Bachelorarbeit

Alexander Timoschenko

Implementierung einer Geschwindigkeitsregelung
als Prozessor-Element auf einer SoC-Plattform
für ein autonomes Fahrzeug

Alexander Timoschenko

Implementierung einer Geschwindigkeitsregelung als Prozessor-Element auf einer SoC-Plattform für ein autonomes Fahrzeug

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

Betreuender Prüfer: Prof. Dr.-Ing. Bernd Schwarz
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Abgegeben am 23.11.2011

Alexander Timoschenko

Thema der Bachelorarbeit

Implementierung einer Geschwindigkeitsregelung als Prozessor-Element auf einer SoC-Plattform für ein autonomes Fahrzeug

Stichworte

Geschwindigkeitsregelung, System-on-Chip, MicroBlaze, FPGA, PI-Regler, Hall-Sensor-Auswertung, Division by Trial Subtraction, Wendetangenten-Verfahren, T-Summen-Verfahren, RTL-Modellierung mit VHDL

Kurzzusammenfassung

Diese Arbeit berichtet über die Entwicklung einer aus VHDL-Modulen realisierten Geschwindigkeitsregelung für den Einsatz in ein autonomes Fahrzeug mit einer Spartan-3E FPGA-basierten SoC-Plattform. Die Geschwindigkeit des Fahrzeugs wird durch die Auswertung der Motor-Hall-Sensoren ermittelt, dazu wurden ein „Division by Trial Subtraction“-Algorithmus und eine Periodendauermessung als RTL modelliert. Der als eine FSM mit Datenpfad implementierte PI-Regler gekoppelt, mit einem PWM-Generator erzeugen PWM-Sequenzen. Eine Variation des PWM-Tastverhältnisses führt zur Motordrehzahländerung und somit zur Geschwindigkeitsmanipulation. Integriert in ein MicroBlaze Entwurf ist die Geschwindigkeitsregelung eine Komponente des SoC-basierten eingebetteten Systems.

Alexander Timoschenko

Title of the paper

Implementation of a speed control for an autonomous vehicle as a processor element on a SoC

Keywords

Speed control system, System-on-Chip, MicroBlaze, FPGA, PI-controller, hall-sensor evaluation, division by trial subtraction, inflexion tangent method, T-sum method, RTL modeling with VHDL

Abstract

This paper reports on the development of a realized VHDL modules, speed control system for use in an autonomous vehicle with an Spartan-3E FPGA-based SoC platform. The speed of the vehicle is determined by the analysis of motor hall-sensors, the "Division by Trial subtraction" algorithm and a period measurement is modeled as RTL. The implemented as a FSM with a datapath PI-controller coupled to a PWM which generates PWM sequences. A variation of the PWM-ratio leads to the motor speed change and thus to speed manipulation. The speed control system, a component of the SoC-based embedded system, is integrated into a MicroBlaze design.

Inhaltsverzeichnis

1. Einleitung

„Das Leben ist kurz, die Geschwindigkeit ist notwendig,
um das Gewünschte in begrenzte Zeit, über die wir verfügen, zu erreichen.“
Soichiro Honda

Seitdem die ersten Reittiere gezähmt, das erste Rad erfunden und die ersten Wagen gebaut wurden, versuchen die Menschen die Geschwindigkeit dieser Bewegungsmittel zu regeln. Zum Einsatz kamen verschiedene Brems- und Beschleunigungsmechanismen wie Zügel, Kratz- und Klotzbremse, Dampfmotoren, Sporen und Peitschen. Die wichtigste Komponente in diesem Regelkreis ist natürlich der Mensch. Er schätzt die Geschwindigkeit und den Energieverbrauch ab, beachtete Wetterbedingungen, die Änderungen des Weges und seiner Umgebung. Mit Hilfe dieser Information regelt und steuert er sein Bewegungsmittel nach eigenem Willen. Mit technischem Vorschritt bekam der Mensch in diesem Bereich mächtige Unterstützung, Bewegungsmitteln werden modifiziert, viele Aufgaben von Sensoren und Mikrocontrollern übernommen. Das erste „speed cruise control device“ patentierte 1945 Ralph Teetor. Die ersten Geschwindigkeitsregelanlagen wurden 1958 in Chrysler Imperial Fahrzeugen eingesetzt[1]. Inzwischen sind die Tempomaten in vielen Fahrzeugen als Zusatz- oder Serienausstattung zu finden. Diese Technologie bringt viele Vorteile und erweist sich besonders nützlich auf Straßen mit Geschwindigkeitsbeschränkungen und bei Fahrten mit Anhänger, sie führt zum entspannten Fahren und reduziert den Kraftstoffverbrauch, trotzt allem darf man die Wachsamkeit dabei nicht verlieren[2].

Der Hochschulwettbewerb „Carolo-Cup“ von TU Braunschweig stellt die Aufgabe, ein Gesamtkonzept eines autonomen Fahrzeugs zu entwickeln. Anforderungen des Wettbewerbs gleichen denen, die im realen Leben zu finden sind, allerdings sind die Fahrzeuge von Carolo-Cup im Maßstab 1:10 gebaut[3]. Das Projekt Fahrerassistenz- und Autonome Systeme (FAUST) der HAW-Hamburg bietet die Gelegenheit, sich mit der System-on-Chip-Technologie auseinander zu setzen, und sie an den mit FPGA-Plattformen ausgestatteten Modellfahrzeugen zu erproben. Dynamische Projekt-Disziplinen, wie Rundstreckenfahrt und Einparken erfordern ein Modul zur Regelung der Geschwindigkeit, einen sogenannten Tempomat.

Die System-on-Chip-Technologie ist ein Schwerpunkt bei der Entwicklung und Implementierung der Geschwindigkeitsregelanlage für ein autonomes Fahrzeug.

Sicherheit, Präzision, zeitkritische Reaktion spielen beim Fahrzeugfahren eine sehr wichtige Rolle, um dies zu gewährleisten, müssen die Fahrerassistenzsysteme sogenannten harten Echtzeitanforderungen genügen, das heißt, korrekte Ergebnisse rechtzeitig liefern[4]. Die Fahrzeuge selbst müssen mit verschiedenen Sensoren ausgestattet sein, um die Umgebung zu erfassen. Für die Wahrnehmung, Analyse, Entscheidung und den Handel ist ein zentrales System notwendig, genau diese Aufgaben erfüllen die eingebetteten Systeme. Heutzutage sind solche Systeme in den Satelliten, Flugzeugen und Fahrzeugen wie in Haushaltsgeräten

und Herzschrittmachern eingebaut[5]. Die Fahrzeuggeschwindigkeitsregelung ist ein eingebettetes Fahrerassistenz-System aus dem Bereich „Autonome Systeme“.

Diese Arbeit setzt als Ziel, die Entwicklung eines Fahrerassistenz-Systems zur Regelung der Geschwindigkeit eines autonomen Modellfahrzeugs, sowie weitere Systemintegration auf einen rekonfigurierbaren FPGA-Chip. Somit wird der Umgang mit „Eingebetteten Systemen“ und System on Chip-Technologie erprobt und trainiert. Dieses Fahrerassistenz-System soll das Fahrzeug auf die vorgegebene Geschwindigkeit beschleunigen, sie konstant halten und den Fahrweg messen. Durch sein Können wird das System in Rundstreckenfahrt- und Einparkdisziplinen eingesetzt und liefert einen Beitrag zu den Fahrspurführung- und Einparkassistenten. Der Geschwindigkeitsregler wird zur Einsparung der FPGA-Ressourcen, als Prozessor-Element mit einer Trennung in Steuer- und Datenpfad entworfen. Das ganze Fahrerassistenz-System wird als ein RTL-Modell in VHDL implementiert.

Die Thematik und die Schwerpunkte der Arbeit bestehen in der Realisierung des Geschwindigkeitsregelung-Systems:

- Analyse der PWM-Pulse zur Fahrzeugsteuerung, mit anschließender Ansteuerung des Motors über den Fahrtensteller, durch PWM-Generator aus SoC-Plattform.
- Ermittlung der Fahrzeuggeschwindigkeit und des Fahrweges durch die Auswertung der Hall-Sensor-Rechteckpulsen des Motors.
- Messtechnische Alyse mit Identifikation des Antriebsstrangs als Regelstrecke, Auswahl des Regler-Typs und Berechnung der Regler-Parameter, Diskretisierung und Simulation des Regelkreises mit MATLAB.
- RTL-Entwurf und -Modellierung des V-Reglers in VHDL und seine Kopplung in Form eines IP-Cores an den MicroBlaze Processor Local Bus (*vgl.* ***Bild 1.1***).
- Simulation des Regelungssystems und Inbetriebnahme des Fahrerassistenten.

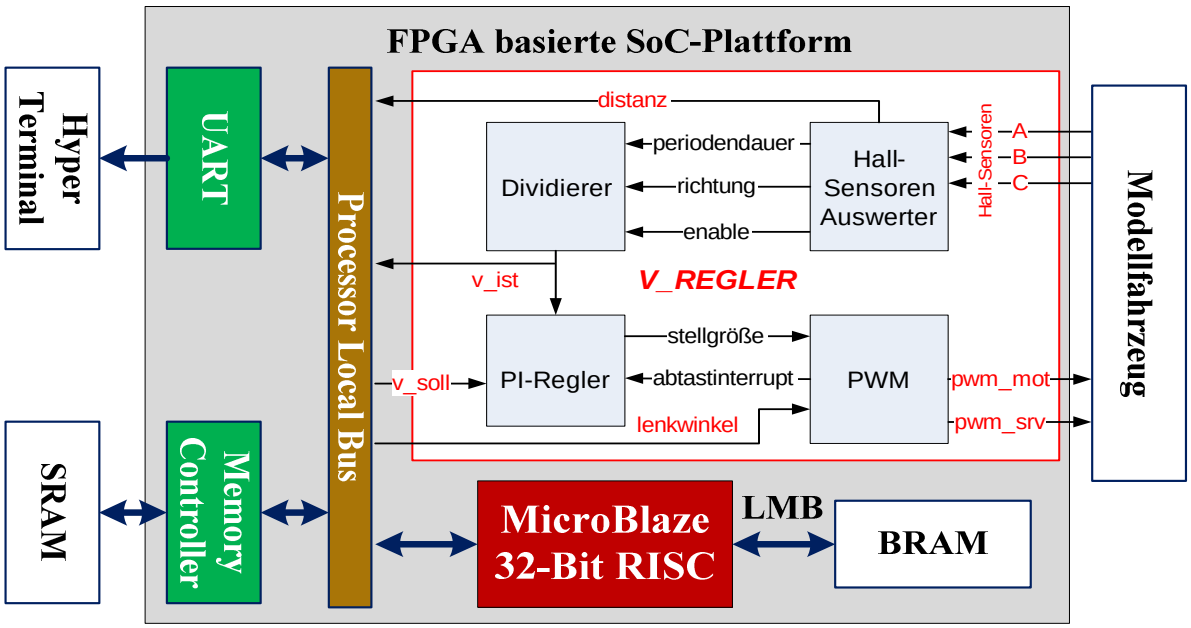

Bild 1.1.: *FPGA basierte SoC-Plattform mit dem Geschwindigkeitsregler als IP-Core am Processor Local Bus des eingebetteten MicroBlaze-Systems*

Kapitelübersicht

In **Kapitel 2** wird die Spartan-3E FPGA basierte SoC-Plattform vorgestellt. Es wird die Struktur des eingebetteten Systems mit dem MicroBlaze Softcore Prozessor beschrieben, und das Nexys2-Board mit den Peripherie-Komponenten, auf dem der Geschwindigkeitsregler in Betrieb genommen wird, präsentiert. Des Weiteren werden in diesem Kapitel die Massen des sensorgesteuerten Fahrzeugs (*SCV*) genannt, sowie der Aufbau und die Kopplung der Fahrzeugelektronik und -mechanik erläutert.

Kapitel 3 stellt die Konzepte und Zusammenhänge der Arbeit vor. Hier wird die Integration der SoC-Plattform in die Fahrzeugelektronik gezeigt, hierbei werden die Struktur und die Komponenten des IP-Cores zur Geschwindigkeitsregelung vorgestellt, sowie die Kommunikation des IP-Cores mit dem MicroBlaze über PLB beschrieben. Außerdem wird hier von den Techniken zur Modellierung eines Prozessor-Elements berichtet, sowie der Aufbau eines digitalen Regelkreises und die Methoden zur seiner Beschreibung aufgelistet.

Die am Fahrzeug durchgeführten Untersuchungen zur Timing-Analyse der PWM-Steuersignale aus der Fernsteuerung, sowie der Hall-Sensoren-Pulsfolgen des Motors werden im **Kapitel 4** beschrieben. Hier wird auch die Methode zur Ermittlung der Fahrzeuggeschwindigkeit aus der Periodendauer einer Motorumdrehung erläutert und deren Auswahl begründet, sowie die Funktionalität eines Motors mit Hall-Sensoren erklärt. Der Fahrzeugantriebstrang ist die Regelstrecke, die Sprungantwortaufnahme identifiziert wurde.

In **Kapitel 5** findet die Wahl des Reglers mit einer Darstellung der Regler-Vorteile statt, dann werden das Funktionsprinzip des ausgewählten Reglers erklärt und die Regler-Parameter bestimmt. Anschließend wird der Regelkreis diskretisiert, im MATLAB-Simulink aufgebaut und simuliert.

Das **Kapitel 6** berichtet über die Funktionalität und Algorithmen der einzelnen Komponenten des Fahrerassistenz-System zur Geschwindigkeitsregelung. Der Aufbau aller VHDL-Module wird als Blackbox definiert und als RTL-Modell präsentiert.

Die Funktionalität des gesamten Systems wurde mit einer messtechnischen Analyse überprüft. In **Kapitel 7** wurden die Simulation und die Testfahrtergebnisse untersucht, ebenso die Test-Software und der FPGA-Ressourcenverbrauch vorgestellt.

2. Technologieübersicht der SoC-Fahrzeug-Komponenten

Dieses Kapitel berichtet über die verwendete Spartan-3E FPGA basierte SoC-Entwicklungs-Plattform, ihre technischen Daten, I/O-Schnittstellen und Peripherie. Hier wird das FAUST-Modellfahrzeug mit seiner Aktorik und Mechanik vorgestellt. Die Kopplung dieser beiden System-Komponenten sowie deren Kommunikation basiert auf digitaler Signalverarbeitung und bildet somit ein komplexes digitales System.

2.1. FPGA-basierte SoC-Plattform

Der Spartan-3E FPGA Baustein des Nexys2 Entwicklungsboards besteht aus folgenden Logik-Ressourcen[6]:

- ***Configurable Logic Blocks(CLBs):*** Bestehen aus vier Slices, jeder Slice enthält zwei Look-Up Tabellen(LUTs), zwei Flip-Flops und carry- und arithmetische Logik. Die im Nexys2 eingesetzte Spartan-3E verfügt über 1200 Slices, mit deren Logik- und Speicherelementen das Geschwindigkeitsregelungssystem realisiert wird.
- ***Input/Output Blocks(IOBs):*** Als Rings umschließen sie die CLBs und verbinden den Datenfluss mit der internen Logik des FPGAs und den I/O Pins. IOBs realisieren eine bidirektionale Datenübertragung mit Three-State Operationen. Zur Synchronisations-zwecken stehen für jeden Pin jeweils ein Flip-Flop für Input und Output, deren Nutzung kann nach Bedarf freigeschaltet werden.
- ***Block RAM(BRAM):*** Liegt als 18Kbit Dual Port Blöcke vor, die synchron getaktet werden.
- ***Digital Clock Manager(DMC):*** Bieten eine digitale Lösung zur Verzögerung, Verteilung, Multiplikation, Division sowie Phasenverschiebung des Taktsignals an.
- ***Embedded Multipliers:*** Können zwei, bis 18Bit große Faktoren multiplizieren und liefern ein 36Bit Ergebnis.

Mit einem FPGA Baustein werden komplexe Hardware/Software Systeme realisiert: System on Chip Technologie (*SoC*). Eine SoC-Plattform besteht aus einem oder mehreren Mikrokontrollern, Peripheriekomponenten und IP-Cores (*intellectual property core*). Die IPs sind anwendungsspezifischen, vorhandenen oder gekauften Modulen, die die Systemfunktionalität erweitern. Die IPs, die von den anderen Projekten stammen, müssen analysiert und angepasst werden, bevor man sie einem SoC basierten eingebetteten Entwurf hinzugefügt[7]. Durch die programmierbaren Arrays FPGAs wird die Funktionalität des Boards ohne Lotvorgänge mit integrierten Schaltkreisen realisiert und erweitert.

Die Bussysteme eines Mikrokontrollers werden für den Daten-, Adress-, und Steuersignalaustausch zwischen den IPs verwendet. Ein Mikrokontroller hat einen System-, einen Peripherie- und einen Register- bzw. Steuerbus, die Schnelligkeit der Busse ist durch die Datentransferrate gegeben. Zur Kommunikation mit den Caches werden Buse verwendet, die Schreibe- und Lesezugriffe in wenigen Taktzyklen erledigen.

2.1.1. MicroBlaze Softcore Prozessor

In Spartan-3E FPGAs wird der MicroBlaze Softcore-Prozessor eingesetzt, er verwendet Big-Endian-Format und verfügt über virtuelle Speicherverwaltung. Der MicroBlaze ist ein 32Bit RISC Prozessor mit Harvard-Architektur. Dank Softcore-Technologie kann man ihn anwendungsspezifisch konfigurieren und seine Funktionen dem Systemdesign anpassen[8]. So können z.B. die Parameter wie Cachegröße, Anzahl der Pipelinestufen, verwendete Peripherie usw. eingestellt werden.

Der MicroBlaze-Entwurf (*vgl.* ***Bild 2.1***) besitzt mehrere Buse, die für den Anschluss der Peripherie und Speicher in einem FPGA vorgesehen sind. Der Prozessor und weitere Peripherien (*Interrupt, Memory Controller, UART*) sind über den Processor Local Bus (*PLB*) miteinander verbunden. Für die Kopplung des internen Speichers und Caches mit dem MicroBlaze wird der schnellere Local Memory Bus (*LMB*) eingesetzt, der für die Speicherzugriffe zwei Taktzyklen braucht, (*mehr, wenn der Schreibpuffer in dem Memory-Controller ausgelastet ist*). Weiterhin verfügt der MicroBlaze über 16 unidirektionale Fast Simplex Link Schnittstellen, die als „point to point“ FIFO basierte Kommunikation implementiert sind. Er unterstützt sowohl synchronen als auch asynchronen FIFO-Modus und erlaubt die Masters und Slaves mit unterschiedlichen Frequenzen zu takten[8].

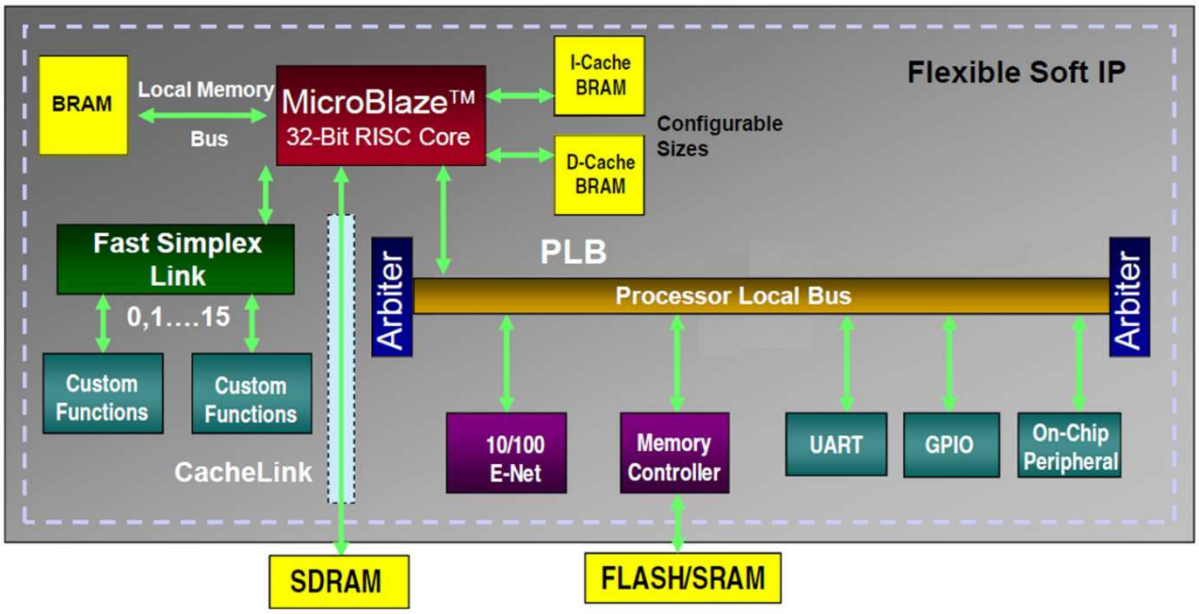

Bild 2.1.: *Aufbau eines SoC basierten eingebetteten Entwurfs mit dem MicroBlaze Softcore Prozessor*[8]

Zur Entwicklung komplexen Eingebetteter Systeme auf FPGA wurde der Embedded Development Kit (*EDK*)[7] verwendet, er besteht aus zwei Entwicklungsumgebungen, die auf Eclipse IDE basieren. Die Xilinx Platform Studio (*XPS*) wird für SoC-Entwurf des Systems aus NGC-, VHDL-, oder Verilog Files verwendet, mit XPS wird der Bitstream-File für die Ziel-FPGA generiert. Der Software Development Kit (*SDK*) dient für die Software-Entwicklung und nutzt als Programmiersprachen C oder C++. Durch das Downloaden des Bitstream-Files in das FPGA wird die Funktionalität des Systems überprüft.

2.1.2. Nexys2 Board mit Spartan 3E FPGA

Als Zielplattform für das entwickelte Geschwindigkeitsregelung-Systeme wird das Nexys2 Board[9] verwendet, die Abmessungen dieser Platine *12cm x 12cm* erlauben, sie mit einem Gestell aus Plexiglas, auf das Fahrzeug anzubringen. Nexys2-Board wird über USB-Kabel, über Netzteil oder über Akku-Zellen mit Eingangsspannung *5V bis 15V* versorgt, die Umschaltung zwischen Versorgungsquellen erfolgt mit einem Power-Jumper. Da das Fahrzeug mit einer NiMH Akku-Zelle (*7.2V / 3700mAh*) ausgestattet ist, wird das FPGA-Board direkt an sie angeschlossen. Die Board-Peripherie-Module sind mit dem Spartan-3E FPGA gekoppelt (*vgl.* ***Bild 2.2)***. Die Funktionalität und der Nutzen von den meistbenötigten Schnittstellen für die Geschwindigkeitsregelung sind weiter unten beschrieben.

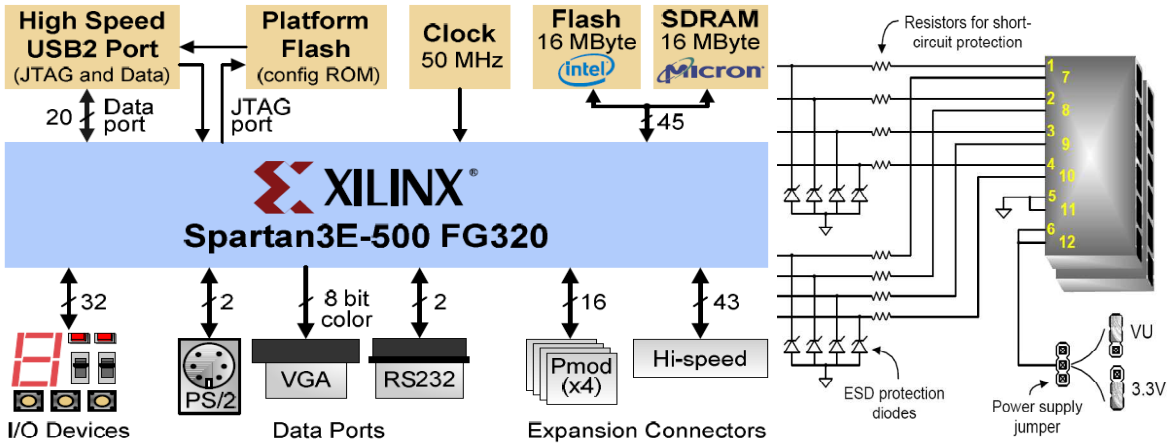

Bild 2.2.: *Nexys2-Entwicklungsboard mit Peripherie und Pmod-Aufbau*[9]

Die 4 Pmod Schnittstellen bieten je 12 Pins wovon 8 für I/O Signale, 2 für GND Pins und 2 für Vdd Pins Verwendung finden. Die Signalleitungen der Pmods wurden für die PWM-Ausgangssignale zur Lenkwinkel- und Motorsteuerung, sowie für die Eingangssignale der Hall-Sensoren zur Geschwindigkeitsmessung genutzt. Die Widerstände und Dioden der Signalpins schützen die Spartan-3E FPGA von den Eigenspannungen, die über 3.3V liegen[9]. Die Vdd Pins können entweder 3.3V Ausgangsspannung oder die Spannung der Akku-Zelle (*7.2V*) liefern, die Umschaltung der Modi erfolgt mit einem Supply-Jumper.

Die RS232 Schnittstelle wird verwendet, wenn die gespeicherten Testdaten in textueller Form von dem Board ausgelesen werden. Dieser Vorgang hat folgenden Ablauf: Von dem MikroBlaze über den UART wird der serielle Datenstrom in Form von logischen Pegeln an den ST3232 Pegelwandler geschickt, dieser wandelt die Pegel in die Spannungswerte der RS232-Kommunikation um. Über ein serielles Kabel werden die Daten an den PC weitergeleitet, dort läuft eine Kommunikationssoftware (*Hyper Terminal*), die so konfiguriert ist, dass sie diese Daten auf dem Bildschirm darstellt.

Für die Steuerung der Systemkomponenten werden 4 Tasten und 7 Schiebeschalter verwendet. Je nach Position erzeugen die Schalter einen high- oder low-Pegel, die Tasten dagegen liefern ein high-Signal nur, wenn sie gedrückt sind.

2.2. Modelfahrzeug im Maßstab 1:10 mit Allradantrieb

Das Fahrzeug des FAUST-Projekts ist ein 1:10 Allrad-Model, sein Fahrgestell basiert auf dem TAMIYA-TT-01 Bausatz[10] (*vgl.* ***Tabelle 2.1***), der vorne und hinten mit Querlenker, Kegeldifferentialen und Spiralfeder ausgestattet ist. Seine Karosserie ist in Form eines Ferrari Enzo (*vgl.* ***Bild 2.3***) aus Polykarbonat gebaut. Die für den Betrieb benötigte Spannung beträgt 7.2V, sie wird an den Fahrtensteller übertragen, eine 5V und eine GND Leitung dienen als Versorgung für den Acoms FR-4 Fahrzeugempfänger (*vgl.* ***Bild 2.4***). Eine Acoms Hayabusa Fernsteuerung[11], betrieben mit acht 1.2V/2500mAh Ni-MH Akkus, versendet Steuersignale mit der 40MHz Frequenz an den Fahrzeugempfänger.

Länge	455 mm
Breite	200 mm
Höhe	110 mm
Bodenfreiheit	9 mm
Gewicht	1490 g
Radstand	255 mm
Motorübersetzung	8.35 : 1

Tabelle 2.1.: *Technische Daten des TAMIYA-TT01Ferrari Enzo Chassis*[10]

Bild 2.3.: *Fahrzeug im Aufbauzustand ohne FPGA basierten SoC-Plattform*[10]

Fahrzeug Aktorik:

- ***Lenkwinkelsteller***: Servo Acoms AS-17 wird mit 5V Spannung von dem Fahrzeugempfänger versorgt, stellt den maximalen Lenkwinkel von 20° ein.
- ***Fahrtensteller***: A.I. Brushless Reverse überwachet die Motordrehzahl über die Hall-Sensoren und steuert mit PWM-Signal vom dem Fahrzeugempfänger den Motor [12].
- ***Motor***: Crawler Brushless 21,5 Turns wird von dem Fahrtensteller über 3 Phasen gesteuert, ist mit drei Hall-Sensoren zur Rotorpositionserkennung ausgestattet[13].

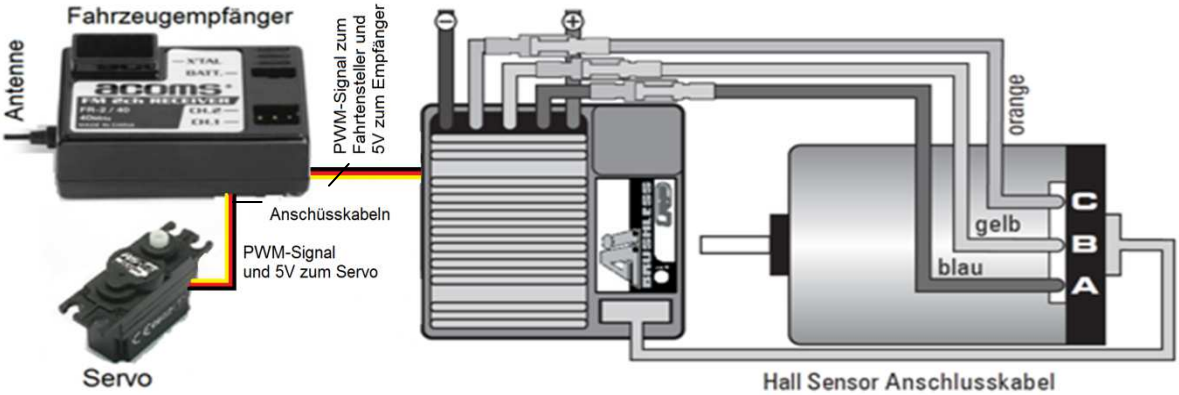

Bild 2.4.: *Aufbau der Fahrzeugelektronik mit den Empfänger-Anschlusskabeln, die den Servo und den Motor über PWM-Signale steuern*[12].

Fahrzeug Mechanik:

Bei dem TT-01Bausatz wird die Motorleistung über die Differentiale auf vier Räder verteilt. In der Mitte und entlang des Fahrzeugs verläuft eine Antriebswelle, das sogenannte Zentraldifferential. Seine Aufgabe ist es, die Kraft des Elektromotors auf die Vorder- und Hinterachse gleichmäßig zu verteilen, dies geschieht aufgrund der identischen Bauweise des vorderen und das hinteren Differentialgetriebes[14].

Für die Querverteilung der Kraft sorgen die Achsdifferentiale der Vorder- und Hinterachse. In einer Kurvenfahrt legen die Räder wegen verschiedender Streckenradien unterschiedlich lange Wege zurück. Daraus erfolgt, dass die Räder nicht mit gleicher Geschwindigkeit rollen. In einer Geradeausfahrt gleichen die Differentiale die Kraftverteilung aus, somit ist das Drehzahl-Verhältnis 50:50. Da das Fahrzeug keine Differentialsperre besitzt, wird beim Blockieren eines Rades die Drehzahl des anderen doppelt so hoch[15]. Dies deutet darauf hin, dass der Motor, über die Differentiale, auf jedes Rad 25% seiner Kraft übersetzt. Somit wird die durchschnittliche Fahrzeuggeschwindigkeit von der Motordrehzahl ermittelt.

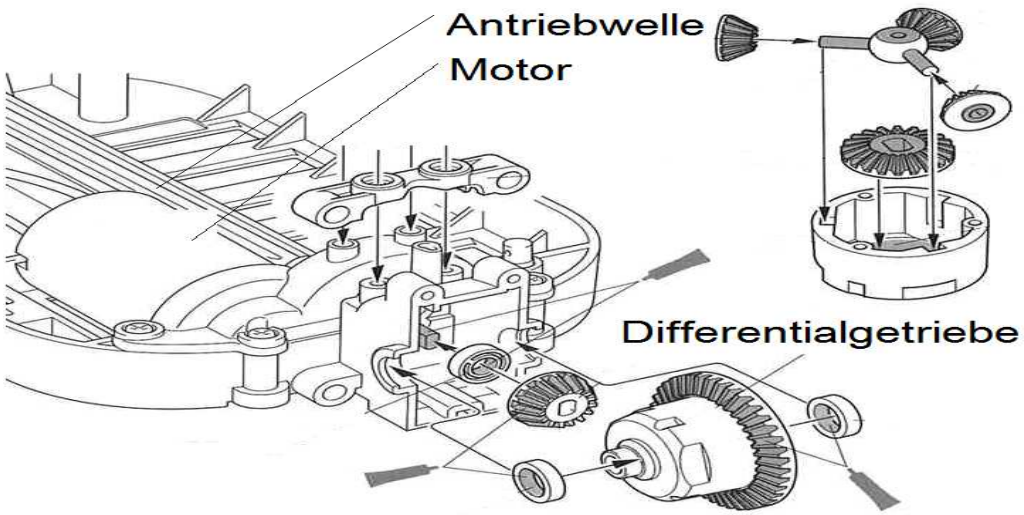

Bild 2.5.: *Aufbau der Fahrzeugmechanik mit Innenaufbau des vorderen und hinteren Differentialgetriebes, zur Verdeutlichung der gleichmäßigen Motor-Kraftverteilung*[14].

