Hochschule
Augsburg University of
Applied Sciences

Fakultät für
Informatik

Bachelorarbeit

Studienrichtung
Technische Informatik

Konfiguration und Optimierung des Embedded-Linux-Betriebssystem für Automotive Image Processing Unit

Betreuer: Mladen Kovacev

in Kooperation mit der Firma: EDAG Engineering GmbH

Prüfer: Hubert Högl

Verfasser:
Hugues landry Nseupi Nono
Salomon-Idler-Str 25
86159 Augsburg
+49 157 79552970
landrynono60@yahoo.de
Matrikelnr.: 2022666

Hochschule für angewandte Wissenschaften Augsburg
An der Hochschule 1
86161 Augsburg
Telefon: +49 (0)821-5586-0
Fax: +49 (0)821-5586-3222
info@hs-augsburg.de

Zusammenfassung

Abstract auf Deutsch. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Abstract

Abstract in English. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACL	Agent Communication Language
AMS	Agent Management System
Cougaar	Cognitive Agent Architecture
CRUD	Create, read, update and delete
CT	Container Table
DARPA	Defense Advanced Research Projects Agency
FIPA	Foundation for Intelligent Physical Agents
FPGA	Field Programmable Gate Array
GADT	Global Agent Descriptor Table
JADE	Java Agent Development Framework
JVM	Java Virtual Machine
KISS	Keep It Small & Simple
LADT	Local Agent Descriptor Table
MAS	Multi Agent System
P2P	Peer-to-Peer
PDF	Portable Document Format
SMAS	Scala Multi Agent System
VM	Virtual Machine
XML	Extensible Markup Language

Abbildungsverzeichnis

Verzeichnis der Listings

1 Einleitung

1.1 Motivation

Aufgrund der Einsätze von immer mehr Geräten, deren Funktionen uns das Leben erreichten. Egal, ob die Waschmaschine zu Hause, der Drucker in unseren Büros, oder die Kaffeemaschine in der Kantine, werden in alle diese Geräte kleine Computer gebaut, damit sie ihre Aufgabe bequem erledigen. Aber durch die gestiegene Rechenleistung und die erweiterten Kapazitäten von Mikroprozessors werden die Aufgaben von solche kleine Computer immer komplexer. Es besteht dann die Möglichkeit, ein vollwertiges Betriebssystem im diesen einzusetzen. Hier hat sich Linux durch die vielseitige Anwendbarkeit und das offene Ökosystem für Embedded Devices besonders bewährt. Am EDAG Engineering GmbH, wurde im Rahme des internen Projekts, ein Imge Processing Unit (IPU) Hardware Plattform auf Basis des Kria KV260 FPGA(Field Programmable Gate Array) entwickelt. Auf dieser Plattform wird dann aufgrund der Komplexität des Projekts ein Linux Betriebssystem eingesetzt, mit dem die 8 wesentlichen Anwendungen des Projekts konfiguriert, kompiliert, und zum User zur Verfügung gestellt wird.

1.2 Ziel der Arbeit

Angesichts der weltweiten Krise auf dem Halbleitermarkt in den letzten Monaten, wurde es immer schwieriger, hochwertige Komponenten, wie die für das Projekt verwendeten Kria KV260 Board zu finden. Statt auf der einzigen Platine des Unternehmens, musste ich meine Arbeit auf einer alternativen Platine durchführen. Also meine Arbeit in den letzten Monaten bei EDAG Engineering GmbH wurde in zwei Aufgaben aufgeteilt. Das erste Ziel dieser Arbeit war es, ein in der Firma entwickelte CAN FD Controller (mcp251xfd), der über SPI mit einem Zynq UltraScale + MPSoC ZCU106 Board verbunden ist, in Betrieb zu nehmen, damit verschiedenen Can Node vom Linux angesprochen wird.

Im Anschluss musste ich 3 von den in der Firma entwickelte Applikationen, im Linux bauen, damit das System automatisch mit den Anwendungen bootet. Dafür

müsste ich Rezepte schreiben, die sich darum kümmern werden, die Applikationen zu konfigurieren, zu kompilieren und zu installieren.

