DIPLOMARBEIT

FPGA-BASIERTE ECHTZEIT-KORREKTUR VON ABBILDUNGSFEHLERN OPTISCHER SYSTEME

Thomas Liebeskind, 17550

1. Betreuer: Prof. Dr. Christian Siemers

2. Betreuer: Dipl.-Ing. Denis Grießbach

DLR Deutsches Zentrum für Luft- und Raumfahrt e.V. in der Helmholtz-Gemeinschaft

FACHHOCHSCHULE NORDHAUSEN University of Applied Sciences

EIDESSTATTLICHE ERKLÄRUNG

Hiermit erkläre und versichere ich, dass ich diese Diplomarbeit selbstständig und ohne fremde Hilfe verfasst habe. Es wurden keine anderen Hilfsmittel als die aufgeführten benutzt. Sämtliche verwendeten Quellen sind im Text oder im Anhang als solche ausgewiesen.

Thomas Liebeskind

Nordhausen, den 2. November 2009

INHALTSVERZEICHNIS

1 VORWORT

Die echtzeitnahe Verarbeitung von Bilddaten nimmt einen immer höher Stellenwert. Vor allem sicherheitsrelevante Anwendungen wie z.B. Straßenüberwachungen oder Kollisionskontrollen, aber auch Anwendungen im Entertainment-Bereich sind immer häufiger an Echtzeitbedingungen geknüpft. Da softwarebasierte Lösungen diese Bedingungen meist nur unzureichend erfüllen, geht der Trend schon seit einiger Zeit zu konfigurierbarer Hardware, insbesondere FPGAs. Die Umsetzung von bildverarbeitenden Algorithmen auf FPGAs unter Berücksichtigung von Echtzeitbedingungen rückt immer mehr in den Mittelpunkt.

1.1 KAPITELÜBERSICHT

*In **Kapitel 2** gibt einen Überblick über die Motivation, Problemstellung und Zielsetzung dieser Diplomarbeit.*

*Im **dritten Kapitel** werden die theoretischen Basiskenntnisse, die grundlegend für den weiteren Verlauf der Diplomarbeit sind, behandelt. Es wird eine detaillierte Übersicht über die zugrundeliegende Bildverarbeitung, die verwendete Hardwareumgebung und das FPGA-Betriebssystem gegeben.*

*In **Kapitel 4** werden das gesamte Hauptmodul und dessen Umsetzung beschrieben. Desweiteren wird näher auf die einzelnen Komponenten und ihre Funktionsweise eingegangen. Zum Schluss des Kapitels werden noch Aussagen über die Simulation und die Resultate der Synthese des Modules getroffen.*

*Dieser **fünfte Abschnitt** beleuchtet die durchgeführten Tests und das Echtzeitverhalten des Modules näher. Es werden einige Testwerkzeuge vorgestellt, die Ergebnisse der Tests interpretiert und ausgewertet und es geprüft, ob die Echtzeitbedingungen eingehalten wurden.*

*Dieses letzte **Kapitel 6** schließt die Diplomarbeit ab. Es werden die im Rahmen der Diplomarbeit gewonnen Ergebnisse erläutert, Anregungen gegeben und ein Ausblick über mögliche Weiterentwicklungen.*

2 EINLEITUNG

Dieses Kapitel gibt einen Überblick über die Motivation, Problemstellung und Zielsetzung dieser Diplomarbeit.

2.1 MOTIVATION

Die Idee zu dieser Diplomarbeit entwickelte sich aus einem Projekt, welches zurzeit am DLR[1] in Berlin bearbeitet wird. Dieses Projekt behandelt verschiedene Möglichkeiten, eindeutige Bildmerkmale in Bildsequenzen zu finden und zu verfolgen, um so eine Aussage über die Bewegung des optischen Systems und der Umgebung zu aufzustellen. Die Umsetzung der nötigen Algorithmen zum Finden und Verfolgen dieser Bildmerkmale, sog. Feature, werden dabei in Software umgesetzt. Dabei ergab sich die Notwendigkeit, ressourcenlastige Bildvorverarbeitungsalgorithmen hardwareseitig auszulagern, um die softwareseitigen Verarbeitung zu entlasten. Die Herausforderung, diese Algorithmen in Form eines Hardwaremoduls in einer Hardwareprogrammiersprache umzusetzen bot genau den richtigen Anreiz, diese Diplomarbeit zu schreiben.

2.2 PROBLEMSTELLUNG

Eines der Hauptprobleme, die beim Abarbeiten der Algorithmen zum Finden und Verfolgen von Merkmalen in Bildern auftreten, ist die Berücksichtigung und mögliche Umgehung der Abbildungsfehler des zugrundeliegenden optischen Systems. Am gravierendsten sind die geometrischen Verzerrungen, die sich durch die natürlichen Eigenschaften der Linse ergeben und sich in erster Linie als Kissen- und Tonnenverzeichnung manifestieren. Allerdings benötigt die softwareseitige Korrektur dieser Fehler zusätzliche Ressourcen und vor allem mehr Rechenzeit, die den gesamten Algorithmus weiter von dem Ziel der Echtzeitfähigkeit entfernt. Anstatt alle Pixel eines Quellbildes seriell abzuarbeiten, muss für jedes Pixel erst eine künstliche Verzeichnung durchgeführt werden, um die Position im verzeichneten Bild zu bestimmen und anschließend mit diesen Koordinaten wieder gearbeitet werden.

[1] Deutsches Zentrum für Luft- und Raumfahrt

So entstand die Notwendigkeit, die Korrektur der Abbildungsfehler aus dem Algorithmus zu extrahieren und diesem vorzuschalten. Das Problem, dass diese Diplomarbeit untersucht, ist die Möglichkeit einer hardwareseitigen Umsetzung eines Algorithmus, welcher die Abbildungsfehler eines optischen Systems korrigiert und die entzerrten Bilder in Echtzeit an andere Systeme weiter geben kann.

2.3 ZIELSETZUNG

Das Ziel dieser Arbeit ist es, die Umsetzbarkeit des Korrektur-Algorithmus in feldprogrammierbarer Hardware zu testen und seine Funktion unter Echtzeitbedingungen zu prüfen. Die Korrektur wird dabei in drei Teilaufgaben geteilt: die beginnende Entzerrung der Tonnen- oder Kissen-Verzeichnung mithilfe der Verzeichnungsparameter der Linse, die anschließende Rektifizierung [2] des Bildes mittels einer Transformationsmatrix und eine abschließende Bilineare Interpolation, um evtl. künstlich entstandene Alias-Effekte zu bereinigen. Sowohl die Verzeichnungsparameter als auch die Transformationsmatrix für die Rektifizierung sind von vornherein bekannt und können dem System als Parameter eingeführt werden. Bei den zu verarbeitenden Daten handelt es sich Graustufen-Bilder mit nur einem 8-Bit-Farbkanal. Um die Bedingung der Echtzeitfähigkeit zu erfüllen, sollen mindestens 25 Bilder pro Sekunde bei einer SVGA[3]-Bildauflösung von maximal 800x600 Pixel korrigiert und ausgegeben werden. Für die Verwendung in zukünftigen Projekten soll das Modul allerdings auch in der Lage sein, Bilder bis zu einer Auflösung von 2048x2048 Pixel zu verarbeiten.

[2] Eliminierung geometrischer Verzerrungen, siehe 2.1.4
[3] Super Video Graphics Array

3 GRUNDLAGEN

Das folgende Kapitel behandelt die theoretischen Basiskenntnisse, die grundlegend für den weiteren Verlauf der Diplomarbeit sind. Es wird eine detaillierte Übersicht über die zugrunde-liegende Bildverarbeitung, die verwendete Hardwareumgebung und das FPGA-Betriebssystem gegeben.

3.1 BILDVERARBEITUNG

3.1.1 ALLGEMEIN

Bildverarbeitung dient der Extraktion von Information aus den ursprünglichen Bilddaten. Im Gegensatz zur Bildbearbeitung werden die Bilddaten nicht manipuliert, sondern analysiert und ausgewertet. Gängige Verfahren der Bildverarbeitung sind die Bildsegmentierung, Mustererkennung und Bewegungsbestimmung. Das verarbeiten von Bilddaten findet in nahezu allen Wissenschaftsbereichen Anwendung. So werden zum Beispiel im Maschinenbau Bauteile gezählt und vermessen, in der Medizin Röntgen- und Ultraschallbilder zur besseren Deutbarkeit verarbeitet und bei der Qualitätssicherung digitale Abbilder des Zielproduktes auf Fehler untersucht.

3.1.2 ABBILDUNGSFEHLER

In optischen Systemen kann es aufgrund der Anordnungen und natürlichen Eigenschaften von Linsen, Spiegeln und Blenden zu Abweichungen von der idealen optischen Abbildung kommen. Diese sogenannten Abbildungsfehler wirken sich in erster Linie in Form eines verzerrten oder verwischten Bildes aus. Die kritischsten Fehler sind sphärische und chromatische Aberrationen[1].

[1] Abbildungsfehler (lat. *aberrare*, abschweifen)

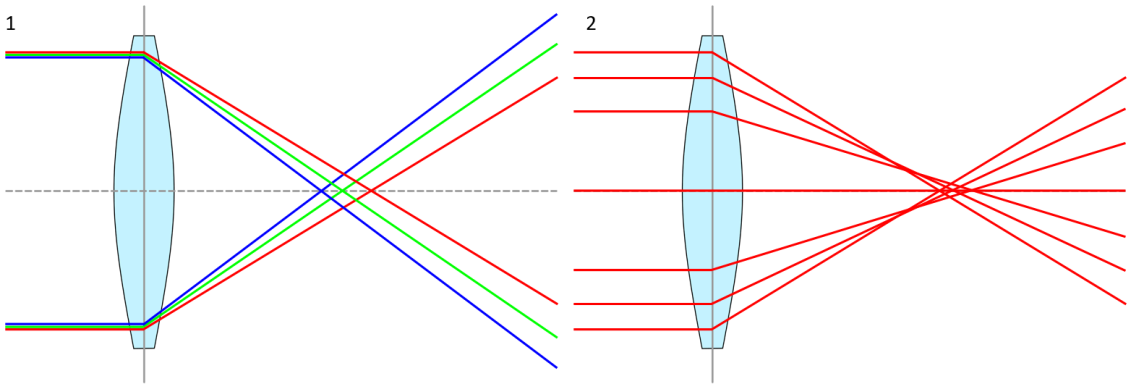


Abbildung 3.1: Chromatische (1) und sphärische (2) Aberration

Bei der chromatischen Aberration wird Licht unterschiedlicher Farbe und Wellenlänge an einer optischen Linse verschieden stark gebrochen, wodurch in der Abbildung an Hell-Dunkel-Übergängen rote und grüne Farbsäume entstehen und das Bild unscharf wirkt. Die sphärische Aberration hat zur Folge, dass achsenparallele und vom gleichen Objektpunkt auf der optischen Achse stammende Lichtstrahlen nicht in einem Punkt zusammenlaufen, da sie nach Durchgang durch das optische System nicht die gleiche Schnittweite haben.

3.1.3 Tonnen- und Kissenverzeichnung

Eine der Folgen einer sphärischen Aberration ist die Verzeichnung der Abbildung, wobei man hauptsächlich zwischen einer Tonnen- und einer Kissenverzeichnung unterscheidet. Solche Verzeichnungen entstehen, wenn der Abstand eines Bildpunktes von der Bildmitte nicht linear von der Höhe des jeweiligen Objektpunktes abhängt. Somit werden ursprünglich gerade Linien, dich nicht durch den Bildmittelpunkt abgebildet werden, gekrümmt dargestellt.

Durch eine Tonnenverzeichnung nimmt der Maßstab der Entfernung eines abgebildeten Punktest zum Mittelpunkt mit zunehmender Höhe ab, wodurch lineare Strukturen nach außen gewölbt werden. So werden zum Beispiel bei einem Quadrat, dessen Mittelpunkt gleich dem Bildmittelpunkt ist, alle Seiten nach außen gewölbt, was an eine Tonne erinnert. Der entgegengesetzte Effekt findet bei einer Kissenverzeichnung statt. Dabei nimmt der Abbildungsmaßstab mit zunehmender Höhe zu und es kommt zu einer Wölbung aller Linien nach in Richtung Bildmitte. So wird besagtes Quadrat, ähnlich einem Kissen, mit nach innen gewölbten Seiten abgebildet.

