Heliumzuiverheidsmeter

Scriptie

In opdracht van: Universiteit Leiden (ELD)

De Haagse Hogeschool

Student:
Thijs Romeijn | Voltijd HHS |

Datum:
4-juni-2021, Boskoop

Opdrachtgever:
Universiteit Leiden, ELD

Begeleiders:
E.J. Visser

1e Coach:
E. Korkmaz

2e Coach:
B. Kuiper

Versie:
1.0

Universiteit Leiden

1 Versiebeheer

Tabel 1: Versiebeheer

Versie	Samenvatting	Auteur	Datum
V0.1	concept	T. Romeijn	8-2-2021
V0.2	Feedback hoofdstukken indeling	I.C. Muller	10-5-2021
V0.3	Inhoudelijke feedback	R. Koehler	27-5-2021
V0.4	Spelling en feedback	J.A. Boer	28-5-2021
V0.5	Samenvatting Engels	N.M. Boer	01-6-2021
V0.6	Feedback inhoudelijk	E.J. Visser	01-6-2021
V0.7	Toevoegen competenties	T. Romeijn	01-6-2021
V0.8	Controleren van de layout scriptie en aanpassen competenties	N.M. Boer	02-6-2021
V0.85	Conclusie aanpassen	E. Korkmaz	02-6-2021
V0.9	Feedback inhoudelijk	E.J. Visser	03-6-2021
V1.0	Scriptie ingeleverd	T. Romeijn	04-6-2021

2 Voorwoord

Het is vier jaar geleden dat ik ben afgestudeerd als Middenkader engineer bij de Universiteit Leiden. Tijdens die afstudeerstage ben ik gemotiveerd geraakt om door te gaan naar het HBO. Om nu na vier jaar terug te zijn voor afstuderen als afronding van de HBO-opleiding elektrotechniek voelt toch speciaal.

In de vier jaar tijdens mijn studie heb ik veel steun gehad. Ik wil daarom N.M. Boer, I.C. Muller en I. van Bommel in het bijzonder bedanken voor alle steun en plezier tijdens mijn opleiding.

Graag wil ik Dhr. R. Overgauw bedanken voor de tijd en moeite die is genomen om dit afstudeerproject te regelen. Ook voor het gebruik van zijn fiets zodat ik van het station naar de Universiteit kon fietsen.

Als laatst wil ik ook Dhr. E.J. Visser en Dhr. R. Koehler bedanken voor alle hulp tijdens het project. Als ik een vraag had kon ik altijd terecht en werd er tijd voor mij vrij gemaakt.

Thijs Romeijn
Boskoop, 4-6-2021

3 Samenvatting

Op de Faculteit natuurkunde binnen de Universiteit Leiden worden veel onderzoeken uitgevoerd met Helium. Het Helium wordt onder andere gebruikt voor onderzoeken bij Cryogene temperaturen. Wanneer het Helium vervuild raakt met zuurstof en stikstof, zorgt dit voor ijsvorming in de opslagtanks. De oplossing is om een meter te plaatsen in de retourleiding van de opstelling en zo de zuiverheid van het Helium te meten. Wanneer het Helium te veel vervuild is, wordt een alarm ingeschakeld.

De Heliumzuiverheidsmeter is een meter die aan de hand van de geluidssnelheid de zuiverheid van het Helium kan meten. Dit kan omdat de geluidssnelheid in Helium ongeveer 3 keer zo hoog is als in lucht.

Het meten van de geluidssnelheid wordt gedaan door gebruik te maken van een luidspreker en microfoon. Op het moment dat de luidspreker een puls (geluidsgolf) uitstuurt, wordt een timer gestart. Wanneer de microfoon de geluidsgolf ontvangt, wordt de timer gestopt en wordt de geluidssnelheid berekend. Het meten van de tijd wordt gedaan met een microcontroller.

Iedere minuut wordt de zuiverheid van het Helium gemeten. Om de kans op statistische uitschieters van een meting te verkleinen, worden er 100 metingen gedaan waarvan het gemiddelde wordt genomen. De zuiverheid wordt naar een server gestuurd samen met de temperatuur van het Helium. De server bepaalt welk type alarm er gegeven moet worden. Met een drukknop aan de voorkant van de behuizing kan het alarm uitgeschakeld worden.

Met een 7-segments display wordt voor de gebruiker de zuiverheid (0 … 100 %) van het Helium weergegeven. Verder wordt op het 7-segmentsdisplay een melding gegeven wanneer er geen verbinding met de server gemaakt kan worden en wanneer de zuiverheid van het Helium minder is dan 95 %.

4 Summary

At the Faculty of Physics at Leiden University, many studies are conducted with Helium. Helium is used, among other things, for studies at cryogenic temperatures. When Helium becomes contaminated with oxygen and nitrogen, it causes ice to form in the storage tanks. The solution is to place a meter in the return pipe of the setup and thus measure the purity of the Helium. When the Helium is too contaminated, an alarm is triggered.

The Helium Purity Meter is a meter that can measure the purity of Helium based on the speed of sound. This is possible because the speed of sound in Helium is about 3 times as high as in air.

Measuring the speed of sound is done using a speaker and microphone. The moment the speaker transmits a pulse (sound wave), a timer is started. When the microphone receives the sound wave, the timer is stopped and the sound speed is calculated. Time measuring is done using a microcontroller.

The Helium's purity is measured every minute. To reduce the chance of static outliers in a measurement, the average of a 100 measurements is taken. The purity data is sent to a server along with the Helium temperature. The server determines which alarm goes off. The alarm can be switched off with a push button on the front of the housing.

With a 7-segment display, the purity (0 … 100 %) of the Helium is displayed to the user. In addition, the 7-segment display notifies you when the server cannot connect and when Helium purity is less than 95%.

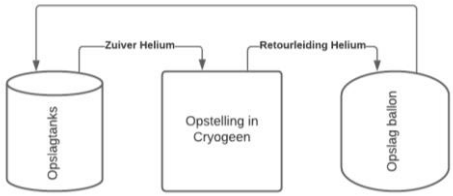
5 Begrippenlijst

PCB	-	Printed circuit board
LED	-	Light emitting diode
Op-Amp	-	Operational amplifier
Tweeter	-	Luidspreker voor de hogere frequenties
Woofer	-	Luidspreker voor de lagere frequenties
Sigma	-	Standaardafwijking
Transducer	-	Omzetter
MOSFET	-	Elektronische schakelaar die schakelt op spanning
Transistor	-	Elektronische schakelaar die schakelt op stroom
HPF	-	Hoog doorlaat filter
LPF	-	Laag doorlaat filter
Solenoid	-	Elektronische klep
Elco	-	Elektrolytische condensator (grote capaciteit, unipolair)
Tantaal	-	Condensator, unipolair met kleinere capaciteit en inwendige weerstand dan de elco
Diode	-	Halfgeleider die de stroom slechts in een richting doorlaat
Zenerdiode	-	Diode die in sperrichting geleidend wordt boven de zenerspanning
SMD	-	Surface Mounted Device: Component met soldeerbevestiging aan componentzijde
Hysterese	-	Verschil in omslagspanning tussen toestand met uitgang=hoog en uitgang = laag
Heatsink	-	Metaal oppervlakte dat warmte overdraagt naar zijn omgeving
ΔØ	-	Faseverschuiving
Headers	-	Verbindingsstukken
U	-	Spanning
I	-	Stroom
R	-	Weerstand

6 Inhoud

7 Inleiding

Op de faculteit natuurkunde binnen de Universiteit Leiden worden veel onderzoeken uitgevoerd met helium. Het helium wordt ingekocht en opgeslagen in verrijdbare opslagtanks. Vervolgens gaat het helium van de opslagtanks naar de opstelling, waarmee er onderzoeken bij Cryogene temperaturen (vloeibaar Helium -273 °C) uitgevoerd worden. Als het helium opwarmt, verandert het van vloeibaar in gas. Het verdampte helium gaat via een retourleiding terug en wordt opgeslagen in een ballon. De ballon fungeert als buffervat. Als de ballon vol is, wordt het helium teruggepompt in de verrijdbare opslagtanks. Het helium wordt op deze manier niet verspild. In Figuur 1 is het systeem proces weergegeven.



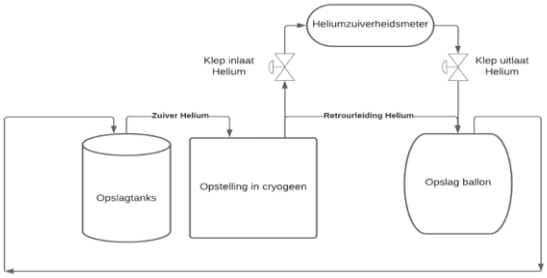
Figuur 1 Systeem Proces

Door een defect in een meetopstelling waarin helium wordt gebruikt, kan het helium vervuild raken met omgevingslucht, waardoor zuurstof en stikstof in het helium terecht komen. Dit zorgt voor ijsvorming in de opslagtanks. Het is daarom belangrijk om een dergelijke verontreiniging zo snel mogelijk te constateren. Wanneer iedere meetopstelling beschikt over een Heliumzuiverheidsmeter, kan zo'n fout lokaal worden opgespoord.

De afdeling Cryogeen van het Leids Instituut Onderzoek Natuurkunde (LION) heeft de Elektronische Dienst (ELD) daarom gevraagd of zij hier een oplossing kunnen realiseren.

Om de zuiverheid van het Helium te meten maakt de ELD gebruik van een meter van het merk: Analox ATA pro. Deze meter maakt gebruik van een chemische sensor. Het probleem van deze meter is dat het geteste helium wordt vrijgelaten in de omgeving in plaats van terug te keren naar de opslagtank. Omdat Helium een kostbaar gas is, mag dit niet verspild worden. Verder is de huidige meter duur (€ 1200,-). Onder continu gebruik zal de sensor snel verouderen. Wanneer de Heliumsensor versleten is, zal deze vervangen moeten worden. De sensor is los te verkrijgen voor € 220.-. Ook zou de meter moeten worden aangepast zodat de data over de zuiverheid van het Helium naar een server gestuurd kan worden.

Voor het afstudeerproject zal een Heliumzuiverheidsmeter gerealiseerd worden zonder deze nadelen. Het doel van het project is om een goedkopere Heliumzuiverheidsmeter te ontwikkelen, die na het meten het Helium terug in het proces kan laten (Figuur 2).



Figuur 2: Gewenste situatie

De Heliumzuiverheidsmeter zal de zuiverheid van het Helium meten aan de hand van de geluidssnelheid. Het meten van de geluidssnelheid wordt gedaan met een luidspreker, microfoon en microcontroller. Een luidspreker voor het uitzenden van een geluidsgolf, een microfoon voor het ontvangen van de geluidsgolf en de microcontroller voor het meten van de tijd.

7.1 Eisen

De naam van het project is: 'Heliumzuiverheidsmeter'. Het primaire doel van de opstelling is het meten van de zuiverheid van het helium. Het project wordt als succesvol ervaren wanneer de meter betrouwbaar een concentratie van 5% vervuiling in het Helium kan meten. De volgende eisen zijn vastgesteld:

- Het kunnen meten van 5 % vervuiling in het Helium;
- Een 7-segments display waarop de zuiverheid van het Helium wordt weergeven;
- Een ethernetverbinding zodat de data verstuurd kan worden naar een plaatselijke server;
- Een hermetisch afgesloten behuizing zodat het Helium niet kan lekken;
- Een alarmering aan de hand van geluidsignalen;
- Een drukknop zodat het alarm uitgeschakeld kan worden;[1]
- De status van de drukknop doorsturen naar server;[1]
- Een temperatuursensor voor het meten van de temperatuur in de opstelling;
- Het sturen van data over de temperatuur in Kelvin (K) naar de server.

7.2 Projectgrenzen

De volgende projectgrenzen zijn vastgesteld:

- De behuizing wordt door de afstuderende student ontworpen, maar wordt door de FMD opgebouwd;
- Er wordt bij de berekening van de Heliumconcentratie uitgegaan van een temperatuur van 21 °C;
- De natuurconstanten met betrekking tot de geluidssnelheden op 21 °C worden door de afdeling natuurkunde aangeleverd;
- Er wordt alleen software geschreven voor de microcontroller;
- De Heliumzuiverheidsmeter is een zelfstandig werkend apparaat met een eigen voeding en een 230V input via een eurostekker. Bij tijdgebrek kan een externe stekkeradapter worden gebruikt.
- Optioneel zal een 2e printplaat ontwikkeld worden met daarop een microcontroller en relais. De microcontroller sluit een relais. Het gesloten relais zorgt dat er iemand gebeld wordt. De hard- en software die hiervoor nodig is, zal als optioneel worden beschouwd.

1. Aangepaste eisen, afwijkend van eisen in PvA.

8 Theoretisch kader meten in Helium

Helium is een edelgas en bevindt zich als 2e element binnen het periodiek stelsel. Helium is een kleur- en reukloos gas en is niet giftig [1].

Helium kent verschillende toepassingen en wordt bijvoorbeeld gebruikt in ballonen. Op de Universiteit Leiden wordt het Helium gebruikt als koelmiddel. Doordat Helium een laag smelt- en kookpunt heeft, maakt dat het uitermate geschikt voor onderzoek in het cryogeen. Veel onderzoeken worden gedaan in het Cryogeen omdat er bij Cryogene temperaturen geen moleculaire beweging is. Dit is gewenst bij het uitvoeren van verschillende experimenten. Een nadelig effect van Helium is dat het een erg dun gas is [2]. Voor de Heliumzuiverheidsmeter moet daarom een hermetisch afgesloten behuizing worden gebruikt.

Het meten van de verontreiniging in Helium kan op verschillende manieren gedaan worden.

- Chemische Heliumdetectie;
- Fasemeting;
- Geluidssnelheid.

8.1 Fasemeting

(8.1.1) $v = \frac{2\pi \cdot f \cdot l}{\Delta \emptyset}$

v: Geluidssnelheid [m/s]
f: Frequentie [Hz]
l: lengte [m]
$\Delta\emptyset$: faseverschuiving [graden]

Door het meten van het faseverschil tussen de in- en uitgang kan de geluidssnelheid worden bepaald [3]. Met formule (8.1.1) kan de geluidssnelheid worden berekend aan de hand van de faseverschuiving. v is de geluidssnelheid in m/s, f is de frequentie, l is de afstand tussen zender en ontvanger. $\Delta\emptyset$ is de faseverschuiving. Wanneer de fase met een frequentie van 1 kHz gemeten wordt, kan er iedere 0,001 seconden een meting gedaan worden. Met het meten van de faseverschuiving moet ook rekening gehouden worden met golflengte van de bijbehorende frequentie. De golflengte wordt berekend met formule (8.1.2).

(8.1.2) $\lambda = \frac{v}{f}$

λ: golflengte [m]
v: geluidssnelheid [m/s]
f: frequentie [Hz]

De golflengte bepaald hoeveelheid signaal er bij de ontvanger aankomt.

8.2 Tijdsmeting

(8.2.1) $v = \frac{S}{T}$

v: snelheid [m/s]
s: afstand [m]
t: tijd [s]

Een andere methode voor het bepalen van de zuiverheid in het Helium is door het meten van de geluidssnelheid door middel van een tijdsmeting. Een geluid kan zich alleen in een medium voortbewegen (vast, vloeibaar of gasvormig). De geluidssnelheid van Helium is

anders dan die van lucht. Dit komt doordat Helium een licht gas is. Helium is een dun gas, waardoor het moeilijk is om voldoende moleculen in beweging te krijgen. Lucht is een mengsel van stikstof (78 %) en zuurstof (20 %) en is een zwaar gas in tegenstelling tot helium.

De geluidssnelheid (bij een temperatuur van 20 °C) van lucht is 343 m/s (1243,8 km/h). Voor Helium is de geluidssnelheid hoger namelijk 965 m/s (3474 km/h) [4].

De snelheid van het geluid is te berekenen met formule (8.2.1). S staat voor de afstand in meters, v staat voor snelheid in m/s en t staat voor tijd in seconde.

De afstand (s) is een constante en wordt gegeven door de lengte van de buis. De snelheid (v) is afhankelijk van de vervuiling in het Helium en kan gemeten worden aan de hand van de tijd.

Een belangrijke parameter is de tijd. De tijd wordt gemeten met een microcontroller. Vanwege de hoge geluidssnelheid moet de microcontroller snel genoeg zijn om de tijd te meten. Hoe hoger de snelheid van de microcontroller, hoe hoger de resolutie. In hoofdstuk 14 staat beschreven welke microcontroller gebruikt is.

Met formule (8.2.1) is de tijd berekend die nodig is om door de buis te gaan (20 °C). De maximale snelheid is berekend op de snelheid van Helium en lucht. In Tabel 2 is een overzicht van de verschillende snelheden afhankelijk van de lengte van de buis.

Tabel 2: Geluidssnelheid afhankelijk van de afstand

Lengte Buis (cm)	Snelheid Helium (m/s)	Snelheid Lucht (m/s)	Tijd in Helium (µs)	Tijd in lucht (µs)
5	965	343	51,81	145,77
10	965	343	103,66	219,55
15	965	343	155,44	437,32
20	965	343	207,25	583,09
25	965	343	259,07	728,86
30	965	343	310,88	874,64
35	965	343	362,69	1020,40
40	965	343	414,51	1166,19
45	965	343	466,32	1311,95
50	965	343	518,13	1457,73

Een belangrijk onderdeel is bepalen van de reflecties in een buis. Een nieuwe meting kan namelijk pas uitgevoerd worden wanneer alle reflecties uitgedoofd zijn. In Hoofdstuk 9 zullen verschillende testen gedaan worden om te zien wat het effect is van de reflecties en hoe deze verminderd kunnen worden.

8.3 Temperatuur afhankelijkheid

(8.3.1) $v = \sqrt{\Upsilon \frac{RT}{M}}$

v: geluidssnelheid [m/s]
Υ: specifieke-warmteverhouding [J/kgK]
R: algemene gasconstante [J/mol K]
T: temperatuur [K]
M: molaire massa[g/mol]

De temperatuur van de omgeving heeft effect op de geluidssnelheid in zowel lucht als Helium. Met formule (8.3.1) kan de geluidssnelheid worden berekend afhankelijk van de temperatuur [5]. v is daarbij de geluidssnelheid in m/s, Υ is de specifieke-warmteverhouding (1,41 voor lucht, 1,63 voor Helium), R is de algemene gasconstante (8,3145 J/(mol K)). T is de absolute temperatuur in Kelvin. M is de molaire massa (0,0289 voor lucht, 0,004 voor Helium) [6]. In Tabel 3 is een overzicht van de geluidssnelheid ten gevolgen van de temperatuur.

Tabel 3: geluidssnelheid en temperatuur

Temperatuur (°C)	Geluidssnelheid lucht (m/s)	Geluidssnelheid Helium (m/s)
18	343,66	993,21
19	344,25	994,91
20	344,43	996,61
21	345,01	998,31
22	345,60	1000,00
23	346,19	1001,70
24	346,19	1003,39

Wanneer de temperatuur in lucht met 6 graden toeneemt zal ook de geluidssnelheid toenemen met: 2,53 m/s. In het Helium zal de geluidssnelheid toenemen met: 10,18 m/s. Een verschil in temperatuur heeft een groter effect in het Helium dan in lucht. Wanneer de temperatuur toeneemt met 6 graden Celsius zal de meting in Helium ongeveer 1 % afwijken.

Aangenomen is dat de omgevingstemperatuur bij de afdeling natuurkunde ligt rond de 21 °C.

8.4 Diameter buis

De diameter van de buis waarin het Helium zich bevindt heeft invloed op de geluidssnelheid. Wanneer de diameter te klein wordt heeft dit effect op de geluidssnelheid [7]. Voor het eindontwerp zal daarom niet gebruikt worden gemaakt van een buis met een diameter kleiner dan 1 mm.

8.5 Het maken/ontvangen van een geluidsgolf

Voor het maken van een geluidsgolf kan een luidspreker gebruikt worden. Een luidspreker kan een elektrische spanning omzetten in een geluidsgolf. Luidsprekers zijn te verkrijgen als *woofers* en *tweeters*. Er is gekozen om gebruik te maken van een tweeter omdat deze een hoger frequentiebereik heeft. Omdat Helium een licht en daarom dun gas is, zal de tweeter genoeg vermogen moeten hebben om voldoende microfoonsignaal op te wekken.

Voor het ontvangen van de geluidsgolf kan een microfoon gebruikt worden. Een microfoon is een transducer die een geluidsgolf omzet in een elektrisch signaal.

8.6 Conclusie

De snelheid van het geluid in Helium is afhankelijk van de lengte van de buis. Hoe langer de buis, hoe langer het duurt voordat de geluidsgolf bij de ontvanger is. Hoe langer de buis is, hoe groter de behuizing wordt. Omdat Helium een licht molecuul is, is er veel vermogen nodig om het geluid zich te laten voortplanten.

De temperatuur van de omgeving in lucht heeft effect op de geluidssnelheid. Wanneer de temperatuur oploopt van 18 naar 24 graden, geeft een verschil van: 10,18 m/s in Helium. In lucht geeft dit een verschil van: 2,53 m/s. De temperatuur heeft dus meer effect in Helium dan in lucht. Wanneer de temperatuur met 1 graad per Celsius toeneemt zal de meting 0,17 % afwijken.

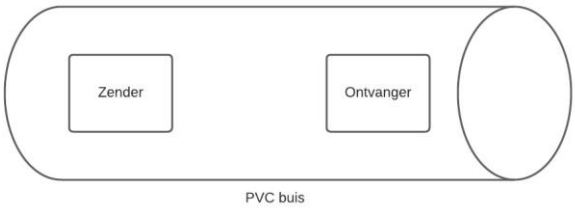
9 Testopstelling

Om te bepalen wat de beste methode is voor het meten van de zuiverheid in het Helium worden verschillende testen uitgevoerd. De hypothese is als volgt: Een buis met een grote diameter geeft veel reflecties maar er komt veel signaal aan bij de ontvanger. Het gebruikt van een dunne buis geeft minder reflecties maar zorgt ook voor een verzwakking bij de ontvanger. De eerste testen worden uitgevoerd in lucht. Wanneer de testen met lucht als medium succesvol zijn, zullen de metingen in het Helium plaats vinden.

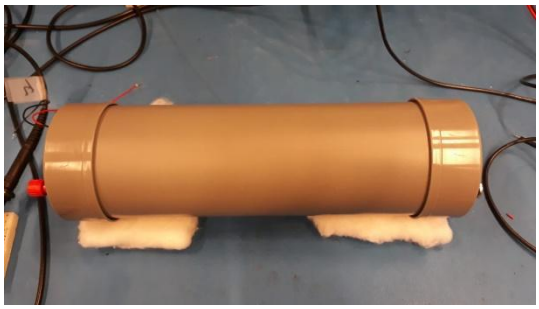
De test procedure is als volgt:

1. De zender stuurt een puls (stap) uit;
2. De ontvanger aan de andere kant van de buis ontvangt de puls;
3. Met de tijd die nodig was om door de buis te gaan, wordt de geluidssnelheid berekend.

Voor de test is een cilindervormige buis gebruikt van 25 cm lang met een diameter van 7 cm. De diameter van de buis is zo gekozen dat de zender erin past. Volgens Tabel 1 is de tijd die nodig is om van de zender naar de ontvanger te gaan 728,86 µs. De test is uitgevoerd volgens het onderstaande schema (Figuur 3 en Figuur 4). De zender en ontvanger bevinden zich aan de binnenkant van de pvc-buis. Aan de kant van de ontvanger is een oscilloscoop aangesloten die het signaal zichtbaar maakt. Met een opamp wordt het inkomende signaal 30 keer versterkt zodat het signaal duidelijk zichtbaar is.

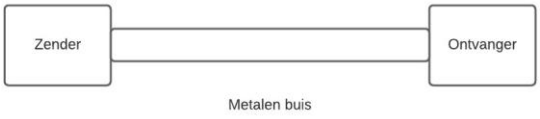


Figuur 3: Schematische weergave testopstelling

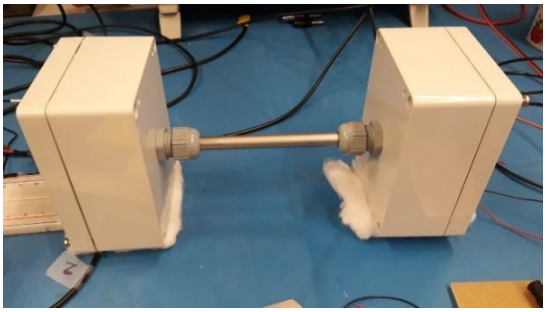


Figuur 4: testopstelling pvc-buis

Met de testopstelling weergegeven in Figuur 4 en Figuur 5 wordt getest wat het effect is van de reflecties in een dunne buis.



Figuur 5: Schematische weergave testopstelling



Figuur 6: Testopstelling metalen buis

De verschillende onderdelen zullen worden gemeten:

- Vertragingstijd tussen zender en ontvanger;
- Tijd totdat de reflecties uitgedoofd zijn;
- De grootte van het ontvangen signaal.

9.1 Zender

De zender is verantwoordelijk voor het uitsturen van een geluidsgolf. Voor de zender zal een luidspreker (tweeter) worden gebruikt. De heeft tweeter die gebruikt wordt is van het merk VISATON van het type: SC-5 (Figuur 7). De diameter van de tweeter is 50 mm. De luidspreker heeft een maximaal uitgangsvermogen van 60 Watt en een impedantie van 8 Ω [8]. Het frequentiebereik van deze luidspreker ligt tussen de 4 en 22 kHz.
Er is gekozen voor een tweeter omdat deze hoge frequenties kan uitsturen. In een stapfunctie zitten theoretisch alle frequenties. Om het ingangssignaal zoveel mogelijk te reproduceren bij de ontvanger moet de tweeter een zo hoog mogelijk frequentiebereik hebben. De tweeter een vermogen van 60 W. Dit vermogen is nodig om de moleculen met zoveel mogelijk energie in beweging te brengen.

Figuur 7: luidspreker visaton SC-5

De hardware die nodig is voor het aansturen van de luidspreker staat beschreven in hoofdstuk 10.

9.2 Ontvanger

De ontvanger zet een geluidsgolf om in een elektrische spanning.

Voor de test is gebruikt gemaakt van een electret microfoon van het type: CMA-4544PF-W [9] (Figuur 8). Een groot voordeel van deze microfoon is dat deze erg goedkoop is (€ 1,42). Het Nadeel is dat er een voorspanning nodig is voor de microfoon om te werken.

Figuur 8: Electret microfoon

9.3 Meetinstrumenten

Voor de meting zijn de volgende meetinstrumenten en voedingen gebruikt:

- Oscilloscoop: Tektronixs MDO3024
- Functiegenerator: Siglent SDG2122x
- Voeding: Delta elektronika EST150

De oscilloscoop wordt gebruikt voor het zichtbaar maken van het signaal bij de ontvanger. De resolutie van de Oscilloscoop is ingesteld op 25 MHz. De functiegenerator wordt gebruikt voor het maken van de stapfunctie. Met de voeding worden de componenten van spanning voorzien.

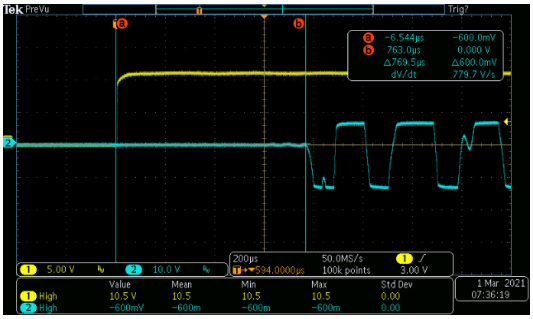
9.4 Metingen pvc-buis met electret microfoon als ontvanger (lucht)

De metingen zijn uitgevoerd volgens Figuur 3. De meetwaardes zijn weergegeven in Tabel 4. Voor de zender wordt een tweeter gebruikt. De electret microfoon wordt als ontvanger gebruikt. De afstand tussen de zender en ontvanger gevarieerd om te zien wat voor invloed dit heeft op de reflecties.

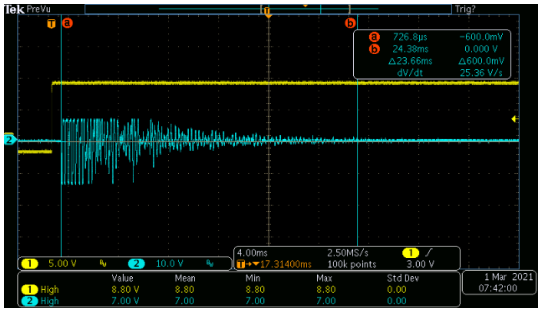
Tabel 4: Tweeter als zender en electret microfoon als ontvanger

Afstand (cm)	Vertragingstijd (µs)	Reflecties (ms)	Spanning (V_{in})	Spanning (V_{out})	Verzwakking (V_{out}/V_{in})
25	769,5	23,66	8,6	6,2	0,72
20	624,8	16,68	8,6	6,2	0,72
15	471,4	16,96	8,6	6,2	0,72
10	326,9	12,30	8,6	6,2	0,72

Wanneer de afstand tussen de zender en ontvanger kleiner wordt, neemt de versterking niet toe. Wel wordt de vertragingstijd die wordt veroorzaakt door de reflecties kleiner. In Figuur 7 is de tijdsvertraging weergegeven tussen de zender en ontvanger. In Figuur 8 is de reflectie van het ingang signaal weergegeven.



Figuur 9: Tijdsvertraging tussen in- uitgang



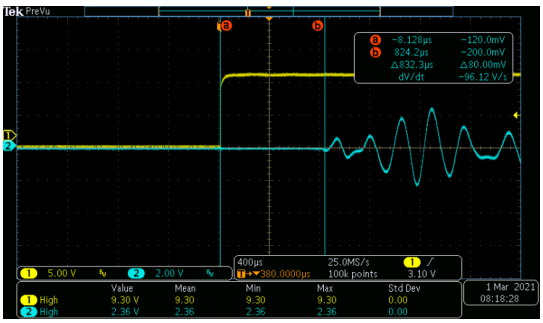
Figuur 10: Reflecties

Metingen in pvc-buis met piëzo als ontvanger (lucht)

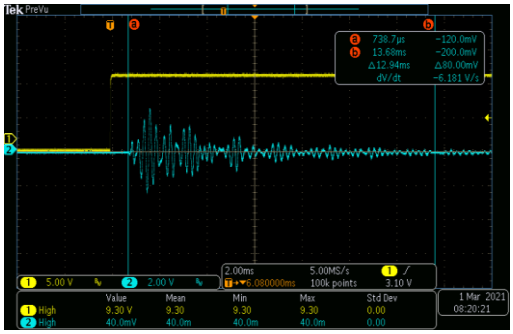
In Tabel 5 is een overzicht van de meetgegevens met daarin de tweeter als zender en een piëzo als ontvanger. Het ontvangen signaal uit de piëzo is erg zwak. Het gebruik van een piëzo geeft vermindering in de reflecties.

