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

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.

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

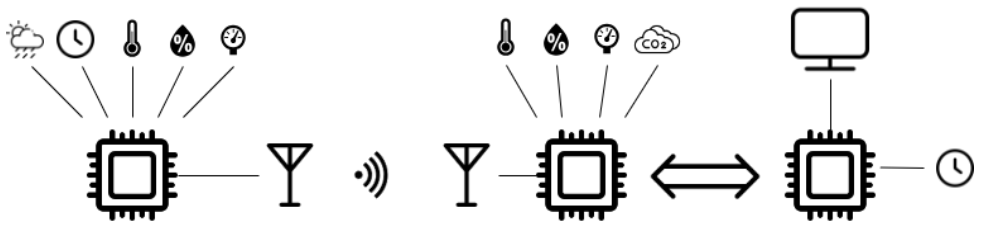

Abbildung 1: Übersicht der Systemkomponenten

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, entschlüsselt 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 μA, weshalb sich der Sensor ideal für die Anwendung in Sensormodulen mit Batteriebetrieb eignet.

Der Sensor verfügt über die folgenden Messbereiche:

- Temperatur: -40°C - +85°C (± 0,5°C)
- Luftfeuchtigkeit: 0 - 100% rel. Feuchtigkeit (± 3%)
- Druck: 250 - 1250 hPa (± 1 hPa)

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

2.2.2 MHZ19C

Der MHZ-19C ist ein Infrarot CO2-Sensor, der die CO2 Konzentration mittels nicht-dispersiver Infrarot-Spektroskopie misst. Dabei wird zyklisch mit einer Infrarotlampe und einem Photosensor die CO2-Konzentration anhand der Reflexion des Lichts durch CO2 Partikel gemessen. Der Sensor verfügt über einen Messbereich von 400-5000 ppm, wobei die Genauigkeit ± 40ppm + 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.

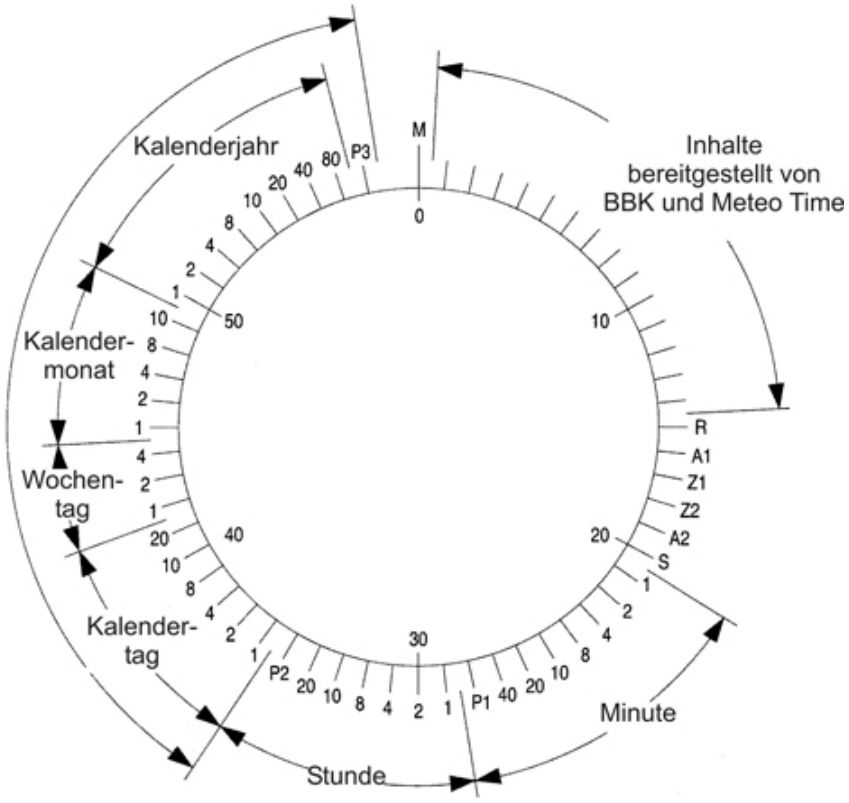

Abbildung 2: Kodierungsschema der DCF77 Zeitinformationen
https://www.ptb.de/cms/ptb/fachabteilungen/abt4/fb-44/ag-442/verbreitung-der-gesetzlichen-zeit/dcf77/zeitcode.html

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 anhanddessen 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 versendet pro Minute 14 Bits mit Wetterinformationen, die von lizensierten 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:
22:00 Uhr - 03:59 Uhr: Aktueller Tag
04:00 Uhr - 09:59 Uhr: Folgender Tag
10:00 Uhr - 15:59 Uhr: Darauffolgender Tag
16:00 Uhr - 18:59 Uhr: Darauffolgender Tag
19:00 Uhr - 21:59 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 sechsstündigen Übertragungszeitraums werden für jede Region nacheinander die Höchst- und Tiefstwerde 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 beim Empfang der Daten trotzdem eine Vorhersage ableiten zu können, sollten Ausweichregionen definiert werden, die im Fehlerfall stattdessen angezeigt werden.
Für Hamburg sind das die Regionen Rostock und Hannover.

