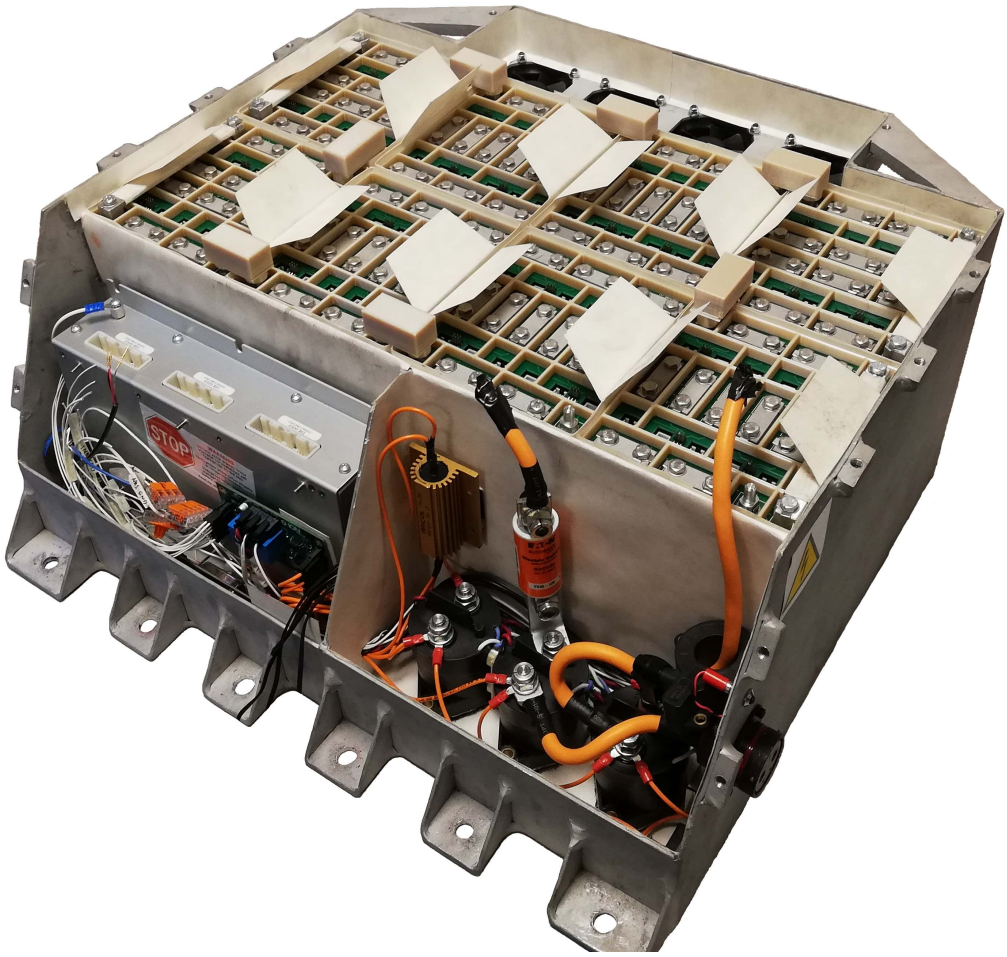
Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Automatisering og elektronikkdesign	Vårsemesteret, 2021 Åpen
Forfatter: Per G. Lund, Andreas Byskov	
Fagansvarlig: Sven Ole Aase Veileder(e): Sven Ole Aase	
Tittel på bachelorpppgaven: Implementering av overvåkingssystem for batteri i elektrisk racerbil Engelsk tittel: Implementation of a battery management system in an electric racecar	
Studiepoeng: 20	
Emneord: Batteri, BMS, ION Racing, Formula Student	Sidetall: 54 + vedlegg/annet: 36 Stavanger, 15.05.2021 dato/år



Implementering av overvåkingssystem for batteri i elektrisk racerbil

Andreas Byskov og Per G. Lund

Universitetet i Stavanger
Vår 2021

Sammendrag

Denne oppgaven er skrevet i samarbeid med ION Racing, en studentorganisasjon ved Universitetet i Stavanger som hvert år bygger en elektrisk racerbil for å konkurrere i Formula Student konkurransen. Oppgaven dreier seg om implementering og ferdigstillelse av et batteriovervåkningssystem i racerbilens batterisystem. Batteriovervåkingssystemet var delvis designet i en tidligere masteroppgave skrevet våren 2020 [12], men pga. koronasituasjonen som oppstod ble lite av de praktiske og programmerings delene i implementeringen fullført. For å implementere et fullstendig system dekker da oppgaven videre design av tidligere tenkt batterisystem, konstruksjon av batteripakker med nødvendige kretskort og programmering av overvåkingssystemet med nødvendig kommunikasjon til eksisterende system i bilen.