Tabel 5: meetgegevens tweeter als zender piëzo als ontvanger

Afstand (cm)	Vertragingstijd (µs)	Reflecties (ms)	Zender (V_{in})	ontvanger (V_{out})	Versterking (V_{out}/V_{in})
25	832,3	12,94	9,5	2,56	0,26
20	680,1	12,91	9,6	1,8	0,19
15	534,9	10,47	9,3	1,12	0,12
10	432,4	6,70	9,1	1,12	0,12



Figuur 12: Tijdsvertraging piëzo als ontvanger



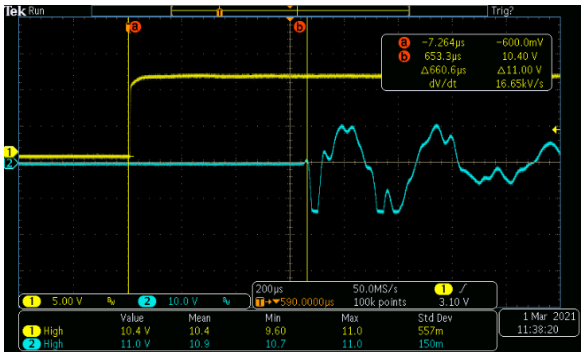
Figuur 11: Reflecties

9.5 Metingen tweeter als ontvanger (lucht)

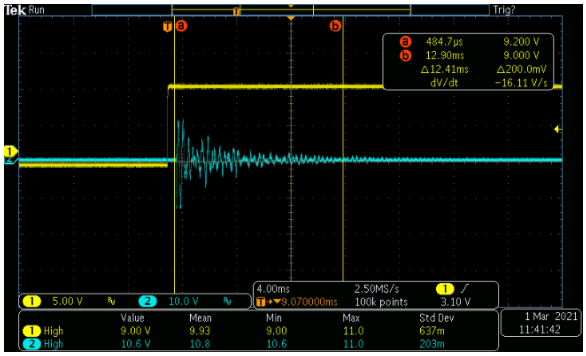
In Tabel 6 zijn de meetgegevens weergegeven waarin een tweeter voor zowel de zender als ontvanger fungeert. De tweeters zijn beide van hetzelfde merk en type. Het gebruikt van een tweeter voor zowel zender als ontvanger zorgt voor vermindering van de reflecties. Dit komt doordat de conus van de tweeter werkt als natuurlijke demper.

Tabel 6: Meetgegevens tweeter als zender en een tweeter als ontvanger

Afstand (cm)	Vertragingstijd (µs)	Reflecties (ms)	Zender (V_{in})	ontvanger (V_{out})	Versterking (V_{out}/V_{in})
25	813,7	6,17	9,9	10	1,01
20	699,7	6,40	10,1	10,8	1,06
15	556,5	5,99	10,7	9,6	1,11
10	388,2	5,54	9,9	11,60	1,17



Figuur 13 : Tijdsvertraging tussen zender en ontvanger



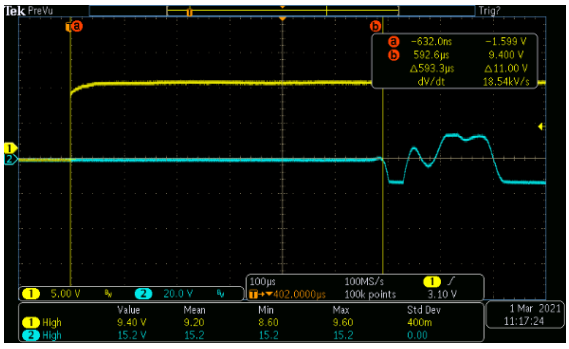
Figuur 14: Reflecties

9.6 Metingen metalen buis

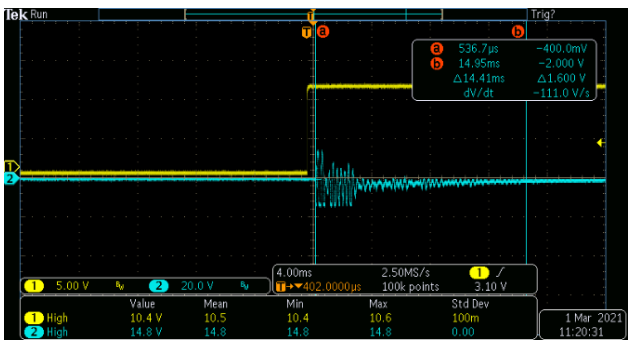
De metingen uit Tabel 7 zijn uitgevoerd volgens Figuur 6. Hierbij is een aluminium pijp gebruikt die zich tussen de zender en ontvanger bevindt. Wanneer een dunne metalen pijp wordt gebruikt, wordt het ontvangen signaal groter ten opzichte van het ingangssignaal. De reflecties die ontstaan worden niet minder met het gebruik van een dunne pijp.

Tabel 7: metingen met metalen pijp

Afstand (cm)	Vertragingstijd (µs)	Reflecties (ms)	Zender (V_{in})	ontvanger (V_{out})	Versterking (V_{out}/V_{in})
20	593,3	14,41	9,4	12,8	1,36



Figuur 16: Tijdsvertraging tussen zender en ontvanger



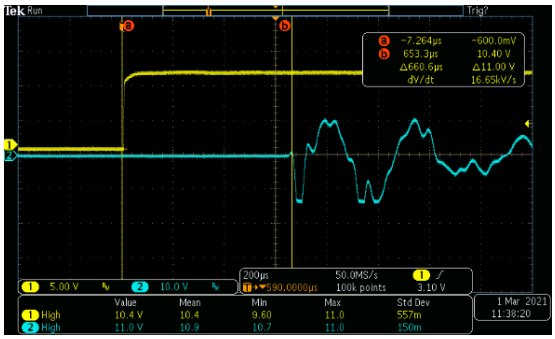
Figuur 15: Reflecties

9.7 Metingen met een rubberen buis

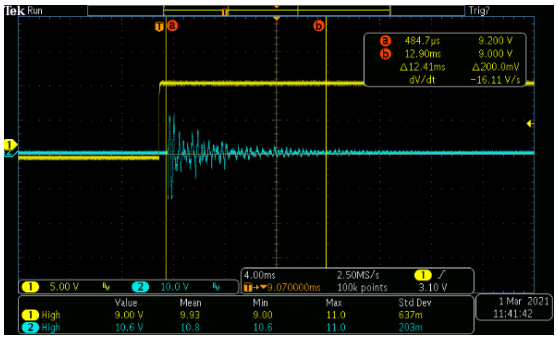
In Tabel 8 zijn de meetgegevens met dezelfde opstelling als in Figuur 2. De metalen buis vervangen door een van rubber. Met het gebruik van een rubberen buis wordt het ontvangen signaal kleiner en worden de reflecties sneller gedempt.

Tabel 8: metingen met rubberen buis

Afstand (cm)	Vertragingstijd (µs)	Reflecties (ms)	Zender (V_{in})	ontvanger (V_{out})	Versterking (V_{out}/V_{in})
20	818,6	11,25	10,6	10,4	0,98



Figuur 18: Tijdsvertraging tussen zender en ontvanger



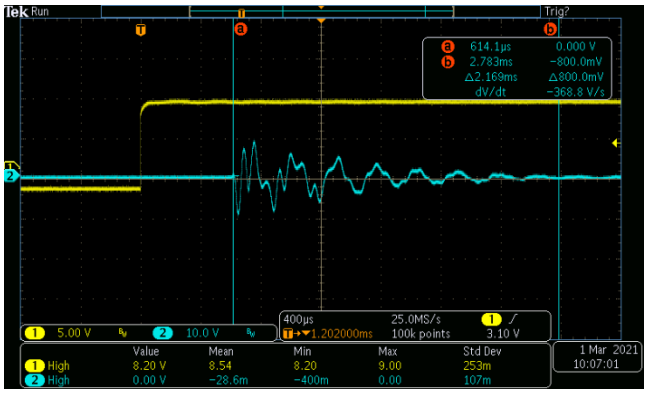
Figuur 17: Reflecties

9.8 Verminderen van reflecties met watten

De metingen uit Tabel 9 zijn uitgevoerd als zoals in Figuur 5. Nu zijn er watten toegevoegd om de reflecties te dempen. De watten geven veel demping op de reflecties. De reflecties zijn uitgedoofd na 2,17 ms.

Tabel 9: Metalen pijp met demping

Afstand (cm)	Vertragingstijd (µs)	Reflecties (ms)	Zender (V_{in})	ontvanger (V_{out})	Versterking (V_{out}/V_{in})
20	613,3	2,17	8,2	8,4	1.02



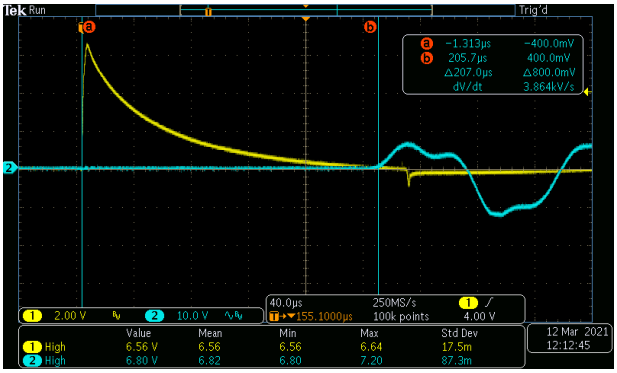
Figuur 19: Reflecties met watten als isolatie

9.9 Metingen in Helium

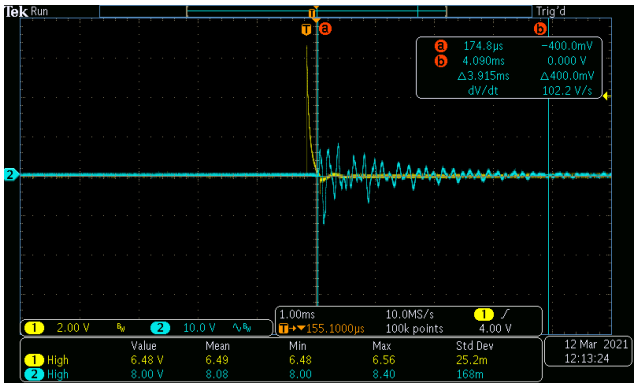
Voor de testen in Helium is de behuizing uit Figuur 5 gebruikt. Dit is zodat de ontvanger zoveel mogelijk signaal ontvangt. Volgens Tabel 2 is de tijd die nodig is in Helium (over een lengte van 20 cm) om van de zender naar de ontvanger te gaan 207,25 µs zijn. Het ontvangen signaal in Helium is kleiner dan in lucht. Dit komt omdat Helium een dun gas is.

Tabel 10: Metingen in Helium

Afstand (cm)	Vertragingstijd (µs)	Reflecties (ms)	Zender (V_{in})	ontvanger (V_{out})	Versterking (V_{out}/V_{in})
20	207,0	3,92	6,80	6,56	0,965



Figuur 20: Tijdsvertraging tussen zender en ontvanger



Figuur 21: Reflecties

9.10 Conclusie

Verschillende metingen zijn uitgevoerd om te zien wat de effecten zijn van de geluidssnelheid meten. De volgende bevindingen gedaan:

Het gebruik van een buis met een kleinere diameter heeft weinig effect op de reflecties. Wanneer de buis dezelfde diameter heeft als de ontvanger geeft dit een versterking bij de ontvanger. Dit is gewenst omdat in het Helium het ontvangen signaal zwakker is dan in lucht. Om deze reden zal de constructie uit Figuur 6 gebruikt worden in het eindontwerp. Een ander voordeel is dat een metalen buis te buigen is, zodat de behuizing compacter gemaakt kan worden.

Wanneer de tweeter voor zowel zender als ontvanger gebruikt wordt zorgt dit dat de reflecties sneller gedempt worden. De reden dat deze weinig reflecties geeft komt doordat de conus van de tweeter een grote diameter heeft en daarmee veel trilling opvangt. Het nadeel is dat de tweeter meer kost dan een electret microfoon en zal om deze reden niet gebruikt worden als ontvanger.

De piëzo als ontvanger geeft geen effect op de reflecties. Ook is het binnenkomende signaal erg zwak. Verder kan de piëzo een piekspanning generen wanneer deze een mechanisme schok krijgt. Hierdoor kan de elektronica die gebruikt wordt voor het versterken van het signaal stuk gaan. De piëzo zal om deze reden niet gebruikt worden als ontvanger.

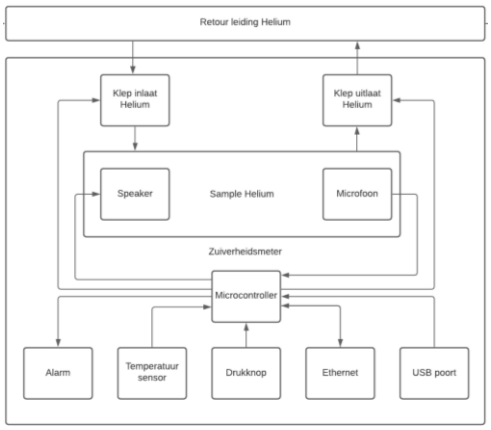
Om de reflecties zo snel mogelijk te laten uitdempen kan een dempend materiaal gebruikt worden zoals watten. De watten geven wel vervuiling in het Helium en kan daardoor niet gebruikt worden in het eindontwerp. De behuizing die voor de test gebruikt is werkt als klankkast. Een kleinere klankkast zorgt voor minder reflecties. In het eindontwerp zal daarom behuizing op de grootte van de tweeter worden aangepast.

Met het gebruik van een rubberen buis geeft dit minder reflecties maar ook het ontvangen signaal wordt kleiner. Het nadeel van een rubberen buis is dat rubber over de jaren kan vergassen wat het Helium kan vervuilen.

Omdat de opstelling uit Figuur 6 het meeste signaal geeft bij de ontvanger zal deze gebruikt worden voor de testen in Helium. De electret microfoon is goedkoop en kan makkelijk worden versterken en zal daarom gebruikt worden in het eindontwerp.

10 Algemene systeem omschrijving

In Figuur 22 is het blokschema van de Heliumzuiverheidsmeter weergegeven.



Figuur 22: Blokschema opstelling

De Heliumzuiverheidsmeter bestaat uit de volgende componenten:

- Inlaatklep Helium;
- Uitlaatklep Helium;
- Luidspreker;
- Microfoon;
- Microcontroller;
- Alarm;
- Temperatuur sensor;
- Drukknop;
- Ethernet aansluiting.

De werking van de zuiverheidsmeter is als volgt:

De kleppen voor het in- en uitlaat Helium gaan open waardoor een sample uit de retourleiding wordt genomen. De kleppen sluiten waarna de meting gestart kan worden. Met een tweeter wordt een geluidspuls geproduceerd. Op het moment dat de puls wordt uitgestuurd, wordt een timer gestart. Wanneer de microfoon de geluidspuls ontvangt wordt timer gestopt en kan de zuiverheid van het Helium worden bepaald. Om een grotere nauwkeurigheid te krijgen worden 100 metingen gedaan. Van de 100 metingen wordt daarna het gemiddelde berekend. Dit wordt gedaan om de kans op statistische uitschieters (foute metingen) tegen te gaan. Na de metingen worden de kleppen geopend en kan een nieuw sample genomen worden.

Het meten van de tijd wordt gedaan met een microcontroller. De door de microcontroller berekende waarden voor de heliumzuiverheid worden gestuurd naar een server die de data verwerkt. Wanneer het Helium te veel vervuild is, stuurt de server een commando terug dat er een alarm gegeven moet worden. Met een USB-kabel kan het IP-adres van de microcontroller, IP-adres van server en het poortnummer van de server worden ingesteld. Met de temperatuursensor wordt de temperatuur van het Helium gemeten. De temperatuur wordt in Kelvin (K) verstuurd naar de server.

Het 7-sigments display en de drukknop voor op de behuizing zijn bedoeld voor de gebruiker. Met de drukknop kan het alarm worden uitgeschakeld. Met het 7-sigmentsdisplay worden de zuiverheid van het Helium en eventuele foutmeldingen weergegeven.

10.1 Manier van meten

De betrouwbaarheid van de meting hangt af van waarop gemeten wordt. In Tabel 11 zijn de verschillende manieren van meten opgenomen. Over het aantal metingen is met Excel de standaarddeviatie berekend. De manier van meten met de kleinste standaardafwijking wordt gebruikt in het eindontwerp. In hoofdstuk 15 wordt hier verder op ingegaan.

Tabel 11: Verschillende manier van sample meten

	100 metingen in 10 sec	100 metingen in 2 min	10 metingen in 10 sec	10 metingen in 2 min
standaarddeviatie	232 ns	753 ns	1,20 us	1,64 us

Met nemen van 100 metingen in 10 seconden geeft de minste spreiding. Dit komt omdat statistische uitschieters op deze manier er zo veel mogelijk worden uitgefilterd. De korte tijd is zodat het Helium niet kan weg lekken. Deze manier van meten zal gebruikt worden in het eindontwerp.

Bij de bespreking hiervan (hoofdstuk 15) zal ook blijken dat op basis van deze spreiding kan worden geconcludeerd dat de ontwerpdoelstelling ruimschoots wordt gehaald.

10.2 1e gebruik

Wanneer de Heliumzuiverheidsmeter voor het eerst in gebruik wordt genomen zal deze eerst moeten worden ingesteld. De onderstaande stappen moeten doorlopen worden voor het instellen van de Heliumzuiverheidsmeter:

1. Sluit de voeding aan;
2. Sluit de USB aan;
3. Sluit de Ethernet kabel aan;
4. Stel het IP-adres van de microcontroller in;
5. Stel het MAC-adres van de microcontroller in;
6. Laat de Heliumzuiverheidsmeter zich aanmelden bij het netwerk (hiervoor moet het MAC-adres worden opgegeven);
7. Stel het IP-adres van de server in;
8. Stel de poort van de server in;
9. Reboot de Heliumzuiverheidsmeter;
10. Sluit de Heliumzuiverheidsmeter aan bij de opstelling in het Cryogeen;
11. De meter is klaar voor gebruik.

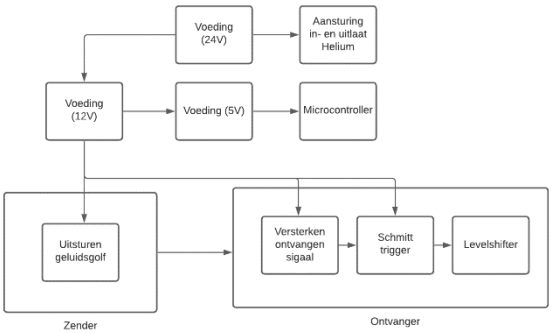
Na het instellen van de Heliumzuiverheidsmeter kan de USB losgekoppeld worden.

11 Hardware analoog

De hardware bestaat uit verschillende onderdelen. Het blokschema van de hardware is weergeven in Figuur 23 en bevat de volgende functies:

-Het voeden van de componenten (voeding);
-Aansturing klep Helium inlaat;
-Aansturing klep Helium uitlaat;
-Aansturing voor uitzenden geluidsgolf (zender);
-Versterken en filteren van ontvangen geluidsgolf;
-*triggeren* op ontvangen geluidsgolf (schmitt trigger);
-Signaal uit ontvanger (microfoon) leesbaar maken voor microcontroller (*Levelshifter*);

Het volledig elektrische schema is weergegeven in bijlage 1.



Figuur 23: Blokschema hardware

De kleppen voor het in- en uitlaten Helium worden gevoed met een spanning van 24 V. Voor het maken van de geluidsgolf, versterken van het ontvanger signaal en de schmitt trigger wordt een voeding gebruikt van 12 V. De microcontroller zal gevoed worden met een spanning van 5 V. De PCB van de microcontroller heeft een eigen spanningsregelaar die van 5 V, 3,3 V maakt.

11.1 Voeding

De testopstelling (Heliumzuiverheidsmeter) wordt gevoed met een spanning van 24 V. De 24 V wordt gemaakt door een schakelende voeding die van 230 V aan de ingang, een uitgangsspanning maakt van 24 V. De schakelende voeding is van het type: LH60A24-P1J en kan een stroom leveren van 2,5 A. Omdat het gaat om een schakelende voeding zijn er op de pcb condensatoren toegevoegd voor het onderdrukken van de schakelruis van de voeding (*decoupling* condensator). Eén condensator van 100 µF en een condensator van 10 µF. De condensatoren worden op de pcb direct na de ingang geplaatst. Met een rode LED wordt aangegeven of de voeding is aangesloten. Met formule (11.1.1) is de weerstand voor de LED berekend.

(11.1.1) $$R_{LED} = \frac{U_{voeding} - U_{LED}}{I_{LED}} = \frac{24-2,6}{0,005} = 4280\ \Omega$$

De 12 V wordt gemaakt met de LM7812. Dit is een lineaire spanningsregelaar die van een ingangsspanning tussen de 14,5 V en 30 V een uitgangsspanning maakt van 12 V. De nominale stroom die de spanningsregulator kan leveren is 1 A. Met deze regulator wordt de schakeling gevoed voor het uitsturen van de geluidsgolf. Het vermogen dat de 12 V-regulator moet leveren is het vermogen van de tweeter en alle digitale componenten. Omdat lineaire spanningsregelaars niet schakelen geeft dit weinig ruis voor de rest van de schakeling.

De tweeter heeft een stroom nodig van 1,5 A, voor 300 µs. Als iedere 10 ms een meting wordt gedaan is het gemiddeld opgenomen vermogen: 0,54 W. Door het gebruik van een grote condensator aan de uitgang van de regulator, kan een piekstroom worden geleverd die groter is dan de maximale uitgangsstroom van de regulator

De 5 V wordt gemaakt door een lineaire spanningsregulator van het type: LM7805. De regelaar maakt van een ingangsspanning tussen de 7 V en 25 V een uitgangspanning van 5 V. Met deze spanning wordt de microcontroller gevoed.

De interne spanningsregulator van de microcontroller kan maximaal 250 mA bij 3,3 V leveren. Wanneer de microcontroller volledig belast wordt zal de regulator een vermogen moeten leveren van: 0,825 W.

11.1.1 Heatsink

Het nadeel van de LM7812 en LM7805 is dat deze spanningsregelaars niet efficiënt de spanning kunnen transformeren. Hoe groter het verschil tussen de in- en uitgangsspanning en hoe groter de stroom hoe meer de regulator opwarmt. Het maximale vermogen dat de LM7812 en LM7805 kunnen dissiperen zonder dat deze te heet worden, is berekend met formule (11.1.1.1).

(11.1.1.1) $$P_{max} = \frac{T_{JMAX} - T_A}{\theta_{JA}} = \frac{125-25}{54} = 1{,}85\ W$$

T_{JMAX}: maximale temperatuur [$°C$]
T_a: Omgevingstemperatuur [$°C$]
θ_{JA}: thermische weerstand [°C/W]

Met formule (11.1.1.2) is het gedissipeerde vermogen van de LM7812 spanningsregulator berekend. De stroom I_{in} is gelijk aan de stroom $I_{out.}$ De stroom door de LM7812 is de stroom van de tweeter (45 mA) en de stroom van de microcontroller (250 mA). De totale stroom is dus 0,295 A.

(11.1.1.2) $$P = (V_{in} - V_{out}) \cdot I = (24 - 12) \cdot 0{,}295 = 2{,}065\ W$$

Met formule (11.1.1.3) is de temperatuur van de regulator berekend wanneer er geen gebruik wordt gemaakt van een *heatsink*.

(11.1.1.3) $$T_{junction} = P_{max} \cdot \theta_{JA} + T_A = 2{,}065 \cdot 54 + 25 = 136{,}5\ °C$$

$T_{Juntion}$ Is de temperatuur die de regulator theoretisch zal bereiken en mag niet boven het maximum van het component uitkomen. Bij dit component is het maximum 125 graden dus is er een *heatsink* nodig [10]. De maximale temperatuur ($T_{Juntion}$) is: 125 °C. Voor de omgevingstemperatuur (TA) is 25 °C gekozen. Dit is omdat de elektronica in een behuizing wordt geplaatst en de warmte dan minder goed kan worden afgestaan aan de omgeving. De thermische weerstand (θ_{JA}) van de behuizing naar de omgeving wordt gegeven door de datasheet en is voor de TO-220 behuizing 54 °C/W. Wanneer er geen *heatsink* gebruikt wordt en de schakeling vol belast wordt zal de temperatuur van de regulator oplopen tot 136,5 °C.

Figuur 24: Heatsink spanningsregulator

Omdat de temperatuur van de regulator te hoog wordt, zal gebruik gemaakt worden van een *heatsink*. De *heatsink* is van het merk: Aavid. De gebruikte *heatsink* is weergeven in Figuur 24. De thermische weerstand (θ_{SA}) van de heatsink is gegeven door het datasheet en is 24 °C/W. De thermische weerstand (θ_{CS}) is de weerstand tussen de behuizing van TO-220 en de *heatsink*. De thermische weerstand (θ_{CS}) is: 2 °C/W. De Thermische weerstand (θ_{JC}) wordt gegeven door de datasheet van de spanningsregulator en is 4 °C/W. Met formule (11.1.1.4) is de interne temperatuur van de regulator berekend.

(11.1.1.4) $$T_{junction\ met\ heatsink} = (\theta_{JC} + \theta_{CS} + \theta_{SA}) \cdot P_{max} + T_A = (4 + 2 + 24) \cdot 2{,}065 + 25 = 86{,}95\ °C$$

De temperatuur van de regulator (LM7812) onder vollast met *heatsink* is 86,95 °C.

Het opgenomen vermogen van de 5 V-regulator (LM7805) is bepaald met formule (11.1.1.5). Het maximaal opgenomen vermogen van de regulator is 1,75 W.

(11.1.1.5) $$P = (V_{in} - V_{out}) \cdot I = (12 - 5) \cdot 0{,}250 = 1{,}75\ W$$

Met formule (11.1.1.6) is de temperatuur ($T_{junction}$) van de 5 V-regulator berekend.

(11.1.1.6) $$T_{junction} = P_{max} \cdot \theta_{JA} + T_A = 1{,}75 \cdot 54 + 25 = 119{,}5\ °C$$

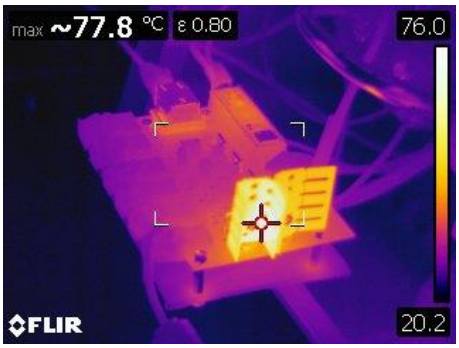
De temperatuur voor de 5 V-regulator is: 119,5 °C. De toegestane maximale temperatuur van de regulator (Tjunction) is 125 °C. Voor de 5 V-regulator is strikt genomen geen heatsink nodig. Maar omdat de temperatuur erg hoog wordt zal dezelfde *heatsink* worden toegepast als voor de LM7812. Met formule (11.1.1.7) is de temperatuur van de LM7805 berekend.

(11.1.1.7) $$T_{junction\ met\ heatsink} = (\theta_{JC} + \theta_{CS} + \theta_{SA}) \cdot P_{max} + T_A = (4 + 2 + 24) \cdot 1{,}75 + 25 = 77{,}5\ °C$$

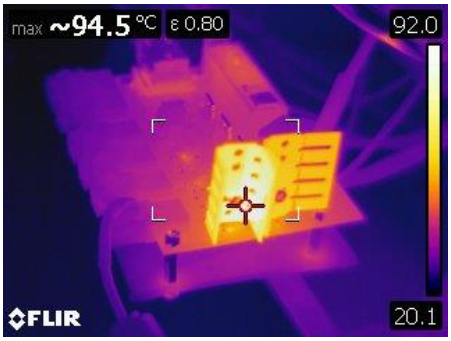
Met *heatsink* wordt temperatuur van de LM7805: $77{,}5\ °C$.

11.2 Thermische meting

Met de thermische meting wordt bekeken of componenten niet te heet worden. In Figuur 25 is temperatuur van de LM7812 weergegeven. De schakeling is in rust als er niks wordt geschakeld. In Figuur 26 is de thermische meting onder vollast weergegeven.



Figuur 25: Temperatuur in rust



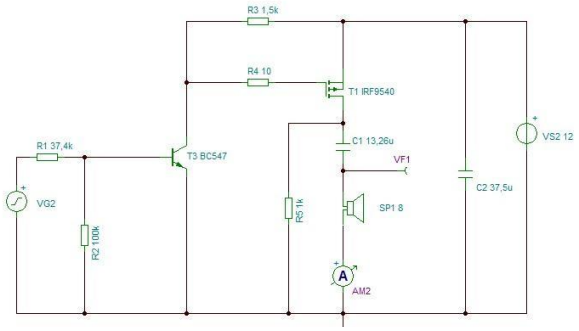
Figuur 26: Tempratuur in vollast

11.3 Beveiliging

De hardware is beschermd tegen overstroom en tegen het omdraaien van de voedingsspanning. Door gebruik te maken van een glaszekering wordt de schakeling beschermd tegen overstroom. Wanneer door een fout de gevraagde stroom te groot wordt, zal de zekering doorbranden. De grootte van de zekering is 1 A. Om de schakeling te beschermen tegen het omdraaien van de voedingsspanning wordt gebruik gemaakt van een diode van het type 1n4007. De diode heeft een spanningsval van 0,6 V. Wanneer de schakeling vol belast wordt, neemt de diode een vermogen op van 0,18 W op.

11.4 Aansturing voor uitzenden geluidsgolf zender

Voor het maken van een geluidsgolf zal een stap gemaakt worden. De schakeling voor het maken van de stap is weergegeven in Figuur 27. Het doel van de schakeling is om met een microcontroller die een spanning kan uitsturen van 3,3 V, een groter vermogen (tweeter) te schakelen.



Figuur 27: elektrische schema aansturen geluidsgolf

De transistor is van het type: BC849 [11]. Met formule (14.4.1) is de collector stroom berekend.

(11.4.1) $$I_c = \frac{U_{voeding}}{R3} = \frac{12}{1500} = 8\ mA$$

De collector stroom is 8 mA. De weerstanden die voor de pcb gebruikt worden zijn SMD-weerstanden (0805). Het maximale vermogen van dit type is 0,125 W. Het vermogen dat door weerstand R_3 wordt gedissipeerd is berekend met formule (11.4.2).

