



BUSSYSTEME UND SENSORIK

ENTWICKLUNG EINES HUBS ZUR ERFASSUNG UND  
GRAPHISCHEN DARSTELLUNG VON SENSORDATEN

WINTERSEMESTER 2023/2024

Ausarbeitung von:

Lasse Kelling  
2590639

Fabian Schmalenbach  
2514071

Abgabedatum: 24.01.2024

Prüfer: Prof. Dr. R. Fitz

# Inhaltsverzeichnis

<b>1</b>	<b>Projektbeschreibung</b>	<b>1</b>
1.1	Anforderungen . . . . .	1
<b>2</b>	<b>Systembeschreibung</b>	<b>2</b>
2.1	Systemaufbau . . . . .	2
2.2	Sensorik . . . . .	2
2.2.1	BME280 . . . . .	3
2.2.2	MHZ19C . . . . .	3
2.2.3	DCF77 . . . . .	3
2.2.3.1	MeteoTime . . . . .	4
2.3	Kommunikation . . . . .	7
2.3.1	Funkstrecke . . . . .	7
2.3.2	Datenpakete . . . . .	7
2.3.3	Übertragungsfehler . . . . .	9
<b>3</b>	<b>Modulbeschreibung</b>	<b>10</b>
3.1	Sensormodul extern . . . . .	10
3.1.1	Platinenentwurf . . . . .	10
3.1.1.1	Schaltplan . . . . .	11
3.1.1.2	Platine . . . . .	11
3.2	Modul intern . . . . .	12
3.2.1	Sensormodul . . . . .	12
3.2.2	Grafikmodul . . . . .	12
3.2.3	Platinenentwurf . . . . .	17
3.2.3.1	Schaltplan . . . . .	17
3.2.3.2	Platine . . . . .	17
<b>4</b>	<b>Funktionstest</b>	<b>17</b>
4.1	Test der Sensorik . . . . .	17
4.2	Test des Grafikmoduls . . . . .	18
<b>5</b>	<b>Fazit</b>	<b>19</b>
<b>A</b>	<b>Ablaufdiagramm externes Modul</b>	<b>21</b>
<b>B</b>	<b>Schaltplan externes Modul</b>	<b>24</b>
<b>C</b>	<b>Platine externes Modul</b>	<b>25</b>
C.1	2D Ansicht . . . . .	25
C.2	3D Ansicht . . . . .	26
<b>D</b>	<b>Ablaufdiagramm internes Sensormodul</b>	<b>28</b>
<b>E</b>	<b>Schaltplan internes Modul</b>	<b>31</b>
<b>F</b>	<b>Platine internes Modul</b>	<b>32</b>
F.1	2D Ansicht . . . . .	32
F.2	3D Ansicht . . . . .	33

# 1 Projektbeschreibung

Ziel des Projekts ist die Entwicklung eines Sensorhubs, der Sensordaten erfasst und auf einem Display darstellt. Bei den Sensoren handelt es sich in erster Linie um Umweltdaten, die aktuelle Parameter der Umgebung erfassen. Das Projekt kann daher grob mit einer Wetterstation verglichen werden. Zusätzlich zu den Sensordaten kann der Sensorhub die aktuelle Zeit und eine Wettervorhersage über das DCF77-Signal empfangen.

## 1.1 Anforderungen

Es wurden keine verpflichtenden Anforderungen gestellt, das Projekt soll thematisch aber zum Modul "Bussysteme und Sensorik" passen. Daraus lassen sich für das spezifische Projekt Anforderungen stellen bzw. ableiten:

- Verwendung eines oder mehrerer Bussysteme zur Kommunikation zwischen Mikrocontrollern
- Nutzung diverser Sensoren mit unterschiedlichen Anbindungen für Vielfältigkeit
- Analog zu herkömmlichen Wetterstationen, soll diese ebenfalls über einen Außensensor verfügen
- Die Wetterstation soll über eine Wettervorhersage verfügen
- Der Sensorhub soll skalier- und erweiterbar sein

Aus diesen Anforderungen lassen sich direkt Vorgaben für das Projekt ableiten:

- Nutzung mehrerer Mikrocontroller, die miteinander über ein Bussystem kommunizieren
- Verwendung digitaler Sensoren, die Standardprotokolle wie I2C, SPI oder UART unterstützen
- Entwicklung eines Außensensors, der drahtlos mit dem Sensorhub kommunizieren kann
- Anbindung des Sensorhubs ans Internet oder Empfang von Wettervorhersagen via Funk (DCF77)
- Nutzung ausreichend leistungsstarker Mikrocontroller, die genügend Leistungs- und Peripheriereserven haben, um weitere Geräte anzubinden

## 2 Systembeschreibung

### 2.1 Systemaufbau

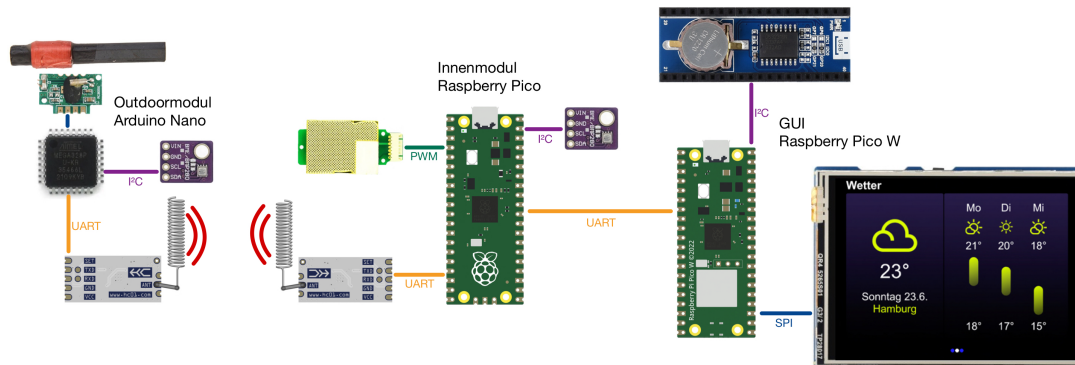


Abbildung 1: Übersicht der Systemkomponenten

Die obige Abbildung 1 zeigt abstrakt alle Komponenten des Systems. Die jeweiligen Komponenten über dem Mikrocontroller Symbol zeigen jeweils die angeschlossenen Sensoren bzw. Bildschirme. Ebenfalls ist grob die Kommunikation zwischen den Mikrocontrollern erkennbar.

Ganz links ist der Außensensor dargestellt, an den ein Sensor zur Messung von Temperatur, Luftfeuchtigkeit und Luftdruck angeschlossen ist. Außerdem verfügt dieser über eine Antenne, um das DCF77-Signal zu empfangen, welches die aktuelle Zeit und eine Wettervorhersage beinhaltet (siehe Abschnitt 2.2.3). Der Sensor sendet die empfangenen Daten zyklisch per 433 MHz Sender an den Sensorhub.

Der Sensorhub besteht aus zwei Mikrocontrollern, die über eine serielle Schnittstelle (UART) miteinander kommunizieren. Der in Abbildung 1 mittlere Mikrocontroller empfängt die Daten des Außensensors und leitet diese weiter an den verbundenen Mikrocontroller. Da die Daten zur Wettervorhersage verschlüsselt sind und der Außensensor möglichst wenig Energie verbrauchen soll, müssen die Daten vom Mikrocontroller entschlüsselt werden. Dazu werden die entsprechenden Datenpakete abgefangen, dekodiert und anschließend weitergesendet. Der Mikrocontroller verfügt außerdem wie der Außensensor über einen Sensor zur Messung von Temperatur, Luftfeuchtigkeit und Luftdruck (wobei der Luftdruck im Innenraum nicht gemessen wird). Zusätzlich ist ein CO2 Sensor verbaut, der die Konzentration im Raum misst.

In der Übersicht ganz rechts ist der Mikrocontroller, der alle Daten empfängt und auf einem Touchdisplay darstellt. Da der Sensor über ein WLAN-Modul verfügt, wäre theoretisch zusätzlich die Übertragung der Daten per WLAN an einen Server o.ä. möglich, der alle Daten speichert und diese anderen Geräten zur Verfügung stellt.

### 2.2 Sensorik

Zur Erfassung der Umweltparameter werden aktuell zwei verschiedene Sensoren eingesetzt. Der BME280 erfasst Lufttemperatur, Luftfeuchtigkeit und Luftdruck. Der MHZ19C wird zur Erfassung der CO2 Konzentration im Raum eingesetzt. In den folgenden beiden Abschnitten werden die Sensoren beschrieben.

Das Außenmodul verfügt außerdem über eine 77,5 kHz Empfangsantenne, um das Zeitzeichensignal DCF77 zu empfangen. Daraus lässt sich die aktuelle Zeit ablesen, außerdem wird eine

Wettervorhersage mit übertragen, die ausgewertet wird.

### 2.2.1 BME280

Beim BME280 handelt es sich um einen effizienten Sensor von Bosch, der zur Erfassung von Temperatur, Luftfeuchtigkeit und Luftdruck eingesetzt wird. Der Sensor verfügt über ein I2C und SPI Interface. Beim zyklischen Auslesen aller Sensordaten mit 1 Hz liegt die Stromaufnahme laut Datenblatt bei  $3,6 \mu\text{A}$ , weshalb sich der Sensor ideal für die Anwendung in Sensormodulen mit Batteriebetrieb eignet.

Der Sensor verfügt über die folgenden Messbereiche:

- Temperatur:  $-40^{\circ}\text{C} - +85^{\circ}\text{C}$  ( $\pm 0,5^{\circ}\text{C}$ )
- Luftfeuchtigkeit: 0 - 100% rel. Feuchtigkeit ( $\pm 3\%$ )
- Druck: 250 - 1250 hPa ( $\pm 1 \text{ hPa}$ )

Mit diesen Spezifikationen eignet sich der Sensor für die Anwendung im Innen- und Außenbereich. Für das Projekt wird ein Sensorshield verwendet, welches nur den I2C Bus herausführt. Die Kommunikation mit dem Sensor erfolgt daher über I2C.