Under designprosessen ble det tydelig at flere endringer måtte utføres på det tidligere designet ettersom det ble funnet feil som ikke ville fungere og designløsninger som ikke var tillatt etter konkurransens regelverk. Det ble dermed gjort flere design endringer på alle de forskjellige kretskortene (tilkoblings-, slave- og master-kretskort) som inngår i batterisystemet. De nydesignede kretskortene har også redusert kompleksitet og økt isolasjonsevne. Batterisystemets konstruksjon har blitt forbedret blant annet med mer isolasjon og redusert sannsynlighet for kortslutning.

Mikrokontrolleren i systemet som styrer mastermodulen har fått implementert fungerende C-kode for overvåking av strøm, spenning og temperatur i batterisystemet. Samtidig med kommunikasjon over isoSPI til styring av slavemoduler, kommunikasjon over CANbus til andre systemer i bilen og kommunikasjon over RS232 til datamaskin. Det er også implementert cellebalansering og enkel estimering av ladestatus. På datamaskinen er det implementert et kjørende grafisk brukergrensesnitt i Python som presenterer all data fra batteriovervåkingssystemet. Her utføres enkel databehandling for å fremstille nøkkeldata og støtter ulike ekstra funksjoner.

Det er altså implementert kode og redesignet nødvendige kretskort for å oppnå alle de opprinnelige mål som medfølgte oppgaveteksten. For å videre ferdigstille batteriovervåkingssystemet mangler en del testing med sammenstilling av hele bilen. Dette er en begrensning som ikke har nådd opp ettersom det er flere andre systemer på bilen fra andre medlemmer i ION Racing som må ferdigstilles før fullstendig testing av batteriet kan gjennomføres. Testene som er utført så langt viser gode resultater som tyder på et vellykket produkt.

Forord

Oppgaven er utført av Andreas Byskov og Per G. Lund i samarbeid med studentprosjektet ION Racing ved universitetet i Stavanger. Vi har begge vært medlem i studentprosjektet i nå 3 år, helt siden starten av studiet. Det har vært krevende og vi har hatt mye arbeid til tider sammen med studiet, men det har samtidig vært enormt lærerikt og spennende. Vi har fått jobbet med mye praktisk elektronikk som har også hjulpet oss igjennom studiene. På starten av denne sesongen var laget i ION Racing svekket av koronasituasjonen med mindre interesse vist blant nye studenter for prosjektet. Dette medførte til mangel på medlemmer og at flere systemer på bilen ikke ble ferdigstilt til den tid som var forventet. Oppgaven vår var kun en av mange systemer på bilen og trengte de andre systemene for å kunne kjøre fullstendig. Allikevel fikk vi implementert et komplett system som kun mangler et par endelige tester for å ferdigstilles.

Takk

Vi ønsker å rette en takk til de følgende:

- Vår veileder for oppgaven, Sven Ole Aase
- Romauld Karol Bernacki, for lån av utstyr.
- Yaaseen Ahmad Amith, for 3D-printing av designet isoleringsdeler til batteriet.
- Alle sponsorer av ION Racing som er med på å gi studenter muligheten til å utfordre seg selv i det som er et svært spennende og lærerikt prosjekt.
- Alle tidligere og aktive medlemmer av ION Racing for 3 år med gode minner. Særlig tidligere medlem Raymond Tjørhom for sine innspill som var med på å starte oppgaven.