(11.4.2) $$P = I^2 \cdot R = (8 \cdot 10^{-3})^2 \cdot 1500 = 96\ mW$$

De weerstand R_3 dissipeert een vermogen van 96 mW. Dit is kleiner dan 125 mW deze weerstandswaarde volstaat. De weerstand zal niet doorbranden.

Met formule (11.4.3) is de basisstroom berekend.

(11.4.3) $$I_b = \frac{I_c}{\beta} = \frac{8\ mA}{200} = 40\ \mu A$$

De basisstroom moet 40 µA zijn om de transistor te schakelen. De versterkingsfactor (HFE) wordt door het datasheet gegeven en is 200 [11]. De weerstand R_{basis} is berekend met formule (11.4.4).

(11.4.4) $$R_{basis} = \frac{U_{microcontroller} - U_{BE}}{I_b} = \frac{3{,}3 - 0{,}7}{40\ \mu A} = 65\ k\Omega$$

De weerstand R_{basis} is 65 k. De weerstand R_2 is om de transistor naar de ground te “trekken”. Dit voorkomt dat de transistor gaat “zweven” als de basis niet is aangesloten. Als gevolg van lekstromen zou de transistor ongewild in geleiding kunnen gaan.

Om de tweeter te schakelen wordt gebruik gemaakt van een P-channel MOSFET (IRF9540). Deze MOSFET heeft een kleine Q_{GS} (61 nC) en kan een vermogen van 50 W sturen. Een kleine Q_{GS} zorgt dat de MOSFET snel geschakeld kan worden. De tweeter heeft een vermogen van 25 W. De MOSFET is daarom geschikt voor het schakelen van deze belasting. De weestand R_4 heeft een waarde van 10 Ω. Deze weerstand is voor het uitdempen van oscillaties geproduceerd door de MOSFET.

De condensator C_1 heeft twee functies: Het voorkomen van een dc-spanning door de tweeter en het is een filter. Met formule (11.4.5) is de waarde van de condensator berekend.

(11.4.5) $$C_1 = \frac{1}{2\pi \cdot Z \cdot F_c} = \frac{1}{2\pi \cdot 8 \cdot 1500} = 13{,}26\ \mu F$$

Voor de weerstand is de impedantie (Z) van de tweeter gebruikt (8 Ω). Het filter is een hoogdoorlaatfilter (HPF) en berekend op de frequentieresponse minimaal toegestane frequentie van de tweeter (1500 Hz). De waarde van de condensator is: 13,26 µF. De weerstand R_5 is voor het ontladen van de condensator.

De tweeter heeft een vermogen van 25 W en wordt aangesloten op een spanning van 12 V. De stroom die de tweeter nodig heeft is berekend met formule (11.4.6).

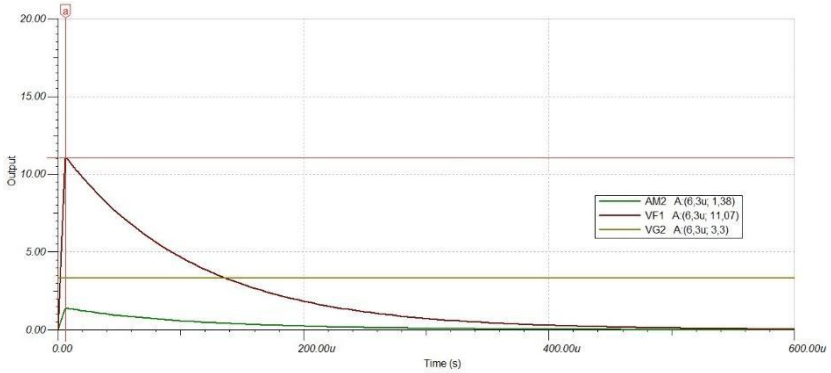
(11.4.6) $$I_{tweeter} = \frac{U}{Z_{tweeter}} = \frac{12}{8} = 1{,}5\ A$$

De stroom door de tweeter is 1,5 A. Om te zorgen dat niet alle stroom uit de voeding (LM7812) hoeft te komen is de condensator C_2 als buffer toegevoegd. De waarde voor de buffer condensator (C_1) heeft een waarde van 100 µF.

Tabel 12: Component waarden aansturing tweeter

Component:	Berekende waarde:
R1	24,23 kΩ
R2	10 kΩ
R3	1 kΩ
R4	10 Ω
R5	10 KΩ
C1	13,26 µF
C2	100 µF

De simulaties zijn uitgevoerd met het simulatieprogramma 'Tina-TI'. De simulatie is uitgevoerd volgens Figuur 27 en Figuur 28. De simulatie is weergegeven in Figuur 28. De voedingsbron (VG2) stelt de microcontroller voor die een spanning (stap) uitstuurt van 3,3 V. VS2 is een 12 V-voedingsbron waarmee de schakeling gevoed wordt. De stroommeter (AM2) geeft de stroom door de tweeter weer. De stroom I_{piek} = 1,38 A. De meter (VF1) geeft de spanning over de tweeter aan. Op de piek is spanning over de tweeter: 11,07 V.



Figuur 28: Simulatie aansturing tweeter

11.5 Aansturing klep Helium in- en uitlaat

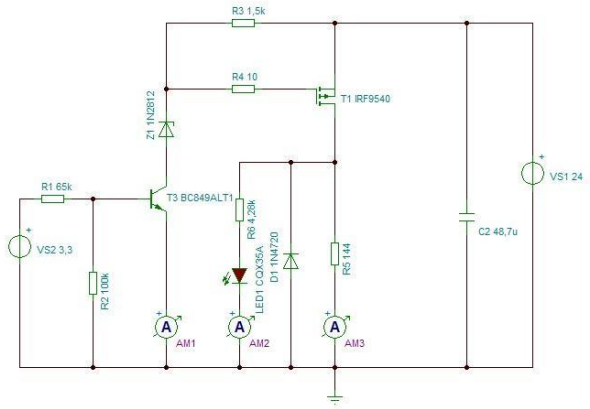
Met het gebruik van solenoïde kleppen wordt het Helium binnen de testopstelling gebracht. Eén klep in voor het inlaten van het Helium. Een tweede klep is aan de andere kant van de

opstelling en zorgt dat het Helium niet terug in het proces kan lopen. Het schema voor het aansturen van de solenoïde kleppen is weergegeven in Figuur 29.

De solenoïde klep werkt op een spanning van 24 V. De solenoïde klep is van het type: 6011A en heeft een vermogen van 4 W. De stroom die nodig is om een klep te schakelen is berekend met formule (11.5.1)

(11.5.1) $$I = \frac{P}{U} = \frac{4}{24} = 167\ mA$$

De stroom voor het opensturen van de klep is 167 mA. De kleppen zijn "*normally closed*" (N.C.). Dit betekent dat wanneer er spanning op de klep wordt gezet, de klep opengaat en het Helium kan door doorstromen. De tijd die nodig is voor de klep om open te gaan ligt mechanisch vast en is tussen de 7 en 10 ms. De tijd die nodig is voor de klep om te sluiten is tussen de 10 en 15 ms [12].



Figuur 29: elektrische schema aansturing klep

De functie van de hardware is om met een spanning van 3,3 V, een spanning van 24 V schakelen. De schakeling voor het aansturen van de in- en uitlaat kleppen voor het Helium is weergegeven in Figuur 25.

De spanningsbron (VG1) stelt de microcontroller voor met een spanning van 3,3 V. De spanningsbron (VS1) levert de spanning voor de schakeling en is 24 V. De collector stroom (I_c) is berekend met formule (11.5.2).

(11.5.2) $$I_c = \frac{U_{voeding} - U_{zener}}{R3} = \frac{24-12}{1500} = 8\ mA$$

De collectorstroom is 8 mA. Met (formule 11.5.3) is de benodigde basisstroom berekend.

(11.5.3) $$I_b = \frac{I_c}{\beta} = \frac{8\ mA}{200} = 40\ \mu A$$

De basisstroom (I_b) is: 40 µA. De versterkingsfactor (H_{FE}) wordt door het datasheet gegeven en is 200 [15]. De basisweerstand is berekend met formule (11.5.4).

(11.5.4) $$R_{basis} = \frac{U_{microcontroller} - U_{BE}}{I_b} = \frac{3{,}3-0{,}7}{40\ \mu A} = 65\ k\Omega$$

De basisweerstand (R_1) is 65 kΩ. De weerstand R_2 is om te zorgen dat transistor niet gaat 'zweven' en ongewenst gedrag kan vertonen.

De zenerdiode (Z_1) is ter bescherming van de MOSFET. De maximale spanning (V_{GS}) van de IRF9540 is 20 V. Omdat de voedingsspanning 24 V is zal de MOSFET-defect raken. De zenerdiode zorgt ervoor dat de spanning (V_{GS}) niet groter wordt dan 12 V.

Voor het schakelen van de solenoïde klep wordt gebruik gemaakt van een *p-channel* MOSFET van het type: IRF9540 [13]. Deze MOSFET kan zowel de voedingsspanning als een vermogen schakelen van 50 W.

Met een groene led (LED1) wordt aangegeven of de klep geschakeld is. Met formule (11.5.5) is de weerstandswaarde voor R_6 berekend.

(11.5.5) $$R_6 = \frac{U_{voeding} - U_{LED}}{I_{LED}} = \frac{24-2,6}{5\ mA} = 4,28\ k\Omega$$

De *forward* spanning voor de groene led is: 2,6 V met een stroom (I_f) van 5 mA [14]. De waarde voor R_6 is 4,28 kΩ.

De condensator C_2 is fungeert als buffer bij het schakelen van de solenoïde klep. De waarde voor de condensator is 100 µF.

Omdat de solenoïde klep bestaat uit een spoel, is een diode (*flybackdiode*) toegevoegd voor het tegengaan van piekspanningen. Wanneer dit niet gedaan wordt, kan de '*drain*' van de MOSFET-defect raken als deze uitschakelt.

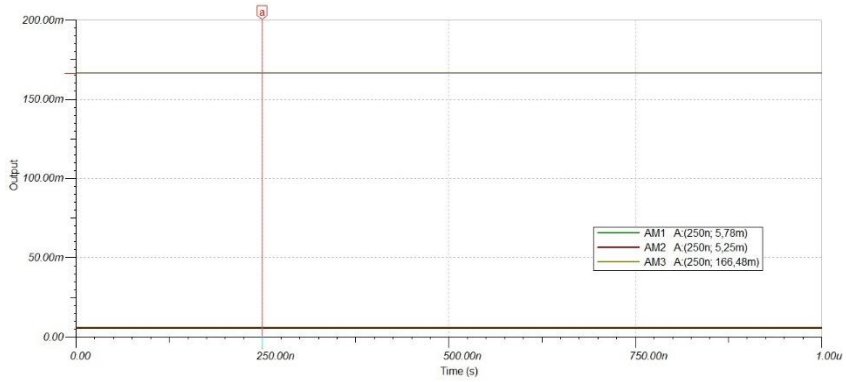
De componenten voor het schakelen van de solenoïde klep zijn gegeven in Tabel 13.

Tabel 13: componenten aansturen in- en uitlaat kleppen

Component	Berekende waarde:
R1	37,1 kΩ
R2	100 kΩ
R3	1,5 kΩ
R4	10 Ω
R5	115 Ω
R6	4,28 KΩ
C1	100 µF

In Figuur 30 is de simulatie voor de aansturing van de solenoïdeklep weergeven. De stroommeter (AM1) is de stroom door de collector en de basis. De meter (AM2) is de stroom door de LED. De meter (AM3) is de stroom door de belasting.

AM1: 5,78 mA
AM2: 5,25 mA
AM3: 166,48 mA



Figuur 30: Simulatie aansturing solenoïdeklep

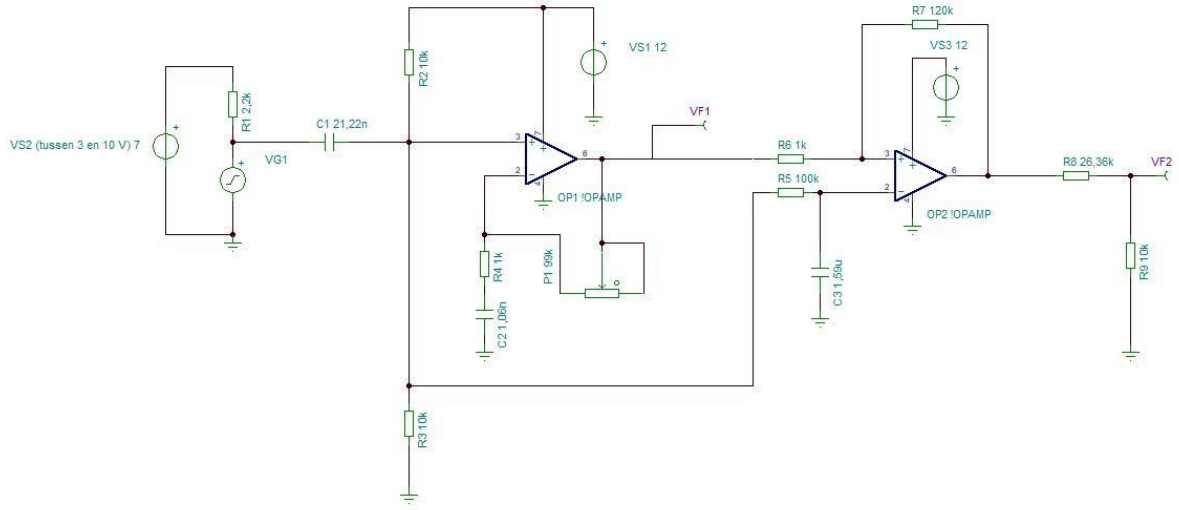
11.6 Ontvanger

De schakeling voor het ontvangen van het geluidssignaal bestaat uit drie delen en is weergegeven in Figuur 22. De drie onderdelen zijn:

- Versterken van ontvangen signaal;
- Triggeren op ingangssignaal;
- Het uitgangssignaal leesbaar maken voor de microcontroller.

De microfoon is een electret microfoon van het type: CMA-4544PF-W [9]. De electret microfoon bestaat uit een condensator. De condensator heeft één vaste en één bewegende plaat. Als de condensator platen ten opzichte van elkaar bewegen, zorgt dit voor een verandering in de spanning. De spanning uit de condensator is klein en zal versterkt worden. Het nadeel van de electret microfoon is dat de condensator continu onder spanning moet staan. In het datasheet staat gegeven dat de microfoon gevoed moet worden met een spanning tussen de 1 en 10 V. Voordeel van de electret microfoon is het relatief lichte (en daardoor snelle) membraan.

Het doel van de schakeling is het zwakke signaal uit de microfoon detecteerbaar maken voor de microcontroller. De schakeling is weergegeven in Figuur 31.



Figuur 31: Schakeling ontvanger microfoon

De spanningsbron (VG2) is de spanning die continu op de microfoon wordt gezet. De spanning op de microfoon is ingesteld op 7 V. De spanningsbron (VG1) stelt de microfoon voor die een wisselspanning (sinus) genereert tussen de 0 en 100 mVpp. De weerstand (R1) wordt gegeven door het datasheet en is 2,2 kΩ.

De condensator C_1 heeft twee functies: het tegenhouden van dc-spanningen en filteren van het ingangssignaal. Voor de meting zijn de hoge frequenties van belang omdat deze de flank van de ontvangen puls vormen. Lagere frequenties, bijvoorbeeld opgevangen omgevingstrillingen mogen de meting niet verstoren.

Doordat de electret microfoon een continue spanning krijgt betekent dit dat het signaal uit de spanningsbron (VG1) een offset heeft van 7 V. De condensator C_1 zorgt dat alleen de spanning van de microfoon bij de opamp komt. C_1 vormt samen met de weerstanden R_2 en R_3 een hoogdoorlaatfilter. Omdat de tweeter een frequentieresponsie heeft tussen de 1,5 en 22 kHz, is het hoogdoorlaatfilter ingesteld op 1,5 kHz. Hiermee worden ook gelijk de laagfrequente ruis signalen eruit gefilterd. Met formule (11.6.1) is de waarde van condensator C_1 berekend.

(11.6.1) $$C_1 = \frac{1}{2\pi \cdot (\frac{R_2 \cdot R_3}{R_2 + R_3}) \cdot F_C} = \frac{1}{2\pi \cdot (\frac{10 \cdot 10^3 \cdot 10 \cdot 10^3}{10 \cdot 10^3 + 10 \cdot 10^3}) \cdot 1500} = 21{,}22\ nF$$

De weerstanden R_2 en R_3 staan effectief parallel en hebben een weerstand van 5 kΩ. De condensator C_1 is: 21,22 nF.

Om te zorgen dat opamp met één voedingsspanning (12 V) kan werken krijgt het ingangssignaal een offset van 6 V (De helft van de voedingsspanning). De offset wordt gemaakt door de weerstanden R_2 en R_3.

De gebruikte opamp is van het type: ADA4841 [15]. Deze opamp werkt met een voedingsspanning van 12 V en heeft een bandbreedte van 80 MHz. Dit is belangrijk want hoe groter de versterking is, des te kleiner de bandbreedte wordt. Wanneer de opamp het inkomende signaal 100 keer versterkt zal de bandbreedte van de opamp 800 kHz zijn. De uitgang van de opamp is "*rail to rail*". Dit betekent dat de opamp volledig tot de voedingsspanning kan uitsturen. Door gebruik te maken van een "*rail to rail*" opamp wordt de temperatuur afhankelijkheid in de verzadigingsspanning vermeden.

De opamp is geconfigureerd als niet-inverterende versterker. Er is geen fasedraaiing tussen de in- en uitgang van de opamp. Met de weerstanden P_1 en R_4 kan de versterkingsfactor (A_v) worden ingesteld. De potmeter (P_1) heeft een waarde tussen de 0 en 99 kΩ. De versterkingsfactor is berekend met formule (11.6.2) en formule (11.6.3).

(11.6.2) $$A_{v\,(min)} = 1 + \frac{P_1}{R_4} = 1 + \frac{0}{1} = 1$$

(11.6.3) $$A_{v\,(max)} = 1 + \frac{P_1}{R_4} = 1 + \frac{99}{1} = 100$$

De versterkingsfactor kan ingesteld worden tussen de 0 en 100 ($A_{v\,(max)}$ en $A_{v\,(min)}$). De condensator C_2 zorgt dat de opamp geen dc-spanning kan versterken. Hierdoor is een voeding met een enkele spanning mogelijk. Wanneer een dc-spanning op de uitgang staat is de versterkingsfactor gelijk aan één. Naast dat het vorkomen van het versterken van dc-spanning is het ook een highpass filter. Met een hoog (F_{CH}) en laag (F_{CL}) kantelpunt. Het lage kantelpunt vormt zich met de weerstand P_1 en R_4. Met het gebruik van formule (11.6.4) is kantelpunt (F_{CL}) berekend op 1,5 kHz.

(11.6.4) $$C_2 = \frac{1}{2\pi \cdot (R_4 + P_1) \cdot F_C} = \frac{1}{2\pi \cdot (1 \cdot 10^3 + 100 \cdot 10^3) \cdot 1500} = 1{,}06\ nF$$

De condensator C_2 heeft een waarde van: 1,06 nF. De waarde voor het hoge kantelpunt is afhankelijk van de versterkingsfactor. Bij de maximale versterking geld dat F_{CH} is: 150,15 kHz.

Het tweede deel van de schakeling bestaat uit een schmitt trigger. De schmitt trigger zorgt ervoor dat wanneer het ingangssignaal opkomt, deze zo snel mogelijk gedetecteerd wordt. De schmitt trigger is ingesteld dat deze bij 0,1 V boven de ingangsspanning de uitgang omhoog gaat. Wanneer de ingang 0,1 V lager is dan de ingangsspanning zal de uitgang van de schmitt trigger weer omlaaggaan. De hysterese spanning is 0,2 V en voorkomt dat de opamp aan de uitgang gaat oscilleren. Een te grote hysterese spanning zorgt dat er niet getriggerd wordt. Een te kleine hysterese spanning zorgt voor valse triggers op bijvoorbeeld ruissignalen. Met formule (11.6.4) en formule (11.6.5) zijn de hoge en lage trigger punten van de schmitt trigger berekend.

(11.6.5) $$V_{TH} = -\frac{R_6}{R_7} \cdot V_L$$

(11.6.6) $$V_{TL} = -\frac{R_6}{R_7} \cdot V_H$$

De offset voor de schmitt trigger wordt op dezelfde ingang van de eerste opamp gezet. Dit is zodat de dc-spanning van de eerste opamp (OP1) gelijk is aan die van de schmitt trigger (OP2). Het geheel wordt zo ongevoelig gemaakt voor drift van de voedingsspanning als

gevolg van opwarming van de regulator. Om te voorkomen dat de ingangsspanning van de microfoon niet het omslagpunt van de smitt-trigger gaat bepalen, is met een weerstand (R_5) en een condensator (C_3) een laagdoorlaatfilter gemaakt. Het laagdoorlaatfilter is berekend met formule (11.6.7) Het kantelpunt F_C is gezet op 1 Hz.

(11.6.7) $$C_3 = \frac{1}{2\pi \cdot R_5 \cdot F_C} = \frac{1}{2\pi \cdot 100 \cdot 10^3 \cdot 1} = 1{,}59\ \mu F$$

De condensator C_3 heeft een waarde van: 1,59 µF.

De spanning uit de schmitt trigger is te hoog voor de microcontroller. De microcontroller kan op een digitale ingang maximaal een spanning aan van 3,3 V. De weerstanden R_8 en R_9 maken een spanningsdeler. De spanningsdeler is berekend met formule (11.6.8).

(11.6.8) $$R_8 = \frac{R_9 \cdot V_{cc}}{V_{out}} - V_9 = \frac{10 \cdot 10^3 \cdot 12}{3{,}3} - 10 \cdot 10^3 = 26{,}37\ k$$

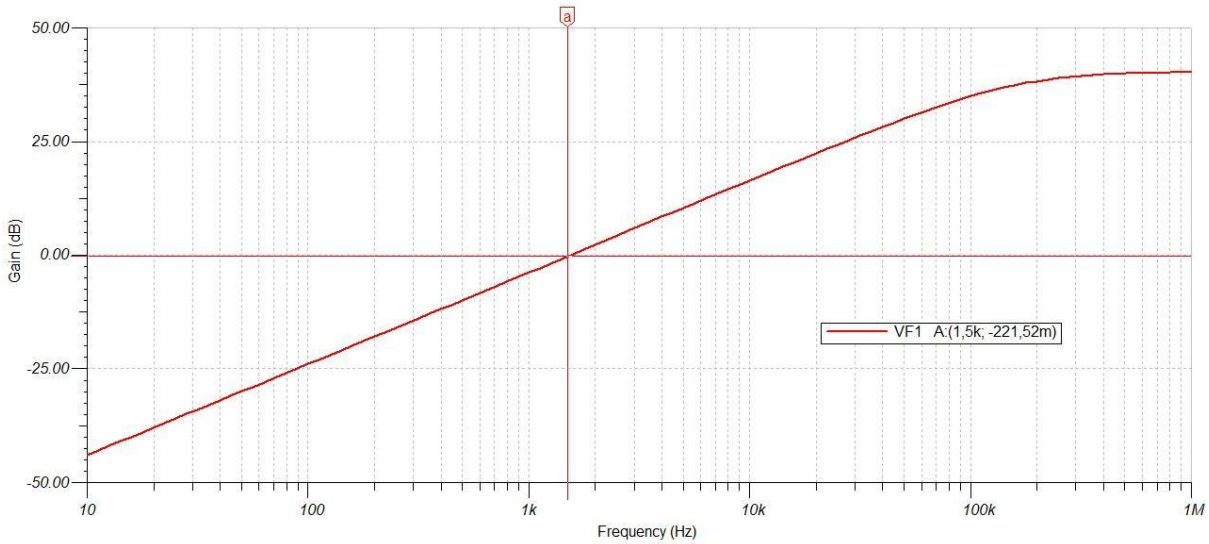
De spanningsdeler maakt van de spanning 12 V, een spanning van 3,3 V. De weerstand R_8 is: 26,37 k.

In Tabel 14 is een overzicht van de berekende componenten

Tabel 14: Overzicht componenten ontvanger

Component:	Berekende waarde:
R1	2,2 kΩ
R2	10 kΩ
R3	10 kΩ
R4	1 kΩ
R5	100 KΩ
R6	1 kΩ
R7	120 k
R8	26,36 k
R9	10 k
P1	0, 100 k
C1	21,22 nF
C2	1,06 nF
C3	1,59 µF

Het schema uit Figuur 31 is gesimuleerd en is weergegeven in Figuur 32.

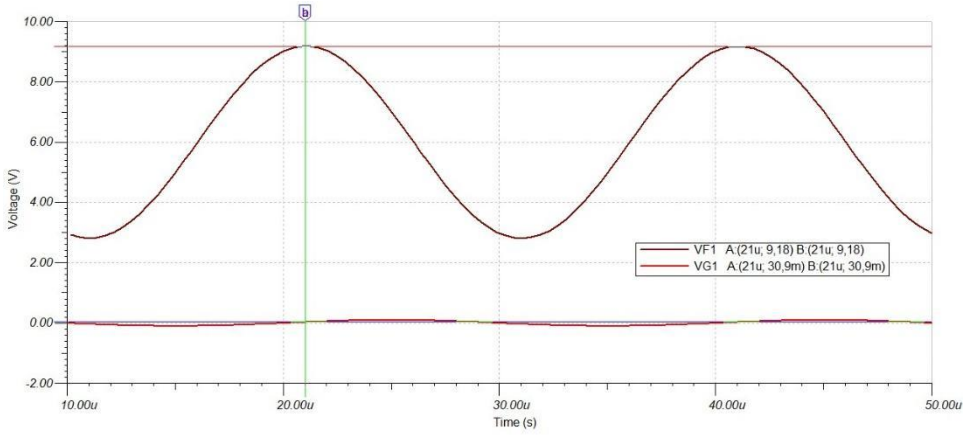


Figuur 32: Frequentie response versterker

De versterkingsfactor van de opamp is ingesteld met de potmeter. De opamp versterkt 100 keer. Met formule (11.6.9) is de versterkingsfactor uitgerekend. Op 1,5 kHz is de versterking 0,94.

(11.6.9) $$A_v = 20^{\frac{-0,221}{10}} = 0,94$$

In Figuur 33 is uitgang van versterker weergegeven. VG1 is het ingangssignaal uit de microfoon. VF1 is de uitgang van de opamp.



Figuur 33: microfoon versterker

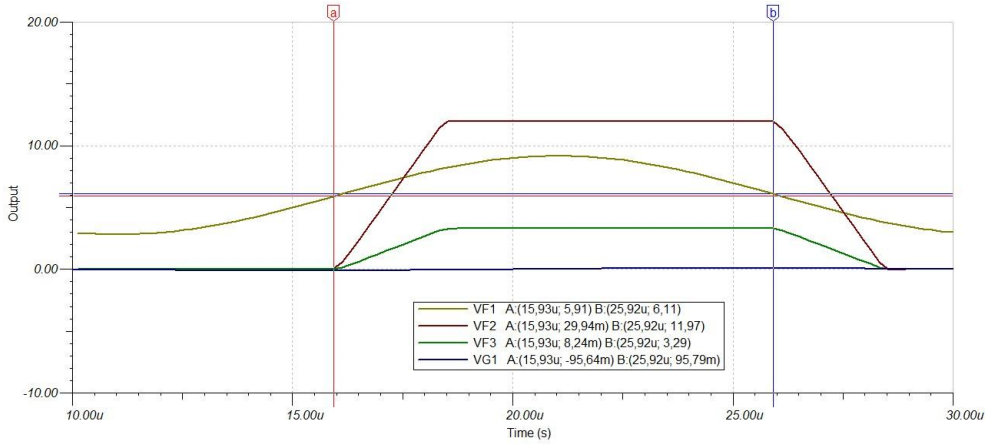
Met formule (11.6.10) is de versterking berekend. Omdat het ingangssignaal een offset heeft van 6 V moet dit van het uitgangssignaal worden afgehaald.

(11.6.10) $$A_v = \frac{V_{out} - V_{offset}}{V_{in}} = \frac{9,18-6}{30,9 \cdot 10^{-3}} = 102,91$$

De versterkingsfactor is 102,91. In Figuur 34 is de simulatie van de hele schakeling weergegeven. Hieronder is een overzicht van de signalen:

-VG1 = Ingangssignaal;
-VF1 = Uitgang opamp microfoonversterker;
-VF2 = uitgang schmitt trigger;
-VF3 = logisch level dat wordt aangeboden aan de microcontroller

Als de ingang van de schmitt trigger 5,91 V is gaat de uitgang van de opamp naar de voedingsspanning. Omdat de opamp niet oneindig snel is duurt het even voor de uitgang van de opamp omhoog gaat. Bij een spanning van 6,11 V verandert de status van 'hoog' naar 'laag'.



Figuur 34: simulatie van gehele schakeling

11.7 Energieverbruik

Het energieverbruik van de Heliumzuiverheidsmeter is weergegeven in Tabel 15. De zuiverheidsmeter neem bij vollast een vermogen op van 15,6 W. Dit is bij het schakelen van de kleppen in- en uitlaat Helium. In normaal gebruik bij het meten van de zuiverheid en het aansturen van het 7-segments display is het opgenomen vermogen 6 W. Wanneer er niks aangestuurd wordt is de schakeling in rust en is het opgenomen 5,04 W.

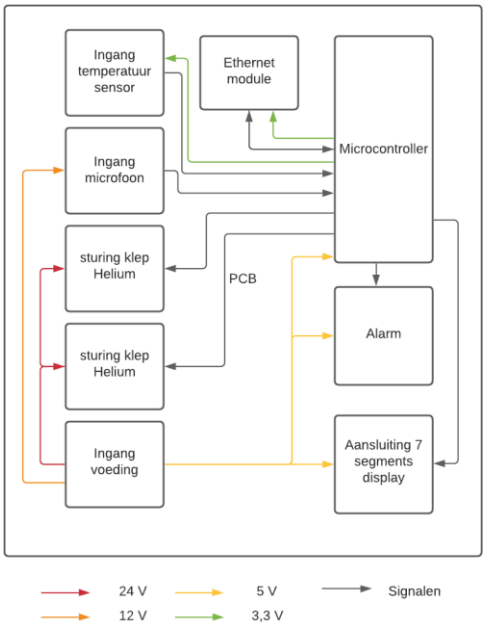
Tabel 15: Energieverbruik

Omschrijving	Spanning (V)	Stroom verbruik (mA)	Opgenomen vermogen (W)
Rust	24	210	5,04
Normaal gebruik	24	250	6
Vollast	24	650	15,6

12 PCB

Een PCB (*printed circuit board*) is de drager voor elektrische componenten. Met koperen sporen worden de componenten met elkaar verbonden.

