HAW HAMBURG

Bachelorarbeit

Kolja Gries

Konzept und Implementierung eines System-on-Chip auf einem FPGA für die Multi-Sensor-Signalverarbeitung in autonomen Fahrzeugen

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department of Information and Electrical Engineering

Kolja Gries

Konzept und Implementierung eines System-on-Chip auf einem FPGA für die Multi-Sensor-Signalverarbeitung in autonomen Fahrzeugen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Lutz Leutelt
Zweitgutachter: Prof. Dr.-Ing. Marc Hensel

Eingereicht am: 10. August 2020

Kolja Gries

Thema der Arbeit

Konzept und Implementierung eines System-on-Chip auf einem FPGA für die Multi-Sensor-Signalverarbeitung in autonomen Fahrzeugen

Stichworte

FPGA, MicroBlaze, AXI4-Lite, Multi-Sensor-Signalverarbeitung, Ultraschallsensor

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Konzeption und Implementierung eines System-on-Chip zur Multi-Sensor-Signalverarbeitung. Dabei wird ein IP-Core mit einer AXI4-Lite-Schnittelle entworfen, welcher an einen Soft-Core-Prozessor angebunden wird. Das hierbei entwickelte System ermöglicht die Ansteuerung und Signalverarbeitung eines Ultraschallsensor-Arrays.

Kolja Gries

Title of Thesis

Concept and implementation of a system-on-chip on a FPGA for multi-sensor signal processing in autonomous vehicles

Keywords

FPGA, MicroBlaze, AXI4-Lite, multi-sensor signal processing, ultrasonic sensor

Abstract

This thesis discusses the design and implementation of a system-on-chip for multi-sensor signal processing. Therefore, an IP-Core with an AXI4-Lite interface is designed, which will be connected to a softcore processor. The developed system enables the control and signal processing of an ultrasonic sensor array.

Inhaltsverzeichnis

1 Einleitung

In den letzten Jahren erfolgten enorme technische Fortschritte im Bereich des autonomen Fahrens. Diese Schlüsseltechnologie des 21. Jahrhunderts hat große technologische, wirtschaftliche und gesellschaftliche Potentiale. Angefangen von der Entwicklung neuer Mobilitätskonzepte für den urbanen Raum, einer Reduzierung von Verkehrsunfällen bis hin zur Weiterentwicklung von künstlicher Intelligenz für die Fahrzeugsteuerung und den damit verbundenen ethischen Fragen. Schon die heutige Generation von Fahrerassistenzsystemen nutzt hierzu modernste Sensor- und Computertechnik. Hierbei wird im Automobilbereich typischerweise eine Kombination von unterschiedlichen Sensorsystemen genutzt. Diese basieren, je nach Anforderung, auf der Radar-, Laser- oder Ultraschalltechnik. Zudem kommen hochleistungsfähige Bildverarbeitungssysteme zur Anwendung. Insbesondere im Nahbereich weisen Ultraschallsensoren jedoch Vorteile gegenüber anderen Systemen auf.
Eine große technische Herausforderung die es hierbei zu bewältigen gilt, ist die Verarbeitung der auftretenden Datenmenge durch die verbaute Sensorik. Mit steigendem Automatisierungsgrad verschärft sich diese Problematik weiter. Zur Signal- und Datenverarbeitung kommen hierbei sogenannte eingebettete Systeme (engl. Embedded Systems) zum Einsatz. Als Regel-, Steuer-, oder Überwachungseinheiten werden entweder Mikroprozessoren, FPGAs oder hybride Architekturen aus beiden Hardware-Komponenten verwendet. Häufig übernehmen diese sicherheitsrelevante Aufgaben, bei dem ein fehlerfreier Betrieb sichergestellt werden muss. Die Fahrzeugelektronik ist dabei optimal für den Aufgabenbereich und deren Randbedingung auszulegen. Ansonsten kann es zu einem unerwünschten Verhalten wie Fehlfunktionen oder Datenverlust kommen. Bei der Entwicklung solcher eingebetteten Systeme, welche eine hohe Flexibilität und Rechenleitung bereit stellen müssen, kann eine problemorientierte Entwicklungsmethodik im Bereich des Hardware-Software-Codesigns angewendet werden. Hierbei erfolgt eine Betrachtung auf Systemebene, basierend auf einem simultanen Entwurf von Hard- und Software. Dadurch ist es möglich, die sich ergebenen Vorteile aus beiden Bereichen zu nutzen. Im Rahmen dieser Arbeit wird diese Entwicklungsmethodik für die Konzeption und Implementierung

eines System-on-Chip zur Multi-Sensor-Signalverarbeitung in autonomen Fahrzeugen genutzt.

1.1 Motivation und Zielsetzung

An der Hochschule für Angewandte Wissenschaften Hamburg werden im Rahmen von Bachelor-, Master- und Forschungsarbeiten diverse Projekte im Bereich des autonomen Fahrens durchgeführt. Hierbei sind unter anderem verteilte und untereinander vernetzte eingebettete Systeme zu konzipieren, welche für die autonome Fahrzeugführung benötigt werden. Während der prototypischen Entwicklung und Implementierung dieser Komponenten, stehen als Systemplattformen verschiedene RC-Modelle zur Verfügung. Hierbei ist die Komplexität der eingebetteten Systeme unterschiedlich ausgeprägt. Bei einfachen Anwendungen kommen Mikroprozessorsysteme mit einer kleinen Hardware-Architektur zum Einsatz. Bei höheren Anforderungen, wie zum Beispiel einer Bildverarbeitung, werden leistungsstärkere Einplatinencomputer eingesetzt. Diese Arbeit befasst sich mit der Konzeption und Umsetzung der ersten Implementierungsstufe eines solchen eingebetteten Systems, welches die Grundlage für weitere Studentenprojekte im Bereich des autonomen Fahrens bildet.
Ziel dieser Bachelorarbeit ist der Entwurf und die Implementierung eines System-on-Chip auf einem FPGA zur Multi-Sensor-Signalverarbeitung. Das System soll die Ansteuerung und Signalverarbeitung eines Ultraschallsensor-Arrays ermöglichen, welches zur Hinderniserkennung in einem autonomen Fahrzeug verwendet werden soll. Die Hard- und Software-Architektur ist hierbei so auszulegen, dass eine leichte Erweiterbarkeit der Funktionalität sichergestellt werden kann, da in Nachfolgeprojekten eine vollständige Implementierung der manuellen und autonomen Fahrzeugsteuerung erfolgt.
Hierfür wird zunächst eine Systemanalyse der bestehenden Fahrzeugelektronik durchgeführt, welche auf vorherigen Projekten basiert. Diese bildet den Ausgangspunkt für die Konzeption des neuen System, welches eine Multi-Sensor-Signalverarbeitung ermöglicht. Dabei werden im Rahmen einer Anforderungsanalyse die geforderten System-Spezifikationen ermittelt. Darauf aufbauend erfolgt die Konzeption und Implementierung eines Coprozessors, welcher die Signalverarbeitung durchführt. Abschließend wird eine Verifikation des Systems durchgeführt.

2 Grundlagen

Dieses Kapitel stellt die Grundlagen der Ultraschallsensorik bei der Anwendung im Automobil vor, um die wesentlichen Bestandteile zur Umsetzung der Multi-Sensor-Signalverarbeitung aufzuzeigen. Anhand des prinzipiellen Sensoraufbaus, wird das zur Entfernungsmessung genutzte Puls-Echo-Verfahren erläutert und anschließend die hierbei zu beachtenden Einflussfaktoren aufgezeigt. Nachfolgend wird der Soft-Core-Prozessor MicroBlaze vorgestellt, welcher zur Umsetzung von System-on-Chip-Architekturen genutzt werden kann. Ergänzend hierzu wird der AXI4-Lite-Schnittstellenstandard beschrieben, der zur Realisierung der On-Chip-Kommunikation von Hardware-Komponenten verwendet wird.

2.1 Anwendung von Ultraschallsensorik im Automobil

Ultraschallsensoren kommen in vielen Anwendungsbereichen zum Einsatz. Hierbei ist das Aufgabenspektrum breit gefächert. Typische Beispiele dafür sind die Hinderniserkennung, Füllstandsmessungen, Oberflächenprüfungen oder Objektvermessungen. Das Prinzip von Ultraschallmessungen beruht auf der Erzeugung und Aussendung eines hochfrequenten Sendeimpulses im Ultraschallbereich (16 kHz bis 1 GHz), welcher von einem Objekt reflektiert wird. Nach [13, S.575] liegt der technisch relevante Frequenzbereich für Luftultraschall hierbei von 20 kHz bis ca. 1 MHz. Im modernen Automobilbereich werden Ultraschallsensoren vorrangig in Verbindung mit Fahrerassistenzsystemen verwendet. Dieses sind beispielsweise Einparkhilfs-Systeme oder Spurwechsel-Assistent-Systeme [14, S.122]. Die Nutzung von Ultraschallsensoren zur Distanzmessung bzw. Objekterkennung bietet hierbei viele Vorteile. So ist durch das physikalischen Funktionsprinzips eine sehr hohe Auflösung der Struktur in Schall-Ausbreitungsrichtung erzielbar. Ultraschallsensoren zeigen sich zudem robust gegenüber rauen Umwelteinflüssen im Vergleich zu anderen Sensorarten. So ergibt sich nur eine geringe Beeinträchtigung durch Verschmutzung,

Nässe oder Nebel. Ein weiterer Vorteil ist eine vereinfachte und beschleunigte Signalverarbeitung gegenüber optischen Systemen, welche zudem eine Abhängigkeit von der Objektbeleuchtung und Objektfarbe aufweisen [20, S.1404]. Die Komplexität des Sensor-Aufbaus ist hierbei Abhängig von der geforderten Messgenauigkeit und dem vorgesehenen Anwendungsbereich. Vor diesem Hintergrund hat sich die Verwendung von Ultraschallsensoren in modernen Fahrzeugen mit Fahrerassistenzsystemen etabliert.
In den nachfolgenden Abschnitten wird der prinzipielle Aufbau von Ultraschallsensoren erläutert sowie die Entfernungsmessung nach dem Puls-Echo-Verfahren vorgestellt. Weiterführend werden verschiedene Einflussfaktoren bei Ultraschall-Messungen betrachtet.

2.1.1 Aufbau und Funktionsweise von Ultraschall-Sensoren

Abbildung 2.1 zeigt schematisch die prinzipielle Architektur eines Ultraschallsensors. Diese besteht aus einer Digitalschaltung mit definierter Zeitbasis zur Laufzeitmessung, einer anwenderspezifischen Hardware und einem Ultraschallwandler. Die Digitalschaltung dient zur Generierung des Sendesignales und Auswertung des Echosignales. Abhängig von der Bauform des Ultraschallwandlers und der benötigten Abstrahlleistung muss das erzeugte Sendesignal noch über einen Verstärker geführt werden. Der Ultraschallwandler setzt das elektrische Signal in eine Ultraschallwelle um. Nach einer Totzeit, welche vom Ausschwingvorgang des Ultraschallwandlers verursacht wird, wechselt die Digitalschaltung in den Empfangsbetrieb. Infolge einer Reflexion der ausgesendeten Impulsfolge an einem Objekt, wird der Ultraschallwandler zum Schwingen angeregt und erzeugt eine Wechselspannung. Dieses Echosignal wird nun mittels Autokorrelation gefiltert, um mögliche Störquellen auszuschließen. Anschließend erfolgt eine Berechnung der Impuls-Laufzeit, aus der die Entfernung zum Objekt bestimmt werden kann (siehe Abschnitt 2.1.2) [12, 71f]. Je nach Ausführung des Sensor-Moduls erfolgt eine weitere Daten- bzw. Signalverarbeitung durch eine anwendungsspezifische Hardware. Diese kann auch eine mögliche Schnittstelle zu externen Hardware-Komponenten bereit stellen.
In der beschriebenen Abbildung wird ein einzelner Ultraschallwandler verwendet, welcher zum Senden und Empfangen des Impulses genutzt wird. Dieses hat den Vorteil eines reduzierten Materialaufwandes, führt allerdings zu der beschriebenen Totzeit des Systems. Alternativ hierzu können kombinierte Sende-/Empfangswandlerpaare eingesetzt werden. Hierdurch entfällt die Totzeit nach dem Sendevorgang. Nachteile sind allerdings das Auftreten eines möglichen Parallaxenfehlers und die Notwendigkeit von mindestens zwei Sensorköpfen [20, S.1406].

Zur Erzeugung eines Luft-Ultraschall-Impulses werden, je nach Frequenzbereich und Anforderung, verschiedene Wandlertypen verwendet. Hierbei kommen piezoelektrische Biegewandler, piezoelektrische Dickenschwinger oder elektrostatische (kapazitive) Wandler zum Einsatz [13, S.575].

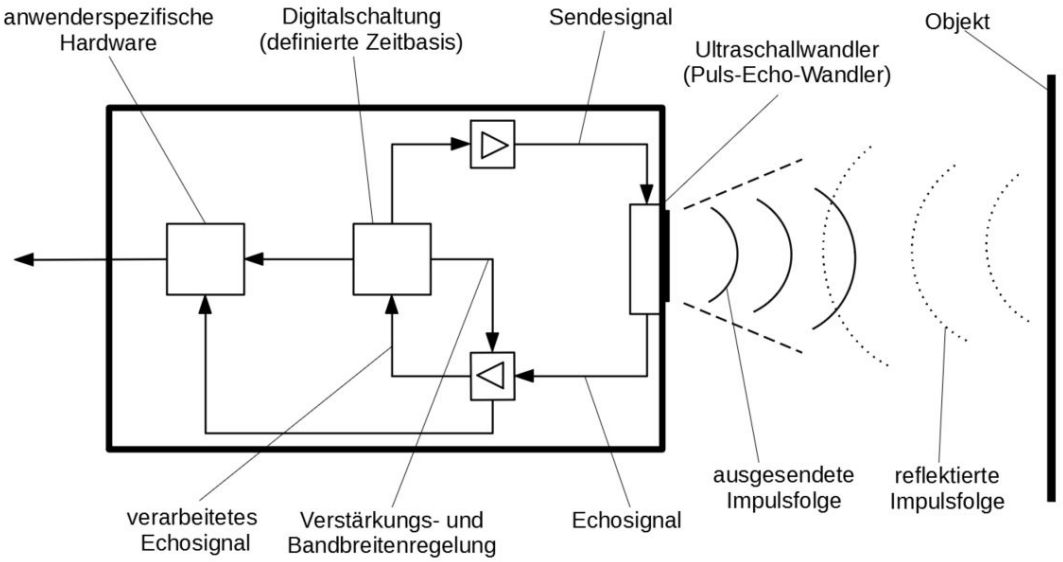

Abbildung 2.1: Schematischer Aufbau eines Ultraschallsensors (nach[13, S.581])

2.1.2 Entfernungsmessung nach dem Puls-Echo-Verfahren

Zur Bestimmung der Objektentfernung wird das Puls-Echo-Verfahren verwendet. Der Ultraschallwandler sendet hierbei eine Ultraschall-Impulsfolge (Burst) aus. Dieser dauert zwischen 100 μs und 1 ms und liegt typischerweise im Frequenzbereich von 40 kHz bis 400 kHz. Das durch Reflexion an einem Objekt entstehende Echosignal wird anschließend wieder vom Sensor detektiert [12, S.71]. Durch eine Laufzeitmessung kann nun die Objektentfernung bestimmt werden. Abbildung 2.2 verdeutlicht die Messmethode.

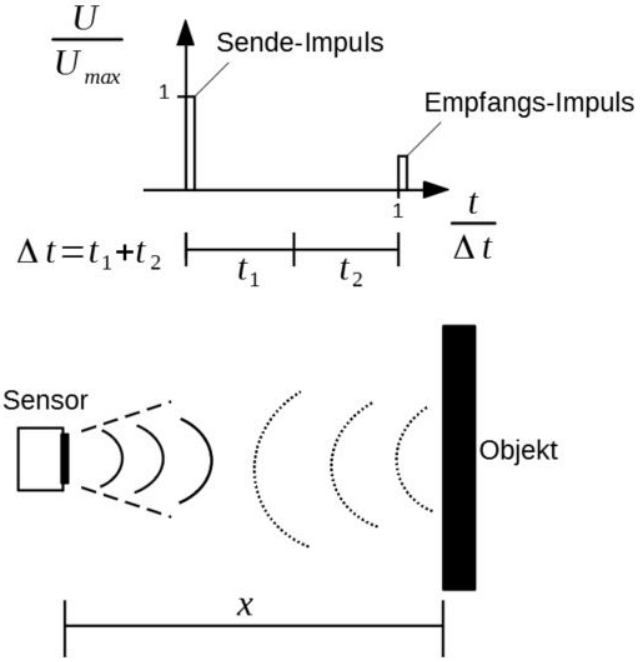

Abbildung 2.2: Entfernungsmessung nach dem Puls-Echo-Verfahren (nach [12, S.73])

Die absolute Genauigkeit der gemessene Entfernung x ist unmittelbar abhängig von der Schallgeschwindigkeit der Luft c_{Luft} , sowie der gemessenen Laufzeit Δt. Nach [13, S.581] ergibt sich die Berechnung der Distanz wie folgt

$$x = c_{Luft}\frac{\Delta t}{2} \tag{2.1}$$

2.1.3 Einflussfaktoren auf die Genauigkeit bei Ultraschall-Messungen

Bei Ultraschallmessungen wirken sich unterschiedliche Einflussfaktoren auf die erzielbare Messgenauigkeit aus. Diese können hierbei in folgende Kategorien eingeteilt werden

- Eigenschaften des Ausbreitungsmediums
- Leistungsfähigkeit des Sensors
- vorliegende Objekteigenschaften

Wie im vorherigen Abschnitt aufgezeigt, ist die Schallgeschwindigkeit eine entscheidende Eigenschaften des Ausbreitungsmediums bei der Entfernungsbestimmung von Objekten. Die Luftzusammensetzung und relative Luftfeuchtigkeit haben nur eine geringe Auswirkung auf die Ausbreitungsgeschwindigkeit der Welle [12, S.70]. Grundlegend gilt bei Gasen, dass die Schallgeschwindigkeit annähernd druckunabhängig ist, jedoch stark temperaturabhängig [20, S.613]. Nach [12, S.70] ergibt sich die Temperaturabhängigkeit der Schallgeschwindigkeit c_{Luft} hierbei aus

$$c_{Luft}(T) = c_0 \cdot \left(1 + \frac{T}{273}\right)^{\frac{1}{2}} \tag{2.2}$$

Dabei ist c_0 die Schallgeschwindigkeit bei 0 $°C$ und T die Temperatur in $°C$. Tabelle 2.1 zeigt für ausgewählte Temperaturen die sich ergebende Ausbreitungsgeschwindigkeit der Welle.

Tabelle 2.1: Schallgeschwindigkeit in Luft bei ausgewählten Temperaturen [12, S.70]

Temperartur/°C	−20	0	+20	+40	+60	+80
Geschwindigkeit/$\frac{m}{s}$	319,3	331,6	343,8	355,3	366,5	377,5

Es zeigt sich deutlich, dass für genaue Distanzmessungen Temperaturkompensationsmaßnahmen erforderlich sind. Dieses können zum Beispiel durch Messen der Umgebungstemperatur und einer anschließenden softwareseitigen Anpassung eines Korrekturkoeffizienten für die Entfernungsberechnung erfolgen.
Neben den Eigenschaften des Ausbreitungsmediums der Ultraschallwelle hat die Leistungsfähigkeit des Messsystems eine Auswirkung auf die Messgenauigkeit. Zur qualitativen Bewertung von Sensoren können nach [20, S.1405ff] folgende Begriffe definiert werden

- **axiale Strukturauflösung** - Erkennung von zwei zur Ausbreitungsrichtung des Schalls dicht hintereinander liegender Objekte
- **laterale Strukturauflösung** - Erkennung von zwei quer zur Ausbreitungsrichtung des Schalls liegender Objekte
- **Entfernungsauflösung** - Erkennung kleiner Entfernungsänderungen eines Objektes

- **Selektivität** - Erkennung kleiner Änderungen in der Objektszene

Die axiale Strukturauflösung wird maßgeblich durch den kleinsten Zeitabstand bestimmt, innerhalb dessen sich die reflektierten Echosignale noch unterscheiden lassen. Die laterale Strukturauflösung hingegen wird vom Öffnungswinkel der Stahlkeule bestimmt. Die Entfernungsauflösung hängt von den Parametern des verwendeten Ultraschallwandlers ab [20, S.1405]. Bei der Auswahl von Ultraschallsensoren muss die Leistungsfähigkeit des Systems mit den vorliegenden Anforderungen abgeglichen werden.
Einen weiteren Einfluss auf die Messgenauigkeit ergibt sich aus den Eigenschaften des Objektes. Je nach Form, Ausrichtung und Oberflächenbeschaffenheit, erfolgt eine Brechung, Streuung, Dämpfung, Abschattung oder Transmission des gesendeten Ultraschall-Impulses. Es kann hierbei auch zu einer Kombination aus den genannten physikalischen Effekten kommen. Dieses führt zu eine fehlerhaften oder ausbleibenden Detektion des Echosignales, was das Messergebnis verfälscht.

2.2 Der Soft-Core-Mikroprozessor MicroBlaze

Der Soft-Core-Mikroprozessor MicroBlaze [33] wird von der Firma Xilinx als IP-Funktionsblock zur Verfügung gestellt wird. Dieses bedeutet, dass der Prozessor nicht als physische Hardware vorliegt, sondern in der Hardwarebeschreibungssprache VHDL bzw. Verilog. Dieser zeichnet sich durch seine umfangreichen Konfigurations- und Erweiterungsmöglichkeiten [33, S.8f] aus, was eine optimierte und anwendungsbezogene Hardware-Implementierung auf einem FPGA ermöglicht.

2.2.1 Hardware-Architektur des Mikroprozessors

Der Soft-Core-Mikroprozessor kann als 32- oder 64-Bit RISC-Prozessor mit Harvard-Architektur implementiert werden. Der Mikroprozessors nutzt die folgenden Schnittstellen

- Local Memory Bus (LMB)
- Advanced eXtensible Interface (AXI4)
- AXI Coherency Extension (ACE)

Der LMB ermöglicht die Nutzung des FPGA internen Block-RAMs (BRAM). Zur Erweiterung der System-Funktionalität, wird eine AXI4-Schnittstelle verwendet, über welche weitere Peripherie-Komponenten (z.B. Interrupt-Controller) angebunden werden können. Der Cache-Zugriff wird über eine ACE- oder AXI4-Schnittstelle ermöglicht [33, S.61]. Die Verwendung der Cache-Speicher ist insbesondere bei der Nutzung FPGA externer Speicher-Komponenten wie zum Beispiel Flash oder SRAM vorteilhaft, da sich die Zugriffszeit auf Speicherelemente deutlich verringert. Hierbei liegt die nutzbare Cache-Kapazität zwischen 64 B und 64 kB [33, S.88 u.93]. Je nach System-Anwendung stehen weitere optionale Komponenten wie Multiplizierer, Teiler, Floting-Point-Unit, Barrel-Shifter oder eine Memory Management Unit zur Verfügung. Abbildung 2.3 zeigt das Block-Diagramm des MicroBlaze. Die farblich grau gekennzeichneten Komponenten sind optionale Prozessorerweiterungen.

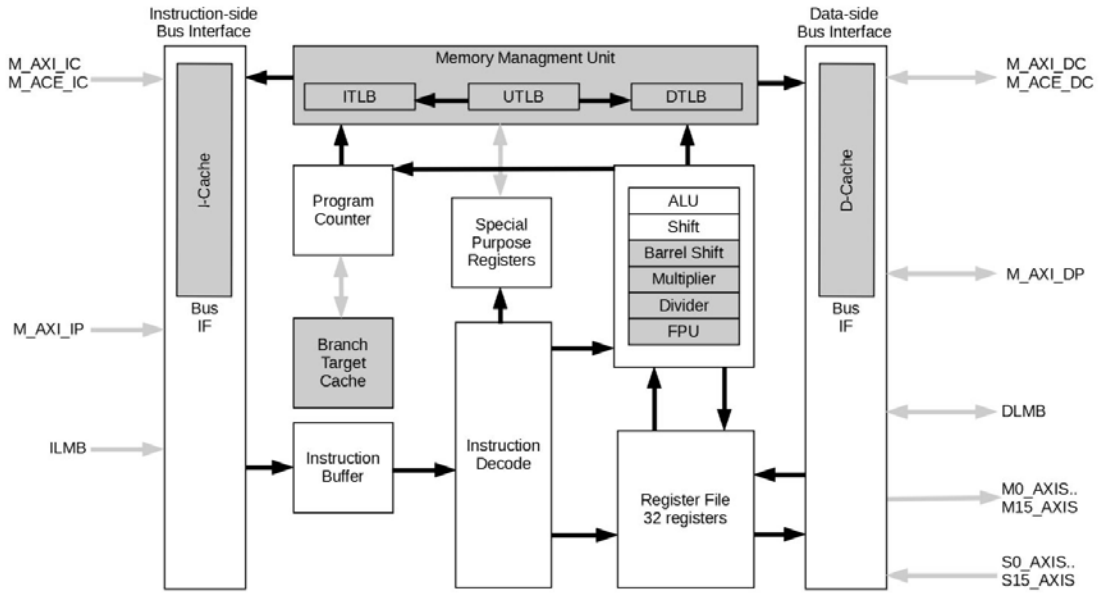

Abbildung 2.3: Hardware-Architektur des Soft-Core-Mikroprozessors MicroBlaze (nach [33, S.7])

2.2.2 Der Local-Memory-Bus

Der Local-Memory-Bus ist ein schneller, lokaler Bus zur Anbindung des Mikroprozessors an Hochgeschwindigkeitsperipheriegeräte. Typischerweise wird über den LMB der FPGA interne BRAM an den Mikroprozessor angebunden. Dieser kann dann als Daten- und Befehlsspeicher durch den MicroBlaze genutzt werden. Abbildung 2.4 zeigt das Blockschaltbild der hierbei eingesetzten IP-Funktionsblöcke. Abhängig von der verwendeten Bus-Datenbreite beträgt die maximale anschließbare nutzbare BRAM-Kapazität 256 kB [30, S.9]. Sollte ein größerer Programmspeicher benötigt werden, muss auf externe Speicherbaugruppen, wie zum Beispiel SRAM, zurück gegriffen werden.

