DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

# Entwicklung eines CO2-Messers zur Simulation einer automatisierten Fensteransteuerung

## Mikrocomputertechnik - Bericht

Studiengang Elektrotechnik

Studienrichtung Fahrzeugelektronik

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Alexander Herrmann  Johannes Ruffer  Serkant Soylu

| | |
|---|---|
| Abgabedatum: | 19.04.2020 |
| Bearbeitungszeitraum: | 01.10.2019 - 19.04.2020 |
| Matrikelnummer: | 9859538 x 1011921 x 9964027 |
| Kurs: | TFE18-2 |
| Gutachter der Dualen Hochschule: | Hans Jürgen Herpel |

# Eidesstattliche Erklärung

Gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2015.

Wir versichern hiermit, dass wir unsere Projektarbeit mit dem Thema:

> *Entwicklung eines CO2-Messers zur Simulation einer automatisierten Fensteransteuerung*

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Friedrichshafen, den 31. März 2020

| Alexander Herrmann | Johannes Ruffer | Serkant Soylu |
|---|---|---|

Autoren

# Kurzfassung

Der Bericht wurde von drei Studierenden an der Duale Hochschule Baden-Würtemberg (DHBW)-Ravensburg Campus Friedrichshafen im Rahmen der Mikrocomputertechnik Vorlesung eigenständig von der Projektplanung, über die Durchführung, bis hin zum Projektabschluss mit Dokumentation durchgeführt.
Ziel der Arbeit ist es, durch die Praxiserfahrung mit Mikrocomputern, Fähigkeiten und Wissen in diesem Bereich zu erwerben. Durch das Vergleichen von anfänglichen Kosten- und Komplexitätsschätzungen mit den späteren Ergebnissen in der Umsetzung können die Studierenden ein Fazit ziehen, inwiefern diese übereinstimmen. So werden auch Fähigkeiten im Bereich des Projektmanagements weiterentwickelt.

# Inhaltsverzeichnis

# 1. Einleitung

Mithilfe eines Arduinos wurde in diesem Projekt ein CO2-gesteuerter Fensterheber simuliert, welcher dazu dienen soll die Räumlichkeiten bei schlechter Luftqualität automatisch zu lüften. Auch das automatische schließen des Fensters nach Wiederherstellung von guter Luftgüte wird simuliert.
Das Öffnen und Schließen der Fenster wird anhand von einer LED simuliert, welche nach der Überschreitung eines bestimmten Grenzwertes aufleuchtet. Sobald nach ausreichendem Lüften die Luftgüte unter einen Schwellwert gerät, soll die LED wieder aus gehen.
Damit dies möglich ist, wurde die Entwicklung von Software-Code, sowie ein Schaltungslayout und der 3D-Druck des Gehäuses selbständig vorgenommen.
Der folgende Bericht dokumentiert die Vorgehensweisen und Umsetzung von der Projektplanung, über die Implementierung von Hardware und Software, bis hin zu durchgeführten Tests und den Projektabschluss.

# 2. Anforderungen

Damit die Bewertung des Projektes erfolgreich wird, müssen zu Projektbeginn Anforderungen erarbeitet und festgelegt werden. Diese sind unveränderbar, da das Ergebnis sonst verfälschen würde.
Eine Anforderung muss zudem messbar und/oder überprüfbar sein, um deren Umsetzung später objektiv bewerten zu können. Somit wird auch eine sogenannte Verifikationsmethode festgelegt, mit der die Anforderungen später überprüft werden können.
Es wird zwischen sechs verschiedenen Verifikationsmethoden unterschieden:

- Similarity: Suche und Abgleich mit bereits vorhandenen Lösungen [Hel, S. 122]
- Insparity: Soll- und Ist-Abgleich von Ein- und Ausgängen nach einem formalen Ablauf (Ein- und Ausgangskriterien sind vorher definiert) [Pet09, vgl. S. 308]
- Review: Soll- und Ist-Abgleich von Ein- und Ausgängen ohne formalen Ablauf (Ein- und Ausgangskriterien sind vorher nicht definiert) [Pet09, vgl. S. 317]
- Measurement: Durchführung einer Reihe von Operationen, die das Ziel haben, einen Wert einer quantitativen oder kategorialen Darstellung eines oder mehrerer Attribute zu bestimmen [Dep, vgl. S. 395]
- Analysis: Basiert auf Heuristiken und Statistiken [...] [, die man] sich als starke Compiler-Typisierung [...] im Rahmen einer ausführlichen Datenfluss-Analyse vorstellen [kann] [Jay, vgl. S. 4]
- Test: Prüft und bewertet Software auf Erfüllung der für ihren Einsatz definierten Anforderungen und misst ihre Qualität [Wik20a]

Daher wurde am Anfang des Projektes eine Tabelle erstellt, indem alle Anforderungen definiert wurden. Zudem sind diese nummeriert und haben individuelle Verifikationsmethoden zugeord-

net bekommen.

| Nummer | Anforderungen | Verifikationsmethode |
|---|---|---|
| 1 | Echtzeitmessung der Luftgüte | Measurement |
| 2 | Mindestmessbereich von 400 ppm bis 5000 ppm | Review |
| 3 | Visualisierung der Luftgüte mithilfe von LEDs (gut, mittel, schlecht) | Test |
| 4 | Ausgabe der Luftgüte mithilfe von LiquidCrystal Display (LCD)-Display | Test |
| 5 | Ansteuern eines Fensterscheibenmotors mithilfe einer LED simulieren | Test |
| 6 | Bei schlechter Luftgüte: Fenster öffnet sich (LED an) | Test |
| 7 | Bei guter Luftgüte: Fenster schließt sich (LED aus) | Test |
| 8 | Speichern im CSV-Format | Test, Analysis |
| 9 | Zugriff auf Messdaten über SD-Karte | Test |
| 10 | Benutzer kann zwischen drei Messprofilen auswählen (Messprofil: Abtastrate) | Test |

**Tabelle 2.1.:** Anforderungen an das Projekt

Die Anforderungen sollen jedoch nicht nur mithilfe einer Tabelle und den dazugehörigen Verifikationsmethoden definiert werden. Wie in 2.1 zu sehen ist, wurde mithilfe von Architect Enterprise ein sogenanntes Use-Case-Diagramm erstellt, welches den Zusammenhang zwischen Anwender und Programmfunktionen aufzeigen soll.
Zu sehen ist, dass der Anwender zunächst zwischen zwei Funktionen wählen kann. So soll zum einen eine neue Messung gestartet werden können. Zum anderen soll der Benutzer in der Lage sein, dem Programm mitteilen zu können, dass die letzte Messung derzeit ausgelesen wird.
Im Fall, dass der Anwender eine neue Messung starten möchte, kann dieser, wie in 2.1 definiert wurde, zwischen drei Messprofilen wählen.
Falls der Benutzer dem Programm zu verstehen gegeben hat, die letzte Messung auslesen zu wollen, geht das Programm in einen Wartezustand. Hier wartet es so lange, bis der Anwender bestätigt hat, dass das Auslesen erfolgreich beendet wurde und die SD-Karte wieder ins Modul eingeführt wurde.

