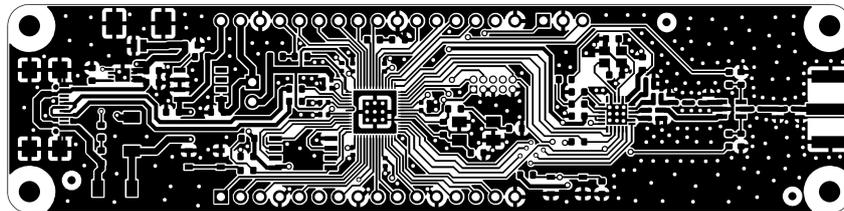


HOCHSCHULE LUZERN
TECHNIK & ARCHITEKTUR

BACHELOR-THESIS DES STUDIENGANGS
ELEKTROTECHNIK UND INFORMATIONSTECHNOLOGIE

FRÜHLINGSSEMESTER 2022

tinyLoRa



Autor:
Julian STAFFELBACH

Betreuender Dozent:
Prof. Erich STYGER

Experte:
Dr. Christian VETTERLI

Industriepartner:
INSTITUT FÜR ELEKTROTECHNIK

Einreikedatum:
10. JUNI 2022

Bachelor-Thesis an der Hochschule Luzern - Technik & Architektur

Titel	tinyLoRa
Diplomandin/Diplomand	Staffelbach, Julian
Bachelor-Studiengang	Bachelor Elektrotechnik und Informationstechnologie
Semester	FS22
Dozentin/Dozent	Styger, Erich
Expertin/Experte	Vetterli, Christian

Abstract Deutsch

Für das Institut für Elektrotechnik an der Hochschule Luzern soll ein kleines, vielseitiges und batteriebetriebenes Mikrocontrollerboard mit integriertem LoRa-Transceiver entwickelt werden. Bei LoRa handelt es sich um eine Modulationstechnik, die dafür ausgelegt ist, mit kleinem Energieaufwand über weite Strecken drahtlos zu kommunizieren. Dies kann verwendet werden, um in abgelegenen und schwer zugänglichen Orten bedeutende Sensordaten mit batteriebetriebenen Geräten zu erfassen und versenden. Am Schluss dieses Projektes soll ein funktionstüchtiges Board vorliegen, das LoRa-Signale senden und empfangen kann. Dafür werden geeignete Komponenten evaluiert, die auf einem PCB kompakt verbaut werden. Der bereits vorhandene Kommunikationsstack wird auf die verwendete Mikrocontrollerarchitektur portiert, die bis anhin noch nicht unterstützt wird. Das Ziel ist, dass zwei Boards über LoRa miteinander kommunizieren können.

Abstract English

In cooperation with the Institute for Electrical Engineering at the Lucerne University of Applied Sciences and Arts, a small and widely useable battery-operated microcontroller board with an integrated LoRa-transceiver has been developed. LoRa is an RF modulation technology that enables communication over long distances, requiring only a small amount of energy. This can be used to gain valuable data with battery-operated devices in remote areas and send them via LoRa on the network. The aim of this project is to develop a fully functional board, which is able to send and receive LoRa-signals. Therefore, suitable components were evaluated, which have been built on the designed PCB. The already existing software stack was ported to the used microcontroller architecture, which has not been supported up until now. The goal is that the two boards should be able to communicate with each other.

Ort, Datum Horw, 10. Juni 2022
© Julian Staffelbach, Hochschule Luzern – Technik & Architektur

Alle Rechte vorbehalten. Die Arbeit oder Teile davon dürfen ohne schriftliche Genehmigung der Rechteinhaber weder in irgendeiner Form reproduziert noch elektronisch gespeichert, verarbeitet, vervielfältigt oder verbreitet werden.

Sofern die Arbeit auf der Website der Hochschule Luzern online veröffentlicht wird, können abweichende Nutzungsbedingungen unter Creative-Commons-Lizenzen gelten. Massgebend ist in diesem Fall die auf der Website angezeigte Creative-Commons-Lizenz.

Inhaltsverzeichnis

1	Einleitung	1
2	Einführung	2
2.1	Projektübersicht	2
2.1.1	Aufgabenstellung und Anforderungen	2
2.1.2	Risiken	3
2.1.3	Vorgehen	4
2.2	LoRa und LoRaWAN	6
2.2.1	Begriffsdefinitionen	6
2.2.2	LoRa	6
2.2.3	LoRaWAN	7
2.2.4	The Things Network	8
2.3	RP2040 Mikrocontroller	8
2.4	Raspberry Pi Pico	9
2.4.1	Werkzeuge	9
2.5	Installation und Einrichtung der Werkzeuge	10
2.5.1	Anlegen der Ordnerstruktur und Installation der Tools	10
2.5.2	Ergänzung und Überprüfung der Systemvariablen	13
2.5.3	Installation und Konfiguration von Visual Studio Code	13
2.5.4	Installation der Debug-Tools	15
2.5.5	Debug-Konfiguration von Visual Studio Code	17
3	Hardware	18
3.1	Komponenten	18
3.1.1	Mikrocontroller: RP2040	18
3.1.2	Flash: W25Q128JVSIM	19
3.1.3	Spannungsregler: AP2112M-3.3TRG1	19
3.1.4	Batterie-Lademanagement: MCP73821-5ACI/MC	19
3.1.5	USB-C Buchse: USB4110-GF-A	19
3.1.6	LoRa-Transceiver: SX1261	20
3.1.7	HF-Schalter: AS213-92LF	20
3.1.8	Antenne: X9000984-4GDSMB	20
3.1.9	Clock	20
3.1.10	LED	21
3.1.11	Button	21
3.2	Schema	21
3.2.1	Spannungsversorgung	22
3.2.2	Mikrocontroller und Flash	23
3.2.3	LoRa-Transceiver, Anpassungsnetzwerk und Antenne	25
3.2.4	Breakout-Header	26
3.3	PCB	27
3.3.1	Spannungsversorgung	28
3.3.2	Mikrocontroller, Flash und Breakout-Pins	28
3.3.3	LoRa-Transceiver, Anpassungsnetzwerk und Antenne	29
3.3.4	Inbetriebnahme und Korrekturen des PCBs	29

4 Software	31
4.1 Struktur der Software	31
4.2 CMake	34
4.3 Pico SDK	36
4.4 Hardwareabstraktion	36
4.4.1 GPIO mit RP2040	36
4.4.2 SPI mit RP2040	37
4.4.3 Flash-Programmierung mit RP2040	37
4.4.4 Timer mit RP2040	38
4.4.5 Weitere Module	39
4.5 Resultate Software	39
5 Messungen	41
5.1 Energiemessung	41
5.1.1 Energieverbrauch tinyLoRa bei Batteriespannung	41
5.1.2 Energieverbrauch tinyLoRa bei USB-Spannung	42
5.1.3 Energieverbrauch Mikrocontroller	42
5.1.4 Fazit aus Energiemessungen	43
5.2 Antenne	43
6 Resultate	46
6.1 Kosten	46
6.2 Anforderungen	46
6.3 Optimierungsmöglichkeiten und Ausblick	47
7 Fazit	48
Verzeichnisse	49
Abbildungsverzeichnis	49
Tabellenverzeichnis	50
Codeverzeichnis	50
Quellenverzeichnis	50
A Aufgabenstellung tinyLoRa	53
B Schema	55
C Bill of Materials	56
D Pinout tinyLoRa	57
E Elektronischer Anhang	58

1 | Einleitung

Da in der modernen Welt die Digitalisierung und Vernetzung aller Lebensaspekte immer schneller voranschreitet, soll es möglich sein, auch in abgelegenen oder schwer zugänglichen Gebieten bedeutende Sensordaten zu erfassen und zu versenden. Dazu bietet sich LoRa an. LoRa ist eine drahtlose Modulationstechnik, die entwickelt wurde, um kleine Mengen an Daten mit wenig Energie über weite Strecken zu versenden. Aufgrund des kleinen Energieverbrauchs ist diese Technologie ein optimales Mittel, um beispielsweise Sensordaten mit einem batteriebetriebenen Gerät zu versenden. Das LoRa-Protokoll wird immer häufiger eingesetzt und findet aufkommende Beliebtheit. Aus diesem Grund soll an der Hochschule Luzern – Technik und Architektur im Rahmen dieser Bachelor-Thesis ein kleines, vielseitiges und batteriebetriebenes Mikrocontrollerboard mit integriertem LoRa-Transceiver entwickelt werden. Das Board soll steckbrettcompatibel sein und in C programmiert werden können.

Die Schwerpunkte dieser Arbeit belaufen sich auf das Hardwaredesign und die Programmierung eines Mikrocontrollerboards. Dazu gehört die Evaluierung von geeigneten Komponenten, um allen Anforderungen an die Hardware gerecht zu werden. Das Hardwaredesign des Boards führt alle Komponenten auf einem PCB zu einem funktionierenden System zusammen. Danach folgt die Portierung des bereits vorhandenen Kommunikationsstacks in C, der auf dem verwendeten Mikrocontroller laufen wird und dazu dient, den LoRa-Transceiver anzusteuern. Als letztes folgt die Auslegung eines Impedanzanpassungsnetzwerks der Antenne, die zum Senden und Empfangen von Daten verwendet wird. Ziel des Projektes ist es, eine erste Version des Boards zu erstellen, mit dem Daten über LoRa versendet und empfangen werden können. Eine definierte Benutzeranwendung ist dafür jedoch nicht vorgesehen. Das zu entwickelnde Board verwendet den LoRa-Transceiver SX1261 vom Hersteller Semtech. Des Weiteren wird der Mikrocontroller RP2040 von der Firma Raspberry Pi verbaut, wobei es sich um einen Dual ARM Cortex M0+ handelt. Auf dem Board soll eine Batterieladeschaltung vorhanden sein, um eine angeschlossene Batterie über USB laden zu können.

Als Hilfestellung zum Entwerfen der Hardware existieren einerseits Referenzdesigns von Semtech und andererseits detaillierte Dokumentationen von Raspberry Pi zum RP2040-Chip. Der Kommunikationsstack wurde von Semtech bereits für verschiedene Mikrocontrollerarchitekturen implementiert, jedoch nicht für den RP2040. Dies bedeutet, dass die Hardwareabstraktionsschicht vom vorhandenen Projekt auf den RP2040 portiert werden muss. Um die Software während der Entwicklung fortlaufend zu testen, kann eine bereits existierende Testhardware verwendet werden. Anschliessend werden verschiedene Impedanzanpassungen ausgelegt, um mit der verwendeten Antenne die grösstmögliche Sendeleistung zu erreichen. Als Funktionsdemonstration werden zwei der entwickelten Boards bestückt, die miteinander drahtlos kommunizieren sollen.

In dem folgenden Kapitel werden die wichtigsten Begriffe und Funktionsweisen von LoRa erklärt und die Ausgangslage des Projektes erläutert. Darauf folgt eine detaillierte Beschreibung der verwendeten Komponenten und Entwicklung des PCBs. Der Softwarestack wird vorgestellt und die Portierung dessen auf die verwendete Mikrocontrollerarchitektur beschrieben. Zum Schluss werden verschiedene Messungen durchgeführt, ein Fazit gezogen und Optimierungsmöglichkeiten des Projektes vorgestellt.

2 | Einführung

Dieses Kapitel befasst sich mit den Anforderungen, der Ausgangslage und dem Vorgehen der Bachelorarbeit *tinyLoRa*. Die Aufgabenstellung und Anforderungen werden definiert und das geplante Vorgehen erklärt. Des Weiteren werden die viel verwendeten Begriffe LoRa und LoRa-WAN erläutert und deren Hintergründe erklärt. Der verwendete Mikrocontroller wird vorgestellt und dabei erklärt, was es braucht, um ein Board mit diesem Mikrocontroller aufzusetzen.

2.1 | Projektübersicht

2.1.1 | Aufgabenstellung und Anforderungen

Die Aufgabenstellung der Bachelorarbeit *tinyLoRa* ist, ein kleines und vielseitiges Mikrocontrollerboard zu entwerfen, welches den LoRa-Transceiver SX1261 von der Firma Semtech verbaut hat und somit LoRaWAN fähig ist. Es soll batteriebetrieben und kostengünstig sein. Ein Kommunikationsstack ist vorhanden, der auf den verwendeten Mikrocontroller portiert werden muss. Das Layout soll mit dem open-source Design-Tool *KiCad* realisiert werden. Die gesamte Aufgabenstellung findet sich im Anhang unter Kapitel (A) .

Für die gegebene Aufgabenstellung wurde eine Anforderungsliste erstellt, welche konkrete Vorgaben zum Projekt definiert. Die wichtigsten Punkte sind im Folgenden erläutert.

Mikrocontroller Es wird der RP2040 Mikrocontroller verwendet. Es handelt sich um denselben Controller, der auf dem Raspberry Pi Pico verbaut ist. Debugging soll zudem möglich sein.

LoRa-Transceiver Der LoRa-Transceiver SX1261 soll für das Projekt verwendet werden. Dazu stehen von der Firma Semtech Referenzdesigns zur Verfügung, die als Ausgangslage benutzt werden können. Der Transceiver wird von dem Mikrocontroller via SPI angesteuert.

Antenne Zum Zeitpunkt des Projektbeginns ist noch nicht klar, ob die Antenne direkt auf dem PCB realisiert wird oder ob diese extern angebracht ist. Die Antenne wird ausgemessen und angepasst.

Board Das Board soll Steckbrettcompatibel sein und es werden möglichst viele Pins nach aussen geführt, um sie dem Benutzer zur Verfügung zu stellen. Auf dem Board werden Jumper und Testpunkte für die Energiemessung und Signalbeobachtungen angebracht. Zusätzlich wird sich auf dem Board ein Stecker für eine Batterie, eine LED, ein Reset- sowie ein Bootselect-Button befinden. Die Verbindung zu einem PC wird durch einen USB-C Stecker realisiert. Der Kostenpunkt des gesamten Boards soll sich auf ungefähr 30 CHF belaufen.

Entwicklungsumgebung Es wird vorzugsweise Visual Studio Code für das Entwickeln von Software verwendet. Eine Alternative würde sich in der Eclipse IDE finden. Programmiert wird in der Sprache C.

Software Ob auf dem Mikrocontroller ein Betriebssystem wie FreeRTOS oder eine Bare-Metal Anwendung läuft, ist noch nicht klar definiert. Es sollen geeignete Softwaremodule erstellt werden, damit der Benutzer das Board einfach verwenden kann.

Anleitung Nach der Inbetriebnahme des Boards soll eine Anleitung geschrieben werden, wie das Board aufgesetzt wird und verwendet werden kann.

2.1.2 | Risiken

Die für das Projekt relevantesten Risiken werden hier aufgeführt und Massnahmen gegen diese erläutert.

1. **Verfügbarkeit und Lieferfristen der Bauteile:** Viele Bauteile sind momentan nicht lieferbar. Lieferfristen können sich spontan ändern und so das Projekt verzögern.
 - (a) Bei der Komponentenauswahl ist darauf zu achten, dass alle Bauteile beim Lieferanten noch zu grösseren Mengen vorhanden sind. Zudem wird die Verfügbarkeit bei anderen Zulieferern geprüft, um so nicht nur von einem Lieferanten abhängig zu sein.
 - (b) Die Hardware soll schnell ausgewählt und bestellt werden, um so im Falle einer unerwarteten Änderung rechtzeitig reagieren zu können.
2. **Antennenanpassung:** Die Antennenanpassung könnte nicht optimal verlaufen, sodass die Antennenleistung nicht gut genug ausgenutzt wird.
 - (a) Mit dem Thema der Antennenanpassung wird sich intensiv auseinandergesetzt. So sollen allfällige Wissenslücken geschlossen werden, um Fehler zu vermeiden.
 - (b) Sollten dennoch Unklarheiten bestehen, können Drittpersonen zu Hilfe genommen werden.
3. **Fehler beim Layout:** Beim Layout können Fehler passieren, die dazu führen, dass die Schaltung nicht funktioniert.
 - (a) Es werden zwei Layout-Iterationen geplant, dass in einem Fehlerfall noch Anpassungen gemacht werden können.
 - (b) Beim Zeichnen des Schemas ist darauf zu achten, dass alle Anschlüsse der Bauteile einem anderen Anschluss zugewiesen sind. Das Schema wird vor dem Layouten des PCBs mit den vorhandenen Referenzdesigns verglichen.
 - (c) Vor dem Layouten des PCBs wird von einer Drittperson ein Review des Schemas gemacht. Dies erhöht die Chance, dass allfällige Fehler noch rechtzeitig entdeckt werden können.
4. **Fehler in der Software:** In der Software, vor allem auf der Treiberebene, können Fehler passieren, die unter Umständen die Funktionalität des Boards stark beeinträchtigen.
 - (a) Die Testhardware wird früh in Betrieb genommen und dort die verschiedenen Softwarefunktionen konfiguriert und getestet.
 - (b) Durch Debugging ist die Möglichkeit vorhanden, Fehler zu erkennen und zu korrigieren.

2.1.3 | Vorgehen

Das Projekt wird in einem Zeitrahmen von 15 Wochen umgesetzt. Im Folgenden wird erklärt, wie die Ausgangslage des Projektes aussieht und wie über den Zeitrahmen von 15 Wochen daraus das tinyLoRa-Board entsteht. Dies ist auch in der untenstehenden Tabelle (1) graphisch dargestellt.

Aufgabe:	Woche:														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Inbetriebnahme Pico & LPC55S16	■	■													
Komponentenwahl- & bestellung		■	■												
Schema & PCB erstellen			■	■	■										
LoRa-Stack auf Pico portieren						■	■	■	■						
PCB testen									■	■					
Schema & PCB korrigieren										■	■				
Antenne anpassen												■	■		
Hard- & Software testen														■	■

Tabelle 1: Projektplan tinyLoRa

Als Ausgangslage dient einerseits das Raspberry Pi Pico, ein Mikrocontrollerboard mit dem RP2040, welches umfangreich dokumentiert ist und andererseits der SX1261MB2BAS-Shield, der den LoRa-Transceiver SX1261 mit der Antennenanpassung und Antennenbuchse enthält. Dieser Shield ist von Semtech und kompatibel mit dem LPC55S16-Board von NXP. Zu diesem Board ist bereits ein Projekt vorhanden, das zusammen mit dem Shield LoRaWAN kompatibel ist. In einem ersten Schritt wird das LPC55S16-Board mit dem LoRa-Shield in Betrieb genommen, was in Abbildung (1) dargestellt ist. Somit ist eine funktionierende Anwendung vorhanden, die für spätere Projektphasen als Leitfaden dient. Zusätzlich wird die gesamte Toolchain für das Raspberry Pi Pico installiert und Visual Studio Code aufgesetzt, um damit verschiedene Boards, die den RP2040 verwenden, programmieren zu können. Die wichtigsten Hardwarefunktionalitäten für die LoRa-Anwendung sind die GPIO-Ansteuerung, SPI-Kommunikation und Flash-Programmierung, welche zu Beginn des Projektes auf dem Raspberry Pi Pico implementiert und getestet werden. So kann ein Verständnis für das API des RP2040 gewonnen werden und es wird sichergestellt, dass die nötige Hard- und Software funktioniert.

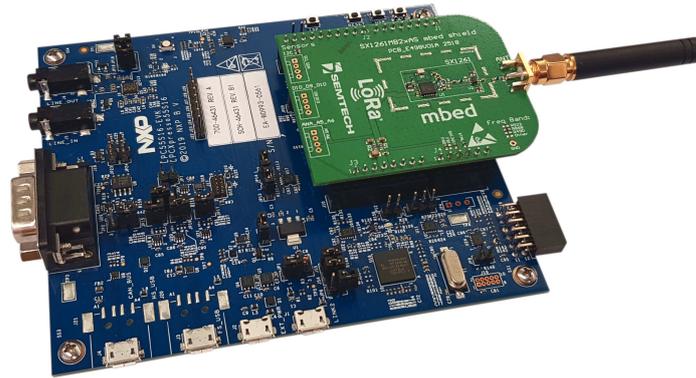


Abbildung 1: LPC55S16 mit SX1261 LoRa-Shield

Nach der erfolgreichen Inbetriebnahme werden die nötigen Komponenten für das tinyLoRa zusammengetragen. Dabei kann auf verschiedene Referenzdesigns und Dokumentationen von Raspberry Pi und Semtech zurückgegriffen werden. Es ist darauf zu achten, dass alle Komponenten noch in grösseren Mengen vorhanden sind und zeitnah angeschafft werden, um lange Lieferfristen zu vermeiden. Gleichzeitig kann das Schema und danach das PCB erstellt werden.

Während dem das PCB gefertigt und geliefert wird, kann damit angefangen werden, den LoRa-Stack auf den Mikrocontroller RP2040 zu portieren. Dieser ist von Semtech bereits für verschiedene Plattformen implementiert, jedoch nicht für den RP2040. Zur Portierung gehört einerseits das Konfigurieren der Toolchain, insbesondere die verschiedenen Buildabhängigkeiten mit CMake, und andererseits das Implementieren der Hardwarefunktionalitäten. Sobald der Stack läuft, wird dieser auf dem Raspberry Pi Pico und dem LoRa-Shield auf Steckbrettern getestet. Dabei werden auf dem Pico die gleichen Pins für die Anwendung verwendet, wie diese die später für das tinyLoRa vorgesehen sind. Durch das Testen auf dem Pico und Shield wird sichergestellt, dass die Hardware einwandfrei funktioniert und alle allfälligen Fehler in der Software ausfindig zu machen sind.

In einem weiteren Schritt wird das PCB bestückt und mit einem simplen Testprogramm das Funktionieren des Mikrocontrollers sichergestellt. Danach werden die verschiedenen Funktionalitäten an den Breakout-Pins getestet und schlussendlich der LoRa-Transceiver in Betrieb genommen. Dazu muss der Kommunikationsstack fertig portiert sein. Durch die sukzessive Inbetriebnahme können Teile des Boards separat getestet werden, was die Fehlersuche vereinfacht.

Falls Fehler in der entworfenen Hardware ausfindig gemacht werden, werden diese schnellstmöglich korrigiert und das PCB wird nochmals bestellt. Dabei ist wichtig, dass dies aufgrund der Lieferzeiten so früh wie möglich gemacht wird.

Zum Schluss werden die Impedanzen des LoRa-Transceivers ausgemessen und mit dem Anpassungsnetzwerk auf dem PCB angepasst, um die grösstmögliche Leistung aus der Antenne zu gewinnen. Dazu werden verschiedene Filter und Impedanzanpassungen berechnet und simuliert.

In verschiedenen Testfällen wird das tinyLoRa getestet und somit können die an das Projekt gestellten Anforderungen verifiziert werden.

2.2 | LoRa und LoRaWAN

Die Informationen der folgenden Unterkapiteln sind aus dem Artikel "What are LoRa[®] and LoRaWAN[®]?" [22] entnommen.

