

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:	30.04.2020
Bearbeitungszeitraum:	01.10.2019 - 30.04.2020
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 21. April 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 Projektabchluss 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.

Dieses Projekt befasst sich mit der Simulation eines Fensterhebers. Dabei wird das Öffnen und Schließen der Fenster mithilfe einer Light Emitting Diode (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 selbstständig vorgenommen.

Der folgende Bericht dokumentiert die Motivation, Vorgehensweise und Umsetzung von der Projektplanung, über die Implementierung von Hardware und Software, bis hin zu durchgeföhrten Tests und dem Projektabchluss.

Inhaltsverzeichnis

1. Einleitung	1
2. Anforderungen	3
3. Projektplan	7
3.1. Arbeitsplan	7
3.2. Aufgabenverteilung	8
3.3. Kostenplan	9
3.4. Cocomo	9
4. Entwurf	11
4.1. Architektur und Verhalten	11
4.1.1. Zustandsdiagramm	11
4.1.2. Top-Level-Architektur	13
4.1.3. Komponentendiagramm	14
4.2. Schaltungslayout	14
4.3. Zeichnung des Computer-Aided Designs	16
4.4. 3D-Druck	17
5. Hardwarekomponenten	19
5.1. Arduino vs. Raspberry PI	19
5.2. Arduino	20
5.3. CCS811	21
5.4. Mikro-SD-Karten-Adapter	23
5.5. I ² C-Adapter	25
6. Softwareimplementierung	27
7. Hardwareimplementierung	31
8. Testing	33

9. Handbuch	35
9.1. Erste Schritte	36
9.2. Durchführung einer Messung	36
9.3. Auslesen einer Messung	37
9.4. Sleepmode/Stromsparmodus	38
10. Installationsanleitung	39
11. Fazit	43
11.1. Software	43
11.2. Hardware	44
12. Ausblick	45
12.1. Software	45
12.2. Hardware	46
Verzeichnis verwendeter Abkürzungen und Formelzeichen	47
Literaturverzeichnis	49
Sachwortverzeichnis	51
Abbildungsverzeichnis	53
Tabellenverzeichnis	55
A. Anhang	57
A.1. Weitere Abbildungen	57

1. Einleitung

Kohlenstoffdioxid ist ein Spurengas, welches im Durchschnitt 0,04% der Luftzusammensetzung einnimmt, trotz dieses nur kleinen prozentualen Anteils wird es aufgrund der Bedeutung für unser Ökosystem zu den Hauptbestandteilen der Luft gezählt. [Wik, vgl.] Die übliche Angabe des Gehalts in der Luft erfolgt in ppm, so liegt der Durchschnittswert etwa bei 400ppm. Bis zu einem Wert von 800ppm spricht man von einer guten Luftqualität, von 1000-1400ppm gilt die Qualität als mäßig. [CiNS, vgl.]

Erste Einwirkungen auf den Menschen sind um 1300ppm zu vermerken, hier noch in erster Linie durch Konzentrationsschwächen und Schläfrigkeit. Gesundheitlich bedenklich wird es ab ca. 5500ppm. Auswirkungen auf die Atmung entstehen bei einer Konzentration von 30000ppm, was dem Kohlenstoffdioxidegehalt in einem Atemzug entspricht. [Umw17, vgl. S. 1364]

Im Alltag werden solch hohe Werte normalerweise jedoch nicht erreicht, solange in regelmäßigen Abständen gelüftet wird und man sich nicht unnötig in engen überfüllten Räumlichkeiten aufhält. In Büros zum Beispiel, in denen sich viele Menschen befinden ist ein Wert zwischen 5000 und 6000ppm nicht unüblich. [Umw17, vgl. S. 1364]

Stellt man sich nun ein Klassenzimmer voller Studierender vor, die den ganzen Tag ihren Vorlesungen folgen, ist schnell ersichtlich, dass ohne ausreichendes Lüften keine geeigneten Lehrbedingungen geschaffen werden können. Um dieses Problem anzugehen wurde in diesem Projekt ein CO₂-gesteuerter Fensterheber, mit Hilfe eines Arduinos, simuliert. Durch diesen soll es möglich sein optimale Bedingungen im Sinne der Luftqualität bieten zu können.

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]
- Inparity: 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]

2. Anforderungen

Daher wurde am Anfang des Projektes eine Tabelle erstellt, indem alle Anforderungen definiert wurden. Zudem sind diese nummeriert und haben individuelle Verifikationsmethoden zugeordnet 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 Abbildung 2.1 zu sehen ist, wurde mithilfe von Architect Enterprise ein sogenanntes Use-Case-Diagramm erstellt, welches den Zusammenhang zwischen Anwender und Programmfunctionen 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 Tabelle 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.

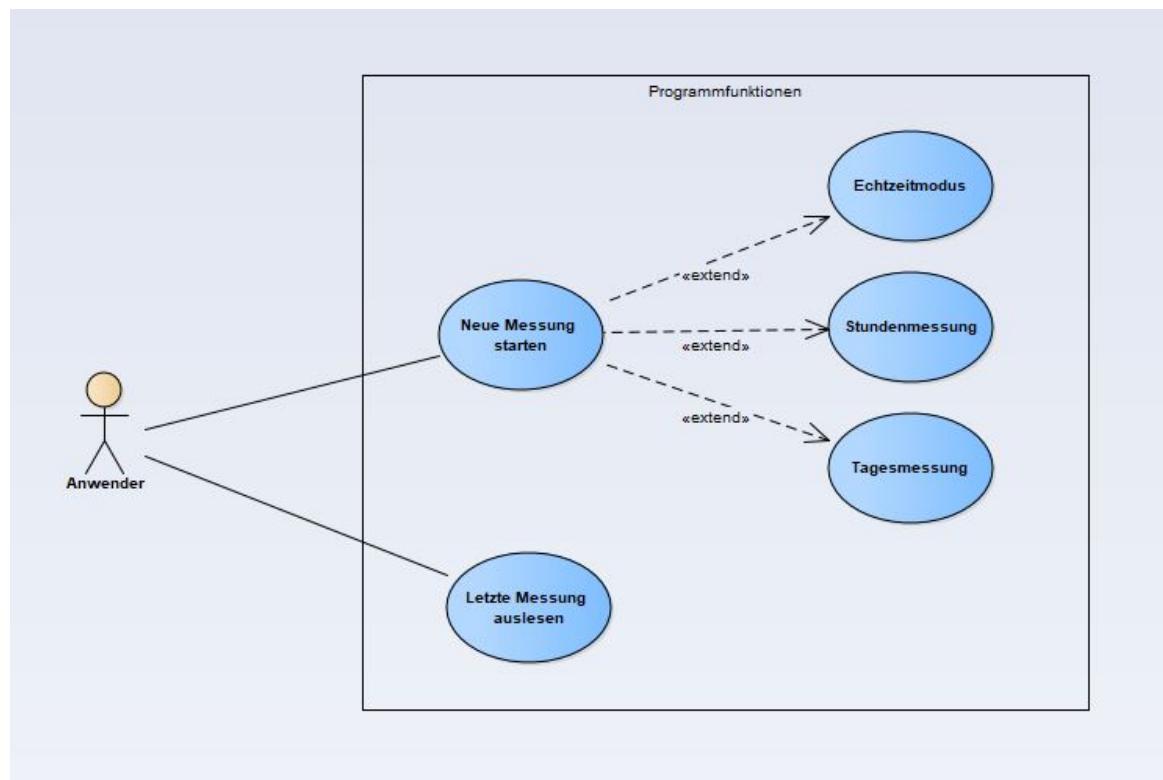


Abbildung 2.1.: Entwurf des Use-Case-Diagramms mithilfe von Enterprise Architect

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. Aufgabenverteilung

Damit jeder weiß, welchen Teil er zum Projekt beiträgt, wurden nach der Erstellung des Arbeitsplans, die Aufgaben innerhalb der Gruppe verteilt. In Abbildung 3.2 wird das Projekt auf die Teammitglieder unterteilt, sodass die Auflistung der Aufgaben darunter Platz finden kann. Bei Nicht-Einhaltung von Anforderungen und Zielen, können mithilfe dieser Liste, die Verantwortlichen herausgelesen werden. So ist es möglich, Gründe für das Scheitern leichter nachzuvollziehen, zu beheben und somit bei späteren Projekten vielleicht sogar zu verhindern.

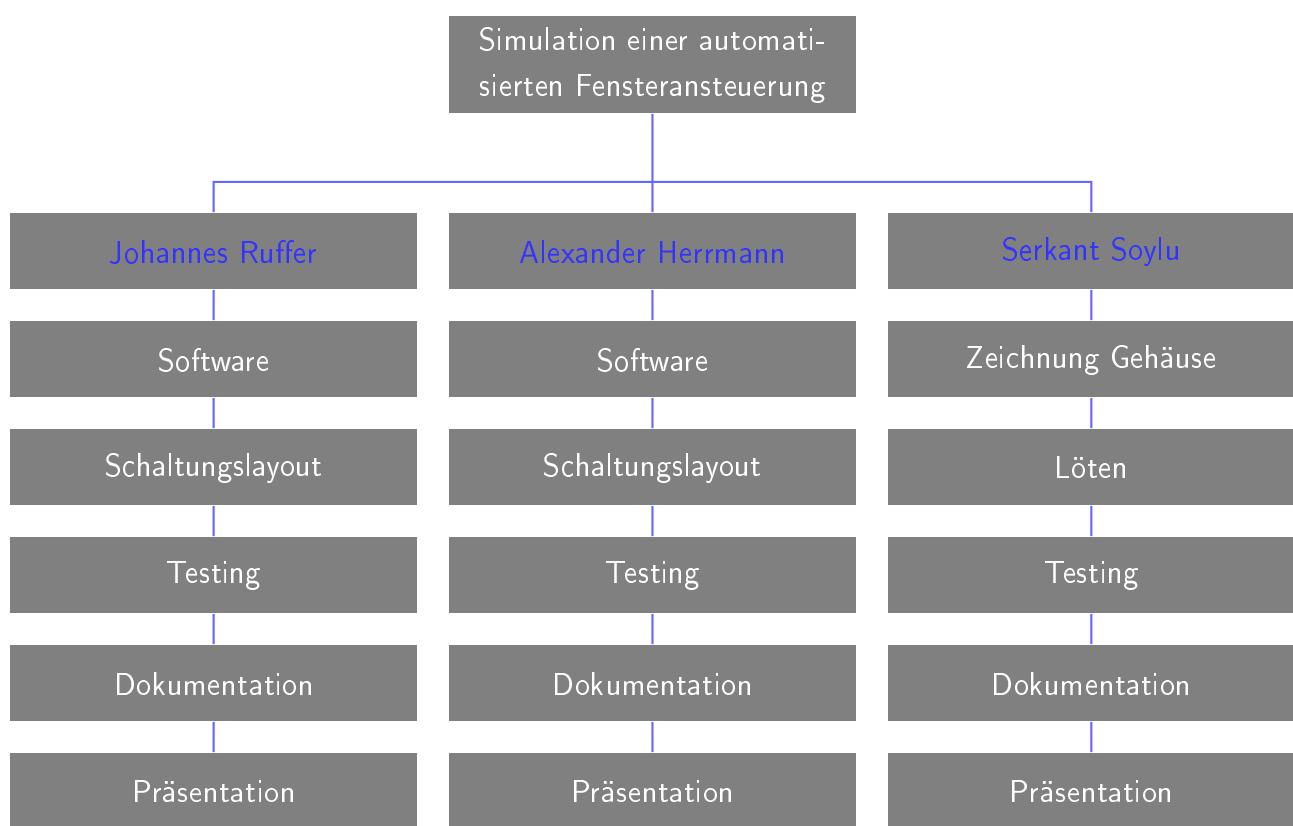


Abbildung 3.2.: Aufgabenverteilung

3.3. Kostenplan

Die Kosten des Projektes belaufen sich, wie in Tabelle 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 ² C-Adapter	3,99€
Mechanische Bauteile	24,99€
Sonstiges	4,56€
Gesamt	77,13€

