HAW HAMBURG

Bachelorarbeit

Markus Kasten

Hardwareplattformen für autonome Straßenfahrzeuge im Maßstab 1:87

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Markus Kasten

Hardwareplattformen für autonome Straßenfahrzeuge im Maßstab 1:87

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

Betreuender Prüfer: Prof. Dr. Tim Tiedemann
Zweitgutachter: Prof. Dr. Stephan Pareigis

Eingereicht am: 22. März 2021

Markus Kasten

Thema der Arbeit

Hardwareplattformen für autonome Straßenfahrzeuge im Maßstab 1:87

Stichworte

Autonomes Fahren, Hardware, Raspberry Pi, Linux, ROS, Miniaturfahrzeuge, H0

Kurzzusammenfassung

Im Rahmen dieser Arbeit wird eine Hardwareplattform für ROS-basierte autonome Straßenfahrzeuge im Maßstab 1:87 entwickelt. Zunächst werden mögliche Konzepte und Komponenten für Recheneinheiten und Sensorik vorgestellt und diskutiert. Anschließend wird auf dessen Basis eine konkrete Umsetzung aufgebaut. Durch die Verwendung eines Raspberry Pi Compute Module 4 sowie zwei Machine-Learning Beschleuniger stehen ausreichend Ressourcen auf dem Fahrzeug bereit, sodass auch fortgeschrittene Algorithmen zum autonomen Fahren in einer Miniaturwelt zum Einsatz kommen können.

Markus Kasten

Title of Thesis

Hardware platforms for autonomous road vehicles at 1:87 scale

Keywords

Autonomous driving, Hardware, Raspberry Pi, Linux, ROS, Miniature vehicles, H0

Abstract

This thesis employs the idea of developing a hardware platform for a ROS-based autonomous street vehicle at 1:87 scale. First of all possible concepts and components for processing units and sensors are presented and discussed, then a concrete implementation is built upon these. By using a Raspberry Pi Compute Module 4 next to two machine learning accelerators the vehicle has the resources to allow even advanced algorithms for autonomous driving in a miniature world.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

BSP Board Support Package.

DPU Deep Learning Processing Unit.

FDM Fused Deposition Modelling.

FFC Flat Flex Cable.

I²C Inter-Integrated Circuit.

RTC Real Time Clock.

SBC Single Board Computer.

SoC System on a Chip.

SoM System on Module.

SPI Serial Peripheral Interface.

TPU Tensor Processing Unit.

UART Universal Asynchronous Receiver Transmitter.

UWB Ultra Wideband.

1 Einleitung

Im Rahmen des autonomen Fahrens gibt es ein großes Themenangebot an zu untersuchenden Aufgaben. Die praktische Umsetzung dieser bregrenzt sich in vielen Fällen aber auf Simulatoren, da Experimente nur auf einem speziellen Testgelände oder unter strengen Auflagen im Straßenverkehr durchgeführt werden können. Solche Experimente sind weiterhin mit hohen Kosten verbunden und für alle Beteiligten gefährlich. Somit ist die Erforschung nur schwer zugänglich.

Um das autonome Fahren in einer sicheren und kosteneffizienten Umgebung zu erforschen eigenet sich eine Modellbau-Umgebung in einem kleineren Maßstab. Hier fallen viele der Einschränkungen von Untersuchungen mit echten Fahrzeugen weg, so dass die Erforschung des autonomen Fahrens auch für kleinere Institutionen zugänglich ist.

Der Maßstab 1:87 (auch H0 genannt) hat sich in der Modellbau-Szene als gängiger Maßstab durchgesetzt. So wurde im Miniatur-Wunderland in Hamburg eine 1499 m^2 große Umgebung mit einem ausgeprägten Straßennetz geschaffen. Eine Vielzahl an Modellbauartikeln ist im Maßstab 1:87 erhältlich, was den Aufbau eigener Modellbau-Welten vereinfacht.

Der Maßstab ist klein genug, um komplexe Welten auf kleinem Raum unterzubringen .In den folgenden Kapiteln soll gezeigt werden, dass es es trotz dieser Umstände möglich ist ein leistungsfähiges autonomes Fahrzeug zu bauen.

1.1 Zielsetzung

Im Rahmen dieser Arbeit soll eine Hardwareplattform entwickelt werden, mit der das autonome Fahren in einer 1:87-Umgebung erforscht werden kann. Zu dieser Plattform gehört die Auswahl einer zentralen Recheneinheit, Sensorik und Aktorik, sowie ein fahrendes Chassis. Softwarekomponenten werden in dieser Arbeit nicht tiefer betrachtet sondern sind Teil der Arbeit von Luk Schwalb[11].

Die Plattform soll das Robot Operating System (ROS) unterstützen. Da für ROS ein Linux-Unterbau benötigt wird, fallen minimale Ansätze mit Mikrocontrollern wie ESP32 oder STM32 als Option weg. Desweiteren sollte die Plattform die nötigen Ressourcen bieten, um auch komplexere Verfahren lokal auf dem Fahrzeug ausführen zu können. Dazu zählen auch Machine-Learning Methoden, die einen speziellen Beschleuniger für einen hohen Durchsatz benötigen.

Weiterhin ist eine Erweiterungsfähigkeit nötig, um in Zukunft weitere Sensoren oder Koprozessoren in das Fahrzeug einbinden zu können.

1.2 Aufbau der Arbeit

Zunächst werden in Kapitel 2 einige für die Arbeit nötige Grundlagen erläutert.

In Kapitel 3 werden verschiedene mögliche Konzepte für autonome Straßenfahrzeuge im Maßstab 1:87 vorgestellt. Dies umfasst verschiedene Hardwareoptionen für die zentrale Recheneinheit, die Sensorik als auch Aktorik.

Das Kapitel 4 fasst die im vorherigen Kapitel vorgestellten Konzepte und Ansätze auf stellt zwei im Rahmen dieser Arbeit erstellte Umsetzungen dieser vor.

In Kapitel 5 wird eine der vorgestellten Umsetzung näher untersucht.

Abschließend wird in Kapitel 6 ein Fazit gezogen und ein Ausblick auf zukünftige Arbeiten gegeben.

2 Grundlagen

Zunächst werden die Grundlagen für die Entwicklung von Hardwareplattformen für autonome Straßenfahrzeuge im Maßstab 1:87 näher gebracht.

2.1 Forschungsstand

Die Arbeit mit Fahrzeugen im Maßstaben 1:87 und die Forschung an solchen ist nicht vollständig neu.

Nils Schönherr baut 2019 im Rahmen der Arbeit "Kamera-basierte Minimalautonomie" ein minimalistisches Fahrzeug "uAC" im selben Maßstab auf[12]. Das Fahrzeug basiert auf einem Espressif ESP32 Mikrocontroller und zielt auf möglichst leichtgewichtige Algorithmen ab, welche mit den eingeschränkten Resourcen arbeiten können. Als Testumgebung dient hier eine möglichst stark vereinfachte Strecke, welche sich auf einen schwarzen Untergrund mit weißen Markierungen beschränkt. Während die Testfahrten in der Umgebung erfolgreich sind, wären die verfügbaren Ressourcen nicht ausreichend um komplexere Algorithmen für den Einsatz in weniger vereinfachten Umgebungen auszuführen.

Eine ähnliche Plattform wird im Rahmen einer Projektarbeit von dem Author dieser Arbeit aufgebaut. Ein ebenfalls im Maßstab 1:87 aufgebauter, auf einem ESP32 basierender Reisebus "ESP-Bus" wird verwendet um im Miniatur-Wunderland Bild- und Lenkdaten aufzunehmen. Die dort gesammelten Daten wurden für das Training von End-to-End Netzen verwendet.

Sebastian Paulsen beschäftigt sich im Rahmen seiner Bachelorarbeit "Entwicklung und Evaluierung kompakter Hardware zur Realisierung künstlicher neuronaler Netze im Bereich autonomer Fahrzeuge" ebenfalls mit der Entwicklung eines 1:87-Fahrzeugs. Dieses soll auf einem leistungsstarken Xilinx Zynq basieren um auch mit neuronalen Netzen

arbeiten zu können[9]. Die dort entworfene Hardware konnte im Rahmen der Arbeit allerdings nicht zur Funktion gebracht werden.

In dieser Arbeit wird an die genannten Arbeiten angeknüpft, es soll eine Fahrzeugplattform geschaffen werden, mit der sich autonomes Fahren in einer realistischen 1:87 Umgebung umsetzen und erforschen lassen soll.

2.2 Miniaturautonomie

Die Miniaturisierung autonomer Fahrzeuge schafft gleichzeitig Vereinfachungen als auch neue Herausfordungen. Zum einen werden die Gefahren und Kosten, die bei der Arbeit mit echten Fahrzeugen entstehen, stark vermindert. Dies bietet Studierenden und Forscher*innen einen vereinfachten Zugang zu dem Gebiet. Gleichzeitig kann nicht mehr auf beinahe beliebig große und leistungsstarke Hardware zurückgegriffen werden, die beispielsweise im Kofferraum eines Fahrzeugs unterkommen würde. Daher muss sowohl hardware- als auch softwareseitig auf die neue Größe optimiert werden.

Der Carolo-Cup ist eine von der Technischen Universität Braunschweig jährlich ausgetragene Veranstaltung, in der Teams verschiedener Universitäten mit Fahrzeugen im Maßstab 1:10 bei verschiedenen Herausforderungen gegeneinander antreten. Hier wird meistens, aber nicht immer, auf kommerziell erhältliche Chassis und Elektronik zurückgegriffen.

2.2.1 Datenverbeitung in miniaturautonomen Fahrzeugen

Bei miniaturautonomen Fahrzeugen bieten sich zwei Arten der Datenverarbeitung an.

Externe Datenverarbeitung

Bei der externen oder entfernten Datenverarbeitung werden die Sensordaten des Fahrzeugs an einen zentralen Rechner übertragen, dort verarbeitet und anschließend die Steuerbefehle zurück an das Fahrzeug übertragen. Auf eine solche Datenverarbeitung wurde beispielsweise bei dem vorher genannten “ESP-Bus” gesetzt. Da für die Übertragung der Daten nur wenig Ressourcen notwendig sind vereinfacht dies die Entwicklung der fahrzeugseitigen Elektronik stark und muss nicht im Vorfeld auf die geplanten Algorithmen

abgestimmt werden. Gleichzeitig kann der zentrale Rechner beinahe beliebig skaliert werden.

Nachteilig ist hingegen die Abhängigkeit der Datenübertragung. Bei der Übertragung entstehen Latenzen, welche eine präzise Steuerung des Fahrzeugs erschweren. Beim "ESP-Bus" wurden Latenzen von etwa 150 ms gemessen. Bei umgerechneten $50\,\mathrm{km\,h^{-1}}$ ergibt sich so eine blinde Strecke von $150\,\mathrm{ms} \cdot \frac{50\,\mathrm{km\,h^{-1}}}{87} = 2{,}4\,\mathrm{cm}$. Aussetzer bei der Übertragung, beispielsweise bei Funkstörungen durch metallische Objekte oder fremde Sender, führen gezwungenermaßen zu einem kompletten Ausfall der Autonomiefunktionen.

Lokale Datenverarbeitung

Bei der lokalen Datenverarbeitung erfolgt die vollständige Datenverarbeitung auf dem Fahrzeug selbst. Während die Hardwareentwicklung so eine größere Herausforderung darstellt, wird die Softwareentwicklung durch das Wegfallen von Datenübermittlung und damit verbundenen Latenzen und möglichen Ausfällen vereinfacht. Die verfügbaren Resourcen müssen allerdings möglichst effizient ausgenutzt werden, da eine Erweiterung in der Regel nicht trivial ist. Denkbar ist auch eine hybride Datenverarbeitung, bei der ein Großteil der Algorithmen lokal auf dem Fahrzeug ausgeführt werden, aber einzelne nicht zeitkritische Schritte, beispielsweise komplexe neuronale Netze, von einer sich nicht auf dem Fahrzeug befindlichen Recheneinheit durchgeführt werden.

Eine lokale Datenverarbeitung vereinfacht außerdem die Skalierung auf mehrere Fahrzeuge. Bei Untersuchungen mit vielen Fahrzeugen muss nicht die gleiche Anzahl an externen Rechnern bereitgestellt werden, weiterhin würde die Zuverlässigkeit der Datenübertragung mit steigender Anzahl an Teilnehmern nachlassen.

2.2.2 Mikrowunderland

Um mit Fahrzeugen im Maßstab 1:87 auch innerhalb der HAW-Hamburg arbeiten zu können wurde das Mikrowunderland geschaffen. Dabei handelt es sich um eine Modellwelt im Maßstab 1:87 mit einem Außenmaß von ungefähr $2\,\mathrm{m} \times 1{,}4\,\mathrm{m}$. Die in Abbildung 2.1 dargestellte Modellwelt besteht aus einem Straßennetz, das es erlauben soll möglichst viele Verkehrssituationen auf kleiner Fläche simulieren zu können. Dazu gehören unter anderem verschiedene Straßenbeläge, unterschiedliche Links- und Rechtskurven, Kreuzungen, Verkehrsinseln, und fehlende Markierungen. Objekte wie Häuser und Bäume

Abbildung 2.1: Vogelperspektive des Mikrowunderlandes

Quelle: Luk Schwalb

lassen sich über Magneten an beliebiger Stelle platzieren, sodass ein schnelles Umgestalten der Modellwelt möglich ist.

Ein unter der Fahrbahn verlegter Fahrdraht erlaubt passive Verkehrsteilnehmer, die über einen Magneten in der Spur gehalten werden. Dieses System kann zum Sammeln von Daten als auch für dritte Verkehrsteilnehmer genutzt werden. Unter den Kreuzungen angebrachte Weichen können diese Verkehrsteilnehmer steuern.

2.2.3 Aufzeichnung von Trainingsdaten

In der Regel sind die für das Training von neuronalen Netzen verwendeten Datensätze auf echten Straßen aufgenommen. Für eine bessere Anpassung der Netze für die Verwendung in einer Modellbauwelt ist es jedoch wünschenswert, Daten aus dem selben Maßstab zu verwenden.

Neben dem autonomen Fahren eignet sich eine Hardwareplattform für autonome Fahrzeuge auch zum Sammeln von Daten. Wird das Fahrzeug ferngesteuert oder durch einen Fahrdraht gelenkt lassen sich die Daten der Kamera und Steuerung aufzeichnen und später als Trainingsdaten für neuronale Netze verwenden. Die gesammelten Daten werden nach der Aufzeichnung sortiert und ggf. von Hand annotiert.

2.3 ROS

ROS (Robot Operating System) ist ein Framework für die Entwicklung von Robotern. Entgegen des Namens handelt es sich nicht um ein eigenes Betriebssystem, sondern stellt Bibliotheken und Infrastruktur für die Entwicklung bereit. Als Betriebssystem wird Debian/Ubuntu empfohlen.

3 Mögliche Konzepte

Im Folgenden werden verschiedene Konzepte für Fahrzeuge im Maßstab 1:87 vorgestellt und miteinander verglichen.

3.1 Zentrale Recheneinheit

Die zentrale Recheneinheit des Fahrzeugs ist das wichtigste Element in dem Entwurf einer neuen Fahrzeugplattform im Miniaturmaßstab. Da sie nicht beliebig klein ausgelegt werden kann, ist sie für den restlichen Aufbau den Fahrzeugs maßgebend. Von ihr hängt daher zunächst die Größe des Fahrzeugs, später aber auch die nutzbare Peripherie, sowie die Leistungsfähigkeit und Flexibiltät von nutzbaren Algorithmen ab.

Die Auswahl der folgenden Optionen beschränkt sich auf als Modul erhältliche Plattformen, auch System on Module (SoM) genannt. Gegenüber der direkten Entwicklung mit System on a Chips (SoCs) ist so eine deutlich einfachere und schnellere Entwicklung möglich, gleichzeitig werden die Design-Risiken und -Kosten so deutlich reduziert.

3.1.1 Raspberry Pi

Raspberry Pis sind sogenannte Single Board Computer (SBC), die von der britischen Raspberry Pi Foundation entwickelt werden. Ursprünglich wurden Raspberry Pis zum Lernen von Programmiersprachen für Kinder vorgesehen, auf Grund der Leistungsfähigkeit, des geringen Preises und dem großen Einsatzspektrum findet man sie nun aber hauptsächlich in den Gebieten der Heimautomatisierung, Robotik und Forschung wieder.

Gegenüber vielen anderen am Markt erhältlichen Einplatinencomputern bietet die Raspberry Pi Foundation offizielle Betriebssystem-Images auf Basis von Debian an. Anwender sind daher nicht auf die Verfügbarkeit von 3rd-party Images mit ungewisser Stabilität

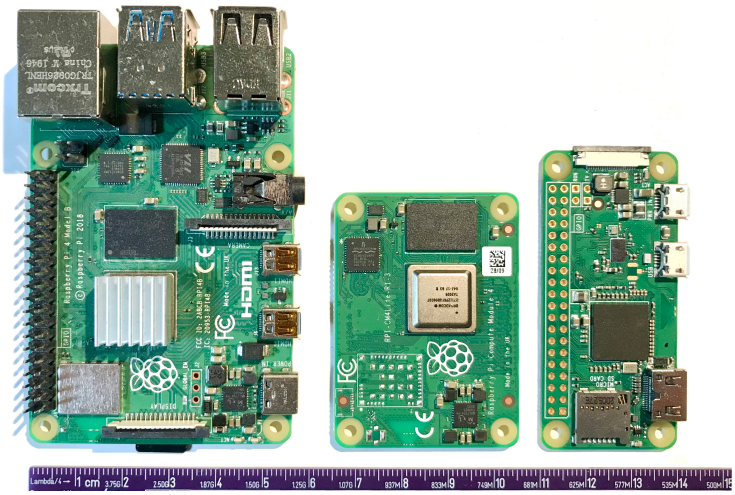


Abbildung 3.1: Größenvergleich einiger Raspberry Pi Modelle: Links Raspberry Pi 4, Mitte Raspberry Pi Compute Module 4, Rechts Raspberry Pi Zero W

Quelle: Markus Kasten

angewiesen. Durch die große Verbreitung und die entstandene Community um Raspberry Pis ist eine breite Auswahl an Dokumentation und Software verfügbar.

Seit dem ersten Raspberry Pi aus 2012 sind eine Reihe verschiedener Modelle in unterschiedlichen Formfaktoren (vgl. Abbildung 3.1) erschienen, von denen im Folgenden zwei als zentrale Recheneinheit näher betrachtet werden. Auf Grund der geringen Größe eignen sich der Raspberry Pi Zero W und das Compute Module 4 besonders für die Verwendung im Maßstab 1:87.

Zero W

Der Raspberry Pi Zero wurde im November 2015 eingeführt, und im Februar 2017 unter dem Namen Zero W um eine WLAN-Schnittstelle erweitert. Der Raspberry Pi Zero W ist 65 mm × 30 mm groß und passt somit liegend und stehend in ein 1:87-Fahrzeug.

Der verwendete SoC ist ein Broadcom BCM2835 mit einer 1-Kern ARM11 CPU @ 1 GHz, einer VideoCore IV GPU und 512 MB Arbeitsspeicher. Als Massenspeicher-Medium wird eine Micro-SD Karte verwendet.

Über die 40-polige 2,54 mm-Pinleiste kann externe Peripherie angebunden werden. Die hier verfügbaren Schnittstellen umfassen GPIOs, eine I^2C-Schnittstelle, Serial Peripheral Interface (SPI) und einen Universal Asynchronous Receiver Transmitter (UART). Weiterhin verfügt der Raspberry Pi Zero über eine CSI-Schnittstelle für die Anbindung einer Kamera.

Compute Module 4

Im Oktober 2020 wurde das auf dem Raspberry Pi 4 basierende Compute Module 4 (auch CM4) veröffentlicht. Im Gegensatz zu anderen Raspberry Pi Modellen handelt es sich bei Compute-Modules nicht um einen Einplatinencomputer, sondern ein System on Module. Diese benötigen eine eigens entwickelte Trägerplatine, in die sie eingesetzt und mittels eines Board-to-Board Steckers verbunden werden. Die Module besitzen selbst keine weiteren Steckverbinder, mit Außnahme eines Antennensteckers, und führen alle Schnittstellen über den Board-to-Board Stecker heraus.

