Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Marco Kirschke

Echtzeitbildverarbeitung mit einer FPGA-basierten Prozessorelementkette in einem Fahrspurerkennungssystem

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Marco Kirschke

Echtzeitbildverarbeitung mit einer FPGA-basierten Prozessorelementkette in einem Fahrspurerkennungssystem

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Technischen 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. rer. nat. Michael Schäfers

Abgegeben am 26. Mai 2009

Marco Kirschke

Thema der Bachelorarbeit

Echtzeitbildverarbeitung mit einer FPGA-basierten Prozessorelementkette in einem Fahrspurerkennungssystem

Stichworte

Autonome Fahrzeugführung, Echtzeit, System-on-Chip, FPGA, Bildverarbeitung, Fahrspurerkennung, Nachbarschaftsoperationen, VHDL-Codegenerator, AccelDSP

Kurzzusammenfassung

In dieser Arbeit wird die Entwicklung einer aus VHDL-Modulen bestehenden Prozessorelementkette zur Echtzeit Verarbeitung von Videodaten für den Einsatz in einem Fahrerassistenzsystem zur Fahrspurerkennung beschrieben. Dazu wurde ein Prototyp entwickelt, der die Aufbereitung und Verarbeitung von Bilddaten einer digitalen Videokamera in einem FPGA-basierten System-on-Chip vornimmt. Die aus den Videodaten ermittelten Messpunkte werden verwendet, um den Verlauf der Fahrspur durch einen Polynom dritten Grades darstellen zu können. Für die weitere Verarbeitung der Messdaten auf der FPGA Plattform, ist der Einsatz eines VHDL-Codegenerators untersucht worden, der die Umsetzung einer abstrakten mathematischen, in MATLAB-Code erstellten, Modellierung in den RTL-Code einer Fixed-Point Implementierung vornimmt.

Marco Kirschke

Title of the paper

Real-time image processing by the use of a FPGA-based pipeline in a lane detection system

Keywords

Autonomous vehicle steering, real-time, System-on-Chip, FPGA, image processing, lane detection, neighborhood operations, VHDL-code generator, AccelDSP

Abstract

In this thesis the development of a real-time image processing pipeline, composed of VHDL-modules, for the usage in lane detection systems is described. Therefore a prefiguration was created, which handles the preparation and the image processing of video data from a digital camera in a FPGA-based system-on-chip. With the detection of measurement points from the video data stream, the lane could be described as a third order polynomial. Referring to the further data processing, a VHDL-code generator tool was evaluated, which performs the translation of an abstract mathematical model, designed in MATLAB-Code, into a fixed-point implementation supporting RTL-Code.

Inhaltsverzeichnis

1 Einleitung

Das Department Informatik der Hochschule für Angewandte Wissenschaften (HAW) Hamburg bietet in dem Forschungsprojekt FAUST (Fahrerassistenz- und Autonome Systeme) eine Umgebung für die Entwicklung von eingebetteten Systemen, die sich an den Vorgaben der derzeitigen Industriestandards orientiert. Hierbei werden Prototypen für Ausweichassistenzsysteme, Fahrzeugregelungen, Stabilisierungsverfahren oder Navigations- und Lokalisierungssysteme entwickelt und zusammen auf unterschiedlichen Plattformen integriert. Ein Bereich dieser Entwicklung ist die *Onyx* Plattform, die für die Teilnahme am Carolo Cup [Carolo-Cup, 2009] der Technischen Universität Braunschweig konzipiert wurde. Der Wettkampf soll Teams aus Studenten verschiedener Hochschulen die Möglichkeit geben, sich mit der Entwicklung und Umsetzung von autonomen Modellfahrzeugen auseinander zu setzen. Dabei treten die Modellfahrzeuge in einem Maßstab von 1:10 in verschiedenen Disziplinen gegeneinander an. In die Bewertung geht neben der Präsentation von Herstellkosten und Energiebilanzen sowie der theoretischen Konzepte des Systems auch die Demonstration anhand von Testfahrten (*Rundstrecke mit Hindernissen, Rundfahrt ohne Hindernisse, Einparken parallel*) mit ein. Gerade für die autonome Fahrt auf einem vorgegebenen Parcours ist die bilddatengestützte Fahrspurerkennung ein zentraler Bestandteil der geregelten Spurführung.

Diese Bachelorarbeit behandelt eine System-on-Chip Bildverarbeitungsplattform, welche in das bestehende System zur Bahnführung des Modellfahrzeuges integriert werden kann. Der Prototyp des Systems soll die Aufgaben einer echtzeitfähigen Fahrspurerkennung erfüllen, um in einem späteren Schritt in dem Bahnführungssystem eingesetzt zu werden. Durch den Einsatz des FPGA-basierten System-on-Chip (SoC) kommt es zu einer besseren Ausnutzung des begrenzten Platzes auf dem Zielfahrzeug, einer Verringerung des Energiebedarfs und zusätzlich zur Entlastung der bereits bestehenden Infrastruktur. Dafür ist zunächst der Bilddatenstrom einer PAL-Kamera durch eine Prozessorelementkette auf ein definiertes RGB-Format zu übertragen, um im Anschluss die Bilddaten für eine Fahrspurerkennung zu verwenden. Dazu werden Messpunkte auf einer Fahrspurmarkierung aus dem Videodatenstrom extrahiert, um die Fahrspur für Datenreduktion mit einem Polynom dritten Grades approximieren zu können. Die rekursive Approximation wird im fließenden Datenstrom mit einem Kalman-Filter erzeugt. Die Integration eines Kalman-Filters auf die FPGA-Entwicklungsplattform wird zudem durch die Verwendung des Xilinx AccelDSP Synthese Werkzeugs untersucht [vgl. Vanevenhoven, 2007].

Das System-on-Chip-basierte Bildverarbeitungssystem umfasst eine Prozessorelementkette, die in einzelne Teilprozesse gegliedert ist (vgl. *Abb. 1*):

- Anbindung der *Sony FCB PV10* Kamera an das *Spartan3E* FPGA des Nexys2 Entwicklungsboards
- Aufbereitung der Kameradaten des ITU-Rec. BT.656 [ITU656, 1998] Formats in ein systeminternes 24 Bit RGB Datenformat
- Bereitstellung der Videodaten über das VGA Interface des Nexys2 zur Darstellung auf einem Monitor
- Vorverarbeitung der Bilddaten zur Kantenextraktion; beinhaltet die Reduzierung des Bilddatenstromes sowie die Anwendung von Nachbarschaftsoperationen auf den Videodatenstrom zur Laufzeit
- Analyse des VHDL-Codegenerators AccelDSP zur Integration abstrakter mathematischer Modelle in das FPGA System, welche die Aufgabe der Fahrspurerkennung realisieren
-

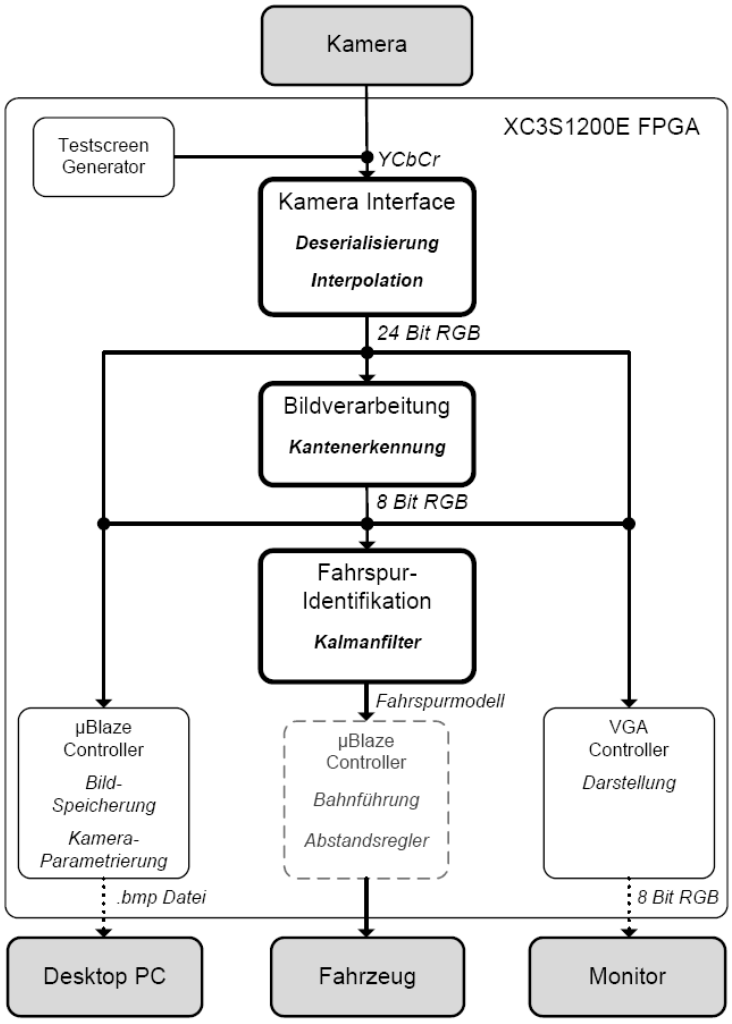


Abb. 1 : Bestandteile des SoC-basierten Bildverarbeitungssystems für ein autonomes Fahrzeug

Neben diesen Kernaufgaben werden folgende Zusatzfunktionen zu Dokumentationszwecken und zur Verringerung des Entwicklungsaufwandes erzeugt:

- Entwicklung eines Testscreen Generators zur Verifikation des Kamera Interfaces ohne die Versorgung mit Videodaten durch die Kamera
- Modul zur Parametrierung der Kamerakonfiguration, um diese an Umweltsituationen anzupassen
- Modul zur Abspeicherung der Bilddaten aus dem laufenden System zu Dokumentationszwecken

Der Aufbau der vorliegenden Arbeit gestaltet sich folgendermaßen:

In **Kapitel 2** werden die verwendeten, bereits bestehenden Hardware- und Softwarekomponenten, sowie das Gesamtsystem im Detail vorgestellt. Desweiteren werden hier die eingesetzten Entwicklungswerkzeuge und Vorgehensweisen genauer betrachtet.

Das **3. Kapitel** behandelt die detailierte Erläuterung der Module des Kamera Interfaces und der entwickelten Hilfsmodule. Zunächst wird das von der Kamera bereitgestellt ITU-Rec. BT.656 Videodatenformat erläutert, und die Erzeugung der Synchronisationssignale anhand des Datenstroms für die Verwendung im VGA Controller beschrieben. Im Anschluss wird die Umwandlung der Bilddaten in das systeminterne 24 Bit RGB Format und der Aufbau des Deinterlace Moduls zur Zeilenverdopplung erläutert. Im zweiten Teil des Abschnittes werden Aufbau und Verhalten des Testscreen Generators, des Kamera Parametrierungsmoduls und des FrameGrabbers zur Bildaufzeichnung beschrieben.

Für die Erkennung der Fahrspurmarkierungen müssen die Bilder des Videodatenstroms bearbeitet werden; das geschieht innerhalb der Bildverarbeitungsmodule und wird in **Kapitel 4** erläutert. Neben einer Einleitung über die Anwendung von Nachbarschaftsoperationen durch Filtermasken, werden in diesem Teil die konkrete Realisierung der Filterfunktion und das Deserialisierungsmodul beschrieben. Dieses hat die Aufgabe die Nachbarschaftsoperationen durch die Zwischenspeicherung mehrerer Zeilen zu ermöglichen.

Das **5. Kapitel** umfasst eine Einführung in das Konzept der Kalman Filterung und die theoretische Integration in ein FPGA-basiertes System. Dazu wird anhand eines Beispiels, die Implementierung eines Kalman-Filters durch den Einsatz von MATLAB und dem Synthese Werkzeug AccelDSP vorgestellt.

Die Inhalte und Ergebnisse dieser Arbeit werden in **Kapitel 6** zusammengefasst.

2 Bestandteile und Schnittstellen des Systems

2.1 Vorstellung der Systemkomponenten

Für die Realisierung der Echtzeitbildverarbeitung wurde eine System-on-Chip-basierte Entwicklungsplattform auf dem Spartan 3E FPGA des Nexys2 Entwicklungsboards [Digilent, 2008] erstellt. Die Sony FCB-PV10 Kamera ist durch ein FFC (Flat Flexible Cable) über einen Adapter an die Pmod Schnittstellen des Nexys2 Boards angeschlossen (vgl *Abb. 2*). Über diese Verbindung kann das FPGA direkt auf die Videodaten der Kamera zugreifen. Es besteht eine zweite, serielle Verbindung zur Kamera, über die eine Parametrierung der Kameraeigenschaften erfolgen kann. Die Versorgungsspannung der Kamera wird für den entwickelten Prototypen über ein separates Netzteil bereitgestellt; kann bei Bedarf aber auch über das Entwicklungsboard bezogen werden. Das Nexys2 Board wird durch eine USB Verbindung zum Entwicklungsrechner mit Strom versorgt.

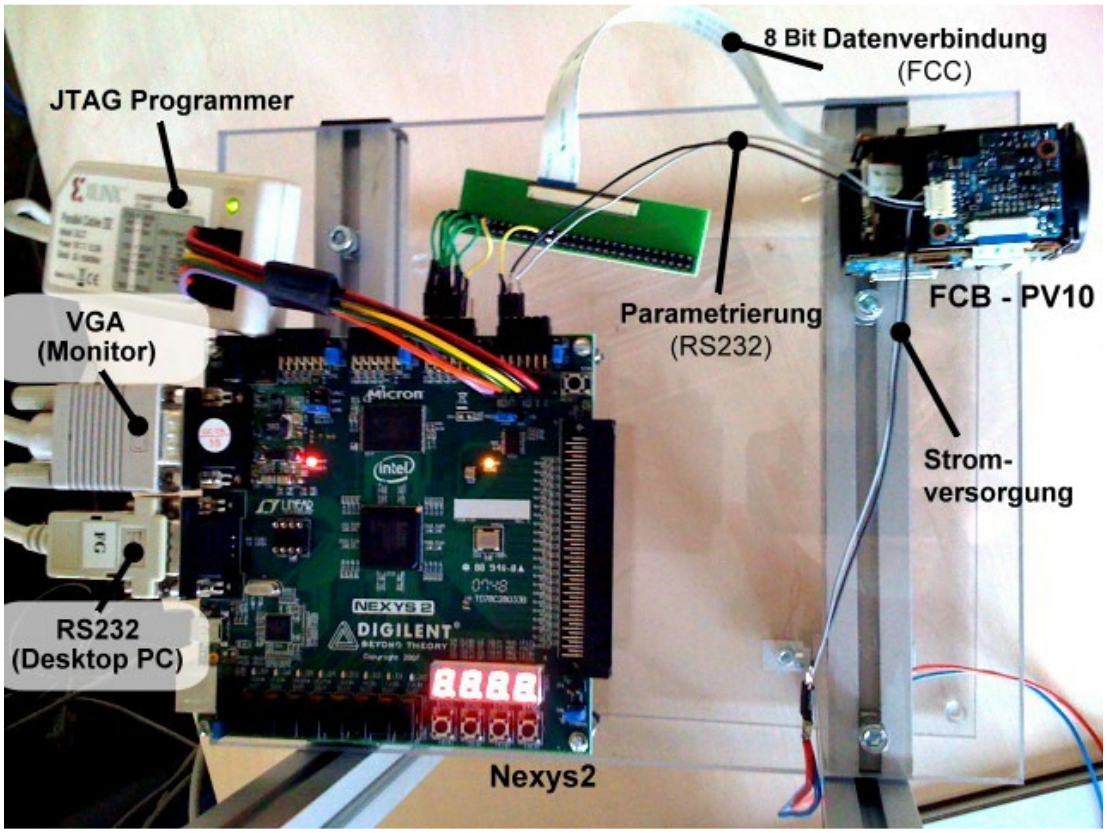


Abb. 2 : Versuchsaufbau der FPGA-basierten Entwicklungsplattform

Zur Darstellung der Videodaten der Verarbeitungspipeline zur Laufzeit ist über die VGA Schnittstelle des Nexys2 Boards ein Monitor mit an das System angeschlossen. Darüberhinaus können über eine RS232 Schnittstelle Daten von und an einen PC übertragen werden. Der Download der Software vom Entwicklungsrechner auf das Board erfolgt über die JTAG Schnittstelle des Nexys2 und einen Xilinx JTAG Programmer.

Für die Hardware Modellierung in RTL- Codestyle wurde ein modularer Ansatz gewählt, so dass sich die Echtzeitpipeline aus einzelnen Komponenten zusammensetzt, die wiederum untergeordnete Module enthalten (vgl. *Abb. 3*). Das Kamera Interface übernimmt die Videodaten der Kamera und wandelt diese aus dem ITU-Rec. BT.656 Format in ein systemeigenes 24 Bit RGB Format um. Zusätzlich werden hier die FVAL- und LVAL Signale erzeugt, die innerhalb der gesamten Pipeline zur Synchronisation auf den Datenstrom benutzt werden. Das VGA Interface dient zur Darstellung des Bilddatenstroms zur Laufzeit auf einem VGA Monitor. Dazu muss der interlaced Videodatenstrom durch eine Bildzeilenverdopplung *„deinterlaced“* werden, bevor er durch den VGA Controller auf dem Monitor angezeigt werden kann. Im Nachbarschaftsoperationen Modul wird die eigentliche Bildbearbeitung über die Anwendung von Filtermasken auf den Bilddatenstrom vorgenommen; für die vorliegende Arbeit wird zur Detektion von markanten Bildpunkten ein Sobel-Filter eingesetzt. Für das Modul der Fahrzeugführung ist bis zu diesem Zeitpunkt keine Implementierung vorgenommen worden. Allerdings wurde das Synthesewerkzeug AccelDSP untersucht, durch dessen Einsatz eine mathematische Modellierung für die Detektion und Prädiktion der Fahrspur in RTL-Code umgesetzt werden kann, was den Entwicklungsaufwand wesentlich verringert.

Neben den VHDL Modulen wird ein Microblaze Softcore Prozessor verwendet, um Softwaremodule in das System zu integrieren. Das Microprozessor System bietet den Vorteil, dass die Peripherieelemente des Entwicklungsboards durch den Einsatz von C Programmen und die Verwendung von Standardbibliotheken relativ einfach angesprochen werden können. Daher verringert sich die Entwicklungszeit von Modulen, deren Ausführung keine unmittelbare zeitliche Relevanz haben. So können die Parameter der Kamera über eine serielle RS232 Verbindung eingestellt werden. Zusätzlich ist das Abspeichern von Bildern innerhalb der Videodatenpipeline möglich. Dazu wird ein Bild über den Fast Simplex Link (FSL) an das Microblaze System übertragen, dort im SRAM abgelegt, anschließend mit einem Bitmap Header versehen und über eine weitere serielle Schnittstelle an einen Desktop PC übertragen.

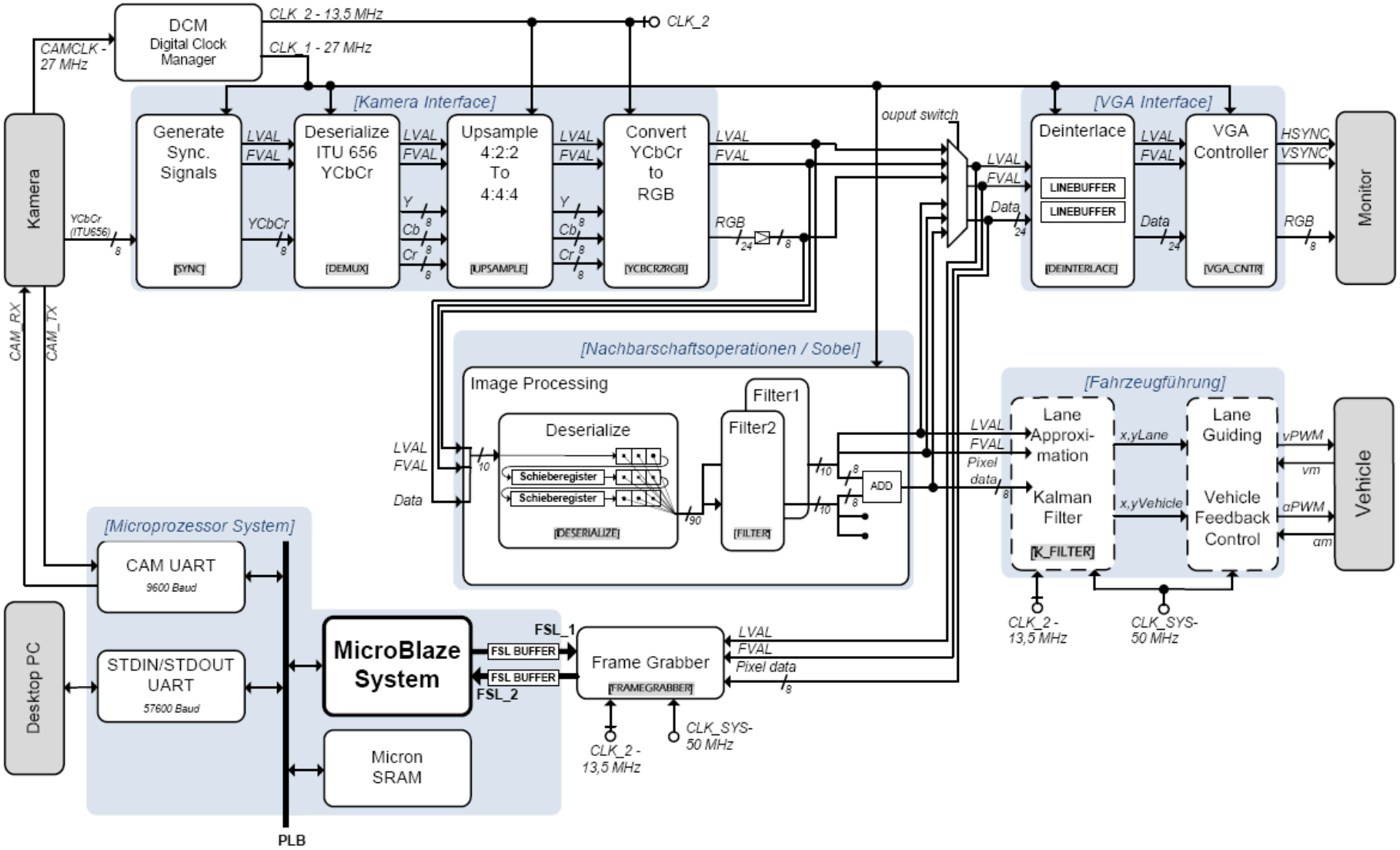


Abb. 3 : Gesamtübersicht mit den Teilkomponenten „Kamera Interface“ zur Bilddatenaufbereitung, „VGA Interface“ zur Darstellung, „Nachbarschaftsoperationen“ für die Kantenextraktion, die „Fahrzeugführung“ zur Auswertung der Messpunkte und das Mircoprozessor System

2.2 Integration bestehender Komponenten

Bei der Entwicklung des in dieser Arbeit vorgestellten Systems, mussten die Schnittstellen zu bereits bestehenden Komponenten berücksichtigt werden. Die Videodaten werden von der Sony FCB PV10 Kamera bezogen und in der entwickelten System-on-Chip Plattform auf einem Spartan 3E FPGA des Nexys2 Entwicklungsboards verarbeitet. Außerdem kommt für die Darstellung der Videodaten auf einem VGA Monitor ein bereits bestehendes VGA Controller Modul zum Einsatz, welches die Synchronisation der Videodaten mit den Vorgaben des Monitors realisiert.

