

Studienarbeit  
**Implementation einer Embedded Machine  
Vision Plattform**

im Studiengang Technische Informatik  
der Fakultät Informationstechnik  
Wintersemester 2022/2023

Florian Korotschenko

**Zeitraum:** 21.02.2023  
**Prüfer:** Prof. Dr. Rer. Nat. Markus Enzweiler  
**Zweitprüfer:** Prof. Dr.-Ing. Rainer Keller

# Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 25. Februar 2023

F. Kretschmer  
Unterschrift

# Kurzzusammenfassung

In dieser Studienarbeit wird ein Teil der Arbeit zum Bau einer mobilen Stereo Vision Plattform für Machine-Vision Experimente dokumentiert. Es sollen Motivation, Hardware- und Software-Entscheidungen sowie die Herangehensweise bei der Umsetzung in logischer Reihenfolge dargestellt werden. Auch Erkenntnisse und Vor- sowie Nachteile bei dieser Plattform sollen erörtert werden. Am Ende wartet ein Fazit und Ausblick auf weitere Arbeit an diesem Projekt.

Als solches handelt es sich bei der Idee zu diesem Projekt um eine eigene des Autors. Teile der Arbeit wurden bereits über Monate hinweg verrichtet, daher ist es möglich, dass manche Daten und Vergleiche zum Zeitpunkt der Abgabe dieser Arbeit bereits nicht mehr dem aktuellsten Stand entsprechen. Dies kann besonders bei der Software der Fall sein.

# Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation . . . . .	1
1.2	Auswahl der Hardware . . . . .	1
1.2.1	Single Board Computer . . . . .	2
1.2.2	Neuronaler Beschleuniger . . . . .	4
1.2.3	Kamera-Sensoren . . . . .	6
1.2.4	Zubehör für mobile Verwendung . . . . .	7
1.3	Auswahl der Software . . . . .	8
1.3.1	Betriebssystem . . . . .	8
1.3.2	Python-Bibliotheken . . . . .	9
1.3.3	Treiber und Frameworks . . . . .	10
2	NPU	11
2.1	Hardware . . . . .	11
2.1.1	Anforderung . . . . .	11
2.1.2	Konzeption des PCIe Adapter . . . . .	13
2.1.3	Implementation des PCIe Adapter - Schematic . . . . .	14
2.1.4	Implementation des PCIe Adapter - Layout . . . . .	16
2.1.5	Fehler und Verbesserungspotential . . . . .	20
2.2	Software . . . . .	21
2.2.1	Die Ausgangssituation . . . . .	21
2.2.2	Der Betriebssystem Buildprozess im Detail . . . . .	22
2.2.3	Die Lösung mittels Patches . . . . .	25
2.2.4	Tests . . . . .	27
2.2.5	Verbesserungspotential . . . . .	29
3	Kamera Sektion	30
3.1	Hardware . . . . .	30
3.2	Software . . . . .	30
4	Schluss	32
4.1	Ausblick . . . . .	32
4.2	Fazit . . . . .	32
A	Kapitel im Anhang	34

Literaturverzeichnis

46

# Abbildungsverzeichnis

1.1	Coral M.2 Beschleuniger mit Dual Edge TPU . . . . .	4
1.2	Überblick grober Aufbau der Coral Edge TPU Architektur . . . . .	5
1.3	Raspberry Pi V2 Kameramodul [21] und Vergleichsbild [32] . . . . .	7
1.4	Beispielbild armbian-config und Armbian Logo . . . . .	9
2.1	Überblick der M.2 Formate . . . . .	12
2.2	JLC7628 4-Layer 0,8mm Dicke Layerstackup . . . . .	18
2.3	Foto von bestücktem PCIe Adapter . . . . .	19
2.4	Foto von PCIe Adapter in Testsystem mit Coral Edge Dual TPU Modul .	19
2.5	Vergleich mit Kontaktfingern einer NVMe SSD . . . . .	20
2.6	Emuliertes Reset Signal Logic Analyzer Aufzeichnung . . . . .	27
4.1	Foto des bisherigen Aufbau . . . . .	33
A.1	Schematic für PCIe Adapter Board . . . . .	43
A.2	Layout für PCIe Adapter Board . . . . .	44
A.3	Schematic für Kamera Adapter Board . . . . .	45

# Abkürzungen

*ASIC* Application Specific Integrated Circuit

*AXI* Advanced eXtensible Interface

*CLI* Command-line interface

*CPU* Central Processing Unit

*CSI-2* Camera Serial Interface 2

*eMMC* embedded MultiMedia Card

*ENIG* Electroless nickel immersion gold

*ESR* Exception Syndrome Register

*FLOPS* Floating Point Operations Per Second

*FoV* Field of View

*FPC* Flexible Printed Circuits

*GPIO* General Purpose Input Output

*GPU* Graphics Processing Unit

*GT/s* Giga-transfers per second

*GUI* Graphical User Interface

*HASL* Hot air solder leveling

*I2C* Inter-IC bus

*ISA* Instruction Set Architecture

*ISP* Image Signal Processor

*MAC* Multiply-Accumulate

*NMI* Non-Maskable Interrupt

*NPU* Neural Processing Unit

*NVMe* Nonvolatile Memory Express

*ONNX* Open Neural Network Exchange

---

*PCB* Printed Circuit Board

*PCIe* Peripheral Component Interconnect Express

*PCM* Puls-Code-Modulation

*POR* Power-on-Reset

*RAS* Reliability Availability and Serviceability

*SATA* Serial AT Attachment

*SBC* Single Board Computer

*SDIO* Secure Digital Input Output

*SMD* Surface-mounted device

*SoC* System on Chip

*SRAM* Static random-access memory

*SSD* Solid State Drive

*SSH* Secure Socket Shell

*TOPS* Trillions of Operations Per Second

*UART* Universal Asynchronous Receiver Transmitter

*USB* Universal Serial Bus

*VPU* Video Processing Unit

# Tabellenverzeichnis

1.1 Vergleich von NanoPi M4V2 und Raspberry Pi 4 Model B . . . . .	3
--	---

# Listings

2.1	Verwendete Build-Optionen	23
2.2	lspci mit erkannten Geräten	27
2.3	Coral TPU Pipelining Demo	28
A.1	PCIe Adapter Kernel Panic Auszug	34
A.2	Finaler Armbian Patch	36

# 1 Einleitung

Diese Einleitung soll die Motivation der Studienarbeit und die Auswahl der Hard- und Software erläutern.

## 1.1 Motivation

In der heutigen Zeit sind Machine-Vision und Robotik-Anwendungen vielfältiger denn je. Um auch eigene Projekte in diesem Bereichen durchführen zu können möchte der Autor unter anderem Versuche mit Stereo-Vision-Systemen machen. Diese Systeme wiederum sollen sich möglichst flexibel einsetzen und verbauen lassen und Daten für das Training und die Inferenz neuronaler Netzwerke liefern. Idealerweise, so die Ansicht des Autors, kann man die Kameras, einen neuronalen Beschleuniger und einen minimalen Aufbau darum herum bei Bedarf mitnehmen. Diese reduzierte Variante, welche nur die visuelle Sensorik eines Roboters beinhaltet eigne sich gut um Hardware-Konfigurationen und Algorithmen mit möglichst wenig Aufwand und Komplexität zu testen. Gleichzeitig stellt sie einen Zwischenschritt zum kompletten Aufbau dar und lässt sich unterwegs nutzen um Trainingsdaten zu sammeln.

Konkret wird für diesen Aufbau ein Single Board Computer (**SBC**) mit einer Neural Processing Unit (**NPU**) und zwei Kameras ausgestattet. Der **SBC** kann dann mittels Universal Serial Bus (**USB**) oder Ethernet bzw. Wi-Fi mit einem Laptop oder selbst Tablet Computer verbunden werden und mit Serial, Secure Socket Shell (**SSH**) oder Remote Desktop gesteuert werden. In diesem Ansatz liegt die Flexibilität, dass auf dem **SBC** eine komplexe Linux-Arbeitsumgebung aufgesetzt werden kann, während das Betriebssystem und der Typ des Interface-Computers keine Rolle spielen, sofern es eine Anbindmöglichkeit zum **SBC** gibt.

## 1.2 Auswahl der Hardware

Dieses Unterkapitel fokussiert sich auf die Abwägungen für die Hardware-Komponenten der Plattform. Es wird erläutert welche Faktoren beim Aussuchen eine Rolle spielten und die Eigenschaften eben dieser Komponenten beleuchtet.

### 1.2.1 Single Board Computer

Seit der Einführung des ersten Raspberry Pi haben sich **SBC** als immer nützlicher in immer mehr Projekten, nicht mehr nur von Firmen sondern auch Privatleuten, sogenannten "Makern", bewiesen. Dabei ist der Raspberry Pi 4 zwar immernoch der Platzhirsch, doch der Markt um **SBC** ist heute größer den je und die Auswahl an eben diesen ist nicht zu unterschätzen. Es gibt **SBC** in verschiedenen Konfigurationen, für verschiedene Gebiete spezialisiert und in vielen Preisregionen. Das Herzstück bildet hierbei immer der System on Chip (**SoC**), welcher über Geschwindigkeit, I/O-Fähigkeiten und Softwarekompatibilität entscheidet.

Für die Embedded Machine Vision Plattform sollte der **SBC** mindestens zwei Ports für Kameras haben, wenn möglich eine Peripheral Component Interconnect Express (**PCIe**)-Schnittstelle für Erweiterungen, ein kompaktes Design, ein möglichst geringen Preis und einen leistungsstarken **SoC** mit genug Arbeitsspeicher und weiter Verbreitung haben. Beim NanoPi M4V2 vom Hersteller FriendlyElec ist dies in allen Kategorien der Fall. Zusätzlich ist dieser **SBC** auch auf diversen Shops im Internet verfügbar, wenn auch Stand Februar 2023 nicht beim Hersteller direkt. Das Printed Circuit Board (**PCB**) hat einen ähnlichen Formfaktor wie der Raspberry Pi und selbst im Metallgehäuse mit aktiver Kühlung, **PCIe** Daughterboard, Antennen, und Kameras ergäbe sich ein sehr kompakter Aufbau [23], [30]. Das Gehäuse des NanoPi M4V2 erlaubt indes sogar, dank Gewinde, das Montieren auf einem Kamerastativ, was ideal für viele Machine-Vision-Anwendungen ist. Auch der eingebaute Lüfter ist kaum hörbar und hält den **SoC** kühl genug.

Beim **SoC** handelt es sich um den RK3399 von der Firma Fuzhou Rockchip Electronics Co., Ltd. [48], welcher mit zwei ARM Cortex-A72 Kernen mit einer Taktrate von 2,0 GHz und vier ARM Cortex-A53 mit 1,5 GHz ausgestattet ist. Alle sechs Kerne basieren auf der ARMv8 64-bit Instruction Set Architecture (**ISA**). Als Graphics Processing Unit (**GPU**) steht eine ARM Mali T860-MP4 parat und zusätzlich eine Video Processing Unit (**VPU**), welche neben den üblichen MPEG-Formaten auch H.264, H.265 und VP9 decodieren und immerhin H.264 sowie VP8 encodieren kann [33]. Auch wenn zum aktuellen Stand des Projekts keine Hardware-Kodierungen notwendig sind, sind solche Fähigkeiten immer vorteilhaft. Dem **SoC** stehen 4 GB LPDDR4 Arbeitsspeicher und ein optionales, bis zu 64 GB großes, embedded MultiMedia Card (**eMMC**)-Modul zu Verfügung. Für die I/O-Auswahl sind fast alle Bedürfnisse abgedeckt. Es befinden sich unter anderem Anschlüsse für Gigabit Ethernet, **USB** 3.0, **USB** Typ-C mit Stromversorgung für den **SBC**, zwei Kameras, HDMI, Audio, eine microSD Karte, Antennen für Wi-Fi und Bluetooth sowie eine 40 Pin General Purpose Input Output (**GPIO**)-Stifteiste. Zusätzlich lässt sich über eine weitere 24-Pin Stifteiste ein Daughterboard mit einem **PCIe** M.2 Key-M Sockel einbinden, der physisch zwei Lanes bzw. zwei sendende und empfangende Leitungspaare der **PCIe**-Generation 2.1 anbietet. Eine einzelne Lane dieses Standards bietet eine Datenbandbreite von bis zu 5 Giga-transfers per second (**GT/s**), in Summe somit 10 **GT/s** für den M.2 Sockel. Die beiden Kameraschnittstellen verwenden einen proprietären 30 Pin Flexible Printed Circuits (**FPC**)-Stecker mit 0,5mm Abstand von Pin zu Pin und sind mit

vier Lanes des MIPI Camera Serial Interface 2 ([CSI-2](#)) Standard an den Dual-Image Signal Processor ([ISP](#)) des RK3399 angebunden.

Eigenschaft	NanoPi M4V2	Raspberry Pi 4 Model B
1. SoC:	Rockchip RK3399	Broadcom BCM2711
2. CPU:	2x Cortex-A72 @2GHz + 4x Cortex-A53 @1,5GHz	4x Cortex-A72 @1,5GHz
3. Arbeitsspeicher:	4 GB LPDDR4	1/2/4/8 GB LPDDR4
4. USB 3.0:	Ja, 4 Ports	Ja, 2 Ports
5. Stromversorgung mit USB Typ-C:	Ja	Ja
6. Ethernet über USB Typ-C:	Unbekannt	Ja [5]
7. Kameraschnittstellen:	2x MIPI CSI-2 4-Lane 30 Pin	1x MIPI CSI-2 2-Lane 15 Pin
8. PCIe Schnittstelle:	Ja, PCIe Gen2 x2 M.2	Nein
9. PCB Maße:	85 mm x 56 mm	85 mm x 56 mm
10. Preis (UVP):	80 USD	ca. 50/63/79/104 Euro (reichelt.de)

**Tab. 1.1:** Vergleich von NanoPi M4V2 und Raspberry Pi 4 Model B

Für eine bessere Darstellung sind einige relevante Eigenschaften in Tabelle 1.1 nochmals aufgelistet. Zusätzlich wird der NanoPi M4V2 mit dem Raspberry Pi 4 Model B verglichen um ein Beispiel für eine Abwägung von [SBCs](#) zu geben. Entscheidend für den NanoPi M4V2 gegenüber vielen anderen [SBCs](#) war vor allem Punkt 7, die Kameraschnittstelle und Punkt 8, die [PCIe](#) Schnittstelle. Einziger Vorteil des Raspberry Pi 4 ist, dass dieser es erlaubt eine direkte Ethernet Verbindung zu einem anderen Computer über den [USB](#) Typ-C Port herzustellen [5]. Somit ist, sofern der angeschlossene Computer genug Strom liefern kann, nur ein einziges [USB](#) Typ-C Kabel notwendig um den Raspberry Pi unterwegs zu verwenden. Für den NanoPi ist indes zusätzlich ein Wi-Fi Netzwerk oder ein RJ-45 Kabel notwendig um über [SSH](#) oder Remote Desktop zuzugreifen.

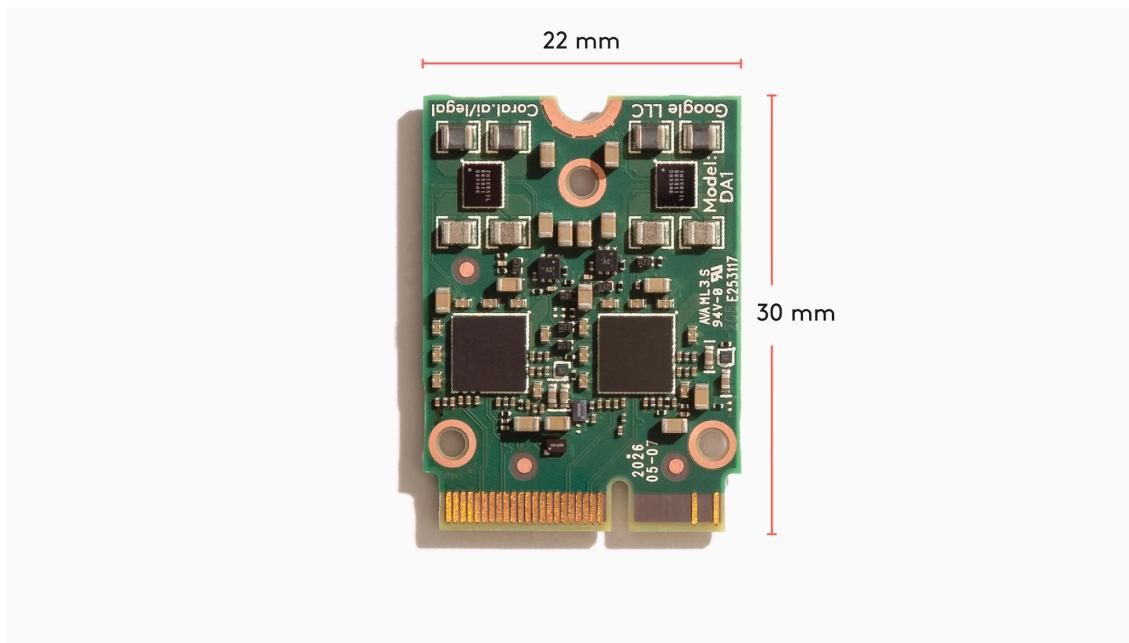
Leider sind weder NanoPi noch Raspberry Pi zum Zeitpunkt des Schreibens dieser Ausarbeitung zur UVP verfügbar. Allerdings findet man den NanoPi M4V2 am 4. Februar 2023 zumindest auf AliExpress für ca. 140 Euro während ein Raspberry Pi 4 Model B mit 4GB Arbeitsspeicher für ca. 200 Euro auf Amazon und diversen anderen Plattformen gehandelt wird. Daher ist der NanoPi, nach Beobachtung des Autors, auch in Zeiten schlechterer Verfügbarkeit günstiger, was nicht zuletzt auch an geringerer Nachfrage seitens der Community liegt. Grund hierfür sind wahrscheinlich der relativ unbekannte Hersteller, deutlich geringere Mengen an Dokumentation sowie Projekten und eine überschaubare Auswahl von offiziell kompatibler Software. Letzterer Punkt soll in späteren Kapiteln aufgegriffen werden.

Weitere Details zum NanoPi M4V2 und Tutorials finden sich auf dem Hersteller-eigenen Wiki [23]. Dort findet sich auch ein komplettes Schematic bzw. ein [PCB](#) Schaltplan zum

SBC.

### 1.2.2 Neuronaler Beschleuniger

Um auf besonders effiziente Weise neuronale Netzwerke in der echten Welt zu testen, eignet sich der **SoC** alleine nur bedingt. Sowohl die Central Processing Unit (**CPU**) als auch **GPU** des RK3399 sind hierbei einfach zu schwach um Inferenz von Modellen zur Objekterkennung und semantischen Segmentierung in Echtzeit durchzuführen [36]. Selbst wenn dies möglich ist, verbrauchen beide hierfür jeweils eine deutlich größere Menge elektrischer Energie da sie nicht von Grund auf für diese Aufgabe konzipiert wurden. Zwar sind **GPUs** in der Regel recht gut für massiv parallele Rechenaufgaben geeignet doch bringt auch ihre Architektur einen unnötigen Overhead mit und somit sollten **GPUs** hauptsächlich für das Training verwendet werden. Zu dieser Erkenntnis kam der Autor bereits bei seinem Praxissemester, bei der Vergleich von Desktop-**CPUs** und **GPUs** mit einem Nvidia Jetson Nano **SBC** und der Coral Edge TPU überzeugende Ergebnisse für letztere ergaben. Aus eben diesen Erfahrungen entstand auch die Entscheidung, eine Coral Edge TPU, genauer den ”M.2 Accelerator with Dual Edge TPU” [9], für dieses Projekt zu verwenden. Die Marke Coral gehört zu Google, welche auch die weit verbreitete Machine Learning Bibliothek Tensorflow entwickeln. Unter anderem darum ist der Software-Unterstützung und die Dokumentation rund um die Coral Edge TPUs sehr groß. Abbildung 1.1 (Bildquelle: [10]) ist ein Foto vom verwendeten Dual Edge TPU Modul mit Bemaßung. Die beiden größten Chips sind jeweils die eigentlichen **NPU** Application Specific Integrated Circuit (**ASIC**)s.