Andreas Byskov, Per G. Lund
Stavanger, Mai 2021

Innhold

Figurer

Kodesnutter

Nomenklatur

AIR Accumulator Isolation Relay
Ved feil aktiveres et mekanisk relé som elektrisk isolerer batteriet og HV-systemet

BMS Battery Management System
Batteriovervåkning system

BSPD Brake System Plausibility Device
Sikkerhetskrets i bilen som hindrer brems og gass samtidig

CAD Computer-aided design
Dataassistert konstruksjon, 3D-modellering og teknisk tegning utført ved hjelp av datamaskin

CAN Controller Area Network
Kommunikasjonsbus

DMA Direct Memory Access
Direkte minneoverføring

ECU Electronic Control Unit
Overordnet kontrollenhet for bilens elektriske system

EV Electric Vehicle
Elektrisk bil

FSG Formula Student Germany
Formula Student konkurranse som arrangeres i Hockenheimring, Tyskland

FSN Formula Student Netherlands
Formula Student konkurranse som arrangeres i Assen, Nederland

FSUK Formula Student United Kingdom
Formula Student konkurranse som arrangeres i Silverstone, England

GLV Grounded Low Voltage
12 V system i bilen til elektronikk utenom HV-systemet

GUI Graphical User Interface
Grafisk brukergrensesnitt

HV High Voltage
Høyspenning (over 60 VDC i Formula Student)

I2C Interconnected Integrated Circuit
Kommunikasjonsprotokoll for integrerte kretser

IC Integrated Circuit
Integrert krets

LED Light Emitting Diode
Lysdiode

PCB Printed Circuit Board
Kretskort

PWM Pulse-width modulation
Pulsbreddemodulasjon

SOA Safe Operating Area
Operasjonsområdet for battericeller, med tanke på grenseverdier av strøm, spenning og temperatur

SOC State of Charge
Ladestatus

SPI Serial Peripheral Interface
Kommunikasjonsprotokoll

TS Tractive System
Drivlinjens elektriske system som inngår i HV-systemet

VDC Volt, Direct Current
Likespenning

Kapittel 1

Introduksjon

1.1 ION Racing og Formula Student

Oppgaven utføres i samarbeid med studentprosjektet ION Racing ved universitetet i Stavanger. Studentprosjektet bygger racerbiler for å delta årlig i Formula Student, en av de største studentkonkurransene i verden. Formula Student handler om å utføre et stort prosjekt med dokumentering, beregning, design og montering av alle systemer som må til for å lage en racerbil som ligner en mindre type formelbil (se illustrasjon i figur 1.1.1 under). Prosjektet har totalt bygget 8 biler over de siste årene hvor de 6 siste bilene har vært elektriske.

Figur 1.1.1: CAD-fremstilling av denne sesongens racerbil.

Konkurransen

Formula Student har flere delkonkurranser over hele verden som gir god tilgang for ulike universitet å komme sammen for å konkurrere. Blant annet i Europa arrangeres det hvert år konkurranser i England, Nederland, Tyskland, Sveits og Ungarn. ION Racing deltok i 2019 på konkurranse ved Silverstone-banen i England (FSUK) og ved Assenbanen i Nederland (FSN). I år (2021) er planen at ION Racing skal delta på den samme konkurransen i England ved Silverstone-banen.

De ulike konkurransene er stort sett like, hvor de er delt i forskjellige kategorier. Dynamiske øvelser måler ytelse og statiske øvelser vurderer design, planlegging, konsept og markedsføring. Statiske øvelser gjennomgår mer spesifikt designet til bilen og hvilke prioritertinger laget har gjort med bruk av tilgjengelige ressurser. Det skal også

bedømmes en forretningsplan for salg av bilen for potensielle investorer. De dynamiske øvelsene gjennomgår bilens kjøreegenskaper på ulike typer baner. Først utføres det øvelser for å teste at bilen tåler alle krefter som vil oppstå under kjøring på banen, dette inkluderer akselerasjonløp (75 m rett bane som tester drivverket), 8-tallsløp (2 sirkelsvinger slått sammen som tester hjulopphenget). Når disse er gjennomgått kan bilen delta på øvelsene ute på banen, som inkluderer sprintløp (en fullstendig runde på kjørebanen) og utholdenhetsløp (flere runder på kjørebanen som tilsvarer 22 km) som er den siste øvelsen og den som utgjør størst andel av poengsummen.

Konkurransens regelverk

For å delta på konkurransene må tilsvarende regelverk for konkurransen følges. Regelverkene er stort sett nokså like, blant annet bruker alle de nevnte konkurransene i Europa regelverket som utgis fra Formula Student Germany (FSG), men typisk legges det til egne små endringer (*supplementary rules*). Som nevnt skal ION Racing forhåpentligvis delta i FSUK dersom koronasituasjonen tillater det, dermed designes bilen rundt regelverket som brukes i FSUK. Her brukes FSG-regelverket [6] som grunnmur og i tillegg noen endringer fra FSUK sine egne tilleggsregler [7].