1.3 Überblick über den Aufbau der Arbeit

Diese Arbeit lässt sich in 4 Hauptkapitel aufteilen:

- **Die Einleitung**: In der Einleitung werden, die Motivation, das Ziel der Arbeit und ein gesamter Überblick auf dem Ablauf der Arbeit behandeln.
- **In den technischen Grundlagen** wird zuerst erklärt, wie das System (CAN Controller und die Zynq Mp Plattform) gebaut und funktionieren soll. Des Weiteren werden, der CAN Bus System und die SPI Interface erklärt. Dann wird dem Grundprinzip von Embedded Linux Systemen und deren Komponenten erläutert. Zum Schluss erfolgt, die Beschreibung der Petalinux Tools Flow, welches der Build System, der verwendet wird, um Linux Distribution für Xilinx Bausteinen zu kompilieren.
- **Im Versuch Aufbau** wird das Projekt, in dem ich gearbeitet habe dargestellt, dann folgt eine tiefe beschreibung der Hardware. Anschließen wird detaliert auf verschiedenen Schritte für das Bauen des System eingegangen.
- **Im Kapitel Fazit und Ausblick** werden aufgetretene Probleme und Herausforderungen erläutert, es wird analysiert, wie weit das Ergebnis von dem Ziel entfernt ist. Und anschließend wird ein Ausblick auf die möglichen Verbesserungen gegeben.

2 Technische Grundlagen

In einem ersten Schritt wird es darum gehen, die Eigenschaften eines solchen Systems zu beschreiben, das aus einem MCP251XFD CAN Controller und einem ZynqMP besteht. Das dient dazu, die Anforderungen an die Hardware und die Konfiguration des Systems verständlicher zu machen. Und dann werden die CAN Bus Systeme und die SPI Schnittstelle tiefer vorgestellt. Danach folgt eine Beschreibung von allgemeine Embedded Linux System. Im letzten Abschnitt wird das verwendete Build System präsentiert.

2.1 Technische Ausgangssituation

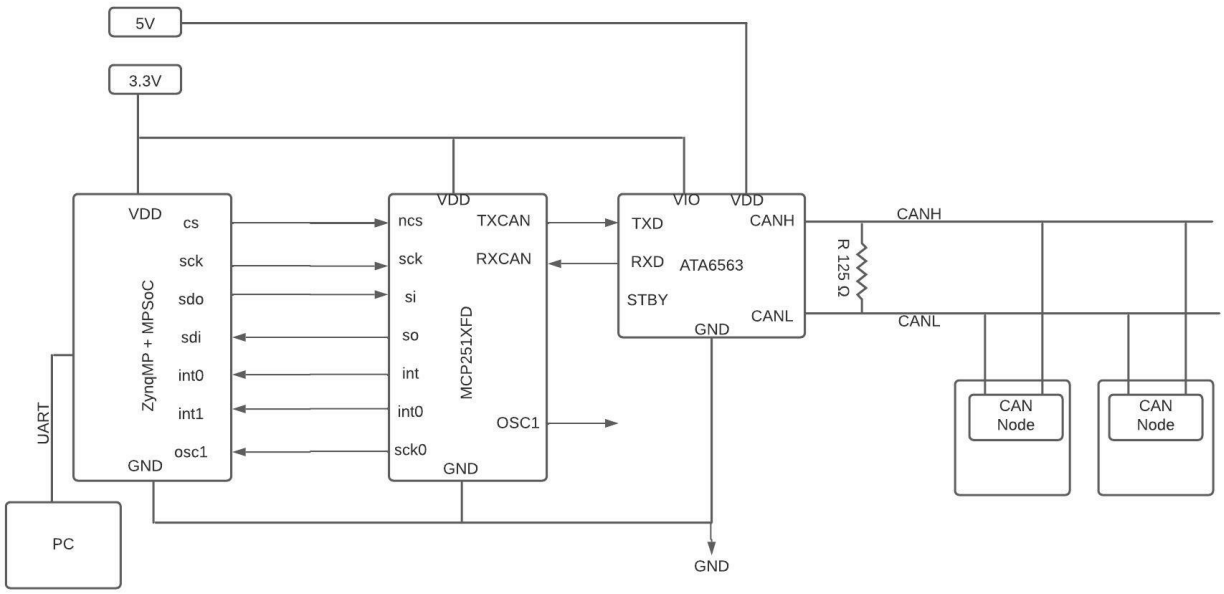

Abbildung 2.1: CAN System Diagram

Das Bild auf der Abbildung 2.1 stellt die Verschaltung zwischen der MCP251xFD CAN Controller und der ZynqMP Plattform dar. Aufgrund der hohen Lizenzgebühren für den internen Ultrascale CAN-Bus-Controller wird ein externer CAN-Controller mit Serial Peripheral Interface (SPI) als Ersatz verwendet, um die Gesamtkosten des Systems zu reduzieren. Das SPI Interface wird also verwenden, um

