Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Masterarbeit

**Erik Andresen**

## ARM-basierter MPSoC unter Linux für eine Fahrspurführung mit Bildverarbeitung

*Fakultät Technik und Informatik*
*Studiendepartment Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

Erik Andresen

# ARM-basierter MPSoC unter Linux für eine Fahrspurführung mit Bildverarbeitung

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science 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. Wolfgang Fohl

Eingereicht am: 20. Juni 2013

**Erik Andresen**

**Thema der Arbeit**
ARM-basierter MPSoC unter Linux für eine Fahrspurführung mit Bildverarbeitung

**Stichworte**
MPSoC, SoC, Linux, Zynq-7000, ARM Cortex-A9, FPGA, AXI, VDMA, UIO, GStreamer, Bildverarbeitung, Spurführung, FAUST

**Kurzzusammenfassung**
Diese Masterarbeit beschreibt die Portierung eines SoC basierten Spurführungssystems auf den kombinierten FPGA-MPSoC Zynq-7000 mit zwei Dual-Core ARM Cortex-A9 Prozessoren unter Verwendung eines Linux Betriebssystems. Die Fahrspur wird mit einer CMOS-Kamera aufgenommen und durch eine Bildverarbeitung im FPGA mit einer Geraden approximiert, die als Eingangsgröße für die Berechnung der Spurführung in Software verwendet wird. Die FPGA Komponenten werden durch das UIO-Subsystem vom Linux-Kernel konfiguriert und ausgelesen. Mit einem VDMA IP-Core werden die Stream basierten Videodaten zur Visualisierung mit dem Framework GStreamer in den Speicher der Prozessoren überführt.

**Title of the paper**
ARM-based MPSoC on Linux for a path following system with image processing

**Keywords**
MPSoC, SoC, Linux, FAUST, Zynq-7000, ARM Cortex-A9, FPGA, AXI, VDMA, UIO, GStreamer, Image Processing, Path Following, FAUST

**Abstract**
This master thesis describes the porting of a SoC-based path following system on the combined FPGA MPSoC Zynq-7000 with two dual-core ARM Cortex-A9 processors using a Linux operating system. The lane is recorded by a CMOS camera and approximated by an image processing unit in the FPGA by a straight line, which is used as an input for the calculation in the path following software. The FPGA components are configured and read with the AXI bus and the UIO subsystem of the Linux kernel. For visualization with the GStreamer framework the stream based video data is transferred into the memory of the processors using the VDMA IP-Core.

# Inhaltsverzeichnis

# 1 Einleitung

In der Arbeitsgruppe „High Performance Embedded Computing“ (HPEC) des Department Informatik werden Algorithmen der Bildverarbeitung und der Regelungstechnik auf eingebetteten Systemen implementiert. Als Anwendung dient ein autonom fahrendes Fahrzeug (Abb. 1.1), das im Rahmen des Forschungsprojekts FAUST (Fahrerassistenz- und autonome Systeme) einer Fahrspur folgt und dabei Hindernissen ausweichen soll. Ausgeführt wird die Anwendung auf FPGAs, also rekonfigurierbaren Schaltkreisen die als System-On-Chip (SoC) Beschleuniger-Module und Mikroprozessoren auf einem Chip vereinen. Die Fahrspur wird mit einer Kamera aufgenommen und von der in der FPGA-Logik realisierten Bildverarbeitungspipeline identifiziert. Daraus wird mit dem Spurführungsalgorithmus die Stellgröße der Lenkung errechnet. Als Mikroprozessor kommen in der FPGA-Logik implementierte MicroBlaze IP-Cores zum Einsatz [48][34], die sich mit einer maximalen Taktfrequenz von 200 MHz betreiben lassen. Rechenintensive Algorithmen wie die Bildverarbeitung werden im FPGA mit Parallelverarbeitung direkt im Pixelstrom ausgeführt.

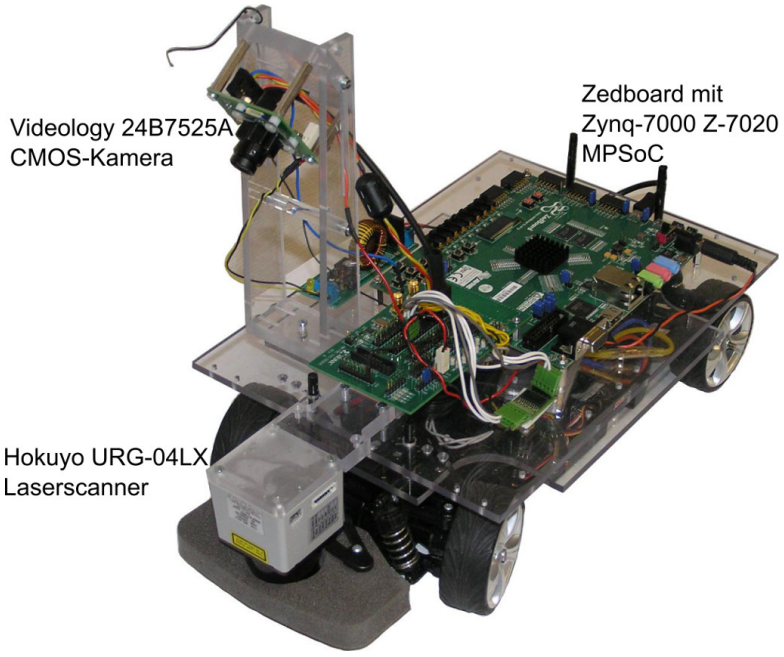



Abbildung 1.1: Das Multiprozessor System-On-Chip Autonomous Vehicle (SAV)

Viele eingebettete Systeme arbeiten unter Batterieversorgung, so daß ein geringer Strombedarf ein Unterscheidungsmerkmal zu anderen Systemen ist. Durch die niedrigere Taktfrequenz liefern Multiprozessorsysteme (MP) mehr Rechenleistung pro Watt als ein vergleichbares Einprozessorsystem. Wegen der geringeren Temperatur von Multiprozessorsystemen werden diese bei eingebetteten Systemen vermehrt eingesetzt [39][11]. Algorithmen mit hohem Rechenaufwand werden in eingebetteten Systemen in DSPs und ASICs realisiert, die zusammen mit dem Prozessor in ein System-on-Chip integriert sind [32][71]. Beispiele sind der Tablet-PC „Samsung Galaxy Tab 2“ und das Smartphone „Motorola Droid Bionic“, die beide als Multiprozessor-System-on-Chip (MPSoC), einen TI OMAP4430 System-On-Chip mit zwei Cortex-A9 ARM-Prozessoren einsetzen. Spielt der Benutzer eines solchen Gerätes ein z.B. mit H.264 komprimiertes Video ab, wird dieses Video von einem ASIC des MPSoC dekodiert.

Ziel der HPEC-Arbeitsgruppe ist das Durchdringen der MPSoC-Technik. Dazu wird die kombinierte FPGA-MPSoC-Plattform „Zynq-7000“ mit zwei bis zu 800MHz getakteten ARM Cortex-A9 Prozessoren eingesetzt, die SMP-Multiprozessorsysteme mit FPGA-Technik auf einem Chip vereint [61]. Gegenüber Entwürfen mit Prozessor IP-Cores sind die dedizierten ARM-Prozessoren schneller getaktet und nicht durch die Logikzellen des FPGAs begrenzt, wodurch mehr Algorithmen eines Produktes in Software implementiert werden können. Dies erlaubt die Wiederverwendung von Intellectual Property (IP), z.B. kann das komplette Hardware-Design für eine Produktreihe einmal festgelegt werden. Kundenspezifische Anforderungen werden in der Software implementiert. Zusätzlich werden die Entwicklungskosten durch die in der Softwareentwicklung kürzeren Entwicklungszeiten und bessere Wartbarkeit gesenkt [46].

Ergebnis dieser Masterarbeit ist die Portierung der bisherigen Architektur [28] des System-On-Chip Autonomous Vehicle (SAV) auf den eingebetteten Zynq-7000 MPSoC mit den folgenden Schwerpunkten:

- Durchführung des Hardware/Software-Codesigns. Entscheidungen mit einer Verzweigung (*if*-Bedingung), z.B. ob der Fahrspur gefolgt, oder ausgewichen werden soll, sind eine Aufgabe, für die die schneller getakteten ARM-Prozessoren besser geeignet sind.
- Partitionierung der Aufgaben zwischen den beiden Prozessoren. Während ein Prozessor die Spurführung aus dem Ergebnis der Bildverarbeitung berechnet, kontrolliert der andere Prozessor den Ablauf der Bildverarbeitung im FPGA und sendet Status- und Debug-Informationen an einen Entwicklungs-PC.

- Portierung der Bildverarbeitungspipeline und Entwicklung einer Prozessor-Anbindung zur Konfiguration und Ausgabe von Daten.
- Anpassung der Bildverarbeitungspipeline an die Videology 24B7525A CMOS-Kamera.
- Installation und Konfiguration des Linux Betriebssystems, sowie Entwicklung von Treibern zur Kommunikation des Betriebssystems mit der Bildverarbeitungspipeline und der Peripherie.
- Portierung der Datalogging-Schnittstellen auf Ethernet und Live-Übertragung der Kamera-Bilder über W-LAN.

## 1.1 Kapitelübersicht

2. **Kapitel**: Vorstellung der eingesetzten Techniken, wie der mit dem Xilinx System Generator entwickelten Bildverarbeitungspipeline zur Fahrspurerkennung. Im zweiten Teil wird ein Überblick über das in dieser Arbeit entwickelte System gegeben.
3. **Kapitel**: Übersicht des Zynq-7000 ARM Cortex-A9 MPSoC mit Peripherie wie I2C, Gigabit Ethernet und USB. Zur Anbindung der FPGA IP-Cores werden die Grundlagen des AXI-Bus erläutert. Der mehrstufige Bootloader-Vorgang von SD-Karte für die Verwendung eines Betriebssystem wird beschrieben.
4. **Kapitel**: Dokumentation der AXI IP-Cores für die Kommunikation der Prozessoren mit dem FPGA. Die Parametrisierung und das Lesen von Berechnungsergebnisse der Bildverarbeitungskette wird über Softwareregister, der Transport der Kamera-Bilder zu den Prozessoren wird mit dem AXI VDMA IP-Core vorgenommen.
5. **Kapitel**: Die Installation und Konfiguration des Linux-Betriebssystems, sowie Entwurf und Implementierung der Software werden Beschrieben. Dazu gehört die Memory Mapped Kommunikation mit den IP-Cores über das UIO-Subsystem, die Visualisierung der Fahrspurerkennung mit GStreamer und das Senden von Daten und Bildern über Ethernet.
6. **Kapitel**: Analysiert die Hard- und Software des entwickelten Systems. Dazu zählt eine Messung des Zeitverhalten, Überprüfung der Datenkonsistenz und Auswertung einer Testfahrt.
7. **Kapitel**: Gibt eine Zusammenfassung und Ausblick.

# 2 Übersicht und Konzepte

Dieses Kapitel gibt im ersten Teil einen Überblick über die eingesetzten Techniken, die aus Vorarbeiten übernommen und weiterentwickelt wurden. Im zweiten Teil wird das entwickelte System vorgestellt.

## 2.1 HW/SW Konzepte und Zusammenhänge

Die Fahrspurerkennung wird in der Bildverarbeitungspipeline des im FPGA synthetisierten *SAV Controllers* (Abb. 2.1) durchgeführt, beide sind das Ergebnis von Vorarbeiten [51][41][34][28] und wurden in dieser Arbeit auf den Zynq-7000 MPSoC portiert.

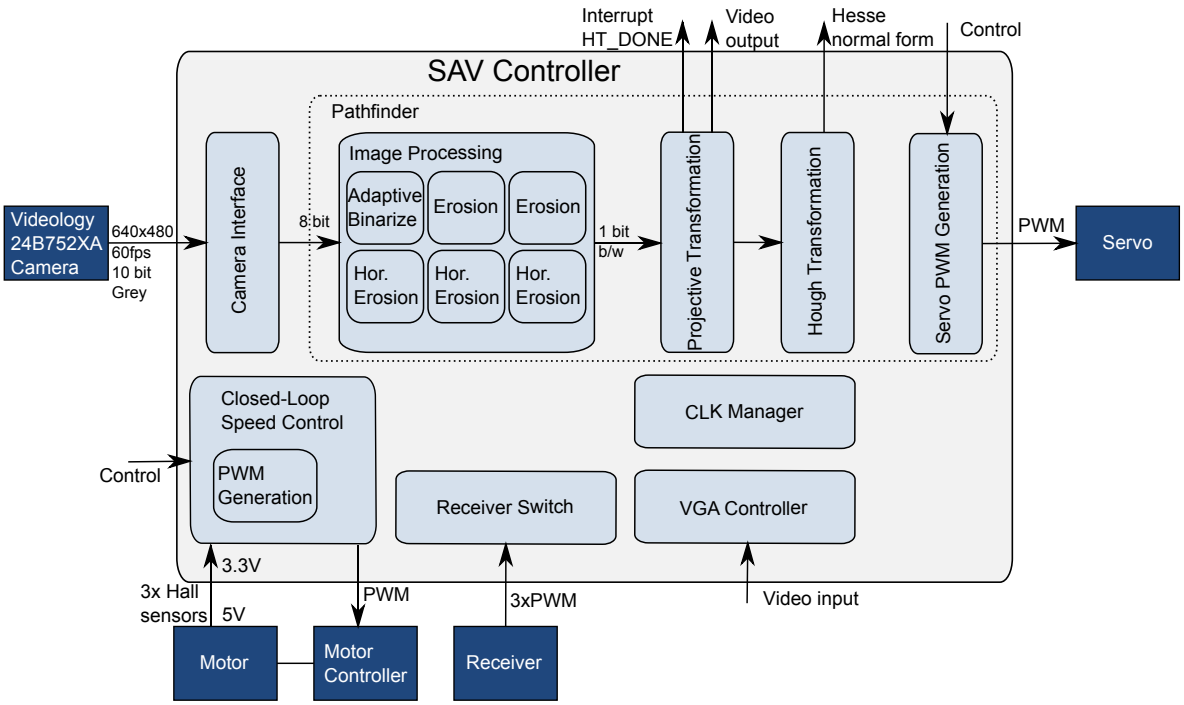



Abbildung 2.1: Funktionsmodule des *SAV Controllers* mit Verbindungen zur Kontrolle und Ausgabe von Daten.

Die Bildverarbeitungspipeline besteht aus

- der adaptiven Binarisierung,
- den Erosionsfiltern,
- der projektiven Transformation

und wird von der Hough-Transformation ausgewertet. Eingabe der Pipeline sind die 8-Bit Grauwerte der Kamera (Kapitel 4.1), Ausgabe ist die Position und der Winkel der Fahrspur in Hesse'scher Normalform (Kapitel 5.3.3, Abb. 5.8). Die Pipeline ist Teil des *Pathfinder*-Moduls, das mit dem auf Matlab Simulink basierten, System Generator implementiert ist. Die PWM für den Servo zur Stellung der Lenkachse ist ebenfalls Teil des *Pathfinder*-Moduls. Liegt ein Ergebnis der Hough-Transformation vor, wird dies der Software mit dem Edge-Level-Interrupt *HT_DONE* signalisiert. Die Software berechnet daraus den Lenkwinkel und gibt diesen als Stellgröße an den *SAV Controller*.

Mit dem Kamera-Interface werden die Kamera-Signale durch Registerketten synchronisiert [47]. Die Takte für die PWM-Generatoren und Multizyklusdatenpfade des *Pathfinder*-Moduls, mit denen mehrere Operationen pro Pixeltakt durchgeführt werden, stellt der Clock Manager bereit (Kapitel 4.1). Für die Visualisierung der Kamerabildes vor- oder nach der Fahrspurerkennung wird der *VGA-Controller* verwendet (Kapitel 4 u. 5.3.5).

Mit der adaptiven Binarisierung wird die Datenmenge auf ein Bit pro Pixel verringert, das resultierende Bild enthält Vordergrund- (Weiß) und Hintergrund-Pixel (Schwarz). Um von verschiedenen Beleuchtungs-Situationen unabhängig zu sein, wird der Schwellwert dynamisch angepasst und aus dem vorherigen Bild berechnet.

Mit den Erosionsfiltern werden einzelne Vordergrund-Pixel erodiert und damit Störeinflüsse minimiert. Bei zwei Filtern wird dafür mit einer 3x3 Faltungsmatrix eine UND-Verknüpfung auf alle Acht Nachbarn jedes Pixels angewendet. Ist mindestens ein Nachbar davon kein Vordergrund-Pixel, wird das zentrale Pixel zum Hintergrund-Pixel. Die drei horizontalen Erosionsfiltern führen diese Operation nur mit den beiden horizontalen Nachbarn aus und verengen damit die Fahrspur, was die Anzahl möglicher Geradenapproximationen reduziert (Abb.2.2).

Die projektive Transformation dreht das von der Kamera aufgenommene Bild in die Vogelperspektive, senkrecht von oben (Kapitel 4.1.2, Abb. 4.2).

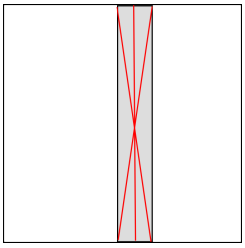


Abbildung 2.2: Menge der Geradenapproximationen (rot) in einer mehreren Pixel breiten Fahrspur (grau).

Mit der Hough-Transformation (HT) werden parametrisierbare geometrische Referenzstrukturen in vorverarbeiteten Bildern detektiert. Die Fahrspur, mit einer Geraden als Referenzstruktur, wird mit der HT innerhalb einer Region-of-Interest (ROI) erkannt. Dazu wird jedes Vordergrund-Pixel innerhalb der ROI in die Hough-Ebene mit der Hesse'sche Normalform (Gleichung 2.1) mit Abstand $r$ und Lotlänge $\Phi$ übertragen. Koordinatenursprung der Hough-Ebene ist die linke obere Ecke der ROI.

$$r = x * cos(\Phi) + y * sin(\Phi) \tag{2.1}$$

Durch Variation des Parameters $\Phi$ in der Hough-Ebene ergibt sich eine Sinuskurve. In der Hough-Ebene wird mit einem Maximum-Verfahren der Punkt gesucht, an dem sich die meisten Kurven schneiden. Dieser Punkt entspricht in der Fahrzeug-Ebene der Geraden, auf der die meisten Vordergrund-Pixel liegen [41][48]. Die im *Pathfinder*-Modul implementierte HT detektiert den Winkel $\Phi$ der Fahrspur innerhalb von ±46° mit einer Schrittweite von 2°, sowie Haltelinien durch einen Winkel von 90°.

Der *Receiver Switch* wird zur Umschaltung zwischen manuellem und autonomen Fahrbetrieb verwendet und wird durch den dritten PWM-Kanal des Empfängers gesteuert. Dadurch kann das Fahrzeug jederzeit durch die Fernbedienung kontrolliert werden. Um im autonomen Fahrbetrieb den Einfluss der Akku-Spannung auf die Geschwindigkeit des Fahrzeugs zu reduzieren wird ein PI-Regler eingesetzt, der die Geschwindigkeit mit den Hall-Sensoren des Motors misst und die Stellgröße als PWM an den Fahrtensteller gibt.

## 2.2 Systemüberblick und HW/SW Co-Design

Ziel dieser Masterarbeit war die Realisierung des Kamera-basierten Spurführungssystems mit dem Zynq-7000 MPSoC (Abb. 2.3).

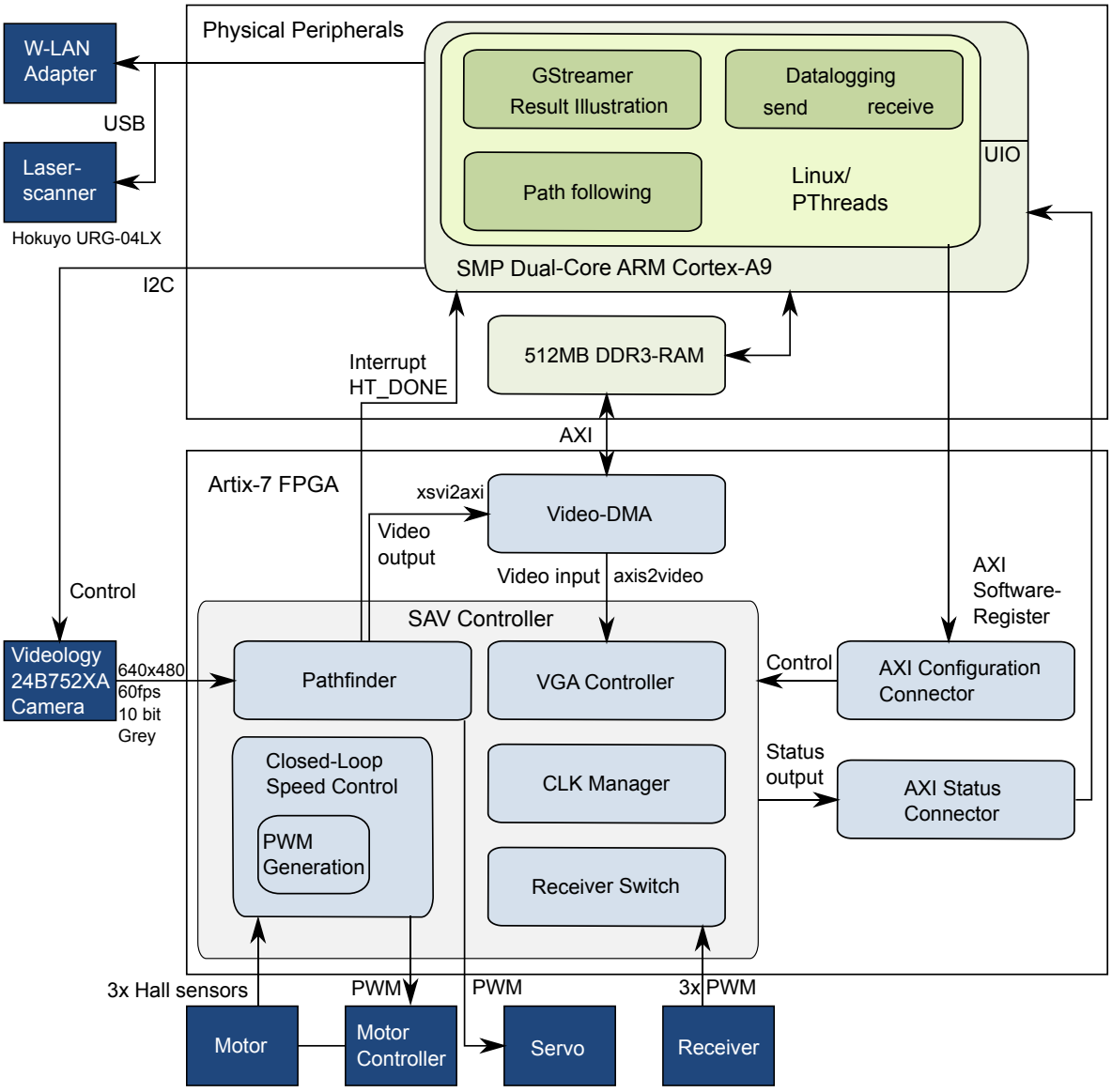



Abbildung 2.3: Die Architektur des SAV im Zynq-7000 Z-7020-MPSoC.

Im Hardware/Software Codesign wurde das System wie folgt eingeteilt:

- Algorithmen mit hohem Datendurchsatz, wie die Bildverarbeitung werden im FPGA berechnet.

- Sequentielle Aufgaben werden in Software durchgeführt.
- Der Spurführungs-Algorithmus wird auf den ARM-Prozessoren unter Verwendung der Fließkommaeinheit realisiert.
- Das Einzeichnen von Orientierungslinien, markanten Punkten sowie Mess- und Stellgrößen zur Visualisierung der Fahrspurerkennung wird von den Prozessoren berechnet, da Erweiterungen in Software mit einer kürzeren Entwicklungszeit implementiert werden können.

Zur Verbindung der FPGA-Komponenten mit dem Prozessorsystem des Zynq-7000 wird der AXI-Bus verwendet. Parameter und Einstellungen des *SAV Controller* werden mit dem *AXI Configuration Connector* in Form von Softwareregistern vorgenommen. Berechnungsergebnisse werden vom *AXI Status Connector* aus Statusregistern gelesen.

Als Betriebssystem wird ein Debian GNU/Linux aufgrund der Unterstützung durch Treiber, wie USB und SW-Bibliotheken (z.B. GStreamer) im SMP-Betrieb auf beiden Prozessoren verwendet. Die parallele Ausführung der Algorithmen auf den CPUs erfolgt mit Pthreads. Auf den FPGA wird unter Linux mit dem UIO-Subsystem des Kernels zugegriffen, das die Memory Mapped Hardware in den virtuellen Adressraum der Anwendung abbildet.

Die Konfiguration des Fahrzeugs, sowie Übertragung von Kamera-Bildern, Mess- und Stellgrößen erfolgt über W-LAN.

Mit dem Video-DMA IP-Core wird die Bildverarbeitungskette an die DDR3-Speicher der Prozessoren angebunden. Dadurch kann der Videostrom zur Visualisierung in allen Zwischenschritten von den Prozessoren gelesen werden. Zur Ausgabe werden die Bilder über VGA direkt auf einen Monitor oder über W-LAN an einen Entwicklungs-PC gesendet. Zum Anschluss des Video-DMA Moduls an die Kamera und den *VGA-Controller* wurden die IP-Cores *xsvi2axi* und *axis2video* entwickelt. Die Bearbeitung der Bilder wird vom Multimedia-Framework GStreamer übernommen, das auch die Daten in JPEG komprimiert und über Netzwerk überträgt.

Die Parametrisierung der CMOS-Kamera erfolgt über den I2C-Bus. Zur vorausschauenden Erkennung von Hindernissen des Fahrzeugs wurde ein 2D-Laserscanner über USB angeschlossen.

Zur Hardware- und Software-Entwicklung wurde Xilinx ISE (Integrated Software Environment) mit dem EDK (Embedded Development Kit) in Version 14.3 eingesetzt. Für den System Generator wurde Matlab Simulink R2012a verwendet.

# 3 Zynq-7000 ARM Multiprozessor System-on-Chip

In diesem Kapitel wird der Zynq-7000 MPSoC mit der Peripherie zur Steuerung der Kamera, für das Datalogging über Netzwerk und Erkennung von Hindernissen beschrieben. Die Algorithmen des autonomen Fahrzeugs werden in der auf zwei physischen Cortex-A9 Prozessoren basierenden MPSoC Plattform Zynq-7000 (vgl. Abb. 3.1) [61] berechnet. Der Zynq-7000 teilt sich in die physische Peripherie (Processing System, PS) sowie den FPGA (Programmable Logic, PL) und verfügt über die folgenden Eigenschaften:

- SMP fähiger Dual-Core Cortex A9 ARM Prozessor mit 32-Bit ARMv7 Befehlssatz
- Je Prozessor zwei 32kB Level-1 Caches, einen für Befehle und einen für Daten sowie eine MMU, Fließkommaeinheit und NEON SIMD-Koprozessor.
- Der 512kB Level-2 Cache wird von den Prozessoren geteilt, die Snoop Control Unit sorgt für die Herstellung der Cache-Kohärenz.
- Physische Peripherie: I2C, SPI, CAN, UART, GPIOs, Gigabit-Ethernet und USB 2.0 Hosts mit DMA, zusätzliche Peripherie kann in der PL modelliert werden.
- Über die Advanced Mikrocontroller Bus Architecture (AMBA) sind die Prozessoren an die Peripherie und die PL angeschlossen.

Der MPSoC ist eine zusätzliche Option zu den bisherigen System-on-Chips, bestehend aus MicroBlaze und programmierbaren Logik und Zwei-Chip Designs mit FPGA und Prozessor auf einer Platine. Die mit bis zu 800MHz getakteten ARM-Prozessoren liefern eine höhere Datenverarbeitungsgeschwindigkeit als die mit max 200MHz getakteten MicroBlaze-Prozessoren, wodurch mehr Algorithmen eines Produktes in Software implementiert werden können.

Im Unterschied zu den bisherigen FPGAs mit integriertem PowerPC-Prozessor, verfügt die Zynq-7000 Plattform über physische Peripherie-Elemente wie Bussysteme, USB 2.0 Host und Gigabit-Ethernet, die es erlauben, den Prozessor ohne die PL zu benutzen. Ein Betriebssystem wie Linux ist ohne die Verwendung von PL-Ressourcen lauffähig.

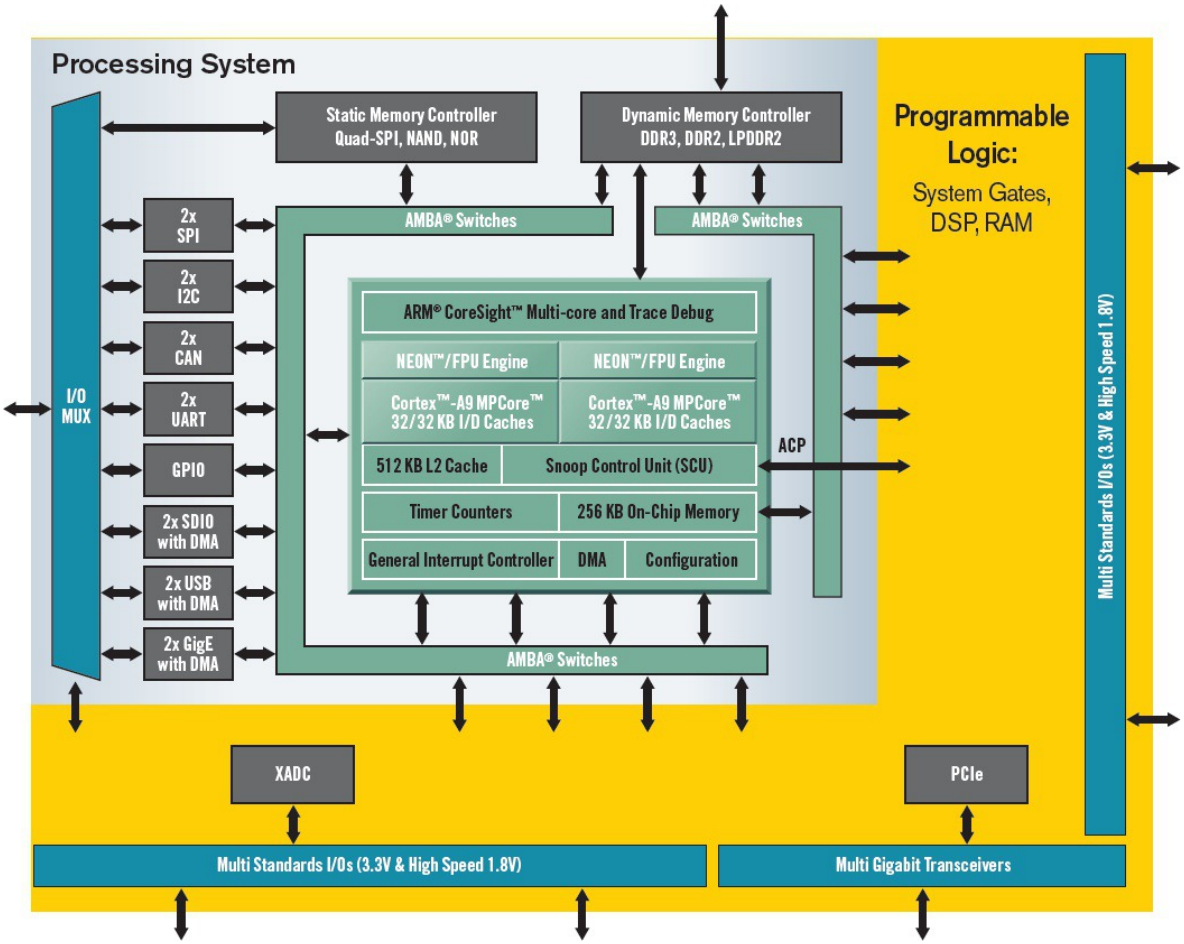



Abbildung 3.1: Die Zynq-7000 Architektur mit Verbindungen zwischen PS und PL. Bild: [61].

Durch die Integration der beiden Komponenten auf einem Chip, sind die Leiterverbindungen gegenüber Zwei-Chip-Designs kürzer, und damit der Bus zwischen FPGA und Prozessor schneller. Gegenüber einer Zwei-Chip-Lösung wird weniger Strom verbraucht, außerdem werden durch den Wegfall eines Bausteins die Gesamtbetriebskosten in der Entwicklung gesenkt und Platz im Produkt gespart.

Als Bussystem wird für die Kommunikation des PS mit der PL in dem Zynq-MPSoC das Advanced eXtensible Interface (AXI) verwendet, das Teil der ARM Advanced Mikrocontroller Bus Architecture (AMBA) ist [2], die der Standard in ARM basierten SoCs ist. Die AXI-Anbindung der PL an das PS ist mit bis zu 9600MB/s angegeben [59].

Bei der ARM Cortex-A9 Architektur kontrolliert die Snoop Control Unit (SCU) den Inhalt der beiden Level 1 Daten-Caches und sichert deren Konsistenz [6]. Ist der Inhalt eines Caches veraltet, aktualisiert die SCU dessen Inhalt. Der Prozessor kann währenddessen andere

Aufgaben erledigen, bis die SCU per Interrupt mitteilt, daß der Cache aktualisiert wurde. Als Besonderheit bei der Zynq-Plattform dient der Accelerator Coherency Port (ACP) analog dazu, beide L1-Caches mit der PL zu synchronisieren und wird für den Datenaustausch eingesetzt, wenn Algorithmen in der PL beschleunigt werden.

Die Peripherie des PS teilt sich die 54 Multiplexed I/O (MIO) Pins. Einer Teilmenge der Peripherie, wie I2C und GPIO, stehen Pins der PL als Extended-MIO (EMIO) Pins zur Verfügung.

Der Interrupt-Controller verarbeitet Interrupts der PS (IDs bis 83) und der PL (IDs 84 bis 91). Die Interrupts können zu jedem Prozessor geroutet werden.

Für die Softwareentwicklung werden die in der ARM-Welt etablierten Werkzeuge, wie die ARM Entwicklungsumgebung DS-5, der GNU C-Compiler und die Lauterbach Entwicklungs- und Debugwerkzeuge eingesetzt.

Zu den bereits unterstützten High-Level-Betriebssystemen (HLOS) gehören Linux, Android und MS-Windows Embedded Compact 7, zu den unterstützten Echtzeitbetriebssystemen (RTOS) zählen Micrium µC/OS-III, FreeRTOS, QNX und T-Kernel. Dabei können die Betriebssysteme sowohl im SMP-Betrieb, als auch im AMP-Betrieb arbeiten. Im AMP-Betrieb können pro Prozessor unterschiedliche Betriebssysteme laufen, wie Linux auf einem Prozessor und QNX auf dem anderen.

Als Zynq-7000-Entwicklungsplattform wird das von den Firmen Digilent und Avnet entwickelte Zedboard [72] (Abbildung 3.2) eingesetzt. Es verfügt über:

- Einen Zynq-7000 Z-7020 System-On-Chip, die ARM Prozessoren lassen sich mit bis zu 667MHz takten.
- 512MB DDR3 Arbeitsspeicher.
- SD-Kartenleser zum laden des Betriebssystems.
- Debug-Schnittstellen: Integrierten USB-JTAG, JTAG-Anschluss und RS-232.
- Kommunikation: Gigabit-Netzwerk Verbindung.
- Ausgabe von Videosignalen über einen HDMI- und VGA-Anschluss.
- Peripherie: USB 2.0 Host sowie ein FMC- und fünf Pmod-Anschlüsse um auf die PL-Pins zuzugreifen.
- Acht LEDs, Acht Schalter, fünf Taster und ein 128x32 OLED-Display.

