Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Edris Sahak

SoC-basierte partielle Rekonfiguration einer modularisierten Bildverarbeitungspipeline

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Edris Sahak

SoC-basierte partielle Rekonfiguration einer modularisierten Bildverarbeitungspipeline

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Bernd Schwarz
Zweitgutachter : Prof. Dr. rer. nat. Thomas Canzler

Abgegeben am 12. August 2011

Edris Sahak

Thema der Bachelorarbeit

SoC-basierte partielle Rekonfiguration einer modularisierten Bildverarbeitungs-pipeline

Stichworte

System-on-Chip, FPGA, Partielle Rekonfiguration, PRM, PRR, ICAP, Echtzeit-Bildverarbeitung, Kantendetektion, VGA, DVI, Xilinx System-Generator, Xilinx PlanAhead, I²C

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Entwicklung einer SoC-basierten partiellen und dynamischen Rekonfiguration von Bildverarbeitungsfiltern in einer Pipeline, die in einer Echtzeit-Bildverarbeitungsplattform für die Erkennung der Fahrbahn eingesetzt werden. Die Bildverarbeitungsfilter in den dynamischen Regionen des FPGA werden während des Betriebes vom ICAP-Treiber rekonfiguriert, sodass die statische Region des FPGA weiterhin funktionsfähig bleibt. Als Bildquelle dient eine Kamera oder der VGA-Anschluss des ML-507 FPGA Boards und das Ergebnis der Bildverarbeitung wird über den DVI-Videocontroller CH7301C auf einem LCD-Bildschirm dargestellt. Die Bildverarbeitungsfilter wurden mit dem System-Generator modelliert und synthetisiert und mit PlanAhead in die SoC-Architektur als partiell rekonfigurierbare Module integriert. Der MicroBlaze-Prozessor steuert die partielle und dynamische Rekonfiguration und parametriert die Videokomponenten.

Edris Sahak

Title of the paper

SoC based partial reconfiguration of a modular image processing pipeline

Keywords

System on chip, FPGA, partial reconfiguration, PRM, PRR, ICAP, real time image processing, edge detection, VGA, DVI, Xilinx System Generator, Xilinx PlanAhead, I²C

Abstract

This report deals with the development of a SoC based partial and dynamic reconfiguration of image processing filters in a pipeline, which is used in a real time image processing platform for the detection of lane markings. The image processing filters in the dynamic regions of the FPGA are reconfigured during the operational mode by the ICAP driver, so that other modules remain working. A camera or the VGA port on the ML-507 FPGA board is used as image source and the result of the image processing is shown by the CH7301C video controller on an LCD screen. The image processing filters will be modeled and synthesized using the System Generator and integrated as partial reconfigurable modules into the SoC architecture with PlanAhead. The MicroBlaze processor controls the partial reconfiguration and configures the video components.

Inhaltsverzeichnis

1 Einleitung

Embedded Systeme sind Hardware- und Softwarekomponenten in einem umfassenderen technischen Gerät, dabei sind sie für den Endanwender nicht sichtbar [1]. Viele technische Gegenstände wie etwa Mobiltelefone, Automobile, Flugzeuge, Kameras und Waschmaschinen nutzen diese Technologie. Gerade mal zwei Prozent der produzierten Prozessoren werden in Heim-PCs oder IT-Systemen eingebaut, der Großteil wird als Embedded Systeme in technischen Geräten integriert [2]. Die Bedeutung der Embedded Systeme für die Gesellschaft zeigen die Wachstumsraten der letzten Jahre von durchschnittlich neun Prozent [1].

Embedded Systeme sind eine Querschnitt-Technologie, die Bereiche der Informatik, Elektronik und Mechanik vereint und somit einen Disziplinübergreifenden Systementwurf erfordert, um eine robuste und wartbare Gesamtarchitektur zu definieren [1]. Sie müssen dafür folgende Herausforderungen meistern.

- **Beschränkung der Größe und des Gewichtes**
  Embedded Systeme in mobilen Geräten zum Beispiel Mobiltelefone oder Audio-Geräten sind möglichst klein und wiegen nicht viel, damit eine einfacher Transport und somit auch die Mobilität gewährleistet ist [1].

- **Echtzeitfähigkeit**
  Die Echtzeitanforderung an Embedded Systeme bedeutet, dass eine Aktion innerhalb einer maximal spezifizierten Zeit ausgeführt und beendet wird. Beispielsweise wird in einem Fahrzeug mit Bremsassistent erwartet, dass dieser den Bremsvorgang einleitet und abschließt, bevor es zu einem Unfall kommt. Bei Störungen durch irgendwelche Umstände muss die Echtzeitfähigkeit dennoch eingehalten werden [2].

- **Zuverlässigkeit**
  Embedded Systeme werden lange Zeit betrieben ohne irgendwelche Wartungen oder Patches, deshalb wird eine hohe Zuverlässigkeit erwartet. Ein Herzschrittmacher muss über Jahrzehnte Betrieben werden ohne Komplikationen. Ein Patch für ein solches Gerät oder auch für eine Waschmaschine ist nicht möglich und auch nicht vorgesehen [2].

- **Begrenzte Ressourcen**
  Embedded Systeme haben oft nur wenig Speicher zur Verfügung. Die geringe Wärmeabfuhr erlaubt auch keine hohen Taktraten für die Mikroprozessoren, woraus eine geringe Datenrate resultiert. Entwurfsmethoden die ein Ressourcen-Sharing unterstützen und mit geringen Datenraten effektiv arbeiten können, sind von großem Nutzen [2].

- **Geringer Stromverbrauch**
  Die Reduzierung des Stromverbrauchs spielt eine wichtige Rolle bei den Embedded Systemen und ist ein primäres Ziel, denn dadurch lassen sich die Batterien und Akkus schonen, was eine hohe Laufzeit der Geräte ermöglicht [3].

Die IEEE Computer Society hat 2009 einen Beitrag der Autoren Christof Ebert und Jürgen Salecker veröffentlicht, indem der Trend der Embedded Systeme in Richtung mehr Funktionalität bei geringerem Stromverbrauch geht, das sich nur mit Multi-Core- oder System-on-Chip-Architekturen realisieren lässt [2]. Multi-Core-Architekturen erfordern für eine effiziente Ausbeutung der Performance eine parallele Programmierung, die wiederum eine Synchronisation der Ressourcen und eine Kommunikation zwischen den Prozessoren erfordert. SoC-Architekturen nutzen einen Hauptprozessor und weitere Beschleuniger-Prozessoren für den speziellen Zweck [4].

Diese Arbeit befasst sich mit der Entwicklung einer Echtzeit-Bildverarbeitungsanwendung zur Technologieerprobung von SoC-Entwürfen für Embedded Systeme [5][6]. Die SoC-Bildverarbeitungsplattform enthält dynamisch austauschbare Bildverarbeitungsfilter, die in einer Pipeline für die Kantendetektion und so zur Erkennung der Fahrbahnmarkierung eingesetzt werden. Der MicroBlaze Prozessor wird als Allzweck-Prozessor eingesetzt und die Komponenten der Bildverarbeitung werden als Beschleuniger im Virtex-5 FPGA eingesetzt. Durch partielle Rekonfiguration werden die Bildverarbeitungsmodule der zwei nacheinander geschalteten Bildfilter im Virtex-5 FPGA während des Betriebes mit anderen Bildfiltern ausgetauscht. Als Bildquelle dient eine Kamera oder der VGA-Anschluss des ML-507 FPGA Boards und das Ergebnis der Bildverarbeitung wird über den DVI-Videocontroller CH7301C auf einem LCD-Bildschirm dargestellt.

Die partielle Rekonfiguration von SoC-Plattformen ist eine innovative Technologie, die derzeit nur von Xilinx bereitgestellt wird [7]. Sie schafft die Grundlage für die Entwicklung von RTL-Modellen mit der Eigenschaft ihre Ressourcen zu teilen. Dadurch werden je nach Betriebszustand der SoC-Plattform RTL-Modelle ausgetauscht, so lassen sich komplexe Designs in Embedded Einheiten mit geringer Größe und Gewicht integrieren. Ein weiterer Vorteil der partiellen Rekonfiguration in Verbindung mit Embedded Systemen ist die Fähigkeit, während des Betriebes die Module auszutauschen, so können Funktionen nach Jahren erweitert und verbessert werden.

Die Schwerpunkte dieser Arbeit bilden drei Entwicklungsschritte.

- Kopplung eines Notebooks über die VGA-Schnittstelle an das ML-507 FPGA Board und die Verarbeitung der übertragenen Bilder in Beschleunigermodulen im Virtex-5 FPGA, mit anschließender Darstellung auf einem LCD-Bildschirm (vgl. Bild 1-1).

- Modell-basierter Entwurf von Bildverarbeitungsfiltern zu Kantendetektion und dessen Integration in die SoC-Plattform (vgl. Bild 3-1).

- Entwicklung einer SoC-Architektur mit zwei partiell rekonfigurierbaren Bildverarbeitungsfiltern, die von der internen FPGA-Konfigurationseinheit ICAP

(Internal Configuration Acces Port) rekonfiguriert werden. Der HWICAP IP-Core bindet den ICAP an den Processor Local Bus (PLB) an. Der MicroBlaze Softcore-Prozessor steuert die Rekonfiguration und überträgt die partiellen Bitfiles von der Compact-Flash Speicherkarte an den HWICAP (vgl. Bild 1-1).

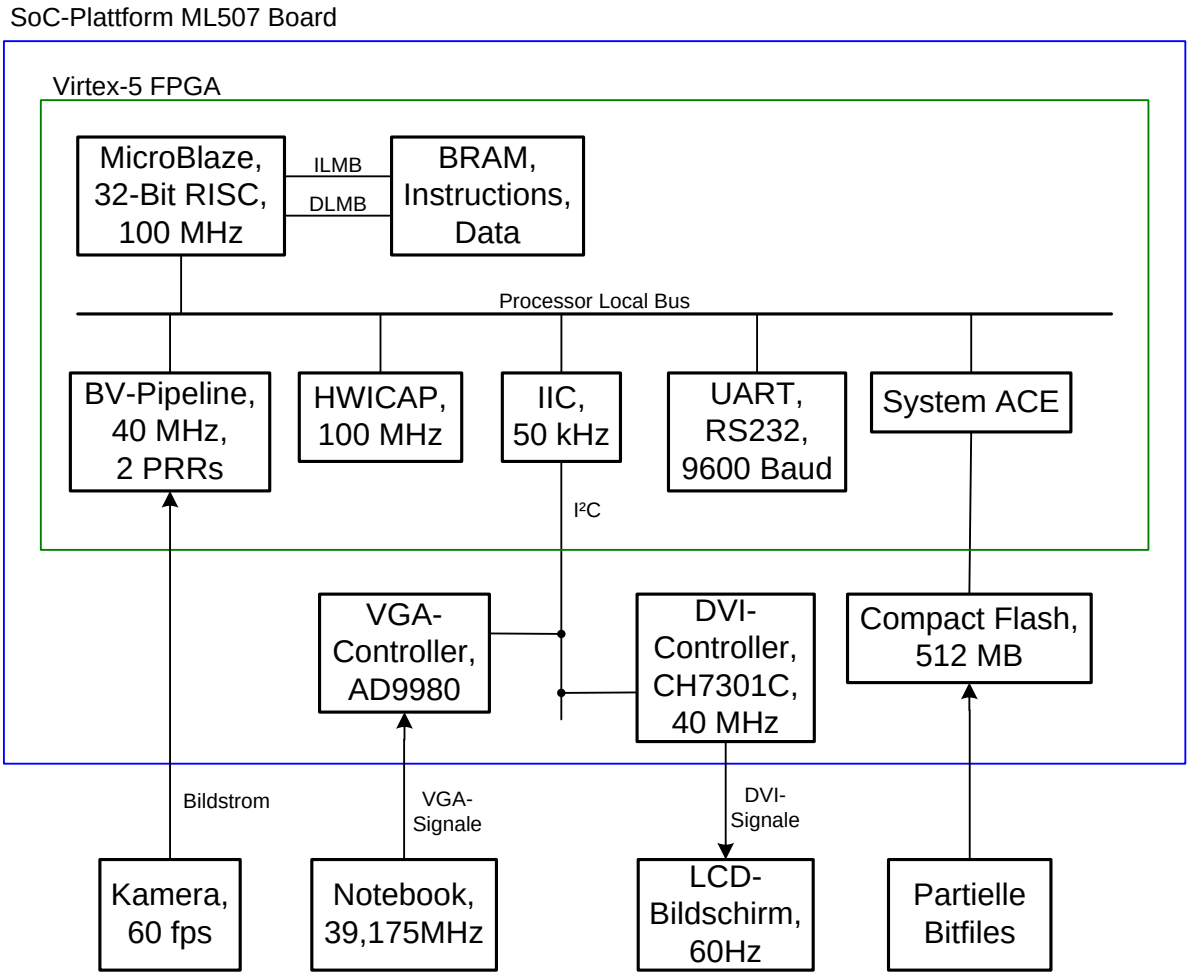


Bild 1-1: SoC-Plattform im Virtex-5 FPGA mit dem MicroBlaze µProzessor, den parallelen Beschleunigern und den partiell rekonfigurierbaren Bildverarbeitungsfiltern in der BV-Pipeline

Für die Kopplung einer Kamera mit dem Virtex-5 FPGA Board und der anschließenden Anzeige auf einem LCD-Bildschirm werden deren technischen Spezifikationen in Kapitel 2 vorgestellt.

Kapitel 3 behandelt die Anwendungsgebiete der SoC-basierten Bildverarbeitung und dessen SoC-Entwurfstechnik. Die Konzepte der partiellen Rekonfiguration und die Bedeutung für die Entwicklung von Embedded Systemen werden ebenfalls erläutert.

Die SoC-Bildverarbeitungs-Komponenten und deren Parametrierung werden in Kapitel 4 vorgestellt, dabei werden auch die DVI- und VGA-Signale näher erläutert. Die Komponenten mit unterschiedlichen Taktfrequenzen werden über einen Zwischenspeicher miteinander gekoppelt.

Kapitel 5 beschreibt die Wirkung und den Ressourcenverbrauch von verschiedenen Bildverarbeitungsfiltern, die mit dem System-Generator Modellierungswerkzeug entworfen und synthetisiert werden.

Die Entwurfsschritte der SoC-Architektur mit dem Xilinx EDK werden in Kapitel 6 behandelt.

Der Entwurfsablauf der SoC-basierten partiellen Rekonfiguration und die Integration der rekonfigurierbaren Module in die SoC-Plattform mit dem PlanAhead Design- und Analysewerkzeug, sowie die Durchführung der partiellen Rekonfiguration über den ICAP-Treiber werden in Kapitel 7 beschrieben.

Die Systemerprobung und die Verifizierung des Implementierten Gesamtaufbaus werden in Kapitel 8 behandelt.

2 Systemübersicht zur SoC-Bildverarbeitungsplattform

Die SoC-Bildverarbeitungsplattform zur Kantendetektion besteht aus einer Bildquelle, einer Bildverarbeitungs-Komponente und einem Anzeigegerät. Als Bildquelle dient eine Kamera, die einen kontinuierlichen Bildstrom mit 60 Bildern pro Sekunde liefert. Die Bildverarbeitung findet in Echtzeit auf dem Virtex-5 FPGA statt und das Ergebnis wird auf einem Bildschirm angezeigt (vgl. Bild 2-1). Die Frequenz des FPGA und die Pixelfrequenz der Kamera sind dieselbe, sodass pro Sekunde 60 Bilder verarbeitet werden.

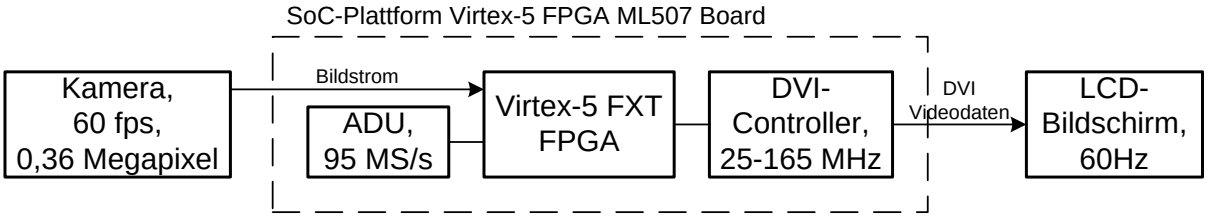


Bild 2-1: Systemkomponenten zur Echtzeit-Bildverarbeitung

2.1 LCD-Bildschirm

Das Ergebnis der Bildverarbeitung wird auf einem LCD-Bildschirm (Liquid crystal display) angezeigt, der ein Raster von Pixeln enthält, die in 16,7 Millionen Farben aufleuchten. Die Anzahl der Pixel ist bei jedem Monitor fest und somit auch die maximale Auflösung, bei der jeder virtuelle Pixel von der Bildquelle auf einen existierenden Pixel des Bildschirms abgebildet wird. Geringere Auflösungen, die den Standards der VESA (Video Electronics Standards Association) entsprechen (vgl. Tabelle 2-1), werden ebenfalls dargestellt.

Auflösung	**Horizontales Blanking – Pixelanzahl der Bilddunkelphase**	**Vertikales Blanking – Zeilenanzahl der Bilddunkelphase**	**Bildwiederhol-rate / Hz**	**Pixelfrequenz / MHz**
640x480	160	45	60	25
800x600	256	28	60	40
1024x768	320	38	60	65
1280x1024	408	42	60	108

Tabelle 2-1: VESA Standards für typische Pixel-Auflösungen [8]

Die LCD-Bildschirme tolerieren minimale Abweichungen von diesen Standards, die über den zusätzlichen Betriebsmodi im jeweiligen Datenblatt spezifiziert sind [9].

2.2 SoC-Entwicklungsplattform Virtex-5 FPGA ML507 Board

Das SoC-Bildverarbeitungs-System wird auf einem ML507 FPGA Board von Xilinx realisiert. Dieses Board verfügt über einen FPGA der Virtex-5 Reihe (vgl. Bild 2-2) und Schnittstellen, wovon folgende für das System in Betrieb genommen worden sind.

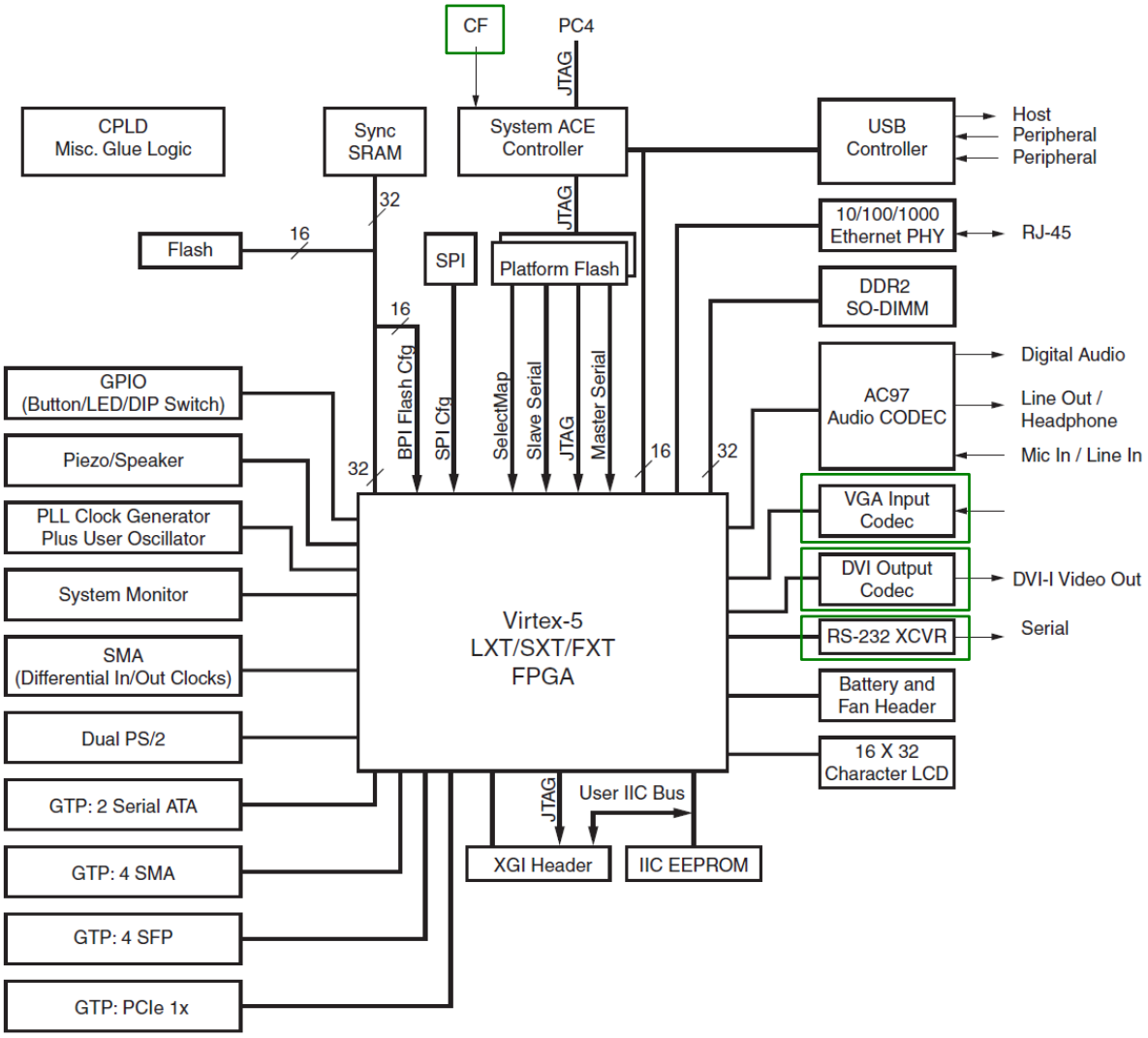


Bild 2-2: Virtex-5 FPGA ML507 SoC Board mit markierten Schnittstellen [10]

- Virtex-5 XC5VFX70T-1FFG1136 FPGA, PowerPC Processor, 11200 Slices, 44800 LUTs, 44800 Flip-Flops, 666 kByte Block-RAM, 640 IOBs [11]
- VGA Connector und Analog Devices AD9980 Video Capture
- DVI Connector und Chrontel CH7301C DVI Transmitter
- Compact Flash Kartenleser
- RS-232 Schnittstelle

Dieses Board besitzt einen DVI Ausgang, der vom Chrontel CH7301C DVI Transmitter getrieben wird und an dem ein Monitor angeschlossen ist (vgl. Bild 1-1). Die höchste Frequenz für den Pixeltakt des CH7301C beträgt 165MHz (vgl. Tabelle 2-2) und die minimale Frequenz von 25MHz (vgl. Tabelle 2-1) wird durch den VESA Standard vorgegeben.