### 2.2.2 MHZ19C

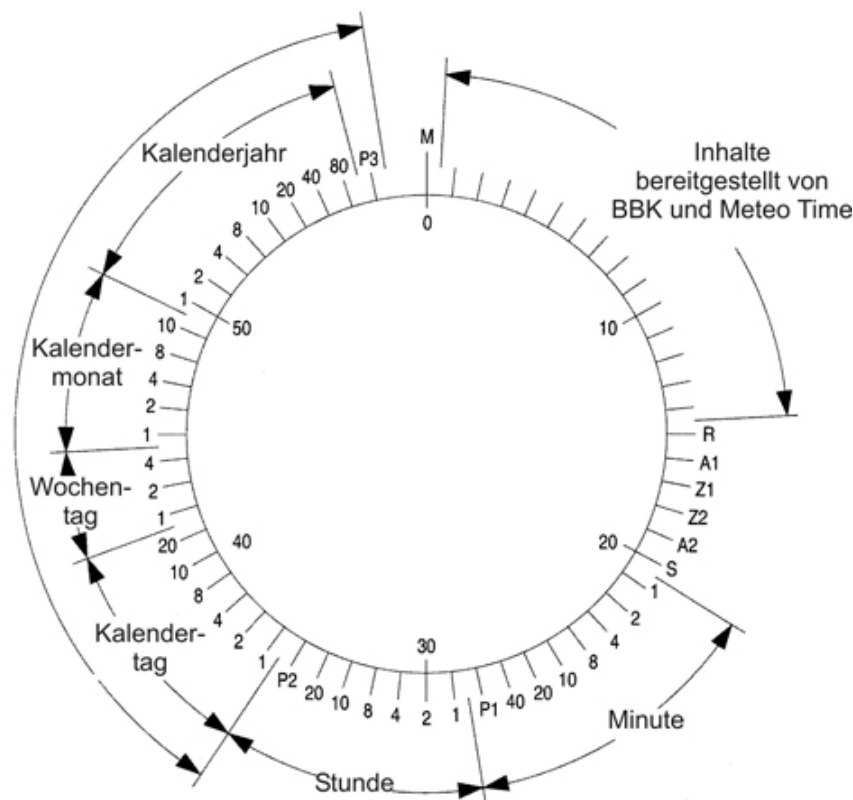
Der MHZ-19C ist ein Infrarot CO<sub>2</sub>-Sensor, der die CO<sub>2</sub> Konzentration mittels nicht-dispersiver Infrarot-Spektroskopie misst. Dabei wird zyklisch mit einer Infrarotlampe und einem Photosensor die CO<sub>2</sub>-Konzentration anhand der Reflexion des Lichts durch CO<sub>2</sub> Partikel gemessen. Der Sensor verfügt über einen Messbereich von 400-5000 ppm, wobei die Genauigkeit  $\pm 40\text{ppm} + 5\%$  des Messwerts beträgt. Da NDIR Sensoren zum Messwertdrift neigen, verfügt der Sensor über eine automatische Kalibrierung, die alle 24 Stunden erfolgt. Der Sensor ist daher für den Dauerbetrieb ausgelegt und sollte dauerhaft aktiv sein. Die Aufheizzeit des Sensors beträgt eine Minute.

Der Sensor ist über eine UART-Schnittstelle konfigurier- und auslesbar, außerdem verfügt er über einen PWM Ausgang.

### 2.2.3 DCF77

Bei DCF77 handelt es sich um einen Zeitzeichensender in der Nähe von Frankfurt am Main, welcher mit einer Trägerfrequenz von 77,5 kHz ein Zeitsignal überträgt. Das Signal ist europaweit empfangbar und wird von den meisten Funkuhren als Referenzsignal verwendet.

Die Datenübertragung erfolgt durch einfache Amplitudenmodulation, indem die Amplitude für eine Dauer von 100 ms (Bit 0) oder 200 ms (Bit 1) auf 25% der Ausgangsamplitude abgesenkt wird. Damit wird eine Bitrate von 1 Bit pro Sekunde erreicht.



**Abbildung 2:** Kodierungsschema der DCF77 Zeitinformationen

<https://www.ptb.de/cms/ptb/fachabteilungen/abt4/fb-44/ag-442/verbreitung-der-gesetzlichen-zeit/dcf77/zeitcode.html>

Abbildung 2 zeigt die Kodierung aller Datenbits innerhalb einer Minute. Daraus ist erkennbar, dass ein Datenpaket 59 Bits lang ist und die Übertragung eine Minute dauert. Ein Paket beginnt zu jeder neuen Minute. In der letzten Sekunde einer Minute erfolgt keine Absenkung des Trägers, sodass anhand dessen das Ende eines Pakets abgeleitet werden kann.

Die neue Minute beginnt mit dem nullten Bit, das immer den Wert null hat. Darauf folgen 14 Bits mit verschlüsselten Wetterinformationen der Firma MeteoTime. Diese werden im nächsten Abschnitt weiter beschrieben.

Bit 15 (R) ist ein Rufbit, dass zur Alarmierung der PTB Mitarbeiter dient, wenn Unregelmäßigkeiten vorliegen.

Bit 16 (A1) zeigt an, dass am Ende der Stunde ein Wechsel von Sommer- zu Winterzeit stattfindet.

Bit 17 (Z1) ist eine Flag für die Sommerzeit.

Bit 18 (Z2) ist eine Flag für die Winterzeit.

Bit 19 (A2) zeigt an, dass am Ende der Stunde eine Schaltsekunde eingefügt wird.

Bit 20 (S) markiert den Beginn der Zeitinformationen und hat den Wert eins.

Wie schon aus der Abbildung hervorgeht, werden in den restlichen Sekunden Informationen zu Zeit und Datum übertragen, Minute, Stunde und Datum haben jeweils ein Paritätsbit eingefügt, um Fehler erkennen zu können. Eine Fehlerkorrektur ist damit allerdings nicht möglich.

### 2.2.3.1 MeteoTime

MeteoTime versendet pro Minute 14 Bits mit Wetterinformationen, die von lizenzierten Geräten entschlüsselt- und verwendet werden können. Ein Informationspaket besteht aus 42 Bits und

benötigt für die Übertragung somit drei Minuten. Ein Tag ist in fünf Abschnitte unterteilt, in denen unterschiedliche Informationen übertragen werden. Da die Wettervorhersage 90 Regionen abdeckt, wird jede Vorhersage nur täglich einmal übertragen.

Es gilt folgende Aufteilung (MEZ, bei MESZ eine Stunde später):

23:00 Uhr - 04:59 Uhr: Aktueller Tag

05:00 Uhr - 10:59 Uhr: Folgender Tag

11:00 Uhr - 16:59 Uhr: Darauffolgender Tag

17:00 Uhr - 19:59 Uhr: Darauffolgender Tag

20:00 Uhr - 22:59 Uhr: Zusatzregionen mit 2-Tages Prognose.

Von den 90 Regionen erhalten 60 die viertägige Vorhersage, die restlichen 30 Regionen erhalten nur eine 2-Tages Vorhersage.

Innerhalb eines sechsständigen Übertragungszeitraums werden für jede Region nacheinander die Höchst- und Tiefstwerte übertragen.

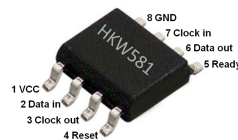
Die Wettervorhersagen decken große Regionen ab, weshalb diese etwas ungenau sind.

Für Hamburg ist beispielsweise die Region Bremerhaven zugeteilt, der Bereich deckt alles vom westlichen Schleswig-Holstein bis zum Nordwesten der Niederlande ab.

Um im Fall eines Lesefehlers trotzdem eine Wettervorhersage anzeigen zu können, werden Ausweichzonen genutzt, dessen Wetterdaten stattdessen angezeigt wird. Schlägt der Empfang für die Region Bremerhaven fehl, wird ausweichend die Vorhersage für Hannover übernommen. Werden auch diese Daten nicht korrekt empfangen, wird die dritte Ausweichregion Rostock genutzt.

Für die Entschlüsselung der Wetterinformationen ist ein Dekodier-IC notwendig, welches vermutlich aus rückgekoppelten Schieberegistern besteht, um die Daten zu dekodieren.

Beim IC handelt es sich um das Modell HKW581 der Firma HKW, die auch Betreiber des MeteoTime Dienstes ist. Der IC ist nur in einem MSOP8 Format erhältlich (SMD).



**Abbildung 3:** Dekodier-IC zum Entschlüsseln der Wettervorhersagen

Der IC verfügt über acht Pins, davon sind neben den Pins zur Spannungsversorgung und Daten Ein- und Ausgabe auch ein Clock IN und Clock OUT Pin vorhanden, damit die Bits taktsynchron ein- bzw. ausgelesen werden. Beim Einlesen der Daten folgt das Clock OUT Signal dem Clock IN Signal, um anzuzeigen, dass die Erfassung des Bits erfolgreich war. Nachdem 82 Bits übertragen wurden, wird das Clock OUT Signal invertiert, bis alle Daten vom IC ausgelesen wurden. Neben einem Resetpin gibt es außerdem einen Ready-Pin, der anzeigt, dass der IC betriebsbereit ist. Anscheinend braucht der IC eine Aufwärmphase.

Als Eingabedaten benötigt der IC einen 82 Bit langen Datenstream, der sich wie folgt zusammensetzt:

- Meteopakete 1 (14 Bit)
- Meteopakete 2 (14 Bit)

- Meteopakete 3 (14 Bit)
- Minute des ersten Pakets (aufgefüllt von 7 auf 8 Bit)
- Stunde des ersten Pakets (aufgefüllt von 6 auf 8 Bit)
- Kalendertag (aufgefüllt von 6 auf 8 Bit)
- Monat (5 Bit)
- Wochentag (3 Bit)
- Jahr (8 Bit)

Als Ausgabe erhält man einen 22 Bit langen Datenstream mit den Wetterinformationen. Je nachdem, ob gerade ein Paket mit Höchst- oder Tiefstwerten gesendet wurde, ergeben sich unterschiedliche Informationen:

Höchstwerte:

- Wetter Tag (4 Bit)
- Wetter Nacht (4 Bit)
- Schweres Wetter (4 Bit)
- Niederschlagswahrscheinlichkeit (3 Bit)
- Wetteranomalie Tag (1 Bit)
- Temperatur Tag (6 Bit)

Tiefstwerte:

- Wetter Tag (4 Bit)
- Wetter Nacht (4 Bit)
- Windrichtung (4 Bit)
- Windstärke (3 Bit)
- Wetteranomalie Nacht (1 Bit)
- Temperatur Nacht (6 Bit)

Die ausgegebenen Zahlenwerte sind wiederum einer Bezeichnung zugeordnet (z.B. Wetter Tag = 3 entspricht vorwiegend bewölkt).



## 2.3 Kommunikation

Die Kommunikation zwischen den Mikrocontrollern erfolgt sowohl kabelgebunden, als auch drahtlos. Für die kabelgebundene Kommunikation wird UART mit einer Baudrate von 115200 bd genutzt. Die Drahtloskommunikation erfolgt über einen 433 MHz Transmitter und Receiver, die jeweils im Halbduplexverfahren wie eine UART Schnittstelle genutzt werden können.

### 2.3.1 Funkstrecke

Bei den Funkmodulen handelt es sich um "HC-12 Wireless RF communication module V2.4" Module. Diese versprechen eine Übertragungsdistanz von bis zu 1800 m und eine einfache Ansteuerung über UART.