Das Compute Module 4 basiert auf dem BCM2711-SoC. Dieser verfügt über vier Cortex-A72 Kerne @ 1,5 GHz und eine VideoCore VI GPU[10]. Durch Übertaktung kann der Takt der CPU auf bis zu 2,143 GHz angehoben werden, sofern eine ausreichende Wärmeabführung sichergestellt ist. Das Modul ist mit 1, 2, 4 und 8 GB LPDDR4 Arbeitsspeicher und mit und ohne WLAN-Schnittstelle erhältlich. Optional verfügt das Modul außerdem über bis zu 32 GB eMMC-Speicher, der die sonst übliche SD-Karte als Massenspeicher ersetzt.

Mit der Trägerplatine wird das Modul über zwei 100-Pin Board-to-Board Verbinder verbunden. Über diese stellt das Compute Module die von anderen Raspberry Pis üblichen 28 GPIOs zur Verfügung, welche mit alternativen Funktionen wie I^2C, SPI oder UART belegt werden können. Außerdem ist eine PCIe 2.0 Schnittstelle, ein OTG-fähiger USB2-Port und zwei CSI-Schnittstellen vorhanden. Da das Modul alle Spannungswandler mitbringt muss es nur mit 5 V versorgt werden, die interne 3,3 V und 1,8 V Spannung werden herausgeführt und können von der Trägerplatine mitgenutzt werden.

3.1.2 Xilinx Zynq MPSoC

Alternativ zu einer Raspberry-Pi basierten Lösung bietet sich ein Xilinx Zynq MPSoC als zentrale Recheneinheit an. Xilinx Zynq MPSoCs kombinieren eine leistungsstarke ARM-

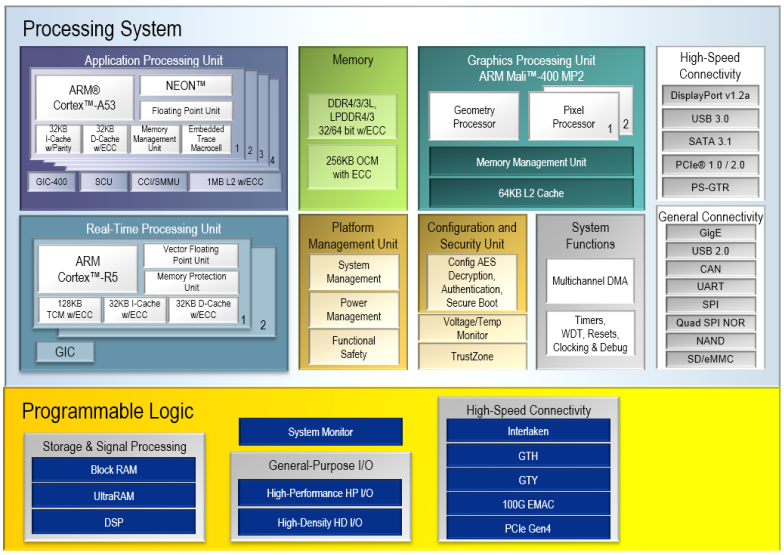


Abbildung 3.2: Blockdiagram Xilinx Zynq UltraScale+ EG

Quelle: Xilinx

CPU (Processing System, PS) mit einem FPGA (Programmable Logic, PL) in einem Chip, sodass sich die Vorteile beider Architekturen nutzen lassen. In Abbildung 3.2 wird dies in einem Blockdiagramm eines Zynq aus der ZU3EG-Reihe gezeigt.

Für eine vereinfachte Hardwareentwicklung sollte auch hier auf fertige Module zurückgegriffen werden, auf denen der SoC inklusive der nötigen Peripherie wie DDR-Speicher, eMMC und Spannungswandler unterkommen. So fallen das aufwändige BGA-Fanout und komplizierte Routing von DDR-Signalen weg, welches gleichzeitig Kosten spart und Risiken minimiert.

Das Mars XU3-Modul der schweizer Firma Enclustra bringt den bereits genannten Xilinx Zynq UltraScale+ ZU3EG inklusive aller Peripherie im Formfaktor eines DDR2-SODIMMs mit. Dieses hat Außenmaße von 67,6 mm × 30 mm und ist somit besonders für den Einsatz in einem 1:87 Fahrzeug geeignet. Das Blockdiagramm in Abbildung 3.3 zeigt den Aufbau und die nach außen geführten Schnittstellen.

Der FPGA des Zynq eignet sich um spezielle Aufgaben sehr effizient zu lösen. So kann die CPU beispielsweise von Bildverarbeitungsketten oder neuronalen Netzen entlastet werden, indem diese Aufgaben von dem FPGA übernommen werden.

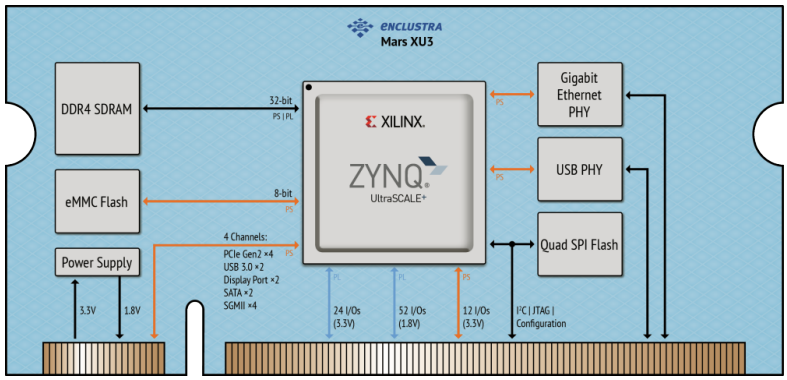


Abbildung 3.3: Blockdiagram Mars XU3

Quelle: Enclustra

Während die Kombination aus CPU und FPGA viel Potential bietet, erschwert dies die Software-Entwicklung gleichzeitig erheblich. Fertige Debian- oder Ubuntu-Images sind in der Regel nicht vorhanden und müssen selbst erstellt werden. Bei der Verwendung von Kameras kann nicht einfach auf fertige Schnittstellen zurückgegriffen werden, wie dies beispielsweise beim Raspberry Pi der Fall ist, sondern müssen zunächst PL-seitig implementiert werden. So entstehen weitere Abhängigkeiten zwischen PL (HDL- und Block-Design) und PS (Bootloader, Device-Trees), welche Entwicklungszyklen weiter in die Länge ziehen.

Auch die Hardwareentwicklung ist gegenüber einem CPU-SoM aufwändiger. Durch die große Flexibilität des FPGA ergeben sich gleichzeitig sehr viele Constraints, wie die IOs genutzt werden können. Bei bestimmten IO-Funktionen, wie beispielsweise CSI, sind sehr geringe IO-Spannungen nötig, was den Einsatz von Level-Shiftern an allen anderen Pins der IO-Bank voraussetzt. Verfügbare Module, wie das Mars XU3, verfügen meistens nicht über eine WLAN-Schnittstelle, welche daher zusätzlich auf der Trägerplatine vorgesehen werden muss.

3.1.3 Erwähnenswerte Alternativen

Für embedded ML ist das NVIDIA Jetson nano weit verbreitet. Dies ist ebenfalls ein SoM und bietet neben einer 4-Kern ARM CPU eine 128-Kern GPU, die sich für die Verwendung mit CUDA eignet. Weiterhin bietet es viele High-Speed Schnittstellen wie CSI, PCIe und USB3, allerdings kein integriertes WLAN. Die Verbindung zur Trägerplatine

erfolgt über einen 260-poligen Randstecker, ähnlich dem eines DDR-Moduls. Das Modul hat die Maße 69,5 mm × 45 mm zuzüglich denen des nötigen Steckverbinders und eines großen Kühlkörpers, daher ist es für die Verwendung in einem 1:87 Fahrzeug nur bedingt geeignet. Im gleichen Formfaktor ist außerdem das NVIDIA Xavier NX verfügbar, welches noch mehr Rechenleistung bietet, allerdings noch mehr Kühlung erfordert.

Google Coral hat das Coral SoM im Angebot, welches dem Raspberry Pi Compute Module 4 stark ähnelt. Neben einer 4-Kern CPU und integriertem WLAN hat es auch eine Coral EdgeTPU an Board. Durch die geringen Maße von 48 mm × 40 mm ist es auch für die Verwendung in einem 1:87 Fahrzeug geeignet. In dieser Arbeit wird es allerdings nicht näher betrachtet.

3.1.4 Gegenüberstellung

In Tabelle 3.1 werden die vorgestellten Optionen für eine zentrale Recheneinheit tabellarisch verglichen.

3.2 Machine-Learning Beschleuniger

Neuronale Netze, die für Machine-Learning Anwendungen sehr häufig zum Einsatz kommen, sind sehr rechenintensiv. Für die Ausführung wird daher in der Regel auf spezialisierte Hardware-Beschleuniger zurückgegriffen, die einen deutlich höheren Durchsatz und geringere Stromaufnahme aufweisen. Dabei handelt es sich entweder um GPUs oder Tensor Processing Units (TPUs).

Der Durchsatz wird für Machine-Learning Beschleuniger, die mit Floating-Point arbeiten, in FLOPS (Floating Point Operations per Second) angegeben, für Beschleuniger die mit Fixed-Point Integern arbeiten in OPS (Operations per Second). Zur Verwendung von neuronalen Netzen mit Integer-basierten Beschleunigern ist in der Regel eine Quantisierung des Netzes für den verwendeten Beschleuniger notwendig.

3.2.1 Google Coral EdgeTPU

Die Google Coral EdgeTPU ist ein von Google entwickelter Machine-Learning Beschleuniger für die lokale Berechnung neuronaler Netze. Der Beschleuniger arbeitet mit 8-bit

	Raspberry Pi Zero W	Raspberry Pi CM4	Jetson Nano	Coral SoM	Mars XU3
CPU Cores	1 @ 1 GHz	4 @ 1,5 GHz	4 @ 1,43 GHz	4 @ 1,5 GHz	4 @ 1,2 GHz
RAM	512 MB	8 GB	4 GB	4 GB	2 GB
PCIe	nein	ja	ja	nein	ja
USB	2.0[3]	2.0	3.2 Gen 1	3.2 Gen 1	3.2 Gen 1
GPU	ja[2]	ja[2]	ja	ja[2] + TPU	ja[2] + FPGA
On-Board WLAN	ja	ja	nein	ja	nein
Debian-Images	ja	ja	ja[5]	ja[5]	nein
Länge	65 mm	55 mm	69,6 mm	48 mm	67,6 mm
Breite	30 mm	40 mm	45 mm[1]	40 mm	30 mm[1]
Leistungs-aufnahme	2,5 W	5 W bis 10 W[4]	5 W bis 10 W	6,2 W	10 W[4]

[1] Auf Grund des Steckverbinders wird mehr Platz benötigt
[2] Nicht für Compute-Aufgaben nutzbar (CUDA/OpenCL)
[3] Nicht auf 40-Pin Stecker vorhanden, muss über zusätzliche Drähte verbunden werden
[4] Keine offiziellen Daten verfügbar, daher Schätzung auf Basis von Maximum-Ratings und Messungen
[5] Nicht getestet

Tabelle 3.1: Gegenüberstellung verschiedener Rechenplattformen

Integern und erreicht einen Durchsatz von 4 TOPS bei einer Leistungsaufnahme von 2 W[4].

Die EdgeTPUs sind in verschiedenen Formfaktoren mit unterschiedlichen Schnittstellen erhältlich, die verbaute TPU ist jedoch immer gleich. Für die Entwicklung eignet sich der Coral USB Accelerator. Dieser hat dieselben Maße wie ein Raspberry Pi Zero und wird über USB-C mit einem Computer verbunden. Für die Verwendung in fertiger Hardware sind weiterhin Varianten in m.2 und Mini PCIe vorhanden, die mittels PCIe mit dem Computer verbunden werden.

Für besonders kompakte Hardware ist die TPU ebenfalls als verlötbares Modul erhältlich. Das in Abbildung 3.4 gezeigte Modul ist 10 mm × 15 mm × 1,5 mm groß und bringt neben der TPU auch einen PMIC in dem Modul unter. Die Anbindung an das Host-System kann über USB2 und PCIe erfolgen.

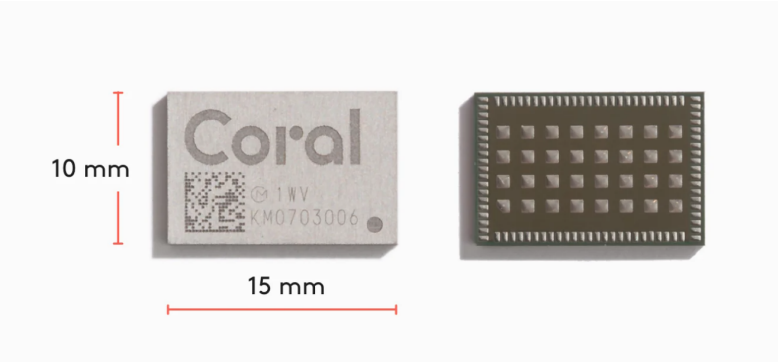


Abbildung 3.4: Bild eines Google Coral EdgeTPU Beschleunigermoduls

Quelle: Google Coral

3.2.2 FPGA

Die frei konfigurierbare Logik von FPGA macht es möglich, eine TPU in dem FPGA abzubilden.

Xilinx stellt für unterstützte Zynq-FPGAs die Vitis-AI Suite zur Verfügung. Kern für die Berechnungen bildet hier die Deep Learning Processing Unit (DPU), welche auf die Ausführung von CNNs optimiert ist und in die programmierbare Logik des Zynq integriert werden kann[15]. Die Suite stellt alle benötigten Werkzeuge bereit, um vorhandene Modelle aus verbreiteten Frameworks wie TensorFlow oder PyTorch für die DPU zu quantisieren, kompilieren und optimieren.

Das Projekt tinyTPU von Jonas Fuhrmann implementiert eine TPU in VHDL[3]. Mit diesem Ansatz kann die TPU in beliebigen FPGAs mit ausreichend Ressourcen verwendet werden. Die vorhandenen Werkzeuge sind allerdings noch nicht so umfangreich wie die anderer Anbieter.

3.2.3 Vergleich

Um eine Vergleichbarkeit herzustellen werden in Tabelle 3.2 einige Netze in Bezug auf die Inferenzgeschwindigkeit zwischen der EdgeTPU und einem FPGA mit Vitis AI verglichen. Da es bei der Beschleunigung von neuronalen Netzen sehr viele Randparameter gibt, dient dieser Vergleich nur zur Einordnung der zu erwartenen Performance. Bei den genannten Zahlen handelt es sich jeweils um Herstellerangaben.

Für die Zahlen der TPU wurde das Coral Dev Board verwendet[1]. Die Zahlen des FPGAs stammen von einem Ultra96 Board, welches den bereits angesprochenen Zynq UltraScale+ ZU3EG verwendet[6].

Der Vergleich zeigt, dass die Performance stark von der verwendeten Netzarchitektur abhängt und keine allgemeine Aussage getroffen werden kann. In der Regel ist jedoch davon auszugehen, dass die Google EdgeTPU in etwa die doppelte Performance eines ZU3EG bieten kann.

Model	TPU	FPGA
InceptionV1 (224x224)	4,5 ms	16,96 ms
InceptionV4 (299x299)	102 ms	88,76 ms
MobileNetV1 (224x224)	2,4 ms	5,97 ms
MobileNetV2 (224x224)	2,6 ms	10,17 ms

Tabelle 3.2: Vergleich der Performance zwischen Google EdgeTPU und FPGA (Vitis AI)

3.3 Sensorik

3.3.1 Odometrie

Unter Odometrie fällt die Schätzung der Bewegung eines Fahrzeugs anhand von Daten, die an den Aktuatoren des Fahrzeugs erhoben werden[13, Kap. 20.1].

In unserem Fall bieten sich die Hinterachse des Fahrzeugs als Odometrie-Quelle, da hier vergleichsweise einfach Sensoren angebracht werden können. Es kommen zwei Möglichkeiten der Datenerhebung in Frage.

Tick-basiert

Bei einer Tick-basierten Odometrie an der Hinterachse kommt ein binärer Sensor zum Einsatz, der ein oder mehrere Impulse pro Radumdrehung erzeugt. Der Sensor kann optisch oder magnetisch arbeiten.

Ein Tick-basierter Ansatz hat den Vorteil, dass der Aufbau sehr einfach ist und nur der Radumfang bekannt sein muss.

Nachteilig ist die geringe Auflösung, da nur ganze Ticks gezählt werden können. Bewegungen, die kürzer als ein Tick sind, werden nicht erfasst. Weiterhin kann mit einem einzelnen Sensor nicht die Drehrichtung erfasst werden. Mittels zweier Sensoren könnte die Drehrichtung über eine Quadratur erfasst werden, dies wäre aber mechanisch aufwändiger.

Hall-basiert

Als Alternative zur Tick-basierten Odometrie an der Hinterachse kann ein 3D-Hallsensor verwendet werden, um die genaue Lage der Hinterachse messen zu können. Somit ergibt sich ein Absolut-Encoder.

Ein 3D-Hall Sensor wie der Infineon TLV493d wird dafür oberhalb der Hinterachse montiert. In einer Aufnahme werden zwei Magneten orthogonal zur Achse angebracht, so dass sich bei der Rotation das Magnetfeld durch den Sensor dreht. Der Aufbau ist in Abbildung 3.5 dargestellt.

Wie in [7] beschrieben kann so nach einer Kalibrierung die Ausrichtung der Hinterachse α kann so mittels des Arcustangens anhand der Feldstärke X und Y berechnet werden (vereinfacht):

$$\alpha = \arctan \frac{Y}{X}$$

Gegenüber der Tick-basierten Odometrie lassen sich so auch sehr kleine Bewegungen im Bereich weniger Grad messen. Mittels zwei aufeinander folgenden Messungen kann außerdem die Drehrichtung bestimmt werden.

3.3.2 Distanzsensoren (LIDAR)

Viele autonome Straßenfahrzeuge setzen auf die Verwendung eines LIDAR-Scanners, sowie Ultraschall- und RADAR-basierte Distanzsensoren. Da derzeit keine für den Maßstab passenden LIDAR-Scanner auf dem Markt erhältlich sind, wird auf dessen Verwendung vorerst verzichtet. Als Distanzsensoren eignen sich beispielsweise STMicrolectronic

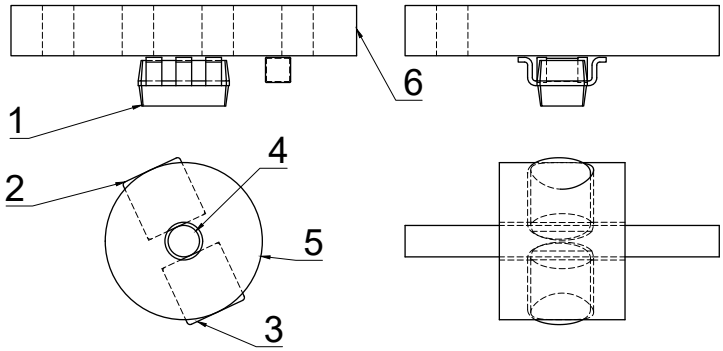


Abbildung 3.5: Aufbau der Hall-Sensor Odometrie. 1: TLV493D Hall-Sensor 2: Magnet (Nordpol) 3: Magnet (Südpol) 4: Achse 5: Aufnahme für Magnete 6: Trägerplatine für TLV493D

Quelle: Markus Kasten

VL53L1X Time of Flight Sensoren. Im Rahmen dieser Arbeit wird auf dessen Verwendung allerdings nicht näher eingegangen.

3.4 Lokalisierung

Für eine Navigation auf einer Karte wird in der Regeln eine globale Position des Fahrzeugs benötigt. Da in einer Modellbau-Welt in Innenräumen kein GNSS wie GPS oder Glonass zur Verfügung steht, muss eine alternative Möglichkeit der Positionsbestimmung gefunden werden. Die Positionsbestimmung in Innenräumen wird auch Indoor-Positioning genannt. Im folgenden werden zwei Ansätze diskutiert, wie dies für ein Straßenfahrzeug im Maßstab 1:87 umgesetzt werden kann.

3.4.1 Overhead-Kamera

Für eine sehr einfache und genaue Positionsbestimmung innerhalb der Modellbau-Welt können eine oder mehrere von oben auf die Welt gerichtete Kameras verwendet werden.

Das Fahrzeug wird dafür mit einem Marker ausgestattet, der im Kamerabild gut zu erkennen ist.