sowohl der CAN-Controller zu konfigurieren, als auch die CAN-Bus-Daten bis zum FPGA zu übertragen. Auf dem Xilinx FPGA Board werden außer viele andere Peripheriegeräte auch ein SPI-Controller implementiert, mit dem Zweck, die Daten, die vom externen Peripheriegeräte kommen, zu kontrollieren. Zwischen dem MCP251XFD und dem physikalischen Zweidraht-CAN-Bus ist ein CAN-FD Transceiver (ATA6563 von Microchip) zu sehen, der als Schnittstelle zwischen beiden dient. Der bietet unterschiedlicher Empfangs- und Sendefähigkeiten mit einer Hochgeschwindigkeits von bis 5Mbit/s. In den folgenden Abschnitt gebe ich die Vorteile, ein CAN-Bus zu verwenden.

2.2 Can Bus Systeme

Ein typischer Bereich, in dem die Nutzung von CAN-Bussen unumgänglich ist, wäre die Automobilindustrie. Moderne Auto verfügen Heutzutage über eine Vielzahl an elektronischen Systemen, die miteinander kommunizieren müssen. Und die übliche Verkabelungen wäre mit dem Vielzahl an Steuergeräten kaum mehr möglich. Der CAN-Bus ist in der CAN-Spezifikation von Bosch (1991) als ein Multicast-Kommunikationsprotokoll definiert, das folgende Vorteile aufweist

- CAN ist ein Multi-Master-Broadcast-System. Das heißt, dass jeder Knoten auf dem Bus mit jedem anderen Knoten kommunizieren kann.
- Der CAN-Bus hat eine Datenübertragungsgeschwindigkeit von bis zu 1 Mbit/s.
- Jeder neue Knoten kann in den Bus eingefügt werden, ohne die ursprüngliche Hardware zu verändern.
- Es bietet eine Fehlerprüfung zur Vermeidung von Busfehlern.
- Das differentielle CAN-Signal bietet eine hohe Rauschunterdrückung.

Da dieses Protokoll sehr viele Vorteile mitbringt, wurde es in den letzten Jahren in der Industrie sehr viel verbreitet. In viele Mikrocontroller werde auf diesem Grund bei der Herstellung ein CAN-Bus eingebaut.

2.2.1 Can Message Frame

In der Sprache des CAN-Standards werden alle Nachrichten als Frames bezeichnet; es gibt Daten-Frames, Remote-Frames, Error-Frames und Overload-Frames. Die an den CAN-Bus gesendeten Informationen müssen definierten Frame-Formaten von

unterschiedlicher, aber begrenzter Länge entsprechen. CAN verfügt über vier verschiedene Arten von Message Frames:

- **Data Frame (Sendet Daten)**: Die Daten werden von einem Sendeknoten zu einem oder mehreren Empfangsknoten übertragen.
- **Remote Frame (Fordert Daten an)**: Jeder Knoten kann Daten von einem Quellknoten anfordern. Auf einen Remote-Frame folgt somit ein Daten-Frame, der die angeforderten Daten enthält
- **Error Frame (Meldet einen Fehlerzustand)**: Jeder Busteilnehmer, egal ob Sender oder Empfänger, kann zu jeder Zeit während einer Daten- oder Remote-Frame-Übertragung einen Fehlerzustand melden.
- **Overload-Frame (Meldet Knotenüberlastung)**: Ein Knoten kann zwischen zwei Daten- oder Remote-Frames eine Verzögerung anfordern, das heißt, dass der Overload-Frame nur zwischen Daten- oder Remote-Frame-Übertragungen auftreten kann.

Im Nachfolgenden gegen wir auf der Architektur von den jeweiligen CAN Frame Typen ein.

