Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Modellierung einer Video-basierten Fahrspurerkennung mit dem Xilinx System Generator für eine SoC-Plattform

Dennis Mellert

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Modellierung einer Video-basierten Fahrspurerkennung mit dem Xilinx System Generator für eine SoC-Plattform

Dennis Mellert

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

Betreuender Prüfer : Prof. Dr.-Ing. Bernd Schwarz
Zweitgutachter : Prof. Dr.-Ing. Andreas Meisel

Abgegeben am 20. April 2010

Dennis Mellert

Thema der Bachelorarbeit

Modellierung einer Video-basierten Fahrspurerkennung mit dem Xilinx System Generator für eine SoC-Plattform

Stichworte

Hough-Transformation, Fahrspurerkennung, projektive Transformation, Projektion, Xilinx System Generator, Multiratensystem, Pipelining, Bildverarbeitung, kinematisches Fahrzeugmodell, VHDL-Codegenerator, System on Chip, FPGA, VHDL

Kurzzusammenfassung

Diese Arbeit behandelt die Entwicklung einer Video-basierten Fahrspurerkennung für eine System on Chip Plattform. Die Modellierung des Pipelinedatenpfades mit übertakteten Pipelinestufen erfolgt mit dem Xilinx System Generator, der eine Generierung von synthesefähigen RTL-Modellen unterstützt. Die Fahrspurapproximation mit der Hough-Transformation wird im Pixelstrom ohne eine Bildzwischenspeicherung durchgeführt. Eine Korrektur perspektivisch verzeichneter Bildpunkte wird über eine projektive Transformation realisiert, in der ebenfalls übertaktete Pipelinestufen eingesetzt werden. Eine Verifikation erfolgt durch Simulationen der System Generator- und VHDL Modelle.

Dennis Mellert

Title of the paper

Modelling of a video-based lane-detection with the Xilinx System Generator for a SoC platform

Keywords

Hough-transform, lane-detection, projective transform, projection, Xilinx System Generator, multi-rate system, pipelining, image processing, kinematic vehicle model, VHDL code generator, System on Chip, FPGA, VHDL

Abstract

This thesis deals with the development of a video-based lane-detection for a System-on-Chip platform. The modeling of the pipeline datapath with overclocked pipeline stages is done with the Xilinx System Generator, which supports a generation of synthesizable RTL-models. The lane-approximation with the Hough-transform is executed in pixel-stream without intermediate image-storage. A correction of perspectively distorted pixels is accomplished by a projective transformation which uses overclocked pipeline stages as well. A verification is done by a simulation of System Generator- and VHDL models.

Inhaltsverzeichnis

1 Einleitung

Kameraunterstützte Fahrerassistenzsysteme basieren auf der Auswertung von Kamerabildern und der daraus resultierenden Extraktion von Informationen. Die Relevanz dieser Systeme ist in den letzten Jahren gestiegen, was durch die folgenden Anwendungsbeispiele verdeutlicht wird:

- **Verkehrszeichenerkennung**
  Unaufmerksamkeiten des Fahrers in unübersichtlichen Situationen werden durch eine Verkehrszeichenerkennung entschärft, indem dem Fahrer Informationen auch nach dem Passieren des Schildes mitgeteilt werden [9].
- **Rückfahrkamera**
  Einparksituationen lassen sich durch einen Kameraeinsatz besser überblicken, wodurch die Gefahr von Auffahrunfällen verringert wird [25].
- **Night-Vision**
  Durch den Einsatz von Kameras werden Fahrsituationen bei Nacht analysiert, wodurch Fußgänger und Tiere erkannt werden. Daraus resultierende Gefahrensituationen werden auf diese Weise verhindert [24].
- **Lane Assist**
  Nähert sich das Fahrzeug unbeabsichtigt der Fahrspurbegrenzung, wird der Fahrer durch eine Vibration des Lenkrades aufgefordert, die Fahrzeugposition zu korrigieren. Dieses Verfahren basiert auf einer Fahrspurerkennung [16][7].

Gerade bei sicherheitsrelevanten Elementen wie einer Fahrspurerkennung sind die Berechnungen und Auswertungen in einer definierten Zeit durchzuführen. Erfolgen die Berechnungen nicht innerhalb dieses Zeitraumes, ist eine korrekte Reaktion des Systems nicht mehr sichergestellt.

Das Forschungsprojekt FAUST (Fahrerassistenz- und Autonome Systeme) der Hochschule für Angewandte Wissenschaften Hamburg bietet eine Umgebung für die Entwicklung von autonomen Fahrzeugen für die Hardware-/Software-Echtzeitsysteme entworfen und implementiert werden. Anregungen liefert die Automobilindustrie durch ihre Forschungs- und Entwicklungsarbeiten im Bereich der Fahrerassistenzsysteme [14]. Solche Systeme erfordern eine hohe Rechengeschwindigkeit der Komponenten für die digitale Signalverarbeitung. FPGA-basierte System-on-Chip (SoC) Plattformen erfüllen diese gegensätzlichen Anforderungen für eine Prototypenerprobung mit parallelen DSP-Funktionselementen und integrierten Prozessoren bei niedrigen Taktfrequenzen.

In dieser Arbeit wird im Kontext der SoC-Technologieerprobung eine Video-basierte Fahrspurerkennung für den Einsatz in autonomen Modellfahrzeugen mit dem System Generator modelliert. Diese kann in die bestehende SoC-Plattform (vgl. Abb. 1) integriert werden und liefert einen Beitrag zu einem Einparkassistenten. Eine Systemmodellierung mit dem System Generator der Firma Xilinx ist auf eine plattformspezifische Implementierung von parallelen Algorithmen in einer Systemmodellierumgebung und auf die Generierung von synthesefähigen RTL-Modellen ausgelegt. Dieses erleichtert den Systementwurf und bietet Vorteile in der Modellverifikation gegenüber einer RTL-Modellierung im VHDL-Stil, was zu einer Verkürzung der Entwicklungszeit führt. Über eine Modellparametrierung aus der Matlab Entwicklungsumgebung wird eine Genauigkeitsuntersuchung unter Aspekten des Ressourcenbedarfs durchführbar.

Schwerpunkte dieser Arbeit konzentrieren sich auf Entwicklungsbeiträge, die für eine Verwendung in einem Einparkassistenten und eine für eine Bahnführung im bestehenden SoC-System ausgelegt sind (vgl. Abb. 1):

- Eine Fahrspurapproximation mit der Hough-Transformation, die eine Pipeline-Struktur mit übertakteten Pipelinestufen einsetzt, um ein Ressource-Sharing und eine direkte Verarbeitung im Pixelstrom ohne Bildzwischenspeicherung durch die Verwendung von Multizyklusoperationen zu erreichen.
- Eine Korrektur perspektivisch verzeichneter Kameraaufnahmen im Pixelstrom mit einer projektiven Transformation, um eine Fahrspurapproximation in der verzeichnungsfreien Fahrzeugebene durchführen zu können.
- Die Implementierung eines kinematischen Einspurfahrzeugmodells, mit dem eine Positionsschätzung realisiert wird, wenn keine verlässlichen Sensordaten zur Verfügung stehen (vgl. Anhang C).

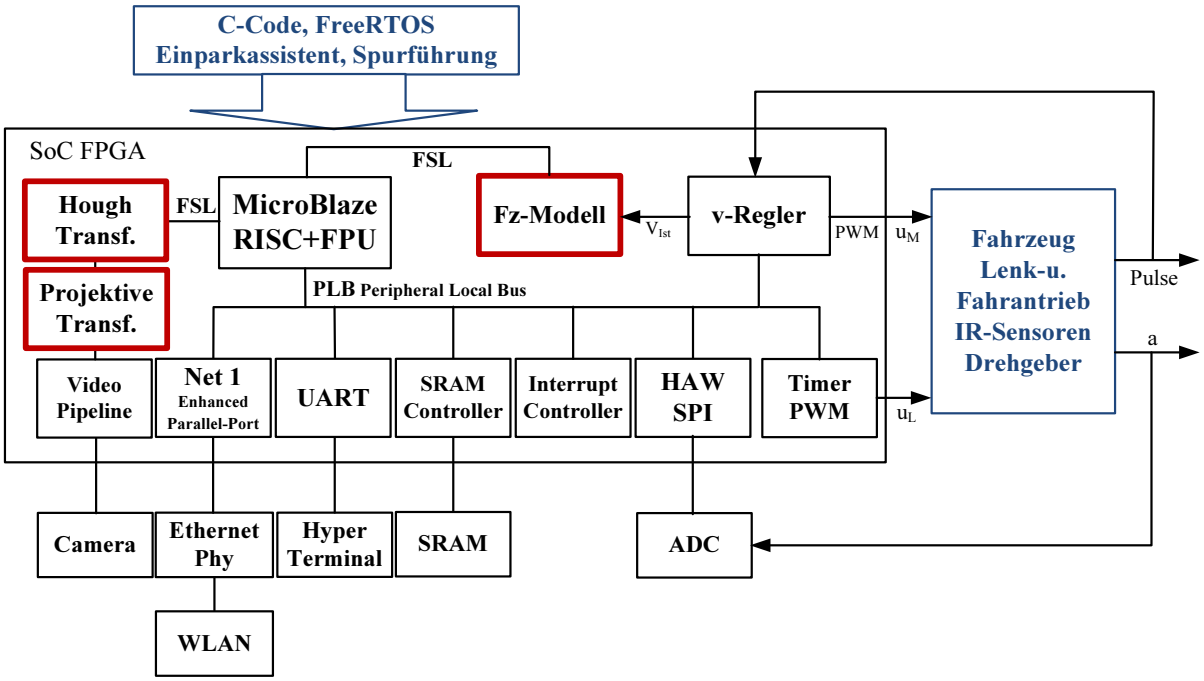

Abb. 1: *SoC-Plattform für die Entwicklung eines Einparkassistenten für ein autonomes Modellfahrzeug*

Die Konzepte und Zusammenhänge dieser Arbeit werden in **Kapitel 2** vorgestellt. Es wird hierbei auf einen strukturellen VHDL-Entwurf eingegangen und die Entwurfsmethodik mit dem System Generator vorgestellt, der zum Systementwurf und zur Simulation sowie zur Generierung von synthesefähigen RTL-Modellen eingesetzt wird.

Das **dritte Kapitel** gibt eine Übersicht zur verwendeten SoC-Plattform. Hierbei wird außerdem auf das Softcore Mikrocontrollersystem MicroBlaze der Firma Xilinx und die zur Fahrspurerkennung verwendete Sony FCB-PV10 Block-Kamera eingegangen.

In **Kapitel 4** wird das bestehende Bildvorverarbeitungssystem vorgestellt, das eine Videopipeline zur Bilddatenaufbereitung und zur Kantenextraktion mit dem Sobel-Filter enthält. Es werden weitere Vorverarbeitungsschritte genannt, die für eine Fahrspurapproximation durchzuführen sind.

Eine Genauigkeitserhöhung der aus dem Kamerabild abgeleiteten Messungen wird durch eine Linsenverzeichnungskorrektur erreicht, die in **Kapitel 5** erläutert wird. Des Weiteren wird

die Modellierung einer projektiven Transformation vorgestellt, die Kameraaufnahmen aus der Bildebene in die Fahrzeugebene abbildet.

Die mathematischen Grundlagen der Hough-Transformation und eine Dimensionierung des als Houghraum bezeichneten Parameterraums werden in **Kapitel 6** vorgestellt. Abschließend wird eine Implementierung der Hough-Transformation in Matlab dargestellt.

In **Kapitel 7** wird eine Modellierung der Hough-Transformation mit dem System Generator vorgestellt. Neben der konzeptionellen Realisierung wird die Analyse der RTL-Modelle mit VHDL-Simulationen vorgestellt.

Eine Ergebnisanalyse der projektiven Transformation mit anschließender Hough-Transformation mit dem System Generator wird in **Kapitel 8** durchgeführt.

Den Abschluss der Arbeit bildet **Kapitel 9** mit einer Zusammenfassung.

2 Konzepte und Zusammenhänge

Die Entwicklungsschwerpunkte dieser Arbeit zeigt folgender Abschnitt auf. Dabei wird auf einen strukturellen VHDL-Entwurf eingegangen, der auf der Partitionierung des digitalen Systems in Daten- und Steuerpfad basiert. Dieses Konzept wird auch durchgängig beim Systementwurf mit dem System Generator genutzt. Durch Entwurfskonzepte wird die Entwicklung systematisiert. Eine Strukturierung bewirkt eine Komplexitätsreduzierung, die zu leichter überprüfbaren Entwicklungsergebnissen führt.

Der System Generator ermöglicht den Entwurf und die Simulation von Systemmodellen, die strukturell und funktional einem Abbild der entworfenen Hardwarearchitektur entsprechen. Eine Generierung von synthesefähigen RTL-Modulen unterstützt den direkten Zugang zur Modulüberprüfung. Im Folgenden wird die Entwurfsmethodik mit dem System Generator vorgestellt und auf wesentliche Funktionsblöcke eingegangen, die dieser zur Verfügung stellt.

2.1 Struktureller VHDL-Entwurf

Ein struktureller VHDL-Entwurf bezeichnet die systematische Gliederung und Entwicklung größerer digitaler Systeme mit VHDL (vgl. Abb. 2). Bei diesem Entwurf werden bereits entworfene VHDL-Entities durch Signale miteinander verbunden, was einen Bottom-Up- bzw. Top-Down-Entwurf unterstützt. Eine Trennung von Daten- und Steuerpfad hat sich beim Entwurf digitaler Systeme bewährt [11]:

- Der **Datenpfad** beinhaltet arithmetische Schaltungskomponenten, die zur Ausführung algorithmischer Operationen eingesetzt werden. Über Dateneingänge gelangen die zu verarbeitenden Daten in das System, wo sie verarbeitet werden und Statussignale erzeugen, die an den Steuerpfad übergeben werden. Die Verarbeitung wird von Steuersignalen kontrolliert.
- Der **Steuerpfad** kann mit einem Zustandsautomaten realisiert werden. Es werden Steuersignale zur Kontrolle der algorithmischen Operationen im Datenpfad generiert. Zu den Eingängen des Steuerpfades gehören die Statussignale des Datenpfades.

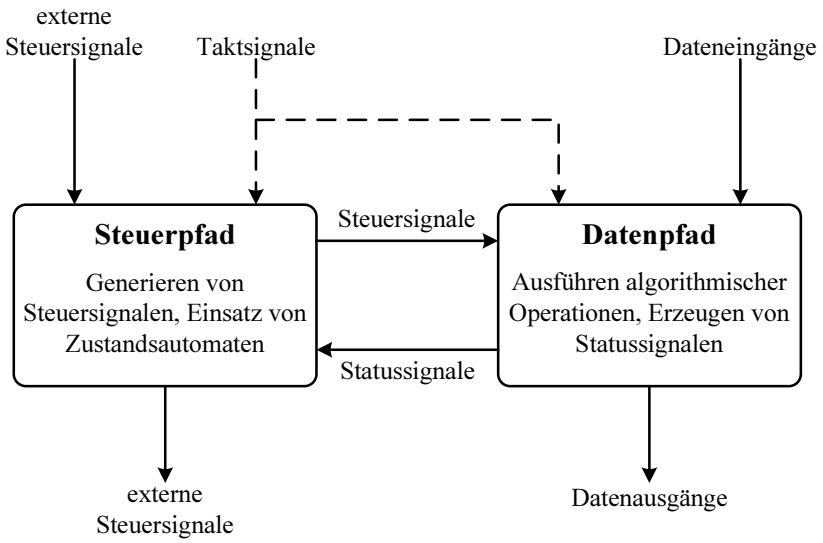

Abb. 2: *Partitionierung eines digitalen Systems in Steuer- und Datenpfad. Die Teilsysteme kommunizieren über Steuer- und Statussignale miteinander. Das Verhalten des Steuerpfades wird über externe Steuersignale beeinflusst.*

Ein struktureller VHDL-Entwurf basiert auf einem Blockschaltbild, in dem alle Datenpfadelemente zur Realisierung der Systemfunktion dargestellt sind. Abgeleitet aus dieser Darstellung ergeben sich die Anforderungen an die Steuersignale. Der Entwurf eines Datenpfades basiert auf der Unterscheidung zwischen einem Pipeliningdatenpfad und einen Multizyklusdatenpfad. Der Datenpfad des Systems zur Berechnung der Hough-Transformation für den Eingangspixelstrom wird mit einem übertakteten Pipeliningdatenpfad realisiert, in dem Multizykluselemente in übertaktete Pipelinestufen integriert sind (vgl. Abb. 3).

Pipeliningdatenpfad

Für den Fall einer hohen Datenrate, die der Systemfrequenz nahe kommt, wird für den Datenpfad die Anwendung einer Pipelinetechnik erforderlich, damit Eingangsdaten ohne Zwischenspeicherung direkt verarbeitet werden [27] (vgl. Abb. 3):

- Komplexe Arithmetikblöcke werden durch den Einsatz von Trennregistern in kürzere Schaltnetzstufen separiert.
- Durch eine taktsynchrone Ausführung aller Abschnitte werden mehrere Eingangsdatensätze parallel in aufeinanderfolgenden Stufen verarbeitet.
- Die Projektion und die Hough-Transformation werden in den Pixelstrom integriert. Jedes aktualisierte Pixel wird in die erste Stufe einer Pipeline eingetaktet, während in tieferen Berechnungsstufen der Pipeline vorangegangene Pixel in Bearbeitung sind.
- In n-Pipeliningstufen werden n Datensätze parallel verarbeitet, was eine Steigerung des Datendurchsatzes bedeutet. Es ergibt sich eine Latenz von n Takten, bis ein Ergebnis am Ausgang anliegt.

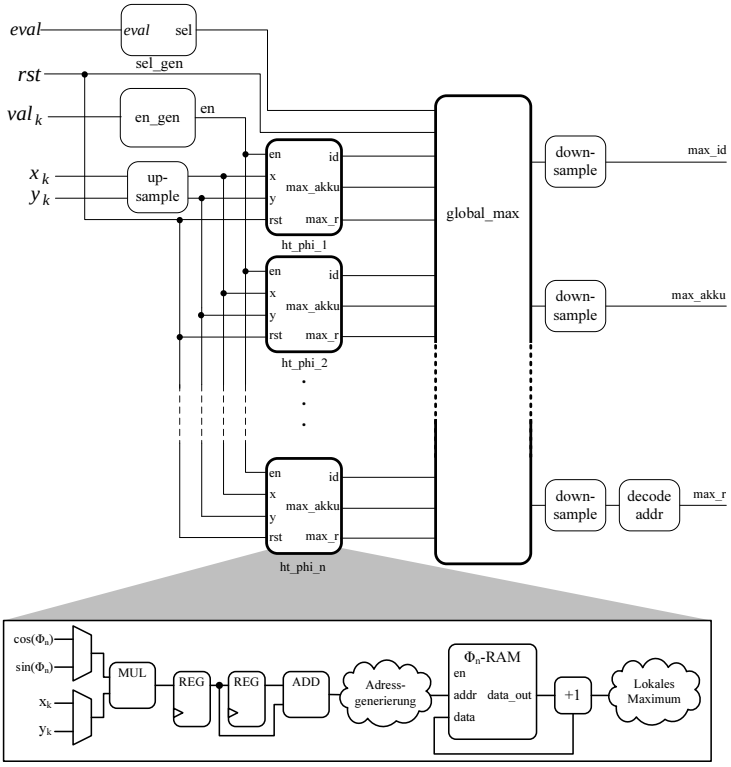

Abb. 3: *Pipeliningdatenpfad der Hough-Transformation mit übertakteten Pipelinestufen mit integrierten Multizykluselementen; komplexe Pipelinestufen werden durch Einsatz von Trennregistern in kürzere Schaltnetzstufen aufgeteilt; mehrere Eingangsdatensätze werden parallel in aufeinanderfolgenden Stufen verarbeitet*

Multizyklusdatenpfad

Ein Multizyklusdatenpfad wird im Falle einer Datenrate eingesetzt, die deutlich geringer als die Systemfrequenz ist [27] (vgl. Abb. 4):

- Arithmetikelemente werden für Berechnungen in aufeinander folgenden Zyklen mit unterschiedlichen Operanden verwendet, so dass diese zum Ressource-Sharing nur einmalig bereitgestellt werden müssen.
- Speicherelemente werden gemeinsam genutzt (Register-Sharing) und nehmen in der Verarbeitungssequenz Ergebnisse unterschiedlicher Berechnungen auf.
- Der Steuerpfad wird durch einen Zustandsautomaten realisiert.
- Die Systemfrequenz muss höher sein als die Datenrate, da alle Berechnungen, die von einer geringen Anzahl Arithmetikelemente durchgeführt werden, bis zur nächsten Aktualisierung des Eingangsdatums beendet sein müssen.

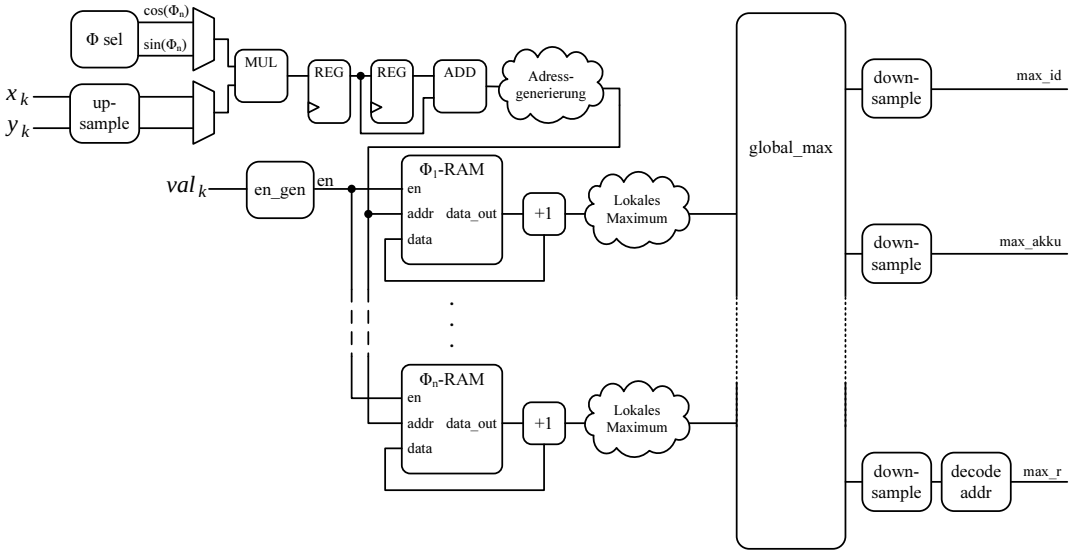

Abb. 4: *Multizyklusdatenpfad mit Ressource-Sharing. Arithmetikressourcen werden für Berechnungen mit unterschiedlichen Operanden verwendet. Durch eine Übertaktung des Systems um den Faktor n, werden alle Berechnungen durchgeführt, bis ein aktualisierter Pixelwert anliegt.*

2.2 Entwurfsmethodik mit dem Xilinx System Generator

Der System Generator ist ein System-Level-Modellierwerkzeug, das plattformspezifisches High-Level FPGA Hardwaredesign in einer Systemmodellierumgebung unterstützt, indem es Erweiterungen in Form eines Blocksets für Matlab/Simulink zur Verfügung stellt und eine Generierung von synthesefähigen RTL-Modellen unterstützt [35]. Die Blöcke stellen Abstraktionen für mathematische-, logische-, Speicher- und DSP-Funktionen dar, die zum Entwurf von Fixed-Point Modellen zeitdiskreter digitaler Systeme eingesetzt werden. System Generator Blöcke sind *Bit-* und *Cycle-true*:

- **Bit-true**
  Blöcke erzeugen in der Simulation Werte, die den in Hardware produzierten bitgenau entsprechen.
- **Cycle-true**
  Korrespondierende Werte werden zu korrespondierenden Zeitpunkten produziert.

2.2.1 Automatische Code Generierung

Der System Generator kompiliert Systementwürfe in synthesefähige RTL-Modelle und generiert neben HDL-Files auch Hilfsdateien, die zur Verifikation des Entwurfs verwendet werden und im Folgenden beschrieben sind.

System Generator Block

Eine Kontrolle von System- und Simulationsparametern eines Modells, das System Generator Blöcke enthält, erfolgt über den System Generator Block. Die Konfiguration des Kompilationsergebnisses als *HDL Netlist* umfasst [34]:

- HDL-Files, die als VHDL- oder Verilog-Files verfügbar sind
- EDIF-Files, die einen Austausch von Netzlisten zwischen elektronischen Systemen ermöglichen [18]
- Hilfsdateien, die Unterstützung bei der Integration des Projektes in Werkzeuge wie Xilinx ISE oder ModelSim bieten

Die Generierung einer Testbench führt zur Erzeugung weiterer Dateien:

- HDL-Testbench, die bei einer Simulation die Simulink-Simulationsergebnisse mit denen des kompilierten Modells vergleicht
- Verifikationsdateien, in denen Vektoren abgelegt sind, die während der System Generator- / Simulink-Simulation an den Systemgrenzen auftreten

Die Einstellung *Simulink System Period* legt die Rate fest, in der Simulationen durchgeführt werden. Die Periode ist der größte gemeinsame Teiler aller im System verwendeten Perioden, die explizit eingestellt werden oder sich durch Vererbung oder Hardware-Oversampling ergeben.

Ergebnisse der Kompilierung

Die Generierung der Dateien erfolgt in einem Verzeichnis, das im System Generator Block angegeben wird. Bei einer Auswahl von VHDL als Hardware Description Language, von XST als Synthesewerkzeug und einer selektierten Option zur Generierung einer Testbench, ergeben sich die in Tabelle 1 dargestellten Hauptdateien [35].

Der System Generator erzeugt ein *constraint*-File, das Anweisungen zum Verarbeiten des Modells für Downstream-Werkzeuge beinhaltet und so eine Steigerung der Implementierungsqualität ermöglicht. Das File beinhaltet:

- die zu verwendende Systemperiode
- die Geschwindigkeit unter Berücksichtigung der Systemfrequenz, mit der Teile des Systems laufen müssen
- die Pins, an denen sich die Ports befinden sollen
- die Geschwindigkeit, mit der die Ports arbeiten müssen

Das Dateiformat ist abhängig von der Wahl des Synthesewerkzeugs und wird unter Auswahl des Xilinx Systhese Tools XST im XCF-Format erzeugt.

Dateiname, -typ	Beschreibung
<modell>.vhd	Beinhaltet einen Großteil der VHDL Beschreibungen des Modells.
<modell>_cw.vhd	VHDL-Wrapper für <modell>.vhd; treibt Clock- und Clock Enable Signale.
.edn und .ngc	Der System Generator ruft den Xilinx CORE Generator auf, der Teile des Modells implementiert.
globals	Key/Value-Paare, die das Modell beschreiben.
<modell>_cw.xcf	Beinhaltet Timing- und Port-Location Constraints; werden vom Synthese- und Implementierwerkzeug verwendet.
<modell>_cw.ise	Öffnet ein Xilinx ISE Projekt, dass die erzeugten VHDL- und EDIF-Files enthält.
hdlFiles	Vollständige Liste der vom System Generator erstellten HDL Files.
vcom.do	Skript zum Kompilieren des VHDLs für eine ModelSim Simulation.
.dat	Simulationsergebnisse aus Simulink; sowohl Eingangs- als auch Ausgangsdaten werden zur Verifikation des Verhaltens eingesetzt.
<modell>_tb.vhd	Testbench-Wrapper für das Modell; bei einer ModelSim Simulation des Modells vergleicht die Testbench Simulink-Simulationsergebnisse mit denen aus ModelSim.
vsim.do	Skript zum Starten einer Testbench-Simulation mit ModelSim.
pn_behavioral.do, pn_postmap.do, pn_postpar.do, pn_posttranslate.do	Erlaubt das Ausführen verschiedener ModelSim-Simulationen aus Xilinx ISE.

Tabelle 1: *Liste der Dateien, die bei einer HDL Netlist Kompilierung mit ausgewählter Testbench-Option generiert werden*

2.2.2 Zahlendarstellung im Q-Format für die Fixed-Point Modellierung

Vorzeichenbehaftete Zahlen im Zweierkomplement-Format werden als *fix* bezeichnet, vorzeichenlose Zahlen als *ufix* [34]. Die Zahlendarstellung im Q-Format bietet den Vorteil, dass keine Floatingpoint-Bibliotheken eingebunden werden müssen, um Festkommazahlen verarbeiten zu können. Für das Synthesewerkzeug erfolgt eine Bearbeitung von Binärzahlen im Integerformat. Ein Vektor *fix_1+n+m_m* im Q-Format beinhaltet:

- ein Vorzeichenbit S
- n Integerbits G_n
- m Nachkommabits F_m

Die Darstellung entspricht der Form:

$$SG_{n-1}G_{n-2} \ldots G_0 \bullet \underbrace{F_{m-1}}_{2^{-1}} F_{m-2} \ldots \underbrace{F_0}_{2^{-m}} \tag{1}$$

Die Kennwerte der Zahlendarstellung des Q-Formates in *fix_9_7*-Darstellung sind in Tabelle 2 aufgezeigt:

Vektorbreite	9 Bit
Kleinste negative Zahl	-2
Größte positive Zahl	$2 - 2^{-7}$ = 1,9921875
Quantisierungsstufe	2^{-7} = 0,0078125
Maximaler Quantisierungsfehler	2^{-7-1} = 0,00390635

Tabelle 2: *Kennwerte der fix_9_7 Q-Formatsdarstellung [3]*

Addition mit vorzeichenrichtiger Erweiterung der Summanden

Eine Addition von Zahlen im Q-Format erfordert für den Fall einer unterschiedlichen Anzahl Nachkommastellen eine Gewichtungsanpassung der Summanden, über eine Konkatenierung mit abschließenden Nullen. Für eine überlauffreie Realisierung von Zweierkomplement-Summen wird eine Erweiterung der Bitbreite des Ergebnisvektors eingesetzt. Die Addition zweier Zahlen im *fix_9_7*-Format erfordert einen Ergebnisvektor der Dimension *fix_10_7* (vgl. Gl. 2), dessen zusätzliches Integerbit ein Sicherheitsbit darstellt, das den Übertrag aus der Summe aufnimmt.

$$\underbrace{SG \bullet F_6F_5 \ldots F_0}_{fix\_9\_7} + \underbrace{SG \bullet F_6F_5 \ldots F_0}_{fix\_9\_7} = \underbrace{SGG \bullet F_6F_5 \ldots F_0}_{fix\_10\_7} \tag{2}$$

Binäre Multiplikation von Zahlen im Q-Format

Der Ergebnisvektor einer Multiplikation ist so breit zu wählen, dass die Summe der Vektorbreiten der Faktoren aufgenommen werden kann. Die Multiplikation einer Zahl im *fix_i_j*-Format und einer Zahl im *fix_n_m*-Format erzeugt ein Ergebnis im *fix_i+n_j+m*-Format (vgl. Gl. 4). Eine Q-Format-Multiplikation liefert zwei Vorzeichenbits, von denen das linke das Vorzeichen des Berechnungsergebnisses angibt und das rechte als Integerbit genutzt wird [27].

$$\underbrace{SG \bullet F_6F_5 \ldots F_0}_{fix\_9\_7} * \underbrace{SG \bullet F_6F_5 \ldots F_0}_{fix\_9\_7} = \underbrace{SSGG \bullet F_{13}F_{12} \ldots F_0}_{fix\_18\_14} \tag{3}$$

$$= SGGG \bullet F_{13}F_{12} \ldots F_0$$

Darstellung von zweiwertigen Signalen

Mit dem Datentypen *boolean* werden ein Bit breite Signale dargestellt, die beispielsweise als Steuersignale für *enable*- oder *reset*-Ports verwendet werden. Der *boolean*-Typ ist eine Variante einer 1-Bit *unsigned* Zahl und besitzt immer einen *low*- oder *high*-Pegel. Er ist daher immer in einem gültigen Zustand, während eine 1-Bit *unsigned* Zahl, die nicht als Variable vom Typ *boolean* definiert ist, ungültig werden kann [34].

2.2.3 Xilinx System Generator Blockset

Jeder System Generator Block verfügt über konfigurierbare Parameter, die für eine Vielzahl Blöcke identisch sind. Die Konfiguration kann dynamisch mit Matlab-Code erfolgen oder über direkte Parametrierung, indem Blockoptionen festgelegt werden [34].

- **Precision** Berechnungen mit dem Xilinx System Generator basieren auf Fixed-Point Arithmetik im Q-Format [27]. Die Präzision der Darstellung von Nachkommazahlen wird durch die Anzahl der Bits rechts vom Binärpunkt bestimmt. Das Abschneiden von niederwertigen Bitpositionen führt zu Quantisierungsfehlern. Über eine Auslegung der Vektorbreite für die Q-Formatsdarstellung als *full precision* stellt der System Generator sicher, dass keine Präzision verloren geht. Die Vektoren werden mit der entsprechenden Bitbreite ausgelegt, was zu einem gesteigerten Ressourcenaufwand gegenüber einer mit Quantisierungsfehlern behafteten Realisierung führt.
- **Type, Number of Bits und Binary Point** Fixed-Point Zahlen werden durch die Bitbreite, der Position des Binärpunktes und den arithmetischen Typ definiert und sind für den System Generator somit handhabbar und interpretierbar. Die maximale Breite einer Fixed-Point Zahl beträgt 4096 Bit.
- **Overflow and Quantization** Bei benutzerdefinierter Präzision führen Überlauf und Quantisierung zu Fehlern. Ein Überlauf tritt auf, wenn Werte außerhalb des darstellbaren Bereichs liegen. Ein Quantisierungsfehler liegt vor, wenn die Anzahl der Bits zur Darstellung des Nachkommaanteils unzureichend ist. Sättigungsfunktionen verhindern Störeffekte durch Überläufe und erzeugen sinnvolle Signalintervallbegrenzungen. Quantisierungsfehler werden über ein Runden auf den nächsten darstellbaren Wert oder über ein Abschneiden von Bits rechts vom LSB behandelt.
- **Latency** Über den Parameter *Latency* wird die Anzahl der Sample Perioden bestimmt, die der Ausgang des Blocks verzögert wird, um einen Laufzeitausgleich zu erreichen.
- **Sample Period** Datenströme werden zu festgelegten periodischen Zeitpunkten verarbeitet. Über die System Generator Blöcke *Up Sample* und *Down Sample* wird die Frequenz erhöht oder verringert.

Gateway In - und Gateway Out Blöcke

Systemgrenzen werden durch Gateway In- und Gateway Out Blöcke identifiziert und werden in RTL-Modellen in Top-Level-Input- und Top-Level-Output-Ports übersetzt. Gateway In Blöcke konvertieren Simulink Integer-, Double- und Fixed-Point-Daten in die System Generator Fixed-Point Darstellung unter Verwendung von Überlauf- und Quantisierungsoptionen. Gateway Out Blöcke stellen die Ausgänge des Modells dar und werden von System Generator Fixed-Point in Simulink Double Datentypen konvertiert. Testpunkte, die über Gateway Out Blöcke in ein Modell integriert sind, werden aus einer RTL-Modell-Darstellung entfernt.

Clock Enable Probe

Einen Mechanismus zur Verwendung von abgeleiteten Clock Enable Signalen, die Bereiche freigeben, die mit niedrigen Taktfrequenzen arbeiten, stellt das Clock Enable Probe zur Verfügung. Der *enable*-Puls wird am Ende einer Sample Periode des Eingangssignals, dessen Enablekonzept ausgewertet wird, für die Dauer einer Simulink System Period angelegt. Das Ausgangssignal des CE-Probes entspricht dem in einem RTL-Modell verwendeten Clock Enable des Signals in einem Multiratensystem (vgl. Kap. 2.2.5), das aus der Sample Periode des Eingangssignals abgeleitet wird. Bei Signalen, deren Sample Periode der Simulink System Periode entspricht, ist der Ausgang des CE-Probes dauerhaft auf einem *high*-Pegel.

Divider Generator 2.0

Ein Dividierer für eine Integerdivision, basierend auf dem Radix-2 oder High-Radix Algorithmus, wird mit dem Divider Generator 2.0 erzeugt. Für ein Spartan 3E FPGA steht der Radix-2 Algorithmus zur Verfügung, der ein Bit des Quotienten pro Takt löst und hierbei Additionen und Subtraktionen einsetzt. Durch ein Pipelining wird ein Durchsatz von einer Division pro Takt ermöglicht. Das Ergebnis der Division besteht aus dem Integeranteil des Quotienten, sowie je nach Konfiguration dem Integer-Rest der Division oder dem Nachkommaanteil [32]. Wird kein Durchsatz von einer Division pro Takt benötigt, wird eine Ressourcenersparnis erreicht.

2.2.4 Formatveränderungen

Zur Konvertierung eines System Generator Signalvektors stehen die Blöcke *Reinterpret*, *Slice*, *Concat* und *Convert* zur Verfügung, mit denen sich Formatveränderungsketten aufbauen lassen, die eine Vektorumwandlung unterstützen (vgl. Abb. 5).