Für die folgenden Berechnungen wird eine Raspberry Pi HQ-Kamera mit 6 mm Objektiv angenommen. Der Sensor hat eine Breite x von 6,287 mm und Höhe y von 4,712 mm, somit ergibt sich folgender Field of View:

$$FoV_x = 2 \arctan \frac{x}{2f} = 2 \arctan \frac{6{,}287\,\text{mm}}{2 \cdot 6\,\text{mm}} = 55{,}3°$$
$$FoV_y = 2 \arctan \frac{y}{2f} = 2 \arctan \frac{4{,}712\,\text{mm}}{2 \cdot 6\,\text{mm}} = 42{,}8°$$

Wird die Kamera in der Höhe $H = 1{,}5\,\text{m}$ über der Modellbauwelt montiert, berechnet sich der Ausschnitt mit der Breite w und Höhe h wie folgt:

$$w = 2H \tan \frac{FoV_x}{2} = 2 \cdot 1{,}5\,\text{m} \tan \frac{55{,}3°}{2} = 1{,}48\,\text{m}$$
$$h = 2H \tan \frac{FoV_y}{2} = 2 \cdot 1{,}5\,\text{m} \tan \frac{42{,}8°}{2} = 1{,}11\,\text{m}$$

Um die vollständige Fläche des Mikrowunderlandes abzudecken sind somit zwei Kameras notwendig. Bei einer Auflösung von 2028 px × 1520 px (volle Auflösung des Sensors mit 2x2 Pixel-Binning) ergibt sich so eine theoretische Genauigkeit (auch Ground Sampling Distance genannt) von

$$\frac{1{,}48\,\text{m}}{2028\,\text{px}} = 0{,}73\,\text{mm}\,\text{px}^{-1}$$

. Um höhere Bildraten zu erreichen kann die Auflösung weiter reduziert werden, ohne dass die Genauigkeit zu gering wird.

Vorteile in diesem Ansatz liegen in der sehr hohen Genauigkeit und dem einfachen Aufbau bei kleinen Modellbau-Welten. Abhängig von den verwendeten Markern kann so sowohl die absolute Position als auch die Ausrichtung des Fahrzeugs erfasst werden.

Nachteilig ist die Abhängigkeit von einer guten Ausleuchtung bei Verwendung von Markern sowie die Beschränkung auf kleine Modellbau-Welten. Würde das Prinzip im Miniatur-Wunderland angewendet werden, wären, angenommen die komplette Welt wäre befahrbar, für die komplette Fläche bis zu 1000 Kameras nötig. Außerdem ist das System von der Sichtbarkeit des Fahrzeugs abhängig, wird das Fahrzeug von Unterführungen oder einer Hand verdeckt ist keine Positionsbestimmung mehr möglich. Durch die Nutzung einer

extern platzierte Kamera ist das Fahrzeug außerdem davon abhängig, dass die Position an das Fahrzeug übermittelt wird und kann die Position nicht selbstständig bestimmen.

3.4.2 Ultra-Wideband

Eine alternative Möglichkeit ist die Positionsbestimmung mittels Ultra Wideband (UWB). UWB ist eine Funktechnologie, die mit geringen Energien und sehr hohen Bandbreiten ($\geq$500 MHz) arbeitet und daher nicht andere Funktechnologien beeinträchtigt. Die hohe Bandbreite erlaubt es, die Signallaufzeit (Time of Flight, ToF) sehr genau zu messen. So kann über eine Triangulation von drei oder mehr Distanzmessungen zu fest aufgestellten Beacons die Position eines Teilnehmers bestimmt werden.

Der Hersteller Decawave bietet Module an, die laut Hersteller bei der Positionsbestimmung eine Genauigkeit von $\pm$10 cm erreichen.

Vorteile in der Verwendung von UWB liegen in der Unabhänigkeit von der Beleuchtungssituation und Sichtbarkeit des Fahrzeugs, sowie der besseren Skalierung für große Modellbau-Welten wie das Miniatur-Wunderland Hamburg. Weiterhin ist das Fahrzeug nicht von der Übermittlung der Position abhängig, da die absolute Position selbstständig bestimmt werden kann und lediglich externe Beacons notwendig sind.

Nachteilig ist die geringere Genauigkeit von bestenfalls $\pm$10 cm, was auf den Maßstab umgerechnet $\pm$8,7 m ergibt. Dies liegt jedoch noch in der Größenordnung der Genauigkeit von GPS[5]. Auf dem Fahrzeug ist außerdem weitere Hardware notwendig, welche sich in dem Maßstab aber ohne Probleme unterbringen lässt.

4 Umsetzung

In diesem Kapitel werden zwei konkrete Umsetzungen eines Fahrzeugs für das autonome Fahren im Maßstab 1:87 vorgestellt und diskutiert. Dabei wird auf die im vorherigen Kapitel gezeigten Konzepte zurückgegriffen

Für die Fahrzeugplattformen wurde die Namensgebung “TinyCar” genutzt.

4.1 TinyCar Interface Standard v1 (TCISv1)

Bevor ein Fahrzeug aufgebaut wird, soll ein einheitlicher Standard geschaffen werden, nach dem zukünftige Fahrzeuge aufgebaut werden können. Dieser wird “TinyCar Interface Standard v1” (kurz TCISv1) genannt. Ziel ist es, Kompatiblität zwischen verschiedenen Hardwareplattformen, sowohl auf Software, als auch Hardwareseite herzustellen. In dem Standard werden die verwendeten Stecker, dessen Pinbelegungen und die verwendete Sensorik und Aktorik weitgehend festgelegt.

Die Entscheidungen für die einzelnen Vorgaben wurden auf Basis von Erfahrungen mit der jeweiligen Peripherie getroffen.

4.1.1 On-Board Peripherie

Die On-Board Peripherie umfasst alles, was auf der Hauptplatine aufgebaut wird.

IMU

Als IMU wird eine Bosch BNO055 genutzt.

Motortreiber

Sofern der vorgesehene Antrieb es erlaubt sollte auf den Texas Instruments DRV8837 Motortreiber zurückgegriffen werden.

4.1.2 Externe Peripherie

Externe Peripherie umfasst die Peripherie, die mittels Steckverbinder mit der Hauptplatine verbunden wird. Im folgenden werden die zu verwendenen Schnittstellen festgelegt.

I^2C-Ports

I^2C-Ports erlauben das Anschließen externer I^2C-Peripherie. Es wird ein vierpoliger JST-SH Steckverbinder vorgeschrieben. Die Spannungsversorgung VCC sollte schaltbar sein, so dass doppelt belegte I2C-Adressen ggf. umkonfiguriert werden können. Die Signalpegel sind auf 3.3V ausgelegt.

Pin	Funktion
1	VCC
2	SDA
3	SCL
4	GND

Tabelle 4.1: Pinbelegung der I^2C-Stecker

Motor

Der Motor wird über einen 2-Pin JST-SH Verbinder angebunden. Die Spannung beträgt nominal 3,8 V und kann bei Betrieb mit einer LiPo-Zelle zwischen 3,0 V und 4,2 V liegen. Die Drehzahl wird mittels PWM geregelt.

Bei positiver Spannung, also 3,8 V an Motor+, fährt das Fahrzeug vorwärts, bei negativer Spannung rückwärts.

Es sollte auf einen 7mm Brushed Motor mit Getriebe zurückgegriffen werden. Die Referenz ist Mikromodellbau.de Artikel Nr. KG-1048.

Pin	Funktion
1	Motor +
2	Motor -

Tabelle 4.2: Pinbelegung der Motor-Stecker

Lenkservo

Der Lenkservo wird über einen JST-ZH Verbinder angeschlossen. Es werden Analog-Servos mit einer Pulseweite zwischen 1 ms und 2 ms verwendet.

Pin	Funktion
1	VCC
2	GND
3	Signal

Tabelle 4.3: Pinbelegung des Servo-Stecker

Als Referenz-Servo wird das Mikro-Servo S25 verwendet.

Kameras

Als Kamera-Verbinder wird ein 22-poliger 0.5mm Flat Flex Cable (FFC)-Verbinder mit der Pinbelegung des Raspberry Pi Zero (siehe Abbildung 4.1) vorgesehen.

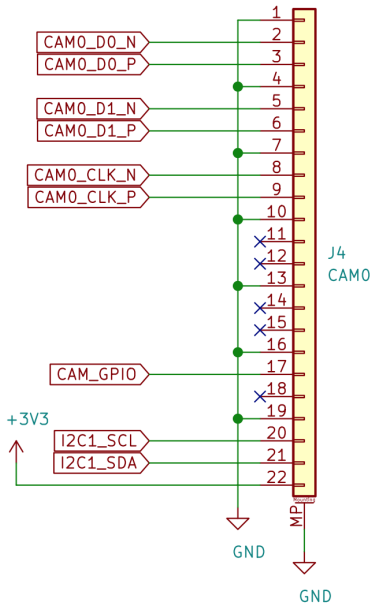


Abbildung 4.1: Pinbelegung des 22-poligen Kamera-Steckers

Quelle: Markus Kasten

Distanzsensoren

Für VL53L1X-Distanzsensoren wird ein 6-poliger 0.5mm FFC-Verbinder vorgesehen. Über diesen lassen sich mehrere Distanzsensoren in Reihe schalten. Die Spannungsversorgung und Signalpegel beträgt 2,8 V, um einen Level-Shifter und Spannungsregler auf den Sensor-Platinen einzusparen. Die Pinbelegung wird in Abbildung 4.2 gezeigt.

Details zur Funktionsweise der Distanzsensoren sind nicht Teil dieser Arbeit.

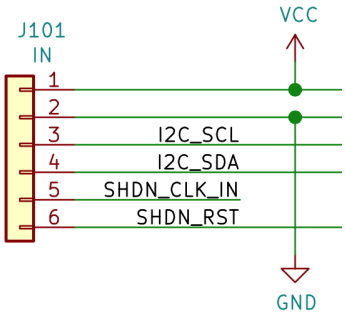


Abbildung 4.2: Pinbelegung des 6-poligen Distanzsensor-Steckers

Quelle: Markus Kasten

4.2 TinyCar Zero

Für erste Versuche wird eine einfache, kostengünstige Plattform auf Basis eines Raspberry Pi Zero W mit dem Namen "TinyCar Zero" aufgebaut. Ziel ist es, erste Erfahrungen mit einem Raspberry Pi als zentrale Recheneinheit zu sammeln und die verwendete Peripherie zu testen.

4.2.1 Prototyp

Als erstes wird ein Prototyp aus einzelnen Modulen wie in Abbildung 4.3 aufgebaut. Alle Komponenten werden an einem 3D-gedrucktem Chassis montiert. Mit diesem Prototypen wird zunächst das Sammeln von Daten erprobt und verfügt über keinen Lenkservo.

Für den Prototypen wird folgende Peripherie verwendet:

- BNO055
- LiPo Charger + 5V DCDC Wandler + 500mAh LiPo Akku
- IMX219 160° Kamera
- TLV493d als Lenkwinkel-Sensor

Abbildung 4.3: “TinyCar Zero”-Prototyp unterwegs im Microwunderland

Quelle: Markus Kasten

- Eine einfache Low-Side MOSFET Schaltung als Motortreiber

Da die ersten Versuche mit dem Prototypen erfolgreich waren sollen nun alle Module und Drähte auf einer Addon-Platine für den Raspberry Pi Zero W untergebracht werden.

4.2.2 Raspberry Pi uHAT

Die Raspberry Pi Foundation hat speziell für Addon-Platinen für Raspberry Pis die HAT (Hardware Attached on Top)-Spezifikation[8] entworfen. Diese macht sowohl mechanische als auch elektrische Vorgaben, wie eine Addon-Platine für Raspberry Pis entworfen werden sollte. Das genaue Einhalten der Spezifikation ist zwar nur für kommerzielle Produkte die den Namen HAT tragen entscheidend, dennoch wird die Spezifikation hier verwendet um die maximale Kompatibiltät zu erzielen.

In diesem Fall wird passend für den Raspberry Pi Zero W ein Micro-HAT (uHAT) entworfen. Dieser hat die Außenmaße 30 mm × 65 mm und ist somit genau so groß wie der Raspberry Pi Zero W und Coral USB Accelerator.

Bei der Auswahl der Peripherie wird iterativ vorgegangen um sicherzugehen, dass alles auf der Platine unterkommt. Folgende Auswahl wurde getroffen:

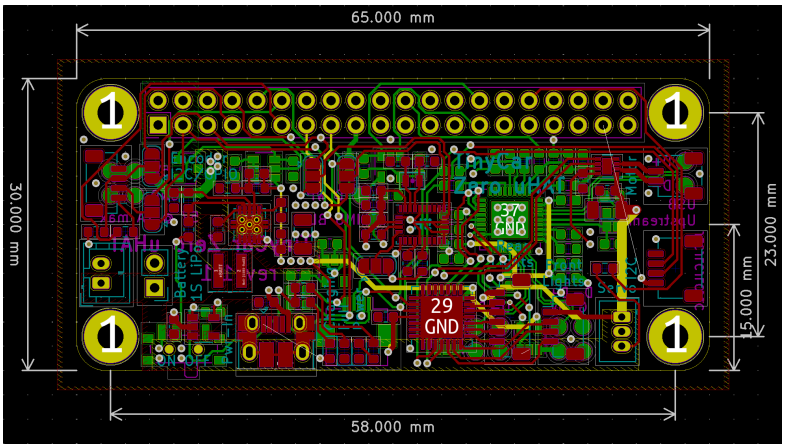


Abbildung 4.4: “TinyCar Zero” uHAT-Layout

Quelle: Markus Kasten

- TPS61230DRC Step-Up Wandler
- MCP73831 LiPo Laderegler
- BNO055 IMU
- DRV8837 Motor-Treiber
- PCA9685 für PWM
- USB-Hub für eventuelle Erweiterungen
- Anschlüsse für externe I^2C-Sensoren
- JST-SH Stecker für TLV493d und Motor

Das fertige Layout der vierlagigen Platine ist in Abbildung 4.4 dargestellt. Der Schaltplan findet sich in Anhang B.

4.2.3 Fahrzeug-Chassis

Es wird ein speziell auf das Fahrzeug zugeschnittenes Chassis entworfen, an dem alle Komponenten befestigt werden können. Das Chassis wird mittels Fused Deposition Modelling (FDM) 3D-Druck aus PLA hergestellt.

4.2.4 Peripherie am Fahrzeug

Die Peripherie wird weitgehend nach den Vorgaben aus dem TCISv1 aufgebaut. An der Hinterachse wird die in Kapitel 3.3.1 angesprochende Hallsensor-basierte Odometrie untergebracht. Vor der Vorderachslenkung wird ein weiterer 3D-Hallsensor montiert um den Lenkwinkel auslesen zu können.

Als Kamera wird eine Raspberry Pi Kamera V2 eingesetzt. Da die Kamera nur einen horizontalen Blickwinkel von ca. 62° hat, wird das Kamera-Modul von der Platine genommen und wie in Abbildung 4.5 gezeigt gegen ein Modul mit 160° Blickwinkel ersetzt.

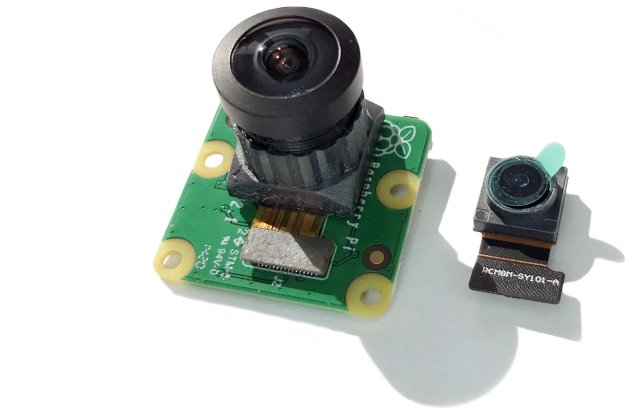


Abbildung 4.5: Modifizierte Raspberry Pi Kamera: Links das Raspberry Pi Kamera-V2 Board mit 160°-Modul, rechts das orignale Kamera-Modul

Quelle: Markus Kasten

Das Lenkservo wird liegend auf der Unterseite des Fahrzeugs befestigt. Über einen Umlenkhebel wird die Lenkung betätigt. Auf der Unterseite wird wie in Abbildung 4.6 gezeigt ebenfalls der Motor mit Getriebe befestigt.

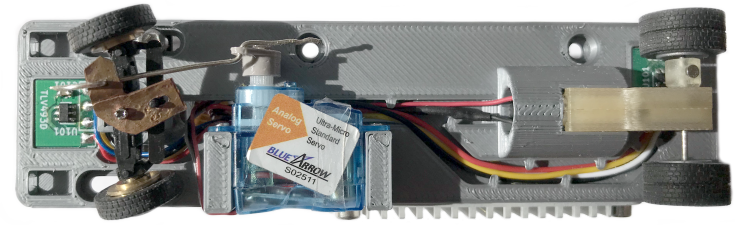


Abbildung 4.6: Auf der Unterseite angebrachtes Servo, mit Umlenkhebel zur Betätigung der Lenkung

Quelle: Markus Kasten

4.2.5 Machine-Learning Beschleuniger

Zur Beschleunigung von Machine-Learning Aufgaben wird ein Coral USB-Accelerator unterhalb des Raspberry Pi angebracht. Die USB-Verbindung wird über die sich auf der Unterseite des Raspberry-Pis befindlichen Testpads hergestellt, da ein Micro-USB Stecker zu breit wäre. Der USB-C Stecker wird aus einem Breakout-Board selbst gebaut.

4.2.6 Sonstige Peripherie

Ein USB-Hub auf dem uHAT soll für eine bessere Erweiterungsfähigkeit in der Zukunft sorgen. Der USB Hub kann mit dem USB-Port des Raspberry Pi Zero W verbunden werden. So können über den USB-Hub beispielsweise mehrere Coral USB-Beschleuniger angeschlossen werden, oder weitere Raspberry-Pis mittels USB over Ethernet.

Bisher ungenutzte Kanäle des PCA9685 werden an zwei weitere JST-SH Steckverbinder angeschlossen, beispielsweise für Beleuchtung am Fahrzeug.

Ein LiPo Lade-IC (MCP73831) wird genutzt um den angeschlossenen LiPo-Akku über USB aufzuladen.

4.2.7 Inbetriebnahme

Nachdem das erste Fahrzeug zusammengebaut wurde, werden einige Funktionstests durchgeführt. Hiermit wird zunächst geprüft, ob alles an Peripherie wie vorgesehen angesprochen werden kann. Dies erfolgt mittels einfacher Python Skripte, die Daten von Sensoren auslesen und die Aktorik wie Motor und Lenkservo ansteuern.

Diese Tests konnten erfolgreich abgeschlossen werden, so dass die Skripte in ROS-Nodes überführt werden. So soll zunächst die Fernsteuerung des Fahrzeugs ermöglicht werden. Hier zeigten sich allerdings erste Performance-Probleme, die zu starken Verzögerungen im Ansprechverhalten führten. Durch das Umschreiben der ROS-Nodes von Python in C++ konnte ein kleiner Performance-Gewinn erzielt werden, jedoch blieb die Auslastung der CPU hoch.

Somit wären nur wenig Ressource für komplexe Algorithmen übrig, wie sie für das autonome Fahren nötig sind. Es müssen also Ansätze gefunden werden, wie die Performance optimiert und Rechenleistung erweitert werden kann. Zur Erweiterung der Rechenleistung ist es möglich, weitere Raspbery Pi Zeros, oder kompatible Alternativen, in dem Fahrzeug unterzubringen. Dies würde die Komplexität der Softwareentwicklung allerdings deutlich erhöhen, da nun zusätzlich Daten über das Netzwerk synchronisiert werden müssen.

Ein weiteres Problem besteht darin, dass die CPU des Raspberry Pi Zero die ARMv6-Architektur nutzt. ARMv6 gilt als veraltet, so dass viel Software nicht vorkompiliert verfügbar ist. Dazu zählt unter anderem der Google Coral (`libedgetpu`) Treiber, TensorFlow-Lite und sämtliche ROS-Pakete. Das eigenständige Kompilieren der Software ist nicht immer einfach und sehr zeitaufwändig, was die Arbeit mit der Plattform stark erschwert.

