Project Weatherstation

Team members
Peter Noack, Jan Reiner, Daniel Schranzhofer

HTBL Bulme Graz-Gösting
schoolyear 2025

supervising teacher
FL Ing. Gimpl Martin, BEd

BULME

htl
bildung mit zukunft

Table of contents

Team member

Project leader: Noack Peter

As the Lead Hardware Engineer, Peter Noack played a key role in the development of the project's electronic systems. He was responsible for designing the circuit schematic, which formed the technical foundation for the device's functionality. Building on this, he created a compact and efficient PCB layout.
His responsibilities also included the sourcing and management of all electronic components, considering availability, compatibility, and cost-effectiveness. He additionally prepared the assembly process and oversaw the placement of components using a pick-and-place machine, ensuring a high level of precision in the final hardware implementation.

Team member: Reiner Jan

As the Principal Software Engineer, Jan Reiner was responsible for designing and implementing the complete software architecture of the project. His work focused on creating a modular and user-friendly codebase that could be extended and maintained with ease, while also delivering robust and stable performance.
In addition, he contributed as an Additive Manufacturing Specialist, handling the 3D printing of all mechanical components. He also supported the mechanical design process by modeling minor components and ensured the quality and dimensional accuracy of printed parts through careful post-processing.

Team member: Schranzhofer Daniel

In his role as Head of Mechanical Design, Daniel Schranzhofer was responsible for creating all major 3D models used in the project. Utilizing advanced CAD tools, he developed detailed housings and mounting

structures that combined aesthetic appeal with functionality and manufacturability via 3D printing.
He also served as the Sensor Integration Specialist, with a key contribution being the implementation of a contactless wind speed measurement system using ultrasonic anemometry. This required not only precise hardware interfacing but also calibration and integration with the system software. He also supported the software development.

Project

Project Description

As part of the BULME Workshops Project 2024/2025, the BlueRubberDuck weather data acquisition module was developed as a comprehensive solution for continuous, precise, and modularly expandable measurement of meteorological and environmental data. The system is designed for both stationary outdoor use (outdoor module) and indoor climate data acquisition (indoor module), thus providing a technological foundation for intelligent, sensor-based environmental monitoring in the context of smart home applications.

Central Processing Unit

The core computing unit of both module variants is based on an ESP32-C6 microcontroller. This platform was selected due to its high energy efficiency, integrated wireless technologies (Wi-Fi 6, BLE, IEEE 802.15.4), and its open development environment (ESP-IDF). A particular focus was placed on future protocol flexibility: through targeted software adjustments, the communication protocol in use - e.g., from Wi-Fi to Thread or other IP-based protocols - can be migrated without modifying the hardware architecture.

Sensors

The following sensors were integrated into the system for environmental data acquisition:

BME688 (Bosch Sensortec):

Multifunctional environmental sensor for measuring temperature, relative humidity, air pressure, and air quality (based on VOC detection),

VEML6030:

High-precision ambient light sensor for continuous illuminance measurement,

DS18B20: Digital 1-Wire temperature sensor for precise point temperature measurements,

MH-Z19B:

NDIR-based CO,ÇÇ sensor for quantitative measurement of CO,ÇÇ concentration in ambient air,

Reed switches with tipping bucket mechanism for rainfall measurement via event counting,

Ultrasonic sensors (quad arrangement): For wind speed and direction measurement based on time-of-flight differences,

WS2812B RGB LED: For visual indication of system status and operational states (e.g., error, network connection, calibration).

Mechanics and Protection

All measurement technology is housed in specially developed, fully plastic, weatherproof enclosures. These were designed to withstand weather influences, UV exposure, condensation, and thermal expansion. Fan-assisted

sensor chambers improve sensor response time to environmental changes and prevent heat buildup under direct sunlight.

Communication and Integration

Communication with higher-level systems is handled via the Matter standard by the Connectivity Standards Alliance (CSA). Matter ensures platform-independent interoperability between devices from different manufacturers while also providing modern security and update mechanisms (e.g., device attestation, OTA firmware updates, multi-admin environments). Integration into common smart home systems (Apple HomeKit, Google Home, Amazon Alexa, etc.) is possible without vendor-specific additional software.

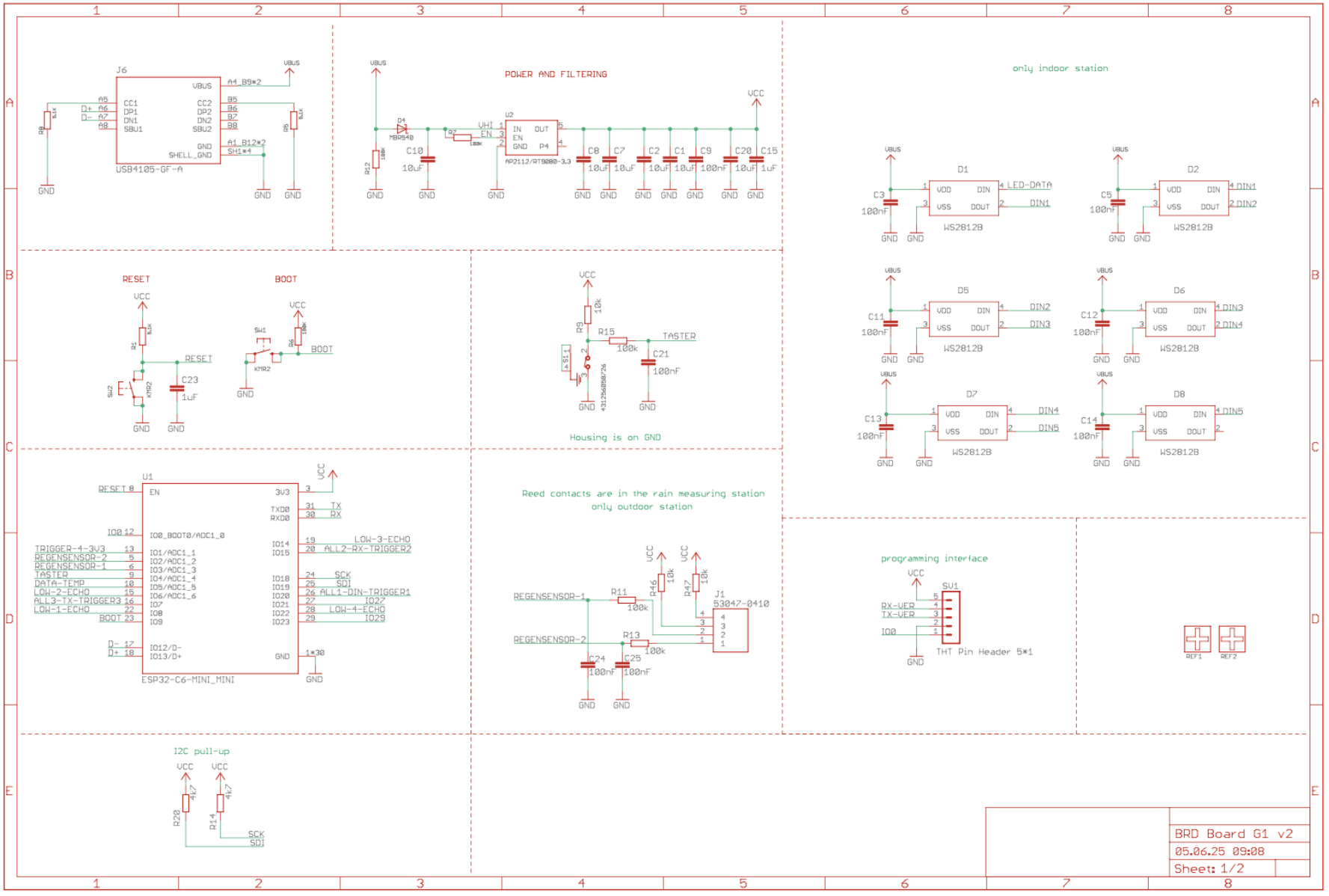
Power Supply

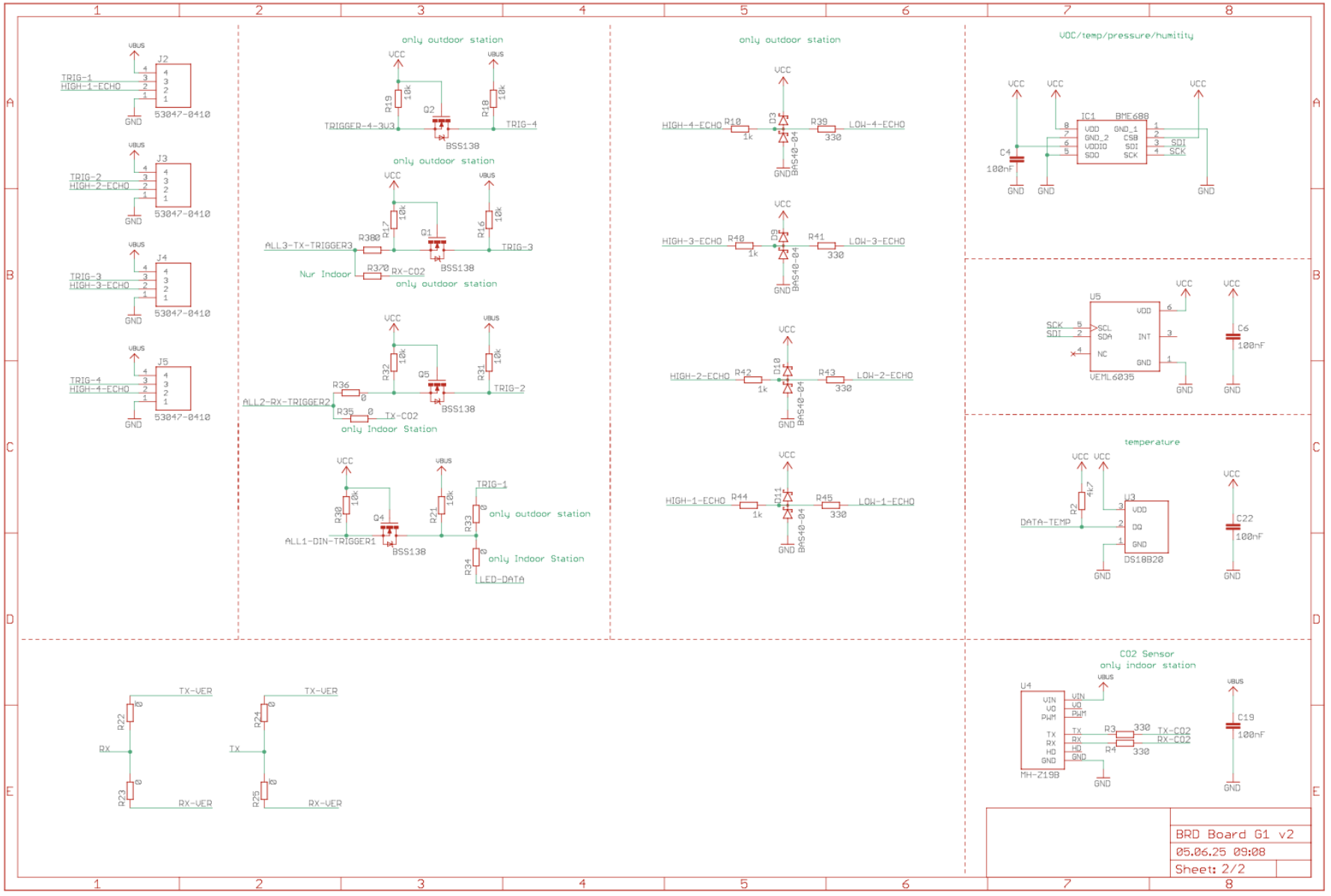
All components are powered via USB-C at 5V, combined with a voltage regulator down to 3.3V for the sensors and logic components. The system is designed for continuous mains operation; optional extensions for photovoltaic operation are currently being planned.

