

# Fachbericht

## Wetterstation mit Solar Energie

Windisch, 18. Januar 2019

<b>Hochschule</b>	Hochschule für Technik - FHNW
<b>Studiengang</b>	Elektro- und Informationstechnik
<b>Autor/-en</b>	Mischa Knupfer, Andres Minder
<b>Betreuer</b>	Prof. Dr. Taoufik Nouri
<b>Auftraggeber</b>	Prof. Dr. Taoufik Nouri
<b>Version</b>	1.2

## **Abstract**

Climate and weather data are the main sources to determine plots for specific plants or plant species for a specific climate. Because of that, farmers need to know the climate and weather data to optimize their work. Swiss farmers have the advantage of getting this information through a federal agency, which does not apply to farmers in subtropical areas. To provide these information for farmers in subtropical areas, a low-priced mobile weather station is required. This project should design a prototype of a weather station, which can record data for air temperature, Rainfall, wind strength and hours of sunshine. In addition to that, the DS3231 real time clock (RTC) should generate a timestamp to the data before it is stored in the data memory. The core of the weather station is the microcontroller ArduinoMega2560, which contains the program code and manages the data storage. The wind strength is being measured by the WD123 from Froggit, the rainfall by an ombrometer from Misol and the air temperature by the BME280 from Bosch that measures also humidity and pressure of the air. Moreover, a wind vane (p/n 80422) from Argent Data Systems allows the weather station to determine the wind direction. Those gained data points are stored on an internal microSD card.

The mobile weather station is able to provide data for temperature, humidity and pressure of the air as well as the Rainfall and the strength and direction of the wind. Furthermore, the mobile weather station is also able to store the data points with timestamp on the microSD card. In a further project, a battery will be implemented which will be supported by photovoltaic. Furthermore, GPS and data query over SMS (GSM) will also be implemented in that further project.

Key Words: mobile weather station, sensors, microSD card, GPS, SMS (GSM)

## Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Auftragsbeschreibung</b>	<b>2</b>
<b>3 Ziele</b>	<b>3</b>
<b>4 Konzept</b>	<b>4</b>
4.1 Prototyping . . . . .	6
<b>5 Interfaces</b>	<b>8</b>
5.1 SPI (Serial Peripheral Interface) . . . . .	8
5.2 I <sup>2</sup> C . . . . .	8
<b>6 MCU</b>	<b>9</b>
<b>7 RTC</b>	<b>10</b>
7.1 Implementation in die Firmware . . . . .	11
<b>8 Sensoren</b>	<b>12</b>
8.1 Ombrometer . . . . .	12
8.2 Anemometer . . . . .	16
8.3 Windrichtungsgeber . . . . .	18
8.4 BME280 . . . . .	20
<b>9 Datenspeicherung</b>	<b>22</b>
9.1 Breakoutboard . . . . .	22
9.2 µSD-Karte . . . . .	23
9.3 Implementation in die Firmware . . . . .	24
<b>10 Firmware</b>	<b>25</b>
10.1 Development Environment . . . . .	25
10.2 UML-Diagramm . . . . .	25
<b>11 Konzeptvalidierung</b>	<b>27</b>
11.1 Validierung der Sensorik . . . . .	27
11.2 Validierung der Firmware . . . . .	27
11.3 Erreichte Ziele . . . . .	28

<b>12 Schluss</b>	<b>29</b>
<b>Abbildungsverzeichnis</b>	<b>30</b>
<b>Tabellenverzeichnis</b>	<b>31</b>
<b>A Lastenheft</b>	<b>32</b>
<b>B Schema</b>	<b>33</b>
<b>C PCB-Design (Top View)</b>	<b>37</b>
<b>D 3D-Modellierung</b>	<b>38</b>

## 1 Einleitung

Pflanzen benötigen eine ihnen entsprechende Umwelt. Durch meteorologische Messdaten kann diese ermittelt und durch Agronome optimal bewirtschaftet werden. Außerdem können bei genügend Messdaten aus einem Erfassungsnetz Wetterprognosen erstellt werden. Die Erhebung solcher Messdaten trägt somit erheblich zum wirtschaftlichen Erfolg in der Agronomie bei und kann bei einem geeigneten grossen Erfassungsnetz Bewohner vor Unwetter warnen. In den ärmeren Teilen Afrikas und Südamerikas sind solche Systeme jedoch nicht verbreitet.

Aus diesem Grund soll eine kostengünstige, erweiterbare und mobile Wetterstation gebaut werden, welche die örtlichen Agronomen unterstützt. Diese Wetterstation soll die Niederschlagsmenge, die Windstärke, die Lufttemperatur und die Sonnenstunden messen können. Außerdem soll die Wetterstation mittels Photovoltaik unterstützt werden, und erhobene Daten via SMS abrufbar sein. Mit einem GPS-Modul soll der Standort der mobilen Wetterstation erfasst werden. Im Projekt 5 wird ein erster Prototyp erstellt, welcher die Datenermittlung mit der Sensorik und Datenspeicherung beinhaltet. Die Stromversorgung (Akku, Solarpanels), das GPS und das Senden der Daten über SMS werden im Projekt 6 implementiert.

Die Lufttemperatur wird über ein BME280 von Bosch gemessen, welcher ebenso Luftfeuchtigkeit und Luftdruck misst. Mittels Kipplöffelprinzip wird die Niederschlagsmenge über ein Ombrometer von Misol ermittelt, wobei die Funktionsweise des Kipplöffels zuerst mit einem Selbstbau überprüft wird. Die Windstärke wird über ein Anemometer von Froggit eruiert. Um die Sonnenstunden zu ermitteln sollen zwei Varianten (zum einen via Ladestrom der Photovoltaikanlage, zum anderen mit einem Sensor) verglichen und die Leistungsärmere implementiert werden, was jedoch erst im Projekt 6 stattfindet. Es wird für das Prototyping ein ArduinoMega2560 verwendet, welcher die Sensoren über die vorhandenen Peripherieanschlüsse mit dem Mikrocontroller verbindet. Die gesammelten Daten werden auf einer  $\mu$ SD Karte abgespeichert, wobei mit Hilfe eines DS3231 (Präzisions-RTC) ein Zeitstempel hinzugefügt wird.

Das Projekt 5 liefert die erwünschten Messdaten von Lufttemperatur, Windgeschwindigkeit und Niederschlagsmenge. Zusätzlich werden Luftdruck, Luftfeuchtigkeit und Windrichtung gemessen. Außerdem werden die Daten wunschgemäß auf einer  $\mu$ SD Karte, mit zugehörigem Zeitstempel versehen, gespeichert.

Der nachfolgende Bericht beinhaltet die Auftragsbeschreibung (Kapitel 2) und die festgelegten Ziele (Kapitel 3), sowie die Kapitel Konzept (Kapitel 4), Interfaces (Kapitel 5), MCU (Kapitel 6), RTC (Kapitel 7), Sensoren (Kapitel 8), Firmware (Kapitel 10) und Datenspeicherung (Kapitel 9), in welchen die einzelnen Elemente der mobilen Wetterstation thematisiert werden. Zuletzt folgt das Kapitel Konzeptvalidierung (Kapitel 11), worin die Funktionalität des Konzepts der mobilen Wetterstation festgestellt wird.

## 2 Auftragsbeschreibung

Das Wetter spielt eine wichtige Rolle in der Agronomie. Regnet es nicht genug, müssen Pflanzen bewässert werden. Trifft auf ein Ort nur wenig Sonnenlicht, so sollten dort nicht die Pflanzen, welche viel Sonnenlicht brauchen, angebaut werden. Windet es zu stark, können Pflanzen beschädigt oder gar zerstört werden. Ist es Tagsüber heiß, so benötigen die Pflanzen mehr Wasser. Hiesige Bauern besitzen den Luxus von guten Wettervorhersagen dank dem Bundesamt für Meteorologie und Klimatologie (MeteoSchweiz). Dieser Luxus ist in anderen Ländern noch nicht gegeben. Prof. Dr. Nouri Taoufik ist aufgefallen, dass in subtropischen Gegenden wie Teile Südamerikas oder Afrikas dieser Luxus ebenso fehlt.

Aus diesem Grund soll eine kostengünstige, erweiterbare und mobile Wetterstation entwickelt werden, welche diese Bauern unterstützt. Diese Wetterstation soll die Regenmenge, die Windstärke, die Lufttemperatur und die Sonnenstunden messen können. Außerdem soll die Wetterstation mittels Photovoltaik unterstützt werden, und erhobene Daten via SMS abrufbar sein.

### 3 Ziele

Die Ziele sind strikt aufgeteilt in die zwei Projekte 5 und 6. Darin enthalten sind die jeweiligen zu erreichenden Muss- und Wunschziele mit ihren quantifizierten Spezifikationen. Diese sind wichtig, da Ortsabhängig unterschiedliche Normwerte gelten und sich dieses Projekt grundsätzlich auf die Schweiz fokussiert.

**Tabelle 3.1:** Ziele P5

	Ziel	Messbereiche	Genauigkeiten	Einheiten
<b>Mussziele P5</b>				
Sensoren	Lufttemperaturmessung	[ -20; 60 ]	± 1	°C
	Windgeschwindigkeitsmessung	[ 10; 25 ]	± 1	m/s
	Niederschlagsmenge	Wasser	± 100	ml/m <sup>2</sup>
Datenspeicherung	Datenabfrage via PuTTY	≥ 9600		Bd/s
RTC	Implementation	Echtzeit	± 1	s/Jahr
<b>Wunschziele P5</b>				
Sensoren	Sonnenstunden Prototyp	Echtzeit		s

Tabelle 3.1 zeigt diverse Ziele im P5, unterteilt in Muss- und Wunschziele. Zu den Musszielen gehören die Lufttemperaturmessung, die Windgeschwindigkeitsmessung, die Niederschlagsmessung, die Implementation des RTC und die mögliche Datenabfrage via Putty vom Datenspeicher. Die Lufttemperatur soll zwischen -20 bis 60 °C ermittelbar sein, mit einer Genauigkeit von ±1 °C. Die Windgeschwindigkeitsmessung soll vor allem stärkere Windgeschwindigkeiten erfassen, um vor Sturm warnen zu können, weshalb niedrigere Windgeschwindigkeiten vernachlässigt werden können. Die Windgeschwindigkeit soll zwischen 10 und 25 m/s auf ±1 m/s genau gemessen werden. Die Niederschlagsmenge soll nur für Regenwasser bestimmt werden mit einer Genauigkeit von ±100 ml/m<sup>2</sup>. Als Wunschziel soll eine Möglichkeit getestet werden um Sonnenstunden zu detektieren, welche dann im P6 umgesetzt wird.

## 4 Konzept

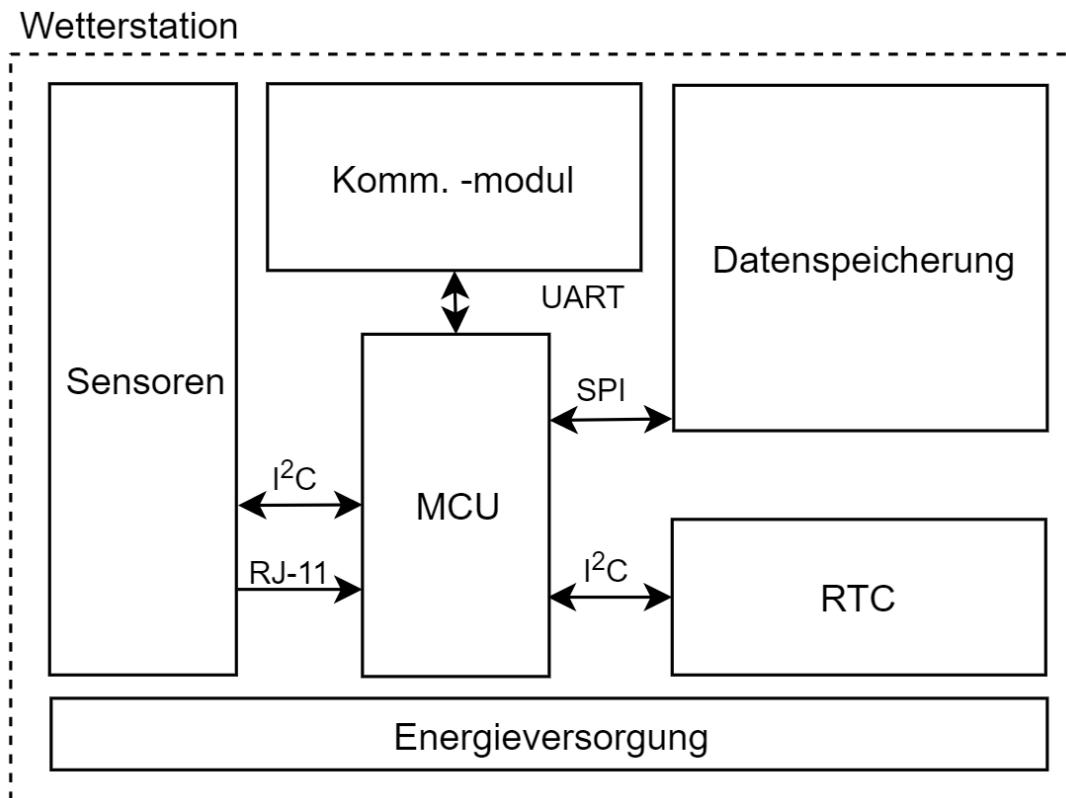
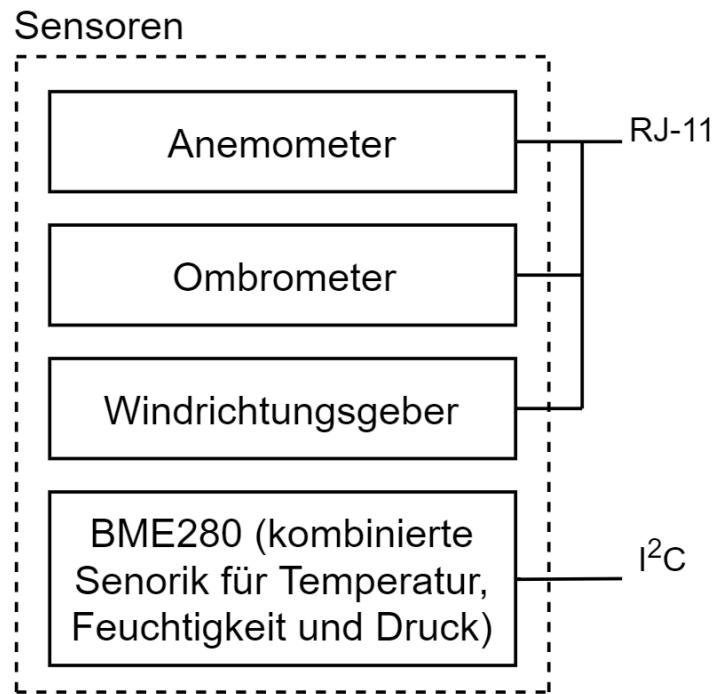


Abbildung 4.1: Grundkonzept

Das gesamte Grundkonzept ist, wie in der Abbildung 4.1 grafisch dargestellt, modular aufgebaut. Auf alle einzelnen Module wird folgend spezifischer eingegangen und genauer erklärt wie die partiellen Module funktionieren, sowie auch das gesamte System aufgebaut ist.