Die Module können jeweils sowohl Daten senden. Die Sendeleistung beträgt bis zu 100 mW und es können 100 verschiedene, einstellbare Kanäle genutzt werden (433,4 MHz bis 473 MHz). Die Baudrate ist zwischen 1200 bps und 115200 bd wählbar.

Aktuell wird eine Baudrate von 9600 bd verwendet, um ein gutes Mittel zwischen Distanz und Geschwindigkeit zu erzielen.

Aufgrund der geltenden Bestimmungen in Deutschland darf nur mit maximal 10 mW gesendet werden, auch ist der wählbare Frequenzbereich teilweise nicht legal. Es können nur die Kanäle 1 bis 4 genutzt werden.

### 2.3.2 Datenpakete

Je nach übertragener Information werden unterschiedliche Datenpakete verwendet, die zur Identifikation jeweils mit einem 4-Byte langen Identifier ausgestattet sind.

Aktuell gibt es fünf verschiedene Identifier:

- 'IBME' Internal BME
- 'EBME' External BME
- 'CO2.' Co2 Konzentration
- 'TIME' Aktuelle Zeit
- 'MTEO' Wettervorhersage

Die Datenpakete haben unterschiedliche Längen und sind in Form von Structs in C definiert. Zum Senden der Daten werden diese dann in Char-Arrays umgewandelt.

Da die Structs unterschiedliche Datentypen beherbergen, fügt der Compiler normalerweise Padding Bytes ein, um gleichmäßige Adressabstände zu gewährleisten. Für die Verarbeitung und Übertragung der Datenpakete ist dies aber problematisch, weil beispielsweise die zu sendenden Datenpakete größer als nötig sind. Aus diesem Grund werden die Structs mit dem Attribut packed versehen, damit auf Padding Bytes verzichtet wird. Alle Datenpakete sind im folgenden dargestellt:

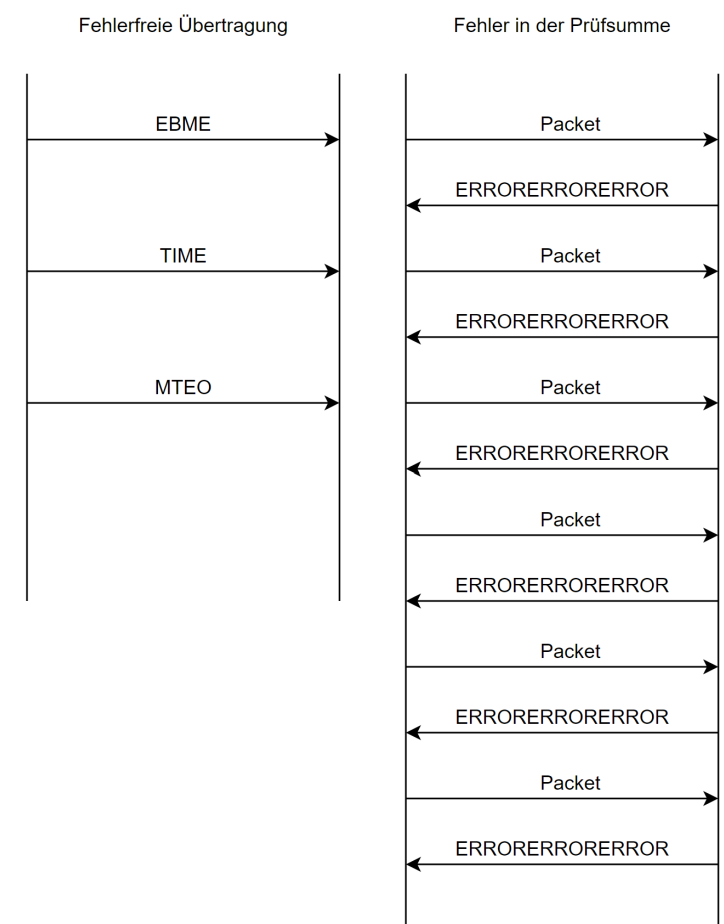
```
1 // Aktuelle Zeit
2 typedef struct __attribute__((packed)){
3     const char dataType[4];
4     uint8_t hour;
5     uint8_t minute;
6     uint8_t second;
7     uint8_t year;
8     uint8_t month;
9     uint8_t day;
10    uint8_t weekDay;
11    uint8_t checksum;
12 } TimeStruct;
13
14 typedef union {
15     char buf[sizeof(TimeStruct)];
16     TimeStruct data;
17 } TimeSendBuf;
18
19 // Dekodierte Wetterdaten
20 typedef struct __attribute__((packed)){
21     const char dataType[4];
22     uint32_t meteoData;
23     uint8_t checksum;
24 } MeteoDecodedStruct;
25
26 typedef union {
27     char buf[sizeof(MeteoDecodedStruct)];
28     MeteoDecodedStruct data;
29 } MeteoDecodedSendBuf;
30
31 // Sensordaten vom BME280
32 typedef struct __attribute__((packed)){
33     const char dataType[4];
34     float temperature;
35     float humidity;
36     float pressure;
37     uint8_t checksum;
38 } BMEStruct;
39
40 typedef union {
41     char buf[sizeof(BMEStruct)];
42     BMEStruct data;
43 } BMESendBuf;
44
45 // CO2 Konzentration
46 typedef struct __attribute__((packed)){
47     const char dataType[4];
48     int concentration;
49 } CO2Struct;
50
51 typedef union {
52     char buf[sizeof(CO2Struct)];
53     CO2Struct data;
54 } CO2SendBuf;
```

Listing 1: Datenpakete

Wie man im obigen Listing sieht, haben alle Datenpakete zusätzlich eine CRC-8 Prüfsumme, die bei der drahtlosen Datenübertragung zur Prüfung des Pakets verwendet wird. Bei der kabelgebundenen Datenübertragung zwischen beiden Mikrocontrollern auf dem Innenmodul wird auf eine Prüfsumme verzichtet.

### 2.3.3 Übertragungsfehler

Da die Meteotime Pakete jeweils nur einmal gesendet werden und ein Paketverlust bei der 433 MHz Datenübertragung problematisch wäre, wird, wie schon erwähnt, eine Prüfsumme berechnet und übertragen und auf der Empfängerseite auf Richtigkeit überprüft. Ist die Prüfsumme fehlerhaft, sendet der Empfänger einen Retransmission-Befehl an den Sender. Im folgenden ist die sowohl die fehlerfreie Übertragung von Paketen als auch die Übertragung mit mehrfacher Retransmission dargestellt.



**Abbildung 4:** Kommunikationsablauf zwischen Innen- und Außenmodul ohne und mit Übertragungsfehler

Aus Abbildung 4 geht hervor, dass das Innenmodul den String "ERRORERRORERROR" an das Außenmodul sendet, wenn die Prüfsumme vom vorher gesendeten Paket fehlerhaft ist. Der String wurde so gewählt, damit eventuell zufällig empfangene einzelne Bits anderer Quellen ignoriert werden und bei schlechtem Empfang zumindest ein paar Bytes empfangen werden. Das Außenmodul reagiert mit einer Retransmission, wenn es fünf Bytes empfangen hat.

Die Zahl der Retransmissions ist auf fünf begrenzt, danach verwirft das Außenmodul das Paket. Ein Retransmission Paket muss vom Innenmodul innerhalb von drei Sekunden an das Außenmodul gesendet werden, andernfalls nimmt das Außenmodul an, dass die Übertragung erfolgreich war.

## 3 Modulbeschreibung

### 3.1 Sensormodul extern

Das externe Sensormodul soll im Außenbereich angebracht werden und möglichst effizient per Batterie betrieben werden können.

Wie schon aus den vorherigen Abschnitten hervorgeht, verfügt das Modul über einen BME280 für Temperatur, Luftfeuchtigkeit und Luftdruck und eine Antenne zum Empfang des DCF77 Signals. Die empfangenen Daten werden dann zyklisch via 433 MHz an den Sensorhub gesendet. Die DCF77 Antenne ist im Außenmodul untergebracht, weil das Signal sehr anfällig für Störungen durch elektrische Geräte oder Abschirmung durch Metalle ist, die vor allem im Innenbereich vorkommen. Im Außenbereich ist der Empfang besser.

Für den DCF77 Empfang wird ein Shield verwendet, das eine auf 77,5 kHz abgestimmte Antenne hat und das Signal filtert und in saubere TTL Pegel umwandelt. Im Mikrocontroller wird eine Bibliothek verwendet, die einen Interrupt mit Trigger auf steigende- und fallende Taktflanke verwendet, um die Dauer der Absenkung des Trägersignals zu messen. In der letzten Sekunde einer Minute findet keine Absenkung statt, anhand dessen wird das Ende eines Datenpakets ermittelt. Da in der letzten Sekunde einer Minute keine Absenkung stattfindet, kann anhand dessen das Ende eines Datenpakets ermittelt werden. Zum Ende jeder Minute wird das Paket auf Gültigkeit überprüft. Dazu werden zum einen die Paritätsbits auf den richtigen Wert überprüft, außerdem wird geprüft, ob die Flag für die Sommerzeit den gegenteiligen Wert der Flag für die Winterzeit hat. Ist die Überprüfung in Ordnung, wird das Datenpaket für die Zeit aus dem erhaltenen Zeitstempel zusammengesetzt und versendet.

Alle drei Minuten ist außerdem ein Meteotime Paket fertig und kann durch den Dechiffrier-IC entschlüsselt werden.

Ein Ablaufdiagramm zur Beschreibung der Software ist im Anhang in Abbildung 11 zu finden. Es wird zyklisch geprüft, ob ein Zeit, Meteo oder Sensorpaket versendet werden kann und ob die Übertragung fehlerfrei abgelaufen ist. Die Dekodierung der Wetterdaten erfolgt in der Klasse Meteo, im Dekodierprozess nacheinander jedes Bit des Pakets in den Decoder geschrieben und anschließend der Bitstream aus dem IC gelesen. War die Entschlüsselung der Daten fehlerhaft, gibt der Decoder den String "10000000000000000000100" aus.

Als Mikrocontroller wurde in ersten Tests zunächst ein ESP8266 verwendet. Da dieser einen relativ hohen Energieverbrauch von ca. 20 mA hat und zudem für den Anwendungszweck leistungstechnisch überdimensioniert ist, wurde als Alternative ein energiesparender MSP430G2553 gewählt, der bei 1 MHz Takt nur 230  $\mu$ A verbrauchen soll und sowohl einen UART, als auch I2C Bus hat. Damit wäre ein langfristiger Betrieb mit Batterie möglich gewesen. Aus Zeitgründen und wegen der fehlenden Portierungsmöglichkeit des ESP8266 Codes auf den MSP430 wurde stattdessen ein Arduino Nano gewählt, der einen ATmega328p Mikrocontroller als Basis verwendet. Modifiziert man einige Teile am Board, lässt sich der Stromverbrauch laut einiger Quellen auf ca. 5 mA drosseln.

