Tartu Ülikool

Loodus- ja täppisteaduste valdkond

Tehnoloogiainstituut

Taavi Sõerd

Mehitamata õhusõiduki autopiloodi õhukiirusandurite trükkplaatide testseade

Bakalaureusetöö (12 EAP)
Arvutitehnika eriala

Juhendaja:

BSc Artur Abels

Tartu 2025

Resümee/Abstract

Mehitamata õhusõiduki autopiloodi õhukiirusandurite trükkplaatide testseade

Sensorite töökindluse tõstmiseks ning võimalike tootmisvigade varajaseks avastamiseks on oluline nende põhjalik testimine. Käesoleva töö eesmärk on välja töötada testseade, mis võimaldab korraga testida kuni 16 mehitamata õhusõiduki autopiloodi õhukiiruse sensori trükkplaati. Selleks projekteeritakse spetsiaalne trükkplaat, mille külge ühendatakse sensorid ning mis kogub ja salvestab vajalikud mõõteandmed. Loodav testseade aitab tuvastada võimalikke defekte juba tootmise varajases etapis, aidates seeläbi parandada trükkplaatide kvaliteeti ja üldist töökindlust.

CERCS: T170 Elektroonika; T125 Automatiseerimine, robootika, control engineering; [1])

Märksõnad: trükkplaat, mehitamata õhusõiduk, autopiloot, testimine

Test device for unmanned aerial vehicle autopilot sensor circuit boards

To improve the reliability of the sensors and enable early detection of potential manufacturing defects, thorough testing is essential. The objective of this work is to develop a test device that allows simultaneous testing of up to 16 unmanned aerial vehicle autopilot airspeed sensor circuit boards. For this purpose a specialized circuit board will be designed to connect the sensors, collect and store the necessary measurement data. The resulting test device will help identify possible defects at an early stage of production, thereby contributing to improved circuit board quality and overall reliability.

CERCS: T170 Electronics; T125 Automation, robotics, control engineering [1])

Keywords: Printed circuit board, unmanned aerial vehicle, autopilot, testing

Sisukord

Joonised

Lühendid

UAV *(Unmanned Aerial Vehicle)* - mehitamata õhusõiduk
IAS *(Indicated Air Speed)* - indikeeritud õhukiirus
GPS *(Global positioning system)* - ülemaailmne asukoha määramise süsteem
CAN *(Controller Area Network)* - kontrollervõrk
SPI *(Serial Peripheral Interface)* - sünkroonne järjestiksuhtluse liides
USB *(Universal Serial Bus)* - Universaalne jadasiin
COM *(Communication)* - suhtlus
SCL *(Serial clock)* - sünkroonne taktsignaal
MOSI *(Master Output Slave Input)* - Ülem välja, alluv sisse
MISO *(Master Input Slave Output)* - Ülem sisse, alluv välja
CS *(Chip select)* - Kiibi valik
SS *(Slave select)* - Alluva valik
CPOL *(Clock Polarity)* - Takti polaarsus
CPHA *(Clock Phase)* - Takti faas
HTTP *(Hypertext Transfer Protocol)* - Hüperteksti edastusprotokoll
API *(Application Programming Interface)* - Rakendusliides
JSON *(JavaScript Object Notation)* - JavaScripti objektide notatsioon
XML *(Extensible Markup Language)* - Laiendatav märgistuskeel
ESD *(Electrostatic Discharge)* - Elektrostaatiline laengulahendus
LDO *(Low Dropout Regulator)* - Lineaarne pingeregulaator
SSID *(Service Set Identifier)* - Teenusekogumi identifikaator / võrgu nimi

1 Sissejuhatus

UAV (mehitamata õhusõiduk, *unmanned aerial vehicle*) on tehnoloogia, mida rakendatakse tänapäeval mitmetes erinevates valdkondades, näiteks päästeoperatsioonides, täppispõllumajanduses, militaarvaldkondades. Autopiloot on süsteem, mis võimaldab UAV-l sõita ilma pideva piloodi juhtimiseta, korrigeerides kurssi, kiirust, kõrgust ja muud sellist automaatselt vastavalt seadistusele. Otsuste langetamiseks on autopiloodil vaja palju erinevat infot UAV hetkeoleku kohta. Selleks on paigaldatud erinevaid sensoreid, mille peale autopiloot toetub välise info jaoks. Seega on väga oluline, et need sensorid oleksid võimalikult usaldusväärsed ja seda ka rasketes tingimustes, näiteks eriti niiskes kliimas ja kõrgetel temperatuuridel.

Üheks põhiliseks infoks, mida autopiloot kasutab on IAS (indikeeritud õhukiirus, *Indicated Air Speed*), mille see saab õhukiiruse sensorilt. See info aitab autopiloodil näiteks täpsemini sooritada manöövreid ja tuvastada ka UAV asukohta.

Käesolev töö keskendub UAV autopiloodi õhukiiruse sensori trükkplaadi testseadme loomisele, mis võimaldab testida kuni 16 sensorit korraga, et vähendada märgatavalt testimiseks vajavat ressurssi ning samas aidata kaasa ka üldisele UAV usaldatavusele.

1.1 Probleemi tutvustus

Õhukiiruse sensori test on umbes kolme tunnine protsess, mille käigus hoitakse testitavat sensorit kontrollitud tingimustes temperatuuridel 30 °C, 60 °C, 90 °C ning 90 °C koos 96% õhuniiskusega. Sellise testiga saab kontrollida sensori töökorda ja usaldusväärsust ka pikemal ajal ekstreemsematel tingimustel olles.

Hetkel kasutatakse õhukiiruse sensori tesimiseks autopiloodi tarkvara. See tarkvara, aga ei ole mõeldud testimiseks, vaid päris lennuks, seega on sellisel seadistusel puuduseid. Üheks suureks puuduseks on võimekus ühes arvutis jooksutada vaid ühte autopiloodi isendit. Lisaks sellele suudab see võtta vastu andmeid vaid ühelt sensorilt. Seega läheb mitme õhukiiruse sensori testimiseks vaja mitut arvutit, mis teeb suurte koguste sensorite testimise mitteskaleeritavaks.

Käesoleva töö eesmärk on disainida ja realiseerida trükkplaat ning tarkvara õhukiiruse anduri efektiivsemaks testimiseks. Lahenduse põhieesmärkideks on:

- võimalus samaaegselt ühendada ja testida kuni 16 õhukiiruse sensorit

- salvestada mõõtetulemused hiljem lugemiseks ja analüüsimiseks
- testi vältel kuvama mõõtetulemusi reaalajas

1.2 Kavandatav töö

Kavandatav seade kasutab mikrokontrollerit, mis loeb õhukiirussensorilt saadetuid andmeid, mida sensor saadab läbi CAN (kontrollervõrk, *Controller Area Network*) protokolli. Kuna kõik testitavad õhukiiruse andurid kasutavad sama CAN ID-d, on vaja leida moodus kuidas nendega eraldi suhelda.

Mõõtetulemused salvestatakse SD-kaardile JSON (JavaScripti objektide notatsioon, *JavaScript Object Notation*) formaadis. See on laialt levinud formaat, mida toetavad paljud süsteemid ja seega on hilisem lugemine ja analüüsimine mugav.

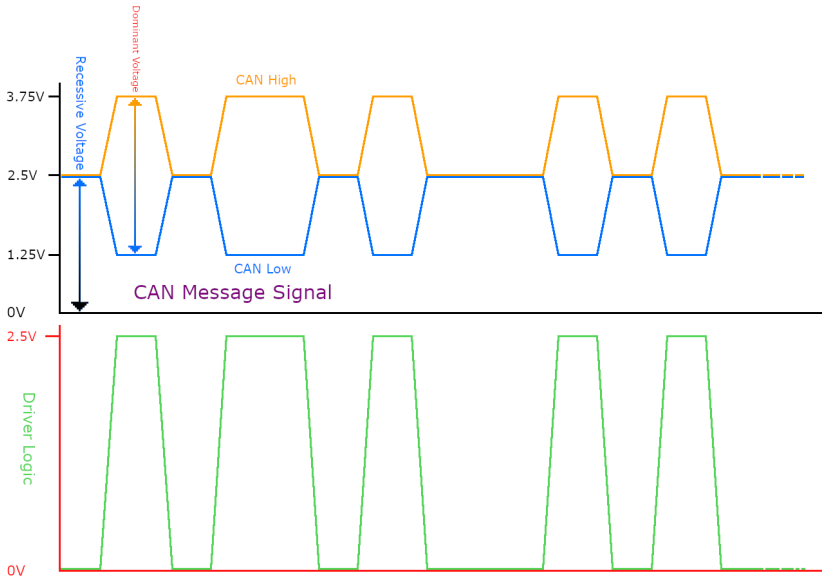
Testi vältel mõõtetulemuste reaalajas kuvamiseks kannab mikrokontroller veebiserverit, kus kuvatakse kõik loetud andmed graafikul, kusjuures on võimalus valida, mis sensori mõõtetulemusi näha tahetakse.