Als Zentralrecheneinheit wird eine *Micro-Controller-Unit (MCU)* verwendet. Dieser ist dafür verantwortlich, dass die Daten richtig verarbeitet und an das dementsprechende Modul weitergeleitet werden. Die Messdaten werden in digitaler Form vom Modul *Sensoren* über das I<sup>2</sup>C-Interface an die *MCU* übertragen. Dieser fügt mit dem *Real-Time-Clock (RTC)* einen Timestamp hinzu, wobei anschließend die Daten in der *Datenspeicherung* nichtflüchtig gespeichert werden. Über das *Kommunikationsmodul* können dann die Daten von Nutznießern abgefragt werden.

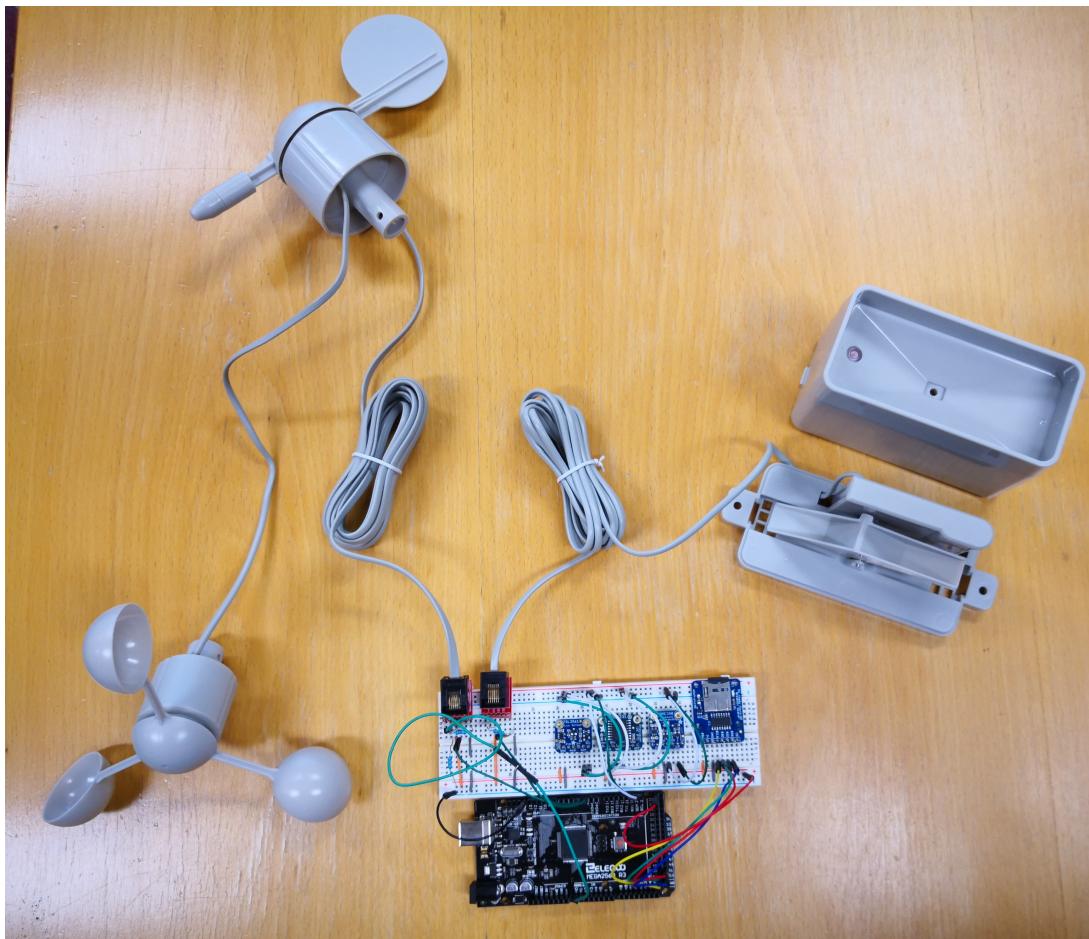
Das Kommunikationsmodul beinhaltet momentan nur das USB-Interface für eine Kommunikation nach außen, wie auch zu Debug-Zwecken des Programms. Auch die Energieversorgung wurde noch nicht implementiert und wurde folgend auch noch nicht genauer erläutert. Diese zwei Teile des Projektes werden in einem weiterführendem Projekt realisiert.



**Abbildung 4.2:** Modul Sensoren mit den verwendeten Sensoren und Messfühlern

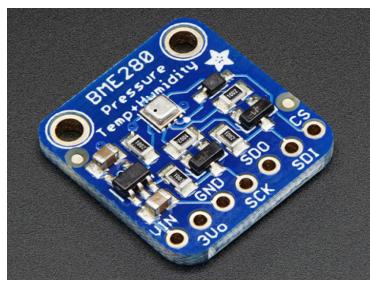
Um eine bessere Übersicht über die verwendeten Sensoren zu geben, wie sie angeschlossen sind und wo sie sich in der gesamten Abstraktion des Konzeptes befinden dient Abb. 4.2. Genauere Spezifikationen der Sensorik werden in den folgenden Kapiteln erläutert.

## 4.1 Prototyping



**Abbildung 4.3:** Prototyping mittels Breakoutboards

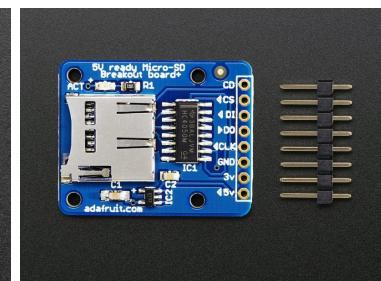
Der erste funktionierende Prototyp wurde auf einem Breadboard realisiert. Für die Implementierung aller partiellen Systemen sind hauptsächlich Breakoutboards von Adafruit mit den entsprechenden Chips verwendet worden.



**Abbildung 4.4:**  
BME280 Breakoutboard  
**LadyAdabme2802018**



**Abbildung 4.5:**  
DS3231 Breakoutboard  
**LadyAda2018ds3231**



**Abbildung 4.6:**  $\mu$ SD-Card Breakoutboard  
**ladyada2018**

Die Abbildungen 4.4, 4.5 und 4.6 zeigen die jeweiligen im Prototyp benötigten Breakoutboards. Wie alles miteinander verbunden wurde, wird auf das Schema im Anhang B verwiesen. Zudem wurde mit diesen Bauteilen ein erster PCB-Entwurf designed (Anhang C), sowie auch davon ein 3D-Modell (Anhang D). Da alle Bauteile mit 3.3V betrieben werden können, ist das gesamte System für eine Betriebsspannung von 3.3V designed worden.

Über die USB-Schnittstelle werden die von der Sensorik erfassten Messdaten ausgegeben. Mittels eines Emulators (z.B. Putty) kann der COM-Port ausgelesen und die Daten dargestellt werden.

Zuerst werden dafür die partiellen Systeme initialisiert und die erfolgreiche Initialisierung ausgegeben. Danach beginnt der Messvorgang, wobei den Daten noch der Zeitstempel zugewiesen wird. Dann erfolgt die Ausgabe der Sensordaten (siehe Abb. 4.7).

```
SD card successfully initialized!
RTC successfully initialized!
BME280 successfully initialized!
TSL2561 successfully initialized!
Friday: 16:52:37      18.1.2019
Temperature:   23.71  Celsius
Humidity:     24.92  %
Airpressure:  975.67  hPa
Windspeed:    0.00   m/s
Windstrength: 0
Winddirection: E
Rainfall:      0.00  ml/m^2
```

Abbildung 4.7: Datenausgabe

## 5 Interfaces

### 5.1 SPI (Serial Peripheral Interface)

Das *Serial Peripheral Interface* ist ein synchrones serielles Datenprotokoll (Datenbus) bestehend aus drei Datenleitungen zur Datenübertragung. Diese sind, wie in Abbildung 5.1 zu sehen, **MISO** (Master In Slave Out), **MOSI** (Master Out Slave In) und **SCLK** (Serial Clock). Auf dem Breakoutboard (Abb. 9.1) sind die Pins mit **DI** (Data In), **DO** (Data Out) und **CLK** (Clock) beschrieben. Zu den Datenleitungen wird noch eine **SS**- (Slave Select) oder auch **CS**-Leitung (Chip Select) benötigt. Damit wird vom Master aus den zur momentanen Kommunikation nötigen Slave selektiert. Große Vorteile von SPI sind die Vollduplexfähigkeit und das Taktfrequenzen bis in den MHz-Bereich reichen. [spiWikipedia2018spi](#)

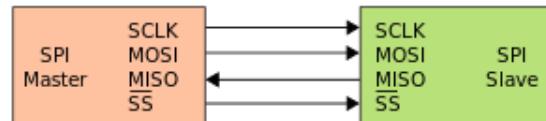


Abbildung 5.1: Einfacher SPI-Datenbus  
Wikipedia2018spi

Möglicherweise noch die Taktfrequenz angeben, resp. die Einstellungen des SPI-Protokolls

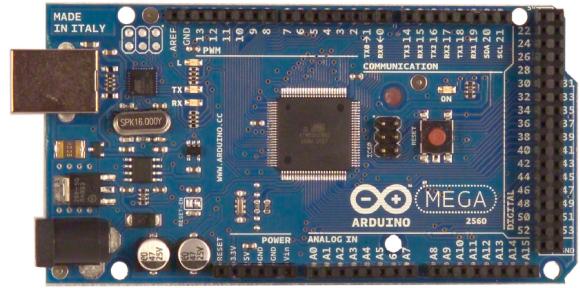
### 5.2 I<sup>2</sup>C

Das **I<sup>2</sup>C** (*inter integrated circuit*) ist ein synchrones serielles Datenprotokoll (Datenbus) der aus zwei bidirektionalen Leitungen besteht. Dabei wird eine der Leitungen für den Transfer des Taktsignals (**SCL**, *serial clock line*) und die andere für den Datentransfer (**SDA**, *serial data line*) verwendet. Dieser Bus wurde speziell dafür entwickelt, dass **ICs** (*integrated circuits*) über kleine Distanzen gut miteinander kommunizieren können. Es können bis zu 128 **ICs** (7-Bit-Adressierung) über einen **I<sup>2</sup>C**-Bus miteinander verbunden werden, wobei mindestens einer davon Master sein muss. Nur Master können aktiv Slaves ansprechen, da die Slaves nicht in der Lage sind selbstständig mit der Kommunikation zu beginnen. Um die Kommunikation mit einem Slave einzuleiten gibt der Master die eindeutige Adresse des **ICs** auf die Datenleitung und teilt zugleich mit, ob er Daten senden oder empfangen möchte. Die Daten werden im Anschluss direkt gesendet und bei Beendigung des Sendevorgangs wird der Bus wieder freigegeben. Bei mehreren Mastern gewährleistet ein Protokoll die sichere Kommunikation durch das Verhindern von Kollisionen. Die einfache Ansteuerung ist ein grosser Vorteil des **I<sup>2</sup>C**-Bus. Außerdem können sowohl langsame als auch sehr schnelle **ICs** eingesetzt werden. Neuere Formen des **I<sup>2</sup>C**-Bus ermöglichen eine 10-Bit-Adressierung, welche ebenso mit 7-Bit-Adressierungsschema kombiniert werden können, was die Skalierbarkeit des Bus nochmals erhöht. Außerdem ist es mit neuen Bauelementen möglich, mehrere identische **ICs** mit derselben Adresse anzuschliessen, was die Skalierbarkeit des Bus nochmals steigert. [mcI2C rnI2C](#)

## 6 MCU

Die Micro Controller Unit (MCU) ist der zentrale Bestandteil für die Kommunikation, resp. für den Datenaustausch zwischen den unterschiedlichen Modulen. Sie interpretiert die Signale der Sensoren und rechnet sie in die interessierenden Messwerte um. Dann weist die MCU jedem Messwert einen Zeitstempel über das RTC zu und übergibt diesen der Datenspeicherung. Wenn die Daten vom Kommunikationsmodul angefordert werden, liest die MCU die Datenspeicherung aus und übergibt sie dem Kommunikationsmodul.

Für die Entwicklung der MCU wird ein Arduino Mega Board verwendet. Der Vorteil besteht darin, dass elementare Bauteile (Hardware) bereits implementiert sind, wie z.B. Oszillator, der USB-Anschluss und die PCB-Connectors für ein schnelles Prototyping. Die wichtigsten technischen Daten sind in der Tabelle 6.1 aufgelistet.



**Abbildung 6.1:** Arduino Mega Board  
**arduinoMega**

**Tabelle 6.1:** Technische Daten arduinoMega

Microcontroller	ATmega2560
Operating Voltage	5V
Digital I/O Pins	54
Analog Input Pins	16
Flash Memory	256 KB, 8 KB werden vom bootloader benötigt
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

## 7 RTC

Um die gewonnenen Messdaten mit einem Zeitstempel zu versehen, wird eine **RTC** (*real time clock*) benötigt. Um eine möglichst kleine Abweichung zu haben, wird eine hohe Präzision vorausgesetzt. Ausserdem soll die **RTC** über das in Kapitel 5.2 erwähnte **I<sup>2</sup>C**-Bus angesteuert werden. Aus diesen Gründen wurde der **DS3231** implementiert, da dieser als Präzisions-**I<sup>2</sup>C-RTC** die Anforderungen erfüllt. Die hohe Präzision des **DS3231** wird mit einem internen Temperatursensor erreicht, welcher Temperaturbedingte Abweichungen des Oszillators kompensiert. Zu sehen ist der **DS3231** mit seinen Anschlüssen in Abbildung 7.1.

