HTBL Hollabrunn

Höhere Lehranstalt für Elektronik und
Technische Informatik

DIPLOMARBEIT

Gesamtprojekt

Advanced Microcontroller Training System

Software ARM Cortex-M3 Minimalsystem
Andreas Mieke 5BHEL Betreuer/in: Dipl.-Ing. Josef Reisinger

Z80 Minimalsystem
Andreas Reischl 5AHEL Betreuer/in: Dipl.-Ing. Josef Reisinger

Hardware ARM Cortex-M3 Minimalsystem
Kevin Schuh 5BHEL Betreuer/in: Dipl.-Ing. Josef Reisinger

Schuljahr 2017/18

Abgabevermerk:

Datum: Übernommen von:

Höhere Technische Bundeslehranstalt Hollabrunn

Höhere Lehranstalt für Elektronik und Technische Informatik

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Andreas Mieke

Andreas Reischl

Kevin Schuh

Hollabrunn, am 4. April 2018

HINWEISE

Die vorliegende Diplomarbeit wurde für die Abteilung Elektronik und Technische Informatik der HTL Hollabrunn ausgeführt.

Die in dieser Diplomarbeit entwickelten Prototypen und Software-Produkte dürfen ganz oder auch in Teilen von Privatpersonen oder Firmen nur dann in Verkehr gebracht werden, wenn sie diese selbst geprüft und für den vorgesehenen Verwendungszweck für geeignet befunden haben. Es wird keinerlei Haftung übernommen für irgendwelche Schäden, die aus der Nutzung der hier entwickelten oder beschriebenen Bestandteile des Projekts resultieren.

Für alle Entwicklungen gilt die GNU General Public License [http://www.gnu.org/licenses/gpl.html] der Free Software Foundation, Boston, USA in der Version 3.

Die Diplomarbeit erfüllt die „Standards für Ingenieur- und Technikerprojekte“ entsprechend dem Rundschreiben Nr. 60 aus 1999 des BMBWK (GZ.17.600/101-II/2b/99). [https://www.bmb.gv.at/ministerium/rs/1999_60.html]

SCHLÜSSELBEGRIFFE

ST-Link V2
ULINK/ME
Cortex-M3
STM32F107RCT(6)
Nextion NX4832T035_011
JTAG
SPI
UART
I^2C
Core-Modul
Basisplatine
USB-to-UART
Altium
μVision 5
ARM

DANKSAGUNGEN

Im Vorhinein möchten wir uns herzlichst bei unserem Diplomarbeitsbetreuer Herrn Dipl.-Ing. Josef Reisinger bedanken, der uns stets kompetent beraten hat und uns sein Wissen zur Verfügung stellte.

Des Weiterem möchten wir uns bei Herrn Dipl.-Ing. Erwin Dobart bedanken, der uns bei technischen Fragen unterstützte.

Weiters möchten wir uns bei Herrn FOL StR Ing. Manfred Resel bedanken, der uns, solange er noch im Dienst war, bei Softwareproblemen und Hardwarefragen aller Art zur Seite stand.

Darüber hinaus möchten wir uns bei Herrn Wolfgang Kauer und Herrn Ferdinand Klampfer bedanken, ohne deren Hilfe wir unsere Leiterkarten nicht hätten bestücken können.

Ebenfalls möchten wir Herrn Dipl.-Ing. Wilfried Trollmann bedanken, welcher uns immer an unsere Fristen und Termine erinnerte und uns jederzeit über aktuelle Wettbewerbe informierte.

Außerdem möchten wir uns bei Thomas Fehringer, unseren Laboranten, bedanken, welcher uns mit Bauteilen für unsere Diplomarbeit versorgte.

htl bildung mit zukunft	Höhere Technische Bundeslehranstalt Hollabrunn
	Fachrichtung: Elektronik und Technische Informatik

DIPLOMARBEIT

Dokumentation

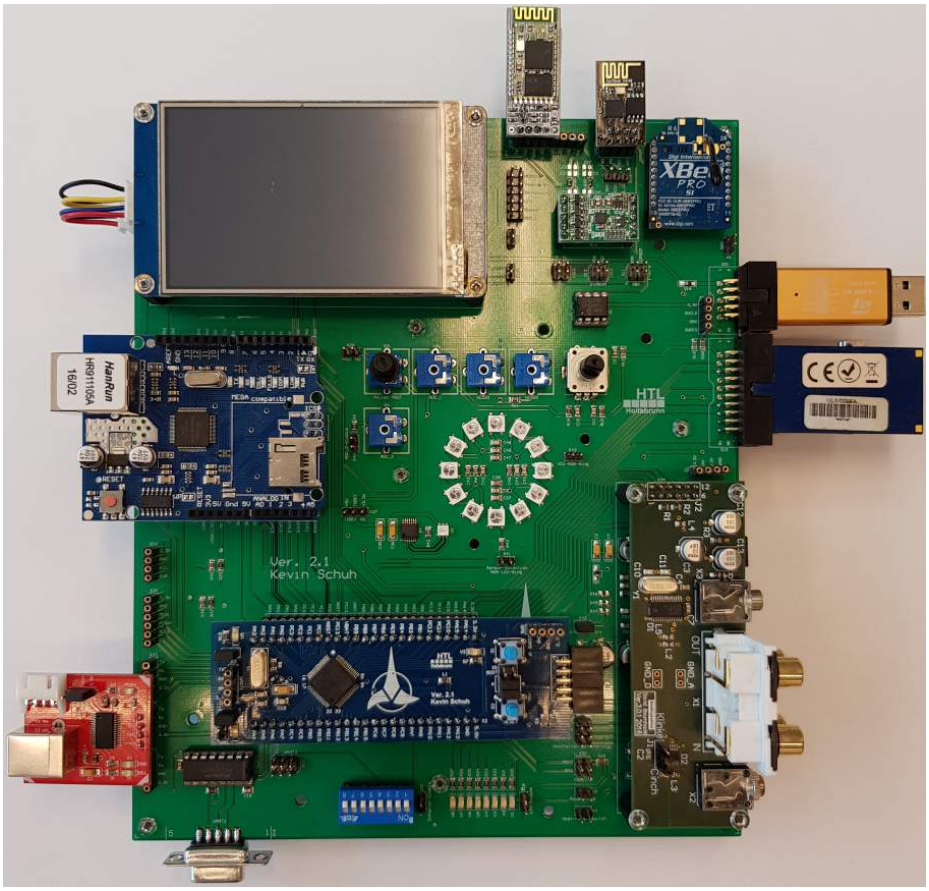
Name der Verfasser/innen	Andreas Mieke, Andreas Reischl, Kevin Schuh
Jahrgang Schuljahr	5xHEL 2017/18
Thema der Diplomarbeit	Advanced Microcontroller Training System
Kooperationspartner	

Aufgabenstellung	Aufgabe soll es sein, eine neue Version für das HTL eigene ARM Minimalsystem zu realisieren. Zunächst soll ein Touchscreen-Display zur Ein- und Ausgabe unterstützt werden. Des Weiteren soll eine Arduino-UNO kompatible Schnittstelle zur Verfügung gestellt werden, um Arduino Shields von verschiedenen Herstellern einsetzen zu können. Darüber hinaus soll das neue System verschiedene Funkmodule unterstützen, um damit eine Kommunikation mit anderer Peripherie zu erleichtern. Ein Audiomodul, welches bereits bei einer Diplomarbeit aus dem Jahre 2015/16 entwickelt wurde, soll ebenso unterstützt werden. Zusätzlich soll noch ein Z80 Minimalsystem, welches im Rahmen mehrerer Diplomarbeiten entstanden ist, für den Einsatz im Laborunterricht vervollständigt werden.

Realisierung	Ziel des Projekts war ein neues modulares HTL ARM-Minimalsystem für den Unterricht im Bereich embedded Systems zu entwickeln. Zu diesem Zweck wurden mehrere Leiterplatten entwickelt. Ein Core-Modul für den Microcontroller welches auf eine Basisplatine gesteckt werden kann sowie ein USB-to-UART Konverter um mit einem PC kommunizieren zu können. Für die Inbetriebnahme bei Fertigung wurde eine Testsoftware sowie Beispielapplikation entwickelt. Weiters wurden mehrere Leiterkarten für ein Z80 Minimalsystem gefertigt, die für den Laborunterricht dienen.

Ergebnisse	Es wurden 2 Komplettsysteme des ARM-Minimalsystems gefertigt und erfolgreich in Betrieb genommen. Die Testsoftware und die Demolibrary wurden erfolgreich demonstriert. Weiters wurden 4 Z80 Minimalsysteme gefertigt und erfolgreich in Betrieb genommen.

htl bildung mit zukunft	Höhere Technische Bundeslehranstalt Hollabrunn
	Fachrichtung: Elektronik und Technische Informatik

Gesamtsystem	

Teilnahme an Wettbewerben, Auszeichnungen	Jugend Innovativ Technik fürs Leben-Preis

Möglichkeiten der Einsichtnahme in die Arbeit	HTL Hollabrunn Anton Ehrenfriedstraße 10 2020 Hollabrunn	
Aprobation (Datum/Unterschrift)	Prüfer/Prüferin	Direktor/Direktorin Abteilungsvorstand/Abteilungsvorständin

htl bildung mit zukunft

Höhere Technische Bundeslehranstalt Hollabrunn
College of Engineering

Department: Electronics and computer engineering

DIPLOMA THESIS

Documentation

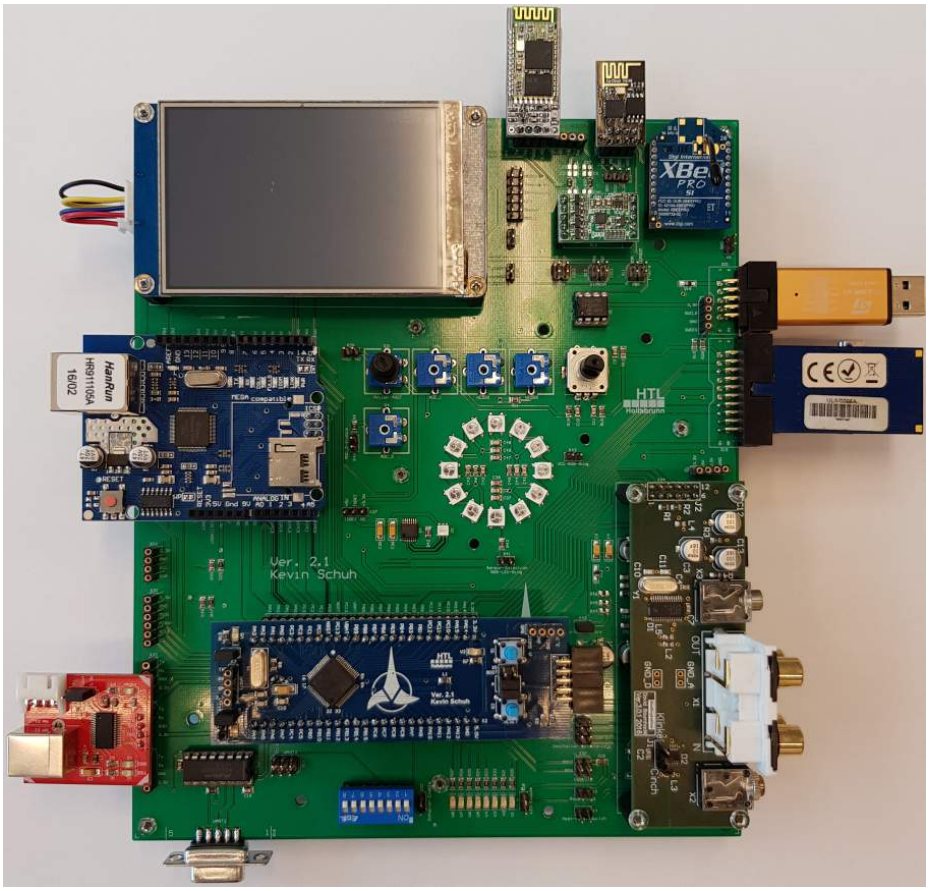
Author(s)	Andreas Mieke, Andreas Reischl, Kevin Schuh
Form Academic year	5xHEL 2017/18
Topic	Advanced Microcontroller Training System
Co-operation partners	

Asignment of tasks	The task should be to realize a new version for the HTL (secondary technical college) own ARM minimal system. At first, a touchscreen display for input and output should be supported. Furthermore, an Arduino-UNO compatible interface should make it possible to use Arduino shields from different manufacturers. In addition, the new system should support various wireless modules to facilitate communication with other peripherals. An audio module, which was already developed in a diploma thesis from the year 2015/16, should also be supported. In addition, a Z80 minimal system, which was created in the context of several diploma theses, should be finalised for the use in laboratory lessons.

Realisation	Goal of the project was to develop a new modular HTL ARM minimal system for embedded systems classes. For this purpose multiple PCBs were developed. A core-module, housing the microcontroller, which can be connected to the basis board as well as a USB-to-UART converter which allows simple communication with a PC. For testing while manufacturing the board a test software and example application was developed. Moreover, multiple PCBs for a Z80 minimal system for usage in laboratory classes was developed.

Results	Two complete ARM minimal systems were built and tested successfully. The test software as well as the demo library were demonstrated successfully. Last but not least, four Z80 minimal systems were built and tested successfully.

htl bildung mit zukunft	Höhere Technische Bundeslehranstalt Hollabrunn College of Engineering
	Department: Electronics and computer engineering

System overview	

Participation in competitions, Awards	Jugend Innovativ Technik fürs Leben-Preis

Accessibility of final project thesis	HTL Hollabrunn Anton Ehrenfriedstraße 10 2020 Hollabrunn	
Approval (Date/Signature)	Examiner/s	Head of Department/College

Advanced Microcontroller Training System

Verlauf

14.09.2017 um 23:53 Die Themenstellung "Advanced Microcontroller Training System" (Andreas Mieke, Andreas Reischl, Kevin Schuh) wurde eingereicht.
15.09.2017 um 16:08 Die Themenstellung "Advanced Microcontroller Training System" (Kevin Schuh) wurde vom Betreuer / von der Betreuerin akzeptiert.
15.09.2017 um 18:05 Die Themenstellung "Advanced Microcontroller Training System" (Kevin Schuh) wurde vom zuständigen Abteilungsvorstand akzeptiert.
20.09.2017 um 13:27 Die Themenstellung "Advanced Microcontroller Training System" (Kevin Schuh) wurde vom Direktor / von der Direktorin akzeptiert.
26.09.2017 um 13:42 Die Themenstellung "Advanced Microcontroller Training System" (Andreas Mieke Andreas Reischl, Kevin Schuh) wurde vom Landesschulinspektor / von der Landesschulinspektorin abgelehnt.
26.09.2017 um 14:38 Die Themenstellung "Advanced Microcontroller Training System" (Andreas Mieke, Andreas Reischl, Kevin Schuh) wurde eingereicht.
26.09.2017 um 15:26 Die Themenstellung "Advanced Microcontroller Training System" (Kevin Schuh) wurde vom Betreuer / von der Betreuerin akzeptiert.
27.09.2017 um 12:17 Die Themenstellung "Advanced Microcontroller Training System" (Kevin Schuh) wurde vom zuständigen Abteilungsvorstand akzeptiert.
02.10.2017 um 14:58 Die Themenstellung "Advanced Microcontroller Training System" (Kevin Schuh) wurde vom Direktor / von der Direktorin akzeptiert.
03.10.2017 um 10:52 Die Themenstellung "Advanced Microcontroller Training System" (Kevin Schuh) wurde vom Landesschulinspektor / von der Landesschulinspektorin genehmigt.

Schule

Höhere technische Bundeslehranstalt HOLLABRUNN

Abteilung(en)

Hauptverantwortlich: Elektronik und Technische Informatik

AV

Hauptverantwortlich: Wilfried Trollmann

Abschließende Prüfung

2018

Betreuer/innen

Hauptverantwortlich: Josef Reisinger

Ausgangslage

Seit mehreren Jahren wird in der HTBLA-Hollabrunn, ein ARM Cortex-M3 Minimalsystem, für die Ausbildung der Schüler im Bereich embedded Systems eingesetzt. Ziel dieser Diplomarbeit ist es, eine neue Version dieses Systems zu realisieren, um einen zukunftsorientierten Unterricht zu ermöglichen. Des Weiterem soll ein Z80 Minimalsystem für den Laborunterricht finalisiert werden.

Projektteam (Arbeitsaufwand)

Name	Individuelle Themenstellung	Klasse	Arbeitsaufwand
Kevin Schuh (Hauptverantwortlich)	Hardware ARM Cortex-M3 Minimalsystem	5BHEL_18	180 Stunden
Andreas Mieke	Software ARM Cortex-M3 Minimalsystem	5BHEL_18	180 Stunden
Andreas Reischl	Z80 Minimalsystem	5AHEL_18	180 Stunden

Projektpartner

Untersuchungsanliegen der individuellen Themenstellungen

Aufgabe soll es sein, eine neue Version für das HTL eigene ARM Minimalsystem zu realisieren. Zunächst soll ein Touchscreen-Display zur Ein- und Ausgabe unterstützt werden. Des Weiteren soll eine Arduino-UNO kompatible Schnittstelle zur Verfügung gestellt werden, um Arduino Shields von verschiedenen Herstellern einsetzen zu können. Darüber hinaus soll das neue System verschiedene Funkmodule unterstützen, um damit eine Kommunikation mit anderer Peripherie zu erleichtern. Ein Audiomodul, welches bereits im Rahmen einer Diplomarbeit aus dem Jahre 2015/16 entwickelt wurde, soll ebenso unterstützt werden. Zusätzlich soll ein Z80 Minimalsystem, welches im Rahmen mehrerer Diplomarbeiten entstanden ist, für den Einsatz im Laborunterricht vervollständigt werden.

Zielsetzung

Ziel dieses Projekts ist es sowohl ein ARM Minimalsystem, als auch ein Z80 Minimalsystem, für den Unterricht an der HTL Hollabrunn mit entsprechender Software zu entwickeln.

Geplantes Ergebnis der Prüfungskandidatin/des Prüfungskandidaten

Zuerst sollen die einzelnen Arbeitsaufträge entwickelt und überprüft werden. Anschließend sollen die einzelnen Systemkomponenten zum fertigen System zusammengefügt und in Betrieb genommen werden. Die Funktion und die einzelnen Entwicklungsschritte zum fertigen Prototypen sollen anschließend durch eine umfangreiche Dokumentation und eine Bedienungsanleitung vervollständigt werden.

Meilensteine

30.06.2017 Konzept für Lehrsysteme entwickelt

17.11.2017 Prototyp für Lehrsysteme gefertigt

17.11.2017 Inbetriebnahme der gefertigten Leiterplatten

26.01.2018 Fertigstellung der Lehrsysteme

23.03.2018 Abgabe des Hardware- und Softwaremanuels

Berufsbildende Schulen

Erklärung

Die unterfertigten Kandidaten/Kandidatinnen haben gemäß § 34 Abs. 3 Z 1 und § 37 Abs. 2 Z 2 des Schulunterrichtsgesetzes in Verbindung mit den Bestimmungen der „Prüfungsordnung BMHS, Bildungsanstalten", BGBl. II Nr. 177/2012 i.d.g.F. die Ausarbeitung einer Diplomarbeit/Abschlussarbeit mit folgender Aufgabenstellung gewählt:

Advanced Microcontroller Training System (Gesamtprojekt)

Individuelle Aufgabenstellungen im Rahmen des Gesamtprojektes:

- Andreas Mieke (5BHEL_18): **Software ARM Cortex-M3 Minimalsystem**
- Andreas Reischl (5AHEL_18): **Z80 Minimalsystem**
- Kevin Schuh (5BHEL_18): **Hardware ARM Cortex-M3 Minimalsystem**

Die Kandidaten/Kandidatinnen nehmen zur Kenntnis, dass die Diplomarbeit/Abschlussarbeit in eigenständiger Weise und außerhalb des Unterrichtes zu bearbeiten und anzufertigen ist, wobei Ergebnisse des Unterrichtes mit einbezogen werden können, die jedenfalls als solche entsprechend kenntlich zu machen sind.

Die Abgabe der vollständigen Diplomarbeit/Abschlussarbeit hat in digitaler und in zweifach ausgedruckter Form bis spätestens **04.04.2018** beim zuständigen Betreuer/der zuständigen Betreuerin zu erfolgen.

Die Kandidaten/Kandidatinnen nehmen weiters zur Kenntnis, dass ein Abbruch der Diplomarbeit/Abschlussarbeit nicht möglich ist.

Kandidaten/Kandidatinnen:	**Datum und Unterschrift bzw.Handysignatur:**
Andreas Mieke (5BHEL_18)	11.09.17
Andreas Reischl (5AHEL_18)	Andreas Reischl 11.09.17
Kevin Schuh (5BHEL_18)	Kevin Schuh 11.09.17

Inhaltsverzeichnis

1 Allgemeines

1.1 Entstehungsgeschichte

Seit mehreren Jahren wird in der HTBL-Hollabrunn, ein ARM Limited (ARM) Cortex-M3 Minimalsystem, für die Ausbildung unserer Schüler, im Bereich „embedded Systems“ eingesetzt.

Wie schon im Abstract beschrieben geht es bei dem neuen System darum, sich neuen Technologien und Anwenderszenarien zu öffnen beziehungsweise schnelles Prototyping (Rapid Protoyping) zu ermöglichen. Mit Hilfe des Nextion-Touchscreen-Displays wird ein modernes Mensch-Maschine-Interface (MMI) bereitgestellt, um Anwendungen leichter und interaktiv bedienbar zu machen. Das Audio-Interface ermöglicht es, Anwendungen für digitale Signalverarbeitung (z.B. digitale Filter) zu realisieren. Das Arduino-Interface ermöglicht es, verschiedenste Arduino-Shields für den Unterricht einzusetzen. Diese Schnittstellen, sowie die Schnittstellen für WLAN, Bluetooth und Funkmodule ermöglichen es auf schnelle Art und Weise Konzepte für Diplomarbeiten zu evaluieren.

1.2 Anwendungsszenarien

Das gesamte ARM-Minimalsystem soll dazu beitragen, mit Hilfe einer Vielzahl an Schnittstellen, hardwarenahe Programmierung zu erlernen, sowie das Bauen und Testen von Prototypen zu erleichtern. Weiters kann aufgrund, des auf der Basisplatine vorhandenen Arduino-Sockels eine Kompatibilität zu allen Arduino-Shields erreicht werden, welche nun über das Core-Modul angesteuert werden können.

Das Hauptaugenmerk wurde jedoch auf folgende Anwendungen gelegt:

- Digitale Signalverarbeitung (DSV)
- Kommunikation mit diversen Schnittstellen (I^2C, SPI, UART, 1-Wire, …)
- Hardwarekompatibilität zu Arduino-Shields
- Graphical User Interface (GUI)

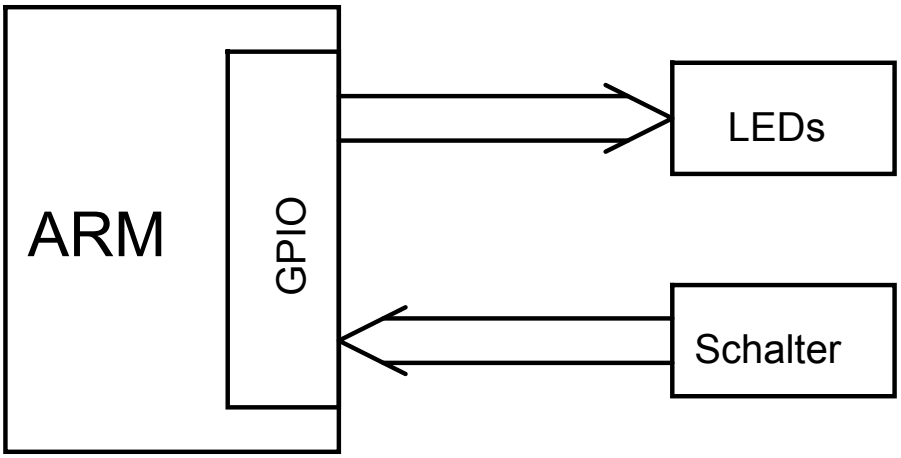


Abbildung 1: Anwendungsszenario: GPIO

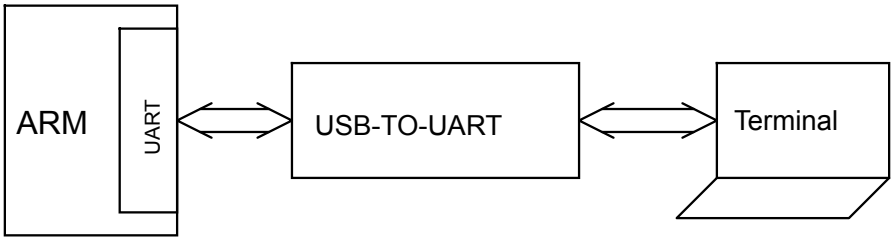


Abbildung 2: Anwendungsszenario: UART

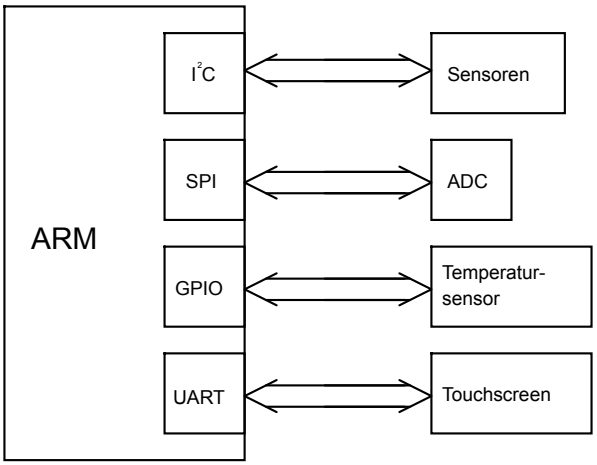


Abbildung 3: Anwendungsszenario: Serielle Kommunikation

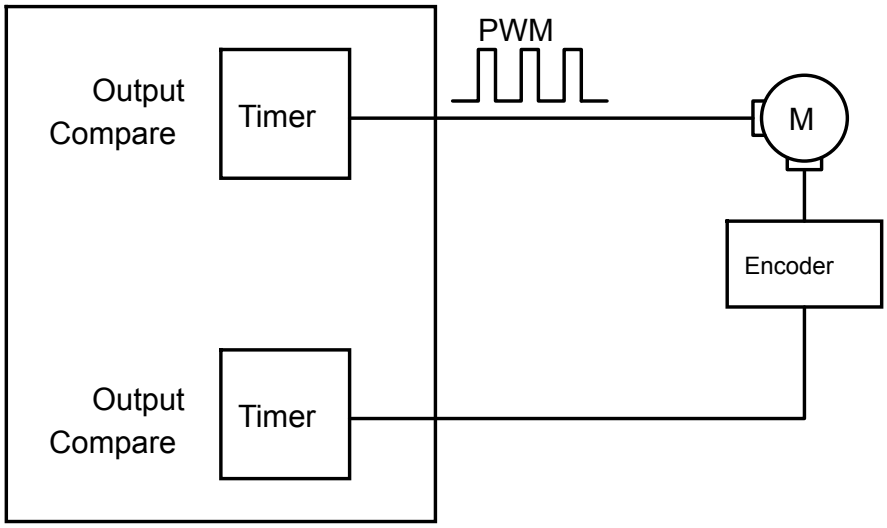


Abbildung 4: Anwendungsszenario: Timer/Interrupt

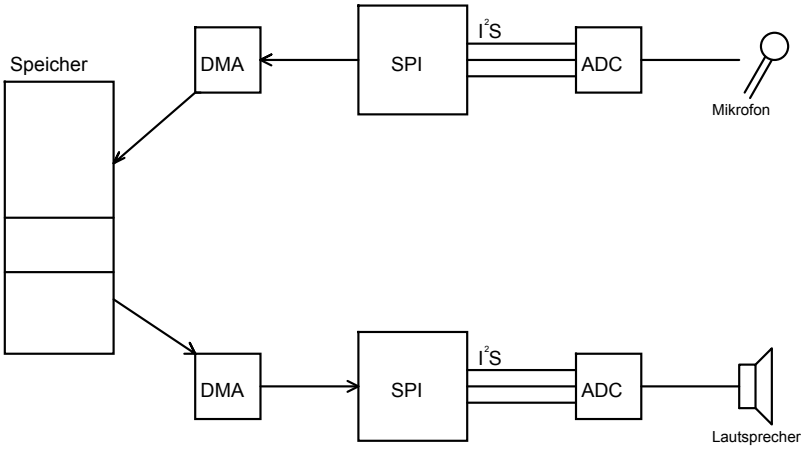


Abbildung 5: Anwendungsszenario: Audioverarbeitung

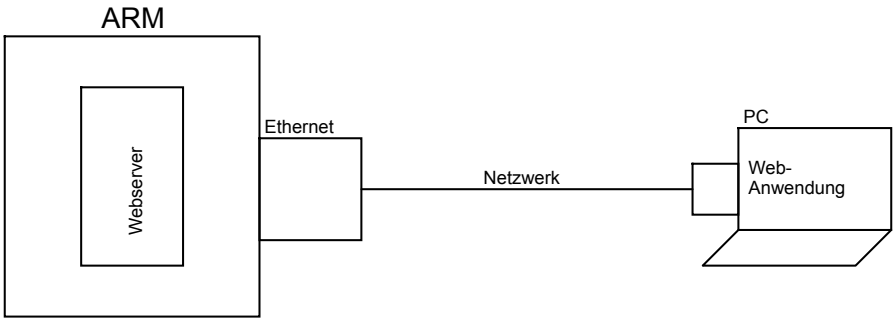


Abbildung 6: Anwendungsszenario: Webanwendung

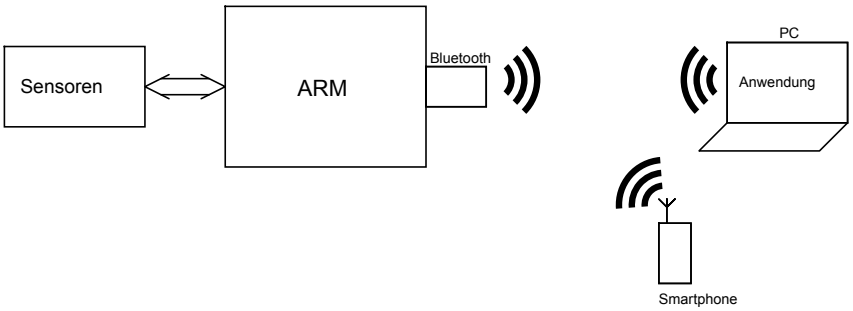


Abbildung 7: Anwendungsszenario: Bluetooth

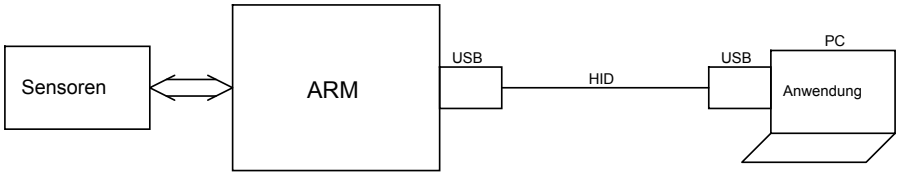


Abbildung 8: Anwendungsszenario: USB HID

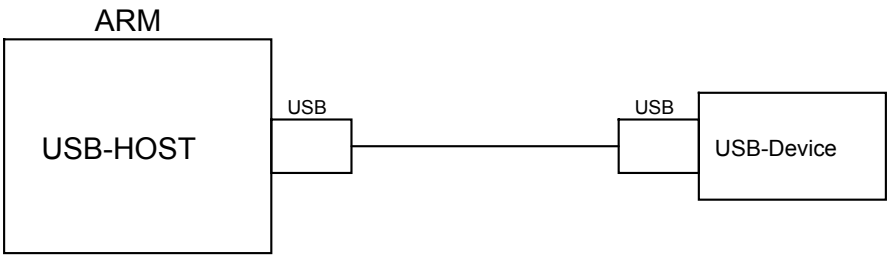


Abbildung 9: Anwendungsszenario: USB Host

1.3 Versionierung

Das Kapitel Versionierung teilt sich in zwei große Teile auf, Semantic Versioning, was die Vergabe der Versionsnummer an sich behandelt und Git, was im generellen Workflow zur Versionskontrolle benutzt wurde.

1.3.1 Semantic Versioning

Semantic Versioning, oder auch semantische Versionierung bezeichnet ein Verfahren zur Vergabe von Versionsnummern welches sich als sehr praktisch zum Versionieren von Softwarekomponenten herausgestellt hat. Heute benutzen sehr viele große Softwareprojekte, vor allem im Open Source Bereich, Semantic Versioning für die Versionierung von Releases.

Bei Semantic Versioning setzt sich die Versionsnummer aus drei Hauptgruppen, welche aus Ziffern bestehen und durch einen Punkt getrennt sind, zusammen. Jede dieser Gruppen hat eine festgelegte Bedeutung, von links nach rechts heißen die Gruppen „Major“, „Minor“ und „Patch“.

Ein Produkt mit der Versionsnummer **2.5.15** hat also die Major-Version **2**, Minor-Version **5** und Patch-Level **15**.

Will man nun eine neue Version der Software (oder des Produkts) veröffentlichen, so muss man, je nach Änderung, die Versionsnummer erhöhen. Hierbei wird meist nach Tabelle 1 vorgegangen.

Gruppe	Bedeutung
Patch	wird erhöht wenn Fehler in der Software ausgebessert werden, jedoch keine neuen Funktionen hinzugefügt werden. Binäre Bibliotheken bleiben untereinander komplett kompatibel.
Minor	wird erhöht wenn neue Funktinen hinzugefügt werden, nebenbei können auch Fehler ausgebessert werden, ohne eine Erhöhung des Patch Levels zu erfordern. Binäre Bibliotheken sind abwärtskompatibel, das heißt Bibliotheken mit Version **2.15.6** können anstatt Version **2.10.0** verwendet werden. Umgekehrt ist das aber nur so lange möglich, so lange mein keine erweiterten Funktionen aus der Bibliothek mit der höheren Minor-Version verwendet.
Major	wird erhöht wenn einerseits neue Funktionen hinzugefügt werden, allerdings gleichzeit auch alte gelöscht oder sonst irgendwie inkompatibel gemacht (Umbennenung, Änderung der Übergabeparameter) werden. Binäre Bibliotheken sind, bis auf wenige Ausnahmen, normalerweise nicht kompatibel.

Tabelle 1: Zifferngruppen

Für die Softwareprodukte, welche im Rahmen dieser Diplomarbeit entstanden sind, wurde Semantic Versioning angewendet, um zukünftigen Nutzern oder Bearbeitern ein solides Fundament in Sachen Kompatibilität zu gewähren.

1.3.2 Git

Git ist ein dezentrales Versionskontrollsystem, welches erlaubt Zeitpunkte in der Softwareentwicklung (mit Kommentaren versehen) festzuhalten, zwischen diesen zu springen und Teile von oder komplette Änderungen rückgängig zu machen, wenn dies nötig sein sollte. Git legt hierbei für jedes Projekt ein dezentrales Repository an, welches, je nach belieben auch quasi-zentral auf einem Server liegen kann. In einem Repository kann man dann entweder alleine, oder zusammen mit einer oder mehreren Personen am selben Projekt arbeiten. Git übernimmt dabei die Versionskontrolle und in den meisten Fällen auch erfolgreich die Konflktlösung.

1.3.2.1 Repository anlegen

Um die Arbeit an einem Projekt beginnen zu können, muss erstmal ein leeres Git-Repository erstellt werden, dies geschieht mit dem Befehl `git init`. Dieser Befehl legt im Verzeichnis, in dem man sich gerade befindet, einen `.git`-Ordner an, in welchem Git seine internen Daten speichert.

1.3.2.2 Der erste Commit

Nun, da ein Repository angelgt ist, kann die eigentliche Arbeit am Projekt beginnen, so wie man das gewöhnt ist. Wenn man nun vorzeigbare Ergebnisse hat, oder seinen Fortschritt vor größeren Änderungen sichern will, muss man einen Commit erstellen, dieser bildet dann den aktuellen Zustand des Projekts (mit Dateiberechtigungen) und kompletten Inhalt ab. Wenn man später zu diesem Zustand zurückkehren will, kann man einfach auf den Commit zurückkehren. Weiters speichert Git die Änderungen von einem zum nächsten Commit in einem Diff-Format, was bedeutet, dass nicht immer die ganze Datei, sondern nur die Änderungen zur letzten Version, gespeichert werden. Dies ist für ASCII-Dateien (wie zum Beispiel auch Source-Code) sehr effizient, da nicht geänderte Zeilen nicht immer abgespeichert werden müssen.

Um nun die gemachten Änderungen in den Staging-Bereich hinzuzufügen führt man nun entweder `git add -A` (für alle Dateien und Verzeichnisse) oder `git add <file.name>` für genau eine (oder mehrere angegebene) Datei(en). Nun können entweder noch weitere Dateien hinzugefügt, Dateien wieder aus dem Staging-Bereich entfernt (`git reset HEAD <file.name>`) oder ein Commit erstellt werden. Letzteres wird mit dem Befehl `git commit` gemacht, dieser öffnet den Standard-Texteditor, in welchem man eine Nachricht (meistens die Änderungen, die man gemacht hat) eingibt. Wird der Befehl mit dem Flag `-m "Nachricht"` ausgeführt, so geht kein Editor auf und der Commit wird direkt mit der angegebenen Nachricht erstellt.

1.3.2.3 Zurück zu einer alten Version

Wenn man nun feststellt, dass das, was man programmiert hat nicht zielführend ist oder sich gar negativ auf das Projekt ausgewirkt hat, kann man relativ einfach wieder auf eine funktionierende Version zurück kehren. Hierzu führt man zuerst `git log` aus, was dann die IDs aller Commits und die erste Zeile der Commit-Nachricht anzeigt, hier sucht man sich nun die ID heraus, zu der man zurück kehren will, und gibt diese bei `git checkout <commitid>` ein. Nun stellt Git wieder die Version her, wie sie zum Zeitpunkt des Commits existierte.

1.4 LaTeX

Zum setzen der Dokumentation und anderer aus dieser Diplomarbeit resultierenden Dokument wurde LaTeX verwendet. Die Verwendung von LaTeX bietet im Gegensatz zu anderer Software einige Vorteile, wie zum Beispiel die einfacher Literaturverwaltung mittels BiBLaTeX und das einfache erstellen von Tabellen und ähnlichem direkt in einem handelsüblichen Texteditor. Des weiteren ist es möglich ganze Dokumente in quasi unendlich kleine Teile aufzuteilen, sodass mehrere Leute parallel an einem Gesamtdokument arbeiten können.

1.5 Nextion Editor

Das Interface für das Display kann neben den zur verfügung stehenden Befehlen auch mit einem grafischen Editor erstellt werden. Die Software die dafür genutzt werden muss ist der Nextion Editor. Damit können auch Bitmap-Schriftarten für das Display erstellt werden. Das Display wird mit einer eigenen Programmiersprache programmiert, die simple Befehle ausführen kann, wie Wechsel von Display-Seiten, User-Interface Elemente beeinflussen und verschiedene Modi des Displays aktivieren oder deaktivieren.

Ein simples Projekt kann mit den in Abbildung 10 gezeigten Einstellungen kofiguriert werden.

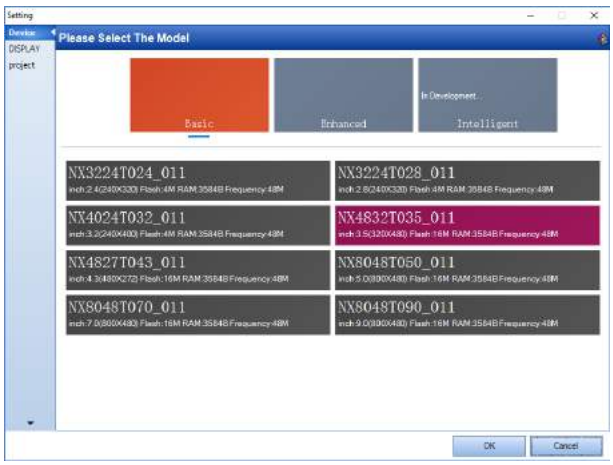


Abbildung 10: Screenshot der Einstellungen die für ein neues Nextion Editor Projekt gewählt werden müssen

Um ein fertiges Programm auf das Display zu flashen muss das Display mit dem USB-to-UART-Adapter mit dem Computer verbunden werden und dann im Nextion Editor das Programm übertragen werden. Die Baudrate und der Port werden vom Editor im

Normalfall autoamtisch ermittelt und müssen nicht eigestellt werden. Des weiteren ist zu beachten, das beim flashen des Programms unter anderem die Display Firmware aktualisiert werden kann und somit andere Programme nicht mehr auf dem Display ausführbar sind, wenn die Firmware die Befehle der Nextion Programmiersprache ändert. Dies kann ggf. den Editor-Changelogs entnommen werden.

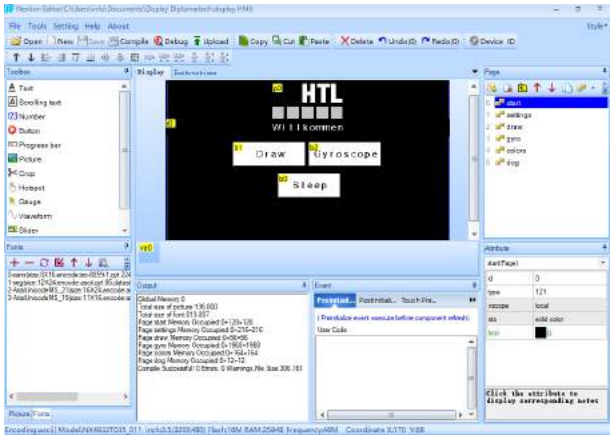


Abbildung 11: Die Nextion Editor Oberfläche

Um das im Nextion Editor erstellte User Interface (beziehungsweise die Display Firmware) auf das Display zu flashen, muss dieses mit dem Display Kabel mit dem USB-to-UART-Adapter verbunden werden. Am Adapter selbst muss die Versorgungsspannung auf 5 V gesetzt werden (Jumper X4), da das Display eine Versorgungsspannung von 5 V benötigt um korrekt zu funktionieren. Die Kerben im Stecker stellen dabei die richtige Polarität sicher.

Abbildung 12: Displayverbindung zum flashen

1.6 Systemaufbau

Das neue ARM-Minimalsystem kann prinzipiell in vier voneinander getrennten Platinen unterteilt werden. Diese Module wären die Basisplatine, das Core-Modul, der USB-to-UART Adapter und der Audioadapter. Jedes dieser Module erfüllt einen bestimmten Zweck, welcher schlussendlich zum Gesamtsystem beträgt.

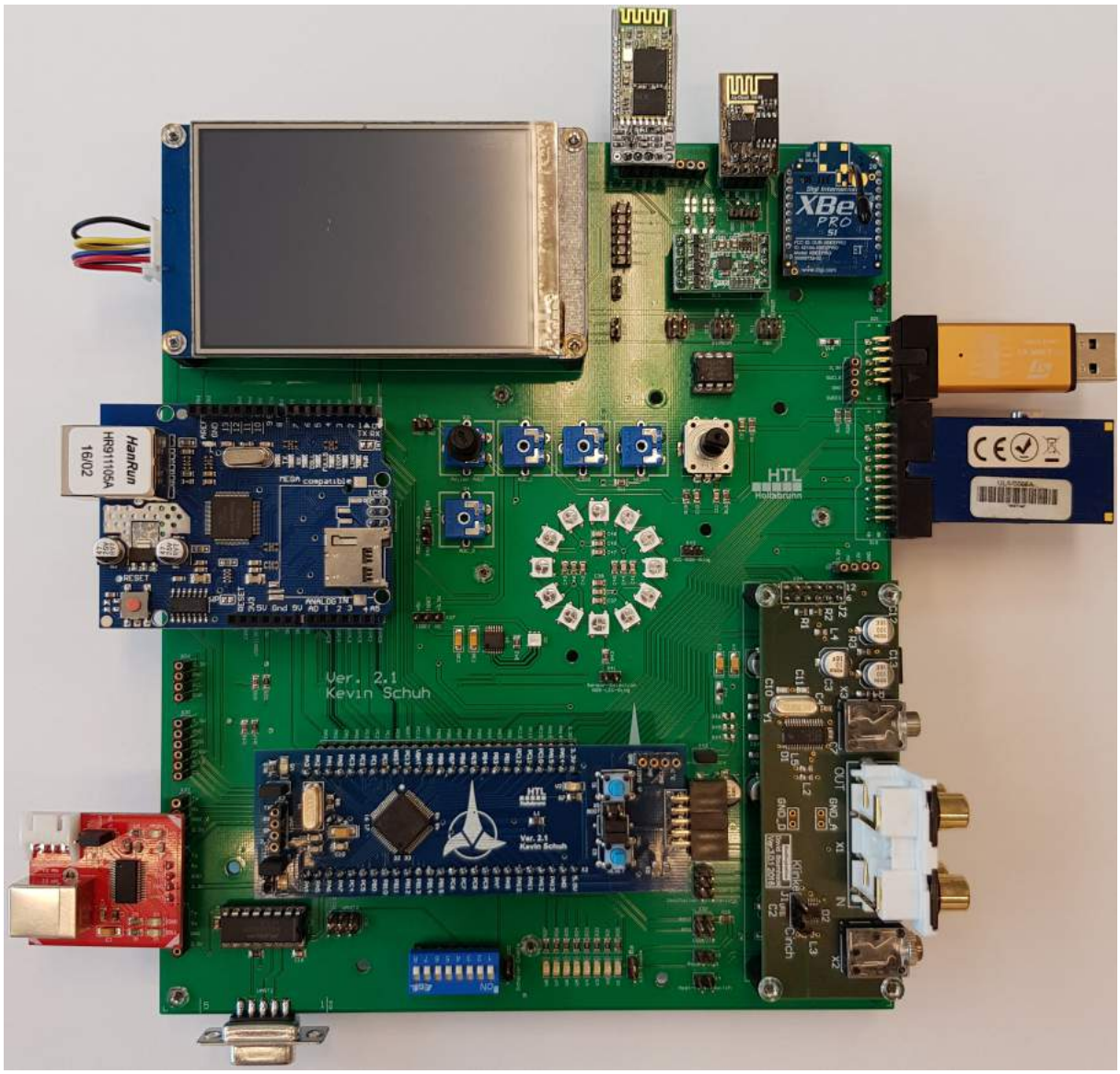


Abbildung 13: Gesamtsystem

2 Core-Modul

Abbildung 14: Core-Modul

2.1 Allgemeines

Das Core-Modul ist das Herzstück des gesamten ARM-Minimalsystems, denn auf diesem befindet sich der Prozessor und alle Komponenten welche für den ordnungsmäßigen Betrieb erforderlich sind. Die einzelnen Port-Pins des Prozessors sind entweder direkt auf dem Core-Modul verwendet oder über externe Anschlüsse nach außen geführt. Weiters verfügt das Core-Modul über alle nötigen Programmierschnittstellen um unabhängig von der Basisplatine oder anderen Programmierplatinen programmiert und verwendet werden zu können. Darüber hinaus kann mit der auf dem Core-Modul befindlichen UART-Schnittstelle eine direkte Kommunikation mit anderen Modulen oder einem Terminal aufgebaut werden.

2.2 Schnittstellen

In Tabelle 2 sind die verfügbaren Schnittstellen des Core-Moduls aufgelistet. In Abbildung 15 ist dargestellt, wo auf der Platine die einzelnen Schnittstellen platziert sind.

Schnittstelle	**Funktion**
USART 1	Universal Synchronous/Asynchronous Receiver/Transmitter, Datenübertragung
SWD	Single Wire Debug (SWD), Programmierung
ST-Link V2	Programmierung auf Basis von SWD
50 poliger Header	Ausführung der Port-Pins auf die Basisplatine

Tabelle 2: Schnittstellen des Core-Moduls

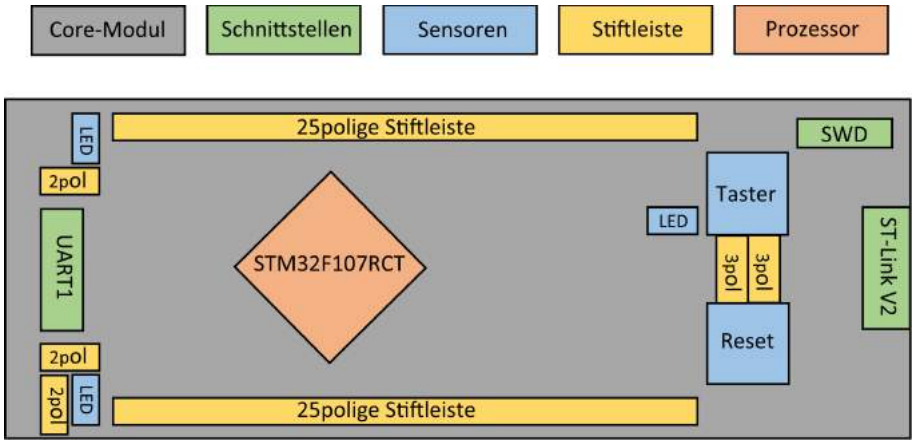


Abbildung 15: Übersichtsplan des Core-Moduls

2.3 Prozessor

Als Prozessor für das Core-Modul wurde der STM32F107RCT(6) von der Firma STMicroelectronics N.V. (STM) verwendet. Die Key-Features sind in Abbildung 16 zusammengefasst.

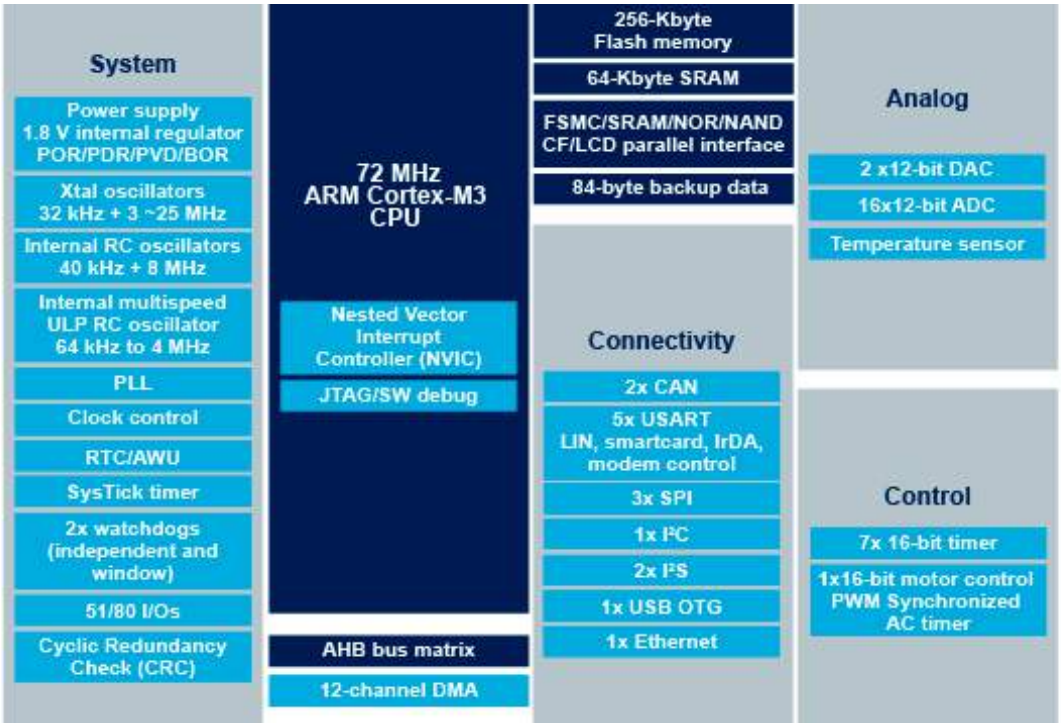


Abbildung 16: Features des Prozessors [1]

2.3.1 Blockschaltbild

Im Gegensatz zum bisher verwendeten STM32F103RB verfügt der STM32F107RCT(6) über zwei I²S-Schnittstellen, welche für den Audioadapter (Abschnitt 5) verwendet werden. Des weiteren verfügt der neue Microcontroller einen Digital-Analog-Converter (DAC), einen Ethernet-Controller (MAC) und ist USB OTG-fähig.

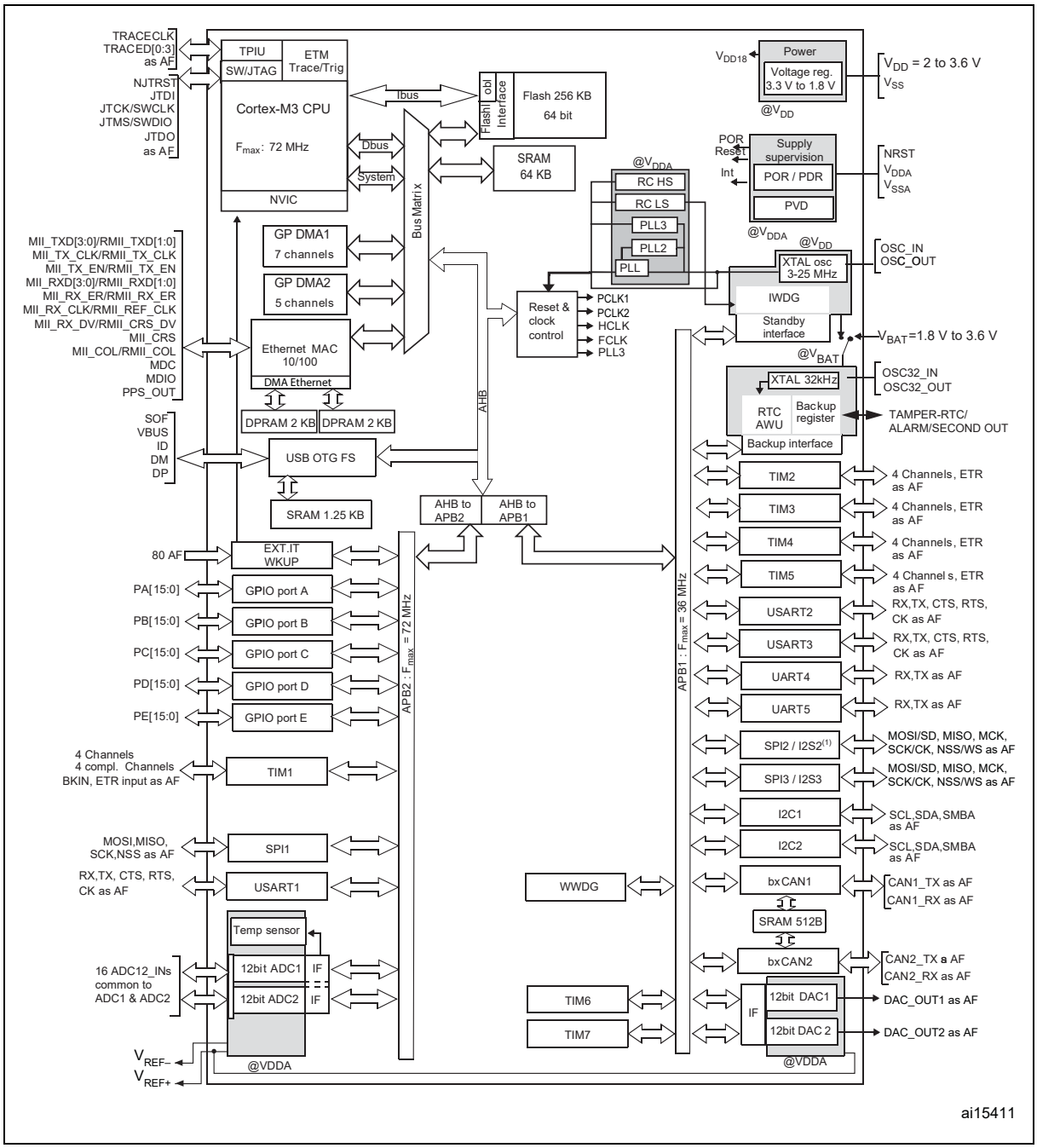


Abbildung 17: Blockschaltbild des Prozessors [2]

2.3.2 Pinning

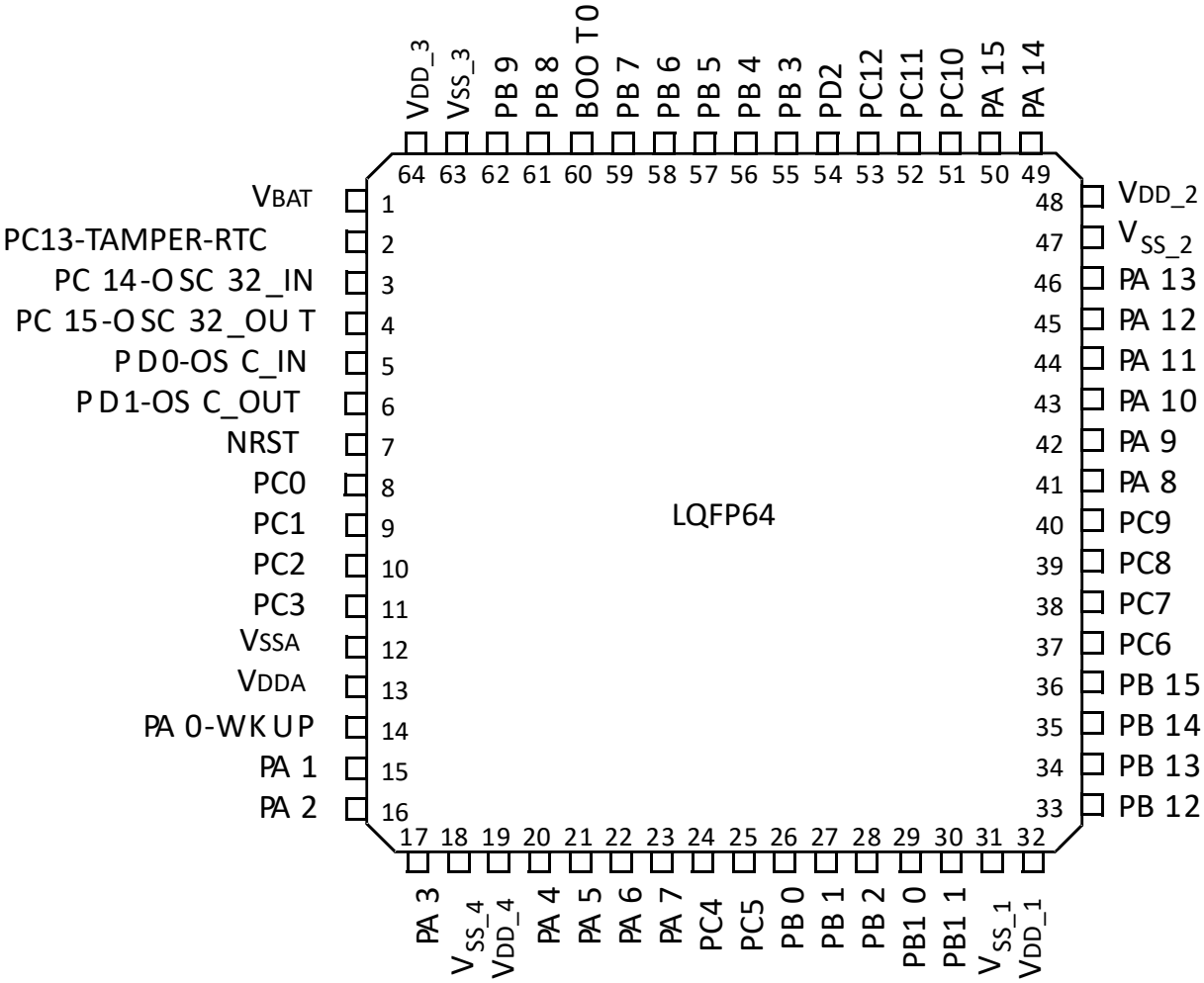


Abbildung 18: Pinning des Prozessors [2]

2.3.3 Abmessungen

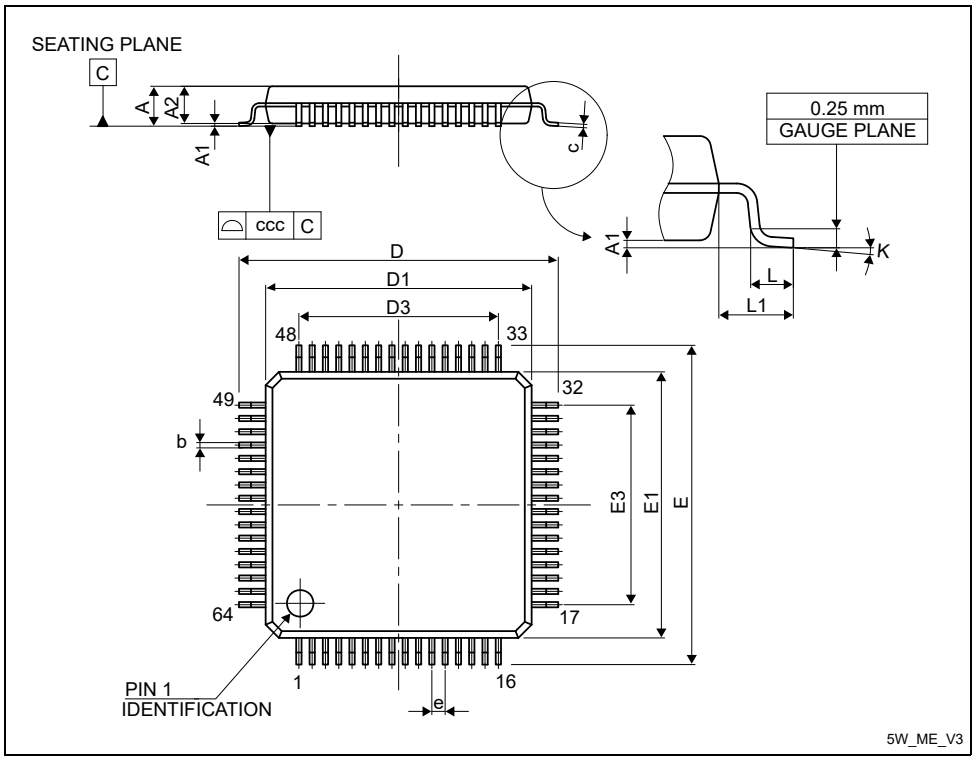


Abbildung 19: Abmessungen des Prozessors [2]

Symbol	millimeters			inches(1)		
	Min	Typ	Max	Min	Typ	Max
A	-	-	1.600	-	-	0.0630
A1	0.050	-	0.150	0.0020	-	0.0059
A2	1.350	1.400	1.450	0.0531	0.0551	0.0571
b	0.170	0.220	0.270	0.0067	0.0087	0.0106
c	0.090	-	0.200	0.0035	-	0.0079
D	-	12.000	-	-	0.4724	-
D1	-	10.000	-	-	0.3937	-
D3	-	7.500	-	-	0.2953	-
E	-	12.000	-	-	0.4724	-
E1	-	10.000	-	-	0.3937	-
E3	-	7.500	-	-	0.2953	-

Tabelle 3: Abmessungen des Prozessors [2]

2.3.4 Pinbelegung

LQFP64 PinNr.	Pin Name	Type	Main Function (after Reset)	Alternate Function	Remap
14	PA0-WKUP	I/O	PA0	WKUP/USART2_CTS/ ADC12_IN0/ TIM2_CH1_ETR/TIM5_CH1/ ETH_MII_CRS_WKUP	-
15	PA1	I/O	PA1	USART2_RTS/ ADC12_IN1/ TIM5_CH2/TIM2_CH2/ ETH_MII_RX_CLK/ ETH_RMII_REF_CLK	-
16	PA2	I/O	PA2	USART2_TX/ TIM5_CH3/TIM2_CH3/ ADC12_IN2/ ETH_MII_MDIO/ETH_RMII_MDIO	-
17	PA3	I/O	PA3	USART2_RX TIM5_CH4/TIM2_CH4/ ADC12_IN3/ ETH_MII_COL	-
20	PA4	I/O	PA4	SPI1_NSS/ DAC_OUT1/ USART2_CK/ ADC12_IN4	SPI3_NSS/
21	PA5	I/O	PA5	SPI1_SCK/ DAC_OUT2/ ADC12_IN5	-
22	PA6	I/O	PA6	SPI1_MISO/ ADC12_IN6/ TIM3_CH1	TIM1_BKIN
23	PA7	I/O	PA7	SPI1_MOSI_/ ADC12_IN7/ TIM3_CH2/ ETH_MII_RX_DV/ ETH_RMII_CRS_DV	TIM1_CH1N
41	PA8	I/O	PA8	USART1_CK/ OTG_FS_SOF/ TIM1_CH1/MCO	-
42	PA9	I/O	PA9	USART1_TX/ TIM1_CH2/ OTG_FS_VBUS	-
43	PA10	I/O	PA10	USART1_RX/ TIM1_CH3/ OTG_FS_ID	-
44	PA11	I/O	PA11	USART1_CTS/ CAN1_RX/ TIM1_CH4/ OTG_FS_DM	-

Tabelle 4: Pinbelegung des Prozessors [2]

LQFP64 PinNr.	Pin Name	Type	Main Function (after Reset)	Alternate Function	Remap
45	PA12	I/O	PA12	USART1_RTS/ OTG_FS_DP/ CAN1_TX/ TIM1_ETR	-
46	PA13	I/O	JTMS-SWDIO	-	PA13
49	PA14	I/O	JTCK-SWCLK	-	PA14
50	PA15	I/O	JTDI	SPI3_NSS/I2S3_WS	TIM2_CH1_ETR/ PA15/ SPI1_NSS
26	PB0	I/O	PB0	ADC12_IN8/ TIM3_CH3/ ETH_MII_RXD2	TIM1_CH2N
27	PB1	I/O	PB1	ADC12_IN9/ TIM3_CH4/ ETH_MII_RXD3	TIM1_CH3N
28	PB2	I/O	PB2/ BOOT1	-	-
55	PB3	I/O	JTD0	SPI3_SCK/I2S3_CK	PB3/ TRACESW0/ TIM2_CH2/ SPI1_SCK
56	PB4	I/O	NJTRST	SPI3_MISO	PB4/ TIM3_CH1/ SPI1_MISO
57	PB5	I/O	PB5	I2C1_SMBA/ SPI3_MOSI/I2S3_SD/ ETH_MII_PPS_OUT/ ETH_RMII_PPS_OUT	TIM3_CH2/ SPI1_MOSI/ CAN2_RX
58	PB6	I/O	PB6	I2C1_SCL/ TIM4_CH1	USART1_TX/ CAN2_TX
59	PB7	I/O	PB7	I2C1_SDA/ TIM4_CH2	USART1_RX
61	PB8	I/O	PB8	TIM4_CH3/ ETH_MII_TXD3	I2C1_SCL/ CAN1_RX
62	PB9	I/O	PB9	TIM4_CH4	I2C1_SDA/ CAN1_TX
29	PB10	I/O	PB10	I2C2_SCL/ USART3_TX/ ETH_MII_RX_ER	TIM2_CH3
30	PB11	I/O	PB11	I2C2_SDA/ USART3_RX/ ETH_MII_TX_EN/ ETH_RMII_TX_EN	TIM2_CH4

Tabelle 4: Pinbelegung des Prozessors [2]

LQFP64 PinNr.	Pin Name	Type	Main Function (after Reset)	Alternate Function	Remap
33	PB12	I/O	PB12	SPI2_NSS/I2S2_WS/I2C2_SMBA/ USART_CK/ TIM1_BKIN/ CAN2_RX/ ETH_MII_TXD0/ETH_RMII_TXD0	-
34	PB13	I/O	PB13	SPI2_SCK/I2S2_CK/ USART3_CTS/ TIM_CH1N/ CAN2_TX/ ETH_MII_TXD1/ETH_RMII_TXD1	-
35	PB14	I/O	PB14	SPI2_MISO/ TIM1_CH2N/ USART3_RTS	-
36	PB15	I/O	PB15	SPI2_MOSI/I2S2_SD/ TIM1_CH3	-
8	PC0	I/O	PC0	ADC12_IN10	-
9	PC1	I/O	PC1	ADC12_IN11/ ETH_MII_MDC/ETH_RMII_MDC	-
10	PC2	I/O	PC2	ADC12_IN12/ ETH_MII_TXD2	-
11	PC3	I/O	PC3	ADC12_IN13/ ETH_MII_TX_CLK	-
24	PC4	I/O	PC4	ADC12_IN14/ ETH_MII_RXD0/ETH_RMII_RXD0	-
25	PC5	I/O	PC5	ADC12_IN15/ ETH_MII_RXD1/ETH_RMII_RXD1	-
37	PC6	I/O	PC6	I2S2_MCK	TIM3_CH1
38	PC7	I/O	PC7	I2S3_MCK	TIM3_CH2
39	PC8	I/O	PC8	-	TIM3_CH3
40	PC9	I/O	PC9	-	TIM3_CH4
51	PC10	I/O	PC10	UART4_TX	USART3_TX/ SPI3_SCK/ I2S3_CK
52	PC11	I/O	PC11	UART4_RX	USART3_RX/ SPI3_MISO
53	PC12	I/O	PC12	USART5_TX	USART3_CK/ SPI3_MOSI/ I2S3_SD
2	PC13-Tamper-RTC	I/O	PC13	TAMPER-RTC	-
3	PC14-OSC_IN	I/O	PC14	OSC32_IN	-
4	PC15-OSC_OUT	I/O	PC15	OSC32_OUT	-
54	PD2	I/O	PD2	TIM3_ETR/ USART5_RX	-

Tabelle 4: Pinbelegung des Prozessors [2]

LQFP64 PinNr.	Pin Name	Type	Main Function (after Reset)	Alternate Function	Remap
5	OSC_IN	I	OSC_IN	-	-
6	OSC_OUT	O	OSC_OUT	-	-
60	BOOT0	I	BOOT0	-	-
7	NRST	I/O	NRST	-	-
31	VSS_1	S	VSS_1	-	-
32	VDD_1	S	VDD_1	-	-
47	VSS_2	S	VSS_2	-	-
48	VDD_2	S	VDD_2	-	-
63	VSS_3	S	VSS_3	-	-
64	VDD_3	S	VDD_3	-	-
18	VSS_4	S	VSS_4	-	-
19	VDD_4	S	VDD_4	-	-
12	VSSA	S	VSSA	-	-
13	VDDA	S	VDDA	-	-
1	VBAT	S	VBAT	-	-

Tabelle 4: Pinbelegung des Prozessors [2]

2.4 Portbelegungsplan

Tabelle 5: Portbelegungsplan des Core-Moduls

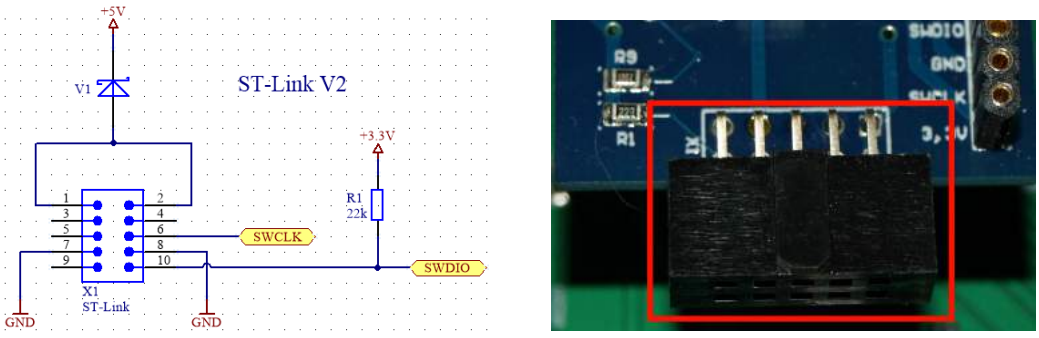
2.4.1 Programmierung mit ST-Link V2

Zur Programmierung und zum Debugging des neuen ARM-Minimalsystems sollte ein **ST-Link V2 Mini** (Abbildung 20) verwendet werden. Dieser Programmer besitzt eine verpolungssichere zweireihige Stiftreihe, welche es ermöglicht Programme mit Hilfe von

SWD auf den Microcontroller zu übertragen oder diese zu debuggen. Darüber hinaus ist der ST-Link V2 Mini der Lieferant der Hauptversorgungsspannung von +5 V.

Um den ST-Link V2 Mini und den Microcontroller im Falle eines Kurzschlusses zwischen der Versorgungsspannung und Masse zu schützen wurde eine Schottky-Diode V1 (Abbildung 21), mit einem maximalen Durchflussstrom von 1 A, vorgesehen. Um die Verpolungssicherheit des ST-Link V2 Mini zu gewährleisten, wurde eine zweireihige Buchsenleiste mit Nase X1 (Abbildung 21; Abbildung 22) verbaut, welche eine Verpolung des ST-Links unmöglich macht.

Abbildung 20: ST-Link V2 Mini



(a) Schematic (b) Hardware

Abbildung 21: ST-Link Schaltung des Core-Moduls

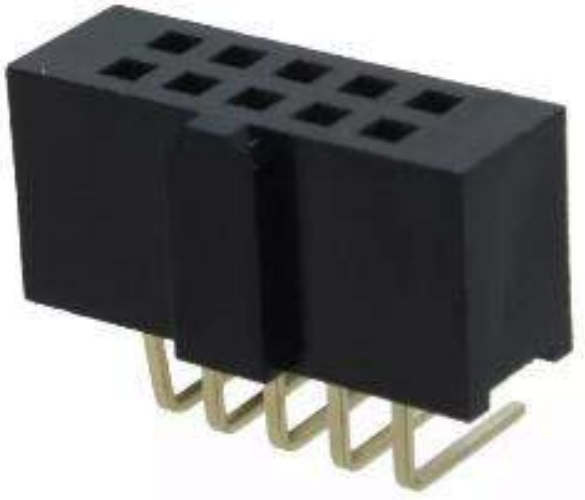
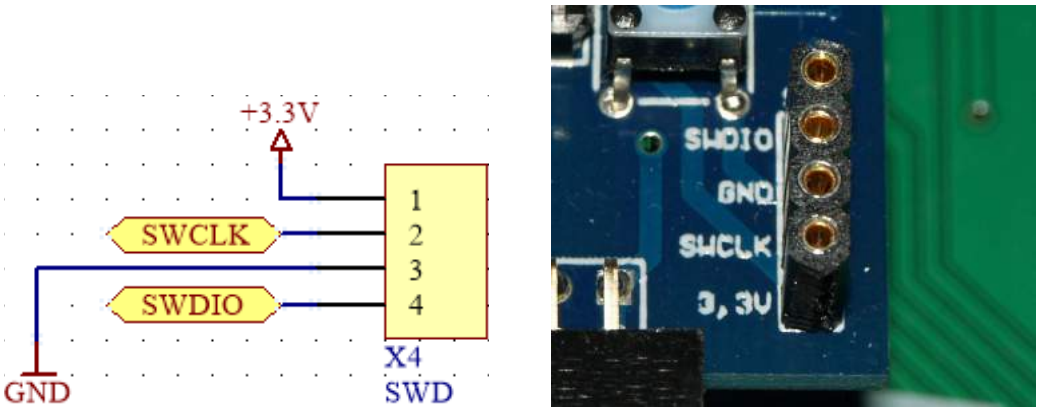


Abbildung 22: Buchse mit Nase

2.4.2 Single Wire Debug (SWD) Adapter

Der als Buchsenleiste ausgeführte SWD-Adapter X4 (Abbildung 23), erfüllt vom Prinzip her die gleiche Funktion wie der bereits in Abschnitt 2.4.1 beschriebene Stecker für den ST-Link V2 Mini. Dieser ermöglicht lediglich Kompatibilität zu anderen SWD-Programmern und Debuggern, welche diesen Stecker nicht besitzen.

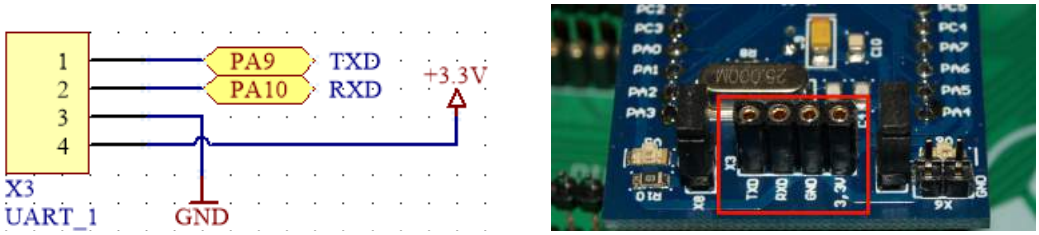


(a) Schematic (b) Hardware

Abbildung 23: SWD-Schaltung des Core-Moduls

2.4.3 Serielle Schnittstelle (USART1)

Die Buchsenleiste X3 (Abbildung 24) dient hauptsächlich zur seriellen Kommunikation. Die Pinanordnung wurde so gewählt, dass die Kommunikation entweder kabelgebunden, über den USB-to-UART Adapter, oder alternativ über ein HC-06 Bluetooth Modul erfolgen kann.



(a) Schematic (b) Hardware

Abbildung 24: UART-Schaltung des Core-Moduls

2.4.4 Bootkonfiguration

Mit Hilfe des zweireihigen Bootjumpers X5 (Abbildung 25) kann der Benutzter entscheiden von welchem Speichermedium der Cortex booten soll.

Die Standardkonfiguration sieht vor, dass man vom Flash-Speicher bootet. Daher muss, wie in Tabelle 6 beschrieben, Boot 0 auf GND gesetzt werden und Boot 1 auf +3.3 V.

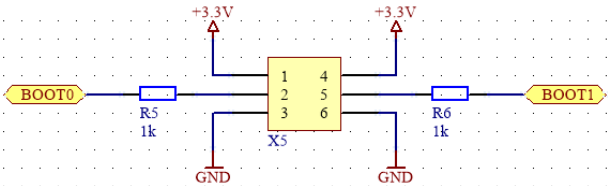


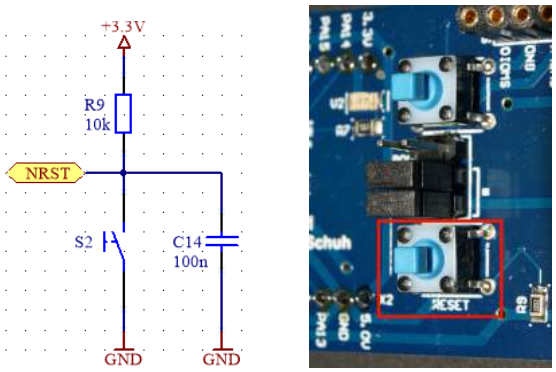
Abbildung 25: Boot-Schaltung des Core-Moduls

Boot 0	Boot 1	Funktion
+3, 3 V	+3, 3 V	Booten von SRAM
+3, 3 V	GND	Booten von Systemspeicher
GND	x	Booten von Flash-Speicher

Tabelle 6: Bootkonfigurationen des Core-Moduls

2.4.5 Reset

Der Kurzhubtaster S2 (Abbildung 26) dient zum Reset des Core-Moduls. Mit Hilfe dieses ist ein erneuter Programmstart möglich, da er den Pin des low-aktiven Reset gegen Masse zieht. Gegebenenfalls angeschlossene Arduino-Shields werden mit diesem Taster ebenfalls zurückgesetzt.



(a) Schematic (b) Hardware

Abbildung 26: Reset-Schaltung des Core-Moduls

2.4.6 5 V Spannungsversorgung

Die +5 V Spannungsversorgung für das Core-Modul kann auf zwei verschiedenen Wegen bezogen werden. Entweder man verwendet den ST-Link V2 Mini als 5 V Spannungsversorgung X1 (Abbildung 21), wie bereits in Abschnitt 2.4.1 beschrieben, oder man speist die +5 V Versorgung über den 5 V-Pin des 50-poligen Headers des Core-Moduls X2 (Abbildung 27) ein.

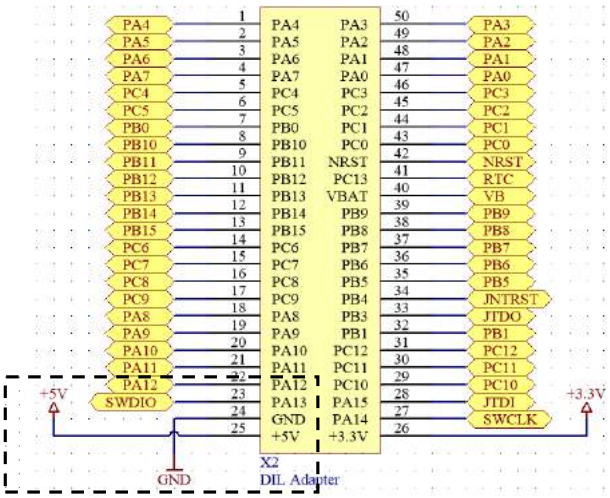


Abbildung 27: Spannungsversorgungsschaltung des Core-Moduls

2.4.7 Batterieversorgung

Am VB-Anschluss des 50-poligen Headers (Abbildung 27) kann eine 3, 3 V-Batterie angeschlossen werden, damit die Echtzeituhr (RTC) auch dann mit Spannung versorgt ist, wenn der Cortex-M3 außer Betrieb ist. Darüber hinaus wird im Falle des stromsparenden Deep-Sleep-Mode des Cortex-M3 ebenfalls eine Puffer-Batterie benötigt. Die Schottky-Diode V4 (Abbildung 28) soll verhindern, dass die Batterie welche die RTC des Cortex betreiben soll, nicht geladen wird. In den meisten Fällen handelt es sich bei dieser Batterie um eine Knopfzelle, welche häufig nicht wiederaufladbar ist. Die Schottky-Diode V3 (Abbildung 28) soll verhindern, dass die Batteriespannung der 3, 3 V-Knopfzelle, mit der über einen Linearregler generierten Versorgungsspannung des Prozessors verbunden wird.

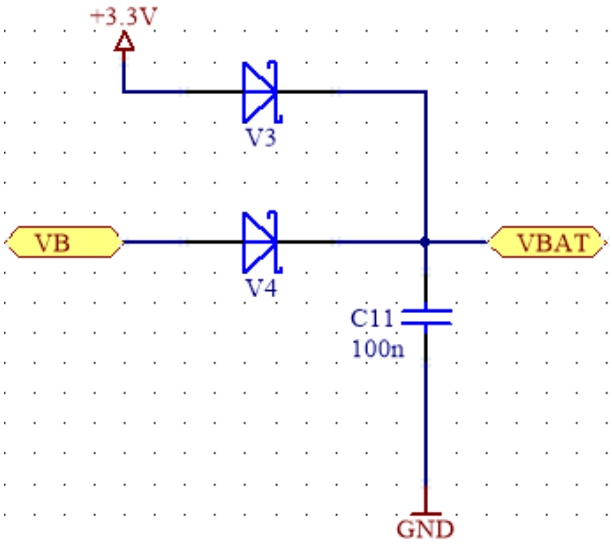
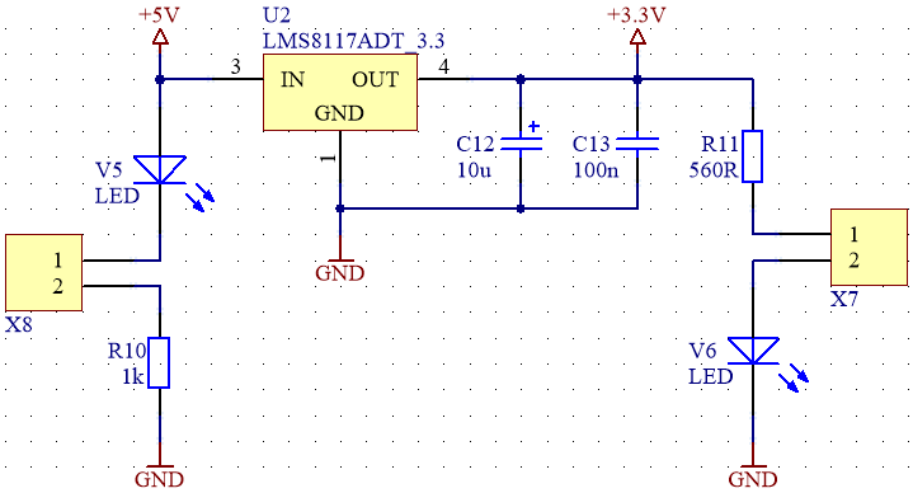


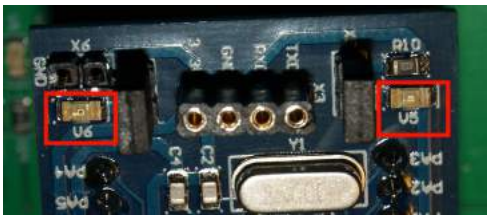
Abbildung 28: Batterieversorgungsschaltung des Core-Moduls

2.4.8 3,3 V Fixspannungsregler

Zur Generierung der für den Prozessor erforderlichen Betriebsspannung wurde ein 3, 3 V-Linearregler U2 (Abbildung 29) verwendet, welcher die Eingangsspannung von +5 V auf +3, 3 V herabsetzt. Bei Verwendung eines Linearreglers ist zu beachten, dass dieser die Spannungsdifferenz zwischen Ausgangsspannung und Eingangsspannung in Wärme umwandelt. Daher sollten keine temperaturempfindlichen Bauteile in dessen Nähe platziert werden. Zur Überprüfung, ob das Modul mit der Betriebsspannung von +5 V versorgt wird, wurde die LED V5 (Abbildung 29) zur optischen Kontrolle eingebaut. Wenn das Modul mit Spannung versorgt wird, leuchtet diese. Zur Überprüfung ob der Prozessor mit seiner Betriebsspannung von 3, 3 V versorgt wird, wurde die LED V6 (Abbildung 29) zur optischen Kontrolle eingebaut. Wenn der Prozessor mit Spannung versorgt wird, leuchtet sie. Um den Stromverbrauch des Core-Moduls im Low-Power Modus weiter zu senken, können die LEDs mit Hilfe von Jumpern außer Betrieb gesetzt werden.



(a) Schematic

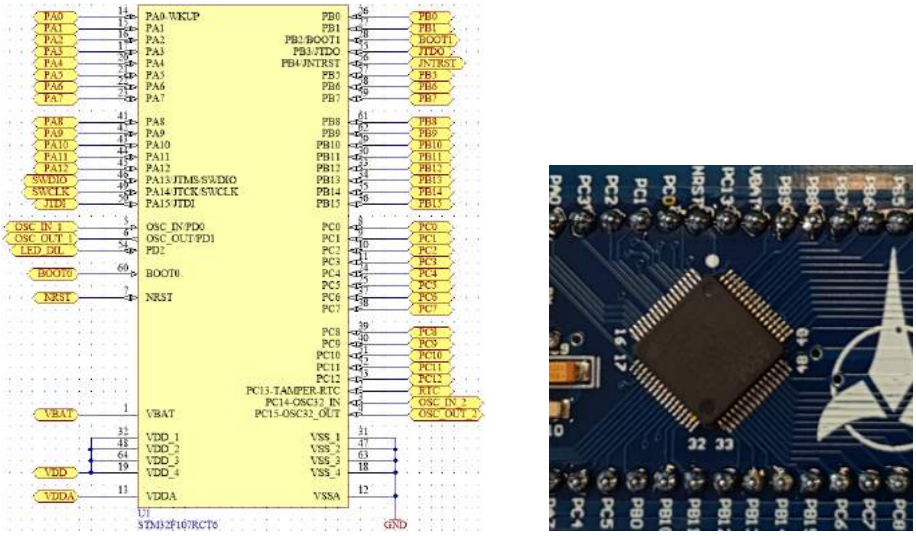


(b) Hardware

Abbildung 29: Fixspannungsregler des Core-Moduls

2.4.9 Prozessor

In Abbildung 30 ist die Pinbelegung des verwendeten Prozessors U1 dargestellt.



(a) Schematic (b) Hardware

Abbildung 30: Prozessor des Core-Moduls

2.4.10 Stützkondensatoren

Während des laufenden Betriebs eines Microcontrollers benötigt dieser unterschiedlich viel Strom. Um Spannungseinbrüche zu vermeiden wurden die Stützkondensatoren C5, C6, C7 und C8 (Abbildung 31) vorgesehen. Die Stützkondensatoren können die in ihnen gespeicherte Ladung bei stark wechselnden Stromaufnahmen abgeben und somit eine ordnungsgemäße Versorgung des Prozessors gewährleisten.

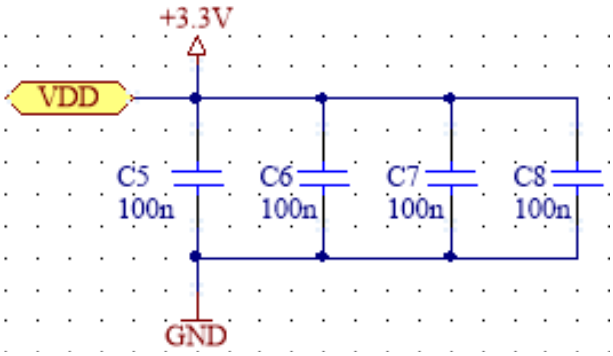


Abbildung 31: Stützkondensatoren des Core-Moduls

2.4.11 DIL-Adapter

Das Schaltplansymbol des DIL-Adapters X2 (Abbildung 32) zeigt das Pinning, welches hardwaremäßig auf den zwei getrennten Buchsenleisten auf der Leiterkarte ausgeführt wurde.

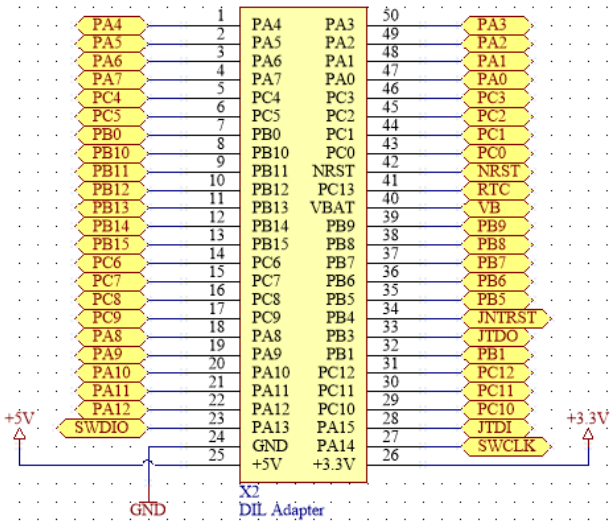


Abbildung 32: DIL-Adapter des Core-Moduls

2.4.12 Schwingquarze

Das Core-Modul besitzt standardmäßig zwei verschiedene Taktquellen. Diese bestehen aus einem 25MHz Quarz Y1 (Abbildung 33), welcher für die Taktfrequenz des Prozessors zuständig ist, und einem 32kHz Quarz Y2 (Abbildung 33), welcher für die interne RTC zur Verfügung steht.

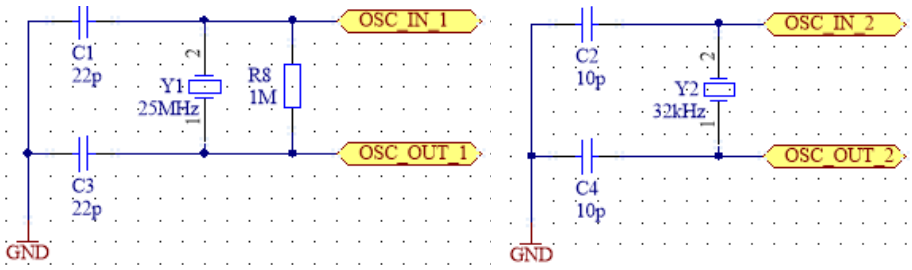
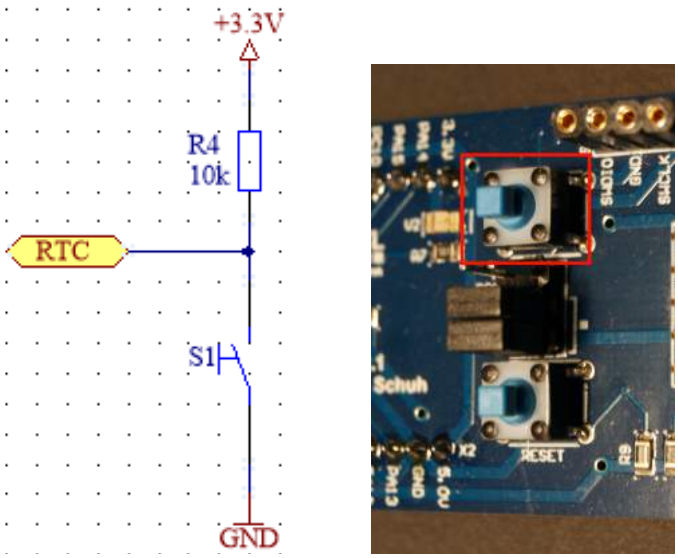


Abbildung 33: Schwingquarze des Core-Moduls

2.4.13 Taster

Auf dem Core-Modul wurde ein Kurzhubtaster S1 (Abbildung 34) vorgesehen, dessen Funktion nach belieben ausprogrammiert werden kann.

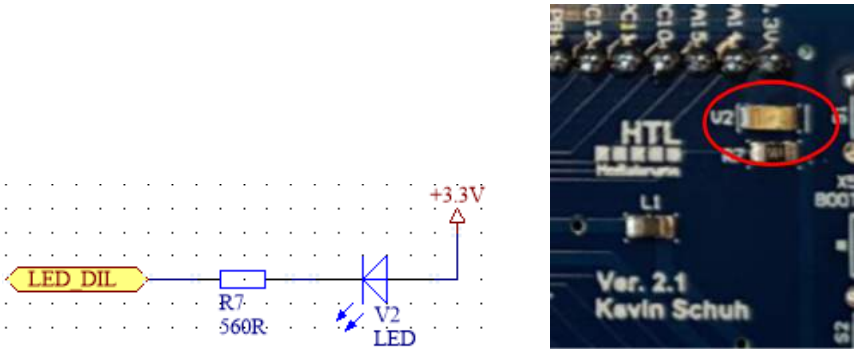


(a) Schematic (b) Hardware

Abbildung 34: Taster des Core-Moduls

2.4.14 LED

Auf dem Core-Modul wurde eine LED V2 (Abbildung 35) vorgesehen, deren Funktion nach belieben ausprogrammiert werden kann.

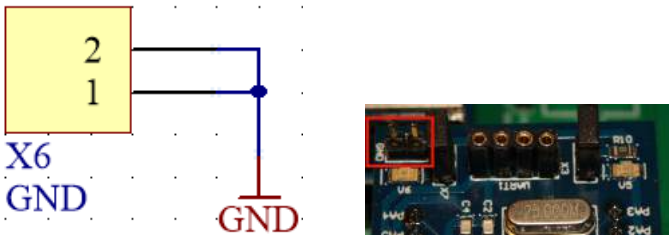


(a) Schematic (b) Hardware

Abbildung 35: LED des Core-Moduls

2.4.15 Masseschleife

Um das Messen mit einem Oszilloskop oder anderen Messgeräten zu vereinfachen wurde eine Masseschleife X6 (Abbildung 36) auf dem Core-Modul vorgesehen.



(a) Schematic (b) Hardware

Abbildung 36: Masseschleife des Core-Moduls

2.5 Gesamtschaltung

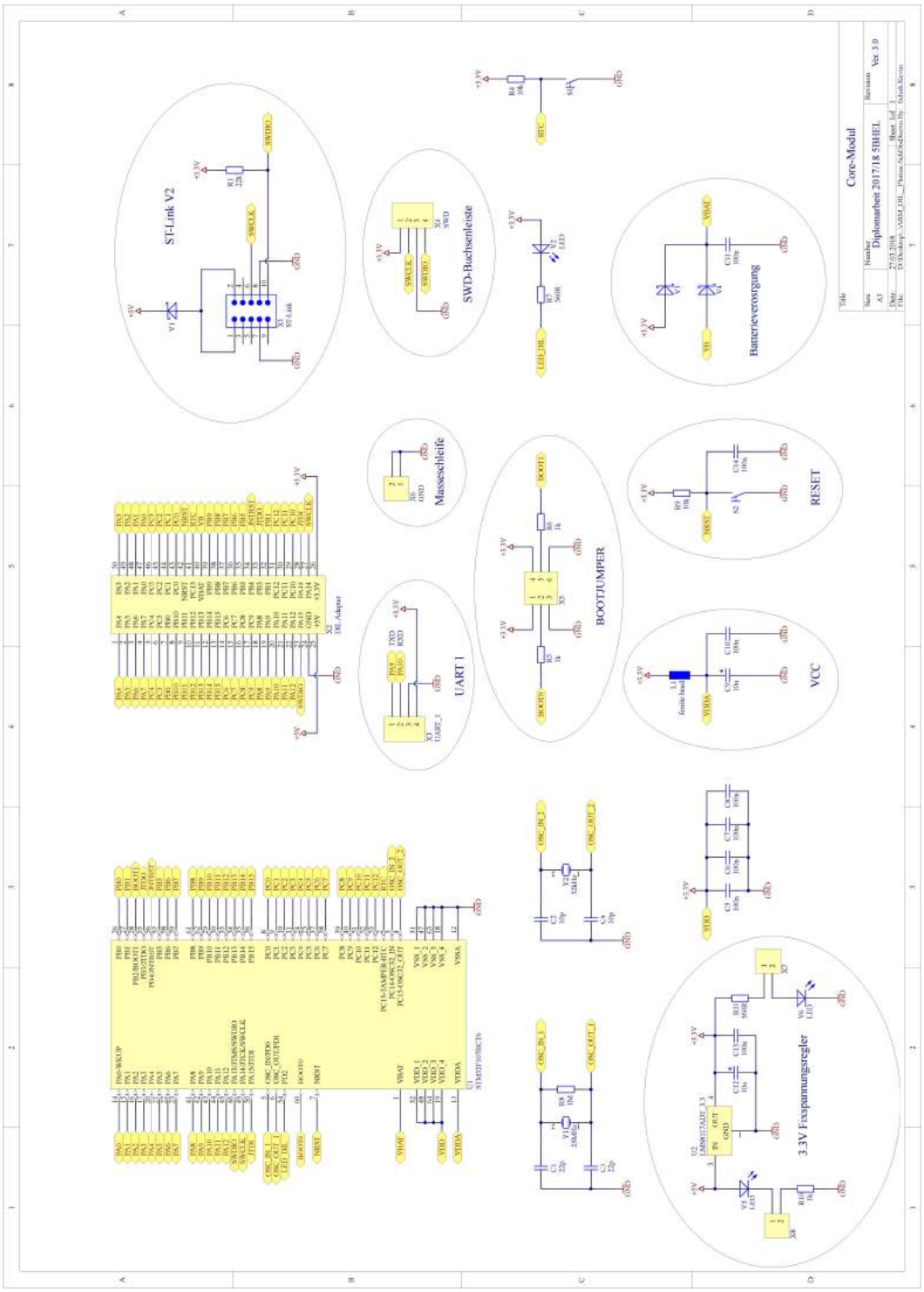


Abbildung 37: Gesamtschaltung des Core-Moduls

2.6 Leiterplattenlayout

2.6.1 Bauteilseite

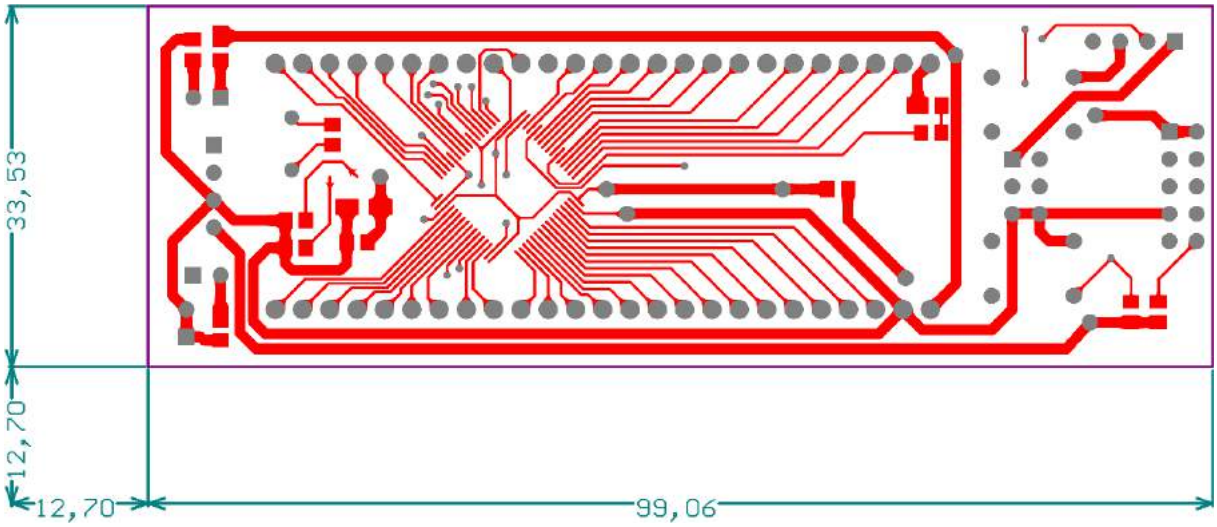


Abbildung 38: Layout Bauteilseite des Core-Moduls

2.6.2 Lötseite

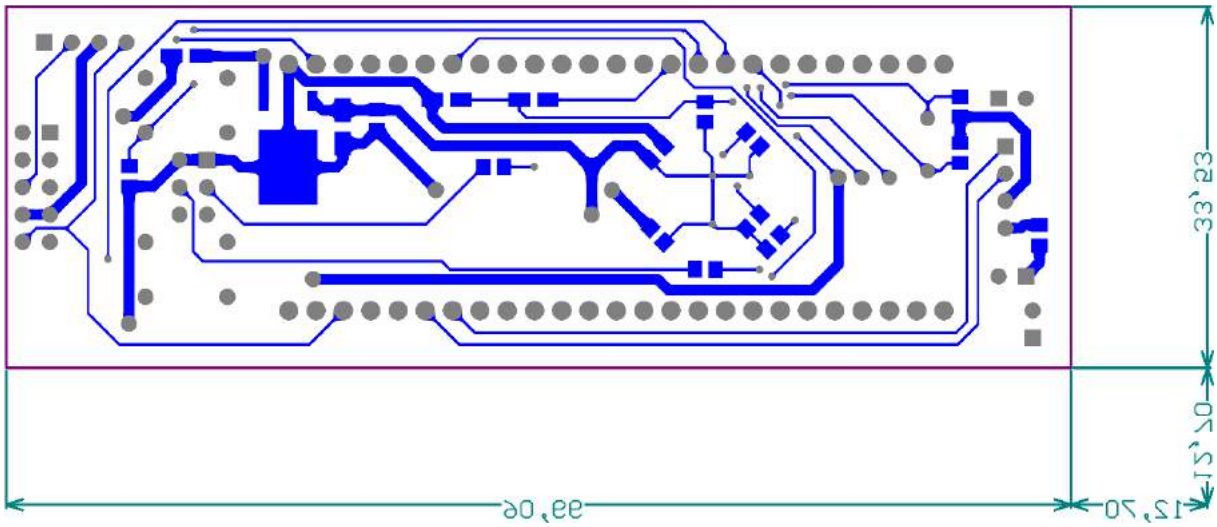


Abbildung 39: Layout Lötseite des Core-Moduls

2.7 Bestückungspläne

2.7.1 Bauteilseite

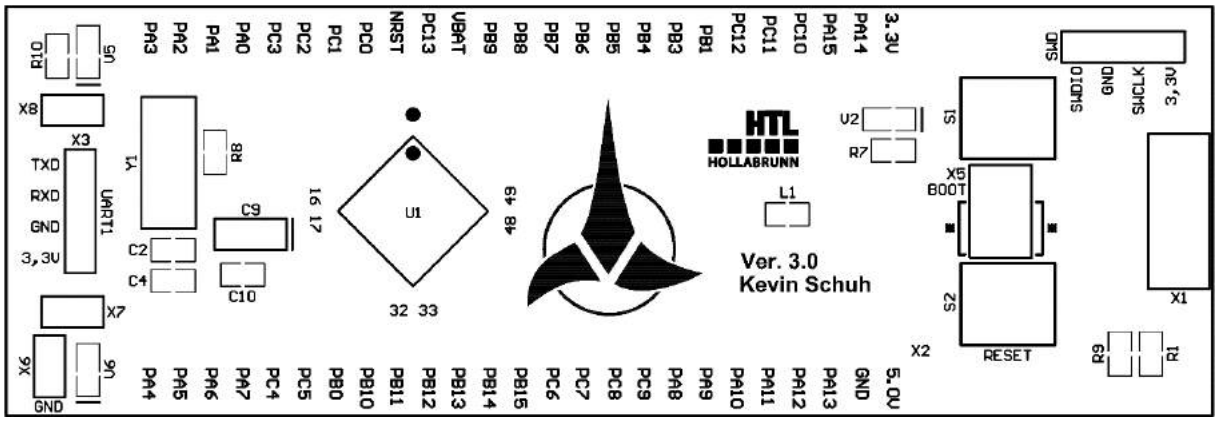


Abbildung 40: Bestückungsplan Bauteilseite des Core-Moduls

2.7.2 Lötseite

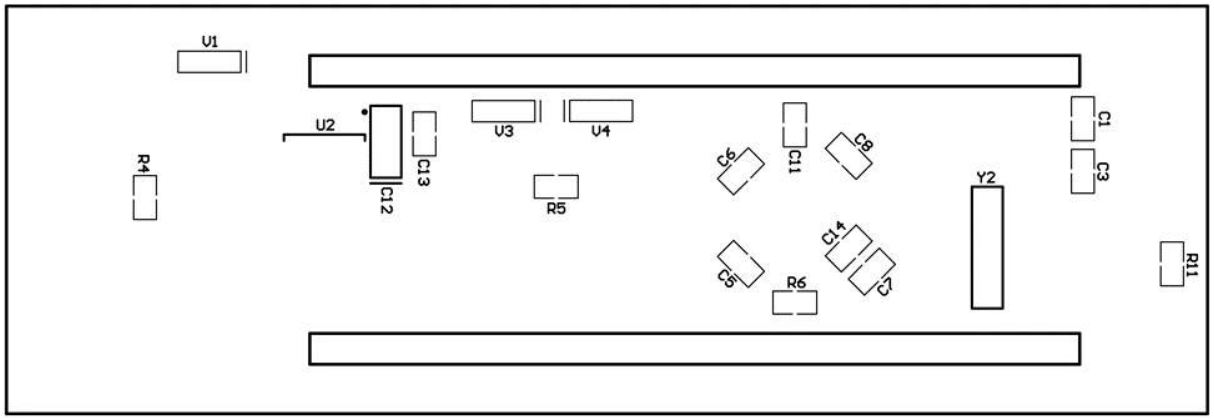


Abbildung 41: Bestückungsplan Lötseite des Core-Moduls

3 Basisplatine

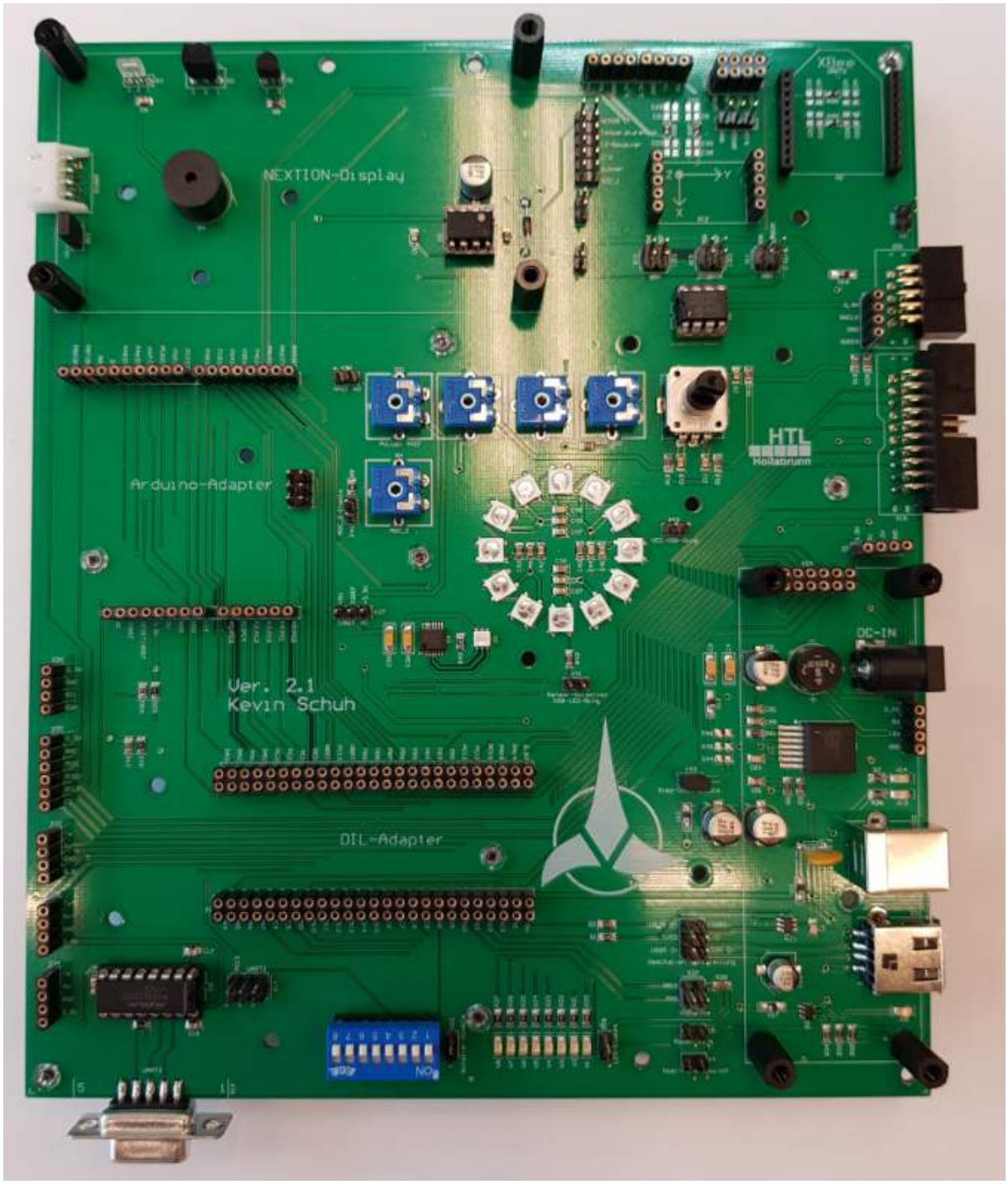


Abbildung 42: Basisplatine

3.1 Allgemeines

Die Basisplatine dient dazu dem Core-Modul eine umfangreiche, moderne und jederzeit erneuerbare Peripherie bereit zu stellen, um verschiedene Anwendungskonzepte schnell und einfach evaluieren zu können. Darüber hinaus soll mit Hilfe der Basisplatine eine Versorgung und Programmierung des Core-Moduls möglich sein. Durch Verwendung des Arduino-Shield-Konnektors können alle am Markt verfügbaren Arduino-Shields verwendet werden, daher kann man jederzeit Hardwarekomponenten tauschen oder selbst entwickeln.

3.2 Schnittstellen

Die Basisplatine verfügt über mehrere in Tabelle 7 angegebene Schnittstellen. In Abbildung 43 dargestellt wie diese auf der Basisplatine platziert sind.

Schnittstelle	**Funktion**
USART 1	Ansteuerung von HC-06, HC-12, USB-to-UART-Adapter, MAX232
USART 2	Ansteuerung von ESP8266, XBee-Pro
USART 3	Ansteuerung von Nextion-Display
SPI 1	
I^2C 1	
SWD	Programmierung auf Basis von SWD
ST-Link V2	Programmierung auf Basis von SWD
JTAG	Programmierung auf Basis von Joint Test Action Group (JTAG)
DE-9 Buchse	Ansteuerung von Nextion-Display
Arduino-Shield-Connector	Verwendung von diversen Arduino-Shields
Audio-Shield-Connector	Verwendung des HTL-internen Audio-Shields
Singe-Wire	Ansteuerung von Piezo, Temperatursensor, LFU, IR-Sensor, BMA020, EEPROM
USB-A	Spannungsversorgung, Datentransport
USB-B	Spannungsversorgung, Datentransport

Tabelle 7: Schnittstellen der Basisplatine

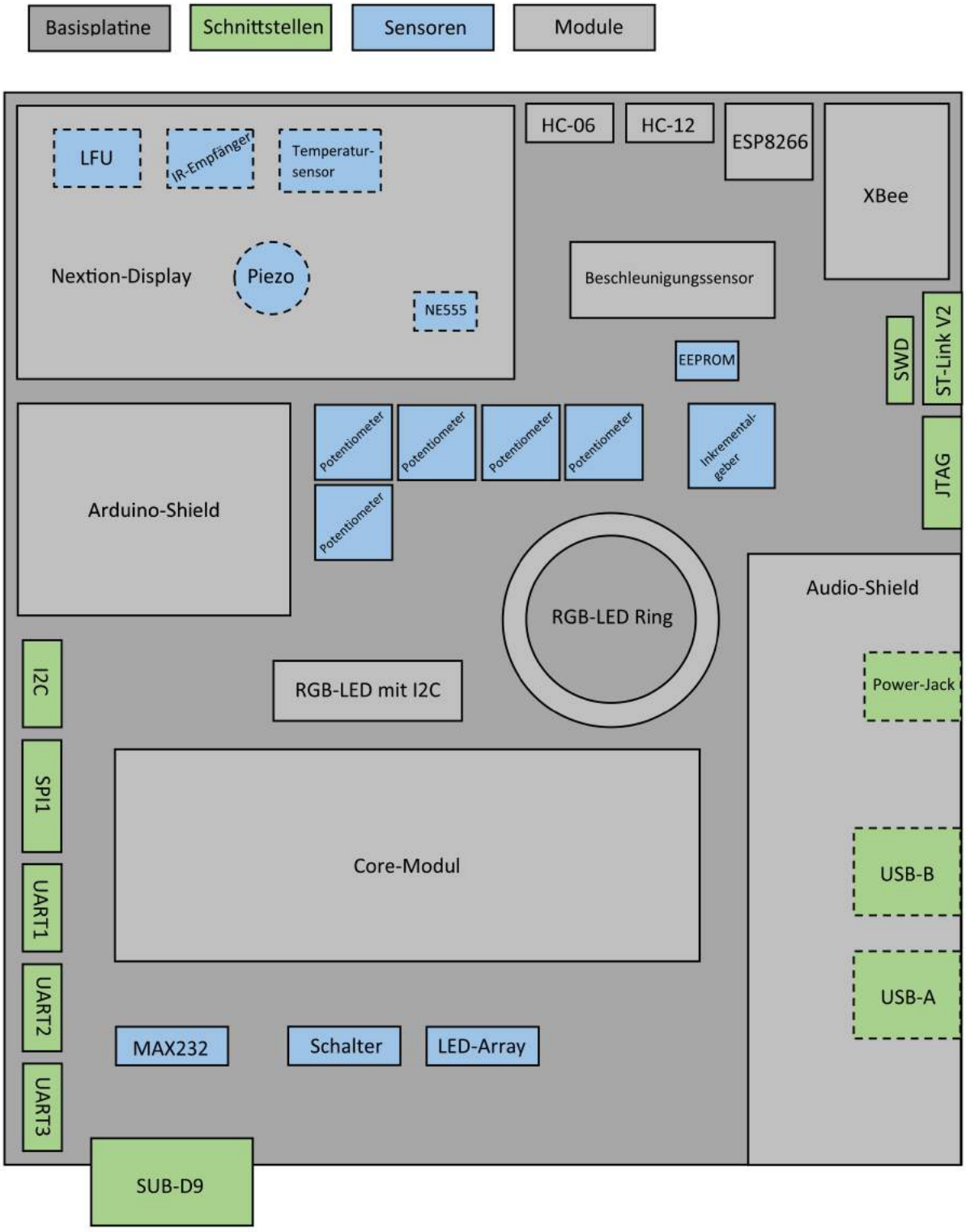


Abbildung 43: Übersichtsplan der Basisplatine

3.3 Portbelegung

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
PA			UART 2		Audio					UART 1				SWD/ST-Link V2			PA
			ESP8266/Xbee		SPI 1					HC-12/HC-067/MAX232		ESD				SPI3	
	Schalter-Array					Schalter-Array				USB-OTG				JTAG			
	Arduino		Arduino			Arduino					Arduino						
	DIL-Adapter																
PB	S-Selection	Poti		JTAG			I2C 1				UART 3		SPI 2				PB
						EEPROM/BMA020		R-Encoder			Nextion				R-Encoder		
				SPI 3			Audio						Audio			Audio	
					Arduino					Arduino							
	DIL-Adapter			DIL-Adapter													
PC		LED-Array					Audio	LED-Array			Audio		Audio				PC
										USB-OTG		R-Encoder		MAX232			
	Arduino							Arduino									
	DIL-Adapter																

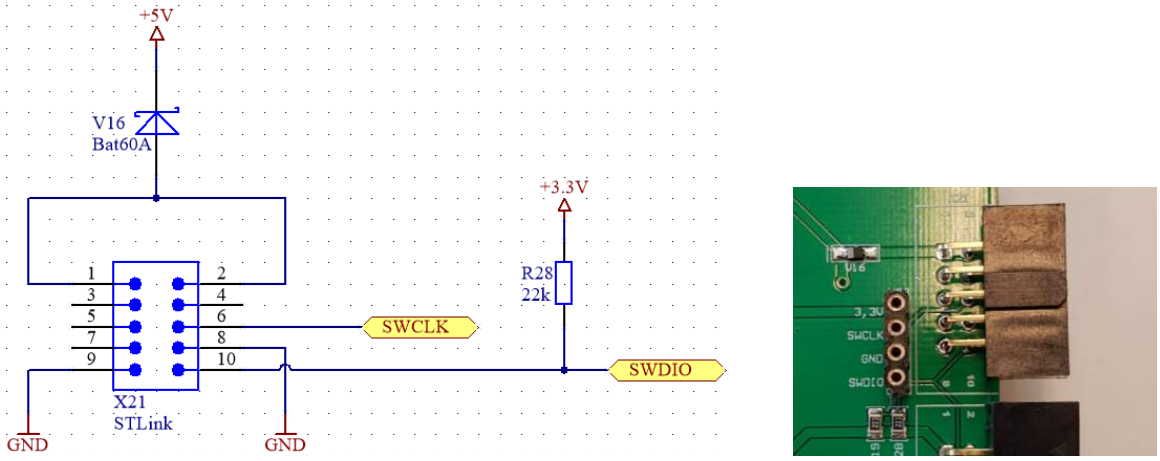
ARDUINO-Shield	Sensoren
I2C	DIL-Adaper
SPI	AUDIO-Shield
UART	JTAG
SWD/ST-Link V2	Sensor-Selection

Tabelle 8: Portbelegungsplan der Basisplatine

3.3.1 ST-Link V2

Zur Programmierung und zum Debugging des neuen ARM-Minimalsystems sollte ein **ST-Link V2 Mini** verwendet werden. Dieser Programmer besitzt eine verpolungssichere zweireihige Stiftreihe, welche es ermöglicht Programme mit Hilfe von SWD auf den Microcontroller zu übertragen oder diese zu debuggen. Darüber hinaus ist der ST-Link V2 Mini der Lieferant der Hauptversorgungsspannung von +5 V.