2.2.1 | Begriffsdefinitionen

Es werden hier die wichtigsten Begriffe, die in den folgenden Unterkapiteln zum Thema LoRaWAN vorkommen, kurz erläutert.

Gateway Empfängt LoRa-modulierte Nachrichten von End-Devices und leitet diese an die Server weiter.

End-Device Ein Gerät, das mit dem LoRaWAN-Netzwerk verbunden ist und Daten an einen oder mehrere Gateways übertragen kann.

Uplink Kommunikationsfluss vom End-Device zu einem oder mehreren Gateways.

Downlink Kommunikationsfluss von einem Gateway zu einem End-Device.

Spreading Factor Beschreibt die Art, in der die Signale moduliert werden. Damit kann die Datenrate und die Reichweite beeinflusst werden.

Time on Air Die Zeit, in der Signale gesendet werden.

2.2.2 | LoRa

LoRa (abgeleitet von Long Range) beschreibt eine drahtlose Modulationstechnik, mit der kleine Datenmengen über grosse Strecken verschickt werden können. Aufgrund einer niedrigen Datenrate ist LoRa ein sehr energieeffizientes Protokoll und deshalb optimal für batteriebetriebene Anwendungen. Typische Beispiele sind das Versenden von Sensordaten, die über weite Strecken wenige Male am Tag oder in der Woche verschickt werden. Wie in Abbildung (2) zu sehen ist, beschreibt LoRa die physikalische Schicht des OSI-Schichtenmodells, es sind also die Modulation der Signale, die Bandbreiten und die Trägerfrequenzen definiert.

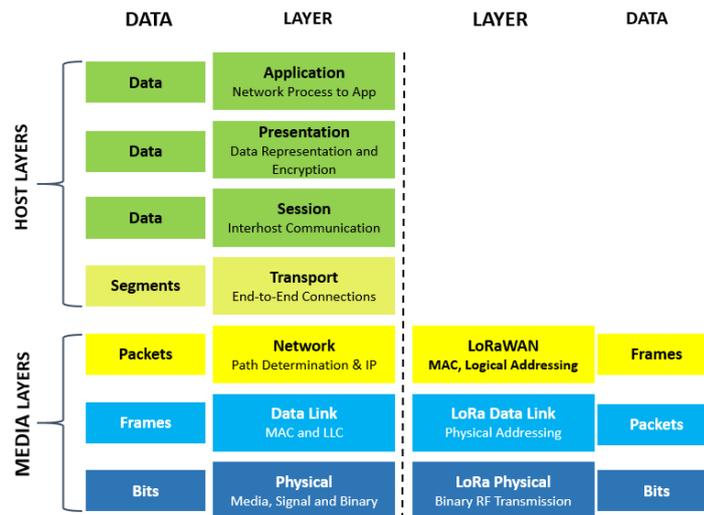


Abbildung 2: OSI-Schichtenmodell mit LoRa und LoRaWAN
(Quelle: Semtech Corporation, 2017 [22])

In der EU beträgt diese Trägerfrequenz ungefähr 868 MHz , die Bandbreiten befinden sich meist um die 125 kHz für Uplink und 500 kHz für Downlink. Mit dem Spreading Factor kann ein Kompromiss aus Datenrate und Reichweite für die jeweilige Anwendung gefunden werden. Ist eine höhere Datenrate gewünscht, nimmt die maximal mögliche Distanz, aber auch die Time on Air ab. Muss über sehr grosse Strecken kommuniziert werden, kann dies mit einem höheren Spreading Factor und entsprechend einer geringeren Datenrate erreicht werden. Unterschiedliche Spreading-Faktoren führen zu unterschiedlichen Modulationen der Signale.

2.2.3 | LoRaWAN

LoRaWAN ist das Netzwerkprotokoll, das der LoRa-Modulation sichere, bidirektionale Datenübertragung, Lokalisierungsdienstleistungen und weitere Netzwerkdienste zur Verfügung stellt. LoRaWAN beschreibt im OSI-Schichtenmodell die Netzwerkschicht (Abbildung (2)). Diese wird im Falle von LoRa durch die LoRa Alliance unterhalten. Das Netzwerk ist so aufgebaut, dass die Gateways die empfangenen Daten von den End-Devices an einen Server weiterleiten. Das Gateway überprüft dabei eine Checksumme. Falls diese inkorrekt ist, wird die vom End-Device gesendete Nachricht ignoriert und nicht weitergeleitet. Der Server stellt die Daten danach online zur Verfügung und somit können sie vom Benutzer abgerufen werden. Wenn ein End-Device eine Nachricht verschickt, wird diese von allen Gateways, die sich in der Nähe befinden, empfangen, was die Redundanz erhöht. Die Architektur von den verschiedenen Teilnehmern im Netzwerk ist in Abbildung (3) illustriert. Bevor ein Gerät eingesetzt werden kann, muss dieses im Netzwerk registriert werden. So wird eine möglichst hohe Sicherheit im Netzwerk garantiert.

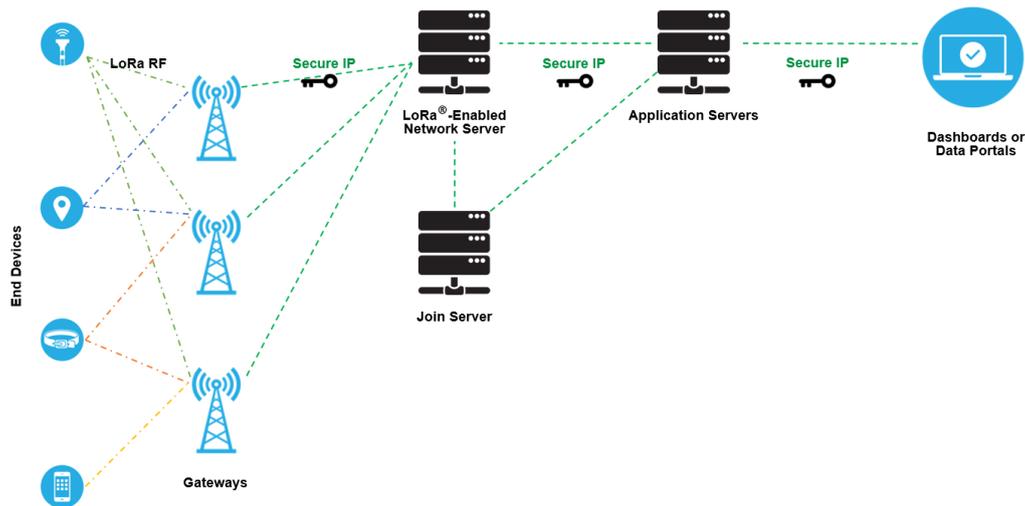


Abbildung 3: LoRaWAN Architektur mit End-Devices, Gateways und Servern
(Quelle: Semtech Corporation, 2017 [22])

2.2.4 | The Things Network

The Things Network ist ein Community-LoRaWAN-Netzwerk. Dies bedeutet, dass jede Person ein Gateway oder End-Device auf dem Server registrieren und dann das Netzwerk benutzen kann. Zu den Vorteilen gehört, dass so jeder kostenlos seine Geräte betreiben und einfach abrufen kann. Der Nachteil besteht darin, dass alle Gateways von privaten Personen oder Organisationen eingerichtet sind und somit nicht jede Region abgedeckt ist. Die Hochschule Luzern Technik & Architektur betreibt beispielsweise ein Gateway für The Things Network und somit ist die Region um Horw sehr gut abgedeckt. Das tinyLoRa soll als End-Device auf diesem Netzwerk registriert und verwendet werden.

2.3 | RP2040 Mikrocontroller

Der in diesem Projekt verwendete Mikrocontroller, der RP2040, wird in diesem Unterkapitel vorgestellt. Beim RP2040 handelt es sich um einen Dual ARM Cortex M0+, der von der Firma Raspberry Pi entwickelt und hergestellt wird. Auf dem Mikrocontroller sind 264 *kB* SRAM sowie 16 *kB* ROM vorhanden. Flash-Speicher ist auf dem Chip nicht vorhanden, extern werden Speicher von bis zu 16 *MB* unterstützt, welche mit QSPI angesteuert werden. Der Chip enthält einen fix programmierten ROM-Bootloader. Es sind 30 GPIO-Pins vorhanden, welche auf folgende Peripherie gemuxt werden können:

- 2 UART
- 2 SPI
- 2 I2C
- 16 PWM Kanäle
- 4 ADC Inputs

(S. 9, S. 147, [17])

Dieser Mikrocontroller wurde gewählt, da dieser trotz dem globalen Halbleiterengpass noch in grossen Stückzahlen und kostengünstig verfügbar ist. Zudem wird von Raspberry Pi eine umfangreiche Dokumentation zur Verfügung gestellt, welche Angaben zum Hardwaredesign, sowie zum Aufsetzen des Mikrocontrollers beinhaltet. Es existiert ein grosses API, mit dem alle Funktionalitäten genutzt werden können. Der Mikrocontroller kann in C programmiert werden und eine kostengünstige Debug-Probe kann mit einem Raspberry Pi Pico Board erstellt werden, die das SWD-Protokoll nutzt.

2.4 | Raspberry Pi Pico

Das Raspberry Pi Pico (Abbildung (4)) ist ein Mikrocontrollerboard, auf dem der RP2040-Chip verbaut ist. Es wird ebenfalls von Raspberry Pi entworfen und vertrieben. Viele Aspekte des tinyLoRa sind an das Raspberry Pi Pico angelehnt. Beispielsweise erfolgt das Aufsetzen und Verwenden beider Boards identisch. Es kann ein Raspberry Pi Pico verwendet werden, um das tinyLoRa-Board zu debuggen. Dazu stellt Raspberry Pi ein spezielles Programm zur Verfügung, welches das SWD Protokoll emuliert.

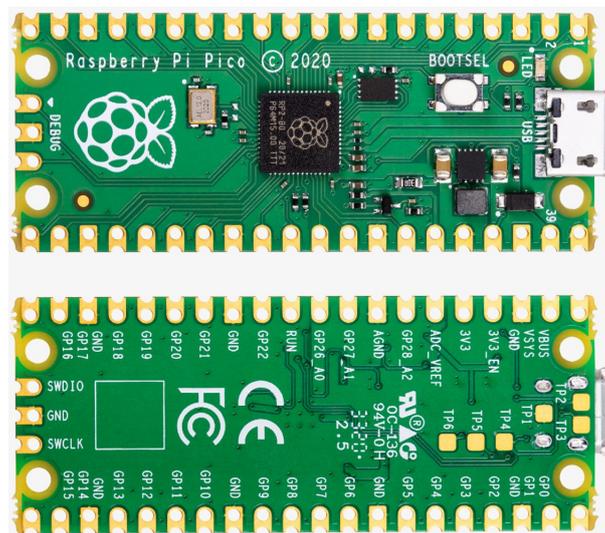


Abbildung 4: Raspberry Pi Pico Mikrocontrollerboard
(Quelle: Raspberry Pi Trading Ltd, 2021 [16])

2.4.1 | Werkzeuge

Hier wird beschrieben, welche Werkzeuge nötig sind, um Programme für das Raspberry Pi Pico zu übersetzen und zu debuggen.

IDE Als IDE wird Visual Studio Code (VS Code) verwendet. VS Code ist im eigentlichen Sinne keine IDE, kann aber mit diversen Plugins zu einer fast vollumfänglichen IDE erweitert werden. Beispielsweise existieren Plugins für die Unterstützung von CMake oder OpenOCD für das Debuggen von Programmen [25].

Toolchain Um die Projekte für den RP2040 zu kompilieren, wird die GNU Arm Toolchain verwendet. Des Weiteren wird MinGW (Minimalist GNU for Windows) verwendet um die Projekte in das richtige Fileformat zu übersetzen, um es auf den RP2040 flashen zu können. Raspberry

Pi schlägt für diesen Schritt Build Tools for Visual Studio vor (S. 25, [13]), jedoch ist dies im Gegensatz zu MinGW nicht open-source, weshalb MinGW verwendet wird [25].

CMake Um den Build-Prozess zu automatisieren und erleichtern, wird CMake verwendet. CMake ist ebenfalls open-source und wird von Raspberry Pi vorgeschlagen (S. 36, [25]).

Pico SDK Das SDK für den RP2040 beinhaltet viele Bibliotheken und APIs, um mit dem RP2040 einfach diverse Funktionalitäten zu benutzen. Es wurde von Raspberry Pi entwickelt. Es sind beispielsweise Funktionen vorhanden um einfach GPIO-Pins, I2C, SPI oder den Flash zu verwenden [15].

OpenOCD Dies ist ein Tool, das zu Laufzeit des Programmes vom Host aus Kommandos an eine Debug-Probe schickt, um das Target zu debuggen. OpenOCD unterstützt den RP2040 nicht offiziell, es muss also dafür konfiguriert und neu übersetzt werden [24].

Picoprobe und Picotool Als Debug-Probe kann ein Raspberry Pi Pico Mikrocontrollerboard verwendet werden. Auf dieses zusätzliche Board kann das Programm Picoprobe geladen werden, welches von Raspberry Pi entwickelt wurde. Dieses Programm emuliert das Debug-Protokoll und so kann man mit der Probe ein anderes Board, das den RP2040 verbaut hat, debuggen. Um eine Verbindung vom Host auf die Debug-Probe mit OpenOCD herzustellen und aufrechtzuerhalten, wird zusätzlich noch das Programm Picotool verwendet, welches ebenfalls von Raspberry Pi entwickelt wird [24].

2.5 | Installation und Einrichtung der Werkzeuge

Die oben genannten Werkzeuge müssen installiert und richtig konfiguriert werden, damit sich mit diesen ein Board, das den RP2040-Chip verwendet, in Betrieb nehmen und debuggen lässt. Dies wird im Folgenden detailliert für das Windows-Betriebssystem beschrieben. Dazu wird die Anleitung von Hymel (2021, [25] und 2021, [24]) befolgt. Die komplette Installation benötigt einen Zeitaufwand von ungefähr sechs bis acht Stunden.

2.5.1 | Anlegen der Ordnerstruktur und Installation der Tools

Für eine möglichst universale Verwendung wird folgende Ordnerstruktur verwendet, damit sich alle Tools an einem klar definierten Ort befinden. Diese muss wie in Abbildung (5) gezeigt auf dem Datenträger **C:** angelegt werden.

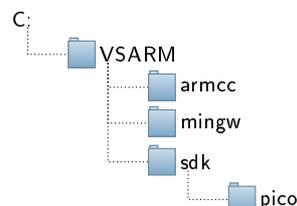


Abbildung 5: Installationsverzeichnis der Toolchain

Als erstes muss die "GNU Arm Embedded Toolchain" installiert werden, welche den C/C++ Compiler enthält, der die Anwendung übersetzt. Dazu finden sich auf dieser [Webseite](https://developer.arm.com/downloads/-/gnu-rm)¹ verschie-

¹<https://developer.arm.com/downloads/-/gnu-rm>

dene Downloads. Es wird der neuste Installer für Windows heruntergeladen, der Name lautet je nach Version unterschiedlich, ein Beispiel wäre:

```
gcc-arm-none-eabi-10.3-2021.10-win32.exe
```

Die Toolchain wird in den Ordner `C:/VSARM/armcc` installiert (Abbildung (6)). Wichtig dabei ist, dass bei der Installation die Option, den Pfad zu den Umgebungsvariablen des Betriebssystems hinzuzufügen, anzuwählen (Abbildung (7)). Somit können die ausführbaren Dateien von der Konsole von jedem Ort aus aufgerufen und ausgeführt werden.

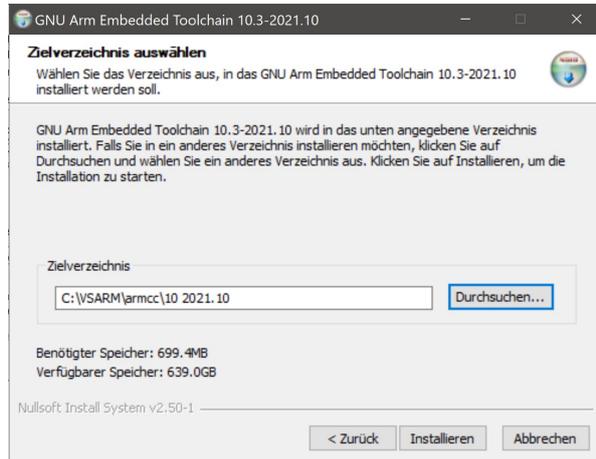


Abbildung 6: Zielpfad der Toolchain

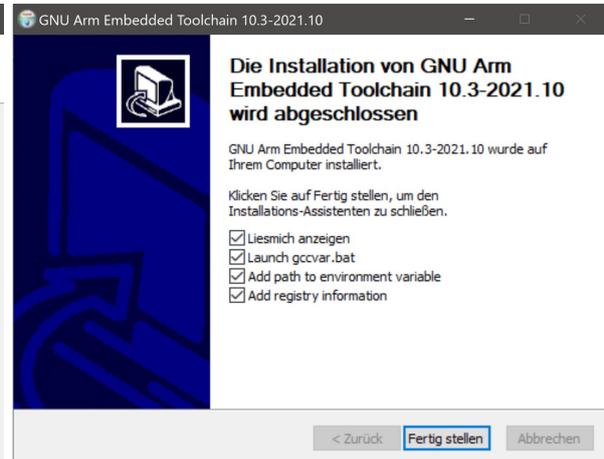


Abbildung 7: Pfadvariable zur Toolchain

Als nächstes wird MinGW in den Unterordner `C:/VSARM/mingw` installiert. MinGW enthält verschiedene Werkzeuge, die nebst dem Compiler benötigt werden, um eine ausführbare Datei zu generieren. Auf dieser [Webseite](#)² finden sich verschiedene Versionen, das benötigte File heisst `i686-posix-sjlj` und muss heruntergeladen und in das oben genannte Verzeichnis entpackt werden. Damit das zum Übersetzten von Applikationen nötige Kommando `make` erkannt wird, wenn es in einem Terminal aufgerufen wird, muss dies noch in einer `.bat` Datei hinterlegt werden. Dazu muss eine Kommandozeilenumgebung geöffnet werden und folgendes Kommando eingegeben werden:

```
1 echo mingw32 -make %* > C:\VSARM\mingw\mingw32\bin\make.bat
```

Bei MinGW ist ebenfalls darauf zu achten, dass sich der Installationsordner in den Umgebungsvariablen des Systems befindet. Die Einstellungen dazu werden in einem späteren Schritt getätigt.

Des Weiteren wird CMake benötigt, um den Build-Prozess zu steuern und kann [hier](#)³ heruntergeladen werden. CMake muss nicht wie die Toolchain in dem separat angelegten Ordner installiert werden, sondern kann am vorgeschlagenen Ort oder anderswo installiert werden. Wichtig ist, dass bei den Installationsoptionen der Systempfad aller Benutzer mit CMake ergänzt wird (Abbildung (8)).

²<https://sourceforge.net/projects/mingw-w64/files/>

³<https://cmake.org/download/>

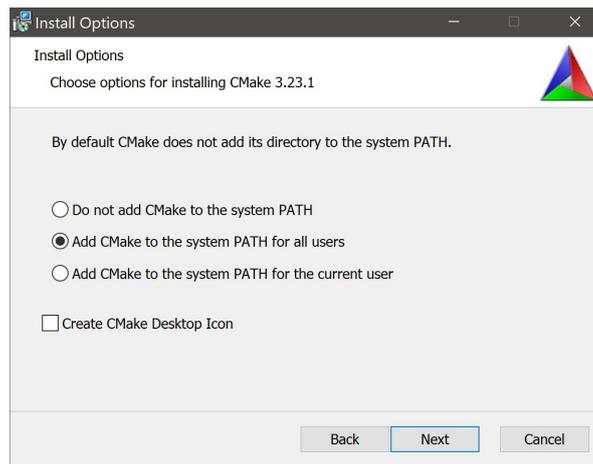


Abbildung 8: Pfadvariable zu CMake

Das Pico-SDK verwendet Python-Skripte, deshalb ist ebenfalls eine Installation von Python nötig. Raspberry Pi schlägt dazu die Version 3.9 von Python vor, die Version 3.10 funktioniert allerdings auch. Python kann von dieser [Website](#)⁴ heruntergeladen werden. Auch hier muss die Option ausgewählt werden, dass die Systemvariablen mit dem Pfad zu Python ergänzt werden (Abbildung (9)). Wo Python auf dem System installiert ist, spielt keine Rolle. Am Ende der Installation muss die Option `MAX_PATH length limit` ausgeschaltet werden.



Abbildung 9: Pfadvariable zu Python

Um das Pico-SDK herunterzuladen und zu installieren, wird Git verwendet, genauer gesagt die Git Bash Konsole. Git kann mit den Standardeinstellungen von [hier](#)⁵ installiert werden.

Nun kann die Git-Bash gestartet werden und über folgende Kommandos das Pico-SDK von GitHub heruntergeladen und initialisiert werden:

```

1 cd /c/VSARM/sdk/pico
2 git clone -b master https://github.com/raspberrypi/pico-sdk.git
3 cd pico-sdk
4 git submodule update --init
5 cd ..
6 git clone -b master https://github.com/raspberrypi/pico-examples.git

```

⁴<https://www.python.org/downloads/>

⁵<https://git-scm.com/download/win>

2.5.2 | Ergänzung und Überprüfung der Systemvariablen

Um alle installierten Programme und Anwendungen von überall im System aus zu verwenden, müssen die Systemvariablen ergänzt werden. Diese Variablen sind dem ganzen System bekannt und so können beispielsweise in einer Kommandozeilenumgebung Befehle, die in den Variablen hinterlegt sind, ausgeführt werden. Dies ist nötig, um später ein Projekt direkt von der IDE aus zu übersetzen. Bei der Installation von CMake oder Python wurde diese Variablen direkt vom Installer ergänzt. Um die Variablen zu konfigurieren und zu überprüfen, wird in der Suchleiste von Windows *env* eingegeben und dann unter *Einstellungen* die Option *Systemumgebungsvariablen bearbeiten* gewählt. Danach gelangt man über den Button *Umgebungsvariablen...* zu den Systemvariablen. Bei den benutzerspezifischen Variablen wird die bestehende Variable *Path* angewählt und diese bearbeitet. Es wird ein neuer Eintrag erstellt:

```
1 C:\VSARM\mingw\mingw32\bin
```

Zudem muss sichergestellt werden, dass folgender Eintrag bereits vorhanden ist, ansonsten muss dieser auch noch erstellt werden:

```
1 C:\VSARM\armcc<version>\bin
```

Nun kann durch einen Klick auf *OK* zurück zu den Umgebungsvariablen gelangt werden und es wird bei den Benutzervariablen einen neuen Eintrag zum Pico-SDK benötigt. Dazu wird eine Variable neu erstellt, was folgendermassen aussehen sollte:

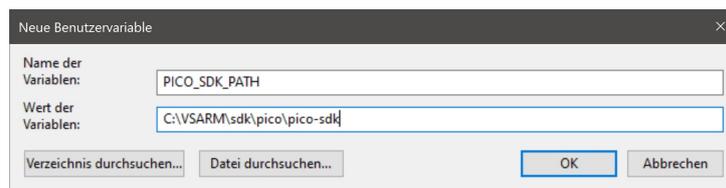


Abbildung 10: Pfadvariable zum SDK

Unter den Systemvariablen für alle Benutzer werden mit einem Klick auf *Path* und *Bearbeiten...* die vorhandenen Variablen überprüft. Folgende Einträge sollten vorhanden sein:

```
1 C:\Program Files\CMake\bin
2 c:\Program Files\Git\cmd
```

2.5.3 | Installation und Konfiguration von Visual Studio Code

Nun muss nur noch Visual Studio Code, die eigentliche IDE, von [hier](https://code.visualstudio.com/)⁶ installiert werden. Dazu können die Standardeinstellungen gewählt werden.