Schematic & PCB Layout

Schematic

We used the schematic from the ESP32-C6 DevKit by Adafruit. Then we deleted the parts we didn’t need. We changed one capacitor from an 0805 to a 0603 footprint and added a 100 nF capacitor to the power supply of the ESP32. We added one BME688, one DS18B20, one VEML6035, one MH-Z19B with 330 Ω resistors for possibility, six WS2812B, one switch, four level shifters (3.3 V to 5 V), four 5 V to 3.3 V level shifters with two resistors and one BAS40-04 diode each, one bridge to switch RX/TX if it were wrongly connected, two references for the pick & place machine and 5-pin connectors for the rain sensor and for the anemometer. Also, we debounced the rain sensor and the new switch. We added a 100 nF capacitor for every sensor and LED.


POWER AND FILTERING
only indoor station
RESET
BOOT
Housing is on GND
Reed contacts are in the rain measuring station
only outdoor station
programming interface
I2C pull-up
USB4105-GF-A
AP2112/RT9080-3.3
WS2812B
THT Pin Header 5*1
ESP32-C6-MINI_MINI
53047-0410
BRD Board G1 v2
05.06.25 09:08
Sheet: 1/2


only outdoor station
only outdoor station
only outdoor station
only Indoor Station
only outdoor station
only Indoor Station
VOC/temp/pressure/humitity
temperature
CO2 Sensor
only indoor station
TRIG-1
HIGH-1-ECHO
TRIG-2
HIGH-2-ECHO
TRIG-3
HIGH-3-ECHO
TRIG-4
HIGH-4-ECHO
53047-0410
TRIGGER-4-3V3
ALL3-TX-TRIGGER3
Nur Indoor
RX-CO2
ALL2-RX-TRIGGER2
TX-CO2
ALL1-DIN-TRIGGER1
LED-DATA
BSS138
BAS40-04
LOW-4-ECHO
LOW-3-ECHO
LOW-2-ECHO
LOW-1-ECHO
BME688
VEML6035
DS18B20
DATA-TEMP
MH-Z19B
TX-VER
RX-VER
RX
TX
BRD Board G1 v2
05.06.25 09:08
Sheet: 2/2

ESP32-C6

The ESP32-C6 microcontroller includes an integrated antenna for Thread, Zigbee, and Wi-Fi data transfer. It also supports direct programming via the USB-C port through an integrated JTAG bridge.

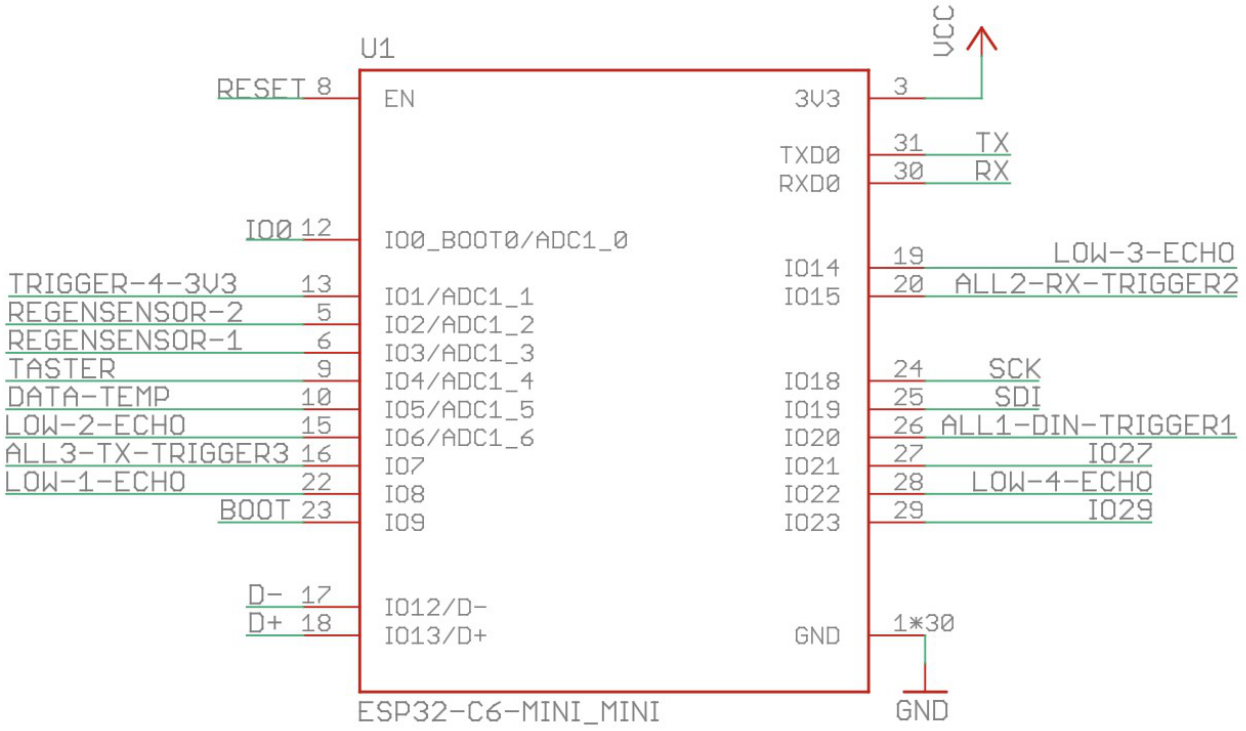


Figure 1 ESP32-C6

Buttons

Buttons to reset/boot the .
This is exactly the same buttons and resistors as on the original dev kit execpt other footprints.

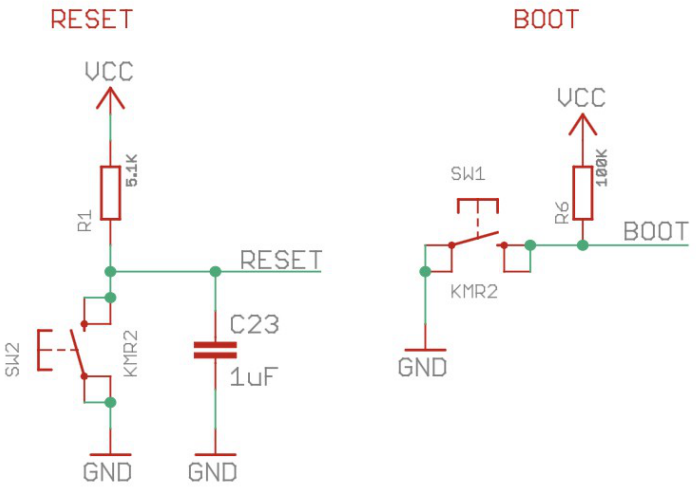


Figure 2 Reset/Boot Button

Casing Button

This Button is used to make a new connection over matter.

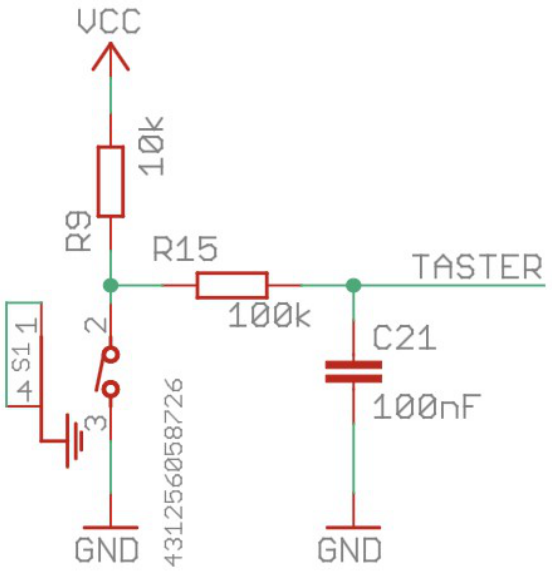


Figure 3 Casing Button

Power supply

This is the Power supply with some capacitors as bypass for the . We used the same schematic as the dev kit but deleted 1 Mosfet.

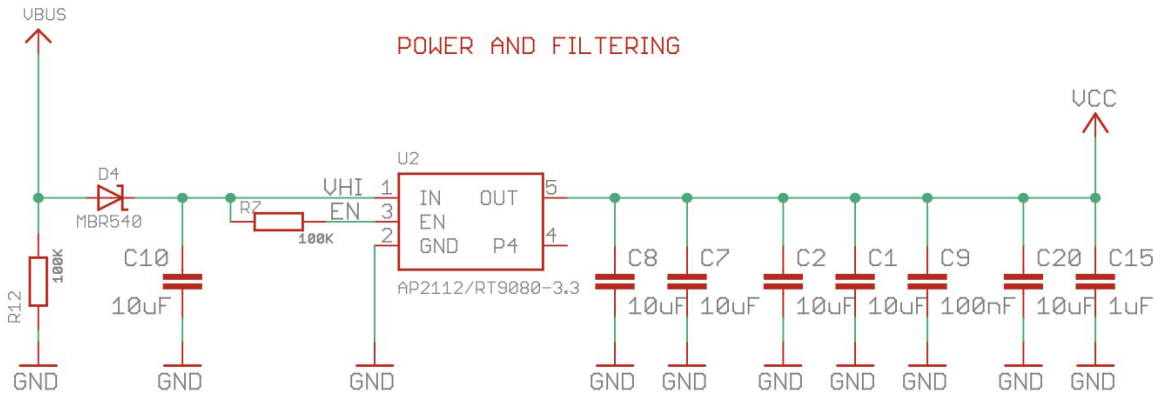


Figure 4 Power supply

USB-C Port

USB-C port supplies power and could also be used for programming. Is a other footprint than the original, because we had some issues with the milling layers and the VCC/VBUS layer because they were shorted.

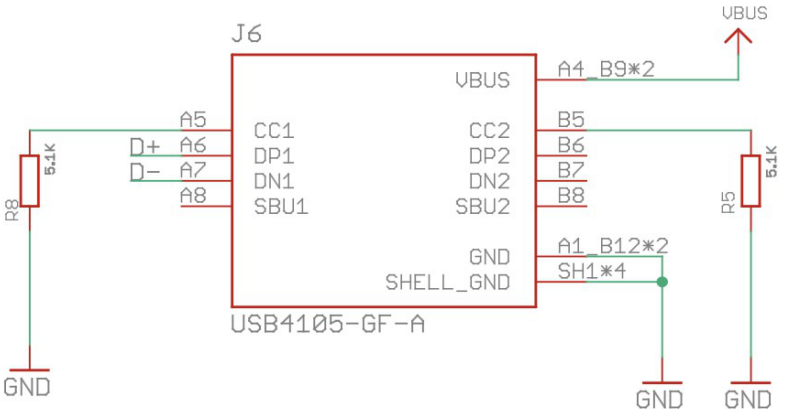


Figure 5 USB-C Port

Rain measurement station

This is our Rain detection station that measures if the reed switch makes a contact. Then we get a signal and read it out through a RC filter for swinging reduction.
We used a Connector so we can disconnect the rain station easily.

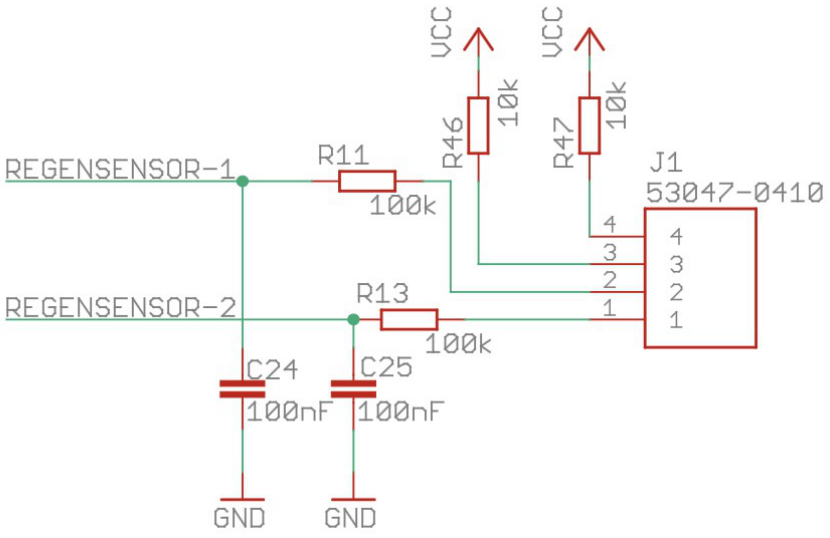


Figure 6 Rain measurement station

Leds

ws2812B LEDs are used to show a user how the air quality is. We Used the ws2812b because we had someone, we could get some for free.
100nF add to every Led to have a bypass and for voltage stabilization.