#### 3.1.1 Platinentwurf

Für beide Module wird eine Platine angefertigt, um das Projekt auch in der Realität anwendbar zu machen. Das Außenmodul ist für den Batteriebetrieb konzipiert und daher mit einigen Aktivbauteilen ausgestattet. Die Bauteile wurden zusammen mit der Platine bei JLCPCB bestellt und dort bestückt.

### 3.1.1.1 Schaltplan

Der Schaltplan ist im Anhang in Abbildung 12 dargestellt. Zentraler Bestandteil ist dabei der Mikrocontroller. Für die Spannungsregulierung werden zwei Schaltregler verwendet, die 5 V und 3,3 V aus der Batteriespannung generieren. Beide Schaltregler arbeiten bis zu einer Minimalspannung von ca. 0,7 V, die angeschlossene Batterie kann also fast vollständig genutzt werden. Beide Schaltregler vertragen maximal 5,5 V am Eingang, weshalb als Spannungsquelle drei Alkaline Batterien in Serie dienen sollen, die eine Spannung von ca. 4,5 V bereitstellen. Die Anschlusskabel des Batteriefachs werden an die Platine gelötet, dafür wurden je 4x4 mm große Pads gewählt. Als Batterien werden aktuell Mono D Zellen verwendet, die das Modul aufgrund ihrer Größe und des Gewichts klobig erscheinen lassen, durch die hohe Ladungsmenge von bis zu 21 Ah eine lange Laufzeit versprechen. Geht man von einem Stromfluss von 10 mA aus, kann das Modul damit über drei Monate betrieben werden. Im Vergleich zu kommerziellen Wetterstationen liegt die Laufzeit damit trotzdem weit unterhalb des Durchschnitts.

An den Mikrocontroller sind wie vorher beschrieben der HKW581 IC, ein BME280, die 433 MHz- und die DCF77-Antenne angeschlossen. Zu Entwicklungszwecken sind einige Zusatzanschlüsse vorhanden, um Modifikationen möglich zu machen.

Der HKW581 ist im MSOP8 Format und damit relativ klein. Um den IC schon bei vorherigen Testaufbauten auf dem Steckbrett nutzen zu können, wurde dieser auf einen Adapter gelötet, der die Anschlusspins auf einfache THT Pins adaptiert. Um den IC zu schonen, wird der Adapter auf der Platine verbaut. Zusätzlich besteht aber die Möglichkeit, später darauf zu verzichten und das Bauteil direkt auf der Platine festzulöten. Diese Lösung findet sich ebenso auf dem später beschriebenen Innenmodul wieder, falls auf das Außenmodul verzichtet werden soll.

Als BME280 soll in erster Linie ein fertiges Shield zum Aufstecken verwendet werden, da der private Bestand davon wesentlich höher ist. Zusätzlich kann manuell aber auch direkt ein BME280 auf die Platine gelötet werden. Die entsprechenden Bauteile sind nicht platziert.

Es können zwei verschiedene Arten von DCF77 Antennen an den Mikrocontroller angeschlossen werden. Dabei wird unterschieden zwischen Shields mit drei- und vier Anschlüssen. Die Shields mit vier Anschlüssen verfügen zusätzlich über einen PON (Power On) Pin, über den das Modul an- und ausgeschaltet werden kann. Der Pin ist normalerweise Low-aktiv und wird daher auf Masse gelegt.

### 3.1.1.2 Platine

Die Platine ist im Anhang in Abbildung 13 und Abbildung 14 dargestellt. Es handelt sich um eine einfache 2-Layer Platine mit Ground Plane auf der Unterseite. An der Oberseite sind Befestigungsmöglichkeiten für die DCF77 Antenne vorgesehen (dort ist auch keine Ground Plane mehr, um Abschirmungen zu vermeiden). Im Test hat sich allerdings herausgestellt, dass die Antenne nichts empfängt, wenn sie auf der Platine angebracht ist.

## 3.2 Modul intern

### 3.2.1 Sensormodul

Das interne Sensormodul darf im Gegensatz zum externen Modul mehr Energie verbrauchen, da es mehr Daten verarbeiten muss und über einen Netzanschluss verfügt. Das Modul empfängt die über 433 MHz gesendeten Daten und liest zusätzlich zyklisch den BME280 und CO2 Sensor aus.

Als Mikrocontroller wird ein Raspberry Pi Pico verwendet, der mit zwei Kernen Multiprocessing erlaubt und damit die Reaktionszeiten beschleunigt. Der Mikrocontroller verfügt über zwei UART Busse und zwei I2C Busse. Die beiden UART Busse werden für die Kommunikation mit dem Grafikmodul und dem 433 MHz Modul verwendet.

Der verwendete CO2 Sensor verfügt zwar auch über eine UART-Schnittstelle, die zur Verfügung stehenden UART Emulatoren für den Mikrocontroller lieferten allerdings falsche Daten. Aus diesem Grund wird der Duty Cycle des zur Verfügung stehenden PWM Pins genutzt. Eine Bibliothek für den Sensor liest den PWM Pins automatisch aus und konvertiert den Duty Cycle in eine CO2 Konzentration. Da das Auslesen des PWM Signals insbesondere bei hohen Konzentrationen recht lang dauern kann, erfolgt dies ausschließlich über den zweiten Kern, der neue Werte global zur Verfügung stellt und dies über eine Flag signalisiert.

### 3.2.2 Grafikmodul

Das Grafikmodul wird beim Endprodukt zusammen mit dem internen Sensormodul als eine Baugruppe ausgeführt sein. Hauptziel ist die grafische Aufarbeitung der über UART vom internen Sensormodul empfangenen Daten. Hierfür wird ebenfalls ein Raspberry Pi Pico in der W-Variante eingesetzt. Diese ist zusätzlich zu den Schnittstellen des Picos W-LAN (IEEE 802.11 b/g/n) und Bluetooth (5.2 LE) fähig, somit kann das Grafikmodul zukünftig ebenfalls die Rolle eines mqtt-Brokers übernehmen und die aufbereiteten Daten online abrufbar machen. Für diese Aufgaben verfügt das Modul über einen externen RTC-Chip und ein 2,8 Zoll Touchdisplay.

Anders als bei den Sensoren wird beim Grafikmodul kein C-Code verwendet sondern ausschließlich in Rust programmiert. Embedded-Rust folgt dem Motto ‘safe, fast, concurrent – pick three’. Rust etabliert viele Konzepte, um die typischen Probleme, die mit vergessenen Sonderfällen einhergehen (dangling pointer, use after free, dead-locks, integer-overflow, uvm.) durch ein gutes Typsystem und einen modernen Compiler bereits zur Kompilierzeit auszuschließen. Ein Embedded-Rust-Projekt, welches erfolgreich kompiliert, wird in der Praxis auf dem Mikrocontroller nie unerwartetes Verhalten aufweisen. Trotzdem wird noch eine von C-Programmen gewohnte Performance erreicht.

Durch die Verwendung von Rust kann ebenfalls die sehr moderne Bibliothek Slint für die graphische Darstellung verwendet werden. Diese kompiliert Slint-Strukturen zu Rust-Quelltext und stellt Schnittstellen bereit, um fertige Pixelbuffer zu erhalten.

Slint-Strukturen gliedern sich in components, globals und structs. Ein Component ist ein auf dem Display anzeigbarer Baustein. Globals stellen einen Adapter zum Rust-Quelltext dar, da man die Daten dieser sowohl aus dem Slint-Skript, als auch aus dem Rust-Programm verändern kann. Structs in Slint definieren wie Structs in C oder Rust eigene zusammengesetzte Datentypen.

```
1 export component AppWindow inherits Window {
2     title: "Wetterstation";
3     min-width: 320px;
4     min-height: 240px;
5     background: Theme.palette.pure-black;
6 }
```

```
7     SmallMain {
8         preferred-width: 100%;
9         preferred-height: 100%;
10    }
11 }
12
13 export global WeatherAdapter {
14     in property <string> title: "Wetter";
15     in property <string> current-day: "Sonntag 23.6.";
16     in property <[BarTileModel]> week-model: [
17         {
18             title: "Mo",
19             icon: Images.cloudy,
20             max: 21,
21             min: 18,
22             absolute-max: 21,
23             absolute-min: 15,
24             unit: "°C"
25         },
26         {
27             title: "Di",
28             icon: Images.sunny,
29             max: 20,
30             min: 17,
31             absolute-max: 21,
32             absolute-min: 15,
33             unit: "°C"
34         },
35         {
36             title: "Mi",
37             icon: Images.cloudy,
38             max: 18,
39             min: 15,
40             absolute-max: 21,
41             absolute-min: 15,
42             unit: "°C"
43         }
44     ];
45 }
46
47 export struct BarTileModel {
48     title: string,
49     icon: image,
50     max: int,
51     min: int,
52     absolute-min: int,
53     absolute-max: int,
54     unit: string,
55 }
```

**Listing 2:** Beispiel eines components globals und structs

Während Slint das Zeichnen der Grafikbausteine übernimmt, muss sich der selbstgeschriebene Code um folgende Funktionen kümmern:

- Initialisierung der Peripherie und Anbindung von Slint
- Ansteuerung des RTC Moduls (per I2C)
- Übertragung der Pixel an das Display (per DMA SPI)

- Übertragung des Touchinputs an den Pico (per DMA SPI)
- Übertragung der Sensordaten (per UART)

Um Rechenleistung einzusparen, ist der Großteil des Programms mithilfe der `wfe` und `sev` Anweisungen des Picos (`WaitForEvent` und `Set/SignalEvent`) Event-basiert implementiert. Ein entsprechendes Interrupt wird ausgelöst, wenn

1. ein UART-Package vom internen Sensormodul empfangen wurde
2. Bildinhalte sich verändert haben (beispielsweise die Uhrzeit jede volle Sekunde)
3. der Touchscreen berührt wird

```

1 loop {
2     slint::platform::update_timers_and_animations();
3
4     if let Some(window) = self.window.borrow().clone() {
5         window.draw_if_needed(|renderer| {
6             let mut buffer_provider = self.buffer_provider.borrow_mut();
7             renderer.render_by_line(&mut *buffer_provider);
8             buffer_provider.flush_frame();
9         });
10
11         // handle touch event
12         let button = PointerEventButton::Left;
13         if let Some(event) = self
14             .touch
15             .borrow_mut()
16             .read()
17             .map_err(|_| ())
18             .unwrap()
19             .map(|point| {
20                 let position = slint::PhysicalPosition::new(
21                     (point.x * DISPLAY_SIZE.width as f32) as _,
22                     (point.y * DISPLAY_SIZE.height as f32) as _,
23                 )
24                 .to_logical(window.scale_factor());
25                 match last_touch.replace(position) {
26                     Some(_) => WindowEvent::PointerMoved { position },
27                     None => WindowEvent::PointerPressed { position, button },
28                 }
29             })
30             .or_else(|| {
31                 last_touch
32                     .take()
33                     .map(|position| WindowEvent::PointerReleased { position,
button })
34             })
35         {
36             let is_pointer_release_event =
37                 matches!(event, WindowEvent::PointerReleased { .. });
38
39             window.dispatch_event(event);
40
41             //Eventuellen Hover-State über Widgets zurücknehmen
42             if is_pointer_release_event {
43                 window.dispatch_event(WindowEvent::PointerExited);
44             }