**Abb. 1.1:** Coral M.2 Beschleuniger mit Dual Edge TPU

Die Coral Edge TPUs sind **ASICs**, also anwendungsspezifische Chips welche einzig dafür ausgelegt wurden, um die Rechenoperationen zur Inferenz von Machine Learning Modellen möglichst schnell und energieeffizient auszuführen. In dieser Arbeit werden sie von nun mehr verallgemeinert als **NPUs** bezeichnet. Sowohl der Aufbau ihres Speichersubsystems, der Rechenwerke und internen Pipelines wurden unter anderem für Multiply-Accumulate (**MAC**)-Instruktionen von beispielsweise Faltungsoperationen optimiert. Ebenfalls zur Architektur gehören Hardware-Blöcke für nichtlineare, differenzierbare Zuweisungen, welche in Aktivierungsfunktionen zu finden sind [28]. Abbildung 1.2 (Bildquelle: Google durch [28]) zeigt den groben Aufbau einer Coral Edge TPU mitsamt der **MAC**-Rechenwerke in gelb und den Eingangs- und Ausgangspuffern mit Addierwerken für die Kombination von Zwischenergebnissen in blau. Für weitere Details hierzu wird auf Quelle [28] verwiesen. Auf zirka 8 MB internem Static random-access memory (**SRAM**)-Cache werden bei der ersten Inferenz die Parameter des Modells für effizienten Zugriff gespeichert. Alle weiteren Durchläufe zur Inferenz werden, sofern das Modell nicht gewechselt wird, danach nicht mehr negativ in der Performance beeinflusst [7]. Ein Inferenz-Durchlauf bedeutet hierbei, dass Eingangsdaten auf der Host-**CPU** vorverarbeitet (Gleitkomma zu Integer) und über einen **USB** oder **PCIe** Bus an die **NPU** übertragen werden. Anschließend durchlaufen die Daten das Modell auf der **NPU** und die Ausgangsdaten werden wieder auf die Host **CPU** zur Nachverarbeitung (Integer zu Gleitkomma) übertragen. Die Parameter des Modells müssen im Voraus durch einen sogenannten Quantisierungsprozess von einer 32-bit Gleitkomma Darstellung auf eine 8-bit Integer Form gekürzt werden. Dieser Prozess büßt Genauigkeit des Modells gegen Speicher- und Rechenzeiteffizienz ein, der Effekt lässt sich allerdings durch angepasste Trainingsroutinen deutlich reduzieren [40].

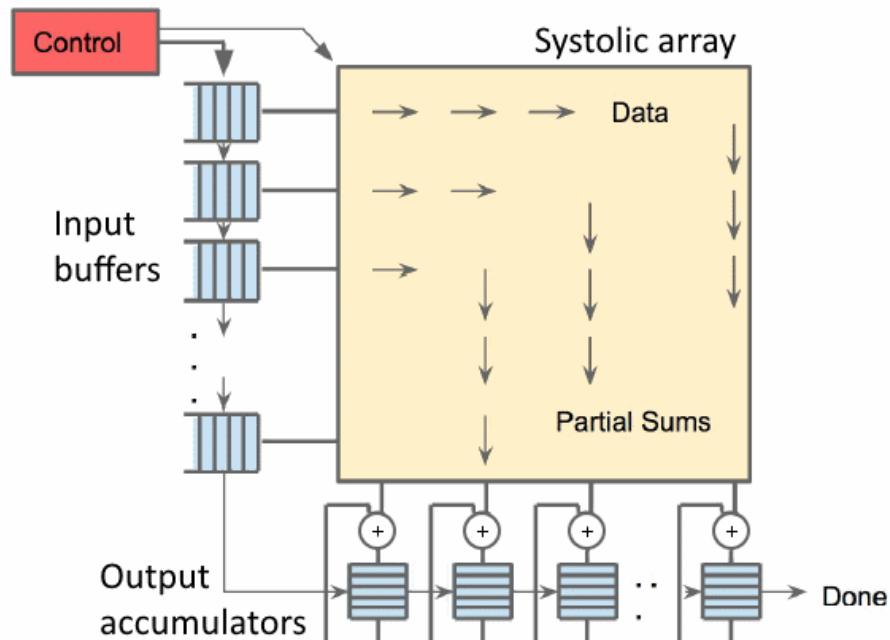


Abb. 1.2: Überblick grober Aufbau der Coral Edge TPU Architektur

Dabei gehören nicht nur geringere Genauigkeit sondern auch mangelnde Flexibilität zu den Hauptnachteilen von **NPUs**. Durch ihre feste und hoch-optimierte Architektur unterstützen sie nur eine Untermenge aller möglichen mathematischen Operationen und Schichten für neuronale Netzwerke, welche die Tensorflow-Bibliothek bietet. Eine Liste aller unterstützten Schichten ist unter [11] zu sehen und in Quelle [39] sind alle aktuellen Tensorflow Keras-Schichten aufgelistet. Auch müssen Tensor-Größen zur Kompilierzeit konstant sein und dürfen nur in maximal drei Dimensionen Werte führen. Werden diese Eigenschaften nicht eingehalten, läuft das Modell entweder gar nicht oder nur partiell auf der **NPU**. All diese Limitationen sorgen dafür, dass nicht alle Modellarchitekturen auf Coral **NPUs** auf Anhieb laufen. Allerdings zeigen kompakte Architekturen wie MobileNetV2 eine vielversprechende Performance [36], wenn genannte Einschränkungen bedacht werden.

Der "M.2 Accelerator with Dual Edge TPU" ist ein M.2 Key-E Modul mit zwei **PCIe** Busen, jeweils einer für eine der beiden verbauten **NPU** Chips. Dadurch, dass dieses Modul zwei Chips kombiniert, verbessert sich die Kompaktheit eines Systems. Durch Pipelining kann ein Modell in zwei geteilt werden und jede Hälfte läuft auf einem der Chips. So ist es möglich größere Modelle zu beschleunigen. Auch können zwei verschiedene Modelle parallel laufen ohne ineffizientes Wechseln der Parameter. Bei einer beworbenen Effizienz von 2 Trillions of Operations Per Second (**TOPS**) pro Watt, und 4 **TOPS** Leistung pro Chip sind so zirka 4 Watt Verbrauch bei 8 **TOPS** Gesamtperformance für das Modul anzunehmen. Die Einheit **TOPS** ist hierbei eine Analogie zu Floating Point Operations Per Second (**FLOPS**) für Gleitkomma-Operationen, angewandt für Rechenoperationen der Inferenz. Allerdings lassen sich **NPUs** weitaus schlechter auf Basis von **TOPS** vergleichen als **GPUs** auf Basis von **FLOPS**, da jede Architektur verschieden ist und je nach Parallelität der Recheneinheiten und Aufbau der Pipeline, wird die theoretisch maximale Menge an Multiplizier- oder Additionsoperationen pro Sekunde nicht immer ausgereizt. Ist ein Eingangsvektor beispielsweise größer als von den Rechenwerken erlaubt, muss dieser aufgeteilt und sukzessiv abgearbeitet werden, was Einbußen in Performance und Akkumulation der Zwischenergebnisse über Puffer bedeutet. Ist ein Eingangsvektor kleiner, wird der Rest mit Nullen aufgefüllt, was ebenfalls Performance vom theoretischen Maximum abzieht [28]. Echte Benchmarks mit verschiedenen Modellen der Bilderkennung oder Sprachverarbeitung sind in der Regel deutlich aussagekräftiger als einfache Angaben von **TOPS**, so die Meinung des Autors.

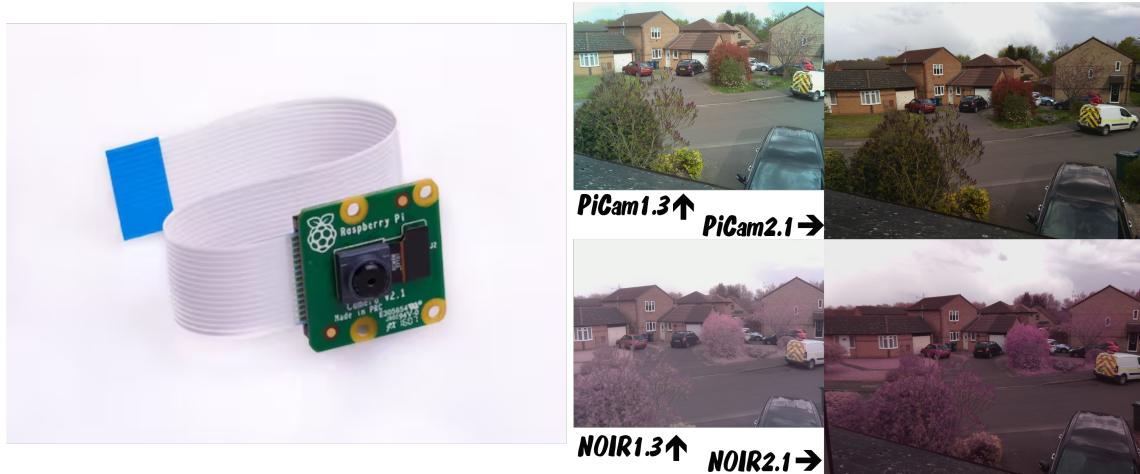
### 1.2.3 Kamera-Sensoren

Für die Kamerasensoren wurden die Raspberry Pi V2 Kameramodule gewählt. Grund hierfür ist eine gute Verfügbarkeit, zahlreiche Dokumentation mit verfügbaren Schematics [31], gute Software-Unterstützung mit Open-Source-freundlichen Frameworks wie libcamera [14], reichhaltiges Zubehör und ein akzeptabler Preis. Das Raspberry Pi V2 Kameramodul wurde 2016 eingeführt und ist aus Ansicht des Autors eine der am weitesten verbreiteten Kameramodule in der "MakerGemeinde". Zwar gibt es bereits einige bessere

Module, was die Performance des Kamerasensors und das Preis/Leistungsverhältnis betrifft, doch dient das Projekt aktuell ohnehin in erster Linie als Machbarkeitsstudie bis der Aufbau final ist. Anschließend soll ein Wechsel auf bessere Sensoren ermöglicht werden.

Das Raspberry Pi V2 Kameramodul besteht aus einem Sony IMX219 Bildsensor auf einer Adapterplatine welche einen **FPC** Anschluss für ein 15 Pin, 1 mm Raster, 15 cm langes, **FPC** Flachbandkabel bereitstellt. Neben zwei Lanes an MIPI **CSI-2**, führt das Kabel noch einen Inter-IC bus (**I2C**) Bus, ein Enable-Signal, ein LED-Signal und eine 3,3 Volt Versorgungsspannung. Der IMX219 Sensor besitzt eine Auflösung von 8 Megapixeln, eine Größe von 1/4 Zoll, eine Pixelgröße von 1,12 auf 1,12  $\mu\text{m}$ , einen Infrarotfilter und unterstützt auf einem Raspberry Pi Video Übertragungsmodi von 1080p30, 720p60 und 480p90. Im Detail entspricht das jeweils einer Auflösung von 1920 auf 1080 Pixeln, 1280 auf 720 Pixeln und 640 auf 480 Pixeln. Die fest verbaute Optik besitzt eine Brennweite von 3,04 mm, eine Blende von F2.0, einen manuell anpassbaren Fokus im Bereich von 10 cm bis in die Unendlichkeit und eine Field of View (**FoV**) von horizontal 62,2° und vertikal 48,8°.

Für die meisten Anwendungen der Objekterkennung und darüber hinaus ist die Bildqualität des Kameramoduls unter guten Lichtverhältnissen ausreichend, so die Ansicht des Autors. Genauere Details zum Modul werden in späteren Kapitel beleuchtet. In Abbildung 1.3 ist ein solches Modul zu sehen (Bildquelle: [21]). Rechts daneben ist ein Vergleich vom alten Modul V1.3 gegenüber dem aktuellen Modul V2.1 jeweils mit und ohne Infrarotfilter (Bildquelle: [32]). Dabei ist gut zu beobachten, dass das neuere Modul eine bessere Auflösung, **FoV**, Dynamikumfang und Farbtreue produziert.



**Abb. 1.3:** Raspberry Pi V2 Kameramodul [21] und Vergleichsbild [32]

### 1.2.4 Zubehör für mobile Verwendung

Um ein kompaktes und mobil nutzbares Gesamtpaket zu erzeugen fehlt noch Zubehör für den **SBC**. Beispielsweise eine Halterung für die Kameras, hier ist geplant einen Deckel für

das Gehäuse des **SBC** mit dem 3D Drucker herzustellen. Dabei muss auf einen möglichst kompakten Aufbau geachtet werden, welcher beide Kameras für Stereo Vision exakt parallel zueinander ausrichtet. Sind die Kameras und der **SBC** mit samt **NPU** verstaut, lohnt sich für den mobilen Einsatz noch ein Kamerastativ.

Das allerwichtigste Zubehör ist aber eine Powerbank zur Stromversorgung. Der **SBC** wird über einen **USB** Typ-C Port mit Strom versorgt. Nach eigenen Tests sollte die Powerbank eine Spannung von 5 bis 5,1 Volt liefern und einen Strom von 3 bis 3,1 Ampere. Das Kabel sollte für diese Last ausgelegt sein. Laut Herstellerangaben ist kein Power Delivery Protokoll über Typ-C unterstützt, somit sind 5 Volt bei 3 Ampere auch das Maximum für den Typ-C Port. Allerdings lässt sich der **SBC** alternativ über Pin 2 und 4 der **GPIO** Steckerleiste versorgen, was eventuell höhere Flexibilität bietet [23].

Zur Kommunikation mit einem anderen Computer empfiehlt es sich noch, sofern kein gemeinsames Wi-Fi Netzwerk verfügbar ist, ein RJ-45 Kabel sowie optional eine externe **USB**-zu-Ethernet Netzwerkkarte einzuplanen.

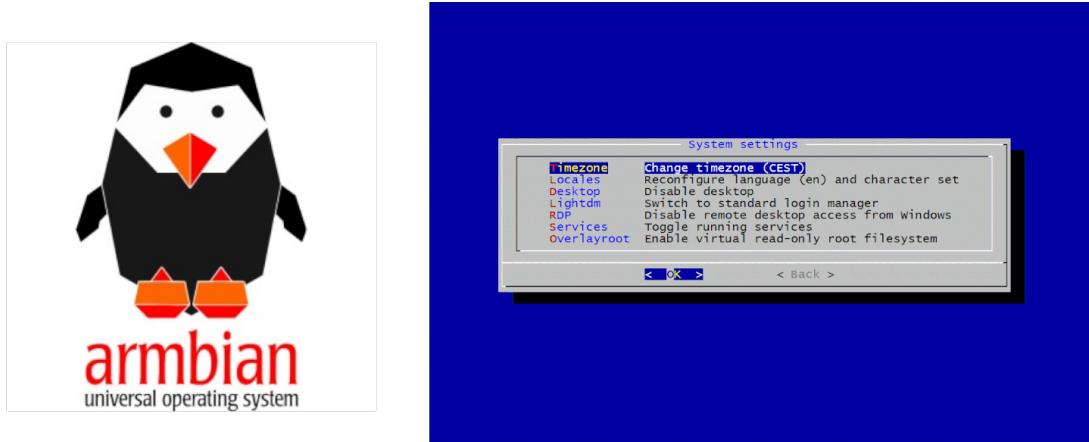
## 1.3 Auswahl der Software

Dieses Unterkapitel fokussiert sich auf die Abwägungen für die Software der Plattform. Jeweils das Betriebssystem und die Bibliotheken bzw. Frameworks.

### 1.3.1 Betriebssystem

Als Betriebssystem wurde die Armbian Linux Distribution gewählt [3]. Armbian bietet ein äußerst Open-Source-freundliches Linux Betriebssystem, welches für viele ARM-basierte Systeme und **SBCs** verfügbar ist und auf Debian und Ubuntu basiert. Dabei wird Armbian von einer großen Gemeinschaft aus Entwicklern auf viele Plattformen portiert. Die Grundlage hierfür ist, dass jeder durch einen vergleichsweise simplen Buildprozess seine eigenen Armbian Images kompilieren kann und leicht eigene Patches für verschiedene Hardware Konfigurationen einbringt. Dies hat große Vorteile, da so Unterstützung für Modifikationen an **SBCs** sowie exotische oder neue Hardware schnell integriert werden kann und man nicht warten muss, dass dies eine Distribution oder Firma erledigt. Auch kann, sofern kein proprietäres, Binär-Modul eines Herstellers inkompatibel ist, der aktuellste Linux Kernel verwendet werden, was positive Auswirkungen auf Performance, Software-Unterstützung und IT-Sicherheit hat. Durch den Buildprozess können unnötige Komponenten für bessere Sicherheit und Performance aus dem Betriebssystem entfernt oder experimentelle Komponenten hinzugefügt werden und auch die Desktop-Umgebung lässt sich wählen. Neben KDE Plasma, Gnome und vielen weiteren, wird auch xfce als leichtgewichtige grafische Umgebung angeboten, was sich optimal für **SBCs** eignet. Mit **armbian-config** ist außerdem ein mächtiges Kommandozeilen-Werkzeug vorinstalliert, welches diverse Einstellungen

am Betriebssystem vornehmen kann [37]. Unter der angegebenen Quelle sind alle Einstellungsoptionen zu finden. In Abbildung 1.4 (Bildquelle: Armbian und [37]) ist das Armbian Logo sowie ein Ausschnitt des armbian-config Werkzeugs zu sehen.



**Abb. 1.4:** Beispielbild armbian-config und Armbian Logo

Alternative Betriebssysteme, welche der Hersteller FriendlyElec direkt für den NanoPi M4V2 bewirbt, existieren ebenfalls [23]. Allerdings alle mitsamt deutlichen Nachteilen gegenüber Armbian, insbesondere in Anbetracht der Anpassungsfähigkeit und Aktualität. FriendlyDesktop, eine Ubuntu 18.04-basierte Distribution, wurde direkt für das **SBC** angepasst und beinhaltet zahlreiche vorinstallierte Software, analog zu Raspberry Pi OS für den gleichnamigen **SBC**. Es bietet einen vergleichsweise einfachen Einstieg in ein Desktop-taugliches Linux und stellt Treiber für **GPU** und **VPU** bereit. Damit sind auch hardware-beschleunigtes Dekodieren von Videos, Darstellen von zwei- und dreidimensionaler Grafik mittels OpenGL und das Nutzen aller I/O Fähigkeiten möglich. Der große Nachteil ist allerdings, dass einige dieser Treiber Closed-Source sind und nur deshalb korrekt funktionieren, da ein veralteter 4.4 Linux-Kernel genutzt wird, gegen diesen die Treiber kompiliert wurden. Des Weiteren wurden einige Hardware-Bugs in diesem Kernel noch nicht behoben und lassen sich auch nicht ausbessern, da eine Modifikation ohne Quellcode und Build Anleitung für den Autor unmöglich ist.

Inzwischen ist auch ein Debian-10-basiertes Betriebssystem für den NanoPi M4V2 verfügbar, welches ebenfalls Hardwarebeschleunigung unterstützt, doch ebenso auf einem veralteten Linux-Kernel 4.19.x setzt. Zusätzlich gibt es noch die Möglichkeit Android 8.1 auf dem **SBC** zu installieren, dies ist allerdings aus Sicht des Autors als Entwicklungsplattform suboptimal geeignet.

### 1.3.2 Python-Bibliotheken

Für Arbeiten an Projekten sind einige Bibliotheken und Treiber notwendig. Unter den Python-Bibliotheken sind vor allem Tensorflow und Tensorflow-Lite notwendig um neu-

ronale Modelle zu bauen, zu trainieren und auch Inferenz zu betreiben. Gerade Tensorflow, das ebenfalls von Google stammt, hat hierbei die beste Garantie direkt mit den **NPUs** zu funktionieren und wurde vom Autor bereits in der Vergangenheit in Kombination mit diesen verwendet. Allerdings ist es auch möglich Modelle aus Pytorch und anderen Machine-Learning-Bibliotheken durch Export und Import im Open Neural Network Exchange (**ONNX**)-Format auf den **NPUs** ausführen zu lassen [35]. Das **ONNX** Format erlaubt es, neuronale Modelle zwischen verschiedenen Frameworks und Werkzeugen auszutauschen und erlaubt Inferenz Durchläufe auf verschiedenen Plattformen [46].

Um ein neuronales Modell, wenn es schließlich in Tensorflow importiert wurde, auch auf einer **NPU** zu verwenden, ist eine weitere Bibliothek notwendig. Tensorflow-Lite stellt Werkzeuge für die Quantisierung, also das Konvertieren in ein Integer-Format bereit. Des weiteren bietet sie auch einen Interpreter, der zur Laufzeit für den Datenaustausch mit der **NPU** notwendig ist. PyCoral als Bibliothek ist ein Wrapper um Tensorflow-Lite und kann substituierend installiert werden. Ein Beispiel für die Nutzung von Tensorflow-Lite und dem Workflow um ein Modell auf der **NPU** auszuführen kann auf Quelle [19] betrachtet werden.