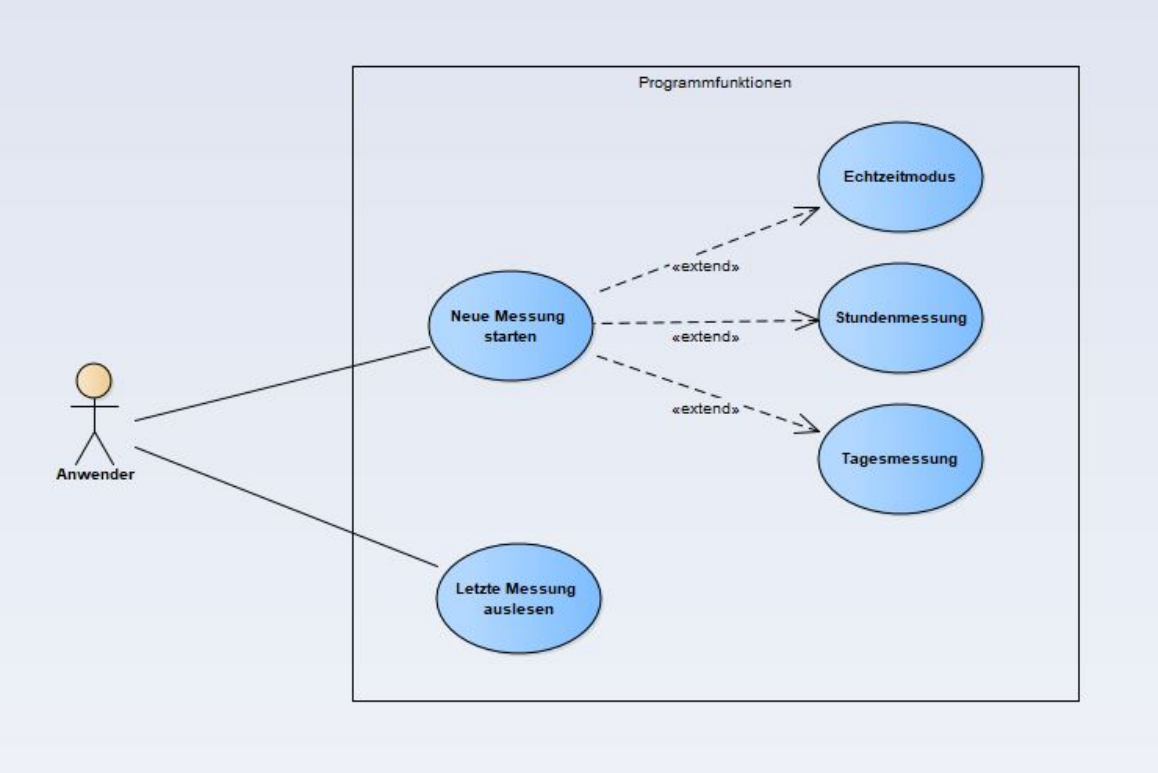



**Abbildung 2.1.:** Entwurf des Use-Case-Diagramms mithilfe von Enterprise Architect

# 3. Projektplan

Der Projektplan dient dazu, das Projekt auf Probleme zu minimieren und eine spätere Bewertung zu ermöglichen. In unserem Projektplan wurden Zeit und Kosten zu einem Portfolio auf Basis von Recherchen und eigenen Ideen zusammengetragen.

## 3.1. Arbeitsplan

Damit wir zu jedem Zeitpunkt einschätzen können, ob wir im Zeitplan sind, haben wir zu Beginn des Jahres den in Abbildung 3.1 zu sehenden Projektplan erstellt. An diesen Ablauf inklusive den Daten, hat sich jedes Gruppenmitglied zu halten. Eine Verzögerung in der Entwicklung, bedeutet eine Verschiebung der Abwicklung des Projektes. Davon hängt somit die Fertigstellung, als auch der Erfolg des Projektes ab.

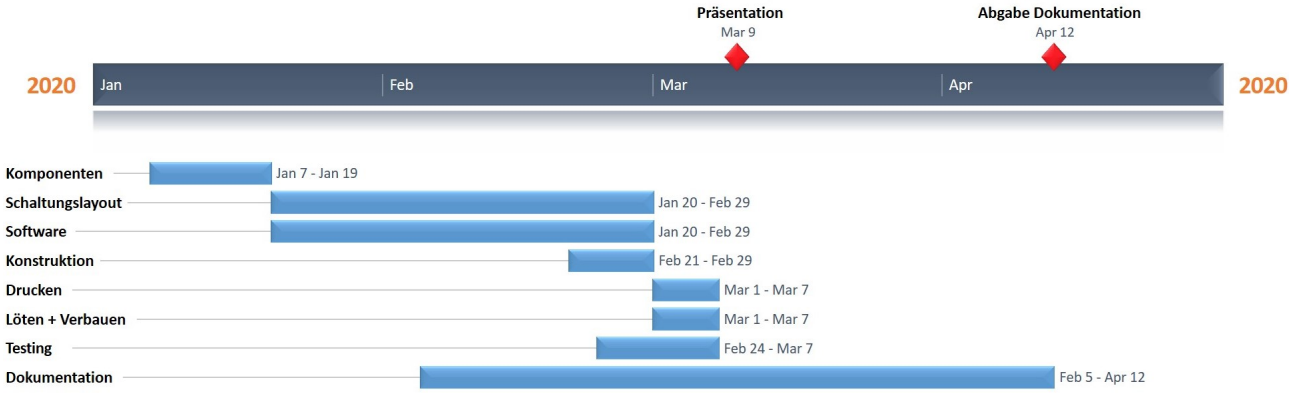



**Abbildung 3.1.:** Ablauf des Projektes von Januar bis April

Auch, wenn die Abgabefrist erst vier Wochen nach Semesterende abläuft, haben wir uns, wie in Abbildung 3.1 zu sehen ist, den 12. April als Meilenstein für die Abgabe des Projektes gesetzt. Die Begründung für diese Entscheidung liegt zum einen darin, dass wir bei Problemen bezüglich der Abgabe einen gewissen Puffer haben. Zum anderen beginnt ab Anfang April die

dritte Praxisphase für alle dualen Studierenden, in welcher diese ab Mitte des Monats schon in ihrem neuen Projekt vertieft sind und somit weniger Zeit für das Mikrocomputertechnik-Projekt aufwenden können.