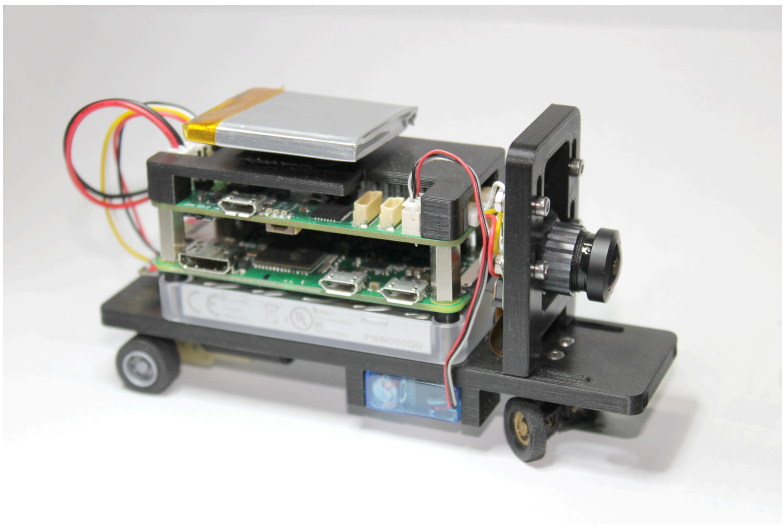


Abbildung 4.7: Aufgebautes TinyCar Zero

Quelle: Markus Kasten

4.3 TinyCar CM4

Da die ersten Tests mit dem Raspberry-Pi Zero W eine nicht zufriedenstellende Performance beim Einsatz von ROS zeigten wird eine weitere Umsetzung auf Basis eines Raspberry Pi Compute Module 4 evaluiert. Die moderne SoC- und CPU-Architektur soll somit auch zukünftigen Aufgaben gewappnet sein.

Das "TinyCar CM4" soll möglichst viele Elemente des "TinyCar Zero" übernehmen um die Entwicklung zu vereinfachen und Softwarekompatiblität zu maximieren. Es wird eine neue Trägerplatine entwickelt, die das Compute-Module aufnimmt.

Der Aufbau des Chassis wird einzig mit einer angepassten Montierung des CM4-Trägers übernommen. So kommen in dem Chassis weiterhin die Sensoren für Odometrie und Lenkung unter, ebenfalls wird die gleiche Kamera verwendet. Auch der Antrieb und die Lenkung werden unverändert übernommen.

4.3.1 Trägerplatine

Die zu entwickelnde Trägerplatine soll möglichst die gleichen Maße wie das Compute Module 4 haben. Erste Layout-Versuche ergaben, dass Außenmaße von 65 mm × 40 mm

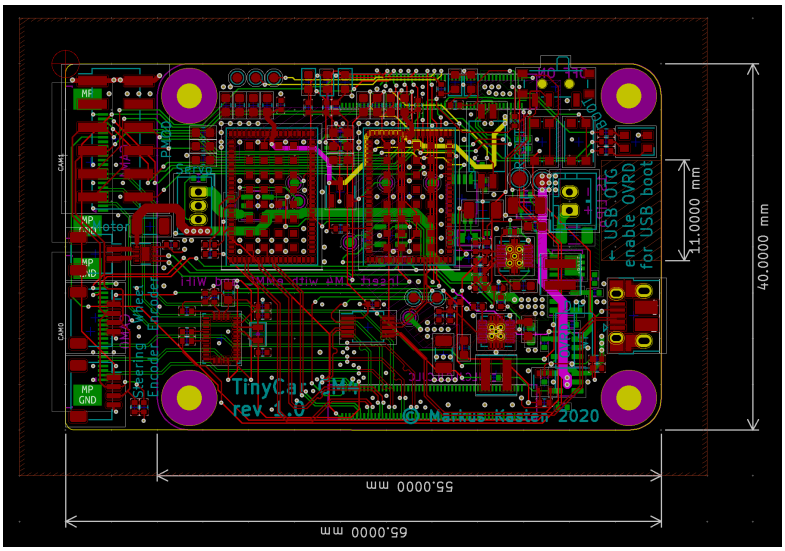


Abbildung 4.8: “TinyCar CM4” Trägerplatine

Quelle: Markus Kasten

realistisch sind. Somit hat die Platine dieselbe Höhe und ist nur 10 mm länger als das Compute-Module selbst, was das Unterbringen von zwei 22-Pin FFC-Verbindern für Kameras erlaubt. Für das finale Layout wird eine doppelseitig bestückte 6-Lagen Platine mit definierten Impedanzen benötigt. Die Platine ist in Abbildung 4.8 zu sehen. Für passive Bauteile wird die Größe 0402 (1 mm × 0,5 mm) verwendet.

Von dem uHAT wird die 5 V-Spannungsversorgung, IMU und Motorsteuerung übernommen, da sich diese als zuverlässig herausgestellt haben. Auch Anschlüsse für die im Chassis verbauten TLV493d Hall-Sensoren werden übernommen.

Der vollständige Schaltplan befindet sich in Anhang C.

4.3.2 Pmod-Schnittstelle

Da die Plattform ohne eine Neuentwicklung der Trägerplatine erweiterungsfähig sein soll, wird eine Schnittstelle für Addon-Boards vorgesehen. Der Pmod-Standard von Digilent[2] bietet sich an um auch mit bereits kommerziell erhältlichen Boards kompatibel zu sein.

Speziell fällt die Wahl auf das Pmod Interface 2A (extended SPI). Dieses bietet eine 4-Wire SPI Schnittstelle, einen Interrupt- und Reset-Pin sowie zwei nicht spezifierte Pins.

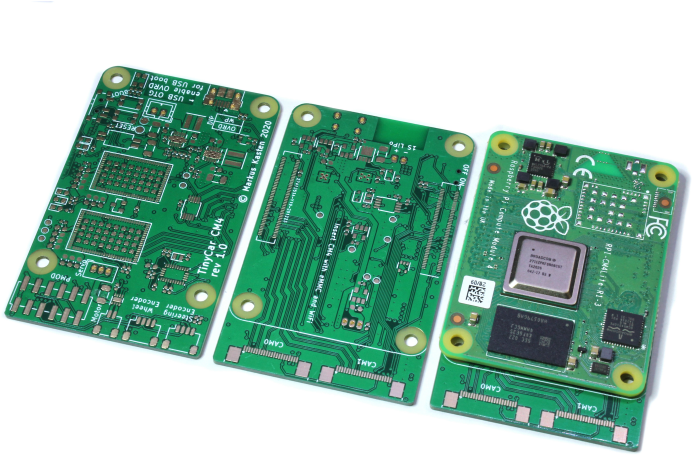


Abbildung 4.9: Platine des “TinyCar CM4”

Quelle: Markus Kasten

Die nicht spezifizierten Pins werden in diesem Fall für eine zusätzliche I^2C-Schnittstelle verwendet. Die Pin-Belegung ist in Abbildung 4.10 zu sehen.

An der Pmod-Schnittstelle kann beispielsweise ein UWB-Modul für Indoor-Lokalisierung angeschlossen werden. Hierfür wurde eine einfache Platine erstellt (siehe Abbildung 4.11), auf der ein DWM1000-Modul, ein IO-Expander sowie auf der Unterseite eine Schnittstelle für externe Distanzsensoren untergebracht sind erstellt.

4.3.3 Anbindung der Peripherie

Das Compute Module 4 bietet gegenüber zu dem Raspberry Pi Zero W einige neue Schnittstellen, die eine bessere Integration erlauben.

Auf dem Stecker des CM4 sind zwei CSI-Schnittstellen, so dass zwei Kameras verbunden werden können. Beide Schnittstellen werden herausgeführt, so dass in Zukunft auch mit Stereokameras oder unterschiedlichen Brennweiten gearbeitet werden kann.

Des weiteren bietet das CM4 einen PCIe Root-Complex. Über diesen wird ein Coral Modul angeschlossen.

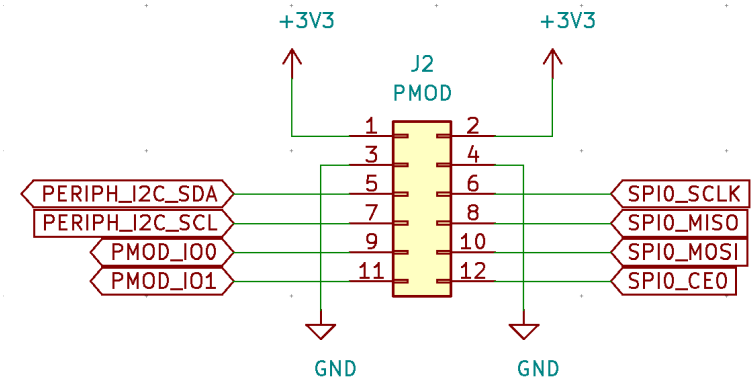


Abbildung 4.10: Pinbelegung des Pmod-Steckers seitens der Trägerplatine

Quelle: Markus Kasten

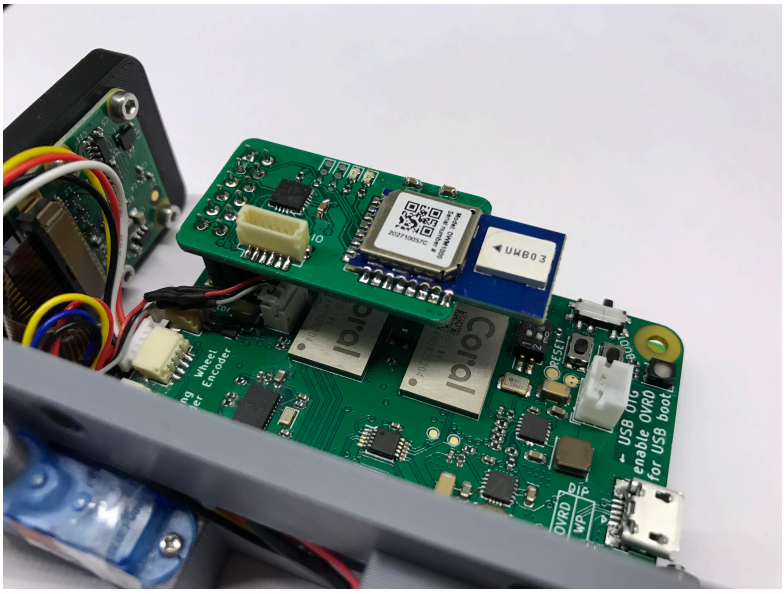


Abbildung 4.11: Pmod-Erweiterung mit IO-Expander und DWM1000 UWB-Modul

Quelle: Markus Kasten

IMU

Die BNO055-IMU wird über einen UART mit dem CM4 verbunden. Um weiterhin eine Debug-Konsole auf UART1 zu ermöglichen fällt die Wahl auf UART3. Eine Verbindung über I^2C ist nicht optimal, da der I^2C-Controller des BCM2711 kein I^2C Clock Stretching unterstützt, von welchem der BNO055 intensiven gebraucht macht.

Kameras

Beide CSI-Schnittstellen des CM4 werden mit jeweils zwei Datenlanes auf je einen 22-Pin FFC-Verbinder gelegt, wie er auch am Raspberry Pi Zero W vorzufinden ist. Auf die zwei zusätlich verfügbaren Datenlanes von CSI1 wird verzichtet, da die verwendeten Kameras nur von zweien gebrauch machen und somit der benötigte Platz für das Routing der Signale gespart werden kann.

Beide Kameras benötigen jeweils ein eigenes I^2C-Interface. I2C0 und I2C1 werden wie im Datenblatt beschrieben zugeordnet[10, Kap. 2.9 und 2.10].

PWM für Motor und Servo

Beim “TinyCar Zero”-uHAT wurden ausschließlich Hardware-PWM Pins des Raspberry Pis sowie ein zusätzlicher PWM-Chip verwendet. Bei der Entwicklung der ROS-Nodes für diese hat sich herausgestellt, dass für präzise PWM-Signale auf dem Raspberry-Pi allerdings kein Hardware-PWM notwendig ist. Die Verwendung von DMA-getriebenen PWM via `pigpiod` erzeugt für die Anwendung ausreichend saubere Signale und erleichtert sowohl die Hardware- als auch Softwareenticklung.

Als Motortreiber wird wie im TCISv1 vorgesehen weiterhin der DRV8837 verwendet.

USB

Das Compute Module bietet nur eine USB2-Schnittstelle. Dieser wird auch benutzt, um den eMMC Speicher von einem Computer über einen Micro-USB Port beschreiben zu können, daher kann der Port nicht direkt mit der TPU verbunden werden. Weiterhin soll die Möglichkeit bestehen, dass externe USB-Geräte über USB-OTG angeschlossen werden können.

Um dies zu ermöglichen wird der USB-Port vom CM4 mit einem USB-Multiplexer (ON Semiconductor FSUSB42MUX) zwischen der TPU und dem Micro-USB Port umgeschaltet. Als SEL-Eingang des Multiplexers wird hier das ID-Signal des Micro-USB Ports und ein DIP-Schalter herangezogen.

Der mit "OVRD" beschriftete DIP-Schalter ermöglicht es, den Multiplexer manuell auf den Micro-USB Port umzuschalten. Dies ist notwendig, wenn das Compute Module als USB-Device betrieben wird, wie es beispielsweise beim Beschreiben des eMMC über USB notwendig ist.

Auf einen zusätzlichen USB-Hub wurde zur Vereinfachung des Designs verzichtet.

I2C-Geräte

I2C0 und I2C1 sind bei der Verwendung mit einer Kamera mit der GPU des BCM2711 verbunden und nicht für das Host-Betriebssystem benutzbar. Daher muss auf einen anderen I^2C ausgewichen werden, die Wahl fällt hier auf I2C6.

I2C6 wird für beide TLV493d, die Pmod-Schnittstelle und die Real Time Clock (RTC) verwendet.

RTC

Auf dem CM4 ist keine RTC verbaut, so dass beim Neustart des Betriebssystems die Uhrzeit nicht bekannt ist und erst von einem NTP-Server bezogen werden muss. Damit die Uhrzeit auch in Umgebungen ohne Internetverbindung stimmt kann eine RTC verwendet werden, welche die Uhrzeit auch im abgeschalteten Zustand speichert und weiterzählt.

Die Wahl fällt auf den PCF85263ATT. Damit die Uhrzeit auch bei kurzzeitigem Abstecken des Akkus erhalten bleibt wird diese über einen Superkondensator gestützt. Wird der Superkondensator auf 3,3 V geladen und darf im Betrieb um 2 V abfallen, ergibt sich folgende Zeit, die der Superkondensator die RTC versorgen kann:

$$\frac{11\,\mathrm{mF} \cdot 2\,\mathrm{V}}{350\,\mathrm{nA}} = 17{,}46\,\mathrm{h}$$

Analog-Eingänge

Das CM4 bietet zwei Analog-Eingänge AIN0 und AIN1, welche mit dem PMIC des Compute Module verbunden sind und über die GPU-Firmware ausgelesen werden können. Die beiden Eingänge werden verwendet um die Batterie- und USB-Versorgungsspannung zu messen und vom Host-Betriebssystem auslesbar zu machen. Ein Spannungsteiler aus zwei 10 kΩ Widerständen bringt die Spannung in den sicheren Eingangsbereich der Eingänge.

Sonstige GPIO

Übrige GPIOs werden mit einer RGB-LED und einem Taster verwendet.

4.3.4 Stromversorgung

Die primäre Stromversorgung des Fahrzeugs erfolgt über eine Li-Ion 18650-Zelle des Typs Samsung "INR18650-35E" mit einer Kapazität von 3500 mA h. Bei einer geschätzten nominalen Stromaufnahme von 1 A ergibt sich so eine Akkulaufzeit von etwa 3 h. Eingangs ist das Board durch eine SMD-Schmelzsicherung sowie einem Verpolungsschutz abgesichert. Ein Unterspannungsschutz (Under Voltage Lockout, UVLO) schützt den Akku vor gefährlichen Tiefentladungen indem er die Spannungsversorgung des Boards unterhalb einer Batteriespannung von 3,0 V abschaltet. Der Eingangsspannungbereich liegt somit bei 3,0 V bis 4,2 V.

Ein vereinfachter Stromlaufplan wird in Abbildung 4.12 gezeigt. Der detailierte Aufbau der Stromversorgung befindet sich auf Seite 2 des Schaltplans.

4.3.5 Machine-Learning Beschleuniger

Für die Beschleunigung von Machine-Learning Aufgaben werden zwei Google Coral Module vorgesehen, so können zwei verschiedene Modelle gleichzeitig ausgeführt werden.

Da das Compute-Module 4 nur einen PCIe Root-Complex besitzt wird ein Coral Modul mittels PCIe und eins mittels USB2 angebunden. Um die Komplexität des Designs gering zu halten wurde auf den Einsatz von PCIe-Switches verzichtet, welche ein Fan-Out des PCIe Root-Complex ermöglicht hätten. Die Anbindung über USB2 hat zwar gegenüber

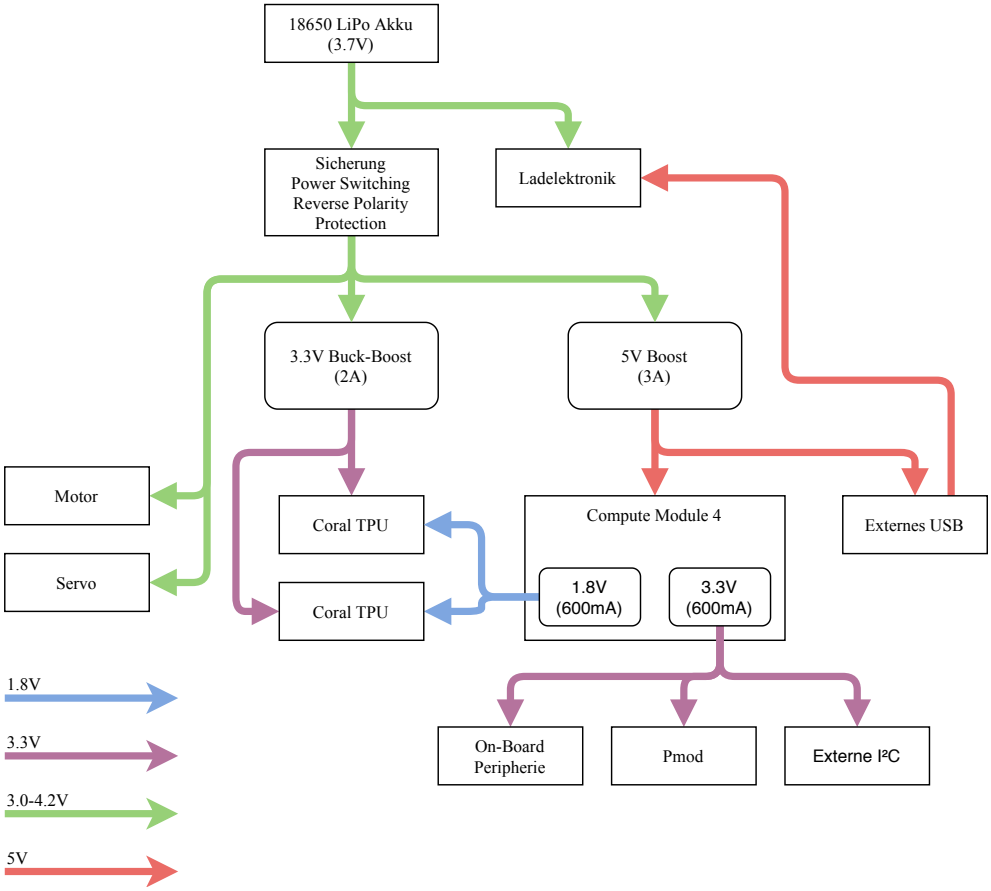


Abbildung 4.12: Vereinfachter Stromlaufplan der CM4-Trägerplatine

Quelle: Markus Kasten

PCIe einen deutlichen Performance-Nachteil, ist jedoch trotzdem um Größenordnungen schneller als die Ausführung auf der CPU.

4.3.6 Kühlkörper

Beim Betrieb des Compute-Modules ist zu erwarten, dass es ohne zusätzliche Kühlkörper zur Wärmeabführung schnell zur Überhitzung und einer damit verbundenen Leistungsreduktion (sog. Thermal Throtteling) kommen wird. Um dem entgegenzuwirken wird ein einfacher CNC-gefräster Aluminiumkühlkörper erstellt und gefertigt, der auf das CM4 geschraubt wird (vgl. Abbildung 4.13). Für eine noch bessere Wärmeabstrahlung wird dieser weiterhin eloxiert. Der Kontakt zum CM4 wird über spezielle Wärmeleitpads auf dem SoC hergestellt.

