2
diffL = L1 + L2
sumH  = diffH + H3      //vier Summen aus drei vertikalen Bildpunkten
sumL  = diffL + L3      //vier Summen aus drei vertikalen Bildpunkten

diffH = sumH - H1
diffL = sumL - L1       //Subtraktion der ersten Bildpunkte

cmpH  = sumH > limit
cmpL  = sumL > limit    //Vergleich mit Schwellwert
xpndH = xpnd(cmpH)
xpndL = xpnd(cmpL)      //Erweiterung von Bit- auf Byte-Maske

resH  = xpndH&sumH
resL  = xpndL&sumL      //Maskierung der Summe

//Beginn der zweiten Iteration (H1 und L1 enthalten die nächsten Bildpunkte)
sumH  = diffH + H1
sumL  = diffL + L1

diffH = sumH - H2
diffL = sumL - L2       //Subtraktion der ersten Bildpunkte
...

//Beginn der dritten Iteration (H2 und L2 enthalten die nächsten Bildpunkte)
sumH  = diffH + H2
sumL  = diffL + L2

diffH = sumH - H3
diffL = sumL - L3       //Subtraktion der ersten Bildpunkte
...
```

Listing 6.2: Pseudocode für den zweiten Durchgang

Die Summe der Bildpunkte wird im Anschluss mit einem Schwellwert verglichen und bei positivem Vergleich wird dem Ergebnis der Wert der Summe zugewiesen. Die bereits erwähnte zusätzliche Maskierung erfolgt, indem das Ergebnis (Register resH, resL) nicht durch eine Und-Verknüpfung mit der Summe (sumH, sumL), sondern mit einer vorher vom Benutzer definierte Maske verknüpft wird. Die Variante ohne diese Maskierung wird nur bei der Erosion beim Closing verwendet, um die anschließende Merkmalsextraktion zu vereinfachen.

Anhand des Pseudocodes aus Listing 6.2 kann nun der in Bild 6.8 dargestellte Abhängigkeitsgraph für diesen Algorithmus erstellt werden. Es werden die einzelnen Instruktionen der ersten und dem Anfang der zweiten Iteratation (unterhalb der Trennlinie) gezeigt. Aus Gründen der Übersicht wurden die verwendeten Hardwareeinheiten nicht dargestellt.

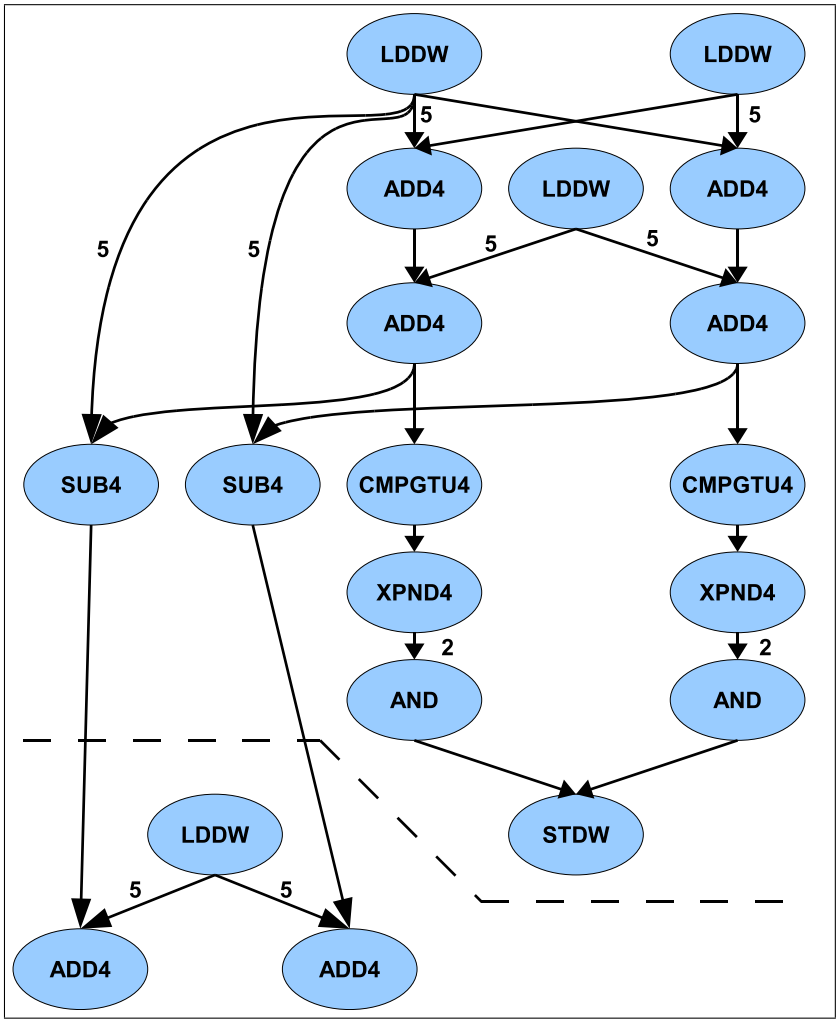


Bild 6.8: Abhängigkeitsgraph - Schritt 2

Aus diesem Abhängigkeitsgraphen kann dann das Softwarepipelining entwickelt werden. Das Ergebnis ist in Tabelle 6.4 dargestellt. Der Algorithmus lässt es zu, dass wieder alle Instruktionen in einer Hälfte des Prozessors ausgeführt werden können. Die übrigen Hardwareeinheiten in der zweiten Hälfte des Prozessors werden dazu verwendet, um die nächste Spalte zu bearbeiten. Es werden also immer zwei Bildspalten parallel bearbeitet.

Da in der zweiten Prozessorhälfte die selben Instruktionen wie in der ersten ausgeführt werden, ist hier zum besseren Verständnis der Code aus Listing 6.2 eingetragen.

Tabelle 6.4: Softwarepipelining - Schritt 2

Takt	Paket	Anfang der Iteration	Ende der Iteration	Label	D1	L1	S1	M1	D2	L2	S2	M2	parallele Instruktionen
25	13						MV		ADD				1
26	14												2
27	15												
28	16	1			LDDW		SHRU		H1:L1		MV		4
29	17				LDDW		SUB		H2:L2		SUB		4
30	18												
31	19			prolog_col	LDDW				H3:L3				2
32	20			outer_loop_col									
33	21												
34	22					ADD4				diffH=H1+H2			2
35	23	2			LDDW	ADD4			H1:L1	diffL=L1+L2			4
36	24					ADD4				sumH=diffH+H3			2
37	25					ADD4	CMPGTU4			sumL=diffL+L3	cmpH=sumH>limit		4
38	26					SUB4	CMPGTU4	XPND4		diffH=sumH-H1	cmpL=sumL>limit	xpndH=xpnd(cmpH)	6
39	27	3			LDDW	SUB4		XPND4	H2:L2	diffL=sumL-L1		xpndL=xpnd(cmpL)	6
40	28				AND	ADD4			resH=xpndH&sumH	sumH=diffH+H1			4
41	29			inner_loop_col	AND	ADD4	CMPGTU4		resL=xpndL&sumL	sumL=diffL+L1	cmpH=sumH>limit		6
42	30		1		STDW	SUB4	CMPGTU4	XPND4	resH:resL	diffH=sumH-H2	cmpL=sumL>limit	xpndH=xpnd(cmpH)	8
43	31	4			LDDW	SUB4		XPND4	H3:L3	diffL=sumL-L2		xpndL=xpnd(cmpL)	6
44	32				AND	ADD4			resH=xpndH&sumH	sumH=diffH+H2			4
45	33				AND	ADD4	CMPGTU4		resL=xpndL&sumL	sumL=diffL+L2	cmpH=sumH>limit		6
46	34		2		STDW	SUB4	CMPGTU4	XPND4	resH:resL	diffH=sumH-H3	cmpL=sumL>limit	xpndH=xpnd(cmpH)	8
47	35	5			LDDW	SUB4	BDEC	XPND4	H1:L1	diffL=sumL-L3		xpndL=xpnd(cmpL)	7
48	36				AND	ADD4			resH=xpndH&sumH	sumH=diffH+H3			4
49	37				AND	ADD4	CMPGTU4		resL=xpndL&sumL	sumL=diffL+L3	cmpH=sumH>limit		6
50	38		3		STDW	SUB4	CMPGTU4	XPND4	resH:resL	diffH=sumH-H1	cmpL=sumL>limit	xpndH=xpnd(cmpH)	8
51	39	6			LDDW	SUB4		XPND4	H2:L2	diffL=sumL-L1		xpndL=xpnd(cmpL)	6
52	40				AND	ADD4			resH=xpndH&sumH	sumH=diffH+H1			4
53	29				AND	ADD4	CMPGTU4		AND	ADD4	CMPGTU4		6
54	30		4		STDW	SUB4	CMPGTU4	XPND4	STDW	SUB4	CMPGTU4	XPND4	8
55	31	7			LDDW	SUB4		XPND4	LDDW	SUB4		XPND4	6
56	32				AND	ADD4			AND	ADD4	CMPGTU4		4
57	33				AND	ADD4	CMPGTU4		AND	ADD4	CMPGTU4		6
58	34		5		STDW	SUB4	CMPGTU4	XPND4	STDW	SUB4	CMPGTU4	XPND4	8
59	35	8			LDDW	SUB4		XPND4	LDDW	SUB4		XPND4	7
60	36				AND	ADD4			AND	ADD4	CMPGTU4		4
61	37				AND	ADD4	CMPGTU4		AND	ADD4	CMPGTU4		6
62	38		6		STDW	SUB4	CMPGTU4	XPND4	STDW	SUB4	CMPGTU4	XPND4	8
63	39				LDDW	SUB4		XPND4	LDDW	SUB4		XPND4	6
64	40				AND	ADD4			AND	ADD4			4
65	41			epilog_col	AND	ADD4	CMPGTU4		AND	ADD4	CMPGTU4		6
66	42		7		STDW		CMPGTU4	XPND4	STDW		CMPGTU4	XPND4	6
67	43							XPND4				XPND4	2
68	44				ADD	AND	ADD		ADD	AND			4
69	45				AND				AND				3
70	46		8		STDW	ZERO	BDEC		STDW	ZERO	MV		5
71	47				STDW				STDW		B		4
72	48				STDW				STDW		SUB		3
73	49				*LDDW*				*LDDW*				2
74	50				*LDDW*				*LDDW*				2
75	51								ADD				1
76	52				*LDDW*	MV			*LDDW*	ADD			4

Das Vorgehen bei dieser Routine erfolgt analog zu den vorangegangenen Assemblerfunktionen. Die Iterationen werden so oft hintereinander wiederholt, bis man eine Wiederholung der Instruktionen erkennt. In diesem Fall beginnt der Kernel-Loop im Takt 41 und endet in Takt 52. Der Kernel-Loop ist bei diesem Algorithmus im Vergleich zu den vorherigen Funktionen relativ lang. Dies kommt durch die Tatsache, dass die letzten drei geladenen Bildpunktsätze vorhanden sein müssen, um die ältesten Werte von der Summe subtrahieren zu können.

Nachdem zwei Bildspalten parallel verarbeitet wurden, werden die Adresszeiger auf den Anfang der folgenden beiden Spalten gesetzt und der Algorithmus startet erneut. Die grau dargestellten Instruktionen am Ende des Algorithmus dienen dabei schon als Vorbereitung für die jeweils folgende Iteration. Ist das Bild komplett bearbeitet, erfolgt der Rücksprung zur aufrufenden Funktion.