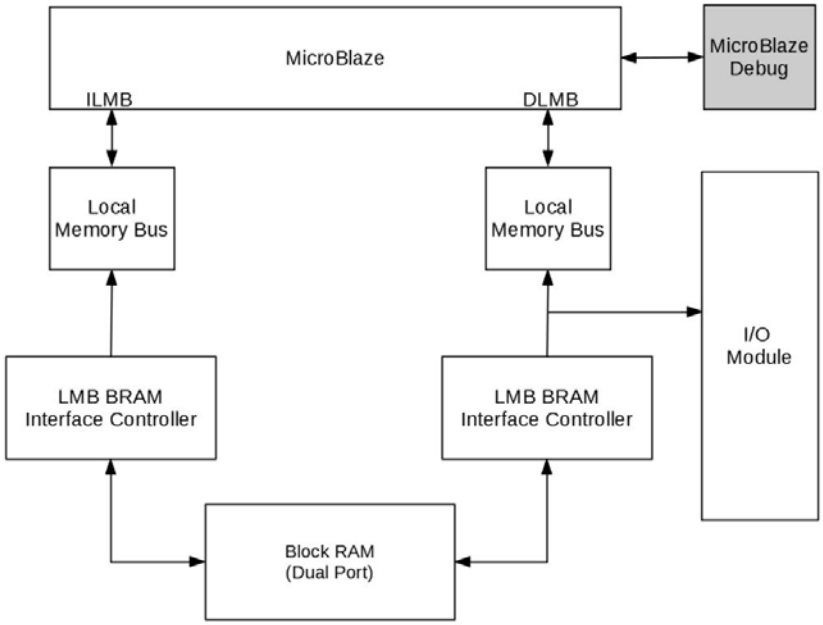

Abbildung 2.4: Anbindung des BRAM an den MicroBlaze Mikroprozessor über den Local-Memory-Bus (nach [24, S.9])

2.3 Der AXI4-Interfacestandard

Der AXI4-Interfacestandard ist Bestandteil des Advanced-Microcontroller-Bus-Architecture (AMBA) Verbindungsstandards [4]. Dieser wurde von der Firma ARM entwickelt und wird vorrangig zur Kommunikation zwischen digitalen Subsystemen und Prozessoren

genutzt [17, S.340]. Mit dem Ziel der Schnittstellen-Vereinheitlichung, hat unter anderem der FPGA Hersteller Xilinx den Standard übernommen [27]. Daher wird er häufig bei der Entwicklung und Implementierung von Sytem-on-Chip-basierten Hardware-Architekturen genutzt. Der Verbindungsstandard definiert dabei folgende Schnittstellen

- **AXI4:** Hochgeschwindigkeitsinterface, bidirektional
- **AXI4-Lite:** Teilmenge von AXI4, reduzierte Übertragungsrate, bidirektional
- **AXI4-Stream:** Punkt-zu-Punkt Stream-Verbindung, unidirektional

In den folgenden Abschnitten wird sich auf die Beschreibung der AXI4- bzw. AXI4-Lite-Schnittstelle und die genutzte Bus-Topologie.

2.3.1 AXI4-Interconnect

Die AXI4- bzw. AXI4-Lite-Schnittstelle ermöglichen eine Multi-Master-Multi-Slave Kommunikation. Zur Verbindung von mehr als zwei Bus-Teilnehmern wird ein zusätzliches AXI–Interconnect-Modul benötigt. Dabei werden Master und Slave über eine Interconnect-Komponente, wie zum Beispiel dem IP-Funktionsblock AXI-Interconnect [26] der Firma Xilinx, verbunden. Hierbei wird die Methode der speicherbezogenen Adressierung (Memory-Mapped I/O) genutzt. Abbildung 2.5 zeigt die physikalische Bus-Topologie.

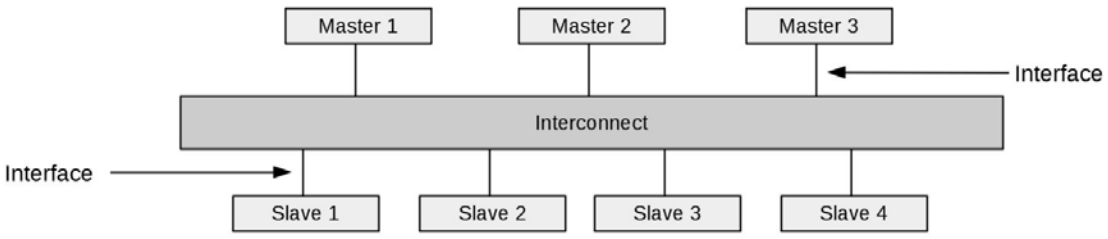

Abbildung 2.5: Physikalische Bus-Topologie der AXI4-/AXI4-Lite-Schnittstelle (nach [5, S. A1-28])

In der Abbildung 2.6 ist das Blockdiagramm des IP-Cores zu sehen. Dieser besitzt, entsprechend der Anzahl angeschlossener Bus-Teilnehmer, Master- und Slave-Schnittstellen.

Intern besitzt das Modul Koppelelemente, welche in der Abbildung als Coupler bezeichnet werden. Nach [26, S.5] bestehen diese aus den folgenden Hardware-Komponenten und ermöglichen so die Kommunikation zwischen den Bus-Teilnehmern

- **AXI Crossbar**: Verbindet ein oder mehrere Master mit ein oder mehreren Slaves
- **AXI Data Width Converter**: Ermöglicht die Konvertierung der Bus-Datenbreite
- **AXI Clock Converter**: Ermöglicht die Konvertierung des genutzten Bus-Taktes
- **AXI Protocol Converter**: Ermöglicht die Konvertierung des Schnittstellentyps
- **AXI Data FIFO**: Ermöglicht die Pufferung von Daten
- **AXI Register Slice**: Ermöglicht die Nutzung von Pipeline-Registern, typischerweise um kritische Timing-Pfad aufzulösen
- **AXI MMU**: Ermöglicht die Adressdekodierung und Adressneuzuordnung

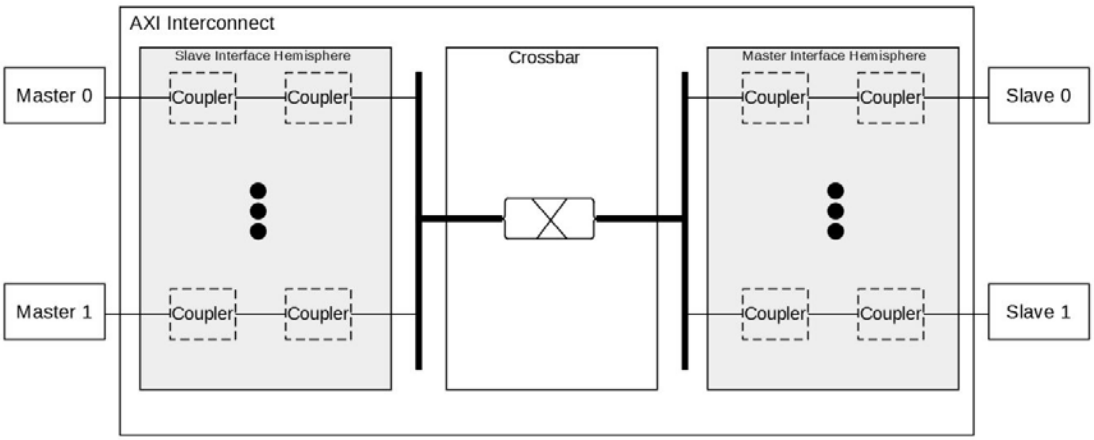

Abbildung 2.6: Blockschaltbild des AXI-Interconnect IP-Cores (nach [26, S.11])

2.3.2 Das AXI4-Protokoll

Das AXI-Protokoll [5, S. A1-25ff] nutzt insgesamt fünf unabhängige Übertragungskanäle zur Kommunikation zwischen den Bus-Teilnehmern. Zwei Adresskanäle, zwei Datenkanäle sowie einen Schreib-Antwort-Kanal. Abbildung 2.8 zeigt Beispielhaft den Master-Schreibvorgang einer AXI4-Schnittstelle. Der Master sendet über den Schreib-Adress-Kanal Information über die Art der zu übertragenden Datenpakete. Danach werden die

Daten an den Salve gesendet. Dieser bestätigt über einen separaten Schreib-Antwort-Kanal den Erhalt der Nachricht. Abbildung 2.8 zeigt einen Master-Lesevorgang. Hierbei werden lediglich zwei Übertragungskanäle benötigt. Der Master nutzt den Schreib-Adress-Kanal um dem Slave mitzuteilen, welche Informationen übertragen werden sollen. Anschließend erfolgt das Übermitteln der Daten zum Master. Einen gesonderten Lese-Antwort-Kanal gibt es nicht. Wie die Abbildungen zeigen, ist bei der Verwendung einer AXI4-Schnittstelle eine Burst-basierte Transaktionen, also das Übertragen mehrerer Datenpakete hintereinander, möglich. Hierbei muss lediglich die Startadresse angegeben werden. Im Gegensatz hierzu ist bei AXI4-Lite die Paketanzahl je Adressierung auf eins begrenzt.

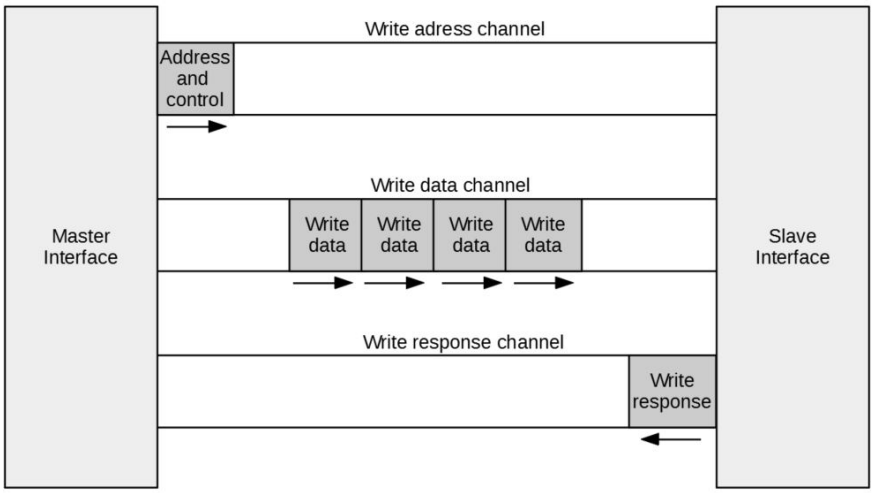

Abbildung 2.7: Master-Schreibtransaktion unter Verwendung einer AXI4-Schnittstelle (nach [5, S.A1-27])

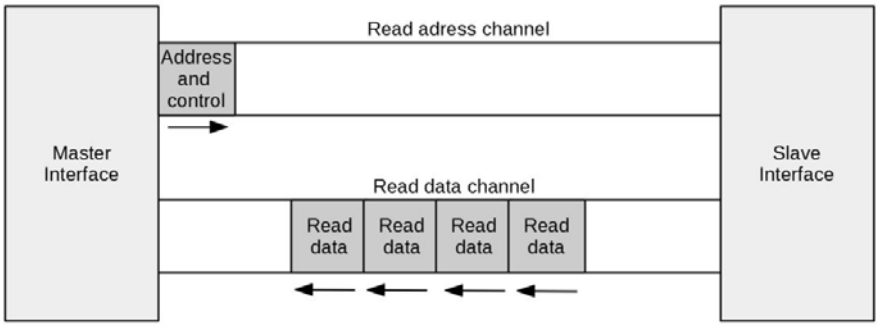

Abbildung 2.8: Master-Lesetransaktion unter Verwendung einer AXI4-Schnittstelle (nach [5, S.A1-27])

Zur Übertragung nutzen alle fünf Übertragungskanäle ein Zwei-Wege-Handshake-Verfahren [5, S. A3-41]. Hierzu wird ein Valid- und Ready-Signal je Kanal verwendet. Abbildung 2.9 zeigt das Prinzip des Handshake-Verfahrens. Das Valid-Signal wird von der Datenquelle erzeugt und zeigt an, dass Adressinformationen, Steuerinformationen oder Daten verfügbar sind (T1). Die Zielquelle erzeugt, sobald sie bereit zum Datentransfer ist, das Ready-Signal (T2). Sind beide Signale aktiv, können die Daten übertragen werden. Die Datenquelle muss hierbei sicherstellen, dass die Informationen für diesen Zeitraum stabil bleiben. Die Übertragung erfolgt bei der nächsten steigenden Taktflanke (T3).

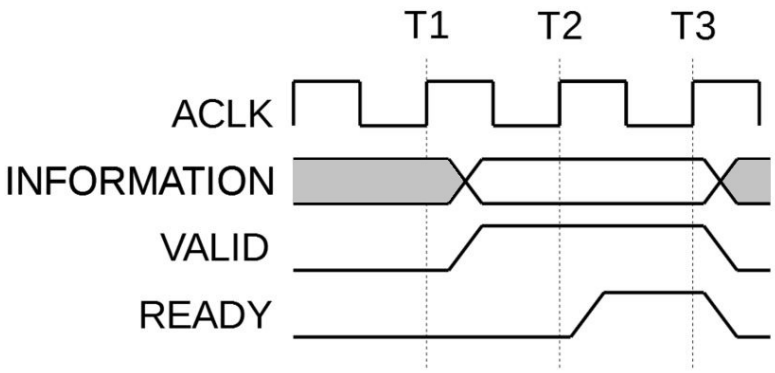

Abbildung 2.9: Prinzip des Zwei-Wege-Handshake-Verfahrens einer AXI4-Schnittstelle (nach [5, S.A3-41])

3 Systemanalyse der bestehenden Fahrzeugelektronik

In diesem Kapitel wird eine Systemanalyse der bestehenden Fahrzeugelektronik durchgeführt. Zunächst wird der Aufbau und die Funktionsweise der manuellen Fahrzeugsteuerung, sowie die Ansteuerung der Ultraschallsensoren erläutert. Ziel der anschließenden Hard- und Softwareanalyse ist es, die Stärken und Schwächen des bestehenden Systems zu erkennen. Zusätzlich werden die aktuellen Systemgrenzen ermittelt und aufgezeigt. Die gewonnenen Erkenntnisse bilden die Grundlage für spätere Entscheidungen in der Konzept- und Designphase.

3.1 Aufbau der bestehenden Fahrzeugelektronik

Abbildung 3.1 zeigt den Aufbau der bestehenden Fahrzeugelektronik. Diese setzt sich aus insgesamt zehn Hardware-Komponenten zusammen. Als zentrale Regel- und Steuereinheit wird ein Arduino Nano, basierend auf dem ATmega328P Mikrocontroller, verwendet. Über diesen erfolgt sowohl die Sensoransteuerung, als auch die Auswertung und Generierung der benötigten PWM-Signale zur manuellen Fahrzeugsteuerung. Die Erzeugung geschieht hierbei indirekt über einen PWM-Servotreiber-Modul. Dieses ist an den Mikrocontroller über eine I2C-Schnittstelle angebunden. Durch das Modul werden die vorhandenen PWM-Aktoren, wie Fahrtregler und Servomotor, angesteuert. Zum Empfang der Steuersignale wird ein UHF-Empfänger verwendet. Im bestehenden System sind zur Hinderniserkennung vier Ultraschallsensoren vom Typ HC-SR04 verbaut.

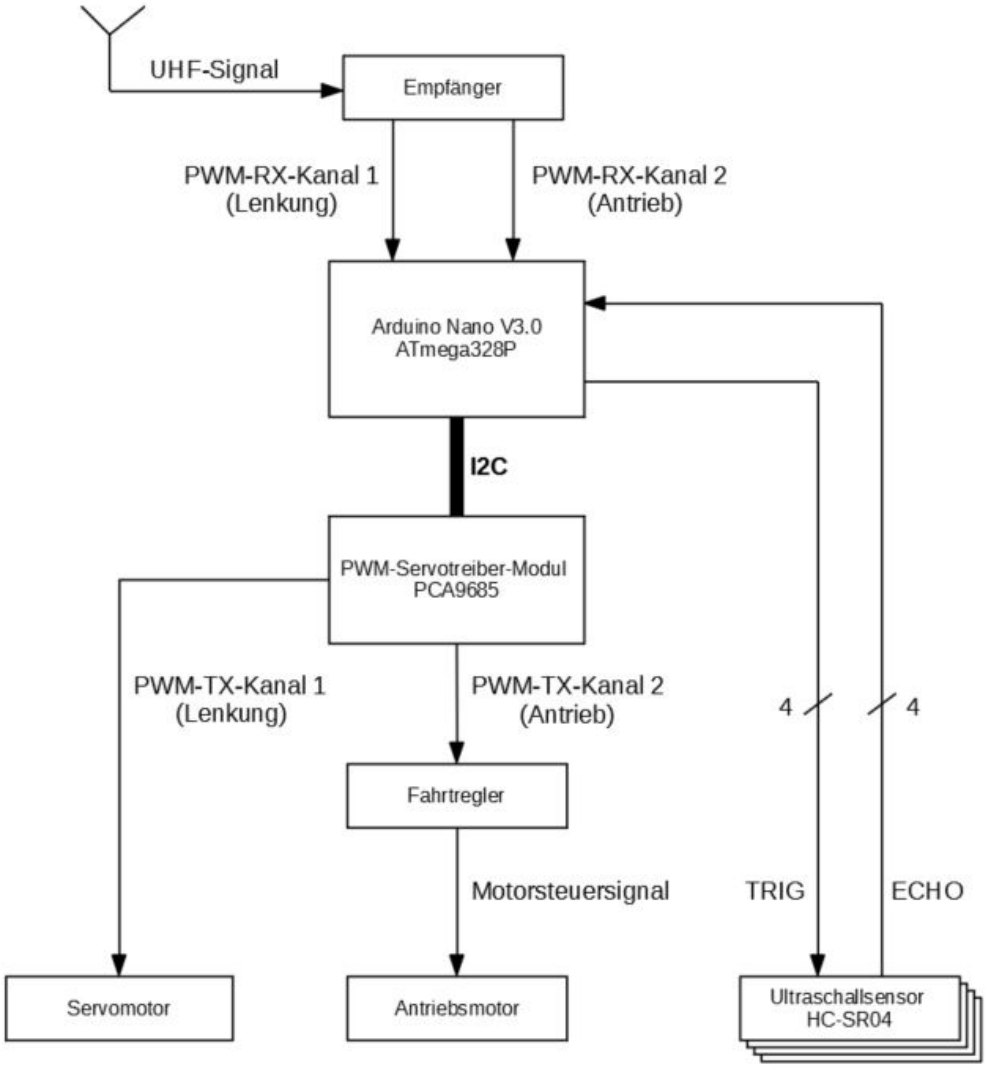

Abbildung 3.1: Aufbau der bestehenden Fahrzeugelektronik

3.2 Funktionsweise der manuellen Fahrzeugsteuerung und Sensoransteuerung

Bei Verwendung der manuellen Steuerung wird das Fahrzeug über eine Funkfernsteuerung gelenkt. Das eingehende Funksignal der Fernbedienung, welches im UHF-Frequenzband liegt, wird vom Empfänger demoduliert. Dabei wird jede Steuerinformation, wie Motordrehzahl und Lenkrichtung, in ein separates PWM-Signal umgesetzt. Der Mikrocontroller ermittelt kontinuierlich die Pulsbreite der eingehenden PWM-Signale des Empfängers. Die so empfangenen Steuerinformationen können durch die Software des Mikrocontrollers weiterverarbeitet und wenn nötig manipuliert werden. Somit ist der Arduino Nano in die Fahrzeugsteuerung eingebunden. Im manuellen Betrieb findet keine Veränderung der Steuerinformationen statt. Über eine I2C-Schnittstelle wird ein angeschlossenes PWM-Servotreiber-Modul so konfiguriert, dass es die gleichen PWM-Signale erzeugt, wie der Empfänger. Im Gegensatz zu den Lenkservomotoren wird der Antriebsmotor hierbei nicht direkt, sondern über einen Fahrtregler angesteuert. Dieser regelt, in Abhängigkeit des eingehenden PWM-Signals, den Effektivwert der Spannung und den Stromfluss zum Antriebsmotor.
Das Fahrzeug ist mit vier Ultraschallsensoren vom Typ HC-SR04 ausgestattet. Die Sensoransteuerung ist nicht durch den Fahrer beeinflussbar und geschieht unabhängig von der Fahrzeugsteuerung. Das Starten einer Ultraschallmessung erfolgt über ein Trigger-Signal, welches vom Arduino Nano erzeugt wird. Die Entfernungsinformation übermittelt der Sensor als PWM-Signal an den Mikrocontroller zurück. Anschließend wird unter Verwendung der ermittelten Pulsdauer die Objektentfernung bestimmt.

3.3 Hardwareanalyse der verwendeten Systemkomponenten

Im folgenden Abschnitt wird eine Hardwareanalyse der verwendeten Systemkomponenten durchgeführt. Zunächst wird der Aufbau und die Funktionsweise der einzelnen Module erläutert. Weiterführend wird untersucht, ob die verwendeten Hardwarekomponenten für die manuelle und in Folgeprojekten zu realisierende autonome Fahrzeugsteuerung, mit zeitgleicher Multi-Sensor-Signalverarbeitung, geeignet sind. Die gewonnenen Erkenntnisse unterstützen die anschließende Konzeptphase und erleichtern die Entscheidungsfindung bei einem eventuell durchzuführenden Hardware-Redesign.

3.3.1 Spannungsversorgung der Fahrzeugelektronik

Abbildung 3.2 zeigt die Spannungsversorgung der Systemkomponenten. Als Spannungsquelle dient ein NiMH-Akkupack. Dieses besteht aus sechs Zellen und hat eine Nennspannung von 7,2 V. Mit einer Kapazität von 1800 mAh ist der Akkupack ausreichend dimensioniert. Es liegen zwei unterschiedliche Bordnetz-Domänen im System vor. Dies ist notwendig, da nicht alle Komponenten direkt mit der Batteriespannung betrieben werden können. Der Arduino Nano und der Fahrtregler sind an das 7,2 V Bordnetz angebunden. Die Generierung der 5 V Domäne erfolgt durch Spannungswandler, welche im Fahrtregler und auf der Arduino Nano Leiterplatte verbaut sind. Hierbei beziehen das PWM-Servotreiber-Modul und die Ultraschallsensoren ihre Betriebsspannung über den Arduino Nano. Der Empfänger, sowie der Servomotor werden durch den Fahrtregler betrieben.

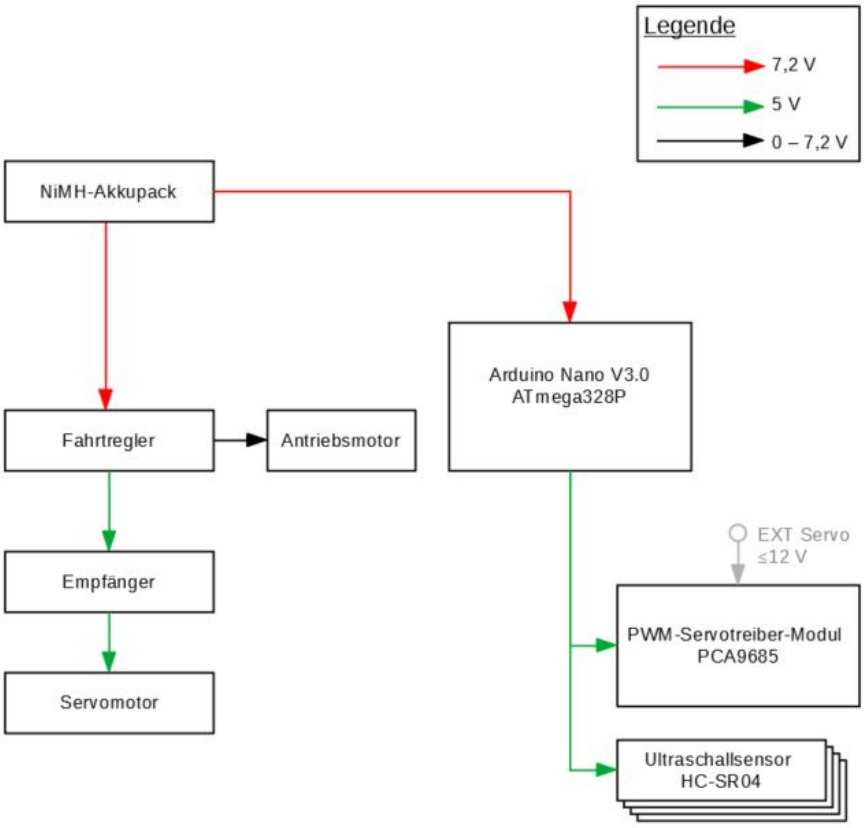

Abbildung 3.2: Spannungsversorgung der bestehenden Fahrzeugelektronik

3.3.2 Mikrocontroller-Plattform Arduino Nano V3.0

Der Arduino Nano V3.0 [1] ist eine auf dem ATmega328P [6] basierende Mikrocontroller-Plattform. Sowohl die Hard- als auch die Software sind quelloffen. Die Programmierung des Mikrocontrollers erfolgt in den Programmiersprachen C/C++ und wird über eine USB- oder ICSP-Schnittstelle geladen. Ein Vorteil der Plattform ist große Anzahl an umfangreichen Bibliotheken und die einfach zu nutzende IDE Arduino Software. Abbildung 3.3 zeigt die Mikrocontroller-Plattform.
Der ATmega328P ist ein 8-Bit AVR RISC Mikrocontroller der Firma Atmel. Auf der Plattform wird er mit seiner maximalen Taktfrequenz von 16 MHz betrieben. Der Mikrocontroller besitzt einen 32 kByte Flash-, einen 2 kByte internen SRAM- und einen 1 kByte EEPROM-Speicher. Insgesamt stehen vier Zähler zur Verfügung. Die Anbindung von weiteren externen Modulen ist unter anderem über eine I2C-Schnittstelle möglich. Die Zuführung der benötigten 5 V Betriebsspannung kann entweder über die USB-Schnittstelle oder über einen herausgeführten Pin-Anschluss erfolgen. Auf der Platine befindet sich ein Spannungsregler, welcher die zulässige Eingangspannung von 7-20 V auf die benötigten 5 V für den Mikrocontroller regelt. Zudem befindet sich ein FTDI-Chip auf dem PCB-Modul, welcher das USB-UART-Interface für die Programmierung zur Verfügung stellt. Zudem besteht die Möglichkeit, externe Komponenten mit einer Spannung von 3,3 V und 5 V zu versorgen. Die Stromaufnahme liegt bei 19 mA. Insgesamt stehen 22 digitale und 6 analoge I/O-Pins zur Verfügung. Tabelle 3.1 zeigt eine Zusammenfassung der technischen Daten.
Bei genauerer Betrachtung ist die Arduino Nano Mikrocontroller-Plattform nicht optimal für die manuelle Fahrzeugsteuerung und Hinderniserkennung geeignet. Hierbei muss eine Vielzahl von PWM-Signalen ausgewertet bzw. generiert werden, was die Nutzung von Zählern voraussetzt. Nach Datenblatt [6, S.1] besitzt der ATmega328P insgesamt vier Hardware-Zähler. Zwei 8-Bit-Zähler (Compare-Mode), einen 16-Bit-Zähler (Compare- und Capture-Mode) und einen Echtzeit-Zähler. Bei der vorhandenen Implementierung sind zwei PWM-Signale des Empfängers zeitgleich auszuwerten und dazu parallel vier PWM-Signale zur Ansteuerung der Ultraschallsensoren sequenziell zu generieren. Die geringe Anzahl an Zählern muss bei der Software-Entwicklung berücksichtigt werden und erfordert eine optimierte Hardware-Nutzung. Auf diese Voraussetzung wird in Abschnitt 3.4 näher eingegangen. Eine zusätzliche Problematik ist die 8-Bit RISC Architektur des Mikrocontrollers. Diese wirkt sich insbesondere bei Berechnungen der Sensorsignalverarbeitung mit 16-Bit-Zahlen und Zahlen vom Datentyp float negativ auf die Prozessorleitung aus. Dieses liegt an der erhöhten Anzahl an Taktzyklen für eine Berechnung,

gegenüber der ausschließlichen Verwendung von 8-Bit Zahlen. Mit der zukünftig vorgesehenen Implementierung der autonomen Fahrzeugsteuerung wird die Prozessor-Last weiter steigen. Ein Lösungsansatz wäre die Erhöhung der Taktfrequenz des Mikrocontrollers, was allerdings nicht möglich ist, da dieser schon mit der maximal zulässigen Taktrate betrieben wird. Des Weiteren bildet die I2C-Schnittstele zur Ansteuerung des PWM-Servotreiber-Moduls eine nicht zu vernachlässigende Interrupt-Last, die mit steigender Anzahl an PWM-Kanälen zunimmt.