Um den ST-Link V2 Mini und den Microcontroller im Falle eines Kurzschlusses zwischen der Versorgungsspannung und Masse zu schützen wurde eine Schottky-Diode V16 (Abbildung 44), mit einem maximalen Durchflussstrom von 3 A, vorgesehen.

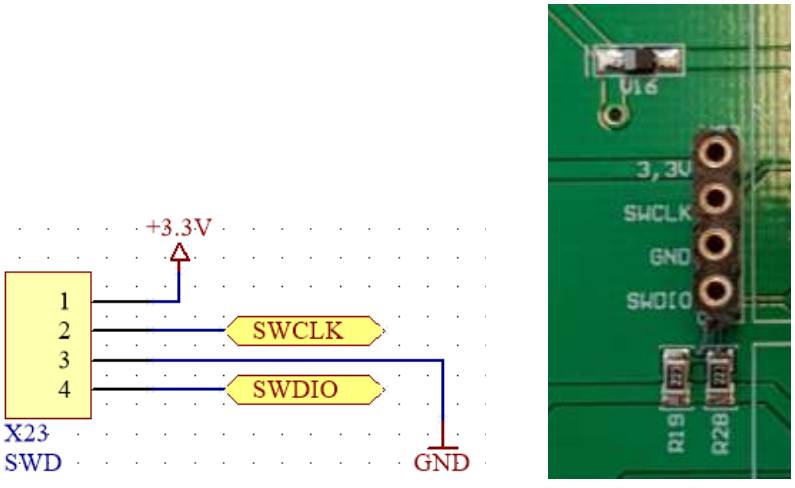


(a) Schematic (b) Hardware

Abbildung 44: ST-Link Schaltung der Basisplatine

3.3.2 SWD-Adapter

Der als Buchsenleiste ausgeführte SWD-Adapter X23 (Abbildung 45), erfüllt vom Prinzip her die gleiche Funktion wie der bereits in Abschnitt 3.3.1 beschriebene Stecker für den ST-Link V2 Mini. Dieser ermöglicht lediglich Kompatibilität zu anderen SWD-Programmern und Debuggern, welche diesen Stecker nicht besitzen.

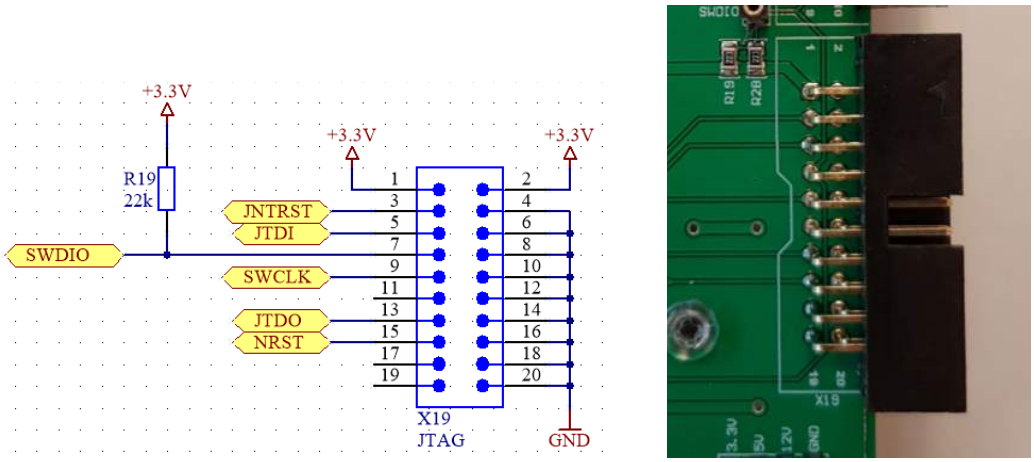


(a) Schematic (b) Hardware

Abbildung 45: SWD-Schaltung der Basisplatine

3.3.3 JTAG

Zur Programmierung und zum Debugging des neuen ARM-Minimalsystems wurde aus kompatibilitätsgründen zum alten System zusätzlich vollwertige JTAG-Schnittstelle vorgesehen, um weiterhin mit Hilfe des ULINK/ME Adapters arbeiten zu können. Auch diese Schnittstelle besitzt eine verpolungssichere zweireihige Buchsenleiste X19 (Abbildung 46), welche es ermöglicht Programme mit Hilfe des JTAG-Protokolls auf den Microcontroller zu übertragen oder diese zu debuggen.

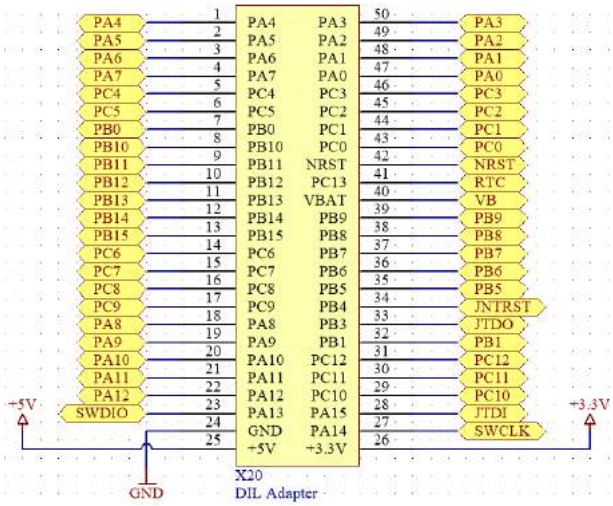


(a) Schematic (b) Hardware

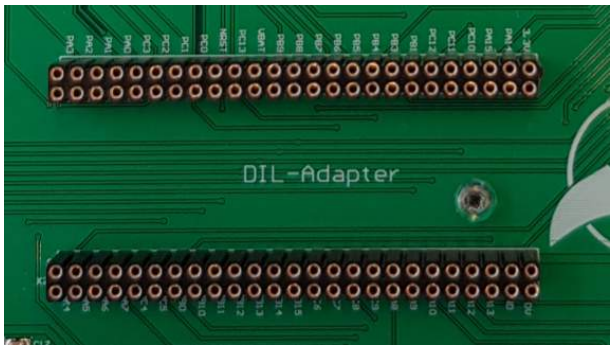
Abbildung 46: JTAG-Schaltung der Basisplatine

3.3.4 Core-Modul-Adapter

Das Schaltplansymbol des Core-Moduls X20 (Abbildung 47) zeigt das Pinning, welches auf zwei Buchsenleisten auf der Basisplatine herausgeführt wurde.



(a) Schematic

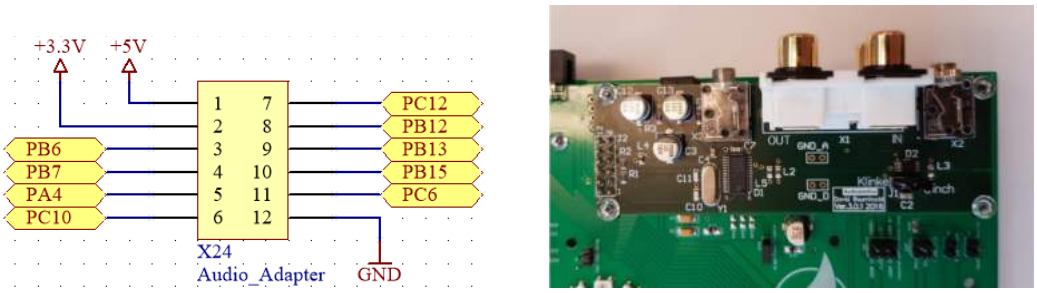


(b) Hardware

Abbildung 47: Core-Modul-Adapter der Basisplatine

3.3.5 Audio-Adapter

Der bereits in einer anderen Diplomarbeit realisierte Audio-Adapter kann auf der zweireihigen Buchsenleiste X24 (Abbildung 48) aufgesteckt und anschließend betrieben werden.

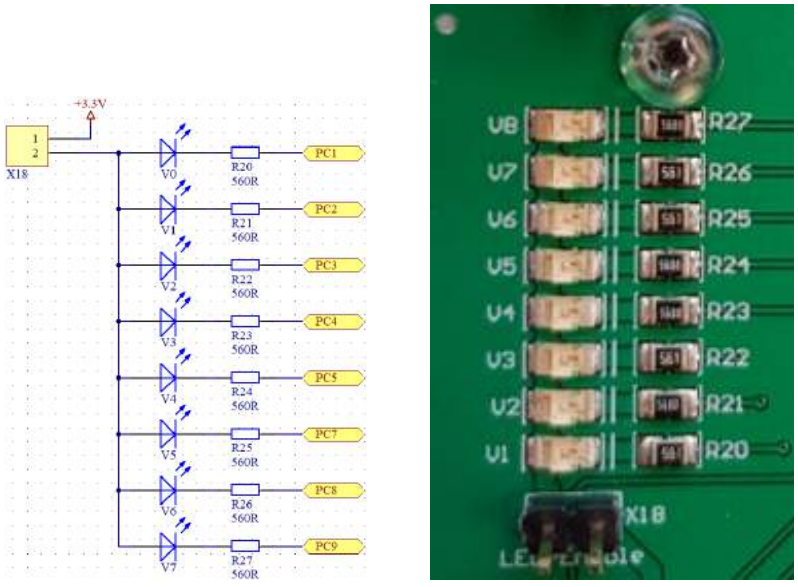


(a) Schematic (b) Hardware

Abbildung 48: Audio-Adapter der Basisplatine

3.3.6 LED-Array

Auf der Basisplatine wurde ein LED-Array realisiert, welches aus acht LEDs besteht und über die Portleitungen PC1 bis PC5 und PC7 bis PC8 (Abbildung 49) angesteuert werden können. Diese LEDs sind in SMD-Form ausgeführt und haben die einheitliche Farbe Grün. Mit Hilfe der Stiftleiste X18 (Abbildung 49) und deinem Jumper können die LEDs aktiviert oder deaktiviert werden. Wenn der Jumper entfernt ist sind die LEDs deaktiviert und die belegten Port-Pins des Core-Moduls sind wieder frei verfügbar.

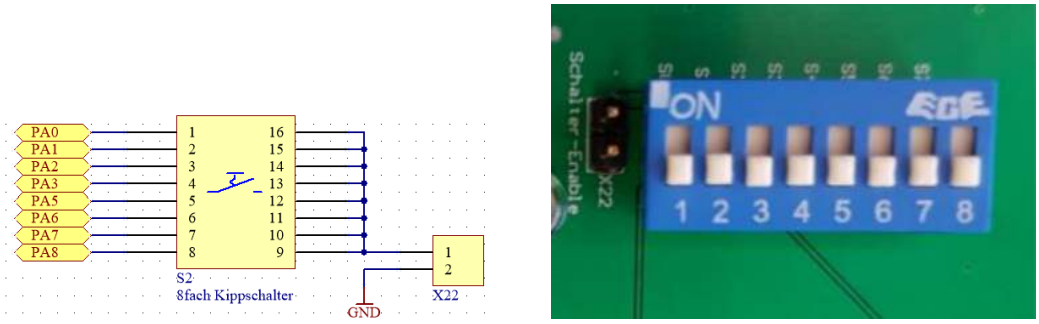


(a) Schematic (b) Hardware

Abbildung 49: LED-Array der Basisplatine

3.3.7 DIP-Switches

Auf der Basisplatine wurde ein achtpoliger DIP-Switch realisiert, welcher intern aus acht getrennten Schalten besteht und über die Portleitungen PA0 bis PA3 und PA5 bis PA8 (Abbildung 50) ausgelesen werden kann. Da im Hardwarelayout keine Pullup-Widerstände vorgesehen wurden, muss bei der Programmierung darauf geachtet werden, dass die internen Pullup-Widerstände des Prozessors aktiviert sind. Da jedoch Port Pins PA2 und PA3 mit der UART2-Schnittstelle verbunden sind, erzeugen diese ein Echo, wenn die Schalter S3 und S4 gleichzeitig geschlossen sind. Weiters können durch Jumpern der Stiftleiste X22 (Abbildung 50) alle Schalter im Kippschalter aktiviert oder deaktiviert werden.



(a) Schematic (b) Hardware

Abbildung 50: DIP-Switches der Basisplatine

3.3.8 USB-Varianten und Versorgung über USB

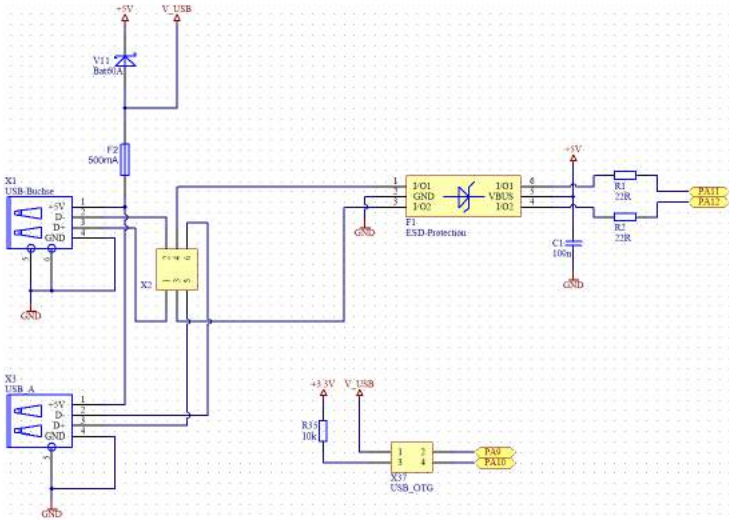
Die Basisplatine unterstützt hardwaremäßig zwei verschiedene USB-Bauformen und drei verschiedene USB-Funktionen. Diese USB-Funktionen sind USB-Device, USB-Host, USB-OTG (on the go) und werden sowohl von der USB-A Buchse X3 (Abbildung 51), als auch von der USB-B Buchse X1 (Abbildung 51) unterstützt. Da beide Buchsenformen keine ID-Leitung besitzen muss für USB-OTG die dafür benötigte Portleitung PA10 mit einem Pullup-Widerstand versehen werden, um den USB-OTG Modus vorzutäuschen. Weiters müssen für den USB-OTG Modus der Pin1 mit dem Pin2 und der Pin3 mit dem Pin4 auf der zweireihigen Stiftleiste X37 (Abbildung 51) gejumpert werden. Über die Portleitung PA10 kann ausgewählt werden ob der Prozessor als USB-Host oder USB-Device arbeitet. Mit der Portleitung PA9 hingegen kann festgestellt werden ob die Versorgung über die USB Buchse funktioniert.

Falls man im Host-Modus Geräte ohne Adapter anschließen möchte, wurde eine zweite USB Buchse parallel geschaltet. Sollte man die USB-B Buchse auswählen wollen muss man den Pin1 mit dem Pin3 und dem Pin2 mit dem Pin4 auf der zweireihigen Stiftleiste X2 (Abbildung 51) jumpern. Um die USB-A Buchse auswählen muss man den Pin4 mit

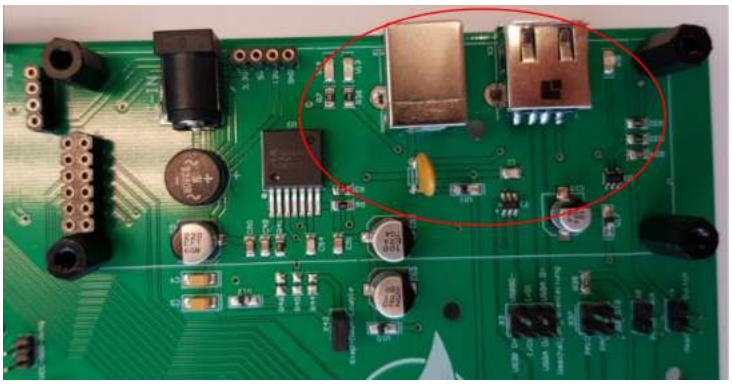
dem Pin6 und dem Pin3 mit dem Pin5 auf der zweireihigen Stiftleiste X2 (Abbildung 51) jumpern.

Die Feinsicherung F2 (Abbildung 51) dient zur Absicherung des USB-Hosts, welcher die Versorgungsspannung normalerweise zur Verfügung stellt. Laut USB-Spezifikation ist hierbei bei USB 2.0 ein maximaler Strom von 500 mA erlaubt. Die Sicherung stellt sicher, dass dieser Strom niemals überschritten wird.

Die ESD Protection F1 (Abbildung 51) stellt sicher, dass die Spannungen auf den Datenleitungen zwischen GND und 5 V bleiben. Durch an- und abstecken an Host-Geräte können Spannungsspitzen auf den Datenleitungen auftreten, die ESD Protection leitet diese ab, sodass der Prozessor nicht beschädigt wird.



(a) Schematic

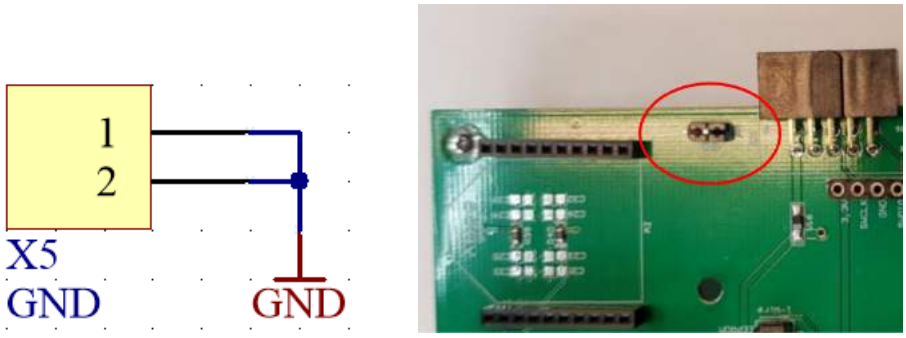


(b) Hardware

Abbildung 51: USB Buchsen der Basisplatine

3.3.9 Masseschleife

Um das Messen mit einem Oszilloskop oder anderen Messgeräten zu vereinfachen wurde eine Masseschleife X5 (Abbildung 52) auf der Basisplatine realisiert.

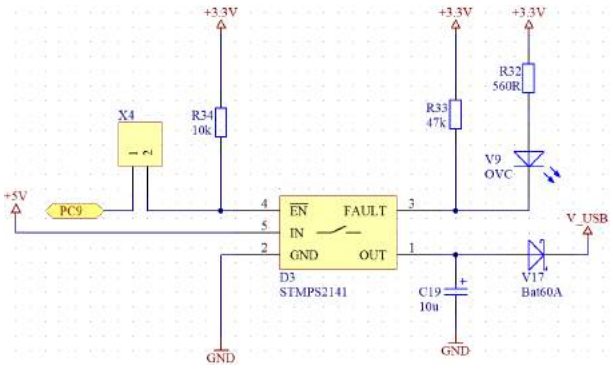


(a) Schematic (b) Hardware

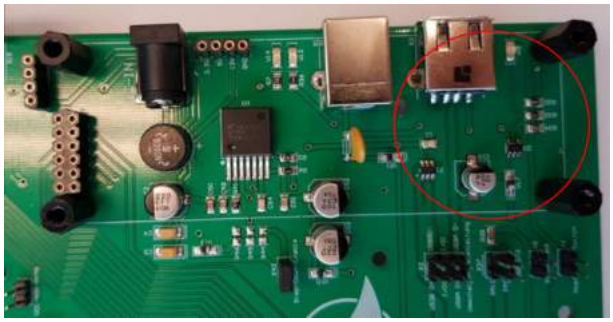
Abbildung 52: Masseschleife der Basisplatine

3.3.10 Powerswitch STMPS2141

Um USB-Geräte zu betreiben, welche selbst keine eigene Spannungsversorgung besitzen (z.B. USB-Sticks) wurde ein Powerswitch D3 (Abbildung 53) verbaut. Dieser ermöglicht es die interne 5 V-Versorgung auf die USB-Buchsen zu legen, damit diese Geräte mit Spannung versorgt werden können. Sollte das Gerät zu viel Strom verbrauchen beginnt die LED V9 (Abbildung 53) zu leuchten. Durch Jumpern der Stiftleiste X4 (Abbildung 53) an den Port-Pin PC9 kann die Versorgung von externen Geräten gesteuert werden.



(a) Schematic

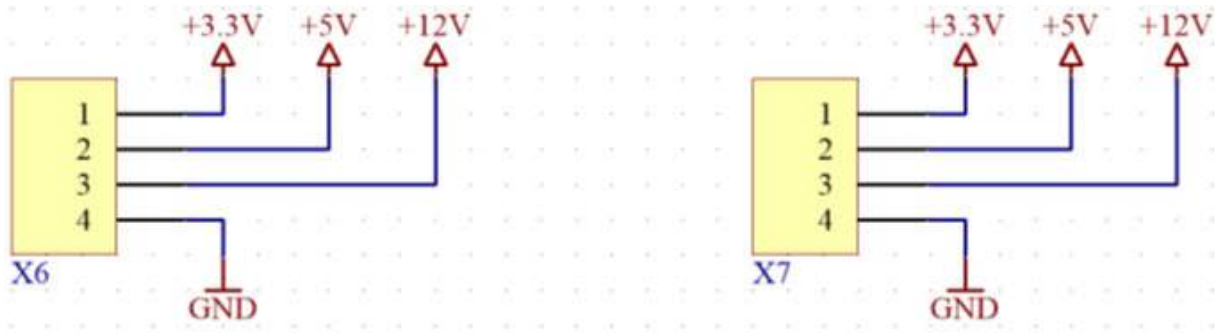


(b) Hardware

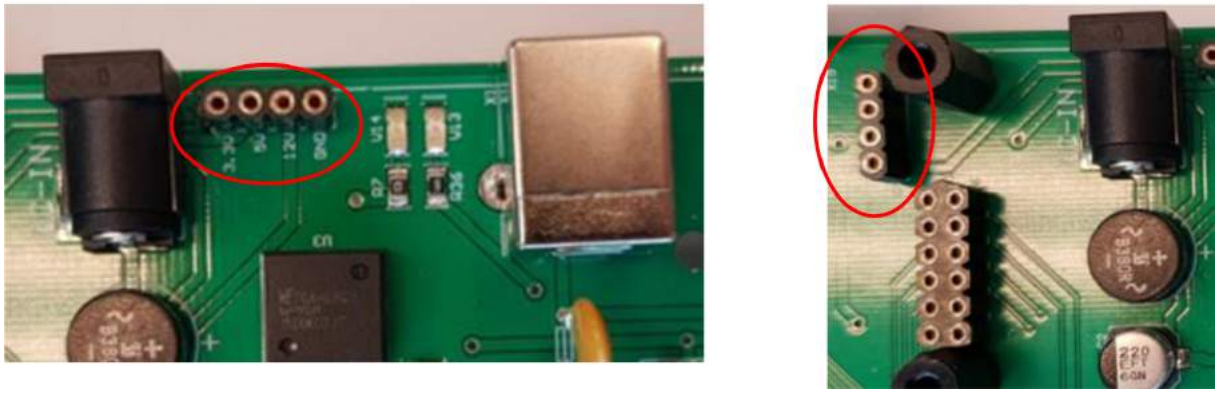
Abbildung 53: USB Powerswitch der Basisplatine

3.3.11 Powerheader

Um die auf der Basisplatine verwendeten Spannungen direkt für Versuchsaufbauten am Steckbrett oder für externe Sensoren verwenden zu können wurden zwei Buchsenleisten X6 und X7 (Abbildung 54) ausgeführt, welche diese Spannungen bereitstellen. Dabei ist zu beachten, dass am 12 V-Ausgang nur +12 V anliegen, wenn die Basisplatine über ein externes Netzteil mit einer Ausgangsspannung von +12 V betrieben wird. Sollte eine geringere Spannung über das Netzgerät eingespeist werden liegt diese am Ausgang an, wenn hingegen kein Netzteil verwendet wird ist dieser Ausgang spannungsfrei.



(a) Schematic

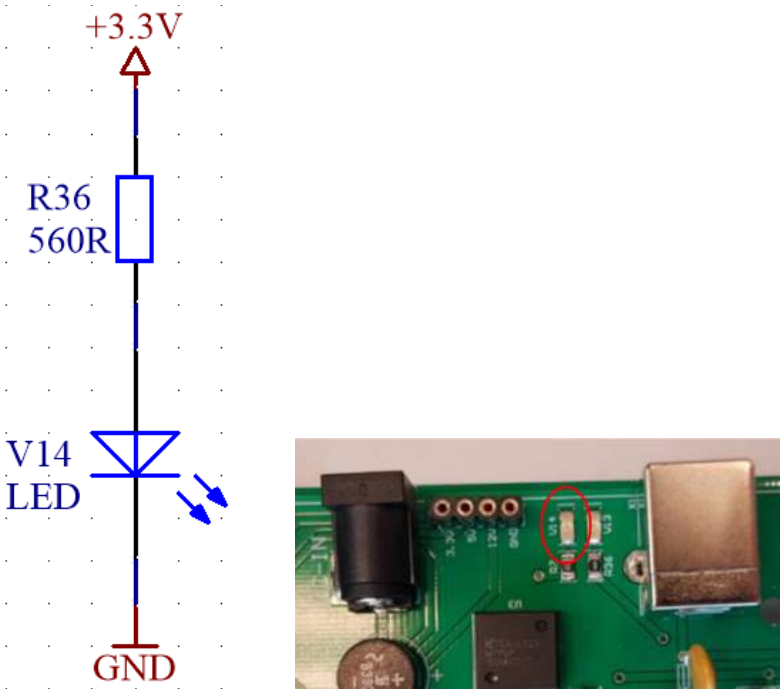


(b) Hardware

Abbildung 54: Powerheader der Basisplatine

3.3.12 3,3 V-Versorgung

Zur Überprüfung ob die Basisplatine mit der Betriebsspannung von $+3,3\,\mathrm{V}$ über das Core-Modul versorgt wird, wurde die LED V14 (Abbildung 55) zur optischen Kontrolle eingebaut. Wenn die Basisplatine mit Spannung versorgt wird, beginnt diese zu leuchten.

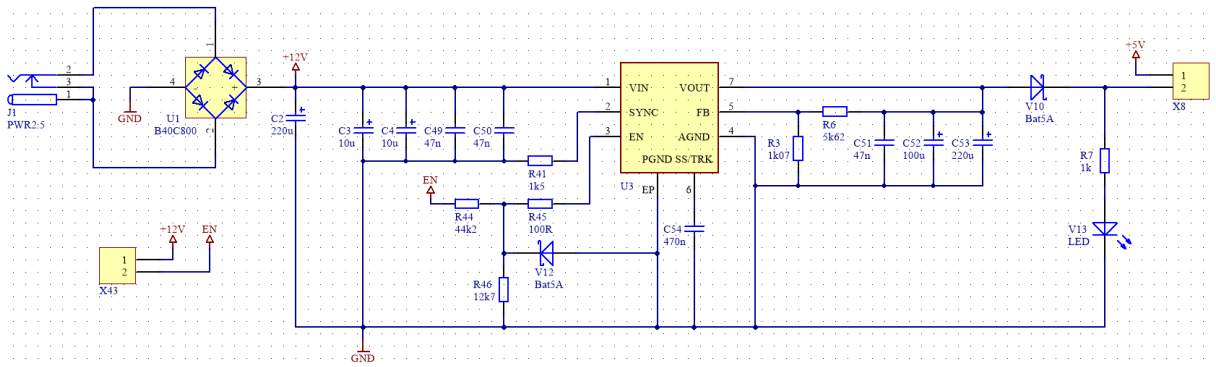


(a) Schematic (b) Hardware

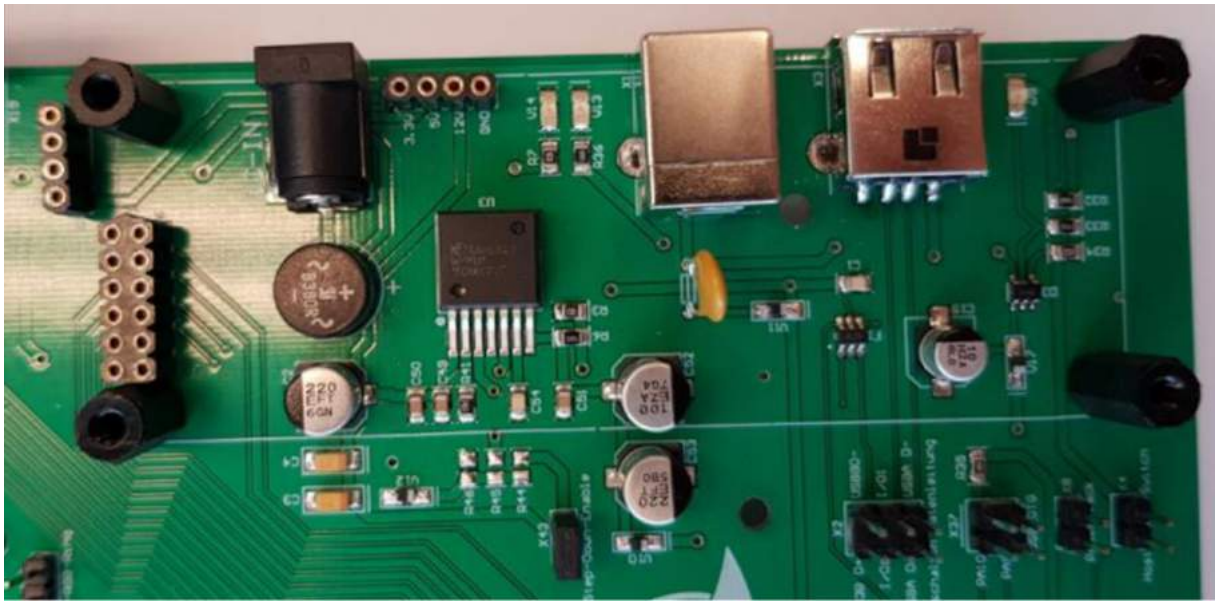
Abbildung 55: 3, 3 V-Versorgung der Basisplatine

3.3.13 Spannungsversorgung über DC-Buchse

Als weitere Spannungsversorgungsmöglichkeit bietet die Basisplatine die Möglichkeit ein 9 V bis 12 V DC-Netzteil am die DC-Buchse J1 (Abbildung 56) anzuschließen um die Basisplatine mit Spannung zu versorgen. Der Brückengleichrichter U1 (Abbildung 56) dient dabei lediglich als Verpolungsschutz und nicht als Gleichrichter für Wechselspannung. Das Anlegen von Wechselspannung an die DC-Buchse J1 (Abbildung 56) sollte aus Sicherheitsgründen unterlassen werden. Um aus der hohen Spannung, welche von der DC-Buchse kommt, die Betriebsspannung von +5 V zu generieren wurde ein Step-Down Modul U3 (Abbildung 56) der Firma Würth verbaut. Zur Überprüfung ob das Step-Down Modul die Betriebsspannung von +5 V generiert, wurde die LED V13 (Abbildung 56) zur optischen Kontrolle eingebaut. Wenn das Step-Down Modul die gewünschte Spannung generiert, beginnt diese zu leuchten. Die Stiftleiste X43 (Abbildung 56) kann im Bedarfsfall gejumpert werden, wenn ein Unterspannungsschutz der Versorgungsspannung gewünscht ist. Um die vom Step-Down Modul generierte Spannung verwenden zu können muss lediglich die Stiftleiste X8 (Abbildung 56) mit einem Jumper versehen werden.



(a) Schematic

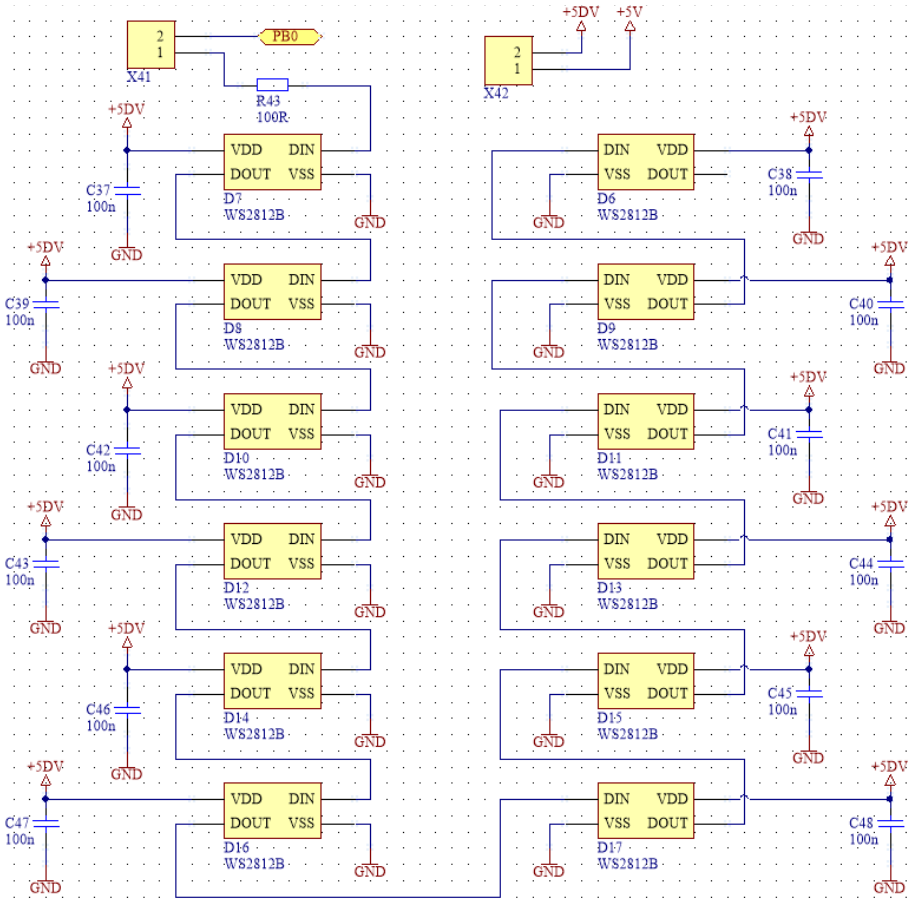


(b) Hardware

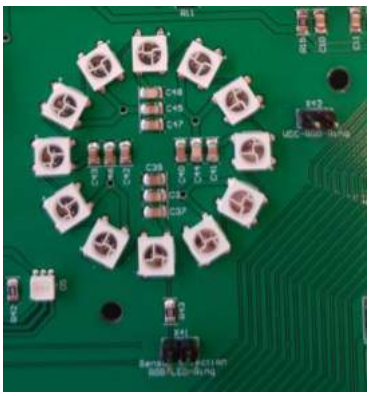
Abbildung 56: DC-Versorgung der Basisplatine

3.3.14 RGB-LED Ring [3]

Auf der Basisplatine wurde ein aus zwölf RGB-LEDs bestehender Ring aufgebaut, welcher in den verschiedensten Farben leuchten kann. Jede RGB-LED besitzt einen eigenen eingebauten Controller. Die Ansteuerung der LEDs wird über den Port-Pin PB0 realisiert. Um diesen verwenden zu können muss zuvor die Stiftleiste X41 (Abbildung 57) mit einem Jumper versehen werden. Weiters muss die Spannungsversorgung der LEDs gewährleistet sein. Dazu muss lediglich die Stiftleiste X42 (Abbildung 57) mit einem Jumper versehen werden.



(a) Schematic



(b) Hardware

Abbildung 57: RGB-LED Ring der Basisplatine

Zur Programmierung der LEDs muss über den Port PB0 ein Datenwort übetragen werden, welches 24 bit pro LED enthält, siehe Abbildung 58. Dies ergibt insgesamt 288 bit, welche übertragen werden müssen. Wie die Übertragung genau aussieht kann aus Abbildung 59 entnommen werden.

Composition of 24bit data:

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

Note: Follow the order of GRB to sent data and the high bit sent at first.

Abbildung 58: RGB-LED Ring Datenstruktur [3]

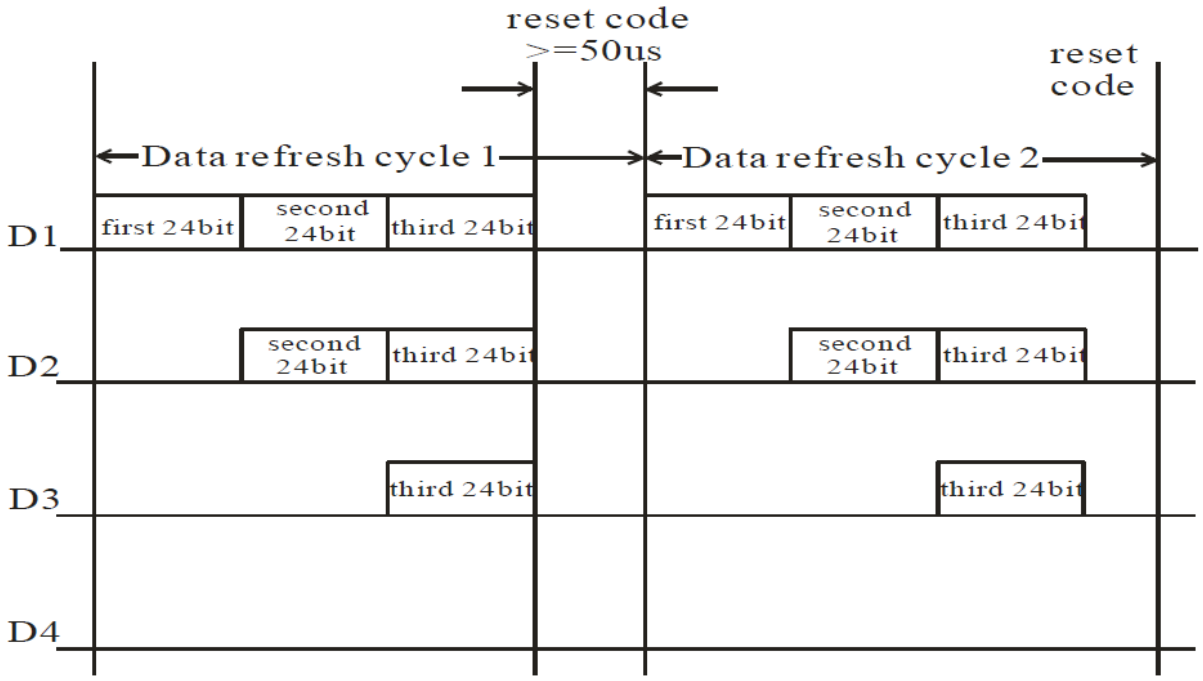


Abbildung 59: RGB-LED Ring Timing Diagram [3]

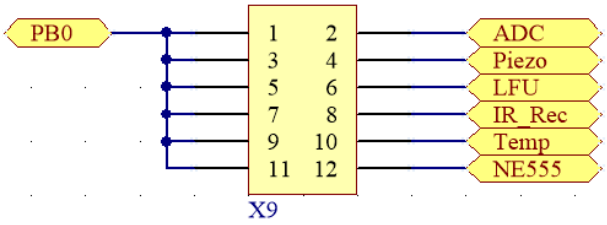
Die Daten für jede LED im Ring werden seriell hintereinander wie in Abbildung 58 beschrieben übertragen, hierbei wird zuerste die erste LED angesprochen, wenn diese die richtigen Daten hat, leitet sie alle weiteren Datenpakete transparent weiter, sodass danach die zweite LED angesprochen wird und so weiter, bis ein Reset Code (50 µs oder länger „High“) gesendet wird. Danach kann wieder die erste LED angesprochen werden. das genau Timing kann hierbai aus dem Datenblatt [3] entnommen werden.

3.3.15 Sensor-Selektion

Da der verbaute Prozessor zu wenig Portleitungen besitzt um alle Sensoren und Module mit einer eigenen Portleitung zu versorgen, werden einige Portleitungen mehrmals verwendet. Um nun einzelne Module oder Sensoren verwenden zu können, müssen diese einsprechend gejumpert werden. Eine dieser Stiftleisten ist der Header X9 (Abbildung 60), welches es ermöglicht zwischen einem Potentiometer (Analog-Digital-Converter (ADC)),

Piezo-Summer, einem LFU, einem Infrarotempfänger, einem Temperaturfühler und einem NE555 auszuwählen.

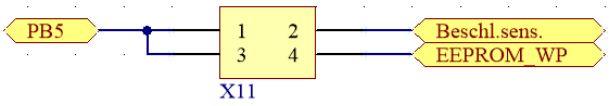
Eine weitere dieser Stiftleisten ist der Header X11 (Abbildung 61), welche es ermöglicht zwischen dem Beschleunigungssensor und dem EEPROM auszuwählen.



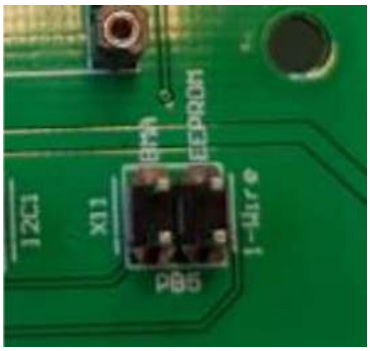
(a) Schematic

(b) Hardware

Abbildung 60: Sensor-Selektion der Basisplatine



(a) Schematic



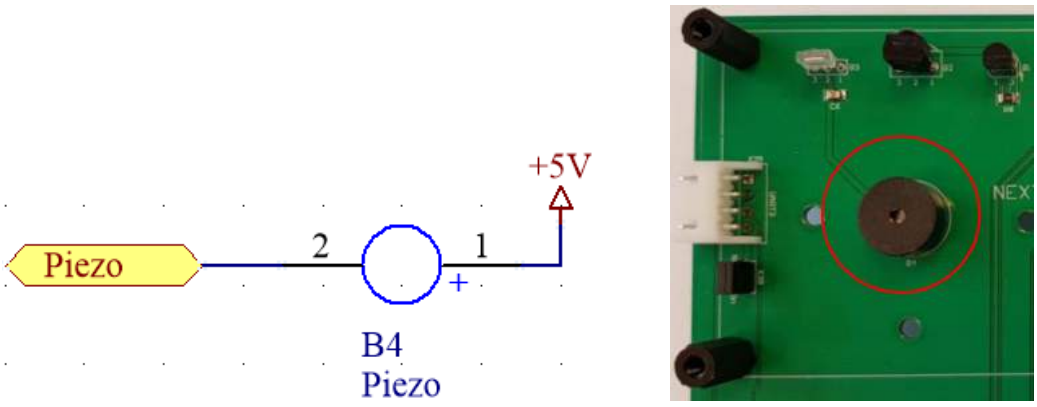
(b) Hardware

Abbildung 61: Sensor-Selektion der Basisplatine

3.3.16 Piezo-Summer

Auf der Basisplatine wurde ein Piezo-Summer B4 (Abbildung 62) verbaut, welcher über den Port-Pin PB0 angesteuert werden kann. Ja nach anliegender Taktfrequenz am Ein-

gang des Summers wird der erzeugte Ton höher oder tiefer. Um den Piezo-Summer verwenden zu können muss lediglich der Pin11 mit dem Pin12, der zweireihige Stiftleiste X9 (Abbildung 60), gejumpert werden.

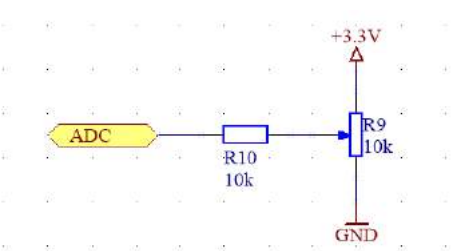


(a) Schematic (b) Hardware

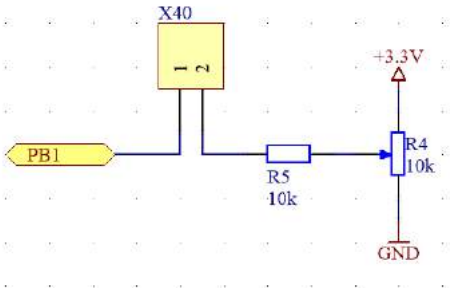
Abbildung 62: Piezo-Summer der Basisplatine

3.3.17 Potentiometer

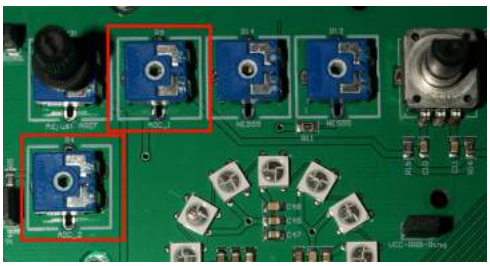
Es wurden zwei Potentiometer R9 und R4 hardwaremäßig auf der Basisplatine vorgesehen. Diese sind direkt mit einem ADC-Eingang des Prozessors verbunden. Der Unterschied zwischen den Potentiometern R9 und R4 besteht darin, dass das Potentiometer R4 über einen Jumper auf der Stiftleiste X40 (Abbildung 63b) aktiviert werden muss und dieses an einem anderen Port-Pin angeschlossen ist. Dieser Aufbau wurde deswegen gewählt da damit, damit an den ADC-Eingang auch optional ein anderes Signal angelegt werden kann.



(a) Schematic



(b) Schematic

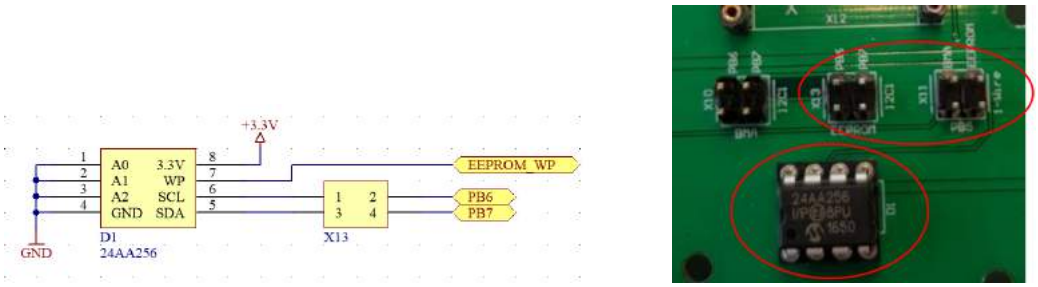


(c) Hardware

Abbildung 63: Potentiometer der Basisplatine

3.3.18 EEPROM

Um Daten permanent speichern zu können wurde das EEPROM D1 (Abbildung 64) vorgesehen. Standardmäßig wird das EEPROM 24AA256 verwendet, welches es ermöglicht 256 kbit abzuspeichern. Dieses EEPROM besitzt eine Write-Protection WP welche es ermöglicht das EEPROM schreibgeschützt zu schalten. Die Write-Protection kann über die Portleitung PB5 gesteuert werden. Damit das Signal am EEPROM ankommt, muss jedoch der Pin3 mit dem Pin4 der zweireihigen Stiftleiste X11 (Abbildung 61) gejumpert werden. Die Kommunikation mit dem EEPROM erfolgt über den I^2C-Bus (Adresse `0xA0`).

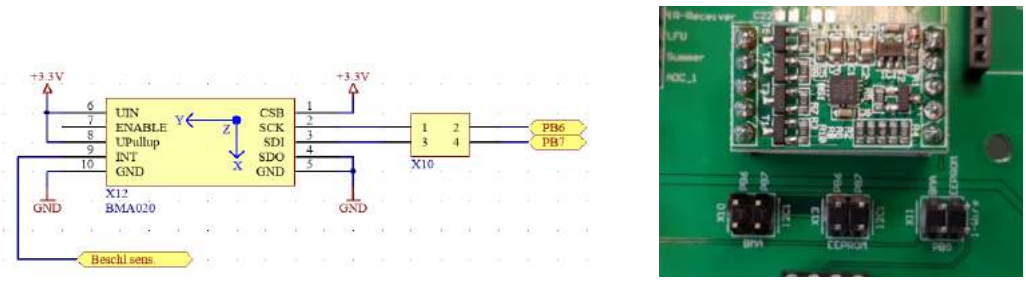


(a) Schematic (b) Hardware

Abbildung 64: EEPROM der Basisplatine

3.3.19 Beschleunigungssensor

Auf der Basisplatine wurde auch ein Beschleunigungssensormodul mit dem Beschleunigungssensor BMA020 vorgesehen. Die Kommunikation mit dem Beschleunigungssensor erfolgt mit Hilfe des I^2C-Buses (Adresse `0x70`). Durch diesen Sensor kann die Neigung, sowie die Kraft welche auf die Platine wirkt in der X, Y und Z-Richtung erfasst werden. Der Beschleunigungssensor besitzt einen Interrupt-Pin, welcher es ermöglicht ein Interrupt auszulösen. Der Interrupt kann über die Portleitung PB5 empfangen werden. Damit das Interrupt-Signal an der Portleitung PB5 ankommt, muss jedoch der Pin1 mit dem Pin2 der zweireihigen Stiftleiste X11 (Abbildung 61) gejumpert werden.



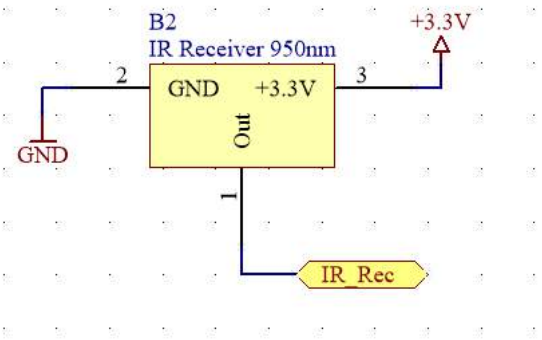
(a) Schematic (b) Hardware

Abbildung 65: Beschleunigungssensor der Basisplatine

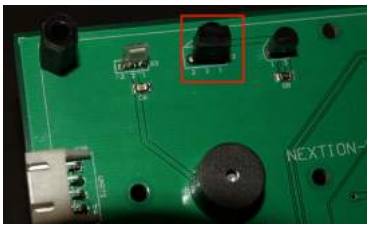
3.3.20 IR-Receiver

Um die Platine auch mit einer Fernbedienung oder einem Mobiltelefon steuern zu können würde ein IR-Receiver verbaut. Dieser Sensor wurde hardwaremäßig unter dem NEXTION-Display angebracht. Der IR-Receiver arbeitet mit einer Wellenlänge von 850nm bis 1000nm und einer Trägerfrequenz von 38kHz. Das vom IR-Receiver empfangene Signal wird direkt

im Receiver demoduliert und anschließend zur Stiftleiste X9 (Abbildung 60) weitergeleitet. Um den IR-Receiver verwenden zu können muss lediglich der Pin7 mit dem Pin8, der Stiftleiste X9 (Abbildung 60), gejumpert werden.



(a) Schematic

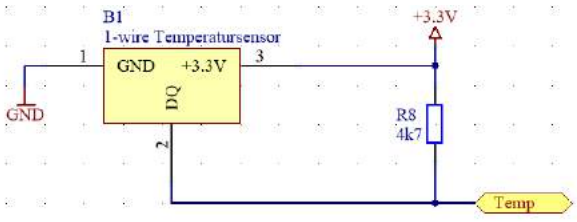


(b) Hardware

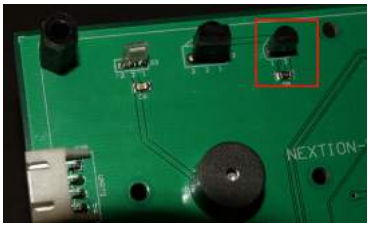
Abbildung 66: IR-Receiver der Basisplatine

3.3.21 Temperatursensor

Um die Umgebungstemperatur feststellen zu können wurde auf der Basisplatine der Temperaturfühler DS18B20 verbaut, welcher mit Hilfe des 1-Wire Protokolls angesprochen werden kann. Die maximale Mesfehler dieses Temperatursensors beträgt ±0,5°C laut Datenblattangabe. Um den Temperaturfühler verwenden zu können muss lediglich der Pin9 mit dem Pin10, der Stiftleiste X9 (Abbildung 60), gejumpert werden.



(a) Schematic



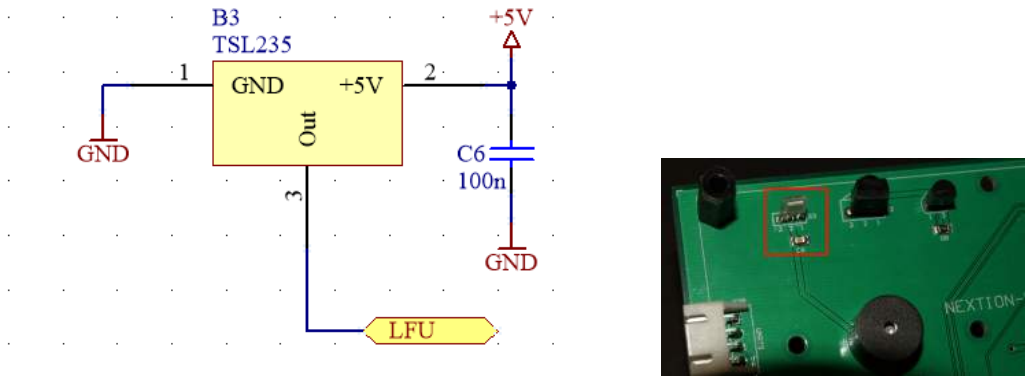
(b) Hardware

Abbildung 67: Temperatursensor der Basisplatine

3.3.22 Lichtwandler LFU

Der auf der Basisplatine realisierte LFU (Licht-Frequenz-Wandler), wandeltet wie der Name bereits sagt Licht in eine bestimmte Frequenz um. Je höher die Bestrahlungsstärke

des Lichts, desto höher wird die über den Output des LFUs ausgegebene Frequenz. Den linearen Zusammenhang zwischen der Bestrahlungsstärke und der ausgegebenen Frequenz kann aus Abbildung 69 entnommen werden. Um den LFU verwenden zu können muss lediglich der Pin5 mit dem Pin6, der Stiftleiste X9 (Abbildung 60), gejumpert werden.



(a) Schematic (b) Hardware

Abbildung 68: LFU der Basisplatine

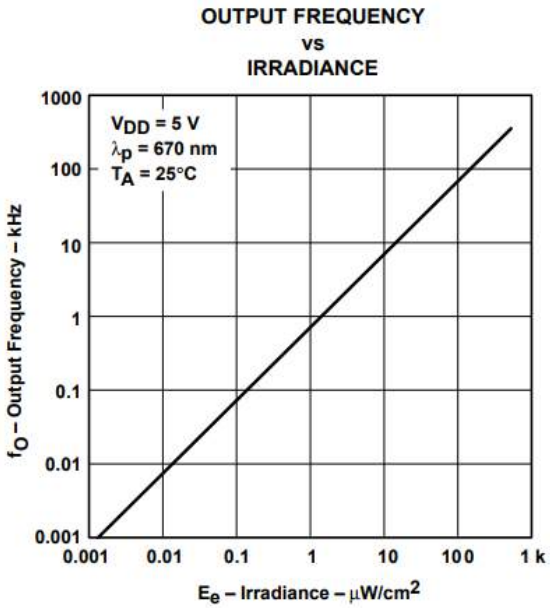
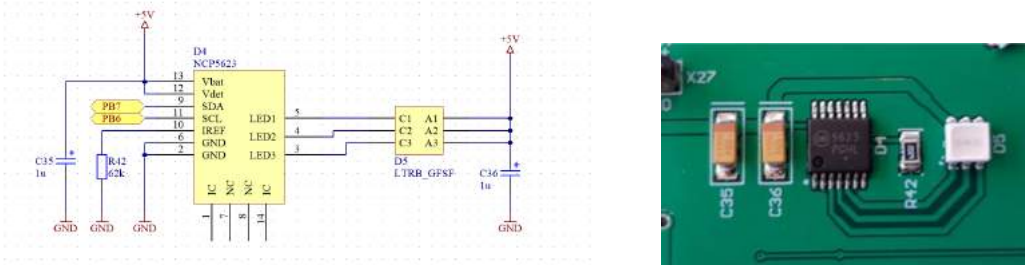


Abbildung 69: Frequenzgang des LFUs

3.3.23 RGB-LED

Auf der Basisplatine wurde ebenso eine RGB-LED [4] verbaut, welche mit Hilfe des LED-Drivers D4 (Abbildung 70) [5], welcher mit dem I^2C-Bus (Adresse `0x70`) angesteuert

werden kann. Dieser LED-Driver hat den Vorteil, dass er eine interne Stromüberwachung besitzt. Dadurch benötigen die einzelnen Anoden der RGB-LED keine Vorwiderstände, da sich der Strom automatisch entsprechend der gewünschten Farbe reguliert.



(a) Schematic (b) Hardware

Abbildung 70: RGB-LED der Basisplatine

B7	B6	B5	B4	B3	B2	B1	B0

Abbildung 71: Befehlsaufbau der RGB-LED [5]

B7	B6	B5	B4	B3	B2	B1	B0	Comments
0	0	0	X	X	X	X	X	Shut down
0	0	1	16	8	4	2	1	LED Current Step, see Figure 5 (Note 9)
0	1	0	BPWM16	BPWM8	BPWM4	BPWM2	BPWM1	Red PWM
0	1	1	BPWM16	BPWM8	BPWM4	BPWM2	BPWM1	Green PWM
1	0	0	BPWM16	BPWM8	BPWM4	BPWM2	BPWM1	Blue PWM
1	0	1	GDIM5 16	GDIM4 8	GDIM3 4	GDIM2 2	GDIM1 1	Set Gradual Dimming Upward Iend Target (Note 10)
1	1	0	GDIM5 16	GDIM4 8	GDIM3 4	GDIM2 2	GDIM1 1	Set Gradual Dimming Downward Iend Target (Note 10)
1	1	1	GDIM5 128 ms	GDIM4 64 ms	GDIM3 32 ms	GDIM2 16 ms	GDIM1 8 ms	Gradual Dimming Time & run

Abbildung 72: Register der RGB-LED [5]

I2C Address	COMMAND Bits[7:0]	Operation	Note
$70	000X XXXX	System Shut Down	Bits[4:0] are irrelevant
$70	0010 0000 0011 1111	Set Up the ILED current	ILED register Bits[4:0] contain the ILED value as defined by the I_{REF} value
$70	0100 0000 0101 1111	Set Up the RED PWM	REDPWM Bits[4:0] contain the PWM value
$70	0110 0000 0111 1111	Set Up the GREEN PWM	GREENPWM Bits[4:0] contain the PWM value
$70	1000 0000 1001 1111	Set Up the BLUE PWM	BLUEPWM Bits[4:0] contain the PWM value
$70	1010 0000 1011 1111	Set Up the IEND Upward	UPWARD Bits[4:0] contain the IEND value
$70	1100 0000 1101 1111	Set Up the IEND Downward	DWNWRD Bits[4:0] contain the IEND value
$70	1110 0001 1111 1111	Set Up the Gradual Dimming time and run the sequence	GRAD Bits[4:0] contain the TIME value

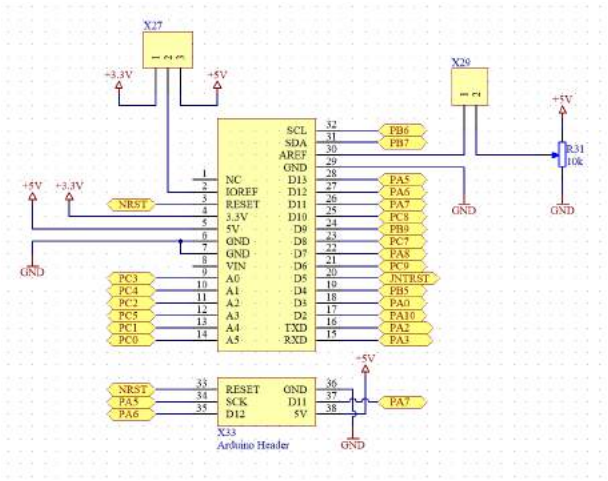
Abbildung 73: Sequenzen der RGB-LED [5]

3.3.24 Arduino-Shield-Header

Um Hardware des Arduino-Mikrocontrollersystems nutzten zu können, ohne selbst großen Aufwand in die Entwicklung entsprechender Module investieren zu müssen, wurde ein Arduino-Shield-Header auf der Basisplatine vorgesehen. Dieser ermöglicht es durch die Portkompatibilität mit einem Arduino, dessen Shields zu verwenden oder selbst Shields entwickeln zu können. Möchte man nun ein Arduino-Shield verwenden muss dieses lediglich in die vorgesehene Buchsenleiste X33 (Abbildung 74) gesteckt werden. Da jedes Arduino-Shield die Möglichkeit besitzt eine Referenzspannung für diverse ADCs zu vergeben wurde das Potentiometer R31 vorgesehen, um diesen Spannungspegel variabel zu gestallten. Darüber hinaus gibt es noch die Möglichkeit bei speziellen Shields zu definieren mit welcher Betriebsspannung die darüberliegenden versorgt werden sollen. Dazu wurde die Stiftleiste X27 verbaut, um festzulegen ob die darüberliegenden Shields mit +5 V oder +3, 3 V versorgt werden. Sollte kein Jumper gesetzt werden wird automatisch auf die +5 V Spannungsversorgung zurückgegriffen.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
PA	D3		D1/TXD	D0/RXD		D13	D12	D11	D7		D2						PA
	DIL-Adaper																
PB					D5	D4	D15	D14		D9							PB
	DIL-Adaper			DIL-Adapter													
PC	A5	A4	A2	A0	A1	A3		D8	D10	D6							PC
	DIL-Adaper																
	Arduino																
	Ausgeführte Port-Pins																

Tabelle 9: Pinning Arduino Header



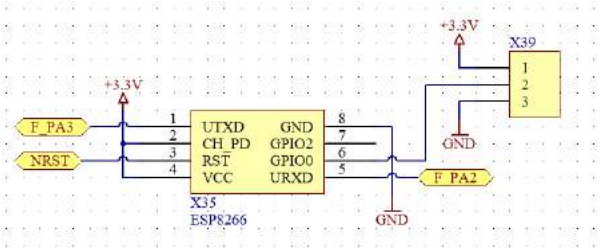
(a) Schematic

(b) Hardware

Abbildung 74: Arduino-Shield-Header der Basisplatine

3.3.25 WLAN-Modul [6]

Um Daten ohne großen Aufwand direkt in das Heimnetzwerk einspeisen zu können, wurde auf der auf der Basisplatine ein Steckplatz X35 (Abbildung 75) für ein ESP8266 W-LAN Modul vorgesehen. Dieses Funkmodul benutzt zu Kommunikation mit dem Prozessor des Core-Moduls die UART2 Schnittstelle und sendet die Daten mit einer Sendefrequenz von $2,4$ GHz im ISM-Band. Wenn man die Firmware des ESP8266 verändern möchte muss man lediglich die Portleitung GPIO0 des ESP8266, durch jumpern von Pin2 und Pin3 der Stiftleiste X39 (Abbildung 75), gegen Masse schalten.



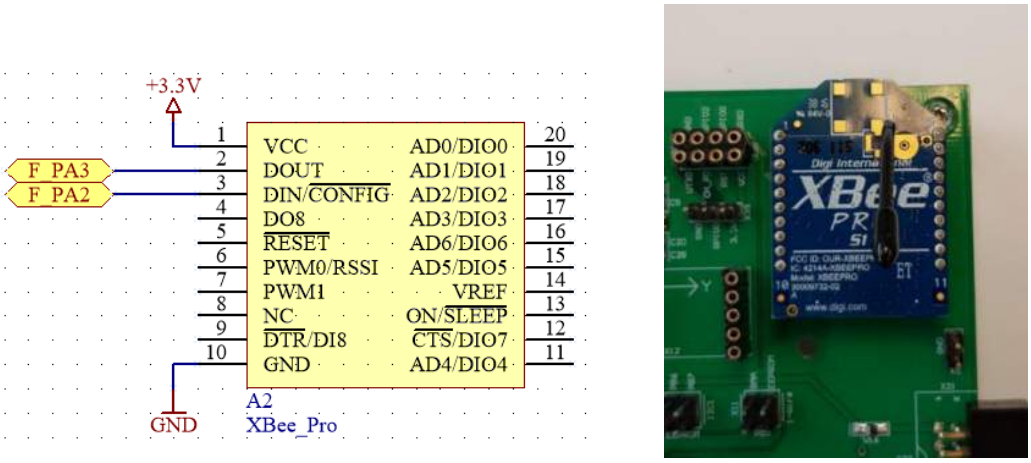
(a) Schematic

(b) Hardware

Abbildung 75: WLAN-Modul der Basisplatine

3.3.26 XBee-Pro-Modul

Um Daten ohne großen Aufwand per Funk übertragen können, wurde auf der auf der Basisplatine ein Steckplatz A2 (Abbildung 76) für ein XBee-Pro Modul vorgesehen. Dieses Funkmodul benutzt zu Kommunikation mit dem Prozessor des Core-Moduls die UART2 Schnittstelle und sendet die Daten mit einer Sendefrequenz von $2,4$ GHz im ISM-Band. Darüber hinaus weist das Funkmodul eine maximale Datenübertragungsrate von 250 kb/s auf und hat eine maximale Reichweite von 1600 m.

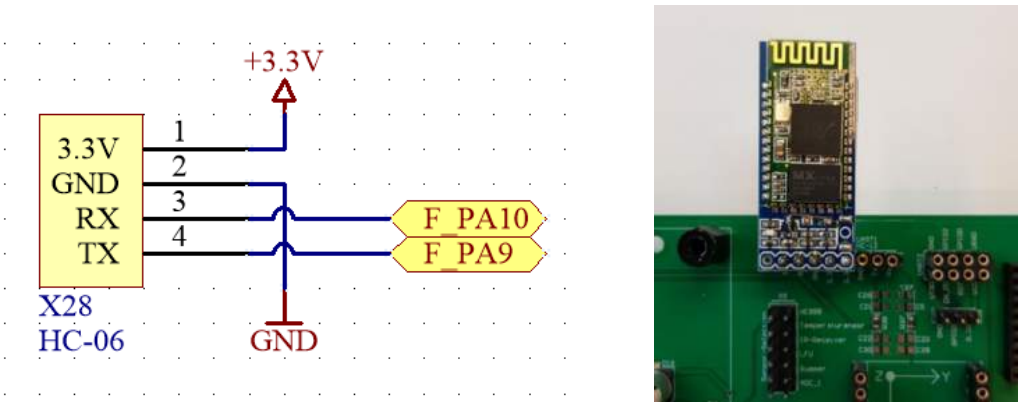


(a) Schematic (b) Hardware

Abbildung 76: XBee-Pro-Modul der Basisplatine

3.3.27 HC-06-Modul [7]

Um Daten ohne großen Aufwand per Bluetooth übertragen können, wurde auf der auf der Basisplatine ein Steckplatz X28 (Abbildung 77) für ein HC-06 Bluetooth-Modul vorgesehen. Dieses Bluetooth-Modul benutzt zur Kommunikation mit dem Prozessor des Core-Moduls die UART1 Schnittstelle und sendet die Daten mit einer Sendefrequenz von $2,4$ GHz im ISM-Band. Darüber hinaus weist das Bluetooth-Modul eine maximale Datenübertragungsrate von 1382400 baud auf.

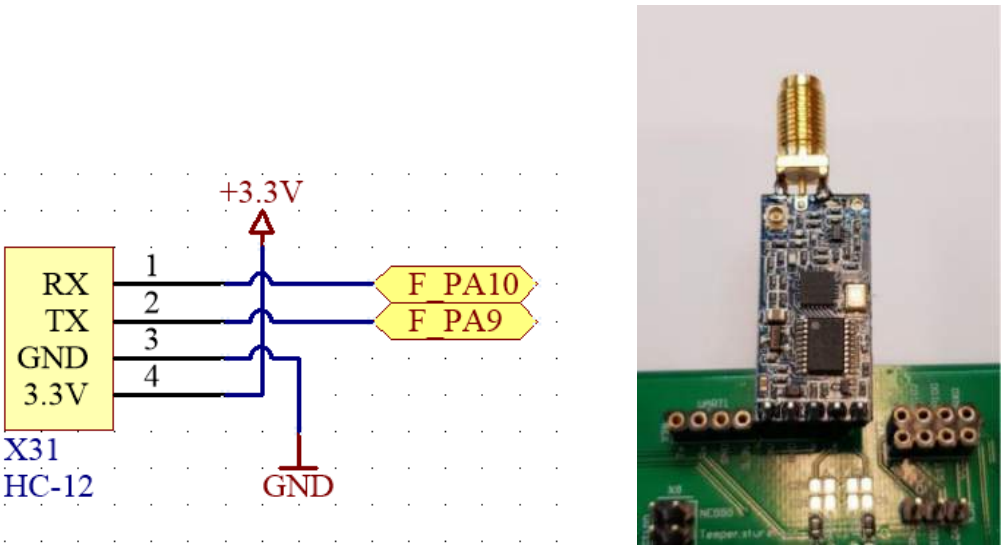


(a) Schematic (b) Hardware

Abbildung 77: HC-06-Modul der Basisplatine

3.3.28 HC-12-Modul [8]

Um Daten ohne großen Aufwand per Funk übertragen können, wurde auf der auf der Basisplatine ein Steckplatz X31 (Abbildung 78) für ein HC-12 Funkmodul vorgesehen. Dieses Funkmodul benutzt zu Kommunikation mit dem Prozessor des Core-Moduls die UART1 Schnittstelle und sendet die Daten mit einer Sendefrequenz von 433, 4 MHz bis 473, 0 MHz im ISM-Band. Darüber hinaus weist das Funkmodul eine maximale Datenübertragungsrate von 115200 baud auf und hat eine maximale Reichweite von 1800 m. Das HC-12 Funkmodul kann mittels AT-Befehlen konfiguriert werden. Dabei kann die maximale Ausgangsleistung, die Sendefrequenz und die maximale Datenübertragungsrate verändert werden.



(a) Schematic (b) Hardware

Abbildung 78: HC-12-Modul der Basisplatine

Um das Modul in den Programming-Mode zu setzen, müssen folgende Schritte ausgeführt werden:

- Um das Modul programmieren zu können muss der Pin5 („SET“) gegen Masse gezogen werden, während das Modul mit Spannung versorgt ist (ca. 40 ms).
- Anschließend muss die Spannungsversorgung unterbrochen werden und der Pin5 („SET“) wieder mit Masse verbunden werden. Erst wenn der SET-Pin mit Masse verbunden ist darf das Modul wieder mit Spannung versorgt werden.

Befehl	**Funktion**
AT	„AT“ ist ein Test-Befehl um festzustellen ob eine Programmierung des Modules möglich ist. Sollte diese möglich sein antwortet das Modul mit „OK“.
AT+B`xxxx`	Mit diesem Befehl kann die Baudrate des Funkmoduls verändert werden. Anstelle der „`x`“ gehört die gewünschte Baudrate eingetragen. Sollte der Befehl korrekt an das Funkmodul weitergegeben worden sein antwortet dieses mit „OK+B`xxxx`“.
AT+C`xxxx`	Mit diesem Befehl kann der Kommunikationskanal des Funkmoduls verändert werden. Anstelle der „`x`“ gehört der gewünschte Kommunikationskanal eingetragen. Sollte der Befehl korrekt an das Funkmodul weitergegeben worden sein antwortet dieses mit „OK+C`xxxx`“.

AT+FU`x`	Mit diesem Befehl kann der Strombedarf des Funkmoduls verändert werden. Anstelle des „`x`“ gehört der gewünschte Strombedarf eingetragen. Sollte der Befehl korrekt an das Funkmodul weitergegeben worden sein antwortet dieses mit „OK+FU`x`“.
AT-P`x`	Mit diesem Befehl kann die maximale Sendeleistung des Funkmoduls verändert werden. Anstelle des „`x`“ gehört die gewünschte maximale Sendeleistung eingetragen. Sollte der Befehl korrekt an das Funkmodul weitergegeben worden sein antwortet dieses mit „OK-P`x`“.
AT-R`y`	Mit diesem Befehl kann man die aktuellen Einstellungen des Funkmoduls abfragen. Anstelle des „`x`“ gehört einer der Buchstaben der AT-Befehle (B, C, F, P) eingetragen. Sollte der Befehl korrekt an das Funkmodul weitergegeben worden sein antwortet dieses mit „OK-R`y`“.
AT+Udps	Mit diesem Befehl können die Anzahl der Datenbits (d), der Paritybits (p) und der Stoppbits (s) verändert werden. Sollte der Befehl korrekt an das Funkmodul weitergegeben worden sein antwortet dieses mit „OK+Upds“.
AT-V	Mit diesem Befehl kann die aktuelle Firmwareversion des Funkmoduls abgefragt werden.

Tabelle 10: Aufbau von AT-Befehlen [8]

Baud-Rate	1200 bps	2400 bps	4800 bps	9600 bps	19.200 bps	38.400 bps	57.600 bps	115.200 bps

Stromverbrauch					
Modus	FU1	FU2	FU3	FU4	Anmerkungen
Idle Strom	3,6mA	80µA	16mA	16mA	Mittelwert
Sendezeit-verzögerung	15-25ms	500ms	4-80ms	1000ms	Senden von einem Byte

Stromverbrauch								
x-Wert	1	2	3	4	5	6	7	8
Sende-leistung	-1dBm (0,8mW)	2dBm (1,6mW)	5dBm (3,2mW)	8dBm (6,3mW)	11dBm (12mW)	14dBm (25mW)	17dBm (50mW)	20dBm (100mW)

Tabelle 11: Übersicht über die HC-12 AT-Befehlsparameter [8]

Um das Funkmodul auf Werkseinstellungen zurückzusetzen muss der AT-Befehl „AT+DEFAULT“ eingegeben werden. Sollte der Befehl korrekt an das Funkmodul weitergegeben worden sein

antwortet dieses mit „OK+DEFAULT“. Um die Firmware des Funkmoduls zu updaten muss der AT-Befehl „AT+UPDATE“ eingegeben werden.

3.3.29 PI-Filter

Da die Basisplatine auf die Verwendung von Funkmodulen ausgelegt ist, welche im HF-Bereich senden und empfangen, wurden als Vorsichtsmaßname PI-Filter für die RX- und TX-Leitungen der UARTs, welche mit den Funkmodulen verbunden sind, vorgesehen um HF-Störungen zu unterdrücken. Je nach Bedarfsfall können nun diese PI-Filter bestückt werden. Im laufendem Betrieb hat sich jedoch gezeigt, dass diese nicht unbedingt erforderlich sind.

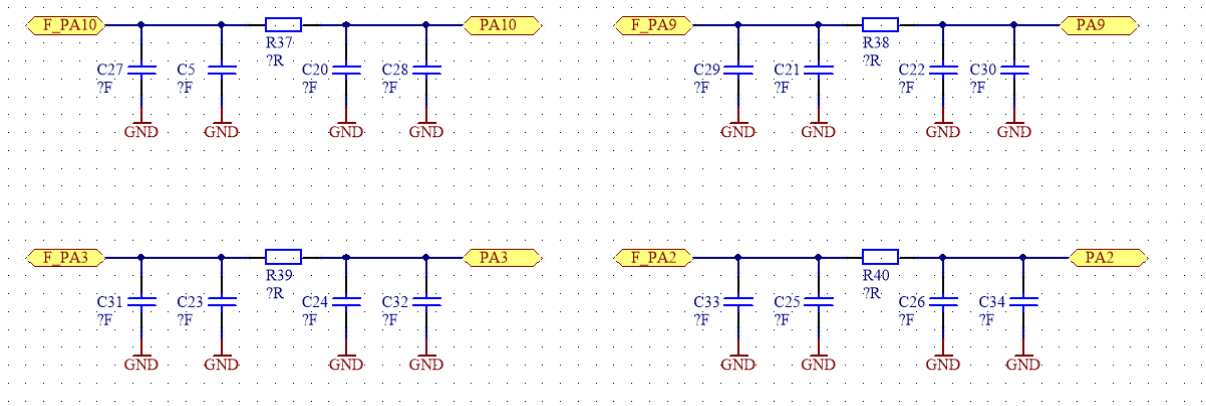
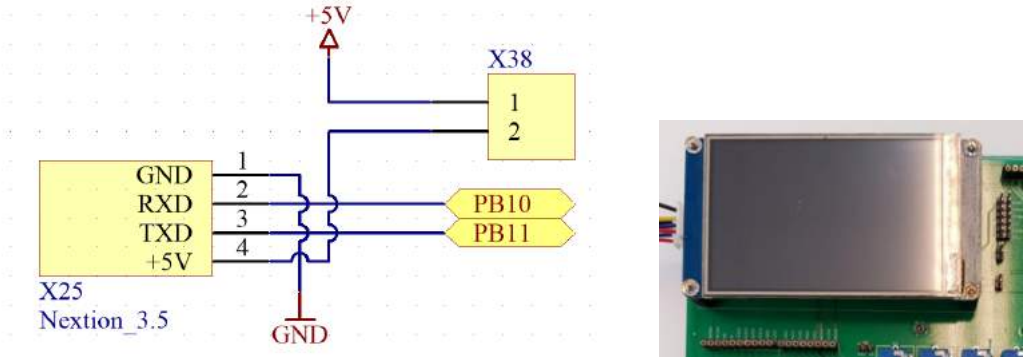


Abbildung 79: PI-Filter der Basisplatine

3.3.30 NEXTION-Display

Um den Benutzer eine grafische Darstellungsmöglichkeit von Messwerten, Bildern oder ähnlichen zu geben wurde auf der Basisplatine der Header X25 (Abbildung 80) vorgesehen. Mit Hilfe dieses Headers ist es möglich ein NEXTION-Display auf der Basisplatine zu befestigen und über die UART3-Schnittstelle anzusteuern. Um das NEXTION-Display verwenden zu können muss lediglich die beiden Pins der Stiftleiste X38 (Abbildung 80) mit einem Jumper verbunden werden. Zur Programmierung des GUI, des NEXTION-Displays wird der selbst entwickelte USB-to-UART-Adapter benutzt, welcher in Abschnitt 4 näher erklärt wird.

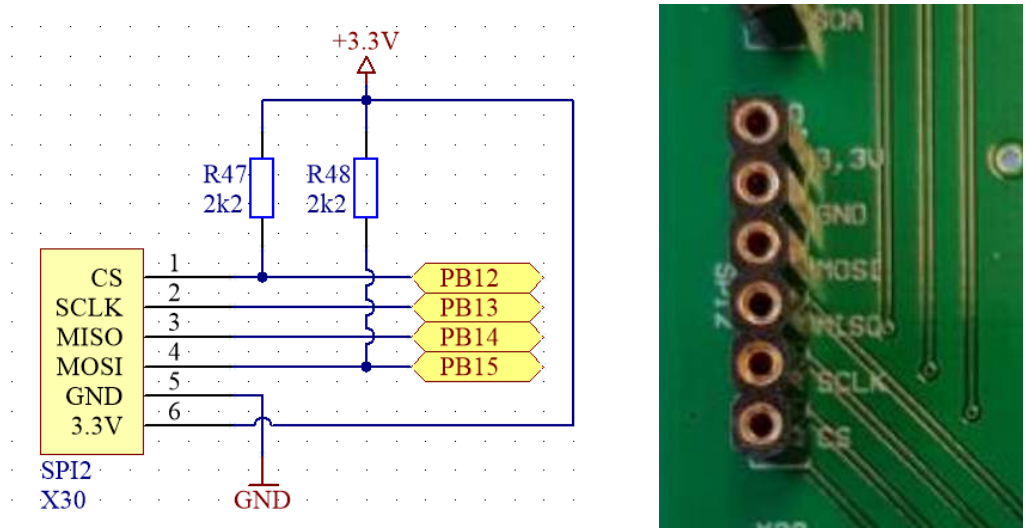


(a) Schematic (b) Hardware

Abbildung 80: NEXTION-Display der Basisplatine

3.3.31 SPI-Schnittstelle

Um zusätzliche Hardware mit der Basisplatine ansteuern zu können wurde die SPI-Schnittstelle X30 (Abbildung 81) vorgesehen, welche mit der SPI2-Schnitstelle des Prozessors verbunden ist. Da die Chip-Select Leitung (CS) von keiner Hardware auf der Basisplatine benötigt wird, kann diese verwendet werden. Sollten jedoch mehrere Geräte an den SPI-Bus angeschlossen werden, müssen weitere Portleitungen als Chip-Select Leitungen verwendet werden.



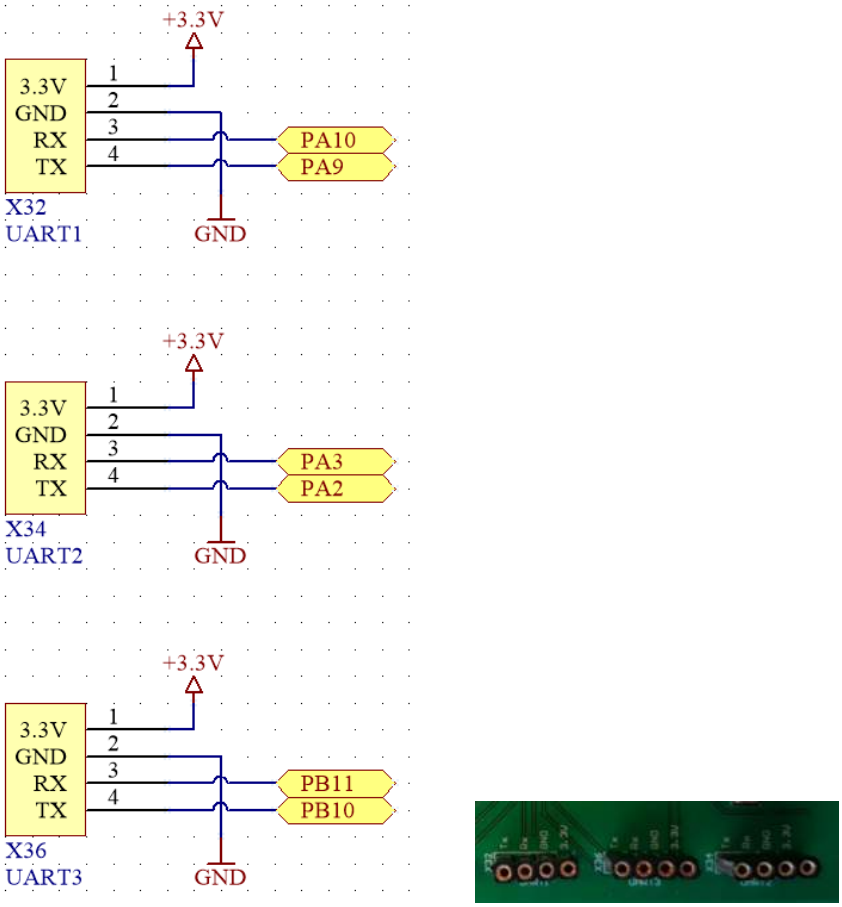
(a) Schematic (b) Hardware

Abbildung 81: SPI-Schnittstelle der Basisplatine

3.3.32 UART-Schnittstelle

Um zusätzliche Hardware mit der Basisplatine ansteuern zu können wurden die Buchsenleisten X32, X34 und X36 (Abbildung 82) realisiert, welche den UART1, UART2, und UART3 des Prozessors verbunden sind. Um eine UART-Schnittstelle verwenden zu können ist es wichtig die RX- und TX-Leitung im Vergleich zum verwendeten Modul zu vertauschen (Null-Modem-Kabel). Alle verwendeten UART-Schnittstellen wurden aus der Sicht des Prozessors bezeichnet.

☞ Anmerkung: Bei der Verwendung der UART2-Schittstelle ist Vorsicht geboten, da wie bereits in Abschnitt 3.3.7 beschrieben, ein Echo auf der Schnittstelle ausgelöst wird, wenn die Kippschaltschalter S3 und S4 gleichzeitig geschlossen sind.

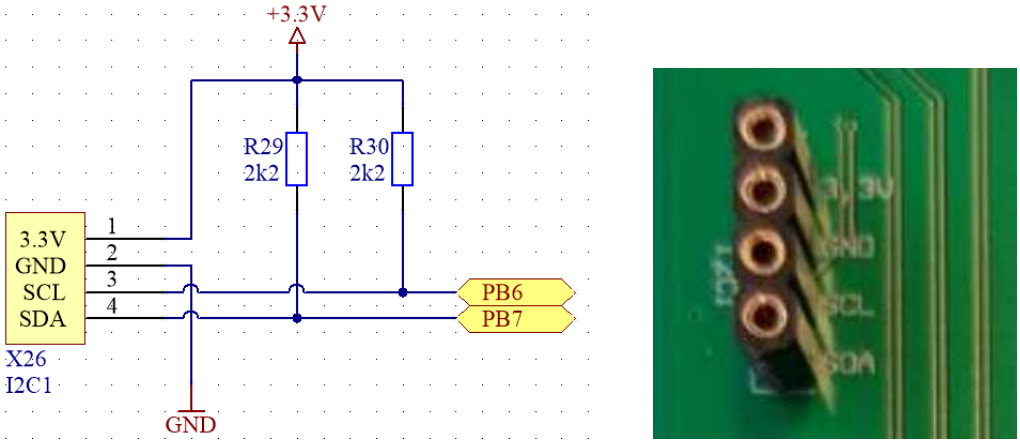


(a) Schematic (b) Hardware

Abbildung 82: UART-Schnittstelle der Basisplatine

3.3.33 I²C-Schnittstelle

Um über die Basisplatine weitere I²C Geräten anschließen zu können wurde die Buchsenleiste X26 (Abbildung 83) vorgesehen. Die Anschlüsse sind direkt mit der I²C1 Schnittstelle des Mikrocontrollers verbunden. Die I²C-Schnittstelle verfügt über zwei PullUp-Widerstände, da der verwendete Mikrocontroller PullUp-Widerstände nur im Input-Modus zur Verfügung stellt.

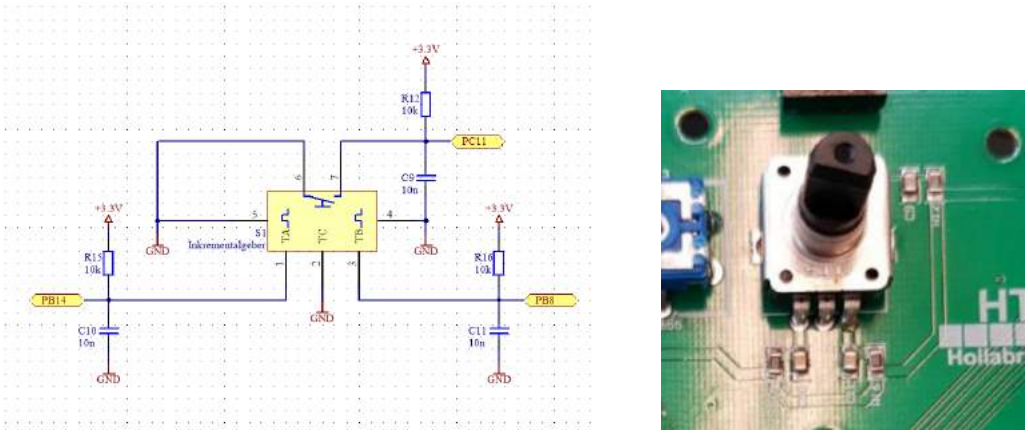


(a) Schematic (b) Hardware

Abbildung 83: I²C-Schnittstelle der Basisplatine

3.3.34 Inkrementalgeber

Um einfache Winkelmessungen, Positionierungsaufgaben, Geschwindigkeitsmessungen oder Weglängenmessungen realisieren zu können wurde auf der Basisplatine ein Inkrementalgeber vorgesehen. Dieser verfügt über einen Taster, welcher über die Portleitung PC11 abgefragt werden kann. Darüber hinaus kann mit Hilfe der Portleitungen PB8 und PB14 festgestellt werden in welche Richtung und um wie viele Schritte der Inkrementalgeber gedreht wurde. Eine komplette Umdrehung des Inkrementalgebers setzt sich aus 24 Einzelschritten zusammen.



(a) Schematic (b) Hardware

Abbildung 84: Inkrementalgeber der Basisplatine

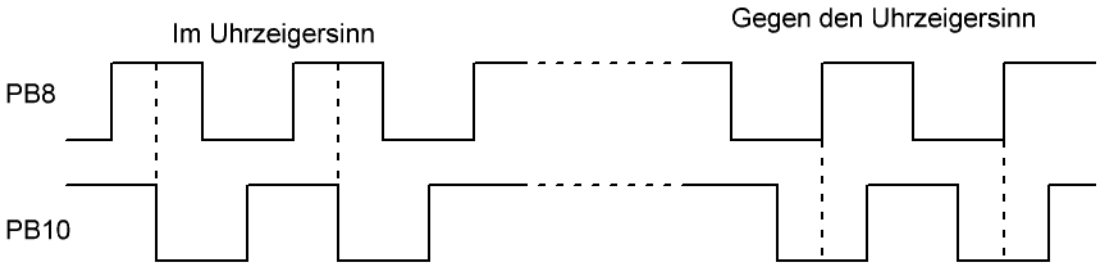
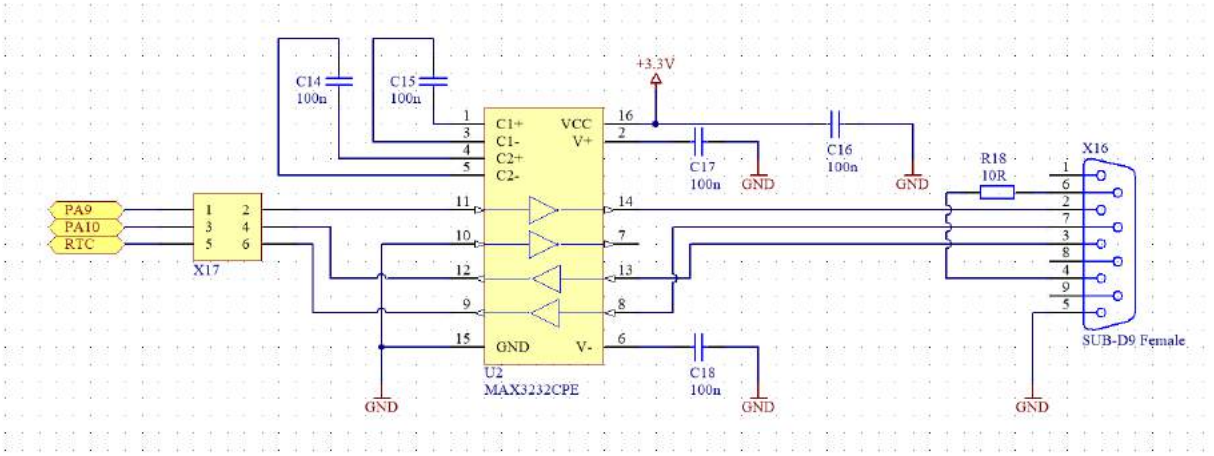


Abbildung 85: Inkrementalgeber Timing-Diagramm

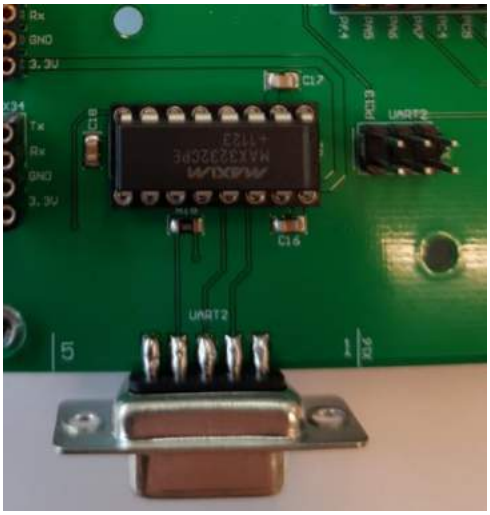
3.3.35 Serielle Schnittstelle

Um mit Messgeräten oder PCs zu kommunizieren unterstützt die Basisplatine eine Serielle-Schnittstelle. Für diese Schnittstelle wird der UART1 des Prozessors verwendet. Die Pegelumwandlung, von ± 12 V auf $3,3$ V, welche für die Kommunikation benötigt werden erfolgt mit Hilfe eines MAX232. Die Portleitung PC13 (Tamper-RTC) kann als Handshakeleitung verwendet werden. Zur Verwendung der Seriellen-Schnittstelle, ohne Handshake-Leitung, muss lediglich der Pin1 mit dem Pin2 und der Pin3 mit dem Pin4, der Stiftleiste X17 (Abbildung 86) miteinander mit Hilfe eines Jumpers verbunden werden. Sollte man die Handshake-Leitung (CTS) verwenden möchten, muss lediglich der Pin5 mit dem Pin6, der Stiftleiste X17 (Abbildung 86) miteinander verbunden werden. Die RTS und CTS Leitung ist direkt miteinander verbunden, es wird also immer von einer Empfangsbereitschaft der Prozessors ausgegangen. Alternativ ist es möglich Software-Handshake mit XON/XOFF

zu verwenden. Die Sub-D9 Buchse X16 (Abbildung 86) ist bereits so ausgeführt (Modem), dass man einen PC direkt (ohne Null-Modem-Kabel) anschließen kann.



(a) Schematic



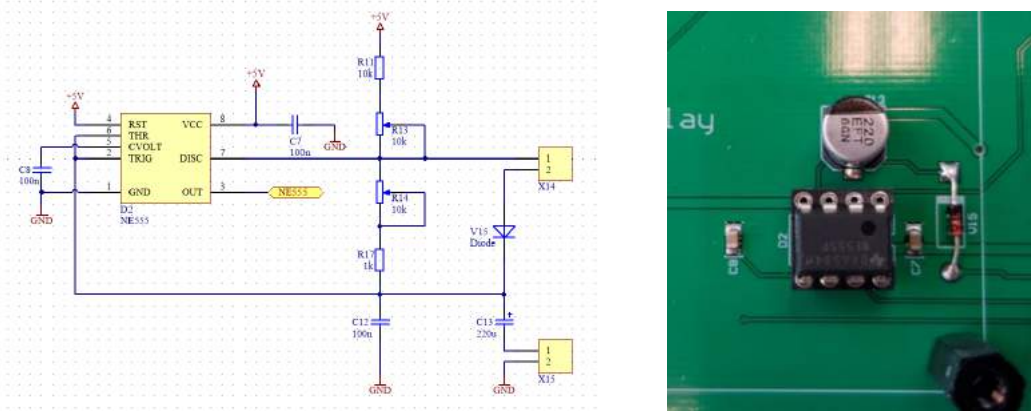
(b) Hardware

Abbildung 86: Serielle-Schnittstelle der Basisplatine

3.3.36 NE555

Der auf der Basisplatine verbaute NE555 kann als externer Taktgenerator verwendet werden. Die Periodendauer von diesen, kann mit Hilfe des Verhältnisses der beiden Potentiometer R13 und R14 (Abbildung 87) verändert werden. Für die Realisierung von großen Zeitkonstanten, kann man durch verbinden der beiden Pins auf der Stiftleiste X15 (Abbildung 87), mit einem Jumper, einen größeren Kondensator C13 (Abbildung 87) parallel zum kleineren Kondensator C12 (Abbildung 87) schalten. Möchte man einen kleineren

Widerstand erzielen, kann man das durch einen Jumper auf der Stiftleiste X14 (Abbildung 87) erzielen. Um den NE555 verwenden zu können muss lediglich der Pin11 mit dem Pin12, der Stiftleiste X9 (Abbildung 60), mit einem Jumper verbunden werden.



(a) Schematic (b) Hardware

Abbildung 87: NE555 der Basisplatine

Die Gesamtperiodendauer des NE555 setzt sich aus Summe der Teilperiodendauern t_1 und t_2 zusammen, kann jedoch auch direkt berechnet werden. Die Formeln zu Berechnung der Periodendauer lauten wie Gleichung (1) zeigt, Abbildung 88 stellt die Beziehung der Werte zueinander grafisch dar.

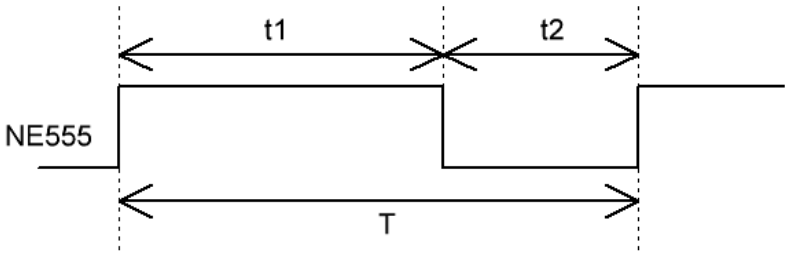


Abbildung 88: Timing des NE555

$$
\begin{aligned}
T &= t_1 + t_2 = 0,7 * [1k\Omega + R_1 3 + 2 * R] * C \\
t_1 &= 0,7 * (1k\Omega + R_1 3 + R) * C \\
t_1 &= 0,7 * R * C
\end{aligned}
\tag{1}
$$

Der Kondensatorwert C entspricht bei nicht gejumperter Stiftleiste X15 (Abbildung 87), den Kondensatorwert von C12 (100 nF). Sollte die Stiftleiste jedoch gejumpert sein, entspricht der Kondensatorwert von C dem Kondensatorwert der Parallelschaltung des Kondensators C12 und C13 ($220,1\ \mu F$). Möchte man einen Tastgrad von 0,5 erzielen, muss lediglich die Stiftleiste X14 (Abbildung 87) gejumpert werden. Wenn diese nicht gejumpert wird ergibt sich der Widerstand R aus der Summe von R14 und R17.