2 Suhtlusprotokollid

Selline seade kasutab mitmeid protokolle, et erinevate komponentidega suhelda. Õhukiiruse sensoriga suhtlemiseks kasutatakse CAN protokolli Aerospace standardit. Samas SD-kaardi mooduliga suhtlemiseks kasutatakse näiteks SPI protokolli.

2.1 Kontrollervõrgu (CAN) protokoll

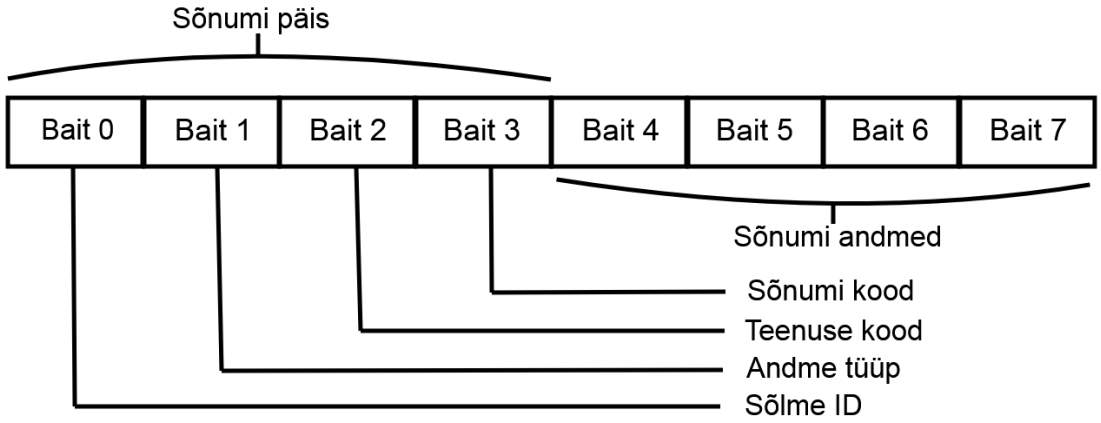
Testitav õhukiiruse sensor kasutab andmete edastamiseks CAN protokolli. CAN on diferentsiaalsignaalil põhinev süsteem, mis töötati algselt välja Boschi poolt aastail 1983-1985 autotööstuse jaoks, kuid on praeguseks laialt kasutatud protokoll seadmetes, mis on infokadu ja vigade tekkimise vastu eriti tundlikud. CAN protokoll toetab sõnumite kinnitamist, kokkupõrgete vältimist, sõnumite filtreerimist ja vigaste sõnumite automaatset uuesti saatmist, mis muudavadki selle sobivaks kriitiliste süsteemide jaoks, kus on vajalik töökindlus ja täpsus [2].
Protokoll kasutab kahte liini: *CAN High* ja *CAN Low* seadmete vahel sõnumite saatmiseks. Sõnumi lugemiseks tuleb mõõta kahe siini erinevust, kus 0 volti tähistab taanduvat signaali, mida loetakse kui loogilist ühte ning suurim erinevus (tavaliselt umbes 2 volti) tähistab dominantset signaal, mida loetakse kui loogilist nulli [joonis 1].



Joonis 1. CAN võrgu signaali näide [3]

CAN siinidel võib elektriliste signaalide saatmisel tekkida otsest peegeldusi. Tagasi peegeldunud signaalid võivad põhjustada sõnumi ebakorrektset dekodeerimist, mille tulemusel võib sõnum kaduda või muutuda. Selliste peegelduste ära hoidmiseks kasutatakse siini otspunktides 120 oomiseid takistusi. Need lõpetamise takistused sobituvad siini karakteristliku impedantsiga, mis on CAN siinil üldjuhul ligikaudu 120 oomi. Kui need puuduvad või väärtus on vale, ei sobi siini impedants, mille tulemusel tekivad peegeldused ja signaali kvaliteet võib langeda.

Antud õhukiirussensor kasutab CAN Aerospace standardit, mis on avatud ja litsentsivaba standard loodud Stock Flight Systems-i poolt aastal 1998. Standardi eesmärgiks on standardiseerida lennundussüsteemide vahelisi sõnumeid, võimaldades erinevate tootjate seadmete lihtsat ühilduvust lennundusvõrkudes. CAN Aerospace sõnum koosneb 8 baidist, kus esimesed 4 baiti on sõnumi päis ning ülejäänud 4 baiti, on sõnumi andmed, ehk sisu[4].



Joonis 2. CAN Aerospace sõnumi ehitus

Bait 0 - Sõnumi esimene bait on Sõlme ID, mis on iga seadme unikaalne identifikaator, et seda eristada teistest sõlmedest.
Bait 1 - Sõnumi teine bait määrab andmetüübi, näiteks täisarv või ujukomaarv.
Bait 2 - Sõnumi kolmas bait määrab teenuse, ehk kas sõnum on näiteks olekuteade,

veateade, tavaline andmepäring jne.
Bait 3 - Sõnumi päise viimane bait on sõnumi kood, mida suurendatakse ühe võrra iga uue sõnumiga kuni 255ni, peale mida alustatakse uuesti nullist.
Bait 4-7 - Sõnumi andmed, näiteks sensori mõõtetulemus

CAN sõnumid ümbritsetakse raamiga, kus on oluline info sõnumi lugemiseks. Näiteks sisaldab see kontrollsummat andmete tervikluse tagamiseks samuti võib see sisaldada teavet lugemise algus- ja lõpp-punkti kohta ning muid lisavälju. CAN Raami päises on ka identifikaator CAN ID, mille abil saadakse teada, milliselt seadmelt on tulnud see sõnum. CAN ID määrab ka sõnumi prioriteedi, saatja sõlme ID ja vajadusel ka kanali, võimaldades andmete suunamist ja võrguliikluse korraldamist. Prioriteet on oluline näiteks juhul kui kaks seadet üritavad samal ajal saada sõnumi CAN siinile. Prioriteet määrab ära, milline sõnum jääb peale.

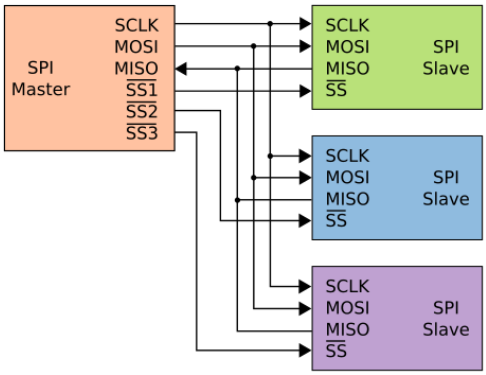
CAN identifier	System parameter name	Data type	Units	Notes
317 ($13D)	Calibrated Airspeed	FLOAT SHORT2	m/s	
321 ($141)	Heading Angle	FLOAT SHORT2	deg	+/-180°
401 ($191)	Roll Control Position	FLOAT SHORT2	%	Right: + Left: -
1008 ($3F0)	Active Nav System Track Error Angle (TKE)	FLOAT SHORT2	deg	Service Code Field contains Waypoint Number
1070 ($42E)	Radio Height	FLOAT SHORT2	m	
1205 ($4B5)	Lateral Center of Gravity	FLOAT SHORT2	% MAC	

Joonis 3. Näide CAN ID-de tabelist [5]

2.2 SPI protokoll ja ühendusviisid

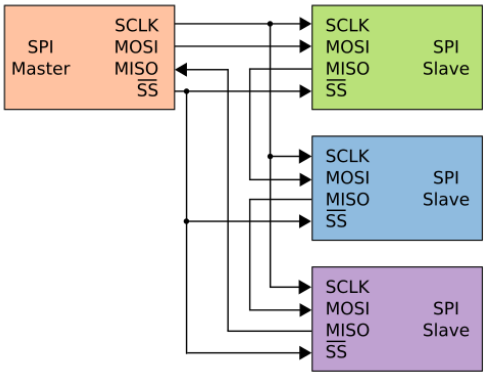
SPI protokolli arendas välja Motorola sünkroonseks järjestikuliseks suhtluseks, mida kasutatakse tihti manussüsteemides. Antud töös on SPI kasutatud kahes kohas: mikrokontrolleri ja SD-kaardi mooduli ning mikrokontrolleri ja analooglülitite vaheliseks

suhtluseks. Protokoll kasutab täisdupleks režiimis suhtlust ülema ja alluva stiilis. Neli liini ühendatakse ülema ja iga alluva vahel:
SCL (*Serial clock*) genereritakse ülema poolt, mis annab taktsignaali, mille järgi loetakse andmeid.
MOSI (*Master Output Slave Input*) saadab andmeid ülemalt alluvale.
MISO (*Master Input Slave Output*) saadab andmeid alluvalt ülemale.
CS/SS (*Chip select/Slave select*) kasutab ülem, et valida millise alluvaga suhelda.
Tüüpiliselt kasutatakse SPI-d sõltumatute alluvate konfiguratisoonis, kus kasutatakse sama SCL, MOSI ja MISO liini kõigi alluvatega, kuid igale alluvale on oma CS liin, st kokku 3 + n arv liini, kus n on alluvate arv. Sellise seadistusega kasutab ülem CS liini, et valida alluv, mis hakkab andmeid kuulama ja vastu saatma [joonis 4]. Selline ühendusviis aga vajab palju liine, mida võib olla mikrokontrolleritel limiteeritud koguses.