```



```

45         //Nach Touch-Input keinen Sleep auslösen
46         continue;
47     }
48
49     if window.has_active_animations() {
50         //Bei laufenden Animationen keinen Sleep auslösen
51         continue;
52     }
53 }
54
55 //voraussichtliche wfe-Zeit berechnen
56 let sleep_duration = match slint::platform::duration_until_next_timer_update
57 () {
58     None => None,
59     Some(d) => {
60         let micros = d.as_micros() as u32;
61         if micros < 10 {
62             //Wenn man weniger als 10µs schlafen will, merk es dir mit einem
63             REIM, NEIN!
64             continue;
65         } else {
66             Some(fugit::MicrosDurationU32::micros(micros))
67         }
68     }
69 };
70
71 //Sleep bis zum nächsten Event auslösen
72 cortex_m::interrupt::free(|cs| {
73     if let Some(duration) = sleep_duration {
74         ALARM0.borrow(cs).borrow_mut().as_mut().unwrap().schedule(duration).
75         unwrap();
76     }
77
78     IRQ_PIN
79         .borrow(cs)
80         .borrow()
81         .as_ref()
82         .unwrap()
83         .set_interrupt_enabled(GpioInterrupt::LevelLow, true);
84 });
85 cortex_m::asm::wfe();
86 }

```

Listing 3: Eventloop mit wfe

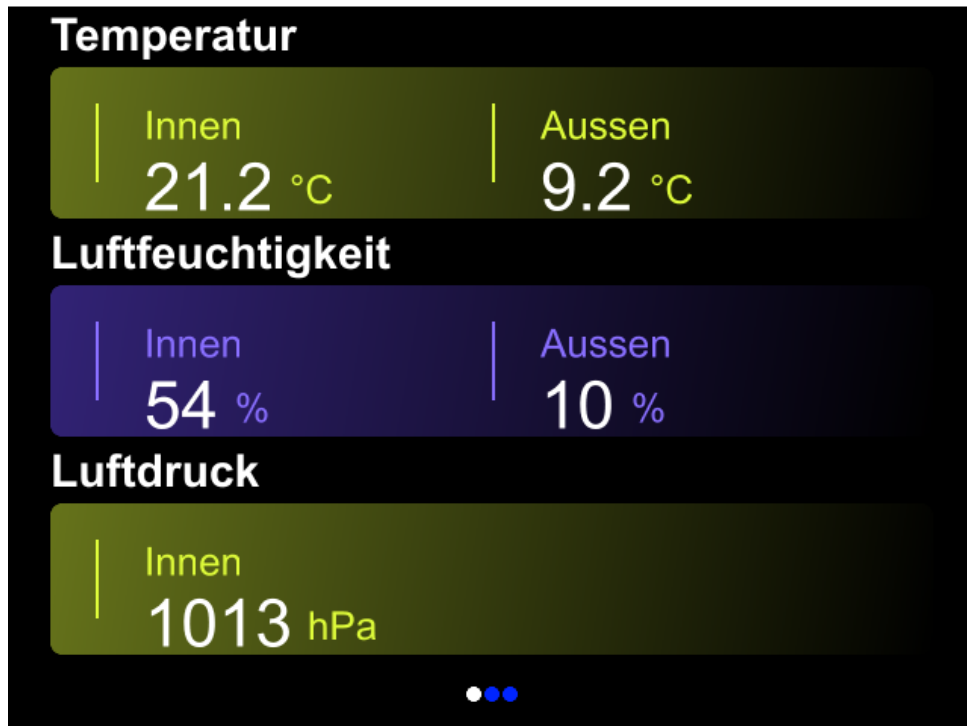


Abbildung 5: Übersichts-Seite des GUI



Abbildung 6: Wetter-Seite des GUI

### 3.2.3 Platinenentwurf

#### 3.2.3.1 Schaltplan

Der Schaltplan ist im Anhang in Abbildung 17 dargestellt. Zentraler Bestandteil sind die beiden Mikrocontroller, die über eine UART Verbindung miteinander kommunizieren. Der Raspberry Pi Pico W, der das Grafikmodul beherbergt, steuert das aufsteckbare Touchdisplay. Zusätzlich wird ein RTC Shield auf den Mikrocontroller gesteckt, um die aktuelle Zeit speichern zu können und Jitter zu minimieren. An den Raspberry Pi Pico, der die Sensoren ausliest, sind ein BME280, der CO2 Sensor und die 433 MHz Antenne angebunden. Zusätzlich kann hier ein HKW581 Modul integriert werden, dieses befindet sich im aktuellen Aufbau aber auf dem Außenmodul. Wie schon im externen Modul sind Möglichkeiten für das Shield und den Adapter vorbehalten. Der HKW581 auf dem internen Modul ist nur sinnvoll, falls auf das Außenmodul verzichtet werden soll (nicht geplant).

#### 3.2.3.2 Platine

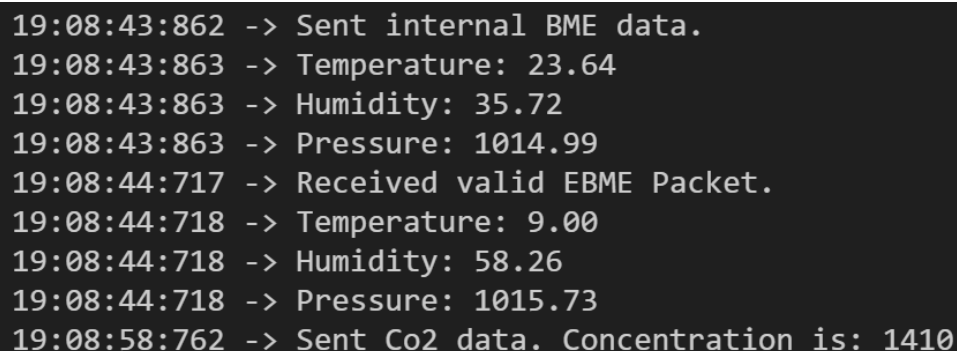
Die Platine ist im Anhang in Abbildung 18 und Abbildung 19 dargestellt. Es handelt sich um eine einfache 2-Layer Platine mit Ground Plane auf der Unterseite. Hervorzuheben sind die Befestigungsmöglichkeiten des Displays sowie Bohrlöcher an den Außenseiten zur Befestigung im Gehäuse, o.ä.. Auf der Platine sind zwei Anschlüsse zur Spannungsversorgung der Platine eingeplant, einer führt das Kabel seitlich nach links, der andere nach unten. Der Anschluss nach unten kann praktisch sein, wenn die Platine aufgehängt werden sollte. Wird diese hingegen aufgestellt, ist eine seitlich Führung des Kabels nötig.

## 4 Funktionstest

Der Funktionstest ist wegen der graphischen Oberfläche auf dem Display nicht ganz einfach zu dokumentieren.

### 4.1 Test der Sensorik

Da der Raspberry Pi Pico im Gegensatz zu herkömmlichen Arduinos zusätzlich zu den UART Bussen serielle Daten über die USB Schnittstelle übertragen kann, sind einfache Print Ausgaben der empfangenen und weitergeleiteten Daten die einfachste Möglichkeit.



```
19:08:43:862 -> Sent internal BME data.
19:08:43:863 -> Temperature: 23.64
19:08:43:863 -> Humidity: 35.72
19:08:43:863 -> Pressure: 1014.99
19:08:44:717 -> Received valid EBME Packet.
19:08:44:718 -> Temperature: 9.00
19:08:44:718 -> Humidity: 58.26
19:08:44:718 -> Pressure: 1015.73
19:08:58:762 -> Sent Co2 data. Concentration is: 1410
```

**Abbildung 7:** Logausgabe von internen und externen Sensordaten und CO2 Konzentration

Abbildung 7 zeigt die Logausgabe von BME Daten des externen und internen Sensors und die aktuell gemessene CO2 Konzentration. “Received valid EBME Packet” zeigt an, dass die

übertragene Prüfsumme korrekt ist und keine Übertragungsfehler detektiert wurden. Die drei Pakete werden anschließend über die UART-Schnittstelle an das Grafikmodul weitergeleitet.

Ebenso werden die übertragenen Zeit- und Meteotimepakete in der Konsole ausgegeben:

```
18:14:59:818 -> Received valid Time packet.
18:14:59:818 -> Hour: 18
18:14:59:819 -> Minute: 15
18:14:59:819 -> Second: 0
18:14:59:819 -> Year: 2024
18:14:59:819 -> Month: 1
18:14:59:819 -> Day: 23
18:14:59:819 -> Weekday: 2
18:15:00:094 -> Received meteodata: 1000100000010010111010
```

**Abbildung 8:** Logausgabe eines Zeit- und Meteotimepakets

Ein Meteotime Paket wird nur übertragen, wenn es relevant für die Wetterstation ist (relevante Zone) und, wenn die Ausgabe des Dekodier-ICs fehlerfrei ist. Beim hier angezeigten Meteotime Paket handelt es sich um die Höchstwerte für den Tag + 3 für die Region Rostock. Das Paket ist 24 bit lang (die beiden führenden Nullen werden hier nicht mit angezeigt).

Aus dem Paket lassen sich folgende Informationen ermitteln:

Binärwert	Lage	Dezimalwert	Beschreibung
0010	Tag	4	bedeckt
0010	Nacht	4	bedeckt
0000	Wetterextrema	0	keine
010	Niederschlagswahrscheinlichkeit	2	30%
0	Wetteranomalie	0	Nein
101110	Temperatur	29	7°C
10	Dekoderstatus	-	OK

**Tabelle 1:** Informationen aus dem Meteotime Paket

## 4.2 Test des Grafikmoduls

Das Grafikmodul muss auf zweierlei Arten betrachtet werden, einerseits muss das Userinterface einfach bedienbar und übersichtlich sein, andererseits müssen auch die Rohdaten der Sensoren korrekt empfangen und verarbeitet werden. Das Erste lässt sich durch simples "durchklicken" der Menüs erledigen.