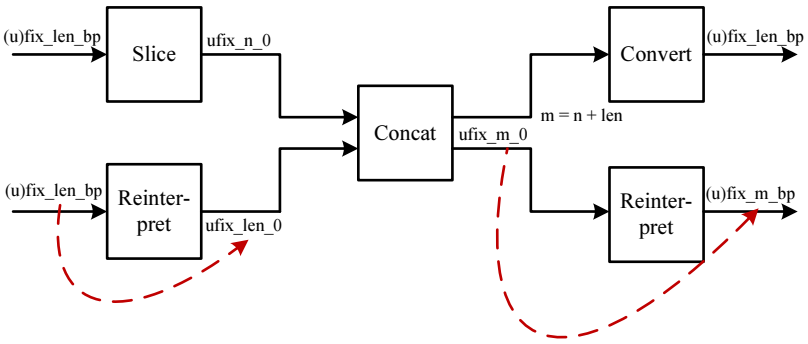

Abb. 5: *Eine Änderung des Formates von System Generator Signalvektoren wird unter Verwendung der Reinterpret-, Slice-, Concat-, und Convert-Blöcke erreicht*

Reinterpret

Eine Änderung des Typs des Eingangswertes ohne das Beibehalten des numerischen Wertes zu berücksichtigen bietet der Reinterpret Block. Die binäre Repräsentation des Ausgangswertes entspricht immer der des Eingangswertes, so dass der Reinterpret Block keine Hardware Ressourcen beansprucht. Eingangsdaten können in eine *signed/ unsigned* Darstellung gewandelt und über eine Binärpunktverschiebung skaliert werden.

Slice

Die Extraktion eines Datenausschnitts erlaubt der Slice Block, der diesen als *unsigned*-Datentyp mit einem Binärpunkt bei Null an den Ausgang des Blocks anlegt. Der Block stellt verschiedene Mechanismen zur Spezifikation des relevanten Datenausschnitts zur Verfügung (vgl. Abb. 6).

Concat

Eine Zusammenfügung von bis zu 1024 *unsigned* Integer Vektoren mit Binärpunkt an Position Null unterstützt der Concat Block. Der Ergebnisvektor darf eine Breite von 4096 Bit nicht überschreiten (vgl. Kap. 2.2.3).

Convert

Eine Überführung des Eingangswertes in einen anderen arithmetischen Datentyp bietet der Convert Block.

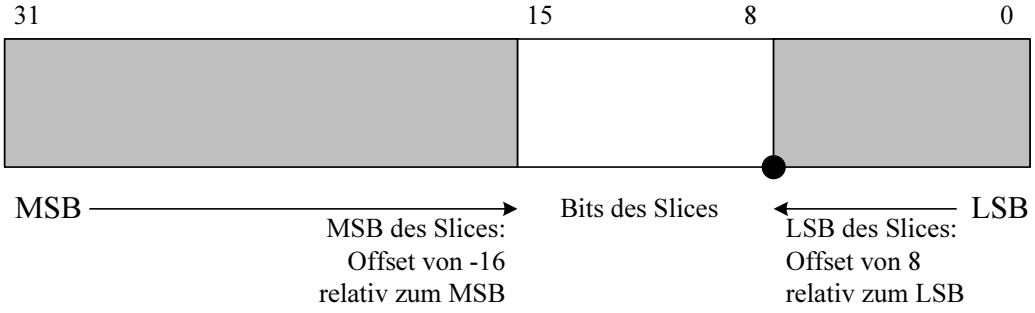

Abb. 6: *Spezifikation eines Slices über Angabe des MSBs des Slices relativ zum MSB des Eingangsvektors und Angabe des LSBs des Slices relativ zum LSB des Eingangsdatums*

2.2.5 Multiratensysteme mit dem System Generator

Für die Berechnung der projektiven Transformation und der Hough-Transformation werden Pipelinestufen innerhalb des Datenpfades mit einer höheren Frequenz als der Pixelfrequenz $f_{13.5}$ = 13,5MHz betrieben, um innerhalb der Pixelperiode $T_{13.5}$ Funktionen, die aus mehreren Rechenschritten bestehen, ausführen zu können.

Ein Übertakten wird im System Generator mit dem *Up Sample* Block erreicht, der die Sample Rate in dem Bereich erhöht, der auf den Block folgt. Bei einem n-fachen Upsampling wird das Eingangsdatum n-mal als Ausgangswert präsentiert (vgl. Abb. 9). Durch eine Erhöhung der Sample Rate um den Faktor n werden Samples n-mal ausgewertet (vgl. Abb. 7 und 8):

- über eine Enablesignalgenerierung unter Einsatz des Clock Enable Probes wird eine Mehrfachauswertung eines Eingangsdatums verhindert
- gültige Werte werden so nur einmalig ausgewertet, können jedoch in übertakteten Pipelinestufen verarbeitet werden

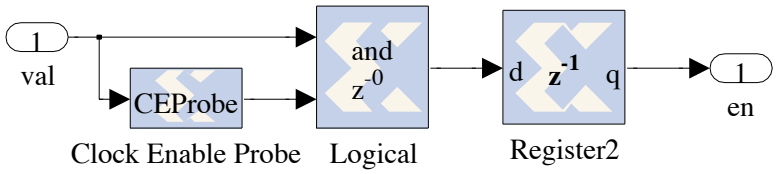

Abb. 7: *Erzeugung eines Clock Enable Signals unter Verwendung des Clock Enable Probes und eines AND Blocks*

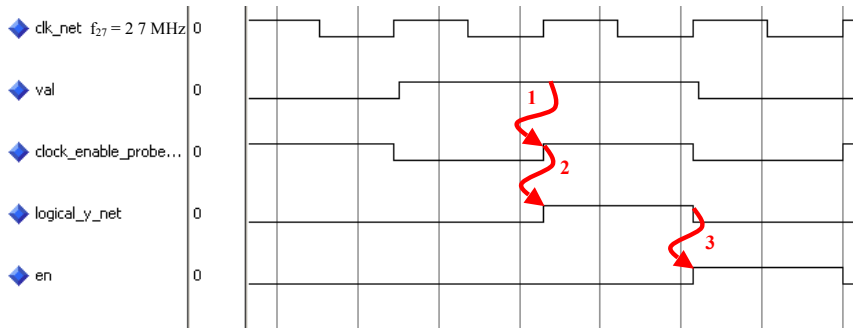

Abb. 8: *Das Clock Enable Probe stellt das Hardware Clock Freigabeschema des Signals val_k mit $f_{13.5}$ = 13,5 MHz dar; durch eine UND-Verknüpfung mit dem Signal val_k wird ein enable-Signal mit einer Periodendauer T_{27} erzeugt*

Die mit dem Up Sample Block erhöhte Sample Rate wird mit dem Down Sample Block reduziert. Der Ausgangswert besteht aus einem Sample pro Frame, das für die Dauer der Sample Periode anliegt (vgl. Abb. 10). In einem System Generator Modell werden alle Sample Perioden als Integervielfache der Systemperiode hergeleitet. Zur Erzeugung unterschiedlicher Sample Perioden wird in dieser Arbeit das Clock Enable Konzept des System Generators verwendet, bei dem alle Perioden die durch Up- und Downsampling erreicht werden, Integervielfache der Systemperiode sind [35].

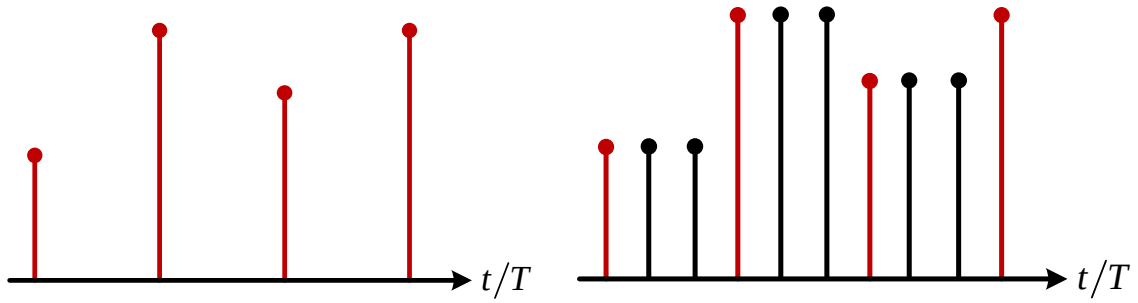

Abb. 9: *Durch n-faches Upsampling werden die Samples n-mal dupliziert und als Ausgangswert präsentiert*

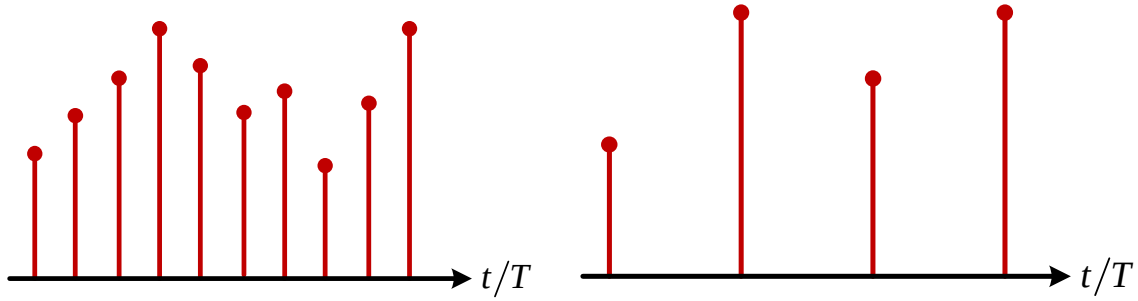

Abb. 10: *Durch Downsampling werden Samples des Ursprungssignals ignoriert*

3 Technologieübersicht zur Bildverarbeitungsplattform

Dieser Abschnitt stellt den für weniger zeitkritische Anwendungen zum Einsatz kommenden MicroBlaze Softcore Prozessor der Firma Xilinx sowie das Nexys2-Entwicklungsboard und die auf dem Fahrzeug angebrachte Sony FCB-PV10 Farbbildkamera vor. Diese liefert Aufnahmen von der Fahrbahn und von Hindernissen nach einer Formaterweiterung mit der Pixelfrequenz $f_{13.5}$ = 13,5 MHz.

3.1 Xilinx MicroBlaze System

Mit System on Chip (SoC) Plattformen können Hardware/Software Systeme auf einem Chip realisiert werden, so dass nicht nur einzelne Hardwaremodule auf einem FPGA implementiert werden, sondern ganze Systeme, die aus Hardwaremodulen, Peripherie und μControllern bestehen. Als μController kommen Hardcore-(Power PC, ARM) oder Softcore-Prozessoren (Xilinx MicroBlaze (vgl. Abb. 11), Altera NiosII) zum Einsatz. Hardcore-Prozessoren sind auf dem Chip integriert und nicht veränderbar, wogegen Softcore-Prozessoren in Form von synthetisierbaren Netzlisten und HDL-Modulen vorliegen und sich der Vorteil der anwendungsspezifischen Konfigurierbarkeit ergibt. Die Firma Xilinx stellt mit dem *Sofware Development Kit (SDK)* ein Werkzeug zur Verwaltung von Softwareprojekten für MicroBlaze Systeme zur Verfügung.

Der MicroBlaze ist ein 32-Bit RISC Softcore-Prozessor mit Harvard Architektur und 3- bzw. 5-stufiger Pipeline, der für die Implementierung auf Xilinx FPGAs optimiert ist [33]. Zur Kopplung des MicroBlaze mit dem internen Speicher wird der *Local Memory Bus (LMB)* verwendet [31]. Weiterhin werden der *Processor Local Bus (PLB)* [37] und der *Fast Simplex Link (FSL)* [36] zur Verbindung der verwendeten Peripherie und des MicroBlazes bereitgestellt.

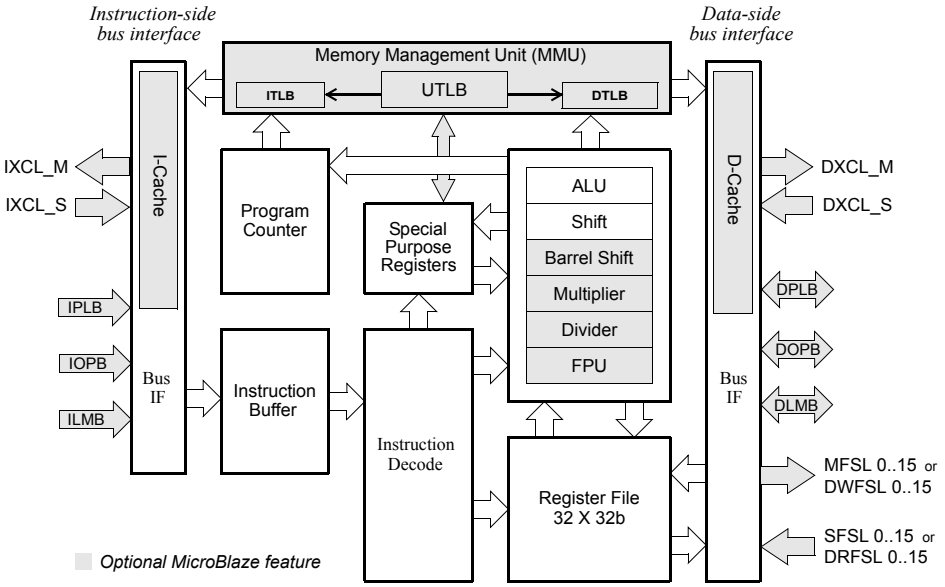

Abb. 11: *Aufbau des MicroBlaze Softcore Prozessors [33]*

Der Vorteil des Einsatzes eines μControllers besteht in der Möglichkeit, über Softwarebibliotheken auf bereits vorliegende sowie eigene IPs zugreifen zu können. So kann beispielsweise auf Hardwareinterrupts mit Interrupt Service Routinen reagiert oder eine Systeminitialisierung durchgeführt werden. Zum Einsatz kommt das MicroBlaze System für die Steuerung der Übertragung von Kameraaufnahmen an die *COM*-Schnittstelle eines PCs zu Debugzwecken [17].

3.2 SoC-Plattform

Das Digilent Nexys2 Entwicklungsboard dient als Zielhardware für das SoC-System und stellt neben einem Spartan 3E FPGA zusätzliche Peripherie wie I/O Schnittstellen und externen Speicher zur Verfügung und ermöglicht so eine Integration in ein Gesamtsystem. Folgende Komponenten stehen auf dem Entwicklungsboard zur Verfügung (vgl. Abb. 12):

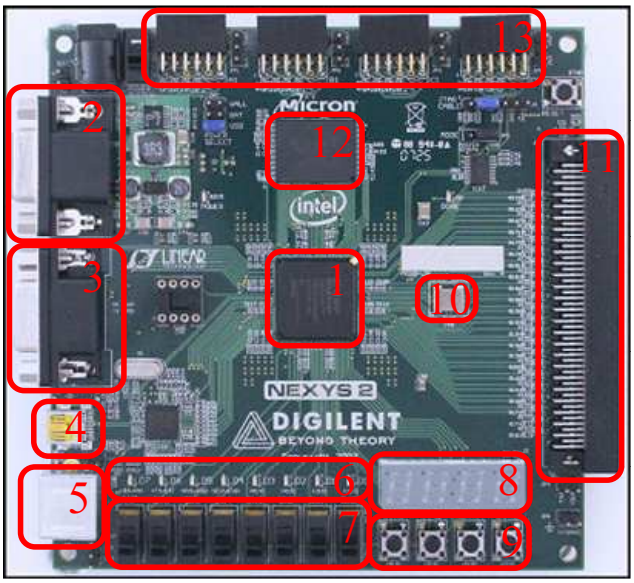

Abb. 12: *Digilent Nexys2 Entwicklungsboard mit Sparten 3E FPGA und Peripherie [6]*

1. Xilinx Spartan-3E XC3S1200E FPGA
2. VGA Schnittstelle
3. RS232 Schnittstelle
4. USB2 Schnittstelle
5. PS2 Schnittstelle
6. 8 LEDs
7. 8 Switches
8. 7-Segment Anzeige
9. 4 Pushbuttons
10. 50 MHz Oszillator
11. FX2 Connector
12. 16 MB SRAM und 16 MB Flash ROM
13. Pmod Schnittstellen

Das Xilinx Spartan-3E XC3S1200E FPGA verfügt über

- 2168 CLBs
- 17344 Flip Flops
- 17344 LUTs
- 250 IOBs
- 28 18x18 Multiplizierer
- 28 18-KBit BRAM
- 8 DCMs

- **Configurable Logic Blocks (CLBs)**
  beinhalten Lookup Tabellen (LUTs), die sowohl Logik als auch Speicherelemente in Form von Flip-Flops oder Latches realisieren. Jedes CLB besteht aus vier Slices, wobei jedes Slice zwei 4-Input LUTs, zwei Register, zwei Multiplexer sowie carry- und arithmetische Logik enthält. Wertetabellen lassen sich zur Realisierung von logischen Funktionen nutzen.
- **Input/Output Blocks (IOBs)**
  kontrollieren den Datenfluss zwischen I/O Pins und der internen Logik des FPGAs. Jeder IOB unterstützt bidirektionalen Datenfluss sowie tristate Operationen.
- **Block RAM (BRAM)**
  bietet Datenspeicherung in Form von 18-Kbit Dual Port Blöcken.
- **Multiplizierer**
  berechnen aus zwei bis zu 18 Bit breiten Faktoren das Produkt.
- **Digital Clock Manager (DCM)**
  dienen zum Verteilen, Verzögern, Multiplizieren und Teilen von Taktsignalen.

3.3 Sony FCB-PV10

Die Sony FCB-PV10 umfasst einen $1/4$-Type Interline-Transfer-CCD Bildsensor, der Bilder in einer VGA-Auflösung mit 640x480 Pixeln und einer Bildrate von 29.97 Bildern pro Sekunde über ein FFC direkt an die Pmods des Entwicklungsboards liefert. Das Datenformat entspricht der Form YCbCr 4:2:2. Die Kamera wird für das entwickelte System im 8-Bit Interlace Modus konfiguriert (vgl. Tabelle 3).

Die Kamera ist über die *CN701*- und die *CN501*-Schnittstelle mit der Entwicklungsplattform verbunden. Über die *CN701*-Schnittstelle (vgl. Abb. 14) wird die Kamera mit der Betriebsspannung von 6 Volt bis 12 Volt versorgt und ist über eine serielle RS232-Schnittstelle parametrierbar, die zur Konfiguration des Ausgangsbildes bereit steht. Eine Parametrierung erfolgt über das von Sony bereit gestellte VISCA-Protokoll [29]. Über die *CN501*-Schnittstelle (vgl. Abb. 13) werden Datenstrom, Synchronisationssignale und Taktsignal parallel übermittelt. Durch eine Erweiterung des Abtastverhältnisses von 4:2:2 auf 4:4:4 wird die Taktfrequenz von f_{27} = 27 MHz auf $f_{13.5}$ = 13,5 MHz verringert.

Modus	Output	SYNC	Frame Rate	Clock
16 Bit Progressive	YUV 16 Bit 4:2:2	HSYNC/VSYNC	29.97 fps	13.5 MHz
8 Bit Progressive	YUV 8 Bit 4:2:2	HSYNC/VSYNC SAV/EAV	29.97 fps	27 MHz
8 Bit Interlace Scan	**YUV 8 Bit 4:2:2**	**HSYNC/VSYNC SAV/EAV**	**29.97 fps**	**27 MHz**

Tabelle 3: *Digital Image Output Modes [29]*

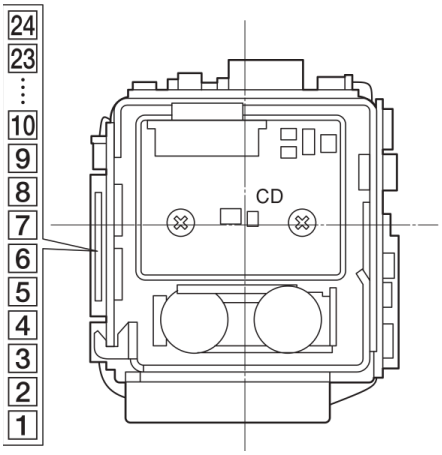

Abb. 13: *CN501 24 FFC Pin Connector [29]*

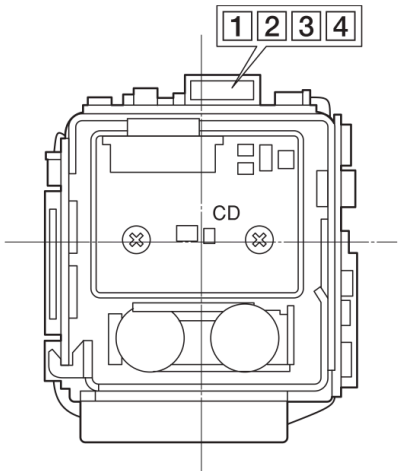

Abb. 14: *CN701 4 Pin Connector [29]*

Pin No.	Name	8-Bit Datenbus	Level	Pin No.	Name	8-Bit Datenbus	Level
1	GND	Signal Ground		13	C2	Hi imp.	
2	Y0	Digital Out 0	0-3.2V	14	C3	Hi imp.	
3	Y1	Digital Out 1	0-3.2V	15	C4	Hi imp.	
4	Y2	Digital Out 2	0-3.2V	16	C5	Hi imp.	
5	Y3	Digital Out 3	0-3.2V	17	C6	Hi imp.	
6	Y4	Digital Out 4	0-3.2V	18	C7	Hi imp.	
7	Y5	Digital Out 5	0-3.2V	19	GND	Signal Ground	
8	Y6	Digital Out 6	0-3.2V	20	VSYNC	Vert. SYNC	0-3.2V
9	Y7	Digital Out 7	0-3.2V	21	HSYNC	Horiz. SYNC	0-3.2V
10	GND	Signal Ground		22	GND	Signal Ground	
11	C0	Hi imp.		23	CLOCK	Clock Signal	0-3.2V
12	C1	Hi imp.		24	GND	Signal Ground	

Tabelle 4: *CN501 Pinbelegung bei 8-Bit Datenbuskonfiguration [29]*

Pin No.	Name	8 Bit Datenbus	Level
1	UNREG	Power Input	6-12V (DC)
2	GND	Signal Ground	
3	TD	Transmit Data	TTL Level (0-5V)
4	RD	Receive Data	TTL Level (0-5V)

Tabelle 5: *CN701 Pinbelegung für Spannungsversorgung sowie Parametrierung der Kamera über RS232 [29]*

4 Bildvorverarbeitung zur Kantenextraktion

Folgender Abschnitt stellt die bestehende Bildvorverarbeitungskette vor und geht hierbei auf die Videopipeline zur Bilddatenaufbereitung sowie die Kantenextraktion mit dem Sobel-Filter ein [17][26]. Es werden weitere Vorverarbeitungsschritte genannt, die vor einer Fahrspurapproximation durchzuführen sind.

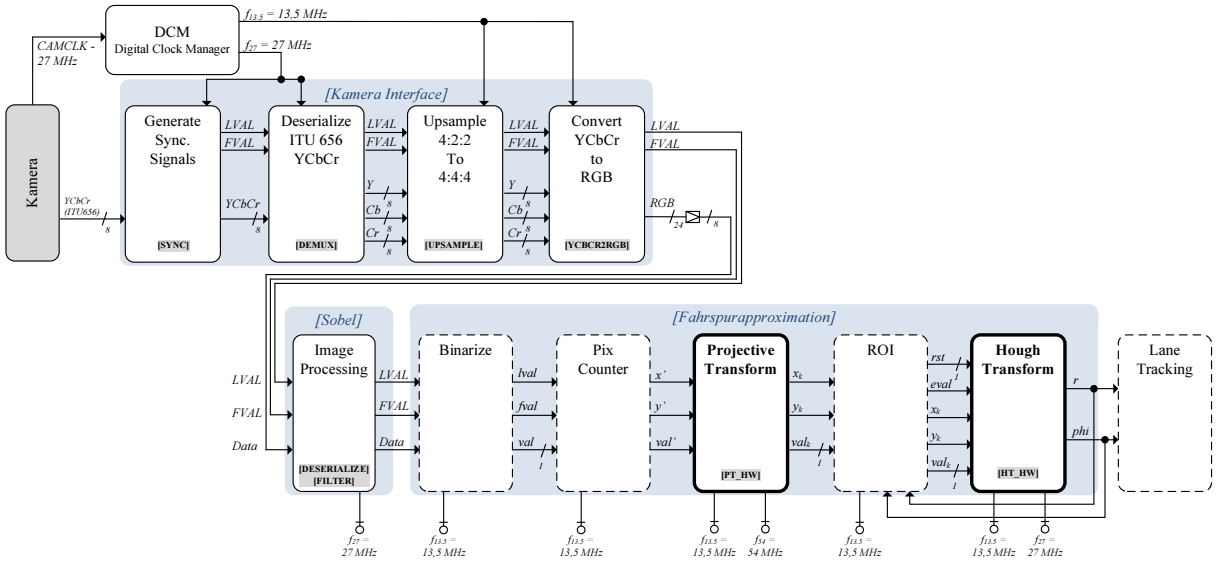

Abb. 15: *Integration der Komponenten zur projektiven Transformation und Hough-Transformation in das bestehende System zur Bildvorverarbeitung.*

4.1 Videopipeline zur Bilddatenaufbereitung

Die Vorverarbeitung der Videodaten erfolgt in einer aus mehreren RTL-Modulen bestehenden Videopipeline (vgl. Abb. 15). Durch eine anschließende Kantenextraktion mit dem Sobel-Filter wird das Bild in eine Form überführt, in der Kantenübergänge hohe Grauwerte und Flächen, in denen keine Kantenübergänge auftreten, niedrige Grauwerte erzeugen.

Die Videodaten werden von der Kamera im YCbCr 4:2:2 Format mit einer Frequenz von f_{27} = 27 MHz bereitgestellt [17]. Im Interlace-Modus (vgl. Tabelle 3) werden in jeder Übertragungsperiode zwei Teilbilder übertragen, wobei diese jeweils nur die Hälfte der Bildzeilen enthalten und durch eine Zeilenverdoppelung auf das Ausgabeformat für einen Monitor von 480 Bildzeilen gebracht werden.

Generierung von Synchronisationssignalen

Die Erzeugung der Synchronisationssignale *lval* (line valid) und *fval* (frame valid), die zur Darstellung auf einem Monitor verwendet werden, erfolgt in dem Modul *Generate Sync. Signals* (vgl. Abb. 15). Das Signal *lval* wird für die Dauer einer gültigen Zeile auf einen *high*-Pegel gesetzt und das Signal *fval* für die Dauer eines gesamten Bildes [17].

Parallelisieren der seriellen Pixeldaten

Im Pixelstrom der Kamera sind die YCbCr-Farbkomponenten seriell angeordnet. Im Deserialisierungsmodul werden die Komponenten parallel angeordnet, wodurch ein 24-Bit YCbCr-Videosignal generiert wird. Das 4:2:2 Abtastverhältnis liefert für jeden zweiten Luminanzwert Y des Videostroms die entsprechenden Chrominanzwerte Cb bzw. Cr, weshalb die Ausgangssignale nach der Parallelisierung nur mit einer halbierten Taktrate von $f_{13.5}$ = 13,5 MHz zur

Verfügung gestellt werden. Chrominanzwerte, zu denen keine Werte im Datenstrom vorliegen, werden zunächst mit Nullen aufgefüllt und in einem Folgeschritt durch interpolierte Werte ersetzt.

Chroma Upsampling

Vor einer Umrechnung der YCbCr-Daten des Videostromes in korrespondierende RGB-Daten werden die Informationen zu den fehlenden Chrominanzwerten aufgefüllt, wobei die Informationen durch Pixelreplikation ermittelt werden. Pixel ohne Chrominanzinformationen übernehmen hierbei die Chrominanzinformationen des vorherigen Pixels [26].

Konvertierung des Pixelstroms in das RGB-Farbmodell

Für die Konvertierung der YCbCr-Werte in RGB-Werte wird eine Umrechnung anhand der folgenden Gleichungen durchgeführt [26]:

$$R = 1,164(Y - 16) + 1,596(Cr - 128) \tag{4}$$
$$G = 1,164(Y - 16) - 0,813(Cr - 128) - 0,391(Cb - 128) \tag{5}$$
$$B = 1,164(Y - 16) + 2,018(Cb - 128) \tag{6}$$

4.2 Kantenextraktion mit dem Sobel-Filter

Die Bildpunkte des Pixelstroms werden vorverarbeitet, um eine Fahrspurapproximation zu ermöglichen. Die Fahrspurmarkierungen unterscheiden sich in ihrer Helligkeit deutlich von der restlichen Fahrspur, so dass eine Fahrspurerkennung auf Basis einer Kantenextraktion durchgeführt werden kann, in der Bildpunkte mit den größten Helligkeitsunterschieden bezogen auf ihre Nachbarn gefunden werden.

Für die Umrechnung der Bildpunkte in Graustufen zur Betonung der Helligkeitsunterschiede zweier Bildpunkte wurde in [17] und [26] kein separates Modul entwickelt, stattdessen wird lediglich der Grün-Anteil der RBG-Information verwendet, der eine höhere Auswirkung auf Helligkeitsanteile hat als die Rot- und Blau-Anteile [17].

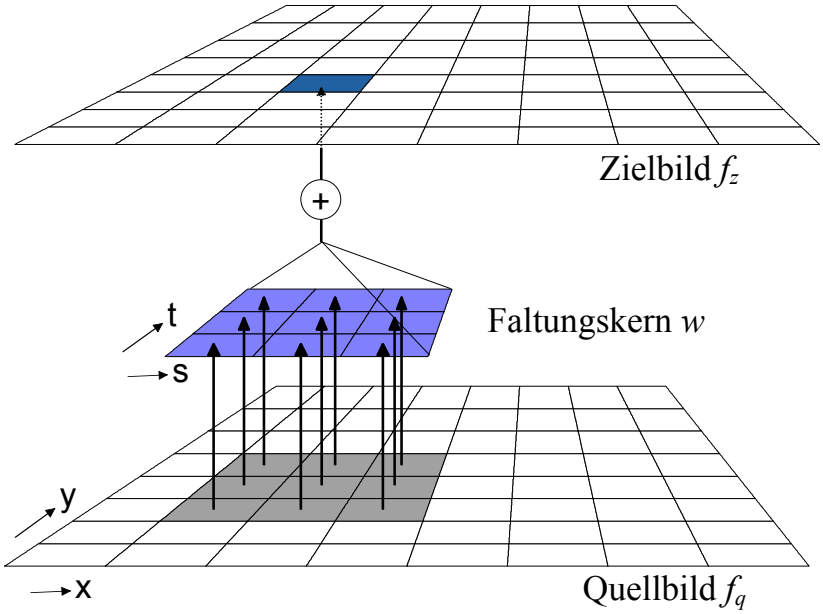

Abb. 16: *Nachbarschaftsoperation mit einem 3x3-Faltungskern [23]*

Eine Nachbarschaftsoperation stellt eine Abbildung eines Quellbildes f_q auf ein Zielbild f_z dar, wobei jedes Pixel im Zielbild f_z durch eine gewichtete Summe seiner Nachbarpixel im Quellbild f_q ersetzt wird (vgl. Abb. 16). Unter Anwendung eines Faltungskerns w mit Seitenlänge $2a + 1$ auf das Quellbild f_q, ergibt sich für das Zielbild f_z:

$$f_z(x, y) = \sum_{s=-a}^{a} \sum_{t=-a}^{a} w(s, t) \cdot f_q(x + s, y + t) \tag{7}$$

Der Sobel-Operator stellt eine Nachbarschaftoperation dar, bei der jeder Bildpunkt auf den Gradienten des Grauwertes seiner Bildposition abgebildet wird. Dadurch werden Kanten die eine starke Grauwertänderung aufweisen hell und durchgehende Flächen durch dunkle Pixel dargestellt. Die Filtermasken G_x (vgl. Gl. 8) und G_y (vgl. Gl. 9) erzeugen Gradientenbilder in horizontaler bzw. vertikaler Richtung [10].

$$G_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} \tag{8}$$

$$G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{pmatrix} \tag{9}$$

Für den Gesamtgradienten G gilt:

$$G = \sqrt{G_x^2 + G_y^2} \tag{10}$$

Zur Vermeidung einer Wurzelberechnung in Hardware wurde auf in [17] und [26] auf diese verzichtet und lediglich eine Addition der Gradienten G_x und G_y durchgeführt (vgl. Abb. 17).

Die Sobel-Filterung des Pixelstroms erfordert die Anwendung der Faltungskerne G_x und G_y. Eine Deserialisierung ermöglicht die Aufnahme aller für die Nachbarschaftsoperation relevanten Pixelwerte in die Berechnungen (vgl. Abb. 17). Über eine Verknüpfung der Ausgangsdaten des Deserialisierungsmoduls mit den Faltungskernmodulen werden die Faltungen parallel ausgeführt und abschließend addiert (vgl. Abb. 17), wobei durch eine Sättigung das Einhalten des 8-Bit *unsigned*-Wertebereichs gewährleistet wird [17].

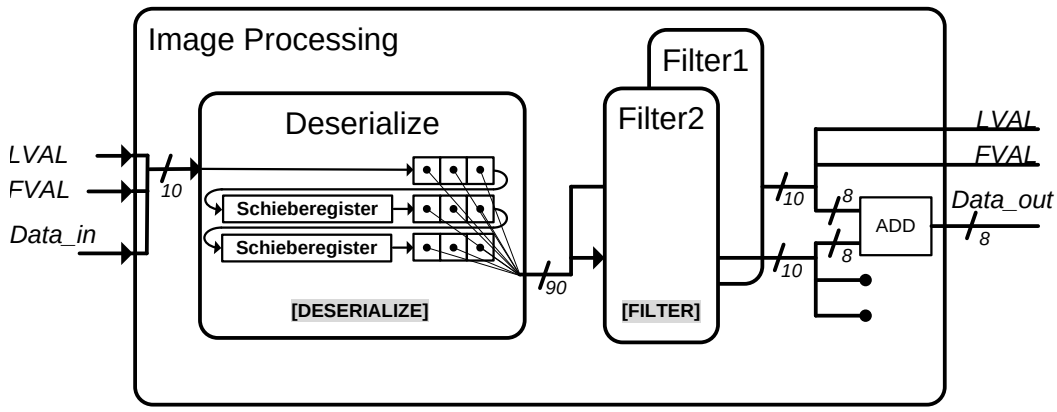

Abb. 17: *Der Pixelstrom wird deserialisiert, um alle Pixelwerte für eine Nachbarschaftsoperation zur Verfügung zu stellen; die Faltung mit den Kernen* $Filter1$ *und* $Filter2$ *erfolgt parallel [17]*

4.3 Vorverarbeitungsschritte zur Fahrspurapproximation

Eine Fahrspurapproximation erfolgt auf Basis weiterer Verarbeitungsschritte, die in Abbildung 15 dargestellt sind:

- Durch eine Binarisierung, in der 8-Bit Grauwertdaten über einen Schwellwert in eine binäre Farbinformation gewandelt werden, wird ein Binärbild erzeugt, in dem Pixelwerte nur noch ein Bit zur Darstellung verwenden.
- Die Fahrspurapproximation mit der Hough Transformation basiert auf der Kenntnis von Pixelkoordinaten in x- und y-Richtung. Mit den Signalen $fval$ und $lval$, die sowohl Bild- als auch Zeilenanfang signalisieren, können Koordinaten über Zählermodule verfügbar gemacht werden.
- Durch eine projektive Transformation, die eine Abbildung von Koordinaten (x', y') in der Bildebene in perspektivisch korrigierte Koordinaten (x_k, y_k) in der Fahrzeugebene darstellt, wird eine Korrektur perspektivischer Verzeichnungen des Bildes durchgeführt.
- Jede projektiv transformierte Koordinate (x_k, y_k) wird auf Zugehörigkeit zur *Region of Interest* überprüft (vgl. Anhang B). Eine dynamische Anpassung der *Region of Interest* kann über eine Rückführung der Transformationsergebnisse aus der Komponente zur Berechnung der Hough-Transformation im Fahrbetrieb erfolgen.

5 Korrektur der Linsenverzeichnung und der perspektivischen Verzerrung

Die Approximation der Fahrspur basiert auf dem Zusammenhang zwischen Koordinaten in der Fahrzeugebene und der Bildebene, die über eine Kameraausrichtung in einem konstanten Verhältnis zueinander stehen:

- Die **Fahrzeugebene**, in der sich die Fahrspur befindet und in der sich das Fahrzeug bewegt.
- Die **Bildebene**, die der Ebene entspricht, auf die Punkte in der Fahrzeugebene im Kamerabild abgebildet werden.

Durch die Linsenkrümmung des Objektivs der Kamera ist eine Linsenverzeichnung bedingt, die sich stärker auswirkt, je weitwinkliger das verwendete Objektiv ist und zu einer Verfälschung der ermittelten Messwerte führen kann. Im Folgenden werden die Verfahrensschritte zur Korrektur der Linsenverzeichnung vorgestellt.

Die Durchführung einer Fahrspurapproximation erfordert ein perspektivisch korrigiertes Bild, um eine Interpretation der ermittelten Fahrspur durchführen zu können und eine Spurführung zu realisieren. Durch die Abbildung von Punkten in der Fahrzeugebene auf Punkte in der Bildebene entsteht eine perspektivische Verzeichnung, die durch eine Projektion ausgeglichen wird. Im Folgenden wird das mathematische Verfahren zur projektiven Transformation erläutert und eine Realisierung mit dem System Generator vorgestellt.

5.1 Darstellung der Bild- und Fahrzeugebene

Die zu approximierende Fahrspur befindet sich in einer Ebene und weist keine Höhenunterschiede auf [5]. Die Kamera ist in einem konstanten Winkel am Fahrzeug angebracht, so dass stets ein konstantes Verhältnis zwischen Bild- und Fahrzeugebene besteht. Unebenheiten in der Fahrspur verändern das Verhältnis der Bildebene zur Fahrzeugebene, was zu einer Verfälschung der Berechnungen führt und eine Fahrspurapproximation auf Basis fehlerhafter Verhältnisse der Ebenen zueinander verursacht.