Es ist somit fast alles bereit, ein Projekt für den RP2040 zu erstellen. Damit dieses übersetzt und geflasht werden kann, muss VS Code noch konfiguriert werden.

VS Code ist an sich ein Tool, um textbasierte Dateien zu editieren, jedoch kann VS Code durch diverse Erweiterungen zu einer fast vollumfänglichen IDE ausgebaut werden. Unter *Extensions* kann *CMake Tools* installiert werden, was verwendet wird, um den ganzen Build-Prozess in VS Code zu tätigen. In *CMake Tools* gelangt man über das Zahnrad-Icon zu den *Extension*

⁶<https://code.visualstudio.com/>

Settings. Dort ist im Feld *CMake Generator* "MinGW Makefiles" einzutragen, um so die verwendete Toolchain in VS Code zu hinterlegen, dies ist in Abbildung (11) gezeigt. Zudem sollte ebenfalls im *Extensions* Menu die *C/C++* Erweiterung installiert werden, um Funktionen wie Code-Completion zu erhalten.

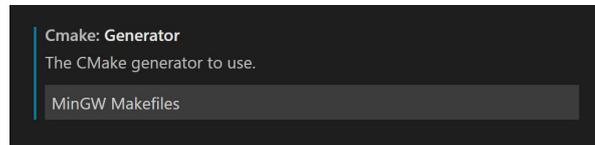


Abbildung 11: Hinterlegen der Toolchain in VS Code

Was alles zu einem minimalen RP2040-Projekt gehört, ist in der Abbildung (12) zu sehen. Dabei dienen die Dateien im Ordner `.vscode` dazu, das Projekt und den Debug-Prozess zu konfigurieren.

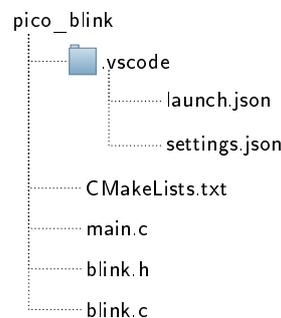


Abbildung 12: Minimale Projektstruktur

Wenn ein Projekt in VS Code geöffnet ist, kann in der blauen Leiste unten der Kit-Button angewählt werden, danach muss *Scan for Kits* angewählt werden. Nach kurzem Warten wird der Kit-Button erneut gedrückt und es wird das Build-Tool *GCC for arm-none-eabi <version>* ausgewählt. Nun kann in der blauen Leiste zuerst *CMake* angeklickt werden, sobald CMake sich fertig konfiguriert hat, kann der Button *Build* angewählt werden (Abbildung (13)). Nun sollte in dem Projekt ein neuer Ordner Namens *build* vorhanden sein, in dem sich das Programm in verschiedenen Dateiformaten befindet [26].

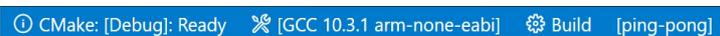


Abbildung 13: CMake-Konfiguration- und Build-Button

Jetzt lassen sich also RP2040-Projekte übersetzen. Nach erfolgreichem Kompilieren kann das Board mit gedrücktem Bootselect-Button direkt über USB an den PC angeschlossen werden. Das Board erscheint nun als Massenspeichergerät und man kann das `.uf2` File, das sich im Build-Ordner des Projektes befindet, auf das Board laden und das Programm wird ausgeführt.

Debuggen ist noch nicht möglich, dazu müssen nochmals verschiedene Schritte getätigt werden, was das Thema im folgenden Abschnitt sein wird.

2.5.4 | Installation der Debug-Tools

Es wird OpenOCD für das Debuggen des Boards verwendet. OpenOCD unterstützt jedoch den RP2040-Mikrocontroller nicht, weshalb Raspberry Pi eine eigene Version von OpenOCD veröffentlicht hat. Der Build dieser Version muss jedoch selbst gemacht werden, und dafür wird *Git for Windows SDK* verwendet. Dieses muss installiert werden und danach werden noch zusätzliche Packages benötigt, die sich ebenfalls über *Git for Windows SDK* herunterladen lassen.

Git for Windows SDK kann von dieser [Webseite](#)⁷ heruntergeladen und danach installiert werden. Es kann die neueste Version verwendet und mit allen Standardeinstellungen installiert werden.

Um verschiedene Packages zu installieren, muss `C:/git-sdk-64/git-bash.exe` ausgeführt werden. Es ist darauf zu achten, dass exakt dieses Programm ausgeführt wird und nicht die Git Bash, die im vorherigen Teil installiert wurde. Wichtig ist, dass das Terminal erst geschlossen wird, nachdem alles installiert ist. Wann das Terminal geschlossen werden kann, wird hier ausdrücklich erwähnt. Folgende Kommandos sind dabei einzugeben:

```
1 pacman -Syu
2 pacman -Su
3 pacman -S mingw-w64-x86_64-toolchain git make libtool pkg-config autoconf automake \
4 texinfo wget
```

Danach sind alle Meldungen, die ausgegeben werden, entweder mit `Enter` oder `Y + Enter` zu bestätigen. Nun werden noch weitere Packages benötigt, dazu sind folgende Kommandos auszuführen und wiederum mit `Y + Enter` zu bestätigen:

```
1 cd ~/Downloads
2 wget http://repo.msys2.org/mingw/x86_64/mingw-w64-x86_64-libusb-1.0.23-1-any.pkg.tar.xz
3 pacman -U mingw-w64-x86_64-libusb-1.0.23-1-any.pkg.tar.xz
```

Wenn dies alles eingerichtet ist, kann von GitHub die OpenOCD Version für den RP2040-Mikrocontroller nach `C:/VSARM/sdk/pico` heruntergeladen und übersetzt werden. Dazu werden im gleichen Terminal wie vorher die untenstehenden Kommandos eingegeben:

```
1 cd /c/VSARM/sdk/pico
2 git clone https://github.com/raspberrypi/openocd.git --branch picoprobe --depth=1
3 cd openocd
4 ./bootstrap
5 ./configure --enable-picoprobe --disable-werror
6 make
```

Jetzt wird OpenOCD übersetzt, was ca. 30 Minuten dauern sollte. Sobald der Vorgang abgeschlossen ist, muss die vorher heruntergeladene `libusb-1.0.dll` in das gleiche Verzeichnis wie die `openocd.exe` kopiert werden. Dies kann durch den folgenden Befehl erreicht werden:

```
1 cp /c/git-sdk-64/mingw64/bin/libusb-1.0.dll src/libusb-1.0.dll
```

Damit die Verbindung vom Host auf das Target über OpenOCD richtig funktioniert, hat Raspberry Pi das Programm *Picotool* entwickelt, was ebenfalls installiert werden muss, was über folgende Kommandos geschieht:

⁷<https://gitforwindows.org/#download-sdk>

```
1 cd /c/VSARM/sdk/pico
2 git clone -b master https://github.com/raspberrypi/picotool.git
3 cd picotool
4 mkdir build
5 cd build
6 cmake -G "MinGW Makefiles" \
7     -DPC_LIBUSB_INCLUDEDIR="/c/git-sdk-64/mingw64/include/libusb-1.0" ..
8 make
```

Wiederum wird eine Kopie von `libusb-1.0.dll` und `libgcc_s_seh-1.dll` in dem gleichen Ordner wie die gerade installierte `picotool.exe` benötigt. Dazu ist Folgendes in das Terminal einzugeben, dabei darf am Schluss der Punkt nicht vergessen werden:

```
1 cp /c/git-sdk-64/mingw64/bin/libusb-1.0.dll .
2 cp /c/git-sdk-64/mingw64/bin/libgcc_s_seh-1.dll .
```

Picotool kann folgendermassen getestet werden, dabei sollte es eine Hilfenachricht ausgeben und dann wieder terminieren:

```
1 ./picotool.exe
```

Wenn nun alles funktioniert, kann das Terminal geschlossen werden und *Git for Windows SDK* durch Löschen des Ordners `C:/git-sdk-64` wieder deinstalliert werden.

Normalerweise wird zum Debuggen von Mikrocontrollern eine Debug-Probe verwendet, die eine Verbindung zwischen Host-PC und Target darstellt. Raspberry Pi hat dazu das Programm Picoprobe entwickelt, das sich auf dem Raspberry Pi Pico ausführen lässt, wobei das Pico das SWD-Protokoll (Single Wire Debug) emuliert. Somit kann ein Raspberry Pi Pico als Debugger verwendet werden, um ein anderes Board mit dem RP2040-Mikrocontroller zu debuggen. Um Picoprobe zu übersetzen muss im Dateisystem an den Ort navigiert werden, wo das Programm Picoprobe installiert werden soll. Über Rechtsklick kann von dort aus die Git Bash gestartet werden und über folgende Kommandos Picoprobe heruntergeladen und übersetzt werden:

```
1 git clone https://github.com/raspberrypi/picoprobe.git
2 cd picoprobe
3 mkdir build
4 cd build
5 cmake -G "MinGW Makefiles" ..
6 make
```

Nun kann im Build-Ordner die Datei `picoprobe.uf2` wie oben beschrieben auf das Raspberry Pi Pico, das als Debug-Probe verwendet wird, geflasht werden. Damit OpenOCD mit der Debug-Probe kommunizieren kann, muss noch ein USB-Treiber installiert werden. Um dies zu machen, muss Zadig⁸ installiert werden. Danach wird Zadig geöffnet und unter *Options* wird *List All Devices* und danach *Picoprobe (Interface 2)* ausgewählt. Dazu muss die Debug-Probe mit dem PC verbunden sein. Nun kann bei den Treibern `libusb-win32` ausgewählt und installiert werden. Jetzt sollte der neue Treiber als `libusb0` erscheinen.

⁸<https://zadig.akeo.ie/>

2.5.5 | Debug-Konfiguration von Visual Studio Code

Als nächsten und letzten Schritt muss nun in VS Code unter *Extensions* die *Cortex-Debug* Erweiterung installiert werden. Jetzt können die beiden Boards, Picoprobe und Target, wie in Abbildung (14) dargestellt, miteinander verbunden werden. Dabei erfolgt die Spannungsversorgung des Targets über das schwarze und rote Kabel, über das Violette und Weisse läuft das SWD-Protokoll und das orange und gelbe Kabel stellen die UART-Verbindung vom Target auf die Debug-Probe dar. Um die Kommandos von OpenOCD an die Debug-Probe zu leiten und die Printausgaben vom Target an den PC, muss die Debug-Probe mit einem USB-Kabel an den PC angeschlossen werden [27].

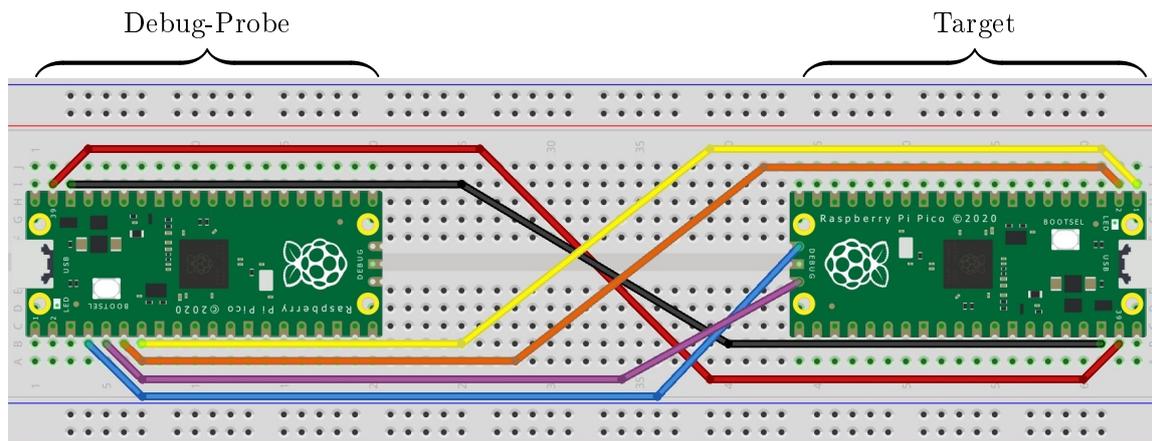


Abbildung 14: Verdrahten der Debug-Probe und des Targets

Zum Schluss kann in VS Code im *Run and Debug* Menu der Debugger gestartet werden, der das Übersetzten und Flashen des Projektes initialisiert und in `main.c` anhält (Abbildung (15)).

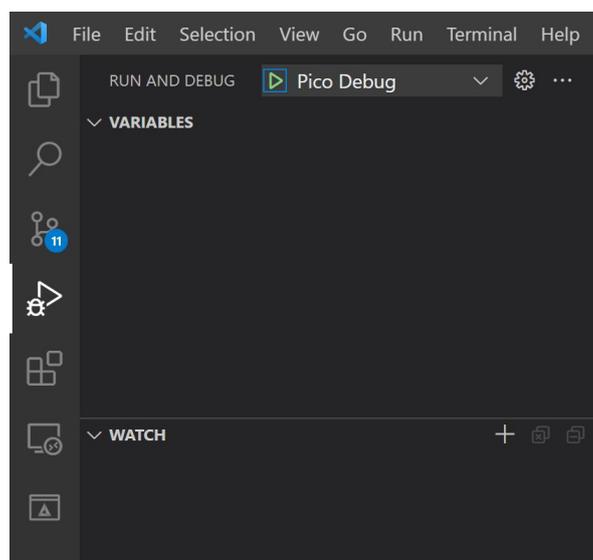


Abbildung 15: Starten der Debug-Session

3 | Hardware

In diesem Kapitel wird auf die verwendeten Bauteile, das Zusammenführen dieser im Schema und auf das Hardwaredesign eingegangen. Als Übersicht über die Hardware dient das in Abbildung (16) zu sehende Blockschaltbild.

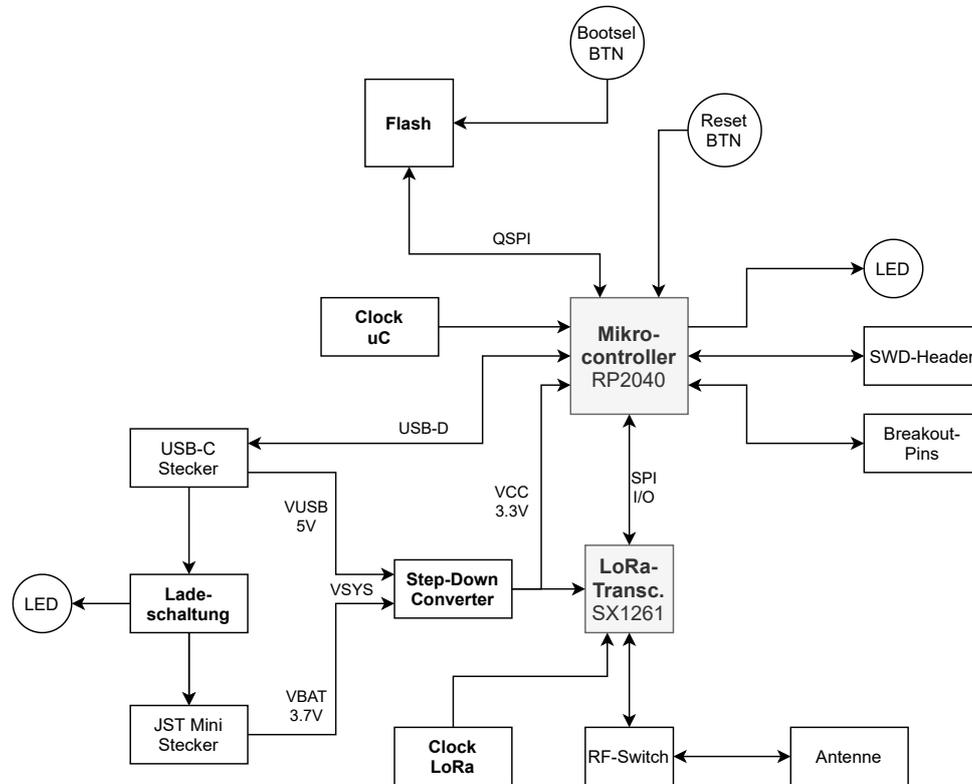


Abbildung 16: Blockschaltbild der Hardware des tinyLoRa-Boards

3.1 | Komponenten

3.1.1 | Mikrocontroller: RP2040

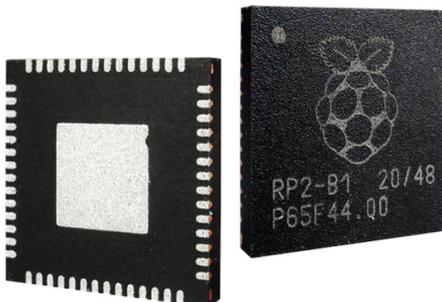


Abbildung 17: RP2040

(Quelle: Mouser Electronics, 2022 [8])

Der RP2040 Mikrocontroller von Raspberry Pi wurde in Kapitel (2.3) bereits kurz vorgestellt. Auf dem tinyLoRa-Board hat er die Aufgabe, die gesamte Kommunikation mit dem LoRa-Transceiver zu führen. Auf dem Mikrocontroller läuft der LoRa Kommunikationsstack, sowie die Benutzeranwendung, die für verschiedenste Aufgaben individuell gestaltet werden kann.

3.1.2 | Flash: W25Q128JVSIM

Der Mikrocontroller hat keinen Flash-Speicher auf dem Chip und deshalb muss ein externer Flash verwendet werden. Dazu wurde der W25Q128JVSIM von dem Hersteller Winbond gewählt. Dieser hat mit 16 *Mbyte* eine sehr grosse Speicherkapazität. Angesteuert wird der Flash mit dem QSPI-Protokoll, für diese Signale sind auf dem Mikrocontroller explizit Pins angebracht (S. 5, [30]). Diese Familie von Flash-Bausteinen ist für die Benutzung mit dem RP2040 von Raspberry Pi empfohlen (S. 8, [14]). Durch die grosse Speicherkapazität und der garantierten Kompatibilität mit dem RP2040 ist dieser Flash-Speicher eine geeignete Wahl.

3.1.3 | Spannungsregler: AP2112M-3.3TRG1

Der Spannungsregler AP2112M-3.3TRG1 sorgt dafür, dass alle Komponenten auf dem Board mit Spannung versorgt werden. Dieser hat eine fixe Ausgangsspannung von 3.3 V und einen maximalen Ausgangsstrom von 600 mA, was einerseits reicht, alle Komponenten auf dem Board mit Strom zu versorgen, aber auch externe Komponenten über die GPIO-Pins. Die maximale Eingangsspannung beträgt 6.5 V und ab einer Spannung am Eingang von 3.4V wird der Ausgang auf 3.3 V geregelt [4]. Diese Low-Dropout-Voltage ist wichtig, denn die Akkuzelle, die das tinyLoRa versorgt, kann bis zu etwa 3.3 V entladen werden. Somit lässt sich die LiPo-Zelle möglichst gut ausnutzen.

3.1.4 | Batterie-Lademanagement: MCP73821-5ACI/MC

Da das tinyLoRa auch batteriebetrieben sein soll, wird direkt auf das Board eine Ladeschaltung integriert. Dies hat den Vorteil, dass sich eine angeschlossene LiPo-Zelle laden kann, wenn das Gerät durch eine USB-Spannung gespiesen wird. Dafür wird der Lademanagement-Baustein MCP73821-5ACI/MC von Microchip Technologie verwendet. Sobald die USB-Spannung am Eingang anliegt, wird die angeschlossene LiPo-Zelle aufgeladen, was durch eine LED auf dem Board angezeigt wird. Der verwendete Baustein lädt die Zelle auf eine maximale Spannung von 4.5 V auf und durch einen externen Widerstand wird der Ladestrom auf knapp 200 mA eingestellt [7].

3.1.5 | USB-C Buchse: USB4110-GF-A



Um Programme auf den RP2040 zu flashen oder die Akkuzelle zu laden, wird ein USB-Anschluss benötigt. Durch die immer grössere Verbreitung wird beim tinyLoRa dafür eine USB-C Buchse verwendet, genauer die USB4110-GF-A Buchse vom Hersteller GCT. Es handelt sich dabei um eine SMD Variante.

Abbildung 18: USB4110-GF-A
(Quelle: Mouser Electronics, 2022 [10])

3.1.6 | LoRa-Transceiver: SX1261

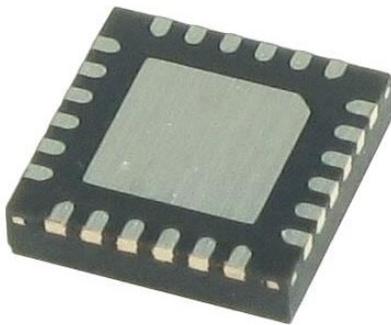


Abbildung 19: SX1261
(Quelle: Mouser Electroics, 2022 [9])

Der LoRa-Transceiver SX1261 von Semtech ist für das Senden und Empfangen der LoRa Signale zuständig. Er empfängt die LoRa modulierten Signale mithilfe einer Antenne, wandelt diese in Binärdaten um und speichert sie anschliessend. Die Daten können danach vom Mikrocontroller per SPI aus dem Buffer gelesen werden. Auf der anderen Seite nimmt der SX1261 Nachrichten vom Mikrocontroller zum Senden entgegen, moduliert diese und sendet sie wiederum per Antenne an die naheliegenden Gateways. Der SX1261 ist für energiesparende Anwendungen konzipiert, jedoch kann der aufgenommene Strom beim Senden durchaus um die 25 mA betragen. Beim Empfangen ist dieser mit maximal 9 mA deutlich kleiner [19].