De PCB is ontwikkeld met het programma 'Altium designer'. In Figuur 35 is het blokschema weergegeven. Ook is in het blokschema de positionering van de componenten weergegeven. In rood is de 24 V-voeding weergegeven, in oranje is de 12 V-voeding weergegeven, in het geel is de 5 V-voeding weergegeven en in het groen is de 3,3 V-voeding weergeven. De zwarte lijnen zijn signalen die over de PCB lopen.



Figuur 35: Blokschema PCB

De PCB bestaat uit twee lagen, een boven en een onderlaag. Bij het ontwerp van de PCB is ervoor gezorgd dat de sporen zoveel mogelijk aan de bovenkant van de PCB lopen. Aan de onderkant van de pcb zit de '*groundlayer*'. Om te zorgen dat zo min mogelijk storing kan ontstaan wordt de '*groundlayer*' onder het analoge gedeelde niet onderbroken. Het ontwerp van de PCB is weergegeven in bijlage 2. De PCB met componenten is weergegeven in Figuur 37 . De onderkant van de PCB is weergegeven in Figuur 36

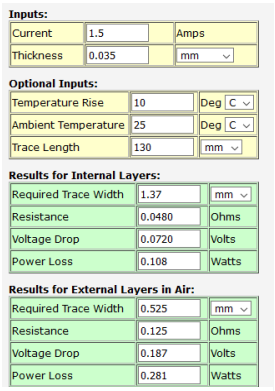
Figuur 37: Bovenkant PCB met gesoldeerde componenten

Figuur 36: Onderkant PCB

12.1 Spoorbreedte

De spoorbreedte bepaalt hoeveel stroom er kan lopen zonder dat het PCB-spoor te heet wordt.

De grootste stroom is de stroom voor de aansturing van de tweeter. De stroom gevraagd voor de tweeter is: 1,5 A. Met de online tool [16] is een spoorbreedte van 0,525 berekend (bij een spoordikte van 0,035 mm). Voor de veiligheid is voor een spoordikte van 0,8 mm gekozen. De sporen waar de signalen over gaan, hebben een breedte van 0,25 mm. De berekening is weergegeven in Figuur 38.



Figuur 38: Berekening spoorbreedte

12.2 Componenten

Voor de PCB zijn zowel smd als through-hole componenten gebruikt. In Tabel 16 is een overzicht van de uitvoering en het formaat van de componenten. De afmetingen van weerstand met het formaat 0805 is: 2 x 1,25 mm. Het formaat van een 1206 is: 3,2 x 1,6 mm. Hieronder is een overzicht van de betekenis van de formaat types:

SOD: small outline diode;
SOT: small outline transistor;
SOIC: Small outline integrated chip;
TO: transistor overview.

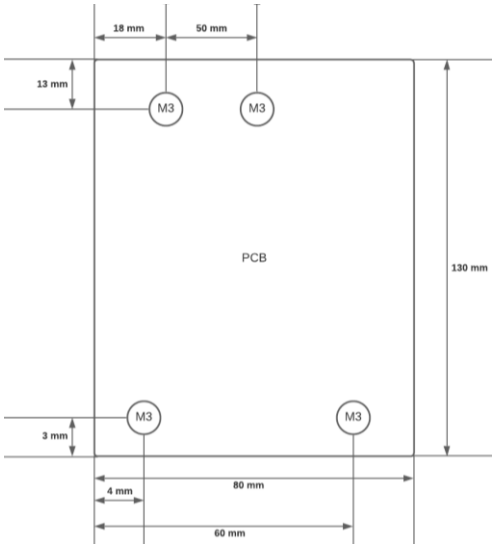
Tabel 16: Overzicht uitvoering componenten

Component:	Uitvoering:	Formaat:
Weerstand	SMD	0805
Condensator (Keramisch)	SMD	0805
LED	SMD	1206
Opamp	SMD	SOIC8
Zenerdiode	SMD	SOD123
transistor	SMD	SOT23
Condensator	Through hole	tantaal
Diode	Through hole	DO31
MOSFET	Through hole	TO220
Zekering	Through hole	

Het nadeel van de weerstanden met het formaat 0805 is dat deze maximaal een vermogen aankunnen van 0,125 W. Er is gebruik gemaakt van tantaal condensatoren omdat deze een langere levensduur hebben en een lagere inwendige weerstand voor puls-signalen. Dit komt omdat een tantaalcondensator beter bestand is tegen hogere temperaturen dan een elektrolytische condensator. De componenten zijn weergegeven in bijlage 3.

12.3 Afmetingen

In Figuur 39 zijn de afmetingen van de PCB weergeven. Met M3 "*spacers*" is de PCB bevestigd aan de behuizing.



Figuur 39: Afmetingen PCB

12.4 Vias

Met een 'via' wordt verbinding gemaakt tussen de boven en onderlaag van de PCB. Omdat de *groundlayer* van de PCB aan de onderkant is, moeten de *ground* pinnen van de SMD-componenten verbonden worden met de onderlaag. De via's hebben een binnen diameter van 0,5 mm en een buitendiameter van 0,9 mm.

12.5 Molex

Om te zorgen dat het 7-segmentsdisplay niet verkeerd aangesloten kan worden wordt gebruik gemaakt van een 4 pins molex stekker.

12.6 Inplugbare eindblokken

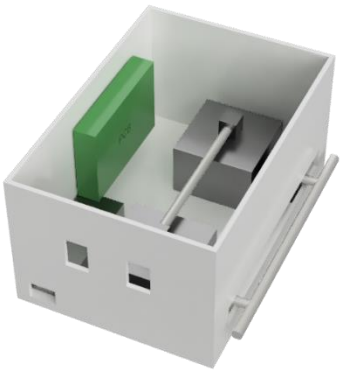
De sensoren en actuatoren worden aangesloten met inplugbare eindblokken. Door gebruik te maken van eindblokken, kan de PCB makkelijk losgekoppeld worden wanneer deze defect is. De voeding, kleppen in- en uitlaat Helium, tweeter en microfoon worden aangesloten met de inplugbare eindblok van het type: 2P 5,08 mm HDR RA. De sensor voor het meten van de temperatuur wordt aangesloten met inplugbare eindblok van het type: 3P 5,08 mm HDR RA.

12.7 Ontwerpkeuzes

De voeding regulators (LM7812 en LM7805) bevinden zich aan de linker onderkant van de PCB. Dit is gedaan zodat er gemakkelijk een *heatsink* aan kan bevestigd worden. Ook de MOSFET's zijn als *though hole* uitgevoerd zodat er gemakkelijk een *heatsink* aan bevestigd kan worden. De USB aansluiting voor de microcontroller en aansluiting voor de Ethernet zit aan de bovenkant van de pcb voor makkelijke toegang. Extra pinnen (*headers*) zijn geplaatst voor het geval de LED's of alarm naar de buitenkant van de behuizing geplaatst moeten worden.

13 Ontwerp heliumzuiverheidsmeter

Het ontwerp van de behuizing is gemaakt met het programma: “Autodesk fusion 360”. De functie van de behuizing is het afschermen van de testopstelling, voeding en elektrische componenten. Met een 7-segments display wordt aan de gebruiker de vervuiling van het Helium weergegeven. Ook kan de gebruiker met een drukknop het alarm uitschakelen. Aan de onderzijde van de behuizing kunnen de voeding, de Ethernet en USB worden aangesloten. Rechts van de behuizing wordt retourleiding van het Helium aangesloten. Doordat de zuiverheidsmeter parallel aan het proces wordt aangesloten wordt het proces nooit onderbroken. In Figuur 40 en bijlage 4 is de 3D weergaven van de behuizing weergegeven. In Figuur 41 is het eindontwerp weergegeven.



Figuur 40:3D ontwerp Heliumzuiverheidsmeter

Figuur 41: Eindontwerp Heliumzuiverheidsmeter

13.1 Behuizing

Om te zorgen dat de gebruiker niet zelf aan de meetopstelling kan komen, wordt een behuizing om de meetopstelling gebruikt. De behuizing is van het merk Schneider Electric. De afmetingen van de kast zijn: 310 (h) x 215 (b) x 160 (d) mm. In deze behuizing is plaats voor de voeding, PCB en zuiverheidsmeter. De gebruikte behuizing is weergegeven in Figuur 42

Figuur 42: Behuizing om de Heliumzuiverheidsmeter

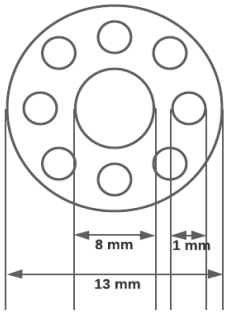
13.2 Meter

Omdat Helium een zeer licht gas is moet dit in een hermetisch afgesloten ruimte zijn. Om dit realiseren is gebruik gemaakt van twee afgesloten behuizingen. In Figuur 43 is een 3D weergave van de meter. De afmetingen van de behuizingen zijn 90 x 90 x 60 mm. In één behuizing wordt de tweeter geplaatst en in de andere behuizing de microfoon. Met een metalen buis van 20 cm worden de twee behuizingen aan elkaar gekoppeld. Een wartel met rubberen afsluiting zorgt voor een hermetisch afgesloten koppeling tussen de metalen buis en de meetkast.

Figuur 43: 3D Ontwerp meter

13.3 Microfoon bevestiging

Met het onderdeel uit Figuur 44 wordt de microfoon in de wartel bevestigd. De binnendiameter van de wartel is 13 mm. Het gat in het midden heeft een diameter van 8 mm en is bestemd voor de microfoon. De gaten rond de microfoon zijn bedoeld voor het doorlaten van het Helium.



Figuur 44: Afmetingen voor plaatsen microfoon in wartel

13.4 Bedrading

Om de sensoren en actoren met de PCB te koppelen wordt koperen bedrading gebruikt. Om zo min mogelijke storing te ontvangen wordt gebruik gemaakt van een kabels met afscherming. In het onderstaande (Tabel 17) zijn de kabels weergeven.

Tabel 17: Gebruikte bedrading

Aantal aders:	Diameter (mm^2):	Lengte (cm):	Omschrijving:
4	0,25	50	7-seg display
3	0,25	30	Temperatuursensor
2	0,25	30	Klep uitlaat
2	0,25	30	Klep inlaat
2	0,25	50	Drukknop
1	0,14	30	Microfoon

14 Software

De software zorgt voor de regeling van het systeem. De software is verantwoordelijk voor het meten van de tijdsduur tussen zender en ontvanger, meten van temperatuur, het aansturen van de in- en uitlaatkleppen Helium en het versturen van data naar de server. De software is geschreven in C++ met behulp van de ontwikkelomgeving 'PlatformIO'. De software is weergegeven in bijlage 5

14.1 Microcontroller

Het gebruikte microcontrollerboard is van het type: 'Teensy 3.6'. Dit *board* heeft 42 I/O pinnen die werken met een spanning van 3,3 V. Wanneer een hogere spanning dan 3,3 V wordt aangeboden, zal de microcontroller defect raken. Verder beschikt de microcontroller over I2C en SPI benodigd voor het 7-segments display en de Ethernet module. De microcontroller op het *board* heeft een klokfrequentie van 180 MHz. Een snelle controller is belangrijk voor het nauwkeurig meten van de geluidssnelheid. Het nadeel van de hoge klokfrequentie is het energieverbruik van de microcontroller toeneemt. In de onderstaande tabellen (Tabel 18 en Tabel 19) is een overzicht van de aansluitingen van de microcontroller.

Tabel 18: I/O microcontroller

I/O/A (Ingang/uitgang/analoog)	Pin:	Omschrijving:
I	29	Microfoon
I	22	Drukknop
O	30	Tweeter
O	34	Alarm
O	31	Klep inlaat Helium
O	32	Klep uitlaat Helium
A	A23	Temperatuur sensor

Tabel 19: I/O microcontroller

Protocol:	Pin:	Omschrijving:
I2C (7-segmentsdisplay)	19	SCL
	18	SDA
SPI (ethernet module)	12	MISO
	11	MOSI
	10	SS
	15	RST
	14	Interrupt

14.2 Tijdsduur meten

De tijdsduur bepaalt de zuiverheid van het Helium. Hoe langer de tijdsduur des te groter de vervuiling.

De tweeter stuurt een geluidspuls uit. Wanneer de microfoon de puls detecteert komt dit binnen op de digitale pin 29 van de microcontroller. De tijd tussen de uitgestuurde puls van de tweeter en het ontvangen van de microfoon is de tijdsduur (geluidssnelheid).

Om de tijd te krijgen wordt gebruik gemaakt van de klokfrequentie van microcontroller. Met de functie ARM_DWT_CYCCNT kan de huidige aantal klokpulsen worden opgevraagd. Wanneer een geluidspuls wordt uitgestuurd, wordt ook de huidige aantal klokpulsen opgevraagd (Cycle Counter start). Wanneer de microfoon de puls ontvangt vindt een

interrupt plaats. In de interrupt routine wordt ook het aantal klokpulsen opgeslagen (Cycle Counter eind). Met formule (14.2.1) kan het totaal aantal pulsen worden uitgerekend.

(14.2.1) $$Aantal\ klokpulsen\ meting = Cycle\ Counter\ eind - Cycle\ Counter\ start$$

De microcontroller heeft een klokfrequentie van 180 MHz. Met formule (14.2.2) is de lengte van een klokpuls berekend.

(14.2.2) $$t_{CPU} = \frac{1}{F_{CPU}} = \frac{1}{180 \cdot 10^6} = 5{,}55\ ns$$

De lengte van een klokpuls is 5,55 ns. Om van het aantal klokpulsen de tijd te berekenen is gebruik gemaakt van formule (14.2.3).

(14.2.3) $$t_{totaal}(ns) = Aantal\ klokpulsen\ meting \cdot t_{CPU}$$

Voor het berekenen van de zuiverheid van het helium zal alleen gebruikt worden van het aantal gemeten klokpulsen.

14.3 7-segments display

Voor het weergeven van de zuiverheid in het Helium wordt een 7-segments display gebruikt. Het 7-segments display werkt met het I2C protocol. Met twee weerstanden van 4,7 kΩ wordt de I2C lijn 'omhoog' getrokken naar de 3,3 V. De voeding (V_s) voor het 7-segments display is 5 V.

Omdat een 7-segments display op grotere afstand nog goed leesbaar is, maakt dit het zeer geschikt voor deze toepassing. Het 7-segments display heeft 4-digits. Het gekozen 7-segments display is weergeven in Figuur 45.

Figuur 45: Gebruikte 7-segments display

Wanneer er geen verbinding met de server gemaakt kan worden, wordt de melding 'Err Conn' op het 7-segments display afgedrukt (Figuur 46). Wanneer de vervuiling te groot is wordt de melding 'Err sys' weergegeven (Figuur 47). Omdat het 7-segments display 4-digits heeft wordt er eerst 'Err' afgedrukt. Met een interval van één seconde wordt de melding die er onderstaat ('Conn' of 'sys') weergeven.

Figuur 46: Err Conn

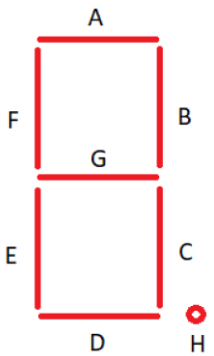
Figuur 47: Err SYS

Het aansturen van een individueel segment wordt gedaan door een byte naar het 7-segmentsdisplay te sturen. De functie voor het aansturen van een individueel segment is als volgt te gebruiken:

Matrix.WriteDigitRaw(segment,B HGFEDCBA);

De eerste 'B' geeft aan dat het een byte is. De letter 'A' tot 'H' corresponderen met de letters uit Figuur 48. Het getal na de komma geeft aan om welk segment het gaat. (0) is het meest linker segment. (4) is het meest rechter segment. Omdat dit 7-segments display ook een dubbele punt in het midden heeft is dit segment (2)

De letter 'E' is te maken door de byte 0b01111001 te sturen naar de display.



Figuur 48: Benaming losse segmenten

14.4 Alarm

Om de gebruiker te alarmeren wordt gebruik gemaakt van een piëzo luidspreker. De piëzo luidspreker is aangesloten op een spanning van 5 V en kan een geluid produceren tot 93 dB. Er zijn drie verschillende alarmeringen mogelijk.

1. Alarm uit;
2. 3x beep;
3. Elke 10 seconde een beep voor 2 minuten lang;
4. 20 keer korte piep daarna elke 10 seconde een beep.

De server bepaalt welk alarm wordt gegenereerd. Met de drukknop voor op de behuizing kan het alarm worden afgeschakeld.

14.5 Aansturing van de kleppen

De functie van de in- en uitlaatkleppen is het nemen van een sample uit het proces. Met één klep wordt het Helium de meetopstelling ingelaten. Met een andere klep wordt gezorgd dat het Helium niet uit de meetopstelling kan lopen. De kleppen worden opengezet zodat het helium kan doorstromen na 10 sec worden de kleppen weer gesloten nadat er gemeten kan worden. Na het sluiten van de klep moet 20 milliseconde gewacht worden met meten zodat solenoïde volledig is uitdenderd.

14.6 Temperatuur sensor

De gebruikte temperatuursensor is van het type 'LM35CAZ'. De sensor heeft 3 pinnen waarvan één pin (V_{out}) aangesloten is op een analoge ingang van de microcontroller. De voeding pin (V_s) is aangesloten op een spanning van 3,3 V. De *ground* is aangesloten op een aparte pin van de microcontroller (*Analog* GND). Dit is zodat de betrouwbaarheid van de meting verhoogd wordt. De temperatuursensor kan temperaturen meten tussen de -55 en 155 °C. De temperatuur sensor heeft een lage '*self heating'* van 0,008 °C. Dit is gunstig omdat de temperatuursensor altijd aanstaat. Voor iedere graad Celsius (°C) geeft de sensor 10 mV op de pin V_{out}. Met de onderstaande formule (16.6.1) wordt de spanning omgerekend naar de temperatuur in graden Celsius.

(14.6.1) $$Temperatuur\ (°C) = \frac{U_{ADC} \cdot (\frac{V_{Teensy}}{Resolutie_{ADC}})}{10\ mV}$$

Om van temperatuur in graden Celsius om te rekenen naar Kelvin wordt gebruik gemaakt van formule (12.6.2).

(14.6.2) $$Temperatuur\ (K) = Temperatuur\ (°C) + 273{,}15$$

De temperatuur in Kelvin (K) wordt verstuurd naar de server. Op deze manier worden negatieve waarden voorkomen, wat het parsen aan serverzijde vergemakkelijkt. Wanneer na het meten de gemiddelde temperatuur bekend is, wordt een correctiefactor over de zuiverheid van het Helium berekend. De correctie over het gemeten Helium wordt berekend met formule (14.6.3).

(14.6.3) $$P_{w=}\ P_g - (T_g - T_c) * 0{,}17$$

P_w: percentage werkelijk [%]
P_g: percentage gemeten [%]
T_w: temperatuur gemeten [$°C$]
T_c: temperatuur gekalibreerd [$°C$]

14.7 EEPROM

EEPROM staat voor (*Electrically erasable programmable read-only memory*) en maakt deel uit van de microcontroller in de Teensy 3.6. De EEPROM zorgt dat data behouden blijft, ook nadat de spanning is afgeschakeld. In de EEPROM wordt het volgende opgeslagen:

1. IP-adres microcontroller
2. IP-adres server
3. Poort server

De EEPROM werkt met adressen. Op ieder adres kan één byte worden opgeslagen. In het onderstaand tabel overzicht is de grote van de data die opgeslagen wordt.

Omschrijving:	Grote in Bytes
IP-adres microcontroller	4
IP-adres server	4
Poort server	2
MAC-adres	6

Voor het aansturen van het EEPROM is gebruik gemaakt van de EEPROM.h *library*.

14.8 Kalibreren

Voordat de Heliumzuiverheidsmeter gebruikt kan worden moet deze eerst gekalibreerd worden. Dit is een eenmalig proces, waarbij de kalibratiewaarden eenmalig worden vastgesteld. Bij gelijke hardware, zullen deze waarden voor ieder exemplaar van de Heliumzuiverheidsmeter identiek zijn. Gekozen is daarom om deze waarden "*hardcoded*" in de software op te nemen.

14.9 Ethernet

Om verbinding te maken met een server wordt de microcontroller uitgerust met een ethernet module. De ethernet module van het type "W5500". Met een RJ45 kabel kan de meetopstelling aangesloten aan het lokale netwerk. Er is gekozen voor een ethernet module omdat deze een hogere veiligheid en betrouwbaarheid heeft dan wifi. Ethernetaansluitingen zijn door het gehele gebouw aanwezig en kunnen via patchkasten op de juiste manier worden gerouteerd naar de verschillende VLANs. Als de verbinding met de server verbroken is zal de Heliumzuiverheidsmeter automatisch opnieuw verbinding met de server proberen te maken.

Voor het versturen van data over Ethernet is gebruik gemaakt van de Ethernet.h *library*.

14.9.1 IP-adres Microcontroller

Het IP-adres van de microcontroller heeft een waarde tussen 0.0.0.0 en 255.255.255.255. Het IP-adres is in te stellen door een usb kabel aan te sluiten en het sturen van het commando:

Setip <aaa.bbb.ccc.ddd>

Met het commando "showip" wordt het huidige IP-adres van de microcontroller teruggegeven.

14.9.2 MAC-adres Microcontroller

Het MAC-adres voor de ethernet module bestaat uit een 6 bytes hexadecimaal getal. Het MAC-adres is een getal tussen 0x00,0x00,0x00,0x00,0x00,0x00 en 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF. De ethernet module heeft van zichzelf geen vast MAC-adres. Het is belangrijk dat het MAC-adres vast staat. Binnen het netwerk van de universiteit wordt op IP-adres gefilterd. Om de Heliumzuiverheidsmeter aan te melden bij het netwerk van de Universiteit moet het MAC-adres worden gekozen en vervolgens worden aangemeld bij de universiteit.

14.9.3 IP-adres Server
Het IP-adres van de server heeft een waarde tussen 0.0.0.0 en 255.255.255.255. Met het onderstaande commando kan het IP-adres van de server worden ingesteld:

Setipserver <aaa.bbb.ccc.ddd>

Met het commando "showipserver" wordt het huidige IP-adres van de server teruggegeven.

14.9.4 Poort Server
Het poortnummer van server is een waarde tussen de 0 en 65535. Deze waarde is op te slaan als 2 bytes *unsigned integer*.

14.9.5 Commando's naar server
De meetopstelling stuurt data naar server waar het vervolgens verwerkt kan worden. De volgende commando's worden gebruikt voor het sturen van data naar de server.

heliumpurity<x>	helium percentage (0,00…100,00)
heliumtemperature<x>	temperatuur helium(in kelvin)
switch<x>	toestand van drukknop("*pushed*")

14.9.6 Commando's van server
De volgende commando's van de server worden verwerkt:

OK	Meting is juist ontvangen door server
Alarm<x>	piëzo alarm gaat af

0 : Alarm uit
1 : 3x beep
2 : elke seconde 10 seconde beep elke 2 minuten
3 : 20x beep daarna elke 10 seconde

14.10 Berekenen van Percentage
Om de zuiverheid van het Helium in percentage uit te drukken wordt gebruik gemaakt van formule (14.10.1). Het aantal pulsen voor lucht (21 $°C$) is vast gesteld op 140282 dit is dus wanneer het Helium 0 % is. Bij 100 % Helium is het aantal pulsen vastgesteld op 52295. Aan de hand van deze gegevens is het percentage Helium berekend. De gebruikte formule is afgeleid van formule (14.10.1).

(14.10.1) $y = ax + b$

De waarde van a is berekend met formule (14.10.2).

(14.10.2) $a = \frac{100\ -0}{Waarde\ helium - waarde\ lucht} = \frac{100 - 0}{52295 - 140282} = -1{,}136 \cdot 10^{-3}$

De waarde van b is berekend met formule (14.10.3).

(14.10.3) $b = 1{,}136 \cdot 10^{-3} * waarde\ lucht = 1{,}136 \cdot 10^{-3} * 140282 = 159{,}43$

Met formule (14.10.4) wordt de waarde van de zuiverheid van het helium berekend.

(14.10.4) $Percentage\ Meting\ (\%) = -1{,}136 \cdot 10^{-3} \cdot Gemeten\ aantal\ pulsen + 159{,}43$

14.11 ISR
ISR staat voor *interrupt service routine*. De ISR is zodanig ingesteld dat deze optreedt bij een opgaande flank (een verandering in het signaal van "laag" naar "hoog"). Het programma wordt onderbroken en de ISR wordt gestart. In de ISR wordt het huidige aantal klokpulsen

opgeslagen. Door de reflecties die ontstaan in de behuizing, vinden er meerdere interrupts plaats. Het aantal pulsen wordt alleen opgeslagen in de eerste interrupt die plaats vindt. Dit is omdat deze maatgevend is voor de tijd van het niet-gereflecteerde geluidssignaal.

14.12 USB

Om te zorgen dat kwaadwillenden geen commando's via Ethernet kunnen sturen om bijvoorbeeld het IP-adres aan te passen, wordt de microcontroller ingesteld via USB. Wanneer de USB verbonden is met de opstelling kan het programma "Putty" gebruikt worden voor het instellen van de microcontroller. Met het gebruik van "CommandHandler.h" en "CommandParser.h" worden de commando's verwerkt. De onderstaande commando's kunnen gebruikt worden om de Heliumzuiverheidsmeter in te stellen.

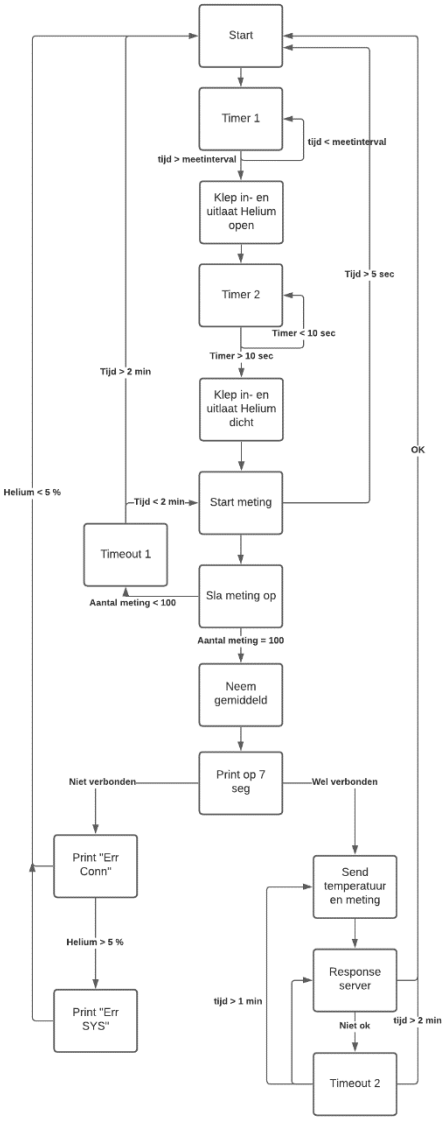
Ver terugmelding: versienummer.	geeft softwareversie nummer
Help terugmelding: helpscherm met alle commando's.	geeft alle commando's weer
!	Herhaalt laatst gegeven commando (zonder enter)
idn	geeft project ID
setip aaa.bbb.ccc.ddd (0.0.0.0 t/m 255.255.255.255) terugmelding: OK	stelt IP-adres microcontroller in
showip	Geeft huidig IP-adres microcontroller
setipserver aaa.bbb.ccc.ddd (0.0.0.0 t/m 255.255.255.255) terugmelding: OK	stelt IP-adres server in
showipserver	Geeft huidig IP-adres server
setmac (0:0:0:0:0:0 t/m 255:255:255:255: 255:255) terugmelding: OK	Stelt het MAC-adres van de microcontroller in
showmac	Geeft huidig MAC-adres van de microcontroller
setpoort <x> (0 t/m 65535) terugmelding: OK (after reboot)	Stelt poortnummer server in
showport	Geeft huidige poort nummer server
Reconnect	verbind opnieuw met server
reboot	reboot microcontroller

14.13 State machine

Met de statemachine (weergegeven in Figuur 49) wordt de meting aangestuurd. Er wordt gebruik gemaakt van een statemachine zodat de rest van het programma niet wordt onderbroken tijdens het uitvoeren van metingen.

De statemachine bestaat uit de volgende "*states*":

- Start;
- Timer1;
- Klep in-en uitlaat Helium open;
- Timer2;
- Klep in-en uitlaat Helium dicht;
- Start meting;
- Sla meting op;
- Timeout 1;
- Neem gemiddeld;
- Print op 7 seg;
- Neem gemiddeld;
- Print op 7 seg;
- Print "Err Conn";
- Print "Err SYS";
- Send temperatuur en meting;
- Response server;
- Timeout 2.



Figuur 49: Statemachine

Start: Wanneer de statemachine wordt gestart komt deze in de eerste state 'Start'. In deze state worden de waardes geïnitialiseerd en gereset, bijvoorbeeld de waarde van het gemiddelde over de metingen.

Timer 1: In deze state wordt gekeken of er tijd van het meetinterval is verlopen. Het meetinterval is ingesteld op 1 minuut. Op deze manier wordt er elke minuut een nieuwe meting gestart.

Klep in-en uitlaat Helium open: In deze state worden de kleppen voor het in-en uitlaat Helium open gestuurd zodat de opstelling een nieuwe sample kan nemen.

Timer 1: In deze state wordt gekeken of er 5 seconde zijn verstreken. Wanneer 10 seconde zijn verstreken wordt doorgegaan naar de volgende state.

Klep in-en uitlaat Helium dicht: In deze state worden de kleppen voor het in-en uitlaat Helium dicht gestuurd zodat het Helium niet terug het proces kan inlopen.

Start meting: In deze state wordt de meting gestart door een puls op de tweeter te zetten.

Sla meting op: In deze state wordt het aantal pulsen opgeslagen die nodig waren om voor het geluid bij de microfoon te komen. Dit wordt pas gedaan wanneer een interrupt heeft plaats gevonden. Bij het opslaan van de meting wordt het aantal metingen verhoogd. Ook wordt bij elke interrupt de temperatuur gemeten. Als het aantal metingen kleiner is dan 100, wordt een nieuwe meting gestart. Als er 100 metingen zijn gedaan wordt er doorgegaan naar de volgende state.

Timeout 1: In deze state wordt gekeken of er 2 minuten zijn verstreken. Als er na 2 minuten nog steeds de state “start meting” wordt uitgevoerd is er iets misgegaan en wordt opnieuw gestart met meten. Op deze manier zal het programma niet vastlopen.

Neem gemiddeld: In deze state wordt het gemiddelde genomen over de 100 metingen. Voor zowel de temperatuur als de metingen. Ook wordt in deze state de percentage vervuiling berekend.

Print op 7 seg: In deze state wordt het percentage vervuiling op het 7-segments display afgedrukt.