Der Z-7020 Zynq-7000 MPSoC enthält einen Artix-7 FPGA im CLG484-Gehäuse mit 13300 Slices und 560KB Block-RAM, ein Slice enthält 4 LUTs und 8 FlipFlops.

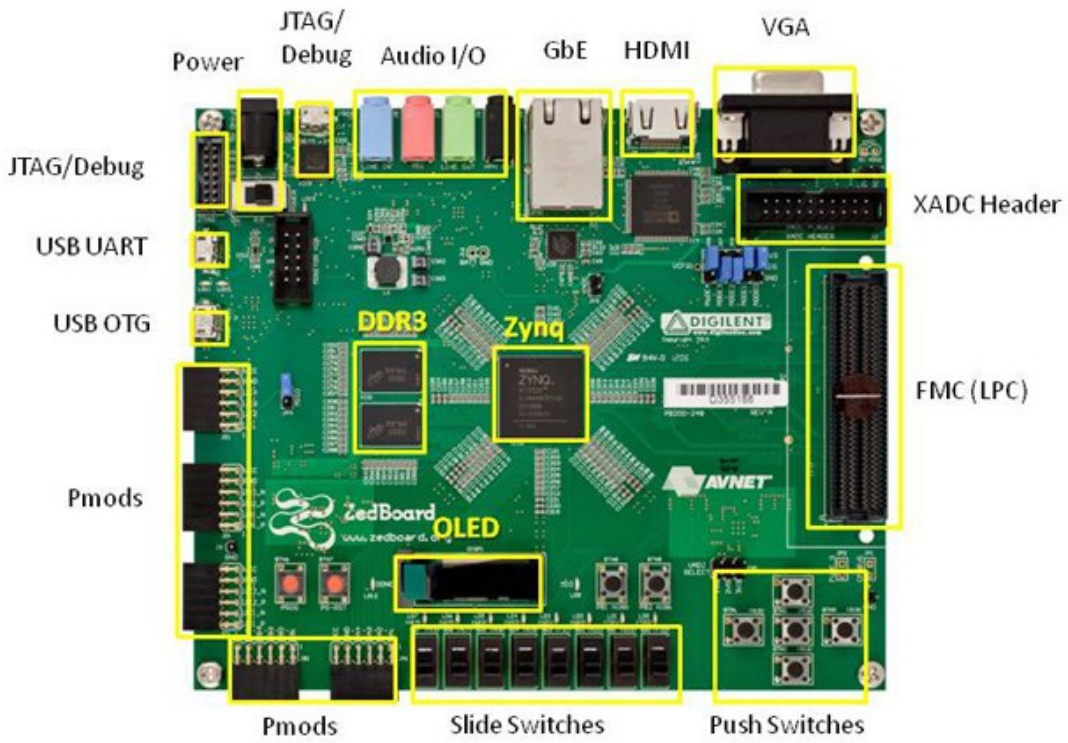



Abbildung 3.2: Zynq-7000 Entwicklungsplattform Zedboard. Bild: [72]

Die Kamera und Peripherie werden über eine FMC XM105 Debug Karte angeschlossen, die 75 Pins des Zynq-7000 im 2.54mm Raster zur Verfügung stellt.

### 3.0.1 Takte im Zynq-7000 MPSoC

Auf dem Zedboard wird der Takt des PS durch einen 33.33MHz Oszillator bereitgestellt. Mit einer Phase-Locked Loop (PLL) wird daraus der *CPU_1x*-Takt mit 111MHz generiert und aus diesem mit *CPU_2x* und *CPU_6x* der doppelte und sechsfache (667MHz) Takt erzeugt. Die drei Takte werden von der Peripherie verwendet, z.B. arbeiten die beiden I2C-Controller mit *CPU_1x* und die Prozessoren mit *CPU_6x*. Die AXI IP-Cores verwenden einen 100MHz Takt, der durch eine andere PLL aus einem Oszillator mit 100MHz am Zedboard bereitgestellt wird.

## 3.1 AXI-Anbindungen an den ARM Cortex-A9 Dual-Core

Als Bussystem für die Kommunikation der PS- und PL-Peripherie mit den Prozessoren wird in dem Zynq-MPSoC das Advanced eXtensible Interface (AXI) verwendet, daß zu der ARM Advanced Mikrocontroller Bus Architecture (AMBA) gehört [2]. AMBA wird standardmäßig für Bussysteme in ARM basierten SoCs eingesetzt, das PS des Zynq-7000 MPSoC implementiert mit AXI3 die AMBA in Version 3.0. Die PL IP-Cores von Xilinx, wie z.B. der Video-DMA Core oder das IP Interface (IPIF) verwenden AXI4, bzw AXI4-Lite aus AMBA4. In AMBA sind die folgenden AXI-Varianten definiert [7][5][69]:

- **AXI4-Lite**: Adressbasierte AXI-Variante für 32-Bit Transaktionen, die mit wenig FPGA-Ressourcen auskommen soll.
- **AXI3/AXI4**: Adressbasierte schnellere Variante, da auf dem Bus per Burst bis zu 256 Datenpakete (16 bei AXI3) mit 32-1024 Bit pro Paket in einer Transaktion transportiert werden. Trägt den gleichen Namen wie die Protokoll-Familie.
- **AXI4-Stream**: Adresslose Variante für Datenstrom-basierte Daten, wie z.B. Video.

Über AXI3/4 und AXI4-Lite angebundene PS- und PL-Peripherie sind Teil des Adressraums der ARM Prozessoren, weswegen in der Software auf die Peripherie mit Memory Mapped I/O zugegriffen wird. Peripherie im Adresslosen AXI4-Stream ist nicht von den ARM Prozessoren aus erreichbar, das AXI4-Stream Protokoll für Verwendung mit Videodaten beschreibt Kapitel 4.2.2.

An einem AXI3/4 und AXI4-Lite Bus gibt es Master, die über ein Interconnect-Element mit mehreren Slaves verbunden sind, ein Datenaustausch wird immer von einem Master zu einem Slave initiiert und kontrolliert. Der Interconnect IP-Core sorgt in der PL für die Arbitrierung der Zugriffe und verbindet AXI3 mit AXI4 Peripherie. Pro Bus gibt es im AXI einen Kontrollkanal, über den u.a. die Adresse gesendet wird, sowie je einen Lese- und Schreib-Kanal, mit denen ein Schreibvorgang parallel zum Lesevorgang auf einer Adresse durchgeführt werden kann.

Während AXI4 sich aufgrund des höheren Datendurchsatzes für das Kopieren von Daten eignet, wird AXI4-Lite für Register-Zugriffe verwendet. Peripherie-Elemente können beide Protokolle verwenden, z.B. ein DMA-Controller, der über AXI4-Lite Register konfiguriert wird und die Daten mit dem Burst fähigen AXI4 transferiert.

Der Zynq-7000 verfügt über Neun AXI3-Interfaces für die Anbindung von AXI IP-Cores der PL:

- Zwei AXI3 General-Purpose (GP) Slave Ports,
- Zwei AXI3 General-Purpose (GP) Master Ports,
- Vier AXI3 High-Performance (HP) Slave Ports,
- Ein Advanced Coherency (ACP) Port.

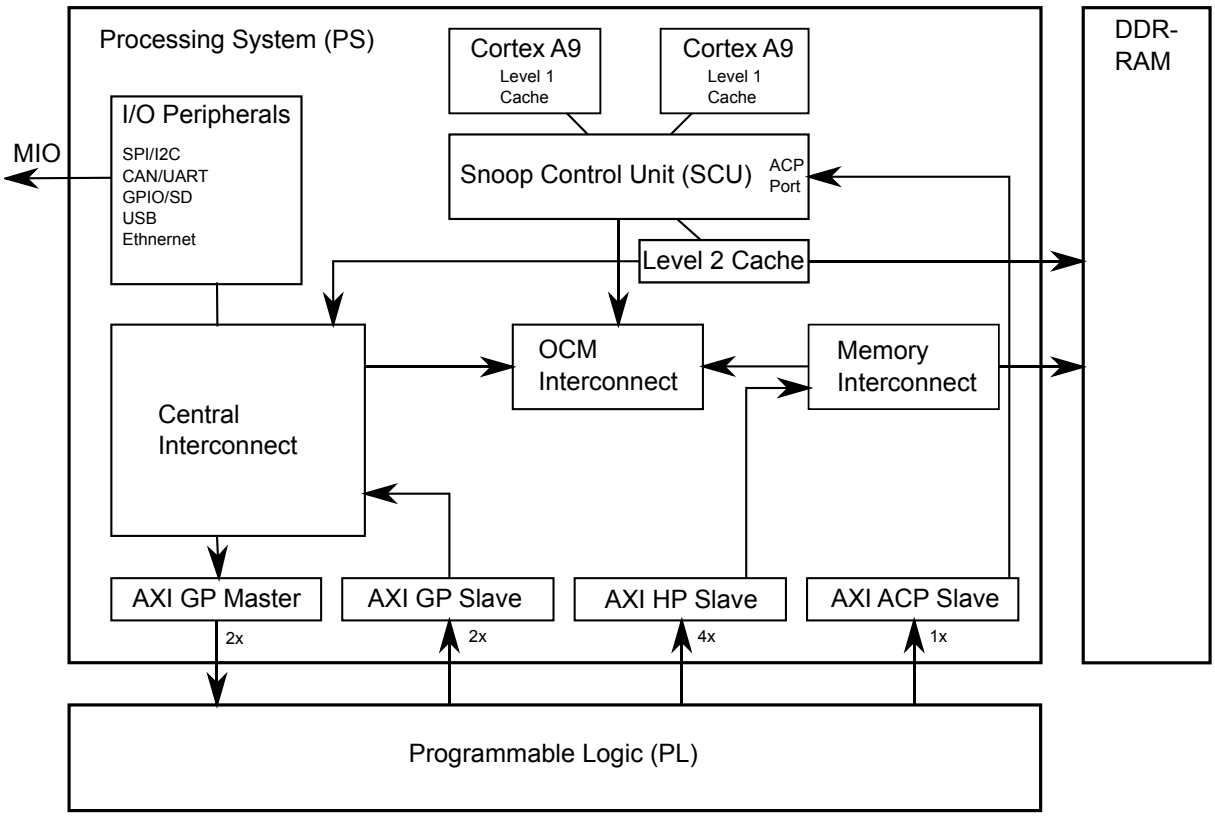



Abbildung 3.3: AXI-Anbindungen des Zynq-7000 mit AMBA-Switchen und MIO-Peripherie. Die Pfeile der AXI-Verbindungen zeigen den Kontrollfluss, Daten werden in beide Richtungen transportiert.

Die Vier AXI3 HP Slave Ports sind mit einer konfigurierbaren 32/64 Bitbreite über den Memory Interconnect mit dem Speicher verbunden. Als Slave Ports initiieren sie keinen Datenaustausch. Zur Vermeidung von Unterbrechungen im Datenstrom werden FIFOs eingesetzt. Sie eignen sich damit für den Anschluss von IP-Cores, die hohen Datendurchsatz

benötigen, wie Burst fähige DMA-Controller zum Kopieren von Daten aus und in den Arbeitsspeicher. Die Datenübertragungsrate ist mit bis zu 1,2GB/s pro Port angegeben und damit u.a. für Videoströme geeignet. Nachteil ist die starke Belegung von FPGA-Ressourcen und die höhere Latenz der AXI3-HP Anbindung [59].

Über den ACP Port können IP-Cores direkt in die Level 1+2 Caches der CPU schreiben, der ACP Port eignet sich dadurch für den Datenaustausch mit geringer Latenz zwischen dem PS und der PL. Verglichen mit der AXI3-HP Anbindung belegt die AXI3-ACP Anbindung mehr FPGA-Ressourcen bei der gleichen Übertragungsrate von 1,2GB/s. Durch den direkten Zugriff auf den CPU-Cache, kann die CPU ausgebremst werden, da weniger Cache für Programme zur Verfügung steht. Durch die L2-Cachegröße von 256kB ist diese Anbindung nur für kleinere Datenmengen geeignet [59].

Die Zwei AXI3 GP Slave und Zwei GP Master Ports werden für den Austausch von Kontrolldaten von den Prozessoren mit IP-Cores verwendet. Da für den Kopiervorgang CPU-Zeit benötigt wird, sind die Transferraten auf 25MB/s begrenzt. Die GP Master Ports eignen sich, um mit den Prozessoren per AXI4-Lite auf Softwareregister der PL zuzugreifen. Über die GP Slave Ports können IP-Cores mit 600MB/s auf Register und Peripherie des PS zugreifen. Die Verwendung von GP Ports hat den geringsten Verbrauch an FPGA-Ressourcen [59].

Die Verwendung der AXI4-Protokolle mit den AXI3-Interfaces zeigt Tabelle 3.1.

Die AXI3-Ports werden in Kapitel 4 zur Anbindung der Fahrspurerkennung an die Prozessoren verwendet. Die Ports sind über die folgenden AMBA-Switche (Abb. 3.3) miteinander verbunden [59]:

- **Snoop Control Unit (SCU)**: Stellt die Kohärenz der Level-1 Daten-Caches sicher. Der Prozessor greift über die SCU auf den Arbeitsspeicher zu.
- **Central Interconnect**: Kern-Switch, der die Peripherie und AXI3 General-Purpose Ports mit dem OCM und dem Level-2 Cache verbindet.
- **Memory Interconnect**: Verbindet die AXI3 High-Performance Slave Ports mit dem Arbeitsspeicher.
- **OCM Interconnect**: Verbindet die Peripherie am Central Interconnect mit den 256kb On Chip Memory (OCM) und den Arbeitsspeicher über den Memory Interconnect.

| PS AXI3-Port / AXI4-Protokoll | AXI3 GP Slave Ports | AXI3 GP Master Ports | AXI3 HP Slave Ports | AXI3 ACP Slave Port |
|---|---|---|---|---|
| **AXI4-Lite** | Über AXI-Interconnect (Kapitel 4.3) | Über AXI-Interconnect | | |
| **AXI4 (Burst)** | | | Über AXI-Interconnect, max 16x Burst a 32/64 Bit, Datendurchsatz mit FIFOs beschleunigt | Über AXI-Interconnect, max 16x Burst a 64 Bit |
| **AXI4-Stream** | | | Mit Datamover DMA, z.B. VDMA in AXI4 (Burst) (Kapitel 4.2) | Mit Datamover DMA in AXI4 (Burst) umgesetzt |

Tabelle 3.1: Matrix mit Verbindungen der PL AXI4-Protokolle mit den AXI3-Ports des PS.

## 3.2 SD Speicherkarte und Linux Boot-Konfiguration

Der Zynq-7000 SoC unterstützt das Booten (Abb. 3.4) von JTAG, NAND- und NOR-Flash, Quad-SPI sowie per SD-Karte, die Boot-Quelle wird über Jumper vorgenommen (vgl. Tabelle 3.2) [59]. Das Zedboard wird mit SD-Karte und integriertem JTAG ausgeliefert [8] und ist im Auslieferungszustand zum Booten von SD-Karte eingestellt.

| | MIO[5] | MIO[4] |
|---|---|---|
| **JTAG** | 0 | 0 |
| **Quad-SPI** | 1 | 0 |
| **SD-Karte** | 1 | 1 |

Tabelle 3.2: Jumper-Position für die Einstellung der Boot-Methode des Zynq-SoCs am Zedboard (1 = 3.3V)

Der Bootvorgang von einem Flash-Speicher läuft in mehreren Stufen ab. Nach dem Einschalten der Versorgungsspannung oder einem Reset-Vorgang wird der Boot-ROM als Startup-Code durch die CPU0 ausgeführt. Zweck des Boot-ROMs ist es den First-Stage-Bootloader (fsbl) über NAND-, NOR-Flash, Quad-SPI, SD-Karte oder JTAG zu laden und auszuführen. Der fsbl wird auf den OCM geladen und ist dadurch auf 192kB begrenzt.

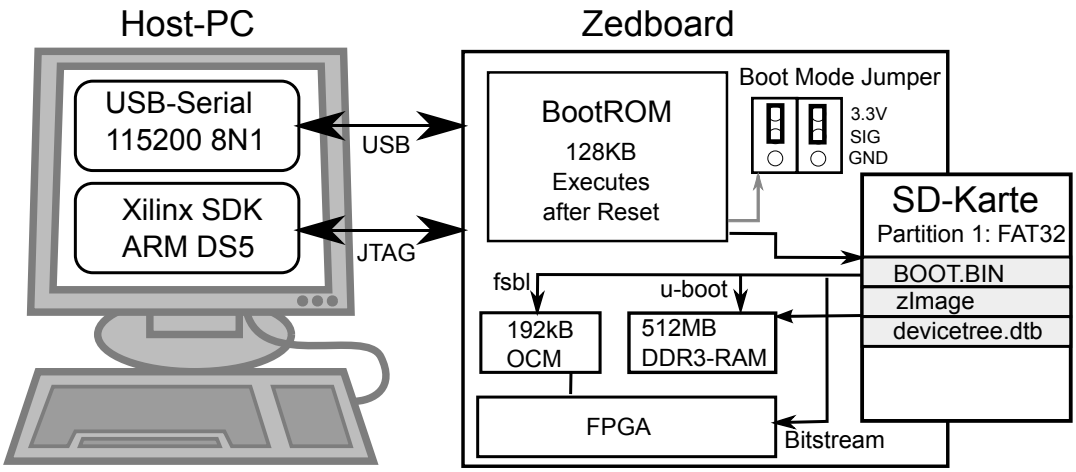



Abbildung 3.4: Bootloader-Vorgang des Zynq-7000.

Ist der JTAG als Boot-Methode gewählt, wird der JTAG-Anschluss durch den Boot-ROM aktiviert und der Prozessor angehalten. Zur Fortsetzung des Bootvorgangs wird dann ein Systemabbild per JTAG in den Speicher geladen.

Bei einem Startvorgang von SD-Karte wird die Datei *BOOT.BIN* durch den Boot-ROM gelesen, die Datei enthält den FPGA-Bitstrom sowie den First- und Second Stage Bootloader. Die Datei muss auf der mit FAT32 formatierten, ersten Partition der max. 32GB großen SD-Karte im „root“-Verzeichnis gespeichert sein und wird mit dem Xilinx Boot Image Creator erzeugt (Abb. 3.5) [60].

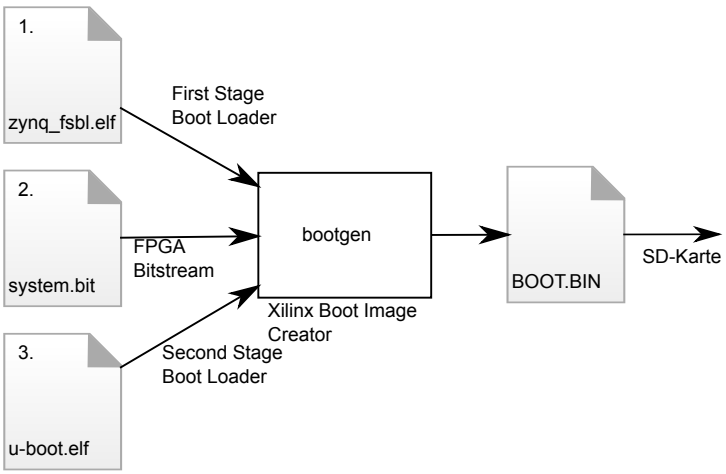



Abbildung 3.5: Erzeugung des *BOOT.BIN*-Abbilds zum kopieren auf die SD-Karte. Die drei Quelldateien müssen in der angegeben Reihenfolge durch das Tool *bootgen* in das Abbild geschrieben werden.

Der First-Stage-Bootloader initialisiert den DDR-RAM, die MIO-Pins und Peripherie wie die RS232-Schnittstelle für Status-Ausgaben und lädt den FPGA-Bitstrom aus dem Speicher nach. Nach der Programmierung der PL wird der Second-Stage-Bootloader U-Boot [15] aus der *BOOT.BIN* geladen und ausgeführt. Der Bootloader U-Boot lädt anschließend den Linux-Kernel als *zImage* mit der Hardware-Konfigurationsdatei *devicetree.dtb*. Die CPU1 bleibt bis zu diesem Punkt im stromsparenden „Wait For Event“ (WFE) Modus. Ist der Linux-Kernel für den SMP-Betrieb konfiguriert, startet er die die CPU1 durch Ausführung des „Signal Event“ (SEV)-Befehls (Abb. 3.6).

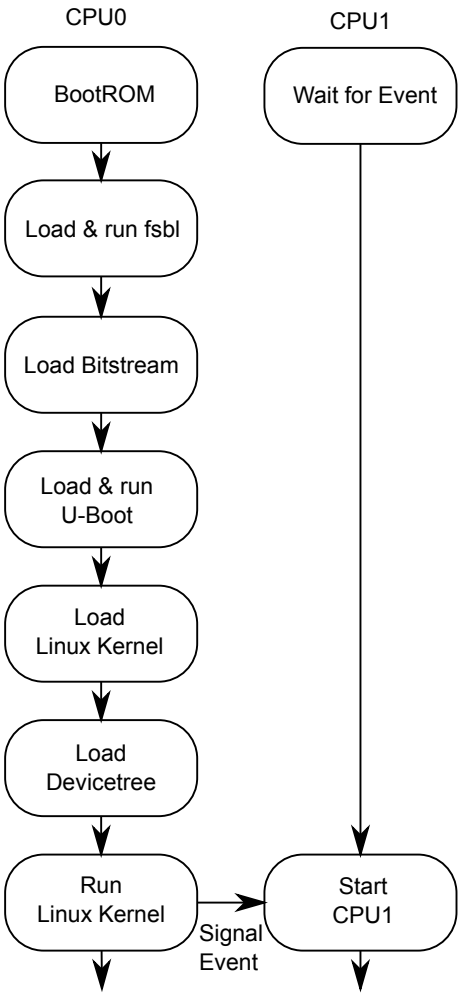



Abbildung 3.6: Zynq Boot-Sequenz zum Start des Linux-Kernels.

## 3.3 Netzwerkkommunikation zum Loggen von Fz. Betriebsdaten

Das Aufzeichnen von Mess- und Stellgrößen, sowie die Parametrisierung über einen entfernten Entwicklungs-PC erfolgte im bisherigen Fahrzeug über die Bluetooth-Schnittstelle [28]. Da zusätzlich die Bilder der Kamera übertragen werden (Kapitel 5.3.5), wird die Übertragung auf das für höhere Datenübertragungsraten ausgelegte Wireless-LAN (W-LAN) umgestellt (Kapitel 5.3.4). Als Hardware wird der Asus WL-167G W-LAN Adapter (Realtek RTL8188SU Chipsatz, Standard 802.11n) verwendet, der über die USB-Schnittstelle angeschlossen wird.

Das Fahrzeug erhält über W-LAN eine feste IP-Adresse, als Gegenstelle der Funkstrecke arbeitet ein W-LAN Wireless Access Point Linksys WRT160NL mit der freien OpenWRT [43] Linux Distribution. Der Access Point ist als Bridge konfiguriert und verbindet damit auf OSI Layer 2 (MAC-Schicht) die W-LAN Geräte mit dem Ethernet-LAN des Entwicklungs-PCs.

Da die Kommunikation auf dem Internet Protokoll (IP) beruht, erfolgt die Kommunikation wahlweise auch über den Gigabit Ethernet Controller.

## 3.4 USB 2.0 Host zur Anbindung externer Peripherie

Das Zedboard ist mit einem, von den zwei im Zynq-7000 MPSoC, USB 2.0 On-The-Go (OTG) Controllern [59][8] bestückt. USB ist ein Host gesteuertes Bussystem, bei dem ein USB-Host als Master die Kommunikation mit bis zu 127 USB-Geräten steuert. Der PC als USB-Host muss die Bus-Spannung bereitstellen und alle Kommunikationsarten (Control, Bulk, Interrupt und Isochronous) sowie alle Geschwindigkeiten (Low-, Full- und High-Speed) unterstützen, während das USB-Gerät sich auf eine Teilmenge beschränkt. Um Zwei USB-Geräte wie Drucker und Kamera ohne PC miteinander zu verbinden, wurde der OTG-Standard entwickelt [53]. Erkennt ein OTG-Gerät am elektrischen Pegel, daß es mit einem USB-Host verbunden ist, schaltet es automatisch in den Device-Mode und umgekehrt. Werden zwei OTG-Geräte miteinander verbunden, wird durch ein definiertes Protokoll ausgehandelt, welches Gerät den Host oder Device-Part übernimmt. Der OTG-Controller des Zynq-7000 MPSoCs unterstützt den Hi-Speed Modus mit einer maximalen Datenübertragungsrate von 480 MB/s. Das Kopieren der Daten von/zum USB-Controller wird mit DMA vorgenommen.

Mit dem Laserscanner und dem W-LAN Netzwerkadapter verfügt das SAV-Fahrzeug über zwei, per USB-Hub angeschlossene, USB-Geräte.

### 3.4.1 Laserscanner basierte Objekterkennung

Für die vorausschauende Erkennung von Hindernissen durch das SoC-Fahrzeug überträgt ein Hokuyo URG-04LX 2D-Laserscanner (Abbildung 3.7) die gemessenen Abstandswerte über eine Mini-USB Schnittstelle an das Prozessorsystem des Zynq. Laserscanner werden in der Automobilbranche zur Erhöhung der Verkehrssicherheit, in der Militärtechnik für Ortung und in der Sicherheitstechnik eingesetzt [27].

| | |
|---|---|
| **Power Source** | 5V DC ± 5% |
| **Detection Range** | 20mm ~ 5.6m: White Square Sheet 70mm 240° |
| **Scan Window** | 240° |
| **Accuracy** | Up to 1m: ± 10mm. Above 1m 1% of distance |
| **Angular Resolution** | 0.3515625° (360° / 1,024 steps) |
| **Light Source** | Laser diode (λ =785nm) Laser safety class 1 |
| **Scan Time** | 100 msec/scan |
| **Sound Level** | Less than 25 dB |
| **Interface** | RS232 + USB 2.0 |
| **Synchronous output** | NPN open collector |
| **Command system** | SCIP 1.1 / 2.0 |
| **Temperature/Humidity** | -10°C -50°C, < 85% RH (no condensation / icing) |
| **Vibration Resistance** | Double amplitude 1.5mm 10 ~ 55 Hz,<br>2 hrs in X, Y, Z direction |
| **Impact Resistance** | 196 m/s², 10 times each in X, Y and Z direction |
| **Weight** | Approx 160g |

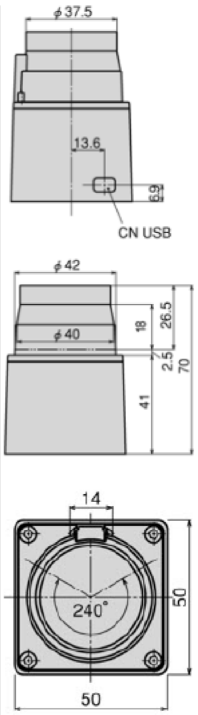



Abbildung 3.7: Spezifizierung und Abmessungen des URG-04LX Laserscanners [25].

Da der Laserscanner mit einer maximalen Stromaufnahme von 800mA die im USB 2.0 Standard erlaubten 500mA [53] überschreitet, werden die benötigten 5V durch einen LM2576 [50] Step-Down Spannungsregler zur Verfügung gestellt. Der LM2576 erzeugt die 5V ab einer Eingangsspannung von ca. 6.5V, die der Akkupack mit 7.2V Nennspannung liefert.

## 3.5 Steuerung der Videology 24B7525A CMOS-Kamera über I2C

Der Zynq-7000 MPSoC verfügt über zwei physische I2C-Controller für den Master- und Slave-Betrieb bei einer maximalen Datenübertragungsrate von 400Kb/s. Weitere I2C Controller können in der PL synthetisiert werden.

Für die Aufnahme der Fahrspur wird die Videology 24B7525A CMOS-Kamera mit parallelem Interface (3.3V LVTTL Pegel) verwendet [55]. Die Kamera liefert ein monochromes Bild mit einer Auflösung von 10 Bit für die Grauwerte. Mit dem 27MHz Oszillator liefert die Kamera, bei einer Auflösung bis zu 752x480 Pixel, 60 Bilder pro Sekunde, bei geringeren Auflösungen proportional mehr Bilder. Für die Kamera sind Weitwinkelobjektive bis 160° vorhanden, zur Zeit ist ein Objektiv mit einem Blickwinkel von ca. 90° montiert [47].

Die Parametrisierung der Kamera erfolgt über den I2C-Bus, die im Bus benötigten Pullup-Widerstände werden durch den FPGA bereitgestellt. Dazu werden die I2C-Signale über die EMIO-Schnittstelle in die PL geroutet und das Schlüsselwort *PULLUP* in der User Constraint File (.ucf-Datei) angegeben [63]. Die Portierung der Kamera-Parametrisierung [47] auf den Zynq-7000 MPSoC beschreibt Kapitel 5.

# 4 Kopplung der Fahrspurerkennung mit dem Prozessorsystem

Die Erkennung der Fahrspur in der PL des Zynq-7000 MPSoC realisiert der *SAV Controller* IP-Core (Kapitel 2.1, Abb. 2.1). Die Anbindung dieses Controllers an die Prozessoren des MPSoCs zur Steuerung und Visualisierung erfolgt über den AXI-Bus.

Die Visualisierung der Fahrspur-Approximation, der Spurführungs-Regelgrößen und der ROI mit Markierungslinien, wurde in [34][28] mit RTL-Modulen implementiert. Dabei wurden die durch Überlagerung generierten Videodaten durch einen VGA-Controller [47] direkt auf einen VGA-Monitor ausgegeben.

Um die Videodaten bei Testfahrten über Netzwerk an einen Entwicklungs-PC zu senden, wird mit einer Software-Lösung der Kamera-Videostrom als Einzelbilder in den Speicher der CPU ablegt. Zusätzlich wird zur Visualisierung das Einzeichnen von Orientierungslinien, markanten Punkten und Mess- und Stellgrößen von den Prozessoren berechnet, da Erweiterungen in Software schneller implementiert werden können. Zur Ausgabe der Bilder über den VGA-Port werden die von der CPU modifizierten Bilder wieder zurück in die PL transferiert. Dafür wird ein VDMA IP-Core verwendet, der zur Konfiguration ein AXI4-Lite Interface, und zum Senden sowie Empfangen je ein Burst fähiges AXI4 Interface verwendet (Abb. 4.1).

Für zwei Bilder mit 640x480 Pixel mit 4 Byte pro Pixel bei 60 Bildern pro Sekunde wird ein Datenaufkommen von $2 * 75MB/s$ erwartet. Zum Transport der Videodaten werden beide Burst fähigen AXI4 Anschlüsse des VDMA IP-Cores mit einem HP Slave Port des Zynq-7000 verbunden. Mit einem theoretischen maximalen Durchsatz von 1,2GB/s durch die AXI3 HP Slave Ports und 4,2GB/s durch das DDR-Arbeitsspeicher Interface [59] steht ausreichend Übertragungsrate zur Verfügung. Um mit den Prozessoren den VDMA IP-Core zu konfigurieren wird das AXI4-Lite Interface mit einem GP Master Port des Zynq-7000 verbunden.

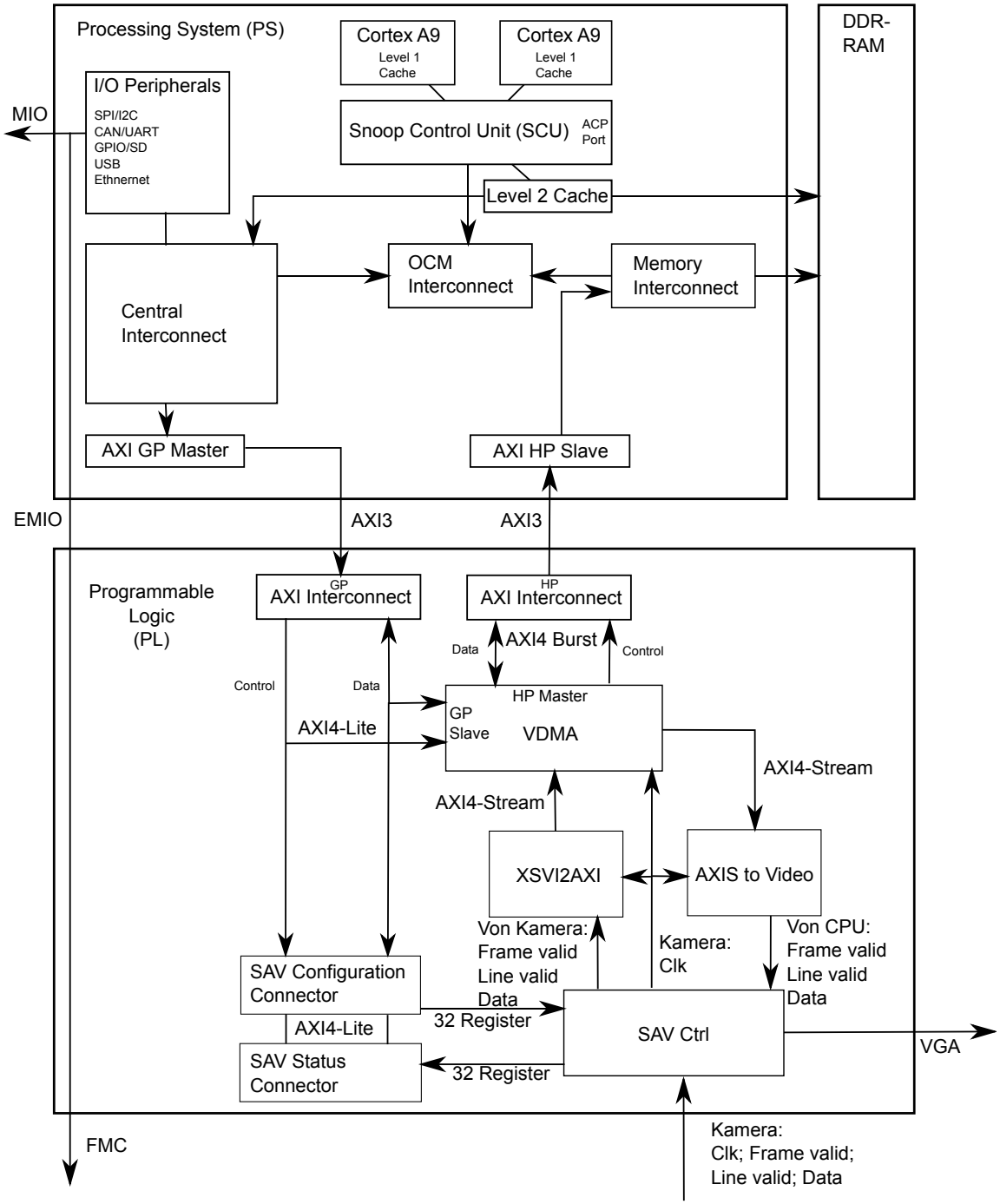



Abbildung 4.1: AXI-Anbindungen des *SAV Controllers* zum Prozessorsystem.

