Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Walli Ahad

HW-/SW-Codesign eines Einparkassistenten mit Infrarot-Abstandssensorik für eine SoC-Plattform eines autonomen Fahrzeugs

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Walli Ahad

HW-/SW-Codesign eines Einparkassistenten mit Infrarot-Abstandssensorik für eine SoC-Plattform eines autonomen Fahrzeugs

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

Betreuender Prüfer : Prof. Dr.-Ing. Bernd Schwarz
Zweitgutachter : Prof. Dr.rer.nat. Reinhard Baran

Abgegeben am 25. November 2011

Walli Ahad

Thema der Bachelorarbeit

HW-/SW-Codesign eines Einparkassistenten mit Infrarot-Abstandssensorik für eine SoC-Plattform eines autonomen Fahrzeugs

Stichworte

System-on-Chip(SoC), Spartan 3E, SoC-Plattform, FPGA, Xillinx, SPI, HAW-SPI, XPS, Xillinx EDK, GP2D12-Infrarotsensor, PmodAD1, Einparkassistent, Polynom-Approximation

Kurzzusammenfassung

Diese Arbeit befasst sich mit der System-on-Chip(SoC)-basierten Übertragung von Distanzinformationen zwischen einem autonomen SoC-Fahrzeug und dessen Hindernissen. Die Distanzinformationen werden von vier Infrarotsensoren erfasst und über Analog-Digital-Umsetzer an die vier Empfangskanäle der seriellen Schnittstelle "HAW-SPI" weitergeleitet. Die vier Infrarotsensoren dienen dem SoC-Fahrzeug zur seitlichen Abstandsmessung u.a. zur Suche nach einer passenden Parklücke für den Parkassistent. Der Übertragungsvorgang wird in einem Zusammenspiel von Hardware- und Softwaremodulen bewerkstelligt.

Walli Ahad

Title of the paper

HW-/SW-Codesign a parking assistant with infrared distance sensor for an SoC platform for an autonomous vehicle

Keywords

System-on-Chip(SoC), Spartan 3E, FPGA, SoC-Plattform, SPI, HAW-SPI, XPS, Xillinx EDK, GP2D12-Infrarotsensor, PmodAD1, Parking Assistant, Polynom-Approximation

Abstract

This thesis deals with the system-on-chip(SoC)-based transmission of distance information between an autonomous vehicle and its obstacles. The distance data are recorded by four infrared sensors and passed trough an analog-digital-converter to the four receiver channels of the serial interface "HAW-SPI". The four infrared sensors are used by the SoC-vehicle for lateral distance measurement etc. to search for a suitable parking space for the parking assistant. The transfer process is accomplished in a combination of hardware and software modules.

Inhaltsverzeichnis

1. Einleitung

Diese Arbeit behandelt die Integration der Infrarotsensorik basierend auf einem System on Chip (SoC)-Design und dient der Erprobung der SoC-Technologie im Rahmen des FAUST-Projekts (vgl. Abbildung 1.1). SoCs vereinigen die Vorteile von Mikrocontrollern und paralleler Hardware. Während die weniger zeitkritischen Aufgaben in Software von einem Mikrocontroller bearbeitet werden, können die rechenintensiven Algorithmen durch parallele Hardware beschleunigt werden. Zudem stellt der SoC umfangreiche Softwarebibliotheken zur Bedienung der Peripherie, wie z.B. zur Parametrisierung von Sensoren, zur Verfügung.

1.1. Ziel und Schwerpunkt der Arbeit

Ziel dieser Arbeit ist die Erprobung der Technologie einer SoC-Plattform für die Signalverarbeitung. Für die Abstandmessung des autonomen SoC-Fahrzeugs zum einem bestimmten Objekt wird eine Infrarotsensor-FPGA-Kombination verwendet.

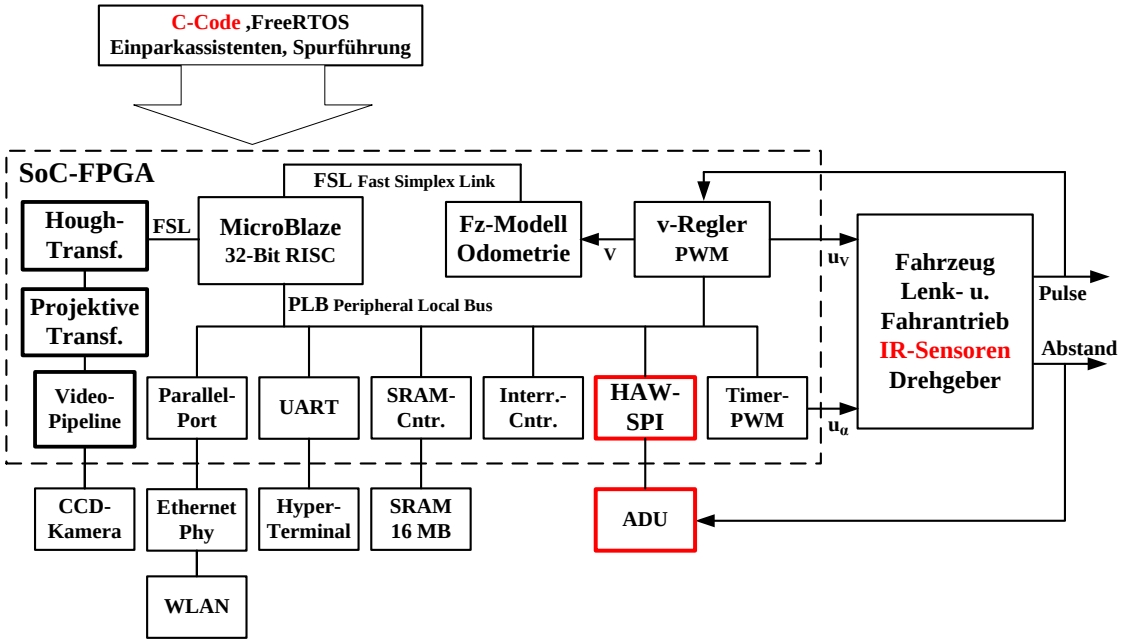

Abb. 1.1.: *System on Chip (SoC)-Plattform für die Entwicklung eines Einparkassistenten für ein autonomes SoC-Fahrzeug*

Die Schwerpunkte des Projekts "Hardware/Software-Codesign eines Einparkassistenten mit Infrarot-Abstandsensorik für eine SoC-Plattform eines autonomes Fahrzeugs" sind:

- die Integration von Infrarotsensoren im SoC-FPGA über den von der HAW überarbeiteten SPI-Bus(HAW-SPI)
- die Analyse des HAW-SPI IPs bezogen auf den Ressourcenbedarf, den Übertragungsvorgang, das Timing-Verhalten und den Energieverbrauch
- Neuentwurf des HAW-SPI IP-Cores mit dem Ziel SoC-Ressourcen zu sparen und das Timing-Verhalten des Datenübertragungsvorgangs zu verbessern, um somit die SoC-Prozessorauslastung zu reduzieren
- den Energieverbrauch zu reduzieren, indem die Übertragung der Messdaten aus dem IR-Sensor ins SoC erst dann erfolgen, wenn diese aktualisiert worden sind und vom SoC angefordert werden
- Einsatz des IR-Sensors für die Suche nach einer Parklücke für das Einparken des SoC-Fahrzeugs

1.2. Kapitelübersicht

In **Kapitel 2** wird die Übersicht der verwendeten SoC-Plattform vorgestellt. Außerdem wird auf den Softcore Prozessor MicroBlaze der Firma Xillinx , den zur Abstandmessung verwendeten GP2D12 Infrarotsensor (IR)-Sensor, den ADCS7476 (AD1) und die serielle Schnittstelle Serial Peripheral Interface (SPI) eingegangen.

Das **Kapitel 3** beschreibt die überarbeite SPI-Schnittstelle (HAW-SPI IP). Hierbei wird außerdem auf die Entwicklung einer Polynom-Funktion in Matlab für die Umrechnung der erfassten Nicht-linearen Messwerte in die tatsächlichen Abstände eingegangen. Das Timing-Verhalten und der Ressourcenbedarf der HAW-SPI wird mit dem Ziel eines besseren Entwurfs für die HAW-SPI ausgewertet.

In **Kapitel 4** werden Maßnahmen ergriffen, die dazu dienen die Anzahl der Softwareregister des HAW-SPI IPs von 14 auf 4, die Kombination der zwei ISRs zur Übertragung eines Datensatzes auf eine ISR und die Beteiligungszeit des Prozessors an der Übertagung zu reduzieren. Hierbei wird Außerdem auf den Entwurf eines integrierten Timers und eines Controller-Moduls für die Steuerung der Datenübertagung, die HAW-SPI Bustransfermessung und das Auswerten des erfassten Messdaten eingegangen.

Kapitel 5 analysiert die Ergebnisse der beiden Entwürfe des HAW-SPI IPs und vergleicht das jeweilige Timingverhalten und den Ressourcenbedarf.

In **Kapitel 6** wird auf den Einsatz des IR-Sensors im Einparkassistenten des SoC-Fahrzeug eingegangen. Es wird ein Software-Modul für die Suche einer Parklücke entworfen und eine Methode für die Positionierung des SoC-Fahrzeuges beim Start des Einparkvorgangs vorgestellt.

Die Ergebnisse dieser Arbeit werden in **Kapitel 7** zusammengefasst.

2. Technologieübersicht zum SoC-Fahrzeug

In diesem Abschnitt wird das System, das im Rahmen dieser Arbeit erprobt wird, vorgestellt(vgl Abbildung 2.1). Als Zielhardware wird das Nexys2 Entwicklungsboard mit dem Xillinx Spartan 3E FPGA gewählt. Als Mikrocontroller wird der Softcore Prozessor MicroBlaze der Firma Xillinx eingesetzt. Der Infrarotsensor GP2D12 wird als Messsensor verwendet. Für die Umsetzung der, von den IR-Sensoren erfassten, analogen Messsignalen in digitale Werte kommt der Analog Digital Umsetzer AD1 zum Einsatz.

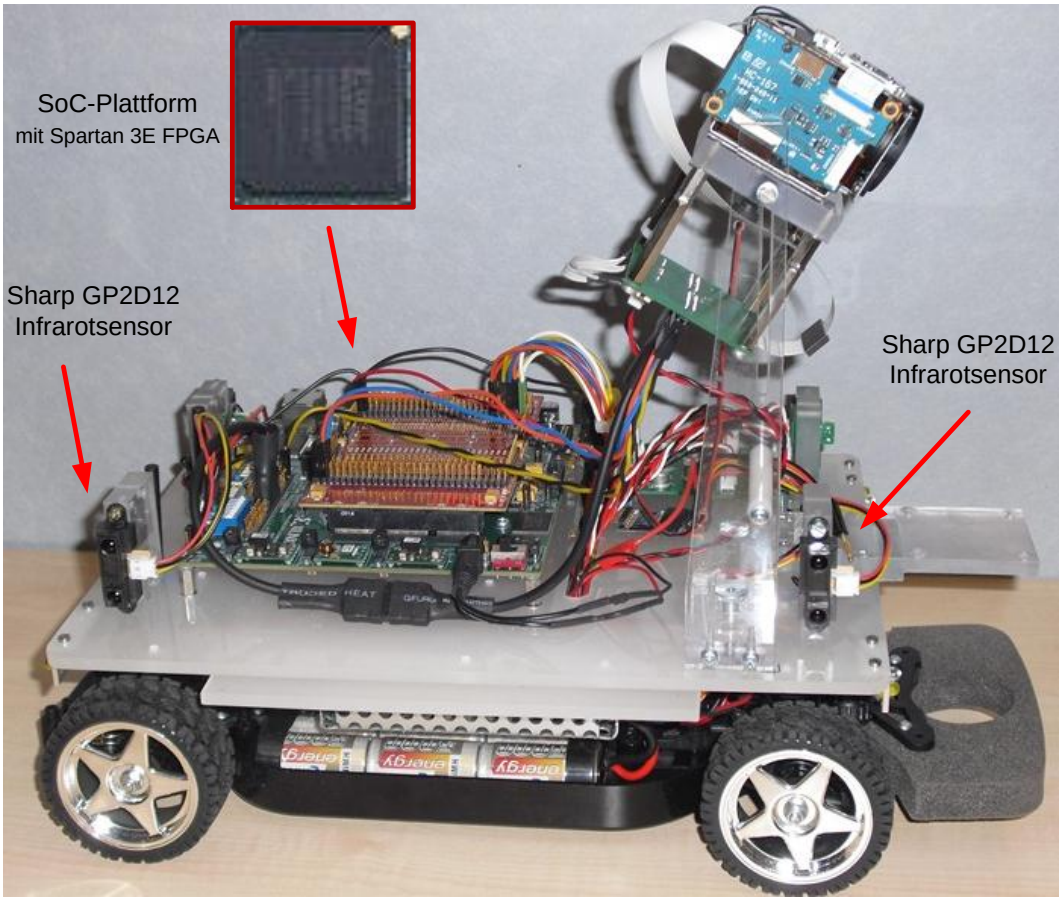

Abb. 2.1.: *SoC-Fahrzeug mit GP2D12 Infrarotsensoren angeschlossen an das Nexys2 Entwicklung-Board mit Spartan 3e-1200FG320 von Xillinx als SoC-Plattform*

2.1. SoC-Plattform

Unter SoC versteht man die Integration aller oder eines großen Teils der Funktionen eines Systems auf einem Chip, also einem Integrierter Schaltkreis(integratd cricuit) (IC). Als System wird dabei eine Kombination unterschiedlicher Elemente (logische Schaltungen, Taktgebung, selbstständiges Anlaufen, Sensoren, usw.) aufgefasst, die zusammen eine bestimmte Funktionalität bereitstellen. SoCs werden üblicherweise in eingebetteten Systemen eingesetzt. Während Systeme anfänglich aus einem Mikroprozessor oder Mikrocontroller-IC und vielen anderen ICs für spezielle Funktionen bestanden, die auf einer Platine aufgelötet waren, lässt die heute mögliche Integrationsdichte die Realisierung möglichst aller Funktionen auf einem einzigen IC zu. Mit einer SoC-Plattform können komplexe SW/HW Systeme auf einem Chip realisiert werden. SoCs werden nicht komplett neu entwickelt, sondern basieren auf den Entwürfen von (Intellectual Property (IP)-Cores. Die IP-Cores können vom Chip-Hersteller stammen oder von Entwicklern selbst entwickelt werden (Custom IP) und bei Bedarf über Parameter dem Verwendungszweck angepasst generiert werden[18]. Die IP-Cores sind über verschiedene Bussysteme am Prozessor angeschlossen (vgl. Abbildung 2.3).

2.1.1. Nexys2 Board

Für die SoC-Plattform wurde das Nexys2 Board der Firma Digilent[4] als Zielhardware gewählt. Das Nexys 2 Board stellt einen Spartan 3E FPGA und zusätzliche Peripherie wie, externe Spei-

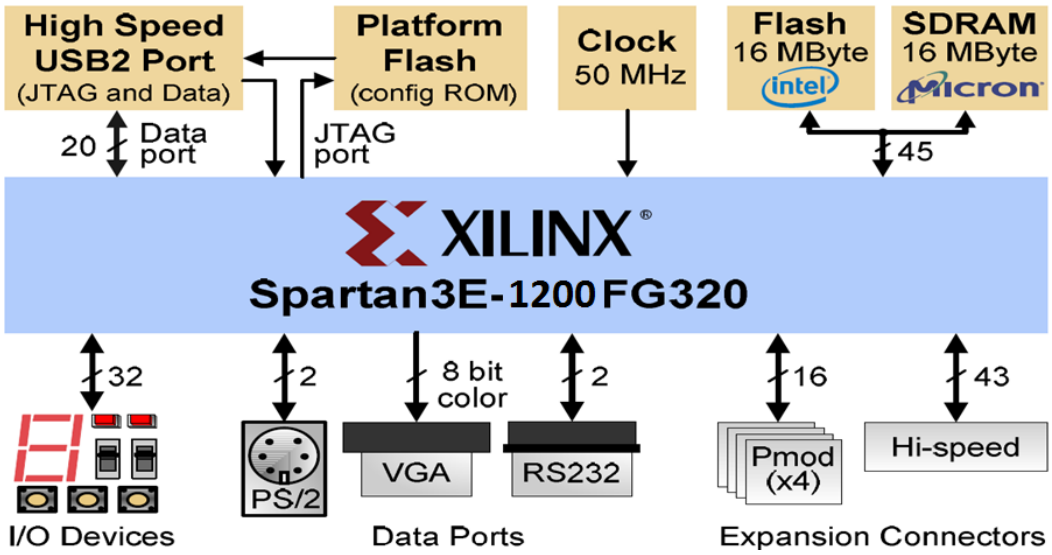

Abb. 2.2.: *Nexys2 Board mit Xillinx Spartan 3e [4]*

cher und I/0 Schnittstellen, zur Verfügung. Das Entwicklungsboard besteht aus den folgenden Komponenten(vgl. Abbildung 2.2) :

- Xillinx Spartan 3e XC3S1200E FPGA
- USB2-Schnittstelle
- 50 MHz Osillator

- 16MB Flash Read Only Memory (ROM) und 16 MB Synchronous Dynamic Random Access Memory (SDRAM)
- 8 Schiebeschlater,4 Pushbuttons,7-Segmentanzeige mit 4 Ziffern, 8 LEDs
- PS2 Schnittstelle
- VGA-Schnittstelle
- RS232 Serielle Schnittstelle
- 4 X 12 Bin PMod
- 40 FPGA I/O(FX2 Connector)

Die 4 IR-Sensoren sind über die 2 PModAD1-Boards an Pmod-Schnittstellen des Nexys2-Boards mit dem SoC-System verbunden(vgl. Abbildung 3.1,2.1). In dem Testfall können die erfassten Messdaten(Abstand zwischen dem SoS-Fahrzeug und einem Hindernis) sowie andere Rechenergebnisse über die serielle Schnittstelle (RS232) per UART auf einen PC übertragen werden.

2.1.2. Spartan3E FPGA

Das Xillinx Spartan-3E XC3s1200E FPGA (vgl. Abbildung. 2.2) bietet die folgenden Ressourcen [22] :

- **Configurable Logic Block (CLB):** Ein CLB besteht aus 4 mit einander verbundenen Slices.
- **Slices:** Eine Slice enthält 4-Input LUTs , 2 D-FF und Cray-und Kontrolllogik.
- **Lookup Table (LUT):** Eine LUT ist ein RAM-basierter Funktionsgenerator. Sie wird für die Umsetzung der logischen Funktionen benutzt, in dem die vorberechneten Werte in einer Wahrheitstabelle eingetragen werden.
- **Input/Out Block (IOB):** Die IOBs sind Kontrollblöcke zwischen den I/O Pins und den CLBs. Der IOB ist eine Programmierbare uni-oder bidirektionale Schnittstelle.
- **Dedicated Multiplier (DM):** Der DM berechnet das Produkt aus 2 Faktoren(bis 18 Bit Breite) und liefert ein 36 Bit breites Ergebnis.
- **Block RAM (BRAM):** Der BRAM ist ein synchroner RAM und bietet Datenspeicherung in Form von 18 KBit Dual Port Blöcken.
- **Digital Clock Manager (DCM):** Der DCM kontrolliert die Taktfrequenz.

ELCs	**LUT**s	**D-FF**s	**CLB**s	**Slices**	**DRAM** bits	**BRAM** bits	**DM**s	**DCM**s	**IOB**s
19512	17344	17344	2168	8672	136K	504K	28	8	250

Tabelle 2.1.: *Ressourcen des Xillinx Spartan-3E XC3S1200E FPGAs*

2.2. System on Chip Design

Die FPGA-basierte SoC-Technologie bietet durch die Bereitstellung der Prozessoren die Möglichkeit, das Hardware/Software-System auf einem Chip zu realisieren. In diesem System wird der MicroBlaze Softcore Prozessor, der auf allen FPGA-Modulen der Firma Xillinx implementiert werden kann, verwendet(vgl. Abbildung 2.3).

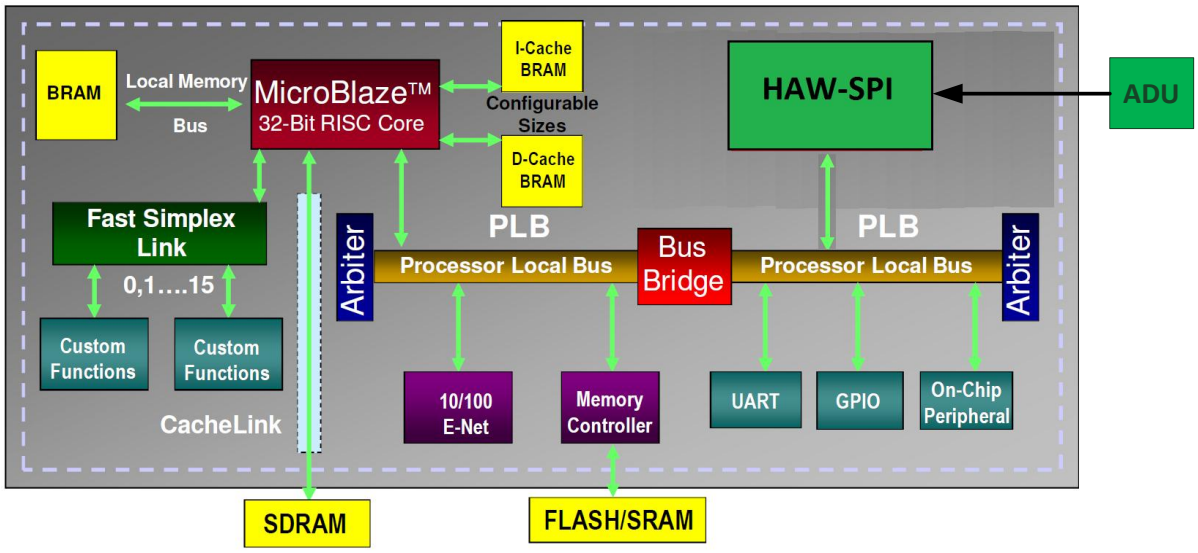

Abb. 2.3.: *Xillinx MicroBalze System Übersicht [20]*

Die Konfiguration des MicroBlaze-Systems bzw. die Einbindung der neu entwickelten oder vorgefertigten IPs erfolgt über das Embedded Development Kit (EDK). Das Xillinx EDK ist eine Entwicklungsumgebung für den Entwurf eingebetteter Systeme für SoC-FPGAs. Das EDK ermöglicht den Entwurf von Hardware- und Software-Komponenten.
Die nicht zeitkritischen Module einer Komponente werden in Software entwickelt, während die zeitkritische Module in Hardware modelliert werden.

Das EDK unterstützt alle Xillix Entwicklungsboards. Die Erzeugung eines IPs auf dem SoC erfolgt durch den Base System Builder (BSB)-Wizard des Xilinx Platform Studio (XPS)s anhand der Spezifikationen des Entwicklungsboards(vgl. Anhang C.1).

MicroBlaze

Der Xillinx MicroBlaze ist ein 32-Bit Softcore Prozessor mit Harvard Reduced Instruction Set Computer (RISC) Architektur. Im Gegensatz zu den im Chip integrierten nicht veränderbaren Hardcore Prozessoren, liegt der MicroBlaze Softcore Prozessor in Form von synthetisierbaren Netzlisten und HDL-Modulen vor und kann für den jeweiligen Einsatzzweck angepasst werden [21].

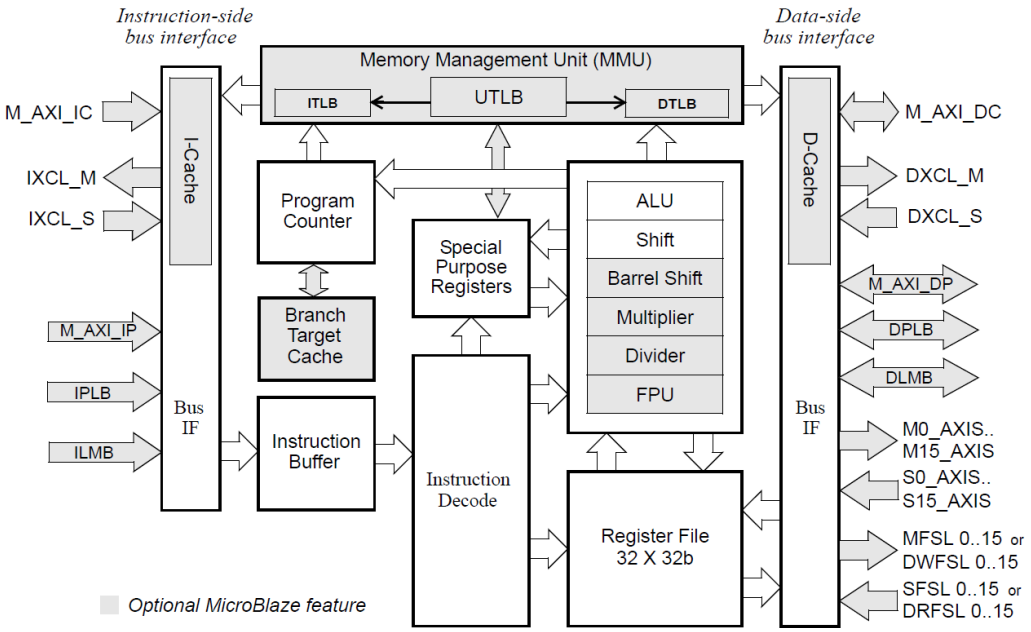

Abb. 2.4.: *Xillinx MicroBlaze Struktur [21]*

Als Betriebssystem für den MicroBlaze-Prozessor wird "standalone" deklariert(vgl. Anhang C.3). "standalone"-OS ist kein eigentliches Betriebssystem , sondern nur eine Ansammlung von Bibliotheken. Das "standalone"-OS wird benutzt, wenn lediglich ein natives C-Programm auf dem SoC laufen soll.

Prozessor Local Bus (PLB)

Der Processor Local Bus (PLB) ermöglicht eine bidirektionale Verbindung mit hoher Bandbreite(128, 64, 32-Bit) zwischen dem MicroBlaze Prozessor und der Peripherie, sowie zwischen zwei IP-Blöcken (vgl. Abbildung 2.4)[24]. Der PLB ist ein Multi-Master/Slave Bus. In der Konfiguration des XPS-Projekts dieser Arbeit wurde nur der MicroBlaze Prozessor als Master und alle anderen IP-Blöcke als Slave konfiguriert(vgl. Abbildung C.1).

Local Memory Bus (LMB)

Der LMB dient zu einer schnellen Kommunikation zwischen dem MicroBlaze Prozessor und dem BRAM.(vgl.Abbildungen 2.4,C.1). Über den LMB werden die getrennten Adressspeicher(ILMB) und Datenspeicher(DLMB) im BRAM mit dem MicroBlaze verbunden.

Interrupt Controller

Da der MicroBlaze nur einen Interrupt-Eingang besitzt und im SoS-System mehrere interruptfähige Komponenten integriert werden können, wird der Interrupt Controller(XPX_INTC) eingesetzt, um die Interrupt-Anfragen dieser Komponenten an den MicroBlaze Interrupt-Eingang anzuschließen. Der Interrupt-Controller verfügt über 32 Interrupt-Eingänge (n=32) und nur einen Interrupt-Ausgang (irq)[25]

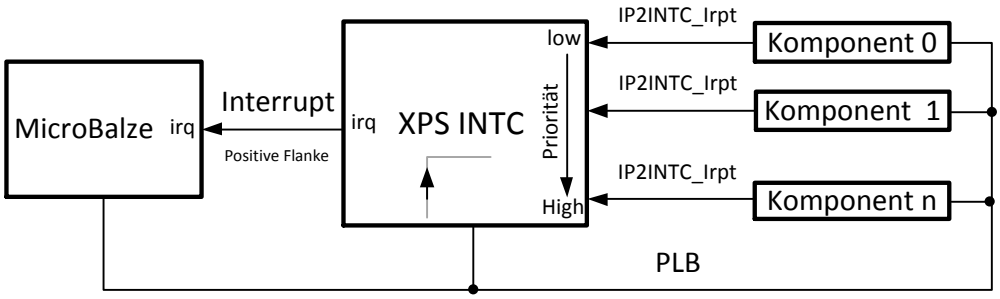

Abb. 2.5.: *XPS priorisierte Interrupt Controller mit bis zu 32 Interrupt-Eingängen und nur einem Interrupt-Ausgang zum MicroBlaze-Prozessor*

Wenn mehrere Interrupt Request (IRQ)s gleichzeitig auftreten wird der IRQ der Komponente mit der höchsten Priorität an den MicroBlaze weitergeleitet(vgl. Abbildung 2.5). Die Ordnung der Priorität ist vom XPS vorgegeben. Der Interrupt Controller ist mit dem MicroBlaze Prozessor sowie mit den anderen Komponenten über den PLB verbunden.

2.3. GP2D12 Infrarotsensor zur Abstandmessung

Der IR ist ein Infrarotsensor zur Abstandmessungen der Firma Sharp[13]. Der Sensor ist in der Lage Abstände zwischen 10cm und 80cm richtig zu erkennen. Die Werte werden durch analoge Spannung V_0 am Ausgang übermittelt. Der Sensor wird mit einer Spannung von 5 Volt V_{cc} versorgt (vgl. Abbildung 2.7).

Der IR besteht aus einem Infrarot-Sender/-Empfänger und arbeitet nach dem Triangulations-

Abb. 2.6.: *GP2D12 von Sharp*

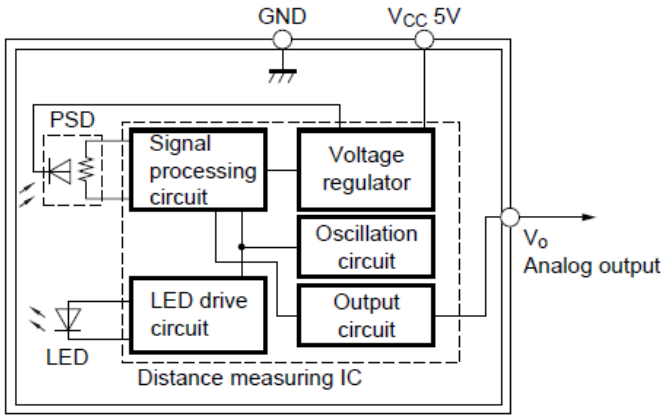

Abb. 2.7.: *Aufbau des GP2D12 IR Sensors V_0 im Intervall von 0.4 bis 2.7 Volt*

prinzip. Der Sender sendet einen modulierten Infrarotstrahl(LED) aus. Das von einem Objekt reflektierte Licht wird von einem sogenannten Position Sensitive Device (PSD) empfangen. Das zwischen Sender, Objekt und Empfänger entstandene Dreieck wir zur Bestimmung der Entfernung benutzt. D.h., nicht wie viel Licht von dem Objekt reflektiert wird, ist entscheidend, sondern in welchem Winkel es reflektiert wird. Dies hat den Vorteil, dass der Sensor relativ unabhängig von der Oberfläche des Objekt ist. Vom Empfänger wird dieser Winkel in analoge Spannung V_0 umgewandelt [3].

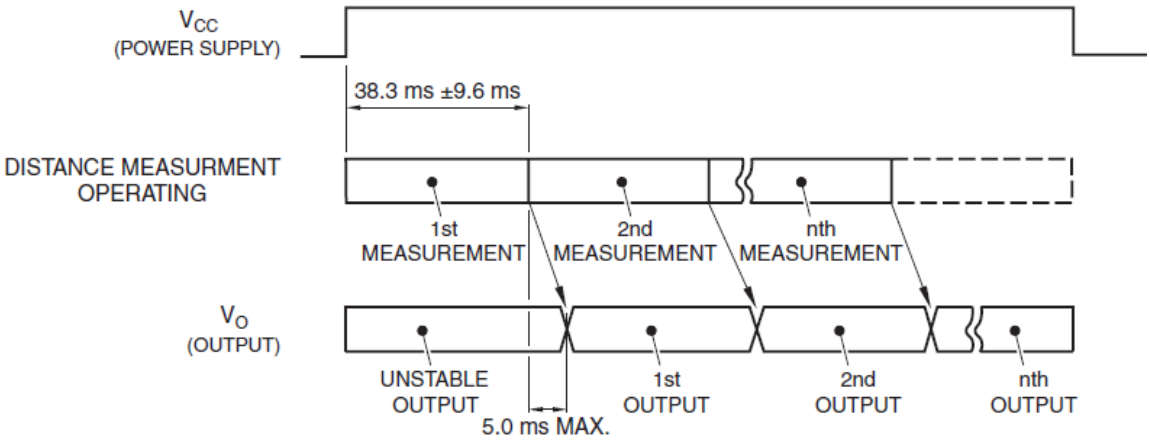

Abb. 2.8.: *Timing-Diagramm des GP2D12-Infrarotsensors;Typische Messzeit 38ms*

Durchschnittlich werden die Messwerte nach 38ms aktualisiert[13]. Die Frequenz des GP2D12-Sensors liegt bei:

$$f_{IR} = \frac{1}{0.038s} = 26.31Hz \tag{2.1}$$

Bei einem Abstand von 80cm liefert der Sensor eine Ausgangsspannung V_0 von 0.4 Volt und bei einem Abstand von 8cm 2.7 Volt [13]. Der Sensor hat eine Nicht-lineare Ausgangsspannung (vgl. Abbildung 2.9).

Die Nicht-Lineare Ausgangsspannung V_0 im Intervall von 0.4 Volt bis 2.7 Volt wird im Analog Digital Umsetzer (ADU) zu einem 12-Bit-Unsigned(M) im Intervall von 500 bis 3200 umgewandelt(vgl. Kapitel 2.4).

$$V_0 = f(L) \longrightarrow M = f(V_0) \tag{2.2}$$

L ist der tatsächliche Abstand zwischen dem Messobjekt und dem IR-Sensor(vgl. Abbildung 2.9). Der Messwert M wird über die Serielle Schnittstelle HAW-SPI in das SoC übertragen. In der HAW-SPI wird der Messwert in einem Polynom-Beschleuniger-Modul in eine in Zentimeter bzw. Millimeter messbare Größe (d) umgerechnet(vgl. Kapitel 3.2.2).

$$V_0 = f(L) \longrightarrow M = f(V_0) \longrightarrow d = f(M) \tag{2.3}$$

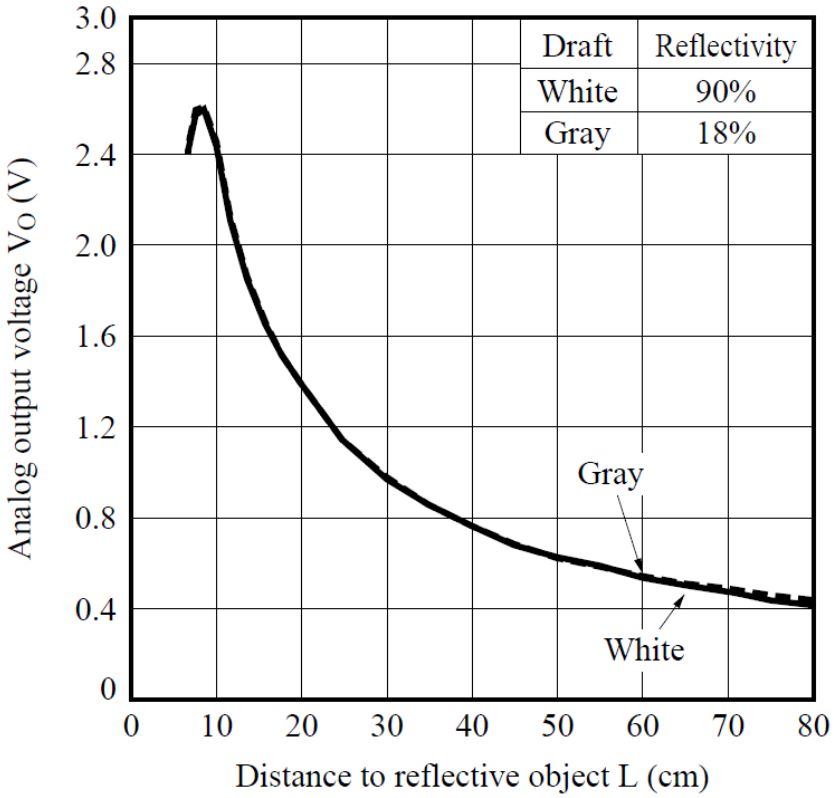

Abb. 2.9.: *Abhängigkeit der Ausgangsspannung V_0 zu dem Abstand zwischen GP2D12 IR und dem Messobjekt*

2.4. PmodAD1 ADU zur Umsetzung der analogen IR-Signale in digitale

Das PmodAd1-Board-Modul von Digilent besteht aus 2 ADUs vom Typ ADCS7476. die die zwei im Bereich 0 bis 3.3 Volt analogen Spannungsausgänge V_0 des IR-Sensors ("P1 und P2") in zwei digitale Werte ("P2 und P3) mit einer 12Bit-Auflösung $(0-(2^{12}-1))$ gleichzeitig umwandelt [8]. Das PmodAd1-Board wird mit einer Spannung V_{cc} von 3.3 Volt(P6) versorgt(vgl. Abbildung 2.11).