Unabdingbar ist auch eine Bibliothek, welche das Manipulieren und Transformieren von Bild- und Videodaten erlaubt. OpenCV ist eine, aus der Machine-Vision-Vorlesung bekannte Bibliothek, die für diese Zwecke mächtige und performante Funktionen bietet. Es können auch diverse Filter angewendet werden.

### 1.3.3 Treiber und Frameworks

Auf Seite der Treiber sind vor allem der Treiber für die **NPUs** und die Kameras zu nennen. Für erstere wird `libedgetpu1-std` installiert. Dieses Paket enthält den Laufzeittreiber für die **NPU**, welches vom `edgetpu-compiler` als Modell-Compiler ergänzt wird. Nähere Informationen zur Installation auf dem **SBC** finden sich in späteren Kapiteln.

Als Framework für die Interaktion mit den Kamerasensoren soll libcamera verwendet werden. Libcamera ist ein Open-Source-freundlicher Kamera-Stack, welcher unter anderem den Sony IMX219 Kamerasensor und **ISP** des RK3399 unterstützt. Er bietet Funktionen an, um Aufrufe für Kameradaten in eigene Software zu integrieren.

# 2 NPU

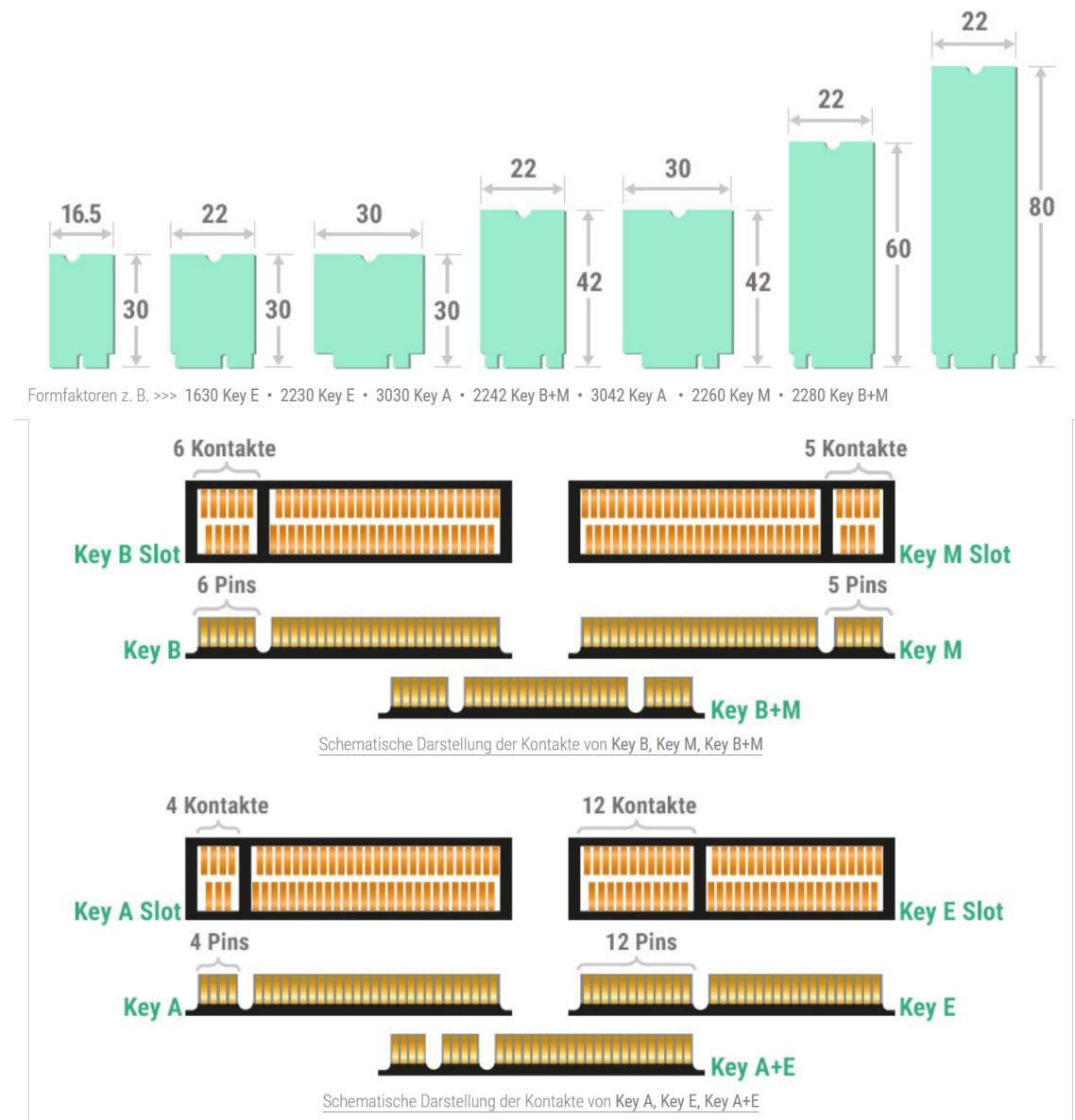
Dieses Kapitel fokussiert sich auf die Arbeit, welche notwendig war um beide Coral Edge TPUs am **SBC** erfolgreich einzusetzen. Es wird unterteilt in die Hardware und Software, welche jeweils notwendig war um letztendlich alle Probleme zu lösen.

## 2.1 Hardware

### 2.1.1 Anforderung

Das "M.2 Accelerator with Dual Edge TPU" Modul ist weder mechanisch noch elektrisch direkt mit dem Add-On M.2 **PCIe** Sockel des NanoPi M4V2 **SBC** kompatibel. Mechanisch, weil das Modul einen M.2 Key-E 2230 Sockel benötigt, dessen Format normalerweise nur für 22 mm breite und 30 mm lange Wi-Fi Karten oder vergleichbare Module verwendet wird. Dem Add-On Daughterboard des **SBC** fehlt hierzu eine Surface-mounted device (**SMD**) Mutter, über die eine solch kurze Karte installiert werden könnte. Außerdem sind die Kontaktfinger auf einem Key-E Modul nicht mit denen eines Key-M oder Key-B+M Modul kompatibel. Um falsche Installation zu vermeiden, sind in der Regel Einkerbungen an diesen Leiterplattenrandverbindern vorhanden [12]. Elektrisch sind diese Steckertypen ebenfalls nicht interoperabel, denn sowohl die Positionierung der **PCIe** Signal- und Spannungsfinger als auch die Menge an **PCIe** Lanes und Bussen unterscheiden sich. Die M.2 Key-M Schnittstelle, welche häufig für Solid State Drive (**SSD**) Module verwendet wird bietet entweder Serial AT Attachment (**SATA**) oder **PCIe** Datenleitungen während M.2 Key-E eine diversere Auswahl an Bussen zur Verfügung stellt. Optional sind hier neben zwie **PCIe** Bussen auch **I2C**, **USB**, Secure Digital Input Output (**SDIO**), Universal Asynchronous Receiver Transmitter (**UART**) und Puls-Code-Modulation (**PCM**) zu finden [45]. Abbildung 2.1 (Bildquelle: [12]) stellt eine Übersichtsgrafik für den mechanischen Unterschied zwischen den M.2 Formaten dar.

Wie eventuell bereits vom Leser erkannt, wird hier bei der **PCIe** Schnittstelle zwischen einem Bus und einer Lane unterschieden. Ein Bus ist bezieht sich in dieser Arbeit immer auf eine für sich geschlossene bidirektionale Verbindung eines **PCIe**-Host und **PCIe**-Device. Beim Host handelt es sich meistens um die **CPU** eines beliebigen Systems und ein Device ist zum Beispiel eine Grafikkarte oder eine Netzwerkkarte, somit eine Erweiterung. Ein Bus kann 1, 2, 4, 8 oder 16 Lanes besitzen, was sich auf die Menge an sendenden (TX)

**Abb. 2.1:** Überblick der M.2 Formate

und empfangenden (RX) differenziellen Signalpaaren bezieht. Zuzüglich hat jeder Bus auch immer ein Taktsignalpaar (CLK).

Daraus ergibt sich nun auch eine weiter Anforderung an einen Adapter für das "M.2 Accelerator with Dual Edge TPU" Modul. Nicht nur muss mechanisch und elektrisch von einem M.2 Key-M auf ein Key-E Interface gewechselt werden, es muss auch aus einem PCIe Bus zwei Busse gemacht werden, da jeder TPU ASIC auf dem Modul jeweils einen eigenen Bus benötigt. Die Antwort auf diese Anforderung soll dabei möglichst kompakt, günstig und, da es sich um eine mobiles Anwendungsszenario handelt, auch energieeffizient sein.

### 2.1.2 Konzeption des PCIe Adapter

Nachdem nun alle Anforderungen gegeben sind, kann mit der Konzeption des Adapters fortgeführt werden. Hierbei stellt sich die Frage wie ein mögliches **PCB** aussehen muss und welche Komponenten in Frage kommen. Für das **PCB** gelten die gleichen Richtlinien an die Form und Maße wie für andere M.2 Key-M Platinen. Dies bedeutet dass das **PCB** eine Länge von 80 mm und Breite von 22 mm hat und 0,8 mm stark sein darf um korrekt im M.2 Stecker des **SBC** zu sitzen. Entlang einer Kante müssen sich Kontaktfinger für den M.2 Key-M Anschluss befinden und auf der gegenüberliegenden Kante eine halbkreisförmige Aussparung zur Befestigung mittels Schraube. Auf dem **PCB** muss ein M.2 Key-E Sockel sitzen und eine **SMD** Mutter über die das **NPU** Modul angebracht wird. Es muss hierbei genug Platz unterhalb des Moduls gelassen werden, so dass der Großteil aller weiteren Komponenten zwischen Sockel und Kontaktfingern zu platzieren sind.

Um aus einem **PCIe** Bus zwei zu erzeugen ist ein sogenannter **PCIe** Packet Switch **ASIC** notwendig. Dabei wird in der Regel ein Upstream Port vom Host in zwei oder mehrere Downstream Ports umgewandelt. **PCIe** Switches beherbergen alle hierfür notwendigen analogen und digitalen Schaltungsblöcke wie die Paketumleitungslogik, Konfigurationsregister und Taktpuffer. Diese Art der Komponenten gibt es von diversen Herstellern in unterschiedlichen Konfigurationen und Verpackungen beziehungsweise Packages. Der Autor entschied sich für diesen Adapter den ASM1182e Packet Switch des Herstellers ASMedia Technology Inc zu verwenden [4], da dieser ein kompaktes Package (QFN48) mit relativ einfacher Beschaltung und einem erschwinglichen Preis verbindet. Zwar akzeptiert dieser nur eine einzelne Lane an **PCIe** Generation 2 (5 GT/s) als Upstream Port, doch würden konkurrierende Chips schlicht nicht auf das **PCB** passen. Somit wird nur eine der beiden Lanes des **SBC** verwendet, doch ist dies ein akzeptabler Kompromiss, da die **PCIe** Schnittstelle ohnehin nicht der Flaschenhals des Systems darstellt. Mehr hierzu in späteren Kapiteln.

Die restlichen Komponenten des Adapters dienen zur Grundbeschaltung des ASM1182e wie beispielsweise die Spannungsversorgung, Konfigurationswiderstände und Stecker für das Debugging. Die Platine wurde in der webbasierten CAD Designsoftware EasyEDA erstellt [13]. Grund hierfür ist, dass EasyEDA im Vergleich zu Branchenstandards wie Altium Designer kostenlos ist im Gegensatz zu Open-Source Alternativen wie KiCad einen großen Teilekatalog anbietet, welcher einfach in eigene Projekte importiert werden kann. Durch die Kollaboration mit dem Komponentendistributor LCSC und Platinenhersteller JLCPCB lassen sich fertige Designs direkt und kostengünstig zur Herstellung inklusive der Komponenten bestellen. Dennoch empfiehlt der Autor für komplexere Designs auf die eben genannten Konkurrenten auszuweichen, da einige der vorgefertigten Teilebibliotheken auch fehlerhaft sein können und insgesamt EasyEDA deutlich limitierter in seinen Werkzeugen und Fähigkeiten ist.

Es ist hierbei anzumerken, dass der Autor für die Idee dieses Adapters von einem Github Projekt des Nutzers magic-blue-smoke inspiriert wurde [17]. Die Auswahl der Komponenten, das Schematic und das Layout wurden aber in eigener Arbeit bewerkstelligt.

### 2.1.3 Implementation des PCIe Adapter - Schematic

Für den verwendeten ASM1182e PCIe Packet Switch gibt es online keine verfügbaren Datenblätter, allerdings war es dem Autor möglich durch direkte Kontaktaufnahme mit dem Hersteller eine Version des Datenblattes zu erhalten. Zusätzlich wurden verfügbare Schematics Reverse Engineered um alle offenen Fragen zu klären. Die Beschaltung des ASM1182e ist vergleichsweise simpel. Für das gesamte Schematic wird auf Abbildung A.1 im Anhang verwiesen.

Auf der ersten Seite des Schematics sind der ASM1182e, zugehörige Konfigurationswiderstände und beide M.2 PCIe Schnittstellen platziert. Bei U4 handelt es sich um eine Bibliothek welche von anderen Nutzern erstellt wurde, was im späteren Verlauf einige Probleme verursacht und dieser Umstand wird in späteren Kapiteln erläutert. Von dessen Kontaktfingern wird eine Lane an PCIe (PET, PER und REFCLK) sowie dessen Hilfssignale, ein SMBus und die 3,3 Volt Versorgungsspannung abgeleitet. Bei den Hilfssignalen handelt es sich um PEWake#, CLKREQ#, PERST# und SUSCLK. Das '#' referenziert hierbei darauf, dass ein Signalpin Active-Low ist, folgend bei negativem Logikpegel durchschaltet. Das PeWake Signal kann von einem PCIe Device genutzt werden um einen Bus aus einem Schlafmodus zu wecken und dessen Spannungs- und Taktversorgung zu reaktivieren. CLKREQ kann vom PCIe Device genutzt werden um den Haupttakt des Busses anzufordern, wenn dieses Signal deaktiviert ist kann somit Energie gespart werden, für einen normalen Datenaustausch muss es allerdings aktiv sein. PERST ist das Resetsignal des Busses und wird vom Host genutzt um zu signalisieren, dass alle Versorgungsspannungen stabil sind. Dieser Mechanismus vermeidet ein fehlerhaftes Hochfahren von Geräten bei zu niedriger Versorgungsspannung was zu Instabilität, Bugs oder gar Beschädigung führen kann. Zusätzlich kann PERST auch einen warmen Reset eines PCIe Device erzwingen. SMBus ist ein optionales Feature des PCIe Standards, ist verwandt mit I2C und dient für Kommunikation- und Managementzwecke zwischen Komponenten ([25] Kapitel 3.2.4). Als Beispiel ist zu nennen, dass manche PCIe Packet Switches über SMBus statt externe Widerstände konfiguriert werden können indem Register gesetzt werden. SUSCLK ist ein optional vom Host getriebener 32,768 kHz Takt [22], welcher in manchen Devices vermutlich für Low-Power Echtzeituhren notwendig ist. Eine genaue Erklärung konnte der Autor nicht finden.

Jedoch sind die meisten dieser Funktionen auf dem Adapter ungenutzt und nur für Zwecke der Widerverwendung und des Debuggings vorhanden. Auf dem NanoPi Daughterboard sind PERST und PEWake standardmäßig durch einen Pull-up immer High, CLKREQ immer Low. Die SMBus und SUSCLK Pins sind nicht verbunden. Diese Information konnte der Autor durch Kommunikation direkt mit dem Hersteller aufnehmen.

An einigen Stellen auf dem PCB befinden sich 0 Ohm SMD Widerstände. Diese dienen jedeglich für Debugging-Zwecke und werden entweder bestückt oder weggelassen. Als U1 ist auf der ersten Seite des Schematic der eigentliche Packet Switch zu betrachten. Alle, aus Sicht des Devices, sendenden Signalpaare (PER und PET\_LINK) müssen über 100 nF Kondensatoren in Serie AC-gekoppelt werden ([24] Kapitel 4.3.5.1). Diese hat zum

Grund, dass durch eine AC-Koppling jegliche Gleichstromanteile des Signals entfernt werden, und so ein Empfänger vor Überspannung geschützt werden kann. Der Packet Switch wandelt nicht nur einen PCIe Bus in zwei sondern reicht auch das PERST Signal an seine Downstream Ports weiter. Dadurch wird, so die Vermutung des Autors, sichergestellt, dass Geräte hinter dem Switch erst hochfahren dürfen, nachdem der Switch selbst hochgefahren ist. Oben Links auf der ersten Seite sind alle Konfigurationswiderstände gelistet, jeweils mit Beschriftung welche Funktion im Falle einer Bestückung erfüllt wird. Diese dienen ebenfalls zu Debugging-Zwecken und sind standardmäßig alle bestückt. Auf genauere Details wird nicht weiter eingegangen, jedoch die Konfiguration auf Basis von anderen Anwendungsbeispielen gewählt. U2 ist der M.2 Key-E Sockel für das NPU Modul.

Auf der zweiten Seite ist ein Spannungsregulator (U3) welcher die benötigte 1,2 Volt Versorgungsspannung aus den 3,3 Volt für den ASM1182e erzeugt. Zusätzlich befindet sich noch ein MOSFET Transistor auf dem Board, welcher dazu dienen soll die 1,2 Volt Spannung erst nach Deaktivierung des PERST Signals hochfahren zu lassen. Allerdings wird dieser nicht verwendet da er für einen einwandfreien Betrieb nicht notwendig ist. Somit ist R18 nicht bestückt, dafür R6. Um die Spannungen für Tests abgreifbar zu machen ist eine 3-Pin Steckerleiste installiert. Alle Spannungen werden mit einem 100 nF Kondensator pro Pin, mit möglichst kleinstem Abstand zum jeweiligen Pin des Chips, gepuffert. Zusätzlich sind noch 10  $\mu$ F und 2,2  $\mu$ F Kondensatoren als größere Zwischenpuffer installiert. Diese werden jeweils auf der, von der Spannungsversorgung weg-zeigenden, Seite des Chips platziert um zusätzlich für eine gleichmäßige kapazitive Pufferung der Spannungsversorgung für alle Pins des Chips zu sorgen.

Bei dem verwendeten Spannungsregulator handelt es sich um den TD6810 des Herstellers Techcode Semicon im SOT-23-5 Package. Der Hersteller ist nicht sonderlich bekannt doch wurde dieser Regulator aufgrund seines günstigen Preises und guter Verfügbarkeit, auf Wunsch auch für die PCB Bestückung, gewählt. Durch einen Spannungsteiler von zwei 150 kOhm Widerständen wird eine Ausgangsspannung von 1,2 Volt selektiert, was so im Datenblatt des Herstellers empfohlen ist. Sowohl der Regulator, als auch die Spule in Serie erlauben einen kontinuierlichen maximalen Strom von 800 mA, was in Anbetracht des maximalen Verbrauches von 480 mA des ASM1182e auf der entsprechenden Rail mehr als genug darstellt. Alle weiteren Komponenten der relativ simplen Beschaltung wurde ebenfalls nach Empfehlung des Datenblatts selektiert. Im normalen Betrieb ist laut Datenblatt mit einer Effizienz von über 85% zu rechnen, was bei Multiplikation von Ausgangsspannung mit Strom und einem resultierenden Faktor von 0,15 einer Abwärme von unter 0,1 Watt entspricht. Diese Rechnung ist sehr vereinfacht, reicht allerdings in diesem Fall aus und somit benötigt der Spannungsregulator keinen Kühlkörper. Hingegen ist der Verbrauch des ASM1182e unter Betrieb mit 0,808 Watt durchaus substanzell und laut eigenen Tests erreicht der Chip eine Oberflächentemperatur von bis zu 42°C bei Zimmertemperatur in der Umgebung. Es ist daher nicht abwegig in Zukunft einen kleinen Kühlkörper auf dem Chip zu installieren, notwendig erscheint dieser aber nicht.