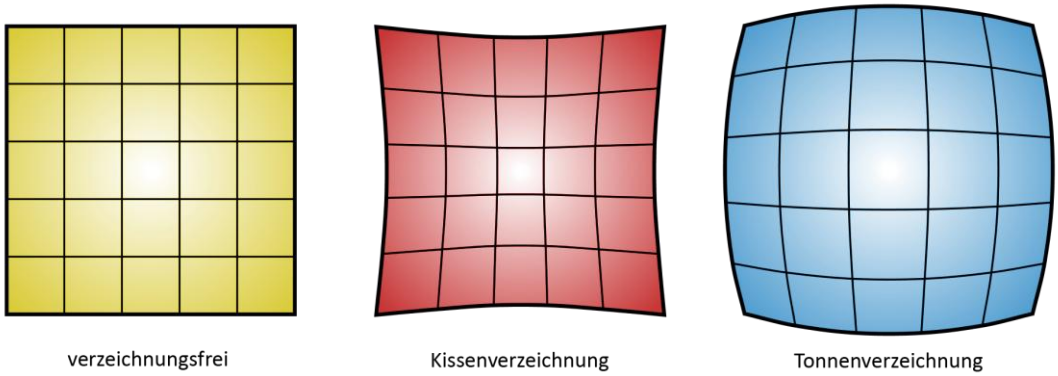


Abbildung 3.2: Tonnen- und Kissenverzeichnung

3.1.4 Bildvorverarbeitung

Das Ziel der Bildvorverarbeitung ist es, die Abbildungsfehler eines optischen Systems zu korrigieren, um darauffolgende Algorithmen zu entlasten. Im Rahmen dieser Diplomarbeit soll ein Eingangsbild von Kissen- oder Tonnenverzeichnungen befreit, eine anschließende Rektifizierung durchgeführt und abschließend eine Interpolation angewendet werden.

3.1.5 Verzeichnungskorrektur

Die Korrektur der Tonnen- oder Kissenverzeichnung, wie sie hier angewendet wird, ist im eigentlichen Sinne keine Entzeichnung. Viel mehr werden alle Pixel seriell durchlaufen und künstlich verzeichnet. Dadurch sind die Koordinaten des verzeichneten Original-Pixels bekannt und es kann mit diesem Pixelwert weiter gerechnet werden.

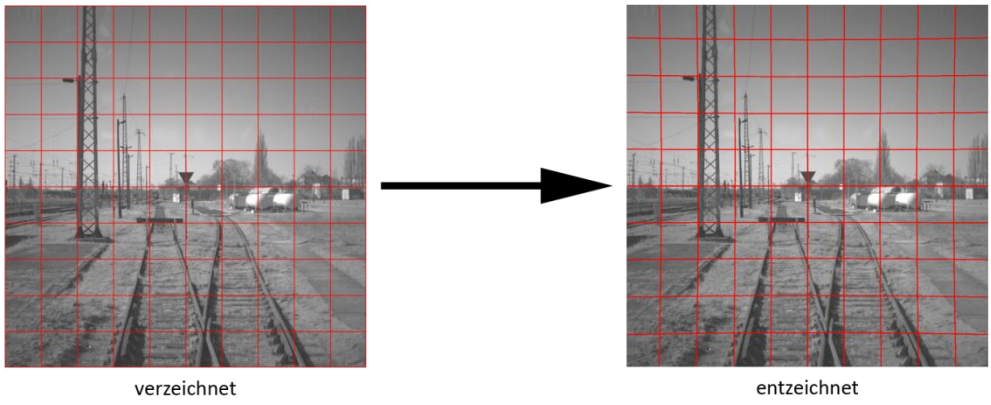


Abbildung 3.3: Beispiel einer Verzeichnungskorrektur

Der Verzeichnungs-Algorithmus bedient sich einiger Parameter, die zur Laufzeit bekannt sind und nicht dynamisch berechnet werden müssen. Diese Parameter sind die X- und Y-Koordinate des physikalischen Bildmittelpunkts (*x0* und *y0*), die Brennweite der Linse (*f*) und die Kameraparameter (*k1*, *k2* und *k3*). Die verzeichneten Koordinaten werden mit folgendem Algorithmus berechnet: Zuerst werden die zu verzeichnenden Koordinaten *x* und *y* um den physikalischen Bildmittelpunk verschoben:

$$u = x - x0 \text{ und } v = y - y0 \tag{3.1}$$

Anschließend wird die radiale Entfernung des Pixels zum Mittelpunkt berechnet:

$$r^2 = u'^2 + v'^2 \text{ mit } u' = \frac{u}{f} \text{ und } v' = \frac{v}{f} \tag{3.2}$$

Danach wird mittels der drei Kameraparameter *k1*, *k2* und *k3* die relative Verschiebung *Δ* des Pixels in radialer Richtung berechnet:

$$\Delta = \mathrm{r}^2 * (\mathrm{k1} + \mathrm{r}^2 * \mathrm{k2} + \mathrm{r}^4 * \mathrm{k3}) \tag{3.3}$$

Dann werden die Eingangskoordinaten *x* und *y* mit *Δ* multipliziert um die Verschiebung der Eingangskoordinaten in X- und Y-Richtung zu erhalten (*Δx* und *Δy*):

$$\Delta\mathrm{x} = \mathrm{x} * \Delta \text{ und } \Delta\mathrm{y} = \mathrm{y} * \Delta \tag{3.4}$$

Durch die abschließende Aufaddierung der Verschiebungen *Δx* und *Δy* auf die ursprünglichen Koordinaten ergeben sich dann die verzeichneten Koordinaten x‘ und y‘:

$$\mathrm{x}' = \mathrm{x} + \Delta \text{ und } \mathrm{y}' = \mathrm{y} + \Delta\mathrm{y} \tag{3.5}$$

Für die Umsetzung in Hardware wurde der Algorithmus allerdings etwas modifiziert. Um eine rechenintensive Division und somit gleichzeitig einige zusätzliche Rechenschritte auszuschließen, werden alle Divisionen durch *f* ausgelagert, sodass

$$r'^2 = u^2 + v^2 = r^2 * f^2 \tag{3.6}$$

und

$$\Delta = \mathrm{r}'^2 * (\mathrm{k1}' + \mathrm{r}'^2 * \mathrm{k2}' + \mathrm{r}^4 * \mathrm{k3}') \tag{3.7}$$

mit

$$\mathrm{k1}' = \frac{\mathrm{k1}}{\mathrm{f}^2}, \mathrm{k2}' = \frac{\mathrm{k2}}{\mathrm{f}^4} \text{ und } \mathrm{k3}' = \frac{\mathrm{k3}}{\mathrm{f}^6} \tag{3.8}$$

Als Konsequenz müssen nun die modifizierten Kameraparameter *k1‘*, *k2‘* und *k3‘* einmalig vorher berechnet und dem System als Parameter übergeben werden.

3.1.6 REKTIFIZIERUNG

Als Rektifizierung bezeichnet man die Eliminierung von Verzerrungen, die aufgrund von unebenem Gelände, einer zentralperspektivischen Aufnahme oder einer falschen Orientierung des Aufnahmesystems entstehen. Dabei werden die Koordinaten eines jeden Pixels mit einer 3x3 Transformationsmatrix *T* multipliziert und anschließend normiert.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \cdot T_{3x3} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}, \text{normiert: } \begin{pmatrix} x'/z' \\ y'/z' \\ 1 \end{pmatrix} \rightarrow x'' = \frac{x'}{z'} \text{ und } y'' = \frac{y'}{z'} \qquad (3.9)$$

Das ursprüngliche Pixel wird dann an die neu errechnete Koordinate *x‘‘* und *y‘‘* verschoben. Die Transformationsmatrix ist statisch und wird dem System als Parameter überliefert. In der Umsetzung des gesamten Korrektur-Algorithmus erfolgt die Rektifizierung nach der Verzeichnungskorrektur und knüpft nahtlos an diese an.

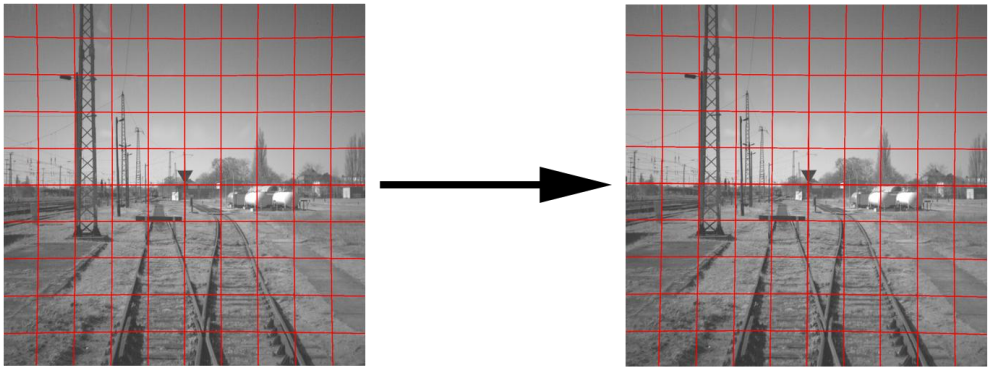


Abbildung 3.4: Beispiel einer Rektifizierung

3.1.7 INTERPOLATION

Sowohl durch die Verzeichnungskorrektur als auch durch die Rektifikation werden Pixel im Bild aus ihrem ursprünglichen Umfeld extrahiert und zu einem bestimmten Teil neu platziert. Dies geschieht vor allem an den stark verzeichneten Rändern des Bildes. Durch diese Neuorientierung entstehen künstliche Treppen-Effekte, die in späteren Algorithmen

zu Fehlinterpretationen führen können. So würde zum Beispiel ein Feature-Extraction[2]-Algorithmus wie der Kanade-Lucas-Tomasi Operator (KLT), welcher Pixel anhand ihrer Gradientenübergänge in horizontaler und vertikaler Richtung bewertet, solch eine Kante unter Umständen fälschlicherweise als interessantes Feature erkennen. Dies kann dazu führen, dass selbst in Bildsequenzen, bei dem sich das optische System oder die Umgebung bewegt, sehr oft die gleichen Bildpunkte inkorrekter weise als Kante erkannt werden, da die Parameter für die Entzeichnung und Rektifikation statisch sind und an gleichen Koordinaten immer das gleiche Ergebnis liefern. Dadurch würde ein anschließender Feature-Tracking[3]-Algorithmus diese oft auftretenden Kanten verfolgen und falsche Schlüsse über die Bewegung des optischen Systems ziehen.

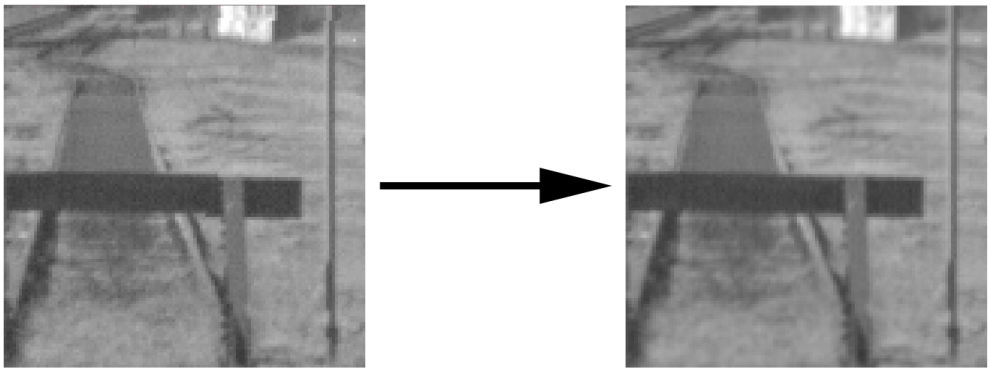


Abbildung 3.5: Beispiel einer bilinearen Interpolation

Es gibt mehrere Arten der Interpolation von Bildern, wobei im Folgenden nur auf die bilineare Interpolation eigegangen werden soll, da die im entwickelten Modul Anwendung findet.