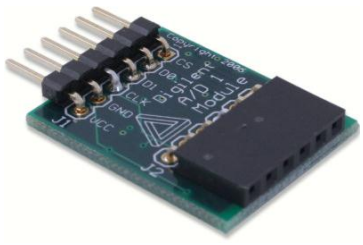

Abb. 2.10.: *PmodAD1 mit zwei integrierten ADUs vom Typ ADCS7476*

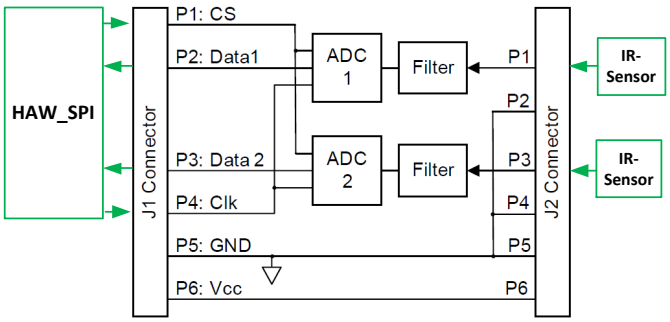

Abb. 2.11.: *Schaltungsdiagramm des PModAD1s*

Das PmodAD1-Board-Modul gibt der SoC-Plattform die Möglichkeit die mit dem IR-Sensor

erfassten Messdaten über die HAW-SPI-Schnittstelle abzulesen.
Die Steuerung der ADUs erfolgt über die HAW-SPI-Schnittstelle, in dem der Pin Chip Select("CS") des ADU an Slave Select (SS) der HAW-SPI-Schnittstelle und der Pin "CLK" des ADU an den Pin "SCK" der HAW-SPI-Schnittstelle angeschlossen werden.(vgl. Kapitel 3.2.1). Die umgewandelten Daten ("Data1 und Data2") werden über MISO-Signalleitungen der SPI-Schnittstelle parallel mit dem HAW-SPI Master-Modul serialisiert. Für eine 12-Bit Datenübertragung benötigt der ADU 16 "SCK"-Takte, wobei mit den 4 ersten Takten die 4 Nullen (ZERO3-0) und mit 12 weiteren Takten die 12-Bit-umgewandelten Daten mit dem Most Significant Bit (MSB)-Bit beginnend übertragen werden(vgl. Abbildung 2.12).

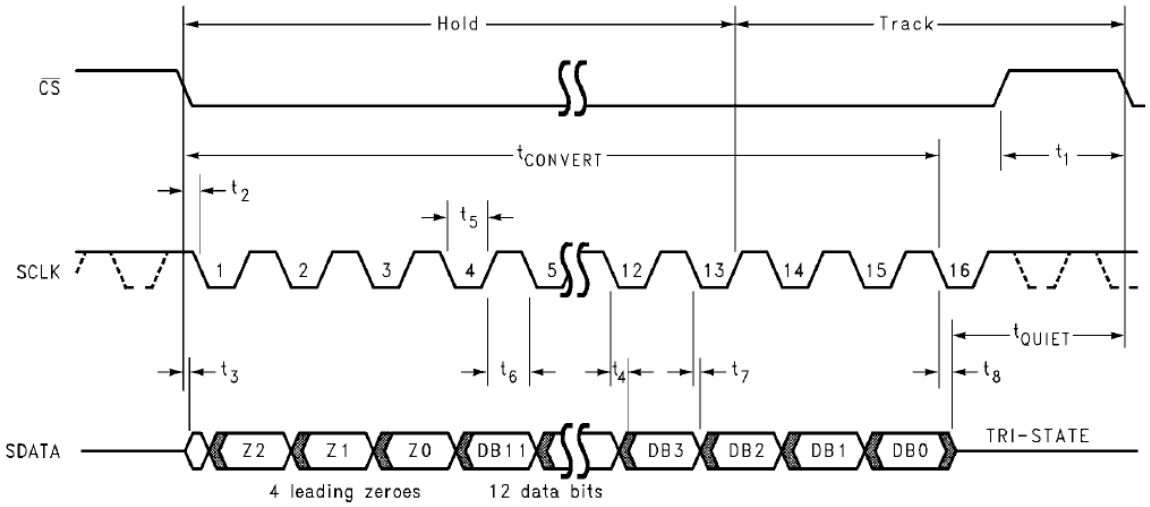

Abb. 2.12.: *Timing-Diagramm der seriellen Schnittstelle des ADCS7476(ADU)*

Die Übertragung beginnt, wenn das Low-Aktive Signal $\overline{CS}$ den Wert '1' und das "SCK"-Signal eine Fallende Flanke hat [11].
Für eine erfolgreiche Master In Slave Out (MISO)-Datenübertragung sollen das Master Device(HAW-SPI) und das Slave Device(ADU) die gleiche Konfiguration haben. D.h. das SCK-Signal sowie das SS-Signal des HAW-SPI IP sollen im Ruhezustand den Wert '1' haben(vgl. Kapitel 2.5).

Die Timing-Verhalten des AD1s bei einer 16-Bit-Datenübertagung ist wie folgt:

f_{sck}**:** Minimum 10 KHz , Maximum 20 MHz

$t_{convert}$**:** 16 x SCK-Periodendauer T_{sck}

t_{quit} minimal erforderliche Ruhezeit bis zur nächsten Umwandlung (50 ns)

t1: minimale $\overline{CS}$ Pulsbreite (10 ns)

t2: minimale Verzögerung zwischen $\overline{CS}$ und SCK (10 ns)

t3: maximale Verzögerungszeit bis SDATA den Tri-State Zustand verlässt (20 ns)

t4: maximale Datenzugriffszeit nach der fallenden SCK-Flanke (20 ns)

t5: minimale fallende SCK-Taktbreite(0.4 x T_{sck})

t6: minimale steigende SCK-Taktbreite(0.4 x T_{sck})

t7: minimale SCK-Zeit für die Datengültigkeit (10 ns)

t8: Zeit für eine fallende Flanke bis zum hochohmigen Zustand des SDATAs(min. 10 ns , max. 25 ns)

2.5. Standard SPI

Das SPI ist ein Bus-System zur synchronen, seriellen Datenübertagung zwischen einem Master und einem oder mehreren Slaves. Die Übertragung funktioniert im Vollduplex. D.h. die Daten fließen in beide Richtungen gleichzeitig. Der Master erzeugt Takt-und Steuersignale und startet eine Datenübertagung indem er einen Slave auswählt. Jeder Slave wird vom Master über eigene Leitung SS angesprochen und ist nur dann aktiv, wenn er ausgewählt wurde [12].
Der SPI-Bus besteht aus zwei Signalleitungen und zwei unidirektionalen Datenleitungen:

- **Slave Select ($\overline{SS}$):** Jeder Slave wird vom Master über eine eigene SS-Leitung angesprochen. Der Master wählt einen bestimmten Slave aus, indem er die entsprechende "SS" Leitung auf '0' setzt.
- **Signal Clock (SCK):** Der Takt wird vom Master erzeugt und dient dazu, alle angeschlossenen Geräte zu synchronisieren.
- **MISO:** Auf der MISO-Leitung überträgt der Slave Daten zum Master.
- **Master Out Slave In (MOSI):** Auf der MOSI-Leitung überträgt der Master Daten zum Slave.

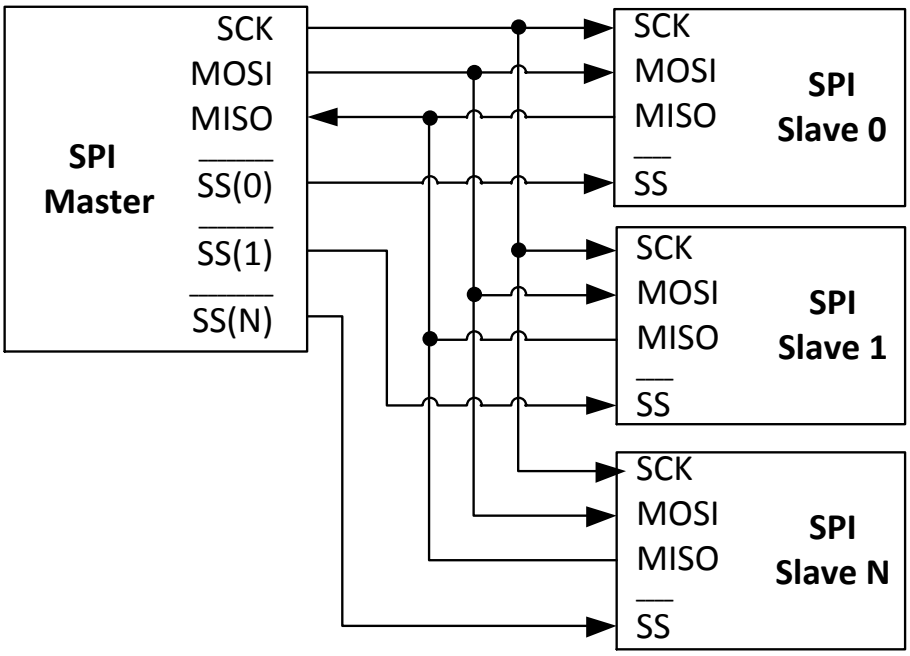

Abb. 2.13.: *Beispiel eines SPI-Bus-Aufbaus mit einem Master und mehreren Slaves in einer Sterntopologie*

Das SPI-Bus-Protokoll kennt zwei Parameter zur Einstellung des SCK-Taktes: die Polarität(CPOL) und die Phase(CPHA). COPL bestimmt, ob das Taktsignal(SCK) im Ruhezustand

auf dem LOW-Pegel(CPOL=0) oder dem HIGH-Pegel(CPOL=1) liegt. CPHA bestimmt, bei welcher Flanke des Taktsignals die Daten übertragen werden. CPHA=0 bedeutet, dass die Daten mit der ersten Flanke übernommen werden, CPHA=1 hingegen, dass sie mit der zweiten Flanke übernommen werden.
Aus der Kombination der Parameter CPOL und CPHA ergeben sich vier verschiedene Übertragungs-Modi(vgl. Tabelle 2.2).

SPI-Modi	CPOL	CPHA	Datenübernahme
0	0	0	Steigende Flanke
1	0	1	Fallende Flanke
2	1	0	Fallende Flanke
3	1	1	Steigende Flanke

Tabelle 2.2.: *SPI-Datenübertragungs-Modi für die Parameter CPOL und CPHA*

Für eine korrekte Datenübertagung ist es wichtig, dass diese Parameter in allen mit dem Bus verbundenen Geräten gleich eingestellt sind [12].

Verlauf der Datenübertagung mit "CPHA=0" :
Wird CPHA auf '0' gesetzt, werden Daten mit der ersten Flanke des SCK-Taktes übernommen. Die CPOL bestimmt, ob es sich dabei um eine steigende oder eine fallende Flanke handelt. Bei "CPOL=0" ist der Takt im Ruhezustand auf dem LOW-Pegel; die erste Flanke ist also eine steigende Flanke. Bei " CPOL=1" ist der SCK-Takt im Ruhezustand auf dem HIGH-Pegel; die erste Flanke ist also eine fallende Flanke(vgl. Abbildung 2.14).
Die Polarität des Taktes beeinflusst jedoch nicht den Zeitpunkt der Datenübernahme und damit auch nicht den Verlauf der Datenübertragung[12].

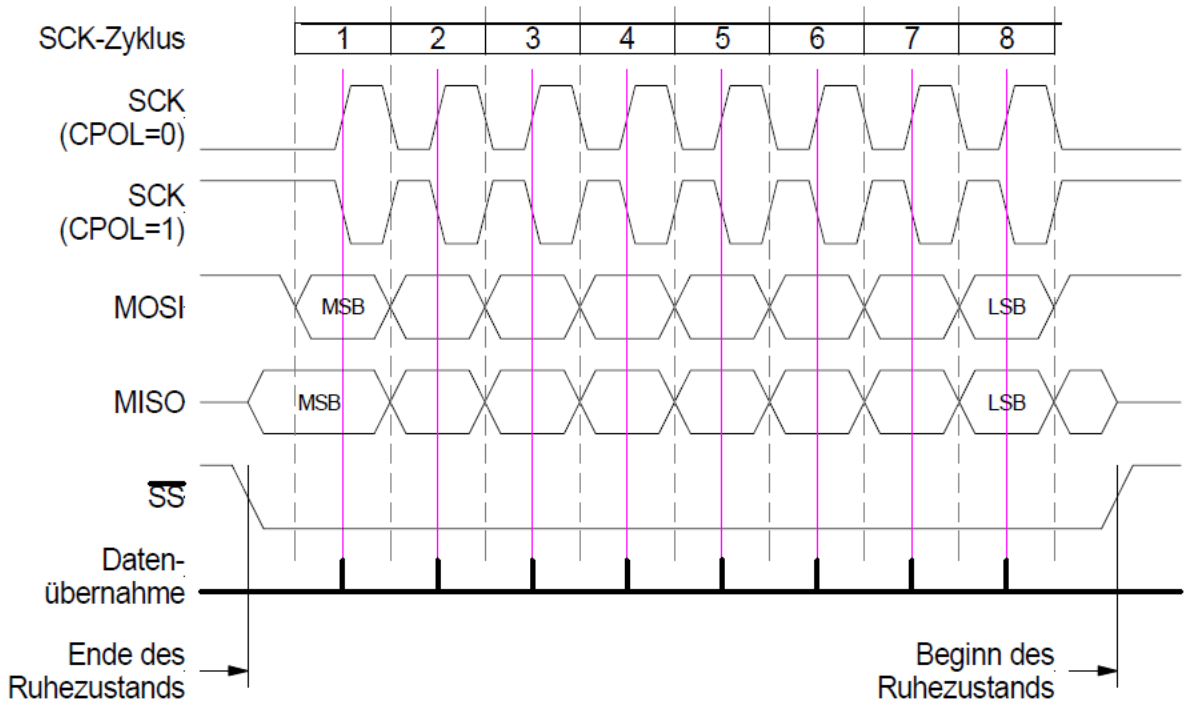

Abb. 2.14.: *Beispiel einer 8-Bit SPI-Datenübertagung für SPHA=0 [12]*

Verlauf der Datenübertagung mit "CPHA=1" :
Wird CPHA auf '1' gesetzt, werden die Daten mit der zweiten Flanke des Taktsignals (SCK) übernommen. Die CPOL bestimmt, ob es sich dabei um eine steigende oder eine fallende Flanke handelt. Bei "CPOL=0" ist der SCK-Takt im Ruhezustand auf dem LOW-Pegel und steigt nach der ersten Flanke auf den HIGH-Pegel; die zweite Flanke ist also eine fallende Flanke. Bei "CPOL=1" ist der SCK-Takt im Ruhezustand auf dem HIGH-Pegel und fällt nach der ersten Flanke auf den LOW-Pegel; die zweite Flanke ist also eine steigende Flanke. Die Polarität des Taktes beeinflusst jedoch nicht den Zeitpunkt der Datenübernahme und damit auch nicht den Verlauf der Datenübertagung(vgl. Abbildung 2.15).

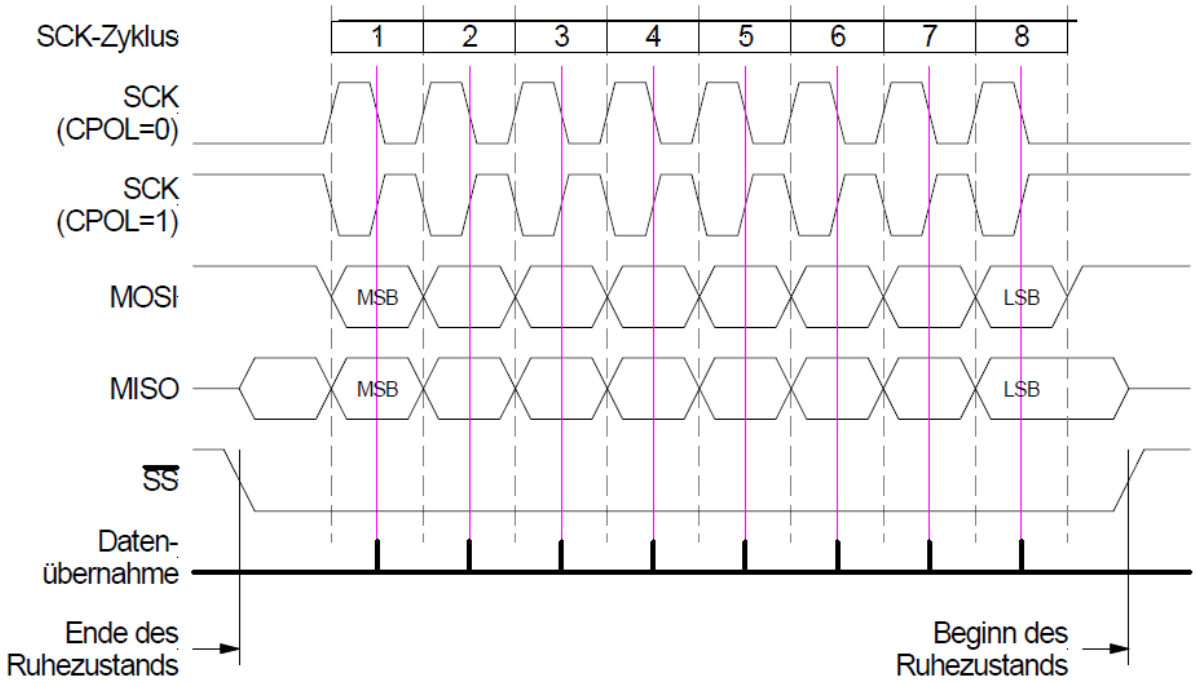

Abb. 2.15.: *Beispiel einer 8-Bit SPI-Datenübertagung für SPHA=1 [12]*

Das HAW-SPI Master-Modul wird mit dem SPI-Modi 3 (CPOL=1 und CPHA=0) konfiguriert, damit alle im Bus verbundene Geräte (ADU und SPI-Master-Modul) die gleiche Konfiguration haben(vgl. Kapitel 2.4).

3. Analyse des HAW-SPI IP Cores

Der HAW-SPI IP Core wurde im Rahmen des Faust-Projektes[7] an der HAW-Hamburg entwickelt[6],[1].
Sinn und Zweck der Entwicklung des HAW_SPI IPs ist die Messung des Abstandes zwischen dem SoC-Fahrzeug und einem Hindernis unter Anwendung von vier Infrarotsensoren.
In diesem Kapitel wird der Aufbau, der Ablauf der Datenübertragung und das Timing-Verhalten der seriellen Schnittstelle(HAW-SPI) analysiert und ausgewertet, um einen besseren Entwurf zu finden.

- Der Ressourcenbedarf des IP-Cores soll reduziert werden, um den Platzbedarf und den Energieverbrauch zu senken ohne das System nachteilig zu beeinflussen.
- Die Beteiligungsdauer des Mikroprozessors des SoCs an der Messdatenübertragung der IR-Sensor soll minimiert werden.
- Es soll ein Steuerungsverfahren für die Master-Slave-Datenübertragung gefunden werden, welches dafür sorgt, dass die Messdaten erst nach einer Aktualisierung übertragen werden. Dies soll zu einer weiteren Minderung des Energieverbrauchs führen.

3.1. Kopplung des HAW-SPI IP-Cores mit dem GP2D12 IR-Sensor

Der HAW-SPI IP Core ermöglicht einen parallelen Empfang der Datenströme von den vier gleichartigen AD1 Slave Devices (vgl. Kapitel 2.4). Der HAW-SPI IP Core ist über zwei PmodAD1-Boards mit zwei integrierten ADUs gekoppelt. Jeder dieser ADUs ist wiederum mit einem IR-Sensor gekoppelt(vgl. Abbildung 3.1). Die erfassten Messdaten werden im gleichen

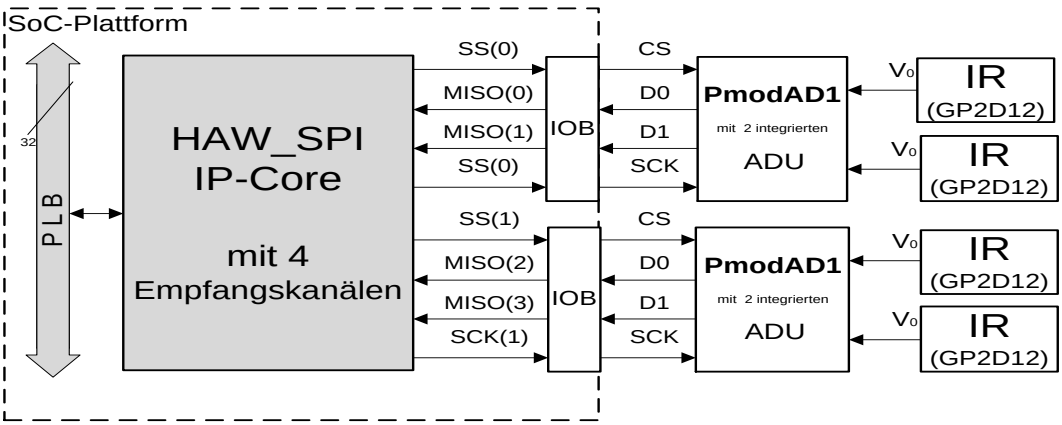

Abb. 3.1.: *Kopplung der GP2D12 Infrarotsensoren mit dem HAW-SPI IP-Core über die PModAD1 ADUs*

SPI Cock Ausgang (SCK)-Takt parallel an das Steuerungssystem des SoC-Fahrzeug übertragen (vgl. Abbildung 2.1). Der HAW-SPI IP verfügt über zwei SS-Ausgänge, mit denen alle ADU-Slave-Devices parallel selektiert werden. Über die zwei SCK-Ausgänge werden die vier ADUs parallel getaktet. Über die vier MISO-Eingänge werden die Datenströme aus den ADUs in die HAW-SPI-Master-Modul-Schieberegister transformiert(vgl. Abbildung 2.11).
Der Inhalt der Schieberegister des Master-Moduls wird vom Polynom-Modul in tatsächliche Abstände umgerechnet und das Ergebnis in Software-Register gespeichert(vgl. Abbildung 3.2).

3.2. Aufbau des HAW-SPI IPs mit vier Empfangskanälen

Der HAW-SPI IP Core besteht aus einem SPI-Master-Modul mit SPI Master-Funktionalitäten aus dem Xillinx XPS SPI[6], vier Polynum-Accelerator-Modulen, in denen die IR Messtaten in tatsächliche Abstände umgerechnet werden und 14 Software Registern(vgl. Abbildung 3.2).

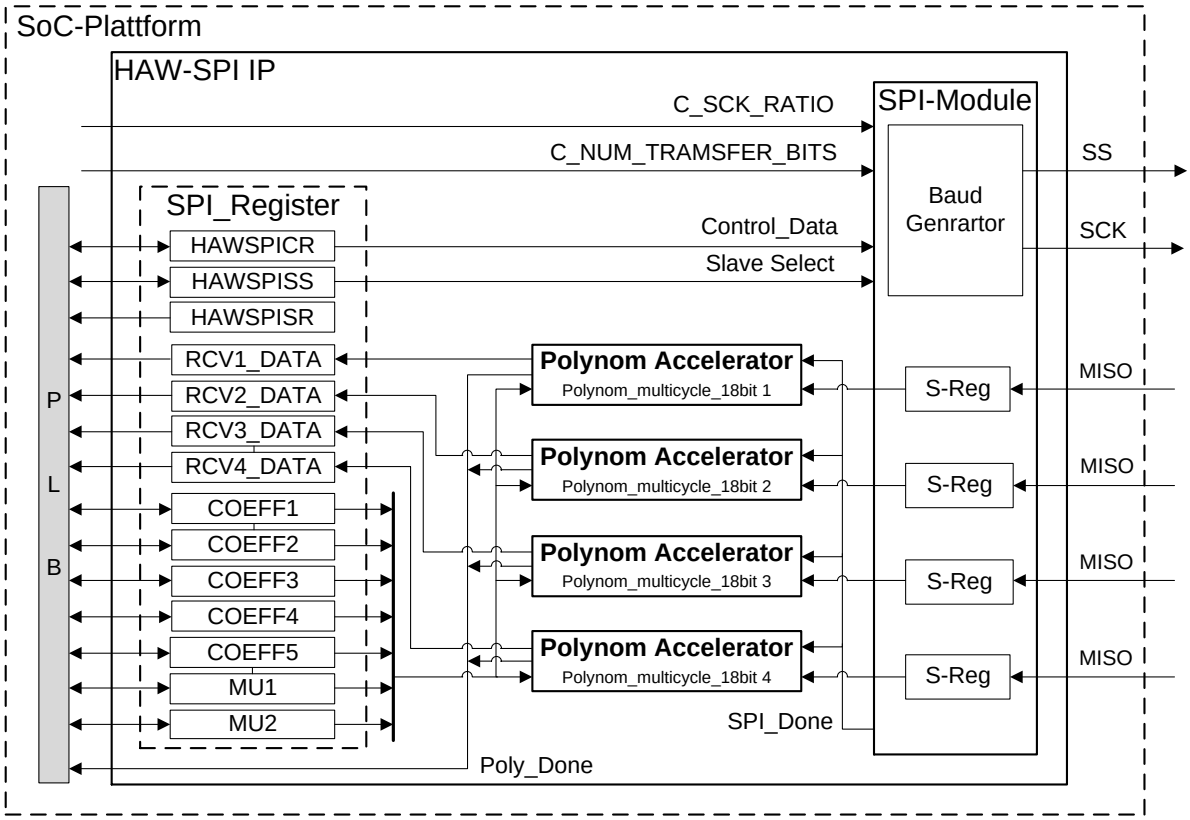

Abb. 3.2.: *Aufbau des HAW_SPI IPs mit vier Empfangskanälen, vier Polynom-Acceleratoren und einem Interrupt-Ausgang("Poly_Done"), verbunden mit einem PLB-BUS*

Über das Signal "SPI_Done ='1' informiert das SPI_Master-Modul die Polynom-Module, dass die 16-Bit MISO-Datenübertragung abgeschlossen ist und die Daten im Schieberegister "S-Reg" vorhanden sind. Die Schieberegister sind über 16-Bit Leitungen mit den Eingängen der "Polynom-Acceleratoren" verbunden. Die 18-Bit Ausgänge des Polynom-Accelerators sind an vier Softwareregister (RCV1_Data - RCV4_Data) angeschlossen. Außerdem existiert eine

direkte Verbindung zwischen den "S-Regs" und den Softwareregistern, die über einen Multiplexer mit Polynom Toggle Enable (PTE)-Bit aus dem "HAWSPICR" getoggelt werden.
Die Softwareregister(COEFF1-COEFF5, MU1 und MU2), mit denen die Koeffizienten und die Skalierungs-Parameter des Polynoms eingespeist werden, sind jeweils mit jedem Polynom-Accelerator verbunden.

3.2.1. Struktur, Schnittstellen und Funktionalität des HAW-SPI-Master-Moduls

Das Master-Modul unterstützt nur die MISO-Datenübertragung. Die Funktionalität des Moduls wurde von Xillinx XPS SPI übernommen und an den HAW-SPI IP CORE angepasst [6]. Dadurch wurden die erforderlichen Standards der SPI Transferformate vom HAW-SPI IP eingehalten. Der Eingang "CLK" entspricht dem System-Takt. Der SPI-Takt ("SCK") wird aus

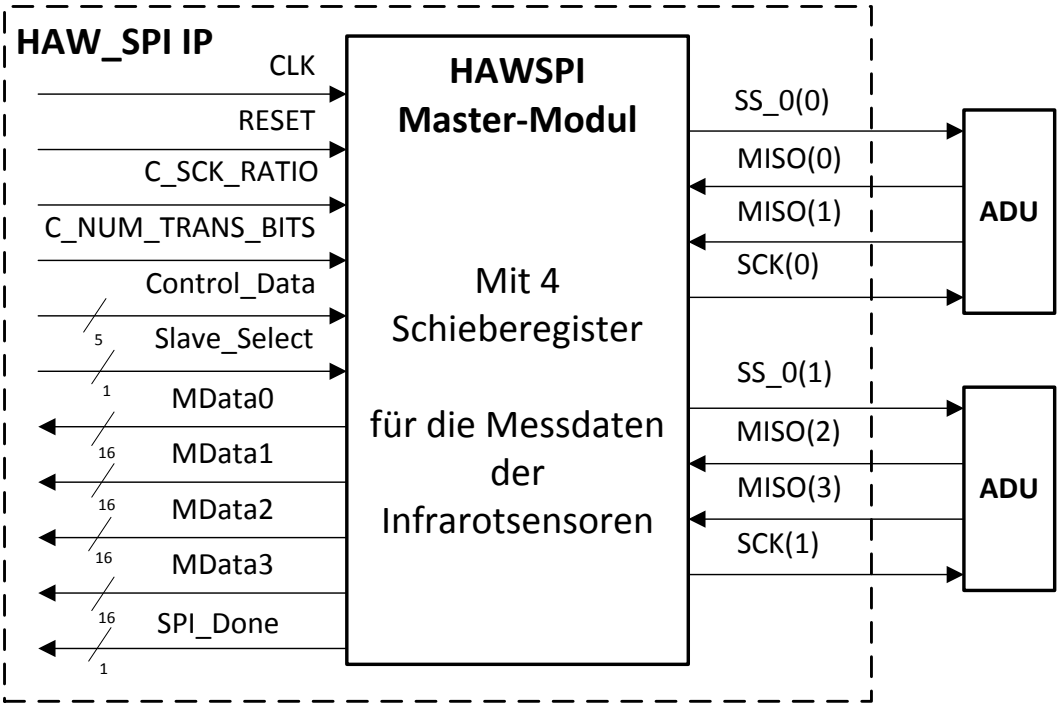

Abb. 3.3.: *Aufbau des HAW-SPI-Master-Moduls mit vier Empfangsschieberegistern(vgl. 3.4)*

dem System-Takt und dem "C_SCK_RATIO "-Signal für die synchrone Datenübertragung erzeugt.

$$f_{SCK} = \frac{F_{CLK}}{C\_SCK\_RATIO}, C\_SCK\_RATIO = [4 : 2048] \tag{3.1}$$

$$T_{SCK} = T_{CLK} \times C\_SCK\_RATIO \tag{3.2}$$

Da der ADU mit einer maximalen Frequenz von 20 MHz arbeitet, ist das C_SCK_RATIO bei einer Systemfrequenz von 50 MHz im unterem Intervall auf den Wert 4 beschränkt(vgl. Gleichung 3.1). Mit dem "SCK" werden die vier Schieberegister parallel zu den ADUs getaktet, um die 16-Bit Daten aus den ADU-Schieberegistern in die SPI-Schieberegister im gleichen Takt zu übertragen. Sowohl die Anzahl der SCK-Takte für eine MISO-Datenübertragung als auch die Bit-Breite der SPI-Schieberegister werden durch den "C_NUM_TRANSFER_BITS"-Eingang bestimmt.

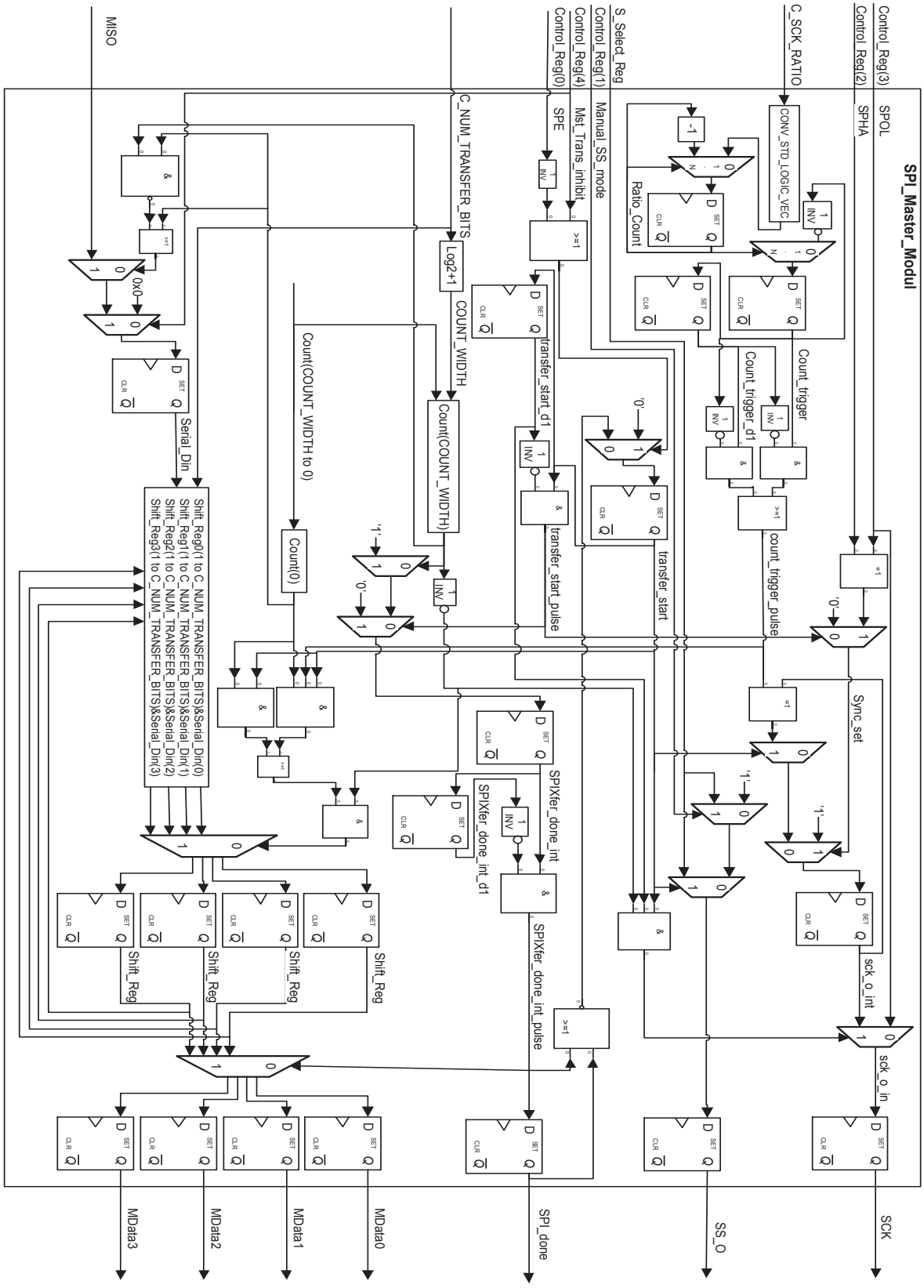

Abb. 3.4.: *Struktur des HAW-SPI-Master-Moduls mit vier Empfangsschieberegistern*

Der "C_NUM_TRANSFER_BITS"-Eingang kann mit den Werten 8, 16 oder 32 parametriert werden. Der Wert 16 entspricht der Bit-Breite des ADUs. Eine MISO-Datenübertragung fängt, je nachdem wie das HAW-SPI-Modul (CPOL,SPHA)[1] konfiguriert ist, mit dem MSB-Bit des ADUs an(vgl. Kapitel 2.4). Das MSB-Bit des ADU-Registers wird beim ersten Takt an die Lost Significant Bit (LSB)-Stelle des SPI-Schieberegisters verschoben(vgl. Listing 3.1).