2.2.1.1 Data Frame

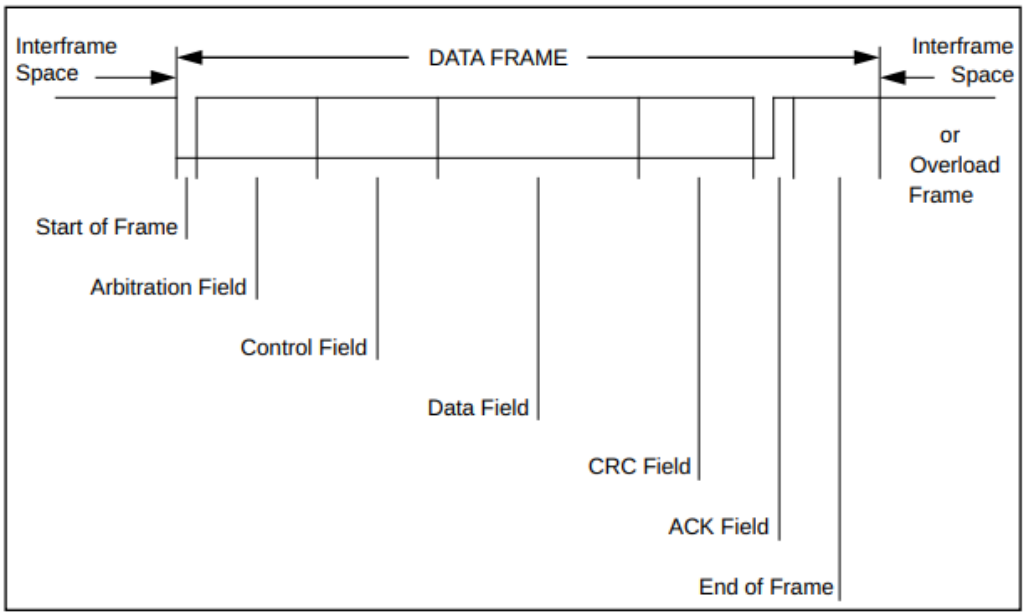

Abbildung 2.2: CAN-Data Frame Architektur Bosch (1991)[p. 12]

Die Abbildung 2.2 beschreibt die 7 Bestandteile, aus denen ein Data Frame besteht, namlich:

- **SOF**(Start of Frame): Zeigt den Beginn von Daten und Remote Frames an.
- **Arbitration Field**: der besteht auf
  - Identifikator: Die Basis-ID besteht aus 11 Bits und die erweiterte ID aus 29 Bits.
  - RTR(Remote Transmission Request)-Bit: Im Data Frame ist das RTR-Bit "0". Im RTR-Frame hingegen ist es "1".
- **Control Field**: Dient zur Bestimmung der Datengröße und der Länge der Nachrichten-ID. der besteht auf 6 Bits.
  - IDE (Identifikator-Erweiterung): Dieses Bit bestimmt den Identifikator als Basis-ID oder Erweiterte ID.
  - R0,R1: reservierte Bits.
  - DLD (Data Length Code): Er wird zur Bestimmung der Datenlänge verwendet.
- **Data Field**: bis zu 8 Byte Datenfeld.
- **CRC-Field (Cyclic Redundancy Check)**: zur Überprüfung der Datenkorrektur.
- **ACK Field (Acknowledgement Field)**: um zu bestimmen, ob die Nachricht empfangen wurde oder nicht. Bei Empfang von Daten wird dieses Bit auf High gezogen.
- **EOF (End of Frame)**: Zeigt das Ende von Daten- und Remote-Frames an.

2.2.1.2 Remote Frame

Die Abbildung 2.3 beschreibt die Bestandtele einem Remote-Frame. Data-Frame und Remote-Frame sind sich beide sehr ähnlich. Im Prinzip ist der Remote Frame ein Data Frame ohne das Datenfeld. Dieser besteht in der Regel aus den gleichen Bestandteilen wie der Data Frame.