Die neuen Koordinaten, die durch Verzeichnungskorrektur und Rektifikation errechnet werden sind, anders als die eingehenden Koordinaten, keine Ganzzahlen mehr sondern Gleitkommazahlen. So kann sich zum Beispiel das Pixel an Stelle [10; 10] im Ausgangsbild aus dem Pixel an Stelle [4,7345; 5,4253] im Originalbild ergeben. Da ein einfaches Abschneiden der Nachkommastellen zum Entstehen von künstlichen Kanten führen

[2] Finden eindeutiger Merkmale in Bildern
[3] Verfolgung eines Features über mehrere Bilder hinweg, um so eine Aussage über die Bewegung des Bildes zu finden

würde, müssen die Reste h_x und h_y mit berücksichtigt und zum Berechnen des neuen Farbwertes des Pixels verwendet werden:

$$C' = C_{0,0}(1-h_x)(1-h_y) + C_{0,1}h_x(1-h_y) + C_{0,0}(1-h_x)h_y + C_{0,0}h_xh_y \tag{3.10}$$

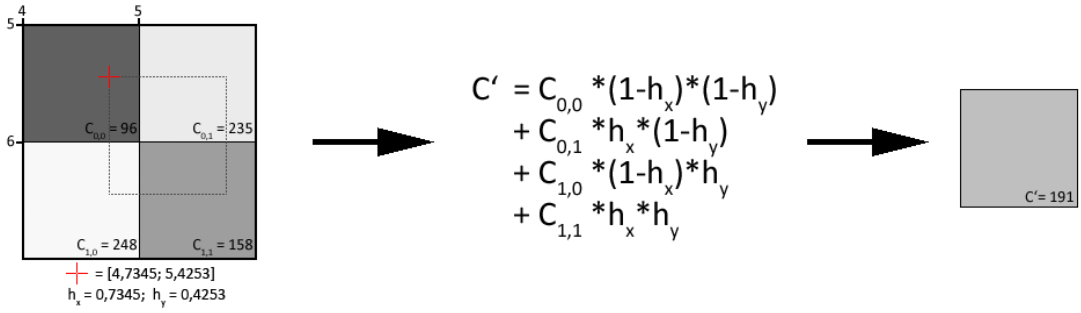


Abbildung 3.6: Rechenbeispiel für eine bilineare Interpolation

3.2 FPGA-BETRIEBSSYSTEM

3.2.1 KONZEPT UND FUNKTION

Das Ziel des FPGA-Betriebssystems ist die Unterstützung des Hardwareentwurfs, unabhängig von der eingesetzten konfigurierbaren Zielhardware. Die Wiederverwendbarkeit der Module des Betriebssystems erhöht die Zuverlässigkeit, verkürzt die Entwicklungszeit und verbessert die Nachnutzbarkeit von Entwürfen. Der größte Vorteil, der sich aus dieser Hardwareunabhängigkeit und Wiederverwendbarkeit ergibt, ist die Tatsache, dass ein VHDL-Programm unter Verwendung der vom Betriebssystem bereitgestellten Module und Bibliotheken nur einmal erstellt werden muss. Anschließend kann es ohne Änderung des Quelltextes für verschiedene FPGA-Karten kompiliert werden, vorausgesetzt diese werden von dem Betriebssystem unterstützt. Dadurch kann ein Modul, welches mithilfe des Betriebssystems erstellt wurde und wiederum auf den Betriebssystemmodulen aufbaut, ohne viel Aufwand für andere und zukünftige Projekte verwendet werden, die ebenfalls auf das FPGA-Betriebssystem aufbauen.

Zur Unterstützung des Entwicklungsprozesses existieren verschiedene Programme und Tools. So dient ein Strukturcompiler der Verbindung des Anwendungsprogrammes mit den Betriebssystemmodulen und der Erzeugung eines VHDL-Codes, da VHDL allein den Anforderungen des Konzeptes nicht gerecht wird. Das so erzeugte VHDL-Programm lässt sich durch gängige Entwicklungswerkzeuge simulieren und synthetisieren. Desweiteren stehen einige Programme, Treiber und Bibliotheken zur Verfügung, welche eine einfache Möglichkeit zur Kommunikation zwischen PC und FPGA-Karte bieten. Dies ermöglicht den Entwicklern ein anwendungsnahes und automatisierbares Testen der VHDL-Programme direkt am PC.

3.2.2 ANWENDUNG

Wie bereits erwähnt ist die Wiederverwendbarkeit der mit dem Betriebssystem erstellten Module einer der größten Vorteile dieses Konzeptes. Auch das im Rahmen dieser Diplomarbeit erstellte Modul soll unter diesen Gesichtspunkten erstellt werden. So

werden zur Ansteuerung und Kommunikation mit den Komponenten und Schnittstellen der FPGA-Karte wie nur vom Betriebssystem bereitgestellte Module verwendet. Dies betrifft in erster Linie den Datentransfer der FPGA-Karte mit externen Geräten, zum Beispiel über PCI[1]-Express (PCIe) oder Universal Serial Bus (USB) und zum anderen die Kommunikation mit Bausteinen, die sich auf dem FPGA-Board befinden, wie zum Beispiel dem DDR2-Speicher.

3.2.3 PROJEKTKONFIGURATION

Aus dem Quelltext des VHDL-Programms selber ist in der Regel nicht ersichtlich, welche Hardware im speziellen Einsatz finden. So wird z.B. für einen Speicherzugriff ein *memory*-Modul (siehe 2.2.5) und für den synchronen Datentransfer eine Pipe-Kommunikation (siehe 2.2.4) verwendet. Es ist nicht ersichtlich, ob es sich bei dem Speicher um DDR2-SDRAM[2], SSRAM[3] oder Block RAM und bei der Kommunikationsschnittstelle um PCI-Express, USB oder Ethernet handelt.

Erst in der Projektkonfiguration wird festgelegt, welche Betriebssystemschnittstellen auf welche realen Schnittstellen das FPGA verweisen. Dazu wird eine Konfigurationsdatei erstellt, die alle wichtigen Informationen über die Zielhardware, die Quellen und die Verwendung zusätzlicher Module enthält.

3.2.4 KOMMUNIKATION

Das FPGA-Betriebssystem definiert 3 Arten der Kommunikation:

- Asynchroner Transfer (Channel-Kommunikation)
- Synchroner Einzelwort-Transfer (SChannel-Kommunikation)
- Synchroner Block-Transfer (Pipe-Kommunikation)

Im Folgenden soll näher auf die Channel- und die Pipe-Kommunikation eingegangen werden, da diese beiden Kommunikationsarten im Korrektur-Modul eingesetzt werden.

[1] Peripheral Component Interconnect
[2] Double Data Rate Synchronous Dynamic Random Access Memory
[3] Synchronous Static Random Access Memory

Bei der Channel-Kommunikation gibt es nur ein Daten-Signal zwischen Sender und Empfänger mit einer vorher definierten Bitbreite. Dieses Signal kann asynchron geschrieben und gelesen werden, bietet aber keinerlei Kontrollmechanismen zur Steuerung und Kontrolle des Datenflusses. Die Abtastungsrate kann der Empfänger also frei bestimmen. So eignet sich diese Kommunikationsart vor allem für das Übermitteln von Parametern von Algorithmen, das sich diese Werte sehr selten ändern.

Die Pipe-Kommunikation hingegen eignet sich sehr gut zum Übertragen von Daten, die in einem logischen Zusammenhang stehen, wie es zum Beispiel bei Bildern der Fall ist. Die Daten werden in so genannten Frames zusammengepackt und bei Ausnutzung spezieller physikalischer Kommunikationsarten ist dadurch ein möglichst hoher Datendurchsatz garantiert. Mithilfe der zusätzlichen Kontrollsignale, die diese Kommunikationsart mit sich bring, kann sicher gestellt werden, dass keine Daten verloren gehen oder überflüssige Daten transferiert werden.

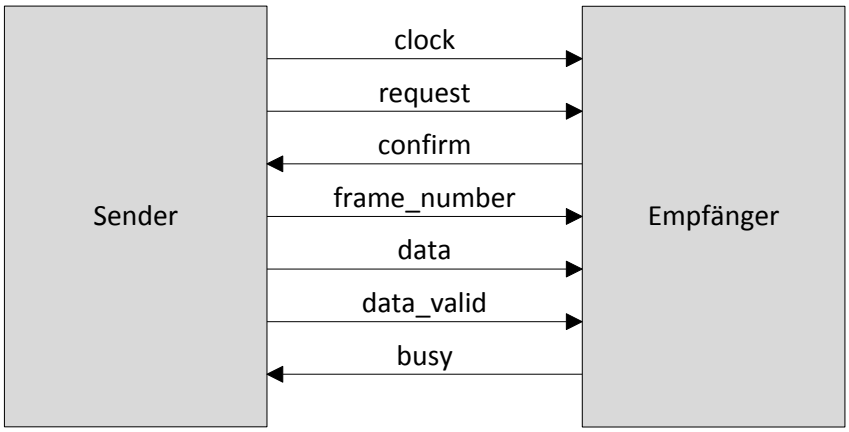


Abbildung 3.7: Signale einer Pipe-Kommunikation

Damit die einzelnen durch Pipes übertragenen Datenworte beim Empfänger wieder zu ihrem ursprünglichen Frame zusammengefasst werden können, wird immer eine 32-bit-Framenummer (*frame_number*) mit gesendet. Zur Abgrenzung eines Frames dienen die Signale *request* und *confirm* und nur wenn beide Signale *valid* sind, dürfen Daten gesendet werden, wohingegen das Signal *frame_number* nur geändert werden darf, wenn beide Signale *invalid* sind. Wenn der Sender eine Frame senden möchte, setzt er das Signal *request* auf *valid*. Sobald der Empfänger bereit ist, ein neues Frame zu empfangen, setzt er das Signal *confirm* auf *valid* und der Sender kann den Datentransfer beginnen. Nach Beendigung des Frames von Seiten des Senders setzt dieser das Signal *request* auf

invalid und sobald der Empfänger das Frame fertig verarbeitet hat, setzt dieser das Signal *confirm* auch auf *invalid*. Nun kann der Sender die Framenummer ändern und wieder anfragen, nach dem gleichen Schema ein neues Frame zu senden. Mit Hilfe des Signales *busy* kann der Empfänger den Transfer unterbrechen, der Sender darf dann nur noch ein Datenwort senden und muss mi dem Weitersenden auf Deaktivierung des *busy*-Signals warten. Das Signal *clock* dienst als Taktsignal und bildet den Bezug für alle Signale. In Abbildung 2.7 sind die einzelnen Signale Pipe-Kommunikation zu sehen.

3.2.5 SPEICHER-INTERFACE

Der Zugriff auf externen Speicher, als auf Speicherbausteine auf dem FPGA-Board, geschieht über das *memory*-Modul. Dieses Modul arbeitet unabhängig von dem tatsächlich eingesetzten Speichertyp. In Abbildung 2.8 sind alle Signale des *memory*-Moduls angegeben.

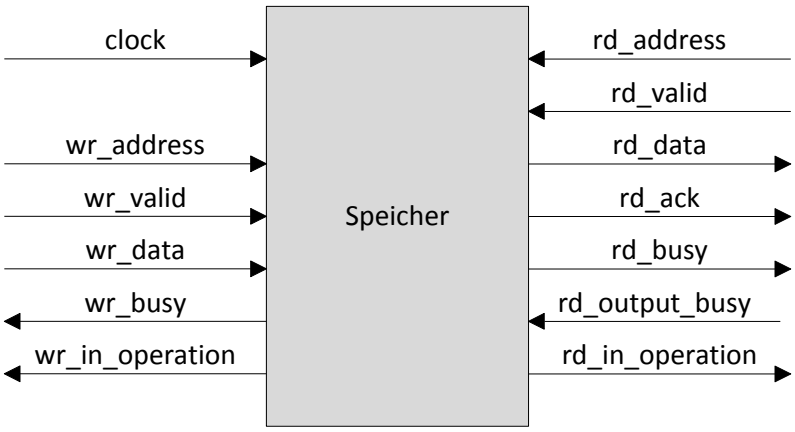


Abbildung 3.8: Signale einers *memory*-Moduls

Um ein solches *memory*-Modul zu nutzen müssen zunächst einige Parameter festgelegt werden. Dies sind die Datenbreite (*DATA_WIDTH*), die Adressweite (*ADDRESS_WIDTH*), die Anzahl der maximal zu speichernden Datenworte (*BUFFER_SIZE*) und die Lese- und Schreib-Prioritäten (*RD_PRIORITY*, *WR_PRIORITY*) für den Zugriff auf den physikalischen Speicher. Die Signale werden nach der Art des Speicherzugriffs unterschieden. Alle Signale, die das Präfix *wr* haben, dienen dem Schreibzugriff, alle mit *rd* als Prafix sind Signale für den lesenden Zugriff. Das Signal *clock* dient auch hier wieder als Taktsignal für alle anderen Signale, es repräsentiert aber nicht den Takt, mit dem der physikalische Speicher angesteuert wird.

Bei einem Schreibzugriff werden die an wr_*data* anliegenden Daten bei *valid* gesetztem wr_*valid* in den Speicher an Adresse wr_*address* geschrieben. Das Signal *wr_valid* zeigt dem Speicher an, ob die anliegenden Daten und die Adresse gültig sind und in den Speicher geschrieben werden können. *wr_valid* sollte nur auf valdid gesetzt werden, wenn *wr_busy* auf *invalid* gesetzt ist und der Speicher den Schreibvorgang auch ausführen kann, da sonst die Daten nicht verloren gehen. Das Signal *wr_in_operation* zeigt an, ob noch Schreiboperationen anstehen oder alle ausgeführt wurden.

Findet ein lesender Speicherzugriff statt, wird bei aktivem *rd_valid* eine Anfrage nach den Daten an Adresse *rd_address* gestartet. Sobald die Daten gelesen wurden, werden sie an das Signal *rd_data* angelegt und *rd_ack* auf *valid* gesetzt. Liegen keine validen Daten an, ist das Signal *rd_ack* auf *invalid* gesetzt. Das Signal *rd_in_operation* gibt auch hier wieder an, ob noch weitere Leseoperationen ausstehen oder alle abgearbeitet wurden. Sollte der Fall eintreten, dass gelesene Daten an *rd_data* anliegen, aber gerade nicht verarbeitet werden können, kann dies dem Speichermodul über das Signal *rd_output_busy* mitgeteilt werden.

Wie unter anderem an den Signalen *wr_in_operation* und *rd_in_operation* zu erkennen ist, kann das Speichermodul nach dem Pipeline-Prinzip angesteuert werden, was wiederum bei entsprechender Auslastung der physikalischen Leitung einen hohen Datendurchsatz erlaubt.