Um die ankommenden Daten zu verifizieren müssen die Debugausgaben des versendenden Raspberry Picos (siehe 4.1) als auch die des empfangenen und anzeigenden Raspberry Picos überwacht und abgeglichen werden. Da der UI-Pico durch das Anzeigen der GUI ziemlich stark ausgelastet ist, kommt die Raspberry Pi Debug Probe zum Einsatz, welche einen seriellen Port (SerialWireDebug) über das CMSIS-DAP Protokoll öffnen kann, und die CPU des Picos nicht zusätzlich belastet.

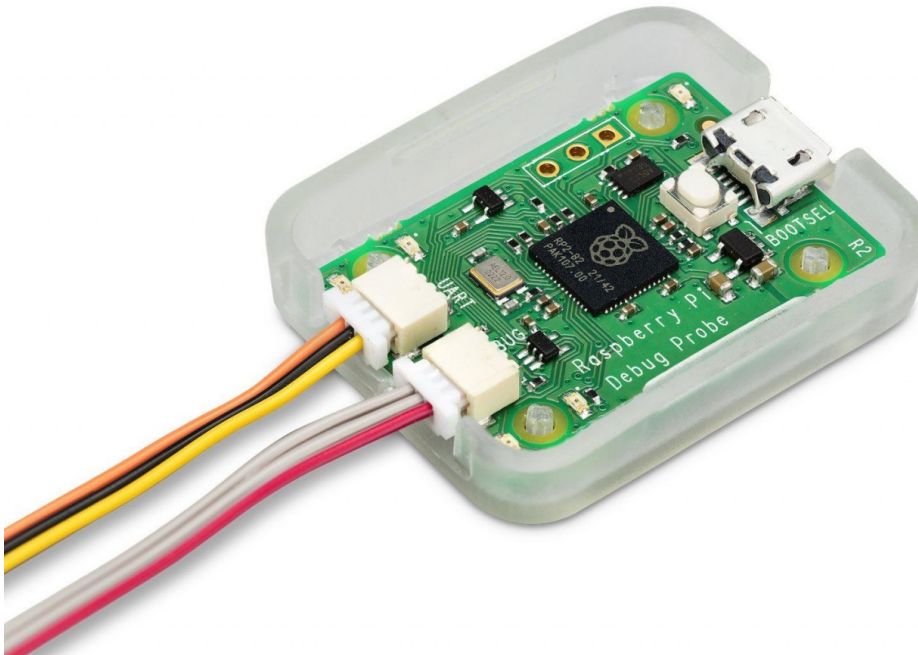


Abbildung 9: Raspberry Pico Debug Probe Bildquelle: [berrybase.de](https://www.berrybase.de)

Nun kann im laufenden Betrieb oder mit einem Testprogramm validiert werden, dass die Daten korrekt übertragen und verarbeitet werden.

```
INFO Externes Paket empfangen
└─ slint_pico::pico_backend::init_timers::{closure#0}::{closure#1} @ src/pico_backend.rs:524
INFO Internes Paket empfangen!
└─ slint_pico::pico_backend::init_timers::{closure#0}::{closure#1} @ src/pico_backend.rs:490
INFO CO2 Paket empfangen
└─ slint_pico::pico_backend::init_timers::{closure#0}::{closure#1} @ src/pico_backend.rs:515
```

Abbildung 10: Funktionstest für BME280-Sensoren und CO2-Sensor

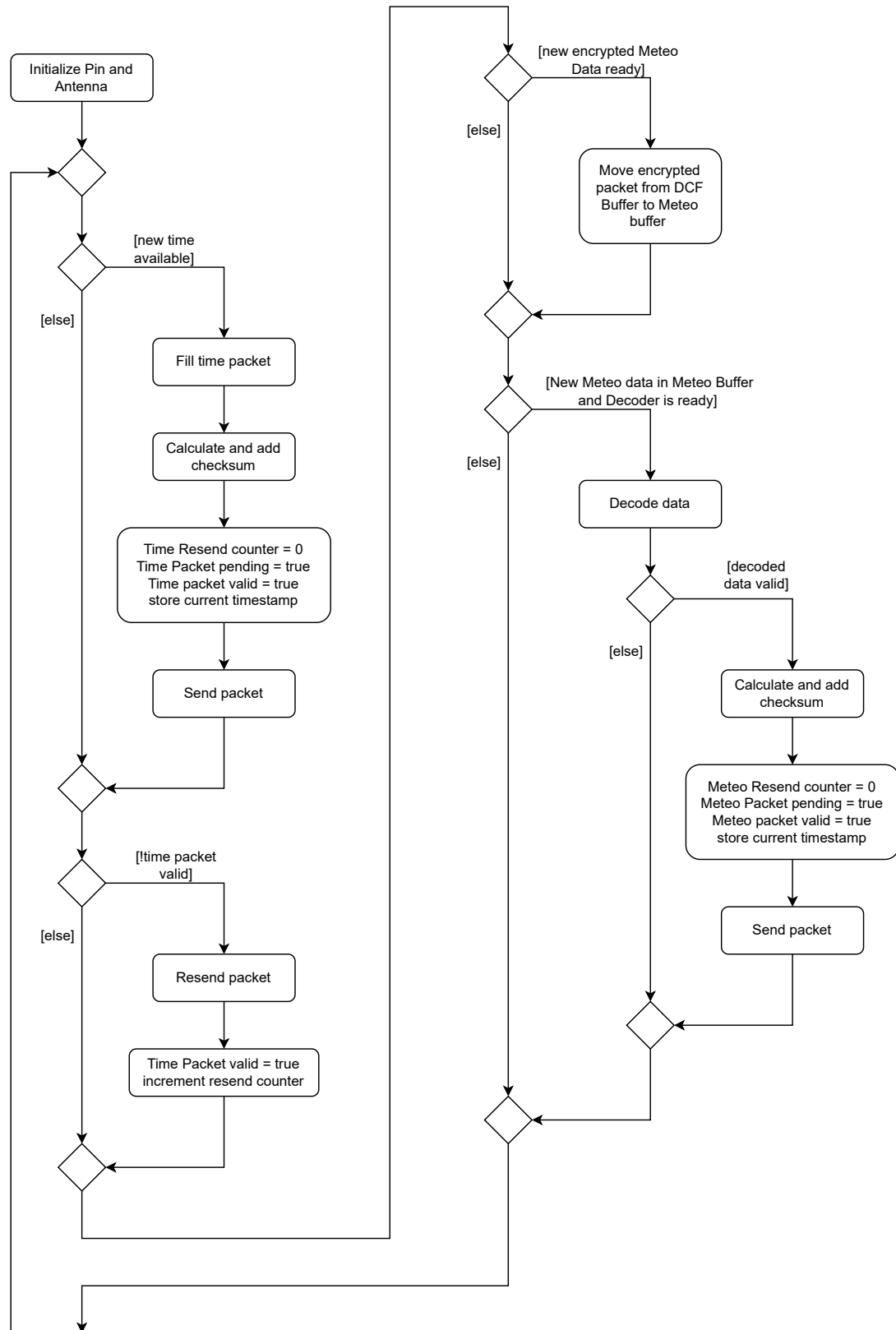
## 5 Fazit

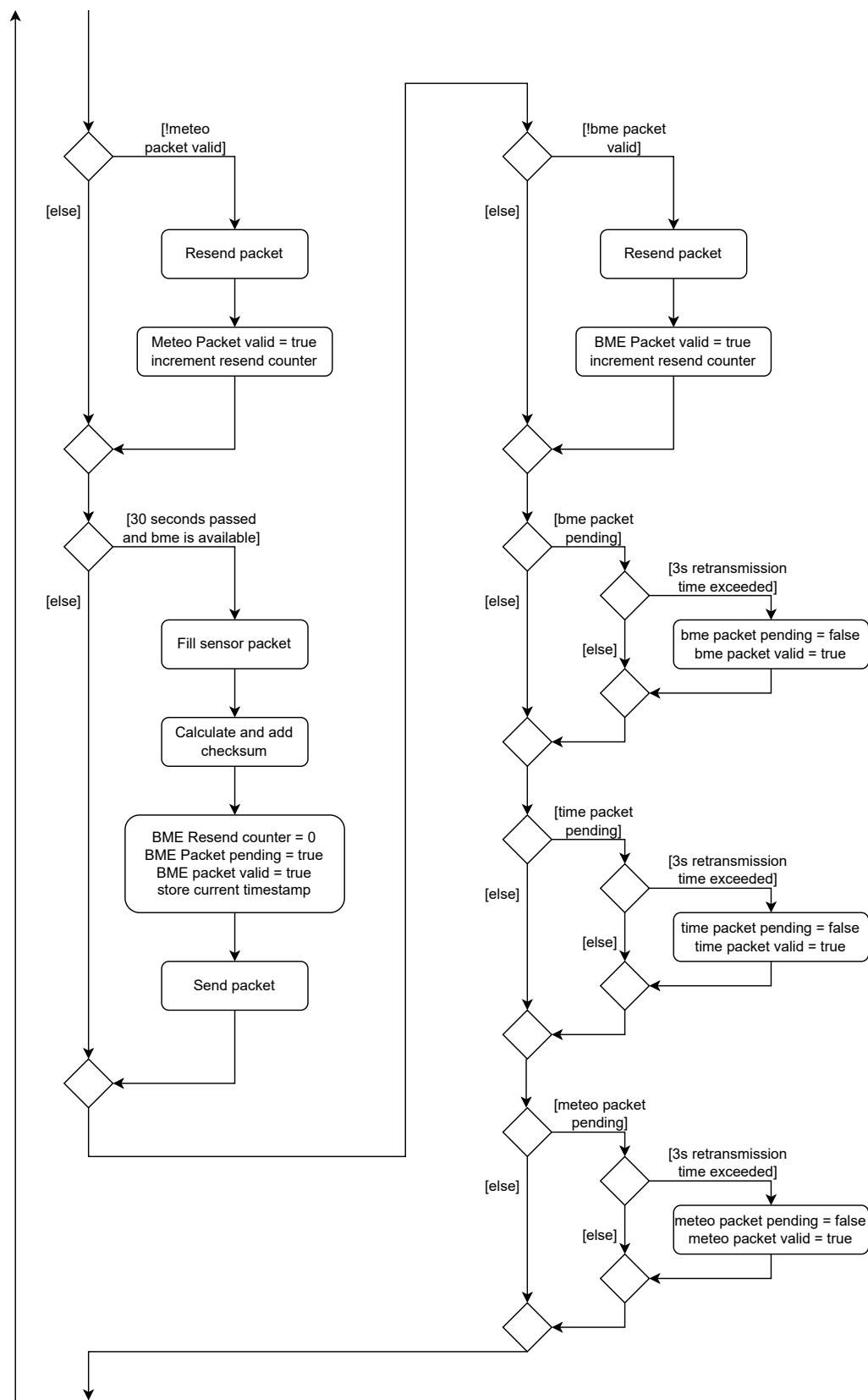
Zum Ende des Projekts ist eine vollständige Wetterstation mit funktionierendem Außenmodul entwickelt und funktionstüchtig. Mit geeigneten Gehäusen sind beide Module theoretisch für den Langzeitgebrauch geeignet und in einer realen Umgebung einsetzbar. Das Projekt kann damit als Erfolg gewertet werden, weil die zu Beginn aufgestellten Anforderungen erfüllt wurden. Im Entwicklungsprozess hat sich insbesondere gezeigt, dass das DCF Signal sehr anfällig für Störungen ist und auf die korrekte Ausrichtung der Antenne und einen geeigneten Aufstellort geachtet werden muss (Antenne sollte in waagerechter Position sein und quer in Richtung Frankfurt a.M. ausgerichtet werden). Damit erklärt sich auch, warum herkömmliche Funkuhren häufig kein Signal in Innenräumen empfangen.