## 3.2. Kostenplan

Die Kosten des Projektes belaufen sich, wie in 3.1 nachzuvollziehen ist, auf 77,13€. An dieser Stelle muss jedoch angefügt werden, dass es sich hierbei nicht um die reinen Materialkosten für das Projekt handelt, da beispielsweise das Arduinoset einiges mehr, als nur die benötigten Materialien geboten hat. Somit kann man sagen, dass sich die Kosten bei einer Bestellung mit Einschränkung auf die benötigten Komponenten senken würden.

| Einkauf | Kosten |
|---|---|
| Arduinoset | 27,95€ |
| CO2-Sensor (CCS811) | 8,50€ |
| MikroSD-Modul | 3,50€ |
| LED-Fassung | 4,59€ |
| $I^2C$-Adapter | 3,99€ |
| Mechanische Bauteile | 24,99€ |
| Sonstiges | 4,56€ |
| Gesamt | 77,13€ |

**Tabelle 3.1.:** Aufgewendete Kosten für das Projekt

# 4. Entwurf

Bevor es an die Umsetzung von Soft- und Hardware ging, wurden für die verschiedenen Gebiete Zeichnungen und Diagramme angefertigt. Zum einen diente es dazu, das Vergessen von Anforderungen zu vermeiden. Zum anderen konnten die bevorstehenden Aufgaben so besser verteilt werden. Auch die Vorstellungen der einzelnen Gruppenmitgliedern in Bezug auf die Umsetzung des Projektes wurden mithilfe dieser Methode zusammengeführt, um spätere Differenzen zu vermeiden.

## 4.1. Architektur und Verhalten

Damit der genaue Ablauf unseres Programms definiert ist, wurde mithilfe von Enterprise Architect ein Zustandsdiagramm erstellt.
Dieses Diagramm ist dazu da, die Zustände, in welches sich das Objekt befindet, zu präsentieren. Auch die Übergänge zwischen den einzelnen Zuständen wird hier deutlich. Zudem zeigt es den Anfangs- und Endpunkt einer Sequenz von Zustandsänderungen. [Jos00, vgl. S. 120]
Auf unser Projekt übertragen bedeutet es, dass das Diagramm vom Einschalten der Hardware, durch Stromversorgung des Arduinos, über jede Kombinationsmöglichkeiten der Taster, bis hin zum wieder Ausschalten der Hardware, alle Zustände zeigt, in welche das Programm kommen kann. Dies soll später dem Softwareentwickler die Arbeit vereinfachen. Des Weiteren können auch Tests mithilfe des Zustandsdiagramms leichter durchgeführt werden, da die Eingangsvariablen mit den darauffolgenden Zustandsänderungen, wie in 4.1, eindeutig zu erkennen sind.

Das Komponentendiagramm, welches in 4.2 zu sehen ist, soll dazu dienen, alle Aktoren und deren Verbindungen darzustellen.

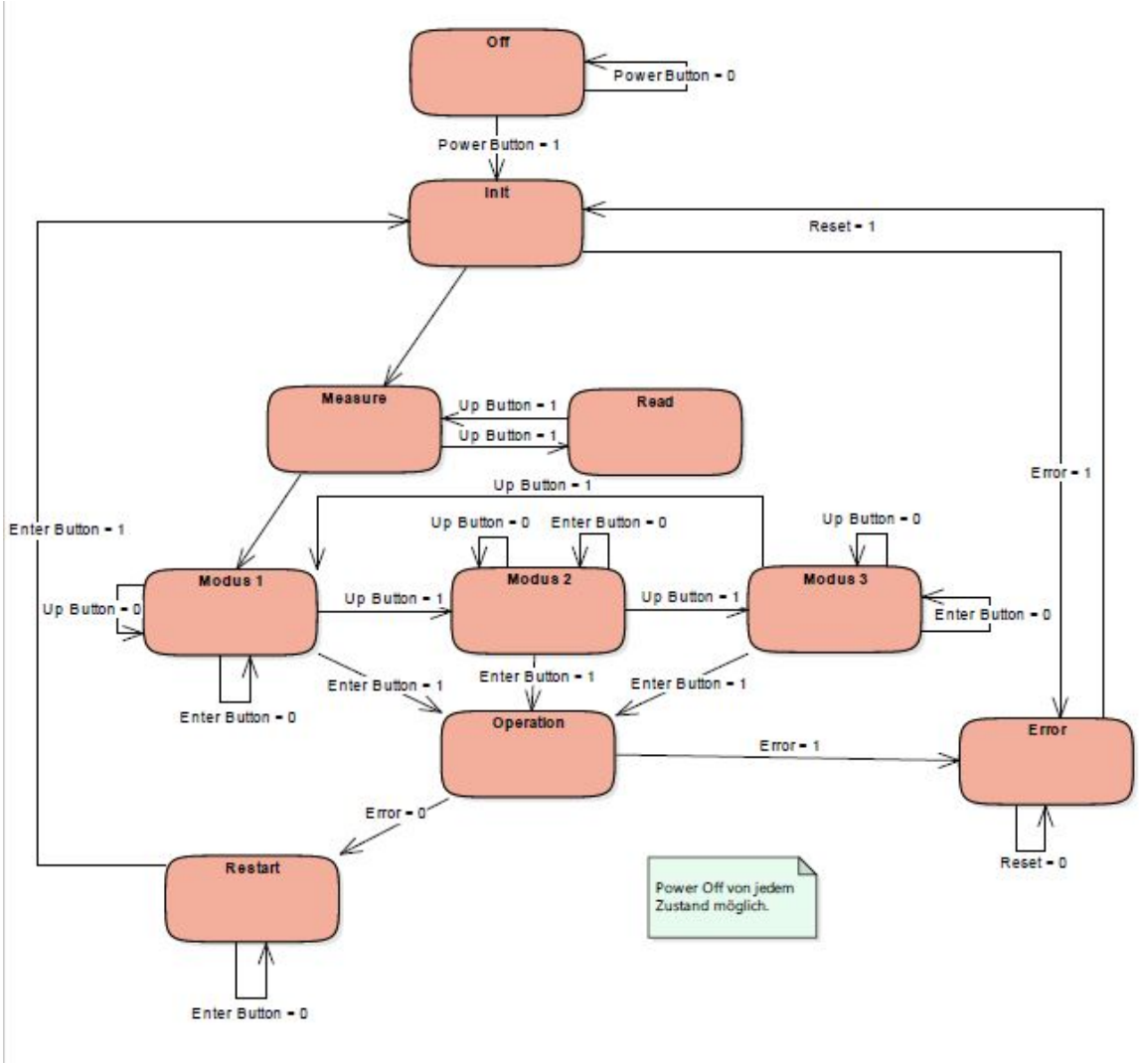



**Abbildung 4.1.:** Entwurf des Zustandsdiagramms mithilfe von Enterprise Architect

## 4.2. Schaltungslayout

In Abbildung 4.3 ist das Schaltungslayout des Projektes zu sehen, welches wir mithilfe von Fritzing angefertigt haben.
Das LCD-Display ist an sechs Pins mit dem Arduino verbunden. Zudem benötigt es eine Versorgungsspannung von ca. 5 Volt. Der Eingang V0 ist dabei über einen Widerstand auf Masse geschalten. Je nach Höhe des Widerstandes ändert sich der Kontrast des LCD-Displays.
Darunter befindet sich in Abbildung 4.3 der CO2-Sensor CCS811, welcher über zwei Pins an den Arduino angeschlossen ist. Einen extra Anschluss an die Versorgungsspannung benötigt dieser nicht, da er die maximal benötigten 3.6 Volt über den $I^2C$-Anschluss bekommt.
Unterhalb des Co2-Sensors ist das SD-Karten-Modul zu sehen, welches eine Versorgungsspannung von 5 Volt auf Masse benötigt. Die restlichen vier Anschlüsse sind mit den Pins 8 bis 11