2.2.1 Sony FCB-PV 10 Kamera

Die FCB-PV10 Kamera von Sony ist mit einem ¼ Type Interline-Transfer CCD Sensor ausgestattet und liefert die digitalen Bilddaten über ein FFC parallel an die Pmod Pins des Entwicklungsboards. Die FCB-PV10 stellt eine VGA Auflösung mit 640x480 Pixeln pro Bild bereit und liefert diese in Bildraten von 29,97 oder 25 Bildern pro Sekunde. Das Format der Daten hat nach der ITU-Rec. BT.656 die Form des YCbCr 4:2:2. Außerdem kann die Kamera auf verschiedene Ausgabemodi [vgl. *Tab. 1*] konfiguriert werden, die sich in der Anzahl der in einem Takt übertragenen Bits und den Taktraten unterscheiden.

Standard	Modus	Ausgang	Sync. Signale	Takt
[ITU601, 1994]	16 Bit Progressive	YCbCr 16 Bit 4:2:2	HSYNC/VSYNC	13,5 MHz
[ITU656, 1998]	8 Bit Progressive	YCbCr 8 Bit 4:2:2	HSYNC/VSYNC & SAV/EAV	27 MHz
[ITU656, 1998]	**8 Bit Interlace**	**YCbCr 8 Bit 4:2:2**	**HSYNC/VSYNC & SAV/EAV**	**27 MHz**

Tab. 1 : Übertragungsmodi der Sony FCB PV10

Im 16 Bit Progressive-Modus wird jeweils zu einem 8 Bit Y Luminazwert abwechselnd der entsprechende 8 Bit Cb bzw. Cr Chrominazwert parallel übertragen. In beiden Progressive-Modi liefert die Kamera die Bilddaten Zeile für Zeile innerhalb des Datenstroms, wodurch bei einer Auflösung von 640x480 Pixeln, alle 480 Zeilen eines Bildes innerhalb einer Periode enthalten sind. Im Interlace-Modus dagegen werden in der Übertragungsperiode eines Bildes zwei Teilbilder, die sogenannten „Fields“, übertragen. Diese Teilbilder enthalten nur die Hälfte der Zeilen. Über ein Registers kann die Kamera den eingestellten Betriebsmodus langfristig speichern, so dass zur bloßen Einstellung der Betriebsart, keine Verbindung über die RS232 Schnittstelle bereitgestellt werden muss.

Für das entwickelte System wird die Kamera auf den 8 Bit Interlace Modus konfiguriert. Innerhalb der Pipelinestufe des Kamera Interfaces muss, durch die Parallelisierung und Erweiterung des 4:2:2 Abtastverhältnisses auf 4:4:4, die Taktfrequenz auf 13,5 MHz verringert werden. Herkömmliche Monitore können die Bilddaten mit dieser geringen Frequenz nicht darstellen, daher können die Progressive Modi nicht eingesetzt werden. Im Interlace Modus werden die beiden einzelnen Fields als Gesamtbilder behandelt und innerhalb des Deinterlace Moduls durch Zeilenverdopplung auf das unterstütze Ausgabeformat von 480 Zeilen gebracht. Für die Zeilenverdopplung wird jede Zeile zwischengespeichert, wodurch die Ausgabe wieder mit der ursprünglichen Frequenz von 27 MHz erfolgen kann; das zweimalige Ausgeben der Zeile mit 27 MHz beansprucht die gleiche Zeit, wie das einmalige Einlesen mit 13,5 MHz.

Die Kamera ist über zwei separate Schnittstellen mit der Entwicklungsplattform verbunden (vgl. *Tab. 2*). Die erste Verbindung (CN701) versorgt die Kamera mit der Netzspannung von 6 bis 12 Volt und stellt die serielle RS232 Schnittstelle zur Parametrierung bereit. Über die Parametrierung können die Eigenschaften des Ausgangsbildes beeinflusst werden; so besteht die Möglichkeit Zoom, Fokussierung, Weißabgleich, Belichtungsautomatik oder Blendeneinstellungen der Kamera konfigurieren zu können. Diese Parametrierung wird über das von Sony bereitgestellte VISCA - Protokoll *(Video System Control Archtitecture)* vorgenommen.

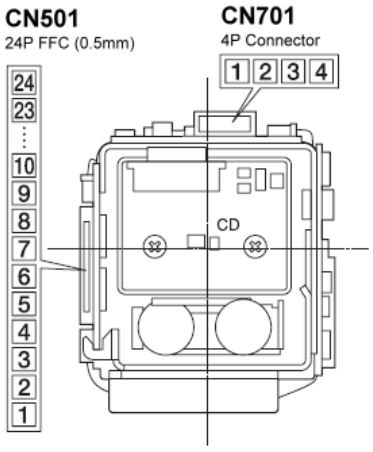
Die zweite Verbindung (CN501) erfolgt über ein FFC, über das der Datenstrom, die Synchronisationssignale und das Takt Signal parallel übermittelt werden. Obwohl die Kamera die Synchronisationssignale HSYNC (Horizontal Synchronisation) und VSYNC (Vertical Synchronisation) liefert, werden diese in der Implementierung nicht verwendet, sondern vielmehr eigene Synchronisationssignale, über die Auswertung der Informationen innerhalb des Videodatenstroms, erzeugt. Durch diese Vorgehensweise werden weniger Pins des verwendeten Nexys2 Boards in Anspruch genommen, was eine spätere Integration von Anwendungen zur Bahnführung oder Fahrzeugsteuerung des Carolo Cup Fahrzeugs auf die verwendete Plattform erleichtert.

CN501 (for 8 Bit data bus configuration)

PIN	Name		Level
1	GND	GROUND	
2	Y0	Digital Out 0	0-3.2 V
3	Y1	Digital Out 1	0-3.2 V
4	Y2	Digital Out 2	0-3.2 V
5	Y3	Digital Out 3	0-3.2 V
6	Y4	Digital Out 4	0-3.2 V
7	Y5	Digital Out 5	0-3.2 V
8	Y6	Digital Out 6	0-3.2 V
9	Y7	Digital Out 7	0-3.2 V
10	GND	GROUND	
11	C0	Hi imp.	
12	C1	Hi imp.	
13	C2	Hi imp.	
14	C3	Hi imp.	
15	C4	Hi imp.	
16	C5	Hi imp.	
17	C6	Hi imp.	
18	C7	Hi imp.	
19	GND	GROUND	
20	VSYNC	Vertical SYNC	0-3.2 V
21	HSYNC	Horizontal SYNC	0-3.2 V
22	GND	GROUND	
23	CLOCK	Clock signal	0-3.2 V
24	GND	GROUND	

CN701

PIN	Name		Level
1	UNREG	Power Input	6-12 V (DC)
2	GND	GROUND	
3	TD		TTL Level (0-5 V)
4	RD		TTL Level (0-5 V)



Tab. 2 : Ein- und Ausgangssignale der digitalen Kamera

Mit ihrer geringen Abmessung von 37,3mm * 43,8mm * 61 mm und dem Gewicht von nur 84 Gramm, eignet sich die Kamera speziell zur Integration in eingebettete Systeme, wie die im Rahmen des Carolo Cup verwendete *Onyx*- Plattform.

2.2.2 Digilent Nexys2 – Board

Das Nexys2 Board [Digilent, 2008] bietet neben dem Spartan 3E FPGA [Spartan-3E, 2008] eine Reihe von Peripherie, wie externen Speicher oder I/O Schnittstellen, mit denen die Integration des FPGA in ein Gesamtsystem erleichtert wird. Die Aufteilung des Nexys2 Boards ist im Folgenden beschrieben und über die Ziffern in *Abb. 5* gekennzeichnet.

[1] Das Xilinx Spartan3E XC3S1200E FPGA besteht aus fünf programmierbaren Elementen:

- *Configuarble Logical Blocks (CLBs)*, diese enthalten flexible Look-Up Tabellen, welche sowohl als logische Einheit als auch zur Datenspeicherung verwendet werden können.
- *Input/Output Blocks (IOBs)*, die die bidirektionale Verbindung von der internen Logik zu den I/O Pins herstellen und 3-State Operationen unterstützen.
- *Block RAM*, zur Datenspeicherung.
- *Multiplizierer*, die zu zwei binären 18-Bit Zahlen das Produkt liefern.
- *Digital Clock Manager Blocks (DCM),* mit deren Hilfe Taktsignale verzögert, multipliziert, dividiert und verschoben werden können.

Die Anordnung auf dem FPGA ist (vgl. *Abb. 4*) so organisiert, dass die CLBs von den IOBs ringförmig umgeben sind.

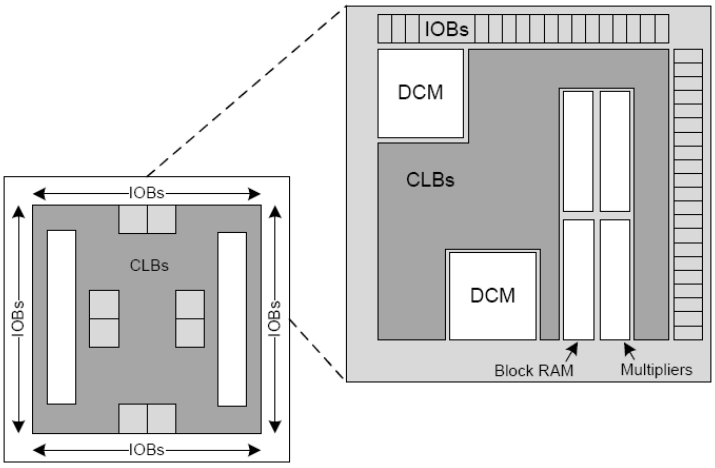


Abb. 4 : Aufbau des Spartan 3E FPGA

[2] Als externes RAM ist auf dem Nexys2 Board ein 128 MBit Micron pseudo-statisches DRAM integriert, das, wie ein typisches SRAM, asynchrone Lese- und Schreibvorgänge in 70ns vornehmen kann. Daneben verfügt das Nexys2 über ein 128MBit Intel StrataFlash ROM das Lesevorgänge innerhalb von 110ns und Schreibvorgänge in 70ns unterstützt.

[3] Eine serielle Schnittstelle ist in Verbindung mit einem Spannungswandler integriert, wobei dieser die -12V bis 12V RS232 Pegel auf die 3.3V Betriebsspannung des FPGA umsetzt.

[4] Das Nexys2 verfügt über 4 Pmod Schnittstellen die direkt mit dem I/O Ports des FPGA verbunden sind. Ein Pmod besteht aus zwölf Pins,wobei acht

Daten-, zwei Masse- und zwei Powerpins zur Verfügung stehen. Für diese Pmod Schnittstellen stellt Digilent eine Reihe von peripheren Modulen, zum Beispiel A/D oder D/A Wandler, Infrarotsensoren, Mikrophone oder Thermometer, bereit, die sich einfach in ein bestehendes System integrieren lassen

[5] Die VGA Schnittstelle verwendet 10 Signale des FPGAs, um 8-Bit Farbinformationen und zwei Synchronisationssignale nach dem VESA Standard für eine Auflösung von 640 mal 480 [Vesa, 2009] bereit zu stellen; sie kann dabei sowohl mit CRT- als auch mit LCD- Bildschirmen verbunden werden.

[6] Über eine PS2 Schnittstelle kann zusätzlich Eingabeperipherie, wie eine Maus oder eine Tastatur, an das Nexys2 angeschlossen werden.

[7] Neben der USB2 Schnittstelle, die auch zum Download von `.Bit` Dateien ins FPGA benutzt werden kann, besteht die Möglichkeit die Spannungsversorgung über eine Netzteilbuchse oder eine Batterie zu beziehen.

[8] Eine Verbindung über ein FX-2 Kabel kann verwendet werden, um Peripherie anzuschließen, die in Taktraten von bis zu 100MHz laufen.

[9] Außerdem werden verschiedene I/O Schnittstellen bereitgestellt; neben acht LEDs und einer vierstelligen Sieben-Segment Anzeige können vier Drucktasten und acht Schiebeschalter verwendet werden.

[10] Es verwendet einen 50 MHz Oszillator und hält einen Socket für einen zusätzlichen Oszillator bereit.

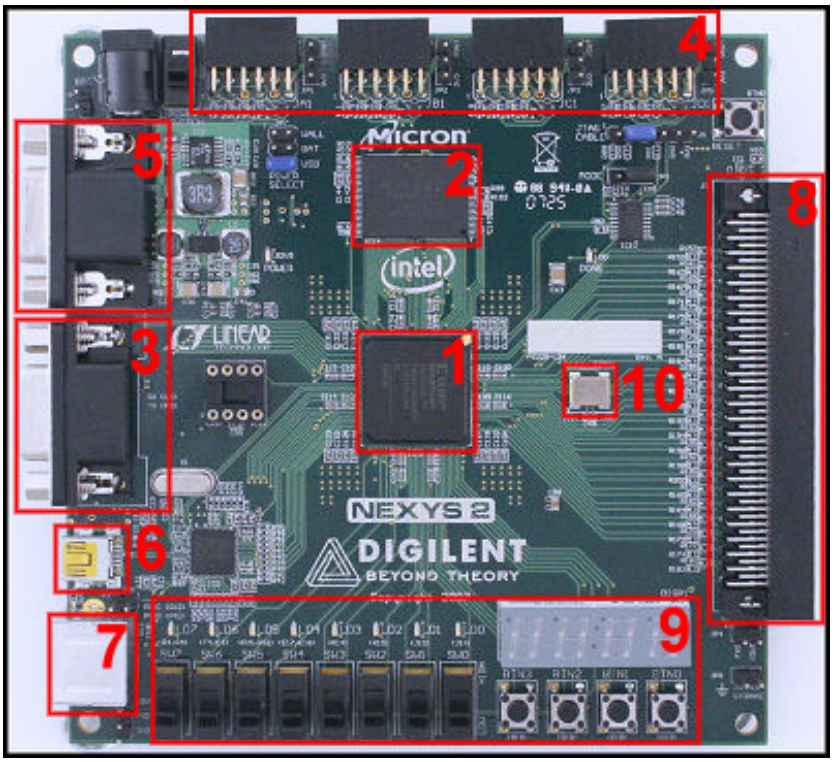


Abb. 5 : Anordnung der Komponenten des Nexys2 Boards

2.2.3 VGA Controller

Die Pixeldaten innerhalb der Videopipeline werden zusätzlich mit einem Monitor dargestellt, damit eine visuelle Echtzeitkontrolle stattfinden kann. Zu diesem Zweck wird eine bereits implementierte Komponente eingesetzt, die standardisierte Synchronisationssignale (HSYNC, VSYNC) für die VGA Schnittstelle des Nexys2 erzeugt. Da diese Schnittstelle lediglich 8 Pins für die Übertragung der Farbinformationen bereithält, werden die RGB Bilddaten am Ausgang des VGA Controllers von 24 Bit auf 8 Bit reduziert. Die jeweils 8 Bit breiten Farbinformationen für Rot, Grün und Blau werden demnach auf einen einzigen 8 Bit Wert mit

- den höchsten 3 Bit des roten Farbwertes
- den höchsten 3 Bit des grünen Farbwertes
- den höchsten 2 Bit des blauen Farbwertes

zusammengefasst. Der VGA Controller nimmt neben den Bilddaten im RGB Format auch die interne Synchronisationssignale (LVAL, FVAL) entgegen, aus denen ermittelt wird, wann ein gültiges Gesamtbild bzw. eine gültige Bildzeile anliegen (vgl. *Abb. 6*). Der VGA Controller enthält einen Initialisierungsautomaten, der sich bei Systemstart auf den Bilddatenstrom synchronisiert.

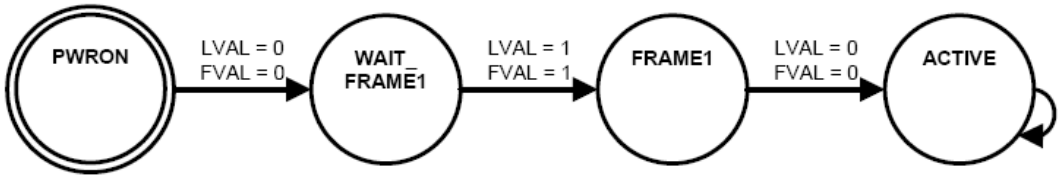


Abb. 6 : Zustandsdiagramm des VGA Controllers

Zur Erzeugung der HSYNC und VSYNC Signale werden zwei Zähler verwendet, die entsprechend der Takte pro Bildzeile bzw. der Bildzeilen pro Gesamtbild inkrementieren. Die HSYNC und VSYNC Signale werden parallel zu den Bilddaten an den Monitor übertragen. Dieser detektiert anhand der Signale, zu welchem Zeitpunkt in den Bilddaten ein Zeilenwechsel bzw. die Darstellung eines neuen Bildes vorgenommen werden soll. Die Dokumentation des Nexys2 liefert eine Übersicht zum Timing des VGA Interfaces mit einer Auflösung von 640x480 Pixeln. Während der Front- bzw. Backporch Zeiten können keine Informationen dargestellt werden [vgl. *Abb. 7*].

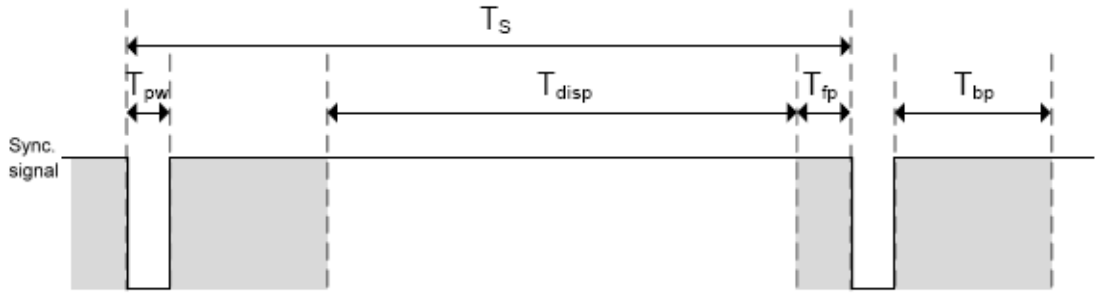


Abb. 7 : Gültigkeitsbereiche der VGA Synchronisationssignale [Digilent, 2008]

Diese Darstellung behandelt ein System mit Zeilen, die eine Länge von 800 Takten haben. Das Datenblatt der Kamera gibt vor, dass die Bilddaten in Zeilen von 858 Takten übertragen werden. Die Anzahl der relevanten Bildpunkte, sowie die Pulsbreiten der Synchronisationssignale sind konstant, also müssen die Anteile der Front- und Backporch Zeiten auf die korrekte Taktanzahl angepasst werden (vgl. *Tab. 3*). Das VHDL Modul des VGA Controllers stellt die Konfiguration der relevanten Zeiteinheit über *GENERICS* zur Verfügung; diese werden mit dem aktuellen Zählerstand verglichen und die entsprechenden Ausgangssignale erzeugt(vgl. *Code 1* und *Code 2*).

Symbol	**Beschreibung**	*VGA Standard*		**Anpassung**	
		Vert./Zeilen	*Horiz./Takte*	**Vert./Zeilen**	**Horiz./Takte**
T_S	Gesamtlänge	*521*	*800*	**525**	**858**
T_{disp}	Relevante Pixel	*480*	*640*	**480**	**640**
T_{pw}	Pulsbreite	*2*	*96*	**2**	**96**
T_{fp}	Front porch	*10*	*16*	**11**	**31**
T_{bp}	Back porch	*29*	*48*	**32**	**91**

Tab. 3 : Anpassung der VGA Controller Parameter an den verwendeten Bilddatenstrom

```
--************Generics for VGA Controller*************************************
generic map (
   --Time data from Digilent, Nexys2 Reference Manual, p.12
   --change timing here to overwrite timing in components:
   T_HT_PULSE_END => 126,
   T_HT_BP_END => 216,
   T_HT_DP_END => 857,
   T_HT_FP_END => 30,
   T_HT_MAX => 857,
   T_VT_PULSE_END => 12,
   T_VT_BP_END => 44,
   T_VT_DP_END => 524,
   T_VT_FP_END => 10,
   T_VT_MAX => 524)
```

Code 1 : Angepasste GENERICS des VGA Controllers

```
--************hsync creation*********************************************************
hsync: process (count, en_in)
begin
   hsync_out <= '1';
   if en_in = '1' and
      ((ht_fp_end > ht_pulse_end and
       (count > ht_fp_end or count <= ht_pulse_end)) or
       (count > ht_fp_end and count <= ht_pulse_end)) then
      hsync_out <= '0';
      -- timing should be variable -> possibility that fp_end is set later
      -- on time ray than pulse_end -> nevertheless: correct sync-pulse
   end if;
end process hsync;
```

Code 2 : Definition der GENERICS und Erzeugung des HSYNC Signals im VGA Controller Modul

Die Timing Simulation der VGA Controller Ausgänge zeigt die relevanten Stellen zur Erzeugung der Synchronisationspulse. Zur Verdeutlichung der Abhängigkeiten werden ebenfalls die LVAL und FVAL Signale und die Bilddaten (R_OUT, G_OUT, B_OUT) angezeigt. Für die Simulation des VGA Controllers wurde der Testscreen Generator verwendet. Auf *Abb. 9* ist gut zu erkennen, dass der Testscreen Generator die Bilder entsprechend der Kamera im Interlace Modus erzeugt. Die Zähler des VGA Controllers sind allerdings auf die Verarbeitung von bereits deinterlaceten Bildern eingestellt, daher erscheinen im Zyklus des vertikalen Zählers zwei Teilbilder mit der Hälfte der Gesamtzeilenanzahl. Das Deinterlace Modul, das im entwickelten System die Zeilenverdopplung vornimmt, wurde für diese Simulation nicht berücksichtigt.