3.1.7 | HF-Schalter: AS213-92LF

Da der SX1261 senden und empfangen kann, muss der Zugang zur Antenne jeweils zwischen dem Sender- und Empfängerkanal des Transceivers gesteuert werden. Dafür wird der HF-Schalter AS213-92LF von Skyworks Solution verwendet. Durch jeweils zwei I/O-Signale kann der Antennenanschluss zwischen Senden und Empfangen umgeschaltet werden [28].

3.1.8 | Antenne: X9000984-4GDSMB



Abbildung 20: X9000984-4GDSMB
(Quelle: Mouser Electroics, 2022 [11])

Die Antenne wird benötigt, um eine möglichst grosse Empfangs- und Sendeleistung zu erreichen. Die eingesetzte Antenne X9000984-4GDSMB von ethertronics ist für den verwendeten Frequenzbereich von 868 MHz konzipiert. Durch dem SMA-Anschluss lässt sie sich leicht an das tinyLoRa-Board per Gewinde befestigen [5].

3.1.9 | Clock

Jeweils für den Mikrocontroller und LoRa-Transceiver wird ein separater Clock verwendet. Für den Mikrocontroller wird dafür ein 12 MHz Clock verwendet, für den LoRa-Transceiver einer mit einer Frequenz von 32 MHz . Dabei wurde sich an die empfohlenen Angaben der Hersteller der Komponenten gehalten und anhand von diesen die Clocks ausgewählt.

3.1.10 | LED



Abbildung 21: LED

(Quelle: Digi-Key Electronics, 2022 [3])

Auf dem Board sind zwei LEDs angebracht. Eine rote LED ist zum Signalisieren, ob die LiPo-Zelle aktuell geladen wird, eine grüne LED ist für Benutzerzwecke bestimmt und kann über einen GPIO-Pin des Mikrocontrollers angesteuert werden. Der Strom durch die LEDs ist durch die Vorwiderstände auf etwa 5 mA begrenzt, um den Energieverbrauch gering zu halten.

3.1.11 | Button



Abbildung 22: Button

(Quelle: Digi-Key Electronics, 2022 [2])

Es befinden sich auf dem tinyLoRa zwei Knöpfe. Einer ist der Reset-Button, mit dem der Mikrocontroller neu gestartet werden kann, der andere ist der Bootselect-Button, welcher gebraucht wird, um von einem PC neue Programme auf den Mikrocontroller zu flashen. Mit der kleinen Bauweise des in Abbildung (22) zu sehendem Drucktaster nimmt dieser auf dem PCB sehr wenig Platz ein.

3.2 | Schema

Die im vorherigen Unterkapitel vorgestellten Komponenten müssen nun mit diversen passiven Elementen zu einem funktionierenden System zusammengefügt werden. Dazu wurden die Anweisungen in den jeweiligen Datenblättern und Manuals befolgt. Das Schema wurde in vier Teile aufgeteilt, welche in den nächsten Unterkapiteln beschrieben werden. Diese sind die ganze Spannungsversorgung (Abbildung (23)), der Mikrocontroller und Flash (Abbildung (24)), der LoRa-Transceiver mit Antennenanpassungsnetzwerk (Abbildung (26)), sowie die I/O-Header (Abbildung (27)). Das gesamte Schema findet sich im Anhang unter (B).

3.2.1 | Spannungsversorgung

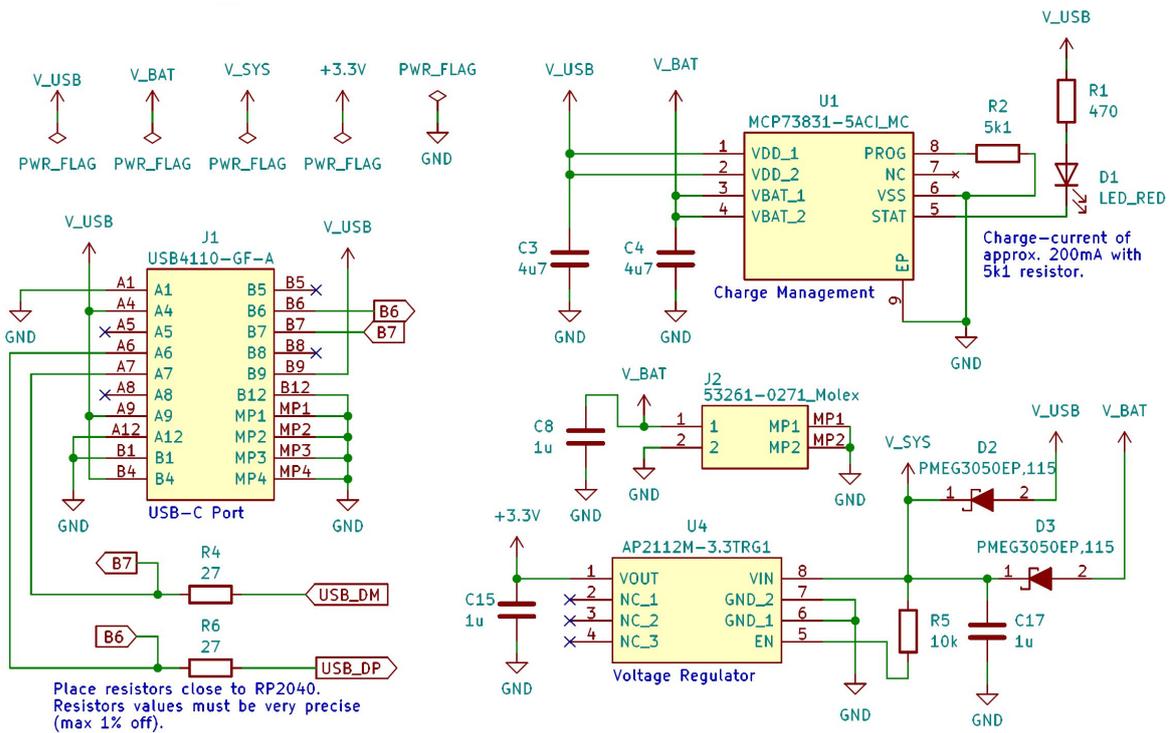


Abbildung 23: Schema: USB-Anschlüsse und Energieversorgung

Das Board kann wie schon erwähnt durch zwei Energiequellen versorgt werden, per USB über die Buchse J1 oder Batterie über den Stecker J2. Wenn eine USB-Spannungsversorgung anliegt, sollen alle Bauteile per USB gespeist werden und die Batterie geladen werden. Um dies zu erreichen, dienen die zwei Dioden D2 und D3, welche die Systemspannung an den Spannungsregler U4 führen. Somit liegt am Eingang des Spannungsreglers immer die höhere der beiden Spannungen an, also die USB-Spannung, wenn diese vorhanden ist. Um den Spannungsabfall gering zu halten, wurden Schottky-Dioden verwendet. Dies ist wichtig, da so die Akkuzelle über einen möglichst grossen Spannungsbereich ausgenutzt wird. Die Batteriespannung muss mindestens 3.6 V betragen, um das Board sauber speisen zu können, da die minimale Eingangsspannung des Spannungsreglers 3.4 V beträgt und an der Diode ca. 0.2 V abfällt [12]. U1 ist das Lade-Management, welcher bei anliegender USB-Spannung die LiPo-Zelle über den Stecker J2 lädt. Durch den Widerstand R2 wird der Ladestrom nach Gleichung (1) aus dem Datenblatt auf knapp 200 mA eingestellt und die LED D1 zeigt an, ob die Batterie momentan geladen wird.

$$i_{ch} = \frac{U_{ref}}{R2} = \frac{1 V}{5.1 k\Omega} = 196 mA \quad (1)$$

Bei der USB-Buchse J1 sind noch zwei Abschlusswiderstände an den Datenleitungen zu sehen, welche im Design sehr nahe an dem Mikrocontroller platziert werden.

3.2.2 | Mikrocontroller und Flash

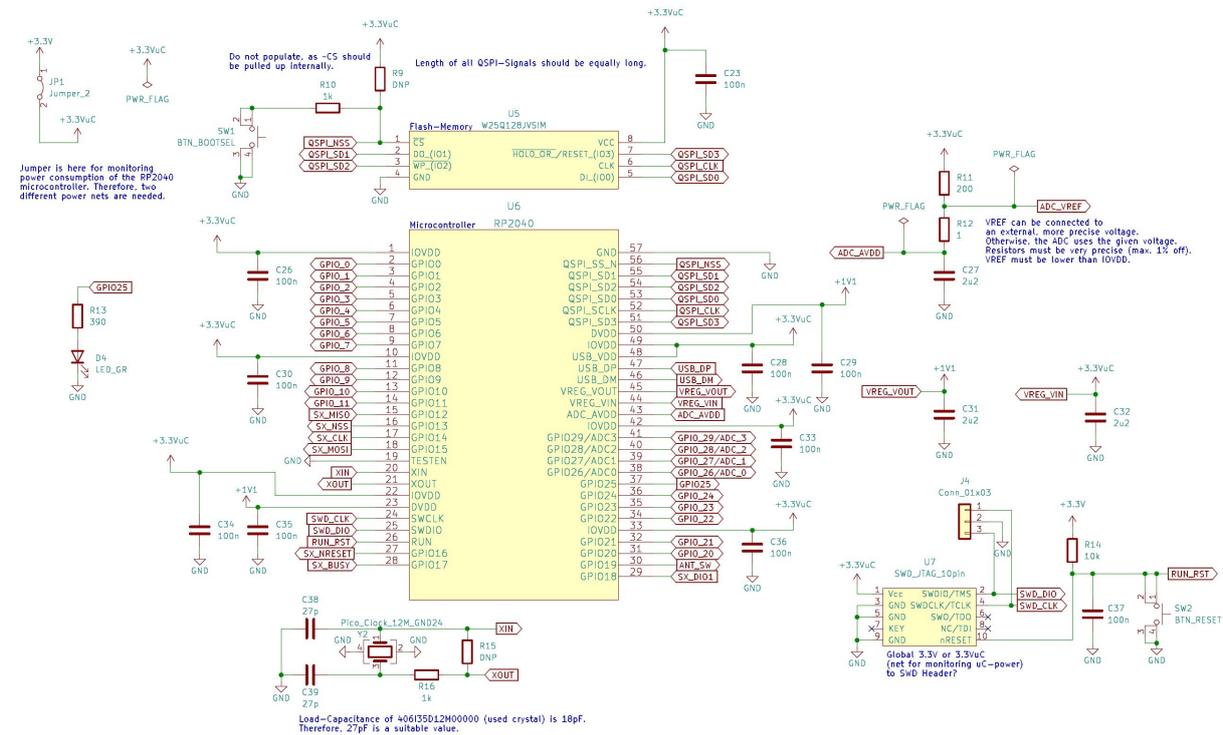


Abbildung 24: Schema: Mikrocontroller und Flash

In diesem Teil des Schemas ist in der Mitte der Mikrocontroller U6 zu sehen, gerade oberhalb befindet sich der Flash-Speicher U5. Diese Komponenten kommunizieren über die angeschlossenen QSPI-Signale. An dem Flash ist zusätzlich der Bootselect-Button SW1 angeschlossen, der das invertierte Enable-Signal vom Flash auf Low zieht und so kann dem Mikrocontroller signalisiert werden, dass der ROM-Bootloader ausgeführt werden soll, um ein neues Programm zu flashen (S. 157, [17]). Der Pullup-Widerstand R9 sollte nicht benötigt werden, da das Signal am \overline{CS} Pin vom Flash intern auf die Versorgungsspannung gezogen werden sollte (S. 8, [14]). Dieser wird deshalb nicht bestückt, ist aber vorhanden falls er trotzdem benötigt wird. Es sind diverse Stützkondensatoren an dem Mikrocontroller und Flash angebracht, die Kapazität und Platzierung dieser wurde aus den Datenblättern entnommen. Der Jumper JP1 dient zur Messung der aufgenommenen Energie des Mikrocontrollers und Flash. Der Stromfluss vom Spannungsregler führt ausschliesslich über JP1 zum Mikrocontroller und Flash. Zu Messzwecken kann also der Jumper entfernt und ein Messgerät angeschlossen werden. Die User-Led D4 ist fix an den GPIO25 Pin angeschlossen, dies wurde vom Raspberry Pi Pico übernommen [18]. J4 und U7 sind Debug-Header, die ebenfalls mit dem Reset-Signal des Mikrocontrollers und somit auch mit dem Reset-Button SW2 verbunden sind. Dies dient dazu, dass der Mikrocontroller vom Debugger neugestartet werden kann. Einer der Debug-Header ist der standardisierte JTAG-Header, der andere ist derselbe, wie auf dem Raspberry Pi Pico verbaut ist. So ist der Vorteil vorhanden, dass beide verwendet werden können. Im untersten Teil des Schemas ist der Oszillator Y2 zu sehen. Dieser hat eine Ladekapazität von $18pF$, somit ergeben sich die Kapazitäten von C38 und C39 zu $27 pF$ nach untenstehenden Formeln (2) und (3) (S. 1, [1]).

$$C_{Load} = \frac{C_{Clk}}{2} + C_{Par} \quad (2)$$

$$C_{Clk} = 2(C_{Load} - C_{Par}) = 2(18 \text{ pF} - 4 \text{ pF}) = 28 \text{ pF} \quad (3)$$

Viele der GPIO-Pins des Mikrocontrollers werden auf den Breakout-Header geführt. Es wird über SPI- und I/O-Signalen mit dem LoRa-Transceiver kommuniziert, was total acht GPIO-Pins beansprucht (Abbildung (25)).

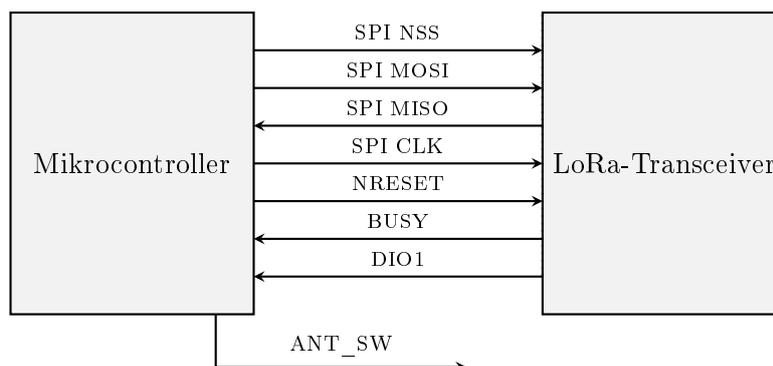


Abbildung 25: Verbindungen vom Mikrocontroller und LoRa-Transceiver

Über `SX_NRESET` kann der LoRa-Transceiver vom Mikrocontroller aus neu gestartet werden. `SX_BUSY` zeigt dem Mikrocontroller an, ob der Transceiver gerade beschäftigt ist, beispielsweise mit dem Empfangen von Daten. `SX_DIO1` ist ein weiteres I/O-Signal, das als Interrupt Anzeige vom SX1261 verwendet wird und der Kommunikation zwischen den beiden Chips dient. Mit `ANT_SW` wird der HF-Schalter angesteuert. Alle diese Signale werden in der Software vom Kommunikationsstack benötigt (S. 104, [19]). Im rechten oberen Teil befindet sich die Spannungsaufbereitung der Versorgungsspannung für die Analog-Digital-Converter, welche ebenfalls vom Raspberry Pi Pico übernommen wurde.

3.2.3 | LoRa-Transceiver, Anpassungsnetzwerk und Antenne

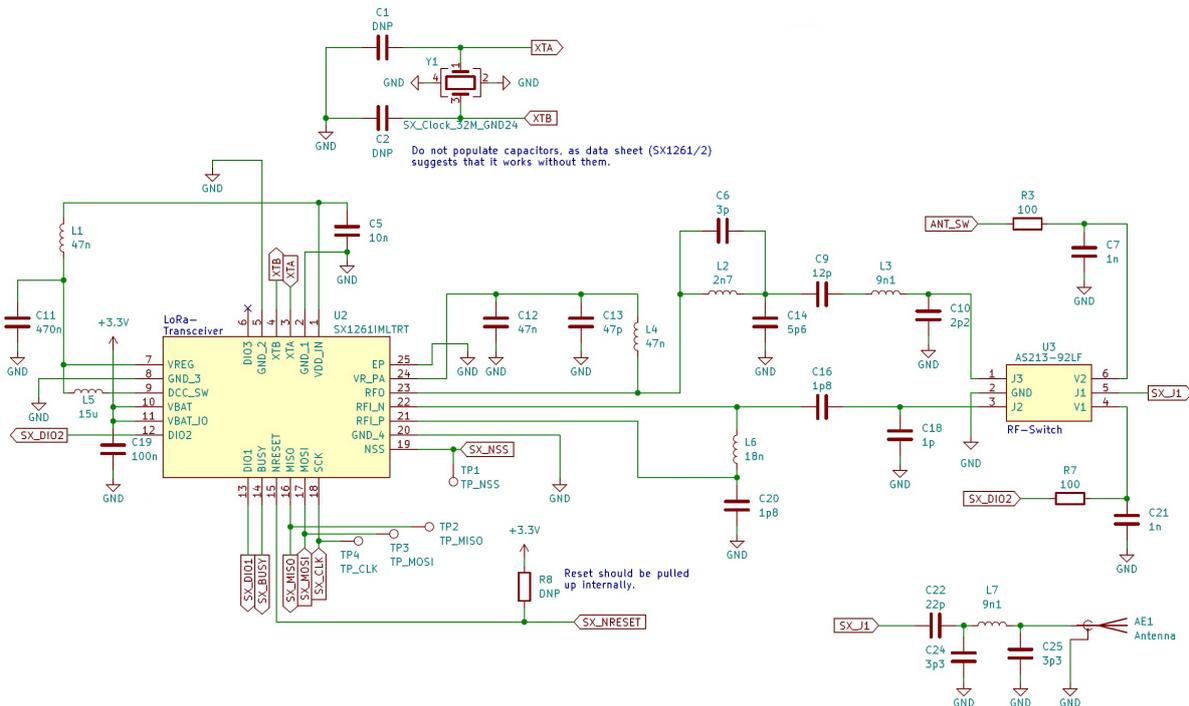


Abbildung 26: Schema: LoRa-Transceiver, Impedanzanpassung und Antenne

Das Schema rund um den LoRa-Transceiver U2 wurde praktisch identisch aus den Referenzdesigns von Semtech übernommen (S. 104, [19]). So lässt sich eine optimale Schaltung garantieren. Welche Signale vom LoRa-Transceiver auf den Mikrocontroller geführt werden, wurde bereits erläutert. Die Spannungsversorgung erfolgt durch die Pins 10 und 11, VBAT und VBAT_IO. An den Pins VR_PA und RFO liegt das Sendeanpassungsnetzwerk an, an den Pins RFI_N und RFI_P das differentielle Empfängernetzwerk (S. 12, [19]). Die Netzwerke beinhalten die Impedanzanpassung der Antenne und verschiedene Filter, was vor allem beim Sendepfad zum Tragen kommt. Dabei dienen L2 und C14 der Impedanzanpassung des RFO-Ausgangs des LoRa-Transceivers. L2 und C6 dienen als Filter der 2. harmonischen Oberwellen und L3, C9, C10 und C14 filtern die höheren Oberwellen aus dem Sendepfad (S. 17, [21]). Die Dimensionierung dieser Bauteile findet sich in Kapitel (5.2). Der HF-Schalter U3 wird durch zwei Signale, eines vom Mikrocontroller und eines vom LoRa-Transceiver angesteuert und verwaltet den Zugriff auf die Antenne AE1. An jedem Eingang benötigt der HF-Schalter einen Kondensator, damit nur die Signale ohne DC-Anteil an dem Schalter anliegen [28]. Vor der Antenne werden die Signale nochmals durch verschiedene passive Bauelemente gefiltert. Diese sind vor allem wichtig, falls die Antenne stark verstimmt ist. Dies sollte aber bei der hier verwendeten Antenne nicht der Fall sein, da diese für den Frequenzbereich von 868 MHz ausgelegt ist [5]. Ganz oben im Schema ist der Oszillator für den LoRa-Transceiver zu sehen, die beiden Kondensatoren dazu sind als *Do Not Fit* (DNF) markiert, da laut Datenblatt des Transceivers diese nicht nötig sind (S. 104, [19]). Falls diese aber trotzdem benötigt werden sollten, wären diese vorhanden. Für eine allfällige Fehlersuche oder um Beobachtungen durchführen zu können, sind die vier SPI-Leitungen noch mit Testpunkten versehen. R8 dient als Pullup-Widerstand, da der Reset-Pin vom Transceiver

ein Active-Low-Signal ist. Dieser ist nicht bestückt, da die Reset-Leitung vom Transceiver intern auf die Versorgungsspannung gezogen werden sollte.

3.2.4 | Breakout-Header

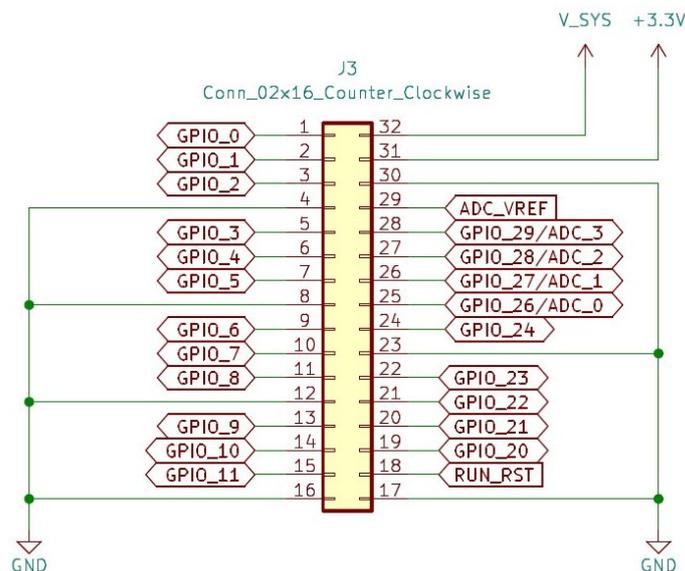


Abbildung 27: Schema: Breakout-Pins

Es existieren 32 Breakout-Pins, die steckbrettcompatibel sind und sich am Rand des Boards befinden. An sieben dieser Pins liegt das GND-Netz. 17 Pins sind direkt auf GPIO-Pins des Mikrocontrollers geführt und können vom Benutzer für verschiedene Zwecke verwendet werden. Des Weiteren existieren vier Pins, die zusätzlich zu den GPIO-Funktionen als analoge Eingänge dienen. An Pin 29 kann eine Referenzspannung für die Analog-Digital-Converter angelegt werden (S. 180, [17]). Durch Pin 18 kann der Mikrocontroller neu gestartet werden, dieser Pin muss dafür auf das Low-Level gezogen werden und ist direkt mit dem Reset-Button und dem Reset-Pin des Mikrocontrollers verbunden. Auf Pin 31 ist die Spannungsversorgung des Boards, also 3.3 V, vorhanden und auf Pin 32 die Systemspannung, also entweder die USB-Spannung von 5 V oder die Batteriespannung von 3.7 V. Das Pinout-Diagramm vom tinyLoRa findet sich im Anhang unter Kapitel (D) .