Tabelle 3.1: Technische Kenndaten der Mikrocontroller-Plattform Arduino Nano [1][6]

Eigenschaft	**Wert**
Mikrocontroller	ATmega328P
Architektur	8-Bit AVR RISC
Flash	32 kByte
SRAM	2 kByte
EEPROM	1 kByte
Taktrate	bis 16 MHz
Zähler	Zwei 8-Bit-Counter Einen 16-Bit-Counter Einen Real-Time-Counter
ADC	8-Kanäle, 10-Bit
Schnittstellen	UART/USART PWM (6 Kanäle) TWI (I2C kompatibel) SPI (Master/Slave)
Betriebsspannung (USB/Pin)	5 V / 7-20 V
Ausgangsspannung	5 V und 3,3 V
Digital I/O Spannung	5 V
Stromverbrauch	19 mA
Ausgangsstrom I/O	max. 40 mA
Ausgangsstrom 3,3 V-Pin	max. 50 mA
Anzahl digitale I/O-Pins	22
Anzahl analoge I/O-Pins	8

3.3.3 PWM-Servotreiber-Modul

Das PWM-Servotreiber-Modul [3] ermöglicht die Generierung von PWM-Signalen. Auf der Platine befindet sich ein 16-Kanal-LED-Controller PCA96855 [16], welcher über eine I2C-Schnittstele zur Konfiguration verfügt. Die vorgesehene Slave-Adresse kann über externe Pull-Up-Widerstände festgelegt werden. Der Controller wird mit 5 V betrieben und erzeugt die benötigten PWM-Signale. Im Anwendungsfall werden jedoch nicht LEDs angesteuert, sondern der Fahrtregler und Servomotor. Außerdem besitzt das Modul einen zusätzlichen Anschluss für die Spannungsversorgung der Servomotoren (siehe Abbildung 3.2), welcher optional verwendet werden kann. Die Nutzung ist in der vorliegenden Systemarchitektur nicht vorgesehen. Die Auflösung für jeden PWM-Ausgang beträgt 12-Bit. Das bedeutet, dass bei einer im Modellbau typischen 50 Hz PWM-Ausgangsfrequenz eine Auflösung der Pulsbreite von $4,88$ µs erreicht wird.
Der Einsatz des Moduls reduziert die Prozessor-Last des ATmega328P deutlich, da dieser die PWM-Signale nicht mehr selbst erzeugen muss. Lediglich die Konfiguration des Controllers, unter Verwendung der I2C-Schnittstelle, muss erfolgen.

3.3.4 Ultraschallsensor HC-SR04

Zur Entfernungsbestimmung von Objekten werden vier Ultraschallsensoren vom Typ HC-SR04 [8] verwendet. Abbildung 3.4 zeigt das Ultraschallsensor-Modul. Es wird mit einer Versorgungsspannung von 5 V betrieben. Die messbare Distanz liegt zwischen 2 cm und ca. 300 cm. Hierbei liegt die Auflösung bei $0,3$ cm. Maximal können 50 Messungen pro Sekunde durchgeführt werden.
Der Messvorgang wird über ein Trigger-Signal bei fallender Signalflanke ausgelöst (siehe Abbildung 3.5). Nach einer Verzögerung von ca. 250 µs wird ein 40 kHz Ultraschall-Burst-Signal über den Ultraschallwandler des Moduls ausgesendet. Die Dauer beträgt hierbei 200 µs. Anschließend wird das Echo-Signal auf High-Pegel gezogen. Insgesamt ergibt sich danach eine Verzögerung von ca. 450 µs zwischen dem Auslösen einer Messung und der steigenden Signalflanke des Echo-Signales (siehe Abbildung 3.6). Bei Empfang des reflektierten Ultraschall-Bursts vom Hochfrequenzmikrofon, wird das Echo-Signal zurück auf Low-Pegel gesetzt. Erfolgt keine Reflexion, ist die maximale Pulsweite auf 200 ms limitiert. Danach kann erneut ein Messvorgang ausgelöst werden. Folglich liegt die Ultraschall-Laufzeit als PWM-Signal vor. Aus dieser Information kann wie im Grundlagenkapitel 2.1 beschrieben, die Entfernung des Gegenstandes berechnet werden. Da der

Sensor keine Temperaturkompensation der Umgebungsluft vornimmt, muss dieses bei der anschließenden Entfernungsberechnung durch die Software zur Signalverarbeitung berücksichtigt werden. Tabelle 3.2 listet die technischen Daten des Sensormoduls auf.

Tabelle 3.2: Technische Daten des Ultraschallsensors HC-SR04 [18]

Eigenschaft	**Wert**
messbare Distanz	2 cm bis ca. 300 cm
Auflösung	0,3 cm
Frequenz Ultraschall-Impuls	40 kHz
Messungen pro Sekunde	max. 50
Ruhestrom	< 2 mA
Arbeitsstrom	15 mA

Zur Hinderniserkennung ist das Modul gut geeignet, da die Genauigkeit von 0,3 cm ausreichend hoch ist. Zudem besitzt es eine geringe Leistungsaufnahme im Ruhebetrieb. Die maximale Reichweite von ca. 3 m zur Erkennung von Gegenständen ist hinreichend, da der Bremsweg des Fahrzeuges bei maximal zulässiger Geschwindigkeit im Testbetrieb kleiner 2 m ist.

3.3.5 UHF-Funkempfänger

Der 3-Kanal UHF-Empfänger des RC-Autos demoduliert die eingehenden Steuersignale der Funkfernsteuerung. Die Empfangsfrequenz liegt bei 2,4 GHz. Das Modul benötigt eine Betriebsspannung von 5 V. Jede Steuerinformation wird als separates PWM-Signal ausgegeben. Die Kanalanzahl ist für die Anwendung der manuellen Fahrzeugsteuerung ausreichend, da nur zwei Kanäle für die Lenkung und den Antrieb benötigt werden. Der dritte Kanal ist nicht belegt.

3.3.6 Fahrtregler und Antriebsmotor

Der Fahrtregler dient zur stufenlosen elektronischen Drehzahlregelung des Antriebsmotors. Bei der Ansteuerung wird eine gepulste Spannung bzw. gepulster Strom an den Antriebsmotor abgegeben. Dieses Modul ist zwingend erforderlich, da der Motor nicht direkt mit einem PWM-Signal angesteuert werden kann. Der Fahrtregler besitzt zudem

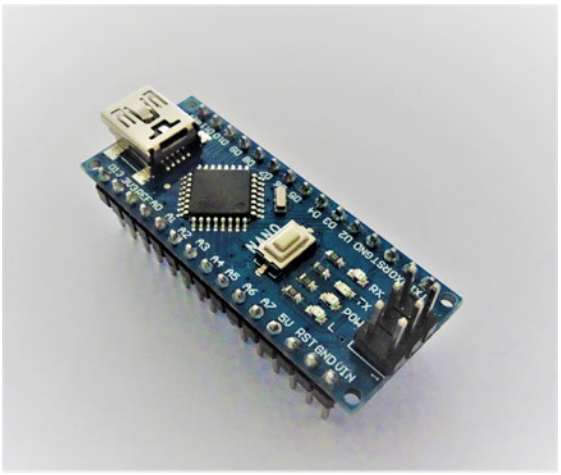

Abbildung 3.3: Arduino Nano V3.0

Abbildung 3.4: Ultraschallsensor HC-SR04

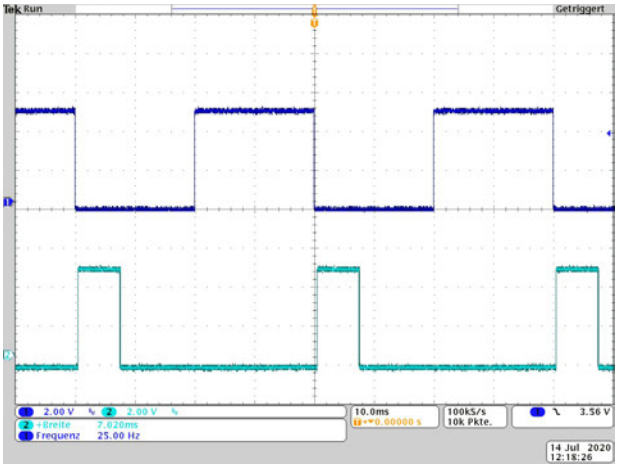

Abbildung 3.5: Verlauf des Trigger- und Echo-Signales (CH1: Trig CH2: Echo)

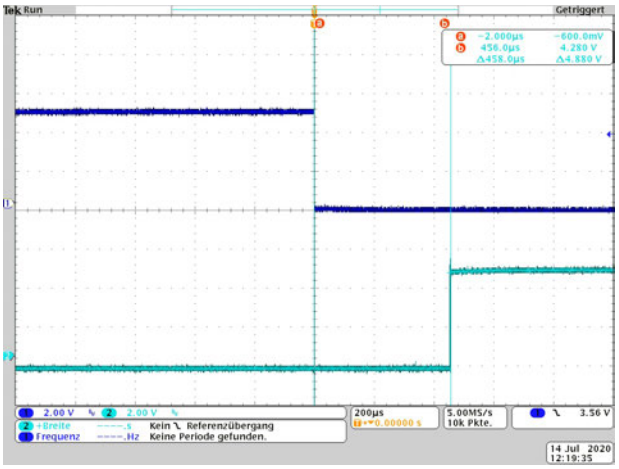

Abbildung 3.6: Zeitverzögerung zwischen Messauslösung und steigender Signalflanke des Echo-Signales (CH1: Trig CH2: Echo)

einen Spannungsregler, welcher die Batteriespannung auf 5 V wandelt. Diese wird zum Betrieb weiterer elektronischen Komponenten, wie dem UHF-Empfänger, benötigt.
Der Antriebsmotor dient zur Beschleunigung des RC-Fahrzeuges. Seine Betriebsspannung beträgt $7,2$ V. Er besitzt eine Abgabeleistung von $36,75$ W und hat eine Leerlauf-Drehzahl von 12292 Umdrehungen pro Minute. Die Umdrehungsfrequenz lässt sich über die angelegte Spannung bzw. den angelegten Strom beeinflussen.
Das Antriebssystem ist ausreichend dimensioniert, um das Gewicht der zusätzlichen Hardware-Komponenten für die Fahrzeugsteuerung und Hinderniserkennung aufzunehmen.

3.3.7 Servomotor

Der Servomotor des RC-Fahrzeuges setzt das angelegte PWM-Signal in eine Drehbewegung um. Je nach Pulsbreite des Signales erfolgt eine Drehung im oder gegen den Uhrzeigersinn. In Neutralstellung findet kein Lenkeinschlag statt und das Fahrzeug verändert seine Fahrtrichtung somit nicht.
Servomotoren können, je nach Ausführung, sehr feinstufige Drehbewegungen durchführen. Daher ist es wichtig, dass das anliegende PWM-Signal sehr genau generiert wird und einen kleinen Jitter aufweist. Bei Belastung oder blockiertem Steuerweg könne hohe Lastströme auftreten, was bei der Hardware-Dimensionierung berücksichtigt werden. Dies ist im vorliegenden System der Fall, da der Fahrtregler die Versorgungsspannung des Servomotors zur Verfügung stellt und dieser hierfür ausgelegt ist. Zudem ermöglicht der verwendete LED-Controller PCA96855 die Generierung ausreichend stabiler PWM-Signale.

3.4 Softwareanalyse der manuellen Fahrzeugsteuerung

Im folgenden Abschnitt wird die Softwareanalyse des Mikrocontrollers durchgeführt. Zunächst wird die Strukturierung der bisherigen Softwarearchitektur erläutert. Anschließend erfolgt eine Analyse der Software-Funktion pulseIn(), welche zur Pulsbreitenbestimmung eines PWM-Signals genutzt wird.

3.4.1 Strukturierung der bisherigen Softwarearchitektur

Die bestehende Implementierung umfasst zwei Software-Module. Zum einen die grundlegende Ansteuerung der Sensoren und Berechnung der Objektentfernung, zum anderen die korrekte Auswertung und Generierung der PWM-Signale, welche für die manuelle Fahrzeugsteuerung benötigt werden. Die Konzeption und Umsetzung der vorhandenen Software-Komponenten erfolgte in kleinen Projektgruppen. Unter diesen gab es keine Absprache bezüglich Schnittstellen der entworfenen Software, da die Entscheidung der Entwicklung einer manuellen Fahrzeugsteuerung nachträglich getroffen wurde. Anschließend erfolgte die Zusammenfassung und Implementierung der beiden Software-Module ohne einer weiteren Optimierung bezüglich Prozessor- und Peripherieauslastung.

3.4.2 Die Verwendung der Software-Funktion pulsIn()

In diesem Abschnitt wird die Funktion pulseIn() [2] zur Pulsbreitenbestimmung eines PWM-Signales erläutert. Diese findet Anwendung sowohl bei der PWM-Signal-Auswertung, als auch bei der Sensoransteuerung. Die Übergabeparameter der Funktion sind wie folgt definiert

```
pulseIn(int pin, int value, unsigned long timeout)
```

Als Parameter sind der GPIO-Pin und der zu messende Puls-Abschnitt (High oder Low) des einzulesenden PWM-Signals anzugeben. Optional kann noch ein Timeout übergeben werden. Als Rückgabewert wird die Pulsbreite in Millisekunden im Datentyp unsigned long zurückgeben.
Bei der Ausführung dieser Funktion wird der angegebene GPIO-Pin zyklisch abgefragt. Bei einer Übereinstimmung mit dem angegebenen Puls-Abschnitt wird ein Software-Timer gestartet. Die Funktion führt daraufhin wieder ein Polling des Eingangspins durch. Bei einem Wechsel des Signalwertes wird der Software-Timer gestoppt. Anschließend wird hieraus die Dauer der Pulsbreite bestimmt. Die maximale Wertezeit auf einen Signalwechsel kann über den Timeout-Parameter festgelegt werden. Dieses führt zu einer erhöhten Prozessor-Last. Der Programmfluss kann hierbei nur noch durch Interrupts unterbrochen werden, was sich negativ auf das Zeitverhalten der Software auswirkt.
Die Verwendung dieser Funktion verdeutlicht, dass wie in Abschnitt 3.4.1 beschrieben, nicht auf eine Prozessor- und Peripherie-Optimierte Konzeption der Software-Module geachtet wurde. Zudem findet keine Interrupt-gesteuerte Nutzung der vorhandenen Hardware-Timer statt.

3.5 Zusammenfassung der Hard- und Softwareanalyse

Aus der Hard- und Softwareanalyse wird deutlich, dass der Mikrocontroller des Arduino Nano eine Vielzahl von Aufgaben wahrnehmen muss. Die Auswertung von bis zu sechs PWM-Signalen erfordert zwingend die Verwendung von Interrupt-gesteuerten Hardware-Zählern, um eine übermäßig hohe Prozessor-Last zu vermeiden. Diese Voraussetzung ist nicht erfüllt, da ein Polling-basierter Software-Zähler verwendet wird. Zudem erzwingt die geringe Anzahl an Hardware-Zählern eine Prozessor- und Peripherie-optimierte Programmierung. Auch die Verwendung einer 8-Bit AVR RISC Mikrocontroller-Architektur stellt sich als Nachteil heraus. Insbesondere bei Berechnungen mit Zahlen vom Datentyp

float reduziert es die Prozessor-Performance. Zusätzlich steigt die Interrupt-Last des Prozessors durch die Verwendung der I2C-Schnittstelle. Hierbei werden die Polling-basierten Programmabschnitte unterbrochen und es kommt zu einer fehlerhaften Pulsbreitenbestimmung der eingehenden PWM-Signale. Somit ist die Erweiterung auf eine autonome Fahrzeugsteuerung mit einer Multi-Sensor-Datenverarbeitung ohne eine Hard- und Softwareanpassung nicht möglich.

4 Anforderungsanalyse und Konzeption des Coprozessor-Systems

Auf Grundlage der gewonnenen Erkenntnisse aus Kapitel 3, ist eine Hard- und Softwareänderung notwendig. Im Folgenden wird eine Anforderungsanalyse des neu zu entwerfenden Systems durchgeführt. Hierbei wird berücksichtigt, dass die Hard- und Software für die manuelle und autonome Fahrzeugsteuerung, unter zeitgleicher Ausführung einer Multi-Sensor-Signalverarbeitung, ausgelegt werden soll. Anschließend erfolgt die Konzeption einer Coprozessor-Architektur, welche die geforderten Spezifikationen erfüllt. Abschließend wird das Demonstrator-System zur Multi-Sensor-Signalverarbeitung entwickelt.

4.1 Anforderungsanalyse

In diesem Abschnitt wird die Anforderungsanalyse des zu entwerfenden Systems durchgeführt. Zunächst ist der erste Schritt eine Definition der Personen/-gruppen (Stakeholder), welche unmittelbar beim Systementwurf zu berücksichtigen sind. Durch eine enge Abstimmung mit diesen, werden Systemanwendungsfälle erarbeitet. Hieraus lassen sich systematisch konkrete und überprüfbare Anforderungen an die zu entwickelnde System-Architektur herleiten. Ziel ist es, die Spezifikationen vollständig und korrekt zu erfassen, da diese die Grundlage für Entscheidungen in der nachfolgenden Konzept- und Designphase bilden.

4.1.1 Stakeholder-Analyse

Im Rahmen der Anforderungsanalyse für diese Arbeit sind fünf Stakeholder ermittelt worden. Diese weisen unterschiedliche Profile auf, die bei der Erarbeitung von Anwendungsfällen und den daraus abgeleiteten System-Spezifikationen berücksichtigt werden

müssen. Hierbei wird die Planungsebene durch den Projektleiter, die Ausführungsebene durch die Entwickler und die Benutzerebene durch den Fahrer in die Anforderungsanalyse mit einbezogen. Dies ermöglicht die unterschiedlichen Interessen, Anforderungen und Erwartungen an das zu entwickelnde System aus unterschiedlichen Perspektiven zu ermitteln und anschließend zu bewerten.
Der Projektleiter ist die bestimmende Kontaktperson bei der Definition von Anwendungsfällen. Durch eine gezielte Fragestellung werden klar definierte Spezifikationen erarbeitet und festgelegt. Die Entwickler der Projektgruppe führen im Anschluss an diese Arbeit ergänzende Hard- und Software-Erweiterungen durch. Daher muss vor allem die Wartbarkeit und Komplexität des zu entwerfenden Systems auf diese Personengruppe abgestimmt sein. Der Fahrer steuert das Automodell und initiiert die Umschaltung der gewünschten Steuerungsart. Dies soll ohne detailliertes Wissen der technischen Funktionsweise des Systems möglich sein. Tabelle 4.1 listet die Stakeholder.

4.1.2 Definition der Systemanwendungsfälle

Für die genaue Spezifikation des zu entwerfenden Systems werden zunächst Systemanwendungsfälle definiert. Diese bilden die Funktionalitäten aus Sicht des Nutzers und die externer System-Schnittstellen ab. Zur Visualisierung der Anwendungsfälle werden sogenannte Anwendungsfall-Diagramme verwendet. Abbildung 4.1 zeigt Darstellung für die zu entwerfende Fahrzeugsteuerung und Multi-Sensor-Signalverarbeitung aus Sicht des Fahrers. Bei der manuellen Steuerung kann dieser die Geschwindigkeit und Richtung des Fahrzeuges beeinflussen. Ist die autonome Steuerung aktiviert übernimmt die Regel- und Steuereinheit des Systems die Kontrolle. Das Erkennen von Gegenständen durch die Ultraschallsensoren erfolgt unabhängig von der gewählten Steuerungsart.

4.1.3 Anforderungsspezifikation

In diesem Abschnitt erfolgt die Anforderungsspezifikation des zu entwerfenden Systems. Durch eine gezielte Fragestellung an die Stakeholder und eine kritische Überprüfung jeder Anforderung wird sichergestellt, dass in dieser wichtigen Projektphase keine Spezifikation vergessen oder falsch interpretiert wird. Hierbei erfolgt eine Unterteilung in zwei Grundtypen. Zum einen in die funktionalen Anforderungen, welche fest legen was das System tun soll und zum anderen die nicht-funktionalen Anforderungen. Diese beschrieben quantitativ wie gut das System funktionieren muss und umfassen unter anderem

Tabelle 4.1: Ermittelte Stakeholder im Rahmen der Anforderungsanalyse

Name	**Profil**
Projektleiter	Seltener Anwender des Systems. Zuweisung von Teilprojekten an Entwickler Level 1-3. Unterstützt bei der Konzeption von Softwaremodulen. Keine Kenntnisse in der Hardwarebeschreibungssprache VHDL.
Entwickler - Level 1	Anwender des Systems beim Entwurf und Umsetzung von Teilprojekten. Häufig ein Student im Grundstudium* mit keiner oder geringer Erfahrung in der Programmiersprache C. Keine Kenntnisse in der Hardwarebeschreibungssprache VHDL.
Entwickler - Level 2	Anwender des Systems beim Entwurf und Umsetzung von Teilprojekten. Häufig ein Student im Hauptstudium* mit Erfahrung in der Programmiersprache C. Gefestigte Kenntnisse in der Hardwarebeschreibungssprache VHDL.
Entwickler - Level 3	Anwender des Systems beim Entwurf und Umsetzung von Teilprojekten. Person mit Bachelor- oder Masterabschluss* und weitreichender Erfahrung in der Programmiersprache C. Umfassende Kenntnisse in der Hardwarebeschreibungssprache VHDL.
Fahrer	Häufiger Anwender des Systems. Wenig bis kein detailliertes Wissen der Fahrzeugelektronik vorhanden. Keine Programmierkenntnisse. Steuert das RC-Automodell.

**Informations- und Elektrotechnik, Informatik, o.Ä.*

Fehlertoleranzen, Antwortzeiten und den festgelegten Ressourcenbedarf.
Tabelle 4.2 zeigt die ermittelten Spezifikationen, aus denen sich die Konzeption eines Coprozessor-Systems ergibt. Diese besitzen zur eindeutigen Zuordnung eine Zahlen-Buchstaben-Kombination. Hierbei steht F für eine funktionale Anforderung und N für eine nicht-funktionale Anforderungen. Zusätzlich wird jeder Spezifikation eine Priorität zugewiesen. Diese gibt an, ob die Umsetzung zwingend erforderlich ist (hoch), in Folgeprojekten erflogen kann (niedrig) oder ob diese nicht zwingend erforderlich ist (optional).

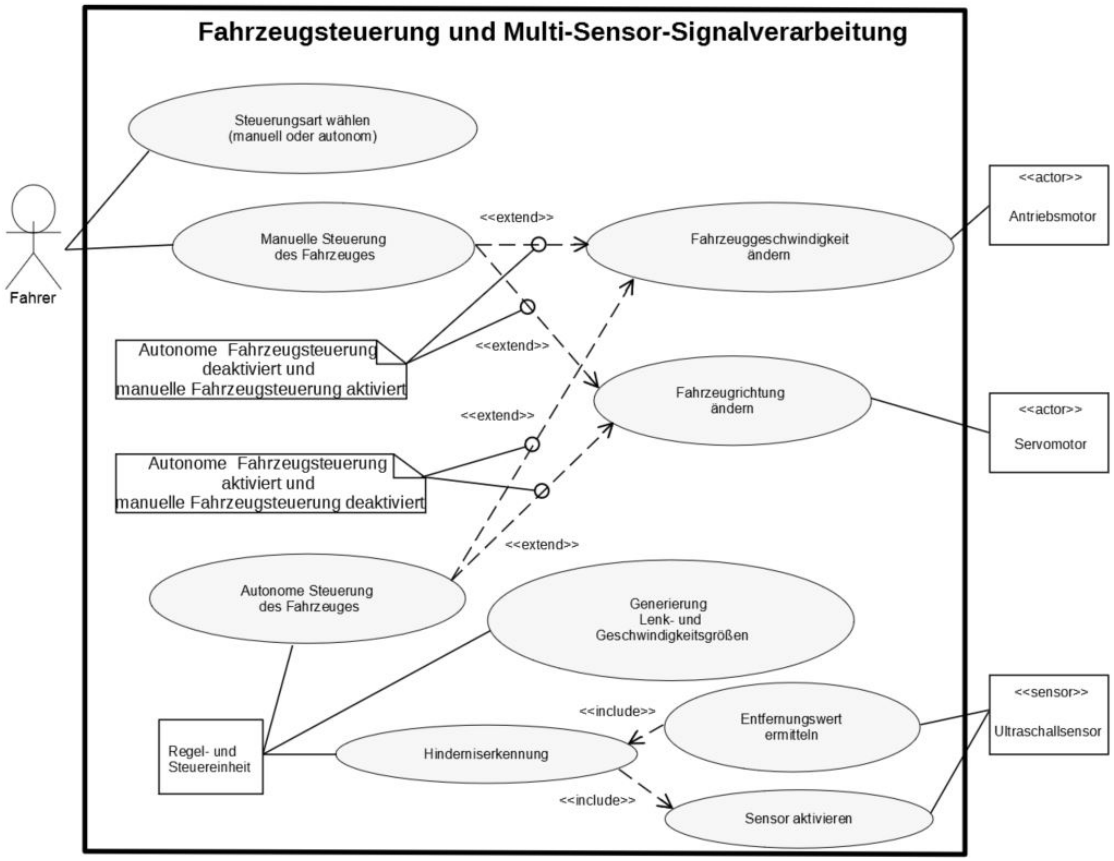

Abbildung 4.1: Anwendungsfall-Diagramm der Fahrzeugsteuerung und Multi-Sensor-Signalverarbeitung

Tabelle 4.2: Ermittelte System-Spezifikationen im Rahmen der Anforderungsanalyse

ID	Anforderung	Priorität
F1	Die System-Hardware soll so ausgelegt sein, dass es die manuelle und autonome Fahrzeugsteuerung durch den Fahrer ermöglicht.	hoch
F2	Das System soll so ausgelegt sein, das es über vorhandene Arduino-Bibliotheken angesteuert werden kann.	hoch
F3	Das System soll die Umstellung der Steuerungsart (manuelle oder autonome Steuerung), welche vom Fahrer initialisiert wird, ermöglichen.	hoch