Tabelle 3.1.: Aufgewendete Kosten für das Projekt

3.4. Cocomo

Cocomo ist eine Anwendung zur Aufwandseinschätzung in der Softwareentwicklung. Dahinter steckt ein mathematisches Modell, welches mit Hilfe einer Vielzahl von Parametern eine Kostenschätzung, sowohl in zeitlicher, als auch in monetärer Hinsicht, erstellt. Gerechnet wird in Personenmonaten oder Personenjahren, um diese möglichst genau angeben zu können sind die meisten gewünschten Parameter sehr firmen- und projektspezifisch. Die Basis für das Ergebnis der Einschätzung sind die im Projekt erwarteten Codezeilen, wobei man zwischen tatsächlich abgegebenem Code an den Kunden und dem gesamten Projekt inklusive der Testsoftware unterscheidet. Ist die Anzahl der benötigten Codezeilen bestimmt geht man über zur Komplexitäts einschätzung. Diese stellt drei Schwierigkeitsstufen zur Verfügung, organic mode, semi-detached und embedded. Dies sind nur beispielhafte oberflächliche Einstellungen, wie sie in diesem Projekt zur Aufwandseinschätzung verwendet wurden.

3. Projektplan

CO2Sensor - Component Structure Report					
SystemStar 3.0 Demo		April 18, 2020 21:32:00		Page: 1	
Estimate Name:	CO2Sensor		Estimate ID:		
Model Name:	COCOMO® II 2000		Model ID:	2000	
Process Model:	COCOMO® II Model		Phases:	Waterfall	
Level	Component Name	Developed Size	Effort in Person-Months	Cost (K\$)	Increment
1	Sourcecode	330	0.9	0.0	1
2	Measure	60	0.2	0.0	1
2	Loop	20	0.1	0.0	1
2	Write_SD	70	0.2	0.0	1
2	Setup	30	0.1	0.0	1
2	Settings	40	0.1	0.0	1
2	Something	30	0.1	0.0	1
2	Measurement_Mode	50	0.1	0.0	1
2	Read_Or_Write	30	0.1	0.0	1

Abbildung 3.3.: Lines of Code Schätzung

Zuerst wurden die Anzahl der Codezeilen für jede einzelne Funktion eingeschätzt, die später auch im Endprodukt stehen sollten. Dabei ergaben sich 330 Zeilen Programmcode, welche laut Abbildung 3.4 in 0,9 Personenmonaten abgearbeitet werden fertiggestellt sind. Die monetären Kosten hierfür belaufen sich auf 0€, da dies eine Projektarbeit ist und somit keine Bezahlung erfolgt.

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

Im folgenden werden drei Diagramme, welche wir im Laufe der Projektplanung anfertigten, beschrieben und erklärt. Diese sind die Grundsteine unseres Projektentwurfs.

Das Zustandsdiagramm 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 ein Zustandsdiagramm den Anfangs- und Endpunkt einer Sequenz von Zustandsänderungen. [Jos00, vgl. S. 120]

Die Top-Level-Architektur soll anschließend alle Ein- und Ausgänge der Hardware aufzeigen. Zu guter Letzt zeigt das Komponentendiagramm, wie der Name schon verrät, alle Komponenten, welche in Bezug auf das Projekt benötigt werden an. Hier sind sowohl Hard-, als auch Software-Komponenten mit den jeweiligen Verbindungen integriert.

4.1.1. Zustandsdiagramm

Damit der genaue Ablauf unseres Programms definiert ist, wurde mithilfe von Enterprise Architect das Zustandsdiagramm aus Abbildung 4.1 gezeichnet.

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, eindeutig zu erkennen sind.

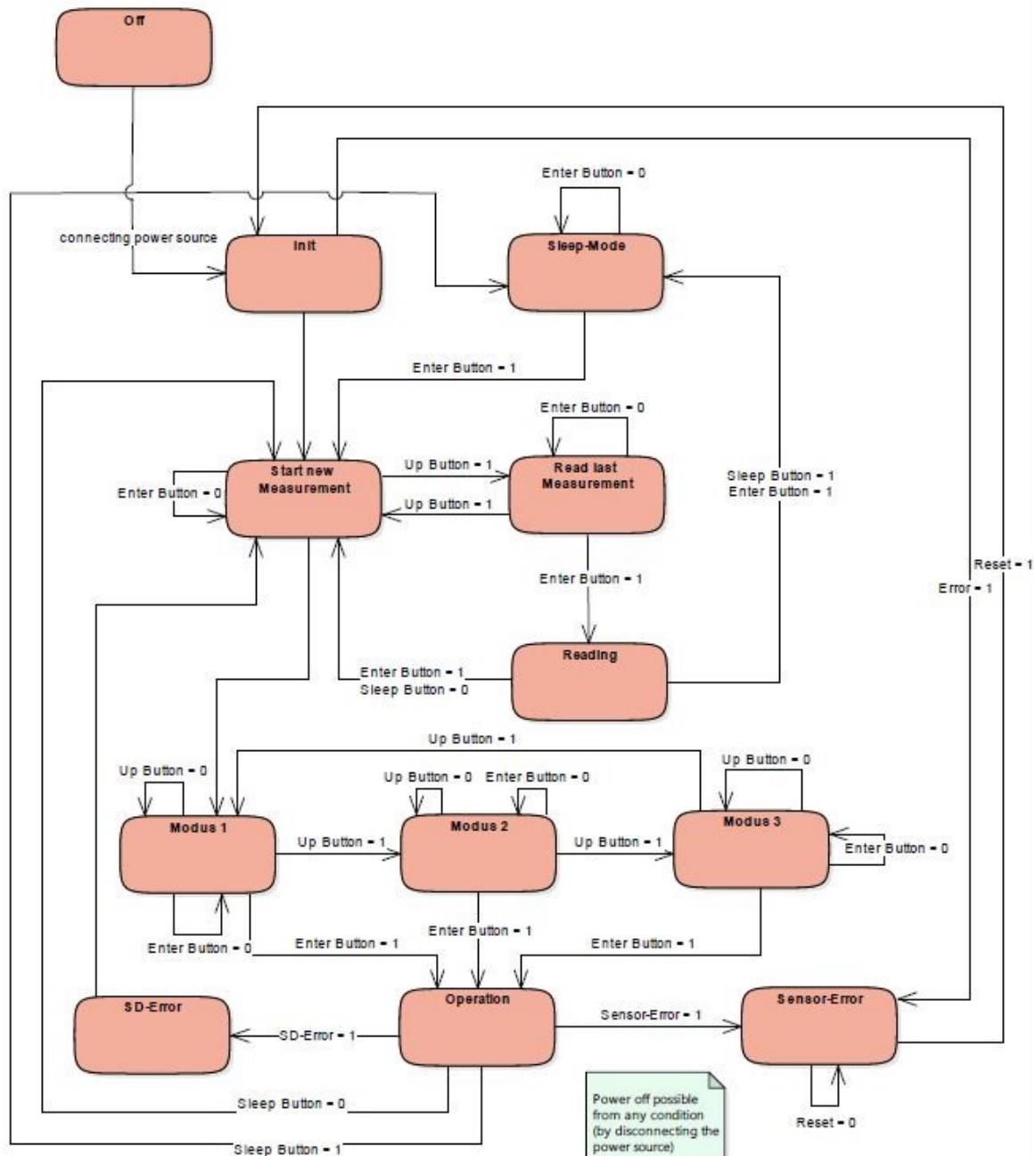


Abbildung 4.1.: Entwurf des Zustandsdiagramms mithilfe von Enterprise Architect

4.1.2. Top-Level-Architektur

Nachdem der genaue Programmablauf niedergeschrieben war, konnten durch die Zeichnung eines Top-Level-Diagramms, alle Ein- und Ausgänge am Arduino definiert werden.

Durch die drei Taster Up Button, Enter Button und Sleep Button, sollte die Bedienung des Menüs ermöglicht werden. Der CO₂-Value in Abbildung 4.2 steht für den, vom Sensor gemessenen Wert, welcher abgespeichert und ausgegeben wird.

Wegen der anfangs definierten Anforderung, die gemessenen CO₂-Werte in einer CSV-kompatiblen Datei auf einer SD-Karte abzuspeichern, muss die Hardware einen Ausgang für die Text-Datei besitzen. Zudem benötigt diese einen weiteren Port für die Ausgabe am LCD-Display und vier weitere für die drei LEDs zur Interpretation der Luftgüte und die, zur Simulation des Fensterscheibenmotors.

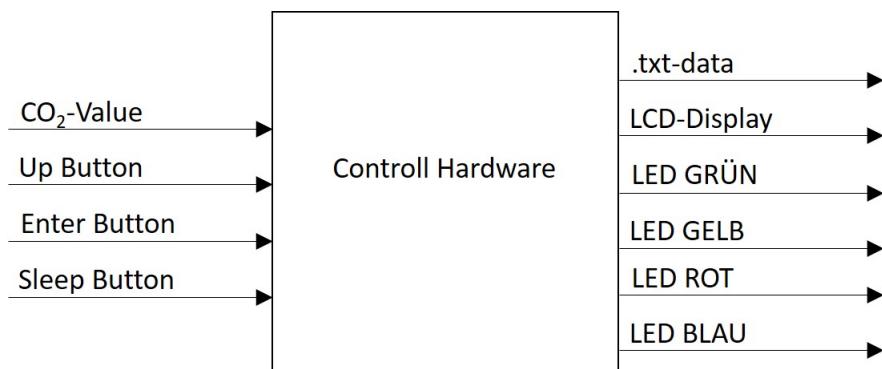


Abbildung 4.2.: Entwurf der Top-Level-Architektur

4.1.3. Komponentendiagramm

Das Komponentendiagramm, welches in 4.3 zu sehen ist, soll dazu dienen, alle Akteure und deren Verbindungen darzustellen.