Sollte das Projekt weiterverfolgt werden, kann im nächsten Iterationsschritt ein effizienterer Mikrocontroller in das Außenmodul eingesetzt werden (MSP430 o.ä.) und weitestgehend auf Shields verzichtet werden, um die Abmessungen der Platinen zu reduzieren. Damit ließe sich die Professionalität weiter steigern.

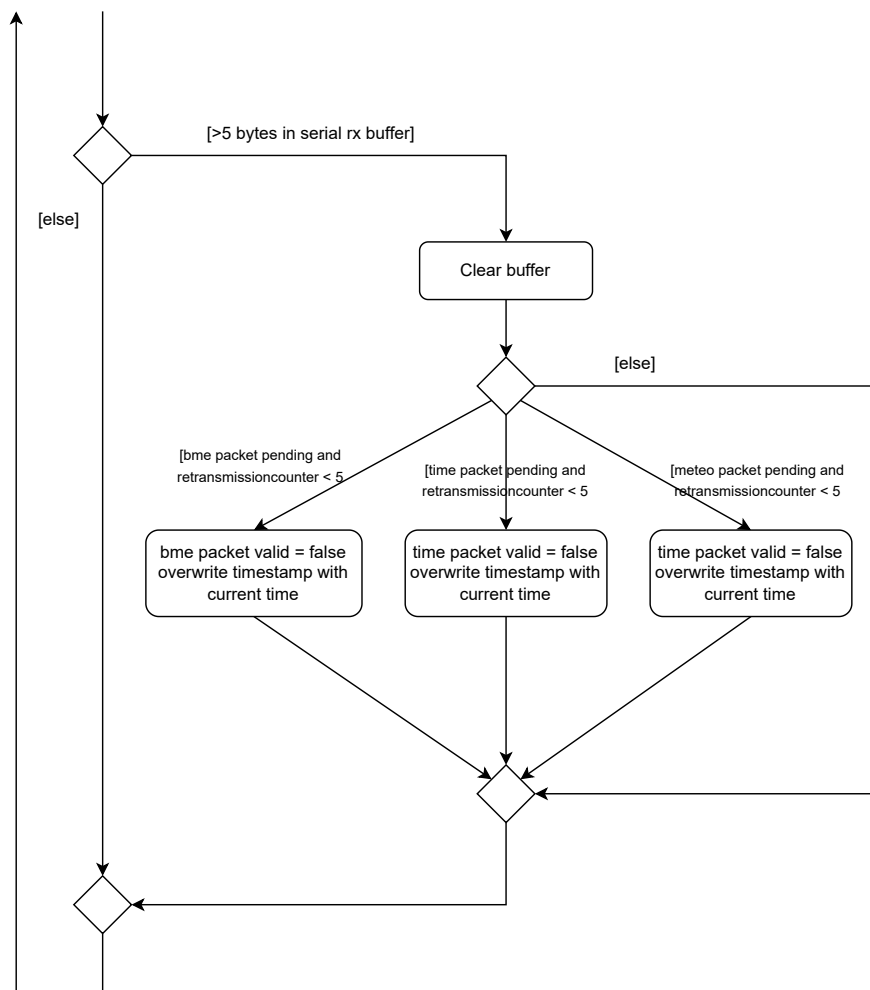


## A Ablaufdiagramm externes Modul









**Abbildung 11:** Abstraktes Ablaufdiagramm zur Beschreibung des Codes auf dem externen Sensormodul

## B Schaltplan externes Modul

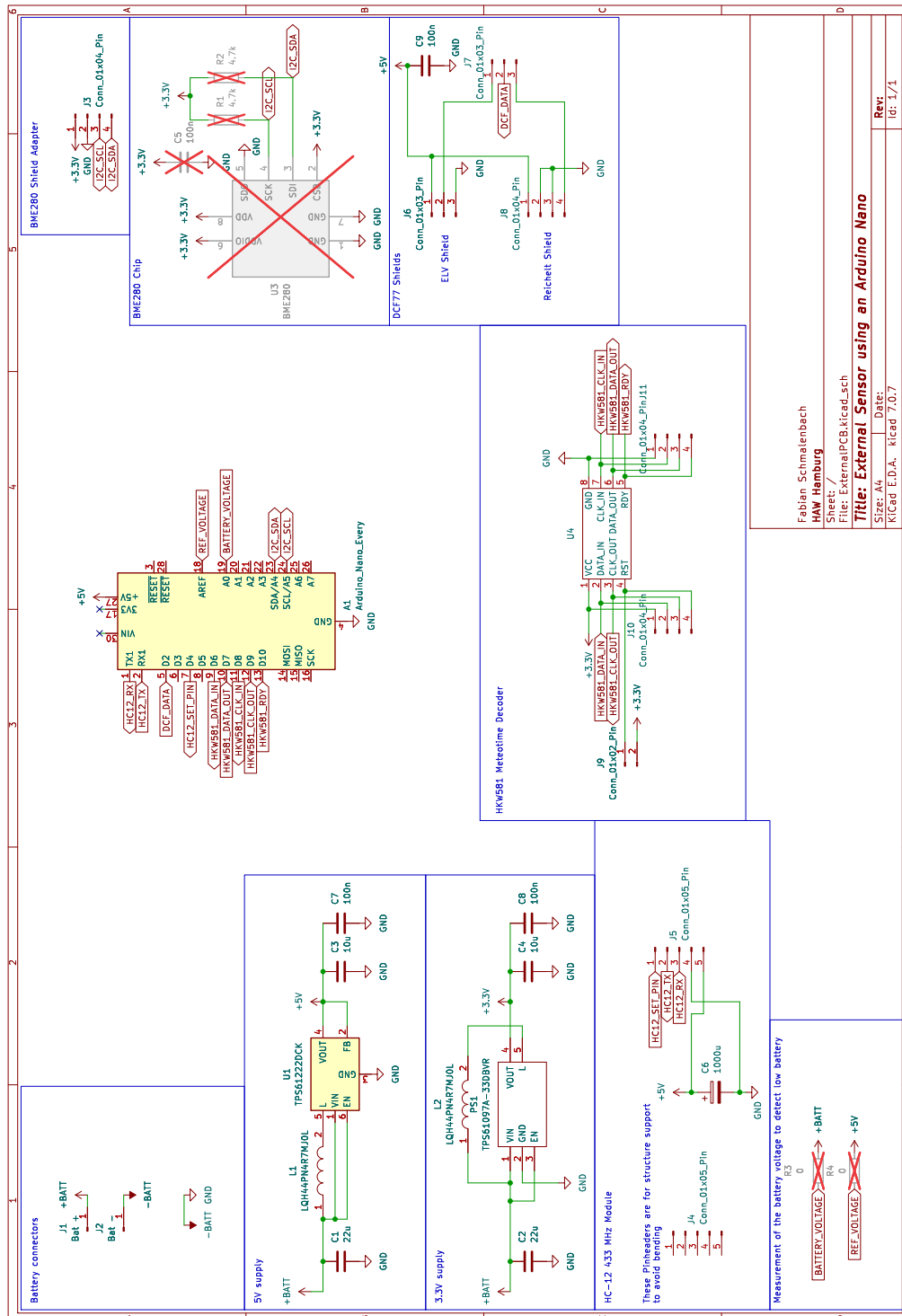
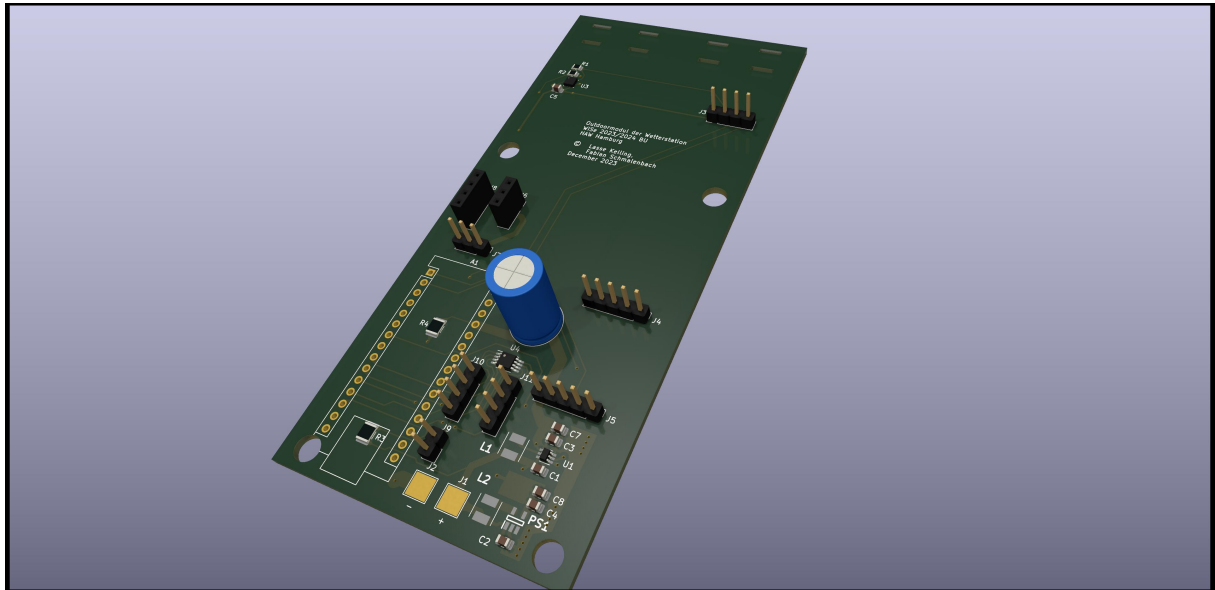