Joonis 4. SPI sõltumatute alluvate konfiguratsioonis [6]

Teiseks valikuks on kasutada pärgvõrgu konfiguratsioonis, kus alluvad teevad koostööd, et saata omavahel edasi ülemalt tulnud andmeid, et see jõuaks õigele alluvale. Selles konfiguratsioonis on MOSI ühendatud ühe alluvaga, ning selle alluva MISO on ühendatud järgmise alluva MOSI-ga, kuni viimase alluvani, mille MISO on ühendatud ülema MISO-ga. Nii saadab ülem andmed esimesele alluvale, mis saadab selle omakorda edasi kuni soovitud alluv on andmed kätte saanud [joonis 5]. Selline ühendusviis vajab ainult nelja liini, mis võimaldab ühendada limiteeritud koguses liinidega mikrokontrolleritel suhelda suurema arvu alluvatega[7].



Joonis 5. SPI pärgvõrgu konfiguratsioonis [8]

2.3 HTTP protokoll ja RESTful API

HTTP (Hüperteksti edastusprotokoll, *Hypertext Transfer Protocol*) on rakenduskihtide protokoll, mis võimaldab andmete edastust klientide ja serverite vahel. HTTP töötab päring-vastus mudelil, ehk klient saadab päringu ja server vastab vastavalt päringu sisule. Need koosnevad üldiselt päisest ja valikulisest sisust. Päised sisaldavad metaandmeid päringu ja vastuse kohta, näiteks teavet andmetüübi, sisu pikkuse, serveri seisundi ja autentimise kohta. Need aitavad kliendil ja serveril õigesti töödelda edastatavaid andmeid. Sisu sisaldab tegelikke andmeid, mida klient või server edastab. HTTP-d kasutatakse koos RESTful API (Rakendusliides, *Application Programming Interface*) päringutega. RESTful API kasutab HTTP protokolli ning defineerib standardsed reeglid, mis muudab süsteemide vahelise suhtluse ühtlaseks ja lihtsalt hallatavaks. Näiteks enim kasutatavad päringud on GET ja POST. Kui klient tahab saada serverilt andmeid võib selleks kasutada GET päringut, millele server üldiselt vastab, kas (kuid mitte ainult) JSON või XML (Laiendatav märgistuskeel, *Extensible Markup Language*) formaadis. Sarnaselt võib klient tahta saata andmeid serverisse kasutades POST päringut, millele server üldiselt vastab koodiga, indikeerimast, kas päring õnnestus (näiteks kood 200).

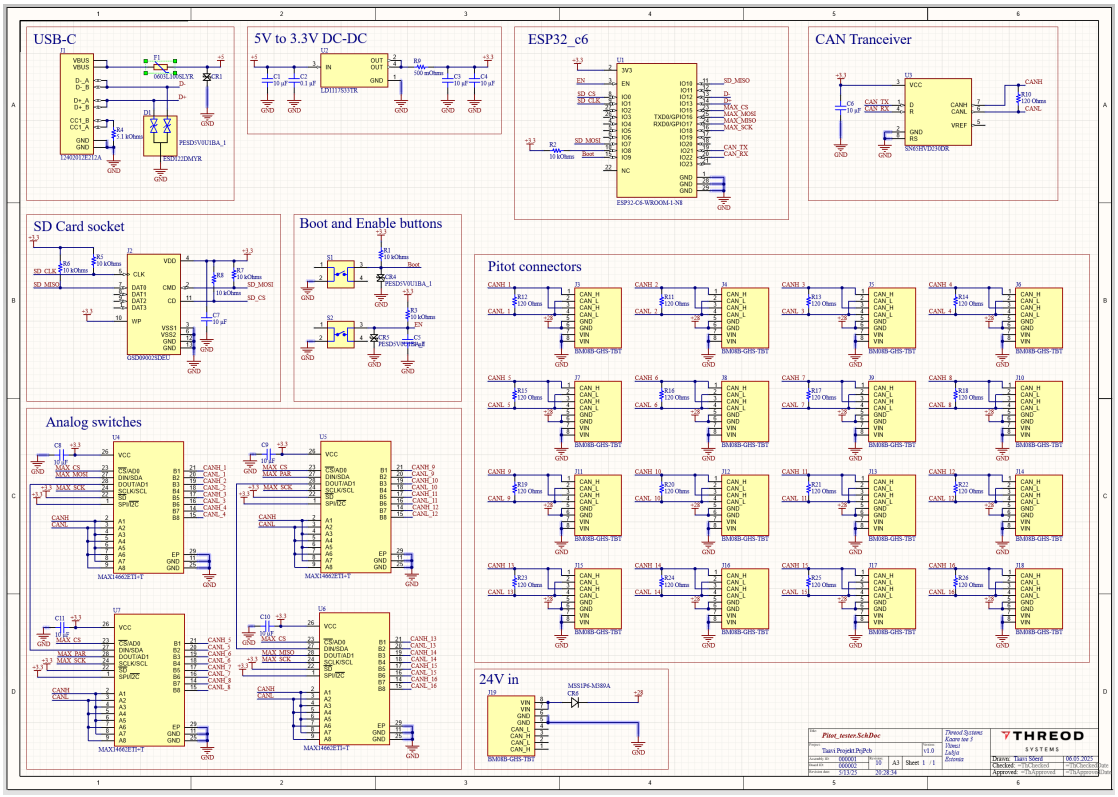
Joonis 6. Näide antud töös kasutavast POST päringust. Kuvatõmmis on tehtud Postman programmist

3 Valminud seade

Töö käigus valmis seade, mis kasutab ülal toodud protokolle ja ühendusi, luues selleks vastavad elektrilised ühendused ja ka tarkvara. Riistvara disainimiseks sai kasutatud Altium Designerit, millega loodi nii skeem kui ka trükkplaadi disain. Tarkvara kirjutati C++ keeles kasutades Arduino IDE-d.

3.1 Riistvara

Joonisel 7 on näha valminud skeemi. Skeem on jaotatud punaste kastidega osadeks, lihtsamaks lugemiseks. Tähtsamad osad on lahti seletatud peatükkides 3.1.1 kuni 3.1.6.