So teilten wir den Arduino als Controll-Hardware in drei Komponenten. Die Profilauswahl importiert drei Eingangssignale, welche bei Betätigung der Taster, gesendet werden. Je nach Menü sind diese Signale unterschiedlich zu interpretieren. Zudem werden die übersetzten Botschaften, je nach Fall, an das LCD-Display oder die Messeinrichtung weitergeleitet.

Der CO₂-Sensor sendet ausschließlich nach Aufforderung der Messeinrichtung den gemessenen CO₂-Wert, welcher an die Mikro-SD-Karte und die CO₂-Auswertung weitergeleitet wird. In der Mikro-SD werden die Daten abgespeichert, während die Auswertung der Messwerte die unterschiedlichen LEDs ansteuert.

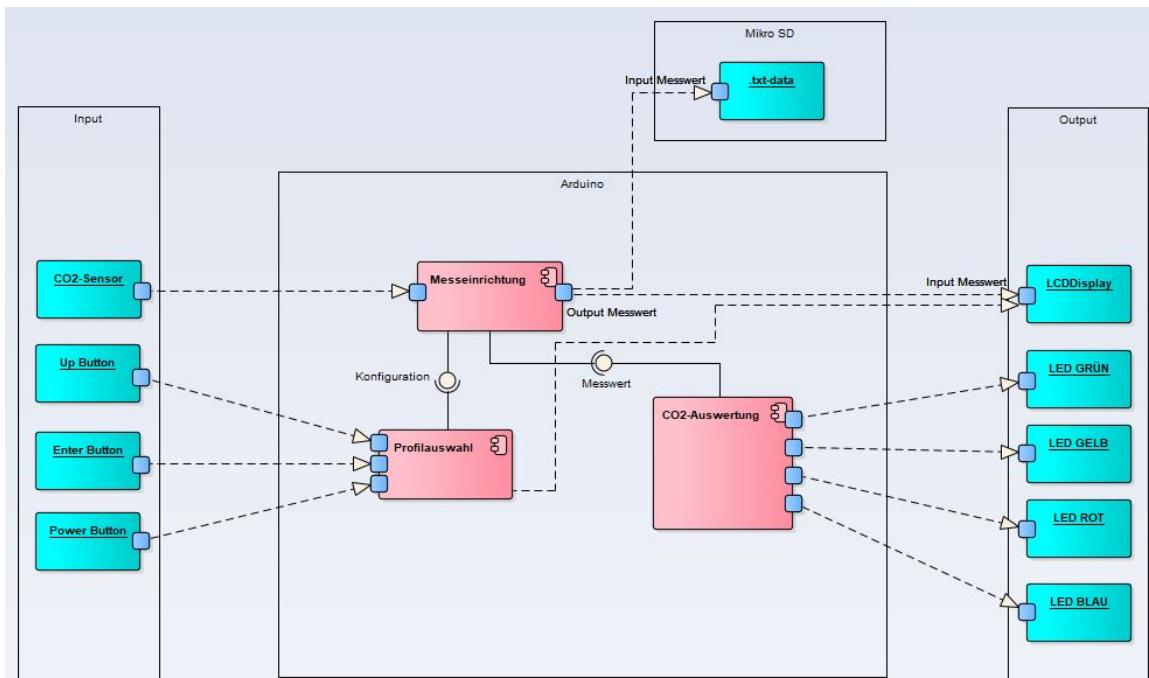


Abbildung 4.3.: Entwurf des Komponentendiagramms mithilfe von Enterprise Architect

4.2. Schaltungslayout

In Abbildung 4.4 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 4.4 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 I²C-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 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.

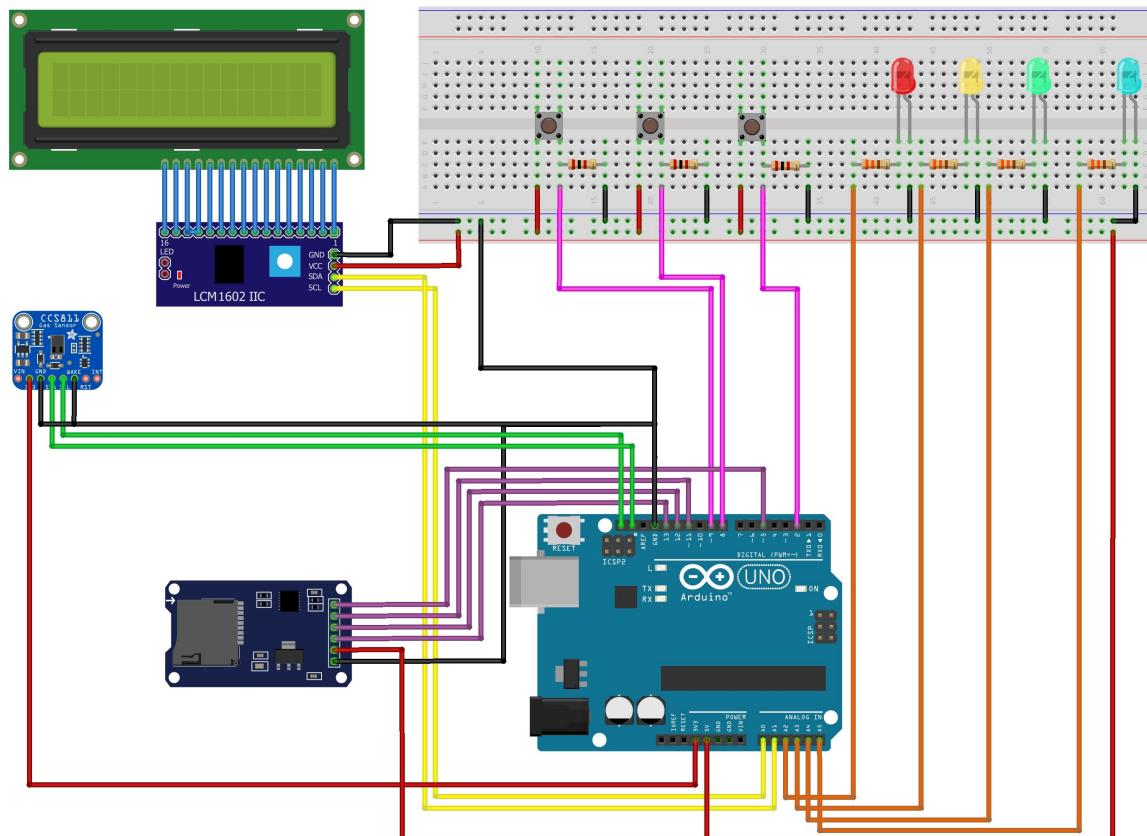


Abbildung 4.4.: Entwurf des Schaltungslayouts mithilfe von Fritzing

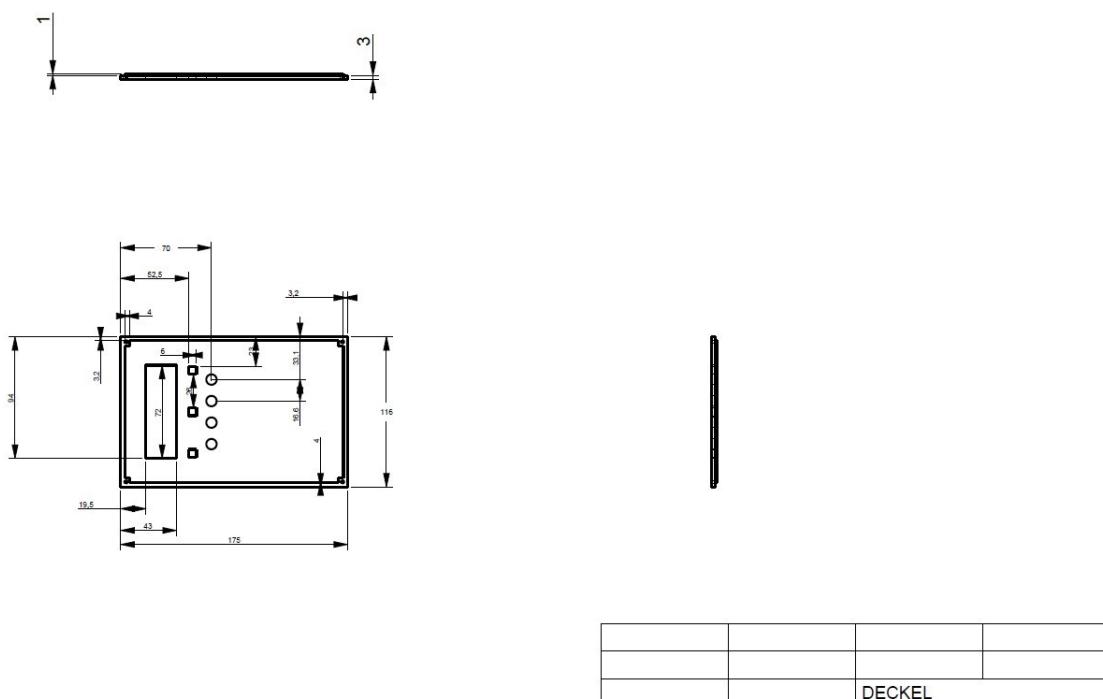
4.3. Zeichnung des Computer-Aided Designs

Wie schon im Kapitel 4 beschrieben wurde, war es für uns als Gruppe von größter Wichtigkeit, unsere gesammelten Konzepte in die Gehäuseentwicklung einfließen zu lassen, um so, die ideale Bauform für unser Messgerät zu entwickeln und zu fertigen.