3.3 | PCB

In diesem Unterkapitel wird auf das Design des PCBs eingegangen. Ähnlich wie beim Schema kann das PCB geometrisch grob in verschiedene Sektoren eingeteilt werden. Diese sind die Spannungsversorgung, der Mikrocontroller mit Flash und Breakout-Pins und der LoRa-Transceiver mit Anpassungsnetzwerk und Antenne. In den untenstehenden Abbildungen (28) und (29) ist das PCB mit den verschiedenen Teilen zu sehen, welche in den nächsten Unterkapiteln genauer beschrieben werden.

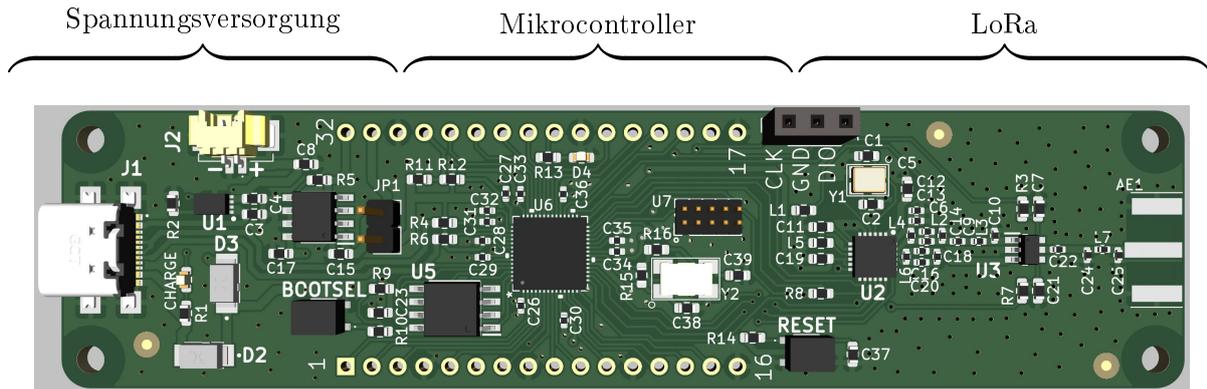


Abbildung 28: 3D-Ansicht der Front des PCBs

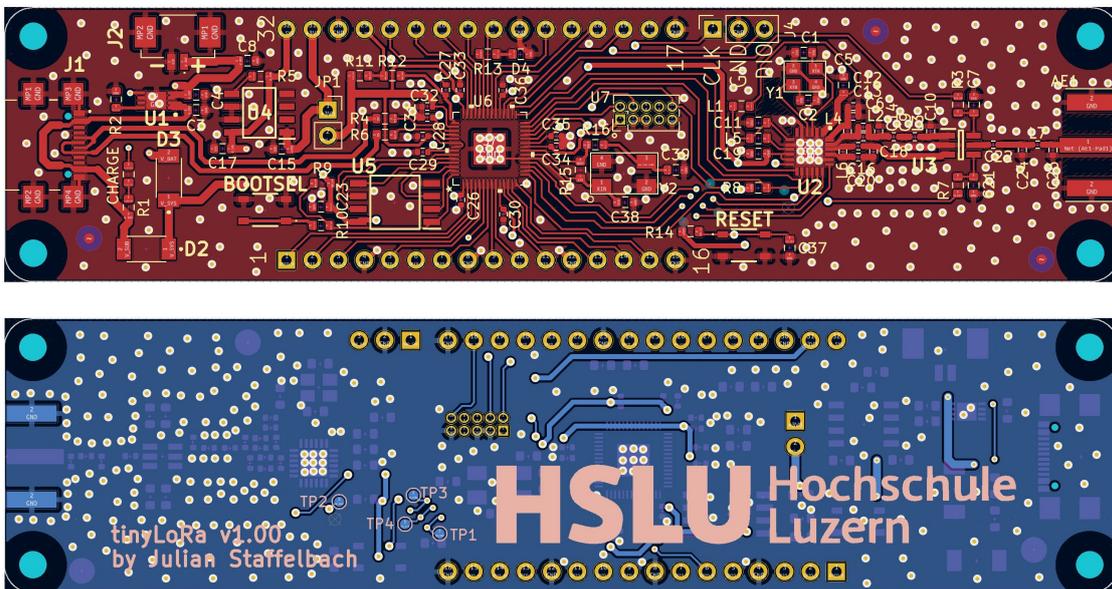


Abbildung 29: Frontansicht und gespiegelte Rückansicht des PCBs

Der Print hat eine Grösse von $109 \times 27 \text{ mm}$ und eine Dicke von 1 mm . Es wurden in den Ecken vier Befestigungslöcher mit einem Durchmesser von 2.7 mm angebracht, für die M2.5 Schrauben angedacht sind. Die USB-Buchse ist bündig mit dem Rand des PCBs. Bei den meisten Komponenten handelt es sich um SMD-Bauteile, lediglich die Breakout-Header, die Debug-Header, sowie der Jumper für die Messung des Energieverbrauchs sind Trough-Hole-Bauteile. Zudem sind alle Komponenten auf der Oberseite des PCBs angebracht. Es wurde versucht, möglichst alle Lei-

bei den SPI-Verbindungen entstand aufgrund der Pinbelegung vom Mikrocontroller und Transceiver eine Kreuzung. Auf der Ober- und Unterseite des Mikrocontrollers sind hauptsächlich GPIO-Pins, die direkt auf die Pin-Header führen, aber auch dort sind je zwei Pins pro Seite für die Spannungsversorgung vorhanden. Zudem befindet sich oberhalb des Mikrocontrollers die User-LED.

3.3.3 | LoRa-Transceiver, Anpassungsnetzwerk und Antenne

Im rechten Teil des Boards befindet sich der LoRa-Transceiver SX1261 von Semtech, das Anpassungsnetzwerk, der RF-Switch und die Antennenbuchse. Der SX1261-Chip verfügt ebenfalls über einen eigenen Clock Y1, es sind Pins für die Spannungsversorgung und SPI-Kommunikation angebracht, sowie Pins für das Empfangen und Senden der Daten. Der Clock befindet sich oberhalb des Transceivers. Zwischen diesen beiden Bauteilen wurde Kupfer auf dem PCB entfernt, um thermische Einflüsse vom Transceiver auf den Clock zu minimieren. Dies wird von Semtech vorgeschlagen, um die Clock Frequenz so stabil wie möglich zu halten (S. 11, [20]). Auf der unteren Seite befinden sich alle Signale, die auf den Mikrocontroller führen. Rechts vom Transceiver ist das Impedanzanpassungsnetzwerk der Antenne platziert. Die Geometrie dieses Netzwerkes wurde ebenfalls von Semtech übernommen, um möglichst gute Antenneneigenschaften zu erreichen. Das Anpassungsnetzwerk ist über den RF-Switch mit der Antennenbuchse verbunden, an der eine Antenne mit SMA-Anschluss aufgeschraubt werden kann.

3.3.4 | Inbetriebnahme und Korrekturen des PCBs

Das PCB wurde bestückt und getestet. Die Inbetriebnahme lief wie folgt ab:

1. Nach dem Bestücken und Löten folgte die optische Kontrolle. Falls Unreinheiten oder ungewollte Verbindungen an Pins ausgemacht werden konnten, wurde dies behoben.
2. Es wurde mit einem Multimeter gemessen, ob alle Komponenten mit deren Spannungsversorgung verbunden sind. Zudem wurde überprüft, dass sich der Widerstand zwischen allen Spannungs- und Ground-Anschlüssen mindestens im zweistelligen Kiloohmbereich befindet.
3. Über den Batteriestecker wurde das Board von einem Netzteil mit Strombegrenzung gespeist. Es wurde mit einem Multimeter gemessen, ob alle nötigen Spannungspegel korrekt vorhanden waren. So konnte sichergestellt werden, dass der Spannungsregler funktioniert und alle Spannungspfade korrekt sind.
4. Das Board wurde per USB gespiesen, zuerst mit einem Netzteil, danach von einem PC. Sobald dies funktioniert hat, wurde eine Demoanwendung auf das tinyLoRa geladen und so konnte die korrekte Funktion von Mikrocontroller und allen GPIO-Pins nachgewiesen werden.
5. Es wurde eine LiPo-Zelle angeschlossen und diese per USB geladen. Die Ladeschaltung funktioniert und die Batterie wurde bis zu einem Spannungspegel von 4.5 V geladen.

Im Abschnitt (2.5.5) ist gezeigt, wie sich mit einem Raspberry Pi Pico ein anderes Pico debuggen lässt. Das gleiche Verfahren lässt sich auf jedes Board mit dem RP2040-Chip und dem entsprechenden Debug-Header anwenden. Abbildung (31) zeigt, wie das Debuggen des tinyLoRa mit einem Raspberry Pi Pico als Debug-Probe möglich ist.

4 | Software

Dieses Kapitel befasst sich mit der Software, die auf dem tinyLoRa und somit auf dem RP2040 läuft. Dabei wird zuerst darauf eingegangen, wie die Software aufgebaut ist und wie die Build-Konfiguration mit CMake aufgesetzt ist. Danach wird das SDK vorgestellt und der Hardware-Layer des Kommunikationsstacks vorgestellt.

4.1 | Struktur der Software

Grosse Teile der verwendeten Software sind bereits von Semtech zur Verfügung gestellt. Auf GitHub findet sich unter dem Benutzer *LoRa-net* das Repository *LoRaMac-node*, welches als Ausgangslage dieses Projektes verwendet wird. Dies bedeutet, dass grosse Teile der Software übernommen werden können. Das Projekt weist folgende Ordnerstruktur auf:

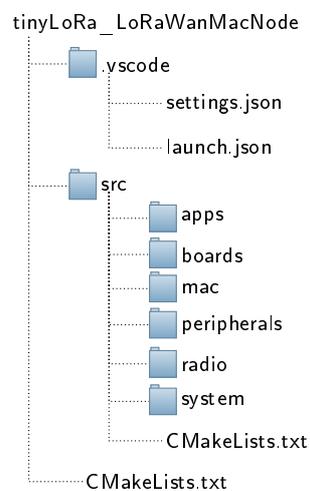


Abbildung 33: Projektordner tinyLoRa

Die plattformabhängigen Teile sind im Ordner `boards` abstrahiert. Es ist klar definiert, dass die Module in diesem Ordner zu ändern, bzw. zu ergänzen sind, wenn der Stack auf eine neue Architektur portiert werden soll, wie dies in diesem Projekt der Fall ist.

Die Software ist so aufgebaut, dass über verschiedene Präprozessoreinstellungen in der Datei `settings.json` das Projekt konfiguriert werden kann. Beispielsweise kann die Region und somit auch die Trägerfrequenz, welche für Europa 868 MHz beträgt, eingestellt werden. Für Nordamerika beträgt diese Frequenz beispielsweise 915 MHz . Des Weiteren sind verschiedene Applikationen vorhanden. Welche davon übersetzt werden soll, lässt sich ebenfalls einstellen, was als Beispiel in dem Codeabschnitt (1) dargestellt ist.

```

1 // Determines the application. You can choose between:
2 // LoRaMac (Default), ping-pong
3 "APPLICATION": "LoRaMac",
4 //"APPLICATION": "ping-pong",

```

Codesequenz 1: Auswählen der Applikation in settings.json

In den Unterordnern des Verzeichnisses `src` sind alle Source- und Headerdateien, sowie weitere CMake-Dateien abgelegt. Dies ist in untenstehender Abbildung (34) dargestellt, dabei wurden der Übersicht halber jeweils die Dateien in einem Ordner zusammengefasst.

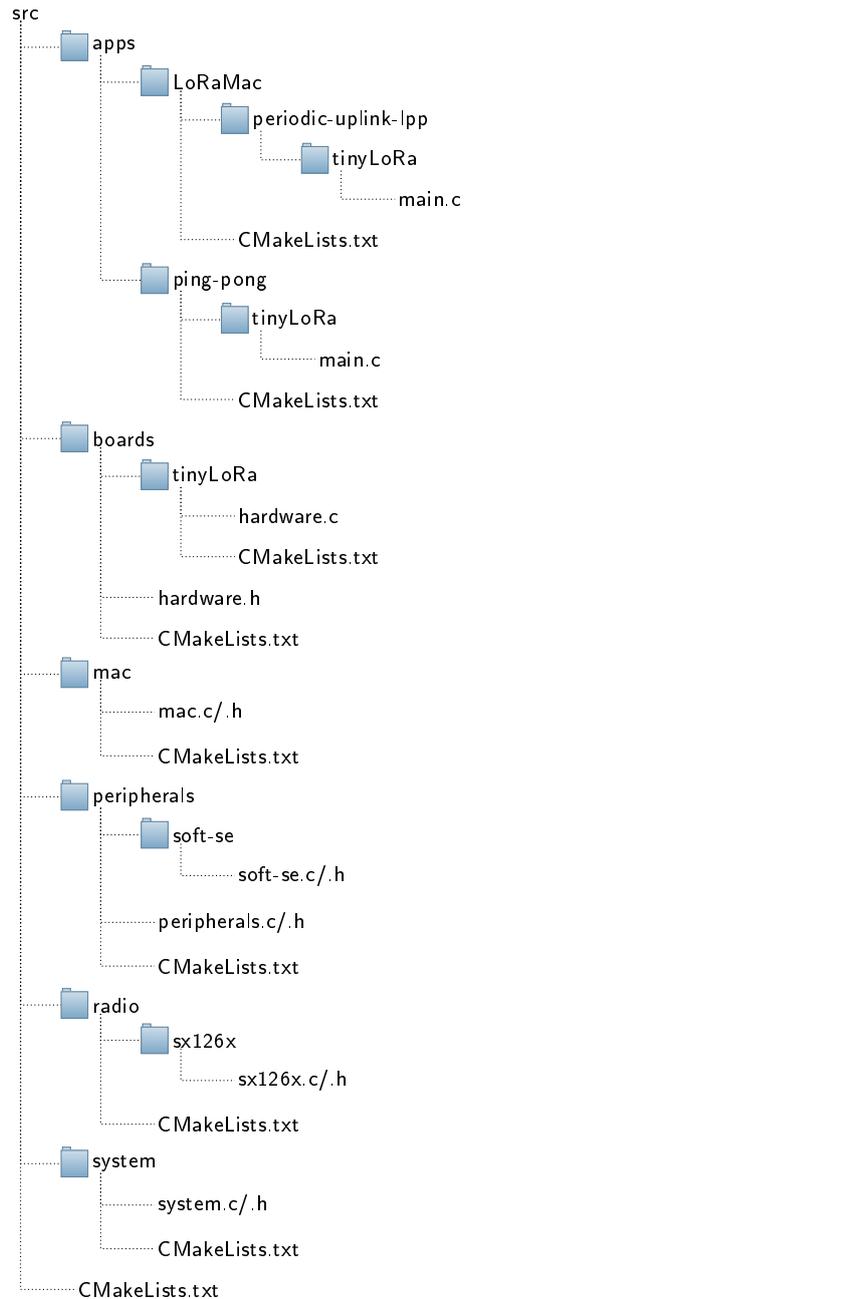


Abbildung 34: Struktur der Source- und Headerfiles

Unter `apps` sind verschiedene Applikationen vorhanden, welche bereits vollständig implementiert sind. Das `tinyLoRa` unterstützt zwei davon:

- `periodic-uplink-lpp`: Diese Anwendung macht Gebrauch von dem LoRa-Netzwerk. Das Gerät muss auf dem *The Things Network* registriert werden und dabei werden Gerätespezifische Schlüssel erstellt. Diese müssen in der Datei `src/peripherals/soft-se/se-identity.h` eingetragen werden. Wenn dies gemacht ist, kann die Anwendung übersetzt werden und sendet danach periodisch Verbindungsanfragen und Daten an das Netzwerk. Dies kann auf der Konsole auf *The Things Network* beobachtet werden. Mit dieser Anwendung kann sichergestellt werden, dass sich das Gerät nahe genug an einem Gateway befindet und die Anwendung lässt sich abändern, sodass beispielsweise Sensordaten verschickt werden können.

- ping-pong: Im Gegensatz zu *periodic-uplink-lpp* muss das Gerät für diese Anwendung nicht mit dem Netzwerk verbunden werden, jedoch sollten dafür zwei Geräte vorhanden sein. Wenn diese Applikation läuft, wird vom Gerät eine Nachricht gesendet und danach wird gewartet, bis eine Nachricht zurückkommt. Nachdem eine Nachricht eingetroffen ist, wird eine gesendet. Somit ist es möglich, zwei End-Devices miteinander kommunizieren zu lassen, ohne dass diese über das Netzwerk verbunden sind.

In dem Verzeichnis `mac` ist der komplette LoRa-Kommunikationsstack implementiert. Das heisst, dass dort die ganzen Nachrichten mit Header, Verschlüsselungen, Buffer, Zeitspannen und Adressen verwaltet werden. Dies kommt vor allem bei einer Anwendung, bei welcher sich auf das Netzwerk verbunden wird, zum Tragen, da dort die Anforderungen, um sich auf das Netzwerk zu verbinden, sehr genau eingehalten werden müssen.

Da LoRa sehr oft für das Versenden von Sensorwerten verwendet wird, ist die Ansteuerung und Auslesung einiger Sensoren in dem Verzeichnis `peripherals` bereits implementiert. Beispielsweise werden Sensoren für Temperatur-, Feuchtigkeits- oder Beschleunigungsmessungen unterstützt. Zudem ist im Ordner `peripherals/soft-se` eine Unterstützung von Secure Elements in Software vorhanden, welche zur Verschlüsselung von Daten verwendet werden und benötigt werden, um das Gerät mit dem Netzwerk zu verbinden.

Im Unterordner `radio` ist die Ansteuerung des LoRa-Transceivers implementiert. Es werden drei verschiedene Transceiver-Familien unterstützt. Die entsprechende Familie kann wiederum in den Präprozessoreinstellungen ausgewählt werden.

Unter `system` sind verschiedene Module implementiert, die auf das Hardwarelayer des entsprechenden Boards zugreifen. Hauptsächlich verwenden diese Module Funktionen aus dem Ordner `boards`, welcher im Folgenden noch beschrieben wird. Zusätzlich werden hier die verschiedenen Hardwarefunktionalitäten verwaltet und konfiguriert.

Um das komplette Projekt auf verschiedenen Plattformen verwenden zu können, braucht es für jede Plattform die entsprechende Hardwareansteuerung. Dies wird unter `boards` umgesetzt, dort finden sich `.h` Dateien für jedes Hardwaremodul und im entsprechenden Unterordner der Plattform, in diesem Fall `boards/tinyLoRa`, befinden sich die `.c` Dateien dazu. Dort finden sich insgesamt 13 Module, jedes davon ist für verschiedene Hardwarefunktionalitäten zuständig. Die Funktionen, die diese Module beinhalten, sind bereits definiert, damit eine klare Schnittstelle zur Verwendung dieser Module besteht. Die Implementation dieser Funktionen muss für jeden Mikrocontroller separat gemacht werden, was für den RP2040 Teil dieser Arbeit ist. Die Module beinhalten jeweils eine Initialisierungs- und Deinitialisierungsfunktion, sowie Funktionen zum Verwenden der Hardware, beispielsweise das Auswerten oder Toggeln eines GPIO-Pins oder das Lesen und Schreiben von SPI, I2C und UART.

4.2 | CMake

Um die im vorherigen Unterkapitel erläuterte Projektstruktur zu übersetzen wird CMake verwendet. Wo überall CMake Dateien vorhanden sind, ist in Abbildung (33) und (34) dargestellt. Auch dazu war bereits viel implementiert und konfiguriert, einige Anpassungen mussten jedoch vorgenommen werden, um das Projekt erfolgreich für den RP2040 übersetzen zu können. Dazu wurde das File `CMakeLists.txt` im Hauptprojektordner folgendermassen konfiguriert:

```

1 # minimum required cmake version
2 cmake_minimum_required(VERSION 3.12)
3
4 # include build functions from Pico SDK
5 include($ENV{PICO_SDK_PATH}/external/pico_sdk_import.cmake)
6
7 # name of project
8 project(tinyLoRa-loramac-node)
9
10 # init sdk
11 pico_sdk_init()
12
13 # add subdirectories
14 add_subdirectory(src)

```

Codesequenz 2: CMakeLists.txt in der obersten Verzeichnisebene

Wichtig ist, dass das komplette SDK jeweils inkludiert und initialisiert wird, um es zu verwenden. Des Weiteren müssen in jedem Verzeichnis, das einzelne Module aus dem SDK verwendet, diese so mit CMake inkludiert werden:

```

1 # add all used libraries from the pico sdk
2 target_link_libraries(${PROJECT_NAME} INTERFACE
3     pico_stdlib
4     pico_time
5     hardware_gpio
6     hardware_spi
7 )

```

Codesequenz 3: Hinzufügen der SDK-Bibliotheken

Am Anfang gab es Probleme mit CMake, denn so wie CMake zu Beginn aufgesetzt war, führte dies zu Linker-Fehler. Es wurden einerseits in dem Verzeichnis `apps`, sowie in `boards` Funktionen aus dem SDK verwendet. CMake war zu Beginn so konfiguriert, dass aus jedem der sechs Verzeichnisse in `src` eine Bibliothek erstellt wurde. Somit entstanden zwei Bibliotheken, die dieselben Funktionen des SDKs beinhalten. Dies hat dazu geführt, dass der Linker zwei Definitionen von derselben Funktion hatte und somit das Projekt nicht gelinkt werden konnte. Die Lösung zum Problem ist, dass das Verzeichnis `boards` nicht in eine Standardbibliothek übersetzt wird, sondern zu einer Interface-Bibliothek. In CMake werden Interface-Bibliotheken nicht separat übersetzt, diese werden zu anderen Bibliotheken hinzugefügt und diese dann zusammen übersetzt. Dies bedeutet, dass nun nur eine Bibliothek direkt Funktionen vom SDK inkludiert und somit das Problem behoben wurde [6].

Der untenstehende Codeausschnitt (4) wird im folgenden erläutert. Die Schritte sind anhand des CMake-Files der periodic-uplink-lpp-Applikation erklärt. In den CMake-Files die sich direkt über der `main.c` Datei befindet, muss das SDK nochmals eingefügt werden und die Datei `main.c` explizit zu den ausführbaren Quelldateien hinzugefügt werden. Da der RP2040 `.uf2` Dateien benötigt, müssen diese durch die Funktion `pico_add_extra_outputs` erzeugt werden.