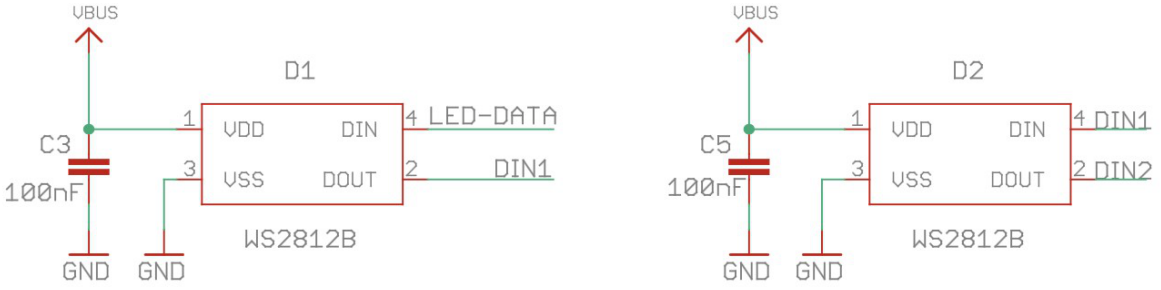


Figure 7 Leds

Programming port

This is the programming port (UART) we used for programming the ESP32-C6 .

.

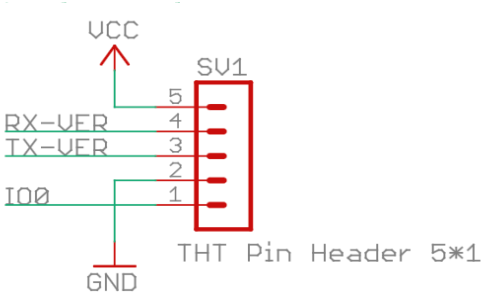


Figure 8 – External UART-Port

RX/TX swaper

This schematic shows a **TX/RX swap (crossover) circuit** using zero-ohm resistors (0R), often used for debugging or compatibility between devices with mismatched TX/RX pinouts.
Also, it is just a security feature if someone makes a mistake.

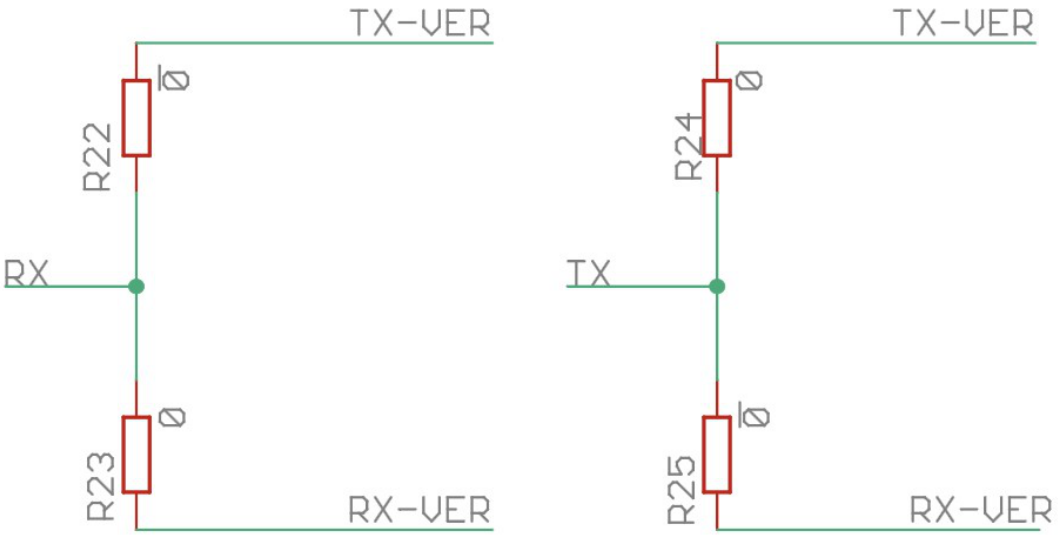


Figure 9 - RX/TX swaper

Fiducials

This are 2 reference points for the pick place machine to help with the coordinates.

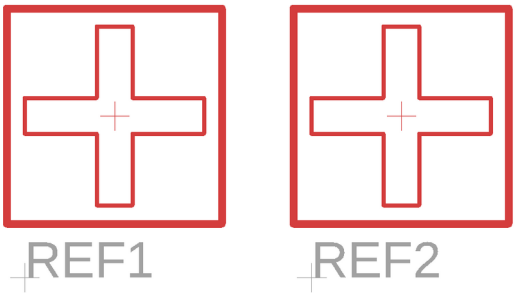


Figure 10 - fiducials

CO2 Sensor

This is the CO2 sensor which is used for the measurement of CO2. The 330R resistors are used to reduce the signal current if it were a 5V output. 100nF capacitor was added to the MH-Z19B as bypass and for voltage stabilization.

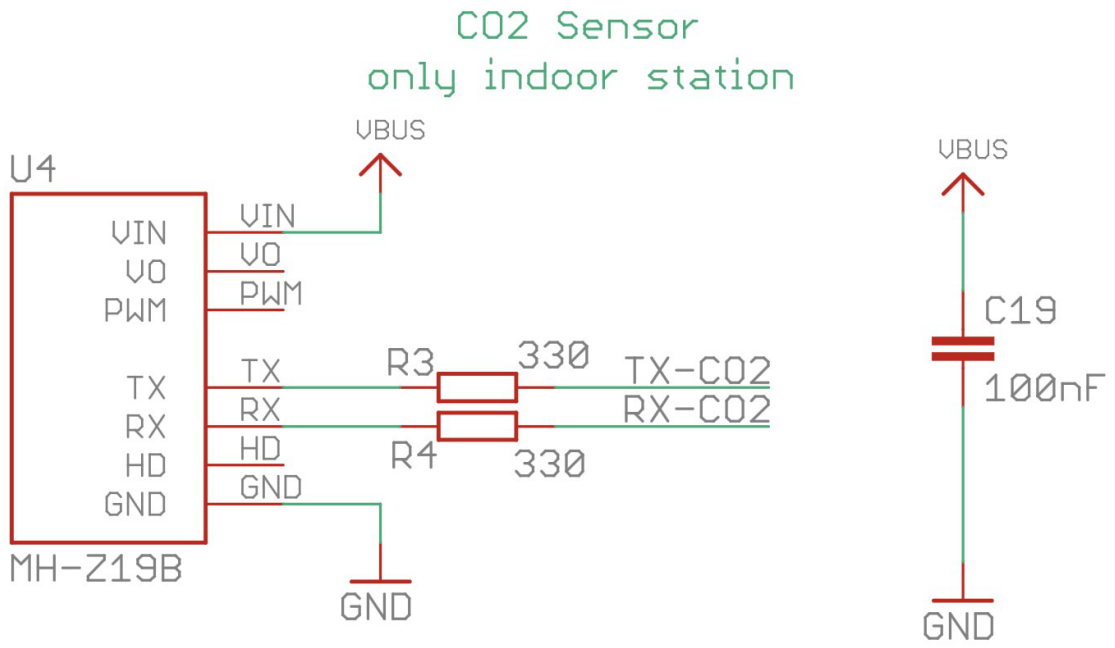


Figure 11 - co2 sensor

Brightness sensor

This sensor measures the light intensity. This sensor also has a 100nF capacitor as voltage stabilizer.

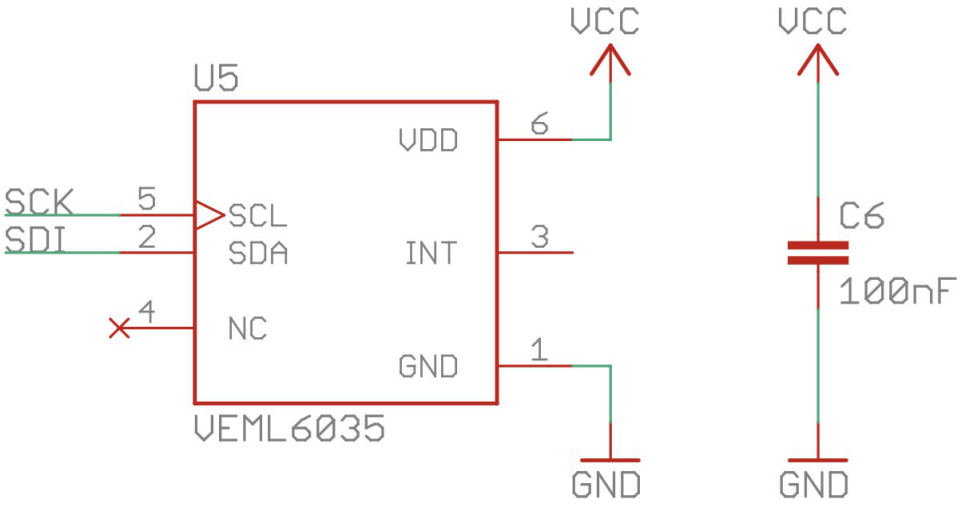


Figure 12- light sensor

BME688

The BME688 can measure gases, temperature, pressure and humidity. It is over I2C connected to the It has 2 4k7 resistors for the I2C connection.

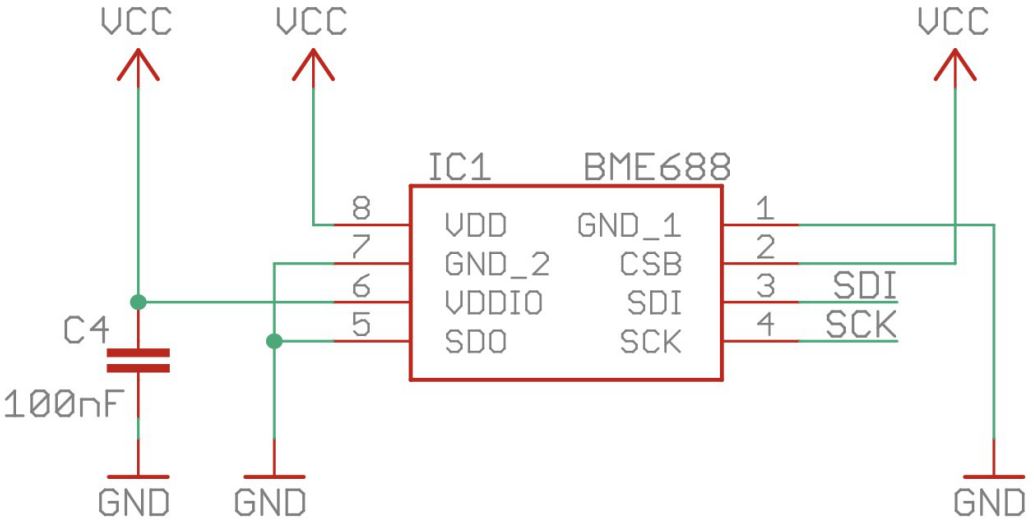


Figure 13 - Temp. sensor

DS18B20

This sensor measures the temperature more accurate than the BME688. Also, the sensor has a serial number which can be used as a serial number for a product in future.

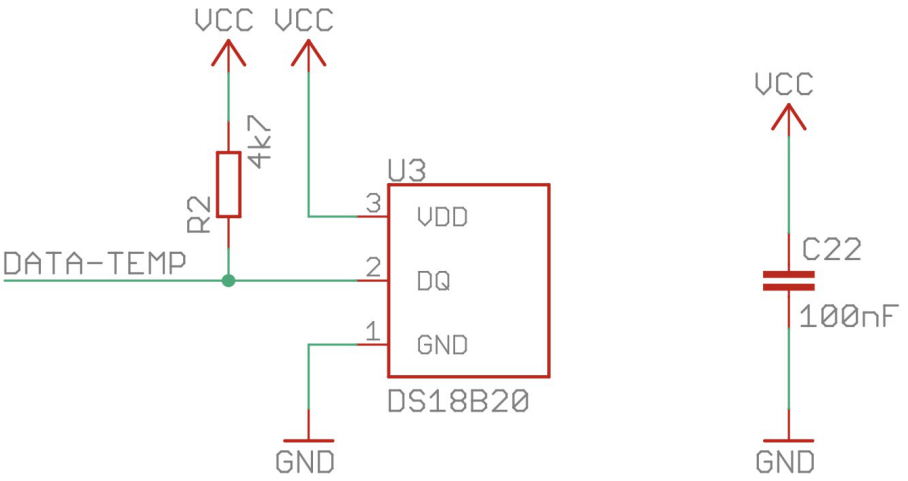


Figure 14 - BME688

5V input for ESP32-C6

The 5V input from the left side will go through a 1k resistor and through a 330R to go to the gpio pin of the **ESP32-C6 .** The Bas40-04 diode is used to let any voltage over 3V3 into the VCC supply voltage. The 330R resistor is used as a current control so there is no current over 5mA flowing into the **ESP32-C6** pins.