Diese mechanischen Eigenschaften verbessern das Fahrverhalten des Fahrzeugs. TAMIYA Ferrari Enzo ist ein robustes Modellfahrzeug mit Geschwindigkeiten bis zu 5m/s. Vor allem in der Kurvenfahrt erweist es sich als besonders spurstabil, denn aufgrund der Allradtechnologie untersteuert oder übersteuert das Fahrzeug nicht.

3. Konzepte und Zusammenhänge

Dieser Abschnitt wurde die Integration der FPGA-basierten SoC-Plattform in die Fahrzeugelektronik vorgestellt. Die Systemkomponenten und deren Kopplung, die Schnittstellen und ihr Signalaustausch wurden beschrieben und die Realisierung der Energieverteilung des SoC-Fahrzeugs bildlich erklärt (*vgl.* ***Bild 3.1***). Ebenso nannte man die Entwurfsmethoden einer Prozessor-Element-Architektur mit seiner Partitionierung in Daten- und Steuerpfad. Es wurden Schritte zum Entwurf der digitalen Geschwindigkeitsregelung aufgelistet, sowie deren Aufgaben, Funktionalität und die Methoden zur Regelkreisbeschreibung präsentiert.

3.1. Systemübersicht des autonomen Fahrzeugs

Die Hauptaufgabe des entwickelten Geschwindigkeitsregelungssystems besteht darin, das SoC-Fahrzeug bis auf eine vorgegebene Geschwindigkeit zu beschleunigen und trotz Ausschöpfung des Akkus diese zuhalten. Die Einhaltung der konstanten Geschwindigkeit ist für die Spurführung-Algorithmen notwendig, da die Geschwindigkeitsschwankungen die Regelstreckenparameter der Spurführung verändern, damit passen ihre Regeleinstellungen nicht mehr[16]. Zur Ermittlung der Fahrzeuggeschwindigkeit des Fahrweges und der Fahrrichtung werden die Rechteckpulse der im Elektromotor eingebauten Hall-Sensoren ausgewertet. In der Streckenrundfahrt-Disziplin realisiert das Geschwindigkeitsregelung-System die Konstant-Haltung der Geschwindigkeit, in der Einpark-Disziplin führt es zusätzlich die Fahrwegmessung aus.

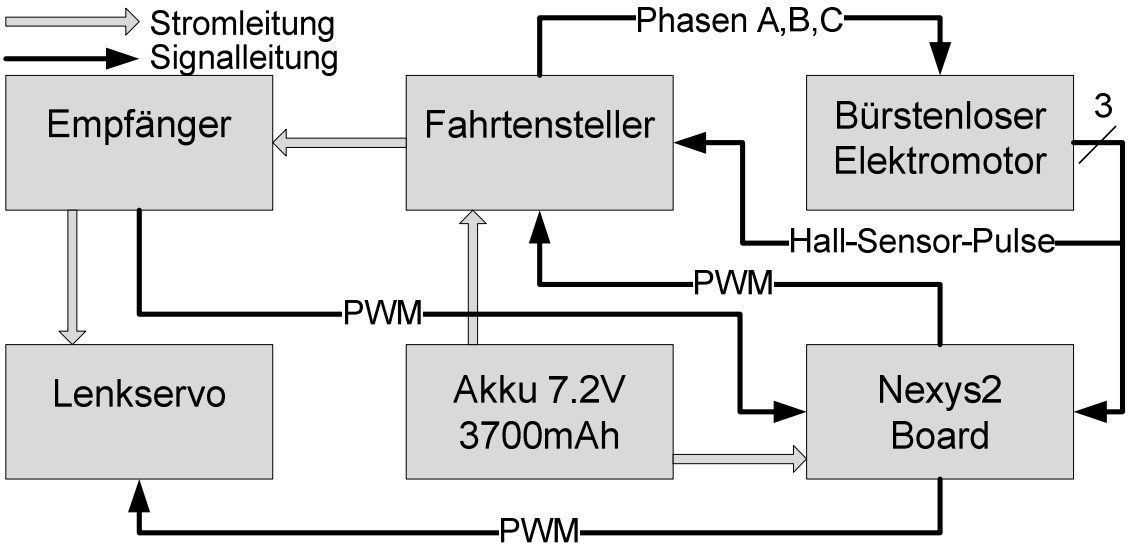

Bild 3.1.: *Kopplung des Nexys2-Boards mit der Fahrzeugelektronik, die PWM vom Empfänger ist zur Umschaltung der Fahrtmodi mit der Fernsteuerung vorgesehen*

Zur Einsparung der Systemressourcen wurde die Geschwindigkeitsregelung sls Prozessor-Element mit einem Multizyklus-Datenpfad, statt Software-oder Pipelinelösung realisiert. Die Softwareressourcen der SoC-Plattform sind für andere Steuermodule des autonomen Fahrzeugs wie Modi-Umschaltung, Einparkassistent u.a. reserviert. Des Weiteren braucht

man für die Softwarelosung einen Timer zur Abtastinterrupt- und PWM-Generierung. Der Standard-Timer aus dem IP-Katalog verbraucht allein fast genauso viele Ressourcen wie das gesamte Geschwindigkeitsregelungssystem. Gegen die Pipelinelösung sprechen niedrige Systemabtastrate (T_a = *17ms*) und höherer Ressourcenverbrauch(*vgl.* ***Anhang A***) im Vergleich zum Prozessor-Element mit dem Ressourcen- und Register-Sharing.

Der IP-Core zur Geschwindigkeitsregelung wurde mit dem Xilinx Entwicklungstool EDK generiert. Bei seiner Erstellung wurde diese Peripherie-Komponente wie folgt beschrieben:

Name:	v_regler_hw;
Softwareregister:	2 x 32Bit;
Gekoppelt an:	Processor Local Bus (*PLB*);
Interrupt-Erzeugung:	Ja;
Entwurfssprache:	VHDL.

Beschreibung der VHDL-Module des Geschwindigkeitsregler-IP-Cores (*vgl.* ***Bild 3.2***):
v_regler_hw: Ist eine Top-Entity, in der die Busschnittstelle, User Logik und der Interrupt-Kontroller (*da IP selbst Interrupts generiert*) instanziiert sind.
uder_logic: Implementiert die Logik zur Beschreibung und zum Auslesen der Softwareregister und instanziiert die Anwender-Logik.
V_REGLER: Implementiert die Funktionalität des Geschwindigkeitsregelungssystems

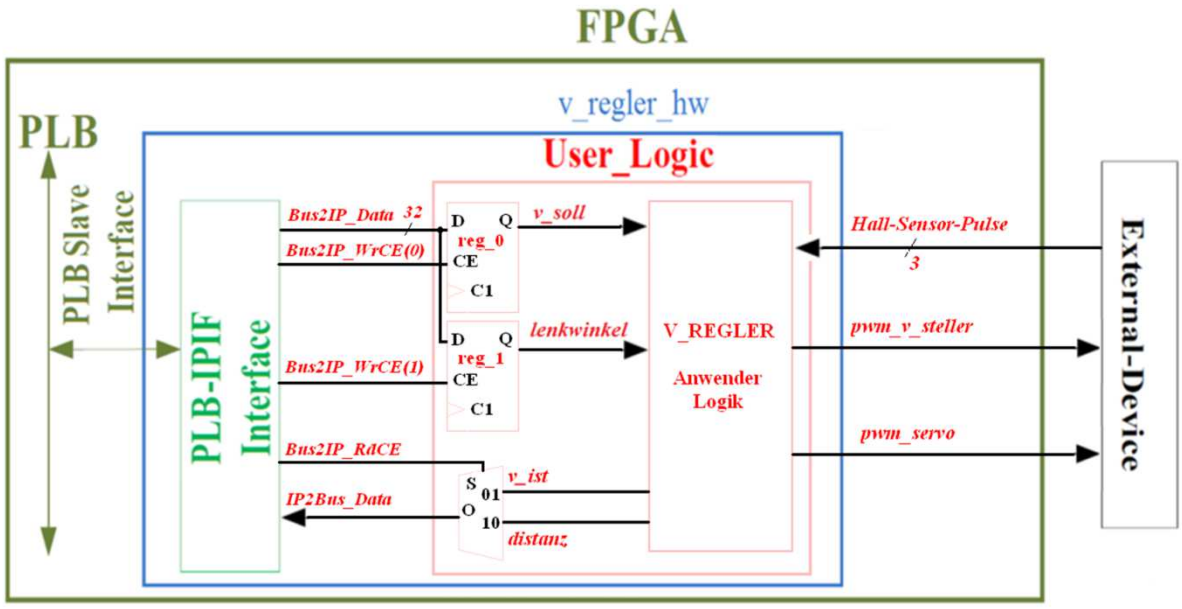

Bild 3.2.: *Komponentenübersicht des generierten IP-Cores, über die Software-Register wird die Anwender Logik konfiguriert und über Mux die Ergebnisse gelesen*

PLB-IPIF Interface: Ist eine bidirektionale Schnittstelle, die Kommunikation der Peripherie-Komponente mit dem MicroBlaze-Prozessor abwickelt[7]. Zu den getakteten Aufnahmen der Soll-Geschwindigkeit und des Lenkwinkels werden die Slave-Register reg0- und reg1 verwendet. Zum kombinatorischen Auslesen der gefahrenen Strecke und der aktuellen Ist-Geschwindigkeit wurde ein Multiplexer mit zwei Eingängen modelliert. Die *v_ist* und *distanz*

werden über PLB von dem MicroBlaze abgeholt, und am Monitor durch Nutzung der RS232 Schnittstelle und Hyper Terminal Software abgebildet (*vgl.* ***Kap.2.1.2***).

Der Nutzer stellt über die Tasten oder Schiebeschalter am Nexys2-Board (*vgl.* ***Bild 2.2***) die Geschwindigkeit ein, die zugehörige Software interpretiert die Peripheriesignale als eine Sollgeschwindigkeitsvorgabe (*v_soll*), der MicroBlaze schreibt dann diese Werte über den PLB in das SW-Register. Der PI-Regler erfasst bei jedem Abtastinterrupt die aktuelle Geschwindigkeit (*v_ist*) und vergleicht sie mit der vorgegebenen (*v_soll*). Der PI-Regler-Algorithmus bildet aus der Geschwindigkeitsdifferenz die Regelabweichung und berechnet die Stellgröße, die dann durch PWM-Generator in ein PWM-Signal für den Fahrtensteller umgewandelt wird. Durch Verstärkung und Integration der Stellgröße wird die Regelabweichung minimiert und so die Fahrzeuggeschwindigkeit angepasst.

Das Konzept zur Geschwindigkeitsmessung mit Hall-Sensoren des Motors basiert auf folgendem Prinzip (*vgl.* ***Gl. (3.1) u. (3.2)***): Eine Hall-Sensor-Pulsperiode entspricht einer Motorwellenumdrehung, je größer die Periodendauer desto langsamer dreht sich die Motorwelle. Eine Wellenumdrehung ist durch eine konstante Fahrstrecke gekennzeichnet. Die Periodendauer ergibt die Dauer einer Wellenumdrehung. Gefahrene Strecke geteilt durch Fahrzeit gleich Geschwindigkeit. Aus dem Radumfang und Übersetzungsverhältnis lässt sich der zurückgelegte Weg pro Wellenumdrehung errechnen.

$$S_{wu} = \frac{U_r}{i_{Antrieb}} \quad (3.1) \qquad\qquad v_{ist} = \frac{S_{wu}}{t_{hs}} \quad (3.2)$$

U_r – Radumfang; $i_{Antrieb}$ – Übersetzungsverhältnis; S_{wu} – Gefahrene Strecke pro Wellenumdrehung des Motors; t_{hs}–Hall-Sensor-Periodendauer; v_{ist}–Fahrzeuggeschwindigkeit

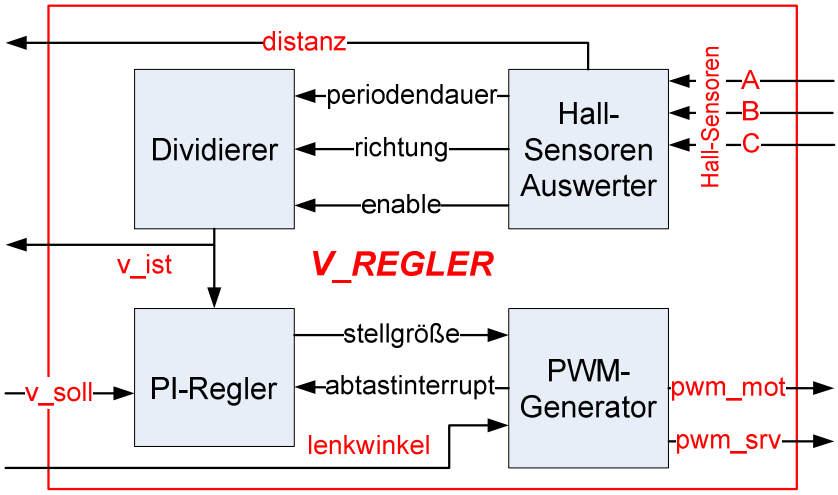

Bild 3.3.: *Anwenderlogik des Geschwindigkeitsreglers*(*vgl.* ***Bild 3.2***)

Die VHDL-Komponente „V_REGLER“ implementiert die Anwender Logik und ist eine Top-Entity, die alle Elemente des Geschwindigkeitsreglers verbindet (*vgl*. ***Bild 3.3***). Sie hat folgende Struktur:

- ***Hall-Sensoren-Auswerter***: Die Aufgabe dieses Moduls besteht darin, die Zeit zwischen zwei aufsteigenden Flanken des Hall-Sensor-Pulses zu messen. Diese Flanken generieren einen Timer-Reset bei dem, die gemessene Dauer festgehalten wird. Der Timer misst dann die nächste Periode und so erhält man mit jeder Flanke ein Periodendauer-Update. Weiterhin bestimmt dieses Modul die Fahrtrichtung und den Fahrweg.
- ***Dividierer***: Da die FPGAs keine dedizierten Dividierer-Ressourcen anbieten, wurde ein „Division by Trial Subtraction“ Dividierer als Prozessor-Element mit Multizyklus-Datenpfad modelliert, um aus Fahrweg und Fahrzeit die Geschwindigkeit zu ermitteln. Seine Architektur ist speziell so implementiert, dass er den konstanten Fahrweg durch die, vom Hall-Sensor-Auswerter gemessene Periodendauer dividiert.
- ***PI-Regler***: Durch die Abtastinterrupts vom PWM-Generator wird der PI-Regler-Algorithmus angestoßen, die aktuelle Geschwindigkeit (*v_ist*) liest der Regler aus dem Register im Dividierer-Modul, die vorgegebene (*v_soll*) aus dem Software-Register aus. Die Geschwindigkeitswerte liegen im Bereich -480 und 480 (*cm/s*) und sind keine Kommazahlen.
- ***PWM-Generator***: Erzeugt Rechteckpulse (pwm_mot, pwm_srv) mit Periodendauer T_{pwm} = 17ms und Tastverhältnis zwischen $5\% \leq a \leq 10\%$, zur Steuerung des Fahrten- und Lenkwinkelstellers. Abhängig von der Stellgrößen werden die Zeitpunkte der fallenden Flanken der PWM-Rechteckpulse gesetzt und somit das PWM-Tastverhältnis manipuliert

3.2. Modellierungstechnik des Prozessor-Elements

Das Prozessor-Element, mit Trennung der Anwenderlogik in Datenpfad und Steuerpfad (*vgl.* ***Bild 3.4***), wird bei Entwurf digitaler Systeme zur Echtzeitdatenverarbeitung eingesetzt. Besonders in den Fällen, wo die FPGA-Ressourcen zu sparen sind und mehrere Systemtakte zwischen den Ergebnisberechnungen liegen. Es hat sich etabliert, die beiden Teile der Logik voneinander zu trennen, da an jedes Teil unterschiedliche Entwurfsstrategien und Optimierungsmethoden angewendet werden[17].

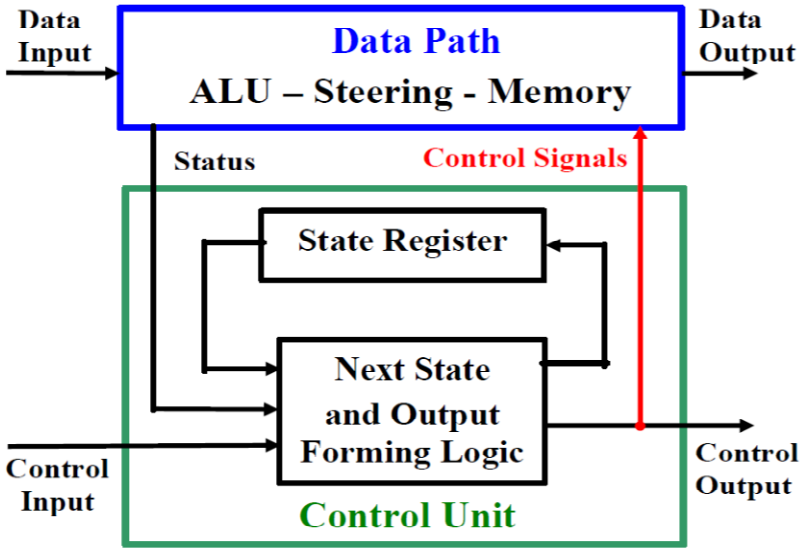

Bild 3.4.: *Datenpfad und Steuerpfad eines digitalen Systems als Prozessor-Element*[18]

Der Entwurfsprozess zeigt folgende Schritte auf:

1. Verhaltensmodell durch mathematische Gleichungen und ASM beschreiben;
2. Ressource- und Register-Sharing durchführen;
3. Entwurf des Steuerpfades als einen taktsynchronen Zustandsautomat;
4. Strukturbild des Datenpfades entwerfen;
5. Register Transfer Level (*RTL*)-Modellierung in VHDL.

ASM – (*Algorithmic State Machine*) dient der Beschreibung von Zustandsautomaten auf einer höheren (*algorithmischen*) Abstraktionsebene. Die ASMs beschreiben die sequentiellen Operationen eines digitalen Systems, das einen Algorithmus implementiert. ASM-Diagramme ähneln den Flussdiagrammen, die in der Computerprogrammierung genutzt werden. Im Gegensatz zu den Flussdiagrammen beschreibt eine ASM ein zeitliches Verhalten, da die Übergänge zwischen den Zuständen durch die Flanken eines Taktsignals ausgelöst werden[17].

Ressource-/Register-Sharing – ist ein Optimierungsvorgang, der zur gemeinsamen Nutzung der Funktionseinheiten (*Addierer, Subtrahierer, Schieberegister, Multiplexer, usw.*) sowie Speichereinheiten (*Flip-Flops*) dient. Zur mehrmaligen Durchführung einer mathematischen Operation benutzt man ein und dieselbe Funktionseinheit. In den meisten Fällen wird die gleiche Ressource zu verschiedenen Zeitpunkten von unterschiedlichen Operanden benötigt,

durch Multiplexer erfolgt das taktabhängige Umschalten der Operanden. Die Produkte der Funktionseinheiten können abhängig von der Nutzdauer in ein und demselben Register platziert werden. Der Zustandsautomat legt fest, welche Operationen in den verschiedenen Takten mit den Registerwerten durchgeführt werden sollen. Durch diese Optimierung können manche Operationen parallel ausgeführt werden. Die Zustände der ASM und die Register lassen sich einsparen, so wird nicht nur FPGA-Ressourcenbedarf reduziert, sondern auch die Zeit zur Berechnung der Ergebnisse verkürzt[19].

Control Unit: Darunter versteht man eine Komponente zur Steuerung der Schaltungssequenz von Registern und Funktionseinheiten. Sie ist als ein taktsynchroner endlicher Zustandsautomat aufgebaut (*FSM*). In jedem Zustand, bezogen auf die Eingangssignale, werden die Aktionen im Datenpfad durch Steuersignale ausgelöst. Die Aktionsausführung erfolgt meistens erst im nächsten Takt, da es sich hier um Freigabesignale handelt[17].

Data Path ist Komponente, die vom Algorithmus beschriebene arithmetische Operationen ausführt. Sie besteht aus kombinatorischen Elementen zur Datenmanipulation und aus getakteten Elementen zur Datenspeicherung. Der aktuelle Zustand wird per Statussignale an den Steuerpfad gesendet. Dargestellt wird sie meistens als eine RTL-Beschreibung – ein Strukturbild das die Verbindungen zwischen Funktionseinheiten und Registern wieder gibt[17].

RTL-Modellierung in VHDL – Das Register Transfer Level dient als Abstraktionsebene zur Beschreibung von Schaltkreisen. Die RTL-Modellierung hat als Ziel die Fertigstellung eines synthetisierbaren Codes[19]. Die Struktur des Codes muss den RTL-Anforderungen genügen, z.B. werden die kombinatorischen Prozesse von den getakteten getrennt, die Module, die verschiedene Aufgaben haben, werden nicht in einer Entity geschrieben. Die Nutzung der VHDL-Elementen wie „after“ und „wait for“ ist verboten. Meistens verwendet man bei dieser Modellierung die logischen Verknüpfungen wie „and“, „or“, „xor“, sowie Entscheidungsblöcke wie „if-then-else“ und „case-when“. Die Einhaltung der Regeln führt zum synthetisierbaren und besser lesbaren VHDL-Code.

3.3. Digitale Regelung, Regelkreiselemente, Beschreibungsmethoden

Im Funktionsprinzip eines digitalen Regelungsverfahrens liegt die Annahme, dass der digitale Regler sich als lineares Abtastsystem definieren lässt. Dies bedeutet: die Abfrage der Soll und Istwerte in konstanten Zeitabständen, auch Abtastperiode oder Abtastzeit genannt[20]. Der Prozess der Geschwindigkeitsregelung besteht im Wesentlichen aus folgenden Schritten:

1. Die aktuellen Geschwindigkeit erfassen (*Regelgröße*)
2. Soll- und Ist-Geschwindigkeiten vergleichen (*Regeldifferenzbildung*)
3. Ein geeignetes PWM-Signal erzeugen (*Stellgröße*)
4. PWM-Signal zum Fahrtensteller senden (*Stellgliedübertragung*)

Diese Aktionen führt das Geschwindigkeitsregelungssystem innerhalb der Abtastzeit T_a= *17ms*, in derselben Reihenfolge periodisch aus, in der Fachliteratur als geschlossener Regelkreis bezeichnet. Der Unterschied einer Regelung von einer Steuerung besteht darin, dass bei einer Regelung die Rückkopplung zwischen Messgröße und Stellgröße existiert[21].

Der Regelkreis (vgl. ***Bild 3.5.***) des Geschwindigkeitsregelungssystems setzt sich aus zwei Haupt-Elementen zusammen, der Regelstrecke, definiert durch das Antriebsverhalten des Fahrzeugs, einschließend seiner mechanischen Eigenschaften, und dem Regler, entworfen als eine digitale Proportional-Integrierende-Regelung. Die Regelstrecke enthält technische und physikalische Prozesse, die linearen Regelstecken werden durch Differenzialgleichungen und kontinuierliche Übertragungs-Funktionen im Zeitbereich beschrieben. Damit die Regelgröße v_{ist} den vorgegebenen Wert v_{soll} erreicht, beeinflusst der PI-Regler über die Stellgröße PWM die Regelstrecke. Der Regler besteht aus dem Messfühler, dem Vergleicher und dem PI-Regler-Algorithmus[21].

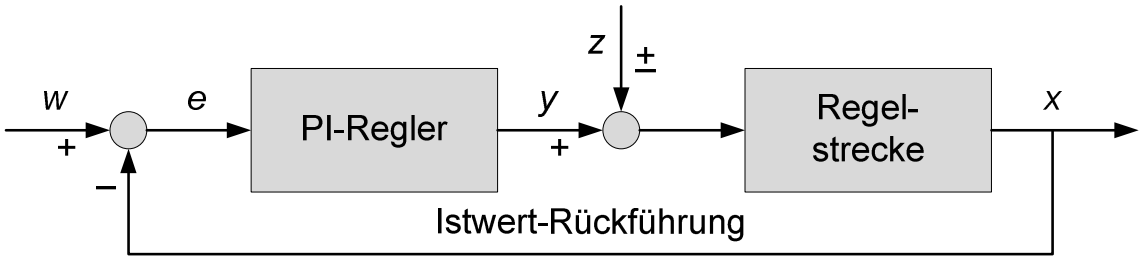

Bild 3.5.: *Wirkungsplan des Regelkreises, mit Sollwert w, Regelabweichung e, Stellgröße y, Störgröße z und Istwert x*[21]

Zum Entwurf des Geschwindigkeitsregelkreises wurden folgende Schritte durchgeführt:

1. Regelstreckenanalyse und Beschreibung mit kontinuierlicher Übertragungsfunktion.
2. Reglerauswahl und Parametrisierung, die Vorteile des PI-Gliedes für die V-Regelung.
3. Diskretisierung des Reglers mit Integration nach Trapezregel, Regelkreissimulation.
4. RTL-Entwurf der Komponenten der Geschwindigkeitsregelung und ihre Skalierung.
5. VHDL-Implementierung der RTL-Module des V-Reglers.

Die Analyse der Regelstrecke gibt Informationen über ihre Art, Reaktion und ihr Zeitverhalten, dieses Wissen ist eine Vorbedingung, um einen Regler zu entwerfen und einen funktionierenden Regelkreis zu bauen. Von Faktoren, wie die Regelgenauigkeit und die Regelgeschwindigkeit, hängt die Auswahl des Reglers ab[21]. Es gibt drei wichtige Regler-Typen: P– (*Proportional*), I– (*Integral*), D– (*Differential*) Regler und jeder reagiert anders auf die Abweichung der Regelgröße. Durch Kombination dieser Regler entstehen neue (PI, PD, PID), diese vereinen in sich die Eigenschaften der einzelnen Regler. Durch eine anwendungsspezifische Regler-Parametrisierung werden Dauerschwingungen, Instabilität, Ungenauigkeit usw. minimiert, aus diesem Grund ist die Untersuchung über das Verhalten des Regelkreises eine wichtige und unverzichtbare Voraussetzung.

Die Systemabtastrate der Geschwindigkeitsregelung deutet daraufhin, dass es eine digitale und keine analoge Regelung ist. Ein digitales Regelungssystem ist ein Kreis, in dem eine Regelstrecke von einem digitalen Regler gesteuert wird. Die Vorteile solcher Regelungen bestehen darin, dass die Methoden und Werkzeuge der Informationstechnologie in vollem Umfang für die Regelungszwecke anwendbar sind. Angefangen von der Analyse über den Regelkreisentwurf und Simulation bis hin zur Erstellung der Dokumentation sind sie eine große Hilfe[21].

Da in digitalen Regelkreisen die Folgen von abgetasteten Istwerten auftreten, sind sie nicht mehr kontinuierlich und lassen sich nicht, mittels Differentialgleichungen und kontinuierlichen Übertragungsfunktionen im Zeitbereich beschreiben[20]. Zur Charakterisierung der digitalen Systeme sind folgende Methoden definiert:

- ***Diskretisierte Beschreibung im Zeitbereich***: Die durch Differentialgleichungen beschriebenen Regelalgorithmen werden diskretisiert, indem das Integral durch Summe und die Differentiation durch Differenzenquotienten ersetzt wird, so entstehen Differenzengleichungen zur Regelkreisbeschreibung[20].
- ***Beschreibung mittels z-Transformation***: Die Übertragungsfunktionen, die analoge Regelalgorithmen beschreiben, lassen sich nach Trapeztegel in die z–Übertragungsfunktionen transformieren[20].

Da alle Prozessoren diskret arbeiten, sind diese Beschreibungen notwendig, um einen digitalen Regelalgorithmus für Geschwindigkeitsregelung zu realisieren.

Mit MATLAB wurde das Regelkreisverhalten untersucht und die PI-Regler-Parameter optimiert. Durch Skalierung der Rechengrößen werden unnötige Rechenoperationen und Größenumwandlungen für VHLD-Implementierung eingespart.

4. Aktorik, Sensorik und Fahrverhalten des Fahrzeugs

In diesem Kapitel werden die Steuerelemente und die Istwertgeber des Fahrzeuggeschwindigkeitsregelkreises analysiert (*vgl.* ***Bild 2.4***). Die Information über die Steuer- und Istwertgeber-Pulse wurde bei der Implementierung von PWM-Modul und Hall-Sensor-Auswerter verwendet. Es folgen die Untersuchungsergebnisse zu den Aktorik- und Sensorikelementen:

- Mit RTL-Modellen zu realisierende Timing-Kennwerte der PWM-Steuersignale für den Fahrten- und den Lenkwinkelsteller. Untersucht wurden die vom Empfänger generierten PWM-Steuerpulse.
- Auswertung der Hall-Sensor-Rechteckpulsfolgen zur Erfassung der Fahrzeuggeschwindigkeit-Istwerte.
- Identifikation eines Fahrzeugregelstreckenmodels zur Bestimmung der Regler-Parameter durch Sprungantwortanalyse des Fahrzeugantriebsystems.

4.1. PWM-Kennwerte des Fahrten- und des Lenkwinkelstellers

Die Pulsbreitenmodulation (*PWM*) ist Mittelwert-Variation mit Pulsbreitenveränderung bei konstanter Frequenz. Ziel der PWM ist es, mit einer digitalen Schaltung eine analoge Steuerausgabe für analoge Geräte zu realisieren.

Die Steuersignale der Fernsteuerung werden von dem Fahrzeugreceiver empfangen und in PWM-Signale umgewandelt (*vgl.* ***Bild 2.4.***). Das Ziel ist es jedoch, das Fahrzeug mit der FPGA-basierten SoC-Plattform durch Generierung der PWM-Pulse zu steuern. Bei dem SoC-Fahrzeug werden mit PWM-Signalen die Geschwindigkeit und der Lenkwinkel eingestellt.

Mit dem PC-Oszilloskop wurde die Periodendauer der vom Empfänger generierten PWM-Pulse gemessen, sie beträgt T_{pwm} = 17ms (*vgl.* ***Bild 4.1.***). Die PWM-Periodendauer sowohl für den Fahrten- als auch für den Lenkwinkelsteller sind identisch. Das Tastverhältnis des PWM-Pulses ist ein prozentualer Anteil der Dauer des Puls-High-Zustandes zur Dauer der Pils-Periode ($a = t_{High} / T_{pwm} * 100\%$).

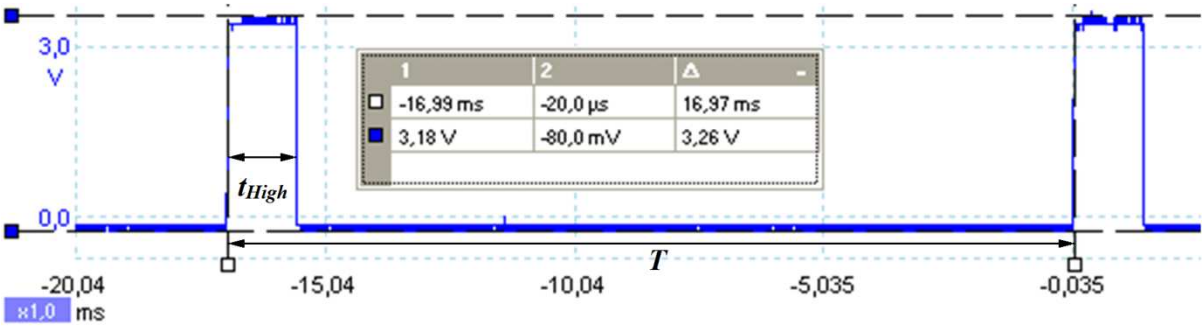

Bild 4.1.: *PWM-Signal des Fahrtensteller bei Fahrzeugstillstand oder des Lenkwinkelstellers bei Mittestellung, Periodendauer T_{pwm} = 17ms, Tastverhältnis $a = t_{High} / T_{pwm} *100\% = 7.5\%$*

In den nächsten Bildern sind die Untersuchungsergebnisse der folgenden Messungen dargestellt:

- ***Versuch 1:*** Von der Fernsteuerung wurden die Steuersignale ***„voll Gas vorwärts“*** und ***„Linksanschlag“*** (*vgl.* ***Bild 4.2***), zum Empfänger gesendet.
- ***Versuch 2:*** Von der Fernsteuerung wurden die Steuersignale ***„voll Gas rückwärts“*** und ***„Rechtsanschlag“*** (*vgl.* ***Bild 4.3***), zum Empfänger gesendet.