Fortsetzung auf der nächsten Seite

Tabelle 4.2: Ermittelte System-Spezifikationen im Rahmen der Anforderungsanalyse

Nr.	Anforderung	Priorität
F4	Das System soll eine Fail-Save-Routine bei einem auftretenden Spannungseinbruch am Mikrocontroller ausführen.	hoch
F5	Ein Mikrocontroller-Ausfall und die aktivierte Steuerungsart soll optisch angezeigt werden.	niedrig
F6	Das System soll die PWM-Signale des Empfängers auslesen.	hoch
F7	Das System stellt dem Mikrocontroller die Empfängerwerte über eine Schnittstelle zur Verfügung.	hoch
F8	Das System stellt dem Mikrocontroller die Sensorwerte zur Verfügung.	hoch
F9	Das System soll eine Erkennung des angeschlossenen Sensor-Typs ermöglichen.	optional
F10	Das System soll die Konfiguration der Sensoransteuerung über eine Schnittstelle ermöglichen.	hoch
F11	Das System soll die Sensoren gemäß der Konfiguration durch den Mikrocontroller ansteuern.	hoch
F12	Der Messvorgang soll separat für jeden Sensor durch den Mikrocontroller gestartet werden können.	hoch
F13	Die Ansteuerung der Sensoren soll sequenziell erfolgen.	hoch
F14	Das System soll die variable Verschaltung der PWM-Rx- und PWM-Tx-Kanäle untereinander durch den Mikrocontroller ermöglichen.	optional
F15	Die PWM-Kanäle sollen separat durch den Mikrocontroller ein- und ausgeschaltet werden können.	hoch
F16	Das System soll die Konfiguration der PWM-Signale zur Ansteuerung der Aktoren über eine Schnittstelle ermöglichen.	hoch
F17	Das System soll die PWM-Aktoren gemäß der Konfiguration durch den Mikrocontroller ansteuern.	hoch

Fortsetzung auf der nächsten Seite

Tabelle 4.2: Ermittelte System-Spezifikationen im Rahmen der Anforderungsanalyse

Nr.	Anforderung	Priorität
N1	Das System soll die Mikrocontroller-Plattform Arduino Nano bzw. Raspberry Pi als zentrale Regel- und Steuereinheit nicht ersetzen (Slave-Funktion).	hoch
N2	Das System soll so ausgelegt werden, dass in der Entwicklungsphase eine hohe Flexibilität von Hard- und Software gegeben ist.	hoch
N3	Das System soll kompatibel für die Plattformen Arduino und Raspberry Pi sein.	hoch
N4	Die Slave-Adresse des Systems soll softwareseitig konfigurierbar sein.	hoch
N5	Das Wiederholintervall der Sensoren soll zwischen 0 s und max. 3 s betragen.	hoch
N6	Das System soll die Ansteuerung von bis zu sechs Ultraschallsensoren vom Typ HC-SR04 ermöglichen.	hoch
N7	Die Auflösung der Ultraschall-Entfernungsmessung soll kleiner 50 mm sein.	hoch
N8	Das System soll die Entfernungsinformation als Zählerwert (Echo-Pulsbreite in CLK-Ticks) an den Mikrocontroller übertragen.	hoch
N9	Die Auslösung der Fail-Save-Routine soll in weniger als 500 ms erfolgen	hoch
N10	Die Sensorwerte sollen als Datentyp unsigned (16 Bit) im Mikrocontroller abgebildet werden.	hoch
N11	Die Auswahl der Steuerungsart erfolgt in der Testphase über einen Kontakt, der vom Mikrocontroller nach dem Reset abgefragt wird.	hoch
N12	Das System soll die Ansteuerung von bis zu vier PWM-Modulen ermöglichen.	hoch

Fortsetzung auf der nächsten Seite

Tabelle 4.2: Ermittelte System-Spezifikationen im Rahmen der Anforderungsanalyse

Nr.	Anforderung	Priorität
N13	Das System soll bis zu vier PWM-Empfängerkanäle auslesen können.	hoch
N14	Das System soll das einmalige Auslösen eines Ultraschallsensors ermöglichen (One-Shot-Mode).	hoch

4.2 Konzeption des Coprozessor-Systems

Auf Grundlage der festgelegten Spezifikationen wird in diesem Abschnitt die Konzeption des Coprozessor-Systems durchgeführt. Zunächst werden verschiedene Realisierungsmöglichkeiten bei der benötigten Datenbus-Architektur und der verwendeten Hardware-Plattformen diskutiert. Die Konzeptentscheidung erfolgt anhand von technischen und wirtschaftlichen Prüfkriterien. Aus dem Konzept werden Funktionsweisen bzw. Interaktionen einzelner Komponenten ersichtlich, die in der anschließenden Design-Phase berücksichtigt werden. Da keine vollständige Implementierung der autonomen Fahrzeugsteuerung erfolgt, werden nicht alle ermittelten Anforderungen in dieser Arbeit umgesetzt. Dessen ungeachtet müssen diese gleichwohl in der Konzept- und Designphase beachtet werden.

4.2.1 Vergleich und Bewertung möglicher Datenbus-Architekturen

Bei der Konzeption eines Coprozessor-Systems hat die Auswahl einer geeigneten Bus-Architektur zur Kommunikation zwischen den Prozessor-Systemen einen erheblichen Einfluss auf dessen Leistungsfähigkeit. Je nach vorgesehener Anwendung müssen hierbei unter anderem die Taktrate, das Übertragungsverfahren und die Anzahl der benötigten Leitungen berücksichtigt werden. Bei geforderten Echtzeitbedingungen an das Coprozessor-System, muss insbesondere die Datenübertragungsrate ausreichend groß sein, um die ansonsten auftretenden Latenzzeiten bei einer Übertragung der Daten gering zu halten. Der Mikrocontroller des Arduino Nano besitzt zwei serielle Schnittstellen, welche für

Tabelle 4.3: Ausgewählte Parameter der SPI- und I2C-Busarchitektur [15] [10]

Eigenschaft	**I2C**	**SPI**
Taktrate	100 kHz (Standard) 400 kHz (Fast-Mode) 1 MHz (Fast-Mode-Plus) 3,4 MHz (High-Speed-Mode) 5 MHz (Ultra-Fast-Mode)	Nicht spezifiziert, Hardware abhängig, typisch: kHz- bis MHz-Bereich
Protokoll	Multi-Master-Multi-Slave	Single-Master-Multi-Slave
Verfahren	seriell, Halb-Duplex	seriell, Voll-Duplex
Anzahl Leitungen	2	4

die Kommunikation verwendet werden können. Eine SPI-Schnittstelle sowie eine I2C-Schnittstelle. Tabelle 4.3 listet ausgewählte Eigenschaften dieser Bussysteme auf.
Der I2C-Bus [15] ein serieller Datenbus. Es stehen unterschiedliche Taktraten zwischen 100 kHz und 5 MHz zur Verfügung. Die Übertragung erfolgt im Halb-Duplex-Verfahren und benötigt zwei Leitungen. Das I2C-Protokoll ermöglicht eine Multi-Master-Multi-Slave Verbindung, wobei zwei Leitungen zur Übertragung verwendet werden.
Das SPI-Bussystem [10] ist ebenfalls ein serieller Datenbus, der im Voll-Duplex-Verfahren arbeitet. Eine maximale Taktfrequenz ist für dieses Bussystem nicht spezifiziert. Diese liegt bei vielen Anwendungen in einem zweistelligen MHz-Bereich, welche allein durch die höchstmögliche Taktgeschwindigkeit der Hardware-Komponenten und durch die Übertragungseigenschaften der Busleitungen begrenzt wird. Bei der Verwendung von je einem Master und Slave, werden vier Leitungen benötigt. Für jeden weiteren Slave kommt eine Slave-Select-Verbindung hinzu.
Bei der Realisierung des Coprozessor-Systems wird eine I2C-Schnittstelle verwendet, da die mögliche maximale Taktrate von 5 MHz ausreichend für die geplante Anwendung ist. Zudem sind weniger Bus-Leitungen notwendig und die angeschlossenen Slave-Module können mit einer Adresse versehen werden, was beim SPI-Bussystem nicht möglich ist. Auch ist die Komplexität der Hard- und Software bei der Verwendung von I2C geringer, da weniger Konfigurationsmöglichkeiten und Betriebsarten zur Verfügung stehen.

4.2.2 Vergleich und Bewertung unterschiedlicher Hardware-Plattformen für den Coprozessor

Zur Realisierung des Coprozessors können unterschiedliche Hardware-Plattformen verwendet werden. Die geforderten Spezifikationen sind sowohl von einem Mikrocontroller, als auch von einem CPLD oder FPGA erfüllbar. Diese Hardware-Plattformen weisen jedoch, je nach Anwendungsfall, unterschiedliche Vor- und Nachteile auf. Zur Entscheidungsfindung sind neben dem allgemeinen Funktionsprinzip, auch weitere Prüfkriterien zu berücksichtigen. Dieses sind unter anderem Stromverbrauch, Signallaufzeiten, vorhandene Peripherie und Stückpreis. Tabelle 4.4 zeigt eine Gegenüberstellung der unterschiedlichen Hardware-Plattformen und der Eigenschaften.
Der Aufbau und das Funktionsprinzip zwischen den einzelnen Komponenten unterscheiden sich erheblich. Bei Mikrocontrollern liegt eine klassische Rechner-Architektur (Harvard oder Von-Neumann) vor, welche Befehle sequenziell verarbeitet. Hierdurch sind diese, im Vergleich zu programmierbaren logischen Schaltungen, langsamer in der Datenverarbeitung. Die vorhandene On-Chip-Peripherie ist starr, das bedeutet sie kann nicht nachträglich erweitert oder ergänzt werden. Bei einer Anpassung der Spezifikationen, wie zum Beispiel dem Benötigen zusätzlicher Schnittstellen, muss der Mikroprozessor getauscht oder durch externe Hardware-Module ergänzt werden. Ein großer Vorteil zeigt sich allerdings bei der Implementierung von komplexen Berechnungen, was bei Mikrocontrollern in der Regel einfacher umzusetzen ist. Zudem liegt der Stückpreis für einfache Modelle weit unter dem von FPGAs. Auch ist die Stromaufnahme in der Regel geringer. FPGAs und CPLDs kommen häufig dort zur Anwendung, wo eine sehr schnelle Datenverarbeitung notwendig ist. Dies resultiert aus der frei wählbaren Implementierung von logischen Schaltungen und einer parallelen Verarbeitung der Signale. Ein Nachteil ist der höhere Stückpreis gegenüber Mikroprozessoren.
Die Unterschiede zwischen CPLDs und FPGAs werden erst bei einer genaueren Betrachtung deutlich. CPLDs weisen einen geringeren komplexen Aufbau auf als FPGAs, was die Signallaufzeiten mit geringerem Aufwand vorhersagbarer macht. Die Erweiterung der Funktionalität durch Peripherie-Schnittstellen ist einfach umzusetzen, da diese bei Bedarf als Logische-Schaltung implementiert werden können.
Die Verwendung von FPGAs ist besonders bei einer parallelen und Durchlaufzeit kritischen Datenverarbeitung vorteilhafter. In Kombination mit einem Mikroprozessorsystem wird die erhöhte Rechenleistung des FPGAs durch die Möglichkeit der Berechnung komplexer Algorithmen auf dem Mikrocontroller ergänzt. Zusätzlich erweitern Peripheriefunktionen wie etwa USB, I2C oder SPI die Applikationsmöglichkeiten. FPGAs sind

deutlich komplexer Aufgebaut als CPLDs, bieten dafür aber mehr Ressourcen zur Implementierung von logischen Schaltungen. Durch den Aufbau sind die Signallaufzeiten schlechter vorhersagbar, was bei komplexeren Schaltungen ggf. eine Timing-Analyse erfordert. Der Strombedarf ist in der Regel größer als bei CPLDs. Die vorhandene Peripherie kann in einem FPGA Bestandteil der Hardware sein oder selbst entworfen werden. Ein Nachteil ist, dass die Funktionalität erst nach dem Einschaltvorgang geladen werden muss. Von allen betrachteten Hardware-Komponenten weisen FPGAs jedoch die größte Flexibilität bezüglich der Einsatzmöglichkeiten auf.

Tabelle 4.4: Ausgewählte Eigenschaften zum Vergleich von Mikrocontrollern, CPLDs und FPGAs

Eigenschaft	**Mikrocontroller**	**CPLD**	**FPGA**
Funktionsprinzip	sequenziell	parallel	parallel
Stromverbrauch	gering bis mittel	gering	mittel bis hoch
Signallaufzeiten	k. A.	einfach vorhersagbar	gering vorhersagbar
Peripherie	starr	flexibel	starr / flexibel
Stückpreis	gering bis hoch	gering bis mittel	mittel bis hoch

Im Folgenden wird ein Abgleich der festgelegten Spezifikationen mit den Eigenschaften der unterschiedlichen Hardware-Plattformen durchgeführt. Besonders die Anforderung N2 (Tabelle 4.2), einer hohen Flexibilität von Hard- und Software in der Entwicklungsphase, ist in diesem Zusammenhang dominant. Diese reduziert das Risiko eines nötigen Austauschs der Ziel-Hardware bei einer Spezifikationsänderung/-erweiterungen in Folgeprojekten und erleichtert den Entwicklern die Umsetzung von Prototyp-Spezifischen Implementierungen in der Entwicklungsphase.
Ein CPLD kann diese Anforderung nur bedingt erfüllen. Die geringe Ressourcen-Kapazität für das zu entwerfende System erfordert eine sehr effiziente und effektive Implementierung der Hardware und lässt somit kaum zusätzliche Erweiterungen zu.
Einen deutlicheren Vorteil hätte die Verwendung eines weiteren Mikrocontrollers. Diese sind wie beschrieben kostengünstig und ermöglichen unter anderem eine Implementierung des Systems in der Programmiersprache C. Allerdings wird auch hierbei die Anforderung N2 nicht vollständig erfüllt, da die Flexibilität der Hardware eingeschränkt ist. Zwar könnte man einen Mikrocontroller mit einer großen Anzahl an Peripherie-Komponenten verwenden, was allerdings den Stückpreis steigen lässt und möglicherweise zu einer Viel-

zahl von ungenutzten Ressourcen führt.
Die höchste Flexibilität bei der Implementierung von Soft- und Hardware-Komponenten bietet ein FPGA unter Nutzung eines Soft-Core-Mikroprozessors. Hierbei wird die Spezifikation N2 erfüllt und alle beteiligten Personengruppen gemäß Tabelle 4.1 können in das Projekt eingebunden werden. Die Programmierung von komplexen Berechnungen oder die Ansteuerung der Peripherieblöcke sowie die Speicherverwaltung des Systems kann hierbei durch den Soft-Core-Mikroprozessor übernommen werden. Zusätzlich besteht die Möglichkeit, benötigte Hardware-Komponenten selbst mittels VHDL zu entwerfen und diese in die Hardware-Architektur des System-on-Chip einzubinden. Hieraus folgt die Entscheidung, ein FPGA als Hardware-Plattform für den Coprozessor zu verwenden.

4.2.3 Konzept der Coprozessor-Architektur

Aus den festgelegten Anforderungen und vorherigen Diskussionen ergibt sich das Konzept des Coprozessor-Systems, welches die geforderten Spezifikationen erfüllt. Abbildung 4.2 zeigt das Konzept der System-Architektur. Als zentrale Regel- und Steuereinheit bleibt weiterhin der Arduino Nano erhalten, welcher als Master in das System eingebunden ist. Dieser hat über einen I2C-Schnittstelle die Möglichkeit, den Co-Prozessor MicroBlaze zu konfigurieren, sowie wesentliche Steuer- und Sensorinformationen abzufragen/zu übermitteln. Der FPGA übernimmt hierbei vollständig die Verarbeitung und Generierung der PWM-Signale sowie die Multi-Sensor-Signalverarbeitung. Neben den Vorteilen, weist das Konzept für allerdings auch Nachteile auf, welche nicht vernachlässigt werden können. Die I2C-Schnittstelle limitiert, durch die maximale Datenübertragungsrate, die Anzahl an Datenpaketen pro Zeiteinheit zwischen den beiden Chips. Zusätzlich steigt die Komplexität des zu entwerfenden Systems bei der Verwendung eines Soft-Core-Prozessors, was sich negativ auf die Fehleranfälligkeit und Wartungsfreundlichkeit auswirkt.

4.3 Entwicklung des Demonstrator-Systems

Im folgenden Abschnitt wird der Entwurf des Demonstrator-Systems zur Multi-Sensor-Signalverarbeitung erläutert. Anschließend werden die hierfür benötigten Hardware-Komponenten vorgestellt. Die Auswahl dieser orientiert sich an den festgelegten Spezifikationen aus der Anforderungsanalyse für die vollständige Umsetzung der manuellen und

autonomen Fahrzeugsteuerung. Dies vermeidet einen eventuell notwendigen Hardware-Austausch in Folgeprojekten.

4.3.1 Entwurf des Demonstrator-Systems

Abbildung 4.3 zeigt das Blockschaltbild des Demonstrator-Systems zur Multi-Sensor-Signalverarbeitung. Das Design ermöglicht die Ansteuerung und anschließende Signalverarbeitung für sechs Ultraschall-Sensoren, was gemäß Spezifikation N6 die maximale Anzahl an Sensoren abbildet. Die Spannungsversorgung des Coprozessor-Systems erfolgt

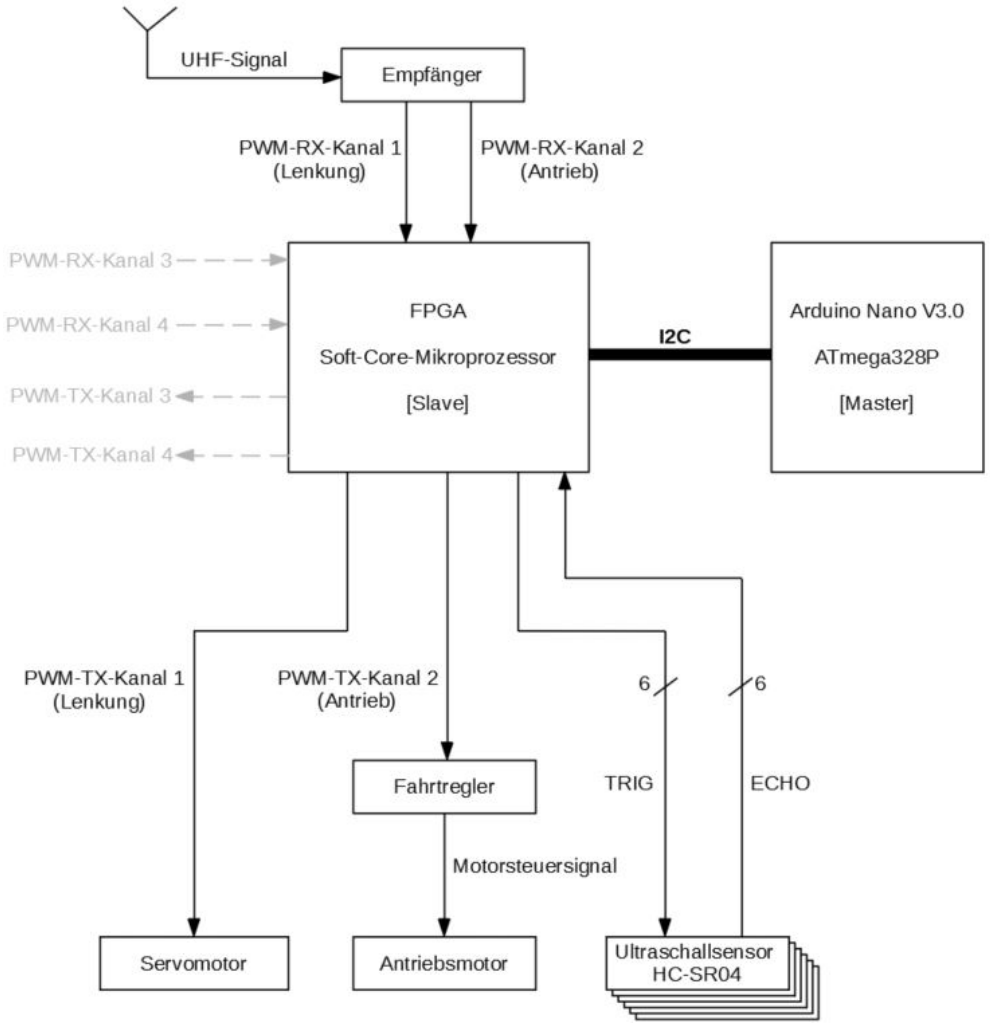

Abbildung 4.2: Konzept des Coprozessor-Systems

direkt über den NiMH-Akkupack. Die benötigte 5 V-Bordnetzdomäne für das FPGA-Modul, die Level-Shifter und Ultraschallsensoren, wird durch ein Spannungsregler-Modul erzeugt. Die Verwendung dieser Baugruppe ersetzt somit die Notwendigkeit einer zusätzlichen Spannungsversorgung über den Fahrtregler. Zudem erleichtert dieser Aufbau den Ein- und Ausbau des Demonstrator-Systems in das RC-Automodell, da der Verkabelungsaufwand geringer ausfällt. Der Arduino Nano kann direkt mit der Batterie-Spannung von $7,2$ V betrieben werden und stellt die $3,3$ V-Bordnetzdomäne für die Level-Shifter bereit. Der Datenaustausch zwischen dem Mikrocontroller ATmega328P und dem FPGA erfolgt, wie im Konzept beschrieben, über eine I2C-Schnittstelle. Da beide Komponenten auf unterschiedlichen Logik-Pegeln arbeiten, ist die Verwendung von Level-Shifter-Modulen zur Ansteuerung der Sensoren und I2C-Datenübertragung notwendig.

4.3.2 FPGA-Modul Cmod A7-35T

Für die Realisierung des Coprozessors wird das FPGA-Modul Cmod A7-35T [9] der Firma Digilent verwendet. Das Modul ist so konzipiert, dass es optimal für die Implementierung von Soft-Core-Prozessoren ausgelegt ist. Das FPGA befindet sich hierbei auf einem 48-DIP Formfaktor-Board, welches das Einsetzten in eine Steckplatine ermöglicht. Abbildung 4.4 zeigt das verwendete FPGA-Modul.
Als FPGA wird ein Artix-7 [29] der Firma Xilinx verwendet. Dieses wird mit einer Versorgungsspannung von $3,3$ V betrieben und ermöglicht durch das Vorhandensein ausreichender Hardware-Ressourcen, die sichere Umsetzung aller festgelegten Anforderungen an den Coprozessor für die manuelle und autonome Fahrzeugsteuerung. Die Block-RAM Kapazität beträgt 225 kB. Neben den technischen Voraussetzungen ergibt sich die Entscheidung für ein FPGA der Firma Xilinx aus zwei weiteren Aspekten. Zum Einen ist die benötigte Software (Vivado Design Suite und Xilinx SDK) für den Hardware-Entwurf aus vorherigen Projekten bekannt, zum Anderen stellt die Firma Xilinx eine umfassende Hard- und Software-Dokumentation zur Verfügung, welche die schnelle Umsetzung von Projekten erleichtert. Auf dem PCB befinden sich zum Betrieb des FPGAs zusätzliche Peripherie-Bausteine, wie ein 4 MB QSPI-Flash-Speicher, ein 512 kB SRAM-Speicher, ein externer 12 MHz Oszillator, sowie eine USB-UART/USB-JTAG-Schnittstelle. Über Letztere erfolgt die Programmierung des FPGAs und die Ausgabemöglichkeit einer Zeichenkette in der SDK-Console. Zur optischen Anzeige verfügt das Modul über mehrere LEDs. Zudem sind zwei konfigurierbaren Taster vorhanden. Tabelle 4.5 listet die Kenndaten des Moduls auf.

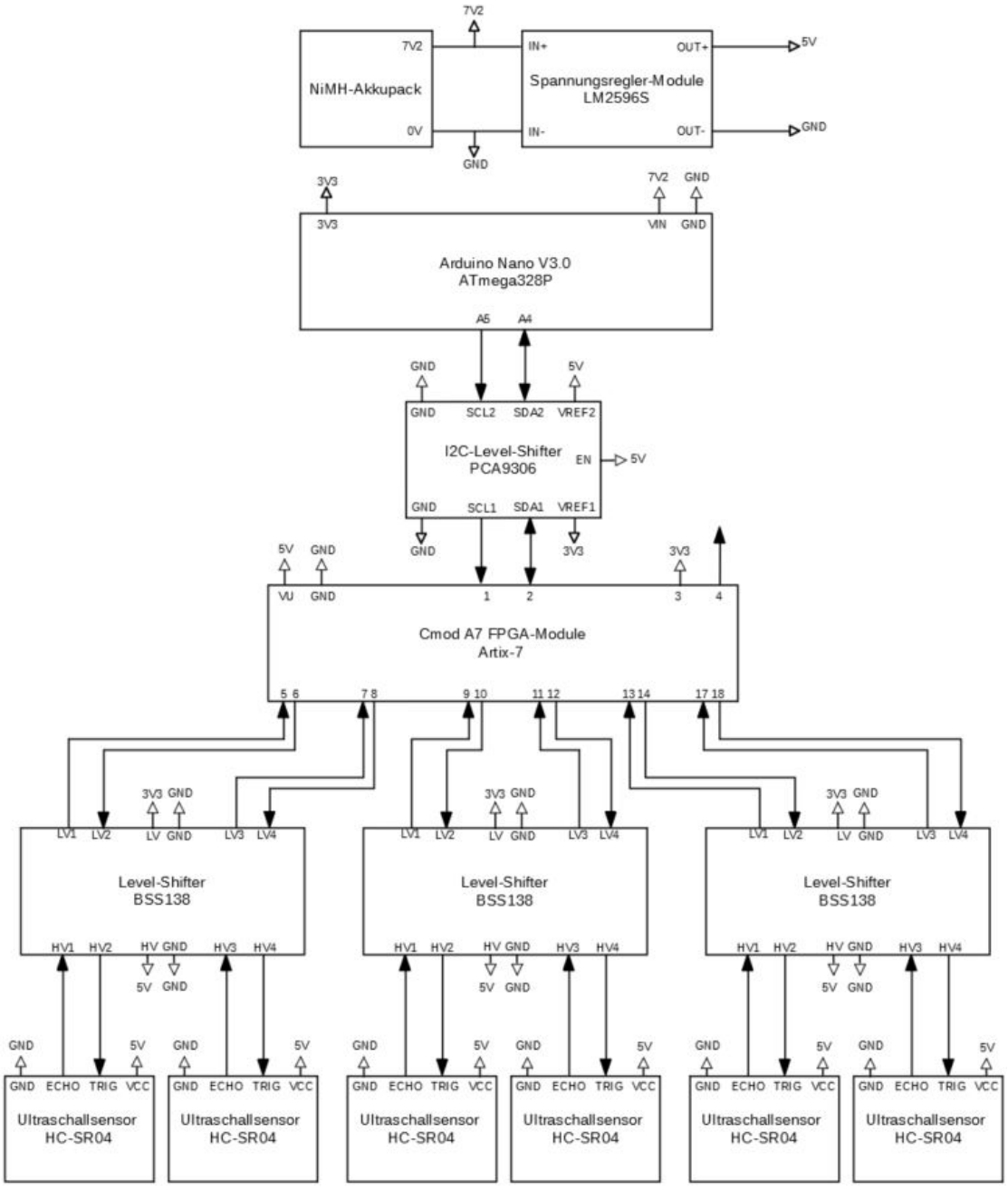

Abbildung 4.3: Blockschaltbild des Demonstrator-Systems zur Multi-Sensor-Signalverarbeitung

Das FPGA-Modul zeichnet sich durch die geringen Abmessungen der Platine aus. Dies ist bei einem späteren Einbau in das RC-Modell von Vorteil ist. Zudem sind kaum unge-

Tabelle 4.5: Technische Daten des FPGA-Moduls Cmod A7-35T [9]

Eigenschaft	**Wert**
FPGA	Artix-7 (XC7A35T-1CPG236C)
LUTs	20800
Flip-Flops	41600
Block RAM	225 kB
DSP Slices	90
Clock Management Tiles	5
ADC	1 MSPS
FPGA Betriebsspannung	3,3 V
Oszillator (extern)	12 MHz
SRAM (extern)	512 kB
Quad-SPI-Flash (extern)	4 MB
Digital I/O	44
Analog I/O	2
Digital I/O Spannung	3,3 V
Betriebsspannung (extern)	3,3 - 5,5 V
Abmessungen PCB	1,778 cm x 6,985 cm
Programmierung über	QSPI-Flash / JTAG
Sonstiges	USB-UART Bridge USB-JTAG Bridge Pmod-Schnittstelle 2 LED 1 RGB-LED 2 Taster

nutzte Ressourcen und Schnittstellen nach der vollständigen Umsetzung des Coprozessor-Systems vorhanden. Ein Nachteil ergibt sich allerdings aus der Verwendung eines SRAM-Basierten FPGAs. Das Programm muss nach jedem Einschalten neu in das FPGA geladen werden, was eine Zeitverzögerung zwischen Einschaltvorgang und Herstellung der Funktionalität bewirkt. Zudem ist wie beschrieben, aufgrund der unterschiedlichen Logik-Pegel, die Verwendung von Level-Shifter-Modulen erforderlich. Diese stellen im Gesamt-

system elektrische Verbraucher dar und steigern die Leistungsaufnahme des Coprozessor-Systems.