Die Korrektur der Linsenverzeichnung basiert auf der Kenntnis des Bildhauptpunktes, der dem Durchstoßpunkt der optischen Achse durch die Bildebene entspricht [21]. Zur Durchführung der Linsenverzeichnungskorrektur werden daher das Bild- und das Bildhauptpunktkoordinatensystem eingeführt, in denen alle von der Kamera zu einem Zeitpunkt wahrgenommenen Pixel angeordnet sind (vgl. Abb. 18).

Bildkoordinatensystem

Koordinaten liegen als zweidimensionale Vektoren vor. Die Angabe der Koordinaten (x_b, y_b) erfolgt in Pixeln, wobei der maximale Pixelwert in x-Richtung der Bildbreite und der maximale Pixelwert in y-Richtung der Bildhöhe entspricht. Der Koordinatenursprung ist die obere linke Ecke des Bildes.

Bildhauptpunktkoordinatensystem

Die Angabe der zweidimensionalen Koordinaten (x_{bh}, y_{bh}) erfolgt in einer metrischen Einheit. Der Koordinatenursprung liegt im Bildhauptpunkt [21].

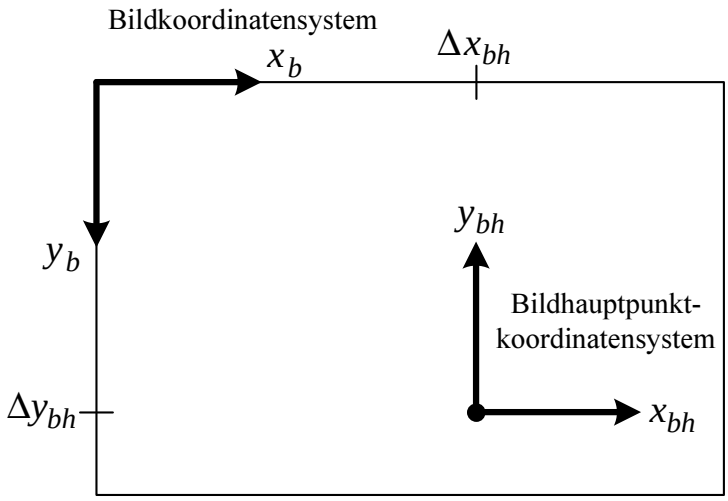

Abb. 18: *Bildkoordinatensystem, mit Ursprung in der oberen linken Ecke des Bildes und Bildhauptpunktkoordinatensystem mit Ursprung im Bildhauptpunkt*

5.2 Kalibrierung der internen Kameraparameter

Bedingt durch die Linsenkrümmung einer Kamera entsteht eine radialsymmetrische und tangentiale Linsenverzeichnung [19]. Als Effekt der radialen Verzeichnung treten Tonnen- und Kisseneffekte im Bild auf, während der Effekt der tangentialen Verzeichnung einer Verdrehung der Bildpunkte um den Verzerrungsmittelpunkt entspricht. Durch eine Kalibrierung werden die internen Parameter der Kamera bestimmt, die zu einer Verzeichnungskorrektur und damit einer Genauigkeitserhöhung genutzt werden. Aufgrund des Fertigungsprozesses variieren diese auch bei baugleichen Kameras [15]. Die internen Parameter lassen sich nicht direkt messen und werden berechnet.

- **Bildhauptpunktkoordinaten** (vgl. Abb. 18), $P_{bh} = (\Delta x_{bh}, \Delta y_{bh})$ in mm, die den Durchstoßpunkt der optischen Achse durch die Bildebene angeben.
- **Linsenverzeichnungsparameter** K_1, K_2 (radial) sowie P_1, P_2 (tangential), wobei eine tangentiale Verzeichnung eine Verdrehung der Bildpunkte und eine radiale Verzeichnung tonnen- und kissenförmige Effekte bewirkt.

Die internen Kameraparameter werden mit der photogrammetrischen Software *PhotoModeler* bestimmt [8], die ein Kalibriermuster zur Verfügung stellt, auf dem Referenzpunkte eindeutig gekennzeichnet sind und so eine Identifizierung der Punkte aus verschiedenen Perspektiven ermöglicht (vgl. Abb. 19).

Zur Kalibrierung werden Aufnahmen des Kalibriermusters von allen vier Seiten sowie jeweils eine um $90°$ gedrehte Aufnahme verwendet [8]. Daraus resultieren acht Aufnahmen, aus denen der *PhotoModeler* die internen Parameter bestimmt (vgl. Tabelle 6) [20].

Bildhauptpunkt Δx_{bh} in mm	**Bildhauptpunkt** Δy_{bh} in mm	**K1**	**K2**	**P1**	**P2**
3.103003	2.277242	0.005379	-0.000114	-0.000048	0.000018

Tabelle 6: *Interne Parameter für die Plattformkamera Sony FCB-PV10*

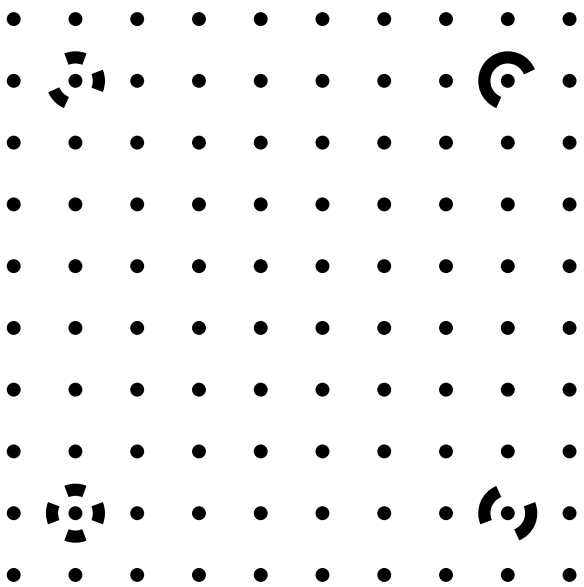

Abb. 19: *Kalibriermuster zur Kalibrierung der internen Parameter einer Kamera [8]*

5.3 Linsenverzeichnungskorrektur zur Genauigkeitserhöhung

Die internen Kameraparameter $(\Delta x_{bh}, \Delta y_{bh}, K1, K2, P1, P2)$ (vgl. Kap. 5.2) werden für die Transformation verzeichneter Bildpunkte in verzeichnungsfreie Bildpunkte verwendet. Die Überführung eines realen, verzeichneten Bildpunktes $(x_v^{\#}, y_v^{\#})$ in Pixelkoordinaten in einen korrigierten Bildpunkt $(x_k^{\#}, y_k^{\#})$, ebenfalls in Pixelkoordinaten, wird in drei Schritten durchgeführt [22].

Transformation in das Bildhauptpunktkoordinatensystem

Die verzeichneten Bildpunktkoordinaten $(x_v^{\#}, y_v^{\#})$ in Pixelkoordinaten werden in ein metrisches Koordinatensystem überführt, in dem der Koordinatenursprung im Bildhauptpunkt liegt. Begründet ist diese Transformation in der vom Bildhauptpunkt nach außen größer werdenden radialen Linsenverzeichnung [21].

$$x_v = \left(\frac{x_v^{\#}}{s_x} - x_{bh} \right) \cdot 1000 \tag{11}$$

$$y_v = \left(\frac{y_v^{\#}}{s_y} - y_{bh} \right) \cdot 1000 \tag{12}$$

Wobei s_x der Anzahl Pixel pro Meter Chipbreite und s_y der Anzahl Pixel pro Meter Chiphöhe entspricht. Eine Normierung der metrischen Bildpunkte für eine Berechnung im Millimeterbereich wird durch eine Multiplikation mit 1000 erreicht.

Verzeichnungskorrektur

Die verzeichneten metrischen Punktkoordinaten (x_v, y_v) werden in korrigierte metrische Koordinaten (x_k, y_k) umgerechnet.

$$x_k = x_v \cdot (1 + K_1 \cdot r^2 + K_2 \cdot r^4) + P_1 \cdot (r^2 + 2x_v^2) + 2 \cdot P_2 \cdot x_v \cdot y_v \tag{13}$$

$$y_k = y_v \cdot (1 + K_1 \cdot r^2 + K_2 \cdot r^4) + P_2 \cdot (r^2 + 2y_v^2) + 2 \cdot P_1 \cdot x_v \cdot y_v \tag{14}$$

mit

$$r = \sqrt{x_v^2 + y_v^2} \tag{15}$$

Rücktransformation ins Bildkoordinatensystem

Die Umrechnung der metrischen Punktkoordinaten (x_k, y_k) in Pixelkoordinaten $(x_k^\#, y_k^\#)$ erfolgt mit:

$$x_k^\# = \left(\frac{x_k}{1000} + x_{bh} \right) \cdot s_x \tag{16}$$

$$y_k^\# = \left(\frac{y_k}{1000} + y_{bh} \right) \cdot s_y \tag{17}$$

Das Resultat der Matlab-Simulation der Linsenverzeichnungskorrektur ist in Abbildung 21 dargestellt. Das im verzeichnungsfreien Bild sichtbare Linienmuster entsteht, da das Bild mit schwarzen Pixeln vorinitialisiert ist und kein korrigierter Bildpunkt der verzeichnungsfreien Bildpunktmenge auf diesen Linien liegt. Das Objektiv der Sony FCB-PV10 Block-Kamera besitzt eine geringe Weitwinkligkeit, so dass keine starke Linsenverzeichnung entsteht, wodurch eine hinreichend genaue projektive Transformation und Hough-Transformation auch ohne Linsenverzeichnungskorrektur durchführbar ist (vgl. Abb. 20 und 21). Eine Hardwareimplementierung der Linsenverzeichnungskorrektur wird aus diesem Grunde in dieser Arbeit nicht realisiert.

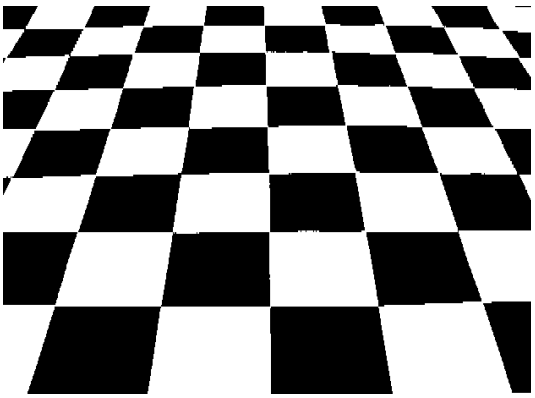

Abb. 20: *Linsenverzeichnetes Originalbild*

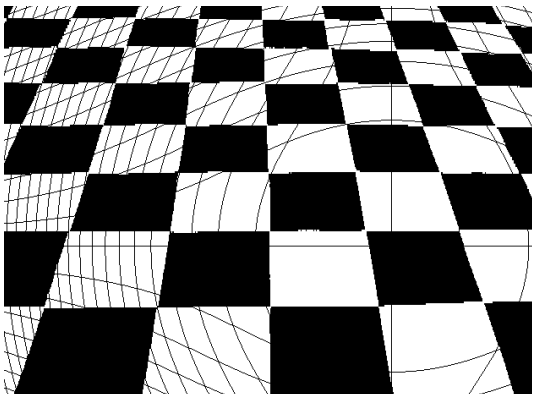

Abb. 21: *Linsenverzeichnungskorrigiertes Bild*

5.4 Projektive Transformation zwischen Bild- und Fahrzeugebene

Punkte in der Fahrzeugebene sind, bedingt durch den Abbildungsprozess der Kamera, in einer anderen Ebene angeordnet als die von der Kamera aufgenommenen Bildebenenpunkte. Die Approximation der Fahrspur wird in der Fahrzeugebene durchgeführt, da sich in der Bildebene eine konische Verzeichnung des Bildes ergibt, die eine Auswertung der ermittelten Messwerte erschwert (vgl. Abb. 22). Eine Entzerrung der perspektivischen Verzeichnung wird durch eine projektive Transformation zwischen Bild- und Fahrzeugebene erreicht, die im Folgenden beschrieben wird.

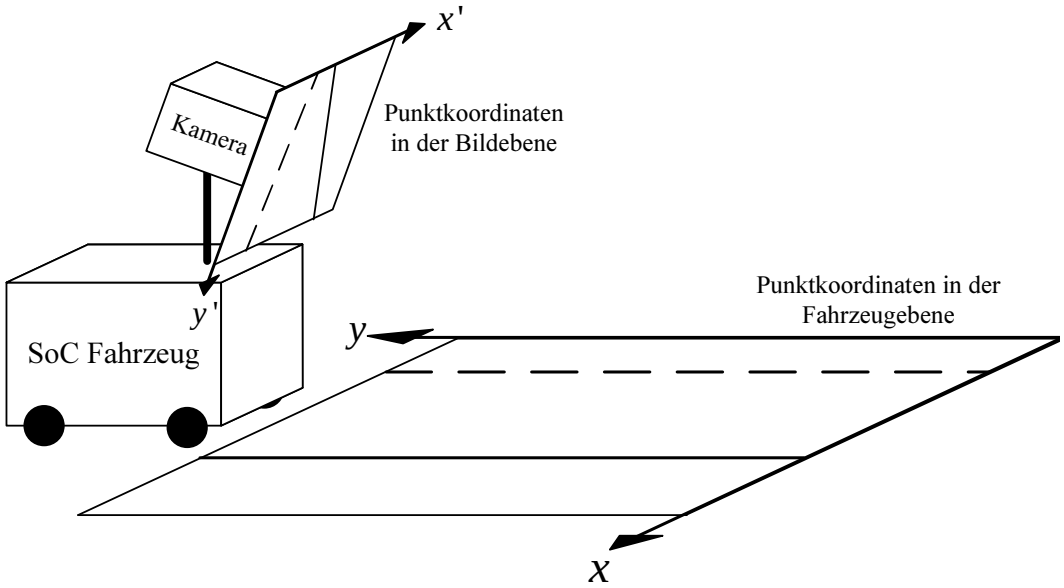

Abb. 22: *Zur Approximation der Fahrspur wird das Kamerabild, das sich in der Bildebene befindet, in die Fahrzeugebene transformiert*

5.4.1 Projektive Transformation mit homogenen Koordinaten

Ein Punkt $\vec{p}_k = (x_k, y_k)$ in kartesischen Koordinaten wird durch die Erweiterung um eine zusätzliche Dimension zu einem Punkt $\vec{p}_h = (x_h, y_h, 1)$ in homogenen Koordinaten. Ein n-dimensionaler Vektor in kartesischen Koordinaten wird so durch einen $(n+1)$-dimensionalen Vektor in homogenen Koordinaten beschrieben. Wird ein zweidimensionaler Vektor um eine dritte Dimension ergänzt, lässt sich eine Translation als lineare Abbildung durchführen.

Ein Vektor ist homogen, wenn gilt:

$$\begin{pmatrix} x_h \\ y_h \\ 1 \end{pmatrix} \quad \Leftrightarrow \quad \begin{pmatrix} sx_h \\ sy_h \\ s \end{pmatrix} \qquad \text{für alle } s \neq 0 \tag{18}$$

In einem homogenen Vektor wird s als Skalierungsfaktor bezeichnet. Zur Transformation eines Punktes $\vec{p}_h$ in homogenen Koordinaten in einen Punkt $\vec{p}$ in kartesischen Koordinaten werden die homogenen Koordinaten durch den Skalierungsfaktor s dividiert.

$$\vec{p}_h = \begin{pmatrix} sx_h \\ sy_h \\ s \end{pmatrix} \qquad \rightarrow \qquad \vec{p}_k = \begin{pmatrix} x_k = \frac{x_h}{s} \\ y_k = \frac{y_h}{s} \end{pmatrix} \tag{19}$$

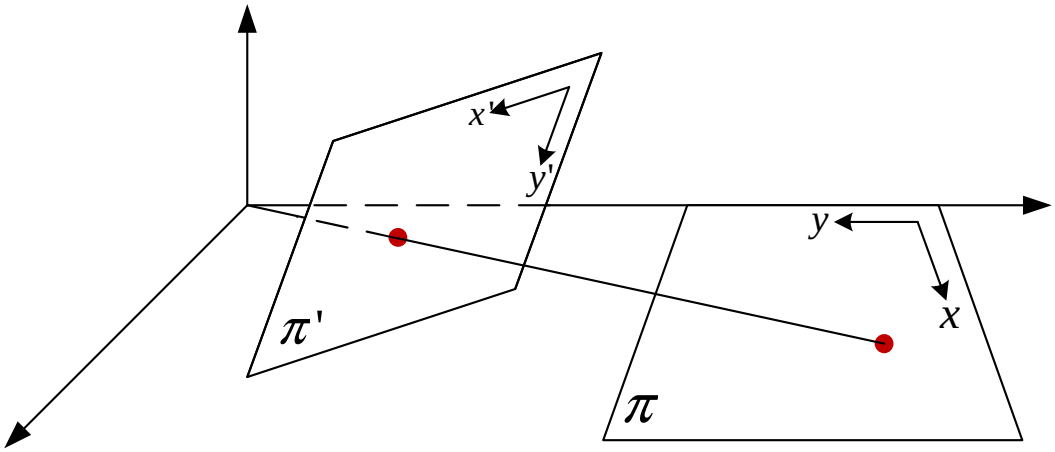

Abb. 23: *Eine Projektion bildet Punkte in einer Ebene π' auf Punkte in einer anderen Ebene π ab. Bildstrukturen bleiben projektiv äquivalent erhalten.*

Eine Projektion beschreibt eine umkehrbare Abbildung h von einer Projektionsebene π' in eine andere Projektionsebene π (vgl. Abb. 23). Punkte, die vor der Projektion auf einer Gerade liegen, liegen auch nach der Projektion auf einer Gerade (vgl. Abb. 23). Eine projektive Transformation der Ebene ist eine lineare Transformation mit homogenen dreidimensionalen Vektoren und einer nichtsingulären 3x3-Matrix der Form:

$$\vec{x}_h = \mathbf{H}\vec{x'_h} \Leftrightarrow \begin{pmatrix} x_h \\ y_h \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \cdot \begin{pmatrix} x'_h \\ y'_h \\ 1 \end{pmatrix} \tag{20}$$

Die als Transformationsmatrix bezeichnete Matrix $\mathbf{H}$ kann durch Multiplikation eines Skalierungsfaktors $s \neq 0$ verändert werden ohne die Projektion zu verändern. Durch Division aller Matrixelemente durch den Projektionsfaktor h_{33} reduziert sich die Matrix auf acht unbekannte Elemente.

$$\vec{x}_h = \mathbf{B}\vec{x'_h} \Leftrightarrow \begin{pmatrix} x_h \\ y_h \\ 1 \end{pmatrix} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & 1 \end{pmatrix} \cdot \begin{pmatrix} x'_h \\ y'_h \\ 1 \end{pmatrix}, \text{ mit } b_{ij} = \frac{h_{ij}}{h_{33}} \tag{21}$$

Die kartesischen Koordinaten (x_k, y_k) ergeben sich mit Gleichung 19 aus den homogenen Koordinaten (x_h, y_h). Eine Skalierung der homogenen Koordinaten (x_h, y_h) mit $s = 1$ ergibt für die kartesischen Koordinaten (x_k, y_k):

$$x_k = \frac{x_h}{1} = \frac{b_{11}x'_h + b_{12}y'_h + b_{13}}{b_{31}x'_h + b_{32}y'_h + 1} \tag{22}$$

$$y_k = \frac{y_h}{1} = \frac{b_{21}x'_h + b_{22}y'_h + b_{23}}{b_{31}x'_h + b_{32}y'_h + 1} \tag{23}$$

Unter Verwendung der Gleichungen 22 und 23 wird für jeden Punkt (x'_k, y'_k) in der Bildebene der in die Fahrzeugebene projizierte Punkt (x_k, y_k) bestimmt.

5.4.2 Kalibrierung der Transformationsmatrix B

Die unbekannten Elemente der Transformationsmatrix **B** (vgl. Gl. 21) werden durch eine Kalibrierung bestimmt. Das Verhältnis der Bildebene zur Fahrzeugebene muss zu jedem Zeitpunkt identisch mit dem Verhältnis zum Zeitpunkt der Kalibrierung sein. Eine Veränderung der Kameraausrichtung führt zu fehlerhaften Transformationsergebnissen und zieht eine Neukalibrierung der Transformationsmatrix **B** nach sich. Als Gegenmaßnahme ist eine konstante Kamerakalibrierung auf dem Fahrzeug zu gewährleisten.

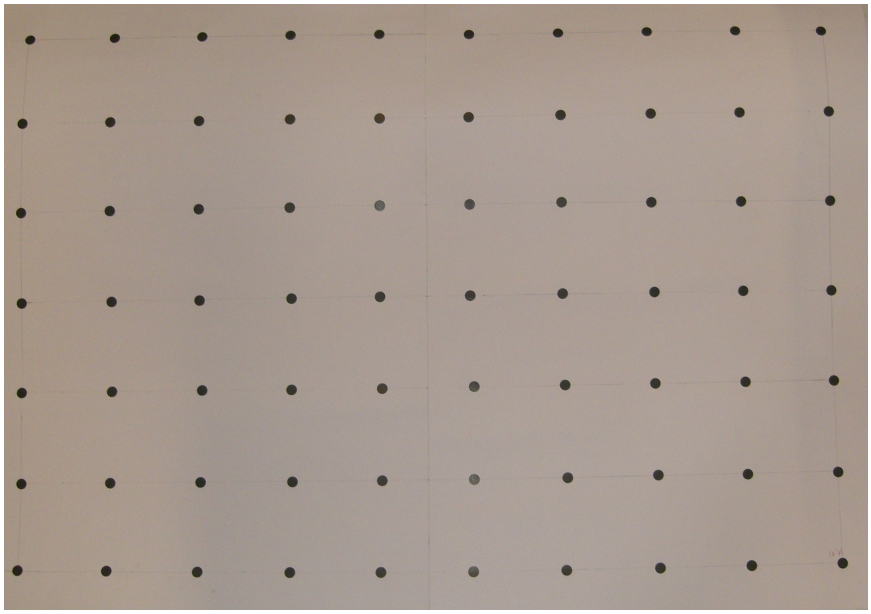

Abb. 24: *Draufsicht auf die in der Fahrzeugebene vorliegende Kalibrieranordnung; die Punkte befinden sich in x- und y-Richtung in einem Abstand von 15 cm zueinander.*

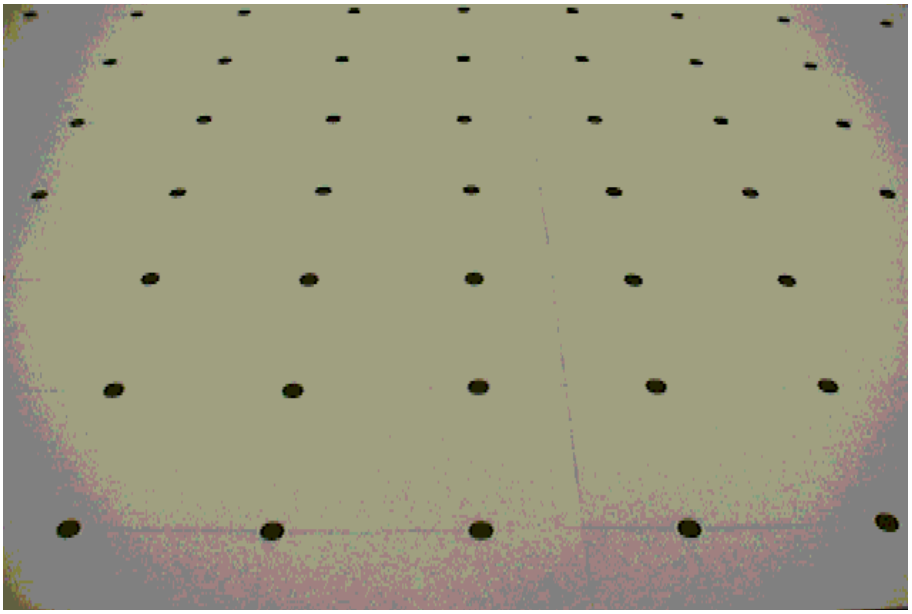

Abb. 25: *Abbildung der in der Fahrzeugebene bekannten Kalibrierpunkte auf die Bildebene; die Kamera befindet sich in einer fest kalibrierten Position, die der Position auf dem Fahrzeug entspricht.*

Zur Kalibrierung wird eine Punktanordnung mit bekannten Koordinaten in der Fahrzeugebene erstellt. Im Kamerabild der Kalibrieranordnung werden die korrespondierenden Punkte ermittelt (vgl. Abb. 25). Mit den Punktkorrespondenzen wird ein lineares Gleichungssystem aufgestellt, deren Grundlage die Gleichungen 22 und 23 darstellen, die schrittweise umgestellt werden, um alle b_{ij} auf eine Seite zu bringen. Unter Verwendung der Gleichung 22 folgt:

$$x_k = \frac{x_h}{1} = \frac{b_{11}x'_k + b_{12}y'_k + b_{13}}{b_{31}x'_k + b_{32}y'_k + 1}$$

$$\Leftrightarrow \quad x_k(b_{31}x'_k + b_{32}y'_k + 1) = b_{11}x'_k + b_{12}y'_k + b_{13}$$

$$\Leftrightarrow \quad x_k = b_{11}x'_k + b_{12}y'_k + b_{13} - b_{31}x'_k x_k - b_{32}y'_k x_k \tag{24}$$

Analog ergibt sich für y_k unter Verwendung von Gleichung 23:

$$y_k = \frac{y_h}{1} = \frac{b_{21}x'_k + b_{22}y'_k + b_{23}}{b_{31}x'_k + b_{32}y'_k + 1}$$

$$\Leftrightarrow \quad y_k(b_{31}x'_k + b_{32}y'_k + 1) = b_{21}x'_k + b_{22}y'_k + b_{23}$$

$$\Leftrightarrow \quad y_k = b_{21}x'_k + b_{22}y'_k + b_{23} - b_{31}x'_k x_k - b_{32}y'_k x_k \tag{25}$$

Aus den Gleichungen 24 und 25 wird ein lineares Gleichungssystem in Matrixform erstellt, wobei für jedes korrespondierende Punktpaar eine Gleichung für die x-Komponente und eine Gleichung für die y-Komponente entstehen. Um ein Gleichungssystem mit acht Unbekannten b_{ij} lösen zu können werden mindestens acht Gleichungen benötigt. Mit $n \geq 4$ korrespondierenden Punkten können die b_{ij} bestimmt werden.

$$\begin{pmatrix} x'_{k_1} & y'_{k_1} & 1 & 0 & 0 & 0 & -x'_{k_1}x_{k_1} & -y'_{k_1}x_{k_1} \\ 0 & 0 & 0 & x'_{k_1} & y'_{k_1} & 1 & -x'_{k_1}x_{k_1} & -y'_{k_1}x_{k_1} \\ x'_{k_2} & y'_{k_2} & 1 & 0 & 0 & 0 & -x'_{k_2}x_{k_2} & -y'_{k_2}x_{k_2} \\ 0 & 0 & 0 & x'_{k_2} & y'_{k_2} & 1 & -x'_{k_2}x_{k_2} & -y'_{k_2}x_{k_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_{k_n} & y'_{k_n} & 1 & 0 & 0 & 0 & -x'_{k_n}x_{k_n} & -y'_{k_n}x_{k_n} \\ 0 & 0 & 0 & x'_{k_n} & y'_{k_n} & 1 & -x'_{k_n}x_{k_n} & -y'_{k_n}x_{k_n} \end{pmatrix} \cdot \begin{pmatrix} b_{11} \\ b_{12} \\ b_{13} \\ b_{21} \\ b_{22} \\ b_{23} \\ b_{31} \\ b_{32} \end{pmatrix} = \begin{pmatrix} x_{k_1} \\ y_{k_1} \\ x_{k_2} \\ y_{k_2} \\ \vdots \\ x_{k_n} \\ y_{k_n} \end{pmatrix} \tag{26}$$

Durch die Verwendung von mehr als vier Punktkorrespondenzen wird der Fehler gering gehalten, der durch Kollinearität der Punkte und Ungenauigkeiten bei der Bestimmung von korrespondierenden Koordinaten entsteht. Durch Einsetzen der Punktkorrespondenzen entsteht ein überbestimmtes Gleichungssystem, deren Lösung, unter Anwendung der linearen Ausgleichsrechnung, die Transformationsmatrix **B** ist und deren Elemente einen minimalen quadratsichen Fehler bezüglich der gewählten Punktkorrespondenzen besitzen (vgl. Anhang A).

5.5 Projektive Transformation mit dem System Generator

Das Modul zur Berechnung der projektiven Transformation bildet einen Punkt (x'_h, y'_h) in der Bildebene in einen Punkt (x_k, y_k) in der Fahrzeugebene ab (vgl. Gl. 22 und 23). Die binarisierte Farbinformation val' wird der Latenz der projektiven Transformation entsprechend verzögert und als val_k an Folgemodule weiter gereicht (vgl. Abb. 26).

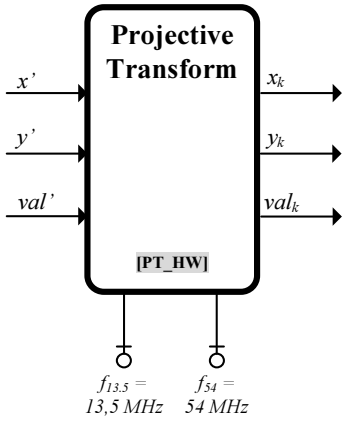

Abb. 26: *Blackboxsymbol des Moduls zur Berechnung der projektiven Transformation nach Gleichung 22 und23 mit zwei Takteingängen*

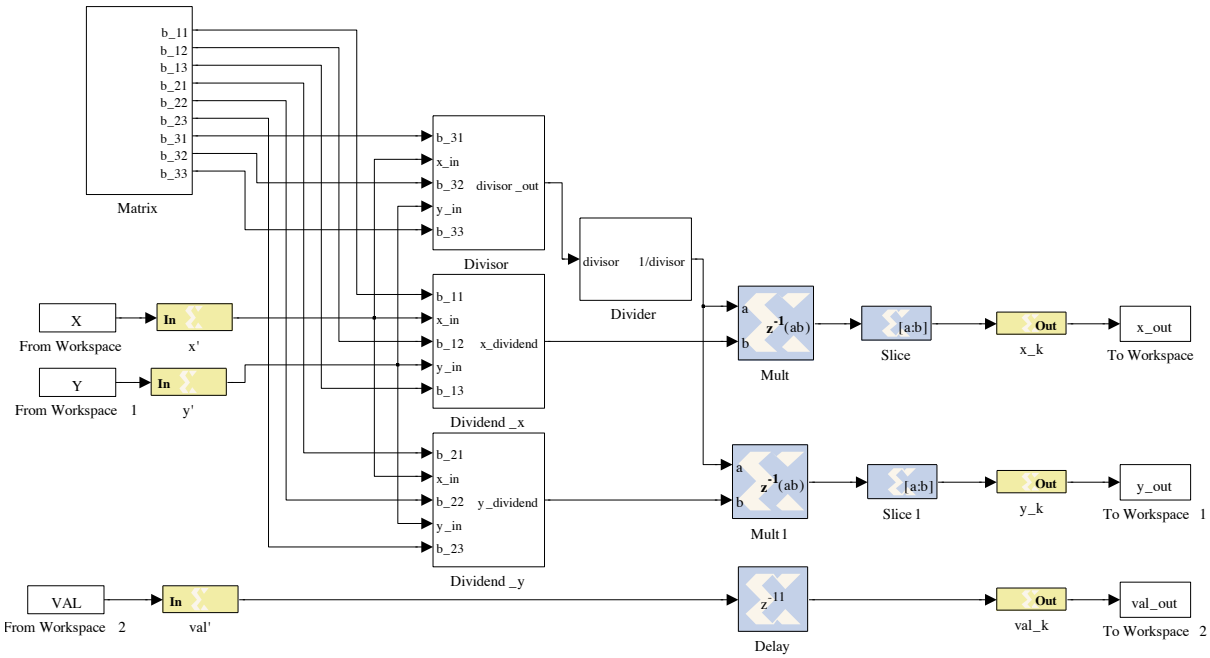

Abb. 27: *Die projektive Transformation mit Transformationsmatrix **B**, Divisor- und Dividendenberechnung sowie Multiplizierern für Kehrwertmultiplikation*

Die Transformationsmatrix **B** geht in eine Dividenden- und Divisorberechnung ein, so dass eine Multiplikation der Dividenden für x_k und y_k mit dem Kehrwert des Divisors durchzuführen ist, um eine Umsetzung der Gleichungen 22 und 23 zu erreichen.

Für die Berechnung der Vektorbreite der Matrixkoeffizienten b_{ij} in einer *signed* Q-Formatsrepräsentation gilt:

$$Vektorbreite = \underbrace{1}_{\text{Vorzeichen}} + \underbrace{ceil(log2(B(m,n)+1))}_{\text{Anzahl Bits Integeranteil}} + \underbrace{fractionals}_{\text{Anzahl Bits Nachkommaanteil}} \tag{27}$$

Die Addition einer 1 verhindert das Logarithmieren einer 0. Die Anzahl der Nachkommastellen $fractionals$ wird über eine Variable in einem Matlab-Skript festgelegt und gilt für alle Transformationsmatrixkoeffizienten. Alle weiteren Bitbreiten sind resultierend aus der Breite der Koordinaten x' und y', der Matrixkoeffizientenbreite und der arithmetischen Operationen mit voller Bitbreite als *full precision* ausgelegt (vgl. Kapitel 2.2), so dass der System Generator die Breite des Ergebnistypen so festlegt, dass kein Präzisionsverlust auftritt [35]. Dieses bietet Optimierungspotential für zukünftige Arbeiten.

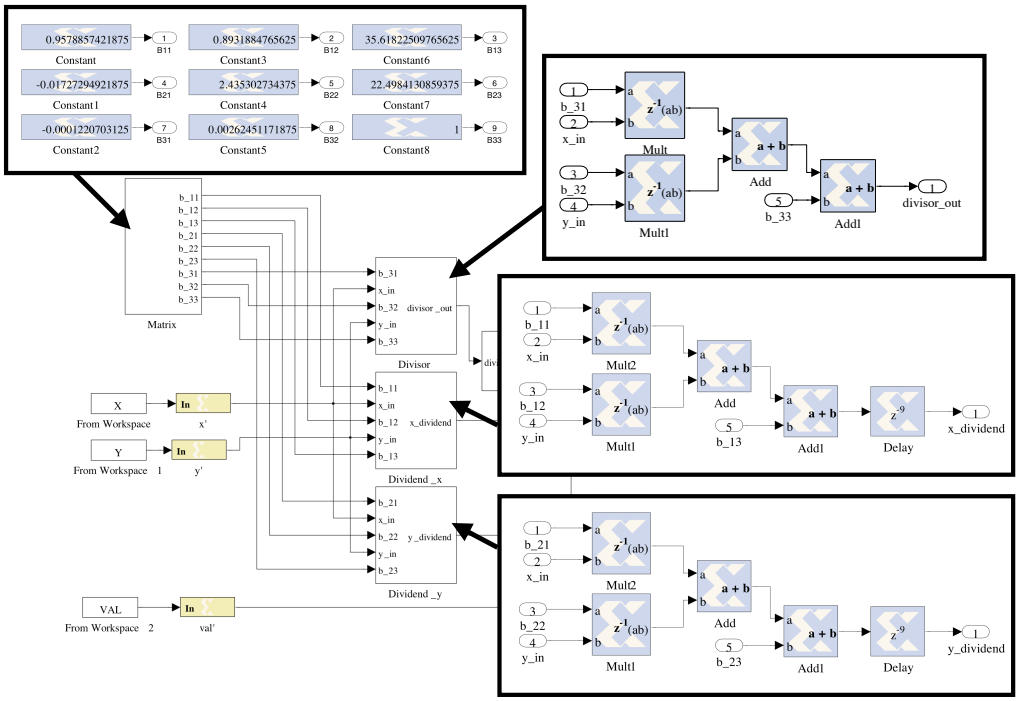

Abb. 28: *Detailansicht der Divisor- und Dividendenberechnung sowie der Realisierung der Transformationsmatrix* ***B***

Zur Berechnung von x_k und y_k, wird jede Berechnung in zwei Teilschritte zerlegt:

- Nebenläufige Berechnung des Divisors $b_{31}x' + b_{32}y' + 1$ sowie der Dividenden $b_{11}x' + b_{12}y' + b_{13}$ und $b_{21}x' + b_{22}y' + b_{23}$ (vgl. Abb. 28)
- Multiplikation der Dividenden $b_{11}x' + b_{12}y' + b_{13}$ und $b_{21}x' + b_{22}y' + b_{23}$ mit dem Kehrwert des Divisors (vgl. Abb. 29)

$$\frac{1}{b_{31}x' + b_{32}y' + 1} \tag{28}$$

Das Ergebnis der Dividendenberechnung $b_{11}x' + b_{12}y' + b_{13}$ und $b_{21}x' + b_{22}y' + b_{23}$ wird zum Latenzausgleich um die Dividiererlatenz verzögert. Der Divisor $b_{31}x' + b_{32}y' + 1$ ist für sowohl für die Berechnung des korrigierten Pixels in x-Richtung x_k, als auch für die Berechnung in y-Richtung y_k gültig. Durch eine Multiplikation mit dem Kehrwert des Divisors (vgl. Gl. 28)

wird nur ein Hardware-Dividierer verwendet. Eine weitere Verringerung des Ressourcenbedarfs des Dividierers wird erreicht, wenn kein Durchsatz von einem Divisionsergebnis pro Takt gefordert ist, da der Dividierer so für Ressource-Sharing optimiert wird [32].