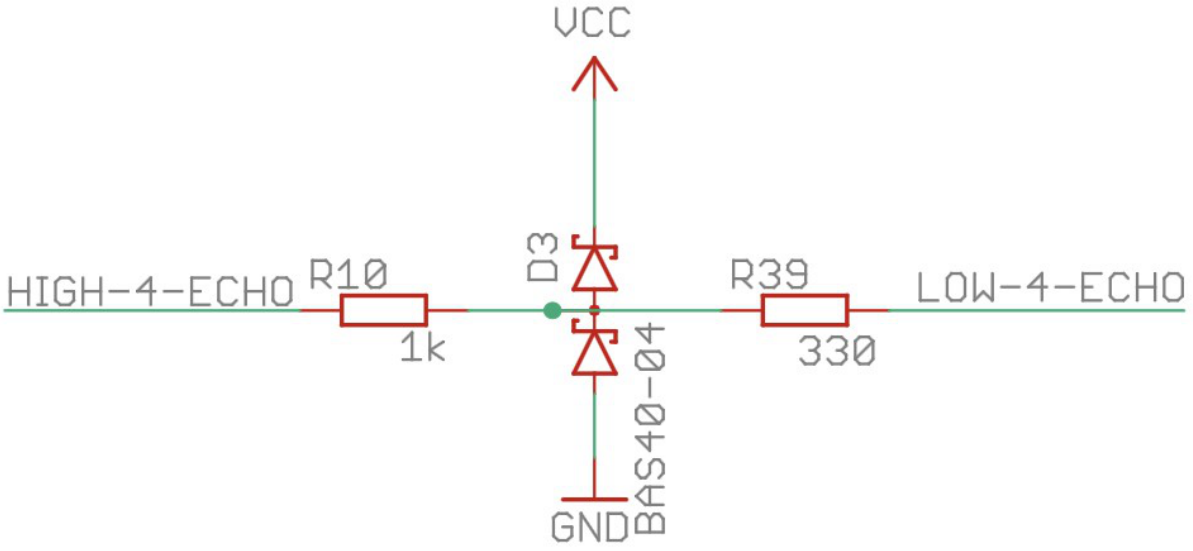


Figure 15 5V input for ESP32-C6

The 3V3 to 5V level shifter

This is the standard level shifter you can find on the internet. When the Trigger-4-3V3 input is pulled LOW, the gate source voltage is 3V3 and there is no conduction. As a result, I have no voltage drop across the resistor at the output and become a 5V signal. If the input is pulled high, the gate source voltage is 0V and a current flows through the resistor at the output and therefore has a voltage drop of 1.7 V and is therefore pulled to 3V3.

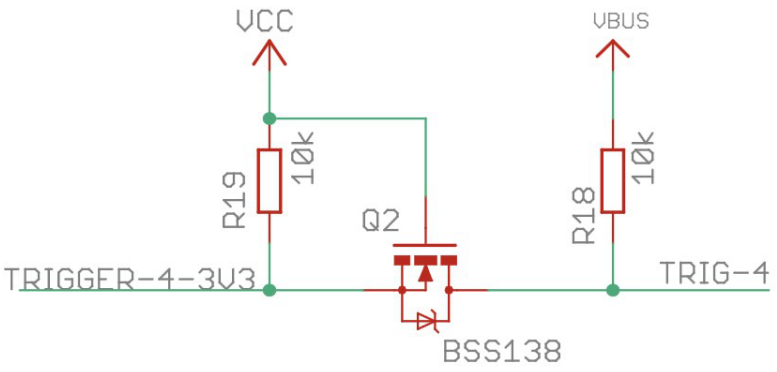


Figure 16 The 3V3 to 5V level shifter

Connector

This is the connector we choose as it had 4 Pins which are relatively easy connect and we have some wires to connect them to the ultrasonic sensor.

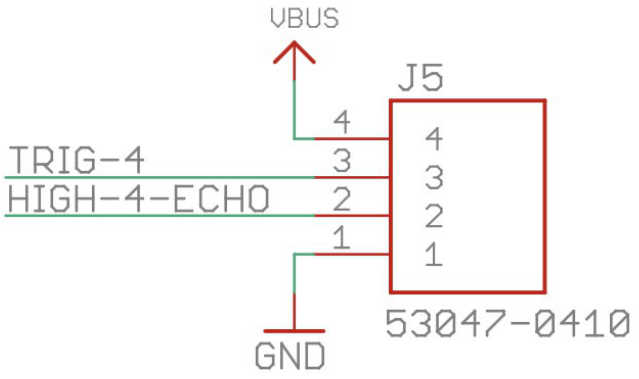


Figure 17 Connector

Layout

The Layout should be as small as possible while also making a led ring for animations. We tried to make most parts smd. We assemabled the pcb with a pick&place machine.
The solder was applied with a manual paste printer.

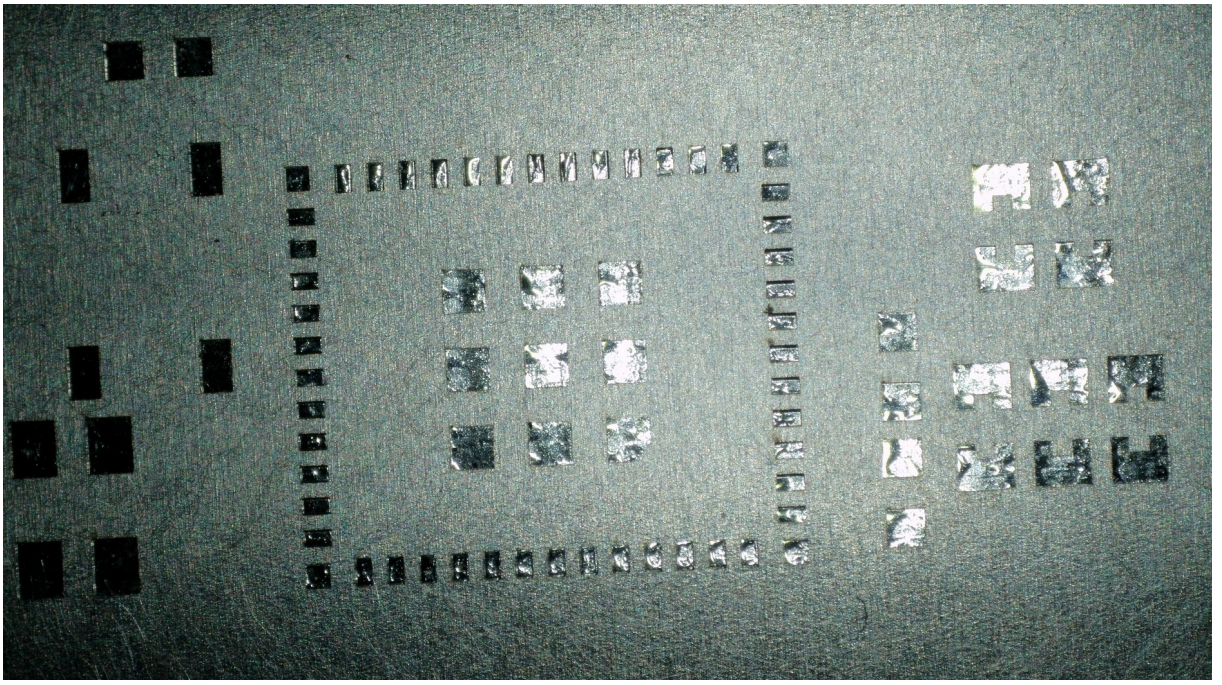


Figure 18 Soldermask

After assemabeling the part we soldered them with a reflow oven.

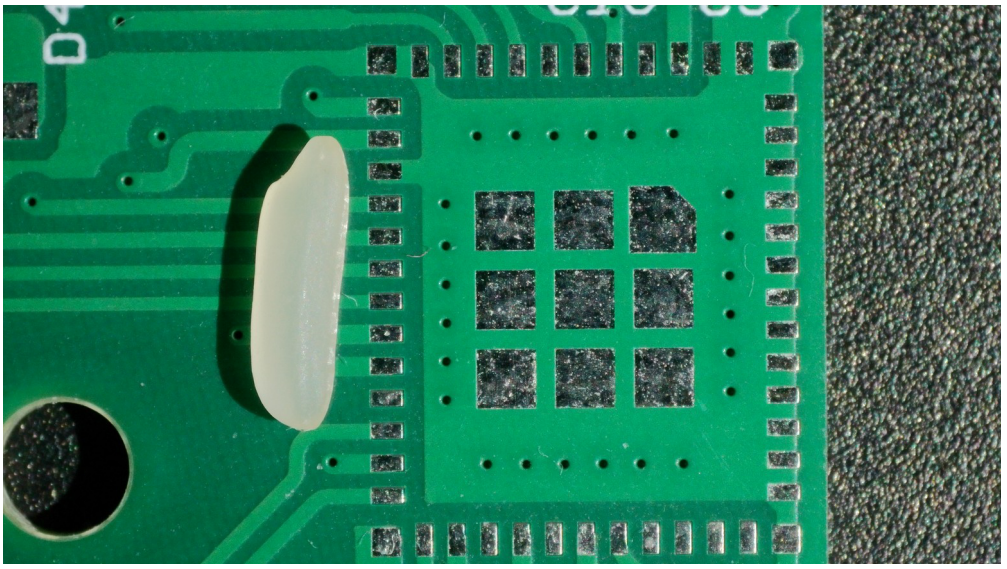
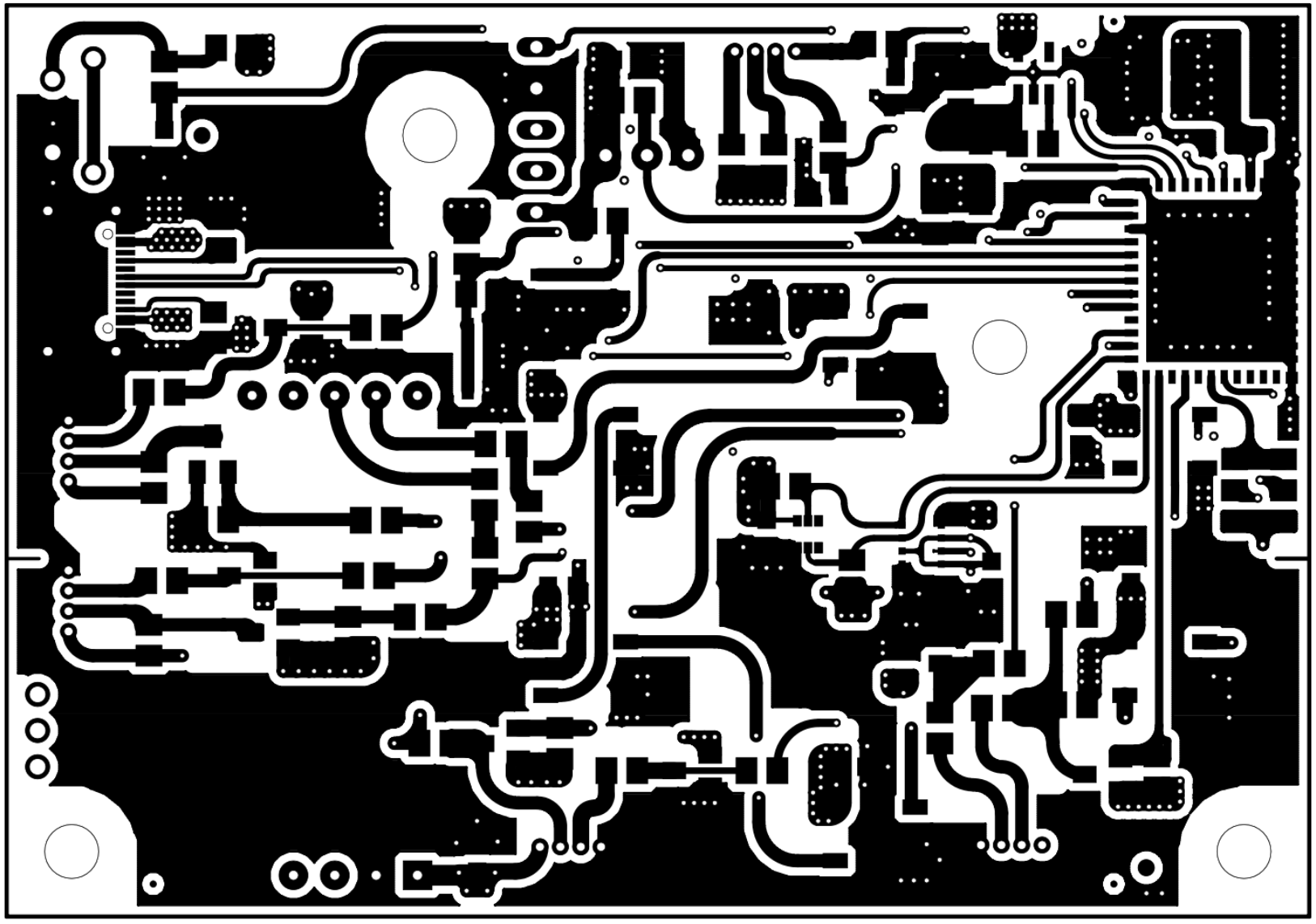