SCK	0 to 15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	SData	MSB	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	LSB
1	S_Reg																MSB
2	S_Reg															MSB	B14
3	S_Reg														MSB	B14	B13
4	S_Reg													MSB	B14	B13	B12
5	S_Reg												MSB	B14	B13	B12	B11
6	S_Reg											MSB	B14	B13	B12	B11	B10
7	S_Reg										MSB	B14	B13	B12	B11	B10	B9
8	S_Reg									MSB	B14	B13	B12	B11	B10	B9	B8
9	S_Reg								MSB	B14	B13	B12	B11	B10	B9	B8	B7
10	S_Reg							MSB	B14	B13	B12	B11	B10	B9	B8	B7	B6
11	S_Reg						MSB	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5
12	S_Reg					MSB	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4
13	S_Reg				MSB	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3
14	S_Reg			MSB	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2
15	S_Reg		MSB	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
16	S_Reg	MSB	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	LSB
	MData	MSB	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	LSB

Tabelle 3.1.: *MISO-Datenübertragung der 16-Bit-Daten(SData) aus dem ADU in die SPI-Schieberegister(S_Reg)*

Bei einer erfolgreichen Datenübertragung stehen nach 16 "SCK"-Takten das MSB-Bit der ADU-Schieberegister(SData) an der MSB-Stelle des HAW-SPI-Schiebereiters(S_Reg). Nach dem 16. "SCK"-Takt stehen die Daten(MData) am Ausgang des HAW-SPI-Master-Moduls zur Verfügung(vgl. Tabelle 3.1). Das Schiebeverfahren erfolgt in allen vier Schieberegistern gleichzeitig, wobei "Serial_Din(0-3)" den Ausgängen der vier ADUs (MISO(0-3)) entsprechen.

```
1  Shift_Reg  <= Shift_Reg (1 to C_NUM_TRANSFER_BITS -1) & Serial_Din(0);
2  Shift_Reg1 <= Shift_Reg1(1 to C_NUM_TRANSFER_BITS -1) & Serial_Din(1);
3  Shift_Reg2 <= Shift_Reg2(1 to C_NUM_TRANSFER_BITS -1) & Serial_Din(2);
4  Shift_Reg3 <= Shift_Reg3(1 to C_NUM_TRANSFER_BITS -1) & Serial_Din(3);
```

Listing 3.1: *Code-Abschnitt für vier Schieberegister aus dem SPI-Master-Modul*

Die Ausgänge (Receive_Data0-3) entsprechen den Schieberegistern (Shift_Reg0-3) aus dem Codeabschnitt 3.1. Das Signal "SPI_Done hat den Wert '1' sobald das LSB-Bit aus den ADUs in die Schieberegister übertragen worden ist. Somit stehen die Messdaten aller vier IR-Sensoren zum Lesen über die Leitungen ("Receive_Data0-4") zur Verfügung. Die vier MSB-Bits (Z3-Z0 aus MData) des Schieberegisters sind Zero-Bits, weil der IR-Sensor nur 12 Bit-Daten erfasst. Der Eingang "Slave-Select " wird an die beiden Ausgänge "SS_0(0)"'und "SS_O=(1)" weitergeleitet. Der Eingang " Controller_Data besteht aus Steuerungsdaten, die den fünf LSB-Bits des HAWSPICRs entsprechen(vgl. Abbildung3.5)

SPE : Das SPI System Enable (SPE)-Bit hat folgende Auswirkungen auf das HAW-SPI-Master-Modul:
SPE = '0' : das HAWSPI System ist deaktiviert; MISO-Eingänge werden ignoriert; es wird kein SCK-Takt generiert.

[1]Die Auswirkung der SPOL und SPHA auf die Messdaten werden in Kapitel 4.5 anhand der Messdaten und der Timing-Bilder veranschaulicht

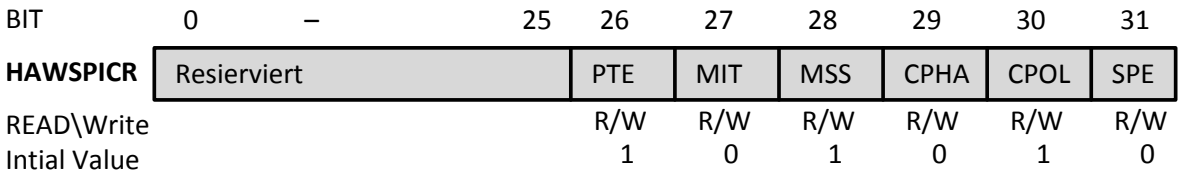

Abb. 3.5.: *HAWSPI-Controll-Register (HAWSPICR)*

	MSB-Bit				LSB-Bit
Control_Data	MIT	MSS	SPHA	CPOL	SPE

Tabelle 3.2.: *fünf Steuerungsbits aus dem HAWSPICR-Register (vgl. Abbildung 3.5) für die Steuerung des HAWSPI-Master-Moduls*

SPE = '1' : Das HAW-SPI System ist aktiviert; SCK ist im IDLE-Zustand; Es werden SCK-Takte generiert, sobald das MTI-Bit den Wert '0' hat.

CPOL: Das Clock Polarity (CPOL)-Bit bestimmt den IDLE-Zustand des SCKs:
CPOL= '0': SCK ist im IDLE-Zustand LOW.
CPOL= '1': SCK ist im IDLE-Zustand HIGH (vgl. Kapitel 2.5).

CPHA: Das Clock Phase (CPHA)-Bit bestimmt das Übertragungsformat bzw. die Übernahme der Transferdaten aus der seriellen Schnittstelle(vgl. Kapitel 2.5):
CPHA = '0': Daten sind gültig mit der ersten SCK-Flanke.
CPHA = '1': Daten sind gültig mit der zweiten SCK-Flanke.

MSS: Mit dem Manual Slave Select Assertion Enable (MSS)-Bit wird zwischen manueller und automatischer Konfiguration des SS-Signales getoggelt:
MSS = '0': Das HAWSPI-Master-Modul toggelt automatisch. Am Anfang einer MISO-Übertragung weist das SS-Bit(LSB-Bit) des HAWSPISS-Registers (vgl. Abbildung 3.2) direkt auf den SS-Ausgang des HAWSPI-Master-Moduls. Das SS-Bit soll den Wert '0' haben(ADU low aktiv). Am Ende der Datenübertragung setzt das HAWSPI-Master-Modul das SS-Signal auf '1', ohne das SS-Bit des HAWSPISS-Registers zu ändern. Wenn ein neue Übertragung beginnt, wird das SS-Bit des HAWSPISS-Registers erneut dem SS-Ausgang zugewiesen(vgl. Anhang 3.4). Diese Konfiguration bietet einen kontinuierlichen Empfang der Daten. Für einen neuen Transfer muss das MTI-Bit oder das SPE-Bit getoggelt werden[6].
MSS = '1': Der SS-Ausgang des HAWSPI-Master-Moduls (vgl. Abbildung 3.3) muss von der Anwendung getoggelt werden. Für jeden MISO-Übertragungsstart muss eine '0' ins HAWSPISS-Register geschrieben werden und am Ende der Übertragung eine '1' (vgl. Anhang B.2).

MTI: Das Master Transaction Inhibit (MTI)-Bit bestimmt, ob es sich um eine Master Transaktion oder eine Slave Transaktion handelt:
MIT = '0' : Das HAWSPI-Master-Modul generiert den SCK-Takt und taktet damit die ADUs. MIT = '1' : Die Master Transaktion ist deaktiviert.

3.2.2. Polynomapproximation der Sensorkennlinie

Aus dem, mit dem IR, erfassten Abstand liefert der AD1 einen 12-Bit Messwert(M) im Bereich 0 bis 4095, wobei nur Werte von 500 bis 3200 relevant sind(vgl. Kapital 2.4). Um eine Zielfunktion zu entwickeln, die aus den erfassten Messwerten(M) im Bereich von 8 cm bis 80 cm die tatsächlichen Abstände berechnet, werden mehrere Abstandsmessungen(dM) zwischen dem Messobjekt und dem GP2D12 IR-Sensor benötigt [13]. Dazu wird das Messobjekt in unterschiedlichen Distanzen im Intervall von 8 cm bis 80 cm zum IR-Sensor positioniert und der jeweilige Messwert(M) erfasst(vgl. Tabelle **??**).
Nun sind die X- und Y-Werte der Zielfunktion vorhanden. Daraus werden die Koeffizienten der Funktion mit der Matlab-Funktion "polyfit" berechnet. Mit $c = polyfit(M, dM, n)$ werden die Koeffizienten c_i ermittelt, wobei n+1 die Anzahl der Koeffizienten ist[10]. Mit $[c, S, mu] = polyfit(M, dM, n)$ werden die Koeffizienten c_i berechnet und M wird skaliert und zentriert. Die Variable Mu besteht aus $\mu 1$ und $\mu 2$, wobei $\mu 1$ der Mittelwert und $\mu 2$ die Standartabweichung von M ist. Mit dem oben vorgestellten Verfahren werden die Koeffizienten der Polynome 3. Ordnung, 4.Ordnung und 5.Ordnung ermittelt. Durch die Skalierung

Polynom Ordnung	c_1	c_2	c_3	c_4	c_5	c_6
3 ohne Skallierung	$-8.99 * 10^{-9}$	$6.43 * 10^{-5}$	-0.15	137.168		
3 mit Skallierung	-7.336	17.213	-13.788	18.343		
4 ohne Skallierung	$5.07 * 10^{-12}$	$-4.75 * 10^{-8}$	$1.64 * 10^{-4}$	-0.2549	170.152	
4 mit Skallierung	3.870	-11.370	10.105	-10.068	19.952	
5 ohne Skallierung	$-3.17 * 10^{-15}$	$3.52 * 10^{-11}$	$-1.54 * 10^{-7}$	$3.36 * 10^{-4}$	-0.38	202.23
5 mit Skallierung	-2.255	6.897	-6.420	5.426	-12.584	20.958

Tabelle 3.3.: *Vergleich der Koeffizienten der Polynome 3.Ordnung, 4.Ordnung und 5.Ordnung mit und ohne Skalierung*

werden Koeffizienten berechnet, die sich leichter binär darstellen lassen(vgl. Tabelle 3.3), was zu einer Einsparung von Hardwareressourcen führt. Um das geeignetste Polynom zu finden, werden die 3 Polynome mit den skalierten Koeffizienten auf die jeweiligen Abweichungen der gemessenen Abstände(dM) zu den berechneten Abständen(p(M)) untersucht.

Das Polynom n. Ordnung lässt sich wie folgt darstellen:

$$p(x) = c_1 x^n + c_2 x^{n-1} .... + c_n x^x c_{n+1} \tag{3.3}$$

Die skalierte Variable $\widehat{x}$ lässt sich wie folgt darstellen[10]:

$$\widehat{x} = \frac{x - \mu 1}{\mu 2} \tag{3.4}$$

Die Variable x aus der Gleichung 3.4 wird zu M und die Variable $\widehat{x}$ zu m umbenannt.

$$m = \frac{M - \mu 1}{\mu 2} = \frac{M - 1652.64}{934.39} \tag{3.5}$$

Mit der Matlab-Funktion $p(m) = polyval(c, m)$ werden die Abstände p(m) berechnet, wobei m die skalierten Messwerte(M) und c die Koeffizienten sind(vgl. Tabelle 3.4).

dM in cm	M	m	p(m) 3.Ord in cm	Abweichung in cm	p(m) 4.Ord in cm	Abweichung in cm	p(m) 5.Ord in cm	Abweichung in cm
8	3360	1,827	5,8631	2,1369	9,0761	1,0761	7,8712	0,1288
9	3275	1,736	7,8948	1,1052	8,5976	0,4024	8,9647	0,0353
10	3050	1,496	11,6831	1,6831	8,8277	1,1723	10,4333	0,4333
11	2842	1,272	13,5517	2,5517	10,2218	0,7782	11,0624	0,0624
12	2643	1,060	14,3309	2,3309	11,9802	0,0198	11,7597	0,2403
13	2468	0,873	14,5436	1,5436	13,5510	0,5510	12,7012	0,2988
14	2312	0,706	14,6067	0,6067	14,8442	0,8442	13,8398	0,1602
15	2195	0,580	14,7043	0,2957	15,7288	0,7288	14,8608	0,1392
16	2073	0,450	14,9557	1,0443	16,5913	0,5913	16,0516	0,0516
17	1973	0,343	15,3432	1,6568	17,2833	0,2833	17,1075	0,1075
18	1864	0,2262	16,0197	1,9803	18,0702	0,0702	18,3318	0,3318
19	1794	0,1513	16,6254	2,3746	18,6228	0,3772	19,1599	0,1599
20	1700	0,0507	17,6871	2,3129	19,4661	0,5339	20,3336	0,3336
25	1396	-0,2736	23,5536	1,4464	23,7172	1,2828	24,9809	0,0191
30	1201	-0,4834	29,8572	0,1428	28,6742	1,3258	29,4695	0,5305
35	1042	-0,6535	36,7525	1,7525	34,7264	0,2736	34,8184	0,1816
40	936	-0,7670	42,3526	2,3526	40,0861	0,0861	39,6829	0,3171
45	848	-0,8611	47,6656	2,6656	45,5041	0,5041	44,7792	0,2208
50	775	-0,9393	52,5583	2,5583	50,7568	0,7568	49,9016	0,0984
55	704	-1,0153	57,7604	2,7604	56,5986	1,5986	55,8053	0,8053
60	655	-1,0677	61,6158	1,6158	61,0886	1,0886	60,4855	0,4855
65	606	-1,1201	65,6951	0,6951	65,9801	0,9801	65,7193	0,7193
70	574	-1,1544	68,4831	1,5169	69,4030	0,5970	69,4622	0,5378
75	531	-1,2004	72,3871	2,6129	74,3008	0,6992	74,9289	0,0711
80	498	-1,2357	75,5084	4,4916	78,3015	1,6985	79,4874	0,5126
Max				4,4916		1,6985		0,8053
Mittel				1,8493		0,7328		0,2793
Min				0,1428		0,0198		0,0191

Tabelle 3.4.: *dM = tatsächlicher Abstand; M = erfasste Messwerte; m = skalierte Messwerte; p(m) X.Ord = mit dem jeweiligen Polynom berechneter Abstand; Abweichung = Abweichung von p(m) zu dM*

Die maximale Abweichung des Polynomen 3. Ordnung beträgt 4.5 cm, was viel zu groß und damit nicht für die Zielfunktion geeignet ist. Die maximale Abweichung des Polynoms 4. Ordnung ist etwa 0,9 cm höher als die des Polynoms 5. Ordnung. Da aber die Realisierung des Polynoms 5. Ordnung unverhältnismäßig mehr Ressourcen und Laufzeit benötigt als das Polynom 4. Ordnung, wurde das Polynom 4. Ordnung für die Zielfunktion gewählt.

$$p(m) = c_1 m^4 + c_2 m^3 + c_3 m^2 + c_4 m + c_5 \tag{3.6}$$

Das Polynom-Accelerator-Modul(Abbildung 3.6) berechnet aus einem ADU-gewandelten Mess-

wert(M) des IR-Sensors (GP2D12) mit der Gleichungen 3.6 und 3.5 einen tatsächlichen Abstandswert(p)[1]

Das Eingangsignal "spi_done" startet das Modul sobald ein neuer Messwert 'M' vorliegt(vgl. Abbildung 3.2). Am Eingang "rcv_data" liegen nur die 12 LSB-Bits des SPI-Schieberegisters "Receive_Data(4 to 15)" an, weil die 4 MSB-Bits des Registers Zero-Bits sind(vgl. Tabelle 3.1). Für die Skalierung des erfassten Messwertes "M" wird die Gleichung 3.5 wie folgt umgestellt:

$$m = \frac{M - \mu 1}{\mu 2} = (M - \mu 1) * \frac{1}{\mu 2} = (M - 1652.64) * \frac{1}{934.39} \tag{3.7}$$

Der skalierte Messwert 'm' wurde in der Gleichung 3.6 eingesetzt und als Register Transfer Level (RTL)-Modul moduliert [1]. Das Ergebnis "p(m)" steht nach 9 Takten am Ausgang [1]. Das Signal "Poly_Done" nimmt denn Wert '1' ein, sobald das Ergebnis am Ausgang zur Verfügung steht.

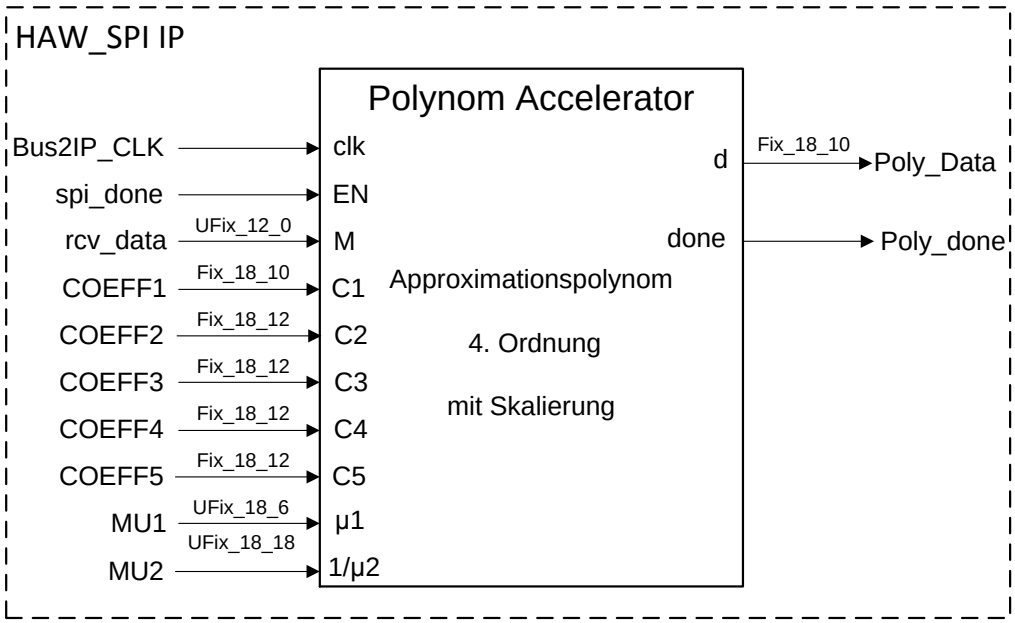

Abb. 3.6.: *Blackbox des Beschleunigermoduls zur Berechnung des erfassten Messwertes(M) zu tatsächlichem Abstand(p(m))*

Die Eingänge "Coeff1-5", "mu1 und mu2" (vgl. Tabelle 3.5), "M" und der Ausgang "Poly_Data " des Polynom-Accelerator-Moduls sind in "Fixed_Pont"- Darstellung(vgl. Kapitel **??**).

Die Koeffizienten(c_{1-5}) sowie die Skalierungsfaktoren ($\mu 1$ und $\mu 2$) werden zu Fixed-Point [9] umgerechnet(vgl. Tabelle 3.5) und bei der Initialisierung des HAW-SPIs in das SW-Register "COEFF 1-5 und MU1 und MU2 " geschrieben(vgl. Abbildung 3.2).

Die Fixed-point Schreibweise "Poly_Done_18_10" deutet darauf, dass der Bit-Vektor die Breite 18 hat, wobei die 8 MSBs Vorkommastellen sind. Das erste MSB ist das Vorzeichen. Die 10 LSBs sind für die Nachkommastellen(vgl. Tabelle 3.6). Die 7 Bit für die Vorkommastellen entsprechen dem Abstand zwischen einem Messobjekt und dem Infrarot-Sensor (GP2D12) in

Fixed-Point	Hex	Integer	Binär	Double
C1 (Fix_18_10)	0xF7B	3963	**0000001**1101111011	3,87
C2 (Fix_18_12)	0x34A16	215574	**110100**101000010110	-11,37
C3 (Fix_18_12)	0xA1B0	41392	**001010**000110110000	10,1
C4 (Fix_18_12)	0x35EEC	220908	**110101**111011101100	-10,07
C5 (Fix_18_12)	0x13F3B	81723	**010011**111100111011	19,95
MU1 (UFix_18_6)	0x19d29	105769	**011001110100**101001	1652,6
MU2(UFix_18_18)	0x119	281	000000000100011001	0,001070217

Tabelle 3.5.: *Koeffizienten und Skalierungsfaktor des Polynomes in Fixed-Point-Darstellung*

Poy_Data	Signed Bit	Vorkommastellen	Nachkommastellen
Fix_18_10	1 Bit	7 Bit	10 Bit
Wertebereich	0 oder 1	-128 bis 127	2^{-10} bis 1-2^{-10}

Tabelle 3.6.: *Fixed-Point Darstellung des berechneten Abstandes "Poly_Data" in cm und dessen Genauigkeit*

cm. Da der berechnete Abstand "p(m)" immer einen positiven Wert hat, werden mit den 7 Bit die Abstände von 0 cm bis 127 cm dargestellt.
Mit den 10 Bit Nachkommastellen werden 2^{-10} cm bis $1 - 2^{-10}$ cm dargestellt. Damit liegt die Genauigkeit des berechneten Abstands bei

$$\frac{1}{\frac{1-2^{-10}}{2^{-10}}} == \frac{1}{\frac{0.9990234375}{0.0009765625}} = \frac{1}{1023} cm = 9.775 \mu m \tag{3.8}$$

3.3. Timing-Verhalten des HAW-SPI IPs bei der MISO-Datenübertragung

In diesem Abschnitt wird der Ablauf des HAW-SPI IPs bei einer MISO-Datenübertragung und die Steuerung des IPs bzw. der Start und der Stopp einer Übertragung über das Schreiben bzw. Lesen der Softwareregister beschrieben. Um einen Datensatz aus einem ADU in das HAW-SPI-Schieberegister zu übertragen, gibt es zwei Möglichkeiten:

1. **Möglichkeit:** Das HAWSPICR-Register wird für eine automatische Übertragung konfiguriert(vgl. Abbildung 3.5). Bei dieser Art der Übertragung werden die 16-Bit Messdaten kontinuierlich aus dem ADU in das HAW-SPI-Schieberegister übertragen. Nach Abschluss der Übertragung wird ein Interrupt durch den HAW-SPI ausgelöst. Beim Empfang dieses Interrupts liest der MicroBlaze die Daten aus dem Softwareregister des HAW-SPI IPs aus. Diese Art der Übertragung macht wenig Sinn, weil dadurch auch veraltete Daten übertragen werden und somit unnötig Energie aufgewendet wird.
2. **Möglichkeit:** Die Abtastrate für die Übertragung wird mit einem externen Timer(XPS-Timer) bestimmt(vgl. Abbildung 3.7). Diese Möglichkeit wird in diesem Abschnitt veranschaulicht.

Der Ablauf eines 16-Bit-Datentransferzyklus aus dem ADU ins HAW-SPI-Schieberegister beginnt mit dem Auslösen eines Interrupts durch den externen Timer “timer_intr” und endet mit dem Auslösen des HAW-SPI-Modul-Interrupts “spi_intr”(vgl. Abbildung 3.7).

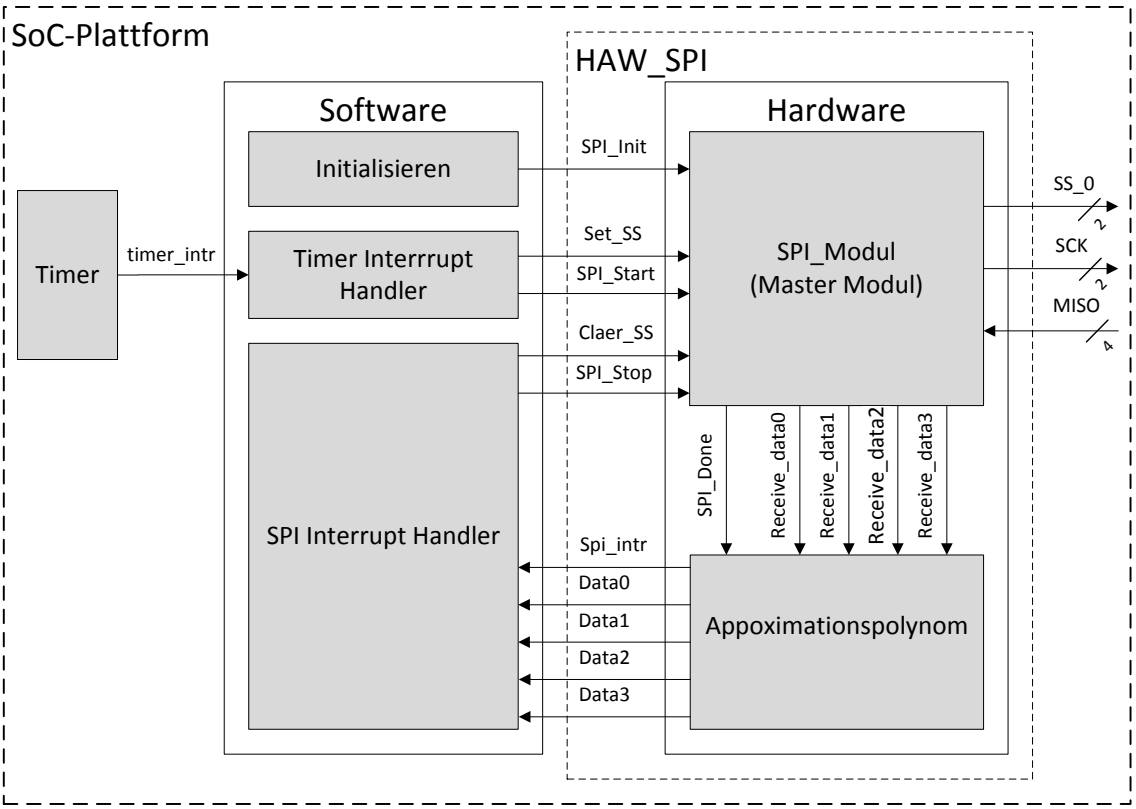

Abb. 3.7.: *Die Kombination der Software -und Hardware -Module für die Steuerung einer MISO-Datenübertragung aus dem ADU in SoC-Fahrzeug und die Bestimmung des Abtastrate der Übertragung durch einen Timer*

Ein 16-Bit-Datentransferzyklus geschieht in folgenden Schritten:

1. Ein externer Timer, der mit der Abtastrate der Übertragung konfiguriert ist, löst periodisch einen Interrupt aus. Der Interrupt wird in dem ISR " Timer Interrupt Handler" wie folgt behandelt: Mit der Funktion "Set_SS" wird der "SS_O"-Ausgang des SPI-Moduls aktiviert bzw. auf '0' gesetzt. Die Funktion " SPI_Start" sorgt dafür, dass das HAW-SPI-Master-Modul das Slave Device(ADU) taktet(vgl. Kapitel 3.2.1). Die Funktion " SPI_Start" operiert auf dem Register "HAWSPICR" und die Funktion "Set_SS" auf dem Register "HAWSPISS"(vgl. Abbildung 3.2).
2. Das HAW-SPI-Master-_Modul schiebt den erfassten Messwert(MISO) in 16 Takten aus dem ADU in das SPI-Schieberegister und setzt anschließend das Ausgangssignal "SPI_Done" auf '1'. Die erfassten Daten("Receive_Data 0-3) stehen am Ausgang des HAW-SPI-Master-Moduls zur Verfügung. Die vier erfassten Datensätze werden parallel in vier Approximationspolynom-Module eingespeist, um die Werte in eine messbare Größe(cm) umzurechnen. Die Polynom-Module stellen die umgerechneten Ergebnisse (Data 0-3) am Ausgang bereit. Das Interrupt-Signal "spi_inter" entspricht dem Signal "SPI_Done", das durch die Approximation-Module weitergeleitet wird.
3. Nach der Umrechnung der erfassten Messdaten in messbare Werte (Data 0-3) lösen die Approximation-Module einen Interrupt (SPI_Intr) aus. Der Interrupt wird von der ISR (" SPI Interrupt Handler") gefangen und wie folgt behandelt: Das Erzeugen des "SCK"-Taktes wird gestoppt("SPI_Stopp"). Anschließend wird das "SS_O" deaktiviert("clear_ss) und die Abstände(Data 0-3) werden aus den Softwareregistern ausgelesen.

Alle oben genannten Lese- und Schreibvorgänge erfolgen über einen 32-Bit PLB Schreib- bzw. Lesezugriff.

Das Vorgehen eines Datentransferzyklus wurde mit dem Oszilloskop(DPO400) gemessen. Diese Messdaten wurden mit ModelSim simuliert, um das Timing-Verhalten aller Komponenten des HAW-SPI darzustellen(vlg. Abbildung 3.8). Das Timing-Verhalten eines Datentransferzyklus

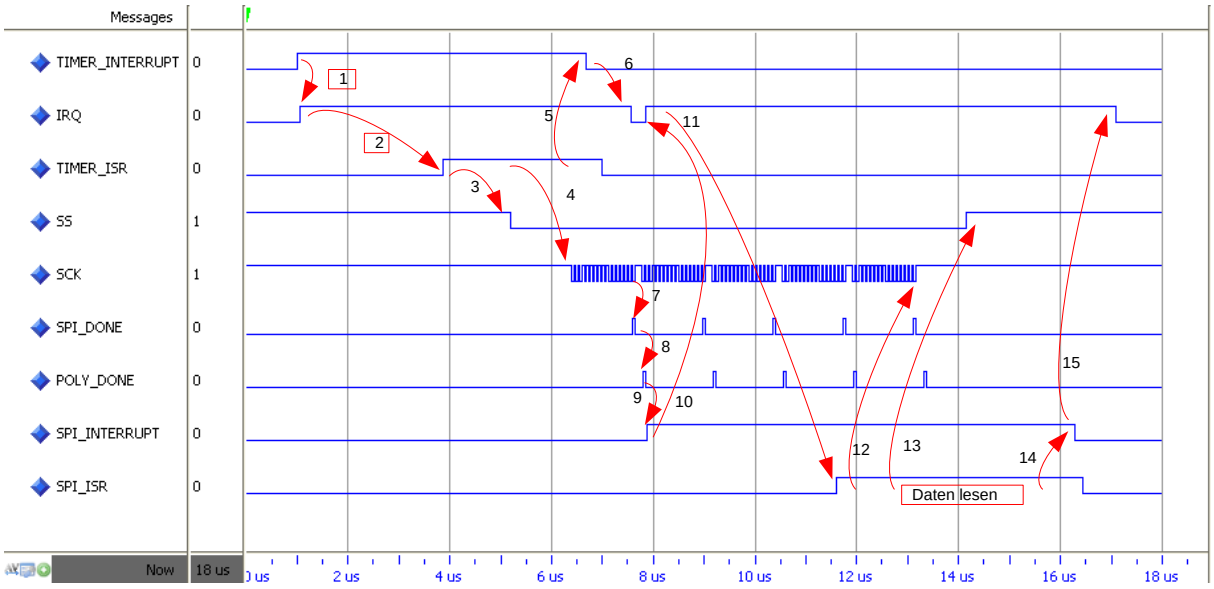

Abb. 3.8.: *Simulation des Timing-Verhaltens eines Datentransferzyklus des HAW-SPI*

der vom IR-Sensor(GP2D12) erfassten Daten aus dem ADU-Schieberegister in das HAW-SPI-Schieberegister lässt sich folgendermaßen beschreiben(vgl. Abbildung 3.8):

1. Der Timer startet mit einer vordefinierten Abtastrate eine MISO-Datenübertragung aus dem ADU in das HAW-SPI-Schieberegister. Nach 3 Systemtakten(60 ns) trifft der Interrupt in den MicroBlaze Interrupt-Eingang ein.
2. Der MicroBlaze startet eine ISR (Timer Interrupt Handler), um den Timer-Interrupt zu behandeln bwz. die Übertragung durch das Schreiben des HAWSPICR-Registers zu starten.
3. In der ISR wird zuerst die Interrupt-Quelle geprüft. Dann wird mit der Funktion "Set_ss" der Ausgang "SS_0" des Slave Device(ADU) selektiert. Die Funktion "Set_ss" erfolgt über zwei PLB-Zugriffe. Daher die Verzögerung.
4. Die Funktion " Start_Spi"(zwei PLB-Zugriffe) wird ausgeführt, um im HAW-Master-Modul SCK-Signale zu erzeugen und damit den ADU zu takten.
5. Die Interrupt-Quelle wird zurückgesetzt, damit die Interrupts anderer Komponenten an den MicroBlaze durch den Interrupt-Controller weitergeleitet werden können.
6. Der Interrupt-Eingang des MicroBlaze wird gelöscht. Der Prozessor steht für weitere Interrupts zur Verfügung.
7. Nach der Erzeugung des SCK-Signals erfolgen 16 "SCK"-Takte, um die 16-Bit-Daten aus dem ADU in das HAW-SPI Schieberegister zu übertragen. Nach Abschluss der Übertragung zeigt das "SPI_Done"-Signal an, dass die Übertragung erfolgte und die Messdaten verfügbar sind.
8. Nachdem "SPI_Done = '1' ist, erfolgt die parallele Umrechnung der vier erfassten Messdaten in die tatsächlichen Abstände durch vier Approximationpolynom-Module in 180 ns(20ns * 9 Takte). Anschließend wird "Poly_Done" auf '1' gesetzt.
9. Der SPI-Interrupt wird durch das "Poly_Done" ausgelöst. Die Messdaten stehen nun in SW-Registern des HAW-SPI IPs zur Verfügung und können vom MicroBlaze gelesen werden.
10. Der SPI-Interrupt trifft am MicroBlaze Interrupt-Eingang ein.
11. Ein Kontextwechsel findet statt. Der SPI-ISR (SPI Interrupt Handler) wird gestartet.
12. Das HAW-SPI-Modul stoppt das Erzeugen des "SCK"-Taktes("SPI_Stopp()").
13. Das "SS_O" wird auf '1' gesetzt(Slave Device nicht selektiert).
14. Nachdem alle vier Daten(Data 0-3) mit vier PLB-Zugriffen gelesen worden sind, wird die Interrupt-Quelle zurückgesetzt.
15. Der MicroBlaze Interrupt-Eingang wird gelöscht.

Für einen Datenübertragzyklus ist die Interrupt-Quelle des MicroBlaze für eine Zeit von $15.74\mu s$ ($6.50\mu s$+$9.24\mu s$) besetzt. Innerhalb dieser Zeit können keine anderen Interrupts empfangen werden. Der Prozessor verbringt insgesamt $7.96\mu s$ seiner Zeit in den beiden ISRs(vgl. Abbildung 3.9).