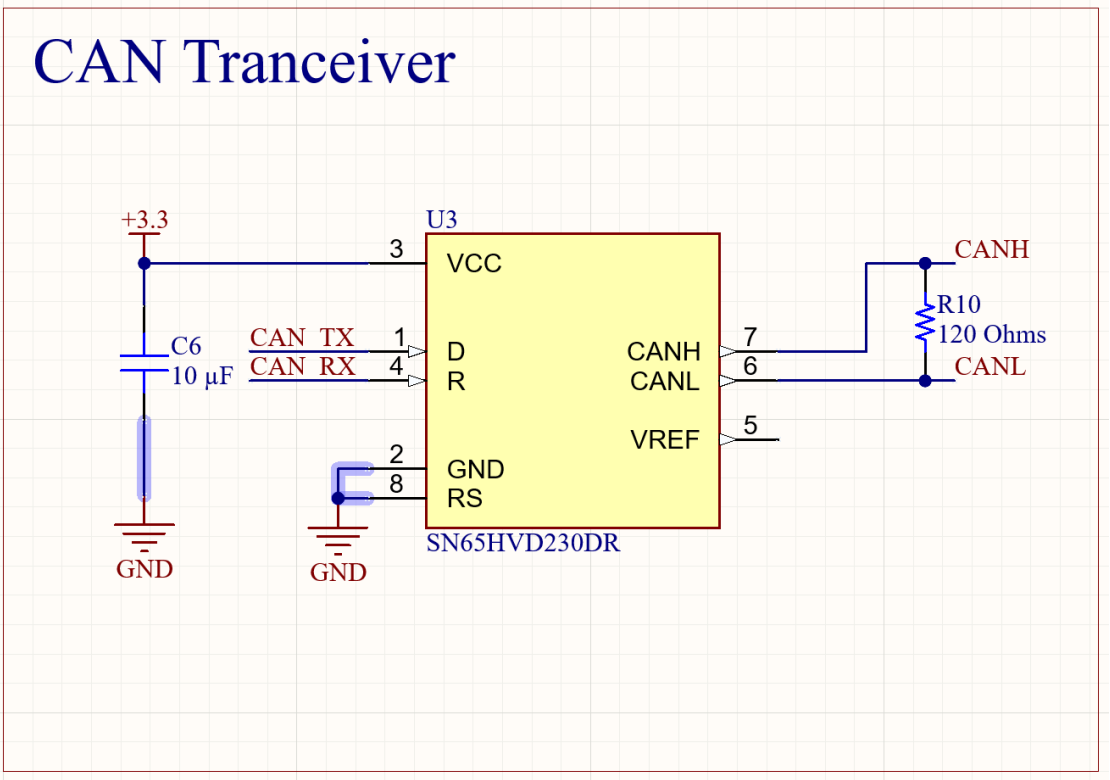
Joonis 7. Töö käigus valminud skeem

3.1.1 Protsessor

Testseadme juhtimiseks oli vaja valida välja mikrokontroller, mis suudaks korraga suhelda mitme sensoriga, salvestada andmeid SD-kaardile ja ka reaalajas kuvada neid andmeid. Lõplikuks valikuks sai ESP32, kuna see on laialt kättesaadav, hästi dokumenteeritud ning lihtsasti programmeeritav, mistõttu on see antud projekti jaoks hea valik. ESP32 perekonnast sai valitud ESP32c6 kuna see oli juba endal olemas ja sai hakata koheselt prototüüpimist alustada. ESP32c6 on Espressifi mikrokontroller, mis põhineb 32-bitisel RISC-V arhitektuuril, see toetab kaasaegseid ja käesoleva projekti jaoks vajalikke suhtlusprotokolle nagu Wi-Fi 6 ja ka TWAI kontroller, mis on CAN protokolli Espressifi poolne rakendus[11].

3.1.2 CAN suhtlus

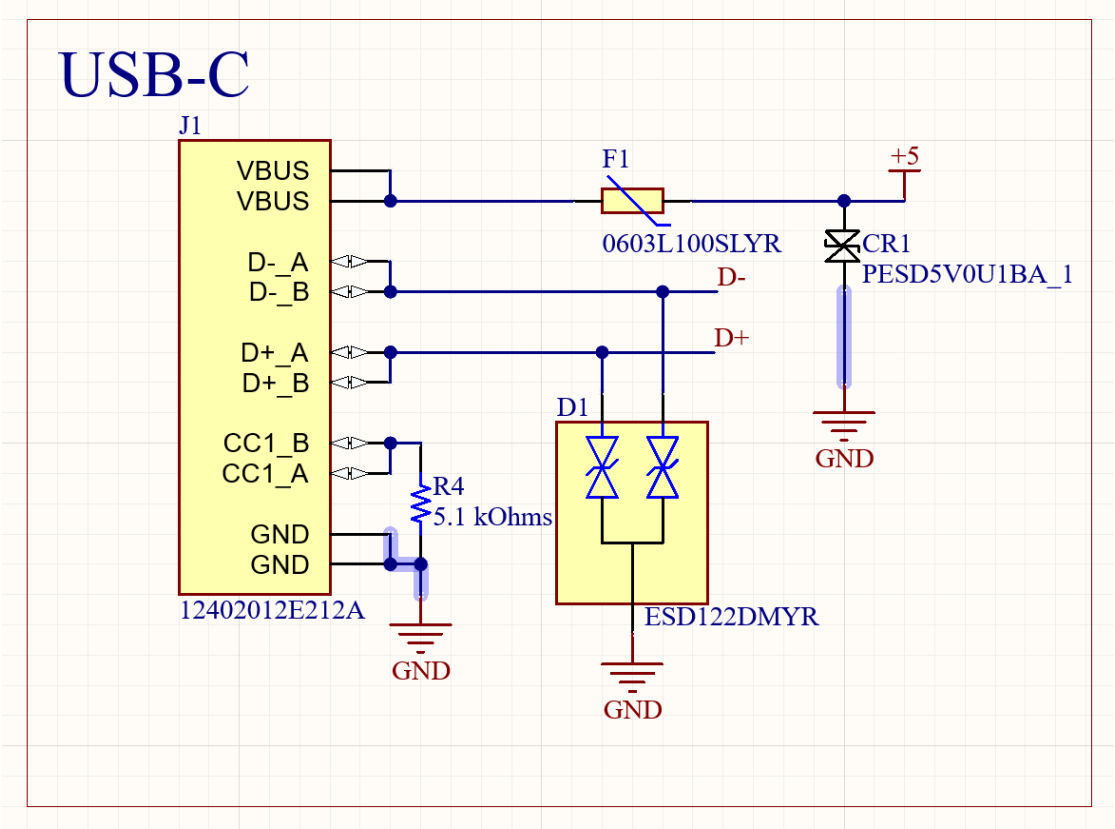
TWAI on CAN kontrolleri liides ESP32 platvormil, mis tegeleb loogikakihi ülesannetega, ehk koostab, dekodeerib ja haldab CAN sõnumeid. Kuna sellel puudub mehhanism, mis muudaks loogikatasandi signaalid diferentsiaalsignaalideks (CAN High ja CAN Low), on vaja juurde lisada väline CAN transiiver. Selleks sai valitud SN65HVD230DR kiip, mis võimaldab teha vajaliku elektrilise teisenduse, et saata ja vastu võtta CAN sõnumeid füüsilises CAN võrgus. ESP32c6 koostab CAN sõnumi ja edastab selle oma sisseehitatud CAN kontrolleri kaudu SN65HVD230DR kiibile, mis omakorda edastab signaali CAN võrku (CAN High ja CAN Low liinidele) [12].



Joonis 8. CAN transiiveri skeem

3.1.3 Toide ja programmeerimine

Enamik plaadil kasutatavaid komponente kasutavad toiteks 3,3 volti pinget, mistõttu oli vaja valida sobiv toiteallikas, mis oleks võimeline tagama stabiilselt sellist pinget. Selleks allikaks sai valitud USB-C liides, kuna selle ühenduskaablid on tänapäeval laialdaselt kättesaadavad ning liidese tehniline spetsifikatsioon on põhjalikult dokumenteeritud. Samuti saab sama liidest kasutada ka ESP32 programmeerimiseks.



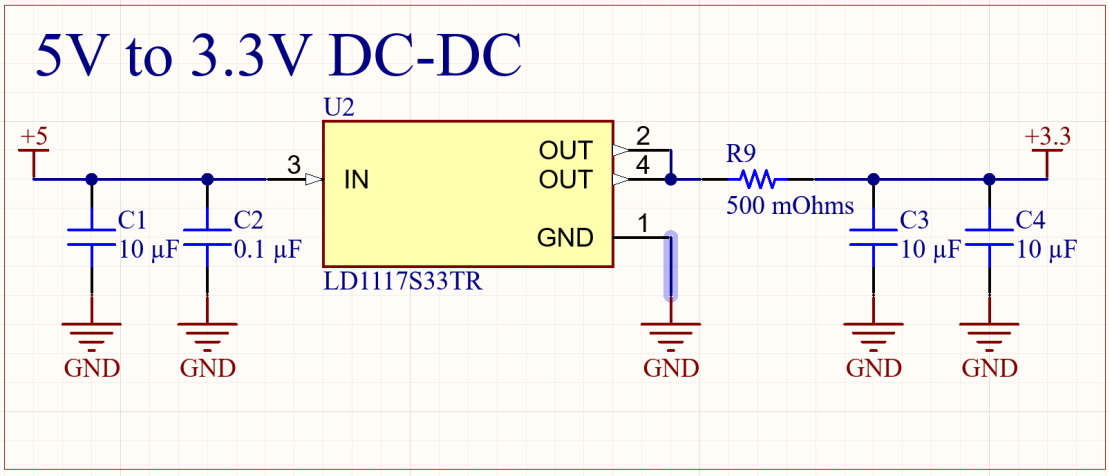
Joonis 9. USB-C liidese skeem

Toitele sai lisatud ülevoolu kaitse. See kaitse koosneb isetaastuvast kaitsmest, mis on näha joonisel 9 märgituna F1, ja ESD (elektrostaatiline laengulahendus, *Electrostatic Discharge*) kaitse dioodist, mis on näha joonisel 9 märgituna CR1. Normaalses olekus laseb F1 voolu läbi, kuid CR1 voolu läbi ei lase, ehk vool jookseb läbi F1 otse edasi, kusjuures CR1 läbimatta. Ülevoolu korral avaneb CR1 ja hakkamb voolu juhtima otse maasse tekitades veel suurema voolu, mis aktiveerib F1 kaitsme. Selle tulemusena enam F1 voolu läbi ei lase ja plaati enam ei toideta. Protsesor tarbib maksimaalselt umbes 0,5 amprit, seega sai valitud F1, mis rakendub 1 ampri juures. See lubab protsessoril tarbida rohkem voolu kui vaja, kuid keelab voolu, mis võib protsessorit või muid komponente lõhkuda.