Zudem kann gewählt werden, ob alle Ausgaben, beispielsweise durch `printf`, über die UART oder über USB ausgegeben werden. Falls das Board an einer Debug-Probe angeschlossen ist, wird die UART-Ausgabe eingeschaltet, um die Daten vom Target über UART an die Debug-Probe zu übermitteln. Falls das Target direkt über den USB-Anschluss vom Host geflasht wird, kann USB ausgewählt werden und danach mit einem Terminal-Programm wie *PuTTY* die Ausgaben beobachtet werden. Zum Schluss werden noch die nötigen Bibliotheken dazugelinkt, im Beispiel von `main.c` wird die Mathematikbibliothek, die `pico_stdlib` für Konsolenausgaben und die Interface-Bibliothek, die im `board` Verzeichnis erstellt wurde, dazugelinkt.

```
1 # set the project name, language versions and include the sdk
2 project(LoRaMac C CXX ASM)
3 cmake_minimum_required(VERSION 3.12)
4
5 include($ENV{PICO_SDK_PATH}/external/pico_sdk_import.cmake)
6
7 set(CMAKE_C_STANDARD 11)
8 set(CMAKE_CXX_STANDARD 17)
9
10 ...
11
12 add_executable(${PROJECT_NAME}-${SUB_PROJECT}
13 ...
14   ${CMAKE_CURRENT_SOURCE_DIR}/periodic-uplink-lpp/tinyLoRa/main.c
15   )
16
17 ...
18
19 # Create map/bin/hex/uf2 files
20 pico_add_extra_outputs(${PROJECT_NAME}-${SUB_PROJECT})
21
22 # disable USB output, enable uart output
23 pico_enable_stdio_usb(${PROJECT_NAME}-${SUB_PROJECT} 0)
24 pico_enable_stdio_uart(${PROJECT_NAME}-${SUB_PROJECT} 1)
25
26 target_link_libraries(${PROJECT_NAME}-${SUB_PROJECT} m pico_stdlib ${BOARD})
```

Codesequenz 4: CMake-File für die Applikation

4.3 | Pico SDK

Das Raspberry Pi Pico SDK beinhaltet ein umfangreiches API, welches Funktionalitäten zur einfachen Initialisierung und Implementierung von allen verbreiteten Hardwarefunktionalitäten bietet. In untenstehender Tabelle (2) werden die im tinyLoRa-Projekt verwendeten Module kurz vorgestellt.

Modul	Beschreibung
<code>pico/stdlib.h</code>	Dies ist eine Sammlung von Modulen, welche die wichtigsten Funktionalitäten für ein simples Programm enthält, beispielsweise Ein- und Ausgabefunktionen oder die Initialisierung der Systemclocks aber auch Informationen für die Runtime oder den Linker werden hier zusammengefasst.
<code>pico/time.h</code>	Enthält Funktionen um Timer-Interrupts aufzusetzen.
<code>hardware/gpio.h</code>	Modul für die einfache Konfiguration, Initialisierung und Ansteuerung von GPIO-Pins.
<code>hardware/spi.h</code>	Definiert Funktionalitäten für die Inbetriebnahme und die Lese- und Schreibzugriffe von SPI-Instanzen.
<code>hardware/i2c.h</code>	Dieses Modul beinhaltet das Aufsetzen und die Betriebsfunktionen für I2C-Instanzen.
<code>hardware/flash.h</code>	Mit diesem Modul lässt sich der externe Flash lesen, beschreiben und löschen.
<code>hardware/sync.h</code>	Implementiert Funktionalitäten für Semaphoren, Mutexen und Critical Sections.
<code>hardware/rtc.h</code>	Beinhaltet Funktionen für die Verwendung eines Real Time Clocks.
<code>hardware/clocks.h</code>	Hier lassen sich die Clocks der verschiedenen Hardwarefunktionen konfigurieren.
<code>hardware/irq.h</code>	API für die Konfiguration und Verwendung von Interrupts.
<code>hardware/timer.h</code>	Modul um den Prozessor für eine gewisse Zeit in einen Wartezustand zu setzen.
<code>hardware/adc.h</code>	Definiert Funktionen zum Aufsetzen und Auslesen von Analog-Digital-Konvertern.

Tabelle 2: Verwendete Module aus dem SDK

4.4 | Hardwareabstraktion

Wie bereits erwähnt befindet sich im Verzeichnis `src/boards` die plattformabhängige Ansteuerung der benötigten Hardware. Dazu gehören unter anderem die Kommunikation über SPI, das Ansteuern von GPIO-Pins und die Inbetriebnahme von Interrupts dazu, die Programmierung des Flashs oder das Aufsetzen von Timern mit Interrupts. Um diese und weitere Funktionalitäten zu implementieren, wird das oben vorgestellte Pico SDK verwendet, was das Umsetzen der Anforderungen vereinfacht. Auf die oben genannten Module wird in den folgenden Unterkapiteln eingegangen.

4.4.1 | GPIO mit RP2040

Der RP2040 bietet 30 GPIO-Pins, welche durch die Schnittstelle `gpio-board.c` konfiguriert und benutzt werden können. Dazu stehen verschiedene Funktionen zur Verfügung, welche als Argument jeweils einen GPIO-Handle erwarten. In der Initialisierungsfunktion wird ein nicht

konfigurierter Handle, sowie alle wichtigen Parameter zur Initialisierung übergeben. Diese Parameter beschreiben beispielsweise, ob der Pin ein Input oder Output ist, ob verschiedene Pull-Widerstände eingeschaltet werden sollen oder auch ob der Pin zur Initialisierung auf das High-Level geschaltet wird. Der Pin wird initialisiert und die wichtigen Informationen in den Handle geschrieben. Neben der Initialisierungsfunktion bestehen noch Funktionen, um einem GPIO-Pin eine Interrupt Service Routine zu hinterlegen, den Pin zu lesen oder zu schreiben. Dazu werden überall die Handles verwendet, welche folgende Struktur aufweisen:

```

1 typedef struct
2 {
3     PinNames pin;
4     uint16_t pinIndex;
5     void *port;
6     uint16_t portIndex;
7     PinTypes pull;
8     void* Context;
9     GpioIrqHandler* IrqHandler;
10 }Gpio_t;

```

Codesequenz 5: GPIO-Handle

Auf dem RP2040 werden die Felder `*port` und `portIndex` nicht verwendet, da lediglich die Nummer des Pins erforderlich ist, um diesen anzusteuern.

4.4.2 | SPI mit RP2040

Der RP2040 Mikrocontroller bietet zwei SPI-Instanzen, für welche je ein Handle erstellt wird. In diesen Handles ist jeweils die Hardwareinstanz hinterlegt, so können mit diesen Handles über Funktionsaufrufe verschiedene SPI-Funktionalitäten benutzt werden. Des Weiteren werden in den Handles die Konfiguration des SPI-Protokolls hinterlegt, wie beispielsweise die Clockphase und -polarität. Im Modul `spi-board.c` sind Funktionen zum Initialisieren und Deinitialisieren der Instanzen, sowie Funktionen zum Senden und Empfangen von Daten vorhanden.

4.4.3 | Flash-Programmierung mit RP2040

In `eprom-board.c` sind Funktionen zum Lesen und Schreiben des Flashs vorhanden. Das Lesen des Flashs ist relativ simpel, es wird ein Pointer erstellt, der auf die Startadresse des reservierten Flashbereichs zeigt. Wenn nun ein Byte an einer bestimmten Adresse gelesen werden soll, kann der Pointer um diese Adresse erhöht werden und dann wird der Inhalt dieser Adresse abgefragt. So lassen sich eine beliebige Anzahl an Bytes lesen. Dies ist im untenstehenden Codeabschnitt gezeigt:

```

1 /* set start address of the buffer to read, XIP_BASE = start address of flash */
2 static const uint8_t *flash_target_contents = (const uint8_t *) (XIP_BASE +
3     FLASH_TARGET_OFFSET);
4
5 /* read flash and store data in buffer */
6 for(int i = 0; i < size; i++){
7     buffer[i] = flash_target_contents[i + addr];
8 }

```

Codesequenz 6: Lesen des Flashs

Das Beschreiben des Flashs ist deutlich aufwendiger. Der Flash ist in Sektoren von 4 *kByte* aufgeteilt. Um einen Bereich des Flashs zu beschreiben, muss dieser vorher immer gelöscht werden, was nur in Blöcken von ganzen Sektoren möglich ist. Das heisst, dass die zu beschreibenden

Blöcke zuerst gelesen werden müssen. Dazu werden diese vom Flash ins RAM kopiert. Danach werden die betroffenen Flashsektoren gelöscht, die alten Daten werden im RAM mit den neuen überschrieben und danach wird alles wieder zurück in den Flash gespeichert. Dies wurde alles in der Funktion `EepromMcuWriteBuffer(int addr, int *buffer, int size)` implementiert. Diese ist ein Beispiel einer Funktion, die vom Kommunikationsstack verwendet wird, deren Implementation aber von Plattform zu Plattform unterschiedlich ist. Mit dieser Funktion kann ein Datenbuffer einer gewissen Länge an eine gewünschte Adresse geschrieben werden. Somit entsteht durch das oben beschriebene Verfahren ein gewisser Rechenaufwand, was das folgende Beispiel aufzeigen soll:

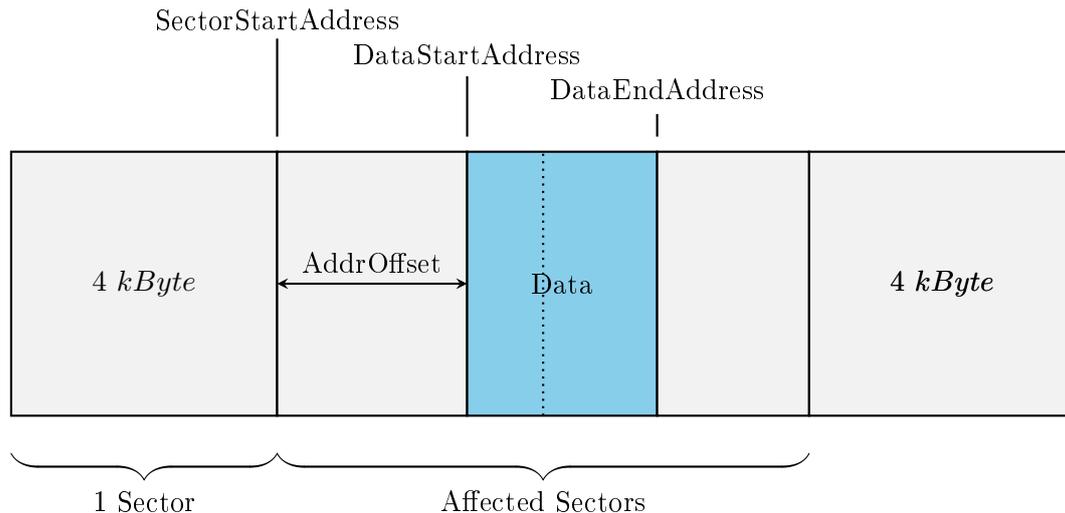


Abbildung 35: Flash Programmierung

Die Adresse ist nicht auf den Anfang eines Sektors ausgerichtet, es muss also die Startadresse des entsprechenden Sektors berechnet werden. Die Differenz zwischen Startadresse und Benutzeradresse ist der Adressenoffset. Danach wird ermittelt, wie viele Sektoren die zu programmierenden Daten mit dem Offset einnehmen. Jetzt müssen alle betroffenen Sektoren komplett vom Flash in den RAM geschrieben werden und danach die entsprechenden Sektoren gelöscht werden. Nun werden die alten Daten mit den neuen im RAM überschrieben, dazu wird die Benutzeradresse, die Länge und die Daten selber benötigt. Nun können alle Daten vom RAM wieder zurück in den Flash geschrieben werden. Die Funktion `EepromMcuWriteBuffer` überprüft am Anfang, ob sich die Adresse in einem gültigen Bereich befindet.

4.4.4 | Timer mit RP2040

Für jegliche Art von Kommunikation ist Zeitmanagement essenziell. Das tinyLoRa verwendet dafür einen Real Time Clock und Timer, welche im File `rtc-board.h` implementiert werden. Der RTC wird benötigt, um die vergangene Zeit seit Systemstart abrufen zu können. Dies wird für Verifikationszwecke gebraucht, wenn das tinyLoRa mit dem Netzwerk verbunden werden soll. Dafür braucht es diverse Zeitstempel, sodass eine gesendete Nachricht vom Netzwerk als gültig erklärt wird [29]. Es existieren dafür zwei Funktionen, mit denen der aktuelle Zeitstempel im Flash abgelegt werden kann und beim nächsten Systemstart wieder gelesen werden kann. Zudem werden Timer benutzt, um verschiedene Timeout-Funktionalitäten zu realisieren und Timerinterrupts zu verwenden. In der Init-Funktion wird der RTC gestartet, ab diesem Zeitpunkt wird

der RTC-Counter jede Sekunde inkrementiert und das Zeitformat wird automatisch angepasst. Durch eine weitere Funktion `RtcGetCalendarTime` kann die momentane Zeit vom Standardzeitformat in die Anzahl vergangener Sekunden seit Systemstart zurückgerechnet werden. Des Weiteren existieren Funktionen für das Setzen von Timern und deren Serviceroutinen. Zudem ist in dem Modul die Variable `RtcTimerContext` angelegt. Dabei handelt es sich um einen 32 Bit Integer, der immer den Zeitstempel in μs des Zeitpunkts, an dem zum letzten Mal ein Timer gesetzt wurde, enthält. Dies wird verwendet, damit die Anwendung immer eine zeitliche Referenz auf den nächsten Interrupt hat.

4.4.5 | Weitere Module

In den vier vorherigen Abschnitten wurden die wichtigsten Module des LoRa-Stacks vorgestellt, dies sind jedoch nicht alle vorhandenen Module, die zur Hardwareabstrahierung gehören. Hier werden die weiteren Module zusammengefasst, welche im LoRa-Stack vorhanden sind. Eine Liste dieser findet sich in der folgenden Tabelle und gibt an, wie weit die Portierung dieser für das tinyLoRa fortgeschritten ist:

Modul	Beschreibung	Status
<code>adc-board.c</code>	Funktionen zum Verwenden des Analog-Digital-Converter	Implementiert
<code>board.c</code>	Allgemeine Funktionen zum Betrieb des Boards	Teilweise implementiert
<code>delay-board.c</code>	Funktionen zum Verzögern von Prozessen	Implementiert
<code>gps-board.c</code>	Funktionen zum Verwenden von GPS-Geräten	Nicht implementiert
<code>i2c-board.c</code>	Funktionen zur Verwendung vom I2C-Protokoll	Implementiert
<code>lpm-board.c</code>	Funktionen zur Benutzung des Low-Power-Modus	Nicht implementiert
<code>uart-board.c</code>	Funktionen zur Verwendung vom UART-Protokoll	Implementiert

Tabelle 3: Module des LoRa-Stacks und deren Implementationsstatus

Die Module für die I2C- und die UART-Kommunikation sind sehr ähnlich aufgebaut wie diese für die SPI-Kommunikation und wurden implementiert. Im Modul `adc-board.h` kann der ADC initialisiert und dann durch die Übergabe eines GPIO-Handles der entsprechende ADC-Kanal ausgelesen werden. In `board.c` sind verschiedenste Funktionen vorhanden, viele davon betreffen den Low-Power-Modus, welche nicht implementiert sind, alle anderen sind vorhanden. Die Module, welche die GPS- und Low-Power-Modus-Unterstützung realisieren, wurden in diesem Projekt nicht auf den RP2040 portiert. Da sich auf dem Board kein GPS-Gerät befindet, ist `gps-board.h` auch nicht notwendig. Falls jedoch das Board für eine batteriebetriebene Anwendung benutzt wird, ist die LPM-Unterstützung sicherlich notwendig.

4.5 | Resultate Software

Das Portieren des Softwarestacks hat um einiges mehr an Zeit beansprucht, als eingeplant war. Bei vielen der implementierten Funktionalitäten konnte eine korrekte Funktionsweise nachgewiesen werden. Dies beispielsweise bei den Modulen für GPIO-Pins, SPI-Kommunikation oder Flash-Programmierung. Probleme sind noch bei den Timern im Zusammenspiel mit dem LoRa-Stack auszumachen. Die Timer funktionieren an und für sich, jedoch sind noch Fehler vorhanden,

wenn diese vom Stack benutzt werden. Für die Anwendung bedeutet das konkret, dass die Ping-Pong-Applikation auf zwei tinyLoRa-Boards funktioniert, dies ist in Abbildung (36) dargestellt.



```
COM4 - PuTTY
Sent a ping or pong!
rx done
Received a Pong! From follower.
Sent a ping or pong!
rx done
Received a Pong! From follower.
Sent a ping or pong!
rx done
Received a Pong! From follower.
Sent a ping or pong!
rx done
Received a Pong! From follower.
Sent a ping or pong!
rx done
Received a Pong! From follower.
```

Abbildung 36: Konsolenausgabe der Ping-Pong-Applikation

Bei der Verbindung auf das Netzwerk sind noch Probleme beim Empfangen von Daten vom Gateway auf das tinyLoRa vorhanden. Der Grund dafür liegt vermutlich in den Timern, denn das tinyLoRa wartet für eine gewisse Zeit auf eine Antwort vom Gateway. Wenn nun die Timer nicht korrekt aufgesetzt werden, dürfte das Zeitmanagement und somit die Synchronisation zwischen Gateway und End-Device nicht stimmen. Die genaue Fehlerursache konnte bis zum Schluss des Projekts nicht gefunden werden. Somit konnte das Ziel, den LoRa-Stack vollumfänglich auf den RP2040 zu portieren, nicht erfüllt werden. Viele von den bearbeiteten Modulen erfüllen ihre Funktionen, allerdings nicht alle.

5 | Messungen

In diesem Kapitel werden die Resultate der durchgeführten Messungen und Tests vorgestellt. Zum einen wurde der Energieverbrauch des tinyLoRa bei verschiedenen Betriebszuständen gemessen, zum anderen wurden zwei verschiedene Impedanzanpassungen einem Reichweiteversuch unterzogen.

5.1 | Energiemessung

Um die Messung des Energieverbrauch durchzuführen, wurde ein Leistungsanalysator verwendet. Dabei handelt es sich um das Modell Otii Arc 1.0⁹ von der Firma Qoitech. Damit konnte der Stromverbrauch des tinyLoRa bei verschiedenen Speisespannungen gemessen werden. Es wurde einerseits der Verbrauch des kompletten Boards bei Speisung an dem Batteriestecker und an dem USB-Anschluss und andererseits der Energieverbrauch des Mikrocontrollers alleine gemessen. Alle Messungen wurden durchgeführt, während die in Abschnitt (4.1) beschriebene Ping-Pong-Applikation auf dem Board lief. In dem Messversuch sendet das Board jede Sekunde eine Nachricht, so ist ersichtlich, welchen Einfluss auf den Strombedarf das Senden einer Nachricht hat. Laut Datenblatt des LoRa-Transceivers benötigt das Senden von Nachrichten deutlich mehr Energie als das Empfangen dieser.

5.1.1 | Energieverbrauch tinyLoRa bei Batteriespannung

Bei dieser Messung wurde das Board am Batteriestecker mit einer Spannung von 3.7 V gespeist. Dies entspricht der Nennspannung der LiPo-Zelle. In der folgenden Abbildung (37) ist der Stromverbrauch dieser Situation dargestellt. Die zwei Graphen stellen den Stromverbrauch mit und ohne Benutzen der User-LED dar.

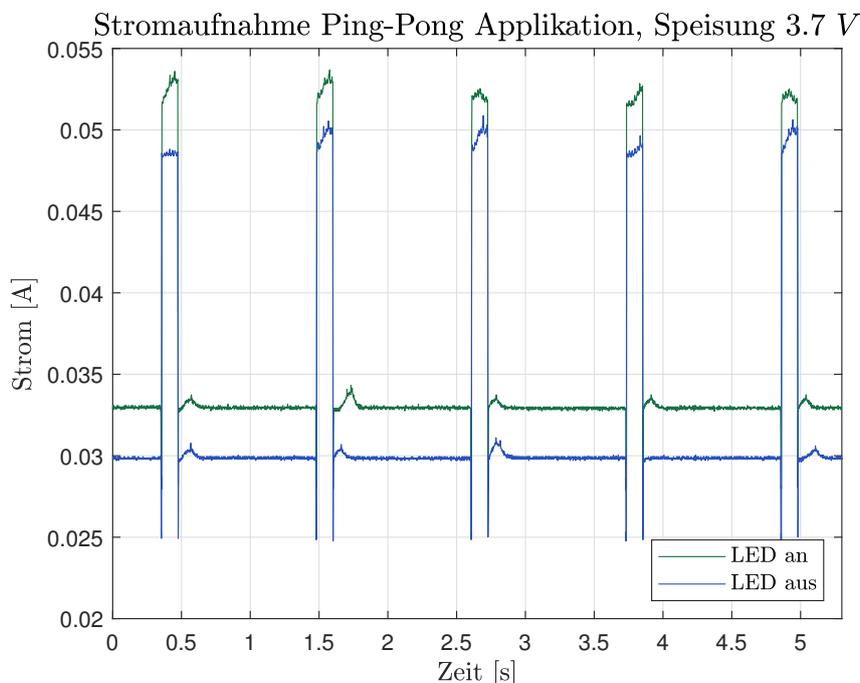


Abbildung 37: Stromaufnahme des tinyLoRa über Batterie gespeisen

⁹Seriennummer: 180915130028

Im Plot ist deutlich zu erkennen, zu welchen Zeiten eine Nachricht gesendet wird. Dort ist der Strombedarf des Boards um ca. 20 mA höher. Dem Datenblatt (S. 17, [19]) ist zu entnehmen, dass der Transceiver beim Senden um die 25 mA an Strom braucht. Somit sind die Stromspitzen in einem plausiblen Bereich. Es ist auch zu erkennen, dass durch das Verwenden der LED der Strombedarf um ca. 3 mA steigt.

5.1.2 | Energieverbrauch tinyLoRa bei USB-Spannung

Nun wird das tinyLoRa über die USB-Buchse mit 5 V gespeisen. Der Strombedarf wurde zweimal aufgezeichnet, einmal mit angeschlossener Batterie und einmal ohne diese. Die LED wurde bei dieser Messung nicht verwendet. So kann beobachtet werden, wie sich das Laden der Batterie auf den Stromverlauf auswirkt. Im Plot (38) ist zu sehen, dass das Laden der Batterie den Stromfluss um ca. 200 mA erhöht. Da der Ladestrom der Batterie auf 196 mA eingestellt ist, liegt diese Messung ebenfalls im erwarteten Rahmen.