Es wurde festgestellt, dass die im Versuch 1 vom Empfänger generierten PWM-Pulse für den Fahrten- und für den Lenkwinkelsteller identische Tastverhältnisse aufweisen. Das gleiche gilt für den Versuch 2.

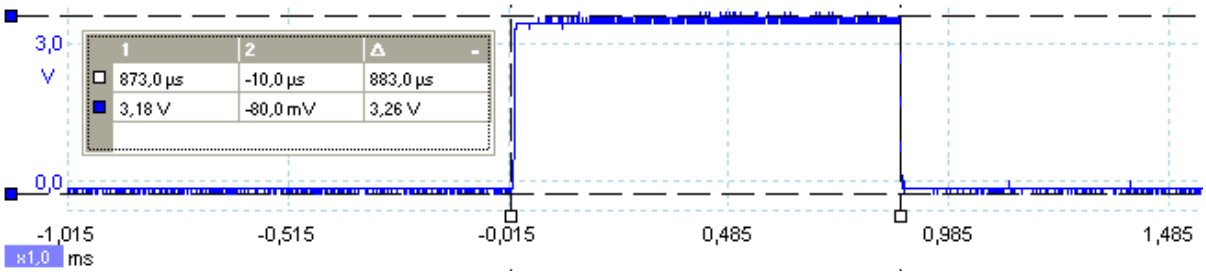

Bild 4.2.: *PWM-Signal bei „voll Gas vorwärts“ oder „Linksanschlag“ Pulsbreite t_{High} = 0.883ms, Tastverhältnis a = t_{High} / T_{pwm} * 100%= 5%*

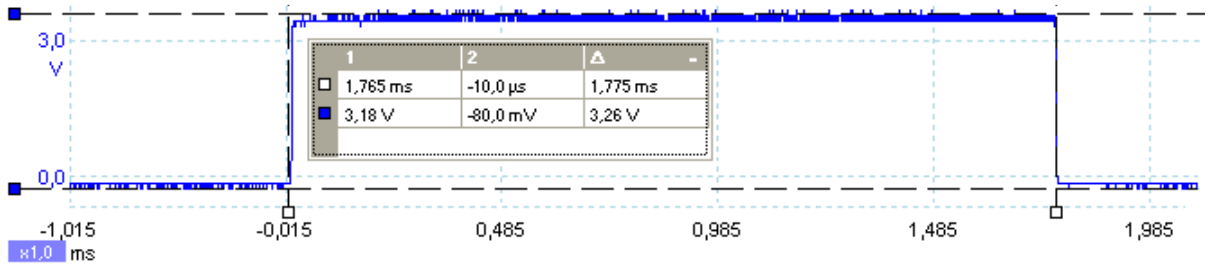

Bild 4.3.: *PWM-Signal bei „voll Gas rückwärts“ oder „Rechtsanschlag“ Pulsbreite t_{High} = 1.775ms, Tastverhältnis a = t_{High} / T_{pwm} *100% = 10%.*

Für eine bessere Übersicht wurden die Timing-Kennwerte der PWM-Steuerpulse in der **Tabelle. 4.1** zusammengefasst. Bei der Generierung dieser Pulse durch die FPGA basierten SoC-Plattform sind die Tastverhältnisgrenzen im Intervall 5% < a < 10% einzuhalten, da ansonsten die Fahrten- und Lenkwinkelsteller zerstört werden.

Pulspegel	3.3V
Periodendauer T_{pwm}	17ms
Tastverhältnisintervall	5% ≤ a ≤ 10%
Pulsbreite t_{High} bei 100% vorwärts und bei 100% links	0.88ms, a = 5%
t_{High} bei 100% rückwärts und bei 100% rechts	1.8ms, a = 10%
t_{High} bei Fahrzeugstillstand und Lenkwinkelmittelstellung	1.35ms, a = 7.5%

Tabelle 4.1.: *PWM-Kennwerte des Fahrten- und des Lenkwinkelstellers*

4.2. Timing-Analyse der Hall-Sensor-Pulsfolgen des Fahrmotors

Die Timing-Analyse der Hall-Sensor-Rechteckpulse wurde zur Einsparung zusätzlicher Mechanik-Elemente durchgeführt. Anstatt mit den Pulsgebern an den Rädern zu arbeiten wird die Fahrzeuggeschwindigkeit aus der Drehzahl der Motorwelle abgeleitet. Durch Auswertung der Hall-Sensor-Rechteckpulse wird die Geschwindigkeit des Fahrzeugs ermittelt und von der FPGA basierten SoC-Plattform über ein PWM-Signal geregelt.

Der bürstenlöse Gleichstrommotor (*BLDC-Motor*) ist eine Art des Synchronmotors, dies bedeutet, dass die vom Stator und vom Rotor erzeugten Magnetfelder mit gleicher Frequenz rotieren. Wehrend der Rotor ein Dauermagnet ist, besteht der Stator aus 3 Wicklungen, die über 3 Phasen von dem Fahrtensteller eingeschaltet werden. Das von den Wicklungen erzeugte Magnetfeld stößt den Dauermagnet ab oder zieht ihn an, so entsteht die Drehkraft[22].

Die BLDC-Motoren werden mit drei Hall-Sensoren ausgestattet (*vgl.* ***Bild 4.4***), durch ihre Auswertung ermittelt der Fahrtensteller die Rotorposition, und führt die richtige Sequenz zur Beschaltung der Motorphasen aus. Die Sensoren werden meistens auf dem nicht angetriebenen Ende des Motors angebracht. Wenn die Magnetpole des Rotors sich in der Nähe des Sensors befinden, liefert er ein high oder low Signal (*Hall-Effekt*)[22].

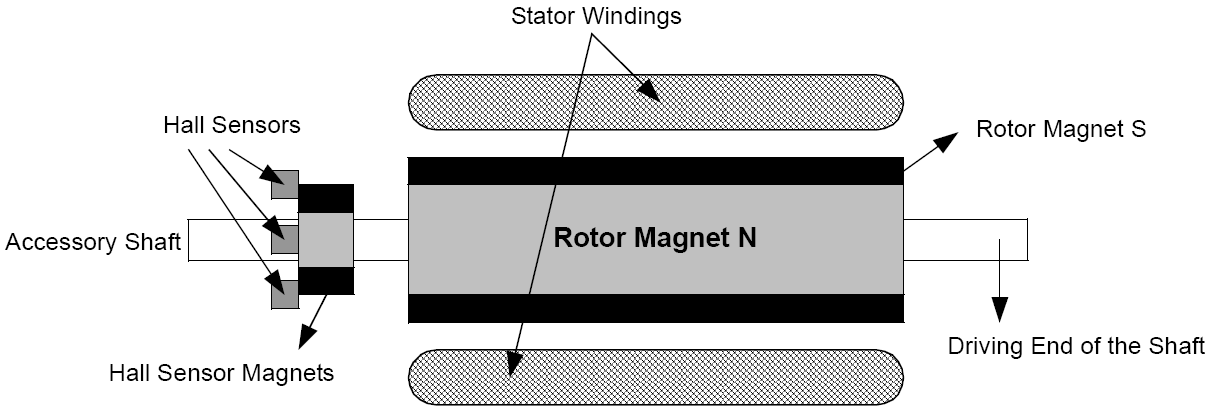

Bild 4.4.: *Querschnitt des bürstenlösen Gleichstrommotors mit N- und S-Polen des Dauermagneten und Hall-Sensoren*[22]

Der Fahrtensteller erhält vom Nexys2-Board das PWM-Signal und wertet die Hall-Sensor-Rechteckpulse aus, entsprechend dieser Steuer- und Stellungsinformationen werden die Wicklungen des Stators über Motorphasen beschaltet.

Die Bauweise der Hall-Sensoren des bürstenlösen Motors sorgt dafür, dass die Signale der Hall-Sensor-Pulsen niemals gleichzeitig high- oder low-Pegel generieren. So werden aus acht Kombinationen, die aus drei high/low Signalen ($2^3 = 8$) entstehen, nur sechs benutzt, um die Rotorstellungen abzubilden. Mit jeder Bitkombination aus drei Hall-Sensor-Rechteckpulsen werden somit 60° der Rotorumdrehung abgebildet (*360°/6*).

Die Periodendauer der Hall-Pulse bei konstanter Geschwindigkeit sind gleich (*vgl.* ***Bild 4.5***). Die Hall-Sensor-Pulse des Motors stehen je nach Drehrichtung des Rotors in folgenden Zusammenhängen:

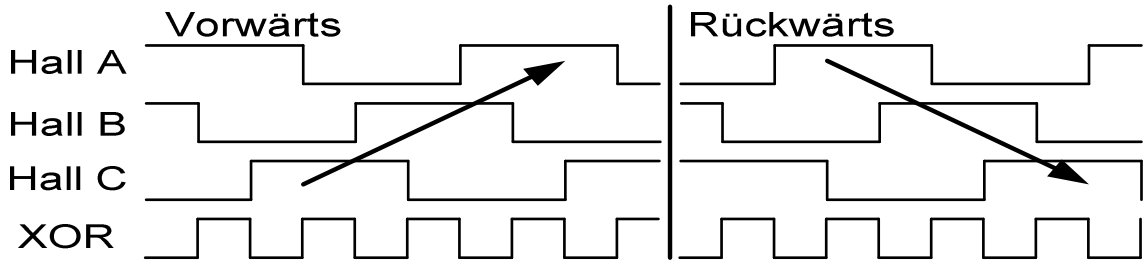

Bild 4.5.: *Hall-Sensor-Pulsfolgen bei Vorwärts- und Rückwärtsfahrt mit konstanter Geschwindigkeit, jede Bitkombinationsänderung bewirkt einen XOR-Pegelwechselt*

Bei einer bestimmten Hall-Sensoren-Kombination befindet sich der Rotor in derselben Position (*vgl.* ***Tab. 4.2. und Bild 4.5.***). Die Sequenz dieser Kombinationen bleibt bei der Fahrt ohne Richtungswechsel konstant.

Vorwärts				**Rückwärts**			
Hall A	Hall B	Hall C	Position	Hall A	Hall B	Hall C	Position
0	0	0	U	0	0	0	U
0	0	1	60°	1	0	1	360°
0	1	1	120°	1	0	0	300°
0	1	0	180°	1	1	0	240°
1	1	0	240°	0	1	0	180°
1	0	0	300°	0	1	1	120°
1	0	1	360°	0	0	1	60°
1	1	1	U	1	1	1	U

Tabelle 4.2.: *Sequenz der Hall-Sensor-Kombinationen bei Vorwärts- und Rückwärtsfahrt mit dazugehöriger Rotorposition*

Die Periode eines Hall-Sensors entspricht einer Motorumdrehung, je länger die Periodendauer ist, desto langsamer dreht sich der Motor und fährt das Fahrzeug, die kürzeren Periodendauer haben eine höhere Geschwindigkeit des Fahrzeugs zur Folge.

Sensoren-Ausgangspegel	5V
Tastverhältnis bei konstanter Geschwindigkeit	50%
Periodendauer bei Fahrzeugstillstand ($V = 0$)	∞ ms
Periodendauer bei max. Geschwindigkeit (*Tmin*)	5ms
Periodendauer bei min. Geschwindigkeit (*Tmax*)	180ms
Wechsel der Hall-Sensoren-Kombination bei V = 0	∞ ms
Wechsel der Hall-Sensoren-Kombination bei Vmax	0.85ms
Wechsel der Hall-Sensoren-Kombination bei Vmin	30ms

Tabelle 4.3.: *Kennwerte der Hall-Sensoren, mit Vmin ist die Geschwindigkeit gemeint, bei der die Motorkraft die Reibwiderstände der Fahrzeugmechanik überwindet*

Die Funktionsweise und die Kennwerte der Hall-Sensoren (*vgl.* ***Tabelle 4.3***) wurden durch Messungen ermittelt. Die Übersetzung von dem Motor auf die Fahrzeugräder (8.35:1) ist vom Hersteller gegeben[10]. Die gefahrene Strecke pro Periode eines Hall-Sensor-Pulses (*einer Motorwellenumdrehung*) wird bestimmt:

Durch Messung ermittelter Radradius $R_r = 0.0325m$ wurde zur Berechnung des Radumfangs in die **Gl. (4.1)** gesetzt:

$$U_r = 2\pi R_r = 0.2042m \tag{4.1}$$

Radumfang: $U_r = 0.2042m$, durch das Setzen dieser Konstante und der Antriebsübersetzung in die **Gl. (3.1)** wurde die gefahrene Strecke pro Wellenumdrehung des Motors berechnet:

$$S_{wu} = \frac{0.2042m}{8.35} = 0.024m \tag{4.2}$$

Bestimmt man die Hall-Sensor-Periodendauer, so erhält man die Zeit des zurückgelegten Weges pro Wellenumdrehung. Da die Periodendauer Tmax und Tmin bekannt sind (*vgl.* ***Tabelle 4.3***), werden mit **Gl. (3.2)** den Periodendauern entsprechende Fahrzeuggeschwindigkeiten (*Vmin und Vmax*) ausgerechnet:

$$Vmin = \frac{S_{wu}}{Tmax} = \frac{0.024m}{0.18s} = 0.13\frac{m}{s} = 0.48\ \frac{km}{h} \tag{4.3}$$

$$Vmax = \frac{S_{wu}}{Tmin} = \frac{0.024m}{0.005s} = 4.8\frac{m}{s} = 17.28\ \frac{km}{h} \tag{4.4}$$

Die Strecke, die ein Rad zwischen zwei Hall-Sensor-Kombinationen zurücklegt, ist mit 4mm bemessen (*24mm/6*), da die Periode eines Hall-Sensor-Rechteckpulses aus sechs solcher Kombinationen besteht (*vgl.* ***Bild 4.5.*** *XOR-Pegelwechsel*). Wenn man die Dauer des Kombination-Wechsels messen würde, käme man mit einem kleineren Timer aus. So ein Timer würde auch sechs Mal öfter Werte aktualisieren als einer, der nur die Periodendauer eines Hall-Sensors misst.

4.3. Identifikation der Regelstrecke durch Analyse des Antriebsverhaltens

Die Mechanik und das Antrieb des SoC-Fahrzeugs ist das Regelstreckengleid des zu projektierenden Geschwindigkeitsregelkreises. Zur Ermittlung der Regelstreckenkennwerte für eine MATLAB-Simulation wurde die Identifikation der Regelstrecke über die Sprungantwortaufnahme durchgeführt. Aus der Sprungantwortanalyse wurde die Regelstreckenart bestimmt und durch eine Übertragungsfunktion beschrieben.

Die Sprungantwort-Methode wird oft in der Praxis verwendet, wenn die Kennwerte der Regelstrecke unbekannt sind[23]. Es wurde eine sprungförmige Funktion auf den Eingang der Regelstrecke geben, das heißt den PWM-Wert von a = 7.5% (*Stillstand*) auf a = 5% (*Vmax*) ändern. Dabei wurde die Regelstreckenausgangsfunktion *v(t)*, über die Auswertung der Hall-Sensoren, aufgenommen.

Das Nexys2-Board wurde zur PWM-Vorgaben und zur Auswertung der Hall-Sensoren genutzt. Bei dem direkten Anschluss der drei Hall-Sensor-Pulse an das Board verringern die Wiederstände und Dioden der Pmods (*vgl.* ***Bild 2.2***) die 5V Pegel der Hall-Sensoren auf 3.3V, um FPGA zu schützen, dies führt zum Ausfall des Fahrtenstellers, da er die Hall-Sensor-Pulsänderungen nicht erkennt. Um die Hall-Sensoren auswerten zu können, ohne die Funktionalität des Fahrtenstellers zu stören, wurde, zwischen dem Motor und dem Board (*vgl.* ***Bild 4.6***), ein 74LVC245 Pegelwandler[24] eingesetzt.

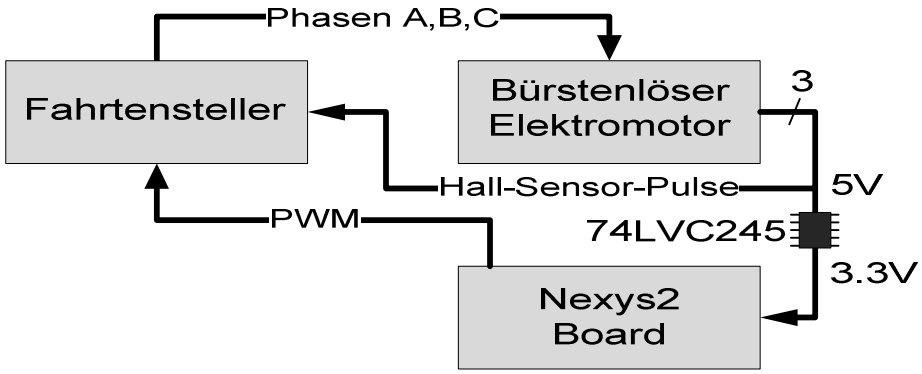

Bild 4.6.: *5V zu 3.3V Pegelwandler 74LVC245, Spannungsversorgung und Konfigurationssignale erhält der Baustein vom Nexys2-Board*

Zur Auswertung der Hall-SensorPulse mit FPGA, wurden zwei Ansätze angewendet:

- ***Ansatz1:*** Messung der Dauer des Kombination-Wechsels der Hall-Sensoren, die Zeit zwischen high und low Flanken des XOR Signals (*vgl.* ***Bild 4.5***).
- ***Ansatz2:*** Messung der Periodendauer nur eines Hall-Pulses, die Zeit zwischen high und high Flanken des Hall C Signals (*vgl.* ***Bild 4.5***).

Zur Realisierung beider Ansätze wurden zwei VHDL-Module entworfen.

Ansatz1: Messung des 1/6 der Hall-Sensor-Puls-Periode

Ablauf: Im abgesetzten Zustand wurde das Fahrzeug auf die maximale Geschwindigkeit beschleunigt, die Regelstercken-Eingangsfunktion ändert sich sprungförmig von a = 7.5% auf a = 5% PWM. Das PWM-Signal bleibt zwei Sekunden unverändert, in dieser Zeit werden die Hall-Sensor-Rechteckpulse mit einem VHDL-Modul (*vgl.* ***Bild 4.7***) ausgewertet.

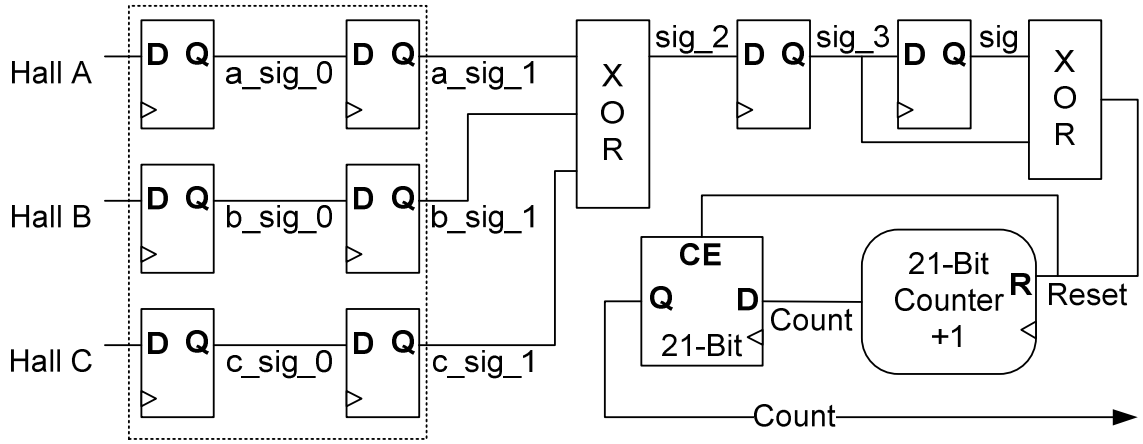

Bild 4.7.: *Modul zur Messung der Dauer des Kombinations-Wechsels der Hall-Sensoren, mit einem Synchronisationsblock (gestrichelt) zur Vermeidung der metastabilen Zustände*

Funktionalität: Sofern das erste Synchronisations-Flipflop in den metastabilen Zustand übergeht, kann sich dieser im Laufe der Taktperiode auflösen, und damit übernimmt das zweite Synchronisations-Flipflop einen korrekten Logikpegel[19]. So ist die Wahrscheinlichkeit, dass das erste XOR-Gatter undefinierte Eingangspegel erhält, erheblich reduziert. Das gesamte System wird mit f_{clk} = 50MHz getaktet, durch den Synchronisationsblock werden die Hall-Sensor-Signale um zwei Takte (*40ns*) verzögert.

Die Aufgabe des ersten XOR-Gatters ist es ein Signal zu erzeugen, wie im **Bild 4.5** dargestellt. Folgende zwei Flipflops und das zweite XOR-Gatter erzeugen bei jedem Flankenwechsel des sig_2 Pulses einen Impuls mit der Länge eines Taktes. Mit diesem Impuls erfolgt die Notation des Counter-Wertes in das Register und der 21-Bit-Counter-Reset.Da die Zeit zwischen dem Wechsel der Hall-Sensor-Kombinationen im Bereich von 0.85ms bis 30ms (*vgl.* ***Tabelle 4.3***) liegt und f_{clk} = 50MHz, wurde ein 21-Bit-Counter verwenden (*30ms / 20ns = 1500000*).

Software: Durch den Timer aus dem IP-Catalog wird die Interrupt Service Rutine (*ISR*) in konstanten (T_a = *17ms*) Zeitabschnitten zwei Sekunden lang angestoßen. Ihre Aufgabe besteht darin, das 21-Bit Register auszulesen und die Werte in einem Array mit der Kapazität von 150 Werten zu speichern. Wie im **Kap.2.1.2** beschrieben, wurden sie über den UART des Nexys2-Board abgeholt. Jede Array-Zahl, multipliziert mit 20ns, ergibt die Dauer der Hall-Sensor-Kombinationen. Die Strecke zwischen ihnen ist als 1/6 des gefahrenen Weges pro Motorwellenumdrehung (*24mm / 6 = 4mm*) bekannt. Dividiert man sie (*4mm*) durch die Zeit-Werte (*vgl.* ***Gl. (3.2)***) aus dem Array, hat man 150 Geschwindigkeitspunkte mit 17ms

Abstand zu einander. Sie ergeben den Beschleunigungsverlauf des Fahrzeugs in den zwei Sekunden, wie auf der Zeitachse (*vgl.* ***Bild 4.8.***) abgebildet. Die entstandene Kurve ist die Sprungantwort der Fahrzeugregelstrecke. Die Umwandlung der aufgenommenen Counter-Werte in die Zeit, die Berechnung der Geschwindigkeit und die grafische Darstellung der Beschleunigungskurve erfolgte mit Mikrosoft Excel.

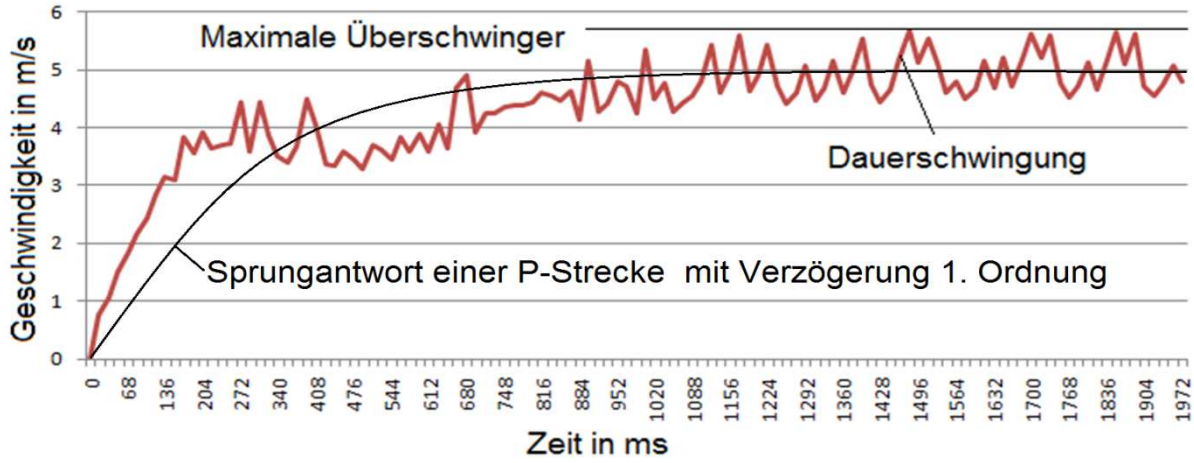

Bild 4.8.:*Sprungantwort der Regelstrecke (Kombinationsdauermessung) mit Beschleunigung auf die maximale Geschwindigkeit (100% PWM) bei Vorwärtsfahrt in abgesetztem Zustand*

Der Verlauf der Sprungantwort-Funktion ähnelt sich der einer P-Strecke mit Verzögerung 1. Ordnung (*PT_1-Glied*). Die Übertragungsfunktion des PT_1-Gliedes[21]:

$$G_s(s) = \frac{K_s}{sT_g + 1} \tag{4.5}$$

T_g ist die Ausgleichszeit und K_s der Übertragungsbeiwert der Strecke.

Fahrverhalten des Fahrzeugs: Nach dem Start des Programms beschleunigte das Fahrzeug ruckelfrei und ohne abzubremsen auf die maximale Geschwindigkeit. Die Geschwindigkeitsschwingungen, wie im **Bild 4.8** gezeigt, wurden nicht erkannt. Das deutet darauf hin, dass die Hall-Sensoren bei gleich bleibender Geschwindigkeit keine konstanten Zeitinterwalle liefern. Die Zeiten zwischen den Hall-Sensor-Kombinationen sollten 1/6 der Periode entsprechen, was jedoch nicht zutraf. Bei dieser Messung wiesen die Hall-Sensor-Werte eine mittlere Abweichung von 7% auf.

Schlussfolgerung: Die dauerschwingenden Zeitinterwalle werden die Regelung unnötig anstoßen, folglich gerät das gesamte Regelkreis-Systems in Schwingung und das Fahrzeug wird nie eine konstante Geschwindigkeit halten können. Deshalb wurde in einem weiteren Versuch die ganze Periode gemessen.

Ansatz2: Messung der ganzen Hall-Sensor-Periode

Der Ablauf dieses Experiments gleicht dem ersten, und das Verhalten des Fahrzeugs unterscheidet sich auch nicht, nur statt drei Hall-Sensoren nur ein ausgewertet wird. Die RTL-Struktur des VHDL-Moduls (*vgl.* ***Bild 4.9***) misst die Periodendauer des Hall-C-Sensors.

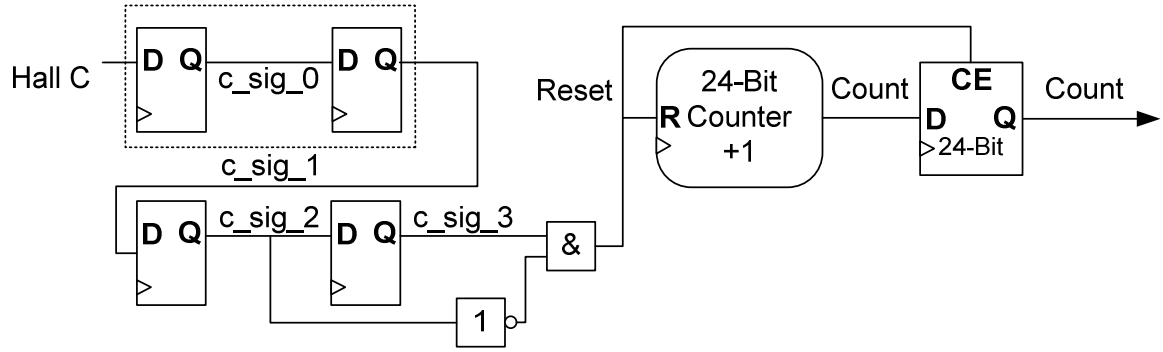

Bild 4.9.: *Modul zur Messung der Periodendauer eines Hall-Sensors mit einem Synchronisationsblock (gestrichelt), Systemtakt f_{clk} = 50MHz*

Funktionalität: Der Inverter, das UND-Gatters und die zwei Flip-Flops erzeugen bei jeder aufsteigenden Flanke des Hall-Sensor-Rechteckpulses einen taktlangen (20ns) Impuls, bei dem der Zählerwert notiert und zurückgesetzt wird. Bei diesem Ansatz werden ein 24-Bit-Zähler und ein 24-Bit-Register verwendet, da die Periodendauer bis zu 180ms lang werden (*vgl.* ***Tabelle 4.3***), das entspricht dem Zählerwert 9000000 bei f_{clk} = 50 MHz. Die Software ist dieselbe wie bei vorherigem Versuch. Die 150 Zähler-Werte wurden über den UART vom Nexys2-Board abgeholt und im Excel verarbeitet. Zur Berechnung der Geschwindigkeitspunkte nutzte man die Strecke, die das Fahrzeug in einer Motorwellen-umdrehung zurücklegt und die einer Pulsperiode des Sensor-Pulses entspricht (*24mm*).

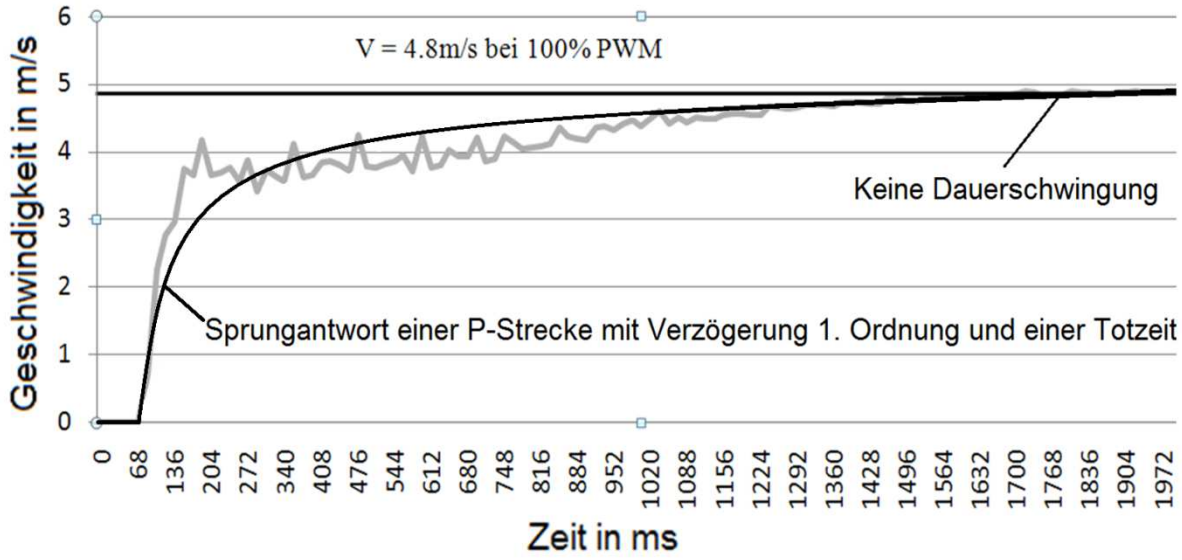

Bild 4.10.: *Sprungantwort der Regelstrecke (Periodendauermessung) mit Beschleunigung auf die maximale Geschwindigkeit (5% PWM) bei Vorwärtsfahrt im abgesetzten Zustand*

Das Verhalten der Ausgangsfunktion wird annäherungsweise durch die Übertragungsfunktion der Sprungantwort einer P-Strecke mit Verzögerung 1. Ordnung (*PT_1-Glied*) und einer Totzeit beschrieben[21]:

$$G_s(s) = \frac{K_s}{sT_g + 1} \cdot e^{-T_t s} \tag{4.6}$$

T_g ist die Ausgleichszeit, T_t die Totzeit, K_s der Übertragungsbeiwert der Strecke. Da die Zeit zwischen dem Kombination-Wechsel sechs Mal kleiner ist als die Periodendauer eines Sensorsignals, dauert es länger, bis zur ersten Wertmessung, so entsteht die Totzeit. Es wurde festgestellt, dass dieser Messungsansatz keine Überschwinger aufweist, die maximale Geschwindigkeit (*4.8m/s*) wird erreicht und gehalten. Dieser Antwort der Regelstrecke (*vgl.* ***Bild 4.10***) stimmt mit dem Fahrverhalten des Fahrzeugs überein, die mittlere Abweichung der Sensor-Werte bei der konstanten Geschwindigkeit beträgt nur 0.3%.

Schlussfolgerung: Die Hauptaufgabe des Reglers ist es, die konstante Geschwindigkeit zu halten selbst beim Nachlasen der Akkuspannung. Da beim zweiten Experiment die Hall-Sensor-Werte nur kleine Abweichungen aufwiesen, stabilisiert sich das gesamte System, deswegen wird dieser Ansatz zur Ermittlung der aktuellen Geschwindigkeit (v_*ist*) realisiert.