Print “Err Conn”: Wanneer er geen verbinding met de server gemaakt kan worden wordt de melding “Err Conn” op het 7 segments afgedrukt.

Print “Err SYS”: Wanneer het Helium meer vervuild is dan 5 procent wordt de melding “Err SYS” op het 7 segments-display afgedrukt.

Send temperatuur en meting: Wanneer de opstelling verbonden is met de server wordt in deze state de zuiverheid van het Helium en de temperatuur in Kelvin naar de server gestuurd.

Response server: In deze state wordt gewacht tot de server de data goed ontvangen heeft en heeft gereageerd met “OK”.

Timeout 2: Als de server geen “OK” heeft terug gegeven wordt na 1 minuut de data opnieuw naar de server gestuurd. Als na 2 minuten nog steeds geen terugmelding van de server is gekomen wordt opnieuw gestart met meten.

15 Verificatie van het systeem

Het verifiëren van het systeem wordt gedaan door de Heliumzuiverheidsmeter te testen en controleren op de eisen, gegeven door de opdrachtgever.

15.1 Betrouwbaarheid van de metingen

Wanneer één meting gedaan wordt zegt dit weinig over de betrouwbaarheid. Om de nauwkeurigheid van de metingen bepalen wordt de standaarddeviatie berekend. De standaarddeviatie is een maat voor de spreiding van de meetwaarde. Meer metingen geven een beter overzicht van de spreiding. Hoe kleiner de spreiding (σ), hoe minder afwijking er zal optreden tussen de meetwaarde en de geijkte waarde. Voor de metingen in lucht en Helium zijn 500 metingen gedaan voor het berekenen van de standaarddeviatie.

15.2 Metingen in lucht

In Tabel 20 is een overzicht van de gemeten waarde in lucht. De tijd is weergegeven in µs. De frequentie geeft aan hoeveel metingen bij die tijdseenheid horen.

De standaarddeviatie is uitgerekend door het programma Excel. De standaarddeviatie(σ) is: 0,1138. De gemiddelde tijd van alle metingen is: 771,23 µs. De statistische maximale waarde is berekend met formule (15.2.1).

(15.2.1) $$max = \mu + 3\sigma = 771{,}23 + (3 \cdot 0{,}1138) = 772{,}96\ \mu s$$

Het statistisch maximum is 772,96 µs.

De statistische minimale waarde is berekend met formule (15.2.2).

(15.2.2) $$min = \mu - 3\sigma = 771{,}23 - (3 \cdot 0{,}1138) = 771{,}16\ \mu s$$

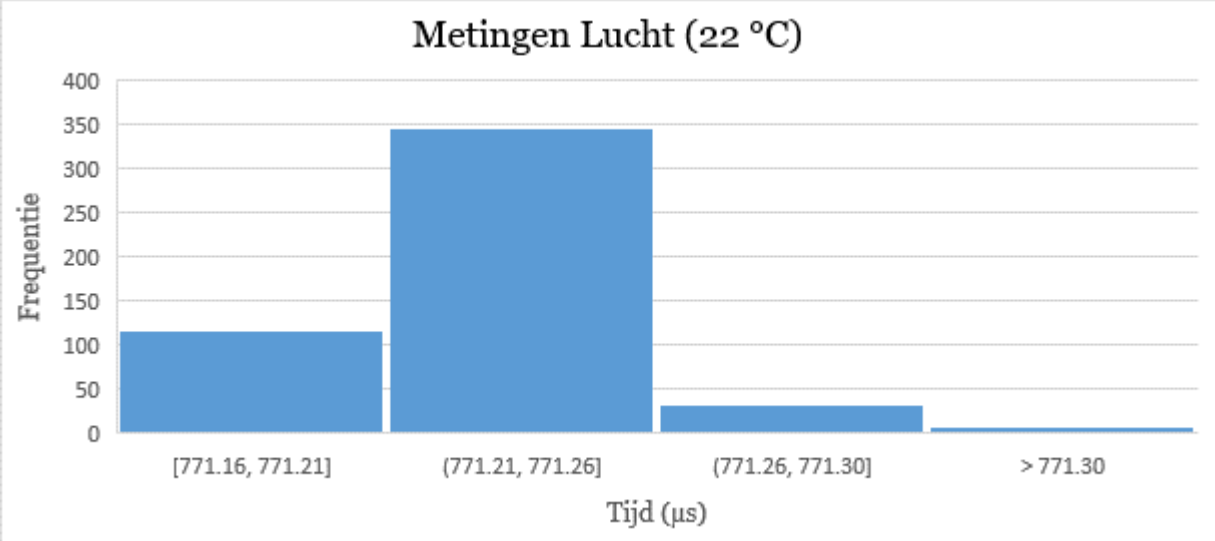
Het statisch minimum is 771,16 µs. De meetwaardes die buiten de minimale en maximale waarde kunnen beschouwd worden als statistische uitschieters. De absolute afwijking is berekend met formule (15.2.3)

(15.2.3) $$Absolute\ afwijking = 3 \cdot \sigma = 3 \cdot 0{,}1138 = 0{,}341\ \mu s$$

De absolute afwijking per meting is: 0,341 µs.

Tabel 20: Frequentietabel metingen lucht

Tijd (µs)	771,0	771,1	771,2	771,3	771,4	771,5	771,6	771,7	771,8	771,9	772,0	772,1	772,2	772,3
Frequentie	0	0	66	428	1	1	1	0	1	0	0	0	0	0



Figuur 50: Histogram metingen lucht

15.3 Metingen in Helium

In Tabel 21 is een overzicht van de gemeten tijd (µs) in Helium. In totaal zijn er 500 metingen gedaan met een interval van 25 ms. De gemiddelde gemeten tijd is 290,59 µs. De standaarddeviatie is door Excel berekend en is: 0,3826. De statistische maximale waarde is berekend met formule (15.3.1)

(15.3.1) $$max = \mu + 3\sigma = 290{,}59 + (3 \cdot 0{,}3826) = 291{,}74\ \mu s$$

Het statisch maximum is: 212,26. De statische minimale waarden is berekend met formule (15.3.2).

(15.3.2) $$min = \mu - 3\sigma = 290{,}59 - (3 \cdot 0{,}3826) = 289{,}44\ \mu s$$

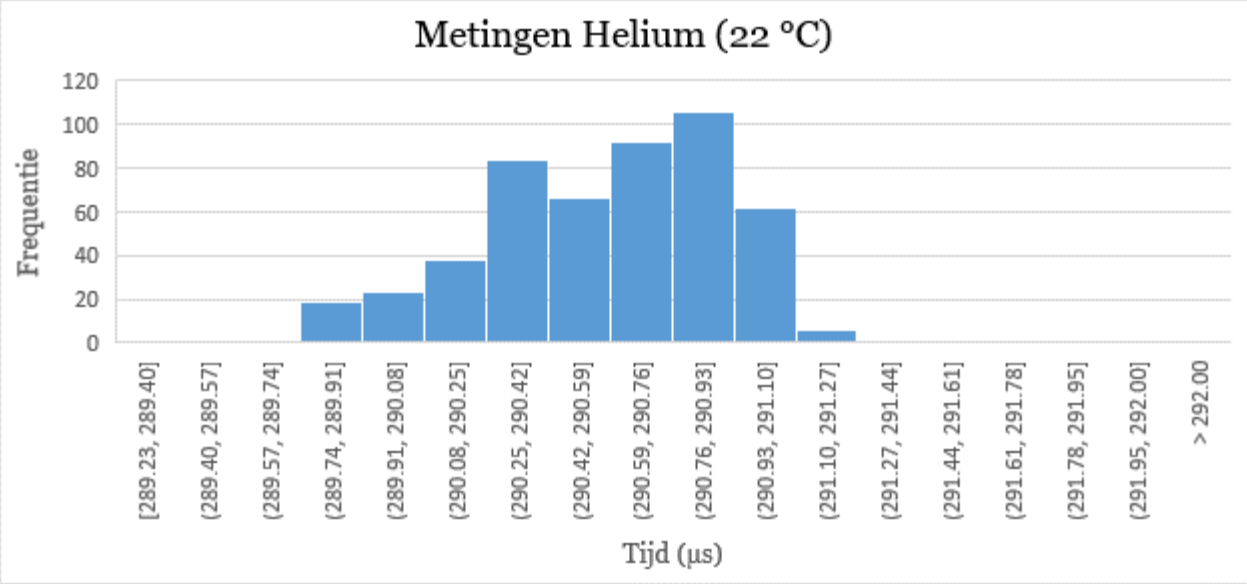
Het statisch minimum is: 210,46 µs. De absolute afwijking is berekend met formule (15.3.3).

(15.3.3) $$Absolute\ afwijking = 3 \cdot \sigma = 3 \cdot 0{,}3826 = 1{,}14\ \mu s$$

De absolute afwijking per meting is: 1,14 µs.

Tabel 21: Frequentietabel metingen Helium

Tijd (µs)	289,8	289,9	290,0	290,1	290,2	290,3	290,4	290,5	290,6	290,7	290,8	290,9	291,0	291,1
Frequentie	3	8	19	14	20	28	43	56	47	47	56	54	62	34



Figuur 51: Histogram metingen Helium

De absolute afwijking is 1,14 µs.

Met formule (15.3.4) is de afwijking van de metingen berekend.

(15.3.4) $$afwijking\ (\%) = \frac{abs\ afwijking}{gemidd\ Helium} \cdot 100 = \frac{1{,}14\ \mu s}{290{,}59\ \mu s} \cdot 100 = 0{,}39\ \%$$

De Heliumzuiverheidsmeter heeft een afwijking van +/- 0,39 % per meting. De Heliumzuiverheidsmeter voldoet hiermee aan de eis om 5% vervuiling in het Helium te meten.

15.4 Drukknop

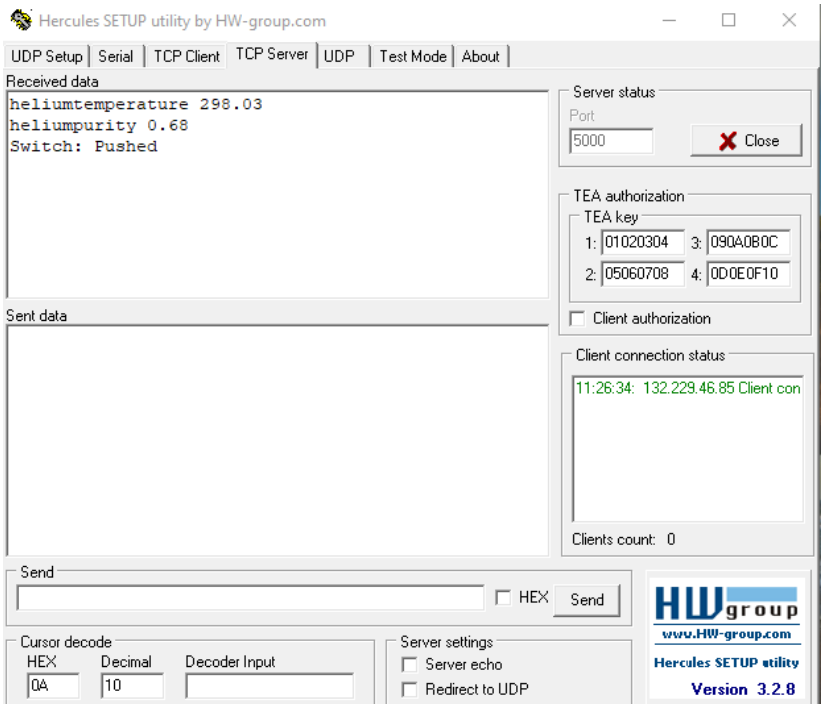
Met de drukknop voor op de behuizing kan het alarm worden uitgeschakeld. Ook wanneer de drukknop wordt ingedrukt wordt de melding “Switch Pushed” (Figuur 52) naar de server verstuurd. De drukknop voldoet hiermee aan de eis om het alarm uitschakelen en de status van de drukknop naar de server versturen.

15.5 Data naar server

De volgende data wordt verstuurd naar de server:

- Temperatuur in de opstelling (K);
- Zuiverheid van het Helium;
- Status van de drukknop.

In Figuur 52 is een screenshot weergeven van de testserver “Hercules”.



Figuur 52: verstuurde data naar testserver

De Heliumzuiverheidsmeter voldoet hiermee aan de eis: Status van de drukknop doorsturen naar de server, Temperatuur in Kelvin doorsturen naar de server en de zuiverheid van het Helium doorsturen naar de server.

15.6 7-segmentsdisplay

Op het 7-segmentsdisplay wordt het volgende weergegeven.

- Zuiverheid van het Helium (0 … 100 %);
- Err Conn;
- Err Sys.

De Heliumzuiverheidsmeter voldoet hiermee aan de eis: “Een 7-segments display waarop de zuiverheid van het Helium wordt weergeven”.

15.7 Alarmering

Met het gebruik van een Piëzo wordt de gebruiker gealarmeerd. Er kunnen 3 alarmen worden gegenereerd:

1. 3x beep;
2. Elke 10 seconde een beep voor 2 minuten lang;
3. 20 keer korte piep daarna elke 10 seconde een beep.

De Heliumzuiverheidsmeter voldoet hiermee aan de eis: “Een alarmering aan de hand van geluidsignalen”.

15.8 Kosten

De totale kosten voor dit project zijn weergegeven in Tabel 22 en Tabel 23

Tabel 22: Kosten componenten hardware

Component	Type	Aantal	Prijs (€)	
Voeding 24V	LH60A24-P1J	1x	24,68	24,68
Voeding 12V	LM7812	1x	1,38	1,38
Voeding 5V	LM7805	1x	1,53	1,53
PCB		1x	50,00	50,00
Solanoid kleppen	6011A	2x	42,13	84,26
microfoon	KECG2740PBJ	1x	2,56	2,56
Weerstanden	0805		~2,00	2,00
Condensator	0805		~2,00	2,00
Zenerdiode	MMSZxxxT1G	2x	0,12	0,24
Diode	1n4007	3x	0,10	0,30
Potmeter	Bourns	1x	0,52	0,52
Led rood	3216	1x	0,17	0,17
Led groen	3216	2x	0,16	0,32
transistor	BC850	3x	0,03	0,09
tweeter	SC 5	1x	11,45	11,45
Opamp	ADA4841	2x	3,65	7,30
MOSFET	IRF7840	3x	0,10	0,30
Voeding	LM7812	1x	1,28	1,28
Voeding	LM7805	1x	1,50	1,50
Totaal:				191,88

Tabel 23: Kosten componenten software

Component	Type:	Aantal:	Prijs per stuk(€):	Totaal(€):
Microcontroller	Teensy 3.6	1x	30,19	30,19
Temperatuur sensor	LM35CAZ	1x	6,16	6,16
Ethernetshield	W5500	1x	7,44	7,44
7-segment display	TM1637	1x	3,31	3,31
Drukknop		1x	0,20	0,20
Behuizing	Schneider electric	1x	110,37	110,37
Totaal:				157,67

De kosten voor het project zijn € 349,55. Hiermee is het doel van het project behaald om een goedkopere oplossing te maken voor van het meten van Helium dan de commerciële oplossing.

16 Advies/verbeteringen

Terugkijkend op het project zijn de volgende aanpassingen aan te bevelen.

- Grotere behuizing: Door gebruik te maken van een grotere behuizing is het makkelijker om storing te zoeken en te meten aan de print;
- Goedkopere behuizing: Door gebruik te maken van een goedkopere behuizing kan aanzienlijk op kosten worden bespaard.
- Meetpinnen: Door het toevoegen van meetpinnen op de pcb kan gemakkelijker storing worden opgezocht;
- Spanningsregulators: De huidige spanningsregulators (LM7812 en LM7805) hebben een lage efficiëntie. Doordat deze een lage efficiëntie hebben worden de regulators warm. Aanbevelen is om gebruik te maken van bijvoorbeeld de: TSR1-2450 en TsR 2-24120;
- Potmeter: De potentiometer kan vervangen worden door een vaste weerstand. Een weerstand van 4 k volstaat. Wanneer er meerdere pcb's gemaakt worden is het beter om een weerstand met een vaste waarde te gebruiken. Op deze manier hoeft niet iedere potentiometer eerst ingesteld worden voordat de zuiverheidsmeter in gebruik genomen kan worden;
- Diode: Door een diode te plaatsen tussen de tussen V_{in} en V_{USB} kan voorkomen worden dat er spanning via de USB-kabel terug naar de USB-poort van de computer kan lopen. De poort kan op deze manier niet beschadigd raken;
- Diode: de diode die de schakeling beschermt tegen het verkeerd aansluiten van de voeding vervangen door een andere schakeling;
- MOSFET: de huidige MOSFET kan een vermogen van 50 W schakelen. Er kan een kleinere MOSFET gebruikt worden voor het schakelen van de kleppen en tweeter.
- Microcontroller: Een kleinere microcontroller. De huidige microcontroller heeft 54 pinnen terwijl er maar 14 pinnen gebruikt worden. Een aanbeveling is om de mogelijk te onderzoeken voor het gebruik van de Teensy 3.2. Deze is kleiner van formaat en goedkoper. De kloksnelheid is echter wel minder;
- Drukknop: Een aansluiting voor de drukknop op de PCB zodat de drukknop eenvoudig aangesloten kan worden;
- Versterker: De versterkingsfactor moet niet te groot gemaakt worden. Wanneer de versterkingsfactor te groot is, wordt ook de ontvangen ruis versterkt. Dit kan leiden tot valse triggers.
- Hysteresespanning: De hysterese spanning is ingesteld op 0,1 V. Door deze te verhogen naar 1 V zal deze schmitt trigger minder triggeren op ruissignalen.
- MAC-adres: Het huidig MAC-adres wordt nu nog opgegeven doormiddel van decimale getallen tussen de 0 en 255. De functie moet aangepast worden zodat deze hexadecimale getallen kan verwerken;
- Onderzoek naar de mogelijkheid om de meter ook de zuiverheid van andere gassen te laten meten.
- Kalibreren: De kalibratie waarden voor 100 % helium en 100 % lucht zijn nu "*hardcoded*" in de microcontroller. Het is aan te bevelen het kalibratieproces door de microcontroller te laten doen en de waardes in de EEPROM op te slaan. Dit is zodat wanneer een nieuwe Heliumzuiverheidsmeter gemaakt is en deze niet identiek is aan de huidige meter, deze via de USB gekalibreerd kan worden. Ook wanneer als gevolg van systeem-drift de nauwkeurigheid afneemt. Kan na verloop van tijd opnieuw worden gekalibreerd.

17 Conclusie

Tijdens het onderzoek om uit te zoeken welke behuizing gebruikt kan worden in het eindontwerp is volgende hypothese gesteld: “Een buis met een grote diameter geeft veel reflecties maar er komt veel signaal aan bij de ontvanger. Het gebruikt van een dunne buis geeft minder reflecties maar zorgt ook voor een verzwakking bij de ontvanger”.

Uit de resultaten is gebleken dat het gebruik van een buis met een kleinere diameter weinig effect heeft op de reflecties (De tijd tot de reflecties zijn uitgedoofd worden niet minder). Wanneer de diameter van de buis, dezelfde diameter heeft als de microfoon geeft dit een versterking bij de ontvanger. Dit is belangrijk omdat in Helium het ontvangen signaal zwakker is dan in lucht. Een manier voor het uitdempen van de reflecties, is door gebruik te maken van een dempend materiaal zoals watten. Maar dat kan vanwege mogelijke heliumverontreiniging in dit geval niet. Daarom zal er een relatief lange tijd tussen de verschillende metingen moeten zitten.

De gestelde hypothese was onjuist. Een buis met een grotere diameter geeft niet meer reflecties en zorgt ook niet voor meer signaal bij de ontvanger. Er is gekozen voor een buis met een kleinere diameter omdat deze meer signaal geeft bij de ontvanger. Ook kan de buis gebogen worden zodat een compacte behuizing gemaakt kan worden. De behuizing afgesteld op de grote van de tweeter om een zo’n klein mogelijke klankkast te krijgen.

Uit het vooronderzoek is gebleken dat de temperatuur invloed heeft op de geluidssnelheid. Wanneer de temperatuur stijgt met 6 graden Celsius zal de geluidssnelheid in Helium toenemen met 10,18 m/s. Dit betekent dat de meting ongeveer 1 % zal afwijken. Door het meten van de temperatuur kan dit worden gecorrigeerd.

Door het gemiddelde te nemen van 100 metingen wordt de kans op statische uitschieters verkleind. De beste manier van meten is om in een korte tijd zoveel mogelijk metingen te doen.

De Heliumzuiverheidsmeter beschikt over de volgende componenten:

- Een 7-segmentsdisplay voor het weergeven van de zuiverheid van het helium;
- Een drukknop voor het uitschakelen van het alarm;
- Een temperatuur sensor voor het meten van de temperatuur in het helium;
- Een ethernet aansluiting voor het versturen van de zuiverheid van het helium en de temperatuur van het helium naar een lokale server;
- Een USB-aansluiting voor het instellen van de microcontroller;
- De zuiverheid van het helium meten met een afwijking van 0,39 %;
- Een behuizing voor het afschermen van de meter.

De Heliumzuiverheidsmeter voldoet hiermee aan de gestelde eisen van de opdrachtgever.

18 Referenties

[1] Mark O. Kimball (9/11/2014). “Introduction to Liquid Helium”, cryogenics and fluids branch [online]. Beschikbaar: https://cryo.gsfc.nasa.gov/introduction/liquid_helium.html#Superfluid%20Helium [Geraadpleegd op: 15/2/2021]

[2] Kenneth Barbalace. (1995 – 2021). “Periodic Table of Elements - Helium – He”. EnvironmentalChemistry.com. [online]. Beschikbaar: https://EnvironmentalChemistry.com/yogi/periodic/He.html [Geraadpleegd op: 15/2/2021].

[3] S. Olfert, M. D. Checkel, and C. R. Koch (2007). “Acoustic method for measuring the sound speed of gases over small path lengths,” [online] . beschikbaar: https://aip.scitation.org/doi/full/10.1063/1.2736406?casa_token=ffNuNotRaCsAAAAA%3AAuDciBjXMRvrDZ-loCQ1fKno6vLjiFtGseVqYp2WcmSwSohWsMleaLvLeuGMr1JYgUytNBAMWPY [Geraadpleegd op: 23/2/2021].

[4] ir. R.E.A. Bouwens, drs. P.A.M. de Groot, drs. W. kranendonk, ir. J.P. van Lune, drs. C.M. Prop-van den berg, J.A.M.H. van riswick, drs. J.J. Westra. BINAS. Pagina: 15, tabel A.

[5] ir. R.E.A. Bouwens, drs. P.A.M. de Groot, drs. W. kranendonk, ir. J.P. van Lune, drs. C.M. Prop-van den berg, J.A.M.H. van riswick, drs. J.J. Westra. BINAS. Pagina: 35, tabel B.

[6] ir. R.E.A. Bouwens, drs. P.A.M. de Groot, drs. W. kranendonk, ir. J.P. van Lune, drs. C.M. Prop-van den berg, J.A.M.H. van riswick, drs. J.J. Westra. BINAS. Pagina: 12.

[7] H. Tijdeman (10/12/2006). “On the propagation of sound waves in cylindrical tubes ,” [online] . beschikbaar: https://www.sciencedirect.com/science/article/pii/S0022460X75802069 [Geraadpleegd op: 28/2/2021].

[8] Visaton. “SC 5 – 8 ohm”. viston.de [online]. beschikbaar: https://www.visaton.de/en/products/drivers/dome-tweeters/sc-5-8-ohm [Geraadpleegd op: 5/3/2021].

[9] Kingstate electronics corp (12/10/2004). “Farnell.com”[online]. beschikbaar: http://www.farnell.com/datasheets/50039.pdf [Geraadpleegd op: 5/3/2021].

[10] Fairchild (September 2014). “mouser.com” [online]. Beschikbaar: https://www.mouser.com/datasheet/2/149/LM7812-461970.pdf [Geraadpleegd op: 15/4/2021].

[11] NXP (01/16/2016). “assets.nexperia.com”[online]. Beschikbaar: https://assets.nexperia.com/documents/data-sheet/BC849_BC850.pdf [Geraadpleegd op: 15/4/2021].

[12] Burkert (20/01/2015) “rs-online.com”[online]. Beschikbaar: https://docs.rs-online.com/57a2/0900766b814069c2.pdf [Geraadpleegd op: 20/4/2021].

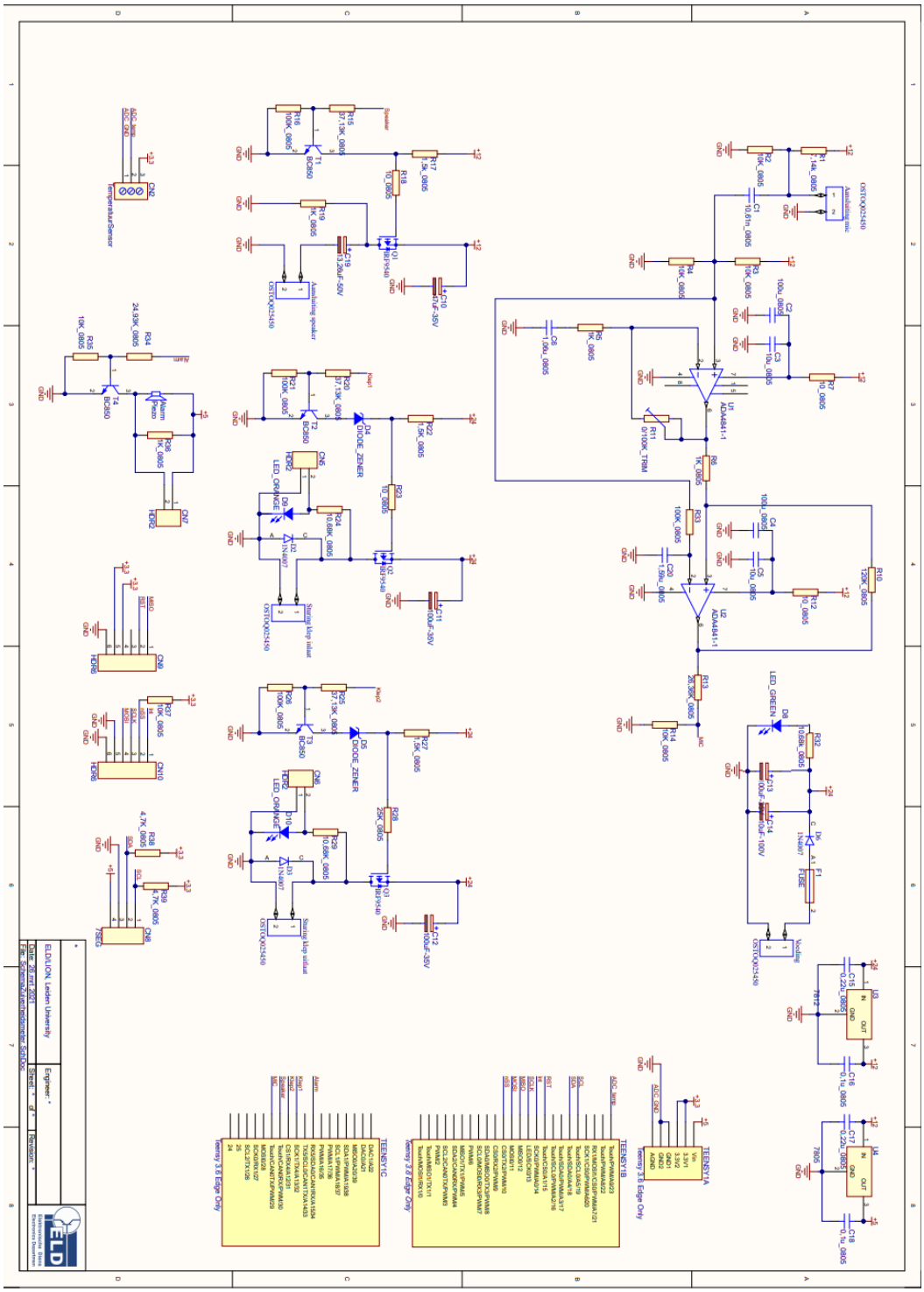
[13] IOR rectifier (23/01/2004) “rs-online.com”[online]. Beschikbaar: https://docs.rs-online.com/3bd4/0900766b8079118c.pdf [Geraadpleegd op: 20/4/2021].

[14] LITE-ON “rs-online.com”[online]. Beschikbaar: https://docs.rs-online.com/ced1/0900766b80dc025c.pdf [Geraadpleegd op: 20/4/2021].

[15] Analog devices (2005-2014) “rs-online.com”[online]. Beschikbaar: https://docs.rs-online.com/ea5b/0900766b810ba956.pdf [Geraadpleegd op: 21/4/2021].

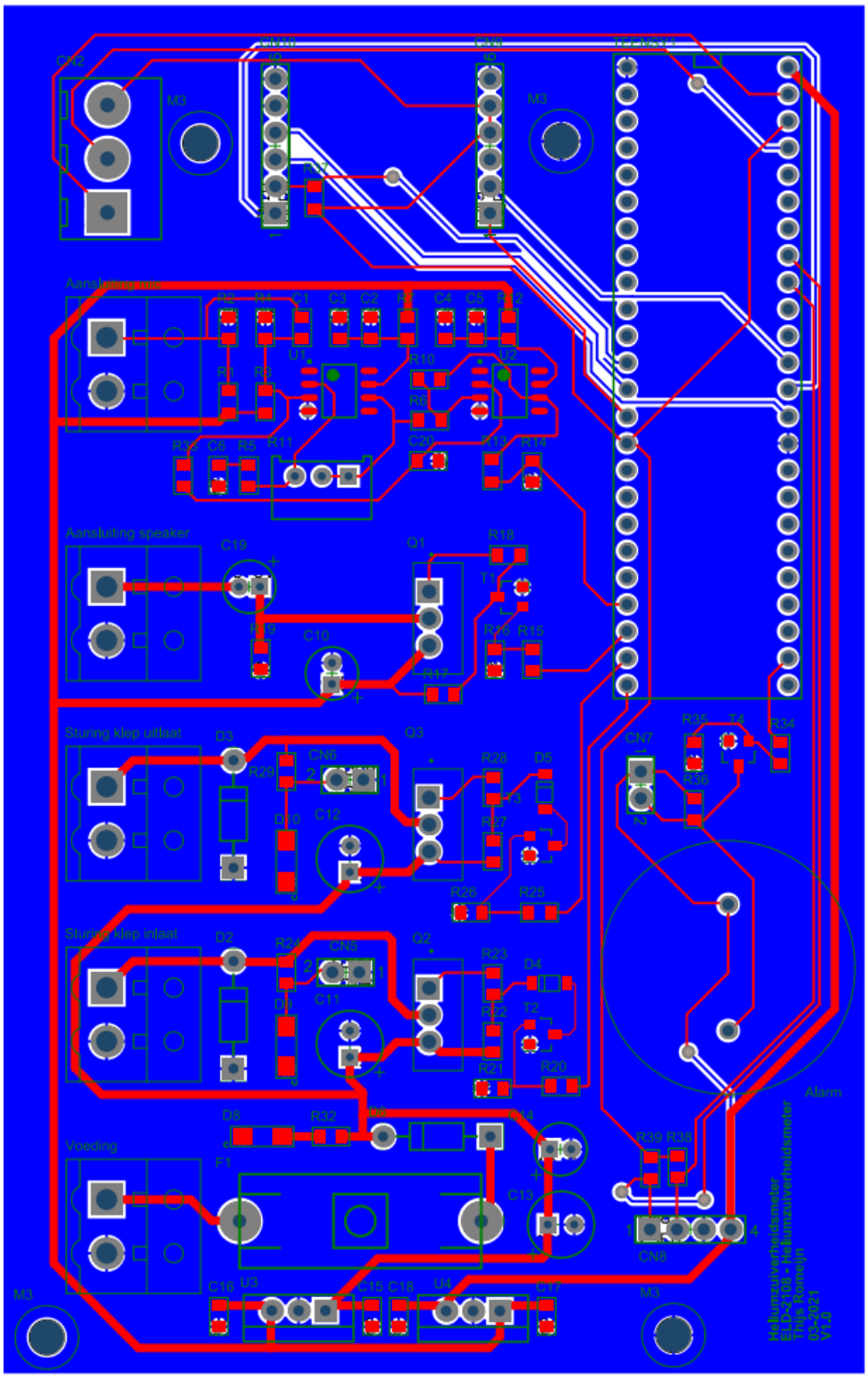
[16] PCB trace width calculator (2018) “www.4pcb.com” [online]. Beschikbaar: https://www.4pcb.com/trace-width-calculator.html [Geraadpleegd op: 3/5/2021].