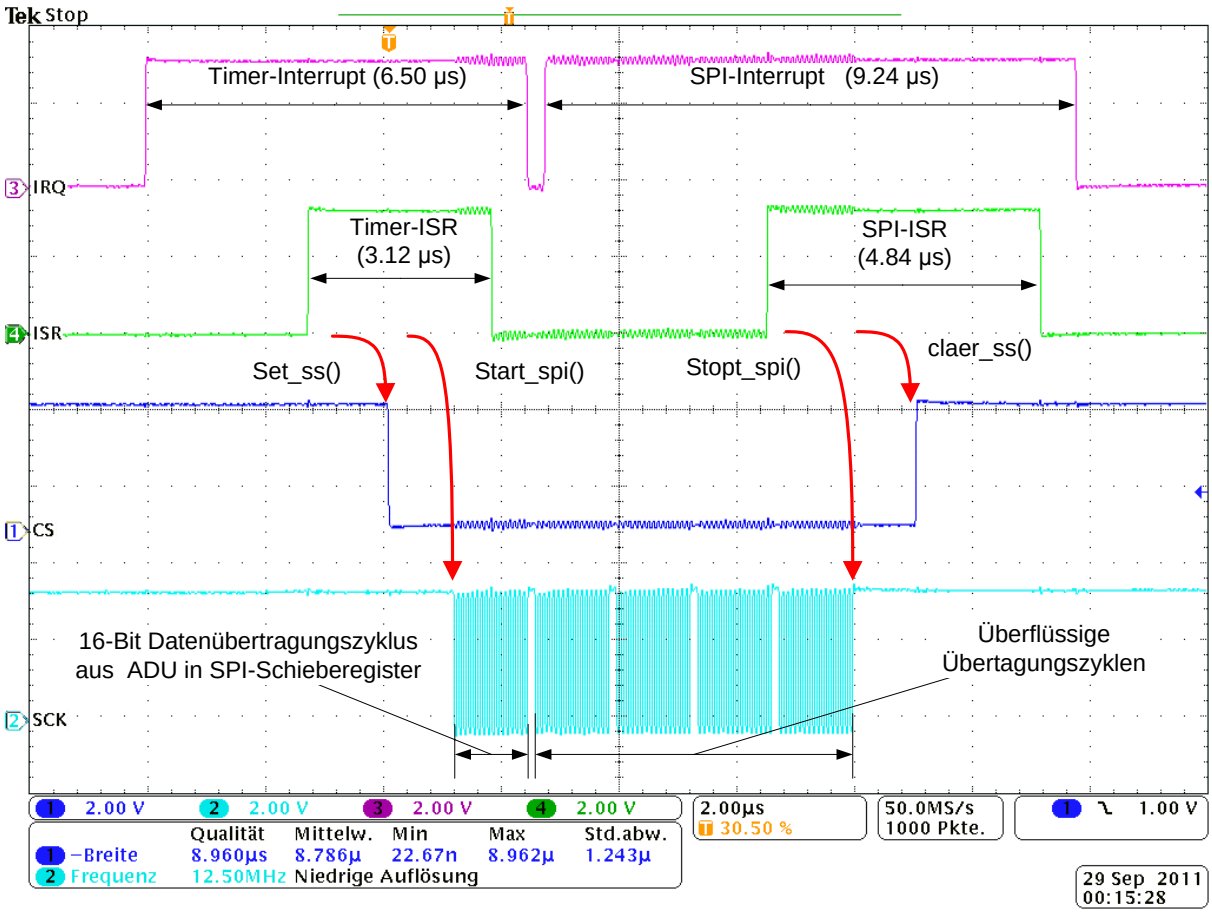

Abb. 3.9.: *Ablauf der Timer- und SPI-Interrupts und Dauer deren ISRs*

Die Anzahl der "SCK"-Takte für einen Übertragungszyklus entsprechen 16 Takten(vgl. Kapitel 3.2.1). Die erfassten Daten sind nach 16 Takten in den Schieberegistern des HAW-SPI-Master-Moduls. Somit sind die weiteren vier Übertragungszyklen(4 * 16 SCK-Takte) überflüssig. Diese lassen sich nicht vermeiden, da der Übertragungsprozess erst in der "SPI-ISR" gestoppt("Stop_spi()") wird(vgl. Abbildung 3.9). Für diese Messungen taktet der HAW-SPI-Bus mit einer Frequenz von 12.5 MHz ("C_SCK_RATIO = 4 ")(vgl. Gleichung 3.1). Die Periodendauer(T) des "SCK"-Taktes entspricht 80ns ($T_{SCK} = T_{sys} * C\_SCK\_RATIO = 20ns * 4 = 80ns$) (vgl. Abbildung 3.10).

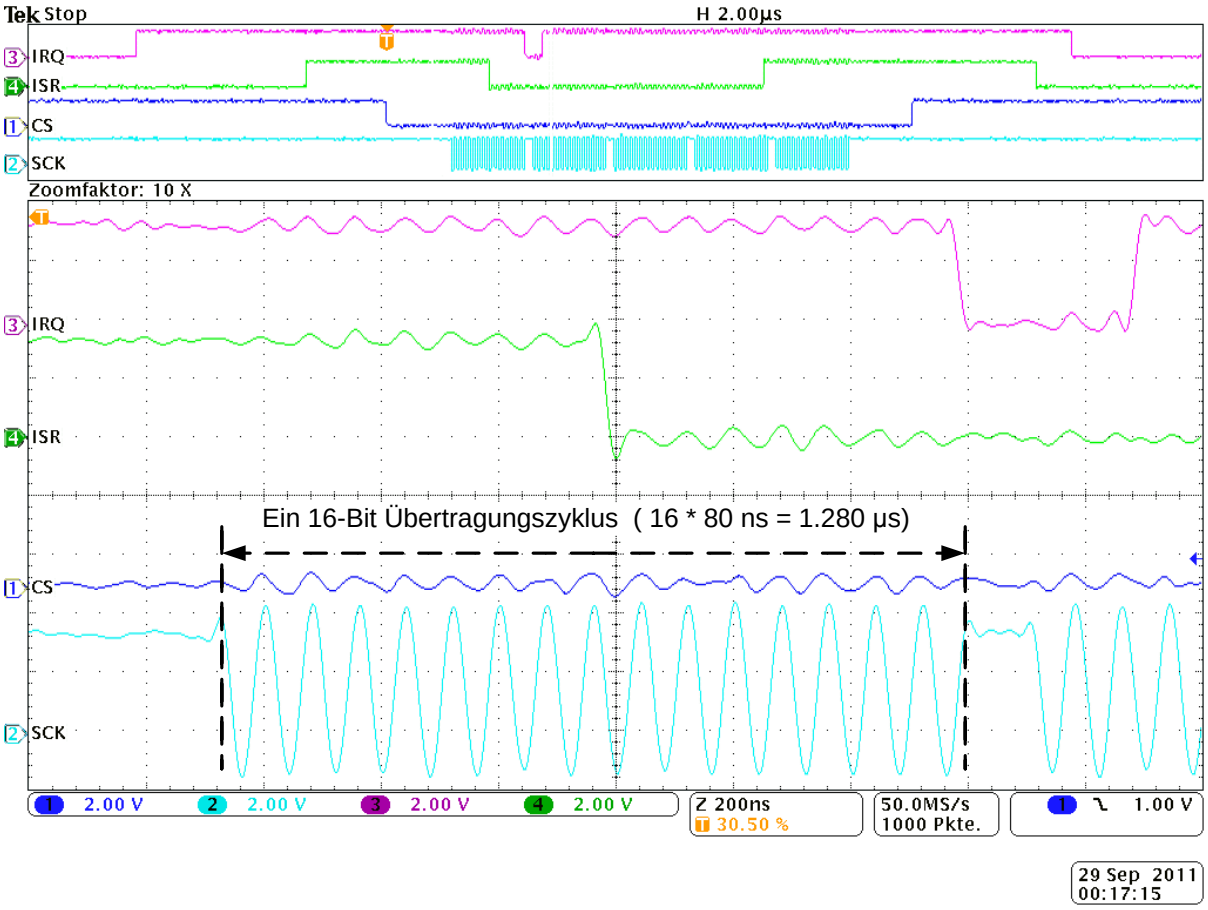

Abb. 3.10.: *Übertragungszeit für 16-Bit Daten aus dem ADU-Schieberegister in das HAW-SPI-Schieberegister bei* $f_{sys} = 12.5$ *MHz*

3.4. Ressourcenbedarf des HAW-SPI IP Cores mit einem externen Timer

Für den Start einer nicht kontinuierlichen MISO-Datenübertragung aus dem ADU in das HAWSPI-Schieberegister wird zu dem HAW-SPI IP zusätzlich der XPS-Timer benötigt, der die Abtastperiode des HAW-SPI erzeugt. Daher zählen zu den verbrauchten Ressourcen(vgl. Tabelle 3.7) des FPGA-Systems (vgl. Kapitel 2.1) durch den HAW-SPI IP auch die verbrauchten Ressourcen (vgl. Tabelle 3.8) durch den XPS-Timer.

Ressourcen	benutzt	Zur Verfügung	benutzt in %
Slices	902	8672	10 %
Slice Flip Flop	677	17344	4 %
4 input LUTs	869	17344	5 %

Tabelle 3.7.: *Ressourcenverbrauch des HAW-SPI IP mit Xillinx Spartan-3E XC3S1200E FPGAs*

Ressourcen	benutzt	Zur Verfügung	benutzt in %
Slices	344	8672	3 %
Slice Flip Flop	298	17344	2 %
4 input LUTs	358	17344	2 %

Tabelle 3.8.: *Verbrauch der Xillinx Spartan-3E XC3S1200E FPGA-Ressourcen durch den XPS_Timer IP*

Die Anzahl der, vom FPGA-Board(Spartan-3E) zur Verfügung gestellten, verbrauchten Ressourcen in der Tabelle 3.9 ergibt sich aus der Summe der Werte der Tabellen 3.7 und 3.8.

Ressourcen	benutzt	Zur Verfügung	benutzt in %
Slices	1246	8672	13 %
Slice Flip Flop	975	17344	6 %
4 input LUTs	1227	17344	7 %

Tabelle 3.9.: *Verbrauch der Xillinx Spartan-3E XC3S1200E FPGA-Ressourcen durch den XPS_Timer IP und der HAW-SPI IP*

4. Reduzierung des Ressourcenbedarfs und der Transferzeit des HAW-SPI IP-Cores

Dieses Kapitel stellt Maßnahmen vor, die folgende Änderungen in dem HAW-SPI-Core und im Timing-Verhalten des MicroBlaze Prozessors beim Empfangsvorgang der von IR-Sensoren erfassten Messdaten durchführt:

- Die Anzahl der 14 Softwareregister wird auf vier reduziert. Zwei dieser SW-Register sind für die IR-Sensor-Messdaten vorgesehen. Die zwei anderen werden für die Status- und Steuerungsdaten des HAW-SPI IP-Cores benötigt. Durch diese Reduzierung sinkt die Anzahl des Ressourcenbedarfs des HAW-SPI IPs(vgl. Abbildung 4.1).
- Die Anzahl der ISRs wird auf eine ISR reduziert. Diese eine ISR dient nur zum Auslesen der von den IR-Sensoren erfassten Messdaten aus den SW-Registern des HAW-SPI IP. Hierdurch sinkt die Beschäftigungszeit des MicroBlaze für einen Datenübertragungszyklus.
- Das Auslesen der vier 16-Bit Messdaten aus den HAW-SPI IP SW-Registern erfolgt paarweise mit jeweils einem 32-PLB-Lesezugriffen. Dadurch werden statt vier nur 2 PLB-Lesezugriffe benötigt.

Für die Steuerung des SPI_Moduls bzw. die Steuerung der MISO-Datenübertragung aus den ADU-Schieberegistern in die HAW-SPI-Schieberegister wird der HAW_SPI IP durch einen SPIController erweitert. Für die Erzeugung der Abtastperiode wird ein interner Timer im HAW-SPI IP integriert.

In Abschnitt 4.1 werden die Funktionalität und die Modellierung des SPI_Controllers, des internen Timers sowie des Timing-Verhaltens des IP Cores vorgestellt. Die Reduzierung von zwei ISRs auf eine ISR hat eine große Auswirkung auf die Laufzeit des Prozessors während der Datenübertragung und auf den Interrupt-Eingang(IRQ) des Prozessors. Diese wird in Abschnitt 4.3 erläutert.

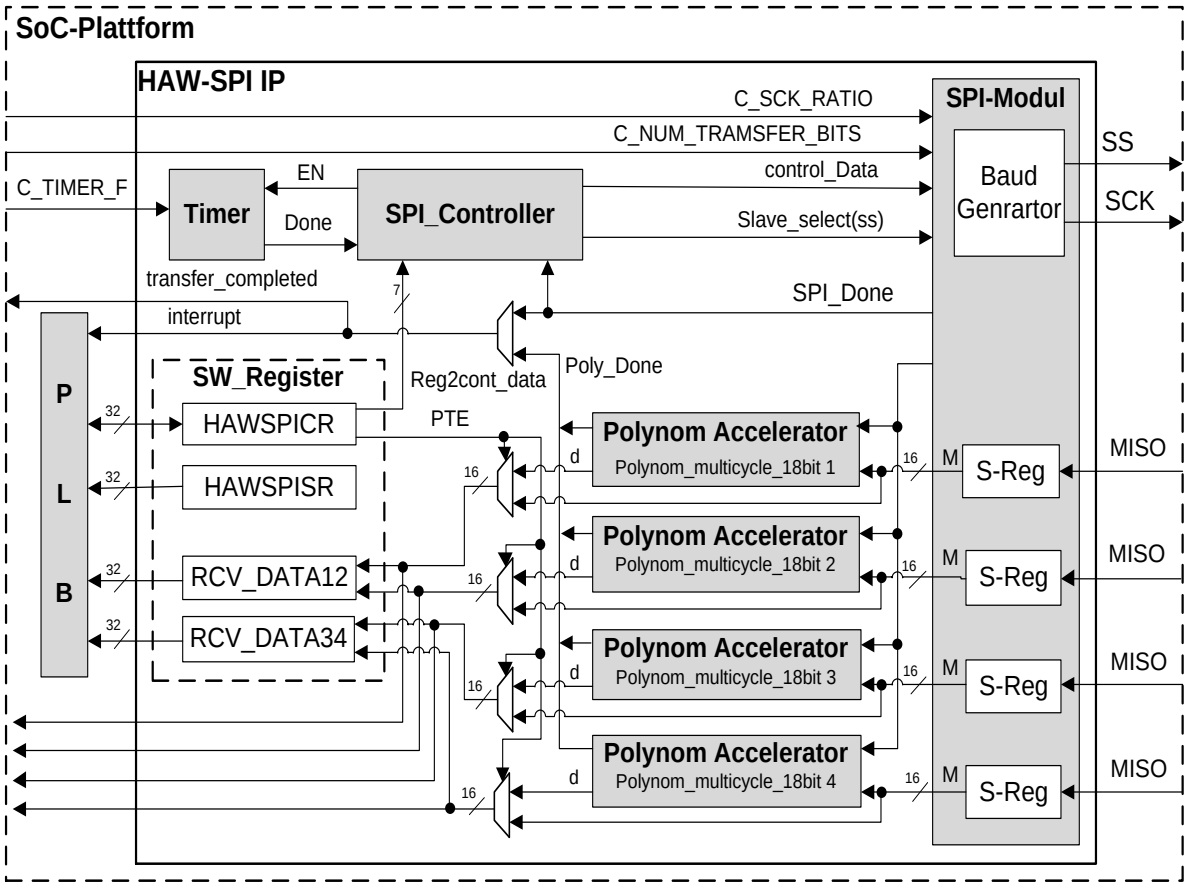

Abb. 4.1.: *Aufbau des HAW_SPI IPs mit reduzierten Softwareregistern und Erweiterung mit einem SPI_Controller durch einen Timer*

4.1. Reduzierung der Interrupt-Sequenz auf eine ISR

Die Steuerung des HAW_SPI für die MISO-Datenübertragung der von den IR-Sensoren erfassten Daten aus den ADUs in die SPI-Schieberegister erfolgte ursprünglich über zwei ISRs(Timer Interrupt Handler und SPI Interrupt Handler). In der ISR "Timer Interrupt Handler" wurde die Übertragung gestartet und in der ISR "SPI Interrupt Handler" wurde diese gestoppt und die erfassten Messdaten aus den Softwareregistern ausgelesen(vgl. Abbildung 3.7).

In diesem Abschnitt wird die ISR "Timer Interrupt Handler" durch einen SPI-Controller ersetzt. Der externe Timer(XPS_Timer) für die Erzeugung der HAW-SPI-Abtastperiode wird durch einen in der HAW-SPI IP intrigierten Timer ersetzt. Die Steuerung bzw. das Starten und Stoppen der MISO-Datenübertragung erfolgt über den SPI-Controller. Daher wird nur eine ISR "SPI Interrupt Handler" zum Auslesen der erfassten Messdaten aus den HAW-SPI-SW-Registern(Data12 und Data34) verwendet(vgl.Abbildung 4.2).

Der Timer erzeugt mit jeder Periode das Timer_Done-Signal. Der SPI-Controller startet nach dem Eintreffen des "Timer_done"-Signals eine MISO-Übertragung. Sobald das SPI-Modul die MISO-Übertragung abgeschlossen hat, wird ein "SPI_Done"-Signal erzeugt(vgl. Listig 4.2).

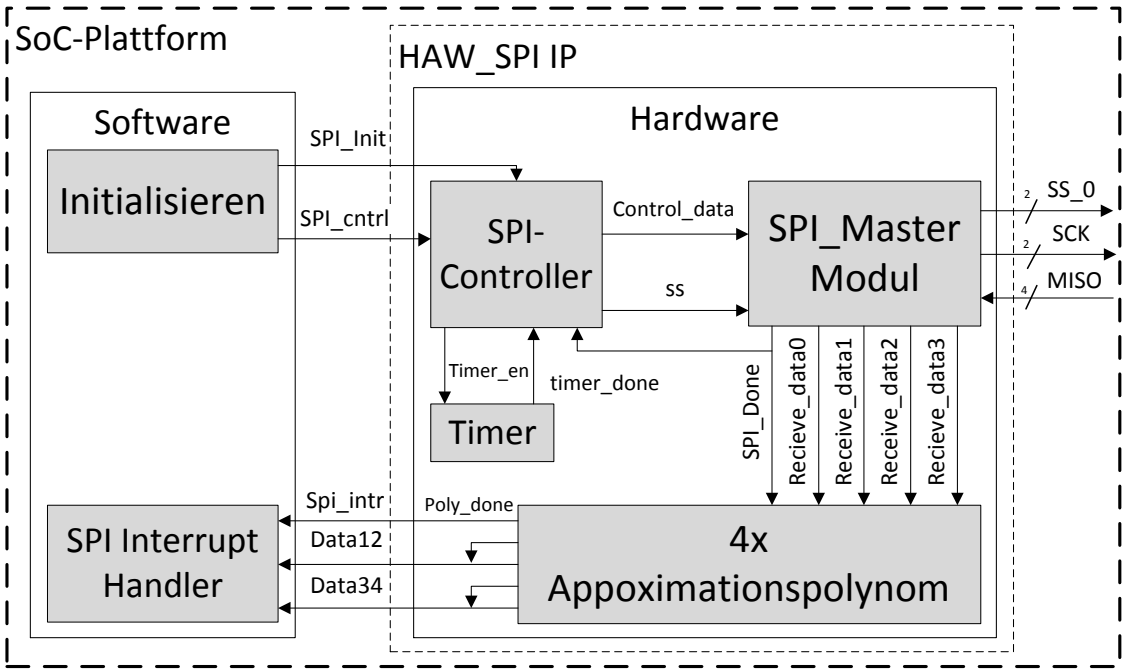

Abb. 4.2.: *Die Zusammenarbeit der Hardware/Software-Module des HAW-SPI IP bei der Steuerung der MISO-Datenübertragung*

4.1.1. Steuerung des SPI-Master-Moduls über den SPI-Controller

Das SPI-Master-Modul startet eine MISO-Übertragung, wenn das MTI-Bit des Eingangs "control_data" den wert '0' und das SPI-Bit den Wert '1' hat(vgl. Abbildung 3.4, Tabelle 4.1).

Nr	SPE	MTI	MISO-Transfer_Start
0	0	0	0
1	0	1	0
2	1	0	1
3	1	1	0

Tabelle 4.1.: *Wahrheitstabelle für das Starten der HAW-SPI MISO-Datenübertragung*

Da das HAW-SPI nur die MISO-Datenübertragung unterstützt, wird das "MTI-Bit mit den Wert '0' initialisiert. Um einen SPI-Empfangsvorgang zu starten bzw. zu stoppen wird nur das SPE-Bit des Master-Modul-Eingangs "control_Data" benötigt(vgl. Kapitel 3.2.1).

- SPE = '1' : SPI-Master Modul erzeugt "SCK"-Takte um den ADU zu takten
- SPE = '0' : SPI-Master Modul stoppt die Erzeugung des SCK-Takts und das SCK geht in den Ruhezustand.

Der SPI-Controller wird über das HAW-SPI Control Register (HAWSPICR) durch das Controller Enable (CEN)-Bit aktiviert bzw. deaktiviert. Für diesen Zweck wird das Register "HAW-SPICR" um das "CEN-Bit" erweitert(vgl. Abbildung 4.3).

Das Register "HAWSPICR" dient als Eingang des SPI-Controllers und wird mit den C Funktionen "SPI_init" und "SPI_cntrl" initialisiert bzw. überschrieben(vgl. Abbildung 4.2 und Anhang B.2).

BIT	0 – 23	24	25	26	27	28	29	30	31
HAWSPICR	Resierviert	PTE	CEN	SS	MIT	MSS	SPHA	CPOL	SPE
READ\Write		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Intial Value		1	1	1	0	1	0	1	0

Abb. 4.3.: *Erweitertes HAWSPICR(Controll register) mit dem CEN-Bit(Controller Enable) für das Toggeln des SPI-Controllers*

Mit dem "CEN-Bit" wird der Controller wie folgt getoggelt:

- CEN='0' (Manueller Modus): In diesem Modus erfolgt der MISO-Übertragungsvorgang über eine Software-Anwendung. Der SPI-Controller sowie der integrierte Timer sind nicht aktiviert. Die 5-LSB-Bits(controll_data)und das SS-Bit des HAWSPICR-Registers werden an das SPI-Master-Modul weitergeleitet (vgl. Listing 4.1 und Abbildungen 4.1, 4.2).
- CEN='1' (Automatischer Modus): In diesem Modus wird der MISO-Übertragungsvorgang von dem SPI-Controller gesteuert. Die vier Steuerungsbits(CPOL, CPHA, MSS, MIT) des HAWSPICR-Registers werden unverändert an SPI-Modul weitergeleitet. Der Controller erzeugt für den MISO-Übertragungsvorgang das SS-Signal und das SPE-Signal neu(vgl. Kapitel 4.1.1.2). Der Abschnitt(Listing 4.1) aus dem VHDL-Code des Controllers(vgl. Anhang A.1) zeigt das Toggeln des Controllers in den beiden Modi.

```
1
2  cen_s                    <= reg2cont_data(6);
3  timer_en                 <= cen_s;
4  control_data             <= control_data_s;
5  ss                       <= ss_s;
6  MUX: process(reg2cont_data,spe_asm,ss_asm,cen_s)
7  begin
8      if(cen_s='1')then                    -- conterollr enable ?
9          ss_s<=ss_asm;                    -- ss aus dem automat
10         control_data_s(0)<=spe_asm;      -- spe aus dem automat
11         control_data_s(4 downto 1)<=reg2cont_data(4 downto 1);
12     else
13         control_data_s<=reg2cont_data(4 downto 0);
14         ss_s<=reg2cont_data(5);          -- ss aus dem sw-register
15     end if;
```

Listing 4.1: *VHDL-Codeabschnitt des SPI-Controllers für das Toggeln zwischen automatischer und manueller MISO-Datenübertragung aus dem ADU in das Schieberegister des SPI-Master-Moduls*

4.1.1.1. Integrierter Timer für die Abtastperiode des HAW-SPIs

Der 21-Bit Timer(vgl. Abbildung4.2) im HAW-SPI erzeugt die Abtastperiode für eine MISO-Datenübertragung aus dem ADU in das Schieberegister des SPI-Master-Moduls. Die Periodendauer des Timers (T_{timer}) ist vom Eingang "C_TIMER_F" und der System-Taktfrequenz

(f_{sys}) abhängig.

$$T_{timer} = T_{sys} * C\_TIMER\_F = (\frac{1}{f_{sys}} = \frac{1}{50MHz} = 20ns) * C\_TIMER\_F \quad (4.1)$$

Ein Zähler(counter), dessen Bit-Breite durch die "C_NUM_TIMER_BITS" bestimmt wird, wird mit jedem Systemtakt bis zu einer bestimmten Obergrenze inkrementiert. Der Wert "C_TIMER_F" ist die Obergrenze des Zählers(counter). Sobald der Zähler diesen Wert erreicht hat, wird das Signal "Timer_Done" erzeugt und der Zähler wieder auf '0' gesetzt(vgl. Listing 4.2). Die Werte "C_TIMER_F" und "C_NUM_TIMER_BITS" können nach Bedarf unter der Berücksichtigung deren minimal bzw. maximal möglicher Werte geändert werden(vgl. Abbildung 4.7). Der Zähler kann in einem Intervall von $C\_TIMER\_F''_{Min}$ bis $C\_TIMER\_F''_{Max}$ zählen.

$$C\_TIMER\_F_{Max} = 2^{21} - 1 = "11111111111111111111" = 2097151 \quad (4.2)$$

$$C\_TIMER\_F_{Min} = C\_NUM\_TRANSFER\_BITS * C\_SCK\_RATIO = 16 * 4 = 64 \quad (4.3)$$

C_NUM_TRANSFER_BITS entspricht der Anzahl der zu übertragenden Datenbits und das C_SCK_RATIO bestimmt die Taktfrequenz des SPI-Taktes(SCK)(vgl. Kapitel 3.2.1).

Die Periodendauer des integrierten Timers bzw. die Abtastperiode der SPI-MISO-Übertragung liegt zwischen T_{Max} und T_{Min}.

$$T_{Max} = T_{sys} * C\_TIMER\_F_{Max} = 20ns * 2097151 = 41,94ms \quad (4.4)$$

$$T_{Min} = T_{sys} * C\_TIMER\_F_{Min} = 20ns * 64 = 1280ns \quad (4.5)$$

```
1  entity spi_timer is
2      generic(
3          C_NUM_TIMER_BITS     : integer ;
4          C_TIMER_F    : std_logic_vector(20 downto 0)
5      );
6     port(
7          Bus2IP_Clk           : in  std_logic;
8          Bus2IP_Reset         : in  std_logic;
9          timer_en             : in  std_logic;
10         timer_done           : out std_logic
11    );
12 end spi_timer;
13 ------------------------------------------------------------------------
14 ------------------------------------------------------------------------
15 architecture imp of spi_timer is
16 signal  timer_done_s     : std_logic;
17 signal  counter          : std_logic_vector(0 to C_NUM_TIMER_BITS-1);
18 begin
19 CONT_PROC : process( Bus2IP_Clk ) is
20 begin
21    if( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
22         if ( Bus2IP_Reset = '1' ) then
23             counter              <= (others => '0');
```

```
24          timer_done_s    <= '0';
25        else
26              if(timer_en='1') then
27                  counter  <= counter + 1;
28                  if ( counter = C_TIMER_F ) then
29                      timer_done_s    <= '1';
30                      counter     <= (others => '0');
31                  else
32                      timer_done_s <= '0';
33                  end if;
34              end if;
35          end if;
36      end if;
37  end process CONT_PROC;
38  timer_done<=timer_done_s;
39  end imp;
```

Listing 4.2: *Integrierter Timer startet periodisch die HAW-SPI-Empfangsvorgänge*

Der Vorteil des integrierten Timers gegenüber dem XPI-Timer(externer Timer):

- Keine PLB-Verbindung
- Taktperiode über IP-Config konfigurierbar(vgl. Abbildung4.7). Im Falle der Verwendung des HAW-SPI ohne Software-Anwendung, werden die Bit-Breite des Zählers(counter) und seine Obergrenze über die Top-Entity des IPs konfiguriert(vgl. Listing 4.3).
- Minimale Nutzung der Ressourcen(21-Bit counter).

4.1.1.2. SPI-Controller als Moore-Automat

Der Moore-Automat stellt den Steuerungsablauf des SPI_Controllers bei einer MISO-Datenübertragung aus den ADU in das HAW-SPI-Schieberegister dar. Der zu entwerfende Automat soll die Kombinationen der drei Eingangssignale CEN, TIMER_DONE und SPI_Done in der Reihenfolge(1xx, 11x, xx1) mit den Ausgängen SS und SPE in der Folge(10, 00, 01, 00, 10) anzeigen. Der Automat soll mit einem synchronen Reset in den Zustand "IDLE"(Ausgang = '10') versetzt werden(vgl. Abbildung 4.4).

Semantik der Eingänge des Automaten:

- **CEN**: Mit CEN wird der SPI-Controller von der Software-Anwendung über das HAW-CPICR aktiviert(vgl. Abbildung 4.3).
- **Timer_done**: Das Ausgangssignal des integrierten Timers des HAW-SPIs entspricht der Abtastperiode des SPI-Übertragungsvorgangs.
- **SPI_Done**: Mit dem SPI_Done signalisiert das SPI-Master-Modul, dass eine 16-Bit-Datenübertragung aus dem ADU in das SPI-Schieberegister erfolgreich abgeschlossen und der HAW-SPI bereit für eine weitere MISO-Übertragung ist.

Semantik der Ausgänge des Automaten:

- **SS**: Das low-aktive Signal SS dient dem Selektieren des ADUs durch das SPI-Master-Modul. SS entspricht dem Signal "ss_asm" aus der Listing 4.1.

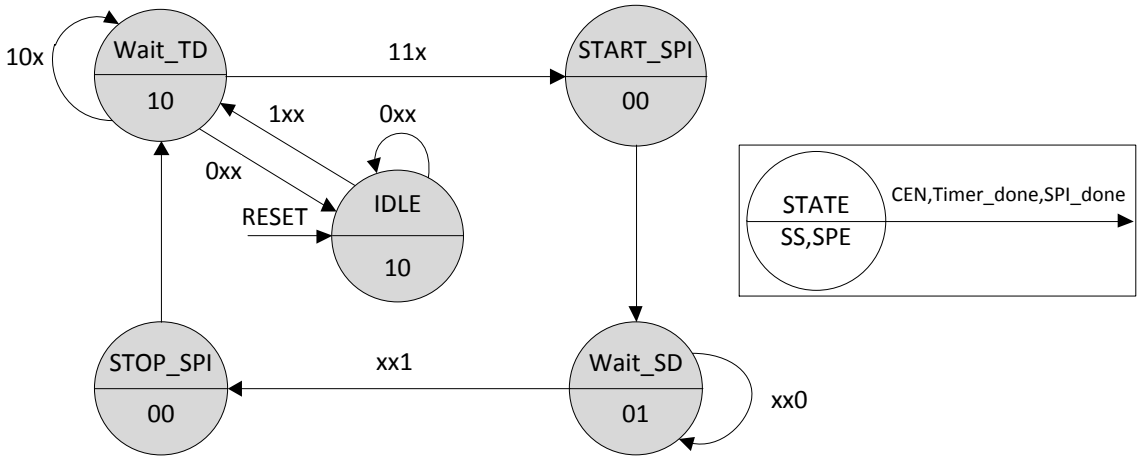

Abb. 4.4.: *Zustandsdiagramm des SPI_Contollers*

- **SPE**: Das SPE startet die Erzeugung des SCK-Taktes durch das SPI-Master-Modul und somit den SPI-Empfangsvorgang(vgl. Kapitel 4.1.1). SPE entspricht dem Signal "spe_asm" aus der Listing 4.1.

Beschreibung der Zustände des Automaten nach Abbildung 4.4:

IDLE : Der Automat ist im Zustand "IDLE" und wartet auf seine Aktivierung durch den Eingang "CEN". In diesem Zustand reagiert bzw. ändert der Automat seinen Zustand nur auf der Eingangskombination "1xx". Die Eingänge Timer_Done und SPI_Done werden ignoriert. Der Ausgang '10' deutet darauf, dass der ADU nicht selektiert wird(SS = '1')[1]. Der Automat wechselt in den Zustand "WAIT_TD", sobald das Aktivierungssignal "CEN" eintrifft.
Die Aktivierung dieses Automaten und das Starten des integrierten Timers erfolgt über das selbe Signal "CEN"(vgl. Anhang A.1).

WAIT_TD : Der Automat wartet in diesem Zustand auf die steigende Flanke des integrierten Timers. Die Ausgänge bleiben wie im IDLE-Zustand. Wurde das Signal Timer_Done erzeugt, wechselt der Automat in den Zustand "START_SPI".
Außerdem wird in diesem Zustand auf den Eingang "CEN" mit einem Wechsel in den IDLE-Zustand reagiert, falls der Automat zuvor von der Software-Anwendung deaktiviert worden ist.

START_SPI : In diesem Zustand Selektiert der Automat den ADU (SS = '0'). Um die minimale Verzögerung (10 ns) zwischen dem Selektieren des ADUs und dem Erzeugen der SPI-Taktflanke zu gewährleisten, braucht der Automat einen zusätzlichen Zustand bzw. eine Systemtaktperiode[2](vgl. Kapitel 2.4). Deswegen geht der Automat in den Zustand "Wait_SD" ohne ein SCK-Signal zu erzeugen.

WAIT_SD : In diesem Zustand wird die MISO-Datenübertragung gestartet bzw. ein SCK-Signal erzeugt und damit das Slave Device(ADU) getaktet. Der Automat wartet auf

[1]Der ADU ist low-aktiv und wird mit SS = '0' selektiert.(siehe Kapitel 2.4)
[2]Bei einer Systemfrequenz von 50 MHz reicht eine Taktperiode(20 ns) für die Verzögerungszeit zwischen SS und SCK aus

das Ende der MISO-Datenübertragung. Nach 16 SCK-Takten sendet das SPI-Master-Modul das Signal "SPI_Done" und der Automat wechselt daraufhin in den Zustand "STOP_SPI". An den Ausgangssignalen ändert sich nichts.

STOP_SPI : Wenn der Automat sich in diesem Zustand befindet, sind die zu übertragenden Messdaten aus dem ADU in das HAW-SPI-Schieberegister übertragen worden. Nun beendet der Automat den ADU mit dem SCK zu takten, deselektiert den ADU und geht unmittelbar in den Zustand "WAIT_TD".

Dieser Automat wurde in zwei nebenläufigen VHDL-Prozessen synthetisiert(vgl. Anhang A.1):

Zustandsaktualisierung : In diesem synchron getakteten Prozess wird das Zustandsregister aktualisiert.

Folgezustandsberechnen : In diesem kombinatorischen Prozess wird der Folgezustand zu dem aktuellem Zustand berechnet. In der Empfindlichkeitsliste dieses Prozesses stehen die Eingangssignale CEN, Timer_Done, SPI_Done sowie der aktuelle Zustand.

4.1.2. VHDL-Simulation des Controllers mit ModelSim

Bevor der SPI-Controller aus der Abbildung 4.2 in Hardware umgesetzt wird, erfolgt eine VHDL-Simulation mit ModelSim, um den Automaten aus der Abbildung 4.4 auf Fehler zu testen. Der Integrierte Timer erzeugt das Signal Timer_Done mit einer Abtastfrequenz von 1 MHZ. Das Signal SPI_Done wird von einem Zähler in TestBench erzeugt. Nach 16 Takten, also der Zeit für eine 16 Bit MISO-Datenübertragung, wird das Timer_Done Signal erzeugt. Die Abbildung 4.5 stellt das Timing-Verhalten des SPI-Controllers bei der 16 Bit MISO-Datenübertragung aus dem ADU in das HAWSPI-Schieberegister.