Das Dividierersubsystem wird als Multiratensystem mit zwei Taktfrequenzen ausgelegt (vgl. Abb. 29). Der Dividierer wird über ein Upsampling mit einer vierfachen Pixeltaktfrequenz von f_{54} = 54 MHz betrieben. Das Divisionsergebnis wird über ein Downsampling anschließend wieder mit der Pixelfrequenz von $f_{13.5}$ = 13,5 MHz abgetastet. Der Dividierer wird mit einem Durchsatz von vier Takten pro Division konfiguriert, liefert jedoch aufgrund des vierfachen Übertaktens auf eine Frequenz f_{54} mit jedem Pixeltakt der Frequenz $f_{13.5}$ ein aktualisiertes Divisionsergebnis.

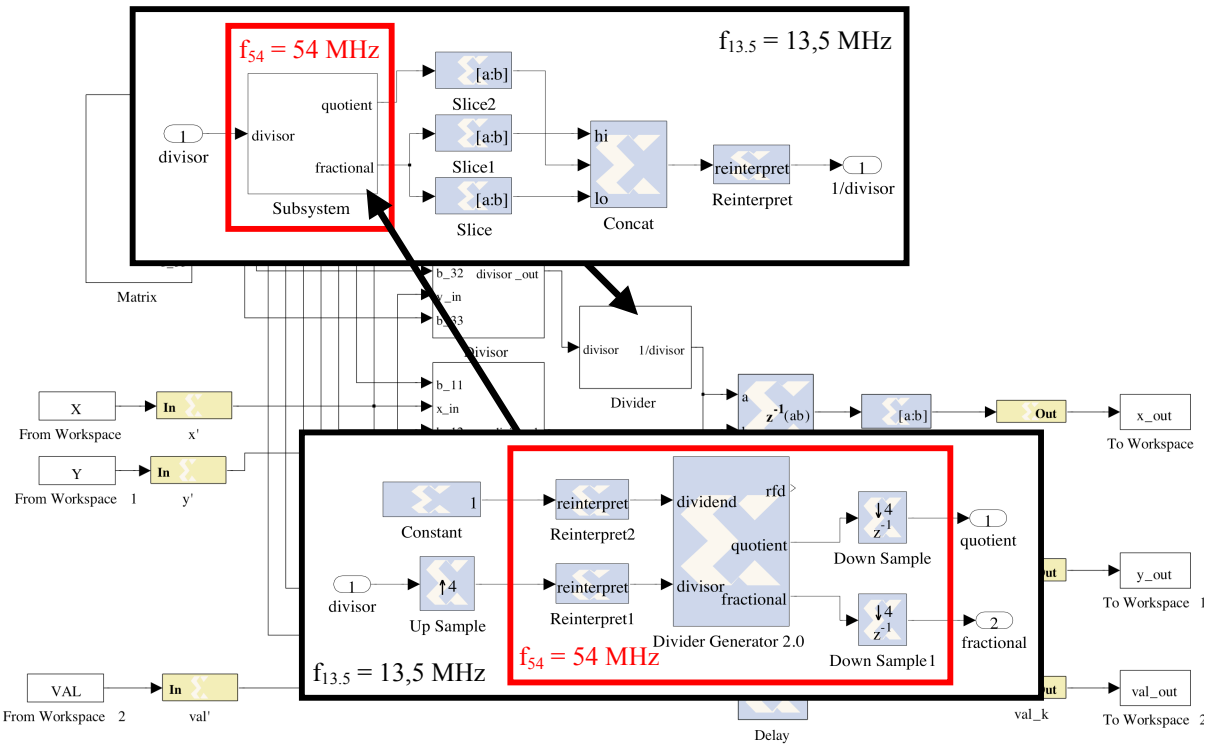

Abb. 29: *Detailansicht des Subsystems zur Kehrwertbildung mit zwei Frequenzbereichen*

Zur Generierung eines Dividieres steht im System Generator der Divider Generator 2.0 zur Verfügung, der auf dem Xilinx Spartan-3E XC3S1200E FPGA einen Dividierer nach dem *Radix-2* Algorithmus implementiert [32]. Der *Radix-2* Algorithmus löst ein Bit des Quotienten pro Takt, indem Additionen und Subtraktionen verwendet werden. Durch ein Pipelining wird ein Durchsatz von einer Division pro Takt ermöglicht. Das Ergebnis der Division besteht aus dem Integeranteil des Quotienten sowie dem Integer-Rest der Division oder dem Fractional-Anteil.

Der Dividierer ist mit einer Latenz behaftet, die sich aus der Breite des Dividenden M und der Breite der Nachkommastellen F berechnet. Für die Berechnung des Kehrwertes des Dividenden wird ein Nachkommaanteil $fractionals$ von 14 Bit vereinbart, der eine hinreichend genaue Darstellung ermöglicht:

- **Breite des Dividenden** M**:** $fractionals + 2$ Bit = 16 Bit. Wobei jeweils ein Bit für das Vorzeichen und ein Bit zur Speicherung einer positiven Eins bereit gestellt werden, die im Falle eines Dividenden von 1 erreicht wird.
- **Breite der Nachkommastellen des Divisionsergebnisses** F**:** $divider\_fractionals = 8$ Bit. Der Dividierer löst 8 Nachkommastellen der Division.

Für einen Dividierer, der vorzeichenbehaftete Eingangswerte verarbeitet und mehr als einen Takt zur Berechnung eines Divisionsergebnisses zur Verfügung hat, ergibt sich eine Latenz von $M + F + 5 = 16 + 8 + 5 = 29$ Takten [32] bei einer vierfachen Pixelfrequenz f_{54} = 54 MHz, was eine Latenz von 8 Takten der Pixelfrequenz $f_{13.5}$ = 13,5 MHz entspricht (vgl. Abb. 30). Zusätzlich zur Latenz des Dividierers tritt noch eine Latenz durch das Downsampling der überabgetasteten Divisionsergebnisse von einem Takt auf (vgl. Abb. 30).

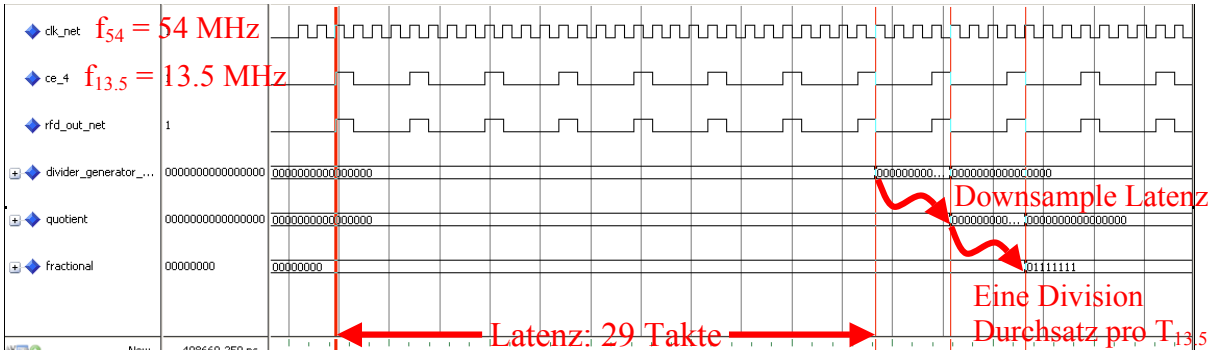

Abb. 30: *VHDL Simulation der projektiven Transformation; Latenz und Durchsatz des Dividierers*

Das Simulationsergebnis wird aus dem System Generator in den Matlab Workspace gesichert und steht dort für eine Ergebnisauswertung zur Verfügung. Wird die Menge der projektiv transformierten Pixel in einem Bild dargestellt, beinhaltet dieses Störungen, da nicht zu jedem Pixel des Zielbildes ein projektiv transformiertes Pixel berechnet wurde (vgl. Abb. 32) und das Bild an diesen Stellen schwarz bleibt. Diese Störungen haben auf die Hough-Transformation keinen Einfluss, da diese keine kompletten Bilder verarbeitet, sondern die im Pixelstrom transformierten Pixel für die Berechnung einsetzt (vgl. Abb. 15). Eine Analyse des längsten Laufzeitpfades mit dem Place & Route Implementierungsschritt ergibt eine Maximalfrequenz f_{max} von ca. 76 MHz für den mit f_{54} übertakteten Bereich.

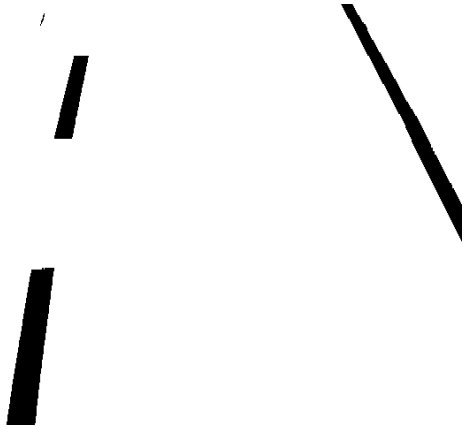

Abb. 31: *Perspektivisch verzeichnete Aufnahme der Fahrspur mit der auf dem SoC-Fahrzeug angebrachten Sony FCB-PV10*

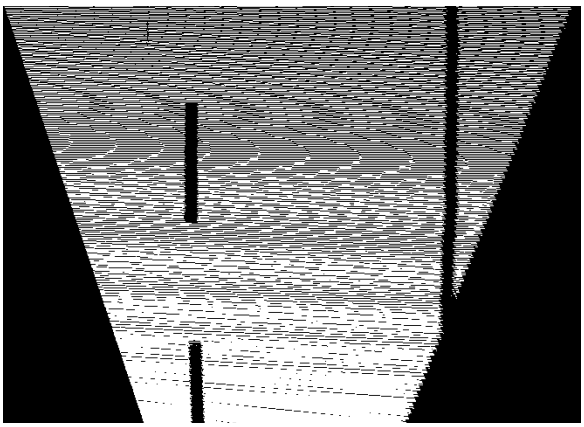

Abb. 32: *Die Darstellung des Ergebnisses der projektiven Transformation beinhaltet schwarze Störungen, die aufgrund einer schwarzen Initialisierung des Zielbildes entstehen.*

6 Fahrspurapproximation mit der Hough-Transformation

Die Hough-Transformation beschreibt ein robustes Verfahren, mit dem parametrisierbare geometrische Referenzstrukturen in vorverarbeiteten Bildern (vgl. Kap. 4) detektiert werden, die mit Störungen behaftet oder teilweise verdeckt sein können. Als geometrische Referenzstruktur für die Fahrspur dient eine Gerade, die in einem zweidimensionalen Bild durch eine Maximumsuche im zweidimensionalen Parameterraum aufgefunden wird. Eine Vorverarbeitung des Eingangspixelstroms mit einem Kantenfilter und anschließender Binarisierung unterteilt das Bild in Vorder- und Hintergrundpixel, wobei die Hough-Transformation auf der Menge der Vordergrundpixel arbeitet.

Folgender Abschnitt stellt die mathematischen Grundlagen der Hough-Transformation vor und erläutert die Dimensionierung des als Houghraum bezeichneten Parameterraums. Anschließend wird eine Matlab Implementierung der dargelegten Konzepte vorgestellt, die eine Softwarerealisierung zu Verifikationszwecken darstellt.

6.1 Mathematische Grundlagen der Hough-Transformation

Die Strukturidentifikation einer Geraden unter Verwendung der Hough-Transformation beinhaltet drei Verfahrensschritte [10]:

1. Parametrisieren der Geraden
2. akkumulierende Abbildung der Vordergrundpixel
3. Maximumsuche im Houghraum als Auswertung der Hough-Transformation

Eine Gerade wird durch eine Geradengleichung eindeutig beschrieben:

$$y = mx + b \tag{29}$$

Die Steigung m und der Achsenabschnitt b charakterisieren die Gerade und die Punkte, die auf dieser liegen und spannen einen (m, b)-Parameterraum auf. Die Hough-Transformation adaptiert diesen Zusammenhang und ermittelt zu jedem Punkt der Menge der Vordergrundpixel (x, y) die Parameterwerte m und b. Die Steigung m ist unendlich, wenn die Gerade g parallel zur y-Achse verläuft, was einem Fahrspurverlauf parallel zur Fahrzeuglängsachse entspricht. Ein (m, b)-Parameterraum ist nicht linear quantisierbar, wenn eine Steigung $m \rightarrow \infty$ erreicht wird, was einem Standardszenario des Fahrspurverlaufes entspricht [12].

$$m = \frac{\Delta y}{\Delta x} = \frac{\Delta y}{x - x} = \frac{\Delta y}{0} = \infty \tag{30}$$

Eine Geradendarstellung, die keine Singularitäten für Geraden aufweist, die parallel zur y-Achse verlaufen und eine lineare Quantisierung der Geradenparameter erlaubt [4], ist die Hessesche Normalform, in der Geraden durch die Ursprungslotlänge r und den Winkel ϕ zwischen Ursprungslot und x-Achse beschrieben werden (vgl. Abb. 33):

$$r = x \cdot cos(\phi) + y \cdot sin(\phi) \tag{31}$$

Eine Gerade im (x, y)-Koordinatensystem entspricht einem Punkt im (r, ϕ)-Parameterraum (vgl. Abb. 33), der als Houghraum bezeichnet wird. Die Hough-Transformation eines Vordergrundpixels (x_i, y_i) wird durch die Berechnung der Ursprungslotlänge r_k für alle ϕ_k durchgeführt, wobei $\phi_k \in \left[-90°, 180°\right]$.

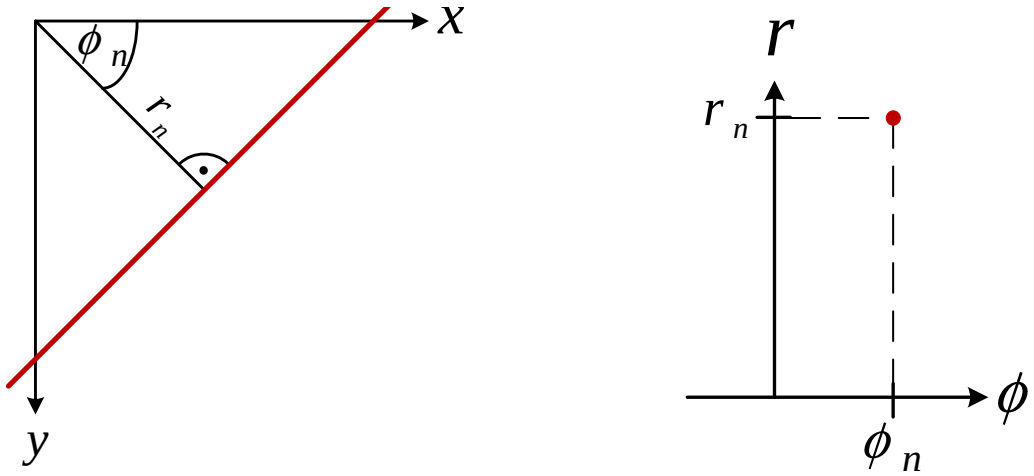

Abb. 33: *Eine Gerade im x-y-Raum entspricht einem Punkt im Houghraum*

Werden die Parameter aller Geraden die durch einen Punkt A verlaufen in den Houghraum aufgetragen, ergibt sich eine sinoidale Kurve (vgl. Abb. 35). Die Parameter eines zweiten Punktes B erzeugt ebenfalls eine derartige Kurve, wobei die Punkte A und B eine Gerade g gemeinsam haben und sich in einem Punkt im Houghraum schneiden, der die Parameter r und ϕ der Gerade identifiziert (vgl. Abb. 34 und 35). Die Gerade g, auf der die meisten Vordergrundpixel liegen, wird durch ein Schnittpunkt-Maximum im Houghraum identifiziert [10].

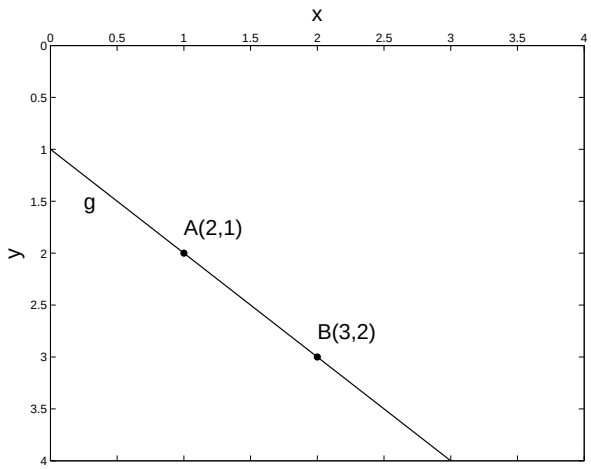

Abb. 34: *Punkte A und B im x-y-Raum haben eine Gerade g gemeinsam*

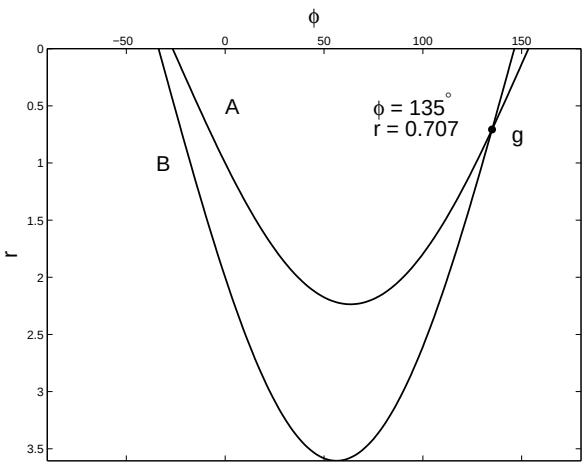

Abb. 35: *Die Gerade g wird im Houghraum durch Schnittpunkte von A und B repräsentiert*

6.2 Dimensionierung des Houghraumes

Die Hough-Transformation eines Vordergrundpixels (x_i, y_i) wird durch Berechnung der Länge r_k des Urspungslots für alle diskretisierten ϕ_k bestimmt. Gültige Winkel ϕ_k stammen dabei aus dem Intervall $\phi_k \in \left[-90°, 180°\right]$. Geraden mit Ursprungsloten außerhalb des Intervalls liegen nicht im Bild (vgl. Abb. 36).

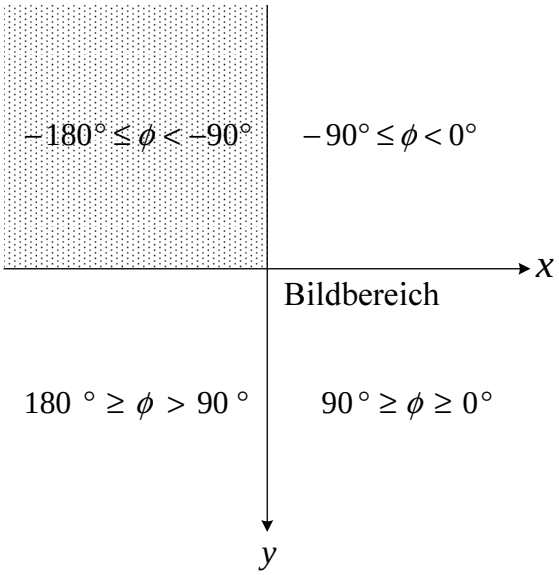

Abb. 36: *Gültigkeitsbereiche des Winkels ϕ; Geraden mit einem Winkel $-180° \leq \phi < -90°$ liegen außerhalb des Bildes*

Durch Diskretisierung des Houghraumes wird dieser in ein zweidimensionales Feld überführt, wobei jede Zelle des Feldes einer Geraden im (x, y)-Raum entspricht [10]. Damit wird er in einem digitalen System handhabbar, da wertkontinuierliche Berechnungsergebnisse auf diskrete Strukturen abgebildet werden. Im diskretisierten Houghraum entspricht eine Zelle (r_n, ϕ_m) einem balkenförmigen Fenster mit:

- unendlicher Länge l
- einer Breite Δr
- einer Ursprungslotlänge r_n
- einem Winkel ϕ_m zwischen Ursprungslot und x-Achse [12]

Jede Zelle des diskretisierten Houghraumes entspricht einem Akkumulator, der inkrementiert wird, wenn die durch die Hough-Transformation eines Punktes bestimmte Ursprungslotlänge r in diesem Feld liegt. Im diskretisierten Houghraum beschränkt sich die Suche der Geraden, auf der die meisten Punkte liegen, auf die Suche des Akkumulators mit dem größten Wert.

Der Winkelbereich, aus dem Winkel zwischen Ursprungslot und x-Achse stammen, wird in dieser Arbeit auf $\phi \in \left[-45°, 45°\right]$ festgelegt, da der Fahrspurverlauf relativ zur Fahrzeuglängsachse dieses Intervall im Falle einer aktiven Spurführung nicht verlässt. Alle Geraden innerhalb eines Bildes werden mit einem Winkel $\phi_k \in \left[-90°, 180°\right]$ und einer Ursprungslotlänge $r \geq 0$ dargestellt (vgl. Abb. 37). Eine Dimensionierung des Winkelbereichs, die nicht alle $\phi_k \in \left[-90°, 180°\right]$ beinhaltet, schließt eine Geradendarstellung in bestimmten Bildbereichen aus. In Abbildung 38 ist eine Gerade dargestellt, dessen Ursprungslot einen Winkel $\phi = -45°$ zur x-Achse besitzt. Unter Ausschluss von negativen Ursprungslotlängen r und einem Winkel $\phi \in \left[-45°, 45°\right]$ lassen sich Geraden im schraffierten Bereich nicht darstellen. Wird der Bereich der akzeptierten Urpsungslotlängen erweitert, wird eine größere Menge an Geraden darstellbar

(vgl. Abb. 39). Eine Gerade g mit negativer Ursprungslotlänge r entspricht einer äquivalenten Geraden g mit positiver Ursprungslotlänge r und einem um $180°$ erhöhten Winkel ϕ (vgl. Abb. 40 und 41). Geraden mit negativer Ursprungslotlänge r und positivem Winkel ϕ liegen außerhalb des Bildbereichs.

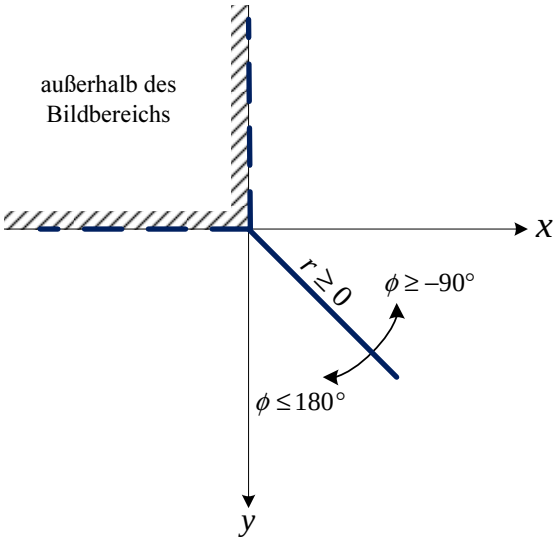

Abb. 37: *Alle Geraden im Bild können mit einem Winkel $-90° \leq \phi \leq 180°$ und positiver Ursprungslotlänge r dargestellt werden*

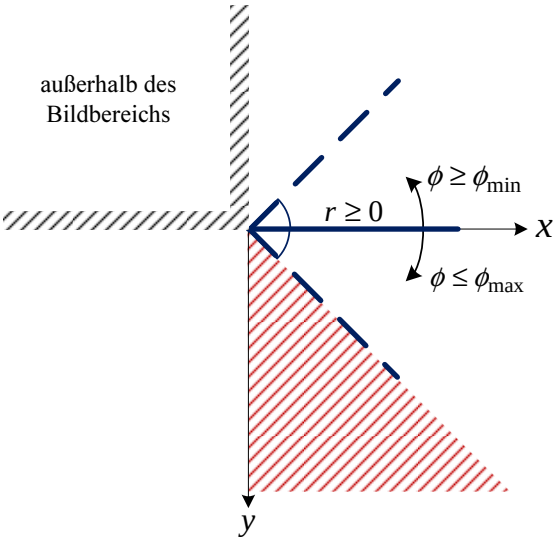

Abb. 38: *Bei einem Winkel $\phi \in [-45°, 45°]$ werden unter Ausschluss negativer Ursprungslotlängen r Geraden im schraffierten Bereich nicht darstellbar*

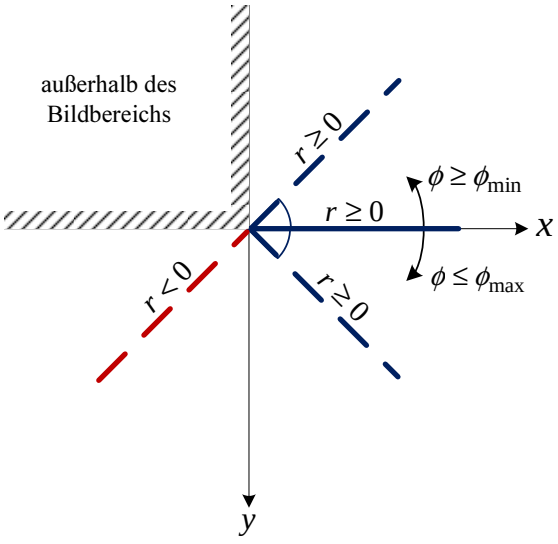

Abb. 39: *Werden negative Ursprungslotlängen r zugelassen, erweitert sich der Bereich der darstellbaren Geraden bei eingeschränktem Winkelbereich ϕ*

Der mögliche Wertebereich der positiven Ursprungslotlänge r_{max} ergibt sich aus dem Cosinus des Winkels $\phi_{max} - tan\left(\frac{y}{x}\right)$ zwischen Bilddiagonale und maximalem Winkel ϕ_{max} multipliziert mit der Länge der Bilddiagonalen. Der Winkel zwischen y-Achse und dem Ursprungslot auf die größte negative Gerade, die im Bild enthalten sein kann, ist $\frac{\pi}{2} + \phi_{min}$. Der mögliche Wertebereich der negativen Lotlänge r_{min} ergibt sich aus dem Cosinus des Winkels $\frac{\pi}{2} + \phi_{min}$

multipliziert mit der Seitenlänge y. Auf diese Weise lassen sich alle Geraden, die im Bild im Winkelbereich von ϕ_{min} bis ϕ_{max} enthalten sein können, sowohl mit positiver als auch mit negativer Ursprungslotlänge r darstellen. Durch ein Erweitern der Ursprungslotlänge auf negative Werte geht die Eindeutigkeit der Geradendarstellung verloren, wenn sowohl der Winkel ϕ_i als auch der Winkel $\phi_i - 180°$ im Winkelbereich enthalten sind. Eine Gerade g wird durch zwei Hessesche Normalformen beschrieben (vgl. Abb. 40 und Abb. 41):

- $r = x \cdot cos(\phi_i) + y \cdot sin(\phi_i)$
- $-r = x \cdot cos(\phi_i - 180°) + y \cdot sin(\phi_i - 180°)$

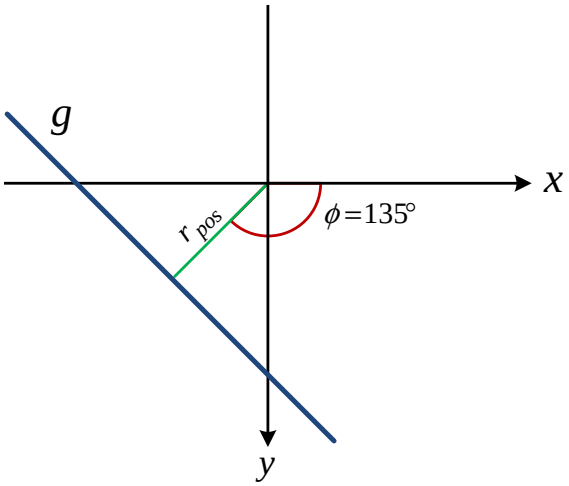

Abb. 40: *Gerade g repräsentiert mit positiver Normalenvektorlänge r_{pos} und Winkel $\phi = 135°$*

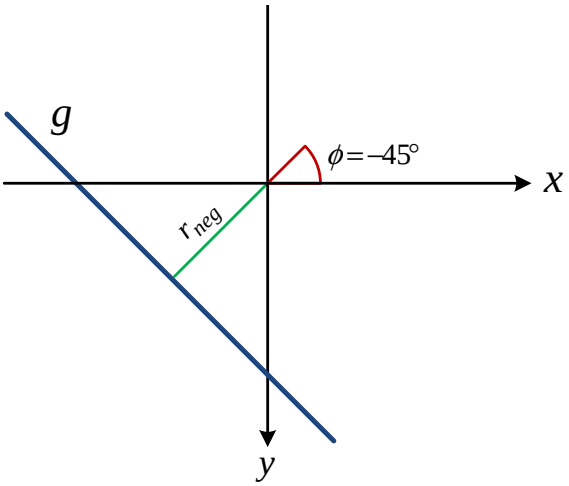

Abb. 41: *Gerade g repräsentiert mit negativer Normalenvektorlänge r_{neg} und Winkel $\phi = -45°$*

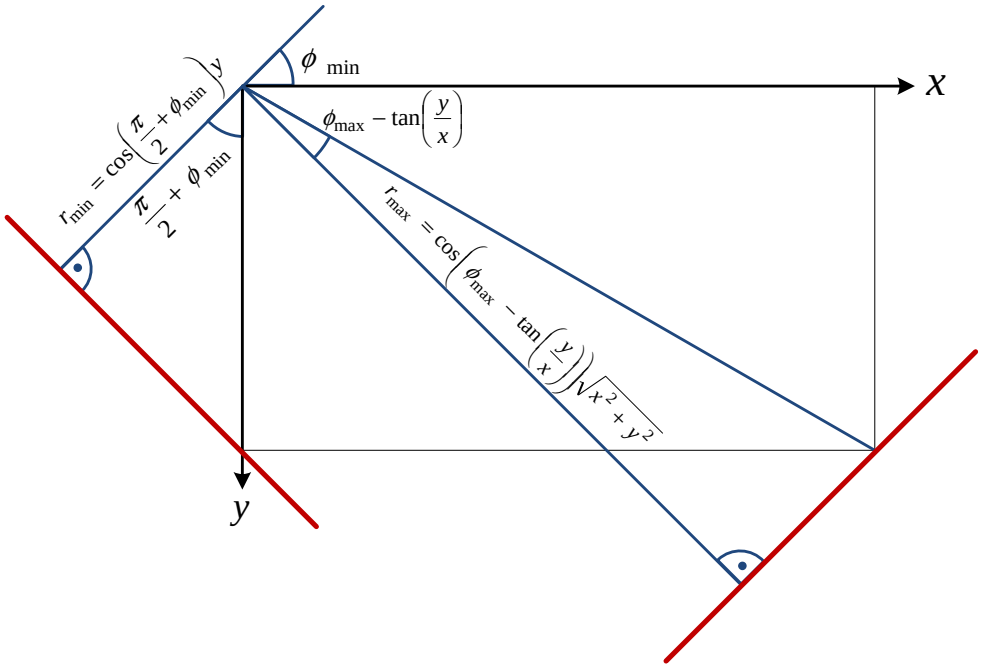

Abb. 42: *Berechnung der Lotlängen r_{max} und r_{min} unter Einfluss der Bildgröße und der zugelassenen Maximalwinkel ϕ_{max} und ϕ_{min}*

6.3 Realisierung der Hough-Transformation in Matlab

Eine Analyse der Hough-Transformation wird in Matlab durchgeführt und bietet eine Verifikationsmöglichkeit für Entwicklungsergebnisse mit dem System Generator. Für die Dimensionierung des Houghraumes gilt (vgl. Kap. 6.2):

- $\phi \in \left[-45°, 45°\right]$
  wobei $\Delta\phi = 5°$
- $r \in \left[-r_{min}, r_{max}\right]$
  wobei $r_{min} = cos\big(90° + \phi_{min}\big) \cdot y$ und $r_{max} = cos\big(\phi_{max} - tan(\frac{y}{x})\big) \cdot \sqrt{x^2 + y^2}$.

Für jedes Vordergrundpixel (x, y) wird die Hough-Transformation mit der Hesseschen Normalform $r = x \cdot cos(\phi_k) + y \cdot sin(\phi_k)$ durchgeführt, wobei $\phi_k \in \left[-45°, 45°\right]$. Der Hardwarerealisierung entsprechend werden entstandene Nachkommastellen abgeschnitten und keine Rundung durchgeführt. Die so entstehende Ganzzahl lässt sich mit einem Adressoffset als Teiladresse für den Houghraum verwenden. Der Winkel ϕ_k, der definitionsgemäß auch negativ sein kann, wird für eine Matlab Simulation in eine positive Adresse gewandelt, so dass eine Abbildung von Winkel auf Adresse entsteht.

$$\begin{aligned} \phi_{min} &\rightarrow Adresse(0) \\ &\ldots \\ \phi_{max} &\rightarrow Adresse(n-1) \end{aligned}$$

Die Adressberechnung für den Winkel ϕ_k erfordert (vgl. Gl. 32):

- eine Subtraktion des Minimalwinkels ϕ_{min} vom aktuellen Winkel ϕ_k, um eine Korrektur von negativen Winkeln in einen positiven Bereich durchzuführen
- eine Division mit $\Delta\phi$, um eine Relation zur Schrittweite $\Delta\phi$ herzustellen
- für eine Matlab Realisierung eine Addition von 1, um eine Adressierung der 0 zu verhindern, da Indizes in Matlab mit 1 beginnen

$$\phi_{addr} = \frac{\phi_k - \phi_{min}}{\Delta\phi} + 1 \tag{32}$$

Eine Korrektur der Ursprungslotlänge r in positive Werte, um eine Indizierung zu ermöglichen, erfolgt durch Addition der kleinsten Adresse r_{min}.

$$r_{addr} = r + r_{min} \tag{33}$$

Wurden alle Vordergrundpixel des Bildes transformiert, lässt sich das Ergebnis durch Maximumsuche im Houghraum und eine Rücktransformation der Matrixadresse in Lotänge r und Winkel ϕ ermitteln:

$$\phi = (\phi - 1) \cdot \Delta\phi + \phi_{min} \tag{34}$$

$$r = r - r_{min} \tag{35}$$

```
I = imread(...);

dphi = 5;           % delta phi in Grad
phi = -45:dphi:45;  % phi Intervall in dphi Schritten
[size_y size_x] = size(I);

% rmax und rmin in ganzzahligen Schritten
rmax = ceil(cos(max(phi).*pi/180-tan(size_y/size_x)).*sqrt(size_x.*size_x+size_y.*size_y));
rmin = ceil(cos((pi/2)+min(phi).*pi/180).*size_y);

houghraum = zeros(rmax+rmin, length(phi));  % Houghraum aufspannen

for y=1:size_y
    for x=1:size_x
        if(I(y,x)>0)
            for phi_i=phi
                r = fix(x.*cosd(phi_i)+y.*sind(phi_i)); % r berechen und Nachkommastellen abschneiden
                phi_addr = ((phi_i-min(phi))/dphi)+1;   % Adresse im houghraum ermitteln
                r_addr = r+rmin;
                houghraum(r_addr,phi_addr) = houghraum(r_addr,phi_addr)+1;
            end;
        end;
    end;
end;

[ht_r,ht_phi] = find(houghraum==max(houghraum(:))); % Maximum ermitteln
% Maximum in r und phi zurückrechnen
ht_r = ht_r - rmin;
ht_phi = (ht_phi-1)*dphi+min(phi);
```

Listing 1: *Matlab Realisierung der Hough-Transformation*

7 Modellierung der Hough-Transformation mit dem System Generator

Bei der Modellierung der Hough-Transformation kommen die in Kapitel 6 vorgestellten mathematischen Konzepte der Berechnung und der Dimensionierung des Houghraumes zur Anwendung. Es werden zwei Methoden zum Entwurf des Pixelstromdatenpfades vorgestellt, wobei die Wahl der Zielplattform entscheidenden Einfluss auf den Entwurf hat, da die Ressourcenverfügbarkeit und die erreichbare Maximalfrequenz zu berücksichtigen sind. Der Datenpfad wird in einer Pipelinestruktur entworfen, in dem übertaktete Pipelinestufen mit Multizyklus-eigenschaften eingesetzt werden, um ein Ressource-Sharing zu nutzen und Berechnungen in mehreren Zyklen durchzuführen (vgl. Abb. 44). Eine reine Multizyklusrealisierung des Datenpfades macht eine Systemfrequenz erforderlich, die sehr viel höher als die Pixelfrequenz ist und von einem Spartan 3E FPGA nicht zur Verfügung gestellt wird [38] (vgl. Abb. 4, Kap. 2.1).