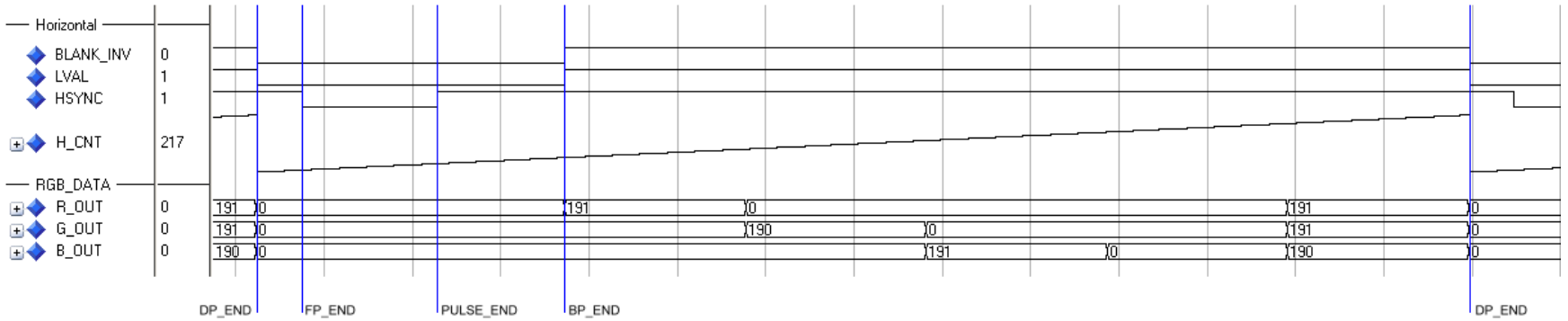


Abb. 8 : Horizontales Timing der VGA Controller Logik

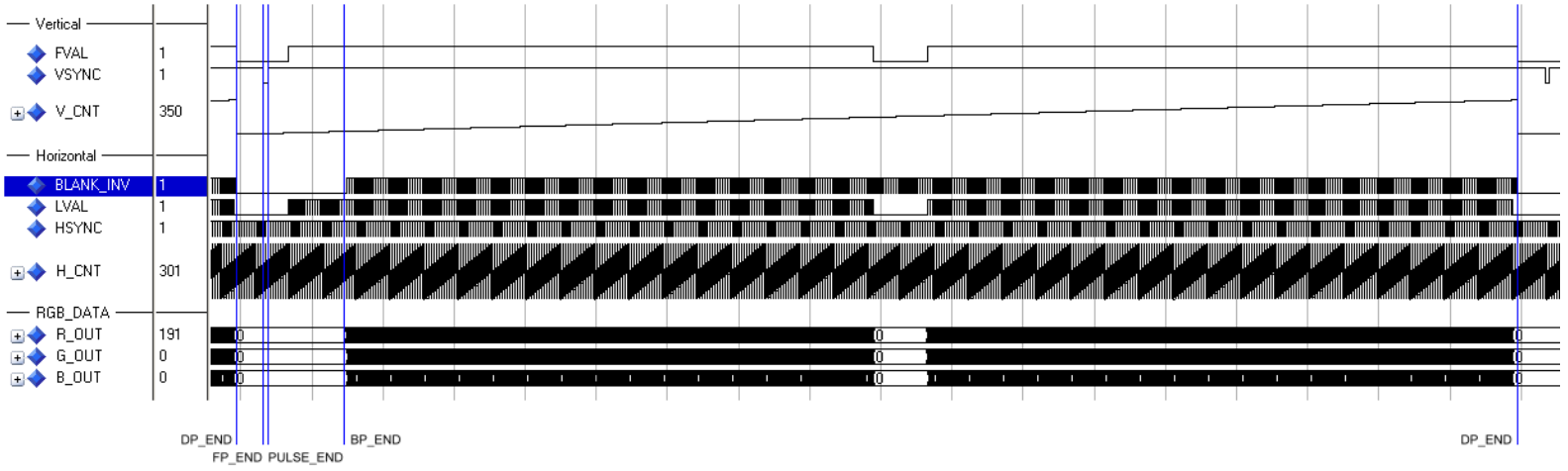


Abb. 9 : Vertikales Timing der VGA Controller Logik mit noch interlaced Bilddaten aus dem Testscreen Generator

2.2.4 Xilinx Microblaze System

Für die Implementierung von unterstützenden Modulen, die unabhängig von zeitlichen Vorgaben realisiert werden können, bietet sich die Verwendung eines µControllers an. Dadurch kann die Realisierungsdauer für diese zu Debug- und Testzwecken entworfenen Komponenten erheblich reduziert werden. Die im entwickelten System eingesetzte FPGA-basierte System-on-Chip Technologie bietet durch die Bereitstellung von integrierten Prozessoren die Möglichkeit des Hardware/Software Codesigns. In diesem Zusammenhang wird der Microblaze Softlogic-Prozessor verwendet, der auf allen FPGA Modellen der Firma Xilinx implementiert werden kann (vgl. *Abb 10*). Außerdem werden eine große Anzahl von vorgefertigten Systembausteinen, den sogenannten Intellectual Properties (IPs), bereitgestellt, die über das Embedded Development Kit (EDK) in ein Microblaze basiertes System integriert werden können. Die Firma Digilent stellt zudem für ihre mit Xilinx FPGAs versehenen Entwicklungsboards, die passenden IPs bereit, die neben den Hardwaretreibern auch Softwarebibliotheken zur Verwendung in C Programmen enthalten. Somit kann relativ einfach auf die Peripheriekomponenten des Entwicklungsboards zugegriffen werden.

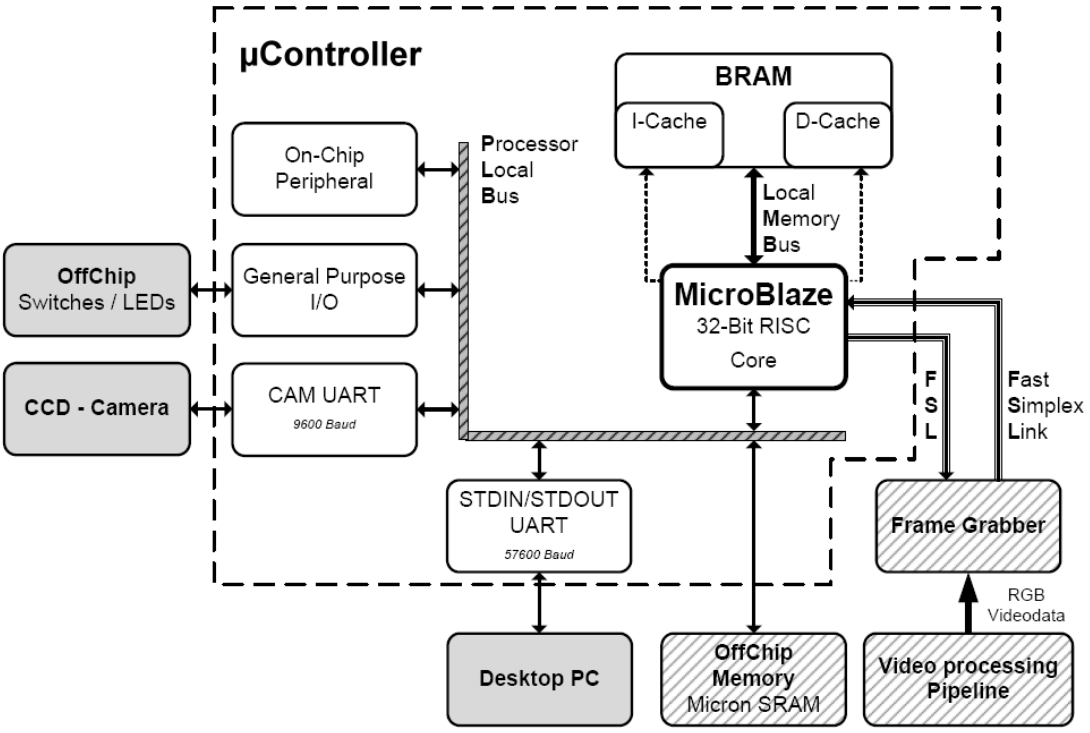


Abb. 10 : Komponenten und Schnittstellen des Microblaze-µController Systems

Das Microblaze System wurde überwiegend für die Realisierung der unterstützenden Komponenten (Kamera Parametrierung und FrameGrabber) verwendet, da somit der Zeitaufwand für die Entwicklung dieser Module erheblich verkürzt werden konnte.

Der Microblaze ist dabei über den PLB (Processor Local Bus) mit

- zwei UART Lite Schnittstellen
- dem 128 MBit Micron SRAM und
- den General Purpose Registern zur Verwendung der Ein- und Ausgabeperipherie des Nexys2 Boards verbunden.

Die UART Lite Schnittstellen werden zur Kommunikation mit der Sony Kamera und einem Desktop PC verwendet. Weiterhin kann die Kamera über diese Verbindung zur Laufzeit parametriert werden, um Einstellungen der Kamerafunktionen gegebenenfalls auf Umwelteinflüsse anzupassen. Dazu stehen eine Reihe von Kommandos zur Verfügung, die dem von Sony entwickelten VISCA Format entsprechen. Die Datenrate dieser Verbindung beträgt 9600 Baud. Die zweite UART Lite Schnittstelle dient zum einen als STDIN/STDOUT, wodurch Systemmeldungen auf ein Terminalprogramm eines angeschlossenen Desktop PCs ausgegeben werden können. Zum anderen wird über diese Verbindung die Bildübertragung für die Abspeicherung eines Bildes aus der Videopipeline mit einer Datenrate von 57600 Baud vorgenommen. Da diese Datenrate im Vergleich zu der Datenrate innerhalb der Videopipeline viel zu gering ist, um die Bilddaten direkt übertragen zu können, müssen die Bilddaten im Micron SRAM zwischengespeichert werden. Dazu kommt eine Kombination aus Hardware- und Softwaremodulen zum Einsatz.

Das Hardwaremodul reduziert die Bilddaten der Videopipeline von 24 Bit auf 8 Bit pro Pixelwert und überträgt diese in den Eingangsbuffer einer FSL (Fast Simplex Link) Verbindung. Da der FSL eine Datenbreite von 32- Bit unterstützt, werden jeweils vier Pixel einer Zeile über ein Schieberegister zusammengefasst, bevor sie in den Buffer geschrieben werden. Der FSL kann zudem mit zwei unterschiedlichen Taktraten betrieben werden; der Eingangsbuffer wird mit einer Frequenz von 13,5 MHz / 4 gefüllt, wogegen das Softwaremodul auf der Microblaze- Seite mit einer Frequenz von 50 MHz aus dem FSL lesen kann. Obwohl die Datenrate des PLB nicht eindeutig vorhersagbar ist, kann durch diese Eigenschaft gewährleistet werden, dass der Vorgang des Auslesens aus dem FSL und das Abspeichern der Daten in einer für diesen Zweck reservierten Speichersektion im SRAM ein genügend großes Zeitfenster zur Verfügung hat. Nachdem das gesamte Bild im SRAM abgelegt ist wird über die serielle Schnittstelle ein Bitmap Header übertragen und anschließend jede Zeile des Bildes zweimal gesendet. Die Zeilenverdopplung ist an dieser Stelle nötig, da die Bilddaten aus der Videopipeline das Deinterlace Modul noch nicht durchlaufen haben und somit noch im Interlace Format mit lediglich 240 Zeilen pro Bild vorliegen.

2.2.5 CORE Generator

Der CORE Generator liefert parametrierbare IPs, die für FPGAs der Firma Xilinx optimiert wurden. Darunter fallen FIFOs und Speicher, Encoder, Decoder, FIR-Filter oder auch Module zur Kommunikation wie Netzwerkschnittstellen. Im bestehenden System wurde der CORE Generator verwendet, um einen Digitalen Clock Manager für das Spartan3E FPGAs zu erstellen. Das Spartan 3E XC3S1200E FPGA besitzt insgesamt 8 DCMs, die alle in einem System verwendet werden können. Der DCM wird eingesetzt, um innerhalb der Prozesskette einen zweiten Systemtakt zu erzeugen. Dazu wird die 27 MHz Eingangsfrequenz der Kamera auf die Hälfte reduziert; das ist notwendig, da der Datenstrom bei der Umstellung des Abtastverhältnisses von 4:2:2 auf 4:4:4 verlangsamt werden muss.

Darüber hinaus sind die Schieberegister, die an verschiedenen Stellen des Systems zum Einsatz kommen, mit Hilfe des CORE Generators erzeugt worden [vgl. Shift Register, 2005]. Der CORE Generator bietet dem Entwickler die Möglichkeit, die zu erzeugenden IPs über eine Benutzeroberfläche zu parametrieren (vgl. *Abb. 11*). Anhand der Vorgaben wird dann das entsprechende IP, mit VHDL Quellcode und Netzliste erzeugt.

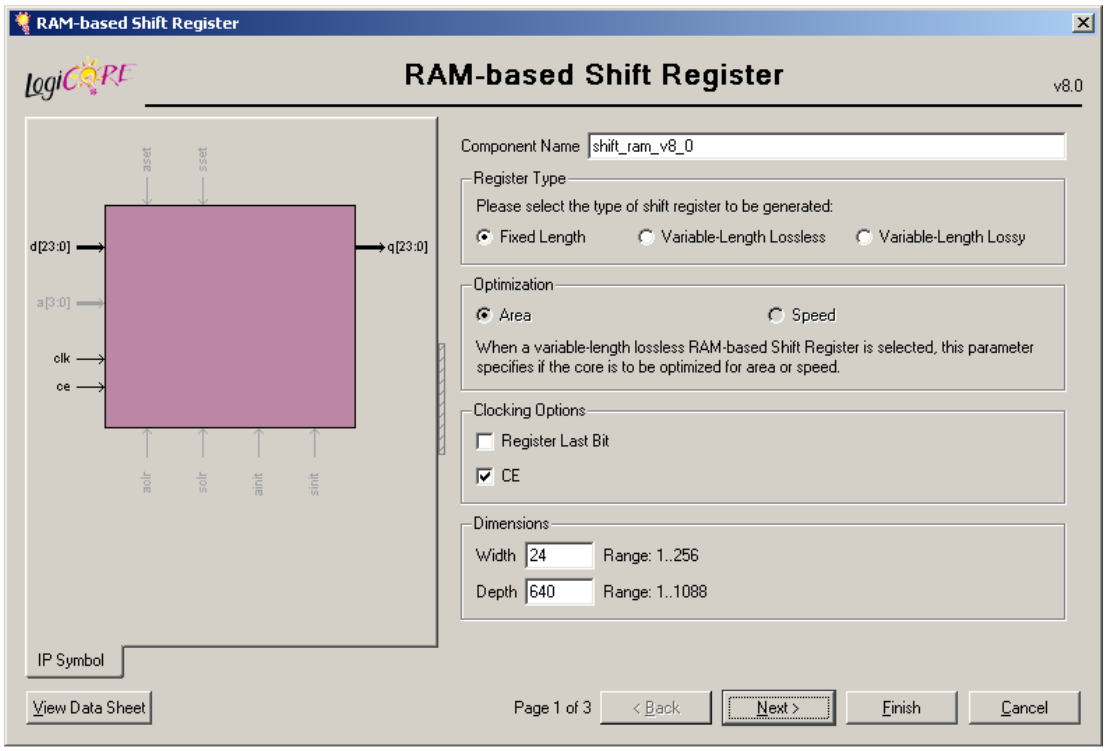


Abb. 11 : Dialog-basiertes Benutzerinterface des CORE Generators

3 Pipeline zur Aufbereitung und Darstellung der Videodaten

3.1 Komponenten der Bilddatenaufbereitung

Die Vorbereitung der Videodaten für die Verwendung in einer Fahrspurerkennung erfolgt in einer Verarbeitungskette, die aus mehreren RTL- Modulen (vgl. *Abb. 12*) besteht. Es werden zwei Taktraten innerhalb der Pipeline verwendet. Die Videoausgangsdaten werden mit einer Frequenz von 27 MHz von der Kamera bereitgestellt. Das Taktsignal wird in einen Digital Clock Manager geführt; dieser erzeugt die beiden verwendeten Systemtakte, in dem er das Eingangssignal zum einen direkt ausgibt und zum anderen die Taktrate halbiert. Durch das ITU Rec. BT656 Format ist es notwendig die ursprüngliche Frequenz am Ausgang der Deserialisierungsstufe auf 13,5 MHz zu verringern, da an dieser Stelle keine Informationen für die Chrominanzwerte jedes zweiten Pixels vorliegen. Die Ausgabe auf herkömmlichen VGA-Monitoren ist mit einer solchen geringen Taktfrequenz nicht möglich, da aber die Videodaten im Interlace Modus vorliegen, kann die Verringerung der Taktrate im Rahmen des Deinterlace Moduls behoben werden. Die Ausgabe der verdoppelten Bildzeilen kann wieder mit einer Frequenz von 27 MHz geschehen, womit sichergestellt ist, dass die Videodaten auf einem Monitor ausgegeben werden können.

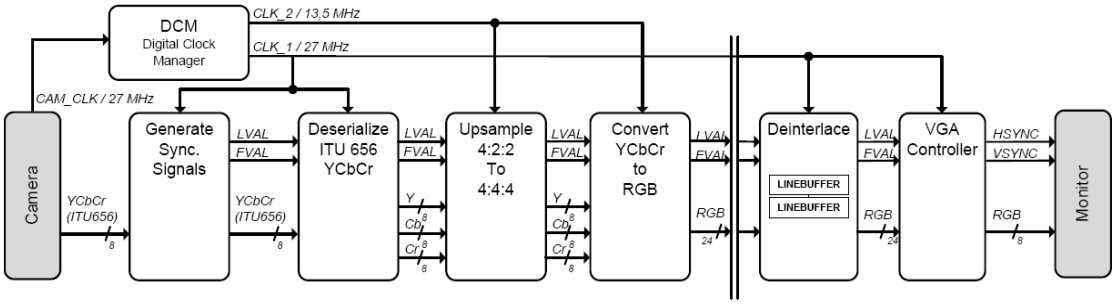


Abb. 12 : Aufbau der Videodatenpipeline

Die Videodatenpipeline wurde in Form eines eigenständigen IPs konstruiert, um diese in das µController System integrieren zu können. Somit kann die Pipeline mit geringem Aufwand auch in andere Systeme integriert werden, falls zum Beispiel die Hardwareanforderungen der verwendeten Plattform zu einem späteren Zeitpunkt nicht mehr genügen sollten.

3.1.1 Das Bilddatenformat nach ITU Rec. BT656

Die Kamera liefert die Bilddaten im YCbCr 4:2:2 Format. Das YCbCr Farbmodell ist eine leichte Abwandlung des YUV Farbmodells und wird vorrangig in der PAL Fernsehtechnik eingesetzt. Auch JPEG-Bilder oder MPEG-Videos verwenden dieses Farbmodell, da dort dessen Eigenschaften zur Datenreduktion verwendet werden können. Das Modell basiert, nicht wie zum Beispiel das RGB- Farbmodell, auf der Bestimmung des Farbwertes durch die Angabe von drei Grundfarben, sondern durch die Angaben von Helligkeit (Luminanz) und Farbigkeit (Chrominanz) (vgl. *Abb. 13*). Es nutzt dabei die Fähigkeit des menschlichen Auges aus, besser auf Luminanzunterschiede reagieren zu können, als auf Farbigkeitsunterschiede. Die relevanten Helligkeitsunterschiede werden in dem Y-Anteil des Farbmodells gespeichert; die Unterschiede der Farbigkeit entsprechend im Cr für die Farbigkeitsunterschiede in der Richtung von Rot zu Türkis und im Cb für die Farbigkeitsunterschiede von Blau zu Gelb.