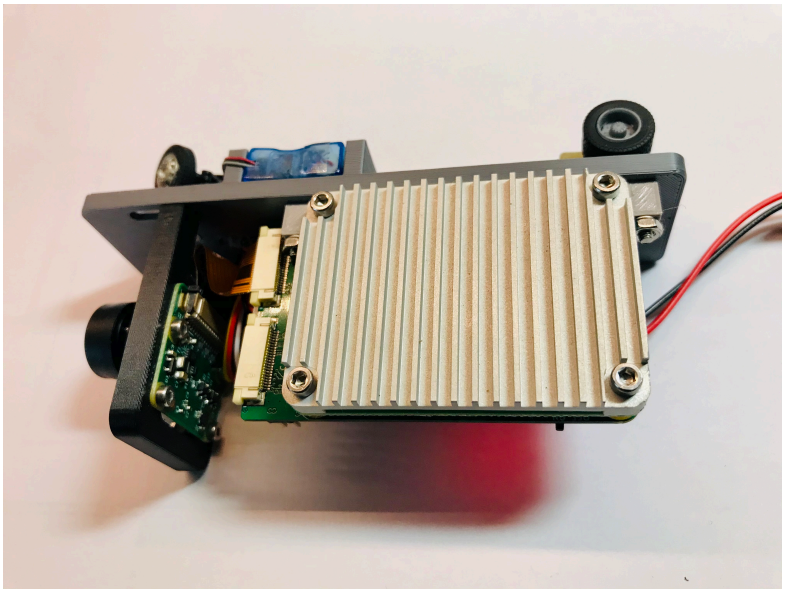


Abbildung 4.13: Auf dem CM4 montierter Kühlkörper aus eloxiertem Aluminium

Quelle: Markus Kasten

4.3.7 Konfiguration

Damit die genutzten Schnittstellen I2C6 und UART3, sowie USB verwendet werden können, müssen diese in der Bootloader-Konfiguration des Raspberry-Pi eingeschaltet werden. Dafür werden in der Datei `/boot/config.txt` folgende Zeilen hinzugefügt:

Abbildung 4.14: Das “TinyCar CM4” unterwegs im Mikrowunderland

Quelle: Markus Kasten

```
# enable I2C6
dtoverlay=i2c6

# enable UART3 for BNO055
dtoverlay=uart3

# enable USB in host mode
dtoverlay=dwc2,dr_mode=host
```

Desweiteren sind eigene Device-Tree-Overlays nötig, um die RTC und Stromversorgung der USB-Buchse zu aktivieren. Diese werden inkl. der nötigen Konfiguration als Board Support Package (BSP) zur Verfügung gestellt.

4.3.8 Aufbau des Fahrzeugs

Die Trägerplatine wird stehend auf dem Fahrzeug befestigt. Neben der Platine kommt die 18650-Zelle unter. Der vorläufige Aufbau ist in Abbildung 4.14 gezeigt.

4.3.9 Inbetriebnahme

Auch hier werden nach dem Aufbau des Fahrzeugs erste Funktionstests durchgeführt.

Nach einer groben optischen und elektrischen Inspektion wird die Trägerplatine zunächst ohne eingesetztes Compute-Modul an ein Labornetzgerät angeschlossen. So wird verhindert, dass beispielsweise falsch bestückte Spannungsteiler der Schaltwandler zur Zerstörung des Moduls führen. Es wird eine Spannung von etwa 3,7 V mit einer Strombegrenzung von 100 mA an die Platine angelegt. Die Stromaufnahme muss nun unter 10 mA liegen. Über die vorhandenen Testpads wird geprüft, ob die Spannungen der 5 V und 3,3 V Schienen korrekt sind.

Nun wird das Compute-Modul eingesetzt und die Strombegrenzung auf 2 A gestellt. Zunächst muss ein Betriebssystem-Image auf das Modul aufgespielt werden, dafür wird der DIP-Schalter für `OVRD` in die ON-Position gesetzt und die Trägerplatine über USB mit einem Computer verbunden. Nun wird das Modul eingeschaltet, während der Boot-Taster gedrückt wird. Mittels des Tools `rpiboot` kann der eMMC-Speicher des Moduls als Massenspeicher eingehangen werden und mit einem Image beschrieben werden. Darauf kann die USB-Verbindung getrennt werden und der DIP-Schalter zurück in die OFF-Position geschoben werden.

Für erste Funktionstests wird weitgehend auf die Skripte des "TinyCar Zero" zurückgegriffen, Anpassungen sind auf Grund der weitgehenden Kompatiblität der Fahrzeuge nur für die Ansteuerung der PWM und verwendeter Ports notwendig.

Zu diesen Funktionstests gehören:

- Ansteuerung der Servos
- Ansteuerung des Motors
- Test, ob Daten der IMU gelesen werden können
- Test, ob beide über I2C verbundene Hall-Sensoren angesprochen werden können
- Test der RGB-LED
- Test der Kameras mittels `raspistill`

Als nächstes wird geprüft, ob die verbauten Coral TPUs erkannt werden. Die mittels PCIe verbundene TPU wird in der Ausgabe von `lspci` (vgl. Abbildung 4.15) angezeigt. Zum Zeitpunkt des Schreibens dieser Arbeit gibt es für das Raspberry-Pi CM4 noch keinen vollständigen Treibersupport, daher können noch keine Inferenzen auf dieser ausgeführt werden. Die mittels USB verbundene TPU wird in der Ausgabe von `lsusb` angezeigt

```
pi@rpi-cm4-2g-32g-nw:~ $ sudo lspci -v
00:00.0 PCI bridge: Broadcom Limited Device 2711 (rev 20) (prog-if 00 [Normal decode])
        Flags: bus master, fast devsel, latency 0, IRQ 60
        Bus: primary=00, secondary=01, subordinate=01, sec-latency=0
        I/O behind bridge: 00000000-00000fff
        Prefetchable memory behind bridge: 00000000f8000000-00000000f81fffff
        Capabilities: [48] Power Management version 3
        Capabilities: [ac] Express Root Port (Slot-), MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [180] Vendor Specific Information: ID=0000 Rev=0 Len=028 <?>
        Capabilities: [240] L1 PM Substates
        Kernel driver in use: pcieport

01:00.0 System peripheral: Device 1ac1:089a (prog-if ff)
        Subsystem: Device 1ac1:089a
        Flags: fast devsel
        Memory at 600100000 (64-bit, prefetchable) [disabled] [size=16K]
        Memory at 600000000 (64-bit, prefetchable) [disabled] [size=1M]
        Capabilities: [80] Express Endpoint, MSI 00
        Capabilities: [d0] MSI-X: Enable- Count=128 Masked-
        Capabilities: [e0] MSI: Enable- Count=1/32 Maskable- 64bit+
        Capabilities: [f8] Power Management version 3
        Capabilities: [100] Vendor Specific Information: ID=1556 Rev=1 Len=008 <?>
        Capabilities: [108] Latency Tolerance Reporting
        Capabilities: [110] L1 PM Substates
        Capabilities: [200] Advanced Error Reporting
```

Abbildung 4.15: Die PCIe Coral TPU wird in der Ausgabe von `lspci` angezeigt (Device 1ac1:089a)