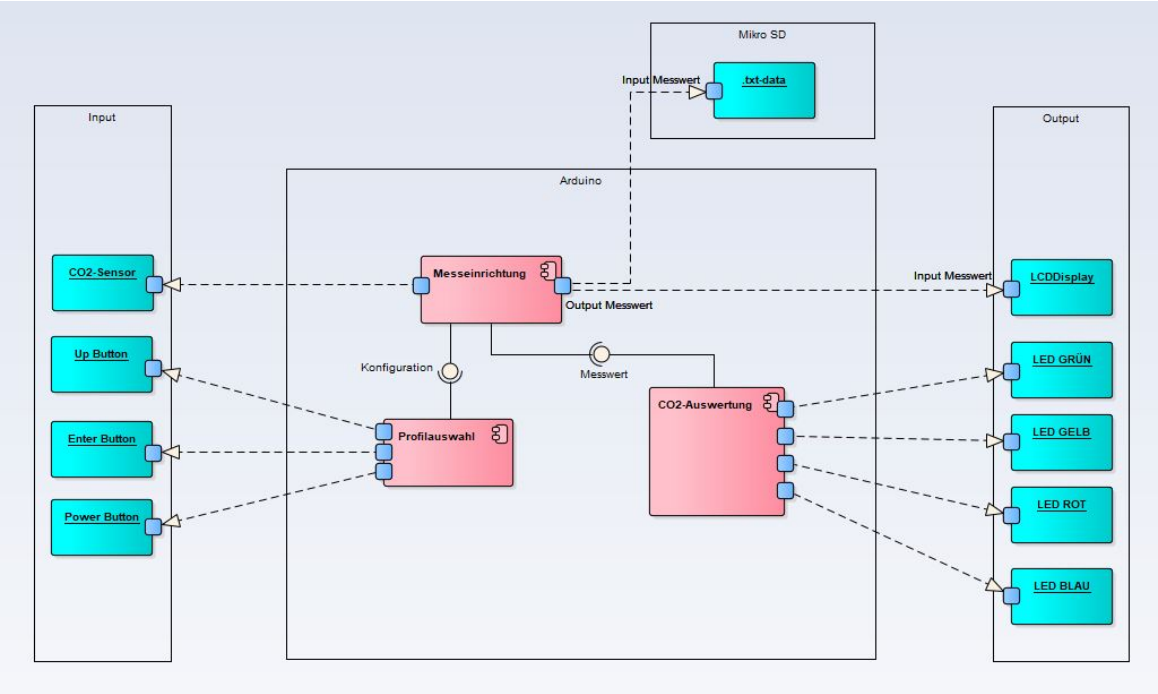



**Abbildung 4.2.:** Entwurf des Komponentendiagramms mithilfe von Enterprise Architect

am Arduino verbunden.
Zwei von den oben zu sehenden Taster dienen für die Menüauswahl auf dem LCD-Display. Einer soll dabei für das Wechseln innerhalb des Menüs dienen, der andere ist für die Bestätigung der Auswahl zuständig. Damit die Schaltung auch ein- und ausgeschaltet werden kann, ohne die Versorgungsspannung zu kappen, haben wir einen dritten Taster eingebaut. Die Vorwiderstände aller Taster belaufen sich auf 2kΩ.
Die vier LEDs sind, wie in den Anforderungen definiert, für die Visualisierung der Luftgüte zuständig. Dabei dienen die Farben Grün, Gelb und Rot für gute, mittlere und schlechte Luftgüte. Die blaue simuliert den angeschlossenen Fensterscheibenmotor. Für die Vorwiderstände der vier LEDs haben wir jeweils 330Ω gewählt.