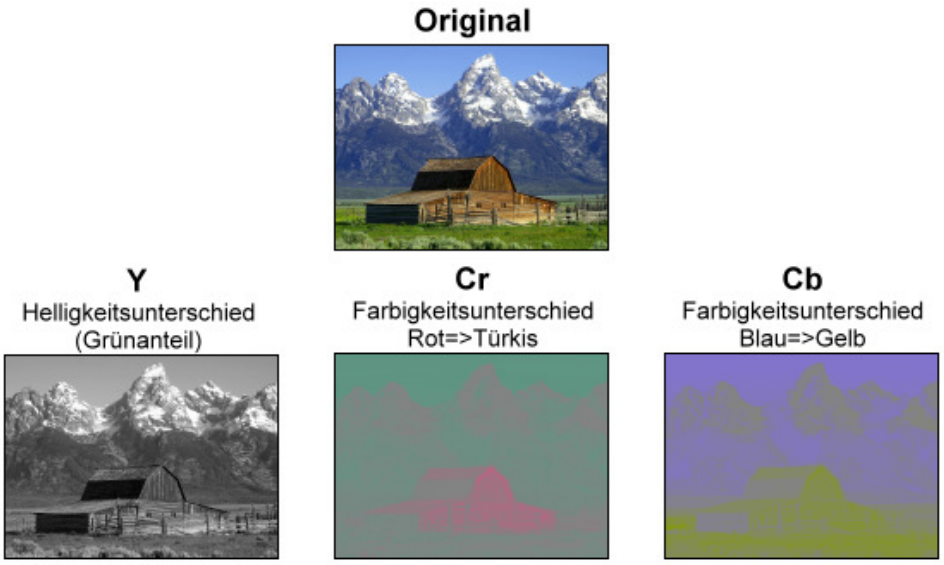


Abb. 13 : Aufteilung der Lumminaz- und Chrominanzanteil im YCrCb Format

Da die Chrominanzunterschiede weniger Gewicht in der Wahrnehmung des menschlichen Auges haben, können diese Anteil im YCbCr Datenformat reduziert werden, ohne große Qualitätsverluste hinnehmen zu müssen. Diese Reduktion wird durch das Abtastverhältnis 4:2:2 beschrieben, was bedeutet, dass die Chrominanzanteile nur für jeden zweiten Bildpunkt geliefert werden. Um die Werte im RGB Farbmodell darstellen zu können, muss eine Erweiterung in ein 4:4:4 Abtastverhältnis vorgenommen werden (vgl. *Tab. 4*).

4:2:2			
Y C_b C_r	Y	Y C_b C_r	Y
Y C_b C_r	Y	Y C_b C_r	Y

C_b,C_r werden nur für jeden zweiten Bildpunkt erfasst

4:4:4			
Y C_b C_r	Y C_b C_r	Y C_b C_r	Y C_b C_r
Y C_b C_r	Y C_b C_r	Y C_b C_r	Y C_b C_r

Y,C_b,C_r werden für jeden einzelnen Bildpunkt erfasst

4:1:1			
Y C_b C_r	Y	Y	Y
Y C_b C_r	Y	Y	Y

C_b,C_r werden nur für jeden vierten Bildpunkt erfasst

4:2:0			
Y C_b	Y	Y C_b	Y
Y C_r	Y	Y C_r	Y

Cb,Cr werden zeilenweise abwechselnd für jeden zweiten Bildpunkt erfasst

Tab. 4 : Verschiedene Formen von Abtastverhältnissen

Das Format des Ausgangsdatenstroms der Kamera mit dem Abtastverhältnis 4:2:2, ist in der ITU Rec. BT656 [ITU656, 1998] beschrieben. Diese sieht vor, dass innerhalb des Datenstroms neben den eigentlichen Videodaten auch Signale zur Synchronisation sowie Blankinganteile, in denen keine gültigen Daten übermittelt werden, vorhanden sind. Die Markierung der Blankingphasen erfolgt ebenfalls in den Synchronisationssignalen. In den Videodaten werden die Bildpunkte einer Bildzeile analog zum 4:2:2 Abtastverhältnis in der Form

Cb, Y, Cr, Y, Cb, Y, Cr, usw.

Zeile für Zeile übertragen. Zur Synchronisation werden die Start- (SAV – *start active video)* bzw. die Endpunkte (EAV – *end active video*) der gültigen Daten innerhalb der Bildzeile markiert. Dazu wurde der Wertebereich der Videodaten um die Werte 00_H (00000000_B) und FF_H (11111111_B) reduziert, um eine eindeutige Identifikation der Synchronisationssignale im Bilddatenstrom gewährleisten zu können.

Die Übertragung eines gesamten Bildes beginnt mit einer Blankingphase, in der mehrere Zeilen ohne relevante Bildinformationen übermittelt werden. Anhand der Blankingbits in den SAV und EAV Signalen kann diese identifiziert werden; folgt auf ein EAV mit Blanking Bit ein SAV ohne Blanking Bit, liegt die erste Zeile des relevanten Bildes vor.

Der Ausgangsbilddatenstrom im Interlace Modus in horizontaler und vertikaler Richtung wird in *Abb. 14* dargestellt. Dabei ist im Horizontalen Timing die Übertragung einer Bildzeile und im Vertikalen die Übertragung eines Gesamtbildes zu sehen.

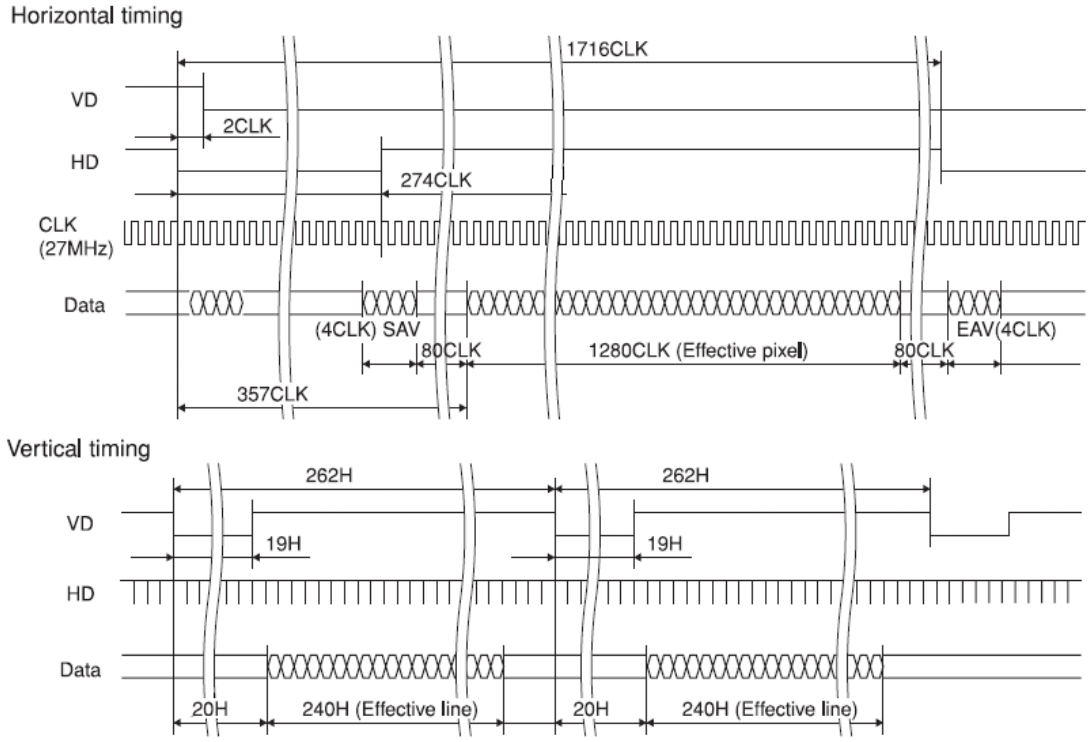


Abb. 14 : Darstellung der Übertragung einer Bildzeile und eines Gesamtbildes [vgl. Sony 2006]

Für den Versuch wird die Kamera im Interlace Modus verwendet. Dabei sind die beiden Fields ebenfalls durch eine Blankingphase von einander getrennt. Dadurch kann zwar die Übertragungsgeschwindigkeit gesteigert werden, da die Informationen eines Bildes in der Hälfte der Zeit übertragen werden, zum anderen muss das Gesamtbild vor der Ausgabe wieder deinterlaced werden. Dazu gibt es eine Reihe von Verfahren. Zum Beispiel können die Zeilen des ersten Fields als diejenigen vom Gesamtbild interpretiert werden, die eine gerade Zeilennummer haben, die Zeilen des zweiten Fields als die Zeilen mit einer ungeraden Zeilennummer. Beim Deinterlacen werden die einzelnen Zeilen der Fields abwechselnd zusammen geführt. Ein weiteres Verfahren, welches auch in der vorliegenden Arbeit verwendet wird, ist die Verdopplung der Zeilen eines jeden Fields. Dadurch erhält man aus den Bildinformationen eines Fields ein komplettes Bild. Für die Aufgabe der Kantenerkennung ist es vollkommen ausreichend den Interlace Modus zu verwenden, da nur über mehrere Zeilen zusammenhängende Strukturen erkannt werden sollen und somit innerhalb eines Gesamtbildes die markanten Bildpunkte einer einzelnen Zeile irrelevant sind. Außerdem bietet die Reduzierung des Datenvolumens einen weiteren Vorteil in der Laufzeitreduktion innerhalb der Prozesskette.

3.1.2 Synchronisation auf den Bilddatenstrom

Die Aufgabe des Synchronisationsmoduls ist die Erzeugung der vom VGA Controller benötigten LVAL (*line valid*) und FVAL (*frame valid*) Signale. Dabei wird das LVAL Signal für die Dauer einer Zeile mit gültigen Pixeldaten „high" gesetzt; das FVAL Signal entsprechend für die Dauer eines gültigen Bildes. Somit erkennt der VGA Controller das gültige Bilddaten anliegen, wenn beide Signale zeitgleich „high" sind. Um diese Signale erzeugen zu können, müssen zum einen die zeitlichen Steuerungsinformationen des Bilddatenstroms (SAV/EAV) interpretiert und zum anderen Zähler eingesetzt werden. Dabei handelt es sich um einen Zähler für die Anzahl der gültigen Pixel (Bytes) innerhalb einer Zeile (H_CNT) und einen Zweiten der die Zeilen zählt (V_CNT). Das ist nötigt, damit die FVAL und LVAL Signale am Ende einer Zeile bzw. eines Bildes taktgenau auf „low" zurückgesetzt werden können. Die SAV/EAV Anteile bestehen aus einer FF_H 00_H 00_H XY_H Sequenz, wobei die Präambel FF_H 00_H 00_H eindeutig identifizierbar ist(vgl. *Tab 5*).

BIT Nr	1. Wort (FF)	2. Wort (00)	3. Wort (00)	4. Wort (XY)
7(MSB)	1	0	0	1
6	1	0	0	F
5	1	0	0	V
4	1	0	0	H
3	1	0	0	P_3
2	1	0	0	P_2
1	1	0	0	P_1
0	1	0	0	P_0

F	=	0 im ersten Field	
		1 im zweiten Field	
V	=	1 im Blanking (Bereich zwischen zwei Frames)	
H	=	0 im SAV	
		1 im EAV	
P_3	=	V *XOR* H	Protection bits
P_2	=	F *XOR* H	
P_1	=	F *XOR* V	
P_0	=	F *XOR* V *XOR* H	

Tab. 5 : Aufbau der Symchronisationsanteile innerhalb des Bidldatenstroms

Für die Erkennung der Sequenzen sind insgesamt acht Zustände nötig. Ein Startzustand, drei Zustände für den Empfang der ersten drei Worte (FF_H 00_H 00_H) und weitere vier Zustände die sich aus den Kombinationen des vierten und des fünften Bits (V und H) des vierten Wortes (XY_H) ergeben. Anhand dieser beiden Bits kann man die aktuelle Position innerhalb des Datenstroms eindeutig bestimmen (vgl. *Tab 6*).

4. Wort	SAV	EAV	SAV im Blanking	EAV im Blanking
1 F **V H** P_3 P_2 P_1 P_0	1 0 **0 0** 0 0 0 0	1 0 **0 1** 1 1 0 1	1 0 **1 0** 1 0 1 1	1 0 **1 1** 0 1 1 0
	80_H	$9D_H$	AB_H	$B6_H$
				bei F=0 (innerhalb des ersten Fields)

Tab. 6 : Zusammensetzung der Positionsbestimmungsbits in den Synchronisationsdaten

Aus den oben genannten Zuständen ergibt sich das in *Abb. 15*. dargestellte Zustandsdiagramm:

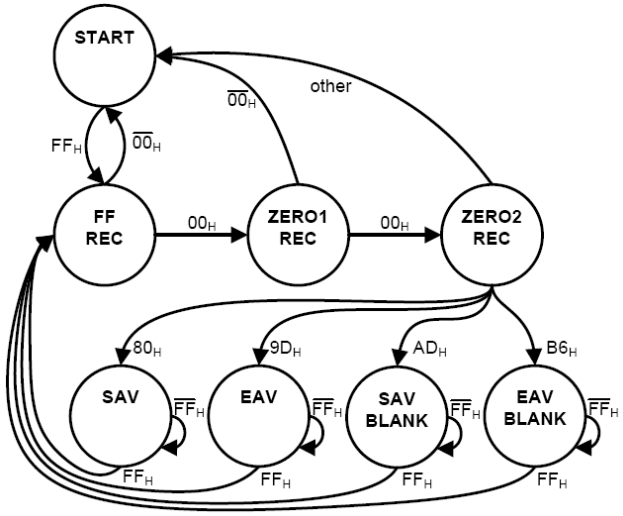


Abb. 15 : Zustandsdiagramm des Synchronisationsmoduls

Der Anfang eines neuen Bildes wird durch die Abfolge, Empfang „SAV“ Sequenz nach einer „EAV im Blanking“ Sequenz signalisiert und das Inkrementieren der Zähler wird gestartet. Dabei wird innerhalb des SAV Zustands, nach Ablauf der ersten 80 Blanktakte, H_CNT solange erhöht bis H_CNT_MAX (1280 Pixel) erreicht wird. Bei Verlassen des Zustands wird V_CNT erhöht bis V_CNT_MAX (240 Zeilen im Interlace Modus) erreicht wird. Zusätzlich werden an dieser Stelle die Synchronisationssignale LVAL und FVAL entsprechend der Zählerstände gesetzt.

```
--************SAV*****************************************************************
   when SAV =>
      If (SAV_CNT < SAV_CNT_MAX) then
         -- count the 80 Blank clocks
         SAV_CNT_EN <= '1';
      else
         if (H_CNT < H_CNT_MAX) and (V_CNT < V_CNT_MAX) then
            -- enable horizontal count incremental
            H_CNT_EN <= '1';
            FVAL <= '1'; -- set frame valid
            LVAL <= '1'; -- set line valid
         else
            LVAL <= '0'; -- H_CNT_MAX reached, line invalid
            if (V_CNT >= V_CNT_MAX-1) and (H_CNT >= H_CNT_MAX) then
               FVAL <= '0'; -- V_CNT_MAX reached, frame invalid
            end if;
         end if;
      end if;
      if DATA_INT = "11111111" then  --FF received
         --change state and enable vertical count incremental
      V_CNT_EN <= '1';
         NEXT_STATE <= FF_REC;
      else
         NEXT_STATE <= SAV;
      end if;
```

Code 3 : Auszug aus der FSM des Synchronisationsmoduls

3.1.3 Umwandlung der seriellen Bilddaten

Nachdem die Synchronisationssignale erzeugt wurden, müssen die Bilddaten aus dem 8Bit breiten seriell anliegenden YCbCr Format in ein paralleles, 24Bit breites RGB Format konvertiert werden. Nach ITU-Rec. BT.656 werden die Bilddaten in der Reihenfolge C_{b0}, Y_0, C_{r0}, Y_1, C_{b2}, Y_2, C_{r2}, Y_3, usw. übertragen. Zunächst muss sichergestellt werden, dass am Ausgang der Parallelisierungsmoduls ein 24 Bit breiter Datenstrom entsteht, wobei dieser durch die drei Ausgangsvektoren Y_OUT, CB_OUT, CR_OUT realisierte ist (vgl. *Abb. 16*). Das 4:2:2 Abtastverhältnis liefert nur für jeden zweiten Luminazwert des Videodatenstroms die entsprechenden Chrominanzwerte, daher können die Ausgangssignale nach dem Parallelisierungsprozess nur mit der halbierten Taktrate von 13,5 MHz bereitgestellt werden. Die Synchronisationssignale FVAL und LVAL werden innerhalb der Demultiplexermoduls in Registern zwischengespeichert, um an den Ausgängen taktrichtig zu den entsprechenden YCbCr Werten ausgegeben werden zu können.

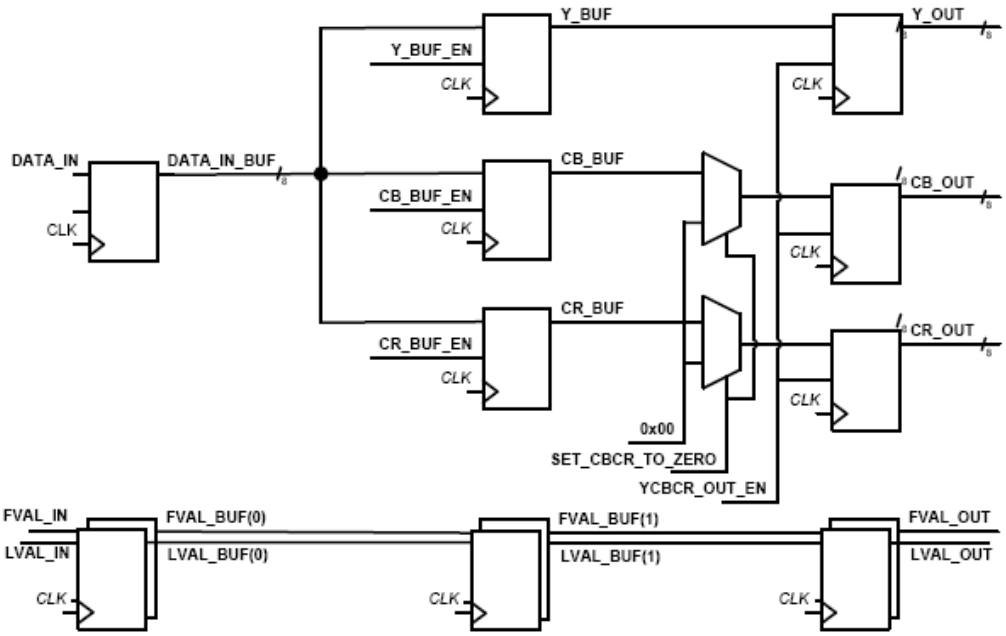


Abb. 16 : Blockschaltbild zur Erzeugung der Y_OUT, CB_OUT und CR_OUT Signale im Parallelisierungsprozess

Die Eingangssignale liegen mit der von der Kamera bereitgestellten Frequenz von 27 MHz an den Ports DATA_IN, FVAL_IN und LVAL_IN an. Die Chrominanz Signale, CB_OUT und CR_OUT, zu denen keine Werte im Datenstrom verfügbar sind, werden zunächst durch Nullen aufgefüllt und in einem späteren Schritt innerhalb der Prozessorkette durch Interpolationen ersetzt. Zur Erweiterung des Datenstroms genügt es, die einzelnen Zeilen des Bilddatenstroms zu betrachten; es muss daher nur der Eingang der gültigen Pixeldaten innerhalb einer Zeile, also das LVAL Synchronisationssignal betrachtet werden.

Der Automat, der den Videodatenstrom auf 24 Bit erweitert, besteht somit aus fünf Zuständen; ein Zustand, der den Anfang einer gültigen Zeile durch das LVAL Signal erkennt, und jeweils einen Zustand für die empfangenen C_{bi}, Y_i, C_{ri} und Y_{i+1}.

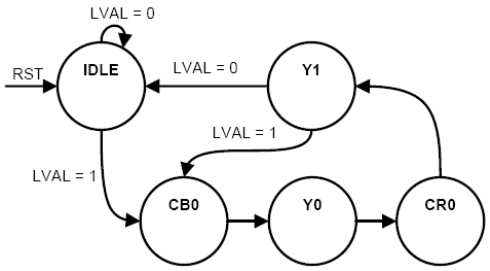


Abb. 17 : Zustandsdiagramm des Parallelisierungsmoduls

Am Ende dieser Verarbeitungsstufe werden für jeden Pixelwert im Datenstrom drei Farbinformationen zur Verfügung gestellt, was in der Form dem RGB Format entspricht. Um diese zu erfüllen, müssen im nächsten Schritt die fehlenden Chrominanzanteile aufgefüllt werden und anschließend die Y-, Cb- und Cr- Werte in die äquivalenten RGB Werte umgerechnet werden.

3.1.4 Das Chroma Upsampling- und das Konvertermodul

Am Ende der Parallelisierungsstufe steht, neben den beiden LVAL und FVAL Synchronisationssignalen, ein 24 Bit breiter Pixeldatenstrom zur Verfügung, der der Form in *Tab. 7* entspricht.

1 Bit	LVAL	LVAL	LVAL	LVAL	LVAL	usw.
1 Bit	FVAL	FVAL	FVAL	FVAL	FVAL	
24 Bit	Y0	Y1	Y2	Y3	Y4	
	Cr0	0	Cr2	0	Cr4	
	Cb0	0	Cb2	0	Cb4	

Tab. 7 : Format des Bilddatenstroms am Eingang des Chroma-Upsamplingmoduls

Für die Umrechnung der YCbCr- Daten des Videodatenstromes in die entsprechenden RGB Werte, müssen die Informationen zu den fehlenden Chrominanzwerten aufgefüllt werden. Zur Ermittlung der fehlenden Informationen gibt es verschiedene Ansätze. In der vorliegenden Arbeit werden diese durch Pixel Replikation ermittelt. Der Aufbau und die Implementierung des Chroma-Upsamplingmoduls ist detailiert in [Peters, 2009] beschrieben.

Nachdem jeder Bildpunkt durch die Angabe von drei 8 Bit Werten beschrieben ist, kann die Umrechnung der YCbCr Werte in RGB Werte erfolgen. Dazu wird ein Konvertierungsmodul eingesetzt das anhand der *Gleichungen 3.1*, *3.2* und *3.3* die RGB Werte ermittelt [vgl. Jack, 2005]**.**

$$R = 1{,}164\,(Y - 16) + 1{,}596(Cr - 128) \tag{3.1}$$

$$G = 1{,}164(Y - 16) - 0{,}813(Cr - 128) - 0{,}391(Cb - 128) \tag{3.2}$$

$$B = 1{,}164(Y - 16) + 2{,}018(Cb - 128) \tag{3.3}$$

Für die Implementierung der oben genannten Gleichungen auf der Entwicklungsplattform, werden diese in Integer Arithmetik umgesetzt. Die oben angegebenen Umrechnungen der YCbCr Werte können innerhalb eines Taktzyklus‘ durchgeführt werden, wodurch die Synchronisationssignale nicht zwischen-gespeichert werden müssen [vgl. Peters, 2009]**.**

3.1.5 Deinterlacen der Bilddaten

Für die weitere Bildbearbeitung zur Kantenerkennung genügt es den Datenstrom im Interlaced Format zu belassen. Zur Ausgabe auf einem Monitor oder der Abspeicherung in einer Bitmap Datei müssen die Zeilen des Datenstroms allerdings auf ein Format von 480 Zeilen deinterlaced werden. Dazu werden die einzelnen Zeilen in einem Deinterlace Modul verdoppelt (vgl. *Abb. 18*). Die einfache Verdopplung der Zeilen hat für die visuelle Wahrnehmung keine relevanten Nachteile und erfordert keine zusätzlichen Rechenoperationen, die die Verarbeitungslaufzeit oder die Auslastung des zur Verfügung stehenden Speichers unnötig belasten würden.