Quelle: Markus Kasten

```
pi@rpi-cm4-2g-16g-wl:~ $ lsusb
Bus 001 Device 003: ID 18d1:9302 Google Inc.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Abbildung 4.16: Die USB Coral TPU wird in der Ausgabe von `lsusb` angezeigt (`18d1:9302 Google Inc.`)

Quelle: Markus Kasten

(vgl. Abbildung 4.16). Auf dieser können auch die von Google bereitgestellten Beispiele erfolgreich ausgeführt werden.

4.3.10 Kostenaufstellung

In Tabelle 4.4 werden die Kosten für das “TinyCar CM4” grob zusammengefasst. Da die Kosten der Trägerplatine stark von der gefertigten Stückzahl abhängt wird ein Schätzwert genommen.

Position	Kosten[1]	Artikel-Nr.
Platine inkl. Bauteile	ca. 300 €[3]	Sonderanfertigung
Kühlkörper[2]	ca. 40 €[4]	Sonderanfertigung
Compute Module 4 (CM4103204)	66,90 €	RP1-402820 (Welectron)
Lenkung	14,99 €	1534267 - 62 (Conrad)
Motor	33,50 €	KG-1048 (Mikromodellbau.de)
Kamera (D160)	13,90 €	WS2-015264 (Welectron)
Kameramodul V2	25,95 €	409521-001 (Welectron)
Servo	8,95 €	245063 - 62 (Conrad)
18650-Zelle	9,99 €	1499572 - 62 (Conrad)
Hall-Sensoren	2 €	Sonderanfertigung
PLA-Filament	ca. 1 €	F10390 (dasfilament)
Summe	ca. 515 €	

[1] Preise inkl. MwSt vom 14.02.2021
[2] Privat gefertigt
[3] Stückkosten bei Stückzahl 3, EuroCircuits
[4] Stückkosten bei Stückzahl 5, 3dhubs

Tabelle 4.4: Kostenaufstellung des “TinyCar CM4”

5 Evaluation

In diesem Kapitel wird das im vorherigen Kapitel aufgebaute “TinyCar CM4” evaluiert und eine Reihe an Daten erfasst. So kann eine Vergleichbarkeit mit anderen Fahrzeugplattformen hergestellt werden.

Bei den erhobenen Messwerten handelt es sich um Werte der ersten Revision der Plattform.

Das Fahrzeug hat insgesamt ein Außenmaß von 115 mm × 33 mm × 64 mm bei einem Radstand von 85 mm. Das Gewicht des Fahrzeugs beträgt inklusive des 48 g schweren Akkus 150 g.

5.1 Odometrie

Für die Evaluation der Hallsensor basierten Odometrie werden die Rohdaten des Hallsensors aufgezeichnet. Aus den Daten lässt sich wie in Kapitel 3.3.1 angesprochen die zurückgelegte Strecke errechnen. Die verwendete Sampling-Frequenz des Sensors liegt bei 50 Hz, der Durchmesser des Reifens beträgt ca. 12 mm.

In Abbildung 5.1 werden die Rohdaten des Sensors mit der daraus errechneten Odometrie gezeigt. Die Kalibrierung erfolgte mit der in der Application Note genannten DFT-Methode[7].

Die Genauigkeit der Odometrie ist somit nur noch von dem Reifendurchmesser sowie dem Schlupf abhängig. Durch die Verwendung von Gummireifen wird der Schlupf minimal gehalten. Der Reifendurchmesser d lässt sich anhand einer Ground-Truth einfach korrigieren:

$$d = d_0 \cdot \frac{s_{groundtruth}}{s_{odometrie}}$$

.

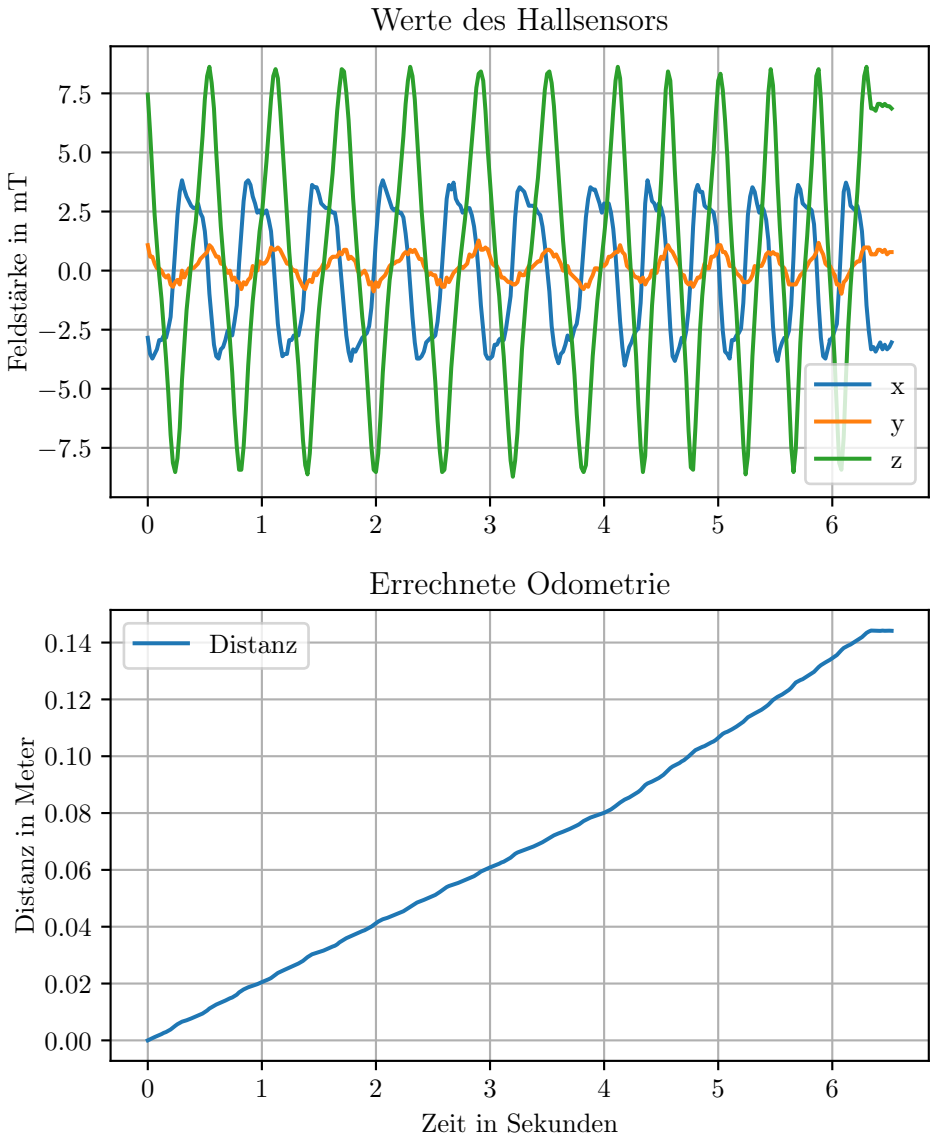


Abbildung 5.1: Visualisierung der Hallsensor-Werte und der daraus errechneten Odometrie

Quelle: Markus Kasten

5.2 Machine-Learning Performance

Die Performance der TPU wird in Kombination mit dem Compute-Module 4 charakterisiert. Als Model wird eine Implementation von BiSeNetV2[16] herangezogen, da dieses Netzwerk auch in der Arbeit von Luk Schwalb verwendet wird. Die Ein- und Ausgabeauflösung beträgt 256 px × 512 px bei zwei Klassen. Da zum Zeitpunkt des Schreibens nur die Treiber der per USB angebundenen TPU funktionsfähig sind können keine Werte für die PCIe-TPU erfasst werden.

Zum Vergleich wird das gleiche Model auf einem Desktop-Computer mit unterschiedlich angebundenen EdgeTPUs ausgeführt. Es werden jeweils 101 Inferenzen durchgeführt, davon wird die erste nicht mit in die Daten einbezogen. Bei der ersten Inferenz wird die TPU initialisiert und das Model in die TPU geladen, was zu erheblichen Latenzen führt und für die Ausführgeschwindigkeit somit nicht relevant ist. Über USB wurde die Taktrate der EdgeTPU jeweils auf 500 MHz (via `libedgetpu1-max`) eingestellt.

Der Vergleich in Abbildung 5.2 zeigt, dass die zu erwartende Inferenzzeit auf dem "TinyCar CM4" mittels USB etwa 15% langsamer als die einer mittels USB2 an einen Desktop-Computer angebundenen EdgeTPU ist. Bei der Ausführung auf dem Desktop-Computer wurden erhebliche Schwankungen in der Inferenzzeit festgestellt, die auf dem "TinyCar CM4" deutlich geringer ausfallen.

5.3 Stromaufnahme

Es wird die Stromaufnahme in verschiedenen Betriebsszenarien dokumentiert. Mit den Erkenntnissen kann die Akkulaufzeit besser abgeschätzt werden.

Tabelle 5.1 zeigt die Strom- und Leistungsaufnahe in verschiedenen Betriebsszenarien. Das verwendete Compute-Modul ist ein Modell mit 2 GB RAM, 16 GB eMMC-Speicher und WiFi. Die Eingangsspannung beträgt jeweils 4,0 V.

5.4 Temperaturentwicklung

Die Temperaturentwicklung ist für einen stabilen Betrieb wichtig und muss daher charakterisiert werden. Für die Simulation von CPU-Last wird das Tool `stress` verwendet,

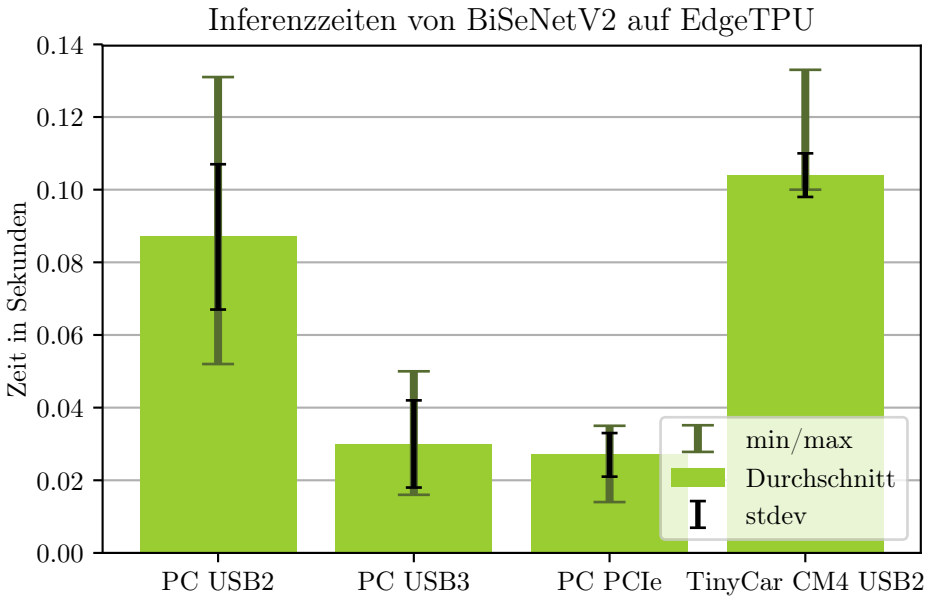


Abbildung 5.2: Vergleich der TPU-Performance verschiedener Schnittstellen zwischen dem "TinyCar CM4" und einem PC

Quelle: Markus Kasten

welches mit der Option `-c 4` vier Threads startet, die `sqrt()` in einer Schleife ausführen. Die Temperatur wird mit `vcgencmd measure_temp` ermittelt. `stress` wurde beim Erreichen einer konstanten Temperatur beendet.

In Abbildung 5.3 wird die Temperaturentwicklung mit verschiedenen Kühloptionen aufgezeichnet: Ohne einen Kühlkörper auf dem Compute-Module 4, mit einem aus Aluminium gefrästen Kühlkörper, sowie mit dem selben Kühlkörper nachdem er eloxiert wurde. Bei einer CPU-Temperatur von etwa 82 °C setzt das Thermal-Throtteling ein, die CPU reduziert also ihre Taktrate um die Temperatur unter 85 °C zu halten.

Szenario	Strom	Leistung
Leerlauf, WiFi verbunden	620 mA	2,48 W
CPU auf 100% (`stress -c 4`)	900 mA	3,6 W
Coral TF-Lite Classification Beispiel mit MobileNetV2 auf USB TPU in Dauerschleife	800 mA	3,2 W
ROS Nodes mit Kamera, Image Segmentation (USB TPU), Übertragung der Bilder auf externen PC	1,2 A	4,8 W
Abgeschaltet (RTC und Ruheströme)	4 µA	16 µW

Tabelle 5.1: Stromaufnahme des "TinyCar CM4" in verschiedenen Betriebsszenarien bei 4 V Eingangsspannung

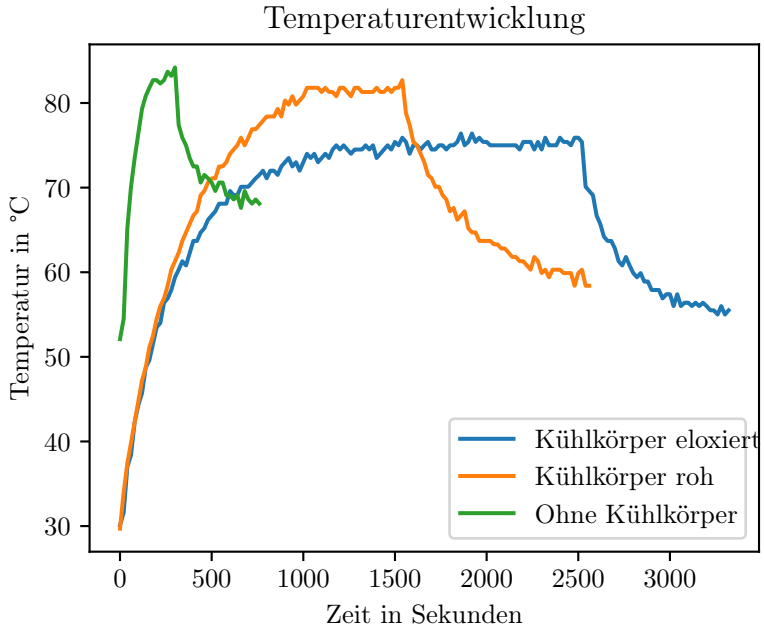


Abbildung 5.3: Temperaturentwicklung der CPU unter `stress -c 4`, Umgebungstemperatur 21 °C

Quelle: Markus Kasten

Der Test zeigt, dass ein Betrieb ohne Kühlkörper sehr schnell zum Thermal Throtteling führt und daher zwingend ein Kühlkörper notwendig ist. Im weiteren Vergleich zwischen dem rohen und eloxierten Kühlkörper bestätigt sich der bis zu etwa 10-Fache Emissionsgrad von eloxiertem Aluminium[14], hier konnte eine weitere Reduktion der Temperatur um etwa 10 K erreicht werden.

Die Trägerplatine wurde weiterhin mit einer Wärmebildkamera analysiert um eventuelle Hotspots zu finden. Wie in Abbildung 5.4 zu sehen wird im Leerlauf lediglich der Spannungswandler für die 5V-Schiene wärmer. Die in der Abbildung als kalt dargestellten TPUs reflektieren stark, daher kann dessen Temperatur mit einer Wärmebildkamera nicht einfach ermittelt werden.

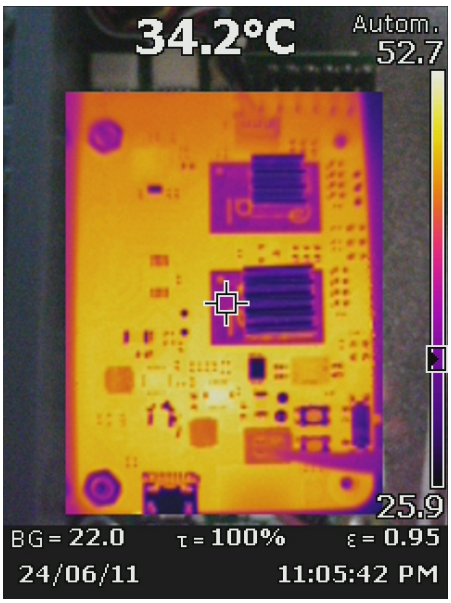


Abbildung 5.4: Aufnahme der Trägerplatine mit einer Wärmebildkamera im Leerlauf

Quelle: Markus Kasten

6 Fazit

Im Rahmen der Arbeit wurden verschiedene Konzepte für autonome Straßenfahrzeuge im Maßstab 1:87 vorgestellt, auf dessen Basis dann zwei aufeinander aufbauende Fahrzeuge umgesetzt wurden.

Zunächst wurde eine einfache Plattform basierend auf einem Raspberry Pi Zero W mit einem eigenem uHAT entwickelt. Zwar konnte diese Plattform nicht die Ressourcenanforderungen erfüllen, die für ROS-basierte Miniaturfahrzeuge benötigt werden, allerdings konnten wichtige Erkenntnisse für eine Weiterentwicklung gesammelt werden. In Zukunft kann dieses Fahrzeug jedoch trotzdem für einfache, weniger ressourcenintensive Anwendungen wie das Aufzeichnen von Daten oder End-to-End Verfahren genutzt werden.

Der Umstieg auf das Raspberry Pi Compute Module 4 hat komplett neue Möglichkeiten gegenüber dem Zero W ermöglicht. Die moderne SoC-Architektur bietet mit seinen vier Cortex-A72 Kernen ein Vielfaches der Rechenleistung des Zero W, was hilft den Ressourcenbedarf von ROS zu decken und in Zukunft ein deutlich größeres Spektrum an Anwendungen erlaubt. Neue Schnittstellen wie PCIe erlauben die schnelle Anbindung von Machine-Learning Beschleunigern und mehr Flexibilität bei der Anbindung von Peripherie.

Mit der eigenen Trägerplatine kann das Modul auf dem Fahrzeug untergebracht werden. Durch die sehr anwendungsspezifische Auslegung konnte die Platine sehr kompakt ausfallen und stellt genau die Peripherie bereit, die auf dem Fahrzeug benötigt wird. Eine standardisierte Erweiterungsschnittstelle erlaubt es trotzdem, weitere Module wie beispielsweise ein UWB-Radio für Lokalisierung in das System einzubinden. Desweiteren erlauben gleich zwei Machine-Learning Beschleuniger von Google die effiziente, parallele Ausführung mehrerer neuronaler Netze, wie sie für viele Anwendungen im Bereich des autonomen Fahrens benötigt werden.

Die Fahrzeugplattform fällt für 1:87 zwar vergleichsweise groß aus, integriert sich jedoch trotzdem gut in eine Welt in dem Maßstab. Trotz des hohen Gewichts durch Akku und