### 2.1.4 Implementation des PCIe Adapter - Layout

Beim Layout des **PCB** musste in erster Linie auf die Verlegung des **PCIe** Busses und die Außenmaße und Löcher des Adapters Augenmerk gelegt werden. Alle relevanten Maße wurden, wie bereits angeschnitten, durch verschiedene Beispiele, Vorgaben und Messungen extrahiert.

Als erster Schritt, werden die Komponenten innerhalb der Platine positioniert. Danach werden die entsprechenden Leitungen beziehungsweise Traces verlegt. Dieser Prozess wird auch als Routing bezeichnet. Für das Routing der **PCIe** Signalleitungen müssen mehrere Regeln eingehalten werden. **PCIe** Pakete werden im Physical Layer durch eine 8b10b Codierung kodiert oder dekodiert und anschließend mit Serializern und Deserializern an HCSL Treiber und Empfänger gereicht. Durch die 8b10 Codierung bei **PCIe** bis Generation 2.1 sind die dual-simplex Lanes (RX und TX separat) selbstgetaktet, somit der Takt auf Empfängerseite aus dem Signal zurückzugewinnen. Bei neueren Versionen des Standards wird eine andere Codierung verwendet zur Selbst-Taktung verwendet. Dennoch muss zusätzlich pro Bus ein 100MHz Referenz Taktsignal verlegt werden [47]. Die eigentliche Datenübertragung bei **PCIe** Generation 2.x findet mit 5 **GT/s** beziehungsweise 2,5 GHz Grundfrequenz statt [41]. Daher fallen besondere Richtlinien als High-Speed-Signal an, denn digitale Signale mit solch hoher Geschwindigkeit haben durch ihre steilen Flanken ein breites, hochfrequentes Frequenzspektrum. Leistungsverlust, Inkonsistenzen entlang der Leitung, Impedanzvariation, elektromagnetische Interferenz, Crosstalk, Reflexion, Jitter, Rauschen und parasitäres Tiefpassverhalten des Systems können die Signalintegrität negativ beeinflussen.

Für den **PCIe** Standard selbst sind die Anforderungen vergleichsweise locker. Ein Längenausgleich in Form von Schlangenlinien muss in der Regel nur innerhalb eines differentiellen Paars ausgeübt werden (0,127 mm Toleranz), die einzelnen Paare hingegen nicht zueinander, da sie asynchron voneinander arbeiten und eine relativ große zeitliche Verzögerung von maximal 8 ns erlaubt ist [26]. Für das Adapterboard musste nur erstere Längenanpassung mittels eines Werkzeugs in der Software getätigt werden. Eine weitere geometrische Richtlinie ist, dass zwischen einzelnen Paaren, so weit möglich, ein Mindestabstand von 0,5 mm eingehalten werden sollte um Crosstalk Interferenz entgegenzuwirken. Auch hier wurde versucht dem möglichst nachzukommen, was bedeutet das Leitungen mit zunehmender Entfernung von Pads an Steckern oder Chips weitestgehend ausfächer sollten. Für bestmögliche Signalintegrität sollten Vias, also Durchkontakteierungen, welche Leitungen auf verschiedenen Platinenebenen verbinden, vermieden werden. Alles was an einer Leitung unnötig heraussteht (Studs) kann als Antenne wirken und unnötige Resonanzen und Rauschen erzeugen. Des weiteren bedeuten Vias ein kurzzeitiger Impedanzsprung entlang der Leitung, was Reflexionen und somit Leistungsverlust und Intersymbolinterferenz Probleme bedeutet. Kurven für Leitungen sollten entweder abgerundet werden oder mit 45° Winkeln gelöst werden, was im Layout des Autors der Fall ist. Die AC-Coupling Kondensatoren sollten möglichst klein sein, symmetrisch platziert und möglichst nah an der jeweiligen Leitung liegen. Auch dieser Anforderung wurde weitestgehend beachtet und Kondensato-

ren im kompakten 0402 Package gewählt. Weitere Empfehlungen für das Design von PCIe Leitungen sind unter [41] zu finden.

Um die Leitungsimpedanz von 50 Ohm und differenzielle Impedanz von 100 Ohm für PCIe Generation 2 einzuhalten wurden alle PCIe Signalleitungen auf der obersten Platinenebene verlegt und die nach unten benachbarte Ebene vollständig mit einer Ground Kupferebene bedeckt. Diese bildet eine Referenzebene für alle Ausgleichsströme der Signale und schützt sie einseitig vor elektromagnetischer Interferenz von außen. Eine Signalleitung sollte nie mals über einen Spalt in der Referenzebene verlaufen oder die Referenzebene wechseln (Ebenenwechsel) ohne beispielsweise Ground-Vias, über die ein Ausgleichsstrom fließen kann [41]. Für möglichst geringen Leistungsverlust ist Impedanzanpassung essenziell. Die Impedanz einer Leitung, in diesem Fall einer stripline, da sie nur einseitig von ihrer Referenzebene bedeckt ist, berechnet sich als Beispiel näherungsweise über folgende Formel nach der Norm IPC-2141 [43]:

$$Z_0 \approx \frac{87}{\sqrt{\varepsilon_r + 1,41}} \times \ln\left(\frac{5,98h}{0,8w + t}\right)$$

Die Impedanz ist unter anderem von der Dicke  $h$  des Dielektrikum mit Dielektrizitätskonstante  $\varepsilon_r$  zwischen Referenzebene und Leitungsebene, der Breite  $w$  der Leitung und Dicke der Kupferschicht  $t$  der Signalebene abhängig. Diese und weitere Faktoren bestimmen über Induktivität, Kapazität und Widerstand der Leitung. Bei differenziellen Leitungen, wenn zwei Striplines mit invertierten Signalpegeln direkt nebeneinander verlaufen, spielen noch Kopplungseffekte zwischen den Leitungen eine Rolle. So entsteht ein weiterer Impedanzwert, die, bereits erwähnte, differentielle Impedanz. Hier spielt zusätzlich noch der Abstand zwischen den beiden Leitungen als Parameter eine Rolle. In der Regel werden die benötigten Impedanzwerte schlicht in eine Rechensoftware von Seiten des Herstellers der Platine oder unabhängige Designsoftware eingegeben und diese schlagen dann notwendige Abstände und Dicken für Leitungen unter dem gegebenen Layerstackup vor. Der Layerstackup ist eine herstellerspezifische Übersicht der Abstände, Dicken und Dielektrizitätskonstanten von Schichten in einer Platine. Die verwendeten Materialien und Strukturen innerhalb des Dielektrikum sind hier hinterlegt und für die Planung der Ebenenbelegung unabdingbar. Für den Adapter wurde JLCPCB's JLC7628 Layerstackup mit gleichnamiger Weave Struktur, einer Dielektrizitätskonstante von 4,6, Standard FR4 Material und vier verfügbaren Ebenen gewählt [6]. Ein Querschnittsdiagramm bietet Abbildung 2.2 (Bildquelle: [6]). FR4 besteht als Platinenbasis aus gewobenen Glasfasern, welche in Epoxidharz eingegossen wurden und deren Webung als Weave bezeichnet wird. Umso feiner und lückenloser die Webung, umso gleichmäßiger und konstanter ist die Dielektrizitätskonstante zweidimensional über die Platine verteilt, was positiv für die Signalintegrität ist. Die gewählte Ebenenbelegung lautet von oben nach unten Signal-Ground-Power-Signal. Alle schnellen Signale verlaufen auf der obersten Ebene, alle langsamen (beispielsweise PERST# oder I2C) dürfen auch auf der unteren verlaufen. Es ist noch anzumerken dass die Power-Plane unter dem ASM1182e aufgeteilt ist, um alle 1,2 Volt Versorgungsspins zu erreichen.

**4-Layer Impedance Control Stackup**

Thickness

0.8mm	1.0mm	1.2mm	1.6mm	2.0mm
-------	-------	-------	-------	-------

a) JLC7628 Stackup:

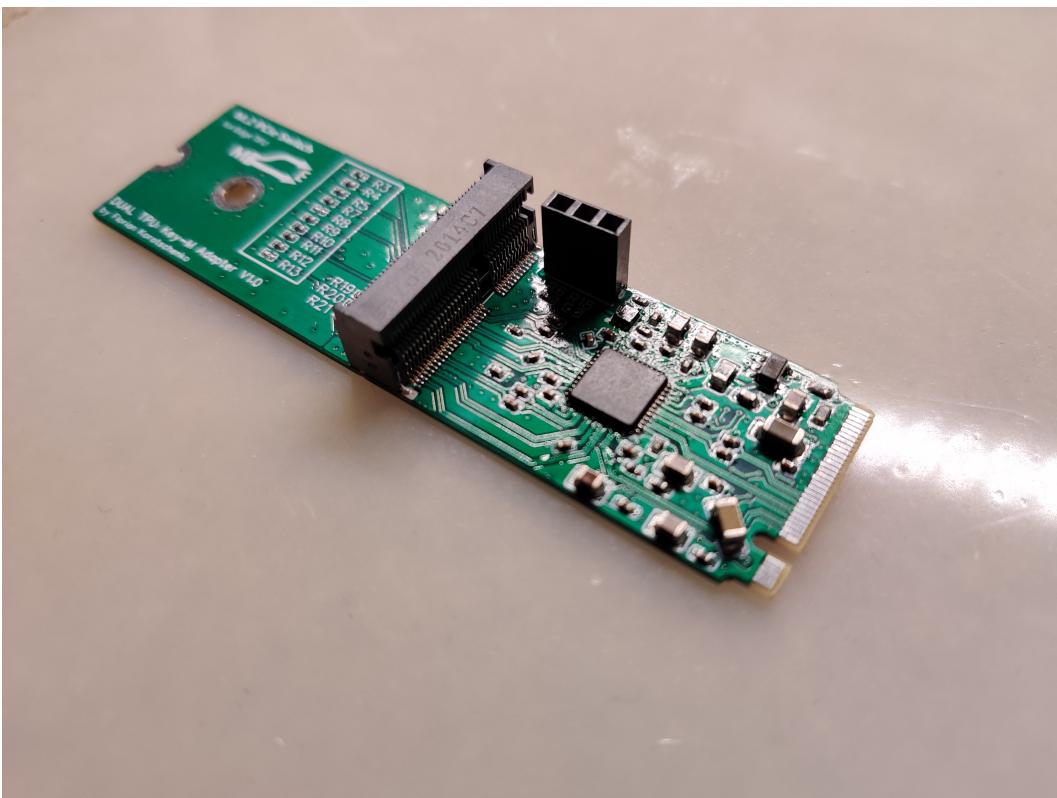
Layer	Material Type	Thickness		
Top Layer1	Copper	0.035 mm	0.25 mm (with copper core)	<b>Signal</b>
Prepreg	7628*1	0.2104 mm		
Inner Layer2	Copper	0.0152 mm		<b>GND</b>
Core	Core	0.2 mm		
Inner Layer3	Copper	0.0152 mm		<b>Power</b>
Prepreg	7628*1	0.2104 mm		
Bottom Layer4	Copper	0.035 mm		<b>Signal</b>

0.2mm (7.87 mil) is nominal thickness of 7628 prepreg. Use 7.1 mil as the thickness when the controlled impedance tracks are on top/bottom, use 8.1 mil when tracks are inside.

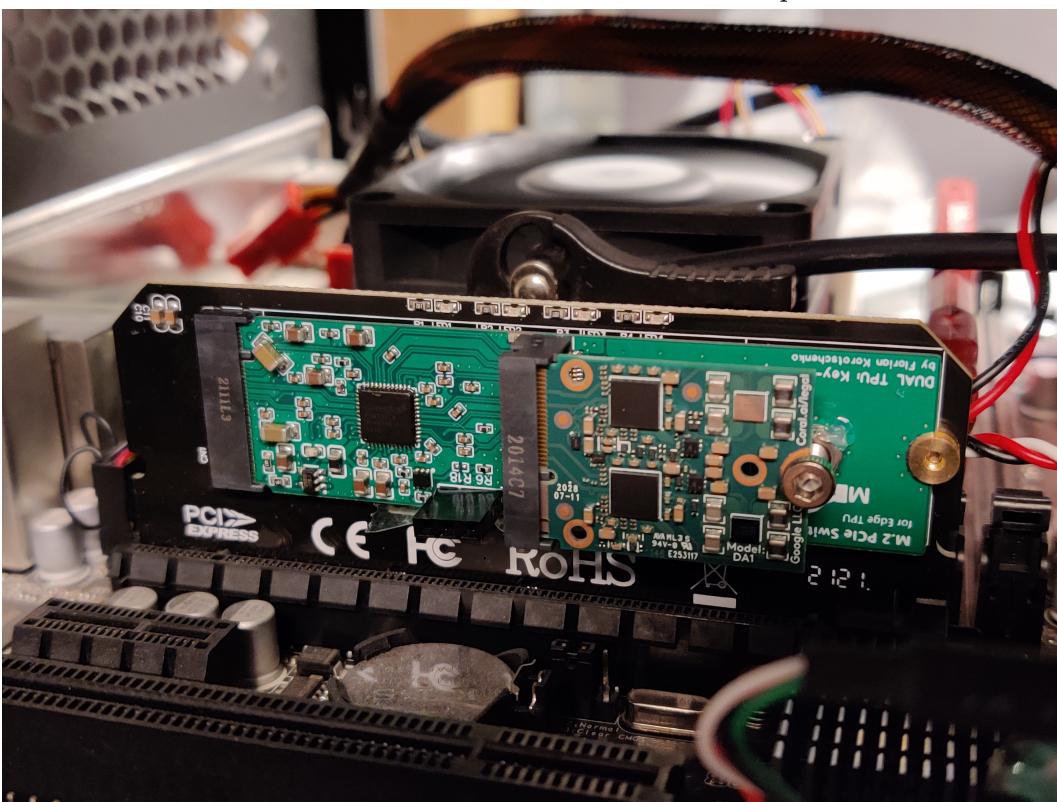
**Abb. 2.2:** JLC7628 4-Layer 0,8mm Dicke Layerstackup

Die einzige weitere erwähnenswerte Eigenschaft des Platinenlayout ist der Leiterplattenrandverbindern für den M.2 Key-M Stecker. Hier wurde auf Empfehlung der Lötstoplack rund um die Kontaktfinger entfernt um keinen unnötigen Widerstand zu erzeugen. Auch wurden alle inneren Kupferebenen der Platine bereits vor den Kontaktfingern abgeschlossen damit diese beim optionalen Abschleifen der Kanten nicht beschädigt werden. Die Kontaktfinger sollten einen Abstand zum Platinenrand von ca. 0,5 mm haben, ebenfalls um vor dem Abschleifen zu schützen. Die Kanten werden in der Regel abgeschliffen, damit eine Platinenplatte ohne viel mechanischen Widerstand in einen Sockel ein- und ausgesteckt werden kann [2]. Dieser Service wurde für den Adapter nicht angeboten, da er zu klein ist, jedoch wäre es in Zukunft möglich zwei Platinen nebeneinander fertigen zu lassen, so dass solch ein Prozess möglich wäre.

Das fertige Design der Platine wurde bei JLCPCB mit dem bereits benannten Layerstackup und grünem Lötstoplack bestellt. Aus Kostengründen wurde als Oberflächen Finish der Pads und Kontakte Hot air solder leveling (**HASL**) statt dem üblichen Electroless nickel immersion gold (**ENIG**) Prozess verwendet. Dies hat zur Folge dass die Kontaktfinger des Leiterplattenrandverbinder eventuell unebener sind und schlechteren Kontakt erzeugen [44]. Bei **ENIG** besteht für **PCB** Fertiger die Option solche Kontaktfinger zu vergolden, was in besonders hoher Langlebigkeit resultiert. Im Anhang sind die einzelnen Kupferebenen des fertigen Layouts vom Adapter unter Abbildung A.2 zu sehen. Fotos der fertigen **PCIe** Adapterplatine sind in Abbildungen 2.3 und 2.4 zu betrachten.



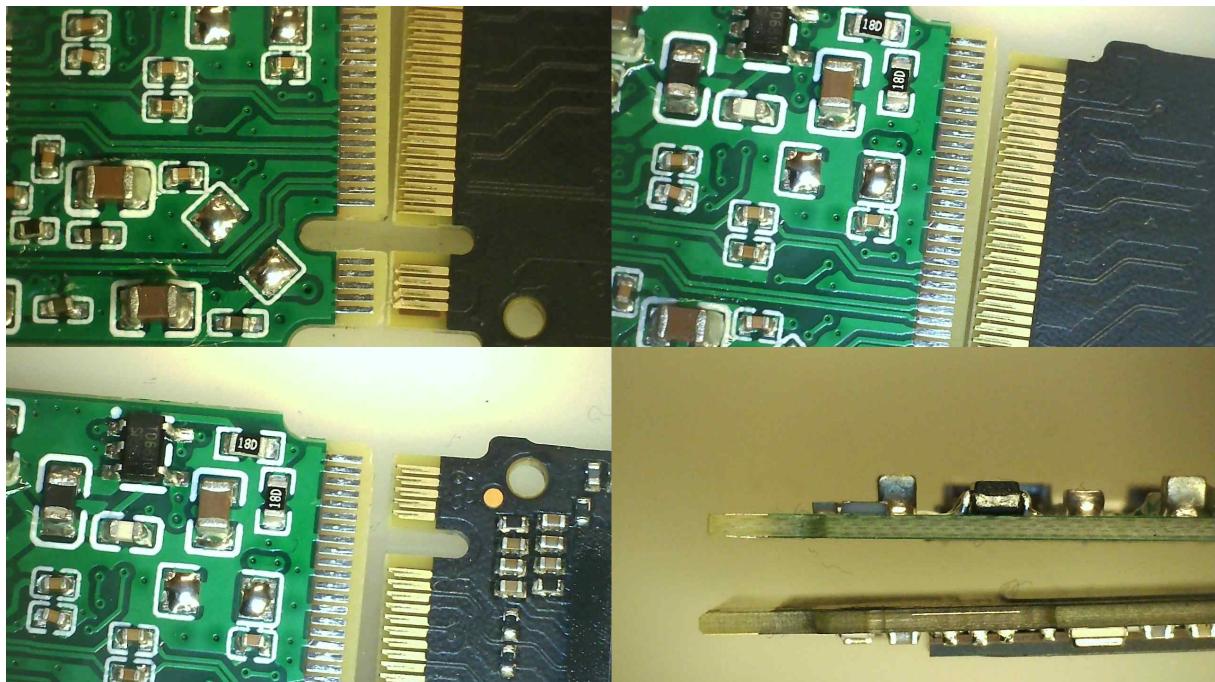
**Abb. 2.3:** Foto von bestücktem PCIe Adapter



**Abb. 2.4:** Foto von PCIe Adapter in Testsystem mit Coral Edge Dual TPU Modul

### 2.1.5 Fehler und Verbesserungspotential

Es gibt mehrere Schwachstellen an der Platine, welche bei der Nutzung teils deutliche Konsequenzen haben. Die erste, bereits angesprochene, Schwachstelle ist der Leiterplattenrandverbindern für den M.2 Key-M Stecker. Dieser hat wegen zu kleiner Dimensionierung der Fingerkontakte und einem suboptimalen Oberflächen Finish sowie mangelndem 45° Winkel an den Kanten Probleme guten Kontakt mit seinem Stecker aufzunehmen. Dies hat zur Auswirkung, dass für einen richtigen Kontakt die Platine leicht schräg eingebaut werden muss, was eine Fixierung mittels Schraube unmöglich macht. Stattdessen halten Gummibänder die Platine auf dem SBC. Wird dieser Kompromiss nicht eingehalten, lässt sich der Switch und dahinter hängende PCIe Geräte nicht vom Betriebssystem erkennen. Die passende Lösung wäre es, den gesammten Steckverbinder zu überarbeiten und die Kontaktfinger breiter und länger zu machen, Aussparungen tiefer anzusetzen, eine ENIG Oberflächenbeschichtung mit zusätzlicher Finger Vergoldung zu nutzen und 45° Anwinkelung der Platinenkante zu ermöglichen. In dem man zwei Platinen auf einmal als zusammenhängendes Panel bestellt, sollten die Größenanforderungen seitens JLCPCB für ein Abschleifen der Kanten erfüllt sein. In den nachfolgenden Fotos von Abbildung 2.5 ist ein Vergleich mit einer handelsüblichen Nonvolatile Memory Express (NVMe) SSD getätigert worden.