Mit dieser Funktion kann durch entsprechende Parameter das Opening durchgeführt werden. Zunächst wird diese Funktion mit einem Schwellwert von acht und einer Maske von null aufgerufen, was einer Erosion entspricht. Alle gesetzten Bildpunkte, bei denen die umliegenden Bildpunkte gesetzt sind, werden auf Eins gesetzt, die übrigen auf Null. Anschließend erfolgt die Dilatation, indem die selbe Funktion mit einem Schwellwert von null und der gleichen Maske wie bei der Erosion ausgeführt wird. In diesem Fall werden alle Bildpunkte gesetzt, die im Originalbild gesetzt sind oder mindestens einen gesetzten Nachbarn haben.

Um das Closing zu berechnen, wird genauso vorgegangen, jedoch werden die Schwellwerte umgekehrt verwendet. Zunächst wird eine Dilatation mit einem Schwellwert von null und anschließend eine Erosion mit einem Schwellwert von fünf durchgeführt. Bei der abschließenden Erosion wird die alternative Funktion ohne die Maskierung verwendet, damit in den einzelnen Bildpunkten jeweils die Summen für die folgende Merkmalsextraktion enthalten sind.

Die Ausführungszeit einer Funktion bei einem Bild mit einer Größe von 320x240 Pixeln beträgt ca. 86 μs. Für das Opening und Closing werden also jeweils 172 μs benötigt. Im Vergleich zur vorangegangenen Farbklassifikation, die bei doppelter Datenmenge 7.20 ms benötigt, stellt dies einen gewaltigen Unterschied dar. Dieser Unterschied kommt dadurch zustande, dass sich die Eingangsdaten für die Farbklassifikation im externen SDRAM und die für das Opening und Closing im deutlich schnelleren internen RAM befinden.

6.5 Merkmalsextraktion

Ausgehend von dem Ergebnis der Erosion beim Closing wird in diesem Abschnitt die Umsetzung der Merkmalsextraktion erläutert. Die gesetzten Bildpunkte im vorliegenden Bild haben die Werte von 6 bis 9. Anhand dieser Bildpunkte müssen nun die benötigten Eigenschaften ermittelt werden, um im folgenden Abschnitt die Position des gesuchten Objekts zu ermitteln. Alle Funktionen und Berechnungen werden in der Funktion *calc_parameter()* durchgeführt. Zum Zählen der Bildpunkte wird eine separate Assemblerfunktion entwickelt, da der Algorithmus recht einfach umzusetzen ist und sich dadurch erheblich Rechenzeit einsparen lässt.

Für die Merkmalsextraktion wird die Anzahl der Bildpunkte nach ihrer Wertigkeit getrennt benötigt. Dazu wird zunächst eine Funktion entwickelt, die die Anzahl der einzelnen Bildpunkte berechnet. Da diese Funktion mehrfach verwendet wird, wird diese auch wieder als Assemblerfunktion entwickelt um in möglichst kurzer Ausführungszeit das Ergebnis zu erhalten. Der Ablauf sieht so aus, dass mit der LDDW Instruktion 8 Byte auf einmal geladen werden. Anschließend werden die einzelnen Byte mit der CMPEQ4 (Compare for Equality, Packed 8-Bit) Instruktion mit dem gesuchten Wert verglichen. Das Ergebnis der vier Vergleiche, die diese Instruktion auf einmal ausführt, wird jeweils in den untersten vier Bit gespeichert. Da mit jeder LDDW Instruktion 8 Bildpunkte geladen werden, muss diese Instruktion zwei mal hintereinander ausgeführt werden. Anschließend wird mit der BITC4 (Bit Count, Packed 8-Bit) Instruktion die Anzahl der gesetzten Bits gezählt. Das Ergebnis kann dementsprechend die Werte von null bis vier im untersten Byte des 32 Bit Registers annehmen. Diese Instruktion wird auf die Ergebnisse beider Vergleichsoperationen angewendet. Anschließend wird die Anzahl der gesetzten Bits in einem extra Register aufaddiert.

Auch bei diesem Algorithmus wird nur eine Hälfte des Prozessors benötigt, so dass mit der zweiten Hälfte parallel dazu die nächste Zeile bearbeitet werden kann. Die LDDW Instruktion benötigt auch hier wieder fünf Takte für die Ausführungen. Bei der BITC4 Instruktion werden zwei Takte benötigt und die übrigen Instruktionen sind alle innerhalb eines Taktes ausgeführt.

Anhand dieses Ablaufs wird wieder das Softwarepipelining durchgeführt, um eine möglichst gute Auslastung der Hardwareeinheiten zu erhalten (siehe Tabelle 6.5). Am Anfang werden die Register vorbereitet und die Pipeline des Prozessors gefüllt. Dann wird der Kernel-Loop so oft hintereinander ausgeführt, bis das Ende der Bildzeilen erreicht ist. Anschließend werden die letzten noch angefangenen Iterationen beendet. Danach wird mit den nächsten beiden Zeilen fortgefahren. Ist das Ende des Bildes erreicht, werden die Ergebnisse aus den geraden und ungeraden Bildzeilen addiert und an die aufrufende Funktion zurückgegeben.

Tabelle 6.5: Softwarepipelining für Bildpunktzählung

Takt	Paket	Anfang der Iteration	Ende der Iteration	Label	D1	L1	S1	M1	D2	L2	S2	M2	parallele Instruktionen
1	1			preparation		MV		MPY				MPY	3
2	2									MV			1
3	3						SHRU				SHRU		2
4	4						SUB				ADD		2
5	5	1		prolog	LDDW	ZERO			LDDW	ZERO			4
6	6					MV				MV			2
7	7												
8	8	2			LDDW				LDDW				2
9	9						BDEC				BDEC		2
10	10						CMPEQ4				CMPEQ4		2
11	11	3			LDDW		CMPEQ4	BITC4	LDDW		CMPEQ4	BITC4	6
12	**12**			inner_loop			**BDEC**	**BITC4**			**BDEC**	**BITC4**	**4**
13	**13**					**ADD**	**CMPEQ4**			**ADD**	**CMPEQ4**		**4**
14	**14**	4	1		**LDDW**	**ADD**	**CMPEQ4**	**BITC4**	**LDDW**	**ADD**	**CMPEQ4**	**BITC4**	**8**
15	**12**						BDEC	BITC4			BDEC	BITC4	4
16	**13**					ADD	CMPEQ4			ADD	CMPEQ4		4
17	**14**	5	2		LDDW	ADD	CMPEQ4	BITC4	LDDW	ADD	CMPEQ4	BITC4	8
18	**12**							BITC4			BDEC	BITC4	4
19	**13**					ADD	CMPEQ4			ADD	CMPEQ4		4
20	**14**		3		LDDW	ADD	CMPEQ4	BITC4	LDDW	ADD	CMPEQ4	BITC4	**8**
21	**12**							BITC4			BDEC	BITC4	**4**
22	**13**					ADD	CMPEQ4			ADD	CMPEQ4		**4**
23	**14**		4		LDDW	ADD	CMPEQ4	BITC4	LDDW	ADD	CMPEQ4	BITC4	8
24	15			epilog				BITC4			B	BITC4	3
25	16					ADD				ADD			2
26	17		5			ADD				ADD			2
27	18					ADD							1
28	19												
29	20												

Mit dieser Funktion werden aus dem Ergebnis der Erosion die Anzahl der unterschiedlichen Bildpunkte bestimmt. Die Bildpunkte können, wie bereits erwähnt, die Werte von sechs bis neun annehmen. Dementsprechend wird diese Funktion vier mal ausgeführt (siehe Listing 6.3). Die Summe der vier Teilergebnisse entspricht dabei dem Flächeninhalt des Objekts. Betrachtet man nur die Bildpunkte mit den Werten von fünf bis acht, so erhält man den Umfang des Objekts. Damit wurden schon einmal zwei Merkmale innerhalb von ca. 144 μs berechnet, da die Ausführungszeit der Funktion bei ca. 36 μs liegt.