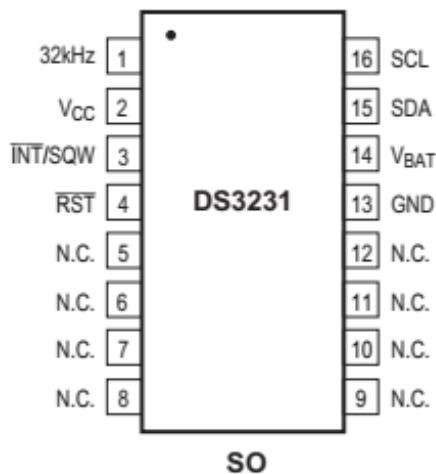


Abbildung 7.1: DS3231 mit seinen Anschlüssen DS3231DS.

Die Anschlüsse des **DS3231** sind **VCC**, **GND**, **SCL**, **SDA**, **BAT**, **32K**, **SQW** und **RST**. **VCC** und **GND** werden für die Speisung benötigt. **SCL** und **SDA** sind Anschlüsse für den **I<sup>2</sup>C**-Bus. **BAT** ist der positive Anschluss der Knopfbatterie, worüber deren Zustand kontrolliert werden, etwas anderes gespiesen oder eine andere Battery als Backup angeschlossen werden kann. **32K** ist ein Anschlusspin um den Output des 32kHz-Oszillators des **RTC** abzugreifen, was jedoch nicht verwendet wird. **SQW** ist ein zusätzlicher Output- oder Interrupt-Pin, welcher jedoch auch nicht verwendet wird. **RST** wird verwendet, um ein externes Element zu reseten oder als Indikator wenn die Hauptspeisung unterbrochen wird. **DS3231DS** Die relevanten Spezifikationen des Chips sind in Tabelle 7.1 aufgelistet.

Parameter	Min.	Typ.	Max.	Einheit	Condition
VCC	2.3	3.3	5.5	V	
VBAT	2.3	3	5.5	V	
Active Supply Current			200	$\mu$ A	3.63 V
			300	$\mu$ A	5.5 V
Standby Supply Current			110	$\mu$ A	3.63 V
			170	$\mu$ A	5.5 V
Crystal Aging	$\pm$ 1			ppm	First Year
	$\pm$ 5			ppm	0-10 Years
Active Battery Current			70	$\mu$ A	3.63 V
			150	$\mu$ A	5.5 V
Timekeeping Battery Current	0.84	3		$\mu$ A	3.63 V
	1	3.5		$\mu$ A	5.5 V

**Tabelle 7.1:** Spezifikationen des **DS3231 DS3231DS**.

Tabelle 7.1 zeigt die für das Projekt relevanten Spezifikationen des **DS3231**. Wichtig ist, dass die Alterung des Oszillators zu einem Fehler führt, dieser jedoch im ppm-Bereich liegt und somit erst über viele Jahre hinweg bemerkbar wird.

## 7.1 Implementation in die Firmware

Für die Implementation wurde die bereits existierende Library **RTClib** von Adafruit in die Firmware integriert. Anschließend konnte das Headerfile `<Adafruit_RTClib.h>` inkludiert werden und mit der Funktion `TimeStamp getTimeStamp()` der aktuelle Zeitstempel abgerufen werden.

Beim flashen des Programms wird die RTC mit der Uhr des angeschlossenen Computers synchronisiert, weshalb es durch die Übertragungsverzögerung zu einer Abweichung kommt, welche der Dauer des flashens entspricht.

## 8 Sensoren

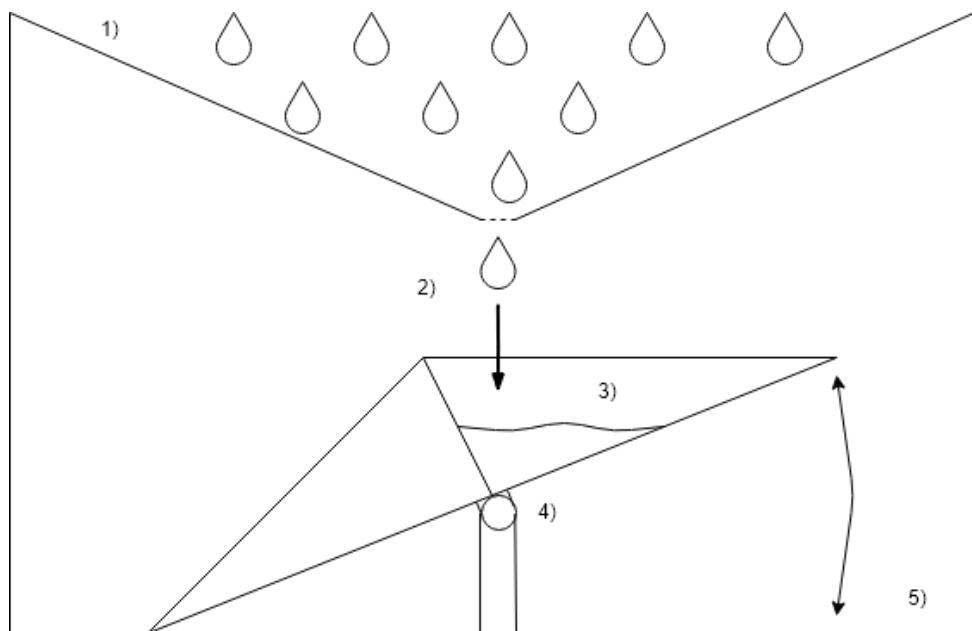
### 8.1 Ombrometer

#### Ermittlung der Niederschlagsmenge

Dieses Unterkapitel befasst sich mit der Realisierung der Niederschlagsmessung. Diese soll nach einem Kipplöffelprinzip funktionieren und gemäss definierten Zielen eine Genauigkeit von  $\pm 100 \text{ ml/m}^2$  aufweisen. Ausserdem soll als alternative zusätzlich ein Messbecher an der Wetterstation installiert werden, damit der Bauer die Niederschlagsmenge anhand einer Skala ablesen kann. In einem ersten Schritt soll das Kipplöffelprinzip näher erläutert und mit einem Selbstbau die Funktionsweise getestet werden. Anschliessend soll ein gekaufter Sensor die Wetterstation erweitern und die Implementation in der Firmware thematisiert werden. Zu guter Letzt soll die Validierung des Teilsystems folgen.

#### Das Kipplöffelprinzip

Das Prinzip des Kipplöffels wird in Abbildung 8.1 graphisch dargestellt.



**Abbildung 8.1:** Darstellung des Kipplöffelprinzips

Abbildung 8.1 zeigt das Kipplöffelprinzip. Der Kipplöffel („3“) besteht im Grunde aus zwei Löffeln und ist in der Mitte drehbar mit dem Gehäuse befestigt („4“). Regenwasser wird über eine Öffnung im Gehäusedeckel (Trichter, „1“) zum Kipplöffel befördert („2“). Ist der Löffel mit Regenwasser gefüllt, so kippt dieser aufgrund des Gewichts und leert das Wasser über eine Öffnung im Gehäuseboden („5“) aus. Durch die Kippung wird der andere Löffel in die Ausgangsposition bewegt und kann sich nun mit Wasser füllen. Mit der Hilfe von Reedkontakte und Magneten wird die Anzahl der Kippbewegungen gezählt. Die Niederschlagsmenge ergibt sich aus der Anzahl Kippbewegungen, multipliziert mit dem Volumen des Kipplöffels.

## Die Realisierung des Niederschlagsmengensensors

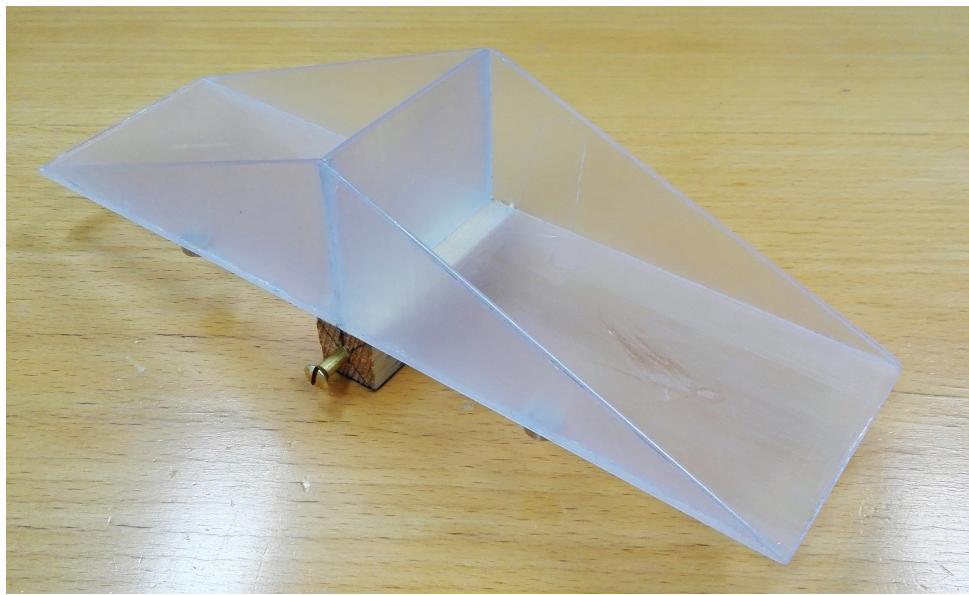
Um die Funktionsweise des Niederschlagsmengensensors zu testen, wird, wie im Pflichtenheft festgehalten, dieser in einem ersten Schritt selbst erstellt. Die Erstellung kann in vier Etappen unterteilt werden. Die erste Etappe ist die Erstellung des Kipplöffels. Die zweite Etappe folgt mit der Erstellung der drehbaren Lagerung. Als dritte Etappe folgt der Trichter und die vierte und letzte Etappe widmet sich dem Gehäuse, wobei der Trichter ein Teil des Gehäuses darstellt. Das Gehäuse wird, bei Verwendung der Eigenproduktion, erst im Projekt 6 mit dem Gehäuse der gesamten Wetterstation erstellt.

### Etappe 1: Realisierung des Kipplöffels

Wichtig für die Erstellung des Kipplöffels sind die Dimensionierung und die Materialwahl.

Das Material soll wetterbeständig, einfach bearbeitbar und günstig sein und eine möglichst glatte Oberfläche haben. Die möglichst glatte Oberfläche ist notwendig, damit das Wasser im Kipplöffel sich nicht an der Oberfläche festhält und somit gut abfließt. Acrylglas erfüllt diese Bedingungen und ist in jedem Baumarkt erhältlich, weshalb es als Material gewählt wird.

Die Dimensionierung ist Abhängig von der gewählten Genauigkeit im Pflichtenheft. Damit eine Genauigkeit von  $\pm 100 \text{ ml}/\text{m}^2$  erreicht werden kann, müssen beide Löffel des Kipplöffels bei genau 100 ml Fassungsvermögen kippen. Damit dies erreicht wird, kann man physikalisch die statische Gleichgewichtsbedingung aufstellen und daraus die Dimensionierung folgern. Dies ist jedoch ein sehr aufwändiger, komplizierter und zeitintensiver Weg. Einfacher ist es, wenn der Kipplöffel extra zu gross dimensioniert und die Füllmenge im Nachhinein justiert wird. Die Justierung erfolgt mittels einer in der Höhe verstellbaren Lagerung, sowie mit in der Höhe verstellbaren Schrauben im Gehäuseboden, welche die Neigung der Endposition des Kipplöffels beeinflusst. Ein weiterer Vorteil dieser Nachjustierung ist, dass auch eine andere Füllmenge einstellbar wäre.



**Abbildung 8.2:** Selbsterstellter Kipplöffel.

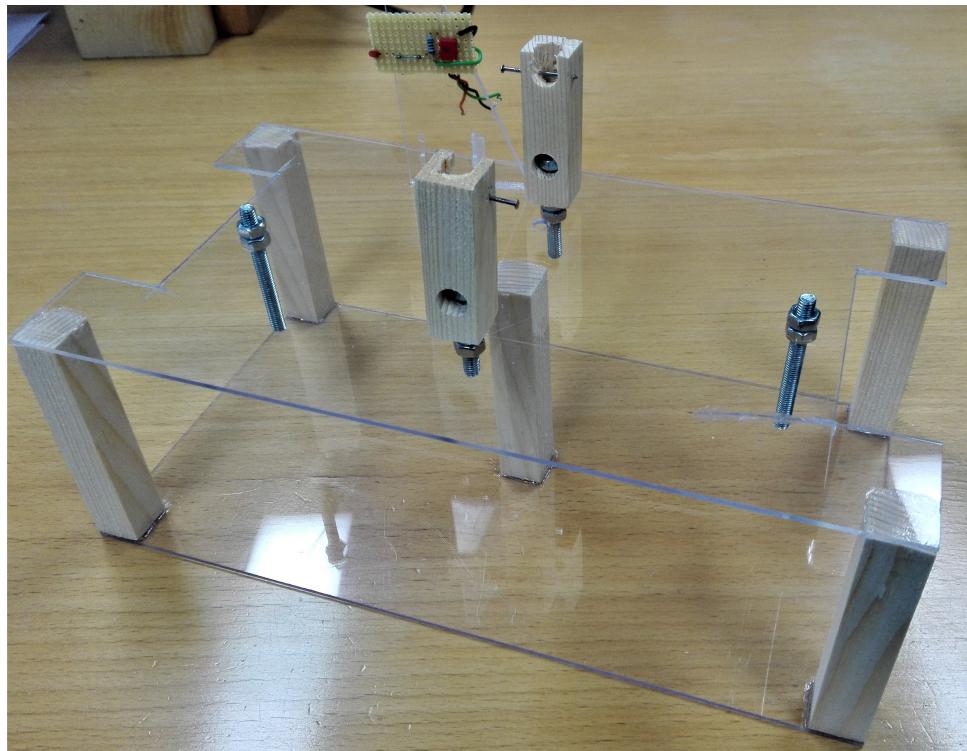
Abbildung 8.2 zeigt den selbsterstellten Kipplöffel aus Acrylglas. Die Drehachse ist mittig unter dem Kipplöffel befestigt und besteht aus einem Holzklotz mit je einer Schraube pro Seite.