## 4.1 Portierung des synchronen Kamera-Interfaces auf den Zynq-MPSoC

Die Fahrspurerkennung in der Bildverarbeitungspipeline [34][28] ist Teil des Prozessor unabhängigen *SAV Controllers*. Die Verbindung an den Prozessor erfolgt über den *SAV Connector* der in Kapitel 4.3 vom Microblaze/PLB-System auf den Zynq-7000 portiert wurde.

Da im neuen Fahrzeug die Videology 24B7525A CMOS-Kamera arbeitet, wurde die Bildverarbeitungspipeline angepasst. Das in einer Vorarbeit entwickelte Interface der CMOS-Kamera [47] wurde auf den letzten Stand [28] der Bildverarbeitungspipeline portiert und der Digital Clock Manager (DCM) an die 27MHz der Kamera angepasst. Die Videology 24B7525A Kamera sendet mit bei einer Taktfrequenz von 27MHz mit 60 Bildern pro Sekunde einen 10-Bit 640x480 Pixel Videostrom.

Da die Bildverarbeitungspipeline mit 8-Bit Graustufen arbeitet, werden die 2 LSBs der 10 Bit Grauwerte abgeschnitten. Die Interlacing-Komponenten wurden aus der Bildverarbeitungspipeline entfernt und diese durch Parametrisierungen [28] an die 640x480 Vollbilder der Kamera angepasst. Für die Visualisierung durch die Prozessoren werden die Videodaten in den RGBA-Farbraum umgewandelt, mit gleichen Anteilen für die Drei Grundfarben Rot, Grün und Blau, die Alphakomponente wird auf 0 gesetzt.

### 4.1.1 Taktung des SAV Controllers

Die Takte im *SAV Controller* inklusive Bildverarbeitungspipeline werden durch die 27MHz der Kamera bereitgestellt.

Aus der Tabelle 4.1 ergeben sich die Taktfrequenzen von 6.75MHz, 54MHz und 108MHz, die von einem Xilinx Digital Clock Manager (DCM) [62] ausgehend von dem 27MHz Taktsignal der Kamera bereitgestellt werden. Der DCM verfügt über Takt-Ausgänge für

- Doppelte Taktfrequenz *CLK2X*, der die 54MHz generiert.
- Geteilte Taktfrequenz *CLKDV*, der durch den auf Vier eingestellten Parameter *CLKDV_DIVIDE* die Taktfrequenz von 6.75MHz erzeugt.
- Frei einstellbaren Taktfrequenz *CLKFX*, der durch Einstellung des Multiplizierers *CLKFX_MULTIPLY* (4) und Teilers *CLKFX_DIVIDE* (1) die 108MHz erzeugt.

| Modul | Benötigte Frequenz | Errechnete Frequenz | Kommentar |
|---|---|---|---|
| **Bildverarbeitungs-Pipeline** | 6.75MHz | 6.75MHz | Erzeugung der PWM |
| | 1x | 27MHz | Pixelfrequenz |
| | 2x | 54MHz | Doppelte Pixelfrequenz für Multizyklusdatenpfade |
| | 4x | 108MHz | Vierfache Pixelfrequenz für Multizyklusdatenpfade |
| **Receiver Controller** | 6.75MHz | 6.75MHz | Empfang von PWM-Daten |
| **VGA Controller** | 1x | 27MHz | Pixelfrequenz |
| **Geschwindigkeits-regler** | 54MHz | 54MHz | - |

Tabelle 4.1: Im *SAV Controller* verwendete Taktfrequenzen.

### 4.1.2 Wahl der Kameraperspektive und Berechnung der Projektiven Transformation

Die Hough-Transformation (HT) zur Fahrspurerkennung arbeitet in der Draufsicht. Da die Fahrspur durch die nach vorn ausgerichtete Kamera aufgenommen wird, erscheint die Fahrspur im Bild verzerrt. Deswegen wird sie durch eine Perspektivische Transformation (PT) in die Vogelperspektive, senkrecht von oben, gedreht und entzerrt (Abb. 4.2). Gleichzeitig wird ein Bezug der Pixel zum 2D-Koordinatensystem des Fahrzeugs hergestellt, das seinen Ursprung auf der Hinterachse in der Fahrzeugmitte hat.

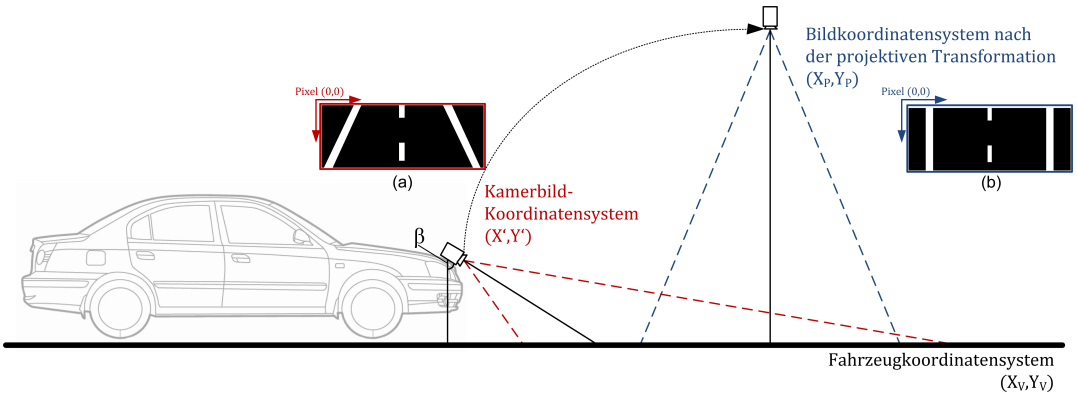



Abbildung 4.2: Eine projektive Transformation schwenkt das aufgenommene Bild der Kamera virtuell in die 2D Vogelperspektive. Bild: [34].

Die Neigung der Kamera ist in vertikaler Richtung einstellbar. Eine weniger geneigte und damit auf den Horizont ausgerichtete Kamera verfügt über einen größeren Sichtbereich, mit dem Nachteil der geringeren Pixel-Auflösung in Fahrzeugnähe [28].

Der Winkel der Kamera zum Boden wird für eine Erfassung der Fahrbahn in der Nähe des Fahrzeugs gewählt, ohne dabei die Vorderkante des Fahrzeugs zu erfassen, damit wird ein Sichtbereich von 95cm horizontal und 102cm vertikal aufgenommen (Abb. 4.3).

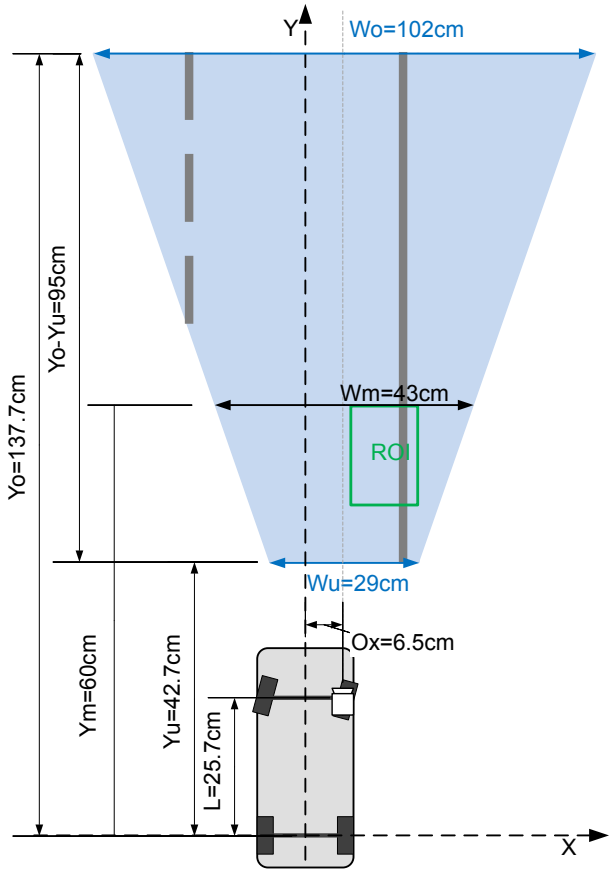
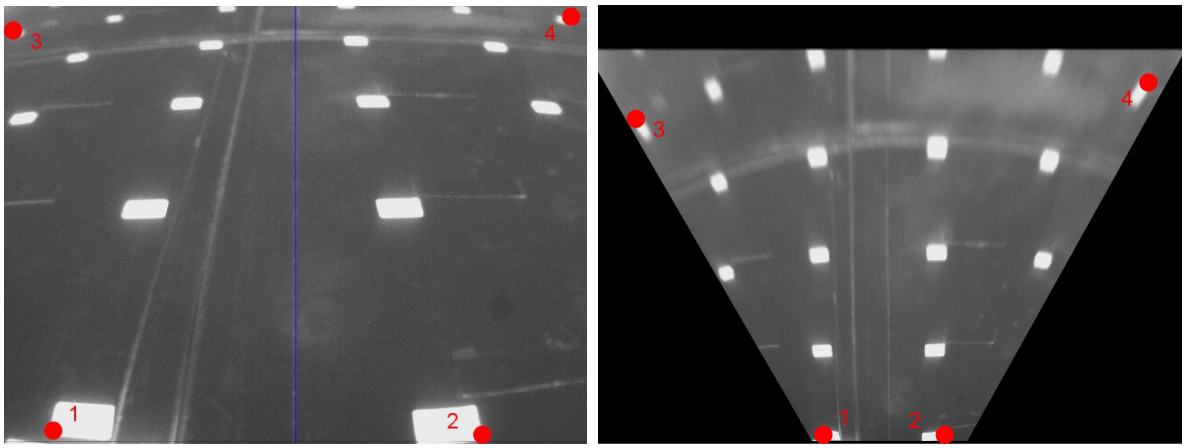



Abbildung 4.3: Kameraperspektive mit Sichtfeld und Entfernungen. Bild: [28].

Die PT wird mit einer Transformationsmatrix (Gleichung 4.1) durchgeführt, dessen Elemente nach dem Verfahren aus [26] bestimmt werden.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & 1 \end{pmatrix} * \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \tag{4.1}$$

Dazu wird eine Kalibrationsmatte verwendet, in der Vier Referenzpunkte gewählt werden, von denen die Fahrzeug-Koordinaten in cm und Pixel-Koordinaten im aufgenommenen Bild (Abb 4.4) ausgemessen werden.




(a) Kalibrationsmatte mit Zentrierungslinie und Referenzpunkten

(b) Kalibrationsmatte nach der PT mit den neuen Positionen der Referenzpunkte

Abbildung 4.4: Aufgenommene Kalibrationsmatte vor und nach der PT, durch die PT wird der untere Teil des Bildes gestaucht was zur Bildung eines Trapez führt.

Für eine Symmetrische Transformation sollten sich die Punkte auf derselben horizontalen Höhe befinden. Die gewählten Referenzpunkte zeigt Tabelle 4.2.

| **Punkt** | **Position im Bild (px)** | **Position im Kamera-Koordinatensystem (cm)** |
|---|---|---|
| 1 | $\begin{pmatrix} 50 \\ 472 \end{pmatrix}$ | $\begin{pmatrix} -11.7 \\ 1 \end{pmatrix}$ |
| 2 | $\begin{pmatrix} 526 \\ 472 \end{pmatrix}$ | $\begin{pmatrix} 8.7 \\ 1 \end{pmatrix}$ |
| 3 | $\begin{pmatrix} 7 \\ 20 \end{pmatrix}$ | $\begin{pmatrix} -46.4 \\ 73 \end{pmatrix}$ |
| 4 | $\begin{pmatrix} 620 \\ 20 \end{pmatrix}$ | $\begin{pmatrix} 43.1 \\ 73 \end{pmatrix}$ |

Tabelle 4.2: Koordinaten der Referenzpunkte (Abb. 4.4) in Pixel- und Kamera-Koordinaten. Der Ursprung des Kamera-Koordinatensystems liegt an der unteren Kante in der Bildmitte.

Das Ergebnis der perspektivischen Transformation zeigt Abbildung 4.5.

(a) Original (b) Binarisiert (c) Nach der PT

Abbildung 4.5: Straße im ursprünglichen Bild, sowie vor und nach der Perspektivischen Transformation (PT). Vor der PT laufen die Fahrspurmarkierungen am Horizont zusammen.

## 4.2 Implementierung des Framegrabbers mit dem Video DMA IP-Core

Zur Visualisierung der Fahrspurerkennung mit dem Dual-Core Prozessor, werden die Videodaten von der Kamera in den Arbeitsspeicher der CPUs kopiert und für die Ausgabe über VGA vom Arbeitsspeicher an die PL gegeben (Abb. 4.6). Dazu wird der AXI4 VDMA (Video Direct Memory Access) IP-Core [66] verwendet, der AXI4-Stream Video Geräte an die Memory Mapped I/O anbindet. Mit dem VDMA IP-Core werden serielle Bilddaten-Ströme an einer Adresse im Speicher abgelegt (Stream to Memory Map, S2MM) und Bilddaten-Ströme aus Daten im Speicher (Memory Map to Stream, MM2S) generiert.

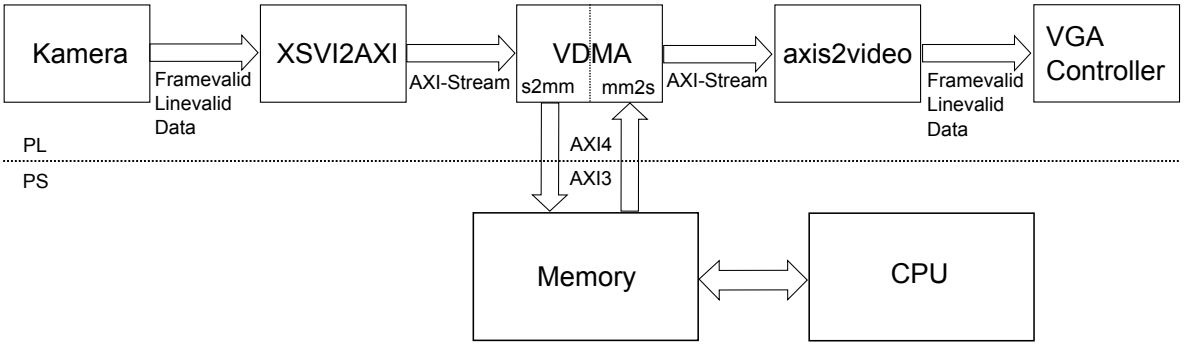



Abbildung 4.6: Die Videodaten der Kamera werden durch den VDMA-IP-Core in den Speicher geladen, wo sie von der CPU bearbeitet und über den VDMA-Core abgeholt und auf den VGA-Monitor ausgegeben werden. Die IP-Cores *xsvi2axi* und *axis2video* wandeln die Daten zwischen Kamera-Format und AXI4-Stream Format.

In der Vorbereitung wird der Kamera-Videostrom durch den *xsvi2axi* IP-Core in das AXI4-Stream Format umgewandelt. Um weiterhin die Videodaten mit dem VGA-Controller auf einen Monitor auszugeben, wurde der IP-Core *axis2video* entwickelt, der den Kamera-Datenstrom mit dem vom VGA-Controller erwartetem Zeitverhalten aus dem AXI4-Stream erstellt. Dabei arbeiten alle Videostrom-basierten Komponenten synchron zum Kamera-Takt (vgl. Abb. 4.1).

Im Unterschied zu den AXI-Varianten *AXI4* und *AXI4-Lite* gibt es bei *AXI4-Stream* keine Adressierung oder max. Größe der Burst-Datenpakete. Bei dem AXI4-Stream Protokoll handelt es sich um einen Unidirektionalen Kanal mit Datenflusssteuerung per Handshake, damit eignet sich das Protokoll für kontinuierliche Datenströme wie Netzwerk- oder Videodaten [69].

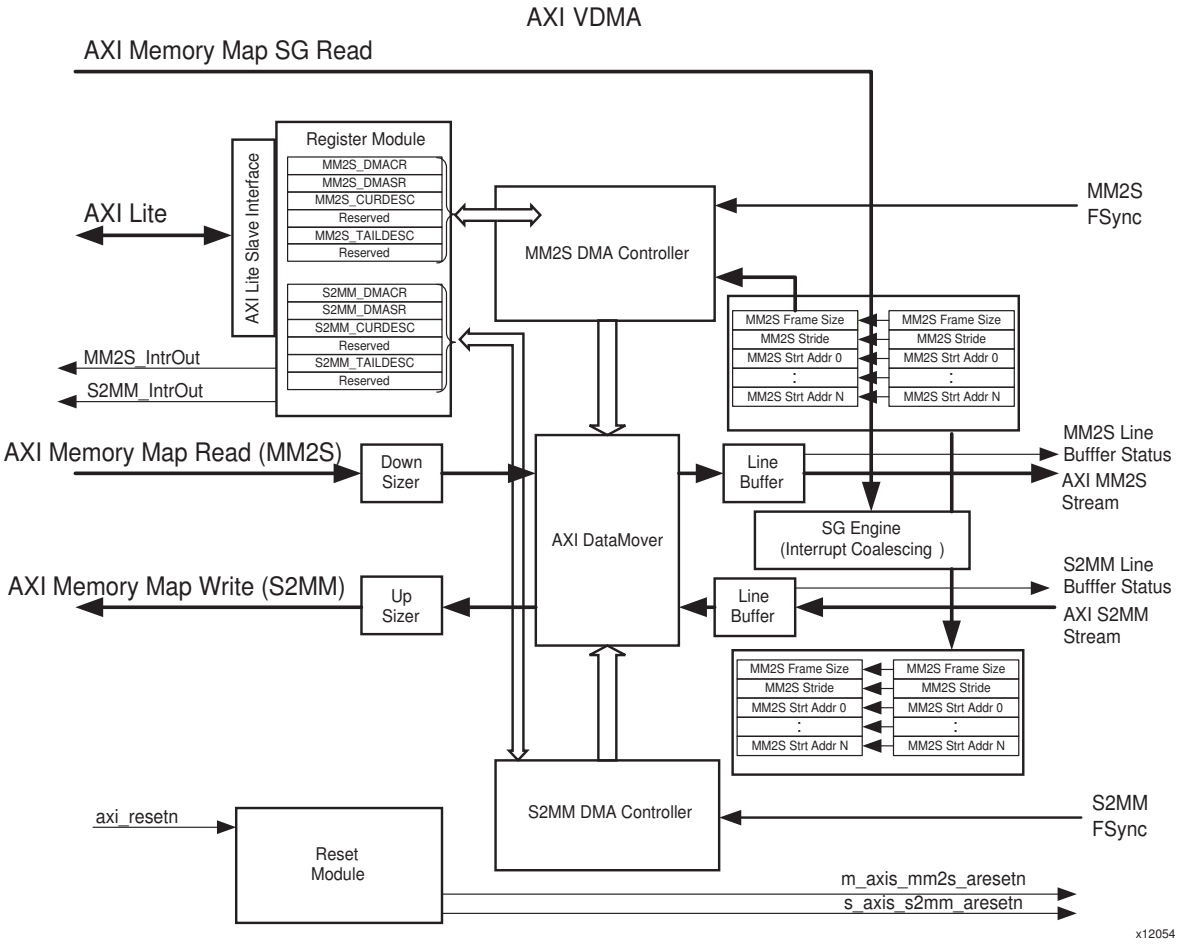



Abbildung 4.7: VDMA Block-Diagramm. Die Konfigurations-Register werden über das AXI4-Lite Interface gesetzt und ausgelesen, die Daten werden durch den Datamover zwischen den AXI4-Stream- und AXI4-Bussen kopiert. Bild: [69].

Um die Datenströme in den Memory Mapped Adressraum der ARM-Prozessoren zu kopieren, wird im VDMA IP-Core ein Datamover DMA IP-Core eingesetzt. Dieser enthält pro AXI4-Stream Lese- und Schreib Kanal einen DMA-Controller zum Kopieren der Daten (Abb. 4.7).

Der VDMA IP-Core unterstützt dabei eine Farbtiefe von 1024 Bit pro Pixel, mit einer max. Auflösung von 65535x8191 Pixel. Die Taktfrequenz der AXI4-Stream Kanäle ist von dem Takt der Memory Mapped AXI4 und AXI4-Lite Bussen unabhängig. Es können bis zu 32 Speicheradressen gelesen oder geschrieben werden, im „Park"-Modus wird das selbe Bild wiederholend übertragen. Zum Schreiben auf eine AXI-Adresse arbeitet das VDMA-Modul als AXI-Master. Das VDMA-Modul arbeitet entweder im „Register Direct Mode", bei dem der Prozessor alle Operationen des Moduls über ein AXI4-Lite Interface steuert, oder im „Scatter Gather Mode", bei der die Operationen nach einer im Speicher vorgegebene Liste abgearbeitet werden [66]. Die Konfiguration des VDMA-Moduls in dieser Arbeit im „Register Direct Mode" zeigt Tabelle 4.3.

| Option | Wert | Beschreibung |
|---|---|---|
| **C_NUM_FSTORES** | 1 | Anzahl Speicheradressen an die Bilder geschrieben oder von der Bilder gelesen werden. Es wird eine je Kanal verwendet. |
| **C_USE_FSYNC** | 1 | S2MM und MM2SS Kanal verwenden beide ein externes *fsync*-Signal zur Bild-Synchronisation. |
| **C_FLUSH_ON_FSYNC** | 1 | S2MM und MM2SS Kanal gehen bei einem Fehler für das nächste Bild in den Ausgangszustand zurück, anstatt anzuhalten. |
| **C_M_AXI_MM2S_DATA_WIDTH**<br>**C_M_AXIS_MM2S_TDATA_WIDTH**<br>**C_M_AXI_S2MM_DATA_WIDTH**<br>**C_M_AXIS_S2MM_TDATA_WIDTH** | 32 | 4 Byte Farbtiefe für beide Kanäle. |
| **C_MM2S_MAX_BURST_LENGTH**<br>**C_S2MM_MAX_BURST_LENGTH** | 32 | 32 Byte Burst-Datenpakete pro Adressierung |
| **C_MM2S_LINEBUFFER_DEPTH**<br>**C_S2MM_LINEBUFFER_DEPTH** | 4096 | 4kB Zwischenspeicher je Kanal. |
| **C_MM2S_LINEBUFFER_THRESH**<br>**C_S2MM_LINEBUFFER_THRESH** | 4064 | Wert, bei dem der Zwischenspeicher als voll angesehen wird, vielfaches der Farbtiefe. |
| **C_MM2S_SOF_ENABLE**<br>**C_S2MM_SOF_ENABLE** | 1 | Der Start eines Bildes wird im AXI4-Stream übertragen. |

Tabelle 4.3: Hier verwendete VDMA Konfiguration für S2MM und MM2S-Kanäle.

Durch Setzen der Einstellung *C_USE_FSYNC=1* wartet jeder VDMA-Kanal für den Start eines Transfers auf eine asynchrone Flanke an seinem *fsync*-Eingang. Die Einstellung *C_FLUSH_ON_FSYNC* wird deaktiviert, da die Bilder nur für die Anzeige an den Prozessor

übertragen werden und einzelne Fehler bei der Übertragung deswegen für den Betrieb unkritisch sind. Da die Anwendung auf dem Prozessor im RGBA-Farbraum arbeitet, wird eine 4-Byte Farbtiefe eingestellt. Die Zwischenspeicher *LINEBUFFER_DEPTH* werden so eingestellt, daß sie eine Bildzeile enthalten, bei einer Farbtiefe von 4 Byte und einer Bildbreite von 640 Pixel sind dies $640 * 4B = 2560B$ (4096 Bytes als die nächst größere Zweierpotenz). Die Schwellwerte *LINEBUFFER_THRESH* werden 32 Bytes kleiner als die Zwischenspeicher-Größe gesetzt. Die Burst-Größe wird auf 32 Datenpakete eingestellt, was das Doppelte der 16 Datenpakete in der AXI3 Spezifikation ist, die das PS implementiert. Da die High Performance Ports des Zynq-7000 mit der doppelten Bitbreite (64 Bit) arbeiten, werden durch den AXI-Interconnect am HP-Interfaces (vgl Abb. 4.1) die 32 Bit RGBA-Daten zu 32 Bursts in 64 Bit Daten zu 16 Bursts umgesetzt (Split transactions) [69][40].

Für die Anbindung der Kamera an den VDMA IP-Core, wird das parallele Interface der Kamera in das AXI4-Stream Protokoll konvertiert (vgl. Abb. 4.1).

### 4.2.1 Parallele Interface der Kamera mit Synchronisations-Signalen

Die Videology 24B7525A CMOS-Kamera sendet mit einem 27MHz Pixeltakt ein 10-Bit Videostrom mit Line- und Framevalid Synchronisations-Signalen und verfügt bei der Konfiguration aus einer Vorarbeit [47] mit 640x480 Pixel bei 60 Bildern pro Sekunde (16.67ms) über folgendes Zeitverhalten (Abb. 4.8):

- Bei Start eines Bildes liefert Framevalid 201 Takte vor Linevalid einen High-Pegel.
- Zwischen zwei Bildzeilen ist Linevalid 224 Takte Low, damit beträgt die volle Länge einer Zeile $640 + 224 = 864$ Pixel.
- Zum Ende des Bildes liefert Framevalid 23 Takte nach Linevalid einen Low-Pegel.
- Zwischen zwei Bildern ist Framevalid 36292 Takte (ca. 42 Zeilen) Low.

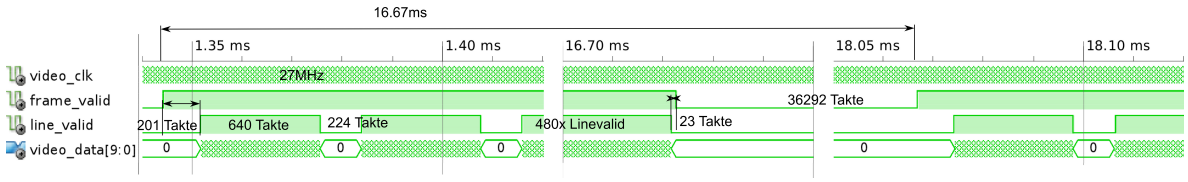



Abbildung 4.8: Signal-Zeitverhalten der Videology 24B7525A CMOS-Kamera.

### 4.2.2 AXI-Stream Protokoll für Video-Signale

Die von Xilinx für die Verwendung mit Video IP-Cores definierte AXI4-Stream Interpretation zeigt Tabelle 4.4.

| Signal | Funktion | Richtung | Bitbreite |
|---|---|---|---|
| tdata | Videodaten, bei bedarf am MSB mit Nullbytes aufgefüllt | out | 8,16..1024 |
| tvalid | Signal in tdata gültig | out | 1 |
| tready | Empfänger bereit zum empfangen von Daten | in | 1 |
| tuser | Start of Frame (SOF) | out | 1 |
| last | End of Line (EOL) | out | 1 |

Tabelle 4.4: AXI4-Stream Signale für Video-Anwendungen aus Sicht des Senders (z.B. xsvi2axi zu VDMA) [69].

Zusätzlich zu den AXI4-Stream Signalen sind der Takt *ACLK*, der Reset *ARESETN* und die optionale Frame-Synchronisation *fsync* (Kapitel 4.2) definiert. Signale arbeiten bei steigender Taktflanke. Takte im AXI4-Stream sind von den Takten in den anderen AXI4-Varianten AXI4 und AXI4-Lite unabhängig.

Die Datenflusssteuerung wird durch das Handshake der Signale *tready* und *tvalid* erzeugt. Die Daten in *tdata* sind nur gültig, wenn der Sender gültige Daten mit *tvalid* und der Empfänger die Bereitschaft Daten zu empfangen mit *tready* signalisiert (Abb. 4.9).

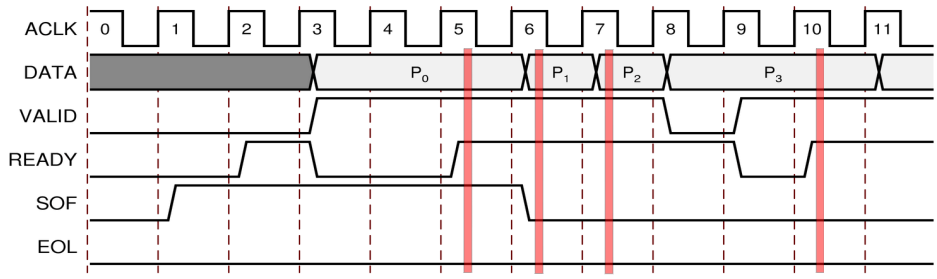



Abbildung 4.9: Datenflusssteuerung mit Handshake im AXI4-Stream Protokoll: Die Daten sind nur gültig wenn *tvalid* und *tready* während der steigenden Taktflanke high sind (rote Balken). Bild: [69].

Das Start of Frame (SOF)-Signal synchronisiert die IP-Cores untereinander und ist beim ersten Pixel jeden Bildes gesetzt. Das End of Line (EOL)-Signal zur Erkennung von Datenverlust wird beim letzten Pixel jeder Bildzeile gesendet. Die Videoströme sind Daten-unabhängig,

ob z.B. ein RGB- oder YUV-Format vorhanden ist, wird vom Anwender festgelegt. Blanking-Intervalle, die u.a. von Monitoren verwendet werden, sind im AXI4-Stream Protokoll von Xilinx nicht enthalten.

### 4.2.3 Umwandlung des Kamera-Signals in AXI-Stream

Für die Anbindung der Kamera an das VDMA Modul wurde ein Xilinx *xsvi2axi* IP-Core modifiziert, welcher Videoströme nach dem früheren Xilinx Streaming Video Interface (XSVI) an das neue AXI4-Stream Protokoll für Video-Anwendungen anbindet [64]. Der IP-Core verwendet einen 36kB FIFO zur Zwischenspeicherung von bis zu Vierzehn 640 Pixel langen Bildzeilen bei einer Farbtiefe von 4 Byte. Die folgenden Änderungen wurden an dem *xsvi2axi* IP-Core durchgeführt:

- Der Kamera-Eingang *framevalid* ersetzt den XSVI-Eingang *fsync_in*.
- Der Kamera-Eingang *linevalid* ersetzt den XSVI-Eingang *active_video*.
- Der AXI4-Stream Ausgang *tuser* wird zur Signalisierung des Start of Frames hinzugefügt.
- Der Ausgang *fsync* zur Frame-Synchronisation mit dem VDMA-Modul wird hinzugefügt.

Der entwickelte Logik des IP-Cores zeigt Abb 4.10. Der FIFO speichert die Videodaten *tuser* (Erstes Pixel des Bildes, SOF), *tdata* und *tlast* (Letztes Pixel der Zeile, EOL), bis sie durch ein gesetztes *tready* vom VDMA-Modul abgeholt werden. Die Gültigkeit der Daten im FIFO wird mit *tvalid* angezeigt, wenn dieser mindestens ein Pixel enthält. Das EOL-Signal wird mit einem Puls-Shorter [45] durch die fallende Flanke von *linevalid* generiert. Mit dem *fsync*-Impuls wird der VDMA IP-Core für den Empfang eines neuen Bildes gestartet und wird für einen stabilen Betrieb des VDMA IP-Cores mit einem Zähler drei Takte lang gehalten. Der Zähler wird durch die *framevalid*-Flanke geladen und wird auch zur Freigabe des SOF-Signal benutzt, das im Gegensatz zum EOL nur in der ersten Bildzeile gesetzt wird, da es nur die ersten Pixel des Bildes anzeigt. Die Simulation (Abb. 4.11) zeigt das Verhalten des IP-Cores.

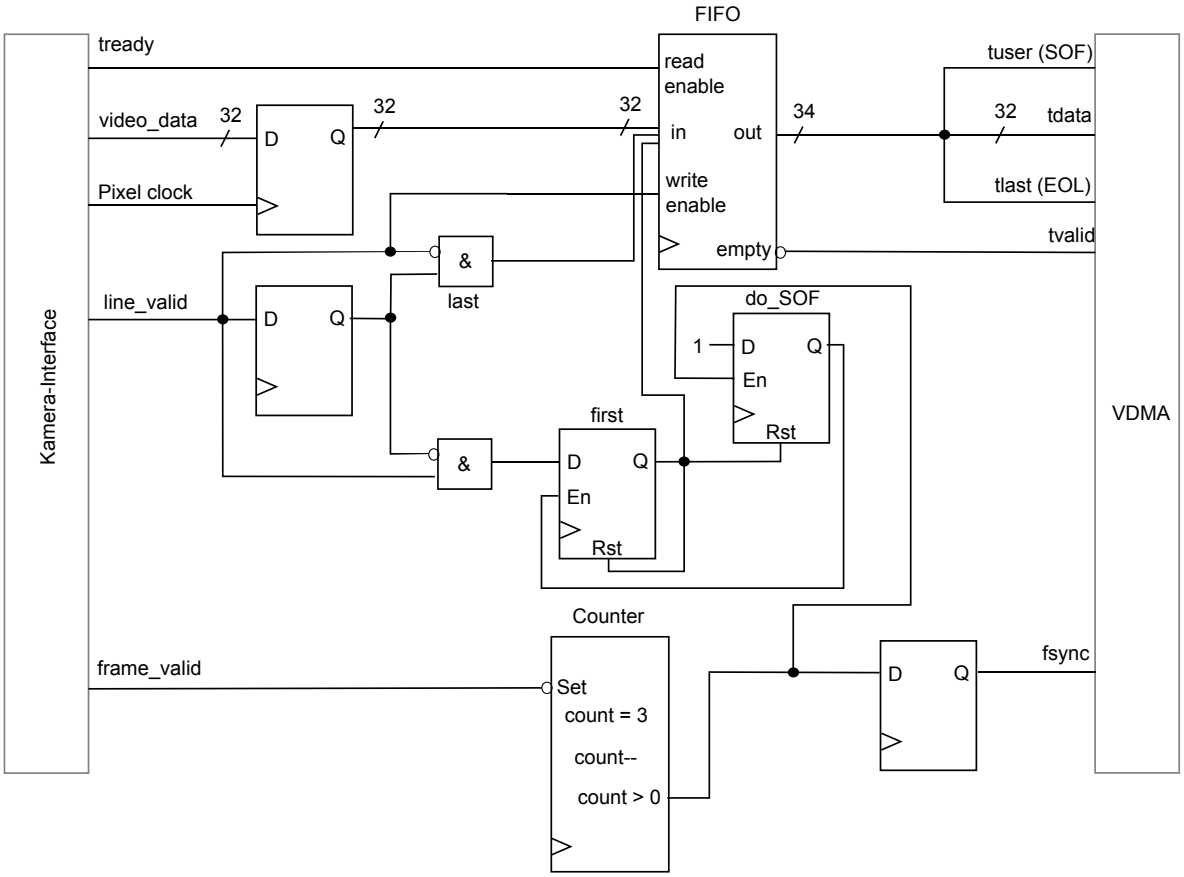



Abbildung 4.10: RTL-Übersicht des *xsvi2axi*-Moduls mit dem der Kamera-Videostrom in AXI4-Stream überführt wird.