Die beiden oben beschriebenen Ansätze basieren auf Periodendauermessung, ein weiterer Ansatz gründet auf der Signal-Pulszählung. Dieses Verfahren wurde nicht behandelt, da die Hall-Sensoren bei niedrigen Geschwindigkeiten zu wenig Pulse liefern (*z.B. bei Vmin 1 Puls in 30ms*).

5. Regelungstechnischer Entwurf des Geschwindigkeitsreglers

Der Erfolg beim Entwurf einer Regelung hängt direkt von Kenntnissen über die Regelstrecke ab. Wenn ein ausreichend genaues Modell der Regelstrecke vorliegt, kann eine anspruchsvolle Einstellung des Reglers erfolgen. In den Fällen, in denen die mathematische Beschreibung der Regelstrecke nicht bzw. nur angenähert bekannt ist, haben sich die empirischen Einstellregeln mit Erfolg bewährt[23].

In diesem Kapitel sind zwei praktischen Verfahren zur Bestimmung der Regler-Parameter vorgestellt (*vgl.* ***Kap. 5.2 und 5.3***): Wendetangenten- und T-Summen-Verfahren, die mit der Sprungantwort der Regelstrecke arbeiten. Durch Analyse Regler- und Fahrzeugenschaften wurde für die Geschwindigkeitsregelung ein PI-Regler ausgewählt. Der PI-Regelungsalgorithmus wurde diskretisiert und in den Regelkreisentwurf eingebaut. Daraus entstandene Regelungssystem wurde anschließend im MATLAB mit Simulink-Programm-Tool aufgebaut und simuliert.

5.1. Auswahl des PI-Reglers und seine Grundlagen

Der Hauptteil des Geschwindigkeitsregelkreises ist der Regler, er erfasst die Regelgröße (*aktuelle Geschwindigkeit v_ist*), vergleicht sie mit einem konstanten Sollwert (*vorgegebene Geschwindigkeit v_soll*) und bildet eine Stellgröße zur Geschwindigkeitssteuerung (*y*), sodass die Geschwindigkeitsdifferenz (*e*) gegen null geht (*vgl.* ***Bild 3.5***). Zur Regelung einer PT1-Regelstrecke mit einer Totzeit eignen sich PI- und PID-Regler[21]. Um die Entscheidung für einen der Regler zu treffen, wurden ihre Eigenschaften aufgelistet:

PID-Regler ist universal und genau, er hat die besten Attribute aller Typen in sich vereint. Die Regler mit D-Anteil sind die schnellsten und kommen meistens dort zum Einsatz, wo ein dynamisches Verhalten und rasche Reaktion des Systems gefordert sind[25].

PI-Regler haben keinen D-Anteil (*vgl.* ***Bild 5.1***), was sich negativ auf seine Reaktionsgeschwindigkeit auswirkt. Wegen des I-Anteils sind diese präzise und genau, deswegen für einfache Regelaufgaben gut geeignet[25].

Die Schnelligkeit des PID-Reglers führe dazu, dass die Anlaufströme des Motors in höherem Bereich legen. Diese Ströme sind mit hohem Energieverbrauch eng verbunden, was die Betriebszeit des autonomen Fahrzeugs verkürzt. Wegen schnell aufsteigender Geschwindigkeit werden die Räder des Fahrzeugs durchdrehen, und das Auto gerät aus der Spur. Zur Begrenzung der Anlaufströme und Gewährleistung der Sicherheit des Fahrzeugs wurde auf D-Anteil des Reglers verzichtet.

Der PI-Regler kombiniert die Eigenschaften von P- und I-Regler, alleinstehend haben sie folgende Nachteile: Der P-Regler kann die Geschwindigkeitsdifferenz nicht ganz ausregeln. Der I-Regler dagegen regelt zwar ganz aus, jedoch neigt zum Schwingen und reagiert langsam. In der Kombination dieser beiden kompensieren sich ihre Nachteile weitgehend. Da der PI-Regler praktisch aus einem P-Regler mit einem geringen I-Anteil besteht, lässt sich

bei einer Störung die Ist-Geschwindigkeit schnell auf die Soll-Geschwindigkeit bringen, ohne dass starke Schwingungen auftreten[25].

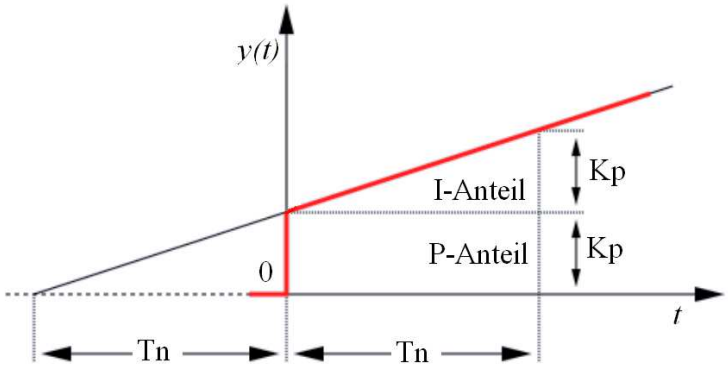

Bild 5.1.: *Sprungantwort eines PI-Reglers mit Nachstellzeit T_n und Proportionalbeiwert K_p*[21]

Die Bezeichnung PI bedeutet, dass der Regler die Eingangsgröße verstärkt und integriert (*vgl.* ***Bild 5.1***). Die Gleichung des PI-Reglers ist durch die Summe der P- und I-Ausgangsgrößen beschrieben[21]:

$$y(t) = K_p \cdot e(t) + \frac{K_p}{T_n} \cdot \int e(t)dt \tag{5.1}$$

K_p wird vor der Klammer gezogen:

$$y(t) = K_p \cdot \left(e(t) + \frac{1}{T_n} \cdot \int e(t)dt\right) \tag{5.2}$$

Aus Laplace-Transformation folgt die Übertragungsfunktion des PI-Reglers:

$$G_r(s) = \frac{y(s)}{e(s)} = K_p \left(1 + \frac{1}{s \cdot T_n}\right) \tag{5.3}$$

Hierin sind e(s) die Eingangs- und y(s) die Ausgangsgröße des PI-Reglers, K_p der Proportionalbeiwert und T_n die Nachstellzeit. K_p und T_n sind die beiden einstellbaren Parameter des PI-Reglers, die das Regelverhalten definieren[21]. Die anwendungsspezifisch gewählten Parameter beseitigen die Geschwindigkeitsdifferenz schnell und vollständig.

5.2. PI-Regler-Parametrisierung durch Wendetangenten-Verfahren

Aus der Sprungantwort wurden nicht nur die Übertragungsfunktion der Regelstrecke als ein PT_1-Glied mit Totzeit identifiziert (*vgl.* ***Bild 4.10***), sondern die zur PI-Regler-Parametrisierung nötigen Konstanten, wie die Totzeit T_t, die Ausgleichszeit T_g und der Übertragungsbeiwert mittels Wendetangente-Verfahren[21] ausgelesen.

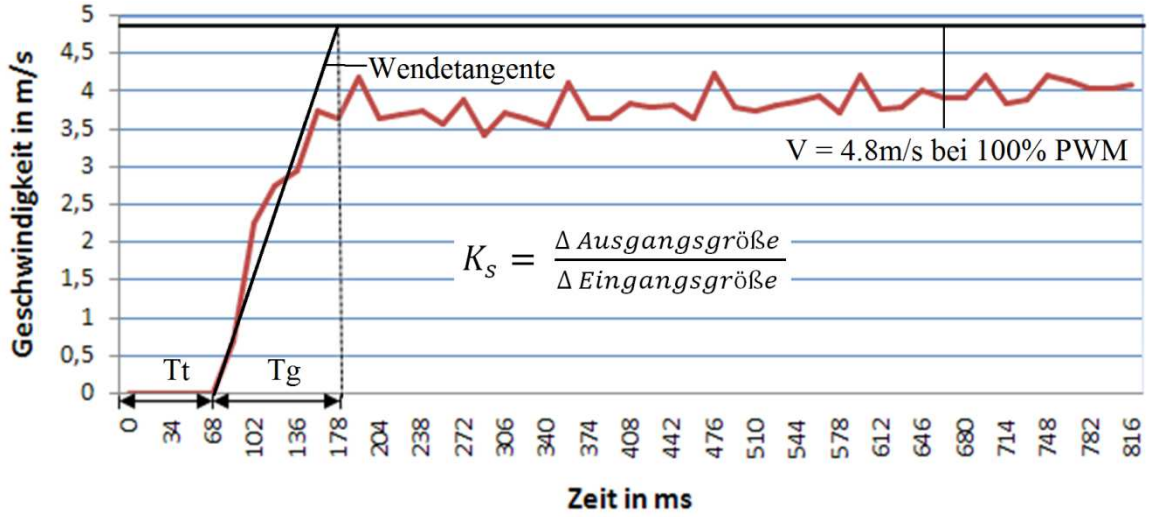

Bild 5.2.: *Bestimmung der Totzeit (T_t = 68ms) und der Ausgleichszeit (T_g = 110 ms) mit dem Wendetangenten-Verfahren aus der Sprungantwort der Fahrzeugregelstrecke*

Die Wendetangente (*vgl.* ***Bild 5.2***) wurde durch den Wendepunkt der Sprungantwort gelegt, sie schneidet die Zeitachse im Totzeitpunkt T_t = *68ms*. Der Ausgangswert im Ruhezustand beträgt 4.8m/s, dargestellt durch jeweils eine horizontale Gerade im **Bild 4.10 und 5.2**. Die Tangente schneidet sie im Ausgleichszeitpunkt T_g = *178ms – 68ms = 110ms*. Der Übertragungsbeiwert der Regelstrecke besagt: K_s = *(4.8m/s – 0m/s)/(100% – 0%) = 4.8m/s.* Das Verhalten: Ausgleichszeit/Totzeit ist die Regelbarkeit des Geschwindigkeitsregelkreises.

Durch Untersuchung der Amplituden- und Phasengängen der verschiedenen aufgeschnittenen Regelkreise gelangten die Forscher zum Erkenntnis, dass die günstigste Regeleinstellung von der Regelbarkeit (T_g/T_t) und dem Übertragungsbeiwert (K_s) der Regelstrecke abhängig ist[21]. Detaillierte Empfehlungstabellen zu Regler-Parametereinstellungen sind neben anderen die Ergebnisse dieser Forschungen.

Zur Parametrisierung des PI-Reglers wurden die Ziegler-Nichols- und Chien-Hornes-Reswick-Empfehlungen[21] verwendet:

Empfehlung	Ziegler-Nichols		Chien-Hornes-Reswick	
Parameter	$K_p / \frac{s}{m}$	T_n / s	$K_p / \frac{s}{m}$	T_n / s
PI-Regler	$\frac{0.9 \cdot T_g}{K_s T_t}$	$3.3 \cdot T_t$	$\frac{0.35 \cdot T_g}{K_s T_t}$	$1.2 \cdot T_t$
Werte	0.30331	0.2244	0.11795	0.0816

Tabelle 5.1.: *Parameterberechnung für die Geschwindigkeitsregelung*

5.3. PI-Regler-Parametrisierung durch T-Summen-Verfahren

Bei dem T-Summen-Verfahren[21], im Unterschied zum Wendetangenten-Verfahren wurde in der Sprungantwortgrafik (*vgl.* ***Bild 5.3***) eine senkrechte Linie zur Bestimmung der „Summe der Zeitkonstanten T_Σ“ gebaut. Sie bildet zwei ungefähr gleiche Flächen S1 und S2 und schneidet die Zeitachse im T_Σ – Punkt.

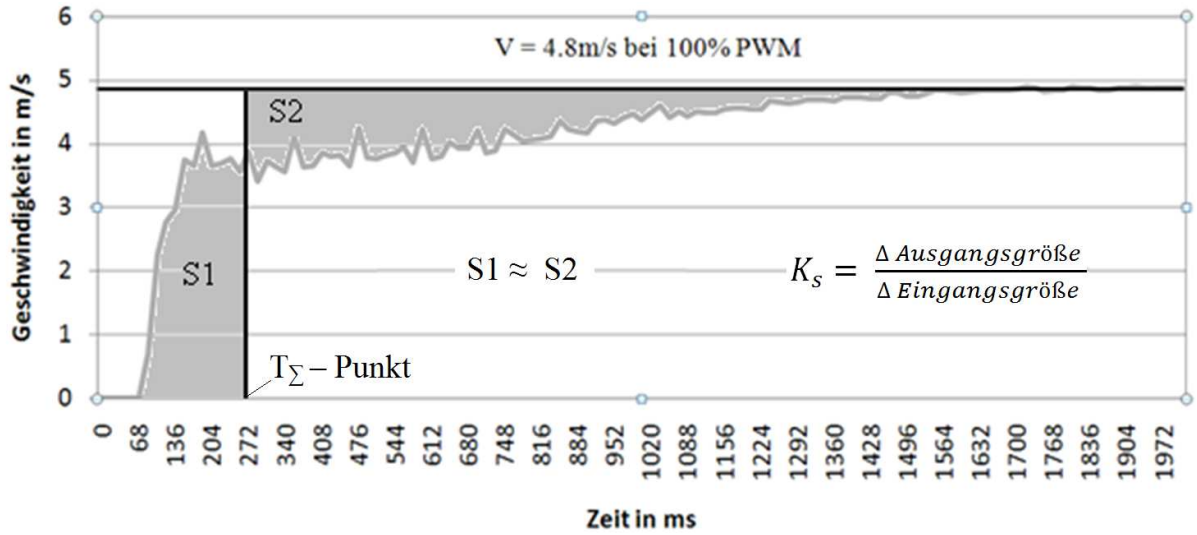

Bild 5.3.: *Die Summe der Zeitkonstanten (T_Σ = 272), abgelesen mit dem T-Summen-Verfahren aus dem Sprungantwort der Fahrzeugregelstrecke*

Daraus folgt ein Einstellverfahren, von U. Kuhn eingeführt[21], nach seiner Empfehlung werden die Regler-Parameter mit dem Übertragungsbeiwert und der Zeitkonstante T_Σ berechnet (*vgl.* ***Tabelle 5.2***).

Empfehlung	Kuhn	
Parameter	$K_p / \frac{s}{m}$	T_n / s
PI-Regler	$\frac{0.5}{K_s}$	$0.5 \cdot T_\Sigma$
Werte	0.10417	0.136

Tabelle 5.2.: *Parameterberechnung für die PI-Geschwindigkeitsregelung nach T-Summen-Verfahren und Empfehlungsregel von U. Kuhn*

5.4. PI-Regler-Diskretisierung und Regelkreissimulation mit MATLAB

Die Übertragungsfunktion des PI-Reglers (*vgl.* ***Gl.5.3***) ist eine kontinuierliche, die der Beschreibung analoger PI-Regler dient. Digitale Systeme arbeiten jedoch diskret, aus dem Grund wurde der Regelalgorithmus diskretisiert. Zur Abtastung der Regelgröße und zur Berechnung der z-Übertragungsfunktion des PI-Reglers wurde die Abtastperiode T_a ermittelt.

In der Praxis orientiert man sich an der Faustregel, die besagt, dass die Abtastzeit als ein Zehntel der Ausgleichszeit gewählt werden soll ($T_a = T_g/10$)[23]. Da diese Konstante ($T_g = 110ms$) aus der Sprungantwortanalyse bekannt ist, wäre die optimale Abtastperiode 11ms lang. Der Motor des Fahrzeugs wird jedoch mit dem Fahrtensteller über ein PWM-Signal gesteuert, dessen Periode T_{pwm} = 17ms lang ist (*vgl.* ***Tabelle 4.1***). Mehrmalige Änderungen der Stellgröße innerhalb der PWM-Periode erkennt der Fahrtensteller nicht, und sie geht verloren. Die Stellgrößenänderung findet einmal in T_{pwm} = 17ms statt, somit ist die Abtastperiode $T_a = 17ms$.

Mit dem Einsatz von z-Transformation lassen sich zeitkontinuierliche Systeme in zeitdiskrete transformieren. Die Integration nach Trapezregel (*Tustin Formel*) stellt einen Zusammenhang zwischen s- und z-Ebenen her, seine Beziehung beschreibt folgender Formel[20]:

$$s = \frac{2(z-1)}{T_a(z+1)} \tag{5.4}$$

Durch das Setzen der Äquivalenz-Beziehung in die Übertragungsfunktion des IP-Reglers (***Gl.5.3***) erhält man die z-Übertragungsfunktion:

$$G_r(z) = \frac{y(z)}{e(z)} = K_p\left(1 + \frac{1}{\left(\frac{2(z-1)}{T_a(z+1)}\right)T_n}\right) = K_p\left(1 + \frac{T_a(z+1)}{2T_n(z-1)}\right) \tag{5.5}$$

Die Übertragungsfunktion der Regelstrecke (*vgl.* ***Gl.4.6***), die z-Übertragungsfunktion des PI-Reglers (*vgl.* ***Gl.5.5***), die Regler-Parameter (*vgl.* ***Tab.5.1*** *und* ***Tab.5.2***) zur Konstruktion des Geschwindigkeitsregelkreises wurden ermittelt. Als Simulations- und Aufbauwerkzeug wurde MATLAB gewählt, ein von Industrie und Forschung anerkanntes Programm. Es verfügt über ein Simulink-Programm-Tool, zur grafischen Darstellung sowie Analyse der Regelkreise. Dieses Werkzeug ist gut dokumentiert und stellt viele Bibliotheken für regelungstechnische Anwendungen zur Verfügung[21].

Die Regelstrecke wurde mit Simulink aufgebaut und mit einer sprungförmigen Funktion simuliert, um die Richtigkeit des Regelstreckenverhaltens zu überprüfen. Die Parameter der Übertragungsfunktion der Regelstrecke, Übertragungsbeiwert K_s = 4.8m/s, Totzeit T_t = 0.068s und Ausgleichszeit T_g = 0.11s wurden im **Kap.5.2** identifiziert, mit dem Setzen dieser Konstanten in die **Gl.4.6** entsteht folgende Übertragungsfunktion der Regelstrecke:

$$G_s(s) = \frac{y(z)}{e(z)} = \frac{K_s}{sT_g + 1} \cdot e^{-T_t s} = \frac{4.8m/s}{s \cdot 0.11s + 1} \cdot e^{-s \cdot 0.068s} \qquad (5.6)$$

Das Verhalten der Regelstrecke wurde mit Elementen aus dem Simulink Library Browser nachgebaut (*vgl.* ***Bild 5.4***). Für den Bruch der Gleichung wurden ein Transfer Fcn Block, für e-Funktion ein Transport Delay Block, für sprungförmige Eingangsfunktion ein Step Block und zur grafischen Darstellung der Ausgangsfunktion ein Scope Block verwendet.

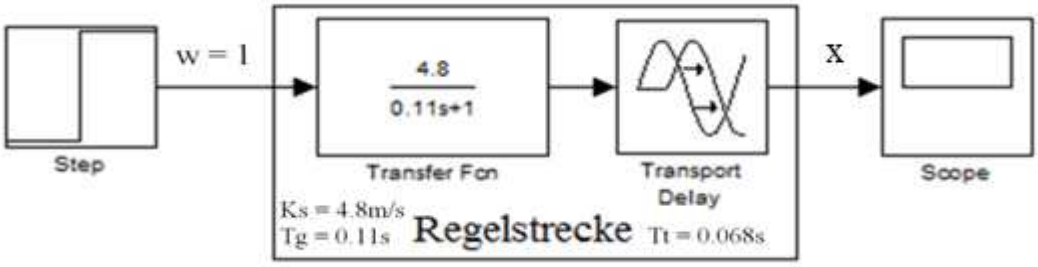

Bild 5.4.: *Simulink Modell zur Simulation der Regelstreckensprungantwort, die Blöcke der Regelstrecke realisieren das Verhalten aus* ***Gl.5.6***

Die Ausgangsfunktion der simulierten Regelstrecke (*vgl.* ***Bild 5.5***) wurde konstatiert und mit der experimentell aufgenommenen Sprungantwort der Fahrzeugregelstrecke verglichen.

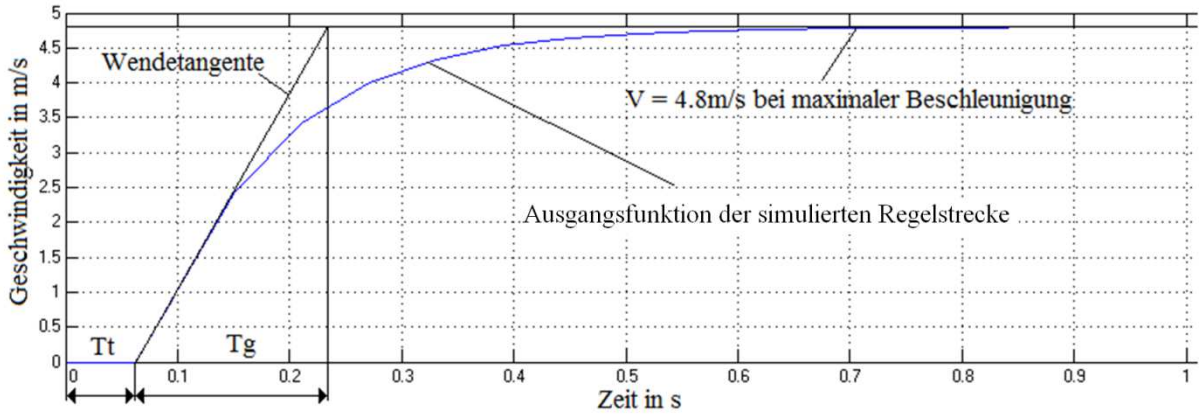

Bild 5.5.: *Sprungantwort der simulierten Regelstrecke, T_t = 0.068ms, T_g = 0,17s*

Die durch Simulation aufgenommene Sprungantwort weist Ähnlichkeiten mit der experimentellen auf. Die simulierte Regelstrecke ist auch ein PT1-Glied mit Totzeit, deren stationärer Wert 4.8m/s beträgt. Die Grafiken unterscheiden sich (*vgl.* ***Bild 5.3*** *und* ***5.5***):

- Die Ausgleichszeit (T_g) ist bei der simulierten Regelstrecke größer.
- Die Stationierung der Regelgröße bei der Simulation 0.9s bei dem Experiment 1.7s.
- Die simulierte Sprungantwort weist keine Überschwinger und Störungen auf.

Diese Differenzen sind dadurch entstanden, dass auf die Simulationsregelstrecke keine physikalischen Kräfte einwirken. Das in Simulink gebaute Modell wurde weiter zur Regelkreissimulation verwendet, da es, durch sein ähnliches Regelstreckenverhalten die Grundvoraussetzungen zu Simulationszwecken erfüllt.

Der digitale PI-Regler wurde durch die z-Übertragungsfunktion beschrieben (*vgl.* ***Gl.5.5***), im Simulink zu Abbindung solcher Funktionen wird ein Discrete Transfer Fcn Block verwendet. Bei der Initialisierung des Blocks wurde die Abtastzeit T_a auf 0.017s eingestellt. Zur Normierung der Blockparamater wurden der Zähler und Nenner der z-Übertragungsfunktion der Musterform "bz + a" angepasst, a und b sind Konstanten.

Schritte zur Umformung der z-Übertragungsfunktion des PI-Reglers **Gl.5.5**:

1. Den Bruch zum selben Nenner führen und mit K_p multiplizieren:

$$G_r(z) = \frac{y(z)}{e(z)} = \frac{2T_nK_p(z-1) \; + \; T_aK_p(z+1)}{2T_n(z-1)} \tag{5.7}$$

2. Im Zähler z ausklammern, im Nenner z mit der Klammer multiplizieren:

$$G_r(z) = \frac{y(z)}{e(z)} = \frac{\overbrace{\left(2T_nK_p \; + \; T_aK_p\right)}^{b1} z - \overbrace{\left(2T_nK_p \; - \; T_aK_p\right)}^{a1}}{\underbrace{2T_n}_{b2} z \; - \; \underbrace{2T_n}_{a2}} \tag{5.8}$$

Die Nachstellzeit T_n, die Abtastperiode T_a, der Proportionalbeiwert K_p wurden nach Wendetangenten- und T-Summen-Verfahren berechnet (*vgl.* ***Tabelle 5.1*** *und* ***5.2***). Die **Gl.5.8** erfüllt die geforderte Form, die Koeffizienten a1, b1 und a2, b2 lassen sich aus den Regelstreckenkonstanten berechnen (*vgl.* ***Tabelle 5.3***):

Verfahren	***Wendetangenten***		***T-Summen***
Koeffizient	Ziegler-Nichols	Chien-Hornes-Reswick	Kuhn
$2T_n$ / s	0.4488	0.1632	0.272
$2T_nK_p \; + \; T_aK_p$ / s^2/m	0.141281798	0.02125459	0.03010513
$2T_nK_p \; - \; T_aK_p$/ s^2/m	0.130969258	0.01724429	0.02656335

Tabelle 5.3.: *Werte der Koeffizienten für die z-Übertragungsfunktion (vgl.* ***Gl.(5.8)****), die das PI-Regler-Verhalten in der Simulation realisiert.*

Bei der Konstruktion des Simulations-Geschwindigkeitsregelkreises wurde zur Bildung der Regelgrößendifferenz ein Sum Block aus dem Simulink Library Browser verwendet. Die Eingänge dieses Blocks sind der Soll- und Istwert der Geschwindigkeit. Die Istwerterfassung realisierte man durch einen Rückführungspfad. Die neuen Komponenten wurden in das vorhandene Simulink-Modell (*vgl.* ***Bild 5.4 und 5.6***) integriert.

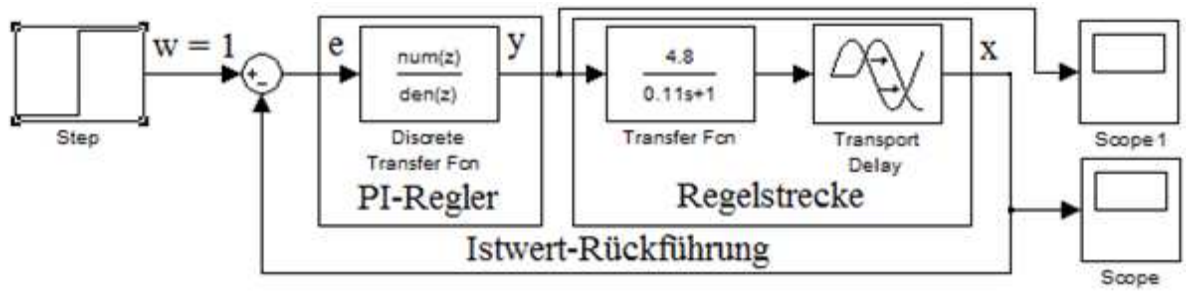

Bild 5.6.: *Regelkreismodell mit der z-Übertragungsfunktion des PI-Reglers und der Übertragungsfunktion der PT1-Regelstrecke mit Totzeit*

Der Sollwert wurde auf die maximale Fahrzeuggeschwindigkeit (*Vmax = 4.8m/s*) eingestellt. Da man drei verschiedene Empfehlungen für die PI-Regler-Einstellung berechnete, wurde zur Zwecken der Darstellung aller Simulationsergebnissen in einer Grafik der Regelkreis verdreifacht. Multiplexern führten die Ausgänge des Stell- und des Regelgrößen zu den Scope-Blocks. Bei der Simulation waren alle Regelkreise, bis auf die Einstellparameter der PI-Regler identisch.

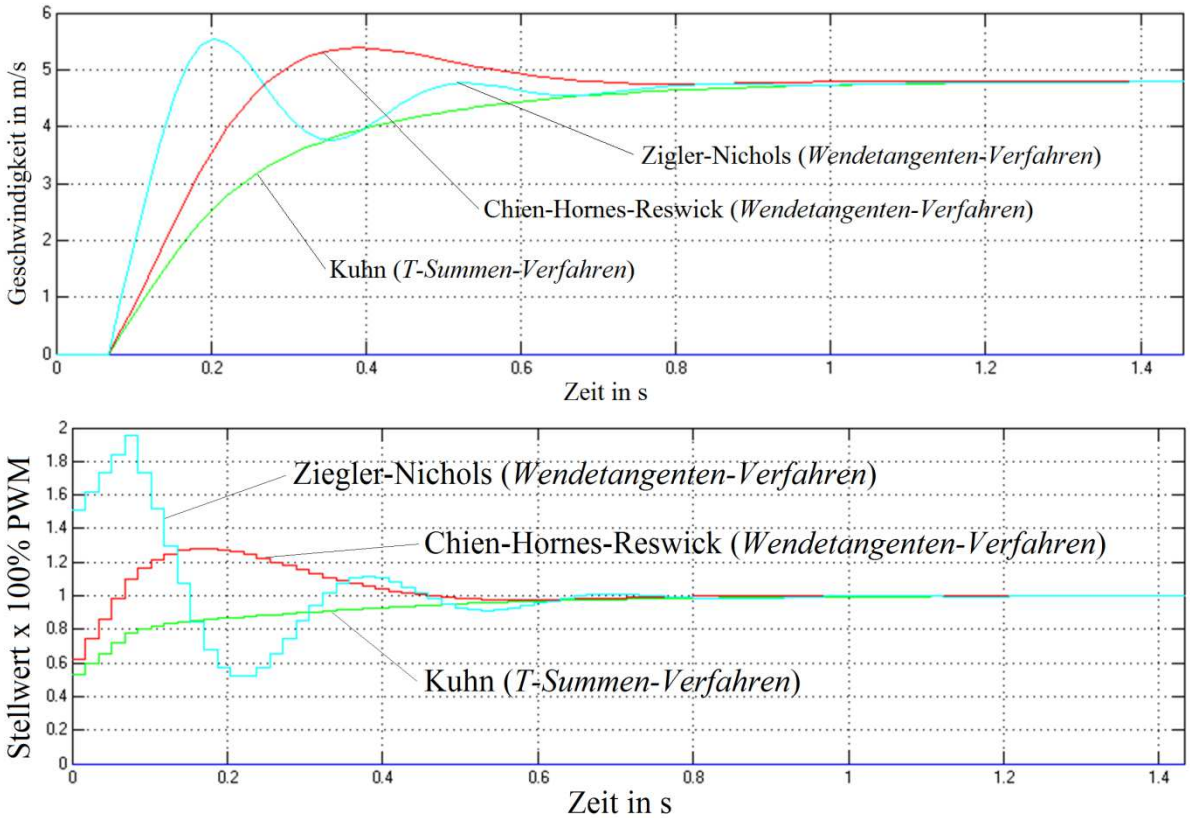

Bild 5.7.: *Sprungantworte der simulierten Regelkreise (oben) und PI-Regler (unten) mit drei Einstellempfehlungen für PI-Regler-Parameter*

Nach Ziegler-Nichols- und Chien-Hornes-Reswick-Empfehlungen eingestellte PI-Regler weisen bei den Sprungantworten die Überschwinger auf. Die letzte Sprungantwort ist leicht gedämpft und hat die größte Ähnlichkeit mit der experimentell aufgenommenen. Die nach Kuhn berechneten Parameter bekamen die höchste Priorität für die Parametrisierung des in VHDL-Entworfenen PI-Reglers zur Geschwindigkeitsregelung.

6. RTL-Modellierung des Geschwindigkeitsreglers

Dieses Kapitel stellt die RTL-Strukturen der einzelnen VHDL-Module aus der Top-Entity (*vgl.* ***Bild 6.1***) des Geschwindigkeitsreglers dar. Deren Ein- und Ausgängen sowie die Logik und Speicherlelemente werden präsentiert. Es wird gezeigt, wie man ausgehend von einem in C-Code beschriebenen Algorithmus ein Prozessor-Element (*vgl.* ***Kap. 3.2***) entwirft.

Das System besteht aus vier Hauptkomponenten (*vgl.* ***Bild 3.3***), wobei der Dividierer und der PI-Regler in Steuer- und Datenpfade unterteilt sind (*vgl.* ***Bild 6.1*** *und* ***Tabelle 6.1***). Die Top-Entity zur Geschwindigkeitsregelung verbindet vier Komponenten der oberen Ebene *HSA_DIV*, *PWM-Generator*, *PI_DATA*, *PI_FSM* und drei der unteren *Hall-Sensor-Auswerter*, *DATA*, *FSM*, insgesamt sind es acht VHDL-Dateien.