### Etappe 2: Realisierung der drehbaren Lagerung

Die drehbare Lagerung des Kipplöffels ist wichtig, damit der Kipplöffel auf beide Seiten kippen kann. Die Drehachse soll direkt unterhalb der Mitte des Kipplöffels befestigt sein um ein gleich-

mässiges Kippen zu ermöglichen. Die Höhe des Kipplöffels wird definiert durch die einstellbare Höhe der Drehachsenlagerung.

Die Drehachse wird aus einem Stück Holz und zwei Schrauben gefertigt, wobei das Holz direkt am Kipplöffel befestigt wird. Die zwei Schrauben werden auf einem höhenverstellbarem Gerüst gelagert, so dass ein drehen möglich ist. Dieses Gerüst wird auch aus Holz gefertigt und enthält eine Metallische Fläche an der Kontaktstelle der zuvor erwähnten Schrauben, um aufkommende Reibkräfte zu verringern. Ausserdem ist dieses Gerüst höhenverstellbar über zwei mit Muttern feststellbaren Gewinden (für jede Seite eine).



**Abbildung 8.3:** Selbsterstelltes Gerüst.

Abbildung 8.3 zeigt das selbsterstellte Gerüst für die Höhenverstellbare, drehbare Lagerung des Kipplöffels. Es kann sowohl die Höhe des Kipplöffels, sowie dessen Neigung in den Endpositionen über Gewinden mit Muttern eingestellt werden. Die Schrauben des Kipplöffels kommen auf einen Stahlnagel zu liegen, womit Reibungsverluste gering gehalten werden.

### **Etappe 3: Realisierung des Trichters**

Der Trichter sorgt dafür, dass der Regen, welcher auf die Trichterfläche fällt, über der Mitte des Kipplöffels in den Löffel fliesst. Die Trichterfläche stellt gleichzeitig die Referenzfläche dar, da die gesamte Regenmenge dieser Fläche über den Kipplöffel erfasst wird. Ist diese Fläche von  $1 m^2$  abweichend, so muss in der Firmware ein Skalierungsfaktor implementiert werden, damit die Regenmenge wie gewünscht gemäss Pflichtenheft ermittelt werden kann. Der Trichter wird aus demselben Material gefertigt wie der Kipplöffel, da hier die gleichen Anforderungen gelten. Es sei angemerkt, dass der Trichter nur bei weiteren Verwendung des selbst erstellten Kipplöffels, zusammen mit dem Gehäuse der gesamten Wetterstation, erstellt wird.

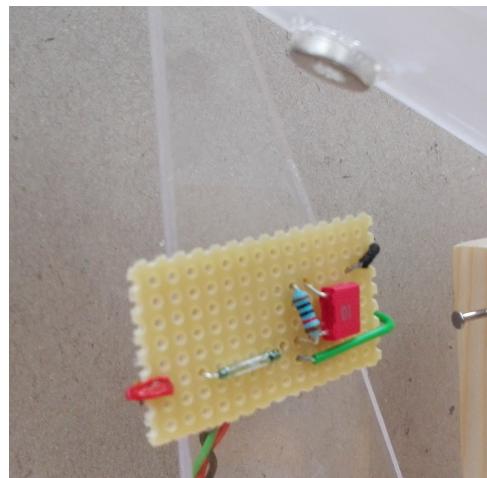
### **Etappe 4: Realisierung des Gehäuses**

Das Gehäuse soll den Sensor vor ungewollten äusseren Einflüssen schützen, sowie umgebende Elektronik vor eventuellen Regenwasserspritzern. Ausserdem soll ein Schaltkreis mit Reedrelais implementiert werden, damit die Kippbewegungen von der Elektronik erfasst werden können.

Es sei erwähnt, dass das Gehäuse nur bei weiteren Verwendung des selbst erstellten Sensors, zusammen mit dem Gehäuse der Wetterstation, konstruiert wird.

### Implementierung des Schaltkreises

Der Schaltkreis, welcher die Kippbewegungen feststellen soll, besteht im wesentlichen aus einem Reedrelais und einem Permanentmagneten. Das Reedrelais ist NO (Normally Open) und wirkt als stromkreisschliessender Schalter, sobald ein magnetisches Feld (z.B. das eines Permanentmagneten) sich in unmittelbarer Nähe befindet. Der Permanentmagnet wird auf dem Kipplöffel befestigt und das Reedrelais als Gegenstück an einem Fixpunkt in der Nähe. Wichtig dabei ist, dass das Reedrelais bei den Endpositionen des Kipplöffels nicht geschlossen ist, damit der Stromkreis geöffnet ist und Strom gespart werden kann. Das Reedrelais benötigt einen seriellen Widerstand, damit bei einem schliessen des Stromkreises kein Kurzschluss auftritt. Außerdem soll ein Kondensator parallel zum Widerstand sein, um die Speisespannung zu glätten und so ein nutzbares Signal zu erhalten. Die Speisespannung stellt den Pegel für ein schliessen des Reedrelais, und somit auch für eine Kippbewegung dar. Um die Kippbewegungen zu zählen, kann somit entweder jede steigende oder jede fallende Flanke des Signals gezählt werden.



**Abbildung 8.4:** Schaltkreis zur detektion der Kippbewegung.

Abbildung 8.4 zeigt den Schaltkreis zur detektion der Kippbewegung. Benutzt wurde ein  $10\text{k}\Omega$  Widerstand mit einem parallel angeschlossenen  $100\text{nF}$  Kondensator. Der Reedkontakt reagiert auf den ebenso sichtbaren Permanentmagneten, welcher an der Unterseite des Kipplöffels befestigt ist.

### Nachteile des Selbstgebauten Niederschlagsmengensensor

Der selbstgebaute Niederschlagssensor beweist, dass das Prinzip des Kipplöffels funktioniert. Dennoch weist der Selbstbau Mängel auf. Der verwendete Permanentmagnet muss geklebt werden, weshalb dessen Magnetfeld massiv an Stärke verliert und die Schaltung deshalb äusserst nahe angebracht werden muss. Außerdem kam es, dadurch dass keine Werkstatt zugänglich war, zu Improvisation bei nahezu allen Fertigungsschritten, was zu unkalkulierbaren Abweichungen führt. Als Beispiel sei das Spiel der drehbaren Lagerung des Kipplöffels auf dem Gerüst angeführt, was jegliche Justierungsversuche der Niederschlagsmenge beeinflusst. Aus den genannten Gründen wird vorgefertigter Sensor mit Kipplöffelprinzip verwendet.

### Implentation in der Firmware

### Validierung der Niederschlagsmessung

## 8.2 Anemometer

Für die Windgeschwindigkeitsmessung wurde ein Ersatz Anemometer von Froggit genommen (Abb. 8.5). Das Anschlusskabel hat einen vier poligen RJ-11 Stecker, dessen Signal über eine Buchse zum MCU geführt wird. Das Anemometer selbst hat allerdings nur zwei Anschlüsse, die Speisung (rot) und das durch einen mit einem Dauermagneten schließbaren Reedkontakt modulierte pulsförmige Ausgangssignal (grün, Abb. 8.6). In der Abb. 8.7 ist ersichtlich, dass das Ausgangssignal über R1 abfällt und C1 als Spannungsstabilisierung dient. Das daraus resultierende Signal ist in der Abb. 8.8 aufgezeigt. Die Windgeschwindigkeit ist nun aus der Anzahl Rechteckpulsen direkt interpretierbar:

Wenn über einen Zeitraum  $T$  die Anzahl Pulse  $A$  gemessen werden, dann kann auf die Winkelgeschwindigkeit  $\omega$  nach

$$\omega = \frac{A}{T} \quad [s^{-1}] \quad (8.1)$$

geschlossen werden. Da allerdings verschiedene Faktoren wie das Trägheitsmoment des Schalenkreuzes, Reibungsverluste bei der Drehbewegung, Verfälschung bei wechselnder Windrichtung usw. zusätzlich auf das Anemometer wirken, wird es sehr komplex die Windgeschwindigkeit exakt zu berechnen. Deshalb wird nur ein Näherungswert ermittelt und mit einem Skalierungsfaktor  $SF$  korrigiert. Somit ergibt sich für die Windgeschwindigkeit  $v_{Wind}$  mit Radius  $r$  des Schalenkreuzes

$$v_{Wind} = \frac{A * r * SF}{T} \quad [m/s]. \quad (8.2)$$

Der Skalierungsfaktor  $SF$  wird mittels Referenzmessungen der Windgeschwindigkeit eines digitalen Anemometers eruiert.

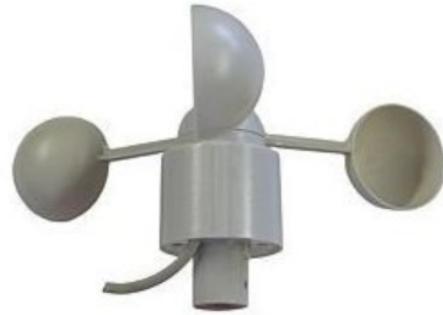


Abbildung 8.5: Anemometer AmazonAnemometer

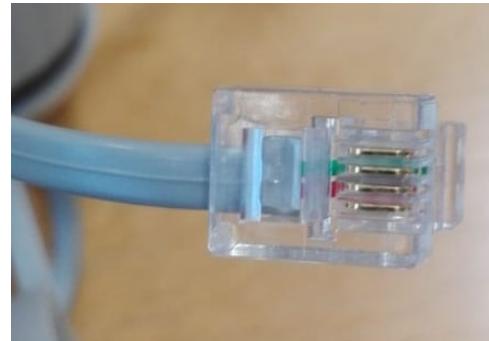


Abbildung 8.6: RJ-11 Stecker

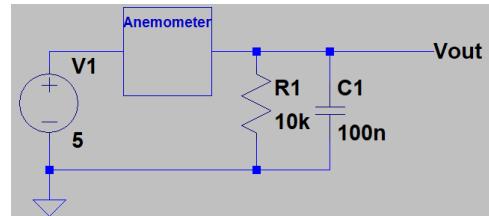


Abbildung 8.7: Beschaltung des Ausgangs des Anemometers.

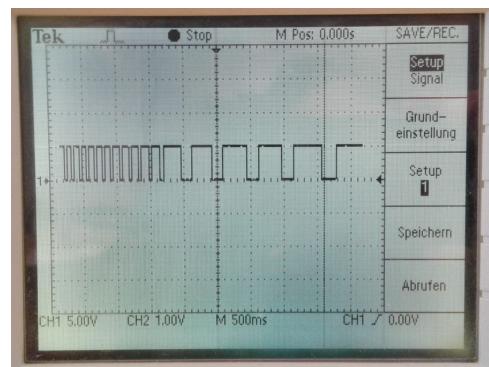


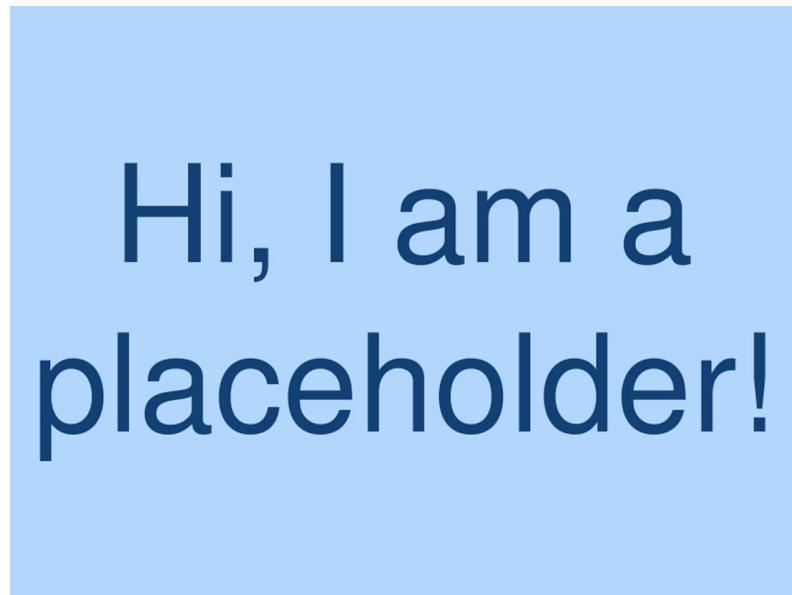
Abbildung 8.8: Ausgangssignal  $V_{out}$

### Implementation in die Firmware

Die Implementation wurde recht simpel gehalten. Der gesamte implementierte Code für das Anemometer ist im Headerfile "Anemometer.h" extern deklariert und im File Anemometer.cpp initialisiert. Das Signal  $V_{out}$  ist mit einen digital Pin des Atmega 2560 (Pinnummer 2 des Arduino Mega Boards) verbunden. Über einen Zeitraum von  $5000ms$ , auf die steigende Flanke getriggert, wird die Anzahl von Pulsen mittels Interrupt<sup>1</sup> gezählt. Dabei wird zuerst der Interrupt auf der Pinnummer 2 aktiviert, mit einem Delay von  $5000ms$  gewartet, wobei bei jedem ausgelösten Interrupt die Funktion `countWind()` ausgeführt und somit bei jeder steigenden Flanke um eins inkrementiert wird. Zum Schluss folgt die Deaktivierung des Interrupts und die Berechnung der Windgeschwindigkeit nach der Gleichung 8.2.

### Validierung

Über eine einigermaßen konstanten Windgeschwindigkeit eines Heizlüfters/Ventilators (mit verschiedenen Stärkestufen) wurden Messpunkte des Anemometers (Abb. 8.5), sowie auch des digitalen Anemometers (Abb. 8.10) erfasst. In der Abb. 8.9 sind diese Messwerte graphisch dargestellt.



**Abbildung 8.10:** Digitales Anemometer (Benetech GM8908) mit einer Auflösung von  $0.1m/s$  und einer Unsicherheit von  $\pm 5\%$

**Abbildung 8.9:** Graph der Messwerte

Hier muss noch die Interpretation der gemessenen Werte geschrieben werden.

<sup>1</sup>es handelt sich hierbei um *external Interrupts*.