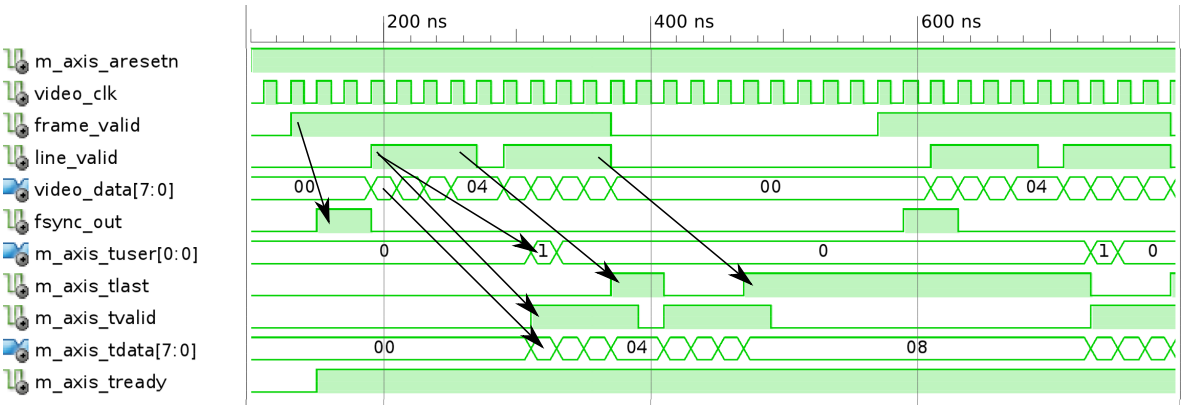



Abbildung 4.11: Simulationsergebnis des *xsvi2axi* IP-Cores zur Umwandlung des Kamera-Videostroms in das AXI4-Stream Protokoll mit einem 4x2 Pixel großen Bild und 2 Takten langen *fsync*-Impuls.

### 4.2.4 Umwandlung der AXI4-Stream Videodaten in Line- und Framevalid Synchronisations-Signale

Der VGA-Controller wurde entwickelt, um das Kamera-Signal direkt auf einem VGA-Monitor auszugeben. Für die Darstellung der von der CPU modifizierten Bilder durch den VGA-Controller, werden mit dem *axis2video* IP-Core die Line- und Framevalid Signale aus dem AXI4-Stream Signal erzeugt. Dabei wird für den VGA-Controller das ursprüngliche im AXI4-Stream nicht enthaltene Zeitverhalten der Kamera mit Blanking-Intervallen (Kapitel 4.2.1) wiederhergestellt. Der *axis2video* IP-Core verwendet dafür einen kombinierten Mealy-Moore-Zustandsautomaten (Abb 4.12) mit dem

- Moore-Ausgang *framevalid*, sowie den
- Mealy-Ausgängen *fsync*, *tready* und *linevalid*.

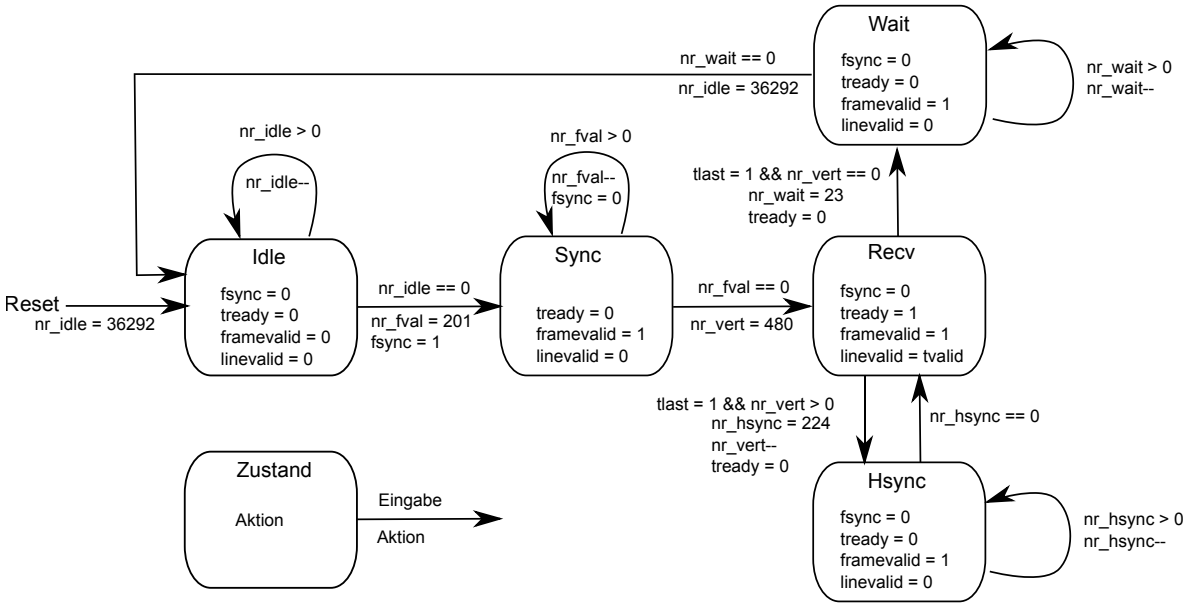



Abbildung 4.12: Zustandsdiagramm des *axis2video*-Moduls mit dem Signale und Timing der Kamera, aus dem AXI4-Stream Videosignal wiederhergestellt werden.

Das Zeitverhalten wird durch Zähler in den Zuständen erreicht, die die Signale *fsync* und *tready* an das VDMA-Modul senden. Das Signal *fsync* ist das Startsignal an das VDMA-Modul, mit dem Senden der Videodaten zu beginnen, das Signal wird zur Erzeugung der vertikalen Blank-Zeiten verzögert. Die horizontalen Blanking-Zeiten werden durch Rücksetzen des

Signal *tready* erreicht, da das VDMA-Modul nur bei gesetztem *tready* die Videodaten senden darf:

- Nach Ablauf des Zählers „idle“ in Zustand *Idle* wird der *fsync*-Impuls für einen Takt gesendet und der *Framevalid*-Ausgang gesetzt. Das VDMA Modul wartet daraufhin auf ein gesetztes *tready*, um mit dem Senden der Daten zu beginnen.
- Der Zähler „fval“ In Zustand *Sync* zählt die Anzahl Takte, die *Framevalid* vor dem ersten *Linevalid* gesetzt sein soll.
- Im Zustand *Recv* wird mit gesetztem *tready* eine Bildzeile vom VDMA empfangen und der *Linevalid*-Ausgang gesetzt. Die empfangenen Zeilen wird mit dem Zähler „vert“ gezählt.
- Das Ende einer Bildzeile wird vom VDMA Modul mit *tlast* signalisiert. Sind alle Zeilen des Bildes empfangen, wird in den Zustand *Wait* gewechselt, ansonsten in den Zustand *Hsync*.
- Im Zustand *Hsync* wird mit dem Zähler „hsync“ die Blanking-Zeit zwischen zwei Bildzeilen erzeugt. Durch Rücksetzen des Signals *tready* wird der Empfang der Videodaten vom VDMA verzögert. Da in diesem Zustand keine Daten gesendet werden, ist *Linevalid* ebenfalls rückgesetzt.
- Der Zähler „wait“ im Zustand *Wait* zählt die Anzahl Takte, die *Framevalid* nach der Aufhebung von *Linevalid* gesetzt sein soll.
- Die Blanking-Zeit zwischen zwei Bildern wird mit dem Zähler „idle“ im Zustand *Idle* durch Verzögerung des *fsync*-Impuls erreicht. *Framevalid* ist in diesem Zustand nicht gesetzt.

Der Logik des IP-Cores zeigt Abb 4.13.

Zur Verifikation des IP-Cores wurde eine Simulation (Abb. 4.14) mit angeschlossenem *VGA-Controller* durchgeführt.

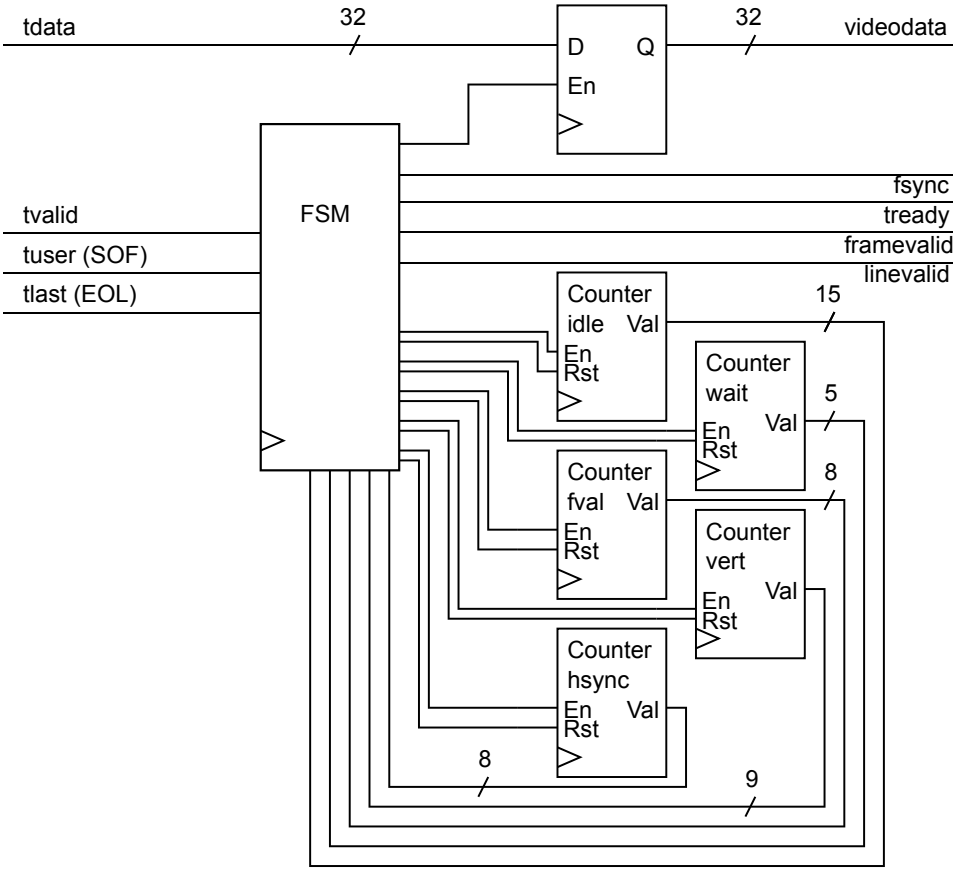



Abbildung 4.13: RTL-Übersicht des *axis2video*-Moduls, mit dem das Kamera-Signal mit Zeitverhalten aus dem AXI4-Stream wiederhergestellt wird.

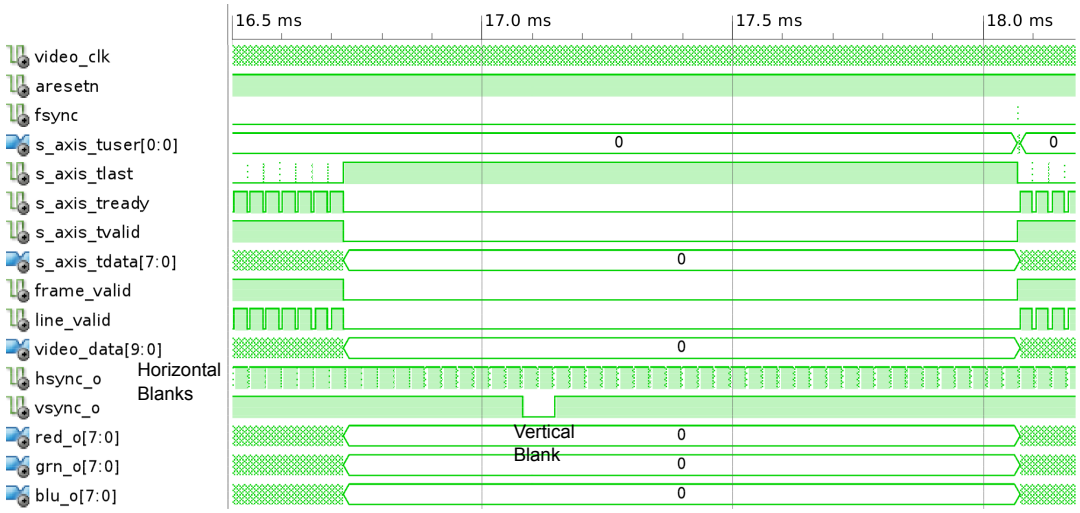



Abbildung 4.14: Simulationsergebnis des *axis2video* IP-Cores zur Erzeugung des Videostroms für den VGA-Controller aus dem AXI4-Stream Protokoll. Die erforderlichen Wartezeiten werden durch Umschalten der Signale *fsync* und *tready* erreicht.

## 4.3 Anbindung der Fahrspurerkennung an die AXI Bus Interfaces

Die Fahrspurerkennung ist durch Konfigurations- und Statusregister mit dem Prozessorsystem verbunden. Da der *SAV Controller* unabhängig vom Bussystem ist, wurde im bisherigen Microblaze/PLB-Fahrzeug ein *SAV Connector* eingesetzt, der zweiunddreißig 32-Bit Softwareregister zur Verfügung stellt. Aufgrund des gestiegenen Datenaustausches zwischen PS und PL wurde die Registeranzahl gegenüber den Vorarbeiten [34][28] verdoppelt und an die Zynq-7000 Architektur angepasst. Der PLB-*SAV Connector* wird damit durch zwei „AXI Connectors“ ersetzt, die je zweiunddreißig 32-Bit Softwareregister zur Verfügung stellen.

Mit dem *AXI Configuration Connector* (sav_axi_cfg) werden Parameter und Einstellungen (Tabelle 4.5) am *SAV Controller* vorgenommen. Auf die Softwareregister des *AXI-Configurations Connector* kann lesend und schreibend zugegriffen werden. Zustände und Berechnungsergebnisse werden von den Softwareregistern (Tabelle 4.6) des *AXI Status Connector* (sav_axi_reader) gelesen, auf die nur lesend zugegriffen werden kann.

| **Wert** | **Format** | **Einheit** | **Beschreibung** |
|---|---|---|---|
| roi_x | U(12.0) | cm | Horizontale Position der ROI bei aktivierter dynamischer ROI |
| phi_id | U(6.0) | - | ID des Hough Transformators |
| phi | S(9.0) | Grad | Winkel Φ der HT |
| r | S(11.0) | px | Lotlänge $r$ der HT |
| ht_max_val | U(10.0) | - | Maxima Wert der HT |
| count | U(12.0) | px | Anzahl Vordergrundpixel in der ROI |
| ht_result_count | U(32.0) | - | Zähler für die Anzahl an HT-Ergebnissen |
| v | S(9.0) | cm/s | Ist-Geschwindigkeit des Fahrzeugs |
| roi_state | U(3.0) | - | Aktueller Zustand der dynamischen ROI |
| ht_invalid | Bool | - | Gültigkeit des HT-Ergebnis |
| version | U(16.0) | - | Versionsnummer des Pathfinder-Moduls |

Tabelle 4.6: Berechnungsergebnisse des *SAV Controller* IP-Cores mit Angabe des Q-Formats der Festkomma-Werte, für die Aufteilung der Register siehe Tabelle B.3 in Anhang B.1

| Parameter | Format | Einheit | Beschreibung |
|---|---|---|---|
| roi_x_pos | U(10.0) | cm | Horizontale Position der ROI wenn dynamische ROI deaktiviert ist |
| roi_y_pos | U(9.0) | cm | Vertikale Position der ROI |
| roi_width | U(10.0) | px | Breite der ROI |
| roi_height | U(10.0) | px | Höhe der ROI |
| v_soll | S(10.0) | cm/s | Sollgeschwindigkeit des Fahrzeugs |
| regler_param1 | S(13.8) | - | Parameter 1 der Geschwindigkeitsregelung |
| regler_param2 | S(13.8) | - | Parameter 2 der Geschwindigkeitsregelung |
| threshold | U(8.0) | - | Schwellwert der Binarisierung wenn adaptive Binarisierung deaktiviert ist |
| pf_alpha | S(12.11) | - | Manueller Stellwinkel für die Lenkung |
| control | 32 Bit | - | Kontrollregister (Register B.1 im Anhang) |
| pwm_manual | U(16.0) | - | Manuelle PWM für den Antrieb unter Umgehung des Geschwindigkeitsreglers |
| pwm_override | Bool | - | Manuelle PWM anstatt Geschwindigkeitsregelung |
| pt_param_b11 | S(21.12) | - | Parameter B11 der Perspektiven Transformationsmatrix |
| pt_param_b12 | S(21.12) | - | Parameter B12 der Perspektiven Transformationsmatrix |
| pt_param_b13 | S(21.12) | - | Parameter B13 der Perspektiven Transformationsmatrix |
| pt_param_b21 | S(21.12) | - | Parameter B11 der Perspektiven Transformationsmatrix |
| pt_param_b22 | S(21.12) | - | Parameter B22 der Perspektiven Transformationsmatrix |
| pt_param_b23 | S(21.12) | - | Parameter B23 der Perspektiven Transformationsmatrix |
| pt_param_b31 | S(21.12) | - | Parameter B31 der Perspektiven Transformationsmatrix |
| pt_param_b32 | S(21.12) | - | Parameter B32 der Perspektiven Transformationsmatrix |
| ht_valid_threshold | U(10.0) | - | Grenzwert für gültige HT-Ergebnisse |
| roi_limit_factor | U(16.14) | - | Linke und Rechte horizontale Begrenzung der dynamischen ROI. Wird für die Begrenzung mit der Vertikalen Position der ROI multipliziert. |
| max_pwm | U(25.0) | - | PWM Begrenzung des Geschwindigkeitsreglers |

Tabelle 4.5: Parameter und Einstellungen des *SAV Controller* IP-Cores mit Angabe des Q-Formats der Festkomma-Werte, für die Aufteilung der Register siehe Tabelle B.1 im Anhang.

Die beiden „AXI Connectors“ verwenden den Logilink AXI4-Lite IP Interface (IPIF) IP-Core [65] für die Implementierung der Register. Auf die 32-Bit Register wird im Memory Mapped I/O mit $Basisadresse + Registernummer * 4$ zugegriffen.

### 4.3.1 Takt-Synchronisation mit FIFOs

Die Register des mit 100MHz getakteten *AXI Configuration Connector* und *AXI Status Connector* sind über FIFOs mit dem 27MHz getakteten *SAV Controller* verbunden, da es sonst aufgrund der unterschiedlichen Frequenzen zu instabilen Registerinhalten [33] kommen kann. Aufgrund der nicht synchronisierten Takte können fehlerhafte Werte auftreten, wenn ein FlipFlop die Daten eines asynchron getakteten FlipFlops liest, solange sein Zustand nicht stabil ist (Clock Domain Crossing). Bei Tests ohne FIFO, bei denen ein Zählwert gelesen wurde, der sechzig Mal in der Sekunde erhöht wird, kam es dadurch in einem, von 100 Millionen Lesevorgängen, zu einem Fehlerhaften Wert. Die für diese Aufgabe geeigneten LogiCore FIFO IP-Cores [67] wurden mit dem Xilinx Core Generator erzeugt [29] und anschließend Simuliert (Abb. 4.15). Pro Softwareregister wird ein 32-Bit FIFO mit einer Größe von 16 Werten instanziiert.

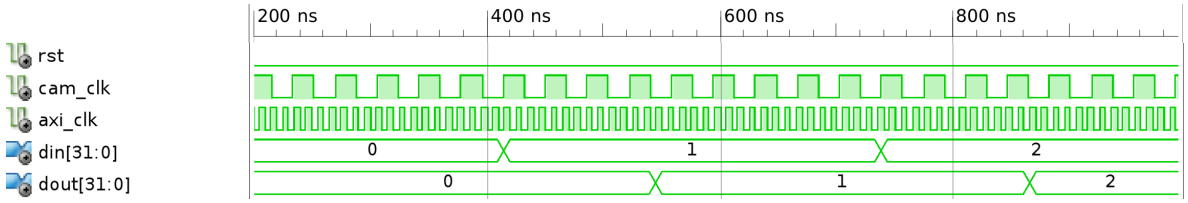



Abbildung 4.15: Clock Domain Crossing FIFOs in der Simulation: Die Daten werden mit dem 27MHz Kamera-Takt in den FIFO geschrieben und mit dem 100MHz AXI-Takt gelesen. Durch die FIFOs werden die Daten um ein paar Takte verzögert.

Der *Full*-Ausgang des FIFOs wird nicht verwendet, da neue Daten mit der Bildfrequenz nur alle 16.67ms anliegen und der Inhalt der FIFOs bereits im folgenden Takt in FlipFlops übertragen wird. Bei einem leerem FIFO wird der letzte gültige Wert ausgegeben, weswegen der *Empty*-Ausgang ebenfalls nicht ausgewertet wird. Die FlipFlops werden im Block-RAM des FPGAs implementiert und verwenden zum Austausch der Adressen unter den beiden Takten einen Gray-Code, bei dem sich nur ein Bit pro Wert ändert (Abb 4.16).

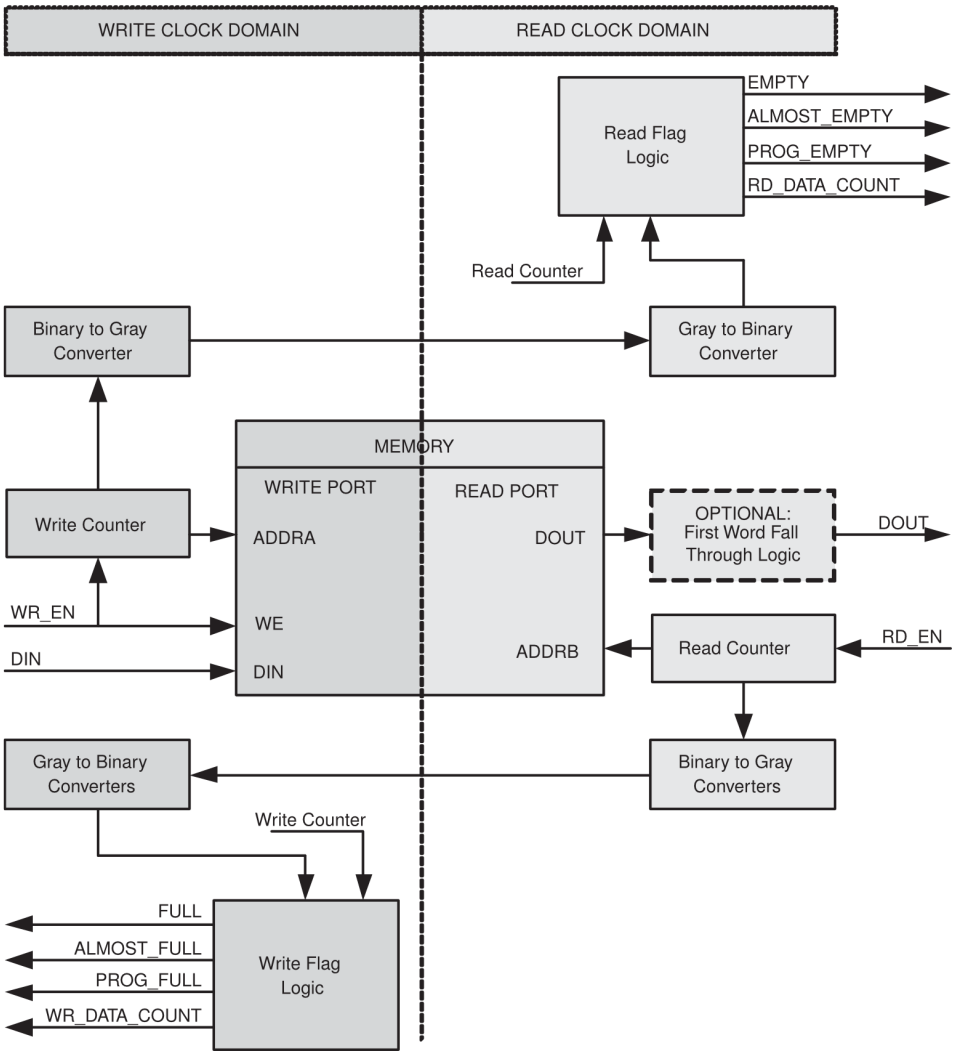



Abbildung 4.16: FIFO zur Synchronisation von Daten mit asynchronen Takten. Bild: [67]

## 4.4 Ansteuerung der Aktoren mit PWM-Signalen

An den HPEC-Fahrzeugen wird mit einem Motor der Vortrieb erzeugt, und über einen Servo mit der Stellung der Lenkachse der Lenkeinschlag gesteuert [34]. Angesteuert werden die Motoren über eine Pulsweitenmodulation (PWM) [23] mit einer Periodendauer von 10ms bis 20ms und einer Impulslänge mit einem High-Pegel von ca. 1ms bis 2ms (vgl Abb. 4.17). Der Pegel der PWM beträgt 3.3V.

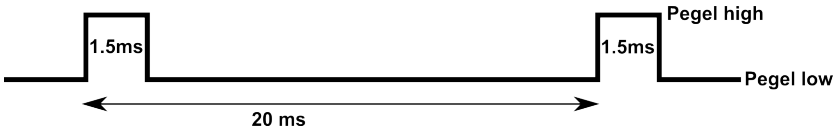



Abbildung 4.17: PWM zur Ansteuerung der Motoren für Antrieb und Lenkung: Ein High-Pegel mit einer Impulslänge von 1ms-2ms (Hier: 1,5ms) wird alle 20ms wiederholt.

Die PWM Komponenten und der Geschwindigkeitsregler des vorhergehenden Fahrzeugs wurden übernommen und angepasst. Mit dem Geschwindigkeitsregler wird der Einfluss der Akku-Spannung auf die Geschwindigkeit reduziert. Bei dem Fahrtensteller für den Antrieb wurden folgende Pulsbreiten für den High-Pegel mit dem Oszilloskop gemessen:

- Mittelstellung (Motor aus) bei 1.4ms,
- maximale Vorwärts-Geschwindigkeit bei 1ms
- und die maximale Rückwärts-Geschwindigkeit bei 1.75ms.

Gegenüber dem vorherigen Fahrzeug, für das der PI-Geschwindigkeitsregler [51] entworfen wurde, wurden zur Anpassung die Pulsbreiten im RTL-Modell invertiert und die Grenzen angepasst.

Um bei ausgeschaltetem Motor ein zu großes Aufsummieren des I-Anteils zu vermeiden, wurde dieser begrenzt, die Begrenzung *max_pwm* ist durch Softwareregister 31 (vgl. Kapitel 4.3) einstellbar. Da der PWM Generator mit 54MHz arbeitet, berechnet sich die Begrenzung in ms durch $max\_pwm/54000$.

Für die Umgehung des Geschwindigkeitsreglers bei Testfahrten wird das Softwareregister 8 direkt an den PWM-Generator angeschlossen. Mit Bit 31 des Registers (Tabelle 4.5) wird der Geschwindigkeitsregler über einen Multiplexer umgangen. Wird Bit 31 im 8. Register gesetzt, wird der PWM-Wert *pwm_manual* aus den unteren 16 Bits des Registers gelesen. Die resultierende Pulsbreite des High-Pegel in ms berechnet sich mit $1.9 - pwm\_manual/54000$.

Der Servo für den Lenkeinschlag hat bei Mittelstellung eine Pulsbreite von 1.4ms und ist damit um 0.1ms vom vorherigen Fahrzeug verschoben. Dies wurde im System Generator Modell angepasst.

# 5 Architektur und Implementierung der Software im SMP-System

Die Software (Abb 5.1) zur Fahrspurführung mit Parametrisierung und Visualisierung sowie zum Datalogging wird als Thread unter einem SMP Linux-Betriebssystem ausgeführt. Dazu wird die Installation und Konfiguration des Betriebssystems, sowie die Memory Mapped Kommunikation mit der Peripherie und IP-Cores dokumentiert. Zur Visualisierung der Fahrspurführung- und Erkennung wurde der VDMA IP-Core und das Multimedia-Framework GStreamer eingerichtet.

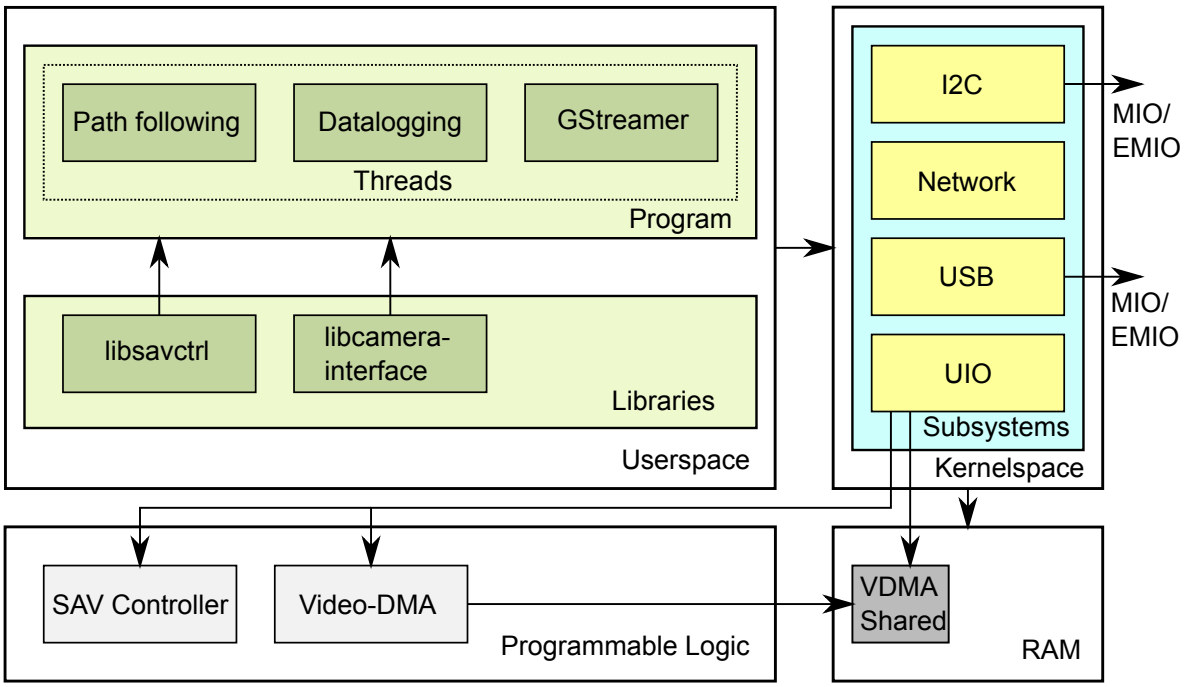



Abbildung 5.1: Die Struktur des SAV unter Linux mit SMP.

Das entwickelte Programm im Userspace des SMP-Linux Betriebssystems enthält die folgenden Threads:

- **Path following**: Liest das Ergebnis der Fahrspur-Approximation aus dem *SAV Controller* und berechnet daraus mit der Spurführung den Lenkwinkel (Kapitel 5.3.3).

- **Datalogging**: Sendet Mess- und Stellgrößen aus dem Fahrbetrieb und empfängt Konfigurationsdaten über Netzwerk (Kapitel 5.3.4). Je ein Thread zum Senden und Empfangen.
- **GStreamer**: Visualisiert das Fahrspurführungs-System mit dem Plugin-basierten GStreamer-Framework. Die Bilder werden per VGA ausgegeben oder über Netzwerk gesendet (Kapitel 5.3.5). Das Rendern des Bildes wird in dem separaten Thread „Result Illustration“ vorgenommen.

Aus dem bestehenden Kamera-Interface [47] zur Parametrisierung der Kamera über I2C wurde die Bibliothek *libcamerainterface* erzeugt. Die Komponenten des Microblaze SAV-Fahrzeug [34][28] zum Datalogging und zur Konfiguration des *SAV Controllers* sind in der Bibliothek *libsavctrl* enthalten. Beide dynamische Bibliotheken werden zur Laufzeit zum Programm gelinkt.

Für die Memory Mapped Kommunikation mit der PL-Peripherie und dem mit dem Video-DMA geteilten Bereich im Arbeitsspeicher wird das UIO-Subsystem des Linux-Kernels verwendet, das die physischen Adressen in den virtuellen Adressraum der Anwendung abbildet (Kapitel 5.2.1). Die Treiber für das I2C-Interface (Kapitel 5.2.3) und den USB W-LAN Adapter werden durch die jeweiligen Subsysteme des Kernels bereitgestellt.

## 5.1 Konfiguration des Linux-Kernels

Die Software auf dem Zynq-7000 läuft aufgrund der Verfügbarkeit von Treibern, wie Gigabit-Ethernet und SW-Bibliotheken (GStreamer) unter einem Linux-Betriebssystem. Der First- und Second-Stage Bootloader (Kapitel 3.2, Abb. 3.5) für den Startvorgang, sowie die Hardware-Konfigurationsdatei wurden dem Basis-Image [18] entnommen.

Die Hardware-Konfigurationsdatei „Device-Tree“ [70] enthält die Plattform-spezifischen Attribute der Peripherie wie Adressen, Interrupts und Treiber-Kompatibilität in einer Datenstruktur. Zweck der Device-Tree-Datei ist die Verwendung des Linux-Kernels auf mehreren Plattformen derselben Architektur ohne neue Kompilierung. Die Device-Tree-Datei wird als Textdatei (.dts) vom Entwickler für eine Plattform erstellt und mit dem Device-Tree-Compiler (dtc, Listing 5.1) in eine Binär-Datei (.dtb) übersetzt. Die .dtb wird vom Bootloader an eine feste Adresse im Speicher geladen und von dort vom Linux-Kernel beim Startvorgang für die Initialisierung der Peripherie verwendet.

Listing 5.1: Übersetzungsvorgang der Device-Tree-Datei in eine Binär-Datei. Der Device-Tree-Compiler (dtc) ist den Linux Kernel-Quellen beigelegt.

```
> linux-digilent/scripts/dtc/dtc -I dts -O dtb -o devicetree.dtb devicetree.dts
```

Die Device-Tree-Datei besteht u.a. aus den Knoten [16]

- **chosen**: Übergeben von Daten an den Kernel, z.B. Startoptionen.
- **cpus**: Angabe der Prozessoren (Art, Cachegrößen, ...)

Die Peripherie-Elemente sind unter dem **ps7_axi_interconnect_0**-Knoten gelistet, in diesen wird die I2C-0 Peripherie hinzugefügt (Listing 5.2), die für die Konfiguration der Kamera (Kapitel 3.5) verwendet wird. Die Namen der Knoten-Eigenschaften sind nicht fest definiert und abhängig vom Treiber, der durch die *compatible*-Eigenschaft gesetzt wird.

Listing 5.2: Konfiguration der I2C-0 Peripherie in der Device-Tree Datei auf eine Geschwindigkeit von 100kHz für die Verwendung mit dem Kernel-Module *PS7-i2c-1.00.a*. Die Basis-Adresse und Interrupt-Nummer (Subtrahiert mit 32 [10]) ist dem Zynq-Datenblatt [59] zu entnehmen, die Eingangs-Taktfrequenz auf dem Zedboard ist 111MHz.

```
i2c {
        compatible = "xlnx,PS7-i2c-1.00.a";
        reg = <0xE0004000 0x1000>;
        interrupts = <0 25 0>;
        interrupt-parent = <1>;
        bus-id = <0>;
        input-clk = <111111111>;
        i2c-clk = <100000>;
};
```

Die Device-Tree-Datei des Zedboards wird zum Startvorgang von der SD-Karte konfiguriert und der Arbeitsspeicher für Linux begrenzt (Listing 5.3). Von den 512MB des externen DDR3-SDRAM werden dem Linux-Betriebssystem 500MB zugewiesen, die letzten 12MB des Arbeitsspeichers werden vom Video-DMA (Kapitel 5.2.2) verwendet.