Bijlage 1: Schema Heliumzuiverheidsmeter

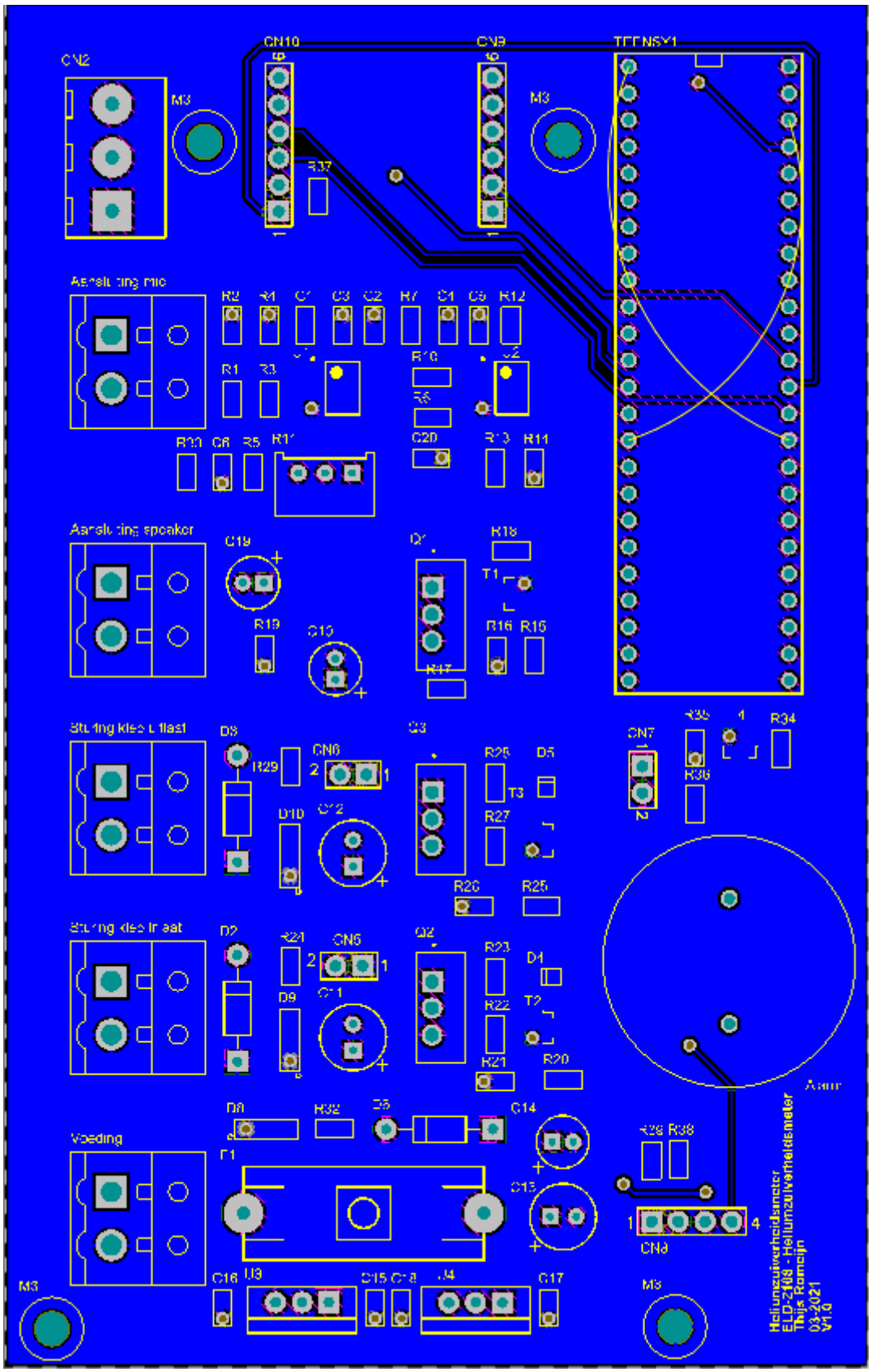


Bijlage 2: PCB Heliumzuiverheidsmeter

PCB bovenlaag



PCB onderlaag



Bijlage 3: Component waarde PCB

Componenten	Gekozen waarde ()	Gemeten waarde ()
R_1	7,5 k	7,53 k
R_2	10 k	10,01 k
R_3	10 k	10,03 k
R_4	10 k	10 k
R_5	1 k	1,003 k
R_6	1 k	999
R_7	10	10
R_{10}	121 k	120,6 k
R_{12}	10	10,1
R_{13}	27,4 k	27,44 k
R_{14}	10 k	10 k
R_{15}	61,9 k	61,82 k
R_{16}	100 k	100,2 k
R_{17}	1,5 k	1,5 k
R_{18}	10	10
R_{19}	1 k	999
R_{20}	61,9 k	61,82 k
R_{21}	100 k	99,9 k
R_{22}	1,5 k	1,49 k
R_{23}	10	10,1
R_{24}	4,32 k	4,32 k
R_{25}	61,9 k	61,75 k
R_{26}	100 k	100,2 k
R_{27}	1,5 k	1,49 k
R_{28}	10	10
R_{29}	4,31 k	4,32 k
R_{32}	4,3 k	4,32 k
R_{33}	100 k	100 k
R_{34}	61,9 k	61,85 k
R_{35}	100 k	99,8 k
R_{36}	1 k	1 k
R_{37}	4,75 k	4,74 k
R_{38}	4,75 k	4,74 k
R_{39}	4,75 k	4,74 k

Componenten	Gekozen waarde ()	Gemeten waarde ()
C_1	21 n	21,88 n
C_2	100 n	92,01 n
C_3	10 u	8,57 u
C_4	100 n	91,82 n
C_5	10 u	8,72 u
C_6	1 u	896 n
C_{10}	100 u	95,54 u
C_{11}	100 u	97,36 u
C_{13}	100 u	95,40 u
C_{14}	10 u	10,22 u
C_{15}	220 n	215 n
C_{16}	100 n	85,35 n
C_{17}	220 n	210 n

C_{18}	100 n	90,89 n
C_{19}	10 u	9,86 u
C_{20}	2 u	2,2 u

Bijlage 4: Bijlage Ontwerp behuizing

MAIN.h

```
/*
    Constants definition for Heliumzuiverheidsmeter
    V0.1
*/

#ifndef MAIN_H
#define MAIN_H

#include <Arduino.h>
#include <stdlib.h>
#include <stdio.h>
#include <Ethernet.h>
#include <Adafruit_I2CDevice.h>                    //need for 7 segmentsdispl
ay
#include <Adafruit_GFX.h>                          //need for 7 segmentsdispl
ay
#include <Adafruit_LEDBackpack.h>                  //need for 7 segmentsdispl
ay

#define AANTALMETINGEN              50             //makes 50 measurements an
d calculates avarage
#define MEETINTERVAL                60000          //1 minuut   (every minute
 an new measurement for purity helium)
#define SLUITKLEPINTERVAL           10000          //10 seconds (time to open
 and close valves)
#define ZUIVERHEIDHELIUM            95
#define BAUDRATE                    115200         //Serial speed

#define IP_SIZE  4
#define NET_SIZE 4
#define GATE_SIZE 4
#define MAC_SIZE 6
#define POORT_SIZE 2
#define MIN_PORT 1
#define MAX_PORT 65535

#define EEPROM_AddressIPTeensy    0
#define EEPROM_AddressMACTeensy   EEPROM_AddressIPTeensy  + IP_SIZE
#define EEPROM_AddressIPServer    EEPROM_AddressMACTeensy + MAC_SIZE
  //not implemented
#define EEPROM_addressNET         EEPROM_AddressIPServer + IP_SIZE

#define EEPROM_addressGATE        EEPROM_addressNET + NET_SIZE
  //not implemented
```

```
#define EEPROM_addressPOORT       EEPROM_addressGATE + GATE_SIZE
  //not implemented

//////////////////////////////////////////////////////Reboot microcontroller

#define CPU_RESTART_ADDR (uint32_t *)0xE000ED0C

#define CPU_RESTART_VAL 0x5FA0004
#define CPU_RESTART (*CPU_RESTART_ADDR = CPU_RESTART_VAL)
#define BAUDRATE 115200
#define BUFSIZE   1000

//////////////////////////////////////////////////////time definition

#define _1SEC_    1000
#define _2SEC_    2000
#define _3SEC_    3000
#define _4SEC_    4000
#define _5SEC_    5000
#define _10SEC_   10000
#define _1MIN_    60000
#define _2MIN_    120000
#define _DebounceDelay_ 500

//////////////////////////////////////////////////////PIN definition

#define PUSHBUTTON           0
#define KLEPINLAATHELIUM     31                          //Pin valve 1
#define KLEPUITLAATHELIUM    32                          //Pin valve 2
#define ALARMPIN             34                          //Pin  piezo alarm
#define SENDPULS             30                          //pin tweeter
#define INTERUPTPINMICROFOON 29                          //pin to detect risin
g flank microphone
const int TEMPERATUURPIN = A9;                           //sensor voor meten t
emperatuur

//////////////////////////////////////////////////////Need for interrupt

extern volatile bool interrupt;
extern volatile bool startmeting;
extern volatile bool savemeting;
extern volatile int AantalInterupts;                      //counts the number
of interrupt
extern volatile unsigned long BeginTijd;
extern volatile unsigned long EindTijd;

//////////////////////////////////////////////////////classes for ethernet and
 7-segment
```

```
extern EthernetClient client;                                   //aanmaken ethernet
class
extern Adafruit_7segment matrix;                                //nodig voor aanstur
en 7-segments display

#endif // main.h
```

MAIN.CPP

```
/*
  Heliumzuiverheidsmeter
  Thijs Romeijn
  2021
  V1.0
*/

#include "main.h"
#include "HeliumPurityMeter.h"
#include <Arduino.h>
#include "commandParser.h"
#include "CommandHandler.h"
#include <SPI.h>
#include <Wire.h>
#include <avr/io.h>                                     //nodig voor instellen van
 interrupt
#include <avr/interrupt.h>                              //nodig voor instellen int
errupt
#include <Adafruit_I2CDevice.h>                         //nodig voor 7 segmentsdis
play
#include <Adafruit_GFX.h>                               //nodig voor 7 segmentsdis
play
#include <Adafruit_LEDBackpack.h>                       //nodig voor 7 segmentsdis
play
#include <Ethernet.h>
#include "eldutil.h"
#include "system.h"
#include <EEPROM.h>
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>
#include <string.h>

volatile bool interrupt             = false;
volatile bool startmeting           = false;
volatile bool savemeting            = false;
```

```
volatile int AantalInterupts        = 0;                    //geeft aan hoeve
el interupt plaats hebben gevonden. reageer alleen op eerste interupt
volatile unsigned long BeginTijd    = 0;
volatile unsigned long EindTijd     = 0;

////////////////////////////////////////////////////////////////////////////
//////////////////

uint8_t g_serial_state;
uint8_t spiBuf[140];
CCommandHandler commandHandler(spiBuf, sizeof(spiBuf));

Adafruit_7segment matrix = Adafruit_7segment();     //nodig voor aansturen 7-
segments display
EthernetClient client;

////////////////////////////////////////////////////////////////////////////
//////////////////Interuptroutine

void ISR()
{                                                   //interupt routine vind pl
aats als microfoon signaal ontvangt
   noInterrupts();                                  //zorg dat geen nieuwe int
erupts plaats kunnen vinden
   ++AantalInterupts;
   if(AantalInterupts == 1 && startmeting == true && savemeting == false)
                    //reageer alleen op eerste interupt
   {
      EindTijd = ARM_DWT_CYCCNT;                //sla de eindtijd op
      interrupt = true;                         //zet vlag omhoog (interupt h
eeft plaats gevonden)
      savemeting = true;
   }
}

////////////////////////////////////////////////////////////////////////////
//////////////////initialize in/outpout pins
void SetPins()
{
    pinMode(PUSHBUTTON,INPUT_PULLUP);
             //push button to stop alarm
    pinMode(ALARMPIN,OUTPUT);
    pinMode(KLEPINLAATHELIUM, OUTPUT);
    pinMode(KLEPUITLAATHELIUM, OUTPUT);
    pinMode(SENDPULS, OUTPUT);
    pinMode(INTERUPTPINMICROFOON, INPUT);
    pinMode(TEMPERATUURPIN, INPUT);
```

```
    attachInterrupt(digitalPinToInterrupt(INTERUPTPINMICROFOON), ISR, RISING);
      //laat interupt plaatsvinden op opgaande flank
}
////////////////////////////////////////////////////////////////////////////
//////////////////Read from EEPROM
void ReadEeprom(uint8_t *data, uint8_t EepromAddress, uint8_t size)
{
  for (int i = 0; i <size; i++)
  {
    data[i] = EEPROM.read(EepromAddress + i);
  }
}
////////////////////////////////////////////////////////////////////////////
//////////////////update EEPROM
void UpdateEeprom(uint8_t *data, uint8_t EepromAddress, uint8_t size)
{
  for (int i = 0; i <size; i++)
  {
    //data[i] = EEPROM.read(EepromAddress + i);
    EEPROM.update(EepromAddress + i, data[i]);
  }
}
////////////////////////////////////////////////////////////////////////////
//////////////////Set ethernetsetting en try to connect
void SetEthernet()
{
    Ethernet.init(10);
    uint8_t IP_Teensy[4];
    ReadEeprom(IP_Teensy, EEPROM_AddressIPTeensy, IP_SIZE);
    IPAddress IPTeensy(IP_Teensy[0], IP_Teensy[1], IP_Teensy[2], IP_Teensy[3])
;

    //get ip address Server
    uint8_t IP_Server[4];
    ReadEeprom(IP_Server, EEPROM_AddressIPServer, IP_SIZE);
    IPAddress IPserver(IP_Server[0], IP_Server[1], IP_Server[2], IP_Server[3])
;

    //byte MAC[] = {0x65,0xdf,0x77,0x7b,0x6d,0x6b};
    uint8_t MAC[6];
    ReadEeprom(MAC, EEPROM_AddressMACTeensy, MAC_SIZE);


    //get port number server
    word poortServer = (EEPROM.read(EEPROM_addressPOORT) << 8) + EEPROM.read(E
EPROM_addressPOORT + 1);

    Ethernet.begin(MAC, IPTeensy);
```

```
    delay(1000);                                    //wait for internet to start
up
    Serial.print("connecting to: ");
    Serial.print(IPserver);
    Serial.println("...");

    if(client.connect(IPserver, poortServer)) {
        Serial.println("connected to:");
        Serial.print("IP server:"); Serial.print(IP_Server[0]);Serial.print(".
");Serial.print(IP_Server[1]);Serial.print(".");Serial.print(IP_Server[2]);Ser
ial.print(".");Serial.println(IP_Server[3]);
        Serial.print("IP Teensy:"); Serial.print(IP_Teensy[0]);Serial.print(".
");Serial.print(IP_Teensy[1]);Serial.print(".");Serial.print(IP_Teensy[2]);Ser
ial.print(".");Serial.println(IP_Teensy[3]);
        Serial.print("MAC: ");
        for (int i = 0; i <6 ; i++)
          {
            Serial.print(MAC[i],HEX);
            Serial.print(".");
          }
          Serial.println("");
          Serial.print("Poort server: ");
          Serial.println(poortServer);
    }
    else
    {
    Serial.println("not connected");
    }
}

void setup() {
    Serial.begin(BAUDRATE);
        //start seriele communicatie

    ARM_DEMCR |= ARM_DEMCR_TRCENA;
      //stel de registers in zodat de clockcycles gelezen kan worden
    ARM_DWT_CTRL |= ARM_DWT_CTRL_CYCCNTENA;
      //stel de registers in zodat de clockcycles gelezen kan worden
    SetPins();
    matrix.begin(0x70);
      //stel het 7 segments display in
    analogReadResolution(16);
      //zet de resolutie van de adc op 16 bits
    SetEthernet();
    delay(1000);
}
```

```
Meter HeliumZuiverheidsMeter;

void ExecSetIP(char *IPAddress)
{
  for (int i = 0; i < 4; i++)
  {
    EEPROM.update(EEPROM_AddressIPTeensy, IPAddress[i]);
    Serial.println(IPAddress[i]);
  }
}

void exec_command_single(char *strResult)
{
  switch (commandHandler.GetCommand()) //Haal het laatst geparste commando op
  {
    case SETIP:
    {
      UpdateEeprom(commandHandler.GetIPv4(),EEPROM_AddressIPTeensy ,IP_SIZE);
      //ExecSetIP(commandHandler.GetIPv4()); //called by reference
      commandHandler.SetCommandAck();
    }
    break;

    case ShowIP:
    {
      uint8_t IP[4];
      ReadEeprom(IP, EEPROM_AddressIPTeensy, IP_SIZE);
      commandHandler.FormatIP(IP, strResult);
      commandHandler.SetCommandAck();
    }
    break;

    case SetPoort:
    {
      u_int16_t poortCMD = commandHandler.GetPoort();   //2 bytes (0-
65535)
      u_int8_t a[2];
      a[0] = highByte(poortCMD);
      a[1] = lowByte(poortCMD);
      EEPROM.update(EEPROM_addressPOORT, a[0]);
      EEPROM.update(EEPROM_addressPOORT + 1, a[1]);
      commandHandler.SetCommandAck();
    }
    break;

    case ShowPoort:
    {
```

```
       word port = (EEPROM.read(EEPROM_addressPOORT) << 8) + EEPROM.read(EEPRO
M_addressPOORT + 1);
       Serial.println(port);
       client.println(port);
       commandHandler.SetCommandAck();
    }
    break;

    case SetIPserver:
    {
       //char *IPAddressServer = commandHandler.GetIPv4();
       UpdateEeprom(commandHandler.GetIPv4(), EEPROM_AddressIPServer, 4);
       commandHandler.SetCommandAck();
    }
    break;

    case ShowIPserver:                                          //le
es EEPROM uit voor ipaddress server
    {
      uint8_t IP[4];
      ReadEeprom(IP, EEPROM_AddressIPServer, 4);
      commandHandler.FormatIP(IP, strResult);
      commandHandler.SetCommandAck();
    }
    break;

    case Reboot:
    {
      commandHandler.SetCommandAck();
      CPU_RESTART;
    }
    break;

    case SetMacaddress:
    {
      UpdateEeprom(commandHandler.GetMAC(), EEPROM_AddressMACTeensy, MAC_SIZE)
;
      commandHandler.SetCommandAck();
    }
    break;

    case GetMACaddress:
    {
      uint8_t MAC[6];
      ReadEeprom(MAC, EEPROM_AddressMACTeensy, MAC_SIZE);
      commandHandler.FormatMAC(MAC, strResult);
      commandHandler.SetCommandAck();
    }
```

```
    break;

    case Reconnect:
    {
       SetEthernet();
       commandHandler.SetCommandAck();
    }

    case CalibrateHelium:
    {
      HeliumZuiverheidsMeter.Kalibreer("Helium");
      commandHandler.SetCommandAck();
    }
    break;

    case CalibrateAir:
    {
        HeliumZuiverheidsMeter.Kalibreer("Air");
        commandHandler.SetCommandAck();
    }
    break;

    case ResponseServer:
    {
        HeliumZuiverheidsMeter.SetResponseServerOK(true);
        commandHandler.SetCommandAck();
    }
    break;

    case ALARM:
    {
      HeliumZuiverheidsMeter.SetResponseServerAlarm(commandHandler.GetAlarm())
;
      HeliumZuiverheidsMeter.SetButtonIsPressed(false);
    }
    break;
  }
}

void poll_serial(void)
{
  static uint8_t charCount = 0;
  static char strCommand[BUFSIZE] = { 0 };
  static char strOldCommand[BUFSIZE] = { 0 };

  if (Serial.available())
  {
    const char c = Serial.read();
```

```
    if (c != 0)
    {
      if (c == '!') // Repeat the previous command but don't execute it yet
      {
        if (charCount == 0 && *strOldCommand)
        {
          strcpy(strCommand, strOldCommand);
          charCount = strlen(strOldCommand);

          if (commandHandler.GetLocalEchoEnabled())
            Serial.print(strCommand);
        }
      }
      else if (c == CR || c == LF)       // if you've gotten to the end of the
 line, process it
      {
        // Linefeed for local echo
        if (commandHandler.GetLocalEchoEnabled())
          Serial.println("");

        // Don't check empty commands
        if (charCount > 0)
        {
          strCommand[charCount] = '\0';

          // Store in old buffer
          strcpy(strOldCommand, strCommand);

          // Reset counter for next round
          charCount = 0;

          // Parse command and get result
          char strResult[BUFSIZE];
          result_code_t result = commandHandler.Execute(strCommand, strResult)
;

          if (result.code == OK)
            exec_command_single(strResult); //get resultstring

          Serial.print(strResult); // Send result
        }
      }
      else if (c == DELETE || c == BACKSPACE) //backspace OR delete (sometimes
 mixed up by terminal programs)
      {
        if (charCount > 0)
        {
          if (commandHandler.GetLocalEchoEnabled())
```

```
          {
            // Backspace
            Serial.write(BACKSPACE);
            // Blank character
            Serial.write(' ');
            // And backspace again since the blank jumps forward
            Serial.write(BACKSPACE);
          }
          charCount--;
        }
      }
      else if (c >= ' ' && c <= '~') // Limit allowed characters
      {
        // Don't overflow + skip leading spaces:
        if (charCount < 140 && !(charCount == 0 && c == ' '))
        {
          strCommand[charCount++] = c;

          if (commandHandler.GetLocalEchoEnabled())
            Serial.write(c);
        }
      }
    }
  }
  else
  {
  }
}

void poll_Ethernet(void)
{
  static uint8_t charCount = 0;
  static char strCommand[BUFSIZE] = { 0 };
  static char strOldCommand[BUFSIZE] = { 0 };

  if (client.available())
  {
    const char c = client.read();
    if (c != 0)
    {
      if (c == '!') // Repeat the previous command but don't execute it yet
      {
        if (charCount == 0 && *strOldCommand)
        {
          strcpy(strCommand, strOldCommand);
          charCount = strlen(strOldCommand);

          //if (commandHandler.GetLocalEchoEnabled())
```

```
          // Serial.print(strCommand);
        }
      }
      else if (c == CR || c == LF)       // if you've gotten to the end of the
 line, process it
      {
        // Linefeed for local echo
        if (commandHandler.GetLocalEchoEnabled())
         // Serial.println("");

        // Don't check empty commands
        if (charCount > 0)
        {
          strCommand[charCount] = '\0';

          // Store in old buffer
          strcpy(strOldCommand, strCommand);

          // Reset counter for next round
          charCount = 0;

          // Parse command and get result
          char strResult[BUFSIZE];
          result_code_t result = commandHandler.Execute_PollEthernet(strComman
d, strResult);
          if (result.code == OK)
            exec_command_single(strResult); //get resultstring

          //Serial.print(strResult); // Send result
        }
      }
      else if (c == DELETE || c == BACKSPACE) //backspace OR delete (sometimes
 mixed up by terminal programs)
      {
        if (charCount > 0)
        {
          if (commandHandler.GetLocalEchoEnabled())
          {
            // Backspace
            Serial.write(BACKSPACE);
            // Blank character
            Serial.write(' ');
            // And backspace again since the blank jumps forward
            Serial.write(BACKSPACE);
          }
          charCount--;
        }
      }
```

```
      else if (c >= ' ' && c <= '~') // Limit allowed characters
      {
        // Don't overflow + skip leading spaces:
        if (charCount < 140 && !(charCount == 0 && c == ' '))
        {
          strCommand[charCount++] = c;

         // if (commandHandler.GetLocalEchoEnabled())
           // Serial.write(c);
        }
      }
    }
  }
  if (!client.connected())             //if not connected try to reconnect
  {
    static unsigned long TimerNoEthernet = millis();
    if(millis() - TimerNoEthernet > _1MIN_)
    {
      TimerNoEthernet = millis();
      SetEthernet();
    }
  }
}

void poll_Button()
{
 static unsigned long startMillissecTimerButton = millis();
 bool stateButton = digitalRead(PUSHBUTTON);

 if((stateButton == false) && (millis() - startMillissecTimerButton) > _Debou
nceDelay_)
 {
   if(stateButton == false)
   {
    HeliumZuiverheidsMeter.SetButtonIsPressed(true);
    client.print("Switch: ");
    client.println("Pushed");
    startMillissecTimerButton = millis();
   }
 }
}

void loop()
{
  poll_serial();
  poll_Ethernet();
  poll_Button();
  HeliumZuiverheidsMeter.FSM(HeliumZuiverheidsMeter.GetResponseServerOK());
```

```
  HeliumZuiverheidsMeter.Alarm(HeliumZuiverheidsMeter.GetResponseServerAlarm()
,HeliumZuiverheidsMeter.GetButtonIsPressed());
  delay(1);            //needs when no usb is connected
}
```

HeliumPurityMeter.h

```
#include <Arduino.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

enum State
{                                                  //states voor statemachine
metingen
    State_Start,
    State_KlepOpen,
    State_Timer1,
    State_Timeout1,
    State_Timer2,
    State_KlepDicht,
    State_StartMeting,
    State_SaveMeting,
    State_NeemGemiddeld,
    State_PrintOp7Seg,
    State_Alarm,
    State_PrintConnErr,
    State_SendToServer,
    State_GetResponseServer,
    State_Server_Timeout,
    State_PrintSysErr,
    State_Reconnect
};

enum AlarmState
{                                                //states voor statemachine metin
gen
    Alarm_Off,
    Beep_3x,
    Beep_10sec_2min,
    Beep_20x_thenEvery_10sec
};

enum PrintERR
{
  NO_ERR,
```

```
  ERR_SYS,
  ERR_CONN,
  ERR_CONNenSYS
};


class Meter
{
    public:
        Meter();                              //constructor
        u_int32_t GetValueAir()             const       { return WaardeLucht;
}
        u_int32_t GetValueHelium()          const       { return WaardeHelium;
 }
        u_int32_t GetResponseServerAlarm()  const       { return ResponseServe
rAlarm; }
        u_int32_t CalculategemiddeldeOverMeting();
        u_int32_t GetgemiddeldeOverMeting() const       {return GemiddeldeAant
alPulsen; }
        float GetGemiddeldeTemperatuur()    const       {return Gemiddeldetemp
eratuur;}
        float GetCalibrateTempurature()     const       { return Calibratedtem
perature; }
        float Getpercentage()               const;
        float CalculategemiddeldeTemperatuur();
        float GetTemperatuur()              const;                  //geeft hui
dige temperatuur terug in Celcius
        float GetTemperatuurKelvin()        const;
        bool GetButtonIsPressed()           const       { return ButtonIsPress
ed; }
        bool GetResponseServerOK()          const       { return responseServe
rOK; }
        float TemperatureCorrectionFactor();

        void SetResponseServerAlarm(u_int8_t alarm);
        void SetResponseServerOK(bool response);
        void SetButtonIsPressed(bool pressed);
        void SetCalibrateTempurature(float temp);
        void SetPercentage();
        void SetvalueAir(int air);
        void SetvalueHelium(int helium);
        void SetgemiddeldeOverMeting(unsigned long aantalpulsen);
        void SetgemiddeldeTemperatuur(float temperatuur);

        void SturingKlep(String sturing);                               //aans
turing klep voor inlaat Helium (parameter "open" = open, "dicht" = close )
        void Alarm(AlarmState sturing, bool IsButtonPressed);          //aanst
uring alarm
        void PrintTo7SEG(PrintERR alarm);
```

```
        void TemperatuurOp7Seg();                                       //geef
t temperatuur op 7 segmentsdisplay
        void StartMeting();                                             //star
t de meting (uitsturen puls tweeter)
        void SaveMeting();
        void SaveTemperatuur();
        void PrintGemiddeldeTemperatuur();                              //prin
t gemiddelde temperatuur af op 7 segments display
        void Kalibreer(String HeliumOrAir);
            //neem 100 metingen en sla gemiddelde daarvan op als nulpunt lucht
        void FSM(bool ResponseServer);

    private:
        u_int32_t WaardeLucht                   =       140282;
        u_int32_t WaardeHelium                  =       52295;
        u_int8_t ResponseServerAlarm            =       0;
        bool ButtonIsPressed                    =       false;
   //active high (pull up)
        bool printError                         =       false;
        bool responseServerOK                   =       false;
        float percentage                        =       0;
        float CorrectionFactor                  =       0;
        float Calibratedtemperature             =       21.0;
        u_int32_t Aantalmetingen                =       0;
        float Gemiddeldetemperatuur             =       0;
        float GemiddeldeAantalPulsen            =       0;
};
```