## 4.3. Gehäuse

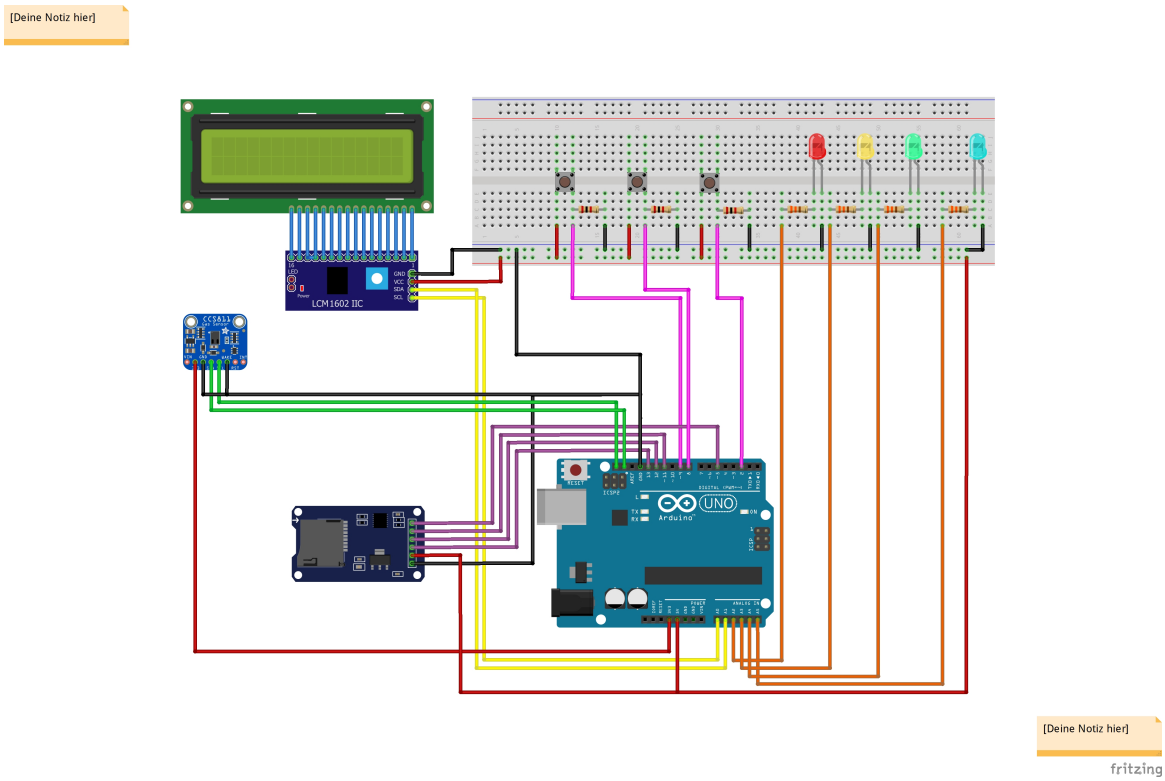



**Abbildung 4.3.:** Entwurf des Schaltungslayouts mithilfe von Fritzing

# 5. Softwareimplementation

Für die entgültige Struktur der Software waren mehrere Anforderungen ausschlaggebend.
Zunächst wurden drei verschiedene Messprofile definiert und implementiert, sodass der Benutzer zu Beginn zwischen einer Echtzeit-, Stunden- und Tagesmessung wählen kann. In den Messprofilen ist definiert, in welchen Abständen und wie lange Messungen durchgeführt werden sollen. Somit konnten Anforderung Nummer 1 und 10 aus der Tabelle 2.1 gemeinsam umgesetzt werden.
Nach der Wahl des Messprofils muss der Arduino den $CO_2$-Sensor ansteuern und richtig konfigurieren. Es muss getestet werden, ob er funktionstüchtig und bereit ist eine Messung zu starten. Zudem kann der verwendete Sensor nicht nur $CO_2$-Werte, sondern beispielsweise auch Temperaturen messen, sodass der Arduino die richtigen Werte anfordern muss. Damit dies möglich ist, mussten wir die Bibliothek <Adafruit_CCS811.h> einbinden. Genaueres zu diesem Algorithmus wird in Kapitel 5.3 erläutert.
Zunächst wird im Setup geprüft, ob der Sensor gestartet werden kann. Falls dies nicht der Fall sein sollte, wird eine Fehlermeldung ausgegeben. Nach erfolgreichem Start ist der Arduino angehalten so lange mit dem Programm zu warten, bis der Sensor zurückmeldet, dass er bereit ist, die Messung zu beginnen.
Auch während dem Programmdurchlauf wird bei jeder neuen Messung kontrolliert, ob der $CO_2$-Sensor funktionstüchtig ist. Danach wird mithilfe der oben genannten Bibliothek der $CO_2$-Wert gemessen und an den Arduino weitergegeben.
Nach Einlesen der Daten, vergleicht der Mikrocomputer diese mit den gegebenen Grenzwerten. Je nach Bewertung des gemessenen Werte wird eine der grün/gelb/roten LEDs eingeschaltet. Auch die blaue LED, welche die Ansteuerung des automatisierten Fenstersscheibenmotors simulieren soll, wird je nach Messwert an- oder ausgeschaltet.
Zudem werden dem Anwender in Echtzeit die jeweiligen Daten im LCD Display aufgegeben, was durch die Bibliothek <LiquidCrystal.h> möglich ist.
Damit der Verlauf der Messung später auf Excel geplottet werden kann, wird auf einer Mikro-SD-Karte der gemessene Wert im .csv-Format als .txt-Datei abgespeichert.
Der Anwender nach nun entscheiden, ob er eine weitere Messung durchführen möchte oder nicht. Bei positiver Eingabe wird das Programm von vorne durchgeführt, während beim Ab-

lehnen einer weiteren Messung der Mikrocomputer in einen sogenannten Sleep-Mode geht.

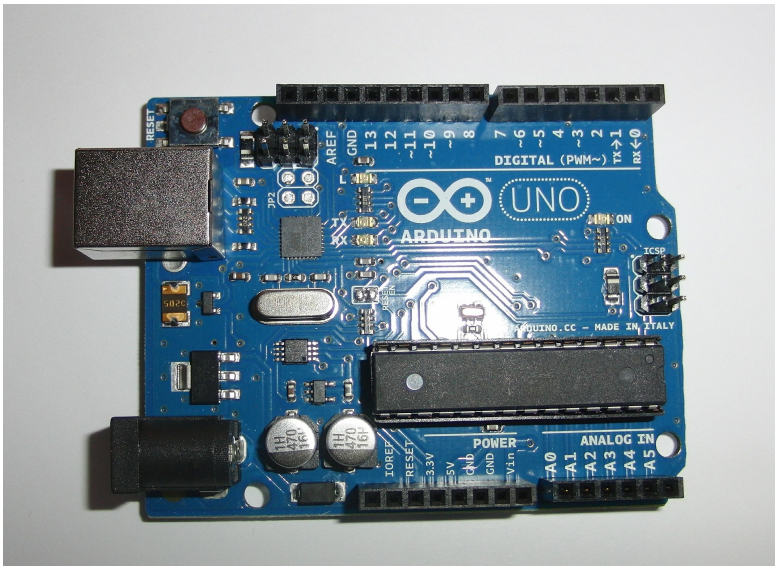
## 5.1. Arduino vs. Raspberry PI

Nach dem Definieren der Anforderungen musste entschieden werden, mit welchem Mikrocontroller oder auch Einplatinencomputer das Projekt umgesetzt werden soll. Hierbei wurden ein Arduino und ein Raspberry PI in Betracht gezogen.
Die Vorteile des Arduinos liegen darin, dass ein sofort einsatzbereites Hardware-/Software-Setup zu Verfügung steht. Des Weiteren hat diser Mikrocontroller eine eigene Entwicklungsumgebung mit plattformübergreifenden Bibliotheken, von welchen wir Gebrauch machen müssen. Zudem ist Arduino eine Plattform, auf Basis von Open-Source-Lizenzen. Dies hat den Vorteil, dass jeder Entwickler sowohl die Quellcodes der Software, als auch die Pläne der Hardware einsehen und für das jeweilige Projekt individuell anpassen kann. [Mir18, vgl.] Nachteile, wie die kostspielige Aufrüstung von Shields sind für unsere Verwendungen nicht von Bedeutung.
Vielmehr überwiegten die Nachteile des Raspberry PI, für welchen man kostenpflichtige Zusatzteile für den eigenständigen Betrieb benötigt. Auch die Vorteile des Einplatinencomputers, wie die Netzwerkfähigkeit und den standardmäßigen High Definition Multimedia Interface (HDMI)-Anschluss, werden in diesem Projekt nicht benötigt und spielen für uns somit keine Rolle. [ION18, vgl.]
All diese Punkte führten letztendlich zu der Entscheidung, einen Arduino und keinen Rapberry PI für das Projekt zu nehmen.

## 5.2. Arduino

Der Arduino wurde 2005 von den beiden Entwicklern Massimo Banzi und David Cuartielles entwickelt. Der Mikrocomputer wurde nach einer Bar, in welcher sich die Entwickler gerne und oft trafen, benannt. Diese wiederum bekam ihren Namen nach Arduin von Ivrea, welcher von 1002 bis 1014 König von Italien war. [Wik20b, vgl.]
Aufgrund der hohen Anzahl von verschiedenen Arduino-Modellen, mussten wir uns darüber informieren, welches am besten für unser Projekt geeignet ist. Da wir uns selbst am Anfang des Projektes als Arduino-Einsteiger bezeichneten, wurde uns der in 5.1 zu sehende Arduino Uno empfohlen. Aufgrund von Rechercheergebnissen, dass dieses Modell das bekannteste und meist genutzte Arduino Board ist, entschieden wir uns dazu, dieses Modell zu nutzen. Diese

Entscheidung hat den Hintergrund, dass uns so sehr viele Tutorials und Projektbeispiele im Internet zur Verfügung stehen. Diese sollen uns bei anfänglichem Erwerb von Grundwissen und späteren Projektproblemen weiterhelfen. [Gen16, vgl.]
Neben 13 digitalen und sechs analogen In- und Output-Pins besitzt der Arduino einen USB-B-Port, mit welchem das Laden eines neuen Programms ermöglicht wird. Die empfohlene Versorgungsspannung liegt beim Arduino Uno zwischen 7V und 12V. [ser20, vgl. S. 2] Er besitzt, wie in 5.1 zu sehen, DC Pins von 5V und 3,3V DC-Spannung.
Speichermöglichkeiten gibt es auf dem Arduino zum Einen auf dem ATmega328, welcher eine Speicherkapazität von 32KB besitzt. Darauf werden allerdings 0,5KB vom Arduino Bootloader belegt. Zum Anderen verfügt er über 2KB SPRAM und 1KB EEPROM. [ser20, vgl. S. 2] Das ist auch der Grund dafür, weshalb die Daten später nicht auf dem Arduino selbst, sondern auf einer Mikro-SD-Karte gespeichert werden soll. Näheres dazu in Kapitel 5.4.