Listing 5.3: Einstellung der *bootargs* in der Device-Tree-Datei im Knoten *chosen* zum Starten von SD-Karte und Begrenzung des Arbeitsspeichers.

```
chosen {
    bootargs = "console=ttyPS0,115200 root=/dev/mmcblk0p2 rootwait ro earlyprintk fixrtc mem=500m";
};
```

Der Linux-Kernel in Version 3.2 mit dem Board Support Package (BSP) des Zedboards wird aus den öffentlich zugänglichen Quellen aus einer Versionsverwaltung [17] geladen und kompiliert (Listing 5.4). Zusätzlich zu den Optionen der Standard-Zedboard Konfiguration werden folgende Optionen aktiviert:

- W-LAN Subsystem,
- der Treiber für den W-LAN Chipsatz (*CONFIG_RTL8187*) zum Datalogging (vgl. Kapitel 3.3),
- und das UserSpace IO (UIO) Subsystem (*CONFIG_UIO*, *CONFIG_UIO_PDRV*, *CONFIG_UIO_PDRV_GENIRQ*) für den Zugriff auf die Peripherie (vgl. Kapitel 5.2.1).

Für die Kompilierung des Kernels ist ein Entwicklungs-PC mit Linux erforderlich.

Listing 5.4: Herunterladen der Kernel-Quellen mit git, Konfiguration und Start der Kompilierung mit ARM-Cross-Compiler.

```
> git clone https://github.com/Digilent/linux-digilent.git # Download Kernel
> make ARCH=arm digilent_zed_defconfig # Load default config
> make ARCH=arm menuconfig # Activate W-LAN
> make CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm zImage modules # Compile
> # Install modules to folder root/
> make CROSS_COMPILE=arm-linux-gnueabi- ARCH=arm INSTALL_MOD_PATH=root/ modules_install
```

Das Linux Betriebssystem des Zedboards wird von einer 8GB SD-Karte geladen, dazu wurde die Karte in zwei Partitionen eingeteilt. Der kompilierte Kernel zusammen mit der Hardwarekonfiguration und den Bootloadern (s.o. und Abb. 3.4) wurde auf die erste, mit FAT32 formatierte, Partition der SD-Karte kopiert. Die zweite Partition wurde mit dem Linux-Dateisystem *ext4* formatiert und als Root-Dateisystem für das Debian [13] GNU/Linux 7.0 Betriebssystems verwendet. Dazu wurde vom Entwicklungs-PC ein Cross-Debootstrap (Listing 5.5) [12] ausgeführt, mit dem ein Debian ohne Installationsmedium aus dem Internet geladen und konfiguriert wird. Da dabei ARM-Binärdateien ausgeführt werden, erfolgte die Konfiguration mit dem Emulator QEMU [9].

Listing 5.5: Cross-Debootstrap von Debian GNU/Linux 7.0 (wheezy) von einem Entwicklungs-PC, die zweite Partition der SD-Karte ist in /mnt eingehängt.

```
# Herunterladen der Distribution
> debootstrap --foreign --arch armhf wheezy /mnt http://ftp.debian.org/debian/
# Kopieren des Emulators
> cp /usr/bin/qemu-arm-static /mnt/usr/bin
# Konfiguration, der Emulator wird dafür automatisch gestartet
> chroot /mnt /debootstrap/debootstrap --second-stage
```

## 5.2 Gerätetreiber zur Steuerung der Peripherie

Für den Zugriff auf den Video-DMA und *SAV Controller* wurden Linux-Treiber entwickelt, die im Userspace arbeiten. Die USB-Peripherie wie der Laserscanner und das I2C-Interface zur Parametrisierung der Kamera werden von Treibern des Linux-Kernels unterstützt. Im Baremetal Betrieb werden für den Video-DMA IP-Core und das I2C-Interface die Xilinx-Treiber verwendet.

### 5.2.1 Memory Mapped Kommunikation mit IP-Cores

Auf die PL und PS-Peripherie wird per Memory Mapped I/O zugegriffen, den Adressraum zeigt Abbildung 5.2.

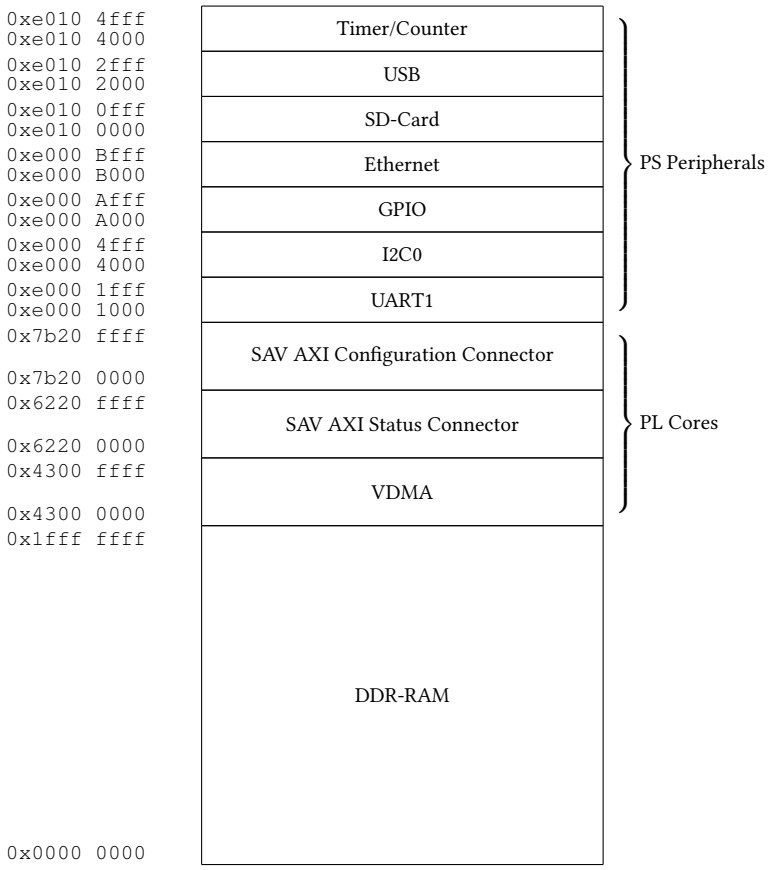



Abbildung 5.2: Zynq-7000 Memory Map

Um die physische Adresse der Peripherie in den virtuellen Adressraum der Threads abzubilden wird das UserSpace IO (UIO) Subsystem von Linux verwendet. Das UIO Framework [36]

wurde zur Treiber-Entwicklung im Userspace für Hardware, wie z.B. FPGAs, für die es kein Kernel-Subsystem gibt, eingeführt. Ein Treiber komplett im Userspace ist einfacher zu entwickeln. Es stehen alle Bibliotheken und Debug-Tools zur Verfügung und bei Fehlern ist kein Neustart des Betriebssystems erforderlich. Aufgrund von Änderungen an der Kernel-API im Linux-Entwicklungsprozess müssen Kernel-Treiber öfter an neue Kernel-Versionen angepasst werden, Userspace-Treiber sind davon nicht betroffen.

Um mit einer Linux-Anwendung auf einen physischen Speicherbereich zuzugreifen, wurde die Device-Datei */dev/mem* entwickelt. Mit dem Systemaufruf *mmap()* lässt sich jede physische Adresse auf den virtuellen Adressraum der Anwendung abbilden. Da dies den gesamten Arbeitsspeicher einschließt, führt das zu Sicherheitsproblemen und Abstürzen [35]. Mit dem UIO-Framework hingegen wird der Zugriff nur auf die in der Konfiguration angegebenen Adressen begrenzt. Zusätzlich unterstützt das UIO-Framework auch die Verwendung von Interrupts. Bei einem *mmap()* Aufruf auf */dev/mem* oder eine UIO-Gerätedatei wird im Linux-Kernel die Funktion *pgprot_noncached()* aufgerufen, die den gemappten Speicherbereich vom Datencache ausschließt. Durch den Aufruf von *pgprot_noncached()* wird die Art des Speicherbereichs in den „Type extension“ Bits (TEX) des Cache Controller auf „Device Memory“ gesetzt, der nicht gecached wird [6].

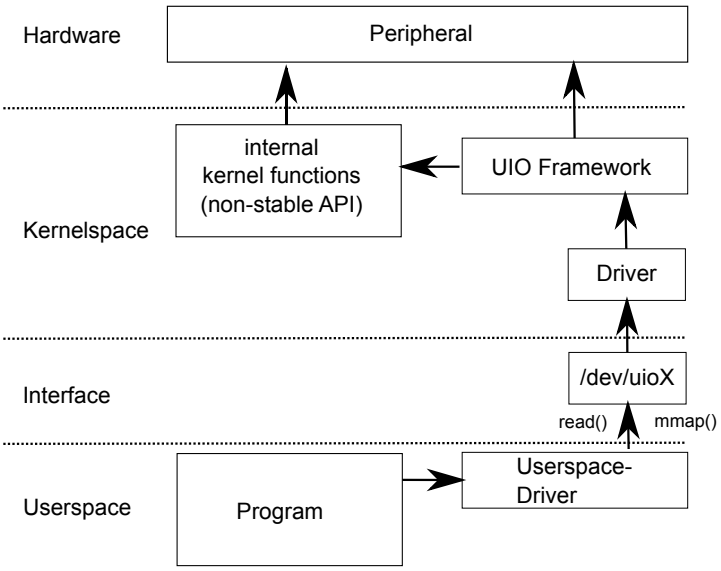



Abbildung 5.3: Mit einem Kernel-Modul zur Konfiguration des UIO-Frameworks wird ein Character-Device erzeugt, auf das ein Userspace-Treiber mit den Systemaufrufen *read()* und *mmap()* zugreift.

Ein UIO-Treiber besteht aus zwei Teilen, einem Kernel-Modul, der den Treiber konfiguriert und Interrupts quittiert, sowie dem Programm im Userspace, das die Arbeit ausführt (Abb.

5.3). Zum Warten auf einen Interrupt führt das Programm den blockierenden Systemaufruf *read()* auf die Gerätedatei aus. Das UIO-Framework setzt den aufrufenden Thread dann in den blockierten Zustand, bis ein Interrupt auftritt (Abb. 5.4). Die Interrupt-Requests werden vom Kernel gezählt und der aktuelle Wert des Zählers als 4-Byte-Wert beim Systemaufruf übertragen.

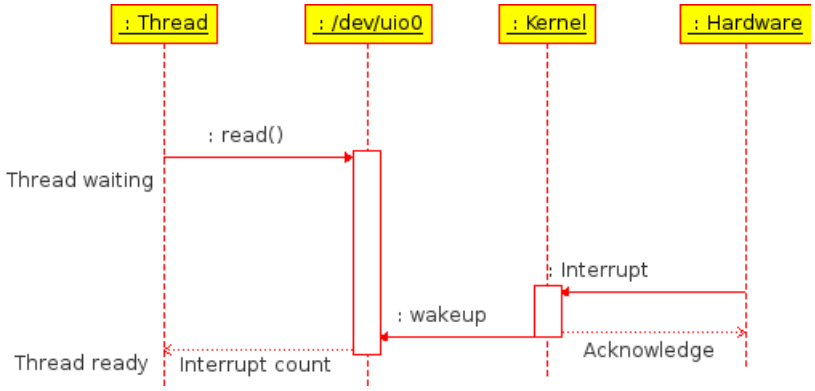



Abbildung 5.4: Zum Warten auf einen Interrupt führt der Thread den blockierenden Systembefehl *read()* auf die UIO-Gerätedatei aus. Ein Interrupt wird vom Kernel quittiert und dem wartenden Thread signalisiert.

Die erhöhte Latenz bei der Verarbeitung des Interrupts im Userspace untersucht Kapitel 6.

Die Konfiguration des UIO-Treibers erfolgt bei dem in dieser Arbeit verwendeten Kernel 3.2 mit einem Kernel-Modul (Listing C.1 in Anhang C). Ab Kernel 3.8 ist UIO per Devicetree konfigurierbar [44]. Zur Konfiguration wird ein Speicherbereich einer Peripherie in der Struktur *resource* mit den Adressen aus dem XPS angegeben (Listing 5.6). Als Typ (*flags*) wird der Wert *IORESOURCE_MEM* angegeben, bei anderen Werten für die Eigenschaft *flags* wird der durch *start* und *end* eingeschlossene Bereich ignoriert. Der Name ist frei wählbar.

Listing 5.6: Zuweisung der Speicherbereiche zur Verwendung mit UIO.

```
1  static struct resource uio_resource[] = {
2          { // VDMA Shared Memory: Last 12MB of RAM
3                  .start = 0x1f400000,
4                  .end   = 0x1fffffff,
5                  .name = "sav_vdma_mem",
6                  .flags = IORESOURCE_MEM
7          },
8          { // VDMA Register
9                  .start = 0x43000000,
10                 .end   = 0x4300ffff,
11                 .name = "sav_vdma_reg",
12                 .flags = IORESOURCE_MEM
13         },
```

```
14          { // AXI Configuration Connector Register
15                  .start = 0x62200000,
16                  .end   = 0x6220ffff,
17                  .name = "sav_axi_reader",
18                  .flags = IORESOURCE_MEM
19          },
20          { // AXI Status Connector Register
21                  .start = 0x7B200000,
22                  .end   = 0x7B20ffff,
23                  .name = "sav_axi_cfg",
24                  .flags = IORESOURCE_MEM
25          },
26  };
```

Vom *SAV Controller* wird für jedes Ergebnis der Hough-Transformation der Edge-Level-Interrupt *HT_DONE* gesendet, der zusammen mit der Interrupt-Callback „irq_handler“ in der Struktur *uio_info* (Listing 5.7) angegeben wird. Die Callback wird zur Ausführung von zeitkritischen Code im Kernelspace vom Interrupthandler des UIO-Frameworks aufgerufen. Dem mit *read()* wartenden Programm im Userspace wird der Interrupt nur signalisiert, wenn die Callback den Wert *IRQ_HANDLED* zurück liefert.

Listing 5.7: Angabe des Interrupts 91 zur Behandlung durch UIO mit Callback.

```
1  static irqreturn_t irq_handler(int irq, struct uio_info *dev_info) {
2          switch(irq) {
3                  case 91:
4                          return IRQ_HANDLED;
5          }
6
7          return IRQ_NONE;
8  }
9
10 static struct uio_info myfpga_uio_info = {
11         .name = "uio_fpga",
12         .version = "0.1",
13         .irq = 91,
14         .handler = irq_handler,
15 };
```

Die Art des Interrupts wird im Linux-Kernel mit *irq_set_irq_type()* (Listing 5.8) konfiguriert, das die entsprechenden Werte im Interrupt-Controller setzt. Zur Initialisierung wird die in *module_init* angegebene Funktion aufgerufen, die mit der Struktur *platform_device* die oben angegebenen Informationen an das UIO-Framework übergibt.

Listing 5.8: Initialisierung des UIO-Moduls und Festlegung des Interrupt-Typs zur Konfiguration des Interrupt-Controllers.

```
static struct platform_device uio_device = {
        .name      = "uio_pdrv", // The Module is handled by the UIO platform driver
        .id     = -1, // Only one instance
        .resource   = uio_resource, // Memory Regions
        .num_resources   = ARRAY_SIZE(uio_resource),
        .dev.platform_data = &myfpga_uio_info, // Interrupt informations
};

static int __init mod_init(void)
{
        platform_device_register(&uio_device);
        irq_set_irq_type(91, IRQ_TYPE_EDGE_RISING);

        return 0;
}
module_init(mod_init);
```

Die Verwendung des UIO-Frameworks für den Zugriff auf einen Speicherbereich von der Anwendung aus zeigt Listing 5.9.

Listing 5.9: Funktion *sav_set_reg()* zum Schreiben von Register *num* des *SAV Controller* ohne Fehlerbehandlung: Der Wert *val* wird in Register *num* geschrieben.

```
void sav_set_reg(int num, uint32_t val) {
        uint32_t *reg;
        int file;
        void *sav;

        file = open("/dev/uio0", O_RDWR);
        // map 0xffff bytes from UIO-Memory, third mapping to sav-pointer
        sav = mmap(NULL, 0xffff, PROT_READ|PROT_WRITE, MAP_SHARED, file, 3*getpagesize());

        // Access 4 Byte Register num from sav-pointer
        reg = (uint32_t *)(sav+num*4);
        *reg = val;

        munmap(sav, 0xffff);
        close(file);
}
```

### 5.2.2 Konfiguration des VDMA IP-Cores

Für die Konfiguration des VDMA IP-Cores im Register Direct Mode (Abbildung 4.7 in Kapitel 4.2) werden die S2MM und MM2S-Register in der folgenden Reihenfolge beschrieben:

1. Kontrollregister *DMACR* für die gewählte Betriebsart initialisieren und Run/Stop (RS)-Bit setzen, um das VDMA-Modul zu starten.

2. Speicheradressen an die Bilder geschrieben oder von der Bilder gelesen werden in die *FSTOREx*-Register schreiben.
3. Anzahl Speicheradressen mit der gearbeitet werden soll im *FRMSTORE*-Register konfigurieren.
4. Die Schrittweite zwischen zwei Bildzeilen im *STRIDE*-Register setzen.
5. Die Bildbreite in das *HSIZE*-Register schreiben.
6. Durch Setzen der Bildhöhe im *VSIZE*-Register wird das Kopieren der Bilder gestartet.

Die Kanäle S2MM und MM2S des VDMA-Moduls werden beide für die Betriebsart „Circular Park“ [66] konfiguriert, bei der alle *FRMSTORE*-Anzahl Speicheradressen in einem Ring durchlaufen werden. Das Modul wird nur für eine Speicheradresse (*FRMSTORE*=1) konfiguriert, der im Register *FSTOREx* gesetzte Speicherbereich enthält also immer das aktuelle Bild. Dabei wird vom S2MM der Speicherbereich 500MB bis 503MB und vom MM2S der Speicherbereich 503MB bis 506MB verwendet. Die Interrupts des VDMA-Moduls werden nicht verwendet, damit findet keine Synchronisation von CPU und VDMA für den Zugriff auf den gemeinsamen Speicher statt. So kann es beim S2MM-Kanal vorkommen, daß der Speicher während des Lesevorgangs der CPU vom VDMA-Modul beschrieben wird. Dies führt zu Tearing, bei dem das von der CPU gelesene Bild zu Teilen aus dem vorhergehendem und nachfolgendem Bild besteht. Da der VDMA-Aufbau nur zur Visualisierung und nicht als Berechnungsgrundlage dient, wird dies als unkritisch betrachtet. Bei Betrachtungen konnte bei 60 Bildern pro Sekunde kein Unterschied zwischen zwei Bildern ausgemacht werden.

### Baremetal Zugriff auf den VDMA IP-Core

Die Konfiguration des VDMA IP-Core ohne Betriebssystem wird mit dem Xilinx axivdma-Treiber, Version v4.01a vorgenommen. Der Treiber befindet sich im *libsrc*-Verzeichnis des BSP. Initialisiert wird der Treiber durch die Struktur *XAxiVdma_Config*, in die VDMA-Parameter aus dem BSP geschrieben werden. Die Video-Parameter werden mit der Struktur *XAxiVdma_DmaSetup* (Listing 5.10) gesetzt. Der S2MM-Kanal wird im Treiber als *write*- und der MM2S-Kanal als *read*-Kanal Bezeichnet. Mit der Funktion *XAxiVdma_StartWriteFrame()* wird der S2MM-Kanal gestartet, dabei werden die Register in der geforderten Reihenfolge beschrieben. Für die gesetzten Speicherbereiche sollte der Datencache ausgeschaltet, oder nach jeder Operation mit *Xil_DCacheFlushRange()* (xil_cache.h) geleert werden [3][58].

Listing 5.10: Baremetal Initialisierung des S2MM-Kanals des ersten VDMA-Moduls für ein 752x480 Bild in der Betriebsart „Circular Park“ mit 3 Speicheradressen.

```
#include "xaxivdma.h"

XAxiVdma vdma;
XAxiVdma_Config *vdmaconf;
XAxiVdma_DmaSetup vdmasetup;

// Get VDMA-0 Configuration data
vdmaconf = XAxiVdma_LookupConfig(XPAR_AXIVDMA_0_DEVICE_ID);
// init VDMA-0
XAxiVdma_CfgInitialize(&vdma, vdmaconf, vdmaconf->BaseAddress);

// Configure Circular Park for 752x480, 1 Byte/Pixel
memset(&vdmasetup, 0, sizeof(XAxiVdma_DmaSetup));
vdmasetup.VertSizeInput = 480;
vdmasetup.HoriSizeInput = 752;
vdmasetup.Stride = vdmasetup.HoriSizeInput;
vdmasetup.EnableCircularBuf = 1;
vdmasetup.FrameStoreStartAddr[0] = data1;
vdmasetup.FrameStoreStartAddr[1] = data2;
vdmasetup.FrameStoreStartAddr[2] = data3;
XAxiVdma_StartWriteFrame(&vdma, &vdmasetup);
```

### VDMA unter Linux mit UIO

Für die Konfiguration des Video-DMA Moduls unter Linux wurden die zwei Funktionen *vdma_write_reg(addr, value)* und *vdma_read_reg(addr)* entwickelt, mit denen über das UIO-Subsystem auf die VDMA-Register zugegriffen wird. Mit den Funktionen wird das VDMA-Modul für die Betriebsart „Circular Park“ (siehe oben) konfiguriert (Listing 5.11). Dem Register *FSTORE1* wird ein Speicherbereich in den letzten 12MB des Arbeitsspeichers gegeben, der nicht von Linux verwaltet wird und deswegen dem VDMA-Modul exklusiv zur Verfügung steht. Da auf den Speicherbereich ebenfalls über das UIO-Subsystem zugegriffen wird, ist dieser vom Datencache ausgenommen. Die Bildbreite wird für die Register *STRIDE* und *HSIZE* mit der Farbtiefe multipliziert, da die Farbtiefe gleich der Anzahl Bytes pro Pixel ist.

Listing 5.11: Konfiguration des VDMA S2MM Channels unter Linux.

```
// Circular_Park=0x2 | Run=0x1
vdma_write_reg(VDMA_REG_S2MM_DMACR, vdma_read_reg(VDMA_REG_S2MM_DMACR) | 0x2 | 0x1);
vdma_write_reg(VDMA_REG_S2MM_FSTORE1, 0x1f400000); // @500MB
vdma_write_reg(VDMA_REG_S2MM_FRMSTORE, 1);
vdma_write_reg(VDMA_REG_S2MM_STRIDE, HSIZE*DEPTH);
vdma_write_reg(VDMA_REG_S2MM_HSIZE, HSIZE*DEPTH);
vdma_write_reg(VDMA_REG_S2MM_VSIZE, VSIZE);
```

### 5.2.3 I2C-Interface zur Parametrisierung der Kamera

Das bisherige Microblaze PLB-Interface zur Parametrisierung der Kamera [47] wurde auf den Zynq-7000 MPSoC für den Betrieb unter Baremetal und Linux portiert.

Für die Portierung wurden die I2C Sende- und Empfangsroutinen portiert, der für den Microblaze geschriebene C-Code auf dem ARM neu kompiliert und als Bibliothek *libsavctrl* in die Software eingebunden. Dabei wurde auch die Byte-Reihenfolge angepasst, da der Zynq-7000 ARM MPSoC ein Little-Endian [59] und der Microblaze ein Big-Endian System [68] ist (Listing 5.12).

Listing 5.12: Parameterübergabe eines Zwei-Byte Wertes in ein Array aufgrund der unterschiedlichen Byte-Reihenfolge zwischen Microblaze und Zynq-ARM. Das Array wird als I2C-Sendepuffer verwendet.

```
1 uint16_t value = 0xBEEF; // I2C parameter
2
3 // Microblaze (big endian)
4 uint8_t *data = (uint8_t *)&value;
5
6 // Zynq ARM (little endian)
7 uint8_t data[2];
8 data[0] = (value >> 8);
9 data[1] = value;
```

#### Baremetal Zugriff auf das Zynq-7000 I2C-Interface

Für den Zugriff auf die I2C-Peripherie ohne Betriebssystem wird der Xilinx iicps-Treiber, Version 1.02a verwendet, die Dokumentation des PS I2C-Treibers ist im BSP des Xilinx SDK enthalten. Initialisiert wird der Treiber durch die Struktur *XIicPs_Config*, in die die Konfigurationsdaten des Interfaces aus dem BSP geschrieben werden. Für die Kamera wird der I2C-Bus im 100kHz Standard-Mode betrieben (Listing 5.13), der I2C-Takt wird aus der einfachen CPU-Taktfrequenz (CPU_1x, Kapitel 3.0.1) generiert (111MHz).

Für den Polling-Betrieb werden zum Lesen und Schreiben auf den Bus die Funktionen *XIicPs_MasterRecvPolled* und *XIicPs_MasterSendPolled()* aus *xiicps.h* verwendet.

Listing 5.13: Baremetal Initialisierung des ersten I2C-Controllers im Standard-Mode mit 100kHz.

```
#include <xiicps.h>

XIicPs_Config *i2cconf;
XIicPs i2cdev;

// Get I2C-0 Configuration data
i2cconf = XIicPs_LookupConfig(0);
// init I2C-0
XIicPs_CfgInitialize(&i2cdev, i2cconf, XPAR_PS7_I2C_0_BASEADDR);
// Use Standard Mode 100kHz
XIicPs_SetSClk(&i2cdev, 100000);
```

### Zynq-7000 I2C-Interface unter Linux

Die I2C-Peripherie wird durch den Kernel Treiber *PS7-i2c-1.00.a* im Xilinx Kernel unterstützt. Mit der I2C-Konfiguration in der Device-Tree-Datei (Kapitel 5.1) wird die I2C-Gerätedatei */dev/i2c-0* erzeugt, mit der von einem Programm aus auf den I2C-Bus zugegriffen wird [14].

Bei dem I2C-dev-Interface handelt es sich um Dateien im Gerätedateisystem */dev*, für jeden I2C-Controller existiert eine Datei. Auf diese Datei wird mit den Befehlen für die Manipulation von Dateien zugegriffen, also

- *open()*: Öffnen der Gerätedatei.
- *read()*: Lesen von Datenströmen von dem I2C-Gerät.
- *write()*: Schreiben von Datenströmen an das I2C-Gerät.
- *close()*: Schließen der Gerätedatei.
- *ioctl()*: Setzt die Adresse des I2C-Gerätes an den die *read()* und *write()* Operationen gehen: Mit dem Befehl wird die I2C-Startbedingung ausgeführt, der Aufruf hat das Format *ioctl(file, I2C_SLAVE, 0x50)*, wobei 0x50 die Adresse ist, die vom I2C-Controller angesprochen werden soll. Für das Makro *I2C_SLAVE* wird die Header-Datei *linux/i2c-dev.h* benötigt.

Das Kamera-Interface ist als Bibliothek *libcamerainterface* in die Software eingebunden.

### 5.2.4 USB-Interface des Hokuyo URG-04LX Laserscanners

Die auf dem OMAP4430-MPSoC mit Betriebssystem Android laufende Objekterkennungs-Software [27] wurde auf den Zynq-7000 MPSoC unter Linux portiert. Da beide MPSoCs auf einem Dual-Core ARM Cortex-A9 Prozessor aufbauen, werden nur die Betriebssystem spezifischen Elemente angepasst.

Der Laserscanner [25] verwendet den USB Standard [53] CDC (Communications Device Class) mit dem Abstract Control Model, die CDC definiert die Architektur für USB Kommunikations-Geräte [52]. CDC-Geräte werden bei Betriebssystemen, die den USB-Standard implementieren, von Standard Treibern unterstützt. Im Abstract Control Model (ACM) werden je ein Bulk-Endpunkt zum Senden- und Empfangen von Daten festgelegt über die mit dem USB-Gerät kommuniziert wird. Unter Linux wird beim Verbinden des Laserscanners durch das *usb_cdc*-Modul des Kernels die Geräte-Datei */dev/ttyACM0* (Abstract Control Model) erzeugt mit der auf den Laserscanner zugegriffen wird.

Auf diese Datei wird mit den Befehlen für die Manipulation von Dateien zugegriffen, also

- *open()*: Öffnen der Gerätedatei.
- *read()*: Lesen von Datenströmen von dem Laserscanner über den Bulk-Endpunkt.
- *write()*: Schreiben von Datenströmen an den Laserscanner über den Bulk-Endpunkt.
- *close()*: Schließen der Gerätedatei.

Das Format der Kommunikation ist in SCIP 2.0 [24] spezifiziert. Mit dem Konfigurationsstring „MD0128064001000\n“ wird der Laserscanner auf einen Scanwinkel von 180° in 512 Schritten eingestellt. Damit wird der gesamte Frontbereich des Fahrzeugs erfasst, sowie Hindernisse hinter Kurven detektiert [27]. Nach der Konfiguration sendet der Laserscanner kontinuierlich die Scanergebnisse, die aus dem Empfangspuffer gelesen werden (Listing 5.14).

Für die Objekterkennung wird der RANSAC (RANdom SAmple Consensus)-Algorithmus eingesetzt, dabei wird jedes Objekt durch zwei Begrenzungspunkte (Anfang und Ende des Objekts), sowie der Punkt mit der geringsten Entfernung identifiziert [27]. Die Scan-Ergebnisse mit RANSAC-Algorithmus werden zur Visualisierung in eine Vektorgrafik-Datei (.svg) [57] geschrieben (vgl. Abb. 5.5), die von einem Webserver ausgeliefert auf dem Entwicklungs-PC angezeigt wird.

Listing 5.14: Laserscanner Kommunikation und Konfiguration.

```
char *initstr = "MD0128064001000\n";

if ((dev = open(/dev/ttyACM0, O_RDWR | O_NOCTTY)) < 0) {
        perror("Laserscanner open");
        return -1;
}

// unset termios settings
bzero(&ioset, sizeof(struct termios));
tcsetattr(dev, TCSANOW, &ioset);

// Configure Device
write(dev, initstr, strlen(initstr));

// Receive scan data
// Function will block until scan completes
bytesread = read(dev, databuffer, 1616);
```

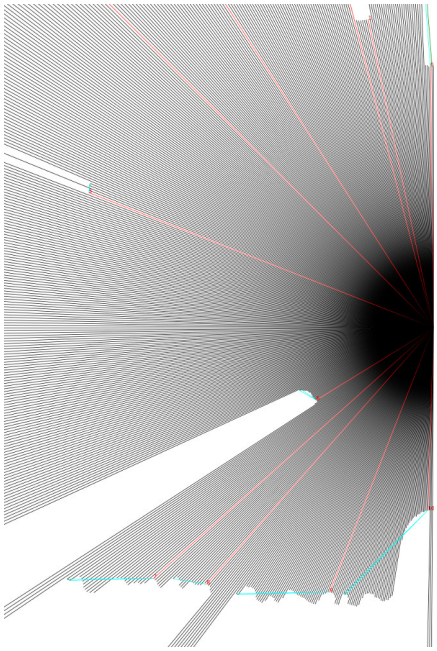


Abbildung 5.5: Ergebnis-Visualisierung des RANSAC-Algorithmus. Die roten Linien identifizieren den Punkt mit der geringsten Entfernung, die blauen Linien Anfang und Ende des Objekts.

## 5.3 Multithreading SW-Entwurf unter SMP Linux

Die Fahrspurführung, das Datalogging und die Visualisierung erfolgen in Threads (Abb. 5.6), pro Prozessor wird der Scheduler mit einer eigenen Liste von Threads ausgeführt. Mittels Lastverteilung werden bei Bedarf Threads periodisch in die Scheduler-Liste eines anderen Prozessors verschoben. Die entwickelte Anwendung zur Ausführung der Threads ist in Anhang B beschrieben.

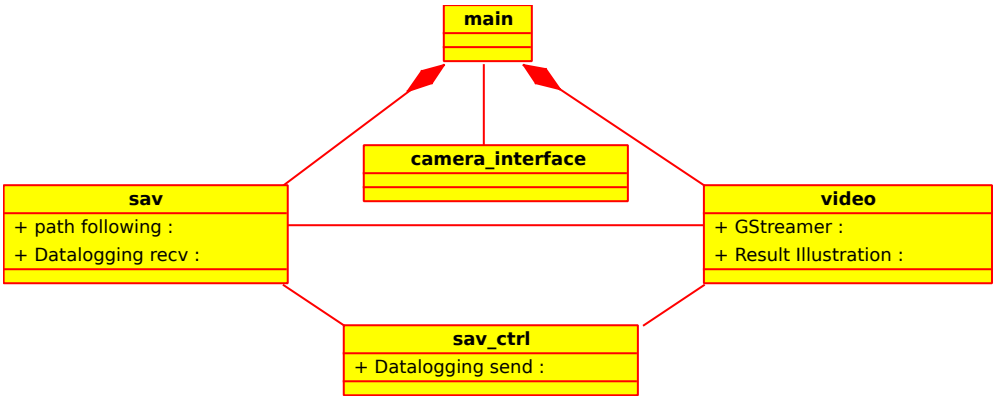



Abbildung 5.6: Aufteilung der Threads auf die C-Module mit Relationen.

### 5.3.1 Steuerung der Fahrspurerkennung und Aktoren

Der *SAV Controller* wird durch die Softwareregister des *AXI Configuration Connectors* eingestellt und initialisiert. Zum Fahren wird eine Sollgeschwindigkeit für den Geschwindigkeitsregler oder die PWM direkt angegeben, die Angabe des Lenkwinkels erfolgt in Grad. Der Geschwindigkeitsregler wird auf eine Pulsbreite von 70µs von der Mittelstellung begrenzt.

Zur Initialisierung des *SAV Controllers* wird ein Reset durchgeführt und die Konfigurations-Register (Tabelle 4.5 in Kapitel 4.3) beschrieben, währenddessen bleibt Bit 31 (Enable) des Konfigurations-Register (15) gelöscht. Der Reset erfolgt durch Schreiben von 0x0000000A an Adresse 256 des *AXI Configuration Connectors*, der dadurch den Reset-Pin für 8 AXI-Takte setzt.

Die reellen Zahlen werden in der Software mit Fließkommazahlen, im FPGA durch Festkommazahlen im Q-Format [41] dargestellt. Zur Umwandlung werden die C-Makros *FLOAT2FIX()* und *FIX2FLOAT()* (Listing 5.15) verwendet.

Listing 5.15: Umwandlungsfunktionen zwischen Fließ - und Festkommazahlen.

```
1 #define FLOAT2FIX(a, frac)      ((int32_t)(a * (1L<<frac)))
2 #define FIX2FLOAT(a, frac)      (a / (float)(1L<<frac))
```

Das erste Argument der Makros ist die Zahl die umgewandelt werden soll, das zweite ist die Anzahl Nachkommastellen der Zahl in der Festkomma Darstellung. Für die Umwandlung wird der Wert mit der Festkomma-Darstellung der Zahl Eins multipliziert.

### 5.3.2 Die Linux-Scheduler

Der Linux-Kernel implementiert die Posix Scheduler-Regeln [1] *SCHED_FIFO*, *SCHED_RR* und *SCHED_OTHER* die Prioritäten gesteuert abgearbeitet werden. Ab Linux 2.6.23 wird *SCHED_OTHER* durch den „Completely Fair Scheduler (CFS)“ implementiert, der standardmäßig für die Ausführung von Tasks eingesetzt wird. Der CFS verwendet als Grundlage Red-Black Trees (Abb. 5.7) [31].