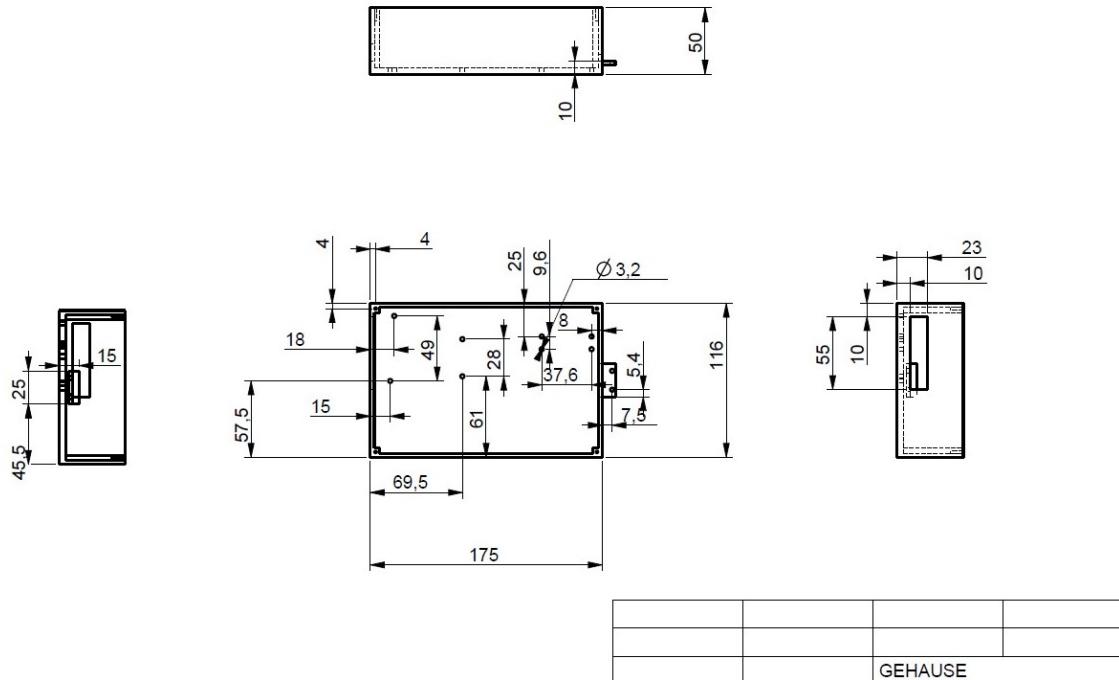
Für die Erstellung der Zeichnung in einem Computer-Aided Design (CAD)-Programm haben einerseits die einfache Anpassungsfähigkeit, die drei Dimensionale Visualisierung des Gehäuses und die Zeichnungsdatei als Quelldatei für den 3D-Druck gesprochen.

Das Gehäuse wurde in zwei Teile aufgeteilt. Zum einen der Deckel, aus Abbildung 4.5, in welchem das LCD-Display, die einzelnen Taster für die Auswahl der Menüpunkte und die LEDs für die Visualisierung der Luftqualität integriert sind.

Zum anderen das Gehäuse, welches in Abbildung 4.6 dargestellt wird. Hier ist das Arduino Modul, das PIN-Breadboard, der Mikro-SD-Karten-Adapter und der CO₂-Sensor platziert. An dieser Stelle war es wichtig, dass der Sensor außerhalb des Gehäuses platziert wird, damit die Richtigkeit der resultierenden Messergebnisse sichergestellt ist. Der Arduino selbst wurde am Gehäuserand platziert, damit ein einfacher Zugang zu den externen Ausgängen gewährleistet ist. So muss nicht bei jeder neuen Softwareversion das gesamte Gehäuse geöffnet werden.



Quelle: eigene Zeichnung

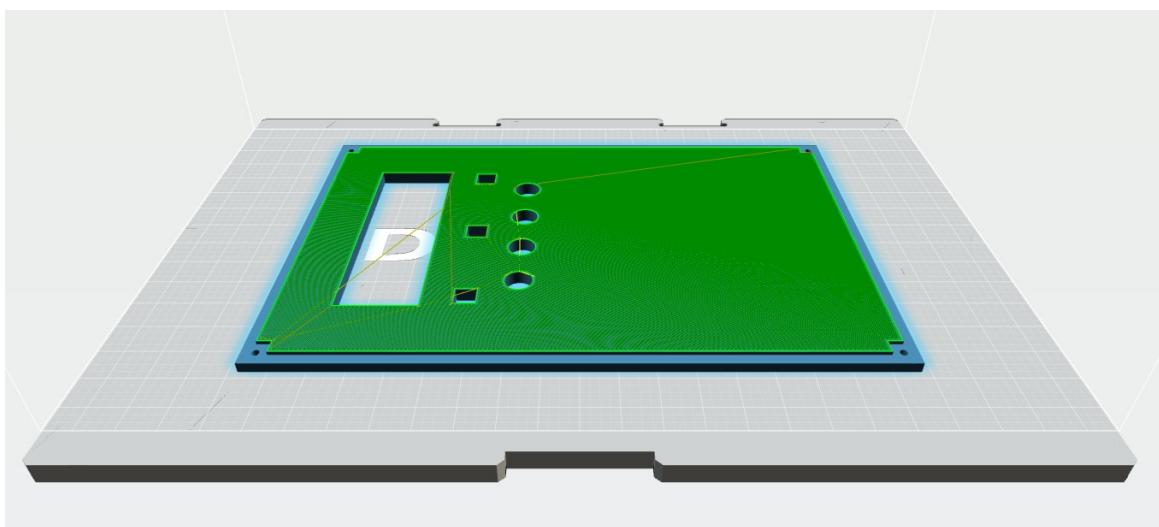


Quelle: eigene Zeichnung

Abbildung 4.6.: CAD-Zeichnung für die Schale des Gehäuses

4.4. 3D-Druck

Wir haben uns für den 3D-Druck entschieden, da uns einerseits der freie Zugang zu einer mechanischen Werkstatt gefehlt hat, andererseits der 3D-Druck uns die Möglichkeit gibt, ein absolut individuelles Gehäuse nach unseren Vorstellungen zu fertigen. Aus der Quelldatei vom CAD-Programm wird mithilfe des Drucker-Eigenen Programms eine STL-Datei erstellt, in welcher die Sehnenhöhe und andere benötigten Daten, welche zum erfolgreichen Drucken des Gehäuses notwendig sind, definiert werden. Nach erfolgreicher Umwandlung wird der Druckprozess simuliert, um feststellen zu können, ob die Zeichnung so angefertigt kann, wie sie vorher gezeichnet wurde. In Abbildung 4.7 ist eine solche Simulation zu sehen, wie sie das Drucker-Eigene Programm ausgibt.



Quelle: eigene Aufnahme

Abbildung 4.7.: Simulation der Standard Template Library (STL)-Datei

5. Hardwarekomponenten

Damit das Projekt den Anforderungen entsprechend umgesetzt werden kann, wurden Hardwarekomponenten ausgewählt. Im folgenden Kapitel sind Komponenten verglichen und analysiert worden, um die bestmögliche Umsetzung zu ermöglichen.

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 dieser 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 Raspberry PI für das Projekt zu nehmen.

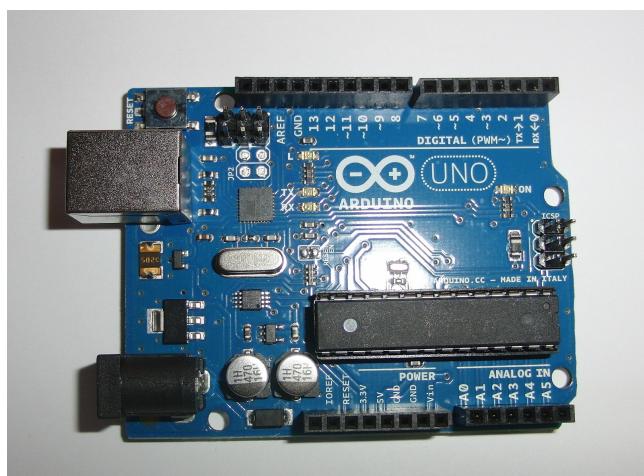
5.2. Arduino

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 Abbildung 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 Abbildung 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.

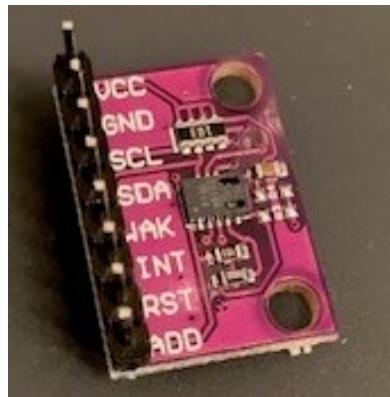


Quelle: <https://pixabay.com/de/photos/arduino-computer-cpu-373994/>

Abbildung 5.1.: Arduino Uno

5.3. CCS811

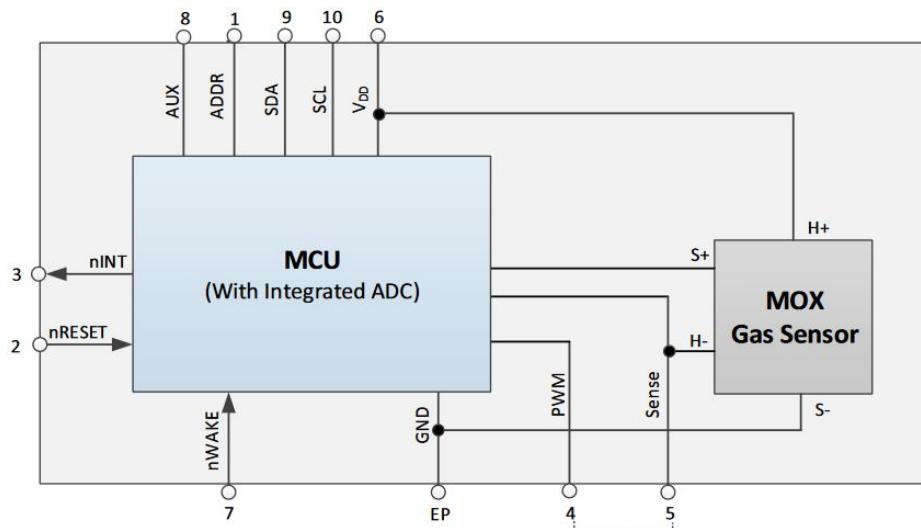
Wir haben den in Abbildung 5.2 zu sehenden CCS811-Sensor verwendet da er einige Vorteile mit sich bringt. Er ist ein digitaler Metalloxid (MOX)-Gassensor mit einem extrem geringen Stromverbrauch. Wie in Abbildung 5.3 zu sehen ist, besitzt er einen Analog-Digital (AD)-Wandler und eine I²C-Schnittstelle, 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.: Komponentendiagramm des CCS811

Der Abruf von Messdaten auf dem I²C erfolgt über das Master-Slave Prinzip. Dabei nimmt der Arduino die Rolle des Masters, und der CO₂-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 CO₂-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 CO₂-Sensor mithilfe der Bibliothek <Ardafruit_CCS811.h>, wie in Abbildung 5.5 gezeigt wird, aufgerufen. Zunächst werden Setups für das I²C-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 control
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 CO₂-Werte erfolgt über eine weitere Prüfung auf Fehler und einer darauffolgenden Messung. Das Array dient hierbei zur Zwischenspeicherung der Daten. Anschließend

werden die gespeicherten Informationen an die Mikro-SD-Karte weitergegeben. Näheres dazu in Kapitel 5.4.

```

while(i < NUMB_MEASURE){
    // checks if co2-sensor is available
    if(CCS.available()){
        if(!CCS.readData()){

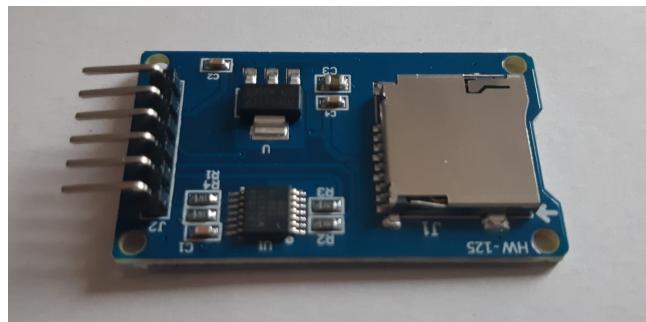
            // measure co2 value
            var_measure = CCS.geteCO2();
    }
}

```

Abbildung 5.6.: Codeausschnitt für die Messung aus dem Quellcode

5.4. Mikro-SD-Karten-Adapter

Zunächst sollten die gemessenen CO₂-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 CO₂-Werte ohne Speicherprobleme gesichert werden.