Auflösung	Bildwiederholrate	Pixelfrequenz
640x480	< 85Hz	< 36MHz
800x600	< 85Hz	< 57MHz
1024x768	< 85Hz	< 95MHz
1280x1024	< 85Hz	< 158MHz
1600x1200	< 60Hz	< 165MHz

Tabelle 2-2: Timing-Spezifikation des Chrontel CH7301C [12]

Als Eingang für Bilddaten bietet das Board einen analogen VGA Anschluss, der die Pegel mit dem Analog Devices AD9980 Video Capture (vgl. Bild 1-1) und einer Abtastfrequenz von 95 MHz (vgl. Bild 2-1) digitalisiert. Als höchste Auflösung wird 1024x768 bei 85Hz Bildwiederholrate angegeben [13].

Das Virtex-5 FPGA Board verfügt über 16 differentielle Verbindungen für die Nutzung mit LVDS (Low Voltage Differential Signaling) und 32 Einzelsignal Verbindungen, die als Eingänge oder Ausgänge genutzt werden [10]. Für serielle Protokolle wie Channel-Link [14] bietet das FPGA an den Pins, direkt bei den IOBs (Input/Output Blocks), ein ISERDES (Input Serializer/Deserializer) bzw. ein OSERDES (Output Serializer/Deserializer) Modul an [15]. Die maximale Frequenz für die IOBs ist abhängig von der Technologie und den Einstellungen der Schnittstellen Treiber (vgl. Tabelle 2-3).

Modul	Speed Grade -3	Speed Grade -2	Speed Grade -1
IOB, Input, LVTTL	1,612GHz	1,428GHz	1,219GHz
IOB, Input, LVCMOS 3,3V	1,612GHz	1,428GHz	1,219GHz
IOB, Input, LVCMOS 1,8V	1,492GHz	1,315GHz	1,123GHz
IOB, Input, LVDS 2,5V	1,25GHz	1,111GHz	943,396MHz
IOB, Output, LVTTL 2mA	301,204MHz	277,008MHz	246,913MHz
IOB, Output, LVTTL 24mA	735,294MHz	657,894MHz	574,712MHz
IOB, Output, LVCMOS 3,3V 2mA	340,136MHz	312,5MHz	278,551MHz
IOB, Output, LVCMOS 3,3V 24mA	740,740MHz	662,251MHz	574,712MHz
IOB, Output, LVCMOS 1,8V 2mA	293,255MHz	269,541MHz	240,38MHz
IOB, Output, LVCMOS 1,8V 16mA	694,444MHz	621,118MHz	537,634MHz
IOB, Output, LVDS 2,5V	884,955MHz	775,193MHz	694,444MHz
ISERDES	2,173GHz	1,96GHz	1,666GHz
OSERDES	1,639GHz	1,612GHz	1,612GHz

Tabelle 2-3: Maximal übertragbare Signalfrequenz der IOBs; Taktfrequenz der ISERDES und OSERDES [16]

2.3 Monochrome 0,36 Megapixel 60Hz Kamera

Die Aufnahme eines Bildes in einer Kamera wird über den Bildsensor realisiert, der aus einem zwei dimensionalem Feld mit lichtempfindlichen CMOS (Complementary Metal Oxide Semiconductor) oder CCD (Charge-Coupled Device) Sensoren besteht (vgl. Bild 2-3). Jeder einzelne Sensor ist in der Lage, die Lichtintensität eines Bereiches des Spektrums in elektrische Signale zu wandeln, deren Pegel mit einem Analog-Digital-Umsetzer in einen Pixelstrom digitalisiert werden.

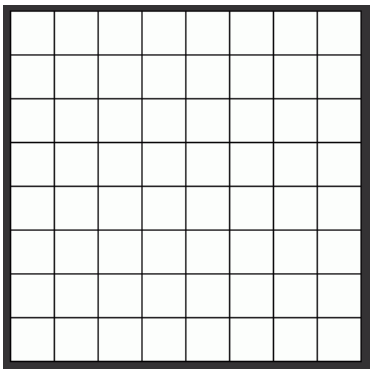


Bild 2-3: Anordnung der lichtempfindlichen Sensoren im zwei dimensionalem Feld [17]; Überlagerung mit Farb-Filtermuster und Demosaicing-Interpolation liefert für jeden Pixel ein RGB-Tripel [18]

Die Kamera liest die einzelnen Werte der Sensoren im Feld pixelweise aus und übermittelt diese parallel oder über LVDS weiter. Die Übertragungsdauer eines kompletten Bildes (T_{Bild}) hängt von der Anzahl der Pixel (H_{Pixel}, V_{Pixel}), der Leerlaufphase des Sensors ($H_{Blanking}$, $V_{Blanking}$) und der maximalen Frequenz des Analog-Digital-Umsetzers (f_{ADU}) ab (vgl. Gleichung 1).

$$T_{Bild} = \frac{\left(H_{Pixel} + H_{Blanking}\right) \times \left(V_{Pixel} + V_{Blanking}\right)}{f_{ADU}} \qquad (1)$$

Die Bilder der Kamera sind nur dann uneingeschränkt auf einem Monitor darstellbar, wenn die Kamera ebenfalls die Standards der VESA unterstützt. Die Kamera muss also mindestens eine Auflösung von 640x480 besitzen und die technischen Spezifikationen müssen eine Bildwiederholfrequenz von 60 Hz erlauben (vgl. Tabelle 2-1).

Die Recherche einer solchen Kamera für die SoC-Bildverarbeitungs-Plattform führte zu dem Ergebnis, dass die Videology 24B752x Kameraplatine mit einen Micron MT9V022 [19] monochrom Bildsensor die Anforderungen erfüllt und mit variabler Linsen-Optik sowie geringer Baugröße für die Verwendung im SoC-Fahrzeug geeignet ist. Der Bildsensor verfügt über eine Auflösung von 752x480 und eine Bildwiederholrate von 60 Hz bei einer maximalen Taktrate von 26,6 MHz. Die Auflösung wird über das I²C (Inter-Integrated Circuit) Protokoll auf 640x 480 Pixel herabgesetzt und der Systemtakt wird auf 25 MHz gesetzt, damit die VESA-Standards erfüllt werden (vgl. Tabelle 2-1). Die Pixeldaten werden seriell im LVDS Standard oder parallel an das Virtex-5 FPGA übertragen (vgl. Bild 2-4) [20].

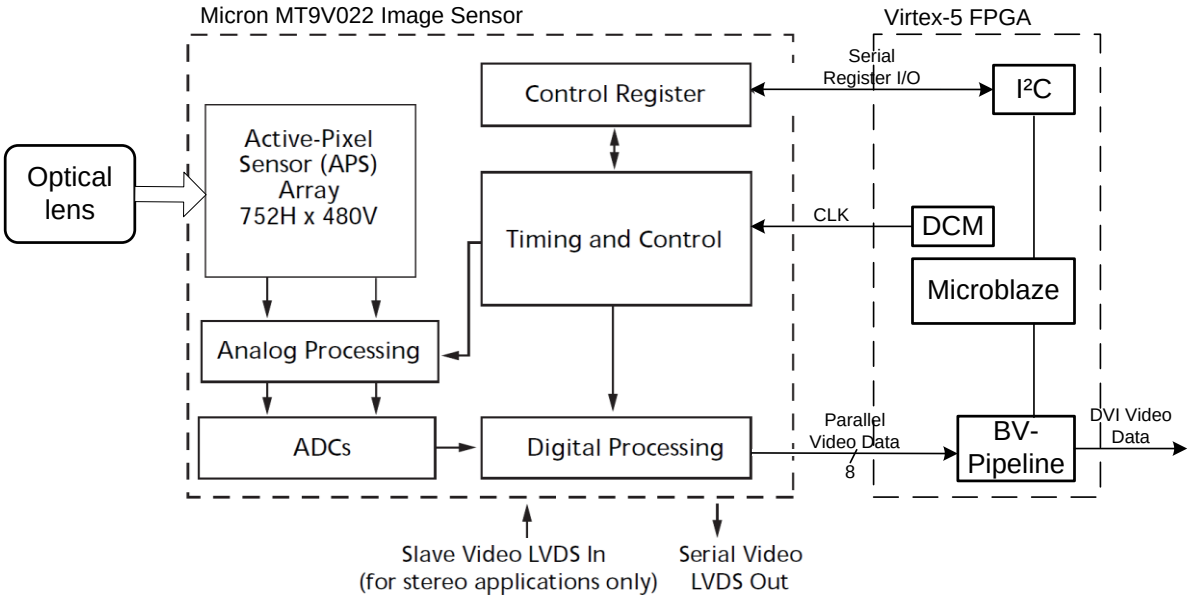


Bild 2-4: Kopplung des Micron MT9V022 Bildsensors an das Virtex-5 FPGA [20]

Bei der parallelen Übertragung wird pro Systemtakt von 25 MHz ein 8-Bit Helligkeitswert generiert und übertragen. Die Übertragung im LVDS Standard nutzt für die differentielle Übertragung einen Kanal, bestehend aus zwei Leitungen, so dass pro Systemtakt ein Paket von 12-Bit Daten übertragen wird. Das Paket besteht aus einem Start Bit, den 8-Bit Helligkeitswert, Line Valid, Frame Valid und dem End Bit [19]. Diese Form der Übertragung erfordert zum Deserialisieren der Daten eine Taktrate von 300 MHz, die mit dem ISERDES Modul im Virtex-5 FXT FPGA realisierbar ist (vgl. Tabelle 2-3).

Das vorgestellte Kamera-FPGA-Interface wird in einem parallelen Projekt entwickelt, deshalb wird hier der VGA Eingang des ML507 Boards als Quelle für den Bildstrom gewählt. An diesen Eingang wird ein Notebook als Bildquelle angeschlossen (vgl. Bild 2-5).

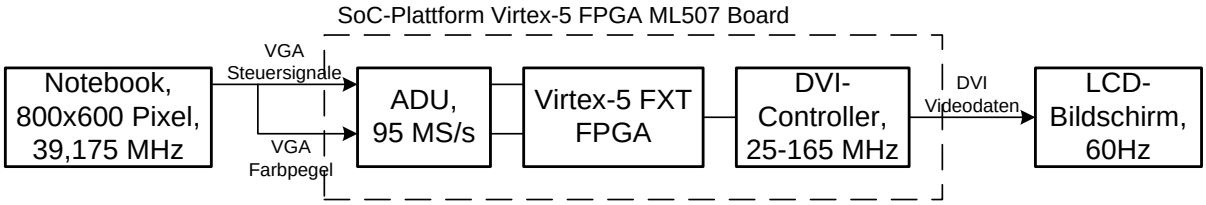


Bild 2-5: SoC-basierte partiell rekonfigurierbare Entwicklungs-Plattform

3 Konzepte und Zusammenhänge

Der Anwendungsbereich dieser Arbeit liegt in der Bildverarbeitung, die über HW-Beschleunigermodule in Echtzeit ausgeführt wird und zur Erkennung von Fahrbahnmarkierungen im SoC-Fahrzeug eingesetzt wird [6]. Die Konfiguration und Steuerung der Komponenten wird über den MicroBlaze-Softcore-Prozessor getätigt (vgl. Bild 1-1), der über das Xilinx EDK integriert wird. Die RTL-Bildverarbeitungsmodule wurden in der Hardware-Beschreibungssprache VHDL und mit dem Xilinx System-Generator, einem modellbasierten Entwicklungstool, entwickelt. Im Folgenden wird der Entwurfsablauf der SoC-Architektur mit partieller Rekonfiguration der RTL-Bildverarbeitungsmodule vorgestellt.

3.1 Anwendungsbereiche der SoC-basierten Bildverarbeitung

Der Fortschritt in der Computertechnologie schaffte die Grundlage für die maschinengestützte und automatisierte Extraktion von Daten aus einem Bild. Mit Bildverarbeitungs-Algorithmen wie Segmentierung, Kantendetektion und Transformation werden beispielsweise Objekte auf einem Bild gezählt und vermessen. Die Industrie nutzt solche Verfahren um Produkte visuell zu inspizieren und auf einen Defekt zu untersuchen [21].

In den letzten Jahren hat ein weiterer Fortschritt der Technologie die Analyse von Bildsequenzen und dadurch die Untersuchung von dynamischen Prozessen ermöglicht [21]. Die Einsatzgebiete der Bildverarbeitung sind gewachsen und an vielen Stellen im Alltag anzutreffen, wie zum Beispiel in Kraftfahrzeugen zur Erkennung der Fahrspur. Die Fahrspurerkennung ist eine Grundvoraussetzung für autonomes Fahren an der auch das Departement für Technische Informatik (TI) der Hochschule für Angewandte Wissenschaften (HAW) in Hamburg arbeitet. Die Modellfahrzeuge sind mit einer Kamera ausgerüstet und ermitteln anhand der Bilder die Fahrbahn, der sie dann folgen [6].

3.2 SoC-Entwurfstechnik mit partieller Rekonfiguration

Die SoC-Architektur besteht aus einem MicroBlaze-Prozessor zur Steuerung und Konfiguration der Komponenten und RTL-Modellen zur Bildverarbeitung, die in der Hardware-Beschreibungssprache VHDL und mit dem Xilinx System-Generator modelliert wurden.

Der MicroBlaze-Prozessor steuert die I²C-Konfiguration des AD9980 Video Capture und des Chrontel CH7301C DVI Transmitter über bereitgestellte Funktionen des I²C-IP-Cores. Die partielle Rekonfiguration über den ICAP-Treiber wird ebenfalls über die Funktionen des HWICAP-IP-Cores gesteuert [22]. Die partiellen Bitfiles sind auf der Compact-Flash Speicherkarte enthalten und werden über den Processor-Local-Bus an den HWICAP vom MicroBlaze-Prozessor übertragen (vgl. Bild 1-1).

Der System-Generator stellt Blocksets für Matlab/Simulink zur Verfügung, die als RTL-Bausteine interpretiert und miteinander verknüpft werden. Aus dem modellierten System wird ein synthesefähiges RTL-Modell generiert, wobei die mathematischen Operationen in Fixed-Point-Arithmetic durchgeführt werden [5].

Die partiell rekonfigurierbaren Bildverarbeitungsfilter werden mit dem System-Generator modelliert und als NGC-Netzlisten synthetisiert. In der erstellten Bildverarbeitungs-IP-Core der SoC-Architektur werden die Bildverarbeitungsfilter als Blackbox instanziiert (vgl. Bild 3-1), dabei entsprechen die Ein- und Ausgangsports der Blackbox den Ports der Filtermodule.

PlanAhead ist ein Design- und Analysewerkzeug für FPGAs, mit dem die Schaltkreise und Pfade im FPGA angelegt werden. Implementierungs-Optionen (z.B. maximale Frequenz oder geringer Ressourcenbedarf) und deren Laufzeiten in den Routing-Pfaden werden mit dem Werkzeug analysiert [23]. Partiell rekonfigurierbare Regionen innerhalb des FPGAs werden definiert und für diese Regionen werden partielle Bitfiles generiert (vgl. Bild 3-2), die als Quelle nur Netzlisten akzeptieren [7]. Im Entwurfsablauf der SoC-Bildverarbeitungs-Plattform wird mit PlanAhead aus der System-Netzliste und den Netzlisten der Bildverarbeitungsmodule eine vollständige System-Bitfile und mehrere partielle Filter-Bitfiles generiert (vgl. Bild 3-1).

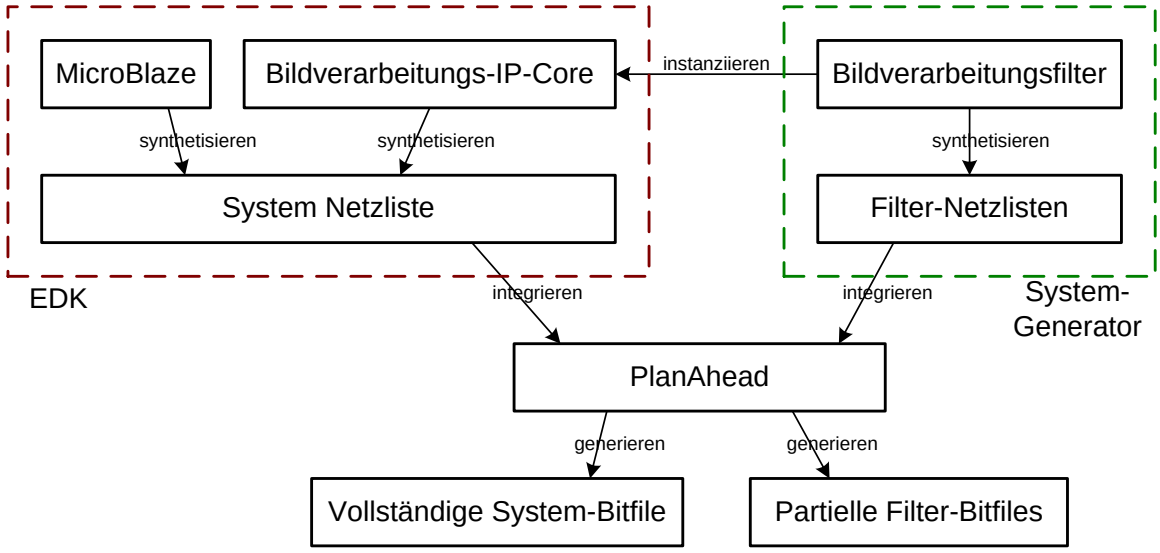


Bild 3-1: Entwurfsablauf für eine SoC-Architektur mit partiell rekonfigurierbaren Modulen

Partielle Rekonfiguration

Für die Nutzung der partiellen Rekonfiguration wird das FPGA in eine statische Region und in eine oder mehrere dynamische und partiell rekonfigurierbare Regionen (PRR) unterteilt (vgl. Bild 3-2). Nicht alle Module werden zur selben Zeit genutzt, somit teilen sich die partiell rekonfigurierbaren Module (PRM) dieselben Ressourcen [7].

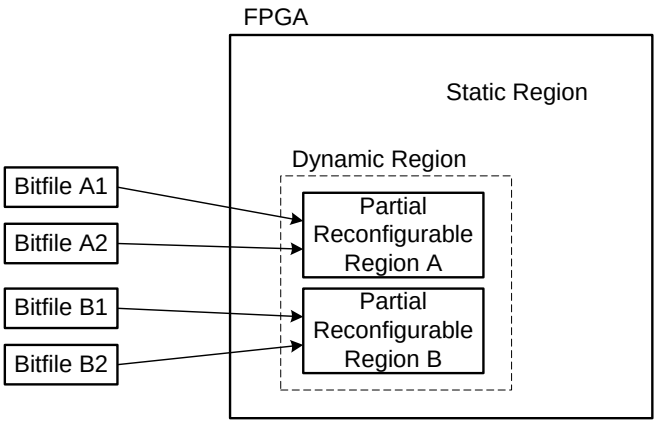


Bild 3-2: FPGA mit zwei rekonfigurierbaren Regionen und dazugehörigen rekonfigurierbaren Modulen

Die partielle Rekonfiguration behandelt drei Design-Anforderungen für die Entwicklung von Embedded-SoC-Plattformen [3].

- **Reduzierung der Kosten und Plattformgröße**
  Ohne die partielle Rekonfiguration wird eine statische Region im FPGA mit der Summe aller Ressourcen der eingesetzten Module angelegt. Die partielle Rekonfiguration erlaubt es Module auszutauschen, die gerade nicht eingesetzt werden. Mit dieser Technologie lassen sich komplexe Designs in FPGAs mit geringen Ressourcen und somit auch geringeren Maßen realisieren, wodurch die Größe der SoC-Plattform verringert wird [3].

- **Ein Implementierung im Betrieb ändern**
  Die Änderung der Konfiguration eines FPGAs hat zur Folge, dass das komplette Design in der Entwicklung neu synthetisiert und intern neu verdrahtet wird. Während der Konfiguration des FPGA wird das System heruntergefahren und dann erneut gestartet. Bei der Nutzung der partiellen Rekonfiguration wird in der Entwicklung nur das modifizierte Modul synthetisiert und mit dem vorhandenen Systemkontext verdrahtet. Während der Rekonfiguration des FPGA ist das PR-Modul nicht verfügbar, jedoch arbeiten die anderen Module im System weiter, so lassen sich Funktionserweiterungen und Verbesserungen leichter im FPGA nachpflegen [3].

- **Reduzierung des Stromverbrauchs**
  Die Senkung des Stromverbrauchs ist für Embedded-SoC-Plattformen ein primäres Ziel, da diese häufig über Batterien und Akkus betrieben werden. Gleichzeitig ist das auch ein Kostenfaktor in der Industrie, denn Akkus mit Kapazitäten über 4000 mAh erhöhen den Preis der SoC-Plattform. Die partielle Rekonfiguration ermöglicht hier eine Senkung des Stromverbrauchs, indem die statisch Region im FPGA klein gehalten wird, da rekonfigurierbare Module in dynamischen Regionen die anwendungsspezifische Funktion ausführen. Sollte eine Funktion zeitweise nicht eingesetzt werden, so wird das Modul durch eine Blackbox mit Durchverdrahtung ersetzt, die keine Schaltlogik enthält. Weniger Logik bedeutet weniger Schaltvorgänge und somit geringere Stromaufnahme [3].

4 Komponenten der Bildstromverarbeitung

Das Notebook übertragt die VGA-Bilddaten, bestehend aus den Steuersignalen und den Farbinformationen (vgl. Bild 2-5), an die SoC-Plattform, die aus den RGB-Tripeln einen Graustufenwert berechnet. Das empfangene Bild wird in einem Dual-Port Block-RAM zwischengespeichert und parallel vom DVI Timing Generator ausgelesen und an zwei partiell rekonfigurierbare Bildverarbeitungsfilter weitergeleitet. Ein Multiplexer entscheidet, welche Daten an den Chrontel DVI Controller übertragen werden, an dessen DVI Anschluss ein Bildschirm angeschlossen wird. Die Einstellungen der Parameter des AD9980 und des Chrontel DVI Controller werden über des I²C Protokoll getätigt (vgl. Bild 4-1).