```
...
//Flächenberechnung
area  = count_pixel_asm ( ptr_src , xmax, ymax, PIXEL9_PACKED );    //36us

//Umfang berechnen
 perimeter   = count_pixel_asm ( ptr_src , xmax, ymax, PIXEL8_PACKED );    //36us
 perimeter  += count_pixel_asm ( ptr_src , xmax, ymax, PIXEL7_PACKED );    //36us
 perimeter  += count_pixel_asm ( ptr_src , xmax, ymax, PIXEL6_PACKED );    //36us

 //der Umfang gehört mit zum Objekt
 area += perimeter;
...
```

Listing 6.3: Berechnung des Flächeninhalts und Umfangs (Ausschnitt aus calc_parameter.c)

Anschließend wird der Massenschwerpunkt ermittelt. Die Berechnung erfolgt nach den Gleichungen 2.6 in X-Richtung und 2.7 in Y-Richtung aus Kapitel 2.4.4. Es werden die Summen der X- und Y-Koordinaten aller Bildpunkte des Objekts addiert und anschließend durch den Flächeninhalt des Objekts geteilt (siehe Listing 6.4).

```
...
// Berechnung des Massenschwerpunktes
 mid_x = 0;
 mid_y = 0;
 for ( y = 0; y < ymax; y++ )
 {
     for ( x = 0; x < xmax; x++ )
     {
       if ( ptr_src[x+y*xmax] != 0x00 )
       {
          mid_x += x;
          mid_y += y;
       }
     }
 }

 mid_x = mid_x / area;
 mid_y = mid_y / area;
...
```

Listing 6.4: Berechnung des Massenschwerpunkts (Ausschnitt aus calc_parameter.c)

Abschließend wird noch die Kompaktheit nach Gleichung 2.4.3 aus Kapitel 2.5 berechnet und die Ergebnisse werden in eine Datenstruktur gespeichert (siehe Listing 6.5). Damit sind alle benötigten Merkmale berechnet, um zu entscheiden, welche Form das Objekt hat und um die Position zu bestimmen.

```
...
//Daten in Struktur eintragen
data->area        = area;
data->perimeter   = perimeter;
data->compactness = ( perimeter * perimeter ) / area;
data->center[0]   = mid_x;
data->center[1]   = mid_y;
...
```

Listing 6.5: Kompaktheit und Ergebnisübergabe (Ausschnitt aus calc_parameter.c)

Diese Funktion wird wie die vorangegangenen Funktionen auf der Stereo-Plattform auf beide Kamerabilder angewendet. Über die Kompaktheit kann eine Entscheidung getroffen werden, ob das Objekt rund ist, was bei einem Ball der Fall sein sollte. Dies kann als Kriterium dafür dienen, ob eventuelle Störungen im Bild aufgetreten sind und daher der Massenschwerpunkt nicht richtig ermittelt wurde.

Anhand des Flächeninhaltes kann eine Aussage über die Qualität der Kameras zueinander getroffen werden. Sofern das gesuchte Objekt bei beiden Kameras vollständig im Bild ist und es sich in gleicher Entfernung von beiden Kameras befindet, sollte der berechnete Flächeninhalt nahezu identisch sein. Es stellte sich dabei heraus, dass die Abweichung beim Flächeninhalt bei über 50 % lag. Durch den Einsatz einer Ersatzkamera lies sich dieses Problem jedoch lösen. Die Unterschiede scheinen durch Fertigungstoleranzen beim Sensor und/oder Farbfilter zu entstehen.

6.6 Positionsbestimmung

Bevor die Position des Objekts anhand der Gleichung aus Kapitel 2.5 bestimmt werden kann, muss zunächst ein Bezugspunkt festgelegt werden, von dem aus die Entfernung berechnet wird. Als Bezugspunkt wird die Kamera der Ethernet-Plattform gewählt.

Die Disparität ergibt sich dann aus der Subtraktion der X-Koordinaten der Massenschwerpunkte. Die zweite Kamera (Stereo-Plattform) ist um $b = 12\,\mathrm{cm}$ nach recht verschoben. Dementsprechend ist die X-Koordinate des Massenschwerpunktes im Bild nach links verschoben und vom Wert her kleiner als die der anderen Kamera. Die Disparität im Bezug auf die erste Kamera ergibt sich danach durch Subtraktion der X-Koordinate der zweiten Kamera von der der ersten Kamera.

Die folgenden drei Gleichungen entsprechen den Gleichungen aus Kapitel 2.5, wobei die Variable a für den Umrechnungsfaktor von Bildpunkten in Metern steht. Außerdem wurde die Gleichung für die Entfernung in die anderen beiden Gleichungen eingesetzt. Dadurch ergibt sich eine Vereinfachung für die Berechnung. Bei allen Koordinaten wird jeweils der Quotient aus dem Abstand der Kameras b und der Disparität d verwendet.

$$Z = \frac{b}{a \cdot d} \cdot f = \frac{b}{d} \cdot \frac{f}{a} \tag{6.4}$$

$$X = \frac{Z \cdot a \cdot x_1}{f} = \frac{b \cdot f}{a \cdot d} \cdot \frac{a \cdot x_1}{f} = \frac{b}{d} \cdot x_1 \tag{6.5}$$

$$Y = \frac{Z \cdot a \cdot y_1}{f} = \frac{b \cdot f}{a \cdot d} \cdot \frac{a \cdot y_1}{f} = \frac{b}{d} \cdot y_1 \tag{6.6}$$

Bevor die eigentliche Berechnung durchgeführt werden kann, muss der Umrechnungsfaktor bestimmt werden. Im Datenblatt ist angegeben, dass bei einer Auflösung von 640x480 Bildpunkten in horizontaler und vertikaler Richtung jeweils nur jeder vierte Pixel des Sensors ausgelesen wird (siehe [10]). Bei einer Pixelkantenlänge von 2.2 μm ergibt sich daraus ein Abstand von 8.8 μm. Durch die Farbklassifikation wird ein Bildpunkt aus einem 2x2 Bildpunkte großen Feld des 640x480 Bildpunkte großen Bildes gebildet. Dementsprechend verdoppelt sich der Pixelabstand auf 17.6 μm.

Außerdem müssen die Koordinaten (x_1 und y_1) des Massenschwerpunktes des Objekts der Bezugskamera auf den Bildmittelpunkt als Ursprung umgerechnet werden. Dadurch stellt der Bildmittelpunkt sowohl horizontal als auch vertikal den Ursprung dar. Dazu muss von der Hälfte der Bildauflösung in der jeweiligen Richtung die entsprechende Koordinate des Massenschwerpunktes subtrahiert werden. Der Wertebereich ändert sich dann für x_1 auf -159 bis $+160$ und für y_1 auf -119 bis $+120$. Darüber hinaus steht die Disparität im Nenner und darf dementsprechend nur verwendet werden, wenn sie größer als Null ist.

Um die Berechnung möglichst schnell auszuführen, wird auf eine Fließkommaberechnung verzichtet. Dazu müssen die Parameter b, f und a vorher entsprechend umgerechnet werden. Außerdem muss darauf geachtet werden, dass der Wertebereich optimal ausgenutzt wird, um Quantisierungsfehler zu minimieren. Der minimale Fehler ergibt sich, wenn der Abstand der Kameras b in Metern mit dem Faktor 2^{21} und der Quotient aus Brennweite und Pixelabstand mit dem Faktor 2^5 multipliziert wird.

Da die Multiplikationen die Gleichung verändern, müssen diese an geeigneter Stelle rückgängig gemacht werden. Listing 6.6 zeigt die sich daraus ergebenden Berechnungen.