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.

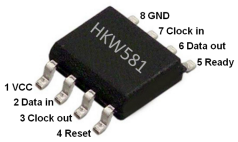

Abbildung 3: Dekodier-IC zum Entschlüsseln der Wettervorhersagen

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. Der Clock OUT Pin folgt dem Clock IN Pin und wird zur Indikation genutzt, ob am Datenausgang ein valider Bitwert anliegt. 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:

- Meteopaket 1 (14 Bit)
- Meteopaket 2 (14 Bit)
- Meteopaket 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 erfolt über einen 433 MHz Transmitter und Receiver, die jeweils über UART beschrieben und gelesen werden können.

2.3.1 Funkstrecke

Bei den Funkmodulen handelt es sich um "HC-12 Wireless RF communicatio module V2.4"Module. Diese versprechen eine Übertragungsdistanz von bis zu 1800 m und eine einfache Ansteuerung über UART.
Die Module haben eine Sendeleistung von bis zu 100 mW und können auf 100 verschiedenen, einstellbaren Kanälen funken (433,4 MHz bis 473 MHz). Die Baudrate ist einstellbar zwischen 1200 bps und 115200 bd.

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 Date werden diese dann in char arrays umgewandelt.
Alle Datenpakete sind im folgenden dargestellt:

```
// Aktuelle Zeit
typedef struct {
    const char dataType[4];
    uint8_t hour;
    uint8_t minute;
    uint8_t second;
    uint8_t year;
    uint8_t month;
    uint8_t day;
} TimeStruct;

typedef union {
    char buf[sizeof(TimeStruct)];
    TimeStruct data;
} TimeSendBuf;

// Kodierte Wetterdaten mit Zeitinformationen zum Dekodieren
typedef struct {
    const char dataType[4];
    uint16_t packet1;
    uint16_t packet2;
    uint16_t packet3;
    uint8_t minute;
    uint8_t hour;
    uint8_t date;
    uint8_t month; // Only 5 bits used
    uint8_t dayInWeek;  // Only 3 bits used
    uint8_t year;
} MeteoRawStruct;

typedef union {
    char buf[sizeof(MeteoRawStruct)];
    MeteoRawStruct data;
} MeteoRawSendBuf;

// Dekodierte Wetterdaten
typedef struct {
    const char dataType[4];
    uint32_t meteoData;
} MeteoDecodedStruct;

typedef union {
    char buf[sizeof(MeteoDecodedStruct)];
    MeteoDecodedStruct data;
} MeteoDecodedSendBuf;

// Sensordaten vom BME280
typedef struct {
    const char dataType[4];
    float temperature;
    float humidity;
    float pressure;
} BMEStruct;

typedef union {
    char buf[sizeof(BMEStruct)];
    BMEStruct data;
```

```
58 } BMESendBuf;
59
60 // CO2 Konzentration
61 typedef struct {
62     const char dataType[4];
63     int concentration;
64 } CO2Struct;
65
66 typedef union {
67     char buf[sizeof(CO2Struct)];
68     CO2Struct data;
69 } CO2SendBuf;
```

Listing 1: Datenpakete

3 Modulbeschreibung

3.1 Sensormodul extern

Das externe Sensormodul soll für den Außenbereich konzipiert werden und möglichst effizient und langfristig mit Batterien betrieben werden können.
Wie schon aus den vorherigen Abschnitten hervorgeht, verfügt das Modul über einen BME280 und eine Antenne zum Empfang von DCF77 Signalen. Die empfangenen Daten werden dann zyklisch via 433 MHz an den Sensorhub gesendet.

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 mit Interrupts auf steigende- und fallende Taktflanke arbeitet und jeweils die Zeiten der Absenkung misst. Da in der letzten Sekunde einer Minute keine Absenkung stattfindet, kann anhanddessen das Ende eines Datenpakets ermittelt werden. Die empfangenen Daten werden anschließend in einem Datenpaket gespeichert und versendet. Wird eine Pegeländerung außerhalb des gültigen Bereichs erkannt, wird aktuell das ganze Minutenpaket als ungültig deklariert und ignoriert. Alle drei Minuten ist außerdem ein MeteoTime Paket Sendebereit, dieses wird anhand der aktuellen Zeit auf Relevanz geprüft und verworfen, wenn die Region irrelevant ist. Die BME280 Daten werden zyklisch alle 30 Sekunden gelesen und dann gesendet.