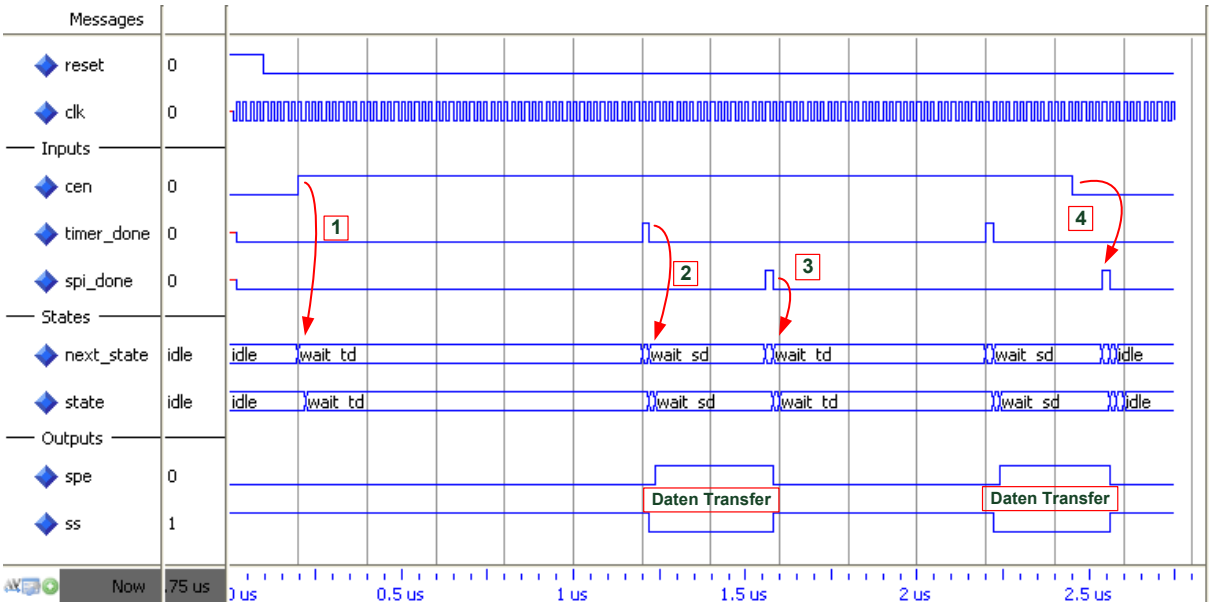

Abb. 4.5.: *VHDL-Simulation des SPI-Controllers(vgl.4.4)*

1. Der SPI-Controller sowie der integrierte Timer werden über das CEN-Bit des HAWSPICR-Registers aktiviert. Das System geht aus dem "IDLE" in den "WAIT_TD" Zustand und wartet auf die Abtastperiode (Timer_Done) des Timers.
2. Nach einer Abtastperiode wird die MISO-Datenübertragung aus dem ADU in das HAW-SPI-Schieberegister gestartet(vgl. Abbildung 4.6).
3. Die MISO-Datenübertragung wurde Abgeschlossen. Der Automat geht in den Zustand "Wait_TD" und wartet auf die nächste Taktperiode des Timers.
4. Das System befindet sich in dem "WAIT_SD" -Zustand. Eine MISO-Datenübertragung findet statt. Der SPI-Controller ist deaktiviert. Der Automat regiert nicht auf den Eingang "CEN" und wartet bis die Übertragung erfolgreich abgeschlossen ist. Erst wenn der Automat sich in dem Zustand "WAIT_TD" befindet, reagiert er auf "CEN" indem er in den Zustand "IDLE" wechselt.

Die Abbildung 4.6 stellt den Ablauf eines 16-Bit MISO-Datenübertragungszyklus.

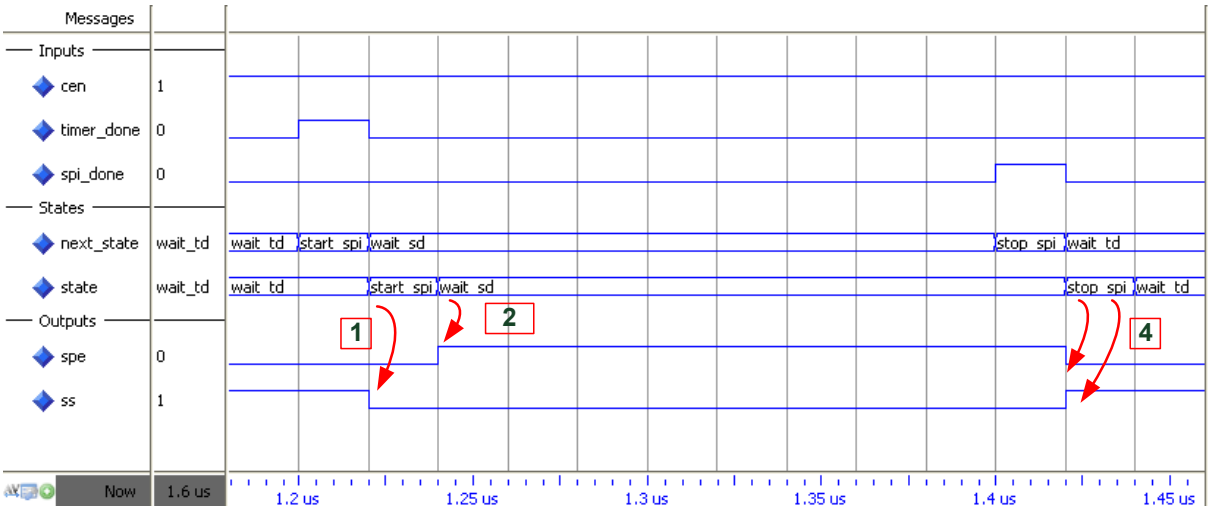

Abb. 4.6.: *Ausschnitt aus der VHDL-Simulation des SPI-Controllers*

1. Der Automat wartet im Zustand "WAIT_TD" auf eine steigende Flanke des "Timer_done"-Signals, um die MISO-Datenübertragung zu starten. In dem Zustand "START_SPI" wird der ADU selektiert und der Automat geht in den nächsten Zustand.
2. Ein Takt nach dem Selektieren des ADU(im Zustand "START_SPI") wird der ADU getaktet. Die Verzögerungszeit zwischen SS und SPE entspricht der Verzögerungszeit zwischen Chip Select (CS) und SCK des ADUs (vgl. Abbildung 2.12).
3. Die 16-Bit MISO-Datenübertragung ist abgeschlossen. Der Automat beendet die Übertragung, geht in den Zustand "WAIT_TD" und wartet auf den Beginn des nächsten MISO-Übertragungszyklus.

4.2. Reduzierung der Anzahl der Softwareregister

HAW-SPI IP verfügt über 14 Softwareregister, 4 Register (RCV_DATA1-4) für das Auslesen der mit den IR-Sensoren erfassten Messdaten, 7 Register(COEFF1-5, MU1, MU2) für die Koeffizienten und die Skalierungsfaktoren des Polynom-Accelerators und 3 Register(HAWSPICR, HAWSPISR, HAWSPISS) für das Controllregister, das Statusregister und das Slave Select-Register des IPs(vgl. Abbildung 3.2).
Da die Werte der Koeffizienten und der Skalierungsfaktoren des Polynoms zur Laufzeit der MISO-Übertragung nicht geändert werden, ist es sinnvoll, dass die in der Top-Entity des IPs über "Generic" erfolgen (vgl. Listing 4.3). Dadurch werden 7 SW-Register gespart.

```
1  entity haw_spi is
2    generic(
3      -- ADD USER GENERICS BELOW THIS LINE ---------------
4      C_SCK_RATIO      : integer           := 4;
5      C_NUM_TRANSFER_BITS: integer         := 16;
6      C_TIMER_F        : std_logic_vector  :="1100001101010000";  --50000;
7      C_POLY_COEFF1    : std_logic_vector  :="0000001111011111011";--X"F7B";
8      C_POLY_COEFF2    : std_logic_vector  :="1101001010000010110";--X"34A16";
9      C_POLY_COEFF3    : std_logic_vector  :="0010100001101100000";--X"A1B0";
10     C_POLY_COEFF4    : std_logic_vector  :="1101011110111011100";--X"35EEC";
11     C_POLY_COEFF5    : std_logic_vector  :="0100111111001110111";--X"13F3B";
12     C_POLY_MU1       : std_logic_vector  :="0110011101001010011";--X"19D29";
13     C_POLY_MU2       : std_logic_vector  :="0000000010001100111";--X"119";
14     -- ADD USER GENERICS ABOVE THIS LINE ---------------
```

Listing 4.3: *GENERIC-Definition des HAW-SPI Top-Entitys*

Bei Bedarf lassen sich die Werte der Generics über "IP-Config" entsprechend ändern, in dem man in der XPS-Entwicklungsumgebung mit der richten Maustaste auf HAW-SPI IP klickt.(vgl. Abbildung 4.7).

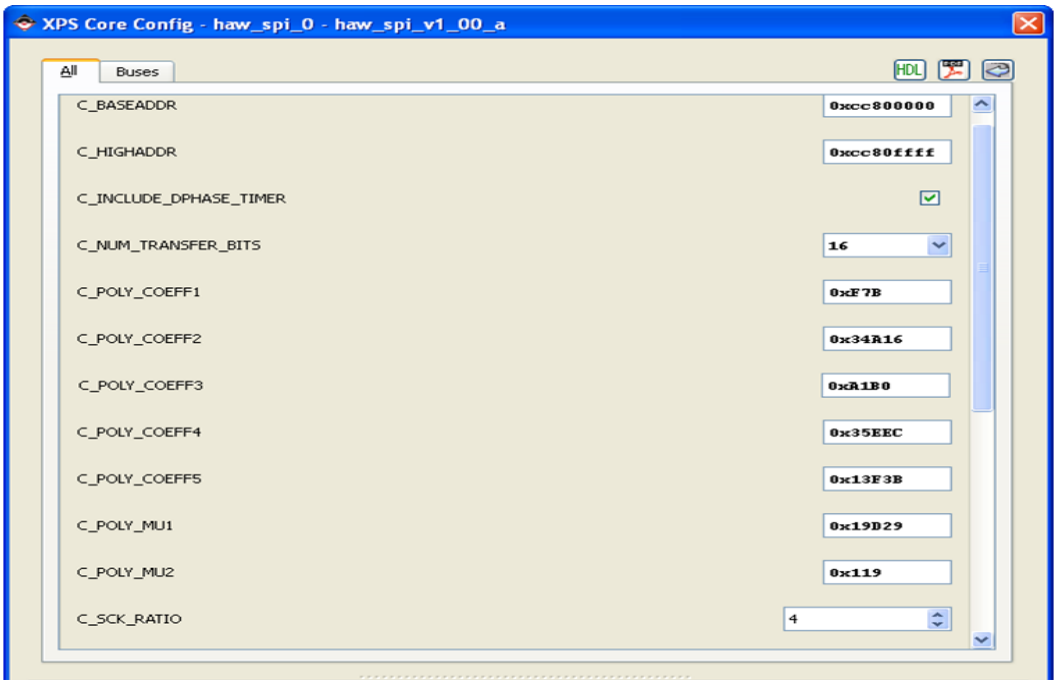

Abb. 4.7.: *Kunfiguration des HAW_IP Cores über XPS*

Ein SW-Register wird von dem HAWSPISS belegt, obwohl nur ein LSB-Bit "SS" des Register genutzt wird. Das SS-Bit wird dem HAWSPICR hinzugefügt und damit ein weiteres SW-Register gespart(vgl. Abbildung 4.8).

BIT	0 – 23	24	25	26	27	28	29	30	31
HAWSPICR	Resierviert	PTE	CEN	SS	MIT	MSS	SPHA	CPOL	SPE
READ\Write		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Intial Value		1	1	1	0	1	0	1	0

Abb. 4.8.: *Erweitertes HAWSPICR(Controll register) mit dem SS(Slave Select)-Bit des HAWSPISS-Registers*

Der PLB-BUS des MicroBlaze-Systems verfügt über eine minimale Breite von 32 Bit. Da die erfassten Messtaten(RVC_DATA1-5) 16 Bit breit sind, können mit einem PLB-Lesezugriff gleich zwei Datensätze aus den Softwareregistern ausgelesen werden(vgl. Listing 4.4 und Listing 4.6). Dadurch werden zwei Registeradressen eingespart.

```
1 case slv_reg_read_sel is
2     when "1000" => slv_ip2bus_data <= "00000000000000000000000000"&
          polynom_toggle&reg2SA_Data_cntrl_int;
3     when "0100" => slv_ip2bus_data <= "00000000000000000000000" &
          reg2SA_Data_status_int;
4     when "0010" => slv_ip2bus_data <= reg2SA_Data0_receive_int&
          reg2SA_Data1_receive_int;
5     when "0001" => slv_ip2bus_data <= reg2SA_Data2_receive_int&
          reg2SA_Data3_receive_int
6     when others => slv_ip2bus_data <= (others => '0');
7  end case;
```

Listing 4.4: *Kombinatorisches Lesen des SW-Registers*

Durch die oben genannten Maßnahmen wird die Anzahl der Softwareregister von 14 auf 4 reduziert(vgl. Abbildung 4.1):

- HAWSPICR: Beinhaltet das Slave Select-Bit und die Kontrollbits für das SPI-Master-Modul(vgl. Abbildung 4.8)
- HAWSPISS: HAW-SPI Status Register
- RCV_DATA12: Erfasste Messtaten der zwei IR-Sensoren
- RCV_DATA34: Erfasste Messtaten der zwei IR-Sensoren

4.3. Timing-Verhalten des HAW_SPI IPs für einen 16-Bit-Übertragungszyklus mit dem SPI-Controller und dem integrierten Timer

Die Messung des Timing-Verhaltens für einen 16-Bit-Datenübertageszyklus aus dem ADU in die Schieberegister des HAW_SPIs, das Umrechnen der erfassten Daten durch den Polynom-Accelerator in metrische Größen und das Auslesen des Messtaten aus den Softwareregistern in der ISR erfolgt über die vier Leitungen(CS, SCK, IRQ und ISR)(vgl. Abbildung 4.10).

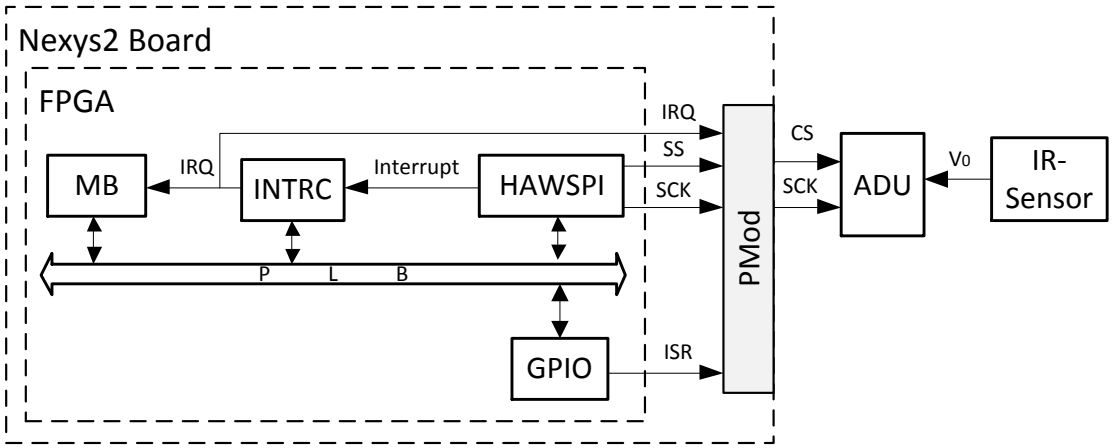

Abb. 4.9.: *Auslesen der Messsignale im SPI Interupt Handler(ISR) am Eingang des MicroBlaze(IRQ) und an den Eingängen des ADUs (CS, SCK)(vgl. Abbildung 4.10)*

Der Ausgang des GPIOs(ISR) und das Ausgangssignal des Interrupt-Controllers bzw. das Eingangssignal des MicroBlaze werden direkt an PMod angebunden. Der GPIO wird am Anfang des SPI-Interrupt-Handlers gesetzt. Nach dem Auslesen des erfassten Daten aus den Softwareregistern wird er wieder gelöscht(vgl. Listing 4.6).
Mit dem Signal "CS" bzw. "SS" selektiert das SPI-Mater-Modul über die Pmod-Schnittstelle den ADU. Mit dem "SCK"-Signal wird der ADU mit einer Frequenz von 12.5 MHz getaktet. Das Timing-Verhalten der gemessen Signale lässt sich wie folgt beschreiben(vgl. Abbildung 4.10)

t1 : die Zeit, in der das SPI-Master-Modul den ADU für einen 16-Bit Datenübertragungszyklus selektiert

t2 : die Zeit, in der der SPI-Takt(SCK) den ADU 16 mal für die 16-Bit MISO-Datenübertragung taktet. Die Periodendauer des SPI-Takts bei einer Frequenz von "$f_{SCK} = 12.4MHz$" ist "$T_{sck} = 80ns$" (vgl. Kapitel 3.2.1). Die gesamte Zeit für 16 SCK-Takte beträgt $t_1 = 16 * T_{SCK} = 16 * 80ns = 1.28\mu s$.

t3 : Die Ausführungszeit des Polynom-Accelerators beträgt $t_3 = 9 * T_{sys} = 9 * 20ns = 180ns$ (vgl. Kapitel 3.2.2). Nach der Ausführung des Polynom-Accelerators stehen die vier Messdaten(Data0-3) in zwei Software-Registern für den MicroBlaze zur Verfügung.

t4 : Nach dem Eintreffen des Interrupts am MicroBlaze-Interrupt-Eingang (IRQ = '1') bis zum Start der ISR(HAW_SPI_Intr_Handler) vom MicroBlaze vergehen $2.525\mu s$. In dieser Zeit wird der "IRQ" bearbeitet und es findet ein Kontextwechsel statt.

t5 : Der MicroBlaze befindet sich in der ISR für $2.520\mu s$. In der ISR wird die Interrupt-Quelle geprüft. Falls der Interrupt von der HAW-SPI ist, werden alle vier 16-Bit-Messdaten mit zwei PLB-Zugriffen ausgelesen. Schließlich wird die Interrupt-Quelle zurückgesetzt(vgl. Listing 4.5).

```
1  void haw_spi_intr_handler(void * baseaddr_p) {
2      XGpio_WriteReg(XPAR_ISR_GPIO_BASEADDR,1,1);
3      Xuint32 IntrStatus, baseaddr;
4      baseaddr = (Xuint32) baseaddr_p;
5
6      IntrStatus = HAW_SPI_mReadReg(baseaddr,HAW_SPI_INTR_IPISR_OFFSET);
7      if ((IntrStatus & INTR_DRR_FULL_MASK) == DRR_INTR) {
8          //HAW_SPI_Stop((void *) XPAR_HAW_SPI_0_BASEADDR);
9          //HAW_SPI_Clear_SS((void *) XPAR_HAW_SPI_0_BASEADDR);
10         HAW_SPI_Read_4in2_IRs();
11         HAW_SPI_mWriteReg(baseaddr,HAW_SPI_INTR_IPISR_OFFSET,DRR_INTR)
               ;
12         spi_done=1;
13     }
14     XGpio_WriteReg(XPAR_ISR_GPIO_BASEADDR,1, 0);
15 }
```

Listing 4.5: *HAW-SPI Interrupt Handler für das Auslesen vier 16-Bit-Messdaten mit zwei 32-Bit-PLB Zugriffen*

Das Auslesen der vier erfassten Messdaten aus den SW-Registern erfolgt anstatt mit vier PLB-Zugriffen mit zwei PLB-Zugriffen. In der ISR werden die vier Messdaten mit Schiebeoperatoren wieder auseinander getrennt. Dadurch werden zwei 32-Bit PLB-Zugriffe gespart, dafür aber zwei 16-Bit-Schiebeoperationen hinzugefügt(vgl. Listing 4.6). Die ISR-Bearbeitungszeit wird damit um $200ns$ reduziert.

```
1  void HAW_SPI_Read_4in2_IRs(void) {
2      Xuint32 IR_01 =
3              HAW_SPI_mReadReg(XPAR_HAW_SPI_0_BASEADDR,
                   HAW_SPI_RCV1_0_OFFSET);
4      Xuint32 IR_23 =
5              HAW_SPI_mReadReg(XPAR_HAW_SPI_0_BASEADDR,
                   HAW_SPI_RCV3_2_OFFSET);
6      reg1 = IR_01;          // LSB-Teil von IR_01 = IR1
7      reg2 = (IR_01 >> 16); // MSB-Teil von IR_01 = IR2
8      reg3 = IR_23;          // LSB-Teil von IR_23 = IR3
9      reg4 = (IR_23 >> 16); // MSB-Teil von IR_23 = IR4
10
11 }
```

Listing 4.6: *Auslesen der vier 16-Bit-Messdaten mit zwei 32-Bit-PLB Zugriffen*

t6 : Der Interrupt-Eingang des MicroBlaze (IRQ) ist für eine Zeit von $5.860\mu s$ belegt. In dieser Zeit empfängt der MicroBlaze keinen Interrupt.

t7 : die Verzögerungszeit, in der die Interrupt-Quelle in der ISR zurückgesetzt wird

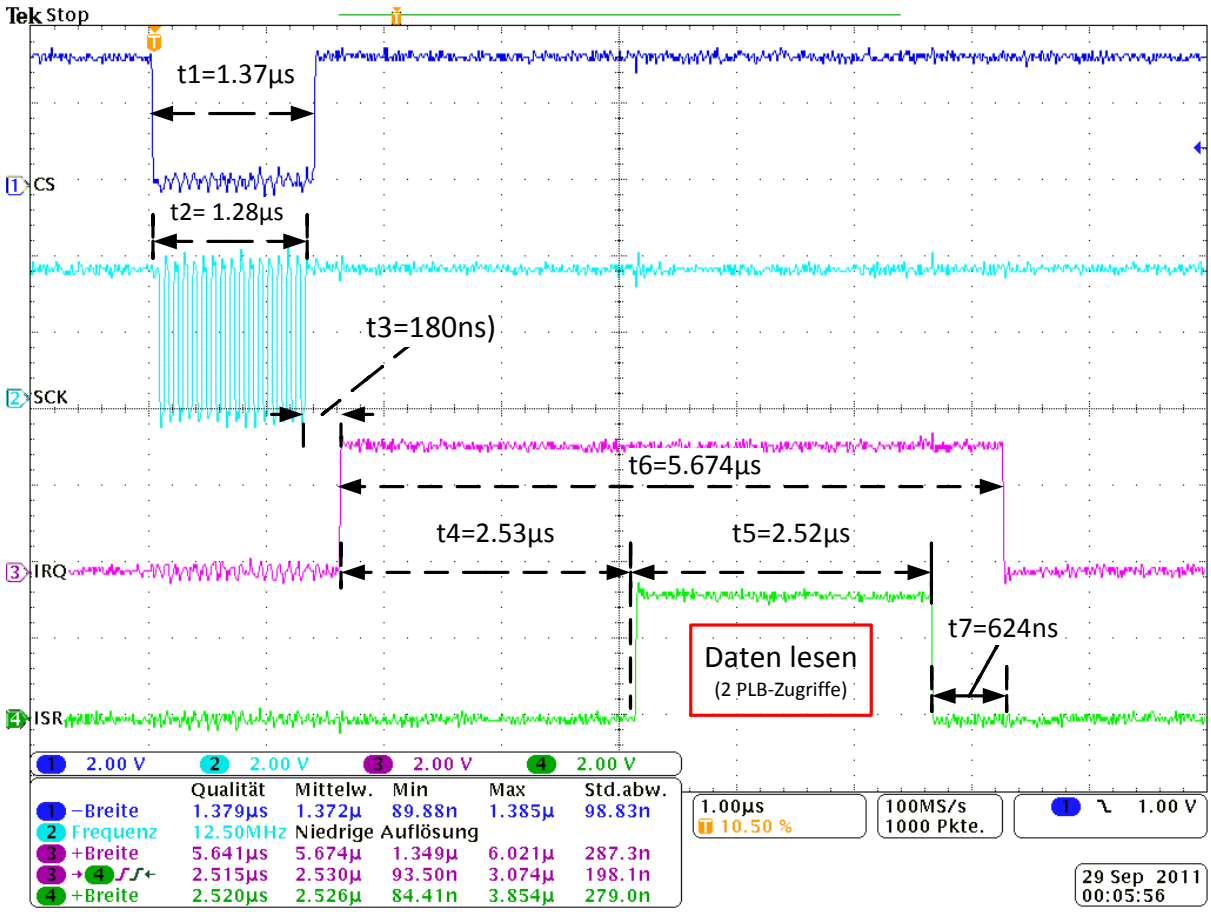

Abb. 4.10.: *Das Empfangen von vier 16-Bit Datensätzen aus den ADUs in die vier HAW_SPI SW-Register und das Auslesen dieser SW-Register mit zwei 32-Bit PLB-Lesezugriffen (2 Messdaten mit einem 32-PLB-Zugriff)*

4.4. HAW-SPI Bustransfermessung

Da das Slave Device (ADU) der MISO-Übertragung in low-aktive $\overline{CS}$ betrieben wird, sollte das HAW-SPI "SS" low-aktiv konfiguriert werden. Für einen erfolgreichen Transfer müssen Master und Slave die gleiche Konfiguration haben[6].

Für diese Bustransfermessung wurde ein Abstand von 12 cm mit dem Konfigurationsmodus " CPOL=1 und CPHA=0" gemessen(vgl. Kapitel 2.5). Das bedeutet, dass das MSB-Bit des ADU-Schieberegisters bei der ersten Taktflanke von dem SPI-Master-Modul übernommen wird. Der ADU(AD1) ist low-aktiv und der SCK-Pegel ist in Ruhezustand HIGH. Der Slave(AD1) hat bei jeden fallenden Flanke eine halbe Taktflanke Zeit für das Schieben eines Datenbits zum Ausgang. Nachdem das neue Datenbit am Ausgang ist, hält der Slave(AD1) es für die andere Hälfte der Taktperiode stabil, damit es von dem Master übernommen wird(vgl. Kapitel 2.4). Der Master (HAW-SPI Modul) tastet das Datenbit in diesem Modus(CPOL=1 und SPHA =0) bei einer fallenden Flanke ab(vgl. Abbildung 4.11).

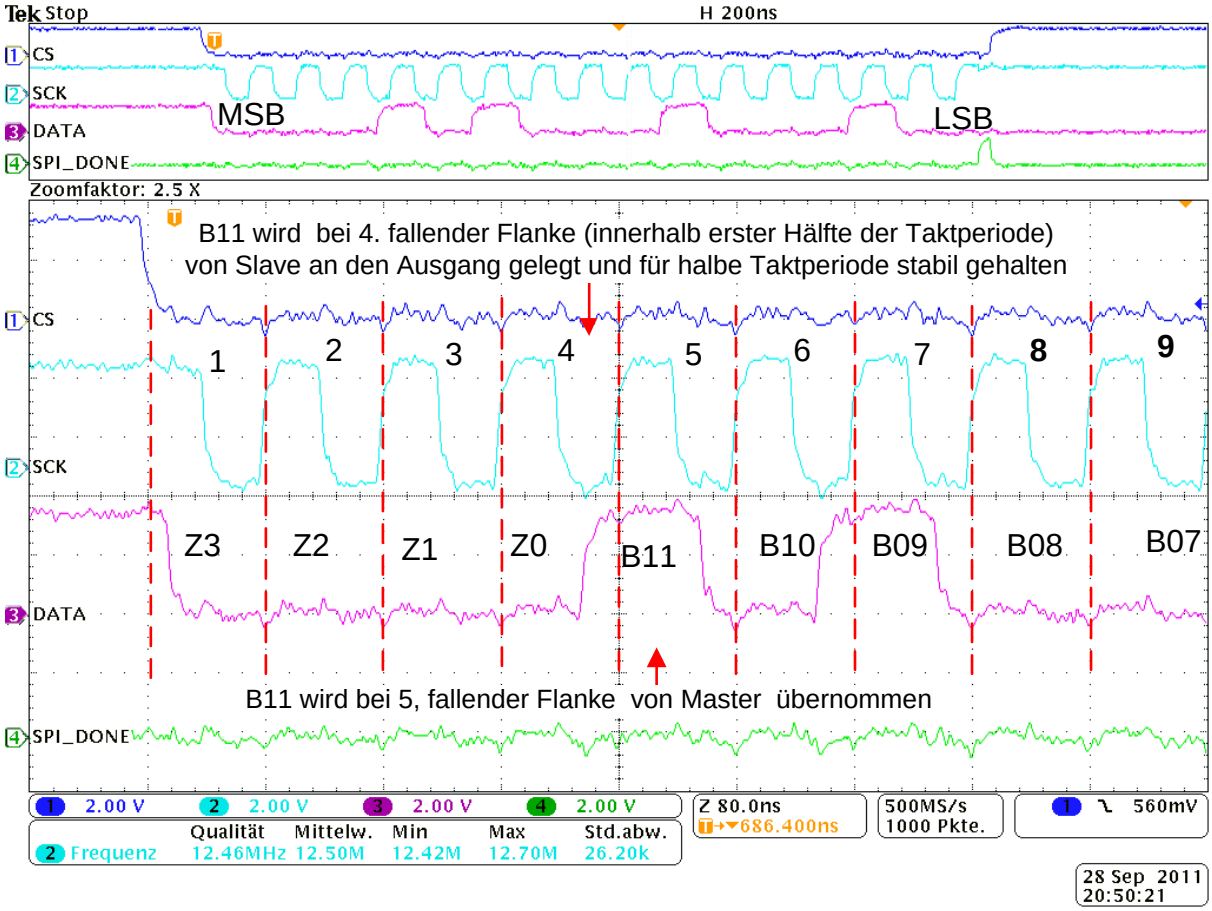

Abb. 4.11.: *Messen eines Abstands von 12 cm, CPOL = 1, CPHA = 0, SCK-Frequenz 15.5 MHz* , Master Datenübernahme bei der ersten fallenden Flanke

Nachdem "SC" den Wert '0' hat, verlässt der Slave(AD1) den hochohmigen Zustand und legt die erste Null(Z3) des "SData"s (Slave Data) an den Ausgang. Erst bei der ersten Flanke schiebt er die zweite Null an den Ausgang. Das erste Datenbit(B11) wird in der 4. fallenden Flanke an den Ausgang gelegt und bei der 5. fallenden Flanke vom Master(MData) übernommen. Das letzte Datenbit(B0) wird bei der 15. fallenden Flanke an den Ausgang gelegt und bei der 16. fallenden Flanke vom Master(MData) übernommen(vgl. Abbildung

4.11).

Der Inhalt der MISO-Daten(MData) lässt sich mit der folgenden Formel berechnen.

$$MData = \sum_{B=0}^{11} d_B * 2^B, dB \in \{0, 1\}. \tag{4.6}$$

SCK	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SData	Z2	Z1	Z0	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0	
MData	Z3	Z2	Z1	Z0	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
Inhalt	0	0	0	0	1	0	1	0	0	0	1	0	0	0	1	0

Tabelle 4.2.: *Berechnung von erfassten Daten*

$$\sum_{B=0}^{11} 2^B = 2^{B11} + 2^{B9} + 2^{B5} + 2^{B1} = 2048 + 512 + 32 + 2 = 2594 = M \tag{4.7}$$

M = 2594 entspricht 12 cm (vlg. Tabelle 4.3)

4.5. Erfassen und Auswertung der Messdaten

Der Abstand zwischen einem Messobjekt und dem GP2D12 IR-Sensor wurde im Bereich von 4 cm bis 90 cm gemessen. Bei jedem Messvorgang wurden die Messdaten aus der Ausgangsspannung des IRs (V_0), aus dem analogen Ausgang des ADUs (d_0), den 16-Bit SPI-Schieberegistern(M) und dem 16-Bit Ausgang des Polynom-Moduls(d) parallel erfasst(vgl. Tabelle 4.3).
Die Ausgangsspannung (V_0) des IR-Sensors wurde mit einem Spannungsmessgerät direkt am Ausgang des Sensors gemessen(vgl. Abbildung 4.12). Die 12-Bit erfassten Messdaten(M) und die daraus durch den Polynom-Accelerator in eine metrische Größe umgerechnete Messstrecke(d) wurde aus den Softwareregistern über den PLB-Bus ausgelesen und mit "xil_printf" über den Hyperterminal an einem PC ausgegeben.

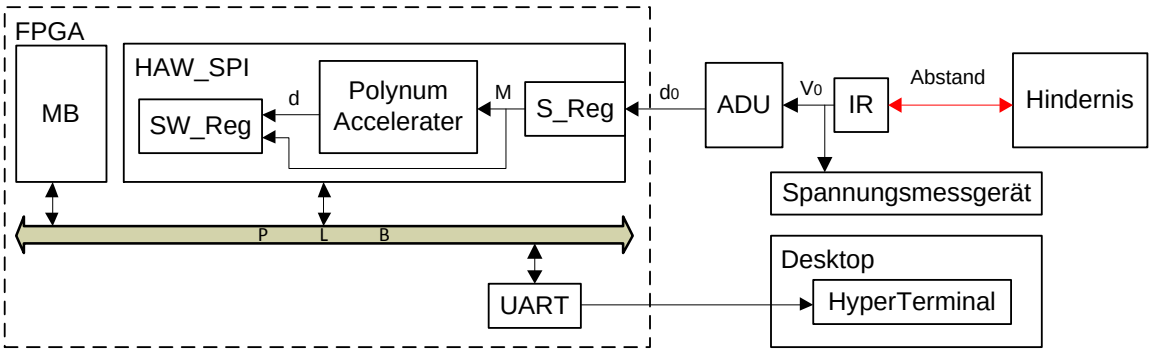

Abb. 4.12.: *HAW-SPI gekoppelt mit einem ADU und einem IR-Sensor zum Messen der Abstände zwischen dem IR-Sensor und dem Messobjekt(Hindernis)*

Der mit dem Polynom-Accellerator berechnete Messabstand wird in der Q-Format Fix_16_9-Darstellung aus dem Softwareregister ausgelesen, wobei die Zahl aus 7 Vorkomma- und 9 Nachkommastellen besteht. Die 7 Vorkommastellen entsprechen der Distanz(d) in cm (vgl. Kapitel 3.2.2). Durch die 9 Nachkommastellen wird der Abstand(d) in einer Genauigkeit von $\frac{1}{500}$ cm dargestellt.
Um den Abstand d in cm auszuwerten, gibt es zwei Möglichkeiten:

1. Mit einer Schiebeoperation den Abstand 'd' um 9 Bits nach rechts schieben. Berechnung anhand eines Beispiels aus der Tabelle 4.3 Zeile 11: Der Tatsächliche Abstand beträgt 16 cm. Der erfasste Abstand beträgt $d = (9277_{10} = 0010010\mathbf{000111101}_2) >> 9 = (18_{10} = 0010000_2)cm$

2. Den erfassten Abstand d durch 500 teilen. Das Ergebnis der Division ist in Integer. Daher gehen die Nachkommastellen des Ergebnisses verloren. Beispiel aus der Zeile 11 der Tabelle 4.3:
   $d = \frac{9277_{10}}{500} = 18,554 = 18cm$

Um den Abstand d in mm zu berechnen, wird d durch 50 geteilt. Beispiel: $d = \frac{9277_{10}}{50} = 185,54 = 185$ mm
Eine Zahl in der Q-Format-Darstellung mit 10 Nachkommastellen hat den Effekt, dass die kleinste mögliche Nachkommastelle $\frac{1}{1000}$ der kleinsten Vorkommastelle der Zahl entspricht. Beispiel Zeile 10 der Tabelle 4.3: Wenn d mit einer Nachkommastelle erweitert wird (d«1),

dann hat d (Fix_17_10) den Wert $d_2 = 100001101001000 = d_{10} = 17224$. Hier sieht man den Effekt, dass die zwei ersten Ziffern ('1' und '7') dem Abstand in Zentimetern entsprechen. Die drei Ziffern ('2', '2' und '4') entsprechen den 10 Nachkommastellen. Die Spalte (" Abstand in cm") der folgenden Tabelle(4.3) entspricht dem tatsächlichen Abstand zwischen dem IR-Sensor und dem Messobjekt(vgl. Abbildung 4.12). Die weiteren zwei Spalten entsprechen der dezimalen und binären Darstellung des erfassten Abstandes(d). Die Spalte($d >> 9$) stellt den erfassten Abstand nach der Schiebeoperation in Zentimetern dar. In der Spalte "d in mm" wird der Abstand d durch Division $\frac{d}{50}$ in Millimeter dargestellt.
Die Spalte "Abweichung" entspricht der Abweichung zwischen dem tatsächlichen Abstand und dem erfassten Abstand "$d >> 9$" in Zentimetern (vgl. Abbildung **??**). Die Spalte "M" entspricht dem Inhalt der 12-Bit Messdaten. Die letzte Spalte entspricht der Messspannung(V_0) des IR-Sensors(vgl. Kapitel 2.3).