```
...
if ( ( g_img_data[0].area > OBJEKT_MINIMUM_AREA ) &&
     ( g_img_data[1].area > OBJEKT_MINIMUM_AREA ) )
{
  b_by_d = b / ((Uint32)(g_img_data[0].center[0]-g_img_data[1].center[0]));

  p2_result->X = ( ((signed int)(( 1000 * b_by_d ) >> 5 )) * (g_img_data[0].center[0]-
                                                    (CAM_FRAMEWIDTH_GRAY/2)) ) >> 16;

  p2_result->Y = ( ((signed int)(( 1000 * b_by_d ) >> 4 )) * (g_img_data[0].center[1]-
                                                    (CAM_FRAMEHEIGHT_GRAY/2)) ) >> 17;

  p2_result->Z = ( ( ( f_by_a * b_by_d ) >> 10 ) * 1000 ) >> 16;
}
   ...
```

Listing 6.6: Postionsberechnung (Ausschnitt aus img_process_stereo.c)

Im ersten Schritt wird der Quotient aus b und d berechnet. Durch die vorherige Multiplikation von b mit 2^{21} ist der Quotient 18 Bit breit. Anschließend wird dieser Wert bei der X- und Y-Koordinate mit dem Faktor 1000 multipliziert um die Einheit Millimeter zu erhalten. Damit es bei der darauf folgenden Berechnung zu keinem Zahlenbereichsüberlauf innerhalb des 32 Bit Registers kommt, wird der Wert entsprechend geschoben. Nachdem die abschließende Multiplikation mit der Objektposition im Bild multipliziert wurde, wird das Ergebnis erneut geschoben, um die anfängliche Multiplikation mit 2^{21} vollständig zu kompensieren.

Bei der Berechnung der Entfernung Z sieht die Berechnung etwas anders aus. Der Quotient aus Brennweite f und Pixelabstand a wurde mit dem Faktor 2^5 multipliziert und ist 14 Bit breit. Das Ergebnis der Multiplikation mit dem 18 Bit breiten Quotienten aus b und d ergibt ein Ergebnis, das maximal 32 Bit verwendet. Damit dieser Wert mit dem Faktor 1000 (10 Bit breit) multipliziert werden kann, muss dieser zunächst um 10 Bit nach rechts geschoben werden, damit es keinen Zahlenbereichsüberlauf gibt. Abschließend wird der Wert um weitere 16 Bit nach rechts geschoben, um die vorherige Multiplikation mit 2^{21} und 2^5 zu kompensieren. Der maximale relative Fehler bei der Festkommaberechnung der Entfernung Z liegt bei ca. 0.27 % (siehe fixpoint.m im Anhang A).

6.7 Grauwertbild

Für die Visualisierung wird aus dem Bayer Mosiak Bild der Kamera ein Grauwertbild extrahiert. Dies geschieht durch die Assemblerfunktion *convert_asm()*. Da die für die Farbklassifikation entwickelte Assemblerroutine schon einen so deutlichen Geschwindigkeitsvorteil ergeben hat, wurde diese Funktion direkt in Assembler programmiert. Daher kann an dieser Stelle keine Aussage über den zu erwartenden Geschwindigkeitsvorteil gemacht werden.

Das Grauwertbild wird aus den grünen Bildpunkten des Originalbildes gebildet, indem die grünen Bildpunkte aus den ungeraden Zeilen verwendet werden. Dadurch ergibt sich eine Auflösung von 320x240 Bildpunkten, die mit der Auflösung der Farbklassifikation übereinstimmt. Da die Bilder die selbe Auflösung besitzen, können die Ergebnisse unmittelbar in das Grauwertbild eingezeichnet werden.

Die Extraktion der grünen Bildpunkte erfolgt in drei einfachen Schritten. Zunächst werden mit der LDDW Instruktion acht Bildpunkte aus dem Bayer Mosaik Bild geladen. Anschließend werden mit der PACKH4 Instruktion die grünen Pixel aus den beiden Registern entnommen und in den Zielspeicher geschrieben. Es werden dabei immer nur die Zeilen bearbeitet, in denen sich blaue und grüne Bildpunkte befinden. Das Ergebnis besteht dann nur aus grünen Bildpunkten, die als Grauwertbild interpretiert werden. Es wurden bewusst die grünen Bildpunkte verwendet, da die menschliche Wahrnehmung bei grün besonders empfindlich ist. Das Ergebnis ist in Bild 6.9 dargestellt.

Bild 6.9: Ergebnis der Grauwertbildberechnung

Diese Funktion benötigt für die Extraktion der grünen Bildpunkte ca. 3.06 ms. Eigentlich sollte die Ausführungsdauer bei ca. 20 μs liegen, dies ist jedoch nicht möglich, da sich das Originalbild aufgrund seiner Größe im langsamen externen SDRAM befindet.

6.8 Ergebniszusammenführung

Sobald auf der Stereo-Plattform alle Berechnungen abgeschlossen sind und die Ergebnisse an die Ethernet-Plattform über McBSP2 übertragen wurden, werden die Ergebnisse in das Grauwertbild eingetragen. Dies geschieht durch die Funktion *merge_image()*.

Anhand der Ergebnisse der Bildverarbeitung auf der Ethernet-Plattform wird die Fläche und der Umfang des Objekts in das Grauwertbild eingezeichnet. Dazu werden die Bildpunkte, die zum Objekt gehören mit dem Wert 255 überschrieben. Die Pixel, die den Umfang darstellen, werden mit dem Wert 254 überschrieben. Anschließend werden die beiden Massenschwerpunkte, die durch die Stereo-Plattform berechnet wurden, durch zwei Kreuze mit den Werten 253 in das Grauwertbild eingetragen.

6.9 Compileroptimierung

In den Projekteinstellungen wird als Compileroptimierung die höchste Stufe O3 verwendet. Bei einigen Dateien muss diese Optimierung allerdings abgeschaltet werden, damit das System ordnungsgemäß arbeitet.