**Abb. 2.5:** Vergleich mit Kontaktfingern einer NVMe SSD

Weitere Punkt, die Potential zur Verbesserung haben sind zum Beispiel die Mutter, über die das NPU Modul befestigt wird, hier sollte in Zukunft eine adäquate Einlötz- oder Einpress SMD-Mutter genutzt werden statt einer normalen M3 Mutter, welche aktuell mittels

Sekundenkleber angebracht wurde. Entsprechend muss auch eine passende Schraube verwendet werden, welche die korrekten Maße für M.2 Halblöcher hat. Auch bietet es sich an, einen flacheren M.2 Key-E Stecker zu verwenden, um so die Platine in der Höhe kompakter zu gestalten. Gleichzeitig kann auf die Steckleiste für die Spannungsmessung verzichtet werden sowie die MOSFET Schaltung da diese unnötig ist. Stattdessen wären flache SMD Steckleisten oder Pad-Testpunkte besser für das Debugging geeignet und platzsparender. Einzig eine Zuführung für ein externes PERST# Signal oder das Abgreifen der NPU-spezifischen ALERT# Signale könnte noch sinnvoll erscheinen. Irrelevante Konfigurationswiderstände sollten ebenso verschwinden und nur noch das Nötigste über DIP Schalter gelöst werden. Selbiges gilt für die großen 100  $\mu\text{F}$  Kondensatoren. Insgesamt biete sich so eine zweite Revision der Adapterplatine an.

## 2.2 Software

Nachdem nun die Hardware des NPU Adapterboards ausführlich erläutert wurde kann zur Inbetriebnahme und der Software übergegangen werden.

### 2.2.1 Die Ausgangssituation

Für die ersten Tests wurde ein vorgefertigtes Armbian Betriebssystem Image (Armbian 22.02 Focal XFCE Stable mit 5.15 Linux Kernel) von der offiziellen Website für den entsprechenden SBC heruntergeladen und auf das eMMC Modul mittels MicroSD Adapter und dem Werkzeug balenaEtcher geschrieben. Um nicht bei jeder Ersteinrichtung eine zusätzliche Tastatur und Monitor verwenden zu müssen, empfiehlt der Autor eine UART-zu-USB-Brücke an die UART-Pins anzuschließen und mittels PuTTY und einer Baudrate von 1500000 Baud/Sekunde auf den SBC zuzugreifen solange kein Remote Zugriff aktiviert ist. Der eigentliche Einrichtungsprozess unterscheidet sich nicht sonderlich von anderen UNIX Betriebssystemen, es wird ein Benutzerkonto angelegt, Zeitzone und Tastaturlayout abgefragt und anschließend hat man bereits Zugriff auf das Command-line interface (CLI) über die Serielle Verbindung. Bei der Passwortwahl sollten am besten anfangs exklusiv Zahlen verwendet werden, da das Tastaturlayout unter Umständen zuerst im US-amerikanischen Modus ist.

Die ersten Schritte beim Austesten eines Betriebssystembuilds sind immer gleich. Zu allererst wird `/boot/armbianEnv.txt` bearbeitet und `extraargs=debug` als letzte Zeile eingefügt. Der Bootparameter "debug" sorgt hierbei dafür, dass beim Hochfahren statt des Splashscreen auf dem Bildschirm beziehungsweise statt Leere in der serielle Konsole alle Bootloader- und Kernel-Meldungen aufgelistet werden, was äußerst wichtig im Debugging ist. Weitere Bootparameter werden mit einem Leerzeichen dahinter angehängt. Anschließend können über `sudo armbian-config` in einer Option alle Pakete für den Remotezugriff mittels beispielsweise Windows RDP installiert und dieser aktiviert werden.

Gegebenenfalls ist es nötig für den Remotezugriff auch einen zweiten Benutzer zu erstellen. Wenn falsch konfiguriert, kann das Tastaturlayout hier ebenso geändert werden. Auch kann in armbian-config der PCIe Modus für den Slot auf Link Generation 2 (5 GT/s pro Lane) hoch gestuft werden. In einigen Fällen hat der Autor Fehlschläge bei der Verbindungsauftahme mit den Repositories des Paketmanagers verzeichnet. Um dieses Problem zu lösen ist es notwendig in armbian-config den API Mirror auf Automated umzuschalten. Der Quellcode und die Header des aktuellen Kernels sowie ein Kernel Update können ebenfalls installiert werden, bei eigens modifizierten Kernels ist vom Herunterladen von Headern oder Quellcode aber logischerweise abzuraten. Es gibt zahlreiche weitere Optionen in armbian-config, doch diese zu erläutern würde den Rahmen sprengen.

Wenn die PCIe Adapterplatine, egal ob mit oder ohne installiertem NPU Modul, in dem M.2 Slot steckt und der SBC hochfährt, führt dies zu einem Kernel Panic, so die Beobachtung des Autors. Bei Verwendung einer handelsüblichen NVMe gab es hingegen keine Probleme und das Gerät wurde korrekt unter `lspci` aufgelistet. In monatelanger Arbeit wurde mittels vieler Debugging-Versuche und Patches die Ursachen für dieses Problem eingegrenzt und behoben. In den nächsten Kapiteln sollen diese und deren Lösung präzisiert werden. Auszug A.1 im Anhang schildert die Details des Kernel Panic als Log-Datei der Kernel Ereignisse ("dmesg"). Nach präzise injizierten Print-outs von Checkpoint Meldungen und virtuellen sowie physischen Adressen im PCIe Kernel Modul während der Initialisierung konnte festgestellt werden, dass der Interrupt beim Lesen einer PCIe Adresse hinter dem Switch in der Funktion `rockchip_pcie_rd_other_conf(...)` ausgelöst wird. Konkret werden in der genannten Funktion 8-bit-, 16-bit- und 32-bit-Wörter für Konfigurationsdaten und Register aus den angeschlossenen PCIe Geräten mittels einem berechneten Offset ausgelesen. Im SoC ist dafür ein spezifischer Adressbereich für den PCIe Controller zugeordnet und Lesebefehle werden mit den Funktionen `readb()`, `readw()`, `readl()` (byte, word, long) auf diesem Bereich ausgeführt [27]. Dieser Bereich erstreckt sich von der Adresse F8000000 bis FE000000 über den internen Advanced eXtensible Interface (AXI) Bus, welcher die CPU, GPU, Arbeitsspeichercontroller und weitere I/O-Controller im SoC verbindet [15] (Seite 29). Anfänglich wurden der augenscheinlich limitierte 32-bit AXI Bus und einhergehend ein zu kleiner Speicherbereich für die Konfigurationsdaten nach Berechnung des Offsets vom Autor für die Fehler verantwortlich gemacht. Die Vermutung wurde daher begründet dass, sobald der Funktionsparameter `pci_bus->number` von 1 auf 2 wechselt und einhergehend ein höherer Adressbereich ausgelesen wird, der Kernel Panic zu Stande kommt. In Wirklichkeit kann der Grund für diese Fehler aber auf einen Bug in der CPU zurückgeführt werden, wodurch unbegründete Fehlalarme ausgelöst werden. Diese äußern sich mit nicht-maskierbaren Interrupt Subroutinen in denen ein Notfallabbruch des Bootvorgangs getriggert wird.

### 2.2.2 Der Betriebssystem Buildprozess im Detail

Um Debuggingversuche auszuführen und Fehler auf Kernel-Ebene zu umgehen muss der Quellcode des Betriebssystems auf tiefgreifende Art und Weise angepasst werden. Armbian

bietet hierfür einen äußerst komfortablen Buildprozess, welcher es erlaubt auch eigene Patches einzubringen. Im Folgenden Unterkapitel wird dieser Prozess beleuchtet.

Als Grundlage dient immer die sogenannte Buildumgebung, welche im Fall von Armbian nach Empfehlung immer eine möglichst aktuelle Ubuntu Installation darstellt. Zur Zeit als der Autor aktiv an der **NPU** Sektion arbeitete waren Ubuntu 21.04 und 21.10 empfohlen. Inzwischen wird Ubuntu 22.04 empfohlen. Das Betriebssystem muss auf einem x86 Computer oder einer virtuellen Maschine laufen. Es werden mindestens 25 GB **SSD** Speicherplatz und 4 GB Arbeitsspeicher sowie vier **CPU**-Kerne empfohlen. Da der Autor unter Windows arbeitete, wurde eine virtuelle Maschine mittels Oracle Virtualbox für diesen Zweck verwendet. Über ein geteiltes Verzeichnis können Dateien zwischen Hostsystem und virtueller Maschine verschoben werden.

Ist die Buildumgebung aufgesetzt und das Armbian Build Repository in ein geeignetes Verzeichnis geklont worden, kann zur Installation aller Abhängigkeiten zum ersten Mal `./compile.sh` im Verzeichnis ausgeführt werden. Nach dem Aufruf wird eine Graphical User Interface (**GUI**) angezeigt in dem das Board der Zielplattform ausgewählt wird. Zusätzlich wird abgefragt, ob ein minimales Image mit nur dem Bootloader und Kernel oder ein vollwertiges Betriebssystem gewünscht ist. Es können auch erste Konfigurationen am Kernel gemacht werden und Betriebssystemversionen von Debian und Ubuntu sowie die diversen grafischen Oberflächen und Zusatzprogramme gewählt werden. Alle weiteren Aufrufe des Buildvorgangs sahen beim Autor folgendermaßen aus:

**List. 2.1:** Verwendete Build-Optionen

```
1 sudo ./compile.sh EXPERT=yes CREATE_PATCHES=yes
    INSTALL_HEADERS=yes INSTALL_KSRC=yes KERNEL_CONFIGURE=no
```

Die Option "EXPERT" erlaubt die Auswahl von mehr Boards und dem experimentellen "edge" Branch, was bedeutet dass der neuste, verfügbare Linux Kernel für das Image verwendet wird. Dies steht im Gegensatz zur "current" Branch welche einen offiziell unterstützten, stabileren Kernel nutzt, der aber dafür älter ist und der "legacy" Branch welche einen alten Linux 4.x Kernel verwendet. In der Regel verwendet der Autor die "edge" Branch. Die Option "CREATE\_PATCHES" erlaubt das Erstellen eigener Patches, in dem der Buildprozess pausiert wird, und der Nutzer in der Zwischenzeit Änderungen am Quellcode machen kann. "INSTALL\_HEADERS" und "INSTALL\_KSRC" signalisieren das Bauen der Header- und Quelldateien des Kernels ins Image, welche später unter `/usr/src` zu finden sind. Wenn der Kernel vom Nutzer modifiziert wurde, ist diese Option unabdingbar, da ein Nachinstallieren der Quelldateien für diesen angepassten Kernel nicht möglich ist. Solche Dateien werden beispielsweise für die **NPU** Treiber verwendet, da dieser jeweils für die Endplattform gegen den verwendeten Kernel neu kompiliert wird. "KERNEL\_CONFIGURE" wurde gewählt um lästige Nachfragen zu brandneuen, noch nicht von Armbian selektierten, Treibern bei Auswahl der "edge" Branch zu vermeiden und stattdessen die empfohlene Kernelkonfiguration zu verwenden. Eine Übersicht aller Build Optionen ist in Quelle [38] zu begutachten. Es ist noch anzumerken, dass auch ein spezifischer Linux Kernel ausgewählt werden kann, hierzu wird in `build/userpatches/lib.config` in

der ersten Zeile mit beispielsweise `KERNELBRANCH="tag:v5.4.28"` ein Tag für das offizielle Github Repository von Linus Torvalds genannt.

Im Folgenden werden nun die eigentlichen Schritte für das Integrieren eigener Patches erläutert.

Als allererstes sollte der Kernel Quellcode, auf den Änderungen angewendet werden sollen, auf den Ursprungszustand zurückgesetzt werden. Dies kann mit Build Optionen wie `"CLEAN_LEVEL=sources"` automatisch getan werden. Jedoch verursacht diese Option aus Erfahrung des Autors manchmal Bugs, wodurch das Einfügen von Patches behindert wird. Unabhängig davon wie man sich entscheidet, es sollte auf jeden Fall sichergestellt werden, dass im Verzeichnis `build/cache/sources/linux-mainline/` kein Ordner für den gewünschten Kernel vorliegt. In dem man dies tut, geht man sicher dass beim nächsten Start des Buildprozesses der Quellcode des gewünschten Kernels unmodifiziert aus dem Linux Repository als benannter Ordner in dieses Verzeichnis neu heruntergeladen wird und so neue Änderungen nicht auf Basis von vorherigen, sondern auf Basis des originalen Quellcodes passieren. Die Dateien in `build/cache/sources/linux-mainline/` dienen immer als Ausgangspunkt für den zu bauenden Kernel.

Standardmäßig werden bereits Patches für die verschiedenen Zielplattformen angewendet. Diese, bereits von der Community bereitgestellten, Patches befinden sich in `build/patch/kernel/rockchip64-edge/`, wobei es entsprechend ein Verzeichnis für den "edge" und eines für den "current" Buildprozess gibt und neben "kernel" auch ein "u-boot" Verzeichnis für Patches des Bootloaders existiert. Neben eben diesen, bereits existierenden, Patches gibt es analog das Verzeichnis `build/userpatches/kernel/rockchip64-edge/` für selbst erstellte Patches. Wünscht man Patches aus `build/patch/` zu überschreiben, kann man die entsprechende .patch-Datei in `build/userpatches/` unter den gleichen Pfad und Namen kopieren und dann bearbeiten. Es gilt aber zu beachten dass beide Arten von Patches nach alphabetischer Reihenfolge angewendet werden, was bei Abhängigkeiten zwischen Patches zu Probleme führen kann.

Sind alle Patches angewendet, pausiert, wie bereits erwähnt, mittels `"CREATE_PATCHES"` der Buildprozess und der Nutzer kann alle Quelldateien unter `build/cache/sources/linux-mainline/` mit dem Editor der Wahl bearbeiten. Anschließend werden die Änderungen mit der Enter-Taste bestätigt, eine neue .patch-Datei unter `build/output/patch/` hinterlegt und der Buildprozess fortgesetzt. Nun hat man erfolgreich einen eigenen Patch erstellt und kann die entsprechende Datei für zukünftige Durchläufe in `build/userpatches/` verschieben.

Das fertige Image findet sich in `build/output/images`. Der beschriebene Prozess kann jederzeit wiederholt werden und je nach Hardware auch über eine Stunde in Anspruch nehmen.

### 2.2.3 Die Lösung mittels Patches

Nachdem nun der Erstellungsprozess für Patches erläutert wurde, geht es nun um den finalen Patch, welcher die fehlerfreie Anbindung der **NPU** an den **SBC** ermöglichte. Die komplette Patch-Datei befindet sich aus Platzgründen im Anhang als [A.2](#). Zeilen, welche im Vergleich zum Original entfernt oder modifiziert wurden, sind mit einem vorangestellten Minus (orange) angeführt. Eingefügte Zeilen mit einem Plus (grün). Außerdem gilt es für den Leser zu beachten, dass eine Patch-Datei immer nur die relevanten Ausschnitte aus dem Quellcode anzeigt, daher sind Sprünge normal und werden mit einem ”@@” (grau) annotiert. In den folgenden Unterkapiteln werden jeweils die einzelnen Aspekte dieses Patches erklärt.

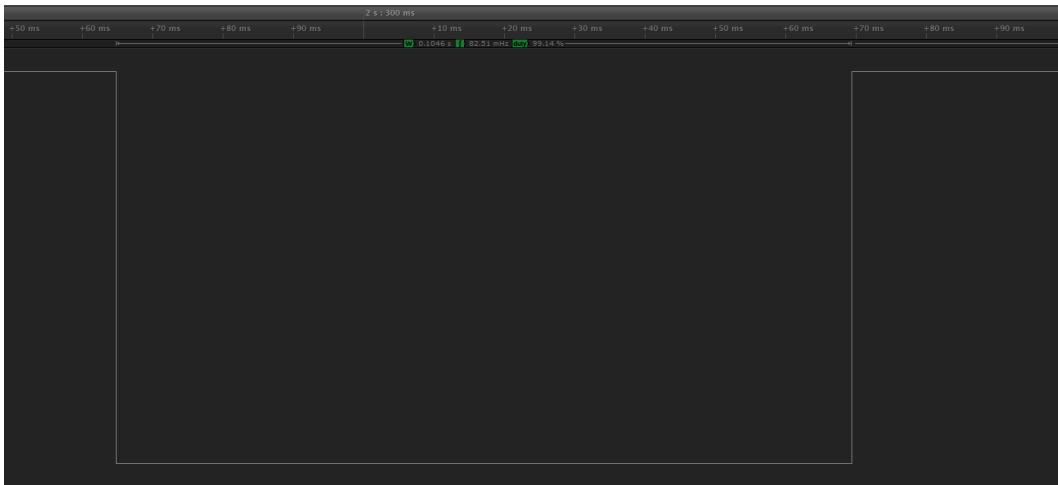
**Umgehung des Kernel Panic** Da der Kernel Panic durch ein Non-Maskable Interrupt (**NMI**) (siehe [A.1](#)) auf niedriger Ebene ausgelöst wird, kann ein Triggern nicht einfach verhindert werden. Jedoch kann die aufgerufene Subroutine angepasst werden, so zu sehen in Zeile 1 bis 18 des Patches. Normalerweise wird in dieser Subroutine das Exception Syndrome Register (**ESR**) der ARM64 **CPU** mittels Bit-Operationen auf fatale Reliability Availability and Serviceability (**RAS**) oder non-**RAS** Fehler untersucht und bei Bedarf mit dem Aufruf `arm64_serror_panic()` ein Notstopp des Bootvorgangs getriggert. Der Wert des **ESR** ist bei allen Versuchen immer 3204448258 in Dezimal oder 0xBF000002 in Hexadezimal beziehungsweise 0x2F in Hexadezimal nach dem dekodierenden Bit-Schieben nach rechts gewesen. Wie bereits vermutet signalisiert dieser Wert einen SError (System Error) Interrupt an die **CPU**. Um einen Kernel Panic nun zu verhindern, wird eine If-Abfrage in die Subroutine eingesetzt welche das **ESR** auf einen SError checkt und in diesem Fall eine Debug Meldung ausgibt und mit `return`; die Subroutine verlässt bevor ein Abbruch beauftragt werden kann [1].

**Cortex-A53 Hardware Bug** Nach dem oben beschriebenen Fix fährt der **SBC** nun erfolgreich hoch mit installiertem **PCIe** Adapterboard hoch. Jedoch ist dies nicht immer der Fall, in zirka 2 von 3 Fällen gibt es immer noch einen Kernel Panic. Auffällig ist hierbei dass dies nur mit **CPU** 0 bis **CPU** 3 der Fall ist. Grund dafür ist, dass die kleinen ARM Cortex-A53 Kerne des **SoC** den Hardware Bug anders handhaben und statt eines SError Interrupt einen ”Synchronous External Abort” erzeugen. Dem Autor ist nicht bekannt ob und wie diese Art von Fehlerfall analog zu den großen Cortex-A72 Kernen behandelt werden kann und ist der Meinung, dass das Umgehen von Interrupt Serviceroutinen bei Hardware Fehlern ohnehin eine suboptimale Lösung ist. Ein manuelles `taskset -c 4 modprobe pcie_rockchip_host` hat hingegen nicht funktioniert. Daher entschied man sich stattdessen die Ausführung des Exception-auslösenden Codes auf dem Kern 5 zu erzwingen. Konkret wird die Bibliothek `linux/smp.h` genutzt um die Funktion `rockchip_pcie_rd_other_conf()` mittels dem Aufruf `smp_call_function_single()` aus `rockchip_pcie_rd_conf()` heraus zu starten. Dazu wurde der eigentliche Aufruf umgeschrieben und die Funktionsdefinition angepasst, da zur Übergabe von Funktionspara-

metern und Ergebnissen nur ein void-Pointer erlaubt ist. Das `struct pcie_argument_carrier_castable` wird von der aufrufenden Funktion hierzu mit Werten vorbereitet, in einen void-Pointer gecastet, übergeben, und von der aufgerufenen Funktion in ein struct zurück gecastet und die Wertefelder gelesen sowie beschrieben. Es ist zu beachten, dass das vierte Argument von `smp_call_function_single()` als Wait-Argument auf `True` gesetzt sein sollte damit der aufrufende Prozess auf den Abschluss der aufgerufenen Funktion wartet. Auch ist zu beachten, dass innerhalb der aufgerufenen Funktion `get_cpu()` beziehungsweise `put_cpu()` für eine nicht-preemptive Code-Ausführung sorgen. Diese Art des Scheduling sorgt dafür, dass der aktuelle Prozess nur unterbrochen werden kann, wenn er abgeschlossen ist oder in einen wartenden Zustand übergeht [29]. Die Zyklen auf der `CPU` muss sich dieser Prozess somit nicht teilen, was in diesem Fall durchaus nützlich erscheint. Negative Auswirkungen auf die Performance oder übermäßig lange Bootvorgänge konnten nicht festgestellt werden. Die Idee zu diesem und dem vorherigen Bugfix stammen aus einer Diskussion in Quelle [20].