7.1 Abbildung des Houghraumes auf eine Hardwarestruktur

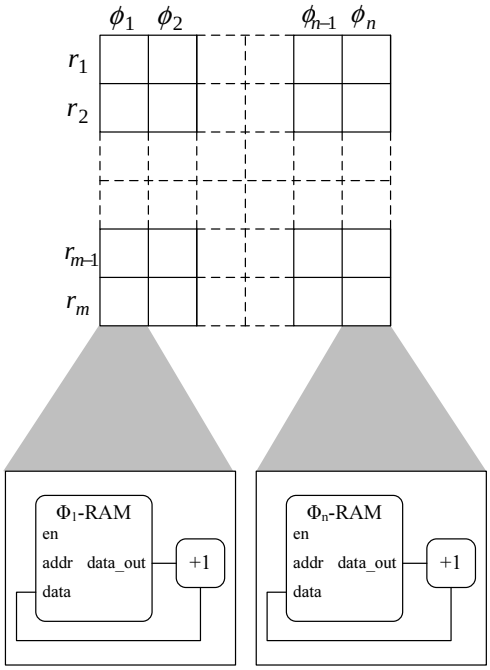

Abb. 43: *Zweidimensionaler Houghraum mit n Spalten und m Zeilen; die Hardwaremodellierung erfolgt mit n RAM-Blöcken (ϕ_n-RAM), die in einer übertakteten Pipelinestufe eine Akkumulatorfunktion erfüllen*

Durch Diskretisierung geht der Houghraum in ein zweidimensionales Feld über (vgl. Kap. 6.2). In einem Hardwaremodell stehen derartige Felder nicht zur Verfügung. Die Hardwaremodellierung eines Hoguhraumes mit n Spalten und m Zeilen erfolgt in Hardware (vgl. Abb. 43):

- RAM-basiert, unter Einsatz von n RAM-Blöcken,
- mit m RAM-Zellen pro RAM-Block

Eine Akkumulatorfunktion im Pixelstrom wird mit einer übertakteten Pipelinestufe realisiert, die eine Frequenz f_{27} = 27 MHz erfordert:

- Adressieren der zu inkrementierenden RAM-Zelle,
- Zurückschreiben des inkrementierten Wertes

Wird eine negative Ursprungslotlänge r zugelassen (vgl. Kap. 6.2), erhöht sich die Tiefe der RAM-Blöcke, da zusätzliche Ursprungslotlängen r zusätzliche Adressen erfordern und so eine Erhöhung der m RAM-Zellen erfolgt. Stehen keine RAM-Strukturen zur Verfügung, ist eine Abbildung des Houghraumes auf Arrays von Vektoren durchführbar, in dem jeder Vektor einer Adresse entspricht, die gelesen und inkrementiert wird [2].

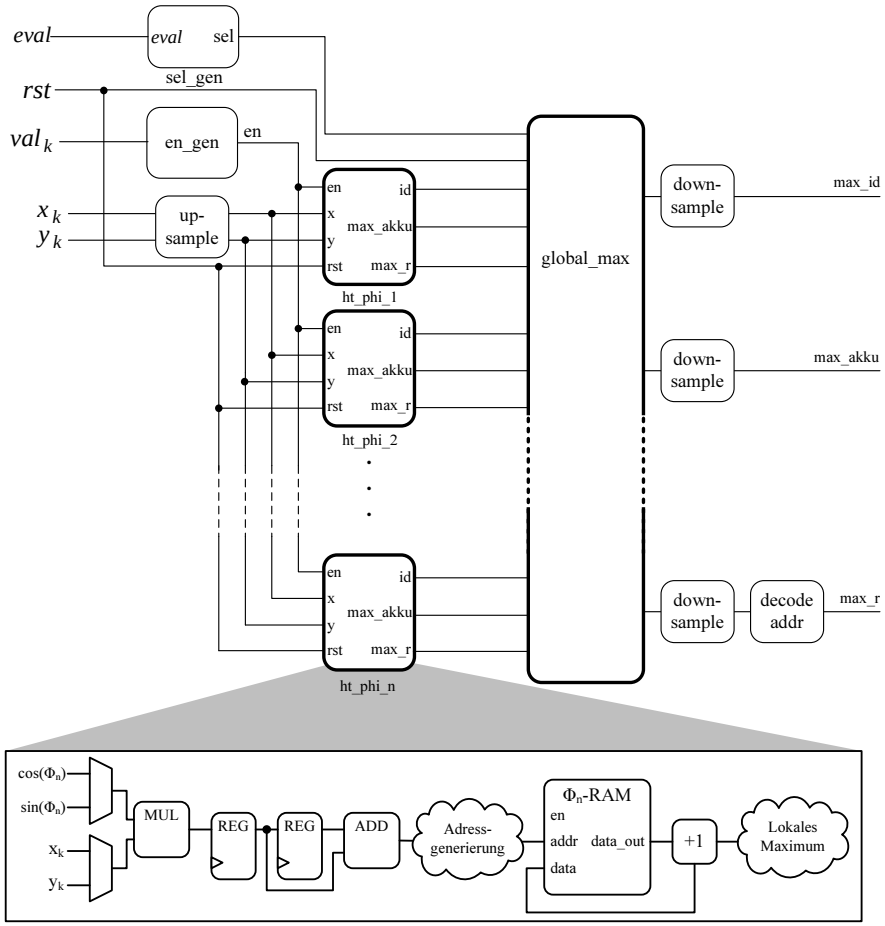

Abb. 44: *Pipeliningdatenpfad mit übertakteten Pipelinestufen, wodurch ein Ressourcesharing realisierbar ist und Multizyklusoperationen im Pixelstrom durchgeführt werden; durch parallele Anordnung von n Houghtransformatoren ht_phi_n werden n Winkelstufen nebenläufig berechnet (vgl. Kap. 6.2)*

Zur Hough-Transformation werden Pixelkoordinaten (x_k, y_k) und zugehörige binarisierte Grauwerte val_k in einem Pixelstromdatenpfad verarbeitet, der alle Arithmetikelemente, Speicher und Multiplexer beinhaltet, die für eine algorithmische Umsetzung der Hough-Transformation eingesetzt werden [27]. Diese kombinatorischen und getakteten Schaltungselemente werden durch Steuersignale kontrolliert. Der Datenpfad wird in einer Pipelinestruktur mit übertakteten Pipelinestufen realisiert (vgl. Abb. 44), in der alle Rechenschritte parallel in einer Taktperiode bearbeitet werden, wobei Zwischenergebnisse in separaten Pipeline-Registern gespeichert werden [27] (vgl. Kap. 2.1). Für eine Koordinatentransformation wird ein Multiplizierer mit gemultiplexten Operanden eingesetzt, wodurch eine Ressourcenersparnis erreicht wird, da Arithmetikressourcen in aufeinanderfolgenden Takten für Berechnungen mit unterschiedlichen Operanden eingesetzt werden (vgl. Kap. 2.1).

Der Einsatz von parallelen Arithmetikressourcen innerhalb der Pipelinestufen erfordert eine niedrigere Taktfrequenz als ein Multizyklusdatenpfad mit einer großen Anzahl von getakte-

ten Rechenschritten innerhalb des Aktualisierungsintervals der Pixel. Steht eine Plattform zur Verfügung, die hohe Taktraten bereitstellt, kann ein Entwurf des Datenpfades als Multizyklusdatenpfad erfolgen (vgl. Abb. 4), bei dem Arithmetikelemente mehrfach verwendet werden und so eine Ressourcenersparnis auf Kosten einer hohen Taktfrequenz für die Abarbeitung aller Berchnungsschritte innerhalb der Pixelperiode erreicht wird.

7.2 Pixelstromdatenpfad mit parallelen Houghtransformatoren

Das Blackbox Symbol des Moduls zur Berechnung der Hough-Transformation umfasst folgende Eingangssignale (vgl. Abb. 45), die aus einem Modul zur Überprüfung auf Zugehörigkeit zur *Region of Interest* angelegt werden (vgl. Anhang B und Kap. 4.3), was eine Fokussierung der Bearbeitung auf relevante Bildausschnitte sicherstellt:

- **clk** Die mit $f_{13.5}$ = 13,5 MHz getaktete Clock wird auf f_{27} = 27 MHz übertaktet, da der Zugriff auf das RAM mit einem Inkrementiervorgang in zwei Taktzyklen erfolgt. Bedingt durch ein Operandenmultiplexen wird eine geringere Anzahl an Multiplizierern eingesetzt. Ein Multiplizierer berechnet jeweils das Produkt zweier Multiplikationen mit unterschiedlichen Faktoren.
- **rst** Das System wird nach jeder abgeschlossenen Hough-Transformation in den Initialzustand gesetzt, um die RAM-Blöcke mit Nullen zu beschreiben und die Register zurückzusetzen. Das *reset*-Signal muss einen f_{27} = 27 MHz Takt für jede Adresse des RAM-Blocks anliegen, da jede Speicherzelle mit Nullen beschrieben wird.
- **eval** Das Ende einer Bearbeitung und der Start einer Maximumauswertung wird über einen *high*-Pegel des *eval* Signals signalisiert.
- **Pixelinformationen** Die projektiv transformierte Pixelkoordinate (x_k, y_k) und der zugehörige binarisierte Grauwert val_k werden zur koordinatenbasierten Berechnung angelegt. Ein Vordergrundpixel (vgl. Kap. 6) erzeugt ein *enable*-Signal, so dass das Ergebnis in die jeweiligen RAM-Blöcke übernommen wird; im Falle eines Hintergrundpixels ist das System aktiv, jedoch werden keine Ergebnisse erzeugt die übernommen werden, da eine Steuersignalgenerierung auf Basis der Grauwertinformation val_k erzeugt wird. Liegt ein Pixel nach der projektiven Transformation nicht in der *Region of Interest*, wird durch einen *low*-Pegel der Grauwertinformation val_k das Erzeugen von *enable*-Signalen verhindert (vgl. Anhang B).

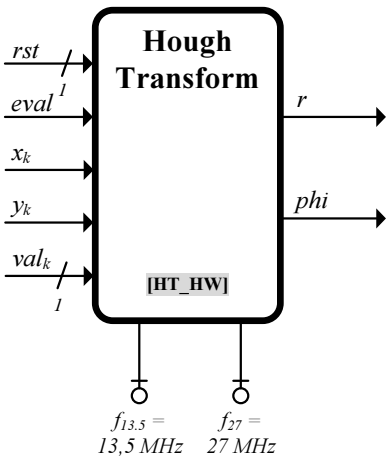

Abb. 45: *Blackbox des Hardwaremoduls zur Hough-Transformation mit zwei Takteingängen für die Pixelfrequenz $f_{13.5}$ und die übertaktete Frequenz f_{27}*

Am Ausgang des Moduls liegen nach abgeschlossener Transformation die folgenden Signale an (vgl. Kap. 6):

- Die Ursprungslotlänge **r** der ermittelten Gerade
- Der Winkel **phi** zwischen x-Achse und Ursprungslot

7.2.1 Write-Enable Generierung für die Houghtransformatoren

Das Subsystem en_gen erzeugt Steuersignale aus den anliegenden binarisierten Grauwertinformationen val_k (vgl. Abb. 46) unter Verwendung des *Clock Enable Probe*, das das Hardware Clock Freigabeschema für die eingehende Signalsamplerate darstellt (vgl. Kap. 2.2), die für ein Operandenmultiplexen und als *write enable*-Signal für die RAM-Blöcke der Houghtransformatoren genutzt werden. Binarisierte Grauwertinformationen val_k liegen für $T_{13.5}$-Perioden an (vgl. Abb. 47), da val_k mit einer Frequenz $f_{13.5}$ = 13,5 MHz abgetastet wird. Das Clock Enable Signal, das aus der binarisierten Grauwertinformation val_k extrahiert wird, besitzt einen *high*-Pegel für eine halbe $T_{13.5}$-Periode, was einer T_{27}-Periode entspricht. Eine Schreibfreigabe wird auf diese Weise nur bei Vordergrundpixeln für die Dauer einer T_{27}-Periode generiert, wodurch eine einmalige Schreibfreigabe der RAM-Blöcke erreicht wird.

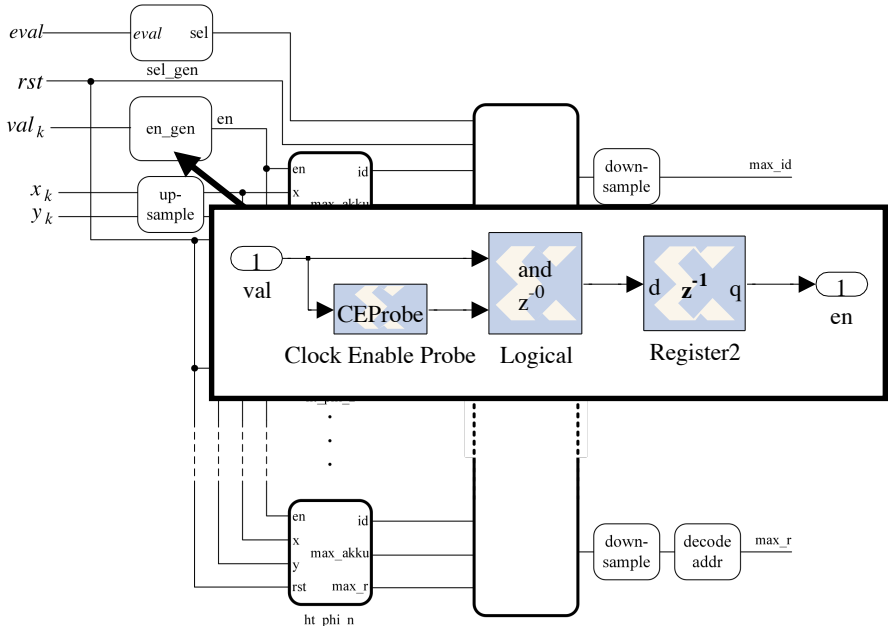

Abb. 46: *Anliegende Vordergrundpixel erzeugen enable-Signale für die Dauer einer T_{27}-Periode, unter Verwendung des Clock Enable Probes*

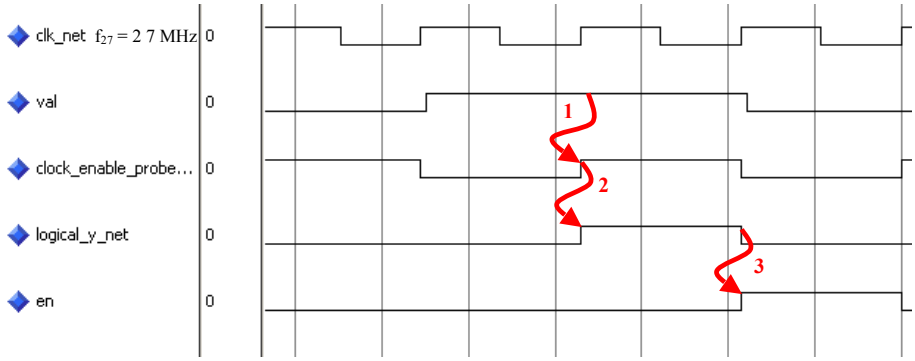

Abb. 47: *VHDL Simulation der Enablesignalgenerierung*

Nr.	Beschreibung
1 & 2	Die *high*-Pegel der binarisierten Grauwertinformation val und der extrahierten Clockfreigabe mittels Clock Enable Probe, erzeugen einen *high*-Pegel am Ausgang des UND-Gatters.
3	Zum Timingausgleich erscheint das *enable*-Signal einen Takt verzögert am Ausgang des Subsystems.

Tabelle 7: *Erläuterung des Timings der VHDL-Simulation in Abb. 47*

7.2.2 Berechnung der Ursprungslotlänge r_i für ein festen Winkel ϕ_i

Die Hough-Transformation der perspektivisch korrigierten Pixelkoordinaten (x_k, y_k) erfolgt in dem Houghtransformator *ht_hw_phi_i* für einen festen Winkel ϕ_i (vgl. Abb. 48). Die Pixelkoordinaten (x_k, y_k) werden mit f_{27} = 27 MHz überabgetastet, wodurch ein Ressource-Sharing über den Einsatz von Operandenmultiplexing erreicht wird:

- Innerhalb einer $T_{13.5}$-Periode werden zwei Multiplikationen mit der übertakteten Frequenz f_{27} = 27 MHz durchgeführt, wodurch ein Multiplizierer für beide Multiplikationen verwendet wird.
- Das durch ein Vordergrundpixel erzeugte *enable*-Signal (vgl. Kap. 7.2.1) schaltet die Multiplexer zur Operandenselektion (vgl. Abb. 49), wodurch die Hessesche Normalform $r_i = x_k \cdot cos(\phi_i) + y_k \cdot sin(\phi_i)$ der Pixelkoordinaten (x_k, y_k) berechnet wird.

Die Teilergebnisse $x_k \cdot cos(\phi_i)$ und $y_k \cdot sin(\phi_i)$ stehen in zwei Registern zur Addition bereit, wobei das Ergebnis der ersten Multiplikation $x_k \cdot cos(\phi_i)$ mit der Berechnung des zweiten Multiplikationsergebnisses $y_k \cdot sin(\phi_i)$ eine Registerstufe weitergeschoben wird (vgl. Abb. 48).

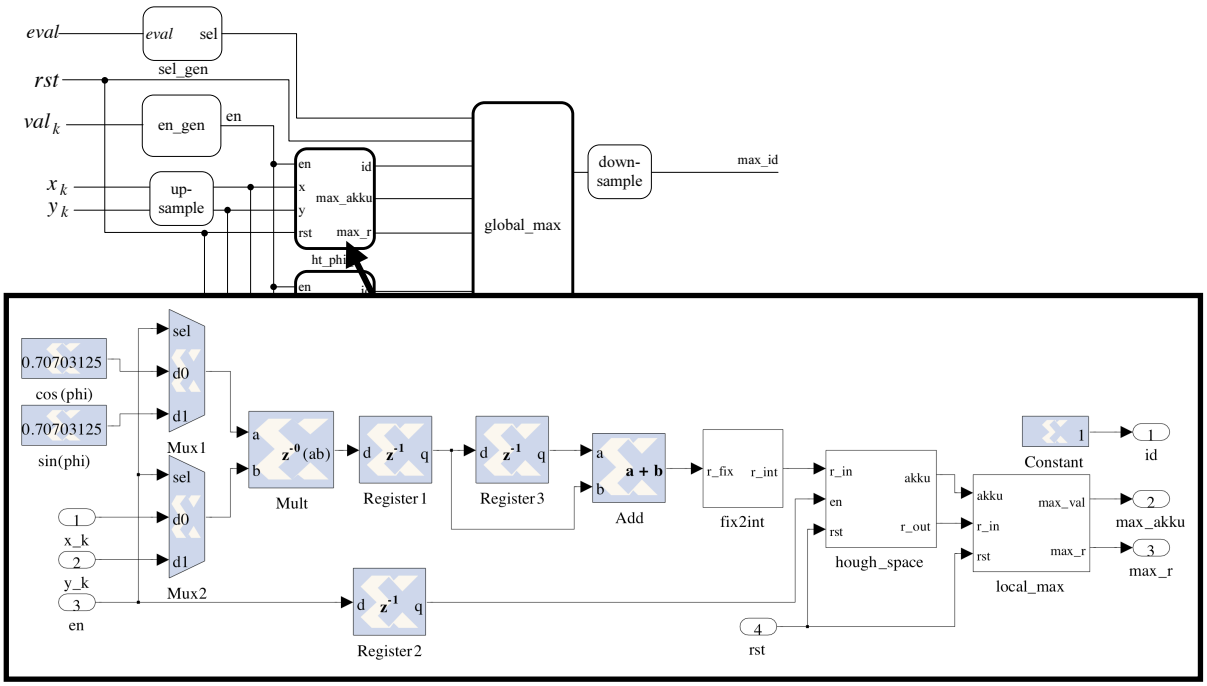

Abb. 48: *Aufbau eines Houghtransformators ht_hw_phi_i zur Berechnung der Hough-Transformation für einen festen Winkel ϕ_i*

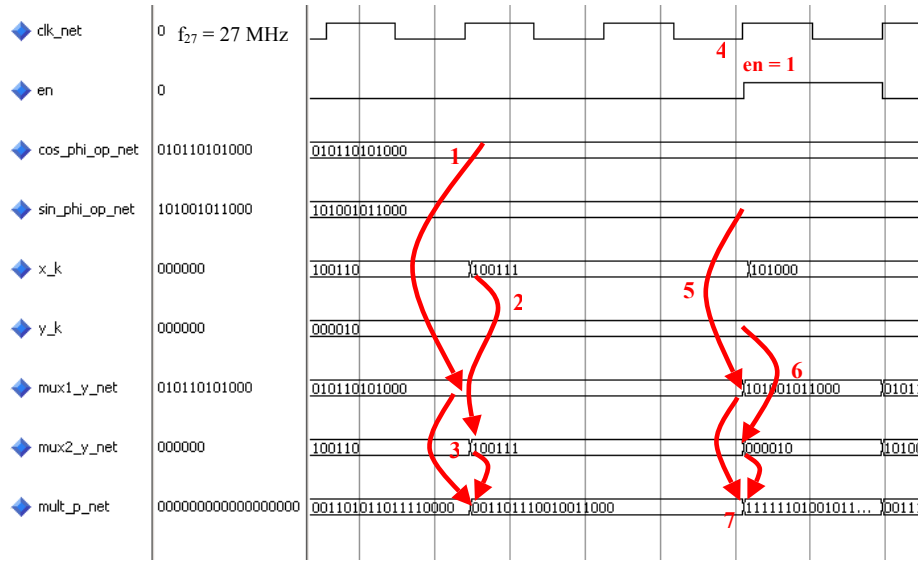

Abb. 49: *Durch Upsampling werden die Multiplikationen $x_k \cdot cos(\phi_i)$ und $y_k \cdot sin(\phi_i)$ mit einem Multiplizierer in zwei T_{27}-Zyklen realisiert. Das enable-Signal steuert das Operandenmultiplexen.*

Nr.	Beschreibung
1 & 2	Der *low*-Pegel des *enable*-Signals, schaltet den Cosinus des Winkels ϕ_i und die Pixelkoordinate x_k auf die Ausgänge der Multiplexer.
3	Das Ergebnis der binären Multiplikation $x_k \cdot cos(\phi_i)$ liegt am Multiplizierausgang an.
4	Das *enable*-Signal nimmt einen *high*-Pegel an.
5 & 6	Da $en = 1$, schalten die Multiplexer den Sinus des Winkels ϕ_i und die Pixelkoordinate y_k.
7	Das Ergebnis der binären Multiplikation $y_k \cdot sin(\phi_i)$ liegt am Multiplizierausgang an.

Tabelle 8: *Erläuterung des Timings der VHDL-Simulation in Abb. 49*

7.2.3 Wandlung der Ursprungslotlänge r_i in eine RAM-Adresse

Das Subsystem *fix2int* (vgl. Abb. 50) wandelt ein Fixed-Point Ergebnis der Berechnung $r_i = x_k \cdot cos(\phi_i) + y_k \cdot sin(\phi_i)$ (vgl. Kap. 7.2.2) in eine positive Ganzzahl und führt eine Intervalleinteilung anhand eines Parameters zur Bestimmung der Intervallgröße durch, durch den niederwertige Nachkommastellen des anliegenden Eingangswertes r_fix abgeschnitten werden. Ein $m + n$-Bit breiter *signed* Vektor mit n Nachkommastellen wird so in einen m Bit breiten *signed* Vektor ohne Nachkommastellen gewandelt (vgl. Kap. 2.2).

Die Hough-Akkumulatoren werden als *Single Port RAM*-Blöcke realisiert, in denen negative Ursprungslotlängen r_i nicht als Adresse genutzt werden, weshalb der zu einer Ganzzahl gewandelte Eingsangwert r_i um einen Offset korrigiert wird, welcher der maximalen negativen Ursprungslotlänge r_{min} entspricht. Die Intervallgröße wird über einen Matlab Parameter bestimmt, der die Anzahl der abzuschneidenen Bits angibt und so die Anzahl der Werte angibt, die in ein gemeinsames Intervall akkumuliert werden. Eine Zusammenfassung in Intervalle bewirkt eine Verkleinerung der Tiefe der RAM-Blöcke, da zur Adressierung des RAMs die intervallbildenden Bits irrelevant sind.

Die Tiefe eines RAM-Blocks ohne Intervallzusammenfassung hat folgende Tiefe:

$$2^{ceil(log2(r_{max}+r_{min}))} \tag{36}$$

Unter Zusammenfassung der n niederwertigsten Bits in ein Intervall ergibt sich einer RAM-Tiefe von:

$$2^{ceil(log2(r_{max}+r_{min}))-n} \tag{37}$$

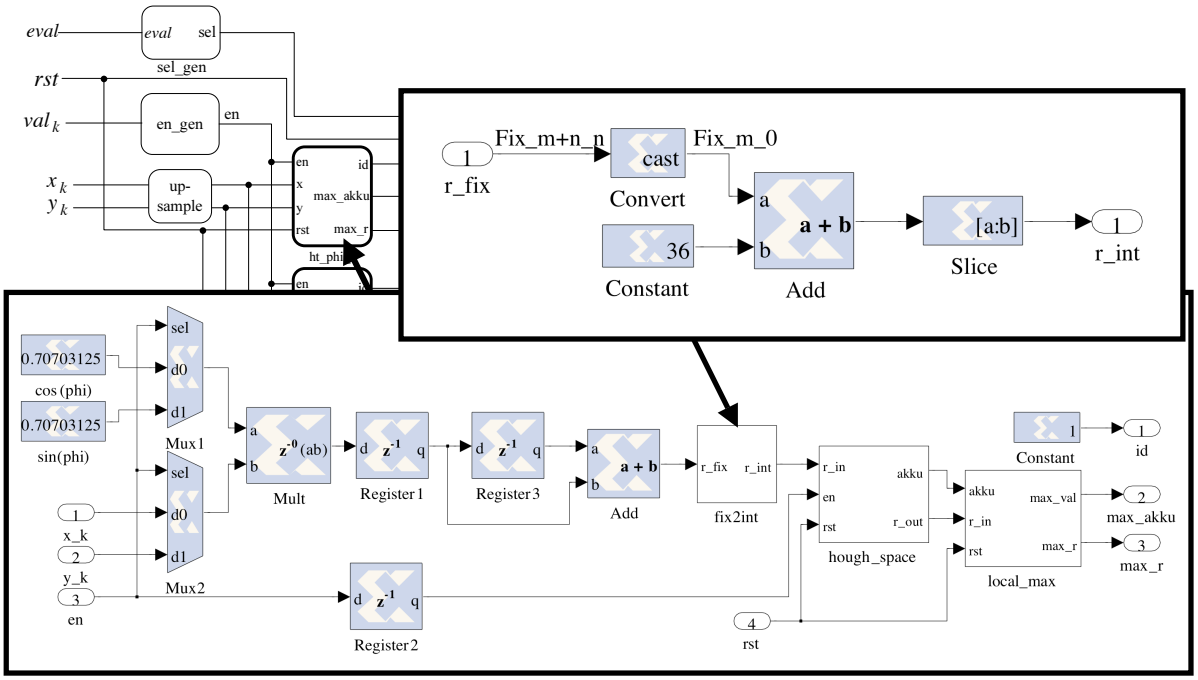

Abb. 50: *Subsystem fix2int, zur Ganzzahlwandlung und Intervallbildung des fixed-point Eingangswertes r_fix*

7.2.4 Akkumulation der Ursprungslotlänge r_i im Houghraum

Die ganzzahlige Ursprungslotlänge r_int (vgl. Kap. 7.2.3) wird als Adresse für den RAM-basierten Houghraum des Houghtransformators *ht_hw_phi_i* verwendet (vgl. Abb. 51). Eine Inkrementierung des Houghraumes an der Speicherzelle r_in erfolgt in zwei Takten:

- Auslesen des aktuellen Akkumulatorwertes an Adresse r_in
- Inkrementieren des gelesenen Wertes und Zurückschreiben an Adresse r_in. Die Adressselektion erfolgt mit dem verzögerten *enable*-Signal (vgl. Kap. 7.2.1)

Der RAM-Block wird bei *enable*-Signal-Generierung durch ein Vordergrundpixel für einen Takt für einen Schreibvorgang freigegeben (vgl. Abb. 52). Die Inkrementierung der ausgelesenen Akkumulatorwerte wird kontinuierlich durchgeführt, der Schreibzugriff auf den RAM-Block jedoch nur bei Schreibfreigabe durch das *enable*-Signal.

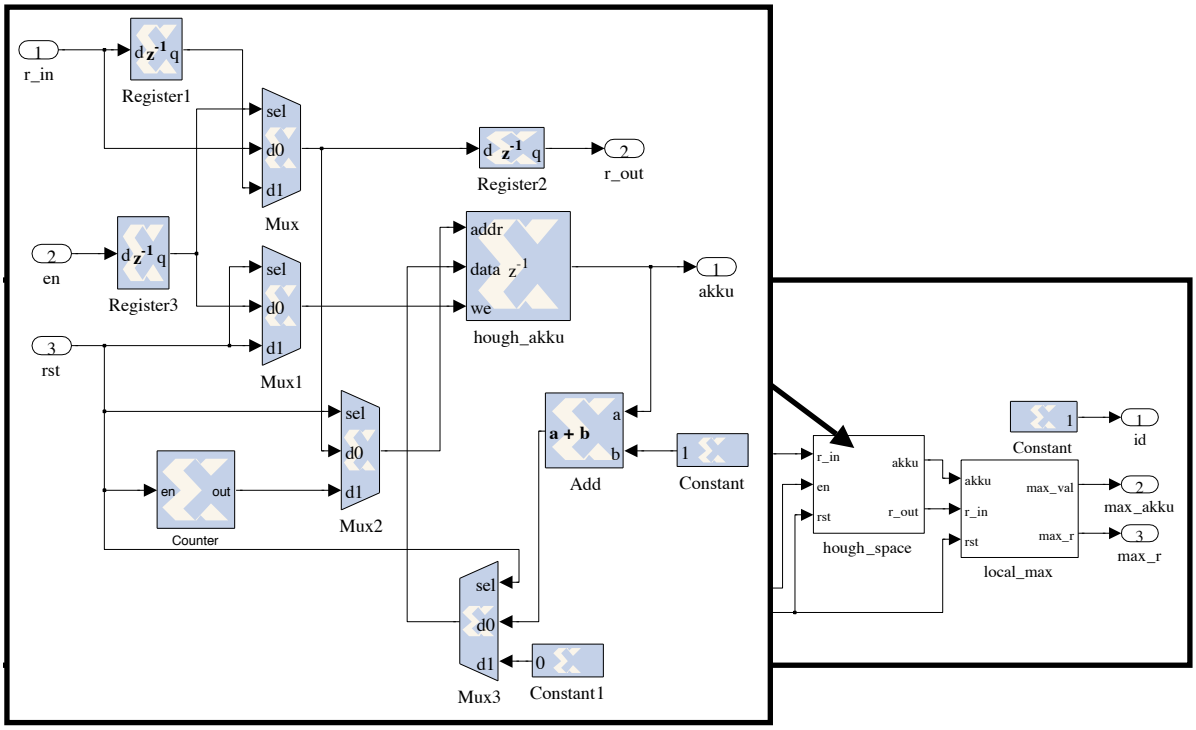

Abb. 51: *Im Subsystem hough_space werden inkrementierte Akkumulatorwerte an Adresse r_in bei anliegendem enable-Signal in zwei Takten im RAM-Block hough_akku gespeichert*

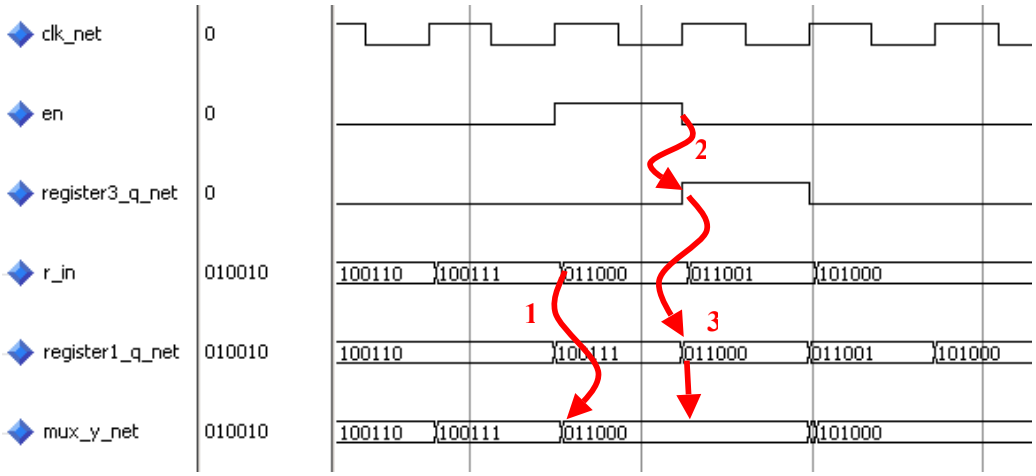

Abb. 52: *VHDL-Simulation der Adressselektion im Subsystem hough_space*

Nr.	Beschreibung
1	Da der Ausgang des *enable*-Registers einen *low*-Pegel besitzt, wird das Eingangssignal r_in auf den Multiplexerausgang geschaltet.
2	Das *enable*-Signal wird in das *enable*-Register übernommen.
3	Der Ausgangspegel des *enable*-Registers ist *low*. Der Inhalt des Register 1wird auf den Ausgang des Multiplexers geschaltet.

Tabelle 9: *Erläuterung des Timings der VHDL-Simulation zu Abb. 52*

7.2.5 Resetverhalten des Houghraumes

Eine Initialisierung der RAM-Blöcke nach einer abgeschlossenen Hough-Transformation erfolgt über ein *reset*-Signal, das mit jeder Taktflanke den RAM-Block an einer über einen Zähler bestimmten Adresse mit Nullen beschreibt (vgl. Abb. 51). Zur Initialisierung eines kompletten RAM-Blocks wird das *reset*-Signal $|r_{min}| + |r_{max}|$ überabgetastete Takte angelegt.

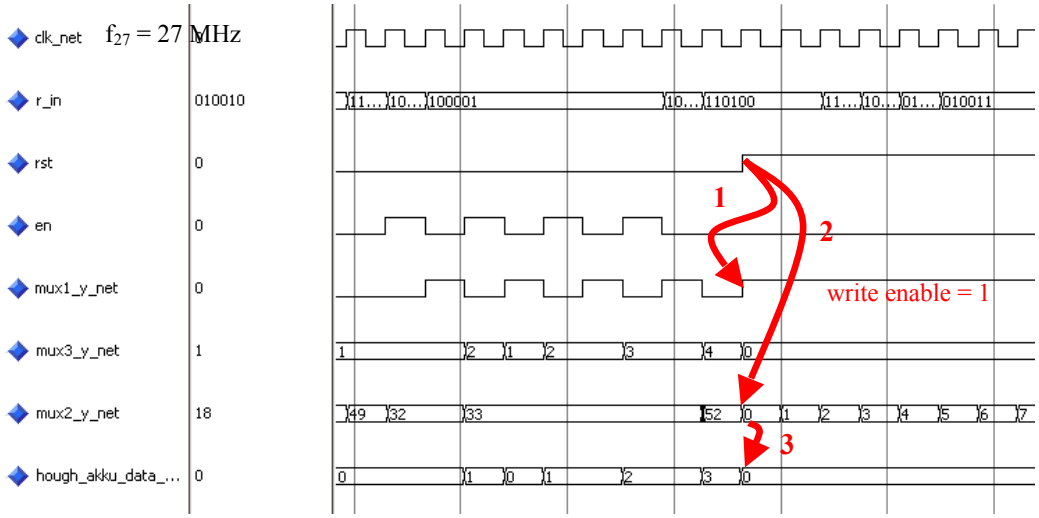

Abb. 53: *VHDL-Simulation des Resetverhaltens des RAM-basierten Houghraumes*

Nr.	Beschreibung
1	Das *reset*-Signal nimmt einen *high*-Pegel an und schaltet diesen auf den *write enable* Port des RAM-Blocks.
2	Das *reset*-Signal gibt den Adresszähler frei, der mit einer Frequenz f_{27} = 27 MHz eine aktualisierte Adresse an den Adressport des RAMs anlegt.
3	Solange das *reset*-Signal einen *high*-Pegel besitzt, wird der Datenport des RAMs mit Nullen beschrieben.

Tabelle 10: *Erläuterung des Timings der VHDL-Simulation zu Abb. 53*

7.2.6 Lokale Maximumbestimmung

Zur Bestimmung des lokalen Maximums des Houghtransformators *ht_hw_phi_i*, wird der aktuelle maximale Akkumulatorwert max_val gespeichert. Das lokale Maximum wird durch den Akkumulatorwert max_val und die Speicheradresse des Akkumulatorwertes max_r identifiziert, die der Ursprungslotlänge entspricht, auf der unter einem Winkel ϕ_i max_val Punkte liegen (vgl. Kap. 6.2). Die Register zur Speicherung der Maxima werden über den Komparatorausgang, der ein *write enable* realisiert, für einen Schreibzugriff freigegeben (vgl. Abb. 54). Über einen Resetmechanismus wird eine Beeinflussung einer aktuellen Maximumsuche durch vorherige Maximalwerte verhindert.