Nr	Abstand in cm	d_{10} in $\frac{1}{500}$ cm	d_2 $in\frac{1}{500}$ cm	$d >> 9$ in cm	Abweichung in cm	d in mm	M	V_0 in Volt
1	4	8843	0010001**010001011**	17	13	176	1974	1,56
2	6	6931	0001101**100010011**	13	7	138	2468	1,99
3	7	4340	0001000**011110100**	8	1	86	3119	2,52
4	8	4450	0001000**101100010**	8	0	89	3298	2,63
5	9	4337	0001000**011110001**	8	1	86	3167	2,56
6	10	4641	0001001**000100001**	9	1	92	3001	2,41
7	11	5330	0001010**011010010**	10	1	106	2817	2,27
8	12	6251	0001100**001101011**	12	0	127	2616	2,08
9	14	7439	0001110**100001111**	14	0	148	2350	1,87
10	16	8612	0010000**110100100**	16	0	172	2039	1,66
11	18	9277	0010010**000111101**	18	0	185	1857	1,51
12	20	10031	0010011**100101111**	19	1	200	1687	1,37
13	25	12312	0011000**000011000**	24	1	246	1381	1,12
14	30	15042	0011101**011000010**	29	1	300	1180	0,95
15	35	17710	0100010**100101110**	34	1	354	1046	0,85
16	40	20242	0100111**100010010**	39	1	404	947	0,77
17	45	23102	0101101**000111110**	45	0	462	855	0,69
18	50	25341	0110001**011111101**	49	1	506	793	0,63
19	55	28467	0110111**100110011**	55	0	569	717	0,59
20	60	29965	0111010**100001101**	58	2	599	684	0,55
21	65	32458	0111111**011001010**	63	2	649	633	0,51
22	70	33818	1000010**000011010**	66	4	677	606	0,50
23	75	36839	1000111**111100111**	71	4	736	553	0,46
24	80	39010	1001100**001100010**	76	4	780	517	0,43
25	85	39953	1001110**000010001**	78	7	800	501	0,40
26	90	42245	1010010**100000101**	82	8	844	467	0,38
26	open	15825	0011110**111010001**	31		316	48	0,03

Tabelle 4.3.: *Erfasste Messdaten M, d , V_0 mit SPI-Übertragungsmodus CPOL =1 und SPHA =0 (vgl. Kapitel2.5)*

Der gemessene Abstand aus Spalte "$d >> 9$ " passt zu den berechneten Abständen in Kapitel

3.2.2. Die Polynom-Approximation (Abbildung 3.6) liefert mit M im Intervall von 3200 bis 500 korrekte Ergebnisse.

Das Verhältnis zwischen dem gemessenen Abstand und der Ausgangsspannung V_0 des Infrarotsensors, wie in der Abbildung 4.13 dargestellt, entspricht den Charakteristika der Kennlinie im Datenblatt des GP2D12-Infrarotsensors(vgl. Kapital 2.3). Die maximale Abweichung zwi-

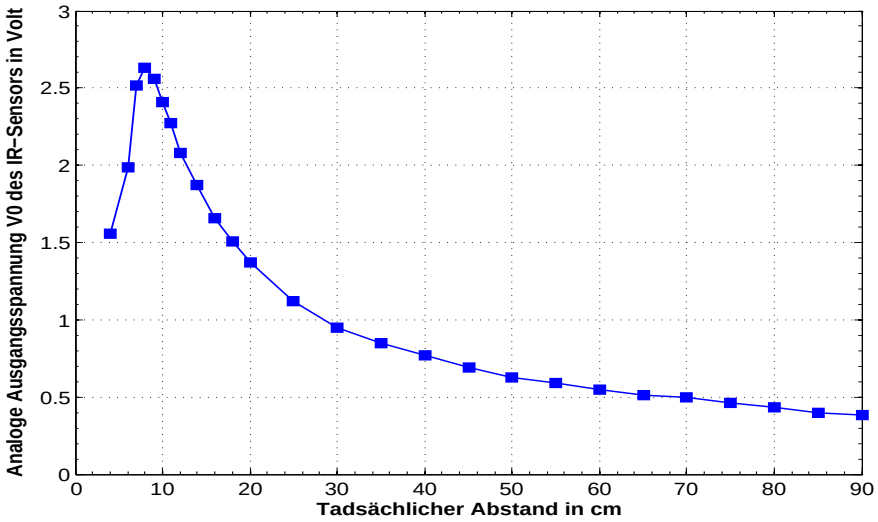

Abb. 4.13.: *Verhältnis zwischen V0 und dem Abstand des IR-Sensors zum Messobjekt*

schen den tatsächlichen Abständen und den erfassten Abständen im Bereich von 8 cm bis 55 cm beträgt 1 cm. Ab 55 cm bis 80 cm sind manche Abweichungen größer als die maximalen Abweichungen des Polynoms 4. Ordnung (vgl. Tabelle 3.4). .

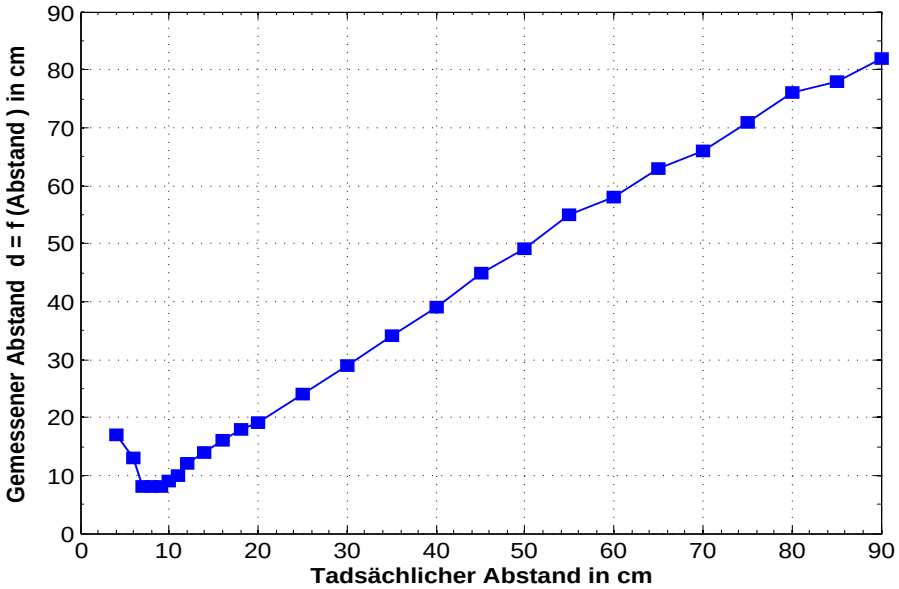

Abb. 4.14.: *Verhältnis zwischen tatsächlichem Abstand (Abstand) und von IR gesessenen Abstand(d) (vgl. Tabelle 4.3)*

5. Ergebnisanalyse der HAW-SPI IP Entwürfe

In diesem Abschnitt wird der Ressourcenbedarf, das Timing-Verhalten des HAW-SPI IPs während der MISO-Datenübertragung und die Auslastung der MicroBlaze bei der Steuerung des Übertragungsvorgang der folgenden Entwürfe des HAW-SPI IP Cores analysiert und ausgewertet.

Entwurf 1: dieser Entwurf des HAW-SPI IP hat folgende Eingenschalten(vgl. Kapitel 3):

- 14 Softwareregistern für Steuer-und Messdaten
- Abtastperiode der HAW-SIP wird durch den externen Timer(XPS-Timer) erzeugt
- Start und Stopp der MISO-Datenübertragung und Auslesen der Messdaten aus den SW-Registern erfolgen in zwei ISRs(vgl. Abbildung 5.1)
- Vier 32-Bit PLB-Zugriffe für das Auslesen des vier 16-Bit Datensatzes aus den SW-Registern des HAW-SPI IPs

Entwurf 2 : Dieser Entwurf des HAW-SPI IP hat folgende Eingenschaften(vgl. Kapitel 4):

- 4 Softwareregistern für Steuer-und Messdaten
- Abtastperiode des HAW-SPI wird durch den integrierten Timer erzeugt
- Start und Stopp der MISO-Datenübertragung erfolgen über den SPI-Controller
- Nur eine ISR für das Auslesen des Messdaten aus den SW-Registern(vgl. Abbildung 5.2)
- zwei 32-Bit PLB-Zugriffe für das Auslesen der vier 16-Bit Datensätze aus den SW-Registern des HAW-SPI IPs

5.1. Ressourcenbedarf

Der HAW-SPI IP in erstem Entwurf verbracht 8 % der von Sparten-3e XC31200E zur Verfügung gestellten Ressourcen(vgl. Tabelle 5.1). Der HAW-SPI IP in dem zweiten Entwurf verbracht im Gegensatz zu dem in erstem Entwurf 5% der Ressourcen(vgl. Tabelle 5.2). Der Grund für Reduzierung der Anzahl der verbrauchten Ressourcen sind: Reduzierung Anzahl der SW-Register, Ersatz des XPS-Timer durch einen im IP Core integrierten Timer.
Der Unterschied des Ressourcenbedarf zwischen den obengenannten HAW-SPI-Entwürfen liegt bei 3% der verfügten Ressourcen. Daher ist der HAW-SPI IP in dem Entwurf in einer SoC-Plattform mit mehreren Beschleuniger-Module geeignet(vgl. Tabelle 5.3).

Ressourcen	benutzt von			Zur Verfügung	benutzt in %
	HAW-SPI IP	XPS-Timer	Gesamt		
Slices	902	344	1246	8672	13 %
Slice Flip Flop	677	298	975	17344	6 %
4 input LUTs	869	358	1227	17344	7 %

Tabelle 5.1.: *Ressourcenbedarf des XPS_Timer IPs und des HAW-SPI IPs in erstem Entwurf mit Xillinx Spartan-3E XC3S1200E FPGAs (vgl. Kapitel 3)*

Ressourcen	benutzt	Zur Verfügung	benutzt in %
Slices	808	8672	9 %
Slice Flip Flop	676	17344	3 %
4 input LUTs	580	17344	3 %

Tabelle 5.2.: *Ressourcenbedarf des HAWSPI IPs mit SPI-Controller und integriertem Timer (vgl. Kapitel 4)*

Ressourcen	HAW-SPI IP mit XPS-Timer	HAW-SPI IP mit integriertem TImer	Unterschied
Slices	1246	808	438
Slice Flip Flop	975	676	299
4 input LUTs	1227	580	647

Tabelle 5.3.: *Ressourcenbedarfsvergleich zwischen den beiden HAW-SPI IP-Entwürfen(zwischen Tabelle 5.1 und 5.2)*

5.2. Timing-Verhalten

Das Ziel ist die Übertragung der Messwerten der vier IR-Sensoren aus den ADU-Schieberegister in SoC-FPGA. Diese MISO-Übertragung erfolgt mit einem SCK-Takt von 12.5 MHz(vgl. Abbildungen 5.1 und 5.2).
Im Gegensatz zu erstem Entwurf wird in dem zweiten Entwurf die Beteiligungszeit des MicroBlaze in dem Übertagungsvorgang reduziert. Der ADU wird nur in MSIO-Übertragungszeit selektiert und getaktet. Unnötiges Takten des ADU durch HAW-SPI-Master-Modul wird vermieden. Diese Vorteile bei dem zweiten Entwurf führt zu dem Energiesparen des SoC-Fahrzeug, was in mobilen Autonomie ein wichtiger Punkt ist.

	HAW-SPI IP **Entwurf 1**	HAW-SPI IP **Entwurf 2**	Unterschied
ADU selektiert	8.96 μs	1.37 μs	7.59μs
Prozessor in ISR	7.96 μs	2.52 μs	5.44μs
IRQ belegt	15.74 μs	5.67 μs	10.07 μs
Anzahl der SCK-Takte	5*16= 80 SCK-Takte	16 SCK-Takte	64 SCK-Takte

Tabelle 5.4.: *Timing-Vergleich zwischen den beiden Entwürfe des HAW-SPI IPs*

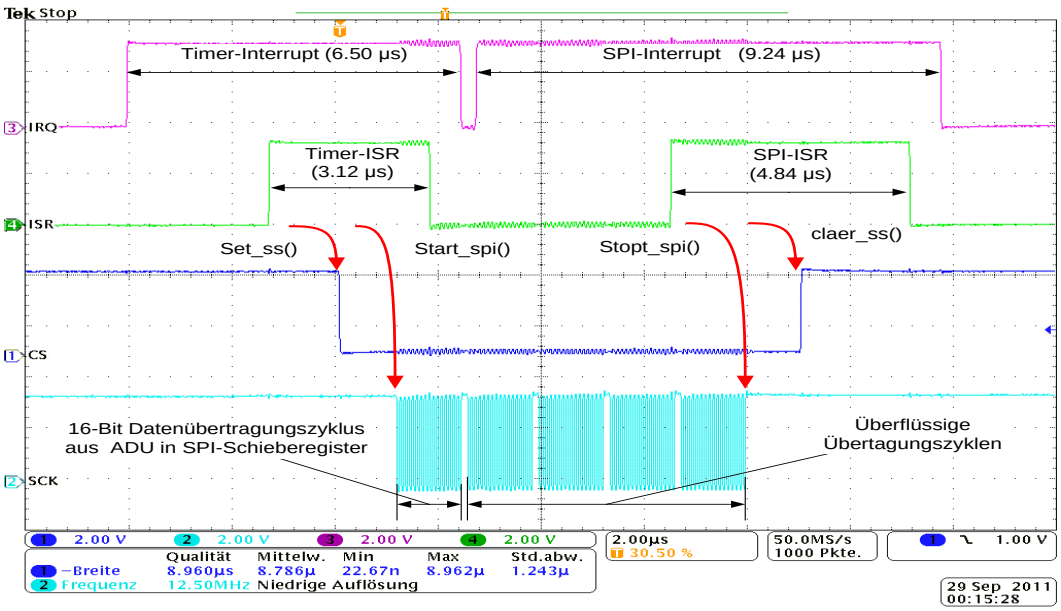

Abb. 5.1.: *Timing-Verhalten des HAW-SPI IPs bei Empfang von 16-Bit Messdaten im erstem Entwurf mit SCK-Frequenz von 12.5MHz*

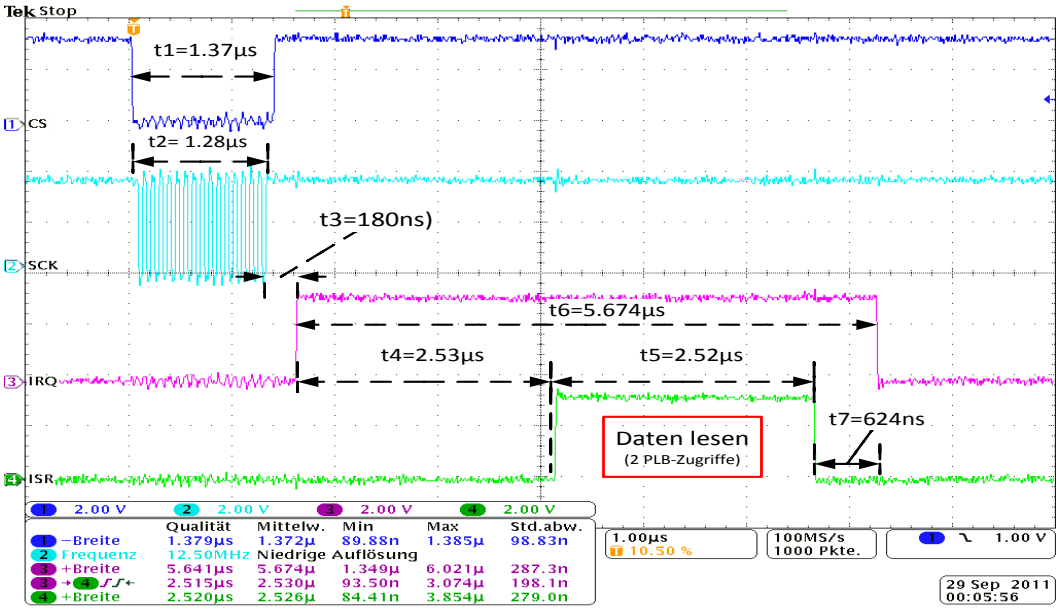

Abb. 5.2.: *Timing-Verhalten des HAW-SPI IPs bei Empfang von 16-Bit Messdaten in zweitem Entwurf mit der SCK-Frequenz von 12.5MHz*

6. Parklückensuche des FZ-Einparkassistenten mit IR-Sensor

Dieses Kapitel stellt den Vorgang eines Einparkassistenten bei der Suche nach einer passenden Parklücke für Einparken des SoC-Fahrzeug vor. Weiterhin wird eine Methode zu Positionierung des SoC-Fahrzeugs beim Start des Einparkvorgangs vorgestellt.

6.1. Berechnung der benötige Parklückenbreite für den SoC-Fahrzeug mit Fahrzeuggeometrie

Die Breite einer Parklücke ist von der Geometrie des Fahrzeuges abhängig. Um eine passende Parklücke für das SoC-Fahrzeuges zu finden, wird die Länge, den maximalen Lenkwinkel und den Abstand zwischen der vorderen und der hinteren Achse des SoC-Fahrzeuges benötigt. Diese Größen das SoC-Fahrzeuges der Hochschule für Angewandte Wissenschaften (HAW) sind vorgegeben(vgl. Tablle 6.1).

Name	Beschreibung	Wert in cm
L	Länge zwischen den Achsen des Fahrzeuges	25.7
FBR	Fahrzeugbreite	20.2
FLV	Abstand Vorderachse zu vorderen Fahrzeugbegrenzung	8
FLH	Abstand Hinterachse zu hinteren Fahrzeugbegrenzung	5.5
FGL	Gesamte Länge des Fahrzeuges	39.2
α_{max}	Maximale Lenkwinkel	$20°$

Tabelle 6.1.: *Fahrzeuggeometrie des SoS-Fahrzeugs*

Mit der Anwendung der Kreistechnik und den gegeben konstanten Größen des FZ-Modells werden den Wenderadius und den Kollisionsradius des Fahrzeugs berechnet [15]. Der Wenderadius(RP) des Fahrzeug lässt sich aus dem rechteckigen Dreieck(D1) wie folgt berechnen(vgl. Abbildung 6.1):

$$tan(\alpha) = \frac{Gegenkathete}{Ankathete} = \frac{L}{RP} \Longrightarrow RP = \frac{L}{tan(\alpha_{max})} \tag{6.1}$$

Der vordere Kollisionsradius(RKA1) und der hintere Kollisionsradius(RAK2) des Fahrzeugs werden aus dem Dreieck(D3) mit Satz des Pythagoras wie folgt berechnet:

$$RKA1 = \sqrt{(RP + \frac{FBR}{2})^2 + (L + FLV)^2} \tag{6.2}$$

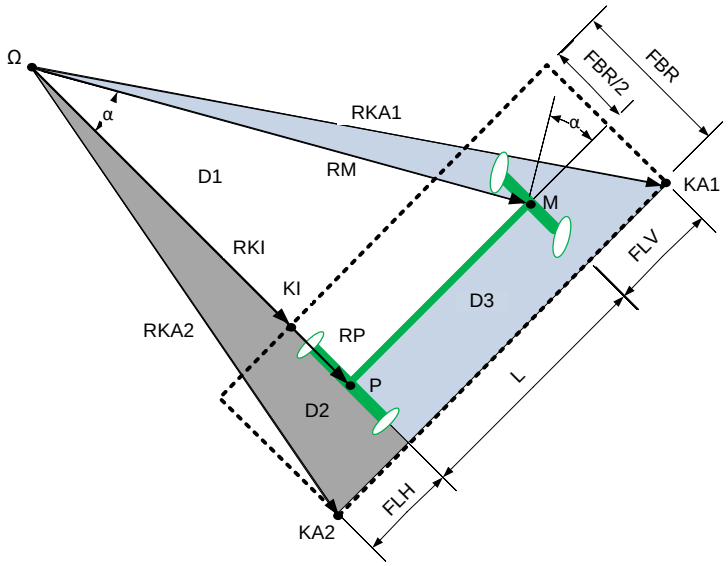

Abb. 6.1.: *Fahrzeuggeometrie,Wende-Radius und Kollision*

$$RKA2 = \sqrt{(RP + \frac{FBR}{2})^2 + (FLV)^2} \tag{6.3}$$

Als Kollisionsradius wird das Maximum aus dem vorderen Kollisionsradius(RKA1) und dem hinteren Kollisionsradius(RK2) gewählt. Wenn das Fahrzeug mit maximalen Lenkwinkel (α_{max}) um einen Kreis mit Radius "RP" fahrt, dann wird eine äußere Kollision mit dem längeren Kollisionsradius verursacht.
Durch das Einsetzen des gegebenen Fahrzeuggeometriegrößen aus der Tabelle 6.1 in den Gleichungen 6.2 , 6.3 und 6.1 werden die unbekannte Größen "RKA" und "RP" berechnet.(vgl: Tabelle 6.2).

Name	Beschreibung	Wert in cm
RP	Wenderadius des zusammengeschoben Hinterrades	70.6
RK1	Innerer Kollisionsradius, RP-FLH	60.5
RKA1	Äußerer Kollisionsradius	87.46
RKA2	Äußerer Kollisionsradius	80.90
RKA	max(RKA1,RKA2)	87.46

Tabelle 6.2.: *Wenderadius und Kollisionsradius des Fahrzeugs*

Für die Suche nach einer passenden Parklücke für das Einparken benötigt der Einparkassistent die Länge der Parklücke, in der das FZ-Modell ohne Kollision in einem oder mehreren Einparkschritten eingeparkt wird. Die maximale Parklücke PL_{max} in der der Einparkassistent den Einparkvorgang in einen Schritt erfolgt, berechnet sich wie folgt(vgl- Abbildung 6.2):

$$PL_{max} = g + FLH \tag{6.4}$$

$$g = \sqrt{RKA^2 - RKI^2} \tag{6.5}$$

Wobei "g" Abstand des rechten Hindernisses zur der Hinterachse des FZ-Modells in der Parklücke(ZIEL) ist(vgl. Abbildung 6.2).

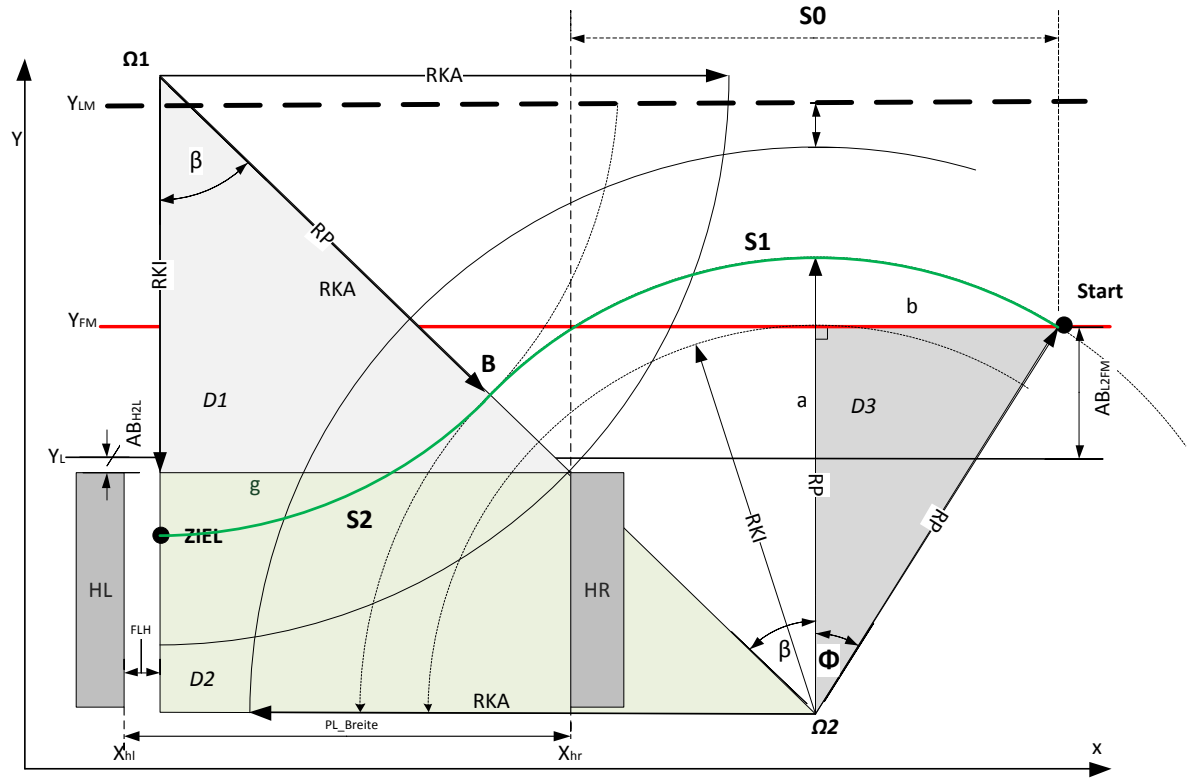

Abb. 6.2.: *Berechnen die Parklückenbreite mit FZ-Model-Geometrie*

6.2. Entwurf und Implementierung eines Parklückenfinder-Software-Modul

In diesem Abschnitt wird ein Software-Modul vorgestellt, mit dem durch die seitliche Abstandmessung mit IR-Sensoren eine Parklücke für Einparken des SoC-Fahrzeug gesucht. Die Suche nach einer Parklücke wird erst dann gestartet, wen das FZ-Modell eine Einparksuche beauftragt hat(Start_suche). Nach der erfolgreichen Suche wird die Länge der Parklücke(PL_breite), den Abstand des SoC-Fahrzeug zu dem rechten Hindernis(H_{hr}) und ein Signal für das Ende der Paralückensuche an das FZ-Modell übergeben. Es wird angenommen , dass die rechte und das linke Hindernis mit gleichem Abstand zur rechter Fahrspurbahn stehen.

Den gefahren Weg wird in dem V-Regler-Modul des SoC-Fahrzeugs mit Hall-Sensoren ge-

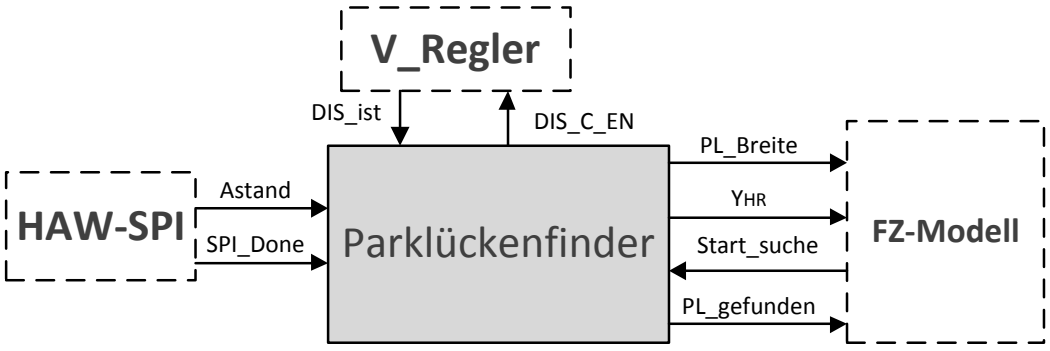

Abb. 6.3.: *Aufbau einen Parklückenfinder mit Abstandsensor(IR) verbunden mit HAW-SPI-IP, V-Regler-Modul und dem FZ-Modell*

messen und die Messdaten stehen am Ausgang(DIE_ist) des V-Regler zur Verfügung[14]. Mit dem Signal "DIS_C_EN" wird V-Regler-Modul beauftragt die Messung des gefahrenen Weges zu starten. Für die seitliche Abstandmessung des SoC-Fahrzug zu dem Hindernis wird der IR-Sensor verwendet. Der entscheidende Punkt im Parklückensuchvorgang ist die Geschicklichkeit des SoC-Fahrzeug. Der IR-Sensor aktualisiert die Messdaten periodisch in 38ms, daher wird die Geschwindigkeit des SoC-Fahrzeuges wird so eingepasst dass bei jedem Abtaten Messwerte aus dem IR-Sensor einen aktuellen Abstand vorliegt.

Geschwindigkeit

$$V = \frac{S}{t} \tag{6.6}$$

Durchschnittliche IR-Sensorzeit für eine Abstandmessung (vgl. Kapitel 2.3) t= 38ms

Die Geschwindigkeit des SoC-Fahrzeuges hängt davon ab, in Welchem Abstand(ΔS) nach

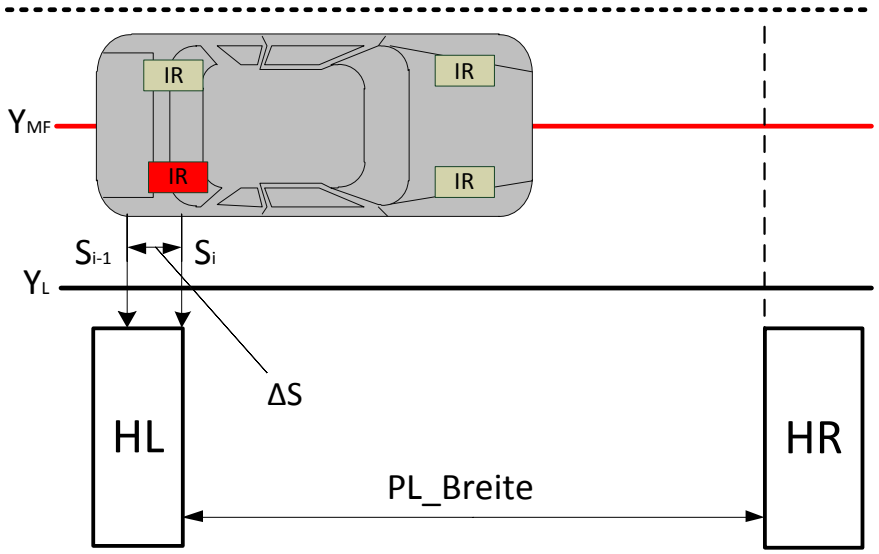

Abb. 6.4.: *Parklückensuchvorgang mit maximalen Geschwindigkeit von 26 cm/s bei einer Abstandsauswertung von 1cm*

einem Hindernis gesucht wird. Beispiel:

ΔS=1cm

$$V_{max} = \frac{\Delta S}{t} = \frac{1cm}{38ms} = \frac{1cm}{0.038s} = 26.31\frac{cm}{s} \tag{6.7}$$

Eine erfolgreiche Parklückensuche fordert eine maximale Geschwindigkeit $V_{max} = 26cm/s$ und HAW-SPI-BUS Abtastrate :

$$HAW - SPI_{abtastrate} = \frac{1}{T} = \frac{1}{38ms} = \frac{1}{0.038s} = 26.31Hz \tag{6.8}$$

Zustandsautomat für die Parkrückenfinder

Die Eingange des Automaten sind:

Abstand: 16-Bit von dem IR-Sensor erfassten Messdaten.

SPI_Done: Entspricht der Abtastperiode des HAW-SPI-BUS. SPI_Done hat erst dann den Wert '1', wenn die aktuellen Messdaten in HAW-SPI SW-Register vorliegen.

DIS_IST: Der gefahrener Weg, der in V-Regler-Modul erfasst und berechnet wird.

Start_suche: Mit diesem Signal startet das FZ-Modell den Parklückensuchvorgang.

Die Ausgänge des Automaten sind:

DIS_C_EN: '1'= Das V-Regler-Modul erfasst den gefahren Weges und legt er den in einem W-Register zu Verfügung; '0'= Das V-Regler-Modul löscht die gerechnete gefahrenen Weg und bei '1' wird es neu berechnet.

PL_Breite: Die Breite der Parklücke.

PL_gefunden: Ein Signal, das dem FZ-Modell benachrichtigt, dass der Parklückenvorgang abgeschlossen ist und der Einparkvorgang ausgeführt werden kann.

Y_{hr}**:** Der Abstand zwischen dem SoC-Fahrzeug von dem rechten Hindernis.

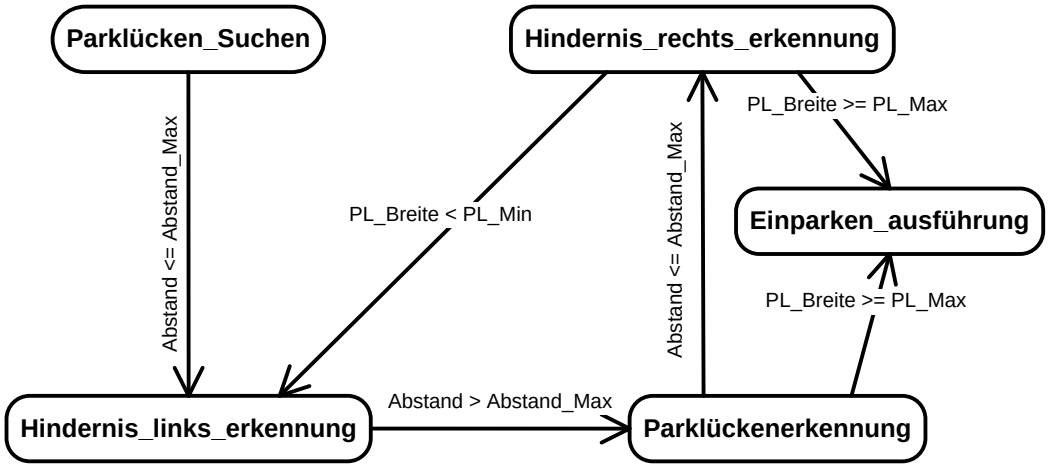

Abb. 6.5.: *Zustandsautomat des Parklückenfinder für den Einparkassistenten*

Die Zustände des Automaten werden wie folgt beschrieben:

Parklücken_Suchen: In diesem Zustand prüft der Automat mit IR-Sensor, ob ein Hindernis in der Seite des SoC-Fahrzeug vorliegt. Falls Ja dann geht der Automat in den Zustand "Hindernis_links_erkenneung".

Hindernis_links_erkennung: In dem Zustand wird es geprüft ob an der Seite des SoC-Fahrzeuges eine Lücke vorkommt. Wenn ja, dann geht der Automat in den Zustand "Parklückenerkennung".

Parklückenerkennung: In diesem Zustand wird das V-Regler-Modul benachrichtigt, dass der gefahrener Weg berechnet werden soll. Falls gefahrener Weg ausreichend für das Einparken ist dann geht der Automat in den Zustand "Einparken_ausführung". Wenn das rechte Hindernis erreicht wird dann geht der Automat in den Zustand "Hindernis_rechts_erkennung".

Hindernis_rechts_erkennung: In diesem Zustand wird die Bereite der Parklücke geprüft, wenn die Parklücke ausreichend ist geht der Automat in den Zustand "Einparken_ausführung" sonst geht der Automat in den Zustand "Hindernis_links_erkennung"; d.h. dass das aktuelle Hindernis als linkes Hinderns ausgewertet wird.

Einparken_ausführung: In diesem Zustand werden die das FZ-Modell informiert(pl_gefunden='1'), dass eine ausreichende Parklücke vorhanden ist und den Einparkvorgang beginnen kann.