**Fehlende PERST# Leitung auf dem PCIe Bus** Nun fährt das System in 100% der Fälle korrekt hoch und die `PCIe` Brücke wird korrekt erkannt, jedoch nicht die `NPU` Module. Zu der Lösung für dieses Problem wurde der Autor durch einen Github Nutzer gebracht, der Patches für den ähnlichen RockPro64 `SBC` des Herstellers PINE64 anwendet um korrekte `PCIe` Funktionalität für seine Anwendungen zu gewährleisten [18]. Dabei fiel dem Autor auf, dass in der Spezifikation des Busses beim Hochfahren ein 100 ms langer aktiver PERST# Puls (active low) gefordert wird um definiertes Power-on-Reset (`POR`) Verhalten zu garantieren. Vermutlich sind vor allem Geräte hinter einer `PCIe` Brücke von dieser Maßnahme abhängig, da diese zeitlich verzögert hochfahren. Jedenfalls bietet der NanoPi M4V2 keine PERST# Leitung sondern, wie bereits erwähnt, einen Pull-up Widerstand. Der Autor löste also ein Kabel an diesen Widerstand welches mit einem `GPIO` Pin des `SBC` verbunden wird um durch diesen einen Reset Puls zu erzwingen. Das Kabel terminiert in einer Y-Spaltung mit zwei Steckern, wobei eines davon für Debugging Zwecke vorhanden ist. Für die Emulation wurde `GPIO` Pin `GPIO1_C2` gewählt da dieser 3,3 Volt kompatibel ist und nicht doppelt belegt wurde. Pin `GPIO2_D4`, welcher der RockPro64 für diese Funktion verwendet, wurde nicht gewählt da dieser im NanoPi M4V2 ein MIPI `CSI-2` Steuersignal bereitstellt. `GPIO1_C2` übersetzt sich nach Schema [34] zu Pinnummer 50. Nach Tests mit kleinen, selbst geschriebenen Linux Kernel Modulen wurde auch folgende Anpassung in den Patch integriert. Die Anpassung, in Zeile 162 bis 233, umfasst die Initialisierung des `GPIO` Ports, das Schreiben der logischen 0 und 1 nach einer 100 ms Wartezeit, eine weitere 500 ms Wartezeit vor dem Link Training und das Freigeben der Ressourcen am Schluss. Alle Pfade, welche die `PCIe` Modul Eintrittsfunktion `rockchip_pcie_host_init_port()` verlassen, sollten garantieren dass angeforderte Ressourcen auch wieder freigegeben werden. Darum wird in Zeile 190 die `GPIO` Ressource bei fehlerhaftem Aufruf freigesetzt, gleiches auch an weiteren Stellen. Es bietet sich gegebenenfalls noch an den Pin vor der Freigabe als Input, also High-Z zu konfigurieren um Beschädigungen des `SoC` zu vermeiden. Für alle Operationen wird die Bibliothek `linux/gpio/consumer.h` verwendet. Die Verwendung von `linux/gpio.h` wurde wegen

Compiler Fehlern aufgrund von Abhängigkeiten unterlassen. Abbildung ... ist ein Screenshot des emulierten PERST# Signal, aufgezeichnet mit einem Logic Analyzer. Durch Code Ausführung zwischen beiden Pin-Schreiboperationen sind, vor allem bei installierten PCIe Geräten, auch Pulse mit einer Länge von über 130 ms zu beobachten. Dies hat allerdings keinen negativen Einfluss auf das POR Verhalten.



**Abb. 2.6:** Emuliertes Reset Signal Logic Analyzer Aufzeichnung

## 2.2.4 Tests

Nachdem alle Bugfixes des vorherigen Kapitels angewendet wurden, können alle notwendigen Apex und gasket-dkms Kernel Treiber sowie Laufzeitpakete nach offizieller Anleitung installiert werden. Optional bietet sich auch an, in zukünftigen Builds des Betriebssystems diese Treiber direkt in der Kernel Konfiguration zu integrieren. War die Installation erfolgreich sollte der Befehl `lspci -nn` folgendes Resultat anzeigen:

**List. 2.2:** lspci mit erkannten Geräten

```

1 nanopim4v2:~:% lspci -nn
2 00:00.0 PCI bridge [0604]: Fuzhou Rockchip Electronics Co., Ltd
   RK3399 PCI Express Root Port [1d87:0100]
3 01:00.0 PCI bridge [0604]: ASMedia Technology Inc. Device
   [1b21:1182]
4 02:03.0 PCI bridge [0604]: ASMedia Technology Inc. Device
   [1b21:1182]
5 02:07.0 PCI bridge [0604]: ASMedia Technology Inc. Device
   [1b21:1182]
6 03:00.0 System peripheral [0880]: Global Unichip Corp. Coral Edge
   TPU [1ac1:089a]
7 04:00.0 System peripheral [0880]: Global Unichip Corp. Coral Edge
   TPU [1ac1:089a]
```

Anschließend können die bereitgestellten Tests ausgeführt werden. Interessant ist hier die Bilderkennungsdemo, welche ein vergleichsweise großes Inception V3 Model nutzt, das zweigeteilt im Pipeline Modus ausgeführt wird. Aufruf und Ergebnis sehen folgendermaßen aus:

**List. 2.3:** Coral TPU Pipelining Demo

```

1 nanopim4v2:pycoral:% python3
2     examples/model_pipelining_classify_image.py \
3         --models \
4             test_data/pipeline/inception_v3_299_quant_segment_%d_of_2\
5                 _edgetpu.tflite \
6             --labels test_data/imagenet_labels.txt \
7             --input test_data/parrot.jpg
8
9 Using devices:  ['pci:0', 'pci:1']
10 Using models:  ['test_data/pipeline/inception_v3_299_quant_segment_0_of_2_edgetpu.tflite', 'test_data/pipeline/inception_v3_299_quant_segment_1_of_2_edgetpu.tflite']
11 WARNING: Logging before InitGoogleLogging() is written to STDERR
12 I20230220 19:33:07.823565 3813 pipelined_model_runner.cc:172]
13     Thread: 281472822870496 receives empty request
14 I20230220 19:33:07.823628 3813 pipelined_model_runner.cc:245]
15     Thread: 281472822870496 is shutting down the pipeline...
16 I20230220 19:33:08.005640 3813 pipelined_model_runner.cc:255]
17     Thread: 281472822870496 Pipeline is off.
18 I20230220 19:33:08.006318 3814 pipelined_model_runner.cc:207]
19     Queue is empty and `StopWaiters()` is called.
20 -----RESULTS-----
21 macaw: 0.99609
22 Average inference time (over 5 iterations): 37.3ms
23 I20230220 19:33:08.006780 3777 pipelined_model_runner.cc:172]
24     Thread: 281473140535312 receives empty request
25 E20230220 19:33:08.006809 3777 pipelined_model_runner.cc:240]
26     Thread: 281473140535312 Pipeline was turned off before.
27 Exception ignored in: <function PipelinedModelRunner.__del__ at
28     0xfffff9082f8b0>
29 Traceback (most recent call last):
30     File "/usr/lib/python3/dist-packages/pycoral/pipeline/
31         pipelined_model_runner.py", line 83, in __del__
32         self.push({})
33     File "/usr/lib/python3/dist-packages/pycoral/pipeline/
34         pipelined_model_runner.py", line 152, in push
35         self._runner.Push(input_tensors)
36 RuntimeError: Pipeline was turned off before.
37 E20230220 19:33:08.007568 3777 pipelined_model_runner.cc:240]
38     Thread: 281473140535312 Pipeline was turned off before.

```

```
32 E20230220 19:33:08.007594 3777 pipelined_model_runner.cc:147]
Failed to shutdown status: INTERNAL: Pipeline was turned off
before.
```

Ignoriert man die Fehlermeldungen, welche Bugs der Demo geschuldet sind, ist die Bilderkennung erfolgreich verlaufen. Der Papagei wurde mit 99,6% Übereinstimmung klassifiziert. Eine ähnliche Demo, welche nur eine NPU und das kleinere MobileNetV2 verwendet, erreicht in der selben Klassifizierungsaufgabe nur 75,8% Confidence. Dies zeigt bereits das Potential für duale NPU Anwendungen. Die MobileNetV2 Demo schließt eine Inferenz innerhalb von 2,8 ms ab. Dies ist mindestens genau so schnell, wie die Vorgabe der offiziellen Anleitung [8].

## 2.2.5 Verbesserungspotential

Verbesserungspotential sieht der Autor, neben bereits genannten Punkten, vor allem in der Aufbesserung des Codes. Kommentare könnten reduziert, Meldungen auf das geringste Log Level (`pr_info()` zu `pr_debug()`) heruntergestuft werden. Es bietet sich ebenfalls an, Anstrengungen für eine Integration in den offiziellen Kernel (Upstream) zu tätigen. Dies würde zukünftige Unannehmlichkeiten für anderen Nutzer vermeiden. Jedoch wurde bei dieser Arbeit kein Werte auf Konformität mit jeglichen Coding Conventions gegeben, von daher ist eine direkte Integration unrealistisch.

# 3 Kamera Sektion

Die Kamerasektion konnte zum jetzigen Zeitpunkt noch nicht vollständig abgeschlossen werden. Da ein weiteres Kapitel wie die [NPU](#) Sektion jedoch ohnehin den Rahmen dieser Arbeit sprengen würde, wird nur kurz beleuchtet was bisher erreicht wurde, und was noch folgt.

## 3.1 Hardware

Für das Anbinden der Raspberry Pi V2 Kameras an den NanoPi M4V2 wurde ein Adapter [PCB](#) entwickelt und gefertigt. Es adaptiert mechanisch und elektrisch vom 30-Pin [FPC](#) Stecker des [SBC](#) auf den 15-Pin [FPC](#). Level Shifter konvertieren 1,8 Volt Steuersignale zu 3,3 Volt Steuersignalen und Multiplexer erlauben es [I2C](#) und Steuersignale für Debugging Zwecken von extern einzuspeisen. Das Board generiert auch eine 3,3 Volt Versorgungsspannung für die Kamera. Da die Platine der Kamera bereits alle weiteren Spannungen für den Sensor und den MCLK Takt selbst erzeugt, war sonst nicht mehr notwendig. Beim Layout musste beachtet werden, dass MIPI [CSI-2](#) als elektrischer Standard Taktsynchron ist. Somit mussten alle differentiellen Paare, also zwei Daten- und ein Taktpaar mittels Längenanpassung aufeinander abgestimmt werden. Eine Differentielle Impedanz von 100 Ohm und musste trotz Vias entlang der Signalleitungen möglichst eingehalten werden. Vias waren notwendig, da ein Signalpaar über Kreuz verlegt werden musste um die Position mit einem anderen zu tauschen. Ob diese Maßnahme die Signalintegrität zum Punkt der Unbrauchbarkeit beeinflusst, ist noch unklar. Für zukünftige Versionen der Adapterplatine bietet sich eventuell die Verwendung eines MIPI DPHY Retimer IC an, welcher zeitliche Abweichungen und Amplitudenverlust teilweise ausbessern kann. Im Anhang [A.3](#) befindet sich das Schematic des Kamera Adapters. Kommunikation über den [I2C](#) Bus mit der Kamera findet beim Hochfahren statt, so viel konnte der Autor mittels Logic Analyzer in Erfahrung bringen.

## 3.2 Software

Bisher konnten leider keine Bilder vom Kameramodul abgegriffen werden. Es wurde das libcamera Framework installiert und `cam -1` aufzurufen erbrachte jede glich dass keine Kameras erkannt werden [16]. Die Idee aktuell lautet nun, den [ISP](#) zunächst mittels Device

Tree einzubinden, da dieser dort bisher nicht zu finden ist. Als Vorlage soll ein Patch für den RockPi 4 dienen [42]. Gleichzeitig macht es Sinn alle Spannungsregulatoren auf dem **SBC**, welche für den Betrieb des **ISP** notwendig sind, im Device Tree gegen zu prüfen.

# 4 Schluss

Abschließend soll ein Ausblick und ein Fazit gegeben werden.

## 4.1 Ausblick

Verbesserungspotential wurde bereits genug in den jeweiligen Kapiteln erwähnt. Als Ausblick sollen stattdessen noch alternative Komponenten erwähnt werden. Offensichtlich zu erwähnen ist, dass der Rockchip RK3399 keineswegs mehr ein moderner **SoC** ist. Mit relativ alten Kernen, Fehlern in der Hardware und Herstellung im 28nm Prozess wirkt dieser veraltet. Performance und Hitzeentwicklung tun dazu ihr übriges. Nachfolger wie der Rockchip RK3566, RK3568 und RK3588 bieten mehr Leistung, bessere Effizienz, modernere und zahlreichere I/O und eine integrierte **NPU**. Ob die Software Unterstützung von letzterer ausreicht, ist, mangels Informationen, eine offene Frage.

Auch auf Seite der Kamera gibt es durchaus Alternativen wie die jüngst eingeführte Raspberry Pi V3 Kamera. Sie ist elektrisch zum Vorgänger kompatibel, bietet aber unter anderem eine höhere Auflösung, Autofokus, größere Pixel, breitere **FoV**, größere Blende und mehr Dynamikumfang. Viele weitere Kamerasensoren sind ebenfalls attraktiv für den Autor und sollte die Unterstützung eben dieser mit dem **SoC** der Wahl schwierig werden, steht noch die Option im Raum auf FPGAs als **ISP** zurückzugreifen.

Für die **NPU** gibt es inzwischen zahlreiche Alternativen auf dem, immer wachsenden, Markt. Beschleuniger für Inferenz Operationen, welche aber auch gute Software liefern, zu finden ist aber keine einfache Aufgabe. Konkret interessant ist beispielsweise das Hailo-8 Starterkit des Herstellers Hailo. Dessen Beschleuniger wirbt mit einer Leistung von 26 **TOPS** bei einem Verbrauch von 5 Watt, verwendet einen ähnlichen M.2 Formfaktor wie die Coral Edge TPU und ist mit zirka 180 USD noch vergleichsweise erschwinglich. Nähere Details wurden aber nicht betrachtet und von daher sind alle Angaben mit Vorsicht zu genießen. Für den Anfang wird die Coral Dual Edge TPU weiterhin verwendet.

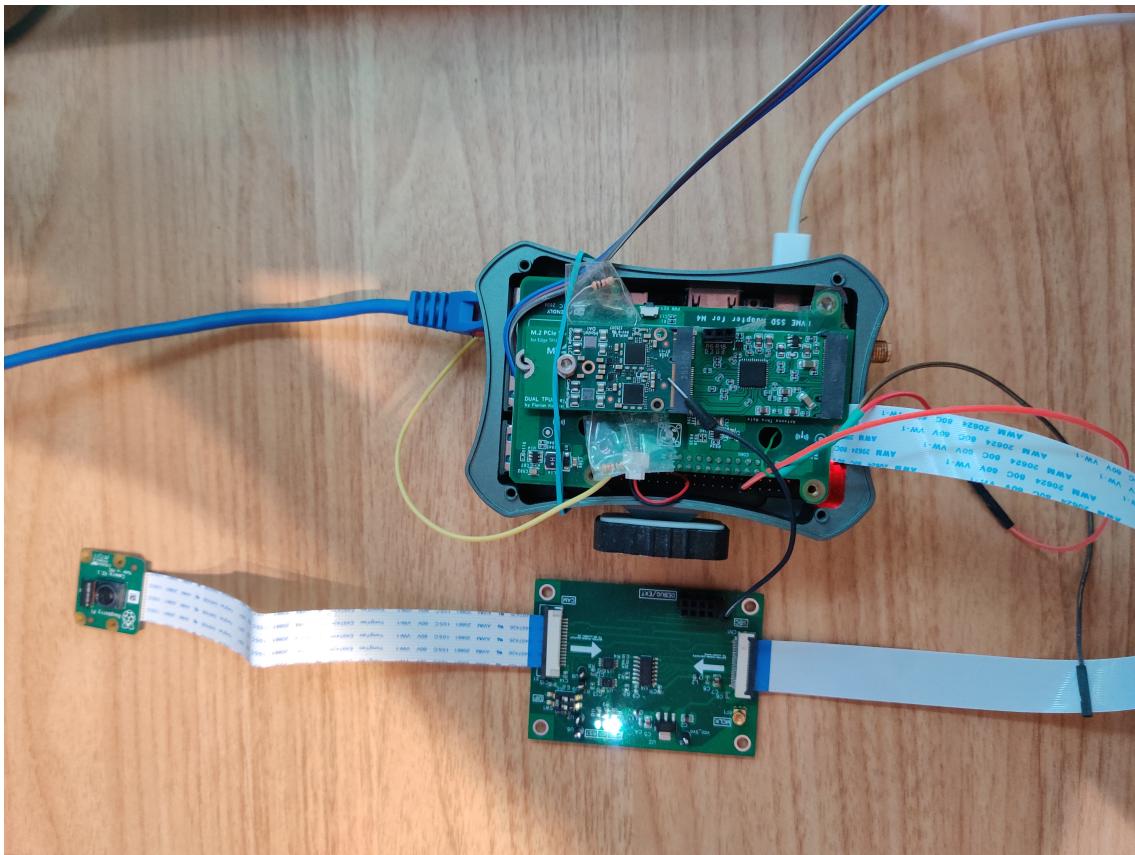
## 4.2 Fazit

Insgesamt ist der Autor bisher zufrieden mit seiner Arbeit und hat einiges mitgenommen. Wissen, Ansätze und Debugging Methoden welche hier erlernt wurden können in zukünftigen Projekten immer von Nutzen sein. Das Projekt beweist, dass auch relativ unbekannte

SBC Plattformen mit entsprechender Arbeit für die gewünschten Anwendungsszenarien vorbereitet werden können. Diese Erkenntnis ist vor allem in Zeiten Chipkrise sehr hilfreich, denn, wie bereits beleuchtet, sind bekannte Plattformen wie der Raspberry Pi teilweise entweder nicht verfügbar oder nur sehr teuer zu erstehen und Alternativen daher gern gesehen.

Jedoch ist der Autor der Meinung, dass große Teile der Arbeit leicht hätten umgehen werden können, hätte er sich für einen anderen der zahlreichen SBCs auf dem Markt entschieden. Ein Beispiel ist der RockPro64 von PINE64, welcher den gleichen SoC nutzt, aber eine bessere Implementation des PCIe Interfaces hat und allgemein in der Rockchip Community verbreiteter erscheint. Daher gibt es für diesen SBC bereits deutlich mehr Patches und Diskussionen.