Figure 19 Comparrison to a rise corn

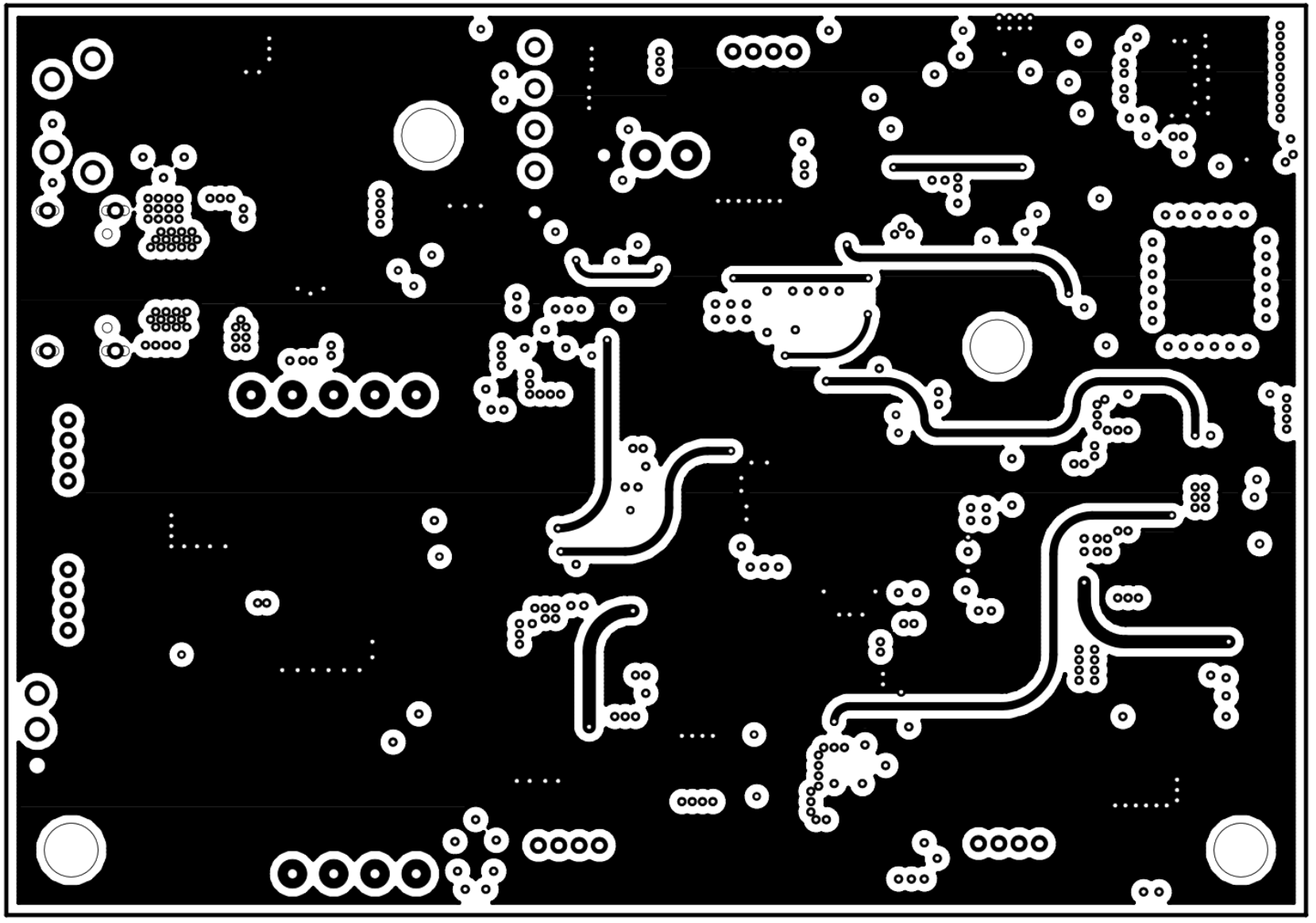
We made 2 horizontal fiducials for pick place.

We also made 3 fiducials for the pick place to measure the correction factor for the X/Y acsis of the pick place machine
All orphans were removed from the GND, VBUS and VCC layers. We added a lot of Vias for lower resistance of connections.
We tried to make the distance between the bypass capacitor and the consumer as small as possible.
We made nearly all the traces round to avoid 90° angels and lower
We added 3 testing points to the bottom left corner of the pcb also we added 4 drillings for the placing of the pcb to the 3d printed housing. The USB-C port and the switch were placed at the opposite end of the pcb to the Esp32. So, you don´t see the switch and USB-C cable, when you place the station into the room and the Esp32 antenna is then exactly directed into the room.