Als Mikrocontroller wird aktuell ein ESP8266 verwendet, der mit einer Stromaufnahme von ca. 20 mA relativ viel Strom benötigt. Wegen des Zeitsignals kommen Schlafmodi nicht infrage. Damit wäre der Batteriebetrieb mit langen Laufzeiten nur schwer realisierbar. Als Alternative wurde daher ein MSP430G2553 ausgewählt, der über einen I2C und einen UART Bus verfügt und bei 1 MHz Takt einen Stromverbrauch von nur 230 μA hat. Damit wäre ein langfristiger Betrieb mit Batterie möglich. Das HC-12 Modul hat im Sendebetrieb allerdings eine Stromaufnahme von bis zu 100 mA, die sich bei häufiger Sendefrequenz sehr negativ auswirken würde. Die Sendefrequenz muss daher so niedrig wie nötig gewählt werden.

3.2 Sensormodul intern

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. Wird ein kodiertes Paket mit Wetterdaten empfangen, wird dieses außerdem entschlüsselt und dann an das Grafikmodul weitergegeben.

Als Mikrocontroller dient hier ein Raspberry Pi Pico, 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 zum einen für die Kommunikation mit dem anderen Mikrocontroller, zum anderen zur Datenübertragung vom 433 MHz Empfänger verwendet. Der CO2 Sensor hat zwar auch eine UART Schnittstelle, da UART Emulatoren aber regelmäßig falsche Daten geliefert haben, werden die Daten stattdessen via PWM ausgelesen. Eine Bibliothek für den Sensor liest den Duty-Cycle aus und ermittelt daraus die 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. Auf Kern 2 findet außerdem der Dekodierprozess der Wetterdaten statt, um den Hauptprozess nicht zu verlangsamen. Auch hier werden fertige Pakete über eine Flag signalisiert.

3.3 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.

```
export component AppWindow inherits Window {
    title: "Wetterstation";
    min-width: 320px;
    min-height: 240px;
    background: Theme.palette.pure-black;

    SmallMain {
        preferred-width: 100%;
        preferred-height: 100%;
    }
}

export global WeatherAdapter {
```

```
    in property <string> title: "Wetter";
    in property <string> current-day: "Sonntag 23.6.";
    in property <[BarTileModel]> week-model: [
        {
            title: "Mo",
            icon: Images.cloudy,
            max: 21,
            min: 18,
            absolute-max: 21,
            absolute-min: 15,
            unit: "°C"
        },
        {
            title: "Di",
            icon: Images.sunny,
            max: 20,
            min: 17,
            absolute-max: 21,
            absolute-min: 15,
            unit: "°C"
        },
        {
            title: "Mi",
            icon: Images.cloudy,
            max: 18,
            min: 15,
            absolute-max: 21,
            absolute-min: 15,
            unit: "°C"
        }
    ];
}

export struct BarTileModel {
    title: string,
    icon: image,
    max: int,
    min: int,
    absolute-min: int,
    absolute-max: int,
    unit: string,
}
```

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 (**W**ait**F**or**E**vent und **S**et/**S**ignal**EV**ent) 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

```
loop {
    slint::platform::update_timers_and_animations();

    if let Some(window) = self.window.borrow().clone() {
        window.draw_if_needed(|renderer| {
            let mut buffer_provider = self.buffer_provider.borrow_mut();
            renderer.render_by_line(&mut *buffer_provider);
            buffer_provider.flush_frame();
        });

        // handle touch event
        let button = PointerEventButton::Left;
        if let Some(event) = self
            .touch
            .borrow_mut()
            .read()
            .map_err(|_| ())
            .unwrap()
            .map(|point| {
                let position = slint::PhysicalPosition::new(
                    (point.x * DISPLAY_SIZE.width as f32) as _,
                    (point.y * DISPLAY_SIZE.height as f32) as _,
                )
                .to_logical(window.scale_factor());
                match last_touch.replace(position) {
                    Some(_) => WindowEvent::PointerMoved { position },
                    None => WindowEvent::PointerPressed { position, button },
                }
            })
            .or_else(|| {
                last_touch
                    .take()
                    .map(|position| WindowEvent::PointerReleased { position,
    button })
            })
        {
            let is_pointer_release_event =
                matches!(event, WindowEvent::PointerReleased { .. });

            window.dispatch_event(event);

            //Eventuellen Hover-State über Widgets zurücknehmen
            if is_pointer_release_event {
                window.dispatch_event(WindowEvent::PointerExited);
            }
            //Nach Touch-Input keinen Sleep auslösen
            continue;
        }

        if window.has_active_animations() {
            //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          None => None,
58          Some(d) => {
59              let micros = d.as_micros() as u32;
60              if micros < 10 {
61                  //Wenn man weniger als 10µs schlafen will, merk es dir mit einem
         REIM, NEIN!
62                  continue;
63              } else {
64                  Some(fugit::MicrosDurationU32::micros(micros))
65              }
66          }
67      };
68
69      //Sleep bis zum nächsten Event auslösen
70      cortex_m::interrupt::free(|cs| {
71          if let Some(duration) = sleep_duration {
72              ALARM0.borrow(cs).borrow_mut().as_mut().unwrap().schedule(duration).
        unwrap();
73          }
74
75          IRQ_PIN
76              .borrow(cs)
77              .borrow()
78              .as_ref()
79              .unwrap()
80              .set_interrupt_enabled(GpioInterrupt::LevelLow, true);
81      });
82      cortex_m::asm::wfe();
83  }
```