6.3. Bestimmung der Startpostion SoC-Fahrzeug bei Einparkvorgang

In dieser Abschnitt stellt eine Methode vor, in der ein Einparkassistent, nachdem die Parklücke gefunden hat, in drei Schritten das SoC-Fahrzeug einparkt(vgl. Abbildung 6.6).

1. Mit konstantem Abstand zu dem rechter Fahrspurbahn die Strecke S0 fahren bis die hintere Achse des SoC-Fahrzeug den Schnittpunkt der Fahrzeugspurmitte(Y_{FM}) mit dem Kreis "$\Omega 2$ im Punkt "Start" erreicht hat.
2. Mit maximalem Lenkwinkel(α_{max}) nach links eine Strecke in Lange 'L'(vgl. Tabelle6.1) rückwärts fahren bis die vordere Achse des SoS-Fahrzeuges den Punkt "Start" erreicht hat. Dann mit maximalem Lenkwinkel(α_{max} nach rechts die Strecke S1(von Start bis Punkt B) rückwärts fahren bis die hintere Achse des SoC-Fahrzeug den Punkt "B" erreicht hat.
3. Im Punkt B mit maximalem Lenkwinkel(α_{max}) nach Links weiter rückwärts die Strecke S2(B bis Ziel) fahren bis die hintere Achse des Fahrzeugs den Punkt "Ziel" erreicht hat.

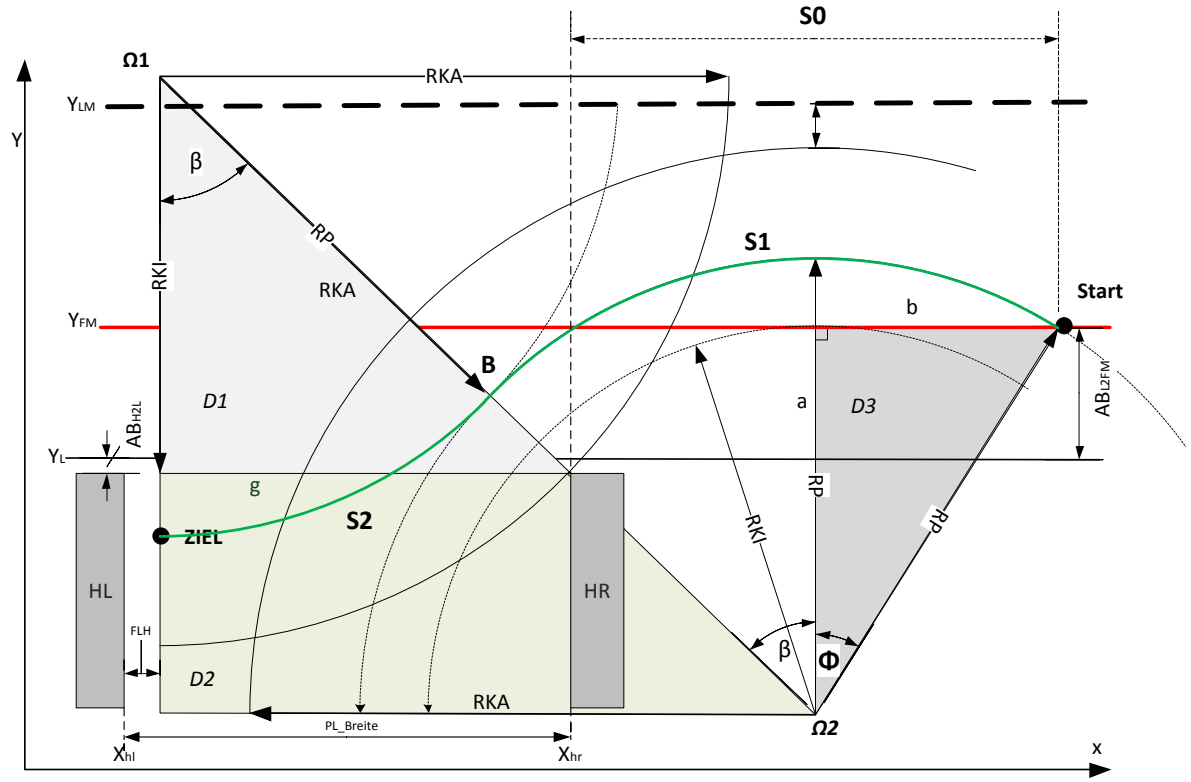

Abb. 6.6.: *Berechnung der Startpostion des SoC-Fahrzeuges und die Strecke von dem Startpunkt bis zum Zielpunkt*

Das SoC-Fahrzeug des SoC-Projekts an der HAW mit dem Fahrspurführungssystem ist in der Lage parallel zu der Fahrbahn fahren [2]. Der Abstand zwischen dem SoC-Fahrzeuges und der rechten Fahrbahn lässt sich von dem Einparkassistenten umstellen. Der gefahrener Weg lässt sich im V-Regler-Modul des SoC-Fahrzeuges messen[14].
Das SoC-Fahrzeug fährt an Y_{FM} entlang. Ab dem Punkt X_{hl} beginnt die Messung der Parklücke und am PunktX_{hr} steht die Breite der Parklücke fest(vgl. Kapitel 6.2).
Im Punkt X_{hr} sind die gemessenen Parklückenbreite(PL_{Breite})(vgl Kapitel **??**) und die berechneten Wege S0,S1,S2 zur Verfügung. Nun soll durch Vergleich den berechneten Weg und den von V-Regler-Modul erfassten Weg soll bis zu dem Startpunkt(Start) gefahren werden. Vorausgesetzt, dass der IR-Sensor , der auf hinterer Achse des SoC-Fahrzeugs,für die seitliche Abstandmessung verwendet werden. Somit liegt das Fahrzeug im Startpunkt mit hinterer Achse genau am Punkt "Start". Die drei Strecke SO werden wie folgt berechnet(vgl. Abbildung 6.6):

So=Die Strecke von dem Punkt X_{hr} bis zum Punkt "Start"

$$S0 = (X_{\Omega 1-\Omega 2} - g) + b \tag{6.9}$$

$$X_{\Omega 1-\Omega 2} = 2RP * \sin(\beta) = 2RP * \frac{g}{RKA} \tag{6.10}$$

Die Variable b wir mit Dreieck D3 wie folgt berechnet(vgl. Abbildung 6.6).

$$b = \sqrt{RP^2 - a^2} \tag{6.11}$$

Da RP schon bekannt ist, soll nur die Variable a aus Dreieck D3 berechnet werden.

$$a = RP - (AB_{L2FM} + (Y_{\Omega 1-\Omega 2} - RKI + AB_{H2L})) \tag{6.12}$$

$$Y_{\Omega 1-\Omega 2} = \sqrt{2RP^2 - X^2_{\Omega 1-\Omega 2}} \tag{6.13}$$

Die Strecken S1 und S2 lassen sich wie folgt berechnen:

$$S1 = \frac{U}{360^\circ} * (\beta + \Phi) \tag{6.14}$$

$$S1 = \frac{U}{360^\circ} * (\beta) \tag{6.15}$$

Der Umfang des Kreises ergibt sich aus Radios dies Kreis und π

$$U = RP * \pi \tag{6.16}$$

$$\Phi = \arcsin \frac{b}{RP} \tag{6.17}$$

$$\beta = \arcsin \frac{g}{RP} \tag{6.18}$$

7. Zusammenfassung

Ziel der Arbeit war die Weiterentwicklung einer SoC-basierte serielle Schnittstelle(HAW-SPI) zum Empfang Infrarot-Messwerten. In dieser Schnittstelle werden aus den erfassten Messwerten von vier auf dem Fahrzeug befindlichen und auf das Hindernis gerichteten GP2D12-Infrarotsensoren reale Abstände zwischen dem Fahrzeug und dem Hindernissen berechnet. Dabei sollte der Ressourcenbedarf, das Timing-Verhalten und der Energieverbrauch des Systems reduziert werden. Die von dem jeweiligen IR-Sensor erfassten Daten werden durch ADUs(PmodAD1) in digitale Werte umgesetzt und über einen SPI-Bus(HAW-SPI) in das SoC übertragen. Dabei überträgt der HAW-SPI die von den vier IR-Sensoren erfassten Messdaten parallel von den ADUs zum SoC.
Für die Umrechnung der erfassten Messdaten in metrische Einheiten wurden die Messdaten mit Polynomen 3., 4. und 5. Ordnung in Matlab approximiert. Diese Polynome wurden miteinander verglichen, um daraus die optimale Umrechnungsfunktion wählen zu können. Als Kompromiss zwischen Genauigkeit, HW-Ressourcenbedarf und Rechenzeit wurde das Polynom 4. Ordnung als Polynom-Accelerator-Modul modelliert.
Der HAW-SPI wurde in Bezug auf den Ressourcenbedarf des IP-Cores und des Timing-Verhalten bei der Datenübertagung analysiert. Es ist festgestellt worden, dass die Erzeugung der Abtastrate durch den XPS-Timer und die Verwendung der SW-Register für die Koeffizienten des Polynoms einen hohen Ressourcenbedarf verursachen. Durch die Kombination von zwei ISRs für den Start und den Stopp des Empfangsvorganges und das Auslesen der 16-Bit Messwerte aus den HAW-SPI-SW-Registern mit 32-PLB-Zugriffen ist der MicroBlaze Prozessor für eine Zeit von $15.74\mu s$ belegt
Für die Erzeugung der HAW-SPI Abtastrate wurde ein 21-Bit Timer in den HAW-SPI IP-Core integriert. Die Anzahl der SW-Register wurde von 14-SW-Register(Parameter des Polynom-Moduls,Status-und Steuerdaten und erfasste Messdaten) auf 4 SW-Register(2 für Steuer und Statusdaten und 2 für Messdaten) reduziert. Die Parametrisierung des Polynom-Moduls erfolgt über Top-Entity des HAW-SPI IPs. Für den Start und den Stopp des Empfangsvorgangs des HAW-SPI-Busses wurde ein SPI-Controller entwickelt und als als Hardware-Modell modelliert. Das Auslesen der erfassten Messwerte aus den HAW-SPI-SW-Registern erfolgt mit zwei 32-Bit-PLB-Zugriffen des MicroBlaze. Dadurch wurde die Beteiligungszeit des MicroBlaze-Prozessors am Auslesevorgang der Software-Register des HAW-SPI IP auf $5.67\mu s$ reduziert und somit mehr Prozessorzeit für das Ausführen anderer Tasks des SoC gewonnen.
Da die Steuerung des Empfangsvorgangs des HAW-SPI durch den SPI-Controller erfolgt, wird die Anzahl der ISRs, die für das Auslesen der Messdaten aus dem HAW-SPI SW-Register benötigt werden, von zwei auf eine ISR reduziert.
Es wurde ein Software-Modul für den Einparkrassisten entworfen, welches durch das Auswerten von IR-Sensoren erfassten Messwerten dem SoC-Fahrzeug ermöglicht eine passende Parklücke zu finden. Außerdem wurde eine Methode entwickelt, die eine Startposition des Fahrzeugs für den Einparkvorgang anhand der Geometrie und des Lenkwinkels des SoC-Fahrzeugs und der Auswertung des gefahrenen Weges ermittelt.

Literaturverzeichnis

[1] "Christian Härtwig". Sensorikintegration in einer fpga-basierte sox-plattform für anwendung in autonomen fahrzeugfunktionen. Master's thesis, "Hochschule für angewandte Wissenschaften Hamburg", 2010.

[2] Cristian Schneider. Ein system-on-chip-basiertes fahrspurführungssystem. Master's thesis, Hochschule für angewandte Wissenschaften Hamburg, 20011.

[3] ct Bot Wiki. Distanzsensoren. URL: http://wiki.ctbot.de/index.php/Distanzsensoren. Stand: 25. November 2011.

[4] Digilint. Digilent nexys2 board refernece manual. URL: http://www.digilent.org/Data/Products/NEXYS2/Nexys2_rm.pdf. Stand: 25. November 2011.

[5] UNI Erlangen-Nuremberg. VHDL-Glossar. URL:http://www.vhdl-online.de/glossary/deutsch/g_9.htm. Stand: 25. November 2011.

[6] Saman Farshbaf Masalehdan. Abstandsregelung für ein autonomes fahrzeug implementiert auf einer fpga basierten soc plattform. Master's thesis, Hochschule für angewandte Wissenschaften Hamburg, 2009.

[7] HAW Hamburg. FAUST Fahrerassistenz- und Autonome Systeme. URL: http://www.informatik.haw-hamburg.de/faust.html. Stand: 25. November 2011.

[8] Digilent Inc. Digilent pmodad1 analog to digital module converter board, reference manual. URL: http://www.digilentinc.com/Data/Products/PMOD-AD1/PmodAD1_rm.pdf. Stand: 25. November 2011.

[9] Bernd Schwarz Jürgen Reichart. *VHDL-Synthese*. Oldenburg Verlag, 2007.

[10] MathWorks. Polynomial curve fitting. URL:http://www.mathworks.de/help/techdoc/ref/polyfit.html. Stand: 25. November 2011.

[11] National Semiconductor. Adcs7476. URL: http://www.national.com/ds/DC/ADCS7476.pdf.pdf. Stand: 25. November 2011.

[12] sensortechnic". SPI-Bus-Kommunikation. URL: http://www.sensortechnics.com/download/AN_SPI-Bus-HCE_D_11156.pdf. Stand: 25. November 2011.

[13] Sharp. GP2D12/GP2D15 General Purpose Type Distance Measuring Sensors. URL: http://pdf1.alldatasheet.com/datasheet-pdf/view/84019/SHARP/GP2D12.html. Stand: 25. November 2011.

[14] Alexander Timotschinko. Implementierung einer geschwindigkeitsreglung als prozessorelement auf einer soc-plattform für ein autonomes fahrzeug. Master's thesis, Hochschule für angewandte Wissenschaften Hamburg, 20011.

[15] Torsten Alpers. Modellierung eines einparkassistenten für autonomes fahrzeug implementiert auf einer soc-plattform. Master's thesis, Hochschule für angewandte Wissenschaften Hamburg, 2010.

[16] Marco Platzer und Lother Thiele. *Hardware/Software Codesign*. 2006/2007.

[17] Wikipedia. Ip-core. URL: http://de.wikipedia.org/wiki/IP-Core. Stand: 25. November 2011.

[18] Wikipedia. System-on-a-Chip. URL: http://de.wikipedia.org/wiki/System-on-a-Chip. Stand: 25. November 2011.

[19] Wikipedia. Triangulation. URL: http://de.wikipedia.org/wiki/Triangulation_(Messtechnik). Stand: 25. November 2011.

[20] "Xillinx. Embedded system design flow workshop and teaching materials, edk 13.2 overview. URL: http://www.xilinx.com/university/workshops/embedded-system-design-flow/index.htm. Stand: 25. November 2011.

[21] Xillinx. Microblaze processor reference guide. URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/mb_ref_guide.pdf. Stand: 25. November 2011.

[22] Xillinx. Spartan-3e fpga family: Data sheet. URL: http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf. Stand: 25. November 2011.

[23] Xillinx. Xilinx platform studio (xps). URL: http://www.xilinx.com/tools/xps.htm. Stand: 25. November 2011.

[24] Xillix. Processor local bus(plb) product specification. URL: http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf. Stand: 25. November 2011.

[25] Xillix. Xps interrupt controller product specification. URL:http://www.xilinx.com/support/documentation/ip_documentation/xps_intc.pdf. Stand: 25. November 2011.

Tabellenverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

AD1	ADCS7476
ADU	Analog Digital Umsetzer
BRAM	Block RAM
BSB	Base System Builder
CEN	Controller Enable
CLB	Configurable Logic Block
CPHA	Clock Phase
CPOL	Clock Polarity
CS	Chip Select
DCM	Digital Clock Manager
DM	Dedicated Multiplier
EDK	Embedded Development Kit
HAW	Hochschule für Angewandte Wissenschaften
HAWSPICR	HAW-SPI Control Register
IC	Integrierter Schaltkreis(integratd cricuit)
IOB	Input/Out Block
IP	(Intellectual Property
IR	GP2D12 Infrarotsensor
IRQ	Interrupt Request
LMB	Local Memory Bus
LSB	Lost Significant Bit
LUT	Lookup Table
MISO	Master In Slave Out
MOSI	Master Out Slave In
MSB	Most Significant Bit
MSS	Manual Slave Select Assertion Enable
MTI	Master Transaction Inhibit
PLB	Processor Local Bus

PSD	Position Sensitive Device
PTE	Polynom Toggle Enable
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTL	Register Transfer Level
SCK	SPI Cock Ausgang
SDRAM	Synchronous Dynamic Random Access Memory
SoC	System on Chip
SPE	SPI System Enable
SPI	Serial Peripheral Interface
SS	Slave Select
XPS	Xilinx Platform Studio

Glossar

Codesign

Der Begriff Codesign wird häufig als *Integrierter Entwurf von Systemen, bestehend aus sowohl Hardware- als auch Softwarekomponenten* aufgefasst. Hardware/Software Systeme existieren bereits seit vielen Jahren. Neu sind jedoch Entwurfsmethoden, die es erlauben, HW- und SW-Komponenten eines System gemeinsam zu entwerfen.
Die Vorsilbe CO erlaubt zahlreiche Interpretationen[16]:

- co(zusammen): Codesign bedeutet den gemeinsamen Entwurf von HW und SW.
- *coordinated*(koordiniert): Codegsign unterstützt einen systematischen Entwurfsfluss für HW/SW-Systeme.
- *concurrent*(nebenläufig): HW-und SW-Komponenten eines Systems arbeiten nebenläufig.
- *complex* (Komplex): Codesign Methoden sind vor allem für komplexe Systeme notwendig.
- *correct*(korrekt): Codesign soll zu korrekten HW/SW System führen.

Generic

Eine Schnittstellen- constant , die in einer block -Überschrift einer block -Anweisung, einer Komponenten- declaration oder einer entity declaration festgelegt wird. Generics unterstützen einen Kanal für statische (static) Informationen, die zu einem block der eigenen Umgebung weitergeleitet werden. Anders dagegen constant , der Wert eines generic kann dennoch extern geliefert werden und zwar entweder innerhalb einer Komponenteninstanziierung oder innerhalb einer configuration specification [5].

Intellectual Property

Als IP-Core (intellectual property core) wird ein wiederbenutzbarer Teil eines Chipdesigns (im Sinne von Bauplänen) in der Halbleiterindustrie bezeichnet. Diese enthält das Geistige Eigentum des Entwicklers/Herstellers. Es gibt Unternehmen, die sich darauf spezialisiert haben, Teile oder auch ganze integrierte Schaltkreise zu entwerfen und Lizenzen dieser Designs zu verkaufen. So kann man z. B. einen Prozessor als fertige Einheit erwerben, um ihn dann in einer eigenen Entwicklung (z. B. als ASIC oder in einem FPGA) zu verwenden [17].

LSB

Analog dem MSB besitzt das LSB-Bit den niedrigsten Stellenwert 1 (Multiplikation mit 2^0.)

MISO
Master in Salve out Datenübertragung. Wird auch SDO (Serial Data out) genannt

ModelSim
Modelsim ist eine mächtige Simulationsumgebung für HDLs. Diese erlaubt die taktsynchrone oder timinggenaue Simulation von digitalen Logikelementen. Anders, als reine Logiksimulatoren erlaubt Modelsim auch das Berechnen und Darstellen von Analogwerten, da Integer und Dezimaltypen intern als normale Variablen (Real) gehandhabt werden. Somit lassen sich auch analoge Modelle verwenden, wenn man sie zuvor in eine gerasterte digitale Darstellung (Signalbus oder Integer überführt.

MOSI
Master out Slave in Tatenübertagung. Wrid auch SDI (Serial Data in) genannt.

MSB
Das Bit, das innerhalb der Zahl an der Stelle mit dem höchsten Stellenwert steht. Bei einer n-stelligen Binärzahl ist also dasjenige Bit das MSB-Bit, das an der n-ten Stelle steht(Multiplikation mit 2^{n-1}).

SDATA
Die 16-Bit Daten in dem Schieberegister des Slave Devices(ADU).

SoC-Fahrzeug
Das SoC-Fahrzeug, das im Rahmen des SoC-Projekts an der Hochschule für Angewandte Informatik Hamburg verendet wird.

SPI
Das Serial Peripheral Interface (kurz SPI) ist ein von Motorola entwickeltes Bus-System mit einem sehr lockeren Standard für einen synchronen seriellen Datenbus, mit dem digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbunden werden können

SS
Slave select bzw. chip Select

TestBench
TestBench generiert Eingangs Stumulis eins VHDL-Models und überprüft die Ausgänge für das richtige Verhalten.

Triangulationsprinzip
Triangulation ist eine Methode der optischen Abstandsmessung mit der trigonometrischen Funktion.[19]

Xillinx
Xilinx, Inc. (NASDAQ: XLNX) ist der weltgrößte Entwickler und Hersteller von programmierbaren Logik-ICs, sogenannten Field-Programmable Gate Arrays (FPGAs). Die Firma wurde von Ross Freeman (FPGA-Erfinder), Bernie Vonderschmitt (Fabless-Pionier) und Jim Barnett 1984 gegründet und in Silicon Valley angesiedelt. Heute ist der Hauptsitz der Firma in San Jose, Kalifornien, der europäische Hauptsitz ist Dublin, Irland, und der asiatische Sitz befindet sich in Singapur. Xilinx besitzt keine eigene Halbleiterfertigung, sondern ist ein Fabless-IC-Hersteller.

XPS

Xilinx Platform Studio (XPS) ermöglicht den Hardware-Entwicklern hoch kundenspezifische Embedded-Prozessor-Systeme in Xilinx FPGAs zu erstellen und integrieren. XPS ist der grafische Gestaltung bzw: Assistent, der den Benutzer durch die notwendigen Schritte zur AXI odr PLB basierten Prozessor-Systeme in wenigen Minuten zusammen zu erstellen führt [23].

Symbolverzeichnis

T_{sck}	Periodendauer der SPI-Clock
PL_{max}	Maximale Parklückenbreite, in der der Einparkassistent das FZ-Modell ohne Kollision mit Hindernis einparken kann
V_0	Ausgangsspannung des IRs (0 bis 3.3 Volt)
V_{cc}	Spannung an den Kollektoren, bei bipolaren ICs positive Versorgungsspannung

A. VHDL-Code

A.1. VHDL-Code für den SPI-Controller des HAW-SPI IPs

```
1  ----------------------------------------------------------------------
2  ----Filename:\MyProcessorIPLib\pcores\haw_spi_v1_00_a\hdl\vhdl\SPI-Controller
       .vdl
3  ----Description:     SPI-Controller fuer HAW-SPI-IP
4  ----Date:            Fri Nov 20 11:36:56 2011
5  ----Designer :       Walli Ahad
6  ----------------------------------------------------------------------
7  library
8  ieee;
9    use ieee.std_logic_1164.all;
10   use ieee.numeric_std.all;
11
12 use ieee.std_logic_1164.all;
13 use ieee.std_logic_arith.all;
14 use ieee.std_logic_unsigned.all;
15
16 entity spi_controller is
17     generic(
18         C_NUM_CR_BITS            : integer
19     );
20     port (
21         reset            : in  std_logic;
22         clk              : in  std_logic;
23         cen              : out std_logic;
24         timer_done       : in  std_logic;
25         spi_done         : in  std_logic;
26         ss               : out std_logic;
27         cont2reg_data    : out std_logic_vector(C_NUM_CR_BITS-1 downto 0 );
28         reg2cont_data    : in  std_logic_vector(C_NUM_CR_BITS-1 downto 0 );
29         cont2spi_data    : out std_logic_vector(4 downto 0)
30     );
31 end entity spi_controller;
32
33
34 architecture archi of spi_controller is
35     type   states is(idle,wait_td,start_spi,wait_sd,stop_spi);--
           Aufzählungstyp
36     signal state , next_state               : states;         --Prozess-
           Kummunikation
37     signal cen_s,ss_s,ss_asm,spe_asm     : std_logic;
38     signal control_data_s                : std_logic_vector(4 downto 0);
39 begin
40    cont2reg_data        <= cen_s&ss_s&control_data_s;
```

```
41
42
43  cen_s                   <= reg2cont_data(6);
44  timer_en                <= cen_s;
45  control_data            <= control_data_s;
46  ss                      <= ss_s;
47  MUX: process(reg2cont_data,spe_asm,ss_asm,cen_s)
48  begin
49      if(cen_s='1')then                    -- conterollr enable ?
50          ss_s<=ss_asm;                    -- ss aus dem automat
51          control_data_s(0)<=spe_asm;     -- spe aus dem automat
52          control_data_s(4 downto 1)<=reg2cont_data(4 downto 1);
53      else
54          control_data_s<=reg2cont_data(4 downto 0);
55          ss_s<=reg2cont_data(5);         -- ss aus dem sw-register
56      end if;
57  end process MUX;
58
59  ------------------------------------------------------------------------
60  ------------------------------------------------------------------------
61  Folgezustandberechen: process(state,timer_done,spi_done,cen_s)
62  begin
63      next_state<=state;
64      case state is
65          when idle =>
66              ss_asm   <= '1';         -- salve not select (ADU not selected)
67              spe_asm  <= '0';         -- MISO-Übertagung stoppen
68              if(cen_s='1')then
69                  next_state<=wait_td;
70              end if;
71          when wait_td =>
72              ss_asm   <= '1';         -- salve not select (ADU not selected)
73              spe_asm  <= '0';         -- MISO-Übertagung stoppen
74              if(cen_s='1')then           -- Controller in Automatikmodus
75                  if(timer_done='1')then -- auf timer_done warten
76                      next_state <= start_spi;
77                  end if;
78              else
79                  next_state<=idle;
80              end if;
81          when start_spi =>   -- salve select (ADU select)
82              ss_asm   <= '0';
83              spe_asm  <= '0';
84              next_state<=wait_sd;
85          when wait_sd =>     -- MISO-Übertagung starten
86              ss_asm   <= '0';
87              spe_asm  <= '1';
88              if(spi_done='1')then -- auf spi_done warten
89                   next_state<=stop_spi;
90              end if;
91          when stop_spi =>    -- MISO-Übertagung stoppen
92              ss_asm   <= '0';
93              spe_asm  <= '0';
94              next_state<=wait_td;
95          end case;
96  end process Folgezustandberechen;
97  Zustandsaktualisierung: process(clk)
```

```
98  begin
99      if clk'event and clk='1' then
100         if (reset = '1') then
101             state <= idle;
102         else
103             state <= next_state;
104         end if;
105     end if;
106 end process Zustandsaktualisierung;
107 end archi;
```

Listing A.1: *VHDL-Code für den SPI-Controller des HAW-SPI IPs*

B. C-Code

B.1. Header Datei des Softwaretreibers des HAW_SPI IPs

```
/*****************************************************************************
* Filename:          MyProcessorIPLib/drivers/haw_spi_v1_00_a/src/haw_spi.h
* Version:           1.00.a
* Description:       haw_spi Driver Header File
* Date:              Fri Jul 15 11:36:56 2011
*****************************************************************************/

#ifndef HAW_SPI_H
#define HAW_SPI_H

/**************************** Include Files ****************************/
#include "xbasic_types.h"
#include "xstatus.h"
#include "xil_io.h"
#include "xgpio.h"
/*************************Globale Varibalen*****************************/
/*Erfassten Messdaten der IR-Sensoren IR1=reg1,IR2=Reg2,IR3=reg3 IR4=Reg4*/
volatile Xuint16 reg1, reg2, reg3, reg4;
volatile Xuint16 spi_done;

/************************** Constant Definitions **************************/
#define HAW_SPI_USER_SLV_SPACE_OFFSET (0x00000000)
#define HAW_SPI_CR_OFFSET (HAW_SPI_USER_SLV_SPACE_OFFSET + 0x00000000)
#define HAW_SPI_SR_OFFSET (HAW_SPI_USER_SLV_SPACE_OFFSET + 0x00000004)
#define HAW_SPI_RCV1_0_OFFSET (HAW_SPI_USER_SLV_SPACE_OFFSET + 0x00000008)
#define HAW_SPI_RCV3_2_OFFSET (HAW_SPI_USER_SLV_SPACE_OFFSET + 0x0000000C)

/*****************************************************************************
 *  HAW_SPI Control (HAWSPCR)-Register Masks
 *****************************************************************************/
#define HAW_SPI_CR_SPI_ENABLE_MASK              0x1
#define HAW_SPI_CR_CLK_POLARITY_MASK            0x2
#define HAW_SPI_CR_CLK_PHASE_MASK               0x4
#define HAW_SPI_CR_MANUAL_SS_MASK               0x8
#define HAW_SPI_CR_TRANS_INHIBIT_MASK           0x10
#define HAW_SPI_CR_SLAVE_SELCT_MASK             0x20
#define HAW_SPI_CR_TIMER_EN_MASK                0x40
#define HAW_SPI_CR_POLY_ENABLE_TOGGLE_MASK      0x80
```

```
39
40  /*****************************************************************
41   *  HAW_SPI Control(HAWSPISR)-Register Masks
42   *****************************************************************
       */
43  #define HAW_SPI_SR_RX1_EMPTY_MASK          0x1    /**< Receive Reg1 is empty */
44  #define HAW_SPI_SR_RX1_FULL_MASK           0x2    /**< Receive Reg1 is full  */
45  #define HAW_SPI_SR_RX2_EMPTY_MASK          0x4    /**< Receive Reg2 is empty */
46  #define HAW_SPI_SR_RX2_FULL_MASK           0x8    /**< Receive Reg2 is full  */
47  #define HAW_SPI_SR_RX3_EMPTY_MASK          0x10   /**< Receive Reg3 is empty */
48  #define HAW_SPI_SR_RX3_FULL_MASK           0x20   /**< Receive Reg3 is full  */
49  #define HAW_SPI_SR_RX4_EMPTY_MASK          0x40   /**< Receive Reg4 is empty */
50  #define HAW_SPI_SR_RX4_FULL_MASK           0x80   /**< Receive Reg4 is full  */
51  #define HAW_SPI_SR_RX1_DRR_Overrun_MASK    0x100  /**< Receive Reg1 Overrun  */
52  #define HAW_SPI_SR_RX2_DRR_Overrun_MASK    0x200  /**< Receive Reg2 Overrun  */
53  #define HAW_SPI_SR_RX3_DRR_Overrun_MASK    0x400  /**< Receive Reg3 Overrun  */
54  #define HAW_SPI_SR_RX4_DRR_Overrun_MASK    0x800  /**< Receive Reg4 Overrun  */
55
56  /*****************************************************************
57   * Interrupt Controller Space Offsets
58   * -- INTR_DGIER : device (peripheral) global interrupt enable register
59   * -- INTR_ISR   : ip (user logic) interrupt status register
60   * -- INTR_IER   : ip (user logic) interrupt enable register
61   *****************************************************************
       */
62  #define HAW_SPI_INTR_CNTRL_SPACE_OFFSET (0x00000100)
63  #define HAW_SPI_INTR_DGIER_OFFSET (HAW_SPI_INTR_CNTRL_SPACE_OFFSET + 0
        x0000001C)
64  #define HAW_SPI_INTR_IPISR_OFFSET (HAW_SPI_INTR_CNTRL_SPACE_OFFSET + 0
        x00000020)
65  #define HAW_SPI_INTR_IPIER_OFFSET (HAW_SPI_INTR_CNTRL_SPACE_OFFSET + 0
        x00000028)
66  /*****************************************************************
67   * Interrupt Controller Masks
68   * -- INTR_TERR_MASK : transaction error
69   * -- INTR_DPTO_MASK : data phase time-out
70   * -- INTR_IPIR_MASK : ip interrupt requeset
71   * -- INTR_RFDL_MASK : read packet fifo deadlock interrupt request
72   * -- INTR_WFDL_MASK : write packet fifo deadlock interrupt request
73   * -- INTR_IID_MASK  : interrupt id
74   * -- INTR_GIE_MASK  : global interrupt enable
75   * -- INTR_NOPEND    : the DIPR has no pending interrupts
76   *****************************************************************
       */
77  #define INTR_TERR_MASK (0x00000001UL)
78  #define INTR_DPTO_MASK (0x00000002UL)
79  #define INTR_IPIR_MASK (0x00000004UL)
80  #define INTR_RFDL_MASK (0x00000020UL)
81  #define INTR_WFDL_MASK (0x00000040UL)
82  #define INTR_IID_MASK (0x000000FFUL)
83  #define INTR_GIE_MASK (0x80000000UL)
84  #define INTR_NOPEND (0x80)
85  #define INTR_DRRIE_MASK (0x00000001)
86  #define INTR_DRR_FULL_MASK (0x00000001)
87  #define DRR_INTR (0x00000001)
88  /*************************** Type Definitions ***************************/
89
```

```
90  /**************** Macros (Inline Functions) Definitions ****************/
91  /**
92   * Write a value to a HAW_SPI register. A 32 bit write is performed.
93   * If the component is implemented in a smaller width, only the least
94   * significant data is written.
95   * @param    BaseAddress is the base address of the HAW_SPI device.
96   * @param    RegOffset is the register offset from the base to write to.
97   * @param    Data is the data written to the register.
98   * C-style signature:
99   * void HAW_SPI_mWriteReg(Xuint32 BaseAddress, unsigned RegOffset, Xuint32
         Data)
100  */
101 #define HAW_SPI_mWriteReg(BaseAddress, RegOffset, Data) \
102     Xil_Out32((BaseAddress) + (RegOffset), (Xuint32)(Data))
103 /**
104  * Read a value from a HAW_SPI register. A 32 bit read is performed.
105  * If the component is implemented in a smaller width, only the least
106  * significant data is read from the register. The most significant data
107  * will be read as 0.
108  * @param    BaseAddress is the base address of the HAW_SPI device.
109  * @param    RegOffset is the register offset from the base to write to.
110  * @return   Data is the data from the register.
111  * C-style signature:
112  * Xuint32 HAW_SPI_mReadReg(Xuint32 BaseAddress, unsigned RegOffset)
113  */
114 #define HAW_SPI_mReadReg(BaseAddress, RegOffset) \
115     Xil_In32((BaseAddress) + (RegOffset))
116
117 /************************** Function Prototypes ****************************/
118 void haw_spi_intr_handler(void * baseaddr_p);
119  void HAW_SPI_Initialize(void * baseaddr_p);
120  void HAW_SPI_Start(void * baseaddr_p);
121  void HAW_SPI_Stop(void * baseaddr_p);
122  void HAW_SPI_Set_SS(void * baseaddr_p);
123  void HAW_SPI_Clear_SS(void * baseaddr_p);
124  void HAW_SPI_Read_4in2_IRs(void);
125  void HAW_SPI_Toggle_PTE_1(void);
126  void HAW_SPI_Toggle_PTE_0(void);
127
128 /**
129  * Enable all possible interrupts from HAW_SPI device.
130  * @param    baseaddr_p is the base address of the HAW_SPI device.
131  * @return   None.
132  * @note     None.
133  */
134 void HAW_SPI_EnableInterrupt(void * baseaddr_p);
135
136 #endif /** HAW_SPI_H */
```