### 8.3 Windrichtungsgeber

Um die Windrichtung angeben zu können, wurde ein Windrichtungsgeber, wie in Abb. 8.13 gezeigt von MISOL verwendet. Er ist genau wie das Ombrometer und das Anemometer mit Reedkontakte realisiert worden (siehe Abb. 8.11). Dafür sind acht Reedkontakte im Kreis angeordnet und jeder hat einen in Serie geschalteten Widerstand von unterschiedlichen Dimensionen. Der Dauermagnet kann, je nach Drehwinkel bis zu zwei Reedkontakte gleichzeitig schließen. Dies erlaubt sechzehn verschiedene Winkelpositionen und somit eine Auflösung von  $22.5^\circ$ . Mit einem externen Widerstand  $R = 10k\Omega$  (siehe Abb. 8.12) wird eine Spannung generiert, welche dann mit dem ADC des Microcontrollers gelesen werden kann. Der Windrichtungsgeber wird mit einer Speisespannung von  $V_+ = 5V$  betrieben. Der Windrichtungsgeber hat einen vierpoligen RJ-11 Anschluss. Zudem hat er auf der unteren Seite noch eine RJ-11 Buchse, bei der das Anemometer direkt angeschlossen werden kann. Wie diese Anschlüsse gemapped sind, wird in der Abb. 8.11 gezeigt.

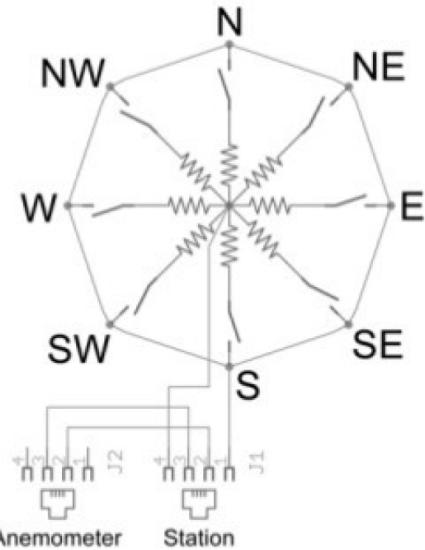
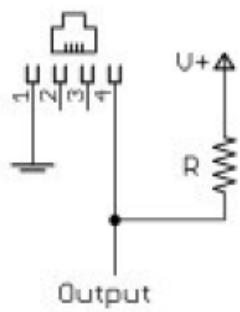


Abbildung 8.11: Interne Schaltung ADSkeineAngabe

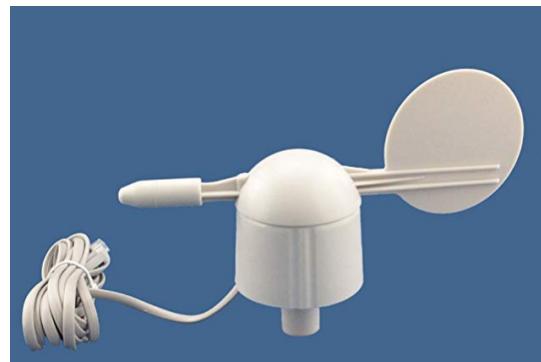
Tabelle 8.1: Technische Werte ADSkeineAngabe

Richtung [°]	Himmelsrichtung	Widerstand [Ω]	Ausgangsspannung [V]
0	N	33k	3.84
22.5		6.57k	1.98
45	NE	8.2k	2.25
67.5		891	0.41
90	E	1k	0.45
112.5		688	0.32
135	SE	2.2k	0.90
157.5		1.41k	0.62
180	S	3.9k	1.40
202.5		3.14k	1.19
225	SW	16k	3.08
247.5		14.12k	2.93
270	W	120k	4.62
292.5		42.12k	4.04
315	NW	64.9k	4.33
337.5		21.88k	3.43

In der Tabelle 8.1 sind die Widerstandswerte der in Abb. 8.11 gezeigten Widerständen und die Werte der Ausgangsspannung bei variabler Winkelposition aufgelistet. Da nun abhängig von der Winkelposition unterschiedliche Widerstände parallel geschalten werden, resultiert am Ausgang eine vom Winkel abhängige Ausgangsspannung.



**Abbildung 8.12:** Spannungsteiler mit  $R = 10k\Omega$  **ADSkeineAngabe**



**Abbildung 8.13:** Windrichtungsgeber von MISOL windrichtungsgeber

### Implementation in die Firmware

Der ADC hat des ATMega2560 hat eine 10 Bit Auflösung und wird mit 5 Volt betrieben. Um das Signal am ADC lesen zu können, wird die Funktion *analogRead()* aufgerufen. Als Rückgabewert wird ein binärer Wert als Dezimalzahl vom Datentyp *float* erhalten. Um diesen Wert in die eigentlich am ADC anliegende Spannung umzurechnen wird die Gleichung 8.3 umgeformt.

$$\frac{\text{AuflösungADC}}{\text{Betriebsspannung}} = \frac{\text{analogRead()}}{\text{Ausgangsspannung}} \quad (8.3)$$

Daraus resultiert:

$$\text{Ausgangsspannung} = \frac{\text{analogRead()} * \text{Betriebsspannung}}{\text{AuflösungADC}} \quad (8.4)$$

Werden nun die Werte in die Gleichung 8.4 eingefügt, ergibt sich für die Ausgangsspannung  $V_{out}$ :

$$V_{out}(\text{analogRead()}) = \frac{\text{analogRead()} * 5V}{10} \quad (8.5)$$

Diese von *analogRead()* abhängige Ausgangsspannung  $V_{out}$  wird dann in *if else* Anweisungen zu der Himmelsrichtung zugewiesen und als String abgespeichert.

## 8.4 BME280

Der *BME280* ist ein low powered digitaler Feuchtigkeits-, Luftdruck- und Temperatursensor von Bosch. Er ist in einem 2.5mm x 2.5mm x 0.93mm metal lid LGA Gehäuse verpackt und kann über die Interfaces I<sup>2</sup>C und SPI kommunizieren. Durch seinen niedrigen Stromverbrauch, große operating range der drei Messgrößen und schnellen Ansprechzeit von etwa 1s eignet er sich für die solarbetriebene mobile Wetterstation besonders.

**Bosch2019**



Abbildung 8.14: BME280 Bosch2019

Tabelle 8.2: Elektrische Spezifikationen **Bosch2019**

Parameter	Min.	Typ.	Max.	Einheit
Versorgungsspannung	1.71	1.8	3.6	V
Stromverbrauch (sleep mode)	0.1	0.3		µA
Stromverbrauch inaktiv (normal mode)	0.2	0.5		µA
Stromverbrauch Feuchtigkeitsmessung	340			µA
Stromverbrauch Luftdruckmessung	714			µA
Stromverbrauch Temperaturmessung	350			µA

Bei einer Messfrequenz von 1Hz für die drei Messgrößen verbraucht der BME280 somit laut Datenblatt nur **3.6µA**. **Bosch2019**

### Feuchtigkeitsmessung

In der Tabelle 8.3 sind die wichtigsten Parameter zur Feuchtigkeitsmessung aufgelistet. Zu vermerken ist, dass die digitalen Werte des BME280 zur Feuchtigkeitsmessung relativ sind und deshalb prozentual angegeben werden.

Tabelle 8.3: Sezifikationen der Feuchtigkeitsmessung **Bosch2019**

Parameter	Min.	Typ.	Max.	Einheit
Operating range	-40	25	85	°C
	0		100	%
Absolute Genauigkeitstoleranz	±3			%
Hysterese	±1			%
Auflösung	0.008			%
Langzeitstabilität	0.5			% pro Jahr

## Luftdruckmessung

Die Genauigkeit der Luftdruckmessung ist an einen Temperaturbereich gebunden. Bei niedrigeren Temperaturen ( $<0^{\circ}\text{C}$ ) weist der Sensor eine höhere Unsicherheit auf als bei Temperaturen von 0 bis  $65^{\circ}\text{C}$  (siehe Tabelle 8.4).

**Tabelle 8.4:** Sezifikationen der Luftdruckmessung **Bosch2019**

Parameter	Temperaturbereich	Min.	Typ.	Max.	Einheit
Operating range		-40	25	85	$^{\circ}\text{C}$
		300		1100	hPa
Absolute Genauigkeit	-20 bis $0^{\circ}\text{C}$		$\pm 1.7$		hPa
	0 bis $65^{\circ}\text{C}$		$\pm 1$		hPa
Auflösung			0.18		hPa
Langzeitstabilität			$\pm 1$		hPa pro Jahr

## Temperaturmessung

Die Wetterstation wird hauptsächlich in einem Temperaturbereich von 0 bis  $65^{\circ}\text{C}$  betrieben, wodurch vom Sensor eine Unsicherheit von max.  $\pm 1^{\circ}\text{C}$  erreicht werden kann.

**Tabelle 8.5:** Sezifikationen der Temperaturmessung **Bosch2019**

Parameter	Temperaturbereich	Min.	Typ.	Max.	Einheit
Operating range		-40	25	85	$^{\circ}\text{C}$
Absolute Genauigkeit	$25^{\circ}\text{C}$		$\pm 0.5$		$^{\circ}\text{C}$
	0 bis $65^{\circ}\text{C}$		$\pm 1$		$^{\circ}\text{C}$
	-20 bis $0^{\circ}\text{C}$		$\pm 1.25$		$^{\circ}\text{C}$
	-40 bis -20 $^{\circ}\text{C}$		$\pm 1.5$		$^{\circ}\text{C}$
Auflösung			0.01		$^{\circ}\text{C}$

## Implementation in die Firmware

Um den BME280 vom Microcontroller aus ansteuern zu können, wurden zwei bereits existierende Librarys von Adafruit verwendet:

- Adafruit BME280 Library
- Adafruit Unified Sensors

Anschließend konnten die Headerfiles `<Adafruit_Sensor.h>` und `<Adafruit_BME280.h>` inkludiert werden und der Sensor über das I<sup>2</sup>C Interface mit den folgenden Funktionen abgefragt werden:

- `float readTemperature()`
- `float readHumidity()`
- `float readPressure()`

## 9 Datenspeicherung

Die Datenspeicherung beinhaltet die gespeicherten Messwerte der Sensoren. Dafür wird als Speichermedium eine  $\mu$ SD-Karte verwendet, welche direkt in das 254  $\mu$ SD-Breakoutboard von Adafruit gesteckt wird. Die Daten werden dann in einem \*.txt-File nicht flüchtig gespeichert. Bei Beschädigung der Hardware können dann die zuletzt erfassten Daten immer noch mittels eines SD-Karten-Adapters von einem Computer ausgelesen werden. Als Kommunikationsprotokoll für das Schreiben und Auslesen der Karte wird SPI verwendet.

### 9.1 Breakoutboard

Das Breakoutboard (siehe Abb. 9.1) kann wegen des intern implementierten *CD74HC4050 high-speed logic level translators*<sup>2</sup> mit 5V betrieben werden. Das Arduino Mega Board und das Breakoutboard werden über SPI (siehe Kapitel 5.1) nach dem Master-Slave Kommunikationsprinzip miteinander verbunden.

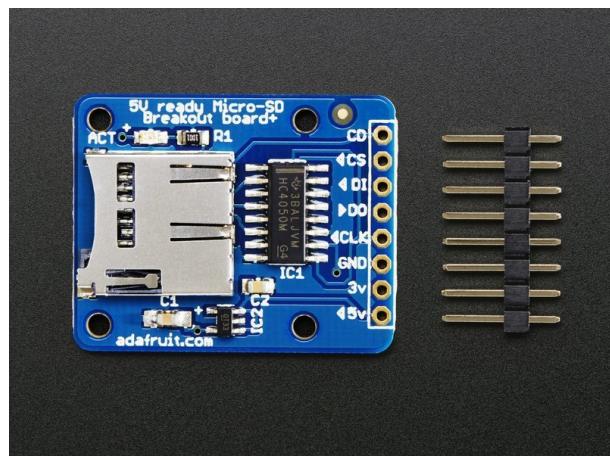


Abbildung 9.1: 254  $\mu$ SD-Breakoutboard von Adafruit ladyada2018

### Verdrahtung

SD-Karten erfordern viel Datenübertragung. Deshalb kann die beste Leistung erbracht werden, wenn sie an die Hardware-SPI-Pins eines Mikrocontrollers angeschlossen werden. Dabei wird es wie folgt miteinander verbunden: ladyada2018

Hier vielleicht noch das Schema hinzufügen wie es hardwaremäßig implementiert wird.

- **5V** und **GND** Pins jeweils auf die **5V** und **GND** Pins des Arduino Mega Boards
- **CLK** auf die Pinnummer **52**
- **DO** auf die Pinnummer **50**
- **DI** auf die Pinnummer **51**
- **CS** auf die Pinnummer **53**

<sup>2</sup>konvertiert eine high-level logik in eine low-level logik

## 9.2 $\mu$ SD-Karte



Abbildung 9.2: 16 GB  $\mu$ SD-Karte  
musdkarte

Bei der  $\mu$ SD-Karte muss auf die Kompatibilität mit dem Breakoutboard geachtet werden. Dafür sind folgende Kriterien zu beachten:

- Die  $\mu$ SD-Karte muss FAT16 oder FAT 32 formatiert sein.
- Es sind nur die SD und SD High Capacity (SDHC) kompatibel.

Für die Umsetzung dieses Projektes wurde eine  $\mu$ SD-Karte der SD-Familie SDHC I verwendet (siehe Abb. 9.2). SDHC sind Kapazitäten bis zu 32GB möglich und FAT32 formatiert.  
**muSDspez**

Es könnte noch auf die Anschlüsse der  $\mu$ SD-Karte eingegangen werden. Gäbe aber nur Sinn wenn ein Print erstellt werden muss wo das Breakoutboar nachkonstruiert wird.

### 9.3 Implementation in die Firmware

Für die Implementation in die Firmware, um mit dem Breakoutboard über SPI zu kommunizieren und die  $\mu$ SD-Karte zu beschreiben, resp. zu lesen, wurden direkt die bereits existierenden Librarys `<SPI.h>`<sup>3</sup> und `<SD.h>` von Arduino inkludiert. In der Arduino IDE können bereits vorgefertigte Example-Codes (siehe Abb. 9.3) zur weiteren Interpretation, wie mit diesen Librarys  $\mu$ SD-Karten gelesen und geschrieben werden können, verwendet werden.