Listing 3: Eventloop mit wfe

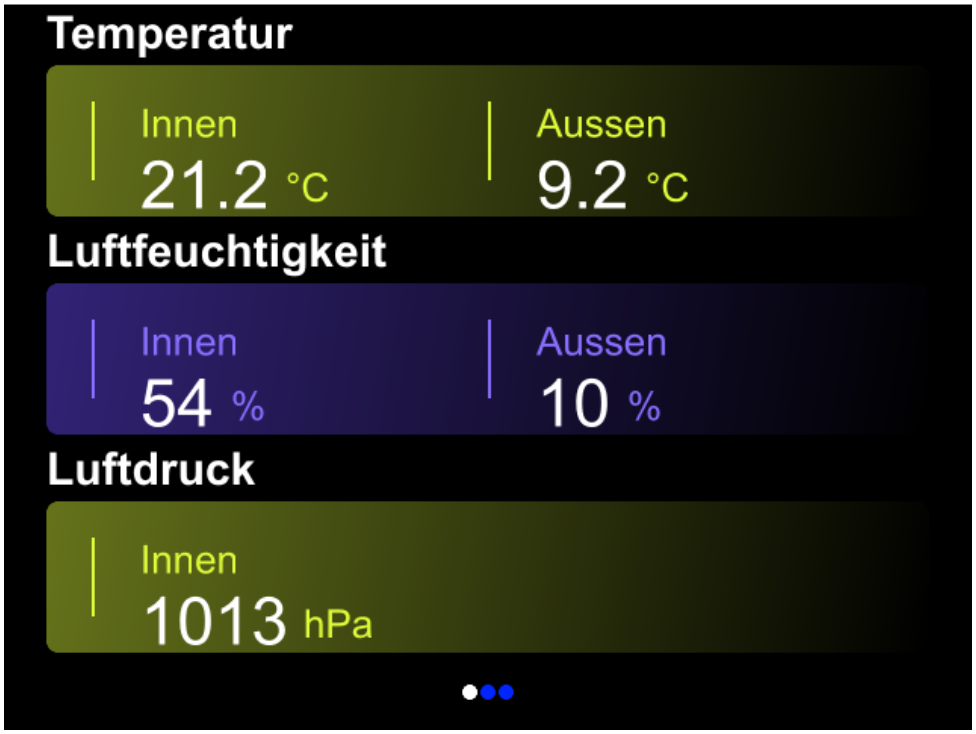

Abbildung 4: Übersichts-Seite des GUI

Abbildung 5: Wetter-Seite des GUI

4 Aktueller Projektstand

Der Entwicklungsprozess der Software ist bereits weit fortgeschritten, eine funktionierende Grafikoberfläche ist bereits vorhanden und auch die Kommunikation zwischen den Sensoren konnte schon erfolgreich getestet werden. Vor kurzem musste der Mikrocontroller des Innenmoduls aus Mangel an GPIO Pins gewechselt werden, die Migration der Software war nicht ohne weiteres möglich. Außerdem ist aktuell der Wechsel vom ESP8266 zum MSP430 geplant, durch die stark limitierten Ressourcen beim MSP430 und der Wechsel der Programmiersprache von C++ zu C machen den Wechsel aber sehr aufwendig und fehlerbehaftet. Eventuell wird der MSP430 aufgrund des hohen Aufwands wieder aus dem Projekt gestrichen, sodass der Testaufbau mit dem (wenig effizienten) ESP8266 stattfinden muss.

Auf die fertige Software folgt die Entwicklung und Bestellung von Leiterplatten für die Module, um einen sauberen Aufbau ohne unnötige Kabel möglich zu machen.