Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Falko Peters

FPGA-basierte Bildverarbeitungspipeline zur
Fahrspurerkennung eines autonomen Fahrzeugs

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Falko Peters

FPGA-basierte Bildverarbeitungspipeline zur Fahrspurerkennung eines autonomen Fahrzeugs

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Technische Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Bernd Schwarz
Zweitgutachter: Prof. Dr. Franz Korf

Abgegeben am 30. April 2009

Falko Peters

Thema der Bachelorarbeit
FPGA-basierte Bildverarbeitungspipeline zur Fahrspurerkennung eines autonomen Fahrzeugs
Stichworte
FAUST, autonomes Fahrzeug, FPGA, System-on-Chip, Echtzeit, Fahrspurerkennung, Bildverarbeitung, Kalman-Filter, Nachbarschaftsoperation
Kurzzusammenfassung
Diese Arbeit behandelt die Entwicklung eines FPGA-basierten System-on-Chip zur Fahrspurerkennung für ein autonomes Fahrzeugs anhand von Echtzeit-Videodaten. Das System umfasst VHDL-Module, welche die Schnittstelle zu einer digitalen Videokamera bilden. Auf dem Videostrom werden Bildverarbeitungsoperationen durchgeführt, wodurch Messpunkte auf der Fahrspur extrahiert werden. Die Fahrspur wird als Polynom dritten Grades modelliert und ihr Verlauf mit der Zeit wird durch ein Kalman-Filter optimal abgeschätzt.

Falko Peters

Title of the paper
FPGA-based image processing-pipeline for lane detection of an autonomous vehicle
Keywords
FAUST, autonomous guided vehicle, FPGA, System-on-chip, real-time, lane detection, image processing, Kalman-filter, neighborhood operation
Abstract
This thesis describes the development of an FPGA-based system-on-chip for lane-detection of an autonomous vehicle, utilising real-time video-data. The system comprises VHDL-modules which form the interface to a digital video-camera. Image manipulation operations process the video-data, extracting measured points on the lane. The lane is modelled as a third order polynomial. It's pathway is optimally estimated by a Kalman-filter.

Inhaltsverzeichnis

1 Einleitung

Im Forschungsprojekt *Fahrerassistenz- und Autonome Systeme*, kurz *FAUST*, am Department für Informatik der HAW-Hamburg werden verteilte, eingebettete Echtzeitsysteme entwickelt. Verschiedene Fahrzeugplattformen werden im Rahmen dieses Projektes genutzt um autonome Systeme aufzubauen, welche beispielsweise Ausweichassistenten, Regelungsstrategien oder zeitgesteuerte Systeme erproben. Diese Systeme unterliegen einer kontinuierlichen Weiterentwicklung, wodurch die Anforderungen an den Leistungsumfang steigen, beziehungsweise völlig neue Anforderungen entstehen. Eine zentrale Bedeutung nimmt derzeit die Entwicklung von Systemen zur Navigation und Positionsbestimmung mit bildverarbeitenden Systemen ein. Zum einen werden dabei Lösungen zur Positionsbestimmung anhand von Land- oder Reflektormarken aufgebaut. Zum anderen wird das Ziel verfolgt, robuste Fahrspurerkennungssysteme zu entwickeln, welche dann die Grundlage der Navigationssysteme autonomer Fahrzeuge bilden (vgl. Manske (2008) und Jenning (2008)).

Gegenstand dieser Arbeit ist die Entwicklung eines FPGA-basierten System-on-Chip zur Fahrspurerkennung. Der Einsatz von einbetteten FPGA-Plattformen für Bildverarbeitungsaufgaben in autonomen Systemen ist ein aktuelles Forschungsthema. Es ist motiviert durch die Anforderungen der Industrie nach robusten und hochperformanten Echtzeitsystemen, die sich in großer Stückzahl kostengünstig herstellen lassen. FPGAs stehen dabei in Konkurrenz zu PC-basierten Systemen (Bonato u. a. (2007), Bagni u. a. (2008)).

Der größte Vorteil von FPGA-basierten Systemen ist gleichzeitig ein potentieller Nachteil: Nach Bedarf lassen sich beliebige Teile des Systems in Hardwarebeschleuniger auslagern. Parallelisierung und Pipelining von aufwändigen Berechnungen lassen FPGAs ihre geringere Taktrate ausgleichen, so dass häufig die Rechenleistung und der Datendurchsatz von PC-basierten Systemen übertroffen werden kann. Das Auslagern von Systemteilen in spezielle Hardwarekomponenten führt jedoch, wegen des geringen Abstraktionsgrads der RTL-Modellierung, zu einem signifikant erhöhten Entwicklungsaufwand gegenüber einer reinen Softwarelösung auf einem PC. Beim aktuellen Stand der Technik lässt sich keine allgemein gültige Aussage treffen, welches System vorzuziehen ist und diese Entscheidung muss von Fall zu Fall abgewägt werden.

Im Bereich der Entwicklungswerkzeuge für FPGA-basierte Systeme finden sich Ansätze, um die Abstraktionsebene der Entwicklung zu erhöhen: Einerseits durch Modellierungswerkzeuge und andererseits mittels Codegeneratoren. Ein aktuelles Thema ist etwa das generieren von RTL-Modellen aus MATLAB Code (vgl. Abschnitt 6). Derartige Werkzeuge vereinfachen die Entwicklung komplexer Systeme auf FPGAs und haben das Potential, den existierenden Vorsprung der Entwicklungswerkzeuge für reine Softwaresysteme zu verringern und somit die Attraktivität der FPGA-basierten Entwicklung zu erhöhen (Vanevenhoven (2007), Kirschke (2009)).

In dieser Arbeit wird im Kontext des Carolo-Cup Wettbewerbs ein Fahrspurerkennungssystem für ein autonomes Modellfahrzeug entwickelt. Die Systemanforderungen im Modellfahrzeug spiegeln die realen Anforderungen der Industrie, etwa nach Echtzeitfähigkeit und Robustheit, wieder. Im Kontext der Modellfahrzeuge ist der, auf Grund der niedrigen Taktraten, geringere Energieumsatz von FPGA-basierten Systemen ein besonderer Vorteil. Bei Modellfahrzeugen ist der Energieaufwand des Rechnersystems gegenüber demjenigen des Antriebssystems nicht mehr, wie bei einem PKW-basierten AGV, vernachlässigbar. Durch einen geringeren Energiebedarf des Rechnersystems ergibt sich dadurch eine deutliche Auswirkung auf die maximale Fahrzeit des Systems pro Batterieladung. Der geringere Energieumsatz hat auch den Wegfall eines aufwändigen Kühlsystems zur Folge, welches von modernen PCs benötigt wird, wodurch Platz auf der Fahrzeugplattform eingespart werden kann.

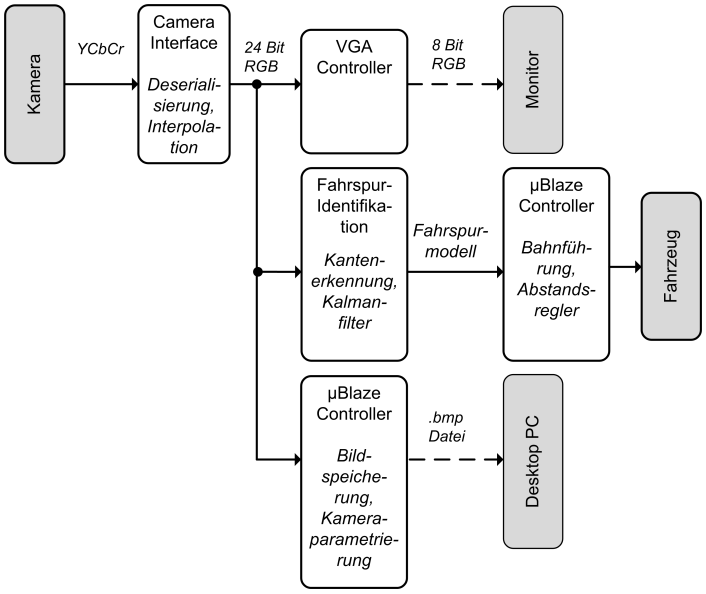

Abbildung 1.1: Systemübersicht

Die folgenden Infrastrukturkomponenten für ein FPGA-basiertes Fahrspurerkennungssystem wurden entwickelt und werden im Verlauf dieser Arbeit dargestellt (vgl. Abbildung 1.1):

- Das Hardwaresystem, bestehend aus der *Sony FCB-PV10* Farbbildkamera sowie dem FPGA-Entwicklungsboard *Digilent Nexys2* mit Xilinx *Spartan3E* FPGA.
- Das systemintern verwendete 24 Bit RGB Videodatenformat samt geeigneter Schnittstellen
  - zum Dateneingang, dem *ITU-R BT.656* kompatiblen interlaced-Videostrom der Kamera und

  - zum VGA-Port des *Nexys2* Entwickungsboards und damit zu einem Computermonitor. Dieser bietet, durch das unmittelbare visuelle Feedback, eine Unterstützung in der (Weiter-) Entwicklungsphase des Systems.
- Das Microblaze Prozessorsystem zur Übertragung von Steuerparametern an die Sony Kamera während der Systemlaufzeit.
- Ein Hardware/Software Systems zum Speichern einzelner Frames aus dem Videostrom als Bilddatei auf einem PC während der Systemlaufzeit. Dieses Subsystem dient in erster Linie der Dokumentation des Systems. Ein Arbeitsschritt des Subsystems besteht darin, die Bilddaten eines Frames im Hauptspeicher des Microblaze Prozessorsystems abzulegen. Damit bietet es die zusätzliche Möglichkeit, Teile des Bildverarbeitungssystems in Softwaremodule auszulagern (vgl. Abschnitt 6.6).
- Eine generischen Hardwarekomponente zur Implementierung von Nachbarschaftsoperationen im Videodatenstrom. Als Beispiel sei hier die Kantenfilterung mit dem Sobeloperator genannt.

Darüber hinaus stellt diese Arbeit ein Konzept für ein Kalman-Filter zur robusten dynamischen Fahrspurverfolgung vor. Im Gegensatz zu statischen Verfahren beziehen dynamische Verfahren den zeitlichen Verlauf der Fahrspur in die Berechnung der Parameter des Fahrspurmodells für den jeweiligen Zeitpunkt mit ein. Dies ermöglicht ein effektives Ausgleichen von Messungenauigkeiten und hat damit eine geringere Störempfindlichkeit als statische Verfahren zur Folge. Die Genauigkeit einer einfachen Tiefpassfilterung der Modellparameter wird vom Kalman-Filter übertroffen, da dieses die stochastischen Eigenschaften des Systems berücksichtigt (Jenning, 2008).

Nachfolgend wird der Aufbau dieser Arbeit dargestellt:

Abschnitt 2 (ab Seite 6) gibt einen Überblick über das System und stellt die Konzepte und die Interaktion der entwickelten Hardware- und Softwaremodule dar.

Die Komponenten der Videopipeline sind in **Abschnitt 3** (ab Seite 15) dokumentiert. Der Konvertierungsprozess, welcher den ITU-R BT.656 Interlaced-Videostrom (vgl. ITU656, 1998) der Kamera in das systemintern verwendete 24 Bit RGB Videoformat umwandelt, wird dargestellt. Zur Ausgabe des RGB-Videostromes auf einem VGA Bildschirm wird im 60Hz Interlaced-Videostrom eine Zeilenverdopplung durchgeführt, wodurch er als Progressive Videostrom interpretiert werden kann. Für diesen Videostrom werden Synchronisationssignale erzeugt, welche den Monitor ansteuern.

Abschnitt 4 (ab Seite 30) stellt zunächst die Komponenten des Microblaze Prozessorsystems vor. Zur Parametrierung der Sony-Kamera mit Einstellungen zu beispielsweise Kontrast und Brennweite wurden Softwarekomponenten entwickelt. Mit ihnen kann die Kamera über eine serielle UART-Schnittstelle im laufenden System konfiguriert werden.

Zum Zweck der Dokumentation des Systems wurden Hardware- und Softwaremodule entwickelt, die einen einzelnen Frame aus dem laufenden Videostrom speichern und als

Bitmap-Datenstrom über eine serielle Schnittstelle an einen angeschlossenen PC senden. Dort wird der Frame als Bitmap Datei gespeichert. Sie sind in **Abschnitt 5** (ab Seite 33) dokumentiert.

In **Abschnitt 6** (ab Seite 38) wird ein Konzept zur Fahrspurerkennung vorgestellt. Ein generisches Filtermodul für Nachbarschaftsoperationen wird vorgestellt und seine Arbeitsweise beispielhaft anhand einer Sobel-Filtermaske erläutert. Zum robusten Tracking der Fahrspur wird ein geeignetes Kalman-Filter vorgeschlagen, welches zur Optimierung die Randbedingungen des Carolo-Cups berücksichtigt. Abschließend wird ein Ausblick auf mögliche Schritte zur Weiterentwicklung des Fahrspurerkennungskonzeptes gegeben.

Die Inhalte und Ergebnisse dieser Arbeit werden in **Abschnitt 7** (ab Seite 54) zusammengefasst.

2 Konzeptionelle Systemübersicht

Dieser Abschnitt liefert einen Überblick über das Gesamtsystem. Die Aufgaben und Funktionsweisen aller beteiligter Komponenten werden erläutert.

2.1 Hardwareplattform

Als Hardwareplattform wird das *Nexys 2* FPGA-Entwicklungsboard der Firma Digilent. Auf ihm stehen die folgenden Komponenten zur Verfügung (vgl. (Digilent, 2008) sowie Abbildung 2.1):

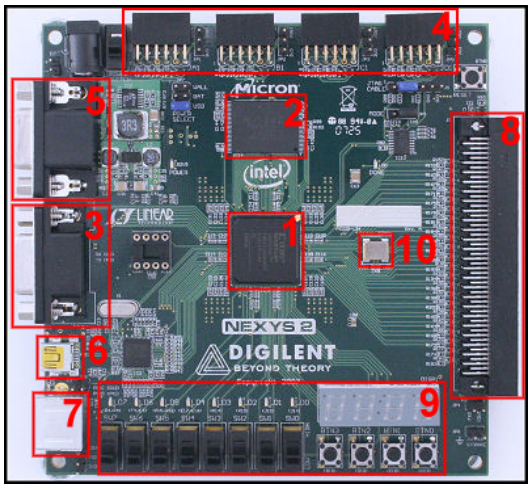

Abbildung 2.1: Komponenten auf dem Digilent *Nexys 2* Board

1 Das Xilinx Spartan3E XC3S1200 FPGA

2 16 MB Micron SRAM

3 Serielle RS232 Schnittstelle

4 4 Pmod Schnittstellen (jeweils 8 Daten-, 2 Masse- sowie 2 Power-Pins),

5 VGA Schnittstelle

6 USB2 Schnittstelle

7 PS2 Schnittstelle

8 FX-2 Konnektor

9 8 LEDs, 8 Schiebeschalter, 4 Drucktaster sowie eine vierstellige Siebensegmentanzeige

10 50 MHz Oszillator

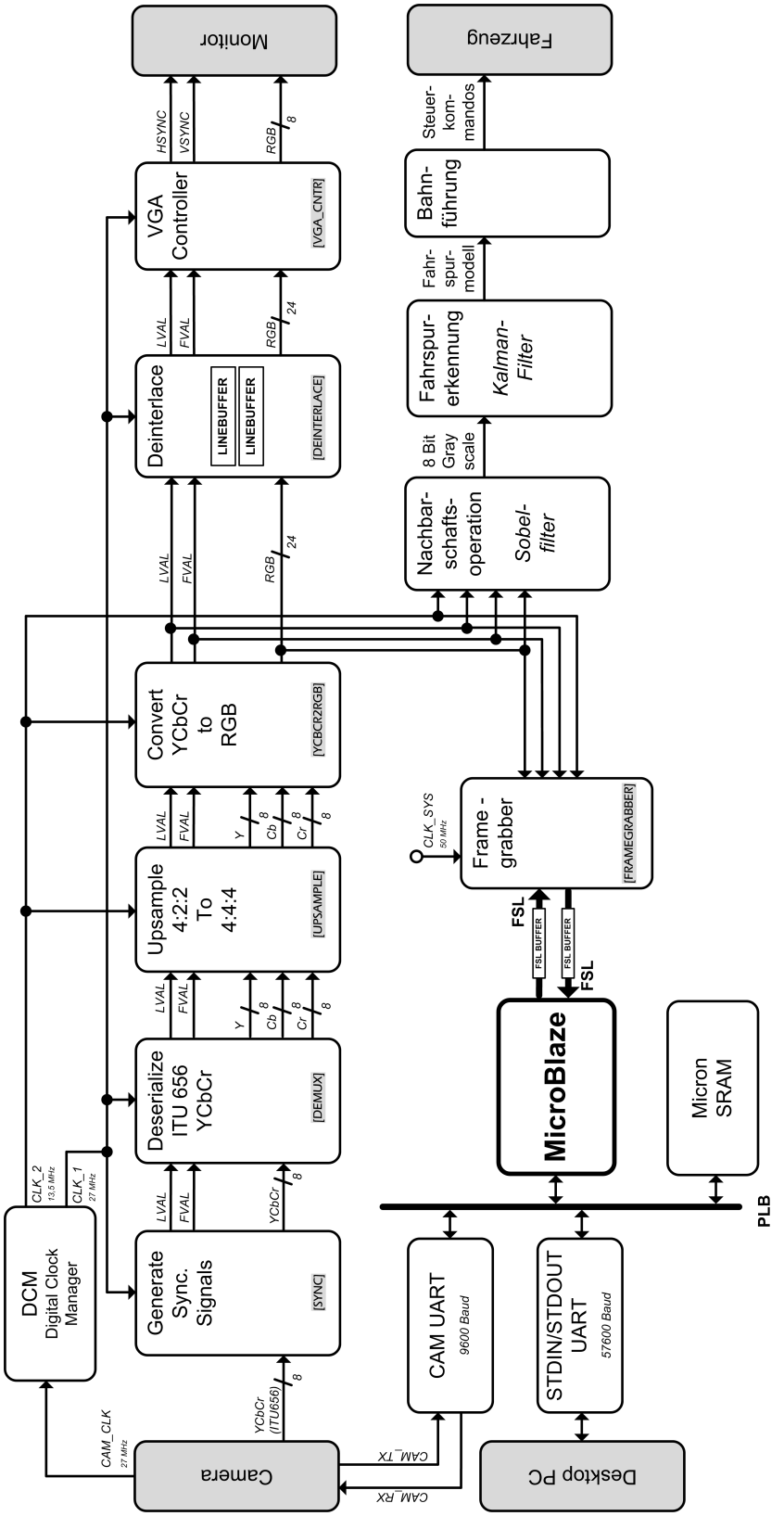

Abbildung 2.2: Gesamtübersicht der Systemkomponenten

2.2 Videopipeline

Die Kamera liefert einen Bildstrom im ITU-R BT.656 Format. Dabei handelt es sich um einen 24 Bit Interlaced-Videostrom mit 640x480 Bildpunkten pro Frame. Jedes Halbbild setzt sich somit aus 240 Zeilen zusammen. Die Pixel sind im 4:2:2 YCbCr-Farbmodell kodiert, wobei die einzelnen Farbkomponenten serialisiert übertragen werden, so dass der Videostrom über eine 8 Bit breite Datenleitung übertragen wird. Die Synchronisationssignale sind in den Videostrom eingebettet und nimmt somit keine zusätzlichen Datenleitungen in Anspruch. Die Leitungen werden über eine Pmod Schnittstelle des Entwicklungsboards zum FPGA geführt (vgl. (ITU656, 1998) sowie Abbildung 2.3).