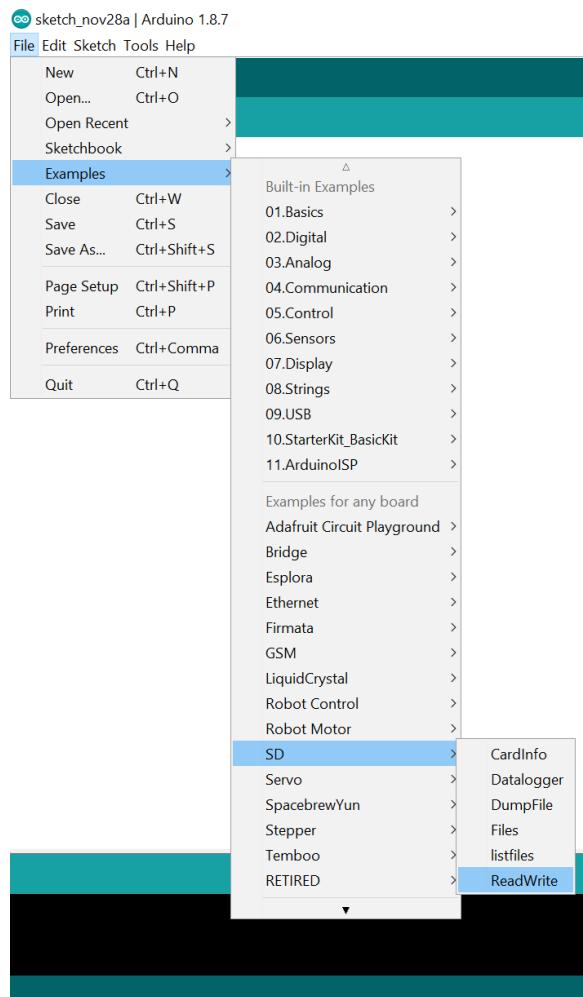


Abbildung 9.3: Example-Codes der Arduino IDE zum Lesen und Schreiben von SD-Karten.

Für eine übersichtlichere Programmstruktur und einfachere Handhabung wurden die Example-Codes für die korrekte Implementation angepasst und in Funktionen verpackt. Die Funktionen sind extern im Headerfile "SDCard.h"<sup>4</sup> deklariert und im SDCard.cpp initialisiert.

- `void getCardInformations()`
- `void readFileSDCard(String filename)`
- `void writeFileSDCard(double value2save, String filename)`
- `void deleteFileSDCard(String filename)`

Funktionen könnten noch beschrieben werden. Noch Abklären, ob die Fußzeilen nötig sind!

<sup>3</sup>`<*.h>` bezieht sich auf ein include-Verzeichnis unter dem Compiler-Installationsverzeichnis

<sup>4</sup>`"*.h"` bezieht sich relativ auf das aktive Projektverzeichnis

## 10 Firmware

### 10.1 Development Environment

Die Firmware der Wetterstation wurde im AtmelStudio 7 (Version: 7.0.1645 @2015) der Atmel Corp. und der Arduino IDE 1.8.5 geschrieben. Beide sind als Freeware erhältlich. Die Arduino IDE 1.8.5 wurde hauptsächlich verwendet, um notwendige Librarys über den *Library Manager* hinzuzufügen. Dies war besonders für die Sensoren wie der BME280 notwendig. Anschliessend konnte das gesamte Projekt ins AtmelStudio importiert werden. Der Vorteil liegt darin, dass nach dem ersten PCB-Entwurf dann mittels einem ISP<sup>5</sup> der Microcontroller programmierbar bleibt. Dies ist mit der Arduino IDE 1.8.5 nicht möglich.

In der C++ Syntax wurde das ganze Programm objektorientiert geschrieben und aufgebaut. Grund dafür ist die Skalierbarkeit des gesamten Projekts. Das objektorientierte Design ist besser strukturiert und kann einfach und übersichtlich erweitert werden.

### 10.2 UML-Diagramm

Das in der Abbildung 10.1 dargestellte UML-Diagramm soll den Aufbau und die logischen Zusammenhänge der Firmware visualisieren. Darin wird gezeigt, welche Headerfiles bei den Klassen benötigt werden. Zudem sind die Attribute, wie auch die Funktionen mit ihren Access Specifier, Argumenten und Rückgabewerten aufgelistet.

Durch diese Struktur ist es möglich, adaptiv mehrere Komponenten hinzuzufügen und anzupassen. Zudem könnten somit auch mehrere Sensoren vom gleichen Typ ohne grossen weiteren Aufwand implementiert werden.

In der Klasse **TempHumidPressSensor** werden die Temperatur, Lufdruck und relative Luftfeuchtigkeit vom BME280 gelesen und in das Struct BME280SensorData geschrieben. Diese können dann vom **Main** über die Funktion *getData()* aufgerufen werden. Über die Klasse **RealTimeClock** können die Zeitdaten vom DS3231 und zu den Messwerten zugeordnet werden. Mittels einer Klasse **Calculate** sollen notwendige Berechnungen vom restlichen Programm getrennt werden. Wenn das Programm startet, wird zuerst das *setup()* aufgerufen und danach befindet sich das Programm in der Endlosschlaufe *loop()*. Dieser Stil ist typisch für die Arduino IDE, wie im Kapitel 10.1 bereits erwähnt worden ist, wird in diesem Projekt in zwei Development Environments gearbeitet.

---

<sup>5</sup>In System Programmer

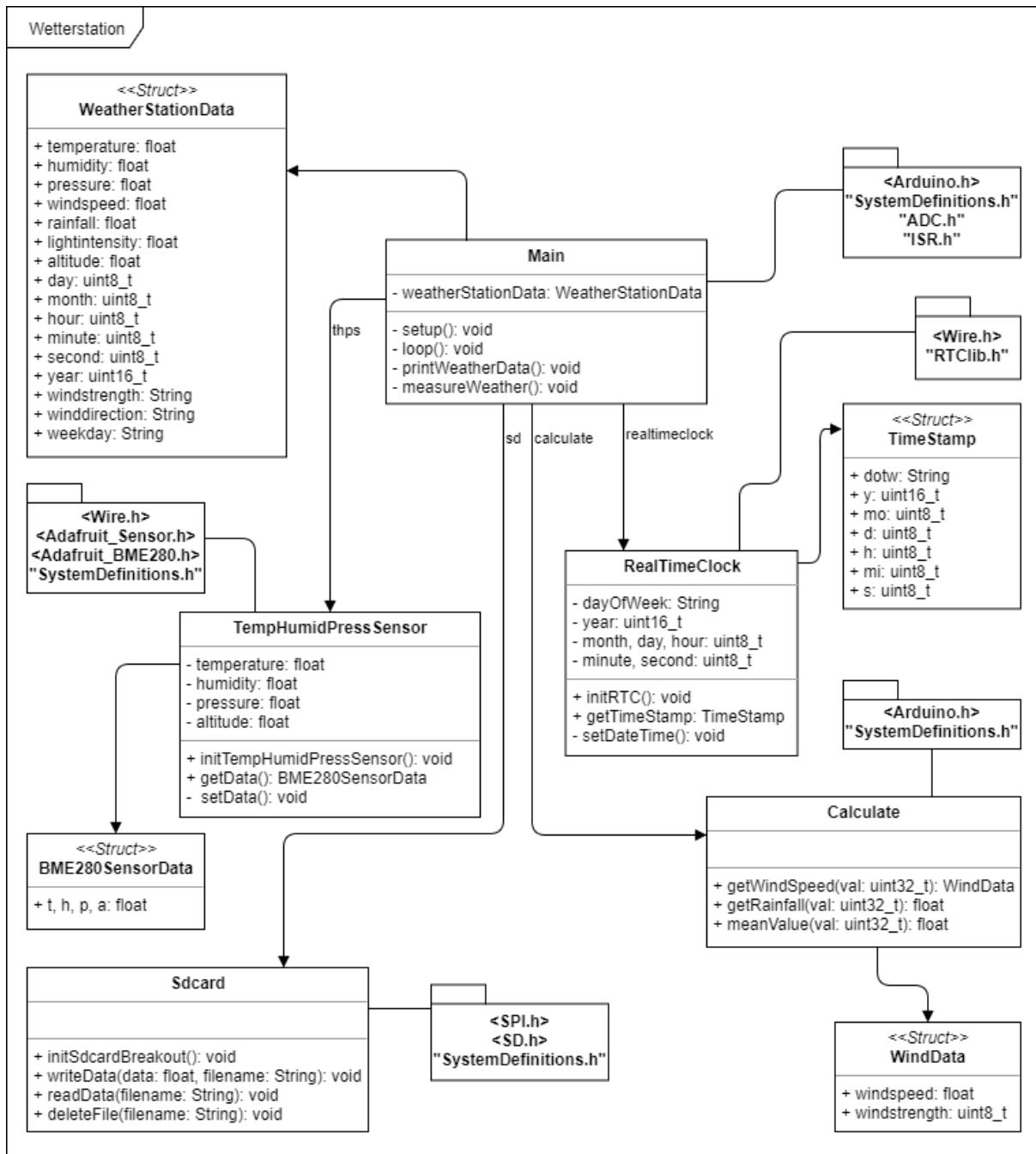


Abbildung 10.1: UML-Diagramm der Wetterstation

## 11 Konzeptvalidierung

In diesem Kapitel wird die Validierung des Gesamtkonzepts beschrieben. Da das Gesamtkonzept aus mehreren separaten Teilstücken besteht, reicht es aus die Teilsysteme auch separat zu testen. Zum einen sind da die jeweiligen Sensoren (für Lufttemperatur, Windgeschwindigkeit, Niederschlagsmenge, Luftfeuchtigkeit, Luftdruck und Windrichtung), zum anderen ist da das Speichern der Messdaten auf der  $\mu$ SD-Karte mit Zeitstempel.

### 11.1 Validierung der Sensorik

Die verschiedenen Sensoren besitzen Abweichungen, welche aus den entsprechenden Datenblättern entnommen sind und in den jeweiligen Kapitel thematisiert wurden. Dennoch müssten die Messwerte zusätzlich verifiziert werden. Um diese zu verifizieren wäre es jedoch notwendig, sehr genaue Referenzwerte zu haben über die gesamten Messbereiche. Da zu diesem Zeitpunkt keine Messvorrichtungen zu Verfügung stehen welche über entsprechenden Genauigkeiten besitzen, um Aussagekräftige Verifizierungen zu ermöglichen, konnte die Sensorik nicht validiert werden.

### 11.2 Validierung der Firmware

Die Validierung der Firmware soll die Funktionsweise des Programms bestätigen. Das Gesamtprogramm wurde getestet, indem durch die Sensorik Messdaten generiert, welche dann mit Zeitstempel auf die  $\mu$ SD-Karte gespeichert und von dort aus über die USB-Schnittstelle ausgegeben wurden. Durch die Abbildung 4.7, aus dem Kapitel 4.1, ist ersichtlich, dass die Funktion des Programms gewährleistet ist. Das Programm selbst besitzt dennoch Verbesserungspotenzial. Bei näherer Betrachtungsweise des UML-Diagramms (Abbildung 10.1) kann eine Redundanz festgestellt werden, welche herausgenommen werden kann.

### 11.3 Erreichte Ziele

Tabelle 11.1 zeigt die geforderten Ziele, deren Erfüllungsgrad und Verbesserungspotenzial. Die Ziele wurden in Kapitel 3 genauer definiert.

**Tabelle 11.1:** Ziele P5

	Ziel	Erfüllungsgrad	Verbesserungspotenzial
<b>Mussziele P5</b>			
Sensoren	Lufttemperaturmessung	teilweise erfüllt	
	Windgeschwindigkeitsmessung	komplett erfüllt	Referenzmessungen
	Niederschlagsmenge	teilweise erfüllt	Vergrösserung der Referenzfläche
Datenspeicherung	Datenabfrage via Putty	komplett erfüllt	
RTC	Implementation	komplett erfüllt	
<b>Wunschziele P5</b>			
Sensoren	Sonnenstunden Prototyp	nicht erfüllt	
<b>Zusätzliche Implementationen P5</b>			
Sensoren	Luftfeuchtigkeitsmessung	komplett erfüllt	
	Luftdruckmessung	komplett erfüllt	
	Windrichtungsmessung	komplett erfüllt	Genauere Richtungsangabe

Der Tabelle 11.1 kann entnommen werden, dass die meisten Ziele komplett erfüllt wurden. Das Wunschziel *Sonnenstunden Prototyp* wurde nicht erreicht, da durch die Implementation weiterer Sensoren die Zeit dafür nicht mehr reichte. Das Mussziel *Lufttemperaturmessung* wurde nur teilweise erfüllt, da der benutzte Sensor im Bereich von  $[-20;0]^\circ\text{C}$  eine Abweichung von  $\pm 1.25^\circ\text{C}$  anstelle der geforderten  $\pm 1^\circ\text{C}$  aufweist. Angesichts der klimatischen Verhältnisse in subtropischen Gebieten, für die diese Wetterstation entwickelt wird, ist dies jedoch vernachlässigbar, da die Temperatur in der Regel nie unter  $0^\circ\text{C}$  fällt **subtropklima**. Das Mussziel *Niederschlagsmenge* wurde nur teilweise erfüllt, da der benutzte Sensor eine kleine Trichterfläche hat und mit dem Skalierungsfaktor in der Software eine zu hohe Abweichung vom geforderten Genauigkeitsziel besitzt. Dem kann Abhilfe geschaffen werden, indem die Trichterfläche vergrössert wird. Die *Windrichtungsmessung* wurde zusätzlich implementiert und weist schwächen in Bezug der Windrichtungsangabe auf, welche Optimierungsbedarf erfordern.

## 12 Schluss

Wetterstationen liefern wichtige Klima- und Wetterdaten für örtliche Agronome. In subtropischen Gebieten wie Teile Afrikas und Südamerikas besitzen die Agronome diese Daten nicht, weshalb eine möglichst günstige, mobile Wetterstation entwickelt werden soll um diese zu unterstützen. In einem ersten Projekt soll die Sensorik und die Datenspeicherung dazu entwickelt werden.

Die Lufttemperatur, Luftdruck und Luftfeuchtigkeit wird über ein BME280 von Bosch gemessen. Die Windgeschwindigkeit über ein Anemometer von Froggit. Für die Windrichtung wird ein Windrichtungsgeber von Misol ermittelt. Über ein Ombrometer von Misol wird die Niederschlagsmenge eruiert. Die gesammelten Messdaten werden mit einem Zeitstempel, generiert von einem integrierten Präzisions-RTC (DS3231), versehen und auf eine  $\mu$ SD-Karte gespeichert. Kern der mobilen Wetterstation bildet eine MCU (ArduinoMega2560), welche über eine USB-Schnittstelle ansteuerbar ist.