Kühlkörper ist eine sichere und hinreichend präzise Steuerung des Fahrzeugs möglich. Die sehr genaue Odometrie an der Hinterachse hat sich bewährt und erlaubt eine gute Abschätzung der Fahrstrecke.

Die Lokalisierung mit einer Overhead-Kamera oder UWB wurde im Rahmen dieser Arbeit nicht näher untersucht. Eine ausführliche Analyse wird in einer zukünftigen Arbeit durchgeführt.

Das gesetzte Ziel der Arbeit konnte somit erreicht werden. Die Entwicklung des "TinyCar CM4" ist jedoch noch nicht abgeschlossen und bietet noch Potential für zukünftige Verbesserungen.

6.1 Ausblick

Zu den nächsten Schritten gehört die Implementation des autonomen Fahrens auf dem "TinyCar CM4". Luk Schwalb setzt in seiner Arbeit bereits eine Grundlage, auf dessen Basis nun weitere Arbeiten aufbauen können.

Das mechanische Design des Fahrzeugs kann noch weiter verbessert werden. Die verwendete Lenkung und der verwendete Antrieb haben sich zwar bewährt, könnten jedoch noch robuster ausfallen. Eine Karosserie, die die Elektronik versteckt und das Fahrzeug wie ein Bus aussehen lässt wäre besonders bei Tests mit mehreren Fahrzeugen wünschenswert.

Seitens der Elektronik müssen in einer zukünftigen Revision die Fehler aus Anhang A behoben werden. Durch die Verwendung eines Mikrocontrollers gegenüber diskreter Logik beispielsweise für die Steuerung von Leistungsschaltern oder Multiplexern würden sich solche Fehler auch in Software beheben lassen.

Die zwei Kamera-Schnittstellen auf der Trägerplatine würden auch die Verwendung von Stereo-Kameras zur Tiefenwahrnehmung ermöglichen. Alternativ kann mit zwei verschiedenen Brennweiten gearbeitet werden, um ein weites Sichtfeld für die Erfassung der unmittelbaren Umgebung und ein nahes Sichtfeld zum Erkennen von Verkehrsschildern zu haben.

Sollte in Zukunft mehr Rechenleistung für die Inferenz neuronaler Netze benötigt werden lassen sich weitere Coral EdgeTPUs auf dem Fahrzeug unterbringen. Auch der Einsatz von FPGAs kann für sehr spezialisierte Anwendungen erneut evaluiert werden. Eine

Trägerplatine für ein FPGA-SoM kann auf Basis des “TinyCar CM4” mit überschaubarem Aufwand entwickelt werden.

Bevor neue Konzepte umgesetzt werden sollte jedoch das Potential des “TinyCar CM4” vollkommen ausgeschöpft werden. Dafür ist besonders softwareseitig noch viel zu tun.

Literaturverzeichnis

[1] Coral, Google: *Edge TPU performance benchmarks*. https://coral.ai/docs/edgetpu/benchmarks/. – Abgerufen: 17.02.2021

[2] Digilent Inc. (Veranst.): *Digilent Pmod™ Interface Specification*. 1.2.0. 10 2017. – URL https://reference.digilentinc.com/_media/reference/pmod/pmod-interface-specification-1_2_0.pdf

[3] Fuhrmann, Jonas: *Implementierung einer Tensor Processing Unit mit dem Fokus auf Embedded Systems und das Internet of Things*. 2018. – URL https://reposit.haw-hamburg.de/handle/20.500.12738/8527

[4] Google LLC. (Veranst.): *Coral Accelerator Module datasheet*. 1.3. 9 2020. – URL https://coral.ai/static/files/Coral-Accelerator-Module-datasheet.pdf

[5] GPS.gov: *GPS Accuracy*. https://www.gps.gov/systems/gps/performance/accuracy/. – Abgerufen: 05.02.2021

[6] Inc., Xilinx: *Vitis AI Model Zoo*. https://github.com/Xilinx/Vitis-AI/tree/master/models/AI-Model-Zoo#performance-on-ultra96. – Abgerufen: 17.02.2021

[7] Infineon Technologies AG (Veranst.): *3D magnetic sensor for out of shaft position detection*. v1.0. 07 2018. – URL https://www.infineon.com/dgdl/Infineon-Out_of_Shaft-AN-v01_00-EN.pdf?fileId=5546d46265257de801653898ba536074

[8] Ltd., Raspberry Pi T.: *Add-on Boards and HATs*. https://github.com/raspberrypi/hats. – Abgerufen: 17.02.2021

[9] Paulsen, Sebastian: *Entwicklung und Evaluierung kompakter Hardware zur Realisierung künstlicher neuronaler Netze im Bereich autonomer Fahrzeuge*. 2019. – URL https://reposit.haw-hamburg.de/handle/20.500.12738/9160

[10] Raspberry Pi Trading Ltd. (Veranst.): *Compute Module 4 datasheet*. 11 2020. – URL https://datasheets.raspberrypi.org/cm4/cm4-datasheet.pdf

[11] Schwalb, Luk: *ROS-Architektur mit Autonomiefunktionen für ein Miniaturfahrzeug*. 2021

[12] Schönherr, Nils: *Kamera-basierte Minimalautonomie*. 2019. – URL https://reposit.haw-hamburg.de/handle/20.500.12738/8823

[13] Siciliano, Bruno ; Khatib, Oussama: *Springer Handbook of Robotics*. Berlin, Heidelberg : Springer-Verlag, 2007. – ISBN 354023957X

[14] Toolbox, Engineering: *Emissivity Coefficient Materials*. https://www.engineeringtoolbox.com/emissivity-coefficients-d_447.html. – Abgerufen: 17.02.2021

[15] Xilinx Inc. (Veranst.): *Zynq DPU v3.3*. 3.3. 2 2021. – URL https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_3/pg338-dpu.pdf

[16] Yu, Changqian ; Gao, Changxin ; Wang, Jingbo ; Yu, Gang ; Shen, Chunhua ; Sang, Nong: *BiSeNet V2: Bilateral Network with Guided Aggregation for Real-time Semantic Segmentation*. 2020

A Anhang: Errata

In diesem Kapitel werden zum Zeitpunkt des Schreibens bekannte Fehler und eventuell mögliche Workarounds dokumentiert.

A.1 "TinyCar CM4" Revision 1.0

A.1.1 USB Controller schaltet nicht auf Host-Mode

Problem Bei Umschaltung auf internes USB bleibt der USB-Controller im Device-Mode.

Ursache ID-Pin wird nicht Low gezogen.

Workaround Boot-Config `dtoverlay=dwc2,dr_mode=host` zwingt den USB-Controller dauerhaft in Host-Mode.

A.1.2 Ladeelektronik aktiv ohne USB Verbindung

Problem Resultierend aus dem Workaround für A.1.1 wird der USB-Port mit 5V versorgt wenn kein Gerät angeschlossen ist. Gleichzeitig ist die Ladeelektronik aktiv, und lädt den Akku daher mit der eigenen Betriebsspannung.

Ursache Da der ID-Pin des Micro-USB Ports High ist wird die Ladelektronik aktiviert. Der Enable Eingang dieser ist direkt mit dem ID-Pin verbunden.

Workaround EN-Verbindung zu Charger-Switch trennen. Somit ist allerdings kein Laden des Akkus mehr möglich.

A.1.3 Ladeelektronik funktioniert nicht nach Shutdown

Problem Die Ladelektronik funktioniert nur, solange das CM4 läuft.

Ursache Der Charger-Switch hängt an dem 3,3 V Ausgang des CM4, welcher nur aktiv ist, wenn der SoC läuft.

Workaround Keiner

A.1.4 Reset der TPU

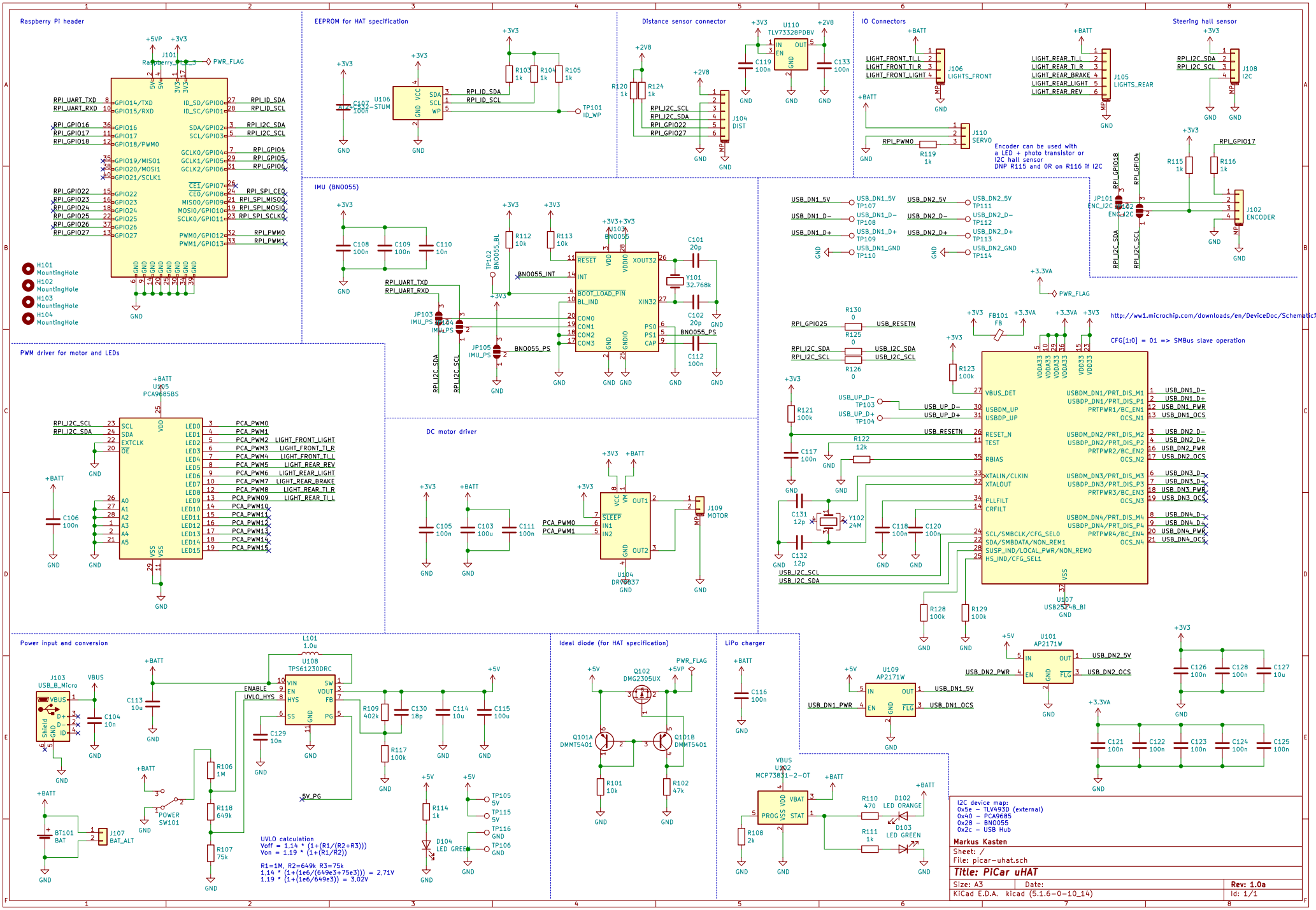
Problem Die (USB-) TPU macht im Betrieb mit maximaler Frequenz (`libedgetpu1-max`) einen Reset. Dies ist insbesondere unterhalb einer Akkuspannung von ca. 3,8 V zu beobachten.

Ursache Die TPU benötigt sehr hohe Stoßströme im Bereich von bis zu 3 A bei der Inferenz. Die Spannungsversorgung sackt dabei so stark ab, dass der Supervisor die TPU in den Reset zieht.

Workaround Erste Möglichkeit: Kapazität auf der +3V3P Rail stark erhöhen. Bei einer Kapazität von 1200 µF konnte die Anfälligkeit für den Reset deutlich verringert werden, dennoch kann nicht der komplette Eingangsspannungsbereich von 3,0 V bis 4,2 V stabil abgedeckt werden. Desweiteren sollte der Wert des Feed-Forward Kondensators des +3V3P Spannungswandlers auf 67 pF erhöht werden.

Zweite Möglichkeit: Die TPU mit halber Frequenz betreiben. Dafür wird `libedgetpu1-std` installiert.

B Anhang: Schaltplan TinyCar Zero uHAT


Raspberry Pi header
EEPROM for HAT specification
Distance sensor connector
IO Connectors
Steering hall sensor
IMU (BNO055)
PWM driver for motor and LEDs
DC motor driver
Power input and conversion
Ideal diode (for HAT specification)
LiPo charger
Encoder can be used with a LED + photo transistor or I2C hall sensor DNP R115 and OR on R116 if I2C
http://ww1.microchip.com/downloads/en/DeviceDoc/Schematic%20Chec
CFG[1:0] = 01 => SMBus slave operation
UVLO calculation
Voff = 1.14 * (1+(R1/(R2+R3)))
Von = 1.19 * (1+(R1/R2))
R1=1M, R2=649k R3=75k
1.14 * (1+(1e6/(649e3+75e3))) = 2.71V
1.19 * (1+(1e6/649e3)) = 3.02V
I2C device map:
0x5e – TLV493D (external)
0x40 – PCA9685
0x28 – BNO055
0x2c – USB Hub
Markus Kasten
Sheet: /
File: picar-uhat.sch
Title: PiCar uHAT
Size: A3
Date:
Rev: 1.0a
KiCad E.D.A. kicad (5.1.6-0-10_14)
Id: 1/1

C Anhang: Schaltplan TinyCar CM4

Power

Power supplies

Lilon Charger

File: Power.kicad_sch

Coral

2x Coral TPU

File: Coral.kicad_sch

CM4

CM4 General Purpose IO

IO assignments

File: CM4.kicad_sch

CM4-HighSpeed

CM4 High Speed IO

Camera

File: HighSpeed.kicad_sch

IO

IMU

Motor driver

RTC

Connectors

File: IO.kicad_sch

USB

USB Mux

USB power switches

File: USB.kicad_sch

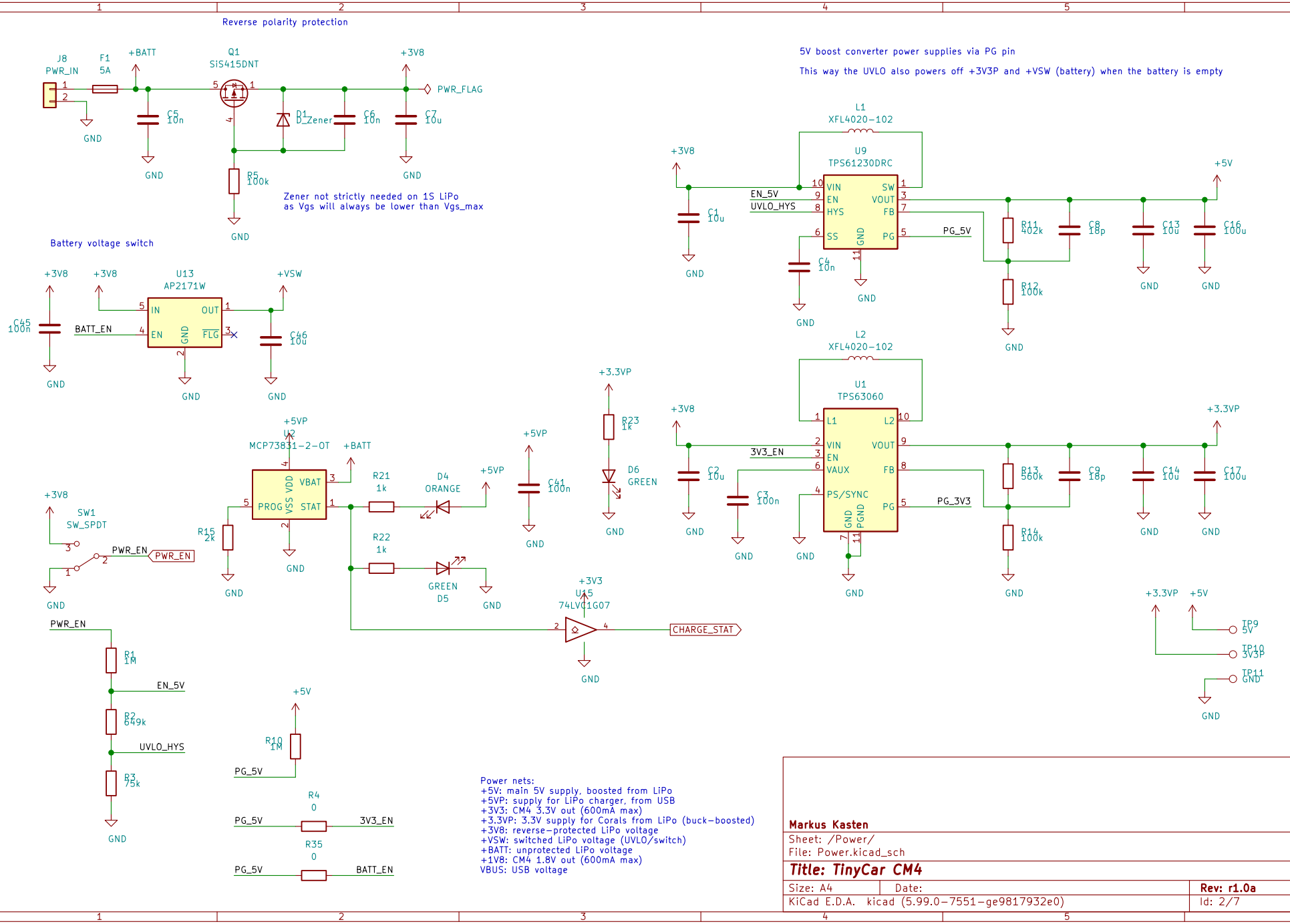
Markus Kasten

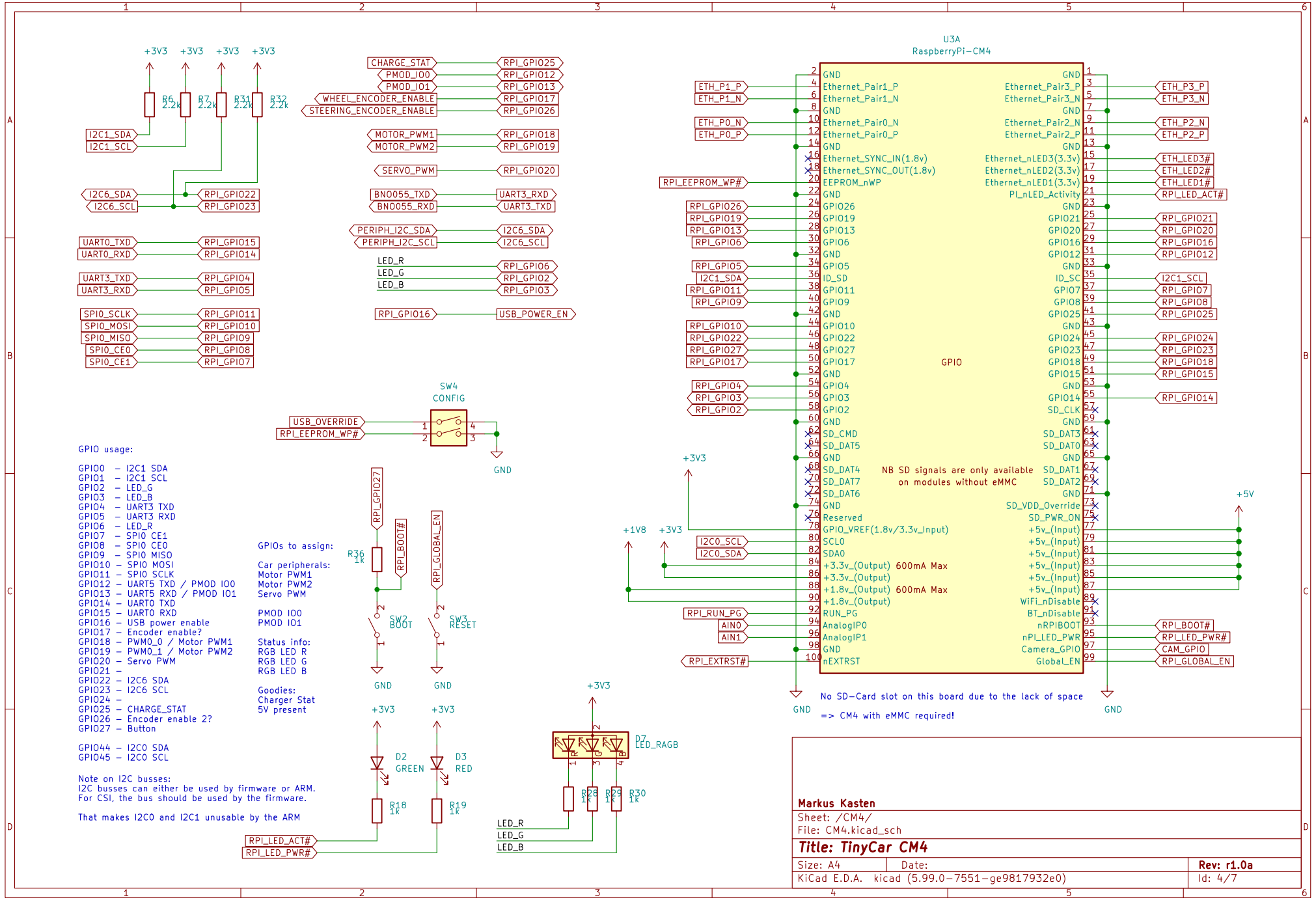
Sheet: /

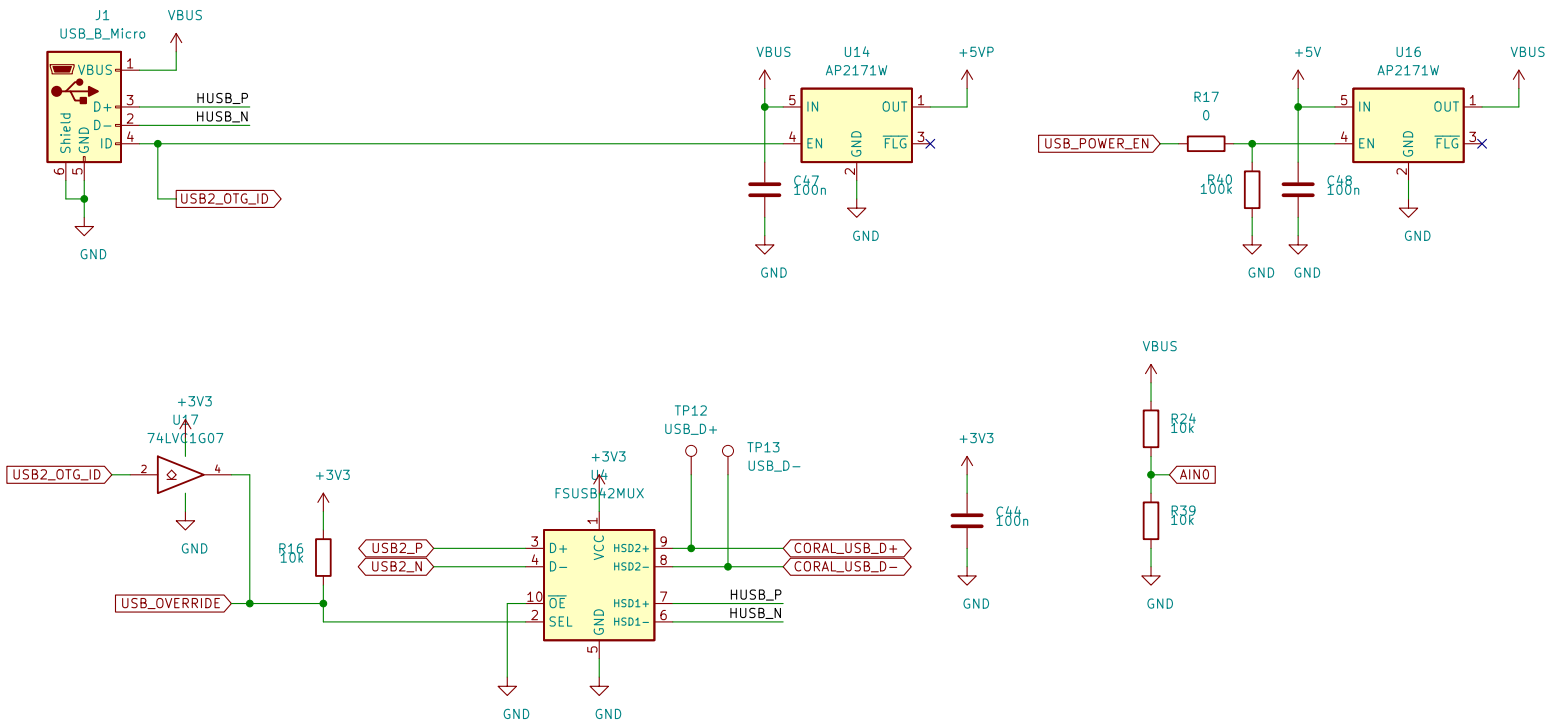
File: CM4-Board.kicad_sch

Title: TinyCar CM4

Size: A4	Date:	**Rev: r1.0a**
KiCad E.D.A. kicad (5.99.0-7551-ge9817932e0)		Id: 1/7


Reverse polarity protection
J8
PWR_IN
F1
5A
+BATT
Q1
SiS415DNT
+3V8
PWR_FLAG
C5
10n
D1
D_Zener
C6
10n
C7
10u
R5
100k
GND
Zener not strictly needed on 1S LiPo
as Vgs will always be lower than Vgs_max
Battery voltage switch
U13
AP2171W
+3V8
+VSW
C45
100n
BATT_EN
C46
10u
5V boost converter power supplies via PG pin
This way the UVLO also powers off +3V3P and +VSW (battery) when the battery is empty
L1
XFL4020-102
U9
TPS61230DRC
EN_5V
UVLO_HYS
PG_5V
C1
10u
C4
10n
R11
402k
R12
100k
C8
18p
C13
10u
C16
100u
+5V
L2
XFL4020-102
U1
TPS63060
3V3_EN
C2
10u
C3
100n
R13
560k
R14
100k
C9
18p
C14