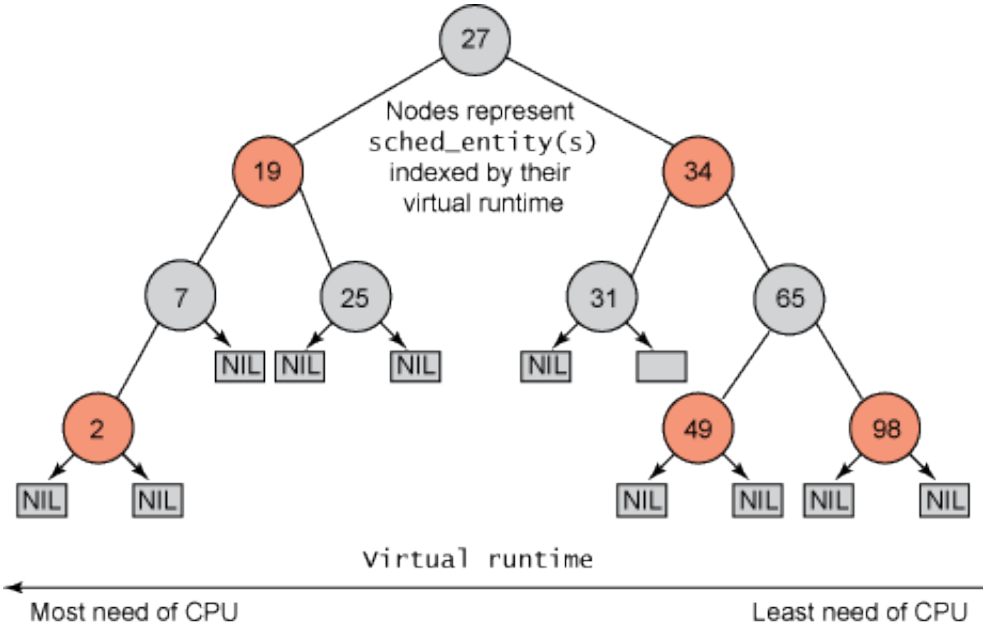



Abbildung 5.7: Die vom CFS Schedulers verwendeten Red-Black Trees. Der Baum enthält alle Tasks im rechenbereiten Zustand, sortiert nach der „virtual runtime“. Tasks im linken Bereich werden ausgeführt. Bild: [31]

Ein Prozess wird vom Linux-Scheduler wie ein Thread behandelt. Der Scheduler verfügt über unterschiedliche Strategien für Echtzeit-Tasks, nicht-Echtzeit-Tasks und nicht unterbrechbare Tasks. Mit aktivierter *CONFIG_PREEMPT*-Option unterstützt der Linux-Kernel weiche

Echtzeitanforderungen [30], mit der sich auch Prozesse, die einen Systembefehl ausführen, durch einen höher priorisierten Prozess unterbrechen lassen. Für harte Echtzeitanforderungen existiert mit *CONFIG_PREEMPT_RT* [42] ein Patch für den Linux-Kernel, der u.a. Prioritätenvererbung implementiert.

Der CFS-Scheduler soll eine faire Verteilung der CPU-Zeit auf die Threads garantieren. Bekommt ein Thread relativ am Bedarf zu wenig CPU-Zeit, soll es als Ausgleich mehr CPU-Zeit bekommen. Um den Ausgleich zu berechnen, behält der CFS-Scheduler mit der „virtual runtime“ die Länge aller CPU-Zeiten, die der Thread bekommen hat. Ein Thread mit einem geringen „virtual runtime“-Wert hat einen hohen Bedarf ausgeführt zu werden. Der CFS-Scheduler verfügt auch über „sleeper“-Fairniss, so daß Threads die im blockierten Zustand sind (z.B. weil sie auf I/O warten), einen vergleichbaren Anteil der CPU bekommen, wenn sie in den Zustand rechenbereit wechseln.

Thread-Prioritäten haben nur Einfluss auf die Ausführungsdauer, bevor ein Thread unterbrochen wird. Hochpriorisierte Threads haben also eine längere Ausführungsdauer als niedrig priorisierte.

Die Echtzeit-Regeln *SCHED_FIFO* und *SCHED_RR* verfügen über je 99 Prioritäten, mit 99 als höchster und 1 als niedrigster Priorität. Pro Priorität wird vom Kernel eine Warteschlange verwaltet.

Bei der *SCHED_FIFO*-Regel wird ein Thread solange ausgeführt bis er die CPU abgibt, oder durch einen höher priorisierten Prozess unterbrochen wird. Ein *SCHED_FIFO*-Thread, der in den rechenbereiten Zustand wechselt wird in den FIFO seiner Priorität eingefügt.

Die *SCHED_RR* arbeitet wie die FIFO-Regel, nur daß die maximale CPU-Zeit begrenzt wird. Ist die maximale Ausführungsdauer erreicht, wird der Prozess unterbrochen und an das Ende des FIFOs angefügt [37].

Der Linux Scheduler prüft zunächst, ob ein Thread der Echtzeit-Klassen ausgeführt werden soll, dabei werden alle Prioritäten durchgegangen. Steht kein Thread der Echtzeit-Klassen zur Ausführung an, wird der CFS-Scheduler ausgeführt.

### 5.3.3 Fahrspurführung als Posix Thread

Mit der Funktion *pthread_setschedparam()* wird der Posix-Thread „Path following“, der die Fahrspurführung aus dem Ergebnis der Hough-Transformation berechnet, auf die *SCHED_FIFO*-Strategie eingestellt.

Eingabe der Fahrspurführung [41][34] ist das Ergebnis der Hough-Transformation, die die Position und den Winkel der Fahrspur in Hesse'scher Normalform relativ zum Fahrzeug (Abb. 5.8) in AXI-Register schreibt. Die Fahrspurerkennung erfolgt in einem Ausschnitt des Bildes, der Region-of-Interrest (ROI). Der Abstand $r$ wird in Pixel von der linken oberen Ecke der ROI, der Winkel $\Phi$ in Grad mit mathematisch negativem Drehsinn angegeben [48]. Für die weitere Berechnung durch die Software wird der Winkel in den positiven Drehsinn umgekehrt und in Rad umgerechnet. Aus der Position und Lage der Fahrspur wird der Look-Ahead-Point (LAP) berechnet, der als Zielpunkt von den Fahrspurführungs-Algorithmen angesteuert wird. Der LAP befindet sich immer mit den Parametern Look-Ahead-Distance LAD vor dem Fahrzeug und Abstand D von der Fahrspur.

Im Unterschied zu früheren Versionen [48] wurde die Drehmatrix der Ebene zum Rotieren des Koordinatensystems angepasst. Die Drehung nach „rechts“ vom Fahrzeugkoordinatensystem in das Hilfskoordinatensystem ergibt sich zu:

$$\begin{pmatrix} X_{ROI_H} \\ Y_{ROI_H} \end{pmatrix} = \begin{pmatrix} cos(-\Phi) & -sin(-\Phi) \\ sin(-\Phi) & cos(-\Phi) \end{pmatrix} * \begin{pmatrix} X_{ROI_V} \\ Y_{ROI_V} \end{pmatrix} \tag{5.1}$$

Die horizontale Koordinate des LAP berechnet sich aus der Position der ROI, dem Abstand der erkannten Fahrspur $r$ und dem Sollabstand $D$

$$X_{LAP_H} = X_{ROI_H} + r - d \tag{5.2}$$

und die vertikale Koordinate des LAP aus dem Satz des Pythagoras (Abb. 5.8)

$$Y_{LAP_H} = \sqrt{LAD^2 - X_{LAP_H}^2} \tag{5.3}$$

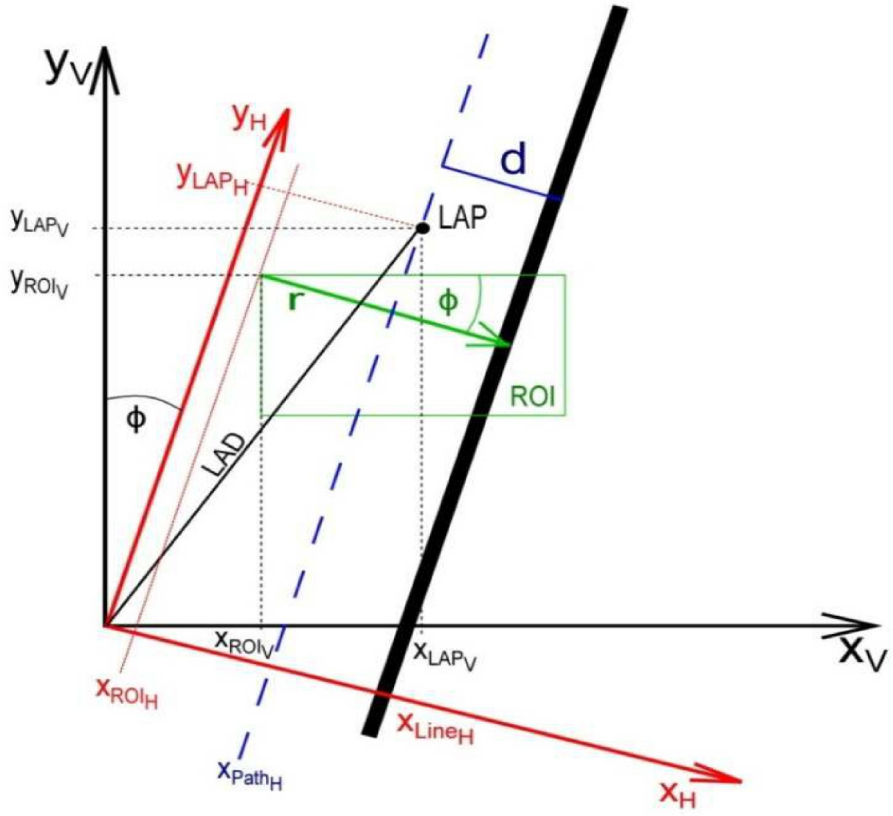



Abbildung 5.8: Für die Berechnung des LAP wird das Fahrzeugkoordinatensystem $(x_V, y_V)$ um den Winkel $\Phi$ in das Hilfskoordinatensystem $(x_H, y_H)$ gedreht. Nullpunkt, LAP und $Y_{LAP_H}$ bilden dann ein Dreieck mit rechtem Winkel mit dem sich der LAP berechnen lässt. Bild: [48]

Die Position des LAP im Fahrzeugkoordinatensystem erhält man durch Rücktransformation mit

$$\begin{pmatrix} X_{LAP_V} \\ Y_{LAP_V} \end{pmatrix} = \begin{pmatrix} cos(\Phi) & -sin(\Phi) \\ sin(\Phi) & cos(\Phi) \end{pmatrix} * \begin{pmatrix} X_{LAP_H} \\ Y_{LAP_H} \end{pmatrix} \tag{5.4}$$

Aus der Position des LAP lassen sich Fahrspurführungs-Algorithmen wie Pure Pursuit berechnen [48].

### 5.3.4 Datalogging über Ethernet und W-Lan

Das Aufzeichnen von Mess- und Stellgrößen aus dem Fahrbetrieb, sowie die Parametrisierung über einen entfernten Entwicklungs-PC wurde auf das TCP/IP Protokoll portiert und wird über W-LAN oder Ethernet gesendet (Kapitel 3.3). Das Datenübertragungsprotokoll (Beschreibung siehe [28]) wurde beibehalten. Für die Portierung wurden die Sende- und Empfangsroutinen durch TCP/IP-Sockets ersetzt, der Microblaze C-Code auf dem ARM neu kompiliert und als Bibliothek *libsavctrl* in die Software eingebunden.

Der Empfang von Konfigurationsdaten und das Senden der Mess- und Stellgrößen erfolgen in jeweils einem Thread. Der Thread zum Empfangen der Konfigurationsdaten blockiert durch den Systembefehl *read()* bis neue Daten anliegen. Um die Messwerte periodisch zu senden, wird der Sende-Thread durch ein *usleep()* gesteuert.

Am Entwicklungs-PC erfolgen der Empfang und die Aufzeichnung der Daten sowie die Parametrisierung über ein bestehendes Java-Programm [28], das von der seriellen Schnittstelle auf Netzwerkkommunikation über TCP/IP Sockets umgestellt wird (Abb. 5.9).

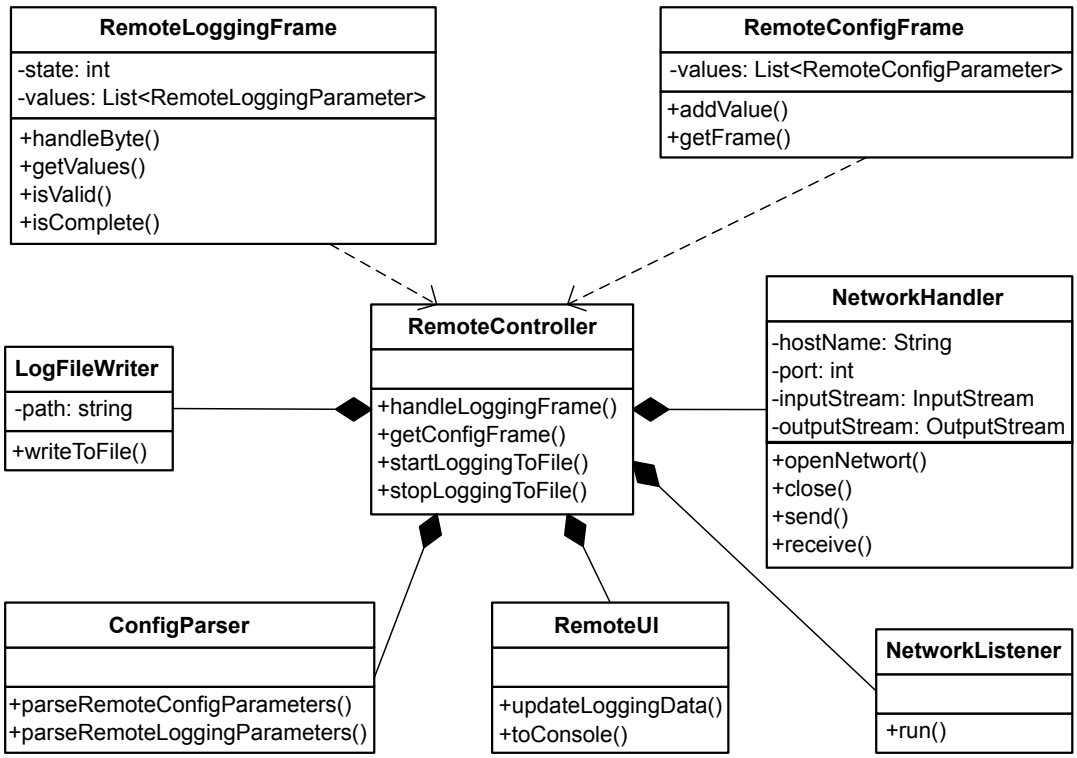



Abbildung 5.9: Klassendiagramm des auf Sockets portierten *Java Remote Tools*, das auf dem Entwicklungs-PC ausgeführt wird. Bild: [28].

Dazu wurden die folgenden Änderungen an den Klassen durchgeführt:

- Klasse *SerialPortHandler* gegen *NetworkHandler* getauscht, die die TCP-Kommunikation mit *java.net.Socket* implementiert.
- Event-Listener *SerialPortListener* für den Empfang von Logging-Daten gegen *Network-Listener* ausgetauscht.
- Im User Interface die Optionen für die serielle Schnittstelle gegen die Angabe der IP-Adresse ausgetauscht, die zum Verbindungsaufbau angegeben wird.

### 5.3.5 Framegrabber und Visualisierung der Fahrspurführung in Software

Für die Visualisierung der Fahrspurerkennung und Fahrspurführung unter Linux wird das freie Multimedia-Framework GStreamer (GST) [56] verwendet. GStreamer gründet auf dem Konzept einer Video-Pipeline und wird unter Linux-PCs und einigen eingebetteten Systemen, wie den Smartphones Nokia N900 und N9, für die Wiedergabe von Multimedia-Inhalten eingesetzt.

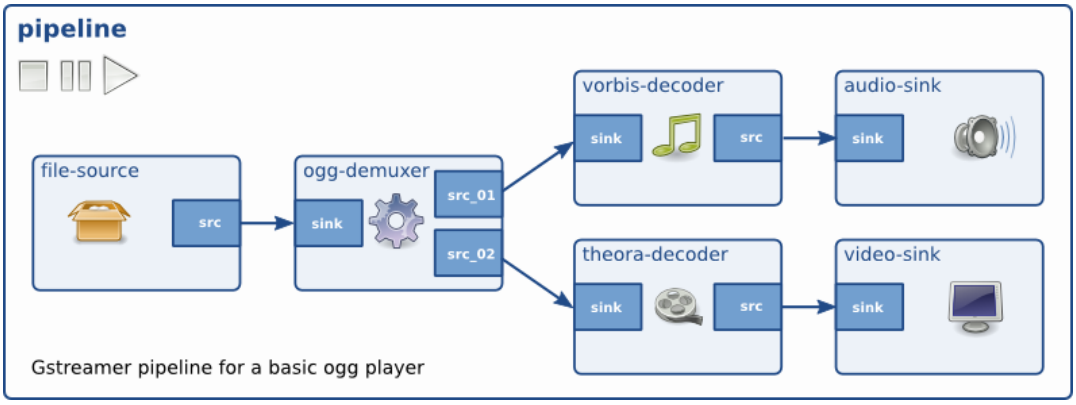



Abbildung 5.10: GStreamer-Pipeline zur Wiedergabe von Audio- und Videodateien, die mit Ogg (theora und vorbis) komprimiert wurden. Bild: [56]

GStreamer ist modular aufgebaut: Jedes Element in der Pipeline führt einen Arbeitsschritt aus, z.B. das Lesen einer Datei, die Dekompression von Daten oder die Ausgabe von Daten auf einen Bildschirm. Durch die Verkettung von Elementen zu einer Pipeline (Abb. 5.10) werden komplexe Aufgaben wie die Wiedergabe von Multimedia-Inhalten durchgeführt. Verbunden werden die Elemente über ihre Pads (Kontaktstellen), ein Pad ist entweder Quelle

(source, kurz src) oder Senke (sink). Über die Quelle werden Daten in das Element eingelesen und über die Senke ausgegeben. Elemente verfügen über ein Pad, mehrere Pads oder kein Pad eines Typs. Ein Pad definiert die Daten mit denen es arbeitet über dessen Capabilities (Eigenschaften, kurz: Caps), mit denen beim Aufbau einer GStreamer-Pipeline auf Fehler, wie die versehentliche Anbindung einer Audio-Quelle an eine Videoausgabe, geprüft wird. Als Beispiel sind die Capabilities eines MPEG4-Dekodier-Elementes die Entgegennahme von komprimierten MPEG4-Videodaten am Quell-Pad und die Ausgabe des dekomprimierten Videos an der Senke.

Die Verarbeitung von Daten durch ein Element erfolgt durch die „Chain“-Funktion, die die zu verarbeitenden Daten als Zeiger auf einen *GstBuffer* übergeben bekommt (Listing 5.16). Der *GstBuffer* enthält Informationen über die zu verarbeitenden Daten wie:

- Zeitstempel und Länge der Aufnahme,
- Eigenschaften, bei Videos sind das Bildbreite, Bildhöhe, Farbraum und Farbtiefe,
- und einen Zeiger auf die Rohdaten des Multimedia-Objektes.

Listing 5.16: Chain-Funktion eines GST-Elementes, das Daten an der Senke entgegennimmt, bearbeitet und an der Quelle ausgibt.

```
GstFlowReturn gst_my_filter_chain(GstPad *sinkpad, GstBuffer *buf)
{
        // Get element this pad belongs to
        GstMyFilter *filter = GST_MY_FILTER(GST_OBJECT_PARENT(sinkpad));
        // ...
        // Do something with the buffer
        // ...
        return gst_pad_push(filter->srcpad, buf); // Pass modified buffer to srcpad
}
```

Das in der Programmiersprache C geschriebene GStreamer basiert auf der Softwarebibliothek GLib, die die Programmiersprache um Objekt- und Ereignisorientierte Programmierung erweitert [22]. Ein GStreamer-Element ist damit ein Objekt, verfügt also über Methoden, Attribute und reagiert auf Ereignisse.

Der Austausch von Daten (Abb. 5.11) findet bei GStreamer-Anwendungen wie folgt statt:

- Bei Elementen über *GstBuffer* die von deren *src*-Pads an die *sink*-Pads des nachfolgenden Elementes weitergegeben werden.
- Mit *events* werden Objekte zwischen Elementen untereinander oder von der Anwendung aus an Elemente gesendet.

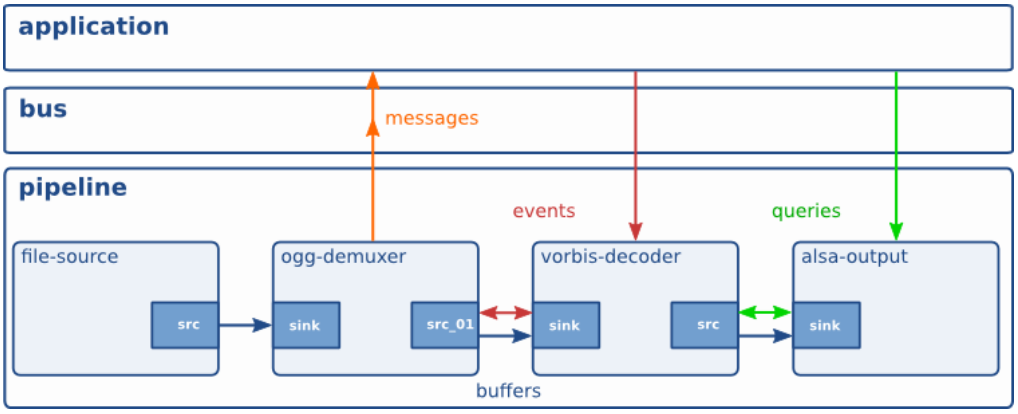



Abbildung 5.11: GStreamer-Pipeline mit Kommunikationsfluss zur Anwendung. Bild: [56]

- Zustandsänderungen und Fehler werden mit, zur Bildfrequenz asynchronen, *Messages* an die Anwendung signalisiert. Dabei dient ein Message-Bus als Zwischenspeicher, bis die Daten von der Anwendung abgeholt werden.
- *Queries* werden von der Anwendung und von Elementen für die Abfrage von Informationen von anderen Elementen, wie Laufzeit und aktuelle Position, verwendet. *Queries* werden mit der Bildfrequenz synchron beantwortet.

### Verwendung von GStreamer zur Visualisierung der Fahrspurführung

Ziel ist die Visualisierung der einzelnen Stufen der Bildverarbeitungspipeline und Fahrspurführung zur Analyse und Dokumentation. Die Ausgabe erfolgt entweder (Anhang B) über

- einen VGA-Monitor,
- über per UDP übertragenes Motion JPEG (MJPEG), bei dem jedes Einzelbild des Videostroms einzeln komprimiert wird,
- Speicherung eines einzelnen Bildes als JPEG auf dem Fahrzeug,
- oder als in einem AVI-Container (Audio Video Interleave) gespeicherten MJPEG.

Dazu wurde mit GStreamer eine Pipeline (Abb. 5.12) entwickelt, die je nach Ausgabe über unterschiedliche Elemente verfügt. Die JPEG-Komprimierung erfolgt mit der CPU.

Der Zugriff auf den mit dem VDMA geteilten Speicherbereich erfolgt mit den Elementen *appsink* und *appsrc*. Das *appsrc*-Element liest zeitgesteuert mit der *timerfd*-Schnittstelle die

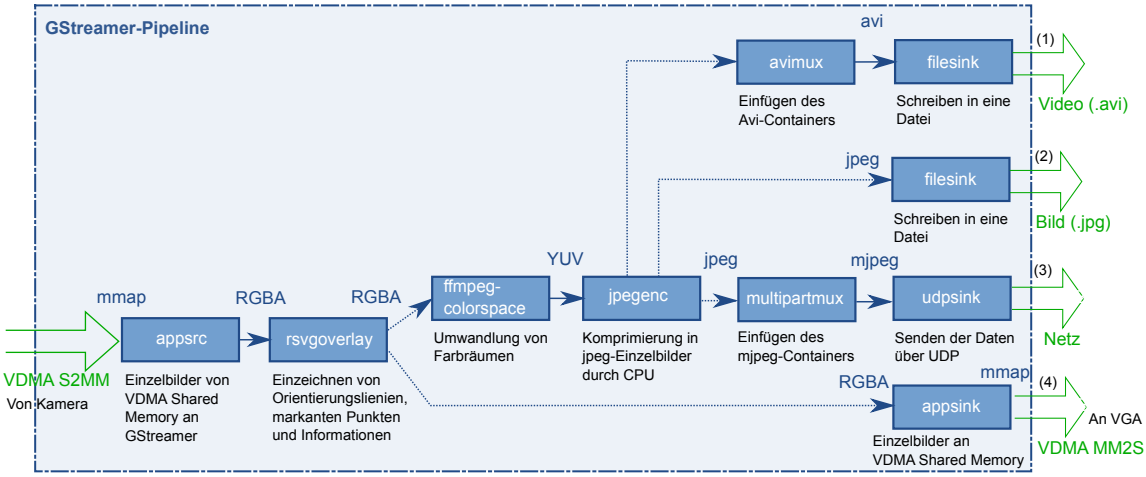



Abbildung 5.12: GStreamer-Pipeline für die Visualisierung, die gepunkteten Pfade sind Varianten von denen eine zur Zeit verwendet wird. Zur Visualisierung werden die Videodaten entweder über Netzwerk gesendet (3) oder über einen VGA-Monitor ausgegeben (4). Die Speicherung erfolgt mit einem Screenshot (2) oder einem Video (1).

Einzelbilder aus dem Speicher und stellt sie der GStreamer-Pipeline zur Verfügung. Um gleichzeitige Modifikationen des Bildes durch die GStreamer-Pipeline und den VDMA zu vermeiden, wird das Einzelbild aus dem, mit dem VDMA geteilten Speicherbereich von 503MB bis 506MB in den Heap der Anwendung kopiert. Die beim MJPEG Videostrom erforderlichen Zeitstempel werden vom *appsrc*-Element durch Setzen der Option *do-timestamp* erzeugt. Für die richtige Interpretation der Daten im geteilten Speicher durch appsrc wird das Videoformat mit der Option *caps* auf eine Auflösung von 640x480 mit Vier Byte RGBA, umgekehrte Byte-Reihenfolge (ABGR), eingestellt. Wenn im Fall eines Screenshots nur ein Bild erzeugt werden soll, wird dies dem *appsrc*-Element durch die Option *num-buffers* mit dem Wert 1 bekannt gegeben.

Erreicht ein Bild in der GStreamer-Pipeline das *appsink*-Element, wird von diesem das *new-buffer*-Event erzeugt, mit dem die Callback „new_video_buffer“ (video.c) aufgerufen wird. Diese kopiert die Bilder aus der GStreamer-Pipeline in den, mit dem VDMA geteilten, Speicherbereich von 503MB bis 506MB. Um zu verhindern, daß bei einem, durch zu hohe CPU-Last hervorgerufenen, Stau zu alte Bilder beschrieben werden, wird die Warteschlange des *appsink*-Elementes auf zwei Bilder begrenzt:

- Ein Bild, das gerade ausgelesen wird und
- ein neues Bild, das von der Kamera gespeichert wird.

Durch die Aktivierung der Option „drop“ arbeitet die Warteschlange als Ringpuffer und verwirft bei einem Überlauf alte Bilder.

Das Einzeichnen von Orientierungslinien wie dem Fahrzeugkoordinatensystem, markanten Punkten wie dem LAP, sowie von Mess- und Stellgrößen in das Bild, wird durch das *rsvgoverlay*-Element vorgenommen. Das Element verarbeitet nur Bilder im RGBA-Farbraum. Da das VDMA-Modul die Videodaten gleich im passenden Format im geteilten Speicherbereich liest und schreibt, entfällt eine Konvertierung. Das *rsvgoverlay*-Element verfügt über ein Attribut *data* für die zu rendernden Grafiken im Scalable Vector Graphics (svg) [57] Format.

Das Komprimieren der Daten in MJPEG und Übertragen der Videodaten über Netzwerk wird durch die Elemente *jpegenc*, *multipartmux* und *udpsink* durchgeführt. Da *jpegenc* an der Senke die Daten im YUV-Farbmodell [20] erwartet, werden diese vorher durch das *ffmpegcolorspace*-Element umgewandelt. Das resultierende Video (Abb. 5.13) wird per UDP an einen Empfänger-PC auf Port 5000 übertragen und mit einem Videoplayer wie vlc [54] mit der Adresse *udp://@ip:5000* abgespielt.

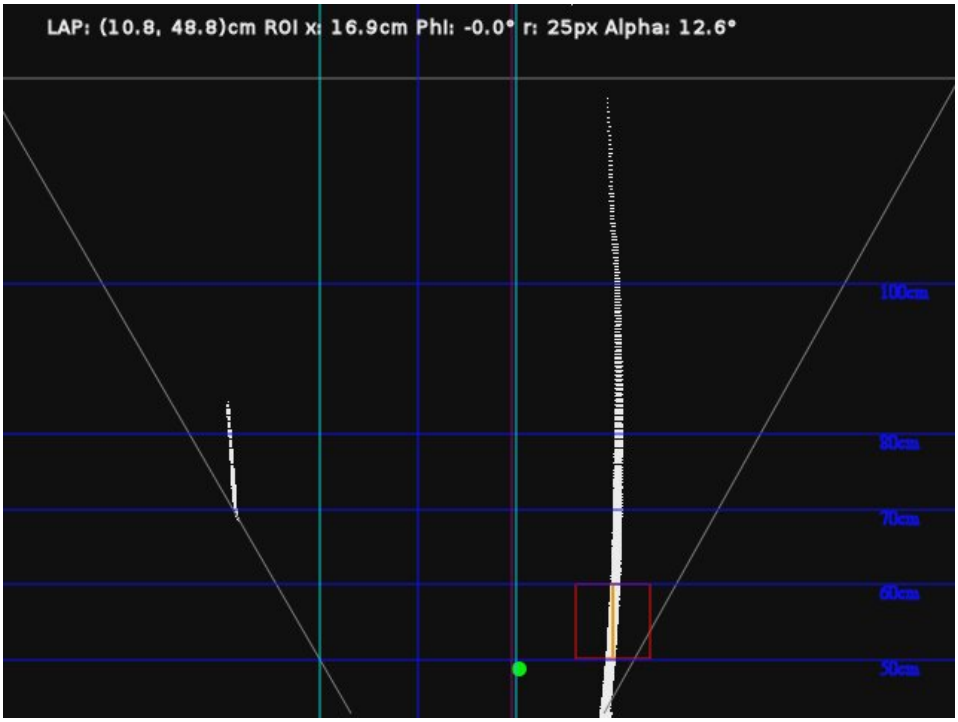



Abbildung 5.13: Straße nach der Projektiven-Transformation mit den Ergebnissen der Hough-Transformation (orange) und Spurführung. Es werden der LAP (grün), die ROI (rot), das Fahrzeugkoordinatensystem (blau), der Sollabstand d (violett) sowie die Kanten des Fahrzeugs (türkis) dargestellt.

Für die Speicherung von einem JPEG-Bild oder MJPEG/Avi-Video auf dem Fahrzeug wird das *filesink*-Element verwendet. Der Avi-Container für die MJPEGs wird durch das *avimux*-Element erzeugt.

### Periodische Ausführung der Visualisierungs-Threads

Das zeitgesteuerte Kopieren der Bilder in die GStreamer-Pipeline mit dem *appsrc*-Element erfolgt in einen Thread, der zehnmal pro Sekunde die Daten aus dem mit dem VDMA geteilten Speicher kopiert. Der Thread „Result Illustration“ erhält daraufhin ein Signal, mit dem Rendern der Grafiken zu beginnen (Abb. 5.14).

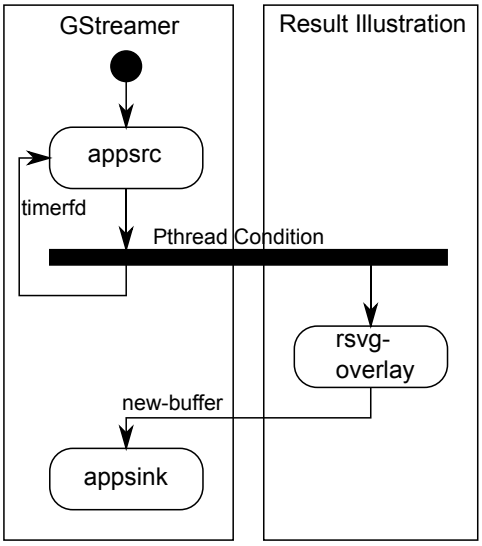



Abbildung 5.14: Aktivitätsdiagramm der GStreamer-Elemente zur Visualisierung der Fahrspurführung.

Die zeitgesteuerte Ausführung wird mit der *timerfd*-Schnittstelle von Linux vorgenommen, welches eine Implementierung der POSIX-Timer als Datei-Deskriptor ist. Der Umgang mit dem Timer erfolgt damit wie mit Dateien [49]. Vorteil gegenüber POSIX-Timern ist die komplette Behandlung der Timer im Thread, die POSIX-Timer arbeiten mit Signalen, die immer an den übergeordneten Prozess gesendet werden. Dazu wird der Timer mit *timerfd_create()* erstellt und das Zeitintervall mit der Struktur *itimerspec* durch *timerfd_settime()* eingestellt. Zum Warten auf ein Timer-Ereignis wird *read()* aufgerufen, dabei wird mit einem Acht-Byte-Wert gelesen wie viele Ereignisse seit dem letzten Aufruf aufgetreten sind (Listing 5.17). Der Typ *CLOCK_MONOTONIC* ist unabhängig von der Realtime-Clock, die Zeitumstellungen unterliegt.

Listing 5.17: Regelmäßige Ausführung eines Threads mit der *timerfd*-Schnittstelle.

```
int timer_fd;
struct itimerspec itval;
uint64_t missed;

// Setup timer
timer_fd = timerfd_create(CLOCK_MONOTONIC, 0);
itval.it_interval.tv_sec = 0;
itval.it_interval.tv_nsec = 100*1000*1000; // period=100ms
itval.it_value.tv_sec = 0;
itval.it_value.tv_nsec = 1; // nonzero value arms the timer
timerfd_settime(timer_fd, 0, &itval, NULL);

while(1) {
        copy_vdma_to_gstreamer();

        // Wait for the next timer event. If we have missed any the
        // number is written to "missed"
        read(timer_fd, &missed, sizeof(missed));
        if (missed > 1) fprintf(stderr, "gst receive thread missed %lld wakeups\n", missed-1);
}
```

Die Signalisierung zum Rendern der Grafiken erfolgt mit Pthread Conditions (Listing 5.18), dabei wartet der Thread solange, bis das Signal gesendet wird. Die Pthread Conditions werden durch einen Pthread Mutex beschützt, die als Futex (Fast Userspace Mutex) [19] implementiert sind. Da bei Futexen die Kommunikation mit dem Kernel nur bei Zustandsänderungen der Threads erfolgt, sind diese schneller als Mutexe [21].