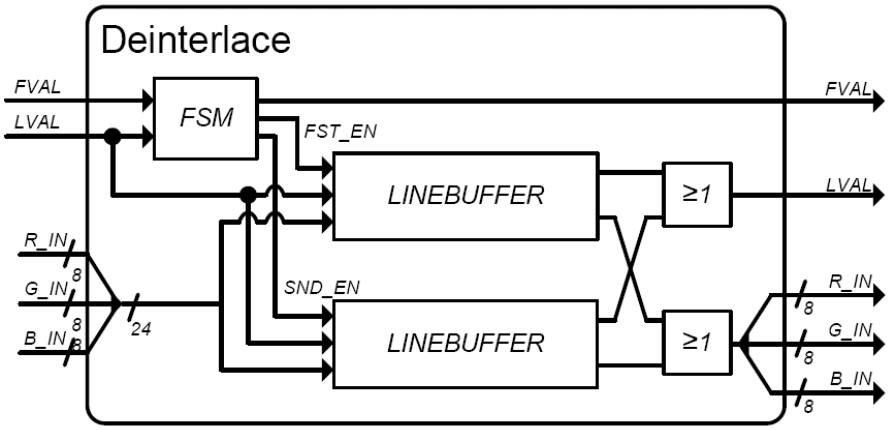


Abb. 18 : Blockschaltbild des Deinterlacemoduls mit den LINEBUFFER Komponenten, die das Zwischenspeichern der Bildzeilen vornehmen

Das Deinterlace Modul besteht aus zwei identischen LINEBUFFER Komponenten die abwechselnd die Bildzeilen zwischenspeichern bzw. ausgeben. Für die Speicherung der Bildzeilen wird in jedem LINEBUFFER ein Schieberegister verwendet, welches durch den Xilinx CORE Generator erstellt wurde. Dieses Schieberegister kann 640 Bildpunkte der 24 Bit breiten Eingangsdaten aufnehmen, womit jeweils eine gesamte Zeile abgespeichert werden kann. Anhand der Synchronisationssignale LVAL und FVAL wird erkannt, zu welchem Zeitpunkt eine gültige Bildzeile anliegt und das Schieberegister des ersten LINEBUFFERs mit der Eingangsfrequenz von 13,5 MHz gefüllt (vgl. *Abb. 19*). Sobald die erste Zeile innerhalb des Schieberegisters ist, wechselt das Deinterlace Modul die Betriebsarten der LINEBUFFER; der erste gibt die soeben gespeicherte Zeile mit der doppelten Frequenz von 27 MHz zweimal aus, während der zweite LINEBUFFER die neue Bildzeile des Eingangsdatenstrom in sein Schiebregister aufnimmt. Sobald die Synchronisationssignale den Beginn einer neuen Zeile angeben, werden die Betriebsarten wieder gewechselt.

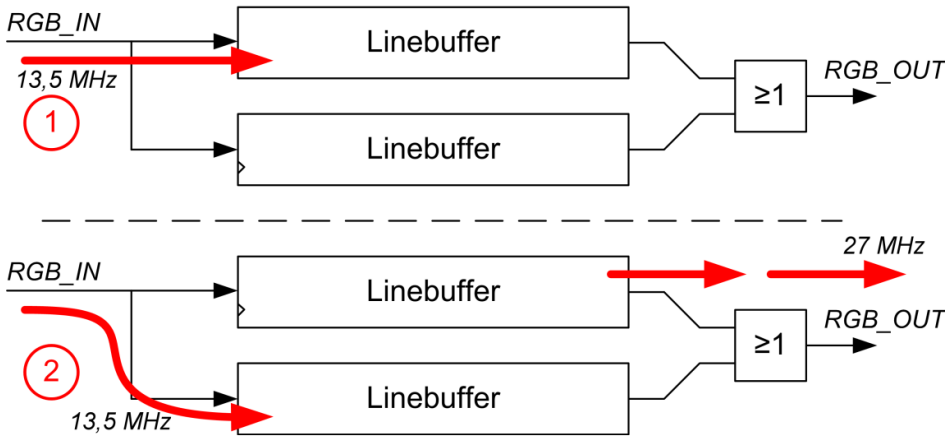


Abb. 19 : Ablauf der Speicherung und Ausgabe einer Bildzeile

Für die Erzeugung der unterschiedlichen Taktraten für das Speichern und das Ausgeben der Zeilen, werden die LINEBUFFER zunächst mit einem Eingangstakt von 27 MHz vom Clock Manager versorgt. Innerhalb der LINEBUFFER wird für das Schreiben in das Schieberegister eine eigene Clock erzeugt, die mit der halben Frequenz arbeitet (vgl. *Code 4*).

```
process (CLK,RST) -- clock enable at half speed
   begin
      if RST='1' then
         LOW_CLK<= '0';
      elsif rising_edge(CLK) then
         LOW_CLK<= not LOW_CLK;
      end if;
   end process;
```

Code 4 : Erzeugung der halbierten Taktfrequenz im Deinterlacemodul

Das LOW_CLK Signal wird während der Speicherung der Bilddaten direkt an den ENABLE Eingang des Schieberegisters angelegt; während des Auslesens wird die Eingangsclock CLK verwendet. Wie in der gesamten Prozesskette, signalisieren die Synchronisationssignale lediglich, dass genau zu diesem Zeitpunkt relevante Daten vorliegen. Aus diesem Grund ist wiederrum der Einsatz von Zählern notwendig, um das Füllen bzw. Leeren der Schieberegister zum korrekten Zeitpunkt beenden zu können. Dabei werden die Zähler so verwendet, dass sie die kompletten Takte einer Bildzeilenübertragung, inklusive Blankinganteilen, mitzählen. Für die Ausgabe wird der Zähler entsprechend 1716 Takte einer Bildzeile weiter inkrementiert, so dass eine vollständige Sequenz immer aus Schreiben in das Schieberegister und zweimaligen Ausgeben besteht. Die Werte am Ausgang des Schieberegisters werden dabei zurück an den Eingang geführt, damit sie ein zweites Mal ausgegeben werden können (vgl. *Code 5*).

```
case STATE is
         when IDLE =>
            if ENABLE = '1' then
               STATE_N <= RUNNING;
            end if;
            CNT_N <= 0;
         when RUNNING =>
            CNT_N <= CNT + 1;
            if CNT < LB_IN_BLANK then      -- slow data in
               SHIFT_EN <= LOW_CLK;
            elsif CNT < LB_OUT_START1 then -- blank in input
               null;
            elsif CNT < LB_OUT_BLANK then  -- first data out
               LVAL_OUT <= '1';
               SHIFT_EN <= '1';
               D_IN <= D_OUT;              -- fill shift reg with outdata again
               DATA_OUT <= D_OUT;
            elsif CNT < LB_OUT_START2 then -- blank in output
               null;
            elsif CNT = LB_OUT_START2 then
               SHIFT_EN <= '1';
               D_IN <= D_OUT;
               DATA_OUT <= D_OUT;
            elsif CNT < LB_CNT_MAX then    --second data out
               LVAL_OUT <= '1';
               SHIFT_EN <= '1';
               D_IN <= D_OUT;
               DATA_OUT <= D_OUT;
            Else                           -- last blank; restart
               STATE_N <= IDLE;
            end if;
      end case;
```

Code 5 : Lesen und Schreiben des Schiebregisters im LINEBUFFER Modul

Durch die Zeilenverdopplung wird das ursprünglich 240 Zeilen umfassende Bild, auf ein für den Monitor darstellbares Format gebracht. Die Auswirkungen auf die Wahrnehmung des menschlichen Auges sind durch die Verdopplung der Zeilen vernachlässigbar, da die Breite einer Pixelzeile, abhängig von der Auflösung und der Größe des Ausgabegerätes, immer noch unter dem Millimeter Bereich liegt, und somit keine zusätzlich Details erkannt werden können (vgl. *Abb. 20*).

Abb. 20 : Bild nach der Zeilenverdopplung im Deinterlace Modul und das Originalbild innerhalb der Videopipeline

3.2 Unterstützende Hardware- Software Komponenten

3.2.1 Der Testscreen Generator im Interlaced format

Zu Beginn der Implementierung wurde ein Testscreen Generator entwickelt, um die Prozesskette auch ohne Datenversorgung durch die Kamera zu testen und entsprechende Timing Simulationen mit den entwickelten Modulen durchführen zu können. Dazu wurde eine Testbench anhand des Kamera Timings (vgl. *3.1.1*) parametriert und in einer entsprechenden VHDL Komponente an den Anfang der Prozesskette gestellt. Die Kernaufgabe dieses Moduls ist die Erzeugung einer Bildzeile, die kontinuierlich ausgegeben wird. Die Bildzeile enthält gleichmäßig verteilte Farbbereiche von 128 Pixeln in den Farben Rot, Grün, Blau, Schwarz und Weiß. Durch die wiederholende Ausgabe der Zeile entsteht ein Gesamtbild mit fünf einheitlich großen Balken in den genannten Farbwerten (vgl. *Abb. 21*).

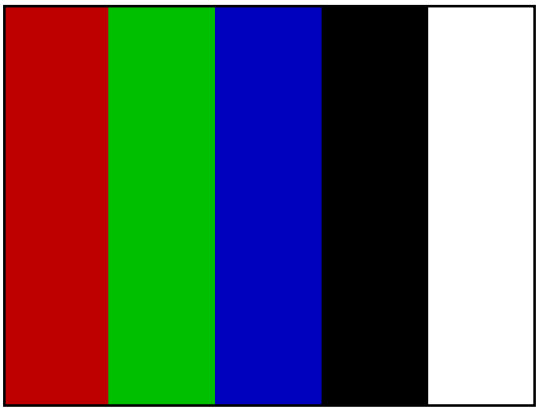


Abb. 21 : Erzeugtes Testbild des Testscreen Generators

Das Testscreen Modul enthält einen horizontalen Zähler (HPOS), für die Bildpunkte in einer Zeile, und eine vertikalen Zähler(VPOS) für die Bildzeilen. Dem Kamera Datenblatt wurde entnommen, dass die Übermittlung einer Bildzeile inklusive Blanking- Anteilen 1716 Takte dauert; demnach inkrementiert HPOS bis 1715. Die Kamera wird wie beschrieben im Interlace Modus betrieben, daher genügt es VPOS bis 262 zu inkrementieren, weil die Übertragungsweisen der beiden Teilbilder identisch ablaufen. Eine Simulation die das Ausgabeformat des Testscreen Generators dokumentiert ist in den *Abbildungen 8* und *9* zu finden.

Der Ablauf der Erzeugung eines Gesamtbildes wird zyklisch in folgender Form vorgenommen:

Zuerst werden in einer Blankingphase 20 Zeilen erzeugt, die jeweils ein SAV und EAV mit gesetztem Blanking Bit enthalten. Nach der 20. Zeile wird das erste SAV ohne Blanking Bit erzeugt. Da der erzeugte Datenstrom dem Interlace Modus angepasst ist (vgl. *Abb. 13*), wird ein weiteres Blanking für die Dauer von 80 Takten erzeugt. Die eigentlichen Farbinformationen werden dann für jeweils 256 Takte, was 128 Pixeln entspricht, im YCbCr Format erzeugt. Somit erhält man für 1280 Takte gültige Pixeldaten. Nach wiederum 80 Takten wird das EAV gesetzt. Auf diese Weise werden alle 240 Zeilen erzeugt. Nach der Übertragung des EAV der letzten Zeile, wird wieder eine Blankingphase von zwei Zeilen eingefügt, bevor die Erzeugung eines neuen Bildes beginnt und der Ablauf erneut startet. Für die Generierung der Pixeldaten wird eine Hilfsvariable eingesetzt, die nur die unteren beiden Bits des HPOS Zählers betrachtet. Anhand dieser kann eindeutig identifiziert werden, welcher Wert erzeugt werden muss.

```
pixelgen: process (hpos,vpos)
variable tmp : std_logic_vector(1 downto 0);
begin
   tmp := hpos(1 downto 0);
   if vpos>=20 and vpos<260 then
      --sav--
      if hpos >=273 and hpos<277 then
         if tmp = "01" then data_out<= "11111111";
         elsif tmp = "10" then data_out<= "00000000";
         elsif tmp = "11" then data_out<= "00000000";
         elsif tmp = "00" then data_out<= "10000000";
         end if;
      --red
      elsif hpos>=357 and hpos<613 then
         if tmp = "01" then data_out<= "01100100";     --cb
         elsif tmp = "10" then data_out<= "01000001";  --y
         elsif tmp = "11" then data_out<= "11010100";  --cr
         elsif tmp = "00" then data_out<= "01000001";  --y
         end if;
```

Code 6 : Erzeugung des SAV und des Rotanteils im Testscreen Generator

3.2.2 Parametrierung der Kamera

Die Kamera kann auf verschiedene Weise parametriert werden, um die Aufnahmeeigenschaften auf unterschiedliche Umgebungssituationen anzupassen. Sie bietet zum Beispiel die Möglichkeit, Funktionen wie

- automatische Fokussierung
- Zoom Tele/Wide
- Weißabgleich
- Belichtungsautomatik
- Blendeneinstellungen
- oder das Ausblenden von Bildbereichen

zu verwenden. Diese Funktionalitäten können über die RS232 Schnittstelle der Kamera parametriert werden. Zur Parametrierung wird das von Sony bereit gestellte VISCA Protokoll verwendet, das seinen Einsatz in einer Reihe von Sony Produkten findet. Im Rahmen dieser Arbeit wurde eine Schnittstelle implementiert, mit der man diese Parametrierung vornehmen kann. Die eigentliche Einstellung der Kamera und die Analyse der am besten zu verwendenden Parameter, ist nicht Bestandteil dieser Arbeit und sollte in einem weiteren Schritt erfolgen.

Für die Aufgabe wird eine der UART Komponenten des Microprozessor Systems genutzt, welche die Kommunikation mit der Kamera vornimmt. Die Schnittstellensignale der Komponente sind mit den Pmod I/0 Pins des Nexys2 verbunden; diese sind direkt an die RS232 Schnittstelle der Kamera angeschlossen. Durch den Einsatz des System-on-Chip kann die Implementierung der Kommunikation in Form eines in der Programmiersprache C geschriebenen Softwaremoduls gelöst werden, diese kann über die Standardfunktionen der UARTLite Bibliothek auf die Komponente zugreifen.

Die UARTLite Komponente wird bei Systemstart zunächst durch die Funktion `XUartLite_Initialize()` initialisiert und die Eingangs- bzw. Ausgangs FIFOs zurückgesetzt. Nun können innerhalb des Softwaremoduls die Funktionen `XUartLite_Send()` und `XUartLite_Recv()` zur Übertragung der VISCA Befehle bzw. zum Empfang der Statusnachrichten von der Kamera verwendet werden.

Nachfolgend wird ein Auszug aus der `main()` Funktion der erstellten Software dargestellt, der verdeutlicht wie das UART Interface initialisiert wird und ein Beispiel Kommando, in diesem Fall wird das Ausgabebild der Kamera auf den Kopf gestellt, versendet wird (vgl. *Code 7*). Dazu wird ein Schiebeschalter auf dem Nexys2 Board verwendet, dessen Status innerhalb der Hauptschleife abgefragt wird.

```
XUartLite cam_uart;
char swtch = 0, isFlipped = 0, led = 0;
print("-- Entering main() --\r\n");

//initialize switches on Nexys2 board
XGpio_mSetDataDirection(XPAR_SWITCHES_8BIT_BASEADDR,XGPIO_IR_CH1_MASK,0xFF);
```

```
//initialize LEDS
XGpio_mSetDataDirection(XPAR_LEDS_8BIT_BASEADDR, XGPIO_IR_CH1_MASK, 0x00);

//initialize UART devices
if (XUartLite_Initialize(&cam_uart,XPAR_CAM_UART_DEVICE_ID)!= XST_SUCCESS)
{
   print("CAM_UART: init failed!\r\n");
   return;
}
//reset/ clear UART FIFOS
XUartLite_ResetFifos(&cam_uart);
//configure the camera display function- turn off
XUartLite_Send(&cam_uart,cam_titleDisplayOFF,6);
XUartLite_Send(&cam_uart,cam_titleClear,6);

//main loop
while(1)
{
   //get states of switch register
   swtch = XGpio_mReadReg(XPAR_SWITCHES_8BIT_BASEADDR, 0x00);

   // The Flipper (switch 0):
   //sends VISCA command to Sony camera to flip display
   if( (swtch & 0x01) && !isFlipped){
      isFlipped = 1;
      XUartLite_Send(&cam_uart,cam_flip_on,6);
      led = XGpio_mGetDataReg(XPAR_LEDS_8BIT_BASEADDR,1);
      XGpio_mSetDataReg(XPAR_LEDS_8BIT_BASEADDR, 1, (led | 0x01) );
      print("CAM_UART: screen flipped!\r\n");
   } else if( !(swtch & 0x01) && isFlipped) {
      isFlipped = 0;
      XUartLite_Send(&cam_uart,cam_flip_off,6);
      led = XGpio_mGetDataReg(XPAR_LEDS_8BIT_BASEADDR,1);
      XGpio_mSetDataReg(XPAR_LEDS_8BIT_BASEADDR, 1, (led & 0xFE) );
      print("CAM_UART: screen unflipped!\r\n");
   }

};
print("-- Exiting main() --\r\n");

return 0;
```

Code 7 : Initialisierung des UART Schnittstelle und Übertragung eines VISCA Kommandos an die Kamera

Für die Datenübertragung zur Kamera wird die UART Verbindung gemäß des Datenblatts der Kamera [Sony, 2006] auf 9600 Baud, 8 Datenbits, 1 Startbit, ohne Paritätsbit (8N1) konfiguriert. Der Funktion `XUartLite_Initialize()` werden über den Parameter `XPAR_CAM_UART_DEVICE_ID` die entsprechenden Informationen zur Parametrierung übergeben. Diese Informationen können über das EDK für die benötigten Zwecke angepasst werden.

```
/* Definitions for peripheral CAM_UART */
#define XPAR_CAM_UART_BASEADDR      0x84020000
#define XPAR_CAM_UART_HIGHADDR      0x8402FFFF
#define XPAR_CAM_UART_DEVICE_ID     2
#define XPAR_CAM_UART_BAUDRATE      9600
#define XPAR_CAM_UART_USE_PARITY    0
#define XPAR_CAM_UART_ODD_PARITY    0
#define XPAR_CAM_UART_DATA_BITS     8
```

3.2.3 Bildaufzeichnung mit dem FrameGrabber

Das entwickelte FrameGrabber Modul wird zur Dokumentation des Bilddatenstroms innerhalb der Prozessorkette eingesetzt. Damit erhält man in Verbindung mit der Darstellung auf dem Monitor, die Möglichkeit, die einzelnen Elemente der Pipeline zu verifizieren und mögliche Fehlerquellen einzugrenzen. Das FrameGrabber Modul ist so konzipiert, dass es an beliebigen Stellen zwischen die Elemente der Prozesskette eingefügt werden kann; allerdings muss sich diese Stelle vor dem Deinterlace Modul befinden, da innerhalb des Software Moduls vor dem Senden über die serielle Schnittstelle ebenfalls eine Zeilenverdopplung vorgenommen wird.

Das Modul besteht aus einer VHDL Komponente, die den Datenstrom über den Fast Simplex Link [FSL, 2007] in das Microblaze System transportiert und einer in C geschriebenen Softwarekomponente, welche die Daten empfängt und im Micron SRAM ablegt (vgl. *Abb. 22*). Die Verbindung des SRAMs mit dem Microblaze wird hierbei über den PLB hergestellt. Eine direkte Anbindung der VHDL Komponente an den PLB und somit die Einsparung des FSL, kann bei der hohen Datenrate innerhalb der Pipeline nicht erfolgen, da der PLB von mehreren Teilnehmern im System verwendet wird, und somit die Arbitrierungszeiten nicht vorhergesagt werden können. Nachdem die Daten im SRAM abgelegt sind wird ein standardisierter Bitmap Header [Wikipedia, 2009a] vor die Rohdaten eingefügt und die so erzeugte Bitmap Datei über die UART Schnittstelle des Microblaze an einen Desktop PC gesendet.