3.4 Gesamtschaltung

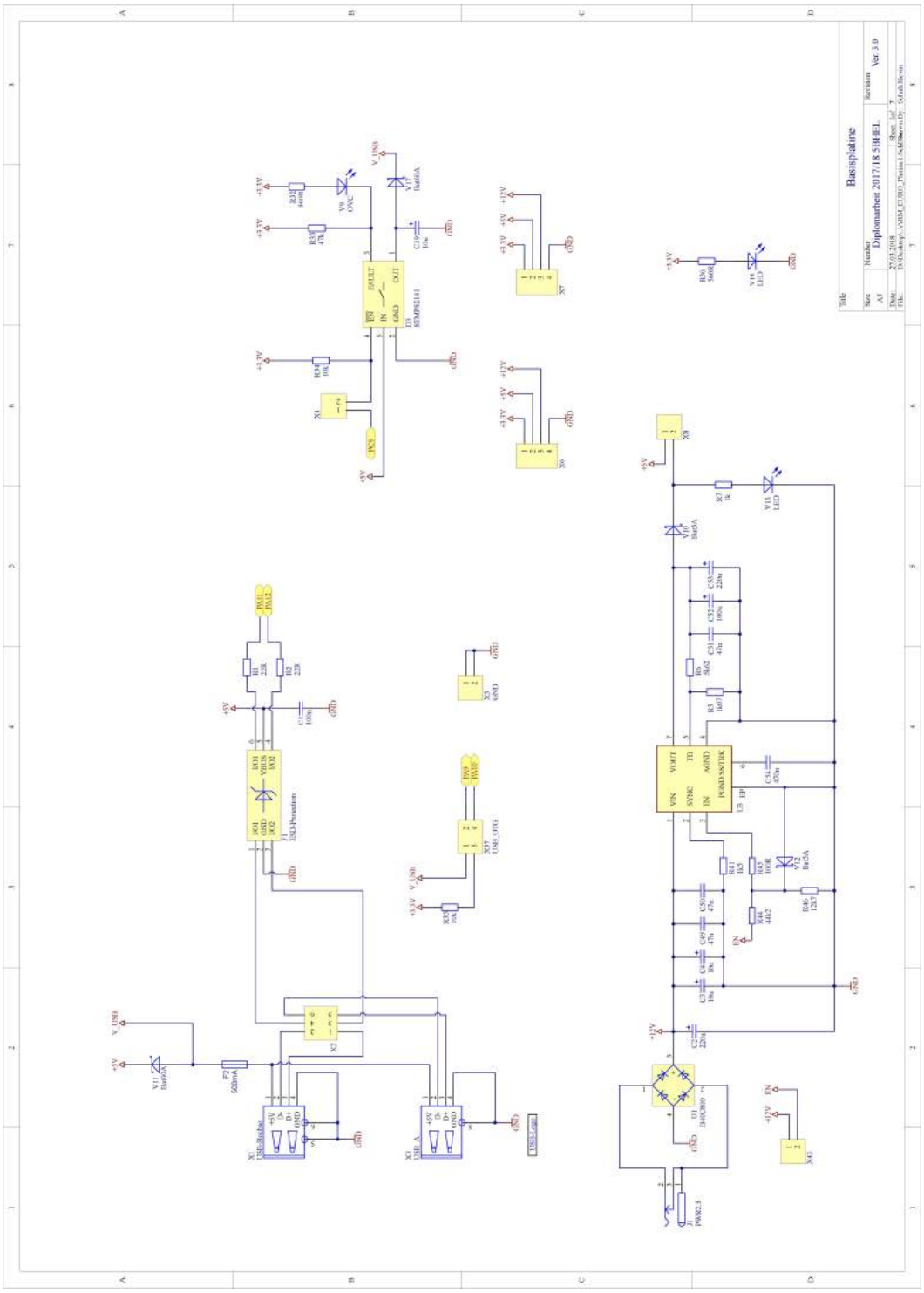


Abbildung 89: Gesamtschaltung der Basisplatine

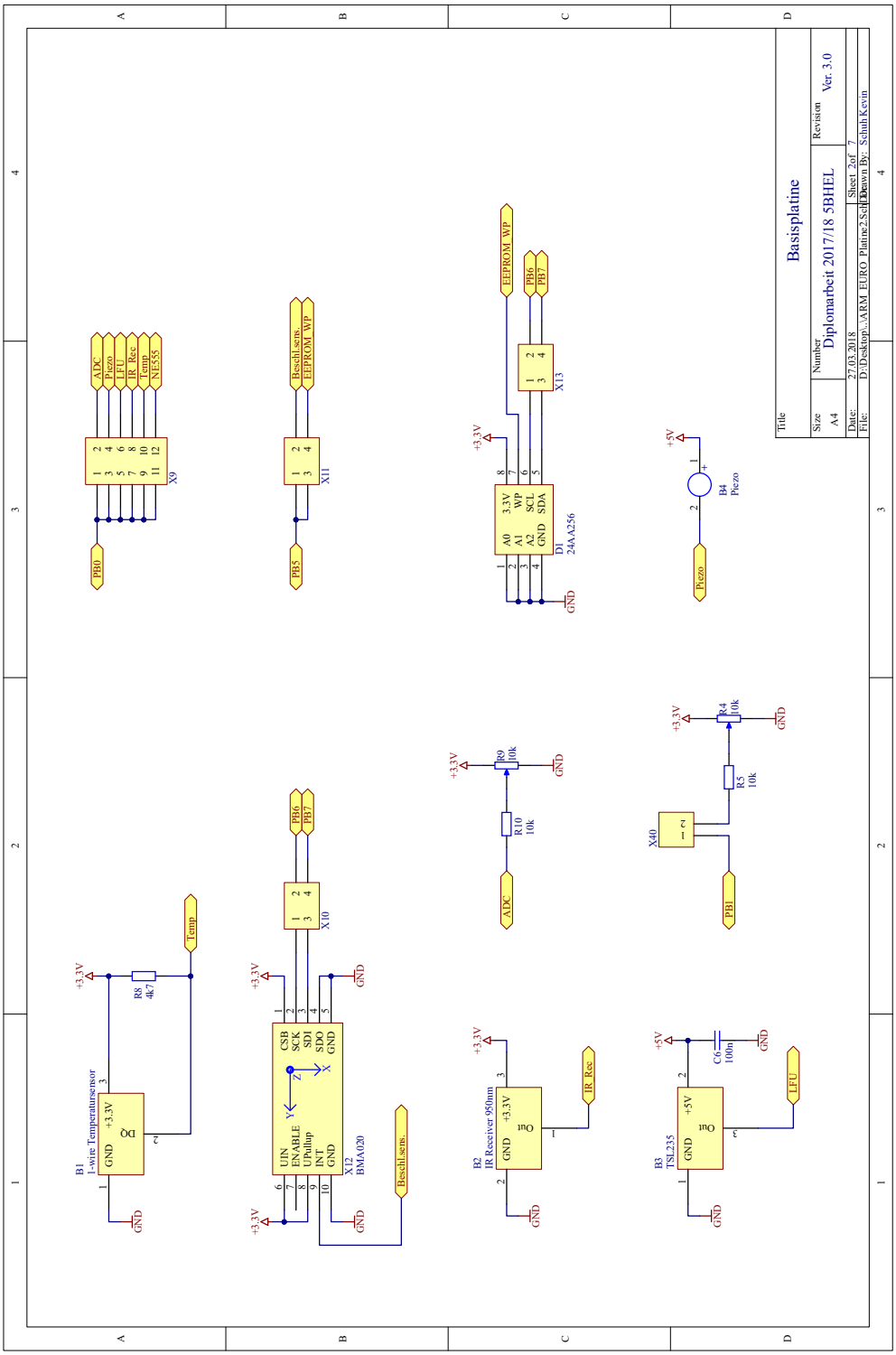


Abbildung 89: Gesamtschaltung der Basisplatine

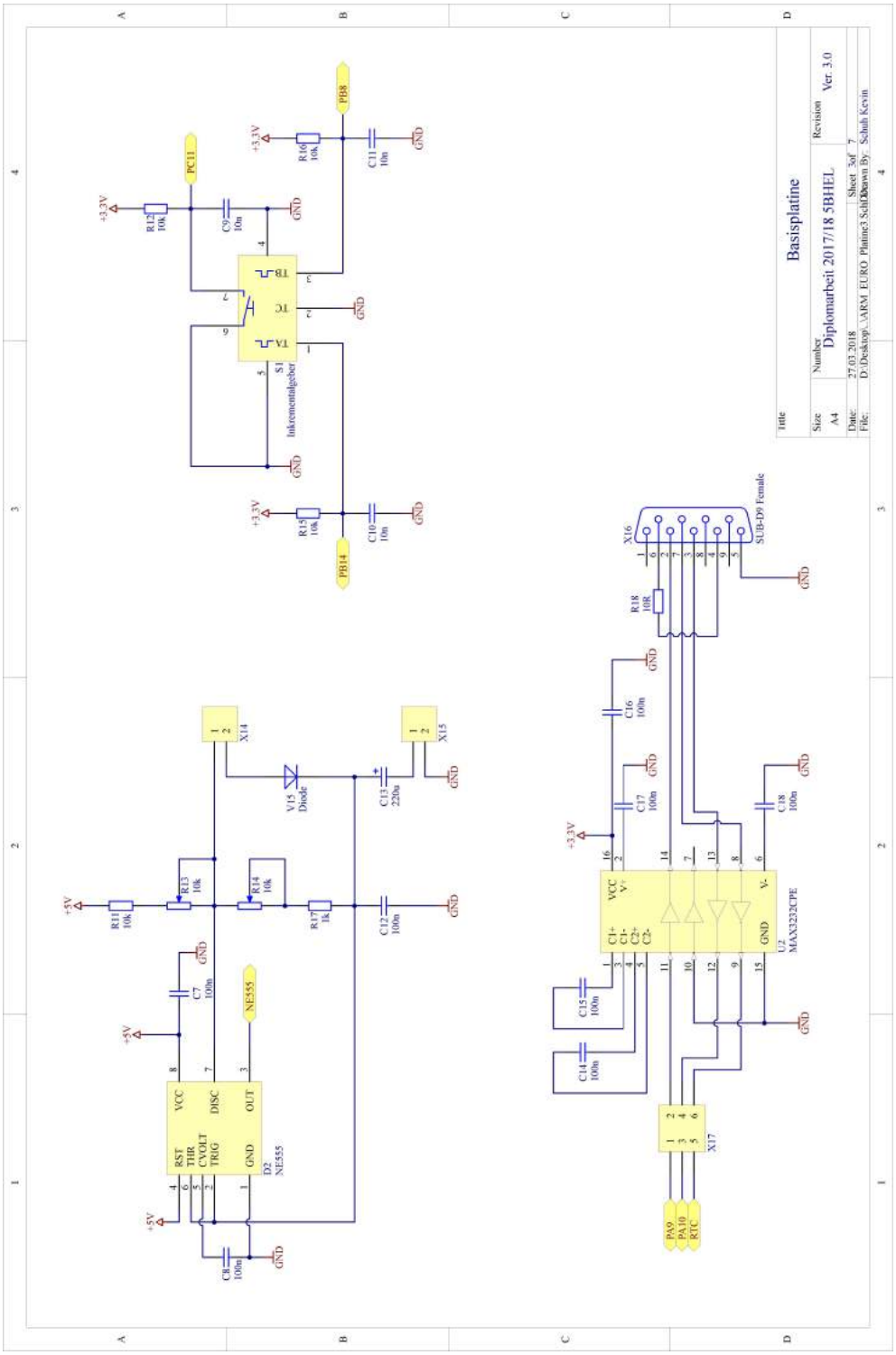


Abbildung 89: Gesamtschaltung der Basisplatine

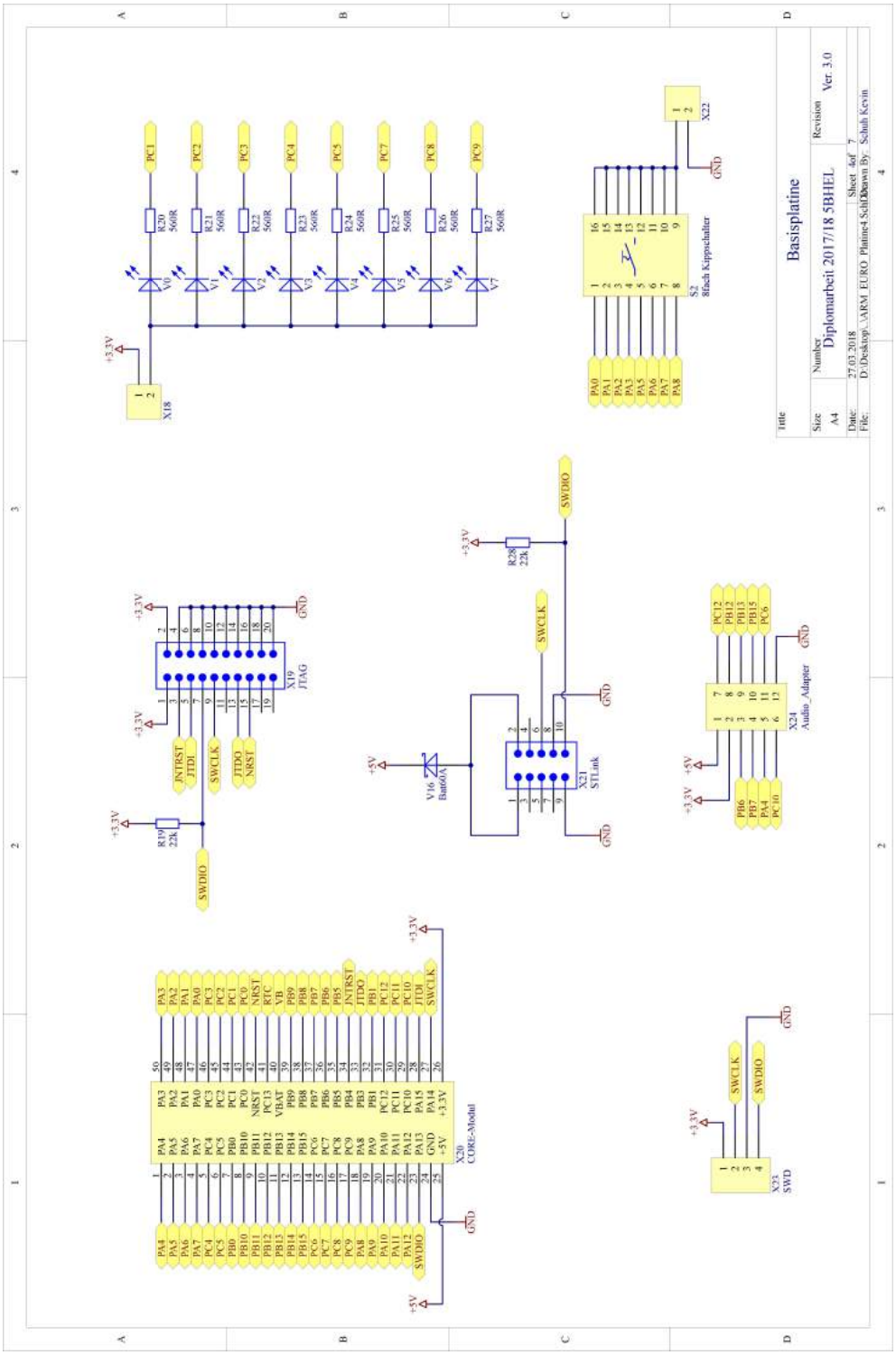


Abbildung 89: Gesamtschaltung der Basisplatine

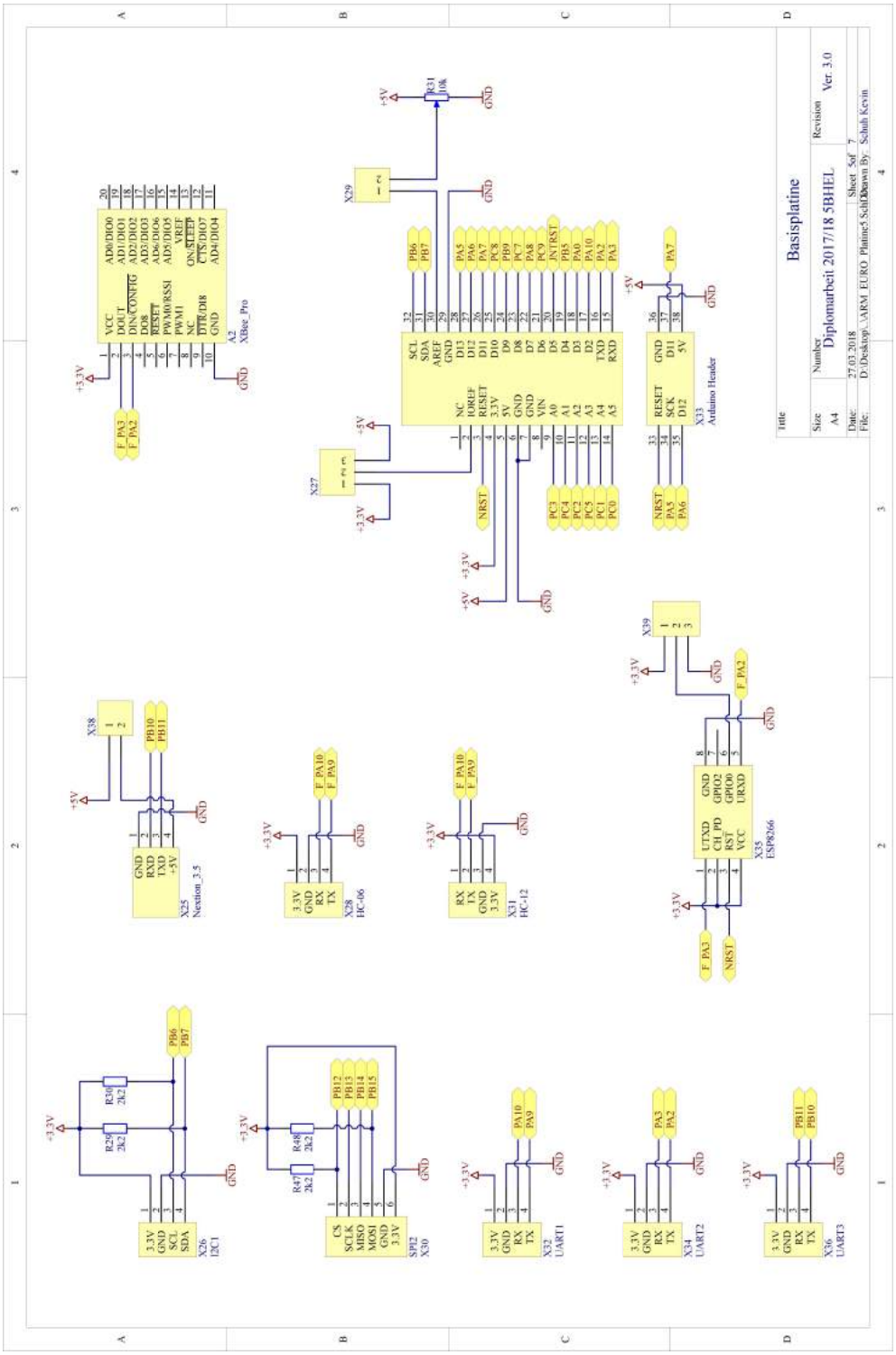


Abbildung 89: Gesamtschaltung der Basisplatine

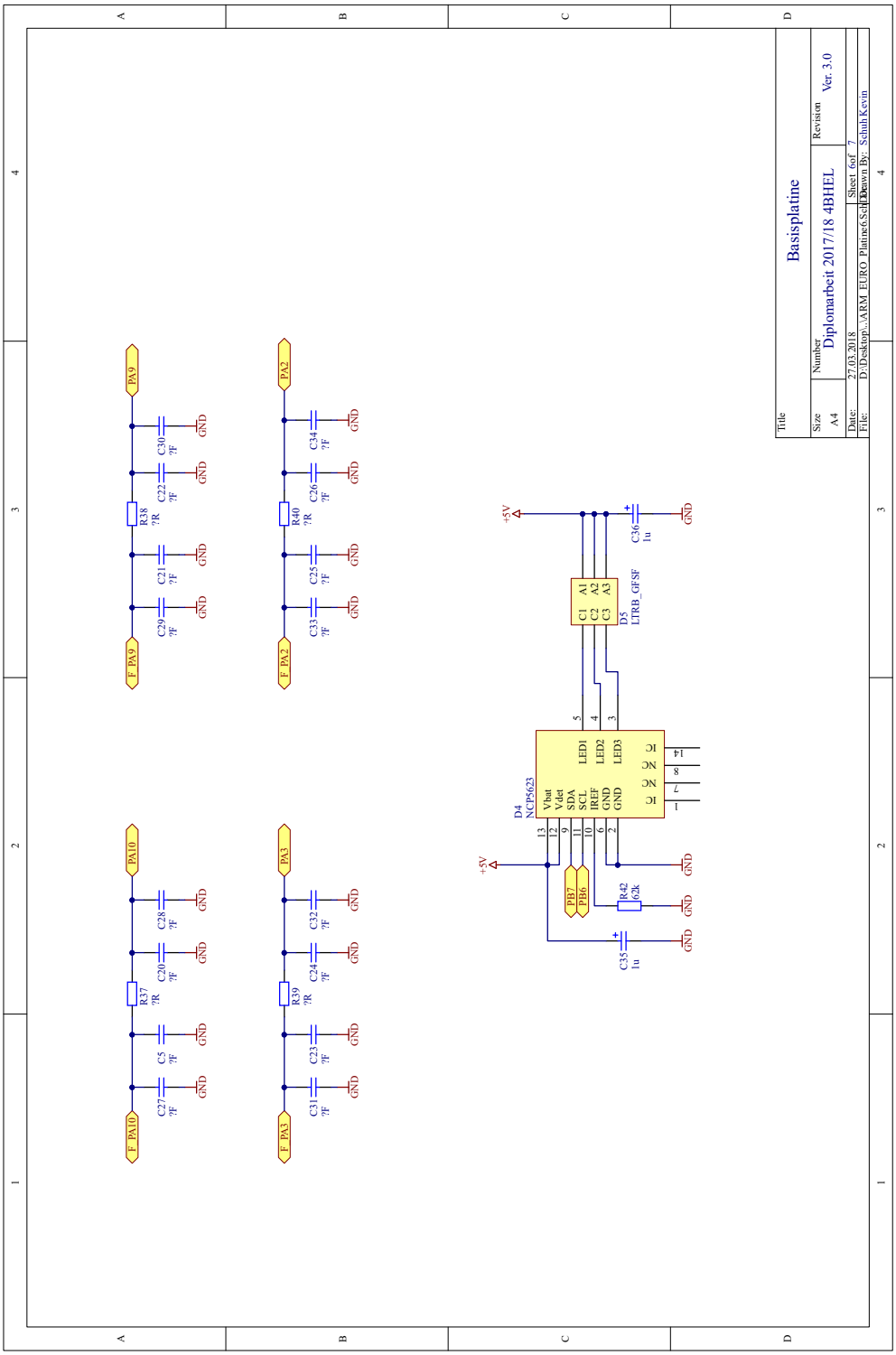


Abbildung 89: Gesamtschaltung der Basisplatine

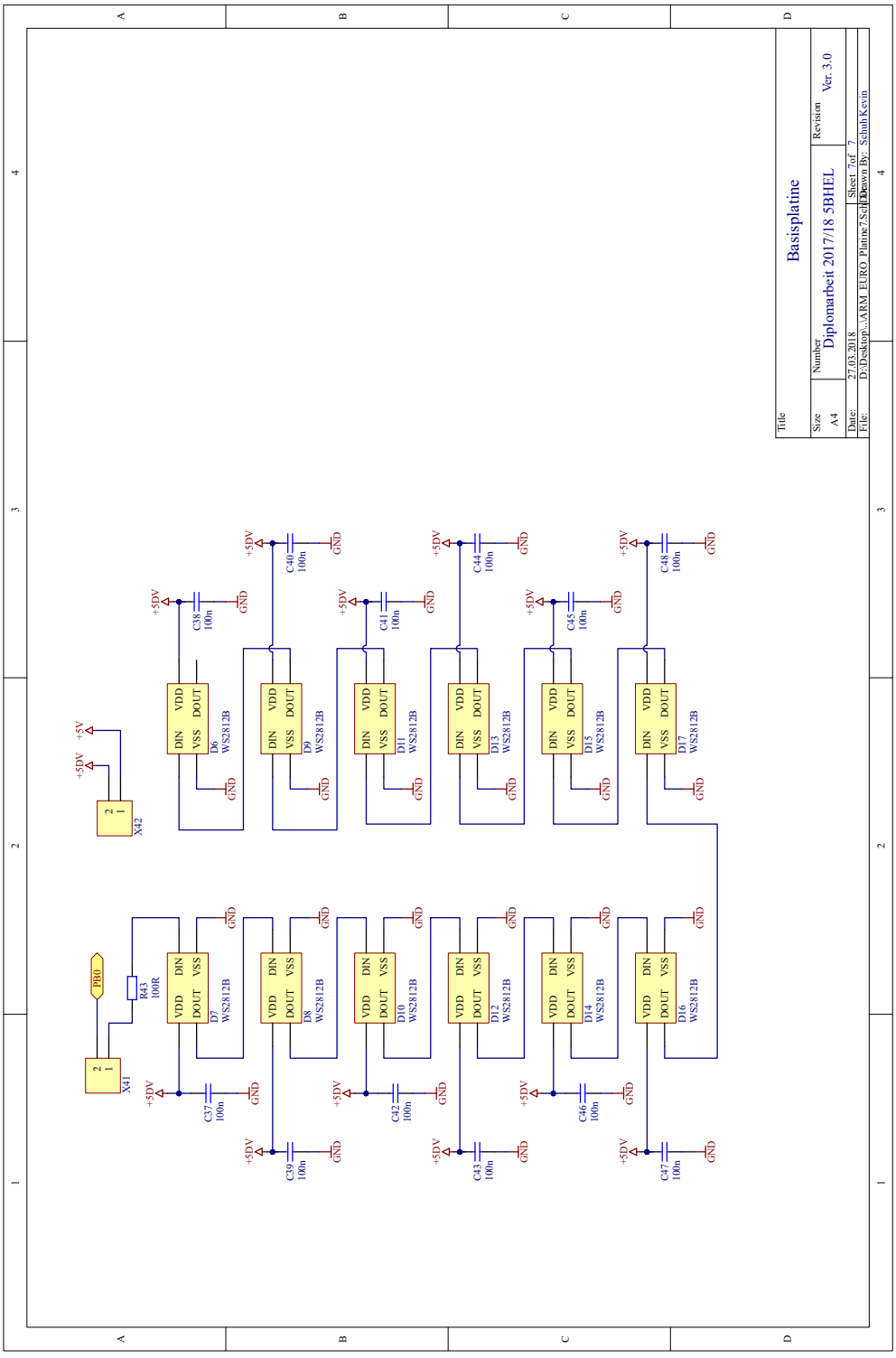


Abbildung 89: Gesamtschaltung der Basisplatine

3.5 Leiterplattenlayout

3.5.1 Bauteilseite

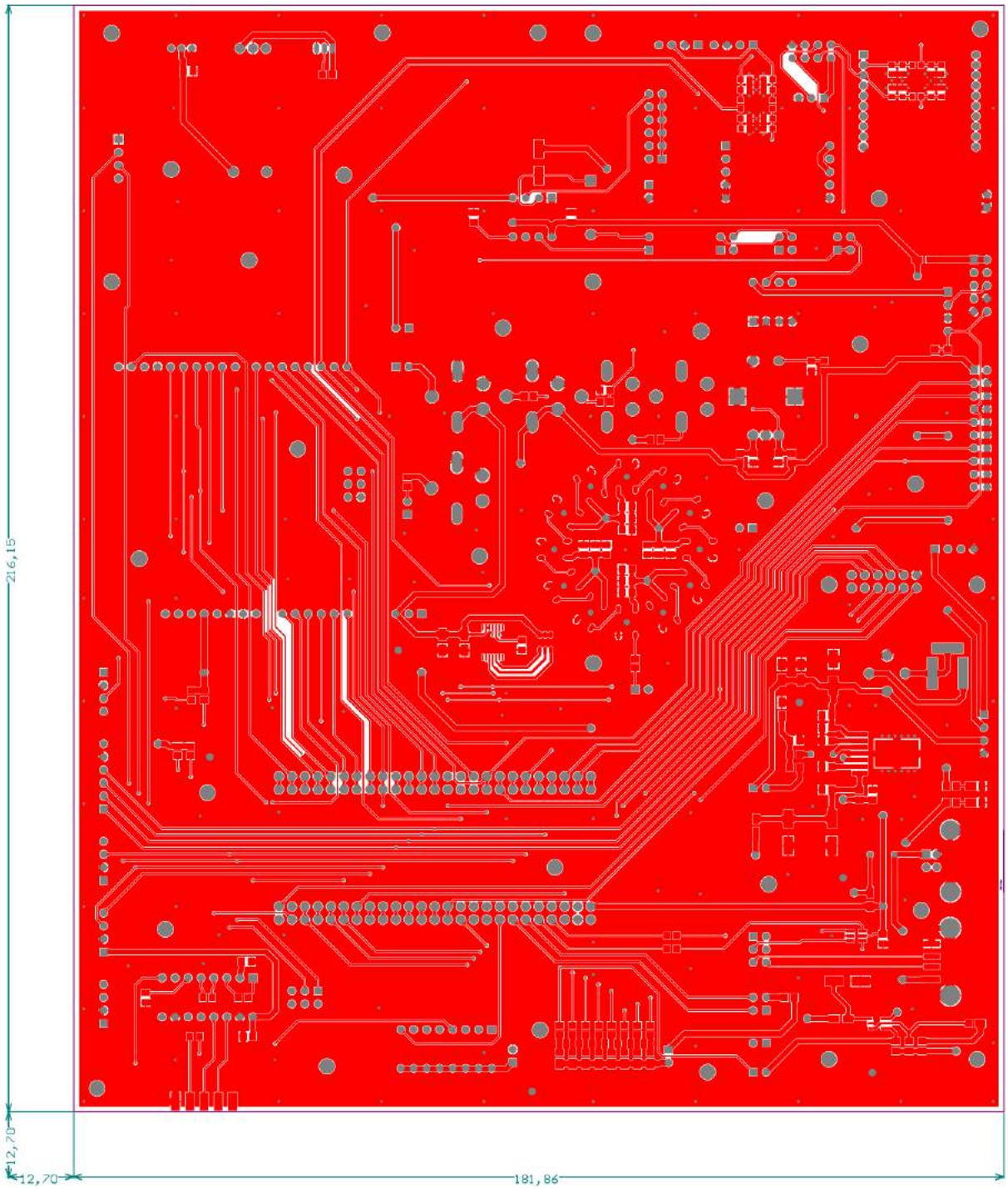


Abbildung 90: Layout Bauteilseite der Basisplatine

3.5.2 Lötseite

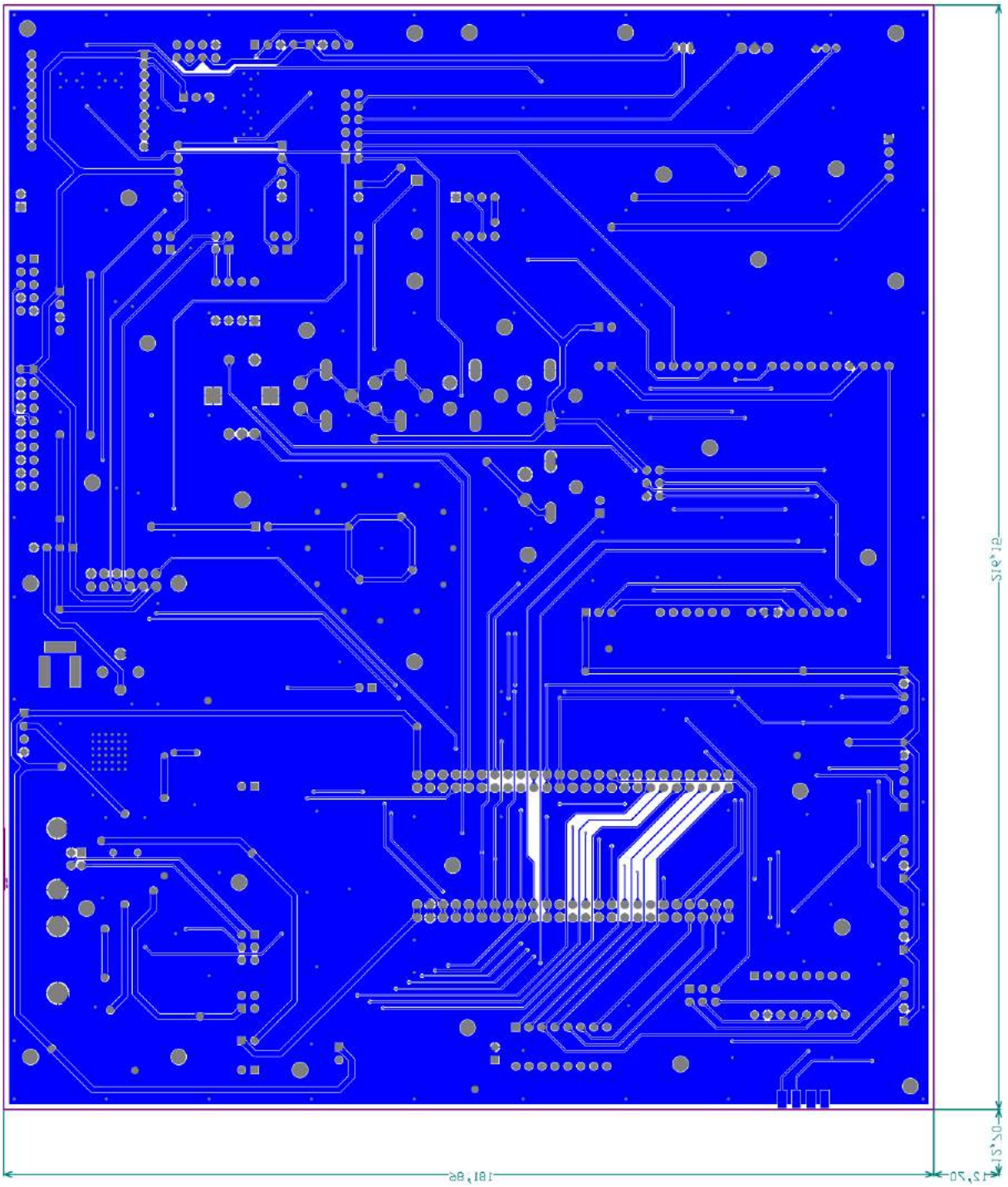


Abbildung 91: Layout Lötseite der Basisplatine

3.6 Bestückungspläne

3.6.1 Bauteilseite

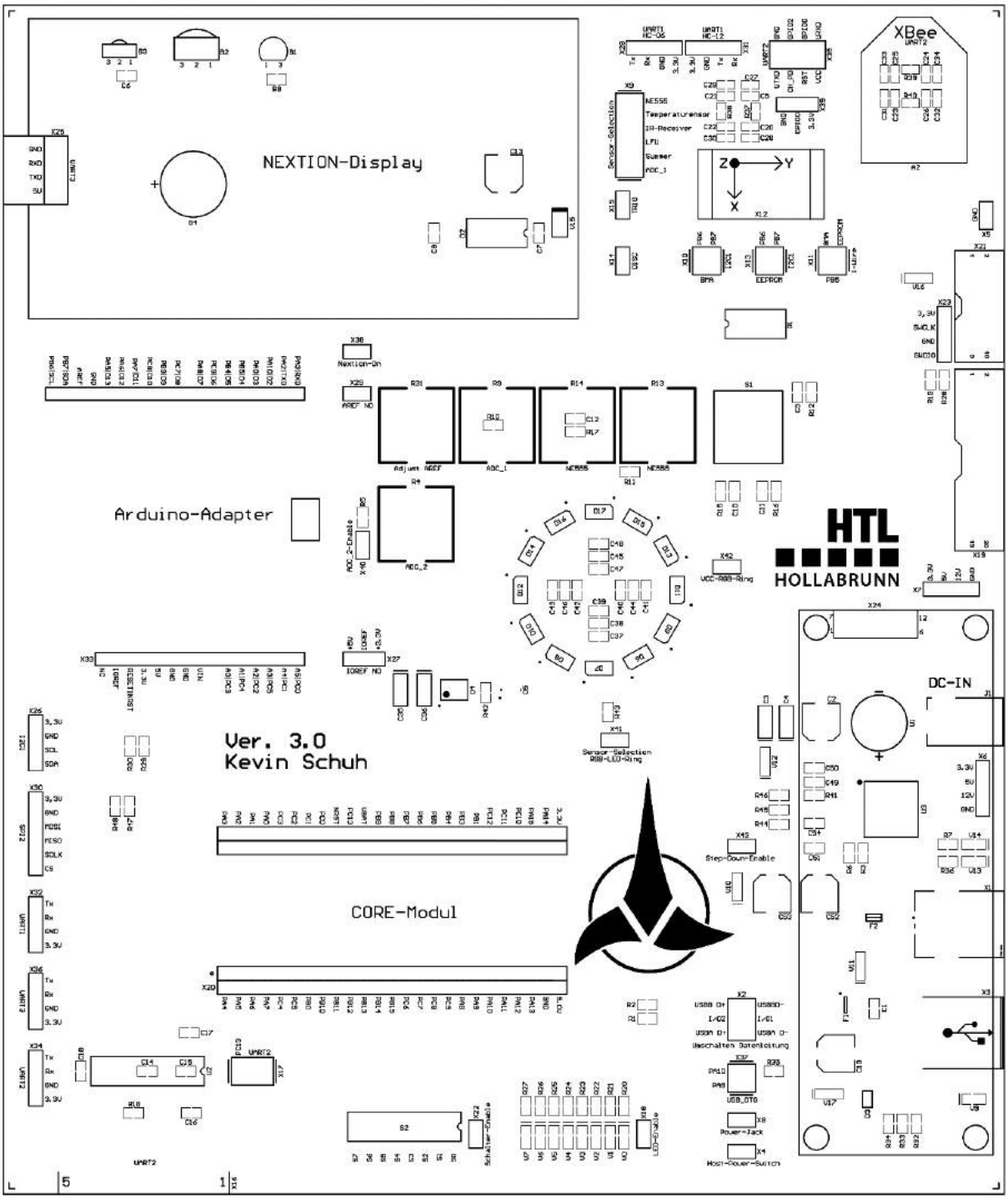


Abbildung 92: Bestückungsplan Bauteilseite der Basisplatine

4 USB-to-UART

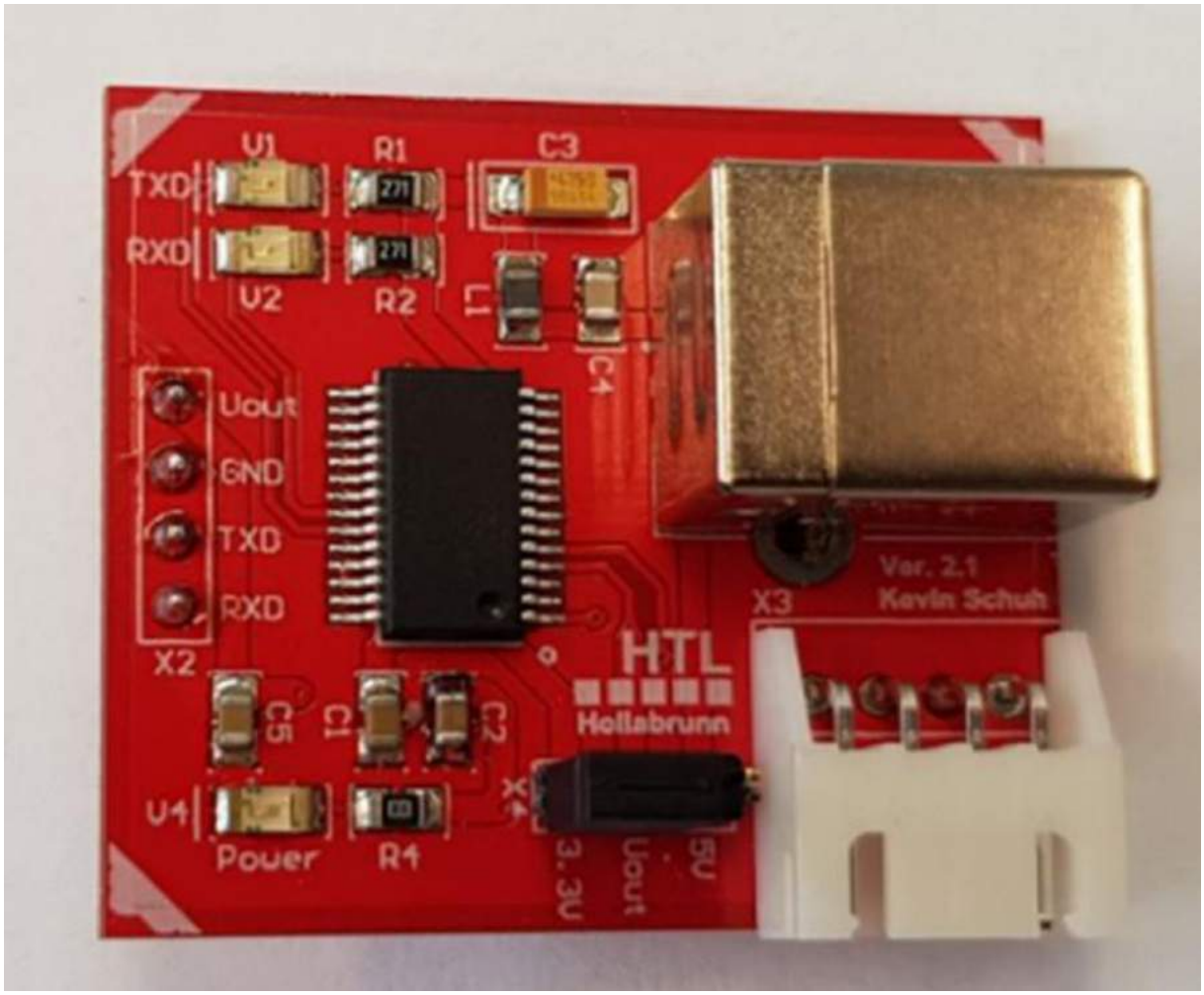


Abbildung 93: USB-to-UART-Adapter

4.1 Allgemeines

Der Adapter ermöglicht den Datenaustausch via USB zwischen einem PC und seriellen Geräten (z.B.: Mikrocontroller mit UART). Darüber hinaus soll dieses Modul eine einfache Programmierung und Kommunikation mit einem NEXTION-Display ermöglichen. Die Kommunikation zwischen den beiden Endgeräten wird durch den FTDI-Chip FT232RL ermöglicht. Dieser Chip ermöglicht eine Emulation einer seriellen Schnittstelle über USB.

4.2 Schnittstellen

Der USB-to-UART-Adapter verfügt über die in Tabelle 12 angegebenen Schnittstellen, welche wie in Abbildung 94 zu sehen platziert sind.

Schnittstelle	Funktion
UART	Senden und Empfangen von Daten vom Terminal oder Controller
USB-B	Weiterleitung der Daten zum PC

Tabelle 12: Schnittstellen des USB-to-UART-Adapter

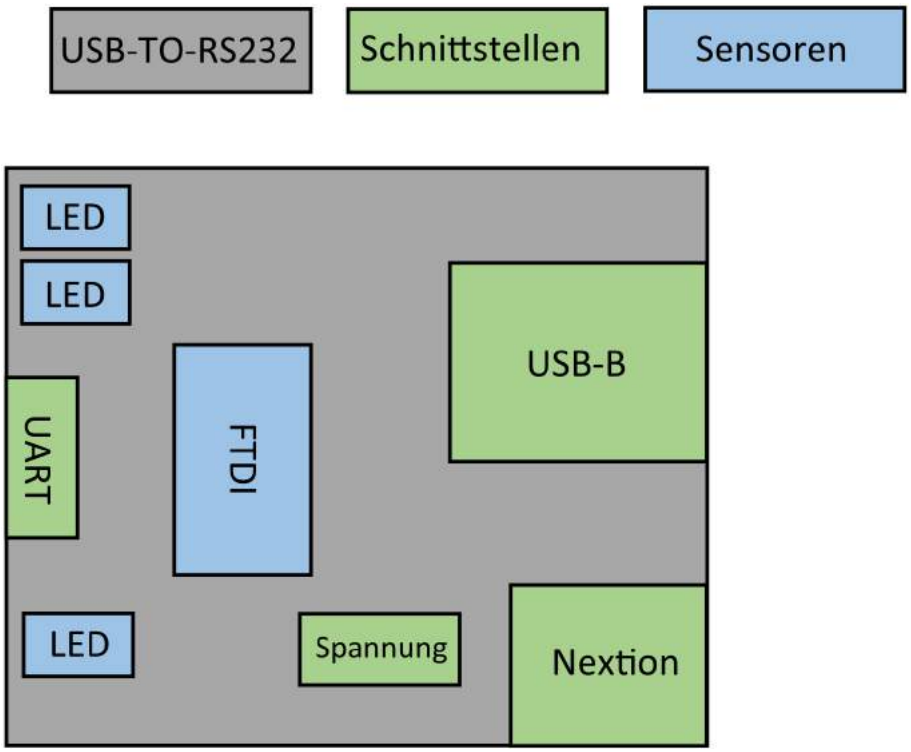
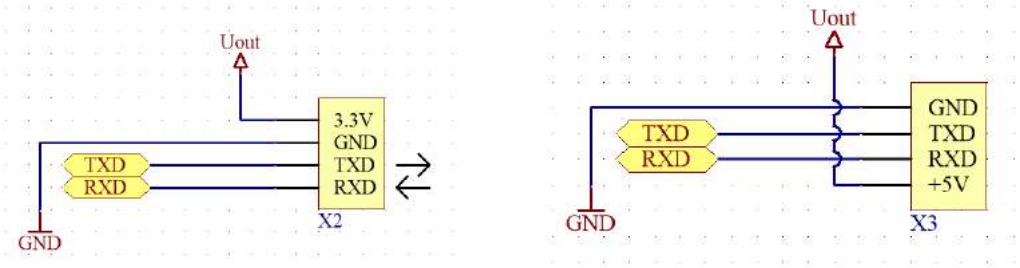


Abbildung 94: Übersichtsplan des USB-to-UART-Adapter

4.2.1 UART

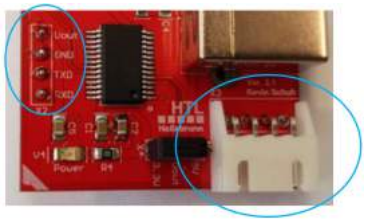
Die UART-Schnittstelle wurde hardwaremäßig zweimal herausgeführt. Einmal zu der Stiftleiste X2 (Abbildung 95a) und einmal zum Header X3 (Abbildung 95b). Die Stiftleiste X2 (Abbildung 95a) dient zur Kommunikation mit den diversen UART-Interfaces auf der Basisplatine, als auch zu Kommunikation mit der UART-Schnittstelle auf dem Core-

Modul. Der Header X3 (Abbildung 95b) dient zur Programmierung und Kommunikation mit einem NEXTION-Display.



(a) Schematic

(b) Schematic für NEXTION-Display



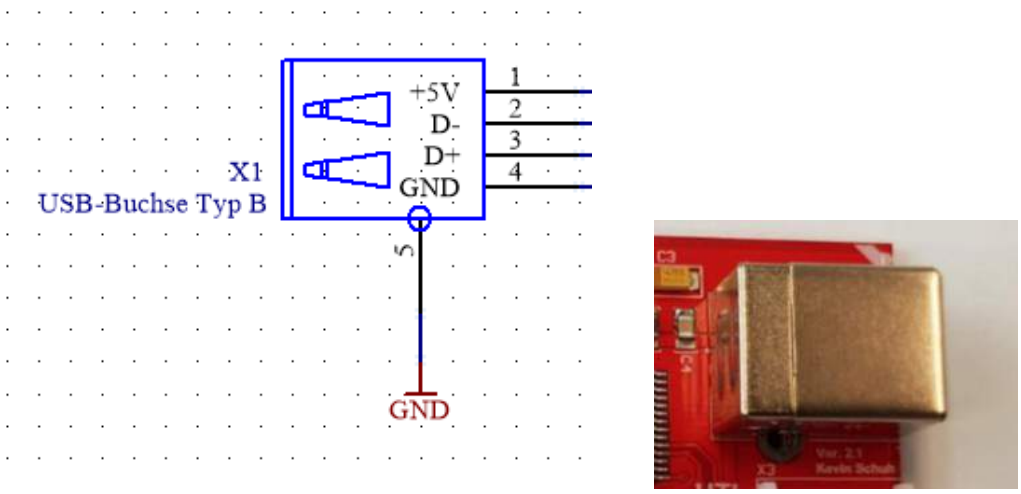
(c) Hardware

Abbildung 95: UART des USB-to-UART-Adapter

4.2.2 Spannungsversorgung

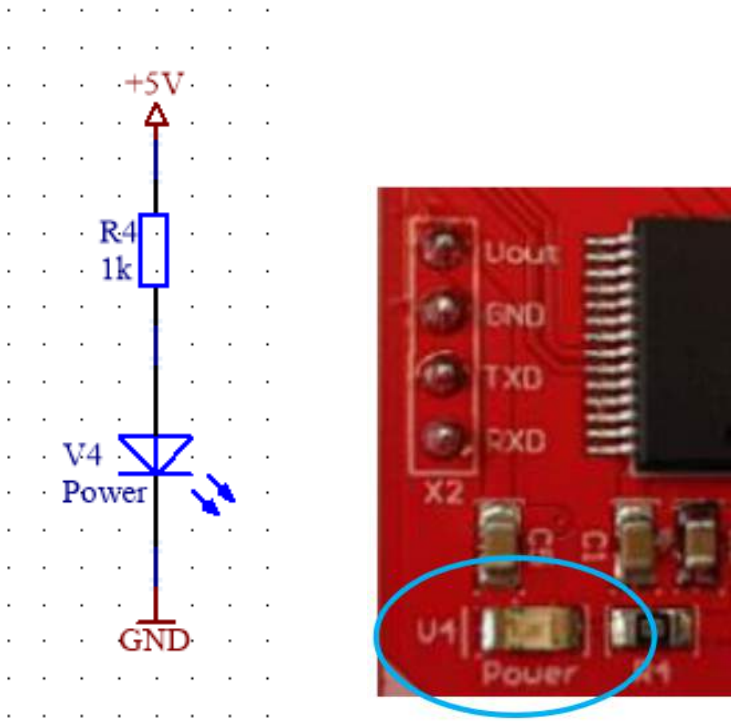
Die Betriebsspannungsversorgung von +5 V sollte prinzipiell nur über die USB-B Buchse X1 (Abbildung 96) zur Verfügung gestellt werden. Diese Spannungsversorgung wird auch zur Programmierung und Kommunikation mit dem NEXTION-Display oder mit anderen 5 V Interfaces benötigt. Zur Überprüfung ob das Modul mit Spannung versorgt wird, wurde die LED V4 (Abbildung 97) zur optischen Kontrolle eingebaut. Wenn das Modul mit Spannung über die USB-B Buchse versorgt wird, beginnt diese zu leuchten.

Es gibt zusätzlich zur Betriebsspannung noch eine vom FTDI-Chip generierte $+3,3$ V Spannungsversorgung, welche zur Kommunikation mit den verwendeten UART-Interfaces auf der Basisplatine und dem Core-Modul benötigt wird. Der Hardwareaufbau erlaubt es je nach verwendeten Interface, durch Setzen von einem Jumper auf der Stiftleiste X4 (Abbildung 98) die Betriebsspannungen für das entsprechende Interface selbst zu wählen. Durch Verbinden des Pin1 mit dem Pin2, wird am Ausgang eine Spannung von +5 V ausgegeben. Durch Verbinden des Pin2 mit dem Pin3, wird am Ausgang eine Spannung von $+3,3$ V ausgegeben.



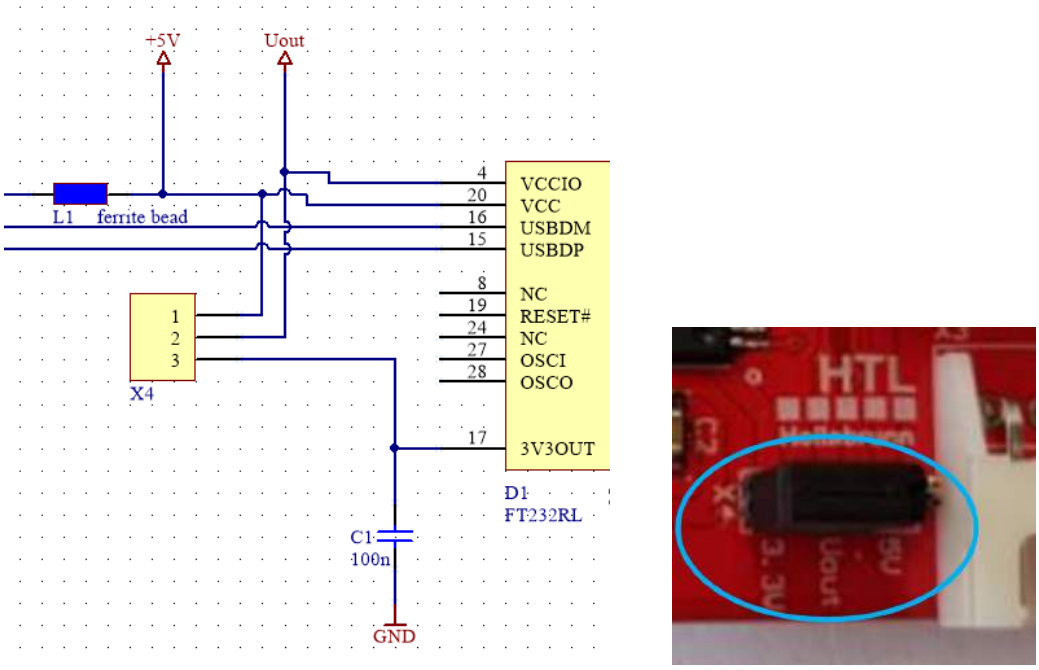
(a) Schematic (b) Hardware

Abbildung 96: Spannungsversorgung des USB-to-UART-Adapter



(a) Schematic (b) Hardware

Abbildung 97: Spannungsversorgungs-LED des USB-to-UART-Adapter

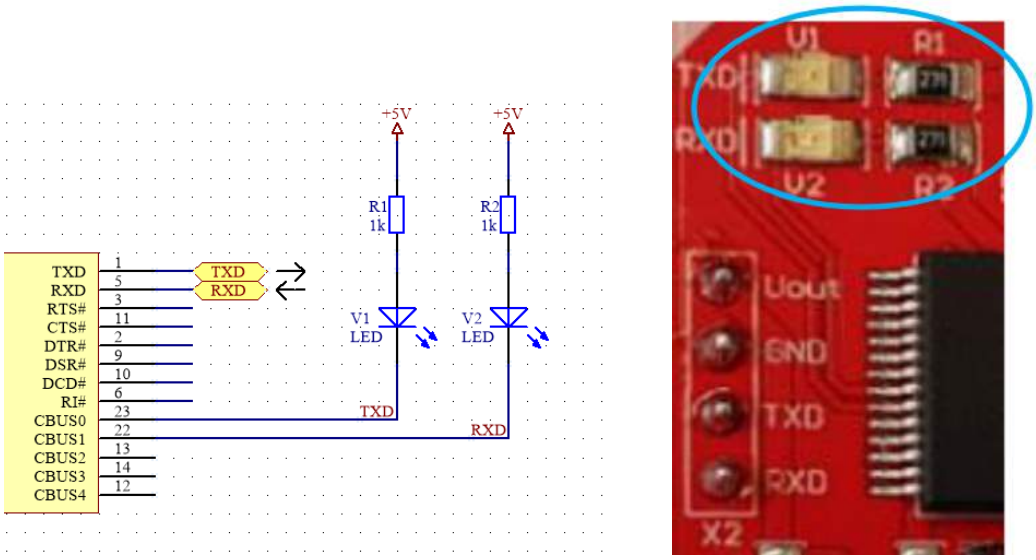


(a) Schematic (b) Hardware

Abbildung 98: Spannungsversorgungs-Jumper des USB-to-UART-Adapter

4.2.3 Status-LEDs

Die Status-LEDs V1 und V2 (Abbildung 99) dienen zu optischen Kontrolle, ob Daten zwischen den Sender den Empfänger ausgetauscht werden. Die LED V1 leuchtet, sobald Daten über das Modul gesendet werden. Die LED V2 hingegen zeigt an ob Daten zum Modul gelangen.

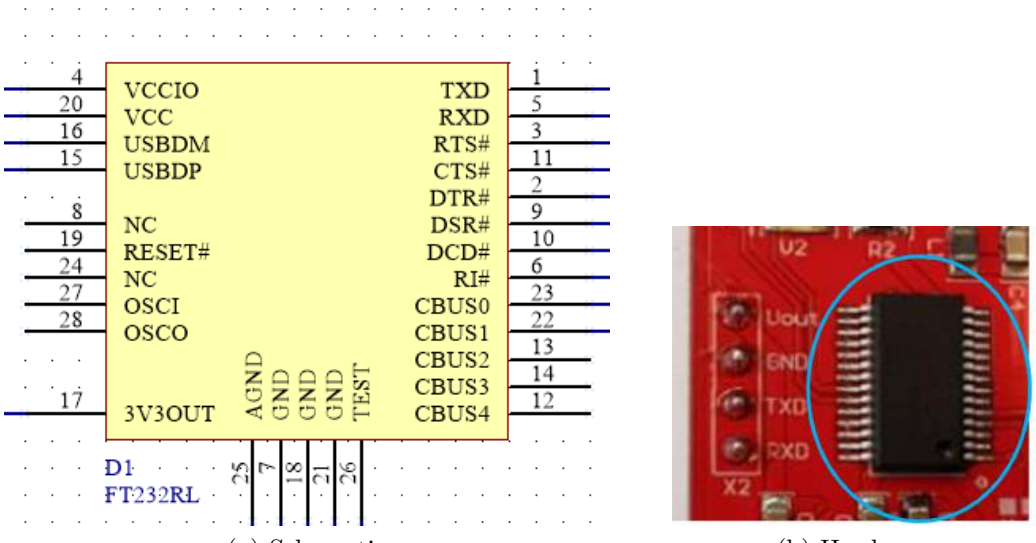


(a) Schematic (b) Hardware

Abbildung 99: Status-LEDs des USB-to-UART-Adapter

4.2.4 FTDI-Chip

Durch den FTDI-Chip FT232RL wird die eine Verwendung einer seriellen Schnittstelle über USB-Geräte ermöglicht.



(a) Schematic (b) Hardware

Abbildung 100: FTDI-Chip des USB-to-UART-Adapter

4.2.4.1 Blockschaltbild

Wie anhand des Blockschaltbildes ersichtlich ist, ist bereits ein $3,3$ V LDO-Regulator im FTDI-Chip eingebaut. Daher kann dieser die Ausgangsspannung von $+3,3$ V selbstständig generieren.

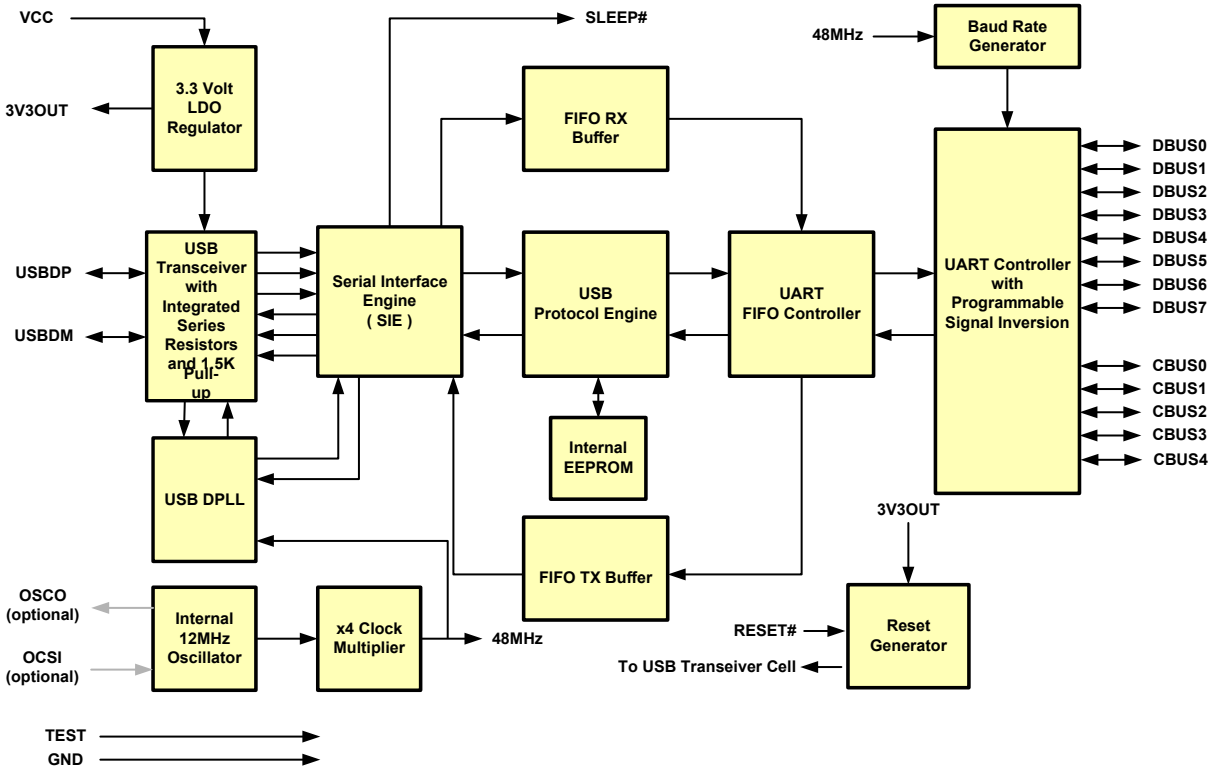


Abbildung 101: Blockschaltbild des FTDI-Chips [9]

4.2.5 Pinbelegung

Pin No.	Name	Type	Description
15	USBDP	I/O	USB Data Signal Plus, incorporating internal series resistor and 1.5kΩ pull up resistor to 3.3V.
16	USBDM	I/O	USB Data Signal Minus, incorporating internal series resistor.

Table 3.1 USB Interface Group

Pin No.	Name	Type	Description
4	VCCIO	PWR	+1.8V to +5.25V supply to the UART Interface and CBUS group pins (1...3, 5, 6, 9...14, 22, 23). In USB bus powered designs connect this pin to 3V3OUT pin to drive out at +3.3V levels, or connect to VCC to drive out at 5V CMOS level. This pin can also be supplied with an external +1.8V to +2.8V supply in order to drive outputs at lower levels. It should be noted that in this case this supply should originate from the same source as the supply to VCC. This means that in bus powered designs a regulator which is supplied by the +5V on the USB bus should

Tabelle 13: Pinbelegung des FTDI [9]

Pin No.	Name	Type	Description
			be used.
7, 18, 21	GND	PWR	Device ground supply pins
17	3V3OUT	Output	+3.3V output from integrated LDO regulator. This pin should be decoupled to ground using a 100nF capacitor. The main use of this pin is to provide the internal +3.3V supply to the USB transceiver cell and the internal 1.5kΩ pull up resistor on USBDP. Up to 50mA can be drawn from this pin to power external logic if required. This pin can also be used to supply the VCCIO pin.
20	VCC	PWR	+3.3V to +5.25V supply to the device core. (see Note 1)
25	AGND	PWR	Device analogue ground supply for internal clock multiplier

Table 3.2 Power and Ground Group

Pin No.	Name	Type	Description
8, 24	NC	NC	No internal connection
19	RESET#	Input	Active low reset pin. This can be used by an external device to reset the FT232R. If not required can be left unconnected, or pulled up to VCC.
26	TEST	Input	Puts the device into IC test mode. Must be tied to GND for normal operation, otherwise the device will appear to fail.
27	OSCI	Input	Input 12MHz Oscillator Cell. Optional – Can be left unconnected for normal operation. (see Note 2)
28	OSCO	Output	Output from 12MHZ Oscillator Cell. Optional – Can be left unconnected for normal operation if internal Oscillator is used. (see Note 2)

Table 3.3 Miscellaneous Signal Group

Pin No.	Name	Type	Description
1	TXD	Output	Transmit Asynchronous Data Output.
2	DTR#	Output	Data Terminal Ready Control Output / Handshake Signal.
3	RTS#	Output	Request to Send Control Output / Handshake Signal.
5	RXD	Input	Receiving Asynchronous Data Input.
6	RI#	Input	Ring Indicator Control Input. When remote wake up is enabled in the internal EEPROM taking RI# low (20ms active low pulse) can be used to resume the PC USB host controller from suspend.
9	DSR#	Input	Data Set Ready Control Input / Handshake Signal.
10	DCD#	Input	Data Carrier Detect Control Input.

Tabelle 13: Pinbelegung des FTDI [9]

Pin No.	Name	Type	Description
11	CTS#	Input	Clear To Send Control Input / Handshake Signal.
12	CBUS4	I/O	Configurable CBUS output only Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is SLEEP#. See CBUS Signal Options, Table 3.99.
13	CBUS2	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is TXDEN. See CBUS Signal Options, Table 3.99.
14	CBUS3	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is PWREN#. See CBUS Signal Options, Table 3.99. PWREN# should be used with a 10kΩ resistor pull up.
22	CBUS1	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is RXLED#. See CBUS Signal Options, Table 3.99.
23	CBUS0	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory default configuration is TXLED#. See CBUS Signal Options, Table 3.99.

Tabelle 13: Pinbelegung des FTDI [9]

4.3 Gesamtschaltung

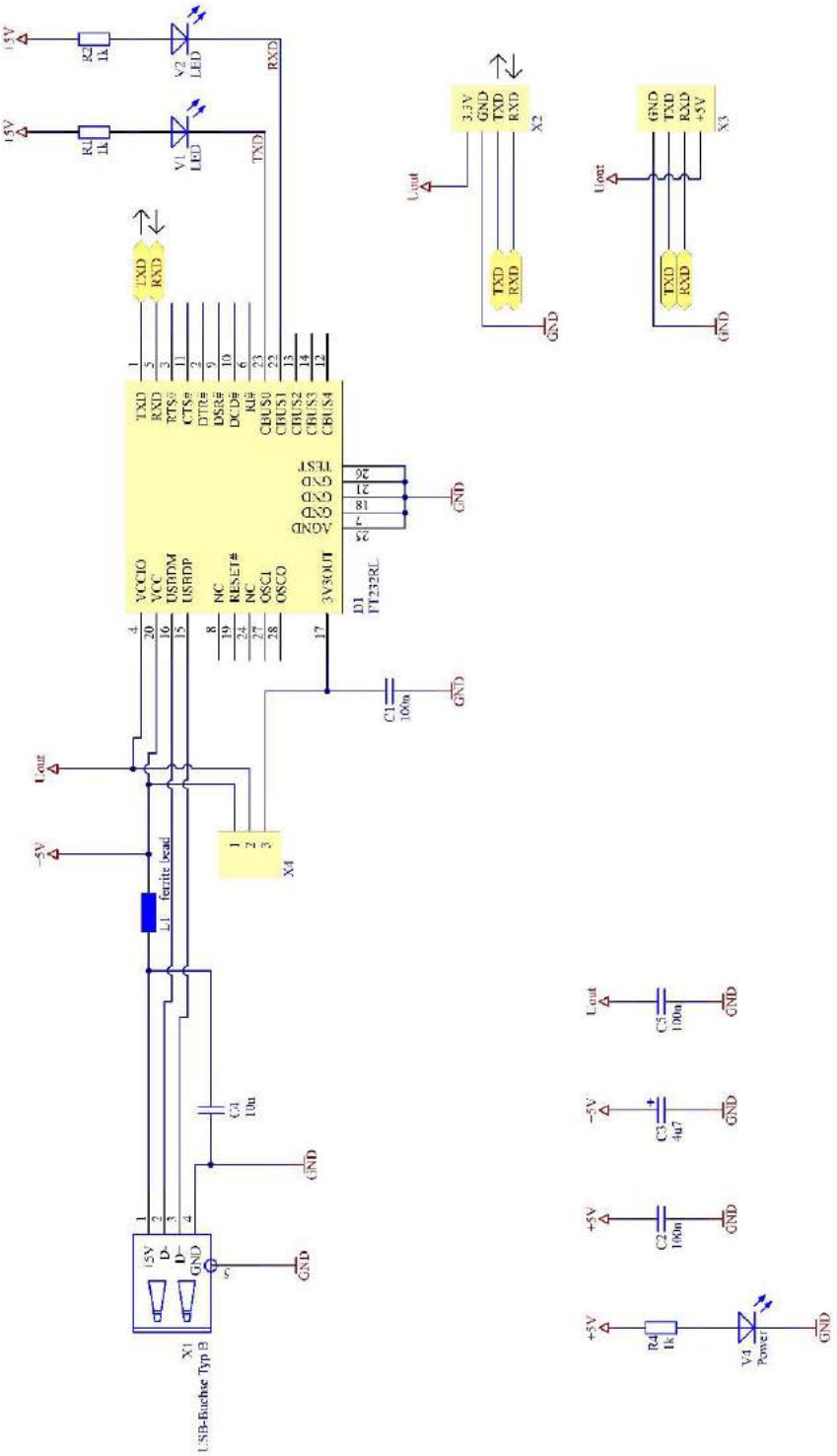


Abbildung 102: Gesamtschaltung des USB-to-UART-Adapter

4.4 Leiterplattenlayout

4.4.1 Bauteilseite

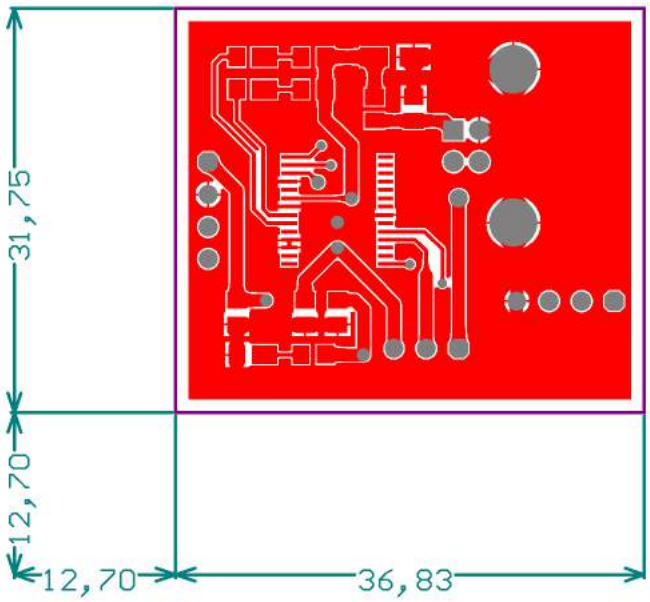


Abbildung 103: Layout Bauteilseite des USB-to-UART-Adapters

4.4.2 Lötseite

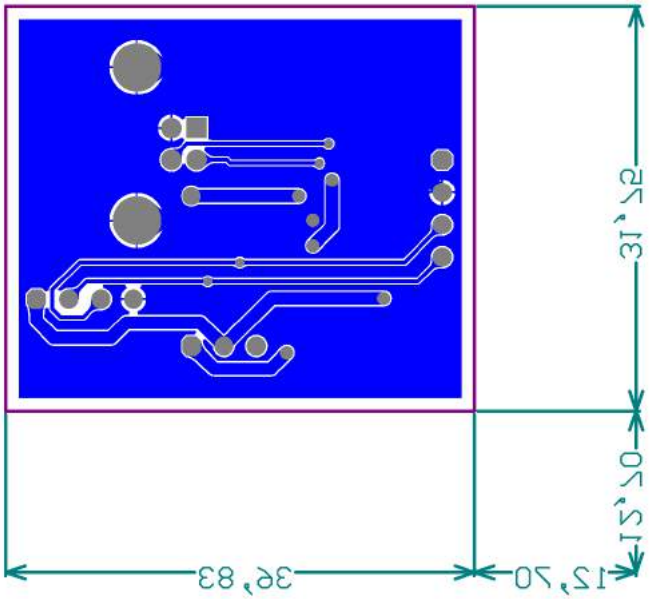


Abbildung 104: Layout Lötseite des USB-to-UART-Adapters

4.5 Bestückungspläne

4.5.1 Bauteilseite

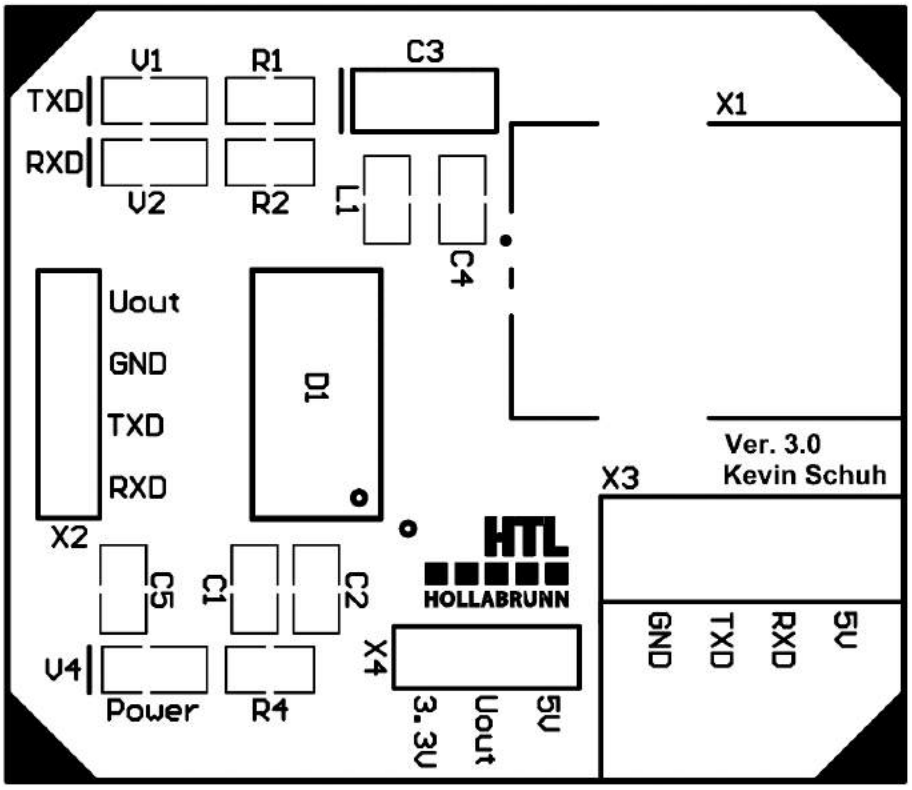


Abbildung 105: Bestückungsplan Bauteilseite des USB-to-UART-Adapters

5 Audioadapter [10]

Der Audioadapter wurde entwickelt, um Echtzeit Audioverarbeitung (Signalverarbeitung) mit dem ARM-Minimalsystem zu ermöglichen. Die eingelesenen Audiosignale, die beispielsweise von einem Mikrofon geliefert werden, werden abgetastet und mit einem ADC in einen digitalen Wert gewandelt und via I^2S zum Microcontroller gesendet. Dort werden sie dann verarbeitet. Schließlich wird das Audiosignal dann via I^2S zu einem DAC gesendet und Schließlich wird das analoge Signal wieder auf einem Lautsprecher ausgegeben.

5.1 Aufbau des neuen Audioadapters

5.1.1 Blockschaltbild

Der Audioadapter besteht prinzipiell aus den Ein- und Ausgängen (Klinke und Cinch), aus einem Audiocodec und einem analogen Schalter (Analog Switch – TS5A22364). Mit dem Analog Switch kann zwischen den beiden Eingängen, Klinke und Cinch, gewählt werden. Dazu muss einfach der Auswahl-Jumper entsprechend umgesteckt werden. Der durchgeschaltete Eingang wird mit den analogen Eingängen (linker und rechter Kanal) des Audiocodecs TLV320AIC23B verbunden. Das analoge Signal wird mit einem ADC in ein digitales Signal umgewandelt. Die genauere Funktion des Audiocodecs ist unter Abschnitt 5.1.3 erklärt. Nach dem DAC erfolgt die Ausgabe auf beide Ausgänge.

Die Auswahl fiel auf den Audiocodec TLV320AIC23B weil dieser einen „Crystal Input“ besitzt. Dadurch kann die Taktversorgung direkt auf dem Audioadapter durch einen 12 MHz Quarz erfolgen. Außerdem gibt es einen fertigen VHDL-Code, der von Professor Hermann Dum geschrieben und zur Verfügung gestellt wurde, mit dem der Audiocodec über das FPGA-Board BASYS2 konfiguriert werden kann. Ein Entwicklungsmodul mit dem Audiocodec TLV320AIC23B wird auch im Labor verwendet. Dies erleichterte das Design der neuen Schaltung für den Audioadapter, die Testung des fertigen Audioadapters, so wie das Konfigurieren des Audiocodecs über I^2C mit dem ARM Minimalsystem.

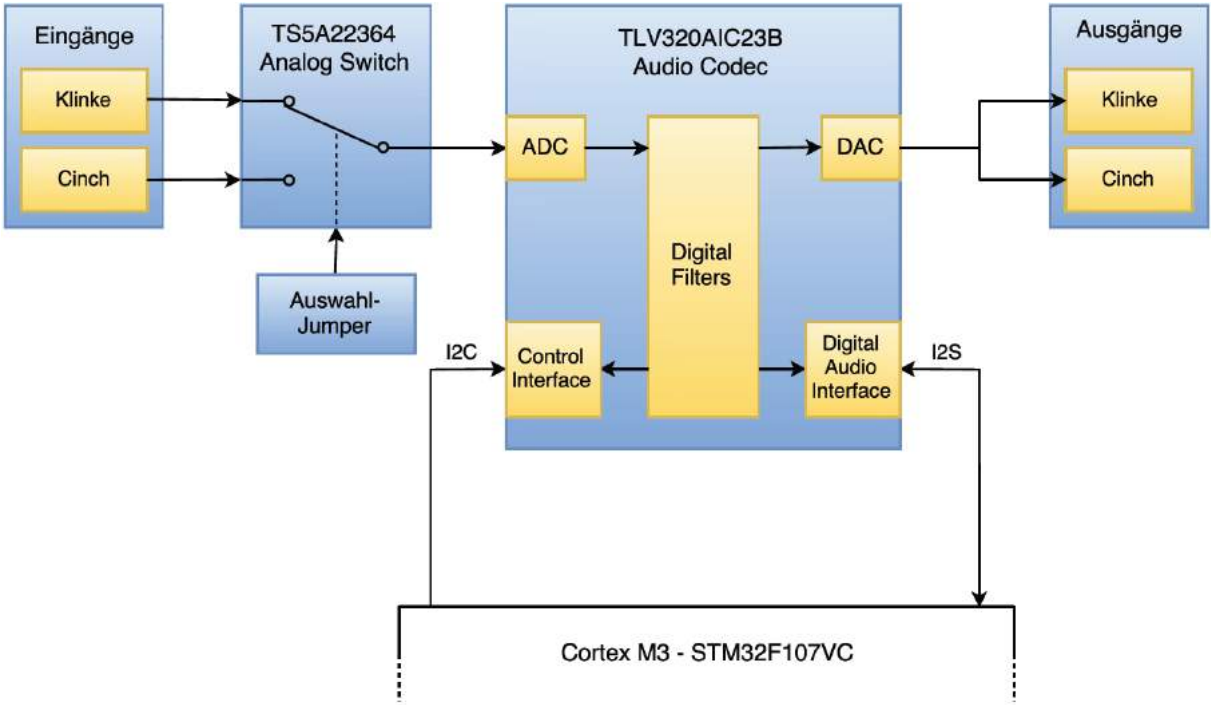


Abbildung 106: Blockschaltbild der Audioplatine

5.1.2 Schematic

In Abbildung 107 sieht man die gesamte Schaltung des Audioadapters. Die einzelnen Funktionsblöcke werden in den nächsten Punkten genauer erklärt.

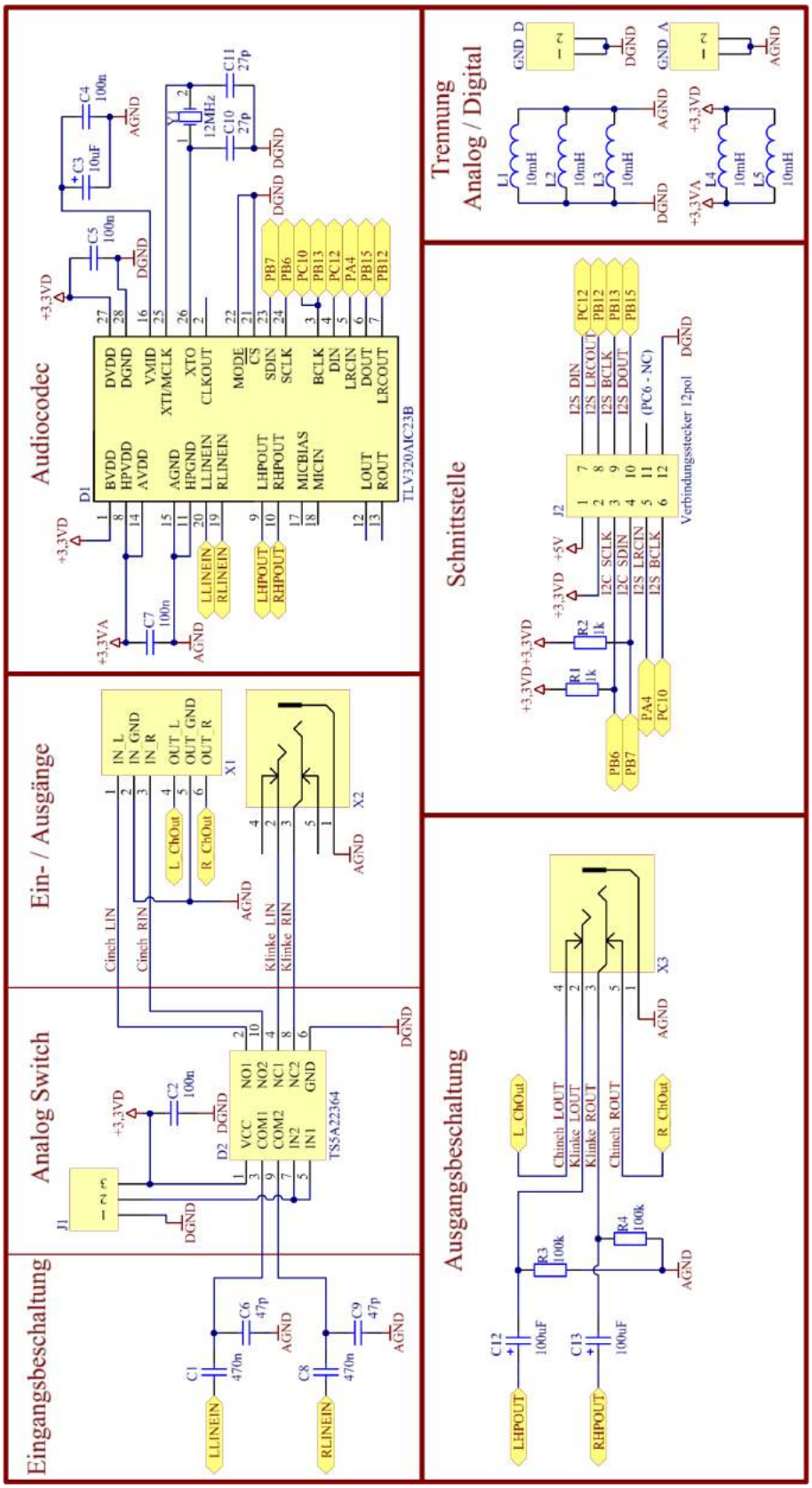


Abbildung 107: Gesamtschematic der Audioplatine

5.1.3 Audio Codec TLV320AIC23B [11]

Daten:

- High Performance Stereo Codec
- 90 dB Signal-Noise-Ratio (SNR) Multibit Sigma-Delta ADC
- 100 dB SNR Multibit Sigma-Delta DAC
- Sample-Frequenz 8 kHz - 96 kHz
- 2-Wire SPI-compatible Serial-Port Protocols
- I^2S-compatible Interface
- Standard I^2S, Most Significant Bit (MSB) or Least Significant Bit (LSB) justified data transfer
- 16/20/24/32 bit Wortbreite

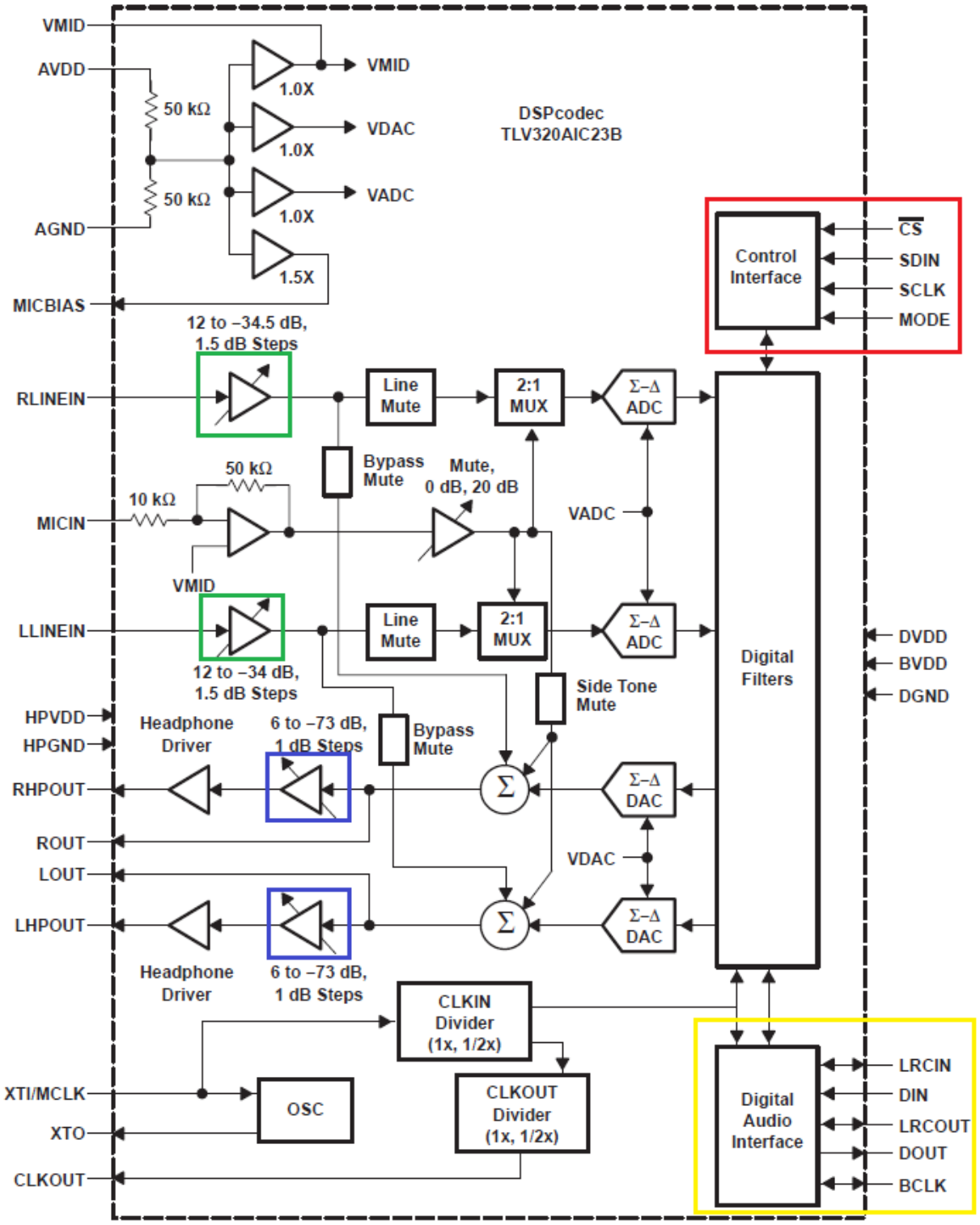


Abbildung 108: Funktionsblockschaltbild des Audiocodec [11]

Von den analogen Eingängen wurden nur LLINEIN und RLINEIN verwendet. Der Ein-

gang MICIN ist unbelegt. Die Eingänge können getrennt durch softwareseitige Konfiguration der entsprechenden Register in $1,5$ dB Schritten von $+12$ dB bis -34 dB verstärkt bzw. abgeschwächt werden (Abbildung 108: grün markiert). Von den Ausgängen wurden nur die Headphone-Ausgänge verwendet. Diese können ebenfalls durch Konfiguration der entsprechenden Register verstärkt oder abgeschwächt werden. Dies ist in 1 dB Schritten von $+6$ dB bis -73 dB möglich (Abbildung 108: blau markiert). Die Konfiguration wird über den Mode-Pin (Pin 22) nach Tabelle 14 festgelegt.

Mode	**Interface**
0	I^2C
1	SPI

Tabelle 14: Modi des Audiocodec [11]

Im SPI-Modus werden die Daten über SDIN (Pin 23) übertragen. SCLK (Pin 24) ist der serielle Takt. $\overline{CS}$ (Pin 21) wird `0` wenn die Übertragung eines Datenworts abgeschlossen ist.

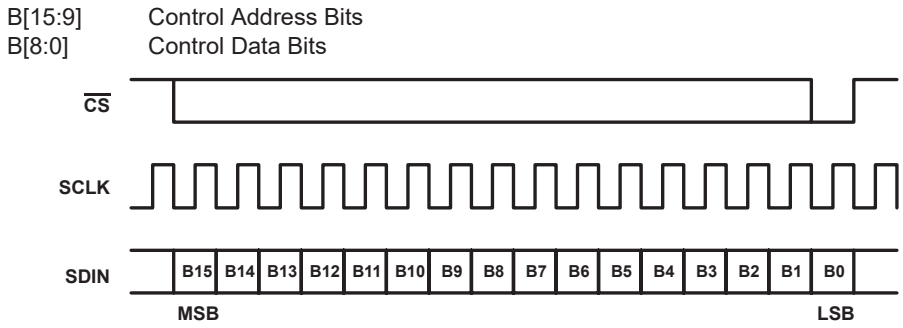


Abbildung 109: SPI-Timing des Audiocodec [11]

Im I^2C-Modus werden die Daten über SDIN (Pin 23) übertragen und der serielle Takt über SCLK (Pin 24). $\overline{CS}$ (Pin 21) stellt die Adresse ein (Tabelle 15) und ist hardwaremäßig auf GND (`0`) geschaltet.

$\overline{CS}$	**Adresse**
0	`0011010`
1	`0011011`

Tabelle 15: I^2C-Adresse des Audiocodec [11]

B[15:9] Control Address Bits
B[8:0] Control Data Bits

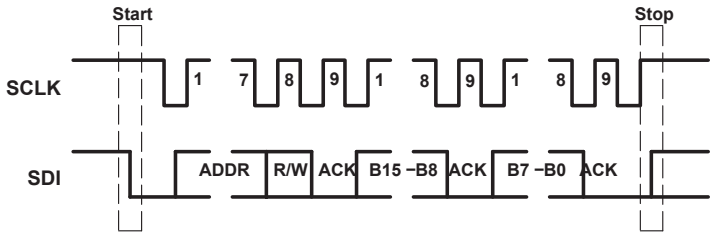


Abbildung 110: I²C-Timing des Audiocodec [11]

Da der Mode-Pin bei der Audioadapterplatine hardwaremäßig auf `0` gelegt wurde, erfolgt die Konfiguration mit I²C über das Control Interface (Abbildung 108: rot markiert). Die Audiodatenübertragung erfolgt im Digital Audio Interface über I²S (Abbildung 108: gelb markiert). Da der Audiocodec bei unserer Anwendung im Master Modus arbeitet (kann im Register „Digital Audio Interface Format“ konfiguriert werden), wird der Takt BCLK, LRCOUT und LRCIN vom Audiocodec erzeugt.

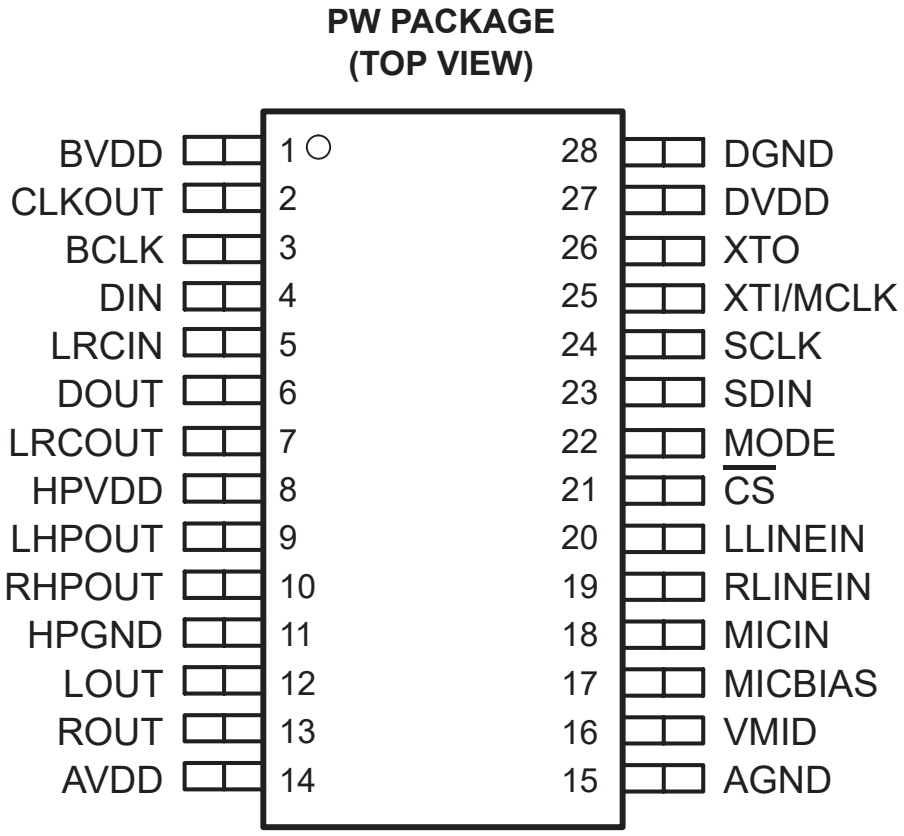


Abbildung 111: Pinbelegung des Audiocodec [11]

Name	**Pin**	**Beschreibung**
AGND	15	Analog supply return
AVDD	14	Analog supply input. Voltage level is $3,3$ V nominal
BCLK	3	I^2S serial-bit clock. In master-mode: AIC23B generates signal; in slave-mode: DSP generates signal
BVDD	1	Buffer supply input, Voltage range is from $2,7$ V to $3,6$ V
CLKOUT	2	Clock output, buffered version of XTI input, 1x or 1/2x frequency
$\overline{CS}$	21	Control port input latch/address select. SPI: data latch control/I^2C: address pin
DIN	4	I^2S format serial data input to the sigma-delta stereo DAC
DGND	28	Digital supply return
DOUT	6	I^2S format serial data output from the sigma-delta stereo ADC
DVDD	27	Digital supply input. Voltage range is $1,4$ V to $3,6$ V.
HPGND	11	Analog headphone amplifier supply return
HPVDD	8	Analog headphone amplifier supply input. Voltage level is $3,3$ V nominal.
LHPOUT	9	Left stereo mixer-channel amplified headphone output. Nominal 0 dB output level is 1 V_{RMS}. Gain of –73 dB to 6 dB is provided in 1 dB steps.
LLINEIN	20	Left stereo-line input channel. Nominal 0 dB input level is 1 V_{RMS}. Gain of $-34,5$ dB to 12 dB is provided in $1,5$ dB steps.
LOUT	12	Left stereo mixer-channel line output. Nominal output level is 1 V_{RMS}.
LRCIN	5	I^2S DAC-word clock signal. In audio master mode, the AIC23B generates this framing signal and sends it to the Digital Signal Processor (DSP). In audio slave mode, the signal is generated by the DSP.
LRCOUT	7	I^2S ADC-word clock signal. In audio master mode, the AIC23B generates this framing signal and sends it to the DSP. In audio slave mode, the signal is generated by the DSP.
MICBIAS	17	Buffered low-noise-voltage output suitable for electret-microphone-capsule biasing. Voltage level is 3/4 AVDD nominal.
MICIN	18	Buffered amplifier input suitable for use with electret-microphone capsules. Without external resistors a default gain of 5 is provided.

MODE	22	Serial-interface-mode input.
RHPOUT	10	Right stereo mixer-channel amplified headphone output. Nominal 0-dB output level is 1 V_{RMS}. Gain of –73 dB to 6 dB is provided in 1 dB steps.
RLINEIN	19	Right stereo-line input channel. Nominal 0-dB input level is 1 V_{RMS}. Gain of $-34,5$ dB to 12 dB is provided in $1,5$ dB steps.
ROUT	13	Right stereo mixer-channel line output. Nominal output level is 1 V_{RMS}.
SCLK	24	Control-port serial-data clock. For SPI and 2-wire control modes this is the serial-clock input.
SDIN	23	Control-port serial-data input. For SPI and 2-wire control modes this is the serial-data input and also is used to select the control protocol after reset.
VMID	16	Midrail voltage decoupling input. 10 µF and $0,1$ µF capacitors should be connected in parallel to this terminal for noise filtering. Voltage level is 1/2 AVDD nominal.
XTI/MCLK	25	Crystal or external-clock input. Used for all internal clocks on the AIC23B.
XTO	26	Crystal output. Connect to external crystal when the AIC23B is the master

Tabelle 16: Pinbelegung des Audiocodec [11]

Name	**Pin**	**Beschreibung**
LOUT	12	Die Ausgänge LOUT und ROUT wurden freigelassen, da nur die verstärkbaren/abschwächbaren Ausgänge LHPOUT und RHPOUT verwendet wurden.
ROUT	13	
MICBIAS	17	MICBIAS und MICIN wurden nicht beschaltet, da für unsere Anwendung nur 2 Eingänge gebraucht werden und der MIC-Eingang einen zusätzlichen Eingang benötigen würde.
MICIN	18	
CLKOUT	2	Der CLKOUT (12 MHz bzw 6 MHz) wird für keine Anwendung benötigt.

Tabelle 17: Nicht benutzte Pins des Audiocodec [11]

ADDRESS	REGISTER
0000000	Left line input channel volume control
0000001	Right line input channel volume control
0000010	Left channel headphone volume control
0000011	Right channel headphone volume control
0000100	Analog audio path control
0000101	Digital audio path control
0000110	Power down control
0000111	Digital audio interface format
0001000	Sample rate control
0001001	Digital interface activation
0001111	Reset register

Tabelle 18: Register des Audiocodec [11]

5.1.4 Analog Switch TS5A22364 [12]

Der TS5A22364 ist ein bidirektionaler, analoger Schalter mit 2 Kanälen. Er wurde ausgewählt, da er im Gegensatz zu anderen analogen Schaltern keine negative Versorgungsspannung benötigt, um trotzdem negative Signale durchzulassen. Die maximale positive Spannung, die er durchlassen kann entspricht VCC, die maximale negative Spannung VCC – 5,5 V. Das ergibt einen Durchlassbereich von $+3,3$ V bis $-2,2$ V und ist somit ausreichend für die Eingangssignale des Audioadapters.

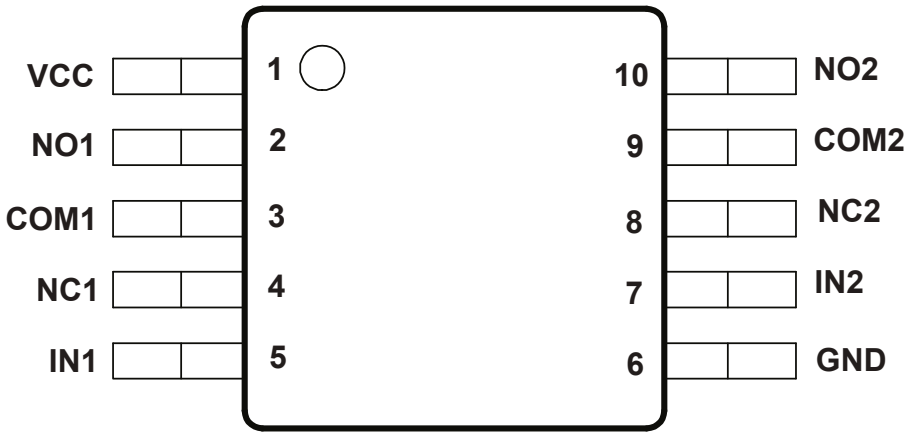


Abbildung 112: Pinbelegung des Analog Switch [12]

Name	Pin	Beschreibung
VCC	1	Power Supply
NO1	2	Normally Open (NO) signal path, Switch 1
COM1	3	Common signal path, Switch 1
NC1	4	Normally Closed (NC) signal path, Switch 1
IN1	5	Digital control pin to connect COM1 to NO1 or NC1, Switch 1
GND	6	Ground
IN2	7	Digital control pin to connect COM2 to NO2 or NC2, Switch 2
NC2	8	Normally Closed (NC) signal path, Switch 2
COM2	9	Common signal path, Switch 2
NO2	10	Normally Open (NO) signal path, Switch 2

Tabelle 19: Pinbelegung des Analog Switch [12]

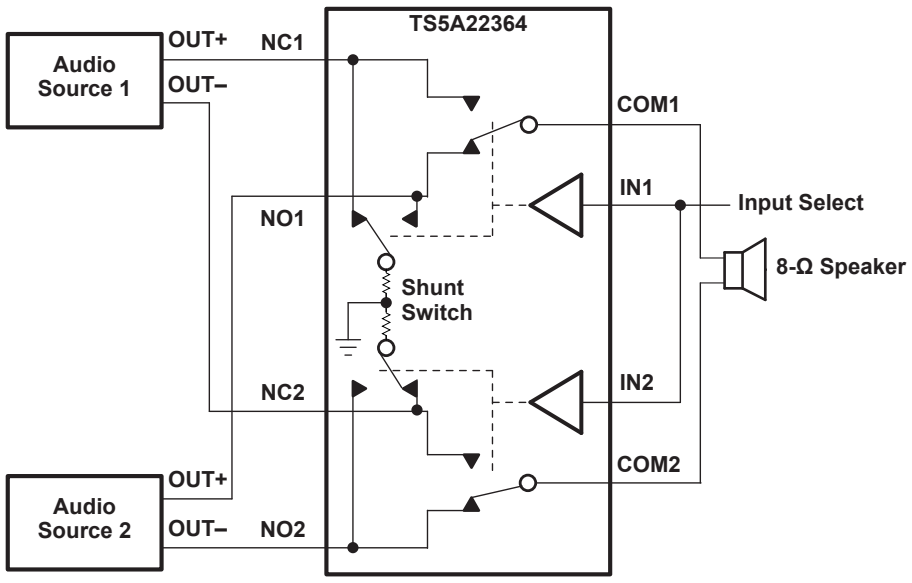


Abbildung 113: Funktionsblockschaltbild des Analog Switch [12]

Die zwei Audioquellen auf der linken Seite stellen die Eingänge des Audioadapters (Klinke und Cinch) dar. Der Lautsprecher stellt den Ausgang dar, und wiedergibt je nach Input Select eine der beiden Audioquellen. Der interne Shunt Switch verhindert „Click and Pop“ beim Umschalten zwischen den Audioquellen. Der Input Select wurde mit einem Jumper zum Wechseln zwischen GND und VCC realisiert. Welche Jumperstellung welchen Eingang durchschaltet ist auf dem Audioadapter beschrieben und wird hier nochmal mit den nächsten beiden Abbildungen dargestellt.

Die in Abbildung 114a gezeigte Jumperstellung verbindet die miteinander verbundenen

IN-Pins mit VCC und der Cincheingang wird durchgeschaltet. Die in Abbildung 114b gezeigte Jumperstellung verbindet die IN-Pins mit GND und der Klinkeneingang wird durchgeschaltet.

(a) Chinch (b) Klinke

Abbildung 114: Analog Switch Jumperpositionen

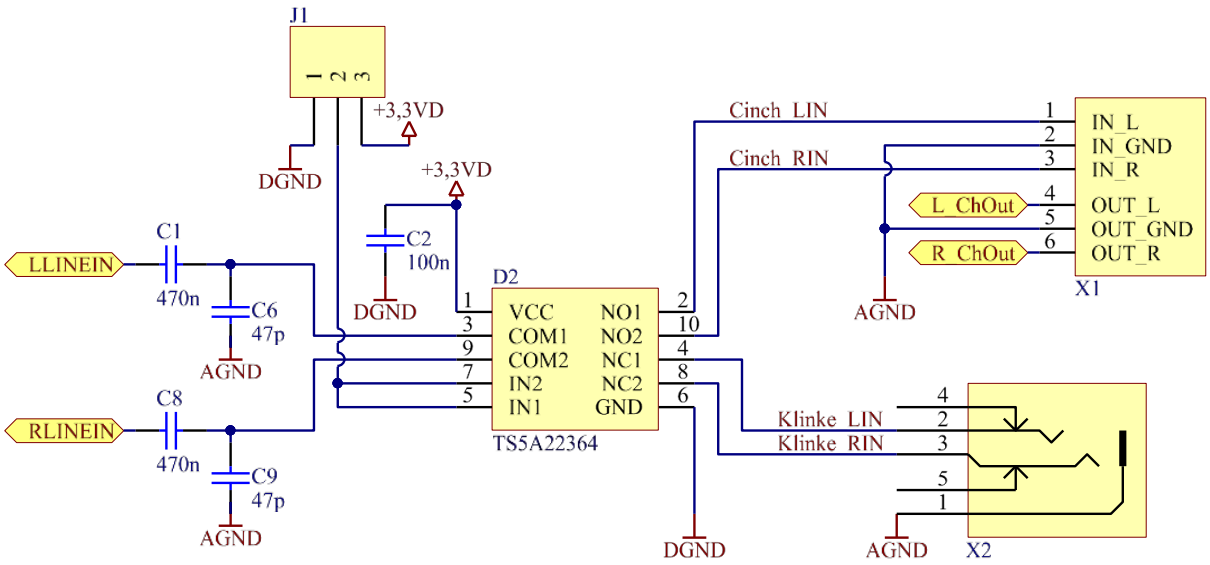


Abbildung 115: Schematic des Analog Switch

Zum Testen des analogen Switches wurde jeweils an einem Kanal der Cincheingänge und an einem Kanal des Klinkeneingangs ein Signal angelegt. Am Cincheingang liegt ein 300 mV Sinussignal mit 1 kHz an (Abbildung 116: gelbes, oberstes Signal). Am Klinkeneingang liegt ein 300 mV Sinussignal mit 2 kHz an (Abbildung 116: blaues, mittleres Signal).

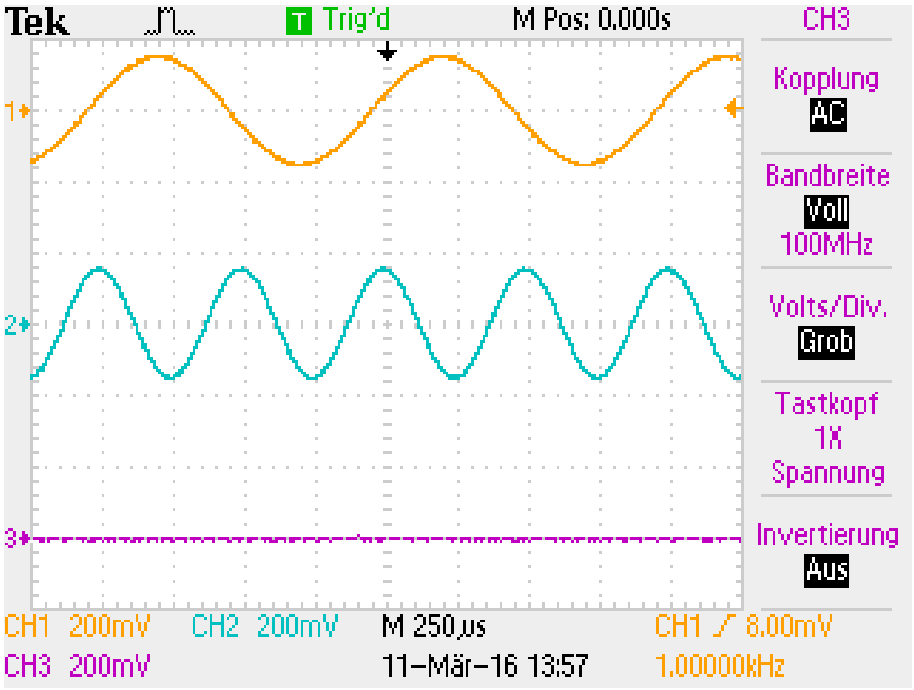


Abbildung 116: Eingangssignale am Analog Switch

Das Ausgangssignal ist in den folgenden Abbildungen als unterstes Signal beim Oszilloskop zu sehen. Die Eingangssignale schauen am Oszilloskop, vermutlich durch das Verwenden der T-Stücke am Frequenzgenerator, etwas verrauscht aus. Als erstes wurde der Jumper J1 auf „Chinch“ umgesteckt.

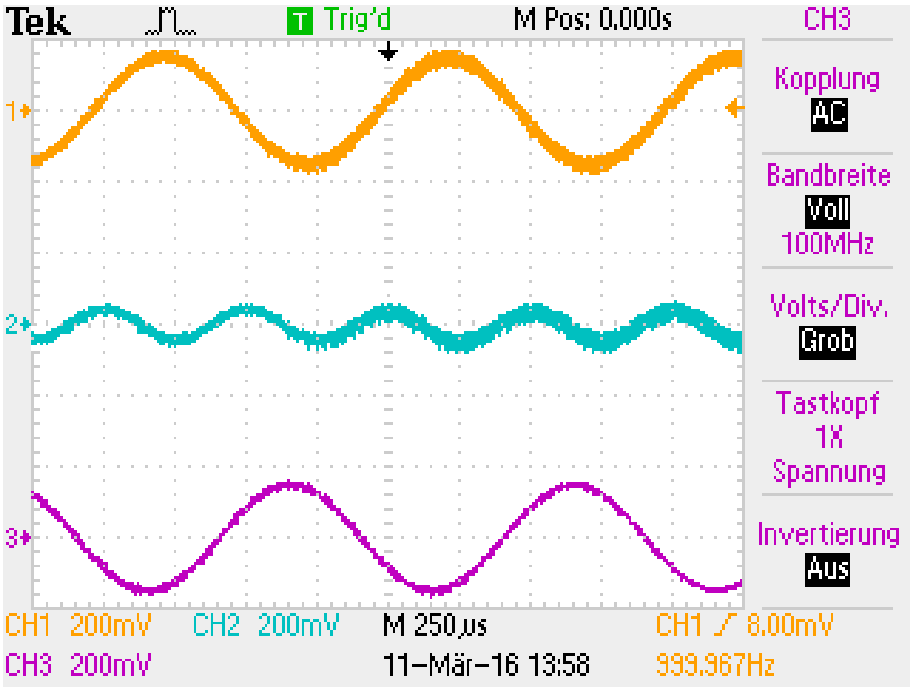


Abbildung 117: Cincheingang am Analog Switch durchgeschaltet

Das Signal am Klinkeneingang wird über den internen Shunt Switch gegen Masse geschaltet. Das Signal vom Cincheingang wird durchgeschaltet und kommt am Ausgang wieder heraus, siehe Abbildung 117. Danach wurde der Jumper auf „Klinke“ umgesteckt.

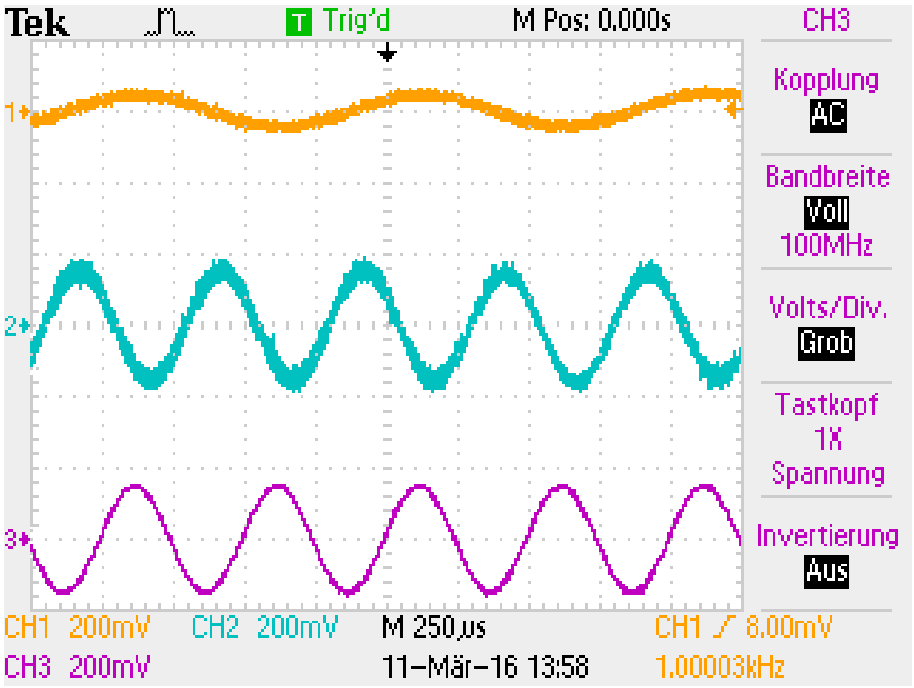
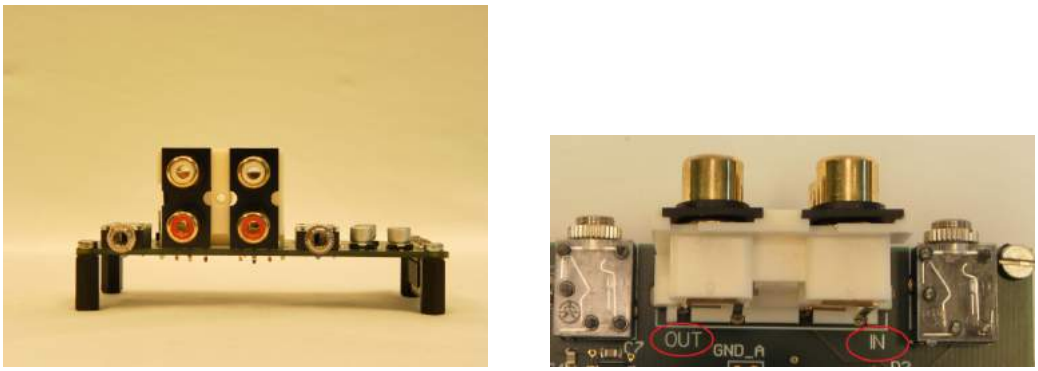


Abbildung 118: Klinkeneingang am Analog Switch durchgeschaltet

Nun wurde das Signal am Cincheingang über den internen Shunt Switch gegen Masse geschaltet. Das Signal vom Klinkeneingang wird durchgeschaltet und kommt am Ausgang wieder heraus, siehe Abbildung 118.

5.1.5 Ein- und Ausgänge

Es stehen zum Anschließen einer Audioquelle ein Cinch-Eingang und ein Klinken-Eingang zur Verfügung. Die Ausgabe erfolgt ebenfalls entweder über Cinch oder Klinke.



(a) Frontansicht (b) Draufsicht

Abbildung 119: Audioplatine Ein- und Ausgänge

Die Eingänge befinden sich in der Frontansicht (Abbildung 119a) auf der linken Seite und in der Draufsicht (Abbildung 119b) auf der rechten Seite. Sie sind auf der Platine zusätzlich mit „IN“ beschriftet. Bei den Cinchbuchsen ist die rote Buchse der rechte Kanal und die weiße Buchse der linke Kanal.

Die Ausgänge befinden sich in der Frontansicht (Abbildung 119a) auf der rechten Seite und in der Draufsicht (Abbildung 119b) auf der linken Seite. Sie sind auf der Platine zusätzlich mit „OUT“ beschriftet. Hier ist ebenfalls bei den Cinchbuchsen die rote Buchse der rechte Kanal und die weiße Buchse der linke Kanal.

5.1.6 Schnittstelle zur Basisplatine

Der Audioadapter wird über die Schnittstelle (Abbildung 120: J2) mit der Basisplatine verbunden. Die Spannungsversorgungen $+3,3$ V und $+5$ V werden von der Basisplatine zur Verfügungen gestellt, wobei nur die $+3,3$ V verwendet werden.

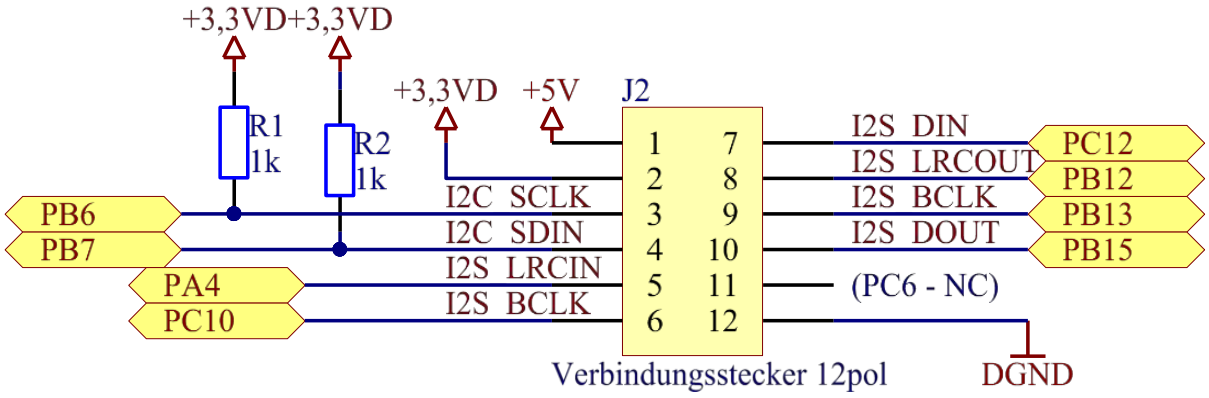


Abbildung 120: Schnittstelle zur Basisplatine

Pin	NAme	Beschreibung
1	+5 V	
2	+3, 3 VD	Versorgung für Audiocodec und Analog Switch
3	PB6 = I2C_SCLK	I^2C1, SCLK-Leitung; Taktleitung von I^2C1-Interface + Pull-Up
4	PB7 = I2C_SDIN	I^2C1, SDIN-Leitung; Datenleitung von I^2C1-Interface + Pull-Up
5	PA4 = I2S_LRCIN	I^2S3, LRCIN; Left-Right-Clock-IN-Leitung von I^2S3-Interface
6	PC10 = I2S_BCLK	I^2S3, BCLK; Bit-Clock-Leitung von I^2S3-Interface
7	PC12 = I2S_DIN	I^2S3, SIN; Serial-Data-IN-Leitung von I^2S3-Interface
8	PB12 = I2S_LRCOUT	I^2S2, LRCOUT; Left-Right-Clock-OUT-Leitung von I^2S2-Interface
9	PB13 = I2S_BCLK	I^2S2, BCLK; Bit-Clock-Leitung von I^2S2-Interface
10	PB15 = I2S_DOUT	I^2S2, DOUT; Serial-Data-OUT-Leitung von I^2S2-Interface
11	PC6	Not connected
12	DGND	Masse für den Audioadapter

Tabelle 20: Schnittstelle zur Basisplatine

☞ Hinweis: Die Bezeichnungen sind immer aus der Sicht des Audiocodecs (bzw. des Audioadapters) beschrieben. I2S_DIN sind also die Daten, die der Audiocodec **empfängt**.

5.1.7 Layout

In Abbildung 121 ist das fertige Layout und die Abmessungen der Platine dargestellt. Geroutet wurde nur am Toplayer (rot) und am Bottomlayer (blau). Die zwei inneren Layer der 4-fach Multilayer Platine sind eine Masse- und eine Versorgungsfläche, jeweils unterteilt in Digital- und Analogteil.

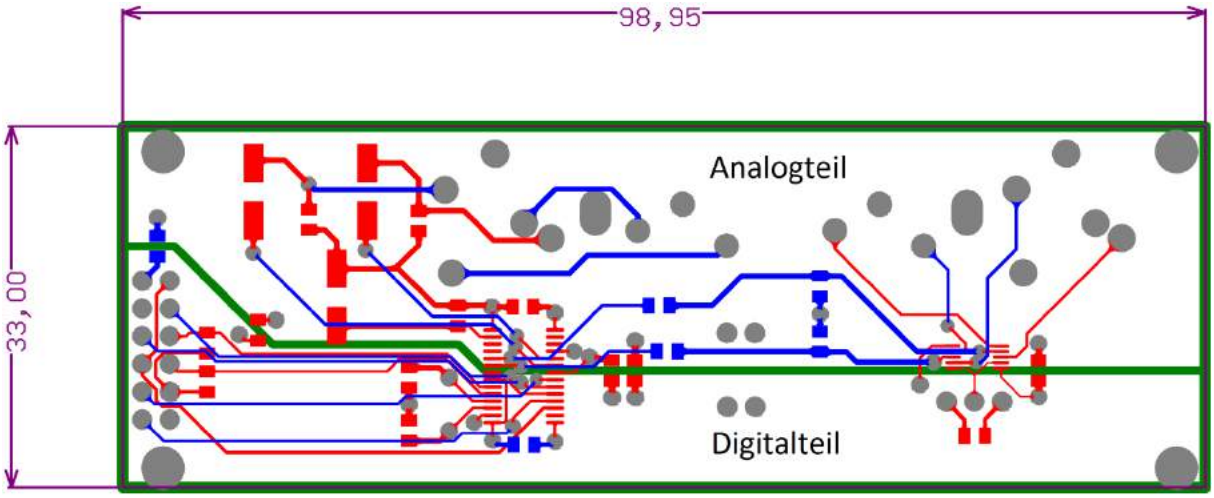


Abbildung 121: Fertig geroutetes Layout

5.1.8 Trennung von digitalen und anlaogen Signalen

Da der Audiocodec eine analoge und digitale Versorgung hat, sollten sowohl die Versorgungsspannung VCC von $+3,3$ V, als auch die Masse GND in einen digitalen und analogen Teil getrennt werden. Dadurch soll der Analogteil vom Digitalteil entkoppelt werden, um z.B. Störungen durch digitale Schaltvorgänge vom Analogteil fernzuhalten. Wichtig ist, dass der analoge Teil und der digitale Teil räumlich möglichst gut getrennt sind und die Massefläche bzw. die Versorgungsfläche möglichst durchgehend ist, was bei einer Multilayerplatine kein Problem darstellt. Außerdem sollten analoge Signale, also die Audiosignale, über dem analogen Teil geführt werden. Digitale Signale, wie z.B. verschiedene Takte und Steuersignale, sollten über dem digitalen Teil geführt werden.

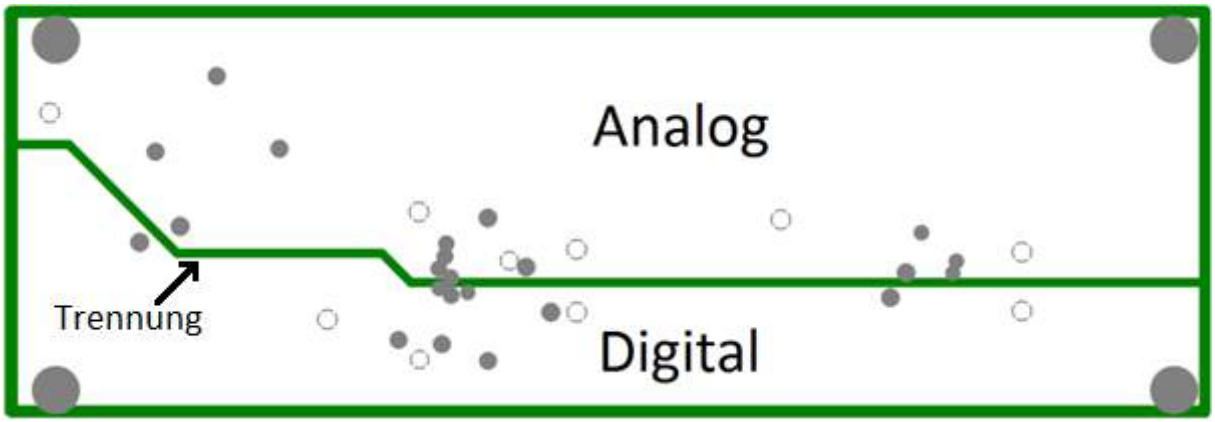


Abbildung 122: GND Layer unterteilt in Analog- und Digitalteil

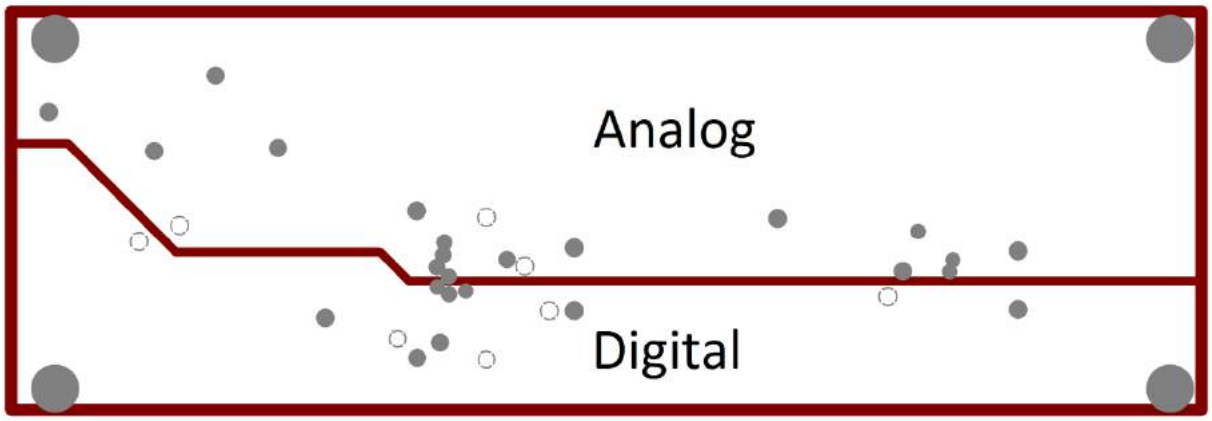


Abbildung 123: VCC Layer unterteilt in Analog- und Digitalteil

Die GND- und VCC-Kupferflächen sind durch eine Unterbrechung getrennt. Die Unterbrechung wird in Abbildung 122 von der grünen bzw. roten (Abbildung 123) Linie, die quer durch die Platine verläuft, dargestellt. Die Layer-Einstellungen können in Altium Designer 14.3 unter `Design` → `Layer Stack Manager` vorgenommen werden.

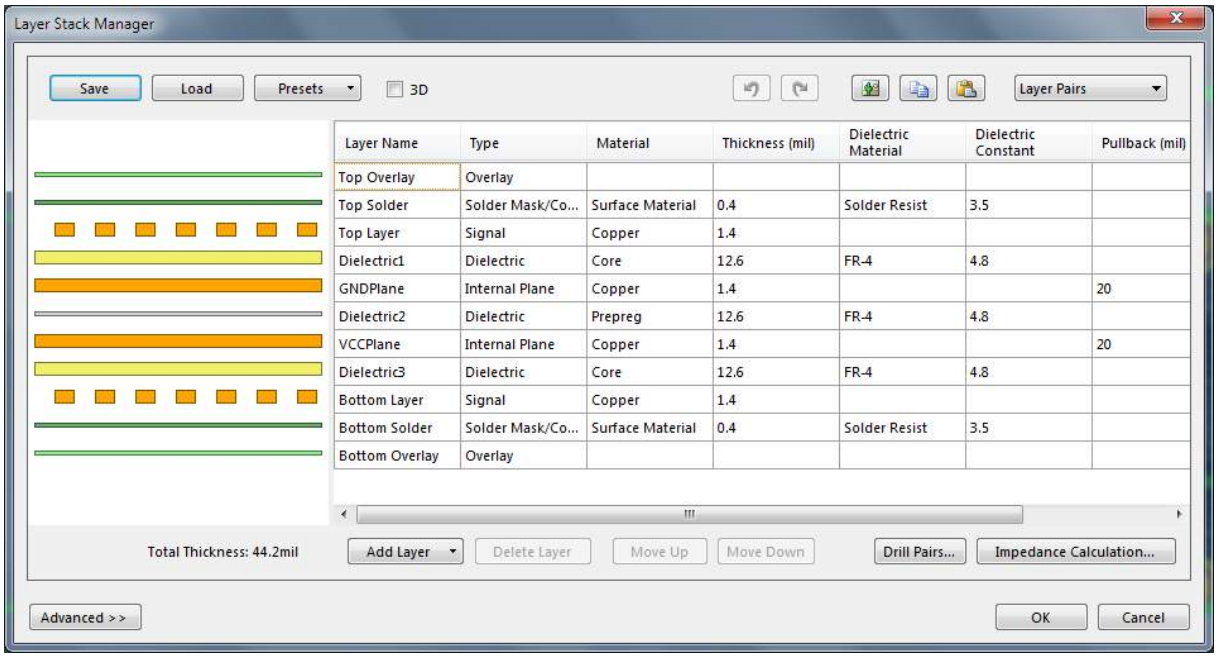


Abbildung 124: Altium: Layer Stack Manager

Im Layer Stack Manager (Abbildung 124) sieht man die zwei Signal-Layer (Top Layer und Bottom Layer) und die zwei internen Flächen bzw. Internal Planes (GND-Plane und VCC-Plane).