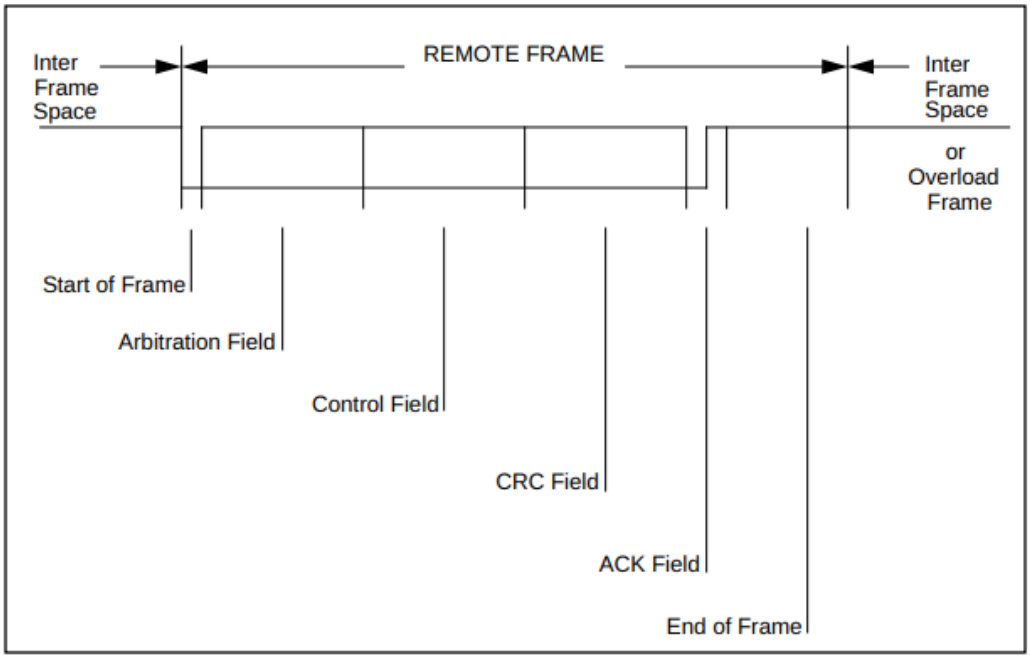

Abbildung 2.3: CAN-Remote Frame Architektur Bosch (1991)[p. 17]

2.2.1.3 Error Frame

Abbildung 2.4 zeigt die Struktur der Error Frame an. Der Error Frame besteht aus zwei Teilen:

- Error Flag: stellt ein Knoten einen Fehlerzustand fest, erzeugt er bis zu 12 Bits "0"für das Fehlerflag.
- Error Delimiter: 8 Bits "1"beenden den Error Frame.

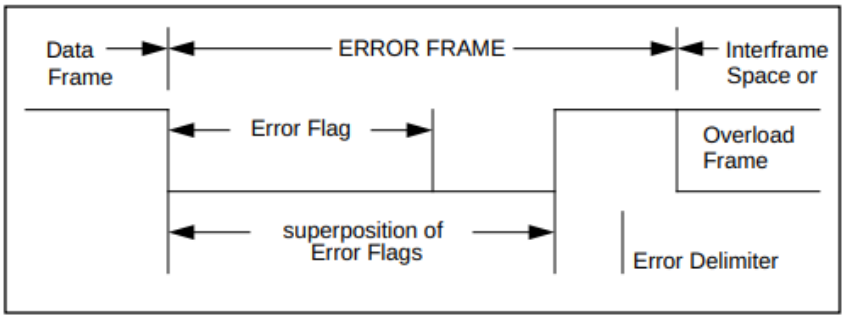

Abbildung 2.4: Can-Error-Frame Bosch (1991)[p. 18]

2.2.1.4 Overload Frame

Abbildung 2.5 ist der Überlastrahmen. Er wird von dem Empfängerknoten erzeugt, um mehr Verzögerung zwischen den Datenrahmen zu erzwingen.

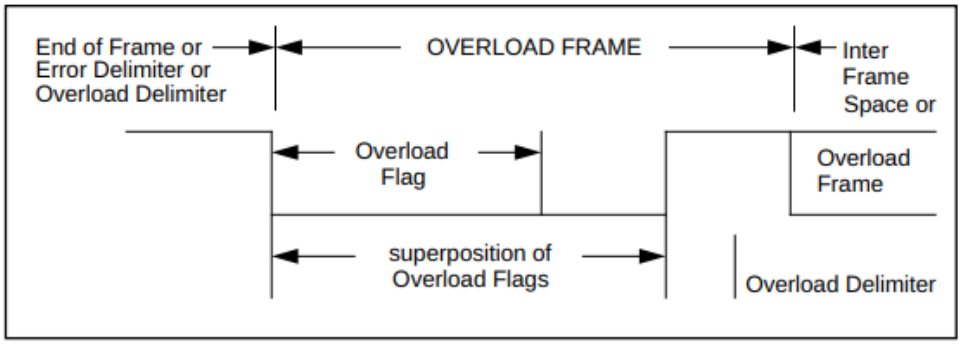

Abbildung 2.5: Can-Overload-Frame Bosch (1991)[p. 19]