HeliumPurityMeter.cpp

```
#include "HeliumPurityMeter.h"
#include "main.h"
#include <Arduino.h>
#include <SPI.h>
#include <Wire.h>
#include <avr/io.h>                         //nodig voor instellen van interrup
t
#include <avr/interrupt.h>                          //nodig voor instellen int
errupt
#include <Adafruit_I2CDevice.h>                     //nodig voor 7 segmentsdis
play
#include <Adafruit_GFX.h>                           //nodig voor 7 segmentsdis
play
#include <Adafruit_LEDBackpack.h>                   //nodig voor 7 segmentsdis
play
#include <Ethernet.h>
#include "eldutil.h"
```

```cpp
#include "system.h"
#include <EEPROM.h>
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>
#include <string.h>

Meter::Meter()
{
  //read value air and helium out of eeprom
}

u_int32_t Meter::CalculategemiddeldeOverMeting()
{
  return GemiddeldeAantalPulsen /= Aantalmetingen;
}

float Meter::CalculategemiddeldeTemperatuur()
{
  return Gemiddeldetemperatuur /= Aantalmetingen;
}

void Meter::SetvalueAir(int air)
{
  WaardeLucht = air;
}

void Meter::SetvalueHelium(int helium)
{
  WaardeHelium = helium;
}

void Meter::SetgemiddeldeOverMeting(unsigned long aantalpulsen)
{
   GemiddeldeAantalPulsen += aantalpulsen;
}

void Meter::SetgemiddeldeTemperatuur(float temperatuur)
{
  Gemiddeldetemperatuur += temperatuur;
}

void Meter::SetButtonIsPressed(bool pressed)
{
  ButtonIsPressed = pressed;
}

void Meter::SetResponseServerAlarm(u_int8_t alarm)
```

```
{
  ResponseServerAlarm = alarm;
}

void Meter::SetResponseServerOK(bool response)
{
  responseServerOK = response;
}

float Meter::Getpercentage() const
{
  return percentage;
}

void Meter::SetPercentage()
{
    percentage = ((-0.001129552 * GemiddeldeAantalPulsen) + 159.0683665);
    float correctedpercentage = TemperatureCorrectionFactor();
    Serial.println(correctedpercentage);
    if(percentage >= 100)
    {
      percentage = 100;
    }
    else if(percentage <= 0)
    {
      percentage = 0;
    }
}

float Meter::TemperatureCorrectionFactor()
{
  return Getpercentage() - ((GetGemiddeldeTemperatuur() - GetCalibrateTempurat
ure()) * 0.017);
}

void Meter::SturingKlep(String sturing)
{
    if (sturing == "open"){
        digitalWrite(KLEPINLAATHELIUM, HIGH);
        digitalWrite(KLEPUITLAATHELIUM, HIGH);
        delay(20);                              //wacht tot klep volledig geop
end is
    }
    else if (sturing == "dicht"){
        digitalWrite(KLEPINLAATHELIUM, LOW);
        digitalWrite(KLEPUITLAATHELIUM, LOW);
        delay(20);                             //wacht tot klep volledig geslo
ten is
```

```
    }
}

void Meter::Alarm(AlarmState sturing, bool IsButtonPressed)
{
static bool beep1 = 1;
static int  beep2 = 1;

if(GetButtonIsPressed() == true)
  {
   digitalWrite(ALARMPIN,LOW);
   beep1 = 1;
   beep2 = 1;
  }
else
{
 switch(sturing)
  {
    case Alarm_Off:
        beep1 = 1;
        beep2 = 1;
        digitalWrite(ALARMPIN,LOW);
    break;

    case Beep_3x:
    {
      if(beep1 == true)
      {
        beep1 = false;
        static int i;
        for(i = 0 ; i < 3 ; i++)
        {
            digitalWrite(ALARMPIN,HIGH);
            delay(100);
            digitalWrite(ALARMPIN,LOW);
            delay(100);
        }
      }
    }
    break;

  case Beep_10sec_2min:
  {
    static unsigned long startMillissec = millis();
    static unsigned long startMillismin = millis();

    if(millis() - startMillismin < _2MIN_)
    {
```

```
      if(millis() - startMillissec > _10SEC_)
        {
         digitalWrite(ALARMPIN,HIGH);
         delay(100);
         digitalWrite(ALARMPIN,LOW);
         startMillissec = millis();
        }
    }
    else
    {
      digitalWrite(ALARMPIN,LOW);
      startMillismin = millis();
    }
  }

  break;

  case Beep_20x_thenEvery_10sec:
  {
    static int i;
    static unsigned long startMillissec = millis();
    if(beep2 == 1)
    {
      for (i = 0 ; i < 10 ; i++)
      {
        digitalWrite(ALARMPIN,HIGH);
        delay(50);
        digitalWrite(ALARMPIN,LOW);
        delay(50);
      }
      beep2 = 2;
    }
    else if(beep2 == 2)
    {
     if(millis() - startMillissec > _10SEC_)
        {
        digitalWrite(ALARMPIN,HIGH);
        delay(100);
        digitalWrite(ALARMPIN,LOW);
        startMillissec = millis();
      }
    }
  }
  break;
  }
 }
}
```

```
void Meter::Kalibreer(String HeliumOrAir)
{
  if(HeliumOrAir == "Air")
  {
  //stuur kleppen open
  unsigned long GemiddeldAantalMetingen = 0;
  SturingKlep("open");
  delay(10000);
  SturingKlep("dicht");
  delay(100);
    for(int i = 0 ; i < 50 ; ++i)
    {
      BeginTijd = ARM_DWT_CYCCNT;
      digitalWrite(SENDPULS, HIGH);
      delayMicroseconds(10);
      while(digitalRead(INTERUPTPINMICROFOON) == 0){}   //wait for interrupt

      EindTijd = ARM_DWT_CYCCNT;                        //save endtime
      unsigned long totaleTijd = EindTijd - BeginTijd;
      //Serial.println(totaleTijd);
      GemiddeldAantalMetingen += totaleTijd;            //save average
      delay(50);
      digitalWrite(SENDPULS, LOW);
      delay(50);
    }
    //Serial.print("Gemiddeld: ");
    GemiddeldAantalMetingen /= 50;
    //Serial.println(GemiddeldAantalMetingen);
    SetvalueAir(GemiddeldAantalMetingen);
    Serial.println(GetValueAir());
    //write to eeprom
  }
  else if(HeliumOrAir == "Helium")
  {
  unsigned long GemiddeldAantalMetingen = 0;
  SturingKlep("open");
  delay(10000);
  SturingKlep("dicht");
  delay(100);
    for(int i = 0 ; i < 50 ; ++i)
    {
      BeginTijd = ARM_DWT_CYCCNT;
      digitalWrite(SENDPULS, HIGH);
      delayMicroseconds(10);
      while(digitalRead(INTERUPTPINMICROFOON) == 0){}   //wait for interrupt

      EindTijd = ARM_DWT_CYCCNT;                        //save endtime
      unsigned long totaleTijd = EindTijd - BeginTijd;
```

```
      //Serial.println(totaleTijd);
      GemiddeldAantalMetingen += totaleTijd;            //save average
      delay(50);
      digitalWrite(SENDPULS, LOW);
      delay(50);
    }
    //Serial.print("Gemiddeld: ");
    GemiddeldAantalMetingen /= 50;
    //Serial.println(GemiddeldAantalMetingen);
    SetvalueHelium(GemiddeldAantalMetingen);
    Serial.println(GetValueHelium());
    Serial.println("Helium calibrate");
    //write to eeprom
  }
  else
  {

  }
}

void Meter::SaveMeting()
{
  if(interrupt == true && startmeting == true && savemeting == true)    //heef
t de interupt plaats gevonden ?
    {
        savemeting = false;
        unsigned long totaleTijd = EindTijd - BeginTijd;          //totale tij
d is eindtijd - begintijd tijd is in klokpulsen
        //Serial.println(totaleTijd);
        SetgemiddeldeOverMeting(totaleTijd);
        SaveTemperatuur();                          //meet de temperatuur na h
et meten van tijd
        delay(50);                                  //wacht op uitdenderen van
 signaal
        startmeting = false;                        //zet de vlag omhoog zodat
 nieuwe meting gestart kan worden
        interrupt = false;
        Aantalmetingen++;                           //telt aantal metingen

        digitalWrite(SENDPULS, LOW);                //Zet de puls uit

    }
}

void Meter::StartMeting()
{
if(startmeting == false && interrupt == false && savemeting == false) //zorgt
dat geen meting gestart kan worden als al een meting loopt
```

```
    {
        startmeting = true;
        digitalWrite(SENDPULS, HIGH);                           //Zend Puls u
it
        BeginTijd = ARM_DWT_CYCCNT;                             //sla de huig
e cpu cycle op
        delayMicroseconds(10);
        AantalInterupts = 0;                                    //resetten aa
ntal interups in serverroutine zorgt dat alleen de eerste interupt herkent wor
dt
        interrupt = false;                                      //Interrupt h
eeft plaats gevonden
        savemeting = false;
       interrupts();
    }
}

void Meter::TemperatuurOp7Seg()
{
    float temperatuur = analogRead(TEMPERATUURPIN);                     //le
es de waarde van de analoge pin uit
    temperatuur = ((temperatuur * (3300.0/65536.0)) / 10.0);        //3300 = s
panning teensy, 65536 = resolutie adc, 10 = mV per graden
    matrix.print(temperatuur,1);                                    //print wa
arde op 7 segmentsdisplay ,1 = 1 decimaal
    matrix.writeDisplay();
}

float Meter::GetTemperatuur() const
{
    float temperatuur = analogRead(TEMPERATUURPIN);
    return temperatuur = ((temperatuur * (3315.0/65536.0)) / 10.0);
}

float Meter::GetTemperatuurKelvin() const
{
  return Gemiddeldetemperatuur + 273.15;          //gives temperature in kelvi
n
}

void Meter::PrintGemiddeldeTemperatuur()
{
    Serial.print("Gemiddelde temperatuur = ");
    Serial.println(Gemiddeldetemperatuur);
    matrix.print(Gemiddeldetemperatuur,1);                      //print gemidd
elde temperatuur af op 7 segments display
    matrix.writeDisplay();
}
```

```
void Meter::SaveTemperatuur()
{
    float temperatuur = analogRead(TEMPERATUURPIN);
    temperatuur = ((temperatuur * (3315.0/65536.0)) / 10.0);
    SetgemiddeldeTemperatuur(temperatuur);
    //Serial.println(temperatuur);
    delay(10);            //wait for ADC to get ready
}

void Meter::PrintTo7SEG(PrintERR alarm)
{
 static unsigned long  startMillis = millis();

  switch(alarm)
  {
    case NO_ERR:
    {
      if(Getpercentage() <= 0)
      {
        matrix.writeDigitRaw(0,B00000000);
        matrix.writeDigitRaw(1,B00000000);
        matrix.writeDigitRaw(3,B00000000);
        matrix.writeDigitRaw(4,B00000000);
      }
      else
      {
        matrix.print(Getpercentage(),1);
      }
    }
    break;

    case ERR_SYS:
    {
      if ((millis() - startMillis) >= _1SEC_)  //test whether the period has e
lapsed
        {
          matrix.print(Getpercentage(),1);
        }
      if ((millis() - startMillis) >= _2SEC_)  //test whether the period has e
lapsed
        {
         matrix.writeDigitRaw(0,B01111001);      //E
         matrix.writeDigitRaw(1,B01010000);      //r
         matrix.writeDigitRaw(3,B01010000);      //r
         matrix.writeDigitRaw(4,B00000000);      //
        }
```

```
      if ((millis() - startMillis) >= _3SEC_)  //test whether the period has e
lapsed
        {
         matrix.writeDigitRaw(0,B01101101);      //s
         matrix.writeDigitRaw(1,B01101110);      //y
         matrix.writeDigitRaw(3,B01101101);      //s
         matrix.writeDigitRaw(4,B00000000);
         startMillis = millis();
        }
    }
    break;

    case ERR_CONN:
    {
      if ((millis() - startMillis) >= _1SEC_)  //test whether the period has e
lapsed
        {
          matrix.print(Getpercentage(),1);
        }

        if ((millis() - startMillis) >= _2SEC_)  //test whether the period has
 elapsed
        {
         matrix.writeDigitRaw(0,B01111001);      //E
         matrix.writeDigitRaw(1,B01010000);      //r
         matrix.writeDigitRaw(3,B01010000);      //r
         matrix.writeDigitRaw(4,B00000000);      //
        }
        if ((millis() - startMillis) >= _3SEC_)  //test whether the period has
 elapsed
        {
          matrix.writeDigitRaw(0,B00111001);      //C
          matrix.writeDigitRaw(1,B01011100);      //O
          matrix.writeDigitRaw(3,B01010100);      //N
          matrix.writeDigitRaw(4,B01010100);      //N
          startMillis = millis();
        }
    }
    break;

    case ERR_CONNenSYS:
    {
    if ((millis() - startMillis) >= _1SEC_)  //test whether the period has ela
psed
      {
        matrix.print(percentage,1);
      }
```

```
    if ((millis() - startMillis) >= _2SEC_)  //test whether the period has ela
psed
      {
       matrix.writeDigitRaw(0,B01111001);      //E
       matrix.writeDigitRaw(1,B01010000);      //r
       matrix.writeDigitRaw(3,B01010000);      //r
       matrix.writeDigitRaw(4,B00000000);      //
      }

   if ((millis() - startMillis) >= _3SEC_)  //test whether the period has elap
sed
      {
        matrix.writeDigitRaw(0,B00111001);      //C
        matrix.writeDigitRaw(1,B01011100);      //O
        matrix.writeDigitRaw(3,B01010100);      //N
        matrix.writeDigitRaw(4,B01010100);      //N

      }
   if ((millis() - startMillis) >= _4SEC_)  //test whether the period has elap
sed
      {
        matrix.writeDigitRaw(0,B01111001);      //E
        matrix.writeDigitRaw(1,B01010000);      //r
        matrix.writeDigitRaw(3,B01010000);      //r
        matrix.writeDigitRaw(4,B00000000);      //

      }
   if ((millis() - startMillis) >= _5SEC_)  //test whether the period has elap
sed
      {
        matrix.writeDigitRaw(0,B01101101);      //s
        matrix.writeDigitRaw(1,B01101110);      //y
        matrix.writeDigitRaw(3,B01101101);      //s
        matrix.writeDigitRaw(4,B00000000);
        startMillis = millis();
      }
    }
    break;

    default:
     {
       Serial.println("Fuck something broke!");
     }
    break;
  }
   matrix.writeDisplay();
}
```

```
void Meter::SetCalibrateTempurature(float temp)
{
  Calibratedtemperature = temp;
}

void Meter::FSM(bool ResponseServerOK)
{
  static State HuidigeState = State::State_Start;
  static PrintERR PrintErr = PrintERR::NO_ERR;
  static unsigned long startMillissecTimerKlep = millis();
  static unsigned long startMillissecTimerFSM = millis();
  static unsigned long startMillissecTimeOutFSM = millis();

    switch(HuidigeState)
    {
        case State_Start:
        {
          if(millis() - startMillissecTimerFSM > MEETINTERVAL)         //start
 measurement purity helium after a interval
          {
            Gemiddeldetemperatuur = 0;                                  //rese
t values
            GemiddeldeAantalPulsen = 0;
            Aantalmetingen = 0;
            HuidigeState = State::State_KlepOpen;                       //next
 state
         }
        }
        break;

        case State_KlepOpen:
        {
            SturingKlep("open");                                        //open
valve
            startMillissecTimerKlep = millis();                         //start
 timer
            HuidigeState = State::State_Timer1;                         //next
state
        }
        break;

        case State_Timer1:
        {
           if((millis() - startMillissecTimerKlep) > SLUITKLEPINTERVAL)   //cl
ose valve after a time interval
            {
               HuidigeState = State::State_KlepDicht;                   //ne
xt state
```

```
            }
        }
        break;

        case State_KlepDicht:
        {
            SturingKlep("dicht");                                   //clo
se valve
            PrintErr = PrintERR::NO_ERR;                            //res
et 7-segments
            startMillissecTimeOutFSM = millis();                    //sta
rt timer voor timeout
            HuidigeState = State::State_StartMeting;
        }
        break;

        case State_StartMeting:
        {
            delay(10);
            StartMeting();                                          //send
 puls
            HuidigeState = State::State_SaveMeting;
        }
        break;

        case State_SaveMeting:
        {
            SaveMeting();                                           //save
 measurements
            delay(50);                                              //wait
 for reflections to demp
            if(Aantalmetingen >= AANTALMETINGEN)
            {
               noInterrupts();
               HuidigeState = State::State_NeemGemiddeld;
            }
            else
            {
               HuidigeState = State::State_Timeout1;
            }
        }
        break;

        case State_Timeout1:
        {
         if((millis() - startMillissecTimeOutFSM) > _1MIN_)       //if 1 minut
e past restart meting
          {
```

```
            HuidigeState = State::State_Start;
          }
        else
        {
          HuidigeState = State::State_StartMeting;
        }
        }
        break;

        case State_NeemGemiddeld:                                   //neem
 gemiddeld van tijd tussen zender en ontvanger en neem gemiddelde temperatuur
        {
          CalculategemiddeldeTemperatuur();                         //calc
ulate percentage
          CalculategemiddeldeOverMeting();                          //calc
ulate average measurements
          SetPercentage();
          /*                                                        //use
for debugging
          Serial.print("Gemiddelde Meting = ");
          Serial.println(GetgemiddeldeOverMeting());
          Serial.print("Gemiddelde Temperatuur = ");
          Serial.println(GetGemiddeldeTemperatuur());

          Serial.print("Percentage = ");
          Serial.println(Getpercentage());
          */
          HuidigeState = State::State_PrintOp7Seg;
        }
        break;

        case State_PrintOp7Seg:                                      //pr
int percentage op 7 segment display
        {
         if(percentage <= ZUIVERHEIDHELIUM)
          {
            PrintErr = PrintERR::ERR_SYS;
          }
         if(client.connected())                                      //con
nected to server
          {
            HuidigeState = State::State_SendToServer;
          }

         if(!client.connected())                                    //not
connected to server
          {
            HuidigeState = State::State_PrintConnErr;
```

```
          }
        }
        break;

        case State_PrintSysErr:
        {
          startMillissecTimerFSM = millis();
          PrintErr = PrintERR::ERR_CONNenSYS;
          HuidigeState = State::State_Start;

        }
        break;

        case State_PrintConnErr:                                    //when not
connected to server
        {
          PrintErr = PrintERR::ERR_CONN;
          if(percentage >= ZUIVERHEIDHELIUM)                        //when heli
um purity higher then 95 % start new measurements
          {
            startMillissecTimerFSM = millis();
            HuidigeState = State::State_Start;

          }
         else if(percentage <= ZUIVERHEIDHELIUM)                    //when heli
um purity lower then 95 % start print error
          {
            HuidigeState = State::State_PrintSysErr;
          }
        }
        break;

        case State_SendToServer:                                     //send pu
rity to server
        {
          client.print("heliumtemperature ");
          client.println(GetTemperatuurKelvin());
          delay(10);
          client.print("heliumpurity ");
          client.println(percentage);

          HuidigeState = State::State_GetResponseServer;
        }
        break;

        case State_GetResponseServer:
        {
```

```
          if(ResponseServerOK == true)                              //received "OK
" from server ?
           {
            startMillissecTimerFSM = millis();                      //reset timer
            HuidigeState = State::State_Start;                      //start new me
asurements
           }

          static bool timeout = false;
          static unsigned long  TIMEOUT = millis();
          if((millis() - TIMEOUT) > _1MIN_ && timeout == false)   //send only
once
             {
              timeout = true;
              HuidigeState = State::State_Server_Timeout;          //resend dat
a to server
             }
          if((millis() - TIMEOUT) > _2MIN_)                         //after 2 mi
nutes restart measurements
             {
              timeout = false;
              TIMEOUT = millis();
              HuidigeState = State::State_PrintConnErr;
             }
           SetResponseServerOK(false);                     //reset response se
rver
        }
        break;

        case State_Server_Timeout:
        {
         HuidigeState = State::State_SendToServer;
        }
        break;

        default:
        {
          Serial.println("Fuck something broke!");
        }
        break;
  }
   PrintTo7SEG(PrintErr);
}
```