4.3.3 Level-Shifter-Modul PCA9306 und BSS138

Durch die Verwendung des Level-Shifter-Moduls PCA9306 der Firma SparkFun wird sichergestellt, dass ungeachtet einer Pegelanpassung, die Übertragungsstrecke innerhalb der I2C-Bus-Spezifikationen betrieben wird. Abbildung 4.5 zeigt das Modul. Als Wandler wird ein IC vom Type PCA9306 [19] der Firma Texas Instruments verwendet, welches speziell für die Anwendung bei Inter-Chip Bus-Systemen, wie I2C oder SMBus, ausgelegt ist. Die Pegel-Konvertierung erfolgt hierbei dual-bidirektional. Die zulässigen Taktraten hängen von der Anwendungsvariante des Bus-Systems ab. Grundsätzlich wird bei I2C der Standard-Mode (100 kHz) und der Fast-Mode (400 kHz) durch die Hardware abgedeckt. Unter bestimmten Applikationsbedingungen werden Taktraten größer 100 MHz unterstützt.
Für die Ansteuerung der Ultraschallsensoren wird ein Level-Shifter-Module von Typ BSS138, ebenfalls von der Firma SparkFun, verwendet. Abbildung 4.6 zeigt das Modul. Dieses ist jedoch wesentlich einfacher aufgebaut als der zuvor beschriebene Pegelwandler. Je Platine sind vier N-Kanal Feldeffekttransistoren [11] verbaut, welche die Pegel-Konvertierung von vier bidirektionalen Kanälen ermöglichen.

4.3.4 Spannungsregler-Modul LM2596S

Das Spannungsregler-Modul LM2596S [7] stellt die 5 V Spannungs-Domäne für das Demonstrator-System bereit. Abbildung 4.7 zeigt das Modul. Der Abwärtswandler ist so abgeglichen, das dieser die 7,2 V Batteriespannung auf 5 V regelt, wobei der maximale Ausgangsstrom ohne Kühlkörper bei 2 A liegt. Die Schaltfrequenz der unsynchronen Gleichrichtung liegt bei 150 kHz. Dies ist für den Betrieb des FPGAs, der Level-Shifter, sowie der angeschlossenen Ultraschallsensoren ausreichend. Mit steigenden Implementierungsstufen des Coprozessor-Systems kann sich der zukünftige Strombedarf erhöhen, wodurch einen Austausch der Komponente erforderlich ist.

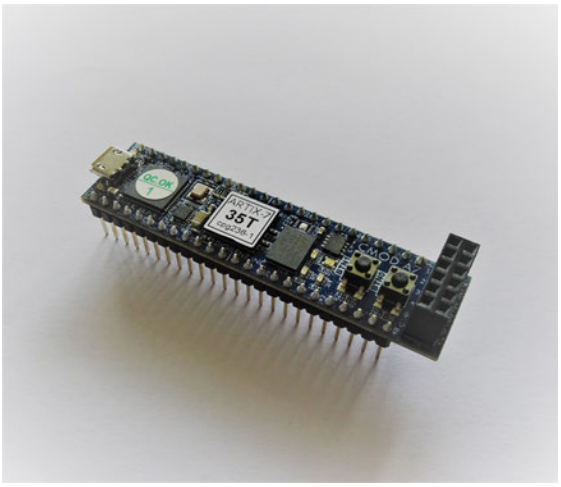

Abbildung 4.4: Abbildung des Artix-7 FPGA-Modul Cmod A7-35T

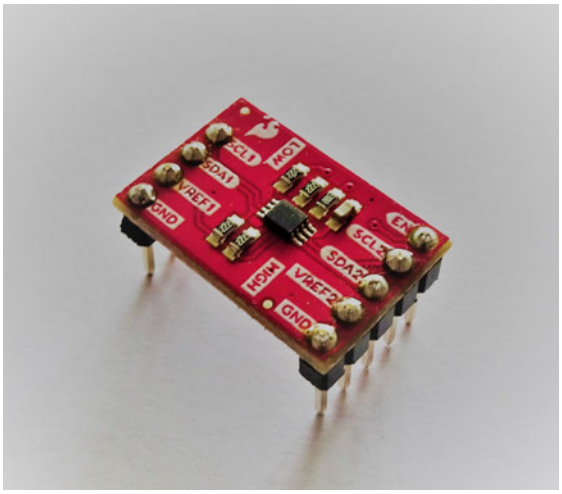

Abbildung 4.5: Abbildung des Level-Shifter-Moduls PCA9306

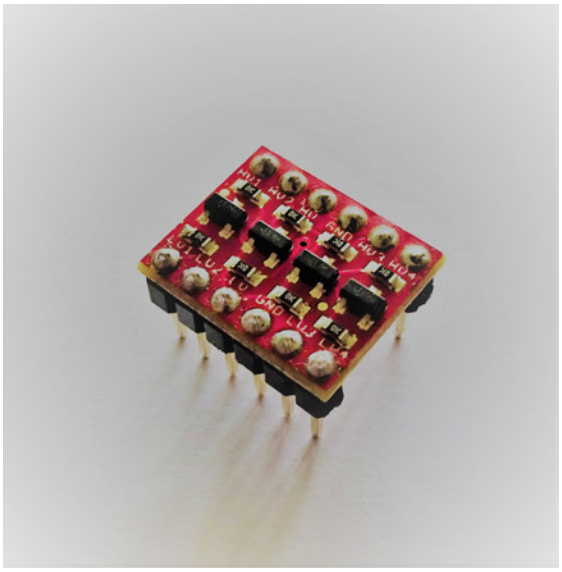

Abbildung 4.6: Abbildung des Level-Shifter-Moduls BSS138

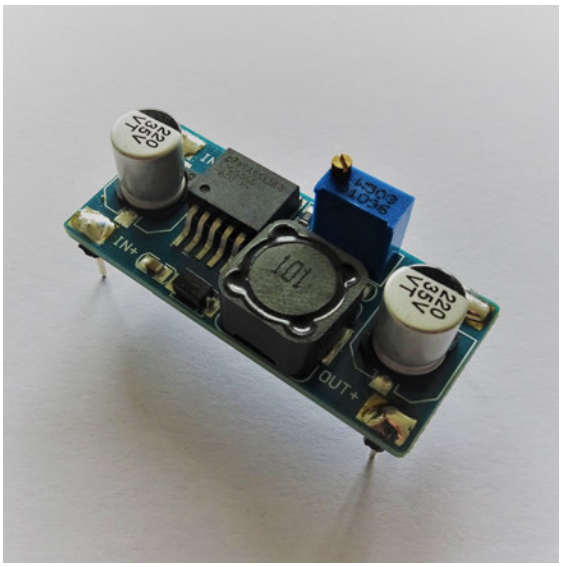

Abbildung 4.7: Abbildung des Spannungsregler-Moduls LM2596S

5 Design und Implementierung des System-on-Chip

In diesem Kapitel wird das Design des Coprozessors erarbeitet und die Implementierung des System-on-Chip auf dem FPGA durchgeführt. Dafür wird zunächst die erforderliche Software-Architektur des System-on-Chip zur I2C-Datenübertragung und Ausführung von Speicherzugriffsroutinen entworfen. Anknüpfend daran erfolgt eine Definition der Hardware-Architektur des Soft-Cores-Prozessors und dessen Peripherie-Komponenten. Schwerpunkt liegt hierbei auf der Konzeption des IP-Funktionsblocks zur Multi-Sensor-Signalverarbeitung. Abschließend erfolgt die Implementierung und Konfiguration des System-on-Chip.

5.1 Software-Design des System-on-Chip

Das Software-Design soll die I2C-Datenübertragung sicherstellen und ein Beschreiben bzw. Auslesen der System-on-Chip internen Register erlauben. Diese Speicherzugriffsroutinen ermöglichen eine Konfiguration und Steuerung der Hardware-Komponenten Coprozessors, sowie das Abfragen von Sensorwerten durch den I2C-Master. Hierzu findet eine Definition des I2C-Paket-Layouts und eine Typisierung der AXI4-Slave-Register der zu entwerfenden IP-Funktionsblöcke statt.

5.1.1 Software-Architektur des System-on-Chip

Aus den festgelegten Spezifikationen und dem Konzept der Coprozessor-Architektur ergeben sich die folgenden Funktionalitäten, welche die Software des System-on-Chip umsetzten muss:

- Initialisierung des Mikroprozessors und weiterer erforderlicher Schnittstellen

- Auslesen des I2C-Empfangsregisters
- Auswertung der empfangenen I2C-Pakete
- Aufbau des I2C-Paket-Layouts
- Beschreiben des I2C-Senderegisters
- Ausführung von Speicherzugriffsroutinen auf AXI4-Lite-Register nach Vorgabe des I2C-Masters

Abbildung 5.1 zeigt das hieraus abgeleitete Flussdiagramm welches die Hauptfunktion zur Umsetzung der I2C-Kommunikation zeigt. Hierbei ist die Implementierung einer Busy-Waiting-Loop, mit zugehöriger Interrupt-Service-Routine, vorgesehen. Dieses Ereignisgesteuerte Design erfordert keine spezielle Hardware und ist eine typische softwarebasierte Methode beim Entwurf eingebetteter Systeme. Wie in der Abbildung zu sehen, erfolgt nach dem Programmstart zunächst die Initialisierung der UART-Schnittstelle, Cache-Zugriffsspeicher und des GPIO-Moduls. Anschließend erfolgt die Initialisierung der I2C-Schnittstelle. Hierbei wird das I2C-Hardware-Modul mit der festgelegten Slave-Adresse konfiguriert, sowie die Funktionalität eines Hardware-Interrupts durch dieses hergestellt. Nachfolgend findet eine Initialisierung des Interrupt-Systems statt. Nach der Freigabe von globalen Interrupts wird eine while(1)-Schleife betreten, die Busy-Waiting-Loop. Erfolgt eine Lese- oder Schreibanfrage durch den I2C-Master, wird ein Interrupt-Request (IRQ) ausgelöst und eine Interrupt-Service-Routine (ISR) aufgerufen, welche den eigentlichen funktionalen Programmteil beinhaltet. Erfolgt kein IRQ durch die I2C-Hardware-Komponente, befindet sich das Programm kontinuierlich in der Busy-Waiting-Loop.
Abbildung 5.2 zeigt das Flussdiagramm der Interrupt-Service-Routine. Wie zusehen erfolgt zunächst eine Prüfung, ob es sich um eine Schreibanfrage des I2C-Masters handelt. Ist dieses der Fall, sollen dementsprechend Daten vom Master an der Slave gesendet werden. Fällt die Abfrage negativ aus, handelt es sich um eine Leseanfrage, bei dem die Datenübertragung in umgekehrter Richtung erfolgt. Bei der Schreibanfrage wird zunächst eine Rx-LED, welche sich auf dem FPGA-Board befindet, eingeschaltet. Wenn alle Datenpakete des Masters erfolgreich empfangen sind, wird das I2C-Empfangsregister ausgelesen. In Abhängigkeit vom Paketinhalt werden die Nutzdaten einem AXI4-Slave-Register eines bestimmten IP-Funktionsblocks zugewiesen. Anschließend wird die Rx-LED ausgeschaltet und die Interrupt-Service-Routine verlassen. Bei einer Leseanfrage des Masters wird eine Tx-LED eingeschaltet und analog zum oben genannten Ablauf ein AXI4-Slave-Register ausgelesen. Anschließend erfolgt der Aufbau des I2C-Paket-Layouts

und die Daten werden an den Master übertragen. Danach wird die Tx-LED ausgeschaltet und die Interrupt-Service-Routine verlassen.

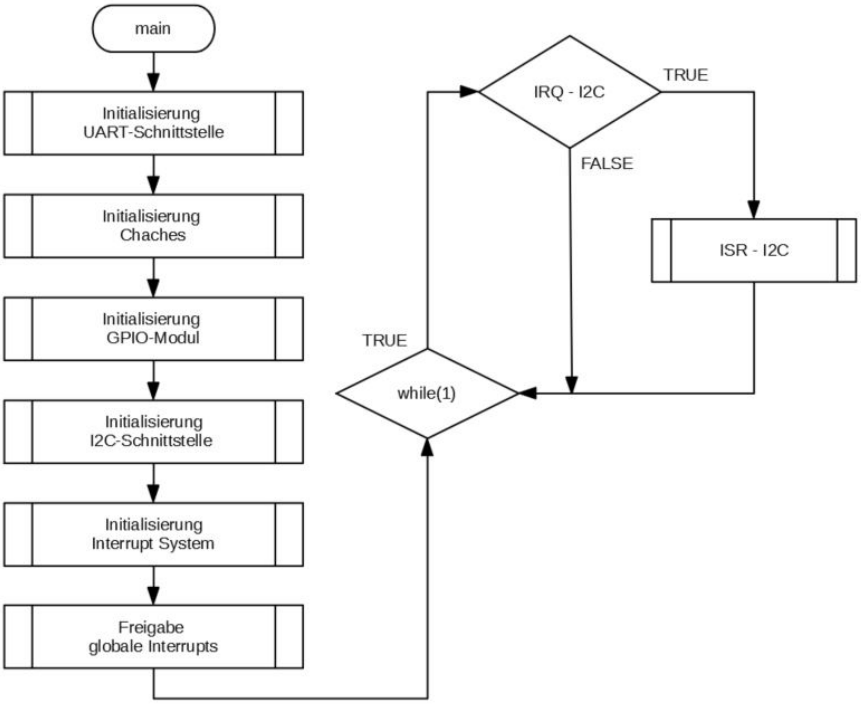

Abbildung 5.1: Flussdiagramm der Hauptfunktion zur Initialisierung des System-on-Chip

5.1.2 Definition des I2C-Paket-Layouts

Im Folgenden wir der Aufbau des I2C-Paket-Layouts definiert. Dieser soll einen Zugriff des I2C-Masters, auf die jeweiligen IP-Funktionsblöcke über eine AXI4-Lite-Schnittstelle ermöglichen. Da der Fokus dieser Arbeit nicht auf der Implementierung einer I2C-Schnittstelle liegt, wird als Grundlage eine Fremdsoftware [28] der Firma Xilinx genutzt. Diese wird um weitere Software-Funktionalitäten ergänzt. Außerdem ist eine statische Anzahl von sechs Byte pro Frame vorgesehen, um die Komplexität der Software-Architektur gering zu halten und in der erstem Implementierungsphase bessere Debug-Möglichkeiten anwenden zu können. Dafür sind zwei unterschiedliche Daten-Frame-Layouts definiert. Abbildung 5.3 zeigt den Aufbau der verwendeten Pakete ohne den I2C-Frame-Header, welcher aus dem Start-Bit,der Slave-Adresse, dem W/R-Bit und ACK-Bit besteht.

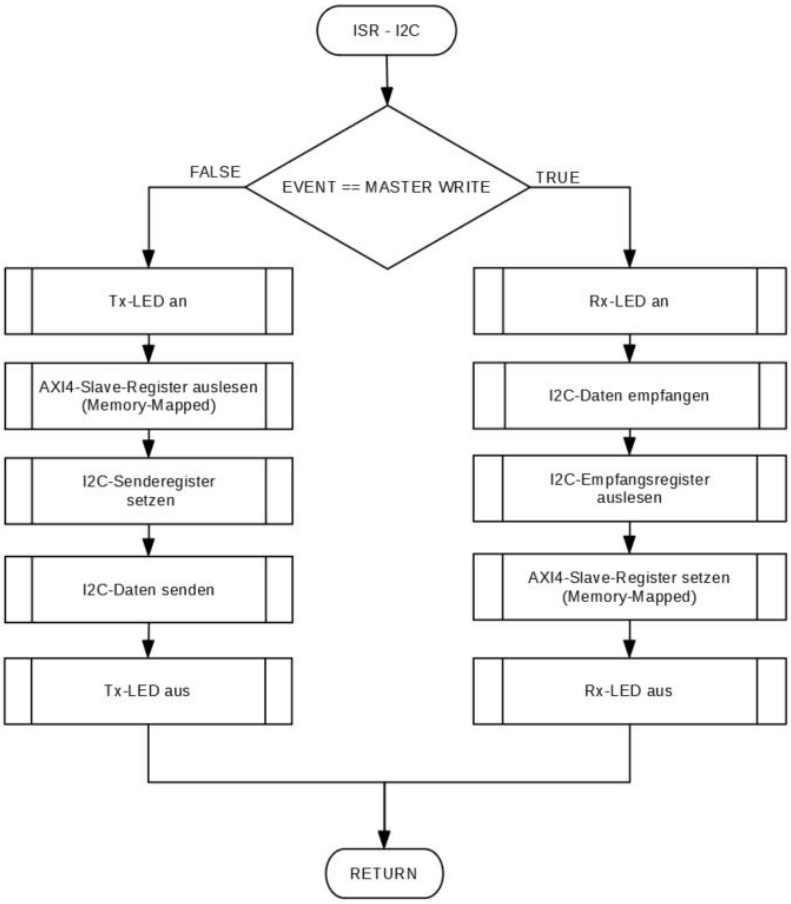

Abbildung 5.2: Flussdiagramm der Interrupt-Service-Routine zur I2C-Kommunikation und Ausführung von Speicherzugriffsroutinen des System-on-Chip

Zum Auslesen und Beschreiben der Slave-Register des Coprozessors durch den Arduino Nano wird der I2C-RX/TX Aufbau genutzt. Die Register-Auswahl des Coprozessors erfolgt über den I2C-Select Paket-Aufbau. Beim I2C-TX/RX-Layout wird durch Byte 0 die IP-Modul-Adresse des Slaves und der Befehls-Typ der I2C-Übertragung festgelegt. Der Befehls-Typ ist zu Debug-Zwecken implementiert worden. Byte 1 definiert das angesprochene Register des Moduls und besitzt einen Reserved-Bereich, welcher für nachfolgende Implementierungen freigehalten ist. Die nachfolgenden vier Byte beinhalten die Nutzdaten. Beim I2C-Select-Layout beinhaltet Byte 0 eine Select-ID. Anhand dieser erkennt der Slave, dass es sich um eine Register-Auswahl durch den Master handelt. Byte 1 und Byte 2 definieren, welcher Registerwert eines bestimmten IP-Moduls in den Schreib-Speicher

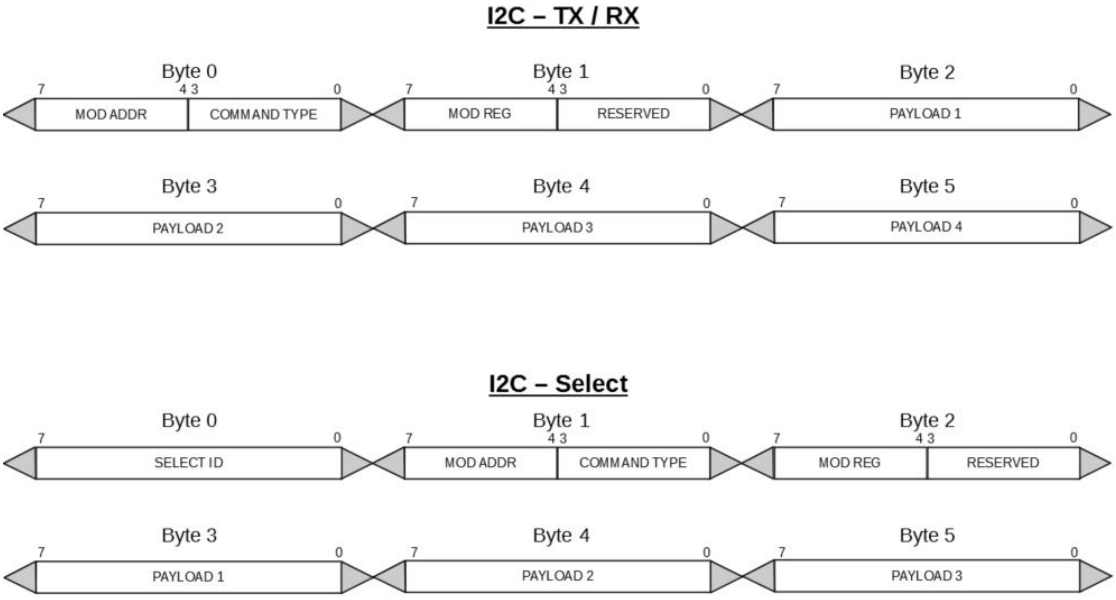

Abbildung 5.3: Aufbau des I2C-Paket-Layouts (Darstellung ohne Frame-Header)

geladen werden soll. Die nachfolgenden drei Byte beinhalten keine Funktion. Tabelle 5.3 listet die gültigen Hexadezimal-Werte zur Datenübertragung.

5.1.3 Typisierung der AXI4-Lite-Slave-Register

Im vorherigen Abschnitt ist eine Definition unterschiedlicher Befehls-Typen der übertragenen I2C-Daten-Pakete erfolgt. Analog dazu wird eine Typisierung der AXI4-Slave-Register vorgenommen. Dies erleichtert die schnelle Zuordnung des Register-Inhalts und unterstützt die Erstellung von Dokumentationsunterlagen. Tabelle 5.2 listet die festgelegten Register-Bezeichnungen sowie die vorgesehene Anwendungsfunktion.

5.2 Hardware-Design des System-on-Chip

In diesem Abschnitt wird das Hardware-Design des System-on-Chip entworfen. Ausgehend von den Spezifikationen und der Anforderungsanalyse wird zunächst die benötigte Hardware-Architektur für eine vollständige Umsetzung der manuellen und autonomen Fahrzeugsteuerung skizziert. Anschließend erfolgt die Eingrenzung der zu implementierenden Komponenten, welche für die Umsetzung der Multi-Sensor-Signalverarbeitung

Tabelle 5.1: Definition des I2C-Paket-Layouts

Bitfeld-Name	**Bedeutung**	**Inhalt**	**Bit-Anzahl**
MOD ADDR	IP-Modul Adresse	0xA - Sensor-Modul 0xB - PWM-RX-Modul 0xC - PWM-TX-Modu 0xD - Gateway-Modul	4
COMMAND TYPE	Befehls-Typ (Debug)	0x0 - Select 0x1 - Konfiguration 0x2 - Status 0x3 - Steuerung 0x4 - Daten	4
MOD REG	Module-Register	0x0 - Register 0 0x1 - Register 1 0x2 - Register 2 0x3 - Register 3 0x4 - Register 4 0x5 - Register 5 0x6 - Register 6 0x7 - Register 7	4
PAYLOAD	Nutzdaten	0x000000 bei I2C-Select Nutzdaten bei I2C-TX/RX	24 32
SELECT ID	Select-ID	0x00	8
RESERVED	Reserviert	0x00	4

Tabelle 5.2: Typisierung der AXI4-Slave-Register

Register-Bezeichnung	**Anwendungsfunktion**
CONFIG	Register konfigurieren das AXI4-Slave-Modul (z.B. vorgegebene Zählerwerte)
STATUS	Zustandsregister des AXI4-Slave-Moduls (z.B. Modul aktiv)
DATA	Datenregister des AXI4-Slave-Moduls (z.B. Sensorwerte, Steuerwerte)

erforderlich sind. Hierbei liegt der Schwerpunkt auf dem Design der zu entwerfenden AXI-Hardware-Module, welche nicht schon als IP-Funktionsblock von der Entwicklungsumgebung zur Verfügung gestellt werden. Ergänzend wird die Definition der erforderlichen Hardware-Architektur des Soft-Core-Prozessors durchgeführt.