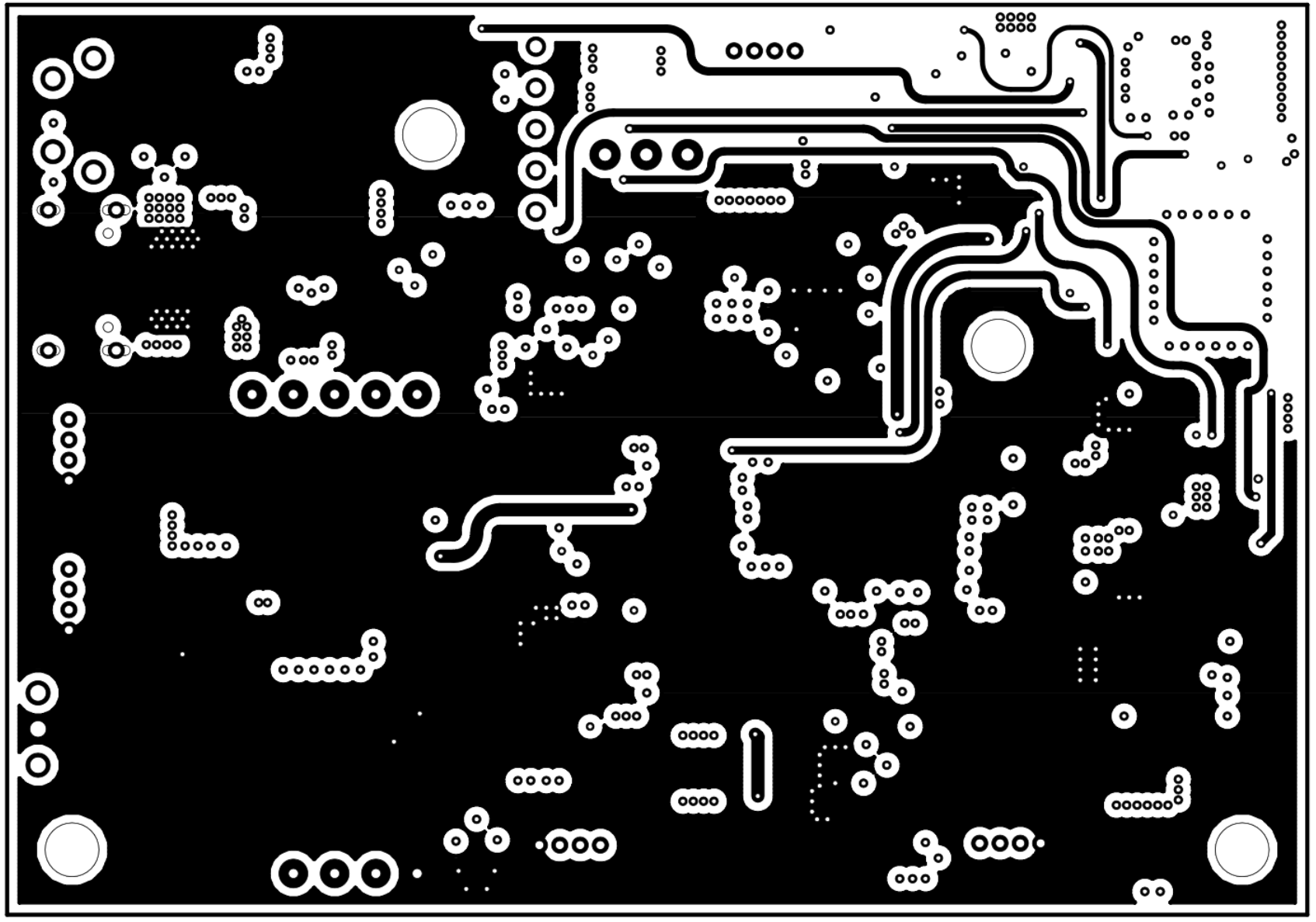
Top



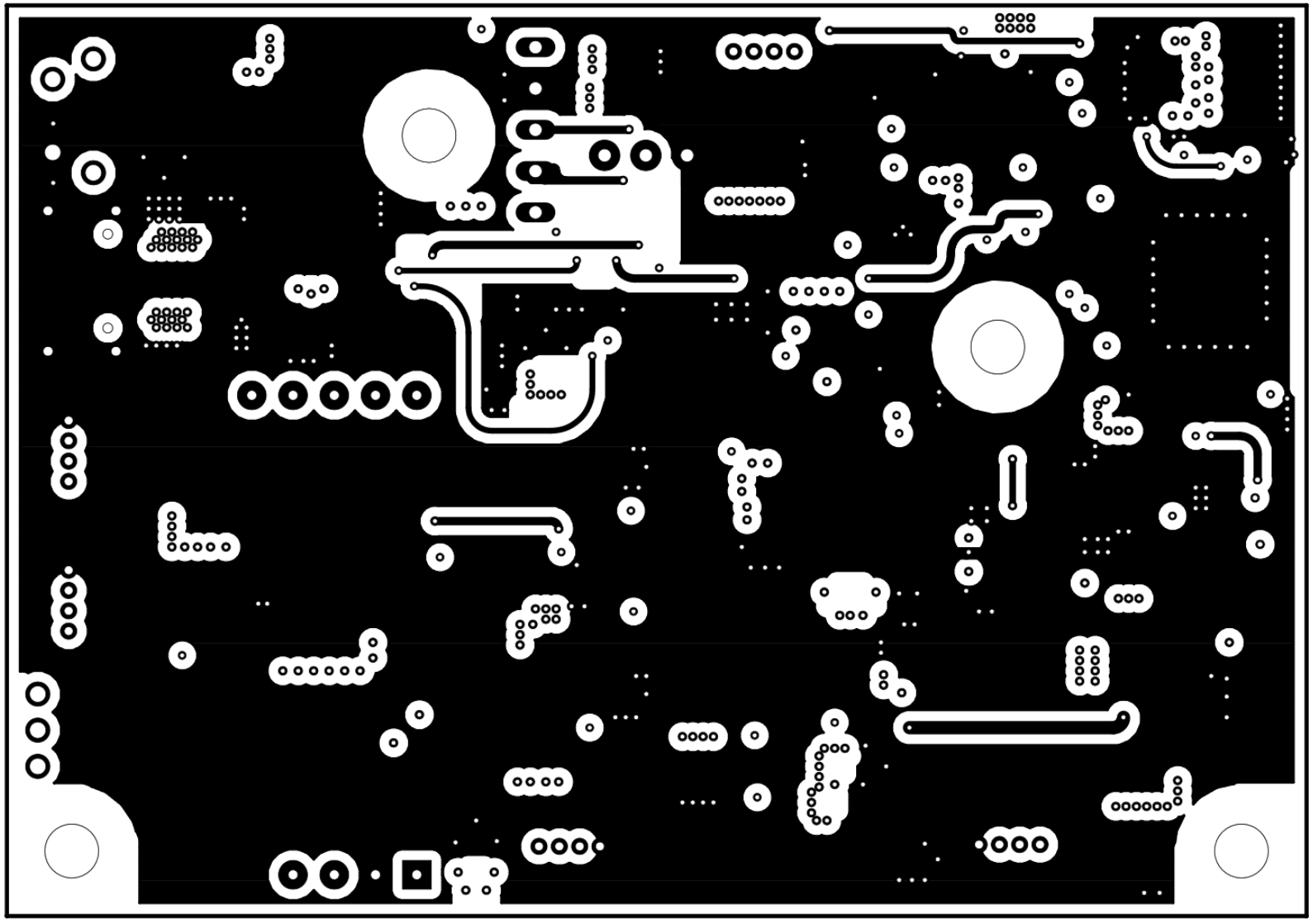
VCC



VBUS



GND



3D- Layout

This is a 3D model of our pcb with all the parts. The only mistake is that all the 0R resistors are connected eventhough some shouldn´t be placed.

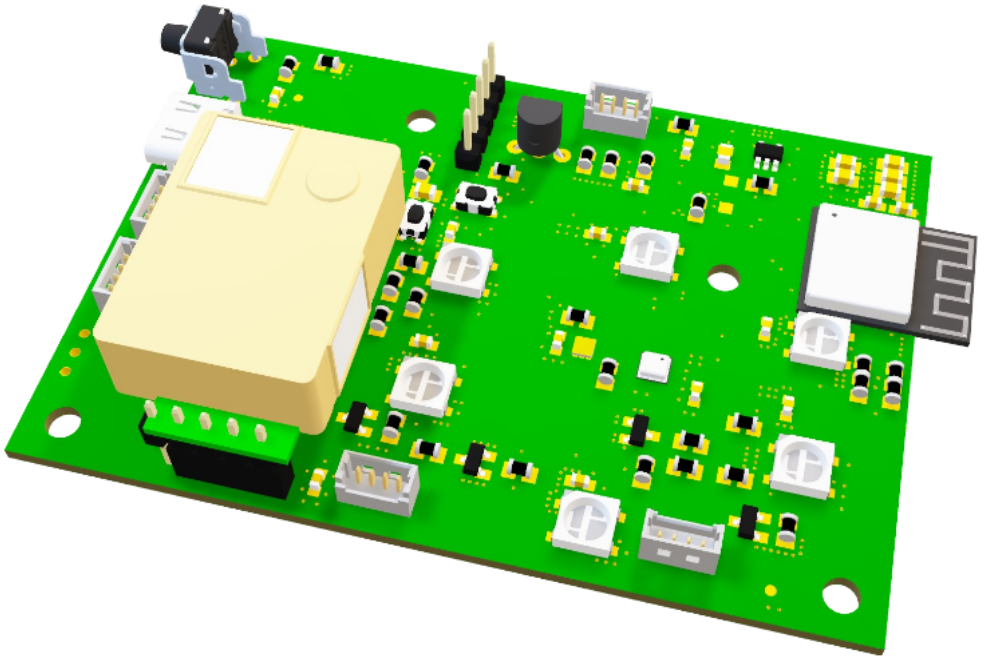


Figure 20 - PCB

PCB

https://github.com/EatingJan1/Wetterstation-Bulme-Werkstaetten-Projekt-2024-2025/tree/main/PCB

3D Design

Outdoorstation

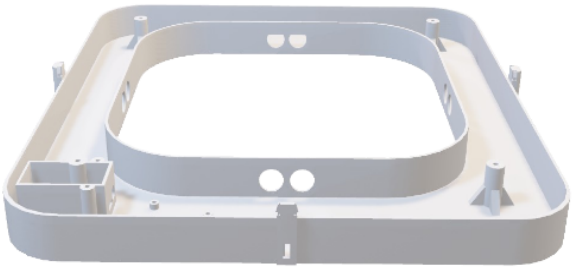


Figure 21 - Outdoor - Side View

Figure 22 - Outdoor - Top View

1. General
The “Outdoor Station” measures 368.5 mm in length by 343 mm in width, making it our largest 3D design to date. The central opening for wind measurement is a rounded square with a side length of 250 mm. Both the inner and outer walls have a uniform thickness of 2 mm, and the base plate is also 2 mm thick.

2. PCB and Mounts
The electronics PCB is secured by four form-fitting mounts. Each mount stands 4 mm above the base plate and has a wall thickness of 3 mm. The board is fastened into these mounts using M3 screws.

3. Cable Box
The cable box measures 50 mm long by 26 mm wide and serves two purposes: it provides access to the PCB’s push-button, and it houses the USB-C connector for power input. The box is open at the top and is enclosed when the lid is installed.

4. Lid Supports
The lid rests on four sturdy, cylindrical standoffs with an outer diameter of 12 mm and an inner diameter of 3.5 mm (designed for M3.5 screws). These supports ensure a precise and stable seating of the lid.

Cover – Outdoor

Figure 23 - Outdoor - Cover - Top View

Figure 24 - Outdoor - Cover - Side View

The cover has been designed so that its footprint exactly matches that of the bottom plate of the outdoor station. To ensure proper rainwater drainage, the top surface is inclined by 2°, allowing any collected water to run off reliably.

For mechanical attachment, four M3.5 holes have been precisely cut into the cover. Simultaneously, the bottom plate, excluding its outer and inner walls, features a 0.5 mm recess. This groove ensures the cover sits flush and prevents any lateral shifting.

A key design element is our logo, the blue rubber duck, which is applied to the front edge of the cover. Immediately beside it is a rectangular cutout that admits ambient light into the 3D enclosure. Its position directly beneath the circuit board housing the light sensor guarantees optimal illumination for accurate brightness measurement.

Connector

The connector provides a secure mechanical attachment and interface for the rain sensor (hyetometer) on the station. It measures 40 mm in height and 11.5 mm in width. A rectangular aperture of 4 × 9.5 mm inside ensures precise sensor alignment. Two integrated clips at the top reliably prevent the hyetometer from slipping out during operation.

Figure 25 - Outdoor - Connector

Hyetometer

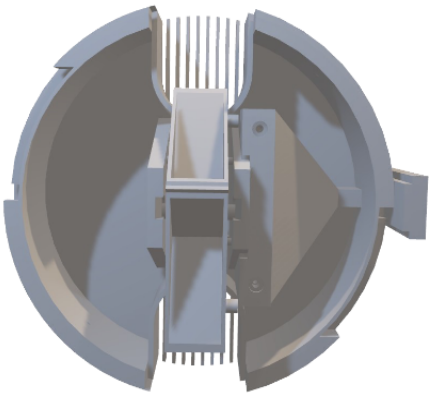


Figure 26 - Hyetometer - Top View

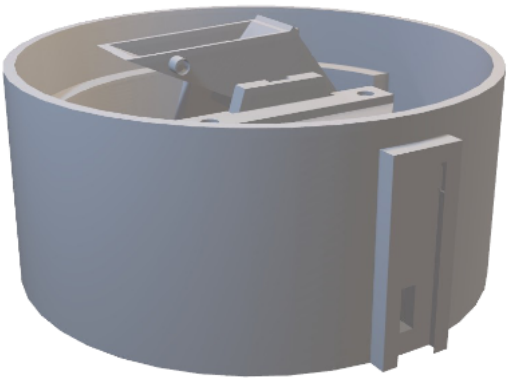


Figure 27 - Hyetometer - Side View

1. **General**
   The housing is cylindrical with a maximum outer radius of 55.00 mm. The base plate has a thickness of 5.00 mm. The overall height of the housing corresponds to the height of the inner/outer wall and is 43.50 mm.

2. **Outerwall**
   The cylindrical outer wall has a thickness of 2.50 mm, an outer radius of 55.00 mm, and extends 43.50 mm vertically above the base plate.

3. **Cable Channel**
   - **Function:**
     The cable channel provides a guided path for the wires from the reed switches to exit the housing.
   - **Description:**
     Wires originating from the reed switches are first gathered in a funnel-shaped guide and then bundled to pass through a rectangular opening measuring 15 × 12.32 mm. This exit is located on the wall where the reed switches are mounted, with a maximum available height of 35 mm for accommodating the funnel and wiring within this space.

     To facilitate assembly and later cable routing, a removable cover is integrated above the channel. The cover is secured by two M3 screws, ensuring stable cable guidance while allowing easy access if cables need repositioning or replacement. Edge protection features or rounded transitions at the funnel and at the edges of the rectangular aperture prevent damage to the wires.

Seesaw

Bracket:

- Length: 45.00 mm
- Width: 26.00 mm
- Height: 40.00 mm

Seesaw:

- Total filling quantity: 3.33 ml
- Tilling surges: 1.66 ml
- Diameter magnet holder: 5.00 mm
- Lenght: 72.00 mm
- Width: 15.00 mm
- Height: 22.00 mm

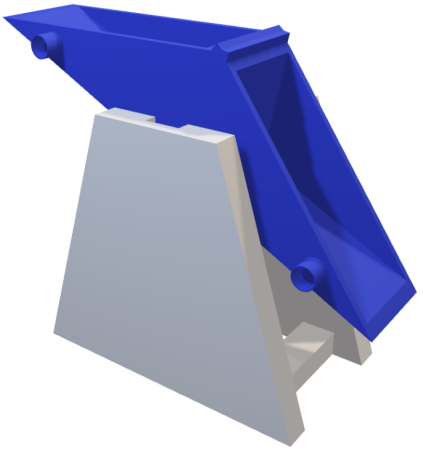


Figure 28 – Hyetometer - Seesaw

On each tilting side, a holder is provided for a magnet (3 mm diameter, 2 mm thickness) which, when tilted, reliably triggers the corresponding reed switch.

Funnel

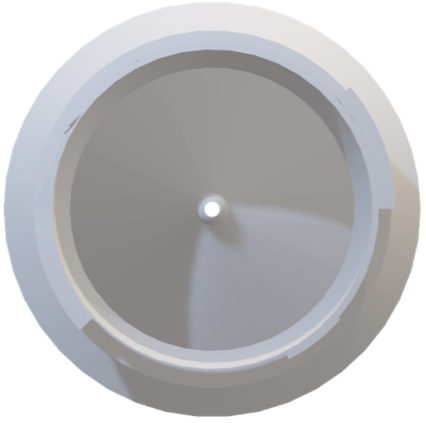


Figure 29 - Hyetometer - Funnel - Bottom View

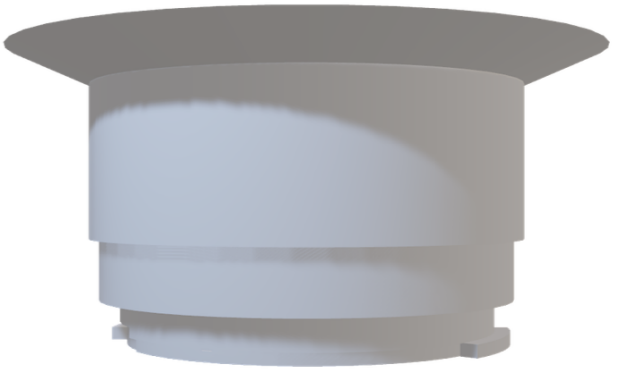


Figure 30 - Hyetometer - Funnel - Side View

The collection area of the funnel has an inner diameter of 140.5 mm with a wall thickness of 5 mm (resulting in an outer diameter of 150 mm). The funnel's overall height is 75 mm, although only 47.5 mm extends into the lower section of the rain gauge. The cylindrical body to which this collecting funnel attaches shares the same outer diameter and wall thickness. Two internal guide elements and two external lugs ensure a secure connection: the funnel is pushed onto the lower part, rotated into a defined latch position, and locked by a mechanism.