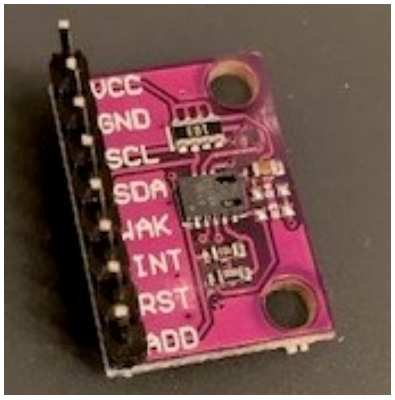



Quelle: https://pixabay.com/de/photos/arduino-computer-cpu-373994/

**Abbildung 5.1.:** Arduino Uno

## 5.3. CCS811

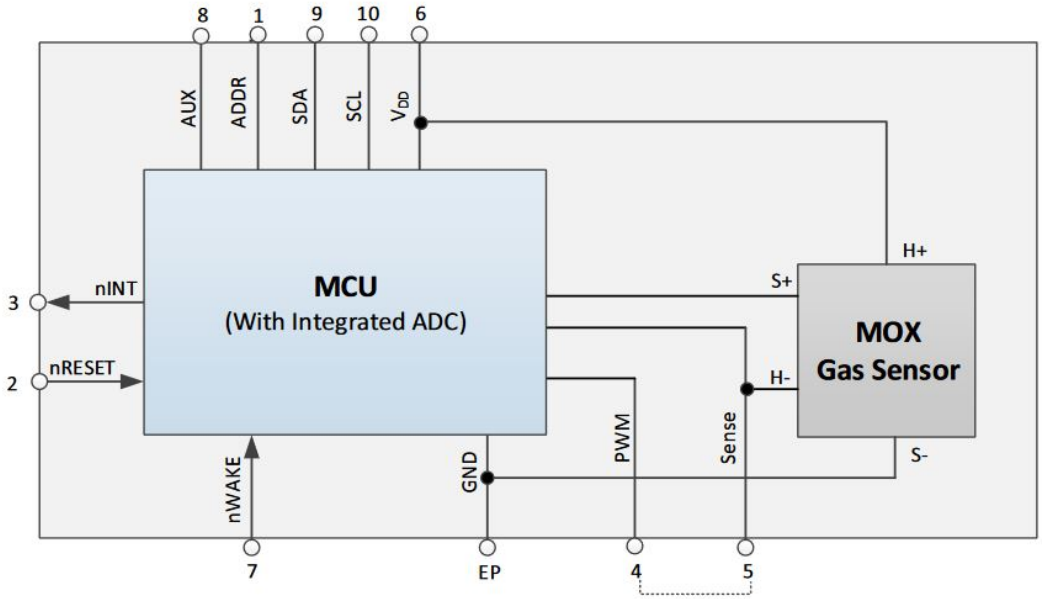
Wir haben den in Abbildung 5.2 zu sehenden CCS811-Sensor verwendet da er einige Vorteile mit sich bringt. Er ist ein digitaler Metalloxid (MOX)-Gassensor mit einem extrem geringen Stromverbrauch. Wie in Abbildung 5.3 zu sehen ist, besitzt er einen Analog-Digital (AD)-Wander und eine $I^2C$-Schnittstelle, was dem Entwickler eine leichte Soft- und Hardware Integration bietet. Zudem soll er eine Lebenszeit von über 5 Jahren nachweisen. [ams16, vgl. S. 1]




Quelle: eigene Aufnahme

**Abbildung 5.2.:** Verwendeter CCS811-MOX-Gassensor

Der CO2-Sensor ist über die Pins SDA und SCL mit dem Arduino verbunden. Pin 7 ist dabei auf Masse kurzgeschlossen. Die restlichen Pins werden in unserem Projekt nicht benötigt.




Quelle: [ams16, S. 3]

**Abbildung 5.3.:** CCS811 Blockdiagramm

Der Abruf von Messdaten auf dem $I^2C$ erfolgt über das Master-Slave Prinzip. Dabei nimmt der Arduino die Rolle des Masters, und der $CO_2$-Sensor die des Slaves ein. Das bedeutet, der CCS811 darf nur Informationen senden, wenn er vom Arduino die Aufforderung bekommen hat.
Um an die richtigen Daten des Sensors zu gelangen, muss der Arduino in unserem Projekt die ersten beiden Bytes auslesen, da diese die nötigen Informationen über den $CO_2$-Gehalt der Luft

zu Verfügung stellen. Die restlichen Bytes werden in unserem Projekt nicht ausgelesen, da wir für diese keine Verwendung haben.

| Byte 0 | Byte 1 | Byte 2 | Byte 3 |
|---|---|---|---|
| $eCO_2$ High Byte | $eCO_2$ Low Byte | TVOC High Byte | TVOC Low Byte |

| Byte 4 | Byte 5 | Byte 6 | Byte 7 |
|---|---|---|---|
| STATUS | ERROR_ID | See RAW_DATA | See RAW_DATA |

Quelle: [ams16, vgl. S. 14]

**Abbildung 5.4.:** Inhalt der 8-Byte-Übertragung des CCS811-MOX-Sensors

In der Softwareimplementierung wird der $Co_2$-Sensor mithilfe der Bibliothek <Ardafruit_CCS811.h>, wie in Abbildung 5.5 gezeigt wird, aufgerufen. Zunächst werden Setups für das $I^2C$-Interface und die Hardware durchgeführt. Danach wird geprüft, ob die Kommunikation mit dem Sensor aufgebaut werden kann. Wenn das der Fall ist, wird solange gewartet, bis der Sensor bereit ist, Daten zu lesen. Falls jedoch ein Problem auftritt, wird eine Fehlermeldung ausgegeben und der Arduino geht in eine sogenannte Endlosschleife, bis das Programm neu geladen oder der Arduino ausgeschalten wird.

```
// sensor controll
if(!CCS.begin()){
  Serial.println("Failed to start sensor! Please check your wiring.");
  while(1);
}
// wait for the sensor to be ready
while(!CCS.available());
```