Beim Bildverarbeitungstask und der Bildverarbeitungsfunktion muss die Optimierung auf beiden Plattformen ausgeschaltet werden. Außerdem muss der Task für die Verbindungsherstellung über die Ethernet-Schnittstelle ebenfalls ohne Optimierung compiliert werden.

Wird der Bildverarbeitungstask dennoch mit O3 compiliert, so funktionieren die Semaphoren erstaunlicherweise nicht mehr. Darüber hinaus wird die Interruptdeaktivierung innerhalb der Bildverarbeitungsfunktion bei eingeschalteter Optimierung einfach weg optimiert und die Assemblerfunktionen arbeiten nicht mehr einwandfrei.

6.10 Visualisierung mit MATLAB

Die Visualisierung der Ergebnisse erfolgt mit MATLAB. Dafür wurden mehrere Dateien entwickelt, die in diesem Abschnitt erläutert werden. Zunächst wird die Datei *streamimage.m* gestartet.

Am Anfang wird der Benutzer nach der IP-Adresse und der Portnummer der Ethernet-Plattform gefragt. Anschließend wird eine Figure erstellt, in der die Ergebnisse dargestellt werden (siehe Bild 6.10). Die Darstellung besteht dabei aus mehreren Teilen. Zum einen wird das bearbeitete Grauwertbild mit den beiden Massenschwerpunkten gezeigt, zum anderen werden die ermittelten Werte als Zahlenwerte ausgegeben. Darunter befinden sich neben den Koordinaten der Massenschwerpunkte auch die Angaben über Flächeninhalt, Umfang, Kompaktheit und die Position im Raum. Zusätzlich ist daneben noch ein 3D-Plot eingefügt, in dem Positionsangaben eingezeichnet werden. Es werden dabei immer nur die letzten 128 Messwerte eingetragen.

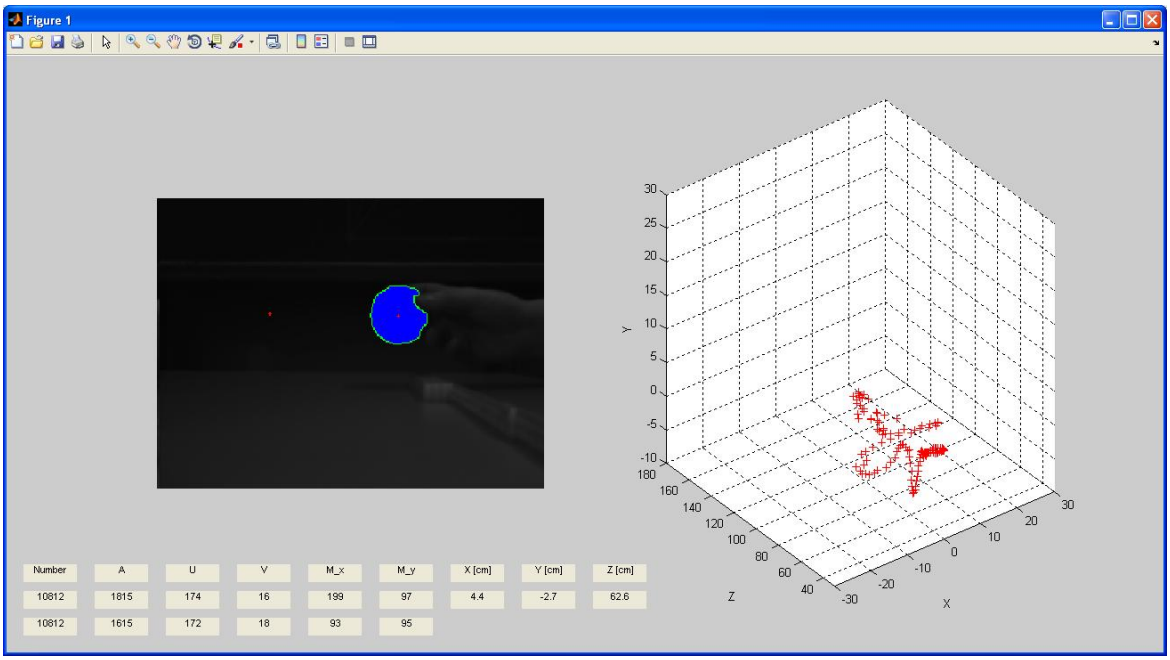


Bild 6.10: Visualisierung der Positionsbestimmung des Objekts im Raum

Nachdem die Figure für die Anzeige aufgebaut ist, wird ein TCP-Objekt erstellt, dass die Verbindung herstellen soll. In dieses Objekt werden die zuvor durch den Benutzer eingegebenen Verbindungsdaten eingetragen. Darüber hinaus wird noch ein Puffer für die ankommenden Daten reserviert. Damit nicht permanent der Füllstand des Puffers abgefragt

werden muss, wird eine Callback-Funktion definiert, die ausgeführt wird, sobald ein komplettes Bild und die Messwerte im Puffer enthalten sind. Im Anschluss an die Konfiguration wird die Verbindung hergestellt und das Skript pausiert bis eine Taste gedrückt wird.

Bei der Callback-Funktion handelt es sich um die Funktion *showFrame* aus der gleichnamigen Datei. Diese Funktion wird, wie bereits erwähnt, immer dann ausgeführt, wenn die benötigte Menge an Daten im Empfangspuffer des TCP-Objekts vorhanden ist. Am Anfang der Funktion werden dann die Daten aus dem Puffer ausgelesen und verarbeitet.

Bevor das Bild letztendlich angezeigt werden kann, muss es zunächst um 90° gedreht werden, da die Interpretation von Zeilen und Spalten in MATLAB anders ist. Außerdem muss das Grauwertbild in ein Indexbild konvertiert werden, damit bestimmte Grauwerte als Farbe dargestellt werden können. Im Kapitel 6.8 wurde den Bildpunkten der Fläche, des Umfangs und den Massenschwerpunkten bestimmte Grauwerte zugewiesen. Diese werden durch Veränderung der Farbpalette als Farbe dargestellt. Anschließend wird das Bild und die Daten in der Anzeige aktualisiert und die Funktion wird beendet.

Während einer laufenden Übertragung kann durch anklicken des Bildes eine Kopie auf der Festplatte gespeichert werden. Dies funktioniert ebenfalls über eine separat definierte Callback-Funktion (*saveImage*). Die beim Schließen der Figure ausgeführte Callback-Funktion (*closeFigure*) ist für das Beenden der Ethernetverbindung und das Aufräumen des Speichers zuständig.

7 Ergebnisse

Dieses Kapitel befasst sich mit den erzielten Ergebnissen. Angefangen wird mit der Hardware über die Software bis hin zum eigentlichen Bildverarbeitungsalgorithmus.

Bild 7.1 zeigt den Aufbau der Hardware mit der Verbindung zwischen den Plattformen. Im oberen Teil des Bildes ist die Ethernet-Plattform und im unteren Teil die Stereo-Plattform zu erkennen.

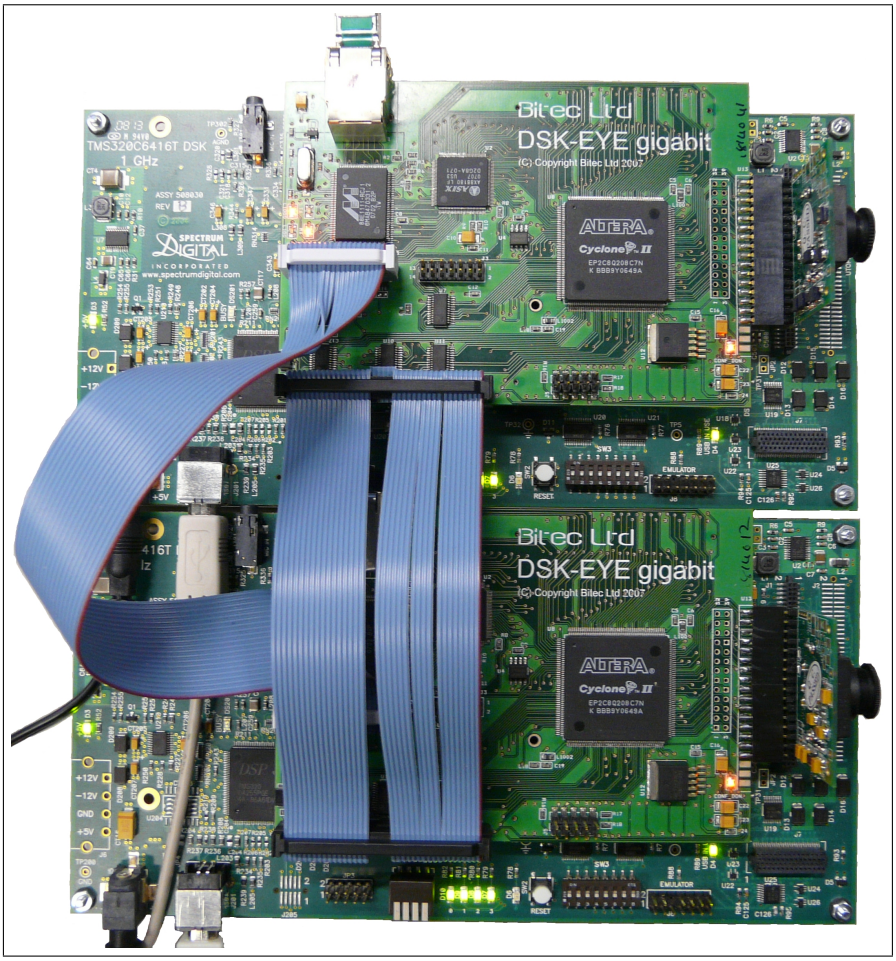


Bild 7.1: Aufbau des Stereokamerasystems

Bei dem mitgelieferten Beispielprojekt kam es hin und wieder vor, dass das Bild anfing zu „rollen“. Die genaue Ursache dafür ließ sich nicht klären, aber dieses Problem ließ sich durch die bereits erwähnte Rückmeldung vom DSP an das FIFO beheben.

Bei diesem Aufbau sind zwei physikalische Probleme aufgetreten, von denen eines umgangen werden konnte. Bei der Übertragung des Kamerataktsignals von der Stereo- zur Ethernet-Plattform kommt es aufgrund einer schlechten Signalintegrität zu einer Taktverdopplung. Dieses Problem wurde durch den Einsatz einer PLL auf der Ethernet-Plattform gelöst, was zugegebener Maßen keine einwandfreie Lösung des Problems darstellt, aber einen stabilen Betrieb zulässt. Bild 7.2 zeigt die Messung des VSYNC Signals der beiden Kameras. Der zeitliche Versatz zwischen den Kameras beträgt lediglich 35.29 ns.

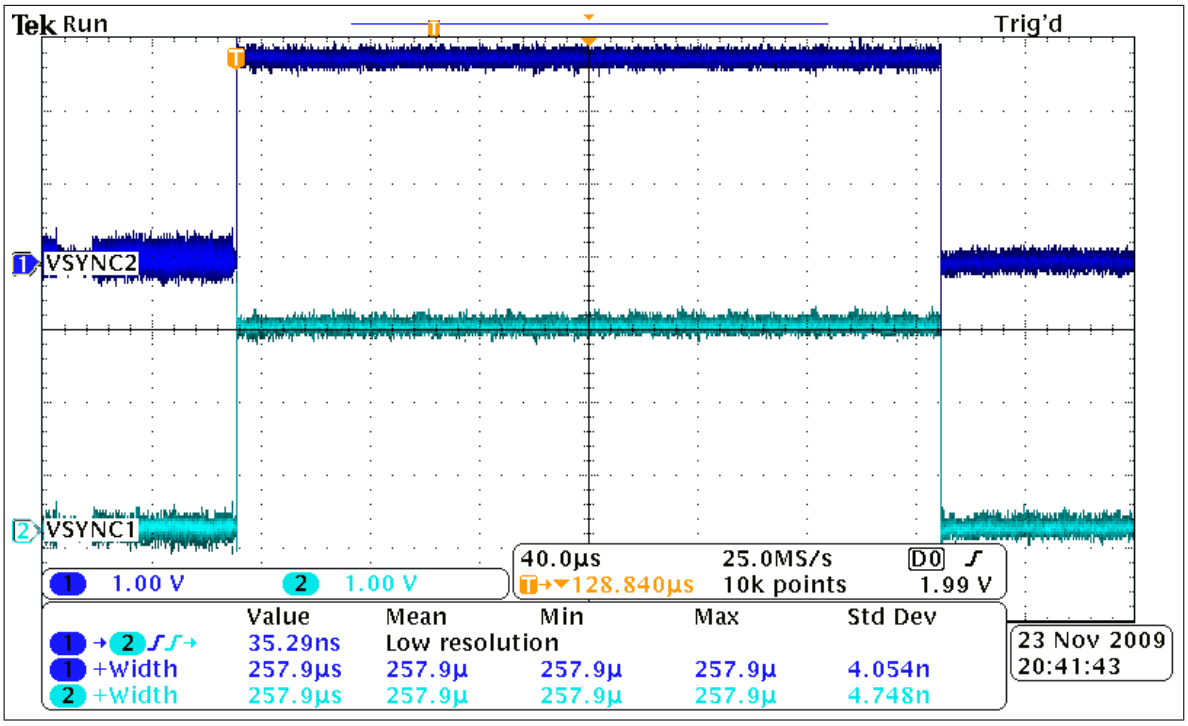


Bild 7.2: Messung der Kamerasynchronität