Die Lufttemperatur wird im Bereich  $[0;60]^\circ\text{C}$  mit einer Abweichung von  $\pm 1^\circ\text{C}$  ermittelt und in einem Bereich von  $[-20;0]^\circ\text{C}$  mit einer Abweichung von  $\pm 1.25^\circ\text{C}$ . Die Luftfeuchtigkeit wird mit einer absoluten Genauigkeitstoleranz von  $\pm 3\%$  gemessen. Die Luftdruckmessung erfolgt mit einer absoluten Genauigkeit von  $\pm 1.7 \text{ hPa}$  in einem Temperaturbereich von  $[-20;0]^\circ\text{C}$  und mit  $\pm 1 \text{ hPa}$  in einem Temperaturbereich von  $[0;65]^\circ\text{C}$ . Die Windrichtung kann in Nord, Süd, West, Ost, Nordost, Südost, Südwest und Nordwest eingeteilt werden. Das Ombrometer misst auf einer Trichterfläche von  $6.325\text{e-}3 \text{ m}^2$  ungefähr 2 ml pro Kippbewegung. Das Anemometer misst eine nicht verifizierte Windstärke. Die Datenspeicherung mit Zeitstempel via MCU funktioniert einwandfrei.

In einem weiteren Projekt wird die Energieversorgung (Akku mit Solarunterstützung), die Kommunikation (Datenabfrage via SMS, GPS), sowie die Ermittlung der Sonnenstunden entwickelt. Wir empfehlen, dass die entwickelte Sensorik mit Hilfe von präzisen Messvorrichtungen verifiziert wird. Vor allem um die Windgeschwindigkeit, welche das Anemometer misst, verifizieren zu können. Außerdem soll die Trichterfläche des Ombrometers vergrössert werden, damit eine genauere Angabe auf eine Referenzfläche von  $1 \text{ m}^2$  erfolgen kann. Die Richtungsangabe des Windrichtungsgebers muss ebenso verbessert werden, damit die 8 genannten Windrichtungen besser erkannt werden.

## Abbildungsverzeichnis

4.1	Grundkonzept . . . . .	4
4.2	Modul Sensoren mit den verwendeten Sensoren und Messfühlern . . . . .	5
4.3	Prototyping mittels Breakoutboards . . . . .	6
4.4	BME280 Breakoutboard <b>LadyAdabme2802018</b> . . . . .	6
4.5	DS3231 Breakoutboard <b>LadyAda2018ds3231</b> . . . . .	6
4.6	$\mu$ SD-Card Breakoutboard <b>ladyada2018</b> . . . . .	6
4.7	Datenausgabe . . . . .	7
5.1	Einfacher SPI-Datenbus <b>Wikipedia2018spi</b> . . . . .	8
6.1	Arduino Mega Board <b>arduinoMega</b> . . . . .	9
7.1	<b>DS3231</b> mit seinen Anschlüssen <b>DS3231DS</b> . . . . .	10
8.1	Darstellung des Kipplöffelprinzips . . . . .	12
8.2	Selbsterstellter Kipplöffel. . . . .	13
8.3	Selbsterstelltes Gerüst. . . . .	14
8.4	Schaltkreis zur detektion der Kippbewegung. . . . .	15
8.5	Anemometer <b>AmazonAnemometer</b> . . . . .	16
8.6	RJ-11 Stecker . . . . .	16
8.7	Beschaltung des Ausgangs des Anemometers. . . . .	16
8.8	Ausgangssignal $V_{out}$ . . . . .	16
8.9	Graph der Messwerte . . . . .	17
8.10	Digitales Anemometer (Benetech GM8908) mit einer Auflösung von $0.1m/s$ und einer Unsicherheit von $\pm 5\%$ <b>digitalesAnemometerBenetech</b> . . . . .	17
8.11	Interne Schaltung <b>ADSkeineAngabe</b> . . . . .	18
8.12	Spannungsteiler mit $R = 10k\Omega$ <b>ADSkeineAngabe</b> . . . . .	19
8.13	Windrichtungsgeber von MISOL <b>windrichtungsgeber</b> . . . . .	19
8.14	BME280 <b>Bosch2019</b> . . . . .	20
9.1	254 $\mu$ SD-Breakoutboard von Adafruit <b>ladyada2018</b> . . . . .	22
9.2	16 GB $\mu$ SD-Karte <b>musdkarte</b> . . . . .	23
9.3	Example-Codes der Arduino IDE zum Lesen und Schreiben von SD-Karten. . . . .	24
10.1	UML-Diagramm der Wetterstation . . . . .	26
D.1	Top View . . . . .	38
D.2	Bottom View . . . . .	38
D.3	Side View . . . . .	38

## Tabellenverzeichnis

3.1	Ziele P5 . . . . .	3
6.1	Technische Daten <b>arduinoMega</b> . . . . .	9
7.1	Spezifikationen des <b>DS3231 DS3231DS</b> . . . . .	11
8.1	Technische Werte <b>ADSkeineAngabe</b> . . . . .	18
8.2	Elektrische Spezifikationen <b>Bosch2019</b> . . . . .	20
8.3	Sezifikationen der Feuchtigkeitsmessung <b>Bosch2019</b> . . . . .	20
8.4	Sezifikationen der Luftdruckmessung <b>Bosch2019</b> . . . . .	21
8.5	Sezifikationen der Temperaturmessung <b>Bosch2019</b> . . . . .	21
11.1	Ziele P5 . . . . .	28

## A Lastenheft



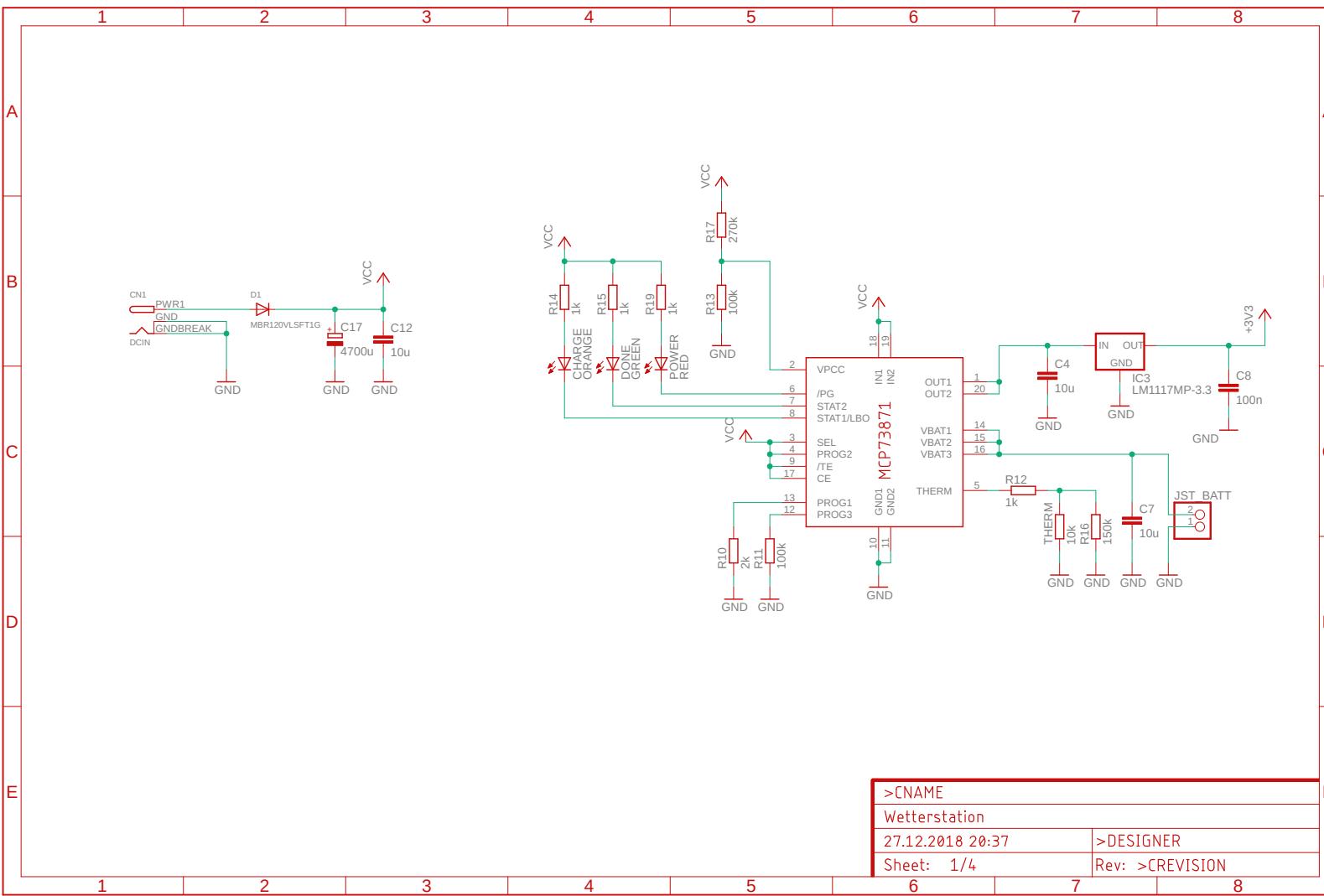
Fachhochschule Nordwestschweiz  
Hochschule für Technik

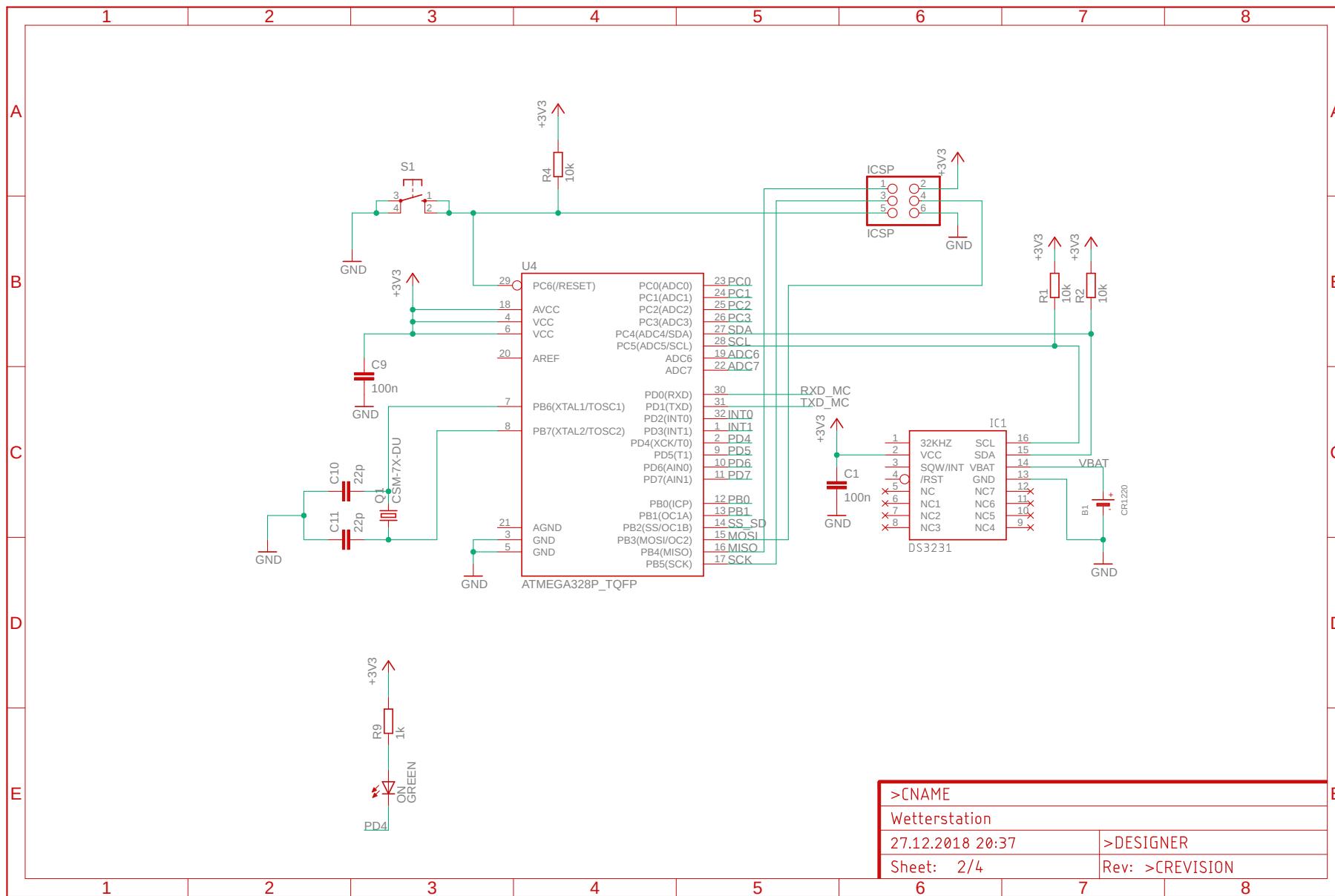
### **Ausschreibung Studierendenprojekt P5/P6 Studiengang Elektro- und Informationstechnik**