**Abbildung 5.5.:** Codeausschnitt zur Sensor-Kontrolle im aus dem Quellcode

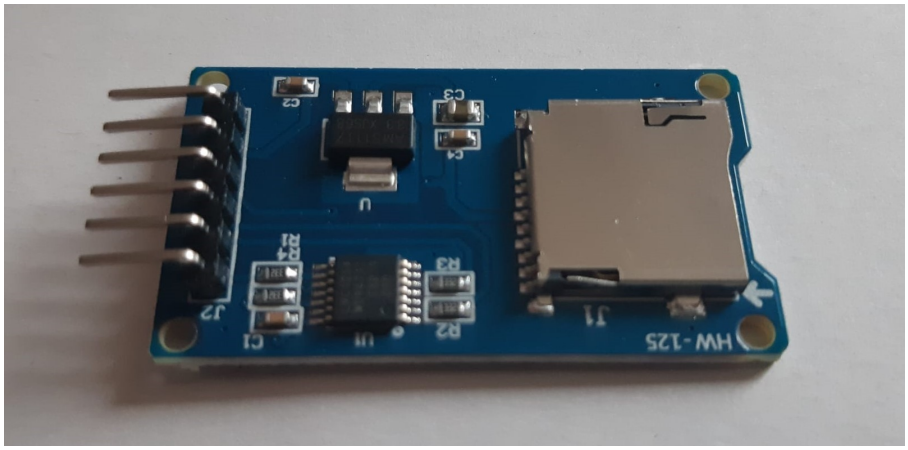
Die Abfrage der $CO_2$-Werten erfolgt über eine weitere Prüfung auf Fehler und einer darauffolgenden Messung. Das in Array dient hierbei zur Zwischenspeicherung der Daten. Anschließend werden die gespeicherten Informationen an die Mikro-SD-Karte weitergegeben. Näheres dazu in Kapitel 5.4.

```
// checks if data is available to be read
if(CCS.available()){
    // read and store the sensor data
    // data can be accessed with geteCO2
   if(!CCS.readData()){
     // measure co2 value
     measurement[i] = CCS.geteCO2();
   }
}
```

**Abbildung 5.6.:** Codeausschnitt für die Messung aus dem Quellcode

## 5.4. Mikro-SD

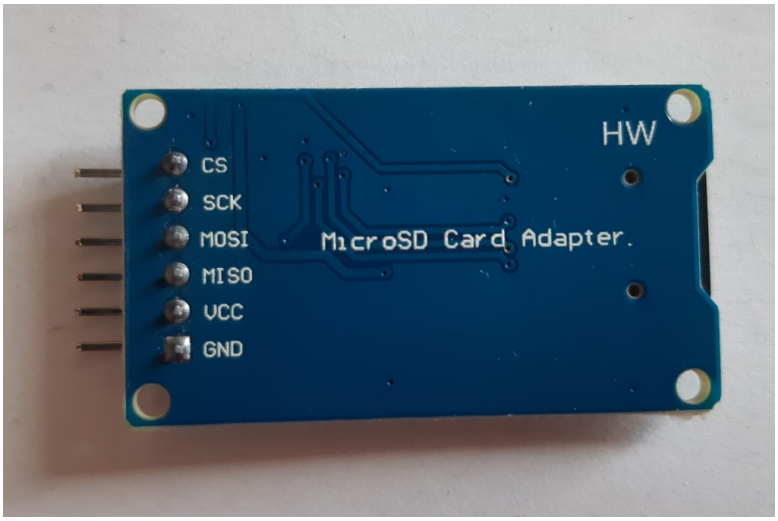
Zunächst sollten die gemessenen $CO_2$-Werte auf dem Arduino selbst abgespeichert werden. Der EEPROM wäre dafür die Lösung gewesen. Jedoch besitzt dieser eine Speicherkapazität von 1KB. Bei größeren Messungen reicht das unter Umständen nicht aus, sodass eine Methode gefunden werden musste, in welcher die Daten extern vom Arduino abgespeichert werden können. Dazu haben wir den, in Abbildung 5.7 zu sehenden Mikro-SD-Karten-Adapter hinzugezogen. Hier können die gemessenen $CO_2$-Werte ohne Speicherprobleme gesichert werden.




Quelle: eigene Aufnahme

**Abbildung 5.7.:** Verwendeter Mikro-SD-Karten-Adapter, Bild von der Vorderseite aufgenommen

Der Mikro-SD-Karten-Adapter ist über die, in Abbildung 5.8 zu sehenden PINs MISO, MOSI, SCK und CS, mit dem Arduino verbunden. Die komplementären PINs des Arduinos sind in Abbildung 4.3 zu sehen. Der Adapter benötigt zudem eine Versorgungsspannung von mindestens $4,5$V bis maximal $5,5$V zur Masse. [eba, vgl. S. 1]




Quelle: eigene Aufnahme

**Abbildung 5.8.:** Verwendeter Mikro-SD-Karten-Adapter, Bild von der Rückseite aufgenommen

Für die Integration der SD-Karte, muss zunächst die Bibliothek <SD.h> eingebunden werden. Anschließend wird eine globale Variable als File deklariert. In unserem Programmablauf wird die anschließende Initialisierung der SD-Karte zweimal durchgeführt. Das erste mal im Setup, vor dem Programmablauf. Die Begründung liegt darin, dass dem Benutzer bei fehlender SD-Karte schon vor der Auswahl des Messmodus eine Fehlermeldung angezeigt werden kann.

```
// sd-card controll
if (!SD.begin(5)) {
  // error-warning
  Serial.println("Initialisation failed!");
  lcd.print("Initialisation");
  lcd.setCursor(0, 1);
  lcd.print("of SD failed");
  delay(8000);
  return;
}
```

Quelle: eigene Aufnahme

**Abbildung 5.9.:** Codeausschnitt zur Initialisierung der SD-Karte im Setup aus dem Quellcode

# 6. Hardware

@Serkant: Bitte auch noch das CAD-Modell einbinden!

# 7. Testing

| Projekt: CO2-Sensor | Datum: |
|---|---|
| ID: CO201 | Version: 1.0 |
| Titel: Visualisierung der Luftqualität auf Basis von CO2-Grenzen | |
| Items: void ask(int) | TestKfg: 01 |
| Zielsetzung: Der Test soll zeigen, dass die Software die gemessenen CO2-Werte richtig interpretieren kann. | |
| Anforderungen: R01 | |
| Erforderliche Inputs zu Testbeginn: CO2 Werte | |

**Tabelle 7.1.:** Testbeschreibung Test 1

| Tester: | Beobachter: |
|---|---|
| Protokolldatei: | |
| Status: | Problembericht: |

**Tabelle 7.2.:** Testprotokoll Test 1

| Projekt: CO2-Sensor | Datum: |
|---|---|
| ID: CO202 | Version: 1.0 |
| Titel: Auswahl von verschiedenen Messprofilen | |
| Items: void ask(int) | TestKfg: 01 |
| Zielsetzung: Der Test soll zeigen, dass der Anwender zwischen drei verschiedenen Messprofilen wählen kann. | |
| Anforderungen: R03 | |
| Erforderliche Inputs zu Testbeginn: Anwender | |