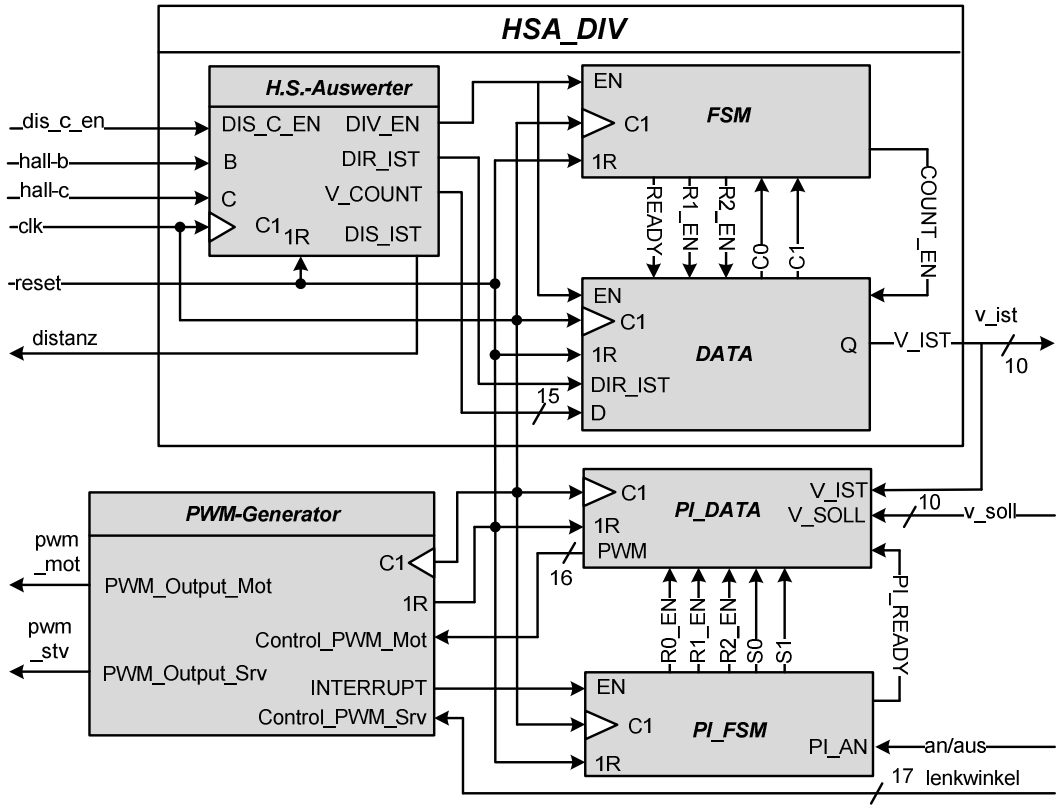

Bild 6.1.: *Top-Entity-Struktur des Geschwindigkeitsreglers. FSM und DATA bilden den Dividierer, PI_FSM und PI_DATA den PI-Regler. Der Hall-Sensor-Auswerter und Dividierer wurden zur einer Entity HSA_DIV zusammengefasst, welche die Ist-Geschwindigkeits- und die Distanzberechnung realisiert.*

clk	Systemtaktsignal 50MHz→Periodendauer 20ns, 50% Torverhältnis
Reset	Setzt synchron das gesamte System in den Anfangszustand
hall-b, hall-c	Externe Hall-Sensor-Pulse werden zur Distanz- Richtung und Ist-Geschwindigkeit- Auswertung verwendet
dis_c_en	Freigabe des Zählers, der die Distanz-Messung ausführt
v_ist	Aktuelle Fahrzeuggeschwindigkeit in cm/s
distanz	Gefahrene Fahrzeugstrecke in mm

v_soll	Vorgegebene Fahrzeuggeschwindigkeit in cm/s
an/aus	Freigabe des PI-Reglers
lenkwinkel	Stellgröße für PWM des Lenkwinkelstellers
pwm_mot	PWM-Ansteuerung des Fahrtenstellers
pwm_srv	PWM-Ansteuerung des Lenkwinkelstellers

Tabelle 6.1.: *I/O-Signale der Top-Entity. PWM-Perioden sowohl für den Fahrten- als auch für den Lenkwinkelsteller sind identisch, so wurden sie in einem PWM-Generator realisiert.*

6.1. RTL-Modellierung des PWM-Generators

Der PWM-Generator (*vlg.* ***Bild 6.2***) erzeugt Signale (*pwm_mot, pwm_srv*) zur Ansteuerung des Fahrten- und den Lenkwinkelstellers (*vgl. Kennwerte in* ***Tabelle 4.1***), und alle T_{PWM} = 17ms einen taktlangen $1/f_{clk}$ = 20ns Interrupt-Impuls zur Abtastung der Ist-Geschwindigkeit. Die Eingänge des PWM-Generators sind CLK, Reset und die Stellgrößen zur Tastverhältnismanipulation der PWM-Pulse.

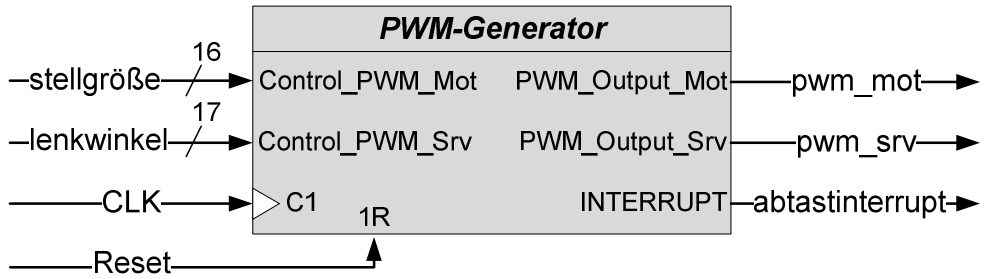

Bild 6.2.: *PWM-Generator wandelt die Stellwerte in die PWM-Rechteckpulse und erzeugt den Abtastinterrupt.*

Die Größe des Aufwärtszählers zur PWM-Signalerzeugung ergibt sich aus der PWM-Periodendauer T_{PWM} und der Systemtaktfrequenz f_{clk}.

$$PWM\_ZS_{max} = \frac{T_{PWM}}{\frac{1}{f_{clk}}} = \frac{17ms}{\frac{1}{50MHz}} = 850000 \rightarrow 20Bit \qquad (6.1)$$

Da die Tastverhältnisintervalle der PWM-Signale (*vgl.* ***Tabelle 4.1***) im Bereich 5% bis 10% liegen, ergeben sich folgende Zählerwerte:

- Vmax vorwärts ist das Tastverhältnis a = 5% → *42500*;
- Vmax rückwärts ist das Tastverhältnis $a = 10\% \rightarrow 85000$;
- V = 0 ist das Tastverhältnis $a = 7.5\% \rightarrow 63750$.

Bei PWM-Pulsen mit Tastverhältnisintervall $7.38\% \leq a \leq 7.62\%$ reicht die Drehkraft des Motors nicht aus, um die Widerstandskräften der Fahrzeugmechanik zu überwinden, folglich setzt sich das Fahrzeug nicht in Bewegung.

Die Funktionalität des PWM-Generators zur Abtastinterrupt- und PWM-Pulserzeugung (*vgl.* ***Bild 6.3***) wurde mit einem 20Bit breiten Aufwärtszähler realisiert, der mit jeder Taktflanke seinen Stand inkrementiert. Sobald der PWM_ZS_{max} erreicht ist, wird ein taktlanger $1/f_{clk}$ = 20ns Interrupt-Impuls erzeugt und der PWM-Ausgang auf „high“ gesetzt.

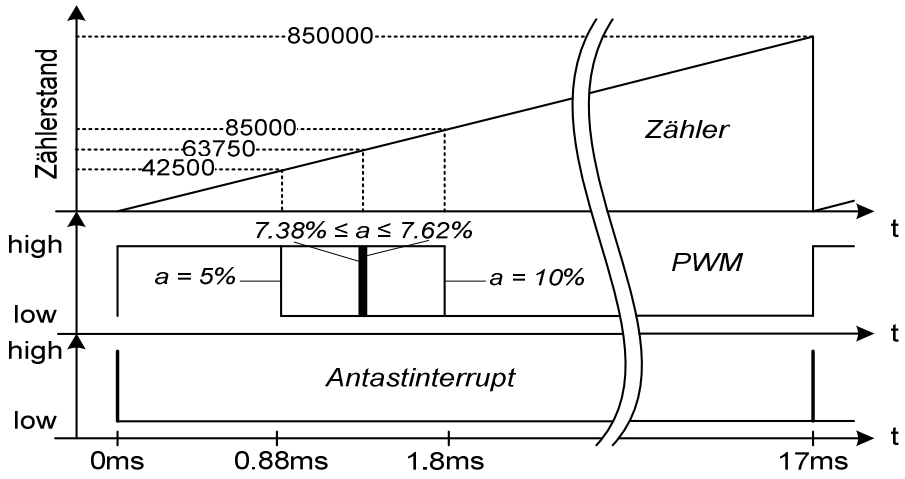

Bild 6.3.: *Interrupt- und PWM-Pulserzeugung mit einem Aufwärtszähler*

Die Stellgrößen des Lenkwinkels und der Geschwindigkeit haben folgende Kennwerte:

- Control_PWM_Srv: [42500; 85000], 5% ≤ *a* ≤ 10%, 17Bot-Vektor
- Control_PWM_Mot: [0; 42500], 0% ≤ *a* ≤ 5%, 16-Bit-Vektor

Der PWM-Ausgangspegel bleibt solange „high“, bis der vorgegebene Stellwert erreicht ist. Für den PWM-Puls des Lenkwinkelstellers werden diese Werte direkt aus einem Software-Register ausgelesen (*vgl.* ***Bild 3.2***). Die Zählerwerte für den Fahrtensteller versendet der Geschwindigkeitsregler (*vgl.* ***Bild 3.3***). Das Stellgrößenintervall der Geschwindigkeit wird mit einem Offset auf das Tastverhältnis-Intervall von 5% bis 10% angepasst. Diese Aufgabe übernimmt der Subtrahierer (*vgl.* ***Bild 3.4***).

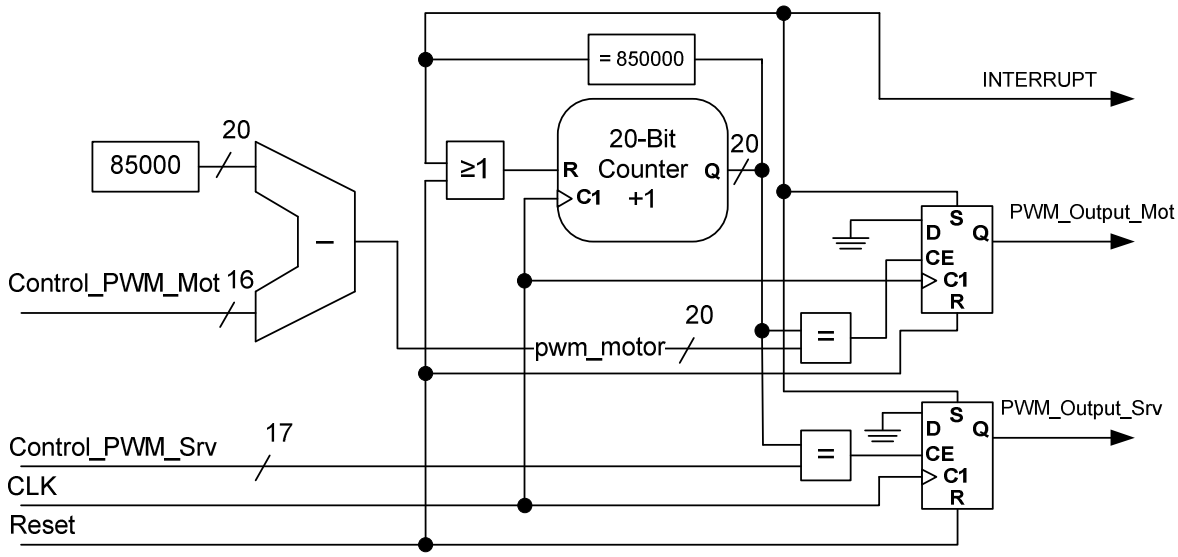

Bild 6.4.: *Modul zur Generierung der Abtastinterrupts und der PWM-Pulse für den Fahrten- und den Lenkwinkelsteller*

Der Komparator =850000 gibt die Aufträge, den Zähler zurückzusetzen und einen taktlangen $1/f_{clk}$ = 20ns Interrupt-Impuls zu erzeugen, sobald der maximale Zählerwert erreicht wurde. Gleichzeitig erzeugen die PWM-Ausgänge der beiden Flip-Flops steigende Flanken. Die Manipulation der PWM-Tastverhältnisse ist durch die Veränderung der Stellsignalen „Control_PWM_Srv“ und „pwm_motor“ realisiert. Sobald diese mit dem Zählerwerte übereinstimmen, werden die Enable-Eingänge der Flip-Flops freigeschaltet und die PWM-Ausgänge generieren fallende Flanken.

Vor Inbetriebnahme des Fahrzeug-Fahrtenstellers ist eine Initialisierungsphase nötig, um das gewünschte Verhalten dieser Stell-Komponente durch PWM-Pulsveränderungen zu realisieren. Die Initialisierungsphase wurde im PWM-Generator implementiert, da dieses VHDL-Modul die Steuersignale für den Fahrtensteller erzeugt. Ablauf der Initialisierung:

- Sende 200 Perioden T_{pwm} die Pulsfolgen mit dem Tastverhältnis a = 7.5%:
  Fahrzeugstillstand erkannt.
- Sende 150 T_{pwm} die Pulsfolgen mit a = 7.3%:
  Vorwärtsfahrt durch die Verringerung des Tastverhältnisses erkannt.
- Sende 150 T_{pwm} die Pulsfolgen mit a = 7.7%:
  Rückwärtsfahrt durch die Vergrößerung des Tastverhältnisses erkannt.
- Zuletzt mit a = 7.5% initialisieren:
  Das Fahrzeug bleibt stehen und wartet auf weitere PWM-Steuersignale.

Die Initialisierung des Fahrzeug-Fahrtenstellers dauert 8.5s, anschließend wird die Geschwindigkeitsregelung durch die Freigabe der Abtastinterrupts aktiviert

Der Fahrzeug-Lenkwinkelsteller erfordert keine Initialisierungsphase und wird durch das direkte Setzen des Software-Registers über die PWM-Pulse mit dem Tastverhältnis a = 7.5% in die Mittelstellung gebracht.

6.2. RTL-Modell zur Auswertung der Hall-Sensor-Pulsfolgen

Der Hall-Sensoren-Auswerter (*vgl.* ***Bild 6.5***) bestimmt die Periodendauer *V_COUNT* eines Hall-Sensor-Pulses, den gefahrenen Weg *DIS_IST* und die Fahrtrichtung *DIR_IST*. Des Weiteren gibt er über ein Enable-Signal *DIV_EN* dem Dividierer-Modul eine Freigabe, die Fahrzeug-Ist-Geschwindigkeit zu berechnen. Die Eingänge des Auswerters: CLK, Reset, die Hall-Sensor-Pulse B und C und das Enable-Signal *DIS_C_EN* zur Freischaltung des Wegzählers.

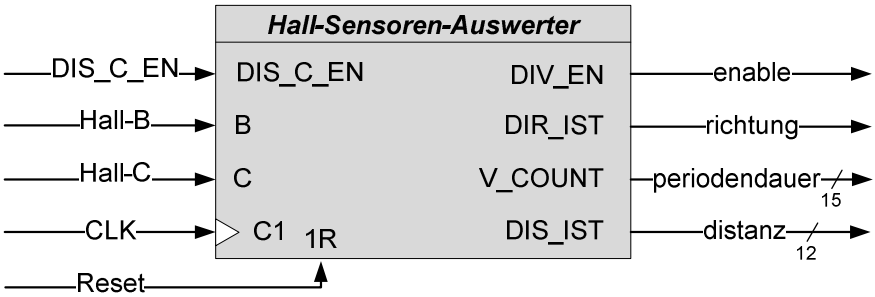

Bild 6.5.: *Modul zur Bestimmung der Fahrtrichtung des Fahrweges und der Pulsperiodendauer aus den Pulsfolgen Hall-B und Hall-C*

Ein Zustandsautomat ermittelt die Periodendauer des Hall-C-Pulses für die Ermitlung der Ist-Geschwindigkeit. Da die Richtung der Pulsperiodenverschiebung die Fahrtrichtung des Fahrzeugs kennzeichnet (*vgl.* ***Bild 4.5***), reichen für die Fahrtrichtungsbestimmung zwei Hall-Pulse aus, so wurde der Hall-A nicht ausgewertet. Die Änderung der Puls-Flanken hat direkten Einfluss auf die Ausgänge des Hall-Sensoren-Auswerters, daher wurde der Automat als eine Mealy-FSM entworfen.

Der Zustandsautomat (*vgl.* ***Bild 6.6***) decodiert die synchronisierte Sensor-Pulse Hall-B und Hall-C (*vgl.* ***Bild 6.6***) und steuert die durch seine Ausgänge Zähler zur Fahrweg- und zur Periodendauermessung. Die Ausgänge des Automaten sind: Die Richtung, das Reset-Signal für den Periodendauerzähler und das Enable-Signal für den Wegzähler.

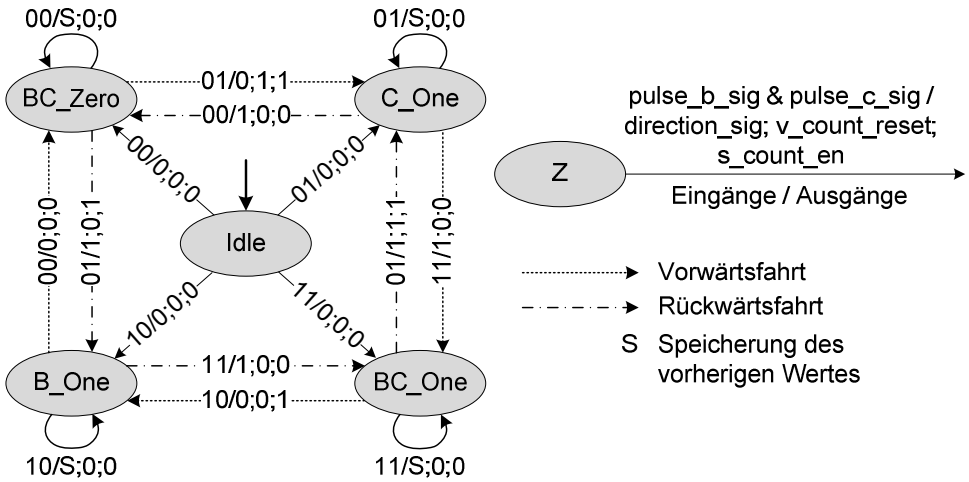

Bild 6.6.: *Mealy-FSM zur Dekodierung der Hall-Sensor-Pulse, zur Fahrtrichtungserkennung und Zähleransteuerung*

Beim den Transitionen von Idle-Zustand andern sich keine Ausgangswerte, da die Informationen über die Drehrichtung und Puls-Pegelwechsel fehlen. Der Periodendauerzähler wird beim Übergang in den C_One-Zustand zurückgesetzt, dort beginnt die neue Hall-C-Periode. Zur höheren Aktualisierungsratte der Distanzgröße wird der Fahrwegzähler bei den Übergängen in den C_One- und den B_One-Zustände eingeschaltet (*1/2 der Periode*). Wenn Störungen auf den Eingangsleitungen auftreten, kehrt der Automat in Idle-Zustand zurück und startet erneut.

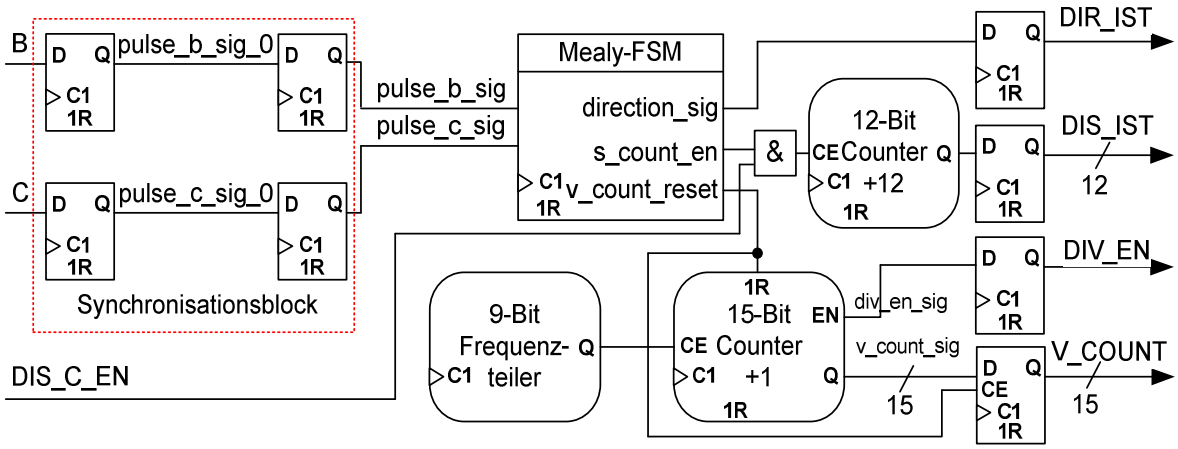

Bild 6.7.: *Logik- und Speicherelemente des Hall-Sensoren-Auswerters mit Ein- und Ausgangssynchronisationsregistern*

Ein 12-Bit Zähler realisiert die ***Distanzmessung***. Mit dem DIS_C_EN-Signal wird er Freigeschaltet, sein Zählerstand ändert sich zweimal innerhalb einer Hall-Pulsperiode (*vgl.* ***Bild 6.6***). Da eine Periode (*Motorwellenumdrehung*) S_{wu} = 24mm des gefahrenen Weges kenzeichnen, vergrößert dieser Zähler seinen Zählerstand um 12. Somit haben die danach folgenden Register den Fahrweg in der Millimetergröße präsent. Der vorhandene 12-Bit Zähler misst bis zu 4m lange Strecken.

Die ***Periodendauermessung*** wurde mit einem 9-Bit-Frequenzteiler und 15-Bit-Zähler, und nicht mit einem 24-Bit Zähler (*wie im* ***Bild 4.9***) implementiert. Somit führt das nachfolgende Dividierer-Modul (*vgl.* ***Bild 6.1***) eine 15-Bit statt 24-Bit Division und ist dadurch entlastet. Der Frequenzteiler erzeugt mit f_{clk} = 100KHz Frequenz einen taktlangen (10µs) Impuls, der an den Enable-Eingang des Zählers angeschlossen ist. In das folgende Register schreibt der 15-Bit Zähler mit jedem v_conter_reset den Zählerwert *v_count_sig* (*vgl.* ***Tabelle 6.2***). Gleichzeitig wird dem Dividierer über DIV_EN der Auftrag erteilt, V_COUNT zu lesen und die Division auszuführen, folglich setzt sich der Zähler auf den Startwert zurück.

Da die Periodendauer der Hall-Sensoren im Intervall [5ms; 180ms] liegen (*vgl.* ***Tabelle 4.3***), ergibt sich für den Zähler bei f_{clk} = 100KHz der Werteintervall [500; 18000]. Um den Überlauf und die Division durch null zu vermeiden, wurden in den 15-Bit Zähler Komparatoren integriert, die für weitere Berechnungen ungeeignete Werte aussortieren:

- Unterhalb der 500-Grenze liegenden Werte, entstehen durch Störungen an der Hall-Sensor-Leitung, da der Fahrzeugmotor nur die Geschwindigkeiten bis 4.8m/s erzeugt. Der V_COUNT wird nicht neuüberschrieben und behält den gespeicherten Wert.
- Die über 18000 liegenden Zählerwerte entstehen, wenn der Motor nicht genug Drehkraft erzeugt, um die Reibwiderstande der Fahrzeugmechanik zu überwinden. Diese Werte führen dazu, dass die Ist-Geschwindigkeit auf null gesetzt wird.

Zustand des Fahrzeugs	***Periodendauer in ms***	***Zählerstand***
Fahrzeugstillstand ($v = 0$)	∞	>18000
Fahrt mit max. Geschwindigkeit ($v = 4.8m/s$)	5	500
Fahrt mit min. Geschwindigkeit ($v = 0.13m/s$)	180	18000

Tabelle 6.2.: *Wertebereich des Zählers für die Periodendauermessung der Hall-Sensorpulse*

6.3. RTL-Modellierung des Dividierers zur Ist-Geschwindigkeitsberechnung

Der Dividierer berechnet aus der konstanten Fahrstrecke pro Motorwellenumdrehung S_{wu} = 24mm und der Hall-Sensor-Periodendauer t_{hs} (*Dauer einer Motorwellenumdrehung*) die v_{ist} Ist-Geschwindigkeit des Fahrzeugs (*vgl.* ***Gl. (3.2)***). Dieser Wert wird von dem PI-Regler im T_a = 17ms abgetastet und daraus die Regeldifferenz gebildet. Die Dauer der Hall-Sensor-Periode erhält der Dividierer vom Hall-Sensoren-Auswerter (*vgl.* ***Bild 6.1***) als Zählerwert *V_COUNT*. Für v_{ist} gilt:

$$v = \frac{S_{wu}}{t_{hs}};\ t_{hs} = \frac{V\_COUNT}{100KHz} = V\_COUNT * 10\mu s;\ \rightarrow v_{ist} = \frac{S_{wu}/10\mu s}{V\_COUNT}; \qquad (6.1)$$

Die Größe $S_{wu}/10\mu s$ ist eine Konstante und ist *2400m/s*, somit liegt v_{ist} im Intervall von 0 bis 4.8m/s. Die Bruchkonstante skaliert man zur Vermeidung der Kommazahlen-Arithmetik mit Q-Format, verwendet statt 2400m/s die 240000cm/s und erhält den v_{ist} Wertebereich zentimetergenau und größer [0cm/s; 480cm/s].

Der Dividiererals wurde als Prozessor-Element (*vgl.* ***Kap. 3.2***) mit Steuer- und Datenpfad modelliert, sein Divisionsalgorithmus, beschrieben durch folgenden C-Code (*vgl.* ***Listing 6.1***), basiert auf Unsigned-Integer Division by Trial Subtraction[26].

```
1  unsigned udiv_simple(unsigned d, unsigned n, unsigned N){
2      unsigned q=0, r=n;
3      do{                                  /* calculate next quotient bit */
4          N--;                             /* move to next bit */
5          if ( (r >> N) >= d ){            /* if r>=d*(1 << N) */
6              r -= (d << N);               /* update remainder */
7              q += (1 << N);               /* update quotient */
8          }
9      } while (N);
10 return q;}
```

Listing 6.1.: *Division by Trial Subtraction Algorithm*

Der Algorithmus berechnet $q = \lfloor n/d \rfloor$ und $r = n \ \% \ d$, wo N die Bitvektorbreite des Quotienten q ist. Angefangen vom höchsten Bit N-1 werden die Quotienten-Bits der Reihe nach gesetzt, wenn der um N nach rechts verschobene Rest r ≥ Nenner d ist. Ein 9-Bit-Vektor umfasst den Geschwindigkeitswertebereich mit dem höchsten Bit Nummer 8.

Das ASM-Diagramm (*vgl.* ***Bild 6.8***) beschreibt das Verhaltensmodell des Dividierers. Nach dem EN oder dem RESET werden Variablen mit den Werten *d = V_COUNT, q = 0, r = 240000, N = 8* initialisiert. Im IS0-Zustand wird ein Takt gewartet, damit nach der Transition IS2 → IS0 der nachfolgende Endscheidungsblock mit dem neuaktualisierten N-Wert arbeiten. Zur Speicherung des Restes r und des Quotienten q benötigt man zwei Register: für q ein 9-Bit-Register, für r ein 15-Bit-Register. Die Transitionen IS2 → IDLE und IS2 → IS0

deuten auf das Mealy-Verhalten des realisierten Zustandsautomaten zur Steuerung des Datenpfades hin.

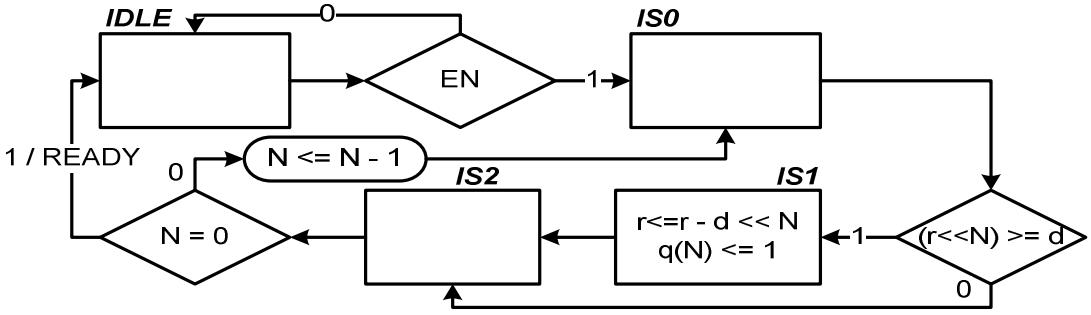

Bild 6.8.:*Verhaltensmodell des Dividierers zur Ist-Geschwindigkeitsberechnung*

Im DATA-Modul des Dividierers wurde die Arithmetik des Division by Trial Subtraction Algorithmuns realisiert und im FSM-Modul ein Zustandsautomat zur Datenpfad-Steuerung implementiert. Ihre Kommunikation erfolgt über interne Status- und Steuersignale: READY, R1_EN, R2_EN, C0, C1. Aus den Eingängen V_COUNT und DIR_IST wird die Fahrzeug-Ist-Geschwindigkeit V_IST berechnet.

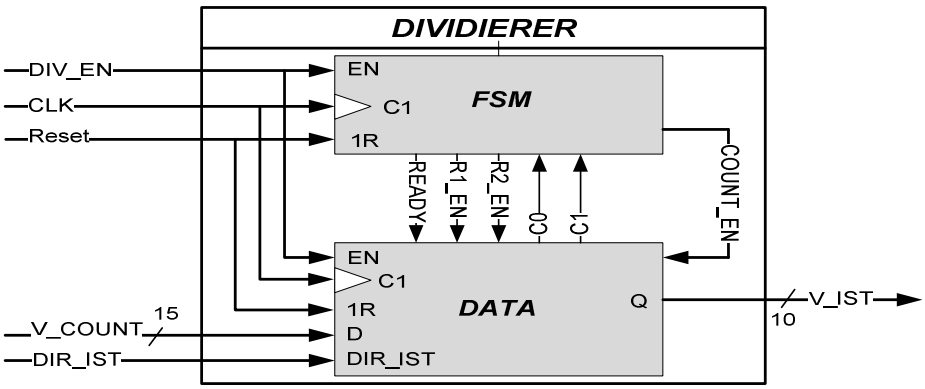

Bild 6.9.: *Dividierer mit Steuer- und Datenpfad zur Ist-Geschwindigkeitsberechnung*

Zur Verkürzung der Berechnungsdauer und zur Einsparung der Moor-Zustände wurde der ***Steuerpfad*** des Divisionsalgorithmus als eine Mealy-FSM entworfen. Der Zustandsautomat beginnt im IDLE-Zustand, nachdem EN = 1 ist, startet der Algorithmus, er arbeitet taktsynchron sodass mit jedem CLK = 1, ein Zustandswechsel stattfindet.

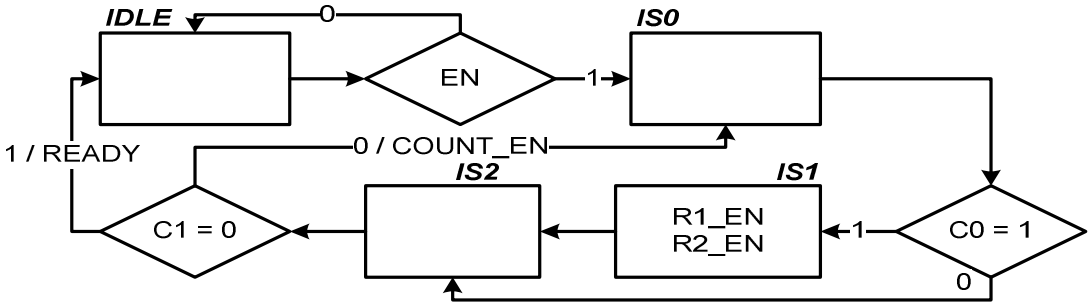

Bild 6.10.: *Mealy-Zustandsdiagramm zur Steuerung des Dividierer-Datenpfades*

Der Zustandsautomat besteht aus vier Zuständen und der Quotienten-Vektor aus 9-Bits. Beim Setzen eines Vektorbits auf eins bzw. null werden entsprechend drei bzw. zwei Transitionen ausgeführt. Die Berechnungsdauer des Quotienten q:

- q_{max} = *0x1FF* → 9 * 3 + 1(*Start*) = 28 Transitionen.
- q_{min} = *0x0* → 9 * 2 + 1(*Start*) = 19 Transitionen.

Das Zeitintervall zur q-Berechnung ist [380ns; 560ns], da der Automat taktsynchron mit der Systemfrequenz f_{clk} = 50MHz arbeitet.