Listing 5.18: Thread-Kommunikation mit Pthread Conditions

```
pthread_mutex_t draw_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t draw_cond = PTHREAD_COND_INITIALIZER;

// Thread 1
while(1) {
        pthread_mutex_lock(&draw_mutex);
        gst_app_src_push_buffer(appsrc, buf); // Buffer to GStreamer
        pthread_cond_signal(&draw_cond); // signal draw-thread to start
        pthread_mutex_unlock(&draw_mutex);
}

// Thread 2
while(1) {
        // wait for new image
        pthread_mutex_lock(&draw_mutex);
        pthread_cond_wait(&draw_cond, &draw_mutex);
        pthread_mutex_unlock(&draw_mutex);

        draw();
}
```

### Testen der Bildverarbeitungspipeline mit vorgegebenen Videodaten

Der MM2S-Kanal lässt sich auch verwenden, um Einzelbilder in die Bildverarbeitungspipeline zur Fahrspurerkennung einzuspeisen. Eine einmal aufgenommene Fahrt lässt sich damit für Testzwecke mit den gleichen Eingaben wiederholen. Dazu wird im Kontrollregister (15) des *AXI Configuration Connector* das *Video Feedback*-Bit (Register B.1 im Anhang) gesetzt, das die Bildverarbeitungspipeline von der Kamera abkoppelt und mit dem MM2S-Kanal des VDMA verbindet. Das Taktsignal der Kamera wird weiterhin verwendet. In der dazugehörigen GStreamer-Pipeline (Abb. 5.15) wird dafür das *decodebin*-Element verwendet, welches zur Dekodierung von Videos selbst eine Pipeline mit GStreamer-Elementen verwendet.

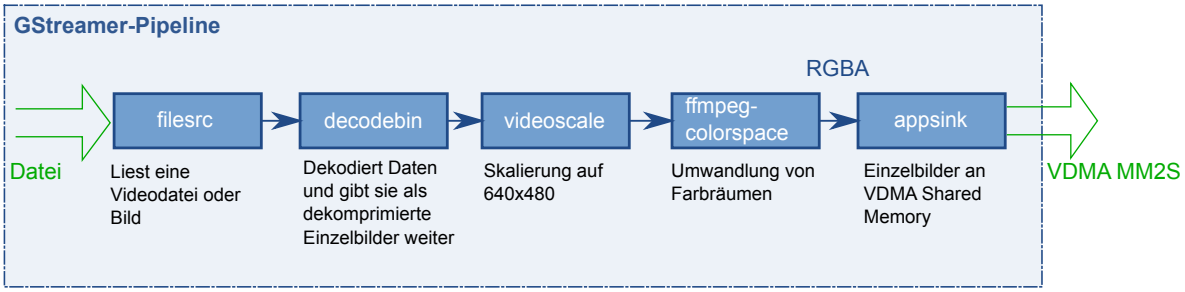



Abbildung 5.15: GStreamer-Pipeline um Videos aus dem Dateisystem an die Bildverarbeitungspipeline zu geben.

# 6 Ergebnisse und Messtechnische Analyse

Zur Analyse der RTL-Module und der Linux basierten Fahrspurführung (Abb. 2.3 in Kapitel 2.2) wurde:

- Der FPGA-Ressourcenbedarf von der Fahrspurerkennung ermittelt. Mit einer Timing-Analyse wurde sichergestellt, daß alle Signale innerhalb ihrer Taktperiode schalten.
- Die Konsistenz der Daten im MPSoC kontrolliert.
- Die zeitliche Ausführung der Spurführung unter Linux ohne Echtzeit-Patches überprüft.
- Eine durchgeführte Testfahrt zur Validierung des Systems ausgewertet.

## 6.1 FPGA Ressourcenbedarf und Timing-Analyse

Den FPGA-Ressourcenbedarf der IP-Cores (Abb. 2.1 in Kapitel 2.1) zeigt Tabelle 6.1. Dabei werden einige der Slices in der Tabelle aufgrund mehrfacher Verwendung doppelt gezählt. Insgesamt sind 8293 (62%) der 13300 Slices des Zynq-7000/Artix FPGAs belegt. Zum Vergleich wurden im *SAV Controller* vom Spartan-3A DSP FPGA des vorherigen Fahrzeugs 16711 Slices belegt [28].

| **Modul** | **IP** | **Slices** | **LUTs** | **BRAM** | **DSP48E1** |
|---|---|---|---|---|---|
| SAV Controller | sav_ctrl_0 | 7742 | 16641 | 37 | 58 |
| VDMA | axi_vdma_0 | 1773 | 2518 | 5 | 0 |
| AXI Configuration Connector | sav_axi_cfg_0 | 813 | 1476 | 0 | 0 |
| AXI HP Interconnect | axi_intconn_hp0 | 794 | 1300 | 3 | 0 |
| SAV Status Connector | sav_axi_reader_0 | 221 | 346 | 0 | 0 |
| AXI GP Interconnect | axi_intconn_gp0 | 160 | 226 | 0 | 0 |
| Kamera zu VDMA | axis2video_0 | 44 | 124 | 0 | 0 |
| VDMA zu SAV/VGA | xsvi2axi_0 | 13 | 8 | 1 | 0 |

Tabelle 6.1: RTL-Ressourcenbedarf der Fahrspurerkennung (Abb. 2.1 in Kapitel 2.1)

Die Differenz im Ressourcenbedarf zwischen den beiden FPGAs ergibt sich aus der Verlegung der Visualisierung sowie Spurführung von der Hardware in die Software und den 2 Bit breiteren Lookup-Tabellen (LUT) des Zynq-7000 FPGAs (6 Bit).

Für die Timing-Analyse wurden die Dateien aus der FPGA-Implementierung (Verzeichnis „implementation“, Prozesse *Translate*, *Map*, *Place and Route*) in den ISE Timing Analyzer importiert. Das Ergebnis wurde pro Taktfrequenz (Tabelle 4.1 in Kapitel 4.1.1) ausgewertet. Aus der Timing-Analyse ergibt sich ein maximaler Pixeltakt von 40.3MHz, der durch die Hough-Transformation begrenzt wird.

**clk_fpga_0: 100MHz AXI Takt**
Taktfrequenz für AXI PL IP-Cores mit einer Periodendauer von 10ns.
Längste Pfade: 10ns in den Logilink AXI4-Lite IP Interfaces des *AXI Configuration Connector* (Kapitel 4.3). Wobei dies die längsten Pfade für die Konfiguration mit 100MHz darstellen. In einer Konfiguration mit 80MHz AXI Takt wurde 12.5ns und mit 125MHz AXI Takt 8ns für die längsten Pfade angegeben.

**TS_clk_4: 27MHz Pixeltakt**
Taktfrequenz der Kamera mit einer Periodendauer von 37ns.
Längste Pfade: 17.3ns in der Arithmetik (Dividend zu Multiplizierer) der Projektiven Transformation (Kapitel 4.1.2). Mit diesen Pfaden ergibt sich ein maximal erreichbarer Pixeltakt von 57.8MHz.

**TS_clk_2: 54MHz doppelter Pixeltakt**
Taktfrequenz für übertaktete Pfade des *Pathfinder*-Moduls mit einer Periodendauer von 18.5ns.
Längste Pfade: 12.4ns (zweitlängste mit 12.2ns) in der Arithmetik (Multiplizierer zu Addierer) der Hough-Transformation (Kapitel 2.1). Mit diesen Pfaden ergibt sich ein maximal erreichbarer Pixeltakt von 40.3MHz.

**TS_clk_1: 108MHz vierfacher Pixeltakt**
Taktfrequenz für übertaktete Pfade des *Pathfinder*-Moduls mit einer Periodendauer von 9.3ns.
Längste Pfade: 5.4ns in den Dividierern der Projektiven Transformation (Kapitel 4.1.2). Mit diesen Pfaden ergibt sich ein maximal erreichbarer Pixeltakt von 46.3MHz.

**TS_clk_16: 6.75MHz PWM Takt**
Taktfrequenz für PWM-Module (Kapitel 4.4) des *SAV Controllers* mit einer Periodendauer von 148.1ns.
Längster Pfad: 7.1ns im *Receiver Switch* zum PWM-Ausgang der Motorsteuerung.

## 6.2 Synchronisation von Zugriffen auf geteilte Speicherbereiche

Zur Überprüfung der Konsistenz von Daten wurden die Speicherbereiche (Abb. 5.2 in Kapitel 5.2.1) und die darauf zugreifenden Threads und Peripherie-Elemente identifiziert (Tabelle 6.2).

| **Speicherbereich** | **PL** | **PS Threads** | | | |
|---|---|---|---|---|---|
| | | **Datalogging** | **Spurführung** | **GStreamer** | **Result Illustration** |
| VDMA S2MM Memory | w | - | - | r | - |
| VDMA MM2S Memory | r | - | - | w | - |
| VDMA Register | rw | - | - | rw | - |
| AXI Config. Conn. | r | rw | rw | - | r |
| AXI Status Conn. | w | r | r | - | r |
| sav_road_distance | - | w | r | - | r |
| sav_lad | - | w | r | - | - |
| sav_lap_x/y | - | r | rw | - | r |
| sav_alpha | - | r | w | - | - |
| GStreamer Bild | - | - | - | rw | w |

Tabelle 6.2: Lesende (r) und schreibende (w) Zugriffe der PL und der Prozessor-Threads (Kapitel 5) auf die geteilten Speicherbereiche.

Die von der PL und den Threads des PS geteilten Variablen und Speicherbereiche sind:

- **VDMA S2MM/MM2S Memory**: Die oberen 12MB des Arbeitsspeichers zum Austausch der Videodaten zwischen PS und PL werden nicht synchronisiert (Kapitel 5.2.2).
- **VDMA Register**: Die von GStreamer gestarteten Callbacks lesen und schreiben die Konfigurationsregister des VDMA IP-Cores (Abb. 4.7 in Kapitel 4.2). Statusregister werden vom VDMA beschrieben.

- **AXI Configuration Connector**: Die Parameter (Tabelle 4.5 in Kapitel 4.3) des *SAV Controllers* werden im Kommunikations-Thread des Dataloggers und im Fahrspurführungs-Thread geschrieben. Da beide nicht dieselben Register schreiben, gibt es hier keine Konflikte. Die 32-Bit Softwareregister werden in einem Takt gelesen und geschrieben, weswegen keine weitere Synchronisation erforderlich ist.
- **AXI Status Connector**: Softwareregister (Tabelle 4.6 in Kapitel 4.3), die in einem Takt von den Threads gelesen und der PL geschrieben werden. Da ein Schreibvorgang nur von der PL erfolgt, gehen beim Schreiben keine Werte verloren, wie sie bei Schreibvorgängen von zwei unterschiedlichen Quellen auftreten.
- **sav_road_distance**: Globale 32-Bit Variable für den Abstand D von der Fahrspur (Abb. 5.8 in Kapitel 5.3.3), die in einem Takt [6] gelesen und geschrieben wird. Wird vom Kommunikations-Threads des Dataloggers geschrieben.
- **sav_lad**: Globale 32-Bit Variable für die Look-Ahead-Distance (Abb. 5.8 in Kapitel 5.3.3), die in einem Takt gelesen und geschrieben wird. Wird nur vom Kommunikations-Threads des Dataloggers geschrieben.
- **sav_lap_x/y**: Horizontale und vertikale Koordinaten des Look-Ahead-Points (Abb. 5.8 in einer Globalen Variable, die nur zusammen gültig sind und deswegen durch einen Pthread Mutex beschützt werden.
- **sav_alpha**: Globale 32-Bit Fließkommazahl für den Stellwinkel der Lenkung (Kapitel 4.4). Fließkommazahlen mit einfacher und doppelter Genauigkeit werden in einem Takt in die FPU geladen und gespeichert (VMOV-Operation) [4]. Wird vom dem Thread der Spurführung geschrieben.
- **GStreamer Bild**: Das Bild der GStreamer-Pipeline (Kapitel 5.3.5) wird vom *appsrc*-Element in die GStreamer-Pipeline geschrieben, vom *rsvgoverlay*-Element bearbeitet und von anderen Elementen der Pipeline gelesen. Die Bearbeitung mit dem *rsvgoverlay*-Element erfolgt in einem von der Pipeline unabhängigem Thread, dem mit Pthread Conditions signalisiert wird, mit dem Schreibvorgang zu beginnen.

## 6.3 Auslastung der SMP-Prozessoren

Die mit *gettimeofday()* gemessene Laufzeit für die Durchführung der Spurführung mit Pure Pursuit (Kapitel 5.3.3) bei Verwendung der FPU beträgt 0.4ms. Da die Berechnung für jedes Bild, also 60x in der Sekunde durchgeführt wird, belegt sie ca. 3% ($60s^{-1} * 0.4ms$) der Zeit auf einem Prozessor, was durch Messungen mit dem Programm *top* bestätigt wurde.

Die Laufzeit für die Visualisierung der Fahrspurerkennung und Fahrspurführung (Kapitel 5.3.5) beträgt 22ms, wobei 15ms davon zum Rendern der Vektorgrafik benötigt werden.

Die Verteilung der Threads auf die beiden Prozessoren im SMP-System wurde mit dem *ps*-Kommando (Listing 6.1) gemessen, das für die Messung alle 50ms aufgerufen wurde.

Listing 6.1: Messen der Thread-Verteilung auf die Prozessoren mit dem Kommando *ps*.

```
# $(pidof framegrabber) = Process ID
# Options:
# psr  = processor the thread is currently assigned to
# comm = thread name
> ps -p $(pidof framegrabber) -L -o psr,comm
```

In der Messung (Abb. 6.1) zeigt sich, daß der Thread für die Spurführung sowie der Main-Thread auf CPU-0 und der Kommunikations-Thread auf CPU-1 ausgeführt werden ohne zu wechseln. Der *Display Video*-Thread, der die Videodaten in die GStreamer-Pipeline kopiert und der Thread zur Berechnung der Visualisierung wechseln zwischen beiden Prozessoren.

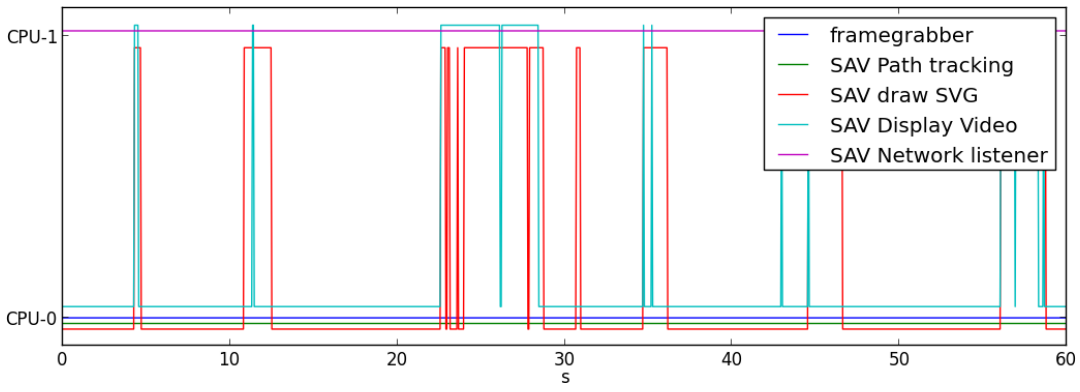



Abbildung 6.1: Verteilung der Threads in der Zeit (Abszisse) auf die beiden SMP Prozessoren (Ordinate).

## 6.4 Jitter und Zeitverhalten der Threads im Vergleich zur Bildfrequenz

Zur Überprüfung der zeitlich exakten Ausführung der Spurführung mit der *SCHED_FIFO* Scheduler-Regel (Kapitel 5.3.2) unter Linux wird bei jeder Berechnung ein GPIO-Pin getoggelt (Listing C.2 in Anhang C) und auf einem Oszilloskop dargestellt (Abb. 6.2). Dabei wurde das geforderte Intervall von 16.67ms für die durch den Interrupt *HT_DONE* (Kapitel 2.1) getriggerte Ausführung der Spurführung im Userspace eingehalten.

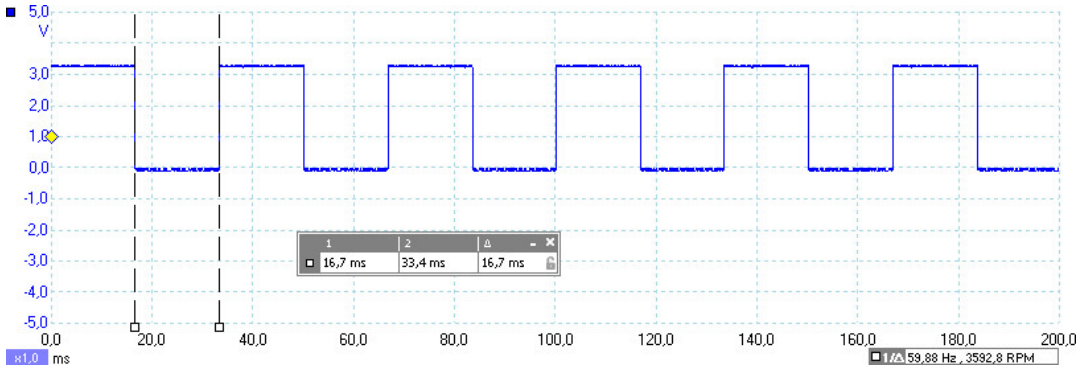



Abbildung 6.2: Ergebnis der Messung zur Einhaltung der Periodendauer von 16.67ms durch den Linux Scheduler.

Die Messung wird mit 100 zusätzlichen, Busy-Loop ausführenden, Threads wiederholt. Die erhöhte Last auf den Prozessoren und dem Scheduler führte zu Abweichungen in der regulären Ausführung (Abb 6.3).

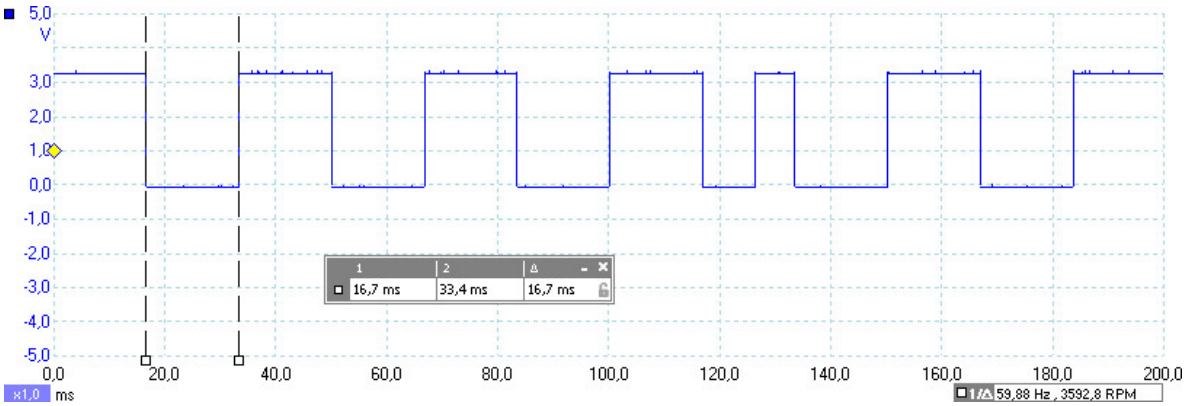



Abbildung 6.3: Ergebnis der Messung zur Einhaltung der Periodendauer von 16.67ms durch den Linux Scheduler bei 100 parallel laufenden Threads.

Pro Ergebnis der Hough-Transformation wird der Interrupt *HT_DONE* generiert, der im Linux-Kernel bearbeitet wird und dem Programm im Userspace signalisiert, mit der Berechnung der Spurführung zu beginnen (Kapitel 5.2.1). Für die Messung der Latenz zwischen der Beendigung der Bildverarbeitung und Berechnung der Spurführung wird ein weiterer GPIO-Pin von der PL bei der Generierung des Interrupts getoggelt. Ergebnis der Messung ist eine Latenz zwischen 125µs und 150µs (Abb. 6.4), wobei etwa 30µs zum Toggeln des GPIO-Pin abzuziehen sind. Da das UIO-Framework bei einem wartenden Programm bei Auftreten eines Interrupts einen Kontextwechsel durchführt, ist die Latenz kleiner als die konfigurierte Zeitspanne (*CONFIG_HZ*) von 100Hz für den Scheduler Tick.

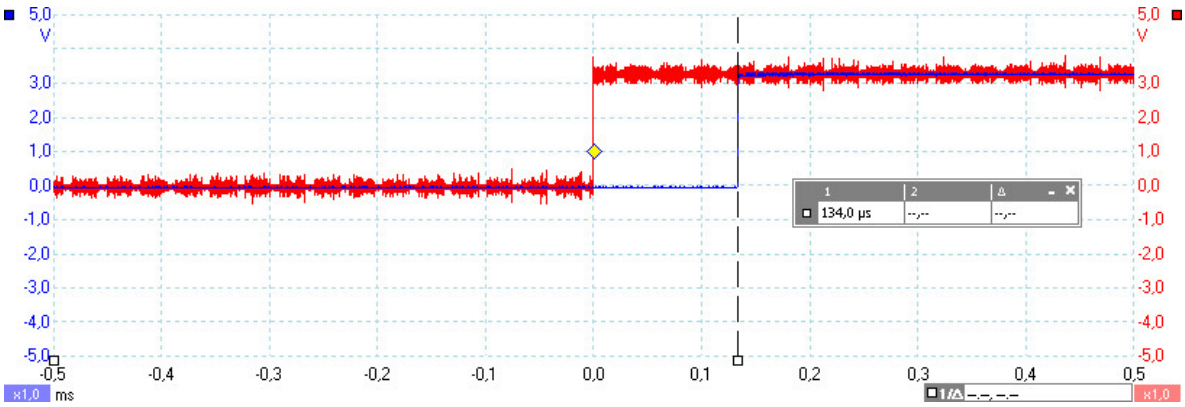



Abbildung 6.4: ca. 135µs Latenz zwischen Auftreten des Interrupts (rot) und Bearbeitung im Userspace (blau).

## 6.5 Testfahrt und Auswertung

Zur Validierung des Systems wurden Fahrten auf einer Teststrecke durchgeführt und ausgewertet. Die Strecke enthält mit einer Linkskurve, einer Rechtskurve und drei Geraden alle Kurventypen (Abb 6.5). Für die Auswertung wurde die Teststrecke in sechs Sektoren unterteilt, jeder Sektor enthält eine Fahrsituation.

Die Testfahrt wurde mit dem *Java Remote Tool* (Kapitel 5.3.4) aufgezeichnet (Abb. 6.6), wobei die Größen mit folgenden Einheiten dargestellt sind:

- Winkel $\Phi$ der Straße zum Fahrzeug in Grad.
- Lenkeinschlag $\alpha$ in Grad.
- Horizontale Position der ROI in cm.

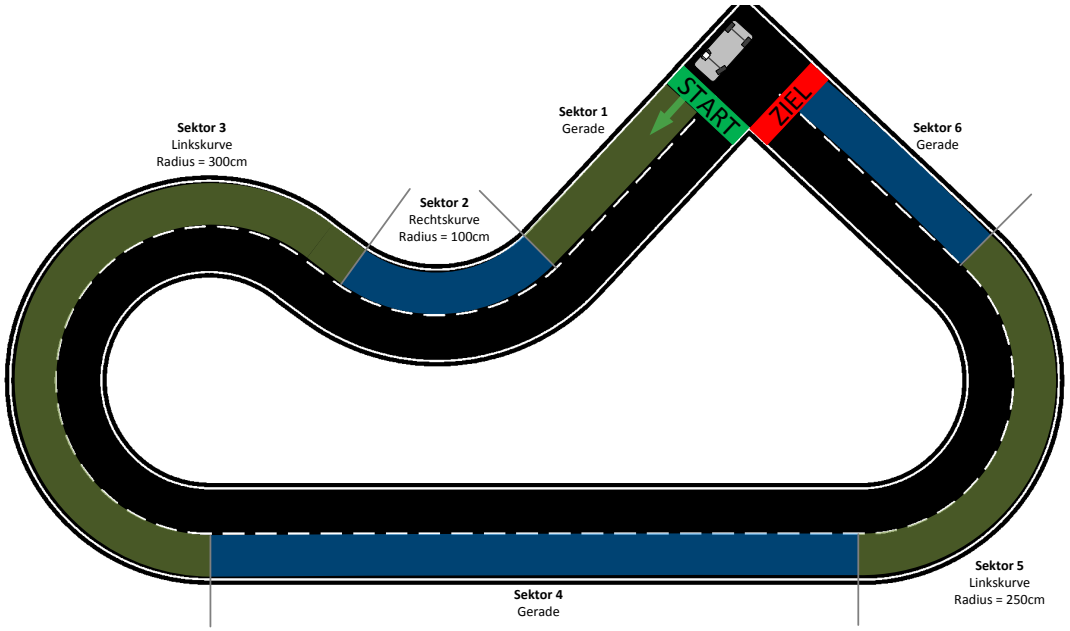



Abbildung 6.5: Die Teststrecke ist in sechs Sektoren unterteilt. [28]

- Die Lotlänge r in Pixel.

Ein positiver Winkel bedeutet eine Rechtskurve bzw einen Lenkeinschlag nach rechts, ein negativer Winkel bedeutet eine Linkskurve bzw einen Lenkeinschlag nach links.

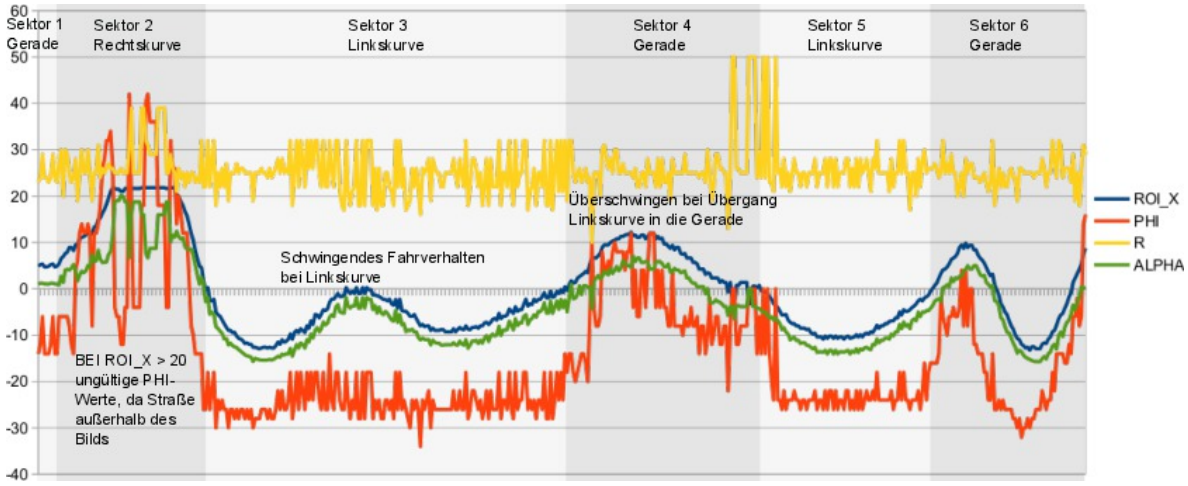



Abbildung 6.6: Logdaten bei einer Fahrt auf der Teststrecke mit $ROI_Y = 60cm$, $LAD = 50cm$ und $D = 10cm$ in der Zeit (Abszisse mit 10 Messungen pro Sekunde).

Bei den Testfahrten folgte das Fahrzeug durch Stellung des Lenkeinschlags allen Kurventypen. Auffällig sind dabei häufige Pegelausschläge der HT-Größen $r$ und $\Phi$, die durch eine zu breite Fahrspur und dadurch hohe Anzahl möglicher Geradenapproximationen (Abb.2.2 in Kapitel 2.1) hervorgerufen wird.

Folgendes Fahrverhalten wurde beobachtet:

2. **Sektor**: Bei der Rechtskurve verschwindet die Fahrspur rechts aus dem Bild, was zu ungültigen Ergebnissen der Hough-Transformation führt.
3. **Sektor**: In der Linkskurve ist ein schwingendes Fahrverhalten erkennbar, da bei konstanter Kurve der Lenkwinkel $\alpha$ Änderungen innerhalb von $\pm 8°$ aufweist.
4. **Sektor**: Beim Übergang von der Linkskurve in die Gerade ist ein Überschwingen erkennbar, weswegen der Lenkwinkel nicht Null beträgt. Am Ende der Geraden erscheint die Wand des Testraums im Kamerabild, was den Weißanteil im Gesamtbild erhöht und deswegen den Schwellwert der adaptiven Binarisierung ändert. Dies führt zu einer Reduzierung von Vordergrund-Pixeln und damit zu ungültigen HT-Ergebnissen. Der Wert von $\Phi$ beträgt an diesen Stellen Null.

Die Reduzierung der Fahrspurbreite lässt sich durch die folgenden Maßnahmen erreichen:

- Hinzufügen von weiteren horizontalen Erosionsfiltern.
- Größere Neigung der Kamera in Richtung des Horizonts, was zu einer geringeren Pixel-Auflösung in der Nähe des Fahrzeugs führt.

Als Strategien gegen das Verschwinden der Fahrspur aus dem Bild bei Kurven eignet sich ein größerer Wert für die ROI und die LAD oder die Verwendung eines Weitwinkelobjektives.

Um den Einfluss von Wänden und auf der Fahrbahn vorhandenen Gegenständen auf die Berechnung des Schwellwertes der adaptiven Binarisierung zu verringern, sollte dieser nur innerhalb der ROI berechnet werden.

Die entwickelte Lösung fährt reproduzierbar vom Start bis zum Ziel. Für verbesserte Fahreigenschaften sollten Testfahrten mit den Werten aus der Parameterstudie ([28], Anhang D) durchgeführt werden. Damit die ROI bei einer veränderten vertikalen Position ($ROI_Y = 77.4cm$) weiterhin in der Mitte des Bildes liegt, ist eine andere Neigung der Kamera und damit verbundene neue Berechnung der Projektiven Transformation erforderlich.

# 7 Zusammenfassung

Mit dieser Masterarbeit wurde das SoC basierte Spurführungssystem des autonomen Fahrzeugs auf die kombinierte FPGA-MPSoC-Plattform Zynq-7000 Z-7020 portiert, die über zwei auf 667MHz getaktete Dual-Core ARM Cortex-A9 Prozessoren verfügt. Als Betriebssystem kommt ein Debian Linux im SMP-Betrieb zum Einsatz, das über einen mehrstufigen Bootvorgang von SD-Karte geladen wird und die Peripherie initialisiert. Zur Aufnahme der Fahrspur wird die per I2C-Bus parametrisierte Videology 24B7525A CMOS-Kamera verwendet, die Bilder mit 640x480 Pixel bei 60 Bildern pro Sekunde liefert.

Die Bildverarbeitungspipeline zur Erkennung der Fahrspur und die PWM-Generatoren für die Aktoren sind in der FPGA-Logik realisiert. Die Parametrisierung der FPGA-Komponenten und das Lesen von Berechnungsergebnissen der Bildverarbeitungskette wird über 32-Bit Softwareregister vorgenommen, die über AXI4-Lite mit den Prozessoren verbunden sind. Unter Linux werden die Memory Mapped Softwareregister mit dem UIO-Subsystem in den virtuellen Adressraum der Anwendung abgebildet.

Die Fahrspurführung wird mit dem Pure Pursuit Algorithmus auf den Prozessoren mit der Echtzeit Strategie *SCHED_FIFO* des Linux-Schedulers alle 16.67ms berechnet. Der Thread wird von der Bildverarbeitungspipeline per Interrupt aufgeweckt, der durch das UIO-Subsystem an das Programm im Userspace weitergegeben wird.

Zur Visualisierung der Fahrspurerkennung und Fahrspurführung wird das Kamera-Signal in das AXI4-Stream Protokoll umgewandelt und mit dem VDMA IP-Core in den DDR3-Arbeitsspeicher kopiert. Von dort werden die Einzelbilder mit dem Multimedia-Framework GStreamer gelesen und bearbeitet. Im svg-Format werden Orientierungslinien, markante Punkte sowie Mess- und Stellgrößen in das Bild eingezeichnet. Erfolgt die Ausgabe über den VGA-Ausgang, werden die Bilder zurück in den Arbeitsspeicher geschrieben, von dort mit dem VDMA IP-Core wieder in ein Stream-basiertes Protokoll umgewandelt und an den *VGA-Controller* gegeben. Alternativ werden die Bilder als komprimiertes MJPEG-Video über einen per USB angeschlossenen W-LAN-Adapter an einen PC gesendet.

Das Aufzeichnen von Mess- und Stellgrößen aus dem Fahrbetrieb, sowie die Parametrisierung des Fahrzeugs erfolgt mit einem Java-Programm über W-LAN.

Zur Validierung ist das entwickelte Fahrzeug reproduzierbar eine Teststrecke von Start bis zum Ziel gefahren.

## 7.1 Ausblick

Zur Weiterentwicklung des Fahrzeugs stehen noch 75% der Rechenzeit von beiden Prozessoren und 40% FPGA-Ressourcen zur Verfügung. Zu den Erweiterungen, die das Fahrverhalten verbessern und die Systemfunktionalität komplettieren, gehören:

- Messung der Akkuspannungen zur Erkennung des Akku-Ladezustands.
- Verwendung der Prozessoren in der AMP-Betriebsart, bei dem auf dem zweiten CPU-Kern zur Einhaltung der Antwortzeiten ein Echtzeit-Betriebssystem eingesetzt wird.
- Für eine zukünftige Integration eines Antikollisionssystems wurde ein URG-04LX Laserscanner zur Objekterkennung über USB angebunden, der noch nicht in die Software integriert ist.
- Zur Vermeidung des Tearing-Effekts durch den gleichzeitigen Zugriff von VDMA IP-Core und GStreamer auf den Speicher kann der VDMA IP-Core um einen weiteren Zwischenspeicher (*C_NUM_FSTORES*) erweitert werden. Die CPU-lastige JPEG-Komprimierung lässt sich im FPGA durchführen. Für eine gleichzeitige Ausgabe der Bilder über VGA und MJPEG kann das *tee*-Element in die GStreamer-Pipeline eingebaut werden, das die Einzelbilder an mehrere Senken gibt.
- Zur Erhöhung der horizontalen Breite des Kamerabildes kann die Kamera auf die maximale Auflösung von 752x480 Pixel eingestellt werden. Zum Fahren von höheren Geschwindigkeiten lässt sich die Bilfrequenz der Kamera verdoppeln oder vervierfachen, dies reduziert jedoch die Auflösung um die Hälfte bzw ein Viertel (*Binning*). Änderungen an der Auflösung erfordern eine Anpassung der Bildverarbeitungspipeline.
- Um den Einfluss von Wänden und auf der Fahrbahn vorhandenen Gegenständen auf die Berechnung des Schwellwertes der adaptiven Binarisierung zu verringern, sollte dieser nur innerhalb der ROI berechnet werden.
- Auftretende Bild-Artefakte durch Spiegelungen auf der Fahrbahn lassen sich durch das „AEC/AGC Desired Bin“-Register (0xA5) der Kamera beeinflussen.