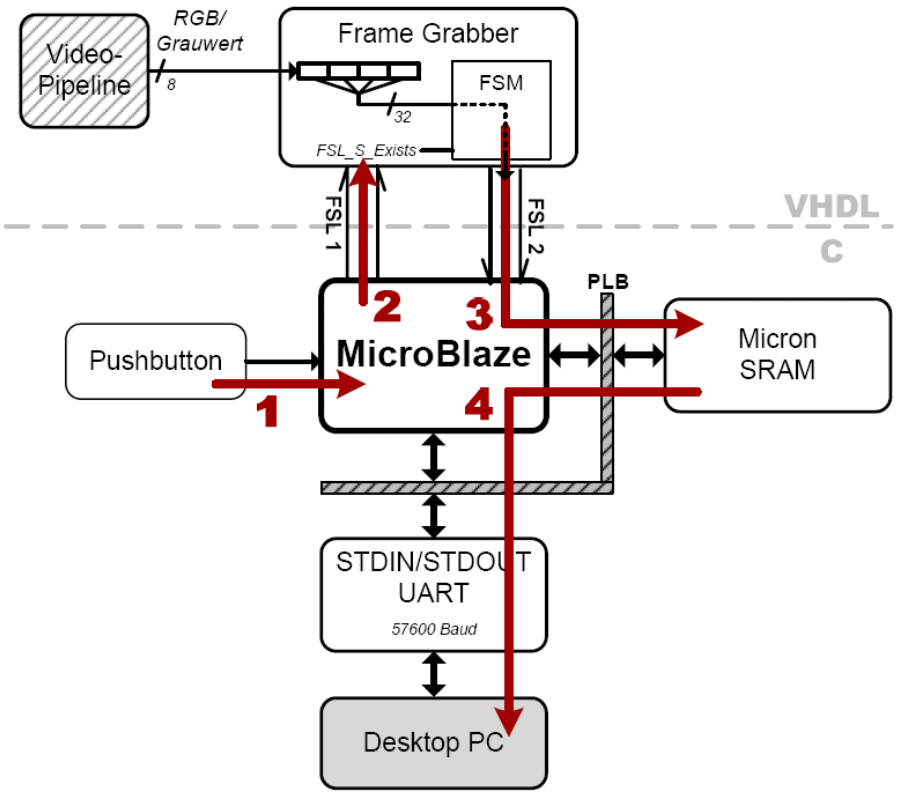


Abb. 22 : Einzelschritte der Abspeicherung eines Bildes aus der Videopipeline zu einem Desktop PC

Der FSL ist eine unidirektionale Verbindung zwischen zwei IPs, daher werden im FrameGrabber zwei FSLs eingesetzt. Durch den ersten FSL wird der FrameGrabber VHDL Komponente signalisiert, dass eine Bildspeicherung erfolgen soll. Der zweite FSL in entgegengesetzter Richtung transportiert die Bilddaten aus der Videopipeline in das Microblaze System.

Die Übertragung einer Bitmap Datei aus den Daten des Videostroms wird durch das Betätigen eines PushButtons auf dem Entwicklungsboard gestartet. Dieser Vorgang wird in der `main()` Funktion der Softwarekomponente durch eine zyklische Statusabfrage des Pushbutton Registers erkannt, woraufhin ein Datum durch die `putfsl()` Funktion in den Buffer des FSL 1 geschrieben wird. Die VHDL Komponente erkennt den Start der Übertragung durch eine zyklische Abfrage des *FSL_S_Exists* Flags des FSL 1 und synchronisiert sich, nachdem dieses gesetzt ist auf den Bilddatenstrom (vgl. *Abb. 23*). Dazu wird der Beginn eines neuen Bildes durch die Auswertung der FVAL und LVAL Signale erkannt.

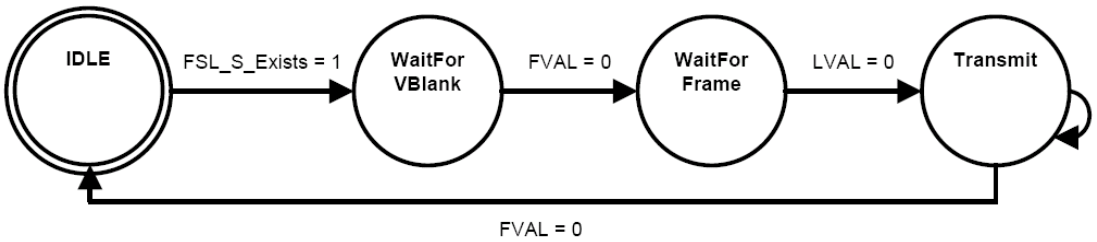


Abb. 23 : Zustandsdiagramm des FrameGrabber Moduls

Sobald der Transmit Zustand erreicht wurde, kann die VHDL Komponente mit der Übertragung der Bilddaten beginnen. Für die Abspeicherung der Bilddaten wird analog zu der Ausgabe der Daten durch den VGA Controller ein auf 8 Bit reduziertes Format der Farbinformationen verwendet. Der FSL unterstützt eine Datenbreite von 32 Bit; demnach können vier Pixelinformationen durch ein Schieberegister zusammengefasst, und in den Buffer des FSL 2 geschrieben werden. Dadurch verringert sich die Frequenz der bereitgestellten Bilddaten, die mit der Eingangsfrequenz von 13,5 MHz anliegen, um das Vierfache. Das Ende der Daten eines gesamten Bildes erkennt die VHDL Komponente des FrameGrabbers anhand des FVAL Signals und beendet die Übertragung.

Für das Auslesen der Daten aus dem Buffer des FSL 2 wird die Eigenschaft genutzt, dass dieser mit zwei verschiedenen Taktraten betrieben werden kann. Während das Schreiben in den FSL Buffer wie beschrieben mit einer Taktrate von 13,5 MHz / 4 erfolgt, kann für das Auslesen die Frequenz des Microprozessor Systems von 50 MHZ verwendet werden. Dadurch kann sichergestellt werden, dass der Microblaze ein genügend großes Zeitfenster zum Ablegen der geschriebenen 32 Bit Werte über den PLB Bus ins SRAM zur Verfügung hat [vgl. Peters, 2009].

Die Bilddaten im FSL Buffer werden innerhalb der Software durch die Funktion `getfsl()` aus diesem gelesen und direkt an eine dafür bereitgestellte Speicheradresse im Micron SRAM geschrieben. Das SRAM ist über den PLB mit dem Microblaze verbunden und ist memory mapped adressierbar. Nachdem das

gesamte Bild im SRAM abgelegt ist, kann die Übertragung der Bilddaten zu einem Desktop PC beginnen. Dazu wird die bereits beschriebene RS232 STDIN/STDOUT Schnittstelle verwendet. Zunächst wird vor den Bilddaten ein standardisierter Bitmap Header übertragen (vgl. *Tab. 8*).

	Offset	Größe	Name	Beschreibung*	Inhalt
Header	0	2 Byte	bfType	*Magic Word*	0x424D
	2	4 Byte	bfSize	*Größe der BMP Datei in Byte*	0x36100E00
	6	4 Byte	bfReserved	*reserviert*	0x0000
	10	4 Byte	bfOffBits	*Offset der Bilddaten in Byte vom Beginn der Datei an.*	0x36000000
Information	14	4 Byte	biSize	*40 (Größe der BITMAPINFOHEADER-Struktur in*	0x28000000
	18	4 Byte	biWidth	*Breite der Bitmap in Pixel.*	0x80020000
	22	4 Byte	biHeight	*Der Betrag gibt die Höhe der Bitmap in Pixel an.*	0x20FEFFFF
	26	2 Byte	biPlanes	*1*	0x0100
	28	2 Byte	biBitCount	*Gibt die Farbtiefe der Bitmap in bpp an.*	0x1800
	30	4 Byte	biCompression	*0 für keine Kompression*	0x00000000
	34	4 Byte	biSizeImage	*Größe der Bilddaten oder 0 bei biCompression = 0*	0x00000000
	38	4 Byte	biXPelsPerMeter	*Horizontale Auflösung des Zieles in Pixel pro Meter*	0x00000000
	42	4 Byte	biYPelsPerMeter	*Vertikale Auflösung des Zieles in Pixel pro Meter*	0x00000000
	46	4 Byte	biClrUsed	*Anzahl der Einträge der Farbtabelle*	0x00000000
	50	4 Byte	biClrImportant	*Anzahl sämtlicher im Bild verwendeterFarben*	0x00000000
	54	x Byte	…	*Bilddaten*	…
					*siehe [Wikipedia, 2009a]

Tab. 8 : Aufbau des Headers eines Bitmap Files mit BITMAP- Header und BITMAP-Information

Dadurch können die übermittelten Daten gleich auf dem Zielrechner dargestellt werden. Da die Bilddaten am Eingang in den FrameGrabber noch im Interlace Format vorliegen, wird vor der Übertragung die Zeilenverdopplung vorgenommen (vgl. *Code 8*). So sendet die Software jede Zeile des Bildes zweimal über die serielle Schnittstelle. Über ein herkömmliches Terminal Programm auf dem Desktop PC können die empfangenen Daten in einer Bitmap Datei abgespeichert werden.

```
// Buffer a frame
   putfsl(42, 0);
   for(reccnt=0; reccnt < read_pixcnt; reccnt++){
      getfsl(target[reccnt],0);
   }

   // Send header
   for(i=0;i<54;i++) xil_printf("%c", bmp_header[i]);

   // Send body
   for(row=0; row<rowcnt; row++){
      // Each line is sent twice ("deinterlace")
      for(j=0; j<2; j++)
         for(col=0; col<colcnt; col++){
            u8 r, g, b;
            i = row*colcnt + col;
            r =  frame[i] & 0xE0;
            g = (frame[i] & 0x1C) << 3;
            b = (frame[i] & 0x03) << 6;
            xil_printf("%c%c%c",b,g,r); //send via UART
         }
   }
```

Code 8 : Die FrameGrabber Software zum Empfangen der Bilddaten aus dem FSL und Versenden über die UART Schnittstelle

4 Bildverarbeitung zur Kantenerkennung

4.1 Vorbereitung der Bildverarbeitung

Die Bildpunkte des Datenstroms müssen vorbearbeitet werden, um eine Fahrspur aus den vorliegenden Bilddaten extrahieren zu können. Die Fahrspurmarkierungen heben sich durch ihre Helligkeit gegenüber der restlichen Fahrspur deutlich von dieser ab, sodass für die Erkennung der Fahrspuren die Bildpunkte mit den größten Helligkeitsunterschieden in Bezug auf ihre Nachbarn gefunden werden müssen. Die eigentliche Farbe der Markierung ist in diesem Fall irrelevant, daher genügt es die Bildpunktinformationen in Grauwerte umzuwandeln und höhere Helligkeitsunterschiede zwischen zwei Bildpunkten zu verstärken.

Die Helligkeitsinformationen eines RGB Bildes lassen sich auch durch ein Grauwertbild darstellen. Das hat den Vorteil, dass die Datenmenge erheblich von 24 Bit auf 8 Bit pro Bildpunkt reduziert wird. Die Gleichung zur Umrechnung eines RGB Farbwertes in den entsprechenden Grauwert lautet [Wikipedia, 2009b]:

$$Grauwert = 0{,}299 * R + 0{,}587 * G + 0{,}114 * B \tag{4.1}$$

Wie man der Gleichung entnehmen kann, hat der Grünanteil eines RGB Bildes eine höhere Auswirkung auf die Helligkeitsinformationen. Für die Umrechnung in Grauwertbilder wurde im entwickelten System kein eigenes Modul entworfen, sondern lediglich der Grünwert für die weiteren Berechnungen verwendet. Der Entwurf und die Integration eines entsprechenden Umrechnungsmoduls, sollte in einer weiterführenden Arbeit vorgenommen werden.

Durch den Einsatz von Filtermasken können verschiedene Operationen zum Beispiel zur Bildschärfung, Bildglättung und Kantenfilterung auf ein Bild angewandt werden. Dabei werden die Werte der benachbarten Bildpunkte in die Ermittlung des neuen Bildpunktwertes einbezogen, um diesen, entsprechend der Helligkeitsunterschiede zu seinen Nachbarn, zu verstärken bzw. zu mindern [vgl. Meisel, 2008]. Die *Abb. 24* zeigt die Anwendung einer Sobel-Filtermaske auf ein Originalbild zur Berechnung des entsprechenden Zielwertes.

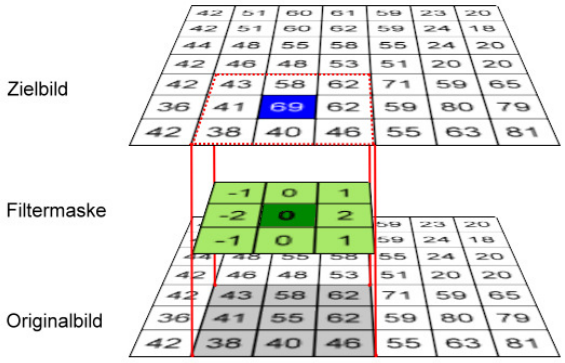


Abb. 24 : Berechnung des Zielbildes durch Verwendung einer Filtermaske

In der vorliegenden Arbeit wird der Sobel Operator zur Kantenextraktion verwendet. Im Gegensatz zum Laplace Filter [vgl. Abmayr, 1994] bietet dieser den Vorteil, dass Rauscheffekte unterdrückt werden, da der eigentliche Ausgangswert des Bildpunktwertes nur nebenläufig in die Berechnung des neuen Wertes eingeht. Die Gewichtung für diesen neuen Wert liegt beim Sobel Filter auf dem Verhältnis der umliegenden Bildpunktwerte [vgl. Nischwitz, Haberäcker, 2004].

Horizontal **Vertikal**

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.2}$$

Beim Sobel Operator wird der zu ermittelnde Wert des Bildpunktes aus den Helligkeitsunterschieden der umliegenden Bildpunkte in horizontaler und vertikaler Richtung ermittelt. Die Teilergebnisse der Richtungsoperationen müssen in einem weiteren Schritt über die Bildung des Betrages der Gradienten zusammengefasst werden.

Betrag des Gradienten

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{4.3}$$

Das resultierende Gesamtbild nach der Anwendung einer Sobel-Filterung zeigt sich in *Abb. 25*.

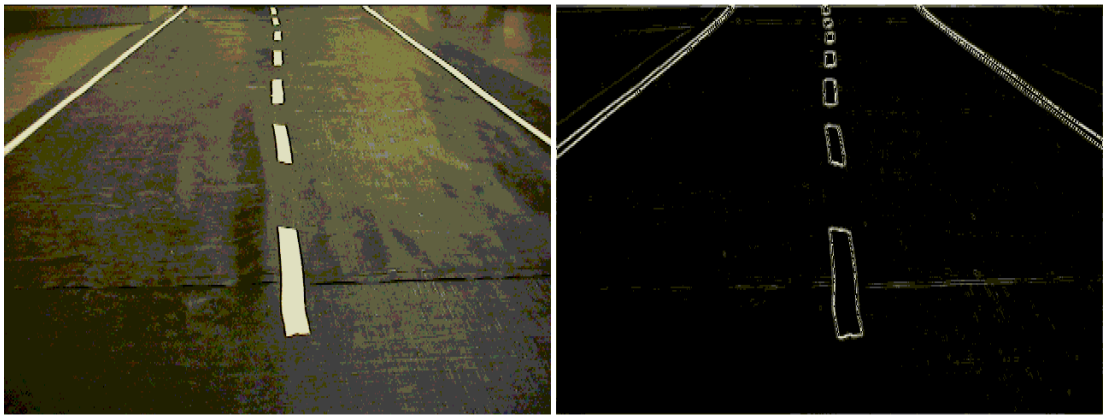


Abb. 25 : Original- und Ergebnisbild nach Anwendung des Sobeloperators

4.1.1 ImageProcessing Modul

Die Anwendung der Sobel-Filterung auf die Bilddaten erfordert die Anwendung von zwei Filtermasken (vgl. *Abb. 26*). Dazu müssen die Daten zunächst deserialisiert werden, um alle relevanten Bildpunkte in die Berechnung aufnehmen zu können. Die Anwendung der Filtermasken kann dann mit den bereitgestellten Bildpunktinformationen parallel erfolgen. Dazu werden innerhalb des ImageProcessing Moduls die Ausgangsdaten des Deserialisierers mit zwei Filter Modulen verknüpft.

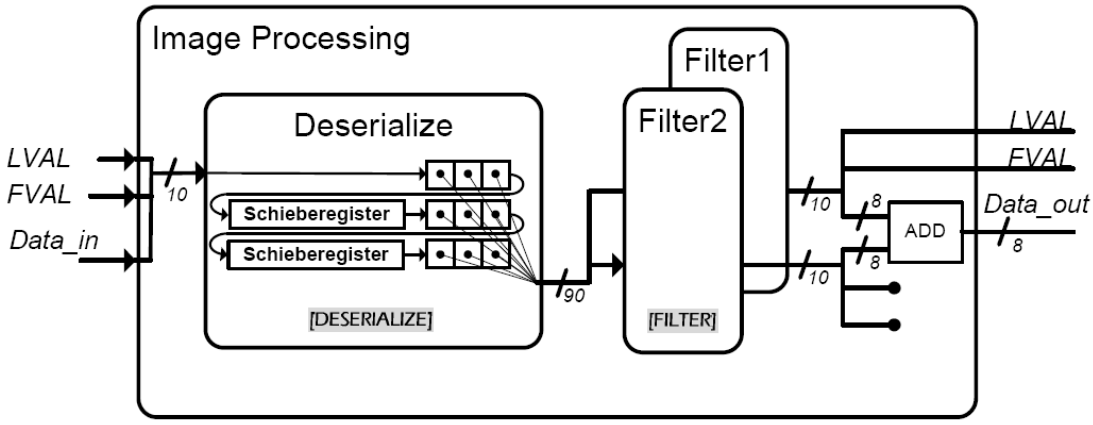


Abb. 26 : Blockschaltbild der ImageProcessing Komponenten

Der Ausgang des ImageProcessing Moduls ist die Summe der Filterausgänge. Innerhalb des Filter Moduls wird gewährleistet, dass der Ausgangsvektor einen gültigen, positiven Wert im Bereich zwischen 0 und 255 entspricht. Dadurch kann die Ermittlung des Betrages der horizontalen und vertikalen Filterergebnisse durch eine einfache Addition gelöst werden. Die Synchronisationssignale werden an die Bildpunktinformationen gebunden und durch die gesamte Prozesskette mitgeführt, was den Effekt hat, dass sie entsprechend der Bildpunktwerte gleichermaßen verzögert werden (vgl. *Code 9*).

```
process(filter_DATA_OUT1,filter_DATA_OUT2)
   variable outsum_v       : std_logic_vector (8 downto 0) := (others =>'0');
   variable data1, data2   : std_logic_vector (7 downto 0);
   begin
      data1 := filter_DATA_OUT1(9 downto 2);
      data2 := filter_DATA_OUT2(9 downto 2);
      outsum_v := data1 + data2;

      if (outsum_v > 255) then
         OUTSUM <= (others => '1');
      else
         OUTSUM <= std_logic_vector(outsum_v(7 downto 0));
      end if;
   end process;

   deser_DATA_IN <= (DATA_IN(7 downto 0) & FVAL_IN & LVAL_IN);
   DATA_OUT <= OUTSUM;
   FVAL_OUT <= filter_DATA_OUT1(1);
   LVAL_OUT <= filter_DATA_OUT1(0);
```

Code 9 : Erzeugung der Ausgänge im IamgeProcessing Modul

4.1.2 Deserialisierung des Bilddatenstroms

Die Anwendung einer Filtermaske auf die Daten des bestehenden Systems erfordert eine Deserialisierung, damit die Werte der umliegenden Bildpunkte bei der Berechnung zur Verfügung stehen. Für das entwickelte System wurde zunächst festgelegt, dass Faltungsmasken in Form von 3x3 Matrizen eingesetzt werden (vgl. *Abb. 27*). Daher müssen für die Berechnung des Bildwertes zwei gesamte Bildzeilen zwischengespeichert werden, bevor die Berechnung des aktuellen Bildpunktes erfolgen kann. Das Deserialisierungs Modul umfasst demnach zwei Schieberegister, die 10 Bit breite Vektoren aufnehmen können und eine Tiefe von 855 Einträgen haben. Dies entspricht der genauen Anzahl der Bildpunkte einer gesamten Bildzeile mit Blankinganteilen. Es werden 10 Bit Vektoren verwendet, da zu jedem 8 Bit breiten Bildpunktwert auch die beiden Synchronisationssignale FVAL und LVAL mitgespeichert und durch die Schieberegister geführt werden. Das Deserialisierungsmodul besitzt intern keinerlei Synchronisationslogik und liefert als Ausgang lediglich die gesamten Werte die für eine Operation mit einer 3x3 Faltungsmatrix benötigt werden; demnach einen Ausgangsvektor der 9 * 10 Bit breit ist.