3.3 HARDWARE UMGEBUNG

3.3.1 XILINX VIRTEX-5 FPGA

Wie schon aus dem Titel dieser Diplomarbeit zu erkennen ist, soll der Korrektur-Algorithmus auf einem FPGA umgesetzt werden. Da am DLR auch schon für frühere Projekte auf Boards mit Xilinx-FPGAs der Virtex-Familie gesetzt werden, wurde auch für dieses Projekt ein solche FPGA verwendet. Konkret handel es sich um einen Virtex-5 SXT FPGA. FPGAs der SXT Serie sind optimiert für leistungsstarke Logik, Digitalsignalverarbeitung und speicherintensive Anwendungen mit serieller Konnektivität und gleichzeitig geringem Stromverbrauch. Der hier verwendete FPGA mit der Bezeichnung mit der Bezeichnung XC5VSX95T besitzt mehr als 14.000 Slices, über 94.000 logische Zellen und knapp 59.000 CLB[1] Flip-Flops. Desweiteren kann der FPGA auf insgesamt 8.784 Kbit Block RAM und maximal 1.520 Kbit Distributed RAM zugreifen. Zudem werden alle gängigen I/O-Standards unterstützt und außerdem sind bereits einige Hard IP-Cores[2], wie z.B. 4 10/100/1000 Ethernet MAC Blocks, 640 DSP48E Slices und einen PCI Express Endpoint Block. [Xil09]

Für die Umsetzung dieses Projekts, wie auch für andere Anwendungen im Bereich der Bild- und Signalverarbeitung, eignet sich dieser FPGA gut, da ausreichend Slices und Logic Cells für die komplexe Logik, genug Block RAM für zahlreiche FIFOs zur Verarbeitung der großen Datenmengen und ausreichend DSP [3] -Slices zur Auslagerung der Gleitkommaberechnungen.

3.3.2 FPGA-KARTE

Als FPGA-Karte wurde ein Xilinx Virtex LXT/SXT PCI Express Developer Board von Avnet eingesetzt (siehe Abbildung 2.9). Konkret handelt es sich um das Board mit der Bezeichnung AES-XLX-V5SXT-PCIE95-G welches wiederum auf den vorher beschriebenen Xilinx Virtex-5-FPGA setzt. Für den seriellen Datentransferstehe stehen verschiedene Schnittstellen zur Verfügung, wie z.B. RS-232, Gigabit Ethernet, PCI Express x8 und USB

[1] Configurable Logic Blocks
[2] Intellectual property core - enthält das geistige Eigentum des Entwicklers
[3] Digitaler Signalprozessor

2.0. Desweiteren sind 63 MP DDR2 SDRAM fest auf dem Board verbaut und es können weiter bis zu 256 MB über einen DDR2 SODIMM Sockel genutzt werden. Das Programmieren des FPGAs kann wahlweise über eine JTAG[4]-Schnittstelle oder die PCI-Express-Brücke vonstattengehen. [Avn09]

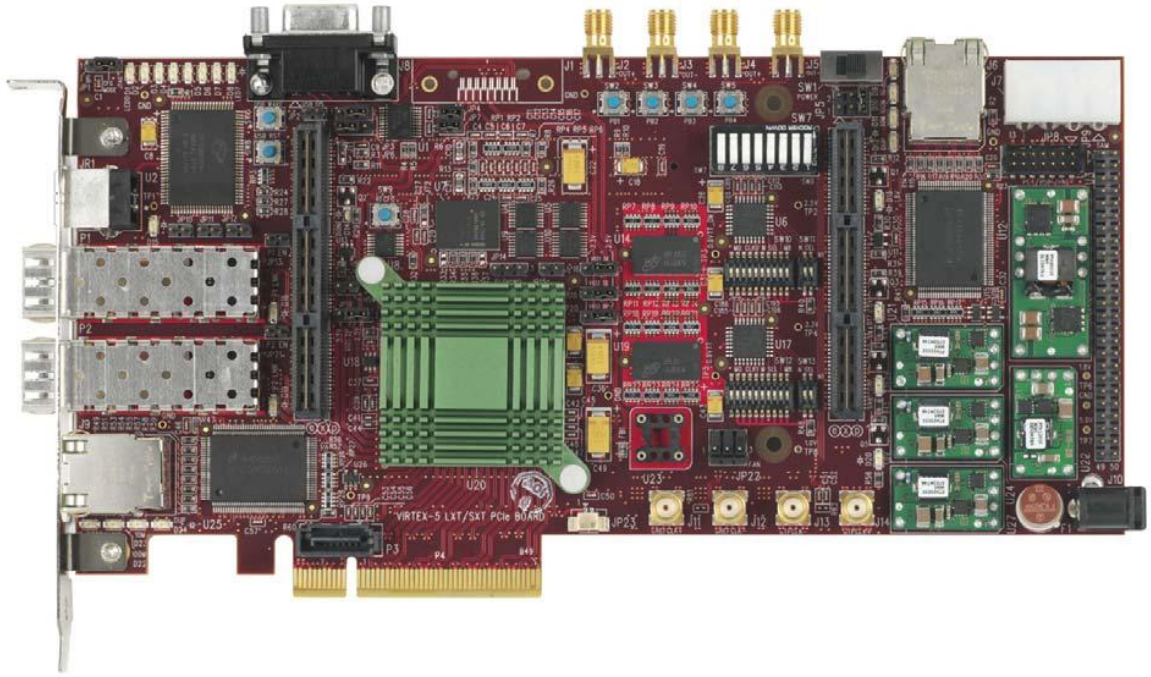


Abbildung 3.9: Avnet Xilinx Virtex-5 PCI Express Development Kit [Avn09]

[4] Joint Test Action Group, IEEE 1149.1

4 DAS HAUPTMODUL

In diesem Kapitel werden das gesamte Hauptmodul und dessen Umsetzung beschrieben. Desweiteren wird näher auf die einzelnen Komponenten und ihre Funktionsweise eingegangen. Zum Schluss des Kapitels werden noch Aussagen über die Simulation und die Resultate der Synthese des Modules getroffen.

4.1 GRUNDLEGENDES

4.1.1 KOMMUNIKATION

Im Abschnitt 2.2.4 wurden die verschiedenen Arten der Kommunikation vorgestellt, die das FPGA-Betriebssystem bereitstellt. Die Kommunikation im Modul, sowohl unter den Teilmodulen als auch zu den Schnittstellen des FPGA, geschieht in erster Linie über Pipes. Dadurch ist sichergestellt, dass keine Daten verloren gehen oder zusätzlich übertragen werden und außerdem die Datenrate hoch genug ist.

So geschieht auch die Eingabe der ursprünglichen Bilddaten und die Ausgabe der korrigierten Bilder über Pipes. Das Einlesen der Konfiguration des Moduls und der Parameter für die Algorithmen geschieht über zwei Channel-Kanäle, ein Kanal für den Konfigurations-Modus und einer für den entsprechenden Wert (siehe auch 3.2.2). Ansonsten finden neben der vom *memory*-Modul vorgegebenen Schnittstelle nur noch einfache Signalübertragungen oder selbstdefinierte, Pipe-ähnliche Kommunikationen statt.

4.1.2 BESONDERHEIT BEI DER INTERPOLATION

Um die Bilineare Interpolation zu beschleunigen, wird selbige nicht mittels 32-Bit-Gleitkommazahlen, wie es bei der Korrektur der Fall ist, berechnet. Stattdessen wird an die Koordinaten des zu interpolierenden Pixels ein nur 3 Bit langer Suffix angehängt, der die Verschiebung im Subpixelbereich repräsentiert. Um dies zu bewerkstelligen, wird das 32-Bit-Gleitkommazahl-Ergbenis, welches der Korrektur-Algorithmus für das aktuelle Pixel liefert, in das in Abbildung 3.1 gezeigte Format *fix_type* (siehe auch 3.1.3) umgewandelt.

Cn-1	Cn-2	Cn-3	Cn-4	…	C3	C2	C1	C0	R2	R1	R0
n-Bit-Ganzanteil-Koordinate									3-Bit-Rest		

Abbildung 4.1: Format *fix_type* der Interpolations-Koordinaten

Der Ganzzahlanteil der Koordinate dient der Generierung der Adresse zum Auslesen des korrigierten Pixels aus dem Speicher. Der 3 Bit lange Nachkommateil wird hingegen wird nur für die Interpolation benötigt. Durch ihn kann der Rest 8 Werte annehmen ([0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875]), womit sich für die Interpolation eines Pixels 64 Unterteilungen ergeben, was für weitere in Verwendungen völlig ausreichend ist. Um die Koordinate im 32-Bit-Gleitkommaformat *fp_type* in obiges Format umzuwandeln, wird due Funktion *to_fixed* genutzt (siehe Listing 3.1).

```
01 function to_fixed(z : fp_type) return fix_type is
02     variable temp : fix_type := (others => '0');
03     variable c : integer := MAXSIZE;
04 begin
05     if z(31) = '1' then
06         return temp;
07     end if;
08     if (z(30) /= '1') then
09         if (z(29 downto 23) = 127) then
10             temp(3) := '1';
11             temp(2 downto 0) := z(22 downto 20);
12             return temp;
13         elsif (unsigned(z(29 downto 25)) < 31) then
14             return temp;
15         elsif (unsigned(z(29 downto 25)) = 124) then
16             return temp+ 1;
17         elsif (unsigned(z(29 downto 25)) = 125) then
18             temp(2 downto 0) := z(24 downto 22);
19             return temp;
20         else
21             temp(2) := '1';
22             temp(1 downto 0) := z(22 downto 21);
23             return temp;
24         end if;
25     end if;
26     c := to_integer(unsigned(z(29 downto 23))) + 1;
27     temp(MAXSIZE+2 downto c+1+3) := (others => '0');
```

```
28      temp(c+3) := '1';
29      temp(c-1+3 downto 0) := z(22 downto 22-c+1-3);
30      return temp;
31  end;
```

Listing 4.1: Funktion *to_fixed*

Die übergebene Gleitkommazahl *z* wird anhand ihres Exponenten und der Mantisse umgewandelt. Da schon vor Aufruf dieser Funktion Koordinaten, die entweder negativ oder größer als die Breite/Höhe des Eingangsbildes sind, auf 0 bzw. Breite/Höhe gesetzt. Dadurch kann zum einen das Vorzeichen bei der Umwandlung ignoriert (Zeile 5) werden und zum anderen besteht nicht die Gefahr eines Überlaufs, da Breite und Höhe des Bildes nicht größer als $2^{MAXSIZE}$ sein dürfen (siehe auch 3.2.1).

4.1.3 Zusätzliche Datentypen

Um die Arbeit am Modul zu vereinfachen und übersichtlicher zu gestalten, wurden einige eigene Datentypen definiert im Packet *custom_pkg* zusammengefasst: *fp_type*, *fix_type*, *window_type*, *coord_type*, *fp_3x3_type* und *fp_pipe_type*. In Listing 3.2 sind die Definitionen der Datentypen zu finden (außer *fp_pipe_type*).

```
01  constant MAXSIZE : integer := 11; -- max. 2048 Pixel Breite/Höhe
02  constant PIXELWIDTH : integer := 8; -- 8 Bit pro Pixel (Graustufen)
03
04  subtype fp_type is std_logic_vector(31 downto 0);
05  subtype window_type is std_logic_vector(4*PIXELWIDTH-1 downto 0);
06  subtype coord_type is std_logic_vector(2*MAXSIZE+5 downto 0);
07  subtype fix_type is std_logic_vector(MAXSIZE+2 downto 0);
08  type fp_3x3_type is array (2 downto 0, 2 downto 0) of fp_type;
```

Listing 4.2: Deklaration der eigenen Datentypen

fp_type ist ein 32 Bit langer *std_logic_vector* und wird für 32-Bit-Gleitkommazahlen nach IEEE 754 genutzt, also 1 Bit für das Vorzeichen, 8 Bit für den Exponent und die restlichen 23 Bit für die Mantisse. *fix_type* ist, wie schon beschrieben, für die selbst definierten Festkommazahlen vorgesehen. Die Länge dieses Datentyps ist von der definierten Maximalgröße von Bildern abhängig und setzt sich aus dieser Maximalbreite/-höhe und den 3 Bit für die Nachkommastelle zusammen.

Der Datentyp *window_type* kann ein Bildfenster von 2x2 Pixel beinhalten, also 32 Bit bei eine Pixelbreite von 8 Bit. Dieser Datentype wird verwendet, um die zur Interpolation benötigten benachbarten Pixel eines zu interpolierenden Pixels zu liefern. Schon beim Einlesen des Bildes wird statt Pixel für Pixel immer ein solches Fenster in den Speicher geschrieben. So sind weniger Speicherzugriffe beim Lesen und Schreiben notwendig. Mithilfe von *coord_type* können die X- und die Y-Koordinate eines Pixels in einem Datentyp zusammengefasst werden. *coord_type* besteht also aus zwei aneinander gehängten Daten vom Type *fix_type*. Daraus ergibt sich für die Länge dieses Subtyps die doppelte Länge von *coord_type*.