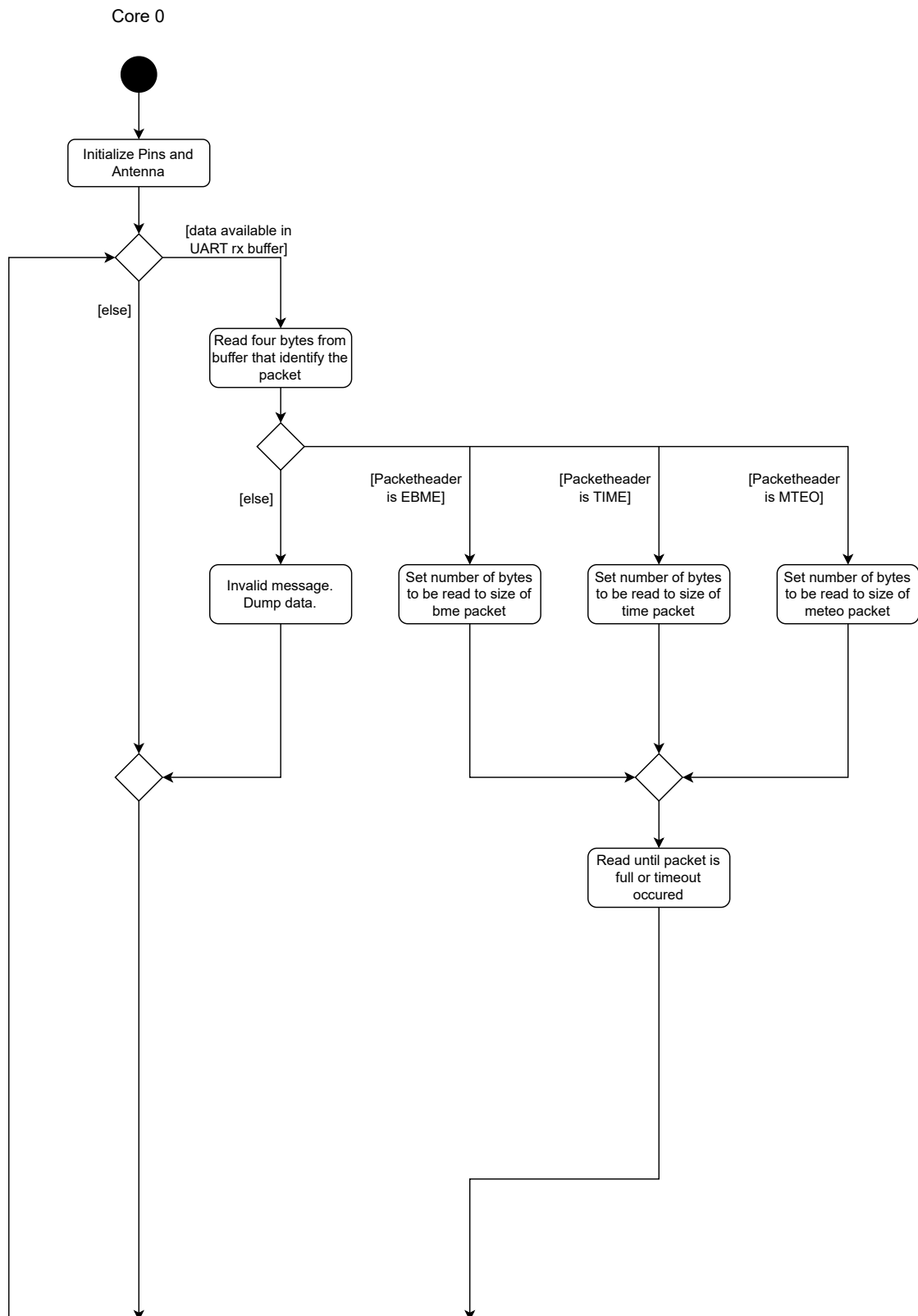
Abbildung 12: Schaltplan des externen Moduls

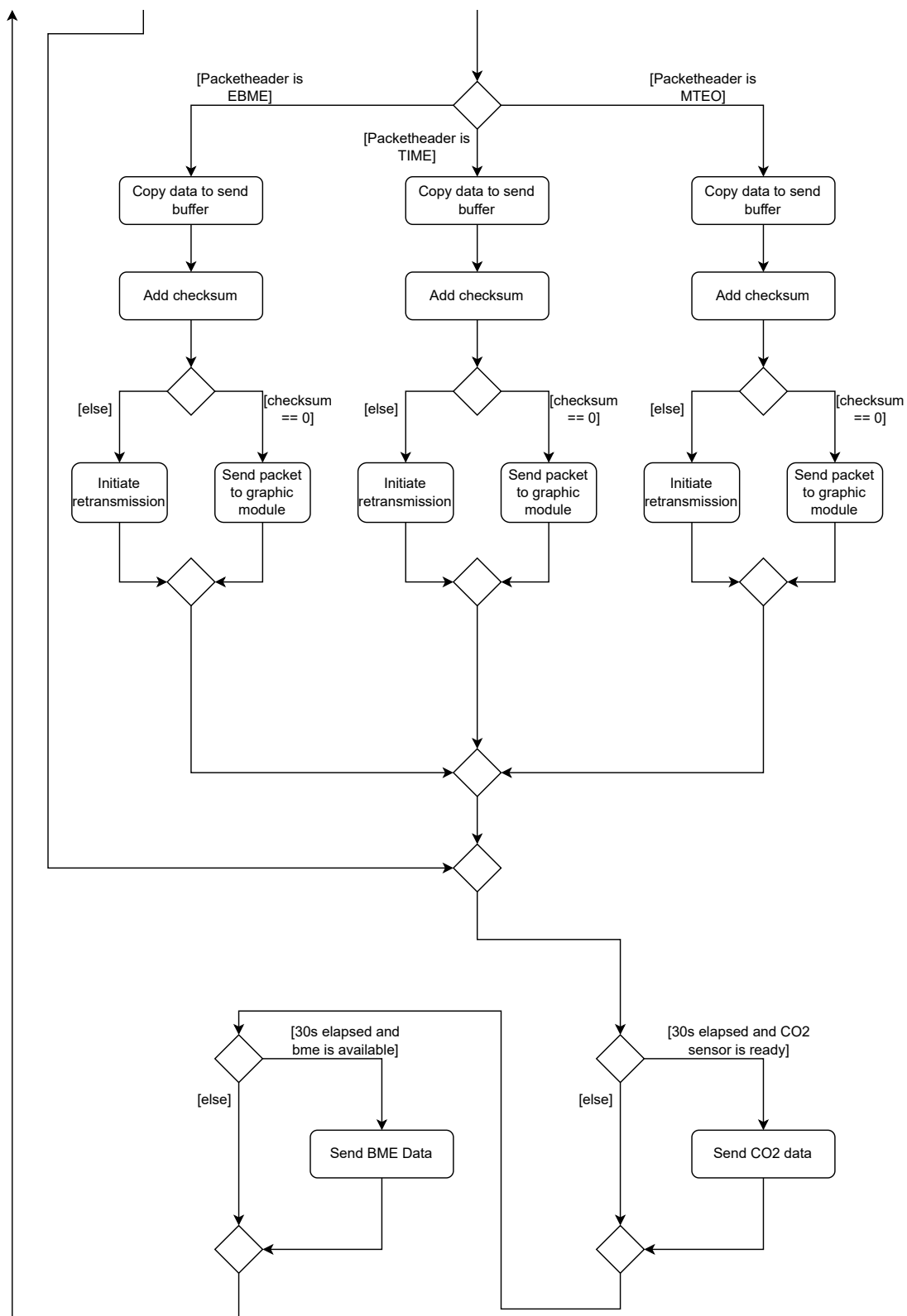




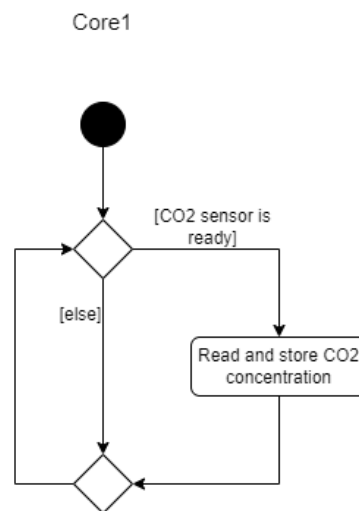


## D Ablaufdiagramm internes Sensormodul





**Abbildung 15:** Abstraktes Ablaufdiagramm zur Beschreibung des Codes auf dem internen Sensormodul (Core 0)



**Abbildung 16:** Abstraktes Ablaufdiagramm zur Beschreibung des Codes auf dem internen Sensormodul (Core 1)







## F.2 3D Ansicht

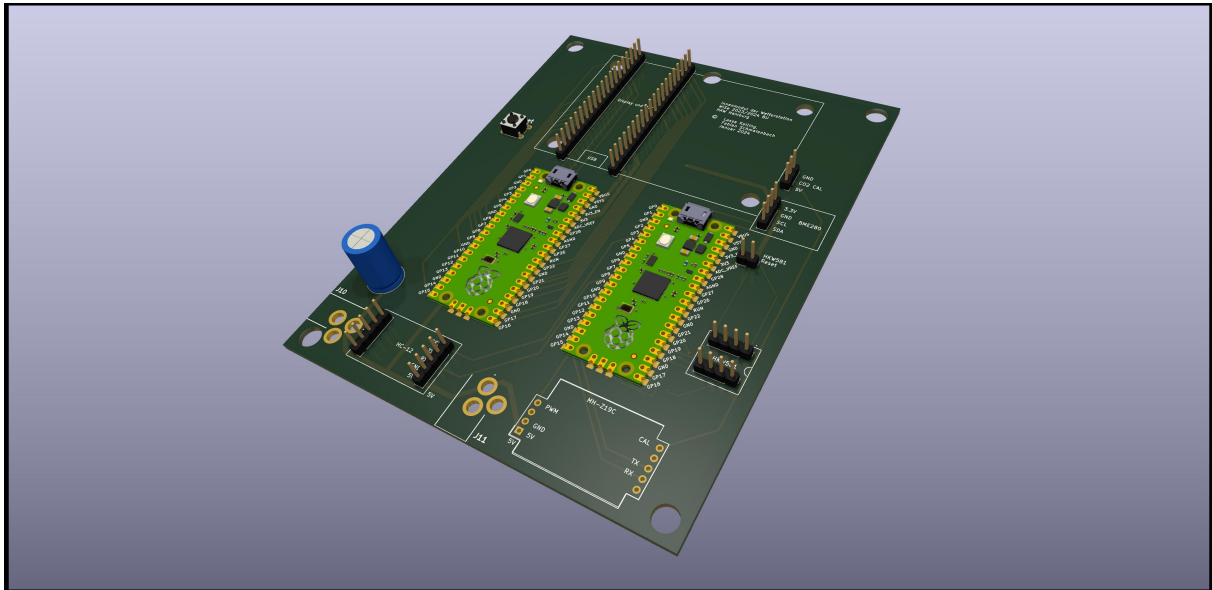


Abbildung 19: 3D Ansicht des Platinenentwurfs