**Tabelle 7.3.:** Testbeschreibung Test 2

| Tester: Alexander Herrmann | Beobachter: Johannes Ruffer |
|---|---|
| Protokoll:<br>Zunächst wurde der Arduino an den Laptop angeschlossen und somit das aktuelle Programm gestartet. Nach Auswahl des Schreibemodus, wurde das Messprofilmenü angezeigt. Nach drei UP-Button-Klicks, wurde der Messmodus 1 (Echtzeitmodus) mit dem Enter-Button bestätigt.<br>Die anschließende Messung wurde in Echtzeit auf dem LCD-Display ausgegeben. Die abgespeicherten Messungen wurden ebenfalls in Echtzeit gemessen. Anschließend wurde das Programm neu gestartet, um den Modus 2 auszuwählen. Auch dieser wurde erfolgreich mit den richtigen Zeitintervallen durchgeführt.<br>Der dritte Start des Programms diente dazu, die Tagesmessung auszuprobieren. Auch hier hat die Auswahl und Durchführung in Bezug auf die Testanforderungen erfolgreich funktioniert. | |
| Status: Erfolgreich | Problembericht: Nicht vorhanden |

**Tabelle 7.4.:** Testprotokoll Test 2

# 8. Handbuch

# 9. Installationsanleitung

Die folgenden Bibliotheken müssen heruntergeladen sein:

- Adafruit_CCS811.h (Arduino driver für den CCS811 Gas-Sensor)
- SPI.h (Ermöglicht die Kommunikation mit dem Serial Peripheral Interface)
- Wire.h (Ermöglicht die Kommunikation über $I^2C$)
- LiquidCrystal.h (Zur Integration des LCD-Displays)

In der folgenden Tabelle sind alle Verbindungen zwischen den einzelnen Komponenten mit dem Arduino aufgelistet. In 4.3 ist das enprechende Layout als Zeichnung zu finden.

| Nummer | Bauteil | Spezifikation | Anschlusspin Arduino |
|---|---|---|---|
| 1 | LCD-Display | RS | D7 |
| | | RW | GND |
| | | E | D6 |
| | | D4 | D5 |
| | | D5 | D4 |
| | | D6 | D3 |
| | | D7 | D2 |
| | | A | 5V |
| | | K | GND |
| 2 | LEDs | Rot | A3 |
| | | Gelb | A2 |
| | | Grün | A1 |
| | | Blau | A0 |
| 3 | Taster | Up-Button | 12 |
| | | Enter-Button | 13 |
| 4 | CCS811 | WAK | GND |
| | | SDA | SDA |
| | | SCL | SCL |
| | | VCC | 5V |
| 5 | MicroSD-Slot | GND | GND |
| | | MISO | D11 |
| | | MOSI | D10 |
| | | SCK | 9 |
| | | CS | 8 |
| | | VCC | 5V |

**Tabelle 9.1.:** Zuordnung der Pins

# 10. Fazit

# Verzeichnis verwendeter Abkürzungen und Formelzeichen

**DHBW** Duale Hochschule Baden-Würtemberg

**MOX** Metalloxid

**AD** Analog-Digital

**LCD** LiquidCrystal Display

**HDMI** High Definition Multimedia Interface

# Literaturverzeichnis


[ams16] AMS AG ; SPARKFUN (Hrsg.): *CCS811: Ultra-Low Power Digital Gas Sensor for Monitoring Indoor Air Quality*. https://cdn.sparkfun.com/assets/learn_tutorials/1/4/3/CCS811_Datasheet-DS000459.pdf. Version: 2016

[Dep] DEPARTMENT OF RESEARCH & DEVELOPMENT, DEPARTMENT OF INFORMATION TECHNOLOGIES AND SYSTEMS (Hrsg.): *A Data Quality Measurement Information Model Based On ISO/IEC15939*. https://s3.amazonaws.com/academia.edu.documents/36881777/ICIQ2007-ADataQualityMeasurementInfromationModelBasedOnISOIEC15939.pdf?response-content-disposition=inline%3B%20filename%3DA_Data_Quality_Measurement_Information_M.pdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWOWYYGZ2Y53UL3A%2F20200310%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20200310T081440Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=68ac8145994178d50baf90ab0a7557af4a341fa239938ed408aba2cd8a3c8e95

[eba] EBAYSEARCH (Hrsg.): *Micro SD Card Micro SDHC Mini TF Card Adapter Reader: Module for Arduino*. http://datalogger.pbworks.com/w/file/fetch/89507207/Datalogger%20-%20SD%20Memory%20Reader%20Datasheet.pdf

[Gen16] GENERATION ROBOTS (Hrsg.): *Wählen Sie das passende Arduino Board aus - Leitfaden: Die Qual der Wahl beim Arduino Board*. https://www.generationrobots.com/blog/de/auswahl-arduino-board/. Version: 26.09.2016

[Hel] HELGA MEYER, Heinz-Josef R.: *Projektmanagement: Von der Definition über die Projektplanung zum erfolgreichen Abschluss*. Bremen : Springer Gabler https://books.google.de/books?id=1ho3CwAAQBAJ&printsec=frontcover&dq=projektmanagement+von+der+definition+%C3%BCber+die+projektplanung+zum+erfolgreichen+abschluss&hl=de&

sa=X&ved=0ahUKEwi66ufKpY3oAhUF06YKHSFvB7wQ6AEIMjAB#v=onepage&q=projektmanagement%20von%20der%20definition%20%C3%BCber%20die%20projektplanung%20zum%20erfolgreichen%20abschluss&f=false. – ISBN 978–3–658–07569–9

[ION18] IONOS (Hrsg.): *Arduino vs. Raspberry PI: Mikrocontroller und Einplatinencomputer im Vergleich.* https://www.ionos.de/digitalguide/server/knowhow/arduino-vs-raspberry-pi/. Version: 2018

[Jay] Jay Abraham, Paul Jones, Raoul Jetley: *Formale Verifikationsmethoden für die Entwicklung von High-Integrity-Software für Medizinische Geräte.* http://files.vogel.de/vogelonline/vogelonline/files/3092.pdf

[Jos00] Joseph Schmuller: *Jetzt lerne ich UML: Der einfache Einstieg in die visuelle Objektmodellierung.* 2. Markt+Technik, 2000. – ISBN 3–8272–6591–6

[Mir18] Mirco Lang ; heise online (Hrsg.): *Was ist ein Arduino.* https://www.heise.de/tipps-tricks/Was-ist-Arduino-4035461.html. Version: 2018

[Pet09] Peter Liggesmeyer: *Software-Qualität: Testen, Analysieren und Verifizieren von Software: Prof. Dr.-Ing.* 2. Heidelberg : Spektrum Akademischer Verlag, 2009 https://link.springer.com/content/pdf/10.1007%2F978-3-8274-2203-3.pdf. – ISBN 978–3–8274–2056–5

[ser20] sertronics (Hrsg.): *Arduino Uno R3: Datenblatt.* https://www.berrybase.de/Pixelpdfdata/Articlepdf/id/1/onumber/A000066. Version: 19.03.2020

[Wik20a] Wikipedia (Hrsg.): *Softwaretest.* https://de.wikipedia.org/wiki/Softwaretest#Literatur. Version: 01.03.2020

[Wik20b] Wikipedia (Hrsg.): *Arduino (Plattform).* https://de.wikipedia.org/wiki/Arduino_(Plattform). Version: 2020

# Abbildungsverzeichnis

# Tabellenverzeichnis

# A. Anhang

## A.1. Weitere Abbildungen