# Literaturverzeichnis


[1] Information technology - Portable Operating System Interface (POSIX) Operating System Interface (POSIX). In: *ISO/IEC/IEEE 9945 (First edition 2009-09-15)* (2009), S. c1–3830

[2] ARM: *AMBA Open Specifications*. – URL http://www.arm.com/products/system-ip/amba/amba-open-specifications.php

[3] ARM: *Cache Operations*. – URL http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0198e/I1014942.html

[4] ARM: *CortexTM-A9 Floating-Point Unit - Technical Reference Manual*. – URL http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0408e/index.html

[5] ARM: *AMBA4 AXI4-Stream Specification*. : , March 2010

[6] ARM: *Cortex-A Series Programmer's Guide*. first. ARM, 2011

[7] ARM: *AMBA AXI and ACE Protocol Specification*. : , February 2013

[8] Avnet: *ZedBoard Hardware User's Guide*. – URL http://www.zedboard.org/documentation

[9] Bellard, Fabrice: *QEMU Virtuelle Machine*. – URL http://www.qemu.org

[10] Billauer, Eli: *Interrupt definitions in DTS (device tree) files for Xilinx Zynq-7000/ARM*. – URL http://billauer.co.il/blog/2012/08/irq-zynq-dts-cortex-a9/

[11] Crowley, Patrick: The future in your pocket. In: *SIGCOMM Comput. Commun. Rev.* 38 (2008), März, Nr. 2, S. 61–64. – URL http://doi.acm.org/10.1145/1355734.1355744. – ISSN 0146-4833

[12] Debian-Projekt: *Cross-installing Debian using debootstrap*. – URL http://wiki.debian.org/EmDebian/CrossDebootstrap

[13] Debian-Projekt: *Webauftritt der Debian GNU/Linux Distribution.* – URL http://www.debian.org/

[14] Delvare, Jean u. a.: *Linux I2C dev-Interface.* – URL http://www.kernel.org/doc/Documentation/i2c/dev-interface

[15] Denx Software Engineering: *Das U-Boot - the Universal Boot Loader.* – URL http://www.denx.de/wiki/U-Boot

[16] devicetree.org: *Device Tree Usage.* – URL http://devicetree.org/Device_Tree_Usage

[17] Digilent Inc.: *Linux Kernel Repository for digilent boards.* – URL https://github.com/Digilent/linux-digilent

[18] Digilent Inc.: *ZedBoard Out Of Box SD Card image and source.* – URL http://www.zedboard.org/node/241

[19] Drepper, Ulrich: *Futexes Are Tricky*

[20] fourcc.org: *YUV pixel formats.* – URL http://www.fourcc.org/yuv.php#NV12

[21] Franke, Russell ; Kirkwood: Fuss, Futexes and Furwocks: Fast Userlevel Locking in Linux. In: *Ottawa Linux Symposium*, 2002

[22] GTK+ Team: *GLib Reference Manual.* – URL http://developer.gnome.org/glib/

[23] Hitec: *Bedienungsanleitung eines HFP-20 Modellbauservos von Hitec.* – URL http://www.hitecrc.de/store/dokumente/HFP-20.pdf

[24] Hokuyo Automatic® co.: *Communication Protocol Specification For SCIP2.0.* – URL http://www.hokuyo-aut.jp/02sensor/07scanner/download/data/URG_SCIP20.pdf

[25] Hokuyo Automatic® co.: *Hochschule URG-04LX Laserscanner.* – URL http://www.hokuyo-aut.jp/02sensor/07scanner/download/products/urg-04lx/

[26] Jestel, Andre: Projekt 2 -WiSe 2011/12 Kalibrierung einer Bildverarbeitungsplattform für eine automatische Fahrspurführung / HAW Hamburg, Department Informatik. 2012. – Master-Ausarbeitung Projekt 2

[27] Jestel, Andre: *Software-Partitionierung einer Laserscannerbasierten Objekterkennung auf einer MPSoC-Plattform mit Android*, Hochschule für Angewandte Wissenschaften - Hamburg, Master Thesis, 2013

[28] Johannsen, Benedikt: *Hardware/Software-Codesign für ein SoC-basiertes Spurführungssystem*, Hochschule für Angewandte Wissenschaften - Hamburg, Master Thesis, 2013

[29] Johnson, Jeff: *Generating Clock Domain Crossing FIFOs*. – URL http://www.fpgadeveloper.com/2009/09/generating-clock-domain-crossing-fifos.html

[30] Jones, M. T.: *Anatomy of real-time Linux architectures*. – URL http://www.ibm.com/developerworks/library/l-real-time-linux/

[31] Jones, M. T.: *Inside the Linux 2.6 Completely Fair Scheduler*. – URL http://www.ibm.com/developerworks/library/l-completely-fair-scheduler/

[32] Khaddour, M. ; Wang, Z. ; Hammami, O.: Implementing block cipher on embedded multiprocessors platform. In: *Multimedia Computing and Systems, 2009. ICMCS '09. International Conference on*, april 2009, S. 193 –198

[33] Kilts, Steve: *Advanced FPGA Design: Architecture, Implementation, and Optimization*. 1. Aufl. John Wiley & Sons, 2007. – ISBN 0470054379

[34] Kirschke, Marco: *FPGA-basierte MPSoC-Plattform zur Integration eines Antikollisionssystems in die Fahrspurführung eines autonomen Fahrzeugs*, HAW Hamburg, Department Informatik, Diplomarbeit, 2012

[35] Koch, Hans J.: *Userspace I/O drivers in a realtime context*

[36] Koch, Hans J.: *The Userspace I/O HOWTO*. – URL https://www.kernel.org/doc/htmldocs/uio-howto.html

[37] Krten, Rob: *Getting Started with QNX Neutrino – A Guide for Realtime Programmers*. 2. Aufl. PARSE Software Devices, 2001. – ISBN 0968250114

[38] Likely, Grant u. a.: *Linux GPIO Interfaces*. – URL http://www.kernel.org/doc/Documentation/gpio.txt

[39] Lin, Tzong-Yen ; Hung, Yu-Ting ; Chang, Rong-Guey: Efficient Hardware/Software Partitioning Approach for Embedded Multiprocessor Systems. In: *VLSI Design, Automation and Test, 2006 International Symposium on*, april 2006, S. 1 –4

[40] Lucero, James ; Arbel, Ygal: *XAPP792: Designing High-Performance Video Systems with the Zynq-7000 All Programmable SoC.* : Xilinx (Veranst.), October 2012. – URL http://www.xilinx.com/support/documentation/application_notes/xapp792-high-performance-video-zynq.pdf

[41] Mellert, Dennis: *Bachelorarbeit Modellierung einer Video-basierten Fahrspurerkennung mit dem Xilinx System Generator für eine SoC-Plattform*, HAW Hamburg, Department Informatik, Diplomarbeit, 2010

[42] Molnar, Ingo u. a.: *Real-Time Linux Wiki.* – URL https://rt.wiki.kernel.org

[43] OpenWRT Entwickler: *OpenWRT Distribution.* – URL http://openwrt.org/

[44] PetaLogix: *UserSpace IO (UIO) Introduction.* – URL http://www.petalogix.com/support/kb/uio_howto

[45] Reichardt, Jürgen ; Schwarz, Bernd: *VHDL-Synthese: Entwurf digitaler Schaltungen und Systeme.* 6. Aufl. Oldenbourg Wissenschaftsverlag, 2012. – ISBN 3486581929

[46] Riemenschneider, Frank: *(Fast) eineiige Zwillinge - Alteras Antwort auf Zynq.* 2012. – URL http://www.elektroniknet.de/bauelemente/technik-know-how/halbleiter/article/88868/0/Fast_eineiige_Zwillinge_-_Alteras_Antwort_auf_Zynq/

[47] Salathé, Jasper: *Integration einer CMOS-Kamera mit parallelem Interface in ein System-on-Chip basiertes Spurführungssystem*, Hochschule für Angewandte Wissenschaften - Hamburg, Bachelor Thesis, 2012

[48] Schneider, Christian: *Ein SoC-basiertes Fahrspurführungssystem*, HAW Hamburg, Department Informatik, Diplomarbeit, 2011

[49] Simmonds, Chris: *Over and over again: periodic tasks in Linux.* – URL http://www.embedded-linux.co.uk/tutorial/periodic_threads

[50] Texas Instruments: *LM2576 SIMPLE SWITCHER® 3A Step-Down Voltage Regulator.* – URL http://www.ti.com/product/lm2576

[51] Timoschenko, Alexander: *Bachelorarbeit Implementierung einer Geschwindigkeitsregelung als Prozessor-Element auf einer SoC-Plattform für ein autonomes Fahrzeug*, HAW Hamburg, Department Informatik, Diplomarbeit, 2011

[52] USB Implementers Forum, Inc: *Universal Serial Bus Class Definitions for Communication Devices.* – URL http://www.usb.org/developers/devclass_docs/usbcdc11.pdf

[53] USB Implementers Forum, Inc: *USB 2.0 Standard.* – URL http://www.usb.org/developers/docs/

[54] VideoLAN-Team: *VLC media player.* – URL http://www.videolan.org/vlc/

[55] Videology Imaging Solutions inc.: *Application Note 24B752XA Wide VGA B&W CMOS Board Camera.* – URL http://www.videologyinc.com/media/products/application%20notes/APN-24B752XA.pdf

[56] Walthinsen, Erik u. a.: *Multimedia-Framework Gstreamer.* – URL http://www.gstreamer.net

[57] World Wide Web Consortium: *Scalable Vector Graphics (SVG) 1.1 Specification.* – URL http://www.w3.org/TR/2003/REC-SVG11-20030114/

[58] Xilinx: *AR51948: Cache Flush.* – URL http://www.xilinx.com/support/answers/51948.htm

[59] Xilinx: *UG585: Zynq-7000 Technical Reference Manual.* – URL http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[60] Xilinx: *UG821: Zynq-7000 All Programmable SoC Software Developers Guide.* – URL http://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf

[61] Xilinx: *Zynq-7000 Extensible Processing Platform.* – URL http://www.xilinx.com/Zynq

[62] Xilinx: *DS485: Digital Clock Manager (DCM) Module - Product Specification.* : , April 2009. – URL http://www.xilinx.com/support/documentation/ip_documentation/dcm_module.pdf

[63] Xilinx: *UG625: Constraints Guide.* : , December 2009. – URL http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/cgd.pdf

[64] Xilinx: *Bridging Xilinx Streaming Video Interface with the AXI4-Stream Protocol.* : , February 2012. – URL http://www.xilinx.com/support/documentation/application_notes/xapp521_XSVI_AXI4.pdf

[65] Xilinx: *DS765: LogiCORE IP AXI4-Lite IPIF - Product Specification.* : , January 2012. – URL http://www.xilinx.com/support/documentation/ip_documentation/axi_lite_ipif_ds765.pdf

[66] Xilinx: *PG020: LogiCORE IP AXI Video Direct Memory Access Product Guide.* : , July 2012. – URL http://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v5_02_a/pg020_axi_vdma.pdf

[67] Xilinx: *PG057: LogiCORE IP FIFO Generator v9.3 - Product Guide.* : , December 2012. – URL http://www.xilinx.com/cgi-bin/docs/ipdoc?c=fifo_generator;v=v9_3;d=pg057-fifo-generator.pdf

[68] Xilinx: *UG081: MicroBlaze Processor Reference Guide.* : , July 2012. – URL http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_2/mb_ref_guide.pdf

[69] Xilinx: *UG761: AXI Reference Guide.* : , April 2012. – URL http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/v14_1/ug761_axi_reference_guide.pdf

[70] Xillybus: *A Tutorial on the Device Tree (Zynq).* – URL http://xillybus.com/tutorials/device-tree-zynq-1

[71] Yue-li, Hu ; Qian, Ding: Design of an architecture for multiprocessor system-on-chip (MPSoC). In: *High Density Microsystem Design and Packaging and Component Failure Analysis, 2006. HDP'06. Conference on*, june 2006, S. 63 –66

[72] zedboard.org: *ZedBoard: Zynq Evaluation & Development Board.* – URL http://www.zedboard.org

# Tabellenverzeichnis

# Abbildungsverzeichnis

# Listings

# Glossar

Abb Abbildung

ACP Accelerator Coherency Port

AMBA Advanced Mikrocontroller Bus Architecture

AMP Asymmetric multiprocessing

API Application Programming Interface

ARM 32-Bit RISC Prozessor Architektur

ASIC Application specific integrated circuit

AVI Audio Video Interleave

AXI Advanced eXtensible Interface Bus

BSP Board Support Package

CFS Completely Fair Scheduler

DCM Digital Clock Manager

DDR3-SDRAM Double Data Rate Synchronous Dynamic Random Access Memory

DMA Direct Memory Access

DSP Digitaler Signalprozessor

EMIO Extended-MIO

EOL End of Line

FAUST Fahrerassistenz- und autonome Systeme

FIFO First In - First Out

FPGA Field Programmable Gate Array

FPU Fließkommaeinheit

Futex Fast Userspace Mutual Exclusion

GP General Purpose

GPIO General Purpose Input/Output

GST Multimedia-Framework GStreamer

HLOS High-Level Operating System

HP High Performance

HPEC High Performance Embedded Computing

HT Hough-Transformation

IP (Adresse) Internet Protocol Address

IP (Core) Intellectual Property

IPIF IP-Core Interface

ISE Xilinx Integrated Software Environment

JPEG Joint Photographic Experts Group

LAD Look-Ahead-Distance

LAN Local Area Network

LAP Look-Ahead-Point

LED Leuchtdiode

LSB Least Significant Bit

LUT Look-up table

MAC Media Access Control

MIO Multiplexed I/O Pins

MJPEG Motion JPEG

MP Mehrkernprozessor

MPSoC Mehrkernprozessor System-on-Chip

Mutex Mutual Exclusion

OCM On Chip Memory

OLED Organische Leuchtdiode

OSI Open Systems Interconnection Model

PL Programmable Logic

PLB Processor Local Bus

PLL Phase-Locked Loop

POSIX Portable Operating System Interface

PS Processing System

PT Projektive Transformation

PWM Pulsweitenmodulation

RGB Red Green Blue Farbraum

RGBA Red Green Blue Alpha Farbraum

ROI Region-of-Interest

RTL Register transfer level, High-level Darstellung von Schaltkreisen.

RTOS Echtzeitbetriebssystem

SAV System-On-Chip Autonomous Vehicle

SCU Snoop Control Unit

SDK Software Development Kit

SMP Symmetric multiprocessing

SoC System-on-Chip

SOF Start of Frame

SSH Secure Shell

TCP Transmission Control Protocol

TI Texas Instruments

UDP User Datagram Protocol

UIO UserSpace IO Subsystem des Linux-Kernels

USB Universal Serial Bus

VDMA Video Direct Memory Access

VHDL Very High Speed Integrated Circuit Hardware Description Language, Hardwarebeschreibungssprache

W-LAN Wireless-LAN

XPS Xilinx Platform Studio

XSVI Xilinx Streaming Video Interface

YUV YUV-Farbmodell mit Luminanz Y und Chrominanz UV

# A Integration in neue Mechanik und Elektrik

Den Aufbau des Fahrzeugs zeigt Abb. A.1.

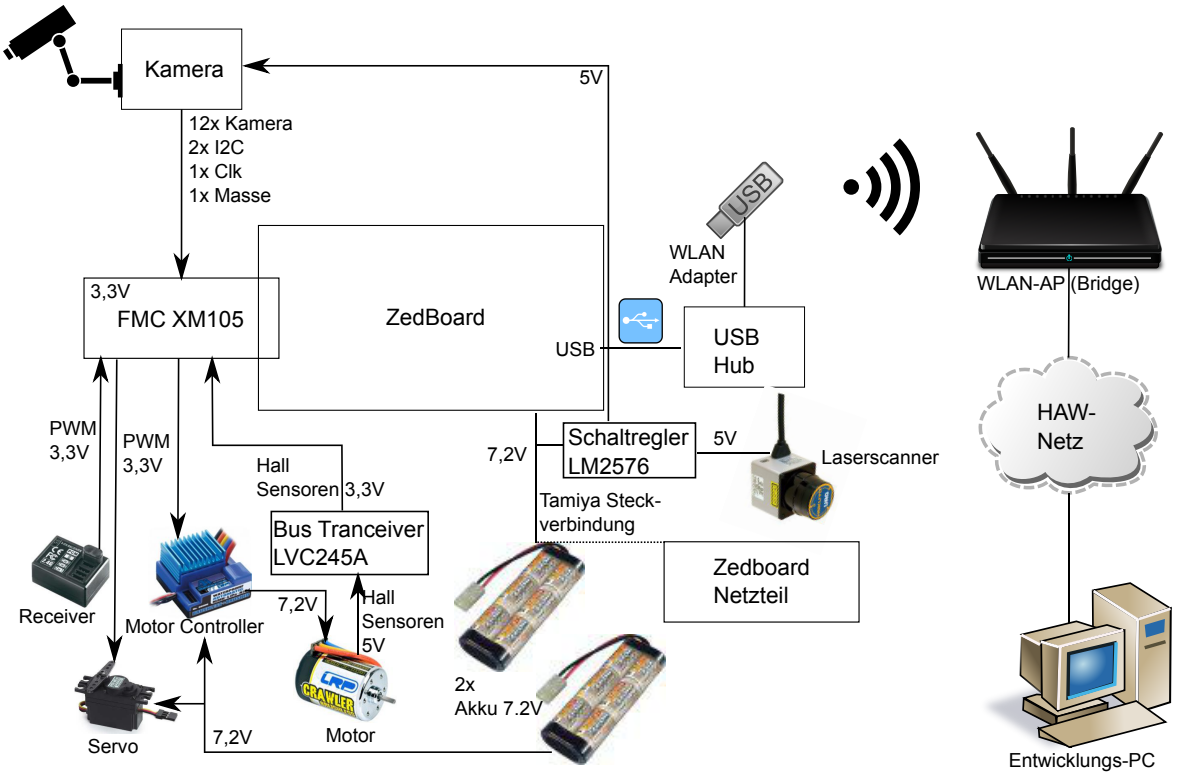



Abbildung A.1: Aufbau des Fahrzeugs.

Die Kamera und die Aktoren sind 3.3V Komponenten, die über das FMC XM105 Adapterboard mit dem Zedboard verbunden werden. Für die Bereitstellung der 3.3V auf dem FMC Board wurden die 3.3V-Pins von Jumper J18 des Zedboards gelötet und verbunden. Aufgrund eines Design-Fehlers funktioniert bei angestecktem FMC-Board die JTAG-Kette nicht mehr, als Abhilfe werden TDI und TDO miteinander verbunden. Die Pinbelegung auf dem FMC XM105 zeigt Tabelle A.1.

| FMC Pin | Zynq Name | Zynq Pin | Beschreibung |
|---|---|---|---|
| J1 1 | FMC_LA00_CC_P | M19 | Kamera Bit 0 |
| J1 2 | FMC_LA10_P | R19 | Kamera Bit 1 |
| J1 3 | FMC_LA00_CC_N | M20 | Kamera Bit 2 |
| J1 4 | FMC_LA10_N | T19 | Kamera Bit 3 |
| J1 5 | FMC_LA01_CC_P | N19 | Kamera Bit 4 |
| J1 6 | FMC_LA11_P | N17 | Kamera Bit 5 |
| J1 7 | FMC_LA01_CC_N | N20 | Kamera Bit 6 |
| J1 8 | FMC_LA11_N | N18 | Kamera Bit 7 |
| J1 9 | FMC_LA02_P | P17 | Kamera Bit 8 |
| J1 10 | FMC_LA12_P | P20 | Kamera Bit 9 |
| J1 11 | FMC_LA02_N | P18 | Kamera Frame valid |
| J1 12 | FMC_LA12_N | P21 | Kamera Line valid |
| J1 13 | FMC_LA03_P | N22 | |
| J1 14 | FMC_LA13_P | L17 | |
| J1 15 | FMC_LA03_N | P22 | I2C SCL |
| J1 16 | FMC_LA13_N | M17 | I2C SDA |
| J1 17 | FMC_LA04_P | M21 | PWM Motor in |
| J1 18 | FMC_LA14_P | K19 | PWM Servo in |
| J1 19 | FMC_LA04_N | M22 | PWM Switch |
| J1 20 | FMC_LA14_N | K20 | |
| J1 21 | FMC_LA05_P | J18 | PWM Motor out |
| J1 22 | FMC_LA15_P | J16 | PWM Servo out |
| J1 23 | FMC_LA05_N | K18 | |
| J1 24 | FMC_LA15_N | J17 | |
| J1 25 | FMC_LA06_P | L21 | SN74LVC245A OE |
| J1 26 | FMC_LA16_P | J20 | SN74LVC245A DIR |
| J1 27 | FMC_LA06_N | L22 | Hall A |
| J1 28 | FMC_LA16_N | K21 | |
| J1 29 | FMC_LA07_P | T16 | Hall B |
| J1 30 | FMC_LA17_CC_P | B19 | Hall C |
| J1 31 | FMC_LA07_N | T17 | |
| J1 32 | FMC_LA17_CC_N | B20 | |
| J1 33 | FMC_LA08_P | J21 | |
| J1 34 | FMC_LA18_CC_P | D20 | Kamera Clk |
| J1 35 | FMC_LA08_N | J22 | |
| J1 36 | FMC_LA18_CC_N | C20 | |
| J1 37 | FMC_LA09_P | R20 | |
| J1 38 | FMC_LA19_P | G15 | |
| J1 39 | FMC_LA09_N | R21 | |
| J1 40 | FMC_LA19_N | G16 | |

Tabelle A.1: FMC Zynq Connections

Um hochfrequente Störungen durch die 27MHz Kamera-Taktleitung auf dem I2C-Bus zu unterdrücken, wurde ein Ferritring auf die I2C-Leitung angebracht. Damit kam es bei Tests mit 100.000 Übertragungen bei einer Übertragungsrate von 100KHz zu keinem Fehler.

Die Umsetzung der 5V der Hall-Sensoren des Motors auf die 3.3V erfolgt mit einem LVC245A als Pegelwandler, die 5V Stromversorgung für Laserscanner und Kamera werden durch einen LM2576 Schaltregler (max 3A) bereitgestellt. Die Stromversorgung erfolgt mit Zwei 7.2V Akkus: Einer für die Aktoren, einer für die Elektronik.

Das Zedboard erzeugt die internen Spannungen aus einer 5V Spannung, die durch einen MAX8686 bereitgestellt werden, der MAX8686 liefert max. 25A. Damit wird das Zedboard auch bei einer Entladeschlussspannung von 6V des Akkus und einer maximalen Last von 24W (Tabelle A.2) innerhalb der Grenzen betrieben.

| **Spannung (V)** | **Strom (mA)** | **Leistung (mW)** |
|---|---:|---:|
| **1.0** | 1872 | 1872 |
| **1.5** | 1035 | 1553 |
| **1.8** | 751 | 1352 |
| **Vadj (3.3V)** | 2000 | 6600 |
| **3.3** | 3814 | 12586 |
| **Summe (5V)** | 4800 | 24000 |

Tabelle A.2: Maximale elektrische Leistung des Zedboards: 24W bei 5V.

# B Handhabungsanleitungen für Funktionskomponenten und SW-Module

Zum Start des Fahrzeugs wird erst die Kamera mit Strom versorgt, dann das Zedboard. Nach dem Bootvorgang wird die Software mit einem Seriellen- oder SSH-Terminal (Benutzer: „root“, Passwort: „zed“) gestartet: Die SAV-Software ist im Programm *framegrabber* enthalten, den Aufruf zeigt Listing B.1. Die aus Vorarbeiten auf den Zynq-7000 portierte Software für das Kamera Interface [47], den Laserscanner [27] und Steuerung des *SAV Controllers* [28] werden als Bibliotheken zum Programm geladen.

Listing B.1: Optionen des SAV-Programms.

```
Usage: ./framegrabber [OPTION]...
-d, --dump-regs                  Dump Camera Registers to stdout
-l, --listen                     Start SAV network listener
-p, --path-tracking              Run Path tracking
-r, --reset                      Reset and init SAV
-R, --reset2                     Reset and init SAV+VDMA
-s, --save-image=filename        Save image (.jpg) or Video (.avi) from Camera
-t, --test-i2c                   Test I2C Interface
-u, --upload-file=filename       Use video
-v, --display-video=hostname|vga Stream Livestream/Show on VGA
-w, --write-regs                 Write Camera Registers
```

Zum Start werden üblicherweise die Kamera-Register einmalig über I2C beschrieben und ein Reset des *SAV Controllers* durchgeführt (Listing B.2).

Listing B.2: Aufruf der SAV-Software zur Inbetriebnahme des Fahrzeugs.

```
# Write Camera Registers using I2C once
> ./framegrabber -w
# Reset and configure SAV Controller
> ./framegrabber -r
# Start Network, Pure Pursuit and output to VGA
> ./framegrabber -l -p -v vga
```

Da zur Initialisierung die Erkennung einer Fahrspur erforderlich ist, muss das Fahrzeug während des Resets auf der Straße stehen. Mit dem Schalter *-p* wird der Netzwerk-Thread

gestartet, mit dem das Fahrzeug über W-LAN durch das *Java Remote Tools* konfiguriert (Kapitel 5.3.4) und auch die Geschwindigkeit (z.B. ein PWM-Wert von 3500 ≙ Impulslänge 65µs von Mittelstellung) eingestellt wird. Die Visualisierung der Fahrspurerkennung und Fahrspurführung erfolgt mit dem Schalter *-v*, dem die Art der Ausgabe mitgegeben wird: Die Art ist „vga“ für Ausgabe über den VGA-Port oder die IP des Entwicklung-PCs, an den der Stream über UDP auf Port 5000 gesendet wird. Der Stream kann mit einem Videoplayer wie vlc [54] mit der Adresse *udp://@ip:5000* abgespielt werden.

## B.1 Registerbelegung

Hier sind die Softwareregister dokumentiert.
Die Beschreibung der Optionen in Register B.1 zeigt Tabelle B.2.

Register B.1: Control (15)

| Enable | V Disable | 29 … 24 | VGA from vdma | Adaptive Binarize | Video Feedback | Hor. Erosion 4x | Hor. Erosion 2x | Hor. Erosion | No PD | 14 … 13 | Pixel Pipeline | Secure Mode | Show Info | Show D | Show Vehicle | Show Grid | Show HT Result | Show LAP | Show ROI | Dynamic ROI | Projective Transf. | Erosion 2x | Erosion Filter | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 … 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 … 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | 1 | 0x3f | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | Reset value |

| Wert | Beschreibung |
|---|---|
| **Erosion Filter** | Schaltet einen Erosionsfilter hinzu |
| **Erosion 2x** | Schaltet einen Erosionsfilter hinzu |
| **Projective Transf.** | Schaltet die Projektive Transformation ein (Kapitel 4.1.2) |
| **Dynamic ROI** | Schaltet die dynamische ROI ein |
| **Show ROI** | Zeigt die ROI in der Visualisierung (Kapitel 5.3.5) |
| **Show LAP** | Zeigt den LAP in der Visualisierung (Kapitel 5.3.5) |
| **Show HT Result** | Zeigt die Hough Transformation in der Visualisierung (Kapitel 5.3.5) |
| **Show Grid** | Zeigt das Koordinatensystem in der Visualisierung (Kapitel 5.3.5) |
| **Show Vehicle** | Zeigt die Fahrzeugbreite in der Visualisierung (Kapitel 5.3.5) |
| **Show D** | Zeigt den Sollabstand zur Fahrspur in der Visualisierung (Kapitel 5.3.5) |
| **Show Info** | Zeigt Text-Informationen in der Visualisierung (Kapitel 5.3.5) |
| **Secure Mode** | Schaltet den *RECEIVER_SWITCH* ein. Abschalten ermöglicht Betrieb ohne Fernbedienung |
| **Pixel Pipeline** | Aktiviert die Pixelpipeline. Abschalten zeigt das ursprüngliche Kamerabild |
| **No PD** | Deaktiviert den PD Regler der Spurführung |
| **Hor. Erosion** | Schaltet einen horizontalen Erosionsfilter zu |
| **Hor. Erosion 2x** | Schaltet einen horizontalen Erosionsfilter zu |
| **Hor. Erosion 4x** | Schaltet einen horizontalen Erosionsfilter zu |
| **Video Feedback** | Schaltet den VDMA vor die Bildverarbeitungspipeline (Kapitel 5.3.5) |
| **Adaptive Binarize** | Aktiviert die adaptive Binarisierung |
| **VGA from vdma** | Schalten den VGA-Eingang an den VDMA (Kapitel 5.3.5) |
| **V Disable** | Deaktiviert den Geschwindigkeitsregler (Kapitel 4.4) |
| **Enable** | Deaktiviert die Bildverarbeitungspipeline während der Register-Konfiguration. |

Tabelle B.2: Beschreibung der Optionen in Register B.2

| Register | Bits 31-16 | Bits 15-0 |
|---|---|---|
| 0 | | |
| 1 | roi_x_pos | roi_y_pos |
| 2 | roi_width | roi_height |
| 3 | v_soll | |
| 4 | regler_param1 | |
| 5 | regler_param2 | |
| 6 | | threshold_in |
| 7 | | sw_pf_alpha |
| 8 | pwm_override (Bit 31) | pwm_manual |
| 9 | | |
| 10 | | |
| 11 | | |
| 12 | | |
| 13 | | |
| 14 | | |
| 15 | control | |
| 16 | pt_param_b11 | |
| 17 | pt_param_b12 | |
| 18 | pt_param_b13 | |
| 19 | pt_param_b21 | |
| 20 | pt_param_b22 | |
| 21 | pt_param_b23 | |
| 22 | pt_param_b31 | |
| 23 | pt_param_b32 | |
| 24 | | |
| 25 | | |
| 26 | | |
| 27 | | |
| 28 | | ht_valid_threshold |
| 29 | | roi_limit_factor |
| 30 | | |
| 31 | | max_pwm |

Tabelle B.1: Historisch gewachsene Registerbelegung des *AXI Configuration Connector*. Für das Kontrollregister (15) siehe Register B.1, für Beschreibung der Parameter Tabelle 4.5 in Kapitel 4.3.

| Register | Bit 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 17 | 0 | | | | | | v | | | | | | | | | | 0 | | | | | | | | | | | | | | | |
| 18 | 0 | | | | roi_x | | | | | | | | | | | | 0 | | | | | | | | | | phi_id | | | | | |
| 19 | 0 | | | | | | phi | | | | | | | | | | 0 | | | r | | | | | | | | | | | | |
| 20 | 0 | | | | | | ht_max_val | | | | | | | | | | 0 | | count | | | | | | | | | | | | | |
| 21 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ht_invalid | roi_state | | |
| 22 | ht_result_count | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 25 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 26 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 27 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 28 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 29 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 30 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 22 | 0 | | | | | | | | | | | | | | | | version | | | | | | | | | | | | | | | |

Tabelle B.3: Historisch gewachsene Registerbelegung des *AXI Status Connector*, für Beschreibung der Werte siehe Tabelle 4.6 in Kapitel 4.3.

# C Quellcode

Listing C.1: Kernel-Modul zur Konfiguration des UIO-Subsystem.

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/platform_device.h>
#include <linux/uio_driver.h>
#include <linux/irq.h>

MODULE_LICENSE("GPL");

#define SAV_IRQ 91

static struct resource uio_resource[] = {
        { // VDMA Shared Memory: Last 12MB of RAM
                .start = 0x1f400000,
                .end   = 0x1fffffff,
                .name = "sav_vdma_mem",
                .flags = IORESOURCE_MEM
        },
        { // VDMA Register
                .start = 0x43000000,
                .end   = 0x4300ffff,
                .name = "sav_vdma_reg",
                .flags = IORESOURCE_MEM
        },
        { // AXI Configuration Connector Register
                .start = 0x62200000,
                .end   = 0x6220ffff,
                .name = "sav_axi_reader",
                .flags = IORESOURCE_MEM
        },
        { // AXI Status Connector Register
                .start = 0x7B200000,
                .end   = 0x7B20ffff,
                .name = "sav_axi_cfg",
                .flags = IORESOURCE_MEM
        },
};

static irqreturn_t irq_handler(int irq, struct uio_info *dev_info) {
        switch(irq) {
                case SAV_IRQ:
                        return IRQ_HANDLED;
        }

        return IRQ_NONE;
}
```

```
46
47 static struct uio_info myfpga_uio_info = {
48         .name = "uio_fpga",
49         .version = "0.1",
50         .irq = SAV_IRQ,
51         .handler = irq_handler,
52 };
53
54 static struct platform_device uio_device = {
55         .name       = "uio_pdrv",
56         .id     = -1,
57         .resource    = uio_resource,
58         .num_resources    = ARRAY_SIZE(uio_resource),
59         .dev.platform_data = &myfpga_uio_info,
60 };
61
62 static int __init mod_init(void)
63 {
64         int ret;
65
66         ret = platform_device_register(&uio_device);
67         irq_set_irq_type(SAV_IRQ, IRQ_TYPE_EDGE_RISING);
68
69         printk(KERN_INFO "UIO Module loaded %d\n", ret);
70
71         return 0;
72 }
73
74 static void __exit mod_exit(void)
75 {
76         platform_device_unregister(&uio_device);
77 }
78
79 module_init(mod_init);
80 module_exit(mod_exit);
```

Listing C.2: Schreiben von GPIO-Pin JA1 mit Linux [38].

```
// JA1 is Bit 28 in EMIO Vector (system.ucf)
// First EMIO GPIO Pin has offset 54
// 54+28=82
void set_debug_pin(char val) {
        int file;
        char buf[2] = {val+0x30, '\n'};

        file = open("/sys/class/gpio/gpio82/value", O_WRONLY);
        if (file < 0) {
                perror("No Debug pin");
                return;
        }

        if (write(file, buf, 1) != 1) {
                perror("Debug pin write error");
        }
        close(file);
}
```

# D PlanAhead und EDK Design Flow Hinweise

Dieses Kapitel dokumentiert Besonderheiten im Umgang mit den Xilinx-Tools.
Das XPS Zedboard-Template ist unter `http://www.zedboard.org/sites/default/files/documentations/zedboard_RevC_v2_XML.zip` zu finden.

### Planahead: Netzliste wird nach Änderungen in XPS nicht neu erzeugt

Es kann vorkommen, daß Planahead keine erneute Synthese durchführt, in diesem Fall sollten in XPS alle generierten Dateien gelöscht werden: $Projekt \rightarrow Clean\ all\ generated\ Files$

### Unbekannte Fehler bei der Synthese

Zu lange Namen können Ursache von einem „Unknown Error“ sein. Die Namen der Projekte, Module und IP-Cores sollten 8 Zeichen nicht überschreiten.

### Software: DMA-Controller kopiert keine Daten

Bei Problemen mit den DMA-Controller sollte der Datencache mit *Xil_DCacheDisable()* ausgeschaltet werden um sicherzustellen, daß der Prozessor und DMA-Controller dieselben Daten lesen. Ist ein inkonsistenter Datencache die Ursache kann dieser mit *Xil_DCacheFlushRange(SourceAddr, Length)* synchronisiert werden. Die Funktionen zum Kontrollieren des Caches befinden sich in der Header-Datei *xil_cache.h* [58].

# E DVD Inhaltsverzeichnis

- **Projekte**
  - **Pathfinder**: *Xilinx System Generator* Projekt der Fahrspurerkennung
  - **ZYNQ_SAV_CTRL**: XPS 14.3 Projekt des SAV
  - **framegrabber**: Linux Programm
  - **Java_Remote_Tool**: Eclipse Projekt des Remote Tools
- **Linux**:
  - **Kernel**: Linux-Kernel Konfigurationsdatei
  - **boot.bin**: Dateien zur Erzeugung der Boot.bin
  - **boot**: Device-Tree Datei
  - **uio_fpga**: UIO Kernel Modul
- **ISE_Projekte**: ISE-Projekte und Simulation für IP-Cores
  - **ise_vregler**: Simulation des Geschwindigkeitsreglers
  - **ise_xsvi2axi**: Simulation des *xsvi2axi* IP-Cores
  - **ise_axi2xsvi**: Simulation des *axis2video* IP-Cores
  - **ClockCrossFIFO**: Xilinx Core Generator Projekt für die Clock Domain Crossing FIFOs
- **Images**: Grafiken
- **docs**: Dokumente und Datenblätter
- **Testfahrt**: Video und Logdatei

# Versicherung über Selbstständigkeit

*Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.*

Hamburg, 20. Juni 2013 Erik Andresen