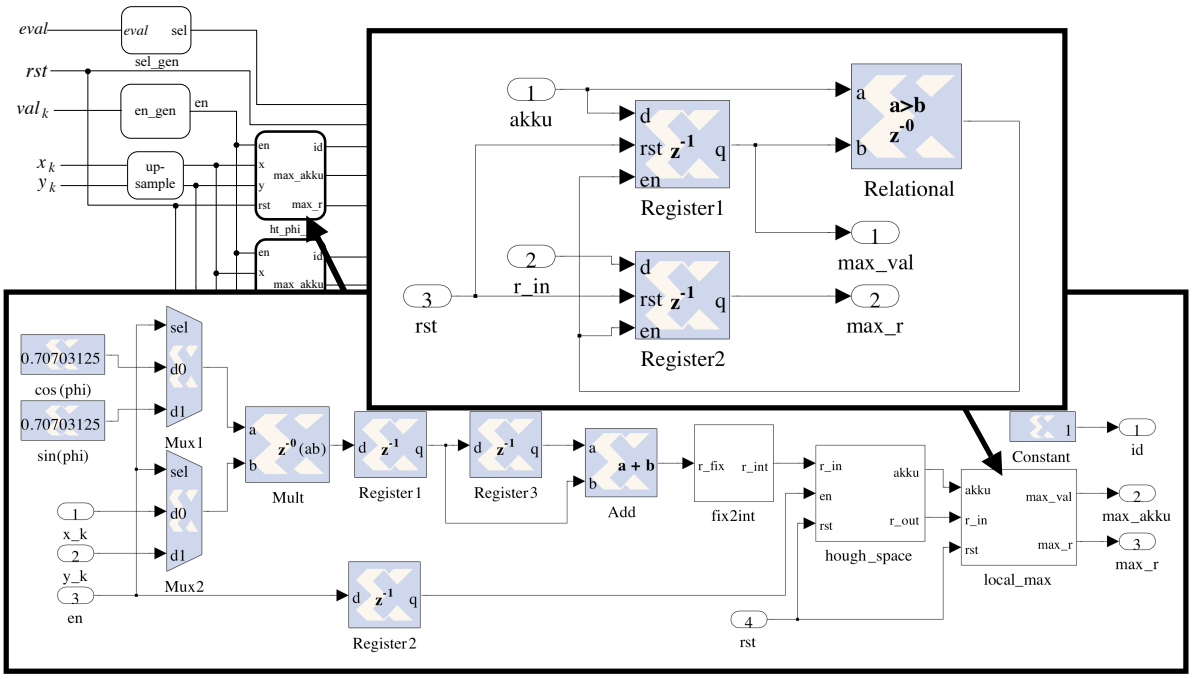

Abb. 54: *Die lokale Maximumsuche verwendet ein komparatorgeneriertes write enable Signal, das die Register für den Schreibzugriff freigibt.*

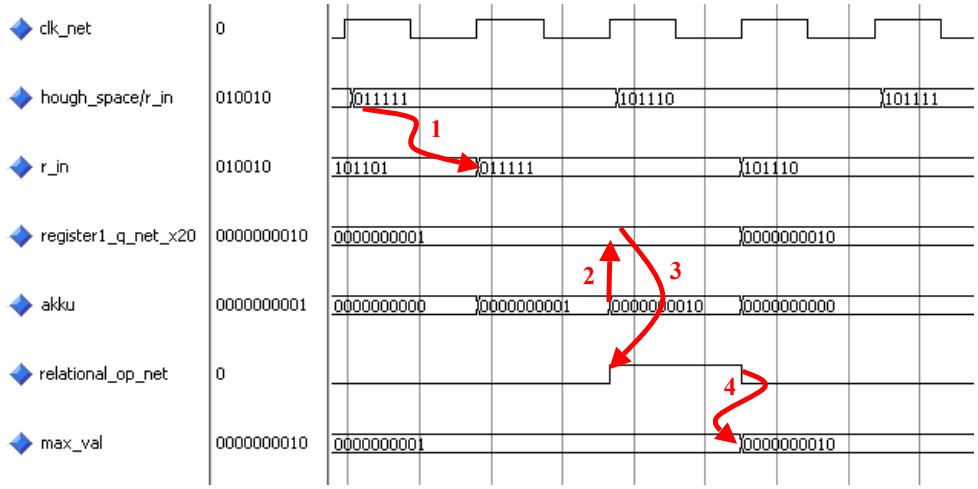

Abb. 55: *VHDL-Simulation der lokalen Maximumermittlung im Subsystem local_max*

Nr.	Beschreibung
1	Der Eingangswert des Subsystems *hough_space* r_in wird um einen Takt verzögert, um so eine Zuordnung des Akkumulatorausganges zur Adresse r_in zu ermöglichen. Das inkrementierte Ergebnis erscheint erst nach dem Schreibzugriff am RAM-Ausgang.
2 & 3	Der Akkumulatorwert ist größer als der im Akkumulatorwertregister gespeicherte. Der Komparator erzeugt ein *enable*-Signal.
4	Der Akkumulatorwert erscheint einen Takt verzögert als aktuelles Maximum am Ausgang des Akkumulatorwertregisters.

Tabelle 11: *Erläuterung zu Abb. 55*

7.2.7 Ergebnisermittlung durch globale Maximumsuche

Die Ergebnisermittlung der Hough-Transformation beschränkt sich auf eine globale Maximumsuche, da auf diese Weise der Punkt (r, ϕ) (vgl. Kap. 6) im Houghraum gefunden wird, in dem sich die meisten Schnittpunkte befinden [10]. Die Komponenten zur Ermittlung des lokalen Maximums (vgl. Kap. 7.2.6) legen die ermittelten Werte an das Subsystem zur Berechnung des globalen Maximums an:

- den Identifikator id_i des jeweiligen Houghtransformators *ht_hw_phi_i*,
- den Maximalen Akkumulatorwert $akku\_i$ des Houghtransformators *ht_hw_phi_i*,
- die Ursprungslotlänge r_i, die der Speicherzelle entspricht, an der der maximale Akkumulatorwert $akku\_i$ gespeichert ist.

Die Ermittlung des globalen Maximums erfolgt unter Auswertung der lokalen Maxima der Houghtransformatoren *ht_hw_phi_i*. Mit jedem Takt wird der aktuell über ein *select*-Signal (vgl. Kap. 7.2.8) selektierte Akkumulatorwert $akku\_i$ mit dem bisher als Maximum bestimmten Akkumulatorwert max_akku verglichen und im Falle einer Maximalität in die Register zur Speicherung übernommen (vgl. Abb. 56). Ein Resetmechanismus initialisiert die Register und entfernt so die vorherigen Transformationsergebnisse.

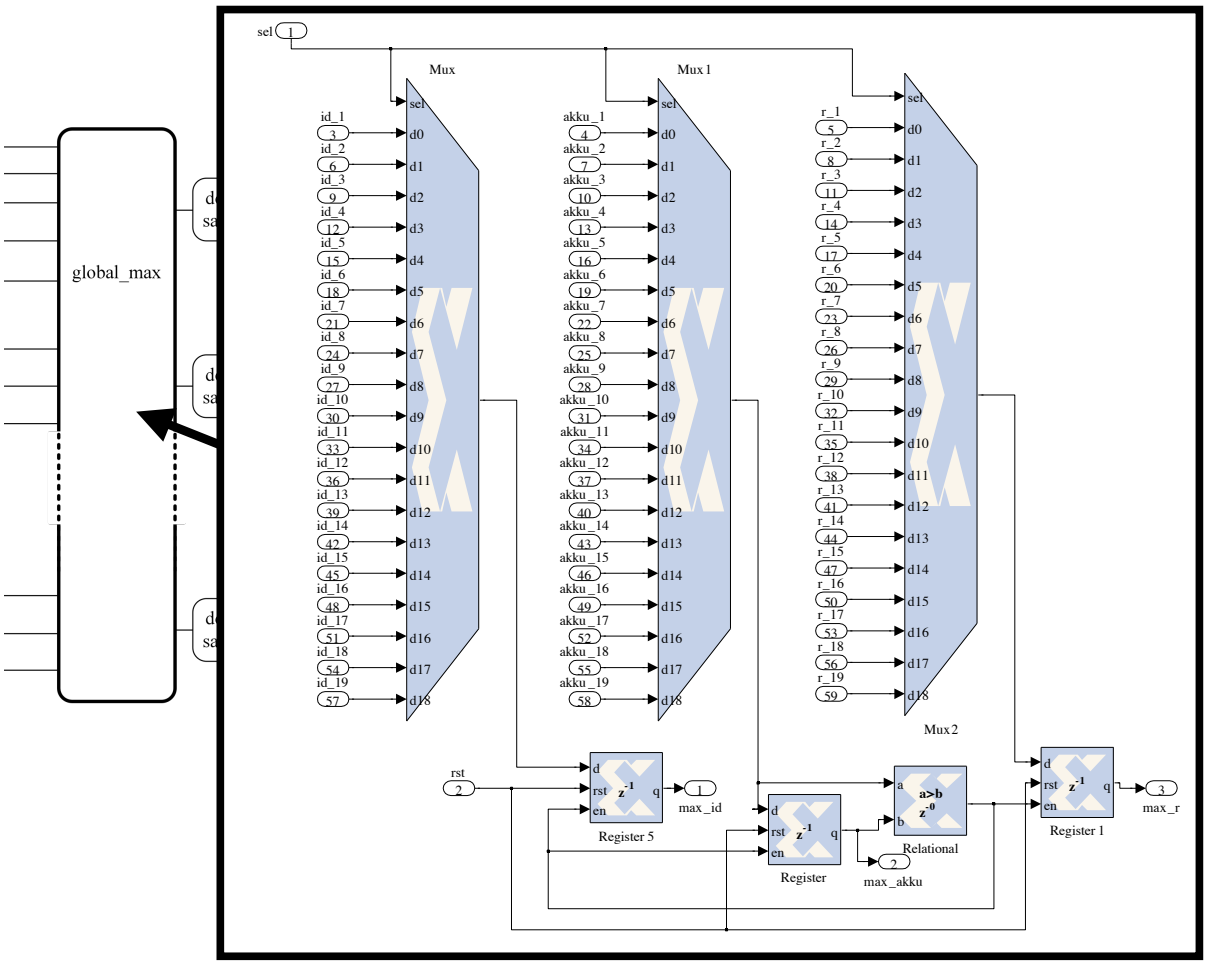

Abb. 56: *Ein maximaler Akkumulatorwert schaltet das Freigabesignal der Register zur Übernahme aktueller Maximalwerte*

7.2.8 Generierung des *select*-Signals zur globalen Maximumsuche

Für die globale Maximumsuche (vgl. Kap. 7.2.7) wird ein *select*-Signal verwendet, über das das lokale Maximum des jeweiligen Houghtransformators *ht_hw_phi_i* ausgewählt wird. Das Subsystem zur Generierung des Signals (vgl. Abb. 57) beinhaltet einen Zähler, der über das $eval$ Signal zurückgesetzt wird und einen Zählvorgang mit der Frequenz f_{27} = 27 MHz startet, der jeden Houghtransformator *ht_hw_phi_i* selektiert.

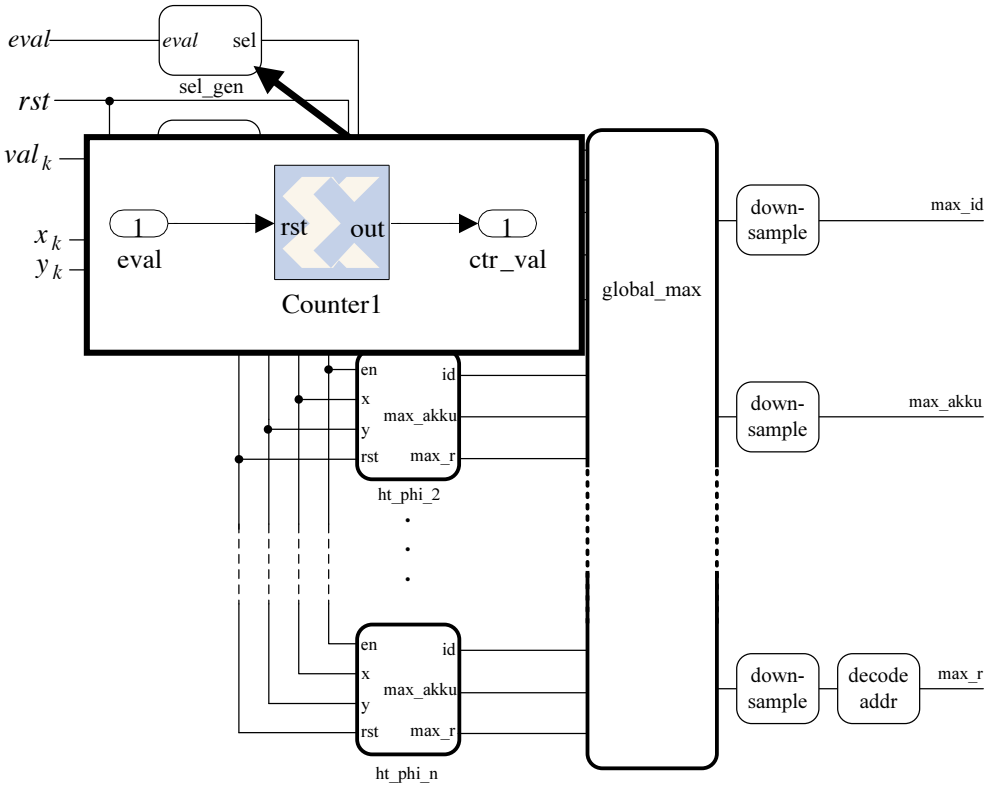

Abb. 57: *Ein high-Pegel des $eval$-Signals startet die select-Signal Generierung für die Multiplexer zur globalen Maximumermittlung (vgl. Kap. 7.2.7) in Form eines Inkrementiervorganges*

7.2.9 Rückgewinnung der Ursprungslotlänge r aus der RAM-Zelle mit maximalem Akkumulatorwert

Die als globales Maximum ermittelte Ursprungslotlänge max_r (vgl. Kap. 7.2.7) stellt einen Wert dar, der durch das Abschneiden von niederwertigen Bits in ein Intervall zusammengefasst wurde (vgl. Kap. 7.2.3) und muss abschliessend wieder in die ursprüngliche ganzzahlige Ursprungslotlänge konvertiert werden, um eine Ergebnisinterpretation zu ermöglichen. Die Rückwandlung erfolgt in zwei Schritten:

- Konkatinieren der durch globale Maximumsuche ermittelten Adresse max_r mit der abgeschnittenen Anzahl Nullen (vgl. Kap. 7.2.3). Ein Anhängen von Nullen erzeugt eine nicht rekonstruierbare Ungenauigkeit, da Bits, die in der Intervallbildung abgeschnitten wurden, durch das Anhängen von Nullen ersetzt werden.
- Subtrahieren des Betrags der minimalen Ursprungslotlänge r_{min}, die zur Offsetkorrektur im Subsystem *fix2int* addiert wurde.

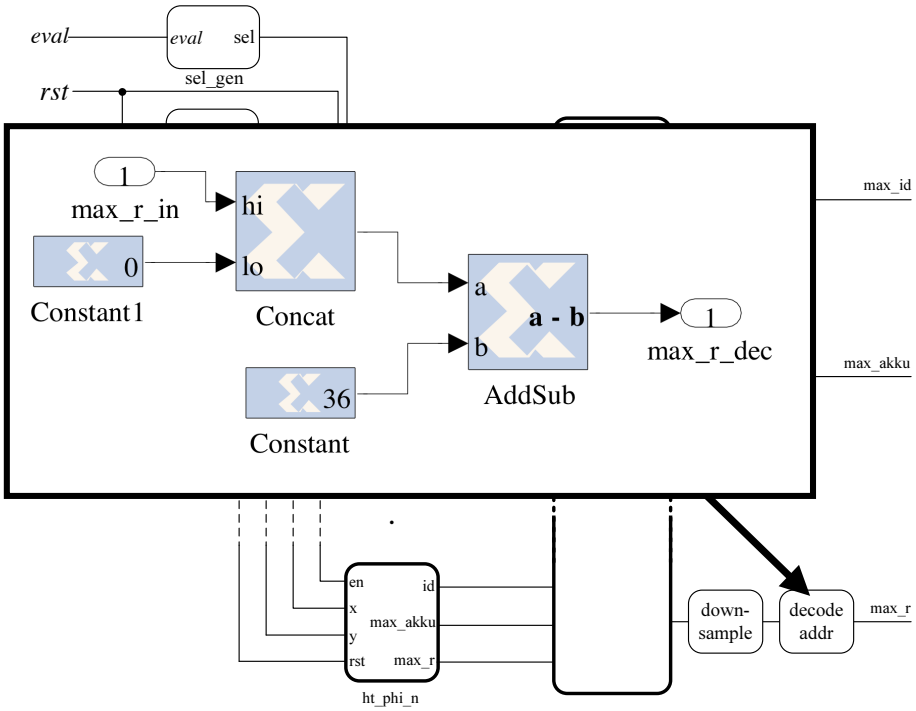

Abb. 58: *Die Rückgewinnung der Ursprungslotlänge r erfolgt durch ein Konkatinieren mit der Anzahl vorher abgeschnittener Nullen und einer Offsetkorrektur*

7.3 Ressourcenbedarf

Das Design inferriert 19 RAM-Blöcke, die als Block-RAM oder als Distributed-RAM konfiguriert werden können. Bei einer Konfiguration als Distributed-RAM werden die Lookup-Tabellen der Slices als RAM-Zellen alloziert und beanspruchen FPGA-Ressourcen. Dieses führt durch einen hohen Verdrahtungsaufwand zu einer Senkung der maximal erreichbaren Taktrate [27]. Weiter bietet das Spartan 3E FPGA spezialisierte Block-RAM Funktionsblöcke, die keine FPGA-Ressourcen verbrauchen.

Block-RAM-Funktionsblöcke beanspruchen zusätzlich Ressourcen des angrenzenden eingebetteten Multiplizieres, der daher unbenutzt bleiben muss. Dieses ist dadurch begründet, dass einige Pins benachbarter Block-RAM-Funktionsblöcke und eingebetteter Multiplizierer Routing Ressourcen gemeinsam verwenden [38].

Der Ressourcenbedarf des Systems unter Verwendung von Block-RAM-Funktionsblöcken und unter Ausschluss von eingebetteten Multiplizierern ist in Tabelle 12 dargestellt. Eine Konfiguration des Systems zur Verwendung von Distributed-RAM und eingebetteten Multiplizierern bewirkt den in Tabelle 13 dargestellten Ressourcenbedarf. Zur Allozierung des Distributed-RAM werden 760 LUTs beansprucht. Ein Vergleich der beiden Implementierungen zeigt, dass die Verwendung von Distributed-RAM ca. 2% mehr Slices und ca. 1% mehr Flip Flops benötigt als eine Block-RAM basierte Realisierung, die 100 4-Eingang LUTs mehr benötigt.

Logiknutzung	Benutzt	Verfügbar	%
Anzahl der Slices	1989	8672	22
Anzahl der Slice Flip Flops	1421	17344	8
Anzahl der 4 Input LUTs	2901	17344	16
Anzahl der BRAMs	19	28	67

Tabelle 12: *Ressourcenbedarf aus dem ISE Synthese Report unter Verwendung von Block-RAM und unter Ausschluss von eingebetteten Multiplizierern*

Logiknutzung	Benutzt	Verfügbar	%
Anzahl der Slices	2122	8672	24
Anzahl der Slice Flip Flops	1608	17344	9
Anzahl der 4 Input LUTs	2800	17344	16
Anzahl der MULT18X18SIOs	19	28	67

Tabelle 13: *Ressourcenbedarf aus dem ISE Synthese Report unter Verwendung von eingebetteten Multiplizierern und Distributed-RAM*

7.4 Analyse des längsten Laufzeitpfades

Der längste Laufzeitpfad in einem digitalen System identifiziert den längsten in der Schaltung existierenden Logikpfad aller Übergangslogikstufen [27]. Durch Optimierung des Laufzeitpfades ist eine Steigerung der erreichbaren Maximalfrequenz erzielbar. Der längste Laufzeitpfad wird durch den Place & Route Implementierungsschritt festgestellt und ist in Abbildung 59 dargestellt. Betroffen sind das im Subsystem *en_gen* generierte *enable*-Signal (vgl. Kap. 7.2.3), das zum Operandenmultiplexen für die Multiplikation $x_k \cdot cos(\phi_i)$ bzw. $y_k \cdot sin(\phi_i)$ verwendet wird und die anschließende Multiplikation (vgl. Kap. 7.2.2).

Die Verzögerung durch den längsten Datenpfad beträgt sowohl für eine Implementierung unter Einsatz von eingebetteten Multiplizieren als auch für Multiplizierer die in LUTs implementiert werden ca. 22 ns, die sich aus

- ca 30% Logiklaufzeit und
- ca 70% Routinglaufzeit zusammensetzt.

Aus der Summe der Logik- und Routinglaufzeit ergibt sich eine maximale Frequenz f_{max} = 45 MHz.

Beim Einsatz von Multiplizierern, die in LUTs implementiert werden, ergeben sich 9 Übergangslogikstufen, da die Multiplizierer aus FPGA-Ressourcen realisiert werden und so zum Laufzeitpfad beisteuern. Bei einer Konfiguration unter Einsatz von eingebetteten Multiplizierern ergeben sich 2 Übergangslogikstufen, da diese eigene Funktionsblöcke darstellen.

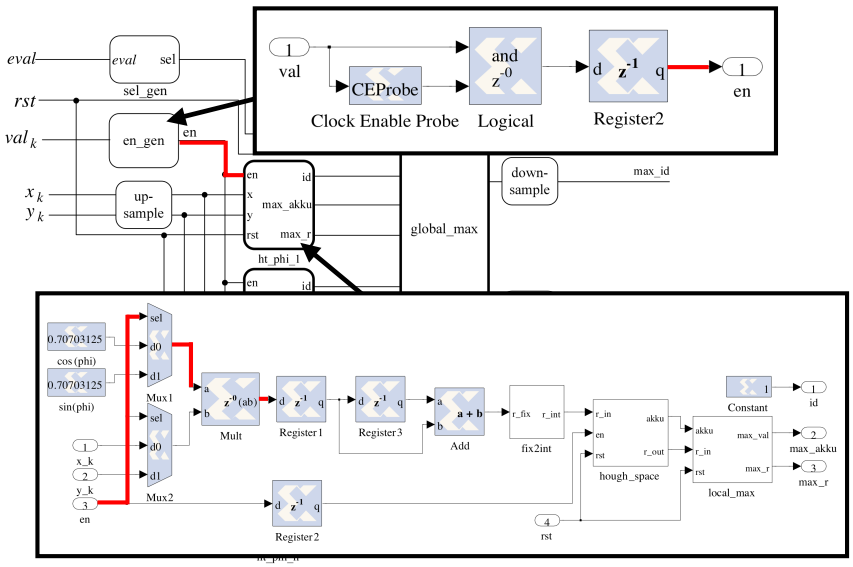

Abb. 59: *Der längste Laufzeitpfad vom enable-Signal Register im Subsystem en_gen zum Multiplizierereausgang im Subsystem ht_ht_phi_i*

8 Ergebnisanalyse

Folgender Abschnitt stellt eine Analyse der Ergebnisse der Hough-Transformation vor und geht hierbei auf Unterschiede ein, die durch eine Intervallbildung mit unterschiedlichen Bitbreiten entstehen (vgl. Kap. 7.2.3). Eine Fahrspurbegrenzung wird nach der Sobel-Filterung (vgl. Kap. 4.2) durch zwei Linien dargestellt, da der Sobel-Filter Kanten detektiert (vgl. Abb. 60). Beide Linien stellen Geraden dar, die durch die Hough-Transformation ermittelt werden, wobei je nach Intervallgröße ein Sprung zwischen den Geraden erfolgt.

Die Auswertung der Transformationsergebnisse erfolgt in dieser Arbeit durch:

- Aufnahme der Fahrspur und Übertragung an einen Desktop PC [17]
- Projektive Transformation der übertragenen Aufnahme (vgl. Kap. 5.4)
- Festlegen der *Region of Interest* (Größe: 100 x 50)
- Berechnung der Hough-Transformation für die Vordergrundpixel, die nach der projektiven Transformation in der *Region of Interest* liegen (vgl. Anhang B) für eine Bildung von Intervallen unter der Zusammenfassung von einer, zwei und drei niederwertigen Bitpositionen (vgl. Kap. 7.2.3)

Eine Implementierung des Moduls zur Überprüfung eines projektiv transformierten Pixels zur *Region of Interest* existiert zum Zeitpunkt der Erstellung dieser Arbeit nicht, weshalb eine Überprüfung in einem Matlab Skript realisiert wird. Bei einer Auswertung der projektiv transformierten Pixelkoordinaten im Pixelstrom werden in dieser Arbeit keine Maßnahmen zur Verhinderung von Mehrfachtransformation der gleichen Pixelkoordinate ergriffen. Die Hough-Transformation des Pixels mit den Koordinaten (x_{k_i}, y_{k_i}) wird bei jedem Vorkommen berechnet.

Für die Hough-Transformation der in Abbildung 61 dargestellten *Region of Interest* der Fahrspuraufnahme (vgl. Abb. 60) ergeben sich die in Tabelle 14 dargestellten Geradenparameter.

Abb. 60: *Aufnahme der Fahrspur von der auf dem Fahrzeug angebrachten Sony FCB-PV10*

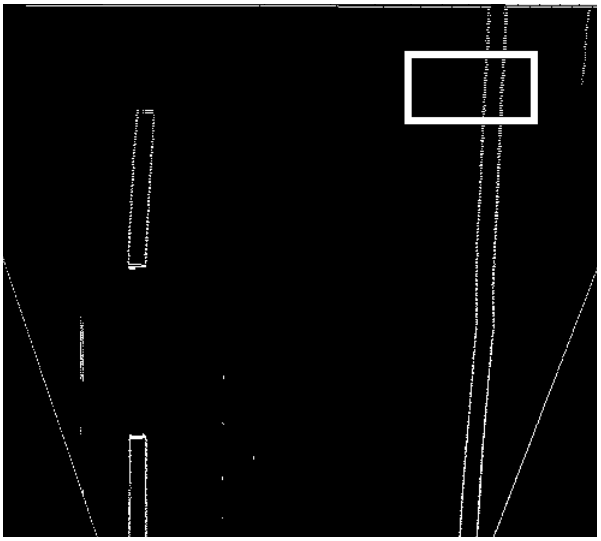

Abb. 61: *Projektive Transformation des Eingangsbildes 60 mit markierter Region of Interest*

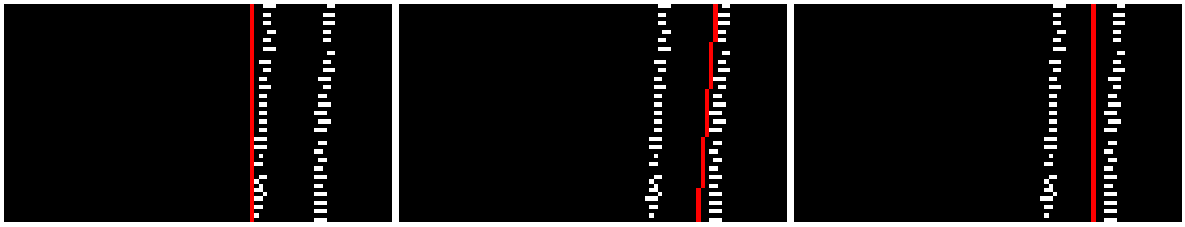

Abb. 62: *Darstellung des Ergebnisses der Hough-Transformation für Pixel, die in der Region of Interest (Abbildung 61) liegen; Intervalle: 1 Bit (links), 2 Bit (mitte), 3 Bit (rechts) (vgl. Kap. 7.2.3)*

	Winkel ϕ zwischen Ursprungslot und x-Achse	**Länge r des Ursprungslots**	**Akkumulatorwert der RAM-Zelle**
1 Bit	0°	60	38
2 Bit	5°	76	61
3 Bit	0°	72	64

Tabelle 14: *Geradenparameter der Hough-Transformation für die Region of Interest in Abbildung 61 mit einem Intervall von 1 Bit, 2 Bit und 3 Bit (vgl. Abb. 62)*

Für die in Abbildung 63 dargestellte Fahrspur ergeben sich für die *Region of Interest* (vgl. Abb. 64) die in Tabelle 15 aufgeführten Geradenparameter.

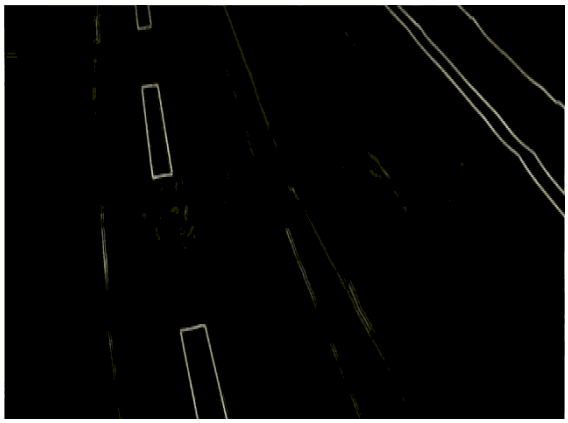

Abb. 63: *Aufnahme der Fahrspur von der auf dem Fahrzeug angebrachten Sony FCB-PV10*

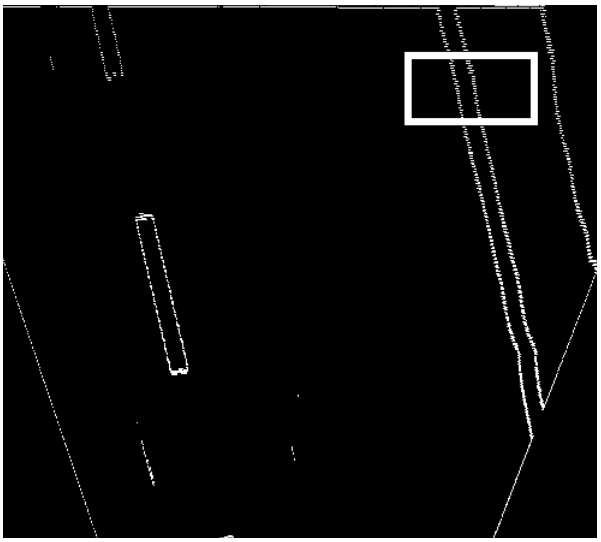

Abb. 64: *Projektive Transformation des Eingangsbildes 63 mit markierter Region of Interest*

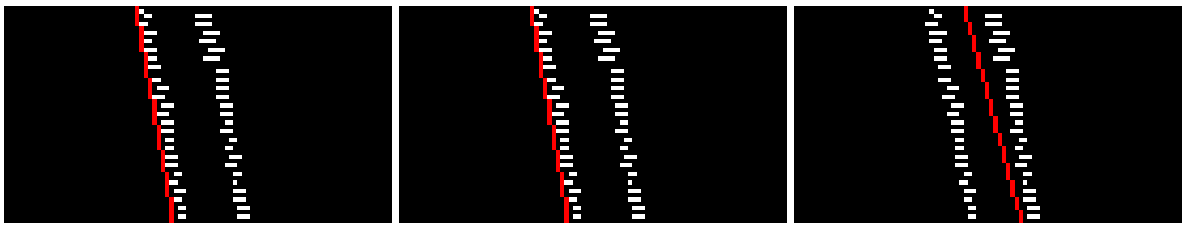

Abb. 65: *Darstellung des Ergebnisses der Hough-Transformation für Pixel, die in der Region of Interest (Abbildung 64) liegen; Intervalle: 1 Bit (links), 2 Bit (mitte), 3 Bit (rechts) (vgl. Kap. 7.2.3)*

	Winkel ϕ zwischen Ursprungslot und x-Achse	**Länge r des Ursprungslots**	**Akkumulatorwert der RAM-Zelle**
1 Bit	$-10°$	32	43
2 Bit	$-10°$	32	63
3 Bit	$-15°$	40	74

Tabelle 15: *Geradenparameter der Hough-Transformation für die Region of Interest in Abbildung 64 mit einem Intervall von 1 Bit, 2 Bit und 3 Bit (vgl. Abb. 65)*

Die Geradenparameter, die sich für die Hough-Transformation der in Abbildung 67 dargestellten *Region of Interest* der Fahrspuraufnahme (vgl. Abb. 66) ergeben, sind in Tabelle 16 dargestellt.

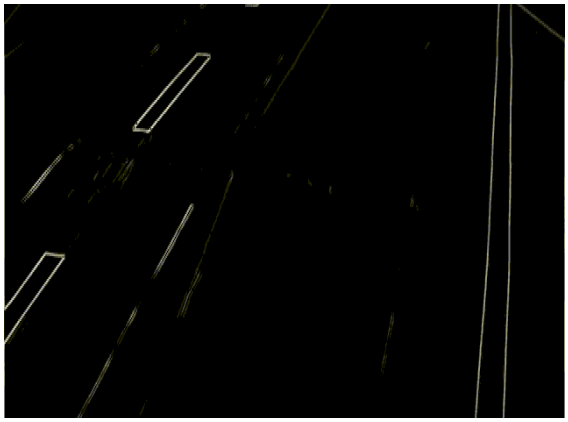

Abb. 66: *Aufnahme der Fahrspur von der auf dem Fahrzeug angebrachten Sony FCB-PV10*

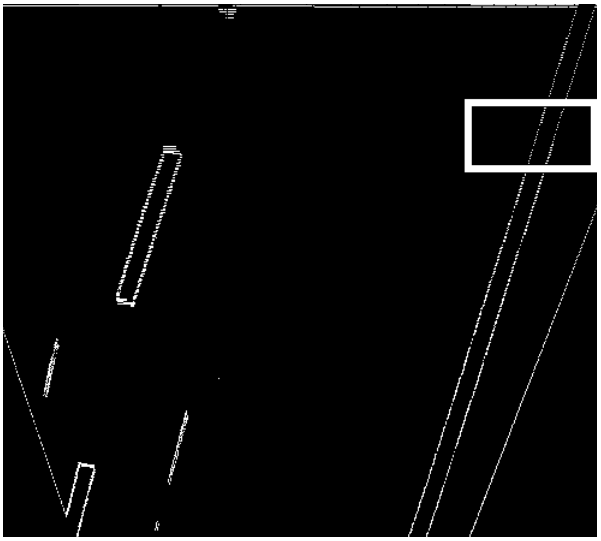

Abb. 67: *Projektive Transformation des Eingangsbildes 66 mit markierter Region of Interest*

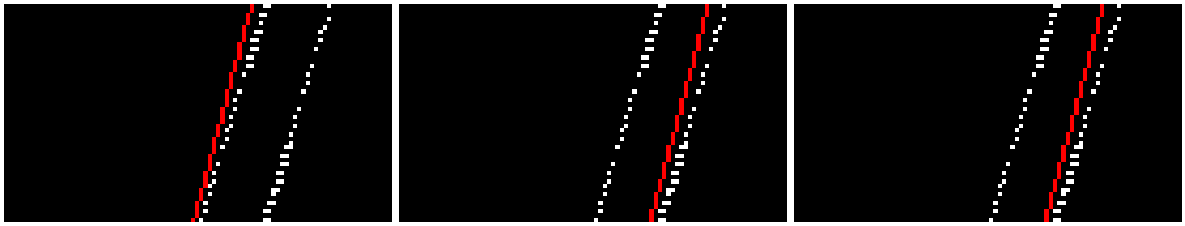

Abb. 68: *Darstellung des Ergebnisses der Hough-Transformation für Pixel, die in der Region of Interest (Abbildung 67) liegen; Intervalle: 1 Bit (links), 2 Bit (mitte), 3 Bit (rechts) (vgl. Kap. 7.2.3)*

	Winkel ϕ zwischen Ursprungslot und x-Achse	**Länge r des Ursprungslots**	**Akkumulatorwert der RAM-Zelle**
1 Bit	$15°$	58	32
2 Bit	$15°$	72	38
3 Bit	$15°$	72	38

Tabelle 16: *Geradenparameter der Hough-Transformation für die Region of Interest in Abbildung 67 mit einem Intervall von 1 Bit, 2 Bit und 3 Bit (vgl. Abb. 68)*

Die in Tabelle 14, 15 und 16 dargestellten Geradenparameter zeigen, dass das Ergebnis der Hough-Transformation je nach Intervallbreite variiert:

- Es ist eine Erhöhung des Akkumulatorwertes der RAM-Zelle r festzustellen, der in der erhöhten Breite begründet ist.
- Es sind Sprünge zwischen den Linien der Fahrspur erkennbar, die unter Veränderung der Intervallbreite auftreten, da eine Veränderung der Vordergrundpixel, die in einem Intervall liegen, durchgeführt wird (vgl. Kap. 6).
- Der Winkel ϕ der Ergebnisgerade verändert sich, da der Winkel der Geraden, die durch ein breites Intervall läuft, nicht genau bestimmbar ist.

9 Zusammenfassung

Diese Arbeit liefert Beiträge für die Entwicklung eines Spurführungssystems im Kontext des Forschungsprojektes FAUST der Hochschule für Angewandte Wissenschaften Hamburg. Aufbauend auf der in [17] und [26] entwickelten Videopipeline zur Bilddatenaufbereitung, wurde in dieser Arbeit eine Video-basierte Fahrspurerkennung mit dem Xilinx System Generator modelliert.

Der Pixelstrom zur Fahrspurapproximation wird mit einer Frequenz von $f_{13.5}$ = 13,5 MHz zur Verfügung gestellt, wobei eine Kantenextraktion mit dem Sobel-Filter durchgeführt wird, da die Fahrspurbegrenzung sich farblich deutlich von der Fahrbahn unterscheidet und so eine Fokussierung auf relevante Fahrspurinformationen erreicht wird.

Die Verzeichnung des Kameraobjektivs der Sony FCB-PV10, die durch den Fertigungsprozess entsteht, wurde in dieser Arbeit untersucht. Da keine starke Verzeichnung entsteht, wurde auf eine Hardwareimplementierung der Linsenverzeichnungskorrketur verzichtet.

Eine Korrektur der perspektivischen Verzeichnung, die durch eine Abbildung des Kamerabildes auf die Bildebene auftritt, wird mit einer projektiven Transformation erreicht, in der Aufnahmen in der Bildebene zurück in die Fahrzeugebene projiziert werden. Das System Generator Modell berechnet die projektive Transformation eines Pixels in einer Pipelinestruktur mit übertakteten Pipelinestufen mit der Frequenz f_{54} = 54 MHz. Die Konfiguration des Dividierers zur Divisorberechnung erfolgt unter Berücksichtigung der übertakteten Pipelinestufe und nutzt vier übertaktete Zyklen zur Berechnung des Divisionsergebnisses, wodurch eine Ressourcenersparnis erreicht wird. Projektiv transformierte Pixel werden in unsortierter Reihenfolge an Folgestufen weitergereicht. Die Latenz des Moduls beträgt 8 Takte bei $T_{13.5}$, die durch die Dividiererlatenz bedingt ist. Eine Laufzeitanalyse nach dem Place & Route Implementierungsschritt ergibt eine Maximalfrequenz f_{max} von ca. 76 MHz für den mit f_{54} übertakteten Bereich.