Listing B.1: *Header Datei des Softwaretreibers des HAW_SPI IPs*

B.2. C-Code des Softwaretreibers des HAW_SPI IPs

```
1  /************************************************************************
```

```
 * Filename    :    \MyProcessorIPLib/drivers/haw_spi_v1_00_a/src/haw_spi.c
 * Version     :    1.00.a
 * Description:    haw_spi Driver Source File
 * Date        :    Fri Jul 15 11:36:56 2011
 ****************************************************************************/

/***************************** Include Files *******************************
    */

#include "haw_spi.h"
#include <stdio.h>
#include <xparameters.h>
#include "xgpio.h"
/************************** Function Definitions ***************************
    */

/****************************************************************************
 * Enable all possible interrupts from HAW_SPI device.
 * @param   baseaddr_p is the base address of the HAW_SPI device.
 * @return  None.
 */
void HAW_SPI_EnableInterrupt(void * baseaddr_p) {
    Xuint32 baseaddr;
    baseaddr = (Xuint32) baseaddr_p;
    /* Enable all interrupt source from user logic.*/
    HAW_SPI_mWriteReg(baseaddr, HAW_SPI_INTR_IPIER_OFFSET, 0x00000001);

    /* Set global interrupt enable.*/
    HAW_SPI_mWriteReg(baseaddr, HAW_SPI_INTR_DGIER_OFFSET, INTR_GIE_MASK);
}
/****************************************************************************
 *
 */
void haw_spi_intr_handler(void * baseaddr_p) {
    XGpio_WriteReg(XPAR_ISR_GPIO_BASEADDR,1,1);
    Xuint32 IntrStatus, baseaddr;
    baseaddr = (Xuint32) baseaddr_p;

    IntrStatus = HAW_SPI_mReadReg(baseaddr,HAW_SPI_INTR_IPISR_OFFSET);
    if ((IntrStatus & INTR_DRR_FULL_MASK) == DRR_INTR) {
        //HAW_SPI_Stop((void *) XPAR_HAW_SPI_0_BASEADDR);
        //HAW_SPI_Clear_SS((void *) XPAR_HAW_SPI_0_BASEADDR);
        HAW_SPI_Read_4in2_IRs();
        HAW_SPI_mWriteReg(baseaddr,HAW_SPI_INTR_IPISR_OFFSET,DRR_INTR);
        spi_done=1;
    }
    XGpio_WriteReg(XPAR_ISR_GPIO_BASEADDR,1, 0);
}

void HAW_SPI_Initialize(void * baseaddr_p) {
    Xuint32 baseaddr, ctrlReg;
    baseaddr = (Xuint32) baseaddr_p;
    HAW_SPI_mWriteReg(baseaddr, HAW_SPI_CR_OFFSET,0);
    ctrlReg = HAW_SPI_mReadReg(baseaddr,HAW_SPI_CR_OFFSET);
    HAW_SPI_mWriteReg(baseaddr, HAW_SPI_CR_OFFSET, ctrlReg |
        HAW_SPI_CR_CLK_POLARITY_MASK);
```

```
55      HAW_SPI_mWriteReg(baseaddr, HAW_SPI_CR_OFFSET, ctrlReg |
            HAW_SPI_CR_CLK_PHASE_MASK);
56      ctrlReg = HAW_SPI_mReadReg(baseaddr,HAW_SPI_CR_OFFSET);
57      HAW_SPI_mWriteReg(baseaddr, HAW_SPI_CR_OFFSET, ctrlReg |
            HAW_SPI_CR_SLAVE_SELCT_MASK);
58      ctrlReg = HAW_SPI_mReadReg(baseaddr,HAW_SPI_CR_OFFSET);
59      HAW_SPI_mWriteReg(baseaddr, HAW_SPI_CR_OFFSET, ctrlReg |
            HAW_SPI_CR_MANUAL_SS_MASK);
60      ctrlReg = HAW_SPI_mReadReg(baseaddr,HAW_SPI_CR_OFFSET);
61      HAW_SPI_mWriteReg(baseaddr, HAW_SPI_CR_OFFSET, ctrlReg |
            HAW_SPI_CR_TIMER_EN_MASK);
62      ctrlReg = HAW_SPI_mReadReg(XPAR_HAW_SPI_0_BASEADDR, HAW_SPI_CR_OFFSET);
63      HAW_SPI_mWriteReg(XPAR_HAW_SPI_0_BASEADDR,HAW_SPI_CR_OFFSET, ctrlReg |
            HAW_SPI_CR_POLY_ENABLE_TOGGLE_MASK);
64  }
65
66  /**
67   * Start The SPI System by clearing Master Tranaction Inhibt bit
68   * @param   baseaddr_p is the base address of the HAW_SPI device.
69   */
70  void HAW_SPI_Start(void * baseaddr_p) {
71      Xuint32 baseaddr, ctrlReg;
72      baseaddr = (Xuint32) baseaddr_p;
73      ctrlReg = HAW_SPI_mReadReg(baseaddr,HAW_SPI_CR_OFFSET);
74      HAW_SPI_mWriteReg(baseaddr,HAW_SPI_CR_OFFSET, ctrlReg & ~
            HAW_SPI_CR_TRANS_INHIBIT_MASK);
75  }
76
77  /**
78   * Stop The SPI System by setting Master Tranaction Inhibt bit
79   * @param   baseaddr_p is the base address of the HAW_SPI device.
80   */
81  void HAW_SPI_Stop(void * baseaddr_p) {
82      Xuint32 baseaddr, ctrlReg;
83      baseaddr = (Xuint32) baseaddr_p;
84      ctrlReg = HAW_SPI_mReadReg(baseaddr, HAW_SPI_CR_OFFSET);
85      HAW_SPI_mWriteReg(baseaddr,HAW_SPI_CR_OFFSET, ctrlReg |
            HAW_SPI_CR_TRANS_INHIBIT_MASK);
86  }
87
88  /**
89   * set the CS Signal to 0 for not SS Signal => not SS goes High
90   * @param   baseaddr_p is the base address of the HAW_SPI device.
91   */
92  void HAW_SPI_Set_SS(void * baseaddr_p) {
93      Xuint32 baseaddr, ctrlReg;
94      baseaddr = (Xuint32) baseaddr_p;
95      ctrlReg = HAW_SPI_mReadReg(baseaddr,HAW_SPI_CR_OFFSET);
96      HAW_SPI_mWriteReg(baseaddr,HAW_SPI_CR_OFFSET, ctrlReg & ~
            HAW_SPI_CR_SLAVE_SELCT_MASK);
97  }
98
99  /**
100  * Clears the CS Signal to 1 for not SS Signal => not SS goes Low
101  * @param   baseaddr_p is the base address of the HAW_SPI device.
102  */
103 void HAW_SPI_Clear_SS(void * baseaddr_p) {
```

```
104     Xuint32 baseaddr, ctrlReg;
105     baseaddr = (Xuint32) baseaddr_p;
106     ctrlReg = HAW_SPI_mReadReg(baseaddr, HAW_SPI_CR_OFFSET);
107     HAW_SPI_mWriteReg(baseaddr,HAW_SPI_CR_OFFSET, ctrlReg |
            HAW_SPI_CR_SLAVE_SELCT_MASK);
108 }
109
110 /*
111  * Lesen des 4 IR-Messdaten aus dem SW-Register mi t2 BLB-Zugriffe,
112  * */
113 void HAW_SPI_Read_4in2_IRs(void) {
114     Xuint32 IR_01 =
115             HAW_SPI_mReadReg(XPAR_HAW_SPI_0_BASEADDR,HAW_SPI_RCV1_0_OFFSET)
                ;
116     Xuint32 IR_23 =
117             HAW_SPI_mReadReg(XPAR_HAW_SPI_0_BASEADDR,HAW_SPI_RCV3_2_OFFSET)
                ;
118     reg1 = IR_01;           // LSB-Teil von IR_01 = IR1
119     reg2 = (IR_01 >> 16); // MSB-Teil von IR_01 = IR2
120     reg3 = IR_23;           // LSB-Teil von IR_23 = IR3
121     reg4 = (IR_23 >> 16); // MSB-Teil von IR_23 = IR4
122
123 }
124 void HAW_SPI_Toggle_PTE_0(void) {
125     Xuint32 ctrlReg;
126     ctrlReg = HAW_SPI_mReadReg(XPAR_HAW_SPI_0_BASEADDR, HAW_SPI_CR_OFFSET);
127     HAW_SPI_mWriteReg(XPAR_HAW_SPI_0_BASEADDR,HAW_SPI_CR_OFFSET, ctrlReg |
            HAW_SPI_CR_POLY_ENABLE_TOGGLE_MASK);
128 }
129 void HAW_SPI_Toggle_PTE_1(void) {
130     Xuint32 ctrlReg;
131     ctrlReg = HAW_SPI_mReadReg(XPAR_HAW_SPI_0_BASEADDR, HAW_SPI_CR_OFFSET);
132     HAW_SPI_mWriteReg(XPAR_HAW_SPI_0_BASEADDR,HAW_SPI_CR_OFFSET, ctrlReg &~
            HAW_SPI_CR_POLY_ENABLE_TOGGLE_MASK);
133 }
```

Listing B.2: *C-Code des Softwaretreibers des HAW_SPI IPs*

B.3. Header Datei für das Parklückenfinder-Software-Modul

```
1 /*
      ****************************************************************************
2 ****************************************************************************
      */
3 #ifndef PARKLUECKENFINDER_H
4 #define PARKLUECKENFINDER_H
5
6 /***************************** Include Files *******************************
      */
7 #include "xbasic_types.h"
8 #include "xstatus.h"
9 #include "xil_io.h"
```

```
10 /************************ global varibals ****************************
       */
11 unsigned int aktuell_zustand;
12 unsigned int pl_gefunden;
13 unsigned int start_suche;
14 unsigned int PL_Breite;
15 unsigned int y_hr;
16 unsigned int abstand;
17
18 /**------------------------Zustände des Automten-------------------------------
       */
19 #define PARKLUEKEN_SUCHEN                0x0
20 #define HINDERNIS_LINKS_ERKENNUNG        0x1
21 #define HINDERNIS_RECHTS_ERKENNEUNG      0x2
22 #define PARKLUECKEN_ERKENNUNG            0x3
23 #define EINPARKEN_AUSFUEHRUNG            0x4
24 #define SPI_DONE                         0x1
25
26 /**------------------------Constanten
       ----------------------------------------------------*/
27 #define ABSTAND_MAX                      25    // 20 cm
28 #define PL_MIN                           600
29 #define PL_MAX                           69    //
30 #define ABS_L2FM                         10    //10cm
31 #define ABS_H2L                          2     //2cm
32 #define START                            1
33 #define DONE                             13
34 /**********************Schreiben in SW-Register
       ****************************/
35 #define WriteReg(BaseAddress, RegOffset, Data) \
36     Xil_Out32((BaseAddress) + (RegOffset), (Xuint32)(Data))
37
38 /********************SW-Register auslesen
       ***********************************/
39 #define ReadReg(BaseAddress, RegOffset) \
40     Xil_In32((BaseAddress) + (RegOffset))
41
42 /************************ Function Prototypes ***************************
       */
43 void parklueckenfinden(void * baseaddr_p,void * dis_ist_offset_p,void *
       dis_c_en_offset_p,unsigned int IR_wert);
44 void parklueken_suchen(void);
45 void hindernis_links_erkennung(Xuint32,Xuint32) ;
46 void parkluecken_erkennung(Xuint32,Xuint32 ) ;
47 void hindernis_rechts_erkenneung(void);
48 void einparken_ausfuehrung(Xuint32 ,Xuint32 );
49 #endif /** PARKLUECKENFINDER_H */
```

Listing B.3: *Header Datei für das Parklückenfinder-Software-Modul*

B.4. C-Code für das Parklückenfinder-Software-Modul

```
1 /*
      ************************************************************************
```

```
2  ******************************************************************************
       */
3
4  /*************************** Include Files ****************************
       */
5  #include "Parklueckenfinder.h"
6  #include "haw_spi.h"
7  #include "xparameters.h"
8  #include <stdio.h>
9  //#include "xgpio.h"
10 /************************** Function Definitions **************************
       */
11
12 /*-------------------------------Parkluecke suchen-------------------------------
       */
13 void parklueckenfinden(void * baseaddr_p,void * dis_ist_offset_p,void *
       dis_c_en_offset_p,unsigned int IR_wert) {
14     pl_gefunden=0;
15     if(start_suche==START){
16         if(spi_done==SPI_DONE){
17             Xuint32 baseaddr;
18             Xuint32 dis_ist_offset;
19             Xuint32 dis_c_en_offset;
20             baseaddr = (Xuint32) baseaddr_p;
21             dis_ist_offset = (Xuint32) dis_ist_offset_p;
22             dis_c_en_offset = (Xuint32) dis_c_en_offset_p;
23             abstand=(IR_wert>>9); // IR-Wert in cm
24             switch (aktuell_zustand) {
25                 case PARKLUEKEN_SUCHEN:
26                     parklueken_suchen();
27                     break;
28                 case HINDERNIS_LINKS_ERKENNUNG:
29                     hindernis_links_erkennung(baseaddr,dis_c_en_offset);
30                     break;
31                 case PARKLUECKEN_ERKENNUNG:
32                     parkluecken_erkennung(baseaddr,dis_ist_offset);
33                     break;
34                 case HINDERNIS_RECHTS_ERKENNEUNG:
35                     hindernis_rechts_erkenneung();
36                     break;
37                 case EINPARKEN_AUSFUEHRUNG:
38                     einparken_ausfuehrung(baseaddr,dis_c_en_offset);
39                     pl_gefunden=1;
40                 break;
41             }
42         }
43     }
44 }
45
46 /*
       ---------------------------------------------------------------------------
       */
47 void parklueken_suchen() {
48     PL_Breite=0;
49     y_hr=0;
50     if (abstand <= ABSTAND_MAX) {
51         aktuell_zustand = HINDERNIS_LINKS_ERKENNUNG;
```

```
52      }
53  }
54  /*
    ------------------------------------------------------------------------
    */
55  void hindernis_links_erkennung(Xuint32 base_add,Xuint32 offset) {
56      if (abstand > ABSTAND_MAX) {
57          WriteReg(base_add, offset, 1);
58          PL_Breite=0;
59          y_hr=0;
60          aktuell_zustand = PARKLUECKEN_ERKENNUNG;
61      }
62  }
63  /*
    ------------------------------------------------------------------------
    */
64  void parkluecken_erkennung(Xuint32 base_add,Xuint32 offset) {
65      PL_Breite=ReadReg(base_add, offset);
66      if (abstand <= ABSTAND_MAX && abstand > 20) {
67          aktuell_zustand = HINDERNIS_RECHTS_ERKENNEUNG;
68      }
69      if (PL_Breite >= PL_MAX) {
70          aktuell_zustand = EINPARKEN_AUSFUEHRUNG;
71      }
72  }
73  /*
    ------------------------------------------------------------------------
    */
74  void hindernis_rechts_erkenneung() {
75      y_hr = abstand;
76      if (PL_Breite >= PL_MIN) {
77          aktuell_zustand = EINPARKEN_AUSFUEHRUNG;
78      } else {
79          aktuell_zustand = HINDERNIS_LINKS_ERKENNUNG;
80      }
81  }
82  /*
    ------------------------------------------------------------------------
    */
83  void einparken_ausfuehrung(Xuint32 base_add,Xuint32 offset) {
84      PL_Breite=ReadReg(base_add, offset);
85  }
```

Listing B.4: *C-Code für das Parklückenfinder-Software-Modul*

B.5. C Main

```
1
2  #include <stdio.h>
3  #include "xparameters.h"
4  #include "xil_types.h"
5  #include "xstatus.h"
6  #include "xil_testmem.h"
7  #include "Parklueckenfinder.h"
```

```
8  #include "haw_spi.h"
9  #include "platform.h"
10 #include "memory_config.h"
11 #include "xgpio.h"
12 #include "xutil.h"
13 #include "mb_interface.h"
14 #include <math.h>
15 #include <xintc_l.h>
16
17 int main() {
18     XGpio push;
19     Xuint32 psb_check = 00;
20     aktuell_zustand = PARKLUEKEN_SUCHEN;
21     spi_done=0;
22     XGpio LEDs8_Bit;
23     init_platform();
24
25     HAW_SPI_EnableInterrupt((void*) XPAR_HAW_SPI_0_BASEADDR);
26
27     XIntc_RegisterHandler(XPAR_XPS_INTC_0_BASEADDR,
28             XPAR_XPS_INTC_0_HAW_SPI_0_IP2INTC_IRPT_INTR,// Interrupt ID
29             haw_spi_intr_handler,// ISR NAME
30             (void*) XPAR_HAW_SPI_0_BASEADDR);
31
32     // Start the interrupt controller
33     XIntc_MasterEnable(XPAR_XPS_INTC_0_BASEADDR);
34     // Enable timer interrupts in the interrupt controller
35     XIntc_EnableIntr(XPAR_XPS_INTC_0_BASEADDR,
           XPAR_HAW_SPI_0_IP2INTC_IRPT_MASK);
36     microblaze_enable_interrupts();
37     HAW_SPI_Initialize((void *) XPAR_HAW_SPI_0_BASEADDR);
38
39     xil_printf("--␣Start␣of␣the␣Program␣--\n\r");
40     XGpio_Initialize(&push, XPAR_PUSH_BUTTONS_3BIT_DEVICE_ID);
41     XGpio_SetDataDirection(&push, 1, 0xffffffff);
42     // initialize and set data direction for LEDs_8Bit device
43     XGpio_Initialize(&LEDs8_Bit, XPAR_LEDS_8BIT_DEVICE_ID);
44     XGpio_SetDataDirection(&LEDs8_Bit, 1, 0x0);
45
46
47     while (1) {
48         psb_check = XGpio_DiscreteRead(&push, 1);
49         if (psb_check == 4) {
50             xil_printf("Fix_16_9␣=␣%d\t",reg1);
51             xil_printf("Abstand:␣␣in␣mm␣=␣%d\t", reg1 / 50);
52             xil_printf("␣in␣cm␣=␣%d\t", reg1 / 500);
53             xil_printf("Fix_16_9␣>>9=␣%d\n\r␣␣", reg1 >> 9);
54         }
55
56         /*************************PARKLÜCKENSUCHE STARTEN
               ****************************
57       Note: Vor dem Suchestart sollten folgenden Paramter mit V_Reger
             vergleichen
58       und anpassen
59       param1 : Basisadresse des V_regler_Moduls
60       param3 : Offset für das Register, in dem den berechnetem Weg
             gespeicht wird
```

```
61          param2 : Offset für das Register, in Wegberechung gestart wird
62          param  : Der Wert des IR-Sensors, mit dem die Abstasnmessung erfolgt
63                   Reg1-4 werden in HAW_SPI_ISR aus den SW-Register gelesen
                         wird
64      */
65      //start_suche=1;
66      //parklueckenfinden((void *)0,(void *) 0,(void *) 0,reg1);
67      }
68      cleanup_platform();
69      return 0;
70
71  }
```

Listing B.5: *main*

C. Systemassemblyview,mhs.File und mss.File

C.1. Systemassemblyview

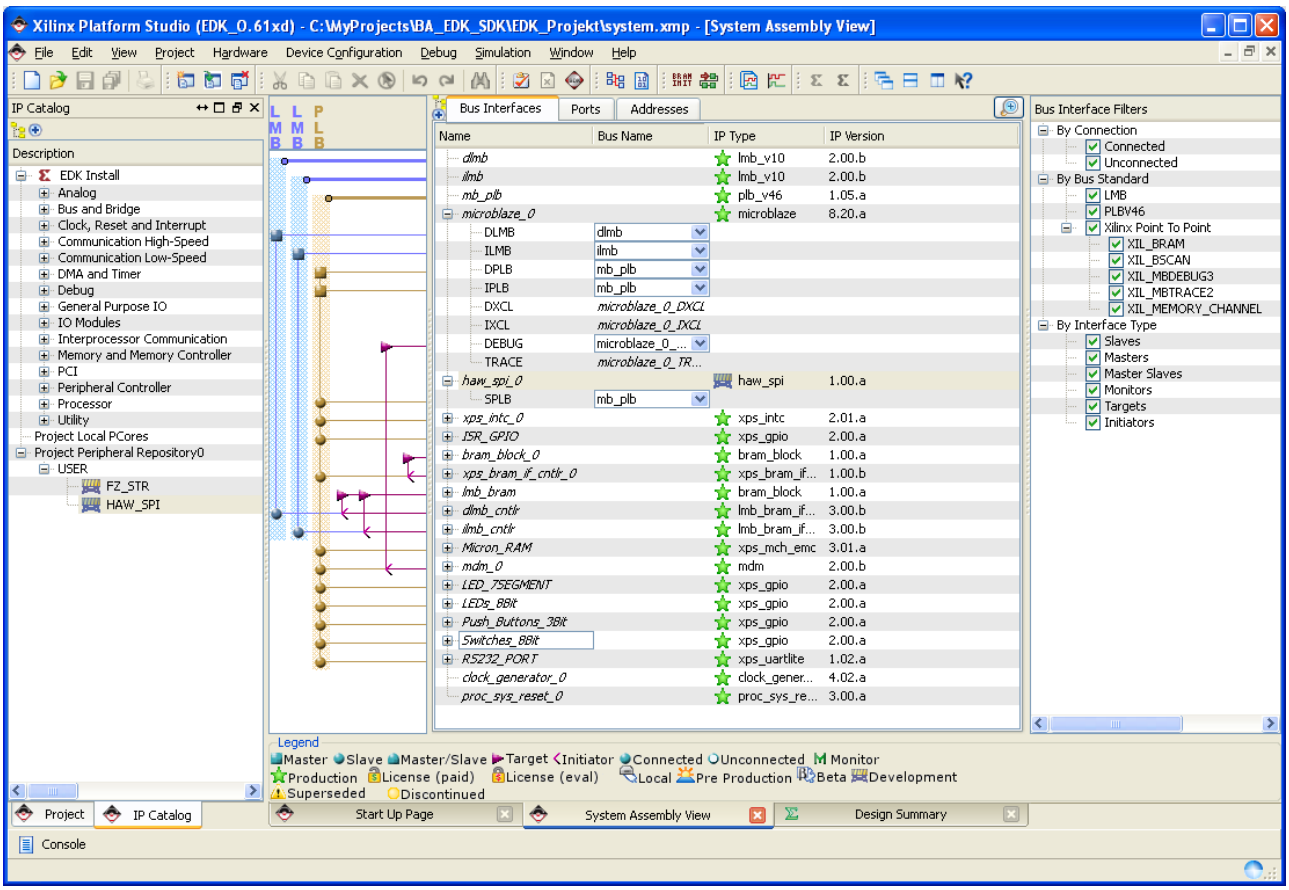

Abb. C.1.: *XPS System Übersicht*

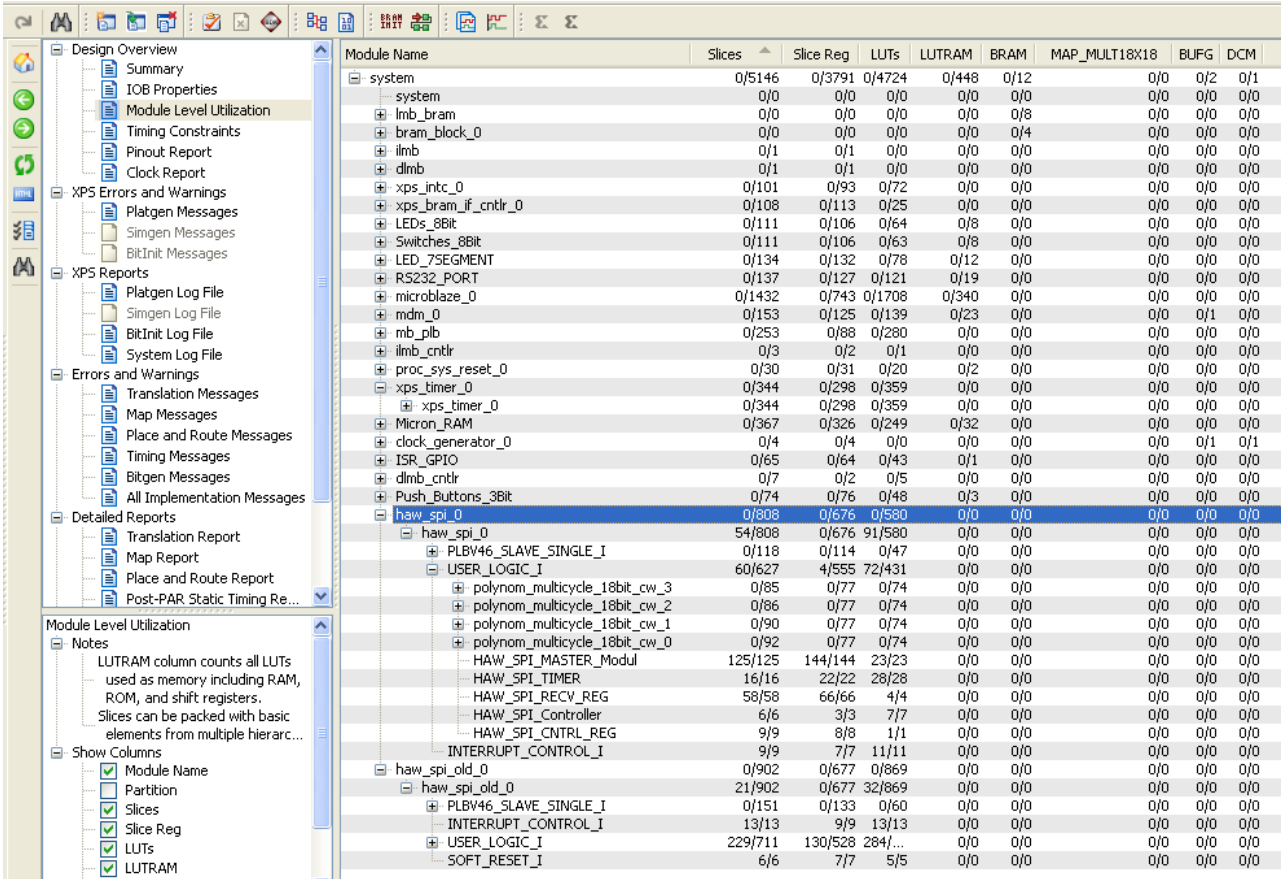

Abb. C.2.: *Übersicht des Ressourcenbedarfes der Komponenten über XPS Model Level Utilization*

C.2. Microprozessor Hardware Spezifikation(MHS).File

```
1
2  # ##############################################################################
3  # Created by Base System Builder Wizard for Xilinx EDK 13.2 Build EDK_O.61
       xd
4  # Tue Jul 12 11:24:49 2011
5  # Target Board:  Digilent Nexys 2-1200 Board Rev C
6  # Family:    spartan3e
7  # Device:    XC3S1200E
8  # Package:   FG320
9  # Speed Grade:  -4
10 # Processor number: 1
11 # Processor 1: microblaze_0
12 # System clock frequency: 50.0
13 # Debug Interface: On-Chip HW Debug Module
14 # ##############################################################################
15 PARAMETER VERSION = 2.1.0
16
17
18 PORT fpga_0_LEDs_8Bit_GPIO_IO_O_pin = fpga_0_LEDs_8Bit_GPIO_IO_O_pin, DIR
       = O, VEC = [0:7]
19 PORT fpga_0_LED_7SEGMENT_GPIO_IO_O_pin = fpga_0_LED_7SEGMENT_GPIO_IO_O_pin
       , DIR = O, VEC = [0:11]
20 PORT fpga_0_Push_Buttons_3Bit_GPIO_IO_I_pin =
       fpga_0_Push_Buttons_3Bit_GPIO_IO_I_pin, DIR = I, VEC = [0:2]
21 PORT fpga_0_Switches_8Bit_GPIO_IO_I_pin =
       fpga_0_Switches_8Bit_GPIO_IO_I_pin, DIR = I, VEC = [0:7]
22 PORT fpga_0_RS232_PORT_RX_pin = fpga_0_RS232_PORT_RX_pin, DIR = I
23 PORT fpga_0_RS232_PORT_TX_pin = fpga_0_RS232_PORT_TX_pin, DIR = O
24 PORT fpga_0_Micron_RAM_Mem_A_pin =
       fpga_0_Micron_RAM_Mem_A_pin_vslice_9_31_concat, DIR = O, VEC = [9:31]
25 PORT fpga_0_Micron_RAM_Mem_OEN_pin = fpga_0_Micron_RAM_Mem_OEN_pin, DIR =
       O
26 PORT fpga_0_Micron_RAM_Mem_WEN_pin = fpga_0_Micron_RAM_Mem_WEN_pin, DIR =
       O
27 PORT fpga_0_Micron_RAM_Mem_BEN_pin = fpga_0_Micron_RAM_Mem_BEN_pin, DIR =
       O, VEC = [0:1]
28 PORT fpga_0_Micron_RAM_Mem_DQ_pin = fpga_0_Micron_RAM_Mem_DQ_pin, DIR = IO
       , VEC = [0:15]
29 PORT fpga_0_clk_1_sys_clk_pin = CLK_S, DIR = I, SIGIS = CLK, CLK_FREQ =
       50000000
30 PORT fpga_0_rst_1_sys_rst_pin = sys_rst_s, DIR = I, SIGIS = RST,
       RST_POLARITY = 1
31 # ------------------------------------------User
       ------------------------------------------
32 PORT fz_str_0_IP2INTC_Irpt_pin = fz_str_0_IP2INTC_Irpt, DIR = O, SIGIS =
       INTERRUPT, SENSITIVITY = LEVEL_HIGH
33 PORT haw_spi_0_IP2INTC_Irpt_pin = haw_spi_0_IP2INTC_Irpt, DIR = O, SIGIS =
        INTERRUPT, SENSITIVITY = LEVEL_HIGH
34
35
```

```
36 BEGIN microblaze
37  PARAMETER INSTANCE = microblaze_0
38  PARAMETER C_AREA_OPTIMIZED = 1
39  PARAMETER C_USE_BARREL = 1
40  PARAMETER C_DEBUG_ENABLED = 1
41  PARAMETER HW_VER = 8.20.a
42  BUS_INTERFACE DLMB = dlmb
43  BUS_INTERFACE ILMB = ilmb
44  BUS_INTERFACE DPLB = mb_plb
45  BUS_INTERFACE IPLB = mb_plb
46  BUS_INTERFACE DEBUG = microblaze_0_mdm_bus
47  PORT MB_RESET = mb_reset
48  PORT INTERRUPT = xps_intc_0_Irq
49 END
50
51 BEGIN plb_v46
52  PARAMETER INSTANCE = mb_plb
53  PARAMETER HW_VER = 1.05.a
54  PORT PLB_Clk = clk_50_0000MHz
55  PORT SYS_Rst = sys_bus_reset
56 END
57
58 BEGIN lmb_v10
59  PARAMETER INSTANCE = ilmb
60  PARAMETER HW_VER = 2.00.b
61  PORT LMB_Clk = clk_50_0000MHz
62  PORT SYS_Rst = sys_bus_reset
63 END
64
65 BEGIN lmb_v10
66  PARAMETER INSTANCE = dlmb
67  PARAMETER HW_VER = 2.00.b
68  PORT LMB_Clk = clk_50_0000MHz
69  PORT SYS_Rst = sys_bus_reset
70 END
71
72 BEGIN lmb_bram_if_cntlr
73  PARAMETER INSTANCE = dlmb_cntlr
74  PARAMETER HW_VER = 3.00.b
75  PARAMETER C_BASEADDR = 0x00000000
76  PARAMETER C_HIGHADDR = 0x00003fff
77  BUS_INTERFACE SLMB = dlmb
78  BUS_INTERFACE BRAM_PORT = dlmb_port
79 END
80
81 BEGIN lmb_bram_if_cntlr
82  PARAMETER INSTANCE = ilmb_cntlr
83  PARAMETER HW_VER = 3.00.b
84  PARAMETER C_BASEADDR = 0x00000000
85  PARAMETER C_HIGHADDR = 0x00003fff
86  BUS_INTERFACE SLMB = ilmb
87  BUS_INTERFACE BRAM_PORT = ilmb_port
88 END
```

Listing C.1: *Microprozessor Hardware Spezifikation(MHS).File*

C.3. Microprozessor Software Specification(MSS).File

```
1
2   PARAMETER VERSION = 2.2.0
3
4   BEGIN OS
5    PARAMETER OS_NAME = standalone
6    PARAMETER OS_VER = 3.01.a
7    PARAMETER PROC_INSTANCE = microblaze_0
8    PARAMETER STDIN = RS232_PORT
9    PARAMETER STDOUT = RS232_PORT
10  END
11
12  BEGIN PROCESSOR
13   PARAMETER DRIVER_NAME = cpu
14   PARAMETER DRIVER_VER = 1.13.a
15   PARAMETER HW_INSTANCE = microblaze_0
16  END
17
18  BEGIN DRIVER
19   PARAMETER DRIVER_NAME = gpio
20   PARAMETER DRIVER_VER = 3.00.a
21   PARAMETER HW_INSTANCE = LED_7SEGMENT
22  END
23
24  BEGIN DRIVER
25   PARAMETER DRIVER_NAME = gpio
26   PARAMETER DRIVER_VER = 3.00.a
27   PARAMETER HW_INSTANCE = LEDs_8Bit
28  END
29
30  BEGIN DRIVER
31   PARAMETER DRIVER_NAME = emc
32   PARAMETER DRIVER_VER = 3.01.a
33   PARAMETER HW_INSTANCE = Micron_RAM
34  END
35
36  BEGIN DRIVER
37   PARAMETER DRIVER_NAME = gpio
38   PARAMETER DRIVER_VER = 3.00.a
39   PARAMETER HW_INSTANCE = Push_Buttons_3Bit
40  END
41
42  BEGIN DRIVER
43   PARAMETER DRIVER_NAME = uartlite
44   PARAMETER DRIVER_VER = 2.00.a
45   PARAMETER HW_INSTANCE = RS232_PORT
46  END
47
48  BEGIN DRIVER
49   PARAMETER DRIVER_NAME = gpio
50   PARAMETER DRIVER_VER = 3.00.a
51   PARAMETER HW_INSTANCE = Switches_8Bit
52  END
53
54  BEGIN DRIVER
```

```
55  PARAMETER DRIVER_NAME = bram
56  PARAMETER DRIVER_VER = 3.00.a
57  PARAMETER HW_INSTANCE = dlmb_cntlr
58 END
59
60 BEGIN DRIVER
61  PARAMETER DRIVER_NAME = bram
62  PARAMETER DRIVER_VER = 3.00.a
63  PARAMETER HW_INSTANCE = ilmb_cntlr
64 END
65
66 BEGIN DRIVER
67  PARAMETER DRIVER_NAME = uartlite
68  PARAMETER DRIVER_VER = 2.00.a
69  PARAMETER HW_INSTANCE = mdm_0
70 END
71
72 BEGIN DRIVER
73  PARAMETER DRIVER_NAME = bram
74  PARAMETER DRIVER_VER = 3.00.a
75  PARAMETER HW_INSTANCE = xps_bram_if_cntlr_0
76 END
77
78 BEGIN DRIVER
79  PARAMETER DRIVER_NAME = intc
80  PARAMETER DRIVER_VER = 2.02.a
81  PARAMETER HW_INSTANCE = xps_intc_0
82 END
83
84 BEGIN DRIVER
85  PARAMETER DRIVER_NAME = generic
86  PARAMETER DRIVER_VER = 1.00.a
87  PARAMETER HW_INSTANCE = haw_spi_0
88 END
```

Listing C.2: *Microprozessor Software Specification(MSS).File*

D. CD: Die folgenden Daten sind auf dem Datenträger

D.1. Xillinx EDK_Porjekt(EDK 13.2)

HAW-SPI IP-Core

D.2. Xillinx SDK_Projekten

Software für HAW-SPI

D.3. MyProzessorIPLib

Boards: Daten für die Nexys2-Boards

drivers: Software Treiber der IPs und MPD-File des Projekts

pcores: IP-Daten

- **data:** ip.bbd,ip.mpd,ip.pao
- **doc** (IP Dokumentation)
- **hdl:** (Alle VHDL Dateien des IPs)
- **netlist** mit System Generator erzeugten Dateien des IPs

D.4. Latex,Microsoft Visio und Matlab Unterlagen und Datenblätter zur Arbeit

Latex-Projekt: Die BA-Arbeit als Latex-Projekt und PDF

Visio-Unterlagen: Alle Abbildungen der Arbeit in Microsoft Visio-Format

Datenblätter: Die Datenblätter,der im Rahmen dieser Arbeit verwendeten Technologien

Versicherung über Selbstständigkeit

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

Hamburg, 25. November 2011

Ort, Datum

Unterschrift