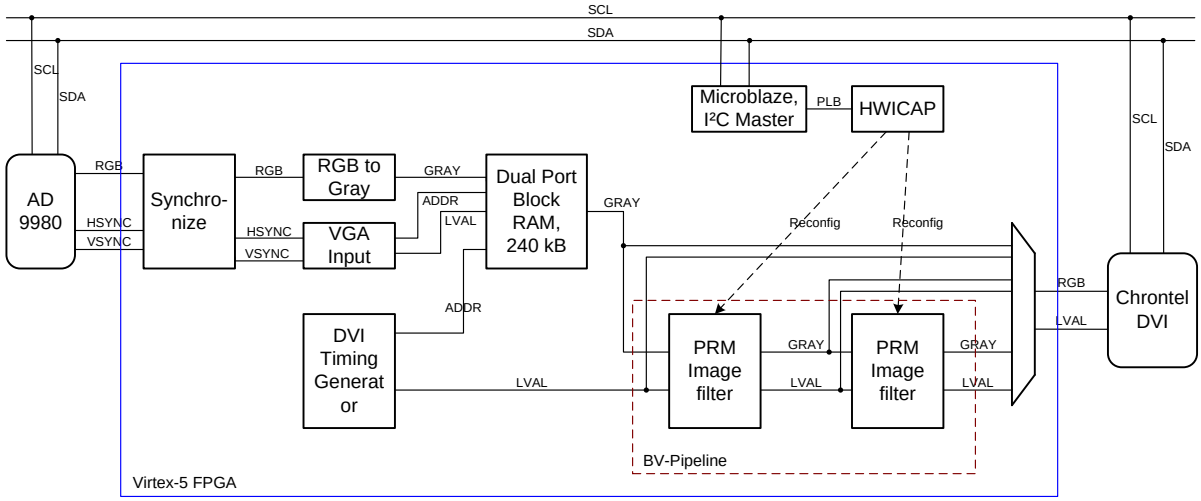


Bild 4-1: Komponenten zur Bildstromverarbeitung und partiell rekonfigurierbare Bildverarbeitungsfilter

4.1 VGA Bildquelle und ADU-Interface für die Erprobungsplattform

VGA beschreibt einen Standard zur Darstellung von visuellen Inhalten auf einem Bildschirm, wobei das Bild bis zur vollständigen Darstellung zeilenweise von oben nach unten aufgebaut wird. Nach der Übertragung der letzten Zeile beginnt die Übertragung der ersten Zeile des neuen Bildes (vgl. Bild 4-2) [24].

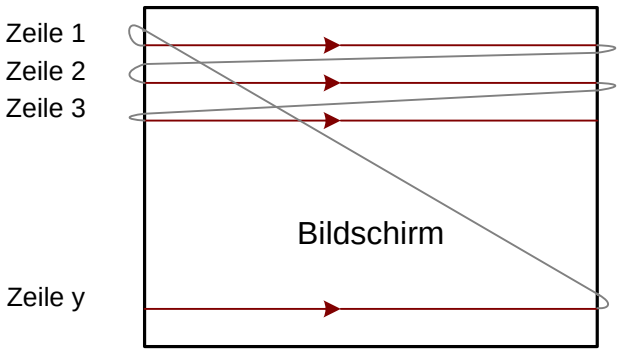


Bild 4-2: Zeilenweiser Aufbau eines Bildes im VGA-Standard

Nach Übertragung einer Zeile beginnt die horizontale Leerlaufphase (Horizontal Blanking), in der das Synchronisationssignal (HSync) den Anfang einer neuen Zeile markiert (vgl. Bild 4-3). Ebenso markiert das vertikale Synchronisationssignal (VSync) den Beginn eines neuen Bildes. Das horizontale Synchronisationssignal wird auch während der vertikalen Leerlaufphase weiterhin übertragen(vgl. Bild 4-4). Innerhalb der Blanking-Phase werden die Zeiten vor und nach dem Synchronisationssignal mit Front- und Back-Porch gekennzeichnet. Die VESA Standards definieren die Polarität der Synchronisationssignale, sowie die Dauer der horizontalen und vertikalen Blanking-Phasen (vgl. Tabelle 2-1, Tabelle 4-1).

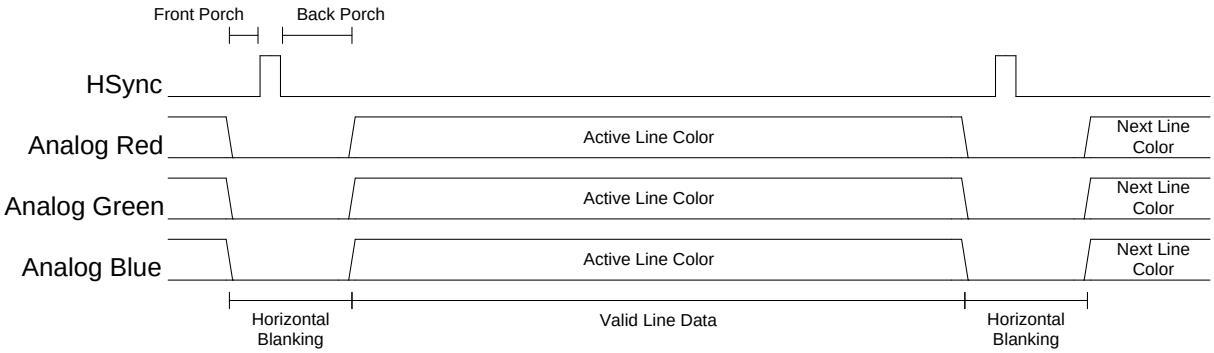


Bild 4-3:Timing-Diagram der VGA-Signale zur Übertragung einer Zeile [24]

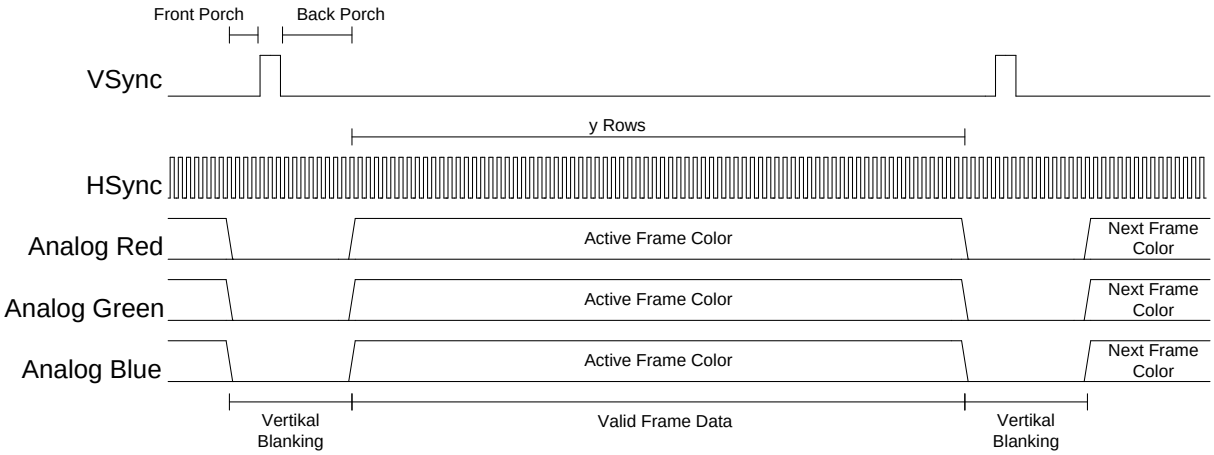


Bild 4-4: Timing-Diagramm der VGA-Signale zur Übertragung eines Bildes [24]

Die Übertragung der Farbinformationen jedes Pixels werden aus der Komposition der Farben Rot, Grün und Blau bestimmt, dabei wird der Anteil über analoge Pegel des jeweiligen Signals bestimmt. Während der Blanking-Phasen werden vom Notebook keine Farben übertragen, sodass dieser die Pegel für die Farben auf ‚0' setzt (vgl. Bild 4-3, Bild 4-4).

Die SoC-Bildverarbeitungs-Plattform nutzt ein Samsung Notebook R510 über VGA als Bildquelle mit einer Auflösung von 800x600 Pixeln (vgl. Bild 2-5) und folgenden Spezifikationen der VESA.

Auflösung	800x600 Pixel
Pixelfrequenz	40Mhz
HSync-Frequenz	37,878kHz
VSync-Frequenz	60Hz
Horizontal Blanking	40 (Front Porch) + 128 (Sync Pulse) + 88 (Back Porch) = 256 Pixel
Vertikal Blanking	1 (Front Porch) + 4 (Sync Pulse) + 23 (Back Porch) = 28 Zeilen

Tabelle 4-1: VESA Spezifikation für eine Auflösung von 800x600 Pixeln [8]

Die VGA Signale werden vom Notebook an den Analog Devices AD9980 Video Capture übertragen (vgl. Bild 1-1), der über drei 8-Bit ADUs verfügt zur Umwandlung der analogen Farbpegel in digitale Werte (vgl. Bild 2-5). Aus der Frequenz des HSync wird der Takt zur Abtastung für die ADUs generiert [13].

Modifikation des VGA-Controller [14]

Die Nutzung des Notebooks als Bildquelle für die SoC-Plattform lässt den Entwickler unabhängig von einer speziellen Kamera werden, daher wird der VGA-Controller [14] modifiziert, sodass ein VGA-Bildstrom wie der Bildstrom einer Kamera (vgl. Kapitel 2.3) interpretiert wird.

Die modifizierte ‚VGA Input' Komponente (vgl. Bild 4-1) berechnet aus den Synchronisationssignalen HSync und VSync die horizontale und vertikale Koordinate des aktuellen Pixels und generiert zusätzlich Line-Valid und Frame-Valid Signale (vgl. Bild 4-6), um als Kamera-Bildstrom interpretiert zu werden. Die genutzte Auflösung und damit verbundene Spezifikation für die Blanking-Phasen werden über Generics eingestellt (vgl. Tabelle 4-2).

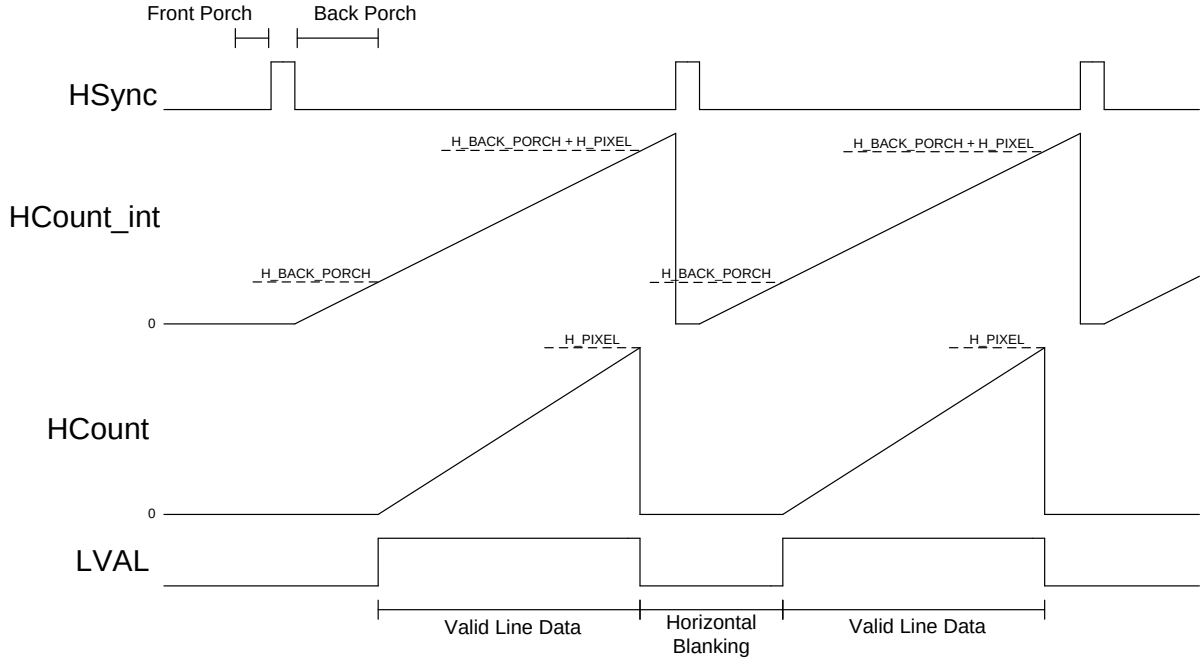


Bild 4-5: Generierung der horizontalen Pixel-Koordinate und des Line-Valid Signals aus dem horizontalen Synchronisationssignal HSync

Generic-Name	Typ	Beschreibung	Konfiguration
H_PIXEL	Natural	Anzahl der Pixel je Bildzeile	800
H_BACK_PORCH	Natural	Dauer des HSync Back Porch in Pixeltaktperioden	88
V_LINES	Natural	Anzahl der Bildzeilen	600
V_BACK_PORCH	Natural	Dauer des VSync Back Porch in Bildzeilenperioden	23

Tabelle 4-2: Generics der ‚VGA Input' Komponente

Die ‚VGA Input' Komponente verfügt über folgende Ein- und Ausgänge, wobei das Taktsignal mit einer Frequenz entsprechend der VESA Spezifikationen zuzuführen ist.

Portname	Typ	Signalbreite	Beschreibung
PIXEL_CLK	Eingang	1	Taktsignal der Komponente
RESET	Eingang	1	Synchroner Reset zum zurücksetzen der Komponente
HSYNC	Eingang	1	Horizontales Synchronisationssignal
VSYNC	Eingang	1	Vertikales Synchronisationssignal
LVAL	Ausgang	1	Line-Valid Signal zur Markierung einer Zeile
FVAL	Ausgang	1	Frame-Valid Signal zur Markierung eines Bildes
HCOUNT	Ausgang	12	Horizontale Koordinate des aktuellen Pixels
VCOUNT	Ausgang	12	Vertikale Koordinate des aktuellen Pixels

Tabelle 4-3: Ports der ‚VGA Input' Komponente

Die ‚VGA Input' Komponente enthält zwei Zustandsautomaten für die Pulserkennung der HSync und VSync Signale, sowie zwei Zählern für die horizontale und vertikale Pixel-Koordinaten (vgl. Bild 4-6). Die horizontale Koordinate wird berechnet, indem ein Zähler jeden Pixeltakt inkrementiert wird. Ein Puls des HSync Signal wird vom Zustandsautomaten erkannt und dieser setzt dann den Zähler zurück (vgl. Bild 4-7). Dasselbe gilt für die vertikale Koordinate, wobei hier der Zähler nach jeder Zeile inkrementiert wird (vgl. Bild 4-8). Von den horizontalen und vertikalen Zählerständen wird immer die jeweilige Back Porch Dauer subtrahiert, so stellen diese dann die Koordinaten des aktuellen Pixels dar (vgl. Bild 4-6). Die Pixel-Koordinaten werden genutzt, um die Adressen im Speicher zu berechnen (vgl. Bild 4-12).

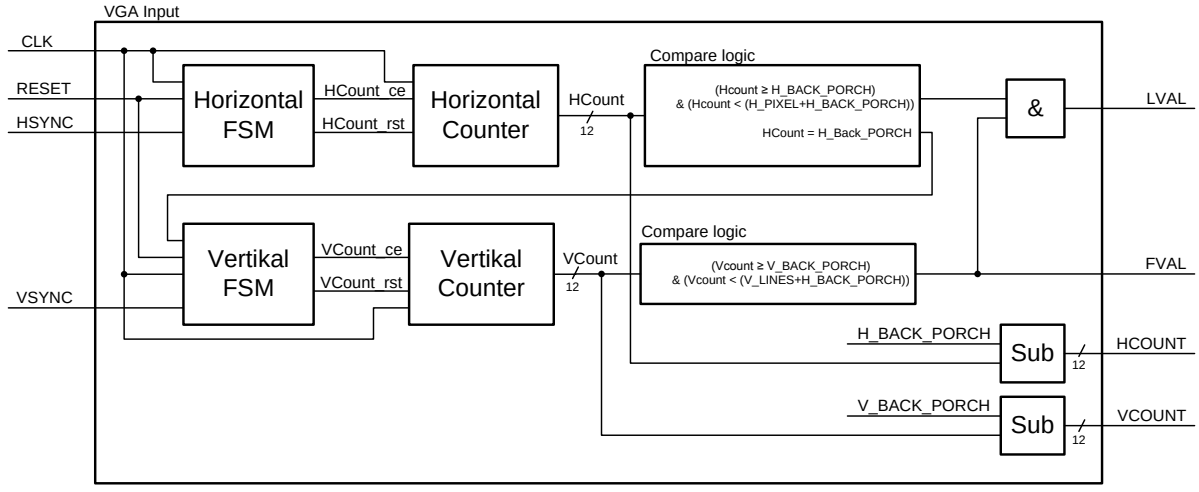


Bild 4-6: Parametrierbare ‚VGA-Input' Komponente (vgl. Bild 4-1) mit den Automaten für die Pulserkennung und Zählern für die Generierung der Pixel-Koordinaten

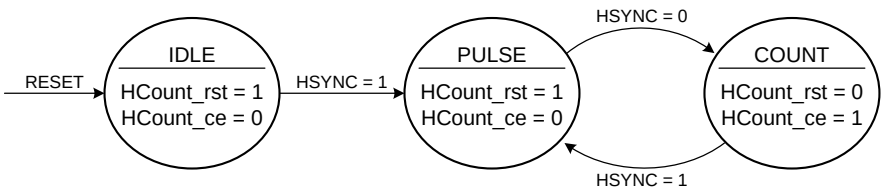


Bild 4-7: Automatenmodell für die Pulserkennung des HSync Synchronisationssignal

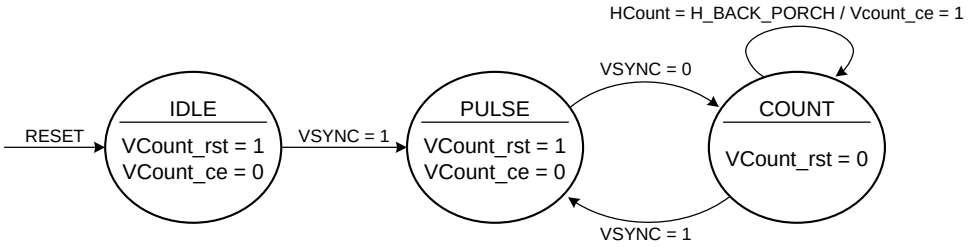


Bild 4-8: Automatenmodell für den Pulserkennung des VSync Synchronisationssignal

Reduzierung der Farbe zu Graustufen

Die verwendeten Bildverarbeitungsfilter (vgl. Kapitel 5) in der SoC-Bildverarbeitungs-Plattform verarbeiteten nur Graustufenbildern, daher ist eine Reduzierung der Farben in Graustufen durchzuführen (vgl. Gleichung 2).

$$Gray = \frac{1}{3} \times (Red + Green + Blue) \qquad (2)$$

Diese Berechnung wird von der ‚RGB to Gray' Komponente (vgl. Bild 4-1) in einer zweistufigen Pipeline ausgeführt, wobei in der ersten Stufe alle Farbwerte addiert werden und in der zweiten Stufe eine Multiplikation mit einer Konstanten stattfindet (vgl. Bild 4-9). Das Trennregister zwischen den Additionen und der Multiplikation minimiert die kombinatorischen Pfade und führt so zu einer höheren Taktung. Die Konstante beschreibt die 1/3 (0x55 in Fixed-Point-Arithmetic) für die Multiplikation.

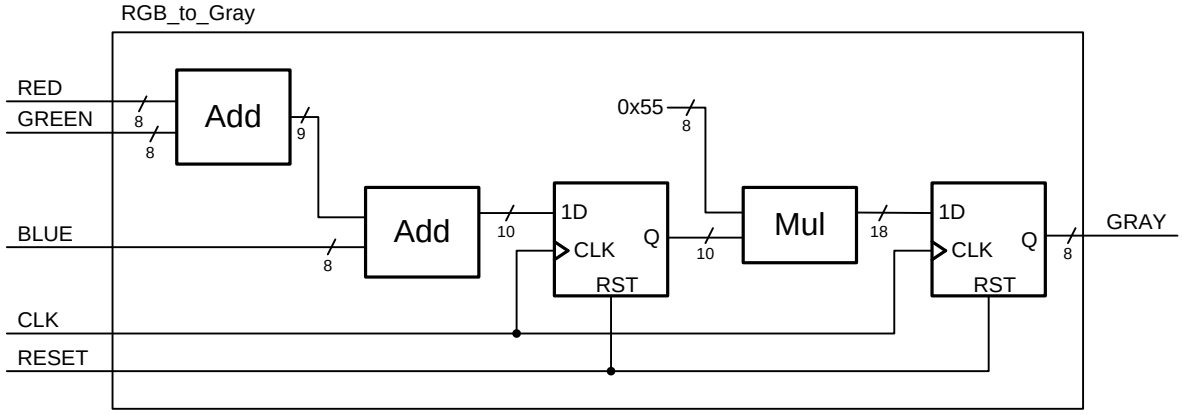


Bild 4-9: ‚RGB to Gray' Komponente (vgl. Bild 4-1) zur Addition der Farbwerte und Multiplikation mit der Konstanten 0x55 (Fixed-Point Repräsentation der 1/3)

4.2 DVI Ausgabe der Bildverarbeitung für einen LCD-Bildschirm

Der Chrontel CH7301C DVI Transmitter serialisiert den Pixelstrom zu einem DVI-Bildstrom und überträgt ihn zur Anzeige an einen Bildschirm (vgl. Bild 1-1). Er nutzt zum Erkennen und Encodieren der HSync und VSync lediglich ein Line-Valid Signal (vgl. Bild 4-1), das die Übertragung von aktiven Video-Daten anzeigt [12]. Insgesamt werden die Pixel-Farbdaten, das Line-Valid Signal und der Pixeltakt als Eingänge an den CH7301C angeschlossen. Das RGB-Tripel der Pixel-Farbdaten wird aus dem Grauwert generiert, indem dieser auf alle drei Kanäle gelegt wird.