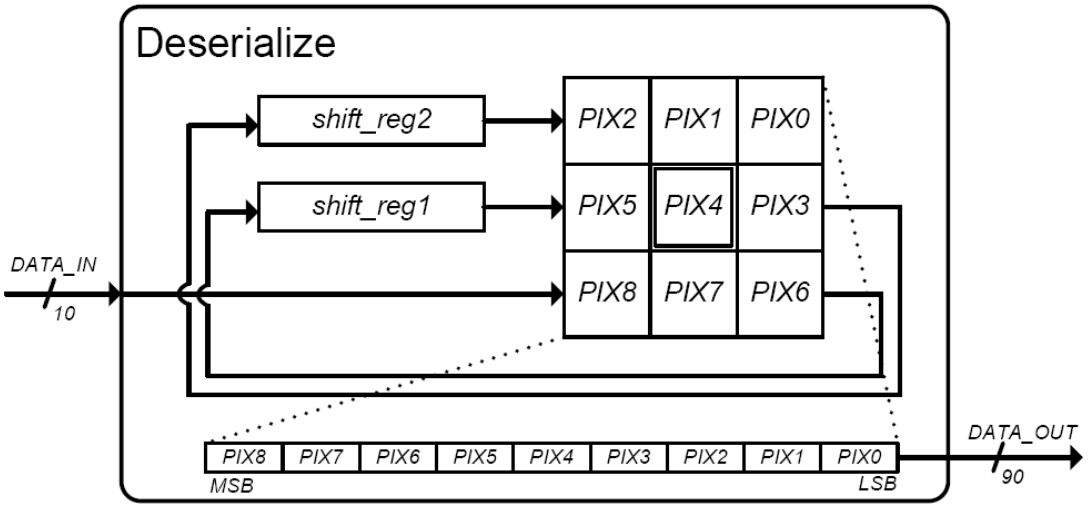


Abb. 27 : Bereitstellung der Farbinformationen des Datenstroms für eine 3x3 Filtermaske

4.1.3 Bildaufbereitung durch Einsatz des Sobel Filters

Das Filtermodul ist durch den Einsatz von *GENERICS* parametrierbar und kann somit für verschiedene Filteroperationen eingesetzt werden. Es nimmt als Eingangssignale einen 9 * 10 Bit breiten Vektor auf und ist somit auf den Einsatz von 3x3 Faltungsmatrizen begrenzt. Innerhalb des Moduls werden die Bildpunktwerte der neun Eingangswerte von den Synchronisationssignalen getrennt und diese mit den jeweiligen angegebenen Filterwerten multipliziert. Dazu werden 12 Bit breite Hilfsvariablen eingesetzt, damit Overflows vermieden werden können (vgl. *Code 10*). Der 10 Bit breite Ausgangsvektor wird dann aus der Summe der neun Zwischenergebnisse und den Synchronisationssignalen des zentralen Bildpunktes gebildet. Zusätzlich wird das Ergebnis der Addition auf den Wertebereich von 0 bis 255 normiert.

```
generic(
   -- DO NOT CHANGE VEC_WIDTH WITHOUT UPDATING THE SHIFT_RAMS!!!
   VEC_WIDTH : integer := 10; -- 1 sign bit + 8 data bit + fval + lval
                              -- => 11 => 10 downto 0 !
   MAT_SIZE  : integer := 3;  -- Matrix size (3x3)
   --FILTER MASK for Sobel Gx filter
   FIL_MASK_0 : integer := -1;
   FIL_MASK_1 : integer := 0;
   FIL_MASK_2 : integer := 1;
   FIL_MASK_3 : integer := -2;
   FIL_MASK_4 : integer := 0;
   FIL_MASK_5 : integer := 2;
   FIL_MASK_6 : integer := -1;
   FIL_MASK_7 : integer := 0;
   FIL_MASK_8 : integer := 1
   );
[...]
begin
   --using signed variables with 4 additional bits to avoid overflows
   sr_8 := signed("0000" & DATA_IN(89 downto 82));
   sr_7 := signed("0000" & DATA_IN(79 downto 72));
   sr_6 := signed("0000" & DATA_IN(69 downto 62));
   sr_5 := signed("0000" & DATA_IN(59 downto 52));
   sr_4 := signed("0000" & DATA_IN(49 downto 42));
   sr_3 := signed("0000" & DATA_IN(39 downto 32));
   sr_2 := signed("0000" & DATA_IN(29 downto 22));
   sr_1 := signed("0000" & DATA_IN(19 downto 12));
   sr_0 := signed("0000" & DATA_IN(9  downto 2));

   sr_8 := sr_8 * FIL_MASK_8;
   sr_7 := sr_7 * FIL_MASK_7;
   sr_6 := sr_6 * FIL_MASK_6;
   sr_5 := sr_5 * FIL_MASK_5;
   sr_4 := sr_4 * FIL_MASK_4;
   sr_3 := sr_3 * FIL_MASK_3;
   sr_2 := sr_2 * FIL_MASK_2;
   sr_1 := sr_1 * FIL_MASK_1;
   sr_0 := sr_0 * FIL_MASK_0;

   resu := (sr_8 + sr_7 + sr_6 + sr_5 + sr_4 + sr_3 + sr_2 + sr_1 + sr_0);

   if resu < 0 then
      resu := -resu;
   end if;

      RESULT <= std_logic_vector(resu(9 downto 2)) & DATA_IN(41 downto 40);
end process;
```

Code 10 : Anwendung der Filterfunktion über eine 3x3 Maske für den Gx Wert des Sobel Operators

5 Fahrspuridentifikation

In diesem Kapitel wird ein Konzept zur Realisierung der Fahrspurerkennung auf dem entwickelten System vorgestellt. Nachdem die Kanten in den Bilddaten durch die Anwendung des Sobel Operators hervorgehoben wurden, können die Bildinformationen zur Identifikation der Fahrbahnmarkierungen genutzt werden. Allerdings sind noch weitere Schritte zur Aufbereitung des Datenstroms für die weitere Interpretation von Vorteil. So sollten die Daten vor der weiteren Verarbeitung binarisiert werden. Dadurch würde sich abermals das Datenvolumen verringern, da nur noch 2 Bit pro Bildpunktwert verwendet werden müssen; zusätzlich würden somit weitere nicht eindeutig identifizierte Bildpunkte herausgefiltert.

Eine weitere Optimierung der Bilddaten besteht darin, die erkannte Fahrspurmarkierung als einen einzelnen Vektor zusammen zufassen. Durch den Sobel Operator werden beide Helligkeitsunterschiede einer Fahrspurmarkierung in Bezug auf die Straße erkannt, somit liegen zwei nahezu vertikal parallel verlaufende Linien im Gesamtbild vor. Diese würden die weitere Verarbeitung unnötig verzögern und sollten demnach zusammengefasst werden. Des Weiteren sollten Lücken innerhalb der erkannten Markierungen, die zum Beispiel durch Rauschen oder Umwelteinflüsse im Originalbild entstanden sind, geschlossen werden. Das kann durch Opening- und Closingoperationen erfolgen, womit gleichzeitig auch die Lücken in der mittleren Fahrspurmarkierung geschlossen werden. Das in dieser Weise vorbearbeite Gesamtbild enthält dann anhand der Grauwertinformationen lokale Maxima, die als Messpunkte in die weitere Verarbeitung zur Bestimmung der Spurführung, eingehen [vgl. Jenning, 2008].

Das im Rahmen dieser Arbeit erstellte Konzept sieht vor, dass ein, zur Darstellung der Fahrspur definiertes, Polynom durch den Einsatz eines Kalman-Filters erzeugt wird [vgl. Peters, 2008]. Dazu wird im ersten Teil dieses Kapitels eine Einführung in den grundsätzlichen Aufbau des Kalman-Filters gegeben. Die Implementierung von komplexen mathematischen Algorithmen auf der erzeugten FPGA Plattform bedeutet einen erhöhten Entwicklungsaufwand. Aus diesem Grunde wird im zweiten Teil der Einsatz eines Codegenerators vorgestellt, der aus der abstrakten mathematischen Beschreibung eines Algorithmus‘ in MATLAB-Code, synthesefähigen VHDL-Code erzeugt. Somit kann der Aufwand für die Realisierung der im Kalman-Filter verwendeten Matrizenoperationen verringert werden.

5.1 Einführung in das Kalman-Filter

Das Kalman-Filter ist seit der Originalarbeit von Kalman aus dem Jahr 1960 eine weitverbreitete Methode zur Integration unterschiedlicher Messdaten aus verschiedenen Sensoren in Gesamtsysteme und wird speziell in der Navigationstechnik eingesetzt. In der Bildverarbeitung wird das Kalman-Filter häufig für Aufgaben zur Objektverfolgung, dem *Tracking*, verwendet.

Das Filter beschreibt eine Vorgehensweise zur Abschätzung des Folgezustands in einem dynamischen System unter Berücksichtigung von Messfehlern. Da die Messungen stets fehlerbehaftet sein können, wird davon ausgegangen, dass der aktuelle Systemzustand zu keinem Zeitpunkt eindeutig bestimmt werden kann. Für die Ermittlung des Folgezustandes werden daher zwei Zustände generiert; aus einer, das System beschreibenden, Gleichung wird ein Systemzustand prädiziert, ein Zweiter wird aus aktuellen Messdaten ermittelt. Den beiden Zuständen wird durch zusätzliche Fehlerkoeffizienten eine Gewichtung gegeben; diese Fehlerkoeffizienten symbolisieren das System- bzw. das Messrauschen. Durch die Abweichungen zwischen den beiden ermittelten Zuständen, kann der aktuelle Systemzustand präziser abgeschätzt werden, und Fehler in den Messdaten wirken sich nicht signifikant auf den Folgezustand aus [Nischwitz, Haberäcker, 2004].

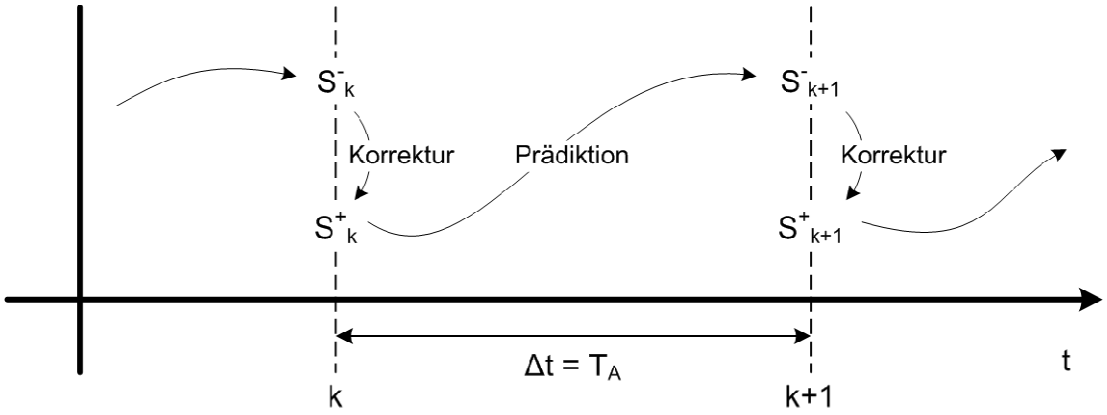


Abb. 28 : Prädiktions- und Korrekturschritte des Kalman-Filters

Die Abschätzung des Systemzustandes erfolgt im laufenden System abwechselnd durch zwei separate Schritte: *Prädiktion des Folgezustandes* und *Korrektur mit den Messdaten* (vgl. *Abb. 28*) Wobei dieser Ablauf den Vorteil hat, dass nur ein Zustand im System gespeichert werden muss. Dadurch eignet sich das Kalman-Filter, gegenüber Filtern die den Zustand aus mehreren Messdaten ermitteln, besonders gut für den Einsatz in Echtzeitsystemen. Für die Abschätzung des Folgezustandes x_{k+1} eines dynamischen Systems, kann dieses durch eine zeitdiskrete Differenzialgleichung beschrieben werden:

$$x_{k+1} = A_k * x_k + B_k * u_k + w_k \tag{5.1}$$

Dabei stellt x_k einen n-dimensionalen Systemzustandsvektor und u_k einen l-dimensionalen Steuervektor zum Zeitpunkt k dar. Die Systemmatrix A_k bildet den

Systemzustand vom Zeitpunkt k auf den Zustand des aktuellen Zeitschritts $k+1$ ab. Über die Eingabematrix B_k wird der Steuervektor u_k auf den aktuellen Systemzustand abgebildet. Durch die Zufallsvariable w_k wird das Systemrauschen in die Gleichung aufgenommen. In die Messgleichung werden lediglich die Messungen, die im direkten Zusammenhang mit dem Zustand des Systems x_k stehen aufgenommen:

$$y_k = H_k * x_k + v_k \tag{5.2}$$

Die Messmatrix H_k bildet den Systemzustand x_k zum Zeitpunkt k auf den Messwert y_k ab. Die Zufallsvariable v_k beschreibt den zweiten Fehlerkoeffizienten, das Messrauschen. Für die Abschätzung des Systemzustandes wird nun in der Prädiktion ein Systemzustand $\hat{x}_k^-$ „a priori“ ermittelt. Der Zustand in den die aktuellen Messungen zum Zeitpunkt k eingegangen sind, bezeichnet man als $\hat{x}_k^+$ („a posteriori“); dieser Zustand enthält demnach den mit der neuen Messung korrigierten Schätzwert. Um den „a posteriori“ Schätzwert $\hat{x}_k^+$ ermitteln zu können, muss sowohl der „a priori“ Schätzwert $\hat{x}_k^-$ als auch die Differenz zwischen aktuellem Messwert y_k und dem vorhergesagten Messwert $H_k * \hat{x}_k^-$ in die Korrekturgleichung des Kalman-Filters eingehen:

$$\hat{x}_k^+ = \hat{x}_k^- + K_k(y_k - H_k * \hat{x}_k^-) \tag{5.3}$$

Durch die Differenz $(y_k - H_k * \hat{x}_k^-)$ kann die Qualität des vorhergesagten Systemzustandes bewertet werden. Sollte der vorhergesagte Messwert $H_k * \hat{x}_k^-$ exakt dem tatsächlich Gemessenen y_k entsprechen, so wird diese Null und der „a posteriori“- entspricht dem „a priori“ Schätzwert des Systemzustandes: $\hat{x}_k^+ = \hat{x}_k^-$ K_k beschreibt die Kalman-Verstärkungsmatrix und wird so gewählt, dass die a posteriori Schätzfehlerkovarianz P_k^+ minimal wird. Über K_k wird die Varianz des Messfehlers R_k in Zusammenhang mit der „a priori“ Schätzfehlerkovarianz P_k^+ gesetzt und in die Gewichtung des geschätzten Wertes $H_k * \hat{x}_k^-$ bzw. des gemessenen Wertes y_k innerhalb der Korrekturgleichung angewandt.

$$K_k = \frac{P_k^- * H_k^T}{H_k * P_k^- * H_k^T + R_k} \tag{5.4}$$

Zur besseren Verdeutlichung sollten die Grenzfälle dieser Gleichung betrachtet werden: Bei einer gegen null strebenden Varianz des Messfehlers R_k, wird K_k maximal und die Daten der Messung werden entsprechend in das korrigierte Ergebnis der Schätzung $\hat{x}_k^+$ übernommen. Sollte die im Voraus bestimmte Schätzungsfehlerkovarianz P_k^- gegen Null streben, wird das Ergebnis für K_k ebenfalls Null und die prädizierte Messung $H_k * \hat{x}_k^-$ geht mit einem höheren Gewicht in die Korrekturgleichung ein, wobei die tatsächliche Messung vernachlässigt wird [Nischwitz, Haberäcker, 2004].

Für die Konzepte zur Implementierung des Kalman-Filters auf der FPGA Plattform wird im Folgenden ein Filter in einer vereinfachten Umsetzung dargestellt. Diese bezieht sich auf die Schätzung eines konstanten Wertes auf der Basis verrauschter Messdaten. Die Gleichungen 5.1 – 5.4 können demnach für das später gezeigte Beispiel dieser Umsetzung vereinfacht werden. Da der Systemzustand x in zeitlicher Konstanz durch einen einzigen Wert beschrieben wird, kann die Systemmatrix als $A_k = 1$ angenommen werden. Ein Steuervektor ist in dieser Umsetzung nicht vorgesehen, wodurch sich mit $u_k = 0$ die Systemzustandsgleichung zu

$$x_{k+1} = x_k + w_k \tag{5.5}$$

vereinfachen lässt. Auch die Messgleichung lässt sich vereinfachen, da die Messmatrix ebenfalls aus einem einzigen Wert besteht und dieser durch die direkte Messbarkeit des Systemzustandes auch mit $H_k = 1$ angegeben werden kann.

$$y_k = x_k + v_k \tag{5.6}$$

Daraus ergeben sich die in *Abb. 29* dargestellten Gleichungen für die Prädiktions- und Korrekturschritte. Diese vereinfachten Gleichungen lassen sich auf das im folgenden Abschnitt vorgestellte Beispiel zur Implementierung des Kalman-Filters auf einer FPGA Plattform anwenden.

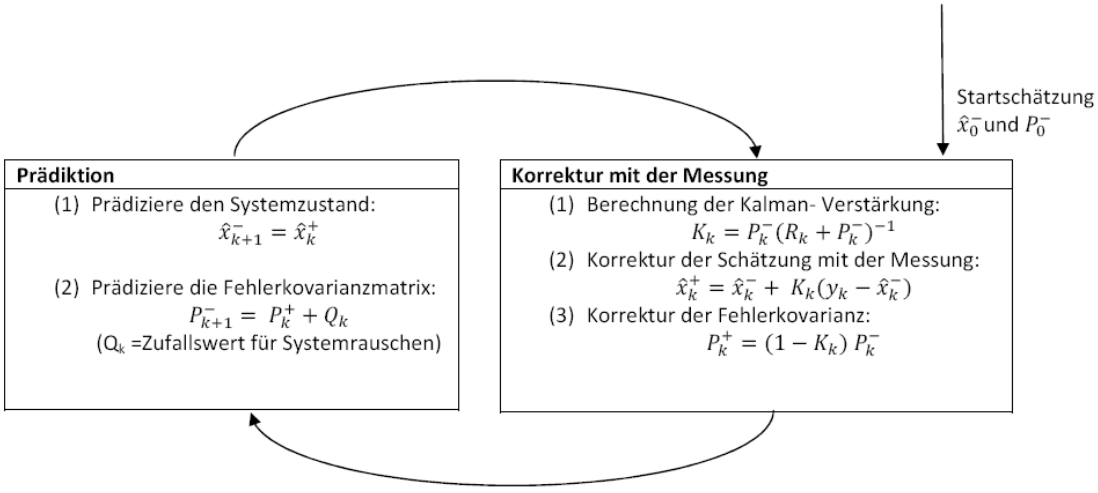


Abb. 29 : Angepasste Gleichungen des vereinfachten Kalman-Filters

5.2 Implementierung des Kalman-Filters auf einer FPGA Plattform

Der Entwicklungsaufwand für die Implementierung komplexer mathematischer Operationen auf FPGA basierten Systemen steigt durch die Verwendung von Matrixoperationen um einen erheblichen Anteil. Die Identifikation und Realisierung von parallel zu verarbeitenden Teilschritten ist in diesen Systemen eine zeitaufwendige und fehlerträchtige Aufgabe, die unter Berücksichtigung der zur Verfügung stehenden Ressourcen und Laufzeitvorgaben mit einem hohen Maß an Sorgfalt erledigt werden muss. Die Komplexität dieser Umsetzung besteht unter anderem darin, dass der Entwickler sowohl die hardwarespezifischen Details der Entwicklungsplattformen als auch die relevanten Techniken zur Erstellung des Hardwarebeschreibungscodes in einen Zusammenhang mit den komplexen mathematischen Vorgaben bringen muss.

Die Hochsprache für technische Berechnungen MATLAB [TheMathWorks, 2009] wird in diesem Zusammenhang häufig zur Entwicklung und Verifikation der Algorithmen in der Entwurfsphase von FPGA basierten Systemen eingesetzt [Vanevenhoven, 2007]. Weiterhin wird eine interaktive Umgebung für die Entwicklung der Algorithmen sowie die Visualisierung und die Analyse der Systemdaten durch MATLAB bereitgestellt, was den Einsatz in einer Vielzahl von Anwendungsgebieten, wie zum Beispiel der Signal- und Bildverarbeitung oder der Entwicklung von Steuerungssystemen, unterstützt. Aufgrund der umfangreichen integrierten Funktionsbibliothek und der Eigenschaft, dass sich der MATLAB-Code in andere Programmiersprachen und Anwendungen integrieren lässt, ist die Software ein weit verbreitetes Entwicklungswerkzeug.

Die Vorteile des relativ einfach zu erzeugenden und zu testenden MATLAB-Codes können durch das Xilinx AccelDSP Synthese Werkzeug für die Entwicklung von komplexen FPGA Systemen verwendet werden. Das AccelDSP Werkzeug transformiert in einer Reihe von Teilschritten den vorgegeben MATLAB-Code in RTL -Code. Aus diesem können dann unter Verwendung des Xilinx System Generators IP Blöcke zur Integration in Gesamtsysteme generiert werden.

Die Ergebnisse der einzelnen Teilschritte vom Floating Point Modell des MATLAB-Codes bis hin zum fertigen Hardware Modul, können nach deren Erzeugung verifiziert werden, um die Einhaltung der Systemvorgaben oder die Auslastung der Systemressourcen während der Erstellung beeinflussen zu können (vgl. *Abb. 30*). Die allgemeine Vorgehensweise zur Erstellung eines Hardware Moduls mit Hilfe von AccelDSP erfolgt in den Teilschritten [AccelDSP, 2008]:

- Definition der Systemparameter, wie FPGA Typ und Modell sowie der verwendeten Taktrate. Außerdem kann definiert werden, ob das zu erzeugende Fixed Point Modell in C oder MATLAB generiert werden und ob als Ausgabeformat der Hardwarebeschreibung VHDL oder Verilog verwendet werden soll
- Überprüfung des MATLAB-Quellcodes, um die Einhaltung von bestimmten Programmierrichtlinien in Bezug auf die Verwendung von AccelDSP sicher zu stellen; dieser Schritt muss durch den Anwender erfolgen.