Quelle: eigene Aufnahme

Abbildung 5.7.: Verwendeter Mikro-SD-Karten-Adapter, mit Sicht auf dessen Vorderseite

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 4.4 zu sehen. Der Adapter benötigt zudem eine Versorgungsspannung von mindestens 4,5V bis maximal 5,5V zur Masse. [eba, vgl. S. 1]

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



Quelle: eigene Aufnahme

Abbildung 5.8.: Verwendeter Mikro-SD-Karten-Adapter, mit Sicht auf dessen Rückseite

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. Das zweite Mal wird die SD-Karte vor dem Start der Messung durchgeführt. Das hat den Hintergrund, dass die SD-Karte kurz davor entfernt werden könnte. Auch hier wird bei fehlender Karte der Benutzer durch eine Ausgabe informiert. Zudem wird das Programm anschließend neu gestartet. Vor dem Start der Messung wird außerdem eine .txt-Datei auf der SD-Karte erstellt. Im Fall, dass eine gleichnamige Textdatei bereits existiert, wird diese zum schreiben geöffnet. An dieser Stelle wird im Programmcode auch definiert, dass der Arduino die SD-Karte beschreiben und nicht auslesen soll.

Während der Messung bleibt die SD-Karte geöffnet und der Arduino im Schreibe-Modus, sodass kein lokaler Speicher für die Übertragung der Daten benötigt wird. Somit können so viele Messungen durchgeführt werden, wie Speicherkapazität auf der SD-Karte vorhanden ist. Nach Beenden der Messung wird die Textdatei abgespeichert und geschlossen.

```
// sd-card control
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;
}
```

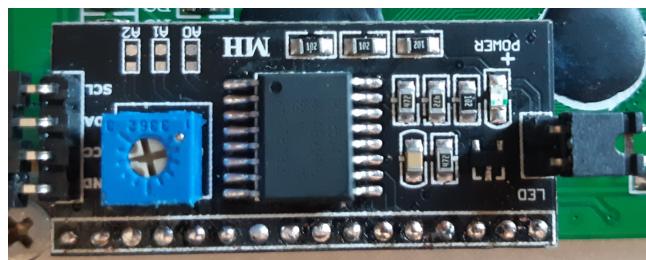
Quelle: eigene Aufnahme

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

5.5. I²C-Adapter

Aufgrund der hohen PIN-Belegung des LCD-Displays und den damit fehlenden digitalen Steckmöglichkeiten am Arduino, haben wir uns dazu entschieden, den in Abbildung 5.10 zu sehenden I²C-Adapter, zu verwenden.

Mit diesem Adapter ist es nun möglich, sechs digitale PINs, welche wir vorher hätten verwenden müssen, auf zwei analoge PINs zu reduzieren. Somit reichen die digitalen PINs am Arduino für drei Taster, einen CO₂-Sensor und eine Mikro-SD-Karten Adapter auf jeden Fall aus.



Quelle: eigene Aufnahme

Abbildung 5.10.: Verwendeter I²C-Adapter

Die Implementierung des I²C-Adapters erfolgt über die Bibliothek <LiquidCrystal_I2C.h>. In dieser können die gleichen Befehle aufgerufen werden, welche schon in der Bibliothek <LiquidCrystal.h> genutzt werden konnten. Die restlichen Befehle funktionieren mit beiden Bibliotheken. Somit muss bei einer späteren Umrüstung nur wenig im Quellcode geändert werden. Die einzige Änderung muss bei der Adressvergabe und Größenangabe des LCD-Displays vorgenommen werden.

So konnten die Adressen am Arduino vorher durch das aneinanderreihen der digitalen PINs im Quellcode definiert werden. Zudem gab es im Setup eine weitere Zeile, in welcher die Anzahl der Zeichen und Reihen des Displays angegeben werden musste.

In Abbildung 5.11 ist der Codeausschnitt zu sehen, welcher zum einen die Adressen definiert, zum anderen aber auch angibt, wie viele Zeichen und Reihen verwendet werden sollen.

Durch drehen des Rades am grau-blauen Aufsatz, welcher in Abbildung 5.10 links im Bild zu sehen ist, wird der Kontrast eingestellt. Ein Vorwiderstand muss somit nicht händisch zwischen Arduino und LCD-Display geschaltet werden.

```
// set the lcd address to 0x27 for a 16 chars and 2 line display
LiquidCrystal_I2C lcd(0x27, 20, 4);
```

Quelle: eigene Aufnahme

Abbildung 5.11.: Codeausschnitt zur Definition von Adresse und Anzahl der zu verwendenden Zeichen und Reihen

6. Softwareimplementierung

Für die entgültige Struktur der Software waren mehrere Anforderungen ausschlaggebend. Zunächst wird beim Anschalten des Arduinos das Setup durchlaufen. Hier wird das LCD-Display, angeschaltet und initialisiert. Dazu muss die Bibliothek <LiquidCrystal.h> eingebunden werden. Auch die Ein- und Ausgänge am Arduino wie Taster und LEDs werden hier definiert. Zudem geprüft, ob mit dem Mikro-SD-Karten-Adapter und CO₂-Sensor kommuniziert werden kann.

Falls der CO₂-Sensor nicht bereit sein sollte, wird eine Fehlermeldung ausgegeben. Nach erfolgreichem Start ist der Arduino angehalten so lange mit dem Starten des Programms zu warten, bis der Sensor zurückmeldet, dass er bereit ist, die Messung zu beginnen.

Aufgrund der Tatsache, dass wir den Sleep-Mode über eine Interrupt-Funktion erreichen, wird an dieser Stelle, mithilfe des Befehls <attachInterrupt()>, wie in Abbildung 6.1 zu sehen ist, die Funktion deklariert.

Anschließend befindet sich der Arduino softwaretechnisch im Loop, einer Endlosschleife. Hier

```
// integration of interrupt command  
attachInterrupt(0, sleepmodeInterrupt, CHANGE);  
}
```

Quelle: eigene Aufnahme

Abbildung 6.1.: Codeausschnitt aus dem Setup zur Deklaration der Interrupt-Funktion

werden zu Beginn jedes Durchlaufs alle LEDs ausgeschaltet. Ebenfalls wird der Cursor des LCD-Displays auf die erste Stelle zurückgesetzt.

Zu Beginn des Programmablaufs wird der Anwender gefragt, ob er eine neue Messung starten, oder lieber die Letzte auslesen möchte. Ist das Auslesen der letzten Messung gewünscht, erleuchten die LEDs grün, gelb und rot. Dies soll dem Anwender die Bestätigung geben, dass die letzte Messung ausgelesen werden kann. Softwaretechnisch passiert an dieser Stelle nichts weiter, als dass der Arduino auf die Betätigung des Enter-Buttons wartet. Diese Auswahl hat den alleinigen Zweck, dass der Benutzer die Mikro-SD-Karte nicht während einer Messung entfernt und somit einen Fehler provoziert.

Falls der Anwender dennoch gegen die Vorgaben der Entwickler handeln sollte und die Mikro-

SD-Karte während einer Messung herausnimmt, wird diese trotzdem bis zum Ende durchgeführt. Die resultierende Konsequenz ist, dass sich auf der Mikro-SD-Karte keine Datei befindet, da der Mikrocomputer nicht die Möglichkeit hatte die angelegte Text-Datei abzuspeichern. Handelt der Anwender im Sinne der Entwickler und wählt zum Entfernen der Mikro-SD-Karte den Lese-Modus, kann er nachdem diese wieder im Adapter ihren Platz gefunden hat, den Vorgang mit dem Enter-Button bestätigen. So öffnet sich wieder das Hauptmenü und eine neue Messung kann auf Wunsch gestartet werden.

Als nächstes wurden drei verschiedene Messprofile definiert und implementiert, sodass der Benutzer zwischen einer Echtzeit-, Stunden- und Tagesmessung wählen kann. In diesen 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. Diese Messmodi sind global mithilfe des Befehls `<# define>` festgelegt.

Nach der Wahl des Messprofils muss der Arduino den CO₂-Sensor ansteuern und richtig konfigurieren. Es muss getestet werden, ob er funktionstüchtig und bereit ist, eine Messung zu starten. Dies konnte durch eine if-Bedingung abgefragt werden.

Zudem kann der verwendete Sensor nicht nur CO₂-Werte, sondern beispielsweise auch Temperaturen messen, sodass der Arduino die richtigen Werte anfordern muss. Damit dies möglich ist, musste die Bibliothek `<Adafruit_CCS811.h>` eingebunden werden. Genaueres zu dem Algorithmus dieser Bibliothek wird in Kapitel 5.3 erläutert.

Auch während dem Programmdurchlauf wird bei jeder neuen Messung kontrolliert, ob der CO₂-Sensor funktionstüchtig ist. Danach wird mithilfe der oben genannten Bibliothek der CO₂-Wert gemessen und an den Arduino weitergegeben.

Nach Einlesen der Daten, vergleicht der Mikrocomputer diese mit den gegebenen Grenzwerten. Je nach Bewertung des gemessenen Wertes wird eine der grün, gelb oder roten LEDs eingeschaltet. Auch die blaue LED, welche die Ansteuerung des automatisierten Fensterscheibenmotors simulieren soll, wird je nach Messwert an- oder ausgeschaltet. Zudem werden dem Anwender die jeweiligen Daten im LCD-Display ausgegeben.

Damit der Verlauf der Messung später auf Excel geplottet werden kann, wird der gemessene Wert im .csv-Format auf einer Mikro-SD-Karte als .txt-Datei abgespeichert. Das bedeutet, dass die Messungen in einer Zeile, getrennt durch Kommas, gesichert werden. Dies wird durch den Befehl `<textfile.print()>` ermöglicht.

Falls Excel trotzdem alle Werte in ein Feld schreiben sollte, anstatt jeden Wert in einem solchen Feld anzuzeigen, kann dieses Problem innerhalb von Excel berichtet werden. In der Menüleiste von Excel muss das Feld `<Daten>` ausgewählt werden. Durch die Markierung des Eintrags und Auswählen des Feldes `<Text in Spalten>`, können die Trennzeichen händisch definiert werden. Nachdem Kommas als Trennzeichen ausgewählt wurden, teilen sich die Messungen in Spalten auf. Nun kann, durch Markierung aller Messwerte und der Auswahl des gewünschten Diagramms, der Verlauf der Messung dargestellt werden.