Der ***Datenpfad*** des Dividierers nach **Bild 6.11**:

1. Die in den Registern stehenden Signale werden beim Start mit Werten V_{ist}: R1_I = 0 und S_{wu}*/10µs*: R2_I = 240000, ebenso der Zähler mit N = 8 initialisiert. Somit sind alle Variablen des Algorithmus bestimmt.
2. Es wird überprüft, ob (V_COUNT << N) >= R2_I ist, und C0 gesetzt. Sobald C0 = 1 ist, wird ein neuer R2_I-Wert berechnet (*R2_I – (V_COUNT << N)*) und im R1_I-Vektor an Stelle N eine ‘1‘ geschrieben. Bei C0 = 0 bleiben die alten Werte in den beiden Registern unverändert.
3. Der Zähler wird dekrementiert, erhält den Wert N-1 und wiederholt die in 2 beschriebene Berechnung.
4. Beim Wert N = 0 wird der Algorithmus beendet, der Automat erzeugt ein READY-Signal. Die Geschwindigkeit steht im R1_I-Register.
5. Der V_IST-Wert wurde vorzeichenbehaftet implementiert, um die Fahrzeugrichtung sofort zu erkennen. Durch Konvertierung wird aus den Unsigned-Signalen DIR_IST und R1_I ein Signed-Signal V_IST. Der Konverter wird durch READY-Puls angestoßen. Für die Rückwärtsfahrt werden negative (*bis -480cm/s*) und für Vorwärtsfahrt positive (*bis 480cm/s*) Geschwindigkeitswerte berechnet.

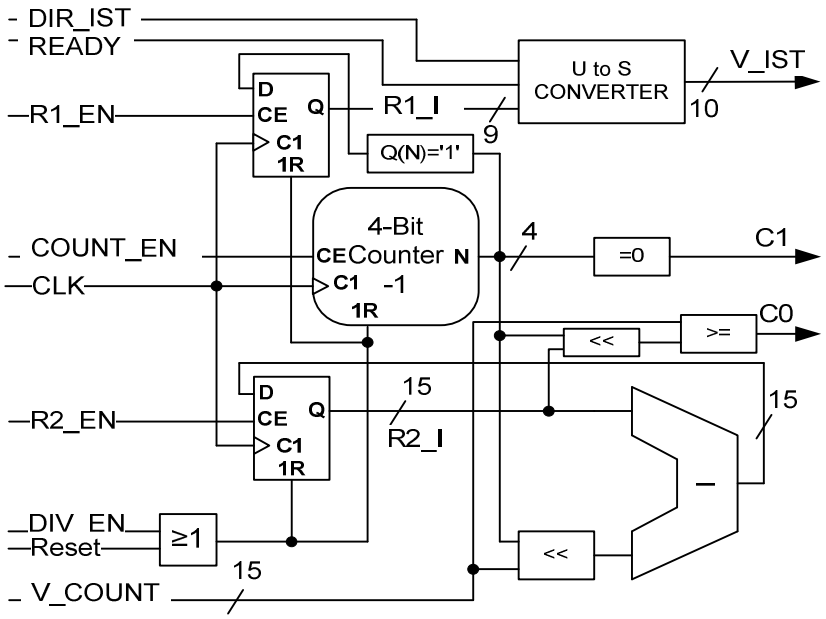

Bild 6.11.: *Datenpfad des Dividierers berechnet die Ist-Geschwindigkeit.*

6.4. RTL-Modellierung des PI-Geschwindigkeitsreglers

In diesem Kapitel werden die Entwurfsschritte zur Realisierung des RTL-Models des PI-Regler-Algorithmus präsentiert. Folgende Aufstellung weist sie auf:

- Differenzengleichung zur Stellgrößenerzeugung bilden.
- Algorithmus zur Geschwindigkeitsregelung definieren.
- Parameter des PI-Reglers skalieren und in Q-Format darstellen.
- Prozessor-Element zum PI-Regler-Algorithmus modellieren.

Die Aufgabe des PI-Reglers ist die Erzeugung der Stellwerte zur Geschwindigkeitsregelung, aus dem Grund wurde die Differenzengleichung der Stellgrößen bestimmt:

1. In **Gl.5.8** konstruierte man die z-Übertragungsfunktion des PI-Reglers mit Konstanten b1, a1, b2, a2. Durch Multiplikation mit dem Delay-Operator z^{-1} erhält man den Bezug zur Vergangenheit der Werte.

$$G_r(z) = \frac{y(z)}{e(z)} = \frac{(b1z - a1)z^{-1}}{(b2z - a2)z^{-1}} = \frac{b1 - a1z^{-1}}{b2 - a2z^{-1}} \tag{6.1}$$

2. Zum Extrahieren der Ausgangsgröße y(z) wurde **Gl. 6.1** zum selben Nenner geführt:

$$\frac{y(z)(b2 - a2z^{-1}) - e(z)(b1 - a1z^{-1})}{e(z)(b2 - a2z^{-1})} = 0 \tag{6.2}$$

3. Der Nenner fällt aus, da nur der Zähler die Ausgangsgröße repräsentiert:

$$y(z)b2 - y(z)a2z^{-1} = e(z)b1 - e(z)a1z^{-1} \tag{6.3}$$

4. Es existiert ein Zusammenhang zwischen der z-Übertragungsfunktion und der entsprechenden Differenzengleichung: *x(z)* → *x(k)* und *x(z)z*$^{-1}$ → *x(k – 1)*[21]. Mit der Beachtung dieser Verschiebungsregel wurde die Differenzengleichung umgeformt und nach Stellgröße y(k) gelöst, durch die Klammerung bleibt die Addition erspart:

$$b2y(k) - a2y(k-1) = b1e(k) - a1e(k-1) \tag{6.4}$$

$$y(k) = \frac{b1}{b2}e(k) - \left(\frac{a1}{b2}e(k-1) - \frac{a2}{b2}y(k-1)\right) \tag{6.5}$$

5. Durch das Ersetzen der Konstanten $a1 = 2T_nK_p - T_aK_p$, $b1 = 2T_nK_p + T_aK_p$, $a2 = b2 = 2T_n$ erhält die **Gl. (6.5)** folgende Form:

$$y(k) = \underbrace{\frac{2T_nK_p + T_aK_p}{2T_n}}_{p1}e(k) - \left(\underbrace{\frac{2T_nK_p - T_aK_p}{2T_n}}_{p2}e(k-1) - y(k-1)\right) \tag{6.6}$$

Die Variablen e(k – 1) und y(k – 1) sind die gespeicherte Regelabweichung und der Stellwert aus der vorherigen Berechnung. Da beim Systemstart noch keine Vergangenheitswerte bekannt sind, initialisiert man die Vergangenheitsvariablen mit den entsprechenden Startwerten der Gegenwartsvariablen. Der Algorithmus ist somit durch nachfolgenden Code, der in Intervallen von T_a = 17ms (*Abtastzeit*) periodisch aufgerufen wird, beschrieben:

```
e = v_soll - v_ist;                    /*Regelabweichung bilden*/
y = p1 * e - (p2 * e_alt - y_alt);     /*Stellwert erzeugen*/
e_alt = e;                             /*Alte Regelabweichung speichern*/
y_alt = y;                             /*Alten Stellwert speichern*/
```

Listing 6.2.: *PI-Regler-Algorithmus als C-Code, der mit jedem Abtastinterrupt aufgerufen wird.*

Der im **Listing 6.2** aufgezeigte C-Code kann z.B. innerhalb einer Interrupt-Service-Routine stehen, die auf in T_a = 17ms Abstand kommenden Interrupts (*von einem Timer*) anspringt. Die Variablen e_alt und y_alt müssen static sein, da sie, nachdem ISR beendet ist, den zugewiesenen Wert behalten sollen.

Skalierung der Regel- und Stellgrößen des PI-Reglers

Zweck: Einsparung der Skalierungs-Komponenten an den Schnittstellen des Reglers.

Nicht skaliert: Der Regler-Algorithmus erzeugt aus dem Regelgrößenintervallen v: [0m/s; 4.8m/s] den Stellgrößen-Intervall pwm: [0%; 100%].

Skaliert: Der Regler-Algorithmus erzeugt aus dem Regelgrößenintervallen v: [-480cm/s; 480cm/s] den Stellgrößen-Intervall pwm: [0; 42500].

Begründung: Der Dividierer (*vgl.* ***Kap. 6.3***) und der PWM-Generator (*vgl.* ***Kap. 6.1***) arbeiten mit skalierten Intervallen.

Realisierung: Die nach den Kuhn-Einstellregeln ermittelten Regler-Parameter p1 und p2 (*vgl.* ***Tabelle 5.3***) werden angepasst (*vgl.* ***Tabelle 6.3***).

Skalierungsgrößen	***Nicht Skaliert***	***Skaliert***
K_s	4.8m/s	0.0226cm/s
K_p nach Kuhn	0.10417s/m	22.124s/cm
T_n nach Kuhn	0.136s	0.136s
$2T_n$	0.272s	0.272s
$2T_nK_p + T_aK_p$	$0.03010513s^2/m$	$6.39384s^2/cm$
$2T_nK_p – T_aK_p$	$0.02656335s^2/m$	$5.64162s^2/cm$
$p1 = (2T_nK_p + T_aK_p)/ 2T_n$	0.11068065s/m	23.506765s/cm
$p2 = (2T_nK_p - T_aK_p)/ 2T_n$	0.09865938s/m	20.74125s/cm

Tabelle 6.3.: *Skalierung der Einstell-Parameter des PI-Reglers. Geschwindigkeit in cm/s statt m/s, Stellgröße im Intervall [0; 42500] statt [0%; 100%]*

Die Parameter p1 und p2 sind Kommazahlen, die im Q-Format dargestellt werden. Der Signed-Integer-Anteil der Parameter passt in 6-Bits, zur Darstellung der Nachkommazahlen wählte man 8-Bits und erreichte dadurch eine 1 / (2^8 – 1) = 0,003922 Präzision. Die Parameter p1 und p2 wurden in $Q_{6.8}$-Format, wie folgt, konvertiert und als ein 14-Bit-Vektor dargestellt:

p1: 23 → "010111"; 0.506765 * 2^8 – 1 = 129 → "10000001"; p1 = "01011110000001"

p2: 20 → "010100"; 0.74125 * 2^8 – 1 = 189 → "10111101"; p2 = "01010010111101"

Der Algorithmus des PI-Reglers (*vgl.* ***Listing 6.2***) wurde als Prozessor-Element (*vgl.* ***Kap 3.2***) mit der Trennung in Steuer- und Datenpfad implementiert. Angestoßen durch Abtastinterrupt, wird das PI-DATA-Modul über interne Steuersignale (*R0_EN*, *R1_EN*, *R2_EN*, *S0*, *S1*, *PI_READY*) von dem PI_FSM-Modul gesteuert. Der PWM-Stellwert für den PWM-Generator wird alle T_a = 17ms neu gebildet.

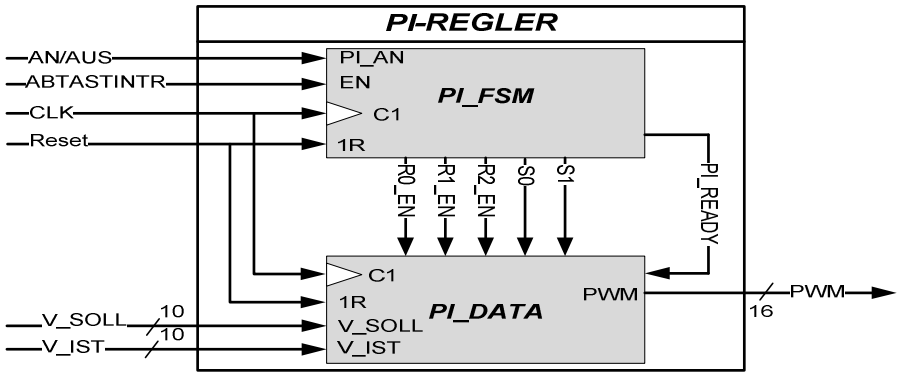

Bild 6.12.: *PI-Regler-Prozessor-Element zur Stellgrößenerzeugung*

Erstellung des ASM-Diagramms: Der PI-Regler-Algorithmus (*vgl.* ***Listing 6.2***) wurde in VHDL nur mit einem Multiplizierer und Subtrahierer realisiert, da in der Gleichung der Stellgröße nur Multiplikations- und Subtraktions-Operationen vorkommen. Zur maximalen Auslastung der Arithmetik-Komponenten wurden die Operanden der Differenzgleichung wurden in Teilaufgaben zerlegt, und werden zu verschiedenen Takten berechnet (*vgl.* ***Bild 6.13***). Mit dem Signal *PI_AN = 1* wird der PI-Regler freigeschaltet, mit jedem Abtastinterrupt bildet der Algorithmus eine neue Stellgröße. In Zuständen IS0 und IS1 werden, zur Berechnung der Operanden für die nächste Stufe, gleichzeitig Subtraktion und Multiplikation ausgeführt. Im Zustand IS2 wird die Stellgröße für den PWM-Generator gebildet, der Algorithmus ist beendet und wartet auf den nächsten Abtastinterrupt.

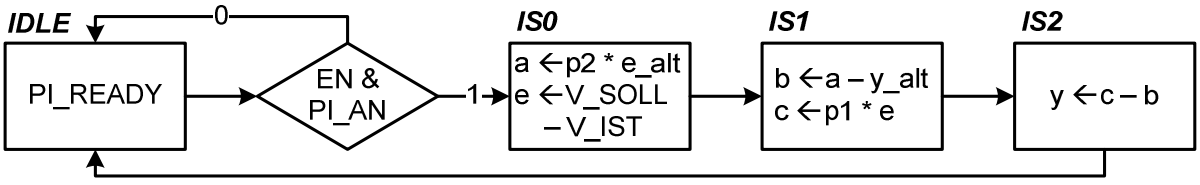

Bild 6.13.: *Algorithmus-Ablauf des PI-Reglers, der einen Subtrahierer*

und einen Multiplizierer zur Stellgrößenberechnung verwendet.

Ressource-/Register-Sharing: Durch die Klammern in der **Gl. 6.5** wurden ein Addierer, sowie durch die Zerlegung der Differenzengleichung weitere Subtrahierer und Multiplizierrer eingespart. Ein Tabellen-Verfahren (*vgl.* ***Tabelle 6.4***) diente zur Durchführung des Register-Sharings, mit X bezeichnet man den Zustand, ab dem die Variable keinen Speicher mehr beansprucht. Die maximale Anzahl der X-e in einer Spalte zeigt die nötige Registermenge, um alle Variablen speichern zu können. Durch das Verfahren reduzierte man die Registeranzahl von sieben auf drei.

Variable/Zustand	IDLE	IS0	IS1	IS2	Register-Nr.
e_alt		X			0
e	X		X	X	0
a			X		1
b				X	2
c				X	1
y_alt		X	X		2
y	X				2

Tabelle 6.4.: *Register-Sharing des PI-Regler-Algorithmus*

Steuerpfad: Ein taktsynchroner Moor-Zustandsautomat (*vgl.* ***Bild 6.14***) mit vier Zuständen wurde konstruiert, um die arithmetischen Komponenten und die Register des PI-Reglers zu steuern. Die Ausgänge gelten in jeweiligem Zustand für eine Taktperiode. Die R0_EN, R1_EN und R2_EN sind die Enable-Signale für die Datenpfad-Register. Mit dem Ausgang S0 bzw. S1 werden die Multiplexer zur Auswahl der Operanden für Subtrahierer bzw. Multiplizierer geschaltet. Da der Zustandsautomat aus vier Zuständen besteht und mit f_{clk} = 50MHz getaktet wird, ist jede Stellgrößenberechnung nach 4 * 20ns = 80ns fertig. Zum Abschneiden der Nachkommastellen wird noch ein Takt = 20ns und die Register-Load-Zeit verbraucht.

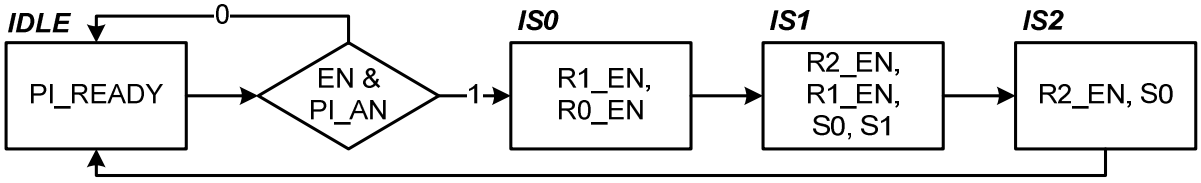

Bild 6.14.: *Moor-Zustandsdiagramm zur Steuerung des PI-Regler-Datenpfades*

Datenpfad (*vgl.* ***Bild 6.15*** *und* ***Bild 7.2***) Ablauf der Stellgrößenberechnung:

- Als Default-Wert wurden alle FSM-Ausgänge mit null initialisiert. Im IS0-Zustand werden die Register1 und Register0 eingeschaltet und sind bereit neue Werte aufzunehmen. Da das Multiplexer-Steuersignal S0 = 0 ist, wird die Regeldifferenz berechnet und im Register0 gespeichert, die Bitbreite des Registerausgangs R0_I ergibt sich aus der maximal erreichbaren Regeldifferenz (*z.B. –480 – 480 = –960*):

V_SOLL (*10Bit*) – V_IST (*10Bit*): $[S|G_8 \ldots G_0] - [S|G_8 \ldots G_0] = [S|G_9 \ldots G_0] \rightarrow$ 11Bit

Die Übernahme der neu berechneten Regelabweichung e findet mit der nächsten positiven Taktflanke statt. Somit ist gewährleistet, dass der Multiplizierer den alten R0_I-Wert mit dem p2-Parameter multipliziert. Das berechnete Multiplikations-Produkt a wird im Register1 platziert, die R1_I-Vektorbreite ergibt sich aus dem Signed-Bit, dem maximalen Multiplikations-Produkt (*15-Bit*) und aus 8 Fractionalbits zur Darstellung von Nachkommazahlen:

$$R0\_I_{max} * max(p1;p2) = 960 * \ 23.506765 = 22566 \rightarrow 15Bit$$

$$[S|G_9 \ldots G_0] * [S|G_4 \ldots G_0|F_7 \ldots F_0] = [S|G_{14} \ldots G_0|F_7 \ldots F_0] \rightarrow 24Bit$$

- Im IS1-Zustand werden die Multiplexer-Steuersignale S0 und S1 auf eins gesetzt. Die neu berechneten Produkte c und b werden mit der nächsten Taktflanke in die Register 1 und 2 übernommen. Der Multiplizierer bildet ein Produkt c aus der im IS0-Zustand berechneten Regelabweichung e (*R0_I-Signal*) und dem p1-Parameter. Hingegen berechnet der Subtrahierer R1_I (*Produkt a*) minus R2_I (*y_alt*) und speichert dieses Ergebniss b im Register2. Aus dem Signed-Bit, dem maximalen Subtraktion-Produkt (*16-Bit*) und 8 Fractionalbits erhält man die Bitbreite des R2_I-Registerausgangs:

$$R1\_I_{max} + Stellwert_{max} = 22566 + 42500 = 65066 \rightarrow 16Bit$$

$$[S|G_{14} \ldots G_0|F_7 \ldots F_0] + [S|G_{15} \ldots G_0|F_7 \ldots F_0] = [S|G_{15} \ldots G_0|F_7 \ldots F_0] \rightarrow 25Bit$$

- Im IS2-Zustand verläuft die gleiche Aktion (*R2_I <= R1_I – R2_I*) wie im IS1-Zustand. Der Stellwert wird berechnet und ein PI_READY im IDLE-Zustand generiert, ausgelöst durch dieses Signal, werden die Fractionalbits des R2_I-Vektors abgeschnitten. Da die Stellwerte immer positiv sind, wird auch des Signed-Bit entfernt. Als ein Unsigned-Vektor werden die 16 Guardbits an den PWM-Generator weitergegeben, das PWM-Signal liegt im Intervall [0; 42500] und entspricht [0%; 5%] des PWM-Tastverhältnis.

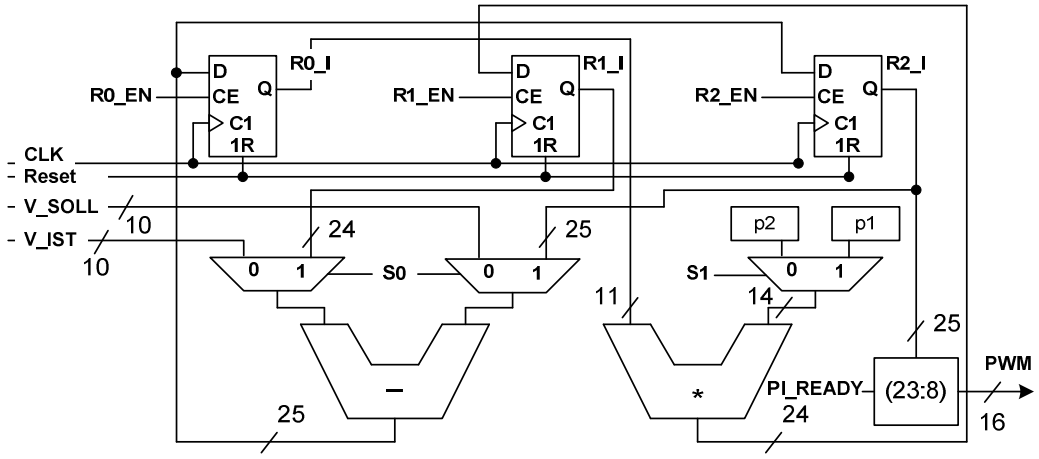

Bild 6.15.: *Datenpfad desPI-Regler-Algorithmus*

7. Messtechnische Analyse der Simulations- und Testergebnisse

In diesem Kapitel wird durch Simulation und Test im Betrieb-Modus die Funktionalität des modellierten Geschwindigkeitsreglers gezeigt. Die Vorgehensweise:

- Der Testablauf wird beschrieben, die erwarteten Testergebnisse werden berechnet.
- Das beschrieben Testszenario wird simuliert und die Simulationsergebnisse werden mit den zuvor errechneten verglichen.
- Die Inbetriebnahme des Geschwindigkeitsreglers findet mit dem Download des Bitstreams in die Spartan-3E FPGA basierte SoC-Plattform statt. Der gleiche Test wird im Betrieb-Modus wiederholt.
- Die Sprungantwort des geschlossenen Regelkreises wird im Betriebs-Modus aufgenommen und analysiert.
- Der Ressourcenverbrauch des SoC-Systems wird präsentiert.

7.1. Testszenario und Regelkreissimulation

Der Testablauf zum Nachweis der PI-Regler-Funktionalität besteht in der Konstanthaltung der Regelabweichung *e* und Beobachtung der Stellgröße *y*. Da die Regelabweichung sich nicht ändert, akkumuliert sich der Stellwert und wird ständig größer, bis er seinen maximalen Wert 42500 erreicht. Der Verlauf der Stellwertakkumulation wird zuerst theoretisch errechnet und dann experimentell aufgenommen, der Vergleich beider Ergebnisse gibt Information über die Funktionalität des Geschwindigkeitsreglers.

Im Testszenario wird die Soll-Geschwindigkeit auf das Maximum *v_soll = 480cm/s* fixiert. Da die Ausgänge der Hall-Sensoren keine Änderung aufweisen, wird dies als Fahrzeugstillstand *v_ist = 0* interpretiert. Der PI-Regler-Algorithmus (*vgl.* ***Listing 6.2***) erkennt die Regelabweichung *e = v_soll – v_ist = 480cm/s* und berechnet alle T_a = 17ms eine neue Stellgröße *y* (*vgl.* ***Tabelle 7.1***):

0	*v_soll=480cm/s; v_ist=0cm/s; e=480cm/s; e_alt=0cm/s; y_alt=21250*	21250.00
t / ms	***p1 * e – (p2 * e_alt – y_alt)***	***y***
17	23.506765s/cm * 480cm/s – (20.74125s/cm * 0cm/s – 21250.00)	32533.25
34	23.506765s/cm * 480cm/s – (20.74125s/cm * 480cm/s – 32533.25)	33860.70
51	23.506765s/cm * 480cm/s – (20.74125s/cm * 480cm/s – 33860.70)	35188.14
68	23.506765s/cm * 480cm/s – (20.74125s/cm * 480cm/s – 35188.14)	36515.60
85	23.506765s/cm * 480cm/s – (20.74125s/cm * 480cm/s – 36515.60)	37843.03
102	23.506765s/cm * 480cm/s – (20.74125s/cm * 480cm/s – 37843.03)	39170.47
119	23.506765s/cm * 480cm/s – (20.74125s/cm * 480cm/s – 39170.47)	40497.93
136	23.506765s/cm * 480cm/s – (20.74125s/cm * 480cm/s – 40497.93)	41825.38
153	23.506765s/cm * 480cm/s – (20.74125s/cm * 480cm/s – 41825.38)	42500.00
170	23.506765s/cm * 480cm/s – (20.74125s/cm * 480cm/s – 42500.00)	42500.00

Tabelle 7.1.: *Berechnete Stellwerte des Geschwindigkeitsreglers beim Testszenario*

Der Algorithmus verstärkt und integriert die Stellgröße uns erreicht nach 153ms den maximalen Stellwert. Der Regler-Algorithmus akkumuliert weiter und berechnet dadurch

einen neuen Stellwert. Jedoch ist dieser größer als der maximal erlaubte, deswegen werden nachfolgende Werte durch einen Komparator auf 42500 heruntergezogen, der Regler erreicht somit einen stationären Zustand.

Der Aufbau des simulierten Moduls wurde im **Bild 6.1** vorgestellt, zur besseren Übersicht speicherte man im .do-File nur für Geschwindigkeitsregelung relevante Signale. Der PWM-Generator implementiert die 8.5s lange Initialisierungsphase (*vgl.* ***Kap.6.1***), die man zur Simulationszwecken abgeschaltet hat. Ihre Aufgabe bestand darin die Initialisierung der PWM-Stellgröße *y* (*control_pwm_mot*) mit dem Startwert 21250 vor zu nehmen, durch die Init-Ausschaltung wurde dieses Signal mit dem Defaultwert 0 initialisiert. Auf den PI-Regler-Algorithmus hatte es keine Auswirkung, da er mit eigenen Registern arbeitet, die mit anwendungsspezifischen Startwerten initialisiert wurden.

Die Abtastinterrupts werden nach Ablauf der PWM-Periode generiert und erfolgen in T_a = 17ms Abständen (*vgl.* ***Bild 7.1***). Jedes Interrupt startet den PI-Regler-Algorithmus, der eine neue Stellgröße zur Erzeugung der fallenden Flanke des PWM-Signals berechnet. Das PWM-Tastverhältnis wandert von 7.5% nach 5%, dies löst beim Fahrzeug die Vorwärtsfahrt aus. Nach 153ms erreichte, wie erwartet, der PI-Regler seinen stationären Zustand. Die durch Simulation erhaltenen Stellgrößen unterscheiden sich von den berechneten (*vgl.* ***Tab. 7.1***) in den Dezimalziffern, diese Abweichung entsteht durch Nutzung von Q-Format-Arithmetik.

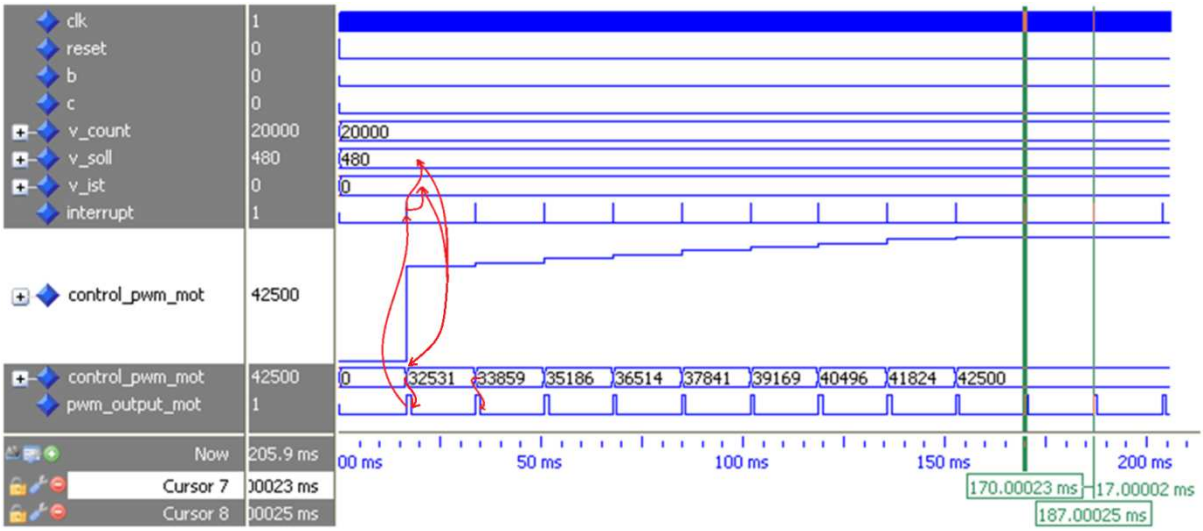

Bild 7.1.: *Stellwertveränderung bei konstanter Regelabweichung*

Die Dauer des PI-Regler-Algorithmus (*vgl.* ***Bild 7.2***) beträgt, wie im **Kap. 6.4** ausgerechnet, 100ns plus der Load-Zeit des Registers (*in der Simulation 5ns*). Die Pfeile in dem Bild haben folgende Bedeutung:

1. Algorithmus-Start, ausgelöst durch Abtastinterrupt.
2. Zustandswechsel des PI_FSM und Einschaltung der Register R0 und R1.
3. Übernahme der berechneten Regelabweichung e = v_soll – v_ist = 480 – 0 = 480.
4. Übernahme des berechneten Produkts a = p2 * e_alt = 20.74125 * 0 = 0.

5. Übernahme des berechneten Produkts c = p1 * e = 23.506765 * 480 = 11283.2472
   0x17.81 * 0x1E0 = 0x2C11.E0 = 11281.878528 ← Abweichung durch Q-Format.
6. Übernahme des berechneten Produkts b= a–y_alt= 0–21250.00=–21250.00= 0x1ACEF.00.
7. Übernahme des berechneten Stellwerts y= c – b= 0x2C11.E0 + 0x1ACEF.00 = 0x7F13.E0.
8. Nachkommastellen des Stellwerts abschneiden pwm = 0x7F13 = 32531.

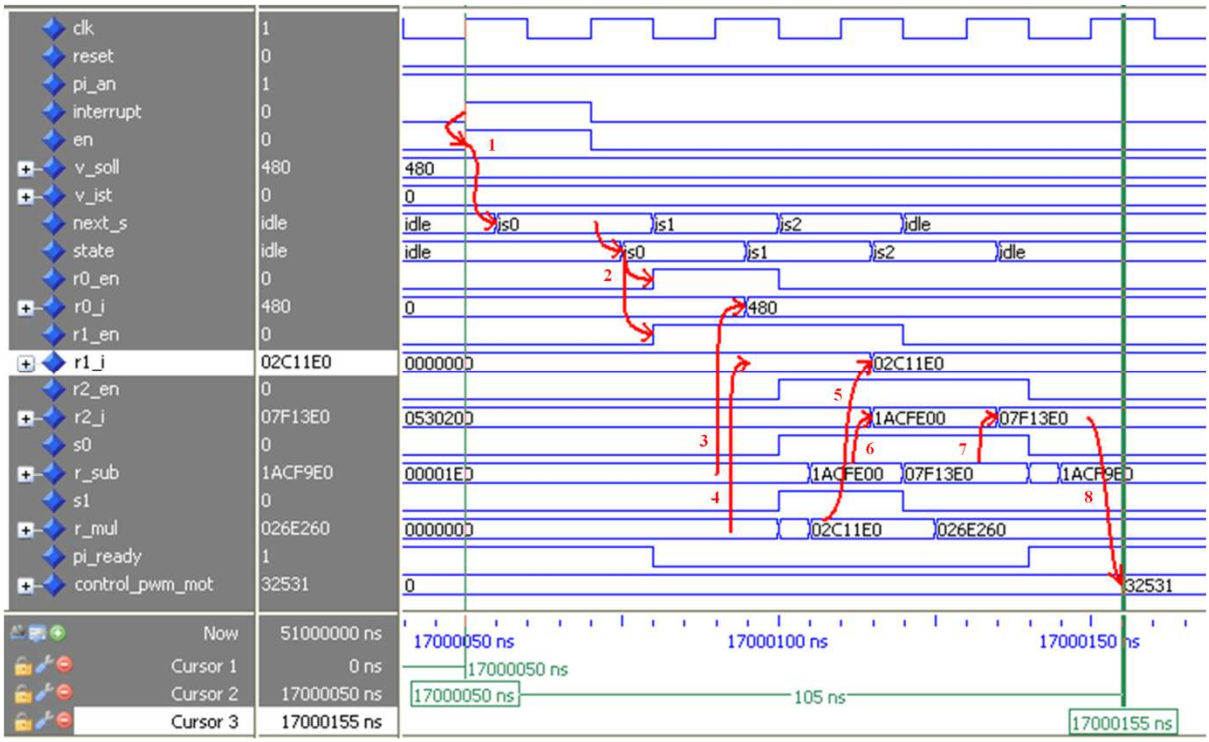

Bild 7.2.: *Algorithmus-Simulation des Geschwindigkeitsreglers*

Nach der Simulation wurde das Nexys2-Board durch EDK-Tool über Download-Bitstream mit dem SoC-Entwurf programmiert. An das Oszilloskop schloss man den PWM-Ausgang an, die Stellgrößen des PI-Reglers wurden über den UART mit der Software abgeholt (*vgl.* ***Kap.2.1.2***) und im Hyper Terminal ausgegeben. Wenn der Anschluss der Hall-Sensoren an das Board nicht erfolgt, interpretiert das entwickelte Programm dies als Fahrzeugstillstand.