Für die Rektifizierung ist eine Multiplikation eines Koordinaten-Vektors mit einer 3x3-Matrix nötig. Da in VHDL weder eine Gleitkomma-Matrix noch entsprechende Operationen zur Verfügung stehen, wird diese Operation wie auch der Entzeichnungs-Algorithmus manuell mit Hilfe der Gleitkomma-IP-Cores ausgeführt. Zu Gunsten der Übersichtlichkeit wurden die 9 skalaren Werte der Matrix in dem Datentyp *fp_3x3_type* zusammengefasst. Bei diesem Datentyp handelt es sich um ein zweidimensionales Array, sodass an die Algorithmen nur ein Signal vom Typ *fp_3x3_type* übergeben werden muss und die einzelnen Element bequem per Doppel-Index angesprochen werden können.

Bei *fp_pipe_type* handelt es sich um einen Datentyp, der die Ausführung der korrigierenden Algorithmen unterstützt. Da diese Algorithmen in eine Pipeline-Struktur abgearbeitet müssen und so zu jeden Takt ein neuer Eingangswert berechnet wird, müssen die Zwischenwerte durch jeden Schritt mit durchgeführt werden, da manche Werte, die am Anfang und im Laufe der Pipeline berechnet werden, auch am Ende der Pipeline verwendet werden. Die genau Definition dieses Datenverbundes in im Anhang A.2 zu finden. In Abschnitt 3.2.5 wird näher auf die einzelnen Elemente dieses Datentyps eingegangen.

4.1.4 Speicherbehandlung

Bei den zu verarbeitenden Daten kann es sich um Bilder mit einer Breite und Höhe von maximal bis zu 2048 Pixel handeln. Da der Algorithmus zum Beheben von Verzeichnungen nicht linear Pixel für Pixel abarbeiten kann, sondern einen größtenteils freien Zugriff auf die Pixel des Bildes benötigt, muss zumindest ein Teil des Bildes zwischengespeichert

werden. Das kann sich abhängig von der Verzeichnung und Verzerrung des Bildes um nur einige Zeilen, aber unter Umständen auch einen Großteil des Bildes handeln. Da vor allem ältere FPGAs, auf denen das Modul aber auch lauffähig sein soll, nicht genügend internen Speicher zur Verfügung haben, muss auf externen Speicher ausgewichen werden. Solch ein externer Speicherbaustein bringt mehr Speicher mit sich, ist aber auch langsamer.

Nach einigen Tests mit nur einem Speichermodul hat sich gezeigt, dass die Geschwindigkeit nicht hoch genug ist, um echtzeitnahe Anwendungen auszuführen, da Schreib- und Lesevorgänge nicht gleichzeitig ausgeführt werden können. Um dieses Problem zu umgehen wurden zwei Speicherbausteine eingesetzt. Dadurch wird QDR[1]-RAM simuliert und es kann gleichzeitig gelesen und geschrieben werden. Durch diese Maßnahme kann nun gleichzeitig ein Bild in Speicher 1 geschrieben und das Vorgängerbild aus Speicher 2 gelesen werden. So ergibt sich insgesamt nur einen zeitlichen Versatz, den der Speicher brauch, um ein Bild auszulesen. Da sich die beiden Speichermodule nun mit dem Schreiben und Lesen abwechseln sind einige Maßnahmen zur Koordination in den Modulen mit Speicherzugriff nötig, damit keine Fehler entstehen (siehe 3.2.4).

[1] Quad Data Rate

4.2 AUFBAU UND UMSETZUNG

4.2.1 HAUPTMODUL

Das Hauptmodul dient als Container für die Teilmodule und stellt alle ein- und ausgehenden Schnittstellen. Diese bestehen aus einer Input-Pipe und einer Output-Pipe, zwei asynchronen Input-Channels und einem Output-Channel (siehe Abbildung 3.2). Die beiden nicht synchronen Kanäle dienen der Konfiguration des Modules. Der erste Kanal ist 8 Bit breit gibt den Konfigurationsmodus an und der zweite Kanal, mit 32 Bit Breite, liefert den Konfigurationswert, der je nach Modus unterschiedlich interpretiert wird. Beide Kanäle werden unverändert an das Konfigurationsmodul weitergegeben. Durch die Input-Pipe wird das Eingangsbild über einen 8 Bit (1 Grauwert-Pixel) breiten Stream eingespeist und direkt an das Speicher-Schreibe-Modul durchgereicht. Nach der Korrektur wird das Ausgangsbild über die ebenfalls 8 Bit breiten Output-Pipe wieder ausgegeben. Eine Übersicht über den Aufbau des gesamten Hauptmodules ist im Anhang A.1 zu finden.

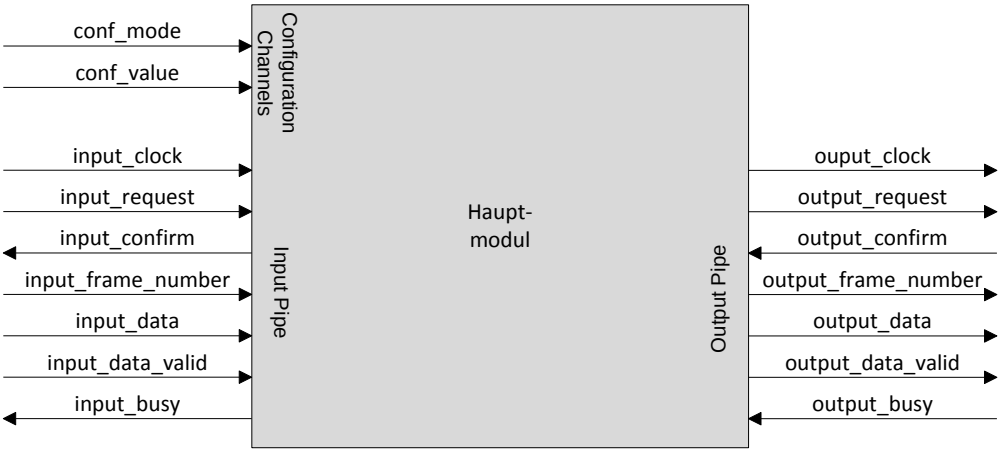


Abbildung 4.2: Kommunikationsschnitstellen des Hauptmoduls

Dieses Modul, auch Top-Modul genannt, beherbergt folgende Submodule: das Konfigurations-Modul (*config*), den Fenster-Generator (*window*), die Module für den Schreibzugriff (*memory_writer*) und den Lesezugriff (*memory_reader*) auf den Speicher, die beiden Speichermodule (*memory1* und *memory2*) und außerdem Entzeichnungs-Modul (*undistort*). Das Modul *config* liefert zum einen die Breite und Höhe des Eingangsbildes, welche vom Fenstergenerator, den beiden Speicherzugriffsmodulen und dem Entzeichnungsmodul verwendet wird, zum anderen die Parameter für die Korrektur-

Algorithmen, welche nur von dem Entzeichnungsmodul verwendet werden. Ansonsten werden neben einigen Flags und einem Adress-Signal für den Lesezugriff alle weiteren Kommunikationen unter den Modulen über Pipes gehandelt.

4.2.2 KONFIGURATION

Die Konfiguration des gesamten Moduls erfolgt die beiden asynchronen Konfigurationskanäle. Dazu werden die eingehenden Werte am Werte-Kanal entsprechend des angelegten Modus interpretiert. Tabelle 3.1 zeigt eine Auflistung aller Modi und die entsprechende Interpretation der Werte.

Modus	Werte-Typ	Interpretation
0x01	32bit Ganzzahl	Breite des Ein- und Ausgangsbildes (*sizeX*)
0x02	32bit Ganzzahl	Höhe des Ein- und Ausgangsbildes (*sizeY*)
0x04	32bit Floating-Point	X-Koordinate des physikalischen Bildmittelpunktes (*U0*)
0x05	32bit Floating-Point	Y-Koordinate des physikalischen Bildmittelpunktes (*V0*)
0x06	32bit Floating-Point	Normiertes K1 (*K1*)
0x07	32bit Floating-Point	Normiertes K2 (*K2*)
0x08	32bit Floating-Point	Normiertes K3 (*K3*)
0x0A	32bit Floating-Point	Rektifizierungs-Transformationsmatrix[0,0] (*T00*)
0x0B	32bit Floating-Point	Rektifizierungs-Transformationsmatrix[0,1] (*T01*)
0x0C	32bit Floating-Point	Rektifizierungs-Transformationsmatrix[0,2] (*T02*)
0x0D	32bit Floating-Point	Rektifizierungs-Transformationsmatrix[1,0] (*T10*)
0x0E	32bit Floating-Point	Rektifizierungs-Transformationsmatrix[1,1] (*T11*)
0x0F	32bit Floating-Point	Rektifizierungs-Transformationsmatrix[1,2] (*T12*)
0x10	32bit Floating-Point	Rektifizierungs-Transformationsmatrix[2,0] (*T20*)
0x11	32bit Floating-Point	Rektifizierungs-Transformationsmatrix[2,1] (*T21*)
0x12	32bit Floating-Point	Rektifizierungs-Transformationsmatrix[2,2] (*T22*)
0xFF	-	Beendigung der Konfiguration (*finished*)

Tabelle 4.1: Konfigurationsmodi

Das Ende der Konfiguration wird durch eine 255 (hex: 0xFF) als Modus eingeläutet und ist gleichzeitig das Signal für alle Teilmodule, die Arbeit aufzunehmen. Sobald im laufenden Betrieb ein Wert geändert wird, stoppt das Modul noch Beendigung des aktuellen Frames mit den alten Werten und arbeitet erst durch ein erneutes Anzeigen des Konfigurationsendes mit den neuen Parametern weiter.

4.2.3 Der Fenstergenerator

Um Speicherzugriffe einzusparen werden statt einzelner Pixel Bildausschnitte in den Speicher geschrieben. Bei diesen Ausschnitten handelt es sich um 2x2-Fenster. Da die Bilddaten seriell eingelesen werden, muss erst eine Zeile zwischengespeichert werden und bei Beginn der nächsten Zeile auf diese zugegriffen werden. Realisiert wird dieses Verfahren durch einen synchronen FIFO.

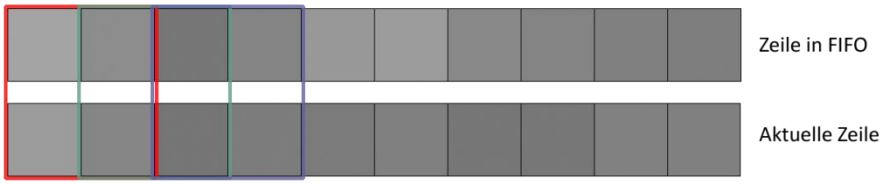


Abbildung 4.3: Fenster-Generator

Am Ende einer Zeile werden für die Pixel rechts oben ([0,1]) und recht unten ([1,1]) einfach die Pixelwerte links daneben wiederholt. Das gleich gilt für Pixel am Ende einer Spalte. So wird z.B. das letzte Pixel unten rechts im Bild durch ein Fenster repräsentier, das aus vier Pixel mit gleichem Wert besteht. Dadurch entstehen bei der Interpolation keine Fehler und es müssen keine Sonderbehandlungen für Randpixel durchgeführt werden.

4.2.4 Speicherzugriff

Wie schon beschrieben weißt der Speicherzugriff einige Besonderheiten auf. Es werden statt klassisch einem gleich zwei Speicherbausteine parallel genutzt. So ist garantiert, dass die Bilder ohne Unterbrechung in den Speicher eingelesen werden und gleichzeitig ausgelesen werden. Die Kontrollmechanismen, die festlegen, auf welchen Speicher zu welchem Zeitpunkt geschrieben und von welchem gelesen werden soll, arbeiten mithilfe von *Write-Enable-* und *Read-Enable-Flags.* Die Abfrage und das Setzen der Flags geschehen nur in und zwischen den beiden Modulen für Schreib- und Lesezugriff auf den Speiche. Bei Start der Anwendung wird das *Write-Enable-Flag* für beide Speichermodule auf *valid* gesetzt und die *Read-Enable-Flags invalid*, da noch keine Bilddaten im Speicher vorhanden sind. Bei eingehenden Daten wird in den ersten schreibbaren Speicherbaustein geschrieben und sobald das Bild komplett geschrieben wurde, wird das

Read-Enable-Flag des aktuellen Speichermodules auf *valid* und das *Write-Enable-Flag* auf *invalid*. Diese Prozedur wechselt sich ständig ab, sodass bei jedem eingehenden Bild der Speicher zum Lesen und Schreiben getauscht wird.

Durch diese Tick-Tack-Verfahren kann ein aktuell eingehendes Bild ohne Unterbrechung eingelesen werden, währen das vorherige gleichzeitig unbehindert ausgelesen werden kann. Dies erlaubt eine sehr schnelle Verarbeitung der Bilder und es entsteht praktisch kein zeitlicher Versatz zwischen dem Auslesen eines Bildes und dem Einlesen des nächsten Bildes. Eine vereinfachte Darstellung der Zustände beim Speicherzugriff ist in Abbildung 3.3 zu sehen.