5.2.1 Hardware-Architektur des System-on-Chip

Abbildung 5.4 zeigt den Aufbau der Hardware-Architektur des System-on-Chip. Das Design besteht sowohl aus bereits existierenden, als auch aus selbst zu entwerfenden IP-Funktionsblöcken. Die Entscheidung für einen modularen Aufbau basierend auf IP-Cores, bietet den Vorteil einer geringeren Implementierungszeit gegenüber eines vollständig selbst zu entwerfenden Systems gleicher Leistung. Ein weiterer positiver Aspekt ist die Möglichkeit über eine grafische Oberfläche in der Entwicklungssoftware grundlegende Systemanpassungen vorzunehmen.
Die Hardware-Architektur besteht aus dem Soft-Prozessor-Core und aus zehn Peripherie-Modulen. Die Interaktion zwischen dem Mikroprozessor und den Modulen erfolgt über den On-Chip-Bus AXI4-Lite. Die UART-Schnittstelle ermöglicht eine serielle Datenübertragung zur UART/USB-Bridge, welche insbesondere beim Software-Debugging genutzt wird. Die Kommunikation mit dem Mikrocontroller ATmega328P erfolgt über eine I2C-Schnittstelle. Die Anbindung des externen Flash-Speichers, zum Laden der FPGA-Konfiguration und des ausführbaren Programms, wird über eine QSPI-Schnittstelle ermöglicht. Die optionale Verwendung des externen SRAM-Speichers, auf den Teile des ausführbaren Programms geladen werden können, erfordert einen External Memory Controller (EMC). Die Anbindung des externen SRAM-Speichers dient zur Erweiterung des FPGA internen Block-RAMs. Dieses ermöglicht in Folgeprojekten die Implementierung größerer Software-Architekturen. Die Ansteuerung von zwei LEDs, welche zur optischen Anzeige der Abfrageart des I2C-Masters dienen, erfolgt über eine AXI-GPIO-Schnittstelle. Alle vorangehend genannten Komponenten stehen als existierende IP-Funktionsblöcke zur Verfügung. Die nachfolgend beschriebenen Hardware-Module sind im Rahmen der Arbeit zu entwerfen.
Durch ein PWM-Rx-Modul wird das Auslesen der Empfängersignale ermöglicht. Dafür wird eine Pulsbreitenbestimmung der eingehenden PWM-Signale durchgeführt, wobei der ermittelte Zeitwert anschließend als Zählerwert je Kanal vorliegt. Unabhängig von der gewählten Steuerungsart des Fahrzeuges erfolgt eine Weitergabe dieser Zählerwerte an das PWM-Gateway-Modul. Dieses setzt zwei geforderte Spezifikationen um. Zum Einen die

wahlfreie Verschaltung der PWM-Kanäle, welches durch ein Vertauschen der Zählerwert-Zuordnung ermöglicht wird. Zum Anderen erfolgt, abhängig von der Steuerungsart des Fahrzeuges, eine Weiterleitung oder Unterbrechung der Pulsbreiten-Information an das PWM-Tx-Modul. Eine Trennung kann ebenfalls durch ein FPGA-internes Trigger-Signal des Fail-Save-Data-Monitoring-Moduls (FSDM) ausgelöst werden. Das PWM-Tx-Modul erzeugt, in Abhängigkeit der vorliegenden Zählerwerte, die PWM-Signale der angeschlossenen PWM-Aktoren. Bei der autonomen Fahrzeugsteuerung werden diese durch den I2C-Master vorgegeben, bei der manuellen Fahrzeugsteuerung hingegen über das PWM-TX-Modul. Zur Ansteuerung der Ultraschallsensoren wird ein Sensor-Modul verwendet. In diesem wird eine Multi-Sensor-Signalverarbeitung durchgeführt. Die Konfiguration und Steuerung des Moduls erfolgt durch den I2C-Master. Das FSDM-Modul weißt zwei geforderte Funktionalitäten auf. Erstens steuert es in Abhängigkeit der empfangenen oder gesendeten I2C-Paket-Art eine RGB-LED an. Zweitens überwacht es kontinuierlich das Vorhandensein der $3,3$ V Ausgangsspannung des Arduino Nano. Fällt diese auf GND-Potential, liegt mit hoher Wahrscheinlichkeit ein Ausfall des ATmega329P vor. Das Modul setzt darauf hin ein Trigger-Signal für das PWM-Gateway-Modul. Dadurch wird bei einer autonomen Fahrzeugsteuerung und einem gleichzeitig auftreten Ausfall des ATmega328P, die Aktivierung der manuellen Steuerung angestoßen. Eine Ausführlichere Beschreibung der letzten beiden Modulen erfolgt in den Abschnitten 5.2.2 und 5.2.3.
Abbildung 5.5 zeigt die in dieser Arbeit beabsichtigte Implementierungsstufe der Hardware-Architektur. Die Umsetzung des PWM-Rx-, PWM-Tx- und PWM-Gateway-Moduls erfolgt nicht. Das Trigger-Signal des FSDM-Moduls wird an einem GPIO-Pin ausgegeben und kann somit für weitere Messungen genutzt werden.

5.2.2 Hardware-Design des Fail-Save-Data-Monitoring Moduls

Das AXI-FSDM-Modul dient zur kontinuierlichen Überwachung der Spannungsversorgung des Mikroprozessors ATmega328P sowie der Ansteuerung einer RGB-LED in Abhängigkeit der gesendeten und empfangenen I2C-Paket-Art. Abbildung 5.6 zeigt das Blockschaltbild der Hardware-Komponente. Das Modul besitzt zur Anbindung an den Mikroprozessor eine AXI4-Lite-Schnittstelle sowie ein Eingangssignal für die $3,3$ V Ausgangsspannung der Arduino Nano Mikrocontroller-Plattform. Die Ansteuerung der RGB-LED erfolgt über drei Ausgangssignale. Die festgelegte Farbcodierung ist in Tabelle 5.3 gelistet. Das Modul besitzt ein AXI4-Lite-Slave-Register, in dem der Befehlstyp des I2C-Frames abgespeichert wird. Abbildung 5.7 zeigt die Register-Beschreibung. Hierbei han-

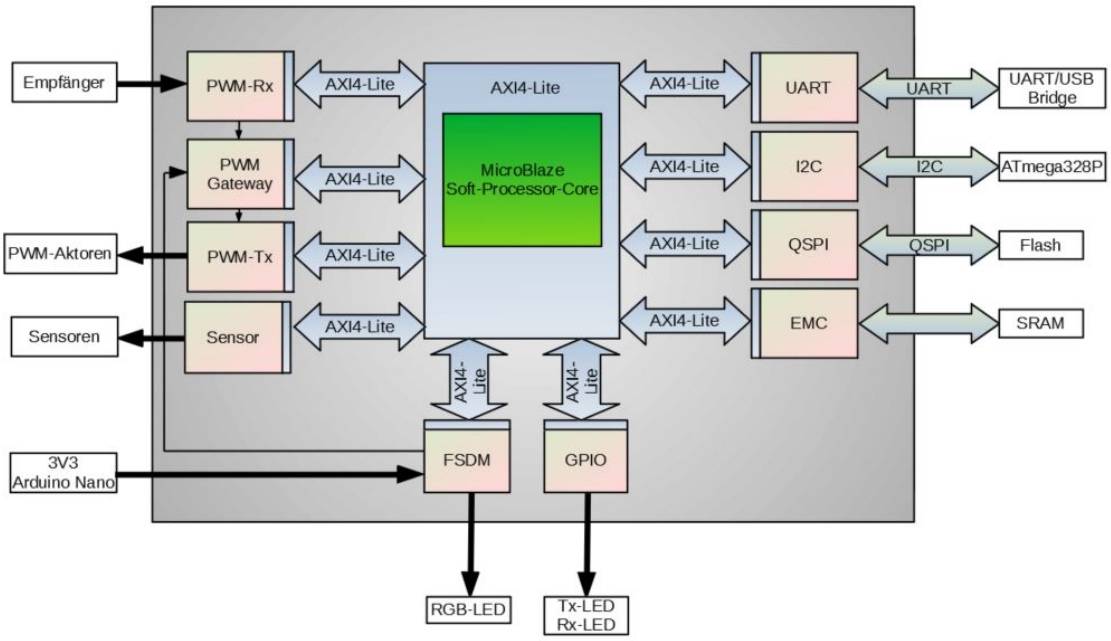

Abbildung 5.4: Hardware-Architektur des System-on-Chip für die manuelle und autonome Fahrzeugsteuerung

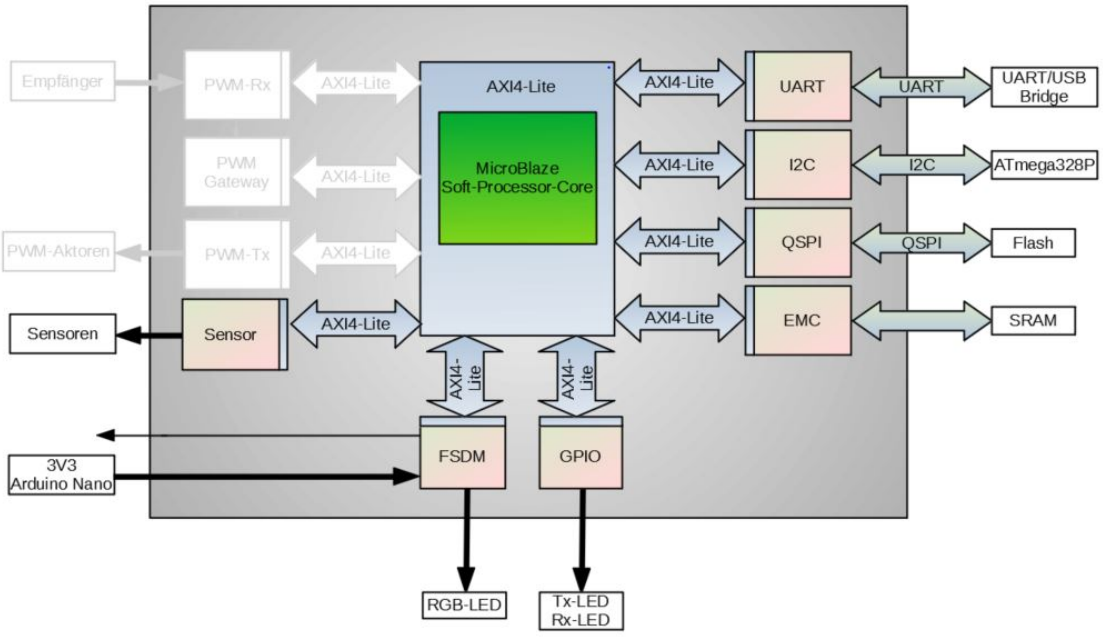

Abbildung 5.5: Hardware-Architektur des System-on-Chip zur Implementierung der Multi-Sensor-Signalverarbeitung

Tabelle 5.3: Farbcodierung der RGB-LED

I2C-Paket-Art	**LED-Farbe**
Select	aus
Konfiguration	weiß
Status	grün
Steuerung	hellblau
Daten	dunkelblau
Sonderfall	**LED-Farbe**
Fehlerfall Arduino Nano	rot

delt es sich um ein Konfigurations-Register des AXI-FSDM-Moduls.
Das Modul ist als Mealy-Automat entworfen und besitzt zwei Zustände (siehe Abbildung 5.8). Diese sind RUN und FAIL. Im Zustand RUN liegt kein Fehlerfall des ATmega238P vor und abhängig vom Registerinhalt wird die RGB-LED angesteuert. Der Zustandswechsel erfolgt bei erkennen eines Low-Pegel der Arduino Nano Ausgangsspannung. Im FAIL Zustand leuchtet die LED rot und das Fail-Flag-Ausgangssignal ist gesetzt. Dieser Zustand kann nur noch durch einen System-Reset verlassen werden.

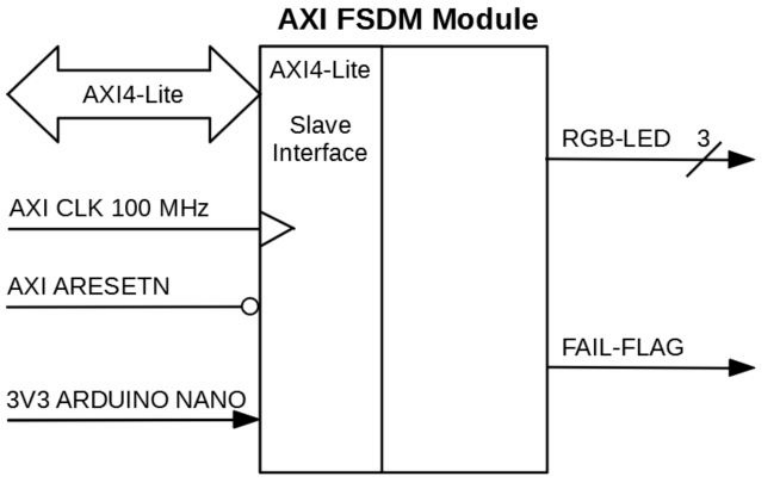

Abbildung 5.6: Blockschaltbild des Fail-Save-Data-Monitoring-Moduls

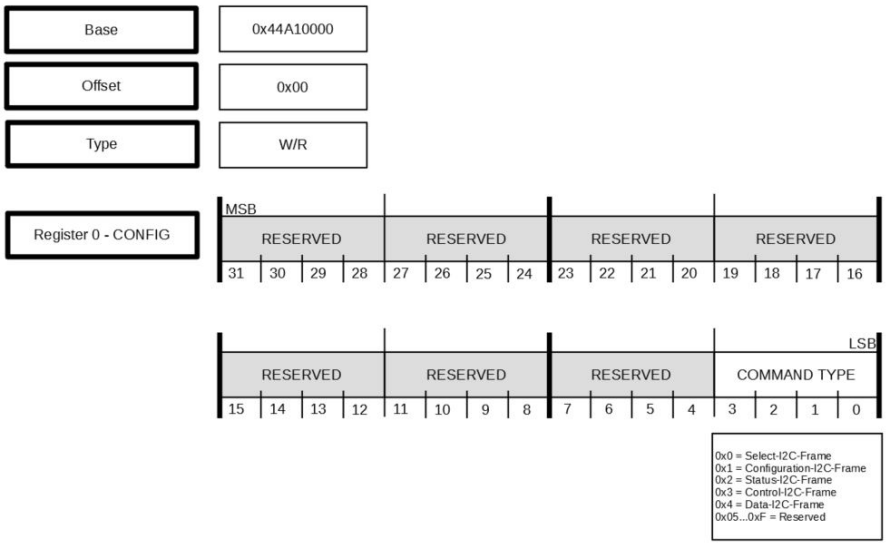

Abbildung 5.7: Register-Beschreibung FSDM-Modul (Register 0)

5.2.3 Hardware-Design der Multi-Sensor-Signalverarbeitung

Dieser Abschnitt befasst sich mit dem Hardware-Design des AXI-Sensor-Moduls. Hauptfunktion der Komponente ist die Ansteuerung der Sensoren nach Vorgabe des I2C-Masters und die Bereitstellung der ermittelten Entfernungsinformationen. Hierzu wird eine Multi-Sensor-Signalverarbeitung entworfen, welche die geforderten Spezifikationen erfüllt. Im Folgenden werden die Hardware-Architekturen der hierfür benötigten Einzel-Komponenten vorgestellt.

5.2.3.1 Hardware-Design des Sensor-IP-Cores

Abbildung 5.9 zeigt das Blockdiagramm des AXI-Sensor-Moduls. Der IP-Funktionsblock nutzt zwei Taktsignale, 100 MHz und 5,2 MHz. Zur Anbindung an den Soft-Prozessor-Core ist eine AXI4-Lite-Slave-Schnittstelle vorhanden. Diese verfügt über sechs Slave-Register, welche die Modul-Konfiguration und -Steuerung ermöglichen. Zusätzlich werden diese zur Speicherung der ermittelten Sensordaten verwendet. Die Hardware-Komponente besteht aus einer Steuereinheit, dem Sensor Array Module und mehreren Sensor Modulen. Für jeden angeschlossenem Ultraschallsensor wird ein Sensor Modul benötigt. Dieses übernimmt die Generierung des zum Start eines Messvorgangs nötigen Trigger-Impuls, sowie die anschließende Bestimmung der Entfernungsinformation aus dem eingehenden

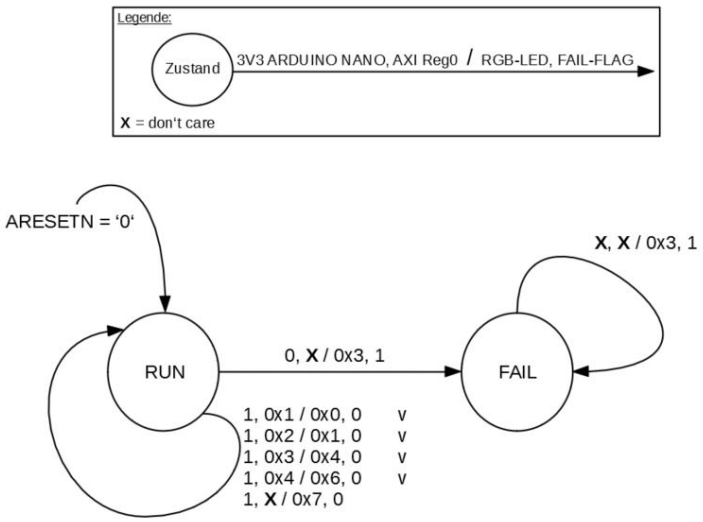

Abbildung 5.8: Zustandsdiagramm FSDM-Modul

Echo-Signal. Nach einem Messdurchlauf werden die Messergebnisse an das Sensor Array Modul übertragen. Das Verhalten dieser Komponente wird direkt über die AXI4-Slave-Registerwerte bestimmt. Die Abbildung 5.7 bis 5.15 zeigen die Register-Beschreibungen. Register 0-2 sind Daten-Register, welche die ermittelten Entfernungsinformationen enthalten. Diese liegen je Sensorwert als 16-Bit-Zahl vor. Register 3 und 4 sind Konfigurations-Register des AXI-Sensor-Moduls. Hierbei beinhaltet Register 3 die nach Spezifikation N5 geforderte Wiederholrate der Messdurchläufe. Über Register 4 erfolgt die Steuerung der Hardware-Komponente. Bit 0-5 ermöglicht die Auswahl der Ultraschallsensoren, welche bei jedem Messdurchlauf aktiviert sein sollen. Über Bit 8-13 wird die nach F12 geforderte Spezifikation eines One-Shot-Modus je Sensor-Modul de- bzw. aktiviert. Bit 16 und 17 werden bei einer Leseanfrage der Sensordaten durch den I2C-Master, für die Sicherstellung der Datenintegrität, benötigt. Bit 31 Startet bzw. Stoppt das AXI-Sensor-Modul. Register 5 ist ein Status-Register, wobei lediglich ein Bit genutzt wird. Dieses zeigt das Vorliegen neuer Sensordaten an, welche zuvor noch nicht durch den I2C-Master abgerufen wurden.
In den nachfolgenden Abschnitten wird der Aufbau und die Funktionsweise der internen Hardware-Komponenten erläutert und auf den Zusammenhang zwischen den Registerwerten und dem Verhalten des AXI-Sensor-Moduls näher eingegangen.

5.2.3.2 Hardware-Design des Sensor Array Moduls

Das Sensor Array Modul ist die zentrale Steuereinheit des IP-Funktionsblocks und koordiniert die Ansteuerung der Sensor Module. Abbildung 5.16 zeigt das Blockschaltbild der Hardware-Komponente. Es wird mit einer Taktfrequenz von 100 MHz betrieben und ist direkt mit den AXI4-Slave-Registern verbunden. Das Modul ist als Finite-State-Machine modelliert. Anhand der vorliegenden Registerwerte, erfolgt die Umsetzung eines definierten Messdurchlaufs.
Durch Setzten von Bit 31 in Register 4 wird ein Messvorgang nach gewählter Konfiguration gestartet. Die Ansteuerung der Sensor Module erfolgt anschließend sequenziell, von Sensor 0 aufsteigend. Das Aktivierungsintervall innerhalb eines Messvorgangs ist zeitlich nicht konstant, sondern erfolgt jeweils nach Vorliegen eines gültigen Messwertes des aktivierten Sensors. Dieses Vorgehen stellt sicher, dass die Wahrscheinlichkeit deiner gegenseitige Beeinflussung der Ultraschallsensoren reduziert wird. Dieses ist zum Beispiel bei einer Interferenz von emittierten Schallwellen der Fall oder bei Empfang eines nicht selbst gesendeten Impulses. Nachdem von allen aktivierten Sensoren ein Messwert vorliegt, wird Bit 0 in Register 5 gesetzt. Die Quittierung und Löschen des Bits erfolgt ausschließlich über den I2C-Master. Der Messvorgang wird so lange fortgesetzt, bis das Bit 31 in Register 4 wieder gelöscht wird.
Eine besondere Herausforderung beim Hardware-Design dieses Moduls liegt in der Problematik des Erhalts der Datenintegrität für die ermittelten Sensordaten. Dies begründet sich in einem nicht vorhersagbarem Abfragezeitpunkt der Werte durch den I2C-Master. Zur Lösung der Problematik wird die im Folgenden beschriebene Zugriffsregelung genutzt.
Die Aktualisierung der Sensorwerte in den AXI-Registern erfolgt jeweils nach einem vollständig durchgeführten Messdurchlauf. Somit sind die Sensorwerte untereinander zeitlich konsistent. Diese werden nach jedem Messdurchlauf durch die nachfolgenden aktuellen Sensorwerte ersetzt. Durch setzten des Bit 17 in Register 4 wird das Ersetzen unterbrochen und die Übertragung der Sensordaten über die I2C-Schnittstelle kann erfolgen. Da die Nutzdatengröße je I2C-Frame auf 32-Bit begrenzt ist, werden für einen vollständigen Lesezugriff bei sechs Sensoren drei Leseanfragen durch den Master benötigt. Anschließend setzt dieser das Bit 16 in Register 4 und das Sensor Array Modul führt eine Löschung der abgerufenen Sensorwerte in den Registern 0-2 durch. Anschließend werden Bit 16 und 17 der Register 4, durch den I2C-Master zurückgesetzt. Damit ist der Auslesevorgang abgeschlossen. Dies quittiert das Sensor Array Modul durch zurücksetzen von Bit 0 des Registers 5.

Gemäß Spezifikation N5 ist eine variable Pausenzeit zwischen den Messabläufen gefordert. Dies wird über einen Zähler realisiert, welches über das Register 3 mit einem Compare-Zählerwert geladen werden kann. Die hierfür benötigte Anzahl von 28-Bit reicht bei einer Taktfrequenz von 100 MHz aus, um die maximal geforderte Zeitspanne von 1 s einzuhalten.

5.2.3.3 Hardware-Design des Sensor Moduls

Dieses befasst sich mit dem Hardware-Design des Sensor Moduls. Abbildung 5.17 zeigt das Blockschaltbild der Hardware-Komponente. Es besteht aus insgesamt vier Modulen. Einer Steuereinheit, einem Zähler, einer Flankenerkennung und einem Taktteiler. Das Hardware-Modul wird mit zwei unterschiedlichen externen Taktfrequenzen betrieben. Die Steuereinheit arbeitet mit dem Systemtakt des Mikroprozessors von 100 MHz. Der Taktteiler ist wiederum an ein Taktsignal von $5,2$ MHz angeschlossen. Zur weiteren Reduzierung der Frequenz auf 325 kHz für den Zähler und die Flankenerkennung, wird die Methode des Clock-Gating genutzt. Dies ist an den vorhandenen Clock-Enable-Eingängen der beiden Module zu erkennen.
Nach der Aktivierung des Sensor Moduls sendet dies ein Trigger-Signal an den angeschlossenen Ultraschallsensor, welcher daraufhin den Messvorgang startet. Als nächstes entsperrt die Steuereinheit den Zähler. Anschließend erfolgt eine Pulsbreiten-Bestimmung des eingehenden Echo-Signales. Dafür wird der Zähler gestartet, sobald der Flankendetektor eine steigende Signalflanke des ECHO-Signales erkennt. Durch eine fallende ECHO-Signalflanke wird der Zähler gestoppt. Die Pulsbreite, welche die Entfernungsinformation darstellt, liegt nun als Zählerwert vor. Über ein Vier-Phasen-Handshake-Verfahren [17, S.331ff] wird der gültige Datenwert von der Steuereinheit eingelesen und in ein internes Register geschrieben. Dieses kann anschließend vom übergeordneten Sensor Array Modul ausgelesen werden. Die Steuereinheit und der Zähler sind hierbei als Finite-State-Maschine entworfen. Das vorliegende Hardware-Design ist eine Abwägung zwischen Funktionalität, Ressourcen-Verbrauch und Modularisierungsgrad.
Die Verwendung von zwei unterschiedlichen Taktsignalen, birgt das Risiko eines unzulässig großen Taktversatzes zwischen diesen. Das Argument für eine Nutzung des zusätzlichen $5,2$ MHz Taktsignals, ergibt sich aus der vollständigen Hardware-Architektur für die autonome und manuelle Fahrzeugsteuerung. Ein Großteil der zu entwerfenden IP-Funktionsblöcke benötigt zur Abtastung und Generierung der PWM-Signalen, interne Zähler bzw. Taktteiler. Die Größe der internen Register hängt direkt von der verwen-

deten Taktfrequenz und der erforderlichen zeitlichen Auflösung ab. Durch die Nutzung einer zusätzlichen Clock-Domäne von 5,2 MHz, ergibt sich eine geringere Anzahl an benötigten Flip-Flops in den Taktteilern. Eine direkte Generierung von Frequenzen kleiner 4,687 MHz durch den MMCM ist bei einer externen 12 MHz Taktquelle nicht möglich. Die Übertragung des Zählerwertes mittels Vier-Phasen-Handshake, stellt selbst bei einem unzulässig großem Taktversatz, eine Datenübertragung zwischen dem Zähler und der Steuereinheit sicher. Hierdurch können während der anfänglichen Implementierungsphase, gültige Messergebnisse ausgelesen werden. Dieses Verfahren erfordert allerdings einen erhöhten Implementierungsaufwand, kann aber in Folgeprojekten vereinfacht werden.
Eine grundlegende Problematik ergibt sich jedoch beim Startvorgang des Zählers. Nachdem das Triggersignal gesendet wird, erwartet der Zähler steigende Signalflanke des Echo-Signales. Ist jedoch kein Ultraschallsensor angeschlossen, tritt dieser Fall nie ein. Das System läuft in einen Lock-Zustand. Eine Ansatz zur Lösung dieses Problems, würde die zusätzliche Funktion des Zählers als Watchdog-Timer sein, welches in dieser Arbeit allerdings nicht umgesetzt wird.

5.3 Implementierung und Konfiguration des System-on-Chip

Dieser Abschnitt stellt die Umsetzung des System-on-Chip zur Multi-Sensor-Signalverarbeitung vor. Zunächst erfolgt die Hardware-Implementierung und Adaptierung des elektronischen System auf das FPGA. Nach der darauffolgenden Software-Implementierung wird eine abschließende Leistungsanalyse und Beurteilung der Entwurfsqualität durchgeführt.

5.3.1 Hardware-Implementierung und Konfiguration des System-on-Chip

Im Folgenden werden zunächst die zuvor konzipierten Hardware-Module zur Multi-Sensor-Signalverarbeitung als IP-Funktionsblöcke realisiert. Anschließend wird die Hardware-Architektur des System-on-Chip umgesetzt. Abbildung 5.18 zeigt das resultierende Block-Design. Tabelle 5.4 listet die verwendeten IP-Funktionsblöcke sowie die gewählte Konfi-

guration und Dimensionierung der Hardware-Komponenten. Dabei wird zwischen der erforderlichen Funktionalität und dem resultierenden Ressourcen-Verbrauch abgewogen.

Wie in Grundlagen-Kapitel 2.2 beschrieben, bietet der IP-Core des Prozessors eine Vielzahl an Implementierungsvarianten und Konfigurationsmöglichkeiten. Die beabsichtigte Anwendung des System-on-Chip als I2C-Slave im Coprozessor-System und einer alleinigen Ausführung von Speicherzugriffsroutinen erfordert keine hohe Rechenleitung. Daraus resultiert eine Komponenten reduzierte Implementierungsvariante der Hardware-Architektur des Soft-Core-Prozessors. Es wird die kleinst möglich einstellbare Wortbreite des MicroBlaze von 32-Bit verwendet. Die Prozessor-Taktfrequenz beträgt 100 MHz.
Da die Anwendung des externen SRAM-Speichers beabsichtigt ist, sind zwei getrennte Cache-Speicher mit je 16 kB vorhanden. Zur Anbindung der benötigten Peripherie Module, besitzt der Soft-Core-Prozessor eine AXI4-Lite-Master-Schnittstelle. Die Größe des lokalen Speichers beträgt 64 kB. Dies ist ausreichend, da die Programmgröße, bei der jetzigen Implementierungsstufe, geringer ist (siehe Abschnitt 5.4).
Darüber hinaus macht das Software-Design die Verwendung eines Interrupt-Controller-Moduls notwendig. Dieses wird mit einer Fast-Interrupt-Schnittstelle implementiert, was die Latenzzeit zwischen einem Interrupt-Request und dem Ausführungsbeginn der Interrupt-Service-Routine reduziert. Dies ist beim vorliegenden Software-Design von Vorteil.
Um ein Debuggen des Coprozessors zu ermöglichen, ist ein MicroBlaze Debug Module integriert. Hierbei ist die minimale Komponenten-Ausführung gewählt, welches ein grundlegendes System-Debugging ermöglicht.
Zur Generierung der internen Taktsignale, wird eine Mixed-Mode Clock Manager verwendet. Dieser erzeugt aus der externen 12 MHz Oszillatorfrequenz die benötigten internen Takt-Domänen. 100 MHz für den Soft-Prozessor-Core und dessen Peripherie-Komponenten, 50 MHz für die QSPI-Schnittstelle sowie die $5,2$ MHz für das AXI-Sensor-Modul.
Der External-Memory-Controller ist entsprechend des externen SRAM-Bausteines konfiguriert. Das AXI-QSPI-Modul wird im Standard Mode betrieben, und nicht als SPI-Schnittstelle, anstatt QSPI-Schnittstelle. Dieses ist in einem Design-Fehler des IP-Funktionsblocks begründet, welcher zum Zeitpunkt der Arbeit noch nicht behoben wurde.
Das UART-Modul ist auf eine Symbolrate von 9600 Bd eingestellt. Das AXI-IIC-Modul ist auf eine 7-Bit Adressbreite konfiguriert. Dieses genügt, da hierbei ausreichend viele Bus-Adressen zur Verfügung stehen. Die I2C-Übertragungsrate von 100 kHz (Standard Mode) vom Master definiert. Das AXI-GPIO-Modul beinhaltet ein 32-Bit AXI4-Slave-Register, über welches die GPIO-Pins angesteuert werden.