Lisaks toiteahelale sai arnane kaitse lisatud ka andmesiinile (D- ja D+). Andmesiini kaitseks sai valitud spetsiaalselt USB-liidestele mõeldud ESD-kaitsekomponent ESD122DMYR, mis sisaldab kahte ESD kaitsedioodi, tagades seega andmeühenduse

kaitse elektrostaatiliste impulsside vastu.[13]

LDO (lineaarne pingeregulaator, *Low Dropout Regulator*) on elektrooniline komponent, mis võimaldab muuta kõrgemat sisendpinget madalamaks ja stabiilseks väljundpingeks. Antud projektis sai LDO-d kasutatud, et USB-st saadud 5 volti toitepinge alandada stabiilseks 3,3 volti pingeks, mis on vajalik komponentide toitmiseks [14].

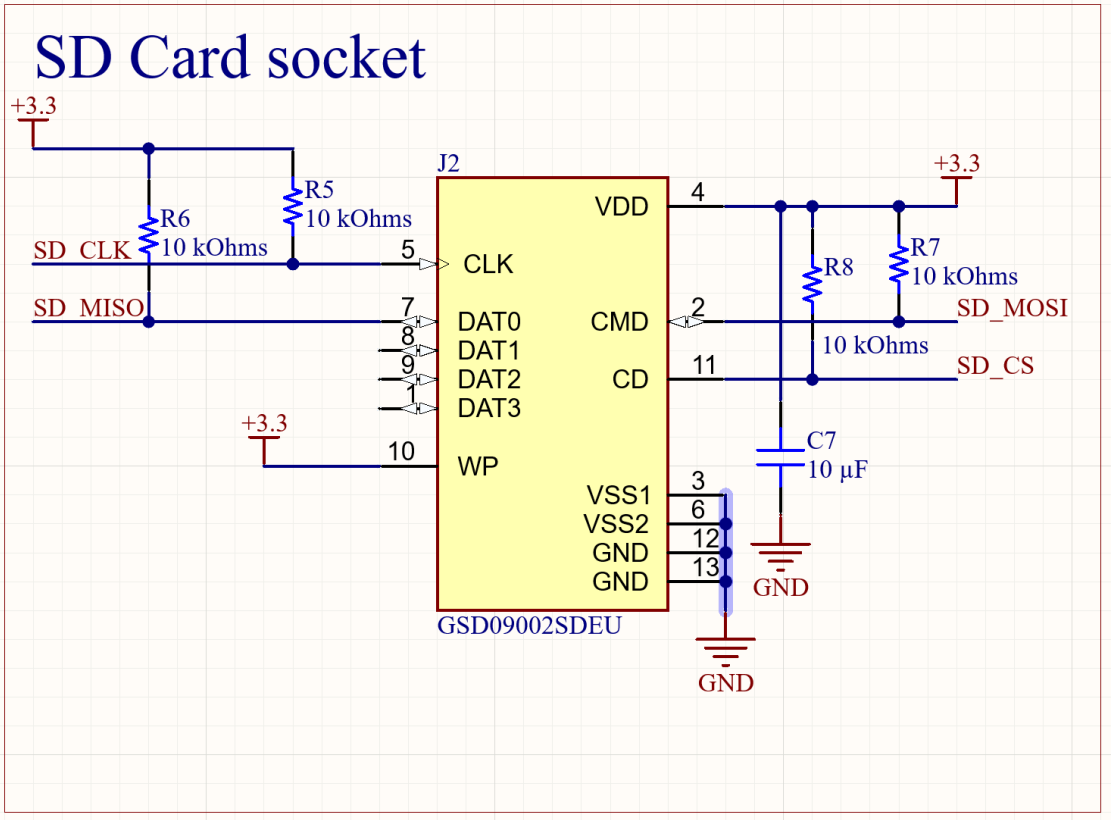


Joonis 10. LDO skeem

3.1.4 SD-kaart

Andmete salvestamiseks SD-kaardile on valisin GSD09002SDEU mooduli, mille saab ühendada ESP32-ga läbi SPI. Igale SPI liinile sai lisatud 10k oomised *pull-up* takistid, mille ülesanneteks on vältida ujuvaid olekuid, ehk kui signaali aktiivselt ei juhita, oleksid liinid kindlas kõrges olekus.

SPI-režiimis kasutab SD-kaart andmete edastamiseks ainult ühte andmeliini (DATA0, joonisel 11 nimetatud DAT0), mitte kõik nelja (DATA0-DATA3), mis on ette nähtud kiiremaks paralleelseks suhtluseks. Antud töös on andmemahud väikesed ja suurt kiirust vaja ei ole. Seega sai kasutusele võetud lihtsam ja levinum suhtlusviis.



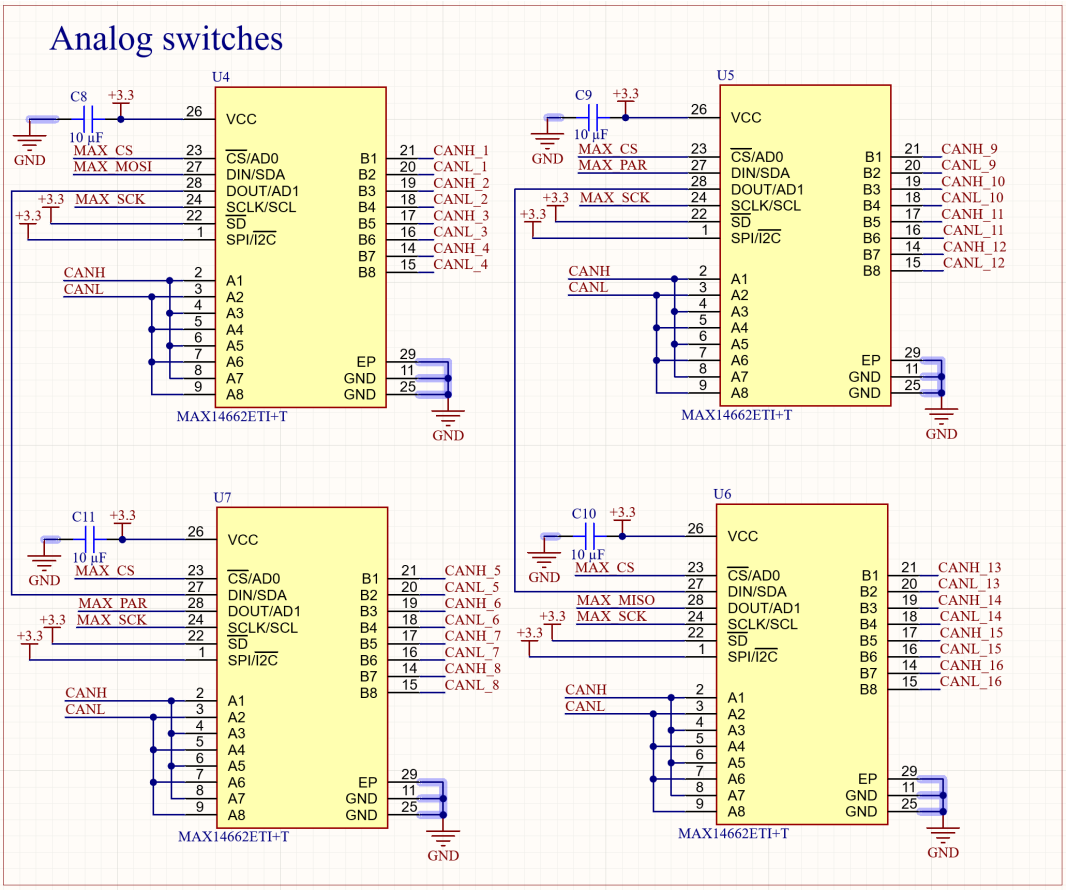
Joonis 11. SD-Kaardi skeem

3.1.5 Analooglülitid

Iga õhukiiruse sensori CAN ID on sama, seega ei ole võimalik neid ühele CAN siinile pannes eristada. Kuna eesmärk on sensorite töökorda testida, siis peab neil vahet tegema. CAN ID muutmine on aga riskantne, kuna sellisel juhul peab peale testi iga sensori CAN ID tagasi muutma. Testi lõpus oleks aga raske kinnitada, et iga sensor sai oma algse CAN ID tagasi, kuna sellisel juhul oleks jälle kõik CAN ID-d samad ja tekiks sarnane probleem, mis oli alguses. Seega said kasutusele võetud analooglülitid, millega ESP lülitab sisse ainult ühe sensori siini korraga, ehk igal ühel ajahetkel on reaalselt ühendatud ainult üks sensor. Selline lähenemine aitab programmaatiliselt kontrollida, mis sensoriga suhelda, ning seega saab ilusti sensoritel vahet teha ilma CAN ID-d muutmata.