- Verifikation des Floating Point Modells anhand des MATLAB-Codes. Dazu werden Eingangsstimuli aus einem Skript File verwendet um die Ausgangsgrößen durch MATLAB zu berechnen und in visueller Form darzustellen.
- Analyse des Floating Point Modells zur Bestimmung der bestmöglichen Hardware Architektur.
- Erzeugung eines Fixed Point Modells aus dem bestehenden Floating Point Modell
- Verifikation des Fixed Point Modells durch einen visuellen Vergleich der Eigenschaften mit denen des Floating Point Modells, wobei das gleiche Skript File verwendet wird, mit dem auch schon das Floating Point Modell getestet wurde.
- Erzeugung des RTL Modells anhand des Fixed Point Modells
- Verifikation des RTL Modell durch den Einsatz eines HDL Simulators wie ModelSim
- Synthese des RTL Modells in eine Gate-Level Netzliste
- Implementierung der Netzliste; verwendet die ISE Implementation Tools zum „Place and Route“ des Designs
- Verifikation des Gate-Level Designs anhand von „bit-level“- Vergleichen mit dem Fixed Point Modell

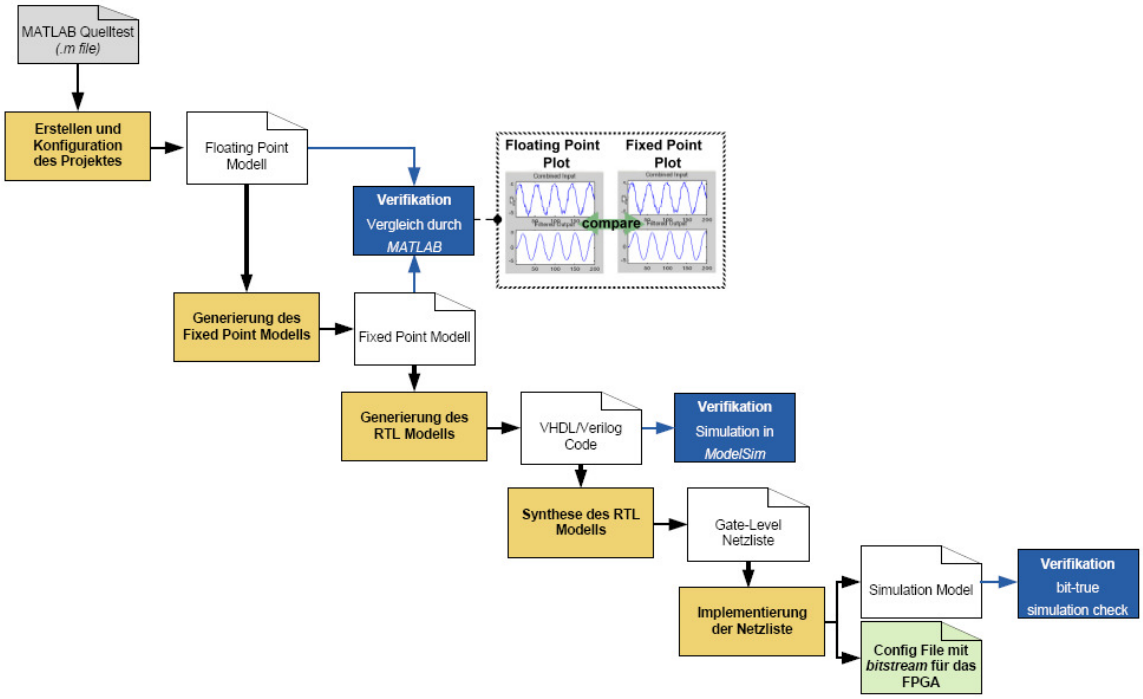


Abb. 30 : Entwicklungsschritte einer Implementierung mit Hilfe von AccelDSP

Im folgenden Beispiel wird das im vorherigen Abschnitt erläuterte vereinfachte Kalman-Filter verwendet, um die Vorgehensweise einer Implementierung mit AccelDSP zu beschreiben. Zunächst werden dafür die Prädiktions- und Korrekturgleichungen in MATBLAB-Code übertragen (vgl. *Tab 9*):

Vereinfachtes Kalman-Filter	**MATLAB**
Korrektur	
$K_k = P_k^-(R_k + P_k^-)^{-1}$	K = P_cap_est * inv(P_cap_est +R)
$\hat{x}_k^+ = \hat{x}_k^- + K_k(y_k - \hat{x}_k^-)$	p = p + K*(A^T-p)
$P_k^+ = (1 - K_k) * P_k^-$	P_cap = (I-K) * P_cap_est
Prädiktion	
$P_{k+1}^- = P_k^+ + Q_k$	P_cap_est = P_cap + I
$\hat{x}_{k+1}^- = \hat{x}_k^+$	

Tab. 9 : Umsetzung der vereinfachten Kalman-Filter Gelichungen in den entsprechenden MATLAB-Code

Das MATLAB-Code File zeigt die Umsetzung der Übertragung der Kalman-Filter Gleichungen (vgl. *Code 11*). Dabei ist das Kalman-Filter als eine Funktion definiert, welche die Messdaten über den Parameter A aufnimmt.

```
function [S] = kalman_filter(A)

DIM = size(A,2);

persistent p P_cap

if isempty(P_cap)
    P_cap = [8 0 0;0 8 0; 0 0 8];
    p = ones(DIM,1)/2;
end;

I = eye(DIM);
R = [128 0 0;0 128 0; 0 0 128];

P_cap_est = P_cap+I;

% correction step:
K=P_cap_est * accel_qr_inverse(P_cap_est+R);
p = p + K * (A' - p);
P_cap = (I - K)*P_cap_est;
S = p';
accel_probe(S);
```

Code 11 : Der MATLAB Quellcode zur Realisierung des vereinfachten Kalman-Filters

Dieser Parameter stellt ein Vektorarray dar, der innerhalb des Skript Files erzeugt und mit Werten von Sinus und Cosinus Funktionen gefüllt wird (vgl. *Code 12*). Anschließend werden die Vektorwerte durch eine Zufallsfunktion „verrauscht“.

```
%Kalman filter example

t=0:0.001:2;
A(:,1) = cos(0:pi/400:(5/3)*pi)/2;
A(:,2) = sin(0:pi/200:(10/3)*pi)/2;
A(:,3) = sin(0:pi/100:(20/3)*pi)/2;
A_in = A + 0.5*(rand(size(A))-.5);
DIM = size(A,2);
LEN = size(A,1);

% assume measurement matrix H is identical:
%H = eye(DIM);

for i=1:LEN,
  [S(i,:)] = kalman_filter(A_in(i,:));
end

subplot(3,1,1);
plot(A(1:length(A),:));
title('Input Signal without Noise')
subplot(3,1,2);
plot(A_in(1:length(A),:));
title('Input Signal with Random Noise')
subplot(3,1,3);
title('Filtered Output Signal')
plot(S);
accel_probe_plot;
```

Code 12 : Das Skript File zur Erzeugung der Testdaten und Verifikation des vereinfachten Kalman-Filters

Innerhalb einer Schleife wird die Kalman-Filter Funktion für alle Werte des Vektorarrays von A aufgerufen und die Ergebnisse in das Vektorarray S übernommen. Am Ende des Skript Files werden die ursprünglichen-, die verrauschten und die schließlich gefilterten Werte über die `plot()` bzw. `subplot()` Funktion ausgegeben.

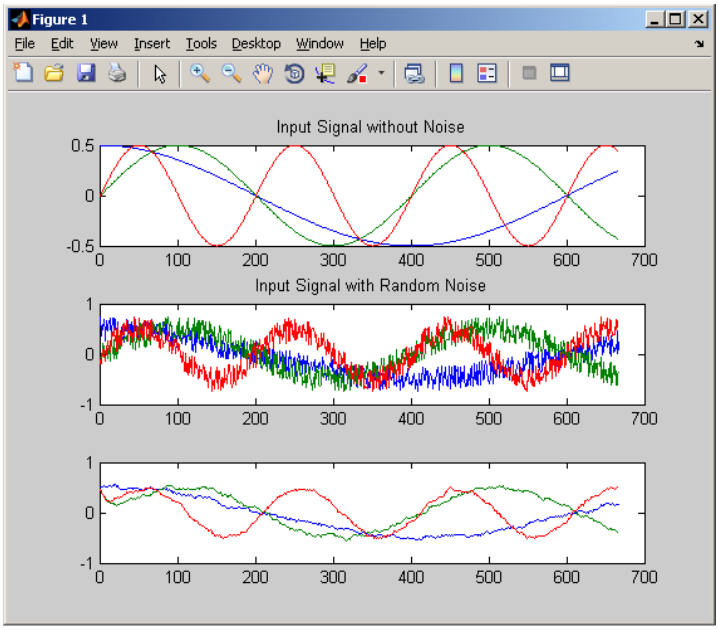


Abb. 31 : Ausgabe der erzeugten und gefilterten Werte durch die Funktion plot()

Durch das Aufrufen der `accel_probe()` Funktion innerhalb des MATLAB-Code Files werden die gefilterten Werte für die Ausgabe in einer weiteren Darstellung registriert. Diese wird durch den Aufruf der `accel_probe_plot()` Funktion am Ende des Skript Files erzeugt und unterstützt den Entwickler beim Vergleich der Ausgangswerte vom Fixed Point Modell gegenüber dem Floating Point Modell (vgl. *Abb. 32*). Diese Übersicht wird von AccelDSP bereitgestellt, da der manuelle Vergleich bei einer großen Anzahl von Eingangs- bzw. Ausgangssignalen einen entsprechenden Zeitaufwand und Fehleranfälligkeit beinhaltet. Bei dieser Ausgabe werden demnach alle erzeugten Filterausgänge des Fixed Point Modells den Ausgängen des Floating Point Modells gegenübergestellt. Außerdem wird die größte auftretende Abweichung angezeigt.

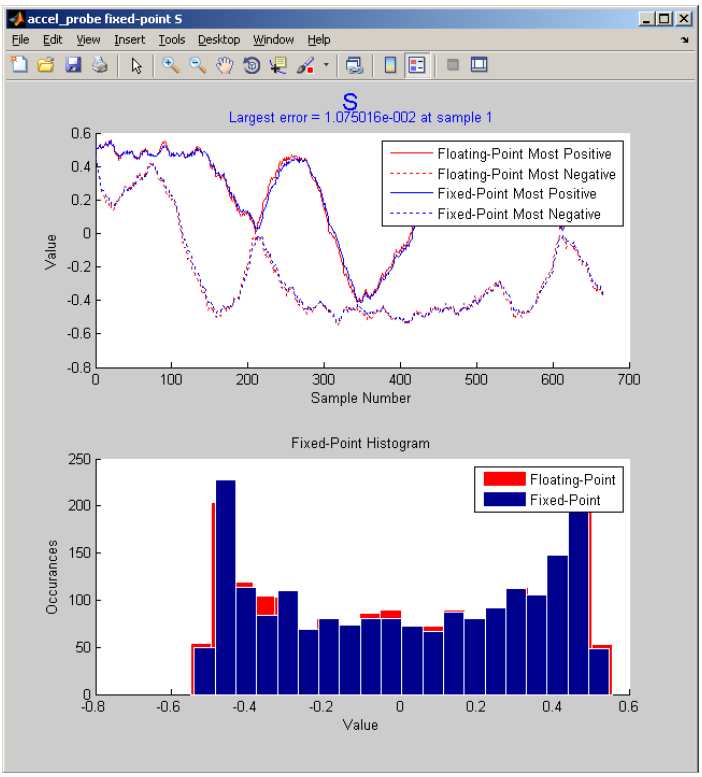


Abb. 32 : Visueller Vergleich der Floating Point und Fixed Point Modelle

Anhand dieser Übersicht kann der Entwickler die Qualität des generierten Fixed Point Modells bewerten und gegebenenfalls die Parameter zur Erzeugung erneut konfigurieren. Das kann einerseits über den Projekt Explorer geschehen oder direkt aus dem erzeugten FixedPointReport (vgl. *Abb. 33*). Sollten die Ausgabewerte des erzeugten Fixed Point Modells zu große Abweichungen von denen des Floating Point Modells haben, können zum Beispiel die Eigenschaften der AccelDSP Quantizer Funktion bearbeitet werden. Die Quantizer Funktion versucht anhand der angegebenen Eingabewerte die optimalen Vektorbreiten der im System verwendeten Hilfs- und Ausgabesignale zu generieren. Sollte zum Beispiel die Zahl 16,125 im System dargestellt werden, so würde die Quantizer Funktion einen 8 Bit breiten Vektor erstellen, wobei 5 Bit zur Darstellung der 16 verwendet werden und 3 Bit für die Nachkommastellen 0,125. Durch das nachträgliche Anpassen dieser Eigenschaften lassen sich höhere Genauigkeiten auf Kosten der Verwendung von

Hardwareressourcen erzielen. Es können noch weitere Eigenschaften zur Erzeugung des Fixed Point Modells bearbeitet werden. So können, die Eigenschaften, die das Verhältnis von *Performance* und *Area* beeinflussen, verändert werden. Der Entwickler hat dabei die Möglichkeit, ob er die Priorität bei der Erzeugung des Systems zum Beispiel auf eine Minimierung der verwendeten Hardwareressourcen oder auf die Optimierung der Laufzeit auslegt.

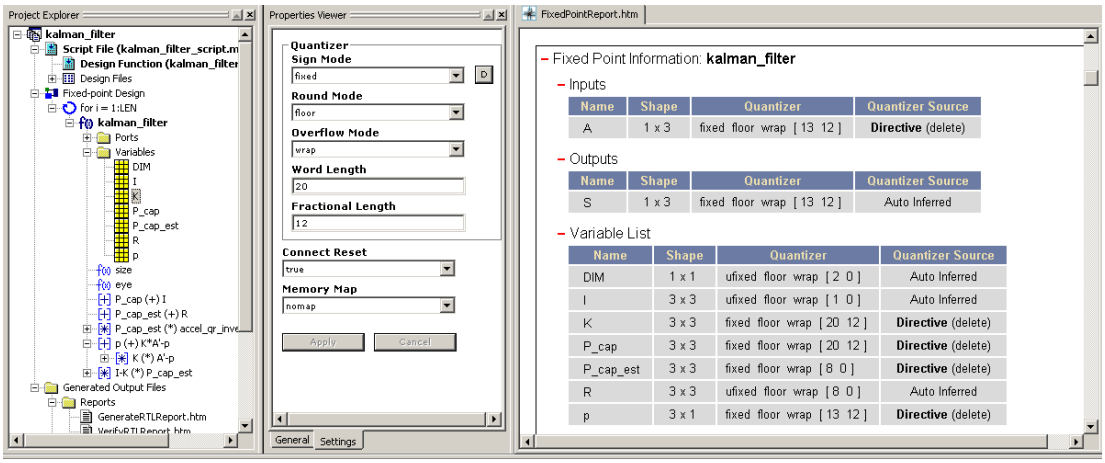


Abb. 33 : Kalibrierung des Systems über den Projekt Explorer oder die erzeugten Reports

Nachdem die Ergebnisse der Fixed Point Modell Erzeugung mit den des Floating Point Modells zufriedenstellend erfüllt wurden, kann das RTL Modell erzeugt werden. In diesem Schritt wird der HDL Code aus den Vorgaben des verifizierten Fixed Point Modells erstelltt, und kann mit Hilfe des ISE Simulators oder durch den Einsatz von ModelSim getestet werden. Der GenerateRTLReport fasst dabei die erzeugten Merkmale des Systems, wie die Anzahl der verwendeten Addierer und Subtrahierer oder die Ein- und Ausgangsignale zusammen. Im nächsten Schritt erfolgt die Synthese des RTL Modells. Dabei wird überprüft, ob die Hardwareressourcen der in den Projekteigenschaften angegebenen Plattform, den Anforderungen des zu generierenden Systems genügen und, ob die Implementierung des Systems auf dieser Plattform überhaupt realisierbar ist. Ein Report fasst wiederum die relevanten Systemeigenschaften zusammen (vgl. *Abb. 34*). Er liefert eine Übersicht zur Anzahl der verwendeten Hardwareelemente, wie Slices, LUTs und IOBs, und Informationen zu zeitkritischen Pfaden und zur *Performance*.

Xilinx XST Synthesis Report

Elapsed Time: 260.08 seconds

– Log File(s)

acceldsp_XST.log

– Part Information

Vendor: Xilinx	
Technology	spartan3e
Device	xc3s1200e
Package	fg320
Speed Grade	-4

– Utilization

Information	Count	Percentage Use
Slices	2732 of 8672	31%
Slice Flip-Flops added for Registered Inputs	14	
Slice Flip Flops	2328 of 17344	13%
4 input LUTs	4965 of 17344	28%
bonded IOBs	0 of 250	0%
MULT18X18SIOs	22 of 28	78%

– Performance Information

Clock Name	Requested Frequency	Estimated Frequency	Estimated Period	Max Throughput	Input Sampling
Clock	50.0 MHz	48.8 MHz	20.4780 ns	355	137.557 KSPS

– Timing Path Information

Clock Name	Path Name	Estimated Frequency	Estimated Period
Clock	Input to Register	236.9 MHz	4.2210 ns
Clock	Register to Register (worst case)	48.8 MHz	20.4780 ns
Clock	Register to Output	1692.0 MHz	0.5910 ns

+ Tool Information

Abb. 34 : Synthese Report zur Darstellung der verwendeten Resourcen und erzielten Laufzeiten

Für die Implementierung des Systems wird dann das Xilinx ISE Werkzeug verwendet, das aus der synthetisierten Netzliste den entsprechenden Bit-Stream innerhalb des Config Files erzeugt. Gleichzeitig wird ein Simulationsmodell erzeugt, um die Funktionalität des erzeugten Systems noch einmal mit der des Fixed Point Modells abzugleichen. Das erzeugte Config File kann in einem weiteren Schritt mit dem Xilinx System Generator zu einem eigenständigen IP Block zusammengefasst werden, was die Verteilung und die Integration in andere Systeme wesentlich vereinfacht.

Für die Umsetzung der Fahrspurerkennung wurde ein Konzept entwickelt [Peters, 2009], welches die geometrischen Zusammenhänge zwischen Bild- und Fahrzeugkoordinatensystem herleitet und die entsprechenden System und Messgleichungen des Kalman-Filters erzeugt. Die Implementierung dieses Konzeptes kann auf Basis des hier vorgestellten Beispiels erfolgen. Dazu müssen die Gleichungen in den entsprechenden MATLAB-Code umgesetzt und die Belegung der Systemressourcen anhand des Synthese Reports analysiert werden.

6 Zusammenfassung

Diese Arbeit unterstützt die Entwicklung von Spurführungssystemen für autonome Fahrzeuge im Kontext des Forschungsprojektes FAUST der HAW Hamburg. Dazu wurde ein FPGA-basiertes System-on-Chip entwickelt, welches eine Fahrspurerkennung durch die Aufbereitung und Verarbeitung von Bilddaten einer digitalen Kamera realisiert.

Die Aufbereitung der parallel bereitgestellten Bilddaten erfolgt durch eine Transformation des ITU-Rec. BT.656 Interlaced Videostrom Formats in ein systemeigenes 24 Bit RGB Format. Dazu wird ein Chroma-Upsampling und die Umrechnung jedes Pixelwertes vom YCbCr Farbraum in den RGB Farbraum vorgenommen Zusätzlich werden die Synchronisationssignale FVAL und LVAL direkt aus dem Datenstrom generiert,

Für die Ausgabe des Bilddatenstroms auf einem VGA-Monitor wurde eine Zeilenverdopplung implementiert. Dadurch kann der Videodatenstrom zur Laufzeit anhand des Monitors verifiziert werden, was für weitere Implementierungen innerhalb der Pipeline einen erheblichen Vorteil darstellt.

Weitere unterstützende Entwicklungen wurden durch den Einsatz eines Microblaze µController Systems auf dem FPGA realisiert. In diesem Rahmen wurde eine Software Komponente entwickelt, welche über eine serielle Schnittstellen mit der Kamera kommunizieren und diese durch die Übertragung von VISCA Befehlen parametrieren kann. Der µController wird darüber hinaus verwendet, um Bilder aus dem laufenden Datenstrom über eine serielle Verbindung des Entwicklungsboards auf einem Desktop PC abspeichern zu können. Die Entwicklung dieser Komponente umfasst zum einen die Erzeugung der Software für die Abspeicherung der Daten und die Kommunikation mit dem PC und zum anderen einen hardwarebasierten Ansatz zum Transport der Daten aus der Pipeline.

Die für die Bildverarbeitung zur Fahrspurerkennung benötigten Nachbarschaftsoperationen, werden über eine Deserialisierung des Videodatenstroms zur Laufzeit realisiert. Somit können relevante Pixel aus dem Gesamtbild extrahiert werden. Diese Pixel stellen Messpunkte auf der Fahrspur dar, die in die Berechnung der Spurführung eingehen.

Um eine robuste Detektion des Fahrspurverlaufs anhand der Messpunkte vornehmen zu können, wurde der Einsatz eines Kalman-Filters vorbereitet. Der erhöhte Aufwand der Implementierung eines Kalman-Filters durch die Verwendung von Matrix Operationen, ist durch den Einsatz des Synthese Werkzeugs AccelDSP optimiert worden. Dieses nimmt in einer Reihe von Entwicklungs- und Verifikationsschritten die Umsetzung von MATLAB-Code in synthesefähigen Quellcode vor, was die Erstellung von komplexen Komponenten eines Systems um ein Vielfaches erleichtert.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literatur

[Abmayr, 1994] ABMAYR,WOLFGANG: *Einführung in die digitale Bildverarbeitung. B. G. Teubner (1994)*

[AccelDSP, 2008] XILINX, INC.: *AccelDSP Synthesis - Tool User Guide (Release 10.1.1). (April, 2008)*

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

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

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

[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 Edition, Elsevier Science & Technology Books (2005)*

[Jenning, 2008] JENNING, EIKE: *Systemidentifikation eines autonomen Fahrzeugs mit einer robusten, kamerabasierten Fahspurerkennung in Echtzeit, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit(2008)*

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

[Nischwitz, Haberäcker, 2004] NISCHWITZ, ALFRED; HABERÄCKER, PETER: *Masterkurs Computergrafik und Bildverarbeitung. Erste Auflage. Vieweg + Teubner (2004)*

[Peters, 2009] PETERS, FALKO: *FPGA-basierte Bildverarbeitungspipeline zur Fahrspurerkennung eines autonomen Fahrzeugs, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit (2009)*

[Shift Register, 2005] XILINX, INC.: *RAM-based Shift Register: Product Specification (v8.0).(April, 2005)*

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

[Spartan-3E, 2008] XILINX, INC.: *Spartan-3E FPGA Family: Product Specification (v3.7). (April, 2008)*

[TheMathWorks, 2009] THEMATHWORKS: *MATLAB Product Homepage (Februat 2009)*
URL: http://www.mathworks.com/products/matlab/?BB=1

[Vanevenhoven, 2007] VANEVENHOVEN, TIM: *Using MATLAB to Create IP for System Generator for DSP. Aus: Xcell Journal, Nr. 62, S. 24-27, (Fourth Quarter 2007)*

[Vesa, 2009] VESA: *Video Electronics Standards Association Homepage. (Februar, 2009) URL: http://www.vesa.org*

[Wikipedia, 2009a] WIKIPEDIA: *BMP file format. (Januar, 2009) URL: http://en.wikipedia.org/wiki/BMP_file_format*

[Wikipedia, 2009b] WIKIPEDIA: *Grauwert Umrechnung. (Januar, 2009) URL: http://de.wikipedia.org/wiki/Grauwert*

A - Quellcode und EDK Projektdateien

Der im Rahmen diese Arbeit erzeugte Quellcode und die Dateien des konfigurierten EDK Projektes befinden sich auf der beigelegten CD.

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, 26. Mai 2009

Ort, Datum

Unterschrift