Tabelle 5.4: Hardware-Komponenten und Konfiguration des System-on-Chip

IP-Modul	**Konfiguration**
MicroBlaze	32 Bit RISC-Architektur Prozessor-Taktfrequenz: 100 MHz Cache-Speicher (Instruction): 16 kB Cache-Speicher (Data): 16 kB AXI4-Lite-Master-Interface
Lokaler-Speicher	Kapazität: 64 kB
Interrupt-Controller [35]	AXI4-Lite-Slave-Interface Fast-Interrupt-Interface
Processor System Reset [21]	Externer High-aktiver Reset
MicroBlaze Debug Module [32]	Basic Debug-Interface
MMCM [37]	Eingangsfrequenz 1: 12 MHz Ausgangsfrequenz 1: 100 MHz Ausgangsfrequenz 2: 50 MHz Ausgangsfrequenz 3: 5,2 MHz
AXI-EMC [34]	AXI4-Lite-Slave-Interface Speicherart: asynchroner SRAM Speicher-Datenbreite: 8 Bit
AXI-QSPI [31]	AXI4-Lite-Slave-Interface Modus: SPI SPI-Taktfrequenz: 3,125 MHz FIFO-Tiefe: 16 x 8 Bit
AXI-UART [36]	AXI4-Lite-Slave-Interface Baudrate: 9600 Bd Datenbitanzahl: 8 Bit Stopbitanzahl: 1 Bit Paritybit: ohne
AXI-ICC [23]	AXI4-Lite-Slave-Interface Slave-Adressbreite: 7 Bit
AXI-GPIO [22]	AXI4-Lite-Slave-Interface AXI-Register-Anzahl: 1x32 Bit
AXI-FSDM	AXI4-Lite-Slave-Interface AXI-Register-Anzahl: 4x32 Bit
AXI-SENSOR	AXI4-Lite-Slave-Interface AXI-Register-Anzahl: 6x32 Bit
AXI-SmartConnect [38]	2x AXI-Slave, 1x AXI-Master
AXI-Interconnect [26]	1x AXI-Slave, 7x AXI-Master
Concat [25]	Eingänge: 3 Ausgänge: 3

Das AXI-FSDM-Modul ist abweichend zum Hardware-Design mit vier Registern ausgestattet, wobei nur eines genutzt. Dies resultiert aus der minimal möglichen Register-Anzahl, welche durch die Entwicklungsumgebung festgelegt ist. Das Sensor-Modul besitzt wie im Hardware-Design beschrieben sechs Register.
Über die Module AXI-SmartConnect und AXI-Interconnect wird der AXI-Master mit den AXI-Slaves verbunden. Der Concat IP-Funktionsblock wird benötigt, um die Hardware-Interrupt-Signale mit dem Interrupt-Controller zu verbinden.
Nachdem die vollständige Implementierung der autonomen und manuellen Fahrzeugsteuerung durchgeführt ist, können in Folgeprojekten im Rahmen einer genaueren Hard- und Software-Analyse, die durchgeführten Konfigurationen und Implementierungen Ressourcen-Orientiert angepasst werden.
Im nachfolgenden Abschnitt wird die Speicherzuordnung (Memory Mapped I/O) des Coprozessors aufgezeigt.

5.3.2 Adressraum des System-on-Chip

Tabelle 5.5 zeigt den Adressraum des Coprozessors (Memory Mapped I/O). Bei der vorliegenden Implementierung ist für die Peripherie-Module der automatisch vorgewählte Standardwert von 64 kB übernommen worden. Lediglich für den lokalen Speicher und den externen SRAM-Speicher erfolgt eine Anpassung der Adressierung, welche mit der vorhandenen Speicher-Kapazität übereinstimmt. Nach der vollständigen Implementierung der autonomen und manuellen Fahrzeugsteuerung in Folgeprojekten, sollte der Adressraum angepasst werden. Die Basis-Adressen des AXI-FSDM-Moduls und AXI-Sensor-Moduls sind entsprechend dem Hardware-Design der Register nach 5.2.3.1 gewählt.

5.3.3 Software-Implementierung des System-on-Chip

Das Programm zur I2C-Datenverarbeitung basiert auf einem frei verfügbaren Beispiel-Programm der Firma Xilinx [28]. Es umfasst wesentliche Software-Bestandteile zur Initialisierung des Coprozessors als I2C-Slave-Komponente. Im Rahmen dieser Arbeit erfolgt eine Modifikation und Erweiterung des Programms, um die im Design beschriebenen Funktionalitäten. Der hierbei gewählte Programmaufbau ermöglicht zudem die leichte Erweiterung der Funktionalität auf die autonome und manuelle Fahrzeugsteuerung in

Tabelle 5.5: Adressraum

IP-Modul	Offset	Größe	Höchste Adresse
Local Memory (Data)	0x00000000	64k	0x0000FFFF
Local Memory (Instruction)	0x00000000	64k	0x0000FFFF
GPIO	0x40000000	64k	0x4000FFFF
UART	0x40600000	64k	0x4060FFFF
IIC	0x40800000	64k	0x4080FFFF
Interrupt-Controller	0x41200000	64k	0x4120FFFF
Quad-SPI	0x44A00000	64k	0x44A0FFFF
FSDM	0x44A10000	64k	0x44A1FFFF
Sensor	0x44A20000	64k	0x44A2FFFF
EMC (Data)	0x60000000	512k	0x6007FFFF
EMC (Instruction)	0x60000000	512k	0x6007FFFF

Folgeprojekten. Tabelle 5.6 listet die verwendeten bzw. erstellten Quelldateien, sowie deren Implementierte Funktionalität. Zu jeder dieser Dateien existiert eine namensgleiche Headerdatei.

5.3.4 Konfiguration des Bootloaders

Das Bootloader-Programm wird ebenfalls von der Firma Xilinx zur Verfügung gestellt. In diesem muss lediglich die Speicheradresse angepasst werden, an der das Programm in den Flash-Speicher geladen werden soll. Hierzu ist nach Herstellerangaben des FPGA-Moduls Cmod A7 in der Headerdatei blconfig.h das Macro FLASH_IMAGE_BASEADDR auf 0x00300000 zu setzten.

5.4 Leistungsanalyse und Ressourcen-Verbrauch

Tabelle 5.7 listet die genutzten Hardware-Ressourcen nach der Implementierung des System-on-Chip. Die erhöhte Nutzung des BRAM, ergibt sich aus der gewählten Größe des lokalen Speichers, sowie aus den vorhandenen Cache-Speichern. Es zeigt sich, dass für

Tabelle 5.6: Quelldateien des System-on-Chip

Quelldatei	**Funktionalität**
microBlazeSystem.c	Beinhaltet die Hauptfunktion sowie die erforderlichen I2C-Interrupt-Service-Routinen
initSystem.c	Initialisierung der Hardware-Komponenten - I2C, GPIO und Interrupt-Controller
platform.c	Initialisierung der Hardware-Komponenten - UART und Cache-Speicher des Soft-Prozessor-Cores
iic.c	I2C-Datenverarbeitung - Lesen des I2C-Empfangsregisters, Setzen des I2C-Senderegisters, Auswertung der I2C-Frame-Paketes, Auswahl der Speicherzugriffsroutine (Modul-Selektion)
gpio.c	Ausführung Speicherzugriffsroutinen - Ansteuerung des AXI-GPIO-Moduls (Rx-/Tx-LED)
fsdm.c	Ausführung Speicherzugriffsroutinen - Ansteuerung des AXI-FSDM-Moduls
sensor.c	Ausführung Speicherzugriffsroutinen - Ansteuerung des AXI-Sensor-Moduls
cleanupSystem.c	Shut-Down-Routinen - Trennung der UART-Schnittstelle und Cach-Speicher leeren

Tabelle 5.7: Hardware-Ressourcen-Verbrauch des System-on-Chip nach der Implementierung

Resourcen	**Anzahl**	**Prozent**
LUT	6817	32,77
LUTRAM	991	10,32
FF	6790	16,32
BRAM	26	52,00
IO	58	54,72
BUFG	5	15,63
MMCM	1	20

anschließende vollständige Umsetzung der autonomen und manuellen Fahrzeugsteuerung noch genügend Ressourcen zur Verfügung stehen.
Bei der momentanen Konfiguration wird ausschließlich der lokale Speicher verwendet. Die optionale Nutzung des externen SRAM-Speichers ist noch nicht erforderlich. Es ergibt sich nach der Implementierung eine Programmgröße mit 23,160 kB.

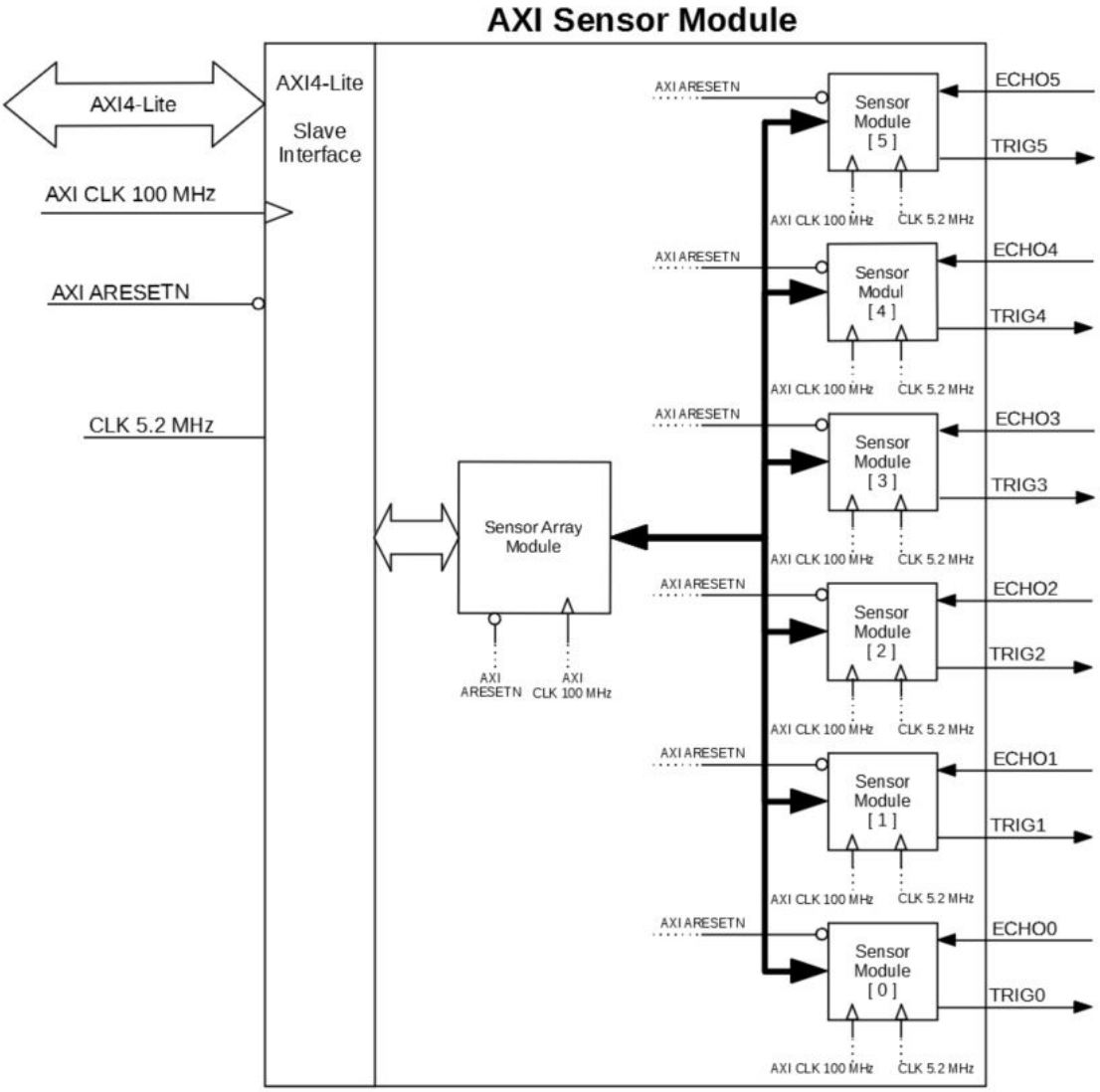

Abbildung 5.9: Blockdiagramm des AXI Sensor Moduls

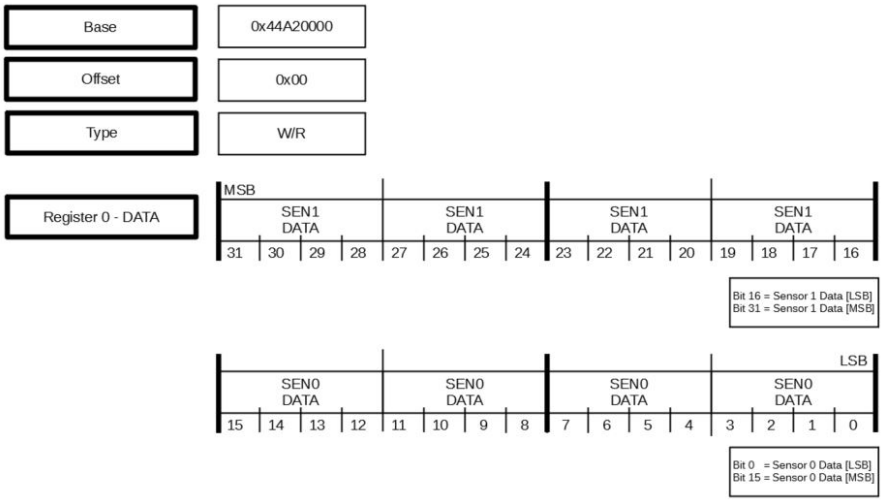

Abbildung 5.10: Register-Beschreibung FSDM-Modul (Register 0)

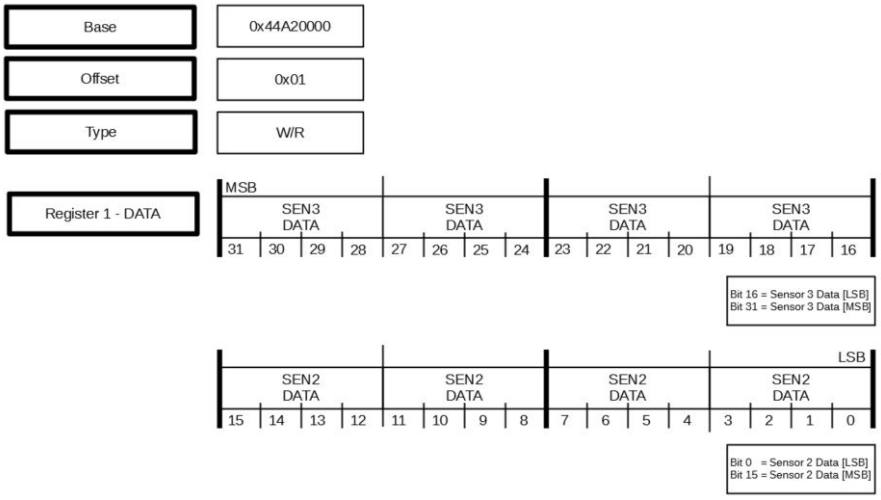

Abbildung 5.11: Register-Beschreibung FSDM-Modul (Register 1)

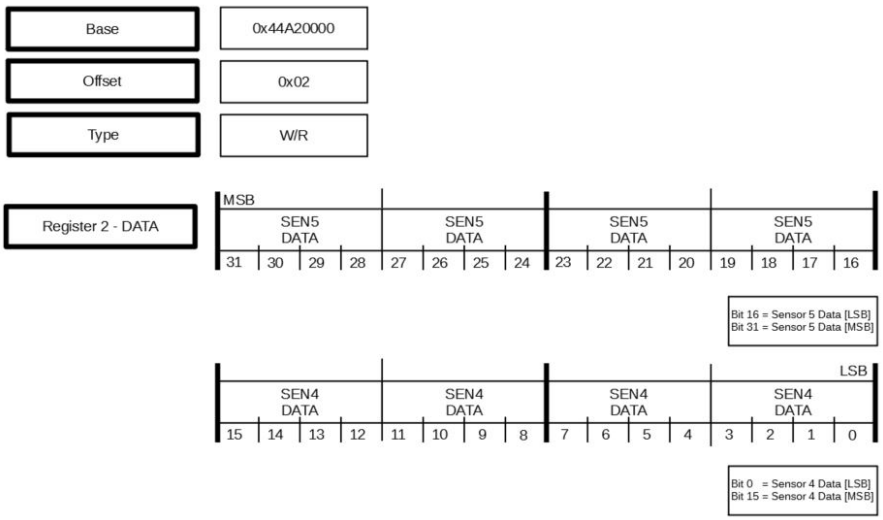

Abbildung 5.12: Register-Beschreibung FSDM-Modul (Register 2)

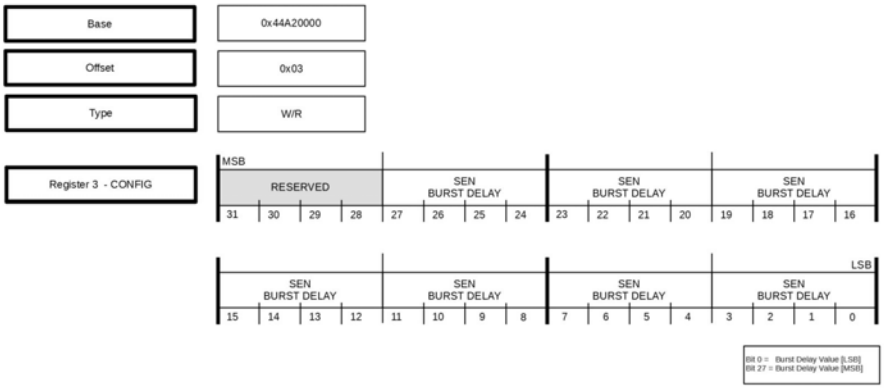

Abbildung 5.13: Register-Beschreibung FSDM-Modul (Register 3)

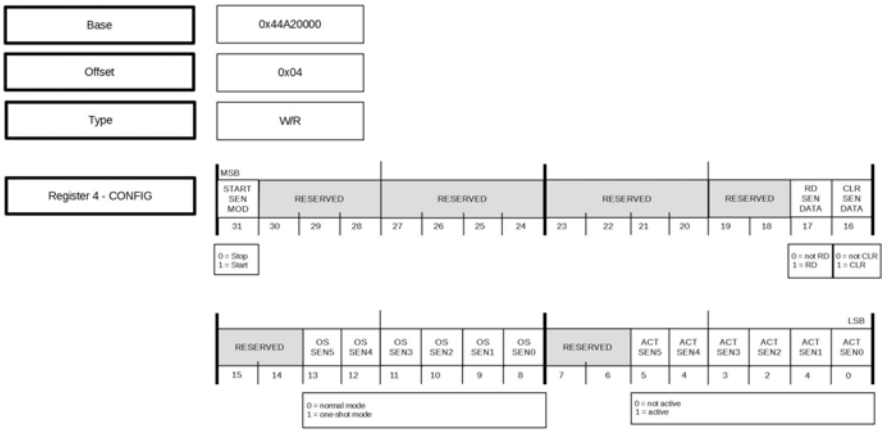

Abbildung 5.14: Register-Beschreibung FSDM-Modul (Register 4)

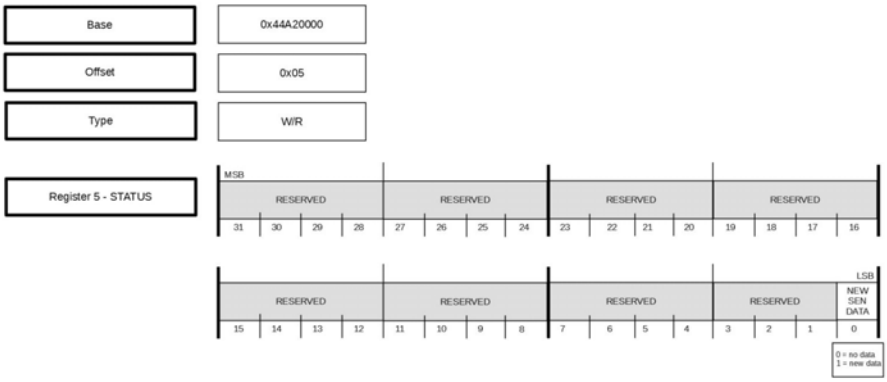

Abbildung 5.15: Register-Beschreibung FSDM-Modul (Register 5)

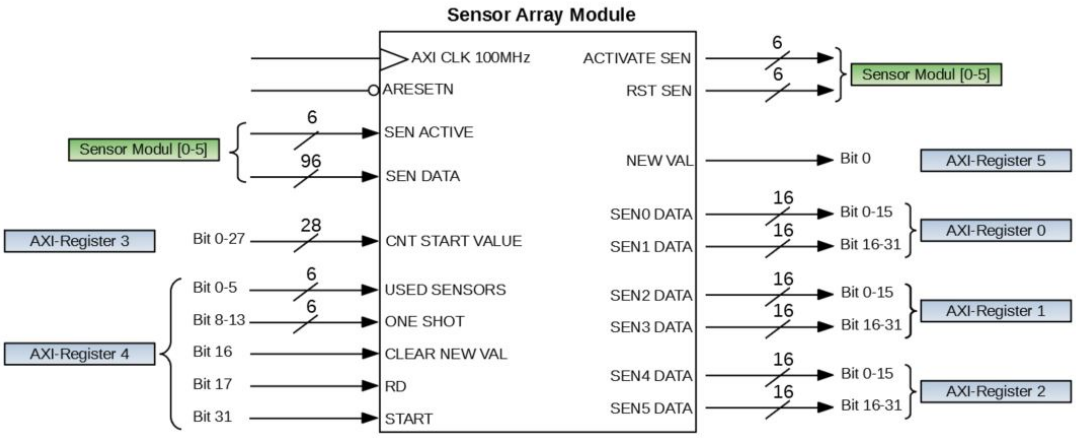

Abbildung 5.16: Blockschaltbild des Sensor Array Moduls

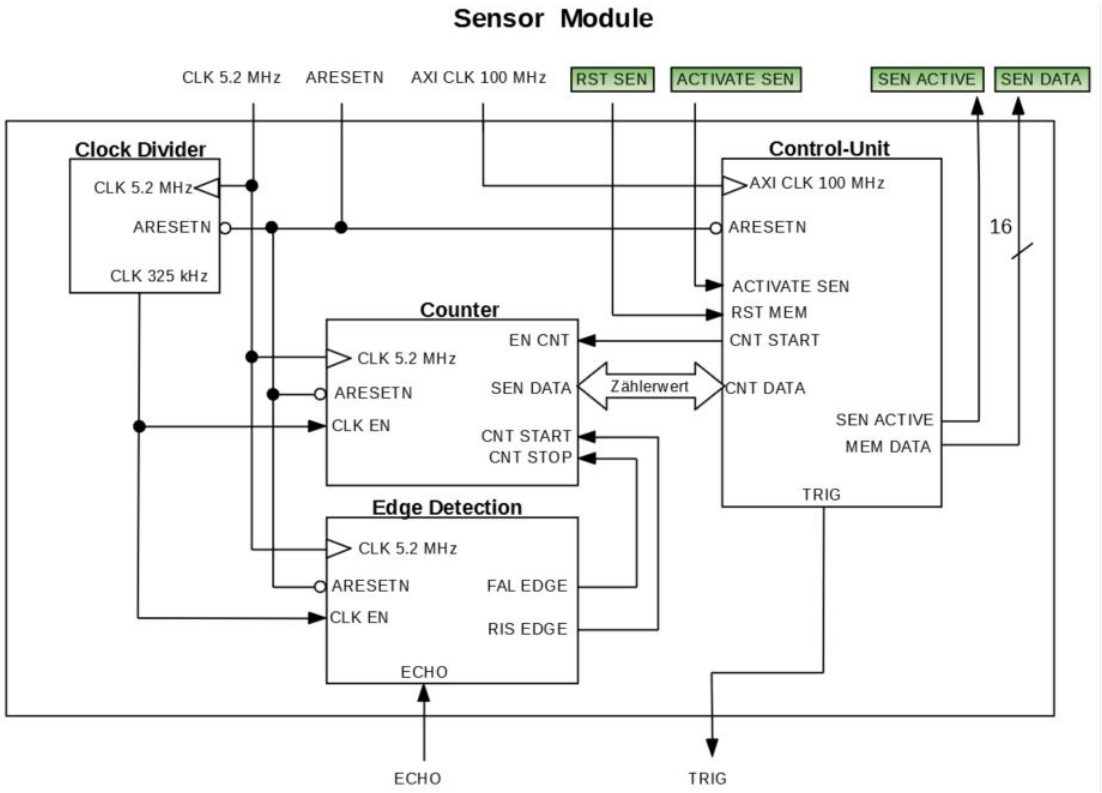

Abbildung 5.17: Blockschaltbild des Sensor Moduls

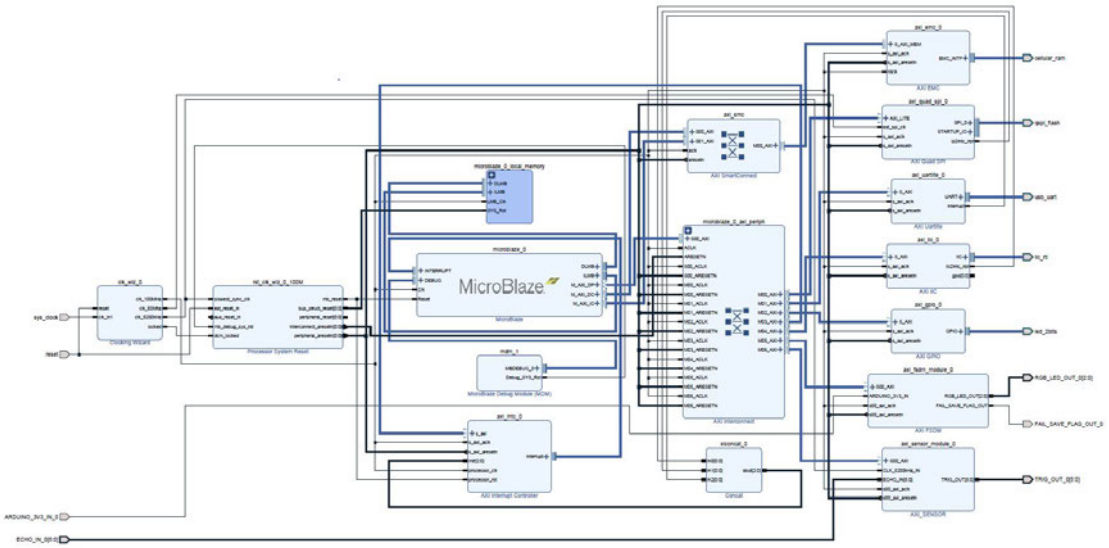

Abbildung 5.18: Block-Design des System-on-Chip

6 Systemtest und Validierung des System-on-Chip

In diesem Kapitel erfolgt der Systemtest und die Validierung des implementierten System-on-Chip zur Multi-Sensor-Signalverarbeitung. Der Systemtest umfasst hierbei die Objekt-Entfernungsmessung mit einem Sensor-Modul, sowie die Überprüfung des Systemverhaltens der Multi-Sensor-Signalverarbeitung bei unterschiedlichen Konfigurationen. Anschließend wird das entworfene Coprozessor-System auf die Einhaltung der geforderten Spezifikationen aus der Anforderungsanalyse hin überprüft.