CommandHandler.cpp

```
/*
```

```
  Command Handler Class
  (C) Copyright 2019-2021 ELD/LION, Leiden University

  Written by       : ing. A.C.J. van Amersfoort
  Dependencies     : CommandHandler, eldutil, eldavrutil
  Initial date     : August 22, 2019
  Last modified    : January 5, 2021
*/

#include "CommandHandler.h"
#include "eldutil.h"
#include "eldavrutil.h"
#include "system.h"

#include <string.h>

// Fixed device IDN
const char DEVICE_IDN_P[] PROGMEM = "ELD-2108";

// Firmware version string
const char VER_STR_P[] PROGMEM = "ELD-
2108 - heliumpuritymeter" VERSION " - (C) 2020-
2021 ELD/LION, Leiden University";

const char HELP_STR_P[] PROGMEM = "\n\r"
                                  "help                                  : This s
creen\n\r"
                                  "!                                     : Repeat
 previous command\n\r"
                                  "echo [on|off]                         : Turn l
ocal echo on/off\n\r"
                                  "idn                                   : Show d
evice ID/serial\n\r"
                                  "ver                                   : Show d
evice version\n\r"
                                  "setip                                 : Set IP
 address\n\r"
                                  "Showip                                : gives
ip address teensy\n\r"
                                  "Showmac                               : gives
MAC address teensy\n\r"
                                  "setmac                                : Set MA
C address teensy\n\r"
                                  "setport                               : set po
or server\n\r"
                                  "Showport                              : show p
ort server\n\r"
```

```
                                    "setipserver                        : set ip
 address server\n\r"
                                    "showipserver                       : show i
p address server\n\r"
                                    "Reconnect                          : Reconn
ect to server\n\r"
                                    "reboot                             : reboot
 teensy to load new settings\n\r"
                                    ;


// Ctor
CCommandHandler::CCommandHandler(uint8_t *pSendBuf, const size_t iSendBufSize)
{
  m_pSendBuf = pSendBuf;
  m_iSendBufSize = iSendBufSize;
  m_bLocalEcho = true;
  m_iDelayLineData = 0x00;
}


void CCommandHandler::SendBufReset(void)
{
  m_iSendBufCount = 0;
}


bool CCommandHandler::SendBufPutByte(const uint8_t dataByte)
{
  // Buffer size check
  if (m_iSendBufCount == m_iSendBufSize)
    return false;

  // Within a 32 bits dword, bytes are ordered in reverse
  // Example: 0xFEDCBA98_76543210 is stored(and sent) in bytes as "0x76 0x54 0
x32 0x10 0xFE 0xDC 0xBA 0x98"
  const uint8_t iPos = (3 - (m_iSendBufCount % 4));

  const uint8_t iBufPos = ((m_iSendBufCount / 4) * 4) + iPos;
  m_pSendBuf[iBufPos] = dataByte;

  m_iSendBufCount++;
  return true;
}


bool CCommandHandler::SendBufPutBlock(const uint8_t *buf, const size_t iBufSiz
e)
{
```

```
  for (size_t it = 0; it < iBufSize; it++)
  {
    if (!SendBufPutByte(buf[it]))
      return false;
  }
  return true;
}

result_code_t CCommandHandler::ParseDecimalArg(const char* args, const uint8_t
 iCommand, const int32_t iMinIntSize, const int32_t iMaxIntSize, const uint8_t
 iOutputByteSize)
{
  result_code_t result = check_arguments(args, INT32_ARG_NUM1);
  if (result.code != OK)
    return result;

  int32_t iValue;
  result = get_int32_from_string(args, &iValue, iMinIntSize, iMaxIntSize, INT3
2_ARG_NUM1);
  if (result.code == OK)
  {
    // Set command
    SendBufPutByte(iCommand);

    // Convert int to []
    const uint8_t *u8dataBytes = (uint8_t*) &iValue;

    SendBufPutBlock(u8dataBytes, iOutputByteSize);
  }

  return result;
}


result_code_t CCommandHandler::ParseHexBinArg(const char* args, const uint8_t
iCommand, const uint8_t iMaxBitSize, const uint8_t iOutputByteSize)
{
  result_code_t result = check_arguments(args, INT32_ARG_NUM1);
  if (result.code != OK)
    return result;

  uint8_t buf[BIN_HEX_MAX_BYTE_SIZE];

  result = get_bytes_from_bin_or_hex_string(args, buf, iMaxBitSize, BIN_HEX_MA
X_BYTE_SIZE, INT32_ARG_NUM1);
  if (result.code != OK)
    return result;

  // Add command byte to buffer
```

```
  SendBufPutByte(iCommand);

  // Store block
  SendBufPutBlock(buf, iOutputByteSize);

  return result;
}


result_code_t CCommandHandler::ParseIPv4(const char* args, char *IPv4)
{
  result_code_t result = check_arguments(args, INT32_ARG_NUM1);
  if (result.code != OK)
    return result;

  return get_ipv4_from_string(args, m_IPv4, INT32_ARG_NUM1);
}

result_code_t CCommandHandler::ParseMAC(const char* args, char *MAC)
{
  result_code_t result = check_arguments(args, INT32_ARG_NUM1);
  if (result.code != OK)
    return result;

  return get_MAC_from_string(args, m_MAC, INT32_ARG_NUM1);
}

result_code_t CCommandHandler::CmdPulseMode(const char *args)
{
  result_code_t result = check_arguments(args, INT32_ARG_NUM1);
  if (result.code != OK)
    return result;

  // Set command
  SendBufPutByte(SPI_CMD_SET_MODE);

  int32_t iValue;

  if (STRIEQUALS_PSTR(args, "off"))
  {
    iValue = PULSE_MODE_OFF;
  }
  else if (STRIEQUALS_PSTR(args, "sfast"))
  {
    iValue = PULSE_MODE_SFAST;
  }
  else if (STRIEQUALS_PSTR(args, "pfast"))
  {
    iValue = PULSE_MODE_PFAST;
```

```
  }
  else if (STRIEQUALS_PSTR(args, "slow"))
  {
    iValue = PULSE_MODE_SLOW;
  }
  else if (STRIEQUALS_PSTR(args, "ext"))
  {
    iValue = PULSE_MODE_EXT;
  }
  else
  {
    return { ARG_VAL_ERR, INT32_ARG_NUM1, DBL_ARG_NONE, DBL_FMT_NONE, nullptr,
 nullptr };
  }

  // Convert int to []
  const uint8_t *u8dataBytes = (uint8_t*) &iValue;
  SendBufPutBlock(u8dataBytes, MODE_BYTE_SIZE);

  return result;
}


result_code_t CCommandHandler::CmdFastSinglePulsePeriod(const char *args)
{
  return ParseDecimalArg(args, SPI_CMD_SET_FAST_PERIOD, 2, FAST_PULSE_PERIOD_M
AX_INT_SIZE, FAST_PULSE_PERIOD_BYTE_SIZE);
}


result_code_t CCommandHandler::CmdFastProgrammedPulseData(const char *args)
{
  return ParseHexBinArg(args, SPI_CMD_SET_FAST_DATA, FAST_PULSE_DATA_MAX_BIT_S
IZE, FAST_PULSE_DATA_BYTE_SIZE);
}


result_code_t CCommandHandler::CmdSlowPulseLowTime(const char *args)
{
  return ParseDecimalArg(args, SPI_CMD_SET_SLOW_LOW_TIME, 1, SLOW_PULSE_LOW_TI
ME_MAX_INT_SIZE, SLOW_PULSE_LOW_TIME_BYTE_SIZE);
}


result_code_t CCommandHandler::CmdSlowPulseHighTime(const char *args)
{
  return ParseDecimalArg(args, SPI_CMD_SET_SLOW_HIGH_TIME, 1, SLOW_PULSE_HIGH_
TIME_MAX_INT_SIZE, SLOW_PULSE_HIGH_TIME_BYTE_SIZE);
}
```

```
// Helper function to update delayline data
void CCommandHandler::UpdateDelay(const uint16_t delay1, const uint16_t delay2
)
{
  // Add command byte to buffer
  SendBufPutByte(SPI_CMD_SET_DELAY_DATA);

  // Combine delay line data
  m_iDelayLineData = (uint32_t)delay1 | ((uint32_t)delay2 << 10);

  // Store block
  SendBufPutBlock((uint8_t*) &m_iDelayLineData, DELAYLINE_DATA_BYTE_SIZE);
}


// Helper function to update data for delay 1
void CCommandHandler::UpdateDelay1(const uint16_t delay1)
{
  const uint16_t delay2 = ((m_iDelayLineData >> 10) & 0x000003FF);
  UpdateDelay(delay1, delay2);
}


// Helper function to update data for delay 2
void CCommandHandler::UpdateDelay2(const uint16_t delay2)
{
  const uint16_t delay1 = (m_iDelayLineData & 0x000003FF);
  UpdateDelay(delay1, delay2);
}


//Helper function to format IP Address
result_code_t CCommandHandler::FormatIP(uint8_t *data, char *strResult)
{
  strResult[0] = '\0';
  for (int i = 0; i < 4; i++)
  {
     char strDec[3];
     strcat(strResult, uint32_to_decstr((uint32_t)data[i], strDec, 4));
     if (i < 3)
      STRCAT_PSTR(strResult,".");
     else
      STRCAT_PSTR(strResult,"\n\r");
  }
  return pack_result_code(OK);
}
```

```
//Helper function to format IP Address
result_code_t CCommandHandler::FormatMAC(uint8_t *data, char *strResult)
{
  strResult[0] = '\0';
  for (int i = 0; i < 6; i++)
  {
     char strDec[3];
     strcat(strResult, uint32_to_decstr((uint32_t)data[i], strDec, 6));
     if (i < 6)
      STRCAT_PSTR(strResult,":");
     else
      STRCAT_PSTR(strResult,"\n\r");
  }
  return pack_result_code(OK);
}


result_code_t GetDelayFromString(const char *str, uint16_t *iDelay, const uint
8_t iMaxBitSize, const size_t iBufSize, const uint8_t arg_num)
{
  char *end;
  const int32_t delay = strtol(str, &end, 0);

  // First check for ps
  if (STRIEQUALS_PSTR(end, "ps"))
  {
    if (delay < 0)
      return pack_result_code_arg(ARG_VAL_MIN_ERR, 0);

    if (delay > BIT_INT_SIZE(10) * 5)
      return pack_result_code_arg(ARG_VAL_MAX_ERR, BIT_INT_SIZE(10) * 5); // =
 max 5115 ps

    *iDelay = delay / 5; // Calculate from ps
  }
  else
  {
    return get_bytes_from_bin_or_hex_string(str, (uint8_t*) iDelay, iMaxBitSiz
e, iBufSize, arg_num);
  }

  return pack_result_code(OK);
}


result_code_t CCommandHandler::CmdSetDelay(char *args)
{
  result_code_t result = check_arguments(args, INT32_ARG_NUM2);
  if (result.code != OK)
```

```
    return result;

  char *strDelay1 = args; // For clarity
  char *strDelay2 = strsplit(args, " ");
  if (strDelay2 == NULL)
    return pack_result_code(ARG_MISSING_ERR);

  uint16_t delay1 = 0;
  uint16_t delay2 = 0;

  result = GetDelayFromString(strDelay1, &delay1, (DELAYLINE_DATA_MAX_BIT_SIZE
 / 2), sizeof(delay1), INT32_ARG_NUM1);
  if (result.code != OK)
    return result;

  result = GetDelayFromString(strDelay2, &delay2, (DELAYLINE_DATA_MAX_BIT_SIZE
 / 2), sizeof(delay2), INT32_ARG_NUM2);
  if (result.code != OK)
    return result;

  UpdateDelay(delay1, delay2);

  return pack_result_code(OK);
}


result_code_t CCommandHandler::CmdSetDelay1(const char *args)
{
  result_code_t result = check_arguments(args, INT32_ARG_NUM1);
  if (result.code != OK)
    return result;

  uint16_t delay = 0;

  result = GetDelayFromString(args, &delay, (DELAYLINE_DATA_MAX_BIT_SIZE / 2),
 sizeof(delay), INT32_ARG_NUM1);
  if (result.code != OK)
    return result;

  UpdateDelay1(delay);

  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdSetDelay2(const char *args)
{
  result_code_t result = check_arguments(args, INT32_ARG_NUM1);
  if (result.code != OK)
    return result;
```

```
  uint16_t delay = 0;

  result = GetDelayFromString(args, &delay, (DELAYLINE_DATA_MAX_BIT_SIZE / 2),
 sizeof(delay), INT32_ARG_NUM1);
  if (result.code != OK)
    return result;

  UpdateDelay2(delay);

  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdSetVBDAC(const char *args)
{
  return ParseHexBinArg(args, SPI_CMD_SET_VB_DAC_DATA, DAC_DATA_MAX_BIT_SIZE,
DAC_DATA_BYTE_SIZE);
}


result_code_t CCommandHandler::CmdSetVRDAC(const char *args)
{
  return ParseHexBinArg(args, SPI_CMD_SET_VR_DAC_DATA, DAC_DATA_MAX_BIT_SIZE,
DAC_DATA_BYTE_SIZE);
}


result_code_t CCommandHandler::CmdFPGAReset(const char *args)
{
  result_code_t result = check_arguments(args, INT32_ARG_NONE);
  if (result.code != OK)
    return result;

  // Fill complete SPI buffer with reset command
  for (uint8_t i = 0; i < m_iSendBufSize; i++)
  {
    // Set command
    SendBufPutByte(SPI_CMD_FPGA_RESET);
  }

  return pack_result_code(OK);
}


// Show copyright + firmware version
void CCommandHandler::GetVersion(char *strResult)
{
  strcpy_P(strResult, VER_STR_P);
  STRCAT_PSTR(strResult, "\n\r");
```

```
}


// Show copyright + firmware version
result_code_t CCommandHandler::CmdShowVersion(const char *args, char *strResul
t)
{
  result_code_t result = check_arguments(args, INT32_ARG_NONE);
  if (result.code != OK)
    return result;

  GetVersion(strResult);

  return pack_result_code(OK_NULL);
}


result_code_t CCommandHandler::CmdShowFWVersion(const char *args, char *strRes
ult)
{
  result_code_t result = check_arguments(args, INT32_ARG_NONE);
  if (result.code != OK)
    return result;

  strcpy_P(strResult, VERSION);
  STRCAT_PSTR(strResult, "\n\r");

  return pack_result_code(OK_NULL);
}


// Show help screen
result_code_t CCommandHandler::CmdShowHelp(const char *args, char *strResult)
{
  if (args != NULL && *args)
    return pack_result_code_arg(TOO_MANY_ARGS, INT32_ARG_NONE);

  strcpy_P(strResult, HELP_STR_P);
  STRCAT_PSTR(strResult, "\n\r");

  return pack_result_code(OK);
}


// Show device IDN
result_code_t CCommandHandler::CmdShowIDN(const char *args, char *strResult)
{
  bool show_fwv = false;
  if (args != NULL && *args)
```

```
  {
    if (STRIEQUALS_PSTR(args, "ext") || STRIEQUALS_PSTR(args, "ex"))
    {
      show_fwv = true;
    }
    else
    {
      return pack_result_code_arg(ARG_VAL_ERR, INT32_ARG_NUM1);
    }
  }

  strcpy_P(strResult, DEVICE_IDN_P);

  if (show_fwv)
  {
    // Suffix with F/W version
//    strcat_pstr(strIDN, "_FW:v");
    strcat_P(strResult, VERSION);
  }

  STRCAT_PSTR(strResult, "\n\r");

  return pack_result_code(OK_NULL); // OK but don't echo OK to user
}


// Handle local echo on/off
result_code_t CCommandHandler::CmdEchoOnOff(const char *args)
{
  if (args == NULL || !*args)
    return pack_result_code(ARG_MISSING_ERR);

  if (STRIEQUALS_PSTR(args, "on"))
    m_bLocalEcho = true;
  else if (STRIEQUALS_PSTR(args, "off"))
    m_bLocalEcho = false;
  else
    return pack_result_code(CMD_SYNTAX_ERR);

  return pack_result_code(OK);
}


// Handle blink on/off
result_code_t CCommandHandler::CmdBlinkOnOff(const char *args)
{
  if (args == NULL || !*args)
    return pack_result_code(ARG_MISSING_ERR);
```

```
  if (STRIEQUALS_PSTR(args, "on"))
    m_bBlink = true;
  else if (STRIEQUALS_PSTR(args, "off"))
    m_bBlink = false;
  else
    return pack_result_code(CMD_SYNTAX_ERR);

  m_iCommand = SETBLINK;
  return pack_result_code(OK);
}




////////////////////////////////////////////////////////////////////////////
//////////////////////////Cmd zuiverheidsmeter
result_code_t CCommandHandler::CmdSetIP(const char *args)
{
  if (args == NULL || !*args)
    return pack_result_code(ARG_MISSING_ERR);

  result_code_t result = ParseIPv4(args, m_IPv4);
  if (result.code != OK)
    return pack_result_code(INVALID_IPV4);

  m_iCommand = SETIP;
  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdGetIP(const char *args)
{
  if (args != NULL && *args)
    return pack_result_code_arg(TOO_MANY_ARGS, INT32_ARG_NONE);
  m_iCommand = ShowIP;
  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdSetMAC(const char *args)
{
  if (args == NULL || !*args)
    return pack_result_code(ARG_MISSING_ERR);

  result_code_t result = ParseMAC(args, m_IPv4);
  if (result.code != OK)
    return pack_result_code(INVALID_MAC);

  m_iCommand = SetMacaddress;
  return pack_result_code(OK);
```

```
}

result_code_t CCommandHandler::CmdSetNetmask(const char *args)
{

}

result_code_t CCommandHandler::CmdSetdNS(const char *args)
{

}

result_code_t CCommandHandler::CmdGetdNS(const char *args)
{

}

result_code_t CCommandHandler::CmdSetDHCP(const char *args)
{

}

result_code_t CCommandHandler::CmdSetPortServer(const char *args)
{
  if (args == NULL || !*args)
    return pack_result_code(ARG_MISSING_ERR);
   result_code_t result = get_int32_from_string(args, &m_iPoort, 0, 65535, INT
32_ARG_NUM1);
  if (result.code != OK)
    return pack_result_code(INVALID_IPV4); //need to change!

  m_iCommand = SetPoort;
  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdGetportServer(const char *args)
{
  if (args != NULL && *args)
    return pack_result_code_arg(TOO_MANY_ARGS, INT32_ARG_NONE);

  m_iCommand = ShowPoort;
  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdGetIPserver(const char *args)
{
  if (args != NULL && *args)
    return pack_result_code_arg(TOO_MANY_ARGS, INT32_ARG_NONE);
```

```
   m_iCommand = ShowIPserver;
  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdSetIPserver(const char *args)
{
  if (args == NULL || !*args)
    return pack_result_code(ARG_MISSING_ERR);

  result_code_t result = ParseIPv4(args, m_IPv4);
  if (result.code != OK)
    return pack_result_code(INVALID_IPV4);

  m_iCommand = SetIPserver;
  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdGetMAC(const char *args)
{
  if (args != NULL && *args)
    return pack_result_code_arg(TOO_MANY_ARGS, INT32_ARG_NONE);

 m_iCommand = GetMACaddress;
 return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdReboot(const char *args)
{
  if (args != NULL && *args)
    return pack_result_code_arg(TOO_MANY_ARGS, INT32_ARG_NONE);

  m_iCommand = Reboot;
  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdSetAlarm(const char *args)
{
  if (args == NULL || !*args)
    return pack_result_code(ARG_MISSING_ERR);

  if (STRIEQUALS_PSTR(args, "0"))
    m_iAlarm = 0;
   // m_iCommand = AlarmOff;
  else if (STRIEQUALS_PSTR(args, "1"))
    m_iAlarm = 1;
  //  m_iCommand = Alarm_3_beep;
  else if (STRIEQUALS_PSTR(args, "2"))
```

```
    m_iAlarm = 2;
  //  m_iCommand = Alarm_10Sec_2min;
  else if (STRIEQUALS_PSTR(args, "3"))
    m_iAlarm = 3;
  // m_iCommand = Alarm_20beep_10sec;
  else
    return pack_result_code(CMD_SYNTAX_ERR);
    m_iCommand = ALARM;
  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdGetNetmask(const char *args)
{

}

result_code_t CCommandHandler::CmdSetgateway(const char *args)
{

}

result_code_t CCommandHandler::CmdGetgateway(const char *args)
{

}

result_code_t CCommandHandler::CmdReconnect(const char *args)
{
  if (args != NULL && *args)
    return pack_result_code_arg(TOO_MANY_ARGS, INT32_ARG_NONE);

  m_iCommand = Reconnect;
  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdGetResponse(const char *args)
{
  if (args != NULL && *args)
    return pack_result_code_arg(TOO_MANY_ARGS, INT32_ARG_NONE);

  m_iCommand = ResponseServer;
  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdCalibrateHelium(const char *args)
{
    if (args != NULL && *args)
    return pack_result_code_arg(TOO_MANY_ARGS, INT32_ARG_NONE);
```

```
  m_iCommand = CalibrateHelium;
  return pack_result_code(OK);
}

result_code_t CCommandHandler::CmdCalibrateAir(const char *args)
{
    if (args != NULL && *args)
    return pack_result_code_arg(TOO_MANY_ARGS, INT32_ARG_NONE);

  m_iCommand = CalibrateAir;
  return pack_result_code(OK);
}


result_code_t CCommandHandler::Execute(char *strCommand, char *strResult)
{
  // Reset send buffer
  SendBufReset();

  // Reset result string
  strResult[0] = 0x00;

  // Trim strCommand to remove leading/trailing space \n \r
  strtrim(strCommand, " \n\r");

  // Preinit return value
  result_code_t result = { CMD_UNKNOWN_ERR, INT32_ARG_NONE, DBL_ARG_NONE, DBL_
FMT_NONE, nullptr, nullptr };

  // Split into strCommand & arguments
  char *args = strsplit(strCommand, " "); // args may be NULL, but is handled
by the functions called below

  // Our strCommand interpreter:
  if (STRIEQUALS_PSTR(strCommand, "*IDN?") || STRIEQUALS_PSTR(strCommand, "IDN
") || STRIEQUALS_PSTR(strCommand, "IDN?"))
  {
    result = CmdShowIDN(args, strResult);
  }
  else if (STRIEQUALS_PSTR(strCommand, "ver"))
  {
    result = CmdShowVersion(args, strResult);
  }
  else if (STRIEQUALS_PSTR(strCommand, "fw") || STRIEQUALS_PSTR(strCommand, "f
wv"))
  {
    result = CmdShowFWVersion(args, strResult);
  }
```

```
  else if (STRIEQUALS_PSTR(strCommand, "help") || STRIEQUALS_PSTR(strCommand,
"?"))
  {
    result = CmdShowHelp(args, strResult);
  }
  else if (STRIEQUALS_PSTR(strCommand, "echo"))
  {
    result = CmdEchoOnOff(args);
  }
  else if (STRIEQUALS_PSTR(strCommand, "setblink"))
  {
    result = CmdBlinkOnOff(args);
  }
  else if (STRIEQUALS_PSTR(strCommand, "setip"))
  {
    result = CmdSetIP(args);
  }
   else if (STRIEQUALS_PSTR(strCommand, "setmac"))
  {
    result = CmdSetMAC(args);
  }
   else if (STRIEQUALS_PSTR(strCommand, "setnetmask"))
  {
    result = CmdSetNetmask(args);
  }
   else if (STRIEQUALS_PSTR(strCommand, "setdns"))
  {
    result = CmdSetdNS(args);
  }
   else if (STRIEQUALS_PSTR(strCommand, "setdhcp"))
  {
    result = CmdSetDHCP(args);
  }
  else if (STRIEQUALS_PSTR(strCommand, "setport"))
  {
    result = CmdSetPortServer(args);
  }
  else if (STRIEQUALS_PSTR(strCommand, "showport"))
  {
    result = CmdGetportServer(args);
  }
  else if (STRIEQUALS_PSTR(strCommand, "showip"))
  {
    result = CmdGetIP(args);
  }
  else if (STRIEQUALS_PSTR(strCommand, "showipserver"))
  {
    result = CmdGetIPserver(args);
```

```
  }
  else if (STRIEQUALS_PSTR(strCommand, "setipserver"))
  {
    result = CmdSetIPserver(args);
  }
  else if (STRIEQUALS_PSTR(strCommand, "showmac"))
  {
    result = CmdGetMAC(args);
  }
  else if (STRIEQUALS_PSTR(strCommand, "reboot"))
  {
    result = CmdReboot(args);
  }
else if (STRIEQUALS_PSTR(strCommand, "shownetmask"))
  {
    result =  CmdGetNetmask(args);
  }
else if (STRIEQUALS_PSTR(strCommand, "setgateway"))
  {
    result =  CmdSetgateway(args);
  }
else if (STRIEQUALS_PSTR(strCommand, "showgateway"))
  {
    result =  CmdGetgateway(args);
  }
else if (STRIEQUALS_PSTR(strCommand, "showDNS"))
  {
    result =  CmdGetdNS(args);
  }
else if (STRIEQUALS_PSTR(strCommand, "Reconnect"))
  {
    result =  CmdReconnect(args);
  }
  else if (STRIEQUALS_PSTR(strCommand, "calibrateHelium"))
  {
    result =  CmdCalibrateHelium(args);
  }

  else if (STRIEQUALS_PSTR(strCommand, "calibrateAir"))
  {
    result =  CmdCalibrateAir(args);
  }

  if (result.code != OK_NULL)
    get_error_string(result, strResult, true); // True means append

  return result;
}
```

```
result_code_t CCommandHandler::Execute_PollEthernet(char *strCommand, char *st
rResult)
{
  // Reset send buffer
  SendBufReset();

  // Reset result string
  strResult[0] = 0x00;

  // Trim strCommand to remove leading/trailing space \n \r
  strtrim(strCommand, " \n\r");

  // Preinit return value
  result_code_t result = { CMD_UNKNOWN_ERR, INT32_ARG_NONE, DBL_ARG_NONE, DBL_
FMT_NONE, nullptr, nullptr };

  // Split into strCommand & arguments
  char *args = strsplit(strCommand, " "); // args may be NULL, but is handled
by the functions called below

  // Our strCommand interpreter:
  if (STRIEQUALS_PSTR(strCommand, "OK"))
  {
    result =  CmdGetResponse(args);
  }
  else if (STRIEQUALS_PSTR(strCommand, "alarm"))
  {
     result = CmdSetAlarm(args);
  }

  if (result.code != OK_NULL)
  get_error_string(result, strResult, true); // True means append

  return result;
}
```

CommandHandler.H

```
#ifndef COMMAND_HANDLER_H
#define COMMAND_HANDLER_H

#include "CommandParser.h"

#include <stdlib.h>

enum uint8_command
{
  NO_COMMAND = 0,
  SETBLINK,
  SETIP,
  ALARM,
  ShowIP,
  SetPoort,
  ShowPoort,
  SetIPserver,
  ShowIPserver,
  Reboot,
  GetMACaddress,
  SetMacaddress,
  ResponseServer,
  ResponseServerAlarm,
  Reconnect,
  ShowMacaddress,
  CalibrateHelium,
  CalibrateAir
};



class CCommandHandler
{
  enum uint8_pulse_mode
  {
    PULSE_MODE_OFF = 0,
    PULSE_MODE_SFAST,
    PULSE_MODE_PFAST,
    PULSE_MODE_SLOW,
    PULSE_MODE_EXT
  };


  public:
    CCommandHandler(uint8_t *pSendBuf, const size_t iSendBufSize);
    ~CCommandHandler(void) {}; // Empty dtor
```

```
    result_code_t Execute(char *strCommand, char *strResult);
    result_code_t Execute_PollEthernet(char *strCommand, char *strResult);
    size_t GetBufCount(void) const { return m_iSendBufCount; };
    bool GetLocalEchoEnabled(void) const { return m_bLocalEcho; };
    uint8_t GetCommand(void) const { return m_iCommand; };
    bool GetLedArg(void) const { return m_bBlink; };
    int GetAlarm(void) const {return m_iAlarm;}
    char* GetIPv4() { return m_IPv4; };
    char* GetMAC() {return m_MAC;}
    int32_t GetPoort(){return m_iPoort; };
    static void GetVersion(char *strResult);
    void SetCommandAck(void) {m_iCommand = NO_COMMAND; };
    result_code_t FormatIP(uint8_t *data, char *strResult);
    result_code_t FormatMAC(uint8_t *data, char *strResult);

  private:
    void SendBufReset(void);
    bool SendBufPutByte(const uint8_t dataByte);
    bool SendBufPutBlock(const uint8_t *buf, const size_t iBufSize);

    result_code_t ParseDecimalArg(const char* args, const uint8_t iCommand, co
nst int32_t iMinIntSize, const int32_t iMaxIntSize, const uint8_t iOutputBitSi
ze);
    result_code_t ParseHexBinArg(const char* args, const uint8_t iCommand, con
st uint8_t iMaxBitSize, const uint8_t iOutputBitSize);
    result_code_t ParseIPv4(const char* args, char* IPv4);
    result_code_t ParseMAC(const char* args, char* MAC);

    result_code_t CmdPulseMode(const char *args);
    result_code_t CmdFastSinglePulsePeriod(const char *args);
    result_code_t CmdFastProgrammedPulseData(const char *args);
    result_code_t CmdSlowPulseLowTime(const char *args);
    result_code_t CmdSlowPulseHighTime(const char *args);
    void UpdateDelay(const uint16_t delay1, const uint16_t delay2);
    void UpdateDelay1(const uint16_t delay);
    void UpdateDelay2(const uint16_t delay);
    result_code_t CmdSetDelay(char *args);
    result_code_t CmdSetDelay1(const char *args);
    result_code_t CmdSetDelay2(const char *args);
    result_code_t CmdSetVBDAC(const char *args);
    result_code_t CmdSetVRDAC(const char *args);
    result_code_t CmdFPGAReset(const char *args);
    result_code_t CmdShowVersion(const char *args, char *strResult);
    result_code_t CmdShowFWVersion(const char *args, char *strResult);
    result_code_t CmdShowHelp(const char *args, char *strResult);
    result_code_t CmdShowIDN(const char *args, char *strResult);
    result_code_t CmdEchoOnOff(const char *args);
    result_code_t CmdBlinkOnOff(const char *args);
```

```
    ////////////////////////////////zuiverheidsmeter
    result_code_t CmdSetIP(const char *args);
    result_code_t CmdGetIP(const char *args);
    result_code_t CmdSetAlarm(const char *args);
    result_code_t CmdSetMAC(const char *args);
    result_code_t CmdGetMAC(const char *args);
    result_code_t CmdSetNetmask(const char *args);
    result_code_t CmdGetNetmask(const char *args);
    result_code_t CmdSetdNS(const char *args);
    result_code_t CmdGetdNS(const char *args);
    result_code_t CmdSetDHCP(const char *args);
    result_code_t CmdSetPortServer(const char *args);
    result_code_t CmdGetportServer(const char *args);
    result_code_t CmdGetIPserver(const char *args);
    result_code_t CmdSetIPserver(const char *args);
    result_code_t CmdReboot(const char *args);
    result_code_t CmdSetgateway(const char *args);
    result_code_t CmdGetgateway(const char *args);
    result_code_t CmdGetResponse(const char *args);
    result_code_t CmdReconnect(const char *args);
    result_code_t CmdCalibrateHelium(const char *args);
    result_code_t CmdCalibrateAir(const char *args);

    size_t m_iSendBufSize;
    uint8_t *m_pSendBuf;
    size_t m_iSendBufCount;
    bool m_bLocalEcho;
    bool m_bBlink;
    bool m_bOK;
    int m_iAlarm ;
    uint8_t m_iCommand = NO_COMMAND;
    char m_IPv4[4];
    char m_MAC[6];
    int32_t m_iPoort;
    uint32_t m_iDelayLineData;
};

#endif // COMMAND_HANDLER_H
```

19 Bewijs van competenties

Tijdens het afstuderen heb ik de volgende competenties kunnen uitvoeren:

- Analyseren;
- Ontwerpen;
- Realiseren;
- Managen;
- Onderzoeken.

Analyseren, ontwerpen en realiseren behoren eindniveau 3 te zijn. Managen en onderzoeken op niveau 2.

19.1 Analyseren

Het project Heliumzuiverheidsmeter bestond uit het ontwerpen en realiseren van de hardware, pcb, software en een behuizing.

Voordat ik aan het project begon, ben ik gestart met het maken van een Plan van Aanpak (PvA). Doordat het project uit meerdere delen bestond, heb ik veel aandacht aan de planning besteed. Als eerste maakte ik een globale planning door het schatten van de hoeveelheid tijd die ik voor een taak nodig zou hebben. Zo heb ik bijvoorbeeld nog nooit met “Altium Designer” gewerkt en heb ik hier extra tijd voor ingepland.

Vervolgens heb ik, om een duidelijk beeld te krijgen van hoe de Heliumzuiverheidsmeter eruit moest komen te zien, als eerste de plek waar deze moest komen te staan bekeken. Dit was onder andere belangrijk om de grote van de behuizing te kunnen bepalen. Omdat de plek waar de Heliumzuiverheidsmeter zou komen te hangen zich in een grote ruimte bevond, heb ik ervoor gekozen gebruik te maken van Ethernet in plaats van Wifi om de data naar de server te sturen.

Daarna heb ik de tijd genomen om te onderzoeken welke programma’s ik het beste kon gebruiken voor het ontwerpen van de Heliumzuiverheidsmeter. Uiteindelijk heb ik gekozen voor het gebruiken van de volgende programma’s:

- Altium Designer (ontwerpen van de PCB);
- Fusion 360 (ontwerpen van de behuizing);
- Platform IO (programmeren van de microcontroller);
- Lucid App (tekenen van blokschema’s);
- Tina TI (simuleren van de hardware);
- GIT (managen van software);
- Seafile (managen van documentatie).

Als laatste heb ik uitgezocht welke componenten besteld moesten worden voor het realiseren van het project. Door Covid-19 was de verwachting dat sommige componenten niet leverbaar waren of er lange levertijden zouden zijn. Ik heb hierop geanticipeerd door tijdig de componenten te bestellen. Ook heb ik van elk onderdeel meerdere bestelt zodat wanneer een component defect raakte, ik deze op voorraad had en niet hoefde te wachten op nieuwe onderdelen.

Door het opstellen van het PvA, het onderzoeken van de locatie waar het project komt te hangen, het bepalen van de juiste programma’s om mee te werken en het anticiperen op levertijden van componenten heb ik aan de competentie ‘Analyseren’ voldaan.

19.2 Ontwerpen

Onderstaande onderdelen zijn ontworpen om het project tot stand te brengen:

- Hardware;
- PCB;
- Behuizing
- Software;

19.2.1 Hardware

Voordat ik begon aan het ontwerpen van de hardware, heb ik een blokschema opgesteld. Het blokschema geeft aan welke functies de hardware moet kunnen uitvoeren. Het blokschema bevat de onderstaande hardware functie:

- Het aansturen van de in- en uitlaat kleppen waardoor de Helium de opstelling in en uit kan stromen.
- Het uitsturen van een geluidsgolf en het ontvangen van een geluidsgolf.

Met deze gegevens ben ik begonnen aan het ontwerpen van de hardware. Nadat de hardware ontworpen was, heb ik met een simulatieprogramma (Tina-TI) de ontwerpen gesimuleerd.

19.2.2 PCB

Na het ontwerpen van de elektronica, heb ik het elektrisch schema in het programma “Altium Designer” gezet. Ook heb ik weer een blokschema gemaakt. In het blokschema zijn de volgende ontwerpkeuzes gemaakt om het analoge en digitale gedeelte zoveel mogelijk te scheiden:

- De voeding is aan de buitenkant van de PCB geplaatst zodat er makkelijk een koellichaam gemonteerd kan worden.
- De aansluitingen van de USB en Ethernet zijn aan dezelfde kant geplaatst.
- Er is gebruik gemaakt van inplugbare eindstukken.

19.2.3 Behuizing

Voordat de behuizing ontworpen werd, ben ik eerst in overleg gegaan met de opdrachtgever over de vraag of er nog specifieke wensen waren voor het ontwerp. Daar kwam uit dat de invoer van het Helium zich aan de onderkant en de uitvoer aan de bovenkant van de behuizing moest bevinden. Aan de hand van dit overleg heb ik een 3D ontwerp gemaakt in “Fusion 360”.

19.2.4 Software

Bij het ontwerpen van de software ben ik begonnen met het maken van een blokschema van de *statemachine*. In de *statemachine* staat beschreven welke stappen worden doorlopen voor het maken van een meting.

Door het maken van blokschema’s, elektrische schema’s en 3D- ontwerpen heb ik voldaan aan de competentie ‘Ontwerpen’.

19.3 Realiseren

Voor het project zijn de volgende onderdelen gerealiseerd: PCB, behuizing, software en scriptie.

PCB

Omdat de PCB over zowel SMD-componenten als "*through hole*" componenten beschikt, zijn als eerst de SMD-componenten geplaatst. Hiervoor is gebruik gemaakt van een *stencil* (meegeleverd door de fabrikant van de PCB) en soldeerpasta. Na het aanbrengen van de soldeerpasta heb ik de componenten met een "*pick and place*" machine geplaatst. Na het plaatsen van de SMD-componenten heb ik de "*through hole*" componenten geplaatst. Ik heb de lage componenten (onder andere de weerstanden) eerst geplaatst. Daarna heb ik de hogere componenten geplaatst (onder andere de spanningsregelaar).

Met de ontwerptekeningen van de behuizing ben ik naar de FMD (Fijn Mechanische Dienst) gegaan, waar de Heliummeter is gecreëerd. Tijdens het creëren van de Heliummeter heb ik nauw contact gehouden met de afdeling en toezicht gehouden op de voortgang. Nadat de meter was gerealiseerd door de FMD, heb ik zelf de behuizing gemaakt met daarin de PCB, voeding en drukknop. Ook heb ik de uitsparingen in de behuizing gemaakt voor het 7-segments display.

Nadat de behuizing af was, heb ik met de oscilloscoop metingen gedaan voor het instellen van de versterkingsfactor van de microfoon.

Voor het creëren van de software heb ik afzonderlijke softwarecomponenten gemaakt (een voorbeeld hiervan is de aansturing van het 7-segments display). Vervolgens heb ik de losse softwarecomponenten in elkaar geïntegreerd.

Door het solderen van de PCB, het testen van de PCB, het maken van de behuizing en het schrijven van de software heb ik voldaan aan de competentie 'Realiseren'.

19.4 Managen

Omdat dit een groot project is, was een belangrijk onderdeel een goede managing van het project. De planning speelde hierin een grote rol in. Aan het begin van het project ben ik begonnen met het maken van een planning. In deze planning heb ik een schatting gemaakt van welke taken meer of minder tijd in beslag gingen nemen. Ik had bijvoorbeeld nog nooit met Altium Designer, Platform IO en de Teensy(microcontroller) gewerkt. Ik had voor deze onderdelen dan ook extra tijd ingepland.

Een groot onderdeel van het project was de scriptie. Tijdens het hele project heb ik gewerkt aan de scriptie. Als ik bijvoorbeeld de hardware ontworpen en gesimuleerd had, documenteerde ik dit gelijk.

Voor het managen van de software is gebruik gemaakt van GIT. Wanneer ik aan de software werkte, deed ik aan de eind van de dag een "push" naar GIT. In GIT worden alle veranderingen van de code bijgehouden. Wanneer ik een verandering in de software maakte, kon ik altijd terug naar een oudere versie.

Voor het managen van het realiseren van de behuizing heb ik nauw contact gehouden met de FMD. Ik heb gevraagd om urgentie, waardoor de meter drie weken eerder klaar was.

De project documentatie is opgeslagen in "Seafile". De bestanden zijn op deze manier goed geback-upt op de lokale server van de ELD.

Tijdens mijn project heb ik veel contact gehad met mijn stagebegeleider. Omdat mijn stagebegeleider veel thuis werkte, stuurde ik dagelijks een bericht met daarin de voortgang van het project. Iedere twee weken hield ik contact met mijn 1e coach Dhr. E. Korkmaz.

Door het bijhouden van de planning, contact houden met mijn stagebegeleider en coach, het gebruik maken van GIT en het bijhouden van documentatie heb ik voldaan aan de competentie ‘Managen’.

19.5 Onderzoeken

Voorafgaand aan het project ben ik een vooronderzoek gestart.

Allereerst heb ik de theoretische kaders omtrent het meten in Helium onderzocht. Ik heb tijdens dit onderzoek geleerd:

- Wat de kenmerken van Helium zijn;
- De verschillende manieren om Helium te meten;
- Wat de geluidssnelheid in Helium is afhankelijk van de afstand;
- Wat de geluidssnelheid in Helium is afhankelijk van de temperatuur;
- Hoe je een geluidsgolf maakt en ontvangt

Nadat het vooronderzoek afgerond was, heb ik een hypothese opgesteld: “Een buis met een grote diameter geeft veel reflecties maar er komt veel signaal aan bij de ontvanger. Het gebruikt van een dunne buis geeft minder reflecties maar zorgt ook voor een verzwakking bij de ontvanger”. Deze hypothese heb ik getest door gebruik te maken van verschillende behuizingen. Voor het meten en verifiëren van de hypothese heb ik gebruik gemaakt van een oscilloscoop. De uiteindelijke data bevestigde mijn hypothese helaas niet. De dikte van de buis had geen invloed op de reflecties.

Door het opstellen van een hypothese, testen van de hypothese en vervolgens een conclusie te trekken heb ik voldaan aan de competentie ‘Onderzoeken’.