Nach dem Programmstart beobachtet man am Oszilloskop die PWM-Tastverhältnisveränderung von 7.5% nach 5%. Die Hyper Terminal Software gab die erwarteten Stellwerte wie folgt aus (*vgl.* ***Bild 7.1***):

0 -- 32531 -- 33859 -- 35186 -- 36514 -- 37841 -- 39169 -- 40496 -- 41824 -- 42500

7.2. Untersuchung des Fahrzeugfahrverhaltens bei einer Testfahrt

Für die Testfahrt wurde die Lenkaufgabe von der Fernsteuerung übernommen. Die Testfahrt überprüfte das Verhalten des Fahrzeugs mit dem PI-Regler beim Nachlassen der Akku-Spannung. Dazu wurde es mit einem schwach aufgeladenen Akku ausgestattet. Beim Nachlassen der Akkuspannung geht auch die PWM-Spannung herunter, als Folge wird das Fahrzeug langsamer. Der Geschwindigkeitsregler soll diesen Spannungsverlust ausregeln, indem er die Stellwerte durch Integration vergrößert, dadurch wird das PWM-Signal breiter und das Fahrzeug akzeleriert. In diesem Fall müssen die Stellgrößen die definierte Grenze (*42500*) überschreiten, deshalb wurde der Komparator zur Begrenzung der Stellgrößen ausgeschaltet.

Zur Steuerung und zum Verfolgen des Testfahrtablaufs wurde eine ISR in C-Code geschrieben (*vgl.* ***Listing 7.1***), sie wird alle T_a = 17ms aufgerufen, gibt die Fahrzeuggeschwindigkeit vor und speichert die Stellgrößen des PI-Reglers und die Ist-Geschwindigkeitswerte des Fahrzeugs. Während des Tests beschleunigte das Fahrzeug auf Vmax = 480cm/s, die Sprungantworten des Regelkreises und die des Reglers wurden im SRAM des Nexys2-Boards gespeichert und anschließend mit Hyper Terminal am Bildschirm dargestellt.

```
1  void isr ( void * baseaddr_p)
2  {
3      int IpStatus;
4      if (counter < 150){
5      XIo_Out32((XPAR_V_REGLER_HW_0_BASEADDR)+(V_REGLER_HW_SLV_REG0_OFFSET), v_soll);//480cm/s
6      //--Array zum Speichern der Ist-Geschwindigkeitswerte
7      vIstOut[counter] = XIo_In32((XPAR_V_REGLER_HW_0_BASEADDR)+(V_REGLER_HW_SLV_REG0_OFFSET));
8      //--Array zum Speichern der Stellwerte
9      pwmOut[counter] = XIo_In32((XPAR_V_REGLER_HW_0_BASEADDR)+(V_REGLER_HW_SLV_REG1_OFFSET));
10     counter++;
11     } else{
12     v_soll = 0;
13     XIo_Out32((XPAR_V_REGLER_HW_0_BASEADDR)+(V_REGLER_HW_SLV_REG0_OFFSET), v_soll);//0m/s
14     }
15     IpStatus = XIo_In32(XPAR_V_REGLER_HW_0_BASEADDR+V_REGLER_HW_INTR_IPISR_OFFSET);
16     XIo_Out32((XPAR_V_REGLER_HW_0_BASEADDR + V_REGLER_HW_INTR_IPISR_OFFSET), IpStatus);
17 }
```

Listing 7.1.: *ISR der Testfahrt-Software reagiert auf die 17ms Interrupts des PWM-Generators, steuert das Fahrzeug und speichert die Testwerte.*

Mit Excel wurden die aufgenommenen Testwerte in Grafiken umgewandelt (*vgl.* ***Bild 7.3***). Man erkennt, dass trotz schwachen Akkus die vorgegebene Geschwindigkeit 480cm/s nach 1,7s erreicht war. Die aufgenommene Sprungantwort des Regelkreises ähnelt derjenigen der Regelstrecke (*vgl.* ***Bild 4.10***), sie zeigt, dass dem Regler gestellte Aufgabe erledigt wird – sogar bei schwachem Akku. Es werden keine Überschwinger beobachtet, kein Durchdrehen der Räder bei der Beschleunigung festgestellt. Der durchgeführte Test beweist, dass der PI-Regler in der Lage ist die Geschwindigkeitsunterschiede auszugleichen.

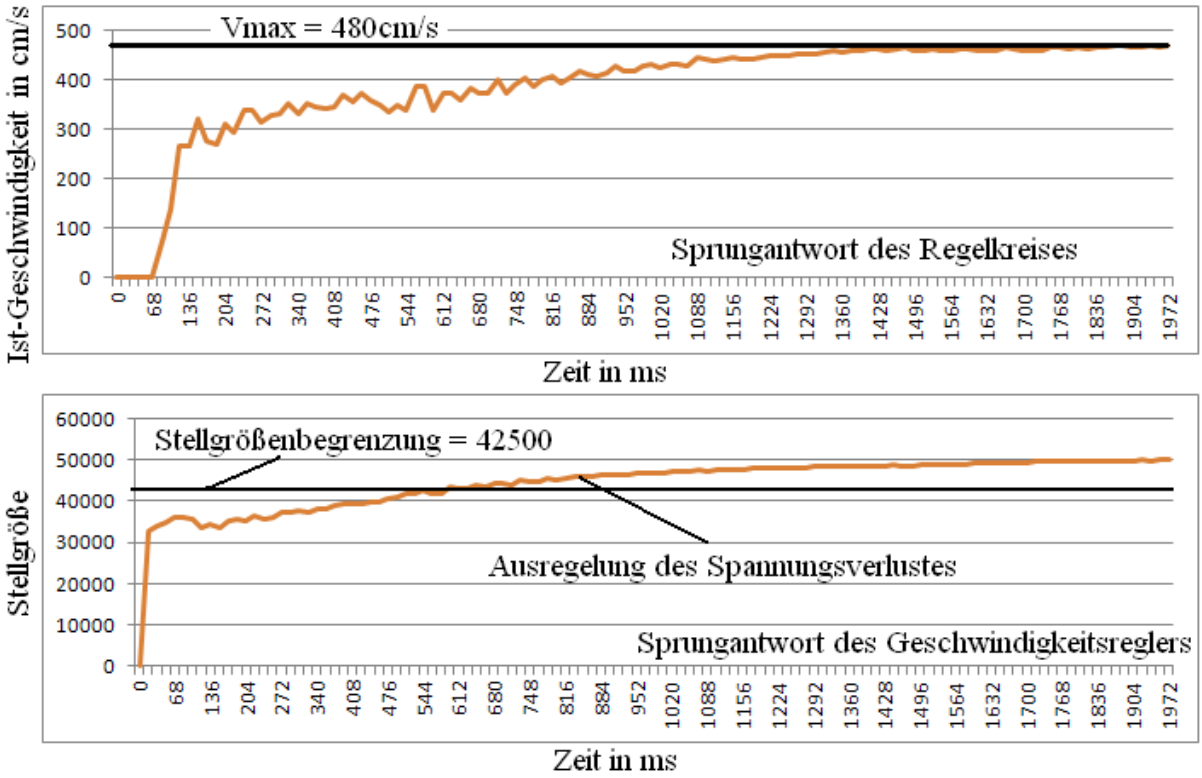

Bild 7.3.: *Sprungantworte des Geschwindigkeitsregelkreises und des -reglers bei der Testfahrt mit schwach aufgeladenem Akku*

Der Ausgleich der Geschwindigkeitsunterscheide findet nicht nur beim Nachlassen des Akkus, sondern auch bei bergab-bergauf Fahrten oder bei Vergrößerung des Fahrzeuggewichts statt. Für die Spurführung ist die Einhaltung der vorgegebenen Geschwindigkeit von Bedeutung, denn sie ist zur Auswahl und Parametrisierung des Spurführungsalgorithmus nötig (*vgl.* [16]).

Der Regler akzeleriert das Fahrzeug sowohl vorwärts als auch rückwärts, die Fahrzeugbeschleunigung kann positiv oder negativ sein, der Geschwindigkeitsausgleich findet in beide Richtungen statt. Den Übergang von der Vorwärts- in die Rückwärtsfahrt führt der Geschwindigkeitsregler problemlos aus.

7.3. Ressourcenverbrauch des SoC-Systems

Die Berechnung des FPGA-Ressourcenverbrauchs wurde mit EDK durchgeführt. Zur seiner Bestimmung für den entwickelten Geschwindigkeitsregler ermittelte man zuerst den Bedarf des MicroBlaze-Systems ohne ihn. Danach koppelte man das entwickelte Regelungssystem mit dem Micro-Blaze über ein PLB und hielt die Steigerung des Ressourcenverbrauchs (*vgl.* ***Tabelle 7.2***) fest.

Name der Ressource	**Verwendete FPGA-Ressourcen des Systems**				**Ressourcenbedarf des Geschwindigkeitsreglers**	
	ohne den Geschwindigkeitsregler		**mit dem Geschwindigkeitsregler**			
	Anzahl	Verbrauch	Anzahl	Verbrauch	Anzahl	Verbrauch
Flip Flops	1905	10.98%	2257	13.01%	352	2.03%
LUTs	2471	14.25%	3115	17.96%	644	3.71%
Slices	2214	25.53%	2624	30.26%	410	4.73%
IOBs	80	32.00%	82	32.80%	2	0.80%
MULTs	3	10.72%	5	17.86%	2	7.14%

Tabelle 7.2.: *Ermittlung des FPGA-Ressourcenverbrauchs der Geschwindigkeitsreglung, der längste Signallaufpfad ist 9.518ns (5.710ns logic, 3.808ns route)*

Die FPGA-Ressourcen reichen aus, um die entworfene Komponente einzeln zu testen. Dennoch ist die Geschwindigkeitsregelung nur ein Teil des Sensor-Gesteuerten-Fahrzeug-System, allein die Fahrspurführung verbrauchte 98% der Slice-Reserve (*vgl.* [16]). Die FPGA-Ressourcen des NEXYS2-Boards genügen nicht, um das gesamte SCV-System zu realisieren. Für die Lösung dieser Aufgabe werden zwei Alternativen vorgeschlagen:

- Nutzung eines anderen Spartan-3A DSP 1800-Boards[27], das im Vergleich zur Nexys2 über mehr FPGA-Ressourcen verfügt. Darunter versteht man die Verlagerung des gesamten SCV-Systems auf eine andere FPGA-basierte Plattform.
- Erweiterung des Nexys2-Boards durch die Ankopplung des TE-320-Boards[28], das weitere FPGA-Ressourcen zur Verfügung stellt. Dabei werden die einzelnen Fahrzeugassistent-Systeme auf zwei Boards verteilt.

Zurzeit untersuchen Bachelor- und Master-Studenten des SCV-Projekts beide Lösungswege. Ihre Untersuchungsergebnisse werden die Vorteile und Nachteile der einzelnen Alternativen auflisten, sowie durch die Bewertung die optimale Lösung wählen.

8. Zusammenfassung

In dieser Arbeit wurde eine Geschwindigkeitsregelung als Prozessor-Element für das SoC-Fahrzeug des Projekts „Fahrerassistenz- und Autonome Systeme“ (FAUST) implementiert. Das entwickelte Fahrerassistenz-System ist mit dem Processor Local Bus des eingebetteten MicroBlaze-Systems gekoppelt und in dem Spartan-3E FPGA integriert. Das Ziel des eingebetteten Entwurfs war es die System-on-Chip-Technologie zu erproben und zu durchdringen.

Die PWM-Signale des Fahrzeugempfängers für den Fahrten- und den Lenkwinkelsteller wurden analysiert, um die betriebstypischen Tastverhältnisse zu bestimmen. Zur Steuerung des Fahrzeugs von dem SoC wurde ein PWM-Generator als Peripheriekomponente entworfen und in das MicroBlaze µC-System integriert. Zur Sprungantwortaufnahmen wurden die Parameter zur Variation der PWM-Tastverhältnisse über die Software-Register eingestellt.

Die Montage der zusätzlichen Sensor-Mechanik an den Fahrzeugrädern wurde eingespart, da das auf dem, Übersetzungsverhalten der Motorwelle auf die Fahrzeugräder basierte Verfahren zur Ist-Geschwindigkeitsberechnung ausgewählt wurde. Zur Bestimmung der Richtung und der Periodendauer der Motordrehung wurden die Pulse der Motor-Hall-Sensoren ausgewertet. Die Ist-Geschwindigkeit wird aus der Dauer und der gefahrenen Strecke pro Periode eines Hall-Sensors ermittelt, da die Analyse der Hall-Sensor-Timing-Werte zeigte, dass die Auswertung eines Hall-Sensors mit 0.3% und die der drei Sensoren mit 7% der mittleren Abweichung belastet sind. Weiterhin liefern die Hall-Sensoren bei niedrigen Geschwindigkeiten zu wenig Pulse, dies führte zum Ausschluss des Pulszählverfahrens.

Mit einem Hall-Sensor-Auswerter wurde die Sprungantwort der Fahrzeugantriebstrangs-Regelstrecke aufgenommen und als PT_1-Glied mit einer Totzeit identifiziert. Ein PI-Regler wurde zur Geschwindigkeitsregelung gewählt und mit dem Wendetangenten- und dem T-Summen-Verfahren die PI-Regler-Parameter berechnet. Zur Entwurf des RTL-Modells wurde der PI-Regler durch die z-Transformation diskretisiert. Der Geschwindigkeitsregler wurde nach T-Summen-Verfahren und Ü.Kuhn Empfehlungen parametrisiert, da die MATLAB-Simulation den geforderten Beschleunigungsverlauf des Fahrzeugs zeigte und keine Überschwinger aufwies.

Die zur Ist-Geschwindigkeitsberechnung verwendete 1/T Division wurde nach dem Division by Trial Subtraction Algorithms als integriertes Prozessor-Element mit einem Multizyklusdatenpfad implementiert, da im FPGA keine dedizierten Dividierer zur Verfügung stehen. Die Vorteile gegenüber dem Dividierer aus dem Core-Generator ist der geringe Ressourcenverbrauch durch die Anpassung an die anwendungsspezifische Aufgabe. Für die Berechnung des 9-Bit Quotienten v_ist benötigt dieses Modul maximal 28 Systemtakte.

Das RTL-Modell des PI-Regler-Algorithmus wurde als Prozessor-Element mit Trennung in Daten- und Steuerpfad entworfen, da im Vergleich zur Pipeline-Lösung weniger FPGA-

Ressourcen benötigt werden. Der Algorithmus startet mit einem Abtastinterrupt und aktualisiert den Stellwert in 5 Systemtakten. Zur Einsparung der Kommazahl-Arithmetik und zusätzlichen Skalierungs-Modulen an den Schnittstellen des PI-Regler-Moduls wurde vor seiner Kopplung mit dem Dividierer und dem PWM-Generator eine Intervall-Skalierung der Ein- und Ausgangsgrößen durchgeführt. Die Erzeugung der Abtastinterrupts führt ein Timer des PWM-Generators aus, da die PWM-und Interrupt-Perioden übereinstimmen. Alle Komponenten der Geschwindigkeitsregelung wurden mit VHDL implementiert und als ein IP-Core an den PLB des MocroBlaze-Systems gekoppelt. Die durch Timing-Analyse ermittelte längste Signal-Laufzeit dieser Peripheriekomponente ist 9.518ns, daraus resultiert die maximale Taktfrequenz f_{clk} = 105MHz.

Die Funktionen der Geschwindigkeitsregelung wurden durch messtechnische Analyse überprüft. Eine Software in C-Code steuerte den Testfahrtablauf, dabei wurden die Ist-Geschwindigkeiten und Regler-Stellgrößen im SRAM des Boards gespeichert und anschließend mit dem Hyper-Terminal eines PCs dargestellt. Diese Messwerte wurden in die Grafiken umgewandelt und zeigen das erwartete Regelungsverhalten des SoC-Fahrzeugs.

Literaturverzeichnis

[1] about.com. [Online] [Zugriff: 04. 10. 2011.] http://inventors.about.com/library/inventors/blcruisecontrol.htm.

[2] Wikipedia. [Online] [Zugriff: 04. 10. 2011.] http://de.wikipedia.org/wiki/Tempomat.

[3] *„Carolo-Cup Regelwerk 2012 vom 17. Juni 2011"*. Braunschweig : Technische Universität Braunschweig, 2011.

[4] Marwedel, Peter und Wehmeyer, Lars. *"Eingebettete Systeme"*. Berlin : Springer, 2007.

[5] ZVEI - Zentralverband Elektrotechnik- und Elektronikindustrie e. V. Kompetenzzentrum Embedded Software & Systems. *"Nationale Roadmap Embedded Systems"*. Frankfurt : s.n., 2009.

[6] Xilinx Inc. *"Spartan-3 Generation FPGA User Guide"*. 2011.

[7] Xilinx Inc. *"Embedded System Tools Reference Manual"*. 2010.

[8] Xilinx Inc. *"MicroBlaze Processor Reference Guide"*. 2010.

[9] Digilent Inc. *"Diigiillentt Nexys2 Board Refference Manuall"*. 2011.

[10] Modellbau Härtle. [Online] [Zugriff: 01. 06. 2011.] http://www.haertle.de/rc+modellbau/rc+autos/tamiya+58302+ferrari+enzo+tt+01+bausatz+1+10.html.

[11] Hayabusa. *"Radio Control Instruction Manual"*. 2004.

[12] LRP electronic GmbH. *"USER MANUAL A.I. Brushless Reverse Digital"*. Remshalden : s.n., 2007.

[13] LRP electronic GmbH.*"USER GUIDE CRAWLER 21.5T"*. Remshalden : s.n., 2008.

[14] Tamiya Inc. *"Bauanleitung des TT-01 Chassis"*. 2005.

[15] Alpers, Thorsten. *"Modellierung eines Einparkassistenten für ein autonomes Fahrzeug implementiert auf einer SoC-Plattform"*. Hamburg : HAW, 2010.

[16] Schneider, Christian. *"Ein System-on-Chip-basiertes Fahrspurführungssystem"*. Hamburg : HAW, 2011.

[17] Brown, Stephen und Vranesic, Zvonko. *"Fundamentals of Digital Logic with VHDL Design"*. New York : McGraw-Hill, 2008.

[18] Gajski, Daniel. *"Fundamentals of Digital Logic with VHDL Design"*. New Jersey : Prentice Hall, 1996.

[19] Reichardt, Jürgen and Schwarz, Bernd. *"VHDL-Synthese: Entwurf digitaler Schaltungen und Systeme"*. München : Oldenbourg Wissenschaftsverlag GmbH, 2009.

[20] Unbehauen, Heinz. *"Regelungstechnik II: Zustandsregelungen, digitale und nichtlineare Regelsysteme"*. Wiesbaden : Vieweg+Teubner Verlag, 2007.

[21] Zacher, Serge and Reuter, Manfred. *"Regelungstechnik für Ingenieure"*. Wiesbaden : Vieweg+Teubner Verlag, 2011.

[22] microchip. [Online] [Zugriff: 07. 06. 2011.] http://ww1.microchip.com/downloads/en/AppNotes/00885a.pdf.

[23] Jaschek, Hilmar und Voos, Holger. *"Grundkurs der Regelungstechnik: Einführung in die praktischen und theoretischen Methoden"*. München : Oldenbourg Wissenschaftsverlag GmbH, 2010.

[24] Philips. *"DATA SHEET 74LVC245"*. 1998.

[25] Regelungstechnik - RN-Wissen. [Online] [Zugriff: 27. 08. 2011.] http://www.rn-wissen.de/index.php/Regelungstechnik.

[26] Andrew N. Sloss, Dominic Symes, Chris Wright. *"ARM system developer's guide: designing and optimizing system software"*. San Francisco : Morgan Kaufmann Publishers, 2004.

[27] Xilinx Inc. *"Spartan-3A DSP Starter Platform User Guide"*. 2009.

[28] Trenz Electronic. *"TE0320 Series User Manual"*. 2011.

[29] Bordasch, Heiko. *"VHDL-Modellierung einer Geschwindigkeitsregelung für ein autonomes Fahrzeug implementiert auf einer SoC - Plattform"*. Hamburg : HAW, 2009.

Bildverzeichnis

Tabellenverzeichnis

A. FPGA-Ressourcenbedarfsvergleich mit dem Pipeline-V-Regler

Den Pipeline-V-Regler entwickelte Heiko Bordasch in seiner Bachelorarbeit[29], im **Kap.7.6** stellte er den System-FPGA-Ressourcenbedarf vor. Durch die Implementierung der Geschwindigkeitsregelung als Prozessor-Element sollten die FPGA-Ressourcen eingespart werden. Um dies zu überprüfen, wurden die EDK-Desing-Summary-Reporte (*vgl.* ***Tabelle A.1***) verglichen. Aus der **Tabelle 7.2** erkennt man den FPGA-Ressourcenverbrauch der Geschwindigkeitsregelung mit dem System-Takt f_{clk} = 50 MHz. Die Erhöhung der System-Takt-Frequenz bis auf f_{clk} = 80MHz reduziert den Slices-Verbrauch von 30.26% auf 29.72%.

Name der Ressource	**Lösung mit Pipeline**			**Lösung mit Prozessor-Element**		
	Verwendet	Vorhanden	Verbrauch	Verwendet	Vorhanden	Verbrauch
Flip Flops	3199	17344	18.44%	2257	17344	13.01%
LUTs	3716	17344	21.43%	3114	17344	17.96%
Slices	2582	8672	29.77%	2578	8672	29.72%
IOBs	87	250	34.80%	82	250	32.80%
MULTs	11	28	39.29%	5	28	17.86%

Tabelle A.1.: *Vergleich des FPGA-Ressourcenverbrauchs von zwei Geschwindigkeitsregler-Systemen, eine als Pipeline und eine als Prozessor-Element implementiert*

Durch mehrfache Nutzung der Flip Flops und Multiplizierer wurde deren Verbrauch erheblich reduziert. Die Einsparung der Ressourcen bei der Prozessor-Element-Lösung vergrößerte die Dauer der Stellgrößenberechnung, sie beträgt 105ns. Da die Stellgröße nur alle T_{pwm} = 17ms aktualisiert wird, ist die Dauer der Stellgrößenberechnung nicht zeitkritisch.

B. Parameteranpassung für die nicht kontinuierliche Regelstrecke

Zur Ermittlung der Regelkreiseigenschaften bei Geschwindigkeitsänderung wurde bei den unterschiedlichen v_{soll}-Werten die v_{ist}-Abweichung gemessen (*vgl.* ***Bild B.1***). Die Untersuchung zeigte, dass der Regelkreis bei Einstellung der Soll-Geschwindigkeit im unteren Bereich schwingt. Je kleiner die v_{soll} eingestellt ist, desto größer weicht die v_{ist} von ihr ab.

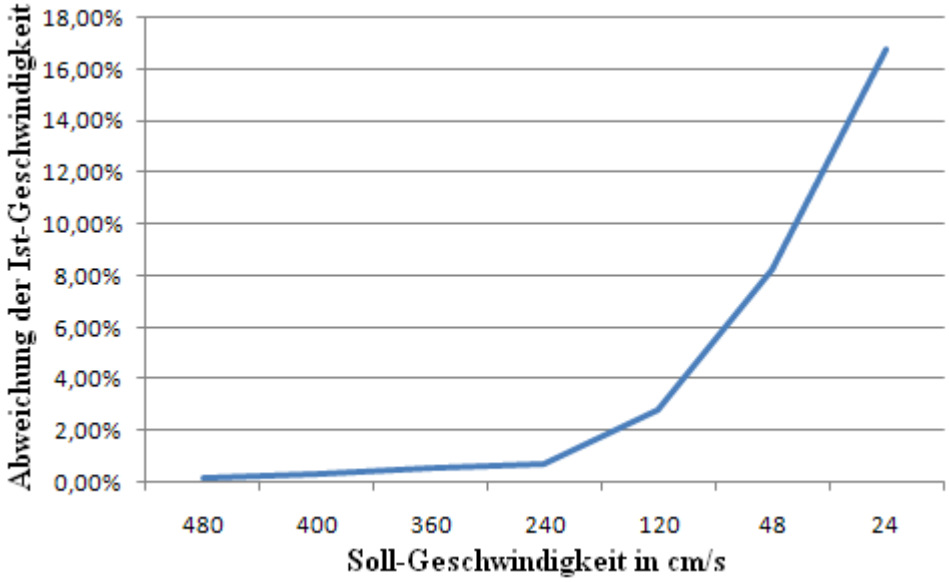

Bild B.1.: *Steigerung der Ist-Geschwindigkeitsabweichung beim Verkleinern der Soll-Geschwindigkeit*

Die Ist-Geschwindigkeitsabweichung bei höheren Geschwindigkeiten von 480cm/s bis 240cm/s liegt unter 1%. Dies deutet darauf hin, dass die PI-Regler-Parameter für höhere Geschwindigkeiten optimal berechnet wurden, für die niedrige Geschwindigkeit gelten sie jedoch nicht. Deshalb wurden die neu Sprungantworte der Regelstrecke für die Geschwindigkeiten 300cm/s, 200cm/s und 100cm/s wie in **Kap. 4.3: Ansatz2** aufgenommen. Diese Sprungantworte unterscheiden sich in der Tot-, Ausgleichszeit und im Übertragungsbeiwert, das weist auf das nicht kontinuierliche Regelstreckenverhalten hin. Um die Ist-Geschwindigkeitsabweichungen zu reduzieren, berechnete man für diese Geschwindigkeiten neue PI-Regler-Parameter nach T-Summen-Verfahren (*vgl.* ***Kap.5.3***) aus den erhaltenen Sprungantworten.

Die Berechnung der Parameter erfolgte mit Skalierten Regel- und Stellgrößen (*vgl.* ***Tabellen B.1-3***), $T_{\sum}$ und V wurden aus den neuaufgenommenen Sprungantworten ermittelt. Das PWM-Signal für den Fahrtensteller wurde durch manuelle Einstellung der PWM-Tastverhältnisse (*vgl.* ***Bild 6.4***) manipuliert.

T-Summen Verfahren: 25% PWM = 5312 $T_{\sum}$ = 0.476s V = 300 cm/s

Konstanten/Parameter	Rechenweg	Koeffizient
K_s	300cm/s / 5312	0.0565cm/s
K_p nach Kuhn	0.5 / 0.0565cm/s	8.85s/cm
T_n nach Kuhn	0.5· 0.476s	0.238s
$2T_n$	2 · 0.238s	0.476s
$2T_nK_p + T_aK_p$	0.476s·8.85s/cm + 0.017·8.85s/cm	$4.36305s^2$/cm
$2T_nK_p – T_aK_p$	0.476s·8.85s/cm – 0.017·8.85s/cm	$4.06215s^2$/cm
$p1 = (2T_nK_p + T_aK_p)/ 2T_n$	$4.36305s^2$/cm /0.476s	9.16607s/cm
$p2 = (2T_nK_p - T_aK_p)/ 2T_n$	$4.06215s^2$/cm /0.476s	8.53393s/cm
p1	**Binär**	**"00100100101010"**
p2	**Binär**	**"00100010001000"**

Tabelle B.1.: *PI-Regler-Parameter für die Beschleunigung auf v_soll = 300cm/s*

T-Summen Verfahren: 15% PWM = 3187 $T_{\sum}$ = 0.578s V = 200 cm/s

Konstanten/Parameter	Rechenweg	Koeffizient
K_s	200cm/s / 3187	0.06275cm/s
K_p nach Kuhn	0.5 / 0.06275cm/s	7.968s/cm
T_n nach Kuhn	0.5· 0.578s	0.289s
$2T_n$	2 · 0. 289s	0.578s
$2T_nK_p + T_aK_p$	0.578s ·7.968s/cm + 0.017·7.968s/cm	$4.74096s^2$/cm
$2T_nK_p – T_aK_p$	0.578s ·7.968s/cm – 0.017·7.968s/cm	$4.47005s^2$/cm
$p1 = (2T_nK_p + T_aK_p)/ 2T_n$	$4.74096s^2$/cm / 0.578s	8.20235s/cm
$p2 = (2T_nK_p - T_aK_p)/ 2T_n$	$4.47005s^2$/cm / 0.578s	7.73365 s/cm
p1	**Binär**	**"00100000110011"**
p2	**Binär**	**"00011110111011"**

Tabelle B.2.: *PI-Regler-Parameter für die Beschleunigung auf v_soll = 200cm/s*

T-Summen Verfahren: 7% PWM = 1487 $T_{\sum}$ = 0.714s V = 100 cm/s

Konstanten/Parameter	Rechenweg	Koeffizient
K_s	100cm/s / 1487	0.06725cm/s
K_p nach Kuhn	0.5 / 0.06725cm/s	7.435s/cm
T_n nach Kuhn	0.5· 0.714s	0.357s
$2T_n$	2 · 0.357s	0.714s
$2T_nK_p + T_aK_p$	0.714s ·7.435s/cm + 0.017·7.435s/cm	$5.43499s^2$/cm
$2T_nK_p – T_aK_p$	0.714s ·7.435s/cm – 0.017·7.435s/cm	$5.1822s^2$/cm
$p1 = (2T_nK_p + T_aK_p)/ 2T_n$	$6.47249s^2$/cm / 0.714s	7.61203 s/cm
$p2 = (2T_nK_p - T_aK_p)/ 2T_n$	$6.17145s^2$/cm / 0.714s	7.25798 s/cm
p1	**Binär**	**"00011110011100"**
p2	**Binär**	**"00011101000001"**

Tabelle B.3.: *PI-Regler-Parameter für die Beschleunigung auf v_soll = 100cm/s*

Das nicht kontinuierliche Verhalten der Regelstrecke hat zufolge, dass für die Einstellung der Fahrzeuggeschwindigkeit passende PI-Regler-Parameter gewählt werden. Die in **Kap.6.4** berechneten Parameter p1 und p2 sind als Konstanten im VHDL fest kodiert und lassen sich nicht zur Laufzeit ändern. Um das System flexibler zu gestalten, erstellte man einen neuen

IP-Core, in dem die Regler-Parameter p1 und p2 die Eingangssignale des PI-Reglers sind (*vgl.* ***Bild B.2***). Die benötigten PI-Regler-Parameter, werden über die Software-Register von der Steuerung-Software übergeben (*vgl.* ***Bild B.3*** *und* ***Bild 3.2***), ansonsten blieb die System-Struktur erhalten. Durch eine anwendungsspezifische Wahl der PI-Regler-Parameter wurde die Ist-Geschwindigkeitsabweichung reduziert, dadurch verfügt der neue IP-Core zur über mehr Stabilität und ist flexibler.