6.1 Systemtest der Multi-Sensor-Signal-Verarbeitung

Der Systemtest erfolgt mit dem in Kapitel 4.3.1 konzipierten Demonstratorsystems. Dieser soll die grundlegenden Funktionalitäten des entwickelten IP-Funktionsblocks zur Multi-Sensor-Signalverarbeitung überprüfen und umfasst daher nur eine eingeschränkte Auswahl an Testfällen.
Zunächst wird eine definierte Objektentfernungsmessung, unter Verwendung eines einzelnen Ultraschallsensors, durchgeführt. Anschließend erfolgt eine Überprüfung des Systemverhaltens unter Verwendung unterschiedlicher Konfigurationsparameter des IP-Funktionsblocks zur Multi-Sensor-Signalverarbeitung. Für die folgenden Versuche wird eine Testsoftware verwendet, welche den ATmega328P als I2C-Master konfiguriert und die Ansteuerung des Coprozessors ermöglicht. Ziel ist die Feststellung der grundlegenden Funktionsfähigkeit.
Abbildung 6.1 zeigt den Versuchsaufbau zur Objekt-Entfernungsbestimmung durch die Multi-Sensor-Signalverarbeitung. Hierbei wird ein Gegenstand mit möglichst optimalen Reflexionseigenschaften verwendet und dieser mit einer Distanz von 10 cm zum Sensor platziert.

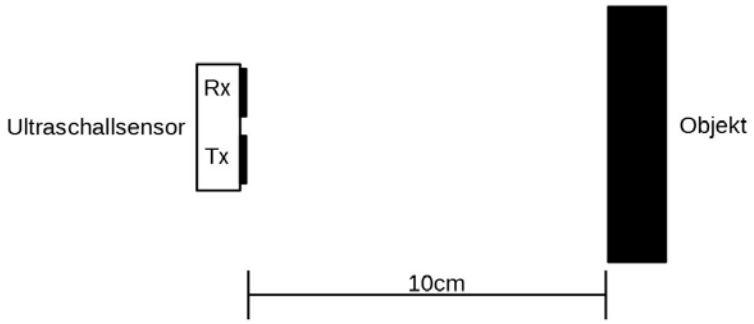

Abbildung 6.1: Versuchsaufbau zur Objekt-Entfernungsbestimmung

Die Messreihe umfasst dabei $n = 50$ Einzel-Messungen. Zur Beurteilung der Messergebnisse werden aus den gewonnenen Messdaten der arithmetische Mittelwert $\bar{x}$, die empirische Varianz s^2_{abs}, die absolute Standardabweichung s_{abs} sowie die relative Standardabweichung s_{rel} aus den Messwerten x_i wie folgt bestimmt

Arithmetischer Mittelwert:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Empirische Varianz:

$$s^2_{abs} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

Absolutee Standardabweichung der Stichprobe:

$$s_{abs} = \sqrt{s^2_{abs}} = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}$$

Relative Standardabweichung der Stichprobe:

$$s_{rel} = \frac{s}{\bar{x}}$$

Tabelle 6.1 zeigt die ermittelten Größen nach der Durchführung des Systemtests. Eine vollständige Auflistung der Messreihe befindet sich im Anhang A.

Tabelle 6.1: Ermittelte Kenngrößen bei der Objekt-Entfernungsmessung unter Verwendung eines Ultraschallsensors

$\bar{x}$	s_{abs}	s_{rel}	x_{min}	x_{max}
9,85 cm	0,18 cm	1,83 %	9,60 cm	10,03 cm

Der sich ergebende Mittelwert liegt bei 9,85 cm. Die absolute Standardabweichung beträgt hierbei 0,18 cm, was einer relativen Standardabweichung von 1,83 % entspricht. Dieses zeigt eine geringe Streuung der Messwerte um den Mittelwert. Die Messung zeigt jedoch auch, dass der Mittelwert unterhalb von 10 cm liegt und somit eine kürze Distanz gemessen wird.
Bei der Interpretation der Messergebnisse müssen allerdings die noch nicht kompensierten systematischen Messfehler berücksichtigt werden. So erfolgt keine Lufttemperatur-Kompensation durch die Hard- oder Software. Auch die Reflexionseigenschaften des Objektes konnten im Vorfeld der Messung nicht exakt messtechnisch bestimmt werden. Eine weitere Fehlerquelle könnten Rundungsfehler innerhalb der Testsoftware sein. Grundsätzlich bestätigt der Systemtest die Funktionsfähigkeit der Objekt-Entfernungsmessung durch den entworfenen IP-Funktionsblock bei der Verwendung eines Ultraschallsensors.
Im zweiten Testabschnitt wird das Verhalten des IP-Funktionsblocks bei unterschiedlichen Konfigurationsparametern überprüft. Dieses umfasst das Einstellen des Abfrageintervalls, das Aktivieren/Deaktivieren einzelner Sensoren, die Überprüfung des One-Shot-Mode und die Objekt-Entfernungsmessung beim Betrieb von bis zu sechs Ultraschallsensoren. Während des Tests werden alle Konfigurationsmöglichkeiten überprüft. Tabelle 6.2 zeigt zusammenfassend die Ergebnisse dieser Überprüfung.

Tabelle 6.2: System-Verhalten bei unterschiedlichen Konfigurationsmöglichkeiten

Überprüfung	**Erfüllt**
setzen des Abfrageintervalls [Register 3]	ja
aktivieren/deaktivieren einzelner Sensoren [Register 4]	ja
aktivieren/deaktivieren One-Shot-Mode [Register 4]	ja
Objekt-Entfernungsmessung mit 6 Ultraschallsensoren [Register 0-2]	ja

6.2 Validierung des System-on-Chip

In diesem Abschnitt wird die Validierung des implementierten System-on-Chip durchgeführt. Hierbei werden die geforderten Spezifikationen der Multi-Sensor-Signalverarbeitung mit den zuvor ermittelten System-Eigenschaften abgeglichen. Tabelle 6.3 zeigt, dass alle spezifizierten Anforderungen an die Multi-Sensor-Signalverarbeitung vom System erfüllt werden.

Tabelle 6.3: Validierung des System-om-Chip zur Multi-Sensor-Signalverarbeitung

ID	Erfüllt	Beschreibung
F2	ja	Die Ansteuerung des Coprozessors erfolgt über vorhandene Arduino-Bibliotheken.
F5	ja	Die optische Anzeige der Steuerungsart und eines Mikrocontroller-Ausfalls erfolgt über eine LED auf dem FPGA-Modul CmodA7.
F8	ja	Der Coprozessor stellt dem Mikrocontroller die Sensorwerte zur Verfügung.
F10	ja	Der Coprozessor besitzt zur Konfiguration der Sensoransteuerung eine I2C-Schnittstelle.
F11	ja	Der Coprozessor ermöglicht die Konfiguration der Sensoransteuerung durch den Mikrocontroller.
F12	ja	Der Messvorgang kann separat für jeden Sensor durch den Mikrocontroller gestartet werden. Ein gezieltes Deaktivieren von Sensoren ist möglich.
F13	ja	Die Ansteuerung der Sensoren erfolgt sequenziell.
N1	ja	Das Coprozessor-System nutzt weiterhin die Mikrocontroller-Plattform Arduino Nano.
N2	ja	Durch die Verwendung eines SoC-FPGAs ist eine hohe Flexibilität von Hard- und Software gegeben.

Fortsetzung auf der nächsten Seite

Tabelle 6.3: Validierung des System-om-Chip zur Multi-Sensor-Signalverarbeitung

Nr.	Erfüllt	Beschreibung
N3	ja	Das Coprozessor-System nutzt weiterhin die Mikrocontroller-Plattform Arduino Nano. Der Anschluss eines Raspberry Pi ist möglich. Voraussetzung ist das Vorhandensein einer I2C-Schnittstelle.
N4	ja	Die Slave-Adresse ist Software-seitig konfigurierbar.
N5	ja	Das Abfrageintervall liegt zwischen 0 s und 2,68 s
N6	ja	Die Ansteuerung von bis zu 6 Sensoren wird ermöglicht.
N7	ja	Die Auflösung ist kleiner als 50 mm
N8	ja	Der Coprozessor überträgt die Entfernungsinformation als Zählerwert an den Mikrocontroller. (Echo-Pulsbreite in CLK-Ticks)
N10	ja	Die Sensorwerte werden vom Datentyp unsigned 16 Bit im Mikrocontroller abgebildet.
N14	ja	Die Sensoren verfügen über einen One-Shot-Mode

7 Zusammenfassung und Ausblick der Arbeit

7.1 Zusammenfassung

In dieser Arbeit wurde erfolgreich die Konzeption und Implementierung eines System-on-Chip auf einem FPGA zur Multi-Sensor-Signalverarbeitung durchgeführt. In einem ersten Schritt wurde eine bestehende Fahrzeugelektronik des RC-Automodells analysiert. Hierbei konnten vorliegende Schwächen in der Hardware-Auswahl und Software-Architektur ermittelt werden. Diese resultierten aus einer unzureichenden Dimensionierung der Hardware-Komponenten sowie den verwendeten Programmbibliotheken für den eingesetzten Mikroprozessor. Darauf folgend wurde eine Anforderungsanalyse durchgeführt. Dabei wurden alle Projektbeteiligten zur Definition der Systemanwendungsfälle mit einbezogen. Aus den sich ergebenen Anforderungsspezifikationen ergab sich die Konzeption eines Coprozessor-Systems zur manuellen und autonomen Fahrzeugsteuerung. Die hierfür erforderliche Hinderniserkennung mittels Ultraschallsensoren erforderte die Konzeption und Implementierung einer Multi-Sensor-Signalverarbeitung. Dabei wurde ein FPGA mit implementierten Soft-Core-Prozessor genutzt, welcher über eine I2C-Schnittstelle verfügt. Abschließend erfolgte die Dimensionierung des benötigten Demonstrator-Systems. Im weiteren Verlauf wurde der Soft-und Hardware-Entwurf des System-on-Chip durchgeführt. Der Schwerpunkt lag hierbei auf dem Hardware-Design des Soft-Core-Prozessors MicroBlaze und den benötigten Peripherie-Komponenten. Dabei erfolgte der Entwurf des IP-Funktionsblocks, welcher die Funktionalität der Multi-Sensor-Signalverarbeitung beinhaltet. Die Software-Architektur des System-on-Chip ermöglicht dabei eine direkte Ansteuerung des entworfenen Hardware-Moduls über die I2C-Schnittstelle durch den Master. Bei der anschließenden Implementierung des System-on-Chip wurde der verwendete Soft-Core-Mikroprozessor Anwendungsbezogen konfiguriert und zusammen mit den erforderlichen Hardware-Modulen auf dem FPGA implementiert. Beim Anschließenden Systemtest konnte die Funktionsfähigkeit des System-on-Chip zur

Multi-Sensor-Signalverarbeitung bestätigt werden. Alle geforderten Spezifikationen aus der Anforderungsanalyse wurden umgesetzt und erfüllt.
Durch diese Arbeit konnte gezeigt werden, dass sich komplexe Hardware-Systeme durch die Methode des Hard- und Software-Codesigns innerhalb kurzer Zeit umsetzen lassen. Die gegebene Funktionsfähigkeit und Erfüllung aller Spezifikationen der Multi-Sensor-Signalverarbeitung zeigt die Tragfähigkeit des entwickelten Konzeptes für den Coprozessor und der entworfenen IP-Funktionsblöcke. Die beim Systemtest durchgeführten Objekt-Entfernungsmessungen zeigen Abweichungen zur exakten Position von unter $0,5$ cm bei einer geringen Streuung der Messergebnisse. Hierbei weist die implementierte Soft- und Hardware-Architektur eine hohe Flexibilität durch ihre Modularität auf, was bei geplanten Folgeprojekten vorteilhaft ist.

7.2 Ausblick

Das entwickelte System, welches die Multi-Sensor-Signalverarbeitung ermöglicht, bildet die Grundlage für nachfolgende Arbeiten. Hierzu zählt etwa die vollständige Umsetzung der manuellen und autonomen Fahrzeugsteuerung unter Verwendung des Coprozessor-Systems. Dazu kann das in der Arbeit erstellte Konzept als Ausgangspunkt verwendet werden. Allerdings ist hierfür eine ergänzende und vertiefende Anforderungsanalyse nötig, da der Fokus dieser Arbeit auf der Implementierung einer Multi-Sensor-Signalverarbeitung lag. Grundsätzlich ermöglicht das entwickelte Coprozessor-System weitreichende Implementierungs- und Anwendungsmöglichkeiten, welche weit über die vorgesehen Einsatz zur Steuerung autonomer Fahrzeuge hinausgehen. Diese Möglichkeiten sollen nicht ungenutzt bleiben.
Im Bezug auf diese Arbeit sind sämtliche Anforderungsspezifikationen an die Multi-Sensor-Signalverarbeitung erfüllt worden, dennoch sind mit Blick auf zukünftige Implementierungen weitergehende technische Fragestellungen zu beantworten und erforderliche Systemanpassungen durchzuführen. Diese Umfassen die im folgenden näher erläuterten Themenbereiche.

Weiterführende Untersuchung der Messabweichung der Multi-Sensor-Signalverarbeitung

Wie im Kapitel 6 beschrieben ist die Funktionsfähigkeit des IP-Funktionsblocks zur Multi-Sensor-Signalverarbeitung gegeben, jedoch ist eine Messabweichung bei der Objekt-

Entfernungsbestimmung festgestellt worden. Daher können im Rahmen einer weiterführenden Untersuchung mögliche Ursachen ermittelt und behoben werden.

Vollständige Implementierung der manuellen und autonomen Fahrzeugsteuerung

Aufbauend auf diese Arbeit kann das im Abschnitt 5.2.1 entwickelte Design der manuellen und autonomen Fahrzeugsteuerung umgesetzt werden. Hierzu sind noch drei weitere IP-Funktionsblöcke zu konzipieren und zu implementieren: Die PWM-Signal-Verarbeitung der Empfängersignale, die Ansteuerung der PWM-Aktoren sowie die Umsetzung eines PWM-Gateways. Ergänzend hierzu muss die Software-Architektur des System-on-Chip angepasst werden.

Optimierung der Hardware- und Software-Architektur des System-on-Chip

Bei der Umsetzung der Hard- und Software-Architektur des System-on-Chip lag der Fokus dieser Arbeit auf der Herstellung der Funktionsfähigkeit der Multi-Sensor-Signalverarbeitung und nur eingeschränkt auf einen geringen FPGA-Ressourcenverbrauch. Dies bietet die Möglichkeit zur Anpassung und Optimierung des System-on-Chip in Nachfolgeprojekten. Beispielhaft sind hier die optionale Verwendung des externen SRAM-Speichers oder die eingestellte Kapazität der Cache-Speicher des Soft-Prozessor-Cores zu nennen. Die Optimierung sollte nach der vollständigen Implementierung der manuellen und autonomen Fahrzeugsteuerung durchgeführt werden, da zu diesem Zeitpunkt der vollständige FPGA Ressourcen-Verbrauch beinhaltet ist.

Optimierung der I2C-Paket-Layouts

Die aktuell verwendeten I2C-Paket-Layouts erleichtern durch ihren Aufbau und Inhalt die Fehlersuche im System. Allerdings erfolgt durch die feste Paket-Anzahl je Frame eine Übertragung von Daten, die nicht für das Coprozessor-System benötigt werden. Insbesondere bei der Übertragung des Select-Befehls, also der Auswahl des Salve-Registers durch den Master, werden unnötige Daten-Pakete übertragen.

Implementierung eines Watchdog-Timers im IP-Funktionsblock der Multi-Sensor-Signalverarbeitung

Der in dieser Arbeit nicht bereitgestellte Funktionalität der Multi-Sensor-Signalverarbeitung, wenn kein Sensor angeschlossen oder dieser ausfällt, sollte behoben werden. Dies könnte zum Beispiel durch die Implementierung einer Watch-Dog-Timer-Funktionalität

geschehen. Hierbei ist es prinzipiell möglich, denselben Zähler zu verwenden, welcher die Pulsbreite des eingehenden Echo-Signales bestimmt.

Austausch des Spannungsreglermodul LM2596S

Mit dem weiteren Ausbau des Demonstrator-Systems steigt der Stromverbrauch. Hierbei muss darauf geachtet werden, dass dieser nicht unzulässig groß wird. Unter Umständen muss das Spannungsreglermodul LM2596S mit einem Kühlkörper versehen oder getauscht werden.

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] *ARDUINO NANO*, Januar 2019. – URL https://store.arduino.cc/arduino-nano. – Abgerufen am: 05.08.2020

[2] *Reference Guide - pulseIn() [Advanced I/O]*, Januar 2019. – URL https://www.arduino.cc/reference/de/language/functions/advanced-io/pulsein/. – Abgerufen am: 05.08.2020

[3] Adafruit Industries: *Adafruit PCA9685 16-Channel Servo Driver*, Juni 2019. – URL https://cdn-learn.adafruit.com/downloads/pdf/16-channel-pwm-servo-driver.pdf?timestamp=1596644670. – Abgerufen am: 05.08.2020

[4] ARM Ldt.: *AMBA-Specifications*. – URL https://www.arm.com/products/silicon-ip-system/embedded-system-design/amba-specifications

[5] ARM Ldt.: *AMBA AXI and ACE Protocol Specification*, März 2020

[6] Atmel Corporation: *8-bit AVR Microcontroller with 32K Bytes In-SystemProgrammable Flash, Rev.:7810D-AVR-01/15*, Januar 2015. – URL http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. – Abgerufen am: 28.05.2020

[7] Conrad Electronic SE: *Voltage Step Down Module*, 2019. – URL https://asset.conrad.com/media10/add/160267/c1/-/gl/002134134ML00/bedienungsanleitung-2134134-makerfactory-spannungsregler-mf-6402402-1-st.pdf. – Abgerufen am: 06.08.2020

[8] Cytron Technologies Sdn. Bhd.: *Product User's Manual – HCSR04Ultrasonic Sensor*, Mai 2013. – URL https://docs.google.com/document/d/1Y-

yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit. – Abgerufen am: 05.08.2020

[9] Digilent Inc.: *Cmod A7 Reference Manual*, Oktober 2019. – URL https://reference.digilentinc.com/_media/reference/programmable-logic/cmod-a7/cmod_a7_rm.pdf. – Abgerufen am: 06.08.2020

[10] Erfinder:Susan C. Hill, Mark R. H.: *Patent US4816996A*. Anmelder: Motorola Inc. (Veranst.). – URL https://worldwide.espacenet.com/patent/search/family/022138902/publication/US4816996A?q=pn=US4816996. – Abgerufen am: 06.08.2020

[11] Fairchild Semiconductor Corporation: *BSS138*, Oktober 2005. – URL http://cdn.sparkfun.com/datasheets/BreakoutBoards/BSS138.pdf. – Abgerufen am: 06.08.2020

[12] Hesse, Stefan ; Schnell, Gerhard: *Sensoren für die Prozess- und Fabrikautomation*. Springer Fachmedien Wiesbaden, 2018

[13] Lerch, Reinhard ; Sessler, Gerhard ; Wolf, Dietrich: *Technische Akustik*. Springer Berlin Heidelberg, 2009

[14] Noll, Martin ; Rapps, Peter: *Ultraschallsensorik*. Vieweg+Teubner Verlag, 2012. – 110–122 S

[15] NXP Semiconductors: *UM10204I2C-bus specification and user manual*, April 2014. – URL https://www.nxp.com/docs/en/user-guide/UM10204.pdf. – Abgerufen am: 06.08.2020

[16] NXP Semiconductors: *PCA9685*, April 2015. – URL https://www.nxp.com/docs/en/data-sheet/PCA9685.pdf. – Abgerufen am: 05.08.2020

[17] Reichardt, Jürgen: *Digitaltechnik*. Gruyter, Walter de GmbH, 2017. – URL https://www.ebook.de/de/product/26452780/juergen_reichardt_digitaltechnik.html. – ISBN 978-3-11-047800-6

[18] Sdn.Bhd., Cytron T.: *Product User's Manual - HC-SR04 Ultrasonic Sensor, V1.0*. Cytron Technologies Sdn.Bhd. (Veranst.), Mai 2013. – URL https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit. – Abgerufen am: 22.05.2020

[19] Texas Instruments Inc.: *PCA9306 Dual Bidirectional I2C Bus and SMBus Voltage-Level Translator*, April 2019. – URL https://cdn.sparkfun.com/assets/d/2/f/7/c/pca9306.pdf. – Abgerufen am: 06.08.2020

[20] Tränkler, Hans-Rolf (Hrsg.) ; Reindl, Leo (Hrsg.): *Sensortechnik*. Springer Berlin Heidelberg, 2014

[21] Xilinx Inc.: *Processor SystemReset Module v5.0*, November 2015. – URL https://www.xilinx.com/support/documentation/ip_documentation/proc_sys_reset/v5_0/pg164-proc-sys-reset.pdf. – Abgerufen am: 07.08.2020

[22] Xilinx Inc.: *AXI GPIO v2.0*, Oktober 2016. – URL https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf. – Abgerufen am: 07.08.2020

[23] Xilinx Inc.: *AXI IIC Bus Interfacev2.0*, Oktober 2016. – URL https://www.xilinx.com/support/documentation/ip_documentation/axi_iic/v2_0/pg090-axi-iic.pdf. – Abgerufen am: 07.08.2020

[24] Xilinx, Inc.: *Local Memory Bus(LMB) v3.0*, April 2016. – URL https://www.xilinx.com/support/documentation/ip_documentation/lmb_v10/v3_0/pg113-lmb-v10.pdf. – Abgerufen am: 04.08.2020

[25] Xilinx Inc.: *LogiCORE IP Concat (v2.1)*, April 2016. – URL https://www.xilinx.com/support/documentation/ip_documentation/xilinx_com_ip_xlconcat/v2_1/pb041-xilinx-com-ip-xlconcat.pdf. – Abgerufen am: 08.08.2020

[26] Xilinx, Inc.: *AXI Interconnect v2.1*, Dezember 2017. – URL https://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf. – Abgerufen am: 05.08.2020

[27] Xilinx, Inc.: *Vivado AXI Referenc*, Juli 2017. – URL https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf. – Abgerufen am: 05.08.2020

[28] Xilinx Inc.: *Xilinx - Embedded Software*, Januar 2017. – URL https://github.com/Xilinx/embeddedsw/blob/master/XilinxProcessorIPLib/

drivers/iic/examples/xiic_slave_example.c. – Abgerufen am: 07.08.2020

[29] Xilinx, Inc.: *7 Series FPGAs Data Sheet: Overview*, Februar 2018. – URL https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf. – Abgerufen am: 06.08.2020

[30] Xilinx, Inc.: *AXI Block RAM (BRAM)Controller v4.1*, Mai 2019. – URL https://www.xilinx.com/support/documentation/ip_documentation/axi_bram_ctrl/v4_1/pg078-axi-bram-ctrl.pdf. – Abgerufen am: 04.08.2020

[31] Xilinx Inc.: *AXI Quad SPI v3.2*, Juli 2019. – URL https://www.xilinx.com/support/documentation/ip_documentation/axi_quad_spi/v3_2/pg153-axi-quad-spi.pdf. – Abgerufen am: 07.08.2020

[32] Xilinx Inc.: *MicroBlaze Debug Module(MDM) v3.2*, November 2019. – URL https://www.xilinx.com/support/documentation/ip_documentation/mdm/v3_2/pg115-mdm.pdf. – Abgerufen am: 07.08.2020

[33] Xilinx, Inc.: *MicroBlaze Processor Reference Guide*, Mai 2019. – URL https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug984-vivado-microblaze-ref.pdf. – Abgerufen am: 03.08.2020

[34] Xilinx Inc.: *AXI EMC v3.0*, April 2020. – URL https://www.xilinx.com/support/documentation/ip_documentation/axi_emc/v3_0/pg100-axi-emc.pdf. – Abgerufen am: 07.08.2020

[35] Xilinx Inc.: *AXI InterruptController (INTC) v4.1*, Juni 2020. – URL https://www.xilinx.com/support/documentation/ip_documentation/axi_intc/v4_1/pg099-axi-intc.pdf. – Abgerufen am: 07.08.2020

[36] Xilinx Inc.: *AXI UART Lite v2.0*, Oktober 2020. – URL https://www.xilinx.com/support/documentation/ip_documentation/axi_uartlite/v2_0/pg142-axi-uartlite.pdf. – Abgerufen am: 07.08.2020

[37] Xilinx Inc.: *Clocking Wizard v6.0*, Mai 2020. – URL https://www.xilinx.com/support/documentation/ip_documentation/clk_wiz/v6_0/pg065-clk-wiz.pdf. – Abgerufen am: 07.08.2020

[38] Xilinx Inc.: *SmartConnect v1.0*, März 2020. – URL https://www.xilinx.com/support/documentation/ip_documentation/smartconnect/v1_0/pg247-smartconnect.pdf. – Abgerufen am: 07.08.2020

A Anhang

A.1 Messreihe - Objekt-Entfernungsmessung mittels Ultraschall

Tabelle A.1: Messreihe - Objekt-Entfernungsmessung mittels Ultraschall

Messung Nr. n	**Abstand** x_n/cm	**Messung Nr.** n	**Abstand** x_n/cm
1	9,60	26	9,60
2	10,03	27	9,97
3	9,97	28	9,97
4	9,66	29	9,66
5	9,66	30	9,97
6	9,97	31	9,66
7	9,66	32	10,03
8	9,60	33	10,03
9	10,03	34	9,97
10	9,97	35	10,03
11	9,97	36	10,03
12	9,97	37	9,97
13	10,03	38	9,66
14	9,97	39	9,60
15	9,97	40	10,03
16	9,97	41	9,60
17	9,97	42	9,66
18	9,60	43	9,60
19	10,03	44	9,97
20	9,60	45	9,60
21	10,03	46	9,97
22	9,60	47	10,03
23	9,66	48	9,60
24	10,03	49	9,97
25	10,03	50	9,97

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original