Trotzdem alledem hat der NanoPi M4V2 als Plattform immer noch Vorteile und der Autor möchte weiterhin Arbeit in dieses Projekt investieren. Er hofft auch dass der Leser einige nützliche Informationen mitnehmen konnte. Zum Ende noch ein Foto des bisherigen Aufbau mit Kamera- und PCIe-Platine und SBC im Gehäuse:



**Abb. 4.1:** Foto des bisherigen Aufbau

# A Kapitel im Anhang

```
1 [    3.223488] rockchip-pcie f8000000.pcie: host bridge /pcie@f8000000
2 ranges:
3 [    3.228305] dwmmc_rockchip fe320000.mmc: Got CD GPIO
4 [    3.232147] rockchip-pcie f8000000.pcie:      MEM 0x00fa000000..0
5 x00fbfffff -> 0x00fa00000
6 [    3.259806] mmc_host mmc1: Bus speed (slot 0) = 400000Hz (slot req
7 400000Hz, actual 400000HZ div = 0)
8 [    3.268876] rockchip-pcie f8000000.pcie:      IO 0x00fbe00000..0
9 x00fbefffff -> 0x00fbe00000
10 [   3.303467] rockchip-pcie f8000000.pcie: no bus scan delay, default
11 to 0 ms
12 [   3.308474] rockchip-pcie f8000000.pcie: no vpcie12v regulator found
13 [   3.313315] rockchip-pcie f8000000.pcie: no vpcie3v3 regulator found
14 [   3.342106] mmc_host mmc0: Bus speed (slot 0) = 14850000Hz (slot req
15 150000000Hz, actual 148500000HZ div = 0)
16 [   3.342916] rockchip-pcie f8000000.pcie: wait 0 ms (from device tree)
17 before bus scan
18 [   3.358938] rockchip-pcie f8000000.pcie: PCI host bridge to bus
19 0000:00
20 [   3.363884] pci_bus 0000:00: root bus resource [bus 00-1f]
21 [   3.368633] pci_bus 0000:00: root bus resource [mem 0xfa000000-0
22 xfbfffff]
23 [   3.373492] pci_bus 0000:00: root bus resource [io 0x0000-0xffff] (
24 bus address [0xfbbe00000-0xfbeffff])
25 [   3.378688] pci 0000:00:00.0: [1d87:0100] type 01 class 0x060400
26 [   3.383741] pci 0000:00:00.0: supports D1
27 [   3.388406] pci 0000:00:00.0: PME# supported from D0 D1 D3hot
28 [   3.398310] pci 0000:00:00.0: bridge configuration invalid ([bus
00-00]), reconfiguring
29 [   3.403587] pci 0000:01:00.0: [1b21:1182] type 01 class 0x060400
30 [   3.408616] pci 0000:01:00.0: Max Payload Size set to 256 (was 128,
31 max 256)
32 [   3.413524] pci 0000:01:00.0: enabling Extended Tags
33 [   3.418471] pci 0000:01:00.0: PME# supported from D0 D3hot D3cold
34 [   3.437383] pci 0000:01:00.0: bridge configuration invalid ([bus
00-00]), reconfiguring
35 [   3.442598] SError Interrupt on CPU5, code 0xbff000002 -- SError
36 [   3.442611] CPU: 5 PID: 7 Comm: kworker/u12:0 Not tainted 5.15.25-
37 rockchip64 #22.02.1
38 [   3.442622] Hardware name: FriendlyElec NanoPi M4 Ver2.0 (DT)
39 [   3.442629] Workqueue: events_unbound deferred_probe_work_func
40 [   3.442649] pstate: 600000c5 (nZCv daIF -PAN -UAO -TC0 -DIT -SSBS
41 BTYPE=--)
```

```

29 [ 3.442660] pc : rockchip_pcie_rd_conf+0xb4/0x230
30 [ 3.442676] lr : rockchip_pcie_rd_conf+0x78/0x230
31 [ 3.442686] sp : ffff800009dfb730
32 [ 3.442690] x29: ffff800009dfb730 x28: 0000000000000000 x27:
0000000000000001
33 [ 3.442706] x26: 0000000000000000 x25: 0000000000000000 x24:
fffff800009dfb874
34 [ 3.442720] x23: ffff000005b40000 x22: 0000000000000000 x21:
0000000000000004
35 [ 3.442733] x20: ffff800009dfb7b4 x19: 0000000000000000 x18:
0000000000000000
36 [ 3.442745] x17: 6572202c295d3030 x16: 0000000000000000 x15:
ffffffffffff
37 [ 3.442758] x14: ffff8000094e4468 x13: ffff000005b18a1c x12:
fffff000005b1829d
38 [ 3.442771] x11: 0000000000000028 x10: 0000000000000001 x9 :
0000000011821b21
39 [ 3.442785] x8 : 000000000000ea60 x7 : ffff800009dfb700 x6 :
0000000000000001
40 [ 3.442797] x5 : 0000000000000003 x4 : 0000000000000000 x3 :
0000000000c00008
41 [ 3.442810] x2 : 00000000080000b x1 : ffff800011c00008 x0 :
fffff800011c0000c
42 [ 3.442825] Kernel panic - not syncing: Asynchronous SError Interrupt
43 [ 3.442831] CPU: 5 PID: 7 Comm: kworker/u12:0 Not tainted 5.15.25-
rockchip64 #22.02.1
44 [ 3.442840] Hardware name: FriendlyElec NanoPi M4 Ver2.0 (DT)
45 [ 3.442845] Workqueue: events_unbound deferred_probe_work_func
46 [ 3.442855] Call trace:
47 [ 3.442858] dump_backtrace+0x0/0x200
48 [ 3.442875] show_stack+0x18/0x28
49 [ 3.442887] dump_stack_lvl+0x68/0x84
50 [ 3.442900] dump_stack+0x18/0x34
51 [ 3.442910] panic+0x164/0x35c
52 [ 3.442917] nmi_panic+0x64/0x98
53 [ 3.442926] arm64_serror_panic+0x64/0x78
54 [ 3.442934] do_serror+0x34/0x80
55 [ 3.442941] el1h_64_error_handler+0x34/0x50
56 [ 3.442953] el1h_64_error+0x74/0x78
57 [ 3.442961] rockchip_pcie_rd_conf+0xb4/0x230
58 [ 3.442972] pci_bus_read_config_dword+0x84/0xd8
59 [ 3.442983] pci_bus_generic_read_dev_vendor_id+0x34/0x1b0
60 [ 3.442995] pci_bus_read_dev_vendor_id+0x4c/0x70
61 [ 3.443005] pci_scan_single_device+0x84/0xd8
62 [ 3.443016] pci_scan_slot+0x38/0x120
63 [ 3.443026] pci_scan_child_bus_extend+0x58/0x330
64 [ 3.443037] pci_scan_bridge_extend+0x340/0x5a0
65 [ 3.443048] pci_scan_child_bus_extend+0x1fc/0x330
66 [ 3.443059] pci_scan_bridge_extend+0x340/0x5a0
67 [ 3.443069] pci_scan_child_bus_extend+0x1fc/0x330
68 [ 3.443080] pci_scan_root_bus_bridge+0xd4/0xf0
69 [ 3.443092] pci_host_probe+0x18/0xb0

```

```

70 [ 3.443102]    rockchip_pcidev_probe+0x4a8/0x4c0
71 [ 3.443113]    platform_probe+0x68/0xd8
72 [ 3.443126]    really_probe+0xbc/0x428
73 [ 3.443134]    __driver_probe_device+0x114/0x188
74 [ 3.443143]    driver_probe_device+0xb0/0x110
75 [ 3.443152]    __device_attach_driver+0x98/0x130
76 [ 3.443161]    bus_for_each_drv+0x7c/0xd0
77 [ 3.443169]    __device_attach+0xe8/0x168
78 [ 3.443178]    device_initial_probe+0x14/0x20
79 [ 3.443186]    bus_probe_device+0x9c/0xa8
80 [ 3.443195]    deferred_probe_work_func+0x9c/0xf0
81 [ 3.443203]    process_one_work+0x20c/0x4c8
82 [ 3.443216]    worker_thread+0x208/0x478
83 [ 3.443227]    kthread+0x138/0x150
84 [ 3.443239]    ret_from_fork+0x10/0x20
85 [ 3.443249] SMP: stopping secondary CPUs
86 [ 3.443262] Kernel Offset: disabled
87 [ 3.443265] CPU features: 0x10001051,20000846
88 [ 3.443272] Memory Limit: none
89 [ 3.759760] ---[ end Kernel panic - not syncing: Asynchronous SError
Interrupt ]---
90 [ 3.442676] lr : rockchip_pcidev_rd_conf+

```

List. A.1: PCIe Adapter Kernel Panic Auszug

```

1 diff --git a/arch/arm64/kernel/traps.c b/arch/arm64/kernel/traps.c
2 index 0529fd575..b8d92a850 100644
3 --- a/arch/arm64/kernel/traps.c
4 +++ b/arch/arm64/kernel/traps.c
5 @@ -946,6 +946,13 @@ bool arm64_is_fatal_ras_error(struct pt_regs
6     *regs, unsigned int esr)
7
7 void do_error(struct pt_regs *regs, unsigned int esr)
8 {
9     /* ignore SError to enable rk3399 PCIe bus enumeration */
10    if (esr >> ESR_ELx_EC_SHIFT == ESR_ELx_EC_SERROR) {
11        pr_debug("ignoring SError Interrupt on CPU%d\n",
12            smp_processor_id());
13        return;
14    }
15
16    /* non-RAS errors are not containable */
17    if (!arm64_is_ras_error(esr) || arm64_is_fatal_ras_error(regs,
18        esr))
19        arm64_serror_panic(regs, esr);
20 diff --git a/drivers/pci/controller/pcie-rockchip-host.c b/drivers
21     /pci/controller/pcie-rockchip-host.c
22 index 5c74bf5e2..bf7ba8b07 100644
23 --- a/drivers/pci/controller/pcie-rockchip-host.c
24 +++ b/drivers/pci/controller/pcie-rockchip-host.c

```

```
23 @@ -36,10 +36,17 @@
24 #include <linux/platform_device.h>
25 #include <linux/reset.h>
26 #include <linux/regmap.h>
27+#include <linux/printk.h>
28+#include <linux/smp.h>
29
30 #include "../pci.h"
31 #include "pcie-rockchip.h"
32
33 //CPU ID to force PCIe RD/WR to run on for catchable interrupt
34 +#define FORCE_CPU_ID (5)
35 //GPIO Pin Global Number for emulated PCIe PERST# Signal (
36     GPIO1_C2) on RK3399
37+#define EMU_GPIO_PCI_PERST (50)
38+
39 static int bus_scan_delay = -1;
40 module_param_named(bus_scan_delay, bus_scan_delay, int, S_IRUGO);
41
42 @@ -104,6 +111,20 @@ static u8 rockchip_pcie_lane_map(struct
43     rockchip_pcie *rockchip)
44     return map;
45 }
46
47 //Carrier struct for function rockchip_pcie_rd_other_conf(),
48 //castable to void*
49+struct pcie_argument_carrier_castable
50+{
51     //Input Parameters
52     struct rockchip_pcie *rockchip;
53     struct pci_bus *bus;
54     u32 devfn;
55     int where;
56     int size;
57     u32 *val;
58     //Return Value
59     int return_value;
60 };
61+
62 static int rockchip_pcie_rd_own_conf(struct rockchip_pcie *
63     rockchip,
64             int where, int size, u32 *val)
65 {
66 @@ -157,37 +178,51 @@ static int rockchip_pcie_wr_own_conf(struct
67     rockchip_pcie *rockchip,
68     return PCIBIOS_SUCCESSFUL;
```

```

64 }
65
66 -static int rockchip_pcie_rd_other_conf(struct rockchip_pcie *
67 -    rockchip,
68 -    struct pci_bus *bus, u32 devfn,
69 -    int where, int size, u32 *val)
70 +static void rockchip_pcie_rd_other_conf(void *arguments)
71 {
72 + //Check if CPU ID is 5
73 + //Also force deactivated Preemptive Execution
74 + int this_cpu = get_cpu();
75 + pr_info("[PCI Subsystem] rockchip_pcie_rd_other_conf() called
76 +     with CPU Core: %d \n", this_cpu);
77 + //Extract Function Arguments, cast to argument struct pointer
78 + struct pcie_argument_carrier_castable *arg_ptr_casted;
79 + arg_ptr_casted = (struct pcie_argument_carrier_castable*)
80 +     arguments;
81 +
82     void __iomem *addr;
83
84 - addr = rockchip->reg_base + PCIE_ECAM_OFFSET(bus->number, devfn,
85 -     where);
86 + addr = arg_ptr_casted->rockchip->reg_base + PCIE_ECAM_OFFSET(
87 +     arg_ptr_casted->bus->number,
88 +     arg_ptr_casted->devfn, arg_ptr_casted->where);
89
90 - if (!IS_ALIGNED((uintptr_t)addr, size)) {
91 -     *val = 0;
92 -     return PCIBIOS_BAD_REGISTER_NUMBER;
93 + if (!IS_ALIGNED((uintptr_t)addr, arg_ptr_casted->size))
94 + {
95 +     *(arg_ptr_casted->val) = 0;
96 +     arg_ptr_casted->return_value = PCIBIOS_BAD_REGISTER_NUMBER;
97 +     put_cpu();
98 +     return;
99 }
100
101 - if (pci_is_root_bus(bus->parent))
102 -     rockchip_pcie_cfg_configuration_accesses(rockchip,
103 + if (pci_is_root_bus(arg_ptr_casted->bus->parent))
104 +     rockchip_pcie_cfg_configuration_accesses(arg_ptr_casted->
105         rockchip,
106             AXI_WRAPPER_TYPE0_CFG);
107 else
108 -     rockchip_pcie_cfg_configuration_accesses(rockchip,
109 +     rockchip_pcie_cfg_configuration_accesses(arg_ptr_casted->

```

```

rockchip,
    AXI_WRAPPER_TYPE1_CFG);

104
105
106 - if (size == 4) {
107 -     *val = readl(addr);
108 - } else if (size == 2) {
109 -     *val = readw(addr);
110 - } else if (size == 1) {
111 -     *val = readb(addr);
112 + if (arg_ptr_casted->size == 4) {
113 +     *(arg_ptr_casted->val) = readl(addr);
114 + } else if (arg_ptr_casted->size == 2) {
115 +     *(arg_ptr_casted->val) = readw(addr);
116 + } else if (arg_ptr_casted->size == 1) {
117 +     *(arg_ptr_casted->val) = readb(addr);
118 } else {
119 -     *val = 0;
120 -     return PCIBIOS_BAD_REGISTER_NUMBER;
121 +     *(arg_ptr_casted->val) = 0;
122 +     arg_ptr_casted->return_value = PCIBIOS_BAD_REGISTER_NUMBER;
123 +     put_cpu();
124 +     return;
125 }
126 - return PCIBIOS_SUCCESSFUL;
127 + arg_ptr_casted->return_value = PCIBIOS_SUCCESSFUL;
128 + put_cpu();
129 + return;
130 }

131
132 static int rockchip_pcie_wr_other_conf(struct rockchip_pcie *
133     rockchip,
134 @@ -231,8 +266,26 @@ static int rockchip_pcie_rd_conf(struct
135     pci_bus *bus, u32 devfn, int where,
136     if (pci_is_root_bus(bus))
137         return rockchip_pcie_rd_own_conf(rockchip, where, size, val);
138
139 - return rockchip_pcie_rd_other_conf(rockchip, bus, devfn, where,
140 -     size,
141 -     val);
142 + //Call rockchip_pcie_rd_other_conf() on CPU ID 5
143 + //Prepare argument carrier...
144 + struct pcie_argument_carrier_castable arg_carrier;
145 + arg_carrier.rockchip = rockchip;
146 + arg_carrier.bus = bus;
147 + arg_carrier.devfn = devfn;
148 + arg_carrier.where = where;

```

```

146 + arg_carrier.size = size;
147 + arg_carrier.val = val;
148 + void *args_ptr = (void*)&arg_carrier;
149 + //...and execute
150 + int ret = smp_call_function_single(FORCE_CPU_ID,
151     rockchip_pcie_rd_other_conf, args_ptr, 1);
152 + //Handle Possible Error
153 + if(ret)
154 + {
155 +     pr_info("[PCI Subsystem] ERROR: Cannot run function
156 +         rockchip_pcie_rd_other_conf() on specified core! \n");
157 +     return -1;
158 + }
159 +
160
161 static int rockchip_pcie_wr_conf(struct pci_bus *bus, u32 devfn,
162 @@ -301,11 +354,32 @@ static int rockchip_pcie_host_init_port(
163     struct rockchip_pcie *rockchip)
164     int err, i = MAX_LANE_NUM;
165     u32 status;
166
167 - gpiod_set_value_cansleep(rockchip->ep_gpio, 0);
168 + //Create GPIO Descriptor for emulated PCIe PERST#
169 + struct gpio_desc *perst_desc;
170 + perst_desc = gpio_to_desc(EMU_GPIO_PCI_PERST);
171 + if(perst_desc == NULL)
172 + {
173 +     pr_info("[PCI Subsystem] ERROR: GPIO %d Descriptor could
174 +         not be created!\n", EMU_GPIO_PCI_PERST);
175 +     return -1;
176 + }
177 + //Set GPIO Descriptor Direction to Output
178 + if(gpiod_direction_output(perst_desc, 0))
179 + {
180 +     pr_info("[PCI Subsystem] ERROR: GPIO %d Setting Output
181 +         Direction not successful!\n", EMU_GPIO_PCI_PERST);
182 +     gpiod_put(perst_desc);
183 +     return -1;
184 + }
185 + gpiod_set_value_cansleep(perst_desc, 0); //Just to be sure it's
186     logical 0
187 + //gpiod_set_value_cansleep(rockchip->ep_gpio, 0);
188 +

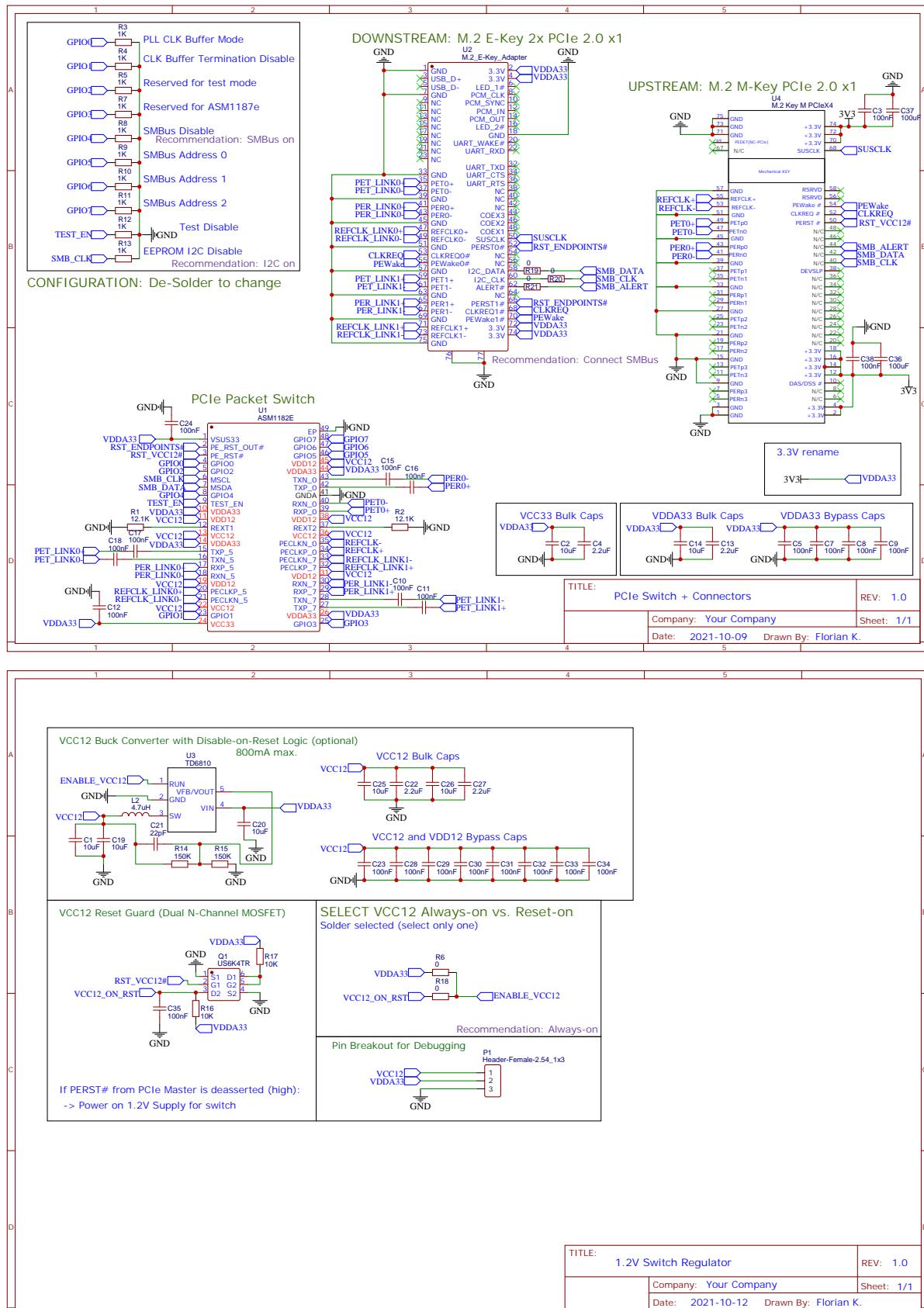
```

```

186     err = rockchip_pcie_init_port(rockchip);
187     if (err)
188     +
189     //Free GPIO Ressource
190     gpiod_put(perst_desc);
191     return err;
192 +
193
194     /* Fix the transmitted FTS count desired to exit from L0s. */
195     status = rockchip_pcie_read(rockchip, PCIE_CORE_CTRL_PLIC1);
196 @@ -325,12 +399,16 @@ static int rockchip_pcie_host_init_port(
197         struct rockchip_pcie *rockchip)
198     status |= PCI_EXP_LNKCTL_RCB;
199     rockchip_pcie_write(rockchip, status, PCIE_RC_CONFIG_LCS);
200
201     //Deassert PCIe PERST# Signal after 100ms to signal boot
202     msleep(100);
203     gpiod_set_value_cansleep(perst_desc, 1);
204     //gpiod_set_value_cansleep(rockchip->ep_gpio, 1);
205     msleep(500);
206 +
207     /* Enable Gen1 training */
208     rockchip_pcie_write(rockchip, PCIE_CLIENT_LINK_TRAIN_ENABLE,
209                         PCIE_CLIENT_CONFIG);
210
211     - gpiod_set_value_cansleep(rockchip->ep_gpio, 1);
212     -
213     /* 500ms timeout value should be enough for Gen1/2 training */
214     err = readl_poll_timeout(rockchip->apb_base +
215                             PCIE_CLIENT_BASIC_STATUS1,
216                             status, PCIE_LINK_UP(status), 20,
217 @@ -393,14 +471,18 @@ static int rockchip_pcie_host_init_port(
218         struct rockchip_pcie *rockchip)
219     status &= ~PCIE_RC_CONFIG_DCSR_MPS_MASK;
220     status |= PCIE_RC_CONFIG_DCSR_MPS_256;
221     rockchip_pcie_write(rockchip, status, PCIE_RC_CONFIG_DCSR);
222
223     +
224     //Free GPIO Ressource
225     gpiod_put(perst_desc);
226     return 0;
227 +
228     err_power_off_phy:
229     while (i--)
230         phy_power_off(rockchip->phys[i]);
231     i = MAX_LANE_NUM;
232     while (i--)
```

```
229     phy_exit(rockchip->phys[i]);  
230 + //Free GPIO Ressource  
231 + gpiod_put(perst_desc);  
232     return err;  
233 }
```