Die analoge und digitale Versorgungsspannung wird normalerweise durch eine Spule bzw. durch eine Ferritperle an einem Punkt verbunden. Auch die analoge und digitale Masse wird getrennt und an einem Punkt verbunden. Es wurden dafür mehrere Stellen vorgesehen. Die Verbindung erfolgt durch das Einlöten eines 0 Ω Widerstands. Eventuell führt das Verwenden einer Ferritperle zu noch besseren Ergebnissen. An welcher Stelle und mit welchem Bauteil (0 Ω Widerstand oder Ferritperle) die Trennung zwischen Analogteil und Digitalteil zum besten Ergebnis führt, müsste durch Messen getestet werden, was allerdings nicht geschah.

5.1.9 Bestückungsplan

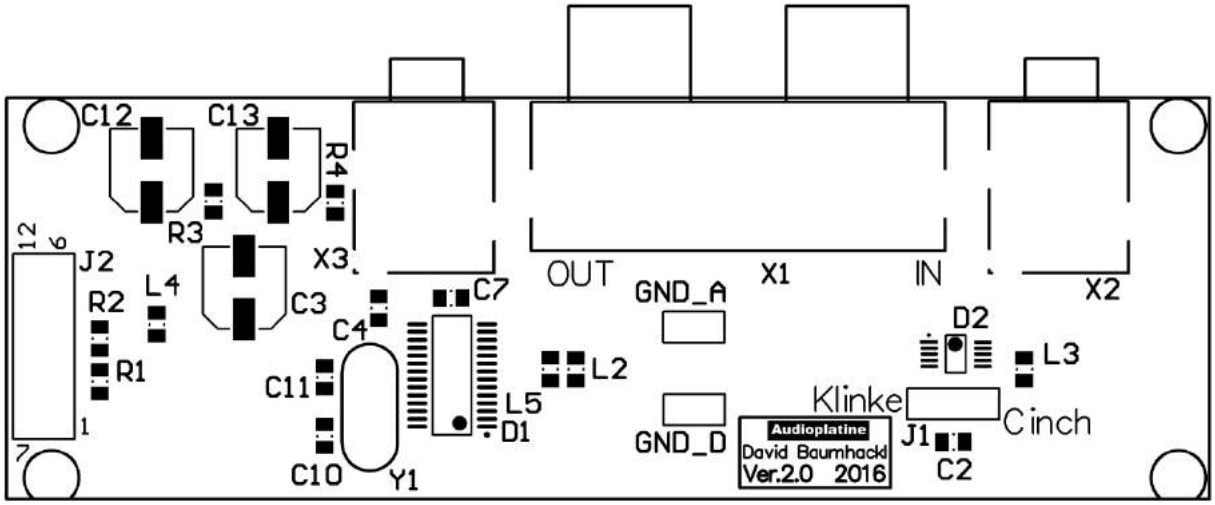


Abbildung 125: Bestückungsplan Top

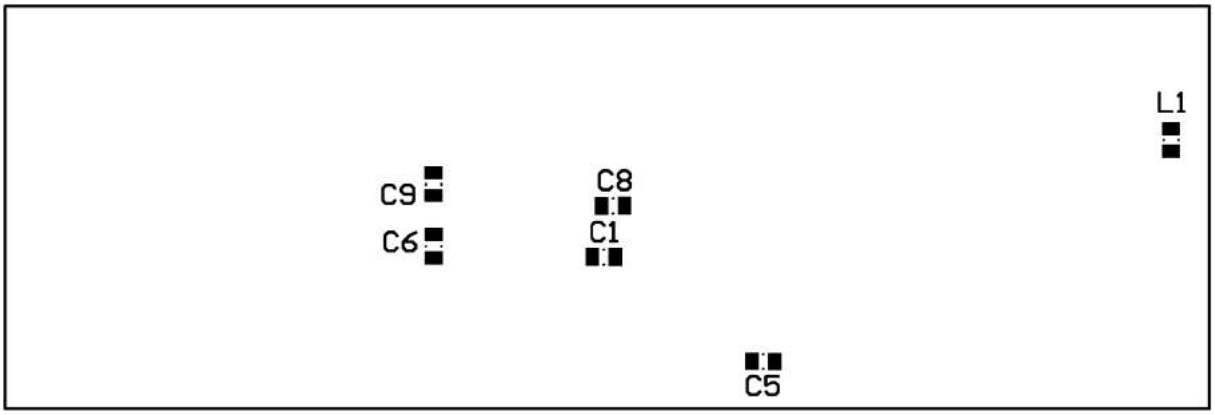


Abbildung 126: Bestückungsplan Bottom

5.1.10 Baugruppen

In Abbildung 127 ist eine Übersicht der wichtigsten Baugruppen des Audioadapters dargestellt und in Abbildung 127 der fertigbestückte Audioadapter.

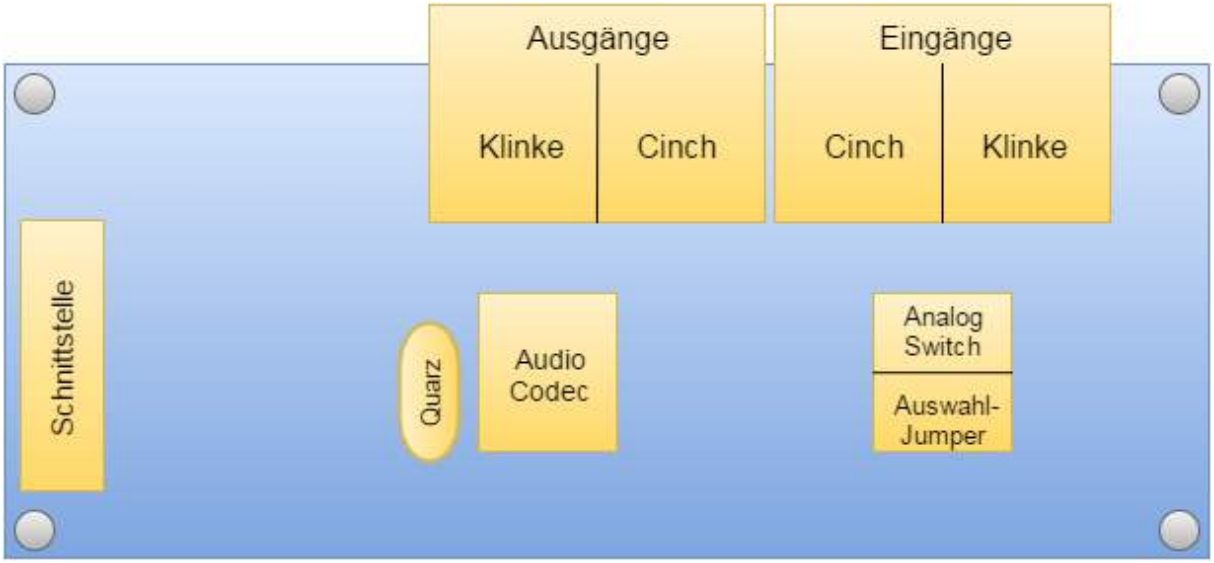


Abbildung 127: Übersicht Baugruppen

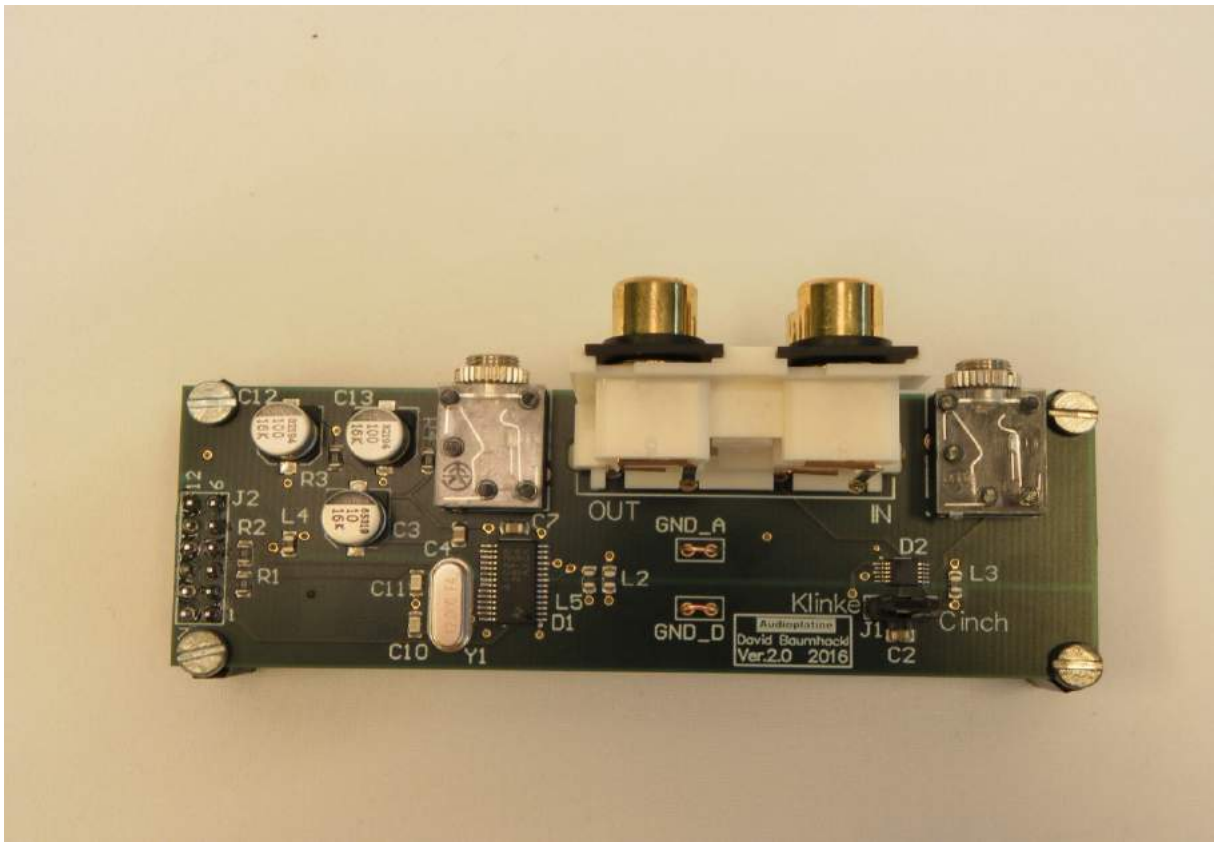


Abbildung 128: Fertige Audioplatine

5.1.11 Testen des Audioadapters mit dem FPGA Board Basys2

Der erste Test des Audioadapters wurde mit dem FPGA Board Basys2 durchgeführt. Da es für diesen Audiocodec bereits eine fertige VHDL Anwendung gibt, musste nur noch eine Adapterplatine gelötet und die Ausgänge im `.ucf` File geändert werden.

5.1.12 Basys2 Adapterplatine

Um eine kompakte Adapterplatine zu löten wurden die benötigen Ein-/Ausgänge im `top_audio_proc1.ucf` File vom VHDL Projekt „audiocodec.xise“ auf die ersten zwei Stecker (JA und JB) gelegt, siehe Listing 1.

Listing 1: User-Constraints für audiocodec.xise

```
1 NET "d_from_codec" LOC = "B2"; # Bank = 1, Signal name = JA1
2 NET "BCLK" LOC = "A3" ; # Bank = 1, Signal name = JA2
3 NET "lrc_from_codec" LOC = "J3" ; # Bank = 1, Signal name = JA3
4 NET "d_to_codec" LOC = "B5" | DRIVE = 2 ; # Bank = 1, Signal name = JA4
```

```
5
6  NET "PIO<76>" LOC = "C6" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JB1
7  NET "SCL" LOC = "B6" | DRIVE = 2 ;# | PULLUP ; # Bank = 1, Signal name = JB2
8  NET "SDA" LOC = "C5" | DRIVE = 2 ; # Bank = 1, Signal name = JB3
9  NET "lrc_to_codec" LOC = "B7" | DRIVE = 2 ; # Bank = 1, Signal name = JB4
```

Die Adapterplatine wurde nachfolgender Schaltung gelötet, wobei hier nur die zwei relevanten Stecker (JA und JB) gezeichnet wurden.

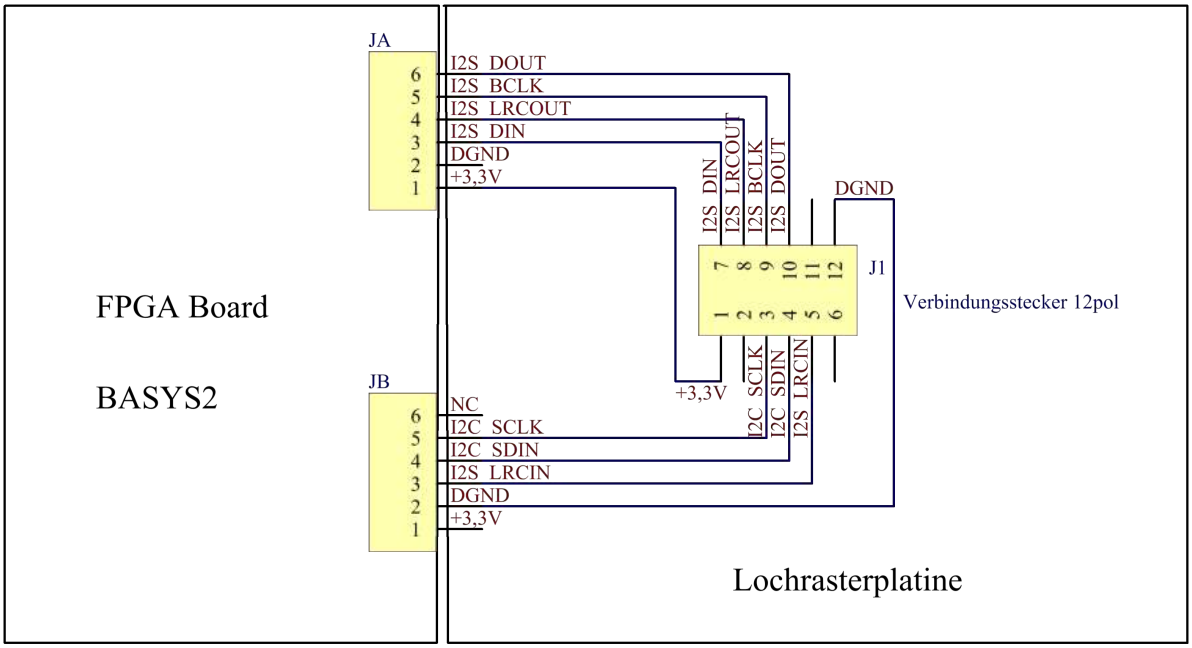


Abbildung 129: Schematic FPGA Adapterplatine

5.1.13 Testaufbau

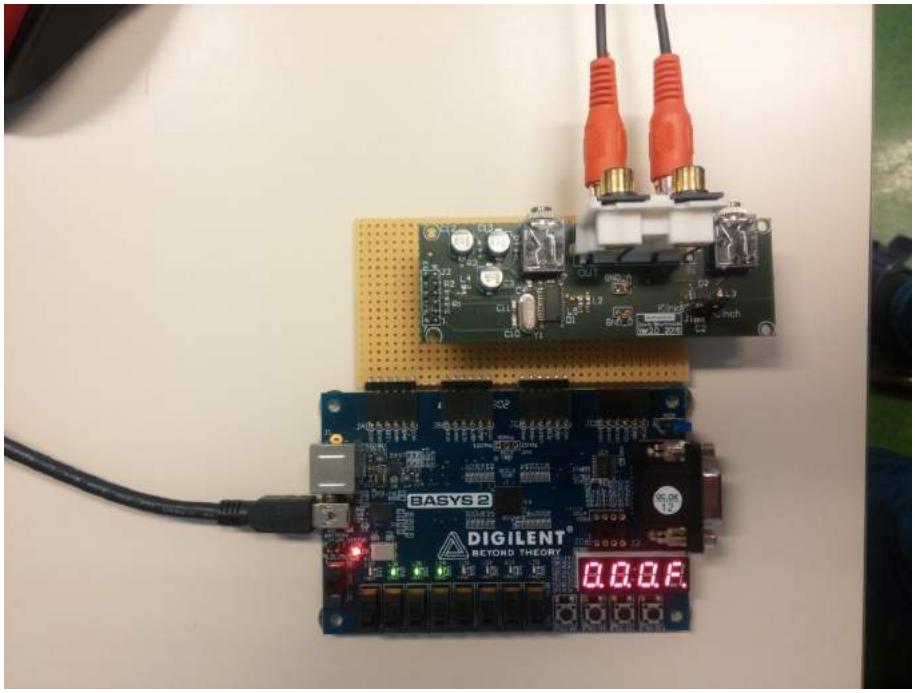


Abbildung 130: Testaufbau mit FPGA Board

Nach dem Einschalten des FPGA-Boards muss der Button `0` gedrückt werden, damit der Audiocodec initialisiert wird. Die Schalter `0` und `1` müssen für eine Verstärkung von 0 dB beide auf „low“ sein.

5.1.14 Messergebnisse

Zum Testen wurde an einem Kanal ein Sinus mit einer Frequenz von 1 kHz und einer Spannung von 500 mV_{pp} angelegt (Abbildung 131: gelbes Signal). Der Audiocodec wurde so konfiguriert, dass die Verstärkung 0 dB ist. Am Ausgang kommt wieder dasselbe Signal heraus (Abbildung 131: blaues Signal).

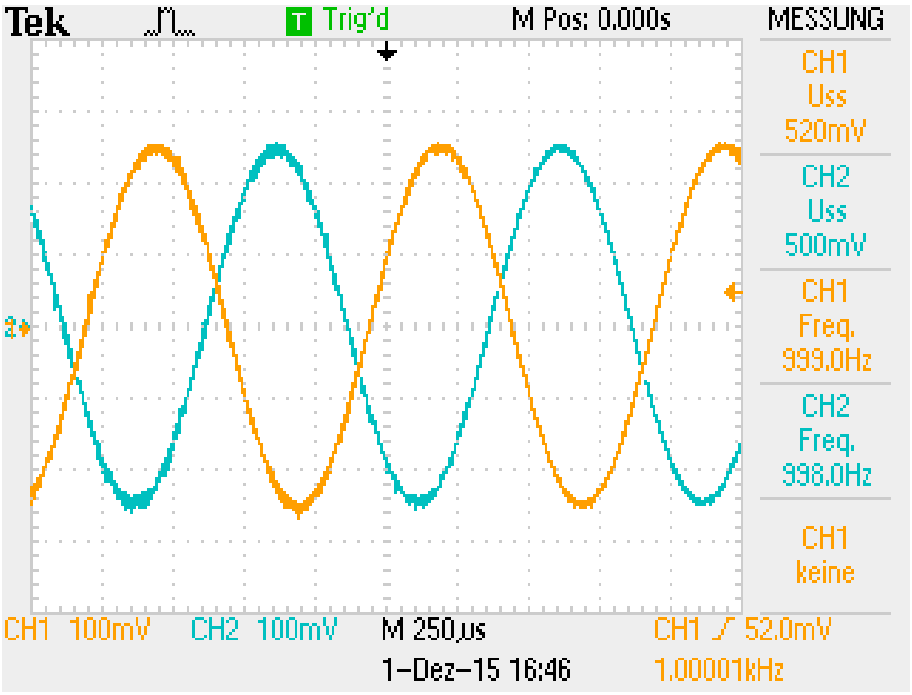


Abbildung 131: Messung des Ein-/Ausgangssignal

Danach wurde die Verstärkung des Audiocodecs getestet. Dazu wurde der Schalter `0` auf „high“ umgestellt. Beim erneuten Konfigurieren wurden die Register „left (& right) line input channel volume control“ auf eine Verstärkung von 6 dB eingestellt. Das Ausgangssignal war doppelt so groß wie das Eingangssignal, siehe Abbildung 132.

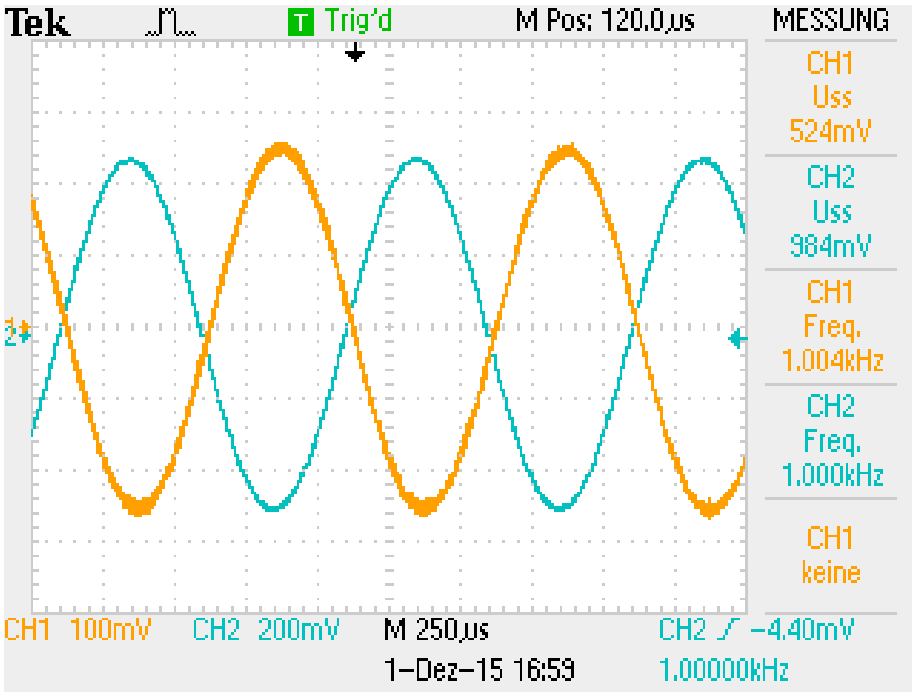


Abbildung 132: Messung der Verstärkung x2

Außerdem wurde noch ermittelt, wie weit der DAC vom Audiocodec aussteuern kann. Dazu wurde der Audiocodec so konfiguriert, dass die Verstärkung 18 dB, also den Faktor 8 beträgt. Das Eingangssignal wurde nun so lange erhöht, bis das Ausgangsignal abgeschnitten wurde. Dies war bei einem Eingangssignal von 450 mV_{pp} der Fall. Das Ausgangssignal wurde erst bei etwa $3,4$ V_{pp} abgeschnitten. Es kann also die gesamte Versorgungsspannung von $3,3$ V ausgenutzt werden, siehe Abbildung 133.

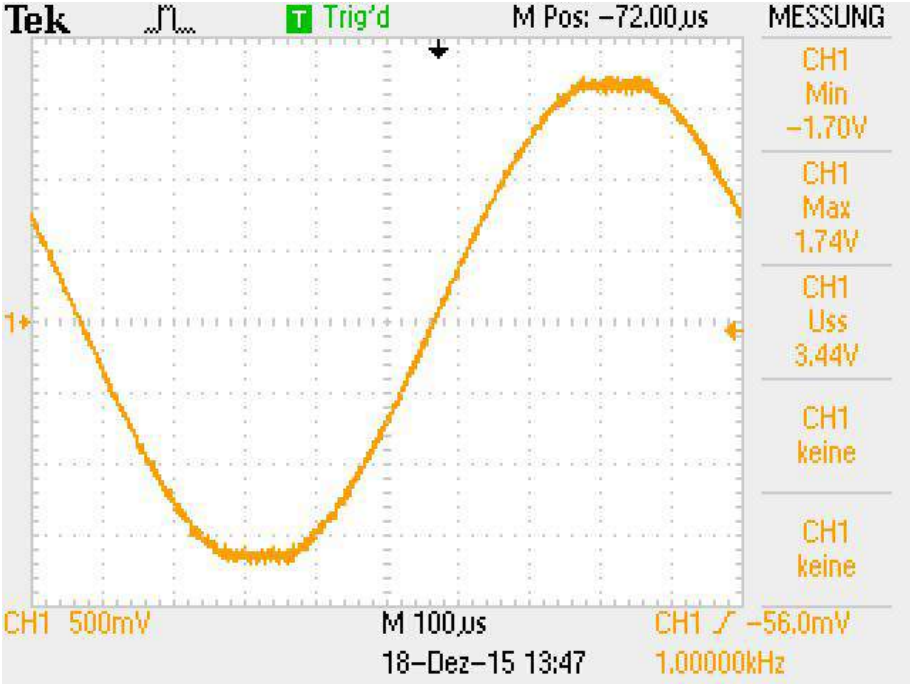


Abbildung 133: Abgeschnittenes Ausgangssignal

5.2 Testen der Audioadapterplatine am Minimalsystem

Beim Testen mit dem ARM Minimalsystem wurden sowohl die Cincheingänge (links und rechts), als auch der Klinkeneingang getestet. Als Eingang wurde ein Sinussignal mit 500 mV_{pp} mit einer Frequenz von 1 kHz eingespeist. Das Eingangssignal ist in Abbildung 134 auf Kanal 1 (gelb) zu sehen. Das Ausgangssignal ist auf Kanal 2 (blau) zu sehen. Die Register wurden so konfiguriert, dass die Verstärkung 0 dB (x1) beträgt. Abbildung 134 zeigt die Messung, bei dem das Signal beim linken Cincheingang angeschlossen wurde und wieder am linken Cinchausgang gemessen wurde.

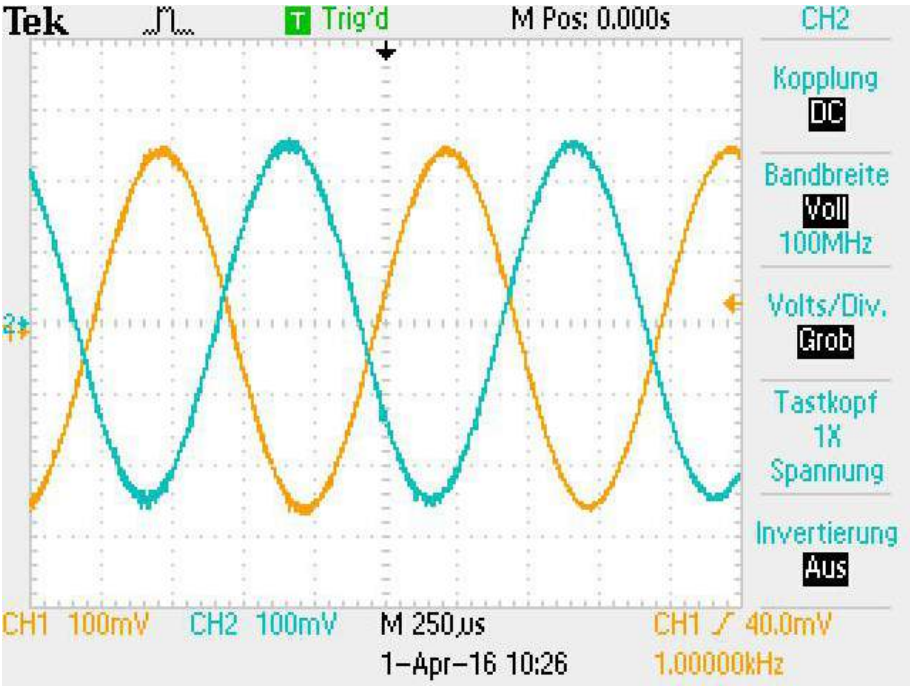


Abbildung 134: Messung mit Minimalsystem

Die Messungen, bei denen das Signal bei einem anderen Eingang angeschlossen wurde, ergaben dasselbe Ergebnis.

5.3 Messung der alten und neuen Audioplatine

5.3.1 Wichtige Eigenschaften von ADCs und DACs

Im Folgenden werden wichtige Eigenschaften von ADCs und DACs näher erläutert. Die Eigenschaften, die hier angeführt werden, wurden bei der „alten“ und bei der „neuen“ Audioplatine mit dem UPV-Audioanalyzer (Abschnitt 5.3.2) gemessen.

5.3.1.1 SNR - Signal to Noise Ratio

Das Signal zu Rausch Verhältnis beschreibt das Verhältnis zwischen der RMS Amplitude des Ausgangssignals zur RMS Amplitude des Rauschens. Die ersten 9 harmonischen

Schwingungen werden nicht in die Berechnung mit einbezogen.

$$SNR = 20 * log(\frac{Signalleistung}{Rauschleistung}) \tag{2}$$

Bei einer Signalamplitude von 1 V und einer durchschnittlichen Rauschamplitude von 1 mV würde das folgendes S/N Verhältnis ergeben:

$$SNR = 20 * log(\frac{1\text{ V}}{1\text{ mV}}) = 60\text{ dB} \tag{3}$$

Der ideale Signal zu Rausch Abstand lässt sich ungefähr mit folgender Formel berechnen, wobei n die Anzahl der Bit (Auflösung) angibt.

$$SNR_{ideal} = 6,02\text{ dB} * n - 1,76\text{ dB} \tag{4}$$

5.3.1.2 Klirrfaktor (THD - Total Harmonic Distortion)

Die Total Harmonic Distortion (= gesamte harmonische Verzerrung) bzw. der Klirrfaktor beschreibt das Verhältnis der summierten Leistungen der Oberschwingungen zur Leistung der Grundschwingung.

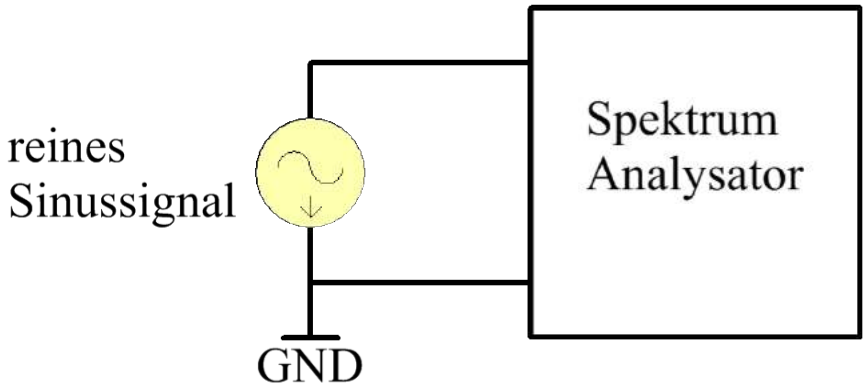


Abbildung 135: reines Sinussignal

Eine reine Sinusschwingung mit der Frequenz f1 ergibt folgendes das Spektrum, welches in Abbildung 136 zu sehen ist. Es ist nur die Frequenz f1 vorhanden und es gibt keine Oberschwingungen.

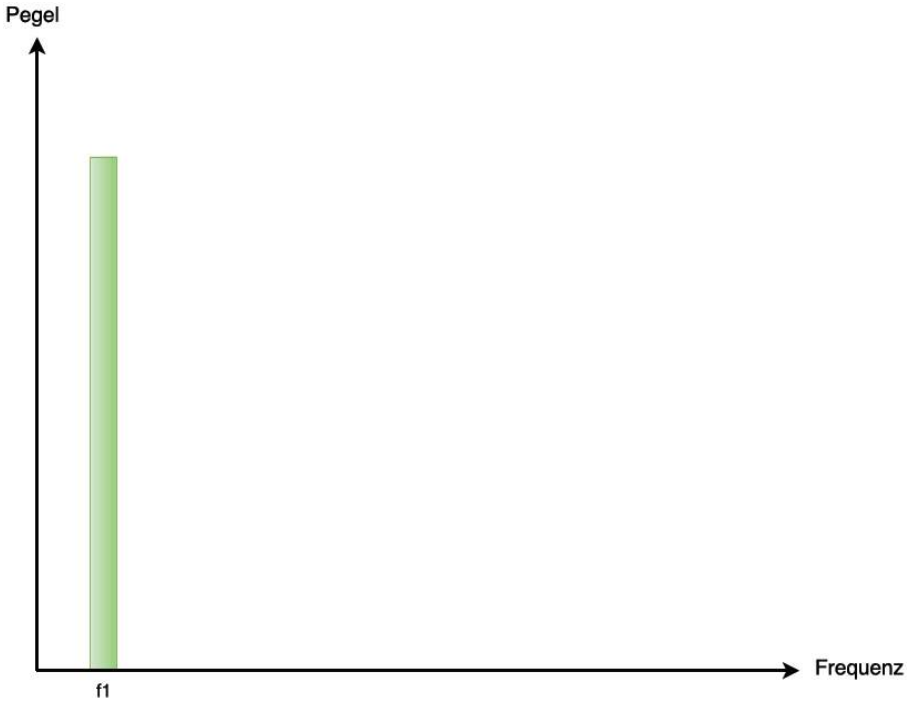


Abbildung 136: Spektrum ohne Harmonische

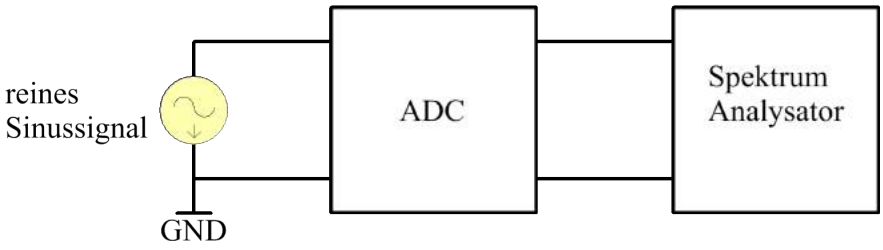


Abbildung 137: Analog-Digital gewandeltes Sinussignal

Wird das reine Sinussignal nun mit einem Analog-Digital Konverter umgewandelt (Abbildung 137) ergeben sich harmonische Schwingungen, die im Spektrum aussehen, wie in Abbildung 138 gezeigt.

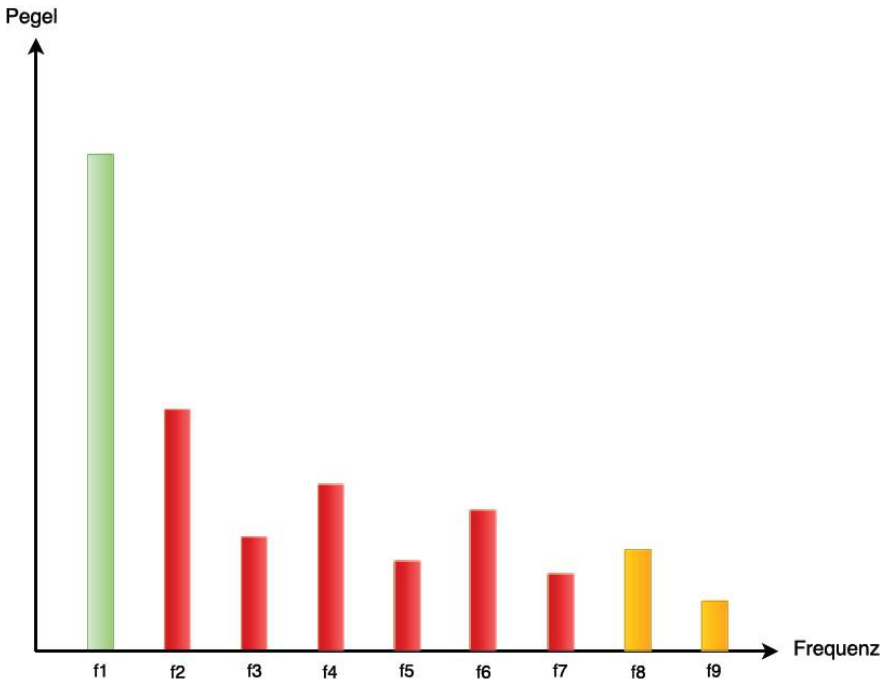


Abbildung 138: Spektrum mit Harmonischen

Wie im vorherigen Spektrum ist wieder das Grundsignal f1 (in grün) zu erkennen. Die weiteren Frequenzen f2 bis f9 (in rot und gelb) sind harmonische Schwingungen. Zur Berechnung der THD werden allerdings nur die ersten 6 harmonischen Schwingungen (in rot) benötigt.

Allgemein werden nichtlineare Verzerrungen in Prozent (Klirrfaktor k) oder in dB (Klirrdämpfung ak) angegeben.

$$THD = \frac{\sqrt{v^2(f_2) + v^2(f_3) + \cdots + v^2(f_7)}}{v(f_1)} \tag{5}$$

Das Umrechnen von dB in Prozent bzw. von Prozent in dB erfolgt mit folgenden Formeln:

$$k(\%) = 100 * 10^{\frac{a_k(dB)}{20}} \tag{6}$$

$$a_k(dB) = 20 * \log_{10}(\frac{k(\%)}{100}) \tag{7}$$

ak(dB) muss bei der Umrechnung als negativer Wert eingesetzt werden. Das Verhältnis zwischen Klirrfaktor und Klirrdämpfung wird in der folgenden Tabelle dargestellt.

Klirrfaktor [%]	**Klirrdämpfung [dB]**
100	0
10	-20
1	-40
0,1	-60

Tabelle 21: Verhältnis zwischen Klirrfaktor und Klirrdämpfung

5.3.1.3 SFDR - Spurious Free Dynamic Range

SFDR („störungsfreier dynamischer Bereich“) ist der Abstand der größten Störung zur Grundschwingung. Die größte Störung kann sowohl eine harmonische Schwingung (wie in der Abbildung dargestellt), als auch Rauschen sein. In Abbildung 139 sieht man den störungsfreien dynamischen Bereich im Spektrum eingezeichnet.

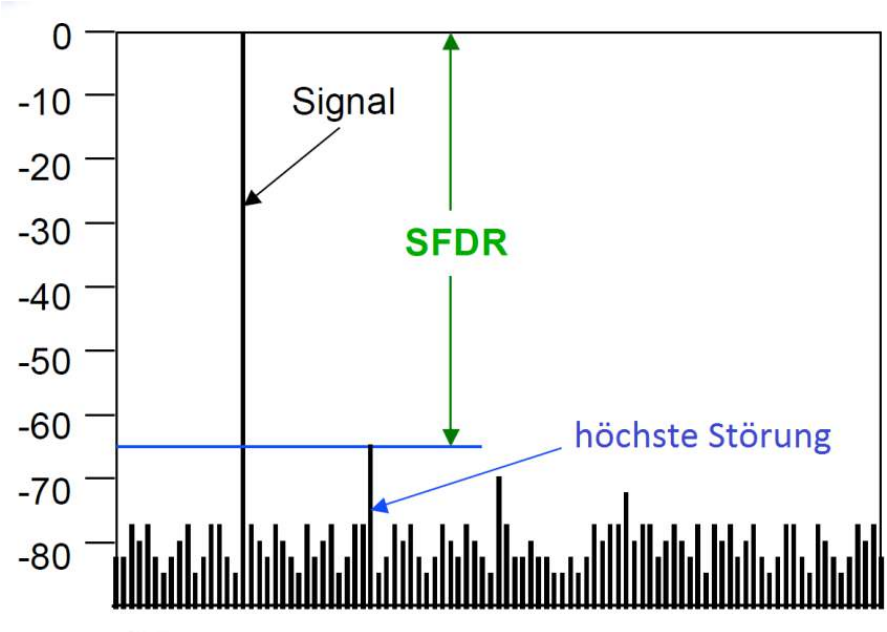


Abbildung 139: Spurious Free Dynamic Range

5.3.1.4 SINAD - Signal to Noise and Distortion

SINAD ist eine Kombination aus den beiden vorherigen Eigenschaften, SNR und THD. Es beschreibt das Verhältnis zwischen der RMS Amplitude des Ausgangssignals zur RMS Amplitude aller anderen Pegel (Rauschen + Harmonische), ausgenommen Gleichspannung. SINAD wird in dB angegeben und kann mit folgender Formel berechnet werden.

$$SINAD = 20 * \log \sqrt{10^{-\frac{SNR}{10}} + 10^{\frac{THD}{10}}} \tag{8}$$

5.3.1.5 ENOB - Effective Number of Bits

ENOB beschreibt die tatsächliche Auflösung eines ADCs bzw. DACs. Zum Beispiel kann ein ADC mit 16 Bit Auflösung durch Verzerrungen und Rauschen eine tatsächliche Auflösung von nur 12 Bit haben.

Die ENOB kann aus der zuvor berechneten SINAD berechnet werden:

$$ENOB = \frac{SINAD - 1,7 \text{ dB}}{6,02 \text{ dB}} \tag{9}$$

5.3.2 Messgerät

Um die Qualität des alten und neuen Audioadapters zu vergleichen, wurden verschiedene Eigenschaften gemessen. Die Messungen wurden mit einem UPV Audioanalyzer von Rohde & Schwarz durchgeführt.

Abbildung 140: UPV Audio Analyzer von Rhode & Schwarz

Mit diesem Messgerät sind zahlreiche Audiomessungen möglich. Der Startbildschirm ist in Abbildung 141 dargestellt.

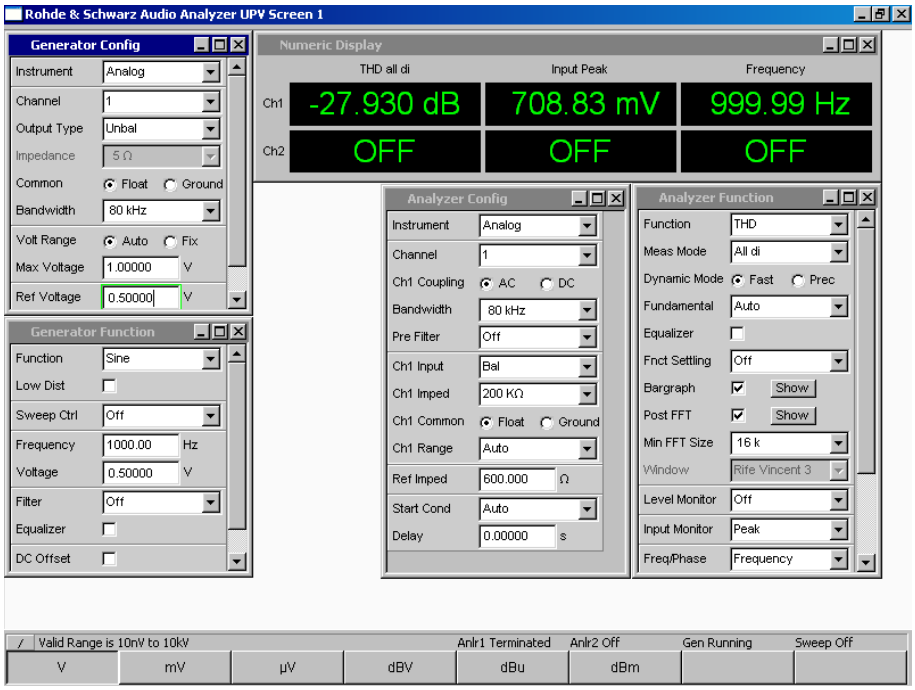


Abbildung 141: UPV Audio Analyzer Startbildschirm

In den Fenstern „Generator Config“ und „Generator Function“ können die Einstellungen für das Signal, das beim Audioadapter eingespeist wird, eingestellt werden. Dazu gehören unter anderem die Spannung, die Frequenz, die Signalform und Funktionen wie Frequenzsweep. In den Fenstern „Analyzer Config“ und „Analyzer Function“ wird eingestellt, was

gemessen werden soll und wie gemessen wird. Unter den Funktionen kann z.B. THD, SNR, FFT, SINAD und vieles mehr eingestellt werden. Außerdem können diverse Filter, wie z.B. das A-Weighting Filter, das bei der SNR Messung benötigt wurde, eingestellt werden. Im „Numeric Display“ werden die Generatorspannung und das Messergebnis angezeigt. Die Messergebnisse können auch grafisch dargestellt werden, wie bei den folgenden Messergebnissen zu sehen ist.

5.3.3 Messergebnisse des alten und neuen Audioadapters

In der Tabelle ist der Klirrfaktor des alten und des Audioadapters bei verschiedenen Eingangsspannungen vergleichbar. Die Messungen wurden bei einer Frequenz von 1 kHz durchgeführt. Die Eingangsspannung wurde bei dieser und allen folgenden Messungen von 200 mV_{RMS} bis 900 mV_{RMS} in 100 mV-Schritten erhöht.

	alter Audioadapter		neuer Audioadapter	
U_{gen} [mV_{RMS}]	THD [dB]	THD [%]	THD [dB]	THD [%]
200	-29,4	9,687	-77,7	0,013
300	-28,8	10,160	-77,9	0,013
400	-29,5	9,611	-76,8	0,014
500	-29,4	9,687	-77,0	0,014
600	-28,4	10,488	-79,5	0,011
700	-28,3	10,572	-81,4	0,009
800	-28,4	10,488	-83,4	0,007
900	-27,8	11,000	-83,9	0,006

Tabelle 22: Messergebnisse Klirrfaktor

Normalerweise erhöht sich der Abstand zwischen der Grundschwingung und den harmonischen Schwingungen mit zunehmender Eingangsspannung (solange die Klippgrenze nicht erreicht bzw. überschritten wird). Bei den Ergebnissen des alten Audioadapters ist dies nicht der Fall. Der Klirrfaktor bleibt über den gesamten Eingangsspannungsbereich ähnlich. Abbildung 142 ist der Klirrfaktor des alten Audioadapters bei einer Eingangsspannung von 900 mV_{RMS} mit einer Frequenz von 1 kHz grafisch dargestellt. Außerdem wird mit dem Marker der Abstand zur höchsten harmonischen Schwingung, also SFDR, markiert. In Abbildung 143 wird der Klirrfaktor des neuen Audioadapters bei der gleichen Eingangsspannung grafisch dargestellt. Außerdem wird wieder SFDR mit dem Marker angezeigt. Der Vergleich zeigt einen deutlich höheren Klirrfaktor des „alten“ Audioadapters, verglichen mit dem „neuen“ Audioadapter. Aus der vorherigen Messtabelle sind folgende Werte für THD bei einer Eingangsspannung von 900 mV_{RMS} zu entnehmen:

U_{gen} [mV_{RMS}]	THD alt [dB]	THD neu [dB]
900	-27,8	-83,9

Tabelle 23: Messergebnisse Klirrfaktor bei 900 mV_{RMS}

Die höchste Störung liegt beim „alten“ Audioadapter bei 2 kHz nur 28, 243 dB unter der Grundschwingung, während beim „neuen“ Audioadapter die größte Störung bei 6 kHz 86, 288 dB unter der Grundschwingung liegt.

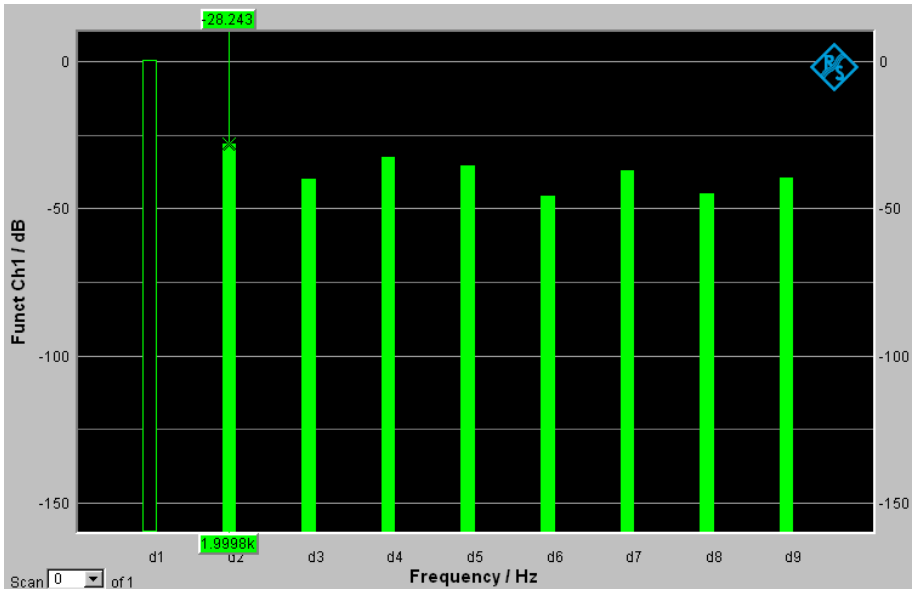


Abbildung 142: THD & SFDR des alten Audioadapters (U_{ein} = 900 mV)

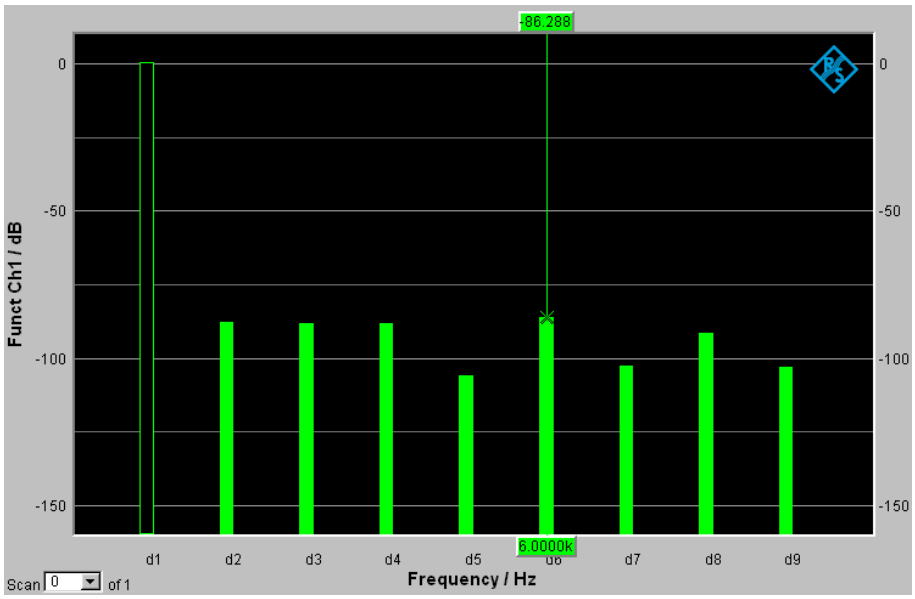


Abbildung 143: THD & SFDR des neuen Audioadapters (U_{ein} = 900 mV)

5.3.3.1 SNR

In der Tabelle ist der Signal-zu-Rausch-Abstand des alten und neuen Audioadapters bei verschiedenen Eingangsspannungen gegenübergestellt. Die Frequenz, mit der gemessen wurde, beträgt 1 kHz. Außerdem wurde ein A-Weighting Filter verwendet, der beim UPV Audioanalyzer im Fenster „Analyzer Function“ eingestellt wurde.

	alter Audioadapter	neuer Audioadapter
U_{gen} [mV_{RMS}]	SNR [dB]	SNR [dB]
200	71,3	69,4
300	74,7	69,4
400	76,9	75,7
500	78,6	77,4
600	80,1	79,2
700	81,4	80,1
800	82,7	81,7
900	84,0	82,5

Tabelle 24: Messergebnisse Signal-Rausch-Abstand

Bei den Ergebnissen ist zu erkennen, dass der Signal-zu-Rausch-Abstand bei der alten Audioplatine minimal besser ist. Mit der Analyzer Funktion FFT ist in den folgenden

zwei Abbildungen das Spektrum dargestellt. Im Spektrum sind sowohl die harmonischen Schwingungen, als auch das Rauschen zu sehen.

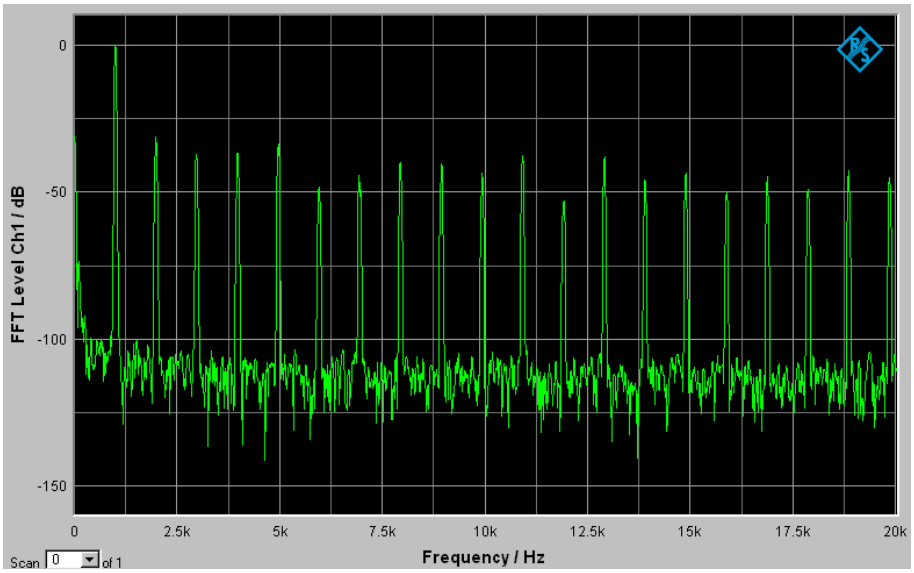


Abbildung 144: FFT des alten Audioadapters (U_{ein} = 900 mV)

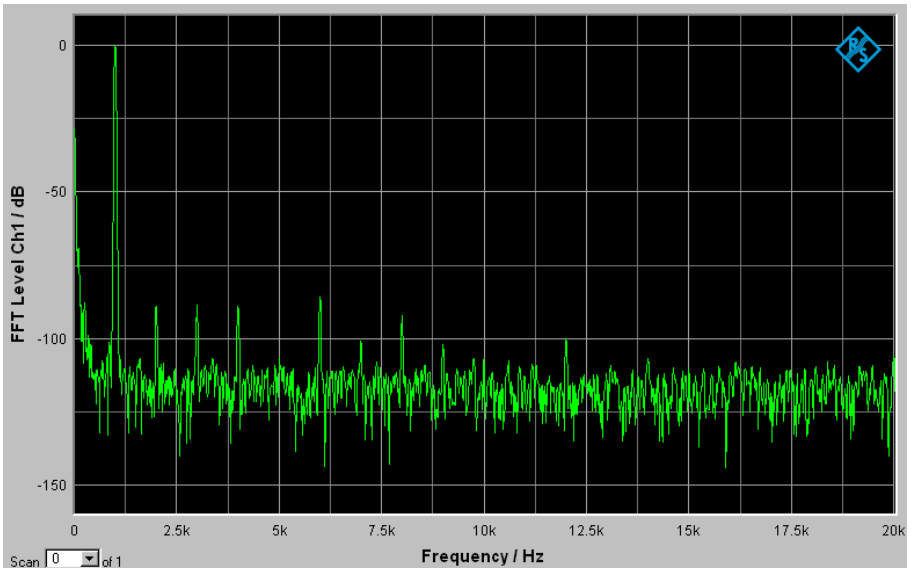


Abbildung 145: FFT des neuen Audioadapters (U_{ein} = 900 mV)

5.3.3.2 SINAD & ENOB

	alter Audioadapter			neuer Audioadapter		
U_{gen} [mV_{RMS}]	SINAD [dB] gemessen	SINAD [dB] berechnet	ENOB berechnet	SINAD [dB] gemessen	SINAD [dB] berechnet	ENOB berechnet
200	22,2	29,4	3	68,7	68,80	11
300	25,8	28,8	4	69,8	68,83	11
400	22,3	29,5	3	70,1	73,20	11
500	22,9	29,4	3	72,3	74,19	11
600	22,3	28,4	3	74,5	76,34	12
700	22,9	28,3	3	76,0	77,69	12
800	22,9	28,4	3	77,0	79,46	12
900	23,8	27,8	3	77,8	80,13	12

Tabelle 25: Messergebnisse SINAD & ENOB

In der Tabelle ist SINAD gemessen und berechnet zu sehen. Auf Grund der hohen harmonischen Störungen beim alten Audioadapter ergibt sich für SINAD ein schlechter Wert. Außerdem wurde aus SINAD die effektive Bitanzahl (ENOB) berechnet. Dieser Wert muss abgerundet werden (z.B. 12,5 Bit → 12 Bit). Dies würde beim alten Audioadapter theoretisch ein ENOB von 3 bis 4 ergeben, da sich die harmonischen Störungen stark auf die Berechnung auswirken. Dieser Wert ist allerdings nicht realistisch, da man diese geringe Auflösung beim Signal im Zeitbereich wahrnehmen würde (stufenförmiges Signal). Die ENOB des neuen Audioadapters ergibt einen Wert von 11 bzw. 12 ab einer Eingangsspannung von 600 mV_{RMS}. Dieser Wert ist durchaus realistisch.

6 Stücklisten

6.1 Core-Modul

Stück/ Board	Wert	BMKZ	Beschreibung
Widerstände			
1	22k	R1	SMD 0805
2	1k	R5, R6, R10	SMD 0805
1	1M	R8	SMD 0805
2	560R	R7, R11	SMD 0805
2	10k	R4, R9	SMD 0805
Kondensatoren			
2	10p	C2, C4	SMD 0805, KERKO
2	22p	C1, C3	SMD 0805, KERKO
8	100n	C5, C6, C7, C8, C10, C11, C13, C14	SMD 0805, KERKO
2	10u	C12, C9	Bauform A, TANTAL
Dioden			
3	Schottky Diode	V1, V3, V4	SOD-323, 1A
1	LED	V6	SMD 0805, rot
1	LED	V5	SMD 0805, blau
1	LED	V2	SMD 0805, grün
ICs			
1	-	U1	STM32F107RBT6
Ferritkern			
1	Ferrit	L1	SMD 0805
Schalter / Taster			
4	-	X	Jumper
2	-	S1, S2	Printtaster
Stift- und Buchsenleisten / Stecker, Buchsen			
1	Buchsenleiste-Buchse	X1	SFH11, ST-Link V2
1	2x3pol. Stiftleiste	X5	2x3, 2.54mm, Stiftleiste
2	4pol. Buchsenleiste	X3, X4	1x4, 2.54mm, Buchsenleiste
2	25pol. Buchsenleiste	X2	1x25, 2.54mm, Buchsenleiste
2	25pol. Stiftleiste	X2	1x25, 2.54mm, Stiftleiste
3	2pol. Stiftleiste	X6, X7, X8	1x2, 2.54mm, Stiftleiste
Spannungswandler			
1	LMS8117ADT_3.3	U2	TO252
Quarz			
1	25MHz	Y1	Schwingquarz
1	32kHz	Y2	Schwingquarz

Tabelle 26: Stückliste Core-Modul

6.2 Basisplatine

Stück/ Board	Werte	BMKZ	Beschreibung
Widerstände			
2	22R	R1, R2	SMD 0805
7	10k	R5, R10, R11, R12, R15, R16, R34, R35	SMD 0805
1	47k	R33	SMD 0805
10	560R	R20, R21, R22, R23, R24 R25, R26, R27, R32, R36	SMD 0805
1	62k	R42	SMD 0805
1	5k6	R6	SMD 0805
1	1k5	R41	SMD 0805
1	4k7	R8	SMD 0805
3	1k	R3, R7, R17	SMD 0805
1	100R	R43	SMD 0805
1	10R	R18	SMD 0805
4	0R	R37, R38, R39, R40	SMD 0603
3	NC	R44, R45, R46	SMD 0805
2	22k	R19, R28	SMD 0805
4	2k2	R29, R30, R47, R48	SMD 0805
Potentiometer			
5	10k	R4, R9, R13, R14, R31	Durchsteckmontage
Kondensatoren			
22	100n	C1, C6, C7, C8, C12, C14, C15, C16, C17, C18, C37, C38, C39, C40, C41, C42, C43, C44, C45, C46, C47, C48	SMD 0805, KERKO
3	47n	C49, C50, C51	SMD 0805, KERKO
1	470n	C54	SMD 0805, KERKO
3	10n	C9, C10, C11	SMD 0805, KERKO
20	NC	C5, C20, C21, C22, C23, C24, C25, C26,C27, C28, C29, C30, C31, C32, C33, C34	SMD 0603
2	1u	C35, C36	Bauform A, TANTAL
3	10u	C3, C4, C19	ELKO, SMD Metal
1	100u	C52	ELKO, SMD Metal
3	220u	C2, C13, C53	ELKO, SMD Metal
Dioden			
1	B40C800	U1	Diodengleichrichter
5	BAT60A	V10, V11, V12, V16, V17	SOD-323, 3A
12	WS2812	D6, D7, D8, D9, D10, D11, D12 D13, D14, D15, D16, D17	SMD RGB-LED, mit Controller
1	LTRB-GFSF	D5	SMD RGB-LED
1	LED	V9	SMD 0805, rot
1	NC	V12	SOD-323, 3A
1	LED	V13	SMD 0805, blau
1	1N4148	V15	DO-204AH
9	LED	V1, V2, V3, V4, V5, V6, V7, V8, V14	SMD 0805, grün

Tabelle 27: Stückliste Basisplatine

Stück/ Board	Werte	BMKZ	Details
ICs			
1	24AA256	D1	EEPROM
1	NCP5623	D4	I^2C LED-Treiber
1	STMPS2141	D3	Power Switch
1	USBLC6-2	F1	ESD-Protection
1	NE555	D2	DIP-8
1	MAX232	U2	DIP-16
Schalter / Taster			
23	-	X	Jumper
1	DIP-Switch	S2	DIP-16, 8-Schalter
1	Inkrementalgeber	S1	Durchsteckmontage
Stift- und Buchsenleisten / Stecker, Buchsen			
12	2pol. Stiftleiste	X4, X5, X8, X14, X15, X18, X22, X29, X40, X41, X42, X43	1x2, 2.54mm, Stiftleiste
1	-	X1	USB-B Buchse
1	-	X3	USB-A Buchse
1	2x6pol. Stiftleiste	X9	2x6, 2.54mm, Stiftleiste
3	2x3pol. Stiftleiste	X2, X17, X33	2x3, 2.54mm, Stiftleiste
4	2x2pol. Stiftleiste	X10, X11, X13, X37	2x2, 2.54mm, Stiftleiste
2	3pol. Stiftleiste	X27, X39	1x3, 2.54mm, Stiftleiste
1	SUB-D9	X16	Stecker mit Lötkelch, Female
9	4pol. Buchsenleiste	X6, X7, X23, X26, X28, X31, X32, X34, X36	1x4 2.54mm, Buchsenleiste
2	25pol. Buchsenleiste	X20	1x25 2.54mm, Buchsenleiste
2	8pol. Buchsenleiste	X33	1x8 2.54mm, Buchsenleiste
1	10pol. Buchsenleiste	X33	1x10 2.54mm, Buchsenleiste
2	5pol. Buchsenleiste	X12	1x5 2.54mm, Buchsenleiste
2	6pol. Buchsenleiste	X30, X33	1x6 2.54mm, Buchsenleiste
1	2x6pol. Buchsenleiste	X24	2x6 2.54mm, Buchsenleiste
1	2x4pol. Buchsenleiste	X35	2x4 2.54mm, Buchsenleiste
1	Buchsenleiste-Buchse	X21	SFH11, ST-Link V2
1	Stiftleisten-Buchse	X19	JTAG
1	Buchsenleiste-Buchse	X25	Header, XH2.54-4P, 90°
2	IC-Sockel	D1, D2	DIP-8
2	IC-Sockel	S1, U2	DIP-16
1	DC-Buchse	J1	2,1mm Stift, Powerjack
2	10pol. Buchsenleiste	A2	XBee-Pro, 1x10, 2mm, Buchsenleiste
Spannungswandler			
1	171050601	U3	TO263-7EP
Sensoren			
1	IR-Receiver	B2	Durchsteckmontage
1	DS18S20	B1	TO-92, Temperatursensor
1	ESP8266	X35	WLAN-Modul
1	Piezo-Summer	B4	Durchsteckmontage
1	TSL235	B3	LFU
1	-	X	Nextion Display
1	-	X	BMA020
1	-	X	HC-06
1	-	X	Xbee-Pro
1	-	X	HC-12

Tabelle 27: Stückliste Basisplatine

Stück/ Board	Werte	BMKZ	Details
Zubehör			
1	-	X	ST-Link V2
4	-	X	JST XHP Buchseneinsatz XH
1	-	X	JST XHP Buchsengehäuse
5	-	X	Potentiometer-Eintellhilfe
38	-	X	Distansbolzen, M3
38	-	X	Halterungsschrauben, M3
Sicherung			
1	-	F2	Polyfuse, 500mA

Tabelle 27: Stückliste Basisplatine

6.3 USB-to-UART Adapter

Stück/ Board	Wert	BMKZ	Beschreibung
Widerstände			
3	1k	R1, R2, R4	SMD 0805
Kondensatoren			
3	100n	C1, C2, C5	SMD 0805, KERKO
1	10n	C4	SMD 0805, KERKO
1	4u7	C3	Bauform A, TANTAL
Dioden			
1	LED	V1	SMD 0805, rot
1	LED	V2	SMD 0805, grün
1	LED	V4	SMD 0805, blau
ICs			
1	FT232RL	D1	SSOP-28
Ferritkern			
1	Ferrit	L1	SMD 0805
Stift- und Buchsenleisten / Stecker, Buchsen			
2	4pol. Stiftleiste	X3, X2	1x4, 2.54mm
1	3pol. Stiftleiste	X4	1x3, 2.54mm
1	-	X1	USB-B Buchse

Tabelle 28: Stückliste USB-to-UART Adapter

6.4 Audioadapter

Stück/ Board	Wert	BMKZ	Beschreibung
Widerstände			
4	100R	R3, R4, R7, R12	SMD 0805
2	4k7	R13, R14	SMD 0805
2	2k2	R1, R2	SMD 0805
4	100k	R5, R6, R8, R11	SMD 0805
2	51R	R9, R10	SMD 0805
Kondensatoren			
3	100n	C2, C8, C12	SMD 0805
4	1n8	C16, C17, C19, C20	SMD 0805
11	1u	C1, C3 ,C4, C5, C6, C7, C9, C10, C13, C18, C23	SMD 0805
2	22n	C21, C22	SMD 0805
3	10u	C11, C14, C15	SMD 0805
ICs			
1	PCM1870	D1	RHF Package
1	TLV320DAC23	D2	PW PACKAGE
Spulen			
5	10uH	L1, L2, L3,L4, L5	SMD 0805
Stift- und Buchsenleisten / Stecker, Buchsen			
1	1x3pol. Stiftleiste	J1	1x3, 2.54mm, Stiftleiste
2	1x2pol. Stiftleiste	GND_A, GND_D	1x2, 2.54mm, Stiftleiste
1	2x6pol. Stiftleiste	J2	2x6, 2.54mm, Stiftleiste
2	Klinkenbuchse 3.5mm	X2, X3	Klinkenbuchse 3.5mm
1	Cinch-Buchse	X1	Cinch-Buchse

Tabelle 29: Stückliste Audioadapter

7 Software

Im Zuge dieser Diplomarbeit entstanden zwei größere Softwareprojekte für das ARM Cortex-M3 Minimalsystem.

7.1 ODDDragon

Für die Tage der offenen Tür der HTBL Hollabrunn im Jahr 2017/18 entstand ein Testprogramm, welches die neuen Features des Minimalsystems demonstrieren sollte. Hierzu wurde ein einfaches GUI für das NEXTION-Display programmiert, welche die X, Y und Z Werte der Beschleunigung vom über SPI angesteuerten Gyroskop ausliest, und in Form eines Graphen anzeigt. Des weiteren wurde eine – nicht 100% funktinierenden – Funktion zum Speichern des Graphen (auf dem verbauten EEPROM) programmiert.

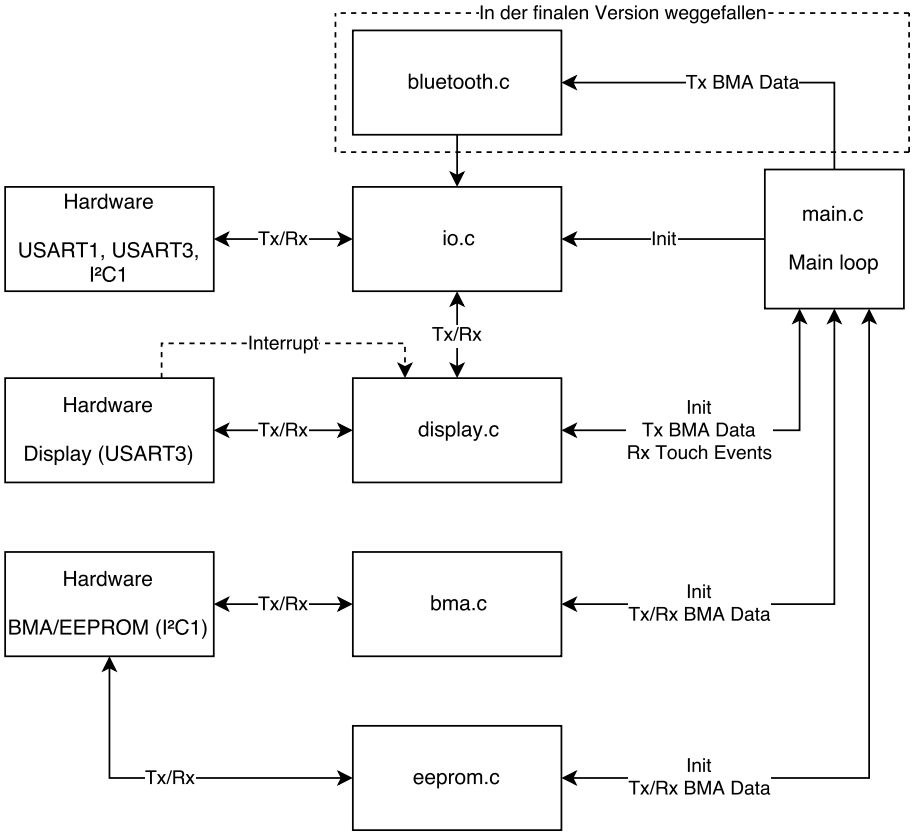


Abbildung 146: Tag der offenen Tür: Blockschaltbild

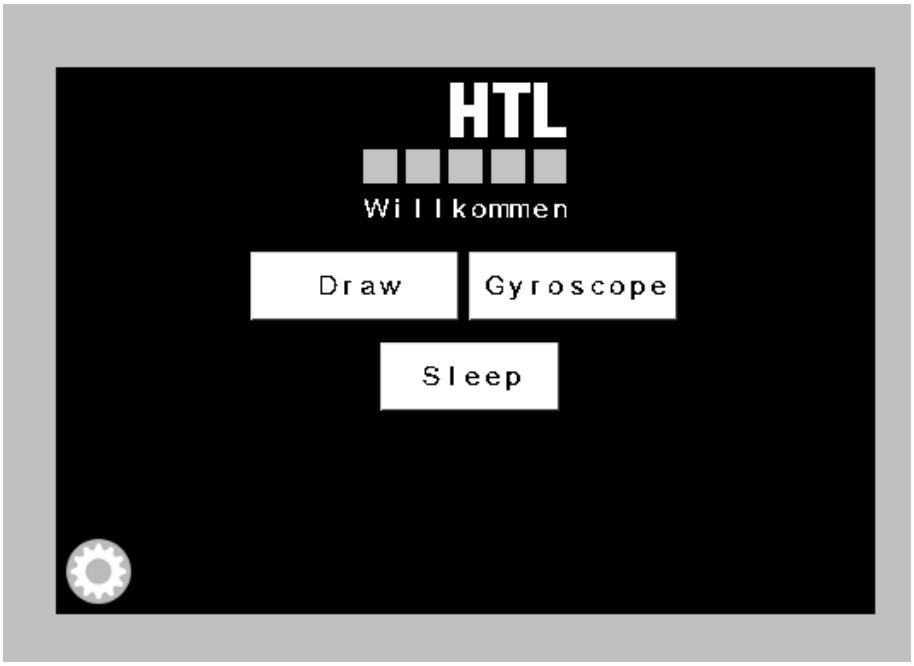


Abbildung 147: Tag der offenen Tür: GUI Hauptansicht

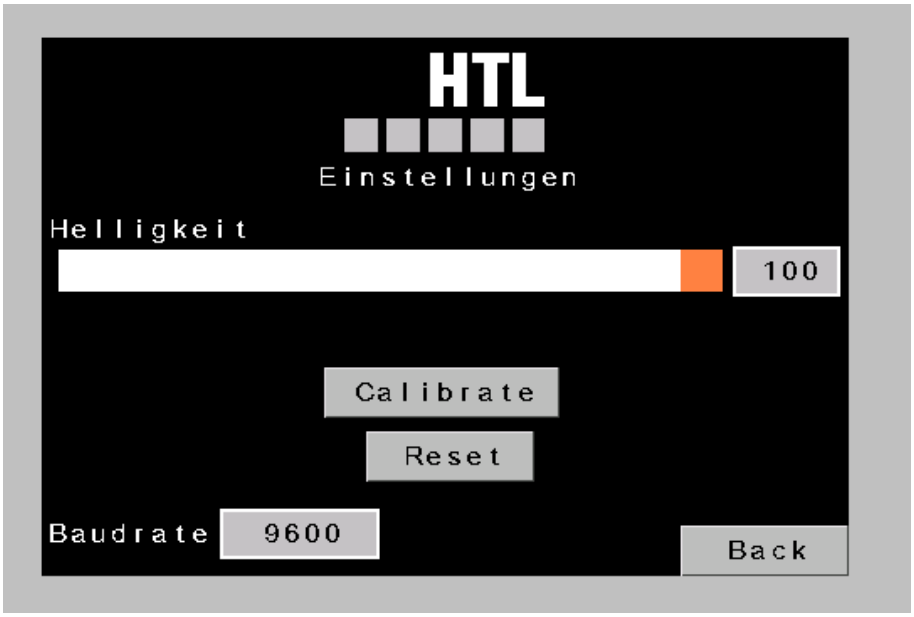


Abbildung 148: Tag der offenen Tür: GUI Einstellungen

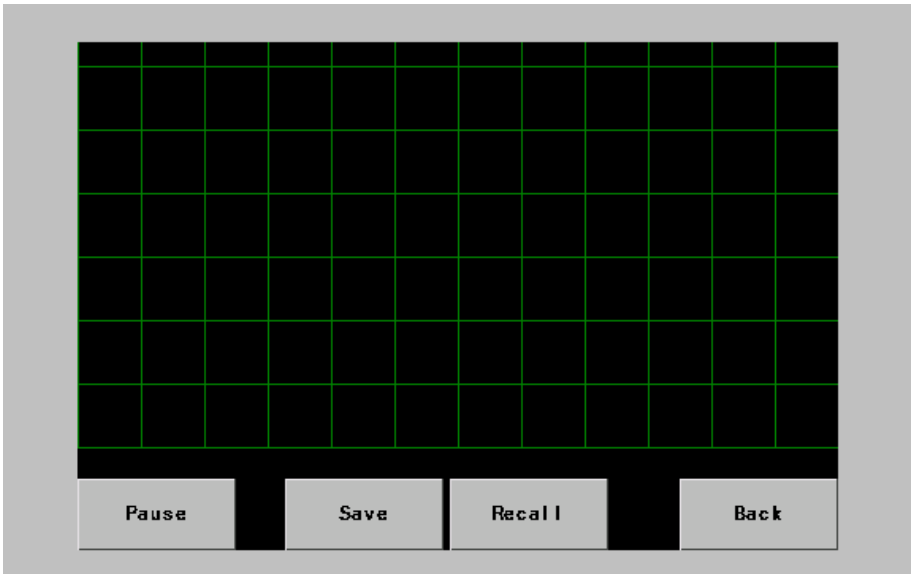


Abbildung 149: Tag der offenen Tür: GUI BMA Daten

7.1.1 main.c

Das Hauptprogramm in `main.c` initsialisiert alle Ports sowie den SysTick Interrupt. Die einzelnen Komponenten wurden für die Übersichtlichkeit in extra Files ausgelagert. Im Hauptprogramm ist außerdem der Main-Loop, welcher die Werte vom BMA einliest und am Display ausgibt. Der Main-Loop fragt auch Displayeingaben ab und reagiert auf diese (Start/Stopp, Speichern und Laden).

Listing 2: Tag der offenen Tür: Hauptprogramm

```
1  /*
2   Demo program for the 2017/18 open door day at HTL Hollabrunn
3      Copyright (C) 2018 Andreas Mieke
4
5      This program is free software: you can redistribute it and/or modify
6      it under the terms of the GNU General Public License as published by
7      the Free Software Foundation, either version 3 of the License, or
8      (at your option) any later version.
9
10     This program is distributed in the hope that it will be useful,
11     but WITHOUT ANY WARRANTY; without even the implied warranty of
12     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13     GNU General Public License for more details.
14
15     You should have received a copy of the GNU General Public License
16     along with this program. If not, see <https://www.gnu.org/licenses/>.
17 */
18
19 #include "display.h"
```

```
20 #include "bma.h"
21 #include "eeprom.h"
22 #include "bluetooth.h"
23
24 #define BUFFER_SIZE 256
25
26 int main()
27 {
28  // Variables needed
29  disp_state_t current_state = DISP_STATE_NONE;
30  uint8_t X, Y, Z, running = 0;
31  uint8_t bX[BUFFER_SIZE] = {128}, bY[BUFFER_SIZE] = {128}, bZ[BUFFER_SIZE] = {128};
        // 128 is the 'zero' line in the display, so the array is initzialized to 128
32  uint16_t buffer_pos = 0;
33
34  // Init everything we need
35  disp_init();
36  systick_init();
37  bma_init();
38  eeprom_init();
39  bluetooth_init();
40
41  // Enable display
42  disp_enable();
43
44  // Main loop
45  for (;;) {
46   // Check if button is pressed
47   current_state = disp_get_last_state();
48   if(current_state == DISP_STATE_START) {
49    // If start button set running to true
50    running = 1;
51   } else if(current_state == DISP_STATE_PAUSE) {
52    // On pause set running to false
53    running = 0;
54   }
55   // Only read and send data if display is actually running
56   if(running == 1) {
57    // Get accelerations
58    bma_get_acc(&X, &Y, &Z);
59    // Put them into the buffer (for save/recall)
60    bX[buffer_pos] = X;
61    bY[buffer_pos] = Y;
62    bZ[buffer_pos] = Z;
63    // Send data to display and bluetooth
64    disp_send_gyro_data(X, Y, Z);
65    bluetooth_send_gyro_data(X, Y, Z);
66    // Increment buffer position
67    buffer_pos++;
68    if(buffer_pos == BUFFER_SIZE) {
69     // If buffer size is reached, return to 0 (ringbuffer)
70     buffer_pos = 0;
71    }
```

```
72    }
73    if(current_state == DISP_STATE_SAVE) {
74     // On save, send the three arrays to the EEPROM
75     eeprom_write(0x0000, bX, BUFFER_SIZE);
76     eeprom_write(0x0400, bY, BUFFER_SIZE);
77     eeprom_write(0x0800, bZ, BUFFER_SIZE);
78    }
79    if(current_state == DISP_STATE_RECALL) {
80     // On recall read the three arrays from EEPROM
81     eeprom_read(0x0000, bX, BUFFER_SIZE);
82     eeprom_read(0x0400, bY, BUFFER_SIZE);
83     eeprom_read(0x0800, bZ, BUFFER_SIZE);
84     // Also update display and bluetooth completely with the new data in the buffer
85     for(uint16_t i = 0; i < BUFFER_SIZE; i++) {
86      disp_send_gyro_data(bX[i], bY[i], bZ[i]);
87      bluetooth_send_gyro_data(bX[i], bY[i], bZ[i]);
88     }
89    }
90   }
91  }
```

7.1.2 io.c

Dieses File enthält die Implementation des Input/Output Teils, hauptsächlich initsialisiert sie die einzelnen Peripherieeinheiten, sie stellt aber mit `USART_SendString()` auch eine häufig genutzte Funktion zum senden von Strings über UART zuer Verfügung, welche die Standard CMSIS Library nicht beinhaltet.

Listing 3: Tag der offenen Tür: I/O Implementation

```
1  /*
2   Demo program for the 2017/18 open door day at HTL Hollabrunn
3      Copyright (C) 2018 Andreas Mieke
4
5      This program is free software: you can redistribute it and/or modify
6      it under the terms of the GNU General Public License as published by
7      the Free Software Foundation, either version 3 of the License, or
8      (at your option) any later version.
9
10     This program is distributed in the hope that it will be useful,
11     but WITHOUT ANY WARRANTY; without even the implied warranty of
12     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13     GNU General Public License for more details.
14
15     You should have received a copy of the GNU General Public License
16     along with this program. If not, see <https://www.gnu.org/licenses/>.
17 */
18
19 #include "io.h"
```

```
20
21  uint8_t i2c1_inited = 0;
22  uint8_t usart1_inited = 0;
23  uint8_t usart3_inited = 0;
24
25  void i2c1_init(void)
26  {
27   // If I2C1 is already inited, do nothing
28   if (i2c1_inited == 1) {
29    return;
30   }
31
32   // Enable GPIOB and I2C1 clocks
33   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
34   RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
35
36   // Create gpio struct and fill default values
37   GPIO_InitTypeDef gpio;
38   GPIO_StructInit(&gpio);
39
40   // Set PB6 to alternate function push pull (SCL)
41   gpio.GPIO_Mode = GPIO_Mode_AF_PP;
42   gpio.GPIO_Pin = GPIO_Pin_6;
43   GPIO_Init(GPIOB, &gpio);
44
45   // Set PB7 to alternate function open drain (SDA)
46   gpio.GPIO_Mode = GPIO_Mode_AF_OD;
47   gpio.GPIO_Pin = GPIO_Pin_7;
48   GPIO_Init(GPIOB, &gpio);
49
50   // Set I2C1 clock to 400 kHz
51   I2C_InitTypeDef i2c;
52   I2C_StructInit(&i2c);
53   i2c.I2C_ClockSpeed = 400000;
54   I2C_Init(I2C1, &i2c);
55
56   // Enable I2C1
57   I2C_Cmd(I2C1, ENABLE);
58   i2c1_inited = 1;
59  }
60
61  void usart1_init(void)
62  {
63   // If USART1 is inited, do nothing
64   if (usart1_inited == 1) {
65    return;
66   }
67
68   // Enable GPIOA and USART1 clocks
69   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
70   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
71
72   // Create gpio struct and fill default values
```

```
73   GPIO_InitTypeDef gpio;
74   GPIO_StructInit(&gpio);
75
76   // Set PA9 to alternate function push pull (TxD)
77   gpio.GPIO_Mode = GPIO_Mode_AF_PP;
78   gpio.GPIO_Pin = GPIO_Pin_9;
79   GPIO_Init(GPIOA, &gpio);
80
81   // Set PA10 to input floating (RxD)
82   gpio.GPIO_Mode = GPIO_Mode_IN_FLOATING;
83   gpio.GPIO_Pin = GPIO_Pin_10;
84   GPIO_Init(GPIOA, &gpio);
85
86   // Set USART1 clock to 115 200 baud
87   USART_InitTypeDef usart;
88   USART_StructInit(&usart);
89   usart.USART_BaudRate = 115200;
90   USART_Init(USART1, &usart);
91
92   // Init USART1 clocks
93   USART_ClockInitTypeDef usartclock;
94   USART_ClockStructInit(&usartclock);
95   USART_ClockInit(USART1, &usartclock);
96
97   // Enable USART1
98   USART_Cmd(USART1, ENABLE);
99   usart1_inited = 1;
100 }
101
102 void usart3_init(void)
103 {
104  // If USART3 is inited, do nothing
105  if (usart3_inited == 1) {
106   return;
107  }
108
109  // Enable GPIOB and USART3 clocks
110  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
111  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);
112
113  // Create gpio struct and fill default values
114  GPIO_InitTypeDef gpio;
115  GPIO_StructInit(&gpio);
116
117  // Set PB10 to alternate function push pull (TxD)
118  gpio.GPIO_Mode = GPIO_Mode_AF_PP;
119  gpio.GPIO_Pin = GPIO_Pin_10;
120  GPIO_Init(GPIOB, &gpio);
121
122  // Set PB11 to input floating (RxD)
123  gpio.GPIO_Mode = GPIO_Mode_IN_FLOATING;
124  gpio.GPIO_Pin = GPIO_Pin_11;
125  GPIO_Init(GPIOB, &gpio);
```

```
126
127  // Set USART3 clock to 115 200 baud
128  USART_InitTypeDef usart;
129  USART_StructInit(&usart);
130  usart.USART_BaudRate = 115200;
131  USART_Init(USART3, &usart);
132
133  // Init USART3 clocks
134  USART_ClockInitTypeDef usartclock;
135  USART_ClockStructInit(&usartclock);
136  USART_ClockInit(USART3, &usartclock);
137
138  // Enable USART3
139  USART_Cmd(USART3, ENABLE);
140  usart3_inited = 1;
141 }
142
143 void USART_SendString(USART_TypeDef *USARTx, char *str)
144 {
145  // Send a string, byte by byte over the specified USART
146  while (*str) {
147   while (USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET);
148   USART_SendData(USARTx, *str++);
149  }
150 }
```

7.1.3 display.c

In diesem File wird zuerst das Display initsialisiert, danach stellt das File noch Funktionen zum senden von Gyro Daten und zum Aus- und Einschalten bereit. Im File ist auch ein Interrupt-Handler für USART3 ausprogrammiert, welcher Button-Klicks welche das Display sendet verarbeitet und zwischenspeichert, bis sie von einer weiteren Funktion, welche im Main-Loop aufgerufen wird, ausgegeben werden.

Listing 4: Tag der offenen Tür: Display Implementation

```
1  /*
2   Demo program for the 2017/18 open door day at HTL Hollabrunn
3     Copyright (C) 2018 Andreas Mieke
4
5     This program is free software: you can redistribute it and/or modify
6     it under the terms of the GNU General Public License as published by
7     the Free Software Foundation, either version 3 of the License, or
8     (at your option) any later version.
9
10    This program is distributed in the hope that it will be useful,
11    but WITHOUT ANY WARRANTY; without even the implied warranty of
12    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13    GNU General Public License for more details.
```

```
14
15      You should have received a copy of the GNU General Public License
16      along with this program. If not, see <https://www.gnu.org/licenses/>.
17  */
18
19  #include "display.h"
20
21  disp_state_t state = DISP_STATE_NONE;
22  uint8_t running = 0;
23
24  void USART3_IRQHandler(void)
25  {
26   // HAndle only USART3 RXNE interrupt
27   if (USART_GetITStatus(USART3, USART_IT_RXNE) == SET) {
28    // Get display state
29    state = (disp_state_t)((USART_ReceiveData(USART3) & 0x00FF) - '0');
30    if (state > 4) {
31     state = DISP_STATE_NONE;
32    }
33   }
34   // Set running according to pressed button
35   if (state == DISP_STATE_START) {
36    running = 1;
37   } else if (state == DISP_STATE_PAUSE) {
38    running = 0;
39   }
40  }
41
42  void disp_init(void)
43  {
44   // Init USART3
45   usart3_init();
46
47   // Init USART3 interrupt
48   NVIC_InitTypeDef nvic;
49   nvic.NVIC_IRQChannel = USART3_IRQn;
50   nvic.NVIC_IRQChannelCmd = ENABLE;
51   nvic.NVIC_IRQChannelPreemptionPriority = 0;
52   nvic.NVIC_IRQChannelSubPriority = 2;
53   NVIC_Init(&nvic);
54   USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);
55
56   // Disable display till startup is complete
57   disp_disable();
58  }
59
60  disp_state_t disp_get_last_state(void)
61  {
62   // Returns the last state (which we got form the interrupt)
63   disp_state_t tmp = state;
64   state = DISP_STATE_NONE;
65   return tmp;
66  }
```

```
67
68 void disp_send_gyro_data(uint8_t X, uint8_t Y, uint8_t Z)
69 {
70  char ___str[128] = {0};
71  char *str = ___str;
72  // Print gyro data to the display
73  sprintf(str, "add 1,0,%d\xFF\xFF\xFF" "add 1,1,%d\xFF\xFF\xFF" "add 1,2,%d\xFF\xFF\xFF", X
        , Y, Z);
74  USART_SendString(USART3, str);
75 }
76
77 void disp_disable(void)
78 {
79  // Disable (dim to 0%) display
80  USART_SendString(USART3, "dim=0\xFF\xFF\xFF");
81 }
82
83 void disp_enable(void)
84 {
85  // Enable display again (dim to 100%)
86  USART_SendString(USART3, "dim=100\xFF\xFF\xFF");
87 }
```

7.1.4 bma.c

Das BMA-File enthält eine Funktion zum auslesen und verarbeiten der drei Beschleunigungsachsen X, Y und Z. Die Funktion wird vom Main Loop aufgerufen und gibt die Beschleunigung auf der übergebenen Pointern bereits für das Display vorverarbeitet (verkleinert und zentriert) zurück.

Listing 5: Tag der offenen Tür: BMA Implementation

```
1  /*
2   Demo program for the 2017/18 open door day at HTL Hollabrunn
3      Copyright (C) 2018 Andreas Mieke
4
5      This program is free software: you can redistribute it and/or modify
6      it under the terms of the GNU General Public License as published by
7      the Free Software Foundation, either version 3 of the License, or
8      (at your option) any later version.
9
10     This program is distributed in the hope that it will be useful,
11     but WITHOUT ANY WARRANTY; without even the implied warranty of
12     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13     GNU General Public License for more details.
14
15     You should have received a copy of the GNU General Public License
16     along with this program. If not, see <https://www.gnu.org/licenses/>.
17 */
```

```
18
19  #include "bma.h"
20
21  #define BMA_ADDR (uint8_t)0x70    // 0b01110000
22            // ----  Vendor address part
23            // --- User address part
24            // - Keep free for R/W bit (set by I2C_Send7bitAddress())
25
26  void bma_init(void)
27  {
28   i2c1_init();
29  }
30
31  void bma_get_acc(uint8_t *X, uint8_t *Y, uint8_t *Z)
32  {
33   int16_t acc[3];
34   // Send first register address to read
35   I2C_GenerateSTART(I2C1, ENABLE);
36   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
37   I2C_Send7bitAddress(I2C1, BMA_ADDR, I2C_Direction_Transmitter);
38   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
39   I2C_SendData(I2C1, 0x02);
40   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
41   I2C_GenerateSTOP(I2C1, ENABLE);
42
43   // Start Rx transmission
44   I2C_GenerateSTART(I2C1, ENABLE);
45   I2C_AcknowledgeConfig(I2C1, ENABLE);
46   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
47   I2C_Send7bitAddress(I2C1, BMA_ADDR, I2C_Direction_Receiver);
48   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
49
50   // Read X LSB
51   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
52   acc[0] = (I2C_ReceiveData(I2C1) & 0xC0) >> 6;
53   // Read X MSB
54   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
55   acc[0] = (I2C_ReceiveData(I2C1) & 0xFF) << 2 | (acc[0] & 0x0003);
56   if(acc[0] & 0x0200) acc[0] |= 0xFC00;
57
58   // Read Y LSB
59   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
60   acc[1] = (I2C_ReceiveData(I2C1) & 0xC0) >> 6;
61   // Read Y MSB
62   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
63   acc[1] = (I2C_ReceiveData(I2C1) & 0xFF) << 2 | (acc[1] & 0x0003);
64   if(acc[1] & 0x0200) acc[1] |= 0xFC00;
65
66   // Read Z LSB
67   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
68   acc[2] = (I2C_ReceiveData(I2C1) & 0xC0) >> 6;
69   // Read Z MSB
70   I2C_AcknowledgeConfig(I2C1, DISABLE);
```

```
71  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
72  acc[2] = (I2C_ReceiveData(I2C1) & 0xFF) << 2 | (acc[2] & 0x0003);
73  if(acc[2] & 0x0200) acc[2] |= 0xFC00;
74
75  // Stop condition
76  I2C_GenerateSTOP(I2C1, ENABLE);
77
78  // Calculate display values, with a maximum amplitude of 2.0g, as described in the
        datasheet
79  *X = (acc[0]/4) + 128;
80  *Y = (acc[1]/4) + 128;
81  *Z = (acc[2]/4) + 128;
82 }
```

7.1.5 eeprom.c

Dieses File enthält zwi Funktionen, eine zum lesen und eine zum schreiben auf das verbaute EEPROM, beide werden von der Main Loop aufgerufen und erlauben es ein Array von Bytes mit der Länge `length` ab einer Übergebenen Adresse zu speichern beziehungsweise in dieses zu schreiben.

Listing 6: Tag der offenen Tür: EEPROM Implementation

```
1  /*
2   Demo program for the 2017/18 open door day at HTL Hollabrunn
3      Copyright (C) 2018 Andreas Mieke
4
5      This program is free software: you can redistribute it and/or modify
6      it under the terms of the GNU General Public License as published by
7      the Free Software Foundation, either version 3 of the License, or
8      (at your option) any later version.
9
10     This program is distributed in the hope that it will be useful,
11     but WITHOUT ANY WARRANTY; without even the implied warranty of
12     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13     GNU General Public License for more details.
14
15     You should have received a copy of the GNU General Public License
16     along with this program. If not, see <https://www.gnu.org/licenses/>.
17 */
18
19 #include "eeprom.h"
20
21 extern volatile uint32_t SysTickCnt;
22
23 // EEPROM addresse
24 #define EEPROM_ADDR (uint8_t)0xA0   // 0b10100000
25           // ----  Vendor address part
26           // --- User address part
```

```
27            // - Keep free for R/W bit (set by I2C_Send7bitAddress())
28
29  uint32_t last_write_tick = 0;
30
31  void eeprom_init(void)
32  {
33   // Init I2C1
34   i2c1_init();
35  }
36
37  void eeprom_read(uint16_t address, uint8_t *data, uint16_t length)
38  {
39   uint16_t cur_pos;
40   // Send address of EEPROM and start address on EEPROM
41   I2C_GenerateSTART(I2C1, ENABLE);
42   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
43   I2C_Send7bitAddress(I2C1, EEPROM_ADDR, I2C_Direction_Transmitter);
44   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
45   I2C_SendData(I2C1, (address & 0xFF00) >> 8);
46   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
47   I2C_SendData(I2C1, (address & 0x00FF));
48   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
49
50   // Switch to receive mode
51   I2C_GenerateSTART(I2C1, ENABLE);
52   I2C_AcknowledgeConfig(I2C1, ENABLE);
53   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
54   I2C_Send7bitAddress(I2C1, EEPROM_ADDR, I2C_Direction_Receiver);
55   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
56
57   // Read all bytes and disable ACK on last byte
58   for(cur_pos = 0; cur_pos < length; cur_pos++) {
59    if(cur_pos == length - 1) {
60     I2C_AcknowledgeConfig(I2C1, DISABLE);
61    }
62    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
63    data[cur_pos] = I2C_ReceiveData(I2C1);
64   }
65
66   // Send stop condition
67   I2C_GenerateSTOP(I2C1, ENABLE);
68  }
69
70  void eeprom_write(uint16_t address, uint8_t *data, uint16_t length)
71  {
72   uint8_t cur_page = 0;
73   uint16_t cur_pos = 0;
74   address = address & 0xFFA0;
75
76   // If more than one page is needed, cycle over the pages
77   for(cur_page = 0; cur_page <= ((length-1)/64); cur_page++) {
78    // Wait 5 ms for the write cycle (see datasheet)
79    while((SysTickCnt - last_write_tick) <= 5);
```

```
80    // Send start condition
81    I2C_GenerateSTART(I2C1, ENABLE);
82    // Send EEPROM address and start address of data to send
83    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
84    I2C_Send7bitAddress(I2C1, EEPROM_ADDR, I2C_Direction_Transmitter);
85    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
86    I2C_SendData(I2C1, (address & 0xFF00) >> 8);
87    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
88    I2C_SendData(I2C1, (address & 0x00FF));
89    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
90
91    // Send max 64 bytes (1 page) of data
92    for(; (cur_pos < length) && (cur_pos%64 <= 63); cur_pos++) {
93     I2C_SendData(I2C1, data[cur_pos]);
94     while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
95    }
96
97    // Generate stop condition and calculate address of next page (if needed)
98    I2C_GenerateSTOP(I2C1, ENABLE);
99    address += 0x0040;
100   last_write_tick = SysTickCnt;
101  }
102 }
```

7.1.6 bluetooth.c

Das Bluetooth-File enthält eine Funktion zum senden von Gyro Data, welche einen String generiert und diesen über USART1 (wo das Bluetooth Modul angeschlossen ist) ausgibt.

> ☞ Im finalen Programm wurde das Senden der Werte über Bluetooth weggelassen, da es die Main-Loop zu sehr verlangsamte, und somit die Displayausgabe nicht mehr flüssig genug war.

Listing 7: Tag der offenen Tür: Bluetooth Implementation

```
1  /*
2   Demo program for the 2017/18 open door day at HTL Hollabrunn
3     Copyright (C) 2018 Andreas Mieke
4
5     This program is free software: you can redistribute it and/or modify
6     it under the terms of the GNU General Public License as published by
7     the Free Software Foundation, either version 3 of the License, or
8     (at your option) any later version.
9
10    This program is distributed in the hope that it will be useful,
11    but WITHOUT ANY WARRANTY; without even the implied warranty of
12    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```
13      GNU General Public License for more details.
14
15      You should have received a copy of the GNU General Public License
16      along with this program. If not, see <https://www.gnu.org/licenses/>.
17  */
18
19  #include "bluetooth.h"
20
21  void bluetooth_init(void)
22  {
23   // Init USART1
24   usart1_init();
25  }
26
27  void bluetooth_send_gyro_data(uint8_t X, uint8_t Y, uint8_t Z)
28  {
29   char ___str[128] = {0};
30   char *str = ___str;
31   // Print data comma-sperated to String
32   sprintf(str, "%d,%d,%d\r\n", X, Y, Z);
33   // Send string over USARt to the bluetooth module
34   USART_SendString(USART1, str);
35  }
```

7.2 Testprogramm Minimalsystem

Um die im Unterricht hergestellten Einheiten des Minimalsystems testen zu können, musste ein Testpgrogramm geschrieben werden, welches alle verwendeten Peripherieeinheiten ansteuern kann. Dabei war es nicht das Ziel eine bestimmte Funktion zu erreichen, sondern zu testen, ob die Busse komplett durchverbunden sind, oder ob sich irgendwo auf der Leiterkarte kalte Lötstellen oder andere Fehler befinden. Dementsprechend testen diese Tests nur ob generell eine Kommunikation mit einer Peripherieeinheit möglich ist, und nicht ob diese auch korrekt funktioniert.

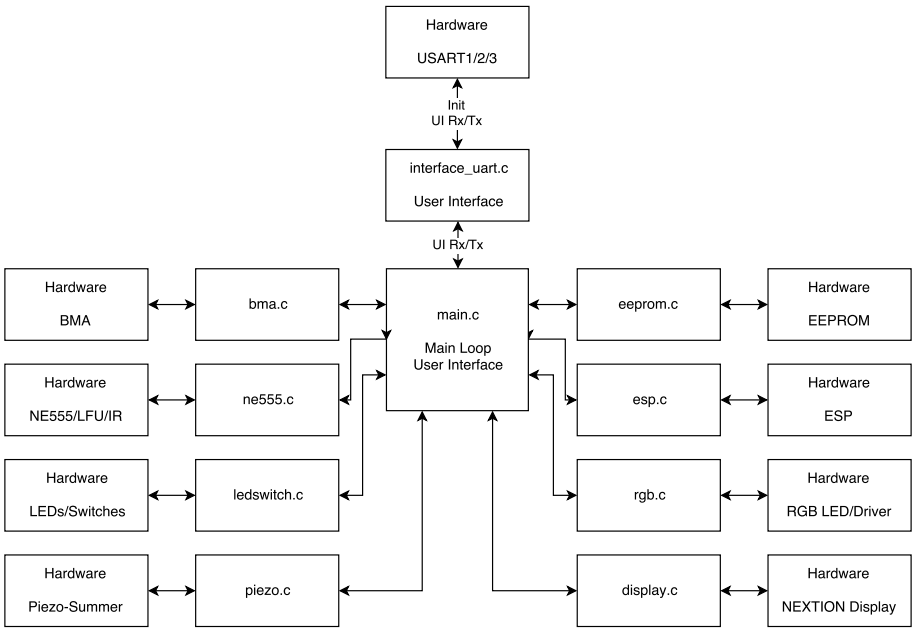


Abbildung 150: Tests: Blockschaltbild

7.2.1 main.c

Im Hauptprogramm werden die UARTs für das anzeigen der Menüs konfiguriert, die Menüs angezeigt und auf Eingaben auf eben diese reagiert. Die einzelnen Tests wurden möglichst kompakt und modular geschrieben. Ein Test stellt dabei immer ein Interface bestehend aus Init-, Run- und DeInit-Funktion.

Zusätzlich zum C-File existiert auch ein `main.h`, welches einige globale Variablen, Defines und Typen enthält.

Listing 8: Tests: Hauptprogramm

```
1  /*
2   Manufacturing tests for the new cortex minimal system
3      Copyright (C) 2018 Andreas Mieke
4
5      This program is free software: you can redistribute it and/or modify
6      it under the terms of the GNU General Public License as published by
7      the Free Software Foundation, either version 3 of the License, or
8      (at your option) any later version.
9
10     This program is distributed in the hope that it will be useful,
11     but WITHOUT ANY WARRANTY; without even the implied warranty of
12     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13     GNU General Public License for more details.
14
15     You should have received a copy of the GNU General Public License
```

```
16      along with this program. If not, see <https://www.gnu.org/licenses/>.
17  */
18
19  #include <stdio.h>
20  #include "interface_uart.h"
21
22  #include "main.h"
23
24  #include "bma.h"
25  #include "ne555.h"
26  #include "ledswitch.h"
27  #include "eeprom.h"
28  #include "esp.h"
29  #include "rgb.h"
30  #include "piezo.h"
31  #include "display.h"
32
33  volatile uint32_t SysTickCnt = 0;
34  USART_TypeDef *used_usart;
35
36  void SysTick_Handler()
37  {
38   // Increment SysTickCnt
39   SysTickCnt++;
40  }
41
42  void USART2_IRQHandler()
43  {
44   // Print received data to the used UART
45   USART_SendData(used_usart, USART_ReceiveData(USART2));
46  }
47
48  void wait(uint32_t ms)
49  {
50   // Wait the specified time in milliseconds
51   uint32_t SysTickCntHold = SysTickCnt;
52   while((SysTickCnt - SysTickCntHold) <= ms);
53  }
54
55  int main()
56  {
57   char buffer[1024];
58   // Create test and next test enum, set both to not init
59   enum test_t current_test = test_not_init, next_test = test_not_init;
60   // Create interface enum, set it to none
61   enum iface_t control_interface = interface_none;
62
63   // Endless main loop
64   for (;;) {
65    // Switch the interface
66    switch (control_interface) {
67     // If no interface has been specified, init everything, print welcome and wait for
          specification
```

```
68      case interface_none:
69       init_all();
70       send_welcome();
71       control_interface = (enum iface_t)wait_for_start();
72       // Switch thru the return value
73       switch (control_interface) {
74        // Set used_usart to the right one, depeinding on the selected interface
75        case interface_usart1:
76         used_usart = USART1;
77         break;
78        case interface_usart2:
79         used_usart = USART2;
80         break;
81        case interface_usart3:
82         used_usart = USART3;
83         break;
84        // If there is no one, set it back to none and do everything again
85        default:
86         control_interface = interface_none;
87         break;
88       }
89       break;
90      // If a interface is specified look up the test to du
91      case interface_usart1:
92      case interface_usart2:
93      case interface_usart3:
94       // Switch thru the tests
95       switch(current_test) {
96        // If not inited, set the test to none
97        case test_not_init:
98         current_test = test_none;
99         break;
100       // If none, print main menu and wait for a test to be selected
101       case test_none:
102        USART_SendString(used_usart, "\x1B[2J\x1B[0;0HManufacturing test software, Version
               " VERSION "\r\n\r\n\
103 \tTo run tests, enter one of the following numbers:\r\n\t\t[2]\tBMA\r\n\t\t[3]\tNE555/
    LFU/IR\r\n\
104 \t\t[4]\tLEDs and Switches\r\n\t\t[5]\tESP\r\n\t\t[6]\tEEPROM\r\n\t\t[7]\tRGB LED\r\n\
    t\t[8]\tPiezo\r\n\t\t[9]\tDisplay\r\n\r\nWaiting for your selection... ");
105        current_test = (enum test_t)wait_for_test(used_usart);
106        break;
107       // Else print the page for the selected test and then run it
108       case test_bma:
109        USART_SendString(used_usart, "\x1B[2J\x1B[0;0HBMA Test\r\n\r\nThis tests the
             correct function\
110  of the used BMA gyroscope as well as I2C port 1. Below you should see the current
     acceleration values printed\
111  for the X, Y and Z axis. Where the Z axis should show something around 1g, as this is
      the gravitational\
112  acceleration on the Earth, of course this value is lower if you run this program on
     the moon!\r\n\r\n\
113 To return to the main menu press a button.\r\n\r\n\r\n");
```

```
114         // Init the test and create variables if needed
115         init_bma();
116         float accs[3];
117         // Test loop
118         for(;;) {
119          // Run the test (here: get the acceleration avlues)
120          run_bma(accs);
121          // Print the values to the interface
122          sprintf(buffer, "\x1B[K\rX: % 02.5f, Y: % 02.5f, Z: % 02.5f", accs[0], accs[1], accs[2]);
123          USART_SendString(used_usart, buffer);
124          // Check if there is a new test (or main menu) selected
125          next_test = (enum test_t)get_test(used_usart);
126          if (next_test != test_not_init) {
127           // If the new test differs from the current one, change the current one so that
                  in the next run of the main loop the right test gets executed
128           current_test = next_test;
129           // DeInit the peripherals used for the test
130           deinit_bma();
131           // Leave the test loop, let the main mloop continue to run
132           break;
133          }
134          // Wait some time to be able to read the returned values
135          wait(50);
136         }
137         break;
138        // Run NE555/LFU/IR test, for more information about the executed commands see BMA
             test (above)
139        case test_ne555:
140         USART_SendString(used_usart, "\x1B[2J\x1B[0;0HNE555/LFU/IR Test\r\n\r\nThis tests
             the correct function\
141   of the NE555/LFU/IR on the board. The currently selected frequency should be printed
       below!\r\n\r\n\
142  To return to the main menu press a button.\r\n\r\n\r\n");
143         init_ne555(&SysTickCnt);
144         float freq;
145         for(;;) {
146          run_ne555(&freq);
147          sprintf(buffer, "\x1B[K\r% 05.1fHz", freq);
148          USART_SendString(used_usart, buffer);
149          next_test = (enum test_t)get_test(used_usart);
150          if (next_test != test_not_init) {
151           current_test = next_test;
152           deinit_ne555();
153           break;
154          }
155          wait(50);
156         }
157         break;
158        // Run LED/Switch test, for more information about the executed commands see BMA
             test (above)
159        case test_led:
160         USART_SendString(used_usart, "\x1B[2J\x1B[0;0HLED/Switch Test\r\n\r\nThis tests
             the correct function\
```

```
161  of the switches and the leds on the board. The currently selected leds are shown
         hexadecimal below!\r\n\r\n\
162 To return to the main menu press a button.\r\n\r\n\r\n");
163       init_ledswitch();
164       uint8_t switches;
165       for(;;) {
166        run_ledswitch(&switches);
167        sprintf(buffer, "\x1B[K\rSelected: 0x%02x", switches);
168        USART_SendString(used_usart, buffer);
169        next_test = (enum test_t)get_test(used_usart);
170        if (next_test != test_not_init) {
171         current_test = next_test;
172         deinit_ledswitch();
173         break;
174        }
175        wait(50);
176       }
177       break;
178      // Run ESP test, for more information about the executed commands see BMA test (
             above)
179      case test_esp:
180       USART_SendString(used_usart, "\x1B[2J\x1B[0;0HESP Test\r\n\r\nThis tests the
             correct function\
181  of the ESP module on the board. The ESP should open a WLAN AP!\r\n\r\n\
182 To return to the main menu press a button.\r\n\r\n\r\n");
183       init_esp();
184       sprintf(buffer, "\x1B[K\rRunning test...\r\n\r\n");
185       USART_SendString(used_usart, buffer);
186       run_esp();
187       for(;;) {
188        next_test = (enum test_t)get_test(used_usart);
189        if (next_test != test_not_init) {
190         current_test = next_test;
191         deinit_esp();
192         break;
193        }
194        wait(50);
195       }
196       break;
197      // Run EEPROM test, for more information about the executed commands see BMA test
             (above)
198      case test_eeprom:
199       USART_SendString(used_usart, "\x1B[2J\x1B[0;0HEEPROM Test\r\n\r\nThis tests the
             correct function\
200  of the EEPROM on the board. The test will run and show an OK or NOK below!\r\n\r\n\
201 To return to the main menu press a button.\r\n\r\n\r\n");
202       init_eeprom(&SysTickCnt);
203       uint8_t success;
204       sprintf(buffer, "\x1B[K\rRunning test...");
205       USART_SendString(used_usart, buffer);
206       run_eeprom(&success);
207       if(success == 1) sprintf(buffer, "OK");
208       else sprintf(buffer, "NOK");
```

```
209         USART_SendString(used_usart, buffer);
210         for(;;) {
211          next_test = (enum test_t)get_test(used_usart);
212          if (next_test != test_not_init) {
213           current_test = next_test;
214           deinit_eeprom();
215           break;
216          }
217          wait(50);
218         }
219         break;
220       // Run RGB LED test, for more information about the executed commands see BMA test
              (above)
221       case test_rgb:
222         USART_SendString(used_usart, "\x1B[2J\x1B[0;0HRGB LED Test\r\n\r\nThis tests the
              correct function\
223  of the RGB LED and it's I2C driver on the board!\r\n\r\n\
224 To return to the main menu press a button.\r\n\r\n\r\n");
225         init_rgb();
226         for(;;) {
227          run_rgb();
228          next_test = (enum test_t)get_test(used_usart);
229          if (next_test != test_not_init) {
230           current_test = next_test;
231           deinit_rgb();
232           break;
233          }
234          wait(50);
235         }
236         break;
237       // Run Piezo test, for more information about the executed commands see BMA test (
              above)
238       case test_piezo:
239         USART_SendString(used_usart, "\x1B[2J\x1B[0;0HPiezo Test\r\n\r\nThis tests the
              correct function\
240  of the Piezo on the board. You should hear the frequency printed below!\r\n\r\n\
241 To return to the main menu press a button.\r\n\r\n\r\n");
242         init_piezo(&SysTickCnt);
243         for(;;) {
244          run_piezo();
245          sprintf(buffer, "\x1B[K\r500Hz");
246          USART_SendString(used_usart, buffer);
247          next_test = (enum test_t)get_test(used_usart);
248          if (next_test != test_not_init) {
249           current_test = next_test;
250           deinit_piezo();
251           break;
252          }
253          wait(50);
254         }
255         break;
256       // Run display test, for more information about the executed commands see BMA test
              (above)
```

```
257       case test_display:
258        USART_SendString(used_usart, "\x1B[2J\x1B[0;0HDisplay Test\r\n\r\nThis tests the
              correct function\
259  of the Display on the board. You should see text printed on the display!\r\n\r\n\
260 To return to the main menu press a button.\r\n\r\n\r\n");
261        init_display();
262        for(;;) {
263         run_display();
264         sprintf(buffer, "OK");
265         USART_SendString(used_usart, buffer);
266         next_test = (enum test_t)get_test(used_usart);
267         if (next_test != test_not_init) {
268          current_test = next_test;
269          deinit_display();
270          break;
271         }
272         wait(50);
273        }
274        break;
275       // If test does not exist, go to main menu
276       default:
277        current_test = test_none;
278        break;
279      }
280     break;
281    default:
282     break;
283   }
284  }
285 }
```

7.2.2 interface_uart.c

Dieses File enthält Funktionen um alle USARTs zu initsialisieren, eine Willkommensnachricht an alle zu senden und passend auf die eingabe alle außer einen zu Deaktivieren, dieser verbleibende USART ist dann der, worüber Kommunikation läuft, während alle anderen für Tests zur Verfügung stehen. Des weiteren existieren zwei Funktionen um auf einen neuen Test (beziehungsweise auf eine neue gedrückte Taste) abzufragen. Dies kann entweder blockierend aufgerufen werden, oder nicht blockierend.

Listing 9: Tests: UART Implementation

```
1  /*
2   Manufacturing tests for the new cortex minimal system
3     Copyright (C) 2018 Andreas Mieke
4
5     This program is free software: you can redistribute it and/or modify
6     it under the terms of the GNU General Public License as published by
7     the Free Software Foundation, either version 3 of the License, or
```

```
8     (at your option) any later version.
9
10    This program is distributed in the hope that it will be useful,
11    but WITHOUT ANY WARRANTY; without even the implied warranty of
12    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13    GNU General Public License for more details.
14
15    You should have received a copy of the GNU General Public License
16    along with this program. If not, see <https://www.gnu.org/licenses/>.
17 */
18
19 #include "interface_uart.h"
20
21 void USART_SendString(USART_TypeDef *USARTx, char *str)
22 {
23  // Sends a string, character for character, over the specified UART
24  while (*str) {
25   while (USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET);
26   USART_SendData(USARTx, *str++);
27  }
28 }
29
30 void init_all(void)
31 {
32  // Enable all GPIO and USART clocks needed for USART1, 2 and 3
33  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
34  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
35  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
36  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
37  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);
38
39  // Create gpio struct and fill it with defaults
40  GPIO_InitTypeDef gpio;
41  GPIO_StructInit(&gpio);
42
43  // Set PA9 to alternate function push pull (Tx)
44  gpio.GPIO_Mode = GPIO_Mode_AF_PP;
45  gpio.GPIO_Pin = GPIO_Pin_9;
46  GPIO_Init(GPIOA, &gpio);
47
48  // Same for PA2
49  gpio.GPIO_Pin = GPIO_Pin_2;
50  GPIO_Init(GPIOA, &gpio);
51
52  // Same for PB10
53  gpio.GPIO_Pin = GPIO_Pin_10;
54  GPIO_Init(GPIOB, &gpio);
55
56  // Set PA10 to input floating (Rx)
57  gpio.GPIO_Mode = GPIO_Mode_IN_FLOATING;
58  gpio.GPIO_Pin = GPIO_Pin_10;
59  GPIO_Init(GPIOA, &gpio);
60
```

```
61   // Same for PA3
62   gpio.GPIO_Pin = GPIO_Pin_3;
63   GPIO_Init(GPIOA, &gpio);
64
65   // Same for PB11
66   gpio.GPIO_Pin = GPIO_Pin_11;
67   GPIO_Init(GPIOB, &gpio);
68
69   // Create usart struct and set baud rate to 115 200, init all three USARTs with this
         baud rate
70   USART_InitTypeDef usart;
71   USART_StructInit(&usart);
72   usart.USART_BaudRate = 115200;
73   USART_Init(USART1, &usart);
74   USART_Init(USART2, &usart);
75   USART_Init(USART3, &usart);
76
77   // Enable USART clocks
78   USART_ClockInitTypeDef usartclock;
79   USART_ClockStructInit(&usartclock);
80   USART_ClockInit(USART1, &usartclock);
81   USART_ClockInit(USART2, &usartclock);
82   USART_ClockInit(USART3, &usartclock);
83
84   // Enable the USARTs
85   USART_Cmd(USART1, ENABLE);
86   USART_Cmd(USART2, ENABLE);
87   USART_Cmd(USART3, ENABLE);
88
89   // Enable the SysTick with T = 1 ms
90   RCC_ClocksTypeDef clocks;
91   RCC_GetClocksFreq(&clocks);
92   SysTick_Config(clocks.HCLK_Frequency/1000 - 1); // SysTick T=1ms
93 }
94
95 void send_welcome(void)
96 {
97   // Send a welcome message to all three USARTs
98   USART_SendString(USART1, "\x1B[2J\x1B[0;0HManufacturing test software, press space
         ...\r\n");
99   USART_SendString(USART2, "\x1B[2J\x1B[0;0HManufacturing test software, press space
         ...\r\n");
100  USART_SendString(USART3, "\x1B[2J\x1B[0;0HManufacturing test software, press space
         ...\r\n");
101 }
102
103 unsigned int wait_for_start(void)
104 {
105   uint8_t data;
106   // Runs until space has been pressed on one UART
107   for(;;) {
108     // Checks if USART1 receive register is not empty
109     if (USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == SET) {
```

```
110     data = USART_ReceiveData(USART1);
111     // Checks if space has been pressed
112     if (data == ' ') {
113      // Disable the other UARTs and return 1 as this is the used UART
114      USART_Cmd(USART2, DISABLE);
115      USART_Cmd(USART3, DISABLE);
116      USART_DeInit(USART2);
117      USART_DeInit(USART3);
118      return 1;
119     } else {
120      USART_Cmd(USART1, DISABLE);
121      USART_DeInit(USART1);
122     }
123    }
124    // Checks if USART2 receive register is not empty
125    if (USART_GetFlagStatus(USART2, USART_FLAG_RXNE) == SET) {
126     data = USART_ReceiveData(USART2);
127     // Checks if space has been pressed
128     if (data == ' ') {
129      // Disable the other UARTs and return 2 as this is the used UART
130      USART_Cmd(USART1, DISABLE);
131      USART_Cmd(USART3, DISABLE);
132      USART_DeInit(USART1);
133      USART_DeInit(USART3);
134      return 2;
135     } else {
136      USART_Cmd(USART2, DISABLE);
137      USART_DeInit(USART2);
138     }
139    }
140    // Checks if USART3 receive register is not empty
141    if (USART_GetFlagStatus(USART3, USART_FLAG_RXNE) == SET) {
142     data = USART_ReceiveData(USART3);
143     // Checks if space has been pressed
144     if (data == ' ') {
145      // Disable the other UARTs and return 3 as this is the used UART
146      USART_Cmd(USART1, DISABLE);
147      USART_Cmd(USART2, DISABLE);
148      USART_DeInit(USART1);
149      USART_DeInit(USART2);
150      return 3;
151     } else {
152      USART_Cmd(USART3, DISABLE);
153      USART_DeInit(USART3);
154     }
155    }
156   }
157  }
158
159  unsigned int wait_for_test(USART_TypeDef *USARTx)
160  {
161   int data;
162   // Blocks until something is pressed
```

```
163  for(;;)
164  {
165   // Checks if receive register is not empty
166   if (USART_GetFlagStatus(USARTx, USART_FLAG_RXNE) == SET) {
167    data = (int)USART_ReceiveData(USARTx);
168    // If data is out of range return 1 (no test)
169    if ((data > 0x09 && data <= 0x30) || data > 0x39) return 1;
170    // Else return the correct test id
171    return (data >= 0x30) ? data - 0x30 : data;
172   }
173  }
174 }
175
176 unsigned int get_test(USART_TypeDef *USARTx)
177 {
178  int data;
179  // Checks if receive register is not empty
180  if (USART_GetFlagStatus(USARTx, USART_FLAG_RXNE) == SET) {
181   data = (int)USART_ReceiveData(USARTx);
182   // If data is out of range return 1 (no test)
183   if ((data > 0x09 && data <= 0x30) || data > 0x39) return 1;
184   // Else return the correct test id
185   return (data >= 0x30) ? data - 0x30 : data;
186  }
187  // If nothing has been sent to the UART, return 0 (test not changed)
188  return 0;
189 }
```

7.2.3 bma.c

Der BMA Test, bestehend aus Init und DeInit Funktion und einer Funktion, welche die aktuellen Beschleunigungswerte über ein float-Array ausgibt.