2.2.2 Can Physical Layer

Der CAN FD Protokoll, der während dieser Arbeit verwendet wird, ist in ISO 1189-1:2015 definiert. Dieses Protokoll beschreibt nicht die mechanischen, Drähte, und Anschlüsse, aber fordert allerdings, dass die Drähte und Anschlüsse den elektrischen Spezifikationen entsprechen müssen. Abbildung 2.6 zeigt eine CAN-Verbindung mit

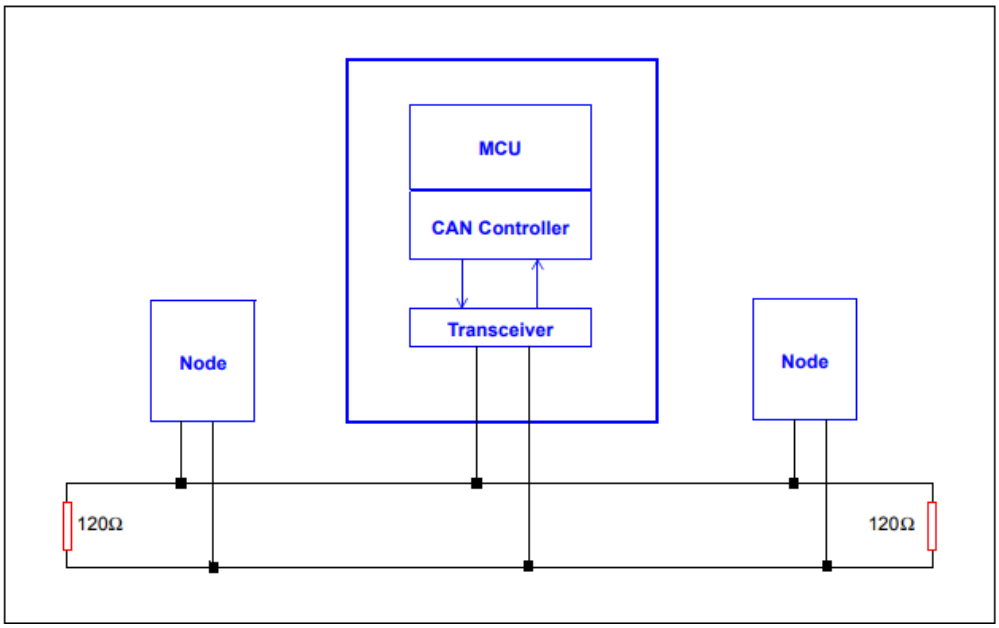

Abbildung 2.6: Can-Bus Connexion Richards (2002)[p. 2]

zwei CAN-Node gemäß der ISO-11898-1 CAN-Spezifikation. CAN High(CAN_H) und CAN Low(CAN_L) verlangen zwei 120Ω-Abschlusswiderstände. Der Transceiver wandelt die von CAN-Knoten kommenden CAN-Signale in ein digitales Rx- und Tx-Signal für den Node Controller um. Des Weiteren handelt es sich bei CAN_H und CAN_L um Differenzsignale. wie auf der Abbildung 2.7 zu sehen ist, wenn die zwei Signale bei 2,5 V liegen, ist dies ein rezessives Signal, also eine logische 0. Wenn CAN_H auf 3,5 V und CAN_L auf auf 1,5 V, dann handelt es sich um ein dominantes Signal, also eine logische 1.