Die mathematischen Grundlagen der Hough-Transformation wurden vorgestellt, um eine Dimensionierung und anschließende Diskretisierung des Houghraumes durchführen zu können. Eine Einschränkung des Winkelbereichs ϕ wird unter Berücksichtigung von negativen Ursprungslotlängen r durchgeführt. In Hardware wird so eine Verringerung des Bedarfs an RAM-basierten Akkumulatoren erreicht, da für jeden Winkelbereichsausschnitt ϕ_k ein RAM-Funktionsblock zur Verfügung gestellt wird. Eine Verkleinerung der Tiefe der RAM-Blöcke wird über eine Intervallbildung über ein Abschneiden von niederwertigen Bits erreicht.

Ein Modul zur Fahrspurapproximation wurde in einer Pipelinestruktur mit übertakteten Pipelinestufen modelliert und so eine Verarbeitung im Pixelstrom ohne Bildzwischenspeicherung erreicht. Die Inkrementierung eines Akkumulatorwertes erfolgt in zwei Zyklen bei einer Frequenz von f_{27} = 27 MHz, die eine Adressierung und eine Inkrementierung des Akkumulatorwertes beinhalten. Das Verhalten der Subsysteme des Modells wurde anhand von VHDL-Simulationen und einer anschließenden Ergebnisanalyse verifiziert.

Auf dem Spartan 3E FPGA schließt der Einsatz von Block-RAM Funktionsblöcken in der vorgestellten Konfiguration den Einsatz von eingebetteten Multiplizieren aus, da diese gemeinsame Ressourcen nutzen. Eine Konfiguration des Systems zur Verwendung von Distributed-RAM ermöglicht den Einsatz von eingebetteten Multiplizierern. Der Ressourcenbedarf wurde sowohl unter Einsatz von Block-RAM als auch unter Einsatz von Distributed-RAM ermittelt und gegenübergestellt. Der längste Laufzeitpfad wurde ermittelt und beträgt ca. 22 ns, was eine Maximalfrequenz von f_{max} = 45 MHz impliziert und eine Grenze der maximalen Pixelfrequenz darstellt.

Literaturverzeichnis

[1] ANGERMANN, Anne ; BEUSCHEL, Michael ; RAU, Martin ; WOHLFARTH, Ulrich: *MATLAB - Simulink - Stateflow - Grundlagen, Toolboxen, Beispiele*. 6. München : Oldenbourg Wissenschaftsverlag, 2009. – ISBN 3-486-58985-6

[2] ASHENDEN, Peter J.: *The Designer's Guide to VHDL*. 3. Amsterdam : Elsevier (The Morgan Kaufmann Series in Systems on Silicon). – ISBN 978-0-12-088785-9

[3] BORDASCH, Heiko: *VHDL-Modellierung einer Geschwindigkeitsregelung für ein autonomes Fahrzeug implementiert auf einer SoC-Plattform*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2009

[4] BURGE, Mark J. ; BURGER, Wilhelm: *Digitale Bildverarbeitung - Eine Einführung mit Java und ImageJ*. 2. Berlin : Springer-Verlag, 2006. – URL http://www.springerlink.com/content/m00486/. – ISBN 978-3-540-30940-6

[5] CAROLO, Cup: *Carolo Cup Regelwerk 2010*. 2010. – URL http://www.carolo-cup.de/uploads/media/20100127_Carolo-Cup_Regelwerk.pdf

[6] DIGILENT, Inc.: *Nexys-2*. 2008. – URL http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,789&Prod=NEXYS2. – Abruf: 20. April 2010

[7] DISCHER, Christian ; ROSSBERG, Dirk ; RUSS, Artur: Das Auto lernt sehen - Implementierung verschiedener Fahrerassistenzfunktionen mit nur einer Kamera. In: *Elektronik Automotive - Fachmagazin für Entwicklungen in der Kfz-Elektronik und Telematik - Sonderausgabe BMW 7er* (2008)

[8] EOS SYSTEMS, Inc.: PhotoModeler Pro 5. (2003). – URL http://www.eossystems.com/

[9] FISCHER, Julian ; ROSSBERG, Dirk ; RUSS, Artur: Schonzeit für das Punktekonto - Kamerabasierte Verkehrzeichenerkennung. In: *Elektronik Automotive - Fachmagazin für Entwicklungen in der Kfz-Elektronik und Telematik - Sonderausgabe BMW 7er* (2008)

[10] FISCHER, Max ; HABERÄCKER, Peter ; NISCHWITZ, Alfred: *Computergrafik und Bildverarbeitung*. 2. Wiesbaden : Vieweg Verlag, 2007. – URL http://www.springerlink.com/content/j17313/. – ISBN 978-3-8348-0186-9

[11] GAJSKI, Daniel D.: *Principles of Digital Design*. New Jersey : Prentice-Hall, 1996. – ISBN 0-13-301144-5

[12] GROEN, Franciscus C. ; VEEN, Theo M. van: Discretization Errors In The Hough Transform. In: *Pattern Recognition* 14 (1981), Nr. 1 - 6, S. 137 – 145. – URL http://www.sciencedirect.com/science/article/B6V14-48MPHT2-9X/2/f94525050b77b2b05aa8f877c21a2f4c

[13] GRÜNE, Lars ; JUNGE, Oliver: *Gewöhnliche Differentialgleichungen - Eine Einführung aus der Perspektive der dynamischen Systeme*. 1. Wiesbaden : Vieweg + Teubner, 2009. – URL http://www.springerlink.com/content/u51273/. – Abruf: 20. April 2010

[14] HAKULI, Stephan ; WINNER, Hermann ; WOLF, Gabriele: *Handbuch Fahrerassistenzsysteme - Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. 1. Wiesbaden : Vieweg + Teubner, 2009. – ISBN 978-3-8348-0287-3

[15] JENNING, Eike: *Systemidentifikation eines autonomen Fahrzeugs mit einer robusten, kamerabasierten Fahrspurerkennung in Echtzeit*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2008

[16] KHLIFI, Rachid ; LÜBCKE, Michael ; NAGLER, Andreas ; NUBER, Heike ; POPKEN, Markus ; SACHER, Heike ; VUKOTICH, Alejandro: Der sechste Sinn - Überblick über die modernen Fahrerassistenzsysteme im neuen A8. In: *Elektronik Automotive - Fachmagazin für Entwicklungen in der Kfz-Elektronik und Telematik - Sonderausgabe Audi A8* (2010)

[17] KIRSCHKE, Marco: *Echtzeitbildverarbeitung mit einer FPGA-basierten Prozessorelementkette in einem Fahrspurerkennungssystem*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2009

[18] LIENIG, Jens: *Layoutsynthese elektronischer Schaltungen - Grundlegende Algorithmen für die Entwurfsautomatisierung*. 1. Berlin : Springer-Verlag, 2006. – ISBN 978-3-540-29627-0

[19] MANSKE, Nico: *Kamerabasierte Präzisionsnavigation mobiler Systeme im Indoor-Bereich*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2008

[20] MANSKE, Nico ; JOST, Thorsten: *Posenbestimmung in Räumen mit einem 3D-Kameramodell*, Hochschule für Angewandte Wissenschaften Hamburg, Projektbericht, 2008

[21] MEISEL, Andreas: *3D-Bildverarbeitung für feste und bewegte Kameras*, Rheinisch Westfälische Technische Hochschule (RWTH), Dissertation, 1994

[22] MEISEL, Andreas: Das 3D-Kameramodell der Technischen Informatik der HAW. (2006)

[23] MEISEL, Andreas: *Robot Vision*, Hochschule für Angewandte Wissenschaften Hamburg, Vorlesungsunterlagen, 2009. – URL http://www.informatik.haw-hamburg.de/wp_robot_vision.html. – Abruf: 20. April 2010

[24] ORECHER, Sebastian ; ROSSBERG, Dirk ; RUSS, Artur ; SEINSCHE, Axel: Souverän und sicher - auch bei Nacht - Nachtsichtgerät mit Personenerkennung. In: *Elektronik Automotive - Fachmagazin für Entwicklungen in der Kfz-Elektronik und Telematik - Sonderausgabe BMW 7er* (2008)

[25] OSZWALD, Florian ; REUTER, Christian ; ROSSBERG, Dirk ; WILHELM, Sascha ; ZELLER, Armin: Blick für das Besondere - Multikamera-System sorgt für Überblick in Park- und Rangiersituationen. In: *Elektronik Automotive - Fachmagazin für Entwicklungen in der Kfz-Elektronik und Telematik - Sonderausgabe BMW 7er* (2008)

[26] PETERS, Falko: *FPGA-basierte Bildverarbeitungspipeline zur Fahrspurerkennung eines autonomen Fahrzeugs*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2009

[27] REICHARDT, Jürgen ; SCHWARZ, Bernd: *VHDL-Synthese - Entwurf digitaler Schaltungen und Systeme*. 5. München : Oldenbourg Verlag. – ISBN 978-3-486-58987-0

[28] SCHETLER, Denis: *Automatischer Ausweichassistent mit einer Laserscanner - basierten Abstandsregelung für ein fahrerloses Transportsystem*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2007

[29] SONY: FCB-PV10 Color Camera Modul - Technical Manual. (2006)

[30] UNBEHAUEN, Heinz: *Regelungstechnik I*. 14. Wiesbaden : Vieweg+Teubner, 2008. – ISBN 3-8348-0497-6

[31] XILINX: Local Memory Bus (LMB) - Product Specification. (2005). – URL http://www.xilinx.com/support/documentation/ip_documentation/lmb.pdf. – Abruf: 20. April 2010

[32] XILINX: LogiCORE Divider 2.0 Product Specifications. (2008)

[33] XILINX: MicroBlaze Processor Reference Guide - Embedded Development Kit EDK 10.1i. (2008). – URL http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf. – Abruf: 20. April 2010

[34] XILINX: System Generator for DSP - Reference Guide. (2008). – URL http://www.xilinx.com/support/documentation/sw_manuals/sysgen_ref.pdf. – Abruf: 20. April 2010

[35] XILINX: System Generator for DSP - User Guide. (2008). – URL http://www.xilinx.com/support/documentation/sw_manuals/sysgen_user.pdf. – Abruf: 20. April 2010

[36] XILINX: Fast Simplex Link (FSL) - Product Specification. (2009). – URL http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf. – Abruf: 20. April 2010

[37] XILINX: Processor Local Bus (PLB) - Product Specification. (2009). – URL http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf. – Abruf: 20. April 2010

[38] XILINX: Xilinx Spartan-3E FPGA Family: Data Sheet. (2009). – URL http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf. – Abruf: 20. April 2010

Abbildungsverzeichnis

Tabellenverzeichnis

A Berechnung der Transformationsmatrix B in Matlab

```
X_STERN = [1   1   2   2   2   2   2   3   3   3 ...
           3   3   3   3   4   4   4   4   4   4 ...
           4   5   5   5   5   5   5   5   6   6 ...
           6   6   6   6   6   7   7   7   7   7 ...
           7   7   8   8   8   8   9];

Y_STERN = [1   2   1   2   3   4   5   1   2   3 ...
           4   5   6   7   1   2   3   4   5   6 ...
           7   1   2   3   4   5   6   7   1   2 ...
           3   4   5   6   7   1   2   3   4   5 ...
           6   7   1   2   3   4   1];

X       = [24  2   100 81  57  31  3   174 161 146 ...
           128 109 82  50  251 243 237 229 220 209 ...
           193 329 329 330 334 336 339 341 405 412 ...
           420 434 449 464 487 480 492 509 531 555 ...
           585 626 554 573 595 627 625];

Y       = [22  56  24  56  99  150 208 21  56  97  ...
           148 208 286 383 20  54  96  147 208 287 ...
           386 19  54  96  146 208 283 384 19  54  ...
           96  146 209 283 384 23  56  98  148 209 ...
           283 379 26  59  100 149 29];

% Umrechnen der gemessenen Punktkoordinaten in Bildgroesse (640 x 480)
X_STERN=X_STERN.*(640/max(X_STERN));
Y_STERN=Y_STERN.*(480/max(Y_STERN));

% Aufbau des ueberbestimmten Gleichungssystems
L_X = [X(:) Y(:) ones(length(X),1) zeros(length(X),3) -X_STERN(:).*X(:) -X_STERN(:).*Y(:)];
L_Y = [zeros(length(X),3) X(:) Y(:) ones(length(X),1) -Y_STERN(:).*X(:) -Y_STERN(:).*Y(:)];
L_GES = [L_X; L_Y];
RES_GES = [X_STERN'; Y_STERN'];

% Ausgleichsloesung des ueberbestimmten Gleichungssystems
% Least-Square-Solution
b = L_GES\RES_GES

% 3x3 Matrix erstellen
B = zeros(3,3);
B(1,1) = b(1);
B(1,2) = b(2);
B(1,3) = b(3);
B(2,1) = b(4);
B(2,2) = b(5);
B(2,3) = b(6);
B(3,1) = b(7);
B(3,2) = b(8);
B(3,3) = 1;
```

Listing 2: *Berechnung der Transformationsmatrix* ***B*** *in Matlab für die ermittelten Punktkorrespondenzen*

Transformationsmatrix B

$B_{11} = 0,95787533743$	$B_{12} = 0,89324889035$	$B_{13} = 35,618185767$
$B_{21} = -0,01733617936$	$B_{22} = 2,4352624963$	$B_{23} = 22,498388567$
$B_{31} = -5,789037762e-05$	$B_{32} = 0,0026128713151$	$B_{33} = 1$

Tabelle 17: *Koeffizienten* B_{ij} *der Transformationsmatix B*

B Konzept zur Überprüfung auf Zugehörigkeit zur *Region of Interest*

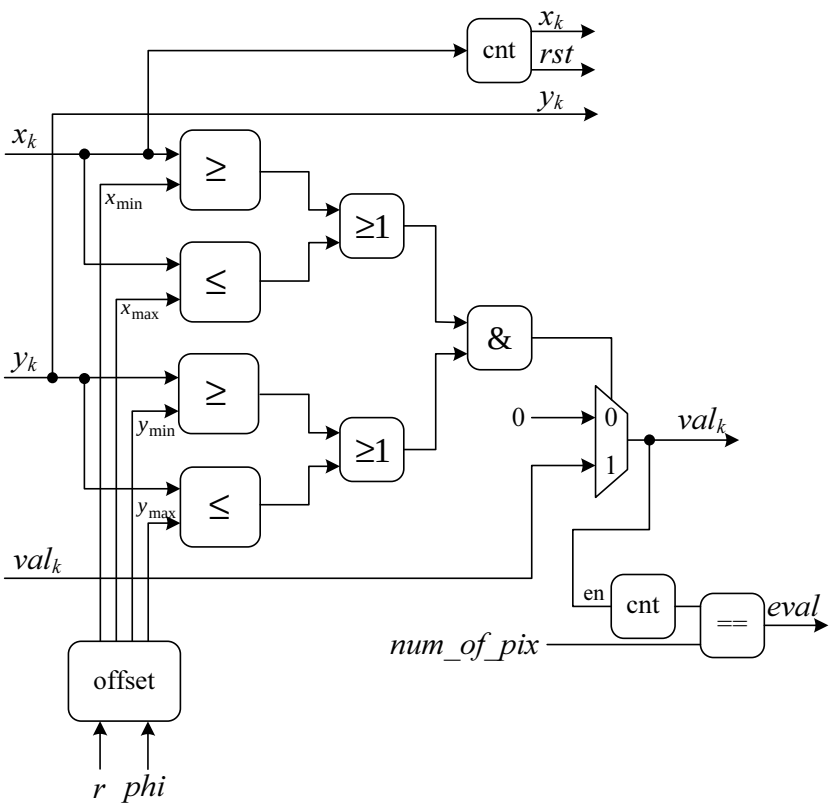

Abb. 69: *Schaltung zur Überprüfung eines projektiv transformierten Pixels auf Zugehörigkeit zur Region of Interest*

- Überprüfung der projektiv transformierten Koordinate (x_k, y_k) auf Zugehörigkeit zur *Region of Interest*:

$$\left[x_{min} \leq x_k \leq x_{max}, y_{min} \leq y_k \leq y_{max}\right]$$

- Bei Zugehörigkeit zur *Region of Interest* wird der Pegel des Eingangssignals val_k auf den Ausgangsport geschrieben. Gehört die Koordinate nicht zur *Region of Interest*, wird ein *low*-Pegel auf den Ausgangsport geschrieben, der eine *write enable* Generierung verhindert (vgl. Kap. 7.2.1)

- Alle eingehenden Pixelkoordinaten in x-Richtung x_k werden gezählt, um ein *reset*-Signal (vgl. Kap. 7.2.5) bei Erreichen eines Vordefinierten Zählstandes zu generieren.

- Offsetkorrektur für $x_{min}, x_{max}, y_{min}$ und y_{max} über eine Rückführung der Geradenparameter r und phi

- Eine Generierung des $eval$-Signals zur *select*-Signal Erzeugung (vgl. Kap. 7.2.8) für die Ermittlung des globalen Maxiumums (vgl. Kap. 7.2.7) wird über ein Zählmechanismus realisiert, der die Vordergrundpixel, die in der *Region of Interest* liegen, zählt und mit der Konstanten num_of_pix vergleicht.

C Kinematisches Einspurfahrzeugmodell

Liegen keine verlässlichen Sensordaten vor, wird die Position näherungsweise über ein kinematischen Einspurfahrzeugmodell bestimmt. Dieses basiert auf der Geschwindigkeit v und dem Lenkwinkel α, die zur Berechnung des Achswinkels θ, der Vorderradposition (x_m, y_m) des beweglichen Vorderrades M und der Hinterradposition (x_p, y_p) des starren Hinterrades P über Integralgleichungen verwendet werden.
Folgender Abschnitt behandelt:

- eine Herleitung des Differenzengleichungen des Einspurfahrzeugmodells mit der Trapezintegration
- ein Vergleich der numerischen Integration mit dem Runge-Kutta-Verfahren und der Trapezintegration, wodurch eine anschließende RTL-Modellierung der Trapezintegratoren gerechtfertigt wird
- ein Vergleich der Simulationergebnisse der Trapezintegration in Matlab mit den VHDL-Simulationsergebnissen, die aufgrund einer Fixedpoint-Arithmetik-Realisierung im Q-Format eine geringere Genauigkeit aufweisen

C.1 Differenzengleichungen des Einspurfahrzeugmodells

Das kinematische Einspurfahrzeugmodell wird durch folgende Integralgleichungen beschrieben [28]:

$$\Theta = \frac{1}{L} \cdot \int v_m(t) \cdot sin(-\alpha) dt \tag{38}$$

$$x_m = \int v_m(t) \cdot cos(\Theta - \alpha) dt \tag{39}$$

$$y_m = \int v_m(t) \cdot sin(\Theta - \alpha) dt \tag{40}$$

$$x_p = \int \underbrace{v_m(t) \cdot cos(\alpha)}_{v_p(t)} \cdot cos(\Theta) dt \tag{41}$$

$$y_p = \int \underbrace{v_m(t) \cdot cos(\alpha)}_{v_p(t)} \cdot sin(\Theta) dt \tag{42}$$

Die Hinterradgeschwindigkeit v_p eines Fahrzeuges unterscheidet sich im Falle $cos(\alpha) \neq 1$ von der Vorderradgeschwindigkeit v_m. Die Winkelgeschwindigkeit ω ist an Vorder- und Hinterrad identisch (vgl. Gl. 43).

$$\omega = \frac{v}{r} = \frac{v_m}{r_m} = \frac{v_p}{r_p} \Rightarrow v_p = v_m \cdot \frac{r_p}{r_m} \Rightarrow v_p = cos(\alpha) \cdot v_m \tag{43}$$

Die Gerade g_m, die orthogonal auf der Längsachse des beweglichen Vorderrades M steht und eine Länge r_m besitzt und die Gerade g_p, die orthogonal auf der Längsachse des starren Hinterrades P steht und eine Länge r_p besitzt, schneiden sich im Beobachtungspunkt Z in einem Winkel α, der dem Lenkwinkel entspricht und den Winkel zwischen beweglichem Vorderrad M und Fahrzeugachse beschreibt (vgl. Abb. 70). Es handelt sich bei dem Lenkwinkel α um eine nicht messbare Größe, die über das Einspeisen eines Soll-Wertes eingestellt wird.

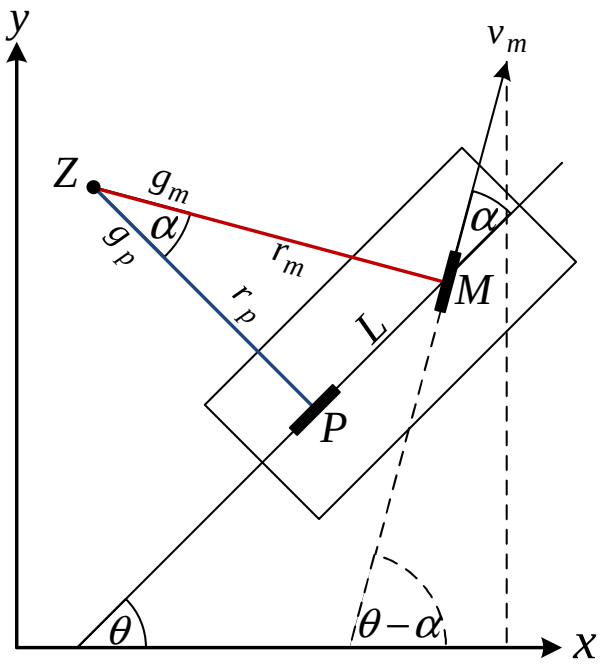

Abb. 70: *Einspurfahrzeugmodell mit gelenktem Vorderrad M und messbarer Geschwindigkeit v_m*

Trapezintegration für den Achswinkel Θ

Die analogen Integralgleichungen 38, 39, 40, 41 und 42 lassen sich in einem digitalen System mit diskreten Abtastperioden nicht berechnen, da die Gleichungen auf zeitkontinuierlichen Werten basieren. Die zeitkontinuierlichen Gleichungen müssen in den zeitdiskreten Bereich transformiert werden.

Das zeitdiskrete Verhalten ist charakterisiert durch eine nicht variable Abtastrate. Verfahren mit variabler Schrittweite, wie das Runge-Kutta-Verfahren [1], sind daher ungeeignet. Ein Verfahren mit konstanter Schrittweite ist die Trapezintegration, die im Folgenden zur Anwendung kommt (vgl. Abb. 71).

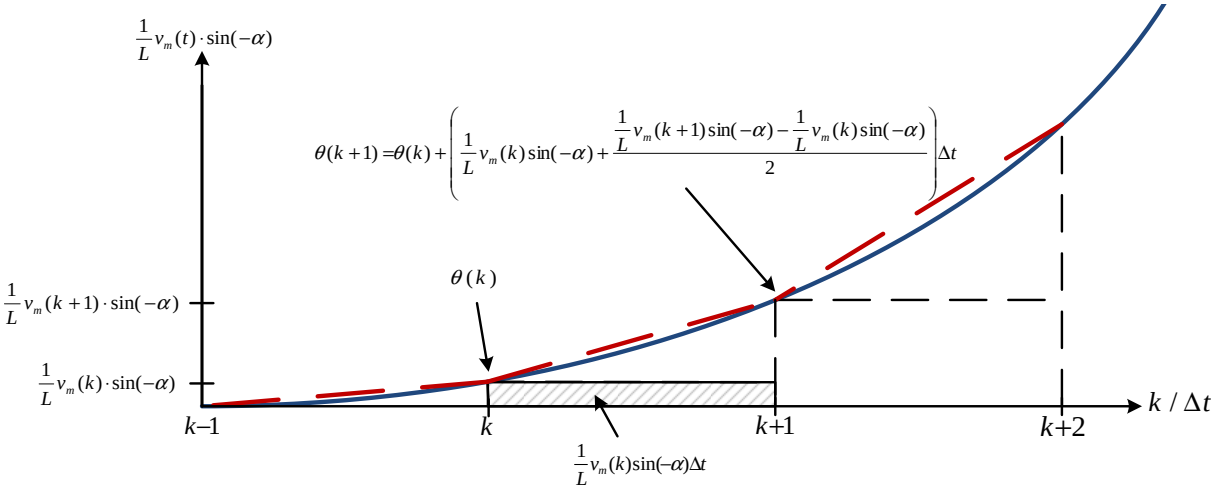

Abb. 71: *Prinzip der Trapezintegration für den Achswinkel Θ mit konstanter Schrittweite*

Zum Zeitpunkt k liegt das Ergebnis $\Theta(k)$ vor (vgl. Abb. 71). Das Folgeergebnis zum Zeitpunkt $k+1$ ergibt sich aus:

- dem vorherigen Achswinkel $\Theta(k)$
- der Fläche, die sich aus der Multiplikation des Funktionswertes $\frac{1}{L}v_m(k)sin(-\alpha)$ mit dem Abtastintervall Δt ergibt
- der Fläche, die einer gemittelten Dreiecksfläche zwischen $\frac{1}{L}v_m(k+1)sin(-\alpha)$ und $\frac{1}{L}v_m(k)sin(-\alpha)$ entspricht:

$$\frac{\frac{1}{L}v_m(k+1)sin(-\alpha) - \frac{1}{L}v_m(k)sin(-\alpha)}{2}\Delta t$$

Die Transformation des Achswinkels mit Trapezintegration ergibt folgende diskrete Form:

$$\begin{aligned}
\Theta(k+1) =& \Theta(k) \\
&+ \frac{1}{L}v_m(k)sin(-\alpha(k))\Delta t \\
&+ \frac{\frac{1}{L}v_m(k+1)sin(-\alpha(k+1)) - \frac{1}{L}v_m(k)sin(-\alpha(k))}{2}\Delta t \\
=& \Theta(k) \\
&+ \frac{1}{L}\Big[v_m(k)sin(-\alpha(k))\Delta t \\
&+ \frac{1}{2}v_m(k+1)sin(-\alpha(k+1))\Delta t - \frac{1}{2}v_m(k)sin(-\alpha(k))\Delta t\Big] \\
=& \Theta(k) \\
&+ \frac{1}{L}\Big[\frac{1}{2}v_m(k)sin(-\alpha(k))\Delta t \\
&+ \frac{1}{2}v_m(k+1)sin(-\alpha(k+1))\Delta t\Big] \\
=& \Theta(k) \\
&+ \frac{\Delta t}{2L}\Big[v_m(k)sin(-\alpha(k)) \\
&+ v_m(k+1)sin(-\alpha(k+1))\Big]
\end{aligned} \tag{44}$$

Diskrete Transformation für die Vorderradkoordinaten x_m und y_m

Die Koordinaten des beweglichen Vorderrades M im kinematischen Einspurfahrzeugmodell werden in x- und y-Richtung beschrieben. Die Gleichungen 39 und 40 werden mit der Trapezintegration in den zeitdiskreten Bereich transformiert.

$$\begin{aligned} x_m(k+1) =& x_m(k) \\ &+ v_m(k)cos(\Theta(k) - \alpha(k))\Delta t \\ &+ \frac{v_m(k+1)cos(\Theta(k+1) - \alpha(k+1)) - v_m(k)cos(\Theta(k) - \alpha(k))}{2}\Delta t \\ =& x_m(k) \\ &+ \frac{\Delta t}{2}\Big[v_m(k)cos(\Theta(k) - \alpha(k)) \\ &+ v_m(k+1)cos(\Theta(k+1) - \alpha(k+1))\Big] \end{aligned} \tag{45}$$

$$\begin{aligned} y_m(k+1) =& y_m(k) \\ &+ v_m(k)sin(\Theta(k) - \alpha(k))\Delta t \\ &+ \frac{v_m(k+1)sin(\Theta(k+1) - \alpha(k+1)) - v_m(k)sin(\Theta(k) - \alpha(k))}{2}\Delta t \\ =& y_m(k) \\ &+ \frac{\Delta t}{2}\Big[v_m(k)sin(\Theta(k) - \alpha(k)) \\ &+ v_m(k+1)sin(\Theta(k+1) - \alpha(k+1))\Big] \end{aligned} \tag{46}$$

Diskrete Transformation für die Hinterradkoordinaten x_p und y_p

Die Position des Hinterrades P wird durch die Koordinaten x_p und y_p beschrieben. Die zeitdiskreten Gleichungen werden hierfür unter Verwendung der zeitkontinuierlichen Gleichungen 41 und 42 hergeleitet.

$$\begin{aligned} x_p(k+1) =& x_p(k) \\ &+ v_m(k)cos(\alpha(k))cos(\Theta(k))\Delta t \\ &+ \frac{v_m(k+1)cos(\alpha(k+1))cos(\Theta(k+1)) - v_m(k)cos(\alpha(k))cos(\Theta(k))}{2}\Delta t \\ =& x_p(k) \\ &+ \frac{\Delta t}{2}\Big[v_m(k)cos(\alpha(k))cos(\Theta(k)) \\ &+ v_m(k+1)cos(\alpha(k+1))cos(\Theta(k+1))\Big] \end{aligned} \tag{47}$$

$$
\begin{aligned}
y_p(k+1) =& y_p(k) \\
&+ v_m(k)cos(\alpha(k))sin(\Theta(k))\Delta t \\
&+ \frac{v_m(k+1)cos(\alpha(k+1))sin(\Theta(k+1)) - v_m(k)cos(\alpha(k))sin(\Theta(k))}{2}\Delta t \\
=& y_p(k) \\
&+ \frac{\Delta t}{2}\Big[v_m(k)cos(\alpha(k))sin(\Theta(k)) \\
&+ v_m(k+1)cos(\alpha(k+1))sin(\Theta(k+1))\Big]
\end{aligned}
\tag{48}
$$

Bestimmung der Lenkverzögerung

Für eine genauere Beschreibung des Systems ist die Lenkverzögerung zu ermitteln. Diese beschreibt die Dauer, die vom Einstellen eines Soll-Lenkwinkel bis zum Erreichen dieses Soll-Lenkwinkels vergeht. Es handelt sich um ein Verzögerungsglied erster Ordnung (PT_1) [28]. Zur Bestimmung der Zeitkonstante T_1 wird ein Näherungswert über Durchführung von Messreihen ermittelt. Da diese Messreihen für das SoC-Fahrzeug noch ausstehen, wird im Folgenden der ermittelte Wert aus [28] übernommen, der für eine Lenkwinkeländerung von $0°$ auf $30°$ ca. 250 ms beträgt.
Durch Ersetzen der Laplace Transformierten s in der zeitkontinuierlichen Gleichung 49 durch die Tustin Formel 50 erhält man die diskrete Übertragsfunktion der Lenkwinkelverzögerung zur zeitdiskreten Simulation des Ausweichvorganges.

$$G(s) = \frac{K_p}{T_1 s + 1} = \frac{\alpha_{ist}}{\alpha_{soll}} \tag{49}$$

$$s \approx \frac{2}{T}\frac{z-1}{z+1} \tag{50}$$

$$
\begin{aligned}
G(s) &= \frac{K_p}{T_1 s + 1} \\
&\approx \frac{1}{T_1 \cdot \frac{2}{T}\frac{z-1}{z+1} + 1} \\
&\approx \frac{T(z+1)}{T_1 \cdot 2 \cdot (z-1) + (z+1) \cdot T} \\
&\approx \frac{Tz + T}{2 \cdot T_1 z - 2 \cdot T_1 + Tz + T} \\
&\approx \frac{Tz + T}{(2T_1 + T) \cdot z + (T - 2T_1)} = \frac{\alpha_{ist}(z)}{\alpha_{soll}(z)}
\end{aligned}
\tag{51}
$$

Das PT_1-Verzögerungsglied liefert ein Ergebnis in folgender Form:

$$\alpha_{ist}(k) = f\big(\alpha_{ist}(k-1), \alpha_{soll}(k-1)\big)$$

Im Frequenzbereich lässt sich Gleichung 51 wie folgt umformen:

$$\alpha_{ist}(z) \cdot \Big(\underbrace{(2T_1 + T)}_{a_0} z + \underbrace{(T - 2T1)}_{a_1} \Big) = \alpha_{soll}(z) \cdot (\underbrace{T}_{b_1} + \underbrace{T}_{b_0} z)$$

$$\alpha_{ist}(z) \cdot \big(a_0 z + a_1\big) = \alpha_{soll}(z) \cdot (b_0 + b_1 z)$$

Verschub um T_a (Abtastfrequenz)

$$\alpha_{ist}(z) \cdot \big(a_0 z + a_1\big) \cdot z^{-1} = \alpha_{soll}(z) \cdot (b_0 + b_1 z) \cdot z^{-1}$$
$$\alpha_{ist}(z) \cdot \big(a_0 + a_1 z^{-1}\big) = \alpha_{soll}(z) \cdot (b_0 z^{-1} + b_1)$$
$$\alpha_{ist}(z) a_0 + \alpha_{ist}(z) a_1 z^{-1} = \alpha_{soll}(z) b_0 z^{-1} + \alpha_{soll}(z) b_1$$

$$\alpha_{ist}(z) = \alpha_{soll}(z) \frac{b_0}{a_0} z^{-1} + \alpha_{soll}(z) \frac{b_1}{a_0} - \alpha_{ist}(z) \frac{a_1}{a_0} z^{-1}$$

Transformation in den Zeitbereich:

$$\underbrace{\alpha_{ist}(k)}_{aktuellerWert} = \underbrace{\alpha_{soll}(k-1)}_{vorherigerWert} \frac{b_0}{a_0} + \underbrace{\alpha_{soll}(k)}_{aktuellerWert} \frac{b_1}{a_0} - \underbrace{\alpha_{ist}(k-1)}_{vorherigerWert} \frac{a_1}{a_0}$$

Rücksubstituieren der Konstanten:

$$\alpha_{ist}(k) = \alpha_{soll}(k-1) \frac{T}{2T_1 + T} + \alpha_{soll}(k) \frac{T}{2T_1 + T} - \alpha_{ist}(k-1) \frac{T - 2T_1}{2T_1 + T}$$

T entspricht der Abtastfrequenz des Systems Δt

$$\alpha_{ist}(k) = \alpha_{soll}(k-1) \frac{\Delta t}{2T_1 + \Delta t} + \alpha_{soll}(k) \frac{\Delta t}{2T_1 + \Delta t} - \alpha_{ist}(k-1) \frac{\Delta t - 2T_1}{2T_1 + \Delta t}$$

$$\alpha_{ist}(k) = \Big(\alpha_{soll}(k-1) + \alpha_{soll}(k) \Big) \frac{\Delta t}{2T_1 + \Delta t} - \alpha_{ist}(k-1) \frac{\Delta t - 2T_1}{2T_1 + \Delta t} \tag{52}$$

C.2 Vergleich zwischen numerischer Integration und Trapezintegration

Eine gewöhnliche Differentialgleichung lässt sich mathematisch mit dem Zustandsvektor x, dem Zeitargument t, einer vektorwertigen Funktion f und dem Anfangszustand x_0 zum Zeitpunkt t_0 als Differentialgleichung erster Ordnung darstellen [1].

$$\dot{x} = f(t, x) \qquad x(t_0) = x_0 \tag{53}$$

Eine Differentialgleichung $\tilde{x}^{(n)} = f\big(t, \tilde{x}, \dot{\tilde{x}}, \ldots, \tilde{x}^{(n-1)}\big)$ n-ter Ordnung wird durch Substitutionen $x_1 = \tilde{x}, x_2 = \dot{\tilde{x}}$ usw. in ein System mit n Differentialgleichungen erster Ordnung umgewandelt werden.

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ &\vdots \\ \dot{x}_{n-1} &= x_n \\ \dot{x}_n &= f(t, x_1, x_2, \dots x_n) \end{aligned} \tag{54}$$

Matlab bietet verschiedene Solver zur Lösung von Differentialgleichungen, basierend auf Integrationsalgorithmen. Der Solver $ode45$ nutzt das Runge-Kutta-Verfahren mit automatischer Schrittweitensteuerung [13], um Lösungen für ein System von Differentialgleichungen erster Ordnung zu berechnen (vgl. Listing 3 und 4).

```
N = 50;                     % Anzahl Iterationen
dt = 0.02;                  % Abtastperiode in s
L = 0.254;                  % Achsabstand SOC-Fahrzeug
vm = 1;                     % Vorderradgeschwindigkeit in m/s
fhzg_init = [0;L;0;0;0];    % Startzustandsvektor [theta, xm, ym, xp, yp]
alpha = pi*(20)/180;        % Lenkwinkel in Grad

[t, fhzg_state] = ode45(@fhzg_modell, [0:dt:N*dt], fhzg_init, [], vm, alpha, L);
fhzg_state_trapez = fhzg_modell_trapez([0:dt:N*dt], vm, alpha, dt, L, N);
```

Listing 3: *Initialisierung und Aufruf von ode45 und Trapezintegration für das kinematische Einspurfahrzeugmodell in Matlab mit konstantem Lenkwinkel α und konstanter Geschwindigkeit v*

```
function fhzg_modell_dot = fhzg_modell(t, fhzg_state, vm, alpha, L)

fhzg_modell_dot = zeros(5, 1);      % [0;0;0;0;0] Spaltenvektor

% Vorderradgeschwindigkeit vm und Lenkwinkel alpha vorberechnet oder
% konstant

% 1. Gleichung für Theta
fhzg_modell_dot(1,1) = 1/L*vm*sin(-alpha);
% 2. Gleichung für xm
fhzg_modell_dot(2,1) = vm*cos(fhzg_state(1,1) - alpha);
% 3. Gleichung für ym
fhzg_modell_dot(3,1) = vm*sin(fhzg_state(1,1) - alpha);
% 4. Gleichung für xp
fhzg_modell_dot(4,1) = vm*cos(alpha)*cos(fhzg_state(1,1));
% 5. Gleichung für yp
fhzg_modell_dot(5,1) = vm*cos(alpha)*sin(fhzg_state(1,1));

end
```

Listing 4: *Matlab Funktion die von ode45 aufgerufen wird und die Ableitungen zum aktuellen Lösungspunkt berechnet*

Neben der Integration mit dem Runge-Kutta-Verfahren wird eine Trapezintegration in Matlab durchgeführt, um einen Vergleich der Verfahren durchführen zu können und die Korrektheit der Differenzengleichungen nachweisen zu können.

```
function fhzg_modell = fhzg_modell_trapez(t, vm, alpha, dt, L, N)

theta   = zeros(1, N+1)';
xm      = zeros(1, N+1)';
ym      = zeros(1, N+1)';
xp      = zeros(1, N+1)';
yp      = zeros(1, N+1)';

xm(1) = L;

for k=1:N
    theta(k+1) = theta(k)-dt/(2*L)*vm*sin(alpha)-dt/(2*L)*vm*sin(alpha);
    xm(k+1) = xm(k) + dt/2*vm*cos(theta(k)-alpha) + dt/2*vm*cos(theta(k+1)-alpha);
    ym(k+1) = ym(k) + dt/2*vm*sin(theta(k)-alpha) + dt/2*vm*sin(theta(k+1)-alpha);
    xp(k+1) = xp(k) + dt/2*vm*cos(alpha)*cos(theta(k)) + dt/2*vm*cos(alpha)*cos(theta(k+1));
    yp(k+1) = yp(k) + dt/2*vm*cos(alpha)*sin(theta(k)) + dt/2*vm*cos(alpha)*sin(theta(k+1));
end

fhzg_modell = [theta xm ym xp yp];

end
```

Listing 5: *Iterative Berechnung der Trapezintegration des kinematischen Einspurfahrzeugmodells mit den hergeleiteten Gleichungen 44, 45, 46, 47 und 48*

Die Simulation wird mit konstantem Lenkwinkel $\alpha = 20°$ und konstanter Geschwindigkeit $v = 1$ m/s für eine Simulationsdauer $T = 1$ s mit $\Delta t = 20$ ms durchgeführt. Die Abweichungen zwischen numerischer Integration mit dem Runge-Kutta-Verfahren und Trapezintegration, die in Abbildung 73, 75, 77, 79 und 81 dargestellt sind, bestätigen die Korrektheit der hergeleiteten Differenzengleichungen 44, 45, 46, 47 und 48 und rechtfertigen eine Realisierung der Trapezintegration in einer VHDL Implementierung.