Nachdem der letzte Wert auf der Mikro-SD-Karte gespeichert wurde, beginnt das Programm durch die Anzeige des Hauptmenüs, automatisch von vorne. Falls der Sleep-Mode-Button während des Programmablaufs gedrückt wurde, springt das Programm in einen Art Stromspar- oder auch Schlafmodus. Hier gehen sowohl alle Lampen, als auch die Beleuchtung des LCD-Displays aus. Wie am Anfang des Kapitels erwähnt, wurde dieser Taster als Interrupt implementiert. Es muss somit nicht ein weiteres Menü kreiert werden, um den Anwender zu fragen, ob dieser den Sleep-Mode aktivieren möchte. Das ermöglicht dem Anwender, den Sleep-Mode-Button zu jedem beliebigen Zeitpunkt zu betätigen. Dem Benutzer ist garantiert, dass der Arduino, nach Beenden der aktuellen Messung, in den Schlafmodus geht.

Das Wieder-Aufwecken der Hardware geschieht durch eine Betätigung des Enter-Buttons. Das Betätigen eines anderen Tasters wird ignoriert. Auch das wiederholte Drücken des Sleep-Mode-Buttons soll nicht von Relevanz sein, damit der Anwender später den Enter-Button nicht öfter drücken muss, da der Schlafmodus mehrere Ebenen kreiert.

Damit dies gewährleistet ist, setzt die Interrupt-Funktion, welche in Abbildung 6.2 zu sehen ist, eine Variable auf den Wert Eins.

```
// interrupt-function to read SLEEP_BUTTON at any time
// interrupt-function is activated when value of sleep-status change
void sleepmodeInterrupt() {
    sleep_status = digitalRead(SLEEP_BUTTON);
    // reads if SLEEP_BUTTON got pressed
    //sleep_status = digitalRead(SLEEP_BUTTON);
    if(digitalRead(SLEEP_BUTTON) == HIGH){
        // delay and second read with if condition to debounce SLEEP_BUTTON
        delay(5);
        //sleep_status = digitalRead(SLEEP_BUTTON);
        if(digitalRead(SLEEP_BUTTON) == HIGH){
            sleeping = 1;
        }
    }
    delay(500);
}
```

Quelle: eigene Aufnahme

Abbildung 6.2.: Codeausschnitt: ausgelagerte Interrupt-Funktion mit Tastenentprellung, für das Werten der Variable, zur späteren Aktivierung des Schlafmodus

Solange die Variable den Wert Eins beibehält, bleibt der Arduino in seinem Schlafmodus. Bei Betätigung des Enter-Buttons wird diese Variable wieder auf Null zurückgesetzt und das Programm kann fortgeführt werden. Die Softwareimplementierung ist in Abbildung 6.3 zu sehen. Wird der Sleep-Button also öfter gedrückt wird der Wert dieser Variablen immer wieder auf Eins gesetzt, was den Zustand des Arduinos nicht verändert.

```
// while sleep-mode is activated
while(sleeping == 1){
    lcd.clear();
    // lcd backlight turns off
    lcd.noBacklight();
    delay(200);
    // reading ENTER_BUTTON to exit the sleep-mode
    if(digitalRead(ENTER_BUTTON) == HIGH){
        // delay of 5 milisec with second condition of ENTER_BUTTON
        // to debounce the button
        delay(5);
        if(digitalRead(ENTER_BUTTON) == HIGH){
            sleeping = 0;
            // lcd backlight turns on
            lcd.backlight();
        }
    }
}
```

Quelle: eigene Aufnahme

Abbildung 6.3.: Codeausschnitt aus dem Loop zur Aktivierung und Deaktivierung des Schlafmodus

Nach Verlassen des Sleep-Modes, durch die Betätigung des Enter-Buttons, erscheint wieder das Hauptmenü. Das Programm kann nun erneut von vorne gestartet werden.

7. Hardwareimplementierung

Nach erfolgreichem Drucken des Gehäuses, wurde das Arduino Modul mithilfe von Schrauben an die dafür vorgesehene Stelle befestigt, das PIN-Breadboard verklebt und der Sensor außerhalb des Gehäuses montiert. Das LCD-Display, die LEDs und die Taster wurden anschließend, wie in Abbildung 7.1 zu sehen ist, auf dem Deckel befestigt.



Quelle: eigene Aufnahme

Abbildung 7.1.: 3D-gedruckter Deckel des Gehäuses

Die Entscheidung, das PIN-Breadboard für die Implementierung, wie in Abbildung 7.2, zu nutzen, hat den Vorteil mit sich gebracht, dass ohne großen Lötaufwand, die mitgelieferten Kabel aus dem Arduino-Kit nach Schaltplan verlegt werden konnten. Somit war die Kabelführung einfacher zu gestalten.



Quelle: eigene Aufnahme

Abbildung 7.2.: Einbau des Arduinos und des PIN-Breadboards in der 3D-gedruckten Schale des Gehäuses

8. Testing

Das Testing war in diesem Projekt hauptsächlich ein Teil der Softwareentwicklung. So konnten mithilfe dieser Methode, die vorher definierten Anforderungen, auf ihre Funktionstüchtigkeit geprüft werden. Im folgenden sind zwei Testprotokolle, welche gegen Ende der Testing-Phase durchgeführt wurden, als Beispiele dargestellt.

Projekt: Entwicklung eines CO ₂ -Messers zur Simulation einer automatisierten Fensteransteuerung	Datum: 05.03.2020
ID: CO201	Version: 1.0
Titel: Visualisierung der Luftqualität auf Basis von CO ₂ -Grenzen	
Items: void ask(int)	TestKfg: 01
Zielsetzung: Der Test soll zeigen, dass die Software die gemessenen CO ₂ -Werte richtig interpretieren kann.	
Anforderungen: R01	
Erforderliche Inputs zu Testbeginn: CO ₂ -Werte	

Tabelle 8.1.: Test 1 - Testbeschreibung

Tester: Serkant Soylu	Beobachter: Alexander Herrmann
Protokoll: Zunächst wurde im Programmcode der Input der CO ₂ -Werte simuliert, indem ein Zähler den CO ₂ -Wert von 0 bis 2000 hochgezählt hat. Es konnte dabei beobachtet werden, dass die grüne LED bis zu dem Wert 799 an geblieben ist. Nach Erreichen des Wertes von 800, ist die grüne aus und die gelbe LED an gegangen. Die gelbe LED ist anschließend bis einschließlich dem Wert von 1399 an geblieben, bis sie letztendlich bei 1400 aus ging. Ab diesem Moment haben nur noch die LEDs rot und blau geleuchtet.	
Status: Erfolgreich	
Problemericht: Nicht vorhanden	

Tabelle 8.2.: Test 1 - Testprotokoll

8. Testing

Projekt: Entwicklung eines CO ₂ -Messers zur Simulation einer automatisierten Fensteransteuerung	Datum: 07.03.2020
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 und diese korrekt durchführt.	
Anforderungen: R03	
Erforderliche Inputs zu Testbeginn: Anwender	

Tabelle 8.3.: Test 2 - Testbeschreibung

Tester: Alexander Herrmann	Beobachter: Johannes Ruffer
Protokoll:	
Zunächst wurde der Arduino an den Laptop angeschlossen und somit das aktuelle Programm gestartet. Nach Auswahl des Schreibe-Modus, wurde das Messprofil-Menü angezeigt. Nach drei UP-Button-Klicks, wurde der Messmodus 1 (Echtzeitmodus) mit dem Enter-Button bestätigt. 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. Der dritte Start des Programms diente dazu, die Tagesmessung zu testen. Auch hier hat die Auswahl und Durchführung in Bezug auf die Testanforderungen erfolgreich funktioniert.	
Status: Erfolgreich	Problembericht: Nicht vorhanden

Tabelle 8.4.: Test 2 - Testprotokoll

9. Handbuch

Die Tabellen 9.1, 9.2 und 9.3 geben allgemeine Informationen über die Verwendung von Tastern und die Interpretation von Ausgaben der LEDs und des LCD-Displays an.

Betätigung	Erklärung
Linker Taster	Durch Betätigung des linken Tasters, wird nach Beenden des begonnenen Programmdurchlaufs das Gerät in den Schlafmodus versetzt.
Mittlerer Taster	In der Menüauswahl dient dieser Taster zur Bestätigung des ausgewählten Menüpunktes. Befindet sich das Gerät im Schlafmodus, wird es durch Betätigung des Tasters wieder in den wachen Zustand versetzt. Das Hauptmenü wird anschließend angezeigt.
Rechter Taster	Dieser Taster dient zum Wechseln der Menüauswahl innerhalb der Menüs.

Tabelle 9.1.: Beschreibung der Taster-Funktionen

Lampen	Erklärung
Grün	Gute Luftgüte (bis 799ppm)
Gelb	Mäßige Luftgüte (800ppm - 1399ppm)
Rot	Schlechte Luftgüte (ab 1400ppm)
Blau	Fensterscheibenmotor öffnet

Tabelle 9.2.: Bedeutung der LEDs

Warnung	Erklärung
Initialisation of SD failed	Die Initialisierung der SD-Karte ist fehlgeschlagen. Kontrollieren Sie, ob die SD-Karte sich in dem dafür vorgesehenen Adapter befindet.
Failed to start sensor	Der CO ₂ -Sensor kann nicht angesteuert werden. Kontrollieren Sie die Steckverbindungen und betätigen Sie den Reset-Taster innerhalb des Gehäuses.
ERROR!	Die Kommunikation mit dem CO ₂ -Sensor ist fehlgeschlagen. Kontrollieren Sie die Steckverbindungen und betätigen Sie den Reset-Taster innerhalb des Gehäuses.
Textfile could not be found	Fehler beim erstellen oder schreiben der Textdatei.

Tabelle 9.3.: Bedeutung der Warnungen

9.1. Erste Schritte

Kontrollieren Sie zunächst, ob sich eine Mikro-SD-Karte in dem dazu vorgesehenen Adapter befindet. Schließen Sie das Gerät zur Inbetriebnahme an ein geeignetes Stromnetz an. Generell können Sie das Gerät ganz leicht mit Ihren Fingern bedienen, indem Sie die Taster auf der Oberfläche antippen.

9.2. Durchführung einer Messung

Sie befinden sich im Hauptmenü und möchten eine neue Messung starten.

- Wählen und bestätigen Sie dazu im Hauptmenü den Punkt <Start new Measurement>.



Quelle: eigene Aufnahme

Abbildung 9.1.: Hauptmenü: <Start new Measurement>

Anschließend können Sie zwischen drei Messmodi wählen.

- Wählen und bestätigen Sie im angezeigten Menü den Messmodus, in welchem Sie verfahren möchten.



Quelle: eigene Aufnahme

Abbildung 9.2.: Messmenü: Auswahl der Messmodi

Nach erfolgreichem Abschluss der Messung, kehren Sie in das Hauptmenü wieder zurück.