Indoorstadtion

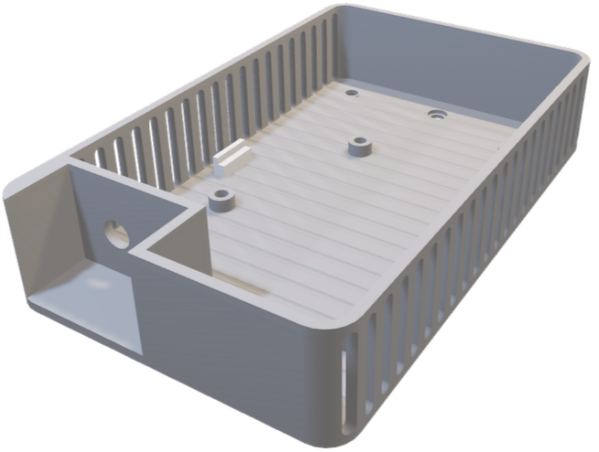


Figure 31 - Indoor - Side View

1. **General**
   The base plate is 2 mm thick. The overall exterior measures 114 mm in length, 70 mm in width, and 23 mm in height.

2. **Outer Wall**
   The continuous outer wall has a thickness of 2 mm. On both long sides, multiple slots have been added for ventilation and to admit light to the onboard light sensor. Each slot is 2.5 mm wide and 17 mm high, with a 2.5 mm gap between adjacent slots.

3. **PCB Mounting**
   The printed circuit board is secured from below by four M3 screws into small cylindrical standoffs. In addition, three retaining tabs hold the board in place during insertion, preventing it from shifting before it is fastened.

4. **Cable Box**
   A recess in one side wall provides a passage for the power cable and access to the PCB push-button. This opening is 14.5 mm deep and 28 mm wide, allowing cables to be routed cleanly and the button to be operated comfortably.

5. **Mounting Points**
   The cover is attached to the base plate with two M3.5 screws. For wall mounting of the indoor station, an additional M3.5 hole is provided on the rear, allowing the unit to be securely fixed by a screw.

Cover - Indoor

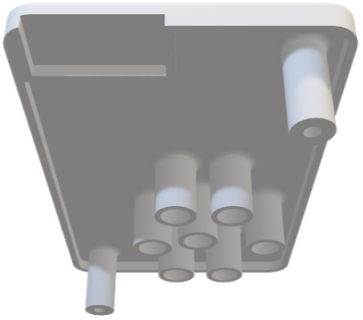


Figure 32 - Indoor - Cover - Bottom View

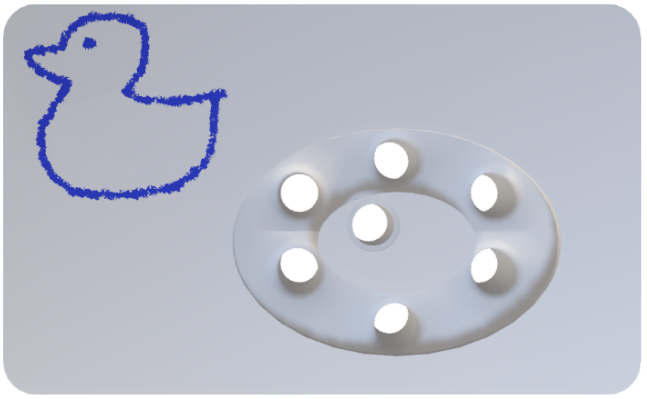


Figure 33 - Indoor - Cover - Top View

The cover features an oval cutout that is primarily used for displaying LED animations. At the center of this oval arrangement of six LEDs, there is an additional tube that extends down to the circuit board and is precisely positioned above the ambient light sensor. This tube serves to channel light directly to the sensor.

The tubes for both the LEDs and the sensor are each 13.5 millimeters deep and have a wall thickness of 1.5 millimeters. In the upper left corner of the cover, our logo, a stylized blue rubber duck, is engraved.

3D Model

https://github.com/EatingJan1/Wetterstation-Bulme-Werkstaetten-Projekt-2024-2025/tree/main/3D%20Model

Programming

Hyetometer-Reed

To measure rainfall, a custom C++ class called mwippe was developed. It simulates the behavior of a tipping bucket rain gauge that uses one or two reed switches. Every time the bucket tips due to accumulated rainwater, one or both reed switches are briefly activated. Each tipping motion corresponds to a fixed amount of rainfall — for example, 0.05 mm per tip, which is passed as the rainunit during initialization.

The class supports both single-contact and dual-contact sensors, offering flexibility and reliability depending on the mechanical setup. Its use in code is straightforward:
#include "mwippe.h"

```
mwippe hyetometer(Reed1_Pin, Reed2_Pin, 0.05); // Two-contact mode with 0.05 mm per tip

void setup()
{
    Serial.begin(115200);
}

void loop()
{
    hyetometer.runcheckerwipp(); // Check for tipping events
    float rainfall = hyetometer.getDayrain(); // Get today's rainfall

    Serial.print("Rain today [mm]: ");
    Serial.println(rainfall);

    delay(500); // Example only - in practice, use in a task or timed loop
}
```

The total daily rainfall can be retrieved using getDayrain(). You can manually set or reset the daily amount using setDayrain() and resetRaintoday(). The runcheckerwipp() method should be called regularly — for example, in the main loop or inside a periodic task — to detect and count tipping events.

This class offers an independent and adaptable way to interface with mechanical rain gauges without relying on third-party libraries. It can be easily integrated into larger environmental monitoring systems and adjusted to match different sensor hardware thanks to its flexible design.

MCAnemometer

The Anemometer class implements a complete ultrasonic anemometer logic for Arduino-compatible systems.
It measures wind speed and direction by using four ultrasonic sensor pairs (north, east, south, and west) arranged orthogonally. This allows the system to calculate wind vectors based on the time it takes for ultrasonic pulses to travel between each direction pair.
Each pair consists of a trigger and echo pin. The constructor requires these 8 pins (4× trigger + 4× echo) and the physical distance between each sensor pair in millimeters.
The class supports either symmetrical (distance) or asymmetrical (distanceNoSo, distanceWeEa) sensor setups.

Wind speed and direction are calculated by comparing the differences in travel time of sound across the opposing sensor pairs (north–south and west–east).
A calibration routine is included to account for environmental and mechanical deviations.

The following example shows how to use the class in code:

```
#include "MCAnemometer.h"

Anemometer windSensor(Trigger_1, Echo_1, Trigger_2, Echo_2, Trigger_3, Echo_3, Trigger_4, Echo_4, distance_wind);

void setup()
{
    Serial.begin(115200);
    windSensor.calibrate(); // optional, improves accuracy
}

void loop()
{
    windSensor.readstate(); // reads ultrasonic measurements and calculates wind data

    float speed = windSensor.getspeed();
    float gust = windSensor.getgustswind();
    float direction = windSensor.getangle();

    Serial.print("Speed [mm/s]: ");
    Serial.print(speed);
    Serial.print(" | Gust: ");
    Serial.print(gust);
    Serial.print(" | Direction [°]: ");
    Serial.println(direction);

    delay(1000); // adjustable sample rate
}
```

The readstate() method should be called regularly to trigger new ultrasonic measurements and update wind data. You can retrieve the current wind speed (getspeed()), strongest gust since the last read (getgustswind()), and wind direction in degrees (getangle()).
This class is well suited for weather stations or experiments requiring low-cost wind measurement without moving parts. Its modular structure also allows for custom calibration and further extensions such as averaging, filtering, or logging.

LEDanimation

The LEDAnimation class provides an easy interface to control NeoPixel LEDs using the Adafruit NeoPixel library combined with FreeRTOS. It supports several predefined animations, allows color customization, and offers adjustable animation speed. Animations run independently in a dedicated FreeRTOS task, ensuring non-blocking operation.

Used Libraries:

- Adafruit_NeoPixel: For driving the LED strip.
- FreeRTOS (integrated in Arduino framework): To run animations asynchronously in the background.

Animations:

Table 1 - Animation Types

AnimationType	Description
None	Turns all LEDs off.
CriticalWarning	Blinking red warning signal.
StartUp	LEDs light up sequentially.
Rotating	Rotating light effect.
Connection	Pulsating brightness effect.
Static	Constantly lit LEDs with set color.

Usage Example:

```
#include <Adafruit_NeoPixel.h>
#include "LEDAnimation.h"

Adafruit_NeoPixel strip(10, 6, NEO_GRB + NEO_KHZ800);
LEDAnimation led(strip, 10);

void setup()
{
    strip.begin();
    strip.show();
    led.setColor(strip.Color(0, 0, 255));          // Set color to blue
    led.setDelaySpeed(5);                          // Set animation speed
    led.setAnimation(AnimationType::StartUp);      // Start the startup animation
}
```

Public Methods

```
1. LEDAnimation(Adafruit_NeoPixel strip, int count);
2. ~LEDAnimation();
3.
4. void setAnimation(AnimationType type);
5. void setColor(uint32_t color);
6. void setDelaySpeed(uint delay);
```

- Constructor / Destructor: Initializes the object and starts a FreeRTOS task for running animations asynchronously. The task is stopped on destruction.
- setAnimation(AnimationType type): Activates a predefined animation.
- setColor(uint32_t color): Sets the color used in color-dependent animations.
- setDelaySpeed(uint delay): Adjusts the animation speed; lower values correspond to faster animations.

Colors:

Colors are specified using the NeoPixel strip.Color(r, g, b) method. Example:

```
1. led.setColor(strip.Color(0, 0, 255)); // Blue
```

This color is used for all animations except CriticalWarning

Speed Control:

The animation speed is controlled via setDelaySpeed(uint delay). The delay value influences internal timing, with smaller values resulting in faster animations:

```
1. led.setDelaySpeed(2); // Fast animation speed
```

Brightness:

Brightness is managed using setBrightness() from the Adafruit NeoPixel library, ranging from **0** (off) to **255** (maximum brightness). Certain animations, such as Connection, dynamically adjust brightness.
Note: Ensure your power supply can provide sufficient current; a single RGB LED at full brightness can consume up to 60mA.

Requirements:

- The Adafruit NeoPixel library must be installed.
- Requires a platform with FreeRTOS support (e.g., ESP32).

AveragePerTime

The AveragePerTime class provides an efficient way to store and process timestamped float values in real time, allowing calculations of sums, averages, and counts over a configurable period. It is designed for applications like sensor data smoothing, time-based analytics, or threshold detection on embedded systems like ESP32 or Arduino boards.

- Stores a time-based history of float values.
- Allows querying for:
  - Sum over a defined period (e.g., last 5 seconds)
  - Average of values over a period
  - Count of entries within a time window
- Automatically cleans up old entries beyond the maximum history (default: 5 hours).

Dependencies:

- Arduino framework: Uses millis() for time tracking.
- <vector> / <algorithm>: For dynamic data storage and cleanup.

Usage Example:

```
#include "average_per_time.h"

AveragePerTime avg;

void loop()
{
    float sensorValue = analogRead(A0);
    avg.add(sensorValue);

    float avgValue = avg.getAverage(60000); // Average over the last 60 seconds
    Serial.println(avgValue);
}
```

Public Interface:

```
void add(float value);
float getSum(uint32_t periodMs);
float getAverage(uint32_t periodMs);
int getCount(uint32_t periodMs);
```

- add(value): Adds a new float value with the current timestamp.
- getSum(periodMs): Returns the sum of all values added within the last periodMs milliseconds.
- getAverage(periodMs): Returns the average value for the period. Returns 0.0 if no values are in range.
- getCount(periodMs): Returns the number of values stored within the time window.