Das Digital Visual Interface (DVI) spezifiziert ein digitales Übertragungsprotokoll zur Anzeige auf DVI-fähigen Bildschirmen. Die Daten werden differentiell über Vier Kanäle mit jeweils zwei Leitungen übertragen, wobei drei Kanäle für Daten und einer für den Takt bestimmt sind (vgl. Bild 4-10). In jedem Takt werden 10-Bit Daten mit folgender Zusammenstellung übertragen [25]:

- Kanal 1: Übertragen des 8-Bit Farbwerts für Blau, HSync und VSync
- Kanal 2: Übertragen des 8-Bit Farbwerts für Grün, CTL0 und CTL1
- Kanal 3: Übertragen des 8-Bit Farbwerts für Rot, CTL2 und CTL3
- Kanal 4: Übertragen des Pixeltakts

Die Signale CTL0 bis CTL4 sind Steuersignale, die für anwendungsspezifische Anpassungen genutzt werden. Zum Serialisieren und Deserialisieren der Daten aus den Kanälen eins bis drei wird eine 10-mal höhere Frequenz als die Pixelfrequenz von Kanal vier generiert [25].

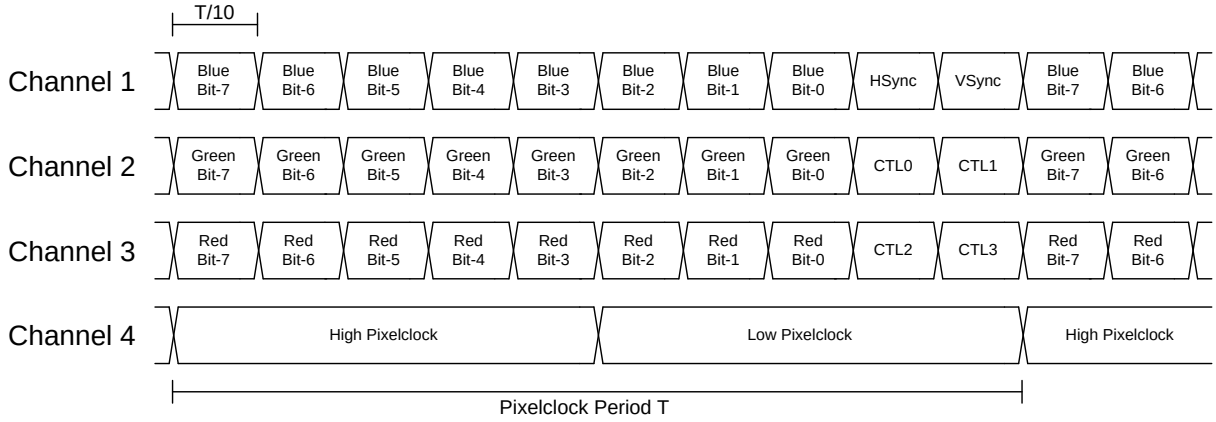


Bild 4-10: Timing-Diagramm der DVI-Signale zur Übertragung des Taktes und eines Pixels über differentielle Leitungen [25]

Das DVI-Protokoll ist unabhängig von den VESA Timing-Spezifikationen, das heißt die Einhaltung von Blanking-Phasen ist nicht erforderlich und somit kann ein Bild in weniger Takten übertragen werden als bei VGA. Da Bildschirme die Blanking-Phasen nutzen, um ihre Zeilen und Spalten auszurichten, ist eine Darstellung auf einem Bildschirm nur unter Einhaltung der Blanking-Phasen möglich. Während der Blanking-Phasen ist nur der Kanal 4 aktiv und überträgt den Pixeltakt [25].

Die ‚DVI Timing Generator' Komponente (vgl. Bild 4-1) generiert horizontale und vertikale Koordinaten (vgl. Bild 4-11) für eine VESA spezifizierte Auflösung, die anhand von Generics konfiguriert wird (vgl. Tabelle 4-4).

Generic-Name	**Typ**	**Beschreibung**	**Konfiguration**
H_PIXEL	Natural	Anzahl der Pixel je Bildzeile	800
H_FRONT_PROCH	Natural	Dauer des HSync Front Porch in Pixeltaktperioden	40
H_SYNC_PULSE	Natural	Dauer des HSync Puls in Pixeltaktperioden	128
H_BACK_PORCH	Natural	Dauer des HSync Back Porch in Pixeltaktperioden	88
V_LINES	Natural	Anzahl der Bildzeilen	600
V_FRONT_PORCH	Natural	Dauer des VSync Front Porch in Bildzeilenperioden	1
V_SYNC_PULSE	Natural	Dauer des VSync Puls in Bildzeilenperioden	4
V_BACK_PORCH	Natural	Dauer des VSync Back Porch in Bildzeilenperioden	23

Tabelle 4-4: Generics der ‚DVI Timing Generator' Komponente

Die generierten Signale Line-Valid und Frame-Valid sind als Ausgänge verfügbar, wovon das Line-Valid Signal von den partiell rekonfigurierbaren Bildverarbeitungsfiltern und dem CH7301C DVI Transmitter genutzt wird (vgl. Bild 4-1).

Portname	**Typ**	**Signalbreite**	**Beschreibung**
PIXEL_CLK	Eingang	1	Taktsignal der Komponente
RESET	Eingang	1	Synchroner Reset zum zurücksetzen der Komponente
LVAL	Ausgang	1	Line-Valid Signal zur Markierung einer Zeile
FVAL	Ausgang	1	Frame-Valid Signal zur Markierung eines Bildes
HCOUNT	Ausgang	12	Horizontale Koordinate des aktuellen Pixels
VCOUNT	Ausgang	12	Vertikale Koordinate des aktuellen Pixels

Tabelle 4-5: Ports der ‚DVI Timing Generator' Komponente

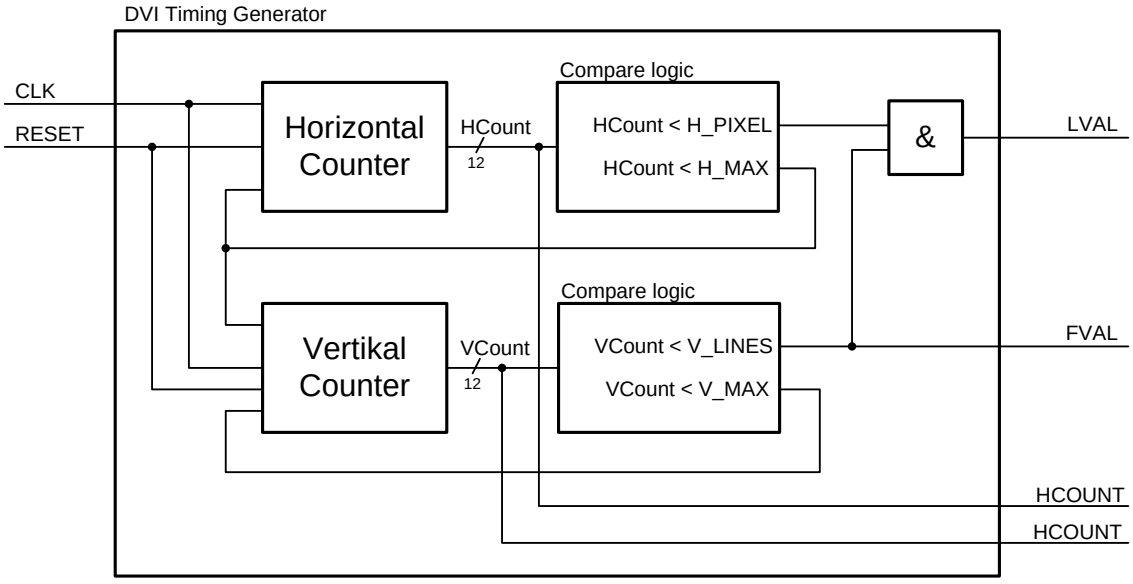


Bild 4-11: Parametrierbare ‚DVI Timing Generator' Komponente (vgl. Bild 4-1) mit den Zählern für die Generierung der Pixel-Koordinaten, dem Line-Valid und dem Frame-Valid Signal

4.3 Zwischenspeicher im Dual-Port Block-RAM

Die VGA Videodaten aus dem Notebook werden mit einer Pixelfrequenz von 39,175MHz an das Virtex-5 FPGA übertragen (vgl. Bild 2-5) und dort mit einer Pixelfrequenz von 40MHz verarbeitet. Es werden somit die Bilddaten mit einer höheren Geschwindigkeit verarbeitet, als sie vom Notebook zum FGPA übertragen werden. In der SoC-Bildverarbeitungs-Pipeline wird durch den Einsatz eines Zwischenspeichers für das Bild (800x600x4) die Kopplung der unterschiedlichen Frequenzen realisiert. Der Einsatz eines FIFOs bietet hier keine Lösung, da dieser nach endlicher Zeit verbraucht wäre und eine wieder Auffüllung aufgrund der niedrigeren Eingangsfrequenz nicht möglich ist. Durch die Zwischenspeicherung des Bildes wird ein kontinuierlicher Bildstrom erzeugt, indem bei nicht vorhandenen aktuellen Pixeldaten (Pixel des aktuell übertragenen Bildes), die Pixeldaten des letzten Bildes in den Bildstrom einfließen. Diese Latenz von maximal einem Bild ist bei einer Bildwiederholrate von 60 Hz für das menschliche Auge nicht zu sehen.

Ein Dual-Port Block-RAM wird als Zwischenspeicher für das Bild genutzt (vgl. Bild 4-12). Der 100MHz Systemtakt der SoC-Plattform im FPGA wird mit einem DCM (Digital Clock Manager) auf 40MHz für die Bildverarbeitung herabgesetzt, da eine Einstellung von 39,175 MHz nicht unterstützt wird. Das VGA Protokoll unterstützt die Übertragung des Pixeltaktes nicht, so ist der Takt der Quelle nicht zu nutzen.

Mit dem Core-Generator wird ein Dual-Port Block-RAM mit folgenden Einstellungen generiert.

- Simple Dual-Port RAM
- Memory size: 240 kByte
- RAM width: 4
- RAM depth: 480000 (800x600)

Aus den 8-Bit Grauwerten der Bildpixel werden lediglich die oberen 4-Bit gespeichert, um die größe des Zwischenspeichers zu reduzieren, dadurch werden nur noch 16 Grauwerte verarbeitet und angezeigt. Der Port-A wird zum Speichern des aktuellen Pixel-Grauwertes in das Block-RAM genutzt, wobei das Line-Valid der ‚VGA Input' Komponente als Write-Enable dient und über den Port-B werden die Bilddaten wieder ausgelesen (vgl. Bild 4-12).

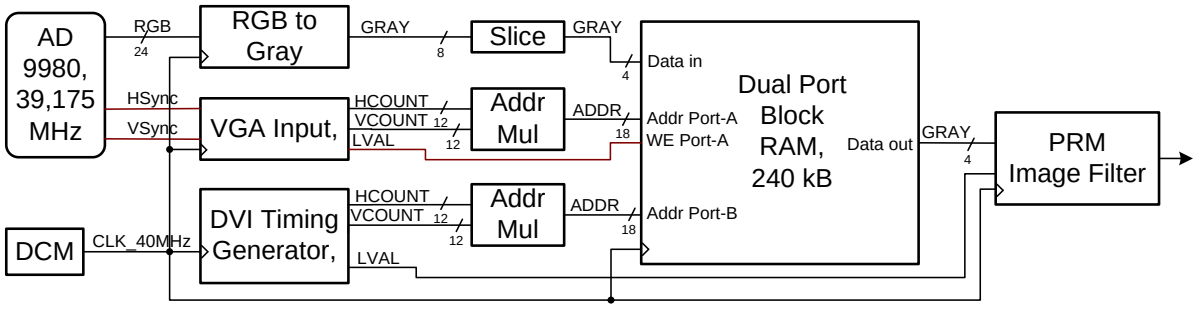


Bild 4-12: Block-RAM als Zwischenspeicher eines Bildes zur Kopplung der Komponenten mit unterschiedlichen Pixelfrequenzen

Der Zugriff auf die Elemente im Speicher geschieht von der ‚VGA Input' und der ‚DVI Timing Generator' Komponente über die Adresse, die sich aus den Pixelkoordinaten berechnen lässt (vgl. Gleichung 3). Sollten Port-A und Port-B im selben Takt auf die gleiche Speicheradresse zugreifen, so wird erst der alte Wert gelesen, bevor der neue eingespeichert wird.

$$Addr = \left(y_{Pixel} \times x_{max}\right) + x_{Pixel} \qquad (3)$$

4.4 Parametrierung der VGA- und DVI-Controller über I²C

Der I²C Bus wurde von Phillips Semiconductors (heute NXP Semiconductors) für die serielle Kommunikation zwischen mehreren Komponenten eines Systems entwickelt. Für die Übertragung werden zwei bidirektionale Leitungen genutzt (vgl. Bild 4-13), die Taktleitung (SCL) und die serielle Datenleitung (SDA), die sowohl die Daten als auch Adressen überträgt [26].

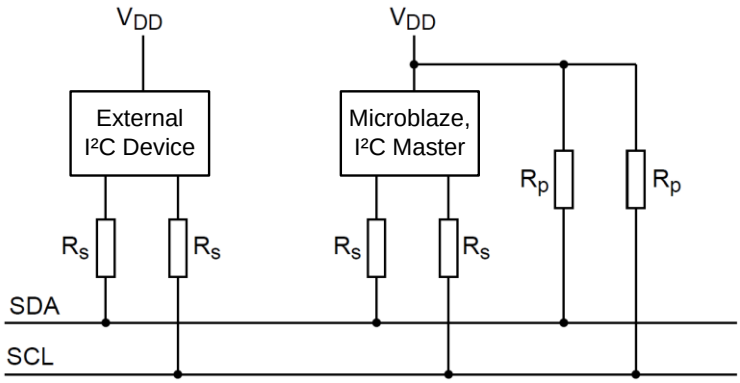


Bild 4-13: I²C Bus mit zwei Komponenten und deren Pullup-Widerständen [26]

Die angeschlossenen Komponenten des I²C Bus agieren als Master oder Slave und verfügen über eine eindeutige Adresse zur Identifizierung. Die Übertragungsrate beträgt 100 kBit pro Sekunde, jedoch ist für spezielle Anwendungen eine Übertragungsrate bis zu 3,4 MBit möglich. Die Anzahl der maximal angeschlossenen Komponenten ist von der Buskapazität abhängig [26].

Die Busleitungen werden im Leerlauf immer auf einen High-Pegel über die Pullup-Widerstände R_p gehalten, sodass nur bei der Übertragung einer digitalen ‚0' die Leitung über den Leitungswiderstand R_s auf einen Low-Pegel gezogen wird. Aufgrund des elektrischen Aufbaus folgt, dass eine Komponente, die eine ‚0' übertragen will, immer die ganze Busleitung auf einen Low-Pegel zieht. Durch diese Eigenschaft werden Kollisionen aufgelöst, indem die Komponente, die gerade eine ‚0' überträgt, priorisiert ist [26].

Eine Übertragung beginnt immer mit einer Start-Kondition und endet immer mit einer Stop-Kondition. Nach jeder Übertragung von acht Bit erfolgt eine Bestätigung der Daten vom Empfänger. Die Übertragung kann angehalten werden, indem der Empfänger die Takt-Leitung auf einen Low-Pegel hält [26].

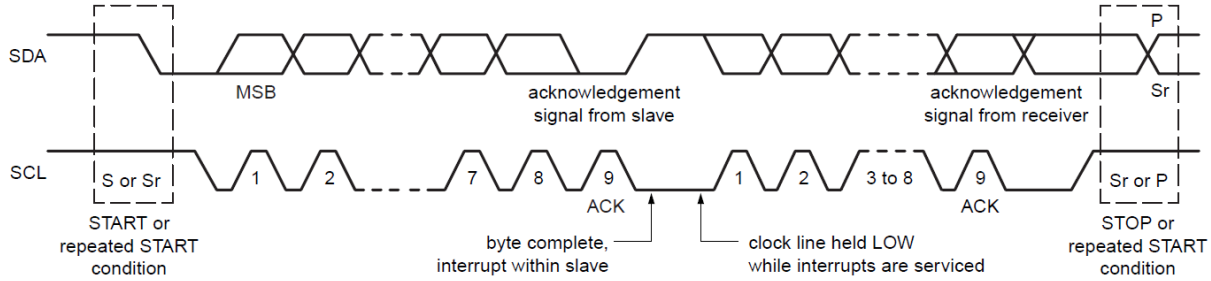


Bild 4-14: Datentransfer über den I²C Bus; Start- und Stop-Kondition markieren Anfang und Ende der Übertragung [26]

Der Chrontel DVI Controller und der AD9980 werden über das I²C Protokoll parametriert (vgl. Bild 1-1, Bild 4-1), dabei werden die Werte für die Register der Komponenten übertragen und gespeichert. Für das Lesen und Schreiben von Registern sind nachfolgende Abläufe einzuhalten [14].

Register beschreiben

- Start-Kondition
- Übertragen der 7-Bit Komponenten-Adresse
- Übertragen einer ‚0' für Schreibzugriff
- Empfang einer ‚0' als Bestätigung
- Übertragen der 8-Bit Register-Adresse
- Empfang einer ‚0' als Bestätigung
- Übertragen der 8-Bit Daten
- Empfang einer ‚0' als Bestätigung
- Stop-Kondition

Register lesen

- Start-Kondition
- Übertragen der 7-Bit Komponenten-Adresse
- Übertragen einer ‚0' für Schreibzugriff
- Empfang einer ‚0' als Bestätigung
- Übertragen der 8-Bit Register-Adresse
- Empfang einer ‚0' als Bestätigung
- Stop Kondition
- Start Kondition
- Übertragen der 7-Bit Komponenten-Adresse
- Übertragen einer ‚1' für Lesezugriff
- Empfang einer ‚0' als Bestätigung
- Empfang der 8-Bit Daten
- Übertragen einer ‚0' zur Bestätigung
- Stop Kondition

Die Repeated-Start-Kondition würde ein Lesezugriff auf ein Register vereinfachen, wird jedoch nicht vom Chrontel CH7301C unterstützt. Die Nutzung der Repeated-Start-Kondition in Verbindung mit dem CH7301C sorgt für einen nicht definierten Low-Pegel auf der Datenleitung, der ein Reset des Systems erfordert [14].

Der AD9980 und der CH7301C benötigen nach einem Zugriff unterschiedlich lange Zeitintervalle, um auf weitere Zugriffe reagieren zu können. Diese Zeitintervalle sind weder im Datenblatt des CH7301C noch im Datenblatt des AD9980 dokumentiert, jedoch ergaben Erprobungen in dieser Hinsicht, dass es bis zu 1ms dauern kann, bis der CH7301C bereit für einen weiteren Zugriff ist.

Xilinx stellt für die Nutzung des I²C Protokolls ein IP-Core Modul bereit, dass Funktionen für das Senden und Empfangen von Daten enthält. Aus den Spezifikationen des AD9980, des CH7301C und der I²C-IP-Core wurden die zwei folgenden Funktionen erstellt.

- *void video_iic_send(u8 slv_addr, u8 reg, u8 byte)*
  Funktion zum Übertragen von 8-Bit Daten über den I²C-Bus (vgl. Anhang C). Über den Parameter *slv_addr* wird die Bus-Adresse des Chrontel DVI-Controller oder des AD9980 eingestellt. Der Parameter *byte* enthält den zu speichernden Wert des Registers *reg*.

- *u8 video_iic_recv(u8 slv_addr, u8 reg)*
  Funktion zum Empfangen von 8-Bit Daten über den I²C-Bus (vgl. Anhang C). Über den Parameter *slv_addr* wird die Bus-Adresse des Chrontel DVI-Controller oder des AD9980 eingestellt. Der Rückgabe-Wert ist der gelesene Wert aus dem adressierten Register *reg*.

5 Bildverarbeitungspipeline zur Kantendetektion

Dieses Kapitel befasst sich mit der Modellierung von Bildverarbeitungsfiltern, die in einer zweistufigen Pipeline zur Kantendetektion eingesetzt werden (vgl. Bild 4-1). Die Bildverarbeitungsmodule werden mit dem System-Generator entwickelt, und anschließend mit PlanAhead als partiell rekonfigurierbare Module in die SoC-Bildverarbeitungs-Plattform integriert (vgl. Bild 3-1). Die System-Generator-Modelle wurden in dem Projekt von Christian Schneider verifiziert [6].

Die modellierten Bildverarbeitungsfilter haben jeweils dieselben Schnittstellen nach außen (vgl. Bild 4-1, Bild 5-1) und sind in jeder Stufe der Pipeline einsetzbar.

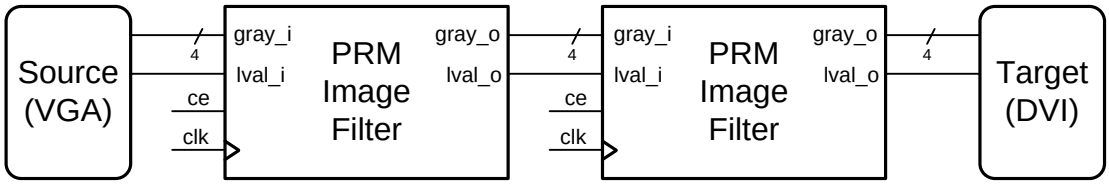


Bild 5-1: partiell Rekonfigurierbare Bildverarbeitungsmodule in einer zweistufigen Pipeline

5.1 Median-Filter zur Reduktion von Rauschen

Lokale Filteroperationen, wie der Medianfilter, berechnen den Wert des Zielpixels aus dem Originalpixel und dessen Umgebung. Der Medianfilter führt eine Rangordnungsoperation durch, indem alle Pixel der Nachbarschaftsmatrix sortiert werden und dann der Median (der mittlere Wert der Folge) als Wert für den Zielpixel gesetzt wird [21].

Bild 5-2: Verrauschte Aufnahme einer Fahrspur; Medianfilter mit einer 3x3 und einer 5x5 Nachbarschaftsmatrix; Diese und die nachfolgenden gefilterten Bilder wurden mit Matlab erzeugt.

In der System-Generator Implementierung wird ein Medianfilter mit einer 3x3 Nachbarschaftsmatrix eingesetzt (vgl. Bild 5-3), um das Rauschen in dem Quellbild zu reduzieren (vgl. Bild 5-2). Eine größere Nachbarschaftsmatrix führt zu einem unscharfen Ergebnisbild.