Listing 10: Tests: BMA Implementation

```
1  /*
2   Manufacturing tests for the new cortex minimal system
3     Copyright (C) 2018 Andreas Mieke
4
5     This program is free software: you can redistribute it and/or modify
6     it under the terms of the GNU General Public License as published by
7     the Free Software Foundation, either version 3 of the License, or
8     (at your option) any later version.
9
10    This program is distributed in the hope that it will be useful,
11    but WITHOUT ANY WARRANTY; without even the implied warranty of
12    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13    GNU General Public License for more details.
14
15    You should have received a copy of the GNU General Public License
```

```
16      along with this program. If not, see <https://www.gnu.org/licenses/>.
17  */
18
19  #include "bma.h"
20
21  void init_bma(void)
22  {
23   // Init GPIOB and I2C1 clocks
24   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
25   RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
26
27   // Create gpio struct and fill it with default values
28   GPIO_InitTypeDef gpio;
29   GPIO_StructInit(&gpio);
30
31   // Change default values for SCL port, init port
32   gpio.GPIO_Mode = GPIO_Mode_AF_PP;
33   gpio.GPIO_Pin = GPIO_Pin_6;
34   GPIO_Init(GPIOB, &gpio);
35
36   // Change default values for SDA port, init port
37   gpio.GPIO_Mode = GPIO_Mode_AF_OD;
38   gpio.GPIO_Pin = GPIO_Pin_7;
39   GPIO_Init(GPIOB, &gpio);
40
41   // Create i2c struct, fill it with default values, change clock to 400 kHz and init
         I2C
42   I2C_InitTypeDef i2c;
43   I2C_StructInit(&i2c);
44   i2c.I2C_ClockSpeed = 400000;
45   I2C_Init(I2C1, &i2c);
46
47   // Enable I2C1 interface
48   I2C_Cmd(I2C1, ENABLE);
49  }
50
51  void run_bma(float acc_f[3])
52  {
53   int16_t acc[3];
54   // Send first register address to read
55   I2C_GenerateSTART(I2C1, ENABLE);
56   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
57   I2C_Send7bitAddress(I2C1, BMA_ADDR, I2C_Direction_Transmitter);
58   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
59   I2C_SendData(I2C1, 0x02);
60   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
61   I2C_GenerateSTOP(I2C1, ENABLE);
62
63   // Start Rx transmission
64   I2C_GenerateSTART(I2C1, ENABLE);
65   I2C_AcknowledgeConfig(I2C1, ENABLE);
66   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
67   I2C_Send7bitAddress(I2C1, BMA_ADDR, I2C_Direction_Receiver);
```

```
68  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
69
70  // Read X LSB
71  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
72  acc[0] = (I2C_ReceiveData(I2C1) & 0xC0) >> 6;
73  // Read X MSB
74  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
75  acc[0] = (I2C_ReceiveData(I2C1) & 0xFF) << 2 | (acc[0] & 0x0003);
76  if(acc[0] & 0x0200) acc[0] |= 0xFC00;
77
78  // Read Y LSB
79  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
80  acc[1] = (I2C_ReceiveData(I2C1) & 0xC0) >> 6;
81  // Read Y MSB
82  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
83  acc[1] = (I2C_ReceiveData(I2C1) & 0xFF) << 2 | (acc[1] & 0x0003);
84  if(acc[1] & 0x0200) acc[1] |= 0xFC00;
85
86  // Read Z LSB
87  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
88  acc[2] = (I2C_ReceiveData(I2C1) & 0xC0) >> 6;
89  // Read Z MSB
90  I2C_AcknowledgeConfig(I2C1, DISABLE);
91  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
92  acc[2] = (I2C_ReceiveData(I2C1) & 0xFF) << 2 | (acc[2] & 0x0003);
93  if(acc[2] & 0x0200) acc[2] |= 0xFC00;
94
95  // Stop condition
96  I2C_GenerateSTOP(I2C1, ENABLE);
97
98  // Calculate float values, with a maximum amplitude of 2.0g, as described in the
        datasheet
99  acc_f[0] = 2.0f * ((float)acc[0]/512);
100 acc_f[1] = 2.0f * ((float)acc[1]/512);
101 acc_f[2] = 2.0f * ((float)acc[2]/512);
102 }
103
104 void deinit_bma(void)
105 {
106  // Disable I2C1 and de-init it
107  I2C_Cmd(I2C1, DISABLE);
108  I2C_DeInit(I2C1);
109 }
```

7.2.4 ne555.c

Der NE555 Test, bestehend aus Init und DeInit Funktion und einer Funktion, welche die Zustandswechsel über eine Sekunde misst und halbiert ausgibt (ergibt eine Frequenz). Dieser Test kann auch für den LFU und den Infrarot Sensor verwendet werden, da die

beiden auch Frequenzen ausgeben.

Listing 11: Tests: NE555 Implementation

```
1  /*
2   Manufacturing tests for the new cortex minimal system
3      Copyright (C) 2018 Andreas Mieke
4
5      This program is free software: you can redistribute it and/or modify
6      it under the terms of the GNU General Public License as published by
7      the Free Software Foundation, either version 3 of the License, or
8      (at your option) any later version.
9
10     This program is distributed in the hope that it will be useful,
11     but WITHOUT ANY WARRANTY; without even the implied warranty of
12     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13     GNU General Public License for more details.
14
15     You should have received a copy of the GNU General Public License
16     along with this program. If not, see <https://www.gnu.org/licenses/>.
17 */
18
19 #include "ne555.h"
20
21 volatile uint32_t *NE555STick, NE555STickCur;
22 uint8_t old_state;
23
24 void init_ne555(volatile uint32_t *SysTickCnt)
25 {
26  // Enable GPIOb clock
27  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
28
29  // Create gpio struct and fill it with default values
30  GPIO_InitTypeDef gpio;
31  GPIO_StructInit(&gpio);
32
33  // Set PB0 to input floating
34  gpio.GPIO_Mode = GPIO_Mode_IN_FLOATING;
35  gpio.GPIO_Pin = GPIO_Pin_0;
36  GPIO_Init(GPIOB, &gpio);
37
38  NE555STick = SysTickCnt;
39  return;
40 }
41
42 void run_ne555(float *freq)
43 {
44  uint8_t state;
45  *freq = 0.0f;
46  NE555STickCur = *NE555STick;
47  // Run for one second
48  while((*NE555STick - NE555STickCur) <= 1000)
49  {
```

```
50    state = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_0);
51    // If the state changes low -> high or high -> low increment freq and save state
52    if (state != old_state)
53    {
54     (*freq)++;
55     old_state = state;
56    }
57   }
58   // Divide freq by two to get the frequency
59   *freq /= 2;
60   return;
61  }
62
63  void deinit_ne555(void)
64  {
65   return;
66  }
```

7.2.5 ledswitch.c

Der LEDs/Switch test liest die Schalterstellungen ein und gibt diese wieder auf den LEDs aus. Bei Verwendung von USART2 kann es hier zu Problemen kommen, da die Schalter unter anderem auf den Rx und Tx Leitungen von USART2 liegen.

Listing 12: Tests: LED/Switch Implementation

```
1  /*
2   Manufacturing tests for the new cortex minimal system
3     Copyright (C) 2018 Andreas Mieke
4
5     This program is free software: you can redistribute it and/or modify
6     it under the terms of the GNU General Public License as published by
7     the Free Software Foundation, either version 3 of the License, or
8     (at your option) any later version.
9
10    This program is distributed in the hope that it will be useful,
11    but WITHOUT ANY WARRANTY; without even the implied warranty of
12    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13    GNU General Public License for more details.
14
15    You should have received a copy of the GNU General Public License
16    along with this program. If not, see <https://www.gnu.org/licenses/>.
17  */
18
19  #include "ledswitch.h"
20
21  void init_ledswitch(void)
22  {
23   // Init GPIOA and GPIOC clocks
24   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
```

```
25  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
26
27  // Create gpio struct and fill it with defaults
28  GPIO_InitTypeDef gpio;
29  GPIO_StructInit(&gpio);
30
31  // Set the LED ports to push pull
32  gpio.GPIO_Mode = GPIO_Mode_Out_PP;
33  gpio.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 |
        GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
34  GPIO_Init(GPIOC, &gpio);
35
36  // Set the switch ports to input pull up
37  gpio.GPIO_Mode = GPIO_Mode_IPU;
38  gpio.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_5 |
        GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8;
39  GPIO_Init(GPIOA, &gpio);
40  return;
41 }
42
43 void run_ledswitch(uint8_t *switches)
44 {
45  // Read switches and shift it to represent a 8 bit number
46  *switches = (GPIO_ReadInputData(GPIOA) & 0x000F) | ((GPIO_ReadInputData(GPIOA) & 0
        x01E0) >> 1);
47  // Write that 8 bit number correctly shifted again to the LEDs
48  GPIO_Write(GPIOC, ((*switches & 0xE0) << 2) | ((*switches & 0x1F) << 1));
49  return;
50 }
51
52 void deinit_ledswitch(void)
53 {
54  return;
55 }
```

7.2.6 eeprom.c

Der EEPROM-Test sendet einige Bytes an den Speicher und liest diese danach sofort wieder aus. Wenn der gesendete und ausgelesene Wert übereinstimmt, dann war der Test erfolgreich.

Listing 13: Tests: EEPROM Implementation

```
1  /*
2   Manufacturing tests for the new cortex minimal system
3     Copyright (C) 2018 Andreas Mieke
4
5     This program is free software: you can redistribute it and/or modify
6     it under the terms of the GNU General Public License as published by
7     the Free Software Foundation, either version 3 of the License, or
```

```
 8     (at your option) any later version.
 9
10     This program is distributed in the hope that it will be useful,
11     but WITHOUT ANY WARRANTY; without even the implied warranty of
12     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13     GNU General Public License for more details.
14
15     You should have received a copy of the GNU General Public License
16     along with this program. If not, see <https://www.gnu.org/licenses/>.
17 */
18
19 #include "eeprom.h"
20
21 volatile uint32_t *EEPROMSTick, EEPROMSTickCur;
22
23 uint8_t load_byte(uint16_t eeprom_addr)
24 {
25  uint8_t data;
26  // Send address to read from
27  I2C_GenerateSTART(I2C1, ENABLE);
28  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
29  I2C_Send7bitAddress(I2C1, EEPROM_ADDR, I2C_Direction_Transmitter);
30  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
31  I2C_SendData(I2C1, (eeprom_addr & 0xFF00) >> 8);
32  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
33  I2C_SendData(I2C1, (eeprom_addr & 0x00FF));
34  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
35
36  // Switch to Rx mode
37  I2C_GenerateSTART(I2C1, ENABLE);
38  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
39  I2C_Send7bitAddress(I2C1, EEPROM_ADDR, I2C_Direction_Receiver);
40  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
41
42  // Load byte
43  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
44  data = I2C_ReceiveData(I2C1);
45  I2C_GenerateSTOP(I2C1, ENABLE);
46
47  // Return data byte
48  return data;
49 }
50
51 void write_byte(uint16_t eeprom_addr, uint8_t data)
52 {
53  // Send address to write to
54  I2C_GenerateSTART(I2C1, ENABLE);
55  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
56  I2C_Send7bitAddress(I2C1, EEPROM_ADDR, I2C_Direction_Transmitter);
57  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
58  I2C_SendData(I2C1, (eeprom_addr & 0xFF00) >> 8);
59  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
60  I2C_SendData(I2C1, (eeprom_addr & 0x00FF));
```

```
61   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
62
63   // Send data byte
64   I2C_SendData(I2C1, data);
65   while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
66   I2C_GenerateSTOP(I2C1, ENABLE);
67
68   // Wait 5 ms to satisfy the write cycle time of the EEPROM
69   EEPROMSTickCur = *EEPROMSTick;
70   while((*EEPROMSTick - EEPROMSTickCur) <= 50); // 5ms write cycle, see datasheet
         param 17
71   return;
72 }
73
74 void init_eeprom(volatile uint32_t *SysTickCnt)
75 {
76   // Enable GPIOB and I2C1 cloc
77   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
78   RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
79
80   // Create gpio struct and fill it with default values
81   GPIO_InitTypeDef gpio;
82   GPIO_StructInit(&gpio);
83
84   // Set PB6 to alternate function push pull (SCL)
85   gpio.GPIO_Mode = GPIO_Mode_AF_PP;
86   gpio.GPIO_Pin = GPIO_Pin_6;
87   GPIO_Init(GPIOB, &gpio);
88
89   // Set PB7 to alternate function open drain (SDA)
90   gpio.GPIO_Mode = GPIO_Mode_AF_OD;
91   gpio.GPIO_Pin = GPIO_Pin_7;
92   GPIO_Init(GPIOB, &gpio);
93
94   // Set I2C clock to 400 kHz
95   I2C_InitTypeDef i2c;
96   I2C_StructInit(&i2c);
97   i2c.I2C_ClockSpeed = 400000;
98   I2C_Init(I2C1, &i2c);
99
100  // Enable I2C1 and save SysTick pointer
101  I2C_Cmd(I2C1, ENABLE);
102  EEPROMSTick = SysTickCnt;
103  return;
104 }
105
106 void run_eeprom(uint8_t *success)
107 {
108  // Write some data to some addresses and check if the date matches after a read, if
         not, return
109  uint8_t set, read;
110  *success = 0;
111  set = 0xAA;
```

```
112   write_byte(0x0000, set);
113   read = load_byte(0x0000);
114   if (set != read) return;
115   set = 0xBA;
116   write_byte(0x0010, set);
117   read = load_byte(0x0010);
118   if (set != read) return;
119   set = 0xAD;
120   write_byte(0x0001, set);
121   read = load_byte(0x0001);
122   if (set != read) return;
123   set = 0x00;
124   write_byte(0x0000, set);
125   read = load_byte(0x0000);
126   if (set != read) return;
127   set = 0x88;
128   write_byte(0x0002, set);
129   read = load_byte(0x0002);
130   if (set != read) return;
131   set = 0x01;
132   write_byte(0x0000, set);
133   read = load_byte(0x0000);
134   if (set != read) return;
135   set = 0x55;
136   write_byte(0x00005, set);
137   read = load_byte(0x0005);
138   if (set != read) return;
139   set = 0xAA;
140   write_byte(0x0005, set);
141   read = load_byte(0x0005);
142   if (set != read) return;
143   // If everything matches, set success to true and then return
144   *success = 1;
145   return;
146 }
147
148 void deinit_eeprom(void)
149 {
150   // Disable I2C1
151   I2C_Cmd(I2C1, DISABLE);
152   I2C_DeInit(I2C1);
153   return;
154 }
```

7.2.7 esp.c

Der ESP Test öffnet einen WLAN Access Point und einen TCP Server mit dem Port 2526, dieser gibt dann alle Daten, welche mit TCP gesendet werden über den verwendeten USART aus.

Listing 14: Tests: ESP Implementation

```
1  /*
2   Manufacturing tests for the new cortex minimal system
3      Copyright (C) 2018 Andreas Mieke
4
5      This program is free software: you can redistribute it and/or modify
6      it under the terms of the GNU General Public License as published by
7      the Free Software Foundation, either version 3 of the License, or
8      (at your option) any later version.
9
10     This program is distributed in the hope that it will be useful,
11     but WITHOUT ANY WARRANTY; without even the implied warranty of
12     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13     GNU General Public License for more details.
14
15     You should have received a copy of the GNU General Public License
16     along with this program. If not, see <https://www.gnu.org/licenses/>.
17 */
18
19 #include "esp.h"
20
21 void init_esp(void)
22 {
23  // Enable GPIOa and USART2 clocks
24  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
25  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
26
27  // Create gpio struct and fill it with default values
28  GPIO_InitTypeDef gpio;
29  GPIO_StructInit(&gpio);
30
31  // Set PB2 to alternate function push pull (Tx)
32  gpio.GPIO_Mode = GPIO_Mode_AF_PP;
33  gpio.GPIO_Pin = GPIO_Pin_2;
34  GPIO_Init(GPIOA, &gpio);
35
36  // Set PB3 to input floating (Rx)
37  gpio.GPIO_Mode = GPIO_Mode_IN_FLOATING;
38  gpio.GPIO_Pin = GPIO_Pin_3;
39  GPIO_Init(GPIOA, &gpio);
40
41  // Create usart struct and init USART2 to 115 200 baud
42  USART_InitTypeDef usart;
43  USART_StructInit(&usart);
44  usart.USART_BaudRate = 115200;
45  USART_Init(USART2, &usart);
46
47  // Init USART2 clock
48  USART_ClockInitTypeDef usartclock;
49  USART_ClockStructInit(&usartclock);
50  USART_ClockInit(USART2, &usartclock);
51
```

```
52  // Init NVIC for USART2 RXNE to see stuff sent to the TCP server. The ISR is in main.
        c
53  NVIC_InitTypeDef nvic;
54  nvic.NVIC_IRQChannel = USART2_IRQn;
55  nvic.NVIC_IRQChannelCmd = ENABLE;
56  nvic.NVIC_IRQChannelPreemptionPriority = 0;
57  nvic.NVIC_IRQChannelSubPriority = 2;
58  NVIC_Init(&nvic);
59  USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
60
61  // Enable USART2
62  USART_Cmd(USART2, ENABLE);
63 }
64
65 void esp_wait()
66 {
67  // Wait some time so the ESP can execute the commands sent
68  int i, j;
69  for(j=0;j<500;j++) {
70   for(i=0;i<65536;i++);
71  }
72 }
73
74 void send_str(char *str) {
75  // Sends a string over UART
76  while(*str) {
77   while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET);
78   USART_SendData(USART2, *str++);
79  }
80 }
81
82 void run_esp(void)
83 {
84  // Restore the ESP to defaults, then wait
85  send_str("AT+RESTORE\r\n");
86  esp_wait();
87  // Print out the current mode
88  send_str("AT+CWMODE?\r\n");
89  esp_wait();
90  // Set mode to access point
91  send_str("AT+CWMODE=2\r\n");
92  esp_wait();
93  // Reset
94  send_str("AT+RST\r\n");
95  esp_wait();
96  // Enable access point
97  send_str("AT+CIPMUX=1\r\n");
98  esp_wait();
99  // Start TCP server on port 2526
100 send_str("AT+CIPSERVER=1,2526\r\n");
101 esp_wait();
102 // Print out the current IP address
103 send_str("AT+CIPAP_CUR?\r\n");
```

```
104 }
105
106 void deinit_esp(void)
107 {
108  // Disable the interrupt, then disable USART2
109  USART_ITConfig(USART2, USART_IT_RXNE, DISABLE);
110  USART_Cmd(USART2, DISABLE);
111  USART_DeInit(USART2);
112 }
```

7.2.8 rgb.c

Der Test für die RGB LED prüft ob die Kommunikation möglich ist, in dem er die LED bunt blinken lässt.

Listing 15: Tests: RGB LED Implementation

```
1  /*
2   Manufacturing tests for the new cortex minimal system
3      Copyright (C) 2018 Andreas Mieke
4
5      This program is free software: you can redistribute it and/or modify
6      it under the terms of the GNU General Public License as published by
7      the Free Software Foundation, either version 3 of the License, or
8      (at your option) any later version.
9
10     This program is distributed in the hope that it will be useful,
11     but WITHOUT ANY WARRANTY; without even the implied warranty of
12     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13     GNU General Public License for more details.
14
15     You should have received a copy of the GNU General Public License
16     along with this program. If not, see <https://www.gnu.org/licenses/>.
17 */
18
19 #include "rgb.h"
20 #include "main.h"
21
22 // Define registers of the RGB LED driver
23 typedef enum {
24  RGB_SHUTDOWN = 0x00,
25  RGB_MAXCUR,
26  RGB_RED,
27  RGB_GREEN,
28  RGB_BLUE,
29  RGB_UPLEND,
30  RGB_DOWNLEND,
31  RGB_DIMSTEP
32 } RGB_T;
33
```

```
34
35 void rgb_send_command(uint8_t data)
36 {
37  // Generate start condition and transmit address
38  I2C_GenerateSTART(I2C1, ENABLE);
39  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
40  I2C_Send7bitAddress(I2C1, RGB_ADDR, I2C_Direction_Transmitter);
41  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
42
43  // Send data
44  I2C_SendData(I2C1, data);
45  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
46
47  // Generate stop condition
48  I2C_GenerateSTOP(I2C1, ENABLE);
49
50  return;
51 }
52
53 void init_rgb(void)
54 {
55  // Enable GPIOB anf I2C1 clocks
56  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
57  RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
58
59  // Create gpio struct and fill it with default values
60  GPIO_InitTypeDef gpio;
61  GPIO_StructInit(&gpio);
62
63  // SET PB6 to alternate function push pull (SCL)
64  gpio.GPIO_Mode = GPIO_Mode_AF_PP;
65  gpio.GPIO_Pin = GPIO_Pin_6;
66  GPIO_Init(GPIOB, &gpio);
67
68  // Set PB7 to alternate function open drain (SDA)
69  gpio.GPIO_Mode = GPIO_Mode_AF_OD;
70  gpio.GPIO_Pin = GPIO_Pin_7;
71  GPIO_Init(GPIOB, &gpio);
72
73  // Init I2C1 to 400 kHz
74  I2C_InitTypeDef i2c;
75  I2C_StructInit(&i2c);
76  i2c.I2C_ClockSpeed = 400000;
77  I2C_Init(I2C1, &i2c);
78
79  // Enable I2C1 driver
80  I2C_Cmd(I2C1, ENABLE);
81
82  // Set the max current the driver should allow
83  // LED actually allows 40mA, but driver only handles around 30
84  rgb_send_command(0x3F);
85  return;
86 }
```

```
87
88  void run_rgb(void)
89  {
90   // Set random colors for 100 ms each
91   rgb_send_command(0x5F);
92   wait(100);
93   rgb_send_command(0x7F);
94   wait(100);
95   rgb_send_command(0x9F);
96   wait(100);
97   rgb_send_command(0x50);
98   wait(100);
99   rgb_send_command(0x70);
100  wait(100);
101  rgb_send_command(0x90);
102  wait(100);
103  rgb_send_command(0x40);
104  wait(100);
105  rgb_send_command(0x60);
106  wait(100);
107  rgb_send_command(0x80);
108  wait(100);
109  return;
110 }
111
112 void deinit_rgb(void)
113 {
114  // Disable I2C1
115  I2C_Cmd(I2C1, DISABLE);
116  I2C_DeInit(I2C1);
117  return;
118 }
```

7.2.9 piezo.c

Der Piezo Test gibt einen ton auf eben diesem aus.

Listing 16: Tests: Piezo Implementation

```
1  /*
2   Manufacturing tests for the new cortex minimal system
3      Copyright (C) 2018 Andreas Mieke
4
5      This program is free software: you can redistribute it and/or modify
6      it under the terms of the GNU General Public License as published by
7      the Free Software Foundation, either version 3 of the License, or
8      (at your option) any later version.
9
10     This program is distributed in the hope that it will be useful,
11     but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
12      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13      GNU General Public License for more details.
14
15      You should have received a copy of the GNU General Public License
16      along with this program. If not, see <https://www.gnu.org/licenses/>.
17  */
18
19  #include "piezo.h"
20
21  volatile uint32_t *PiezoSTick, PiezoSTickCur, Freq;
22
23  void init_piezo(volatile uint32_t *SysTickCnt)
24  {
25   // Enable GPIOB clock
26   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
27
28   // Create gpio struct and fill it with default values
29   GPIO_InitTypeDef gpio;
30   GPIO_StructInit(&gpio);
31
32   // Set PB0 to output push pull
33   gpio.GPIO_Mode = GPIO_Mode_Out_PP;
34   gpio.GPIO_Pin = GPIO_Pin_0;
35   GPIO_Init(GPIOB, &gpio);
36
37   PiezoSTick = SysTickCnt;
38   return;
39  }
40
41  void run_piezo()
42  {
43   uint8_t pstate = 0x0;
44   PiezoSTickCur = *PiezoSTick;
45   // Run for one second
46   while((*PiezoSTick - PiezoSTickCur) <= 1000)
47   {
48    Freq = *PiezoSTick;
49    // Wait for 1 millisecond
50    while((*PiezoSTick - Freq) <= 1);
51    // Toggle output
52    if (pstate) {
53     pstate = ~pstate;
54     GPIO_WriteBit(GPIOB, GPIO_Pin_0, Bit_SET);
55    } else {
56     pstate = ~pstate;
57     GPIO_WriteBit(GPIOB, GPIO_Pin_0, Bit_RESET);
58    }
59   }
60   return;
61  }
62
63  void deinit_piezo(void)
64  {
```

```
65  return;
66 }
```

7.2.10 display.c

Um das Display zu testen, wird mehrmals der Screen gelöscht und in einer anderen Farbe gezeichnet. Dieser Befehl sollte auch ohne extra Test-Programm, unabhänging von dem zur Zeit geflashten Programm, auf dem Display funktionieren.

Listing 17: Tests: Display Implementation

```
1  /*
2   Manufacturing tests for the new cortex minimal system
3      Copyright (C) 2018 Andreas Mieke
4
5      This program is free software: you can redistribute it and/or modify
6      it under the terms of the GNU General Public License as published by
7      the Free Software Foundation, either version 3 of the License, or
8      (at your option) any later version.
9
10     This program is distributed in the hope that it will be useful,
11     but WITHOUT ANY WARRANTY; without even the implied warranty of
12     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13     GNU General Public License for more details.
14
15     You should have received a copy of the GNU General Public License
16     along with this program. If not, see <https://www.gnu.org/licenses/>.
17 */
18
19 #include "display.h"
20
21 #define COLORS 8
22
23 uint8_t i = 0;
24 char colors[COLORS][7] = {
25  "RED",
26  "BLUE",
27  "GRAY",
28  "BLACK",
29  "WHITE",
30  "GREEN",
31  "BROWN",
32  "YELLOW"
33 };
34
35 void init_display(void)
36 {
37  // Init GPIOB and USART3 clocks
38  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
39  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);
```

```
40
41   // Create gpio struct and fill it with default values
42   GPIO_InitTypeDef gpio;
43   GPIO_StructInit(&gpio);
44
45   // Set PB10 to alternate function push pull
46   gpio.GPIO_Mode = GPIO_Mode_AF_PP;
47   gpio.GPIO_Pin = GPIO_Pin_10;
48   GPIO_Init(GPIOB, &gpio);
49
50   // Set PB11 to input floating
51   gpio.GPIO_Mode = GPIO_Mode_IN_FLOATING;
52   gpio.GPIO_Pin = GPIO_Pin_11;
53   GPIO_Init(GPIOB, &gpio);
54
55   // Create usart struct and set USART3 to 9600 baud
56   USART_InitTypeDef usart;
57   USART_StructInit(&usart);
58   usart.USART_BaudRate = 9600;
59   USART_Init(USART3, &usart);
60
61   // Init USART clock
62   USART_ClockInitTypeDef usartclock;
63   USART_ClockStructInit(&usartclock);
64   USART_ClockInit(USART3, &usartclock);
65
66   // Enable USART3
67   USART_Cmd(USART3, ENABLE);
68  }
69
70  void run_display(void)
71  {
72   // String to store command
73   char cmd[16], *cptr;
74   cptr = cmd;
75   // Fill string with every color from colors[]
76   sprintf(cmd, "cls %s\xFF\xFF\xFF", colors[i++]);
77   // If the last color is reached, begin again
78   if (i == COLORS) i = 0;
79   // Send character for character to USART3 (the display)
80   while(*cptr) {
81    while(USART_GetFlagStatus(USART3, USART_FLAG_TXE) == RESET);
82    USART_SendData(USART3, *cptr++);
83   }
84  }
85
86  void deinit_display(void)
87  {
88   // Reset command
89   char *cmd = "rest\xFF\xFF\xFF";
90   // Reset the display after the test finishes
91   while(*cmd) {
92    while(USART_GetFlagStatus(USART3, USART_FLAG_TXE) == RESET);
```

```
93    USART_SendData(USART3, *cmd++);
94   }
95   // Wait till the last character is sent
96   while(USART_GetFlagStatus(USART3, USART_FLAG_TXE) == RESET);
97   // Then disable USART3 again
98   USART_Cmd(USART3, DISABLE);
99   USART_DeInit(USART3);
100 }
```

7.3 Ethernet

Ein Ziel dieser Diplomarbeit war es eine Library für das Arduino Ethernet Shield (Version 1.0) zu entwickeln, dies ist bis dato nicht 100%ig gelungen, da die SPI Peripherie des STM32F107RCT(6) Probleme bereitet und in manchen Situationen das LSB ignoriert. Zum Beispiel, wenn der Ethernet Controller den Status `0x13` sendet, beinhaltet das Datenregister nur `0x12`, das LSB ist also immer `0`. Das derzeitige Hauptaugenmerk liegt beim Debugging dieses Fehlers. Eine mögliche Lösung wäre das SPI Interface selbst zu implementieren, und nicht die vorhandene Peripherieeinheit zu verwenden. Der nun folgende Programmcode ist deshalb keine Library, sondern ein eigenständiges Test-Programm, welches noch nicht 100% funktioniert.

7.3.1 main.c

Das Hauptpgrogramm initsialisiert das Ethernet Modul und setzt die entsprechenden IP- und MAC-Adressen, danach wird ein TCP Server auf Port 80 gestartet, und übermittelte Pakete über UART ausgegeben.

Listing 18: Ethernet: Hauptpgrogramm

```
1  /*
2   Ethernet shield test program
3     Copyright (C) 2018 Andreas Mieke
4
5     This program is free software: you can redistribute it and/or modify
6     it under the terms of the GNU General Public License as published by
7     the Free Software Foundation, either version 3 of the License, or
8     (at your option) any later version.
9
10    This program is distributed in the hope that it will be useful,
11    but WITHOUT ANY WARRANTY; without even the implied warranty of
12    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13    GNU General Public License for more details.
14
15    You should have received a copy of the GNU General Public License
16    along with this program. If not, see <https://www.gnu.org/licenses/>.
```

```
17 */
18
19 #include "stm32f10x.h"
20 #include "stm32f10x_usart.h" // Keil::Device:StdPeriph Drivers:USART
21
22 #include "w5100.h"
23 #include "socket.h"
24
25 #include "string.h"
26
27 #define BUFFLEN 256
28
29 // Socket (TCP) with ID 1
30 SOCKET tcp = 1;
31 uint8_t buffer[BUFFLEN] = {0};
32 uint8_t len = 0;
33
34 void USART_SendString(USART_TypeDef *USARTx, uint8_t *str)
35 {
36  // Send a string character by character over UART
37  while(*str) {
38   while(USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET);
39   USART_SendData(USARTx, *str++);
40  }
41 }
42
43 int main() {
44  // Enable USART 1 and GPIOA clock
45  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
46  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
47
48  // Create gpio strct and set default values
49  GPIO_InitTypeDef gpio;
50  GPIO_StructInit(&gpio);
51
52  // Set PA9 to alternate function push pull (TxD)
53  gpio.GPIO_Mode = GPIO_Mode_AF_PP;
54  gpio.GPIO_Pin = GPIO_Pin_9;
55  GPIO_Init(GPIOA, &gpio);
56
57  // Set PA10 to input floating (RxD)
58  gpio.GPIO_Mode = GPIO_Mode_IN_FLOATING;
59  gpio.GPIO_Pin = GPIO_Pin_10;
60  GPIO_Init(GPIOA, &gpio);
61
62  // Set up USART1
63  USART_InitTypeDef usart;
64  USART_StructInit(&usart);
65  USART_Init(USART1, &usart);
66
67  // Set up USART1 clocks
68  USART_ClockInitTypeDef usartclock;
69  USART_ClockStructInit(&usartclock);
```

```
70   USART_ClockInit(USART1, &usartclock);
71
72   // Enable USART1
73   USART_Cmd(USART1, ENABLE);
74
75   // Initzialize Ethernet module
76   ETH_init();
77
78   // Set gateway IP
79   uint8_t gid[4] = {192, 168, 10, 1};
80
81   // Set subnet mask
82   uint8_t sma[4] = {255, 255, 255, 0};
83
84   // Set MAC address (unused testing address space)
85   uint8_t mac[6] = {0x1E, 0x8E, 0xA8, 0x88, 0x1C, 0xAA};
86
87   // Set source IP (IP of the Ethernet module)
88   uint8_t sip[4] = {192, 168, 10, 2};
89
90   // Transmit all these values to the module
91   ETH_set_gateway_IP(gid);
92   ETH_set_subnet_mask(sma);
93   ETH_set_mac(mac);
94   ETH_set_IP(sip);
95
96   // Try to create a TCP socket on port 80 as long as it does not work
97   do {
98    ETH_socket(tcp, ETH_SMR_TCP, 80, 0);
99   } while (ETH_socket_status(tcp) != ETH_SSR_INIT);
100
101  // Wait for the socket to finish creating
102  while(!ETH_listen(tcp));
103
104  // Endless loop
105  do {
106   // Wait for data to be available
107   while(ETH_recv_available(tcp) == 0);
108   // Receive data
109   buffer[ETH_recv(tcp, buffer, BUFFLEN)] = 0;
110   // Send data to USART
111   USART_SendString(USART1, buffer);
112   // Send string over tcp
113   ETH_send(tcp, "Hallo, Welt!", 12);
114  } while (1);
115 }
```

7.3.2 socket.c

In `socket.c` wird eine Socket API, welche der POSIX Socket API ähnlich ist, zur Verfügung gestellt. Das Headerfile `socket.h` enthält hierbei nur die Funktionsdeklarationen. Anders als bei der POSIX Socket API, muss hier jeder Socket manuell eine ID **vor** dem Erstellen erhalten. Diese ID muss zwischen 1 und 4 liegen, und repräsentiert die 4 Sockets (Register und Speicher), welche am Ethernet-Controller vorhanden sind.

Der Code wurde größtenteils von der entsprechenden Arduino Library [13] übernommen und auf C adaptiert.

Listing 19: Ethernet: Socket API

```
1  /*
2    Ethernet shield test program
3     Copyright (C) 2018 Andreas Mieke
4     Copyright (C) 2010 Arduino LLC
5
6     This program is free software: you can redistribute it and/or modify
7     it under the terms of the GNU General Public License as published by
8     the Free Software Foundation, either version 3 of the License, or
9     (at your option) any later version.
10
11    This program is distributed in the hope that it will be useful,
12    but WITHOUT ANY WARRANTY; without even the implied warranty of
13    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
14    GNU General Public License for more details.
15
16    You should have received a copy of the GNU General Public License
17    along with this program. If not, see <https://www.gnu.org/licenses/>.
18 */
19
20 #include "socket.h"
21
22 // Check if address is a net addres 0.0.0.0
23 #define is_net_addr(addr)    ((addr[0] == 0x00) && \
24                   (addr[1] == 0x00) && \
25                   (addr[2] == 0x00) && \
26                   (addr[3] == 0x00))
27
28 // Check if address is broadcast address 255.255.255.255
29 #define is_broadcast_addr(addr) ((addr[0] == 0xFF) && \
30                   (addr[1] == 0xFF) && \
31                   (addr[2] == 0xFF) && \
32                   (addr[3] == 0xFF))
33
34 // Check if port is valid (not 0)
35 #define is_valid_port(port)   (port != 0x00)
36
37 static uint16_t local_port;
38
```

```
39 uint8_t ETH_socket(SOCKET s, uint8_t protocol, uint16_t port, uint8_t flag)
40 {
41   // Check if protocol is supported
42  if ((protocol == ETH_SMR_TCP) || (protocol == ETH_SMR_UDP) || (protocol ==
       ETH_SMR_IPRAW) || (protocol == ETH_SMR_MACRAW) || (protocol ==
       ETH_SMR_PPPoE))
43   {
44     // Close socket, if open
45     ETH_close(s);
46     // Set the mode register to the protocol, plus eventual flags
47     ETH_writeSnMR(s, protocol | flag);
48     if (port != 0) {
49       // If port is not 0 (0 on client mode) write port to the port register
50       ETH_writeSnPORT(s, port);
51     }
52     else {
53       // Write a random port to the port register (for client mode)
54       local_port++;
55       ETH_writeSnPORT(s, local_port);
56     }
57
58     // Execute the open socket command
59     ETH_exec_socket_cmd(s, ETH_SCR_OPEN);
60     return 1;
61   }
62   return 0;
63 }
64
65 uint8_t ETH_socket_status(SOCKET s)
66 {
67   // Read the status register and return it
68  uint8_t tmp = ETH_readSnSR(s);
69   return tmp;
70 }
71
72 void ETH_close(SOCKET s)
73 {
74   // Execute close socket and disable interrupts
75  ETH_exec_socket_cmd(s, ETH_SCR_CLOSE);
76   ETH_writeSnIR(s, 0xFF);
77 }
78
79 uint8_t ETH_connect(SOCKET s, uint8_t * addr, uint16_t port)
80 {
81   // Check if port and address is valid
82  if (is_net_addr(addr) || is_broadcast_addr(addr) || !is_valid_port(port))
83     return 0;
84
85   // Write IP to destination register
86   ETH_writeSnDIPR(s, addr);
87   // Write port to destination register
88   ETH_writeSnDPORT(s, port);
89   // Execute connect
```

```
90    ETH_exec_socket_cmd(s, ETH_SCR_CONNECT);
91    return 1;
92  }
93
94  void ETH_disconnect(SOCKET s)
95  {
96    // Execute disconnect
97   ETH_exec_socket_cmd(s, ETH_SCR_DISCON);
98  }
99
100 uint8_t ETH_listen(SOCKET s)
101 {
102   // If state is not initzialized, return
103   if (ETH_readSnSR(s) != ETH_SSR_INIT) {
104     return 0;
105   }
106   // Execute the listen command
107   ETH_exec_socket_cmd(s, ETH_SCR_LISTEN);
108   return 1;
109 }
110
111 uint16_t ETH_send(SOCKET s, const uint8_t * buf, uint16_t len)
112 {
113  uint8_t status=0;
114   uint16_t ret=0;
115   uint16_t freesize=0;
116
117   // If data is bigger then Tx memory, split it
118   if (len > ETH_SSIZE)
119     ret = ETH_SSIZE;
120   else
121     ret = len;
122
123   do
124   {
125     // Write data to Tx memory
126     freesize = ETH_get_TX_free_size(s);
127     status = ETH_readSnSR(s);
128     if ((status != ETH_SSR_ESTABLISHED) && (status != ETH_SSR_CLOSE_WAIT))
129     {
130       ret = 0;
131       break;
132     }
133   }
134   while (freesize < ret);
135
136   // Start data processing and execute send command
137   ETH_send_data_processing(s, (uint8_t *)buf, ret);
138   ETH_exec_socket_cmd(s, ETH_SCR_SEND);
139
140   // Check interrupt if everything is okay
141   while ((ETH_readSnIR(s) & ETH_SIR_SEND_OK) != ETH_SIR_SEND_OK )
142   {
```

```
143     // If socket is closed set the state locally right
144     if (ETH_readSnSR(s) == ETH_SSR_CLOSED )
145     {
146       ETH_close(s);
147       return 0;
148     }
149   }
150   // Mark interrupt as handled
151   ETH_writeSnIR(s, ETH_SIR_SEND_OK);
152   return ret;
153 }
154
155 int16_t ETH_recv(SOCKET s, uint8_t * buf, int16_t len)
156 {
157   // Check remaining size
158   int16_t ret = ETH_get_RX_received_size(s);
159   if (ret == 0)
160   {
161     // Read status
162     uint8_t status = ETH_readSnSR(s);
163     // If status is wrong, return 0 read bytes
164     if (status == ETH_SSR_LISTEN || status == ETH_SSR_CLOSED || status ==
          ETH_SSR_CLOSE_WAIT)
165     {
166       ret = 0;
167     }
168     else
169     {
170       // Return -1 if there is nothing to read
171       ret = -1;
172     }
173   }
174   // If there is more to read then we have space, just read to the supplied limit
175   else if (ret > len)
176   {
177     ret = len;
178   }
179
180   // If there is data to read and we have space, beginn processing and reading
181   if (ret > 0)
182   {
183     ETH_recv_data_processing(s, buf, ret, 0x00);
184     ETH_exec_socket_cmd(s, ETH_SCR_RECV);
185   }
186   return ret;
187 }
188
189 int16_t ETH_recv_available(SOCKET s)
190 {
191   // Check if data is available to read
192  return ETH_get_RX_received_size(s);
193 }
194
```

```
195 uint16_t ETH_peek(SOCKET s, uint8_t *buf)
196 {
197   // Receive one byte
198  ETH_recv_data_processing(s, buf, 1, 1);
199  return 1;
200 }
201
202 uint16_t ETH_sendto(SOCKET s, const uint8_t * buf, uint16_t len, uint8_t * addr, uint16_t port)
203 {
204  uint16_t ret=0;
205
206   // If write size is bigger then memory, limit to memory
207   if (len > ETH_SSIZE) ret = ETH_SSIZE;
208   else ret = len;
209
210   // Check if address and port is right
211   if (is_net_addr(addr) || !is_valid_port(port) || ret == 0) {
212     ret = 0;
213   }
214   else
215   {
216     // Set address and port to send to
217     ETH_writeSnDIPR(s, addr);
218     ETH_writeSnDPORT(s, port);
219
220     // Process data and send it
221     ETH_send_data_processing(s, (uint8_t *)buf, ret);
222     ETH_exec_socket_cmd(s, ETH_SCR_SEND);
223
224     // Wait for the data to be sent
225     while ((ETH_readSnIR(s) & ETH_SIR_SEND_OK) != ETH_SIR_SEND_OK)
226     {
227       // If time out reset send and timout interrupts
228       if (ETH_readSnIR(s) & ETH_SIR_TIMEOUT)
229       {
230         ETH_writeSnIR(s, (ETH_SIR_SEND_OK | ETH_SIR_TIMEOUT));
231         return 0;
232       }
233     }
234
235     // Reset send ok interrupt
236     ETH_writeSnIR(s, ETH_SIR_SEND_OK);
237   }
238   return ret;
239 }
240
241 uint16_t ETH_recvfrom(SOCKET s, uint8_t * buf, uint16_t len, uint8_t * addr, uint16_t *port)
242 {
243  uint8_t head[8];
244   uint16_t data_len=0;
245   uint16_t ptr=0;
246
247   // Check if there is something to receive
```

```
248   if (len > 0)
249   {
250     // Get the receive pointer
251     ptr = ETH_readSnRX_RD(s);
252     // Handle receiving according to socket type
253     switch (ETH_readSnMR(s) & 0x07)
254     {
255     case ETH_SMR_UDP :
256       ETH_read_data(s, ptr, head, 0x08);
257       ptr += 8;
258       addr[0] = head[0];
259       addr[1] = head[1];
260       addr[2] = head[2];
261       addr[3] = head[3];
262       *port = head[4];
263       *port = (*port << 8) + head[5];
264       data_len = head[6];
265       data_len = (data_len << 8) + head[7];
266
267       ETH_read_data(s, ptr, buf, data_len);
268       ptr += data_len;
269
270       ETH_writeSnRX_RD(s, ptr);
271       break;
272
273     case ETH_SMR_IPRAW :
274       ETH_read_data(s, ptr, head, 0x06);
275       ptr += 6;
276
277       addr[0] = head[0];
278       addr[1] = head[1];
279       addr[2] = head[2];
280       addr[3] = head[3];
281       data_len = head[4];
282       data_len = (data_len << 8) + head[5];
283
284       ETH_read_data(s, ptr, buf, data_len);
285       ptr += data_len;
286
287       ETH_writeSnRX_RD(s, ptr);
288       break;
289
290     case ETH_SMR_MACRAW:
291       ETH_read_data(s, ptr, head, 2);
292       ptr+=2;
293       data_len = head[0];
294       data_len = (data_len<<8) + head[1] - 2;
295
296       ETH_read_data(s, ptr, buf, data_len);
297       ptr += data_len;
298       ETH_writeSnRX_RD(s, ptr);
299       break;
300
```

```
301       default :
302         break;
303       }
304       ETH_exec_socket_cmd(s, ETH_SCR_RECV);
305     }
306     return data_len;
307   }
308
309   void ETH_flush(SOCKET s)
310   {
311    // TODO
312   }
313
314   uint16_t ETH_igmpsend(SOCKET s, const uint8_t * buf, uint16_t len)
315   {
316    uint16_t ret=0;
317
318     if (len > ETH_SSIZE)
319       ret = ETH_SSIZE;
320     else
321       ret = len;
322
323     if (ret == 0)
324       return 0;
325
326     // Process and send IGMP data
327     ETH_send_data_processing(s, (uint8_t *)buf, ret);
328     ETH_exec_socket_cmd(s, ETH_SCR_SEND);
329
330     // Wairt for data to be sent or timeout
331     while ((ETH_readSnIR(s) & ETH_SIR_SEND_OK) != ETH_SIR_SEND_OK)
332     {
333       if (ETH_readSnIR(s) & ETH_SIR_TIMEOUT)
334       {
335         ETH_close(s);
336         return 0;
337       }
338     }
339
340     // Reset interrupt
341     ETH_writeSnIR(s, ETH_SIR_SEND_OK);
342     return ret;
343   }
344
345   int ETH_start_UDP(SOCKET s, uint8_t* addr, uint16_t port)
346   {
347     // Check if address and port are valid
348    if (is_net_addr(addr) || !is_valid_port(port))
349     {
350       return 0;
351     }
352     else
353     {
```

```
354       // Write IP and port
355       ETH_writeSnDIPR(s, addr);
356       ETH_writeSnDPORT(s, port);
357       return 1;
358     }
359   }
360
361   uint16_t ETH_buffer_data(SOCKET s, uint16_t offset, const uint8_t* buf, uint16_t len)
362   {
363    uint16_t ret = 0;
364     if (len > ETH_get_TX_free_size(s))
365     {
366       ret = ETH_get_TX_free_size(s);
367     }
368     else
369     {
370       ret = len;
371     }
372     ETH_send_data_processing_offset(s, offset, buf, ret);
373     return ret;
374   }
375
376   int ETH_send_UDP(SOCKET s)
377   {
378     ETH_exec_socket_cmd(s, ETH_SCR_SEND);
379
380     while ((ETH_readSnIR(s) & ETH_SIR_SEND_OK) != ETH_SIR_SEND_OK)
381     {
382       if (ETH_readSnIR(s) & ETH_SIR_TIMEOUT)
383       {
384         ETH_writeSnIR(s, (ETH_SIR_SEND_OK | ETH_SIR_TIMEOUT));
385         return 0;
386       }
387     }
388
389     ETH_writeSnIR(s, ETH_SIR_SEND_OK);
390     return 1;
391   }
```

7.3.3 w5100.h

Der Ethernet Controller selbst ist ein W5100 [14]. Dieses Headerfile enthält sämtliche Funktionsdeklarationen für diesen Controller, aber auch structs und enums welche die verschiedenen Register und Memory Bereiche spezifizieren.

Der Code wurde größtenteils von der entsprechenden Arduino Library [13] übernommen und auf C adaptiert.

Listing 20: Ethernet: W5100 Header

```
1  /*
2    Ethernet shield test program
3      Copyright (C) 2018 Andreas Mieke
4      Copyright (C) 2010 Arduino LLC
5
6      This program is free software: you can redistribute it and/or modify
7      it under the terms of the GNU General Public License as published by
8      the Free Software Foundation, either version 3 of the License, or
9      (at your option) any later version.
10
11     This program is distributed in the hope that it will be useful,
12     but WITHOUT ANY WARRANTY; without even the implied warranty of
13     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
14     GNU General Public License for more details.
15
16     You should have received a copy of the GNU General Public License
17     along with this program. If not, see <https://www.gnu.org/licenses/>.
18 */
19
20 #ifndef ___W5100_H
21 #define ___W5100_H
22
23 #include "stm32f10x.h" // Device header
24 #include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
25 #include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
26 #include "stm32f10x_spi.h" // Keil::Device:StdPeriph Drivers:SPI
27
28 #define MAX_SOCK_NUM 4
29 typedef uint8_t SOCKET;
30
31 enum {
32  ETH_MR_RST  = 0x80,
33  ETH_MR_PB   = 0x10,
34  ETH_MR_PPPoE = 0x08,
35  ETH_MR_AI   = 0x02,
36  ETH_MR_IND  = 0x01
37 };
38
39 enum {
40  ETH_IR_CONFLICT  = 0x80,
41  ETH_IR_UNREACH  = 0x40,
42  ETH_IR_PPPoE   = 0x20,
43  ETH_IR_S3_INT   = 0x08,
44  ETH_IR_S2_INT   = 0x04,
45  ETH_IR_S1_INT   = 0x02,
46  ETH_IR_S0_INT   = 0x01
47 };
48
49 enum {
50  ETH_IMR_IR7  = 0x80,
51  ETH_IMR_IR6  = 0x40,
52  ETH_IMR_IR5  = 0x20,
53  ETH_IMR_IR3  = 0x08,
```

```
54   ETH_IMR_IR2  = 0x04,
55   ETH_IMR_IR1  = 0x02,
56   ETH_IMR_IR0  = 0x01
57  };
58
59  enum {
60   ETH_SMR_MULTI  = 0x80,
61   ETH_SMR_ND   = 0x20,
62   ETH_SMR_CLOSED = 0x00,
63   ETH_SMR_TCP   = 0x01,
64   ETH_SMR_UDP   = 0x02,
65   ETH_SMR_IPRAW  = 0x03,
66   ETH_SMR_MACRAW = 0x04,
67   ETH_SMR_PPPoE  = 0x05
68  };
69
70  enum {
71   ETH_SCR_OPEN   = 0x01,
72   ETH_SCR_LISTEN  = 0x02,
73   ETH_SCR_CONNECT  = 0x04,
74   ETH_SCR_DISCON  = 0x08,
75   ETH_SCR_CLOSE   = 0x10,
76   ETH_SCR_SEND   = 0x20,
77   ETH_SCR_SEND_MAC = 0x21,
78   ETH_SCR_SEND_KEEP = 0x22,
79   ETH_SCR_RECV   = 0x40
80  };
81
82  enum {
83   ETH_SIR_SEND_OK  = 0x10,
84   ETH_SIR_TIMEOUT  = 0x08,
85   ETH_SIR_RECV   = 0x04,
86   ETH_SIR_DISCON  = 0x02,
87   ETH_SIR_CON    = 0x01
88  };
89
90  enum {
91   ETH_SSR_CLOSED    = 0x00,
92   ETH_SSR_INIT     = 0x12, // Actually 13, Cortex doesn't seem to register the LSB
93   ETH_SSR_LISTEN    = 0x14,
94   ETH_SSR_SYNSENT    = 0x15,
95   ETH_SSR_SYNRECV    = 0x16,
96   ETH_SSR_ESTABLISHED  = 0x17,
97   ETH_SSR_FIN_WAIT   = 0x18,
98   ETH_SSR_CLOSING    = 0x1A,
99   ETH_SSR_TIME_WAIT   = 0x1B,
100  ETH_SSR_CLOSE_WAIT  = 0x1C,
101  ETH_SSR_LAST_ACK   = 0x1D,
102  ETH_SSR_UDP      = 0x22,
103  ETH_SSR_IPRAW     = 0x32,
104  ETH_SSR_MACRAW    = 0x42,
105  ETH_SSR_PPPoE     = 0x5F
106 };
```

```
107
108 enum {
109  ETH_SPROTO_IP = 0,
110   ETH_SPROTO_ICMP = 1,
111   ETH_SPROTO_IGMP = 2,
112   ETH_SPROTO_GGP = 3,
113   ETH_SPROTO_TCP = 6,
114   ETH_SPROTO_PUP = 12,
115   ETH_SPROTO_UDP = 17,
116   ETH_SPROTO_IDP = 22,
117   ETH_SPROTO_ND = 77,
118   ETH_SPROTO_RAW = 255
119 };
120
121 void ETH_init(void);
122 void ETH_read_data(SOCKET s, volatile uint16_t src, volatile uint8_t * dst, uint16_t len);
123 void ETH_send_data_processing(SOCKET s, const uint8_t *data, uint16_t len);
124 void ETH_send_data_processing_offset(SOCKET s, uint16_t data_offset, const uint8_t *data,
      uint16_t len);
125 void ETH_recv_data_processing(SOCKET s, uint8_t *data, uint16_t len, uint8_t peek);
126
127 inline void ETH_set_gateway_IP(uint8_t *addr);
128 inline void ETH_get_gateway_IP(uint8_t *addr);
129
130 inline void ETH_set_subnet_mask(uint8_t *addr);
131 inline void ETH_get_subnet_mask(uint8_t *addr);
132
133 inline void ETH_set_mac(uint8_t * addr);
134 inline void ETH_get_mac(uint8_t * addr);
135
136 inline void ETH_set_IP(uint8_t * addr);
137 inline void ETH_get_IP(uint8_t * addr);
138
139 inline void ETH_set_retransmission_time(uint16_t timeout);
140 inline void ETH_set_retransmission_count(uint8_t retry);
141
142 inline void ETH_exec_socket_cmd(SOCKET s, uint8_t cmd);
143
144 uint16_t ETH_get_TX_free_size(SOCKET s);
145 uint16_t ETH_get_RX_received_size(SOCKET s);
146
147 uint8_t ETH_write_8(uint16_t addr, uint8_t data);
148 uint16_t ETH_write_n(uint16_t addr, const uint8_t *buf, uint16_t len);
149
150 uint8_t ETH_read_8(uint16_t addr);
151 uint16_t ETH_read_n(uint16_t addr, uint8_t *buf, uint16_t len);
152
153 #define ___GP_REGISTER8(name, address)          \
154   static inline void ETH_write##name(uint8_t _data) {  \
155     ETH_write_8(address, _data);            \
156   }                         \
157   static inline uint8_t ETH_read##name() {       \
158     return ETH_read_8(address);             \
```

```
159   }
160 #define ___GP_REGISTER16(name, address)         \
161   static void ETH_write##name(uint16_t _data) {     \
162     ETH_write_8(address, _data >> 8);         \
163     ETH_write_8(address+1, _data & 0xFF);         \
164   }                         \
165   static uint16_t ETH_read##name() {          \
166     uint16_t res = ETH_read_8(address);         \
167     res = (res << 8) + ETH_read_8(address + 1);     \
168     return res;                   \
169   }
170 #define ___GP_REGISTER_N(name, address, size)       \
171   static uint16_t ETH_write##name(uint8_t *_buff) {   \
172     return ETH_write_n(address, _buff, size);       \
173   }                         \
174   static uint16_t ETH_read##name(uint8_t *_buff) {   \
175     return ETH_read_n(address, _buff, size);       \
176   }
177
178 ___GP_REGISTER8 (MR, 0x0000); // Mode
179 ___GP_REGISTER_N(GAR, 0x0001, 4); // Gateway IP address
180 ___GP_REGISTER_N(SUBR, 0x0005, 4); // Subnet mask address
181 ___GP_REGISTER_N(SHAR, 0x0009, 6); // Source MAC address
182 ___GP_REGISTER_N(SIPR, 0x000F, 4); // Source IP address
183 ___GP_REGISTER8 (IR, 0x0015); // Interrupt
184 ___GP_REGISTER8 (IMR, 0x0016); // Interrupt Mask
185 ___GP_REGISTER16(RTR, 0x0017); // Timeout address
186 ___GP_REGISTER8 (RCR, 0x0019); // Retry count
187 ___GP_REGISTER8 (RMSR, 0x001A); // Receive memory size
188 ___GP_REGISTER8 (TMSR, 0x001B); // Transmit memory size
189 ___GP_REGISTER8 (PATR, 0x001C); // Authentication type address in PPPoE mode
190 ___GP_REGISTER8 (PTIMER, 0x0028); // PPP LCP Request Timer
191 ___GP_REGISTER8 (PMAGIC, 0x0029); // PPP LCP Magic Number
192 ___GP_REGISTER_N(UIPR, 0x002A, 4); // Unreachable IP address in UDP mode
193 ___GP_REGISTER16(UPORT, 0x002E); // Unreachable Port address in UDP mode
194
195 #undef ___GP_REGISTER8
196 #undef ___GP_REGISTER16
197 #undef ___GP_REGISTER_N
198
199 static inline uint8_t ETH_sock_read_8(SOCKET _s, uint16_t _addr);
200 static inline uint8_t ETH_sock_write_8(SOCKET _s, uint16_t _addr, uint8_t _data);
201 static inline uint16_t ETH_sock_read_n(SOCKET _s, uint16_t _addr, uint8_t *_buf, uint16_t
      len);
202 static inline uint16_t ETH_sock_write_n(SOCKET _s, uint16_t _addr, uint8_t *_buf, uint16_t
      len);
203
204 static const uint16_t ETH_CH_BASE = 0x0400;
205 static const uint16_t ETH_CH_SIZE = 0x0100;
206
207 #define ___SOCKET_REGISTER8(name, address)          \
208   static inline void ETH_write##name(SOCKET _s, uint8_t _data) {  \
209     ETH_sock_write_8(_s, address, _data);          \
```

```
210   }                                 \
211   static inline uint8_t ETH_read##name(SOCKET _s) {       \
212     return ETH_sock_read_8(_s, address);            \
213   }
214 #define ___SOCKET_REGISTER16(name, address)           \
215   static void ETH_write##name(SOCKET _s, uint16_t _data) {    \
216     ETH_sock_write_8(_s, address, _data >> 8);        \
217     ETH_sock_write_8(_s, address+1, _data & 0xFF);      \
218   }                                 \
219   static uint16_t ETH_read##name(SOCKET _s) {         \
220     uint16_t res = ETH_sock_read_8(_s, address);        \
221     uint16_t res2 = ETH_sock_read_8(_s,address + 1);      \
222     res = res << 8;                       \
223     res2 = res2 & 0xFF;                       \
224     res = res | res2;                     \
225     return res;                         \
226   }
227 #define ___SOCKET_REGISTER_N(name, address, size)        \
228   static uint16_t ETH_write##name(SOCKET _s, uint8_t *_buff) {   \
229     return ETH_sock_write_n(_s, address, _buff, size);      \
230   }                                 \
231   static uint16_t ETH_read##name(SOCKET _s, uint8_t *_buff) {    \
232     return ETH_sock_read_n(_s, address, _buff, size);       \
233   }
234
235 ___SOCKET_REGISTER8(SnMR, 0x0000) // Mode
236 ___SOCKET_REGISTER8(SnCR, 0x0001) // Command
237 ___SOCKET_REGISTER8(SnIR, 0x0002) // Interrupt
238 ___SOCKET_REGISTER8(SnSR, 0x0003) // Status
239 ___SOCKET_REGISTER16(SnPORT, 0x0004) // Source Port
240 ___SOCKET_REGISTER_N(SnDHAR, 0x0006, 6) // Destination Hardw Addr
241 ___SOCKET_REGISTER_N(SnDIPR, 0x000C, 4) // Destination IP Addr
242 ___SOCKET_REGISTER16(SnDPORT, 0x0010) // Destination Port
243 ___SOCKET_REGISTER16(SnMSSR, 0x0012) // Max Segment Size
244 ___SOCKET_REGISTER8(SnPROTO, 0x0014) // Protocol in IP RAW Mode
245 ___SOCKET_REGISTER8(SnTOS, 0x0015) // IP TOS
246 ___SOCKET_REGISTER8(SnTTL, 0x0016) // IP TTL
247 ___SOCKET_REGISTER16(SnTX_FSR, 0x0020) // TX Free Size
248 ___SOCKET_REGISTER16(SnTX_RD, 0x0022) // TX Read Pointer
249 ___SOCKET_REGISTER16(SnTX_WR, 0x0024) // TX Write Pointer
250 ___SOCKET_REGISTER16(SnRX_RSR, 0x0026) // RX Free Size
251 ___SOCKET_REGISTER16(SnRX_RD, 0x0028) // RX Read Pointer
252 ___SOCKET_REGISTER16(SnRX_WR, 0x002A) // RX Write Pointer (supported?)
253
254 #undef ___SOCKET_REGISTER8
255 #undef ___SOCKET_REGISTER16
256 #undef ___SOCKET_REGISTER_N
257
258 static const int ETH_SOCKETS = 4;
259 static const uint16_t ETH_SMASK = 0x07FF; // Tx buffer MASK
260 static const uint16_t ETH_RMASK = 0x07FF; // Rx buffer MASK
261 static const uint16_t ETH_SSIZE = 2048; // Max Tx buffer size
262 static const uint16_t ETH_RSIZE = 2048; // Max Rx buffer size
```

```
263
264 extern uint16_t ETH_SBASE[ETH_SOCKETS]; // Tx buffer base address
265 extern uint16_t ETH_RBASE[ETH_SOCKETS]; // Rx buffer base address
266
267 uint8_t ETH_sock_read_8(SOCKET s, uint16_t addr)
268 {
269  return ETH_read_8(ETH_CH_BASE + s * ETH_CH_SIZE + addr);
270 }
271
272 uint8_t ETH_sock_write_8(SOCKET s, uint16_t addr, uint8_t data)
273 {
274  return ETH_write_8(ETH_CH_BASE + s * ETH_CH_SIZE + addr, data);
275 }
276
277 uint16_t ETH_sock_read_n(SOCKET s, uint16_t addr, uint8_t *buf, uint16_t len)
278 {
279  return ETH_read_n(ETH_CH_BASE + s * ETH_CH_SIZE + addr, buf, len);
280 }
281
282 uint16_t ETH_sock_write_n(SOCKET s, uint16_t addr, uint8_t *buf, uint16_t len)
283 {
284  return ETH_write_n(ETH_CH_BASE + s * ETH_CH_SIZE + addr, buf, len);
285 }
286
287 void ETH_set_gateway_IP(uint8_t *addr) { ETH_writeGAR(addr); }
288 void ETH_get_gateway_IP(uint8_t *addr) { ETH_readGAR(addr); }
289
290 void ETH_set_subnet_mask(uint8_t *addr) { ETH_writeSUBR(addr); }
291 void ETH_get_subnet_mask(uint8_t *addr) { ETH_readSUBR(addr); }
292
293 void ETH_set_mac(uint8_t * addr) { ETH_writeSHAR(addr); }
294 void ETH_get_mac(uint8_t * addr) { ETH_readSHAR(addr); }
295
296 void ETH_set_IP(uint8_t * addr) { ETH_writeSIPR(addr); }
297 void ETH_get_IP(uint8_t * addr) { ETH_readSIPR(addr); }
298
299 void ETH_set_retransmission_time(uint16_t timeout) { ETH_writeRTR(timeout); }
300 void ETH_set_retransmission_count(uint8_t retry) { ETH_writeRCR(retry); }
301
302 #endif
```

7.3.4 w5100.c

Der Ethernet Controller selbst ist ein W5100 [14]. Diese Implementation enthält sämtliche Funktionen für diesen Controller.

Der Code wurde größtenteils von der entsprechenden Arduino Library [13] übernommen und auf C adaptiert.

Listing 21: Ethernet: W5100 Implementation

```
1  /*
2   Ethernet shield test program
3     Copyright (C) 2018 Andreas Mieke
4     Copyright (C) 2010 Arduino LLC
5
6     This program is free software: you can redistribute it and/or modify
7     it under the terms of the GNU General Public License as published by
8     the Free Software Foundation, either version 3 of the License, or
9     (at your option) any later version.
10
11    This program is distributed in the hope that it will be useful,
12    but WITHOUT ANY WARRANTY; without even the implied warranty of
13    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
14    GNU General Public License for more details.
15
16    You should have received a copy of the GNU General Public License
17    along with this program. If not, see <https://www.gnu.org/licenses/>.
18 */
19
20 #include "w5100.h"
21
22 uint16_t ETH_SBASE[ETH_SOCKETS]; // Tx buffer base address
23 uint16_t ETH_RBASE[ETH_SOCKETS]; // Rx buffer base address
24
25 #define TX_RX_MAX_BUF_SIZE 2048
26 #define TX_BUF 0x1100
27 #define RX_BUF (TX_BUF + TX_RX_MAX_BUF_SIZE)
28
29 #define ETH_TXBUF_BASE 0x4000
30 #define ETH_RXBUF_BASE 0x6000
31
32 void SPI_SetSS(void)
33 {
34  GPIO_WriteBit(GPIOC, GPIO_Pin_8, Bit_RESET);
35 }
36
37 void SPI_ResetSS(void)
38 {
39  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) != RESET);
40  GPIO_WriteBit(GPIOC, GPIO_Pin_8, Bit_SET);
41 }
42
43 uint8_t SPI_WriteRead(uint8_t write)
44 {
45  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
46  SPI1->DR = write;
47  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
48  return SPI1->DR;
49 }
50
51 void ETH_init(void)
```

```
52  {
53   // Port init stuff
54   // PA5 -> SCK
55   // PA6 -> MISO
56   // PA7 -> MOSI
57   // PC8 -> /SS
58
59   // Set RCC Registers for GPIOA/C
60   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
61   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
62   RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
63
64   // Init gpio structure to default values
65   GPIO_InitTypeDef gpio;
66   GPIO_StructInit(&gpio);
67
68   // Set PA5 to alternate function PushPull
69   gpio.GPIO_Mode = GPIO_Mode_AF_PP;
70   gpio.GPIO_Pin = GPIO_Pin_5;
71   GPIO_Init(GPIOA, &gpio);
72
73   // Set PA7 to alternate function push pull
74   gpio.GPIO_Pin = GPIO_Pin_7;
75   GPIO_Init(GPIOA, &gpio);
76
77   // Set PC8 to push pull
78   gpio.GPIO_Mode = GPIO_Mode_Out_PP;
79   gpio.GPIO_Pin = GPIO_Pin_8;
80   GPIO_Init(GPIOC, &gpio);
81   GPIO_WriteBit(GPIOC, GPIO_Pin_8, Bit_SET);
82
83   // Set PA6 to input floating
84   gpio.GPIO_Mode = GPIO_Mode_IN_FLOATING;
85   gpio.GPIO_Pin = GPIO_Pin_6;
86   GPIO_Init(GPIOA, &gpio);
87
88   // Init SPI engine
89   // Init struct to default values
90   SPI_InitTypeDef spi;
91   SPI_StructInit(&spi);
92
93   // We are master
94   spi.SPI_Mode = SPI_Mode_Master;
95   spi.SPI_NSS = SPI_NSS_Soft;
96   //spi.SPI_CPHA = SPI_CPHA_2Edge;
97   spi.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_16;
98
99   // Write the registers
100  SPI_Init(SPI1, &spi);
101
102  // Enable SPI1
103  SPI_Cmd(SPI1, ENABLE);
104
```

```
105   for(uint16_t i = 0; i < 65535; i++);
106
107   ETH_writeMR(ETH_MR_RST);
108   ETH_writeRMSR(0x55);
109   ETH_writeTMSR(0x55);
110
111   for (int i=0; i<MAX_SOCK_NUM; i++) {
112       ETH_SBASE[i] = ETH_TXBUF_BASE + ETH_SSIZE * i;
113       ETH_RBASE[i] = ETH_RXBUF_BASE + ETH_RSIZE * i;
114     }
115  }
116
117  uint8_t ETH_write_8(uint16_t addr, uint8_t data)
118  {
119   SPI_SetSS();
120   SPI_WriteRead(0xF0);
121   SPI_WriteRead(addr >> 8);
122   SPI_WriteRead(addr & 0xFF);
123   SPI_WriteRead(data);
124   SPI_ResetSS();
125   return 8;
126  }
127
128  uint16_t ETH_write_n(uint16_t addr, const uint8_t *buf, uint16_t len)
129  {
130   for (uint16_t i = 0; i < len; i++) {
131    SPI_SetSS();
132    SPI_WriteRead(0xF0);
133    SPI_WriteRead(addr >> 8);
134    SPI_WriteRead(addr & 0xFF);
135    addr++;
136    SPI_WriteRead(buf[i]);
137    SPI_ResetSS();
138   }
139   return len;
140  }
141
142  uint8_t ETH_read_8(uint16_t addr)
143  {
144   uint8_t data;
145   SPI_SetSS();
146   SPI_WriteRead(0x0F);
147   SPI_WriteRead(addr >> 8);
148   SPI_WriteRead(addr & 0xFF);
149   data = SPI_WriteRead(0x00);
150   SPI_ResetSS();
151   return data;
152  }
153
154  uint16_t ETH_read_n(uint16_t addr, uint8_t *buf, uint16_t len)
155  {
156   for (uint16_t i = 0; i < len; i++) {
157    SPI_SetSS();
```

```
158     SPI_WriteRead(0x0F);
159     SPI_WriteRead(addr >> 8);
160     SPI_WriteRead(addr & 0xFF);
161     addr++;
162     buf[i] = SPI_WriteRead(0x00);
163     SPI_ResetSS();
164    }
165    return len;
166  }
167
168  uint16_t ETH_get_TX_free_size(SOCKET s)
169  {
170   uint16_t val=0, val1=0;
171    do {
172      val1 = ETH_readSnTX_FSR(s);
173      if (val1 != 0)
174        val = ETH_readSnTX_FSR(s);
175    }
176    while (val != val1);
177    return val;
178  }
179
180  uint16_t ETH_get_RX_received_size(SOCKET s)
181  {
182   uint16_t val=0,val1=0;
183    do {
184      val1 = ETH_readSnRX_RSR(s);
185      if (val1 != 0)
186        val = ETH_readSnRX_RSR(s);
187    }
188    while (val != val1);
189    return val;
190  }
191
192  void ETH_read_data(SOCKET s, volatile uint16_t src, volatile uint8_t * dst, uint16_t len)
193  {
194   uint16_t size;
195    uint16_t src_mask;
196    uint16_t src_ptr;
197
198    src_mask = src & ETH_RMASK;
199    src_ptr = ETH_RBASE[s] + src_mask;
200
201    if( (src_mask + len) > ETH_RSIZE )
202    {
203      size = ETH_RSIZE - src_mask;
204      ETH_read_n(src_ptr, (uint8_t *)dst, size);
205      dst += size;
206      ETH_read_n(ETH_RBASE[s], (uint8_t *) dst, len - size);
207    }
208    else
209      ETH_read_n(src_ptr, (uint8_t *) dst, len);
210  }
```

```
211
212 void ETH_send_data_processing(SOCKET s, const uint8_t *data, uint16_t len)
213 {
214  ETH_send_data_processing_offset(s, 0, data, len);
215 }
216
217 void ETH_send_data_processing_offset(SOCKET s, uint16_t data_offset, const uint8_t *data,
        uint16_t len)
218 {
219  uint16_t ptr = ETH_readSnTX_WR(s);
220    ptr += data_offset;
221    uint16_t offset = ptr & ETH_SMASK;
222    uint16_t dstAddr = offset + ETH_SBASE[s];
223
224    if (offset + len > ETH_SSIZE)
225    {
226      uint16_t size = ETH_SSIZE - offset;
227      ETH_write_n(dstAddr, data, size);
228      ETH_write_n(ETH_SBASE[s], data + size, len - size);
229    }
230    else {
231      ETH_write_n(dstAddr, data, len);
232    }
233
234    ptr += len;
235    ETH_writeSnTX_WR(s, ptr);
236 }
237
238 void ETH_recv_data_processing(SOCKET s, uint8_t *data, uint16_t len, uint8_t peek)
239 {
240  uint16_t ptr;
241    ptr = ETH_readSnRX_RD(s);
242    ETH_read_data(s, ptr, data, len);
243    if (!peek)
244    {
245      ptr += len;
246      ETH_writeSnRX_RD(s, ptr);
247    }
248 }
249
250 void ETH_exec_socket_cmd(SOCKET s, uint8_t cmd)
251 {
252  ETH_writeSnCR(s, cmd);
253    while(ETH_readSnCR(s)) {
254    // Wait for command to be executed
255   }
256 }
```

7.4 Keil μVision 5

Zur Programmierung des neuen Minimalsystems wurde die integrierte Entwicklungsumgebung (IDE) Keil μVision 5 verwendet. Da sich diese erheblich von der Version 4 unterscheidet, und das Projekt weiters auch im Unterricht verwendet werden solle, wurde eine Anleitung für eben diese neue Version 5 der IDE verfasst, welche alle Schritte von der Installation bis zum Debugging erklärt und demonstriert. Weiters wurde der Debugging-Adapter ausgetauscht, anstelle eine Keil Elektronik GmbH (Keil) ULINK/ME kommt nun standardmäßig ein ST-Link zum Einsatz.

In den nun folgenden Kapiteln wurde dieses Tutorial, in leicht abgewandelter Form, übernommen, das Originaldokument kann unter [15] gefunden werden.

7.4.1 Einführung

7.4.1.1 Warum der Umstieg zu μVision 5?

In der HTBL Hollabrunn wurde in den letzten Jahren laufend die Version 4 der IDE μVision verwendet, ein Umstieg auf die neuere Version 5 war weder nötig noch wirklich sinnvoll.

Allerdings hat die μVision in der Version 4 einen großen Nachteil, welcher die Verwendbarkeit in der Zukunft stark einschränkt. Denn ein kompilieren von Programmen für Cortex-M4 oder höher ist bei dieser Version nicht möglich, und Version 5 wird zur zwingenden Vorraussetzung. Da die Entwicklung weiter voran schreitet, ist es für die HTBL Hollabrunn nicht mehr praktikabel die veraltete Version 4 einzusetzen.

Dieses Dokument wird kurz auf die Mindestanforderungen der neuen Software, und des Weiteren auf die Inbetriebnahme mittels einfacher Beispielprogramme eingehen. Alte μVision 4 Projekte können auf diese Weise in die neue Umgebung übertragen werden.

7.4.1.2 Mindestsystemanforderungen

	Minimum	Empfohlen
Prozessor	1 GHz (32/64 bit)	2 GHz (64 bit) oder mehr
RAM	1 GB	4 GB oder mehr
Festplattenspeicher	2 GB	5 GB oder mehr
Internet		2 Mb/s oder mehr (für Pack Installer)

Tabelle 30: Systemanforderungen der Keil μVision 5

Alle Windows Versionen ab Windows Vista (32/64 bit) werden unterstützt.

☞ Achtung: Im Gegensatz zu μVision in Version 4, wird von dieser Version das Betriebssystem Microsoft Windows XP nicht mehr unterstützt!

7.4.2 Das erste μVision 5 Projekt

7.4.2.1 Die Installation

Bevor mit der eigentlichen Installation der IDE begonnen werden kann, muss diese von der offiziellen Keil Webseite heruntergeladen werden. Dies kann unter diesem Link getan werden: `https://www.keil.com/demo/eval/arm.htm`

Abbildung 151: Der heruntergeladene μVision 5 Installer

Nachdem der Installer heruntergeladen wurde, sollte eine ausführbare Datei wie in Abbildung 151 zu sehen ist vorhanden sein, eventuell sollte die Größe dieser Datei überprüft werden, um auszuschließen, dass es beim Download zu einem Fehler kam.

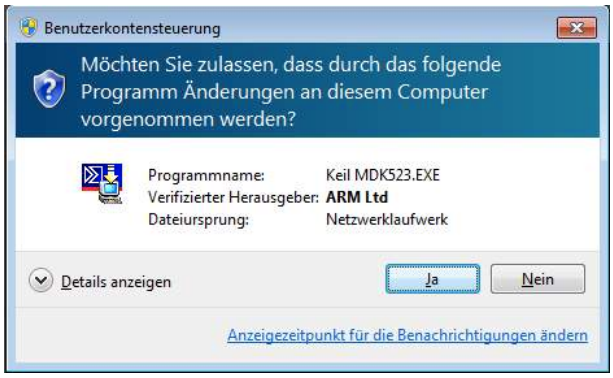


Abbildung 152: Dialog von Windows User Account Control

Danach muss dieser mit einem Doppelklick gestartet werden. Eventuell zeigt Windows einen Bestätigungsdialog (Abbildung 152) an, dieser ist mit einem Klick auf den Button `Ja` zu bestätigen.

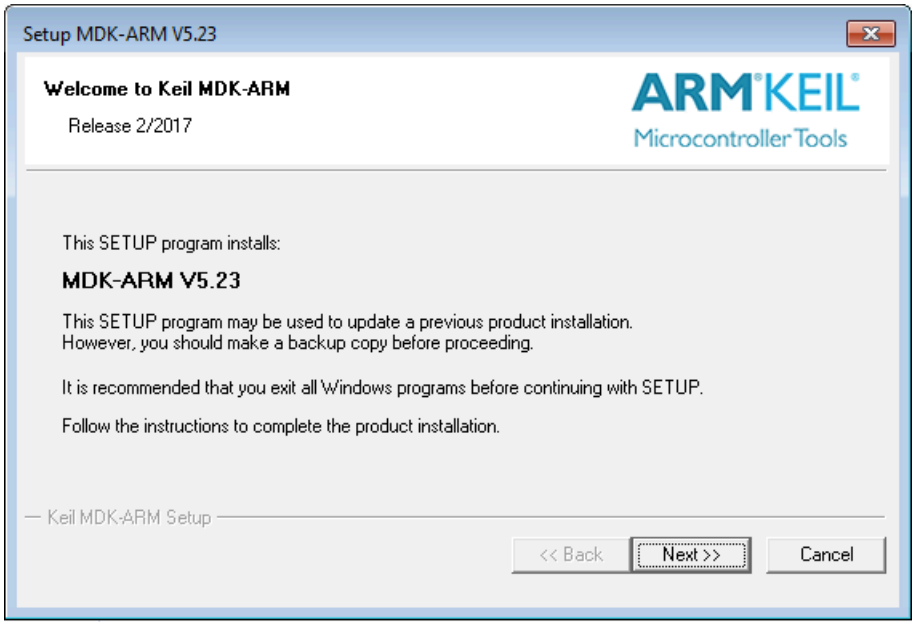


Abbildung 153: Begrüßungsbildschirm des Installers

Wenn der Dialog bestätigt wurde, startet der eigentliche Installationsprozess. Im Begrüßungsbildschirm (Abbildung 153) wird nochmals erläutert welche Version der Software zur Zeit installiert wird (in diesem Fall μVision in Version `5.24`). Die Richtigkeit dieser Angaben wird mit einem Klick auf `Next` bestätigt.

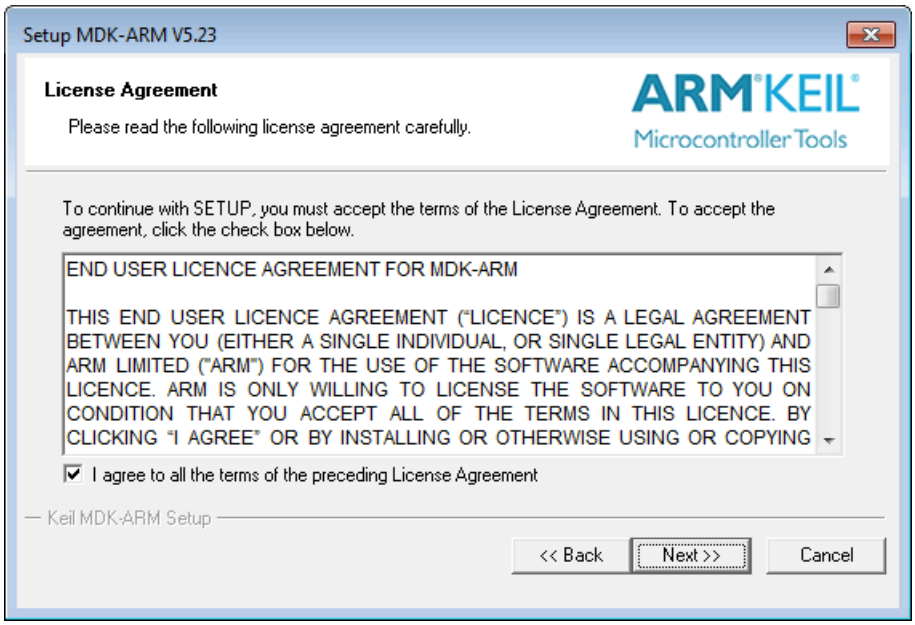


Abbildung 154: Lizenzbedingungen

Nun müssen die Lizenzbedingungen akzeptiert werden (Abbildung 154), hierzu muss der Hacken in der Checkbox gesetzt werden und wieder mit einem Klick auf `Next` bestätigt werden.

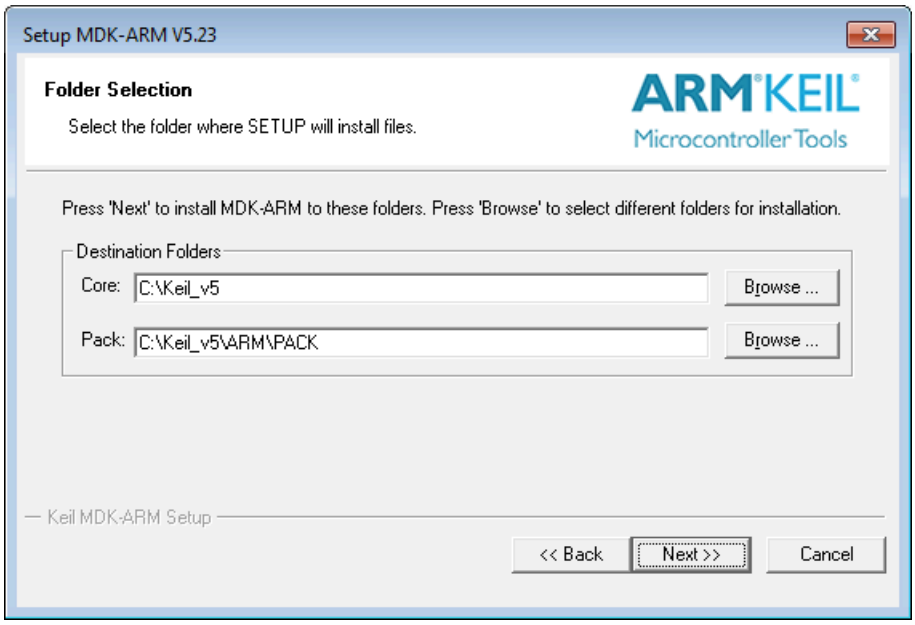


Abbildung 155: Auswahl der Installationspfade

Im nächsten Bildschirm (Abbildung 155) wird der Installationsort für den Compiler und

die IDE, sowie der Pfad für die Installation von Cortex Microcontroller Software Interface Standard (CMSIS)-Packs (Abbildung 162) abgefragt. Grundsätzlich können beide Felder auf beliebige Pfade gesetzt werden, aufgrund der Kompatibilität und der vereinfachten Fehlersuche wird aber empfohlen den Standard beizubehalten.

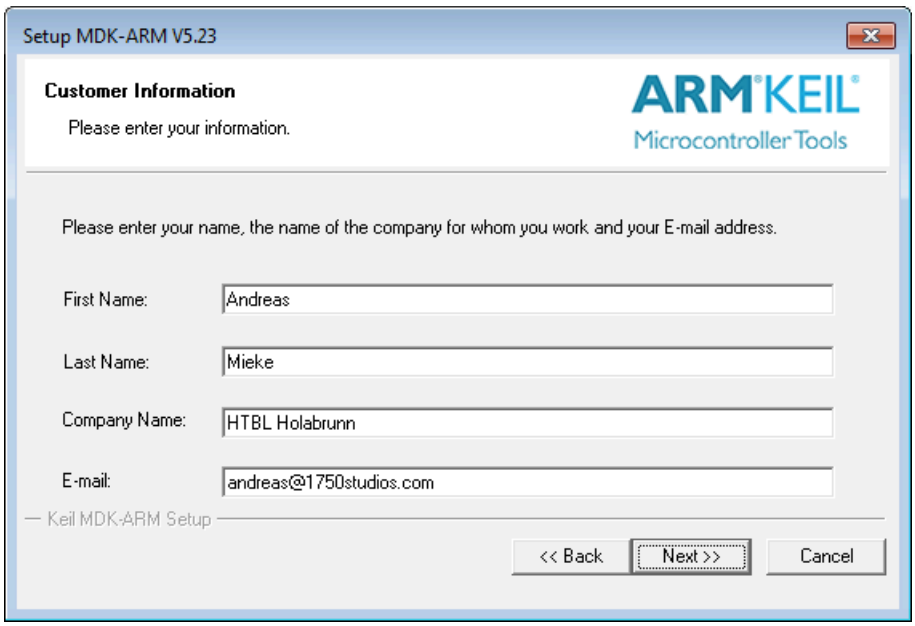


Abbildung 156: Eingabe der Benutzerdaten

Danach werden Daten zum Benutzer abgefragt (Abbildung 156). Diese Felder können entweder mit erfundenen Daten, oder – wenn man später ggf. eine Lizenz hinzufügen will – mit den realen Daten des Benutzers gefüllt werden.

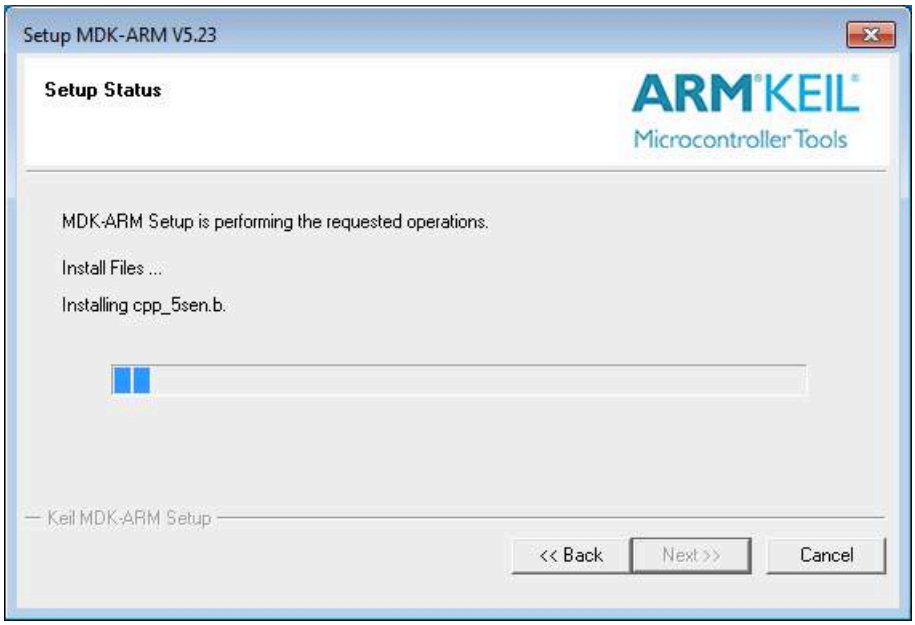


Abbildung 157: Installationsfortschritt

Jetzt wird mit der eigentlichen Installation der Dateien begonnen. Der Fortschritt wird in einem eigenen Bildschirm (Abbildung 157) angezeigt. Nun muss gewartet werden, bis der Balken komplett durchgelaufen ist.

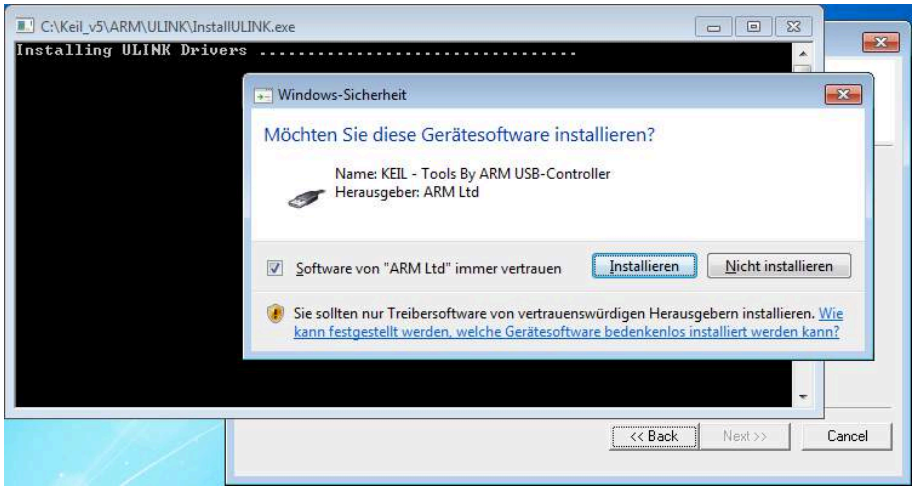


Abbildung 158: Warnung der Windows-Sicherheit

Je nach Windows Version kann ein Konsolenfenster auf gehen, welches einfach ignoriert werden kann. Gegebenenfalls wird auch eine Warnung der Windows-Sicherheit bezüglich der Installation von Treibern angezeigt (Abbildung 158). Diese ist mit einem Klick auf `Installieren` zu bestätigen.

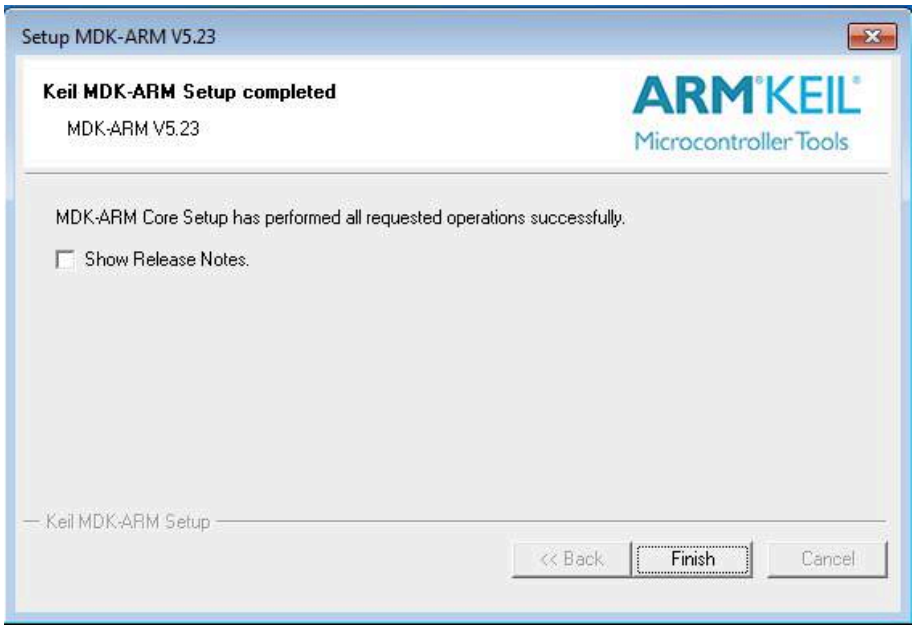


Abbildung 159: Erfolgreiche Installation

Am letzten Bildschirm (Abbildung 159) wird nachgefragt ob die Release Notes angezeigt werden sollen, dies ist nicht nötig und somit sollte in der Checkbox auch kein Hacken sein. Mit einem klick auf den `Finish` Button wird die Installation beendet. Ein Neustart des Rechners sollte nicht nötig sein.

7.4.2.2 Der Pack Installer

Nachdem die Installation erfolgreich beendet wurde, startet der `Pack Installer` der μVision. Dieses Programm verwaltet alle CMSIS-Pakete welche im Laufe der Verwendung der IDE heruntergeladen und installiert werden. Der Installer hat im groben zwei Spalten, auf der linken Seite kann man den Prozessor, welchen man verwenden will, aussuchen und auf der rechten Seite werden dann die für diesen Prozessor verfügbaren Pakete angezeigt.

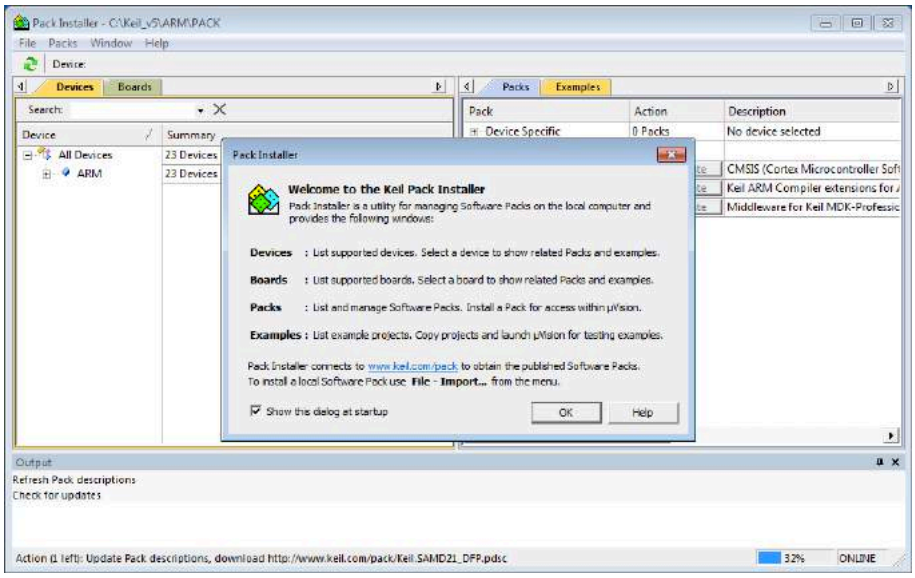


Abbildung 160: Der Pack Installer nach dem ersten Start

Beim ersten Start des Pack Installers wird eine Willkommensnachricht, welche diesen kurz beschreibt, angezeigt (Abbildung 160). Diese kann entweder durch deaktivieren der entsprechenden Checkbox für immer versteckt, oder mittels Button geschlossen werden.

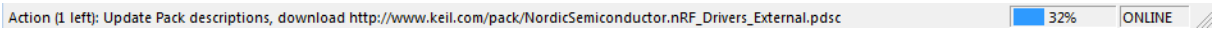


Abbildung 161: Downloadfortschritt

Beim ersten Start sind nur die direkt von ARM ausgelieferten Prozessorkerne im Installer verfügbar (Abbildung 160), allerdings wird gleichzeitig auch ein Updateprozess gestartet, welcher alle nötigen Paket-Infos herunterlädt und gegebenenfalls bereits installierte Pakete updatet. Um den Installer richtig verwenden zu können, müssen wir dieses erste Update abwarten. In der Statusleiste des Programms (Abbildung 161) sieht man den entsprechenden Fortschritt. Wenn dieser auf 100% steigt, beziehungsweise der Text komplett verschwindet ist das Update beendet und wir können fortfahren.

☞ Zu beachten: Es kann vorkommen, dass die Fortschrittsanzeige beim installieren der einzelnen Pakete kurz verschwindet, es sollte also zur Sicherheit einige Sekunden gewartet werden, wenn der Text verschwindet um sicher zu gehen, dass das Update auch wirklich fertig eingespielt wurde.

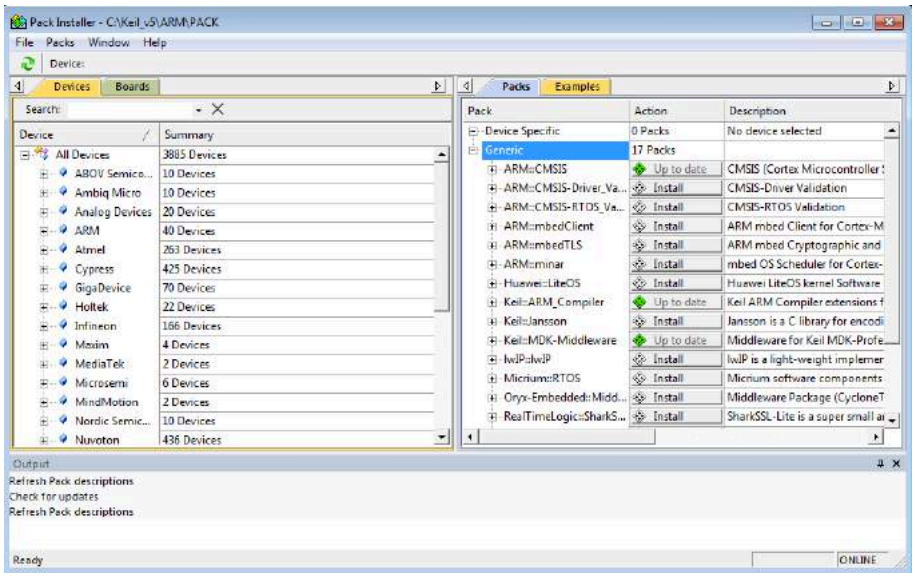


Abbildung 162: Verfügbare Pakete

Nachdem die Updates erfolgreich eingespielt wurden sollte sich die gerade noch leere Liste (Abbildung 162) mit knapp 4000 verfügbaren Prozessoren gefüllt haben.

Abbildung 163: Prozessor des Minimalsystems

Danach muss der passende Prozessor ausgewählt werden (Abbildung 163). Im Zuge dieser Diplomarbeit wurde der Prozessor für den Schulgebrauch von `STM32F103RB` zu einem `STM32F107RCT(6)` geändert, dieser bietet mehr Features als der alte Prozessor. Man kann den richtigen Prozessor entweder über die Liste auswählen, oder einfach das Suchfeld oben links verwenden um direkt den Richtigen angezeigt zu bekommen.

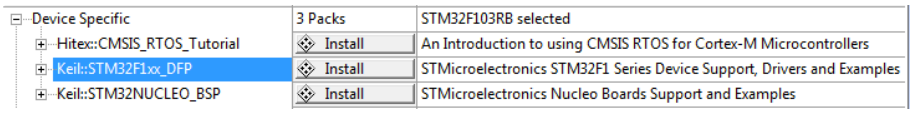


Abbildung 164: Verfügbare Pakete für den Prozessor des Minimalsystems

Auf der rechten Seite (Abbildung 164) scheinen, sobald der richtige Prozessor ausgewählt ist, die für diesen Prozessor verfügbaren CMSIS-Pakete auf, diese können nun mittels Klick auf den entsprechenden Button installiert und danach in der IDE verwendet werden. Für den Schulgebrauch ist das Paket `Keil::STM32F1xx_DFP` nötig und ausreichend. Dieses beinhaltet alle verwendeten Libraries und auch Beispielprogramme. Während der

Installation des Pakets ist wieder auf die Statusleiste und den Fortschritt in dieser zu achten, wenn alles erfolgreich installiert wurde, sollte unser verwendeter Prozessor auf der linken Übersichtsseite grün hinterlegt sein, siehe dazu Abbildung 165. Des weiteren sollten sämtliche Packs, bei welchen `Update` steht geupdatet werden.

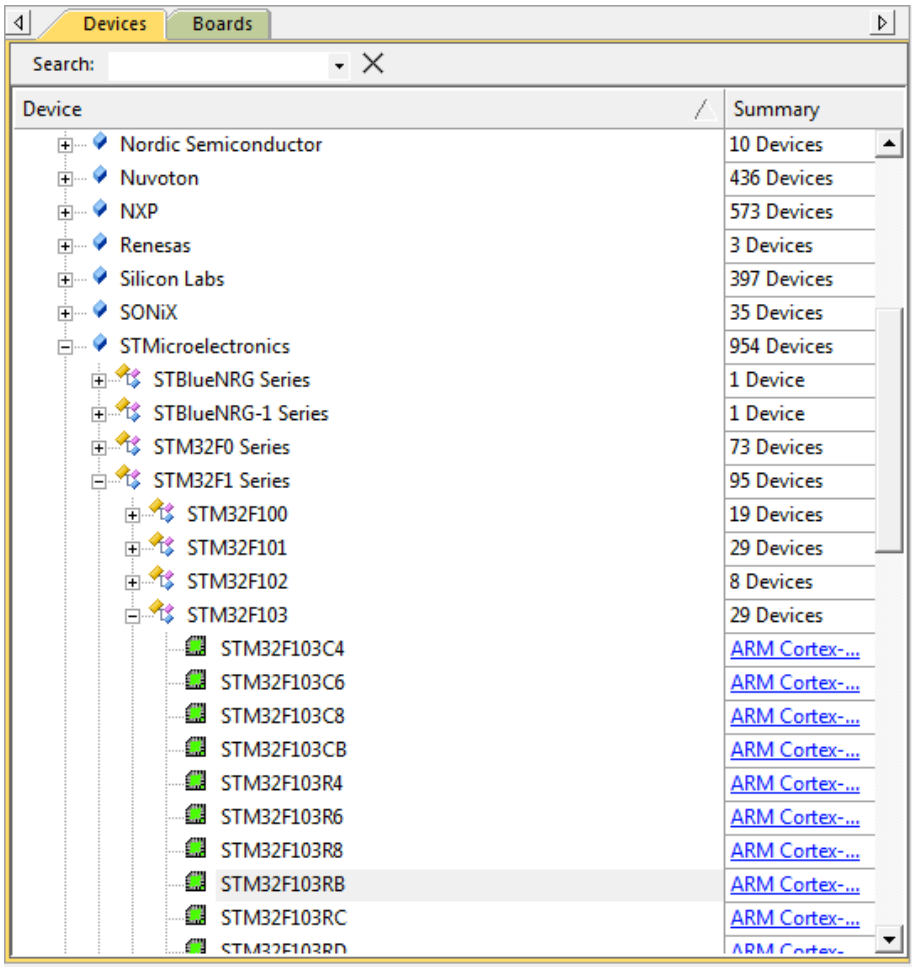


Abbildung 165: Erfolgreich installierter Prozessor

7.4.2.3 Installation des HTBL Packs

Um die spezifischen Libraries und Header Files der HTBL Hollabrunn einfach zur Verfügung zu stellen, gibt es für eben diese ein eigenes CMSIS-Pack. Dieses wird aber anders als die anderen Packs (noch) nicht über das Internet vertrieben, sondern vom Lehrer auf einem Medium wie USB-Stick oder CD ausgeteilt. Dementsprechend muss dieses Pack manuell in den Pack-Manager hinzugefügt werden.

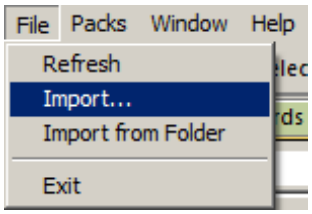


Abbildung 166: Menüpunkt zum manuellen Import von Packs

Hierzu muss auf `File`, und dann auf `Import...` geklickt werden, wie in Abbildung 166 dargestellt. Danach öffnet sich ein Explorer Fenster, in welchem man die entsprechende `.pack`-Datei auswählen muss. Wenn dies geschehen ist wird die Datei eingelesen und automatisch installiert. Sollte aus irgendeinem Grund die Zieldatei schon existieren, so ist das überschreiben mittels Klick auf `Ja` im entsprechenden Dialogfenster zu bestätigen. Nach erfolgter Installation sollte das Übersichtsfenster wie in Abbildung 167 aussehen.

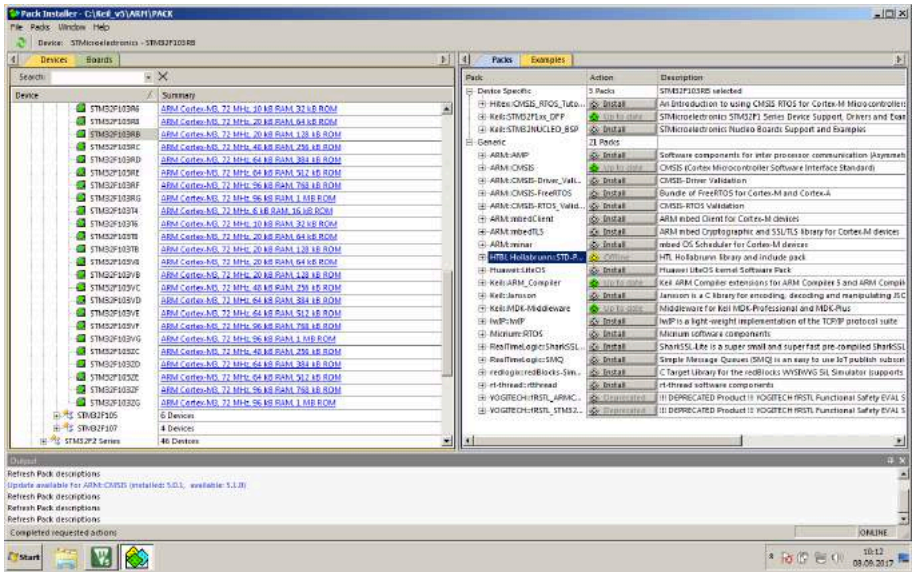


Abbildung 167: Hauptfenster nach erfolgreicher Installation des HTBL Packs

7.4.3 Die Projekterstellung

Als nächstes muss ein μVision Projekt erstellt werden. Projekte dienen zur Organisation der Source-Files und der verwendeten Bibliotheken, sowohl CMSIS- als auch eigene Bibliotheken. Das Projekt kann mittels grafischem Interface einfach konfiguriert und mit CMSIS-Libraries versehen werden.

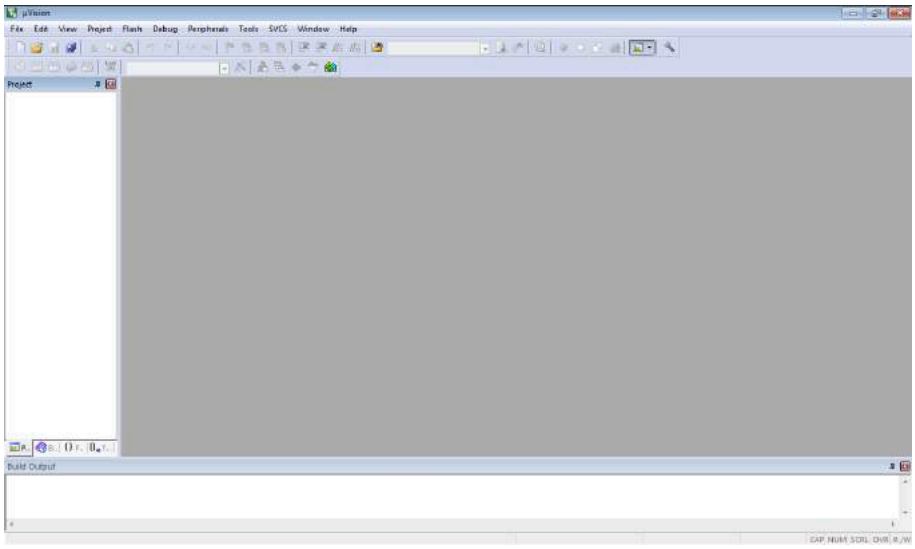


Abbildung 168: Hauptfenster der IDE

Nach dem Start der IDE ist zunächst ein leeres Fenster zu sehen, siehe dazu Abbildung 168. Dieses besteht aus einer Menüleiste ganz oben und den Schnellzugriffsschaltflächen direkt darunter. Auf der linken Seite ist ein noch leerer Projektbaum zu finden und rechts im großen grauen Feld der eigentliche Texteditor für die Source-Files. Darunter befindet sich noch ein Fenster für Log und Compiler Ausgaben, unter diesem die Statusleiste.

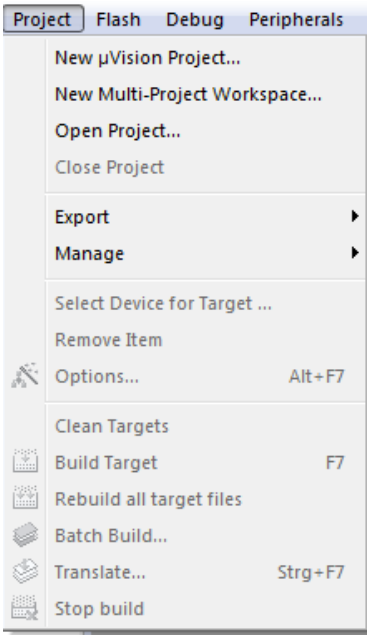


Abbildung 169: Projekt-Menü

Mit einem Klick auf den entsprechenden Menüpunkt (Abbildung 169) kann ein neues μVision Projekt erstellt werden.

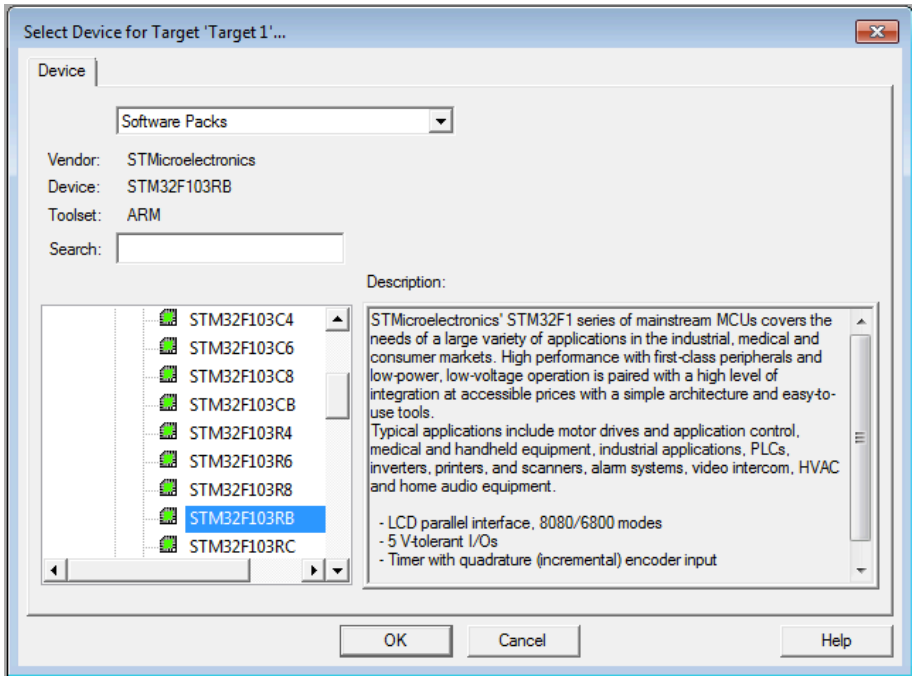


Abbildung 170: Prozessorauswahldialog

Im nächsten Bildschirm (Abbildung 170) wird abgefragt welcher Prozessor verwendet werden soll, in unserem Fall ist dies wieder der `STM32F107RCT(6)`. Man kann den richtigen Prozessor entweder in der Liste heraus suchen, oder den Namen direkt in das Suchfeld eingeben. Auf der rechten Seite wird ein Beschreibungstext mit den Features des gewählten Prozessors angezeigt.

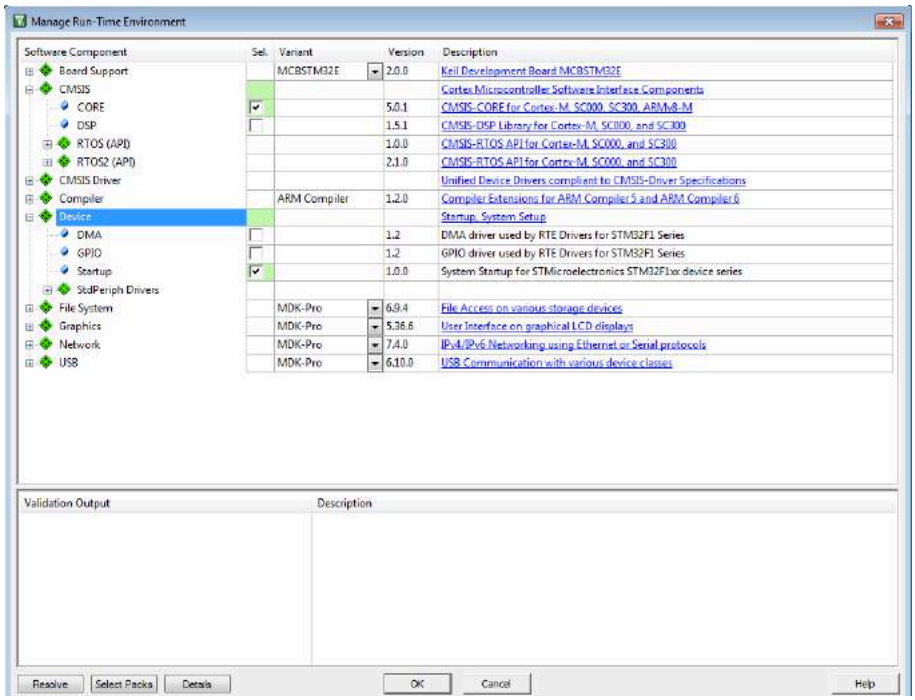


Abbildung 171: Laufzeitumgebungskonfigurationsfenster

Danach wird gefragt welche CMSIS-Libraries eingebunden werden sollen (Abbildung 171), hier ist mindestens `CORE` im Reiter `CMSIS` und `Startup` im Reiter `Device` zu wählen. Werden mehr Features benötigt, können diese in den entsprechenden Reitern aktiviert werden. Ein späteres Ändern der Laufzeitumgebung ist über einen Klick auf die entsprechende Schnellzugriffsschaltfläche ohne größere Umstände möglich. Die Konfiguration ist mittels Klick auf `OK` zu bestätigen.

☞ Zu beachten: Während die meisten CMSIS-Libraries frei verfügbar sind, gibt es auch einige wenige (File System, Graphics, Network, USB) welche nur mit einer Pro Version von μVision verwendbar sind. Ist dies der Fall muss auf freie Libraries ausgewichen, oder eine Pro Version erworben werden.