9.3. Auslesen einer Messung

Sie befinden sich im Hauptmenü, möchten die Mikro-SD-Karte entfernen und die letzte Messung auslesen.

- Wählen und bestätigen Sie im Hauptmenü den Punkt <Read last Measurement>.



Quelle: eigene Aufnahme

Abbildung 9.3.: Hauptmenü: <Read last Measurement>

Anschließend leuchten die LEDs in den Farben grün, gelb und rot auf. Sie können nun die Mikro-SD-Karte entfernen und ihre Messung analysieren. Lesen Sie dazu die Mikro-SD-Karte mit Ihrem Computer aus.



Quelle: eigene Aufnahme

Abbildung 9.4.: Auslesemodus

- Öffnen Sie die Datei <MEASURE.txt> mit Excel.
- Markieren Sie die angezeigten Werte und wählen Sie ein passendes Diagramm aus.

Ihre Messung wird nun als Funktion über die Anzahl der Messungen dargestellt. Bestätigen Sie nach dem Wiedereinfügen der SD-Karte mit dem mittleren Taster. Sie befinden sich nun wieder im Hauptmenü.

9.4. Sleepmode/Stromsparmodus

Sie möchten den Sleepmode oder auch Stromsparmodus aktivieren.

- Betätigen Sie hierzu den linken Taster (Sleepmode-Button).

nach Beenden des gerade ausgeführten Programmdurchlaufs werden sowohl die LEDs, als auch das LCD-Display aus gehen.

- Betätigen Sie zum aufwecken des Programms den mittleren Taster (Enter-Button).

Das Programm wird nun erneut gestartet. Der Startpunkt ist dabei das Hauptmenü.
Das Drücken der anderen beiden Tasten wird während dem Sleepmode ignoriert.

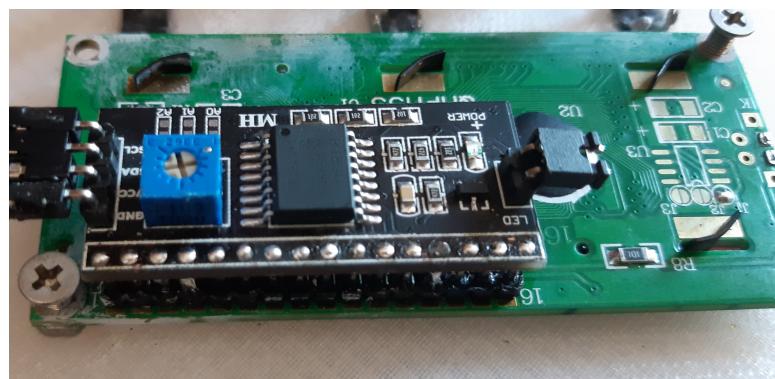
10. Installationsanleitung

Nachfolgend finden Sie eine Anleitung, wie Sie den CO₂-Messer in Betrieb nehmen. Bitte beachten Sie auch die zusätzlichen Hinweise.

Damit die Software erfolgreich vom Computer auf den Arduino geladen werden kann, müssen die folgenden Bibliotheken 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)
- SD.h (Ermöglicht die Kommunikation mit dem Mikro-SD-Karten-Adapter)

Zunächst muss der I²C-Adapter, wie in Abbildung 10.1 abgebildet ist, an das LCD-Display angeschlossen werden. Achten Sie auf die Richtung der beiden Elemente.



Quelle: eigene Aufnahme

Abbildung 10.1.: Anschluss des I²C-Adapters an das LCD-Display

In der Tabelle 10.1 sind alle Verbindungen zwischen den einzelnen Komponenten mit dem Arduino aufgelistet. In Abbildung 4.4 ist das entsprechende Layout als Zeichnung zu finden. Achten Sie darauf, dass der Arduino während dieser Arbeit vom Stromnetz getrennt ist, um Kurzschlüsse zu vermeiden.

Nummer	Bauteil	Spezifikation	Anschlusspin Arduino
1	I ² C-Adapter	GND SDA SCL	GND A1 A0
2	LEDs	Rot Gelb Grün Blau	A3 A2 A1 A0
3	Taster	Up-Button Enter-Button	12 13
4	CCS811	WAK SDA SCL VCC	GND SDA SCL 5V
5	Mikro-SD-Adapter	GND MISO MOSI SCK CS VCC	GND D11 D10 9 8 5V

Tabelle 10.1.: Zuordnung der Pins

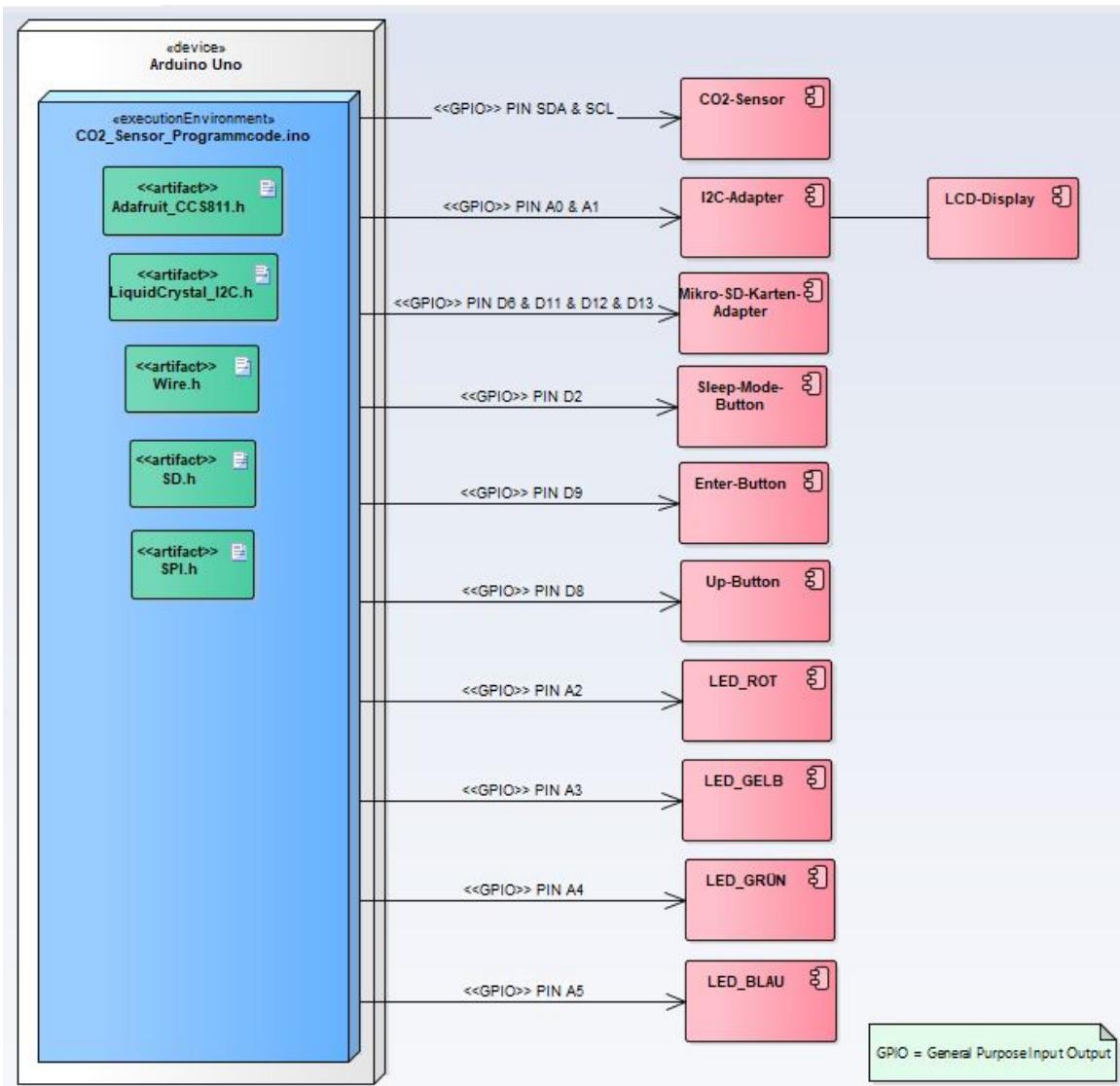
Auch die Vorwiderstände, welche Sie in Tabelle 10.2 sehen können, sind essentiell, um eine erfolgreiche Installation zu garantieren.

Nummer	Bauteil	Vorwiderstand
1	LEDs	330Ω
2	Taster	2KΩ

Tabelle 10.2.: Verwendete Vorwiderstände innerhalb der Schaltung

Bevor der Arduino an das Stromnetz geschlossen werden kann, kontrollieren Sie, ob sich eine Mikro-SD-Karte in dem dafür vorgesehenen Adapter befindet. Verbinden Sie anschließend das Gerät mit einer geeigneten Stromquelle. Das Menü öffnet sich und das Gerät ist installiert. Achten Sie darauf, dass der Arduino während einer Messung nicht von der Stromquelle getrennt

wird.



Quelle: eigene Aufnahme

Abbildung 10.2.: Zeichnung des Deploymentdiagramm, zum Verständnis der Integration von Soft- und Hardware und deren Verbindungen, mithilfe von Enterprise Architect

11. Fazit

In dem folgenden Abschnitt ist das Fazit in zwei Teile gegliedert. Die Begründung liegt darin, dass die Entwicklung von Soft- und Hardware parallel geschehen ist. Es werden hierbei sowohl positive Aspekte des Projektes aufgegriffen, als auch auf mögliche Verbesserungen aufmerksam gemacht.

11.1. Software

Nachdem wir uns auf ein Projekt geeinigt hatten, wurden alle Anforderungen dafür definiert. Die Planung konnte anschließend erfolgen, in welcher wir uns überlegt haben, welche Komponenten wir für die Umsetzung benötigen.

Während der Softwareentwicklung ergab sich dann trotz dem anfangs entworfenem Schaltungslayout, dass wir zu wenig digitale PINs am Arduino zu Verfügung hatten. Diese späte Erkenntnis kam zu Stande, da wir während der Entwicklung festgestellt haben, dass an die Arduino-PINs Null und Eins keine Hardwarekomponenten angeschlossen werden können. Auch an den PIN AREF, konnten keine von uns verwendeten Komponenten angeschlossen werden. Somit fehlten Steckmöglichkeiten, sodass wir kurzfristig einen I₂C-Adapter an das LCD-Display anschließen mussten. So konnten wir zum einen Anschlussmöglichkeiten am Arduino sparen, zum anderen wird der Adapter mit analogen PINs verbunden, sodass uns umso mehr digitale PINs zu Verfügung standen.