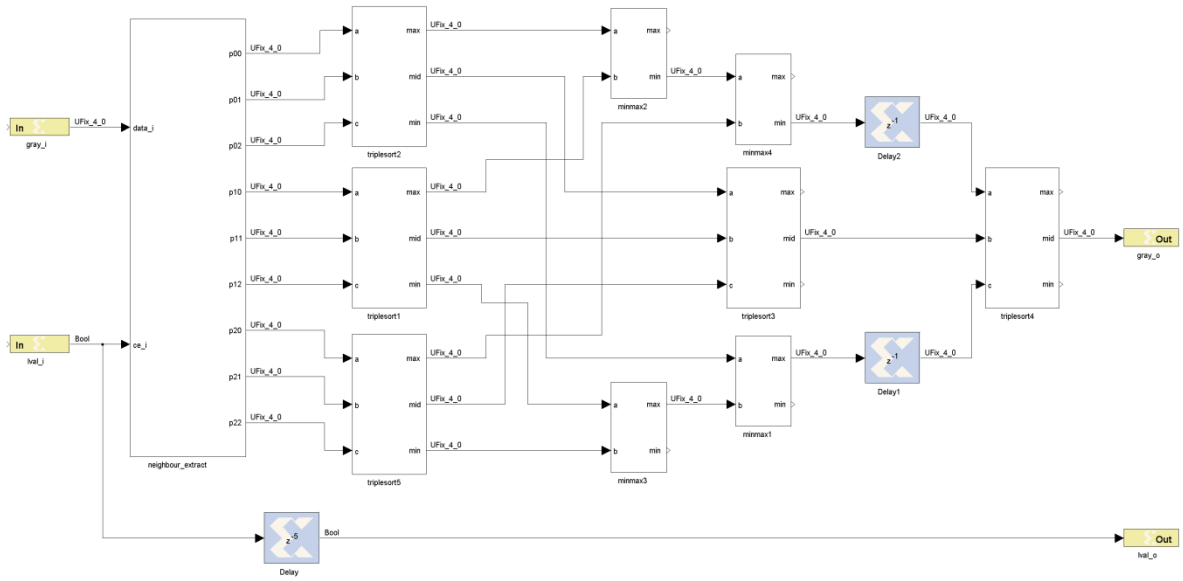


Bild 5-3:3x3 Medianfilter zur Rauschunterdrückung in drei Pipelinestufen (vgl. Anhang A)

Die Nachbarschaftsmatrix wird über eine Kette von Schieberegistern erzeugt, wobei zwei Zeilen zwischengespeichert werden (vgl. Bild 5-4).

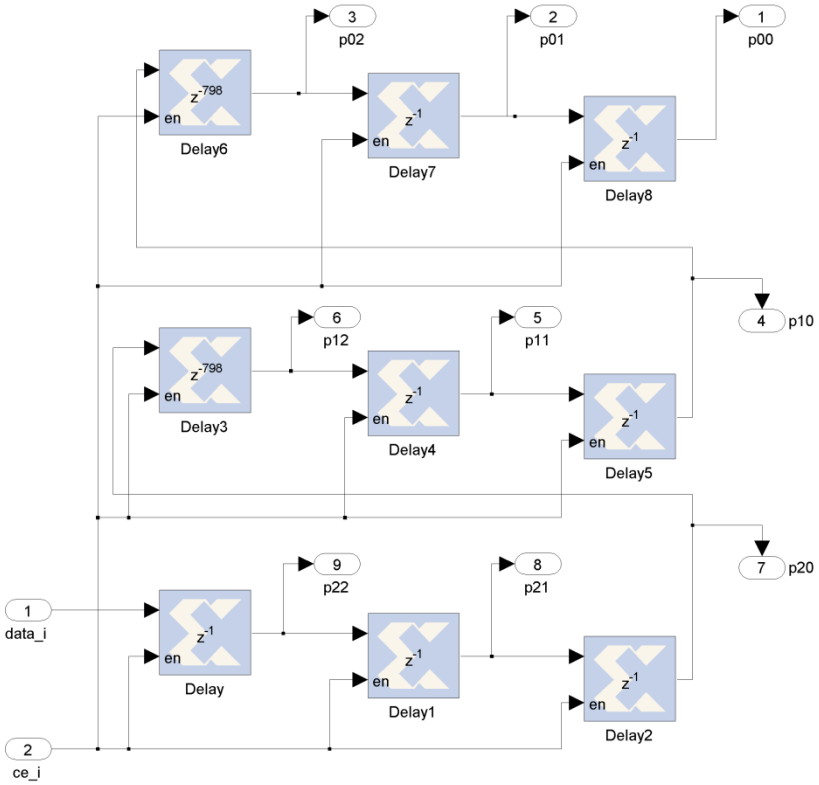


Bild 5-4: Schieberegister-Kette zur Deserialisierung des Pixelstroms zu einer 3x3 Nachbarschaftsmatrix

Die größer/kleiner Vergleiche von zwei oder drei Elementen für die Sortierung (vgl. Bild 5-5) werden in mehreren Pipelinestufen durchgeführt. Die letzte Pipelinestufe enthält den Medianwert der 3x3 Nachbarschaftsmatrix [27].

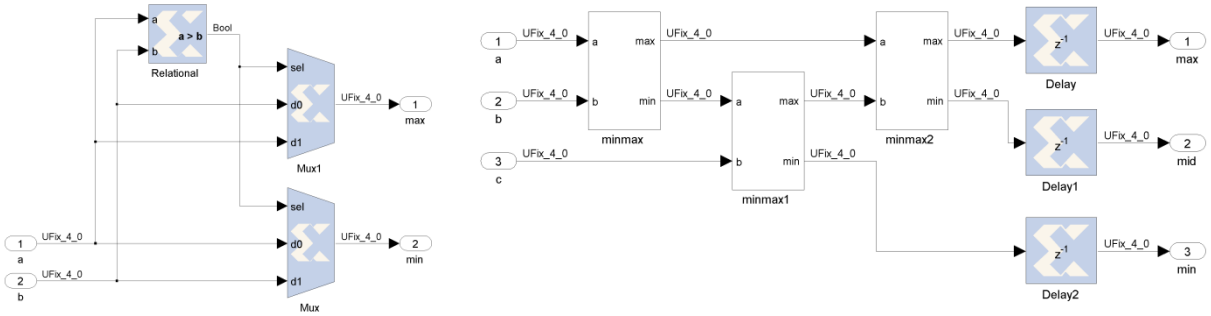


Bild 5-5: Sortierung von zwei bzw. drei Elementen in Fixed-Point-Arithmetic

Die Synthese des Medianfilters nutzt folgende Ressourcen des Virtex-5 FPGA.

Erforderliche Logik	**Anzahl**
Slice Register	458
Slice LUTs	513
Ein-/Ausgänge	12
Maximale Taktfrequenz	123,092 MHz

Tabelle 5-1: FPGA Ressourcenverbrauch und maximale Taktfrequenz des Medianfilters

5.2 Binarisierung zur Hervorhebung der Fahrbahnmarkierung

Bei der Binarisierung handelt es sich um eine Bildvorbereitungs-Operation, die das Graustufen-Bild in ein Schwarz-Weiß-Bild konvertiert, zur Unterscheidung von Vordergrund- und Hintergrund-Daten[21].

Bei der Binarisierung werden alle Werte unterhalb einer definierten Schwelle in Schwarz umgewandelt und alle Werte oberhalb in Weiß. Die Schwelle wird für jede Anwendung subjektiv definiert, um die relevanten Bilddetails (hier die Fahrbahnmarkierungen) zu erhalten (vgl. Bild 5-6). Konstante Schwellwerte sind einfacher zu implementieren als adaptive Schwellwerte, aber im Gegensatz dazu führen sie in kontrastarmen Quellbilder zu schlechten Ergebnissen[21].

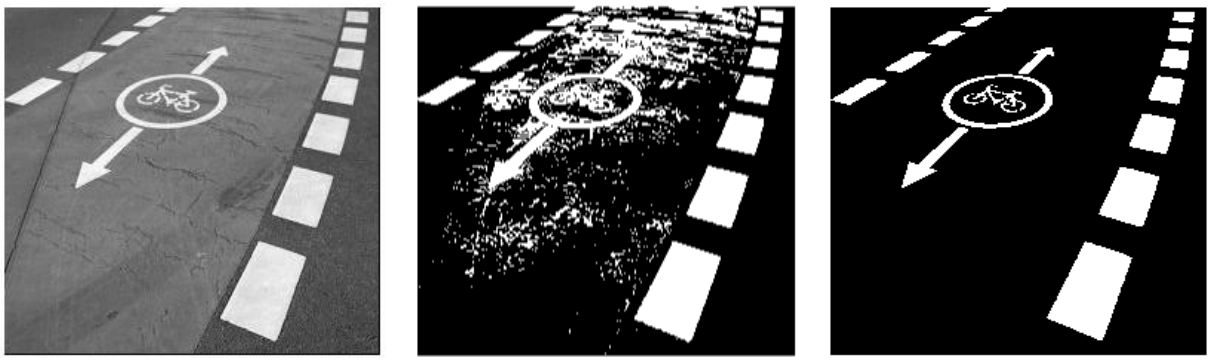


Bild 5-6: Aufnahme einer Fahrspur; Binarisiert mit einem Schwellwert von 50% und einem Schwellwert von 75%

Für die SoC-Bildverarbeitungs-Plattform wurde ein Binarisierungsfilter mit einem konstanten Schwellwert bei einer Intensität von 11 (73,3%) modelliert (vgl. Bild 5-7). In dem System-Generator-Modell wird mit einem Komparator geprüft, ob der Eingangs-Pixelwert oberhalb der Schwelle liegt. Sollte das der Fall sein, so wird der Ausgangswert auf 15 (weiß) andernfalls auf 0 (schwarz) gesetzt.

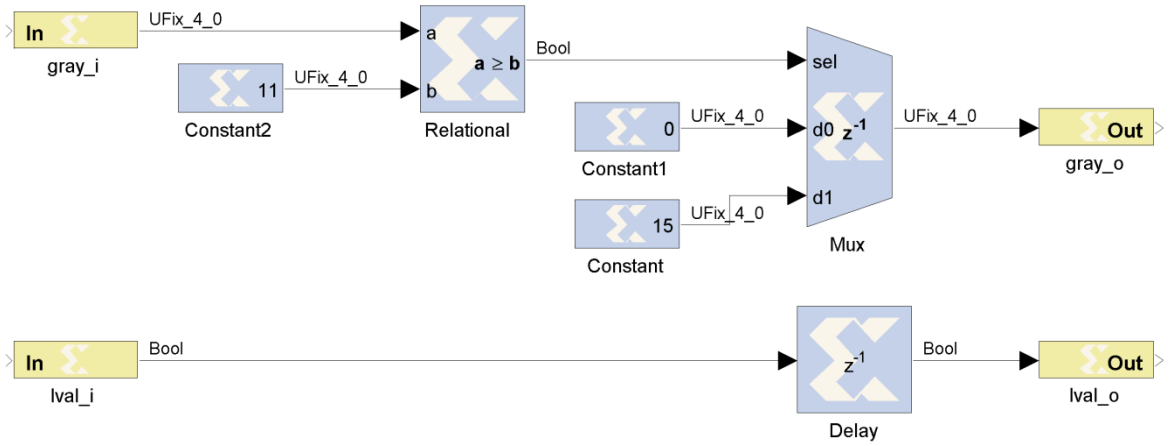


Bild 5-7: Binarisierung der 16 Pixel-Graustufen mit einem Schwellwert von 11 (73,3%)

Die Synthese des Binarisierungfilters nutzt folgende Ressourcen des Virtex-5 FPGA.

Erforderliche Logik	**Anzahl**
Slice Register	1
Slice LUTs	1
Ein-/Ausgänge	12
Maximale Taktfrequenz	618,046 MHz

Tabelle 5-2: FPGA Ressourcenverbrauch und maximale Taktfrequenz des Binarisierungsfilters

5.3 Erosion-Filter zur Ausdünnung der Fahrbahnmarkierung

Der Erosionsfilter ist eine morphologische Nachbarschaftsoperation und führt in Binärbildern zum Schrumpfen von weißen Bildregionen. In Binärbildern wird eine Regionsmatrix aus den Nachbarpixeln des zu verarbeitenden Pixels gebildet. Auf diese Matrix wird eine UND-Operation angewendet, das heißt sollten alle Pixel in dieser Matrix weiß sein, so wird der Pixel im Ziel-Bild auch weiß, in allen anderen Fällen wird der Pixel schwarz. Die Größe der Nachbarschaftsmatrix entscheidet über den Schrumpfgrad (vgl. Bild 5-8).

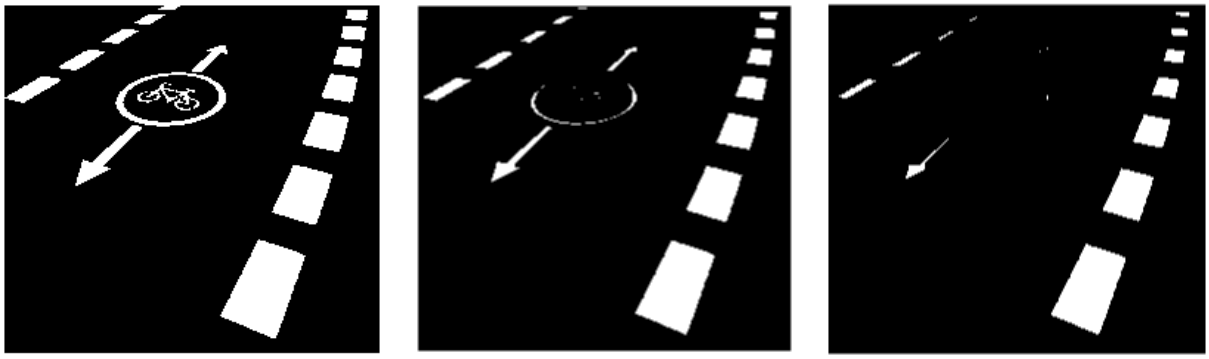


Bild 5-8: Binarisierte Aufnahme (Schwellwert 75%); Erosionsfilter mit einer 3x3 und einer 5x5 Nachbarschaftsmatrix

Die Implementierung für die SoC-Bildverarbeitungs-Plattform nutzt eine 5x5 Nachbarschaftsmatrix (vgl. Bild 5-9), um in einem Binärbild die Fahrbahnmarkierungen auszudünnen.

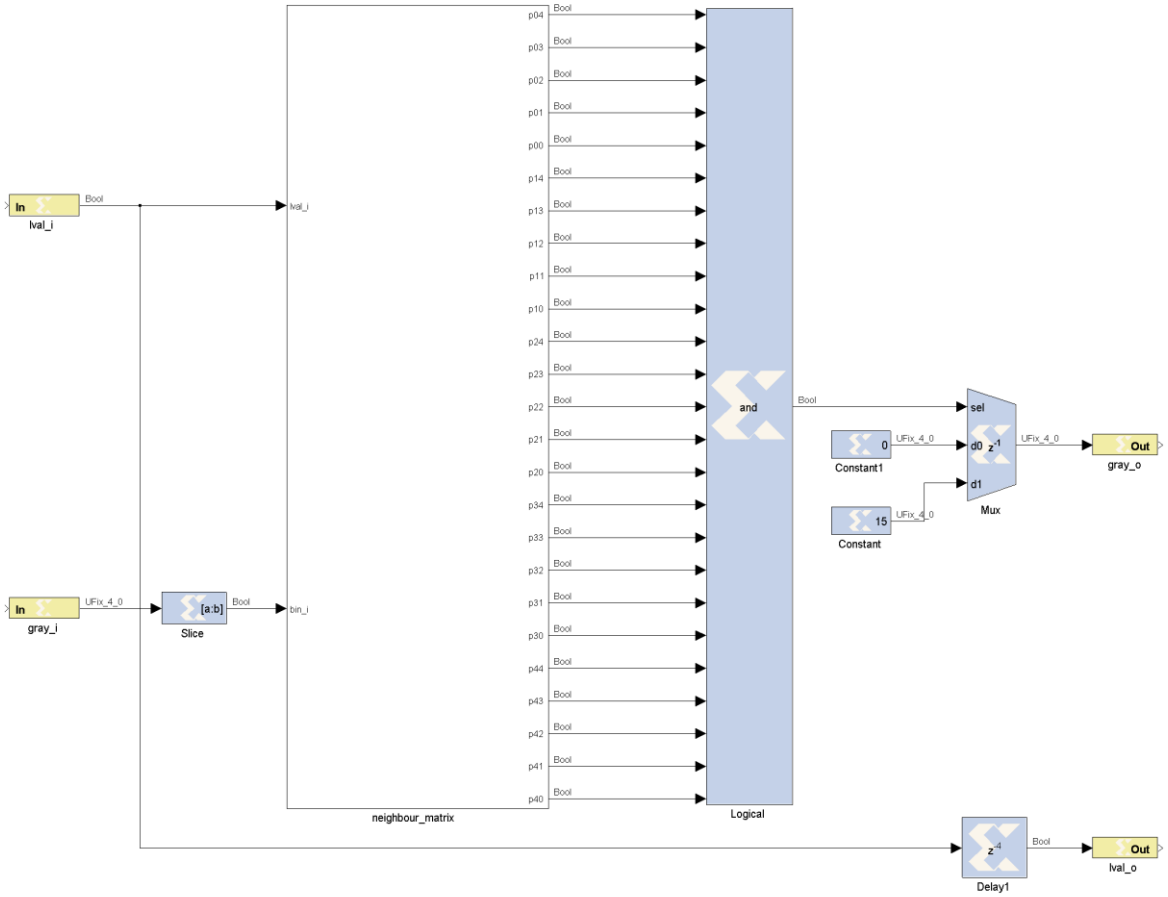


Bild 5-9: 5x5 Erosionsfilter für Binär-Bilder mit logischer UND-Verknüpfung

Die Synthese des Erosionfilters nutzt folgende Ressourcen des Virtex-5 FPGA.

Erforderliche Logik	**Anzahl**
Slice Register	212
Slice LUTs	195
Ein-/Ausgänge	12
Maximale Taktfrequenz	409,333 MHz

Tabelle 5-3: FPGA Ressourcenverbrauch und maximale Taktfrequenz des Erosionfilters

5.4 Sobel-Filter zur Extraktion der Kanten

Das Sobel-Filter detektiert mit einer Faltungsoperation die Kanten in einem Bild. Die 3x3 Nachbarschaftsmatrix des aktuell verarbeitenden Pixels wird mit einer Matrix (Gx) für die vertikale Richtung und mit einer weiteren Matrix (Gy) für die horizontale Richtung multipliziert. Die Beträge beider Produkte werden summiert und deuten bei hohen Werten (in einem Graustufen-Bild entspricht das Weiß) auf eine Kante hin. Die Filterung mit dem Sobel-Filter generiert ein Graustufen-Bild, in dem die Flächen schwarz dargestellt werden und die Kanten weiß (vgl. Bild 5-10). Die Richtungsmatrizen lauten wie folgt[21].

$$Gx = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

Faltungsmatrix für die x-Richtung

$$Gy = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Faltungsmatrix für die y-Richtung

Bild 5-10: Aufnahme einer Fahrspur und Anwendung des Sobelsfilters

Die Modellierung des Sobelfilters für die SoC-Bildverarbeitungs-Plattform multipliziert die Nachbarschaftsmatrix mit der Faltungsmatrix für die jeweilige Richtung (vgl. Bild 5-11). Die Berechnungen werden in der Fixed-Point-Arithmetic durchgeführt.