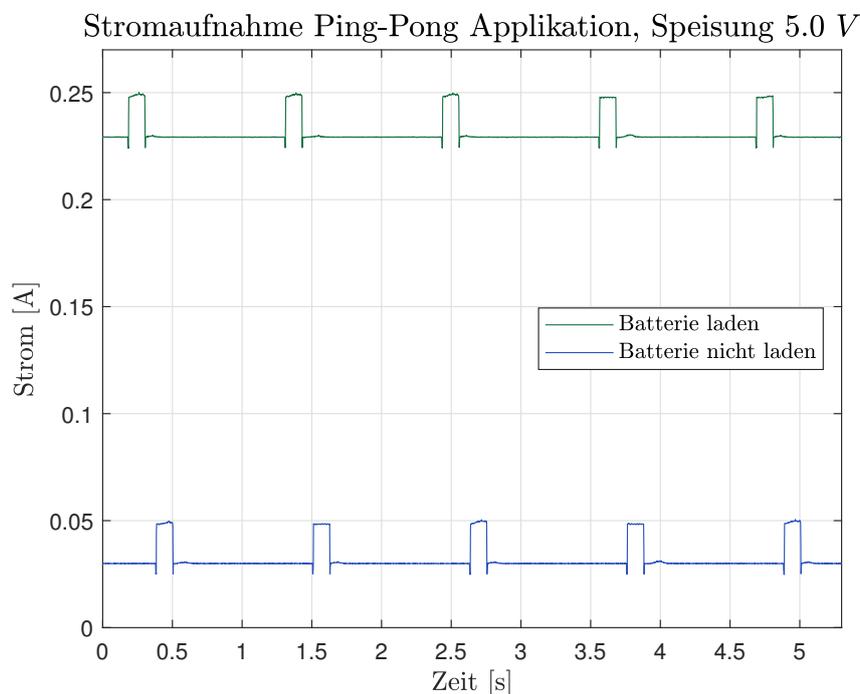


Abbildung 38: Stromaufnahme des tinyLoRa über USB gespeisen

Spannend hier ist, dass der Strombedarf trotz höherer Speisespannung bei Betrieb ohne Batterie gleich hoch ist, wie bei der Messung im vorherigen Abschnitt, wo das Board mit 3.7 V gespeisen wurde. Dies liegt an dem verwendeten Spannungsregler, welcher ein Linearregler ist. Somit wird die überschüssige Energie in Wärmeenergie umgewandelt, was bei einer energieeffizienten Anwendung sicherlich nicht optimal ist. Es wurde ein Linearregler gewählt, da dieser noch in höheren Stückzahlen zu einem guten Preis auf dem Markt vorhanden war. Dieser sollte aber für künftige, energiesparende Anwendungen durch einen effizienteren Regler ersetzt werden.

5.1.3 | Energieverbrauch Mikrocontroller

Als letztes wurde der Energiebedarf vom Mikrocontroller alleine gemessen. Dafür diente der Jumper JP1, der in Abbildung (24) zu sehen ist. In der untenstehenden Abbildung (39) ist der Stromverlauf dargestellt, auch hier wurde einmal die LED verwendet und einmal nicht.

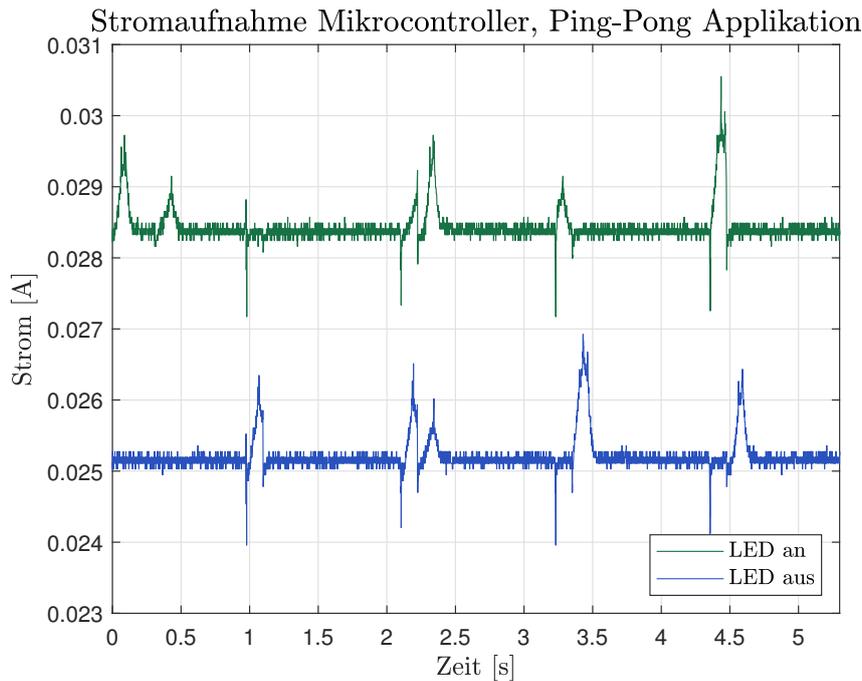


Abbildung 39: Stromaufnahme des Mikrocontroller

Es ist zu sehen, wie der Mikrocontroller periodisch mit dem LoRa-Transceiver kommuniziert. Generell beläuft sich der Strombedarf auf etwas mehr als 25 mA , beziehungsweise 28 mA . Dies allerdings ohne die Verwendung eines Low-Power-Modus, welcher grosses Einsparpotential hätte.

5.1.4 | Fazit aus Energiemessungen

Aus den obigen Plots lässt sich feststellen, dass sich bei Speisung durch eine Batterie und regelmäßigem Senden von Daten der Strombedarf des tinyLoRa-Boards im Schnitt auf ca. $35\text{--}40\text{ mA}$ beläuft, was relativ hoch ist. Dieser Verbrauch liesse sich durch das Verwenden eines Low-Power-Modus und einem geschalteten Spannungsregler deutlich reduzieren. Das Senden von Daten benötigt viel Energie, es ist aber unüblich, dass Daten im Sekundentakt versendet werden, und somit würde sich bei einer konkreten Anwendung der Verbrauch nochmals reduzieren. Die untenstehende Formel (4) soll ein mögliches Szenario für den Energieverbrauch zeigen:

$$t = \frac{C_{LiPo}}{i_{Board}} = \frac{500\text{ mAh}}{40\text{ mA}} = 12.5\text{ h} \quad (4)$$

Mit einem durchschnittlichen Verbrauch von 40 mA würde eine LiPo-Zelle mit einer Kapazität von 500 mAh das Board also für 12.5 h Speisen können. Wenn nun der Verbrauch auf angenommen 15 mA reduziert werden könne, läge nach obiger Gleichung die Zeit bei 33.3 h .

5.2 | Antenne

Die Impedanzanpassung der Antenne konnte aufgrund mangelnder Zeit nicht wie geplant mit Simulationen und Messungen explizit für das tinyLoRa durchgeführt werden. Allerdings waren zwei Referenzdesigns vorhanden, von welchen die Bauteilgrößen für das in Kapitel (3.2.3) beschriebene Impedanzanpassungsnetzwerk für das tinyLoRa verwendet wurden. Beim ersten

Board, von hieran als tinyLoRa.1 benannt, wurden die Bauteilwerte vom Projekt Tardigrade¹⁰ verwendet. Tardigrade ist ein Projekt, das den LoRa-Transceiver SX1262 von der Firma Semtech verwendet. Das andere Board (tinyLoRa.2) wurde mit den gleichen Bauteilen bestückt, welche auf dem LoRa-Shield (Abbildung (1)) vorhanden sind [23]. Die Impedanzanpassung und Filter sind in Abbildung (40) dargestellt und die jeweiligen Bauteilwerte in Tabelle (4) zu finden.

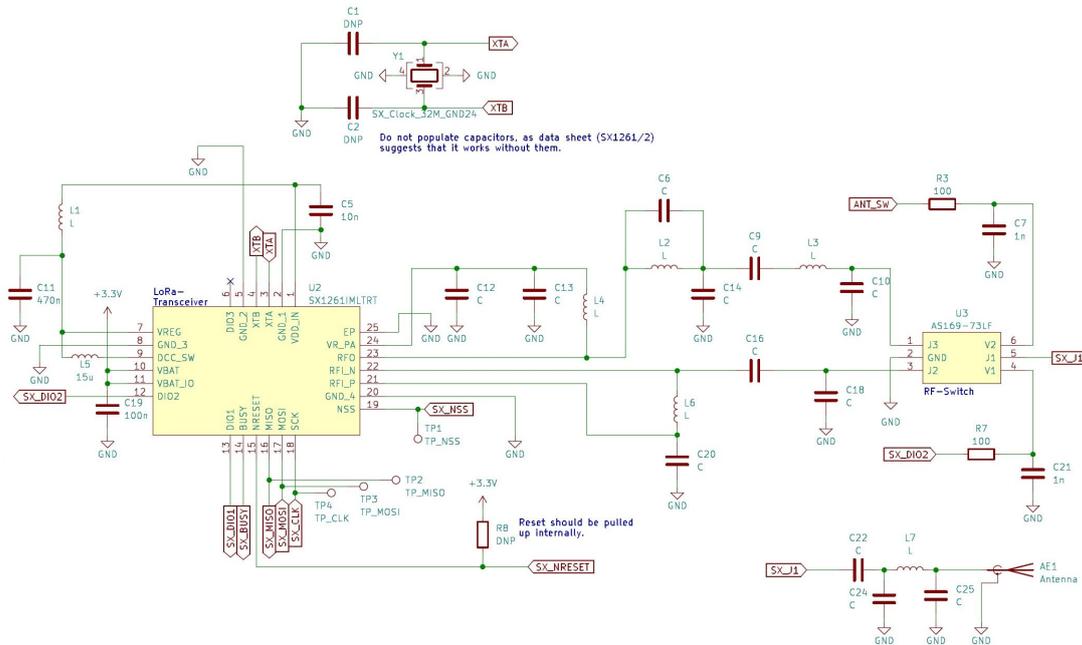


Abbildung 40: Impedanzanpassung und Filter

Bauteil	tinyLoRa.1	tinyLoRa.2	Einheit
C6	3.3	3.0	pF
C9	39	12.0	pF
C10	2.2	2.2	pF
C12	47.0	47.0	nF
C13	47.0	47.0	pF
C14	5.6	5.6	pF
C16	2.4	1.8	pF
C18	DNP	1.0	pF
C20	1.8	1.8	pF
C22	39	22.0	pF
C24	3.3	3.3	pF
C25	3.3	3.3	pF
L2	2.5	2.7	nH
L3	4.7	9.1	nH
L6	15.0	18.0	nH
L7	9.1	9.1	nH

Tabelle 4: Bauteildimensionierung der beiden Boards

¹⁰<https://github.com/nortismo/tardigrade>

Durch einen Versuch soll nun herausgefunden werden, welches der beiden Boards die grössere Sendeleistung abgibt. Dazu wird das LPC55S16-Board an einem fixen Ort mit der Ping-Pong-Applikation betrieben. Auf beiden tinyLoRa-Boards läuft ebenfalls die Ping-Pong-Applikation. Die Distanz zwischen LPC55S16 und den Boards wird nun in urbanem Gebiet zwischen Kriens und Horw sukzessive erhöht und so soll geschaut werden, welches der Boards über eine weitere Distanz mit dem LPC55S16 kommunizieren kann. Die Messung hat ergeben, dass mit dem tinyLoRa.1 über eine Distanz von nur gerade 480 m kommuniziert werden konnte. Mit dem tinyLoRa.2 ist die Distanz mit 1040 m deutlich höher und somit wird an der Antenne mehr Leistung abgegeben, allerdings ist anzumerken, dass diese Distanz ebenfalls kleiner als erwünscht ist (Abbildung (41)). Laut Semtech sind in urbanem Gebiet mit LoRa Sendedistanzen von bis zu 5 km möglich [22]. Beim Versuch wurde das LPC55S16 und die beiden Boards auf Bodenhöhe betrieben. Es ist denkbar, dass grössere Distanzen möglich sind, wenn mindestens eines der kommunizierenden Geräte höher gelegen wäre. Zudem wurde der Versuch in einer Umgebung durchgeführt, wo viele grössere Wohnblöcke und Industriegebäude liegen, was ebenfalls einen suboptimalen Einfluss auf die Übertragungsdistanz haben kann. Allerdings wurde aus dem Versuch klar, dass die Antennenanpassung vom tinyLoRa.2 deutlich die bessere ist.



Abbildung 41: Reichweite des tinyLoRa.2 mit Ping-Pong-Applikation

Es ist anzumerken, dass aus der durchgeführten Messung keine genauen Informationen über die effektive Sendeleistung, die von der Antenne abgegeben wird, gewonnen wurde. Ebenfalls ist die Reproduzierbarkeit des Messversuchs nicht gegeben. Die Messung wurde durchgeführt, um eine grobe Vorstellung der möglichen Reichweite zu erhalten und zu überprüfen, welches der beiden Designs geeigneter ist.

6 | Resultate

Die erzielten Resultate werden hier vorgestellt. Dazu gehören der Kostenpunkt des tinyLoRa, den Erfüllungsgrad der Anforderungen, verschiedene Verbesserungsmöglichkeiten und einen Ausblick. Für das Fazit aus dem PCB-Design ist auf das Kapitel (3.3.4) verwiesen, für die Resultate der Software kann Kapitel (4.5) konsultiert werden und für durchgeführte Messungen kann Kapitel (5) betrachtet werden. Diese Resultate sind aufgrund des besseren Leseflusses bei den entsprechenden Kapiteln bereits dokumentiert.

6.1 | Kosten

In untenstehender Tabelle (5) sind die totalen Kosten für ein Board aufgelistet. Dabei wird der Übersicht halber auf die genaue Preisangabe der einzelnen passiven Komponenten verzichtet und wurden zusammengefasst.

Bauteil	Kosten [CHF]	Bauteil	Kosten [CHF]
Spannungsregler	0.34	LiPo-Lademanagement	0.77
Mikrocontroller	1.21	Flash	1.57
LoRa-Transceiver	6.97	RF-Switch	0.79
Clock 32 MHz	0.62	Clock 12 MHz	0.70
USB-Stecker	0.99	Batteriestecker	0.40
LEDs	0.51	Buttons	0.98
Antennenbuchse	4.78	Antenne	4.85
Schottky-Dioden	0.88	Pin-Header	0.64
Passive Elemente	3.35	PCB	1.85
		Total	32.20

Tabelle 5: Totaler Kostenaufwand für ein tinyLoRa

Somit konnte die Anforderung, dass der Preis für ein Board bei ungefähr 30 CHF liegen soll, eingehalten werden. Einsparungspotential wäre sicher beim Flash vorhanden, denn dieser ist mit 16 *MByte* sehr gross gewählt, eine kleinere Speicherkapazität würde ebenfalls reichen und wäre kostengünstiger. Des Weiteren sind aufgrund der momentanen Halbleiterknappheit viele Bauteile teurer als normalerweise, was den Preis für ein Board ebenfalls in die Höhe trieb.

6.2 | Anforderungen

Die Anforderungen wurden in Kapitel (2.1.1) zusammengefasst dargestellt, hier wird auf diese Punkte nochmals eingegangen und deren Erfüllungsgrad ausgewertet.

Hardware Die Anforderungen an die Hardware konnte grossmehrheitlich erfüllt werden. Im Rahmen dieses Projekts wurde ein batteriebetriebenes, steckbrettcompatibles Board entwickelt, das über LoRa kommunizieren kann. Der Mikrocontroller kann problemlos geflasht und benutzt werden, Debugging ist vorhanden und es funktionieren ebenfalls alle Funktionalitäten des Controllers. Das Board lässt sich über eine LiPo-Zelle betreiben und diese kann über USB geladen werden. Die Pinverbindungen zwischen Mikrocontroller und LoRa-Transceiver sind allesamt korrekt und die SPI-Kommunikation funktioniert. Die Anforderungen, die Impedanzanpassung der Antenne explizit für das tinyLoRa auszulegen, konnte aus zeitlichen Gründen nicht realisiert

werden. Allerdings konnte durch die übernommene Impedanzanpassung das Funktionsprinzip nachgewiesen werden. Zudem ist das Board steckbrettcompatibel und mit 21 GPIO-Pins ist eine grosse Anzahl für den Benutzer vorhanden.

Software Es wurde die Anfangs definierte IDE VS Code verwendet, die Alternative, das Board in Eclipse zu programmieren, wurde nicht benötigt. Der Softwarestack wurde mehrheitlich auf den RP2040 portiert, die Ping-Pong-Applikation zwischen zwei tinyLoRa-Boards funktioniert. Bei der Verbindung auf das LoRa-Netzwerk sind noch Probleme vorhanden, welche höchstwahrscheinlich bei dem Zeitmanagement mit Timern auszumachen sind. Somit wurden die Anforderungen an die Software nur teilweise erfüllt. Es wurde gezeigt, dass die Kommunikation mit dem tinyLoRa möglich ist, allerdings ist die Software noch nicht so weit, dass damit eine konkrete Anwendung realisiert werden könnte. Zum Schluss des Projekts war nicht genügend Zeit vorhanden, die Probleme in der Software gründlich zu untersuchen.

6.3 | Optimierungsmöglichkeiten und Ausblick

Der nächste Schritt ist sicherlich, die oben genannten Probleme in der Software zu beheben und somit die Verbindung auf das Netzwerk zu ermöglichen, womit das tinyLoRa für diverse Anwendungsfälle verwendet werden könnte. Setzen diese Anwendungen eine energiesparende Funktionsweise des Boards voraus, ist die Implementierung eines Low-Power-Modus notwendig. Um eine weitere Optimierung des Energieverbrauchs vorzunehmen, kann ein anderer Spannungsregler verwendet werden, welcher kein Linearregler, sondern ein geschalteter Spannungsregler ist. Bei batteriebetriebenen Anwendungen wäre es ebenfalls von Vorteil, wenn bei einer Revision des Boards die Batteriespannung mit einem ADC-Input des Mikrocontroller verbunden wird, um so die Batterie zu überwachen. Die Impedanzanpassung der Antenne sollte sauber simuliert, berechnet und ausgemessen werden. Dafür müssten verschiedene Simulationen zur Geometrie des Boards durchgeführt werden und dann anhand von Messungen des LoRa-Transceivers und Antenne das Netzwerk aus passiven Elementen berechnet werden. Die erste Version des Boards hat eine Dicke von 1 *mm*. Dadurch ist das Board ziemlich flexibel und es wäre um einiges stabiler, wenn die Dicke auf 1.6 *mm* erhöht wird, was auch das Aufstecken und Wegnehmen von einem Steckbrett angenehmer macht.

7 | Fazit

Das Ziel dieser Arbeit, ein LoRa-fähiges Mikrocontrollerboard zu entwickeln darf als mehrheitlich erfüllt betrachtet werden. Die Hardware funktioniert einwandfrei, der Mikrocontroller und all seine Funktionalitäten lassen sich benutzen. Das entwickelte Board hatte von Anfang an keine grossen Fehler und es war keine zweite, korrigierte Version nötig. Allerdings wurden kleine Korrekturen vorgenommen, dass für eine allfällige nächste Version die aufgetretenen Fehler behoben sind. Das Board lässt sich mit einer Batterie betreiben und diese kann auch über das Board geladen werden. Mit dem portierten Kommunikationsstack konnte eine Funktionsdemonstration in Form der Ping-Pong-Applikation durchgeführt werden. Die Anforderungen an die Software konnten bis zum Ende des Projekts nicht vollumfänglich erfüllt werden, gerade wenn sich das tinyLoRa auf das Netzwerk verbinden soll. Um die vorhandenen Probleme ganz aufzuklären, hätte es noch einiges an Zeit benötigt. Trotzdem konnte der LoRa-Stack mehrheitlich portiert werden und bei vielen Funktionalitäten konnte die Funktionstüchtigkeit nachgewiesen werden. Die Reichweite der Antenne fiel am Ende etwas ernüchternd aus, auch in diesem Aspekt müsste noch Zeit investiert werden, um die Antenne sauber anzupassen, um so die Reichweite auf ein Maximum zu erhöhen. Trotzdem konnte über eine gewisse Distanz kommuniziert werden, was zeigt, dass die Geometrie des Boards und Anordnung der Bauteile sicherlich nicht allzu unpassend sind. Als abschliessendes Fazit lässt sich sagen, dass ein funktionstüchtiger Prototyp erstellt wurde, der die Anforderungen mehrheitlich erfüllt, der aber an verschiedenen Orten noch Optimierungspotential aufweist.

Verzeichnisse

Abbildungsverzeichnis

1	LPC55S16 mit SX1261 LoRa-Shield	5
2	OSI-Schichtenmodell mit LoRa und LoRaWAN	7
3	LoRaWAN Architektur mit End-Devices, Gateways und Servern	8
4	Raspberry Pi Pico Mikrocontrollerboard	9
5	Installationsverzeichnis der Toolchain	10
6	Zielpfad der Toolchain	11
7	Pfadvariable zur Toolchain	11
8	Pfadvariable zu CMake	12
9	Pfadvariable zu Python	12
10	Pfadvariable zum SDK	13
11	Hinterlegen der Toolchain in VS Code	14
12	Minimale Projektstruktur	14
13	CMake-Konfiguration- und Build-Button	14
14	Verdrahten der Debug-Probe und des Targets	17
15	Starten der Debug-Session	17
16	Blockschaltbild der Hardware des tinyLoRa-Boards	18
17	RP2040	18
18	USB4110-GF-A	19
19	SX1261	20
20	X9000984-4GDSMB	20
21	LED	21
22	Button	21
23	Schema: USB-Anschlüsse und Energieversorgung	22
24	Schema: Mikrocontroller und Flash	23
25	Verbindungen vom Mikrocontroller und LoRa-Transceiver	24
26	Schema: LoRa-Transceiver, Impedanzanpassung und Antenne	25
27	Schema: Breakout-Pins	26
28	3D-Ansicht der Front des PCBs	27
29	Frontansicht und gespiegelte Rückansicht des PCBs	27
30	Überkreuzungen von Leiterbahnen auf der Front- und Rückseite	28
31	Debuggen des tinyLoRa	30
32	Bestücktes und funktionstüchtiges tinyLoRa	30
33	Projektordner tinyLoRa	31
34	Struktur der Source- und Headerfiles	32
35	Flash Programmierung	38
36	Konsolenausgabe der Ping-Pong-Applikation	40
37	Stromaufnahme des tinyLoRa über Batterie gespiesen	41
38	Stromaufnahme des tinyLoRa über USB gespiesen	42
39	Stromaufnahme des Mikrocontroller	43
40	Impedanzanpassung und Filter	44
41	Reichweite des tinyLoRa.2 mit Ping-Pong-Applikation	45

Tabellenverzeichnis

1	Projektplan tinyLoRa	4
2	Verwendete Module aus dem SDK	36
3	Module des LoRa-Stacks und deren Implementationsstatus	39
4	Bauteildimensionierung der beiden Boards	44
5	Totaler Kostenaufwand für ein tinyLoRa	46

Codeverzeichnis

1	Auswählen der Applikation in settings.json	31
2	CMakeLists.txt in der obersten Verzeichnisebene	34
3	Hinzufügen der SDK-Bibliotheken	34
4	CMake-File für die Applikation	35
5	GPIO-Handle	37
6	Lesen des Flashs	37

Literatur

- [1] CTS Electronic Components. Model 406 Surface Mount Quartz Crystal. <https://www.mouser.ch/datasheet/2/96/008-0260-0-786182.pdf> (abgerufen am: 30.05.2022), online.
- [2] Digi-Key Electronics. 434123025826. https://www.digikey.ch/de/products/detail/w%C3%BCrth-elektronik/434123025826/9950832?utm_adgroup=Tactile%20Switches&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Switches&utm_term=&productid=9950832&gclid=Cj0KCQjwz7uRBhDRARIsAFqjulmy-eKVJ73xaPzjg0swpj1pYoes9uppz__YP-fAqmNgz9_gLftErv0aAv2SEALw_wcB (abgerufen am: 04.04.2022), April 2022.
- [3] Digi-Key Electronics. XZVG53W-8. <https://www.digikey.ch/de/products/detail/sunled/XZVG53W-8/4745971> (abgerufen am: 04.04.2022), April 2022.
- [4] Diodes Incorporated. AP2112. <https://www.diodes.com/assets/Datasheets/AP2112.pdf> (abgerufen am: 05.04.2022), Juni 2017.
- [5] Ethertronics. Part No. X9000984-4GDSMB. https://www.mouser.ch/datasheet/2/40/AVX_E_X9000984-2256054.pdf (abgerufen am: 05.04.2022), Juni 2020.
- [6] Kitware Inc. add_library. https://cmake.org/cmake/help/latest/command/add_library.html (abgerufen am: 17.05.2022), online.
- [7] Microchip Technologie Inc. MCP73831/2. https://componentsearchengine.com/Datasheets/1/MCP73831-5ACI_MC.pdf (abgerufen am: 05.04.2022), Januar 2008.
- [8] Mouser Electronics, Inc. Raspberry Pi RP2040 Microcontroller Chip. <https://www.mouser.ch/new/raspberry-pi/raspberry-pi-rp2040-chip/> (abgerufen am: 04.04.2022), März 2022.