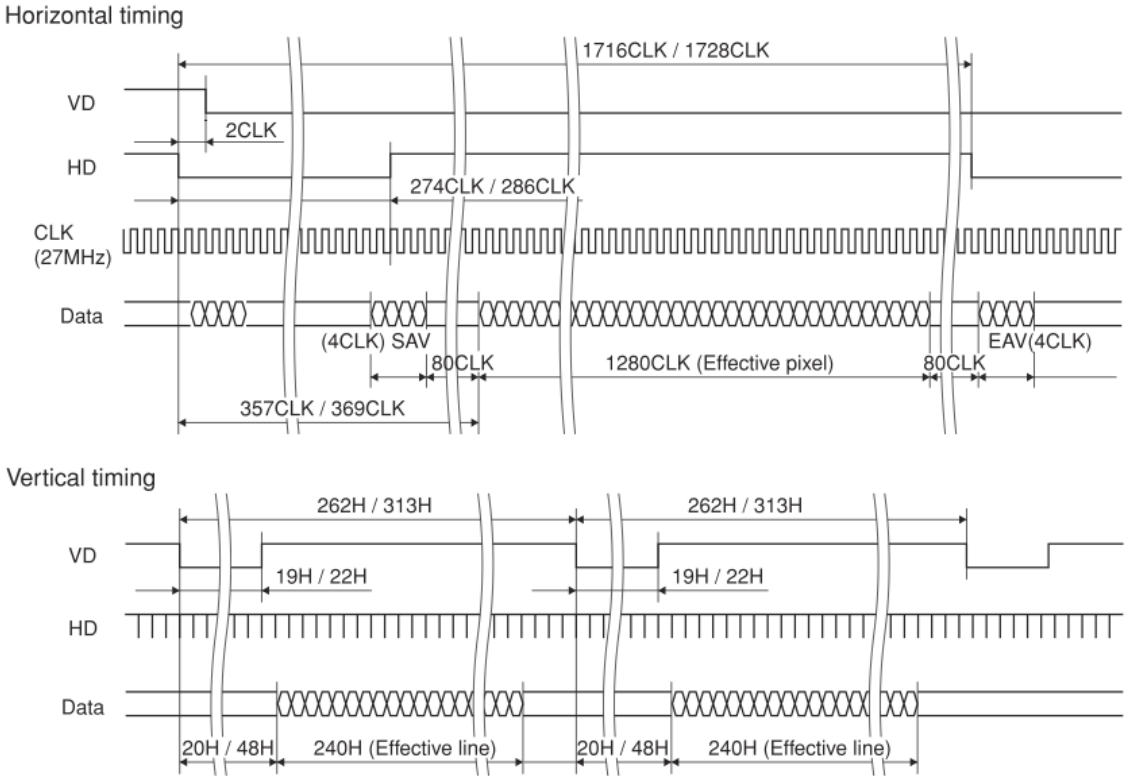

Abbildung 2.3: Timing des *ITU-R BT.656 Interlaced* Videoformats am Kameraausgang. Quelle: Datenblatt der Kamera (Sony, 2006).

Systemintern wird der Videostrom in einem 24 Bit RGB-Format behandelt. Jede der Farbkomponenten ist dabei mit acht Bit kodiert. Als Synchronisationssignale werden die Signale *Framevalid (FVAL)* und *Linevalid (LVAL)* genutzt. Diese zeigen durch einen high-Pegel an, dass gültige Pixeldaten auf der Datenleitung liegen. Das Signal *LVAL* hat eine steigende Flanke zeitgleich mit Anliegen des ersten Pixels jeder Bildzeile. Unmittelbar nach der

Übertragung des letzten Pixels der Zeile fällt *LVAL* auf low-Pegel. Das Signal *FVAL* steigt auf high-Pegel, zeitgleich mit dem *LVAL*-Signal für die erste Bildzeile. Die fallende Flanke ist zeitgleich mit der fallenden Flanke des *LVAL*-Signals der letzten Bildzeile (vgl. Abbildung 2.4).

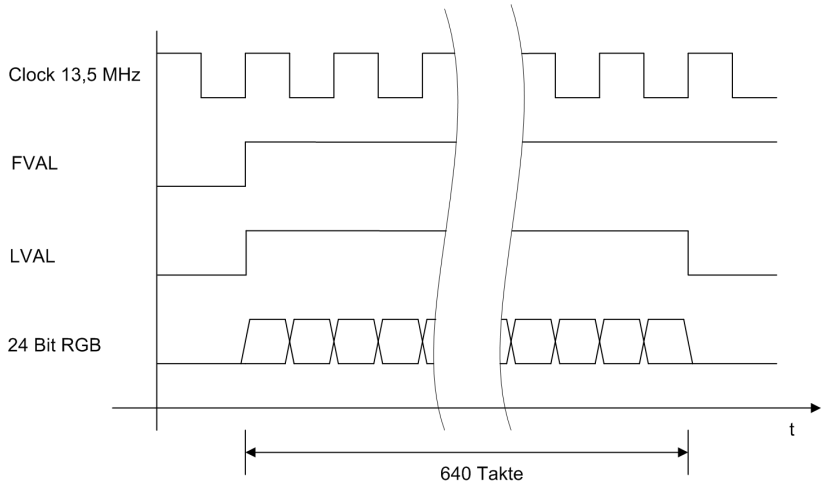

Abbildung 2.4: Timing einer Bildzeile des systemintern verwendeten 24 Bit RGB Videostroms

Das Kamerainterface wandelt den Bildstrom der Kamera in das interne Format um. Es beinhaltet die folgenden Funktionsblöcke:

Synchronisationssignale *LVAL* und *FVAL* erzeugen: In den *ITU-R BT.656* Videostrom der Kamera sind die Synchronisationssignale *SAV* (Start of Active Video) und *EAV* (End of Active Video) eingebettet (vgl. Abbildung 2.3). Diese bestehen aus Datenfolgen, welche außerhalb des Wertebereichs der YCbCr-Pixeldaten liegen und somit erkannt werden können. Die Bytefolge $0xFF - 0x00 - 0x00$ zeigt ein eingebettetes Synchronisationssignal an. Darauf folgt ein Byte mit Statusinformationen, an dem abgelesen werden kann, ob es sich um ein *SAV*- oder ein *EAV*-Signal handelt und ob dieses am Anfang oder Ende eines Frames liegt. Aus diesen Signalen werden die (nun externen) Synchronisationssignale *FVAL* und *LVAL* generiert. In den folgenden Bearbeitungsstufen werden die Synchronisationssignale jeweils parallel zu den Nutzdaten gehalten und entsprechend, wo dies nötig ist, in Registern verzögert. Die Nutzdaten bleiben in dieser Komponente unverändert (vgl. Abschnitt 3.3).

Deserialisieren der YCbCr-Farbkomponenten: Im acht Bit Kameradatenstrom sind die YCbCr-Farbkomponenten seriell angeordnet. Sie werden in diesem Modul zu einem 24 Bit Datenstrom parallelisiert, so dass jeder der Y-, Cb- und Cr-Werte nun acht Bit belegt. Der Pixelstrom ist im Datenreduzierten 4:2:2 YCbCr-Format kodiert. In

diesem Format sind die Farbinformationen Cb und Cr nur für jedes zweite Pixel vorhanden. Diejenigen Pixel, für die nur der Y-Wert vorhanden ist, werden mit Nullen aufgefüllt (vgl. Abschnitt 3.4).

Chroma Upsampling auf 4:4:4 YCbCr: Im 4:4:4 YCbCr-Format stehen für jedes Pixel die drei Farbwerte Y, Cb und Cr zur Verfügung. Das Chroma-upsampling Modul füllt die Chroma (Cb und Cr) Werte an den fehlenden Stellen auf. Das Auffüllen erfolgt durch kopieren der Werte aus dem vorhergehenden, vollständigen Pixel. Somit ist jedes Pixel mit 24 Bit kodiert und vollständig mit Farbwerten besetzt (vgl. Abschnitt 3.5).

Konvertierung in das RGB Format: Die 4:4:4 YCbCr-Pixel werden in das RGB-Farbmodel umgerechnet. Die Pixelwerte des Kameradatenstroms haben einen begrenzten Wertebereich von 16 bis 240 je Komponente, wodurch die Randbereiche $[0,15]$ und $[241,255]$ für die eingebetteten Synchronisationssignale genutzt werden können. Im systeminternen Datenformat existieren die gesonderten Synchronisationssignale *FVAL* und *LVAL*. Daher wird bei der Umrechnung in das RGB-Format auch der Wertebereich auf die vollen 0 bis 255 je Komponente erweitert (vgl. (Jack, 2005) sowie Abschnitt 3.6).

Die VGA-Schnittstelle des *Nexys 2* Boards wird genutzt um einen VGA-Monitor anzusteuern. Dazu werden die folgenden drei Schritte durchgeführt:

Deinterlacing des Videostromes: Auf dem Monitor wird das Videobild in seiner vollen Auflösung von 640x480 Bildpunkten angezeigt. Dazu wird jede Zeile des Interlaced-Bildstroms verdoppelt, wodurch effektiv ein 60Hz Progressive Videostrom entsteht (Jack, 2005) . Zur Zeilenverdopplung werden die Pixeldaten jeder Zeile zunächst in einem Schieberegister zwischengespeichert und anschließend, zusammen mit entsprechenden Synchronisationssignalen *FVAL* und *LVAL*, zweimal ausgegeben. Das Deinterlacing-Hardwaremodul enthält zwei solcher Schieberegister, die wechselweise angesteuert werden. Während eines mit den Pixeldaten einer Bildzeile gefüllt wird, gibt das jeweils andere, seinen Inhalt zweimal aus (vgl. Abbildung 2.5 sowie Abschnitt 3.7).

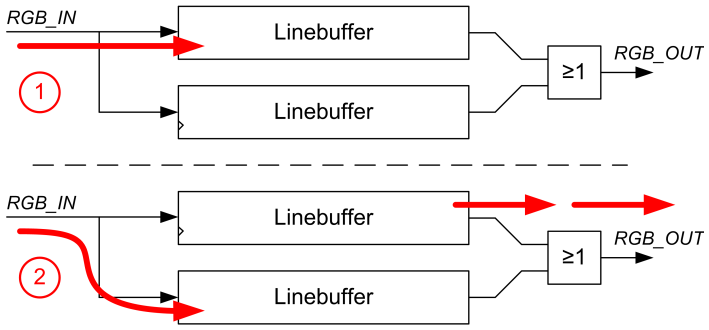

Abbildung 2.5: Sequenz der Arbeitsschritte beim Deinterlacing

Erzeugen der Synchronisationssignale für den Monitor: Der Monitor wird mit den Synchronisationssignalen *VSYNC* (Vertikale Synchronisation, also Synchronisation zwischen den Frames) und *HSYNC* (Horizontale Synchronisation, zwischen den Bildzeilen) angesteuert. Beim Signal *HSYNC* handelt es sich um *low*-Pulse während der Blankingphase zwischen den Pixeldaten jeweils zweier Bildzeilen. *VSYNC* wird durch ebensolche Pulse während des Blankings zwischen zwei Frames gebildet (vgl. Abschnitt 3.8).

Reduktion der Datenbreite auf 8 Bit: Der VGA Port des Nexys Boards nutzt 10 Signalleitungen. Neben den zwei Leitungen für die Synchronisationssignale *HSYNC* und *VSYNC* stehen damit 8 Leitungen für die Pixeldaten zur Verfügung. Zur Ansteuerung des VGA Ports wird daher eine Datenreduktion von 24 auf 8 Bit je Pixel vorgenommen. In jedem Pixel werden drei Bit für den roten Farbanteil, drei Bit für den grünen Farbanteil und zwei Bit für den blauen Farbanteil genutzt. Der blaue Farbkanal kommt mit weniger Informationen aus, da das menschliche Auge verhältnismäßig wenig empfindlich auf Schwankungen im blauen Farbanteil reagiert. Die geringere Blauauflösung wird daher nicht als störend empfunden.

Die Module des Kamera- sowie des VGA-Interfaces sind in zwei Clockdomains organisiert. Die Kamera übertragt die Daten mit einer Pixelrate von 27MHz. Dieses Clocksignal wird von der Kamera neben den Pixeldaten übertragen. Über einen digitale Clockmanager (DCM) wird aus dem Pixeltaktsignal ein zweites Clocksignal mit der halben Frequenz (13,5MHz) generiert. Die ersten beiden Stufen der Videopipeline, Generieren der Synchronisationssignale sowie Deserialisierung, werden mit dem Kameratakt von 27MHz angesteuert. Im Datenstrom der Kamera sind die Farbkomponente der einzelnen Pixel seriell angeordnet. Nach der Deserialisierungsstufe liegen diese Werte parallel an. Um weiterhin mit jedem Takt ein Datum zu übertragen wird in den folgenden Stufen, dem Upsampling- und dem RGB-Konvertierungsmodul, mit der halben Taktrate gearbeitet. Im Deinterlacing-Modul wird durch die Zeilenverdopplung auch die Pixelfrequenz am Ausgang verdoppelt. Die Komponenten des VGA-Interfaces nutzen daher beide den 27MHz Takt (vgl. Abbildung 2.2).

2.3 Microblaze Prozessorsystem

Ein Xilinx *Microblaze* Softcore-Prozessor wird genutzt, um Systemkomponenten in Softwaremodule auszulagern. Über den *Processor Local Bus (PLB)* ist der Microblaze mit den folgenden Peripheriekomponenten verbunden (vgl. (Xilinx, 2008) sowie Abbildung 2.2):

- 2 *UART Lite* serielle Schnittstellen sowie
- 16 Megabyte *Micron* SRAM des Nexys2 Boards.

Eine der seriellen Schnittstellen dient zur Kommunikation mit der Kamera, um diese mit Steuerparametern zu beschicken. Die zweite UART Verbindung dient als STDIN/STDOUT und wird zur Datenübertragung, insbesondere der Einzelbildübertragung, an einen angeschlossenen PC genutzt. Das *Micron* SRAM wird als Puffer genutzt um Frames aus dem Videostrom vor einer solchen Bildübertragung zwischen zu speichern.

Die Sony Kamera ist über eine serielle Schnittstelle zur Systemlaufzeit parametrierbar. Es wird eine Vielzahl von Kommandos und Einstellmöglichkeiten unterstützt von denen einige wichtige hier beispielhaft genannt seien:

- Power On/Off
- Zoom Tele/Wide
- Einstellen des Fokus, Autofokus/Manueller Fokus
- Einstellen der Blende
- und viele weitere (vgl. Sony, 2006).

Die Steuerung der Kamera erfolgt über eine UART Schnittstelle mit einer Datenrate von 9600 Baud. Die Kommandos für die Kamera entsprechen dem von Sony Entwickelten *VISCA*-Format. Die *VISCA*-Kommandos werden von einem Softwaremodul über die serielle Schnittstelle verschickt. Die Datenleitungen werden über eine Pmod Schnittstelle aus dem Nexys2 Board heraus geführt.

2.4 Bildspeicherung

Die Systemkomponente zur Speicherung einzelner Frames aus dem laufenden Videostrom ist eine Kombination aus Hardware- und Softwaremodulen. Ziel ist es, einen Bitmap Datenstrom mit den Bilddaten eines Frames auf einem angeschlossenen PC zu speichern. Als Datenverbindung zwischen FPGA-Plattform und PC wird eine serielle Schnittstelle mit 57600 Baud verwendet. Die Datenrate dieser Schnittstelle beträgt nur einen Bruchteil der Datenrate des Videostroms, wodurch eine unmittelbare Übertragung des Bildes nicht möglich ist. Stattdessen werden die Daten zunächst im SRAM zwischengespeichert. Nachdem ein Frame vollständig gespeichert ist, werden ein entsprechender Bitmap-Dateiheader und danach die Bilddaten über die vergleichsweise langsame serielle Schnittstelle STDOUT versendet. Auf der PC-Seite kann der Datenstrom unmittelbar als Bitmap-Datei (.bmp) gespeichert werden. Als I/O-Schnittstelle wird die Onboard RS232 Schnittstelle des Entwicklungsboards genutzt (vgl. Abschnitt 5).

Das Hardwaremodul zur Bildspeicherung ist über eine *Fast Simplex Link (FSL)* Schnittstelle an den Microblaze Prozessor angebunden. Der FSL-Bus ist eine gepufferte unidirektionale Punkt-zu-Punkt Verbindung welche die Funktionalität eines FIFOs zur Datenübertragung zwischen zwei Hardwaremodulen bietet. Das FSL Interface steht auch auf dem Microblaze

Prozessor zur Verfügung. Jeweils zwei FSL-Buskomponenten werden verwendet um Daten bidirektional zwischen dem Registerfile des Prozessors und weiteren Hardwarekomponenten im FPGA zu übertragen. Der FSL-Bus hat eine Datenbreite von 32 Bit (Xilinx, 2007a).

Das Framegrabber-Modul nimmt den Videostrom in einer Datenreduzierten Variante entgegen. Analog zur Datenreduktion für die Ansteuerung des VGA-Displays werden die 24 Bit Bilddaten auf acht Bit pro Pixel reduziert. Um die Datenbreite des FSL-Busses auszunutzen werden die acht Bit Pixeldaten im Framegrabber über ein Schieberegister parallelisiert. Die Daten von jeweils vier Pixeln werden gleichzeitig mit dem Bus übertragen. Auf Prozessorseite werden die empfangenen Datenpakete per Software in einer für diesen Zweck reservierten Speichersektion im *Micron* SRAM abgelegt. Das SRAM ist mit dem Microblaze über den, verglichen mit dem FSL, langsamen PLB-Bus verbunden. Auf Grund der Datenreduktion sowie der Paketbildung einerseits und der verglichen mit der Videopipeline höheren Taktrate des Microblaze Prozessors (50 MHz vs. 13,5MHz) andererseits, stehen zum Lesen eines Pakets vom FSL-Bus und Schreiben in das SRAM insgesamt mindestens 14 Prozessortakte zur Verfügung. Die Datenrate des PLB ist ausreichend um in dieser Zeit einen Schreibvorgang durchzuführen.

Auf das vollständige Speichern eines Frames folgt das Bilden eines Bitmap Datenstroms. Dazu wird zunächst ein Bitmap-Dateiheader für ein 640x480 Pixel großes Bild mit 24 Bit Farbtiefe erzeugt. Dieser wird auf STDOUT (die UART Schnittstelle) geschrieben. Nach dem Dateiheader werden die Pixeldaten übertragen. Da der Videostrom *interlaced* ist, wird von der Software eine Zeilenverdopplung durchgeführt um die volle vertikale Auflösung von 480 Zeilen wieder herzustellen. Zu diesem Zweck werden die Daten jeder Bildzeile zweimal gesendet.

2.5 Fahrspurerkennung

Zur Verfolgung der Fahrspur werden im Bild Messpunkte gesucht, welche auf der Fahrbahnbegrenzung liegen. Das Extrahieren der Messpunkte geschieht über Nachbarschaftsoperationen, bei denen jeder Bildpunkt im Frame durch die gewichtete Summe seiner Nachbarpixel ersetzt wird. Ein Beispiel einer solchen Operation ist der Sobelfilter, der jedem Pixel in einem Grauwertbild den Betrag des Grauwertgradienten an dieser Stelle zuordnet. Befinden sich Kanten im Bild, wie etwa eine Fahrspurmarkierung, ist an dieser Stelle der Betrag des Gradienten groß und die Kante wird als helle Linie dargestellt.

Für die Durchführung der Nachbarschaftsoperation wird auf die Pixelwerte von drei Bildzeilen zugegriffen. Es wurde eine Hardwarekomponente entwickelt, welche zwei Bildzeilen in einem Schieberegister zwischenspeichert und über eine Registermatrix den Zugriff auf den jeweiligen Pixel und seine acht Nachbarn ermöglicht. Mit den Pixelwerten werden auch die zu jedem Pixel gehörenden Synchronisationssignale *FVAL* und *LVAL* in Schieberegistern

mitgeführt. Die zum aktuellen, also dem mittleren, Pixel gehörenden Synchronisationssignale werden zusammen mit den neun Pixelwerten aus dieser Deserialisierungskomponente herausgeführt. Am Ausgang des Deserialisierers liegt somit ein vollständiger Videostrom mit Synchronisationssignalen an, wobei für jedes Pixel auch die Werte seiner acht Nachbarn zur Verfügung stehen. Bei den Pixeln am Rand des Urbildes gibt es keine Nachbarn. Am Ausgang liegen in diesem Fall die Datenwerte aus der Blankingphase an. Es liegt in der Verantwortung der auf die Deserialisierung folgenden Reduktionsstufe, mit diesen Werten sinnvoll umzugehen.

In der Videopipeline folgt auf einen Deserialisierer immer eine Reduktionsstufe, welche die neun Pixelwerte in einer der Operation entsprechenden Weise mit ihren Koeffizienten multipliziert und aufsummiert. Am Ausgang der Reduktionsstufe steht dann wieder ein einfacher Videostrom, das Ergebnis der Nachbarschaftsoperation zur Verfügung. Zur Implementierung einer Nachbarschaftsoperation wird die generische Deserialisierungskomponente instantiiert auf die eine operationsspezifische Reduktionskomponente folgt. Auf das genaue Vorgehen und die Abfolge von Nachbarschaftsoperationen zum auffinden der Messpunkte geht diese Arbeit nicht ein. Die Arbeiten (Jenning, 2008) sowie (Risack u. a., 1998) erläutern hierfür Vorgehensvarianten. Bei den weiteren Überlegungen wird davon ausgegangen, dass eine Menge von Messpunkten, welche auf der Fahrspurbegrenzung liegen, zur Verfügung steht.

Die Fahrspur wird als Polynom dritten Grades approximiert. Um eine robuste und präzise Abschätzung dieses Polynoms zu ermöglichen wird der Verlauf der Fahrspur mit der Zeit als ein dynamisches System modelliert. Die entsprechenden Systemgleichungen wurden hergeleitet und es wurde ein Kalman-Filter entwickelt, welches eine optimale Abschätzung des Fahrspurpolynoms anhand der Messpunkte erlaubt. Die Randbedingungen des Carolo-Cup lassen eine Reihe von Vereinfachungen zu, welche in die Entwicklung des Kalman-Filters eingeflossen sind.

3 Videopipeline

Dieser Abschnitt beschreibt die Komponenten der Videopipeline. Diese unterteilt sich in das Kamera-Interface sowie das VGA-Interface. Das Kamera-Interface wandelt den Datenstrom der Kamera vom ITU-R BT.656 Format in das systemintern genutzte 24 Bit RGB Videoformat um. Das VGA-Interface generiert aus dem systeminternen Format geeignete Signale zum Ansteuern des VGA-Ports des Nexys Boards, so dass das Videobild der Kamera auf einem VGA Bildschirm dargestellt werden kann.

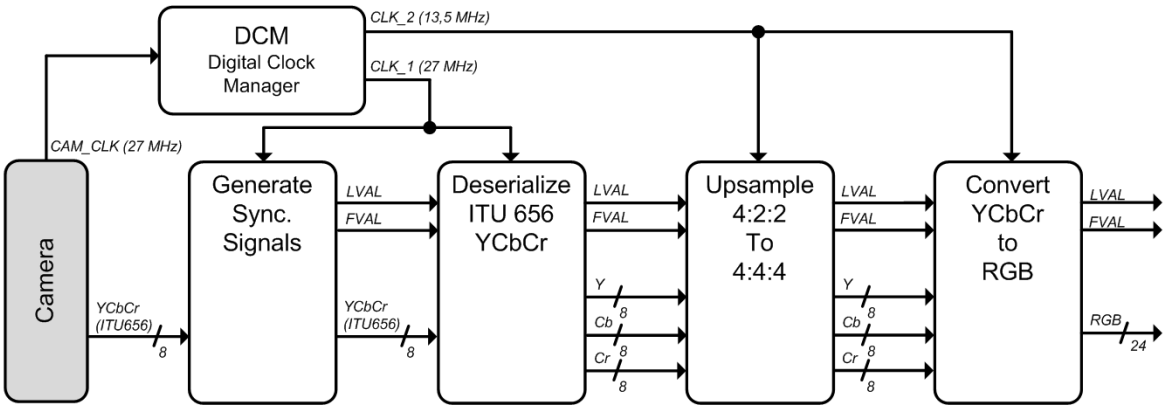

Abbildung 3.1: Blockschaltbild des Kamerainterface

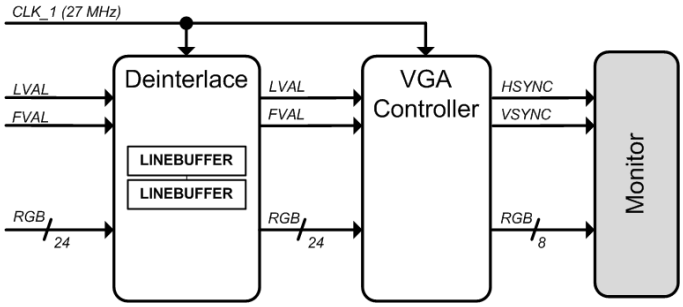

Abbildung 3.2: Blockschaltbild des VGA-Interface

3.1 Sony FCB-PV10 Farbbildkamera

Die Sony FCB-PV10 Kamera ist eine digitale Farbbild-Videokamera. Sie liefert einen Videostrom mit einer Bildauflösung von 640×480 Pixeln, wobei zwischen drei Videoforma-

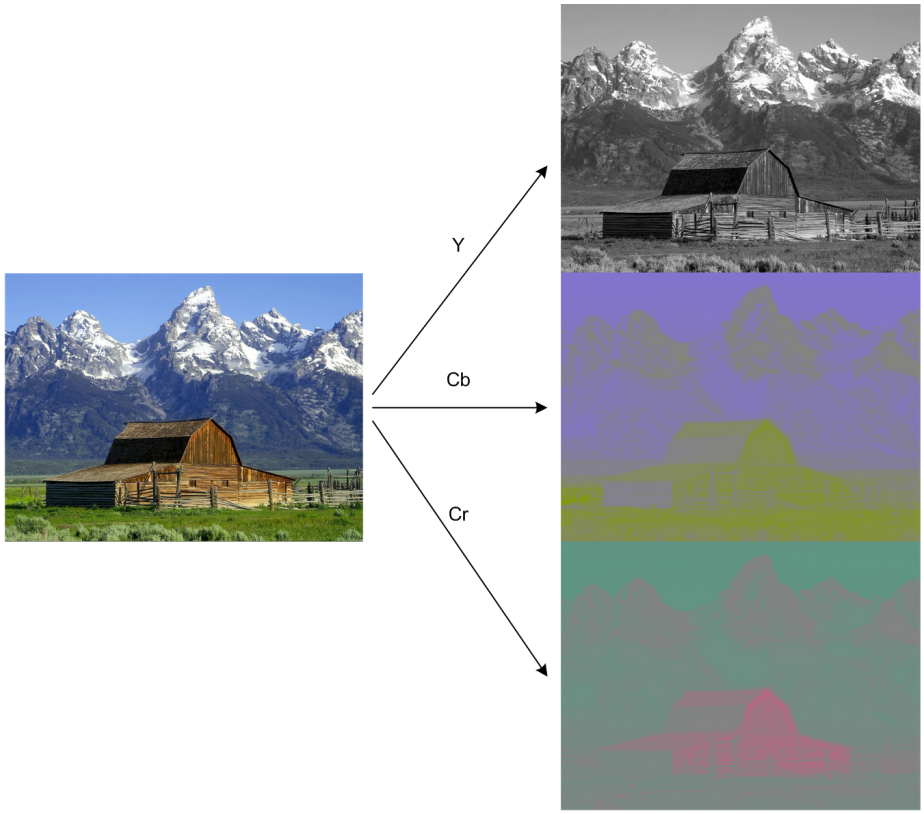

Abbildung 3.3: Originalbild und Aufspaltung des Bildes in die Komponenten Y, Cb und Cr (Wikipedia, 2006).

ten gewählt werden kann (siehe unten). In allen Formaten sind die Pixel im 4:2:2 YCbCr Format mit 8 Bit pro Wert kodiert (Sony, 2006).

Das 4:2:2 YCbCr Format ist ein Datenreduziertes Farbformat, welches die schwache Farbempfindlichkeit des menschlichen Auges ausnutzt. Der Y-Wert entspricht weitgehend dem Grauwert des jeweiligen Pixels (vgl. Abbildung 3.3). Solange dieser in voller Genauigkeit dargestellt wird, ist es möglich die Farbinformationen (Cb- und Cr-Werte) durch Subsampling zu reduzieren, ohne dass es zu einer für Menschen sichtbaren Verringerung der Bildqualität kommt. Dies wird durch die Cb- und Cr- Bilder in Abbildung 3.3 anschaulich gemacht, die für das menschliche Empfinden wenig Informationen enthalten. Im 4:2:2 YCbCr Format gibt es für jedes Pixel einen korrespondierenden Y-Wert. Cb- und Cr- Werte stehen für jedes zweite Pixel zur Verfügung (vgl. Abbildung 3.4). Bei einer Kodierung mit 8 Bit pro Wert ergibt sich somit ein Datenaufkommen von durchschnittlich 16 Bit pro Pixel, was gegenüber einer Kodierung mit korrespondierenden Cb- und Cr-Werten zu jedem Y-Wert (der 4:4:4 Kodierung, vgl. Abschnitt 3.5) einer Datenreduktion um 33% entspricht (ITU601, 1994).

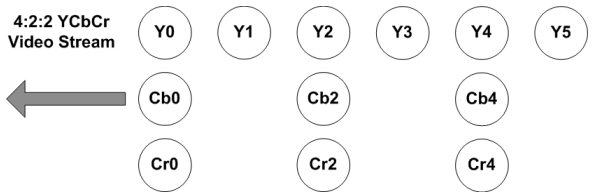

Abbildung 3.4: Pixelstrom im 4:2:2 YCbCr Datenformat. Cb- und Cr-Werte sind nur für jeden zweiten Pixel vorhanden.

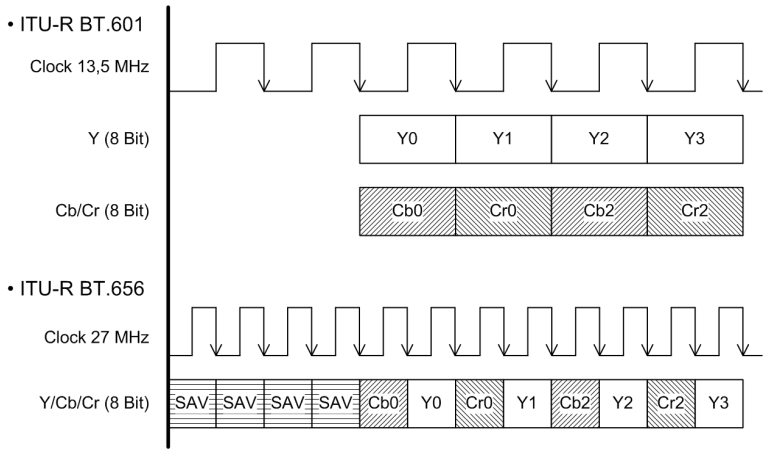

Abbildung 3.5: Datenformat in den Kamera-Interfacemodi BT.601 und BT.656

Die Sony-Kamera bietet drei verschiedene digitale Interface-Modi. Für dieses Projekt wurde die Betriebsart *ITU-R BT.656 Interlaced* (ITU656, 1998) ausgewählt. Begründet ist diese Wahl in den Eigenschaften der verschiedenen Betriebsarten (vgl. Tabelle 3.1), deren Vor- und Nachteile im folgenden erläutert werden:

Interlaced vs. Progressive: Wird der Interlaced-Bildstrom als Progressive interpretiert, entspricht er durch die ausgelassenen Zeilen einem Videostrom mit halber vertikaler Abtastrate. Dies hat eine Datenreduktion um 50% je Frame zur Folge. Das geringere Datenaufkommen nimmt Arbeit von folgenden Verarbeitungsstufen. Potentieller Nachteil ist die Minderung der Bildqualität und der damit verbundene Informationsverlust.

Datenbreite: Die Datenbreite bestimmt, wie viele Pins des Entwicklungsboards und damit des FPGAs von der Datenleitung der Kamera belegt werden. Im Kontext einer späteren Systemintegration in das Carolo-Cup Fahrzeug, bei der eine Anzahl weitere Anwendungen (z.B. Bahnführungssystem) auf dem gleichen Chip in Betrieb genommen werden, ist es sinnvoll, möglichst wenige Pins zu beanspruchen, da diese eine sehr begrenzte Ressource darstellen.

Synchronisationssignale: Eng mit dem Problem des Pinbedarfs hängt die Art der ho-

Tabelle 3.1: Interface-Modi der Sony FCB-PV10 Kamera (Sony, 2006), entsprechend den Standards (ITU601, 1994) und (ITU656, 1998).

Modus	Datenbreite	Sync. Signale	Frames/sec.	Clock Freq.
BT.601 Progressive	16 Bit	HSYNC/VSYNC	29.97	13,5 MHz
BT.656 Progressive	8 Bit	SAV/EAV	29.97	27 MHz
BT.656 Interlaced	8 Bit	SAV/EAV	59.94	27 MHz

rizontalen und vertikalen Synchronisationssignale zusammen. Die Signale *HSYNC* und *VSYNC* werden über gesonderte Signalleitungen übertragen und benötigen somit zusätzliche Pins. Dagegen sind die Synchronisationssignale *SAV* und *EAV* in den Videostrom *eingebettet*. Daher werden zu ihrer Übertragung keine zusätzlichen Leitungen bzw. Pins benötigt. In der Folge kommt es aber zu einem erhöhten Aufwand bei der Dekodierung des Videostroms, aus dem die *SAV/EAV*-Signale erst wieder extrahiert werden müssen (vgl. Abschnitt 3.3).

Bildrate: Eine hohe Bildrate ist prinzipiell wünschenswert, da somit mehr Informationen in der gleichen Zeit transportiert werden können. Dies hat für die vorliegende Arbeit vor allem zur Folge, dass für die Fahrspurerkennung (vgl. Abschnitt 6) in einem gleichen Zeitraum mehr Messpunkte zur Verfügung stehen, wodurch die Genauigkeit der Fahrspurerkennung erhöht wird.

Der *ITU-R BT.656 Interlaced* Modus ist somit am besten geeignet, da er einerseits einen geringen Pinbedarf sowie andererseits eine hohe Bildrate – und damit eine erhöhte Messgenauigkeit – bietet.

3.2 Testbildgenerator nach ITU-R BT.656 Interlaced Videostandard

Um die Videopipeline unabhängig von der Kamera entwickeln und testen zu können wurde ein Testbildgenerator implementiert. Dieser erzeugt einen *ITU-R BT.656 Interlaced* Videostrom welcher identisch mit dem Videoformat der Kamera ist. Als Testbild werden fünf vertikale Streifen in den Farben Rot, Grün, Blau, Schwarz und Weiß generiert. In jeder Zeile (640 Bildpunkte) werden 128 aufeinander folgende Pixel in der jeweiligen Farbe generiert. Dies wird für alle Zeilen identisch durchgeführt, wodurch vertikale Streifen entstehen (vgl. Abbildung 3.6).

Zur Steuerung des Bildgenerators werden zwei Zähler mit einem Takt von 27Mhz betrieben. Sie bestimmen die horizontale und die vertkale Position im Bildstrom. Entsprechend

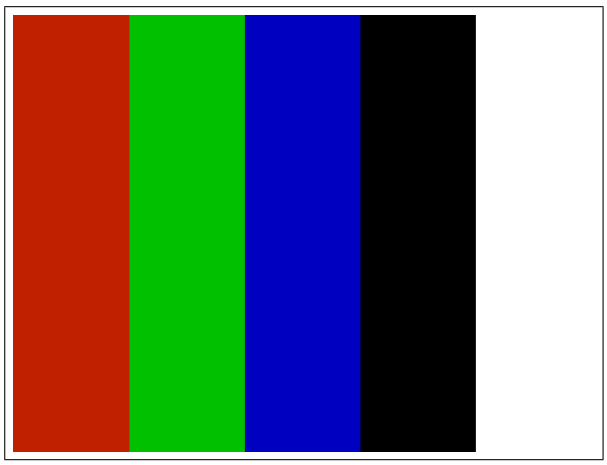

Abbildung 3.6: Bild des Testscreengenerators

dem Timingdiagramm der Kamera (Abbildung 2.3) werden 1716 Take für die horizontale Bildposition gezählt. Danach wird der horizontale Zähler zurückgesetzt und gleichzeitig der vertikale Zähler inkrementiert. Der vertikale Zähler wird jeweils nach 262 Zeilen zurückgesetzt. Anhand von Bereichen der Bildpositionszähler wird die Ausgabe bestimmt: Die jeweiligen Farbwerte in den Bildzeilen, EAV und SAV Synchronisationssignale sowie Nullen während der Blankingphasen.

Die Farbwerte werden nach dem 4:2:2 YCbCr Farbsystem so kodiert, dass für jeweils zwei Pixel zwei Y-Werte, ein Cb- und ein Cr-Wert zur Verfügung stehen (vgl. Abbildung 3.5). Zur Erzeugung dieser Werte wird anhand der beiden LSBs des horizontalen Zählers bestimmt, welcher Wert aktuell ausgegeben wird. Listing 3.1 zeigt dies beispielhaft an der Generierung der roten Pixel. Die Farbwerte bilden ein Rot mit 75% Intensität (Jack, 2005, S. 18).

Listing 3.1: Generieren von roten Pixeln in einem Abschnitt einer Bildzeile

```
1  ...
2  tmp := hpos(1 downto 0);
3
4  if vpos>=20 and vpos<260 then
5      if hpos ...
6      ...
7      --red
8      elsif hpos>=357 and hpos<613 then
9          if     tmp = "01" then data_out<= "01100100"; --cb
10         elsif tmp = "10" then data_out<= "01000001"; --y
11         elsif tmp = "11" then data_out<= "11010100"; --cr
12         elsif tmp = "00" then data_out<= "01000001"; --y
13         end if;
14     ...
```

3.3 Synchronisationssignale *LVAL* und *FVAL*

Synchronisationssignale in einem Videostrom dienen dazu, die Elemente des Datenstroms den korrekten Bildpunkten zu zuordnen. Die Synchronisationssignale zeigen Beginn oder Ende eines Frames bzw. einer Bildzeile an. Es kann zwischen eingebetteten und externen Synchronisationssignalen unterschieden werden. Eingebettete Signale nutzen die gleichen Signalleitungen wie die Daten. Bei externen Signalen stehen gesonderte Leitungen für die Synchronisationssignale zur Verfügung. Der Datenstrom der Kamera nutzt interne Synchronisationssignale, *EAV* und *SAV*. Aus diesen werden die externen Signale *Framevalid* (*FVAL*) und *Linevalid* (*LVAL*) generiert (vgl. Abschnitt 2).

Die *SAV* und *EAV* Sequenzen im Kameradatenstrom werden durch die Bytefolge $0xFF$, $0x00$, $0x00$ angezeigt. Diese Werte liegen außerhalb des Wertebereichs der Pixeldaten und können somit eindeutig erkannt werden. Auf diese drei Byte folgt ein Statusbyte, welches die Position im Bildstrom eindeutig beschreibt (vgl. Tabelle 3.2).

Tabelle 3.2: Datenformat der SAV und EAV Sequenzen (ITU656, 1998)

	8-bit Datenwort							
	D7(MSB)	D6	D5	D4	D3	D2	D1	D0(LSB)
Byte 0	1	1	1	1	1	1	1	1
Byte 1	0	0	0	0	0	0	0	0
Byte 2	0	0	0	0	0	0	0	0
Byte 3	1	F	V	H	P3	P2	P1	P0

(F)ield: Zeigt im Interlaced-Modus das Feld an, zu dem der aktuelle Frame gehört. ’0’ für Feld 1, ’1’ für Feld 2.

(V)ertical: ’1’ während des vertikalen Blankings, sonst ’0’.

(H)orizontal: ’0’ bei *SAV*, ’1’ bei *EAV*.

(P)rotection: Schutzbits, welche das Erkennen von Ein- und Zweibitfehlern sowie die Korrektur von Einbitfehlern ermöglichen (Jack, 2005).

Die Hardwarekomponente generiert die Signale *LVAL* und *FVAL* synchron zum Videodatenstrom. Dieser bleibt jedoch zunächst unverändert. Eine tiefergehende Erläuterung der Arbeitsweise dieser Komponente findet sich in (Kirschke, 2009).

3.4 Parallelisieren des Pixelstroms

Im ITU-R BT.656 Videostrom sind die Y, Cb und Cr Farbkomponenten seriell angeordnet (vgl. Abbildung 3.5). Das Deserialisierungsmodul ordnet die Komponenten für jedes Pixel parallel an und generiert ein 24 Bit YCbCr-Videosignal. Da es sich um einen Datenreduzierten 4:2:2 Videostrom handelt sind die Cb- und Cr-Farbkomponenten nur für jedes zweite Pixel vorhanden. Diejenigen Pixel, denen diese Werte fehlen haben nur einen Y-Farbwert und werden mit Nullen aufgefüllt. Die Folge ist eine Halbierung des Pixeltakts auf 13,5 MHz. Den folgenden Verarbeitungsstufen stehen damit alle Farbwerte für ein Pixel gleichzeitig zur Verfügung (Kirschke, 2009).

3.5 4:2:2 YCbCr Chroma Upsampling

Vor der Umwandlung der YCbCr Daten in das RGB Format müssen die 4:2:2 YCbCr Daten in 4:4:4 YCbCr Daten umgewandelt werden. Dies ist notwendig, da für den Umwandlungsprozess von YCbCr nach RGB für jedes Pixel die zugehörigen Y-, Cb- und Cr-Werte zur Verfügung stehen müssen.

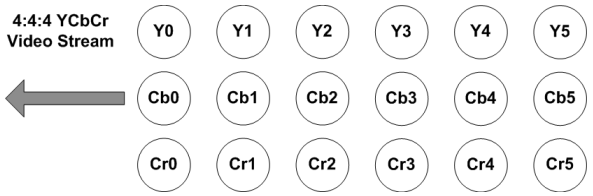

Abbildung 3.7: Pixelstrom im 4:4:4 YCbCr Datenformat. Für alle Pixel sind Y-, Cb- und Cb-Werte vorhanden.

Im 4:4:4 YCbCr Format besteht jedes Pixel aus einem Y-Wert, einem Cb-Wert und einem CR-Wert (vgl. Abbildung 3.7). Jede dieser Komponenten ist mit 8 Bit kodiert, so dass jedes Pixel mit insgesamt 24 Bit kodiert ist. Die Umwandlung vom 4:2:2 in das 4:4:4 YCbCr Format findet dadurch statt, dass für diejenigen Pixel, die keine Cb- und Cr- Werte besitzen, Cb- und Cr-Werte generiert werden. Für diesen, als Upsampling bezeichneten Vorgang sind die drei folgenden Strategien möglich (Poynton, 2005):

Pixel Replikation: Wenn gewöhnliche VHS-Bildqualität das Designziel hinreichend erfüllt können die fehlenden Cb- und Cr-Werte vom vorhergehenden Pixel übernommen werden (vgl. Abbildung 3.8).

$$Cb[i] = Cb[i-1] \tag{3.1}$$

Für den Cr-Wert gilt hier, sowie auch bei den beiden folgenden Verfahren, eine entsprechende Formel.

Lineare Interpolation: Eine gegenüber der Pixel Replikation erhöhte Bildqualität lässt sich durch lineare Interpolation der fehlenden Cb- und Cr- Werte aus den Werten der beiden Nachbarpixel erzielen.

$$Cb[i] = \frac{Cb[i-1] + Cb[i+1]}{2} \tag{3.2}$$

FIR Filterung: Das beste Ergebnis wird durch den Einsatz eines geeigneten FIR-Filters erzielt. Bei der Implementierung eines solchen Filters ist auf mögliche Fehler durch Vorzeichen, Skalierung, Overflow und Anfangs- sowie Endbedingungen zu achten. Dazu können beispielsweise die Bilddaten mit Null-Samples umgeben werden. Wenn die Summe der Filterkoeffizienten ungleich eins ist, sollte das Ergebnis passend skaliert werden. Die Koeffizienten des folgenden Filters ergeben in der Summe 256. Das Ergebnis wird dementsprechend durch 256 geteilt, was sich in Hardware effizient als Schiebeoperation implementieren lässt.

$$\begin{aligned} Cb[i] = (&+160(Cb[i-1] + Cb[i+1]) - 48(Cb[i-3] + Cb[i+3]) \\ &+ 24(Cb[i-5] + Cb[i+5]) - 12(Cb[i-7] + Cb[i+7]) \\ &+ 6(Cb[i-9] + Cb[i+9]) - 2(Cb[i-11] + Cb[i+11]))/256 \end{aligned} \tag{3.3}$$

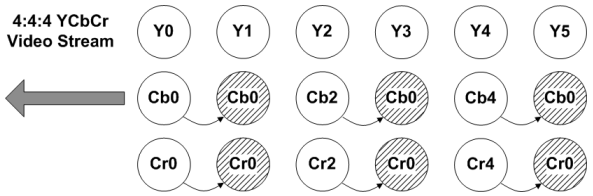

Abbildung 3.8: Upsampling vom 4:2:2 zum 4:4:4 YCbCr-Format durch Pixel Replikation

Die vorliegende Arbeit implementiert die erste Methode, Pixel Replikation, indem die Pixel ohne Cb- und Cr-Werte diese Werte vom vorherigen Pixel übernehmen (vgl. Abbildung 3.8). Die beiden anderen Methoden seien hier als Perspektive für eine mögliche Weiterentwicklung im Kontext der Systemintegration im Carolo-Cup Fahrzeug genannt.

Listing 3.2: Schnittstellendefinition der UPSAMPLE Entity

```
1  entity UPSAMPLE is
2      port (
3          CLK          : in  std_logic;
4          RST          : in  std_logic;
5
6          LVAL_IN      : in std_logic;
7          FVAL_IN      : in std_logic;
8          Y_IN         : in std_logic_vector (7 downto 0);
9          CB_IN        : in std_logic_vector (7 downto 0);
10         CR_IN        : in std_logic_vector (7 downto 0);
11
```

```
12          LVAL_OUT    : out std_logic;
13          FVAL_OUT    : out std_logic;
14          Y_OUT       : out std_logic_vector (7 downto 0);
15          CB_OUT      : out std_logic_vector (7 downto 0);
16          CR_OUT      : out std_logic_vector (7 downto 0)
17      );
18  end UPSAMPLE;
```

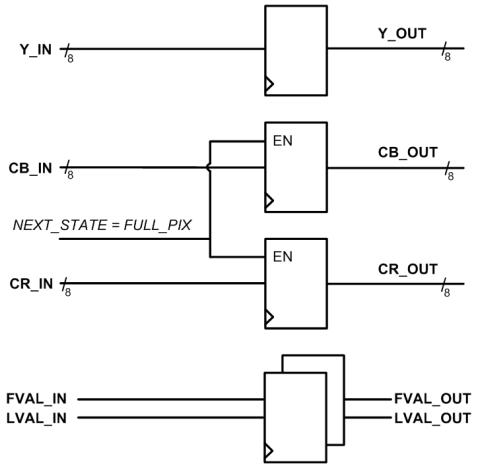

Abbildung 3.9: Datenpfad des Upsampling Moduls

Die Schnittstelle des Upsampling Moduls ist in Listing 3.2 beschrieben. Abbildung 3.9 zeigt das Blockschaltbild des Datenpfades des Upsampling Moduls. Mit jedem Takt werden die Signale LVAL_IN, FVAL_IN und Y_IN in Registern gespeichert, deren Ausgänge die Signale LVAL_OUT, FVAL_OUT und Y_OUT sind. Die Replikation der Cb- und Cr-Werte findet dadurch statt, dass die Cb- und Cr-Werte eines vollständigen Pixels ebenfalls direkt in die Ausgangsregister gespeichert werden. Liegt dagegen ein unvollständiges Pixel am Eingang an, werden die Cb- und Cr-Werte des vorhergehenden (vollständigen) Pixels in den Ausgangsregistern gehalten.

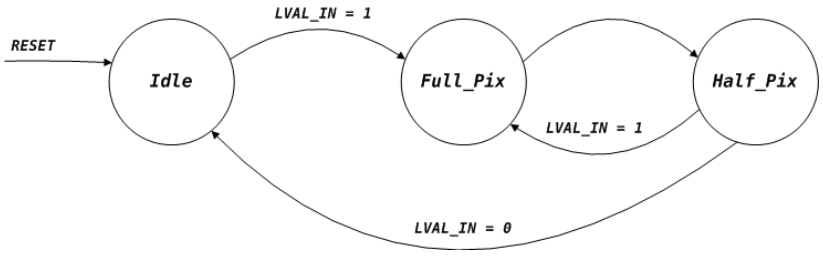

Abbildung 3.10: Steuerungs-FSM des Upsampling Moduls

Die Steuerung dieses Systems erfolgt durch einen Zustandsautomaten gemäß Abbildung 3.10. Mit einer steigenden Flanke des LVAL_IN Signals, also mit dem Beginn einer

neuen Bildzeile, springt der Automat in den Zustand FULL_PIX, der ein Pixel anzeigt, bei dem der Y-Wert sowie die Cb- und Cr-Werte vorhanden sind. Mit dem nächsten Pixel wechselt der Automat den Zustand zu HALF_PIX, der ein Pixel ohne Cb- und Cr-Werte bedeutet. Zwischen diesen Zuständen springt der Automat während der Dauer einer Bildzeile hin und her. Am Ende der Bildzeile (LVAL_IN = 0) wechselt der Automat in den IDLE Zustand. Das Übernehmen bzw. Halten der Cb- und Cr-Werte in den Ausgangsregistern erfolgt durch ein Enable-Signal, welches gesetzt ist, wenn der Folgezustand der FSM FULL_PIX ist, also immer dann, wenn ein vollständiges Pixel am Eingang anliegt. In allen anderen Fällen werden die Werte in den CB_OUT und CR_OUT Registern gehalten.

3.6 Konvertieren des Pixelstroms in das RGB Format

Das Konvertierungsmodul wandelt die Komponenten eines YCbCr-Pixels in 24 bit RGB-Werte um. Neben der Umrechnung wird der Wertebereich der Pixeldaten angepasst. Der Wertebereich der Y-Werte ist $[16,235]$, der Wertebereich der Cb- und Cr-Werte ist $[16,240]$. Ziel der Konvertierung sind RGB-Werte mit einem Wertebereich von jeweils $[0,255]$. Diese Umrechnung erfolgt entsprechend den folgenden Zusammenhängen (Jack, 2005).

$$R = 1.164(Y-16) + 1.596(Cr-128) \tag{3.4}$$
$$G = 1.164(Y-16) - 0.813(Cr-128) - 0.391(Cb-128) \tag{3.5}$$
$$B = 1.164(Y-16) + 2.018(Cb-128) \tag{3.6}$$

Für die Implementierung im FPGA ist es sinnvoll die Rechnung in Integer-Arithmetik durchzuführen. Hierzu werden die o.g. Gleichungen mit einer Zweierpotenz erweitert, vereinfacht und schließlich wieder durch die gleiche Zweierpotenz geteilt. Prinzipiell ist das Erweitern und Teilen mit einer beliebigen ganzen Zahl möglich, eine Zweierpotenz hat jedoch in der Praxis den Vorteil, dass sich die abschließende Division effizient als Schiebeoperation implementieren lässt. Die entstehenden Rundungsfehler lassen sich reduzieren, indem eine möglichst große Zahl zum Erweitern/Teilen gewählt wird. Die vorliegende Arbeit erweitert die o.g. Gleichungen mit $2^8 = 256$. Daraus ergeben sich die folgenden Gleichungen für die Konvertierung von YCbCR nach RGB.

$$R = (298Y + 409Cr - 57065)/256 \tag{3.7}$$
$$G = (298Y - 100Cb - 208Cr + 34658)/256 \tag{3.8}$$
$$B = (298Y + 516Cb - 70894)/256 \tag{3.9}$$

Listing 3.3: Schnittstellendefinition des YCbCr to RGB Konvertermoduls

```
entity YCBCR2RGB is
    port (
        CLK         : in  std_logic;
        RST         : in  std_logic;

        LVAL_IN     : in std_logic;
        FVAL_IN     : in std_logic;
        Y_IN        : in std_logic_vector (7 downto 0);
        CB_IN       : in std_logic_vector (7 downto 0);
        CR_IN       : in std_logic_vector (7 downto 0);

        LVAL_OUT    : out std_logic;
        FVAL_OUT    : out std_logic;
        R_OUT       : out std_logic_vector (7 downto 0);
        G_OUT       : out std_logic_vector (7 downto 0);
        B_OUT       : out std_logic_vector (7 downto 0)
    );
end YCBCR2RGB;
```

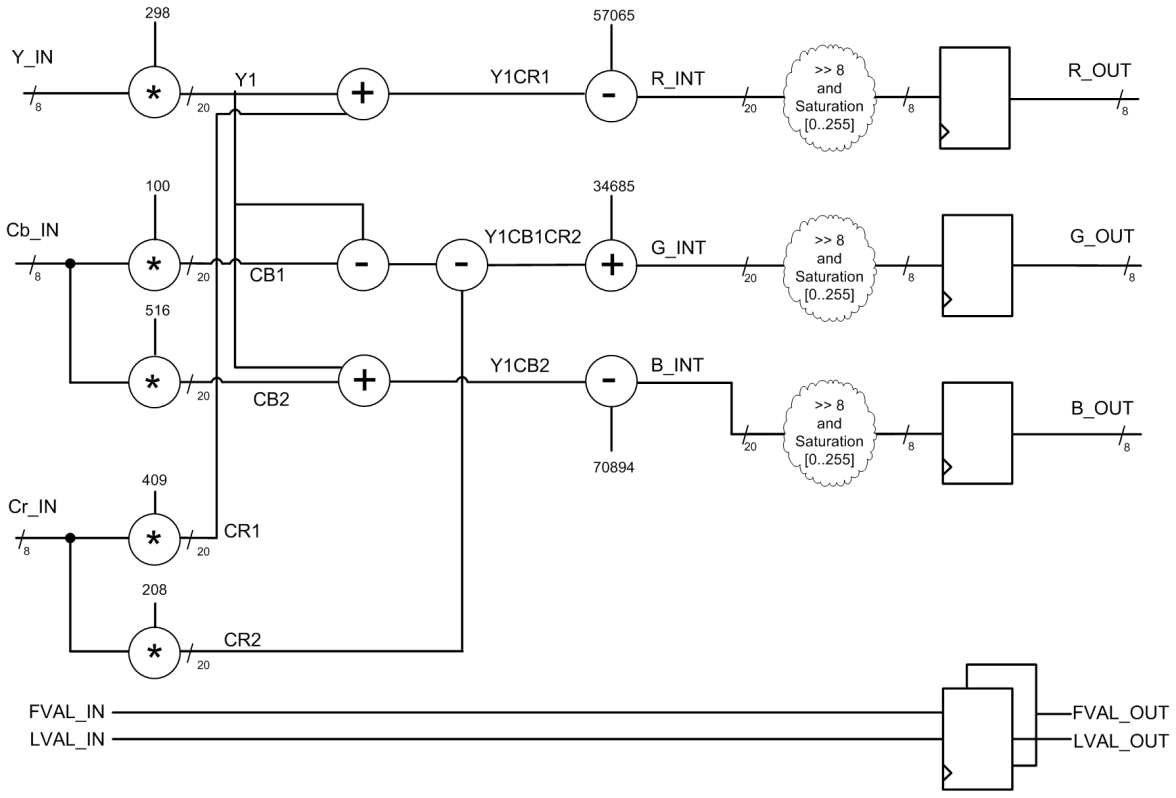

Abbildung 3.11: Datenpfad zur Konvertierung vom YCbCr- in das RGB-Format

Die Schnittstelle des Konvertermoduls wird in Listing 3.3 angegeben. Ein Blockschaltbild des Datenpfads entsprechend den Gleichungen (3.7) bis (3.9) ist in Abbildung 3.11 gegeben. Durch Rundungsfehler in den Gleichungen (3.7) bis (3.9) kommt es für bestimmte YCbCr-

Werte zu einem Verlassen des Wertebereichs [0..255] der einzelnen RGB-Komponenten. Aus diesem Grund werden die Ausgangssignale auf den Wertebereich [0..255] gesättigt. Dies geschieht in einem Schritt mit der Division durch 256 (vgl. Abbildung 3.11). Der Codeausschnitt in Listing 3.4 zeigt diesen Rechenschritt anhand des R-Wertes. Die G- und B-Werte werden auf identische Weise behandelt. Die Vektorbreiten der Signale für die Zwischenergebnisse (20 Bit) sind so gewählt, dass es durch die Rechnung nicht zu einem Over- bzw. Underflow kommen kann. Dadurch ist ein Verlassen des Zielwertebereichs [0,255] leicht zu erkennen.

Listing 3.4: Division durch 256 und Sättigung am Beispiel des R-Signals

```
1  ...
2  if r_int >= 2**16 then
3      r_out <= x"ff";
4  elsif r_int < 0 then
5      r_out <= x"00";
6  else
7      -- shift right by 8 bits
8      r_out <= std_logic_vector( r_int(15 downto 8) );
9  end if;
10 ...
```

3.7 Deinterlacing durch Zeilenverdopplung

Das Ansteuern des VGA-Monitors erfolgt mit einem Progressive Videostrom der vollen Auflösung von 640x480 Bildpunkten. Durch *Deinterlacing* wird der 60 Hz Interlaced Videostrom in einen 60 Hz Progressive Videostrom umgewandelt. Im Interlaced Videostrom sind für jeden Frame nur die Pixeldaten jeder zweiten Bildzeile vorhanden. Ziel des Deinterlacing ist das Auffüllen der “fehlenden” Bildzeilen.

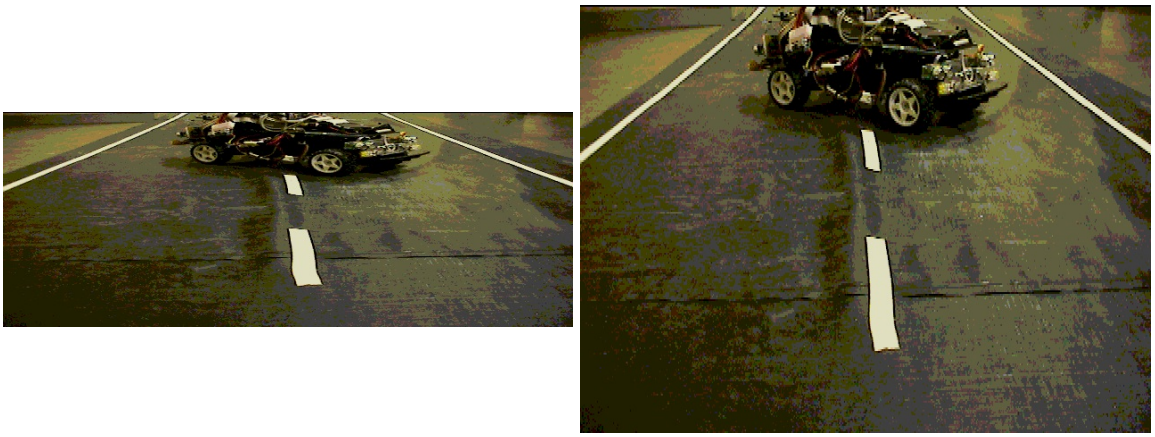

Abbildung 3.12: Ein Frame aus dem Videostrom vor (links) und nach dem *Deinterlacing* (rechts).

Eine mögliche Vorgehensweise zum Deinterlacing ist die Zeilenverdopplung. Dabei wird jede Bildzeile des Interlaced Videostroms zweifach ausgegeben. Weitere Varianten zum Deinterlacing sind eine FIR-Filterung der über mehrere Bildzeilen oder sogar über mehrere Frames. Die Zeilenverdoppelung wurde als Vorgehensweise für diese Arbeit gewählt, da sie den geringsten Speicherbedarf mit sich bringt: Es müssen zwei Bildzeilen zwischengespeichert werden. Bei den komplexeren FIR-Varianten stiege der Speicheraufwand dagegen auf mehrere vollständige Frames an (Jack, 2005).

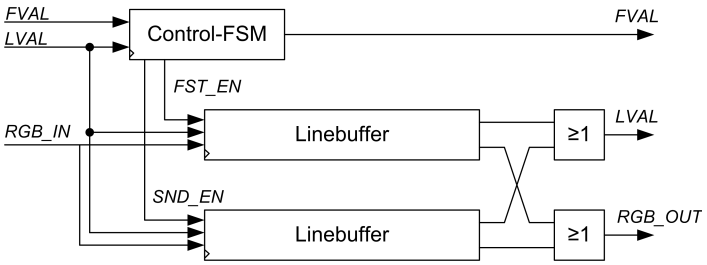

Abbildung 3.13: Blockdiagramm des Deinterlacing Moduls

Das Hardwaremodul arbeitet in zwei Schritten, die wechselweise ausgeführt werden (vgl. Abbildung 2.5):

1. Das Speichern einer Bildzeile im ersten der beiden Schieberegister.
2. Während die folgende Bildzeile im zweiten Schieberegister zwischengespeichert wird, wird die im ersten Schritt gespeicherte Bildzeile zweimal ausgegeben. Dies geht mit einer Erhöhung des Pixeltakts von 13,5 MHz auf 27 MHz einher (vgl. Abbildung 2.2). Im folgenden Arbeitsschritt wird erneut eine Bildzeile im ersten Schieberegister gespeichert, während der Inhalt des zweiten Registers zweimal ausgegeben wird.

Mit jeder neuen Bildzeile am Eingang startet die Steuer-FSM des Hardwaremoduls wechselweise die beiden Linebuffer mit dem Enable-Signal *FST_EN* bzw. *SND_EN* an. Die Ausgangssignale RGB_OUT und LVAL_OUT werden durch *OR*-Verknüpfung der Ausgänge der beiden Linebuffer gebildet (vgl. Abbildung 3.13).

Die Linebuffer werden intern von Rampenzählern gesteuert, die mit dem Enable-Signal gestartet werden. Anhand des Zählerstandes wird die Ausgabe bestimmt: Einspeichern der Bildzeile mit halber Taktrate, abwarten einer Blankingphase am Eingang, zweifaches Ausgeben der gespeicherten Zeile (mit dem dazu gehörigen *LVAL*-Signal) sowie des Blankings zwischen den Ausgabezeilen.

3.8 VGA Controller

Die VGA-Controller Komponente erzeugt die Synchronisationssignale *HSYNC* (horizontale Synchronisation) und *VSYNC* (vertikale Synchronisation) zur Ansteuerung eine Monitors. Beide Signale stellen sich als *low-pulse* während der jeweiligen Blankingphase dar (vgl. Abbildung 3.14). Die Timingparameter sind in Tabelle 3.3 zusammengefasst (vgl. Kirschke, 2009).

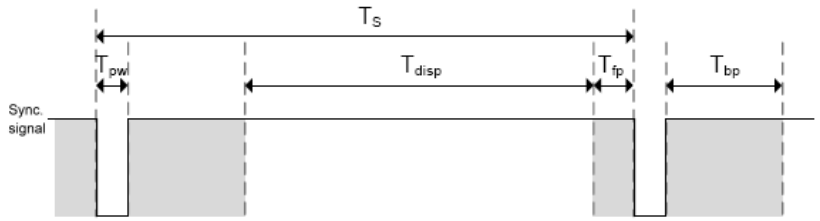

Abbildung 3.14: Timingbezeichner der VGA-Synchronisationssignale (Digilent, 2008).

Die Signale *HSYNC* und *VSYNC* werden anhand von Rampenzählern (*H_CNT* und *V_CNT*) sowie Komparatoren für die jeweiligen Zählerstände generiert. Mit der fallenden Flanke von *LVAL* wird das Ende der Pixeldaten einer Bildzeile angezeigt. Zu diesem Zeitpunkt startet der Rampenzähler *H_CNT* und wird mit jedem weiteren Takt inkrementiert. *HSYNC* ist *low*, wenn sich der Rampenzähler zwischen den Werten *FP_END* und *PULSE_END* befindet, sonst ist das Signal *high*. Mit dem erreichen von *DP_END* wird *H_CNT* zurück gesetzt. Das vertikale Synchronisationssignal *VSYNC* ist analog dazu implementiert: Er startet mit der fallenden Flanke von *FVAL* und wird mit jeder Bildzeile inkrementiert (vgl. Abbildung 3.15).

Tabelle 3.3: Timing der VGA-Synchronisationssignale

Parameter	Symbol	Vertikal Lines	Horizontal CLKs
Periodendauer	T_s	525	858
Display Time	T_{disp}	480	640
Pulse Width	T_{pw}	2	96
Front Porch	T_{fp}	11	31
Back Porch	T_{bp}	32	91

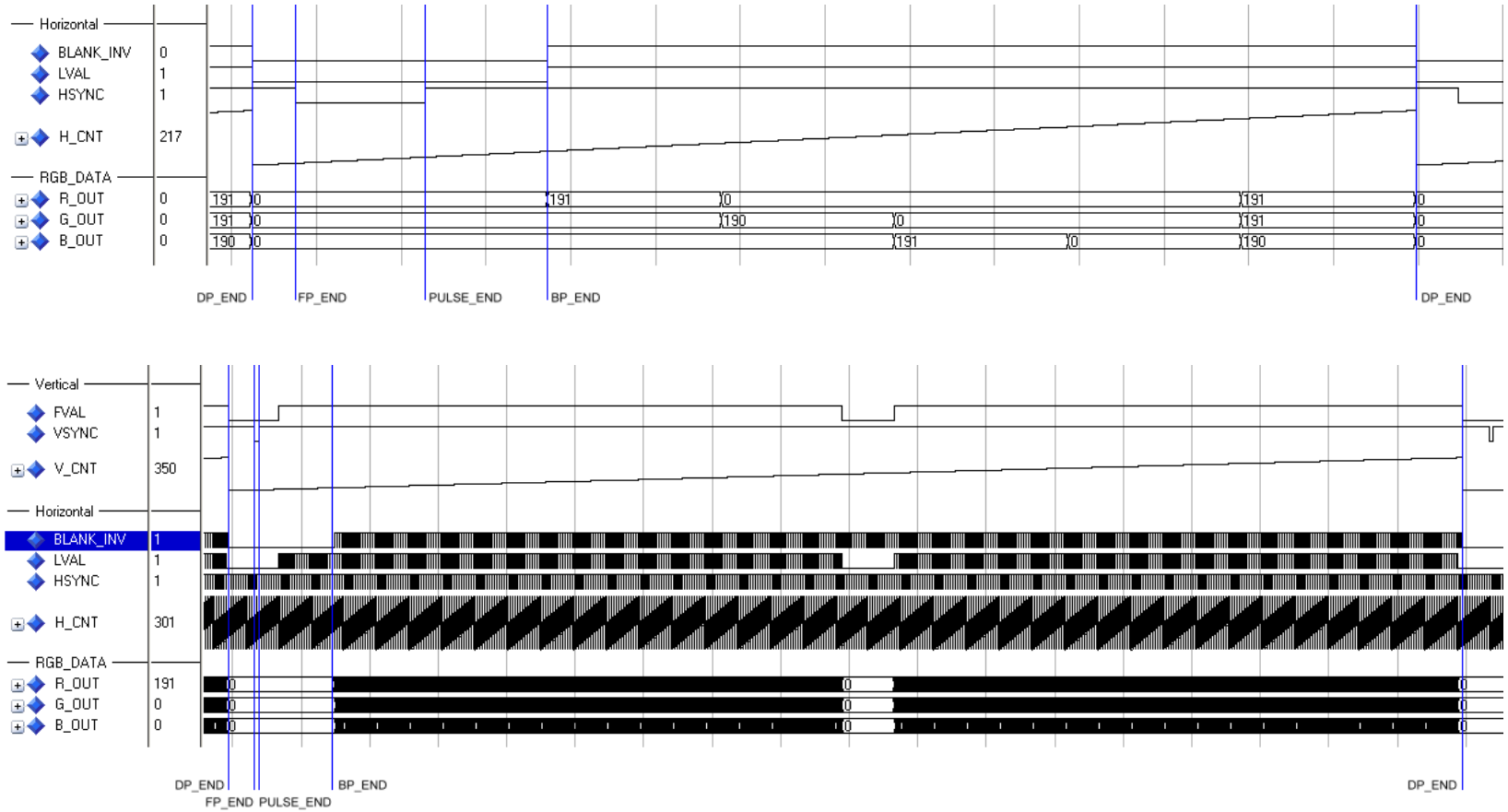

Abbildung 3.15: Timing der VGA-Synchronisationssignale

4 Kameraparametrierung zur Systemlaufzeit

Dieser Abschnitt gibt zunächst einen Überblick über das Microblaze Prozessorsystem und seine Komponenten. Darüber hinaus wird auf die Treibersoftware eingegangen, mit der die Kamera im laufenden System angesteuert wird.

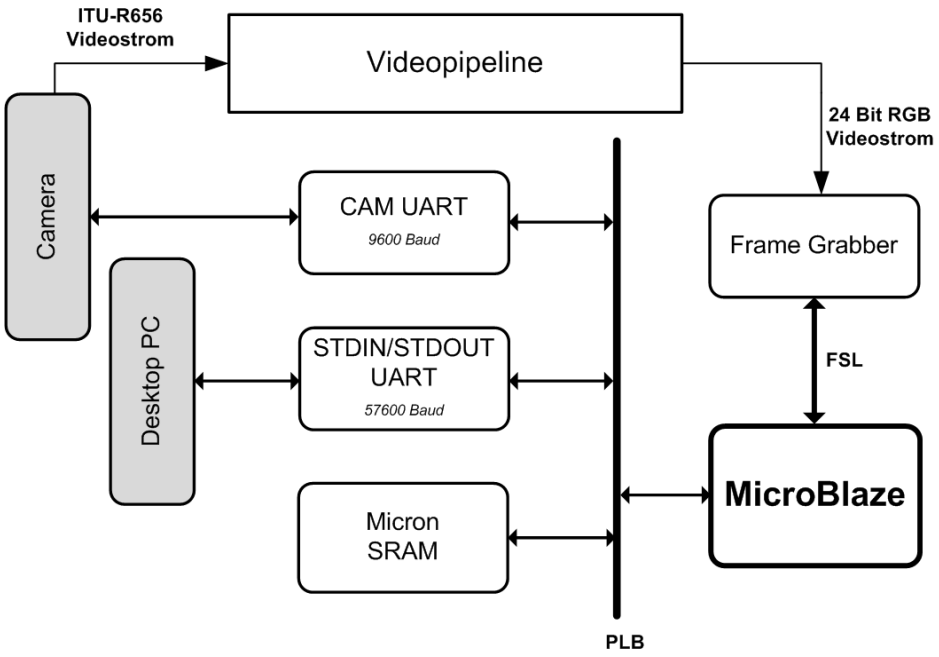

Abbildung 4.1: Komponenten des Microblaze Prozessorsystems

4.1 Das Microblaze Prozessorsystem

Der Microblaze Prozessor ist ein 32 Bit RISC Prozessor in Harvard Architektur. Als *Softcore* Prozessor erlaubt er ein hohes Maß an Konfiguration durch den Benutzer und bietet hohe Flexibilität bei der Auswahl der Peripheriekomponenten. In dieser Arbeit werden die folgenden Komponenten genutzt (vgl. Abbildung 4.1). Eine detaillierte Übersicht aller Komponenten und ihrer Busverbindungen gibt Abbildung 4.2:

- 2 *UART Lite* serielle Schnittstellen. Eine dieser Schnittstellen dient zur Übertragung von Steuerkommandos an die Kamera. Die andere UART-Schnittstelle wird als STDIN/STDOUT für den Prozessor genutzt. Über sie lassen sich Ausgaben an einen angeschlossenen PC versenden und dort (mit Hilfe von geeigneter Software, z.B. HyperTerminal) anzeigen. Eine Anwendung dieser Schnittstelle ist die Übertragung von gespeicherten Bildern aus dem Videostrom (vgl. Abschnitt 5).

- 16 Megabyte *Micron* SRAM des Nexys2 Boards. Das SRAM dient der Zwischenspeicherung von Bilddaten, zum Zweck der Ausgabe über STDOUT als Bitmap-Datei.

Die über den PLB angeschlossenen Komponenten werden *Memory-mapped* adressiert. Bei den seriellen Schnittstellen sind dies sowohl die Kontroll- als auch die Datenregister. Das SRAM erweitert den Hauptspeicher, wobei in der Software beim Speicherzugriff nicht zwischen den Speicherbereichen unterschieden werden muss. In den Zugriffszeiten ergeben sich jedoch auf Grund der Arbitrierungszeit des Busses Verzögerungen gegenüber der Verwendung des FPGA-internen Blockrams.

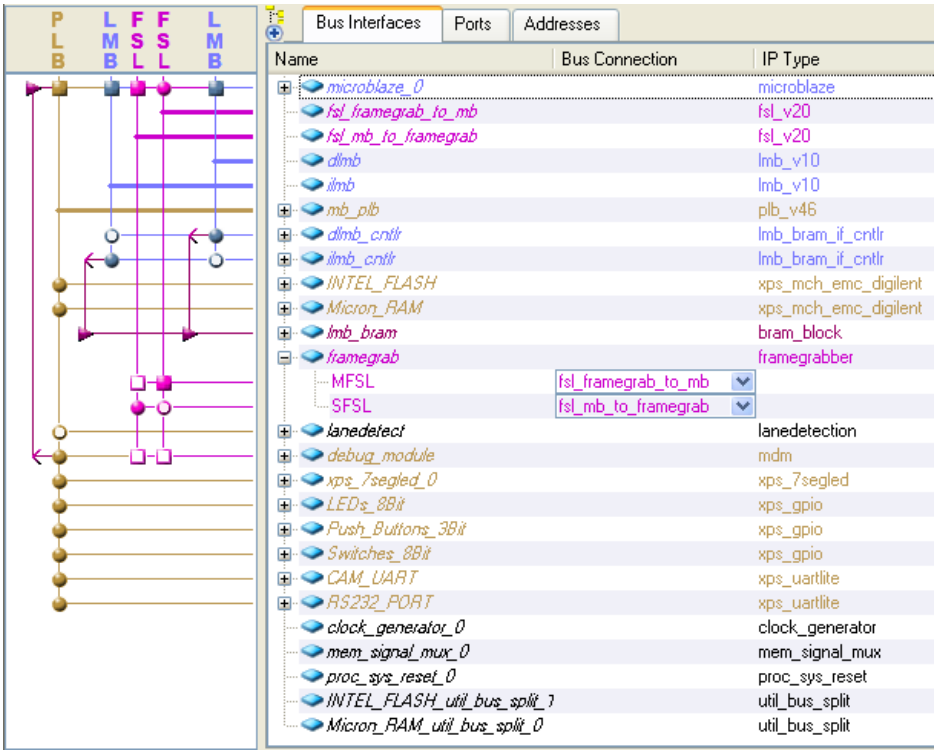

Abbildung 4.2: System Assembly View des Systems im Xilinx EDK

4.2 Treibersoftware zur Steuerung der Kamera

Die Kamera wird mit Kommandos des von Sony entwickelten *VISCA*-Protokolls gesteuert. In einem *VISCA* System gibt es einen *Controller*, im Falle dieser Arbeit den Microblaze Prozessor, sowie bis bis zu sieben Peripheriegeräte, welche über die serielle RS-232 Schnittstelle mit dem Controller verbunden sind. Die Datenübertragung der seriellen Schnittstelle erfolgt mit 9600 Baud, 8 Datenbits, 1 Startbit und ohne Paritätsbit (8N1) (vgl. Sony, 2006).

Mit den *VISCA* Kommandos lassen sich eine Vielzahl der Kamerafunktionen steuern, beispielsweise Zoom, Power On/Off, Fokus, Blende und viele weitere. Der Aufbau der Kommandosequenzen ist im Datenblatt der Kamera erläutert (Sony, 2006).

5 Bildspeicherung und -übertragung an den PC

Zu Debug- und Dokumentationszwecken wurde ein System zum speichern einzelner Frames aus dem Videostrom entwickelt. Es kann außerdem in späteren Ausbaustufen des Gesamtsystems genutzt werden um Bildverarbeitungsaufgaben in Software durchzuführen (vgl. Abschnitt 6.6). Das System führt die folgenden Arbeitsschritte durch (vgl. Abbildung 5.1):

1. Der Übertragungsprozess wird vom Microblaze Prozessor per Software initiiert. Der Prozessor sowie der PLB-Bus sind während des Speicher- und Übertragungsvorgangs ausgelastet und stehen nicht für weiter Aufgaben zur Verfügung.
2. Die Pixeldaten des nächsten vollständigen Frames werden im Micron SRAM des Nexys 2 Boards abgelegt. Das Speichern erfolgt dabei mit der Datenrate des Videostroms. Der Videostrom und weitere Verarbeitungsschritte im Hardwaresystem werden nicht beeinflusst.
3. Die Daten werden zu einem Microsoft-Bitmap kompatiblen Datenstrom kodiert. Dieser Datenstrom wird über die serielle Schnittstelle *STDOUT* an einen angeschlossenen PC versendet. Dort wird der Datenstrom direkt als Bitmap Datei gespeichert.

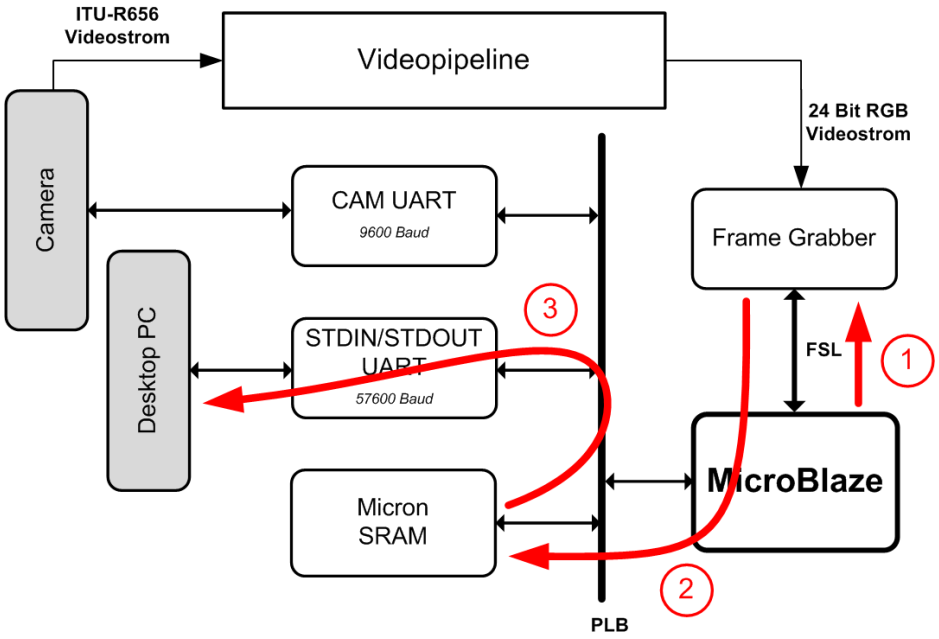

Abbildung 5.1: Arbeitsschritte und Datenfluss im Subsystem zur Bildspeicherung

5.1 Fast Simplex Link vom Framegrabber zum Microblaze

Aufgrund der hohen Datenrate des Videostroms (24 Bit Pixel mit einem Pixeltakt von 13,5 MHz) unterliegt der Vorgang des Ablegens der Daten im SRAM strikten Zeitbeschränkungen. Insbesondere der PLB-Bus mit seinen nicht garantierten Arbitrierungszeiten stellt dabei einen Engpass dar. Aus diesem Grund erfolgt die Datenübergabe aus der Videopipeline an den Prozessor über die FSL-Schnittstelle des Microblaze. Das SRAM ist über den PLB Bus mit dem Microblaze verbunden und steht als Hauptspeicher zur Verfügung. Für das Ablegen der Bilddaten ist die Datenrate des PLB hinreichend (Xilinx, 2007b).

Die Framegrabber Hardwarekomponente stellt die Verbindung zwischen dem Videodatenstrom und dem Microblaze Prozessor dar. Dazu hat er einerseits eine Schnittstelle zum Videostrom mit den Signalen *LVAL*, *FVAL* und *DATA*. *DATA* ist dabei bereits auf acht Bit Datenbreite reduziert (drei Bit Rot, drei Bit Grün und zwei Bit Blau). Die Farbauflösung ist damit identisch mit derjenigen des VGA-Ausgangs. Zum Anderen hat der Framegrabber jeweils ein Master- und Slave-FSL-Interface (vgl. Listing 5.1). Über das Slave-Interface bekommt er vom Microblaze das Signal, eine Bildübertragung zu starten. Mit dem Beginn des nächsten Frames werden die Daten des 8 Bit Videostroms über Schieberegister parallelisiert. Jeweils vier aufeinander folgende Pixel werden zu einem 32 Bit Paket zusammengefasst. Diese Pakete werden über das FSL Master-Interface an den Microblaze Prozessor übergeben (Xilinx, 2007a).

Listing 5.1: Schnittstellendefinition der Framegrabber Entity

```
1  entity framegrabber is
2      port (
3          clk             : in std_logic;
4          lval            : in std_logic;
5          fval            : in std_logic;
6          data            : in std_logic_vector(7 downto 0);
7          -- fsl interface ------------------------------------
8          fsl_clk         : in std_logic;
9          fsl_rst         : in std_logic;
10         -- slave interface
11         fsl_s_data      : in std_logic_vector(0 to 31);
12         fsl_s_control   : in std_logic;
13         fsl_s_exists    : in std_logic;
14         fsl_s_clk       : out std_logic;
15         fsl_s_read      : out std_logic;
16         -- master interface
17         fsl_m_full      : in std_logic
18         fsl_m_clk       : out std_logic;
19         fsl_m_write     : out std_logic;
20         fsl_m_data      : out std_logic_vector(0 to 31);
21         fsl_m_control   : out std_logic;
22     );
```

Der Framegrabber wird von einem Zustandsautomaten gesteuert. Dieser durchläuft die Zustände *Idle*, *WaitForVBlank*, *WaitForFrame* und *Transmit*. Die Zustände werden immer in dieser Reihenfolge durchlaufen.

Idle: Der Framegrabber befindet sich im Leerlaufzustand und wartet auf das Startsignal zur Datenübertragung. Diese wird dadurch ausgelöst, dass vom Microblaze ein beliebiges Datenwort auf den FSL-Bus geschrieben wird. Dadurch wird das Signal *FSL_S_Exists* gesetzt, was einen Zustandsübergang zu *WaitForVBlank* auslöst.

WaitForVBlank: Warten auf das nächste vertikale Blanking. Wenn der Videostrom sich in der vertikalen Blankingphase befindet, findet ein Zustandswechsel zu *WaitForFrame* statt.

WaitForFrame: Der Zustand wird solange gehalten, bis der Beginn des nächsten Frames am Dateneingang anliegt, was durch einen high-Pegel von *LVAL* angezeigt wird. Der Zustand wechselt zu *Transmit*.

Transmit: Mit dem Übergang nach *Transmit* wird ein Zähler auf Null gesetzt. Dieser zählt jeweils vier Pixel ab und setzt dann das Schreibsignal *FSL_M_Write* am Masterinterface. Dadurch werden die vier Pixel im Schieberegister in das FSL-FIFO gespeichert. Während des horizontalen Blankings wird der Zähler angehalten, wodurch auch die Datenübertragung unterbrochen ist. Mit dem Ende des Frames, angezeigt durch einen low-Pegel von *FVAL* am Eingang, springt der Zustandsautomat wieder in den *Idle*-Zustand, womit die Bildübertragung abgeschlossen ist.

Das Timing dieses Systems erfüllt die Anforderungen, die sich aus dem Zeitverhalten des PLB-Bus ergeben. Durch die Paketbildung aus jeweils vier Pixeln wird der Pixeltakt effektiv geviertelt. Zudem arbeitet der Microblaze Prozessor mit einer Taktrate von 50 MHz (der FSL Bus kann mit unterschiedlichen Taktraten an Master- und Slaveinterface betrieben werden). Dadurch ergibt sich ein Zeitraum von

$$\frac{50MHz}{\frac{13{,}5MHz}{4\text{Takte}}} > 14\text{Takte}. \tag{5.1}$$

14 Takte (bzw. $14 \cdot 20ns = 280ns$) sind ein hinreichender Zeitraum um mit dem Microblaze ein 32 Bit Datenwort von der FSL Schnittstelle zu lesen und dieses über den PLB im SRAM abzulegen (Xilinx (2007a), Xilinx (2007b)).

5.2 Bitmap Transfer an den PC

Das Speichern eines Frames wird per Software durch Aufruf der `getFrame()`-Methode (vgl. Listing 5.2) ausgelöst.

Diese Methode führ die folgenden Arbeitsschritte durch:

1. Erzeugen eines Bitmap-Dateiheaders `bmp_header`. Details zu dessen Aufbau finden sich in (Wikipedia, 2009).

2. Initiieren der Frameübertragung durch Schreiben auf den FSL-Bus mit `putfsl()`. Der genaue geschriebene Wert spielt dabei keine Rolle.
3. Speichern der Pixeldaten im `target` Array. Es werden jeweils 32 Bit = 4 Pixel vom FSL-Bus gelesen und Blockweise in das Zielarray geschrieben. `target` befindet sich in der gesonderten Speichersektion `.frame` des SRAM.
4. Versenden des Bitmapdatenstroms über die serielle Schnittstelle an STDOUT. Dazu wird zunächst der Header und anschließend die Nutzdaten verschickt. Es findet zudem eine Zeilenverdopplung von 240 auf 480 Zeilen statt (vgl. Abschnitt 3.7).

Listing 5.2: Speichern eines Frames, Kodierung als Bitmap-Datenstrom und versenden über die serielle Schnittstelle (STDOUT).

```
1  u8 frame[640*240] __attribute__ ((section(".frame")));
2
3  void getFrame(){
4      int rowcnt = 240, colcnt = 640;
5      int read_pixcnt = rowcnt * colcnt / 4; //4 Pixels transferred at a time.
6      int i, j, reccnt = 0;
7      int row,col;
8      int* target = (int*)frame;
9      char bmp_header[] = {
10         0x42, 0x4D,             // Magic Number
11         0x36, 0x10, 0x0E, 0x00, // Size of Bitmap
12         0x00, 0x00, 0x00, 0x00, // reserved, always 0
13         0x36, 0x00, 0x00, 0x00, // Offset of pixel data
14         0x28, 0x00, 0x00, 0x00, // Size of BitmapInfoHeader (=40d)
15         0x80, 0x02, 0x00, 0x00, // Width of Bitmap in Pixel
16         0x20, 0xFE, 0xFF, 0xFF, // Height of Bitmap in Pixel (see Wikipedia)
17         0x01, 0x00,             // planes (always 1)
18         0x18, 0x00,             // Color depth in bit per pixel
19         0x00, 0x00, 0x00, 0x00, // Compression
20         0x00, 0x00, 0x00, 0x00, // Size of Image, can be 0 if compression==0
21         0x00, 0x00, 0x00, 0x00, // horizontal resolution in pixel/meter
22         0x00, 0x00, 0x00, 0x00, // vertical resolution in pixel/meter
23         0x00, 0x00, 0x00, 0x00, // Color used (0 = standard)
24         0x00, 0x00, 0x00, 0x00};// ClrImportant
25
26     // Init transfer by writing to FSL
27     putfsl(42, 0);
28
29     // Save data in target array
30     for(reccnt = 0; reccnt < read_pixcnt; reccnt++){
31         getfsl(target[reccnt],0);
32     }
33
34     // Send header
35     for(i = 0;i < 54;i++) xil_printf("%c", bmp_header[i]);
36
37     // Send body
38     for(row=0; row<rowcnt; row++){
39         // Each line is sent twice ("deinterlace")
40         for(j = 0; j < 2; j++)
41             for(col = 0; col < colcnt; col++){
42                 u8 r, g, b;
43                 i = row*colcnt + col;
44                 r =  frame[i] & 0xE0;
45                 g = (frame[i] & 0x1C) << 3;
46                 b = (frame[i] & 0x03) << 6;
```

```
47                      xil_printf("%c%c%c",b,g,r);
48                  }
49              }
50          }
```

Auf der PC Seite wird der per serieller Schnittstelle empfangene Datenstrom unverändert in einer `.bmp` Datei gespeichert und kann dann mit beliebigen Bildbetrachtungswerkzeugen geöffnet werden.

6 Fahrspurerkennung

Dieser Abschnitt beschreibt ein Konzept zur Fahrspurerkennung auf Basis des entwickelten Hardware-/Softwaresystems. Mit einer Kette von Nachbarschaftsoperationen werden Messpunkte aus dem Bild extrahiert. Diese Messpunkte liegen auf der Fahrspurbegrenzung und beschreiben somit den Verlauf der Fahrbahn relativ zum Fahrzeug. Der Verlauf der Fahrspur über die Zeit wird als dynamisches System betrachtet. Zur Abschätzung des Systemzustandes, also des Fahrspurverlaufs zu einem gegebenen Zeitpunkt, wird ein geeignetes Kalman-Filter entworfen, welches die Fahrspur als Polynom dritten Grades modelliert. Die Schritte hierzu sind die Herleitung der geometrischen Zusammenhänge zwischen Bild- und Fahrzeugkoordinatensystem sowie die Entwicklung der System- und Messgleichungen des Systems. Die Komplexität des Kalman-Filters, im Sinne der Anzahl der notwendigen Rechenschritte, wird diskutiert und mit dem aktuell im Carolo-Cup Fahrzeug der HAW genutzten Verfahren, der statischen Polynomapproximation, verglichen. Schließlich wird ein Ausblick auf Möglichkeiten zur Weiterentwicklung des Verfahrens gegeben.

6.1 Bildvorverarbeitung mit Nachbarschaftsoperationen

Zum Tracken der Fahrspur werden Messpunkte im Bild gesucht, welche auf der Fahrspurbegrenzung (dem weißen Seitenstreifen) liegen. Durch Bildvorverarbeitungsschritte sollen möglichst gute Kandidaten für solche Bildpunkte gefunden werden. Eine geeignete Vorgehensweise hierfür sind Ketten von Nachbarschaftsoperationen, wie Kantenfilterung, Opening- und Closingoperationen.

Nachbarschaftsoperationen, stellen eine Abbildung eines Bildes (Quellbild) auf ein Zielbild dar. Dabei wird im Zielbild jedes Pixel durch eine gewichtete Summe seiner Nachbarpixel im Quellbild ersetzt. Sei f_q das Quellbild und w ein Filtermaske mit Seitenlänge $2a+1$, dann ist das Ergebnis der Nachbarschaftsoperation, das Zielbild f_z, wie folgt definiert (vgl. Abbildung 6.1):

$$f_z(x,y) = \sum_{s=-a}^{a} \sum_{t=-a}^{a} w(s,t) \cdot f_q(x+s, y+t) \tag{6.1}$$

Ein Beispiel für eine Nachbarschaftsoperation ist der Sobeloperator. Dabei wird jeder Bildpunkt auf den Gradienten des Grauwertes an seiner Bildposition abgebildet. Der Sobelfilter stellt somit eine Kantenfilter dar: Bereiche mit starken Grauwertänderungen, z.B. Kanten

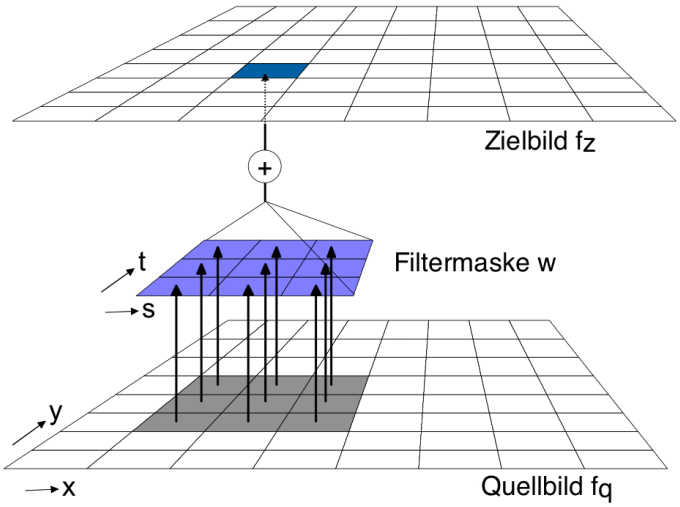

Abbildung 6.1: Nachbarschaftsoperation mit einer Filtermaske der Seitenlänge drei (Meisel, 2008).

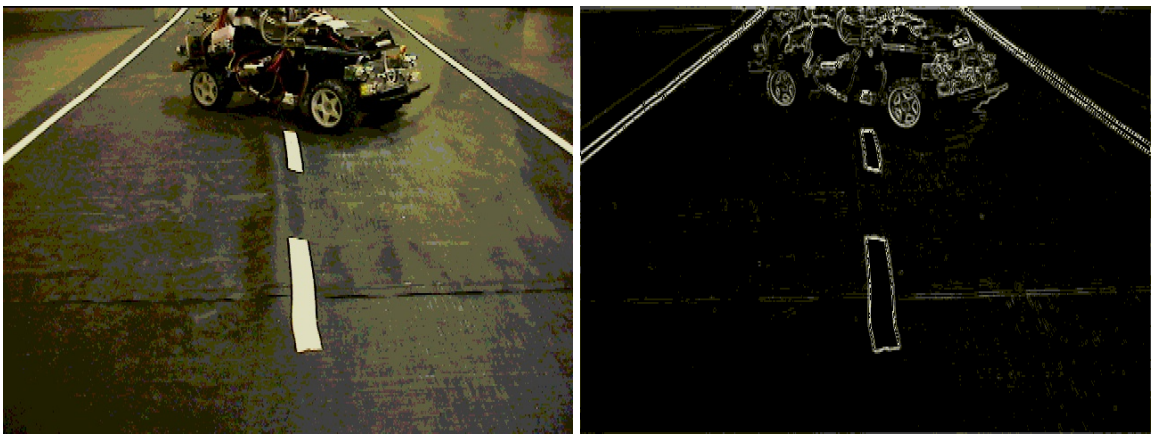

Abbildung 6.2: Bild im Original (links) und nach Anwendung der Sobelfilterung (rechts).

von Fahrbahnmarkierungen, werden hell dargestellt. Homogene Bildbereiche bleiben dagegen dunkel (vgl. Abbildung 6.2). Die beiden folgenden Filtermasken erzeugen Gradientenbilder in horizontaler (G_x) und vertikaler (G_y) Richtung (Nischwitz und Haberäcker, 2004):

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \text{ und } G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{6.2}$$

Der Gesamtgradient wird korrekterweise über die Gleichung $G = \sqrt{G_x^2 + G_y^2}$ berechnet.

Um die aufwändige Wurzelberechnung in Hardware zu vermeiden werden die beiden Komponenten im Falle dieser Arbeit addiert. Durch eine Sättigung auf ein Maximum von 255 wird der Wertebereich des 8 Bit Grauwertbildes eingehalten (vgl. Abbildung 6.3).

Zur Implementierung einer Nachbarschaftsoperation muss, bei einer Filtermaske der Seitenlänge drei, auf Pixelwerte aus drei Bildzeilen zugegriffen werden. Um dies im Betrieb der Pipeline zu gewährleisten werden jeweils zwei Zeilen in Schieberegistern zwischengespeichert. Über eine Registermatrix steht jeweils ein Nachbarschaftsbereich von neun Pixeln parallel zur Verfügung (vgl. Abbildung 6.4). Diese Komponente ist zunächst unabhängig von der gewünschten Nachbarschaftsoperation.

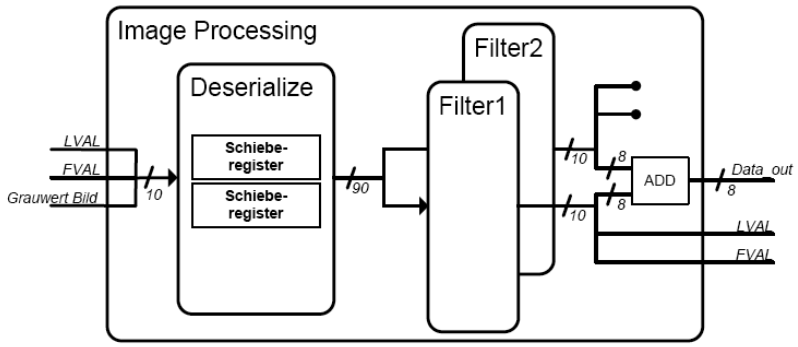

Abbildung 6.3: Blockschaltbild des Hardwaremoduls für die Bildverarbeitungsoperationen (Kirschke, 2009)

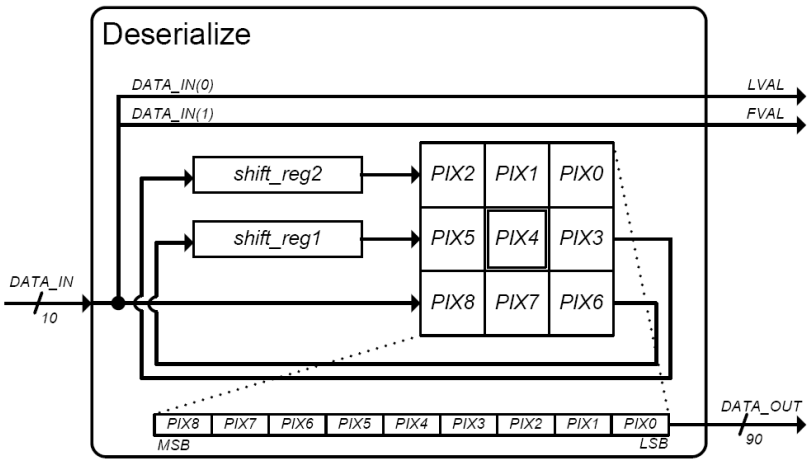

Abbildung 6.4: Blockschaltbild des Deserialisierungsmoduls zur Durchführung der Nachbarschaftsoperationen (Kirschke, 2009)

Die Filtermaske ist eine Hardwarekomponente, welche eine *Sum-of-Products* Rechnung implementiert. Die Elemente der Registermatrix werden mit ihren jeweiligen Koeffizienten multipliziert. Auf die Multiplikation folgt ein Reduktionsschritt in Form der Addition der Produkte.

Eine mögliche Vorgehensvariante zur Extraktion der Messpunkte ist, zunächst Lücken im Kantenbild (entstanden z.B. durch Rauschen) zu schließen. Dies kann durch Opening- bzw. Closingoperationen erfolgen. Lokale Maxima im Gradientenbild sind dann Kandidaten für Messpunkte (Risack u. a. (1998), Jenning (2008)).

6.2 Einführung in das Kalman-Filter

Ein Kalman-Filter ist ein Filter, welches den Zustand eines dynamischen Systems anhand von Messungen optimal abschätzt. Die Grundidee ist, dass wegen ungenauer Messungen der wahre Systemzustand immer unbekannt ist und daher abgeschätzt werden muss. Das Kalman-Filter macht sich bei der Schätzung des Zustandes zunutze, dass die Erwartungswerte sowohl für das System- als auch für das Messrauschen bekannt sind und liefert unter Berücksichtigung dieses Rauschens einen optimalen Schätzwert für den Systemzustand (Nischwitz und Haberäcker, 2004).

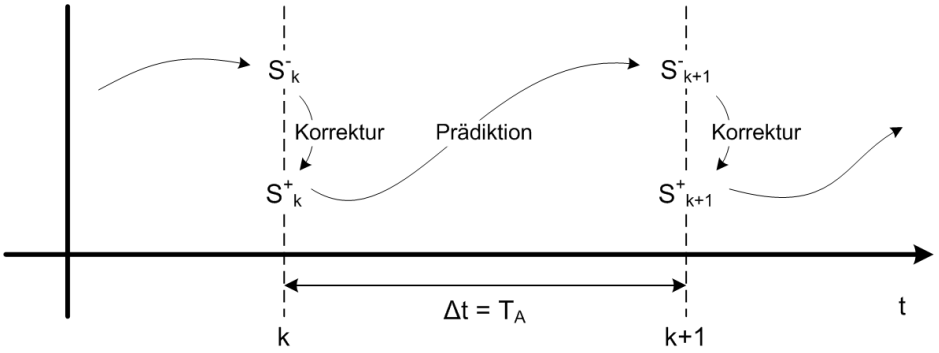

Abbildung 6.5: Berechnungssequenz des Kalman-Filters mit logischer zeitlicher Zuordnung der Zustände s_i sowie realer zeitlicher Zuordnung der Berechnungsvorgänge.

Ausgehend von einem bekannten Startzustand s_0 führt das Kalman-Filter abwechselnd die Arbeitsschritte *Prädiktion* und *Korrektur* durch (vgl. Abbildung 6.5).

Prädiktion: Nach einer Messung zum Zeitpunkt k ist der geschätzte Systemzustand s_k^+ bekannt. Ausgehend von diesem Zustand wird auf Basis eines Systemmodells der voraussichtliche Folgezustand s_{k+1}^- (das "$-$" bedeutet hier *a priori*) prädiziert.

Korrektur: Zum Zeitpunkt $k+1$ wird eine weitere Messung durchgeführt. Anhand dieser Messung und der Prädiktion s_{k+1}^- wird *a posteriori* auf den wahrscheinlichen Systemzustand s_{k+1}^+ zum Zeitpunkt $k+1$ geschlossen. In diese Korrektur fließen die *a priori* Schätzung sowie die Messung unterschiedlich gewichtet ein. Die Gewichtung hängt von der Varianz der Prädiktion und der Messwerte ab. Je kleiner die Varianz eines Wertes, desto mehr wird diesem Wert vertraut und er fließt zu einem größeren Anteil in die Schätzung ein.

Wegen dieser rekursiven Arbeitsweise muss das Kalman-Filter stets nur den aktuellen Systemzustand speichern. Daraus ergibt sich ein vergleichsweise geringer Speicherbedarf sowie ein geringerer Rechenaufwand gegenüber Filtern welche mehrere Messwerte für eine Zustandsschätzung verwenden. Aus diesem Grund ist das Kalman-Filter besonders für den Einsatz in Echtzeitsystemen geeignet.

Ein zeitdiskretes dynamisches System wird durch seine Systemgleichung und seine Messgleichung beschrieben. Die Systemgleichung beschreibt den Zustandsübergang des Systemzustandes s_k zum Zeitpunkt k auf den Systemzustand s_{k+1} zum Zeitpunkt $k+1$. Die Systemmatrix A bildet den Systemzustand auf den Folgezustand ab, ohne dabei äußere Einflüsse u_k, wie etwa Regeleingriffe, zu berücksichtigen. Solche äußeren Einflüsse werden durch die Eingabematrix B auf den Systemzustand abgebildet. Die Zufallsvariable w_k repräsentiert das Systemrauschen. Daraus ergibt sich die folgende Systemgleichung, welche auch den Prädiktionsschritt des Kalman-Filters beschreibt:

$$s_{k+1} = As_k + Bu_k + w_k. \tag{6.3}$$

Der Zustand s eines Systems ist im allgemeinen nicht direkt messbar. Stattdessen sind nur Messungen beobachtbar, welche sich über die Messgleichung aus dem Systemzustand ergeben. Der Systemzustand s_k wird dabei über die Messmatrix H auf die Messung y_k abgebildet. Auf die Messung wirkt sich zudem das Messrauschen über die Zufallsvariable v_k aus.

$$y_k = Hs_k + v_k \tag{6.4}$$

Für den Korrekturschritt des Kalman-Filters wird ein Zustand s_k^+ berechnet. In diese Berechnung fließt einerseits die Messung y_k und andererseits der prädizierte Systemzustand s_k^- ein. Die Differenz von Messwert y_k und prädiziertem Messwert $H \cdot s_k^-$ ist ein Maß für die Genauigkeit der Prädiktion. Ist diese Differenz klein wird der Prädiktion s_k^- mehr vertraut und diese fließt zu einem größeren Anteil in die Korrektur ein. Ist die Differenz dagegen groß, wirkt sich die Messung y_k stärker aus.

$$s_k^+ = s_k^- + K_k(y_k - Hs_k^-) \tag{6.5}$$

Die Kalman-Verstärkung K_k wird dazu so gewählt, dass die Schätzfehlerkovarianz P_k^+ der korrigierten Schätzung s_k^+ minimal wird. Eine mögliche Form von K ist im folgenden gegeben (vgl. Nischwitz und Haberäcker, 2004).

$$K_k = \frac{P_k^- H^T}{H P_k^- H^T + R} \tag{6.6}$$

R ist dabei die Messfehlerkovarianz. Anhand zweier Grenzfälle lässt sich die Auswirkung von Gleichung (6.6) verdeutlichen:

- Geht die Messfehlerkovarianz R gegen Null, so wird K maximal. Dies bedeutet, dass bei einer sehr genauen Messung, diese zu einem großen Anteil in die Korrektur einfließt.
- Geht andererseits die Schätzfehlerkovarianz P_k^- der Prädiktion s_k^- gegen Null, so geht auch K gegen Null. Bei einer sehr genauen Prädiktion wird dieser somit vertraut und die Messung hat nur einen minimalen Einfluss.

6.3 Perspektivische Transformation der Bildkoordinaten

Der Verlauf der Fahrspur wird als Polynom im Fahrzeugkoordinatensystem modelliert, die Messungen für das Kalman-Filter werden jedoch im Bildkoordinatensystem vorgenommen. Zur späteren Herleitung der Messgleichung des Kalman-Filters wird hier zunächst der geometrische Zusammenhang zwischen den Koordinaten der Messpunkte im Bild und dem korrespondierenden Punkt im Fahrzeugkoordinatensystem hergeleitet.

Das Fahrzeugkoordinatensystem wird aufgespannt von den drei orthogonalen Achsen x, y und z. Die z-Achse entspricht dabei der Fahrzeuglängsachse, wobei positive z-Koordinaten vor dem Fahrzeug liegen. Die y-Achse entspricht der Querachse des Fahrzeugs mit positiven Koordinaten rechts der Fahrzeugmittellinie. Die vertikale x-Achse entspricht der Fahrzeughochachse. Die Bildebene wird in horizontaler Richtung aufgespannt von der y-Achse (des Fahrzeugkoordinatensystems) sowie vertikal von einer Achse welche der x-Achse, gekippt um den Winkel φ_2, entspricht (vgl. Abbildungen 6.6 und 6.7).

Im folgenden wird angenommen, dass die y-Koordinate der Kamera Null ist. Außerdem wird angenommen, dass die Fahrspur in einer Ebene liegt, so dass die x-Fahrzeugkoordinaten aller Messpunkte gleich Null sind. Alle Längenbezeichner haben die Einheit Meter.

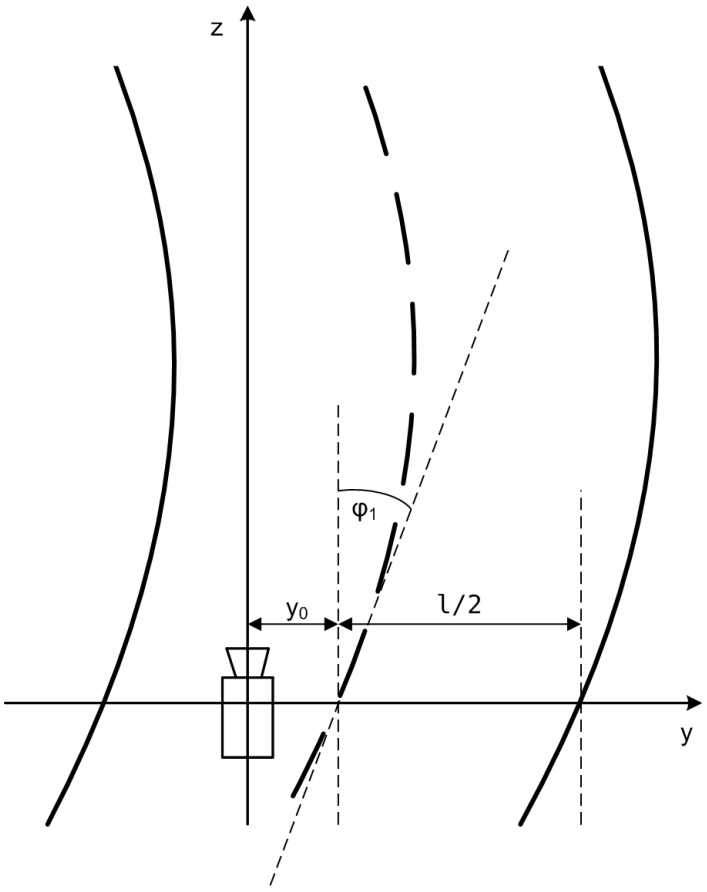

Abbildung 6.6: Draufsicht des Fahrzeugkoordinatensystems

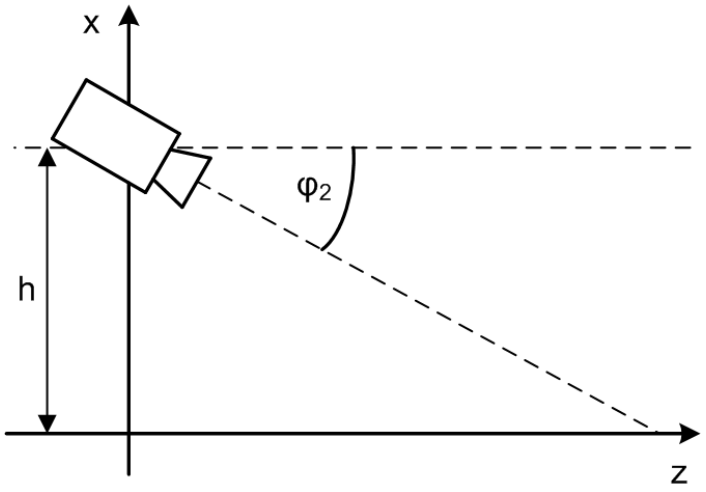

Abbildung 6.7: Seitenansicht des Fahrzeugkoordinatensystems

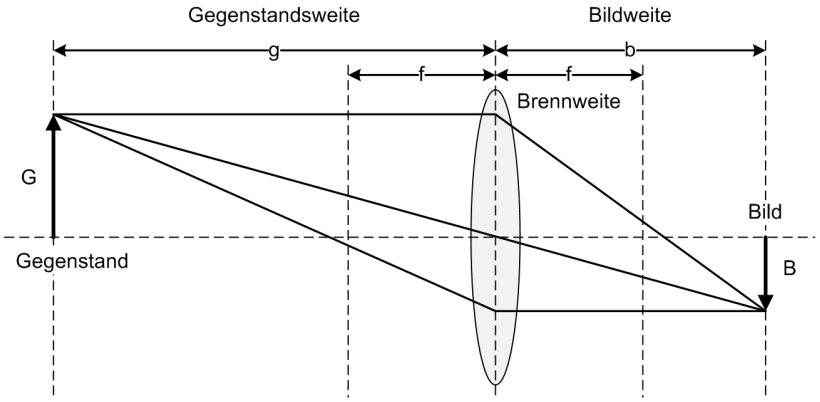

Abbildung 6.8: Bezeichnungen im Linsensystem

In einem Linsensystem, wie es die Kamera ist, ergibt sich der Zusammenhang zwischen Bildgröße B und Gegenstandsgröße G unter Berücksichtigung der Bildweite b sowie der Gegenstandsweite g anhand des geometrischen Strahlensatzes (vgl. Abbildung 6.8).

$$\frac{B}{G} = \frac{b}{g} \tag{6.7}$$

Zunächst wird der Zusammenhang zwischen horizontaler Bildgröße B_y und horizontaler Gegenstandsgröße G_y betrachtet. In diesem Fall ist die Gegenstandsweite eines Messpunktes

seine z-Koordinate. Die Gegenstandsgröße entspricht seiner y-Koordinate. Löst man Gleichung (6.7) nach der Bildgröße auf ergibt sich

$$B_y = \frac{b \cdot G_y}{g_z}. \tag{6.8}$$

Der Pixelindex y_b in der Bildzeile hängt neben der horizontalen Bildgröße B_y auch von der horizontalen Bildelementdichte k_y der Kamera mit Einheit $[k] = Pixel/m$ ab. Um Pixelindizes relativ zum linken Bildrand zu erhalten fließt der horizontale Pixelindex c_y der Bildmitte als Offset ein (Risack u. a., 1998, Gl. (1)).

$$y_b = \frac{b \cdot k_y}{g_z} G_y + c_y \tag{6.9}$$

Da alle Messpunkte in der Ebene $x = 0$ liegen, wird die Gegenstandsgröße in vertikaler Richtung G_x von der Montagehöhe h sowie dem Kippwinkel φ_2 der Kamera beeinflusst. Zwei Grenzfälle machen den Zusammenhang deutlich (vgl. Abbildung 6.7):

- Bei einer horizontal ausgerichteten Kamera ($\varphi_2 = 0$) ist die Gegenstandsgröße gleich der inversen Montagehöhe $-h$. Die horizontale Entfernung, also die z-Koordinate des Messpunktes spielt dann keine Rolle.
- Ist die Kamera dagegen senkrecht nach unten ausgerichtet ($\varphi_2 = \frac{\pi}{2}$) ist die Gegenstandsgröße gleich der horizontalen Entfernung des Messpunktes. In diesem Fall hat die Montagehöhe keinen Einfluss auf die Gegenstandsgröße.

Diese Eigenschaften werden von der Sinus- bzw. Kosinusfunktion erfüllt. Nimmt man zudem φ_2 als klein an, ist die Gegenstandsweite (entlang der Kameralängsachse) annähernd gleich der z-Koordinate des Messpunktes. Die vertikale Bildgröße B_x ergibt sich damit als

$$B_x = \frac{b \cdot (\sin(\varphi_2) g_z - \cos(\varphi_2) h)}{g_z}. \tag{6.10}$$

Die Annahme, dass φ_2 klein ist, erlaubt weitere Vereinfachungen. Für kleine Argumente verhält sich der Sinus wie eine Identitätsabbildung, d.h. sein Funktionswert ist in diesem Fall annähernd gleich seinem Argument. Der Kosinus verhält sich für kleine Argumente annähernd wie eine Konstante mit Wert Eins. Damit, und analog zum horizontalen Fall, ergibt sich der Zeilenindex (relativ zum oberen Bildrand) eines Messpunktes als

$$x_b = \frac{b \cdot k_x}{g_z} (\varphi_2 g_z - h) + c_x \tag{6.11}$$

mit der vertikalen Bildelementdichte k_x der Kamera sowie dem Zeilenindex c_x der Bildmitte.

6.4 Herleitung der System- und Messgleichungen

Der Systemzustand des Kalman-Filters ist im Falle dieser Arbeit die Lage und Form der Fahrspur im Fahrzeugkoordinatensystem, repräsentiert durch ein Polynom dritten Grades. In der Literatur wird die Fahrspur typischerweise als ein Polynom im dreidimensionalen Raum angenähert. Da die Fahrspur im Carolo-Cup jedoch immer in einer Ebene liegt, kann sie hier vereinfachend als ein Polynom im zweidimensionalen Raum modelliert werden (vgl. Carolo-Cup (2009), Risack u. a. (1998)).

In der Folge kann, gegenüber der Literatur, auf einige Komponenten des Zustandsvektors verzichtet werden. Diejenigen Komponenten welche den vertikalen Verlauf der Fahrbahn beschreiben werden als konstant angenommen. Der Neigungswinkel φ_2 der Kamera zur Fahrbahn wird hier ebenfalls als konstant angenommen. Durch Bewegungen des Fahrzeugs auf seiner Federung unterliegt dieser Winkel Veränderungen, wodurch die Abbildungsgeometrie zwischen Welt- und Bildkoordinatensystem Schwankungen unterliegt. Das Carolo-Cup Fahrzeugs der HAW ist jedoch mit einer Hartgummifederung ausgerüstet, welche Schwingungen des Fahrzeugs gering hält. Die Breite l der Fahrbahn wird ebenfalls als konstant angenommen. Im Carolo-Cup ist dies laut Regelwerk der Fall, aber auch auf vielen realen Straßen, wie etwa Autobahnen, ist diese Annahme gültig. Diese konstanten Größen sind nicht Teil des Zustandsvektors sondern fließen direkt in die System- und Messgleichungen ein.

Der Zustandsvektor s ist eine Repräsentation des Fahrspurpolynoms $y(z)$ als Vektor seiner Ableitungen an der Stelle $z = 0$. Diese Repräsentation ist äquivalent zu einer Repräsentation durch die Koeffizienten des Polynoms und beide Darstellungen lassen sich in die jeweils andere umrechnen (vgl. Gleichung (6.13)). Die Darstellung mit Ableitungen vereinfacht, wie im folgenden gezeigt wird, die Systemmatrix A. Zudem ist diese Darstellung im Kontext eines auf die Fahrspurerkennung folgenden Auswertungsschrittes aussagekräftiger. So lässt sich beispielsweise die Fahrzeuglage auf der Fahrbahn direkt aus den Werten $y(0)$ (lateraler Versatz zur Fahrspurmitte) und $y'(0)$ (Gierwinkel) ablesen. Im konkreten Fall des Polynoms dritter Ordnung ergibt sich ein vierdimensionaler Zustandsvektor mit den folgenden Komponenten (vgl. Abbildung 6.6):

$$s = \begin{pmatrix} y(0) \\ y'(0) \\ y''(0) \\ y'''(0) \end{pmatrix} = \begin{pmatrix} y_0 \\ \varphi_1 \\ c_0 \\ c_1 \end{pmatrix} \tag{6.12}$$

y_0 **Lateraler Versatz:** Der seitliche Abstand der Fahrspurmitte zum Fahrzeug. Bei einem positiven Wert befindet sich die Fahrspurmitte (in Fahrtrichtung) rechts vom Fahrzeug.

φ_1 **Gierwinkel:** Der Winkel, im Bogenmaß, zwischen der Längsachse (z-Achse) des Fahrzeugs und der Fahrspurtangente an der Stelle $z = 0$. Der korrekte Wert der ersten Ableitung des Fahrspurpolynoms $y'(0)$ ist $\sin(\varphi_1)$. Da sich der Sinus für kleine Winkel (etwa $\varphi_1 < \frac{\pi}{10}$) annähernd wie eine Identitätsabbildung verhält kann der Gierwinkel vereinfachend direkt in den Zustandsvektor aufgenommen werden. Ein positiver Wert bedeutet eine nach rechts geneigte Fahrspurtangente.

c_0 **Fahrspurkrümmung:** Die Änderungsrate des Gierwinkels. Ein positiver Wert korrespondiert zu einer nach rechts gekrümmten Fahrspur.

c_1 **Änderungsrate der Krümmung:** Bei einem positiven Wert nimmt die horizontale Krümmung der Fahrspur mit der Entfernung zu.

Anhand der Komponenten des Zustandsvektors lässt sich die Fahrspur im Fahrzeugkoordinatensystem als Polynom dritten Grades eindeutig beschreiben. Der Zustandsvektor beschreibt ein Polynom entlang der Fahrspurmitte. Daher wird zum Funktionswert noch ein Offset von der halben Fahrspurbreite $\frac{l}{2}$ addiert. Dabei ist der Offset für die rechte Fahrspurbegrenzung positiv und für die linke Begrenzung negativ.

$$y(z) = \pm\frac{l}{2} + y_0 + \varphi_1 z + \frac{1}{2}c_0 z^2 + \frac{1}{6}c_1 z^3 \tag{6.13}$$

Die Systemmatrix A beschreibt den Systemübergang von einem Frame (Abtastpunkt) zum nächsten mit zeitlichem Abstand $T_A = \text{Bildrate}^{-1}$. Das Fahrzeugmodell berücksichtigt bei der Berechnung des Folgezustandes allein die Geschwindigkeit des Fahrzeugs. Diese und die Abtastperiode T_A gehen somit in die Systemmatrix A ein. Für kleine Lenkwinkel und kleine Abtastperioden ist der Einfluss des Lenkwinkels auf den Folgezustand vernachlässigbar. Daher fließt er hier nicht in das Fahrzeugmodell ein. Die Fahrzeugbewegung wird somit als geradlinig entlang der z-Achse des Fahrzeugkoordinatensystem angenommen. Während der Zeit T_A zwischen zwei Messpunkten k und $k+1$ legt das Fahrzeug bei Geschwindigkeit v den Weg $\Delta z = v \cdot T_A$ zurück. Der Folgezustand kann dann prädiziert werden als

$$s_{k+1} = \begin{pmatrix} y_k(v \cdot T_A) \\ y'_k(v \cdot T_A) \\ y''_k(v \cdot T_A) \\ y'''_k(v \cdot T_A) \end{pmatrix} = a(s_k). \tag{6.14}$$

Diese Abbildung a ist nicht-linear, weshalb es keine Matrix A gibt, die zu ihr äquivalent ist. Daher wird das Fahrspurpolynom an der Stelle $x_0 = 0$ linearisiert, entsprechend dem Zusammenhang

$$f(x) \approx f(x_0) + \left.\frac{df}{dx}\right|_{x_0} \cdot (x - x_0). \tag{6.15}$$

Daraus ergibt sich der vereinfachte Zusammenhang für den Zustandsübergang:

$$s_{k+1} = \begin{pmatrix} y_{0k} + \varphi_{1k} \cdot v \cdot T_A \\ \varphi_{1k} + c_{0k} \cdot v \cdot T_A \\ c_{0k} + c_{1k} \cdot v \cdot T_A \\ c_{1k} \end{pmatrix} = A \cdot \begin{pmatrix} y_{0k} \\ \varphi_{1k} \\ c_{0k} \\ c_{1k} \end{pmatrix} \tag{6.16}$$

Die Systemmatrix A ist demnach

$$A = \begin{pmatrix} 1 & v \cdot T_A & 0 & 0 \\ 0 & 1 & v \cdot T_A & 0 \\ 0 & 0 & 1 & v \cdot T_A \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{6.17}$$

Die Eingabematrix B bildet einen Regeleingriff u_k auf seinen Einfluss auf den Systemzustand ab. Komponenten des Vektors u könnten beispielsweise Beschleunigungen oder Änderungen des Lenkwinkels sein. In der Regel sind es Ableitungen derjenigen Größen, welche in die Systemmatrix A einfließen. Kriterium für die Zuteilung einer Größe zur Matrix A oder B ist, dass die Matrix A den Zustandsübergang für den Fall beschreibt, dass keine äußeren Einflüsse wirken. So ist beispielsweise die Geschwindigkeit konstant, wenn keine beschleunigende Kraft auf das Fahrzeug wirkt (die Beschleunigung ist die erste Ableitung der Geschwindigkeit). Für die weitere Diskussion wird davon ausgegangen, dass Regeleingriffe nicht vorhanden bzw. vernachlässigbar klein sind. Die Matrix B spielt daher in den weiteren Betrachtungen keine Rolle, bzw. kann als Null angenommen werden (Risack u. a., 1998).

Die Abbildung des Systemzustands s_k auf eine Messung y_k geschieht über die Messmatrix H. Die Messung ist dabei eine Menge von Bildkoordinaten (x_b, y_b) die auf der Fahrspurbegrenzung liegen. Da die Messung für jeden Zeitschritt in den gleichen Bildzeilen durchgeführt wird enthält der Messvektor y nur die jeweiligen horizontalen y_b Koordinaten. Der Zusammenhang zwischen Messwert und Systemzustand ergibt sich dabei aus den Gleichungen (6.9) und (6.13).

$$y_b = \pm\frac{b \cdot k_y \cdot l}{2z} + \frac{b \cdot k_y}{z} y_0 + b \cdot k_y \cdot \varphi_1 + \frac{b \cdot k_y \cdot z}{2} c_0 + \frac{b \cdot k_y \cdot z^2}{6} c_1 + c_y \tag{6.18}$$

Aus dem bekannten Zeilenindex x_b des Bildpunktes lässt sich durch Auflösen von Gleichung (6.11) nach g_z die entsprechende z-Koordinate im Fahrzeugkoordinatensystem berechnen.

Da die Fahrspur in einer Ebene liegt ist dieser Wert unabhängig vom Systemzustand und daher konstant für jede Bildzeile x_b.

$$z = \frac{-h}{\frac{x_b - c_x}{b \cdot k_x} - \varphi_2} \tag{6.19}$$

Bei n Messpunkten ergibt sich die Messmatrix H als $n \times 4$-Matrix. Die Werte z_i hängen dabei von den Zeilenindizes x_b der Messpunkte ab. Der Zustandsvektor s beschreibt den Verlauf der Fahrspur*mitte*. Um daraus die Messpunkte auf der Fahrbahnbegrenzung zu bestimmen wird ein konstanter Offset o auf den Messvektor aufaddiert. Dadurch werden die Messpunkte in horizontaler Richtung verschoben.

$$y_k = H \cdot s_k + o = \begin{pmatrix} \frac{b \cdot k_y}{z_1} & b \cdot k_y & \frac{b \cdot k_y \cdot z_1}{2} & \frac{b \cdot k_y \cdot z_1^2}{6} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{b \cdot k_y}{z_n} & b \cdot k_y & \frac{b \cdot k_y \cdot z_n}{2} & \frac{b \cdot k_y \cdot z_n^2}{6} \end{pmatrix} \cdot \begin{pmatrix} y_{0k}^- \\ \varphi_{1k}^- \\ c_{0k}^- \\ c_{1k}^- \end{pmatrix} + \begin{pmatrix} c_y \pm \frac{b \cdot k_y \cdot l}{2 z_1} \\ \vdots \\ c_y \pm \frac{b \cdot k_y \cdot l}{2 z_n} \end{pmatrix} \tag{6.20}$$

Damit sind sowohl die System- als auch die Messgleichung linear, so dass die Zustandsschätzung mit einem einfachen Kalman-Filter erfolgen kann. Die Kovarianzmatrizen P und R werden als konstant angenommen, wodurch sich eine konstante Kalman-Verstärkung K ergibt (Risack u. a., 1998).

6.5 Komplexitätsbetrachtung

Dieser Abschnitt vergleicht den Rechenaufwand des Kalman-Filters mit dem bisher im Carolo-Cup verwendeten Polynomapproximationsverfahren. Dabei zeigt sich, dass für das Kalman-Filter, verglichen mit dem gegenwärtig im Carolo-Cup Fahrzeug der HAW verwendeten Approximationsverfahren, ein mindestens um dem Faktor Zwei höherer Rechenaufwand entsteht. Dies bezieht alle bereits diskutierten Vereinfachungen des Kalman-Filters mit ein (vgl. Tabelle 6.1). Betrachtet wird hier die Anzahl der zur Berechnung notwendigen Multiplikationen, da im FPGA zunächst nur eine begrenzte Anzahl an Multiplizierern zur Verfügung steht. Es können zwar weitere Multiplizierer synthetisiert werden, womit jedoch ein hoher Ressourcenaufwand verbunden ist und die Gefahr besteht, dass lange Laufzeitpfade gebildet werden.

Die Anzahl der skalaren Multiplikationen bei der Berechnung des Produktes zweier Matrizen hängt von der deren Zeilen- und Spaltenanzahl ab. Wenn A eine $p \times q$ Matrix und B eine $q \times r$ Matrix sind, dann sind zur Berechnung von AB im allgemeinen Fall pqr skalare Multiplikationen notwendig (Cormen u. a., 2001).

Tabelle 6.1: Vergleich des Rechenaufwands von Kalman-Filter und statischer *least squares* Polynomapproximation bei n Messpunkten.

Verfahren	Anzahl Multiplikationen
Normalengleichung	$4n$
Kalman-Filter	$8n+3$

In früheren Projekten zur Fahrspurerkennung im Rahmen des Carolo-Cup wird ein statisches Verfahren zur Fahrspurerkennung genutzt. Das Fahrspurpolynom wird anhand der Messpunkte mit einem *least squares* Approximationsverfahren bestimmt. Aus dem Zusammenhang der Messgleichung (6.4) lässt sich die Normalengleichung $H^T Hs = H^T y$ aufstellen. Mit der Pseudoinversen H^+ von H lässt sich der Systemzustand s approximieren (Jenning, 2008).

$$s_{\text{approx}} = H^+ y = ((H^T H)^{-1} H^T) y \tag{6.21}$$

Da H konstant ist, muss die Pseudoinverse H^+ nicht zur Systemlaufzeit bestimmt werden. Der Rechenaufwand reduziert sich somit auf die Multiplikation $H^+ y$. Aus den Dimensionen von H^+ $(4 \times n)$ und y $(n \times 1)$ ergibt sich ein Aufwand von $4n$ skalaren Multiplikationen.

Der Rechenaufwand für das Kalman-Filter ist die Summe der Rechenaufwände für den Prädiktionsschritt sowie für den Korrekturschritt. Bei n Messpunkten sind insgesamt $8n+3$ skalare Multiplikationen notwendig.

Tabelle 6.2: Matrixdimensionen beim Kalman-Filter mit n Messpunkten

Matrix	Dimensionen
A	4×4
H	$n \times 4$
K	$4 \times n$
s	4×1
y	$n \times 1$

Aufwand der Prädiktion: Es wird das Matrizenprodukt As berechnet. Aus den Matrizengrößen entsprechend Tabelle 6.2 ergibt sich ein Aufwand von 16 Multiplikationen. Es kann jedoch die Struktur von A (Gleichung (6.17)) genutzt werden. Multiplikationen mit Null bzw. mit Eins müssen nicht durchgeführt werden. Daher sind zur Berechnung von $A \cdot s$ nur drei skalare Multiplikationen nötig.

Aufwand der Korrektur: Zur Berechnung der korrigierten Zustandsschätzung müssen zwei Produkte berechnet werden: Hs und Ky. Entsprechend Tabelle 6.2 ergibt sich für beide Matrixmultiplikationen ein Aufwand von jeweils $4n$ skalaren Multiplikationen.

Der Aufwand eines Arbeitsschrittes des Kalman-Filters steigt weiter, wenn einige der bisher getroffenen Vereinfachungen außer Acht gelassen werden. Wird beispielsweise die Schätzfehlerkovarianz P nicht als konstant angenommen muss für diese in jedem Zeitschritt eine Schätzung vorgenommen werden. In der Folge wäre auch der Wert der Kalman-Verstärkung K nicht mehr konstant, wodurch die Anzahl der notwendigen Multiplikationen signifikant anstiege.

6.6 Ausblick auf weiterführende Arbeiten

Zur Implementierung des Fahrspurerkennungssystems ist geplant, das Xilinx *AccelDSP* Synthesewerkzeug zu verwenden. *AccelDSP* generiert synthesefähigen VHDL-Code (ebenfalls Verilog) aus in *MATLAB* beschriebenen Algorithmen. Dies verspricht die Möglichkeit, das in dieser Arbeit beschriebene Kalman-Filter mit einem hohen Abstraktionsgrad zu implementieren. Eine Untersuchung der Anwendbarkeit des *AccelDSP* Tools im Kontext des *FAUST* Projektes sowie seine Auswirkungen auf den Ressourcenbedarf im FPGA wird derzeit vorgenommen (Kirschke, 2009).

Für den ersten Arbeitsschritt ist das Kalman-Filter einen bekannten Startzustand s_0 angewiesen. Eine möglichst genaue Annahme für den Startzustand verringert die Einschwingzeit des Systems auf genaue Schätzungen. Die Bestimmung dieses Startzustands kann außerhalb des regulären Betriebs bei einem stehenden Fahrzeug erfolgen, womit sie keinen Echtzeitanforderungen unterliegt. Laufzeiten bis zu einigen Sekunden wären hinreichend. Hierfür kann ein, in Software implementiertes, statisches Erkennungsverfahren angewandt werden. Über das Framegrabber-Modul kann der Microblaze Prozessor auf die Bilddaten eines Frames zugreifen (vgl. Abschnitt 5). Denkbar ist beispielsweise das Speichern eines Frames aus dem Sobel-gefilterten Bildstrom. Auf diesem kann dann ein robustes Erkennungsverfahren angewandt werden. Als eine Variante dafür sei hier die Mittelwertbildung über mehrere Anwendungen eines *least squares* Polynomapproximationsverfahrens genannt (Jenning, 2008).

Die Präzision des Kalman-Filters, und damit der Zustandsschätzungen, kann erhöht werden, indem das Filter mit dem Regelsystem für Geschwindigkeit und Lenkwinkel integriert wird. Der aktuelle Lenkwinkel kann dann, neben der Geschwindigkeit, über ein kinematisches Modell des Fahrzeugs, in die Systemgleichung (Systemmatrix A) einfließen. Eine weitere Steigerung der Genauigkeit ließe sich erzielen, indem eine Eingabematrix B aufgestellt wird, so dass sich Beschleunigungen und Änderungen des Lenkwinkels auf den Prädiktionsschritt auswirken. Vor einer Umsetzung dieser Schritte sollte untersucht werden, ob ihr Einfluss auf die Genauigkeit des Kalman-Filters den Entwicklungsaufwand rechtfertigt.

Das vorgeschlagene Kalman-Filter ließe sich auch in anderen Umfeldern als dem Carolo-Cup einsetzen. Eine signifikante Änderung ergibt sich jedoch, wenn die Fahrspur nicht in einer Ebene liegt sondern einen dreidimensionalen Verlauf hat. In diesem Fall wäre die Messgleichung nicht-linear. Für die Zustandsabschätzung müsste dann ein erweitertes Kalman-Filter verwendet werden, bzw. ein *iterated extended* Kalman-Filter. Das erweiterte Kalman-Filter linearisiert die System- und Messgleichungen um den aktuellen Schätzwert. Prädiktions- und Korrekturschritt erfolgen dann analog zum einfachen Kalman-Filter. Das *iterated extended* Kalman-Filter erhöht die Genauigkeit des erweiterten Kalman-Filters. Aus der linearisierten Messgleichung ergibt sich ein Schätzwert für den Zustand. Um diesen Wert wird die Messgleichung erneut linearisiert und es wird eine weitere, genauere Schätzung getroffen. Die Iteration wird bis zur Konvergenz, bzw. für eine feste Anzahl an Schritten, wiederholt. (Risack u. a. (1998), Nischwitz und Haberäcker (2004)).

7 Zusammenfasung

Im Rahmen des Forschungsprojektes zu Fahrerassistenz- und Autonomen Systemen werden Führungskonzepte für autonome Fahrzeuge entwickelt. Diese Arbeit leistet hierzu einen Beitrag mit der Entwicklung eines FPGA-basierten System-on-Chip zur Fahrspurerkennung, welches als Grundlage für Bahnführungssysteme autonomer Fahrzeuge dienen soll.

Über die implementierte Schnittstelle wird der ITU-R BT.656 Interlaced-Videostrom der Sony Kamera in das intern verwendete 24 Bit RGB Format umgewandelt: Die Synchronisationssignale *FVAL* und *LVAL* werden aus dem Datenstrom extrahiert, durch Chroma Upsampling wird ein 4:4:4 YCbCr Videostrom erzeugt, dessen Pixelwerte dann in das RGB-Format umgerechnet werden.

Auf dem in das FPGA eingebetteten *Microblaze* Microprozessorsystem wurde ein Softwaremodul zur Steuerung der Kamera im laufenden Betrieb entworfen. Dieses sendet über eine serielle Schnittstelle Kommandos im *VISCA*-Format an die Sony-Kamera. Kameraparameter wie Brennweite, Fokus oder Kontrast lassen sich mit diesem System einstellen.

Mit der Schnittstelle zum VGA-Monitor steht ein wertvolles Hilfsmittel für die Weiterentwicklung des Systems zur Verfügung. Durch unmittelbares visuelles Feedback wird die Analyse und die Fehlersuche signifikant erleichtert. Zur Ausgabe auf dem Monitor wird im Interlaced Videostrom eine Zeilenverdopplung vorgenommen, um die volle vertikale Bildauflösung herzustellen. Ebenfalls eine Debuggingunterstützung stellen die Hard- und Softwarekomponenten zur Bildspeicherung dar. Einzelne Frames werden über eine FSL-Schnittstelle aus der Videopipeline extrahiert und dem Microprozessorsystem zur Speicherung im SRAM übergeben. Von dort aus werden die Bilddaten, kodiert als Bitmap Datenstrom, über eine serielle Schnittstelle an einen PC gesendet, wo sie gespeichert werden.

Für die Fahrspurerkennung wurde ein generisches Hardwaremodul entwickelt, welches einen parallelen Zugriff auf benachbarte Pixel ermöglicht. Die Grundlage für die Implementierung einer Kette von Nachbarschaftsoperationen wurde damit geschaffen. Eine solche Bildverarbeitungskette dient der Extraktion von Messpunkten auf dem Fahrbahnrand, anhand derer das Erkennungssystem seine Berechnungen durchführt.

Für das robuste Erkennen der Fahrspur anhand der Messpunkte im Bild wurde ein Kalman-Filter entworfen. Dieses liefert eine optimale Schätzung des Fahrspurverlaufs im Fahrzeugkoordinatensystem, in Form eines Polynoms dritter Ordnung. Die geometrische Transformation zwischen Bild- und Fahrzeugkoordinaten fließt über die Messgleichung des Kalman-Filters in die Zustandsschätzung ein. Eine Komplexitätsbetrachtung zeigte, dass der Rechenaufwand für das Kalman-Filter höher ist als bei dem bislang verwendeten Verfahren zur statischen Polynomapproximation mit einem Normalengleichungssystem.

Literatur

[Bagni u. a. 2008] BAGNI, Daniele ; MARZOTTO, Roberto ; ZORATTI, Paul: Building Automotive Driver Assistance System Algorithms with Xilinx FPGA Platforms. In: *Xcell Journal* (2008), Nr. 66, S. 20–26

[Bonato u. a. 2007] BONATO, Vanderlei ; PERON, Rafael ; WOLF, Denis F. ; HOLANDA, José A. M. de ; MARQUES, Eduardo ; CARDOSO, Joã.: An FPGA Implementation for a Kalman Filter with Application to Mobile Robotics. In: *SIES*, 2007, S. 148–155

[Carolo-Cup 2009] CAROLO-CUP: *Homepage des Carolo-Cup*. 2009. – URL `http://www.carolo-cup.de`

[Cormen u. a. 2001] CORMEN, Thomas H. ; LEISERSON, Charles E. ; RIVEST, Ronald L. ; STEIN, Clifford: *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001. – ISBN 0262531968

[Digilent 2008] DIGILENT, INC.: *Nexys2 Board Reference Manual*. Datenblatt. 2008

[ITU601 1994] THE ITU RADIOCOMMUNICATION ASSEMBLY: *Encoding Parameters of Digital Television for Studios*. 1994. – URL `http://www.itu.int/rec/R-REC-BT.601/en`

[ITU656 1998] THE ITU RADIOCOMMUNICATION ASSEMBLY: *Interfaces for Digital Component Video Signals in 525-Line and 625-Line Television Systems Operating at the 4:2:2 Level of Recommendation ITU-R BT.601*. 1998. – URL `http://www.itu.int/rec/R-REC-BT.656/en`

[Jack 2005] JACK, Keith: *Video Demystified: A Handbook for the Digital Engineer*. Fourth Ed. Elsevier Science & Technology Books, 2005

[Jenning 2008] JENNING, Eike: *Systemidentikation eines autonomen Fahrzeugs mit einer robusten, kamerabasierten Fahrspurerkennung in Echtzeit*, HAW-Hamburg, Masterarbeit, 2008

[Kirschke 2009] KIRSCHKE, Marco: *Echtzeitbildverarbeitung mit einer FPGA-basierten Prozessorelementkette in einem Fahrspurerkennungssystem*, HAW-Hamburg, Bachelorarbeit, 2009

[Manske 2008] MANSKE, Nico: *Kamerabasierte Präzisionsnavigation mobiler Systeme im Indoor-Bereich*, HAW-Hamburg, Masterarbeit, 2008

[Meisel 2008] MEISEL, Andreas: *Robot Vision*, Hochschule für Angewandte Wissenschaften Hamburg, Vorlesungsfolien, 2008

[Nischwitz und Haberäcker 2004] NISCHWITZ, Alfred ; HABERÄCKER, Peter: *Masterkurs Computergrafik und Bildverarbeitung*. Erste Auflage. Vieweg + Teubner, 2004

[Poynton 2005] POYNTON, Charles: *Converting YCbCr to RGB*. April 2005. – URL `http://www.poynton.com/notes/short_subjects/video/YCbCr_to_RGB`

[Risack u. a. 1998] RISACK, R. ; KLAUSMANN, P. ; KRUGER, W. ; ENKELMANN, W.: Robust lane recognition embedded in a real-time driver assistance system. In: *in Proc. IEEE IV*, 1998, S. 35–40

[Sony 2006] SONY CORPORATION: *FCB-PV10 Color Camera Module - Technical Manual*. Datenblatt. 2006

[Vanevenhoven 2007] VANEVENHOVEN, Tim: Using MATLAB to Create IP for System Generator for DSP. In: *Xcell Journal* (2007), Nr. 62, S. 24–27

[Wikipedia 2006] WIKIPEDIA: *Barns grand tetons YCbCr separation.jpg*. Dezember 2006. – URL `http://de.wikipedia.org/w/index.php?title=Datei:Barns_grand_tetons_YCbCr_separation.jpg`

[Wikipedia 2009] WIKIPEDIA: *Windows Bitmap*. Januar 2009. – URL `http://de.wikipedia.org/wiki/Windows_Bitmap`

[Xilinx 2007a] XILINX, INC.: *Fast Simplex Link (FSL) Bus (v2.11a)*. Datenblatt. 2007

[Xilinx 2007b] XILINX, INC.: *Processor Local Bus (PLB) v4.6 (v1.00a)*. Datenblatt. 2007

[Xilinx 2008] XILINX, INC.: *MicroBlaze Processor Reference Guide*. Datenblatt. 2008

Versicherung über Selbstständigkeit

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

Hamburg, 30. April 2009	
Ort, Datum	Unterschrift