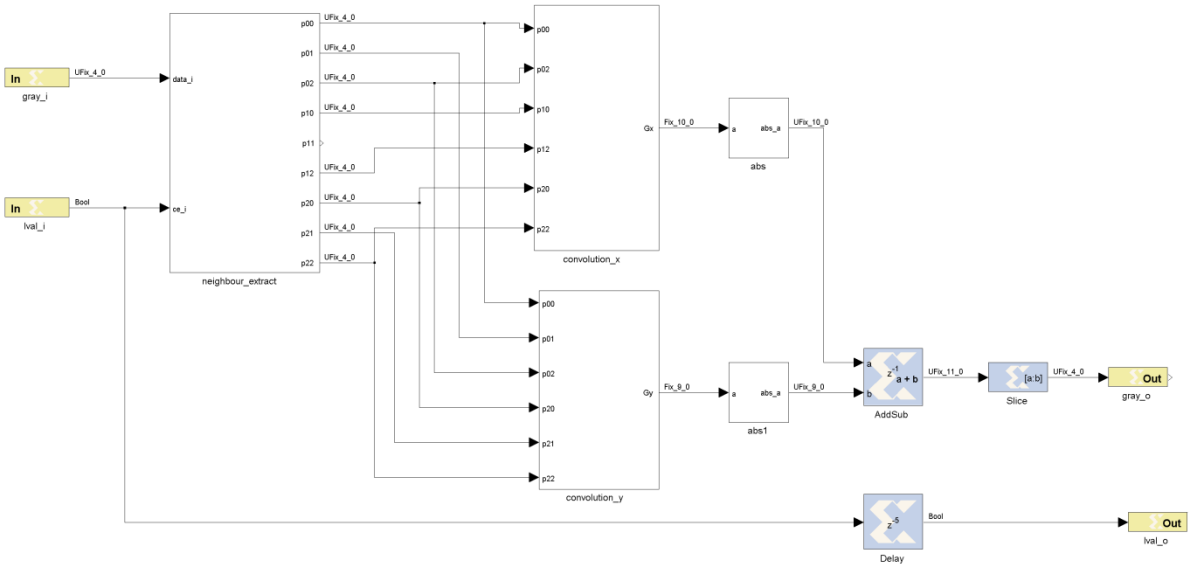


Bild 5-11: Sobelfilter zur Kantenerkennung mit den Faltungsmatrizen für die x- und y-Richtung (vgl. Anhang B)

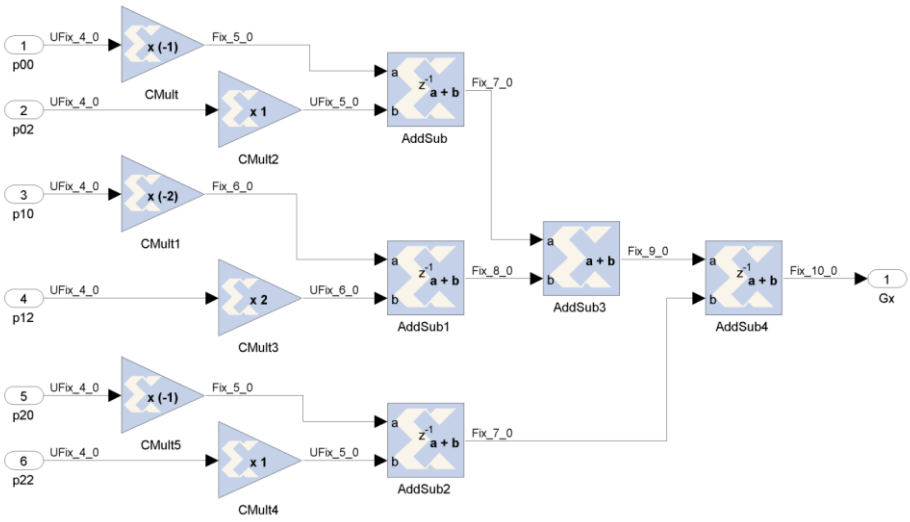


Bild 5-12: Modellierung der Faltungsmatrix in x-Richtung mit Multiplikation in allen Kanälen, um leicht änderbar für weitere Faltungsoperationen (Scharr-Operator, Prewitt-Operator) einsetzbar zu sein

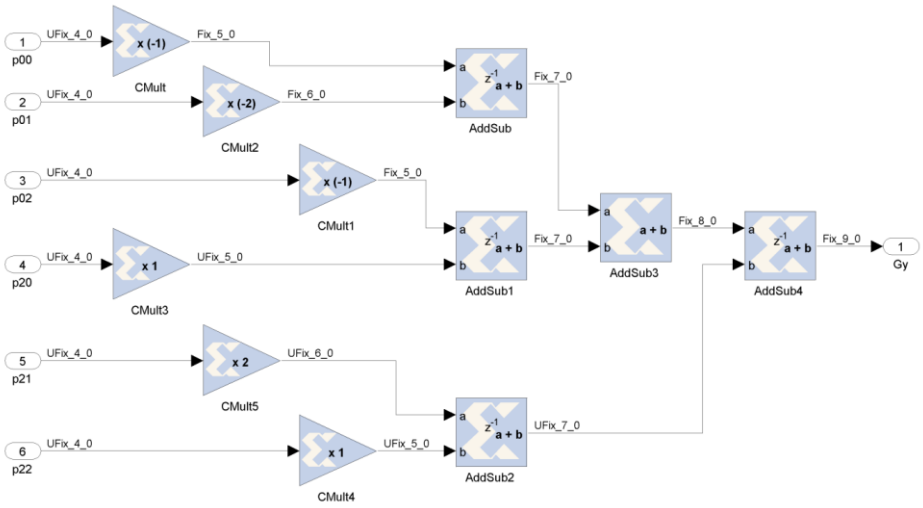


Bild 5-13: Modellierung der Faltungsmatrix in y-Richtung mit Multiplikation in allen Kanälen (vgl. Bild 5-12)

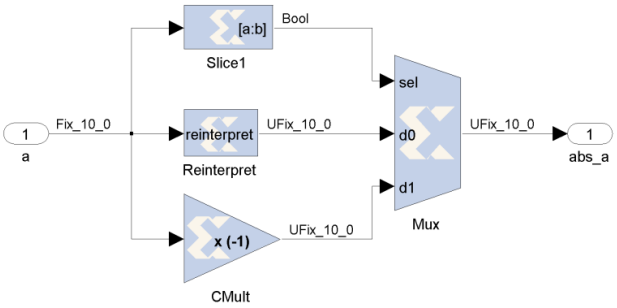


Bild 5-14: Berechnung des Betrages in Abhängigkeit vom Vorzeichen

Die Synthese des Sobel-Filters nutzt folgende Ressourcen des Virtex-5 FPGA.

Erforderliche Logik	**Anzahl**
Slice Register	406
Slice LUTs	509
Ein-/Ausgänge	12
Maximale Taktfrequenz	529,318 MHz

Tabelle 5-4: FPGA Ressourcenverbrauch und maximale Taktfrequenz des Sobel-Filters

6 SoC-Entwurfsablauf im EDK

Die SoC-Architektur wurde über das Xilinx EDK für den Betrieb mit einem MicroBlaze Prozessor erstellt. Die entwickelten Komponenten zur Bildstromverarbeitung (vgl. Kapitel 4) und die modellierten Bildverarbeitungsfilter (vgl. Kapitel 5) wurden in der User-IP integriert. PlanAhead erstellt die Bitfile der SoC-Architektur, sowie die Bitfiles für die PRMs (vgl. Kapitel 7).

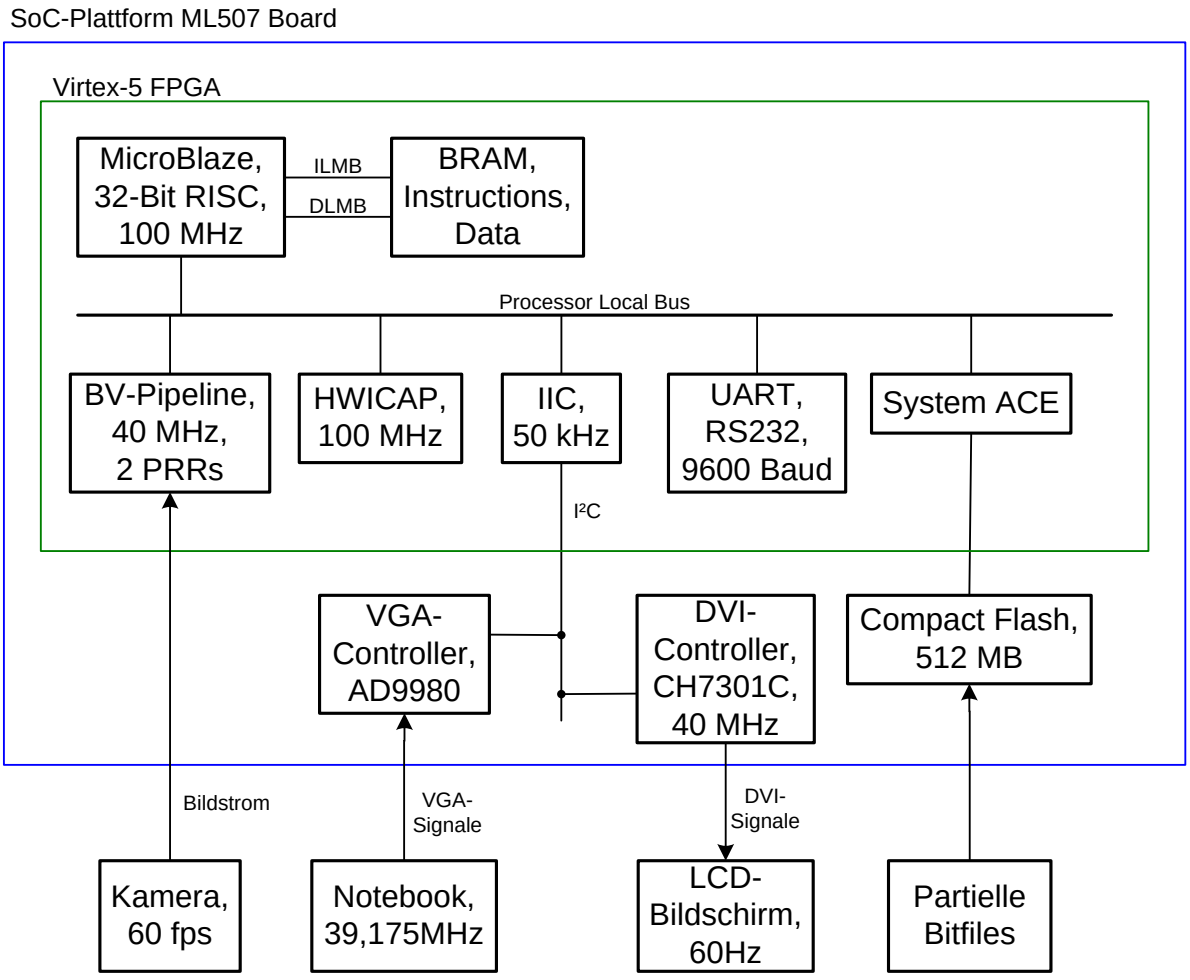


Bild 6-1: SoC-Architektur im Virtex-5 FPGA mit dem MicroBlaze µProzessor, der HWICAP IP-Core zur partiellen Rekonfiguration der Bildverarbeitungsfiltern in der BV-Pipeline und der I²C IP-Core zur Parametrierung des AD9980 und des CH7301C

- **MicroBlaze**, *32-Bit RISC, 100 MHz, 64 kB local Memory*
  Der MicroBlaze ist ein Softcore-RISC-Prozessor, der zur Steuerung der partiellen Rekonfiguration und zur Parametrierung des AD9980 und des CH7301C über das I²C Protokoll (vgl. Kapitel 4.4) genutzt wird.

- **HWICAP**, *100 MHz, 32-Bit Datenbus*
  Diese IP-Core kapselt den ICAP-Treiber für die Nutzung mit dem PLB [22]. Der ICAP-Treiber rekonfiguriert die PRRs in der Bildverarbeitungspipeline mit den PRMs.

- **BV-Pipeline**, *User-IP, 40 MHz, 1 x 32-Bit Register, 2 Partiell rekonfigurierbare Regionen*
  In dieser User-IP ist die Bildverarbeitungs-Kette (vgl. Bild 4-1) implementiert, bestehend aus den VHDL-Komponenten für die VGA- und DVI-Verarbeitung (vgl. Kapitel 4.1, Kapitel 4.2), der NGC-Netzliste des Dual-Port Block-RAM (vgl. Kapitel 4.3) und der Instanziierung der PRMs zur Bildverarbeitung (vgl. Anhang E). In der „user_ip.bbd“ Datei aus dem „pcores/user_ip/data“ Verzeichnis werden für die PRMs keine Netzlisten angegeben, so werden diese als Blackbox Komponenten integriert [28]. Das 32-Bit Register wird für die Schaltung des MUX (vgl. Bild 4-1) genutzt, um zwischen original und gefiltertem Bild zu wechseln.

- **SysACE**, *FAT32 Dateisystem, 512 Byte Blockgröße*
  Die partiellen Bitfiles der PRMs auf der CompactFlash-Speicherkarte werden mit dieser IP-Core über SW-Treiber vom MicroBlaze gelesen und über den HWICAP an in die PRRs übertragen.

- **GPIO-Switches**, *8-Bit*
  Die Auswahl der eingesetzten Filter und die Steuerung des MUX der BV-Pipeline wird über 8 Switches gesetzt.

- **I²C**, *50 kHz, 7-Bit Address mode*
  Diese IP-Core dient zur Konfiguration der Register des CH7301C und des AD9980 über das I²C Protokoll (vgl. Kapitel 4.4).

- **RS232 UART**, *9600 Baud*
  Die serielle RS232 Schnittstelle dient zur Ausgabe von Programmabläufen auf einem Terminal.

Die konfigurierte SoC-Architektur wird mit dem Xilinx EDK zur „system.ngc“ Netzliste ins „implementation“ Verzeichnis synthetisiert. Die partiell rekonfigurierbaren Bildverarbeitungsfilter werden über den System-Generator synthetisiert, sodass diese als Netzlisten verfügbar sind (vgl. Kapitel 5). Nachfolgende Tabelle zeigt die genutzten Ressourcen der SoC-Architektur:

Ressource	**Erforderliche Anzahl**	**Insgesamt verfügbar**
Slice Register	3206 (7%)	44800
Slice LUTs	3136 (7%)	44800
Block-RAM/FIFO	76 (51%)	148
ICAP	1 (50%)	2
Ein-/Ausgänge	94 (15%)	640
Maximale Taktfrequenz MicroBlaze	117,343 MHz	
Maximale Taktfrequenz BV-System	158,227 MHz	

Tabelle 6-1: FPGA Ressourcenverbrauch und maximale Frequenz der SoC-Architektur ohne die partiell rekonfigurierbaren Bildverarbeitungsfilter

7 µProzessor gesteuerte partielle Rekonfiguration des SoC

Der MicroBlaze-Prozessor steuert die partielle Rekonfiguration, indem dieser die partiellen Bitfiles von der Speicherkarte zum ICAP-Treiber überträgt (vgl. Bild 7-3). Nachfolgend wird die Erzeugung der partiellen Bitfiles der Bildverarbeitungsfilter (vgl. Kapitel 5), die Funktionsweise des ICAP-Treiber und die Dauer der partiellen Rekonfiguration gezeigt.

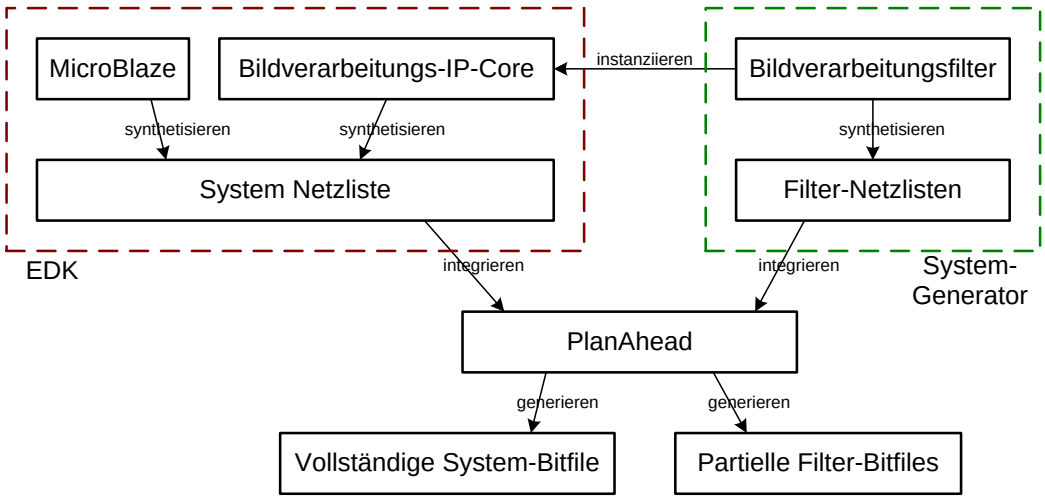


Bild 7-1: Entwurfsablauf zur Generierung von Bitfiles über PlanAhead für das Virtex-5 FPGA; System-Bitfile zur Konfiguration der statischen Region, partielle Bitfiles zur Rekonfiguration der PRMs in den dynamischen Regionen

7.1 Partielle Bitstrom-Generierung mit PlanAhead

PlanAhead ist ein Design- und Analyse-Tool von FPGA-Implementierungen und der Entwicklung von partiell rekonfigurierbaren Designs [23][29]. Nachfolgend werden die Entwurfsschritte erläutert zur Generierung der partiellen Bitfiles für die PRR und der System Bitfile mit der Konfiguration für die statische und dynamische Region des FPGA.

- **PR-Projekt Erstellung**
  PR-Projekte werden nur über Netzlisten erstellt, also keine Unterstützung für VHDL-Code, daher sind alle Module als NGC oder EDIF Netzlisten bereitzustellen

- **Import der System.ngc**
  Die SoC-Architektur in der „system.ngc“ wird aus dem „implementation“ Verzeichnis des EDK (vgl. Kapitel 6) importiert, dabei sind die PR-Bildverarbeitungsfilter in der User-IP VHDL-Datei als Blackbox gekennzeichnet (vgl. Anhang E), da hierfür noch keine Netzlisten vorhanden sind.

- **Definition und Import der PRM**
  Im Netzlisten Editor von PlanAhead werden die Bildverarbeitungsfilter als PRM gekennzeichnet, dadurch lassen sich mehrere Netzlisten diesem Modul zuweisen.
  Für beide PRM werden die generierten Netzlisten importiert, die vorher im EDK getestet wurden, sodass für die PRM vier Netzlisten als Quelle verfügbar sind [28][29].

- **Definition der PRR**
  Im Design Planner von PlanAhead werden zwei identische Regionen bezüglich ihrer verfügbaren Ressourcen markiert (vgl. Bild 7-2), die als PR-Regionen genutzt werden [7][28] [29]. Die PR-Region enthält mindestens so viele Logik-Ressourcen, wie das größte PR-Modul fordert. Für bessere Routing Ergebnisse und spätere Nutzung mit größeren PR-Modulen werden hier PR-Regionen mit fünfmal mehr Logik-Ressourcen definiert (vgl. Tabelle 7-1). Die PRR werden von PlanAhead in der UCF-Datei mit folgender Syntax eingetragen:

```
INST "bv_system_0/bv_system_0/USER_LOGIC_I/FIRST_FILTER_INST" AREA_GROUP =
"pblock_bv_system_0/USER_LOGIC_I/FIRST_FILTER_INST";

AREA_GROUP "pblock_bv_system_0/USER_LOGIC_I/FIRST_FILTER_INST" RANGE=SLICE_X8Y110:SLICE_X19Y159;

INST "bv_system_0/bv_system_0/USER_LOGIC_I/SECOND_FILTER_INST" AREA_GROUP =
"pblock_bv_system_0/USER_LOGIC_I/SECOND_FILTER_INST";

AREA_GROUP "pblock_bv_system_0/USER_LOGIC_I/SECOND_FILTER_INST" RANGE=SLICE_X20Y110:SLICE_X31Y159;
```

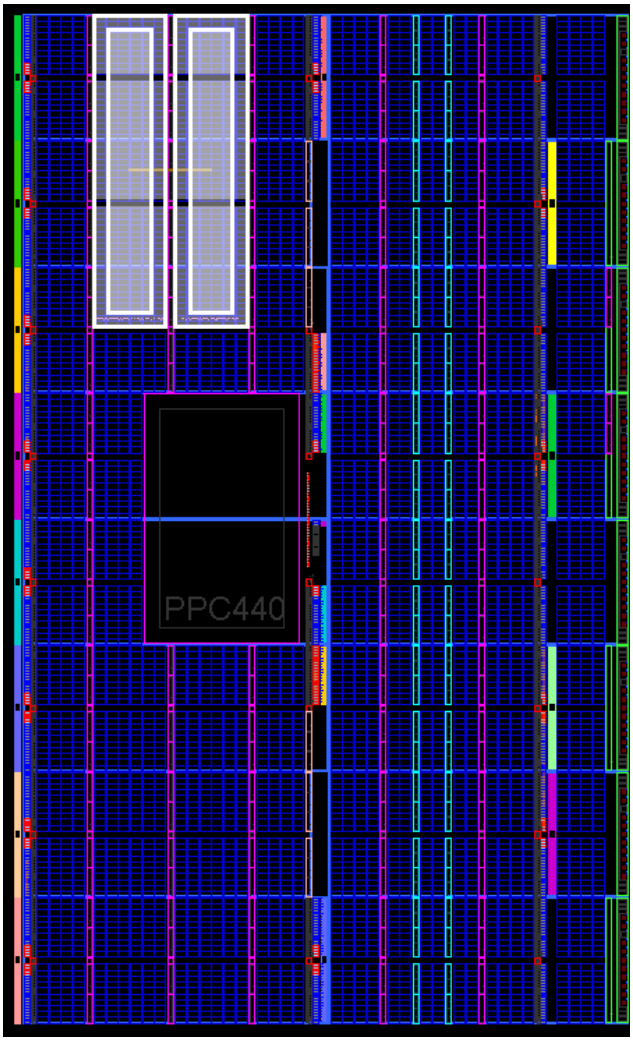


Bild 7-2: Design Planner Ansicht des Virtex-5 FPGA; PR-Regionen sind weiß markiert

Ressource	Insgesamt verfügbar	Erforderliche Anzahl
Slice Blöcke	600	
Slice Register	2400	458 (19%)
Slice LUTs	2400	513 (21%)

Tabelle 7-1: Verfügbare FPGA Ressourcen der PRR und maximal erforderliche Anzahl von der PRM (vgl. Tabelle 5-1); Beide PRR belegen 11% der Virtex-5 FPGA Ressourcen

- **Synthese zu Bitfiles**
  Es werden für beide PR-Bildverarbeitungsfilter ein Initial-Modul aus den vier PRM ausgewählt (hier die Binarisierung-PRM) und zusammen mit der SoC-Architektur zu einer System Bitfile mit der Konfiguration für die statische und dynamische Region des FPGA synthetisiert. Für jedes PR-Modul wird eine Konfiguration erstellt, in der die System-Bitfile als Import gewählt wird, damit die Konfiguration für die statische Region des FPGA nicht nochmals synthetisiert wird. Die einzelnen Konfigurationen werden als partielle Bitfiles synthetisiert [28][29].

- **Integration des C-Code**
  Der C-Code wird mit dem SDK zu einer ELF (Executable and Linkable Format) Datei kompiliert. Über die Xilinx Bash-Shell wird aus der System-Bitfile und der ELF Datei ein SoC-Bitfile generiert, die direkt in das FPGA geladen wird [28].

7.2 ICAP-Treiber für den Bitstromtransfer der PRMs

Der ICAP-Treiber (Internal Configuration Acces Port) ist ein Hardware-Modul im FPGA, welches im laufenden Betrieb die Konfiguration des FPGAs ändert. Diese Technologie ändert die Konfiguration der Slices und Flip-Flops in den CLBs und modifiziert dadurch die Schaltkreise und Funktionalität des FPGA [22].

Die HWICAP-IP-Core kapselt den ICAP Treiber und bietet eine Schnittstelle zum PLB an [22]. Der MicroBlaze startet die Rekonfiguration des FPGA durch die Übertragung der partiellen Bitfile an den HWICAP [28]. Die eingehenden Bitfiles werden vom HWICAP in einem 32-Bit breiten und mindestens 64 stufigen FIFO zwischengespeichert und anschließend an den ICAP-Treiber weitergeleitet [22]. Die Bitfiles werden an den MicroBlaze über Protokolle, wie SPI oder TCP-IP, aus Quellen, wie Netzwerk oder Speicherkarten übertragen (vgl. Bild 7-3) [7].