Eine weitere Änderung im Layout musste aufgrund der Interrupt-Funktion des Sleep-Mode-Buttons vorgenommen werden. Die Tatsache, dass nur die digitalen PINs Zwei und Drei am Arduino für Interrupts genutzt werden können, war uns anfangs nicht bewusst. Somit wurde auch hier während der Entwicklung das in der Planung gezeichnete Schaltungslayout geändert. Die softwaretechnische Integration des CO₂-Sensors, hat sich aufgrund der Suche nach einer geeigneten Bibliothek und dem anschließenden Einarbeiten in jene, zeitlich etwas gestreckt. Auch in der softwaretechnischen Integration des Mikro-SD-Karten-Adapters, haben sich der zeitliche Aufwand von Finden und Verstehen einer passenden Bibliothek, bemerkbar gemacht.

Abschließend kann man sagen, dass trotz kleinen Komplikationen während der Softwareentwicklung, das Projekt dank einer guten Planung erfolgreich umgesetzt wurde.

11.2. Hardware

Als Endergebnis ist die Hardwareimplementierung als ein ebenso spannendes Feld wie die restlichen Entwicklungsschritte anzusehen. Trotz einer guten Vorarbeit können hier Komplikationen, wie beispielsweise eine nicht korrekt angeschlossene Leitung, auftreten.

Die Entscheidung, das Gehäuse selbst im 3D-Druck durchzuführen, hat sich als riesen Vorteil herausgestellt. Auch, wenn es zunächst einen größeren Aufwand bedeutet, die Skizzen anzufertigen und diese sogar über einen ganzen Tag hinweg drucken zu lassen, konnten wir relativ schnell mehrere Varianten des Gehäuses drucken. Es war nun möglich, diese verschiedenen Versionen schnell zu testen, da wir nach der ersten Zeichnung relativ flexibel für Änderungen waren. Zudem war garantiert, dass das Gehäuse all unseren Anforderungen entsprechen wird.

12. Ausblick

In dem folgenden Abschnitt ist der Ausblick, ebenso wie das Fazit, aufgrund der parallelen Entwicklung während des Projektes, in zwei Teile gegliedert. Der Ausblick soll zeigen, welche Möglichkeiten es gibt, dieses Projekt weiterzuentwickeln und zu optimieren.

12.1. Software

Hinsichtlich der Softwareentwicklung wäre es möglich, das LCD-Display durch ein Organic Light Emitting Diode (OLED)-Display zu ersetzen. Diese Änderung kann zu einer anschaulicheren Darstellung für den Benutzer eingesetzt werden. Ein Beispiel hierfür wäre eine übersichtlicher Darstellung der Menüanzeige.

Auch das Speichern der, zu den Messungen passenden Uhrzeiten, wäre eine Ergänzung zu dem jetzigen Format. So könnte das plotten der Graphiken nicht über die Anzahl der Messungen, sondern über die Zeit ermöglicht werden. Dazu müsste jedoch garantiert sein, dass der Arduino über eine Internet- oder Computerverbindung verfügt, sodass er die korrekte Uhrzeit auslesen kann.

Zudem könnte der Arduino das Plotten der Messungen in Form von Graphiken übernehmen. Diese könnten beispielsweise auf der bereits implementierten Mikro-SD-Karte abgespeichert werden.

Zu guter Letzt könnte die zeitliche Ansteuerung des Fensterscheibenmotors optimiert werden. Da der Fensterscheibenmotor bisher mit der roten LED gleichgeschaltet ist, könnte es zu einem Toggeln zwischen mäßiger und schlechter Luftqualität kommen. Dies würde durch das Trennen dieser Gleichschaltung verhindert werden. Zudem würde die Räumlichkeit, bis zum Erreichen einer guten Luftqualität, gelüftet werden, falls dies gewünscht ist. Dazu müsste die Funktion, welche für die Interpretation der Luftgüte und somit für das An- und Ausschalten der LEDs zuständig ist, angepasst werden. Die Funktion muss dafür vom Grundaufbau geändert werden. Dies hat zur Folge, dass die Funktion sofort komplexer wird. In diesem Zusammenhang fände das Auslagern und Aufrufen jener Funktion nicht mehr in dieser Form statt.

12.2. Hardware

Eine Möglichkeit, unsere bestehende Hardware weiter zu verfeinern ist, auf ein kleineres PIN-Breadboard zurückzugreifen. Durch eine solche Verfeinerung könnte das gesamte Gehäuse kleiner und handlicher werden.

Zudem ist durch die Verwendung von anderen Leitungen eine einfachere Kabelführung möglich, da die uns mitgelieferten Kabel unflexibel und starr waren. Somit ist eine weitere Verkleinerung des Gehäuses möglich.

Die Integration einer Spannungsversorgung über eine Batterie in das Gehäuse würde zudem die Mobilität wesentlich erhöhen. Im aktuellen Prototyp ist die Spannungsversorgung über einen USB-Port am Laptop geregelt wird, wodurch der CO₂-Besser unhandlicher ist. Allerdings muss dabei beachtet werden, dass die Integration einer Batterie auch ein größeres Gehäuse mit sich bringt.

Verzeichnis verwendeter Abkürzungen und Formelzeichen

DHBW Duale Hochschule Baden-Würtemberg

MOX Metalloxid

AD Analog-Digital

LCD LiquidCrystal Display

OLED Organic Light Emitting Diode

HDMI High Definition Multimedia Interface

LED Light Emitting Diode

CAD Computer-Aided Design

STL Standard Template Library

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
- [.CiNS] *Kohlenstoffdioxid in Räumen messen und überwachen: CO₂ - Definition und Bedeutung.* <https://www.cik-solutions.com/anwendungen/co2-im-innenraum/>. Version: CiK SOLUTIONS
- [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 Defi-*

nition 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
- [Umw17] UMWELTBUNDESAMT (Hrsg.): *Gesundheitliche Bewertung von Kohlendioxid in der Innenraumluft: Mitteilungen der Ad-hoc-Arbeitsgruppe Innenraumrichtwerte der Innenraumluft-hygiene-Kommission des Umweltbundesamtes und der Obersten Landesgesundheitsbehörden.* https://www.umweltbundesamt.de/sites/default/files/medien/pdfs/kohlendioxid_2008.pdf. Version: 2017
- [Wik] WIKIPEDIA (Hrsg.): *Kohlenstoffdioxid in der Erdatmosphäre.* https://de.wikipedia.org/w/index.php?title=Kohlenstoffdioxid_in_der_Erdatmosph%C3%A4re&oldid=19000000.

[wikipedia.org/wiki/Kohlenstoffdioxid_in_der_Erdatmosph%C3%A4re](https://de.wikipedia.org/wikipedia/Kohlenstoffdioxid_in_der_Erdatmosph%C3%A4re)

[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_\(Plattform\)](https://de.wikipedia.org/wiki/Arduino_(Plattform)). Version: 2020

Abbildungsverzeichnis

2.1. Entwurf des Use-Case-Diagramms mithilfe von Enterprise Architect	5
3.1. Ablauf des Projektes von Januar bis April	7
3.2. Aufgabenverteilung	8
3.3. Lines of Code Schätzung	10
4.1. Entwurf des Zustandsdiagramms mithilfe von Enterprise Architect	12
4.2. Entwurf der Top-Level-Architektur	13
4.3. Entwurf des Komponentendiagramms mithilfe von Enterprise Architect	14
4.4. Entwurf des Schaltungslayouts mithilfe von Fritzing	15
4.5. CAD-Zeichnung für den Deckel des Gehäuses	16
4.6. CAD-Zeichnung für die Schale des Gehäuses	17
4.7. Simulation der STL-Datei	18
5.1. Arduino Uno	20
5.2. Verwendeter CCS811-MOX-Gassensor	21
5.3. Komponentendiagramm des CCS811	21
5.4. Inhalt der 8-Byte-Übertragung des CCS811-MOX-Sensors	22
5.5. Codeausschnitt zur Sensor-Kontrolle im aus dem Quellcode	22
5.6. Codeausschnitt für die Messung aus dem Quellcode	23
5.7. Verwendeter Mikro-SD-Karten-Adapter, mit Sicht auf dessen Vorderseite	23
5.8. Verwendeter Mikro-SD-Karten-Adapter, mit Sicht auf dessen Rückseite	24
5.9. Codeausschnitt zur Initialisierung der SD-Karte im Setup aus dem Quellcode	24
5.10. Verwendeter I ² C-Adapter	25
5.11. Codeausschnitt zur Definition von Adresse und Anzahl der zu verwendenden Zeichen und Reihen	25
6.1. Codeausschnitt aus dem Setup zur Deklaration der Interrupt-Funktion	27
6.2. Codeausschnitt: ausgelagerte Interrupt-Funktion mit Tastenentprellung, für das Werten der Variable, zur späteren Aktivierung des Schlafmodus	29
6.3. Codeausschnitt aus dem Loop zur Aktivierung und Deaktivierung des Schlafmodus	30

Abbildungsverzeichnis

7.1.	3D-gedruckter Deckel des Gehäuses	31
7.2.	Einbau des Arduinos und des PIN-Breadboards in der 3D-gedruckten Schale des Gehäuses	31
9.1.	Hauptmenü: <Start new Measurement>	36
9.2.	Messmenü: Auswahl der Messmodi	37
9.3.	Hauptmenü: <Read last Measurement>	37
9.4.	Auslesemodus	37
10.1.	Anschluss des I ² C-Adapters an das LCD-Display	39
10.2.	Zeichnung des Deploymentdiagramm, zum Verständnis der Integration von Software und Hardware und deren Verbindungen, mithilfe von Enterprise Architect	41
A.1.	Draufsicht auf fertig verbaute Hardware	I
A.2.	Slot für Spannungsversorgung in fertig verbauter Hardware	I
A.3.	Herausgelegter Mikro-SD-Karten-Adapter in fertig verbauter Hardware	II
A.4.	Herausgelegter CO ₂ -Sensor in fertig verbauter Hardware	II

Tabellenverzeichnis

2.1. Anforderungen an das Projekt	4
3.1. Aufgewendete Kosten für das Projekt	9
8.1. Test 1 - Testbeschreibung	33
8.2. Test 1 - Testprotokoll	33
8.3. Test 2 - Testbeschreibung	34
8.4. Test 2 - Testprotokoll	34
9.1. Beschreibung der Taster-Funktionen	35
9.2. Bedeutung der LEDs	35
9.3. Bedeutung der Warnungen	36
10.1. Zuordnung der Pins	40
10.2. Verwendete Vorwiderstände innerhalb der Schaltung	40

A. Anhang

A.1. Weitere Abbildungen



Quelle: eigene Aufnahme

Abbildung A.1.: Draufsicht auf fertig verbaute Hardware



Quelle: eigene Aufnahme

Abbildung A.2.: Slot für Spannungsversorgung in fertig verbauter Hardware



Quelle: eigene Aufnahme

Abbildung A.3.: Herausgelegter Mikro-SD-Karten-Adapter in fertig verbauter Hardware



Quelle: eigene Aufnahme

Abbildung A.4.: Herausgelegter CO₂-Sensor in fertig verbauter Hardware