Das zweite Problem ließ sich allerdings nicht lösen. Bei der paketorientierten Datenübertragung über McBSP mit dem EDMA Kontroller kommt es in unbestimmten Zeitabständen vor, dass Datenworte verloren gehen. Dadurch kommt der EDMA Kontroller durcheinander und die einzelnen Parameter sind nicht mehr in der richtigen Reihenfolge im Speicher. Eine Synchronisation, wie zwischen EDMA Kontroller und dem FIFO im FPGA ist nicht möglich, da die dafür benötigten Signale nicht mit dem FPGA verbunden sind.

Um dennoch einen stabilen Betrieb zu erreichen, war es nötig, die paketorientierte Übertragung auf eine Richtung zu beschränken. Die Schwellwerte für die Farbklassifikation, die von der Ethernet-Plattform an die Stereo-Plattform gesendet werden sollten, werden nicht mehr übertragen, sondern durch Konstanten ersetzt. Die Übertragung der Ergebnisse der Bildverarbeitung von der Stereo- zur Ethernet-Plattform funktioniert nach der Abschaltung der anderen Übertragungsrichtung einwandfrei.

Über die Einschaltdauer von LED2 kann die Ausführungszeit der Bildverarbeitungstasks auf beiden Plattformen gemessen werden. Das Ergebnis ist in Bild 7.3 dargestellt. Damit das System in Echtzeit arbeitet, dürfen die Bildverarbeitungstasks maximal 32 ms für die Berechnungen benötigen. Bei den hier implementierten Algorithmen liegt die Ausführungsdauer bei etwa 17 ms auf der Stereo-Plattform und bei etwa 18 ms auf der Ethernet-Plattform. Es stehen demnach noch genügend Reserven für weitere Algorithmen zur Verfügung.

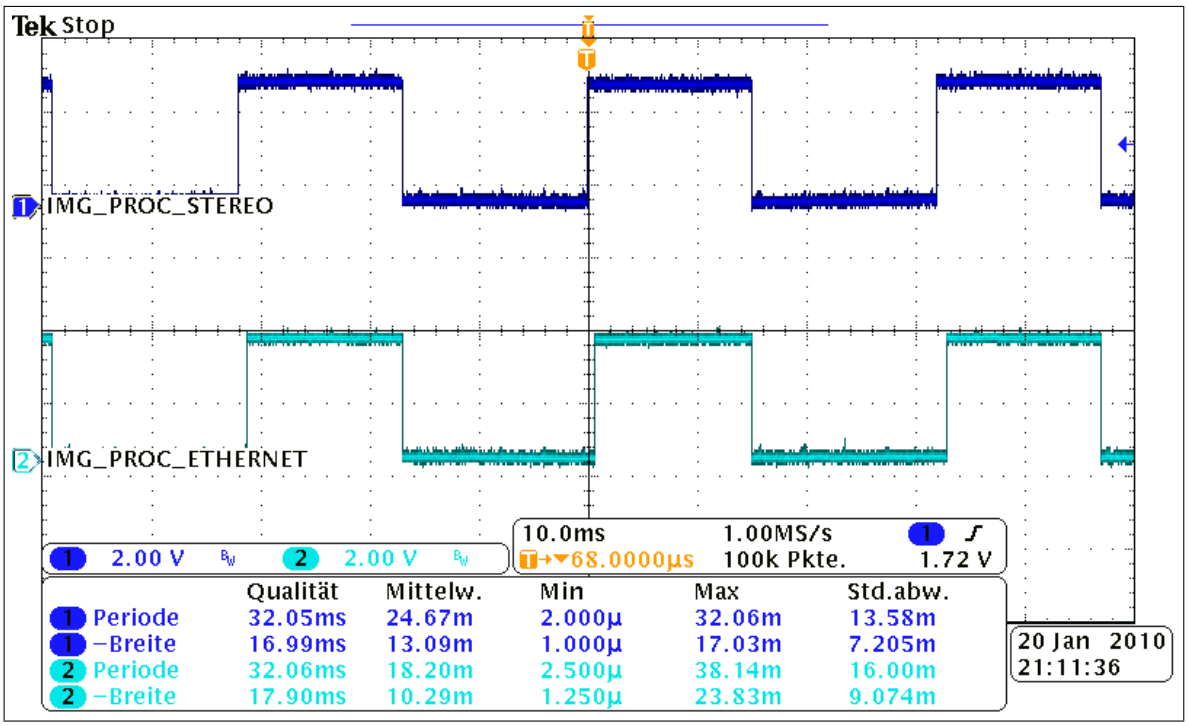
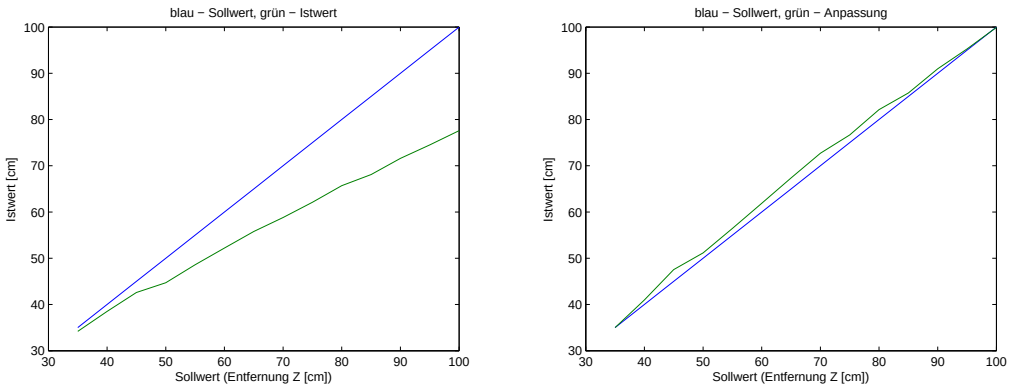


Bild 7.3: Messung der Laufzeit des Bildverarbeitungstasks

Das Echtzeitverhalten ist damit sichergestellt. Im folgenden geht es darum, die Genauigkeit der Positionsbestimmung nachzuweisen. Bild 7.4(a) zeigt eine Messung, bei der der Abstand des Objekts in Z-Richtung von 35 bis 100 cm in 5 cm Schritten gemessen wurde (X- und Y-Richtung sind konstant). Die blaue Kurve stellt den Sollwert und die grüne den Messwert dar.



(a) Messung (b) optimierte Messung

Bild 7.4: Messung vor und nach Optimierung der Position in Z-Richtung

Mit steigender Entfernung wird die Abweichung zwar größer, aber der Verlauf scheint dennoch linear zu sein. Aus diesem Grund wurde der Offset und der Mittelwert der Steigung über diese Messwerte berechnet (siehe Z_anpassung.m in Anhang A). Diese Werte werden dann bei der Berechnung der Position in Z-Richtung berücksichtigt. Mit dieser Anpassung lassen sich die Messfehler verringern. Bild 7.4(b) zeigt eine erneute Messung mit korrigierten Werten. Es ist deutlich zu erkennen, dass die Abweichung vom Sollwert geringer geworden ist. In Bild 7.5 ist der absolute Fehler mit (grün) und ohne (blau) Anpassung zu sehen.

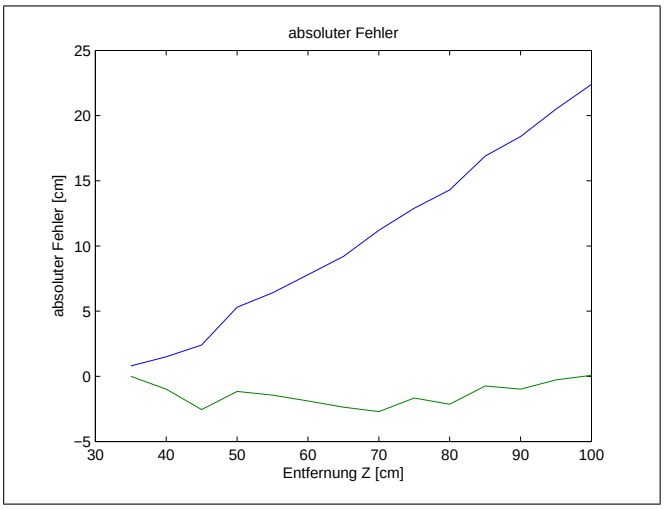


Bild 7.5: Vergleich des absoluten Fehlers in Z-Richtung

Tabelle 7.1 zeigt die errechneten Parameter der beiden Messungen. Bei der angepassten Messung wird auf der Stereo-Plattform die Entfernung Z mit einem Korrekturfaktor von 1.4977 multipliziert und anschließend wird noch ein Offset von 16.4 cm hinzu addiert. Dadurch wird die Entfernungsberechnung deutlich verbessert. Die maximale Abweichung vom Sollwert beträgt jetzt nur noch 2.7 cm und nicht mehr 22.4 cm.

Tabelle 7.1: Ergebnisse der Korrekturrechnung in Z-Richtung

	Messung	optimierte Messung
Abweichung vom Sollwert	0.8 . . . 22.4 cm	-2.7 . . . 0.1 cm
Mittelwert der Steigung	0.66769	0.99888
relativer Fehler	-22.4% . . . -2.28%	-0.07% . . . 5.66%

Nachdem die Berechnung der Position in Z-Richtung optimiert ist, wird dieses Verfahren für die anderen beiden Koordinaten durchgeführt. Der Messbereich ist dabei auf Grund des eingeschränkten Sichtfeldes deutlich kleiner. Die Messung erfolgt im Bereich von 0 bis 12 cm in 2 cm Schritten. Bild 7.6 zeigt den absoluten Fehler der Messung (blau) und der erneuten Messung (grün) mit der Optimierung. Bei dieser Messung wurde die Position in Y- und Z-Richtung konstant gehalten.

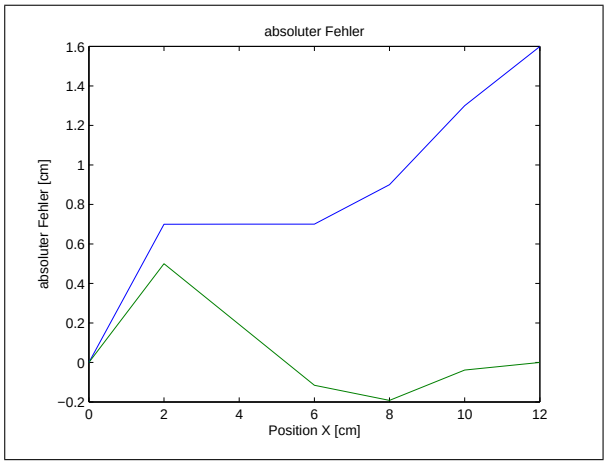


Bild 7.6: Vergleich des absoluten Fehlers in X-Richtung

Der Korrekturfaktor beträgt für die Messung in X-Richtung 1.1538. Dadurch verringert sich der absolute Fehler von 1.6 cm auf 0.5 cm (siehe XY_anpassung.m in Anhang A).

Bei der Messung in Y-Richtung wird ein Korrekturfaktor von 1.1881 und ein Offset von 0.4 cm verwendet. Der absolute Fehler (siehe Bild 7.7) verringert sich dadurch von 1.5 cm (blau) auf 0.34 cm (grün).

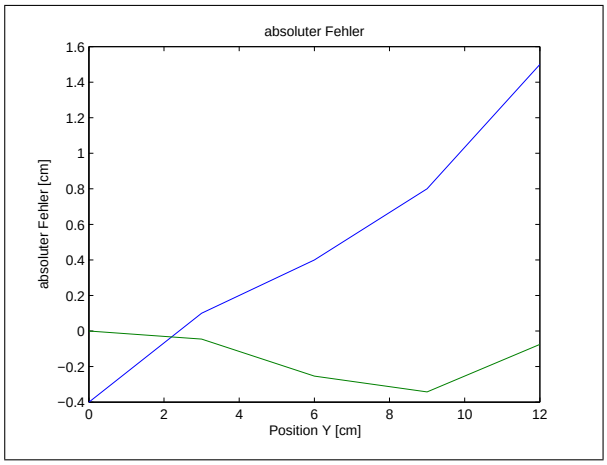


Bild 7.7: Vergleich des absoluten Fehlers in Y-Richtung

Zusammenfassend ist zu sagen, dass die Messung der Position im Raum je nach Richtung mit einer gewissen Ungenauigkeit belastet ist. Dies kommt zum einen durch die Optik, die Verzerrungen verursacht, und zum anderen durch die Bildverarbeitung selbst. Wenn Störungen im Bild einer Kamera auftreten, weicht der Massenschwerpunkt eventuell vom Mittelpunkt des Objekts ab. Dies hat zur Folge, dass sich die Disparität ändert. Da diese aber ausschlaggebend für die Berechnung der Position ist, tritt eine Abweichung auf.