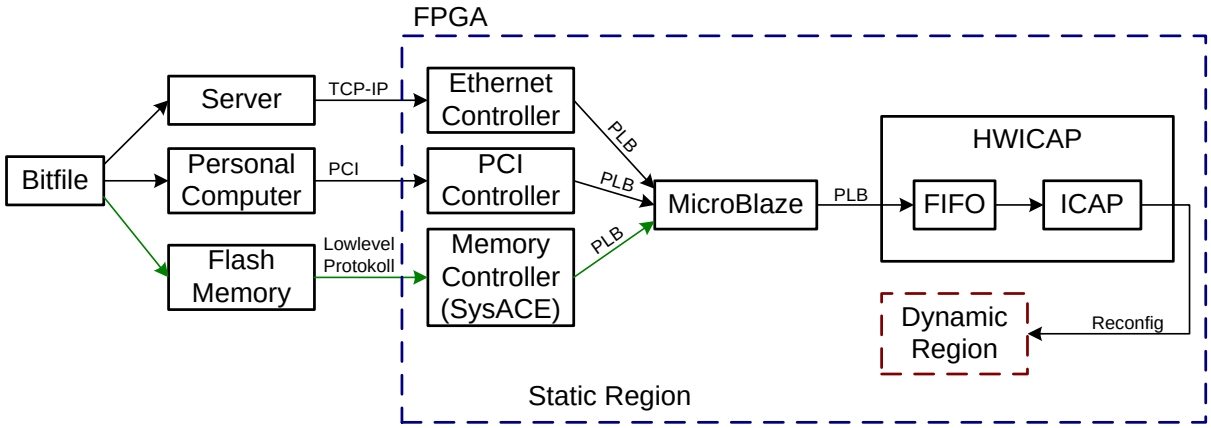


Bild 7-3: Übertragungsweg der Bitfile an den MicroBlaze und den ICAP-Treiber; In dieser Arbeit wurde der Weg über den Flash-Memory und den SysACE Memory-Controller realisiert

In der SoC-Plattform liest der MicroBlaze die Bitfiles von der Speicherkarte und überträgt diese über den PLB-Bus an den HWICAP. Zu diesem Zweck wurde folgende Funktion genutzt [28].

- *int XHwIcap_CF2Icap(XHwIcap *hwicap, char *filename)*
  Die Funktion liest die Bitfile mit dem Namen *filename* von der CompactFlash Speicherkarte und überträgt diese an die *hwicap* Instanz (vgl. Anhang D). Der Rückgabe-Wert ist ‚0' oder ein Fehlercode.

Die Dauer der Rekonfiguration der PRMs aus dieser Arbeit wurde mit einem Oszilloskop gemessen, indem während der Rekonfiguration ein Pin des FPGA auf einen High-Pegel gesetzt wurde und nach erfolgreicher Rekonfiguration wieder auf einen Low-Pegel gesetzt wurde (vgl. Bild 7-4). Die partiellen Bitfiles haben eine Größe von 104 kB, die von der Speicherkarte zum HWICAP übertragen werden.

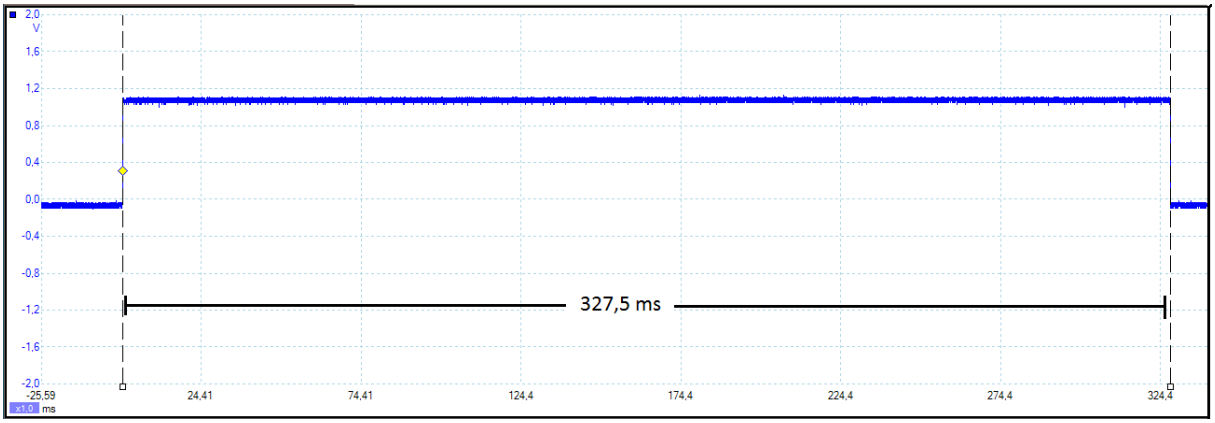


Bild 7-4: Messung der Rekonfigurations-Dauer einer PRM, bei Übertragung der partiellen Bitfile von der Speicherkarte an den ICAP

8 Systemerprobung der SoC-Bildverarbeitungs-Plattform

Die SoC-Bildverarbeitungs-Plattform ist für den speziellen Einsatz in autonomen Fahrzeugen zur Fahrspurerkennung entwickelt. Die modellierten Filter werden genutzt, um die Fahrspur deutlich hervorzuheben und dadurch die Nachverarbeitung zu vereinfachen [6]. Das gesamte System ist für den Echtzeitbetrieb ausgelegt, das heißt die Verarbeitung der Bilder geschieht in derselben Geschwindigkeit, in der sie eintreffen, wobei die maximale Latenz ein Bild beträgt (vgl. 4.3). In der zweistufigen Bildverarbeitungs-Pipeline der SoC-Plattform werden folgende Kombinationen der Bildverarbeitungsfilter eingesetzt.

- **Median – Binarisierung**
  Das Median-Filter reduziert das Rauschen in der Aufnahme und das Binarisierungs-Filter hebt die Fahrspurlinien hervor (vgl. Bild 8-1). Die Hervorhebung funktioniert immer dann, wenn die Lichtintensität der Fahrbahnmarkierung höher ist, als die der Umgebung, das heißt die Fahrbahnmarkierungen müssen heller sein als die Straße.

Bild 8-1: Anwendung des Medianfilter und Binarisierung mit einem Schwellwert von 75%

- **Binarisierung – Erosion**
  Das Binarisierungs-Filter hebt die Fahrbahnmarkierung hervor und das Erosions-Filter reduziert die Breite der Markierungen (vgl. Bild 8-2), um dünnere Linien zu erhalten, aus denen man die Richtung der Fahrspur extrahiert. Dicke Linien erlauben eine hohe Toleranz bezüglich des Richtungswinkels der Fahrspur [6].

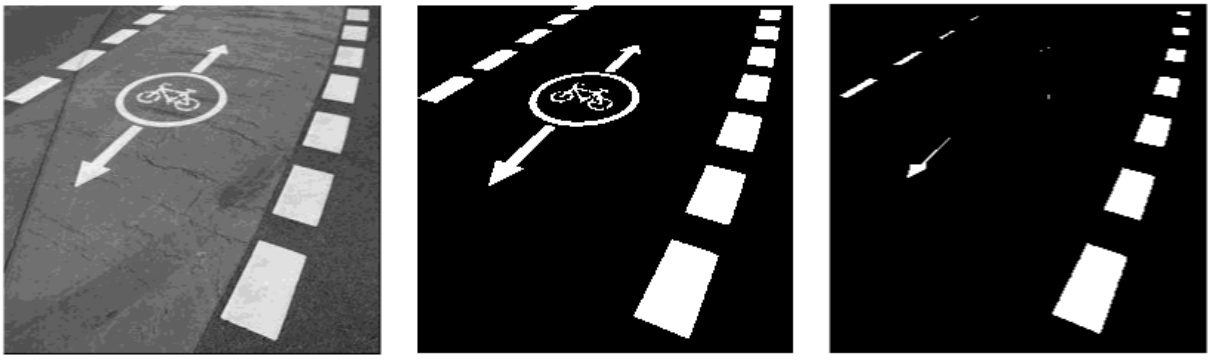


Bild 8-2: Anwendung der Binarisierung mit einem Schwellwert von 75% und des Erosionsfilter

- **Median – Sobel**
  Das Median-Filter reduziert das Rauschen in der Aufnahme und das Sobel-Filter markiert die Kanten der Fahrbahnmarkierung, indem diese als helle Linien dargestellt werden (vgl. Bild 8-3), dabei ist der Graustufenwert der Fahrbahnmarkierung nicht von Bedeutung. Aus diesen Linien lässt sich die Richtung der Fahrspur extrahieren [6].

Bild 8-3: Anwendung des Medianfilter und des Sobelfilter

- **Sobel – Binarisierung**
  Das Sobel-Filter markiert die Kanten der Fahrbahn und die anschließende Binarisierung hebt die Kanten nochmal hervor, indem sie weiß dargestellt werden (vgl. Bild 8-4). Über den Schwellwert des Binarisierungs-Filter lässt sich einstellen, wie stark der graustufen Unterschied der Kanten sein müssen, um dargestellt zu werden.

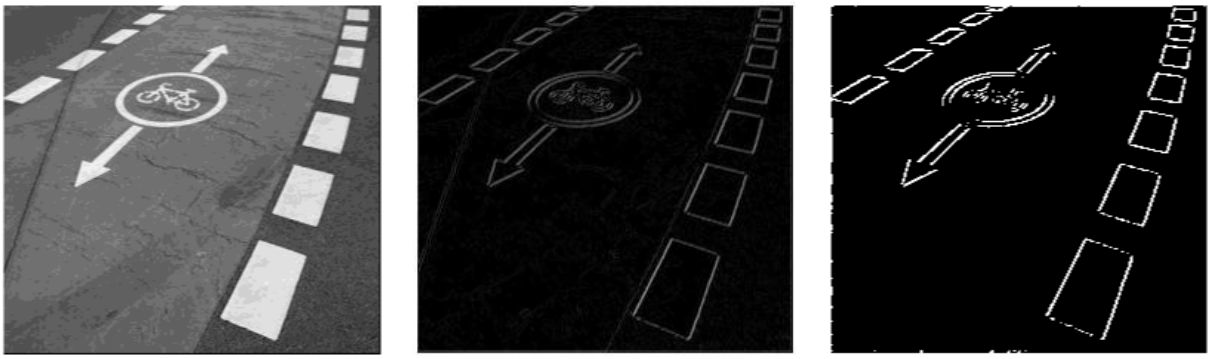


Bild 8-4: Anwendung des Sobelfilter und der Binarisierung mit einem Schwellwert von 20%

9 Zusammenfassung

Im Rahmen dieser Arbeit wurde eine SoC-basierte Bildverarbeitungs-Plattform mit partiell austauschbaren Bildverarbeitungsfiltern entwickelt, die in einer zweistufigen Pipeline zur Kantendetektion und so zur Erkennung der Fahrbahnmarkierung eingesetzt werden.

Es wurden die Hardwarekomponenten der SoC-Plattform vorgestellt, bestehend aus der Videology 24B752x Kamera als Bildquelle, dem Virtex-5 FPGA Board für die Bildverarbeitung und dem LCD-Bildschirm für die Darstellung der Ergebnisse. Die Kopplung der Komponenten aufgrund ihrer Spezifikationen wurde geprüft.

Die Anwendungsgebiete der SoC-Bildverarbeitung und dessen Entwurfstechnik in Verbindung mit der partiellen Rekonfiguration wurden vorgestellt, dabei wurden die Konzepte der partiellen Rekonfiguration und deren Bedeutung für die Entwicklung von Embedded Systemen erläutert.

Die VGA- und DVI-Signale wurden erläutert und eine parametrierbare VGA-Input Komponente wurde entwickelt, um die VGA-Signale zu interpretieren. Zur Kopplung der unterschiedlichen Pixelfrequenzen der VGA- und DVI-Signale wurde ein Bild-Zwischenspeicher im Block-RAM realisiert. Der AD9980 Video Capture und der Chrontel CH7301C DVI Transmitter wurden über den I²C-Bus parametriert.

Mit dem System-Generator wurden vier Bildverarbeitungsfilter Median, Binarisierung, Erosion und Sobel modelliert und synthetisiert und deren Wirkung, sowie der Ressourcenverbrauch wurden gezeigt. Die Integration der Bildverarbeitungsfilter als partiell rekonfigurierbare Module in die SoC-Architektur wurde mit PlanAhead durchgeführt.

Der Entwurfsablauf für die SoC-basierte partielle Rekonfiguration und die partielle Bitfile-Generierung mit PlanAhead, sowie die Nutzung des ICAP-Treibers für die dynamische Rekonfiguration wurde erläutert.

Die Bildverarbeitungs-Komponenten wurden als IP-Core in die SoC-Architektur mit dem MicroBlaze Softcore-Prozessor integriert, wobei die partiellen Bitfiles von der Speicherkarte über dem MicroBlaze an den ICAP-Treiber zur Verfügung gestellt wurden. Die Funktionalität der SoC-Architektur und der Austausch der Bildverarbeitungsmodule wurden durch die Darstellung auf einem LCD-Bildschirm verifiziert.

Literaturverzeichnis

1. **ZVEI - Zentralverband Elektrotechnik und Elektronikindustrie e.V.** *Nationale Roadmap Embedded Systems.* Frankfurt : s.n., 2009.

2. **Ebert, Christof und Salecker, Jürgen.** *Embedded Software Technologies and Trends.* s.l. : IEEE Computer Society, 2009.

3. **Xilinx.** Partial Reconfiguration of Virtex FPGAs in ISE 12. [Online] 23. Juli 2010. [Zitat vom: 26. April 2011.] http://www.xilinx.com/support/documentation/white_papers/wp374_Partial_Reconfig_Virtex_FPGAs.pdf.

4. **Mignolet, Jean-Yves und Wuyts, Roel.** *Embedded Multiprocessor Systems-on-Chip Programming.* s.l. : IEEE Computer Society, 2009.

5. **Mellert, Dennis.** *Modellierung einer Video-basierten Fahrspurerkennung mit dem Xilinx System Generator für eine SoC-Plattform.* [Bachelorarbeit] HAW Hamburg : s.n., 2010.

6. **Schneider, Christian.** *Ein System-on-Chip-basiertes Fahrspurführungssystem.* [Bachelorarbeit] HAW Hamburg : s.n., 2011.

7. **Xilinx.** Partial Reconfiguration User Guide. [Online] 03. Mai 2010. [Zitat vom: 26. April 2011.] http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/ug702.pdf.

8. **TinyVGA.** VGA Signal Timing. [Online] [Zitat vom: 26. April 2011.] http://tinyvga.com/vga-timing.

9. **HP.** Technical Specifications - HP Compaq LA2405wg 24-inch Widescreen LCD Monitor. [Online] 01. Oktober 2009. [Zitat vom: 26. April 2011.] http://h20195.www2.hp.com/v2/GetPDF.aspx/c01920535.pdf.

10. **Xilinx.** ML507 User Guide. [Online] 7. Oktober 2009. [Zitat vom: 26. April 2011.] http://www.xilinx.com/support/documentation/boards_and_kits/ug349.pdf.

11. —. Virtex-5 Family Overview. [Online] 06. Februar 2009. [Zitat vom: 26. April 2011.] http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.

12. **Chrontel.** CH7301C DVI Transmitter Device. [Online] 17. März 2010. [Zitat vom: 26. April 2011.] http://www.chrontel.com/pdf/7301ds.pdf.

13. **Analog Devices.** AD9980 Video Capture Device. [Online] 2005. [Zitat vom: 26. April 2011.] http://www.analog.com/static/imported-files/data_sheets/AD9980.pdf.

14. **Jeyrani-Mameghani, Ramin.** *SoC-basierte Erprobungsplattform für CCD-Kameras mit Channel Link Interface.* [Diplomarbeit] HAW Hamburg : s.n., 2009.

15. **Xilinx.** Virtex-5 FPGA User Guide. [Online] 17. Mai 2010. [Zitat vom: 26. April 2011.] http://www.xilinx.com/support/documentation/user_guides/ug190.pdf.

16. —. Virtex-5 FPGA Data Sheet. [Online] 5. Mai 2010. [Zitat vom: 26. April 2011.] http://www.xilinx.com/support/documentation/data_sheets/ds202.pdf.

17. **Wikipedia.** CCD-Sensor. [Online] [Zitat vom: 26. April 2011.] http://de.wikipedia.org/wiki/CCD-Sensor.

18. —. Demosaicing. [Online] [Zitat vom: 28. Juli 2011.] http://de.wikipedia.org/wiki/Demosaicing.

19. **Videology Imaging Solutions.** *Application Note 24B752x Preliminary.* [Datasheet] 5405 Netherlands : s.n., 2007.

20. **1stVision.** Micron MT9V022 CMOS Digital Image Sensor. [Online] 2006. [Zitat vom: 27. April 2011.] http://www.1stvision.com/cameras/sensor_specs/Micron_MT9V022.pdf.

21. **Jähne, Bernd.** *Digitale Bildverarbeitung.* Heidelberg : Springer-Verlag, 2005. ISBN 3-540-24999-0.

22. **Xilinx.** LogiCORE IP XPS HWICAP - Product Specification. [Online] 23. Juli 2010. [Zitat vom: 26. April 2011.] http://www.xilinx.com/support/documentation/ip_documentation/xps_hwicap.pdf.

23. —. PlanAhead User Guide. [Online] 21. Dezember 2010. [Zitat vom: 26. April 2011.] http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/PlanAhead_UserGuide.pdf.

24. **Hwang, Enoch.** Build a VGA Monitor Controller. [Online] 17. November 2004. [Zitat vom: 26. April 2011.] http://faculty.lasierra.edu/~ehwang/public/mypublications/VGA%20Monitor%20Controller.pdf.

25. **Digital Display Working Group.** Digital Visual Interface - DVI. [Online] 02. April 1999. [Zitat vom: 24. April 2011.] http://www.ddwg.org/lib/dvi_10.pdf.

26. **NXP.** I²C-bus specification and user manual. [Online] 19. Juni 2007. [Zitat vom: 02. Mai 2011.] http://www.nxp.com/documents/user_manual/UM10204.pdf.

27. **Hütter, Wolfgang.** *Implementierung von Bildverarbeitungsalgorithmen mit VHDL für einen FPGA-Coprozessor.* [Diplomarbeit] HAW Hamburg : s.n., 1999.

28. **Xilinx.** PlanAhead Software Tutorial - Partial Reconfiguration of a Processor Peripheral. [Online] 21. September 2010. [Zitat vom: 26. April 2011.] http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/PlanAhead_Tutorial_Reconfigurable_Processor_Peripheral.pdf.

29. —. Partial Reconfiguration Flow - Presentation Manual. [Online] 2010. [Zitat vom: 26. April 2011.] http://insercion-deteccion-errores-sistemasfpga.googlecode.com/svn/trunk/Bibliografia/PR_lab_manual.pdf.

Glossar

Adaptiver Schwellwert	Ein an die Umgebung angepasster Schwellwert.
Analog-Digital-Umsetzer (ADU)	Elektronisches Bauteil zum digitalisieren analoger Eingangsdaten.
Bildstrom	Fortlaufende Übertragung von Einzelbildern.
Bildwiederholrate	Anzahl der Einzelbilder pro Sekunde.
Binärbild	Digitale Rastergrafik, deren Pixel nur die zwei Farben Schwarz und Weiß annehmen.
Bitfile	Datei mit Konfigurationsdaten für ein FPGA
Blackbox	Geschlossenes System unter Vernachlässigung des inneren Aufbaus.
Block-RAM	Spezielles Speichermodul im FPGA.
Channel-Link	Serielle Hochgeschiwindigkeits-Datenübertragung über LVDS.
Charge-Coupled Device (CCD) Sensor	Lichtempfindliche elektronische Bauelemente, die vor allem in Kameras verbaut werden.
Complementary Metal Oxide Semiconductor (CMOS)	Bezeichnung für Halbleiterbauelemente zur Realisierung von digitalen und analogen Schaltungen.
Configurable Logik Block (CLB)	Bauteile im FPGA, deren internen Elemente programmiert werden.
Demosaicing	Rekonstruktion einer farbigen Rastergrafik aus den unvollständigen Farbwerten eines mit Mosaik-Farbfiltern überlagerten Bildsensors.
Deserialisieren	Umwandlung eines Bytestroms in ein Objekt.
Digital Clock Manager (DCM)	Bauteil im FPGA zur Manipulation von Taktsignalen.
Digital Visual Interface (DVI)	Elektrische Schnittstelle zur Übertragung von Videodaten.
Dynamische Region	Bereich im FPGA, der während des Betriebes mit PRM konfiguriert wird.
Echtzeit	Vorhersehbare Dauer einer Aktion, die gestartet, ausgeführt und beendet wird.
Elektronic Design Interchange Format (EDIF)	Elektronisches Format zum austauschen von Netzlisten und Schaltplänen.
Embedded Systeme	Elektronischer Rechner, der in einem technischen Kontext eingebunden ist.
Executable and Linkable Format (ELF)	Dateiformat für ausführbaren Programm-Code.
Field Programmable Gate Array (FPGA)	Integrierter Schaltkreis der Digitaltechnik, in dem eine logische Schaltung programmiert wird.
First-In First-Out (FIFO)	Pufferspeicher, wobei das erste eingefügte Element auch als

	erstes herausgeholt wird.
Fixed-Point-Arithmetic	Mathematische Berechnungen in der Festkomma-Zahlendarstellung.
Graustufen-Bild	Bild, das nur aus den Daten für Schwarz, Weiß und den Grauabstufungen dazwischen besteht.
Input Serializer/Deserializer (ISERDES)	Seriell zu parallel Konverter im FPGA
Input/Output Block (IOB)	Sammlung von Elementen zur Implementierung der Ein- und Ausgänge des FPGA.
Intellectual Property Core (IP-Core)	Bezeichnung für wiederverwendbare Teile eines Chipdesigns in Form von Code oder Netzliste.
Inter-Integrated Circuit (I^2C)	Serieller Datenbus zur Kommunikation zwischen verschiedenen digitalen Schaltungen, über zwei bidirektionale Leitungen.
Internal Configuration Access Port (ICAP)	Interne Konfigurationsschnittstelle des FPGA genutzt zur partiellen Rekonfiguration.
Liquid Crystal Display (LCD)	Bildschirm aus Flüssigkristallen zur Anzeige eines Bildstrom.
Low Voltage Differential Signaling (LVDS)	Schnittstellen-Standard für Hochgeschwindigkeits-Datenübertragung über differentielle Spannungspegel.
Matlab / Simulink	Software zur Modellering von Systemen und ausführen mathematischer Berechnungen.
MicroBlaze	32 Bit RISC Mikrocontroller im FPGA.
Multi-Core-Architektur	Bezeichnung eines Systems mit mehr als einem Hauptprozessor.
NGC-Netzliste	Netzliste bestehend aus logischen Schaltungen und Design-Einschränkungen.
Output Serializer/Deserializer (OSERDES)	Parallel zu seriell Konverter im FPGA.
Partiell rekonfigurierbare Region (PRR)	Bereich im FPGA, der während des Betriebes mit PRM konfiguriert wird.
Partiell rekonfigurierbares Modul (PRM)	RTL-Modell, das die partielle Rekonfiguration unterstützt.
Partielle Rekonfiguration (PR)	Xilinx Technologie zur Rekonfiguration eines FPGA.
Pipeline	Parallele Abarbeitung von Befehlen durch Zerlegung dieser in Teilaufgaben, um einen hohen Durchsatz zu erhalten.
Pixelfrequenz	Anzahl der Pixel pro Sekunde.
Pixelstrom	Fortlaufende Übertragung von einzelnen Pixel.
Processor Local Bus (PLB)	Datenbus in SoCs zur Kommunikation zwischen Prozessorkernen und Bus-Modulen.