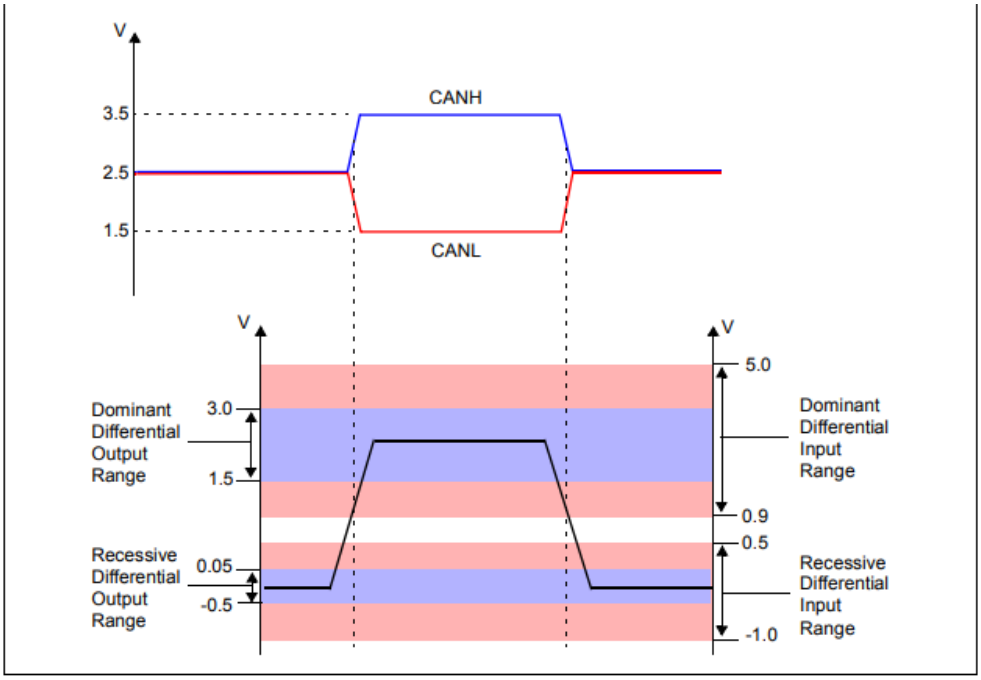

Abbildung 2.7: CAN_H and CAN_L Richards (2002)[p. 3]

2.3 SPI Interface

Die *Serial Peripheral Interface* (SPI) ist eine der am häufigsten verwendeten Schnittstellen zwischen Mikrocontrollern und Peripherie-ICs wie Sensoren, ADCs, DACs, Schieberegistern, SRAM und anderen. Die Schnittstelle SPI ist eine synchrone, auf Voll-Duplex basierte Master-Slave-Schnittstelle. Die Daten vom Master oder Slave werden mit der aufsteigenden oder abfallenden Taktflanke synchronisiert. Dabei können sowohl Master als auch Slave gleichzeitig Daten übertragen.
Das SPI arbeitet aber nach dem Single-Master-Prinzip. Das bedeutet, dass ein zentrales Gerät die gesamte Kommunikation mit den Slaves initiiert. Der Master sendet Daten auf der MOSI-Signalleitung und empfängt Daten auf der MISO-Signalleitung,

so dass der Busmaster gleichzeitig Daten senden und empfangen kann. wie auf dem Bild A der Abbildung 2.8 zu sehen ist. Alle Datenübertragungen müssen zwischen dem Bus-Master und den Slaves stattfinden. Datenübertragungen die direkt zwischen zwei Slave-Geräten stattfinden sind nicht erlaubt.