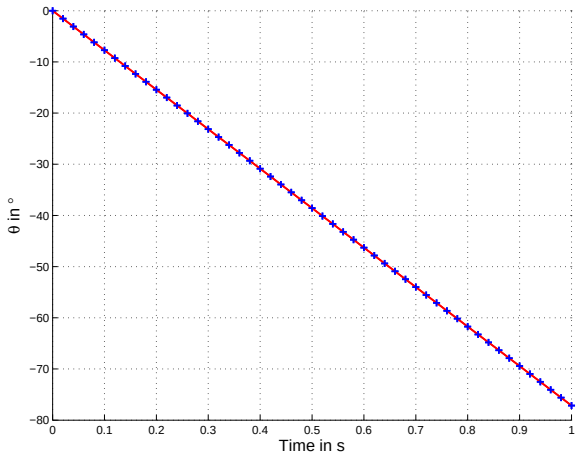

Abb. 72: *Numerische Integration (rot) und Trapezintegration (blaue Kreuze) des Achswinkels θ*

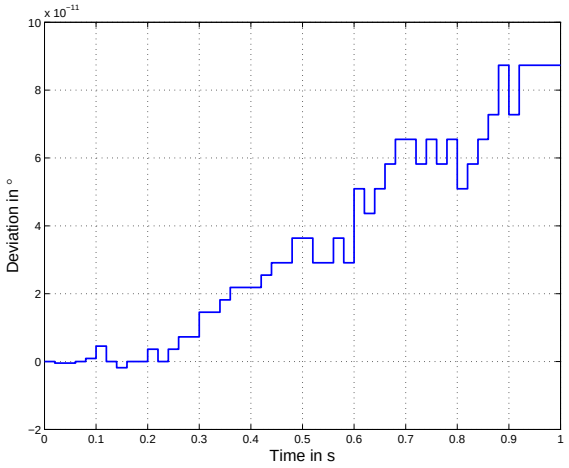

Abb. 73: *Absolute Abweichung zwischen numerischer Integration und Trapezintegration des Achswinkels θ in Grad*

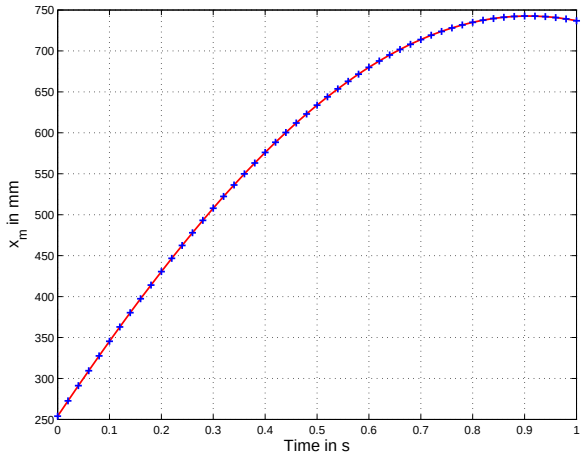

Abb. 74: *Numerische Integration (rot) und Trapezintegration (blaue Kreuze) des Vorderrades M in x-Richtung x_m*

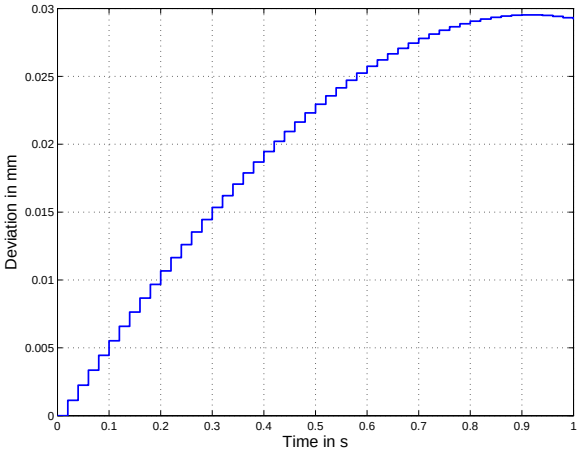

Abb. 75: *Absolute Abweichung zwischen numerischer Integration und Trapezintegration der Vorderradposition x_m in mm*

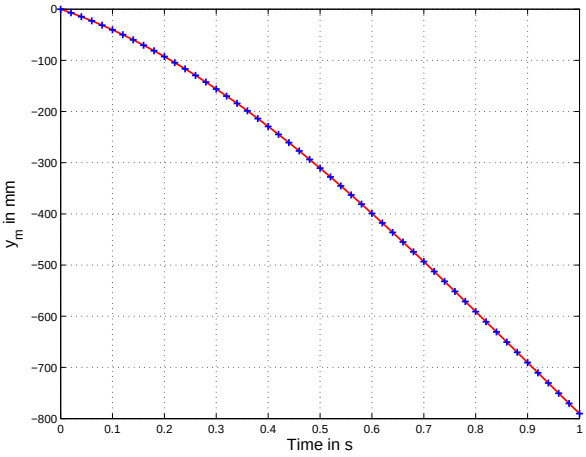

Abb. 76: *Numerische Integration (rot) und Trapezintegration (blaue Kreuze) des Vorderrades M in y-Richtung y_m*

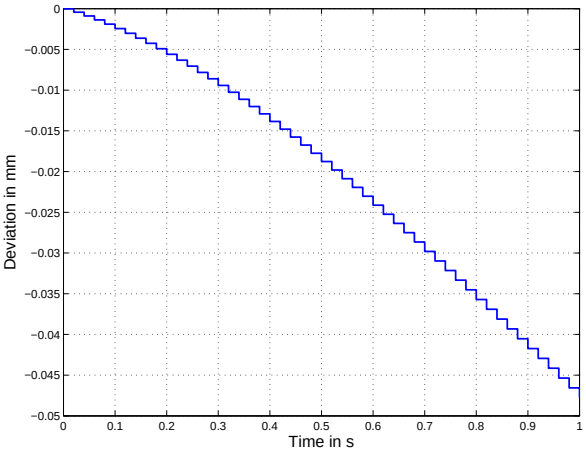

Abb. 77: *Absolute Abweichung zwischen numerischer Integration und Trapezintegration der Vorderradposition y_m in mm*

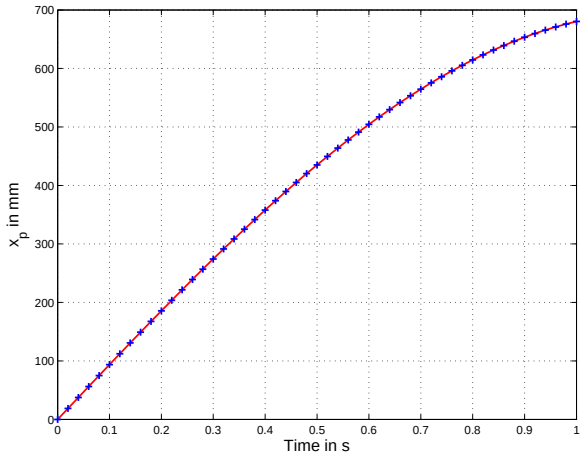

Abb. 78: *Numerische Integration (rot) und Trapezintegration (blaue Kreuze) des Hinterrades P in x-Richtung x_p*

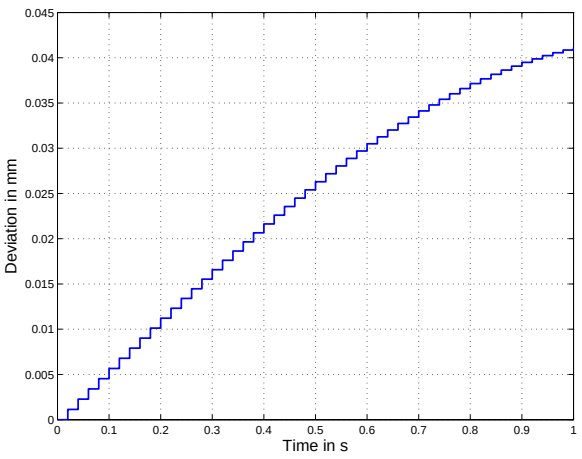

Abb. 79: *Absolute Abweichung zwischen numerischer Integration und Trapezintegration der Hinterradposition x_p in mm*

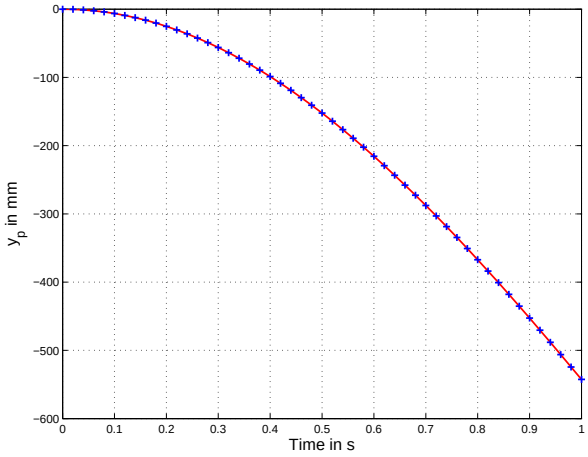

Abb. 80: *Numerische Integration (rot) und Trapezintegration (blaue Kreuze) des Hinterrades P in y-Richtung y_p*

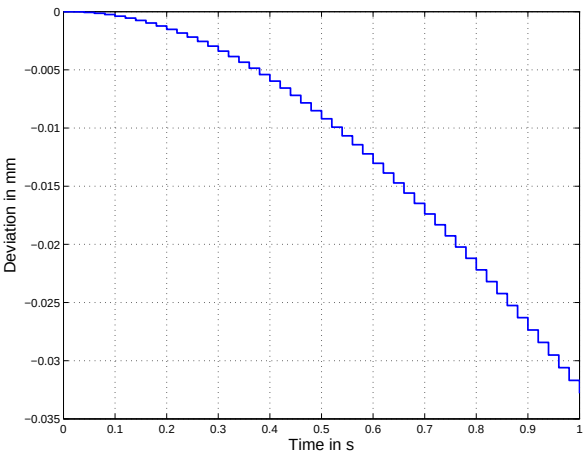

Abb. 81: *Absolute Abweichung zwischen numerischer Integration und Trapezintegration der Hinterradposition y_p in mm*

C.3 RTL-Modelle und Diskussion des Vektorbreiten

Als Basis der Zahlendarstellung für die Modellierung der Trapezintegratoren in VHDL wird das auf Fixedpointarithmetik basierende Q-Format (vgl. Kapitel 2.2.2) gewählt. Ein Systementwurf unter Einsatz des Q-Formats erlaubt das anforderungs- und erfahrungsbasierte Dimensionieren der Vektorbreiten.
Eine Zahl im *ufix_a_b*-Format wird als *unsigned* interpretiert und beinhaltet a-Bits mit b-Bits rechts vom Binärpunkt. Eine Zahl im *fix_a_b*-Format wird als *signed* interpretiert und beinhaltet a-Bits mit b-Bits rechts vom Binärpunkt, wobei das MSB das Vorzeichenbit darstellt.

Ist-Lenkwinkelverzögerung α_{ist}

Das Ausgangssignal α_{ist} ist ein Vektor im fix_15_7-Format. Der als *signed* interpretierte 8 Bit breite Integeranteil erlaubt die Darstellung von Winkeln zwischen $-128°$ und $127°$, wobei der Lenkwinkel α Werte zwischen $-90°$ und $90°$ annimmt.
Bei einem Verzögerungsglied 1. Ordnung (PT_1-Glied) nähert sich die Ausgangsgröße nach einer sprungförmigen Änderung des Istwertes exponentiell mit einer bestimmten Zeitkonstanten T_1 asymptotisch an den Sollwert an [30]. Es kann aufgrund der asymptotischen Annäherung an den Sollwert α_{soll} nicht zu Überschwingern kommen, weshalb der Integeranteil des Istwertes α_{ist} mit dem die trigonometrischen LUTs adressiert werden nicht breiter dimensioniert wird als der Integeranteil des Sollwertes α_{soll}. Es handelt sich bei dem Soll-Lenkwinkel α_{soll} um eine nicht messbare Größe. Der Lenkmotor wird über eine Pulsweiten Modulation auf den Soll-Wert gestellt und nähert sich diesem mit dem durch das PT_1-Glied beschriebene Verhalten an. Zur Aufnahme eines zusätzlichen Carry-Bits wird der Ergebnisvektor der Addition von $\alpha_{soll}(k+1)$ und $\alpha_{soll}(k)$ ein Bit breiter dimensioniert als die Vektoren der Summanden (vgl. Kapitel 2.2.2).

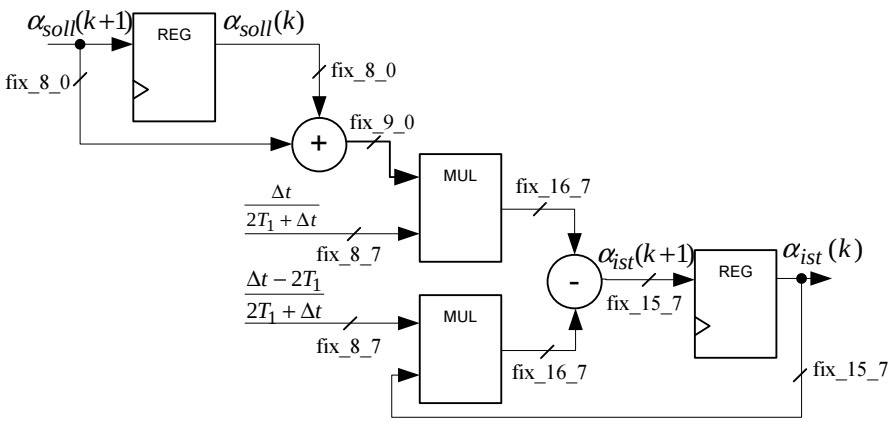

Abb. 82: *RTL-Modell der Ist-Lenkwinkelverzögerung (vgl. Gl. 52)*

Der Vektor $\frac{\Delta t}{2T1+\Delta t}$ enthält ein Vorzeichen- und sieben Nachkommabits. Aufgrund des Parameter des SOC-Fahrzeuges ergibt sich:

$$\frac{\Delta t}{2T1+\Delta t} = \frac{0.02s}{0.5s+0.02s} = \frac{1}{26} = 0.0384615$$

Binär wird die Konstante 0.0000101_2 erreicht, was dezimal 0.0390625_{10} entspricht. Der relative Fehler bei dieser Darstellung beträgt:

$$\left|\frac{(2^{-5}+2^{-7})-\frac{1}{26}}{\frac{1}{26}}\right| = \left|\frac{1}{64}\right| = 0.015625 \approx 1.5\%$$

Unter Berücksichtigung der Fahrzeugparameter ergibt sich für die im fix_8_7-Format gewählte Konstante $\frac{\Delta t - 2T1}{2T1+\Delta t}$:

$$\frac{0.02s - 2 \cdot 0.25s}{2 \cdot 0.25s + 0.02s} = -\frac{0.48}{0.52} = -0.923076923$$

Achswinkel Θ

Für den Achswinkel Θ wird ein Vektor im fix_24_16-Format gewählt, wobei 8 Bit den als *signed* interpretierten Integeranteil repräsentieren und 16 Bit für eine interne Speicherung von Nachkommastellen verwendet werden. Mit einem 8 Bit Integeranteil sind Werte von $-128°$ bis $127°$ darstellbar, wobei der Achswinkel Θ Werte zwischen $-90°$ und $90°$ annimmt. Die Verwendung von 16 Bit zur Speicherung von Dezimalstellen erhöht die Genauigkeit der Folgeberechnungen gegenüber Berechnungen, in denen Dezimalstellen ignoriert werden.

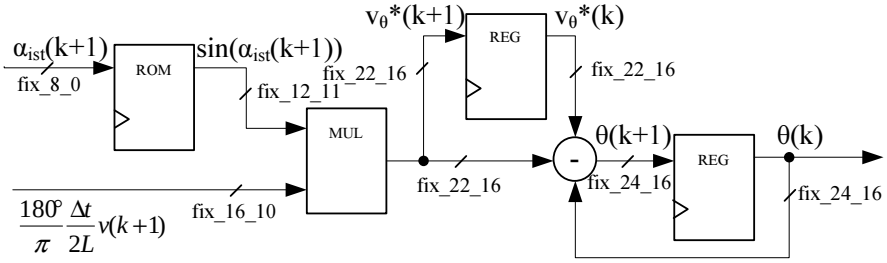

Abb. 83: *RTL-Modell zur Berechnung des Achswinkels Θ (vgl. Gl. 44)*

Das Ausgangssignal des Moduls zur Ist-Lenkwinkelbestimmung $\alpha_{ist}(k+1)$ geht ohne die intern zur Verbesserung der Rechengenauigkeit gespeicherten Fractional-Bits in die Adressbildung ein, da Adressen positiv ganzzahlig sind. Eine Realisierung positiver Adressen, auch bei einem negativen Lenkwinkel α_{ist}, wird über einen Adressoffset realisiert.
Die Konstante $\frac{180°}{\pi}\frac{\Delta t}{2L}v(k+1)$ entspricht einer dimensionslosen, gewichteten Geschwindigkeit v und wird als Vektor im fix_16_10-Format gewählt und kann so gewichtete Geschwindigkeiten von -32 bis $32 - 2^{-10}$ darstellen.
Das Signal v^*_Θ wird als Zahl im fix_22_16-Format gewählt. Eine Multiplikation von $sin(\alpha_{ist}(k+1)$ mit $\frac{180°}{\pi}\frac{\Delta t}{2L}v(k+1)$ resultiert in einem fix_28_21 Vektor, von dem nicht alle Fractional-Bits abgegriffen werden, um eine Vektorbreitenreduzierung und damit eine Ressourcenersparnis zu erreichen.

Vorderradposition x_m und Hinterradposition x_p

Die Ausgangssignale x_m, x_p, y_m und y_p sind als Zahlen im fix_25_13-Format gewählt. Unter Verwendung von 12 Integer-Bits und 13 Fractional-Bits werden Werte im Bereich von -2048 mm bis $2047 - 2^{-13}$ mm vom kinematischen Einspurfahrzeugmodell erfassbar. Werte außerhalb des Bereiches erzeugen einen Überlauf. Die Verwendung von 13 Fractional-Bits erhöht die Genauigkeit des Folgeergebnisses gegenüber der Verarbeitung ohne Nachkommastellen.

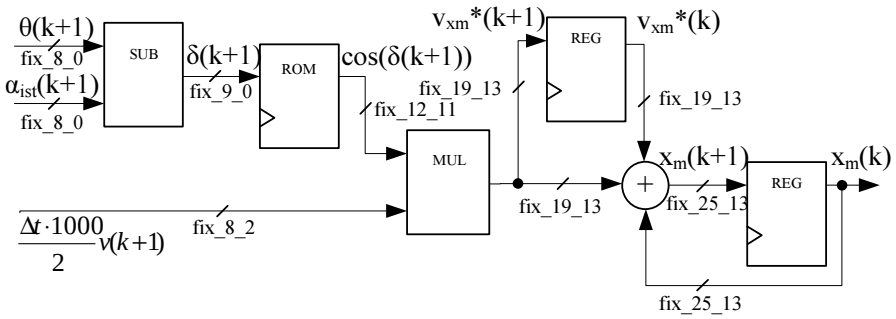

Abb. 84: *RTL-Modell zur Berechnung der Vorderradposition x_m (vgl. Gl. 45)*

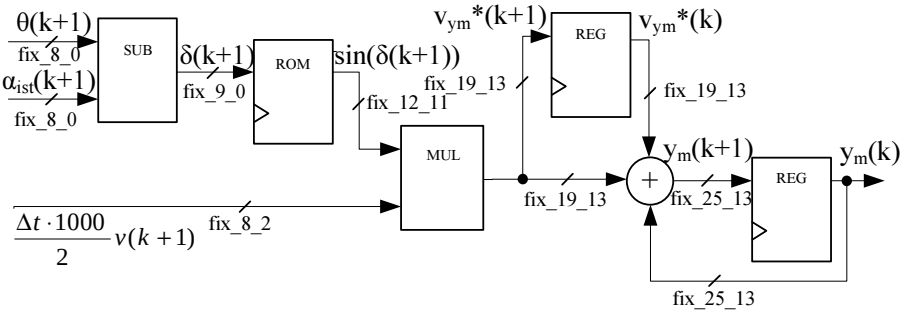

Abb. 85: *RTL-Modell zur Berechnung der Vorderradposition y_m (vgl. Gl. 46)*

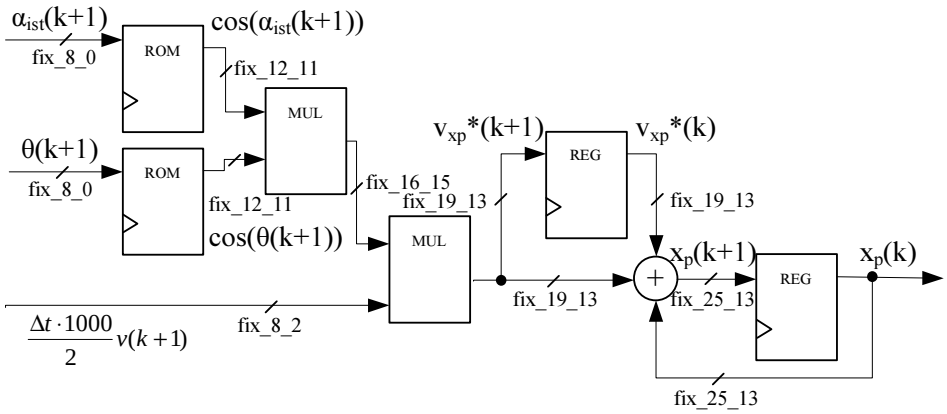

Abb. 86: *RTL-Modell zur Berechnung der Hinterradposition x_p (vgl. Gl. 47)*

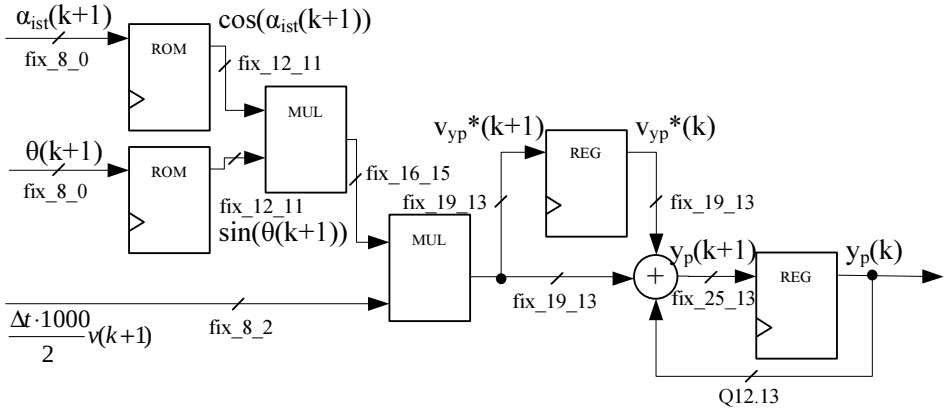

Abb. 87: *RTL-Modell zur Berechnung der Hinterradposition y_p (vgl. Gl. 48)*

Das als Zahl im fix_8_2-Format gewählte Signal $\frac{\Delta t \cdot 1000}{2} v(k+1)$ repräsentiert eine gewichtete Geschwindigkeit. Die Geschwindigkeit v, die das kinematische Einspurfahrzeugmodell aus dem IP v-Regler [3] bereitgestellt bekommt, besitzt die Einheit m/s. Durch eine Multiplikation der Geschwindigkeit v mit der Konstanten 1000 und eine Gewichtung mit der Abtastperiode Δt, wird die Geschwindigkeit v in den mm-Bereich überführt.

$$\left[\frac{\Delta t}{2} v(k+1) \cdot 1000\right] = mm \qquad \left[\frac{0.02}{2} v(k+1) \cdot 1000\right] = mm \qquad \left[10 v(k+1)\right] = mm$$

Die Höchstgeschwindigkeit des Fahrzeuges v_{max} beträgt $\pm 3m/s$, was die Dimensionierung der gewichteten Geschwindigkeit $\frac{\Delta t \cdot 1000}{2} v(k+1)$ als Vektor im fix_8_2-Format rechtfertigt, da $v_{max} \cdot 10 \leq 31$ bzw. $v_{max} \cdot 10 \geq -32$ ist.
Die Herleitung der Vektorbreiten der Multiplikationen basieren auf den Regeln der binären Multiplikation im Q-Format (vgl. Kapitel 2.2.2), wobei Nachkommaanteile binärer Multiplikationen nicht vollständig abgegriffen werden, da sich so eine Ressourcenersparnis ergibt.

C.4 Vergleich der Trapezintegration und der VHDL Simulation

Ein Vergleich zwischen den VHDL-Simulationsergebnissen und den Matlab-Simulationsergebnissen der Trapezintegration wird im Folgenden für den Achswinkel θ, Vorderrad M in x-Richtung x_m und y-Richtung y_m, sowie Hinterrad P in x-Richtung x_p und y-Richtung y_p durchgeführt. Basis für die VHDL-Simulation sind die in Abschnitt C.3 vorgestellten RTL-Modelle und Vektorbreiten. Die VHDL-Simulationsergebnisse werden aus einer Testbench exportiert und stehen für eine Verarbeitung in Matlab zur Verfügung.

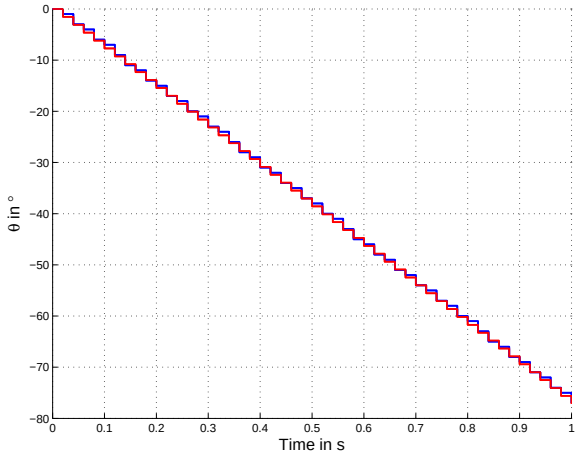

Abb. 88: *Trapezintegration (rot) und VHDL-Simulation (blau) des Achswinkels θ*

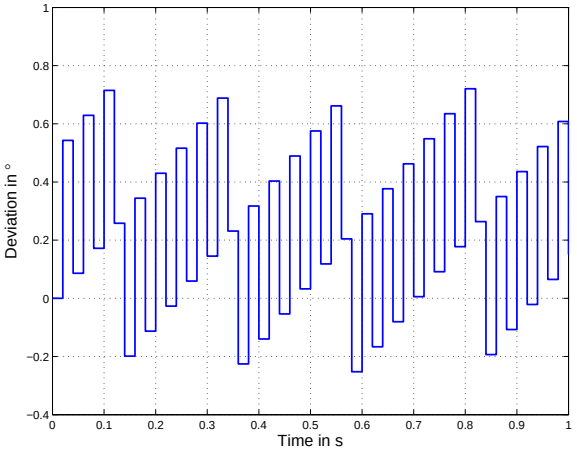

Abb. 89: *Absolute Abweichung zwischen Trapezintegration und VHDL-Simulation des Achswinkels θ in Grad*

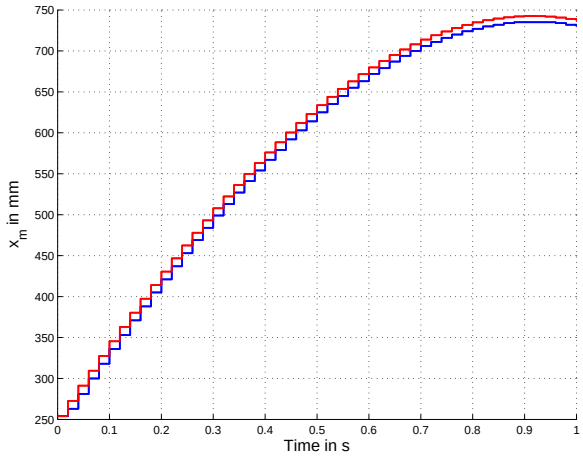

Abb. 90: *Trapezintegration (rot) und VHDL-Simulation (blau) des Vorderrades M in x-Richtung x_m*

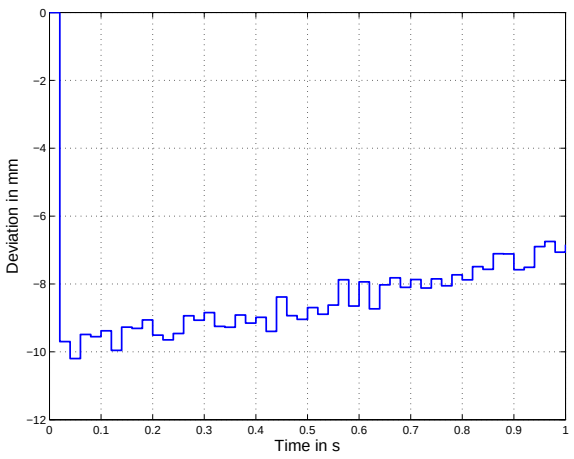

Abb. 91: *Absolute Abweichung zwischen Trapezintegration und VHDL-Simulation der Vorderradposition x_m in mm*

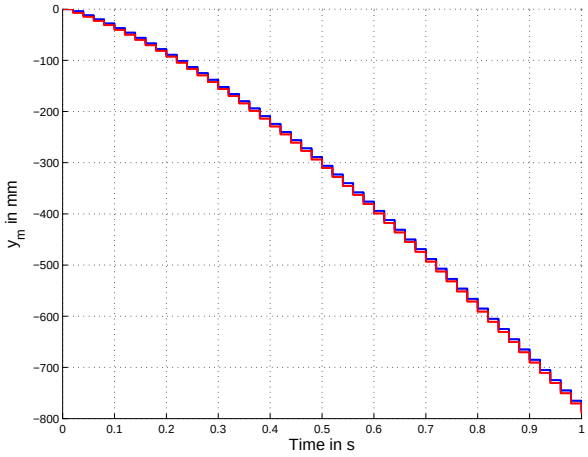

Abb. 92: *Trapezintegration (rot) und VHDL-Simulation (blau) des Vorderrades M in y-Richtung y_m*

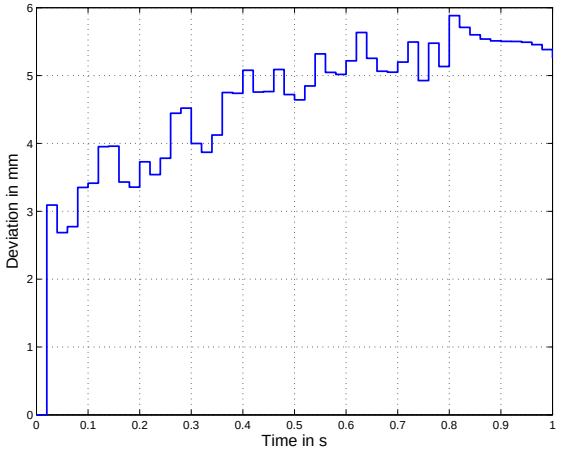

Abb. 93: *Absolute Abweichung zwischen Trapezintegration und VHDL-Simulation der Vorderradposition y_m in mm*

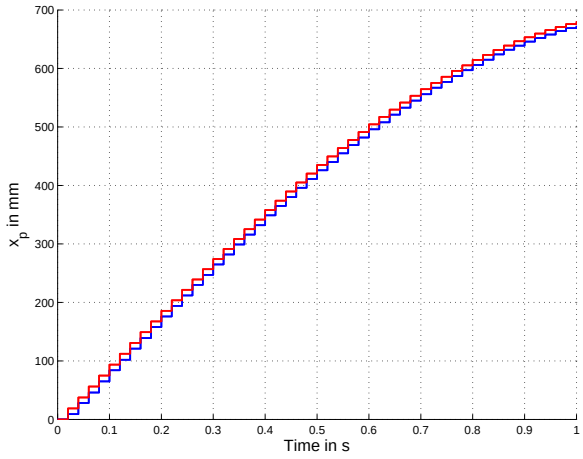

Abb. 94: *Trapezintegration (rot) und VHDL-Simulation (blau) des Hinterrades P in x-Richtung x_p*

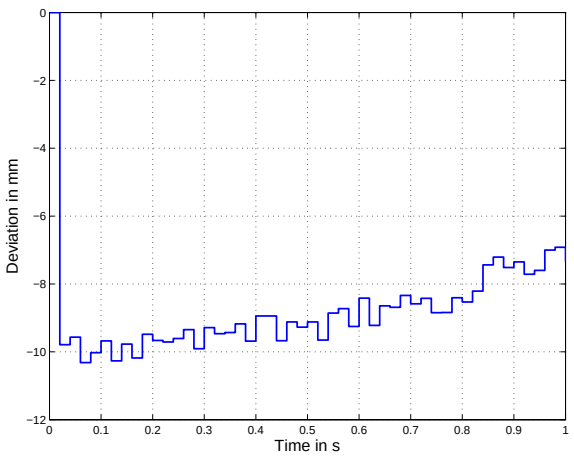

Abb. 95: *Absolute Abweichung zwischen Trapezintegration und VHDL-Simulation der Hinterradposition x_p in mm*

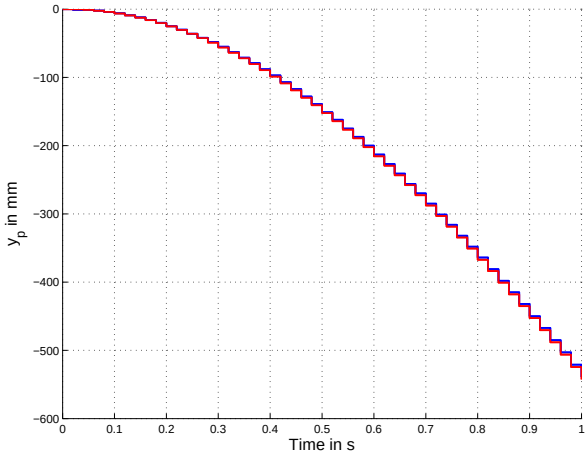

Abb. 96: *Trapezintegration (rot) und VHDL-Simulation (blau) des Hinterrades P in y-Richtung y_p*

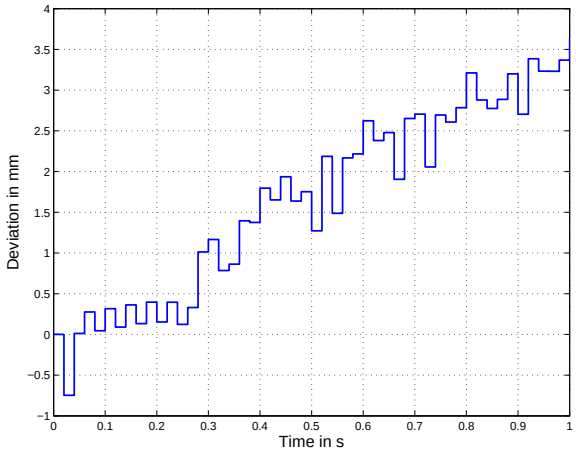

Abb. 97: *Absolute Abweichung zwischen Trapezintegration und VHDL-Simulation der Hinterradposition y_p in mm*

D CD: System Generator Projektdateien, Matlab- und VHDL Code

Versicherung über Selbstständigkeit

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

Hamburg, den 20. April 2010	
Ort, Datum	Unterschrift