Außerdem spielt die Ausrichtung der Kameras eine große Rolle. Bei diesem Aufbau können die Kameras nur nach Augenmaß zueinander ausgerichtet werden. Die vertikale Ausrichtung kann über die y-Koordinaten der Massenschwerpunkte angepasst werden, da diese in beiden Bildern bei optimaler Ausrichtung der Kameras identisch sein muss. Tabelle 7.2 zeigt eine Zusammenfassung der absoluten Fehler nach der Optimierung der Berechnung der drei Raumkoordinaten.

Tabelle 7.2: Abweichung der Raumkoordinaten vom Sollwert

Richtung	Abweichung vom Sollwert	Messbereich	Schrittweite
X	-0.19 . . . 0.5 cm	0 . . . 12 cm	2 cm
Y	-0.34 . . . 0 cm	0 . . . 12 cm	3 cm
Z	-2.71 . . . 0.07 cm	35 . . . 100 cm	5 cm

8 Weitere Entwicklungen

Aufbauend auf den erzielten Ergebnissen und im Hinblick auf den Einsatz auf einem Roboter sind weitere Entwicklungen möglich, die in diesem Kapitel angesprochen werden.

Um die Positionsbestimmung noch weiter zu verbessern, sollten die Kameras auf einer separaten Platine angebracht werden. Dabei muss darauf geachtet werden, dass diese parallel zueinander sind, sowohl horizontal als auch vertikal. Ein entsprechender Rahmen, der die Kameras fixiert, wäre dafür sicherlich sehr hilfreich. Dies ist vor allem wichtig, wenn man bedenkt, dass sich der Roboter bewegt und die dabei auftretenden Kräfte die Ausrichtung der Kameras verändern könnten.

Bei einem Einsatz des Stereokamerasystems auf einem Roboter wäre die Visualisierung nicht mehr nötig und die Ethernet-Plattform könnte entfallen. Dadurch halbieren sich auch fast die Kosten für die Hardware. Die Kameras müssten dann aber an einer DSKEye Platine angeschlossen werden (siehe Bild 8.1). Die Länge der Flachbandkabel sollte dabei nicht wesentlich größer werden als die zur Zeit verwendeten und somit keine Probleme bei der Signalübertragung hervorrufen.

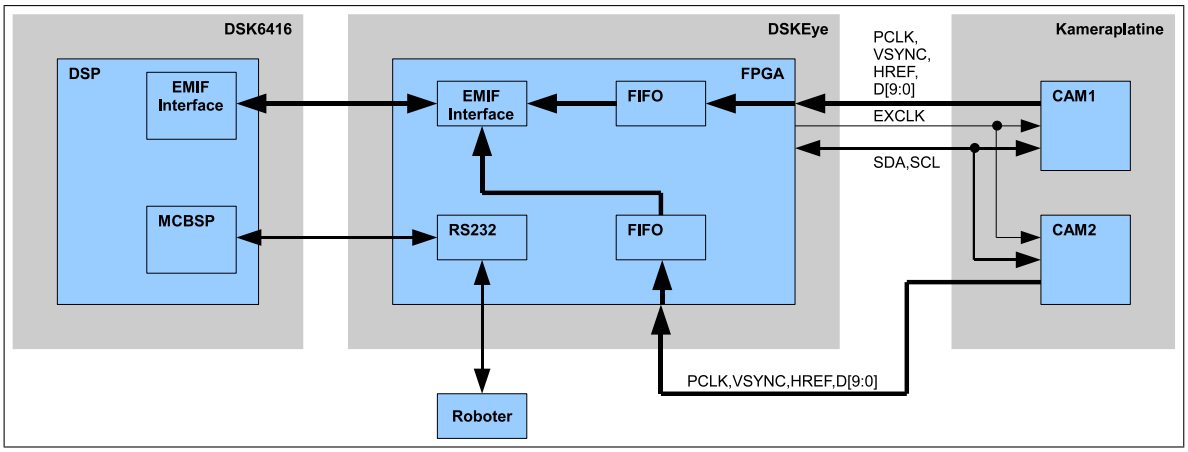


Bild 8.1: Stereokamerasystem für Robotereinsatz

Wie in dem Bild zu sehen ist, könnte das Taktsignal direkt auf der Kameraplatine mit beiden Kameras verbunden werden. Damit wäre dann wahrscheinlich auch das Problem behoben, dass bei der Taktübertragung von Board zu Board auftritt.

Die Übermittlung der Position des Objekts könnte außerdem über eine einfache serielle Schnittstelle (RS232 oder SPI), anstatt über das Ethernet, übertragen werden, da die meisten Roboter eine solche Schnittstelle besitzen. Dadurch wird die Kommunikation auch entsprechend einfacher und schneller, da unter anderem die Latenzzeiten bei TCP entfallen.

Alternativ dazu könnte man auch eine eigene Platine entwickeln, die das DSKEye Board ersetzt. Das Ethernet Interface könnte man entfernen und eventuell ein anderes FPGA verwenden. Dann wäre es auch eine Überlegung wert, teile des Bildverarbeitungsalgorithmus im FPGA zu realisieren, um dem DSP für weitere Aufgaben mehr Zeit bereitzustellen. Das Leiterplattenlayout könnte dann so gestaltet werden, dass ein synchroner Datentransfer zum DSP erfolgen kann, was die Bandbreite weiter erhöht.

Im Rahmen des Forschungsprojekts ELEKTRA an der HAW-Hamburg wird unter anderem der in Bild 8.2 gezeigt Roboter eingesetzt. Bei einem Einsatz des entwickelten Stereokamerasystems auf diesem Roboter stellt sich die Frage, inwieweit das Sichtfeld der Kameras ausreicht. Horizontal stellt dies sicherlich kein Problem dar, da sich der Roboter um die Hochachse drehen kann und somit einen Rundumblick hat. Das vertikale Sichtfeld ist bei diesem Aufbau nicht veränderbar. Um dies zu erweitern könnte man die Kameras vertikal schwenkbar über Servomotoren oder ähnliches ansteuern. Der dabei eingestellte Winkel müsste dann aber jeweils bestimmt werden und in die Berechnung mit einfließen.

Bild 8.2: Festo Robotino®[20]

9 Zusammenfassung

In dieser Arbeit wurden zwei DSP-Plattformen verwendet, um ein Stereokamerasystem zur Positionsbestimmung eines Objekts zu entwickeln.

Als Erstes ging es um die Inbetriebnahme der Hardware unter Verwendung der mitgelieferten Beispielprojekte. Anhand dieser Projekte wurden zunächst einmal mehrere Bilder aufgenommen, damit im Anschluss daran mit Hilfe von MATLAB der Bildverarbeitungsalgorithmus zum Lösen der Aufgabe entwickelt werden konnte.

Nachdem die Simulation des Algorithmus zur Objekterkennung abgeschlossen war ging es im zweiten Schritt darum, das FPGA-Design auf Grundlage der Beispielprojekte zu entwickeln. Durch das Zusammenspiels zwischen Hardware und Software mussten dabei immer wieder die Fähigkeiten des DSPs bzw. des EDMA Kontrollers berücksichtigt werden.

Zusammen mit der Hardware wurde dann der entwickelte Bildverarbeitungsalgorithmus für eine Kamera auf dem DSP implementiert. Für das geforderte Echtzeitverhalten war es nötig, einige Routinen in Assembler zu entwickeln. Alle Funktionen, bei denen das komplette Bild eingelesen wird, wurden in Assembler programmiert, da diese am Aufwendigsten waren. Der erlangte Geschwindigkeitsvorteil trug dazu bei, dass das System in Echtzeit arbeitet.