**List. A.2:** Finaler Armbian Patch



**Abb. A.1:** Schematic für PCIe Adapter Board

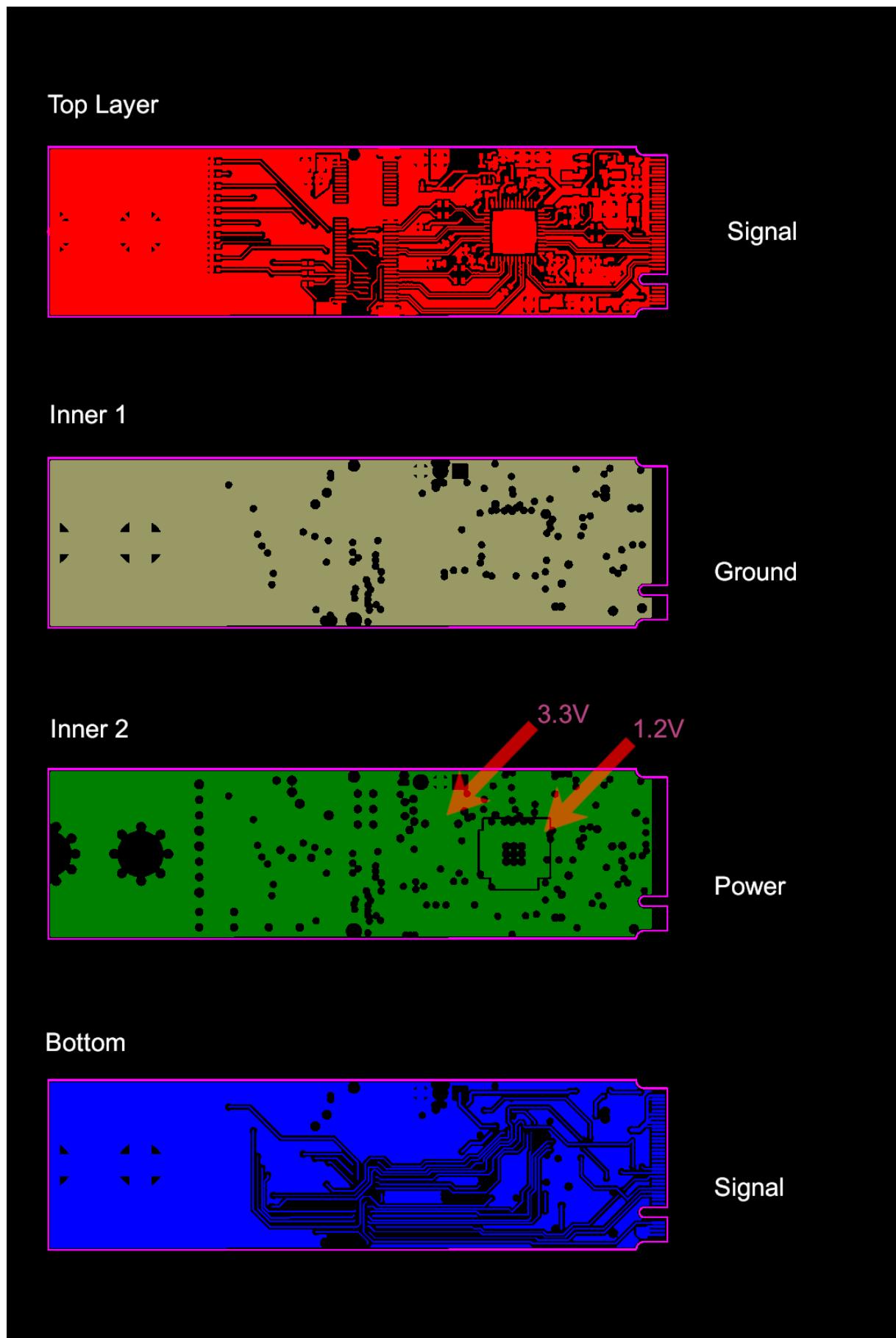
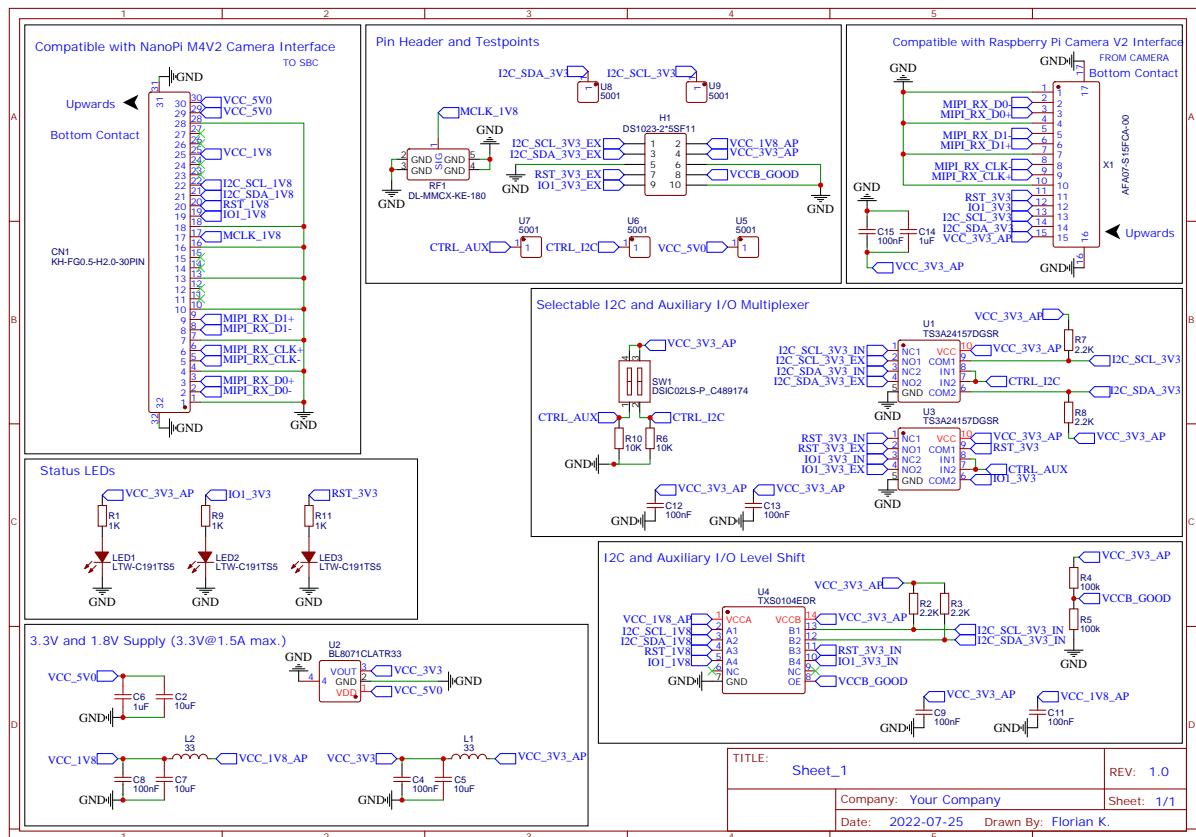


Abb. A.2: Layout für PCIe Adapter Board



**Abb. A.3:** Schematic für Kamera Adapter Board

# Literaturverzeichnis

- [1] „AArch64 Exception and Interrupt Handling“. In: Arm (29.06.2020). URL: <https://developer.arm.com/documentation/100933/0100/Synchronous-and-asynchronous-exceptions>.
- [2] ABSOLUTE ELECTRONICS SERVICES. *PCB Gold Fingers - Detailed guide on PCB gold fingers*. 2021. (Besucht am 13.02.2023).
- [3] Armbian. 8.02.2023. URL: <https://www.armbian.com/>.
- [4] ASM1182e/ASMedia Technology Inc. 11.02.2023. URL: <https://www.asmedia.com.tw/product/213yQcasx8gNAzS4/b7FyQBCxz2URbzg0>.
- [5] BEN'S PLACE. *Pi4 USB-C Gadget*. 2019. URL: <https://www.hardill.me.uk/wordpress/2019/11/02/pi4-usb-c-gadget/> (besucht am 04.02.2023).
- [6] *Controlled Impedance PCB Layer Stackup- JLCPCB*. 13.02.2023. URL: [https://cart.jlcpcb.com/impedance?\\_ga=2.109258054.2129155813.1617134999-848782461.1614671184](https://cart.jlcpcb.com/impedance?_ga=2.109258054.2129155813.1617134999-848782461.1614671184) (besucht am 13.02.2023).
- [7] CORAL. *Edge TPU Compiler / Coral*. 7.02.2023. URL: <https://coral.ai/docs/edgetpu/compiler/#system-requirements> (besucht am 07.02.2023).
- [8] CORAL. *Get started with the M.2 or Mini PCIe Accelerator / Coral*. 20.02.2023. URL: <https://coral.ai/docs/m2/get-started/#4-run-a-model-on-the-edge-tpu> (besucht am 20.02.2023).
- [9] CORAL. *M.2 Accelerator with Dual Edge TPU / Coral*. 7.02.2023. URL: <https://coral.ai/products/m2-accelerator-dual-edgetpu/> (besucht am 07.02.2023).
- [10] CORAL. *M.2 Accelerator with Dual Edge TPU / Coral*. 8.02.2023. URL: <https://coral.ai/products/m2-accelerator-dual-edgetpu> (besucht am 08.02.2023).
- [11] CORAL. *TensorFlow models on the Edge TPU / Coral*. 7.02.2023. URL: <https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview> (besucht am 07.02.2023).
- [12] *Die M.2 Schnittstelle*. 11.02.2023. URL: <https://www.delock.de/infothek/M.2/M.2.html> (besucht am 11.02.2023).
- [13] *EasyEDA - Online Schaltungs-, PCB design & Schaltplansimulator*. 11.02.2023. URL: <https://easyeda.com/de>.
- [14] *Frequently Asked Questions — libcamera*. 8.02.2023. URL: <https://libcamera.org/faq.html> (besucht am 08.02.2023).

- [15] FUZHOU ROCKCHIP ELECTRONICS Co., LTD. *Rockchip RK3399 TRM: Datasheet*. 2016. (Besucht am 18. 02. 2023).
- [16] *Getting Started — libcamera*. 20.02.2023. URL: <https://libcamera.org/getting-started.html> (besucht am 20. 02. 2023).
- [17] GITHUB. *GitHub - magic-blue-smoke/Dual-Edge-TPU-Adapter: Dual Edge TPU Adapter to use it on a system with single PCIe port on m.2 A/B/E/M slot*. 11.02.2023. URL: <https://github.com/magic-blue-smoke/Dual-Edge-TPU-Adapter> (besucht am 11. 02. 2023).
- [18] GITHUB. *GitHub - s-moch/linux-rockpro64: Linux kernel source tree + RockPro64 patches*. 20.02.2023. URL: <https://github.com/s-moch/linux-rockpro64> (besucht am 20. 02. 2023).
- [19] Google Colaboratory. 10.02.2023. URL: [https://colab.research.google.com/github/google-coral/tutorials/blob/master/retrain\\_classification\\_ptq\\_tf1.ipynb#scrollTo=Dh0zAdzF3Dyk](https://colab.research.google.com/github/google-coral/tutorials/blob/master/retrain_classification_ptq_tf1.ipynb#scrollTo=Dh0zAdzF3Dyk).
- [20] LinuxLists.cc - [BUG] PCI: rockchip: rk3399: pcie switch support. 20.02.2023. URL: [https://linuxlists.cc/1/1/linux-kernel/t/3524270/\(bug\)\\_pci:\\_rockchip:\\_rk3399:\\_pcie\\_switch\\_support](https://linuxlists.cc/1/1/linux-kernel/t/3524270/(bug)_pci:_rockchip:_rk3399:_pcie_switch_support) (besucht am 20. 02. 2023).
- [21] Raspberry Pi LTD. *Buy a Raspberry Pi Camera Module 2 – Raspberry Pi*. 7.02.2023. URL: <https://www.raspberrypi.com/products/camera-module-v2/> (besucht am 08. 02. 2023).
- [22] M.2 (NGFF) connector pinout diagram @ pinoutguide.com. 11.02.2023. URL: [https://pinoutguide.com/HD/M.2\\_NGFF\\_connector\\_pinout.shtml](https://pinoutguide.com/HD/M.2_NGFF_connector_pinout.shtml) (besucht am 11. 02. 2023).
- [23] NanoPi M4 V2 - FriendlyELEC WiKi. 11.01.2023. URL: [https://wiki.friendlyelec.com/wiki/index.php/NanoPi\\_M4V2](https://wiki.friendlyelec.com/wiki/index.php/NanoPi_M4V2) (besucht am 04. 02. 2023).
- [24] PCI-SIG KONSORTIUM. *PCI Express® Base Specification Revision 3.0*. 2009.
- [25] PCI-SIG KONSORTIUM. *PCI Express® Mini Card Electromechanical Specification Revision 1.2*. 2007.
- [26] *PCIe Hardware Design Guide*. 2015. (Besucht am 13. 02. 2023).
- [27] *pcie-rockchip-host.c - drivers/pci/controller/pcie-rockchip-host.c - Linux source code (v5.18.9) - Bootlin*. 19.02.2023. URL: <https://elixir.bootlin.com/linux/v5.18.9/source/drivers/pci/controller/pcie-rockchip-host.c>.
- [28] Rients POLITIEK. „Google Coral Edge TPU explained in depth“. In: *Q-engineering* (4.11.2021). URL: <https://qengineering.eu/google-corals-tpu-explained.html> (besucht am 07. 02. 2023).
- [29] „Preemptive and Non Preemptive Scheduling“. In: *GeeksforGeeks* (13.11.2018). URL: <https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/> (besucht am 20. 02. 2023).

- [30] *Raspberry Pi 4 Model B specifications – Raspberry Pi*. 31.01.2023. URL: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/> (besucht am 04.02.2023).
- [31] *Raspberry Pi Documentation - Camera*. 1.02.2023. URL: <https://www.raspberrypi.com/documentation/accessories/camera.html> (besucht am 08.02.2023).
- [32] RASPI.TV. *New and old Raspberry Pi Camera comparison shots (1.3, 2.1 & NOIR)*. 2016. URL: <https://raspi.tv/2016/new-and-old-raspberry-pi-camera-comparison-shots-1-3-2-1-noir> (besucht am 08.02.2023).
- [33] *RK3399 - Rockchip Wiki*. 4.02.2023. URL: <http://rockchip.wikidot.com/rk3399> (besucht am 04.02.2023).
- [34] *Rock4/hardware/gpio - Radxa Wiki*. 26.11.2022. URL: <https://wiki.radxa.com/Rockpi4/hardware/gpio> (besucht am 20.02.2023).
- [35] Ran RUBIN. „My Journey in Converting PyTorch to TensorFlow Lite“. In: *Towards Data Science* (29.09.2020). URL: <https://towardsdatascience.com/my-journey-in-converting-pytorch-to-tensorflow-lite-d244376beed> (besucht am 10.02.2023).
- [36] Raccoons-Sam STERCKVAL. *Performance comparison : Coral Edge TPU vs Jetson Nano / Raccoons*. 7.02.2023. URL: <https://blog.raccoons.be/coral-tpu-jetson-nano-performance> (besucht am 07.02.2023).
- [37] Armbian TEAM. *Armbian config - Armbian Documentation*. 8.02.2023. URL: [https://docs.armbian.com/User-Guide\\_Armbian-Config/](https://docs.armbian.com/User-Guide_Armbian-Config/) (besucht am 08.02.2023).
- [38] Armbian TEAM. *Build Options - Armbian Documentation*. 19.02.2023. URL: [https://docs.armbian.com/Developer-Guide\\_Build-Options/#hidden-options-for-advanced-users-default-values-are-marked-bold](https://docs.armbian.com/Developer-Guide_Build-Options/#hidden-options-for-advanced-users-default-values-are-marked-bold) (besucht am 19.02.2023).
- [39] TENSORFLOW. *Module: tf.keras.layers*; *TensorFlow v2.11.0*. 8.12.2022. URL: [https://www.tensorflow.org/api\\_docs/python/tf/keras/layers](https://www.tensorflow.org/api_docs/python/tf/keras/layers) (besucht am 07.02.2023).
- [40] TENSORFLOW. *Quantization aware training*; *TensorFlow Model Optimization*. 3.08.2022. URL: [https://www.tensorflow.org/model\\_optimization/guide/quantization/training](https://www.tensorflow.org/model_optimization/guide/quantization/training) (besucht am 07.02.2023).
- [41] TEXAS INSTRUMENTS INCORPORATED. *Layout Guidelines of PCIe® Gen 4.0 Application With the TMUXHS4412 Multiplexer: Application Report*. 2021. URL: <https://www.ti.com/lit/an/slaae45/slaae45.pdf?ts=1676300606957> (besucht am 13.02.2023).
- [42] *tmp: isp and rockpi 4 dts (7842ca07) · Commits · Helen Mae Koike Fornazier / linux · GitLab*. 21.2.2023. URL: <https://gitlab.collabora.com/koike/linux/-/commit/7842ca07b75828785ffcd217362a82eaa9cc1e21> (besucht am 21.02.2023).

- [43] *Trace Impedance Calculator / DigiKey*. 13.02.2023. URL: <https://www.digikey.de/en/resources/conversion-calculators/conversion-calculator-pcb-trace-impedance>.
- [44] WIKIPEDIA, Herausgeber. *Electroless nickel immersion gold*. 2021. URL: [https://en.wikipedia.org/w/index.php?title=Electroless\\_nickel\\_immersion\\_gold&oldid=1021354363](https://en.wikipedia.org/w/index.php?title=Electroless_nickel_immersion_gold&oldid=1021354363) (besucht am 13.02.2023).
- [45] WIKIPEDIA, Herausgeber. *M.2*. 2023. URL: <https://en.wikipedia.org/w/index.php?title=M.2&oldid=1136576017> (besucht am 11.02.2023).
- [46] WIKIPEDIA, Herausgeber. *ONNX*. 2021. URL: <https://de.wikipedia.org/w/index.php?title=ONNX&oldid=215136676> (besucht am 10.02.2023).
- [47] WIKIPEDIA, Herausgeber. *PCI Express*. 2023. URL: [https://de.wikipedia.org/w/index.php?title=PCI\\_Express&oldid=230678293](https://de.wikipedia.org/w/index.php?title=PCI_Express&oldid=230678293) (besucht am 13.02.2023).
- [48] WIKIPEDIA, Herausgeber. *Rockchip*. 2023. URL: <https://de.wikipedia.org/w/index.php?title=Rockchip&oldid=230342293> (besucht am 04.02.2023).