Analooglülitiks sai kasutusele võetud MAX14662 kiip. Sellel kiibil on kaheksa paari (16 kanalit) kanaleid. Antud töös sai kasutusele võetud neli sellist kiipi kasutades SPI

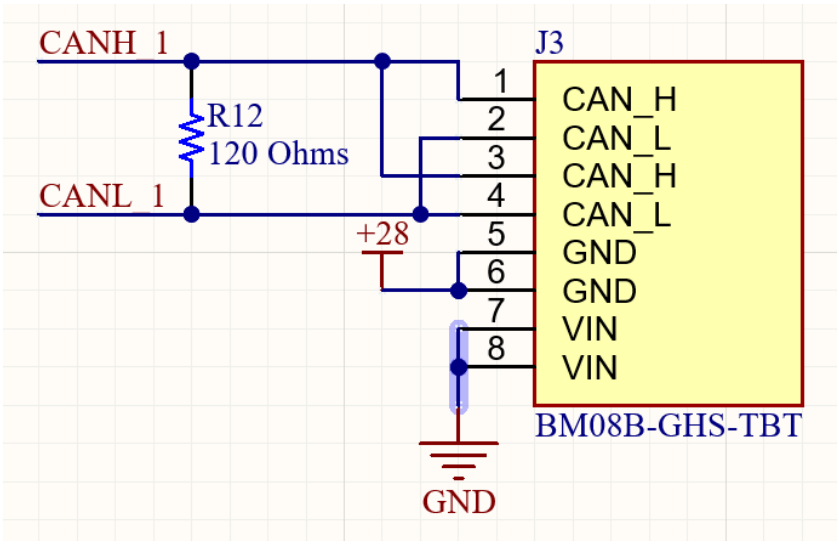
pärgvõrgu konfiguratsiooni. Iga sensori CAN siin on ühendatud ühte lüliti kanalipaari. Kanalipaari sisselülitamiseks seatakse vastavad bitid kõrgeks, ülejäänud kanalid jäävad väljalülitatuks (signaal läbi ei lähe). Kiipide kõik ühendused on näidatud joonisel 12.



Joonis 12. Analooglülitite skeem

3.1.6 Õhukiirussensorite ühendus

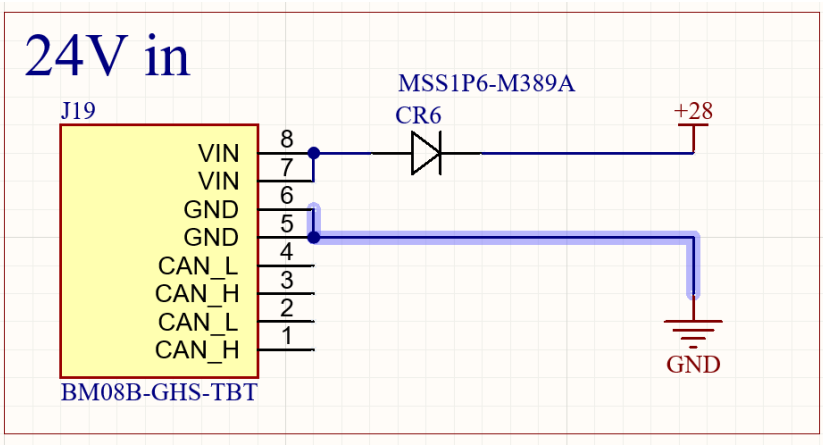
Iga õhukiirussensoril on oma ühendus läbi BM08B-GHS-TBT pesa. Analooglülitid ühendavad CANH ja CANL liinid CAN transiiveriga. CANH ja CANL liinid on dubleeritud töökindluse eesmärgil - kui üks liin peaks katkema jääb teine alles. Selline dubleeritud on õhukiiruse sensorisse sisseehitatud ning sensoriga suhtlemiseks peab kasutama mõlemat siini.



Joonis 13. Õhukiirussensori ühenduse pesa skeem

Õhukiiruse sensor ei tööta 3.3 voldi ega 5 voldi peal, mis on juba plaadil olemas, vaid 28 voldi peal. Selleks on vaja eraldi toidet, kuna USB-C nii suurt pinget otse ei anna. Probleemi lahendamiseks paigaltati plaadile lisa BM08B-GHS-TBT pesa, mis tagab 28 V toitepinge ja maa-ühenduse.

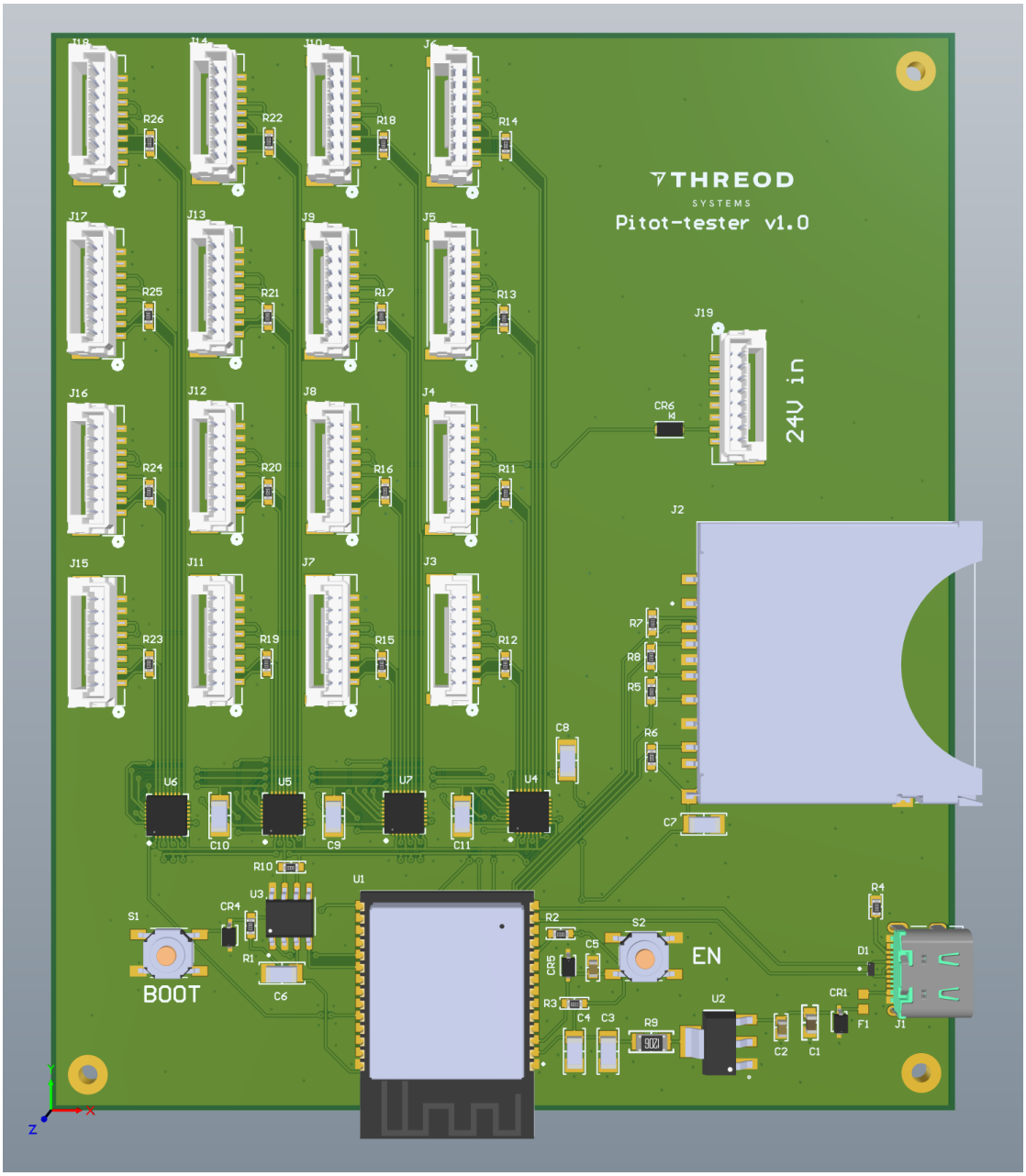
Pesaga on jadamisi ühendatud ka diood (joonisel 14 märgituna CR6), mille ülesandeks hoida ära kahjustused valepidi ühendustest.



Joonis 14. Õhukiirusanduri pesa toite skeem

3.1.7 Valminud trükkplaat

Ülal toodud skeemi alusel disainiti valmis trükkplaat, mis on näha joonisel 15.



Joonis 15. Valminud trükkplaadi 3D mudel

3.2 Tarkvara

Seadmele sai välja töötatud tarkvara, mis haldab komponentide juhtimist, mäluhaldust, andmete töötlemist ja salvestamist ning reaalajas kuvamist. Järgnevates peatükkides kirjeldatakse lahti selle tarkvara tööpõhimõtted.

3.2.1 Andmete salvestamine

Sensori andmete salvestamiseks sai valitud JSON formaat. Mõõteandmete kogumiseks sai loodud kaks puhvrit muutmällu. Esimese puhvri täitumisel (milleks on 2000 andmepunkti) salvestatakse see puhver ümber SD-kaardile. Andmete salvestamine aga võtab aega, mil ajal peab seade suutma paralleelselt siiski koguma mõõtetulemusi. Selleks tuleb kasutusele teine puhver, mida kasutatakse samal ajal kui esimene on hõivatud. Selline loogika võimaldab andmete kõgumist ja salvestamist paralleelselt üksteist segamata.