Bei einem schreibenden Zugriff werden die Adressen intern im *memory_writer*-Modul hochgezählt und nach Ende eines Frames zurückgesetzt. Dies ist möglich, da die Daten aus dem Fenstergenerator seriell durch eine Pipe kommen und nicht extra zugeordnet werden müssen, sodass die Daten ebenfalls seriell in den Speicher geschrieben werden können. Das Schreibmodul kennt drei Zustände: *IDLE*, *OPERATING* und *DOWN*. Der Zustand *IDLE* kennzeichnet den Ruhezustand des Moduls, der immer dann eintritt, wenn ein Bild fertig in den Speicher geschrieben wurde. Dies ist auch der Reset-Zustand dieses Moduls. Ist das Modul im operierenden Zustand, werden Daten aus der entsprechenden Speicherstelle gelesen. Sollte der Zustand des Speichers auf *busy* wechseln, findet kein Zustandswechsel statt, stattdessen werden nur das automatische Hochzählen der Adresse und die Schreibanfrage unterbrochen und ebenfalls ein *busy*-Signal an den Fenstergenerator geschickt. Sobald der Speicher wieder bereit für weiter Instruktionen ist und dessen *busy*-Signal auf *invalid* wechselt, nimmt auch das Schreibmodul seine Arbeit wieder auf, ohne seinen Zustand zu ändern. Erst wenn der Fenstergenerator durch Setzen des *input_request*-Signals auf *invalid* anzeigt, dass der Frame vollständig übermittel wurde, wir in den Zustand *DOWN* gesprungen. In diesem Zustand wird das *Read-Enable-Flag* des aktuellen Speichers auf *valid* gesetzt, der Schreibe-Zeiger auf den anderen Speicher gewechselt und sobald dieser durch setzendes *Write-Enable-Flags* anzeigt, dass er schreibbereit ist, der nächste Frame eingelesen.

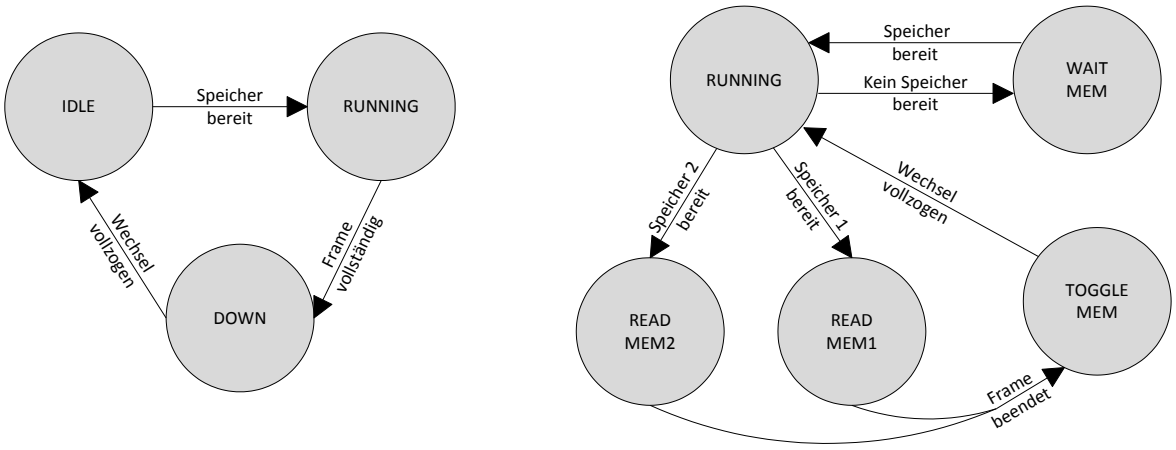


Abbildung 4.4: Vereinfachte Darstellt und der Zustände beim Speicherzugriff

Der Lesevorgang weist einige Unterschiede zum Schreibvorgang auf. Statt die Adresse intern hochzuzählen, wird sie durch das Entzeichnungsmodul übermittelt, da sich erst durch Ausführen der Korrekturalgorithmen die Koordinaten des gesuchten Pixels ergeben, aus denen wiederum die Speicheradresse generiert wird. Die Zustände dieses Moduls sind trotz anderer Nomenklatur ähnlich den Zuständen des Schreibmoduls, allerdings wurden sie um einen Wartezustand *WAIT_MEM* erweitert. Desweiteren wurden die Zustände für die tatsächlichen Lesezugriffe getrennt, sodass jeder Speicher seinen eigenen Lesezustand hat (*READ_MEM_1* und *READ_MEM_2*). Ansonsten findet auch hier eine Umschaltung zwischen den Speichermodulen statt, wenn ein Bild vollständig ausgelesen wurde. Dazu wird das *Write-Enable-Flag* des aktiven Speichers aktiviert und intern die Speicher-ID umgeschaltet.

4.2.5 KORREKTUR

Die Korrektur besteht aus zwei Teilaufgaben. Zuerst werden die durch Kissen- oder Tonnenverzeichnung entstandenen Bildfehler behoben und anschließen eine Rektifizierung durchgeführt. Dadurch werden die gravierendsten geometrischen Abbildungsfehler der Kamera bzw. der Linse berichtigt. Die Korretur findet nicht in dem Entzeichnungsmodul *undistort*, sondern wurde in ein extra Modul *correction* ausgelagert, welches von *undisort* genutzt wird.

Im Modul *correction* wird die für die Durchführung der beiden Algorithmen nötigen Schritte in eine Pipeline-Struktur ausgeführt. Diese Pipeline hat insgesamt 13 Schritte, 9 für die Entzeichnung und 4 für die Rektifikation. Debei werden in den meisten Schritten mehrere Rechenoperationen parallel ausgeführt, inswofern dies der Rechenweg erlaubt. Tabelle 3.2 zeigt die einzelnen Schritte und ihre Operationen entsprechend der unter 2.1.5 und 2.1.6 beschriebenen Allgorithmen auf.

Schritt	Operationen	Formel
1	`u = x0 - U0; v = y0 - V0`	2.1
2	`u2 = u * u; v2 = v * v`	2.2 bzw. 2.6
3	`r2 = u2 + v2`	2.2 bzw. 2.6
4	`r4 = r2 * r2; k2r2 = K2 * r2`	2.3 bzw. 2.7
5	`k3r4 = K3 * r4; k12 = K1 + k2r2`	2.3 bzw. 2.7
6	`k123 = k12 + k3r4`	2.3 bzw. 2.7
7	`tmp = k124 + r2`	2.3 bzw. 2.7
8	`t_x = u * tmp; t_y = v * tmp`	2.4
9	`r_x = x0 + t_x; r_y = y0 + t_y`	2.5

10	`rt00 = r_x * T[0,0]; rt01 = r_y * T[0,1];`
	`rt10 = r_x * T[1,0]; rt11 = r_y * T[1,1];`
	`rt20 = r_x * T[2,0]; rt21 = r_y * T[2,1]`
		2.9

11	`r0x = rt00 + rt01; r0y = rt10 + rt11;` `r0z = rt20 + rt21`	2.9
12	`x_r = r0x + T[0,2]; y_r = r0y + T[1,2];` `z_r = r0z + T[2,2];`	2.9
13	`x = x_r / z_r; y = y_r / z_r`	

Tabelle 4.2: Schritte der Pipeline

Bei allen kursiv markierten Signalen handelt es sich um die konstante Parameter, die durch das Konfigurationsmodul bereitgestellt werden. Signale, die durch mehrere Schritte in der Pipeline durchgereicht werden müssen, weil sie in späteren Schritten verwendet werden sind rot markiert. Die Bezeichnungen der Signale wurden so gewählt wie in der tatsächlichen Umsetzung der Algorithmen. Für jeden Schritt ist in der Spalte „Formel“ außerdem ein Verweis auf die entsprechende Formel aus den Abschnitten 2.1 und 2.1.6 gegeben.

Wie bereits erwähnt wurde für die Abarbeitung in der Pipeline-Struktur zur Steigerung der Übersichtlichkeit ein extra Datentyp definiert. In diesem Datenverbund werden neben dem eigentlichen skalaren Wert des Signals alle Signale mitgespeichert, die über mehrere

Schritte hinweg verwendet werden. So werden *u*, *v*, *x0*, *y0* und *r2* für jeden Schritt entsprechend mitgeliefert und durch die Pipeline durchgeführt.

Bei den eigentlichen Rechenoperationen handelt es sich um Gleitkommarechenoperation, welche durch 3 Gleitkomma-Recheneinheiten ausgeführt werden. Es gibt eine Additions-, eine Multiplikations- und eine Divisionseinheit. Bei diesen Einheiten handelt es sich um Xilinx IP-Cores, welche die DSP-Slices des FPGA nutzen können, um die Anzahl der verwendeten Logik-Gatter zu verringern. Diese IP-Cores wurden mit dem Xilinx CORE Generator™ erstellt als Module in das FPGA-Betriebssystem eingebunden. Für jede dieser Gleitkommarecheneinheiten kann eingestellt werden, wie viel Takte für die Berechnung verwendet (*Latency*) werden sollen und wie viel Zyklen mindestens zwischen den Eingaben erfolgen müssen (*Cycles per Operation*). Im Rahmen dieses Projekts werden jeden Takt neue Eingaben angelegt und nach 6 Takten die entsprechenden Ergebnisse erwartet, da so ein gutes Verhältnis zwischen Platzverbrauch und Performance erreicht wird. Durch die einheitliche Festlegung dieser Werte wird die Abarbeitung und Steuerung der Pipeline wesentlich einfacher, da keine Wartezyklen eingelegt werden müssen.

4.2.6 INTERPOLATION

Die Interpolation ist im eigentlichen Sinne kein eigenständiges Modul, sondern direkt in das Entzeichnungs-Modul integriert und wird ebenfalls über eine Pipeline ausgeführt. Die Interpolations-Pipeline hat 6 Schritte und orientiert sich an der in Abschnitt 2.1.7 angegebenen Interpolationsformel (2.10). Diese Schritte sind wie folgt aufgebaut:

1. - Trennen der Nachkommastellen von den Koordinaten: h_x (*h0*) und h_y (*w0*)
   - Berechnen von 1-h_x und 1-h_y (*h1 = „111“ xor h0, w1 = „111“ xor w0*)
   - Multiplikation der 4 Grauwerte mit 100 (*a_0*, *b_0*, *c_0*, *d_0*)
2. - Berechnen von (1-h_x)*(1-h_y), (1-h_x)*h_y, h_x*(1-h_y), h_x*h_y (*h1w1*, *h120*, *h0w1*, *h0w0*)
   - Division der 4 erweiterten Grauwerte durch 49 (*a_1*, *b_1*, *c_1*, *d_1*)
3. - Multiplikation der geteilten Grauwerte mit den entsprechenden Verschiebungen (*a_2 = a_1 * h1w1*, *b_2 = b_1 * h0w1*, *c_2 = c_1 * h1w0*, *d_2 = ad_1 * h0w0*)
4. - Addition der 4 Zwischenprodukte (res_0)
5. - Normierung des Ergebnisses durch Division durch 100 (res_1)

6. - Weiterleitung des Ergebnisses in Ausgabe-FIFO

Durch die Multiplikation mit 100 am Anfang der Pipeline und die abschließende Normierung brauchen zur Interpolation keine aufwändigen Gleitoperationen ausgeführt werden. Da es vorkommen kann, dass der Ausgang durch Setzen des *output_busy*-Signals auf *valid* anzeigt, dass keine weiteren Datenworte angenommen werden können, wird zwischen Interpolation und Ausgabe noch ein Ausgabe-FIFO zwischengeschaltet. So muss nicht bei jeder Unterbrechung die Pipeline unterbrochen werden und kann durchweg arbeiten. Nur bei längeren Unterbrechungen stoppt die Pipeline rechtzeitig, wenn der Ausgabe-FIFO Gefahr läuft, überzulaufen.

4.3 SIMULATION UND SYNTHESE

4.3.1 SIMULATION

Die Simulation des Projektes wurde in erster Linie mit dem Simulationsprogamm ModelSim [Xil09] von Mentor Graphics durchgeführt. Allerdings bietet das FPGA-Betriebssystem einige Werkzeuge, mit denen die Simulation stark erleichtert wird. So können ohne weiteres Bilddaten und andere Werte dem Simulationsprozess übergeben und auch wieder abgefangen werden. Dazu werden vor der Simulation entsprechende Kommando-Dateien erstell, welche dann während der Simulation ähnlich einer Stapelverarbeitung linear abgearbeitet werden. Die zugrundeliegende Verarbeitung der Kommandos nutzt als Kommunikationsschnittstellen des obersten Moduls. Es können Kommando-Datei für Channel-, SChannel- und Pipe-Kommunikation angelegt werden, sowohl für den In- als auch für den Output. Ein Beispiel für eine Kommando-Datei, die die gesamten Korrektur-Parameter über die beiden Konfigurationskanäle an das Hauptmodul übergibt findet sich im Anhang. Durch den Befehl *WR* kann z.B. einem bestimmten Kanal ein übergebener Wert zugewiesen werden. Um ein Bild im Rohdatenformat durch eine Pipe-Schnittstelle zu senden, wird der Befehl *WRRAW* verwendet, der unter Angabe des Pipe-ID und der Framenummer das Bild über die entsprechende Schnittstelle des Modules streamt. Mit dem Befehl SETRAW, dem Pipe-ID und Dateiendung übergeben werden, können ausgehende Daten auf die Festplatte geschrieben werden. Damit die Kommando-Dateien berücksichtigt werden, müssen sie einer bestimmten Nomenklatur entsprechen. *channel_input_command.txt* und *channel_output_command.txt* dienen der Kommunikation über asynchrone Kanäle, *pipe_input_command.txt* und *pipe_output_command.txt* werden abgearbeitet, wenn Daten über die Pipe-Schnittstelle transferiert werden sollen.