Matter – Arduino esp32 matter extra endpoints

We are working with the Matter library provided by Espressif, integrated into the Arduino version (arduino-esp32).
However, the number of available endpoints in the official library is still limited. For instance, endpoints for rain detection, CO_2 concentration, ambient light, or flow sensors are not yet available. To close this gap, we have created our own addon that extends this lib with additional endpoints

extra endpoint includes

```
# #include <Matter.h> // extra endpoint requires this
#include <WiFi.h> //Matter.h requires this (or Matter.h requires Openthread)
#include <scr/MatterAirQualitySensor.h>
#include <scr/MatterFlowSensor.h>
#include <scr/MatterLightSensor.h>
#include <scr/MatterRainSensorSensor.h>
```

Overview all endpoints

Official Espressif Endpoints (arduino-esp32 Matter library)

Table 2 - Matter.h endpoints

Endpoint	Einheit / Typ	Beschreibung
MatterGenericSwitch	Boolean / State	Schalter, unterstützt mehrere Schaltlogiken
MatterOnOffLight	Boolean	Einfaches Licht: An/Aus
MatterDimmableLight	Prozent [%] (0–100)	Dimmbares Licht
MatterColorTemperatureLight	Farbtemperatur [Kelvin]	Weißton einstellbar
MatterColorLight	RGB-Farbe / HSL	Licht mit Farbsteuerung
MatterEnhancedColorLight	RGB / erweiterte HSL	Licht mit erweiterten Farbräumen
MatterFan	Prozent [%] Geschwindigkeit	Lüftersteuerung
MatterTemperatureSensor	Grad Celsius [°C]	Temperaturmessung
MatterHumiditySensor	Relative Luftfeuchtigkeit [%]	Luftfeuchtesensor
MatterContactSensor	Boolean (offen/geschlossen)	Tür-/Fenstersensor
MatterPressureSensor	Druck [hPa]	Luft-/Systemdruck
MatterOccupancySensor	Boolean (Bewegung erkannt)	Präsenz-/ Bewegungssensor
MatterOnOffPlugin	Boolean (An/Aus)	Steckdose, Schaltaktor
MatterThermostat	Temperaturregelung [°C], Modi	Heizungs-/ Klimasteuerung

Custom Endpoints (arduino-esp32-matter-extra-endpoints)

Table 3 - Extra ednpoints

Endpoint	Unit / Type	Description
MatterAirQualitySensor	CO_2 [ppm], AirQualityIndex [1–5]	Air quality sensor, partial CO_2/VOC support
MatterRainSensor	Boolean (rain: yes/no)	Rain detection switch
MatterAmbientLightSensor	Illuminance [Lux]	Light level sensor
MatterFlowSensor	Flow rate [m^3/min]	Flow sensor for water, air, etc.

⚠ Note on AirQuality:
The current implementation supports only CO_2 via the MatterAirQualitySensor. Due to the lack of additional data such as VOCs or PM2.5, the Air Quality Index (AQI) is calculated using CO2 only

Method Overview

All endpoints – both official and custom – follow a consistent method structure:

begin(...)

Initializes the endpoint with a starting value.

```
1. airSensor.begin();            // Don't support start Value
2. rainSensor.begin(false);           // Rain state
3. lightSensor.begin(100.0);          // Lux
4. flowSensor.begin(0.0);             // m3 per minute
```

set...(...)

Updates the current sensor value internally and publishes it to the Matter layer.

```
1. airSensor.setCO2(co2);             // CO₂ in ppm
2. rainSensor.setRainDetected(true);  // Rain detected
3. lightSensor.setIlluminance(lux);   // Light level in Lux
4. flowSensor.setFlow(flow);          // Flow rate in m3/min
```

get...()

Returns the last set value.

```
1. int currentAQI = airSensor.getAirQuality()
2. float currentCO2 = airSensor.getCO2();
3. bool isRaining = rainSensor.getRainDetected();
4. float currentLux = lightSensor.getIlluminance();
5. float flowRate = flowSensor.getFlow();
```

BME68x

To measure environmental data such as temperature, humidity, air pressure, and air quality, this project uses a sensor from the BME68x series by Bosch. Communication with the sensor is handled via the official BME68x library provided by Bosch Sensortec. The library supports both I2C and SPI interfaces and allows for easy initialization and periodic reading of all relevant values. In our project, the sensor is connected via I2C. The library offers structured functions for reading and processing the current sensor data.

SparkFun VEML6030 Ambient Light Sensor

For reading ambient light levels from the VEML6030 sensor, we rely on the official SparkFun VEML6030 library. It handles I2C communication and provides user-friendly functions to adjust gain, resolution, and integration time. The library also includes a built-in method to retrieve the lux value, which is automatically calculated from the raw sensor data. This abstraction reduces complexity and speeds up development by eliminating the need for manual data conversion.

MHZ19 & SoftwareSerial

To monitor CO_2 levels, we use the MH-Z19 infrared gas sensor. Communication is handled via a serial interface using the MHZ19 library by Jonathan Dempsey, which provides a straightforward way to initialize the sensor and retrieve CO_2 readings. For the serial connection, we rely on the SoftwareSerial library by Dirk Kaar, allowing us to define custom GPIO pins for flexible integration.

pinout_brd_V1:0.h

The file pinout_brd_V1:0.h serves as a configuration file that maps the microcontroller's GPIO pins to the various hardware components used in the project. Using #define directives, specific pins are assigned meaningful names for interfaces such as SPI (SDI, SCK), I2C addresses of sensors (BME_ADDR, AL_ADDR), serial communication with the CO_2 sensor (TX_CO2, RX_CO2), digital inputs for buttons and reed switches, and trigger/echo pins for ultrasonic sensors. This structured approach improves code readability and avoids hardcoded pin numbers scattered throughout the source files.

One of the main benefits of this setup is its modularity. When the hardware layout changes—such as when using a new PCB version—only this one header file needs to be updated or replaced. The rest of the software remains

shortens development time. This makes it easy to reuse the same software across different hardware platforms simply by switching to the appropriate pinout definition. Overall, this structure enhances flexibility and scalability within the project.

Reasons for decisions

Matter

Instead of using a traditional web server, I deliberately chose the modern smart home standard Matter. The main reason: Matter is cross-platform, future-proof, and user-friendly. While web server-based devices require individual IPs and separate web pages, a Matter device can be integrated directly into existing smart home apps like Apple Home, Google Home, or Alexa – all devices in one app, no extra software needed.

Matter also includes built-in security features, firmware updates, and multi-admin support – things that would have to be implemented manually with a web server approach.

Universal PCB

A costum PCB is a good and easy way to get all the sensors and parts on 1 PCB that is easily manufacturable with only smd parts except for the programming port, the

Ultrasonic instead of windweel

Mechanical wind sensors like weather vanes with windmills suffer from wear and tear due to moving parts. They respond slowly, are vulnerable to dirt, ice, or mechanical jams .

Ultrasonic sensors have no moving parts and measure wind speed and direction precisely using time-of-flight differences. This makes them low-maintenance, durable, and fast-reacting – ideal for long-term outdoor use.

ESP-C6

The ESP32-C6 was chosen because it natively supports Thread (IEEE 802.15.4) – a key requirement for Matter over Thread. While the ESP32-S3 offers solid performance, it lacks integrated Thread support, making it less suitable for modern smart home protocols like Matter.

Second temperature sensor

The Bme688 uses a lot of power and gets warm so wie have a second one (ds18b20).

Manuel

Indoor

Connect the Device

Simply scan the Matter QR code using your preferred smart home app (e.g. Apple Home, Google Home, Alexa).
Follow the instructions in the app.

🔧 Note:
Automatic Wi-Fi provisioning is not yet supported in the current software version.

Toggle Co2 lgiht
A short button press toggles the CO_2 visualization via LED on or off

Factory Reset
Press and hold the **button for 20 seconds** to reset the device and remove it from any existing Matter networks.

CO_2 Sensor Calibration
Press and hold the button for 10 seconds to recalibrate the CO_2 sensor.
➤ Best done in fresh outdoor air.

Outdoor

Connect the Device
Simply scan the Matter QR code using your preferred smart home app (e.g. Apple Home, Google Home, Alexa).
Follow the instructions in the app.

🔧 Note:
Automatic Wi-Fi provisioning is not yet supported in the current software version.

Factory Reset
Press and hold the **button for 20 seconds** to reset the device and remove it from any existing Matter networks.

Ultrasonic wind sensor Calibration
Press and hold the button for 10 seconds to recalibrate the ultrasonic wind sensor sensor.
➤ Please do this when there is no wind at all, so the sensor can set the correct zero point.

Feedback

Learning Effects

Don’t heat PVC

We wanted to use a matte transparent film or sheet as a cover for the LEDs and the light sensor in our indoor station. In a laser material test set, we found a PVC film that we liked visually. However, during our online research for laser settings, we discovered that heating PVC releases toxic fumes. Therefore, we decided to cut the material by hand instead.

Matter on ESP32

Everything takes longer than expected

What went badly / self-criticism

Not connected GND testing point

3D Design could have been better

convey ideas or concepts in such a way that they are understood

What went well

PCB worked in first version

untouched, which simplifies maintenance, reduces the likelihood of errors, and

BME688 worked without testing

3D printing the housing

Costs

Table 4 - costs

PCS	Name	Producer	Shop	Price per pcs.	Total Price	Price for Members
6	Ultrasonic sensor	diymore-IOT	www.amazon.de	€ 1,44	€ 8,66	€ 2,89
8	Ultrasonic Transceiver	Hailege	www.amazon.de	€ 2,27	€ 18,12	€ 6,04
3	ESP-C6 DevKitC-1	ESPRESSIF	www.digikey.com	€ 8,00	€ 24,00	€ 8,00
3	ESP32-C6-MINI-1-H4	Digikey	www.digikey.com	€ 3,00	€ 9,00	€ 3,00
3	BME688	Bosch	www.mouser.com	€ 8,00	€ 24,00	€ 8,00
1	3D Print water measurement	3D Printer	----------	€ 4,00	€ 4,00	€ 1,33
1	3D Print outdoor Station	3D Printer	----------	€ 10,78	€ 10,78	€ 3,59
1	3D Print indoor Station	3D Printer	----------	€ 0,73	€ 0,73	€ 0,24
5	Main PCB	JLCPCB	JLCPCB.coom	€ 10,00	€ 50,00	€ 16,67
15	M3.5 x 5mm Metal screws	AERZETIX	www.amazon.de	€ 0,59	€ 8,87	€ 2,96
1	Resistors/capacitors		Lager	€ 2,00	€ 2,00	€ 0,67
30	Reed contact	AZ Delivery	www.amazon.de	€ 0,25	€ 7,55	€ 2,52
1	CO2	Winson	www.reichelt.com	30	30	€ 10,00
5	Various small components		mouser & digikey	€ 11,00	€ 55,00	€ 18,33

Table 5 - Outdoor Costs

PCS	Description	Producer	Total Price
1	Various small components	----------	€ 11
1	BME688	Bosch	€ 8
1	ESP32-C6-MINI-1-H4	Digikey	€ 3
1	3D Print water measurement	----------	€ 4
1	3D Print outdoor Station	----------	€ 10,78
4	Ultrasonic Sensor	Diymore-IOT	€ 5,76
1	Main-PCB	JLCPCB	€ 10
2	Reed contact	AZ Delivery	€ 0,5
10	M3.5 x 5mm Metal screws	AERZETIX	€ 5,9

Table 6 - Outdoor Costs

PCS	Description	Producer	Total Price
1	Various small components	----------	€ 11
1	CO2	Winson	€ 30
1	BME688	Bosch	€ 8
1	ESP32-C6-MINI-1-H4	Digikey	€ 3
1	Main-PCB	JLCPCB	€ 10
1	Indoor Gehäuse	----------	€ 0,73

Table 7 - per member

Total Costs	227,71 €
sponsors	€ 110,00
Total Costs per Member	39,24 €

Directories

Abbildungsverzeichnis

Table directory