Andmete salvestamiseks SD-kaardile sai kasutusele võetud SD-kaardi ja SPI protokolli teek, mis võimaldab standartset kommunikatsiooni andmekandjaga. Rakenduse käivitamisel initsialseeritakse SD-kaart määrates vajalikud SPI jalad ning kuvatakse läbi Seriali SD-kaardi info. JSON faili lisatakse kõik mõõtetulemused vastava sensori objekti alla. Selline struktuur võimaldab efektiivselt hallata ja organiseerida suurel hulgal sensoritest pärinevaid andmeid ning lihtsustab hilisemalt analüüsi ja ka visualiseerimist.


```
{
  "sensor1": {
    "Wind-speed": [
      {
        "ts": 1747583148,
        "value": 0.370694
      },
      {
        "ts": 1747583150,
        "value": 0.293446
      }
    ]
  },
  "sensor2": {
    "Wind-speed": [
      {
        "ts": 1747583148,
        "value": 0.367228
      },
      {
        "ts": 1747583150,
        "value": 0.293986
      }
    ]
  }
}
```

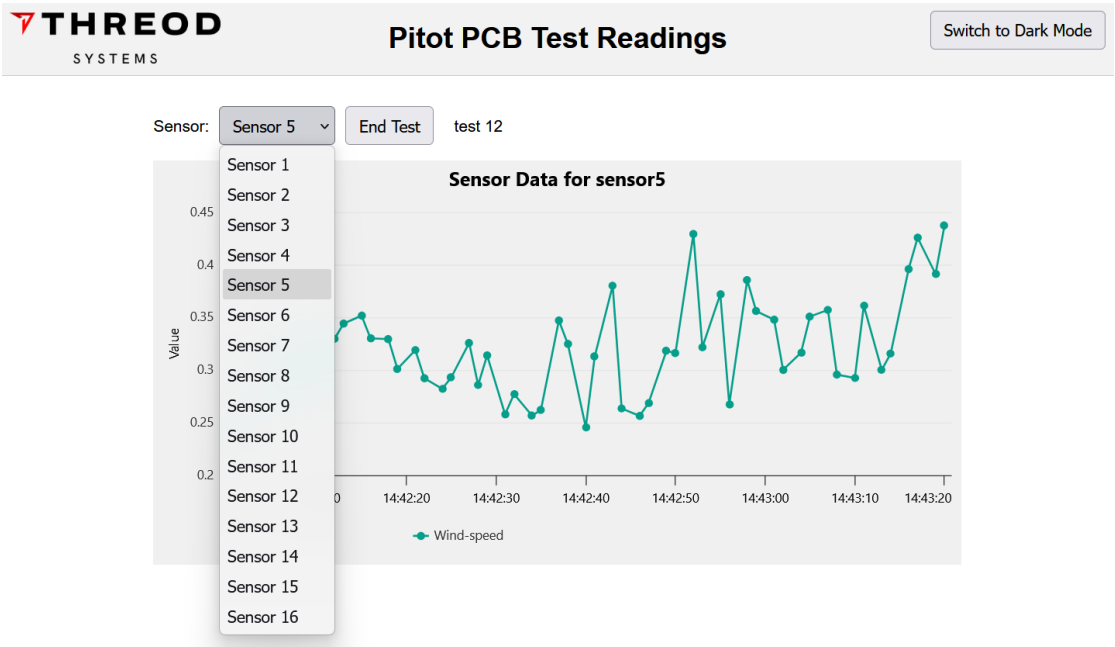

Joonis 16. Näide salvestatud JSON failist (lühendatult)

3.2.2 Andmete reaalajas kuvamine

ESP32 ühendub võrguga, mis on seadistatav SD-kaardiga. Seadme tööle panekul loeb see sisse SD-kaardilt konfiguratsiooni faili, mis peab sisaldama Wi-Fi SSID ja parooli. Neid andmeid kasutades üritab seade logida võrku ja hakata seal kandma veebiserverit.

Veebiserver näitab Highcharts teeki kasutades interaktiivset graafikut mõõtetulemustest. Iga kolme sekundi tagant tehakse GET päring koos ajatempliga viimasest mõõtetulemusest. ESP32 vastab sellele päringule JSON formaadis failiga, milles on mõõtetulemused, mis on loetud peale päringus olevat aega.

Selleks, et kiirendada andmete liikumist võrgus ja vähendada koormust saadetakse korraga ainult ühe sensori mõõtetulemused. Läbi kasutajaliidese saab valida 16 sensori vahel ning vastavalt sellele valikule saadetaksegi andmete päring (vt. joonis 17).



Joonis 17. Kuvatõmmis veebiserverist ja sensori valikust

Kasutajaliidesega on ka võimalik alustada ja lõpetada testi. Testi alustamisel on vajalik seada testile nimi, millest tuleneb hiljem salvestatud faili nimi. Testi staatuse informatsioon saadetakse ka eelnevalt kirjutatud andmete päringuga kaasa, et vältida olukorda, kus üks kasutaja alustab testi ning hiljem alustab teine kasutaja uut, kuigi üks

juba on aktiivne. Peale testi alustamist ilmub "End test"nupp, millega saab salvestada mõõdetud andmed SD-kaardile ning lõpetada testi. Iga testi alustamise ja lõpetamisega tühjendatakse ka graafik, et vähendada liigset informatsiooni.

Kuna andmed salvestatakse ainult juhul, kui test on alustatud, siis ilma aktiivse testita on kasutajaliidesel graafiku all punaselt kirjas "no active test". See aitab vältida olukorda, kus kasutaja alustab testiga, kuid unustas vajutada ka kasutajaliidesel "start test"ja seega andmed ei salvestu.

3.2.3 Analooglülitid ja CAN võrgu suhtlus

Sensorite andmete kogumiseks ja haldamiseks kasutatakse analoog lüliteid ja CAN suhtlust. Konkreetse sensori valimiseks kasutatakse meetodit *selectSensor()*, mis esmalt kontrollib, et parasjagu ei toimuks CAN sõnumi edastust, vältimaks lülitust ülekande ajal. Seejärel arvutatakse välja, millise lüliti kiibi ja ja kanalipaari kaudu sensoriga ühendus luua. Kiip ning kanalipaar määratakse valemitega:

$$kiip = \frac{\text{sensori indeks}}{8\ (paaride\ arv\ ühes\ kiibis)}$$

$$kanalipaar = (sensori\ indeks\ \%\ 8\ (paaride\ arv\ ühes\ kiibis)) * 2$$

Tulemuseks saadakse 16-bitine baitide massiiv, kus iga bait vastab ühele analooglüliti kiibile. See massiiv saadetakse SPI protokolli abil välja, mis aktiveerib soovitud ühendused.

Sensoritelt andmete küsimiseks kasutatakse ülesannet *pollSensorsTask()*, mille ülesandeks on tsükliliselt küsida kõigilt sensoritelt andmeid. Selle käigus valitakse kõigepealt sensor eespool kirjeldatud meetodiga *selectSensor()*. Seejärel koostatakse CAN Aerospace standardi järgi päringud. Esmalt tehakse identifitseerimispäring ja seejärel õhukiiruse päring. Mõlemad sõnumid saadetakse üksteise järel välja TWAI ja CAN transiiveri kaudu. Päringule oodatakse vastust kuni 100 millisekundit. Kui vastus saabu, siis see dekodeeritakse ja salvestatakse mõõtetulemus koos ajatempliga muutmällu.

Tsükli ajastus on planeeritud nii, et igalt sensorilt saaks mõõtetulemuse ligikaudu iga 1,5 sekundi tagant. Selle saavutamiseks on ühe sensori küsitlemise tsükli aeg umbes 94

millisekundit.

$$t_{\text{delta}} = \frac{\text{tskli aeg } (1500 \text{ ms})}{sensorite\ arv\ (16\ tk)} \approx 94\ ms$$

Kui kogu tsükkel saab valmis kiiremini, rakendatakse täiendavat viivitust, et tagada konstantne tsükli kestvus ja vältida mikrokontrolleri liigset koormamist.

4 Tulemused

Valminud seadet ei olnud võimalik testida kliimakambris, kuna kliimakamber ei olnud veel valmis tehtud. Selle asemel viidi test läbi toatermperatuuril, ventilaatoriga õhuvoogu tekitades, jagatuna kahte 45-minutilisesse faasi. Esimeses faasis seisid sensorid ilma ventilaatorita ning teises faasis tekitati ventilaatoriga õhuvoog umbes 1,5 m/s.