Durch die sehr realitätsnahe Umsetzung der simulierten Speichercontroller und der Taktbehandlung durch das Betriebssystem konnten schon während des Simulationsprozesses erste Vermutungen und Aussagen über die tatsächliche Geschwindigkeit getroffen werden. So hat sich schon am Anfang der Entwicklungsphase herausgestellt, dass die Nutzung von nur einem Speichermodul keine Geschwindigkeit im Rahmen der Echtzeitbedingungen zulässt und eine Alternative gefunden werden musste.

Desweiteren hat sich schon in diesem Stadium der deutliche Geschwindigkeitsvorteil einer Pipeline-artigen Abarbeitung der Algorithmen im Vergleich zu einer zustandsbezogenen Abarbeitung.

4.3.2 Vorbereitung zur Synthese

Bevor der Syntheseprozess gestartet werden konnte, mussten erst einige Vorbereitungen getroffen werden. Da das Korrektur-Modul zur Gleitkommaberechnung IP-Cores benutzt, mussten diese entsprechend der Zielhardware neu erstellt und als Module in das Betriebssystem integriert werden. Diese Module wurden so konfiguriert, dass die auf diesen Modulen aufbauenden Recheneinheiten automatisch als fixe, bereits als Netzlisten verfügbare Komponenten in den Synthesevorgang eingebunden werden.

Da als Testhardware das in Abschnitt 2.3 Beschriebene FPGA-Board verwendet wurde und zu diesem Zeitpunkt der PCI-Express Controller noch nicht in das FPGA-Betriebssystem integriert war, finden die Pipe- und die Channel-Kommunikation des Hauptmodules physikalisch über die USB-Schnittstelle des FPGA-Boards statt. Diese Tatsache wird in der entsprechenden Projekt-Datei, welche bei Start des Syntheseprozesses interpretiert wird, festgelegt. Desweiterer werden in dieser Datei Einstellungen zu den verwendeten Speicherbausteinen, den Takt-Quellen und die Verwendung weiterer Betriebssystemmodule wie z.B. FIFOs festgelegt.

4.3.3 Syntheseergebnisse

Nach erfolgreichem Abschluss des kompletten Synthesevorgangs hat sich folgende Zusammenfassung ergeben:

- Das gesamt Module verbrauchte ca. 24% der vorhandenen Slice-Register, wobei nahezu alle für Flip Flops verwendet wurden
- Insgesamt wurden ca. 40% aller Slices für die logische Verteilung verwendet
- 13% der DSP-Slices wurden verwendet
- Ca. 80% des FPGA-internen RAM werden von dem Modul belegt

Eine Analyse der Timings hat ergeben, dass der längste Datenpfad bei der Division bei der Interpolation mit ca. 22 Nanosekunden gegeben ist. Dadurch ergibt sich eine maximale theoretische Taktfrequenz von 45 Mhz. Für die verwendete FPGA-Karte wurde eine Taktfrequenz von 33 Mhz gewählt.

5 TESTS UND ECHTZEITVERHALTEN

Dieser kurze Abschnitt beleuchtet die Tests und das Echtzeitverhalten des Modules näher. Es werden einige Testwerkzeuge vorgestellt, die Ergebnisse der Test dargestellt und geprüft, ob die Echtzeitbedingungen eingehalten wurden.

5.1.1 TESTWERKZEUGE

Für den ausführlichen Test des Moduls war ursprünglich geplant, eine Kamera per CameraLink[1] an ein FPGA-Board anzuschließen und die von der Kamera gesendeten Bildsequenzen in Echtzeit zu verarbeiten. Allerdings verfügt die neue Zielhardware über keine CameraLin-Schnittstelle, sodass diese Möglichkeit nicht mehr zur Verfügung stand. Stattdessen wurden für die Tests kurze Bildsequenzen wiederholend über einen längeren Zeitraum mit Hilfe des Programms *PipeGui* in das Modul eingespeist. So wurde eine externe Kamera simuliert und es konnten alle relevanten Zeitverhalten gemessen und aufgenommen werden.

Zum Konfigurieren des Moduls, welches wie schon beschrieben über zwei Inpuut-Channels geschieht, wurde eine Batch-Datei erstellt, da mithilfe des Tools *fpgabatch* ausgeführt werden kann. Dadurch brauchten alle nötigen Einstellungen nur einmal angelegt werden und können schnell und einfach immer wieder aufgerufen werden und auf den FPGA übertragen werden.

Um den Output des FPGAs zu loggen wurde das Programm *fpgalogger* genutzt. Dieses Tool bietet die Möglichkeit, alle ausgehenden Kanäle und Pipes aufzuzeichnen um sie anschließen zur Auswertung bereitzustellen. Die Kanäle werden dabei jede Sekunde aufgenommen und alle Pipes in RAW-Dateien gestreamt.

5.1.2 AUSFÜHRUNG DER TESTS

Insgesamt wurden für die finalen Tests drei Testreihen durchgeführt. Zuerst wurde eine Bildsequenz mit einer Bildgröße von 800x600 Pixel, anschließend eine 1024x1024 Pixel große und zuletzt eine 2048x2048 Pixel große Bildsequenz durchlaufen. Nach Auswertung

[1] Schnittstelle für schnelle Bildübertragung

der ersten Ergebnisse wurde die anschließend die Auflösung berechnet, bei der das Modul an die Grenzen der Echtzeitbedingung stößt. Diese Auflösung beträgt ungefähr 724x724 Pixel und es wurde dafür noch eine zusätzliche Testreihe durchgeführt. Das Ergebnis der Korrektur und der Unterschied zwischen unkorrigiertem und korrigiertem Bild können im Anhang unter A3 eingesehen werden. Für die drei Testreihen ergaben sich folgende Geschwindigkeiten:

Testsequenz	Ø Bilder pro Sekunde	Ø ms pro Bild
800x600 Pixel	27,1	36,92
724x724 Pixel	24,9	40,1
1024x1024 Pixel	12,5	79,58
2048x2048 Pixel	3,8	259,23

Tabelle 5.1: Geschwindigkeiten der Testreihen

Bei allen drei Testreihen sind die einzelnen Messwerte nur sehr gering und nicht ausschlaggebend von den Durchschnittswerten abgewichen. Es gab keine auffällige Einbrüche oder unerwartete Geschwindigkeitszuwächse.

Um die Zeiten, die das Modul zum Verarbeiten eines Modules benötigt, zu ermitteln, wurden zwei Verfahren angewendet. Um eine erste Aussage über die Geschwindigkeit zu treffen, wurden die Differenzen der durch das Programm *fpgalogger* aufgezeichneten Zeitstempel für die einzelnen Frames ermittelt und umgerechnet. Für die Ermittlung genauerer Werte wurde in das Korrektur-Modul eine Funktion integriert, die den Zeitraum zwischen Ausgabestart und -ende eines Frames misst und über einen Debug-Kanal ausgibt. Die Zeitwerte dieses Kanals wurden ebenfalls mit *fpgalogger* aufgezeichnet und anschließend ausgewertet.

5.2 ECHTZEITVERHALTEN

5.2.1 ANSPRUCH

Damit das Modul dem Anspruch der Echtzeitfähigkeit gerecht wird, soll es mindestens 25 Bilder pro Sekunde bei einer SVGA-Auflösung von 800x600 Pixel pro Bild verarbeiten und ausgeben. Ausgelegt ist das Modul bis zu einer Bildgröße von 2048x2048 Pixel, sodass es auch in zukünftigen Projekten für entsprechend Große Bildsequenzen eingesetzt werden kann.

5.2.2 TATSÄCHLICHES GESCHWINDIGKEIT

Die Tests haben ergeben, dass das Modul den gegebenen Ansprüchen gerecht wird. Bei einer SVGA-Bildauflösung werden über 27 Bilder pro Sekunde verarbeitet und bei einer Auflösung von 724x724 Bildpunkten bewegt sich das Modul genau auf der Grenze der gegeben Echtzeitbedingung. Der Algorithmus ist auch in der Lage, Bilder bis zu einer Auflösung von 2048x2048 Pixel zu verarbeiten, allerdings nicht mehr im Rahmen der Echtzeitbedingungen, stattdessen werden nur knapp 4 Bilder pro Sekunde verarbeitet.

5.2.3 STELLUNGNAHME

Die Geschwindigkeit des Moduls liegt völlig im Rahmen der Bedingungen. Um jedoch auch Bildsequenzen mit einer Auflösung jenseits von SVGA in Echtzeit zu verarbeiten, ist der Algorithmus nicht schnell genug. Diese Tatsache ist auf zwei Hauptgründe zurückzuführen. Zum einen kann der DDR2-Speicher der Karte nicht sein ganzes Potential ausreizen, da Speichercontroller noch nicht optimal umgesetzt wurde. Desweiteren ist die Division während der Interpolation die Operation, die mit Abstand die meiste Zeit braucht und dadurch den maximal möglichen Systemtakt stark beschränkt. Durch eine Optimierung an dieser Stelle, wie z.B. eine Auslagerung dieser Division auf einen DSP oder die Umstellung der gesamten Interpolation auf Gleitkomma-IP-Cores, kann zu enormen Geschwindigkeitszuwächsen führen.

6 SCHLUSSBETRACHTUNG

Dieses Kapitel erläutert die im Rahmen der Diplomarbeit gewonnen Ergebnisse und gibt Anregungen und Vorschläge zur Verbesserung.

6.1 ERGEBNISSE

Diese Diplomarbeit hat gezeigt, dass die hardwareseitige Auslagerung von bildverarbeitenden Algorithmen sehr gut als wiederverwendbare Module umzusetzen ist. Mit Unterstützung es des FPGA-Betriebssystem konnte die Aufgabe, der sich diese Diplomarbeit gestellt hat, elegant gelöst werden. Das erstellte Modul lässt sich durch die einfach definierten Kommunikationsschnittstellen als Komponente in weiteren Hardware-Designs verwenden, aber auch als leistungsfähige Anwendung zur Unterstützung von Softwareprozessen einsetzen. Im Laufe der Diplomarbeit wurden folgende Punkte abgearbeitet:

- Umsetzung der grundlegenden Bildverarbeitungs-Algorithmen in VHDL,
- Integration dieser Algorithmen in ein von einem FPGA-Betriebssystem gestütztes, komplexes Modul,
- erlauben einer dynamischen Konfiguration des Moduls,
- entwickeln einer performanten Speicheranbindung,
- Verarbeitung von Bildern mit einer maximalen Auflösung von 2048x2048 Pixel erlauben,
- erfolgreiches Synthetisieren des Moduls und anschließendes
- ausführliches Testen auf
- Einhaltung der Echtzeitbedingungen.