Nachdem die Positionsbestimmung funktionierte, wurde zunächst die Synchronisation zwischen den DSPs entworfen, damit die Visualisierung vom anderen DSP übernommen werden konnte. Dabei wurde darauf geachtet, dass die beiden DSPs nicht in einen Deadlock gelangen und sich blockieren.

Zum Abschluss wurde noch die Übertragung der Ergebnisse zwischen den Plattformen implementiert. Es trat das Problem auf, dass diese nur einseitig möglich ist, da ansonsten durch Störungen die Kommunikation zusammenbricht.

Trotz dieser kleinen Schwierigkeiten wurde ein DSP-basiertes Stereokamerasystem zur Positionsbestimmung eines Objekts im Raum in Echtzeit entwickelt, dass durch weitere Entwicklungen auf einem Roboter eingesetzt werden kann.

A Inhalt des Datenträgers

Der Datenträger zu dieser Arbeit ist bei Herrn Prof. Dr.-Ing. Hans Peter Kölzer oder Herrn Prof. Dr.-Ing. Stephan Hußmann einzusehen.

Inhaltlich ist der Datenträger wie folgt gegliedert:

- **/01_pdf/**
  Enthält diese Arbeit im PDF Format.

- **/02_matlab/**
  Alle MATLAB-Skripte, die für diese Arbeit erstellt wurden, befinden sind in diesem Ordner.

- **/03_FPGA/**
  Die Projektdateien und Quelltexte zur Programmierung der FPGAs, sowie die Skript- und Testbench-Dateien für die Simulation sind hier zu finden.

- **/04_DSK/**
  In diesem Ordner sind die Softwareprojekte für das Code Composer Studio mit allen benötigten Quelltexten gespeichert. Darüber hinaus sind die Tabellen für das Softwarepipelining der Assemblerfunktionen enthalten.

Literaturverzeichnis

[1] HAW-HAMBURG: Projekt ELEKTRA. http://ieweb.etech.haw-hamburg.de/yamlt3/ELEKTRA.60.0.html, Abruf: 13. Januar 2010

[2] WIKIPEDIA: HSV-Farbraum. (3. JuLi 2006). http://de.wikipedia.org/wiki/HSV-Farbraum, Abruf: 13. Januar 2010

[3] TEXAS INSTRUMENTS: TMS320C6414T, TMS320C6415T, TMS320C6416T FIXED-POINT DIGITAL SIGNAL PROZESSORS - SPRS226M. (November 2003). http://focus.ti.com/lit/ds/symlink/tms320c6416t.pdf, Abruf: 18. Januar 2010

[4] TEXAS INSTRUMENTS: TMS320C64x Technical Overview - SPRU395B. (Januar 2001). http://focus.ti.com/lit/ug/spru395b/spru395b.pdf, Abruf: 18. Januar 2010

[5] TEXAS INSTRUMENTS: Using the DSP/BIOS Kernel in Real-Time DSP Applications - SPRA781. (August 2001). http://focus.ti.com.cn/cn/lit/an/spra781/spra781.pdf, Abruf: 21. Januar 2010

[6] SPECTRUM DIGITAL: TMS320C6416T DSK Technical Reference. (November 2004). http://c6000.spectrumdigital.com/dsk6416/V3/docs/dsk6416_TechRef.pdf, Abruf: 22. Januar 2010

[7] TEXAS INSTRUMENTS: TMS320C6000 DSP External Memory Interface (EMIF) - SPRU266E. (April 2008). http://www.ti.com/litv/pdf/spru266e, Abruf: 22. Januar 2010

[8] TEXAS INSTRUMENTS: TMS320C6000 DSP Multichannel Buffered Serial Port (McBSP) - SPRU580G. (Dezember 2006). http://focus.ti.com/lit/ug/spru580g/spru580g.pdf, Abruf: 22. Januar 2010

[9] BITEC LTD: DSK-EYE gigabit User manual Version 2.0. (2008)

[10] OMNIVISON: OV5620 Color CMOS QSXGA CameraChip™ Sensor with OmniPixel2™ Technology. (15 Februar 2007)

[11] ALTERA CORPORATION: Cyclone II Device Handbook, Volume 1. (Februar 2007). http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf, Abruf: 25. Januar 2010

[12] INFORMATION SCIENCES INSTITUTE UNIVERSITY OF SOUTHERN CALIFORNIA: RFC 791 - INTERNET PROTOCOL - DARPA INTERNET PROGRAM - PROTOCOL SPECIFICATION. (September 1981). http://www.ietf.org/rfc/rfc0791.txt, Abruf: 17. Juli 2008

[13] INFORMATION SCIENCES INSTITUTE UNIVERSITY OF SOUTHERN CALIFORNIA: RFC 793 - TRANSMISSION CONTROL PROTOCOL - DARPA INTERNET PROGRAM - PROTOCOL SPECIFICATION. (September 1981). http://www.ietf.org/rfc/rfc0793.txt, Abruf: 17. Juli 2008

[14] WIKIPEDIA: Transmission Control Protocol. http://de.wikipedia.org/wiki/Transmission_Control_Protocol, Abruf: 17. Juli 2008

[15] INFORMATION SCIENCES INSTITUTE UNIVERSITY OF SOUTHERN CALIFORNIA: RFC 768 - User Datagram Protocol. (28. August 1980). http://www.ietf.org/rfc/rfc0768.txt, Abruf: 17. Juli 2008

[16] BITEC LTD: DSKEye Schematics Rev 1.0. (06. September 2007)

[17] TEXAS INSTRUMENTS: TMS320C6000 McBSP Initialization - SPRA488C. (März 2004). http://www.ti.com/litv/pdf/spra488c, Abruf: 22. Januar 2010

[18] TEXAS INSTRUMENTS: TMS320C64x/C64x+ DSP - CPU and Instruction Set - SPRU732H. (Oktober 2008). http://focus.ti.com/lit/ug/spru732h/spru732h.pdf, Abruf: 08. Februar 2010

[19] CHASSAING, Rulph ; REAY, Donald: *Digital Signal Processing and Applications with the TMS320C6713 and TMS320C6416 DSK*. 2. Wiley-Intersience, 2008

[20] KLAUKIN, Christoph: *Entwicklung und Realisierung eines Systems für die automatische Spracherkennung mit einem erweiterten TI DSP Modul zur Robotersteuerung*. HAW-Hamburg, 2010

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit im Sinne der Prüfungsordnung nach §22(6) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Ort, Datum	Unterschrift