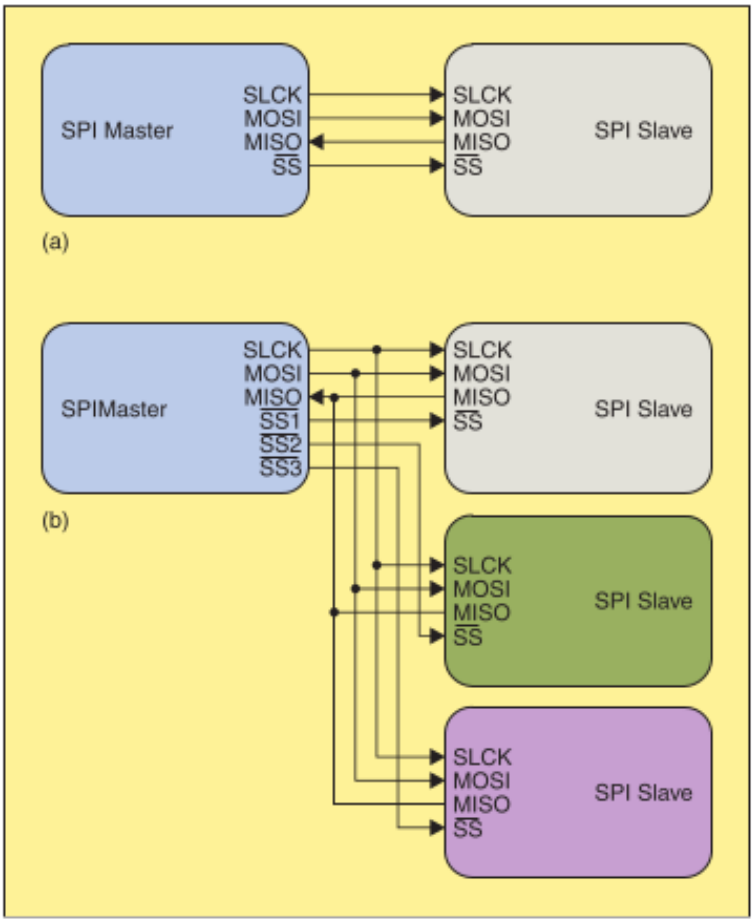

Abbildung 2.8: SPI Bus Leens (2009)[p. 9]

Möchte der SPI-Master Daten an einen Slave senden und/oder von ihm Informationen anfordern, dann wählt er einen Slave aus, und zwar durch Ziehen der entsprechenden SS-Leitung nach unten, während er das Taktsignal mit einer für den Master und den Slave nutzbaren Taktfrequenz aktiviert. SPI ist eine Protokoll mit vier Signalleitungen, wie auf Leens (2009)[p. 9] zu lesen ist.

- **Ein Clock-Signal (SCLK)**, welches vom Bus-Master an alle Slaves gesendet wird; alle SPI-Signale sind mit diesem Clock-Signal synchronisiert.
- **Der Slave Select Signal**: der zur Auswahl des Slaves dient, mit dem der Master kommuniziert.

- **Eine Datenleitung vom Master zu den Slaves**, bezeichnet als Master Out-Slave In (MOSI).
- **Eine Datenleitung von den Slaves zum Master**, bezeichnet als Master In-Slave Out (MISO).

2.4 Embedded Linux

2.5 Komponente eines Embedded Linux Systems

2.6 Petalinux Tool Flow

3 Versuch Aufbau

3.1 Allgemein über das Projekt

3.2 Hardware Platform

3.2.1 MCP251XFD CAN Controller

3.2.2 Zynq UltraScale + MPSoC ZCU106 Evaluation Board

3.3 Konfiguration und Bauen des Systems

4 Fazit und Ausblick

4.1 Fazit

4.2 Ausblick

Literaturverzeichnis

[**Bosch 1991**]

Bosch, Robert: CAN Specification Version 2.0. In: *Rober Bousch GmbH, Postfach* 300240 (1991), 72. http://esd.cs.ucr.edu/webres/can20.pdf 2.2, 2.2, 2.3, 2.4, 2.5

[**Leens 2009**]

Leens, Frédéric: An introduction to I2C and SPI protocols. In: *IEEE Instrumentation and Measurement Magazine* 12 (2009), Nr. 1, S. 8–13. http://dx.doi.org/10.1109/MIM.2009.4762946. – DOI 10.1109/MIM.2009.4762946. – ISSN 10946969 2.8, 2.3

[**Richards 2002**]

Richards, Pat: A CAN Physical Layer Discussion. In: *Technology* (2002), S. 1–12 2.6, 2.7

Ich, Hugues landry Nseupi Nono, Matrikel-Nr. 2022666, versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema

> *Konfiguration und Optimierung des Embedded-Linux-Betriebssystem für Automotive Image Processing Unit - Betreuer: Mladen Kovacev*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Augsburg, den 7. März 2022

Hugues landry Nseupi Nono

A Anhang

A.1 Inhalt des Datenträgers

Der dieser Arbeit beigelegte Datenträger beinhaltet zusätzliche Materialen. Neben der Arbeit selbst im Portable Document Format (PDF) befinden sich sowohl die Sources der Implementierungen als auch die lauffähigen Pakete.

`./all-my-packages/`
: Sources der Packages

`./Architektur/`
: UML-Diagramme der Architektur

`./Thesis_Vorname_Nachname_123456.pdf`
: PDF Version dieser Arbeit

`./ThesisVM.ova`
: Virtual Box Image mit lauffähiger Demoumgebung