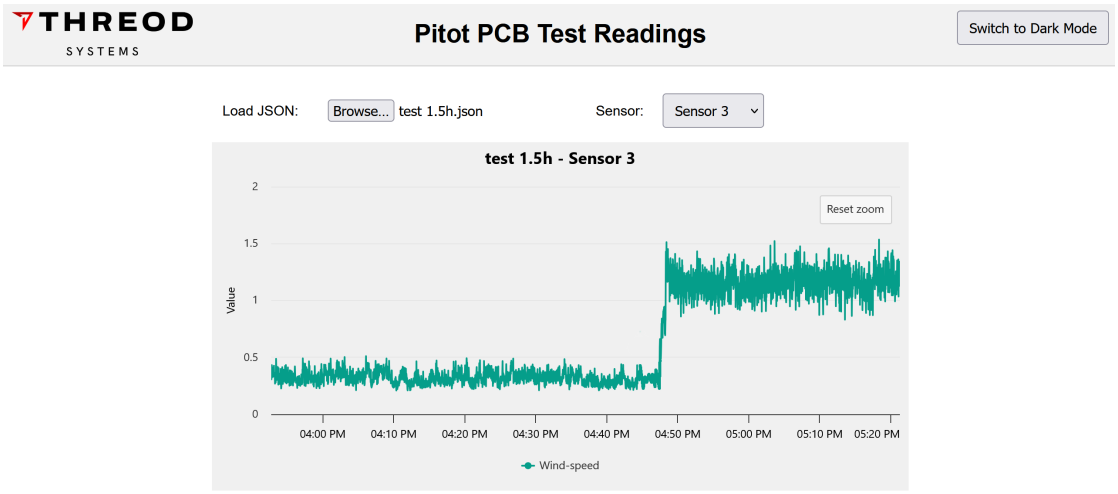
Peale testi sooritamist tuli kogutud andmed analüüsida, et hinnata süsteemi toimivust ja tuvastada võimalikke vigu. Selleks on SD-kaardile paigaldatud veebiliides (analyze.html), mis omab ülesehituselt sarnast struktuuri ESP32 peal toimuva reaalajas töötava veebiserveri lahendusega. Peamine erinevus seisneb selles, et andmed ei laeta enam otse ESP32 pealt, vaid loetakse sisse eelnevalt salvestatud JSON formaadis failidest. Kuna kõik testiandmed salvestuvad automaatselt SD-kaardile JSON failidena, võimaldab kasutajaliides mugavalt valida konkreetse katse ning kuvada selle tulemused terviklikult ja interaktiivselt.

4.1 Põhieesmärkide täitmine

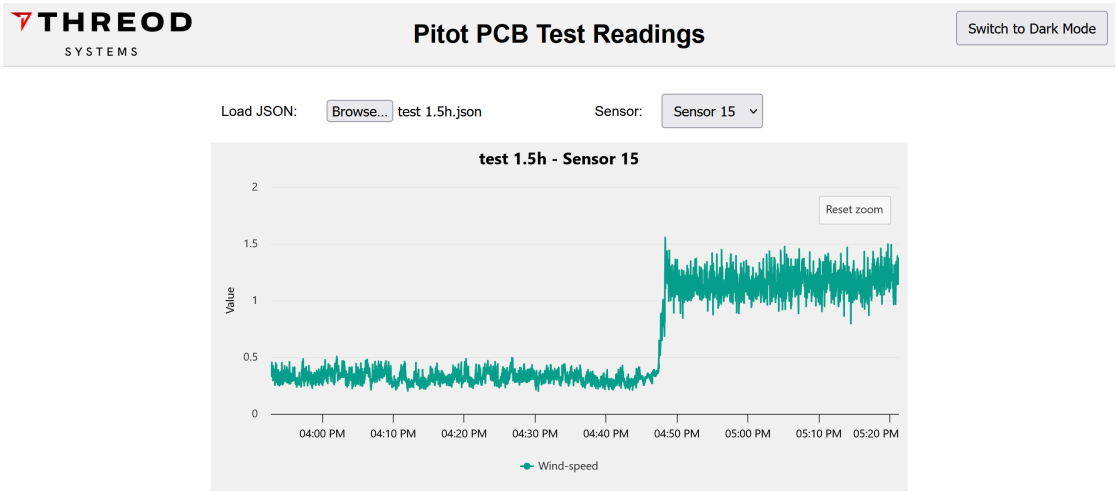
Testi käigus küsiti mõõteandmeid kõigilt 16 sensorilt, mille tulemused laekusid graafikule umbes 1,5 sekundiliste intervallidega. Sensorite mõõtetulemustes ei ilmnenud müra ega valeandmeid, seega toimusid kõik kanalivahetused ja sensori valimised ootuspäraselt.

Kahepuhvritöötlus tagas, et iga 2000 punktiline puhver kirjutati korrektselt SD-kaardile ning samalajal jätkus töö teise puhvriga. Katse lõpus oli näha, et puudusid katkised read või ajatempli ebajärjekindlused.

ESP32 veebiserver vastas GET-päringutele iga kolme sekundi tagant ning näitas kasutajaliideses korrektselt testi olekut ning valitud sensori väärtusi.



Joonis 18. Pooleteise tunnise testi sensor 3 tulemus analyze.html lehel



Joonis 19. Pooleteise tunnise testi sensor 15 tulemus analyze.html lehel

Tänuavaldused

Autor soovib avaldada tänu töö juhendajale, Artur Abelsile, kelle juhendamisel töö valmis.

Samuti sooviks tänu avaldada Threod Systemsi kollektiivile, kes andsid vahendid projekti realiseerimiseks ja abistasid töö käiku.

Viited

[1] Common European Research Classification Scheme (CERCS) Teadusvaldkondade ja -erialade klassifikaator. Vaadatud 15.05.2025 https://wiki.ut.ee/download/attachments/16581162/Common%20European%20Research%20Classification%20Scheme.pdf

[2] R. Došek, P. Janků, M. Bližňák, and P. Vařacha (2016), "Secure High Level Communication Protocol for CAN Bus"Lk. 1009–1015. Vaadatud 3.11.2024 https://publikace.k.utb.cz/handle/10563/1006778?locale-attribute=en

[3] kkohlhase (2022), ”Overview of the CAN Bus Protocol"Vaadatud 3.11.2024 https://forum.digikey.com/t/overview-of-the-can-bus-protocol/21170

[4] M. Stock (2001), "CAN Aerospace, Interface specification for airborne CAN applications"Vaadatud 05.11.2025 http://www.wetzel-technology.de/files/public/canas.pdf

[5] BavarianAviator (2009), "CANaerospace_4.jpg"Vaadatud 5.11.2024 https://en.wikipedia.org/wiki/File:CANaerospace_4.jpg

[6] Cburnett (2006), "SPI three slaves.svg"Vaadatud 5.11.2024 https://et.wikipedia.org/wiki/Fail:SPI_three_slaves.svg

[7] P. Visconti, G. Giannotta, R. Brama, P. Primiceri, A. Malvasi ja A. Centuori (2017), "Features, operation principle and limits of SPI and I²C communication protocols for smart objects: A novel SPI-based hybrid protocol especially suitable for IoT applications,"*International Journal on Smart Sensing and Intelligent Systems* Lk. 274-276. Vaadatud 05.03.2025 https://doi.org/10.21307/ijssis-2017-211

[8] Cburnett (2006), "SPI three slaves daisy chained.svg"Vaadatud 5.11.2024 https://et.wikipedia.org/wiki/Fail:SPI_three_slaves_daisy_chained.svg

[9] Cburnett, Jordsan (2010), "SPI timing diagram2.svg"Vaadatud 11.5.2025 https://et.wikipedia.org/wiki/Fail:SPI_timing_diagram2.svg

[10] M.-E. Stanciu, R.-M. Teodorescu, G.-A. Iordachescu, and D.-A. Visan (2023), "Transmission Techniques in Data Acquisition Systems using Arduino and ESP32,"in

2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Lk. 1–6. doi: 10.1109/ECAI58194.2023.10193921.

[11] Espressif Systems (2023), „ESP32-C6 Technical Reference Manual“, Vaadatud 3.05.2025 https://www.espressif.com/sites/default/files/documentation/esp32-c6_technical_reference_manual_en.pdf

[12] Texas Instruments (2018), „SN65HVD23x 3.3-V CAN Bus Transceivers“, Vaadatud 3.05.2025 https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=https%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Fsn65hvd230

[13] Texas Instruments (2018), „ESD122 2-Channel ESD Protection Diode for USB Type-C and HDMI 2.0“, Vaadatud 15.05.2025 https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=https%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Fesd122

[14] STMicroelectronics (2025), „Adjustable and fixed low drop positive voltage regulator“, Vaadatud 15.05.2025 https://www.st.com/content/ccc/resource/technical/document/datasheet/99/3b/7d/91/91/51/4b/be/CD00000544.pdf/files/CD00000544.pdf/jcr:content/translations/en.CD00000544.pdf

Lisad

Lisa 1. Tarkvara failid .zip failina

Lisa 2. Altium Designeri projekti failid valminud trükkplaadist

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Taavi Sõerd

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   “Mehitamata õhusõiduki autopiloodi õhukiirusandurite trükkplaatide testseade”

   mille juhendaja on Artur Abels

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace’i kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Taavi Sõerd
20.05.2025