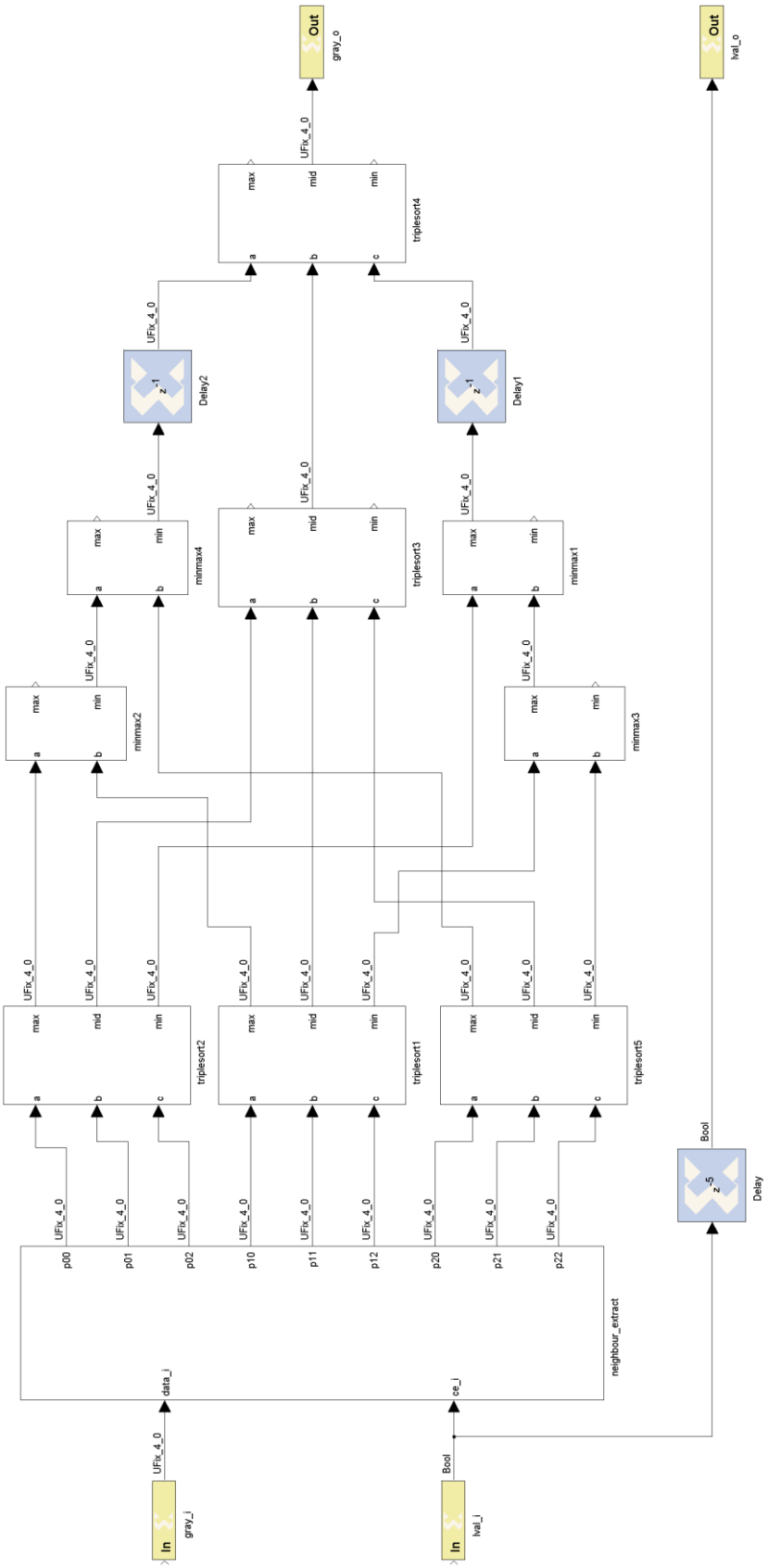
Reduced Instruction Set Computer (RISC)	Rechner mit reduziertem Befehlssatz, wodurch eine schnelle Taktung realisierbar ist.
Register Transfer Level (RTL)	Abstraktionsebene in der Hardware-Modellierung von digitalen Schaltkreisen.
Ressourcen-Sharing	Gemeinsame Nutzung von Betriebsmittel.
RGB-Tripel	Ein zusammengesetzter Wert, bestehend aus den Werten für Rot, Grün und Blau
Routing	Festlegen von Wegen für Datenströme.
RS232	Standard für eine serielle Schnittstelle.
Serial Peripheral Interface (SPI)	Serieller Datenbus zur Kommunikation zwischen verschiedenen digitalen Schaltungen, über drei Leitungen.
Serialisieren	Umwandlung eines Objekts in einen Bytestrom.
Statische Region	Bereich im FPGA, der nicht rekonfiguriert wird.
Synthese	Generierung von elektronischen Schaltungen aus VHDL-Dateien.
System Advanced Configuration Environment (SysACE)	IP-Core für den Zugriff auf Daten in einer Speicherkarte.
System-on-Chip	Integration aller oder eines großen Teils der Funktionen eines Systems auf einem Chip.
Very High Speed Integrated Circuit Hardware Description Language (VHDL)	Textbasierte Hardwarebeschreibungssprache um digitale Systeme zu beschreiben.
Video Electronics Standards Association (VESA)	Organisation um einheitliche Spezifikationen für Videostandards zu erstellen.
Video Graphics Array (VGA)	Videostandard zur Übertragung eines Bildstroms.
Xilinx Bash-Shell	Software Anwendung zur Verarbeitung von Kommando-Zeilen Befehle
Xilinx Core-Generator	Designwerkzeug zur Generierung von HW-Modulen für FPGAs.
Xilinx Embedded Development Kit (EDK)	Designwerkzeug zur Erstellung von SoCs.
Xilinx PlanAhead	Design- und Analysewerkzeug für FPGAs.
Xilinx Software Development Kit (SDK)	Werkzeug zur Erstellung von Software für SoCs.
Xilinx System-Generator	Modellbasiertes Entwicklungstool zur Erstellung von RTL-Modellen.

Abbildungsverzeichnis

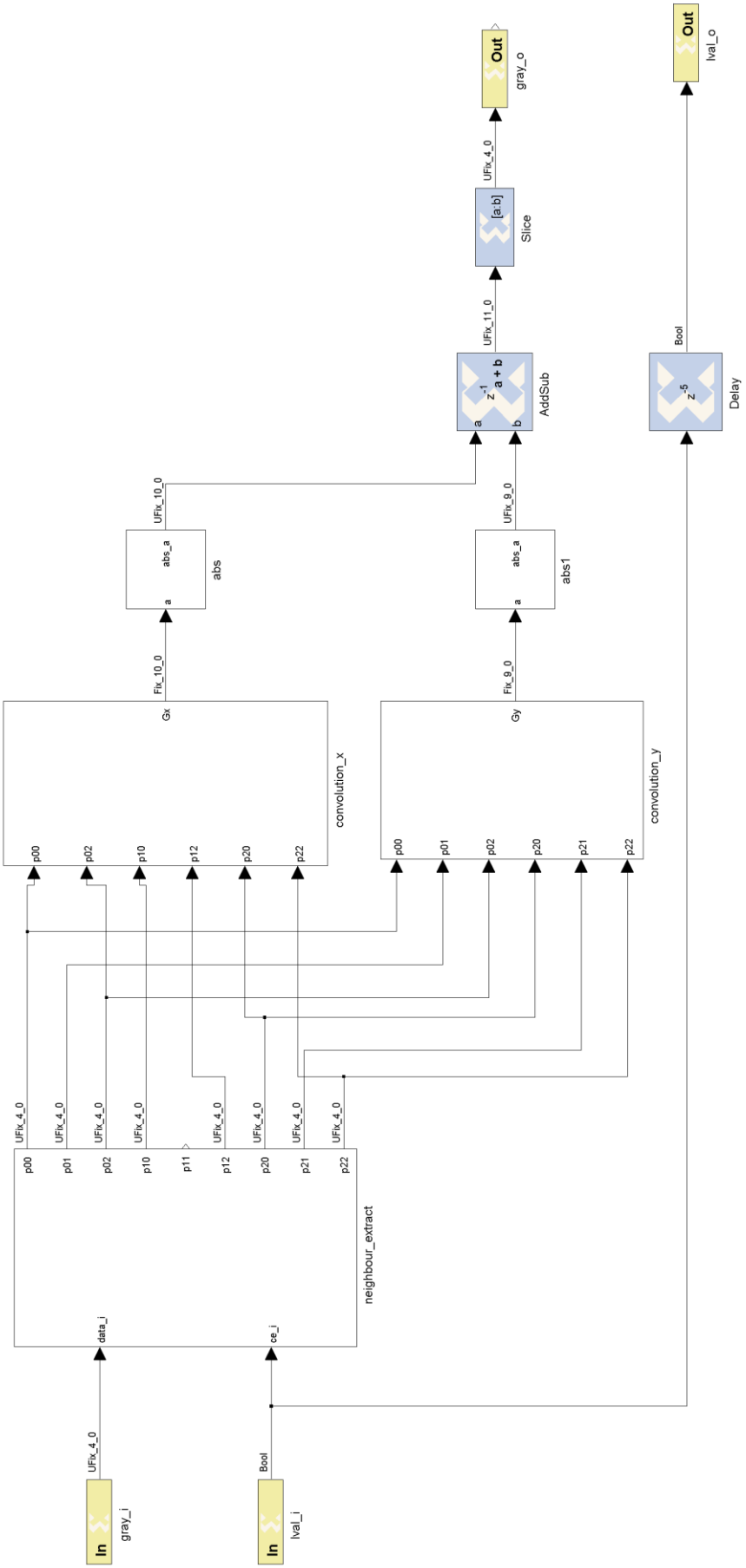
Tabellenverzeichnis

Anhang

A. System-Generator-Modellierung des Median-Filter



B. System-Generator-Modellierung des Sobel-Filter



C. C-Code für den I²C-Bus Zugriff

```
void video_iic_send(u8 slv_addr, u8 reg, u8 byte)
{
	u8 buffer[2];
	buffer[0] = reg;
	buffer[1] = byte;
	XIic_Send(XPAR_XPS_IIC_VIDEO_BASEADDR, slv_addr, buffer, 2, XIIC_STOP);
	usleep(10000);
}


u8 video_iic_recv(u8 slv_addr, u8 reg)
{
	u8 byte = 0x00;
	XIic_Send(XPAR_XPS_IIC_VIDEO_BASEADDR, slv_addr, &reg, 1, XIIC_STOP);
	XIic_Recv(XPAR_XPS_IIC_VIDEO_BASEADDR, slv_addr, &byte, 1, XIIC_STOP);
	usleep(10000);
	return byte;
}
```

D. C-Code für Bitfile-Transfer von der Speicherkarte zum HWICAP

```
/* Parse Bitfile header */
XHwIcap_Bit_Header XHwIcap_ReadHeader(Xuint8 *Data, Xuint32 Size)
{
    Xuint32 I;
    Xuint32 Len;
    Xuint32 Tmp;
    XHwIcap_Bit_Header Header;
    Xuint32 Index;

    /* Start Index at start of bitstream */
    Index=0;

    /* Initialize HeaderLength.  If header returned early inidicates
     * failure.
     */
    Header.HeaderLength = XHI_BIT_HEADER_FAILURE;
    Header.DesignName = NULL;
    Header.Date = NULL;
    Header.PartName = NULL;
    Header.BitstreamLength = 0;
    Header.Time = NULL;

    /* Get "Magic" length */
    Header.MagicLength = Data[Index++];
    Header.MagicLength = (Header.MagicLength << 8) | Data[Index++];

    /* Read in "magic" */
    for (I=0; I<Header.MagicLength-1; I++) {
        Tmp = Data[Index++];
        if (I%2==0 && Tmp != XHI_EVEN_MAGIC_BYTE)
        {
          return Header;   /* INVALID_FILE_HEADER_ERROR */
        }
        if (I%2==1 && Tmp != XHI_ODD_MAGIC_BYTE)
        {
            return Header;   /* INVALID_FILE_HEADER_ERROR */
        }
    }

    /* Read null end of magic data. */
    Tmp = Data[Index++];

    /* Read 0x01 (short) */
    Tmp = Data[Index++];
    Tmp = (Tmp << 8) | Data[Index++];
```

```
    /* Check the "0x01" half word */
    if (Tmp != 0x01)
    {
       return Header;   /* INVALID_FILE_HEADER_ERROR */
    }


    /* Read 'a' */
    Tmp = Data[Index++];
    if (Tmp != 'a')
    {
        return Header;    /* INVALID_FILE_HEADER_ERROR  */
    }

    /* Get Design Name length */
    Len = Data[Index++];
    Len = (Len << 8) | Data[Index++];

    /* allocate space for design name and final null character. */
    Header.DesignName = (Xuint8 *)malloc(Len);

    /* Read in Design Name */
    for (I=0; I<Len; I++)
    {
        Header.DesignName[I] = Data[Index++];
    }

    /* Read 'b' */
    Tmp = Data[Index++];
    if (Tmp != 'b')
    {
        return Header;  /* INVALID_FILE_HEADER_ERROR */
    }

    /* Get Part Name length */
    Len = Data[Index++];
    Len = (Len << 8) | Data[Index++];

    /* allocate space for part name and final null character. */
    Header.PartName = (Xuint8 *)malloc(Len);

    /* Read in part name */
    for (I=0; I<Len; I++)
    {
        Header.PartName[I] = Data[Index++];
    }

    /* Read 'c' */
    Tmp = Data[Index++];
    if (Tmp != 'c')
    {
        return Header;  /* INVALID_FILE_HEADER_ERROR */
    }

    /* Get date length */
    Len = Data[Index++];
    Len = (Len << 8) | Data[Index++];

    /* allocate space for date and final null character. */
    Header.Date = (Xuint8 *)malloc(Len);

    /* Read in date name */
    for (I=0; I<Len; I++)
    {
        Header.Date[I] = Data[Index++];
    }

    /* Read 'd' */
    Tmp = Data[Index++];
    if (Tmp != 'd')
    {
        return Header;  /* INVALID_FILE_HEADER_ERROR  */
    }

    /* Get time length */
```

```
    Len = Data[Index++];
    Len = (Len << 8) | Data[Index++];

    /* allocate space for time and final null character. */
    Header.Time = (Xuint8 *)malloc(Len);

    /* Read in time name */
    for (I=0; I<Len; I++)
    {
        Header.Time[I] = Data[Index++];
    }

    /* Read 'e' */
    Tmp = Data[Index++];
    if (Tmp != 'e')
    {
        return Header;  /* INVALID_FILE_HEADER_ERROR */
    }

    /* Get byte length of bitstream */
    Header.BitstreamLength = Data[Index++];
    Header.BitstreamLength = (Header.BitstreamLength << 8) | Data[Index++];
    Header.BitstreamLength = (Header.BitstreamLength << 8) | Data[Index++];
    Header.BitstreamLength = (Header.BitstreamLength << 8) | Data[Index++];

    Header.HeaderLength = Index;

    return Header;

}

//====================================================

/** Loads a partial bitstream from CF into ICAP */
int XHwIcap_CF2Icap(XHwIcap *hwicap, char* filename)
{
         int i, numCharsRead, ace_buf_count, rc, SectorNumber;
         SYSACE_FILE *stream;
    XHwIcap_Bit_Header bit_header;
    Xuint8 data3,data2,data1,data0;
    Xuint8 systemACE_Buffer[XSA_CF_SECTOR_SIZE];
    Xuint32 word[2];
    XStatus Status;

#if DEBUG
         print("In CF2ICAP\r\n");
         xil_printf("filename = %s\r\n",filename);
#endif

    /* Opening file */
    if ((stream = sysace_fopen(filename, "r")) == NULL) {
        xil_printf("Can't open file (%s)\r\n", filename);
        return -1;
    }

#if DEBUG
         print("In CF2ICAP ..File Opened..\r\n");
#endif

    /* Read from systemAce */
         numCharsRead = sysace_fread(systemACE_Buffer, 1, XSA_CF_SECTOR_SIZE,
                                     stream);
    if (numCharsRead <= 0) {
        xil_printf("Error reading from system ace (%d)\r\n", numCharsRead);
        return -1;
    }

#if DEBUG
         print("In CF2ICAP ..Header Sector Read..\r\n");
#endif

    /* Read the bitstream header */
    bit_header = XHwIcap_ReadHeader(systemACE_Buffer,0);

        /* close systemAce file handle */
        rc = sysace_fclose (stream);
```

```
        if (rc < 0) {
                /* Can't close */
                xil_printf("can't close file\r\n");
                return -1;
    }

#if DEBUG
         print("In CF2ICAP ..File Closed..\r\n");
#endif

        /* Now that we have info about the bitstream,
         * re-open and skip the header.
         */

    if ((stream = sysace_fopen(filename, "r")) == NULL) {
        /* Can't open file */
        xil_printf("can't open file\r\n");
        return -1;
    }

#if DEBUG
         print("In CF2ICAP ..File Opened..\r\n");
#endif

         /* Read the header (effectively skipping it) */
         numCharsRead = sysace_fread(systemACE_Buffer, 1, bit_header.HeaderLength,
                                stream);
#if DEBUG
         xil_printf("Number of chars read = %d, Sector Number = %d\r\n",numCharsRead,SectorNumber);
         print("In CF2ICAP ..Skip Header..\r\n");
#endif

         numCharsRead = sysace_fread(systemACE_Buffer, 1, XSA_CF_SECTOR_SIZE,
                                stream);

         SectorNumber = 1;

    /* Loop through all bitstream data and write to ICAP */
    ace_buf_count = 0;
    for (i=0; i<bit_header.BitstreamLength; i+=4)
         {

                  /* Convert 4 chars into an integer */
        data3 = systemACE_Buffer[ace_buf_count++];
        data2 = systemACE_Buffer[ace_buf_count++];
        data1 = systemACE_Buffer[ace_buf_count++];
        data0 = systemACE_Buffer[ace_buf_count++];
        word[0] = ((data3 << 24) | (data2 << 16) | (data1 << 8) | (data0));
        i+=4;
                if(i<bit_header.BitstreamLength) // store another word if even number of words
                {
                          /* Convert 4 chars into an integer */
                data3 = systemACE_Buffer[ace_buf_count++];
                data2 = systemACE_Buffer[ace_buf_count++];
                data1 = systemACE_Buffer[ace_buf_count++];
                data0 = systemACE_Buffer[ace_buf_count++];
                word[1] = ((data3 << 24) | (data2 << 16) | (data1 << 8) | (data0));
                }
                else
                {
                        word[1] = 0x0; // store 0- this is work around for hwicap bug in 11.4
                }
            Status = XHwIcap_DeviceWrite(hwicap, word, 2);
        if (Status != XST_SUCCESS)
        {
                                /* Error writing to ICAP */
                           xil_printf("error writing to ICAP (%d)\r\n", Status);
           return -1;
        }
#if DEBUG
                xil_printf("In CF2ICAP ..Writing Word Number %d from current sector to
ICAP..\r\n",ace_buf_count);
#endif

                  /* Check to see if we need to read from CF again */
        if (ace_buf_count == XSA_CF_SECTOR_SIZE)
```

```
        {
#if DEBUG
                                print("In CF2ICAP ..Reading Next Sector..\r\n");
#endif
            /* read next sector from CF */
            numCharsRead = sysace_fread(systemACE_Buffer, 1, XSA_CF_SECTOR_SIZE,
                                        stream);
            ace_buf_count = 0;
#if DEBUG
                                xil_printf("Number of chars read = %d, sector number =
%d\r\n",numCharsRead, SectorNumber);
#endif
                                SectorNumber++;
        }
                }

#if DEBUG
        print("In CF2ICAP ..All Words Written to ICAP..\r\n");
#endif
        /* close systemAce file handle */
        rc = sysace_fclose (stream);
        if (rc < 0) {
                /* Can't close */
                xil_printf("can't close file\r\n");
                return -1;
    }
        return 0;
}
```

E. VHDL-Code für die Instanziierung der Bildverarbeitungs-PRM

```
-- PR Module Interface as in .ngc
  component image_filter_cw
       port (
                    clk : in std_logic;
                    ce : in std_logic;
                    gray_i : in std_logic_vector(3 downto 0);
                    gray_o : out std_logic_vector(3 downto 0);
                    lval_i : in std_logic;
                    lval_o : out std_logic
                    );
  end component image_filter_cw;

----------------------------
-- Image filter
----------------------------

  FIRST_FILTER_INST: image_filter_cw
  port map( clk => pixel_clk,
        ce => '1',
        gray_i => gray_16,
        gray_o => first_fil_gray,
        lval_i => dvi_lval_d,
        lval_o => first_fil_lval);

  SECOND_FILTER_INST: image_filter_cw
  port map( clk => pixel_clk,
        ce => '1',
        gray_i => first_fil_gray,
        gray_o => second_fil_gray,
        lval_i => first_fil_lval,
        lval_o => second_fil_lval);
```

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 12. August 2011 _____

Ort, Datum Unterschrift