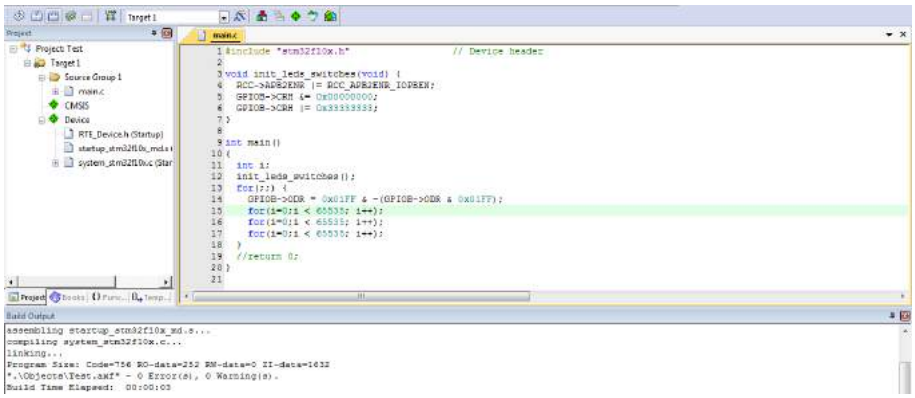


Abbildung 172: Beispielprogramm

In Abbildung 172 ist ein kleines Beispielprogramm zu sehen, welches die Schalter der Basisplatine einliest und die LEDs dementsprechend leuchten lässt. Links zu sehen ist der Projektbaum. C-Files werden über den entsprechenden Menüpunkt (Rechtsklick auf die `Source Group 1` und dann `Add Item to Group...`) angelegt und direkt in das Projekt eingebunden. Der passende Dateiname und Typ ist im folgenden Fenster (Abbildung 173) entsprechend zu wählen. Das Listing 22 kann einfach in das C-File kopiert werden.

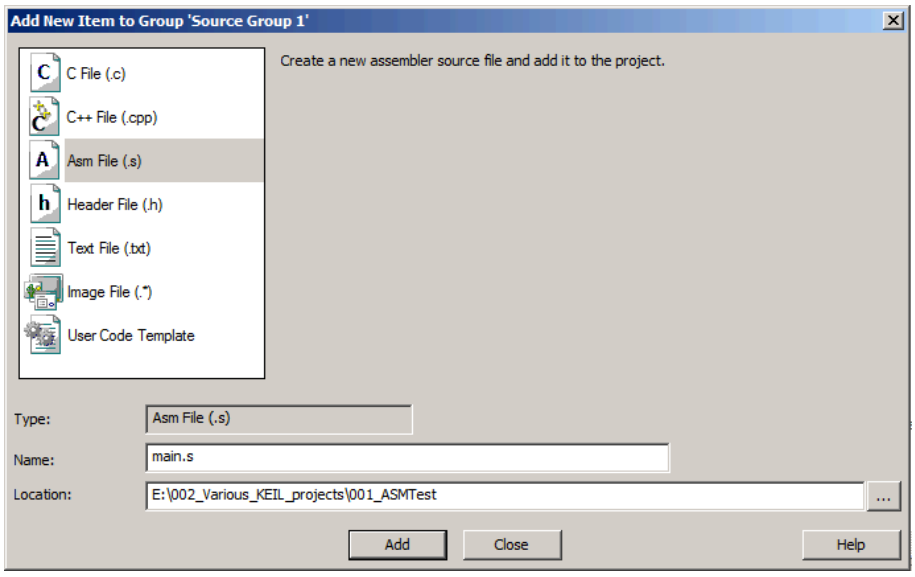


Abbildung 173: Dateierstellungsdialog

Listing 22: LED/Schalter Test

```
1  #include "stm32f10x.h" // Device header
2  #include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
3  #include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
4
5  int main()
```

```
6  {
7   uint8_t data;
8
9   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
10  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
11
12  GPIO_InitTypeDef gpio;
13  GPIO_StructInit(&gpio);
14
15  gpio.GPIO_Mode = GPIO_Mode_Out_PP;
16  gpio.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 |
        GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
17  GPIO_Init(GPIOC, &gpio);
18
19  gpio.GPIO_Mode = GPIO_Mode_IPU;
20  gpio.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_5 |
        GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8;
21  GPIO_Init(GPIOA, &gpio);
22
23  for(;;) {
24   data = (GPIO_ReadInputData(GPIOA) & 0x000F) | ((GPIO_ReadInputData(GPIOA) & 0x01E0)
         >> 1);
25   GPIO_Write(GPIOC, ((data & 0xE0) << 2) | ((data & 0x1F) << 1));
26  }
27 }
```

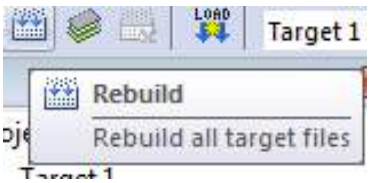


Abbildung 174: Schaltfläche zum kompilieren

Als nächstes muss das geschriebene Programm kompiliert werden, dazu ist einfach auf die Schaltfläche (Abbildung 174) zu klicken. Nach einigen wenigen Sekunden sollte dieser Vorgang erfolgreich abgeschlossen sein (Ausgabe im Logfenster (Übersicht: Abbildung 168) beachten).

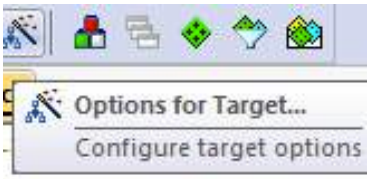


Abbildung 175: Optionsschaltfläche

Bevor nun aber das fertig kompilierte Programm auf das Minimalsystem geflasht werden kann, muss der Debugging Adapter[1] angeschlossen (Abbildung 176) werden. Dies kann mittels Klick auf den Zauberstab (Abbildung 175) kann der richtige Debugger ausgewählt werden.

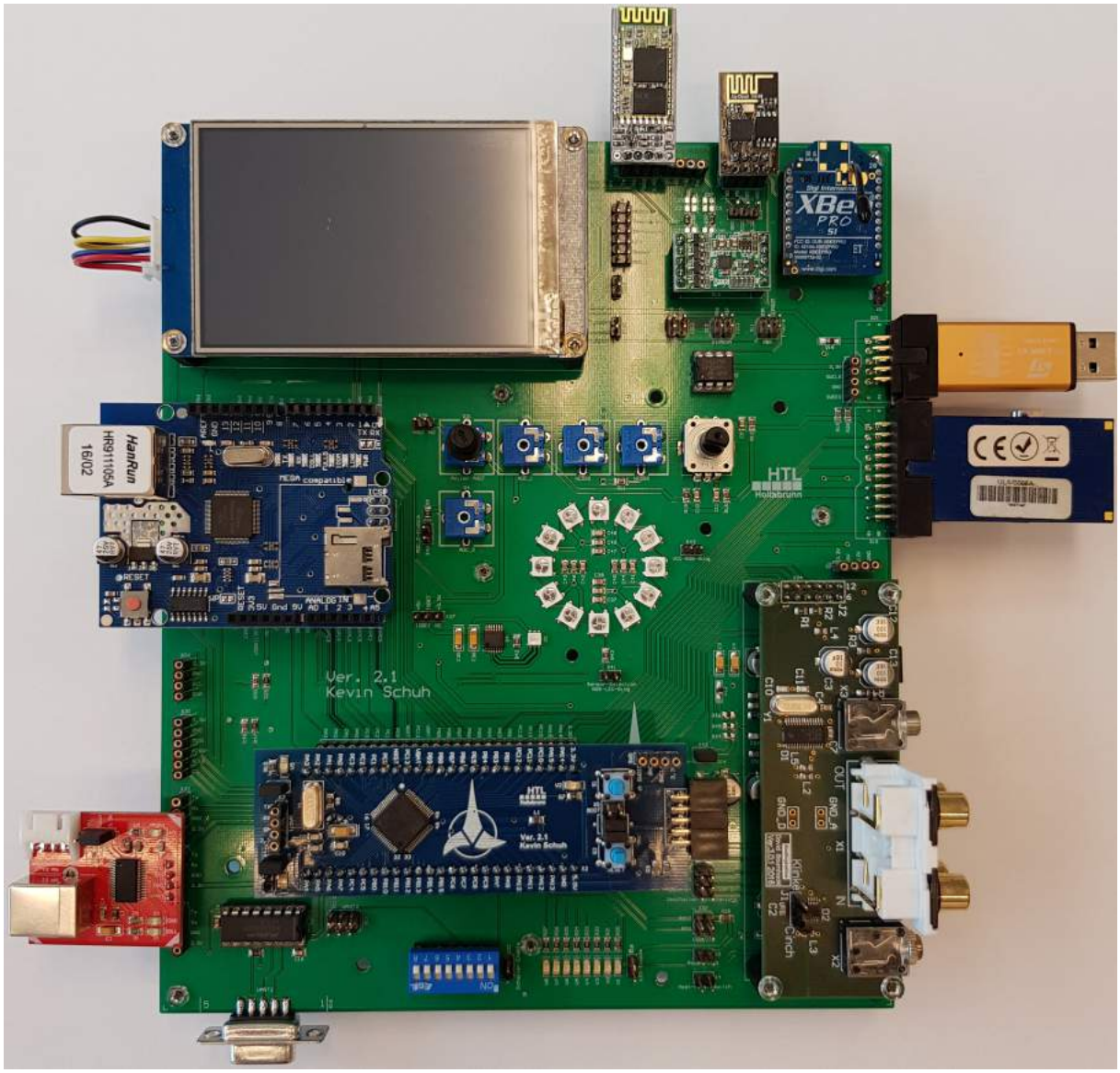


Abbildung 176: Aufbau des Minimalsystems

[1] In unserem Fall ein ST Link V2

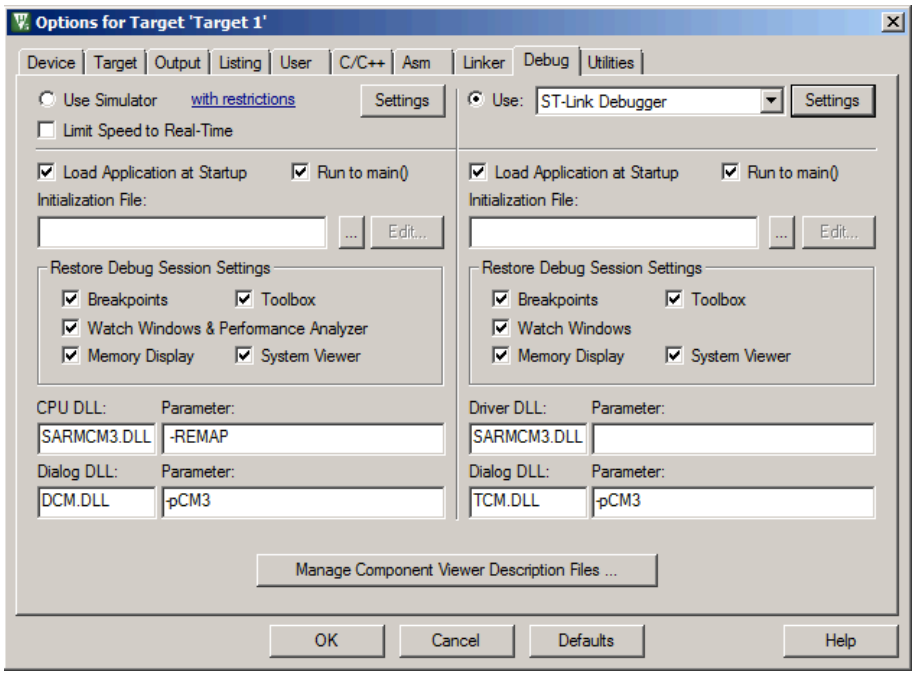


Abbildung 177: Optionsfenster

Danach geht das in Abbildung 177 abgebildete Fenster auf. Eventuell steht die Auswahl nicht auf dem Debug Reiter oben, dann ist dies manuell mittels Mausklick durchzuführen. Auf der rechten Seite muss der Radio Button bei `Use` ausgewählt werden und im Dropdown daneben `ST-Link Debugger`.

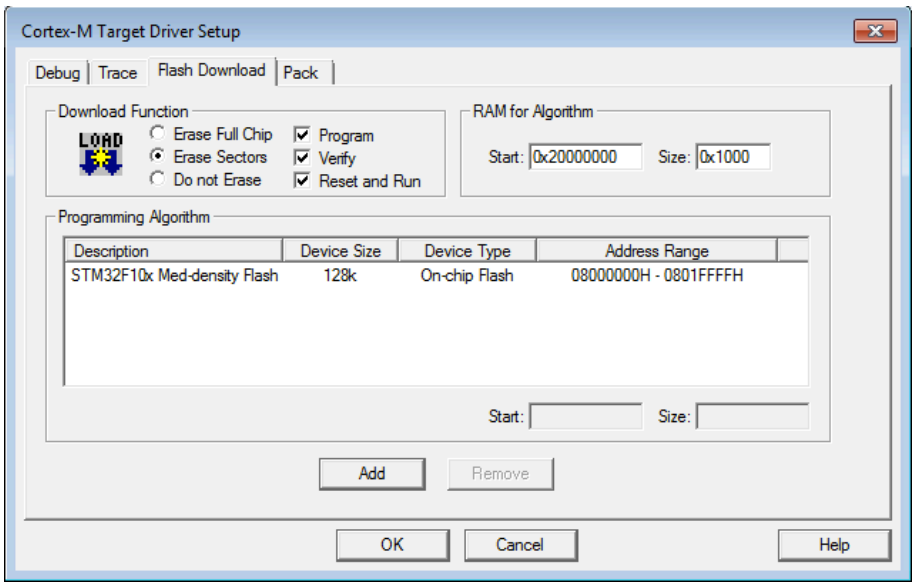


Abbildung 178: Debugger Einstellungen

Danach klickt man auf `Settings` rechts daneben. Im nun auf gegangenen Fenster (Abbildung 178) sollte die Checkbox `Reset and Run` aktiviert werden. Dies ist zwar für den korrekten Betrieb nicht nötig, erleichtert aber das Arbeiten sehr, da nicht nach jedem mal neu flashen der Reset Knopf des Prozessors gedrückt werden muss.

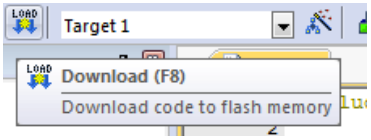


Abbildung 179: Load Button

Nun kann das kompilierte Programm mittels des Load Buttons (Abbildung 179) auf den Zielprozessor geladen werden. Wenn die Einstellungen, welche bei Abbildung 178 zu sehen sind richtig angewendet wurden, sollte nun die LEDs der Basisplatine entsprechend den Schalterpositionen leuchten. Damit ist die μVision IDE in der Version 5 fertig eingerichtet und bereit verwendet zu werden!

7.4.4 Debugging

Im nächsten Teil dieses Tutorials wird auf den Debugger der Keil μVision 5 eingegangen werden, und des weiteren auch erläutert, wie sich eben dieser vom Debugger der μVision 4 unterscheidet.

In der Symbolleiste (Abbildung 180) befinden sich entsprechende Icons zum starten des Debuggers und um Breakpoints zu verwalten. Mit Hilfe dieser Buttons kann der Debugger gestartet und gestoppt werden (Lupe ganz links) und Breakpoints hinzugefügt (roter Punkt), aktiviert und deaktiviert (weißer Punkt) werden. Des weiteren können alle Breakpoints gleichzeitig mittels Klick auf das Icon mit den zwei weißen Kreisen mit rotem Rand deaktiviert werden. Der Button rechts daneben löscht alle Breakpoints komplett.

Abbildung 180: Debugging-Symbolleiste

Wird der Debugger nun mittels Klick auf die Lupe in Abbildung 180 aktiviert, so wird das aktuelle Programm neu auf den Prozessor geflasht und gestartet, gleichzeitig erweitert sich auch das User Interface der IDE (Abbildung 181) um für das Debugging relevante Fenster. Weiters wird, wenn kein Breakpoint gesetzt ist, das Programm bis zum `main`-Aufruf ausgeführt und dort angehalten.

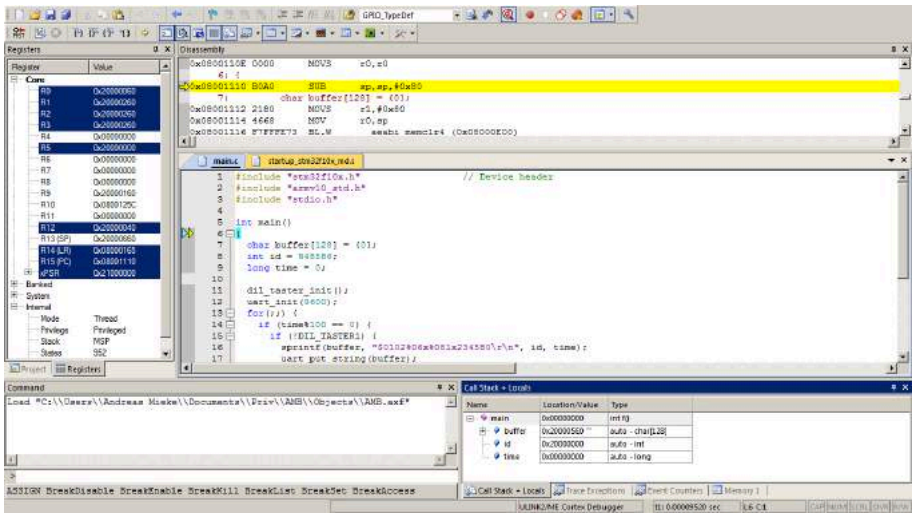


Abbildung 181: User Interface der Keil μVision 5 im Debugging Modus

Im Register Fenster (Abbildung 182) sind alle Register des Microcontrollers zu sehen, zusätzlich dazu sind jene, welche sich seit dem letzten Step verändert haben, blau eingefärbt. Das Register Fenster ist für das Debugging von C/C++ Programmen nicht wirklich sinnvoll, für Assembler-Programme aber eine große Hilfe.

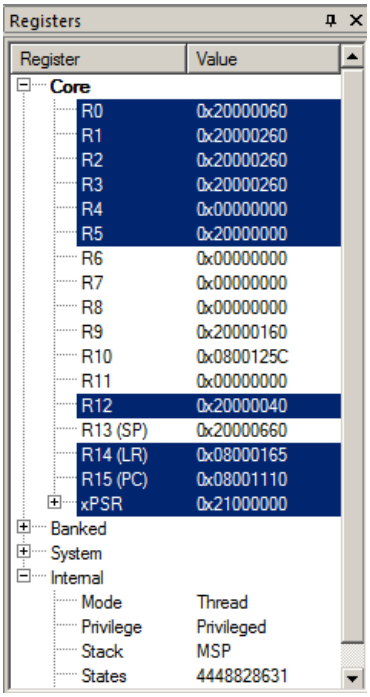


Abbildung 182: Register Fenster

Ein weiteres Fenster ist das Disassembly-Fenster, welches den zur Zeit ausgeführten Binär-

code in Form von Assemblerbefehlen zeigt, siehe Abbildung 183. Dies ist sinnvoll, wenn eine binäre Library analysiert und auf Fehler geprüft werden muss. Des weiteren kann man hier gut sehen bei welchem Teil einer Libraryfunktion das Programm zum Beispiel abstürzt oder hängen bleibt.

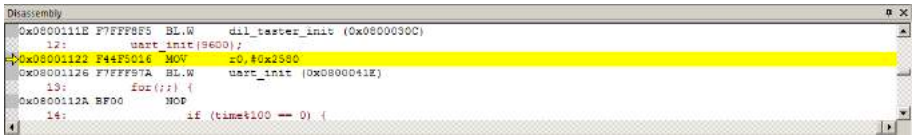


Abbildung 183: Disassembly Fenster

Im Hauptfenster (Abbildung 184) wird der zur Zeit ausgeführte Programmcode angezeigt, mit zwei Pfeilen auf der Seite, welche die aktuelle Position und die Cursor-Position anzeigen. Der Cursor-Pfeil macht Sinn, wenn man die Funktion „Run to Line“ benutzen will. Dies ist der zweite Button von rechts in Abbildung 185.

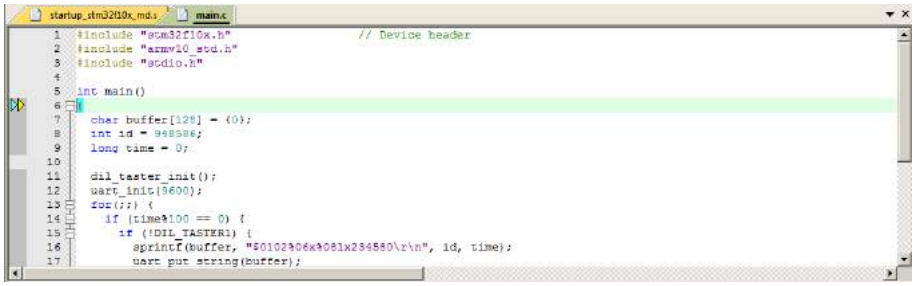


Abbildung 184: Hauptfenster

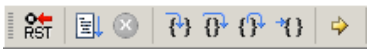


Abbildung 185: Stepping-Buttons

Des weiteren kann durch einen Klick auf die graue Fläche neben den Zeilennummern für die entsprechende Zeile ein Breakpoint aktiviert (und deaktiviert) werden. Siehe dazu Abbildung 186. Ein Breakpoint hält die Ausführung der laufenden Anwendung an dieser Stelle an und setzt diese erst nach einem Klick auf Run (Abbildung 185, zweiter Button von links) oder auf einen der Stepping-Buttons (zweite Buttongruppe von rechts) wieder im entsprechenden Modus fort. Dies ist sehr sinnvoll um sich zum Beispiel den Inhalt von Registern anzusehen und so Fehler zu suchen.

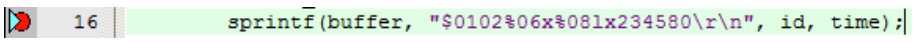


Abbildung 186: Programmzeile mit Breakpoint und Ausführungs-Pfeil

Neben den Hauptregistern selbst, Abbildung 182, kann im Debugger auch die Konfiguration von einzelnen Peripherieeinheiten angesehen werden. Hierfür gibt es jeweils Extra Fenster, welche über das Hauptmenü geöffnet werden kann, zu sehen in Abbildung 187, in diesem Beispiel sehen wir uns die Konfiguration von GPIOA an.

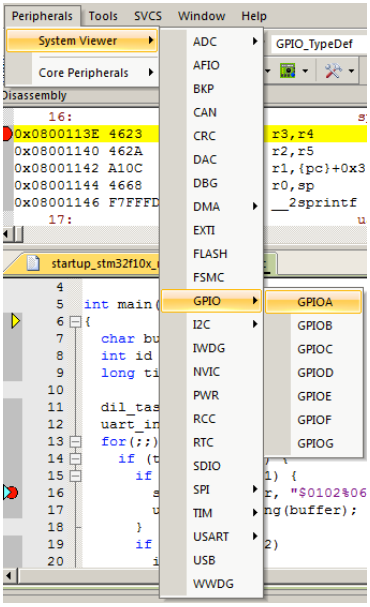


Abbildung 187: Peripherie-Hauptmenü

Nach dem Klick auf den entsprechenden Menüeintrag öffnet sich ein weiteres Fenster, Abbildung 188, entweder rechts angedockt, oder frei über der IDE liegend. Sollte es rechts angedockt sein, empfehle ich das Fenster raus zu ziehen und etwas größer zu machen, da dies der Übersicht hilft.

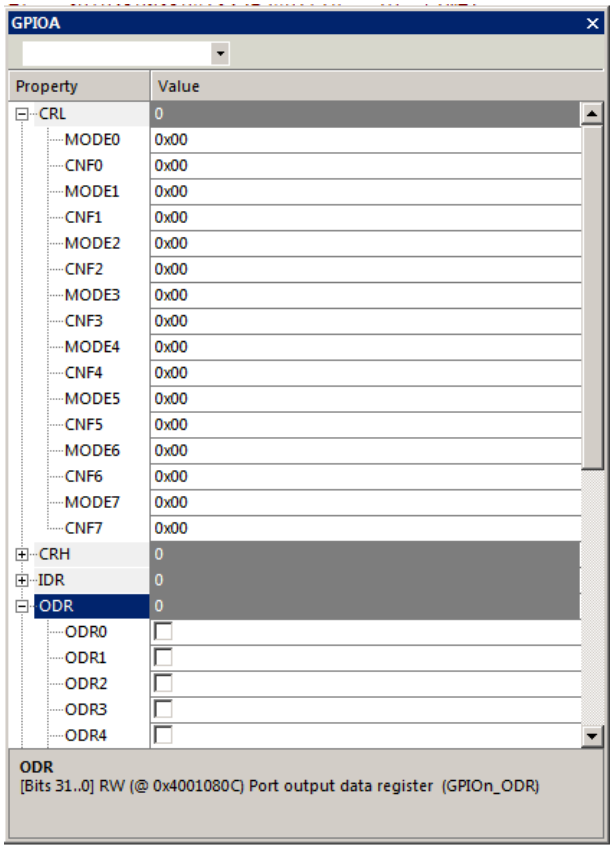


Abbildung 188: GPIOA Registerfenster

Mit Hilfe dieses Fensters kann nun die Konfiguration der entsprechenden Peripherieeinheit angesehen werden. Anders als bei Keil μVision 4, werden in diesem Fenster nur noch die Konfigurationswerte in Hexadezimal dargestellt, beziehungsweise binäre Werte in Form von Checkboxen. Dies kann leider auch nach Nachfrage beim Keil Support nicht umgestellt werden, was das einzige Manko der μVision 5 gegenüber der Version 4 ist.

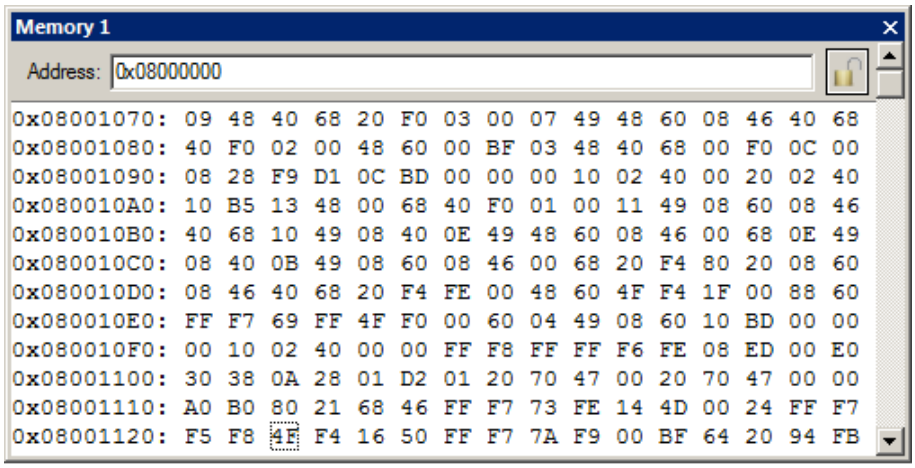


Abbildung 189: Memory Fenster

Eine weitere Möglichkeit des Debuggings ist es, das Memory Fenster zu benutzen. Dieses Fenster (Abbildung 189) zeigt den Inhalt des Speichers ab einer eingegebenen Adresse an. Hier zu sehen ist die Instruktion, welche gerade in Abbildung 183 ausgeführt wird. Weiters ist es Möglich mittels Rechtsklick den Inhalt von Speicherbereichen zu ändern. Dies ist insbesondere dann sinnvoll, wenn man den Op-Code einer Instruktion oder ähnliches während das Programm läuft ändern möchte.

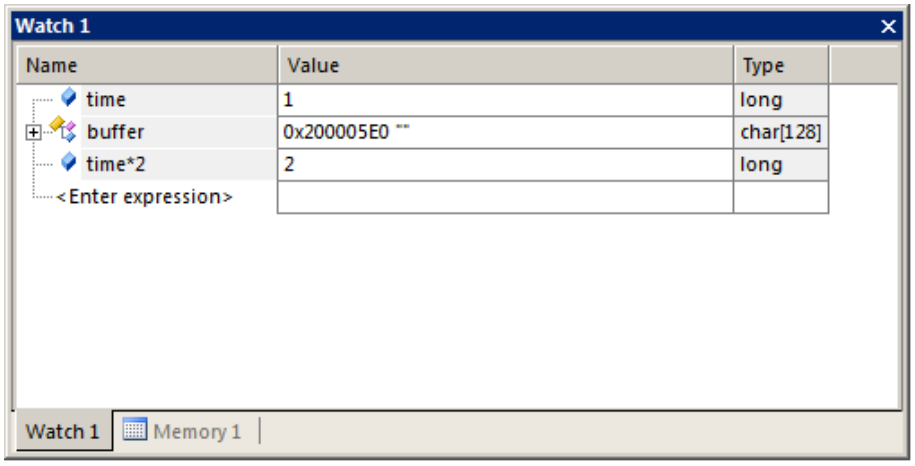


Abbildung 190: Watches Fenster

Mit Hilfe des Watches Fenster (Abbildung 190) können Variablen im C Programm „beobachtet“ werden. Das Fenster zeigt, sofern auf die Variable an der entsprechenden Programmstelle zugegriffen werden kann, immer den aktuellen Wert eben dieser an. Auch hier ist es möglich den Wert in Echtzeit zu ändern, außerdem können einfache Berechnungen durchgeführt werden.

7.5 CMSIS-Packs

Libraries und Beispielprojekte für die μVision 5 werden in sogenannten CMSIS-Packs verwaltet. CMSIS-Packs sind ZIP-Dateien, welche eine Beschreibungs-Datei im `.pdsc`-Format enthält. Dies ist intern eine Extensible Markup Language (XML)-Datei, welche den Inhalt und die Abhängigkeiten eines CMSIS-Packs beschreibt.

Im Zuge dieser Diplomarbeit entstand ein CMSIS-Pack für die HTL HTL Standard Library (STDLib) und einige Assembler Helper-Dateien, welche für das alte Minimalsystem verwendet werden. Dies macht das Arbeiten mit der alten Platform in Verbindung mit μVision 5 einfacher als dies mit der Version 4 war. Des weiteren bietet diese Pack-Strukturierung eine einfache Möglichkeit des Updatens von Libraries, was mit der μVision 4 in dieser Form überhaupt nicht möglich war.

7.5.1 Die Erstellung

Die Erstellung eines minimalen CMSIS-Packs ist nicht allzu schwer und in einigen Minuten erledigt. Die Erstellung von komplexeren Packs mit Abhängigkeiten, Beispielprojekten und so weiter kann unter [16] eingesehen werden. Im folgenden Beispiel wird das STDLib-Pack erstellt.

☞ Hinweis: Hier wird nur auf die Erstellung des Packs an sich, nicht aber auf die Erstellung der im Pack inkludierten Libraries eingegangen.

7.5.1.1 Inhalt

Der Inhalt des STDLib CMSIS-Packs kann in Abbildung 191 gesehen werden.

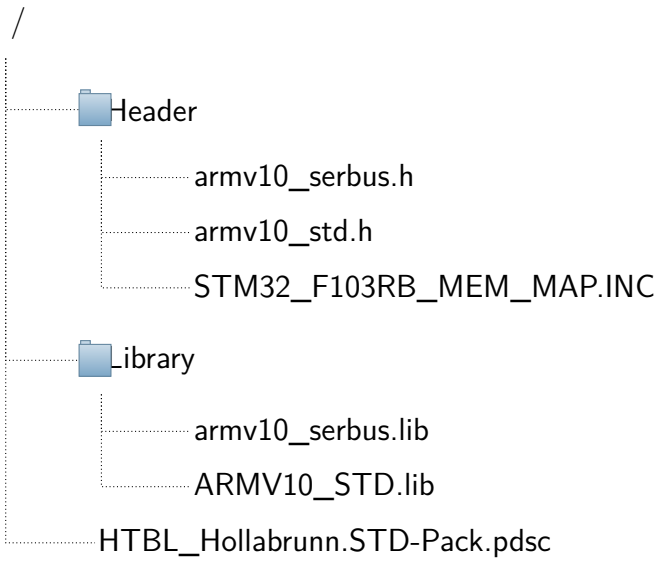


Abbildung 191: Inhalt des STDLib CMSIS-Packs

Das Pack beinhaltet somit Header-Files, genauer das für die STDLib und für eine serielle Bus Library, im Unterordner `Header`. Des weiteren wurden in diesem Ordner auch Include-Files für die Assembler-Programmierung abgelegt. Dies ist zwar laut offizieller Dokumentation (siehe: [16]) nicht der passende Ort für solche Dateien, wurde aber zur Einfachkeit trotzdem so gewählt. Daneben gibt es noch das Verzeichnis `Library`, in welchem die kompilierten Bibliotheken gespiechert werden. Alle Versionen und Abhängigkeiten werden zentral im `.pdsc`-File im Wurzelverzeichnis verwaltet, der Inhalt dieses Files kann in Listing 23 eingesehen werden.

Listing 23: Inhalt des PDSC-Files der STDLib

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <package schemaVersion="1.4" xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
       xs:noNamespaceSchemaLocation="PACK.xsd">
3      <vendor>HTL Hollabrunn</vendor>
4      <name>STD-Pack</name>
5      <description>HTL Hollabrunn library and include pack</description>
6      <url></url>
7      <supportContact></supportContact>
8
9      <releases>
10         <release version="1.0.0" date="2017-07-16">
11             Initial version -- Mieke
12         </release>
13         <release version="2.0.0" date="2017-07-20">
14             Major version push -- REJ
15         </release>
16         <release version="2.1.0" date="2018-04-02">
17             Removing Start Up code as it is no longer needed, due to code which exists within the
                   CMSIS packs -- Mieke
```

```
18          </release>
19      </releases>
20
21      <keywords>
22          <keyword>HTL Hollabrunn</keyword>
23          <keyword>STD Library</keyword>
24          <keyword>Serbus Library</keyword>
25      </keywords>
26
27      <components>
28          <component Cclass="HTL Hollabrunn" Cgroup="Libraries" Csub="STDLib" Cversion="
                2.0.0">
29              <description>Standard library for the HTL</description>
30              <files>
31                  <file category="header" name="Header/armv10_std.h"/>
32                  <file category="library" name="Library/ARMV10_STD.lib"/>
33              </files>
34          </component>
35          <component Cclass="HTL Hollabrunn" Cgroup="Libraries" Csub="SerbusLib" Cversion=
                "1.0.0" condition="STDLib">
36              <description>Serial bus library for the HTL</description>
37              <files>
38                  <file category="header" name="Header/armv10_serbus.h"/>
39                  <file category="library" name="Library/armv10_serbus.lib"/>
40              </files>
41          </component>
42          <component Cclass="HTL Hollabrunn" Cgroup="Assembler" Csub="Includes" Cversion="
                2.0.0">
43              <description>Assembler includes for the HTL</description>
44              <files>
45                  <file category="header" name="Header/STM32_F103RB_MEM_MAP.INC"/>
46              </files>
47          </component>
48      </components>
49
50      <conditions>
51          <condition id="STDLib">
52              <description>Standard library</description>
53              <require Cclass="HTL Hollabrunn" Cgroup="Libraries" Csub="STDLib" />
54          </condition>
55      </conditions>
56  </package>
```

Die Attribute der einzelnen Komponenten werden in der offiziellen Dokumentation genauer erläutert, die Abbildung 192 stellt nur eine kurze Übersicht dar. Neben dem Namen und der Beschreibung gibt die `PDSC`-Datei auch an welche Datei zu welchem Software Pack gehört, welche Version eines Packs zur Verfügung gestellt wird und welche Abhängigkeiten bestehen (zum Beispiel hängt die Serbus Library von der STDLib ab).

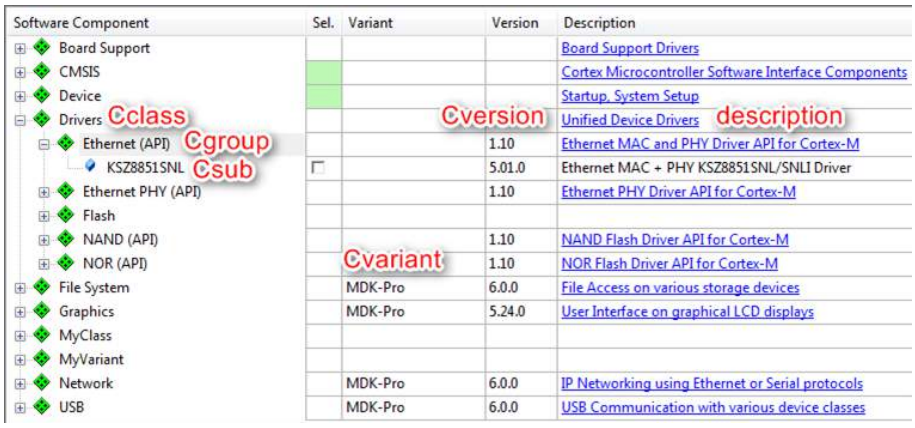


Abbildung 192: Attribute von CMSIS-Packs mit dem Namen aus der PDSC-Datei. Nach: [16]

☞ Hinweis: Bei der Versionierung sollte auf Semantic Versioning zurückgegriffen werden, da sich dieses Verfahren im Bereich von Software und dessen Abhängigkeiten am besten bewährt hat.

7.5.1.2 Erstellung

Wenn dann der Inhalt fertig ist, also alle Files am richtigen Ort liegen und richtig im PDSC-File eingetragen sind, kann das eigentlich Pack erstellt werden. Hierzu wurde ein kurzes Shellskript (Listing 24) erstellt, welches nichts anderes macht als 7-Zip aufzurufen und damit ein ZIP-File mit der Endung `.pack` erzeugt. Das Skript liest die Werte für den Packnamen nicht aus dem PDSC-File aus, sondern vergibt diesen immer fest. Dies sollte also angepasst werden, wenn das Skript produktiv eingesetzt wird.

Listing 24: 7-Zip Aufruf