- [9] Mouser Electronics, Inc. SX1261IMLTRT. <https://www.mouser.ch/ProductDetail/Semtech/SX1261IMLTRT?qs=sGAEpiMZZMvONwIthf1Bi9KYG80IE%252BZCvxXNg7KCP%2FI%3D> (abgerufen am: 04.04.2022), April 2022.
- [10] Mouser Electronics, Inc. USB4110-GF-A. <https://www.mouser.ch/ProductDetail/GCT/USB4110-GF-A?qs=KUoIvG%2F9I1YiZvIXQjyJeA%3D%3D> (abgerufen am: 04.04.2022), April 2022.
- [11] Mouser Electronics, Inc. X9000984-4GDSMW. <https://www.mouser.ch/ProductDetail/Ethertronics-Kyocera-AVX/X9000984-4GDSMW?qs=sGAEpiMZZMvONwIthf1BiyuCxipJQ8D7qgkwuioaag%3D> (abgerufen am: 04.04.2022), April 2022.
- [12] Nexperia. PMEG3050EP. <https://www.mouser.ch/datasheet/2/916/PMEG3050EP-2938519.pdf> (abgerufen am: 02.06.2022), Dezember 2017.
- [13] Raspberry Pi Trading Ltd. Getting started with Raspberry Pi Pico. <https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf> (abgerufen am: 22.03.2022), November 2021.
- [14] Raspberry Pi Trading Ltd. Hardware design with RP2040. <https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf> (abgerufen am: 05.04.2022), November 2021.
- [15] Raspberry Pi Trading Ltd. Raspberry Pi Pico C/C++ SDK. <https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf> (abgerufen am: 22.03.2022), November 2021.
- [16] Raspberry Pi Trading Ltd. Raspberry Pi Pico Datasheet. <https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf> (abgerufen am: 05.04.2022), November 2021.
- [17] Raspberry Pi Trading Ltd. RP2040 Datasheet. <https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf> (abgerufen am: 05.04.2022), November 2021.
- [18] Raspberry Pi Trading Ltd. Raspberry Pi Pico Pinout. <https://datasheets.raspberrypi.com/pico/Pico-R3-A4-Pinout.pdf> (abgerufen am: 30.05.2022), online.
- [19] Semtech Corporation. SX1261/2. <https://componentsearchengine.com/Datasheets/1/SX1261IMLTRT.pdf> (abgerufen am: 05.04.2022), Dezember 2017.
- [20] Semtech Corporation. Application Note: Recommendations for Best Performance. <https://semtech.my.salesforce.com/sfc/p/#E0000000Je1G/a/2R000000HSRX/x.Z011VNOR4sHFmdUe6NcnHdMHfAiTpsbEcINLkWDvE> (abgerufen am: 06.06.2022), Januar 2018.
- [21] Semtech Corporation. Application Note: Reference Design Explanation. https://semtech.my.salesforce.com/sfc/p/#E0000000Je1G/a/2R000000HSQj/MmehlfBHMs5N326wBVRKw68_1EoDBCvcB71X7BUQXY (abgerufen am: 31.05.2022), Mai 2018.
- [22] Semtech Corporation. What are LoRa® and LoRaWAN®? <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/> (abgerufen am: 21.03.2022), Februar 2020.

- [23] Semtech Corporation. SX1261MB2BAS 868 MHz mbed Shield. https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R000000Uhr2/bfkojP4F7NB73vzaCSBmNPUyRSe13v_scZ097FMY604 (abgerufen am: 31.05.2022), online.
- [24] Shawn Hymel. How to Build OpenOCD and Picotool for the Raspberry Pi Pico on Windows. <https://shawnhymel.com/2168/how-to-build-openocd-and-picotool-for-the-raspberry-pi-pico-on-windows/> (abgerufen am: 22.03.2022), April 2021.
- [25] Shawn Hymel. How to Set Up Raspberry Pi Pico C/C++ Toolchain on Windows with VS Code. <https://shawnhymel.com/2096/how-to-set-up-raspberry-pi-pico-c-c-toolchain-on-windows-with-vs-code/> (abgerufen am: 22.03.2022), April 2021.
- [26] Shawn Hymel. Raspberry Pi Pico and RP2040 Getting Started with C/C++ Part 1: VS Code and Blink. <https://www.digikey.ch/en/maker/projects/raspberry-pi-pico-and-rp2040-cc-part-1-blink-and-vs-code/7102fb8bca95452e9df6150f39ae8422> (abgerufen am: 11.05.2022), online.
- [27] Shawn Hymel. Raspberry Pi Pico and RP2040 Getting Started with C/C++ Part 2: Debugging with VS Code. <https://www.digikey.ch/en/maker/projects/raspberry-pi-pico-and-rp2040-cc-part-2-debugging-with-vs-code/470abc7efb07432b82c95f6f67f184c0> (abgerufen am: 11.05.2022), online.
- [28] Skyworks Solutions, Inc. AS213-92, AS213-92LF. https://www.mouser.ch/datasheet/2/472/SKYSS01666_1-2515045.pdf (abgerufen am: 31.05.2022), Dezember 2008.
- [29] The Things Industries. LoRaWAN Security. <https://www.thethingsnetwork.org/docs/lorawan/security/> (abgerufen am: 31.05.2022), online.
- [30] Winbond Electronics Corporation. W25Q128JV-DTR. https://www.winbond.com/resource-files/w25q128jv_dtr%20revc%2003272018%20plus.pdf (abgerufen am: 05.04.2022), März 2018.

A | Aufgabenstellung tinyLoRa

Lucerne University of
Applied Sciences and Arts

**HOCHSCHULE
LUZERN**

Technik & Architektur

Horw, 21. Februar 2022
Seite 1/2

Bachelor Thesis im Studiengang Elektrotechnik und Informationstechnologie

Aufgabe für Herrn Julian Staffelbach

tinyLoRa

Fachliche Schwerpunkte

Nachrichtentechnik / Signal Processing, Technische Informatik (Embedded Systems)

Einleitung

Der SX126x Transceiver von Semtech ist ein neuer Sub-GHz LoRa Transceiver, welcher für IoT Anwendungen gut geeignet ist. Mit diesem soll ein kleines und vielseitiges Board erstellt werden, welches für LoRaWAN und IoT Anwendungen gebraucht werden kann.

Aufgabenstellung

Analog zum tinyK22 soll basierend auf dem Semtech SX1261 oder SX1262 ein kleines und vielseitiges Board mit einem Mikrocontroller erstellt werden, welches batteriebetriebenen Sensoraufgaben übernehmen kann. Als Ausgangslage stehen Referenz Designs von Semtech zur Verfügung. Die Schaltung sollte möglichst klein, batteriebetrieben und kostengünstig sein. Die Performance und Anpassung der Antenne sollen ausgemessen werden. Für die Software steht eine Portierung des LoRa Stacks für den LPC55S16 zur Verfügung, welche angepasst werden muss. Das Ziel ist ein funktionierendes Funktionsmuster aus Hard- und Software unter Verwendung von FreeRTOS. Da momentan Halbleiter schwer erhältlich sind, soll ein flexibles und änderbares Konzept realisiert werden, das mit verschiedenen MCUs oder Boards funktioniert. Da das Projekt offen und auch in studentischen Projekten Verwendung finden soll, soll KiCAD als EDA/Design-Tool verwendet werden. Das Ziel ist ein Demonstrator mit mehreren Knoten, welche über LoRaWAN kommunizieren: dazu läuft auch eine Master Arbeit, welche einbezogen werden kann. Die konkreten Spezifikationen sind zu Beginn der Arbeit zu vereinbaren und festzulegen.

Termine

Start der Arbeit:	Montag, 21.2.2022
Zwischenpräsentation:	Zu vereinbaren im Zeitraum 11.4. – 6.5.2022
Abgabe Schlussbericht:	Freitag, 10. Juni 2022, vor 16:00 im D311
Abgabe Digitale Doku:	Gemäss separater Anweisung der Studiengangleitung
Abschlusspräsentation:	Zu vereinbaren im Zeitraum 13.6. – 1.7.2022
Diplomausstellung:	Freitag, 8. Juli 2022 (Teilnahme obligatorisch!)

FH Zentralschweiz

Horw, 21.2.2022
Seite 2/2
Diplomarbeit im Fachbereich
Elektrotechnik und Informationstechnologie

Dokumentation

Der gebundene Schlussbericht enthält keine Selbständigkeitserklärung und ist in einfacher Ausführung zu erstellen. Er enthält zudem zwingend

- einen sehr kurzen, englischen Abstract.
- Ein Titelblatt, gleich hinter dem Deckblatt, gemäss Weisungen der Studiengangleitung
- Eine SD-Hülle, innen, auf der Rückseite des Berichtes für den Betreuer

Alle Exemplare des Schlussberichtes müssen komplett und termingerecht gemäss Angaben der Studiengangleitung abgegeben werden. Zusätzlich muss eine SD-Speicherkarte mit dem Bericht (inkl. Anhänge), dem Poster und den Präsentationen, Messdaten, Programmen, Auswertungen, usw. unmittelbar nach der Präsentation abgegeben werden.

Die gesamte Dokumentation ist zudem gemäss Anweisungen der Studiengangleitung elektronisch auf einen Server zu laden. Sämtliche abzugebende Teile der Dokumentation sind Bestandteile der Beurteilung.

Fachliteratur/Web-Links/Hilfsmittel

SX126x Transceiver, EVK Boards nach Bedarf, LoRaWAN Stack/Software.

KiCAD (<https://www.kicad.org/>)

Geheimhaltungsstufe:

Öffentlich im Internet

Verantwortlicher Dozent/Betreuungsteam, Industriepartner

Dozent Prof. Erich Styger erich.styger@hslu.ch

Industriepartner IET

Experte

Dr. Christian Vetterli
christian.vetterli@mt.com Tel. +41 79 260 15 16

Hochschule Luzern
Technik & Architektur

Prof. Erich Styger

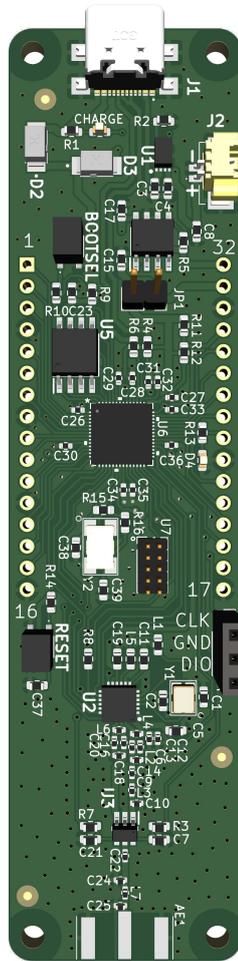
C | Bill of Materials

tinyLoRa		BOM				v1.01 06.06.2022	
RefDes	MPN	Geometry	Value	Qty	Description	Manufacturer	
IC							
U1	MCP73831-5ACI MC	-	-	1	Battery Management Li-Ion/Li-Poly Chrg Mgmt 4.5V Vreg out	Microchip	
U2	SK1261M1TRT	-	-	1	RF Transceiver LoRa, Low Power sub-GHz RF Transceiver	SEMTECH	
U3	AS213-92LF	-	-	1	SKYWORKS SOLUTIONS - AS213-92LF - IC, SWITCH, RF, SPDT, 0.1-3.0GHZ	Skyworks	
U4	AP2112M-3.3TRG1	-	-	1	LDO Voltage Regulators LDO CMOS HiCurr	Diodes Inc.	
U5	W25Q128VISM	-	-	1	NOR-Flash spiFlash, 3V, 128M-bit, 4Kb Uniform Sector, DTR	Winbond	
U6	RPI Chip RP2040-7500	-	-	1	Dual ARM Cortex M0+ Microcontroller	Raspberry Pi	
Resistors							
R1	-	0603_1608	470	1	-	-	
R2	-	0603_1608	5k1	1	-	-	
R3	-	0603_1608	100	1	-	-	
R4	CMP0603AFX-27ROELF	0603_1608	27	1	Thick Film Resistors ResHighPowerA 0603 27R 1% 1/4W TC100	Bourns	
R5	-	0603_1608	10k	1	-	-	
R6	CMP0603AFX-27ROELF	0603_1608	27	1	Thick Film Resistors ResHighPowerA 0603 27R 1% 1/4W TC100	Bourns	
R7	-	0603_1608	100	1	-	-	
R8	-	0603_1608	DNP	1	-	-	
R9	-	0603_1608	DNP	1	-	-	
R10	-	0603_1608	1k	1	-	-	
R11	ERJ-UP3F2000V	0603_1608	200	1	Thick Film Resistors 0603 0.25W 1% 200ohm AEC-Q200	Panasonic	
R12	ERJ-U03F1R00V	0603_1608	1	1	Thick Film Resistors 0603 1% 1ohm AEC-Q200 Anti-Sulfur	Panasonic	
R13	-	0603_1608	390	1	-	-	
R14	-	0603_1608	10k	1	-	-	
R15	-	0603_1608	DNP	1	-	-	
R16	-	0603_1608	1k	1	-	-	
Capacitors							
C1	-	0603_1608	DNP	1	-	-	
C2	-	0603_1608	DNP	1	-	-	
C3	ZRB18AR6YA475KE05L	0603_1608	4u7	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C4	ZRB18AR6YA475KE05L	0603_1608	4u7	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C5	-	0603_1608	10n	1	-	-	
C6	GRM1555C1H3R0BA01D	0402_1005	3p	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C7	-	0603_1608	1n	1	-	-	
C8	-	0603_1608	1u	1	-	-	
C9	GRM1555C1H120JA01J	0402_1005	12p	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C10	UVK105CH2R2JW-F	0402_1005	2p2	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Taiyo Yuden	
C11	GC188R71C474KA12D	0603_1608	470n	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C12	GRM155C71H473JE19D	0402_1005	47n	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C13	GCM1555C1H470JA16J	0402_1005	47p	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C14	C0402C569J5GACTU	0402_1005	5p6	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	KEMET	
C15	-	0603_1608	1u	1	-	-	
C16	GRM1555C1H18WA01D	0402_1005	1p8	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C17	-	0603_1608	1u	1	-	-	
C18	GRM1555C1H1R0BA01D	0402_1005	1p	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C19	-	0603_1608	100n	1	-	-	
C20	GRM1555C1H18WA01D	0402_1005	1p8	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C21	-	0603_1608	1n	1	-	-	
C22	GRM1555C1H220JA01D	0402_1005	22p	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C23	-	0603_1608	100n	1	-	-	
C24	UVK105CG3R3JW-F	0402_1005	3p3	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Taiyo Yuden	
C25	UVK105CG3R3JW-F	0402_1005	3p3	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Taiyo Yuden	
C26	CGA2B1X7R1C104K050BC	0402_1005	100n	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT CGA	TDK	
C27	C1005X5R1V225K050BE	0402_1005	2u2	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	TDK	
C28	CGA2B1X7R1C104K050BC	0402_1005	100n	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT CGA	TDK	
C29	CGA2B1X7R1C104K050BC	0402_1005	100n	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT CGA	TDK	
C30	CGA2B1X7R1C104K050BC	0402_1005	100n	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT CGA	TDK	
C31	C1005X5R1V225K050BE	0402_1005	2u2	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	TDK	
C32	C1005X5R1V225K050BE	0402_1005	2u2	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	TDK	
C33	CGA2B1X7R1C104K050BC	0402_1005	100n	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT CGA	TDK	
C34	CGA2B1X7R1C104K050BC	0402_1005	100n	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT CGA	TDK	
C35	CGA2B1X7R1C104K050BC	0402_1005	100n	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT CGA	TDK	
C36	CGA2B1X7R1C104K050BC	0402_1005	100n	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT CGA	TDK	
C37	-	0603_1608	100n	1	-	-	
C38	GRM1885C2E270JW07D	0603_1608	27p	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
C39	GRM1885C2E270JW07D	0603_1608	27p	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT	Murata Electronics	
Inductors							
L1	LQG15WZ47NH02D	0402_1005	47n	1	Fixed Inductors Fixed IND 47nH 190mA	Murata Electronics	
L2	LQW15AN2N7C00D	0402_1005	2n7	1	Fixed Inductors 2.7 NH +-2NH	Murata Electronics	
L3	LQP15MN9N1B02D	0402_1005	9n1	1	Fixed Inductors	Murata Electronics	
L4	LQG15WZ47NH02D	0402_1005	47n	1	Fixed Inductors Fixed IND 47nH 190mA	Murata Electronics	
L5	LQM18DN150M70L	0603_1608	15u	1	Fixed Inductors	Murata Electronics	
L6	LQW15AN18NH00D	0402_1005	18n	1	Fixed Inductors 18 NH 3%	Murata Electronics	
L7	LQP15MN9N1B02D	0402_1005	9n1	1	Fixed Inductors	Murata Electronics	
Diodes & LEDs							
D1	XZMDK53W-8	0603_1608	LED Red	1	LED RED CLEAR CHIP SMD	SunLED	
D2	PMEG3050EP.115	2.7 x 4 mm	-	1	SMT Schottky Diode, 30V 5A, 2-Pin SOD-128	Nexperia	
D3	PMEG3050EP.115	2.7 x 4 mm	-	1	SMT Schottky Diode, 30V 5A, 2-Pin SOD-128	Nexperia	
D4	XZV653W-8	0603_1608	LED Green	1	LED GREEN CLEAR CHIP SMD	SunLED	
Crystals							
Y1	LFXTAL060258Reel	2.7 x 3.4 mm	32.0 MHz	1	Crystals 32MHz 10pF -20C 70C	IQD	
Y2	406135D12M00000	3.5 x 6 mm	12.0 MHz	1	Crystals 12MHz 30ppm 18pF -40C + 85C	CIS Electronic Components	
Connectors, Buttons & other							
AE1	CON5MA020.042-G	14.2 x 9.5 x 7.9 mm	-	1	RF Connectors SMA Connector Straight Jack (Female), PCB Edge Mount	Linx Technologies	
J1	USB4110-GF-A	10.2 x 7.7 mm	USB-C	1	CONN USB 2.0 TYPE-C R/A SMT	GCT	
J2	53261-0271	1.25 mm	1X2	1	Headers & Wire Housings 1.25MM HDR 2 CKT Right-Angle	Molex	
J3	-	2.54 mm	2x16	1	Pin-Header male 2.54 mm	-	
J4	-	2.54 mm	1x3	1	Pin-Header female 2.54 mm	-	
JP1	-	2.54 mm	1x2	1	Pin-Header male 2.54 mm	-	
U7	-	1.27 mm	2x5	1	Pin-Header male 1.27 mm	-	
SW1	434123025826	2.8 x 5.2 mm	Bootsel	1	WS-TASV J-Bend SMT Tact Switch	Würth Elektronik	
SW2	434123025826	2.8 x 5.2 mm	Reset	1	WS-TASV J-Bend SMT Tact Switch	Würth Elektronik	

D | Pinout tinyLoRa

UART 0 TX	I2C 0 SDA	SPI 0 MISO	GPIO 0	1
UART 0 RX	I2C 0 SCL	SPI 0 CSn	GPIO 1	2
	I2C 1 SDA	SPI 0 CLK	GPIO 2	3
			GND	4
	I2C 1 SCL	SPI 0 MOSI	GPIO 3	5
UART 1 TX	I2C 0 SDA	SPI 0 MISO	GPIO 4	6
UART 1 RX	I2C 0 SCL	SPI 0 CSn	GPIO 5	7
			GND	8
	I2C 1 SDA	SPI 0 CLK	GPIO 6	9
	I2C 1 SCL	SPI 0 MOSI	GPIO 7	10
UART 1 TX	I2C 0 SDA		GPIO 8	11
			GND	12
UART 1 RX	I2C 0 SCL		GPIO 9	13
	I2C 1 SDA		GPIO 10	14
	I2C 1 SCL		GPIO 11	15
			GND	16

Power
Ground
GPIO, PIO, PWM
ADC
SPI
I2C
UART



32	V_SYS		
31	3.3 V		
30	GND		
29	ADC V_REF		
28	GPIO 29	ADC 3	
27	GPIO 28	ADC 2	
26	GPIO 27	ADC 1	I2C 1 SCL
25	GPIO 26	ADC 0	I2C 1 SDA
24	GPIO 24		
23	GND		
22	GPIO 23	SPI 0 MOSI	I2C 1 SCL
21	GPIO 22	SPI 0 CLK	I2C 1 SDA
20	GPIO 21	SPI 0 CSn	I2C 0 SCL
19	GPIO 20	SPI 0 MISO	I2C 0 SDA
18	RESET		
17	GND		

E | Elektronischer Anhang

Inhalte des elektronischen Anhangs:

