TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND

Elektroenergeetika ja mehhatroonika instituut

ÕHU VOOLUHULGA MÕÕTEMOODULI VÄLJATÖÖTAMINE JA LIIDESTAMINE DIAGNOSTIKASEADMEGA

DEVELOPMENT OF AIRFLOW MEASUREMENT MODULE AND INTERFACING WITH DIAGNOSTIC SYSTEM

BAKALAUREUSETÖÖ

Üliõpilane: Reimo Saart

Üliõpilaskood: 154988

Juhendaja: Taavi Möller, insener

Kaasjuhendaja: Eiko Priidel

Tallinn, 2019

(Tiitellehe pöördel)

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.

Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

"....." ..... 201.....

Autor: .....

/ allkiri /

Töö vastab bakalaureusetöö esitatud nõuetele

"....." ..... 201.....

Juhendaja: .....

/ allkiri /

Kaitsmisele lubatud

"....."..... 201... .

Kaitsmiskomisjoni esimees .....

/ nimi ja allkiri /

TTÜ elektroenergeetika ja mehhatroonika instituut

LÕPUTÖÖ ÜLESANNE

Üliõpilane: Reimo Saart 154988MAHB

Õppekava: MAHB, Mehhatroonika

Juhendaja: Insener, Taavi Möller, 620 3706

Kaasjuhendaja: Engineering Manager, Eiko Priidel, Teleplan Estonia OÜ, 5062482, eiko.priidel@teleplan.com

Lõputöö teema:

Õhu vooluhulga mõõtemooduli väljatöötamine ja liidestamine diagnostikaseadmega

Development of airflow measurement module and interfacing with diagnostic system

Lõputöö põhieesmärgid:

1. Õhu vooluhulga mõõtmisvõimaluste võrdlus ja sobivuse analüüs
2. Elektroonika komponentide ja seadmete valik ning seadme valmis ehitamine
3. Tarkvararakenduse loomine testpingiga liidestamiseks Modbus andmesideprotokolliga

Lõputöö etapid ja ajakava:

Nr	Ülesande kirjeldus	Tähtaeg
1	Õhu vooluhulga mõõtmisvõimaluste võrdlus ja sobivuse analüüs	25.03.2019
2	Elektroonika projekteerimine(elektriskeem, trükkplaat)	08.04.2019
3	Tarkvararakenduste loomine ja seadistamine testpingiga liidestamiseks Modbus andmesideprotokolliga	23.04.2019
4	Dokumentatsioon	30.04.2019
5	Töö vormistamine, trükkimine, köitmine	14.05.2019

Töö keel: Eesti keel **Lõputöö esitamise tähtaeg:** "21."mai 2019a

Üliõpilane: Reimo Saart ..... "....."....201….a
/allkiri/

Juhendaja: Taavi Möller ..... "....."....201….a
/allkiri

Kaasjuhendaja: Eiko Priidel ..... "....." 201….a

SISUKORD

EESSÕNA

Käesolev bakalaureuse lõputöö on koostatud autori töökohas Teleplan Estonia OÜ. Autor tänab oma juhendajaid töö valmimisele kaasa aitamise eest.

LÜHENDITE JA TÄHISTE LOETELU

CRC - Veakontrollkood (*Cycle Redundancy Check*)

DC - Alalisvool (*Direct Current*)

EEPROM - Programmeeritav elekterkustutusega püsimälu (*Electrical Erasable Programmable Read-Only Memory*)

I2C - Kahejuhtmeliides (*Inter-Integrated Circuit*)

JTAG - Ühendatud Testimisrühm (*Joint Test Action Group*)

PC - Personaalarvuti (*Personal Computer*)

UART - Universiaalne asünkroonne saatja-edastaja (*Universal Asynchronous Receiver-Transmitter*)

SISSEJUHATUS

Ettevõte Teleplan Estonia OÜ tegeleb erinevate elektroonika - ja optikaseadmete remondiga. Ettevõte pakub teenust suurtele rahvusvahelistele telekommunikatsiooni firmadele, kes soovivad oma seadmeid parandada. Selleks, et tuvastada probleeme, neid parandada ning seadme parandamise õigsust kontrollida, on ettevõttel loodud erinevad automatiseeritud testpingid selle testimiseks.

Lõputöö teema valik baseerus Teleplan Estonia OÜ-le kliendi poolt esitatud nõudest lisaks tüüpilisele testimisele testida nende digibokside ventilaatorite töökorda. Ettevõttel on olemas diagnostikaseade, mida kasutatakse digibokside erinevate liideste testimiseks kuid ventilaatori tesimiseks puudus vastav riist– ja tarkvara. Digibokside parandamisel tekkis olukordi, kus digibokside lahti võtmisel ning kokku panemisel panid tehnikud ventilaatori või selle juhtmed valet pidi tagasi ning selle tagajärjel töötas ventilaator valet pidi. Samuti võib ventilaatori rikkeid tekitada selles olev tolm või muu mustus.

Lõputööga seonduva elektroonikaskeemide ning trükkplaadi disainimiseks kasutati Altium Designer tarkvara, C programmikoodid kirjutati Eclipse keskkonnas ning Pythoni programmikoodid kirjutati IDLE keskkonnas.

Lõputöö esimeses peatükis antakse lühike ülevaade lõputööga seonduvast diagnostikaseadmest. Samuti kirjeldatakse õhu vooluhulga mõõtmiseks erinevaid kasutatavaid mõõtmisviise, mõõtemooduli valikut ning selle tehnilisi parameetreid.

Teine peatükk keskendub mõõtemooduli elektroonika disainimisele ja koostamisele. Põhjendatakse elektroonikas kasutatavate põhikomponentide valikut, kirjeldatakse nende parameetreid, elektriskeeme ning nende põhjal trükkplaadi disainimist.

Kolmandas peatükis kirjeldatakse antud bakalaureuse töö raames kasutatud andmesidmeprotokolle mõõteseadme ja mikrokontrolleri ning mikrokontrolleri ning diagnostikaseadme vahel. Samuti kirjeldatakse mõõtemooduli tarkvara loomist kasutatava riistvara ning andmesideprotokollide toimimiseks.

Neljandas peatükis räägitakse mõõtemooduli Modbus andmesideprotokolli tarkvara testimisest selle liidestamisel diagnostikaseadmega.

Lõputöö lisades on välja toodud antud töö käigus loodud elektroonikaskeem ja programmikoodid. Samuti on välja toodud Teleplan mehaanika osakonna poolt loodud mehaanilise lahenduse joonised antud mõõtemoodulile. Programmikoodid on kirjutatud C ja Python'i programmeerimiskeeles.

1. ÕHU VOOLUHULGA MÕÕTEMOODULI VALIK

Gaaside vooluhulga mõõtmiseks leidub turul erinevaid seadmeid, mis kasutavad vooluhulga mõõtmiseks erinevaid mõõtmisviise. Üldjuhul leidub vooluhulga mõõtmiseks seadmeid, mis on mõeldud kasutamiseks hoonete või muude suuremate objektide ventilatsioonitorudes. Mõõtemooduli valimisel tuleb arvestada toote hinnaga, antud toote füüsiliste parameetritega ning mis tüüpi gaasi tahetakse mõõta. Mõõtemoodul asetatakse Teleplani diagnostikaseadmesse Titan. Titan on Teleplani poolt loodud automatiseeritud diagnostikaseade digiboksi erinevate funktsioonide testimiseks. Antud diagnostikaseadmel on 16 pesa, mille iga pesa jaoks on vajalik toota mõõtemoodul, ning ta võimaldab teostada digiboksidele järgnevaid teste [1]:

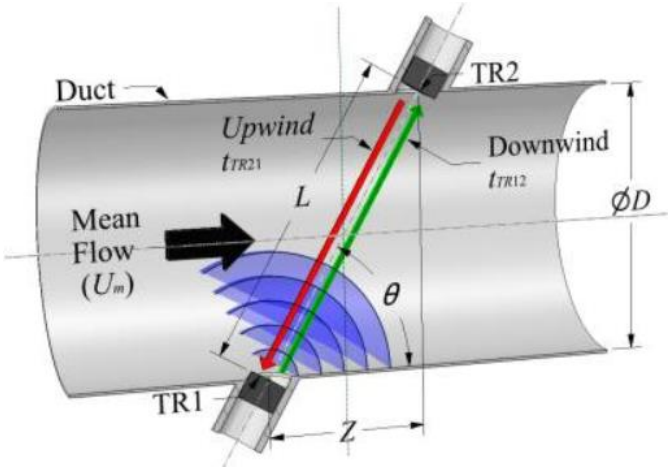
- RF
- Analoog telefon
- Ethernet
- Jada
- Audio/Video

Joonis 1.1 Diagnostikaseade Titan

1.1 Ultraheliandurid

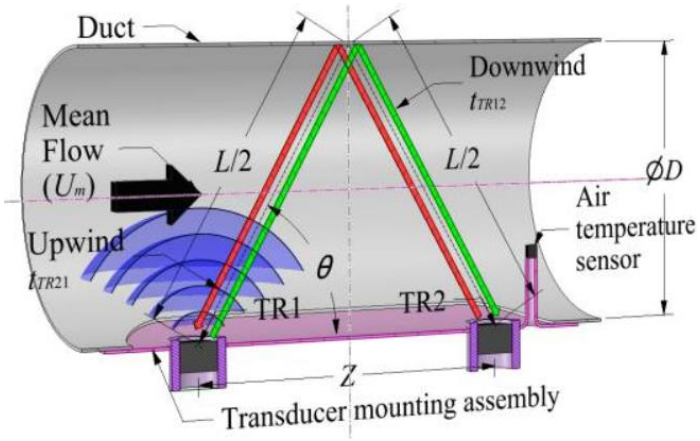
Gaaside vooluhulga mõõtmiseks on võimalik kasutada ultraheli andureid. Üldjuhul kasutatakse antud andureid torujuhtmes oleva vedeliku vooluhulga mõõtmiseks. Leidub ka erinevaid kommertsprodukte, millega mõõdetakse tööstustes heitgaase, kuid nende kõrge hind piirab nende kasutamist. [2]

Ultraheli anduriga gaasi vooluhulga mõõtmine töötab põhimõttel, et helilained liiguvad vooluhulgaga samasuunalisel liikumisel kiiremini ning vastandsuunalisel liikumisel aeglasemalt. Joonis 1.2 illustreerib ühte vooluhulga mõõtmisviisidest ultraheliandurite ga. Andurid on paigutatud üksteise suhtes toru vastandpooltele kindla nurga all. Arvestades toru ja andurite geomeetrilisi parameetreid on võimalik arvutada andurite vaheline kaugus ning selle abil võrrelda kui kaua aega kulub andurilt TR1 andurile TR2 helilainete saatmine ning vastupidi. Antud joonisel jõuaksid andurilt TR1 helilained TR2 andurisse kiiremini kui vastupidiselt, kuna vooluhulga suund U_m on helilainetega samasuunalised. [2]



Joonis 1.2 Ultraheli andurite abil vooluhulga mõõtmine [2]

Teine võimalus on illustreeritud Joonis 1.3. Paigutades ultraheli andurid toru samale poolele horisontaalselt. Ultraheli andurite poolt genereeritud helilained peegelduvad vastas pinnalt tagasi kõrval olevasse andurisse. Selline andurite paigutamisviis on optimaalsem, kuna erinevate juhtmete ühendamine ja liidestamine muutub lihtsamaks, kuid geomeetria ja muude parameetrite arvutamine muutub keerulisemaks. [2]



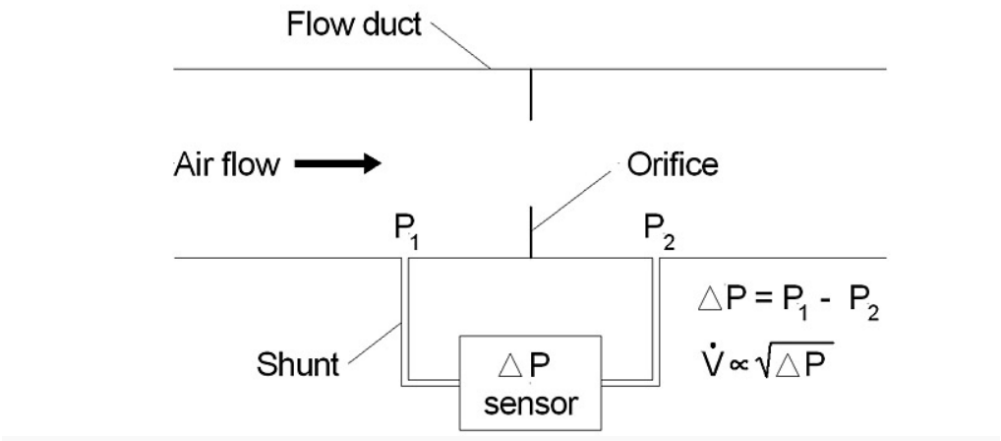
Joonis 1.3 Ultraheli andurite abil vooluhulga mõõtmine V-kujul [2]

Tabel 1.1 Ultraheli anduritega vooluhulga mõõtmise eelise ja puudused [2]

Eelised	**Puudused**
Paindlik paigutus	Hind
Täpsus	Keerukus
Temperatuuri taluvus	Dimensioonid suured
Suur mõõtepiirkond	Nõuab mehhaanilist lisamoodulit (toru), et siduda Titan pesades asuvate digiboksidega.
	Segavad faktorid – heli, müra

1.2 Diferentsiaalrõhu andurid

Diferentsiaalrõhu (*Differential pressure*) printsiip on väga laialtlevinud mõõtmisviis gaaside vooluhulga mõõtmiseks. Diferentsiaalrõhu andurid opereerivad põhimõttel, et rõhu langemine anduri mõõteväljal on võrdne vooluhulga ruutjuurega. Diferentsiaalrõhu mehhaaniline osa hõlmab anduri osa, millega tekitatakse rõhkude erinevus ning elektrooniline osa hõlmab rõhu erinevuste mõõtmist. Diferentsiaalrõhu printsiipi rakendavad paljud tootjad oma vooluhulga mõõtemoodulitel, antud andurid leiavad rakendust kõige enam meditsiinivaldkonnas. Joonis 1.4 on rõhkude erinevus tekitatud Orifice'i kettaga ning mõlema ketta pool olevaid rõhke mõõdetakse rõhuanduriga. [3]



Joonis 1.4 Diferentsiaalrõhusensori põhimõtte diagramm [3]

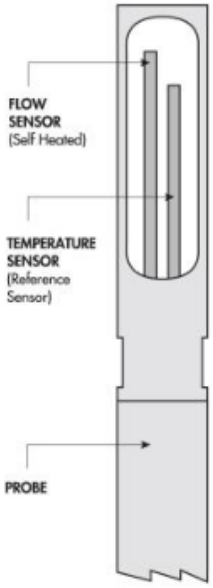
Joonis 1.5 Diferentsiaalrõhusensor [3]

Tabel 1.2 Diferentsiaalrõhusensoriga vooluhulga mõõtmise eelised ja puudused [4]

Eelised	**Puudused**
Kompaktne	Nõuab mehhaanilisi lisamooduleid, et tekitada rõhkude erinevust (näiteks Orifice'i ketas) ja mõõtevälja (toru)
Hind	Täpsus
	Takistab vooluhulga liikumist
	Nõuab head teostust

1.3 Soojuselemendiga andurid

Soojuselemendiga vooluhulga(*Thermal mass flow*) mõõtmisprintsiip on samuti väga populaarne erinevate tootjate poolt. Soojuselemendi mõõtmine baseerub soojusjuhtivuse ideest. Andur koosneb kahest sensorist: esimest sensorit (*Flow sensor*) kuumutatakse sisse integreeritud elektroonika abil ning mõõdetava gaasi liikumisel ümber antud sensori liigub soojus sellest sensorist eemale, toimub sensori jahtumine. Teine sensor (*Temperature sensor*) mõõdab gaasi enda temperatuuri. Esimese sensori jahtumisel vajab element rohkem voolu enda temperatuuri hoidmiseks ning selle ja temperatuuri sensori erinevuste põhjal leitakse vooluhulk, mida mõõdetakse. [5]



Joonis 1.6 Soojuselemendiga anduri tööpõhimõte [5]

Tabel 1.3 Soojuselemendiga vooluhulga mõõtmise eelised ja puudused [5]

Eelised	**Puudused**
Kompaktne	Tundlik niiskusele
Hind	Vajab kalibreerimist mõõdetavale gaasile
Täpsus	

1.4 Anemomeetrid

Hoonete ventilatsiooni testimiseks kasutavad tehnikud üldjuhul anemomeetrit. Anemomeeter on seade, millega on võimalik mõõta gaasi vooluhulka nii kinnises keskkonnas, näiteks ventilatsioonitorudes, kui ka avatud keskkonnas, näiteks tuule kiiruse mõõtmiseks. Anemomeeter on kõige populaarsem vooluhulga mõõtemoodul. Põhilised anemomeetreid tüübid on [6]:

- Tiivikanemomeeter (*Vane anemometer*)
- Termoanemomeeter (*Thermal anemometer*)
- Kaussanemomeeter (*Cup anemometer*)

Tiivikanemomeeter on mehaanilise tiiviku abil vooluhulka mõõtev seade. Tiiviku telg peab olema suunatud vooluhulga liikumisega paralleelselt. Kasutatakse kaevanduste ja hoonete ventilatsioonitorude testimisel. [6]

Joonis 1.7 Tiivikuga anemomeeter [7]

Termosanemomeeter on silindrilise kuumutatud vasetraadi või soojuselemendi abil vooluhulka mõõtev seade. Õhk, mis liigub läbi elemendi, jahutab seda ning kuna elektriline takistus on sõltuv temperatuurist on võimalik saavutada seoseid elemendi elektrilise takistuse ja vooluhulga abil. Termoanemomeetrid on kiire reageerimissagedusega ning seetõttu kasutatakse selliseid mõõtemooduleid üldiselt kohtades, kus vooluhulga kõikumise jälgimine on oluline. [6]

Joonis 1.8 Soojusanemomeeter [8]

Kaussanemomeeter koosneb kolmest või neljast poolkerakujulisest kausist, mis on horisontaalselt kinnitatud telje külge. Vooluhulk, mis liigub mööda antud kausse, paneb need ümber oma vertikaalse telje pöörlema kiirusel, mis on proportsionaalselt selle vooluhulgaga võrdne. [6]

Joonis 1.9 Kaussanemomeeter [9]

Antud bakalaureusetöö ülesandes ei saaks anemomeetrid kasutada nende liiga suurte dimensioonide, hinna (kalibreeritud anemomeeter on kallis) ning puuduva automatiseerimisvõimaluse tõttu, antud mõõteseadet poleks võimalik ühendada Teleplani Titan diagnostikaseadmega.

Tabel 1.4 Anemomeetriga vooluhulga mõõtmise eelised ja puudused [6]

Eelised	**Puudused**
Kalibreeritud seade täpne	Hind
	Automatiseerimisevõimalused puuduvad
	Dimensioonid suured

1.5 Valitud seade

SFM3000-200C on Sensirion'i ettevõtte poolt pakutav andur õhu vooluhulga mõõtmiseks. SFM3000-200C töötab soojuselemendi mõõteprintsiibil. Antud anduriga on võimalik mõõta õhku, hapniku ja teisi mitte-agressiivseid gaase. Andur on võimeline tuvastame gaasi vooluhulga liikumist mõlemas suunas, andes anduri korpuse peale märgitud noolega samas suunas liikuva vooluhulga

korral positiivse väärtuse ning vastupidises negatiivse väärtuse. Mõõtemoodul on õhu suhtes kalibreeritud tootja poolt. [10]

Seade sai valitud selle kompaktse ning paindliku geomeetria pärast. Seda on lihtne siduda kasutaja poolt valmistatud elektroonikaga ning võrreldes teiste turul pakutud anduritega vajab ta kõige vähem mehhaanilisi lisamooduleid, et rakendada antud moodulit bakalaureusetöö ülesande lahendamise raames.

Joonis 1.10 SFM3000-200C [11]

Vastavalt anduri andmelehe leheküljel 3, tuleb anduri reaalse väärtuse kuvamiseks kasutada järgnevat valemit [11]:

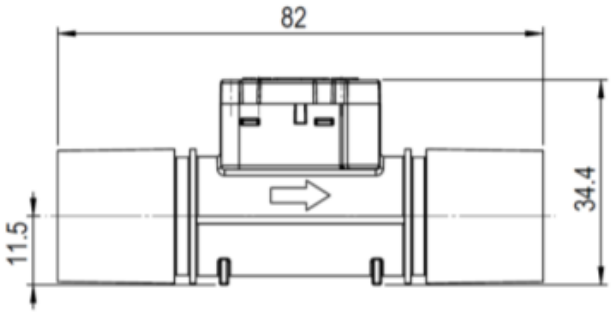
$$flow[slm] = \frac{measured\ value - offset\ flow}{scale\ factor\ flow}$$

Kus: *flow* = mõõdetava vooluhulga väärtus liitrit/minutis

measured value = I2C kaudu mikrokontrolleri saadetud väärtus

offset flow = vooluhulga väärtuse kõrvalekalde konstant: 32 000

scale Factor flow = õhu vooluhulga skaalategur: 140 1/slm



Joonis 1.11 Anduri mõõtmed [11]

Tabel 1.5 SFM3000-200C tehnilised andmed [11]

Parameeter	**Väärtus**
Tööpinge	5 V +/- 5 %
Võimsustarve	< 50 mW
Õhu vooluhulga vahemik	-200 slm – 200 slm
Täpsus	+/- 2.5 slm
Kalibreeritud gaasidele	Õhk, 0_2, N_2
Suhtlusliides	I2C
I2C vaikimis aadress	64 (0x40)
I2C kellasagedus	400 kHZ
Maksimaalne/minimaalne töötemperatuur	+ 80^o C / -20^o C

2. MÕÕTEMOODULI ELEKTROONIKA DISAIN

2.1 Elektroonika

Projekti elektroonikaskeemid ning trükkplaadi disain koostati Altium Designer tarkvaraga. Disaini reeglid, komponentide skeeme ja mudeleid hõlmavad teegid ning kõik muu projektiga seonduv elektroonika osa baseerub Teleplan Estonia alusmalli põhjal. Trükkplaadid telliti Brandner PCB ettevõtte poolt ning seetõttu kasutati trükkplaadi disainimisel antud ettevõtte poolt paika pandud disainireegleid.

2.1.1 Elektroonika põhikomponendid

Joonis 2.1 TM4C1294KCPDTI3R [12]

Mikrokontroller TM4C1294KCPDTI3R- Kogu loogika ja elektroonika juhtimiseks vajalik 128 viiguline aktiivkomponent. Tegemist on Texas Instrument'i poolt pakutava Tiva C seeria mikrokontroller tootega. Antud mikrokontroller on laialt kasutusel Teleplan Estonia ettevõttes ning seetõttu oli nii majanduslikult kui ka ajaliselt mõistlikum valida läbitöötatud lahendus.

Tabel 2.1 TM4C1294KCPDTI3R tehnilised andmed [13]

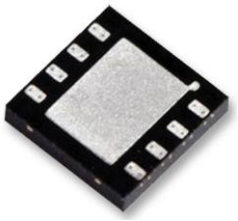
Parameeter	**Väärtus**
Tööpinge	3,3 V alalisvool
Digitaalseid I/O viike	15
Taktsagedus	120 MHz
Flash mälu	512 KB
RAM mälu	256 KB

Joonis 2.2 PCA9507DP, 118 [14]

PCA9507DP,118 on vajalik I2C suhtlusprotokolli jaoks õhuvooluhulga seadme ja trükkplaadil asuva mikrokontrolleri omavaheliseks suhtlemiseks. Õhuvooluhulga seadmest tulev I2C signaal on 5 V pingega, mis läbi antud komponendi redutseeritakse 3,3 V pingega signaaliks kuna mikrokontrolleri viigud on kuni 3,3 V tolerantsed. Komponendi valik baseerus selle kättesaadavuse ning hinna põhjal.

Tabel 2.2 PCA9507DP,118 tehnilised andmed [15]

Parameeter	**Väärtus**
Tööpinge ja väljundpinge	2,7 V – 5,5 V
Maksimaalne väljundvool	50 mA
I2C sisendviigud	2
I2C väljundviigud	2



Joonis 2.3 TPS62175DQCT [16]

TPS62175DQCT on DC/DC pinget alandav impulssmuundur, millega konverteeritakse trükkplaadi 12 V sisendpinge 5 V väljundpingeks. See on vajalik õhu vooluhulga mõõtemooduli tööpinge tagamiseks. Impulssmuunduri valik põhines asjaolul, et antud trükkplaati oleks võimalik ka vajadusel 24 V pingega ühendada ning sellises juhul oleks komponendil tekkiv pingelangust tulenev võimsus olnud liiga suur, et kasutada lineaarregulaatorit.

Tabel 2.3 TPS62175DQCT tehnilised andmed [17]

Parameeter	**Väärtus**
Sisendpinge	4,75 V – 28 V
Väljundpinge	1 V – 6 V
Väljundpinge nimiväärtus	Reguleeritav
Väljundvool	500 mA
Lülitamissagedus	1 MHz

Joonis 2.4 TLV1117LV33DCYR [18]

TLV1117LV33DCYR on fikseeritud väljundpingega 3,3 V DC/DC lineaarregulaator. Antud komponendiga muundatakse TPS62175DQCT tulev 5 V pinge 3,3 V pingeks, mis on vajalik mikrokontrolleri ja RS-485 transiiveri tööpingeks. Kuna mikrokontroller ja RS-485 transiiver ei vaja töötamiseks palju voolu, kasutati impulssregulaatori asemel lineaarregulaatorit selle odava maksumuse tõttu.

Tabel 2.4 TLV1117LV33DCYR tehnilised andmed [19]

Parameeter	**Väärtus**
Tööpinge	2 V – 5,5 V
Väljundpinge	3,3 V
Väljundpinge nimiväärtus	Fikseeritud
Maksimaalne väljundvool	1 A

Joonis 2.5 SN75HVD08DR [20]

SN75HVD08DR on RS-485 standardil baseeruv aktiivkomponent, mille abil toimub suhtlus mikrokontrolleri ja PC vahel. Tegemist on pooldupleks transiiveriga, mis tähendab, et antud komponendiga on võimalik informatsiooni saata ja ka vastu võtta, kuid mitte samal ajal erinevalt dupleksist, mis võimaldab samaaegset saatmist ja vastuvõtmist. Antud transiiveri vastuvõtmist kontrollitakse kahe sisendviiguga.

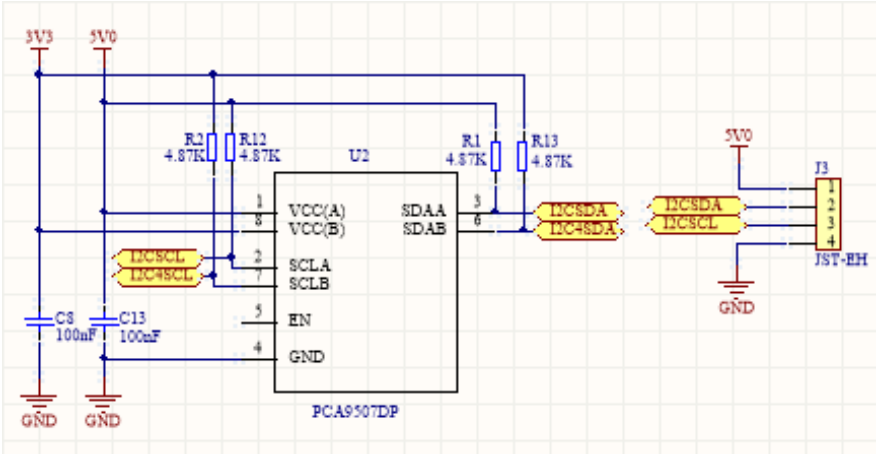
Tabel 2.5 SN75HVD08DR tehnilised andmed [21]

Parameeter	**Väärtus**
Tööpinge	3,0 V – 5,5 V
Sisendvool	Kuni 16 mA
Funktsionaalsus	Transiiver
Suhtlussüsteem	Pooldupleks
Lülitamissagedus	18 ns – 40 ns

2.1.2 Elektroonikaskeemi disainimine

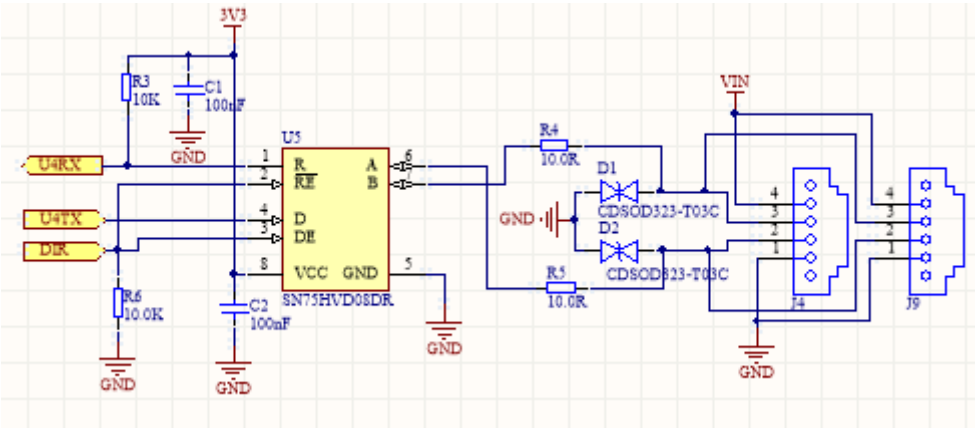
I2C elektriskeem on illustreeritud Joonis 2.6. Õhu vooluhulga seadme sisendpinge, maandus ning I2C siini kella- ja andmeliin on ühendatud trükkplaadile konnektori J3 abil. I2C siini kella- ja andmeliin on ühendatud PCA9507DP komponenti vastavalt SCLA (I2CSCL port) ja SDAA (I2CSCDA port) viikudesse, VCC(A) sisendpinge on võrdne õhu vooluhulga seadme pingega ehk 5 V. Mikrokontrolleri I2C siini kella (I2C4SCL port) - ja andmeliin (I2C4SDA port) on ühendatud PCA9507DP komponenti vastavalt SCLB ja SDAB viikudesse, VCC(B) sisendpinge on võrdne mikrokontrolleri I2C siini ja toitepingega. Kõik I2C siini liinidele on rakendatud *pull-up* 4,87K takistit. PCA9507DP muundab ühendatud I2C siinide pinged vastavalt suunale ehk mikrokontrolleri

väljaminevad ja sissetulevad siinid on alati 3,3 V pingega ning õhu vooluhulga seadme väljaminevad ja sissetulevad siinid on alati 5 V pingega.



Joonis 2.6 I2C elektroonikaskeem. Väljavõte Lisa 1 Elektroonikaskeem.

RS-485 poolduplekstransiiveri elektriskeem on illustreeritud Joonis 2.7. SN75HVD08DR võimaldab suhtlust mikrokontrolleri ja PC vahel, kasutades UART andmesideprotokolli. Mikrokontrolleri UART viigud(U4RX ja U4TX) on ühendatud vastavalt RS-485 transiiveri R ja D viikudesse, samuti on ühendatud ühe digitaalse I/O viiguga DIR liin $\overline{\mathrm{RE}}$ ja $\overline{\mathrm{DE}}$ viikudesse. DIR ühendusega kontrollitakse UARTI suhtlemissuunda ehk kas mikrokontroller saadab või võtab vastu sõnumeid. Kui mikrokontrollerist antakse DIR sisendile kõrge signaalpinge, on transiiver saatmisolekus ning vastupidiselt madala signaalpinge korral on transiiver vastuvõtvas olekus. J4 ja J9 on RJ11 konnektorid, mis on omavahel ühendatud ning nende kaudu tagatakse trükkplaadile toitepinge ning suhtlus PC ja teiste plaatide vahel. J4 konnektorist saab trükkplaat omale toitepinge ning J9 konnektorist antakse toitepinge edasi järgmisele trükkplaadile. Dioodid D1 ja D2 on elektrostaatilise laengu kaitseks.



Joonis 2.7 RS-485 elektroonikaskeem. Väljavõte Lisa 1 Elektroonikaskeem.

12 V toitepinge muundamine 5 V pingeks on illustreeritud Joonis 2.8. TPS62175 muundab trükkplaadile antud toitepinge 5 V pingeks. Antud skeemile on rakendatud ka ülevoolukaitse(F1) ja ülepingekaitse (D3). TPS62175 väljundpinge sõltub pingejaguri R7 ja R8 takistite väärtustest. Komponendi andmelehel välja toodud valemi 5 põhjal on võimalik välja arvutada oma takistite väärtus vastavalt soovitud väljundpingele [17]:

$$R_1 = R_2\left(\frac{V_{OUT}}{V_{REF}} - 1\right)$$

kus: R_1- pingejaguri ülemine takisti

R_2 – pingejaguri alumine takisti

V_{OUT} – soovitud väljundpinge

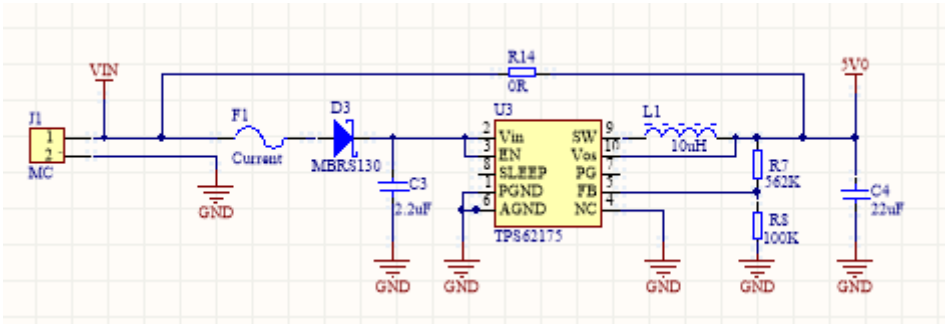
V_{REF}- komponendisisene referentspinge

Võttes R_2 100K Ω, saab arvutada R_1 takisti väärtuse:

$$R_1 = 100\left(\frac{5\,V}{0{,}8\,V} - 1\right) = 525K\,\Omega$$

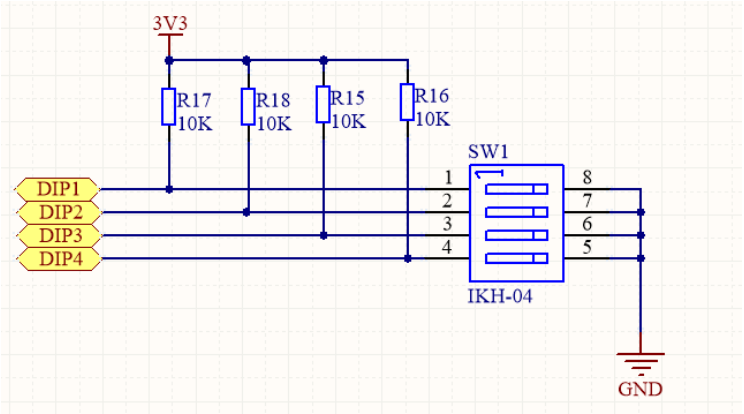
Kuna takistite nominaalväärtustele on tavaliselt juurde antud nende tolerants, mis antud takisti korral on +/- 10%, siis valiti R_1 takisti väärtus suurem kui teoreetiliselt arvutatud takistus.

Induktori L1, nimiväärtusega 10 uH, valikul lähtuti vastavalt TPS61275 andmelehel väljatoodud induktorite valiku tabelile 3. [17]



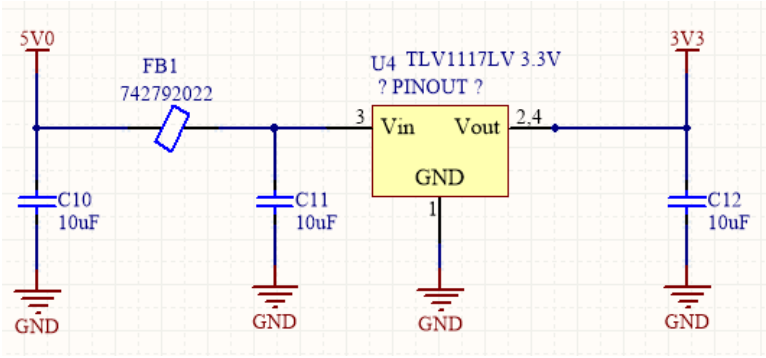
Joonis 2.8 Toitepinge muundamine 5 V pingeks. Väljavõte Lisa 1 Elektroonikaskeem.

DIP lüliti elektriskeem on illustreeritud Joonis 2.9. DIP lülitit kasutatakse igale plaadile unikaalse aadressi andmiseks. Vastavalt lõputöö teemas paika pandud nõuetele peab trükkplaate korraga ühe PC taga kokku tulema 16 ning Modbus protokolli töötamiseks on vajalik et igal „slave'l" oleks oma aadress. DIP lüliti viigud on ühendatud mikrokontrollerisse DIP1, DIP2, DIP3 ja DIP4 liinide kaudu mikrokontrolleri digitaalsetesse I/O viikudesse. Operaatoril on võimalik füüsiliselt iga trükkplaadi aadressi muuta vastavalt vajadusele. Kui DIP lüliti üks lüliti on aktiveeritud annab see mikrokontrollerile sisendi väärtusega 0 ning väljalülitatud lülitid annavad mikrokontrollerile sisendi väärtusega 1. Antud DIP lülitil on 4 lüliti, mis tähendab, et aadresside väärtus võib olla vahemikus 0-16. Aadresside väärtus määratakse kahendarvuna mikrolülitite kombinatsiooniga.



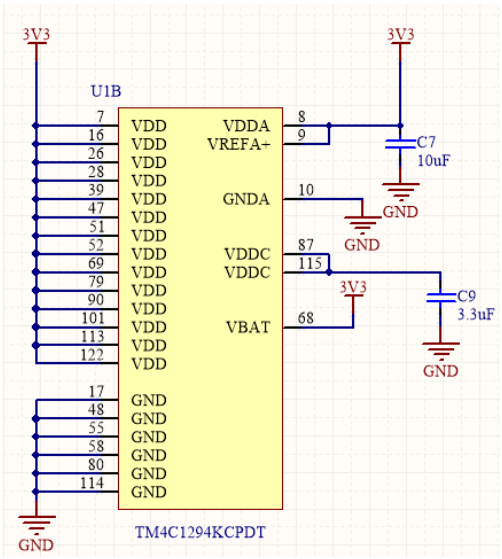
Joonis 2.9 DIP lüliti elektriskeem. Väljavõte Lisa 1 Elektroonikaskeem.

3,3 V toitepinge saamine on illustreeritud Joonis 2.10. Elektriskeemis kasutatud kondensaatorid ja ferriit filter on kasutatud häirete summutamiseks ning komponendi sisend – ja väljundpinge silumiseks. 3,3 V pinge on vajalik mikrokontrolleri ja RS-485 transiiveri toitepingeks.



Joonis 2.10 5 V pinge muundamine 3,3 V pingeks. Väljavõte Lisa 1 Elektroonikaskeem.

Mikrokontrolleri toitepinge elektriskeem on illustreeritud Joonis 2.11. Mikrokontroller TM4C1294KCPDT toimimiseks vajalik toitepinge on 3,3 V, mis ühendatakse VBAT, VDDA, VREFA- ja VDD viikudesse ning maandus vastavalt GND viikudesse.

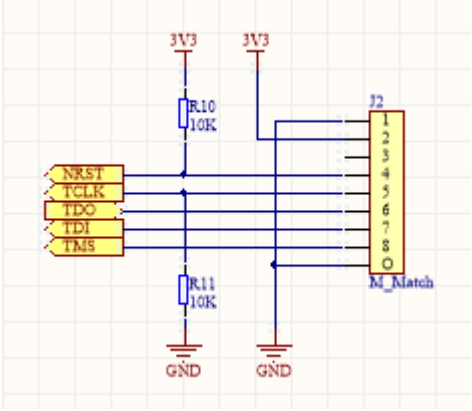


Joonis 2.11 Mikrokontrolleri toitepinge elektriskeem. Väljavõte Lisa 1 Elektroonikaskeem.

JTAG standardit kasutati algselt komponendi või ühenduse vigade tuvastamiseks. Selle abil on võimalik jälgida registreid ja andmete teekonda komponendi siseselt, kasutades 4 viigulist jadaühendust. Tänapäeval on JTAG enamkasutusel mikrokontrolleritele koodi kirjutamisel FLASH ja EEPROM mäludesse ning koodisilurina (*debugger)*. Koodisiluriga on võimalik reaalajas programmikoodi valitud kohas peatada ning selle abil tuvastada tarkvara või riistvara rikkeid. [22]

JTAG ühendus programmaatoriga on illustreeritud Joonis 2.12. J2 konnektorisse ühendatakse ARM-USB-OCD-H programmaator, millega võimaldatakse tarkvara üleslaadimist mikrokontrolleril flash mälusse PC-st. TCLK (*clock line*), TMS (*mode-select line*) ühendatakse mitme seadme korral ühe

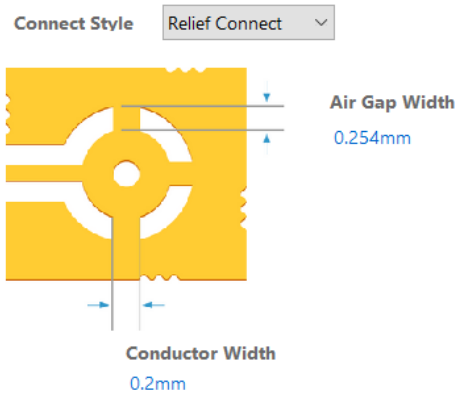
siiniga nendesse seadmetesse ning nende liinidega määratakse kuidas andmed kogu siinis liiguvad., TDI (*data in*) ja TDO (*data out*) on andmeid edastavad liinid. NRST (*reset*) viigu abil on kogu protsessi võimalik taaskäivitada, NRST viik ei ole JTAG siinis kohustuslik. JTAG ühendatakse arvutisse USB konnektoriga. [22]



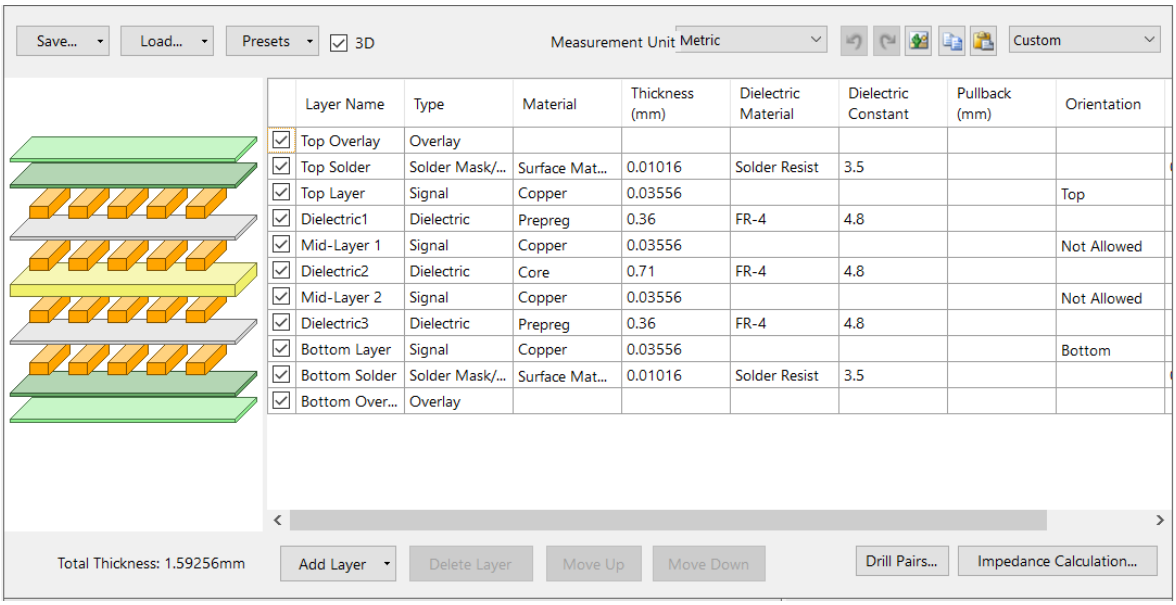
Joonis 2.12 JTAG elektriskeem. Väljavõte Lisa 1 Elektroonikaskeem.

2.1.3 Trükkplaadi disain ja koostamine

Trükkplaat koosneb 4-st kihist – Ülemine kiht (*Top Layer)*, alumine kiht *(Bottom Layer)* ja 2 keskmist kihti *(Mid-Layer 1 ja Mid-Layer 2).* Esimene keskmine kiht kaetakse vasega, kasutades Altium Designer tööriista *Polygon Pour*, mida kasutatakse maanduse *(GND)* kihina. Teine keskmine kiht kaetakse sama moodi vasega ning seda kasutatakse 3,3 V pinge kihina. Selline 4-kihiline trükkplaadi disain elimineerib liigsete radade arvu tekkimist kuna maanduse ja 3,3 V pinge saamiseks on nüüd võimalik kasutada VIA-sid ühenduse saamiseks. VIA on trükkplaati läbiv auk, millega on võimalik tekitada elektriline ühendus erinevate plaadikihtide vahel. VIA ja kihi vahelise ühenduse jaoks on Altium Designeris kindlad reeglid, mida saab vastavalt vajadusele modifitseerida. Joonis 2.13 illustreerib VIA ühendamisstiili kihtide vahel, antud ühendamiseviisi eesmärk on jätta vasega katmata ala VIA ühenduse vahele ühendatava kihiga, sellise stiili eesmärk on teha plaati läbivate komponentide jootmine lihtsamaks, kuna ilma vahet jätmata hakkaks VIA kaudu soojus kihtidesse eralduma, mis takistaks tinajoodise valgumist trükkplaadile. Ülemisse ja alumisse kihti on paigutatud komponendid ning nende signaali – ja toitepinge rajad. Trükkplaadi kihtide kogupaksus on ümardatult 1,6 mm. Kihtide parameetrid on kirjeldatud Joonis 2.14

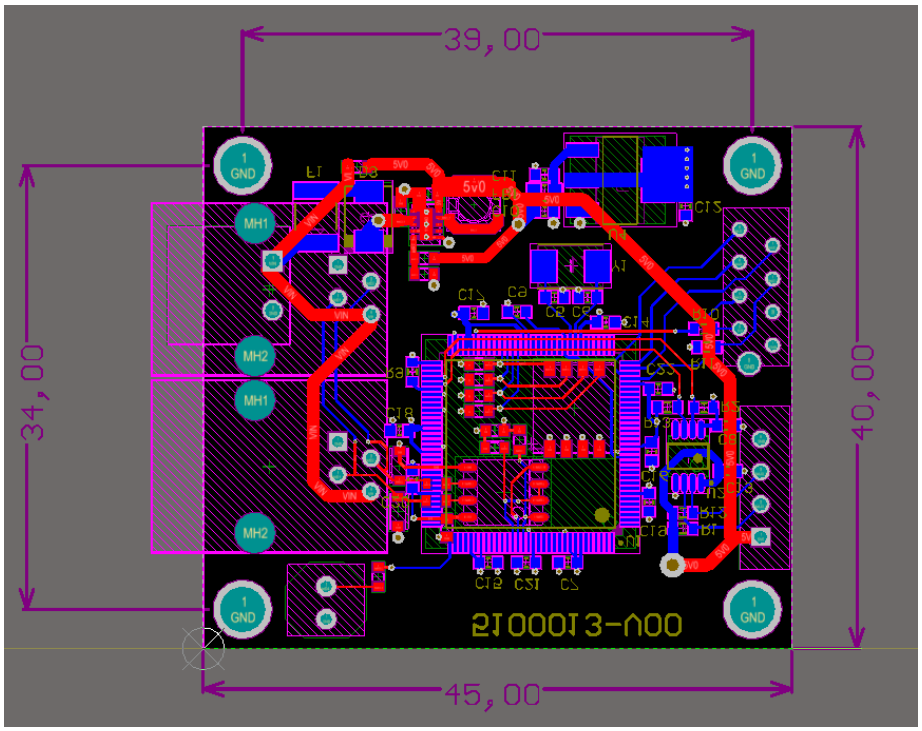


Joonis 2.13 VIA kihi ühendamise reegel

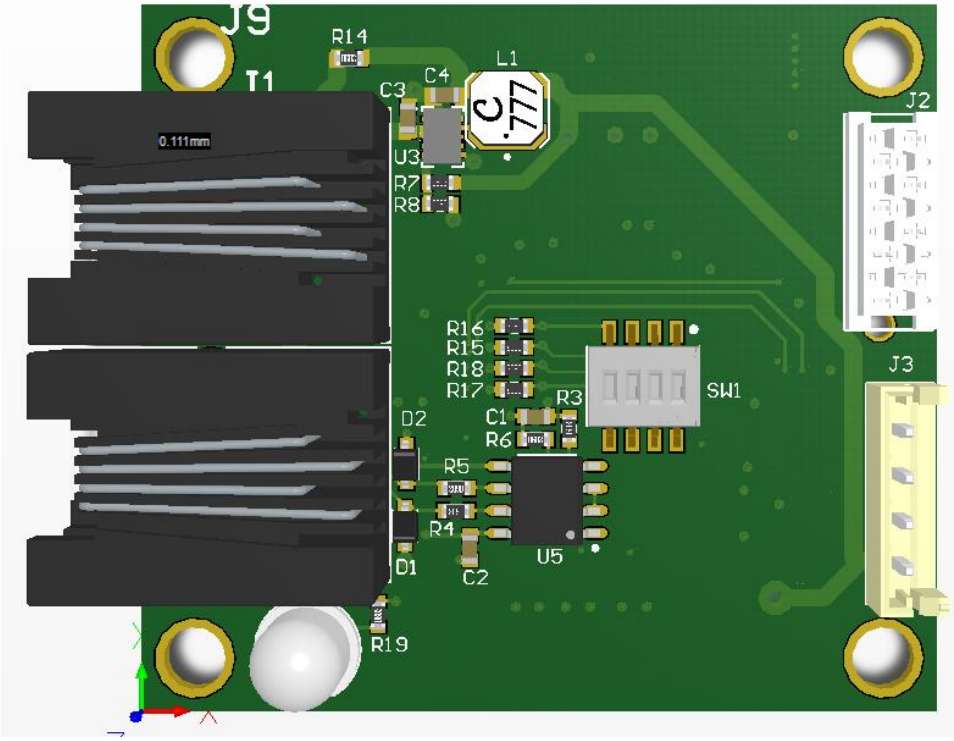


Joonis 2.14 Trükkplaadi kihid

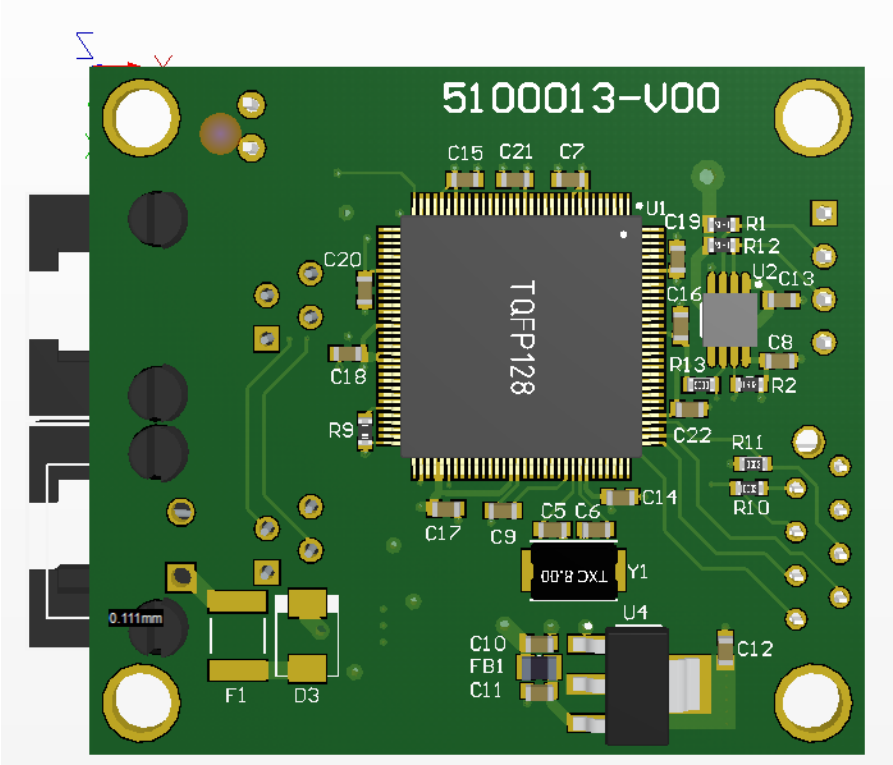
Trükkplaadi suurusel lähtuti põhimõttest, et trükkplaat tuleks mõõtmetel võimalikult väike. Lõplikud mõõtmed trükkplaadile defineeriti pärast kõikide komponentide ja nende radade paigutamist. Lõplikud trükkplaadi mõõtmed on 45 mm x 40 mm. Trükkplaadile on igasse nurka tekitatud augud, mille abil on võimalik trükkplaat kinnitada korpuse külge. Trükkplaadi 2D disaini ja vajalikke mõõtmeid illustreerib Joonis 2.15.



Joonis 2.15 Trükkplaadi 2D vaade ja selle mõõtmed

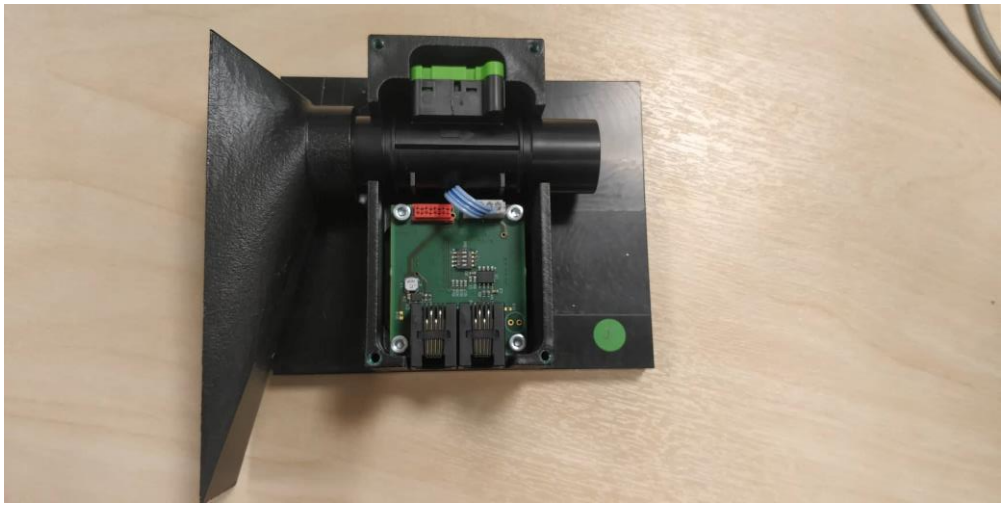


Joonis 2.16 Trükkplaadi 3D pealtvaade



Joonis 2.17 Trükkplaadi 3D altvaade

Vastavalt trükkplaadi dimensioonidele arendati Teleplan Estonia mehhaanika meeskonna poolt välja mõõtemooduli mehaaniline korpus, mis koosneb anduri otsa paigutatud lehtrist ning trükkplaati katvast korpusest. Antud korpuse mehaaniline joonised välja toodud Lisa 6 Mõõtemooduli korpuse mehaaniline joonis.

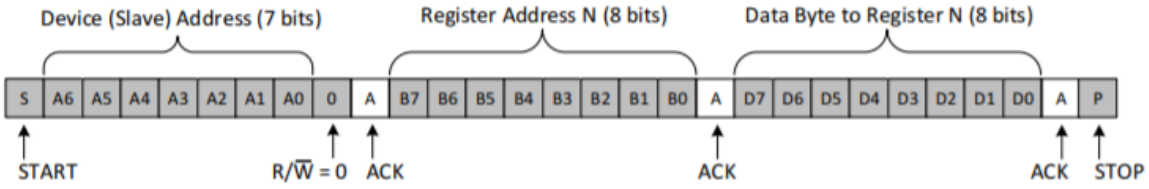


Joonis 2.18 Mehaaniline korpus koos seadmega

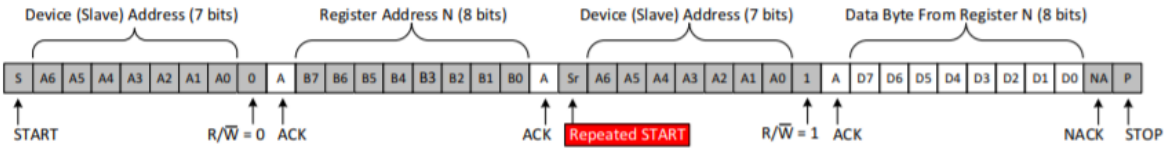
3. MÕÕTEMOODULI TARKVARA

3.1 Mõõteandmete edastus mõõtemoodulist mikrokontrollerisse

Suhtlus õhu vooluhulga mõõtemooduli ja mikrokontrolleri vahel toimub läbi I2C siini. . I2C on ülemseade/alamseade tüüpi protokoll, millel võib iga siinil olev seade olla ülemseade, kui eelmine ülemseade on STOP bittiga oma kontrolli ära andnud. I2C siinil on 2 liini: andme (*Data*) – ja kellaliin (*Clock*), mõlemad liinid on ühendatud kõikidesse alamseadmetesse ning mõlemale liinile peavad olema rakendatud *pull-up* takistid. I2C siin on pooldupleks, mis tähendab, et andmed saavad korraga liikuda ainult ühes suunas. Joonis 3.1 on kirjeldatud I2C kirjutamine alamseadmele ülemseadme pool. Andmeliini kaudu saadetakse alamseadme aadress (7 bitti) ja saatmise suund (1 bitt). Seejärel saadab alamseade ACK biti, et kinnitada käsu vastu võtmist, millele järgnevad üks või enam andmebaiti ülemseadmelt alamseadmele või vastupidiselt, olenevalt ülemseadme poolt täpsustatud suunast. Kogu protsess on lõpetatud kui ülemseade on saatnud STOP bitti pärast viimast andmebaidi edastamist. START ja STOP käsku saab initsialiseerida kui I2C kellaliin on madalaks tõmmatud. [22]



Joonis 3.1 I2C kirjutamine alamseadmele [23]



Joonis 3.2 I2C lugemine alamseadmelt [23]

3.2 Mõõteandmete edastus mikrokontrollerist diagnostikasedmesse

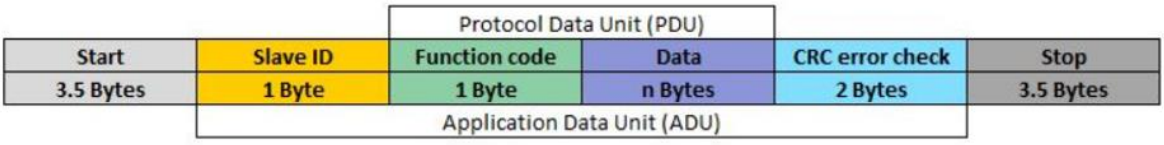
Suhtlus diagnostikaseadme Titan ja mikrokontrolleri vahel toimub läbi Modbus protokolli. Modbus on päring-vastus tüüpi andmesideprotokoll klient-server, ülemseade-alamseade MDB-siinil (*multidrop bus*). Modbus on võimeline töötama RS-232 ja RS-485 standartidel, millest viimane on

rohkem levinud suuremate vahemaade ja kiiruste tõttu. Modbus võrgus on üldjuhul 1 ülemseade ning näiteks ühe baidise adresseerimisvälja puhul kuni 247 alamseadet. Igal alamseadmel peab olema unikaalne aadress, millele ülemseadme päringu korral peab vastava aadressiga alamseade vastama. Andmevahetuse initsialiseerijaks on alati ülemseade. Tuntumad Modbus protokolli versioonid on ASCII, TCP ja RTU (*Remote Terminal Unit*), millest viimast käsitletakse koos RS-485 standardiga ka antud bakalaureusetöö raames. [24]

Modbus RTU protokollis käsitletakse sõnumeid kodeeritud kahendkoodis, sõnumite veakontroll teostatakse CRC (*Cyclycal Redundancy Check*) abil. Iga kaadri lõpus ja alguses on 3,5 sümboli pikkune paus ning kaadri maksimaalne suurus on 256 baiti. [24]

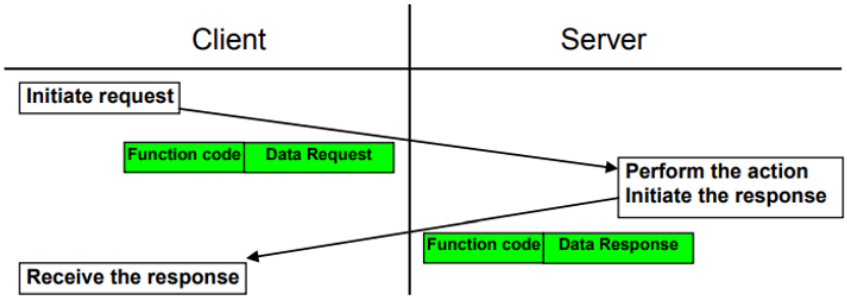
Modbus RTU protokolli sõnum koosneb 4-st väljast [24]:

- Adresseeritava alamseadme aadress (*slave ID*)
- Käsu - ja sõnumi tüübi määrava funktsiooni kood (*Function code*)
- Andmed (*Data*)
- Veakontroll (*CRC error check*)

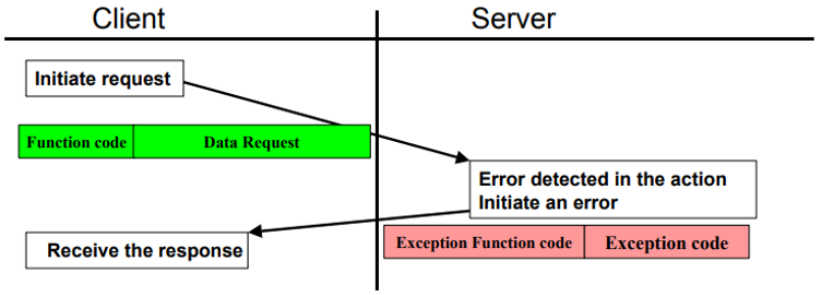


Joonis 3.3 Modbus RTU sõnumiraamistik [24]

Vigadeta päring/vastuse korral saadab alamseade ülemseadmele tagasi päringuga sama funktsiooni koodiga sõnumi, millele on lisatud alamseadmelt nõutud andmed. Lubatud funktsioonikoodid on vahemikus 1-256, kus vahemikus 128-255 on reserveeritud veateadete edastamiseks. Veateatega päring/vastuse korral saadab alamseade samuti päringuga sama funktsiooni koodiga sõnumi, kuid funktsioonikoodi MSB (*Most significant bit*) on loogiline „1“ ning lisab sõnumile vastava veateate. [24]



Joonis 3.4 Vigadeta päring/vastus [24]



Joonis 3.5 Veateatega päring/vastus [24]

Tabel 3.1 Põhilised Modbus veateated [24]

Kood	**Tekst**	**Detailid**
1	*Illegal Function*	Vastuvõetud funktsioonikood on tundmatu või keelatud alamseadme jaoks.
2	*Illegal Data Address*	Küsitud andmeregister on keelatud või ei eksisteeri.
3	*Illegal Data Value*	Väärtus mitte aktsepteeritud alamseadme poolt.
4	*Slave Device Failure*	Alamseadme on tuvastanud rikke käsu täitmisel.
5	*Acknowledge*	Alamseade on päringu vastu võtnud ning töötleb seda, kuid vajab selleks rohkem aega. Antud vastus tagab rikke välistamist ülemseadmes
6	*Slave Device Busy*	Alamseade on hõivatud. Ülemseade peaks hiljem uuesti proovima.
7	*Negative Acknowledge*	Alamseade ei suuda programmi käsku läbi viia. Ülemseade peaks nõudma diagnoosi või veakoodi alamseadmelt.

Modbus funktsiooni koodid jagatakse kolme klassi [24]:

1. Standardiseeritud käsud
2. Kasutajakäsud
3. Reserveeritud käsud

Standardiseeritud käsud on käsud, mis on mõeldud avalikuks kasutamiseks. Sellistel käskudel on olemas vastav dokumentatsioon, kus kirjeldatakse käskude sisu. Põhilisi standardiseeritud käske iseloomustab Tabel 3.2. [24]

Tabel 3.2 Põhilised standardiseeritud käsud [24]

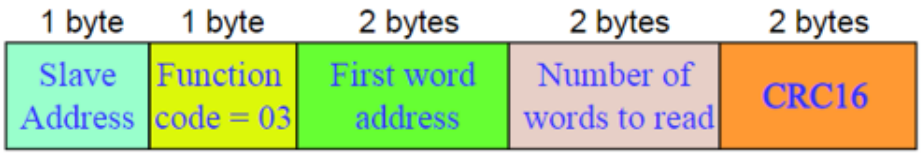
Käsk	**Funktsioonikood**
Loe väljundi väärtus	01
Loe sisendit	02
Loe hoidvaid registreid	03
Loe sisendregisters	04
Kirjuta väljundi väärtus	05
Kirjuta registrisse	06
Kirjuta mitu väljundit	15
Kirjuta mitmesse registrisse	16

Kasutajakäsud on spetsiifilised käsud, mida kasutaja saab luua oma seadme funktsioneerimiseks. [24]

Reserveeritud käsud on käsud, mis ei ole mõeldud avalikuks kasutamiseks. Sellised käsud on kasutusel vanematel seadmetele. [24]

Vastavalt õhu vooluhulga mõõtemooduli spetsiifikale kasutatakse Modbusi standardkäsku 0x03. Ülemseade saadab alamseadmele päringu, mis sisaldab järgnevaid väljasid [24]:

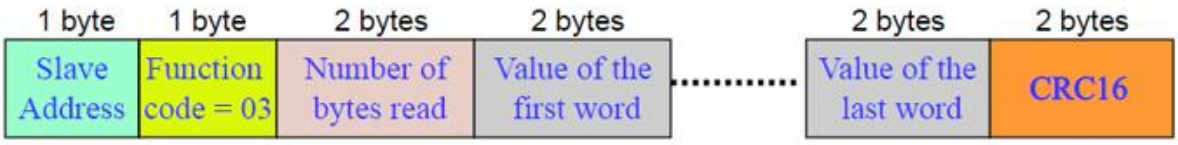
- Alamseadme aadress (*Slave adress*)
- Funktsiooni kood (*Function code*)
- Küsitavaid andmeid sisaldavate registrite aadress (*First word address*)
- Registrite arv (*Number of words to read*)
- Veakontroll (*CRC16*)



Joonis 3.6 Ülemseadme päring 0x03 käsu korral [24]

Alamseade saadab ülemseadme päringule vastuse, mis sisaldab järgnevaid väljasid [24]:

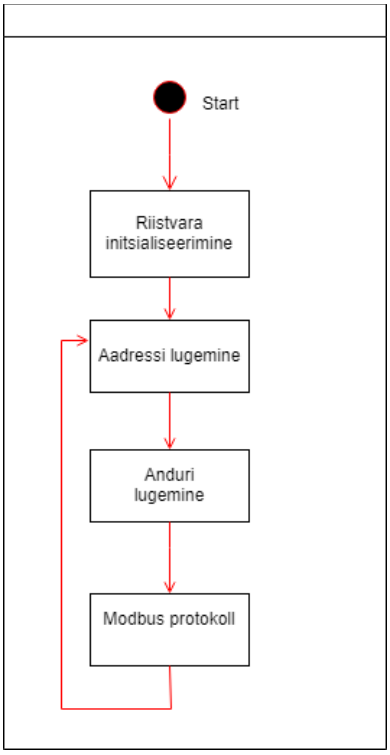
- Alamseadme aadress (*Slave adress*)
- Funktsiooni kood (*Function code*)
- Baitide arv (*Number of bytes to read*)
- Esimese registri sisu (*Value of the first word)*
- Viimase registri sisu (*Value of the last word*)
- Veakontroll (*CRC16*)



Joonis 3.7 Alamseadme vastus ülemseadme päringule 0x03 käsu korral [24]

3.3 Mõõtemooduli tarkvara struktuur

Joonis 3.8 kirjeldab mõõtemooduli tarkvara põhiprogrammi töötsüklit.



Joonis 3.8 Mõõtemooduli tarkvara lihtsustatud töödiagramm

Riistvara initsialiseerimine tähendab kõikide mikrokontrolleril kasutavate digitaalsete I/O viikude, perifeeria, UART ja Modbus protokolli töötamiseks vajalike kontrollerite ning mõõtemooduli suhtlemiseks vajaliku I2C protokolli tarkvaralist initsialiseerimist

Aadressi lugemine tähendab DIP-lüliti baasil etteantud alamaaadressi väärtuse lugemist ja salvestamist 4 digitaalse I/O viigu põhjal. Antud tarkvaralist lahendust kirjeldab Joonis 3.9 funktsioon ModBusSlave().

```
void ModbusSlave()
{
    uint8_t dip1 = 0;
    uint8_t dip2 = 0;
    uint8_t dip3 = 0;
    uint8_t dip4 = 0;
    uint8_t address = 0;
    dip4 = GPIOPinRead(GPIO_PORTM_BASE,GPIO_PIN_0) && GPIO_PIN_0;
    dip3 = GPIOPinRead(GPIO_PORTM_BASE,GPIO_PIN_1) && GPIO_PIN_1;
    dip2 = GPIOPinRead(GPIO_PORTM_BASE,GPIO_PIN_2) && GPIO_PIN_2;
    dip1 = GPIOPinRead(GPIO_PORTM_BASE,GPIO_PIN_3) && GPIO_PIN_3;
    address = ((dip1) + (dip2 << 1) + (dip3 << 2) + (dip4 << 3));
    if ((address >= 0) && (address <= 16)) {
        _slave = address;
    }
}
```

Joonis 3.9 DIP lüliti aadressi lugemine tarkvaraliselt. Väljavõtte Lisa 3 Modbus protokoll C-Programmeerimiskeeles.

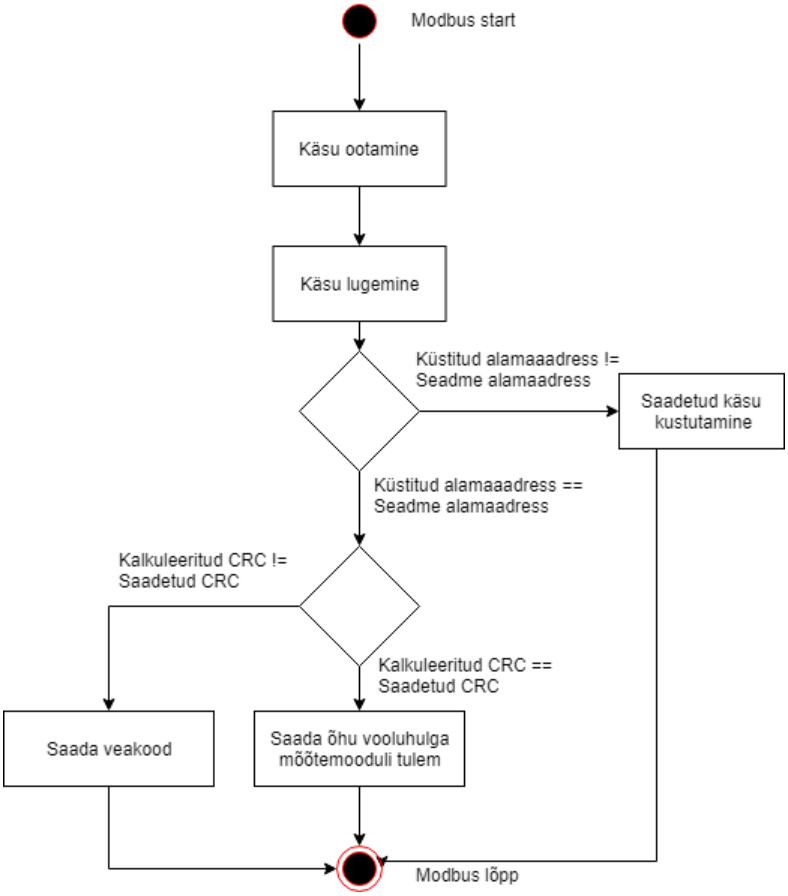
Anduri lugemine tähendab I2C protokolli kaudu õhu vooluhulga mõõtemoodulilt lugemi salvestamist mälusse. Antud info konverteeritakse tarkvaraliselt SLM ühikule Joonis 3.10 kirjeldatud funktsioonis vFlow().

```
static void vFlow()
{
        i32IntegerPart = 0;
        i32FractionPart = 0;
        flow = 0;
        flowread(&flow);

        flow = ((flow & 0x00FF00) << 0) | ((flow & 0xFF0000) >> 16); //reverse bytes
        flow = (flow-offset)*2000/scale + 1;
        flow = flow/2;
        i32IntegerPart = flow/1000;
        tab_reg[0] = i32IntegerPart;
        i32FractionPart = flow;
        i32FractionPart = i32FractionPart - (i32IntegerPart * 1000);
        if(i32FractionPart < 0)
        {
            i32FractionPart *= -1;
        }

        tab_reg[1] = i32FractionPart;

}
```

Joonis 3.10 Õhu vooluhulga mõõtemooduli lugemine tarkvaraliselt. Väljavõte Lisa 2 Põhiprogramm C-programmeerimiskeeles.

Modbus protokoll on lihtsustatud kujul illustreeritud Joonis 3.11. Programm seisab nii kaua käsu ootamise seisus kuni ülema küsitakse infot, alam loeb käsu läbi ning otsustab kas see on talle adresseeritud või mitte. Kui loetud käsu lugemisega selgub, et aadress, millelt küsitakse, klapib

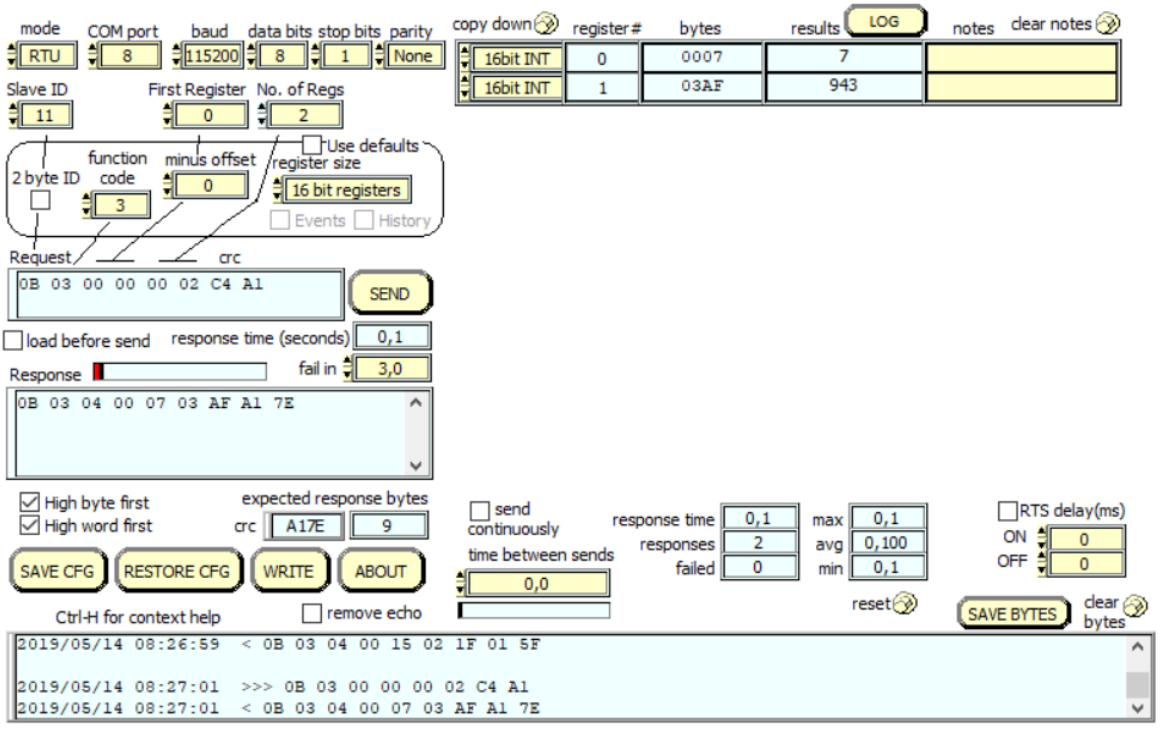
alama aadressiga, arvutatakse käsu CRC ja võrreldakse saadetud CRC-ga, vastavalt tulemusele saadab alam oma vastuse. Juhul kui käsu lugemise käigus selgub, et käsk ei ole alamale adresseeritud, kustutatakse käsk ning oodatakse uut käsku. Reaalne programmikood on väljatoodud Lisa 3 Modbus protokoll C-Programmeerimiskeeles [25].



Joonis 3.11 Modbus algoritm

4. MÕÕTEMOODULI TARKVARA TESTIMINE

Ülema (PC) ja alama (mikrokontroller) omavahelist suhtlemist katsetati „Simply Modbus“ tarkavaraga. Antud tarkvara simuleerib Modbus protokollis ülema *(Master)* rolli. Tarkvaras tuleb eelnevalt ära defineerida Modbus protokolli tüüp (*mode*), alamaadress (*slave ID*), edastuskiirus (*baudrate*), registrite arv (*No. of Regs*), mida tahetakse lugetakse, funktsioonikood (*function code*) ja kasutatava COM porti number (*COM port*). Joonis 4.1 paremal ülevas servas on võimalik näha saadetud vastust. Antud näite korral salvestatakse mõõdemooduli täisarvuline väärtus registrisse „0“ ja murdosa registrisse „1“.



Joonis 4.1 Simply Modbus tarkvara

Diagnostikaseadme Titan PC-sse rakendati Python-programmeerimiskeeles kirjutatud tarkvara kood. Programm kasutab „minimalmodbus“ Pythoni tarkvara teeke ning küsib vastavalt koodis defineeritud alamaadressitele lugemit. Antud kood on väljatoodud Lisa 5 Modbus master Python-programmeerimiskeeles. Joonis 4.2 illustreerib olukorda, kus 5 korda küsitakse aadressiga „2“ alamseadme vastust ning seejärel 5 korda aadressiga „11“ alamseadme vastust. Vastused koos küsitud aadressiga prinditakse ekraanile.

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bi
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\home\AppData\Local\Programs\Python\Python37\airflowtest1.py
address= 2
0.028
address= 2
2.657
address= 2
2.714
address= 2
2.629
address= 2
2.686
address= 2
2.943
address= 11
3.343
address= 11
3.543
address= 11
3.457
address= 11
3.429
address= 11
3.400
>>>
```

Joonis 4.2 Kahe mõõtemooduli vastused Python programmeerimiskeeles

KOKKUVÕTE

Lõputöö loodi kuna Teleplan Estonia OÜ-le tuli kliendi poolt soov testida oma digiboksi ventilaatorite korrasolekut ning ettevõttes olevatel diagnostikaseadmetele puudus funktsioon selle testimiseks. Selle jaoks oli tarvilik luua mõõtemoodul, mille abil saab mõõta ventilaatoritest tulevat vooluhulka ning liidestada see juba kasutusel oleva diagnostikaseadme Titan'iga. Antud diagnostikaseadmes on 16 pesa, millest igasse on vaja mõõtemoodulit. Selle jaoks oli vajalik valida vastav mõõteseade ning antud seadmele luua trükkplaat ning programmeerida vastav tarkvara.

Mõõteseadme SFM3000-200C valik baseerus sellele sobiva geomeetria, lihtsa liidestuse ning asjaolul, et antud seade oli kalibreeritud õhu suhtes. Mõõteseadme suhtlus mikrokontrolleriga toimus I2C andmesideprotokolli vahendusel.

Mõõteseadme valikule järgnevalt disainiti elektriskeem ja trükkplaat. Elektriskeemi jaoks valiti vastavalt hinnale ja vajadusele elektroonikakomponendid. Mõõteseade saadab mõõdetud nimiväärtused mikrokontrollerile, mis edastab antud väärtused Modbus andmesideprotokolli vahendusel need edasi diagnostikaseadme PC-le. Modbus andmesideprotokolli jaoks oli tarvilik trükkplaadile paigutada ka füüsiline mikrolüliti, selleks, et anda igale diagnostikaseadme pesades asetseva mõõtemoodulile unikaalne aadress, mille põhjal PC saata päringu mõõtemoodulile tulemi saatmiseks.

Koostatud riistvara, Modbus ja I2C andmesideprotokolli juhtimiseks loodi C programmeerimiskeeles programmikood. Programmikoodi testiti algselt „Simply Modbus" tarkvaraga ning hiljem diagnostikaseadme liidestamiseks mõõtemooduliga loodi Pythoni programmeerimiskeeles programmikood Modbus andmesideprotokolli jaoks.

Katsetatud tulemuste põhjal võib väita, et lõputöö raames koostatud süsteem on funktsioneeriv ning võib rakendada eelnevalt mainitud diagnostikaseadmesse püsivalt. Tarkvara katsetamise käigus tekkis algselt probleeme mõõtemoodulilt diagnostikaseadme PC-sse mõõdetud tulemuse saatmisega, kui antud PC-sse oli ühendatud korraga 16 seadet, kuid antud probleem sai tarkvaraliselt lahendatud.

Antud lahenduse edasiarenduse võimaluseks nähakse kalibreerimistabeli loomist. Kuigi valitud mõõteseade on selle andmelehe põhjal tootja poolt kalibreeritud õhu vooluhulga mõõtmiseks, ei pruugi antud väide tõene olla. Seetõttu tuleks valitud mõõteseadmed kindla etalon õhuvooluhulgaga üle kontrollida ning erinevuste korral koostada tarkvaraliselt kalibreerimistabel, mille põhjal redigeeritakse mikrokontrollerist saadetav õhu vooluhulga nimiväärtus täpsemaks.

SUMMARY

The thesis was created because Teleplan Estonia OÜ had the customer's desire to test the condition of their digibox fans and the diagnostic equipment in the company had no functions to test it. For this, it was necessary to create a measurement module that can measure the flow from fans and interfere with the already existing diagnostic device Titan. This diagnostic device has 16 sockets, each with a measurement module. For this, it was necessary to select the appropriate measuring device and to create a printed circuit board for this device and to program the corresponding software.

The choice of the SFM3000-200C was based on the geometry, the simple interface and the calibration of the device relative to the air. The communication of the measuring device with the microcontroller took place via the I2C data communication protocol.

Following the selection of the measuring device, the circuit diagram and the circuit board were designed. Electronic components were selected for the electric scheme according to price and need. The measuring device sends the measured nominal values to the microcontroller, which transmits these values to the PC of the diagnostic device via the Modbus data communication protocol. For the Modbus data communication protocol, it was also necessary to place a physical microswitch on the printed circuit board in order to provide a unique address to each of the measuring modules in the diagnostic device slots, on the basis of which the PC sends a request to the measurement module to send the result.

A program code was created in C programming language to control the built-in hardware, Modbus and I2C data protocol. The program code was originally tested with the "Simply Modbus" software, and later, in the Python programming language, a code for the Modbus communication protocol was created to connect the diagnostic device to the measurement module.

Based on the results tested, it can be argued that the system developed in the thesis is functional and can be applied permanently to the aforementioned diagnostic device. During the testing of the software, there were initially problems with sending the measured result from the measuring module to the diagnostic device PC when 16 devices were connected to the PC at one time, but the problem was solved by software.

The possibility of further developing this solution is to create a calibration table. Although the selected measuring device is calibrated by the manufacturer for measuring air flow based on its data sheet, this statement may not be valid. Therefore, the selected measuring equipment should be checked with a specific reference air flow rate and, in case of discrepancies, a software

calibration table should be drawn up on the basis of which the air flow rate sent from the microcontroller is edited more accurately.

KASUTATUD KIRJANDUS

[1] „Teleplan,“ [Võrgumaterjal]. Saadaval: http://www.teleplan.com/manage/wp-content/uploads/Teleplan-Titan-flyer-1.pdf. [Kasutatud 10 Mai 2019].

[2] A. B. Raine, N. Aslam, C. P. Underwood ja S. Danaher, „Development of an Ultrasonic Airflow Measurement Device for Ducted Air,“ [Võrgumaterjal]. Saadaval: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4481905/. [Kasutatud 25 Aprill 2019].

[3] „Applicance Design,“ [Saadaval]. Available: https://www.appliancedesign.com/articles/93801-lba-differential-pressure-sensors. [Kasutatud 27 Aprill 2019].

[4] D. Robo, „Differential Pressure flow measurement – How it works ? Advantages and Disadvantages of DP flow measurement,“ [Võrgumaterjal]. Saadaval: https://automationforum.co/differential-pressure-flow-measurement-how-it-works-advantages-and-disadvantages-of-dp-flow-measurement/. [Kasutatud 1 Mai 2019].

[5] B. Steinberg, „Sage Metering, Inc.,“ [Võrgumaterjal]. Saadaval: https://sagemetering.com/back-to-basics/thermal-mass-flow-meter-working-principle-and-theory/. [Kasutatud 27 Aprill 2019].

[6] „Omega Engineering Inc.,“ [Võrgumaterjal]. Saadaval: https://www.omega.com/en-us/resources/anemometers#section1. [Kasutatud 25 Aprill 2019].

[7] [Võrgumaterjal]. Saadaval: https://www.brannan.co.uk/media/catalog/product/cache/1/image/9df78eab33525d08d6e5fb8d27136e95/a/n/anemometer-and-thermometer-13-460-0.jpg. [Kasutatud 11 Mai 2019].

[8] „Testmeter,“ [Võrgumaterjal]. Saadaval: https://www.testmeter.sg/webshaper/pcm/gallery/lg/244139600fffe7aa7cc956010ef5b3221489987338-lg.jpg. [Kasutatud 11 Mai 2019].

[9] „PCE Instruments,“ [Võrgumaterjal]. Saadaval: https://www.pce-instruments.com/english/measuring-instruments/test-meters/anemometer-pce-instruments-cup-anemometer-pce-a420-det_56311.htm. [Kasutatud 27 Aprill 2019].

[10] „Sensirion,“ [Võrgumaterjal]. Saadaval: https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/0_Datasheets/Differential_Pressure/Sensirion_Differential_Pressure_Sensors_SDP3x_Digital_Datasheet.pdf. [Kasutatud 5 Mai 2019].

[11] „Mouser,“ [Võrgumaterjal]. Saadaval: https://www.mouser.ee/ProductDetail/Sensirion/SFM3000-200C?qs=lAYGPh4Fp%252bzH9WcDkeu8Ww==.

[12] „Mouser,“ [Võrgumaterjal]. Saadaval: https://www.mouser.ee/ProductDetail/Texas-Instruments/TM4C1294KCPDTI3R?qs=sGAEpiMZZMuoKKEcg8mMKOIXatb8SC9JFKfRf%2FvnKvBSs%2Fpnh%252BgigA%3D%3D. [Kasutatud 5 Mai 2019].

[13] „Texas Instruments,“ [Võrgumaterjal]. Saadaval: http://www.ti.com/lit/ds/symlink/tm4c1294kcpdt.pdf. [Kasutatud 2 Mai 2019].

[14] „Mouser Electronics, Inc,“ [Võrgumaterjal]. Saadaval: https://www.mouser.ee/ProductDetail/NXP-Semiconductors/PCA9507DP118?qs=%2Fha2pyFaduivPer%2FvR8rWry%252BIYHm6T5F6qav%2F%252BkCj9cIh%2FMpBfllVg%3D%3D.

[15] „Mouser,“ [Võrgumaterjal]. Saadaval: https://www.mouser.ee/datasheet/2/302/PCA9507-1127511.pdf. [Kasutatud 2 Mai 2019].

[16] „Mouser Electronics, Inc,“ [Võrgumaterjal]. Saadaval: https://www.mouser.ee/ProductDetail/Texas-Instruments/TPS62175DQCT?qs=sGAEpiMZZMtitjHzVIkrqTKPkmFoYJTjks0VtTYi6%2Fs%3D.

[17] „Texas Instruments,“ [Võrgumaterjal]. Saadaval: http://www.ti.com/lit/ds/symlink/tps62177.pdf. [Kasutatud 2 Mai 2019].

[18] „Mouser,“ [Võrgumaterjal]. Saadaval: https://www.mouser.ee/ProductDetail/Texas-Instruments/TLV1117LV33DCYR?qs=sGAEpiMZZMsGz1a6aV8DcEb4aZWauOpD3eEIe4FcZDw%3D. [Kasutatud 5 Mai 2019].

[19] „Texas Instruments,“ [Võrgumaterjal]. Saadaval: http://www.ti.com/lit/ds/symlink/tlv1117lv.pdf. [Kasutatud 2 Mai 2019].

[20] „Mouser,“ [Võrgumaterjal]. Saadaval: https://www.mouser.ee/ProductDetail/Texas-Instruments/SN75HVD10D?qs=sGAEpiMZZMtk5jbcouIbS8mzv14vtZgI5iJVbBo6UJM%3D. [Kasutatud 5 Mai 2019].

[21] „Texas Instruments,“ [Võrgumaterjal]. Saadaval: http://www.ti.com/lit/ds/symlink/sn75hvd08.pdf. [Kasutatud 2 Mai 2019].

[22] P. Horowitz ja W. Hill, The Art of Electronics Third Edition, Cambridge University, 2015.

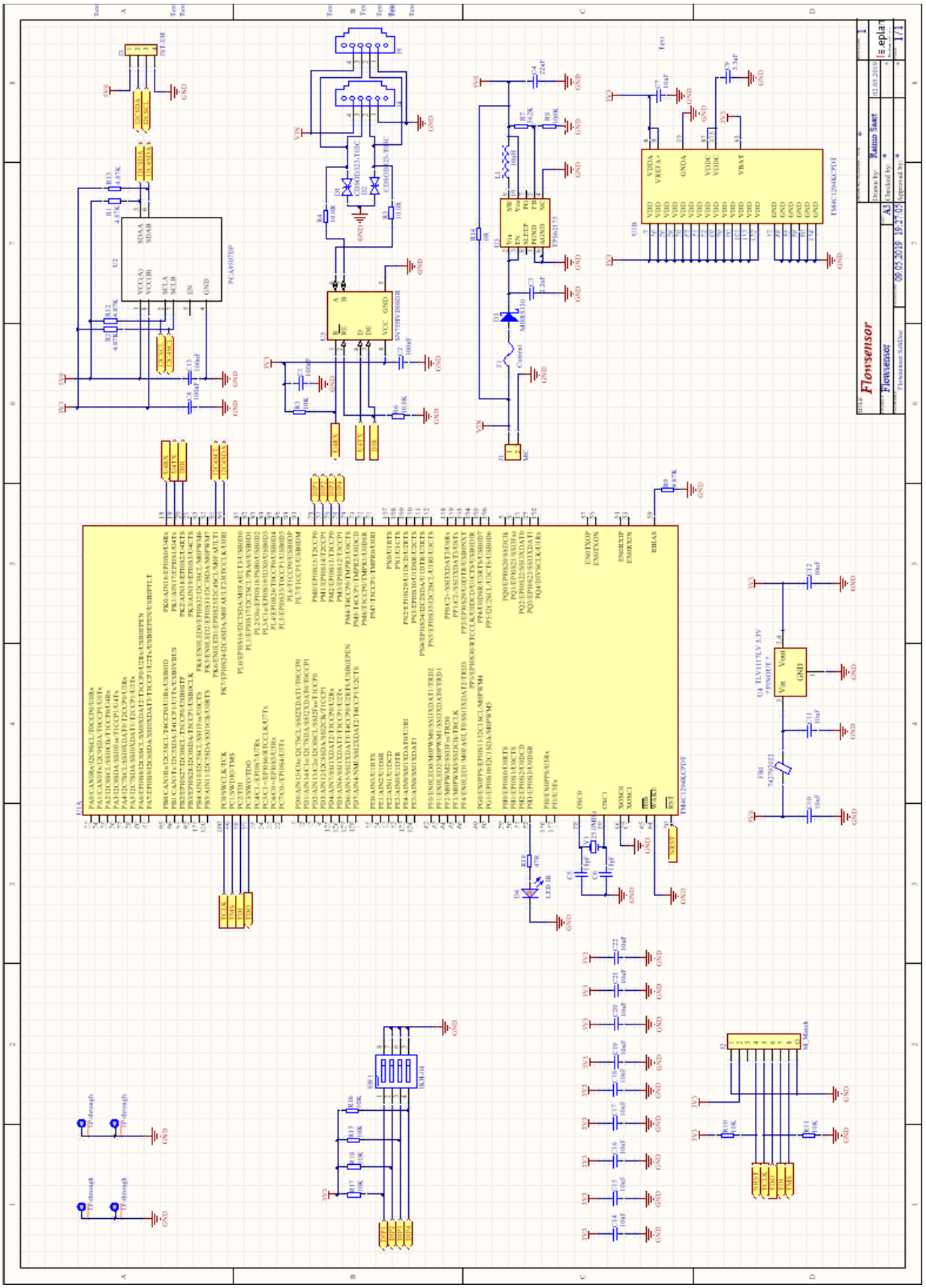
[23] J. Valdez ja J. Becker, „Understanding the I2C bus,“ Texas Instruments, [Võrgumaterjal]. Saadaval: http://www.ti.com/lit/an/slva704/slva704.pdf. [Kasutatud 10 Mai 2019].

[24] E. Priidel, *Traadiga tööstuslikud võrgud – Modbus, CANopen, DeviceNet, Andmeside ja –haldus (IEE1131) õppeaine slaidid.*

[25] S. Raimbault. [Võrgumaterjal]. Saadaval: https://github.com/stephane/modbusino. [Kasutatud 25 Aprill 2019].

LISAD

Lisa 1 Elektroonikaskeem



```
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>


#include "Board.h"


/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"


/*driverlib*/
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "driverlib/interrupt.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"

#include "driverlib/uart.h"
#include "utils/uartstdio.h"

#include "I2C.h"
#include "I2CHelpers.h"
#include "I2CFrameWork.h"
#include "IOControl.h"
#include "flowsensor.h"
#include "modbus.h"
#include "uart.h"
#include "ModBusUSART.h"




/*****************************************************************************
***
 * DEFINES
*****************************************************************************
**/
TickType_t mainBLINK_DELAY=                                pdMS_TO_TICKS(50);
uint32_t g_ui32SysClock;
uint32_t ui32Index;

/* Task stack sizes */
#define mainBLINK_TASK_STACK_SIZE                          (
configMINIMAL_STACK_SIZE + 800 )
#define mainBOX_TASK_STACK_SIZE                            (
configMINIMAL_STACK_SIZE + 800 )

/* Task priorities. */
#define mainBLINK_TASK_PRIORITY                            ( tskIDLE_PRIORITY
+1 )
```

```
#define mainBOX_TASK_PRIORITY                                ( tskIDLE_PRIORITY
+1 )

#define NUM_I2C_DATA 3


static void vFlow();
static void vMBus(void *pvParameters);
void vSetupHardware(void);
int32_t flow = 0;
int32_t tab_reg[10]={0};
int32_t i32IntegerPart = 0;
int32_t i32FractionPart = 0;


uint16_t offset = 32000; //sensor offset
uint16_t scale = 140;//scale factor for air
SimpleCommander_T IOCommander;

//*****************************************************************************
//
// The error routine that is called if the driver library encounters an error.
//
//*****************************************************************************
#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif
/*****************************************************************************
***
 * FUNCTIONS
*****************************************************************************
**/
int main(void)
{
	vSetupHardware();
	vI2CInit();
	xMBInit(115200, (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
UART_CONFIG_PAR_NONE));
	flowstart();
	/* Start the tasks defined within this file/specific to this demo. */
	xTaskCreate( vMBus, "MBUS",mainBLINK_TASK_STACK_SIZE, NULL,
mainBLINK_TASK_PRIORITY, NULL );

	/* Start the scheduler. */
	vTaskStartScheduler();

    /* Will only get here if there was insufficient memory to create the idle
    task. */

	return 0;
}

uint8_t name[EEPROM_PAGE_SIZE * 2];

void vApplicationStackOverflowHook( TaskHandle_t *pxTask, signed portCHAR
*pcTaskName )
```

```
{
	char cName[50];
	int i;

	for (i= 0; i<20; i++)
	{
		cName[i]= pcTaskName[i];
	}

	//DBG("TASK %s CRASHED!",cName);

	__asm("BKPT #0\n") ; // Break into the debugger

	while(1)
	{
		if (cName[i]) i++;
	}

}
void vApplicationMallocFailedHook( void ){
    __asm("BKPT #0\n") ; // Break into the debugger

    /* When the following line is hit, the variables contain the register
values. */
    for( ;; );
}
/*************************************************************************
***
 * FUNCTIONS
*************************************************************************
**/
void vSetupHardware(void)
{

	MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                                            SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |
                                            SYSCTL_CFG_VCO_480), 120000000);


    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOQ);


      //reset the peripherals
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOA);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOB);
```

```
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOC);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOD);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOE);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOF);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOG);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOH);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOJ);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOK);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOL);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOM);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPION);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOP);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_GPIOQ);

    extern void FPULazyStackingEnable(void);
    FPULazyStackingEnable();
}


static void vFlow()
{
             i32IntegerPart = 0;
             i32FractionPart = 0;
             flow = 0;
             flowread(&flow);

             flow = ((flow & 0x00FF00) << 0) | ((flow & 0xFF0000) >> 16);
//reverse bytes
             flow = (flow-offset)*2000/scale + 1;
             flow = flow/2;
             i32IntegerPart = flow/1000;
             tab_reg[0] = i32IntegerPart;
          i32FractionPart = flow;
          i32FractionPart = i32FractionPart - (i32IntegerPart * 1000);
          if(i32FractionPart < 0)
          {
             i32FractionPart *= -1;
          }

          tab_reg[1] = i32FractionPart;

}
static void vMBus(void *pvParameters)
{
       GPIOPinTypeGPIOInput(GPIO_PORTM_BASE, GPIO_PIN_0 | GPIO_PIN_1 |

       GPIO_PIN_2 | GPIO_PIN_3);
       for(;;)
       {
             memset(tab_reg, 0, 10);
             _slave = 0;
             ModbusSlave();
             vFlow();
             loop(tab_reg, 2);
             vTaskDelay(mainBLINK_DELAY);
       }
}
```

```
#include "modbus.h"
#include "uart.h"
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "ModBusUSART.h"

#define _MODBUS_RTU_SLAVE 0
#define _MODBUS_RTU_FUNCTION 1
#define _MODBUS_RTU_PRESET_REQ_LENGTH 6
#define _MODBUS_RTU_PRESET_RSP_LENGTH 2

#define _MODBUS_RTU_CHECKSUM_LENGTH 2

#define _MODBUSINO_RTU_MAX_ADU_LENGTH 128

/* Supported function codes */
#define _FC_READ_HOLDING_REGISTERS 0x03
#define _FC_WRITE_MULTIPLE_REGISTERS 0x10

enum { _STEP_FUNCTION = 0x01, _STEP_META, _STEP_DATA };
uint32_t g_ui32SysClock;
uint8_t lastslave = 0;
uint8_t status = 0;

static uint16_t crc16(uint8_t *req, uint8_t req_length)
{
    uint8_t j;
    uint16_t crc;

    crc = 0xFFFF;
    while (req_length--) {
        crc = crc ^ *req++;
        for (j = 0; j < 8; j++) {
            if (crc & 0x0001)
                crc = (crc >> 1) ^ 0xA001;
            else
                crc = crc >> 1;
        }
    }
    return (crc << 8 | crc >> 8);
}

void ModbusSlave()
{
      uint8_t dip1 = 0;
```

```
      uint8_t dip2 = 0;
      uint8_t dip3 = 0;
      uint8_t dip4 = 0;
      uint8_t address = 0;
      dip4 = GPIOPinRead(GPIO_PORTM_BASE,GPIO_PIN_0) && GPIO_PIN_0;
      dip3 = GPIOPinRead(GPIO_PORTM_BASE,GPIO_PIN_1) && GPIO_PIN_1;
      dip2 = GPIOPinRead(GPIO_PORTM_BASE,GPIO_PIN_2) && GPIO_PIN_2;
      dip1 = GPIOPinRead(GPIO_PORTM_BASE,GPIO_PIN_3) && GPIO_PIN_3;
      address = ((dip1) + (dip2 << 1) + (dip3 << 2) + (dip4 << 3));
    if ((address >= 0) && (address <= 16)) {
        _slave = address;
    }
}
static int check_integrity(uint8_t *msg, uint8_t msg_length)
{
    uint16_t crc_calculated;
    uint16_t crc_received;

    if (msg_length < 2)
        return -1;

    crc_calculated = crc16(msg, msg_length - 2);
    crc_received = (msg[msg_length - 2] << 8) | msg[msg_length - 1];

    /* Check CRC of msg */
    if (crc_calculated == crc_received) {
        return msg_length;
    } else {
        return -1;
    }
}

static void send_msg(uint8_t *msg, uint8_t msg_length)
{
    uint16_t crc = crc16(msg, msg_length);

    msg[msg_length++] = crc >> 8;
    msg[msg_length++] = crc & 0x00FF;
    for(int i=0; i< msg_length;i++)
    {
      xMBSendChar(msg[i]);
    }
}
static int build_response_basis(uint8_t slave, uint8_t function, uint8_t *rsp)
{
    rsp[0] = slave;
    rsp[1] = function;
    return _MODBUS_RTU_PRESET_RSP_LENGTH;
}
static uint8_t response_exception(uint8_t slave, uint8_t function,
                                  uint8_t exception_code, uint8_t *rsp)
{
    uint8_t rsp_length;

    rsp_length = build_response_basis(slave, function + 0x80, rsp);

    /* Positive exception code */
    rsp[rsp_length++] = exception_code;
```

```
    return rsp_length;
}

static void flush(void)
{
    uint8_t i = 0;

    /* Wait a moment to receive the remaining garbage but avoid getting stuck
     * because the line is saturated */
    while (i++ < 10) {
       xMBDeleteQueue();
        SysCtlDelay(3);
    }
}

static int receive(uint8_t *req, uint8_t _slave, uint8_t status)
{
    uint8_t length_to_read = 0;
    uint8_t req_index = 0;
    uint8_t step = 0;
    uint8_t function = 0;
    uint8_t currentslave = 0;
    /* We need to analyse the message step by step.  At the first step, we want
     * to reach the function code because all packets contain this
     * information. */
    step = _STEP_FUNCTION;
    length_to_read = _MODBUS_RTU_FUNCTION + 1;

    req_index = 0;
    while (length_to_read != 0) {

        xMBReceiveChar((int8_t*)&req[req_index]);
        /* Moves the pointer to receive other data */
             req_index++;
        /* Computes remaining bytes */
             length_to_read--;
             if (length_to_read == 0) {
                if (req[_MODBUS_RTU_SLAVE] != _slave) {
                    flush();
                    return -1; /*- MODBUS_INFORMATIVE_NOT_FOR_US;*/
                }
            switch (step) {

            case _STEP_FUNCTION:
                /* Function code position */
                function = req[_MODBUS_RTU_FUNCTION];
                currentslave = req[_MODBUS_RTU_SLAVE];
                if (function == _FC_READ_HOLDING_REGISTERS) {
                    length_to_read = 4;
                }
                 else {
                    flush();
                    if (req[_MODBUS_RTU_SLAVE] == _slave) {
                        uint8_t rsp_length = response_exception(
                            _slave, function,
MODBUS_EXCEPTION_ILLEGAL_FUNCTION,
                            req);
                        send_msg(req, rsp_length);
                        return -1 - MODBUS_EXCEPTION_ILLEGAL_FUNCTION;
```

```
                    }

                    return -1;
                }
                step = _STEP_META;
                break;
            case _STEP_META:

             if(status == 1 && req[_MODBUS_RTU_SLAVE] == lastslave &&
                         !(req[_MODBUS_RTU_SLAVE] == _slave)){
                   length_to_read = _MODBUS_RTU_CHECKSUM_LENGTH + 1;
             }
             else
             {
                   length_to_read = _MODBUS_RTU_CHECKSUM_LENGTH;
                   status = 0;
             }

                if ((req_index + length_to_read)
                    > _MODBUSINO_RTU_MAX_ADU_LENGTH) {
                    flush();
                    if (req[_MODBUS_RTU_SLAVE] == _slave) {
                        uint8_t rsp_length = response_exception(
                            _slave, function,
                            MODBUS_EXCEPTION_ILLEGAL_DATA_VALUE, req);
                        send_msg(req, rsp_length);
                        return -1 - MODBUS_EXCEPTION_ILLEGAL_FUNCTION;
                    }
                    return -1;
                }
                step = _STEP_DATA;
                break;
            default:
                break;
            }
        }
    }
    lastslave = currentslave;
    return check_integrity(req, req_index);
}

static void reply(int32_t *tab_reg, uint16_t nb_reg, uint8_t *req,
                  uint8_t req_length, uint8_t _slave)
{
    uint8_t slave = req[_MODBUS_RTU_SLAVE];
    uint8_t function = req[_MODBUS_RTU_FUNCTION];
    uint16_t address =
        (req[_MODBUS_RTU_FUNCTION + 1] << 8) + req[_MODBUS_RTU_FUNCTION + 2];
    uint16_t nb =
        (req[_MODBUS_RTU_FUNCTION + 3] << 8) + req[_MODBUS_RTU_FUNCTION + 4];
    uint8_t rsp[_MODBUSINO_RTU_MAX_ADU_LENGTH];
    uint8_t rsp_length = 0;

    if (slave != _slave) {
        return;
    }

    if (address + nb > nb_reg) {
        rsp_length = response_exception(
```

```
            slave, function, MODBUS_EXCEPTION_ILLEGAL_DATA_ADDRESS, rsp);
    } else {
        req_length -= _MODBUS_RTU_CHECKSUM_LENGTH;

        if (function == _FC_READ_HOLDING_REGISTERS) {
            uint16_t i;

            rsp_length = build_response_basis(slave, function, rsp);
            rsp[rsp_length++] = nb << 1;
            for (i = address; i < address + nb; i++) {
                rsp[rsp_length++] = tab_reg[i] >> 8;
                rsp[rsp_length++] = tab_reg[i] & 0xFF;
            }
        } else {
            uint16_t i, j;

            for (i = address, j = 6; i < address + nb; i++, j += 2) {
                /* 6 and 7 = first value */
                tab_reg[i] = (req[_MODBUS_RTU_FUNCTION + j] << 8)
                             + req[_MODBUS_RTU_FUNCTION + j + 1];
            }

            rsp_length = build_response_basis(slave, function, rsp);
            /* 4 to copy the address (2) and the no. of registers */
            memcpy(rsp + rsp_length, req + rsp_length, 4);
            rsp_length += 4;
        }
    }

    send_msg(rsp, rsp_length);
}


int loop(int32_t *tab_reg, uint16_t nb_reg)
{
    int rc = 0;
    uint8_t req[_MODBUSINO_RTU_MAX_ADU_LENGTH];

    if (!(UARTBusy(UART4_BASE))) {
        rc = receive(req, _slave, status);

        if (rc > 0) {
            reply(tab_reg, nb_reg, req, rc, _slave);
        }
    }
    /* Returns a positive value if successful,
       0 if a slave filtering has occured,
       -1 if an undefined error has occured,
       -2 for MODBUS_EXCEPTION_ILLEGAL_FUNCTION
       etc */

    return rc;
}
```

Lisa 4 Õhu vooluhulga mõõtemooduli I2C initsialiseerimine C-programmeerimiskeeles

```
#include "flowsensor.h"
#include "I2C.h"


#define NUM_I2C_DATA 3

int flowstart(void)
{
	uint32_t TXdata = 0x0010;
	return xI2CTX(I2C_4, 0x40, 2,(unsigned char*) &TXdata, true);
}

int flowread(int16_t*sample)
{
	return xI2CRX(I2C_4, 0x40, NUM_I2C_DATA, (unsigned char*)sample, true);
}
```

```
from __future__ import division
import serial.tools.list_ports
import minimalmodbus
import time

def findSerialPort(intVendorID = 6790):
    selectedPort = -1

    ports = list(serial.tools.list_ports.comports())

    if len(ports) < 1:
        return ""

    for i in range(0, len(ports)):
        if ports[i].vid == intVendorID:
            selectedPort = i;

    if selectedPort < 0:
        return ""

    return ports[selectedPort].device

serialPortName = findSerialPort()

minimalmodbus.BAUDRATE = 115200
address = 2
enabled = True
i = 0

while enabled:
    #print ("address=", str(address))
     airflow = minimalmodbus.Instrument(serialPortName, address)
     try:
        airflowValue = 0.2
        reg_values = airflow.read_registers(0, 2);
        time.sleep(0.1)
        airflowValue = reg_values[1] / 1000
        if reg_values[0] < 0x8000:
            airflowValue += reg_values[0]
        else:
            airflowValue += (reg_values[0] - 0xffff)
        print ("address=", str(address))
        print ("%.3f" % airflowValue)
     except (IOError, ValueError) as e:
        print ("address=", str(address))
        print ("Failed to read value!" + str(e))
     i = i + 1
     time.sleep(0.1)
     if i > 5:
         address = 11
     if i > 10:
         enabled = False
```

Lisa 6 Mõõtemooduli korpuse mehaaniline joonis