```
1 #!/bin/bash
2 cd Files
3 7z a "../HTBL_Hollabrunn.STD-Pack.2.1.0.pack" * -tzip
4 cd ..
```

7.5.1.3 Installation

Nachdem das Pack erfolgreich erstellt wurde, sollte es Testweise im Pack-Manager, siehe Abschnitt 7.4.2.3, installiert werden um zu überprüfen, ob alles richtig erstellt wurde. Ist die Installation erfolgreich, sollte sich ein Bild wie in Abbildung 193 ergeben.

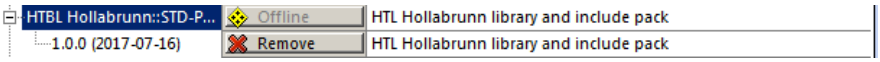


Abbildung 193: Das HTL CMSIS-Pack nach erfolgreicher Installation

Nach erfolgreicher Installation kann man wie gewünscht die Teile des Packs aktivieren, welche man für sein Projekt braucht, siehe Abbildung 194.

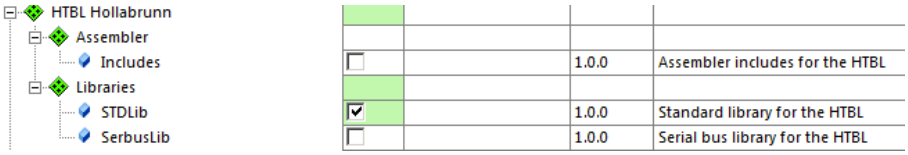


Abbildung 194: Das HTL CMSIS-Pack mit selektierten Paketteilen

Wenn man nun mit dieser Auswahl ein neues μVision 5 Projekt erstellt, sieht das Paket im Projektbaum aus, wie in Abbildung 195 zu sehen ist.

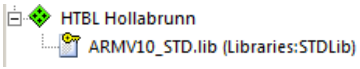


Abbildung 195: Das HTL CMSIS-Pack mit selektierten Paketteilen in einem μVision 5 Projekt

8 Z80 Minimalsystem

8.1 Projektidee

8.1.1 Warum ein Z80 Minimalsystem?

Das Z80 Minimalsystem wurde für die Anwendung im Laborunterricht entworfen, wo es gemeinsam mit einem Logikanalysator eingesetzt wird, um Bustimings des Z80 zu erfassen und zu analysieren. Die Wahl fiel auf den Z80, da der Befehlssatz und der Aufbau der „von-Neumann-Architektur“ bereits durch den Theorieunterricht bekannt ist und die Möglichkeit der Aufzeichnung der Timings der systeminternen Busse besteht, da es sich um ein Mikroprozessorsystem und nicht um einen Mikrocontroller handelt.

Das Lehrsystem Microprofessor µPF 1, welches im DIC-Unterricht (Digitale Systeme) verwendet wird, arbeitet ebenfalls mit einer Z80 CPU und stellt die Grundlage für das Z80 Minimalsystem dar. Der Lerncomputer µPF 1 ist auf eine möglichst einfache Programmierbarkeit ausgelegt und eignet sich deshalb bestens für das Testen neu entwickelter Programme. Hingegen sind im Laborunterricht kurze Vorbereitungszeiten gefragt, weshalb das Minimalsystem keine direkte Programmiermöglichkeit wie der µPF 1 besitzt. Der µPF 1 verfügt über eine Tastatur und ein Display, um mithilfe eines Monitorprogramms Programme in den SRAM zu schreiben, sie auszuführen und sie zu debuggen. Beim Z80 Minimalsystem wird das Programm auf einem PC erstellt und übersetzt und mithilfe eines Programmiergeräts fix in einem EPROM abgelegt. Weiters verfügt das Minimalsystem im Gegensatz zum µPF 1 über einen DMA Controller für direkte Zugriffe der Peripherie auf den Speicher und über eine RS232-Schnittstelle. Um ein Bustiming aufzeichnen zu können, sind Datenbus, Adressbus und Steuerbus extra an Stiftleisten herausgeführt.

8.2 Aufbau

8.2.1 Blockschaltbild des Gesamtsystems

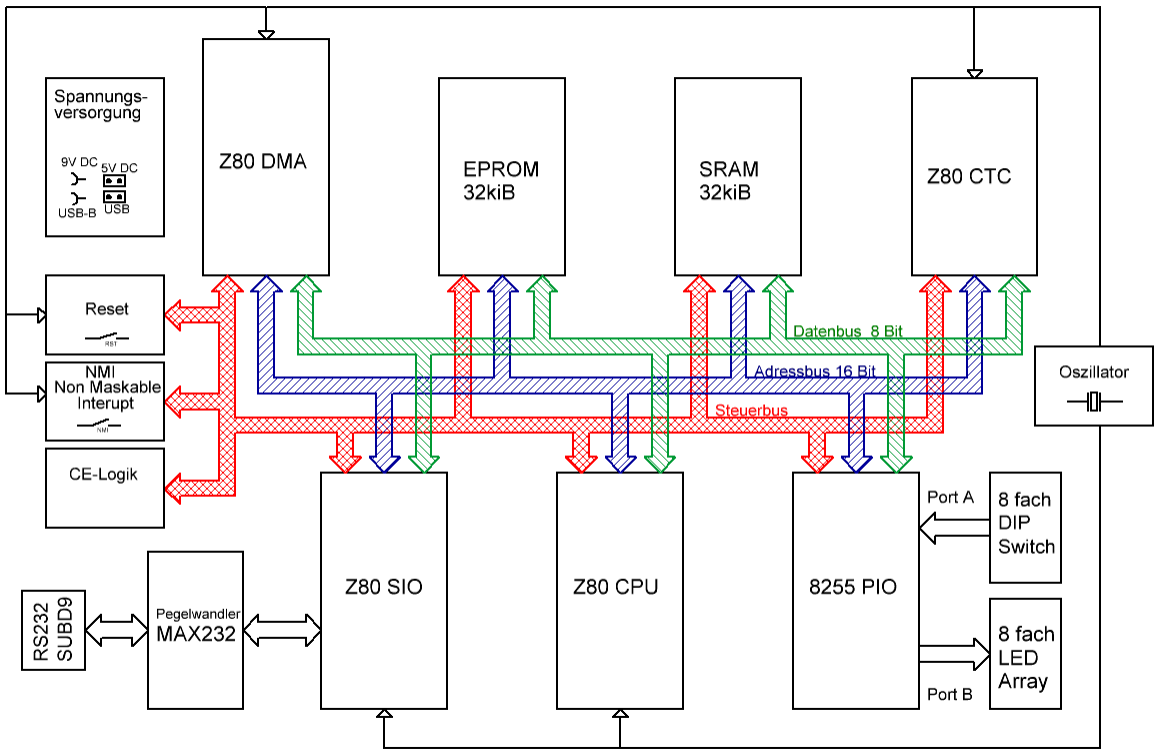


Abbildung 196: Blockschaltbild

Wie im obigen Blockschaltbilde zu sehen ist, besteht das System aus einer CPU, dem Speicherwerk und der Peripherie. Die einzelnen Baugruppen des Mikroprozessorsystems sind miteinander mit dem 16 Bit breiten unidirektionalen Adressbus, den 8 Bit breiten bidirektionalen Datenbus und dem Steuerbus verbunden.

Abbildung 197: Fotografie des Minimalsystems

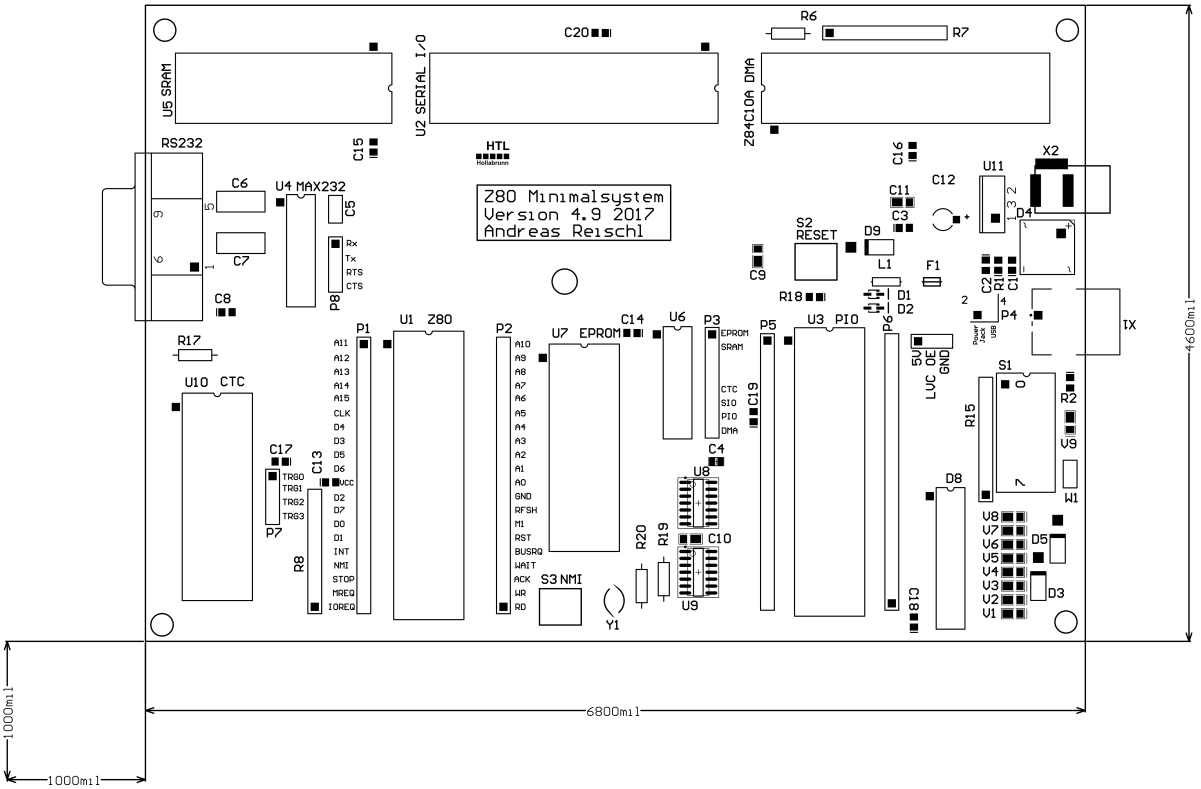


Abbildung 198: Draufsicht des Minimalsystems

In diesen beiden Abbildungen ist zu sehen, wo welche Komponenten des Z80 Minimalsystems platziert sind, wie diese aussehen und wie sie benannt werden. Die Bezeichnungen der einzelnen Teile können dem zweiten Bild entnommen werden, eine Beschreibung ist im nächsten Kapitel zu finden.

8.3 Baugruppen

8.3.1 Spannungsversorgung

Die gesamte Hardware, die beim Z80 Minimalsystem zum Einsatz kommt, benötigt eine Betriebsspannung von 5V Gleichspannung. Die erlaubte Schwankungsbreite der Spannung liegt bei allen systemspezifischen Komponenten und bei den verwendeten Logikgattern laut den einzelnen Datenblättern in einem Bereich von 4,75V bis 5,25V. Die Anforderung an die Spannungsversorgung war neben der guten Verfügbarkeit der einzelnen Versorgungsarten ein möglichst geringer Störungsanteil bei gleichzeitig einfacher Umsetzbarkeit.

Deshalb fiel die Wahl auf 2 verschiedene Versorgungsmöglichkeiten: Die erste Möglichkeit ist die Versorgung mittels Netzteil mit Linearregler auf dem Minimalsystem. Bei der zweiten Variante handelt es sich um eine USB-Versorgung, deren Schnittstelle im Gegensatz zu den vorherigen Versionen nicht als Micro-USB ausgeführt ist, sondern als USB Typ B. Dadurch wird eine bessere mechanische Stabilität der Steckverbindung ermöglicht, außerdem werden USB-B Kabel mit größeren Leitungsquerschnitten angeboten, wodurch der Spannungsabfall an der Versorgung so weit reduziert werden kann, dass die Untergrenze von 4,75V Betriebsspannung selbst bei der Verwendung von TTL-Logik anstatt der aktuell verwendeten NMOS bzw. CMOS-Technologie eingehalten werden kann.

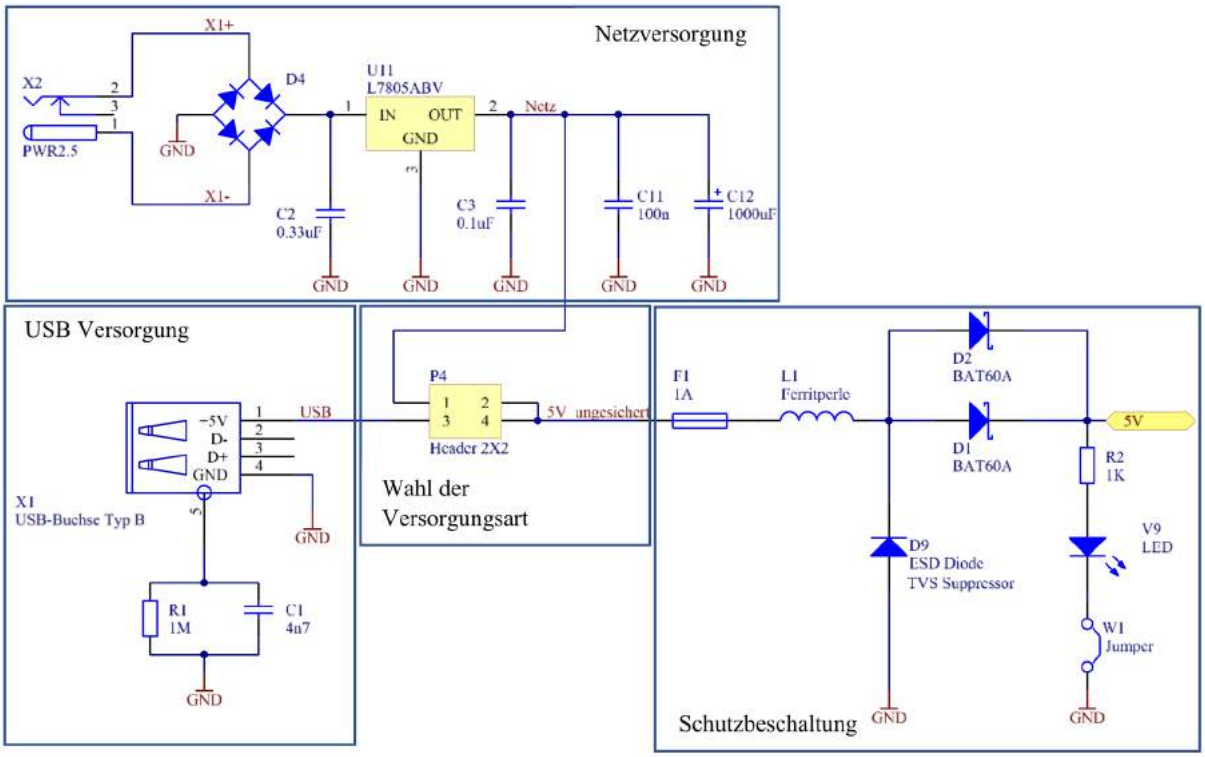
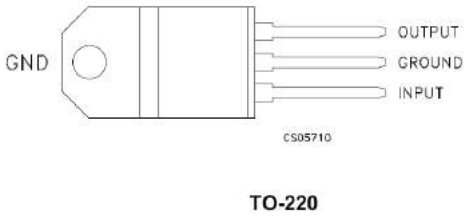


Abbildung 199: Spannungsversorgung

8.3.1.1 Fixspannungsregler/Netzversorgung

Bei dem verwendeten Fixspannungsregler vom Typ L7805CV handelt es sich um einen Linearregler des Herstellers ST Microelectronics. Dieser verfügt über eine Ausgangsspannung von 5V bei einem Maximalstrom von 1A. Die minimale Spannung am Ausgang liegt bei 4,8V, das Maximum beträgt 5,2V. Am Eingang dürfen abhängig vom Ausgangsstrom, der daraus resultierenden Verlustleistung und der Tatsache, ob und welcher Kühlkörper verwendet wird, theoretisch bis zu 35V anliegen. Um einen störungsfreien Betrieb zu gewährleisten, ist der Linearregler entsprechend Datenblatt am Eingang mit einer Kapazität von 330nF und am Ausgang mit 100nF zu versehen.

(a) Gehäuse



(b) Pinning

Abbildung 200: Linearregler Gehäuse und Pinning [17]

Der am Minimalsystem verbaute Fixspannungsregler wurde am Eingang um einen Brückengleichrichter ergänzt, um einer Verpolung entgegenzuwirken und das System vor etwaigen Schäden durch den Betrieb mit einem AC-Netzteil zu schützen. Wenn möglich, sollte aber ein DC-Netzteil mit einer Ausgangsspannung 9V verwendet werden, um die Dropout Voltage nicht zu unterschreiten und gleichzeitig die Verlustleistung gering zu halten, da kein Kühlkörper verbaut wurde. Schnittstelle zwischen Minimalsystem und Netzteil bildet ein Niedervoltsteckverbinder, auch als PowerJack bezeichnet. Der verwendete Hohlstecker besitzt einen Außendurchmesser von 5,5 mm und einen Stift mit 2mm Durchmesser.

Abbildung 201: Netzversorgung

8.3.1.2 USB Versorgung

Das Minimalsystem verfügt über eine USB Typ B Buchse, um das Minimalsystem mithilfe des USB-Treibers eines Computers oder mit einem Ladegerät für Smartphones zu versorgen.

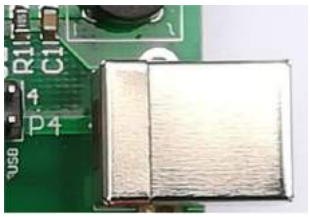


Abbildung 202: Versorgung USB

8.3.1.3 Wahl der Versorgungsart

Das Umschalten zwischen USB- und Netzversorgung erfolgt mittels Jumper P4, der auf einer 2x2 poligen Stiftleiste umgesteckt wird. Wenn man Pin 1 und 2 verbindet, wird das Minimalsystem vom Fixspannungsregler versorgt, stellt man eine Verbindung zwischen Pin 3 und 4 her, wird die USB-Buchse mit den Versorgungsleitungen des Gesamtsystems verbunden.

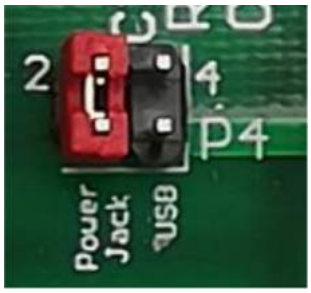


Abbildung 203: Jumper P4

8.3.1.4 Schutzbeschaltung

Um das System vor Störungen zu schützen, ist es mit einer Schutzbeschaltung versehen. Zum Schutz vor Kurzschlüssen wird eine reversible 1,1A Sicherung mit PTC-Charakteristik verbaut. Weiters sind als Verpolungsschutz 2 Schottky-Dioden verbaut, wobei die Parallelschaltung der Verringerung des Spannungsabfalls dient. Die bidirektionale TVS-Suppressordiode verhindert Auswirkungen von elektrostatischen Entladungen (ESD) und Überspannungen auf das Minimalsystem und die HF-Drossel schützt vor hochfrequenten Einstreuungen.

8.3.2 Takterzeugung/Oszillatorschaltung

Die Oszillatorschaltung besteht aus einem Quarzoszillator mit einer Resonanzfrequenz von 3,684MHz, zwei Schmitt-Trigger-Invertern und einem nachgeschalteten D-Flipflop zur Halbierung des Systemtaktes auf 1,84MHz. Die Schmitt-Trigger Inverter und das Flip-Flop dienen unter anderem dafür, gültige Logikpegel beim Systemtakt zu erreichen.

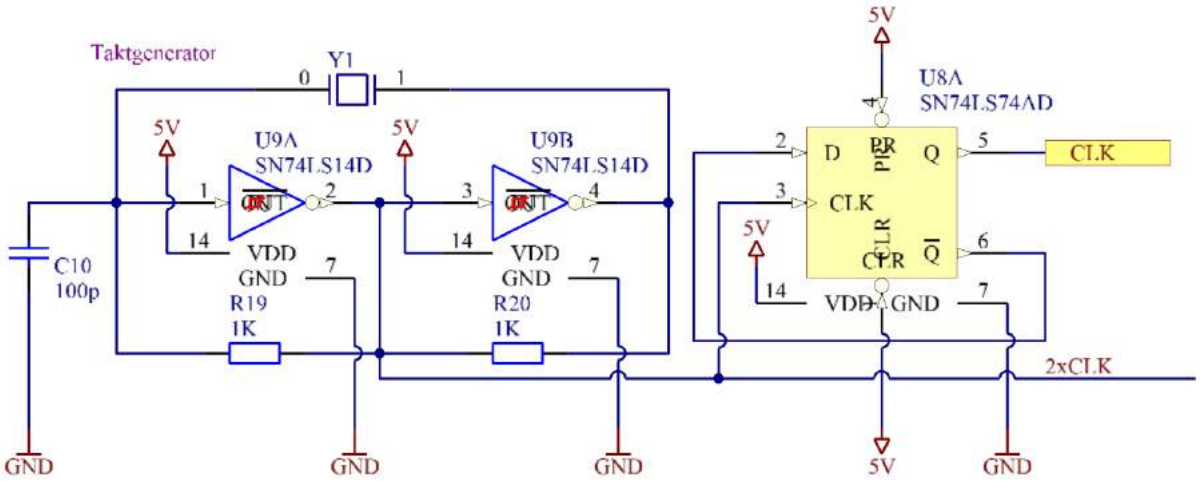


Abbildung 204: Taktgenerator

8.3.3 Resetbeschaltung

Für den Reset muss ein Impuls mit einer minimalen Pulsbreite von 3 Taktzyklen erzeugt werden. Gleichzeitig ist ein eventuelles Prellen des Tasters zu filtern, weshalb dieser mit einem RC-Glied versehen ist. Die Zeitkonstante von rund 50 ms gewährleistet, dass auch ein mehrfaches Prellen des Tasters keinen unvollständigen Reset einleitet. Durch das RC-Glied entsteht ein für den Ladevorgang eines Kondensators typischer Spannungsverlauf, welcher jedoch eine sehr geringe Flankensteilheit besitzt. Zur Erhöhung der Flankensteilheit wird die Resetschaltung um 2 Schmitt-Trigger-Inverter ergänzt. Das Flip-Flop stellt die Synchronität des Resets sicher und liefert sowohl einen invertierten als auch einen nicht invertierten Resetimpuls, welcher für den PIO benötigt wird.

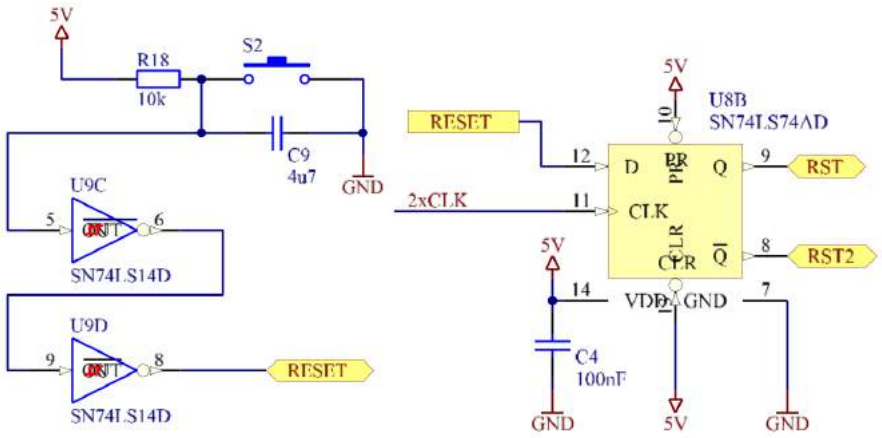


Abbildung 205: Resetbeschaltung

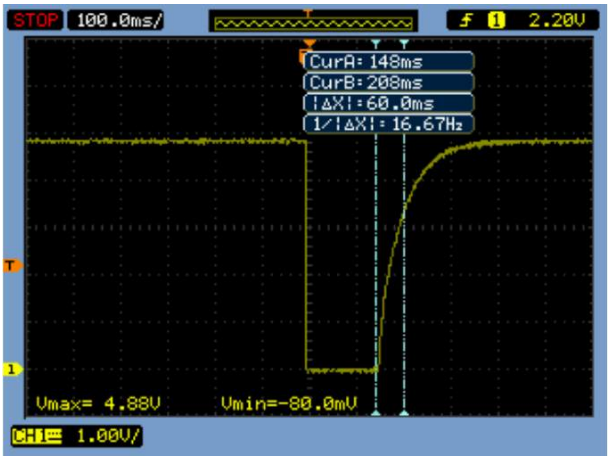


Abbildung 206: Resetimpuls am Taster

8.3.4 Z80 CPU – Central Processing Unit

8.3.4.1 Pinning

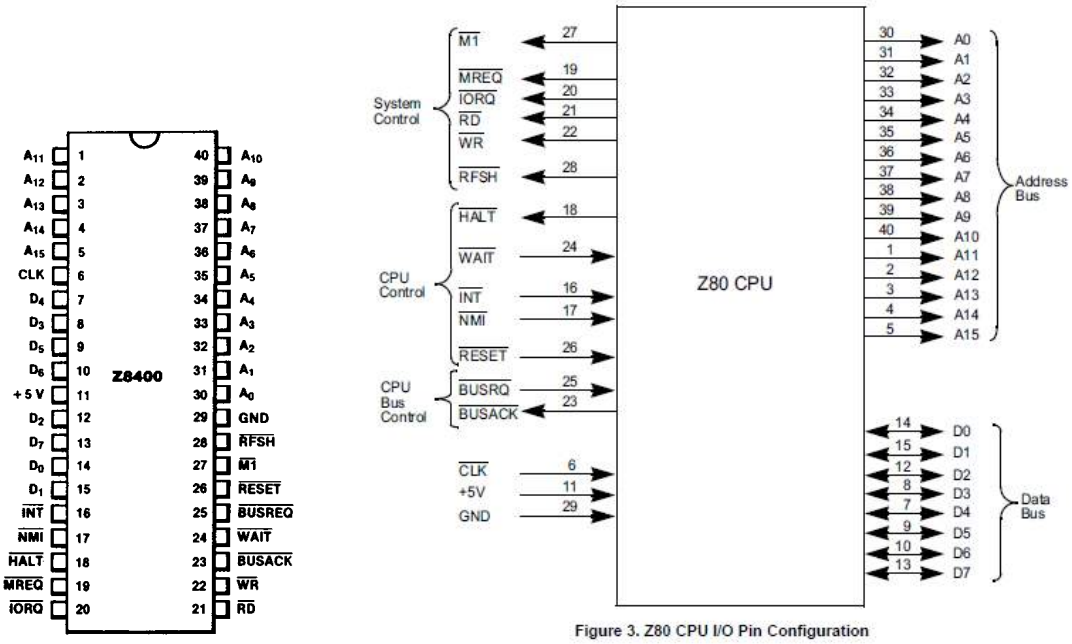


Figure 3. Z80 CPU I/O Pin Configuration

Abbildung 207: CPU Pinning

8.3.4.2 Funktionsweise und Blockschaltbild

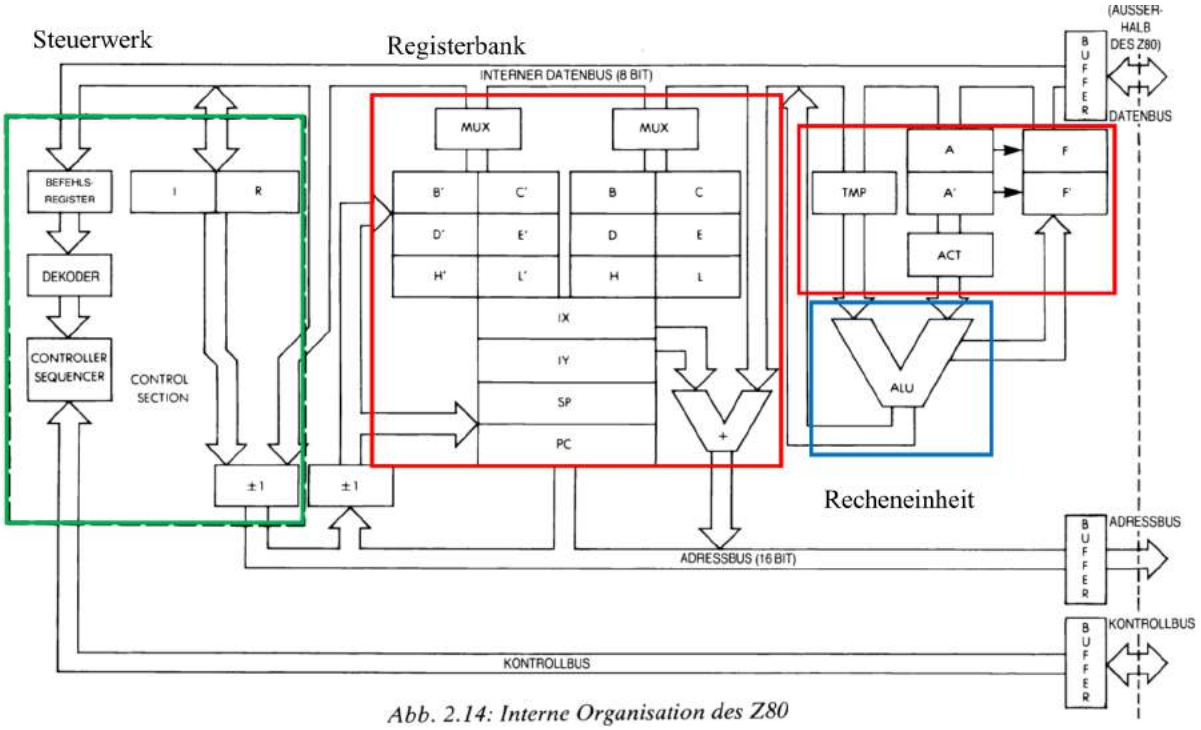


Abb. 2.14: Interne Organisation des Z80

Abbildung 208: CPU Blockschaltbild

Die Z80 CPU ist ein 8-Bit-Mikroprozessor, der von Zilog aufbauend auf dem Intel 8080 entwickelt wurde. Der Mikroprozessor basiert auf einer Von-Neumann Architektur und ist als CISC-Maschine (Complex Instruction Set Computer) ausgeführt.

Die Architektur besteht aus Steuerwerk, Registerbank und Recheneinheit. Das Steuerwerk setzt sich aus dem Befehlsregister zum Abspeichern des auszuführenden Befehls, dem Decoder zum Entschlüsseln des in Hex-Code abgespeicherten Befehls und dem Controller Sequencer zum Erzeugen von Steuersignalen zusammen. Die Registerbank besteht aus sechs 8 Bit Universalregistern und speziellen Registern wie Akkumulator, Flagregister, Stackpointer und Program Counter. Die Spiegelung der Universalregister wird für bestimmte Befehle wie etwa Interrupts benötigt, wo der Letztstand vor dem Auslösen eines Interrupts nicht extra im Speicher abgelegt werden muss. Die Ausführung der Befehle erfolgt durch die Recheneinheit, deren zentrales Element die ALU, die Arithmetic Logic Unit, bildet.

Die Kommunikation mit der Peripherie erfolgt über einen 16 Bit-Adressbus, einen 8 Bit breiten Datenbus und den Steuerbus, welcher im obigen Bild als Kontrollbus bezeichnet wird. Der Speicher wird für Programmcode und Daten gleichzeitig genutzt, gleiches gilt für den Datenbus.

Die Programmierung des Z80 erfolgt in einer Assemblersprache. Der Befehlssatz für die Z80 CPU umfasst maximal 256 Befehle, von denen 252 genutzt werden. Die übliche Länge eines Befehls bewegt sich im Bereich von einem bis vier Byte, die Ausführung eines solchen Befehls nimmt eine je nach Art des Befehls im Regelfall einen Zeitraum zwischen einem und 5 Zyklen in Anspruch.

Die verwendete Halbleitertechnologie ist üblicherweise CMOS (Complementary Metal-Oxide Semiconductor) - Technologie, ursprünglich wurde NMOS (N-type Metal-Oxide Semiconductor) - und TTL (Transistor-Transistor Logik) - Technologie verwendet. Besitzt der Baustein ein C in seiner Bezeichnung, so handelt es sich um CMOS-Technologie. Alle auf Feldeffekttransistoren basierenden Baugruppen sind TTL-kompatibel.

8.3.5 CE-Logik

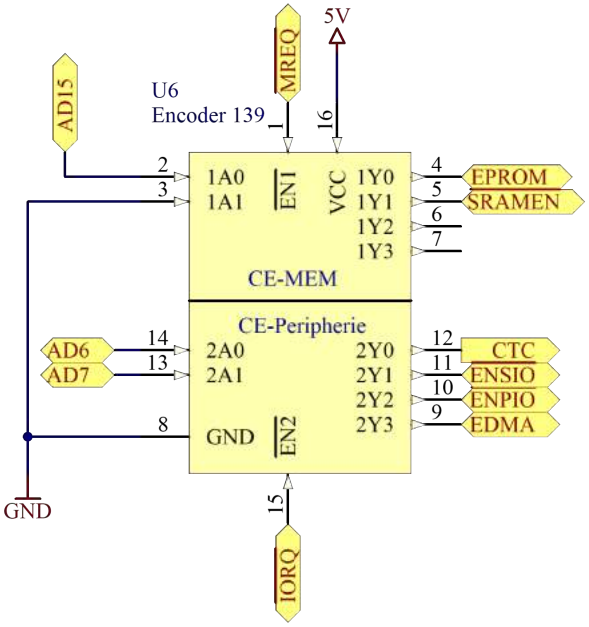


Abbildung 209: CE-Logik - 74LS244 Decoder Blockschaltbild

Die CE-Logik besteht aus 2 Demultiplexern, einem für die Peripherie und einem für das Speicherwerk. Jeder dieser Demultiplexer oder Decoder besitzt 4 Ausgänge, welche für das Enablen der einzelnen Bausteine, also das Aktivieren des jeweiligen CS- bzw. CE-Eingangs, zuständig ist. Die beiden Decoder bilden einen gemeinsamen IC, bei welchem es sich um einen Standard-IC vom Typ 74LS139 handelt.

Der Decoder für den Speicher ermittelt anhand des 16. Bits des Adressbusses, ob der EPROM oder der SRAM ausgewählt wird. Da nur 2 der 4 Ausgänge verwendet werden,

wird der Eingang 1A1 permanent gegen Masse gelegt, der Eingang 1A0 wird mit der Leitung A15, dem 16. Bit des Adressbusses, verbunden. Ist das 16. Bit 1, wird der SRAM enabled, bei 0 der EPROM. Wie der Wahrheitstabelle zu entnehmen ist, wird der jeweilige Chip erst enabled, wenn am Enable-Eingang des Multiplexers ein Low-Pegel anliegt. Dieser Eingang ist im Fall der CE-Logik für EPROM und SRAM mit dem Memory Request (MREQ) belegt.

Der zweite Decoder ist für die Peripherie zuständig Anhand des 7. und 8. Bits des Adressbusses wird bei 00 der CTC, bei 01 der SIO, bei 10 der PIO und bei 11 der DMA Controller aktiviert. Hier gilt wiederum, dass der Decoder für die Peripherie erstdurch den I/O Request enabled werden muss.

Anmerkung: Da sowohl der Datenbus als auch der Adressbus mit einer negativen Logik arbeiten handelt es sich bei logisch 1 um einen Low-Pegel und bei logisch 0 um einen High-Pegel.

INPUTS			OUTPUTS			
$n\overline{E}$	nA_0	nA_1	$n\overline{Y}_0$	$n\overline{Y}_1$	$n\overline{Y}_2$	$n\overline{Y}_3$
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	H	L	H	L	H	H
L	L	H	H	H	L	H
L	H	H	H	H	H	L

Notes

1. H = HIGH voltage level
   L = LOW voltage level
   X = don't care

Tabelle 31: Wahrheitstabelle Demultiplexer

8.3.6 PIO – Parallel Input/Output Controller

8.3.6.1 Pinning

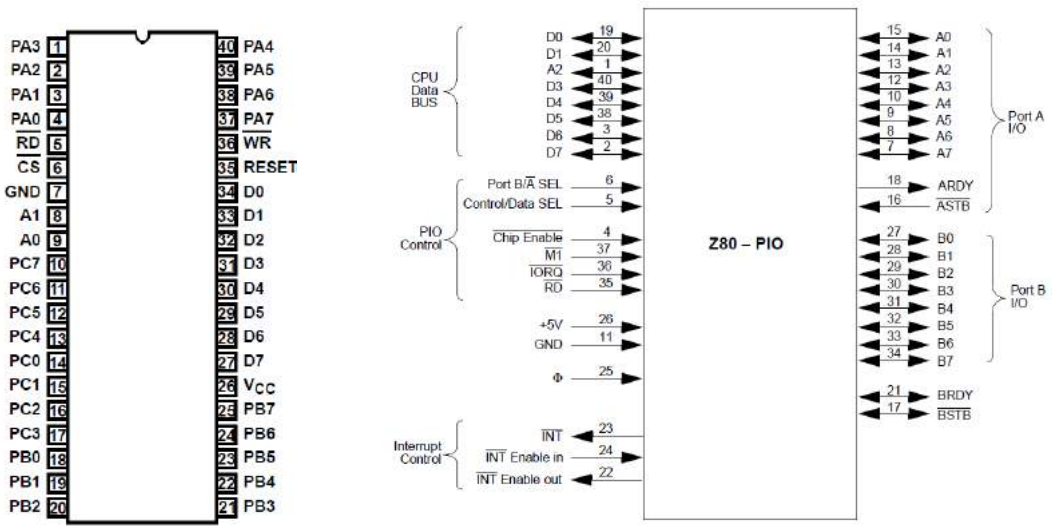


Abbildung 210: 8255 PIO Pinning [18]

8.3.6.2 Blockschaltbild und Funktionsbeschreibung

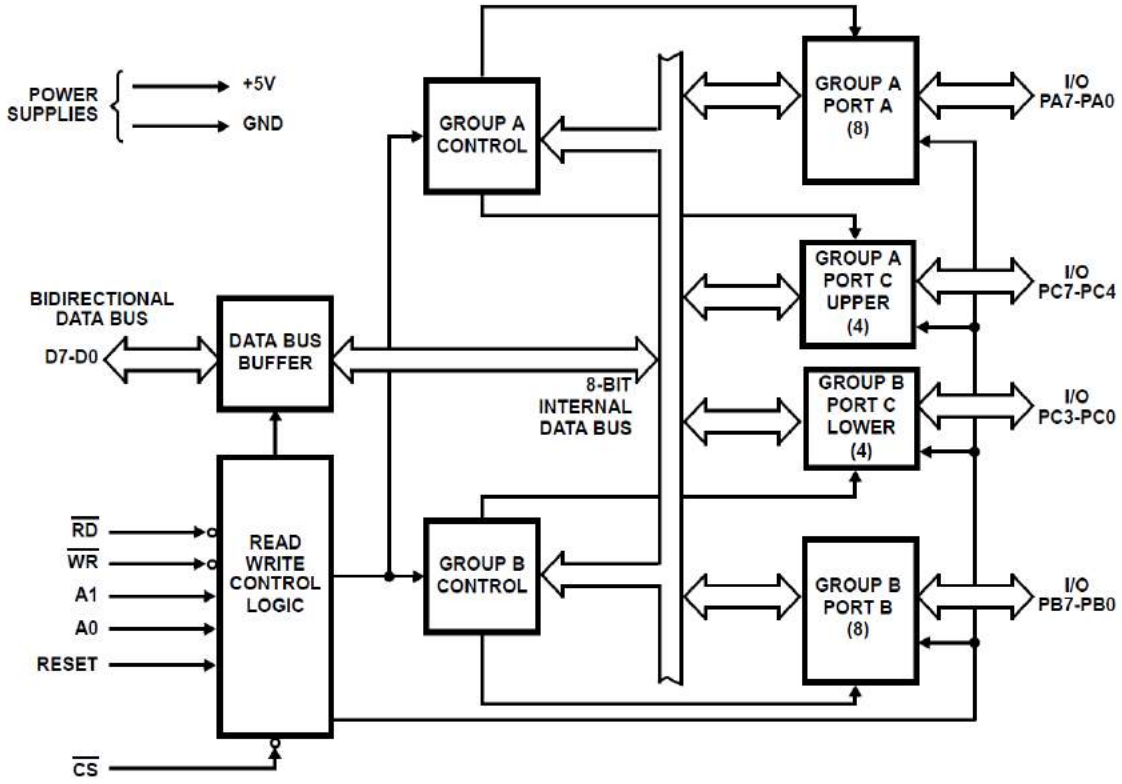


Abbildung 211: 8255 PIO Blockschaltbild [18]

Der 82C55 ist ein von Intel entwickelter Interface-Baustein, welcher als paralleler I/O Port fungiert, deshalb die Bezeichnung PIO. Als solcher bildet er die Schnittstelle zwischen Ein- oder Ausgabeeinheit und Datenbus. Der PIO besitzt 3 Ports zu je 8 Portleitungen, wodurch insgesamt 24 Ein-/Ausgänge eingesetzt werden können. Jeder der 3 Ports besitzt ein Portregister, die jeweils über eine eigene Adresse verfügen. Alle 3 Ports können einzeln konfiguriert werden, als Eingang oder als Ausgang. Die Konfiguration der einzelnen Ports erfolgt über das Control Register des PIO, welches ebenfalls über eine eigene Adresse verfügt.

8.3.6.3 Konfiguration des PIO

Der Port B des Parallel Input/Output Controllers soll als Output für die Ausgabe über LED konfiguriert werden, Port A als Input für das Einlesen der Schalterstellungen des DIL-Schalters. Die Konfiguration erfolgt durch Senden der Konfiguration an die Adresse des Steuerregisters.

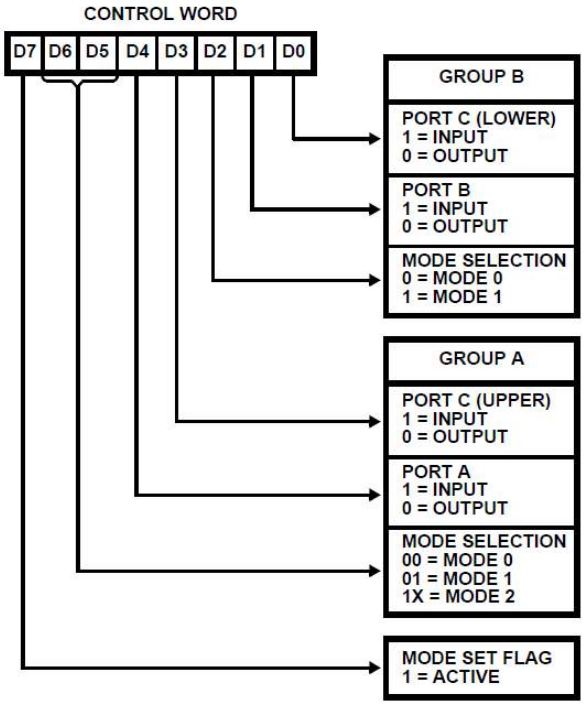


FIGURE 4. MODE DEFINITION FORMAT

Abbildung 212: Konfiguration PIO 8255 [18]

8.3.7 SIO – Serial Input/Output Controller

8.3.7.1 Pinbelegung

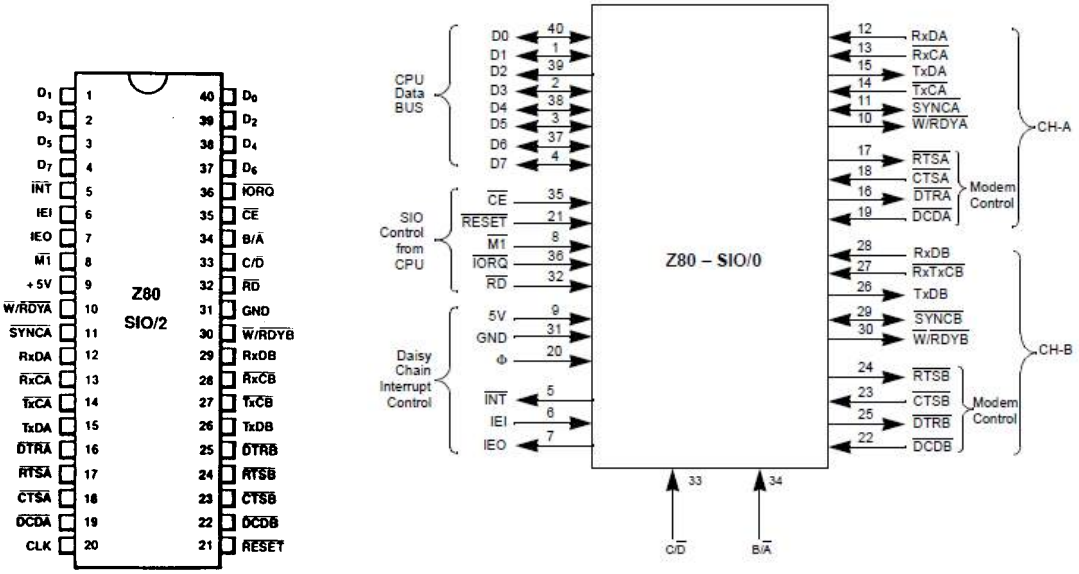


Abbildung 213: SIO Pinning [19]

8.3.7.2 Blockschaltbild und Funktionsbeschreibung

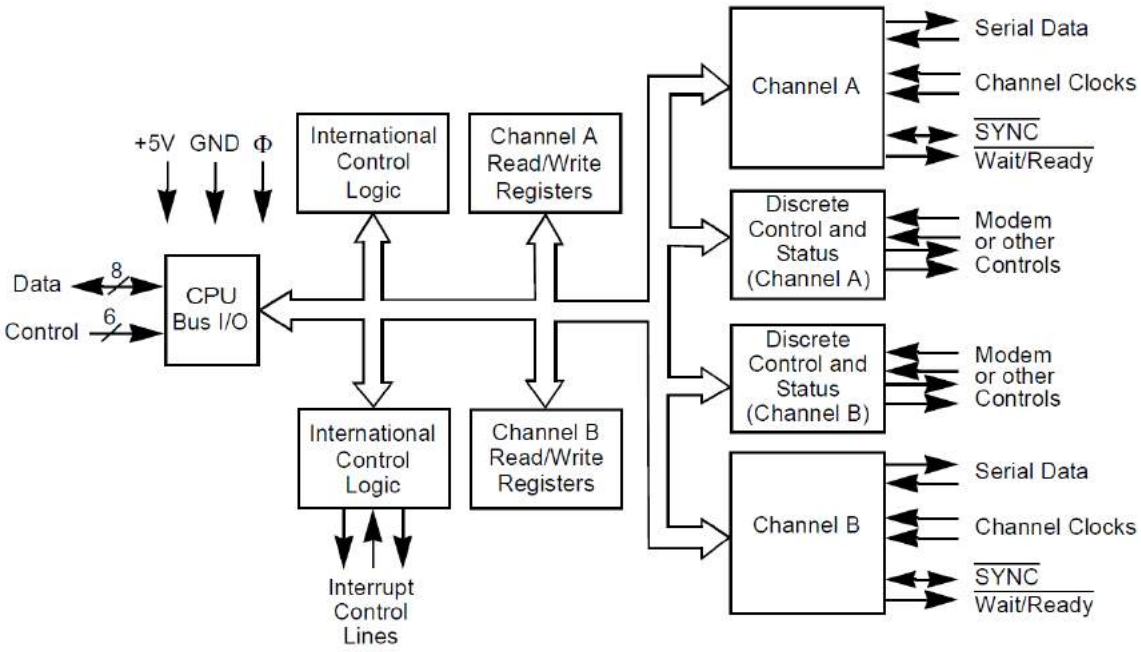


Abbildung 214: SIO Blockschaltbild [19]

Der SIO dient dazu, dass die Z80 CPU über serielle Schnittstellen Daten austauschen kann. Der Baustein wurde von Zilog entwickelt und trägt die Bezeichnung Z84C40, das verwendete Modell Z84C4006. Der Serial Input/Output Controller verfügt über 2 Kanäle, es können also bis zu 2 serielle Schnittstellen genutzt werden. Als Seriell/Parallel-Parallel/Seriell Konverter kann der SIO synchrone als auch asynchrone Protokolle verarbeiten. Weiters besitzt der SIO die Fähigkeit, Interrupts auszulösen.

Adressiert werden kann beim SIO nur für jeden Kanal ein Daten- und ein Steuerregister (Channel A/B Data/Control Register).

8.3.7.3 Konfiguration

Da die Konfiguration des SIO recht umfangreich ist und somit den Rahmen der Dokumentation sprengen würde, muss an dieser Stelle auf die vorgegebene Konfiguration in den Beispielprogrammen und auf das Z80 Peripherals User Manual verwiesen werden.

Write Register:

- WR0: Register pointers, CRC initialize, initialization commands for the various modes and more
- WR1: Transmit/Receive interrupt and data transfer mode definition
- WR2: Interrupt vector (Channel B only)
- WR3: Receive parameters and controls
- WR4: Transmit/Receive miscellaneous parameters and modes
- WR5: Transmit parameters and controls
- WR6: Sync character or SDLC address field
- WR7: Sync character or SDLC flag

Read Register:

- RR0: Transmit/Receive buffer status, interrupt status, and external status
- RR1: Special Receive Condition status
- RR2: Modified interrupt vector (Channel B only)

Die im Programm SIO V24 Echo vorgenommene Konfiguration für den SIO:

- Baudrate: 9600 Baud/sek
- Stoppbits: 1
- Startbits: 1
- Wortlänge: 8 Bit
- Paritätsbits: keines

8.3.8 Einbindung der Ein- und Ausgabeeinheiten (SIO, PIO)

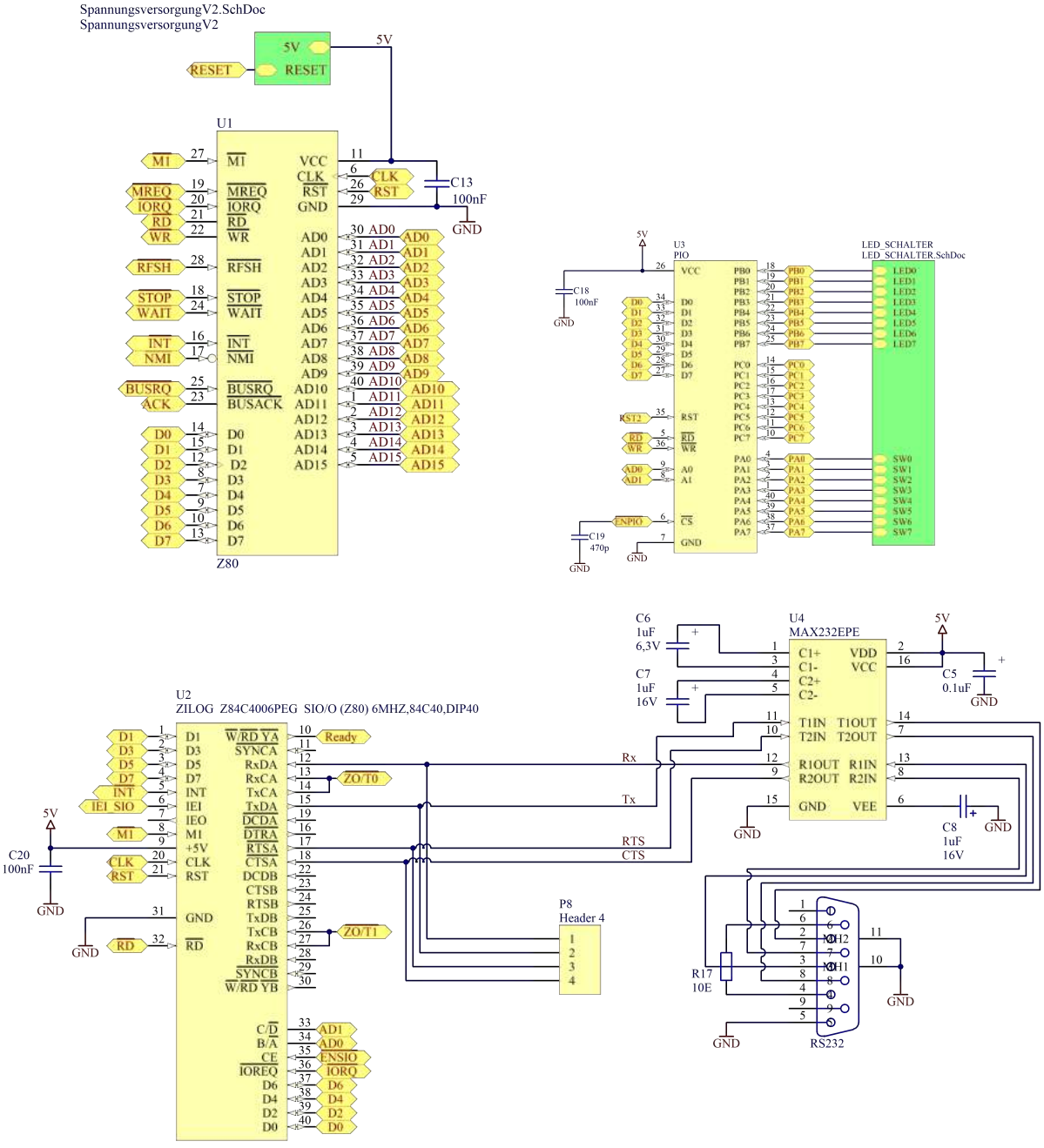


Abbildung 215: Ein- und Ausgabeeinheiten

Zur Kommunikation mit der Außenwelt besitzt das Z80 Minimalsystem einen 8-fach DIL-Schalter und ein 8-fach LED-Array sowie eine serielle Schnittstelle (RS232) für den Datenaustausch mit einem PC oder einem Terminal.

Der 8-fach Schalter und das LED-Array sind über den PIO Baustein 8255 mit der CPU verbunden, die serielle Schnittstelle wird über den Z80 SIO Baustein realisiert. Die 8 Schalter sind am Port 8 des PIO angeschlossen, das LED-Array ist mit dem Port B verbunden. Der SIO nutzt den Port A für die serielle Schnittstelle. Da der SIO nur TTL-Pegel liefert, wird ein Pegelwandler vom Typ MAX232 für die Umwandlung in RS232-Pegel eingesetzt.

Beide ICs sind über den 8 Bit breiten Datenbus und dem Low Byte des Adressbusses mit der CPU verbunden. Die die Datenflusssteuerung erfolgt über die Signale RD und WR. Beide Bausteine besitzen intern 4 adressierbare Register. Mithilfe dieser Register wird die Konfiguration und der Datenaustausch mit der CPU durchgeführt.

Die ICs selbst besitzen nur jeweils 2 Adressleitungen, A0 und A1. Das 7. und 8. Bit des Adressbusses nutzt die CE-Logik zum Enablen der Peripherieeinheiten. A3 bis A5 bleiben bei der Kommunikation mit der Peripherie ungenutzt ebenso wie das gesamte High Byte des Adressbusses.

Für den PIO lauten die Adressen der Portregister 80 für Port A, 81 für Port B und 82 für Port C. Das Konfigurationsregister besitzt die Adresse 83.

Der SIO nutzt den Datenbus dafür, um zu übermitteln, welcher Port (AD0: 0=A; 1=B) angesprochen wird und ob konfiguriert wird (AD1=1) oder ob Daten übertragen werden (AD1=0). Bei der Anwendung mit dem Z80 Minimalsystem besitzt das Datenregister des Kanals A die Adresse 40, das Datenregister des Kanals B die Adresse 41, das Steuerregister des Kanals A die Adresse 42 und das Steuerregister des Kanals B die Adresse 43. M1, also die Leitung, welche einen Machine Cycle One und damit einen Instruction Fetch kennzeichnet, bewirkt gemeinsam mit einem I/O Request einen Interrupt, wenn über die Daisy Chain (IEI) der SIO die höchste Priorität erhält.

8.3.9 CTC – Counter Timer Circuit

8.3.9.1 Pinning

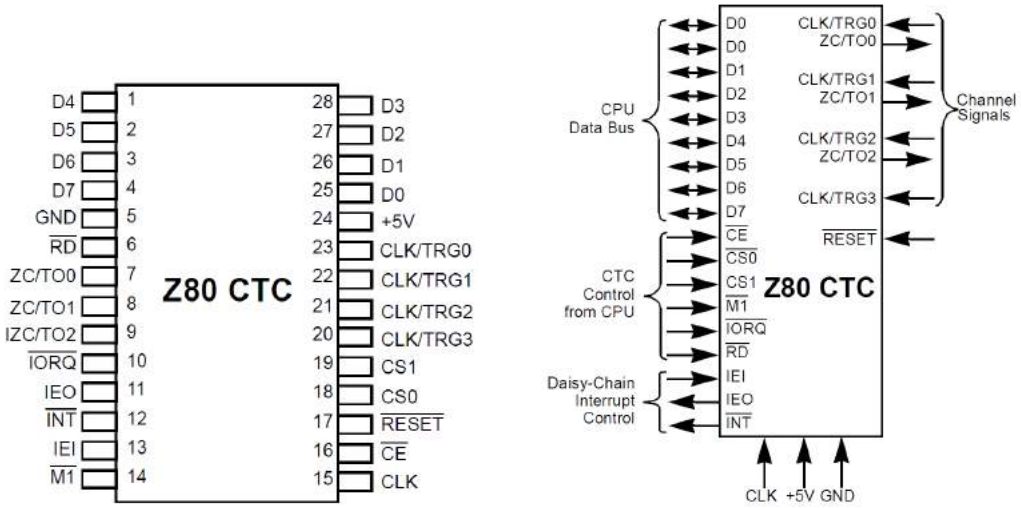


Abbildung 216: CTC Pinning [20]

8.3.9.2 Blockschaltbild und Funktionsbeschreibung

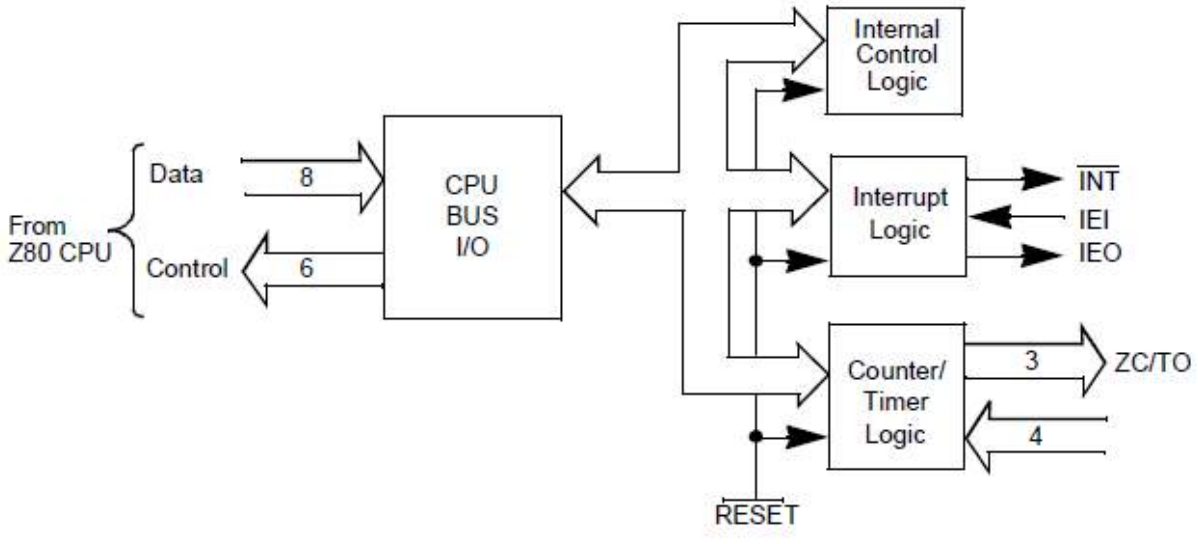


Figure 1. CTC Block Diagram

Abbildung 217: CTC Blockschaltbild [20]

Beim Counter Timer Circuit handelt es sich um einen Baustein für die Steuerung von zeitabhängigen Vorgängen. Der Z80 CTC trägt die generelle Bezeichnung Z84C30, der verwendete IC heißt 84C3006PEG. Der CTC stellt grundsätzlich 3 verschiedene Funktionen bereit, nämlich den Betrieb als Zähler, als Counter und die Fähigkeit, Interrupts auszulösen. Intern verfügt der Counter Timer Circuit über insgesamt 4 voneinander unabhängige Kanäle. Jeder Kanal besteht aus einem Zähler, der als Zähler und als Time betrieben werden kann. Für die Konfiguration und die Steuerung der internen Abläufe ist die Steuerlogik zuständig, das Interrupt Handling und das Erzeugen solcher wird durch die Interrupt-Logik vorgenommen.

Als Zähler arbeitet das System synchron mit dem Systemtakt. Die höchstmögliche Zählfrequenz entspricht der Frequenz des Systemtaktes. Der Zähler ist ein Downcounter wird 0 erreicht, wird je nach Konfiguration der Inhalt des Zeitkonstantenregisters (Time Constant Register, siehe Aufbau eines CTC-Kanals) neu geladen und ein Interrupt ausgelöst. So wie der Zähler neben dem Systemtakt auch Ereignisse am Pin TRGx des jeweiligen Kanals zählen kann, kann auch im Timerbetrieb dieser Eingang genutzt werden, um den Timer zu starten. Auf das Signal am Pin TRGx des jeweiligen Kanals wird also getriggert. Der Timer zählt die Impulse des Systemtakts nach dem Prescaler, der den Takt entweder um den Faktor 16 oder 256 teilt. Beim Nulldurchgang des Zählers wird der Downcounter des Timers aus dem Zeitkonstantenregister neu geladen, ein Signal am Pin ZC/TOx des jeweiligen Kanals erzeugt und bei Bedarf ein Interrupt generiert.

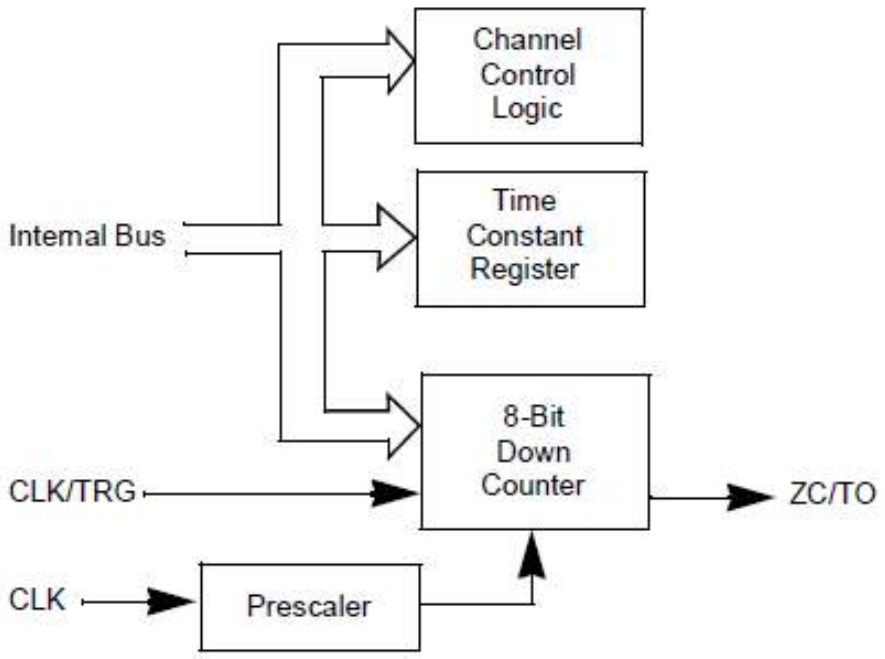


Abbildung 218: Aufbau eines CTC-Kanals [20]

8.3.9.3 Konfiguration des CTC

Ansteuerung der Kanäle über den Adressbus (AD0=CS0, DA1=CS1):

Table 1. Channel Values

	CS0	CS1
Channel 0	0	0
Channel 1	0	1
Channel 2	1	0
Channel 3	1	1

Tabelle 32: Belegung CTC Kanäle [20]

Die Konfiguration wird ähnlich wie beim PIO durch Adressierung des Steuerregisters vorgenommen, allerdings besitzt jeder Kanal ein eigenes Steuerregister (Control Register), weshalb die Konfiguration für jeden Kanal einzeln vorgenommen wird. Weiters besitzt jeder Kanal ein Zeitkonstantenregister und ein Interruptvektorregister. Die Konfiguration aller 3 Register erfolgt immer mit derselben Adresse, da die Daten intern im CTC in das richtige Register weitergeschoben werden. Deshalb ist es wichtig, die Reihenfolge der Konfiguration einzuhalten und alle 3 Register zu konfigurieren.

Table 5. Channel Control Register

7	6	5	4	3	2	1	0
Interrupt	Mode	Prescaler Value*	CLK/TRG Section	Time Trigger*	Time Constant	Reset	Control or Vector
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Tabelle 33: CTC Channel Control Register Teil 1 [20]

Table 2. Channel Control Register

7	6	5	4	3	2	1	0
Interrupt	Mode	Prescaler Value*	CLK/TRG Section	Time Trigger*	Time Constant	Reset	Control or Vector
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit Number	Field	R/W	Value	Description
7	Interrupt	R/W	1 0	Enable Interrupt Disable Interrupt
6	Mode	R/W	1 0	COUNTER Mode TIMER Mode
5	Prescaler Value*	R/W	1 0	256 16
4	CLK/TRG Edge Section	R/W	1 0	Rising Edge Falling Edge
3	Time Trigger*	R/W	1 0	CLK/TRG Pulse Starts Timer Automatic trigger when time constant is loaded
2	Time Constant	R/W	1 0	Time Constant Follows No Time Constant Follows
1	Reset	R/W	1 0	Software Reset Continue Operation
0	Control or Vector	R/W	1 0	Control Vector
*TIMER mode only				

Tabelle 34: CTC Channel Control Register Teil 2 [20]

Table 6. Time Constant Register

7	6	5	4	3	2	1	0
TC7	TC6	TC5	TC4	TC3	TC2	TC1	TC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Tabelle 35: CTC Zeitkonstantenregister [20]

Table 3. Interrupt Vector Register

7	6	5	4	3	2	1	0
Supplied by User					Channel Identifier		Word
R/W					R/W		R/W

Bit Number	Field	R/W	Value	Description
7–3	Reserved	R/W		Supplied by User
2–1	Channel Identifier (Automatically inserted by CTC)	R/W	11 10 01 00	Channel 3 Channel 2 Channel 1 Channel 0
0	Word	R/W	1 0	Control Interrupt Vector

Tabelle 36: CTC Interrupt Vector Register und Konfiguration [20]

In dieser Anwendung wird nur der Kanal A konfiguriert:

- A5: Enable Interrupt; Timer Mode; Prescaler 256; Trigger: fallende Flanke; Automatischer Trigger, wenn die Zeitkonstante geladen wird; Zeitkonstante wird ebenfalls geladen; kein Software-Reset; Control

- FF: Zeitkonstante = 256

- A8: Kanal 0, Interrupt Vektor (Zum Auslösen eines Interrupts)

8.3.9.4 Verbindung des Timerbausteins mit der CPU

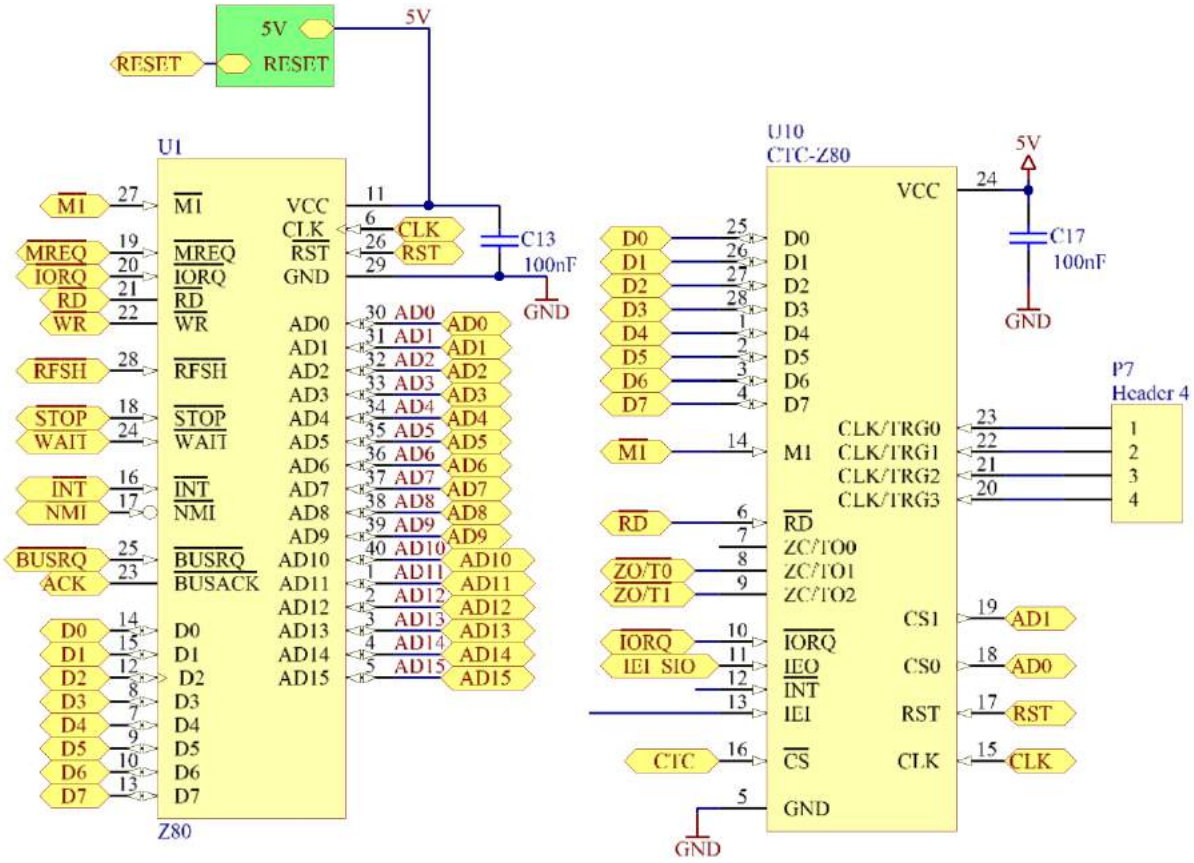


Abbildung 219: Zeitabhängige Vorgänge

Der Counter-Timer-Circuit kann sowohl als Timer als auch als Zähler eingesetzt werden. Er besitzt 3 Kanäle, von denen jeder über einen Timer verfügt. Dieser Timer kann sowohl als Timer als auch als Zähler genutzt werden. Jeder Kanal einen Eingang CKT/TRGx auf den getriggert werden kann. Im Timer Mode kann der Timer durch diesen Triggereingang gestartet werden und im Counter Mode können dort auftretende Imulse gezählt werden. Jeder Kanal besitzt ein 3 Byte großes Schieberegister zur Konfiguration, welches über die Adressleitungen A0 und A1 adressiert werden kann. Die Adressen für das Z80 Minimalsystem lauten 00 für den Kanal 1 bis 03 für den Kanal 4. als Eingang für den Zähler oder Start für den Timer genutzt werden kann.

8.3.10 EEPROM – Erasable Programmable ReadOnly Memory

8.3.10.1 Pinning

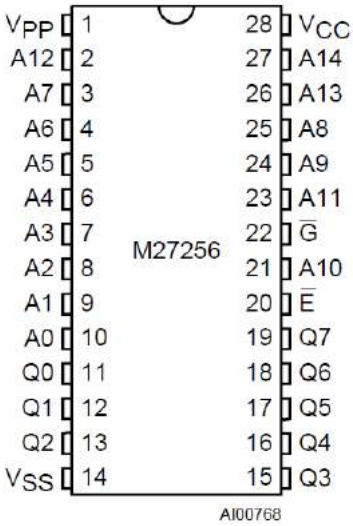


Abbildung 220: EEPROM Pinning [21]

8.3.10.2 Funktionsweise

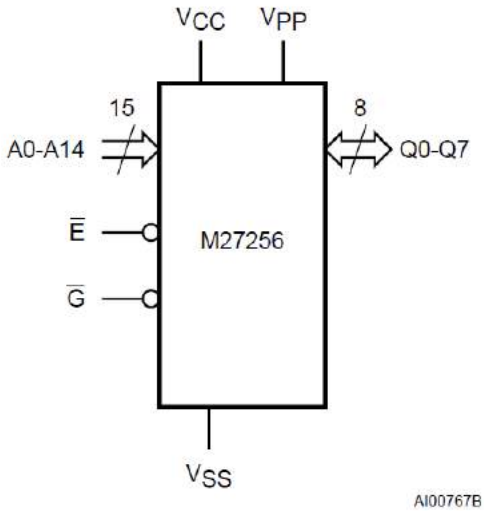


Abbildung 221: Aufbau einer Speicherzelle [21]

Beim EPROM handelt es sich um einen 32kiB großen NMOS EPROM, welcher beim Z80 Minimalsystem als Programm- und Datenspeicher dient und von diversen Herstellern

wie etwa ST Microelectronics, NEC oder Intel gefertigt wurde bzw. immer noch wird. Organisiert ist der 27256 EPROM als 32k * 8 Bit Speicher, es können also 32768 Wörter zu je 8 Bit über den 15 Bit breiten Adressbus und den Steuerbus angesprochen und über den 8-Bit-Datenbus abgerufen werden. Beschrieben kann der EPROM von der CPU nicht werden. Die Programmierung des EPROM erfolgt nicht in der Schaltung, sondern dieser muss aus der Schaltung entfernt werden und mit einem eigenen Programmiergerät beschrieben werden.

8.3.11 SRAM – Static Random Access Memory

8.3.11.1 Pinning

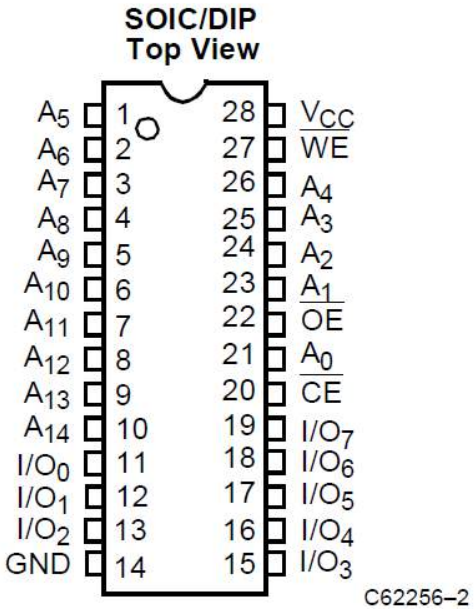


Abbildung 222: 62256 SRAM Pinning [22]

8.3.11.2 Funktionsbeschreibung und Blockschaltbild

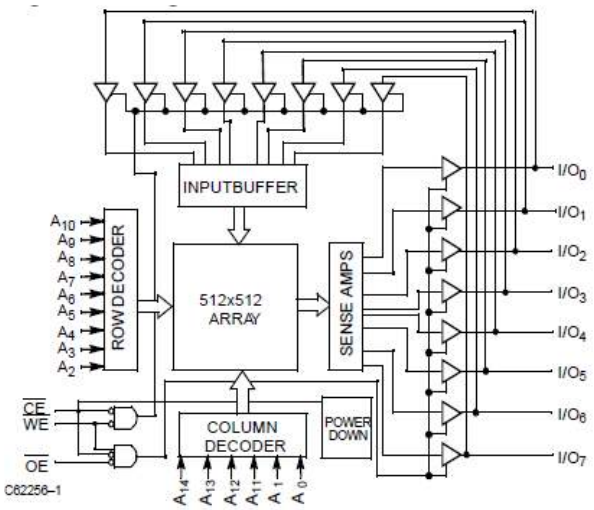


Abbildung 223: 62256 SRAM Blockschaltbild [22]

Der SRAM ist ebenfalls wie der EPROM 32kiB groß und als 32k x 8 Bit Speicher organisiert. Da der RAM im Gegensatz zum EPROM während des Betriebes des Minimalsystems auch beschrieben werden kann, wird anhand von WE und OE entschieden, ob Daten gelesen oder gespeichert werden. Dabei muss der Chip immer über CE enabled sein. Wie dem obigen Blockschaltbild zu entnehmen ist, kann immer nur der Dateneingang oder der Datenausgang aktiv sein.

8.3.12 Verbindung des Speichers mit der CPU

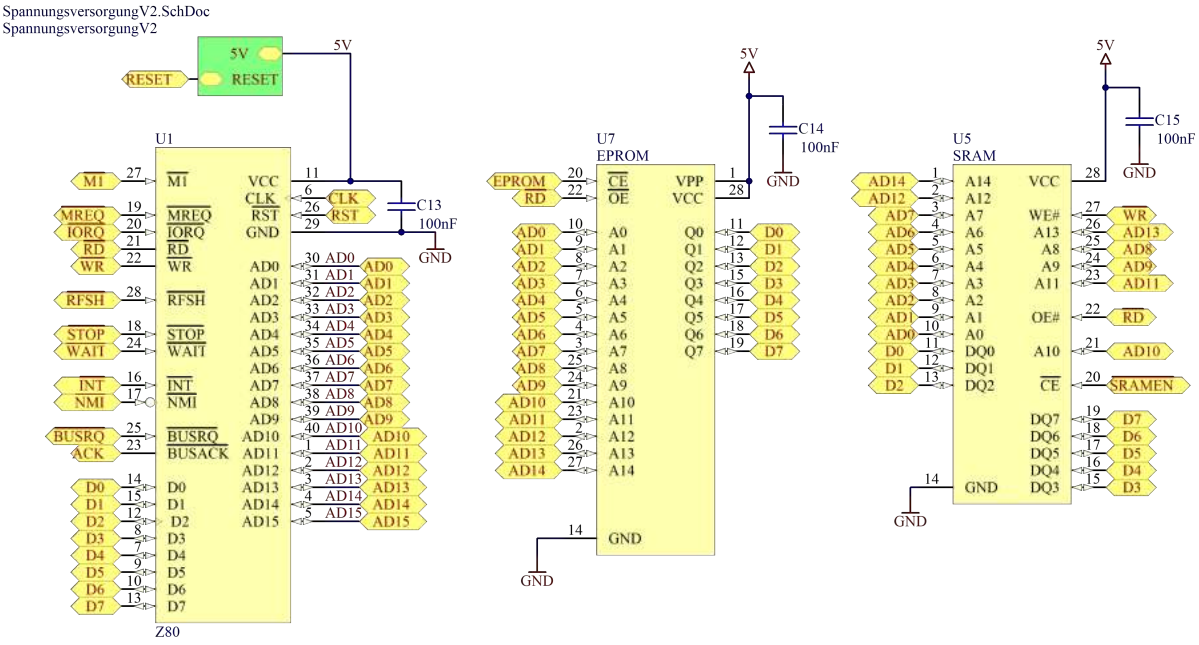


Abbildung 224: Speicheraufbau

Die Speicher (32k x 8 EPROM und 32k x 8 SRAM) sind in erster Linie über den Daten- und den Adressbus mit der CPU verbunden. Die byteweise Adressierung der Speicherzellen erfolgt über 15 der 16 Bit des Adressbusses (AD0 bis AD14), das 16. Bit (AD15) wird von der Chip-Enable-Logik verwendet. Liegt am Eingang des CE-Decoders 0 in Kombination mit einem Memory Request (MREQ) an, wird der CE-Eingang des EPROMS aktiviert, bei 1 wird vom Decoder der CE-Eingang des SRAMs (SRAMEN bzw. CE) aktiv. Wie im Timing ersichtlich ist Verbindung mit dem Datenbus aber erst dann vorhanden, wenn die CPU ein READ (RD) für einen Lesevorgang oder ein WRITE (WR) für einen Schreibvorgang erzeugt. Diese Signale bewirken ein Output Enable bzw. ein Write Enable.

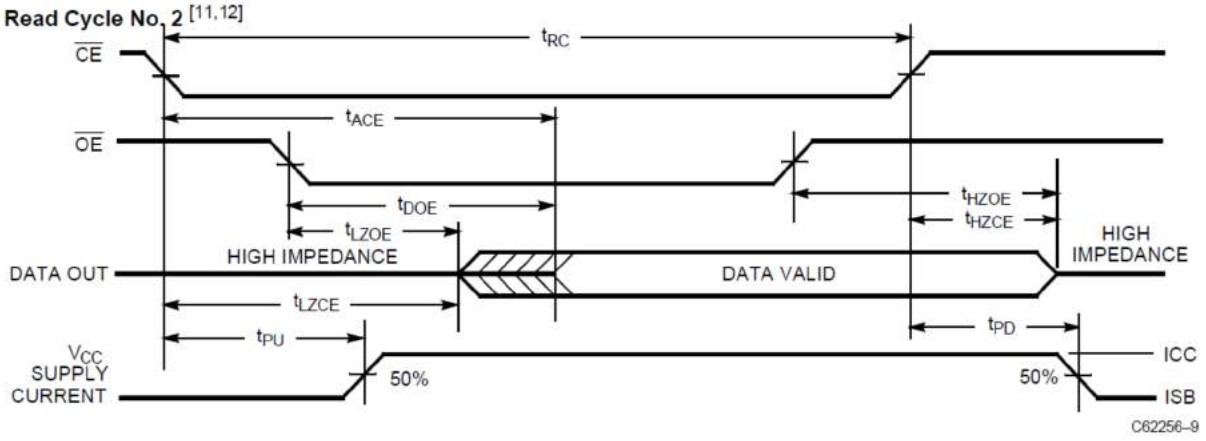


Abbildung 225: EPROM Timing

8.3.13 DMA-Controller – Direct Memory Access Controller

8.3.13.1 Pinning

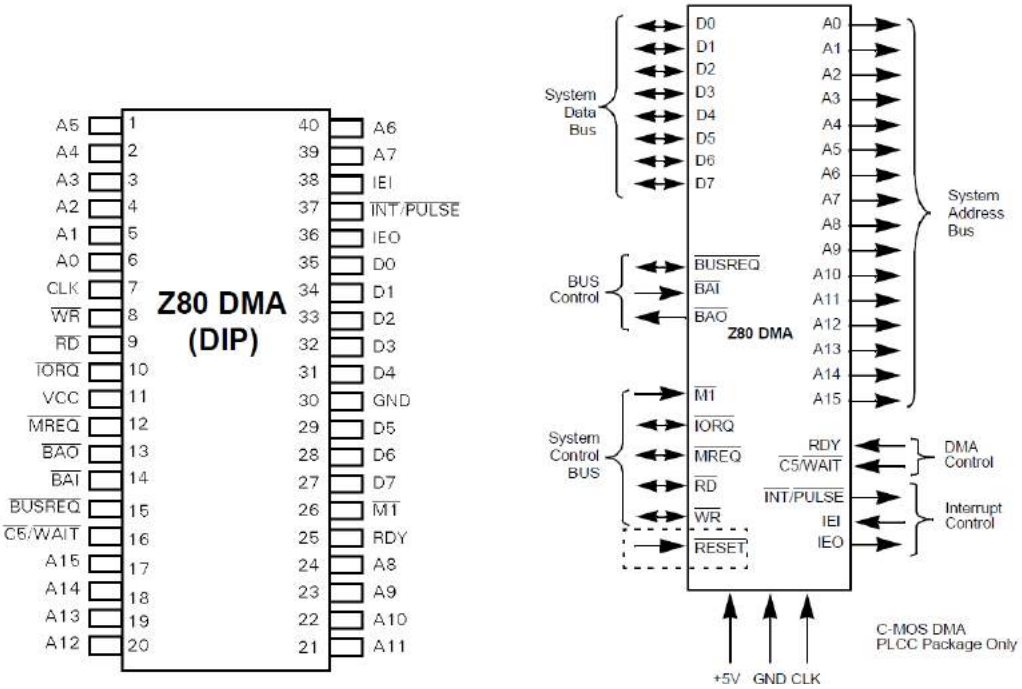


Abbildung 226: DMA Pinning [23]

8.3.13.2 Funktionsbeschreibung und Blockschaltbild

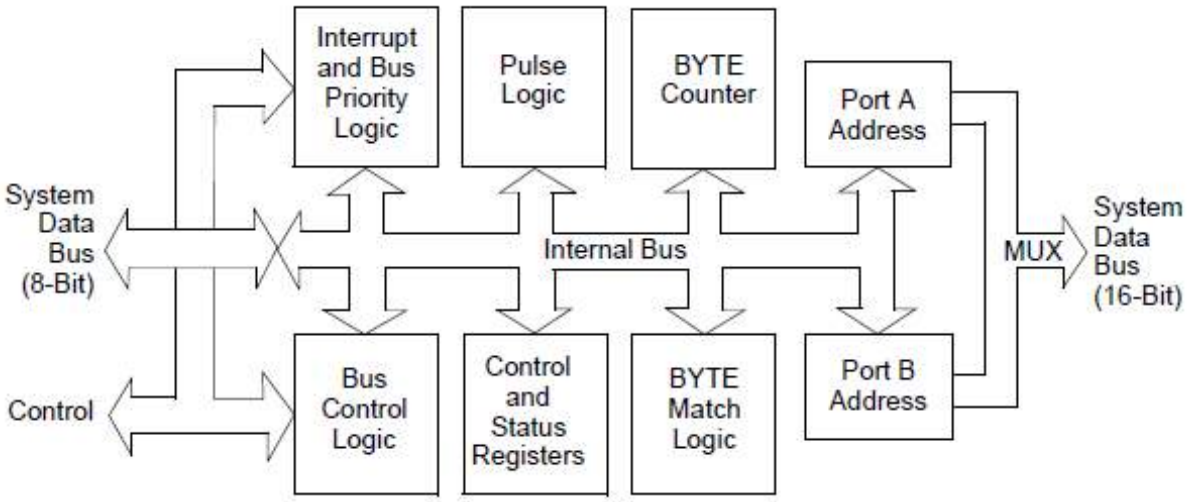


Abbildung 227: DMA Blockschaltbild [23]

Der DMA Controller ist ein für den Z80 und den Z8000 entwickelter IC mit der Bezeichnung Z8410 bzw. Z84C10. Er ermöglicht einen Speicherdirektzugriff, es werden die Daten also nicht mithilfe der CPU aus dem Speicher geholt und dann von der CPU an die entsprechende Peripherieeinheit übermittelt, sondern direkt zwischen dem Speicher und der jeweiligen Peripherieeinheit ausgetauscht. Diese direkte Kommunikation der Peripheriegeräte mit dem Speicher ist performanter als der Datentransfer via CPU, da ein Zwischenspeichern in den Registern der CPU und ein Instruction Fetch bei jedem Transfer wegfällt. Neben dem Datentransfer zwischen den 2 Ports des DMA Controllers existiert auch eine Suchfunktion. Der DMA kann in 3 verschiedenen Modi betrieben werden: Im Byte- oder Single Mode steht nach jedem übertragenen Byte kann die CPU den Datenbus nutzen, im Burst Mode werden solange Daten übertragen, bis die CPU den Übertragungsvorgang beendet und im Continuous Mode wird die Übertragung der Daten entweder durch das Erfüllen der Stopp-Bedingung oder durch das Erreichen des Endes des zu übertragenden Datensatzes beendet.

Da der DMA Controller in keinem momentan existenten Programm für das Z80 Minimalsystem Verwendung findet, wird auf die Konfiguration und den Aufbau nicht näher eingegangen.

8.3.14 NMI – Non Maskable Interrupt

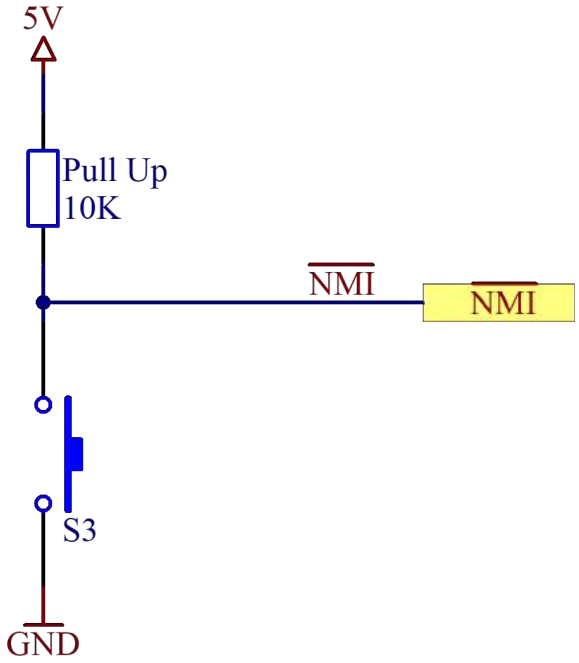


Abbildung 228: NMI Blockschaltbild

Soll ein NMI, ein nicht maskierbarer Interrupt durch den Anwender ausgelöst werden, so muss dieser den Taster S1 betätigen. Die Leitung NMI, welche mit einem Pull-Up Widerstand versehen ist, ist mit der CPU verbunden und diese löst dann einen Interrupt aus.

8.3.15 I/O Einheiten

8.3.15.1 Ausgabeeinheit

Der Port A des PIO ist mit einem 8-fach LED-Array als parallele Ausgabeeinheit ausgestattet (siehe Abbildung unten). Um den Ausgang des PIO nicht zu belasten, wird ein 8-fach Bustreiber verwendet, um die LEDs zu versorgen. Dieser Treiber wirkt invertierend und seine Ausgänge sind mit den Eingängen OE1 und OE2 mittels Jumper J3 deaktivierbar. Ist der Jumper J3 so gesetzt, dass die LEDs versorgt werden können, leuchtet auch eine zusätzliche LED (V9). Wird an den Eingang OE Masse angelegt, wird der Treiberbaustein enabled bzw. aktiviert, da es sich um einen invertierten Eingang handelt. Um die Helligkeit der LEDs von der Anzahl der aktivierten LEDs unabhängig zu machen, sind

2 Dioden in Serie zu den 8 LEDs geschaltet. Diese 2 Dioden ersetzen den sonst üblichen Serienwiderstand bei jeder einzelnen LED. Der Grund für 2 Dioden liegt darin, dass bei der Anwendung von einer einzelnen Diode der Strom durch die LEDs zu hoch ist und damit die Helligkeit.

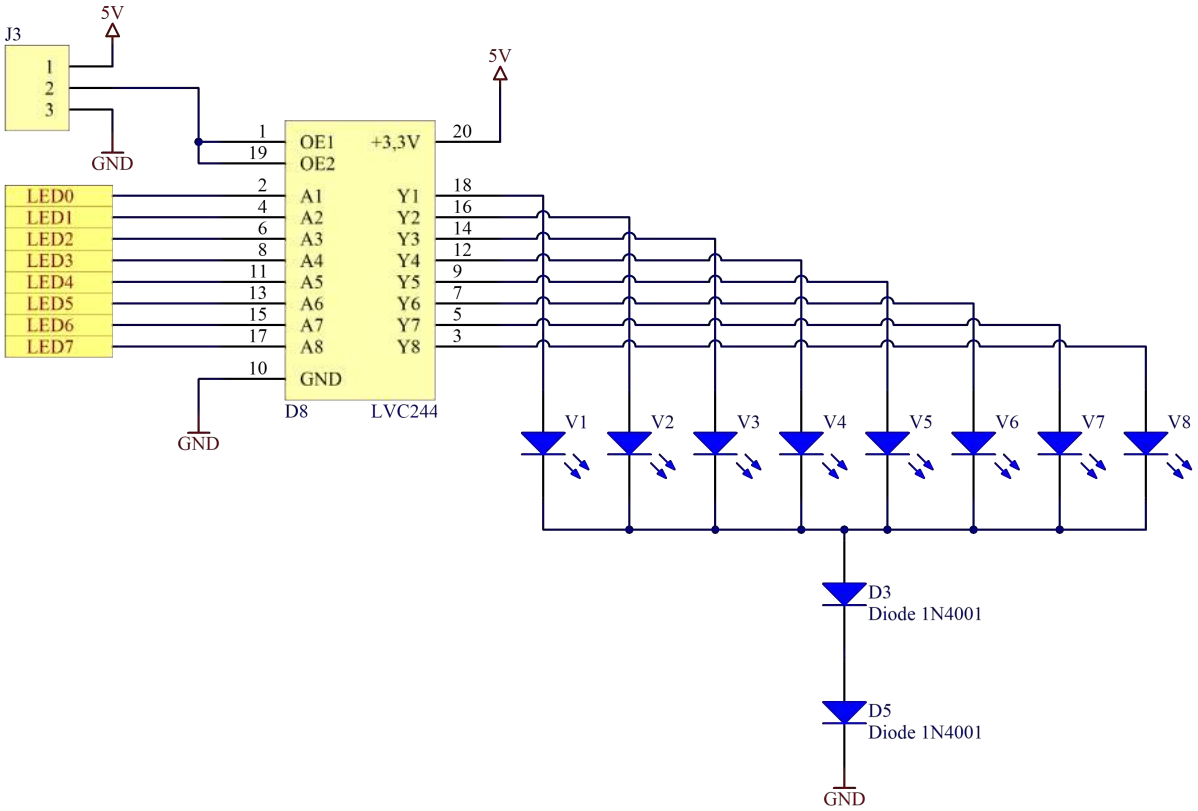


Abbildung 229: Ausgabe LEDs

8.3.15.2 Eingabeeinheit

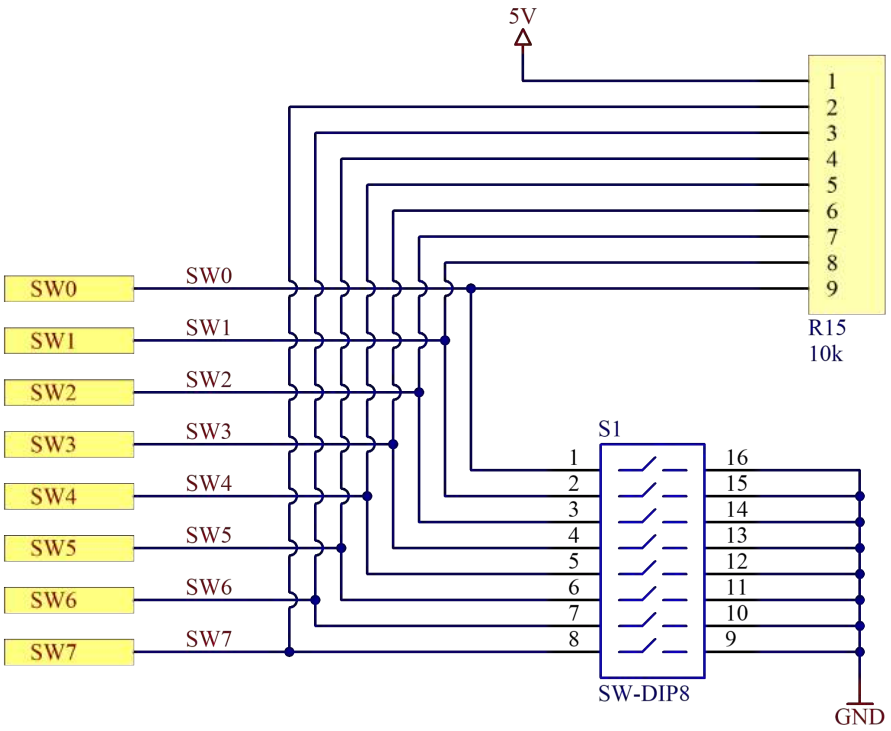


Abbildung 230: Eingabe-Schalter

Die Eingabeeinheit besteht aus einem 8-fach DIP-Schalter, dessen Ausgang mit Pull-Up Widerständen versehen sind. Ist ein Schalter also offen, liegen am Eingang des Port B des PIO 5V an, wird der Schalter geschlossen, wird der Eingang des PIO mit Masse verbunden und am entsprechenden PIN des PIO liegen 0V an. Da alle Ein- und Ausgänge des Datenbusses invertiert sind, interpretiert die CPU einen offenen Schalter als logisch 0 und einen geschlossenen Schalter als 1.

8.3.16 RS232 Schnittstelle

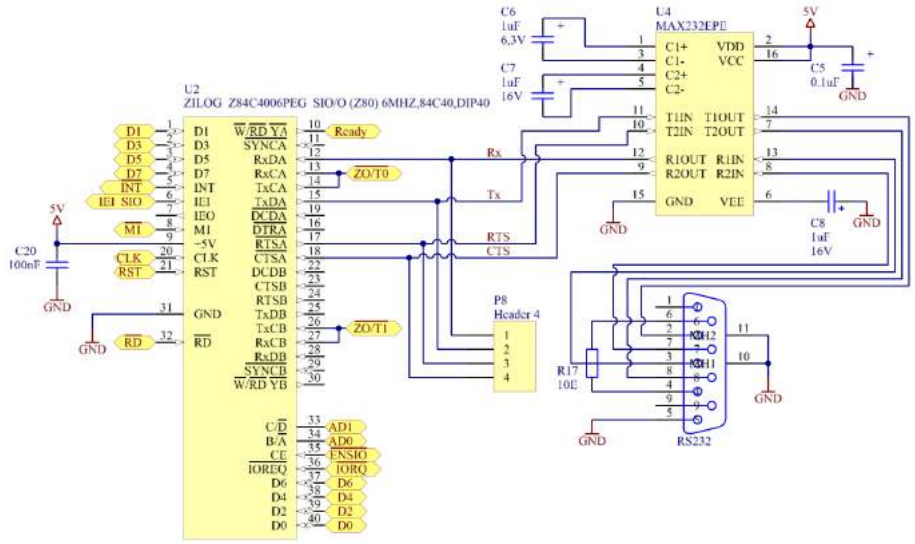


Abbildung 231: UART-R232

Die RS 232 Schnittstelle wird durch den SIO gesteuert, dieser fungiert also als UART, was auch bedeutet, dass die Anzahl von Daten-, Start- und Stoppbits, die Verwendung eines Sign Bits und die Baudrate mit der Konfiguration des Ports A des Z80 SIO festgelegt werden. Der Z80 wird als Modem (DÜE – Datenübertragungseinrichtung) betreiben, das bedeutet, dass die Bezeichnung der Datenleitungen Rx (Receive Data) und Tx (Transmit Data) und die Steuerleitungen RTS (Request to Send) und CTS (Clear to Send) nur aus Sicht des Terminals, der DEE (Datenempfangseinrichtung) gültig sind. Da zwar der UART, nicht aber RS232 mit Logikpegeln arbeitet, müssen 0V und 5V auf +/-15V mittels Pegelwandler umgesetzt werden. Diese Aufgabe übernimmt der MAX232.

8.3.17 Pull-Ups

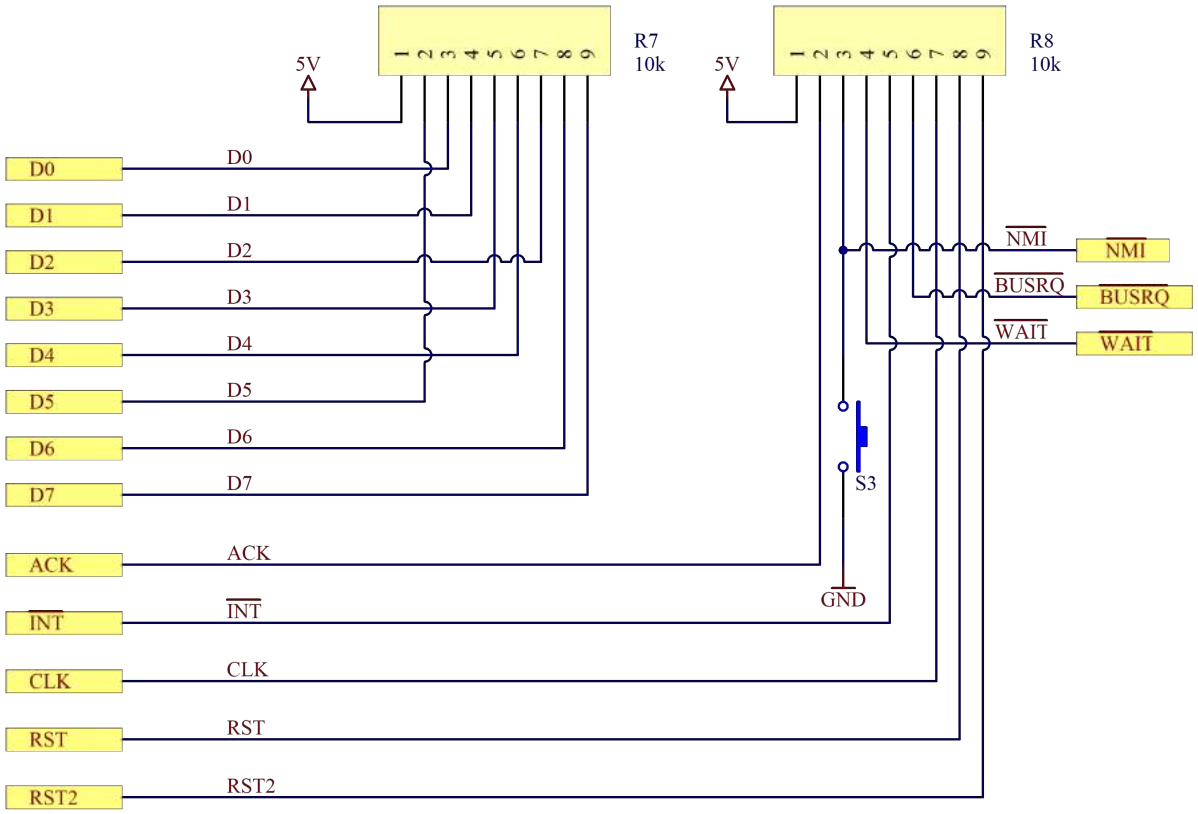


Abbildung 232: Pull-Ups

Da alle Eingänge der Busse als Pull-Up Eingänge ausgeführt sind, ist es notwendig, alle Leitungen der Busse, deren Zustand bei Nichtbenutzung nie ungleich Null sein darf, mit einem Widerstand von in diesem Fall 10kΩ gegen Betriebsspannung = 5V zu versehen. Dies geschieht am Einfachsten mit Widerstands-Arrays, wo intern 8 Widerstände parallel gegen einen einzelnen Pin geschalten werden, welcher dann mit der Betriebsspannung verbunden wird. Das Z80 Minimalsystem besitzt solche Widerstandsarrays für den Daten- und für Teile des Steuerbusses.

8.3.18 Daisy Chain

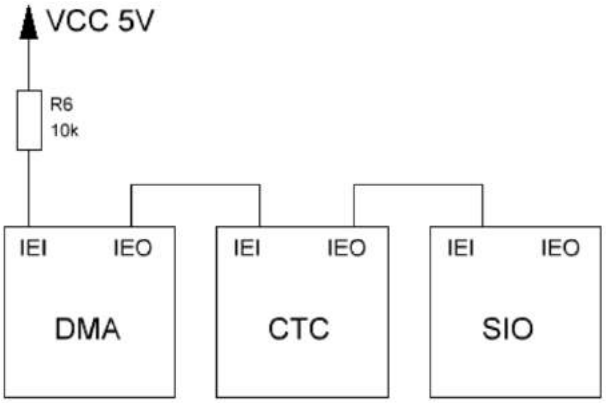


Abbildung 233: Daisy Chain

Unter Daisy Chain versteht man das Hintereinanderschalten von Bausteinen in einer Kette zur Priorisierung von Interrupts. Das Peripheriegerät mit der höchsten Priorität, in diesem Fall der DMA Controller, steht am Beginn der Kette. Der Interrupt Enable Input (IEI) ist in diesem Fall immer High, was durch einen Pull-Up Widerstand garantiert wird. Falls der DMA Controller einen Interrupt auslösen will, wird der Ausgang, welcher mit der INT des Prozessors verbunden ist, auf einen Low-Pegel gelegt. Der DMA Controller keinen Interrupt auslösen, legt er seinen Interrupt Enable Output (IEO) auf High und erlaubt damit den nachfolgenden Geräten, einen Interrupt auszulösen. Die Position in der Kette bestimmt dadurch die Priorität des Geräts. Löst ein Gerät hoher Priorität einen Interrupt aus, werden die Anforderungen für einen Interrupt der Bausteine mit niedrigerer Priorität nur gespeichert. Die einzelnen Geräte horchen am Datenbus mit und beim Befehl RETI wird das auslösende Interrupt Flag automatisch gelöscht.

Anmerkung: Die Interrupt-Priorisierung arbeitet mit nicht invertierter, also high-aktiver bzw. positiver Logik. Im Gegensatz dazu ist der INT-Eingang des Prozessors invertiert und damit low-aktiv bzw. es handelt sich um negative Logik.

8.3.19 Gesamtschaltung

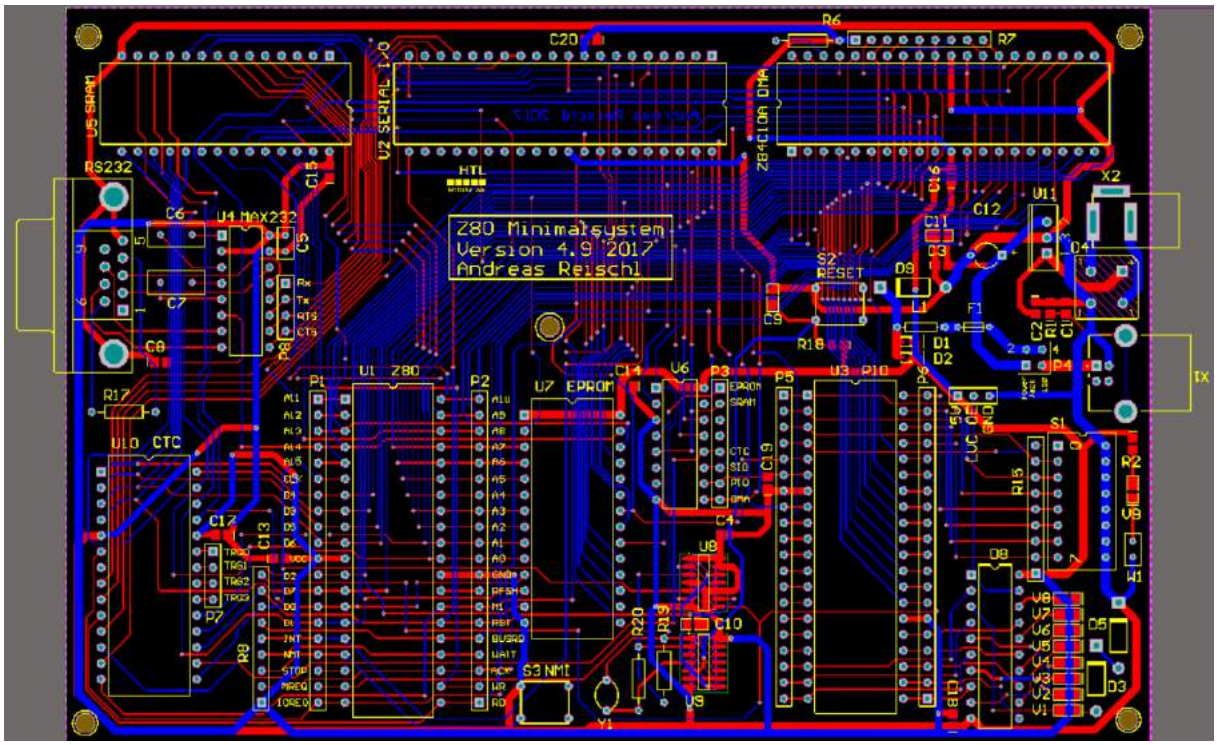
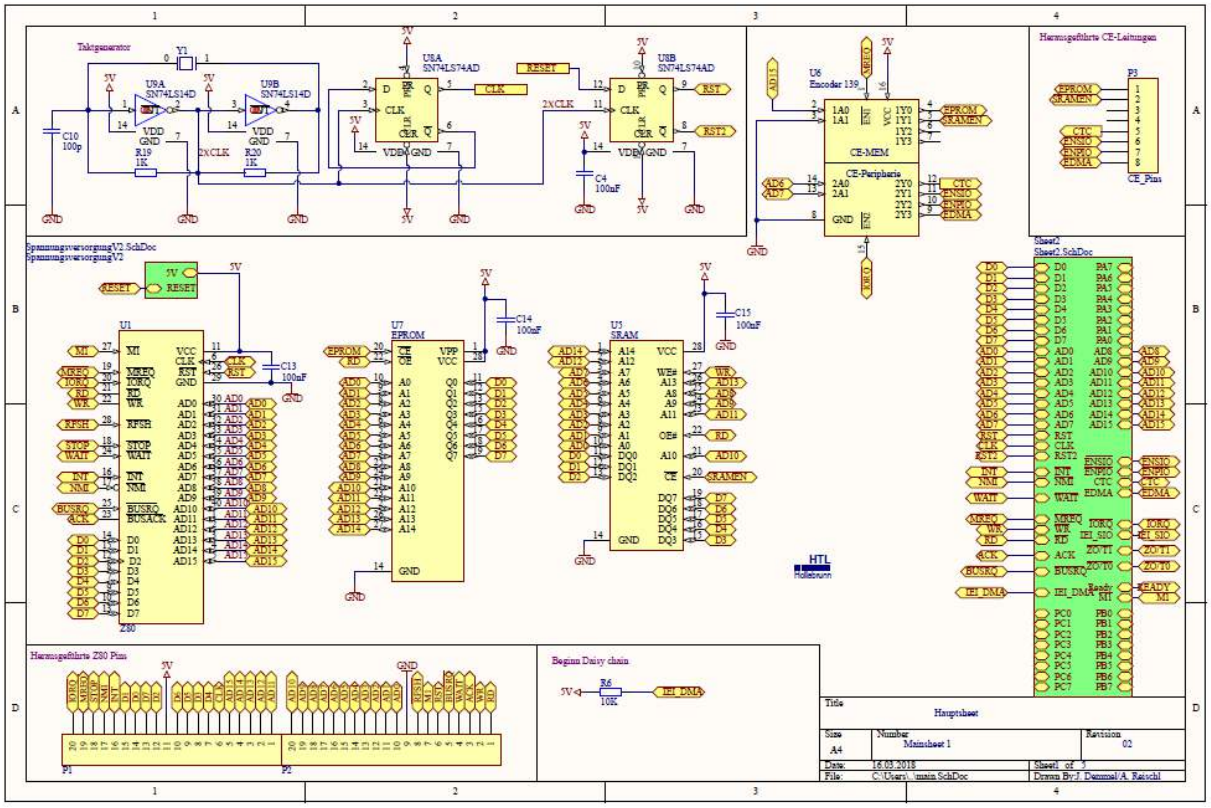
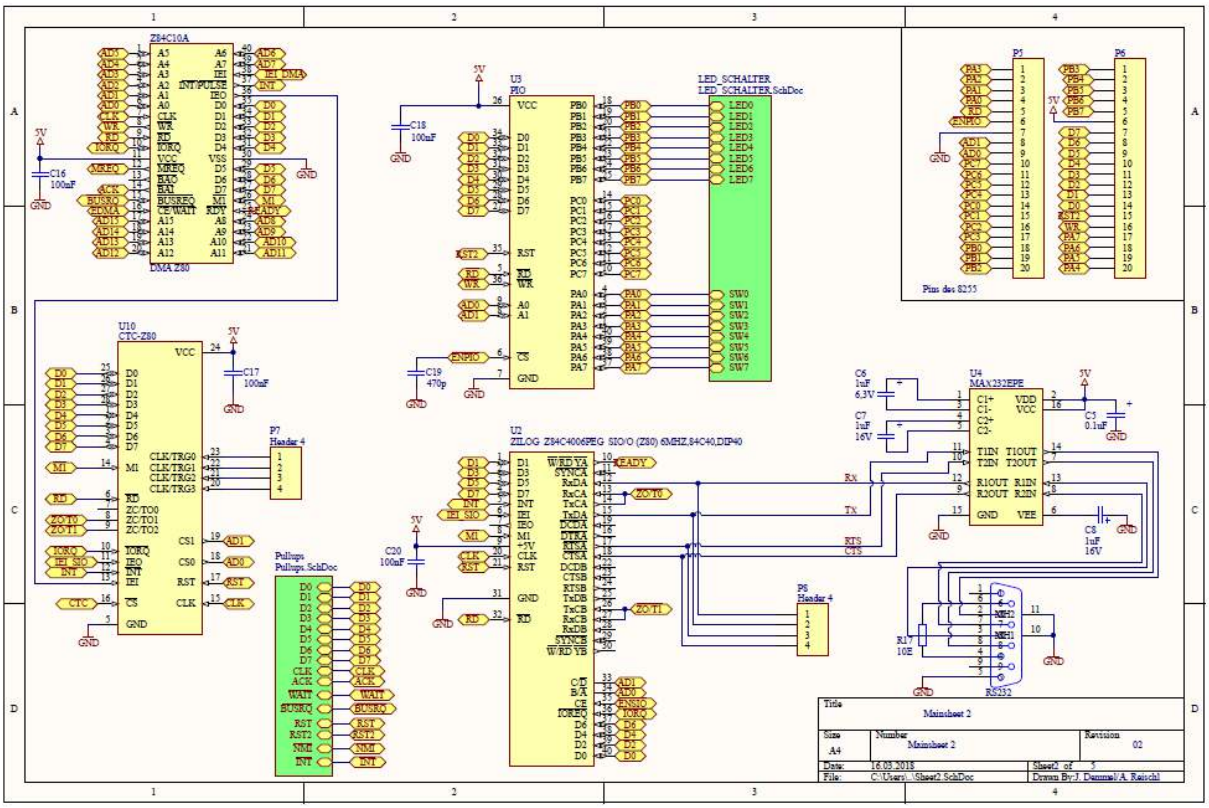


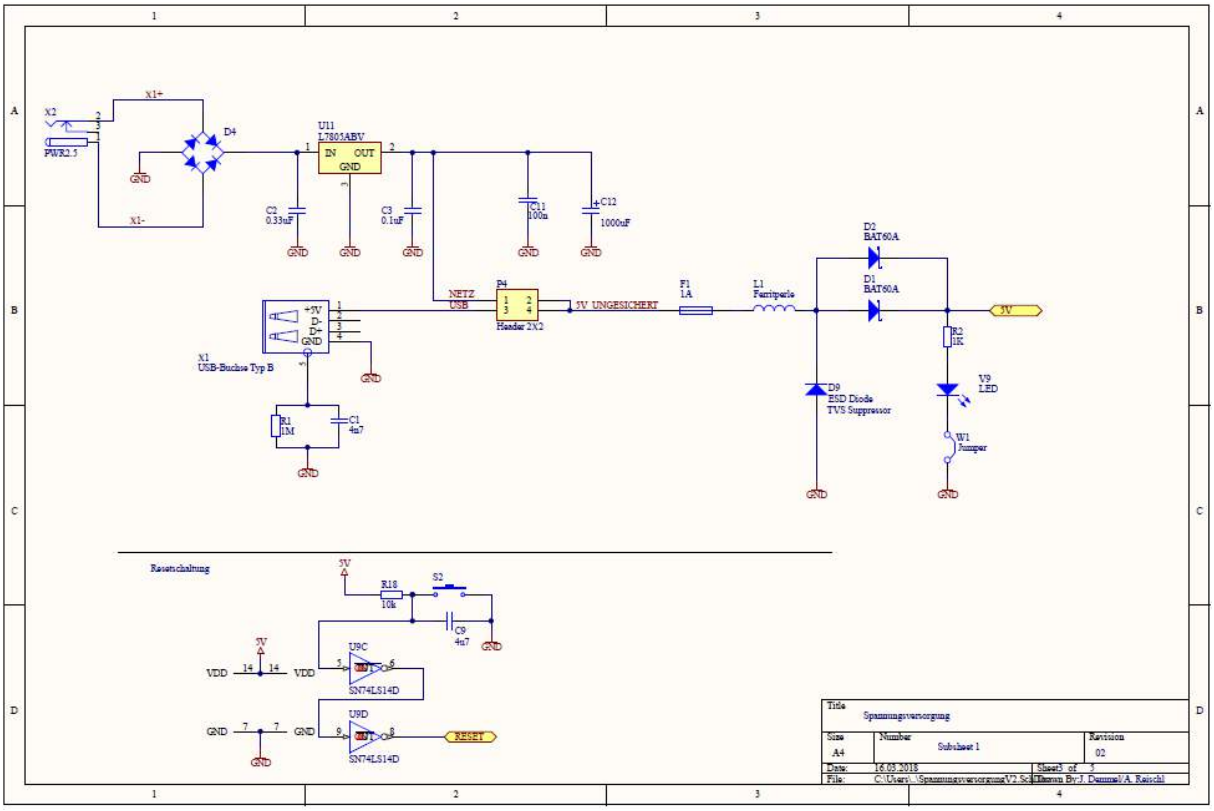
Abbildung 234: PCB



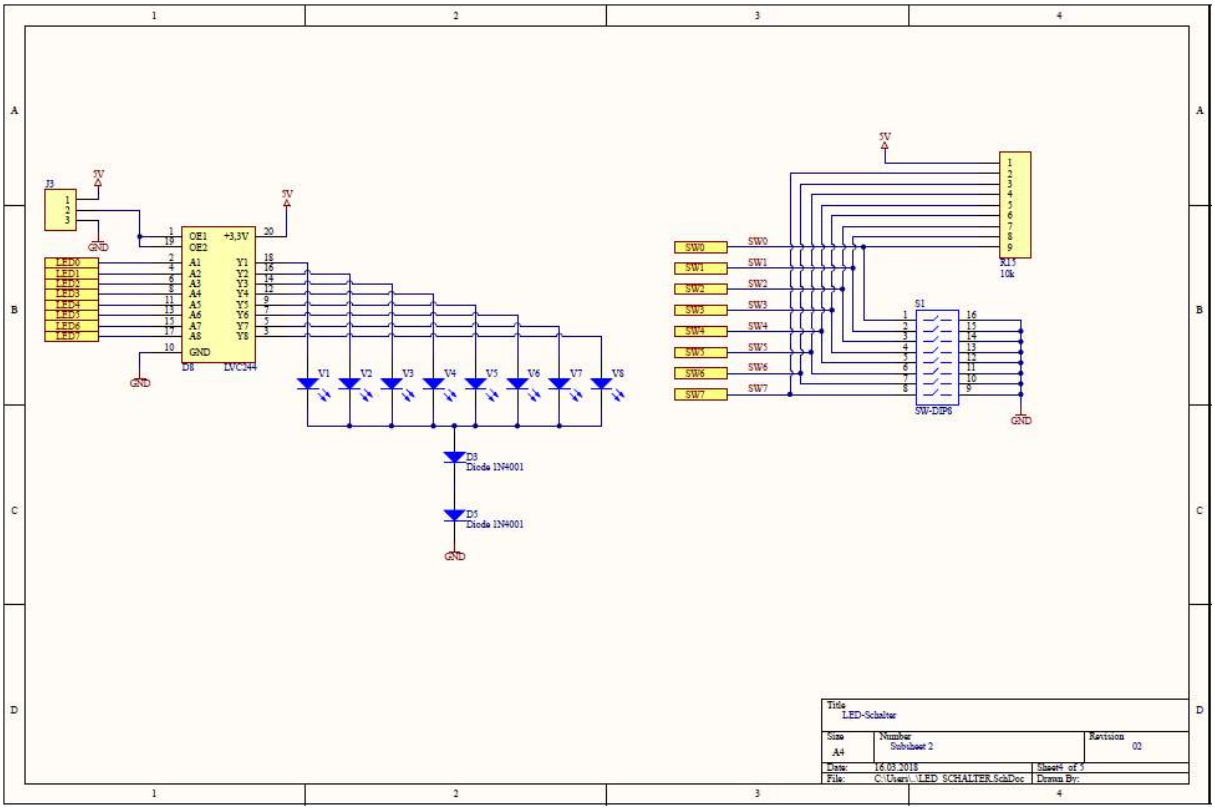
(a) Mainsheet 1



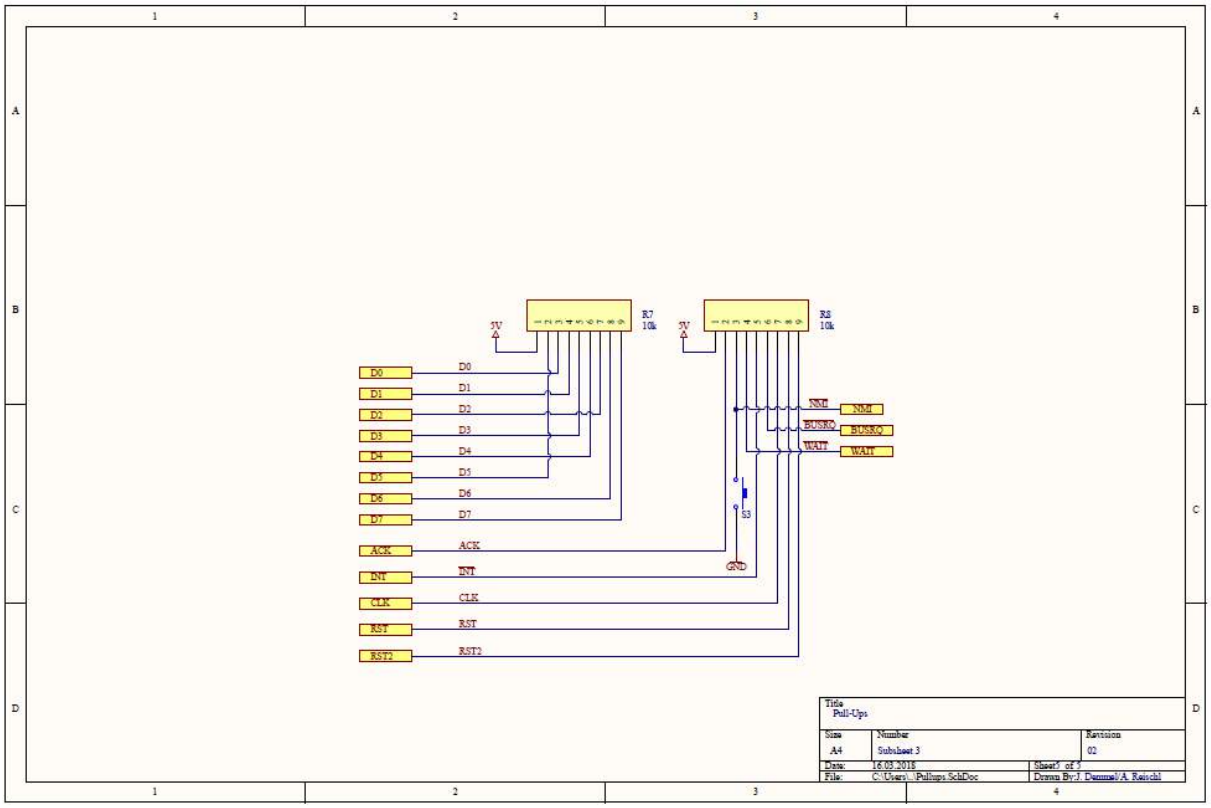
(b) Mainsheet 2



(c) Spannungsversorgung



(d) LED und Schalter (I/O)



(e) Pull-Ups

Abbildung 233: Schematics

8.4 Verbesserungen im Vergleich zur Version 4.5

8.4.1 Spannungsversorgung

Im Vergleich zu den Vorgängerversionen wurde die Spannungsversorgung stark modifiziert. Beim Z80 Minimalsystem V4.5 wurde eine Leiterbahnbreite von unter 25 mil = 0,64mm für die Versorgungsleitungen vorgesehen, welche bei der Verwendung alter TTL-Komponenten, wo der Stromverbrauch abhängig vom angewendeten Programm 400 bis zu 440 mA beträgt, zu einem Spannungsabfall von 0,6V zwischen Linearregler und 5V Pin der CPU führten. Es konnte das in den Datenblättern der Z80-spezifischen Komponenten vorgegebenen Minimum von 4,75V nicht erreicht werden. Der gemessene Spannungswert an den herausgeführten Pins der CPU ergab eine Spannungsdifferenz von minimal 4,35V zwischen VCC und GND. Bei der aktuellen Version 4.9 sind die Leiterbahnen für 5V und Ground mit mindestens 50 mil Breite ausgeführt, ebenso ist anstelle der Schottky-Diode, welche als zusätzlicher Verpolungsschutz dient, eine Parallelschaltung zweier Schottky-Dioden zur Minimierung der Spannung an der Diode in Verwendung. Als Ergebnis dieser Überdimensionierung erhält man eine Versorgungsspannung von mindestens 4,75V

an den Versorgungspins der CPU, auch dann, wenn TTL- anstatt CMOS- oder NMOS-Technologie eingesetzt wird.

Neben sehr vielen kleineren Änderungen sollte auch der Wechsel von Micro USB auf USB Typ B erwähnt werden. Durch diese Änderung wird die mechanische Stabilität des Steckers wesentliche verbessert und dadurch Störfälle minimiert.

8.4.2 Reset

Der Reset wurde ursprünglich ohne die Verwendung eines RC-Gliedes zur Stabilisierung des Signals beschaltet, was sehr viele kurze Störimpulse durch das Prellen des Schalters und dadurch einen unvollständigen Reset zur Folge hatte. Nunmehr ergibt ich durch den Kondensator C9 parallel zum Reset-Taster ein für den Ladevorgang eines Kondensators typischer Spannungsverlauf, der einer Exponentialfunktion entspricht und frei von Störimpulsen ist. Mithilfe des Schmitt-Triggers wird eine diskrete Schaltschwelle geschaffen und die Flankensteilheit des Impulses erhöht, sodass dieser für digitale Eingänge nutzbar wird.

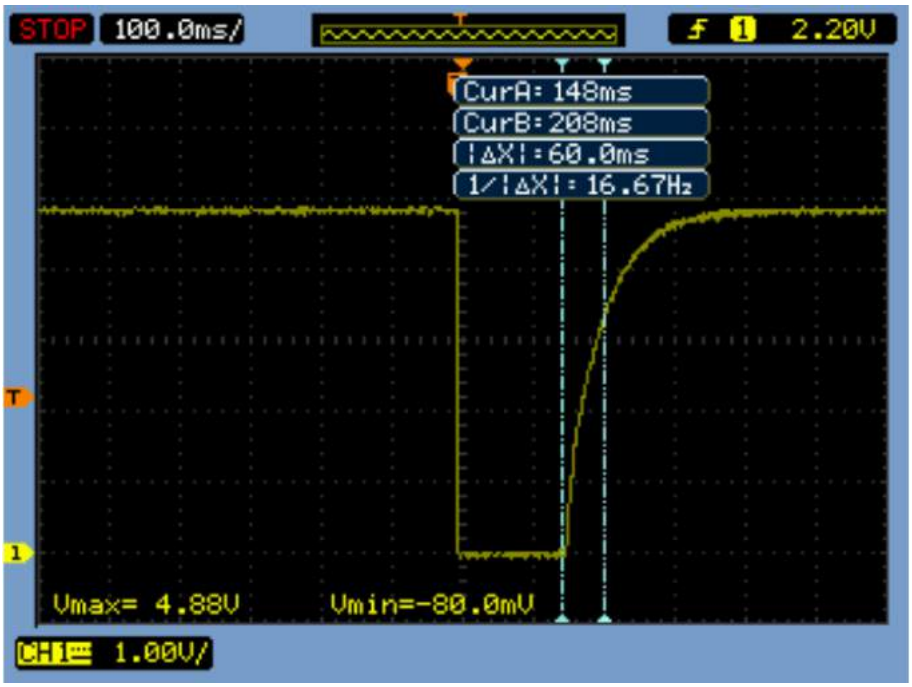


Abbildung 234: Resetimpuls am Taster S2

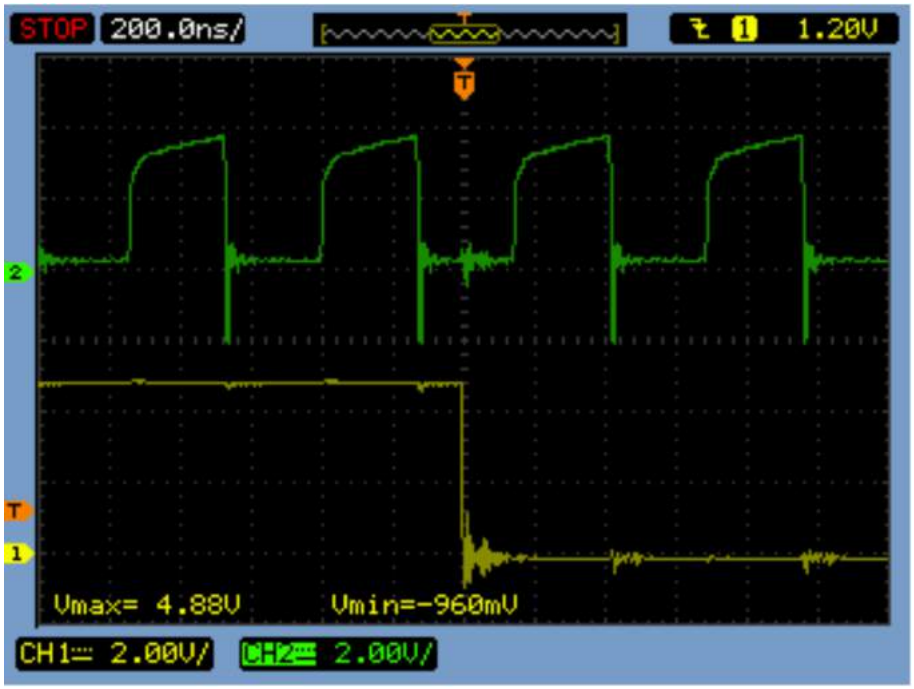


Abbildung 235: Taktsignal und Reset an der CPU

8.4.3 Taktsignal

Das Taktsignal wirkt sich nicht negativ auf die Funktion des Minimalsystems aus, jedoch ist das Signal noch verrauscht und die Pegel des Systemtaktes weichen von den Vorgaben des Datenblattes der CPU ab. Diese Vorgaben besagen, dass der Low-Pegel eine Spannung von minimal -0,3V und maximal 0,45V haben darf und der High-Pegel nicht um mehr als -0,6V und +0,3V von der Versorgungsspannung abweichen darf. Abhilfe würde durch die Verwendung von SMD-Widerständen aufgrund geringerer parasitärer Kapazitäten und Induktivitäten und den Wechsel auf schnellere CMOS-Gatter für die Takterzeugung schaffen. Die Gatter vom Typ HCT bzw. AHCT weisen kürzere Schaltzeiten auf, besitzen jedoch die gleichen Schwellen wie die TTL-Gatter vom Typ LS.

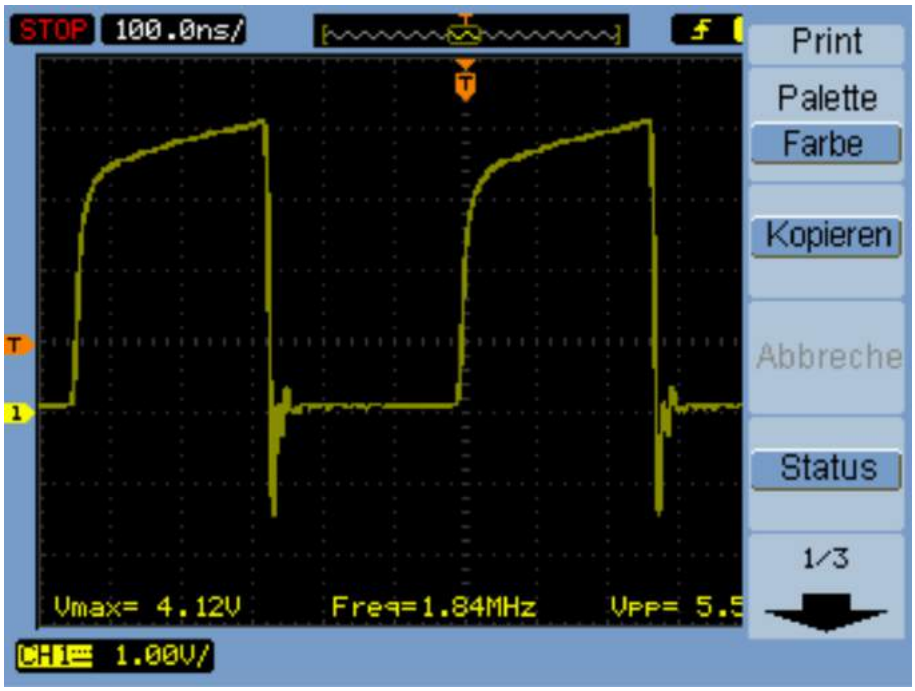


Abbildung 236: Taktsignal

8.5 Programmierung

8.5.1 Programmierung des EPROMs

Die Programmierung des EPROMS wird mit einem Programmiergerät vorgenommen. Dieses Programmiergerät vom Typ MiniPRO TL866 wird mittels USB mit einem Computer verbunden. Um die Software MiniPro Programmer V6.1 für das Programmiergerät zu installieren, muss die beigefügte CD ausgeführt werden. Bei der Installation können die Standardeinstellungen übernommen werden. Ist die Installation ausgeführt worden, kann die Programmierung erfolgen. Dafür müssen die Speicherzellen entsprechend der vorgesehenen Adressierung mit HEX Code befüllt werden. Dieser Hex-Code kann entweder aus einem File mit File -> Open geladen werden oder direkt in die Speicherzellen eingetragen werden. Um den EPROM flashen zu können, muss in der Software mit Select IC -> Search and Select der zu programmierende IC ausgewählt werden. Mit Device -> Program wird mithilfe des über USB mit dem PC verbundenen Programmiergeräts das Programm in den EPROM geschrieben.

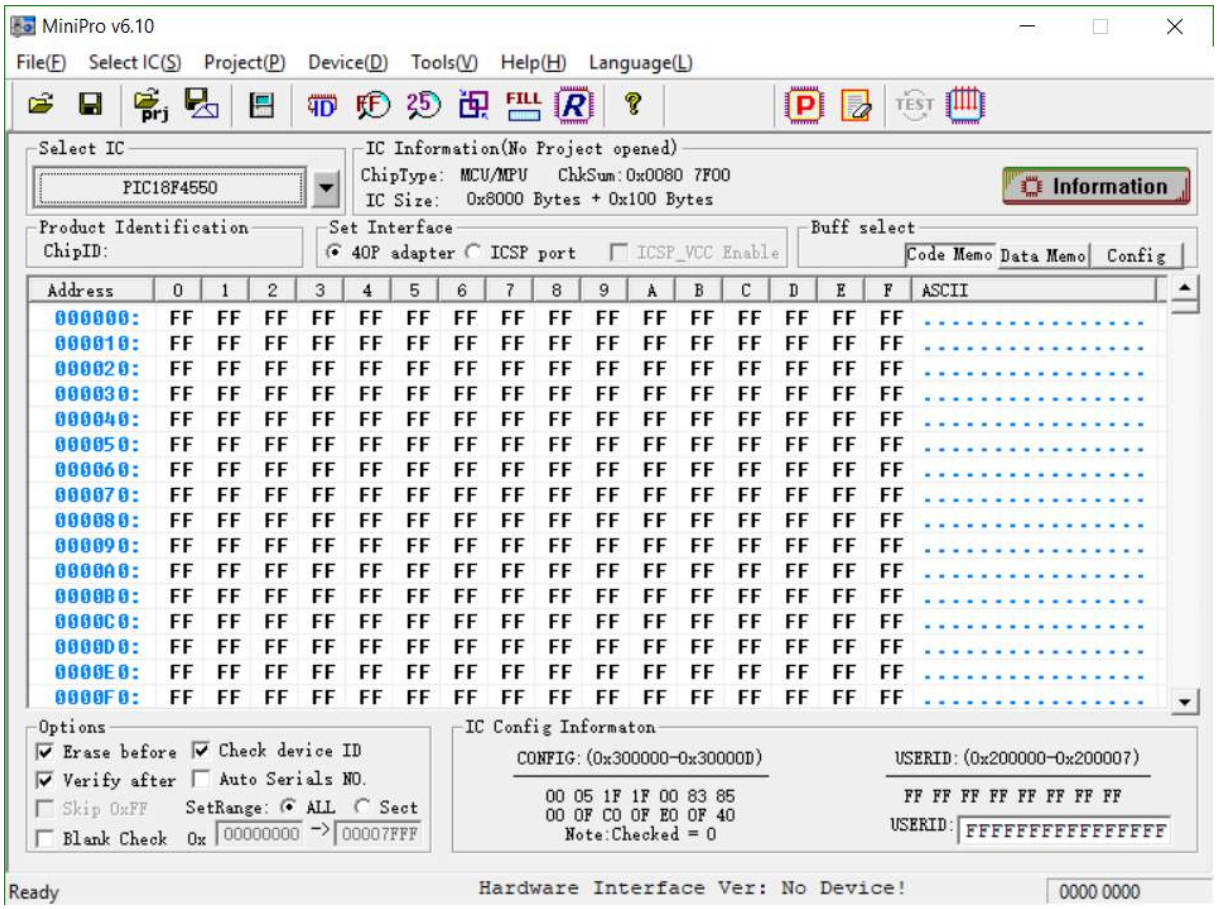


Abbildung 237: EPROM Programmierung - Mini Pro V6.10

8.5.2 Z80 Assembler

Grundsätzlich kann die Übersetzung des Assemblercodes in Intel HEX-Code von Hand erfolgen. Bei größeren Programmen empfiehlt es sich, einen Assembler zu verwenden, wie etwa das Crossware Embedded Development Studio, welcher sowohl als Assembler als auch als Simulator verwendet werden kann. Da dieser Assembler standardmäßig für die Anwendung mit dem MPF 1B konfiguriert ist, muss die Startadresse auf 0000 statt 1800 geändert werden ebenso wie die Größe des Speichers und die Auswahl des EPROMS als Programmspeichermedium. Bei der Übersetzung des Programms liefert der Assembler ein HEX-File, welches für die Programmierung des EPROMS mit der dafür vorgesehenen Software MiniPro Programmer geeignet ist. Genaue Informationen zu Installation und Anwendung liegen der Software in englischer Sprache bei.

8.6 Software und Analyse

8.6.1 Beschreibung der Hardware

Eine genaue Beschreibung der Hardware ist dem Kapitel Aufbau des Z80 Minimalsystems und Beschreibung der Baugruppen zu entnehmen.

8.6.2 Der Von-Neumann Zyklus

Der von-Neumann-Zyklus, benannt nach dem österreich-ungarischen, später US-amerikanischen Mathematiker János/Johann/John von Neumann, beschreibt den Ablauf der Befehlsverarbeitung in einer entsprechend den Vorschlägen von Neumanns entworfenen Architektur. Diese besteht aus einer Rechenwerk mit der ALU (Arithmethic Logic Unit), einer Control Unit (dem Steuerwerk), einem Bussystem, einem Speicherwerk, welches sowohl Daten als auch Instruktionen gleichwertig speichert und den Ein- und Ausgabeeinheiten. Die Befehlsausführung wird in 5 Schritten vorgenommen:

1. **Instruction Fetch**: Instruktion aus den Programmspeicher in das Befehlsregister laden

2. **Instruction Decode**: Der Befehlsdecoder decodiert den HEX-Code aus dem Befehlsregister, sodass der Controller Sequenzer die nötigen Steuersignale erzeugen kann.

3. **Operanden Fetch**: Entsprechend dem decodierten Befehl werden, wenn erforderlich, die nötigen Operanden aus dem Speicher geladen.

4. **Execute**: Ausführung des Befehls

5. **Write Back**: Die erhaltenen Ergebnisse werden, wenn im Befehl enthalten, in die Register oder in den Arbeitsspeicher zurückgeschrieben.

☞ Anmerkung: Für den von-Neumann Zyklus existieren mehrere verschiedene Varianten. Eine Variante beinhaltet das Zurückschreiben der Ergebnisse als fünften Teil, in der zweite Variante ist das Zurückschreiben ein Teil der Ausführung und der letzte Schritt das Laden der Adresse für den nächsten Befehl, also das Aktualisieren des Program Counters. In anderen Varianten wiederum besteht der von-Neumann Zyklus aus nur vier Schritten und endet mit der Ausführung.

8.6.3 Vorbereitung zur Analyse der Z80-Befehlsabarbeitung

8.6.3.1 Installation der DigiView-Software

Um die mittels Logikanalysator aufgenommenen Signalverläufe auf einem PC anzeigen zu können bzw. die Analyse durchzuführen, muss das Programm DigiView installiert werden, welche auf der dem Logikanalysator beigelegten Installations-CD enthalten ist. Bei der Installation der Software muss nach der vorgenommenen Installation noch der Treiber für den Logikanalysator nachinstalliert werden. Dies geschieht üblicherweise über die Systemsteuerung, wo als Quelle für den Treiber die CD zu wählen ist.

8.6.3.2 Kanalkonfiguration in DigiView

Um die Signale erfassen und analysieren zu können, muss für jeden einzelnen Kanal, also für jeden Eingang des Logikanalysators, eine Bezeichnung vergeben werden. Für die anschließende Analyse muss festgelegt werden, um welche Art von Signal es sich handelt. Grundsätzlich wird zwischen einzelnen Leitungen (Boolean) und zwischen einem Bündel von Leitungen (Bus) unterschieden.

Konfiguration für das Programm PIO Test #2:

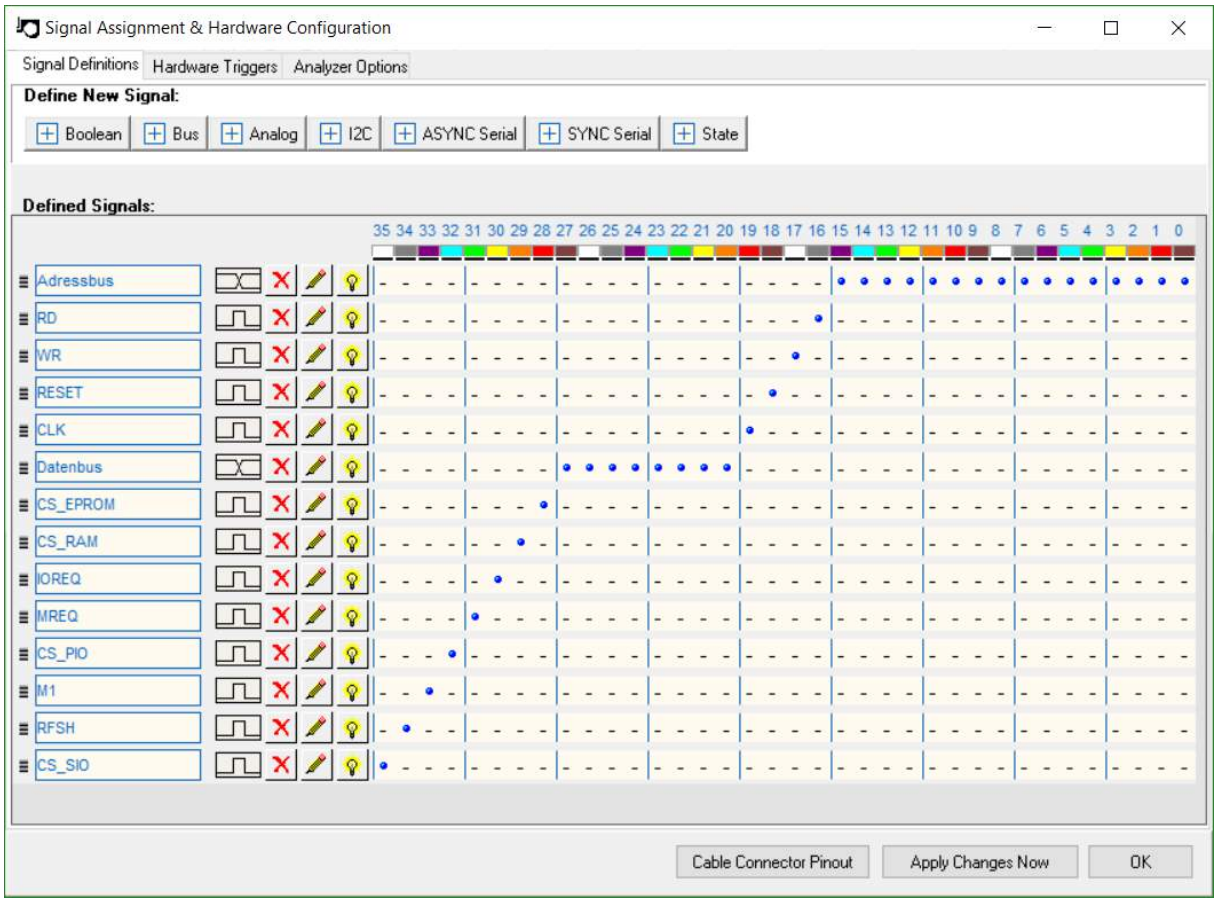


Abbildung 238: DigiView Kanalkonfiguration PIO Test 2

☞ Anmerkung: Diese in der Abbildung gezeigte Belegung ist beispielhaft und entspricht nur der Belegung der Kanäle zur Erfassung des Bustimings des Programms PIO_TEST#2. Die erfassten Signale bzw. die Auswahl und Belegung der Kanäle unterscheidet sich bei den jeweiligen Programmen, weshalb sie bei jedem Einzelnen separat angeführt ist.

8.6.3.3 Festlegung der Triggerbedingung in DigiView

Für das Auslösen des Aufzeichnungsvorgangs muss wie bei einem Oszilloskop ein Trigger gesetzt werden. Dieser Trigger wird als Hardware Trigger ausgeführt, die Triggerung wird mittels logischer Verknüpfung eines oder mehrerer Kanäle unter dem Menüpunkt Config → Trigger erstellt.

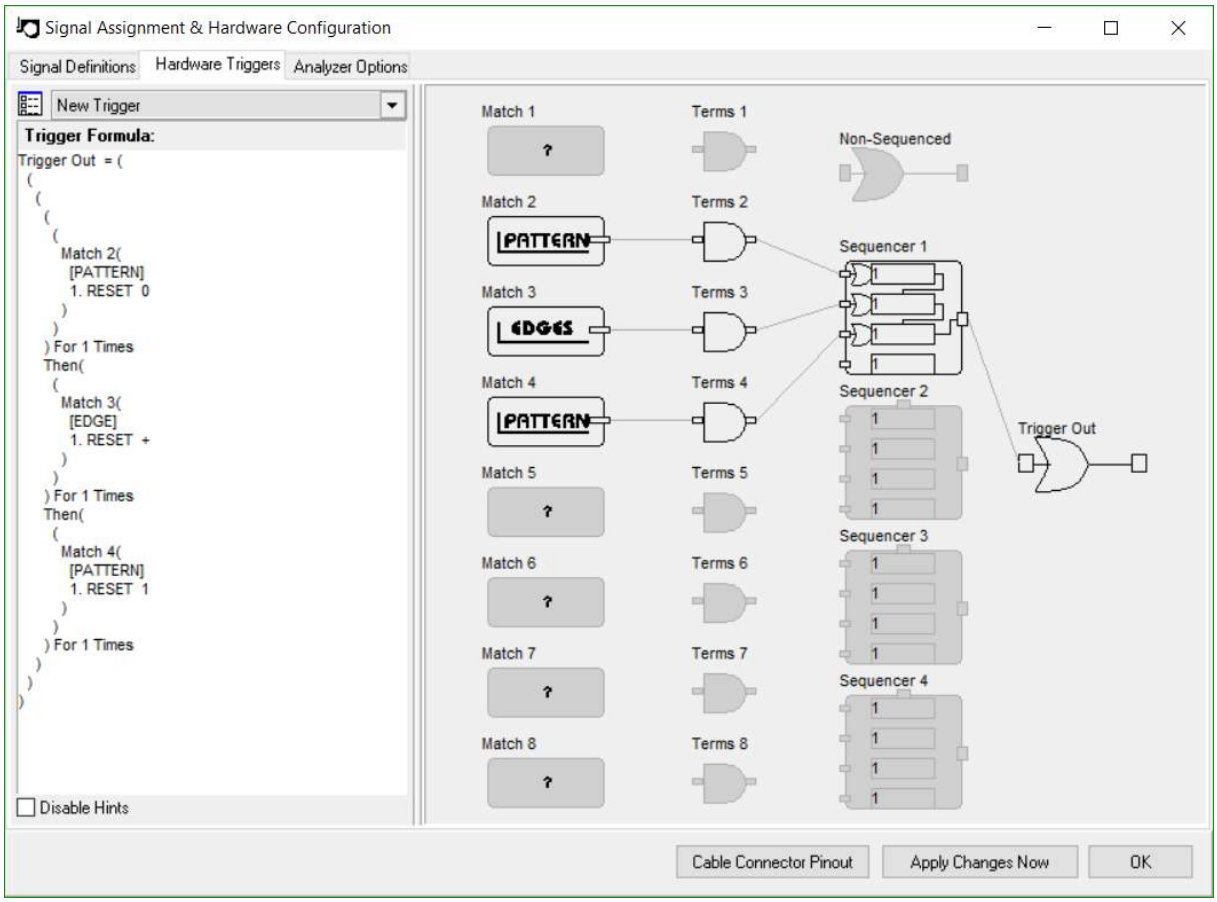


Abbildung 239: DigiView Trigger

Die in der obigen Abbildung dargestellte Trigger-Bedingung besteht aus 3 Elementen: Der erste Teil der Bedingung wird erfüllt, sobald der Kanal Reset, welcher dem Ausgang der Reset-Schaltung des Minimalsystems entspricht, für eine Zeit von mindestens 10us (Minimum: 3 Taktzyklen entspricht 1,61us) LOW ist. Folgt darauf eine positive Flanke und ein HIGH von 10us, so sind auch der 2. und 3. Teil der Bedingung erfüllt, der Trigger wird ausgelöst und die Aufzeichnung startet. Der erste und zweite Teil der Bedingung vermeiden, dass der Logikanalysator auf ein Störsignal triggert, der dritte Teil prüft auf unerwünschte, störungsbedingte Reset-Impulse und ist nicht zwingend erforderlich.

8.6.4 Verbindung des Logikanalysators mit dem Z80 Minimalsystem

Der Logikanalysator, welcher in der Laborübung verwendet wird, besitzt 36 Kanäle, die über 2 Stecker mit jeweils 18 Kanälen und 2 Masseverbindungen nach außen geführt werden. Die einzelnen Leitungen besitzen eine festgelegte farbige Markierung, um die Zuordnung zu erleichtern.

Messleitungen/Kanäle des Logikanalysators:

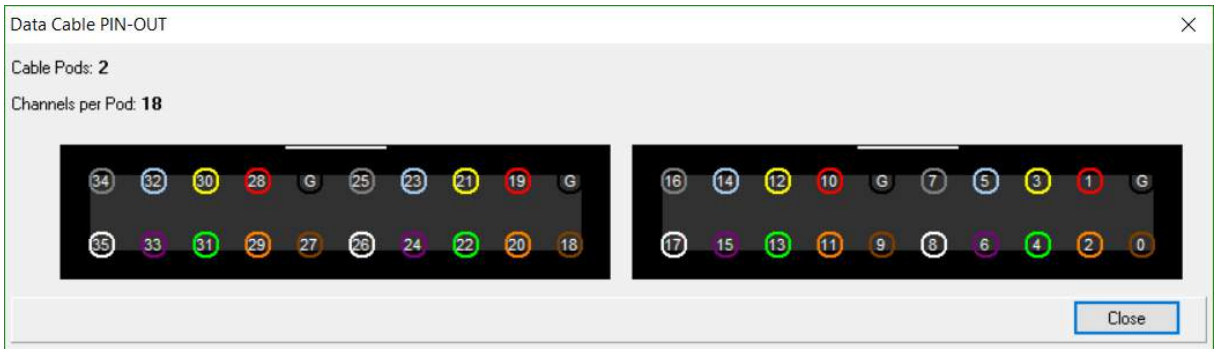


Abbildung 240: DigiView Kanäle des Logikanalysators

Entsprechend dieser Zuordnung und der zuvor festgelegten Kanäle müssen die herausgeführten Pins von Adressbus, Datenbus, CE-Logik und der Steuerleitungen des Z80 Minimalsystems mit den Leitungen der einzelnen Kanäle verbunden werden. Die Zuordnung der auf die Stiftleisten herausgeführten Leitungen kann entsprechend dem Aufdruck auf der Platine und den Schaltplänen vorgenommen werden, es sollte aber dringend darauf geachtet werden, dass auch GND (fixe Belegung am Logikanalysator, bezeichnet mit G) verbunden wird.

8.6.4.1 Belegung der herausgeführten Leitungen auf dem Minimalsystem

☞ Anmerkung: Um etwaige Fehler bei der späteren Interpretation der Timing Diagramme zu vermeiden, sollte angemerkt werden, dass sämtliche Leitungen für Daten-, Steuer- und Adressbus invertierte Ein- und Ausgänge besitzen.

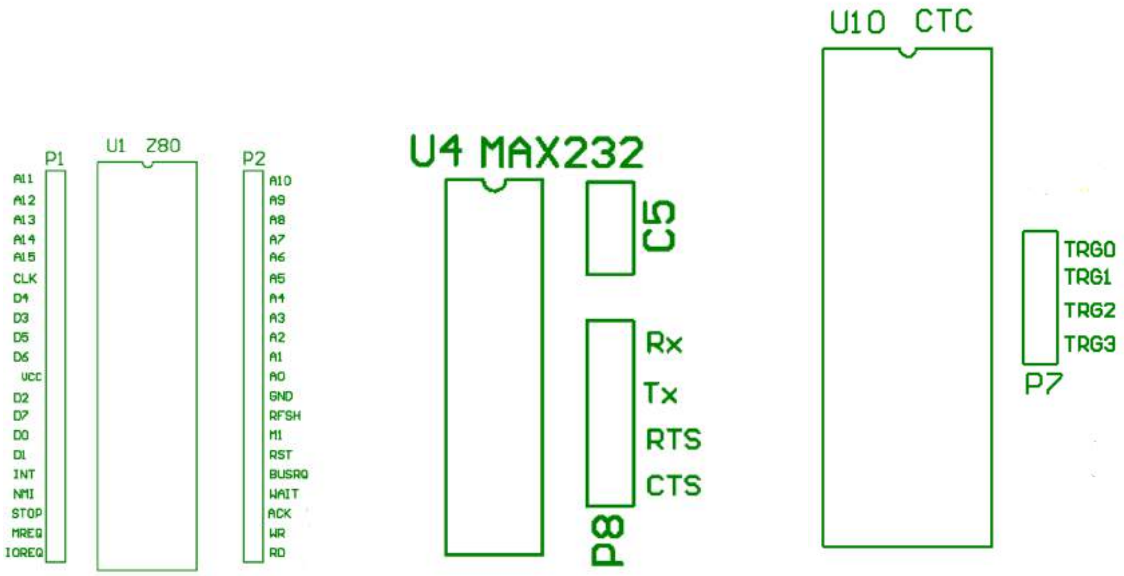


Abbildung 241: Belegung der Stiftleisten an CPU, CTC und UART

Wie dem Bild zu entnehmen ist, werden an den Stiftleisten P1 und P2 der gesamte Adress- und Datenbus und die meisten Leitungen des Steuerbusses herausgeführt. An der Stiftleiste P8 werden die Daten- und Steuerleitungen für die RS232-Schnittstelle abgerufen werden. Die Triggerleitungen des CTC sind an der Stiftleiste P7 herausgeführt. Die Stiftleisten P5 und P6 sind mit den Pins des PIO verbunden und nicht beschriftet, da sie im Regelfall nicht verwendet werden. Sollten sie benötigt werden, kann die Belegung den Schematics entnommen werden.

8.6.5 PIO Testprogramm

8.6.5.1 Aufgabenstellung

Es ist mithilfe eines Logikanalysators der Befehlsablauf einer Von-Neumann-Architektur anhand des Z80 Minimalsystems zu analysieren. Das zu verwendende Programm für den Zilog Z80 lautet auf dem Namen PIO_TEST#2. Es liest mithilfe des PIO (8255) die Schalterstellung aus und zeigt diese über die LEDs an. Es soll eine Erfassung des Befehlsablaufes ab dem Reset vorgenommen werden. Anschließend zu analysieren sind vor allem die Instruktionen, die in das Control-Register des PIO geschrieben werden. Weiters soll eine Messung der Zugriffszeiten auf die einzelnen Komponenten des Systems, in diesem Fall auf EPROM und PIO, erfolgen und die Gatterlaufzeit der CE-Logik ermittelt werden.

8.6.5.2 Source Code

Das Programm für den Z80 wird in Assemblersprache verfasst. Um das Programm ausführen zu können, muss die Assemblersprache in Hex-Code übersetzt werden und dieser mit einem Programmiergerät in ein EPROM geschrieben werden.

Listing 25: Z80 PIO Test

```
1  ;*****************************************************************************
2  ;* Z80 Assembler program *
3  ;* Josef Reisinger *
4  ;* josef.reisinger@htl-hl.ac.at *
5  ;* 26/04/2015 *
6  ;*****************************************************************************
7
8  ;************************ HARDWARE IO ADR ***********************************
9  ; PIO 82C55 I/O
10 PIO1A: EQU $80 ; INPUT - DIP SWITCHES
11 PIO1B: EQU $81 ; OUTPUT - LEDS
12 PIO1C: EQU $82 ; (INPUT)
13 PIO1CONT: EQU $83 ; CONTROL BYTE PIO 82C55
14
15 ;*************************** CONSTANTS **************************************
16 RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
17
18
19 ;*****************************************************************************
20 ;* START AFTER RESET, *
21 ;* Function....: ready system and restart *
22 ;*****************************************************************************
23    ORG $0000
24    ;DI  ; Disable interrupt
25    ;LD SP,RAMTOP  ; Set stack pointer to top off ram
26    LD A,$99  ; PA0-PA7=IN (DIP SWITCHES), PB0-PB7=OUT (LEDS),
27 ; PC0-PC7=IN, Mode 0 Selection
28
29    OUT (PIO1CONT),A
30    IN A,(PIO1CONT)
31
32 AGAIN:
33    IN A,(PIO1A)  ; Read actual status of Switches (PA0-PA7)
34    OUT (PIO1B),A  ; Output Status to LEDs (PB0-PB7)
35    JP AGAIN  ; Endlos
```

8.6.5.3 HEX-Code

Der hier abgebildete Source Code ist um die Speicheradressen und um den Hex Code der jeweiligen Befehle ergänzt. Abgespeichert wird diese Datei als List File (.lst).

Listing 26: Z80 PIO Test HEX-Code

```
1  1 ;*****************************************************************************
2  2 ;* Z80 Assemblerprogramm *
3  3 ;* Josef Reisinger *
4  4 ;* josef.reisinger@htl-hl.ac.at *
5  5 ;* 26/04/2015 *
6  6 ;*****************************************************************************
7  7
8  8
9  9 ; ---------------------------- PIO 82C55 I/O --------------------------------
10 10 0080 PIO_A: EQU $80 ; (INPUT)
11 11 0081 PIO_B: EQU $81 ; (OUTPUT) OUT TO LEDS
12 12 0082 PIO_C: EQU $82 ; (INPUT) IN from DIP SWITCHES
13 13 0083 PIO_CON: EQU $83 ; CONTROL BYTE PIO 82C55
14 14
15 15 ; ---------------------------- CTC Z80 Timer Counter -------------------------
16 16 0000 CTC0 EQU $00 ; Channel 0
17 17 0001 CTC1 EQU $01 ; Channel 1
18 18 0002 CTC2 EQU $02 ; Channel 2
19 19 0003 CTC3 EQU $03 ; Channel 3
20 20
21 21 ; ------------------------- SIO (USART) ---------------------------------------
22 22 0040 SIO_A_D: EQU $40 ; Channel A Data Register
23 23 0041 SIO_B_D: EQU $41 ; Channel B Data Register
24 24 0042 SIO_A_C: EQU $42 ; Channel A Control Register
25 25 0043 SIO_B_C: EQU $43 ; Channel B Control Register
26 26
27 27
28 28 ;-------------------------- CONSTANTS ---------------------------------------
29 29 FFFF RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
30 30
31 31
32 32 ;*****************************************************************************
33 33 ;* START AFTER RESET, *
34 34 ;* Function....: ready system and restart *
35 35 ;*****************************************************************************
36 36 0000 ORG $0000
37 37 0000 F3 DI ; Disable interrupt
38 38 0001 31 FF FF LD SP,RAMTOP ; Set stack pointer to top off ram
39 39 0004 3E 99 LD A,$99 ; PA0-PA7=IN (DIP SWITCHES), PB0-PB7=OUT (LEDS), PC0-PC7=IN, Mode
       0 Selektion
40 40 0006 D3 83 OUT (PIO_CON),A
41 41 0008 DB 83 IN A,(PIO_CON)
42 42 000A DB 80 AGAIN: IN A,(PIO_A) ; Read actual status of Switches (PA0-PA7)
43 43 000C D3 81 OUT (PIO_B),A ; Output Status to LEDs (PB0-PB7)
44 44 000E C3 0A 00 JP AGAIN ; Endless
45 45
46 46
47 47
48
49 Symbol table:
50
```

```
51  AGAIN 000A CTC0 0000 CTC1 0001 CTC2 0002
52  CTC3 0003 PIO_A 0080 PIO_B 0081 PIO_C 0082
53  PIO_CON 0083 RAMTOP FFFF SIO_A_C 0042 SIO_A_D 0040
54  SIO_B_C 0043 SIO_B_D 0041
55  14 symbols
```

8.6.5.4 Konfiguration des Logikanalysators

Die Kanalbelegung des Logikanalysators ist der Erläuterung der Bedienung des Logikanalysators in Abschnitt 8.6.3.2 zu entnehmen.

8.6.5.5 Analyse

Die Beschreibung der einzelnen Zyklen erfolgt entsprechend der Bezeichnung der einzelnen Marker T, A-D, X und Y. Bei dieser Abbildung wurde ein Marker mit der Bezeichnung Z ergänzt.

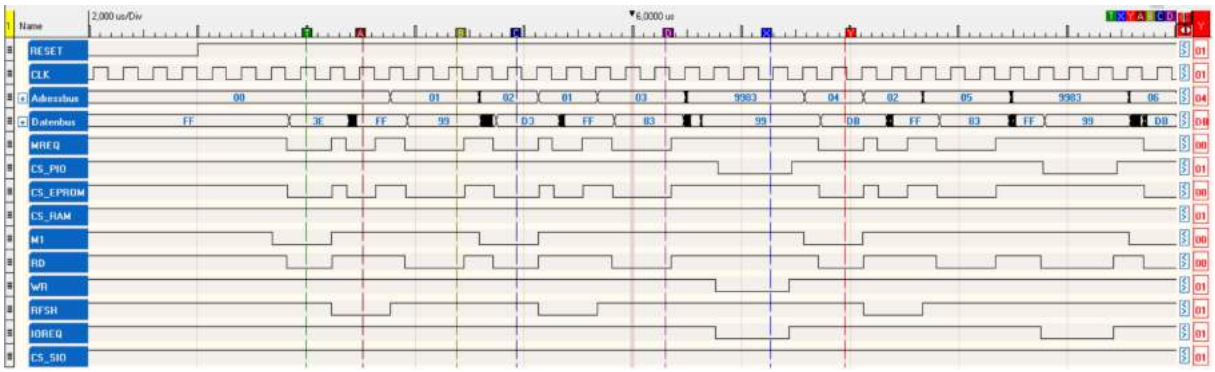


Abbildung 242: DigiView PIO Test 2 Teil 1

Marker T: Mit dem Reset wurde der Program Counter auf die Adresse 0000 zurückgesetzt. Die Adresse 0000 ist als die erste Adresse des auszuführenden Programms festgelegt, weshalb die CPU einen Opcode Fetch durchführt. Dafür aktiviert die CPU die Leitungen M1 (Machine Cycle One), MREQ (Memory Request) und RD (Read). Da die Leitungen bzw. Die Ein- und Ausgänge von Adress-, Daten- und Steuerbus invertiert sind, werden sie auf Low gesetzt. Die CE-Logik enabled anhand des Memory Requests und des 16. Bits des Datenbusses den EPROM. Im EPROM wird der Inhalt der Speicherzelle mit der Adresse 0000 an den Datenbus gelegt. Vom Datenbus wird der Opcode 3E in das Befehlsregister geladen und vom Befehlsdecoder decodiert. Der Befehl mit dem Opcode 3E ist ein 8-Bit-Transferbefehl, welcher einen Wert aus dem Speicher in den Akkumulator lädt.

Marker A: Da die CPU auch die Verwendung dynamischer RAMs unterstützt und deren Speicherinhalt flüchtig ist, wird alle 5 s ein Refresh aufgelöst, um die Speicherzellen aufzufrischen und somit den Speicherinhalt zu halten. Diese Funktion wird standardmäßig ausgeführt, findet aber beim Z80 Minimalsystem keine Anwendung.

Marker B: Nach dem Refresh wird der Inhalt des Program Counters wiederhergestellt, um 1 erhöht und an den Datenbus gelegt. Um den ersten Operanden mittels Speicher-Lesezugriff aus dem EPROM zu holen, wird der Inhalt des Befehlszählers an den Adressbus gelegt und die CPU gibt einen Memory Request für einen Datenaustausch mit dem Speicher und ein Read für einen Lesevorgang aus. Anhand des Memory Requests und des 16. Bits des Adressbusses, welches 0 ist, generiert die CE-Logik am Ausgang ein CE-Signal für den EPROM. Dieser legt nach einigen Nanosekunden (Access Time) den Inhalt der Speicherzelle, 99, an den Datenbus. Von dort werden die Daten in den Akkumulator geladen. Die in den Akku geladene Konstante ist für die Konfiguration des PIOs (8255) bestimmt. Der Port B des Parallel Input/Output Controllers soll als Output für die Ausgabe über LED konfiguriert werden, Port A als Input für das Einlesen der Schalterstellungen des DIL-Schalters, als Modus wird der Mode 0 gewählt. Die Ausführung des Befehls ist abgeschlossen, wenn der Befehlszähler inkrementiert ist und die Adresse für den nächsten Befehl am Adressbus anliegt.

Marker C: Mit diesem Zyklus wird ein Opcode Fetch durchgeführt. Der dabei erhaltene Opcode D3 entspricht einem Ausgabebefehl für die Peripherie.

Marker D: Nach dem Inkrementieren des Program Counters wird der Inhalt am Adressbus ausgegeben. Von dieser Adresse soll der erste Operand, die Zieladresse für den Ausgabebefehl geladen werden. Mittels Speicher-Lesezugriff wird die Adresse 83 vom EPROM geholt und im W-Register abgelegt.

Marker X: Nachdem sowohl die Adresse als auch die zu übertragenden Daten aus dem Speicher geholt wurden, kann die Ausgabe der Daten und damit die Konfiguration des PIO erfolgen. Dazu wird die Adresse 83 an den Adressbus gelegt. Die Adressen für die Peripherie sind immer nur 8 Bit breit, der Datenbus jedoch 16 Bit. Somit entspricht das Low Byte der Adresse 83 und das nicht verwendete High Byte wird mit den momentan auf dem Datenbus befindlichen Daten beaufschlagt. Dadurch ergibt sich die irreführende Adresse 9983. Für das Beschreiben des Control Registers des PIOs werden von der CPU die Signale IOREQ (I/O Request) und WR (Write) erzeugt. Anhand der Adresse und der Steuersignale enabled die CE-Logik nach einer Laufzeit den PIO und die Datenübertragung kann stattfinden.

Marker Y: Nach dem Inkrementieren des Program Counters wird die dabei erhaltene Adresse 0004 an den Adressbus gelegt und ein Opcode Fetch durchgeführt. Der am Datenbus anliegende Befehl mit dem Hex-Code DB ist für das Auslesen von Peripherieeinheiten und das anschließende Speichern der ausgelesenen Information im Akku zuständig.

☞ Anmerkung: Nach jedem Zugriff auf den Programmcode, sei es zum Abruf des Opcodes oder zum Laden von Operanden, wird der Program Counter inkrementiert. Beim nächsten Zugriff auf den Programmcode wird der Inhalt des PC dann auf dem Adressbus ausgegeben.

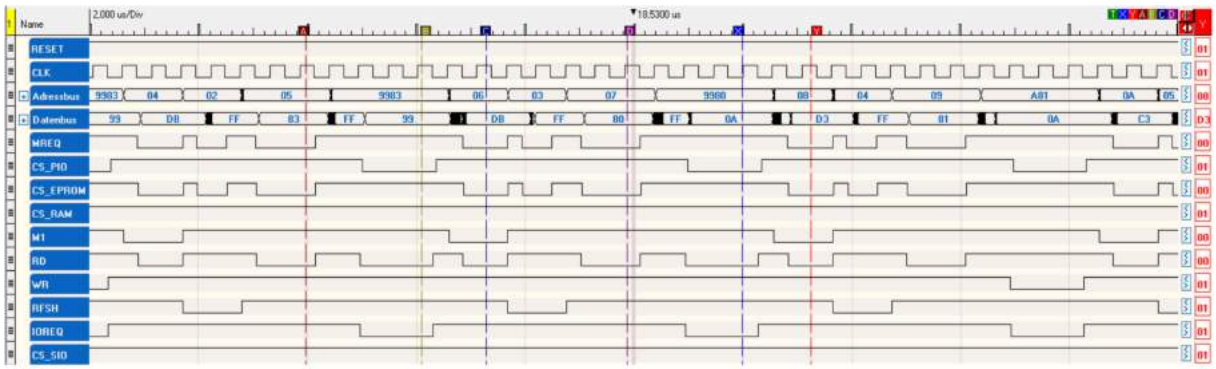


Abbildung 243: DigiView PIO Test 2 Teil 2

Marker A: Mittels Speicher-Lesezugriff wird die Adresse 83 zur Abfrage der Daten im Control Register des PIO aus dem EPROM geladen.

Marker B: Für die Ausführung des Befehls legt die CPU die Adresse 83 an den Datenbus. Für die Abfrage der Daten erzeugt sie einen I/O Request und ein Read und die CE-Logik enabled den PIO. Der PIO legt nun den Inhalt des Control Registers an den Datenbus, von wo die CPU die Daten in den Akku lädt.

Marker C: Mit dem Opcode Fetch wird wieder der Opcode DB in das Befehlsregister der CPU geladen.

Marker D: Die Adresse 80 wird als erster Operand aus dem EPROM geholt.

Marker X: Das Portregister A des PIO wird über den Adressbus adressiert und die Daten am Datenbus ausgegeben. Von dort werden sie in den Akku geladen. Die Daten 0A entsprechen der Schalterstellung des 8-fach DIL-Schalters.

Marker Y: Mit dem Opcode Fetch wird der Opcode D3 aus dem EPROM geladen. Der sich im Befehlsregister bzw im Befehlsdecoder befindliche Befehl D3 entspricht einem Ausgabebefehl für die Peripherie.

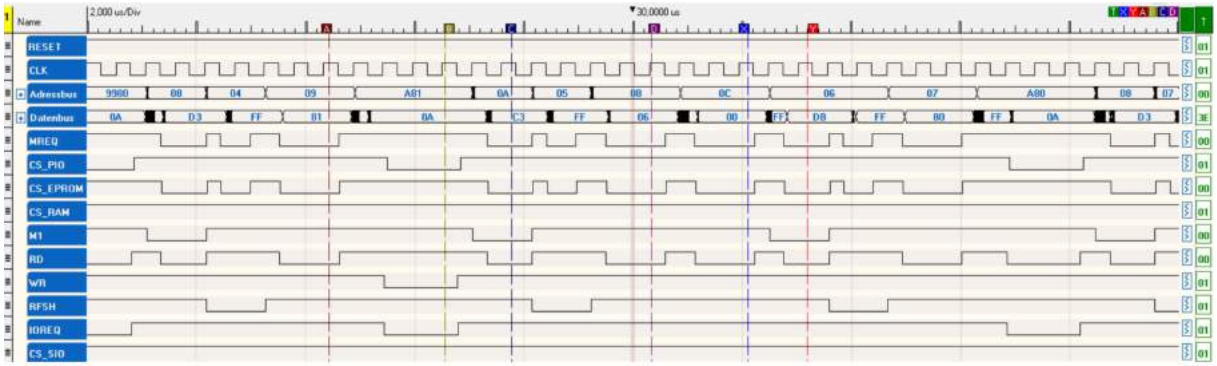


Abbildung 244: DigiView PIO Test 2 Teil 3

Marker A: Die Zieladresse 81 für die Ausgabe wird aus dem EPROM geladen.

Marker B: Der Port B des PIO wird mit 81 adressiert und die Daten an den EPROM übermittelt. Nun wird die hexadezimale Zahl 0A binär über das LED-Array ausgegeben.

Marker C: Die CPU führt einen Opcode Fetch durch, bei dem der Befehl mit dem Opcode C3 geladen wird. Dieser steht für einen Sprungbefehl auf eine durch den Programmierer vorzugebende Adresse.

Marker D: Da die Adresse für den Sprung 16 Bit groß ist, der Datenbus aber nur 8 Bit breit, wird die Adresse in zwei Schritten geladen. Im ersten Schritt wird das Low Byte der Sprungadresse mit einem Speicher-Lesezugriff aus dem EPROM geholt und im W-Register abgelegt.

Marker X: Das High Byte wird in einem weiteren Speicher-Lesezugriff aus dem EPROM geholt und im Z-Register abgelegt. Nun kann der Sprung durchgeführt werden, also der Inhalt des WZ-Registers in den Befehlszähler geladen werden.

Marker Y: Nach dem Sprung legt der Sequencer den Inhalt des Program Counters, die Adresse 0006, an den Adressbus und die CPU führt einen Opcode Fetch durch. Geladen wird der Opcode DB, der Daten von der Peripherie abfragt.

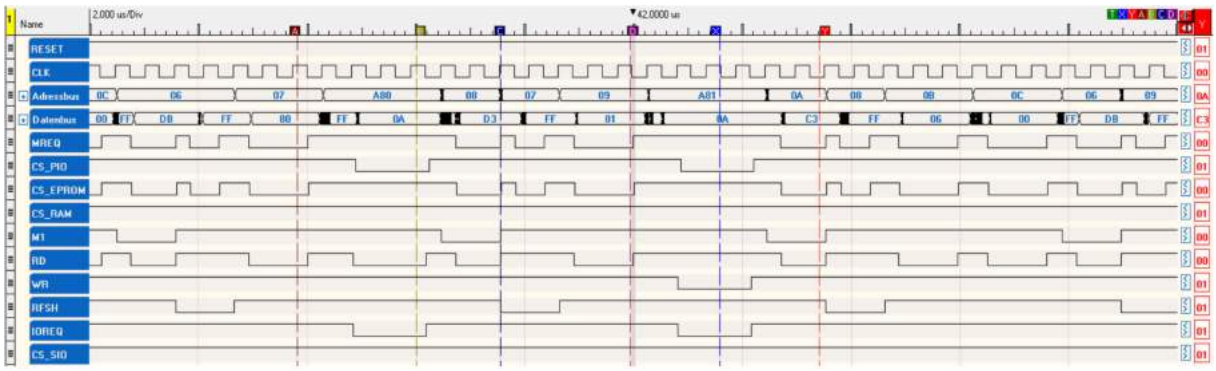


Abbildung 245: DigiView PIO Test 2 Teil 4

Marker A: Für die Abfrage der Daten wird die 8-Bit-Adresse 80 aus dem EPROM geladen.

Marker B: Die Schalterstellung des 8-fach DIL-Schalters wird vom Portregister A des PIO mit der Adresse 80 abgerufen und in den Akkumulator geladen.

Marker C: die CPU führt einen Opcode Fetch durch und lädt dabei den Befehl mit dem Opcode D3, welcher Daten an die Peripherie ausgibt.

Hier wird die Analyse beendet, da keine neuen Befehle mehr ausgeführt werden, sondern in einer Endlosschleift der Port A des PIO ausgelesen und die erhaltene Information über den Port B wieder ausgegeben wird.

8.6.5.6 Zugriffszeiten auf den EPROM laut Datenblatt

Auszug aus dem Datenblatt (ST M27256-1)

Figure 5. Read Mode AC Waveforms

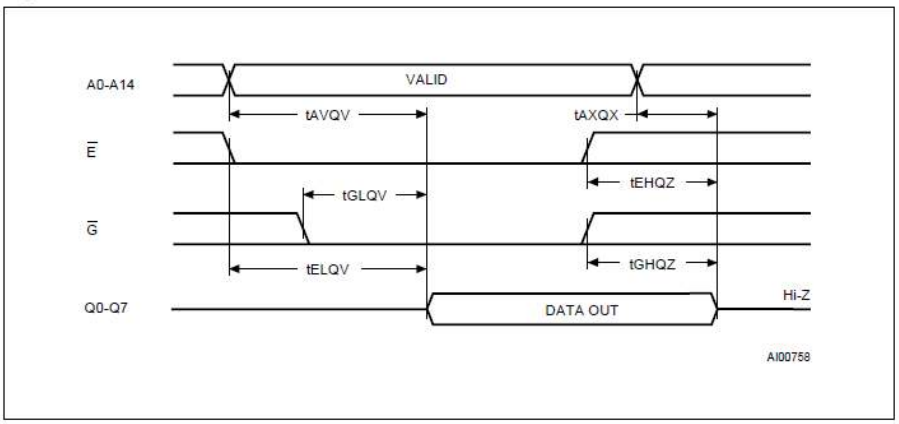


Abbildung 246: Datenblattauszug EPROM [21]

t_{ELGV} = 200 ns maximal t_{GLQV} = 75 ns maximal

8.6.5.7 Ermittlung der Zugriffszeit auf den EPROM

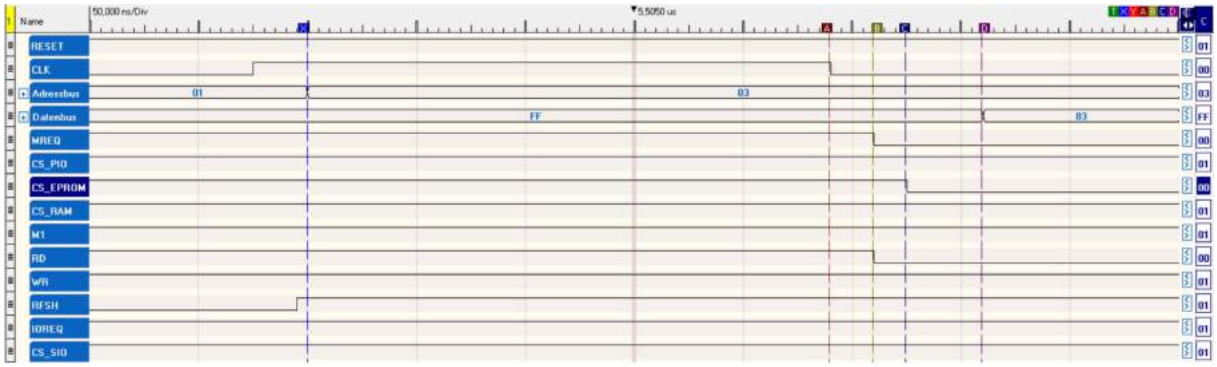


Abbildung 247: Messung Zugriffszeit EPROM

Zu den Zugriffszeiten der CPU auf den RAM können folgende Aussagen getroffen werden:

T(X)= 5,3550μs; T(A)=5,5950μs; T(B)=5,6150μs; T(C)=5,6300μs; T(D)=5,6650μs

Durch diese Messwerte ergibt sich eine Gesamtzugriffszeit vom Anlegen der Adresse der Speicherzelle an den Adressbus bis zur Ausgabe des Inhalts der Speicherzelle auf den Datenbus eine Zeitspanne von 310ns. Die Gatterlaufzeit der CE-Logik, also die Zeitspanne zwischen dem Erzeugen des Memory Requests und des Reads und dem Selektieren des EPROMs, beläuft sich auf 15ns. Im EPROM selbst kommt es zu einer Laufzeit von 25ns,

was der Zugriffszeit entspricht. Da die Maximaldauer zwischen dem Enablen und dem Ausgeben der Daten mit maximal 200ns bzw. 75ns größer ist als die gemessenen 50ns bzw. 25ns, werden diese Vorgaben eingehalten.

8.6.5.8 Zugriffszeit auf den PIO

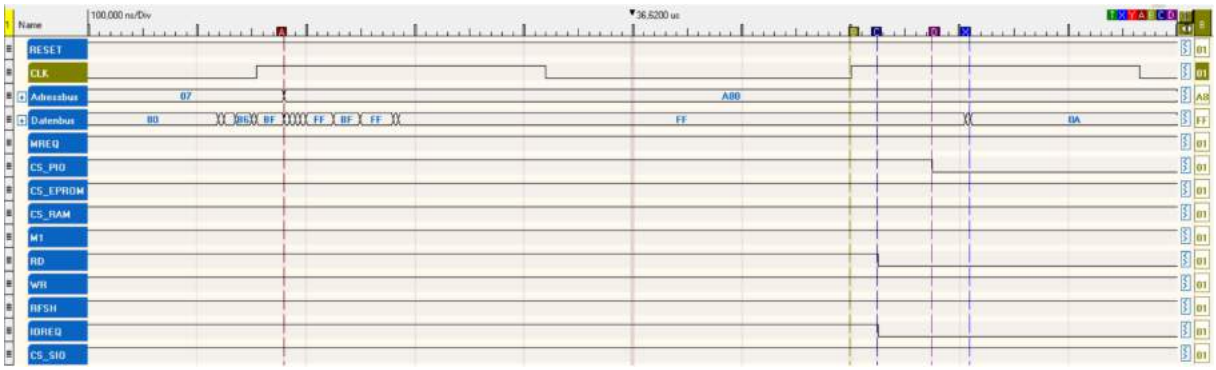


Abbildung 248: Messung Zugriffszeit PIO

Zeiten der einzelnen Messpunkte:

T(A)= 36,300μs; T(B)=36,8200μs; T(C)=36,8450μs; T(D)=36,8950μs; T(X)=36,9300μs

Die gesamte Zugriffszeit vom Anlegen der Adresse des Port A des PIO bis zur Ausgabe der Schalterstellung auf den Datenbus ergibt sich zu 630ns. Vor der positiven Taktflanke bis zum Ausgabezeitpunkt der Steuersignale vergehen 25ns, die Chipselect-Logik besitzt eine Gatterlaufzeit von 50ns. Die Reaktionszeit des PIO vom Zeitpunkt der Auswahl durch die CE-Logik bis zum Anlegen der momentanen Schalterstellung beträgt 35ns. Im Datenblatt wird dafür ein Zeitraum von maximal 120ns angegeben, in dem die Daten gültig sein müssen, somit werden die Grenzwerte eingehalten.

8.6.6 Programm PIO_RAM_COUNTER

8.6.6.1 Aufgabenstellung

Auf dem Z80 Minimalsystem ist das Programm PIO_RAM_COUNTER auszuführen. Mittels Logikanalysator ist ein Timing-Diagramm aufzuzeichnen, welches anschließend analysiert werden soll. Zusätzlich zur Analyse ist eine Messung der Zugriffszeit der CPU auf den RAM durchzuführen.

8.6.6.2 Konfiguration des Logikanalysators

Die Belegung der Kanäle für die Aufzeichnung und die Analyse des Timing-Diagrammes des Programmes PIO_RAM_COUNTER sieht folgendermaßen aus:

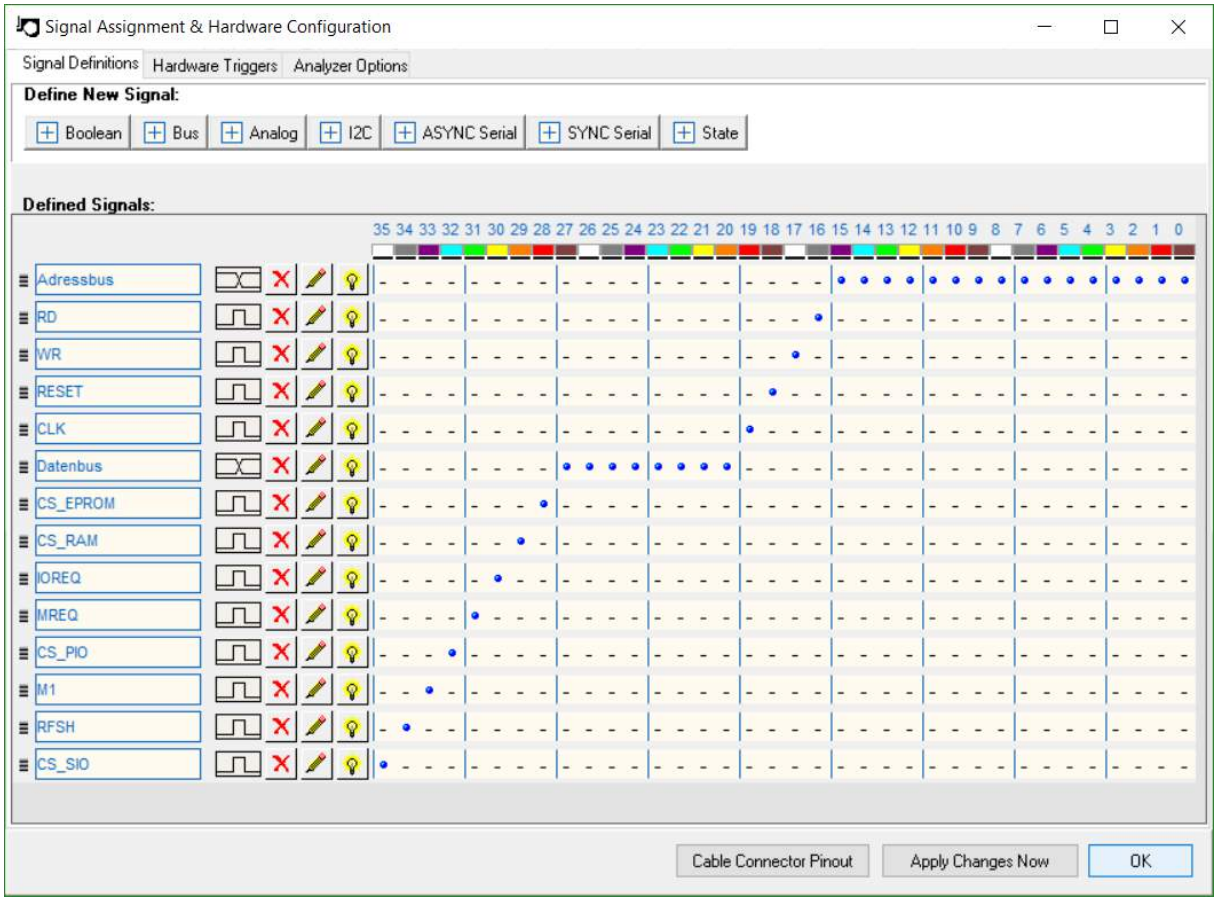


Abbildung 249: DigiView Kanalkonfiguration PIO RAM Counter

8.6.6.3 Assemblercode

Listing 27: Z80 PIO_RAM_COUNTER

```
1  ;************************************************************************
2  ;* Z80 Assembler program *
3  ;* Josef Reisinger *
4  ;* josef.reisinger@htl-hl.ac.at *
5  ;* 10/07/2017 *
6  ;************************************************************************
7
8  ; ---------------------------- PIO 82C55 I/O --------------------------------
9  PIO_A: EQU $80 ; (INPUT)
```

```
10  PIO_B: EQU $81 ; (OUTPUT) OUT TO LEDS
11  PIO_C: EQU $82 ; (INPUT) IN from DIP SWITCHES
12  PIO_CON: EQU $83 ; CONTROL BYTE PIO 82C55
13
14  ; --------------------------- CTC Z80 Timer Counter -------------------------
15  CTC0 EQU $00 ; Channel 0
16  CTC1 EQU $01 ; Channel 1
17  CTC2 EQU $02 ; Channel 2
18  CTC3 EQU $03 ; Channel 3
19
20  ; --------------------------- SIO (USART) -----------------------------------------
21  SIO_A_D: EQU $40 ; Channel A Data Register
22  SIO_B_D: EQU $41 ; Channel B Data Register
23  SIO_A_C: EQU $42 ; Channel A Control Register
24  SIO_B_C: EQU $43 ; Channel B Control Register
25
26
27  ;--------------------------- CONSTANTS ---------------------------------------
28  RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
29  COUNTER: EQU $8000 ; RAM Counter
30
31  ;*****************************************************************************
32  ;* RESET HANDLER *
33  ;* Function: Initialize system and start Main Program *
34  ;*****************************************************************************
35              ORG $0000
36              DI ; Disable interrupt
37              LD SP,RAMTOP ; Set stack pointer
38              JP MAIN  ; jump to Main program
39
40
41  ;*****************************************************************
42  ;* MAIN PROGRAM *
43  ;*****************************************************************
44  ORG $100
45  MAIN: LD A,$99  ; Initialize 8255: PA0-PA7=IN (DIP SWITCHES), PB0-PB7=OUT (LEDS),
46              OUT (PIO_CON),A  ; PC0-PC7=IN, Mode 0 Selection
47          LD A,$01  ; Initialize RAM Counter
48          LD HL,COUNTER  ; Load RAM Counter Address
49          LD (HL),A  ; Store Counter in RAM cell
50  AGAIN:  LD A,(HL)  ; Load RAM Counter
51          OUT (PIO_B),A ; Output RAM Counter to LEDs (PB0-PB7)
52          INC (HL)   ; Increment RAM Counter
53          CALL _WAIT  ; UP Warteschleife aufrufen
54          JP AGAIN  ; Endlos
55
56
57
58  ;*****************************************************************
59  ;* Warteschleife 0,5s *
60  ;*****************************************************************
61  _WAIT: LD D,$FF ;
62  _OUTER: LD E,$FF ;
```

```
63 _INNER: DEC E
64         JP NZ,_INNER
65          DEC D
66          JP NZ,_OUTER
67          RET
```

8.6.6.4 HEX-Code

Listing 28: Z80 PIO_RAM_COUNTER HEX-Code

```
1  1 ;*****************************************************************************
2  2 ;* Z80 Assembler program *
3  3 ;* Josef Reisinger *
4  4 ;* josef.reisinger@htl-hl.ac.at *
5  5 ;* 10/07/2017 *
6  6 ;*****************************************************************************
7  7
8  8 ; ---------------------------- PIO 82C55 I/O --------------------------------
9  9 0080 PIO_A: EQU $80 ; (INPUT)
10 10 0081 PIO_B: EQU $81 ; (OUTPUT) OUT TO LEDS
11 11 0082 PIO_C: EQU $82 ; (INPUT) IN from DIP SWITCHES
12 12 0083 PIO_CON: EQU $83 ; CONTROL BYTE PIO 82C55
13 13
14 14 ; ---------------------------- CTC Z80 Timer Counter -------------------------
15 15 0000 CTC0 EQU $00 ; Channel 0
16 16 0001 CTC1 EQU $01 ; Channel 1
17 17 0002 CTC2 EQU $02 ; Channel 2
18 18 0003 CTC3 EQU $03 ; Channel 3
19 19
20 20 ; ------------------------- SIO (USART) ---------------------------------------
21 21 0040 SIO_A_D: EQU $40 ; Channel A Data Register
22 22 0041 SIO_B_D: EQU $41 ; Channel B Data Register
23 23 0042 SIO_A_C: EQU $42 ; Channel A Control Register
24 24 0043 SIO_B_C: EQU $43 ; Channel B Control Register
25 25
26 26
27 27 ;-------------------------- CONSTANTS ------------------------------------
28 28 FFFF RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
29 29 8000 COUNTER: EQU $8000 ; RAM Counter
30 30
31 31 ;*****************************************************************************
32 32 ;* RESET HANDLER *
33 33 ;* Function: Initalize system and start Main Programm *
34 34 ;*****************************************************************************
35 35 0000 ORG $0000
36 36 0000 F3 DI ; Disable interrupt
37 37 0001 31 FF FF LD SP,RAMTOP ; Set stack pointer
38 38 0004 C3 00 01 JP MAIN ; jump to Main program
39 39
40 40
41 41 ;*****************************************************************
42 42 ;* MAIN PROGRAM *
```