10u
C17
100u
PG_3V3
+3.3VP
U2
MCP73831-2-OT
+5VP
R15
2k
R21
1k
R22
1k
D4
ORANGE
GREEN
D5
C41
100n
R23
1k
D6
GREEN
SW1
SW_SPDT
PWR_EN
U15
74LVC1G07
+3V3
CHARGE_STAT
R1
1M
R2
649k
R3
75k
R10
1M
R4
0
R35
0
3V3_EN
BATT_EN
IP9
5V
IP10
3V3P
IP11
GND
Power nets:
+5V: main 5V supply, boosted from LiPo
+5VP: supply for LiPo charger, from USB
+3V3: CM4 3.3V out (600mA max)
+3.3VP: 3.3V supply for Corals from LiPo (buck-boosted)
+3V8: reverse-protected LiPo voltage
+VSW: switched LiPo voltage (UVLO/switch)
+BATT: unprotected LiPo voltage
+1V8: CM4 1.8V out (600mA max)
VBUS: USB voltage
Markus Kasten
Sheet: /Power/
File: Power.kicad_sch
Title: TinyCar CM4
Size: A4
Date:
Rev: r1.0a
KiCad E.D.A. kicad (5.99.0-7551-ge9817932e0)
Id: 2/7


+3V3
R6 2.2k
R7 2.2k
R31 2.2k
R32 2.2k
I2C1_SDA
I2C1_SCL
I2C6_SDA
I2C6_SCL
RPI_GPIO22
RPI_GPIO23
UART0_TXD
UART0_RXD
RPI_GPIO15
RPI_GPIO14
UART3_TXD
UART3_RXD
RPI_GPIO4
RPI_GPIO5
SPI0_SCLK
SPI0_MOSI
SPI0_MISO
SPI0_CE0
SPI0_CE1
RPI_GPIO11
RPI_GPIO10
RPI_GPIO9
RPI_GPIO8
RPI_GPIO7
CHARGE_STAT
PMOD_IO0
PMOD_IO1
WHEEL_ENCODER_ENABLE
STEERING_ENCODER_ENABLE
RPI_GPIO25
RPI_GPIO12
RPI_GPIO13
RPI_GPIO17
RPI_GPIO26
MOTOR_PWM1
MOTOR_PWM2
RPI_GPIO18
RPI_GPIO19
SERVO_PWM
RPI_GPIO20
BNO055_TXD
BNO055_RXD
UART3_RXD
UART3_TXD
PERIPH_I2C_SDA
PERIPH_I2C_SCL
I2C6_SDA
I2C6_SCL
LED_R
LED_G
LED_B
RPI_GPIO6
RPI_GPIO2
RPI_GPIO3
RPI_GPIO16
USB_POWER_EN
SW4
CONFIG
USB_OVERRIDE
RPI_EEPROM_WP#
GND
GPIO usage:
GPIO0 – I2C1 SDA
GPIO1 – I2C1 SCL
GPIO2 – LED_G
GPIO3 – LED_B
GPIO4 – UART3 TXD
GPIO5 – UART3 RXD
GPIO6 – LED_R
GPIO7 – SPI0 CE1
GPIO8 – SPI0 CE0
GPIO9 – SPI0 MISO
GPIO10 – SPI0 MOSI
GPIO11 – SPI0 SCLK
GPIO12 – UART5 TXD / PMOD IO0
GPIO13 – UART5 RXD / PMOD IO1
GPIO14 – UART0 TXD
GPIO15 – UART0 RXD
GPIO16 – USB power enable
GPIO17 – Encoder enable?
GPIO18 – PWM0_0 / Motor PWM1
GPIO19 – PWM0_1 / Motor PWM2
GPIO20 – Servo PWM
GPIO21 –
GPIO22 – I2C6 SDA
GPIO23 – I2C6 SCL
GPIO24 –
GPIO25 – CHARGE_STAT
GPIO26 – Encoder enable 2?
GPIO27 – Button
GPIO44 – I2C0 SDA
GPIO45 – I2C0 SCL
Note on I2C busses:
I2C busses can either be used by firmware or ARM.
For CSI, the bus should be used by the firmware.
That makes I2C0 and I2C1 unusable by the ARM
GPIOs to assign:
Car peripherals:
Motor PWM1
Motor PWM2
Servo PWM
PMOD IO0
PMOD IO1
Status info:
RGB LED R
RGB LED G
RGB LED B
Goodies:
Charger Stat
5V present
RPI_GPIO27
R36 1k
RPI_BOOT#
RPI_GLOBAL_EN
SW2 BOOT
SW3 RESET
GND
GND
+3V3
+3V3
D2 GREEN
D3 RED
R18 1k
R19 1k
RPI_LED_ACT#
RPI_LED_PWR#
+3V3
D7 LED_RAGB
R28 1k
R29 1k
R30 1k
LED_R
LED_G
LED_B
U3A
RaspberryPi-CM4
GND
Ethernet_Pair1_P
Ethernet_Pair1_N
Ethernet_Pair0_N
Ethernet_Pair0_P
Ethernet_SYNC_IN(1.8v)
Ethernet_SYNC_OUT(1.8v)
EEPROM_nWP
GPIO26
GPIO19
GPIO13
GPIO6
GPIO5
ID_SD
GPIO11
GPIO9
GPIO10
GPIO22
GPIO27
GPIO17
GPIO4
GPIO3
GPIO2
SD_CMD
SD_DAT5
SD_DAT4
SD_DAT7
SD_DAT6
Reserved
GPIO_VREF(1.8v/3.3v_Input)
SCL0
SDA0
+3.3v_(Output) 600mA Max
+3.3v_(Output)
+1.8v_(Output) 600mA Max
+1.8v_(Output)
RUN_PG
AnalogIP0
AnalogIP1
nEXTRST
Ethernet_Pair3_P
Ethernet_Pair3_N
Ethernet_Pair2_N
Ethernet_Pair2_P
Ethernet_nLED3(3.3v)
Ethernet_nLED2(3.3v)
Ethernet_nLED1(3.3v)
PI_nLED_Activity
GPIO21
GPIO20
GPIO16
GPIO12
ID_SC
GPIO7
GPIO8
GPIO25
GPIO24
GPIO23
GPIO18
GPIO15
SD_CLK
SD_DAT3
SD_DAT0
SD_DAT1
SD_DAT2
SD_VDD_Override
SD_PWR_ON
+5v_(Input)
WiFi_nDisable
BT_nDisable
nRPIBOOT
nPI_LED_PWR
Camera_GPIO
Global_EN
GPIO
NB SD signals are only available on modules without eMMC
ETH_P1_P
ETH_P1_N
ETH_P0_N
ETH_P0_P
RPI_EEPROM_WP#
RPI_GPIO26
RPI_GPIO19
RPI_GPIO13
RPI_GPIO6
RPI_GPIO5
I2C1_SDA
RPI_GPIO11
RPI_GPIO9
RPI_GPIO10
RPI_GPIO22
RPI_GPIO27
RPI_GPIO17
RPI_GPIO4
RPI_GPIO3
RPI_GPIO2
+3V3
+1V8
+3V3
I2C0_SCL
I2C0_SDA
RPI_RUN_PG
AIN0
AIN1
RPI_EXTRST#
GND
+5V
ETH_P3_P
ETH_P3_N
ETH_P2_N
ETH_P2_P
ETH_LED3#
ETH_LED2#
ETH_LED1#
RPI_LED_ACT#
RPI_GPIO21
RPI_GPIO20
RPI_GPIO16
RPI_GPIO12
I2C1_SCL
RPI_GPIO7
RPI_GPIO8
RPI_GPIO25
RPI_GPIO24
RPI_GPIO23
RPI_GPIO18
RPI_GPIO15
RPI_GPIO14
RPI_BOOT#
RPI_LED_PWR#
CAM_GPIO
RPI_GLOBAL_EN
GND
GND
No SD-Card slot on this board due to the lack of space
=> CM4 with eMMC required!
Markus Kasten
Sheet: /CM4/
File: CM4.kicad_sch
Title: TinyCar CM4
Size: A4
Date:
Rev: r1.0a
KiCad E.D.A. kicad (5.99.0-7551-ge9817932e0)
Id: 4/7



USB_OVERRIDE gets pulled low, so it doesn't disconnect the internal USB

USB	Override	OTG	VBUS switch	5VP switch	Mux sel
Host	off	0	on	off	1
Host	on	0	on	off	1
Device	off	1	off	on	2
Device	on	1	off	on	1

USB mode defaults to device when no plug is inserted since OTG is pulled up.

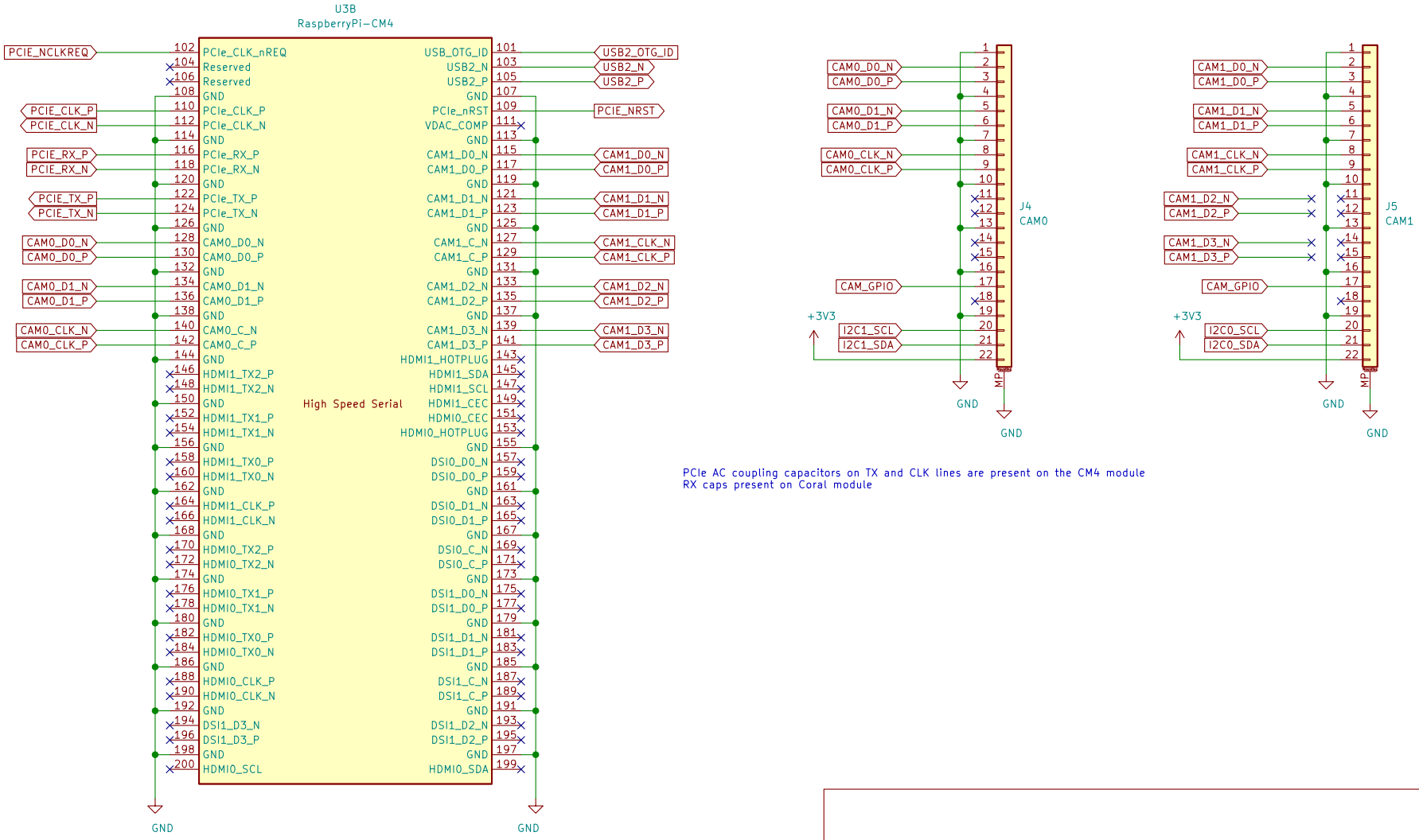
USB_OVERRIDE forces the mux to use the micro USB

Markus Kasten

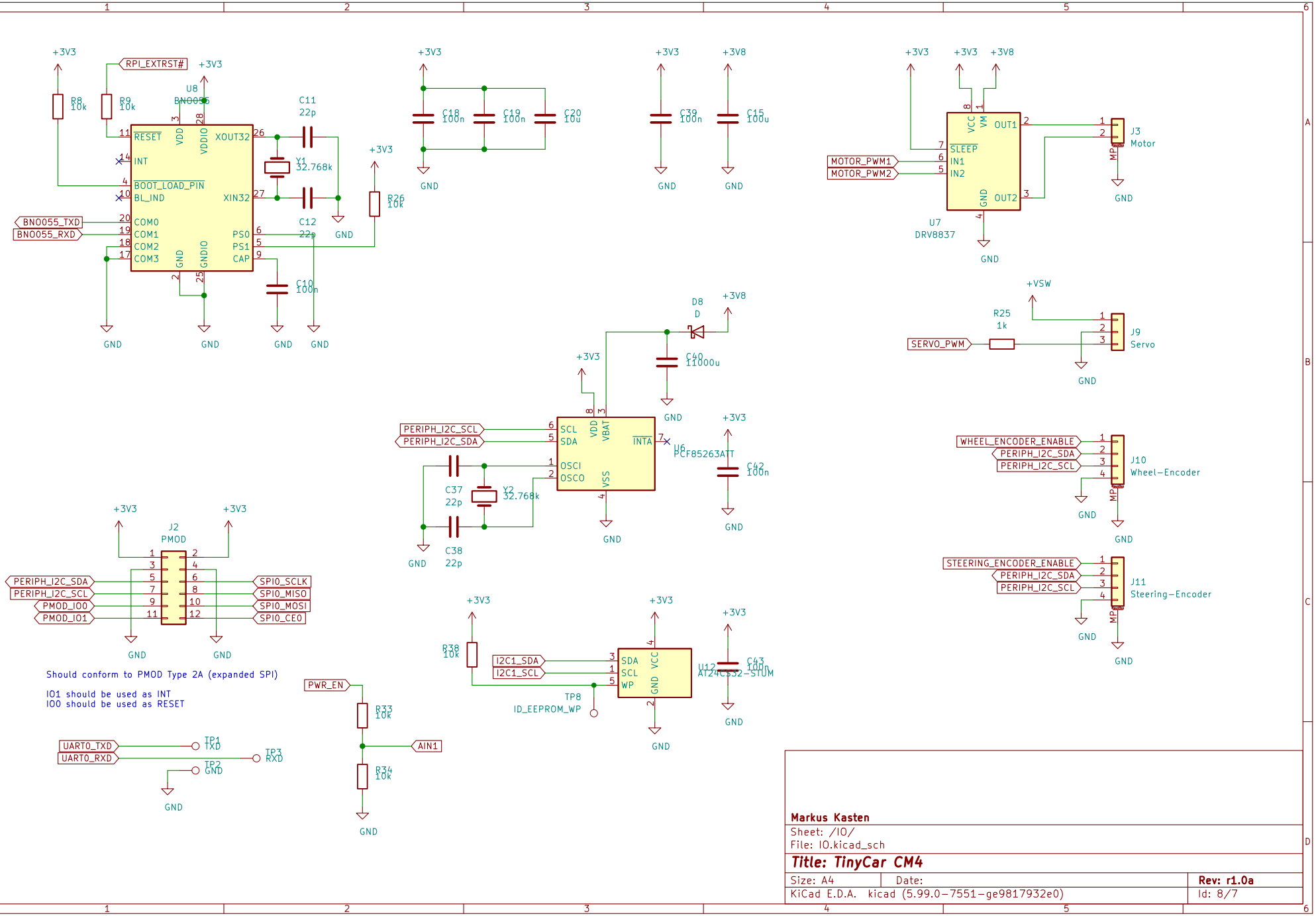
Sheet: /USB/
File: USB.kicad_sch

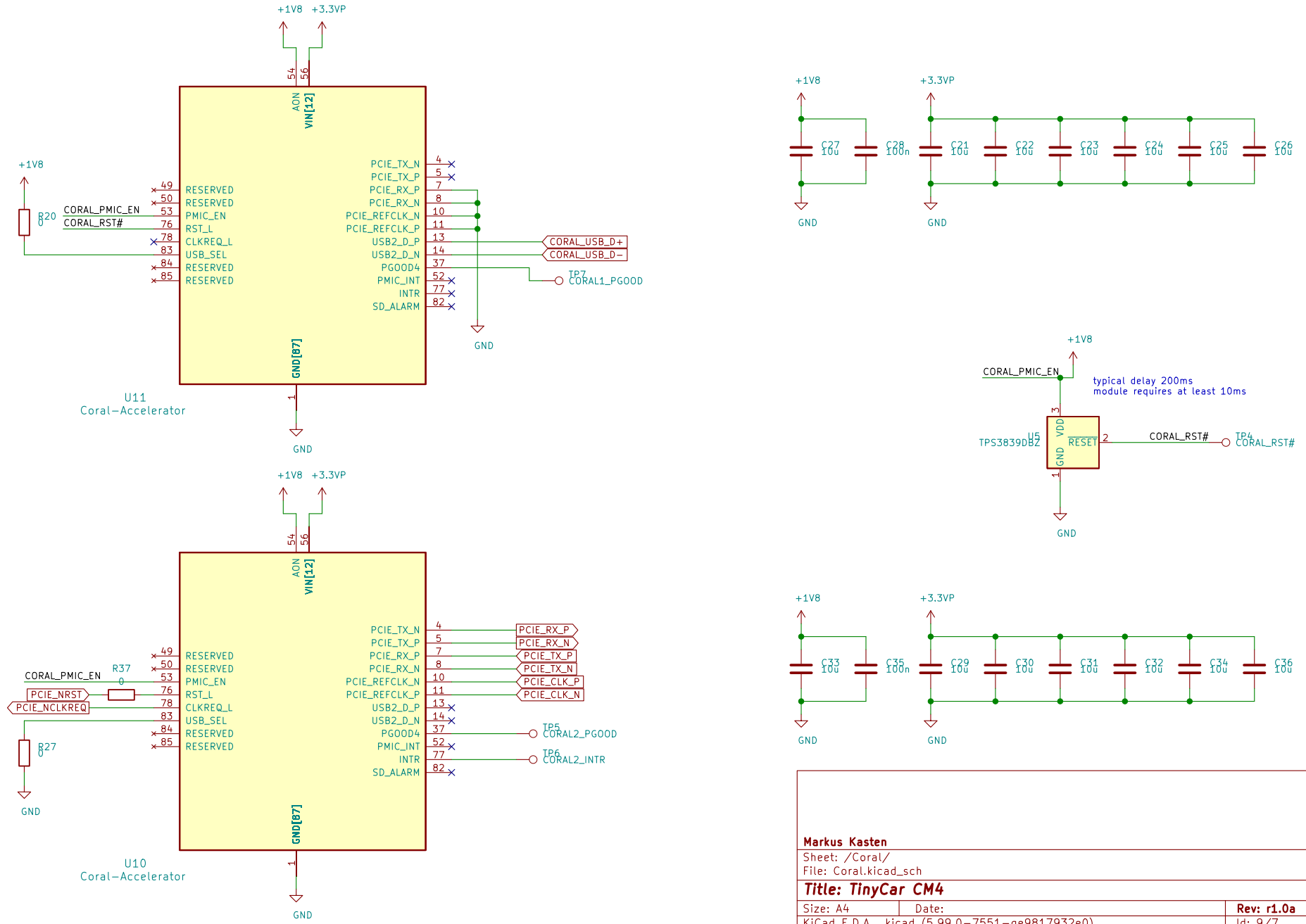
Title: TinyCar CM4

Size: A4	Date:	**Rev: r1.0a**
KiCad E.D.A. kicad (5.99.0-7551-ge9817932e0)		Id: 5/7

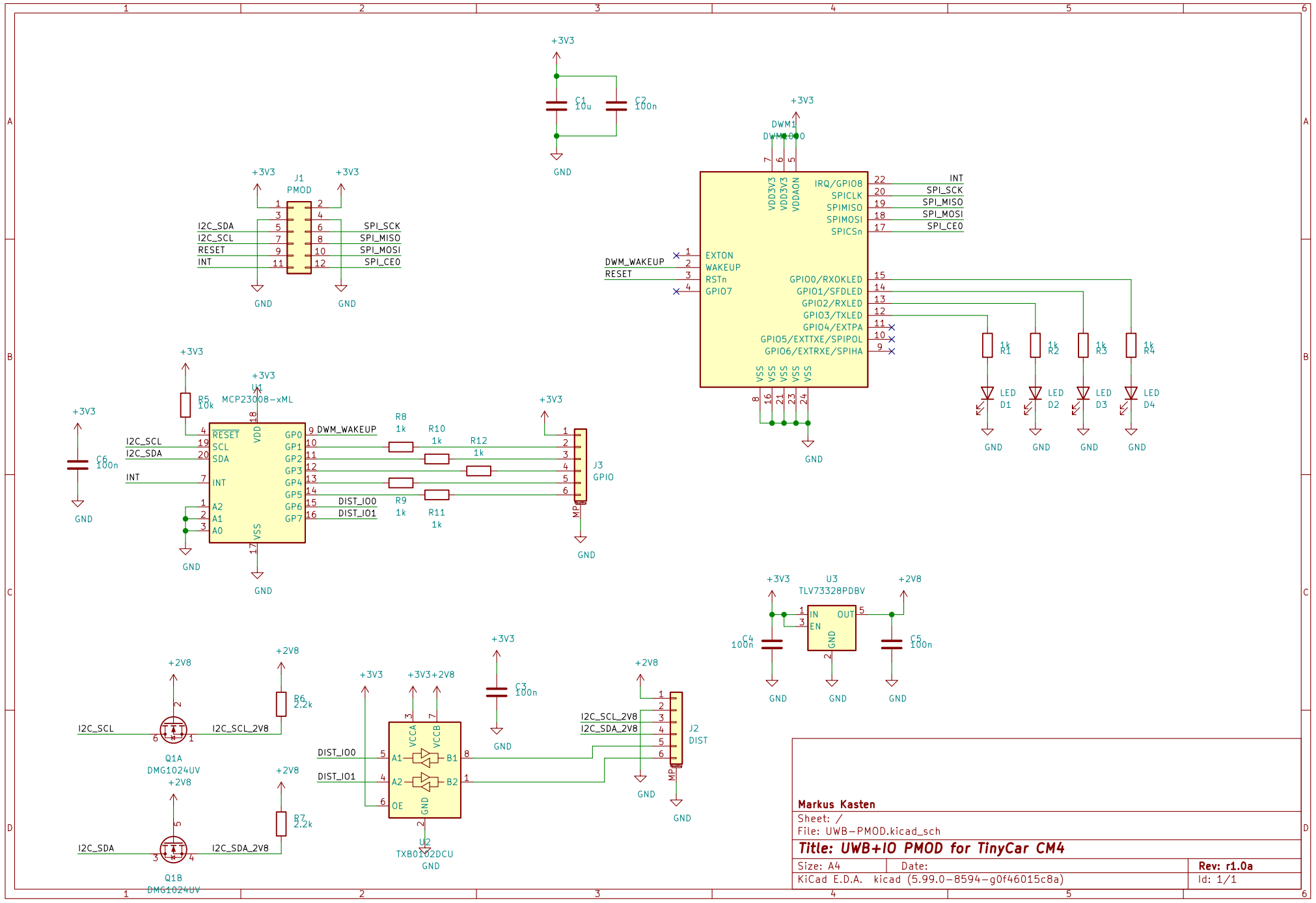


Markus Kasten	
Sheet: /CM4-HighSpeed/	
File: HighSpeed.kicad_sch	
Title: TinyCar CM4	
Size: A4	Date:
KiCad E.D.A. kicad (5.99.0-7551-ge9817932e0)	**Rev: r1.0a**
	Id: 8/7


+3V3
RPI_EXTRST#
R8
10k
R9
10k
U8
BNO055
RESET
VDD
VDDIO
XOUT32
INT
BOOT_LOAD_PIN
BL_IND
XIN32
COM0
COM1
COM2
COM3
GND
GNDIO
PS0
PS1
CAP
BNO055_TXD
BNO055_RXD
C11
22p
Y1
32.768k
C12
22p
C10
100n
R26
10k
C18
100n
C19
100n
C20
10u
C39
100n
C15
100u
+3V8
VCC
VM
OUT1
OUT2
SLEEP
IN1
IN2
MOTOR_PWM1
MOTOR_PWM2
U7
DRV8837
J3
Motor
MP
D8
D
C40
11000u
+VSW
R25
1k
SERVO_PWM
J9
Servo
PERIPH_I2C_SCL
PERIPH_I2C_SDA
SCL
SDA
VBAT
INTA
OSCI
OSCO
VSS
U6
PCF85263ATT
C37
22p
Y2
32.768k
C38
22p
C42
100n
WHEEL_ENCODER_ENABLE
J10
Wheel-Encoder
STEERING_ENCODER_ENABLE
J11
Steering-Encoder
J2
PMOD
PMOD_IO0
PMOD_IO1
SPI0_SCLK
SPI0_MISO
SPI0_MOSI
SPI0_CE0
Should conform to PMOD Type 2A (expanded SPI)
IO1 should be used as INT
IO0 should be used as RESET
R38
10k
I2C1_SDA
I2C1_SCL
WP
U12
AT24C32-STUM
C43
100n
TP8
ID_EEPROM_WP
PWR_EN
R33
10k
R34
10k
AIN1
UART0_TXD
UART0_RXD
TP1
TXD
TP2
GND
TP3
RXD
Markus Kasten
Sheet: /IO/
File: IO.kicad_sch
Title: TinyCar CM4
Size: A4
Date:
Rev: r1.0a
KiCad E.D.A. kicad (5.99.0-7551-ge9817932e0)
Id: 8/7


+1V8
+3.3VP
AON
VIN[12]
GND[87]
U11
Coral-Accelerator
R20
0
CORAL_PMIC_EN
CORAL_RST#
RESERVED
PMIC_EN
RST_L
CLKREQ_L
USB_SEL
PCIE_TX_N
PCIE_TX_P
PCIE_RX_P
PCIE_RX_N
PCIE_REFCLK_N
PCIE_REFCLK_P
USB2_D_P
USB2_D_N
PGOOD4
PMIC_INT
INTR
SD_ALARM
CORAL_USB_D+
CORAL_USB_D-
TP7
CORAL1_PGOOD
GND
U10
Coral-Accelerator
R37
R27
PCIE_NRST
PCIE_NCLKREQ
PCIE_RX_P
PCIE_RX_N
PCIE_TX_P
PCIE_TX_N
PCIE_CLK_P
PCIE_CLK_N
TP5
CORAL2_PGOOD
TP6
CORAL2_INTR
C27
10u
C28
100n
C21
C22
C23
C24
C25
C26
C33
C35
C29
C30
C31
C32
C34
C36
U5
TPS3839DBZ
VDD
RESET
typical delay 200ms
module requires at least 10ms
TP4
CORAL_RST#
Markus Kasten
Sheet: /Coral/
File: Coral.kicad_sch
Title: TinyCar CM4
Size: A4
Date:
Rev: r1.0a
KiCad E.D.A. kicad (5.99.0-7551-ge9817932e0)
Id: 9/7

D Anhang: Schaltplan UWB Pmod Board


+3V3
C1
10u
C2
100n
GND
J1
PMOD
I2C_SDA
I2C_SCL
RESET
INT
SPI_SCK
SPI_MISO
SPI_MOSI
SPI_CE0
DWM1
DWM1000
VDD3V3
VDD3V3
VDDAON
IRQ/GPIO8
SPICLK
SPIMISO
SPIMOSI
SPICSn
EXTON
WAKEUP
RSTn
GPIO7
DWM_WAKEUP
RESET
GPIO0/RXOKLED
GPIO1/SFDLED
GPIO2/RXLED
GPIO3/TXLED
GPIO4/EXTPA
GPIO5/EXTTXE/SPIPOL
GPIO6/EXTRXE/SPIHA
VSS
LED
D1
D2
D3
D4
R1
R2
R3
R4
1k
R5
10k
C6
100n
U1
MCP23008-xML
RESET
SCL
SDA
INT
A2
A1
A0
VDD
VSS
GP0
GP1
GP2
GP3
GP4
GP5
GP6
GP7
DWM_WAKEUP
DIST_IO0
DIST_IO1
R8
R9
R10
R11
R12
J3
GPIO
U3
TLV73328PDBV
IN
EN
OUT
GND
C4
100n
C5
100n
+2V8
R6
2.2k
R7
2.2k
Q1A
DMG1024UV
Q1B
DMG1024UV
I2C_SCL
I2C_SDA
I2C_SCL_2V8
I2C_SDA_2V8
C3
100n
U2
TXB0102DCU
VCCA
VCCB
OE
A1
A2
B1
B2
J2
DIST
Markus Kasten
Sheet: /
File: UWB-PMOD.kicad_sch
Title: UWB+IO PMOD for TinyCar CM4
Size: A4
Date:
Rev: r1.0a
KiCad E.D.A. kicad (5.99.0-8594-g0f46015c8a)
Id: 1/1

Glossar

CSI Camera Serial Interface (CSI) ist eine Spezifikation der MIPI Alliance für die Verbindung einer Kamera mit einem Hostsystem. CSI verwendet eine serielle Datenübertragung mit Übertragungsgeschwindigkeiten von bis zu mehreren Gigabit pro Sekunde. Eine CSI-Schnittstelle verfügt in der Regel über zwei oder vier Datenlanes..

CUDA CUDA ist eine Technik von NVIDIA zur Programmierung von Anwendungen, die auf GPUs stark parallelisiert ausgeführt werden können. Sie ist unter anderem im Bereich des Machine-Learnings stark verbreitet..

I^{2}C Clock Stretching Beim I2C Clock-Stretching wird der Master vom Slave dazu aufgefordert, zu warten. Dies geschiet indem der Slave die Clock-Leitung für die benötigte Zeit Low gezogen..

PMIC Ein Power Management IC (PMIC) ist ein integrierter Schaltkreis, der die Spannungsversorgung beispielsweise für eine CPU oder einen FPGA bereitstellt. Er umfasst in der Regel ein oder mehrere Spannungswandler sowie Überwachungsfunktionen..

USB OTG USB On-The-Go. Erlaubt ein Gerät mit Micro-USB Buchse sowohl als Host als auch als Gerät zu agieren. Der jeweils verwendete Modus wird über den ID-Pin des Steckers geschaltet.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Hardwareplattformen für autonome Straßenfahrzeuge im Maßstab 1:87

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

_____	_____	_____
Ort	Datum	Unterschrift im Original