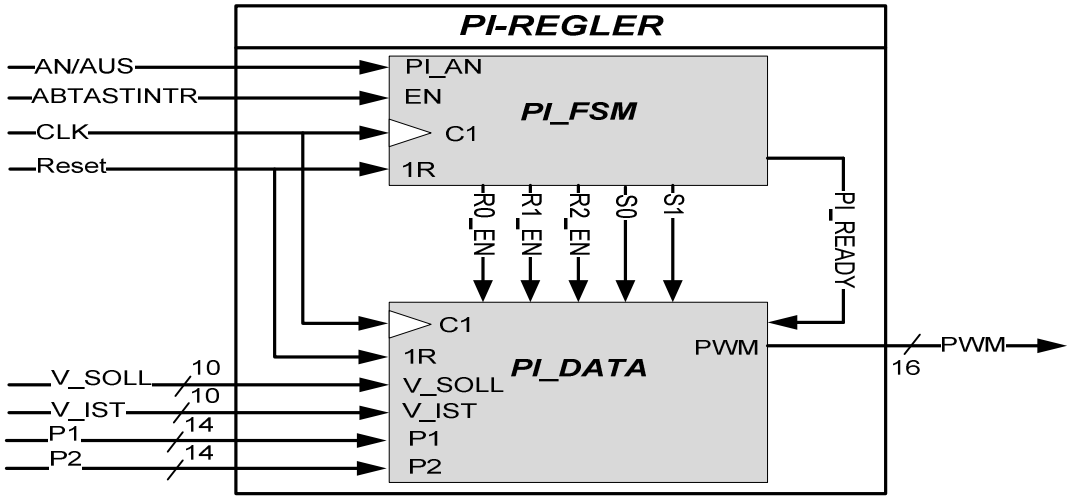

Bild B.2.: *Struktur des PI-Reglers zur Geschwindigkeitsregelung. Die Regler-Parameter sind als Eingangssignale und nicht als Konstanten in VHDL definiert.*

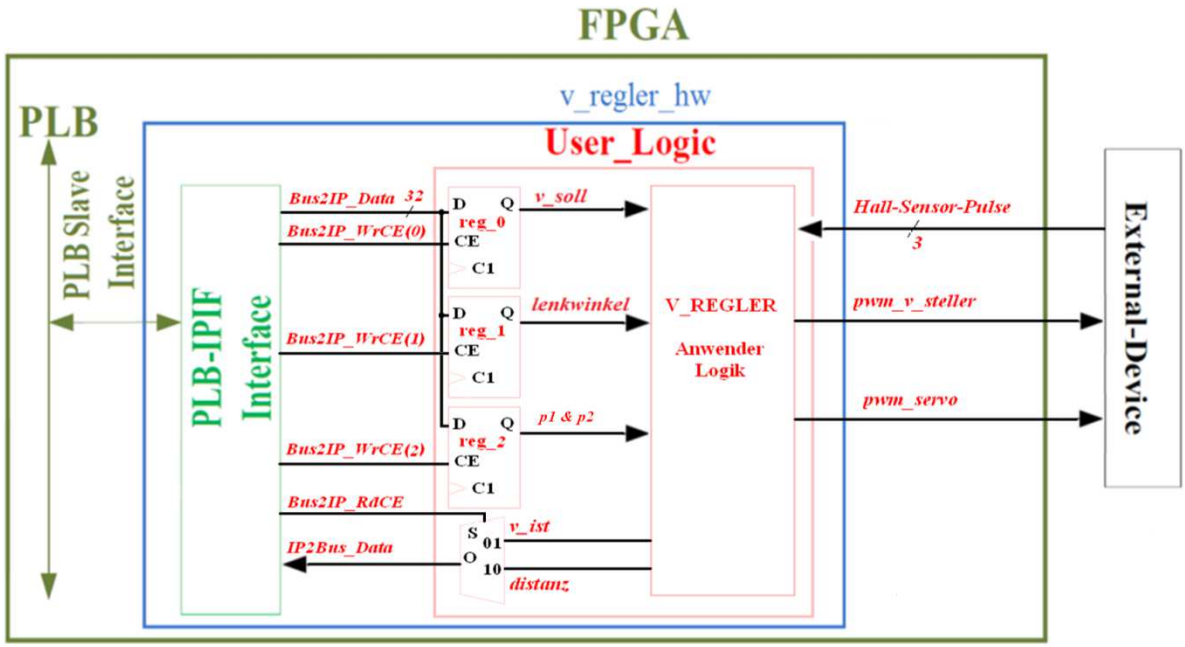

Bild B.3.: *Struktur des IP-Cores, in den die PI-Regler-Parameter p1 & p2 von der Steuerungs-Software an die Anwenderlogik über die Software-Register übergeben werden*

C. Berechnung der Fahrwege für den Einparkvorgang

Die Modellierung eines Einparkassistenten wurde in der Bachelorarbeit[15] von Herrn Alpers ausführlich beschrieben: Er stellt zwei Szenarien zur Ausführung des Einparkmanövers vor. Die Aufgabe dieses Kapitels ist es, anhand dieser Szenarien, der Fahrzeugmassen und Informationen aus dem Carolo-Cup-Regelwerk[3] den schnellsten Einparkvorgang zu berechnen. Die Fahrzeugkonstanten (*vgl.* ***Bild C.1***): FLG – Fahrzeuglänge; L – Radstand; FBR – Fahrzeugbreite; RKA – äußerer Kollisionsradius; RKI – innerer Kollisionsradius; RP – Radius zur Mitte der Hinterachse; RM – Radius zur Mitte der Vorderachse; Ω – Mittelpunkt aller Kreise.

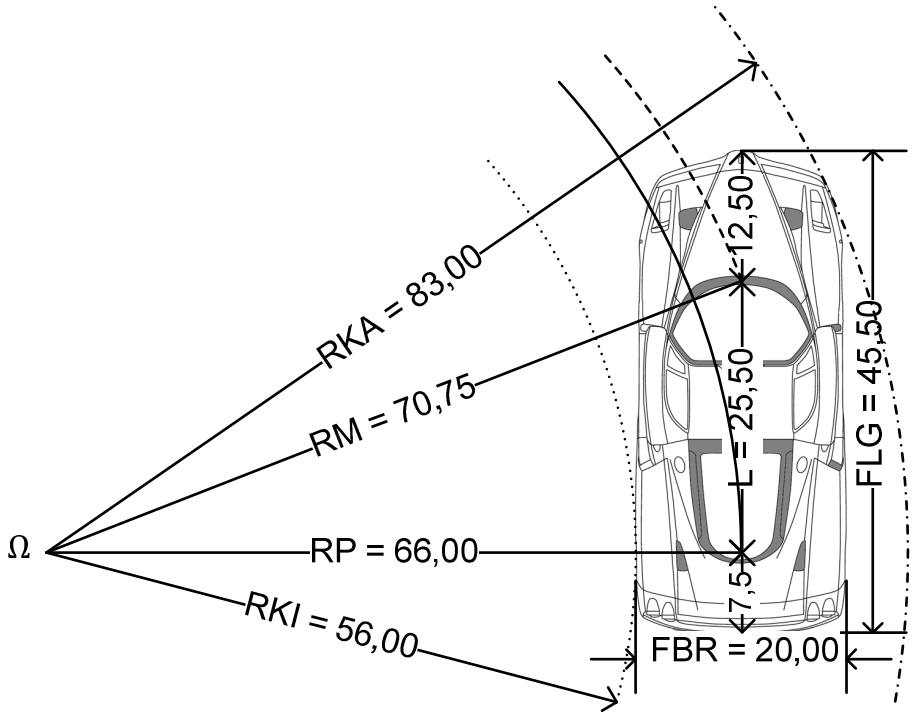

Bild C.1.: *Fahrzeugmassen sowie Radien der Kreise, die bei der Kurvenfahrt mit dem maximalen Lenkwinkelanschlag entstehen. Alle Zahlenangaben in cm*

Die Informationen zu den Fahrbahnmassen, Parklückenbreiten und Hindernispositionen wurden dem Carolo-Cup-Regelwerk[3] entnommen und in das **Bild C.2** übertragen. Für das Einparken „in einem Schritt“ wurde die größte Parklücke (*70cm*) gewählt. Beim Einparken fährt das Fahrzeug zwei gleiche Bogen ab, dadurch positioniert es sich tiefer in die Parklücke. Zur Berechnung dieser Bogenlängen wurden folgende Konstruktionsschritte entworfen und in das **Bild C.2** eingetragen:

1. Dreieck t1: Wenn sich die Mitte der Hinterachse P1 im Schnittpunkt der Katteten befindet, steht das Fahrzeug innerhalb der Parklücke, direkt an der Fahrbahnbegrenzung und am 1. Hindernis.

2. Die Katteten wurden dupliziert und im Abstand RP oben eingefügt, um Ω1 zu konstruieren. Damit das Fahrzeug mit der linken oberen Ecke des 2. Hindernisses nicht kollidiert, wurde von dieser im Abstand RKA ein Bogen gebildet, der die Katteten des t2

Dreiecks schneidet und somit die Hypotenuse des t2 Dreiecks bildet. Solange Ω1 innerhalb des t2 Dreiecks liegt, wird das Fahrzeug mit keinem der beiden Hindernisse kollidieren und innerhalb der Parklücke bleiben. Aus diesem Grund wurde entschieden, den Ω1-Punkt in die Mitte des t2 Dreiecks im 1cm Abstand von den t1 Katteten zu setzen, so steht das Fahrzeug im 1cm Abstand zum 1. Hindernis und zur Fahrbahnbegrenzung.

3. Der Mittelpunkt des zweiten Kreises Ω2 liegt im Abstand 2RP von Ω1, da die zu abfahrenden Bogenlängen gleich sind. Man konstruierte den Bogen mit dem Radius 2PR und Mittelpunkt Ω1, durch die Punkte X und Y ist der Bogenbereich bezeichnet, auf dem Ω2 liegen soll. Liegt Ω2 im Punkt X, so fährt das Fahrzeug dicht zur Fahrbahnbegrenzung, im Punkt Y hingegen streift es mit dem äußeren Kollisionsradius die Fahrbahnmitte. Der Abstand des Fahrzeugs zur Fahrbahnbegrenzung liegt somit im Bereich von 0cm bis 13cm. Für das Einparkmanöver wurde mit einem 5cm Abstand weitergearbeitet, dabei liegt der Mittelpunkt des zweiten Kreises im Punkt Ω2.

4. Der P1 Punkt entsteht durch den senkrechten RP nach unten von Ω1 und der P2 Punkt den durch senkrechten RP nach oben von Ω2. P2 bezeichnet den Startpunkt und P1 den Endpunkt des Einparkmanövers, in diesen Punkten steht das Fahrzeug parallel zur Fahrbahnbegrenzung. Die Kreise mit den Mittelpunkten Ω1 und Ω2, und den Radien der Länge RP schneiden sich im Punkt B und bilden zwei gleichlange Bogen P2B und BP1.

5. Die Formel für die Berechnung der Bogenlänge lautet:

$$Lb = \frac{2\pi R \cdot \beta}{360°} \qquad (D.1)$$

β ist der Bogenwinkel.

Zur Berechnung des Bogenwinkels benötigt man das Hilfsrechteck Ω1CΩ2D, die Gerade Ω1Ω2 bildet zwei gleiche, rechtwinklige Dreiecke Ω1CΩ2 und Ω1DΩ2, sie ist auch ihre Hypotenuse und hat die Länge 2RP = 132cm. Wenn die Längen der DΩ2 oder CΩ1 bekannt sind errechnet man die Bogenwinkelgröße nach der Formel:

$$\beta = \arccos\left(\frac{a}{c}\right) \qquad (D.2)$$

a = DΩ2 oder CΩ1 und c = Ω1Ω2.

Die Länge der Rechteckseiten DΩ2 oder CΩ1 (*vgl.* ***Bild C.2***):

$$D\Omega 2 = C\Omega 1 = \Omega 2P2 + (\Omega 1P1 - AP1) = 66cm + (66cm - 28cm) = 104cm \qquad (D.3)$$

Ω1P1 = Ω2P2 = RP = 66cm; AP1 = 10cm + 5cm + 2cm + 11cm = 28cm;
β = arccos(104cm/132) = 38.01°;
P2B = BP1 = (2π * 66cm * 38.01°) / 360° = 43.78cm

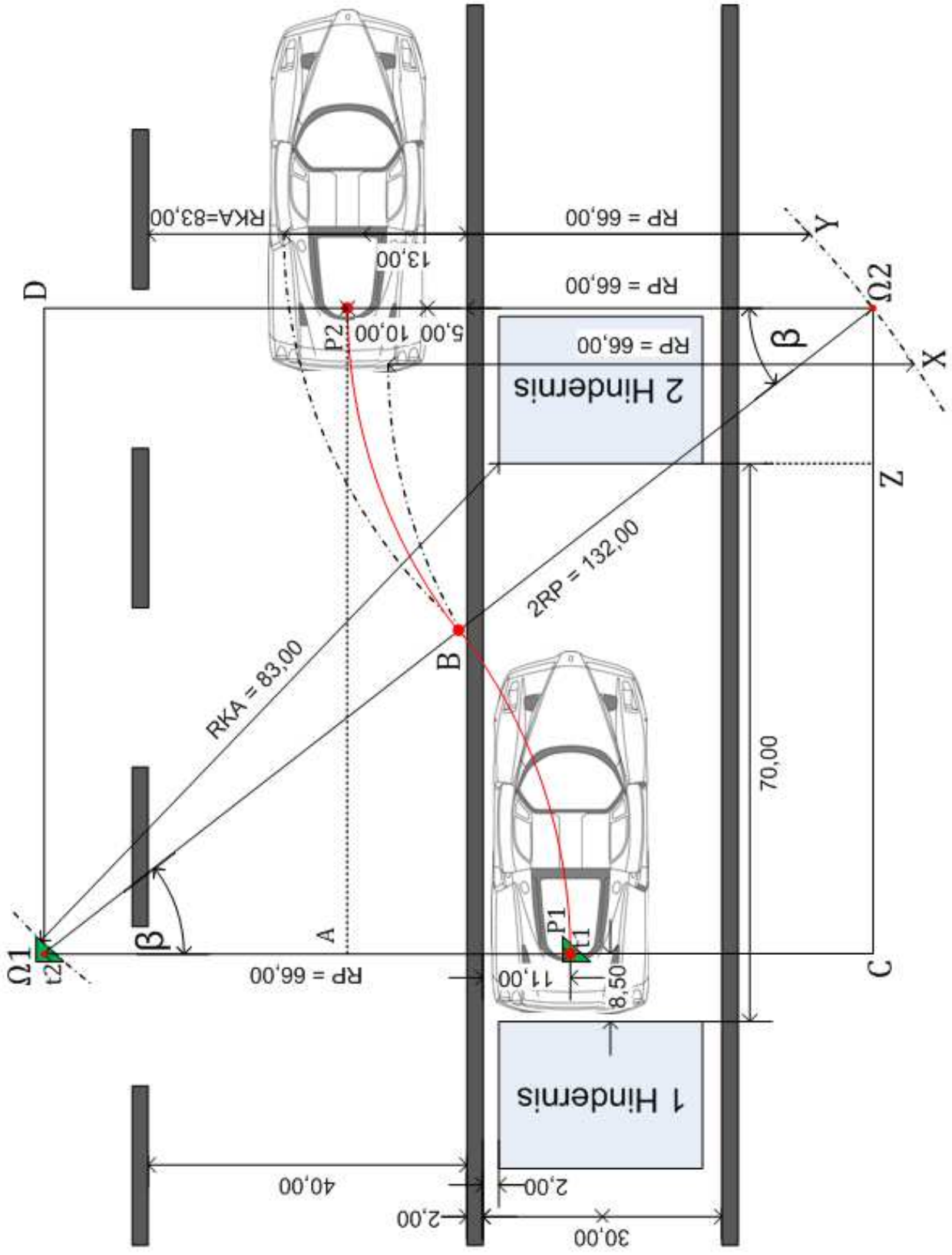

Bild C.2.: *Berechnung der Fahrwege, die das Fahrzeug für das Einparken in einem Schritt abfährt. Alle Zahlenangaben in cm*

Folgende Phasen repräsentieren den Einparkvorgang:

1. Parklückensuche: Die Fahrspurführung wird auf 5cm Abstand von der Fahrbahnbegrenzung eingestellt, die Soll-Geschwindigkeit (*z.B. 100cm/s*) vorgegeben. Der Infrarotsensor, auf der rechten Fahrzeugseite oberhalb der Hinterachse positioniert, erkennt den Anfang und das Ende der Parklücke. Durch seine Auswertung werden dem V-Regler-Modul die Aufträge zur Wegmessung erteilt und kleinere als 70cm Parklücken verworfen.

2. Positionierung: Im **Bild C.2** ist zu erkennen, dass der Positionierungspunkt P2 sich in $C\Omega 1 + 8.50cm = 89.79cm$ Abstand vom rechten Rand des 1. Hindernisses befindet ($C\Omega 1^2 = \Omega 2\ \Omega 1^2 - \Omega 2C^2 = 81.29$). Sobald die 70cm Parklücke gefunden, sowie der P2-Punkt erreicht wurden, beginnt das Fahrzeug zu bremsen (*v_soll = 0*) und misst dabei die Länge des Bremsweges.

3. Einparkphase: Wenn das Fahrzeug steht, wird die Rückwärts-Soll-Geschwindigkeit vorgegeben (*z.B. -40cm/s*), dabei führt das Fahrzeug folgende Schritte aus (*vgl.* ***Bild C.2***):

- Um den P2-Punkt zu erreichen, legt das Fahrzeug die Länge des gemessenen Bremsweges zurück.
- Im P2-Punkt führt der Lenkwinkelsteller einen Rechtsanschlag aus, das Fahrzeug fährt einen 43.78cm langen P2B-Bogen und erreicht B-Punkt.
- Im B-Punkt führt der Lenkwinkelsteller einen Linksanschlag aus, das Fahrzeug fährt einen 43.78cm langen BP1-Bogen und bleibt im P1-Punkt stehen.

Nun befindet sich das Fahrzeug in der Parklücke und steht parallel zur Fahrbahnbegrenzung, jedoch zu dicht zum 1. Hindernis, um die Parklückenmitte zu erreichen, muss das Fahrzeug noch 11.25cm vorwärts fahren und dort stehenbleiben.

D. Trial Subtraction Dividierer für 32-Bit Unsigned Integer

Der Trial Subtraction Algorithmus für die Division ist im **Listing 6.1** beschrieben; hier wird der VHDL-Entwurf eines Dividierers vorgestellt, der eine 32Bit Unsigned Integer Division ausführt. Dies bedeutet:

- Die Nachkommastellen des Divisionsprodukts werden einfach verworfen $q = \lfloor n/d \rfloor$.
- Bei Divisionen, in denen der Zähler < Nenner ist, berechnet der Divider eine 0.
- Bei Divisionen durch null besteht der 32-Bit-Quotient q nur aus Einsen.

Der Dividierer wurde nach den im **Kap.3.2** beschriebenen Techniken als Prozessor-Element mit Steuer- und Datenpfad modelliert. Das im **Kap. 6.3** vorgestellten ASM-Diagramm (*vgl.* ***Bild 6.8***) und der Zustandsautomat des Steuerpfades (*vgl.* ***Bild 6.10***) wurden weiter zur Realisierung des 32-Bit Unsigned Integer Dividierers verwendet. Man änderte nur die Eingangsparameter im Datenpfad des Dividierers (*vgl.* ***Bild D.1*** *und* ***Bild 6.9***).

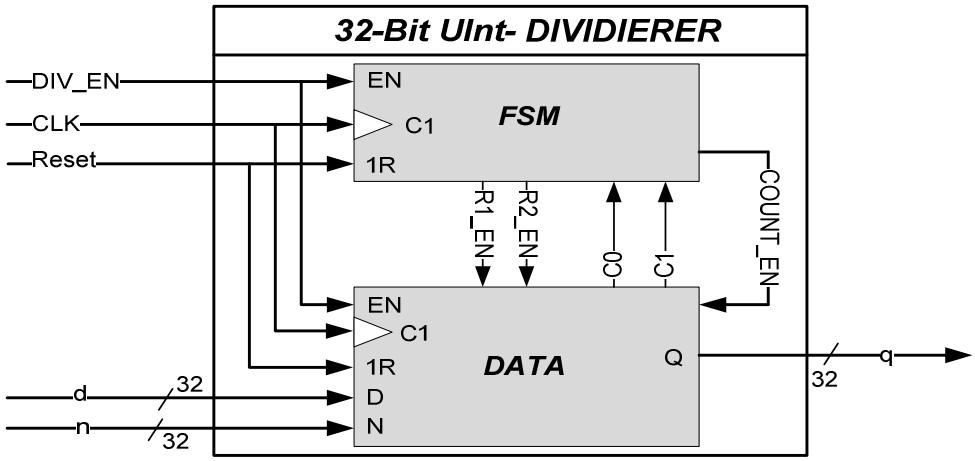

Bild D.1.: *Prozessor-Element mit Steuer- und Datenpfad, der ein 32Bit-Unsigned-Integer-Division ausführt.*

Der 32-Bit UInt-DIVIEDER wurde synthetisiert und auf das Timing-Verhalten analysiert. In der **Tabelle D.1** erkennt man die Synthesis- Place and Route- und Timig-Analyzer-Reports.

Synthesis Report			***Place and Route Report***		
Slices:	95 out of 8672	1%	External IOBs	70 out of 190	36%
Flip Flops:	69 out of 17344	1%	External Input IOBs		38
4 input LUTs:	145 out of 17344	1%	External Output IOBs		32
IOs:		100	BUFGMUXs	1 out of 24	4%
IOBs:	70 out of 190	36%	Slices	95 out of 8672	1%
GCLKs:	1 out of 24	4%	SLICEMs	0 out of 4336	0%
Timing Analyzer Report					
Der längste Signallaufpfad			Total 10.115ns (6.713ns logic, 3.442ns route)		

Tabelle D.1.: *Ressourcenbedarf und die längste Signallaufzeit des Prozessor-Elements zur 32-Bit Unsigned-Integer-Division*

D.1. Datenpfad des 32-Bit-UIntDividers als VHDL-Code

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity DATA is
   port (CLK, RESET, EN, COUNT_EN, R1_EN, R2_EN: in std_logic;
                  D: in std_logic_VECTOR (31 downto 0); --MSB...LSB
                  N: in std_logic_VECTOR (31 downto 0); --MSB...LSB
                  C0, C1: out std_logic;
                  Q: out std_logic_VECTOR (31 downto 0)); --MSB...LSB
end DATA;

architecture DATA_ARCH of DATA is
signal R1_I,R2_I,R_SUB : std_logic_VECTOR (31 downto 0); --MSB...LSB
signal COUNT : std_logic_VECTOR(4 downto 0); -- 31 to 0
begin

    SUB : process (COUNT, D, R2_I)
        variable ZERO: std_logic_vector(31 downto 0);
        begin
        ZERO:=(others=>'0');
        ZERO(31 downto CONV_INTEGER(COUNT)):=
        D((31-CONV_INTEGER(COUNT)) downto 0); -- ZERO :=  D << COUNT
        R_SUB <= R2_I - ZERO after 5 ns; -- R -= D << COUNT
    end process SUB;

    COMP0: process(COUNT, D, R2_I)
    begin
     if R2_I(31 downto CONV_INTEGER(COUNT)) >= D then -- (R >> COUNT) >= D
       C0 <= '1' after 5 ns; -- Steuerpfad => IS1-Zustand
     else
       C0 <= '0' after 5 ns; -- Steuerpfad => IS2-Zustand
     end if;
    end process COMP0;

    COMP1: process(COUNT)
    begin
     if COUNT = 0 then
        C1 <= '1' after 5 ns; -- Steuerpfad => IDLE-Zustand
     else
        C1 <= '0' after 5 ns; -- Steuerpfad => IS0-Zustand
     end if;
    end process COMP1;

    COUNTER : process (CLK)
    begin
```

```
    if CLK='1' and CLK'event then  --positive Flanke
     if RESET='1' or EN ='1' then
       COUNT <= "11111" after 5 ns; -- 31
     elsif COUNT_EN='1' then
       COUNT <= COUNT - '1' after 5 ns;
     end if;
    end if;
   end process COUNTER;

   REG1 : process (CLK)
   begin
    if CLK='1' and CLK'event then  --positive Flanke
     if RESET='1' or EN ='1' then
       R1_I <=(others => '0') after 5 ns;
      elsif R1_EN='1' then
       R1_I(CONV_INTEGER(COUNT))<='1'after 5 ns; --(1 << COUNT)
      end if;
    end if;
   end process REG1;

   REG2 : process (CLK)
   begin
    if CLK='1' and CLK'event then  --positive Flanke
     if RESET='1' or EN ='1' then
       R2_I <= N after 5 ns;
      elsif R2_EN='1' then
       R2_I <= R_SUB after 5 ns; -- R -= D << COUNT
      end if;
    end if;
   end process REG2;

Q <= R1_I after 5 ns;
end DATA_ARCH;
```

D.2. Steuerpfad des 32-Bit-UIntDividers als VHDL-Code

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity FSM is
   port (CLK, RESET, EN, C0, C1: in std_logic;
         COUNT_EN, R1_EN, R2_EN, READY: out std_logic);
end FSM;
architecture FSM_ARCH of FSM is
type STATES is (IDLE,IS0,IS1,IS2);
```

```
signal STATE, NEXT_S : STATES;
begin

  STATE_I : process (CLK)
    begin
     if CLK='1' and CLK'event then  --positive Flanke
      if RESET='1' then
         STATE <= IDLE after 5 ns;
      else
         STATE <= NEXT_S after 5 ns;    -- nächster Zustand
      end if;
     end if;
    end process STATE_I;

  STATE_M : process (EN,STATE, C0, C1)
    begin -- default Initialisierung
      COUNT_EN <= '0' after 5 ns;
      R1_EN <= '0' after 5 ns;
      R2_EN <= '0' after 5 ns;
      READY <= '0' after 5 ns;
      NEXT_S <= STATE after 5 ns;
      case STATE is
        when IDLE => if EN ='1' then
                              NEXT_S <= IS0 after 5 ns;
                     end if;
        when IS0 => if C0 ='1' then --(R >> COUNT) >= D
                              NEXT_S <= IS1 after 5 ns;
                    else              -- (R >> COUNT) < D
                              NEXT_S <= IS2 after 5 ns;
                    end if;
        when IS1 =>  NEXT_S <= IS2 after 5 ns;
                     R2_EN <='1' after 5 ns;
                     R1_EN <='1' after 5 ns;
        when IS2 =>  if C1='1' then -- COUNT = 0
                              NEXT_S <= IDLE after 5 ns;
                              READY <= '1' after 5 ns;
                     else           -- COUNT > 0
                              NEXT_S <= IS0 after 5 ns;
                              COUNT_EN <='1' after 5 ns;
                     end if;
        when others => null;
      end case;
  end process STATE_M;
end FSM_ARCH;
```

D.3. Simulation des 32-Bit-UIntDividers mit ModelSim

Die Dauer der Quotienten-Berechnung liegt im Intervall von 1300ns bis 1940ns (*vgl.* ***Bild D.3.1***). Sie ist von der Anzahl der Einsen im 32-Bit-Quotient-Vektor abhängig. Die Anzahl der Einsen im Bit-Vektor verlängert die Dauer der Berechnung. Der größte Quotient enthält 32 Einsen, für seine Berechnung laufen der Steuerpfad 32 * 3 + 1 Start = 97 Transitionen ab, mit einer pro Takt und f_{clk} = 50MHz Frequenz sind es 97 * 20ns = 2580ns. Die Berechnung einer 0, des kleinsten Quotienten, erfordert 32 * 2 + 1 = 65 * 20ns = 1920ns.

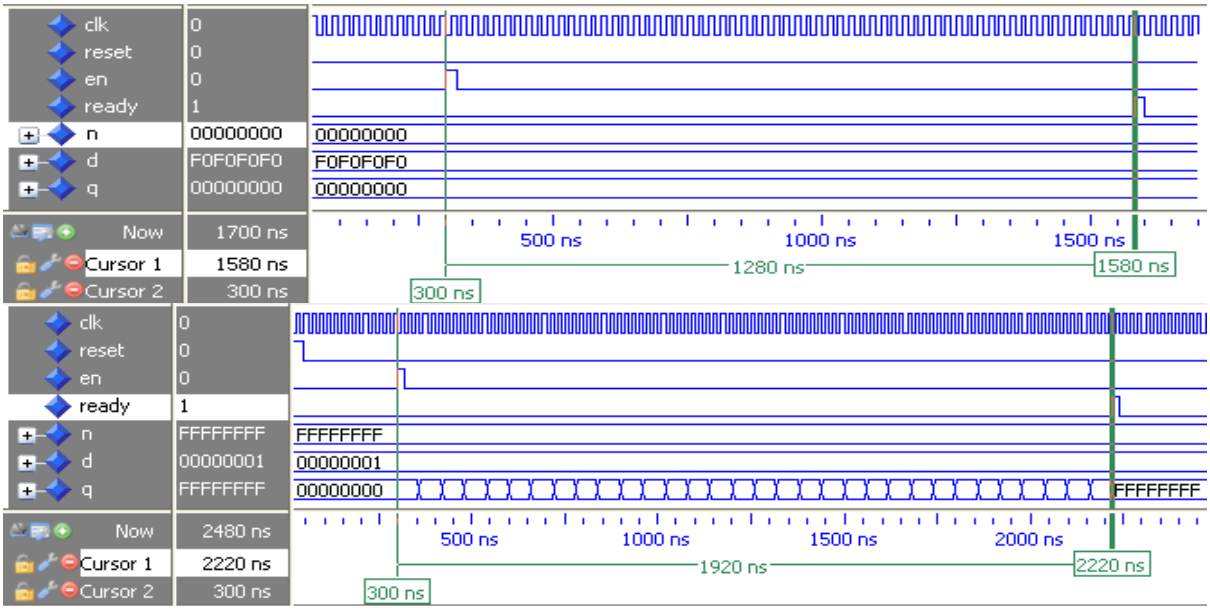

Bild D.3.1.: *Dauer der Berechnung des kleinsten und des größten Quotienten*

Eine Übersicht der Zusammenhänge zwischen Daten- und Steuerpfad des Dividierers gibt das **Bild D.3.2**. Hier sind die Sequenzen des Systems zur Berechnung der zwei höchsten Bits des Quotienten mit Pfeilen dargestellt.

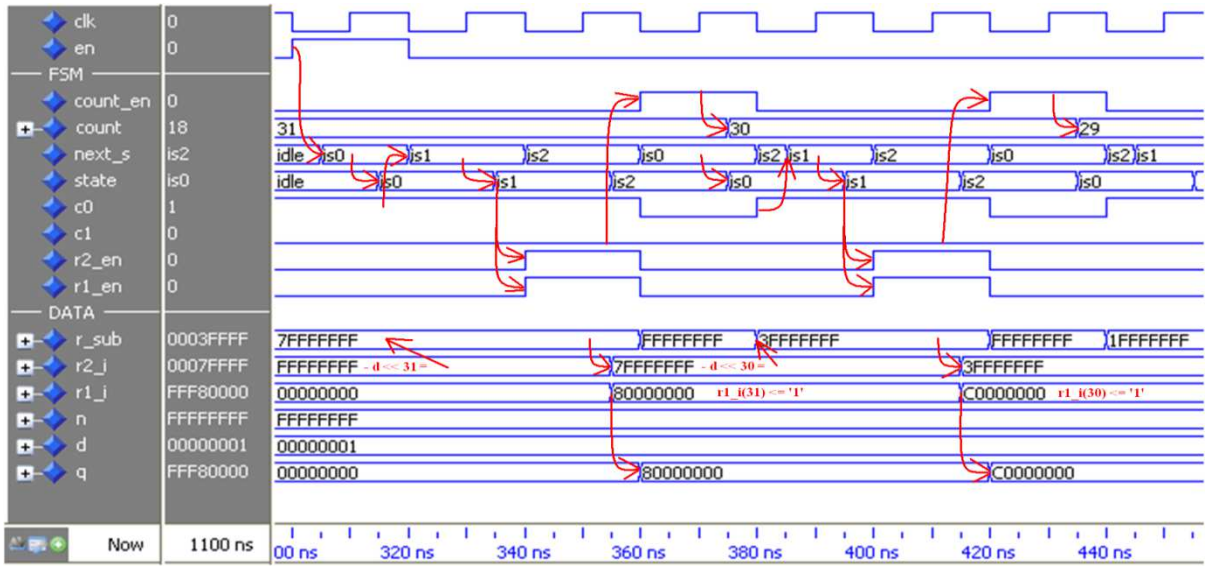

Bild D.3.2.: *Simulation des Prozessor-Elements, das eine 32-Bit Unsigned-Integer-Division nach Trial Subtraction Algorithmus ausführt.*

E. Inhaltsverzeichnis der Compact Disk

+EDK12.1_Projekte

Board Support Package der Nexys2-Plattform

EDK-Projekte der Geschwindigkeitsregelung:

- P1 und P2: PI-Regler mit einstellbaren Regler-Parametern
- PI_REG_HW: PI-Regler als Prozessor-Element
- PI_REG_SW: PI-Regler als Software-Implementierung

+ FZG-Videos

Videoaufnahmen mit dem Einsatz der entwickelten Geschwindigkeitsregelung

- 2 Video-Dateien zur Eiparkmanöverausführung
- 5 Video-Dateien zur Streckenrundfahrt

+ Grafiken

Die in der Bachelorarbeit verwendete Bilder und Visio-Dateien

+ ISE 12.1_Projekte

- UIntDiv: 32Bit Unsigned Integer Dividierer
- V_REG_62_5MHz 20ms: Die an Systemtakt f_{clk} = 62.5MHz uns PWM-Periodendauer T_{pwm} = 20ms angepasste Geschwindigkeitsregelung.

+ MATLAB

Die zur Simulation verwendeten Simulink-Modellen

+ Messungen

Die aufgenommene Sprungantworten des Fahrzeugantriebstrangs und die gemessenen Hall-Sensor-Abweichungen

+ Quellen

Verwendete Literatur und Datenblätter