```
43 43 ;*************************************************************************
44 44 0100 ORG $100
45 45 0100 3E 99 MAIN: LD A,$99 ; Initialize 8255: PA0-PA7=IN (DIP SWITCHES), PB0-PB7=OUT (
       LEDS),
46 46 0102 D3 83 OUT (PIO_CON),A ; PC0-PC7=IN, Mode 0 Selection
47 47 0104 3E 01 LD A,$01 ; Initialize RAM Counter
48 48 0106 21 00 80 LD HL,COUNTER ; Load RAM Counter Address
49 49 0109 77 LD (HL),A ; Store Counter in RAM cell
50 50 010A 7E AGAIN: LD A,(HL) ; Load RAM Counter
51 51 010B D3 81 OUT (PIO_B),A ; Output RAM Counter to LEDs (PB0-PB7)
52 52 010D 34 INC (HL) : Increment RAM Counter
53 53 010E CD 14 01 CALL _WAIT
54 54 0111 C3 0A 01 JP AGAIN ; Endless
55 55
56 56
57 57
58 58 ;*************************************************************************
59 59 ;* Warteschleife 0,5s *
60 60 ;*************************************************************************
61 61 0114 16 FF _WAIT: LD D,$FF ;
62 62 0116 1E FF _OUTER: LD E,$FF ;
63 63 0118 1D _INNER: DEC E
64 64 0119 C2 18 01 JP NZ,_INNER
65 65 011C 15 DEC D
66 66 011D C2 16 01 JP NZ,_OUTER
67 67 0120 C9 RET
68 68
69 69
70 70
71 71
72 72
73
74 Symbol table:
75 AGAIN 010A COUNTER 8000 CTC0 0000 CTC1 0001
76 CTC2 0002 CTC3 0003 MAIN 0100 PIO_A 0080
77 PIO_B 0081 PIO_C 0082 PIO_CON 0083 RAMTOP FFFF
78 SIO_A_C 0042 SIO_A_D 0040 SIO_B_C 0043 SIO_B_D 0041
79 _INNER 0118 _OUTER 0116 _WAIT 0114
80 19 symbols.
```

8.6.6.5 Funktionsbeschreibung

Am Beginn der Ausführung wird der PIO initialisiert. Anschließend wird die Adresse des SRAMs in das HL Register geschrieben, der Inhalt der adressierten Zelle wird in den Akku geladen, über die LEDs ausgegeben und anschließend inkrementiert. Nachdem der Wert in der mittels HL Register adressierten Speicherzelle um 1 erhöht wurde, wird eine Warteschleife aufgerufen, welche eine Wartezeit von 0,5 Sekunden bewirkt, bevor die Schleife von vorne beginnt und der aktuelle Zählerstand wieder ausgegeben und inkrementiert wird. Die Warteschleife besteht aus 2 ineinander verschachtelten Zählschleifen. Jede

der Zählschleifen dekrementiert einen Registerinhalt von ursprünglich 255 (FF) solange bis dieser gleich 0 ist und der Sprung zum Beginn der Schleife nicht mehr durchgeführt wird.

8.6.6.6 Analyse

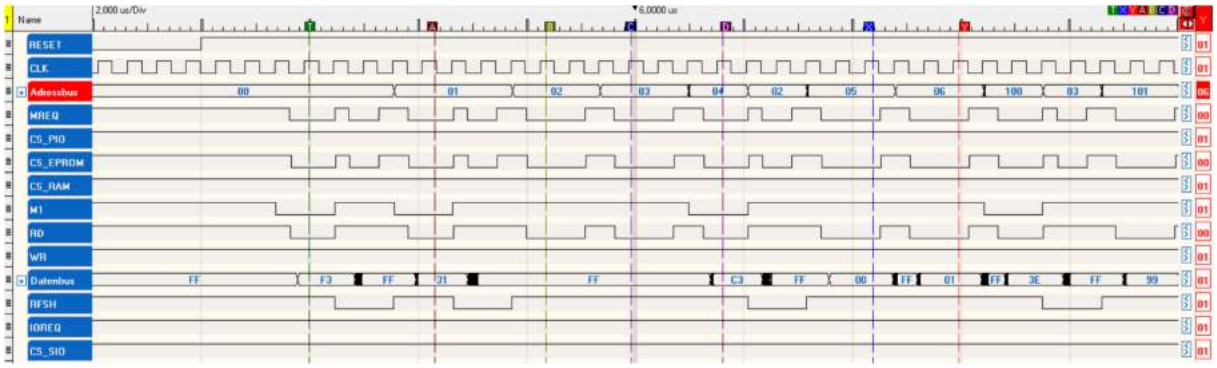


Abbildung 250: DigiView PIO RAM Counter Teil 1

Marker T: Mit dem ersten Takt nach dem Reset liegt am Adressbus die Adresse 0000 an und die CPU führt einen Opcode Fetch durch. Der erhaltene Opcode F3 bewirkt eine Sperre aller Interrupts.

Marker A: Da der Sperrvorgang der Interrupts nur einen Zyklus lang ist, wird bereits im nächsten Zyklus ein erneuter Opcode Fetch durchgeführt. Der Befehl mit dem hexadezimalen Opcode 31 ist ein 16-Bit-Transferbefehl, in diesem Fall für das Setzen des Stackpointers.

Marker B: Mittels Speicher-Lesezugriff wird vom EPROM das Low Byte des Stackpointers geladen.

Marker C: Ein weiterer Speicher-Lesezugriff holt aus dem Speicher das High Byte des Stackpointers.

Marker D: Es wird ein Opcode Fetch durchgeführt. Der Befehl mit dem Opcode 3C ist ein Sprungbefehl mit im Befehl angegebener Sprungadresse.

Marker X: Das Low Byte der Sprungadresse wird mittels Speicher-Lesezugriff aus dem EPROM geholt und im W-Register abgelegt.

Marker Y: In einem weiteren Speicher-Lesezugriff wird das High Byte der Sprungadresse vom EPROM in das Z-Register geladen. Anschließend wird der Inhalt vom WZ-

Registerpaar in den Program Counter geladen.

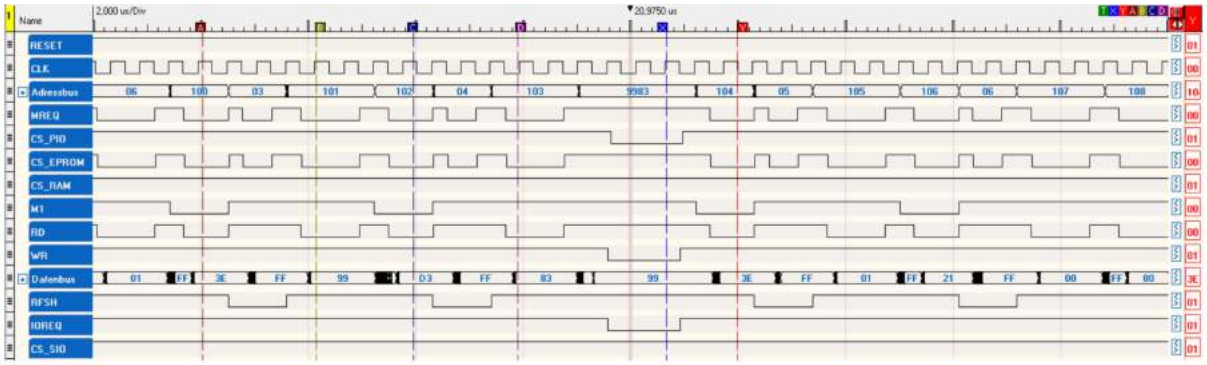


Abbildung 251: DigiView PIO RAM Counter Teil 2

Marker A: Die CPU führt einen Opcode Fetch durch. Der enthaltene Befehl mit dem Opcode 3E ist ein Transferbefehl (Load), welcher eine durch den Programmierer festgelegte Zahl in den Akku der CPU lädt.

Marker B: Aus der Zelle mit der Adresse 0101 wird mittels Speicher-Lesezugriff die Konfiguration des PIO, 99, in den Akku geladen.

Marker C: Mit Beginn des nächsten Zyklus wird ein Opcode Fetch durchgeführt, bei dem aus der Speicherzelle 0102 der Befehl mit dem Opcode D3 geholt wird. Dieser Befehl ist ein Ausgabebefehl für die Peripherie.

Marker D: Als erster Operand wird die Adresse 83 vom EPROM in die CPU geladen.

Marker X: Die Ausgabe der Daten an die Peripherie wird durchgeführt, 99 wird an die Adresse 83 übermittelt.

Marker Y: Der nächste Zyklus beginnt mit einem Instruction Fetch, bei welchem der Opcode 3E geladen wird. Es handelt sich dabei um einen Transferbefehl, welcher im Programmcode festgelegte Daten in den Akkumulator lädt.

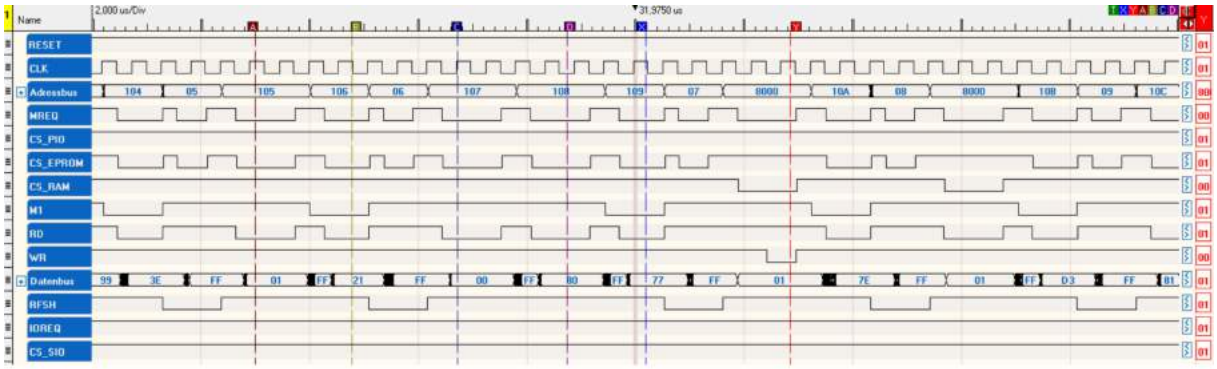


Abbildung 252: DigiView PIO RAM Counter Teil 3

Marker A: Für die Ausführung des Transfers benötigt die CPU die zu übertragenden Daten, welche mittels Speicher-Lesezugriff vom EPROM geholt werden.

Marker B: Beim darauffolgenden Opcode Fetch wird der Befehl 21 geladen. Bei diesem Befehl handelt es sich um einen 16-Bit-Transferbefehl, bei dem durch den Programmierer festgelegte Daten in das HL-Registerpaar geladen werden.

Marker C: Mittels Speicher-Lesezugriff werden die ersten 8 Bit bzw. das Low Byte, 00, über den Datenbus in das L-Register geladen.

Marker D: Das High Byte, 80, und somit der Inhalt des H-Registers wird ebenfalls an sein Ziel übermittelt.

Marker X: Der bei diesem Opcode Fetch übermittelte Befehl 77 ist ein Transferbefehl, bei dem der Inhalt des Akkus in die Speicherzelle geladen wird, welche im HL-Registerpaar adressiert wird.

Marker Y: Am Adressbus liegt 8000, der Inhalt des HL-Registerpaares, an. Die CPU holt mittels Speicher-Lesezugriff 01 vom RAM. Dabei handelt es sich um den ersten Zählerstand des Zählers, der später initialisiert wird.

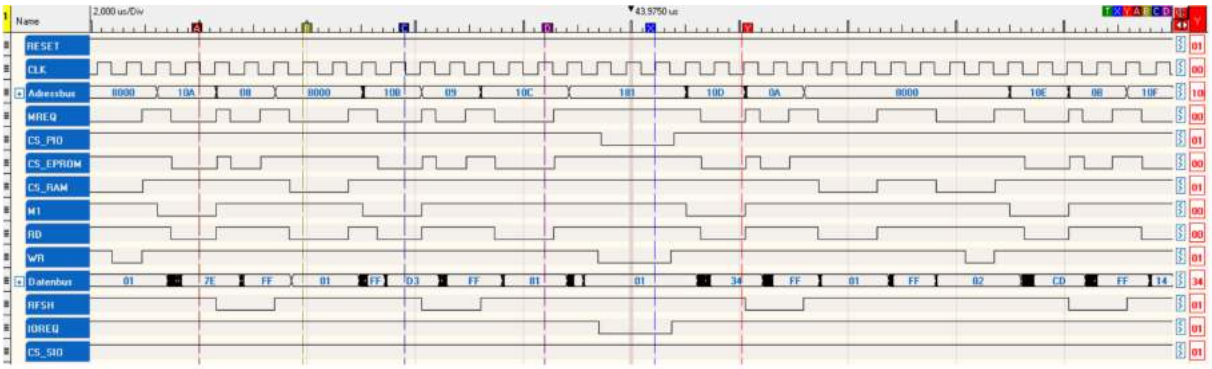


Abbildung 253: DigiView PIO RAM Counter Teil 4

Marker A: Ein Opcode Fetch wird durchgeführt. Der vom EPROM über den Datenbus ausgegebene Befehl besitzt den Opcode 7E. Bei dem Befehl handelt es sich um einen Transferbefehl, bei dem die mit dem HL-Registerpaar adressierte Speicherzelle in den Akku geladen wird.

Marker B: Die im HL-Register enthaltene Adresse, 8000 (welches die erste Adresse im RAM darstellt), wird an den Adressbus angelegt und mittels Speicher-Lesezugriff auf den RAM wird von dort der Inhalt der adressierten Zelle, 01, an den Datenbus angelegt und in den Akku geschrieben.

Marker C: Für die Ausführung des nächsten Befehles wird ein Opcode Fetch durchgeführt. Der geladene Befehl, der Inhalt der Speicherzelle 010B, ist ein Ausgabebefehl mit dem Opcode D3, welcher den Inhalt des Akkus an eine im Programm festgelegte Adresse sendet.

Marker D: Um die Zieladresse 81 zu erhalten, wird mittels Speicher-Lesezugriff der erste Operand aus dem EPROM geholt.

Marker X: Für die Ausgabe des aktuellen Zählerstandes über das LED-Array wird der Port B des PIO mit 81 adressiert und die zuvor eingelesene Schalterstellung am Adressbus ausgebeben.

Marker Y: Es findet ein Opcode Fetch statt, bei dem der Opcode 34 aus dem EPROM ausgelesen wird. Dieser Arithmetikbefehl inkrementiert den mittels HL-Register adressierten Inhalt einer Speicherzelle um 1.

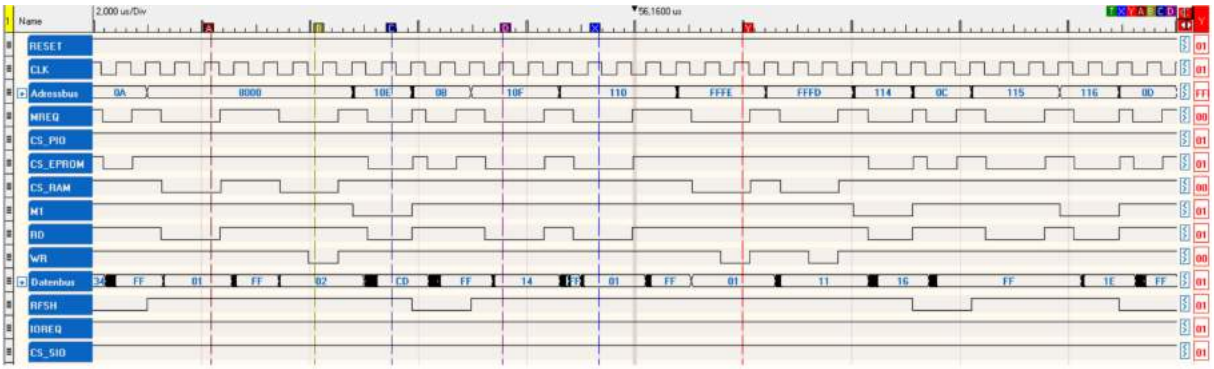


Abbildung 254: DigiView PIO RAM Counter Teil 5

Marker A: Mit einem Speicher-Lesezugriff auf den RAM liest die CPU den Inhalt der Speicherzelle 8000 des SRAMs, 01 aus um ihn zu inkrementieren.

Marker B: Der um 1 erhöhte neue Inhalt der Speicherzelle wird mittels eines Speicher-Schreibzugriffes in die selbe Speicherzelle des RAMs zurückgeschrieben.

Marker C: Beim Opcode Fetch wird der Befehl CD geladen. Bei diesem Befehl handelt es sich um ein Call, den Aufruf eines Unterprogramms mit im Programmcode festgelegter Sprungadresse.

Marker D: Nach einem Refresh-Zyklus wird aus dem Speicher die Sprungadresse für den Aufruf des Unterprogramms aus dem EPROM geladen. Die Adresse ist 16 Bit lang und wird nach dem Big Endian Format in 2 Teile geteilt, deshalb wird in diesem Schritt das Low Byte in das W-Register geladen.

Marker X: Der 2. Teil der Sprungadresse, das High Byte, wird in einem weiteren Speicher-Lesezugriff aus dem EPROM geladen. Die Adresse lautet 0114 und dient für den Aufruf der Warteschleife.

Marker Y: Um nach der Ausführung des Unterprogramms wieder in den Hauptspeicher zurückkehren zu können, muss eine Rücksprungadresse im Stack abgelegt werden. Dieser Stack befindet sich im obersten Adressbereich des RAMs, wo die Daten ausgehend von der höchstwertigen Adresse absteigend abgelegt werden. Die Rücksprungadresse ist 16 Bit groß, deshalb wird in diesem Schritt lediglich das Low Byte auf den Stack geschrieben.

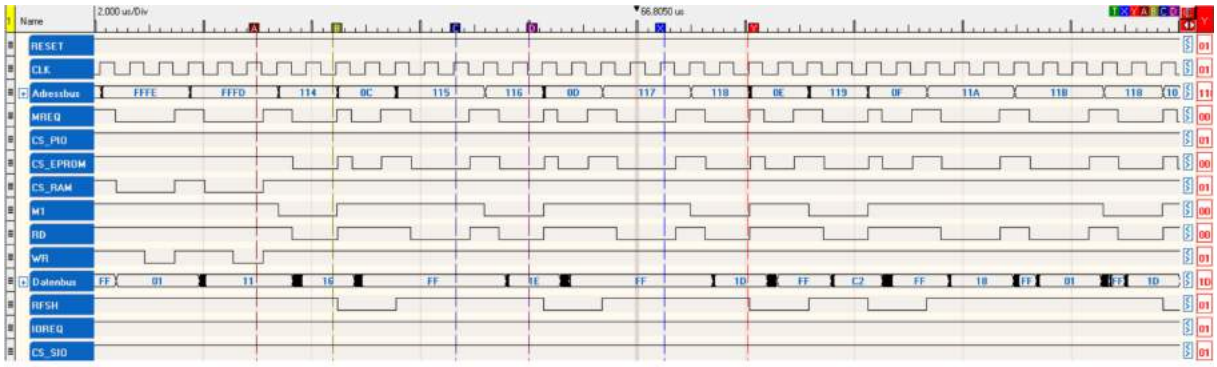


Abbildung 255: DigiView PIO RAM Counter Teil 6

Marker A: In einem weiteren Speicher-Schreibzugriff wird auch das High Byte in den Stack geladen und anschießend das Unterprogramm aufgerufen, indem der Program Counter mit der ersten Adresse des Unterprogramms, 0114, beschrieben wird.

Marker B: Am Beginn des Unterprogramms wird ein Opcode Fetch durchgeführt. Bei diesem Vorgang wird der Opcode 16 in den Decoder geladen. Es handelt sich dabei um einen Transferbefehl, der durch den Programmierer vorgegebenen Daten in das D-Register lädt.

Marker C: Mit einem Speicher-Lesezugriff werden die Daten (FF) aus dem EPROM in das D-Register geschrieben.

Marker D: Im nächsten Zyklus wird ein Opcode Fetch durchgeführt, bei dem der Opcode 1E geladen wird. Dieser Befehl ist ein Transferbefehl und bewirkt, dass ein im Speicher festgelegter Wert in das E-Register geladen wird.

Marker X: In einem Speicher-Lesezugriff wird der Wert FF aus der Speicherzelle mit der Adresse 0115 in das E-Register geladen.

Marker Y: Nach dem abgeschlossenen Datentransfer wird ein Opcode Fetch durchgeführt. Der erhaltene Befehl 1D ist ein Arithmetikbefehl, der den Inhalt des E-Registers um 1 dekrementiert.

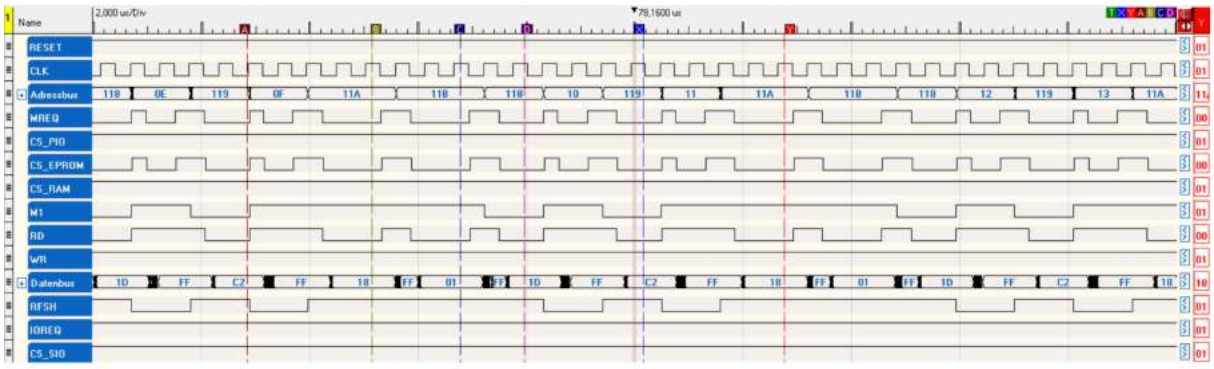


Abbildung 256: DigiView PIO RAM Counter Teil 7

Marker A: Es folgt der nächste Opcode-Fetch. Dabei wird der Befehl mit dem Opcode C2 geladen. Bei diesem Befehl handelt es sich um einen bedingten Sprung, der solange stattfindet, solange das Ergebnis des vorherigen Befehls ungleich Null ist.

Marker B: Um den bedingten Sprung ausführen zu können, muss die Adresse, die in den Program Counter geladen werden soll, aus dem Speicher geholt und im WZ-Registerpaar abgelegt werden. Bei der Adresse handelt es sich um eine 16-Bit-Adresse, daher muss die Abfrage in 2 Schritten erfolgen. Mit diesem Schritt wird das Low Byte geladen.

Marker C: Im 2. Teil des Ladevorganges Wird das High Byte der Sprungadresse geladen. Nun kann der Sprung überprüft und vorgenommen werden, indem bei Erfüllung der Bedingung die Adresse im WZ-Registerpaar in den Befehlszähler geladen wird.

Marker D: Die innere Schleife der Warteschleife beginnt von Neuem, indem mittels Opcode Fetch der Befehl zum Dekrementieren des E-Registers aus dem Speicher geladen und decodiert wird.

Alle weiteren Vorgänge des aufgezeichneten Signals sind Teil der Warteschleife. Die vorkommenden Befehle gleichen den in den letzten 8 Punkten beschriebenen Befehlen, lediglich der Rücksprung vom Unterprogramm zurück in das Hauptprogramm würde einen Neuheitswert darstellen. Dieser Vorgang konnte allerdings nicht aufgezeichnet werden, da es mit dem verwendeten Logikanalysator nicht möglich ist, Vorgänge über mehr als 50 ms aufzuzeichnen und die Warteschleife rund 500 ms lang ist.

8.6.7 Programm CTC_BLINKY_Interrupt

8.6.7.1 Aufgabenstellung

Mithilfe des Z80 Minimalsystems ist das Programm CTC_Blinky_Interrupt auszuführen und das Timing aufzuzeichnen. Dafür ist ein Logikanalysator zu verwenden. Nachdem das Programm vom Reset an aufgezeichnet worden ist, ist das erfasste Timing zu analysieren und eine Zugriffszeitmessung auf den CTC durchzuführen.

8.6.7.2 Konfiguration des DigiView Logikanalysators

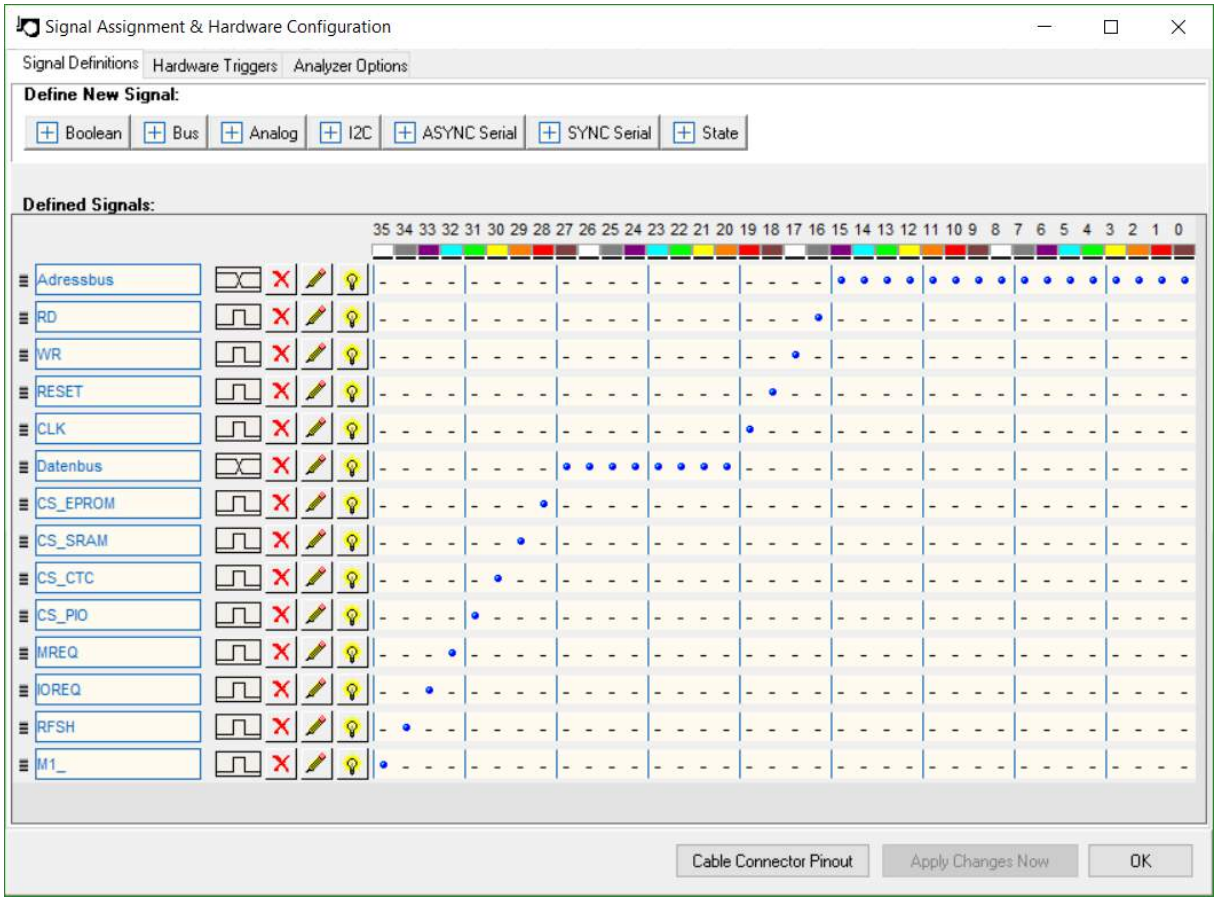


Abbildung 257: DigiView Kanalkonfiguration CTC Blinky Interrupt

8.6.7.3 Source Code

Listing 29: Z80 CTC_BLINKY_Interrupt

```
1  ;*****************************************************************************
2  ;* Z80 Assemblerprogramm *
3  ;* Josef Reisinger *
4  ;* josef.reisinger@htl-hl.ac.at *
5  ;* 26/04/2015 *
6  ;*****************************************************************************
7
8  ; ---------------------------- PIO 82C55 I/O ---------------------------------
9  PIO_A: EQU $80 ; (INPUT)
10 PIO_B: EQU $81 ; (OUTPUT) OUT TO LEDS
11 PIO_C: EQU $82 ; (INPUT) IN from DIP SWITCHES
12 PIO_CON: EQU $83 ; CONTROL BYTE PIO 82C55
13
14 ; ---------------------------- CTC Z80 Timer Counter -------------------------
15 CTC0 EQU $00 ; Channel 0
16 CTC1 EQU $01 ; Channel 1
17 CTC2 EQU $02 ; Channel 2
18 CTC3 EQU $03 ; Channel 3
19
20 ; --------------------------- SIO (USART) ----------------------------------------
21 SIO_A_D: EQU $40 ; Channel A Data Register
22 SIO_B_D: EQU $41 ; Channel B Data Register
23 SIO_A_C: EQU $42 ; Channel A Control Register
24 SIO_B_C: EQU $43 ; Channel B Control Register
25
26 ;--------------------------- CONSTANTS ---------------------------------------
27 RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
28 COUNTER: EQU $8000 ; RAM Counter
29
30 ;*****************************************************************************
31 ;* RESET HANDLER *
32 ;* Function: Initalize system and start Main Programm *
33 ;*****************************************************************************
34             ORG $0000
35             DI ; Disable interrupt
36             LD SP,RAMTOP ; Set stack pointer
37             JP MAIN ; jump to Main program
38
39 ;*****************************************************************
40 ;* MAIN PROGRAM *
41 ;*****************************************************************
42             ORG $0100
43 MAIN: LD HL,COUNTER ; Reset RAM Counter
44         LD A,$00
45         LD (HL),A
46
47         ; ------------------ Initialize PIO ----------------
48         LD A,$99  ; Initialize 8255: PA0-PA7=IN (DIP SWITCHES), PB0-PB7=OUT (LEDS),
49             OUT (PIO_CON),A ; PC0-PC7=IN, Mode 0 Selektion
50
51         ;------------------ Configure CTC ----------------------
52         LD A,$A5  ; Configure CTC Channel 0:Interrupt, Timer Mode, Prescaler = 256,
```

```
53          OUT (CTC0),A ; trigger on positive edge, next word = time constant, Channel
                continuous result operation
54          LD A,$FF  ; Write Time constant 255*256*552ns= 36,03ms
55          OUT (CTC0),A
56          LD A,$A8  ; Loading Interrupt Vector register
57          OUT (CTC0),A  ; trigger on positive edge, next word = time constant, Channel
                continuous result operation
58
59          ;---------------------- Configure Interrupt -----------------
60          LD A,$01 ; Loading Interrupt Register
61          LD I,A
62          IM 2 ; Interrupt Mode 2
63          EI ; Enable Interrupt
64
65          ;--------------------- Main Program --------------------------
66  AGAIN: JP AGAIN ; Endlos
67
68
69  ;*****************************************************************************
70  ;* INTERRUPT SERVICE ROUTINE *
71  ;* CTC Channel 0 *
72  ;*****************************************************************************
73              ORG $01A8
74          DEFW _INT_CTC
75  _INT_CTC: PUSH AF
76              PUSH HL
77          LD HL,COUNTER
78              LD A,(HL) ; Read Counter
79          INC A ; Increment Counter
80          CP $07 ; 252,21ms reached?
81          JP NZ,_END_INT
82          LD A,$00 ; Reset Counter
83          IN A,(PIO_B) ; Toggle LED's with 2Hz
84          CPL
85          OUT (PIO_B),A ;
86  _END_INT: LD (HL),A ; Store Counter
87              POP HL
88              POP AF
89              EI ; Entry Point of Interrupt Service Routine
90          RETI
```

8.6.7.4 HEX-Code

Listing 30: Z80 CTC_BLINKY_Interrupt HEX-Code

```
1  1 ;*****************************************************************************
2  2 ;* Z80 Assembler program *
3  3 ;* Josef Reisinger *
4  4 ;* josef.reisinger@htl-hl.ac.at *
5  5 ;* 26/04/2015 *
6  6 ;*****************************************************************************
7  7
```

```
8  8 ; ---------------------------- PIO 82C55 I/O ---------------------------------
9  9 0080 PIO_A: EQU $80 ; (INPUT)
10 10 0081 PIO_B: EQU $81 ; (OUTPUT) OUT TO LEDS
11 11 0082 PIO_C: EQU $82 ; (INPUT) IN from DIP SWITCHES
12 12 0083 PIO_CON: EQU $83 ; CONTROL BYTE PIO 82C55
13 13
14 14 ; ---------------------------- CTC Z80 Timer Counter -------------------------
15 15 0000 CTC0 EQU $00 ; Channel 0
16 16 0001 CTC1 EQU $01 ; Channel 1
17 17 0002 CTC2 EQU $02 ; Channel 2
18 18 0003 CTC3 EQU $03 ; Channel 3
19 19
20 20 ; ---------------------------- SIO (USART) -----------------------------------------
21 21 0040 SIO_A_D: EQU $40 ; Channel A Data Register
22 22 0041 SIO_B_D: EQU $41 ; Channel B Data Register
23 23 0042 SIO_A_C: EQU $42 ; Channel A Control Register
24 24 0043 SIO_B_C: EQU $43 ; Channel B Control Register
25 25
26 26
27 27 ;---------------------------- CONSTANTS ---------------------------------------
28 28 FFFF RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
29 29 8000 COUNTER: EQU $8000 ; RAM Counter
30 30
31 31
32 32 ;*******************************************************************************
33 33 ;* RESET HANDLER *
34 34 ;* Function: Initialize system and start Main Program *
35 35 ;*******************************************************************************
36 36 0000 ORG $0000
37 37 0000 F3 DI ; Disable interrupt
38 38 0001 31 FF FF LD SP,RAMTOP ; Set stack pointer
39 39 0004 C3 00 01 JP MAIN ; jump to Main program
40 40
41 41
42 42 ;*****************************************************************
43 43 ;* MAIN PROGRAM *
44 44 ;*****************************************************************
45 45 0100 ORG $0100
46 46 0100 21 00 80 MAIN: LD HL,COUNTER ; Reset RAM Counter
47 47 0103 3E 00 LD A,$00
48 48 0105 77 LD (HL),A
49 49
50 50   ; ------------------ Initialize PIO ----------------
51 51 0106 3E 99 LD A,$99 ; Initialize 8255: PA0-PA7=IN (DIP SWITCHES), PB0-PB7=OUT (LEDS),
52 52 0108 D3 83 OUT (PIO_CON),A ; PC0-PC7=IN, Mode 0 Selection
53 53
54 54   ;------------------ Configure CTC ----------------------
55 55 010A 3E A5 LD A,$A5 ; Configure CTC Channel 0: Interrupt, Timer Mode, Prescaler =
       256,
56 56 010C D3 00 OUT (CTC0),A ; trigger on positive edge, next word = time constant,
       Channel continuous result operation
57 57 010E 3E FF LD A,$FF ; Write Time constant 255*256*552ns= 36,03ms
58 58 0110 D3 00 OUT (CTC0),A
```

```
59  59 0112 3E A8 LD A,$A8 ; Loading Interrupt Vector register
60  60 0114 D3 00 OUT (CTC0),A ; trigger on positive edge, next word = time constant,
        Channel continuous result operation
61  61
62  62  ;---------------------- Configure Interrupt -----------------
63  63 0116 3E 01 LD A,$01 ; Loading Interrupt Register
64  64 0118 ED 47 LD I,A
65  65 011A ED 5E IM 2 ; Interrupt Mode 2
66  66 011C FB EI ; Enable Interrupt
67  67
68  68   ;---------------------- Main Program -------------------------
69  69 011D C3 1D 01 AGAIN: JP AGAIN ; Endless
70  70
71  71
72  72
73  73 ;*****************************************************************************
74  74 ;* INTERRUPT SERVICE ROUTINE *
75  75 ;* CTC Channel 0 *
76  76 ;*****************************************************************************
77  77 01A8 ORG $01A8
78  78 01A8 AA 01 DEFW _INT_CTC
79  79 01AA F5 _INT_CTC: PUSH AF
80  80 01AB E5 PUSH HL
81  81 01AC 21 00 80 LD HL,COUNTER
82  82 01AF 7E LD A,(HL) ; Read Counter
83  83 01B0 3C INC A ; Increment Counter
84  84 01B1 FE 07 CP $07 ; 252,21ms reached?
85  85 01B3 C2 BD 01 JP NZ,_END_INT
86  86 01B6 3E 00 LD A,$00 ; Reset Counter
87  87 01B8 DB 81 IN A,(PIO_B) ; Toggle LED's with 2Hz
88  88 01BA 2F CPL
89  89 01BB D3 81 OUT (PIO_B),A ;
90  90 01BD 77 _END_INT: LD (HL),A ; Store Counter
91  91 01BE E1 POP HL
92  92 01BF F1 POP AF
93  93 01C0 FB EI ; Entry Point of Interrupt Service Routine
94  94 01C1 ED 4D RETI
95  95
96  96
97  97
98  98
99  99
100 100
101
102 Symbol table:
103
104 AGAIN 011D COUNTER 8000 CTC0 0000 CTC1 0001
105 CTC2 0002 CTC3 0003 MAIN 0100 PIO_A 0080
106 PIO_B 0081 PIO_C 0082 PIO_CON 0083 RAMTOP FFFF
107 SIO_A_C 0042 SIO_A_D 0040 SIO_B_C 0043 SIO_B_D 0041
108 _END_INT 01BD _INT_CTC 01AA
109 18 symbols.
```

8.6.7.5 Funktionsbeschreibung

Das Programm besteht aus einem Hauptprogramm und einer Interrupt Service Routine. Im Hauptprogramm werden alle Interrupts disabled und der PIO und der CTC konfiguriert, anschließend wird ein Interrupt im Modus 2 konfiguriert und Interrupts wieder aktiviert. Ist die Konfiguration abgeschlossen, läuft eine Endlosschleife. Die Interrupt Service Routine verwendet den 1. Kanal des CTC als Zähler, welcher viermal pro Sekunde die LEDs toggelt.

8.6.7.6 Analyse

☞ Anmerkung: Die Analyse dieses Programms beschränkt sich auf die Konfiguration des CTC.

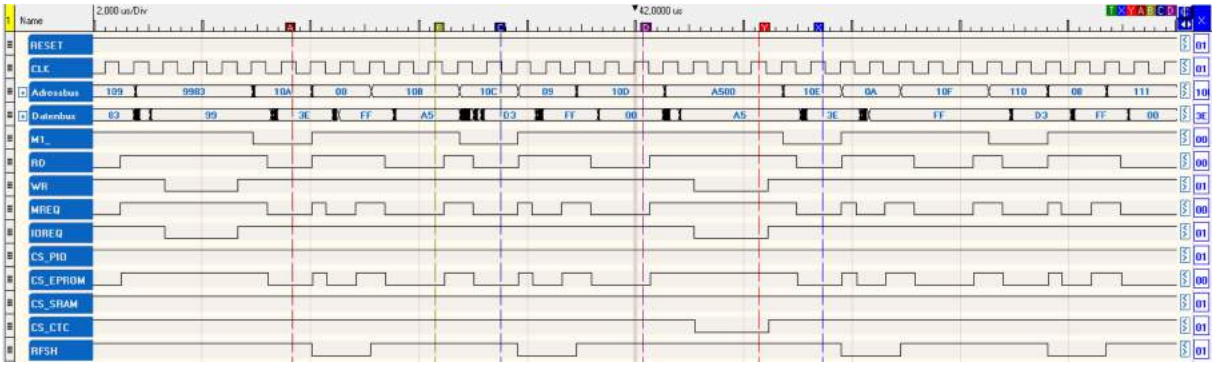


Abbildung 258: DigiView CTC Blinky Interrupt Teil 1

Marker A: Die CPU führt einen Opcode Fetch durch. Der geladene Opcode 3E steht für einen 8-Bit-Transferbefehl.

Marker B: Mittels Speicher-Lesezugriff wird der erste Operand aus dem Speicher geholt.

Marker C: Der nächste Zyklus ist ein Opcode Fetch, wo der Befehl D3, ein Ausgabebefehl für die Peripherie, geladen wird.

Marker D: Der erste Operand, die Zieladresse 00 für die Ausgabe, wird aus dem Speicher eingelesen.

Marker Y: Die Ausgabe der Daten an die Adresse 00 erfolgt. Dort wird die Konfiguration A5 in das Steuerregister des Kanals 0 geschrieben.

Marker X: Bei diesem Opcode Fetch wird ein Transferbefehl geladen, welcher für das Laden von Daten aus dem Speicher in den Akku zuständig ist.

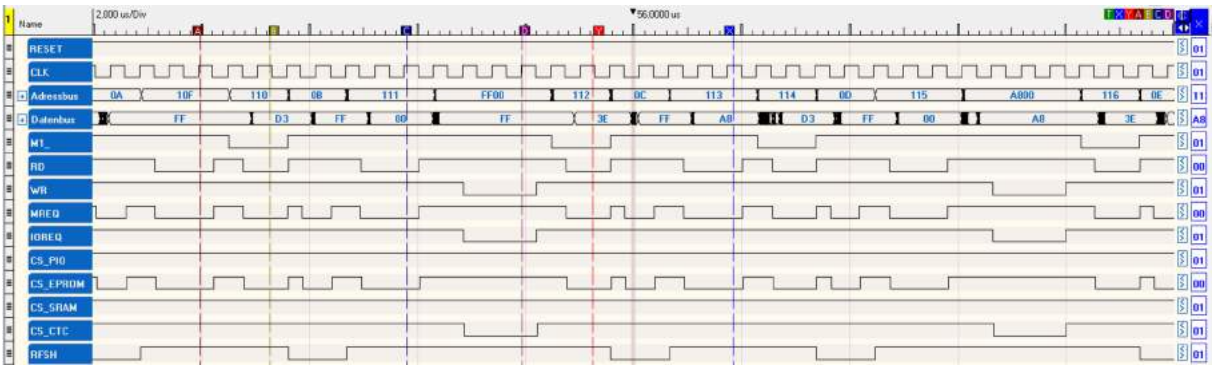


Abbildung 259: DigiView CTC Blinky Interrupt Teil 2

Marker A: Mittels Speicher-Transferbefehl wird der Wert FF aus dem EPROM geholt. Dabei handelt es sich um die Zeitkonstante für den Timer, es müssen also 256 (Prescaler) x 256 (Zeitkonstante) = 65536 Takte vergehen, bis ein Interrupt ausgelöst wird.

Marker B: Es wird ein Opcode Fetch durchgeführt, bei welchem der Opcode D3 geladen wird. Dieser Opcode bewirkt eine Ausgabe von Daten an die Peripherie.

Marker C: Der erste Operand, die Zieladresse 00, wird aus dem EPROM geladen.

Marker D: Der Ausgabebefehl wird ausgeführt und somit das Zeitkonstantenregister des Kanal 0 des CTC konfiguriert.

Marker X: Nach Abschluss der Konfiguration wird ein Opcode Fetch durchgeführt, wodurch 3E, ein Speicher-Transferbefehl geladen wird.

Marker Y: Die Konfiguration A8 des Interrupt Vector Register wird mittels Speicher-Lesezugriff aus dem EPROM geholt.

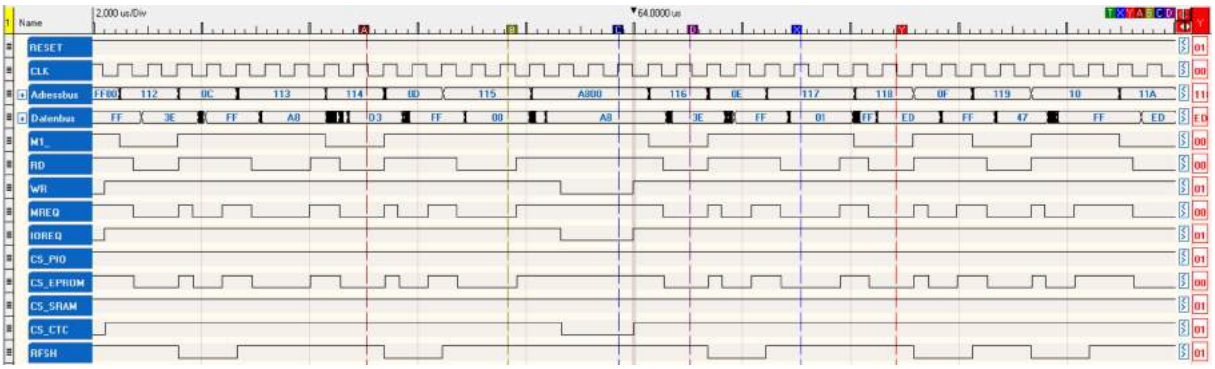


Abbildung 260: DigiView CTC Blinky Interrupt Teil 3

Marker A: Mit diesem Zyklus wird ein Opcode Fetch) durchgeführt und D3 geladen. D3 ist der Opcode eines Ausgabebefehls für die Peripherie.

Marker B: Um die Daten aus dem Akku adressieren zu können, muss aus dem EPROM der Operand, die Zieladresse 00, geholt werden.

Marker C: Die Ausgabe der Daten und somit die Konfiguration des Interrupt Vector Registers erfolgt.

Somit ist die Konfiguration abgeschlossen.

8.6.8 Programm CTC_Counter

8.6.8.1 Source Code

Listing 31: Z80 CTC_Counter

```
1  ;*****************************************************************************
2  ;* Z80 Assembler program *
3  ;* Josef Reisinger *
4  ;* josef.reisinger@htl-hl.ac.at *
5  ;* 26/04/2015 *
6  ;*****************************************************************************
7
8  ; ----------------------------- PIO 82C55 I/O ---------------------------------
9  PIO_A: EQU $80 ; (INPUT)
10 PIO_B: EQU $81 ; (OUTPUT) OUT TO LEDS
11 PIO_C: EQU $82 ; (INPUT) IN from DIP SWITCHES
12 PIO_CON: EQU $83 ; CONTROL BYTE PIO 82C55
13
14 ; -------------------------- CTC Z80 Timer Counter ------------------------
15 CTC0 EQU $00 ; Channel 0
```

```
16 CTC1 EQU $01 ; Channel 1
17 CTC2 EQU $02 ; Channel 2
18 CTC3 EQU $03 ; Channel 3
19
20 ; --------------------------- SIO (USART) ----------------------------------------
21 SIO_A_D: EQU $40 ; Channel A Data Register
22 SIO_B_D: EQU $41 ; Channel B Data Register
23 SIO_A_C: EQU $42 ; Channel A Control Register
24 SIO_B_C: EQU $43 ; Channel B Control Register
25
26
27 ;--------------------------- CONSTANTS ----------------------------------------
28 RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
29
30 ;*****************************************************************************
31 ;* START AFTER RESET, *
32 ;* Function....: ready system and restart *
33 ;*****************************************************************************
34     ORG $0000
35     DI  ; Disable interrupt
36     LD SP,RAMTOP  ; Set stack pointer to top off ram
37     LD A,$15  ; Configure CTC Channel 0:No Interrupt, Timer Mode, No Prescaler,
38     OUT (CTC0),A  ; trigger on positive edge, next word = time constant, Channel
           continuous result operation
39     LD A,186  ; Write Time constant 186*560ns= 104µs
40     OUT (CTC0),A
41 AGAIN: IN A,(CTC0)  ; Read actual status of CTC Channel 0
42     JP AGAIN   ; Endlos
```

8.6.8.2 HEX-Code

Listing 32: Z80 CTC_Counter HEX-Code

```
1  1 ;*****************************************************************************
2  2 ;* Z80 Assembler program *
3  3 ;* Josef Reisinger *
4  4 ;* josef.reisinger@htl-hl.ac.at *
5  5 ;* 26/04/2015 *
6  6 ;*****************************************************************************
7  7
8  8 ; --------------------------- PIO 82C55 I/O --------------------------------
9  9 0080 PIO_A: EQU $80 ; (INPUT)
10 10 0081 PIO_B: EQU $81 ; (OUTPUT) OUT TO LEDS
11 11 0082 PIO_C: EQU $82 ; (INPUT) IN from DIP SWITCHES
12 12 0083 PIO_CON: EQU $83 ; CONTROL BYTE PIO 82C55
13 13
14 14 ; --------------------------- CTC Z80 Timer Counter ------------------------
15 15 0000 CTC0 EQU $00 ; Channel 0
16 16 0001 CTC1 EQU $01 ; Channel 1
17 17 0002 CTC2 EQU $02 ; Channel 2
18 18 0003 CTC3 EQU $03 ; Channel 3
19 19
```

```
20 20 ; ------------------------- SIO (USART) ---------------------------------------
21 21 0040 SIO_A_D: EQU $40 ; Channel A Data Register
22 22 0041 SIO_B_D: EQU $41 ; Channel B Data Register
23 23 0042 SIO_A_C: EQU $42 ; Channel A Control Register
24 24 0043 SIO_B_C: EQU $43 ; Channel B Control Register
25 25
26 26
27 27 ;------------------------- CONSTANTS ---------------------------------------
28 28 FFFF RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
29 29
30 30 ;*******************************************************************************
31 31 ;* START AFTER RESET, *
32 32 ;* Function....: ready system and restart *
33 33 ;*******************************************************************************
34 34 0000 ORG $0000
35 35 0000 F3 DI  ; Disable interrupt
36 36 0001 31 FF FF LD SP,RAMTOP  ; Set stack pointer to top off ram
37 37 0004 3E 15 LD A,$15  ; Configure CTC Channel 0:No Interrupt, Timer Mode, No Prescaler,
38 38 0006 D3 00 OUT (CTC0),A  ; trigger on positive edge, next word = time constant,
       Channel continuous result operation
39 39 0008 3E BA LD A,186  ; Write Time constant 186*560ns= 104µs
40 40 000A D3 00 OUT (CTC0),A
41 41 000C DB 00 AGAIN: IN A,(CTC0)  ; Read actual status of CTC Channel 0
42 42 000E C3 0C 00 JP AGAIN  ; Endlos
43 43
44
45 Symbol table:
46
47 AGAIN 000C CTC0 0000 CTC1 0001 CTC2 0002
48 CTC3 0003 PIO_A 0080 PIO_B 0081 PIO_C 0082
49 PIO_CON 0083 RAMTOP FFFF SIO_A_C 0042 SIO_A_D 0040
50 SIO_B_C 0043 SIO_B_D 0041
51 14 symbols.
```

8.6.9 Programm SIO_V24_Echo_Interrupt

8.6.9.1 Source Code

Listing 33: Z80 SIO_V24_Echo_Interrupt

```
1  ;*******************************************************************************
2  ;* Z80 Assemblerprogramm *
3  ;* Josef Reisinger *
4  ;* josef.reisinger@htl-hl.ac.at *
5  ;* 10/07/2017 *
6  ;*******************************************************************************
7
8
9  ; ---------------------------- PIO 82C55 I/O --------------------------------
10 PIO_A: EQU $80 ; (INPUT)
11 PIO_B: EQU $81 ; (OUTPUT) OUT TO LEDS
```

```
12  PIO_C: EQU $82 ; (INPUT) IN from DIP SWITCHES
13  PIO_CON: EQU $83 ; CONTROL BYTE PIO 82C55
14
15
16  ; --------------------------- CTC Z80 Timer Counter -------------------------
17  CTC0 EQU $00 ; Channel 0
18  CTC1 EQU $01 ; Channel 1
19  CTC2 EQU $02 ; Channel 2
20  CTC3 EQU $03 ; Channel 3
21
22  ; --------------------------- SIO (USART) -----------------------------------------
23  SIO_A_D: EQU $40 ; Channel A Data Register
24  SIO_B_D: EQU $41 ; Channel B Data Register
25  SIO_A_C: EQU $42 ; Channel A Control Register
26  SIO_B_C: EQU $43 ; Channel B Control Register
27
28
29  ;--------------------------- CONSTANTS ---------------------------------------
30  RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
31  CR: EQU $0D
32  LF: EQU $0A
33  SPACE: EQU $20
34
35
36  ;*****************************************************************************
37  ;* RESET HANDLER *
38  ;* Function: Start Main Programm *
39  ;*****************************************************************************
40              ORG $0000
41              JP MAIN ; jump to Main program
42
43
44  ;*****************************************************************************
45  ;* SIO INTERRUPT HANDLER *
46  ;* Function: Start Main Programm *
47  ;*****************************************************************************
48              ORG $000C
49          DEFW RX_CHA_AVAILABLE
50
51              ORG $000E
52          DEFW SPEC_RX_CONDITION
53
54
55  ;*****************************************************************************
56  ;* NMI HANDLER *
57  ;* Handles NMI Interupt Request *
58  ;*****************************************************************************
59              ORG $0066
60          LD HL,NMI_TEXT ; Send NMI to V24
61          CALL SIO_PUT_STRING
62          RETN
63
64
```

```
65  ;*****************************************************************************
66  ;* MAIN PROGRAM *
67  ;*****************************************************************************
68          ORG $100
69  MAIN: DI ; Disable interrupt
70              LD SP,RAMTOP ; Set stack pointer
71          CALL PIO_INIT ; Init PIO (8255)
72          CALL CTC_INIT ; Initalize CTC Channl1 for 9600 Baud (SIO Channel A)
73          CALL SIO_INIT ; Inialize SIO for charecter based transfer (9600,n,8,1)
74
75          LD HL,START_TEXT ; Send Welcome Text to V24
76          CALL SIO_PUT_STRING
77
78          LD A,0
79          LD I,A ; Load I Register with zero
80          IM 2 ; Set Interupt 2
81          EI ; Enable Interrupt
82
83          LD A,$01 ; Initalize moving light
84  _AGAIN: OUT (PIO_B),A ; output Moving light to LED
85          RL A ; next bit
86          CALL _WAIT ; wait 0,5 s
87          JP _AGAIN ; endless
88
89
90  ;*****************************************************************************
91  ;* Initalize PIO (8255) *
92  ;*****************************************************************************
93  PIO_INIT: LD A,$99 ; Init PIO 8255 Control Word:
94                                  ; PA0-PA7=IN (DIP SWITCHES)
95                              ; PB0-PB7=OUT (LEDS),
96          OUT (PIO_CON),A ; PC0-PC7=IN, Mode 0 Selection
97          RET
98
99  ;*****************************************************************************
100 ;* Initalize CTC Channel 1 (SIO Channel A Clock) *
101 ;*****************************************************************************
102 CTC_INIT: LD A,$05 ; Init Timer Counter - Channel 1
103         OUT (CTC1),A ; for Baudrate 9600 (No Interrupt, Timer Mode, PSC=16,
104                             ; trigger on positive edge
105         LD A,$0C ; Write Time constant 12*16*552ns= 105,98s
106         OUT (CTC1),A
107         RET
108
109
110 ;*****************************************************************************
111 ;* Initalize SIO Channel A for character based transfer *
112 ;* Interupt on Received characters on Channel A */
113 ;*****************************************************************************
114 SIO_INIT: LD A,$30 ; Write to WR0 Register --> Error Reset
115         OUT (SIO_A_C),A
116         LD A,$18 ; Write to WR0 Register --> Channel Reset
117         OUT (SIO_A_C),A
```

```
118         LD A,$04 ; Select WR4 Register
119         OUT (SIO_A_C),A
120         LD A,$04 ; CLK*1, 1STOP Bit, No Parity
121         OUT (SIO_A_C),A
122
123         CALL A_RTS_ON ; TX on,TX 8 Bit,DTR inactive, RTS active; Break off
124
125         LD A,$1 ; Select WR1 Register Channel B
126         OUT (SIO_B_C),A
127         LD A,$04 ; no Interrupt on Channel B, status affects Vector
128             OUT (SIO_B_C),A ;
129         LD A,$2 ; Select WR2 Register Channel B
130         OUT (SIO_B_C),A
131         LD A,$00 ; Definition Interrupt vector. Bits D3,D2,D1 are changed according to
132             OUT (SIO_B_C),A ; RX condition (see interupt vector table)
133
134         LD A,$1 ; Select WR1 Register
135         OUT (SIO_A_C),A
136         LD A,$18 ; Interupts on all RX Characters, Parity is not a spec RX Condition
137             OUT (SIO_A_C),A ; Buffer overrun ist a special condition, TX no Interrupt
138
139         CALL SIO_A_EN ; Enable RX Channel A
140         RET
141
142
143 ;*****************************************************************************
144 ;* Enable RX Channel A *
145 ;*****************************************************************************
146 SIO_A_EN: LD A,$03 ; Select WR3 Register
147         OUT (SIO_A_C),A
148         LD A,$C1 ; RX enable,8 Data Bits
149         OUT (SIO_A_C),A
150         RET
151
152 ;*****************************************************************************
153 ;* Disable RX Channel A *
154 ;*****************************************************************************
155 SIO_A_DI: LD A,$03 ; Select WR3 Register
156         OUT (SIO_A_C),A
157         LD A,$C0 ; RX diable,8 Data Bits
158         OUT (SIO_A_C),A
159         RET
160
161 ;*****************************************************************************
162 ;* Channel A RTS inactive (RTS = 1) *
163 ;*****************************************************************************
164 A_RTS_OFF: LD A,$05 ; Select WR5 Register
165         OUT (SIO_A_C),A
166         LD A,$68 ; TX on,TX 8 Bit, DTR inactive,RTS inactive; Break off,
167         OUT (SIO_A_C),A
168         RET
169
170 ;*****************************************************************************
```

```
171 ;* Channel A RTS inactive (RTS=0) *
172 ;*********************************************************************************
173 A_RTS_ON: LD A,$05 ; Select WR5 Register
174         OUT (SIO_A_C),A
175         LD A,$6A ; TX on,TX 8 Bit,DTR inactive, RTS active; Break off
176         OUT (SIO_A_C),A
177         RET
178
179 ;*********************************************************************************
180 ;* Send one Character Via SIO Channel A(Polling Mode) *
181 ;* D- Register: Character to send (ASCII Code) *
182 ;*********************************************************************************
183 SIO_PUT_CHAR: PUSH AF
184 _TX_READY: IN A,(SIO_A_C) ; Read RR0 Register
185               BIT 2,A ; TX Buffer empty ?
186            JP Z,_TX_READY ; No --> Wait
187            LD A,D ; load character in A
188            OUT (SIO_A_D),A ; Send character (Transfer Buffer)
189            POP AF
190            RET
191
192
193
194
195
196 ;*********************************************************************************
197 ;* SEND STRING to V24 via SIO *
198 ;* HL: contains start address of string *
199 ;*********************************************************************************
200 SIO_PUT_STRING: PUSH AF
201 _NEXT_CHAR: LD A,(HL) ; get charcter
202                CP $00 ; END of String ?
203                JP Z,_TEXT_END
204             LD D,A
205             CALL SIO_PUT_CHAR ; send character
206             INC HL ; next character
207                JP _NEXT_CHAR
208 _TEXT_END: POP AF
209                RET
210
211
212 ;*********************************************************************************
213 ;* INTERUTPT HANDLE SIO CHANNEL A CHARACTER RECEIVE *
214 ;*********************************************************************************
215 RX_CHA_AVAILABLE: PUSH AF
216                  CALL A_RTS_OFF
217          IN A,(SIO_A_D) ; Read RX Character
218             LD D,A ; laod Character in D
219          CALL SIO_PUT_CHAR ; Echo Char back to Host
220
221 _NEXT_RX_CHAR: LD A,$0 ;Select RR0 Register
222          OUT (SIO_A_C),A
223                 IN A,(SIO_A_C) ; Read RR0 Register
```

```
224                 BIT 0,A ; RX Character Available ?
225              JP Z,_NEXT_TX_CHAR ; No --> OK
226           IN A,(SIO_A_D) ; Read that character
227              LD D,A ; load Character in D
228           CALL SIO_PUT_CHAR ; Echo Char back to Host
229              JP _NEXT_RX_CHAR
230
231 _NEXT_TX_CHAR: LD A,$1 ; Select RR1 Register
232           OUT (SIO_A_C),A
233           IN A,(SIO_A_C) ; Read RR1 Register
234           BIT 0,A ; ALL Characters sent ?
235           JP Z,_NEXT_TX_CHAR
236
237 _EO_CH_AV: EI
238           CALL A_RTS_ON
239           POP AF
240           RETI
241
242 ;*****************************************************************************
243 ;* INTERUTPT HANDLE SIO CHANNEL A ERROR *
244 ;*****************************************************************************
245 SPEC_RX_CONDITION
246                 JP MAIN ; Restart -> jump to Main program (RESTART)
247
248
249 ;*********************************************************************
250 ;* Warteschleife 0,5s *
251 ;*********************************************************************
252 _WAIT: LD B,$FF ;
253 _OUTER: LD C,$FF ;
254 _INNER: DEC C
255              JP NZ,_INNER
256              DEC B
257              JP NZ,_OUTER
258              RET
259
260 ;*****************************************************************************
261 ;* TEXT DEFINITIONS *
262 ;*****************************************************************************
263 START_TEXT: DEFB CR,LF,'Z','8','0',SPACE,'D','E','M','O',SPACE,'V','1','.','0',$00
264 NMI_TEXT: DEFB CR,LF,'N','M','I',$00
```

8.6.9.2 HEX-Code

Listing 34: Z80 SIO_V24_Echo_Interrupt HEX-Code

```
1 1 ;*****************************************************************************
2 2 ;* Z80 Assembler program *
3 3 ;* Josef Reisinger *
4 4 ;* josef.reisinger@htl-hl.ac.at *
5 5 ;* 10/07/2017 *
6 6 ;*****************************************************************************
```

```
 7  7
 8  8
 9  9 ; --------------------------- PIO 82C55 I/O --------------------------------
10  10 0080 PIO_A: EQU $80 ; (INPUT)
11  11 0081 PIO_B: EQU $81 ; (OUTPUT) OUT TO LEDS
12  12 0082 PIO_C: EQU $82 ; (INPUT) IN from DIP SWITCHES
13  13 0083 PIO_CON: EQU $83 ; CONTROL BYTE PIO 82C55
14  14
15  15
16  16 ; --------------------------- CTC Z80 Timer Counter -------------------------
17  17 0000 CTC0 EQU $00 ; Channel 0
18  18 0001 CTC1 EQU $01 ; Channel 1
19  19 0002 CTC2 EQU $02 ; Channel 2
20  20 0003 CTC3 EQU $03 ; Channel 3
21  21
22  22 ; ------------------------- SIO (USART) ------------------------------------
23  23 0040 SIO_A_D: EQU $40 ; Channel A Data Register
24  24 0041 SIO_B_D: EQU $41 ; Channel B Data Register
25  25 0042 SIO_A_C: EQU $42 ; Channel A Control Register
26  26 0043 SIO_B_C: EQU $43 ; Channel B Control Register
27  27
28  28
29  29 ;-------------------------- CONSTANTS ---------------------------------------
30  30 FFFF RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
31  31 000D CR: EQU $0D
32  32 000A LF: EQU $0A
33  33 0020 SPACE: EQU $20
34  34
35  35 ;*******************************************************************************
36  36 ;* RESET HANDLER *
37  37 ;* Function: Start Main Program *
38  38 ;*******************************************************************************
39  39 0000 ORG $0000
40  40 0000 C3 00 01 JP MAIN ; jump to Main program
41  41
42  42
43  43 ;*******************************************************************************
44  44 ;* SIO INTERRUPT HANDLER *
45  45 ;* Function: Start Main Program *
46  46 ;*******************************************************************************
47  47 000C ORG $000C
48  48 000C A5 01 DEFW RX_CHA_AVAILABLE
49  49
50  50 000E ORG $000E
51  51 000E D5 01 DEFW SPEC_RX_CONDITION
52  52
53  53
54  54 ;*******************************************************************************
55  55 ;* NMI HANDLER *
56  56 ;* Handles NMI Interrupt Request *
57  57 ;*******************************************************************************
58  58 0066 ORG $0066
59  59 0066 21 F5 01 LD HL,NMI_TEXT ; Send NMI to V24
```

```
60  60 0069 CD 94 01 CALL SIO_PUT_STRING
61  61 006C ED 45 RETN
62  62
63  63 ;*******************************************************************************
64  64 ;* MAIN PROGRAM *
65  65 ;*******************************************************************************
66  66 0100 ORG $100
67  67 0100 F3 MAIN: DI ; Disable interrupt
68  68 0101 31 FF FF LD SP,RAMTOP ; Set stack pointer
69  69 0104 CD 26 01 CALL PIO_INIT ; Init PIO (8255)
70  70 0107 CD 2B 01 CALL CTC_INIT ; Initialize CTC Channel 1 for 9600 Baud (SIO Channel A
        )
71  71 010A CD 34 01 CALL SIO_INIT ; Initialize SIO for character based transfer (
        9600,n,8,1)
72  72
73  73 010D 21 E5 01 LD HL,START_TEXT ; Send Welcome Text to V24
74  74 0110 CD 94 01 CALL SIO_PUT_STRING
75  75
76  76 0113 3E 00 LD A,0
77  77 0115 ED 47 LD I,A ; Load I Register with zero
78  78 0117 ED 5E IM 2 ; Set Interrupt 2
79  79 0119 FB EI ; Enable Interrupt
80  80
81  81 011A 3E 01 LD A,$01 ; Initialize moving light
82  82 011C D3 81 _AGAIN: OUT (PIO_B),A ; output Moving light to LED
83  83 011E CB 17 RL A ; next bit
84  84 0120 CD D8 01 CALL _WAIT ; wait 0,5 s
85  85 0123 C3 1C 01 JP _AGAIN ; endless
86  86
87  87
88  88 ;*******************************************************************************
89  89 ;* Initialize PIO (8255) *
90  90 ;*******************************************************************************
91  91 0126 3E 99 PIO_INIT: LD A,$99 ; Init PIO 8255 Control Word:
92  92 0128 ; PA0-PA7=IN (DIP SWITCHES)
93  93 0128 ; PB0-PB7=OUT (LEDS),
94  94 0128 D3 83 OUT (PIO_CON),A ; PC0-PC7=IN, Mode 0 Selection
95  95 012A C9 RET
96  96
97  97 ;*******************************************************************************
98  98 ;* Initialize CTC Channel 1 (SIO Channel A Clock) *
99  99 ;*******************************************************************************
100 100 012B 3E 05 CTC_INIT: LD A,$05 ; Init Timer Counter - Channel 1
101 101 012D D3 01 OUT (CTC1),A ; for Baudrate 9600 (No Interrupt, Timer Mode, PSC=16,
102 102 012F ; trigger on positive edge
103 103 012F 3E 0C LD A,$0C ; Write Time constant 12*16*552ns= 105,98s
104 104 0131 D3 01 OUT (CTC1),A
105 105 0133 C9 RET
106 106
107 107
108 108 ;*******************************************************************************
109 109 ;* Initialize SIO Channel A for character based transfer *
110 110 ;* Interrupt on Received characters on Channel A */
```

```
111  111 ;*******************************************************************************
112  112 0134 3E 30 SIO_INIT: LD A,$30 ; Write to WR0 Register --> Error Reset
113  113 0136 D3 42 OUT (SIO_A_C),A
114  114 0138 3E 18 LD A,$18 ; Write to WR0 Register --> Channel Reset
115  115 013A D3 42 OUT (SIO_A_C),A
116  116 013C 3E 04 LD A,$04 ; Select WR4 Register
117  117 013E D3 42 OUT (SIO_A_C),A
118  118 0140 3E 04 LD A,$04 ; CLK*1, 1STOP Bit, No Parity
119  119 0142 D3 42 OUT (SIO_A_C),A
120  120
121  121 0144 CD 7E 01 CALL A_RTS_ON  ; TX on, TX 8Bit, DTR inactive, RTS active; Break off
122  122
123  123 0147 3E 01 LD A,$1 ; Select WR1 Register Channel B
124  124 0149 D3 43 OUT (SIO_B_C),A
125  125 014B 3E 04 LD A,$04 ; no Interrupt on Channel B, status affects Vector
126  126 014D D3 43 OUT (SIO_B_C),A ;
127  127 014F 3E 02 LD A,$2 ; Select WR2 Register Channel B
128  128 0151 D3 43 OUT (SIO_B_C),A
129  129 0153 3E 00 LD A,$00 ; Definition Interrupt vector. Bits D3,D2,D1 are changed
         according to
130  130 0155 D3 43 OUT (SIO_B_C),A ; RX condition (see interrupt vector table)
131  131
132  132 0157 3E 01 LD A,$1 ; Select WR1 Register
133  133 0159 D3 42 OUT (SIO_A_C),A
134  134 015B 3E 18 LD A,$18 ; Interrupts on all RX Characters, Parity is not a spec RX
         Condition
135  135 015D D3 42 OUT (SIO_A_C),A ; Buffer overrun is a special condition, TX no Interrupt
136  136
137  137 015F CD 63 01 CALL SIO_A_EN ; Enable RX Channel A
138  138 0162 C9 RET
139  139
140  140
141  141 ;*******************************************************************************
142  142 ;* Enable RX Channel A *
143  143 ;*******************************************************************************
144  144 0163 3E 03 SIO_A_EN: LD A,$03 ; Select WR3 Register
145  145 0165 D3 42 OUT (SIO_A_C),A
146  146 0167 3E C1 LD A,$C1 ; RX enable,8 Data Bits
147  147 0169 D3 42 OUT (SIO_A_C),A
148  148 016B C9 RET
149  149
150  150 ;*******************************************************************************
151  151 ;* Disable RX Channel A *
152  152 ;*******************************************************************************
153  153 016C 3E 03 SIO_A_DI: LD A,$03 ; Select WR3 Register
154  154 016E D3 42 OUT (SIO_A_C),A
155  155 0170 3E C0 LD A,$C0 ; RX diable,8 Data Bits
156  156 0172 D3 42 OUT (SIO_A_C),A
157  157 0174 C9 RET
158  158
159  159 ;*******************************************************************************
160  160 ;* Channel A RTS inactive (RTS = 1) *
161  161 ;*******************************************************************************
```

```
162  162 0175 3E 05 A_RTS_OFF: LD A,$05 ; Select WR5 Register
163  163 0177 D3 42 OUT (SIO_A_C),A
164  164 0179 3E 68 LD A,$68 ; TX on,TX 8 Bit, DTR inactive,RTS inactive; Break off,
165  165 017B D3 42 OUT (SIO_A_C),A
166  166 017D C9 RET
167  167
168  168 ;*****************************************************************************
169  169 ;* Channel A RTS inactive (RTS=0) *
170  170 ;*****************************************************************************
171  171 017E 3E 05 A_RTS_ON: LD A,$05 ; Select WR5 Register
172  172 0180 D3 42 OUT (SIO_A_C),A
173  173 0182 3E 6A LD A,$6A ; TX on,TX 8 Bit,DTR inactive, RTS active; Break off
174  174 0184 D3 42 OUT (SIO_A_C),A
175  175 0186 C9 RET
176  176
177  177 ;*****************************************************************************
178  178 ;* Send one Character Via SIO Channel A(Polling Mode) *
179  179 ;* D- Register: Character to send (ASCII Code) *
180  180 ;*****************************************************************************
181  181 0187 F5 SIO_PUT_CHAR: PUSH AF
182  182 0188 DB 42 _TX_READY: IN A,(SIO_A_C) ; Read RR0 Register
183  183 018A CB 57 BIT 2,A ; TX Buffer empty ?
184  184 018C CA 88 01 JP Z,_TX_READY ; No --> Wait
185  185 018F 7A LD A,D ; load character in A
186  186 0190 D3 40 OUT (SIO_A_D),A ; Send character (Transfer Buffer)
187  187 0192 F1 POP AF
188  188 0193 C9 RET
189  189
190  190
191  191 ;*****************************************************************************
192  192 ;* SEND STRING to V24 via SIO *
193  193 ;* HL: contains start address of string *
194  194 ;*****************************************************************************
195  195 0194 F5 SIO_PUT_STRING: PUSH AF
196  196 0195 7E _NEXT_CHAR: LD A,(HL) ; get character
197  197 0196 FE 00 CP $00 ; END of String ?
198  198 0198 CA A3 01 JP Z,_TEXT_END
199  199 019B 57 LD D,A
200  200 019C CD 87 01 CALL SIO_PUT_CHAR ; send character
201  201 019F 23 INC HL ; next character
202  202 01A0 C3 95 01 JP _NEXT_CHAR
203  203 01A3 F1 _TEXT_END: POP AF
204  204 01A4 C9 RET
205  205
206  206
207  207 ;*****************************************************************************
208  208 ;* INTERUTPT HANDLE SIO CHANNEL A CHARACTER RECEIVE *
209  209 ;*****************************************************************************
210  210 01A5 F5 RX_CHA_AVAILABLE: PUSH AF
211  211 01A6 CD 75 01 CALL A_RTS_OFF
212  212 01A9 DB 40 IN A,(SIO_A_D) ; Read RX Character
213  213 01AB 57 LD D,A ; load Character in D
214  214 01AC CD 87 01 CALL SIO_PUT_CHAR ; Echo Char back to Host
```

```
215  215
216  216 01AF 3E 00 _NEXT_RX_CHAR: LD A,$0 ;Select RR0 Register
217  217 01B1 D3 42 OUT (SIO_A_C),A
218  218 01B3 DB 42 IN A,(SIO_A_C) ; Read RR0 Register
219  219 01B5 CB 47 BIT 0,A ; RX Character Available ?
220  220 01B7 CA C3 01 JP Z,_NEXT_TX_CHAR ; No --> OK
221  221 01BA DB 40 IN A,(SIO_A_D) ; Read that character
222  222 01BC 57 LD D,A ; load Character in D
223  223 01BD CD 87 01 CALL SIO_PUT_CHAR ; Echo Char back to Host
224  224 01C0 C3 AF 01 JP _NEXT_RX_CHAR
225  225
226  226 01C3 3E 01 _NEXT_TX_CHAR: LD A,$1 ; Select RR1 Register
227  227 01C5 D3 42 OUT (SIO_A_C),A
228  228 01C7 DB 42 IN A,(SIO_A_C) ; Read RR1 Register
229  229 01C9 CB 47 BIT 0,A ; ALL Characters sent ?
230  230 01CB CA C3 01 JP Z,_NEXT_TX_CHAR
231  231
232  232 01CE FB _EO_CH_AV: EI
233  233 01CF CD 7E 01 CALL A_RTS_ON
234  234 01D2 F1 POP AF
235  235 01D3 ED 4D RETI
236  236
237  237 ;*****************************************************************************
238  238 ;* INTERUTPT HANDLE SIO CHANNEL A ERROR *
239  239 ;*****************************************************************************
240  240 01D5 SPEC_RX_CONDITION
241  241 01D5 C3 00 01 JP MAIN ; Restart -> jump to Main program (RESTART)
242  242
243  243
244  244
245  245 ;*********************************************************************
246  246 ;* Warteschleife 0,5s *
247  247 ;*********************************************************************
248  248 01D8 06 FF _WAIT: LD B,$FF ;
249  249 01DA 0E FF _OUTER: LD C,$FF ;
250  250 01DC 0D _INNER: DEC C
251  251 01DD C2 DC 01 JP NZ,_INNER
252  252 01E0 05 DEC B
253  253 01E1 C2 DA 01 JP NZ,_OUTER
254  254 01E4 C9 RET
255  255
256  256 ;*****************************************************************************
257  257 ;* TEXT DEFINITIONS *
258  258 ;*****************************************************************************
259  259 01E5 0D 0A 5A 38 START_TEXT: DEFB CR,LF,'Z','8','0',SPACE,'D','E','M','O',SPACE,'V','1','
          .','0',$00
260  01E9 30 20 44 45
261  01ED 4D 4F 20 56
262  01F1 31 2E 30 00
263  260 01F5 0D 0A 4E 4D NMI_TEXT: DEFB CR,LF,'N','M','I',$00
264  01F9 49 00
265  261
266  262
```

```
263
264
265
266
267
268
269

Symbol table:

A_RTS_OFF 0175 A_RTS_ON 017E CR 000D
CTC0 0000 CTC1 0001 CTC2 0002
CTC3 0003 CTC_INIT 012B LF 000A
MAIN 0100 NMI_TEXT 01F5 PIO_A 0080
PIO_B 0081 PIO_C 0082 PIO_CON 0083
PIO_INIT 0126 RAMTOP FFFF RX_CHA_AVAILABLE 01A5
SIO_A_C 0042 SIO_A_D 0040 SIO_A_DI 016C
SIO_A_EN 0163 SIO_B_C 0043 SIO_B_D 0041
SIO_INIT 0134 SIO_PUT_CHAR 0187 SIO_PUT_STRING 0194
SPACE 0020 SPEC_RX_CONDITION 01D5 START_TEXT 01E5
_AGAIN 011C _EO_CH_AV 01CE _INNER 01DC
_NEXT_CHAR 0195 _NEXT_RX_CHAR 01AF _NEXT_TX_CHAR 01C3
_OUTER 01DA _TEXT_END 01A3 _TX_READY 0188
_WAIT 01D8
40 symbols.
```

8.6.9.3 Funktionsbeschreibung

Das Programm konfiguriert zuerst SIO, PIO und CTC und versendet dann über den UART, welcher auf eine Baudrate von 9600 Baud konfiguriert wurde, einen String (Z80 DEMO V1.0). Wird von einem Terminal ein Zeichen an das Minimalsystem gesendet, antwortet es mit dem gleichen Zeichen. Dies geschieht mit einem Interrupt, welcher durch den Kanal A des SIO ausgelöst wird. Weiters bewirkt das Auslösen eines Interrupts ebenfalls das Versenden eines Strings (NMI).

8.6.10 Programm SIO_V24_Echo_Polling

8.6.10.1 Source Code

Listing 35: Z80 SIO_V24_Echo_Polling

```
;*****************************************************************************
;* Z80 Assembler program *
;* Josef Reisinger *
;* josef.reisinger@htl-hl.ac.at *
;* 10/07/2017 *
;*****************************************************************************
```

```
 7
 8
 9  ; ---------------------------- PIO 82C55 I/O --------------------------------
10  PIO_A: EQU $80 ; (INPUT)
11  PIO_B: EQU $81 ; (OUTPUT) OUT TO LEDS
12  PIO_C: EQU $82 ; (INPUT) IN from DIP SWITCHES
13  PIO_CON: EQU $83 ; CONTROL BYTE PIO 82C55
14
15
16  ; ---------------------------- CTC Z80 Timer Counter -------------------------
17  CTC0 EQU $00 ; Channel 0
18  CTC1 EQU $01 ; Channel 1
19  CTC2 EQU $02 ; Channel 2
20  CTC3 EQU $03 ; Channel 3
21
22  ; ---------------------------- SIO (USART) ----------------------------------------
23  SIO_A_D: EQU $40 ; Channel A Data Register
24  SIO_B_D: EQU $41 ; Channel B Data Register
25  SIO_A_C: EQU $42 ; Channel A Control Register
26  SIO_B_C: EQU $43 ; Channel B Control Register
27
28
29  ;---------------------------- CONSTANTS ----------------------------------------
30  RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
31  CR: EQU $0D
32  LF: EQU $0A
33  SPACE: EQU $20
34
35  ;*******************************************************************************
36  ;* RESET HANDLER *
37  ;* Function: Start Main Programm *
38  ;*******************************************************************************
39              ORG $0000
40              JP MAIN ; jump to Main program
41
42
43  ;*******************************************************************************
44  ;* NMI HANDLER *
45  ;* Handles NMI Interrupt Request *
46  ;*******************************************************************************
47              ORG $0066
48          LD HL,NMI_TEXT ; Send NMI to V24
49          CALL SIO_PUT_STRING
50          RETN
51
52  ;*******************************************************************************
53  ;* MAIN PROGRAM *
54  ;*******************************************************************************
55          ORG $100
56  MAIN: DI ; Disable interrupt
57          LD SP,RAMTOP ; Set stack pointer
58          CALL PIO_INIT  ; Init PIO (8255)
59          CALL CTC_INIT  ; Initialize CTC Channel 1 for 9600 Baud (SIO Channel A)
```

```
60          CALL SIO_INIT  ; Initialize SIO for character based transfer (9600,n,8,1)
61          LD HL,START_TEXT  ; Send Welcome Text to V24
62          CALL SIO_PUT_STRING
63  _AGAIN: CALL SIO_GET_CHAR ; Wait for Character on V24
64              LD A,E  ; Output Error to LED's
65          OUT (PIO_B),A
66          CALL SIO_PUT_CHAR  ; Echo Character to V24
67          JP _AGAIN
68        ;
69
70  ;*************************************************************************
71  ;* Initialize PIO (8255) *
72  ;*************************************************************************
73  PIO_INIT: LD A,$99 ; Init PIO 8255 Control Word:
74                             ; PA0-PA7=IN (DIP SWITCHES)
75                             ; PB0-PB7=OUT (LEDS),
76          OUT (PIO_CON),A ; PC0-PC7=IN, Mode 0 Selection
77          RET
78
79  ;*************************************************************************
80  ;* Initialize CTC Channel 1 (SIO Channel A Clock) *
81  ;*************************************************************************
82  CTC_INIT: LD A,$05  ; Init Timer Counter - Channel 1
83          OUT (CTC1),A  ; for Baudrate 9600 (No Interrupt, Timer Mode, PSC=16,
84                        ; trigger on positive edge
85          LD A,$0C ; Write Time constant 12*16*552ns= 105,98s
86          OUT (CTC1),A
87          RET
88
89
90  ;*************************************************************************
91  ;* Initialize SIO Channel A for character based transfer *
92  ;*************************************************************************
93  SIO_INIT: LD A,$30 ; Write to WR0 Register --> Error Reset
94          OUT (SIO_A_C),A
95          LD A,$18 ; Write to WR0 Register --> Channel Reset
96          OUT (SIO_A_C),A
97          LD A,$04 ; Select WR4 Register
98          OUT (SIO_A_C),A
99          LD A,$04 ; CLK*1, 1STOP Bit, No Parity
100         OUT (SIO_A_C),A
101         LD A,$03 ; Select WR3 Register
102         OUT (SIO_A_C),A
103         LD A,$C1 ; RX enable,8 Data Bits
104         OUT (SIO_A_C),A
105         LD A,$05 ; Select WR5 Register
106         OUT (SIO_A_C),A
107         LD A,$68 ; TX enable,8 Data Bits, DTR inactive, RTS inactive, Break off
108         OUT (SIO_A_C),A
109         LD A,$1 ; Select WR1 Register
110         OUT (SIO_A_C),A
111         LD A,$0 ; No Interrupts for Rx and Tx Characters
112             OUT (SIO_A_C),A
```

```
113         RET
114
115
116
117 ;*****************************************************************************
118 ;* Send one Character Via SIO Channel A(Polling Mode) *
119 ;* D- Register: Character to send (ASCII Code) *
120 ;*****************************************************************************
121 SIO_PUT_CHAR: PUSH AF
122 _TX_READY: IN A,(SIO_A_C) ; Read RR0 Register
123             BIT 2,A   ; TX Buffer empty ?
124           JP Z,_TX_READY  ; No --> Wait
125           LD A,D   ; load character in A
126           OUT (SIO_A_D),A  ; Send character (Transfer Buffer)
127           POP AF
128           RET
129
130
131 ;*****************************************************************************
132 ;* Receive one Character Via SIO Channel A(Polling Mode) *
133 ;* D- Register: Character received (ASCII Code) *
134 ;* E-Register: Error Code *
135 ;*****************************************************************************
136 SIO_GET_CHAR: PUSH AF
137 _RX_READY: IN A,(SIO_A_C) ; Read RR0 Register
138             BIT 0,A ; RX Character Available ?
139           JP Z,_RX_READY ; No --> Wait
140           IN A,(SIO_A_D) ; Store character
141           LD D,A
142           LD A,$01 ; Select WR1 Register
143           OUT (SIO_A_C),A
144           IN A,(SIO_A_C) ; Read Error Register
145           LD E,A ; store Error Status
146           AND $70 ; only D6(CRC Framing Error),D5(Rx Overrun Error) und D4 (Parity
                Error)
147           JP Z,_RX_EXIT ; return if no error
148           LD A,$30 ; reset Error
149             OUT (SIO_A_C),A
150 _RX_EXIT: POP AF
151           RET
152
153
154 ;*****************************************************************************
155 ;* SEND STRING to V24 via SIO *
156 ;* HL: contains start address of string *
157 ;*****************************************************************************
158 SIO_PUT_STRING: PUSH AF
159 _NEXT_CHAR: LD A,(HL) ; get character
160              CP $00 ; END of String ?
161              JP Z,_TEXT_END
162            LD D,A
163            CALL SIO_PUT_CHAR ; send character
164            INC HL ; next character
```

```
165                 JP _NEXT_CHAR
166 _TEXT_END: POP AF
167                 RET
168
169
170 ;*****************************************************************************
171 ;* TEXT DEFINITIONS *
172 ;*****************************************************************************
173 START_TEXT: DEFB CR,LF,'Z','8','0',SPACE,'D','E','M','O',SPACE,'V','1','.','0',$00
174 NMI_TEXT: DEFB CR,LF,'N','M','I',$00
```

8.6.10.2 HEX-Code

Listing 36: Z80 SIO_V24_Echo_Polling HEX-Code

```
1  1 ;*****************************************************************************
2  2 ;* Z80 Assembler program *
3  3 ;* Josef Reisinger *
4  4 ;* josef.reisinger@htl-hl.ac.at *
5  5 ;* 10/07/2017 *
6  6 ;*****************************************************************************
7  7
8  8
9  9 ; ---------------------------- PIO 82C55 I/O ---------------------------------
10 10 0080 PIO_A: EQU $80 ; (INPUT)
11 11 0081 PIO_B: EQU $81 ; (OUTPUT) OUT TO LEDS
12 12 0082 PIO_C: EQU $82 ; (INPUT) IN from DIP SWITCHES
13 13 0083 PIO_CON: EQU $83 ; CONTROL BYTE PIO 82C55
14 14
15 15
16 16 ; --------------------------- CTC Z80 Timer Counter --------------------------
17 17 0000 CTC0 EQU $00 ; Channel 0
18 18 0001 CTC1 EQU $01 ; Channel 1
19 19 0002 CTC2 EQU $02 ; Channel 2
20 20 0003 CTC3 EQU $03 ; Channel 3
21 21
22 22 ; ------------------------- SIO (USART) ------------------------------------
23 23 0040 SIO_A_D: EQU $40 ; Channel A Data Register
24 24 0041 SIO_B_D: EQU $41 ; Channel B Data Register
25 25 0042 SIO_A_C: EQU $42 ; Channel A Control Register
26 26 0043 SIO_B_C: EQU $43 ; Channel B Control Register
27 27
28 28
29 29 ;-------------------------- CONSTANTS ---------------------------------------
30 30 FFFF RAMTOP: EQU $FFFF ; 32Kb RAM 8000H-FFFFH
31 31 000D CR: EQU $0D
32 32 000A LF: EQU $0A
33 33 0020 SPACE: EQU $20
34 34
35 35 ;*****************************************************************************
36 36 ;* RESET HANDLER *
37 37 ;* Function: Start Main Program *
```

```
38 38 ;*******************************************************************************
39 39 0000 ORG $0000
40 40 0000 C3 00 01 JP MAIN ; jump to Main program
41 41
42 42
43 43 ;*******************************************************************************
44 44 ;* NMI HANDLER *
45 45 ;* Handles NMI Interrupt Request *
46 46 ;*******************************************************************************
47 47 0066 ORG $0066
48 48 0066 21 A1 01 LD HL,NMI_TEXT ; Send NMI to V24
49 49 0069 CD 80 01 CALL SIO_PUT_STRING
50 50 006C ED 45 RETN
51 51
52 52 ;*******************************************************************************
53 53 ;* MAIN PROGRAM *
54 54 ;*******************************************************************************
55 55 0100 ORG $100
56 56 0100 F3 MAIN: DI ; Disable interrupt
57 57 0101 31 FF FF LD SP,RAMTOP ; Set stack pointer
58 58 0104 CD 1F 01 CALL PIO_INIT ; Init PIO (8255)
59 59 0107 CD 24 01 CALL CTC_INIT ; Initialize CTC Channl1 for 9600 Baud (SIO Channel A)
60 60 010A CD 2D 01 CALL SIO_INIT ; Animalize SIO for character based transfer (9600,n,8,1
    )
61 61 010D 21 91 01 LD HL,START_TEXT ; Send Welcome Text to V24
62 62 0110 CD 80 01 CALL SIO_PUT_STRING
63 63 0113 CD 63 01 _AGAIN: CALL SIO_GET_CHAR ; Wait for Character on V24
64 64 0116 7B LD A,E ; Output Error to LED's
65 65 0117 D3 81 OUT (PIO_B),A
66 66 0119 CD 56 01 CALL SIO_PUT_CHAR ; Echo Character to V24
67 67 011C C3 13 01 JP _AGAIN
68 68 011F ;
69 69
70 70 ;*******************************************************************************
71 71 ;* Initialize PIO (8255) *
72 72 ;*******************************************************************************
73 73 011F 3E 99 PIO_INIT: LD A,$99 ; Init PIO 8255 Control Word:
74 74 0121 ; PA0-PA7=IN (DIP SWITCHES)
75 75 0121 ; PB0-PB7=OUT (LEDS),
76 76 0121 D3 83 OUT (PIO_CON),A ; PC0-PC7=IN, Mode 0 Selection
77 77 0123 C9 RET
78 78
79 79 ;*******************************************************************************
80 80 ;* Initialize CTC Channel 1 (SIO Channel A Clock) *
81 81 ;*******************************************************************************
82 82 0124 3E 05 CTC_INIT: LD A,$05 ; Init Timer Counter - Channel 1
83 83 0126 D3 01 OUT (CTC1),A ; for Baudrate 9600 (No Interrupt, Timer Mode, PSC=16,
84 84 0128 ; trigger on positive edge
85 85 0128 3E 0C LD A,$0C ; Write Time constant 12*16*552ns= 105,98s
86 86 012A D3 01 OUT (CTC1),A
87 87 012C C9 RET
88 88
89 89
```

```
 90  90 ;*****************************************************************************
 91  91 ;* Initialize SIO Channel A for character based transfer *
 92  92 ;*****************************************************************************
 93  93 012D 3E 30 SIO_INIT: LD A,$30 ; Write to WR0 Register --> Error Reset
 94  94 012F D3 42 OUT (SIO_A_C),A
 95  95 0131 3E 18 LD A,$18 ; Write to WR0 Register --> Channel Reset
 96  96 0133 D3 42 OUT (SIO_A_C),A
 97  97 0135 3E 04 LD A,$04 ; Select WR4 Register
 98  98 0137 D3 42 OUT (SIO_A_C),A
 99  99 0139 3E 04 LD A,$04 ; CLK*1, 1STOP Bit, No Parity
100 100 013B D3 42 OUT (SIO_A_C),A
101 101 013D 3E 03 LD A,$03 ; Select WR3 Register
102 102 013F D3 42 OUT (SIO_A_C),A
103 103 0141 3E C1 LD A,$C1 ; RX enable,8 Data Bits
104 104 0143 D3 42 OUT (SIO_A_C),A
105 105 0145 3E 05 LD A,$05 ; Select WR5 Register
106 106 0147 D3 42 OUT (SIO_A_C),A
107 107 0149 3E 68 LD A,$68 ; TX enable,8 Data Bits, DTR inactive, RTS inactive,Break off
108 108 014B D3 42 OUT (SIO_A_C),A
109 109 014D 3E 01 LD A,$1 ; Select WR1 Register
110 110 014F D3 42 OUT (SIO_A_C),A
111 111 0151 3E 00 LD A,$0 ; No Interrupts for Rx and Tx Characters
112 112 0153 D3 42 OUT (SIO_A_C),A
113 113 0155 C9 RET
114 114
115 115
116 116
117 117 ;*****************************************************************************
118 118 ;* Send one Character Via SIO Channel A(Polling Mode) *
119 119 ;* D- Register: Character to send (ASCII Code) *
120 120 ;*****************************************************************************
121 121 0156 F5 SIO_PUT_CHAR: PUSH AF
122 122 0157 DB 42 _TX_READY: IN A,(SIO_A_C) ; Read RR0 Register
123 123 0159 CB 57 BIT 2,A ; TX Buffer empty ?
124 124 015B CA 57 01 JP Z,_TX_READY ; No --> Wait
125 125 015E 7A LD A,D ; load character in A
126 126 015F D3 40 OUT (SIO_A_D),A ; Send character (Transfer Buffer)
127 127 0161 F1 POP AF
128 128 0162 C9 RET
129 129
130 130
131 131 ;*****************************************************************************
132 132 ;* Receive one Character Via SIO Channel A(Polling Mode) *
133 133 ;* D- Register: Character received (ASCII Code)  *
134 134 ;* E-Register: Error Code *
135 135 ;*****************************************************************************
136 136 0163 F5 SIO_GET_CHAR: PUSH AF
137 137 0164 DB 42 _RX_READY: IN A,(SIO_A_C) ; Read RR0 Register
138 138 0166 CB 47 BIT 0,A ; RX Character Available ?
139 139 0168 CA 64 01 JP Z,_RX_READY ; No --> Wait
140 140 016B DB 40 IN A,(SIO_A_D) ; Store character
141 141 016D 57 LD D,A
142 142 016E 3E 01 LD A,$01 ; Select WR1 Register
```

```
143  143 0170 D3 42 OUT (SIO_A_C),A
144  144 0172 DB 42 IN A,(SIO_A_C) ; Read Error Register
145  145 0174 5F LD E,A ; store Error Status
146  146 0175 E6 70 AND $70 ; only D6(CRC Framing Error),D5(Rx Overrun Error) und D4 (Parity
         Error)
147  147 0177 CA 7E 01 JP Z,_RX_EXIT ; return if no error
148  148 017A 3E 30 LD A,$30 ; reset Error
149  149 017C D3 42 OUT (SIO_A_C),A
150  150 017E F1 _RX_EXIT: POP AF
151  151 017F C9 RET
152  152
153  153
154  154 ;*******************************************************************************
155  155 ;* SEND STRING to V24 via SIO *
156  156 ;* HL: contains start address of string *
157  157 ;*******************************************************************************
158  158 0180 F5 SIO_PUT_STRING: PUSH AF
159  159 0181 7E _NEXT_CHAR: LD A,(HL) ; get character
160  160 0182 FE 00 CP $00 ; END of String ?
161  161 0184 CA 8F 01 JP Z,_TEXT_END
162  162 0187 57 LD D,A
163  163 0188 CD 56 01 CALL SIO_PUT_CHAR ; send character
164  164 018B 23 INC HL ; next character
165  165 018C C3 81 01 JP _NEXT_CHAR
166  166 018F F1 _TEXT_END: POP AF
167  167 0190 C9 RET
168  168
169  169
170  170 ;*******************************************************************************
171  171 ;* TEXT DEFINITIONS *
172  172 ;*******************************************************************************
173  173 0191 0D 0A 5A 38 START_TEXT: DEFB CR,LF,'Z','8','0',SPACE,'D','E','M','O',SPACE,'V','1','.
         ','0',$00
174       0195 30 20 44 45
175       0199 4D 4F 20 56
176       019D 31 2E 30 00
177  174 01A1 0D 0A 4E 4D NMI_TEXT: DEFB CR,LF,'N','M','I',$00
178       01A5 49 00
179  175
180  176
181  177
182  178
183  179
184  180
185  181
186  182
187  183
188
189  Symbol table:
190
191  CR 000D CTC0 0000 CTC1 0001
192  CTC2 0002 CTC3 0003 CTC_INIT 0124
193  LF 000A MAIN 0100 NMI_TEXT 01A1
```

```
194 PIO_A 0080 PIO_B 0081 PIO_C 0082
195 PIO_CON 0083 PIO_INIT 011F RAMTOP FFFF
196 SIO_A_C 0042 SIO_A_D 0040 SIO_B_C 0043
197 SIO_B_D 0041 SIO_GET_CHAR 0163 SIO_INIT 012D
198 SIO_PUT_CHAR 0156 SIO_PUT_STRING 0180 SPACE 0020
199 START_TEXT 0191 _AGAIN 0113 _NEXT_CHAR 0181
200 _RX_EXIT 017E _RX_READY 0164 _TEXT_END 018F
201 _TX_READY 0157
202 31 symbols.
```

8.6.10.3 Funktionsbeschreibung

Das Programm gleicht in seiner Funktion dem Programm SIO_V24_Echo_Interrupt, jedoch wird das Rücksenden der Zeichen nicht durch eine Interrupt Service Routine realisiert, sondern durch das regelmäßige Abfragen des Eingangs.

9 Kostenkalkulation

9.1 ARM Minimalsystem

Die Kostenkalkulation für die einzelnen Platinen bezieht sich je auf den Fertigungspreis von einem Stück bei einer Gesamtauflage von hundert Stück. Weiters wurden bei der Kostenkalkulation alle verwendbaren Sensoren und Features berücksichtigt. Darüber hinaus wurde davon ausgegangen, dass jedes Stück einzeln erwerbbar ist und es sich um eine kostenlose Lieferung der einzelnen Bauteile handelt.

Daraus ergaben sich folgende Preise im Dezember 2017:

- Core-Modul:
  - Leiterkarte: 0,93 €
  - Bauteile: 29,9508 €
  - Summe: 30,8808 €
- Basisplatine:
  - Leiterkarte: 5,33 €
  - Bauteile: 181,1938 €
  - Summe: 186,5238 €
- USB-to-UART:
  - Leiterkarte: 0,71 €
  - Bauteile: 6,8192 €
  - Summe: 7,5292 €
- Audio Adapter:
  - Leiterkarte: 1,65 €
  - Bauteile: 18,0208 €

  - Summe: 19,6708 €

Anhand dieser Preise ergibt sich ein Gesamtpreis von **244,6046 €** für alle Systeme.

Für die Entwicklung des ARM Minimalsystems entstand ein Kostenaufwand von **3463,34 €**.

9.2 Z80 Minimalsystem

Kosten für die Fertigung von 100 PCBs bei PCBWay inklusive Frachtkosten: 332 $ = 271,01 €

Fertigungskosten für ein Minimalsystem (alle Preise Stand März 2018):

- Leiterkarte: 2,71 €
- Bauteile: 75,50 €
- Summe: 78,21 €

Für die Entwicklung des Z80 Minimalsystems entstand ein Kostenaufwand von **1008,01 €**.

Literaturverzeichnis

[1] STMicroelectronics N.V. (2018). STM32F107RC, Mainstream Connectivity line, ARM Cortex-M3 MCU with 256 Kbytes Flash, 72 MHz CPU, Ethernet MAC, CAN and USB 2.0 OTG, Adresse: `http://www.st.com/en/microcontrollers/stm32f107rc.html` (besucht am 15. 03. 2018).

[2] *Connectivity line, ARM®-based 32-bit MCU with 64/256 KB Flash, USB OTG, Ethernet, 10 timers, 2 CANs, 2 ADCs, 14 communication interfaces*, STM32F107xx, Rev 10, STMicroelectronics N.V., März 2017. Adresse: `http://www.st.com/content/ccc/resource/technical/document/datasheet/e4/f3/1a/89/5a/02/46/ae/CD00220364.pdf/files/CD00220364.pdf/jcr:content/translations/en.CD00220364.pdf` (besucht am 16. 03. 2018).

[3] *Intelligent control LED integrated light source*, WS2812B, WORLDSEMI CO., LIMITED. Adresse: `http://www.seeedstudio.com/document/pdf/WS2812B%20Datasheet.pdf` (besucht am 18. 03. 2018).

[4] *MULTILED Enhanced optical Power LED (ThinFilm / ThinGaN)*, LTRB GFSF, Rev. 2014-08-26, OSRAM Opto Semiconductors GmbH, Aug. 2014. Adresse: `https://docs-emea.rs-online.com/webdocs/0e24/0900766b80e24286.pdf` (besucht am 18. 03. 2018).

[5] *Triple Output I2C Controlled RGB LED Driver*, NCP5623, Rev. 6, Semiconductor Components Industries, LLC, Okt. 2008. Adresse: `https://www.onsemi.com/pub/Collateral/NCP5623-D.PDF` (besucht am 18. 03. 2018).

[6] *Datasheet*, ESP8266EX, Version 5.8, espressif, 2008. Adresse: `https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf` (besucht am 18. 03. 2018).

[7] *Product Data Sheet*, HC-06, Rev. 2.0, Guangzhou HC Information Technology Co., Ltd., Sep. 2006. Adresse: `https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf` (besucht am 18. 03. 2018).

[8] *HC-12 WIRELESS RF UART COMMUNICATION MODULE V2.4 USER MANUAL*, HC-12, V2.4, Dez. 2016. Adresse: `http://statics3.seeedstudio.com/assets/file/bazaar/product/HC-12_english_datasheets.pdf` (besucht am 18. 03. 2018).

[9] *USB UART IC*, FT232R, Version 2.13, Future Technology Devices International Limited, 2015. Adresse: `http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf` (besucht am 16. 03. 2018).

[10] D. Baumhackl und S. Uhl, „Spotlight Positioning System“, Diplomarbeit, HTBL Hollabrunn, März 2016, 135 S. Adresse: `https://netstorage.htl-hl.ac.at/oneNet/NetStorage/DriveG%40TIDATA/_ARCHIV_201516/DA_SpotlightPositioningSystem/Dokumentation/DA-Dokumentation/Doku_SpotlightPositioningSystem_V8.2_gesamt.pdf` (besucht am 18. 03. 2018).

[11] *Stereo Audio CODEC, 8- to 96-kHz, With Integrated Headphone Amplifier*, TLV320AIC23B, Rev. H, Texas Instruments Incorporated, Feb. 2004. Adresse: `http://www.ti.com/lit/ds/symlink/tlv320aic23b.pdf` (besucht am 16. 03. 2018).

[12] *0.65-Ω Dual SPDT Analog Switches With Negative Signaling Capability*, TS5A22364, Rev. H, Texas Instruments Incorporated, Juni 2017. Adresse: `http://www.ti.com/lit/ds/symlink/ts5a22364.pdf` (besucht am 16. 03. 2018).

[13] D. Tinitigan, *Arduino Ethernet Library*, 2016. Adresse: `https://github.com/arduino/Arduino/tree/master/libraries/Ethernet/src/utility` (besucht am 03. 04. 2018).

[14] *W5100 Datasheet*, W5100, Version 1.1.6, WIZnet Co., Inc., Jan. 2008. Adresse: `https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf` (besucht am 03. 04. 2018).

[15] A. Mieke. (2018). Keil µVision 5 Tutorial, Adresse: `https://git.1750studios.com/diploma-thesis/001_Keil_MDK5_Tutorial` (besucht am 14. 03. 2018).

[16] ARM Limited. (2018). Pack with Software Components, Adresse: `https://www.keil.com/pack/doc/CMSIS/Pack/html/cp_SWComponents.html` (besucht am 15. 03. 2018).

[17] *Positive voltage regulator ICs*, L78, Rev. 34, STMicroelectronics N.V., Nov. 2016. Adresse: `https://cdn.hackaday.io/files/256641098008576/27c256.pdf` (besucht am 18. 03. 2018).

[18] *CMOS Programmable Peripheral Interface*, 82C55A, Rev. 11.00, Intersil Corporation, Dez. 2015. Adresse: `https://www.intersil.com/content/dam/Intersil/documents/82c5/82c55a.pdf` (besucht am 18. 03. 2018).

[19] *Z80-SIO*, ZiLOG, Inc., März 1978. Adresse: `ftp://major.butt.care/mirrors/Tandy%20Radio%20Shack%20Software/model2archive/Hardware/Zilog_Z80_SIO_Specification.pdf` (besucht am 18. 03. 2018).

[20] *Z80® CTC Counter/Timer Circuit*, Z8430, ZiLOG, Inc., März 1981. Adresse: `ftp://major.butt.care/mirrors/Tandy%20Radio%20Shack%20Software/model2archive/Hardware/Zilog_Z80_SIO_Specification.pdf` (besucht am 18. 03. 2018).

[21] *256K (32K * 8) PRODUCTION AND UV ERASABLE PROMS*, 27256, Intel Corporation, Sep. 1989. Adresse: `http://datasheet.octopart.com/D27256-2-Intel-datasheet-17852618.pdf` (besucht am 18. 03. 2018).

[22] *NMOS 256 Kbit (32Kb x 8) UV EPROM*, M27256, STMicroelectronics N.V., Nov. 2000. Adresse: `https://cdn.hackaday.io/files/256641098008576/27c256.pdf` (besucht am 18. 03. 2018).

[23] *NMOS/CMOS Z80® DMA Direct Memory Access Controller*, Z8410/Z84C10, ZiLOG, Inc., Sep. 1988. Adresse: `http://datasheet.octopart.com/Z8F0423SJ005EC-Zilog-datasheet-14114258.pdf` (besucht am 18. 03. 2018).

[24] Wikipedia. (2017). ARM Limited, Adresse: `https://de.wikipedia.org/w/index.php?title=ARM_Limited&oldid=172245346` (besucht am 14. 03. 2018).

[25] D. Grabham. (Juli 2013). From a small Acorn to 37 billion chips: ARM's ascent to tech superpower, Adresse: https://www.techradar.com/news/computing/from-a-small-acorn-to-37-billion-chips-arm-s-ascent-to-tech-superpower-1167034 (besucht am 14. 03. 2018).

[26] Wikipedia. (2018). C (Programmiersprache), Adresse: https://de.wikipedia.org/w/index.php?title=C_(Programmiersprache)&oldid=174856768 (besucht am 15. 03. 2018).

[27] ——, (2018). C++, Adresse: https://de.wikipedia.org/w/index.php?title=C%2B%2B&oldid=174161963 (besucht am 15. 03. 2018).

[28] ARM Limited. (2018). Cortex Microcontroller Software Interface Standard, Adresse: https://developer.arm.com/embedded/cmsis (besucht am 14. 03. 2018).

[29] Wikipedia. (2018). Debugger, Adresse: https://de.wikipedia.org/w/index.php?title=Debugger&oldid=173107853 (besucht am 14. 03. 2018).

[30] ——, (2018). Echtzeituhr, Adresse: https://de.wikipedia.org/w/index.php?title=Echtzeituhr&oldid=174079576 (besucht am 17. 03. 2018).

[31] ——, (2018). Integrierte Entwicklungsumgebung, Adresse: https://de.wikipedia.org/w/index.php?title=Integrierte_Entwicklungsumgebung&oldid=174180343 (besucht am 14. 03. 2018).

[32] „IEEE Standard for Test Access Port and Boundary-Scan Architecture", *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, S. 1–444, Mai 2013. DOI: 10.1109/IEEESTD.2013.6515989.

[33] ——, (2018). Keil (company), Adresse: https://en.wikipedia.org/w/index.php?title=Keil_(company)&oldid=822646239 (besucht am 15. 03. 2018).

[34] C. Edwards. (März 2016). ARM and the man, Adresse: http://www.techdesignforums.com/practice/technique/arm-and-the-man/ (besucht am 15. 03. 2018).

[35] Wikipedia. (2018). STMicroelectronics, Adresse: https://de.wikipedia.org/w/index.php?title=STMicroelectronics&oldid=174221034 (besucht am 15. 03. 2018).

Abbildungsverzeichnis

Tabellenverzeichnis

Begriffsverzeichnis

ADC Analog-Digital-Converter. 73, 75, 118, 121, 125, 145, 150, 369, *Siehe:* Analog-Digital-Converter

Analog-Digital-Converter zu deutsch: Analog-Digital-Wandler, ist ein Bauteil, welches Wert- und Zeitkontinuirliche Signale Abtastet und Quantisiert um sie digital weiter verarbeiten zu können. 73, 369

ARM ARM Limited. 21, 30, 32, 40, 59, 61, 118, 144, 163, 234, 369, 371, *Siehe:* ARM Limited

ARM Limited früher: Advanced RISC Machines Ltd. ist ein zur japanischen Firma Softbank gehörender Hersteller von IP (Intellectual property) Software im Bereich Mikroprozessoren. Die gleichnamige Mikroprozessorarchitektur, ARM, ist zur Zeit weltweit am weitesten verbreitet [24] [25]. 21, 369

Basisplatine ist die Baugruppe, auf welche das Core-Modul gesteckt wird. Sie bietet Schalter, LEDs, Sensoren und ein Arduino Shield Interface. Zusammen mit dem Core-Modul komplettiert sie das Minimalsystem. 3, 15, 21, 30, 32, 56–64, 66–72, 74–85, 87–93, 95–104, 106, 107, 133, 134, 158–160, 162, 241, 245

C ist eine Programmiersprache, welche sowohl zur System- als auch zur Anwendungsprogrammierung eingesetzt wird. C ist eine der am weitesten verbreiteten Programmiersprachen weltweit und wurde in den 1970er-Jahren von Dennis Ritchie erfunden. Siehe: [26]. 241, 246, 250, 369

C++ ist eine objektorientierte Erweiterung zu C. C++ wurde 1979 von Bjarne Stroustrup entwickelt. Siehe: [27]. 246

CMSIS Cortex Microcontroller Software Interface Standard. 231, 233, 235–237, 240, 251, 252, 254, 255, 369, *Siehe:* Cortex Microcontroller Software Interface Standard

Core-Modul ist die Baugruppe, auf welcher der Cortex-M3 Prozessor sitzt und Teil des neuen Minimalsystems. 3, 21, 30, 32, 33, 40, 42–55, 57, 61–63, 69, 83, 84, 106, 107, 157, 161, 369

Cortex Microcontroller Software Interface Standard ist ein von ARM erstellter Standard, welcher das Verwenden von Software zwischen verschiedenen Cortex-Prozessoren verschiedener Chip-Hersteller ohne große Anpassungen ermöglichen soll. Hierfür stellt ARM einige Definitionen – wie zum Beispiel CORE, RTOS, DSP, … – zur Verfügung, welche von den Chip-Herstellern implementiert werden, diese stellen

dann CMSIS-Packs zur Verfügung, welche in Softwareprojekte eingebunden werden können. Siehe: [28]. 231, 369

DAC Digital-Analog-Converter. 34, 118, 121, 125, 143, 145, 150, 370, *Siehe:* Digital-Analog-Converter

Debugging oder „Debuggen“ beschreibt das finden und entfernen von Bugs (engl. für Käfer, hier: Programmfehler) mit Hilfe eines Debuggers. Siehe: [29]. 40, 59, 61, 205, 227, 245, 246, 371

Digital Signal Processor zu deutsch: Digitaler Signalprozessor, wird verwendet um digitalisierte Signale weiterzuverarbeiten. 125, 370

Digital-Analog-Converter zu deutsch: Digital-Analog-Wandler, ist ein Bauteil, welches Wert- und Zeitdiskrete Signale analog ausgibt. 34, 370

Digitale Signalverarbeitung ist eine Methodik um ursprünglich analoge Bauelemente wie Filter oder Oszillatoren digital zu realisieren, Siehe auch: DSP. 21, 370

DSP Digital Signal Processor. 125, 370, *Siehe:* Digital Signal Processor

DSV Digitale Signalverarbeitung. 21, 370, *Siehe:* Digitale Signalverarbeitung

Echtzeituhr (englisch *real-time clock*, *RTC*) oder physikalische Uhr ist eine Uhr, welche die physikalische Zeit misst. Im Gegensatz dazu misst eine logische Uhr eine relative Zeit, die nicht der aktuellen Uhrzeit entspricht. Im Bereich der Elektrotechnik bzw. technischen Informatik ist eine Echtzeituhr Teil eines computergesteuerten Gerätes bzw. des Betriebssystems und hält die Uhrzeit vor. Es werden Vorkehrungen getroffen, damit die Uhrzeit nach erneutem Einschalten wieder zur Verfügung steht. Insbesondere ist eine Echtzeituhr ein Schaltkreis, welcher die Uhrzeit (mittels eines eigenen Energiespeichers, etwa einer Batterie) auch bei ausgeschaltetem Gerät fortschreiben kann. Siehe: [30]. 46, 372

Extensible Markup Language ist eine Auszeichnungssprache, welche zur Abspeicherung von strukturierten Daten verwendet wird. 251, 373

Graphical User Interface englisch für „grafische Benutzeroberfläche“. Teil des MMI. 21, 370

GUI Graphical User Interface. 21, 87, 163, 370, *Siehe:* Graphical User Interface

HTL Standard Library ist eine Library für den Cortex-M3, welche HTL-spezifische Funktionen, vor allem im Bereich I/O enthält. 251, 372

IDE integrierte Entwicklungsumgebung. 227, 228, 233, 235, 238, 245, 371, *Siehe:* integrierte Entwicklungsumgebung

integrierte Entwicklungsumgebung ist Deutsch für „Integrated Development Environment“ und beschreibt eine Sammlung von Programmen (Editor, Compiler, Linker, Loader, Debugger), welche zum programmieren verwendet wird [31]. 227, 371

Joint Test Action Group ist ein Synonym für den IEEE Standard 1149.1, welcher eine Methodik zum Debugging von Hardware auf Leiterplatten beschreibt. Siehe: [32]. 57, 371

JTAG Joint Test Action Group. 57, 371, 372, *Siehe:* Joint Test Action Group

Keil Keil Elektronik GmbH. 227, 228, 235, 245, 246, 249, 371, *Siehe:* Keil Elektronik GmbH

Keil Elektronik GmbH war eine deutsche Firma (Anfangs: GbR), gegründet 1982 von Günther und Reinhard Keil. Das Hauptaufgabengebiet lag bei der Entwicklung von Evaluation Boards und der μVision IDE. Keil wurde 2005 von ARM aufgekauft. Siehe: [33] [34]. 227, 371

Least Significant Bit zu deutsch: irrelevantestes Bit, das Bit mit dem kleinsten Wert. 121, 371

Least Significant Byte zu deutsch: irrelevantestes Byte, das Byte mit dem kleinsten Wert. 371

LSB Least Significant Byte. 371, *Siehe:* Least Significant Byte

LSB Least Significant Bit. 121, 205, 371, *Siehe:* Least Significant Bit

Mensch-Maschine-Interface ist ein Interface (z.B.: Display, Tastatur) um die Kommunikation von einem Menschen mit einer Maschine zu ermöglichen. 21, 372

Minimalsystem beschreibt das im Unterricht üblicherweise verwendete – aber auch erweiterbare – Microcontroller System. 15, 21, 30, 32, 40, 59, 61, 118, 144, 163, 177, 227, 235, 243, 251, 369, 372

MMI Mensch-Maschine-Interface. 21, 370, 372, *Siehe:* Mensch-Maschine-Interface

Most Significant Bit zu deutsch: relevantestes Bit, das Bit mit dem höchsten Wert. 121, 372

Most Significant Byte zu deutsch: relevantestes Byte, das Byte mit dem höchsten Wert. 372

MSB Most Significant Byte. 372, *Siehe:* Most Significant Byte

MSB Most Significant Bit. 121, 372, *Siehe:* Most Significant Bit

RTC Echtzeituhr. 46, 51, 372, *Siehe:* Echtzeituhr

Semantic Versioning beschreibt eine Art der Versionierung von Software, welche aus 3 einzelnen Versionsnummern im Format A.B.C besteht, C steht hierbei für Patches (Bugfixes, keine neue Funktionalität), B für Minor Versions (neue Funktionalität, aber weiterhin kompatibel zur Vorgängerversion) und A, was Major Versions (inkompatibel zu älteren Versionen) darstellt. 254

Signal-Noise-Ratio zu deutsch: Signal-Rausch-Abstand, gibt an, wie viele dBzwischen Signal und Rauschen liegen. 121, 372

Single Wire Debug ist ein Subset von JTAG, welches mit weniger Portleitungen auskommt. 32, 372

SNR Signal-Noise-Ratio. 121, 372, *Siehe:* Signal-Noise-Ratio

STDLib HTL Standard Library. 251–253, 372, *Siehe:* HTL Standard Library

STM STMicroelectronics N.V.. 33, 372, *Siehe:* STMicroelectronics N.V.

STM32F107RCT(6) ist der in dieser Diplomarbeit verwendete Microcontroller. 3, 33, 34, 205, 235, 239

STMicroelectronics N.V. ist ein europäischer Halbleiterhersteller mit Sitz in den Niederlanden. Siehe: [35]. 33, 372

SWD Single Wire Debug. 32, 41, 57, 59, 372, *Siehe:* Single Wire Debug

USB-to-UART ist die Baugruppe, welche ein UART Gerät über USB emuliert. Es verwendet hierfür einen FTDI-Chip und ist Teil des neuen Minimalsystems. 3, 29, 30,

43, 57, 87, 105–110, 115–117, 160, 162

XML Extensible Markup Language. 251, 373, *Siehe:* Extensible Markup Language

ZIP ist ein weit verbreitetes Dateiformat, welches zur Archivierung und Kompression von Dateien und Ordnern verwendet wird. Der Name leitet sich aus dem englischen Wort „zipper“ (Reißverschluss) ab. 251, 254