6.2 AUSBLICK UND FAZIT

Das Ziel dieser Diplomarbeit war es, die Möglichkeiten zu untersuchen, Abbildungsfehler in optischen Systemen in Echtzeit auf Basis von FPGAs zu korrigieren. Im Laufe der Diplomarbeit haben sich weitere Schwerpunkte ergeben, wie z.B. die Auseinandersetzung

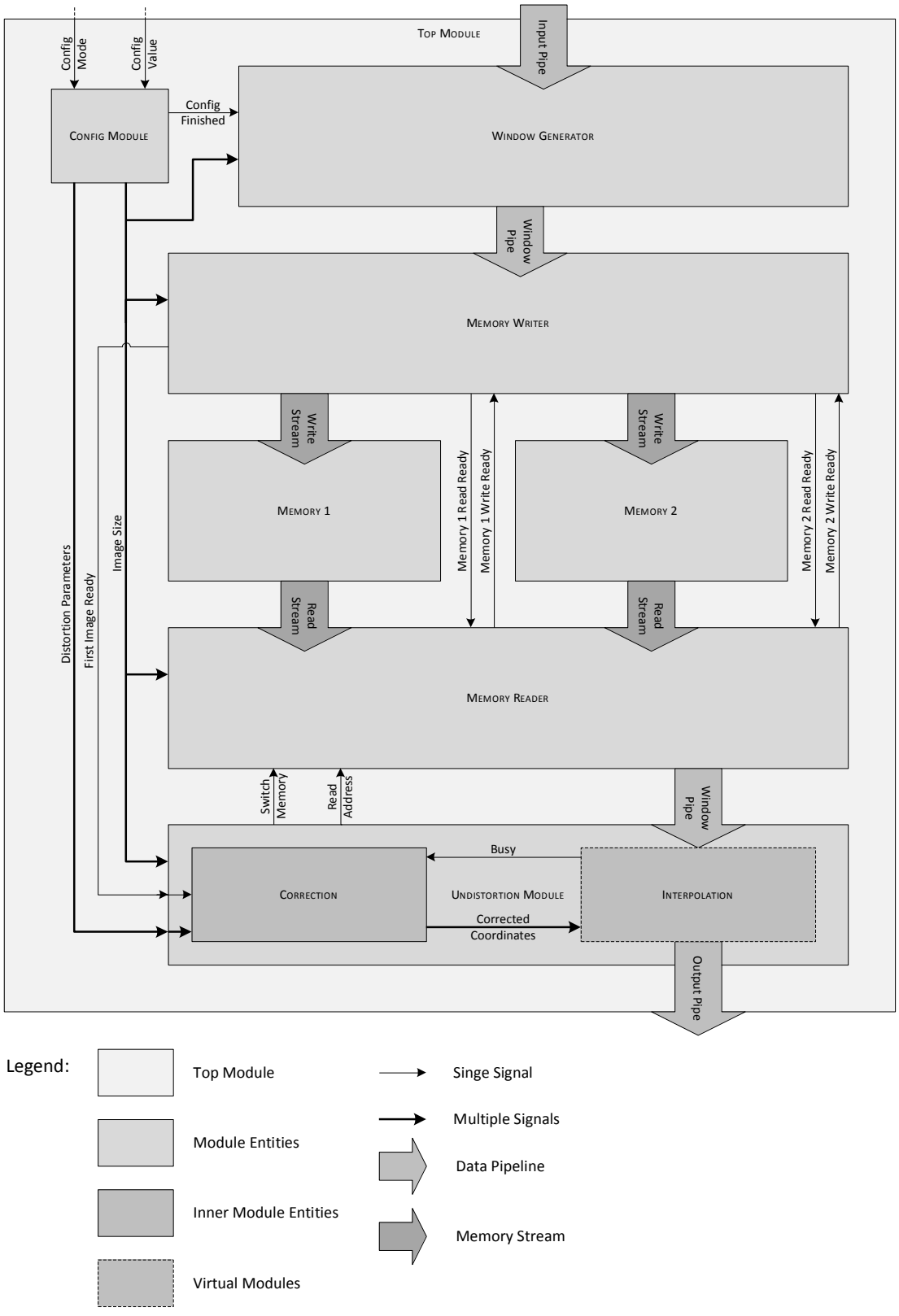
mit dem FPGA-Betriebssystem, die Umsetzung und Integration der Korrekturalgorithmen in ein Betriebssystemmodul und der möglichst effektive Umgang mit externem Speicher.

Zukünftigen Anwendungen von FPGAs im Bereich Bildverarbeitung steht nichts im Weg. FPGAs besitzen durch Ihre Fähigkeit, logische Befehle parallel abzuarbeiten enormes Potential. Da in Zukunft immer neue FPGAs entwickelt werden, bei denen sich auch der interne Speicher vergrößert, können Lösungen wie die in dieser Diplomarbeit Vorgestelle auf externen Speicher verzichten und durch Verwendung des FPGA-internen RAM eine spürbaren Geschwindigkeitsschub erreichen. Durch immer schnellere und höher taktbare FPGAs steigt auch die grundlegende Systemgeschwindigkeit und mit der Entwicklung neuer und schnellerer Kommunikationsarten können Bilddatenströme noch rasanter verarbeitet werden. Eine mögliche Erweiterung dieses Moduls ist die Unterstützung von Farbbildern. Zurzeit können von dem Modul nur Graustufenbilder verarbeitet werden. Allerdings kann der Algorithmus so modifiziert werden, dass auch Farbbilder verarbeitet werden können, indem jeder Farbkanal alle Stufen des Moduls durchläuft. Da die Farbkanäle voneinander unabhängig sind, können sie ebenfalls parallel verarbeitet werden, was also theoretisch zu keinem gravierenden Geschwindigkeitsverlust führen sollte.

Trotz anfänglicher Schwierigkeiten mit der Umsetzung der Algorithmen und der Speicherkommunikation wird nun diese Diplomarbeit und mit Ihr das Projekt erfolgreich abgeschlossen. Die neu gewonnen Kenntnisse und Erfahrungen bilden eine solide Grundlage für zukünftige Tätigkeit im Bereich der Hardwareentwicklung und der Erfolg des Projektes und die Erfüllung der gestellten Ziele bilden einen guten Abschluss dieser Diplomarbeit.

A Anhang

A.1 Blockdiagramm des Hauptmodules



A.2 LISTINGS

A.2.1 DEKLARATION DES DATENTYPES *FP_PIPE_TYPE*

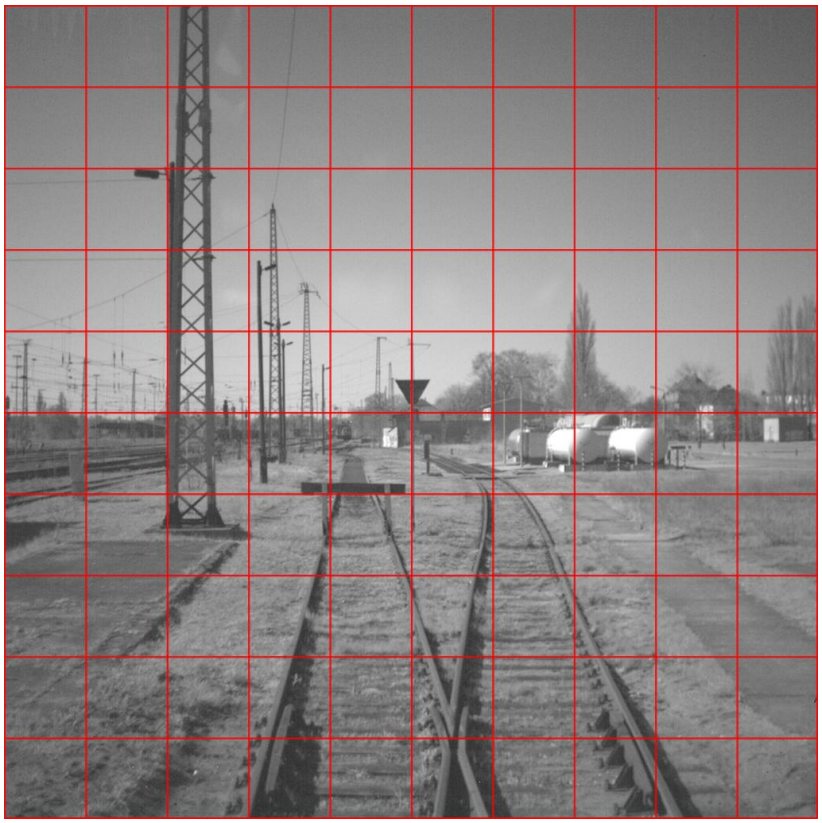
```
01 type fp_pipe_type is record
02     res       : fp_type;
03     x0        : fp_type;
04     y0        : fp_type;
05     x01       : fp_type;
06     y01       : fp_type;
07     x02       : fp_type;
08     y02       : fp_type;
09     x03       : fp_type;
10     y03       : fp_type;
11     x04       : fp_type;
12     y04       : fp_type;
13     x05       : fp_type;
14     y05       : fp_type;
15     u0        : fp_type;
16     v0        : fp_type;
17     u01       : fp_type;
18     v01       : fp_type;
19     u02       : fp_type;
20     v02       : fp_type;
21     u03       : fp_type;
```

```
21     v03       : fp_type;
22     u04       : fp_type;
23     v04       : fp_type;
24     u05       : fp_type;
25     v05       : fp_type;
26     r2        : fp_type;
27     r21       : fp_type;
28     r22       : fp_type;
29     r23       : fp_type;
30     r24       : fp_type;
31     r25       : fp_type;
32     en        : valid_type;
33     en1       : valid_type;
34     en2       : valid_type;
35     en3       : valid_type;
36     en4       : valid_type;
37     en5       : valid_type;
38     valid     : valid_type;
39     addr      : fp_type;
40 end record;
```

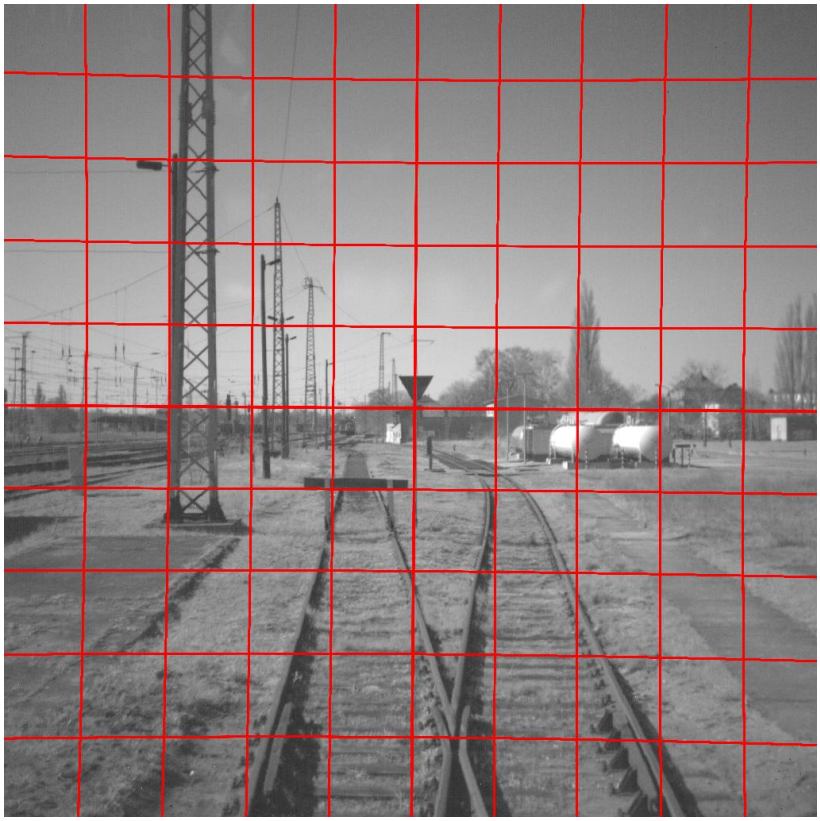
A.2.1 Beispiel einer Channel-Input-Kommandodatei

```
# sizeX = 724
WR 0001 01
WR 0002 000002D4
# sizeY = 724
WR 0001 02
WR 0002 000002D4

# params for undistortion
#U0 = 364.1
WR 0001 04
WR 0002 43B60CCD
#V0 = 349.5
WR 0001 05
WR 0002 43AEC000
#K1' = -7.526225E-8
WR 0001 06
WR 0002 B3A19FDC
#K2' = 3.7014947E-14
WR 0001 07
WR 0002 2926B354
#K3' = 0.0
WR 0001 08
WR 0002 00000000

# params for rectification
#T00 = 0.999946
WR 0001 0A
WR 0002 3F7FFC76
#T01 = 0.0052620866
WR 0001 0B
WR 0002 3BAC6D95
#T02 = 0.39999574
WR 0001 0C
WR 0002 3ECCCC3E
#T10 = -0.005607451
WR 0001 0D
WR 0002 BBB7BEB5
#T11 = 1.000307
WR 0001 0E
WR 0002 3F800A0F
#T12 = 5.573575
WR 0001 0F
WR 0002 40B25ABA
#T20 = -9.881853E-7
WR 0001 10
WR 0002 B584A1CA
#T21 = 8.277314E-25
WR 0001 11
WR 0002 178015D2
#T22 = 1.0003588
WR 0001 12
WR 0002 3F800BC2

#FIN
WR 0001 FF
```

A.3 BILDERVERGLEICH



Unkorrigiertes Bild



Korrigiertes Bild

LITERATURVERZEICHNIS

[Avn09] Avnet, Inc. Avnet Electronics Marketing - Xilinx® Virtex®-5 LXT/SXT PCI Express Development Kit. [Online] [Zitat vom: 2. Oktober 2009.] http://www.em.avnet.com/evk/home/0,1719,RID=0&CID=37133&CCD=USA&SID=32214&DID=DF2&LID=32232&BID=DF2&CTP=EVK,00.html.

[Kru06] Krutz, David. 2006. Ein Betriebssystem für konfigurierbare Hardware. [Online] 2006. http://deposit.d-nb.de/cgi-bin/dokserv?idn=983406014&dok_var=d1&dok_ext=pdf&filename=983406014.pdf.

[Men09] Mentor Graphics. ModelSim Asic and FPGA Design Simulator. [Online] http://www.mentor.com/products/fpga/simulation/modelsim.

[Xil09] Xilinx. Virtex-5 SXT FPGAs. [Online] Xilinx. http://www.xilinx.com/products/virtex5/sxt.htm.

ABBILDUNGSVERZEICHNIS

TABELLEN UND LISTINGS