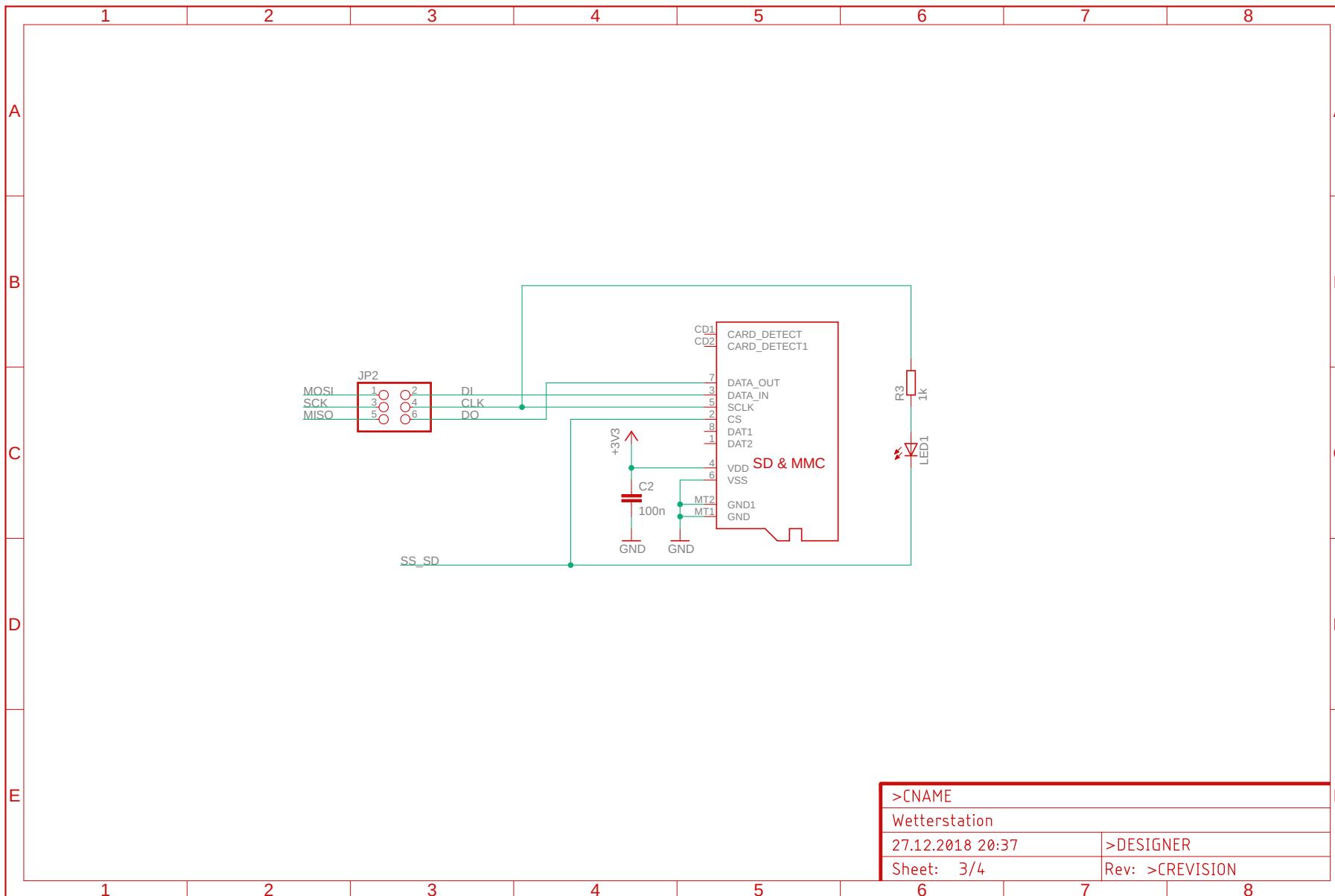
<b>Titel:</b>
Wetterstation mit Solar Energie
<b>Betreuer:</b>
Prof. Dr. Taoufik Nouri (Institut für Mobile und Verteilte Systeme)
<b>Auftraggeber:</b>
Prof. Dr. Taoufik Nouri (Institut für Mobile und Verteilte Systeme)
<b>Aufgabenbeschreibung:</b>
<p>Ausgangslage:          Wetterstation sind viele verlangt besonders im Gebiete ohne Strom. Wir schlagen solche Möglichkeit zu realisieren.</p> <p>Zielsetzung:          1. Diese Wetterstation misst Regen, Wind- Geschwindigkeit, -Richtung, Temperatur, Sonnenlicht, Feuchtigkeit, Zeit usw.          2. Sie ist dotiert mit verschiedener Kommunikation Module wie GPS, SIM Karte.          3. Sie ist fern abfragbar durch Handy          4. Sie speichert regelmässig die verschiedenen Parameter (Journal).          5. Sie ist komplett automatisiert z.B. Regenwasser wird automatisch ausgeleert.</p> <p>Schlüsselwörter: Energie, Mikrokontroller, Programmierung, Elektronik</p>

## B Schema

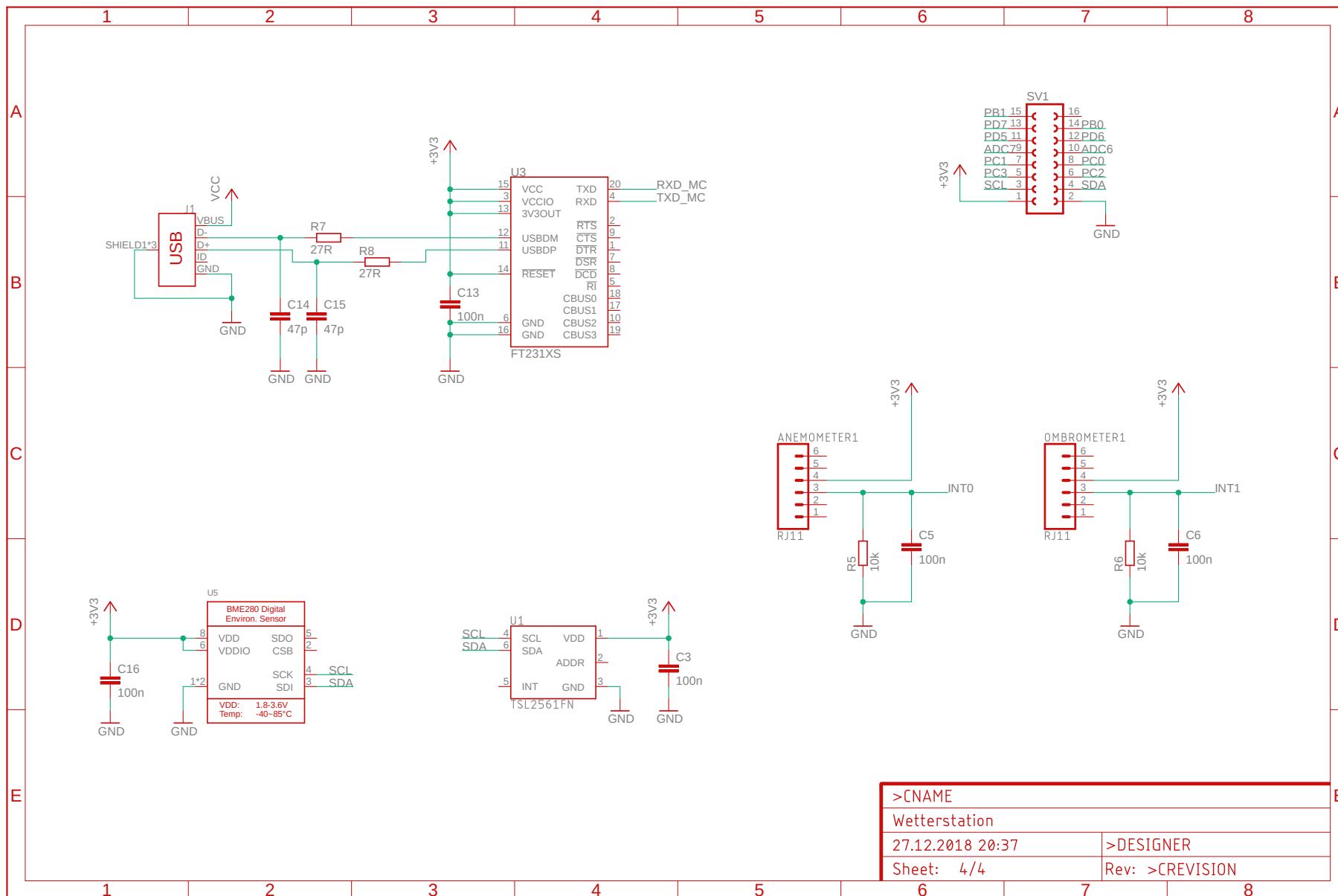
Hier wird das Schema der Wetterstation gezeigt, Dies ist ein Entwurf, welches anhand der Breakoutboards entworfen wurde und kann sich daher noch ändern.





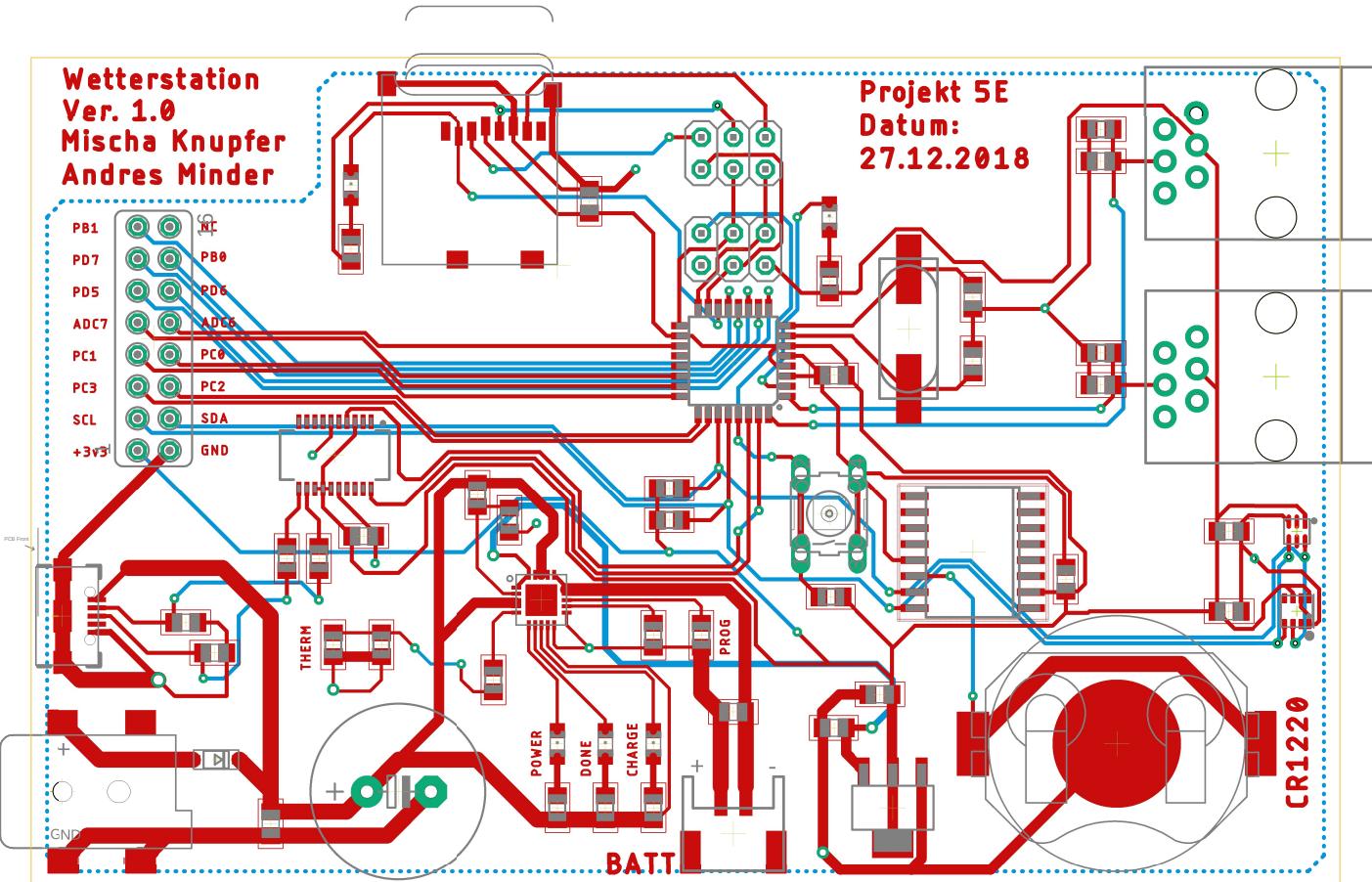


>CNAME	E
Wetterstation	
27.12.2018 20:37	>DESIGNER
Sheet: 3/4	Rev: >CREVISION



## C PCB-Design (Top View)

Ein provisorisches PCB-Layout wurde designed, damit in einem weiterführenden Projekt Zeit eingespart werden kann. Im P5 wurde gemäss Aufgabenstellung jedoch auf ein Print verzichtet und nach Kapitel 4.1 vorgegangen.



## D 3D-Modellierung

Die 3D-Modellierung zeigt ein 3D-Modell des erstellten provisorischen PCB-Layouts. Dieses PCB-Layout, und damit auch das anschauliche 3D-Modell, sind nur provisorisch und können sich noch ändern.

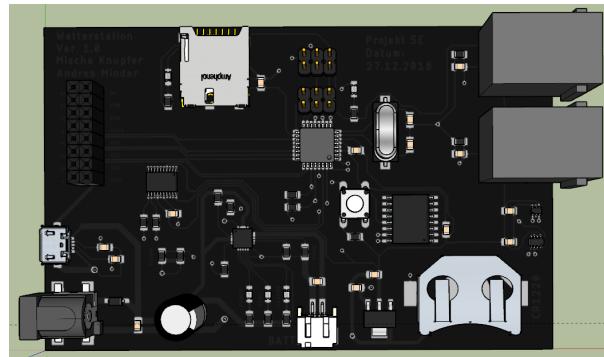


Abbildung D.1: Top View

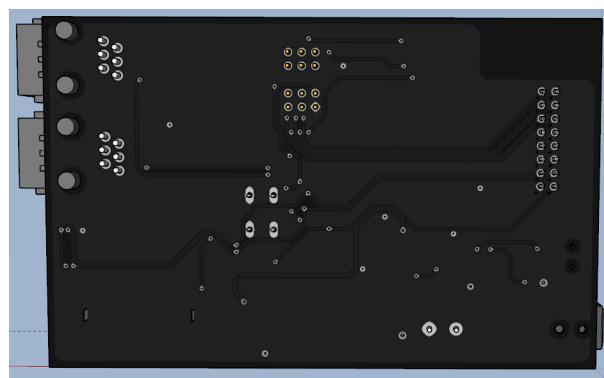


Abbildung D.2: Bottom View

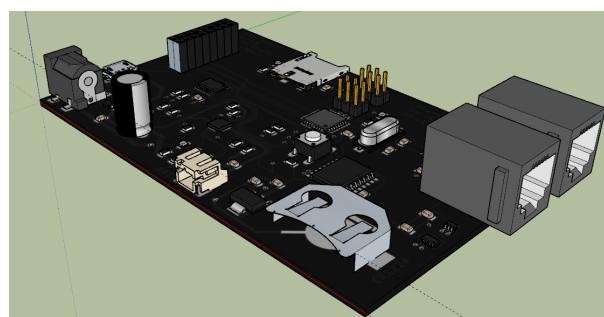


Abbildung D.3: Side View