1.2 Motivasjon

Helt siden 2017-sesongen har ikke ION Racing fått tillatelse til å kjøre under konkurransene pga. mangel på fungerende sikkerhetsfunksjoner som kreves av regelverket. Hovedsakelig er det sikkerhetsfunksjoner som overvåker batteriet under lading og kjøring som har vært det største problemet. Dette førte til at det ble satt et fokus i fjorårets og årets sesong på at sikkerhetssystemene rundt batteriet skulle ha første prioritet. I fjorårets sesong ble en masteroppgave utlyst for å løse problemmet, men pga. blant annet koronasituasjonen som oppstod ble ikke denne delen av oppgaven ferdigstilt. I årets sesong ble det igjen utlyst oppgave for problemet, men enda mer rettet mot overvåkingssystemet og fikk tittelen *Implementering av overvåkingssystem for batteri i elektrisk racerbil.*

1.3 Oppgavetekst

Implementering av overvåkingssystem for batteri i elektrisk racerbil

Det skal designes, konstrueres og testes et styringssystem (ofte betegnet BMS - battery management system) som overvåker og beskytter batteribanken som brukes i årets racerbil. Styringssystemet skal opprettholde begrensninger i form av gitte regelverk og andre systemer som er i bilen.

Målet med oppgaven er å utvikle et styringssystem til batteribanken med følgende begrensninger og funksjoner:

- Batteribanken som skal overvåkes består av 8 batteripakker med 12 cellemoduler koblet i serie, hvor hver av modulene består av 10 celler koblet i parallell.
- Overvåking av temperatur og spenning i battericellene for å sikre at de opererer innenfor sikre grenser.
- Overvåking av strømtrekk til overstrømsbeskyttelse.
- Styring av kjølesystem for å holde et sikkert temperaturområde.
- Styring av lader og balansering av celler for å hindre overlading av batteri.
- Estimering av ladestatus.
- Presentasjon av celledata gjennom brukergrensesnitt på datamaskin for verifisering.
- Integrasjon med det eksisterende systemet i racerbilen.
- Tilfredsstille konkurransens regelverk.

Kapittel 2

Bakgrunn og teori

2.1 Tidligere arbeid

Oppgaven bygger på tidligere design utført for ION Racing, hovedsakelig en masteroppgave skrevet våren 2020 [12]. Den nevnte oppgaven fikk ikke utført den praktiske implementeringen av batteriovervåkingssystemet. Dermed er situasjonen at noe design for konstruksjonen av batterisystemene er utført, men lite er realisert i praksis. Samtidig er det også kun designet av hardware som er utført og svært lite er gjort av nødvendig software/programmering som trengs til systemet.

2.2 Batterisystemet

Batterisystemet i bilen forsyner det elektriske drivverket, som hovedsaklig består av motorkontrolleren og motoren. Regelverket i formula student stiller strenge krav til batterisystemet ettersom det inneholder mye energi og er en av de farligste komponentene i bilen. Batteriet må plasseres i en batterikasse som skal kunne tåle opp til 40 g for å beskytte det under kollisjoner. Batteriet må deles opp i batterisegment som ikke overskrider 120 V, 6 MJ eller 12 kg. Disse må kobles sammen med vedlikeholdsplugger, som enkelt skal kunne kobles fra for å de-energisere batterisystemet når batterikassen er åpen. Mellom batteriet og eksterne tilkoblinger må batterisystemet også ha minst én sikring og ett relé per fase, som skal koble lasten fra batteriet hvis det oppstår usikre omstendigheter.

Et batteriovervåkning system (Battery Management System, BMS) må overvåke tilstanden til alle battericellene ved å måle strøm, spenning og temperaturen de opereres under. Systemet må beskytte cellene hvis de opereres utenfor sikker tilstand ved å aktivere et feilsignal som åpner releene til batteriet. Feilsignalet må være aktiv lavt for å kunne detektere en åpen krets. Det må også kontrollere et kjølesystem som kjøler batterisystemet for å unngå overoppheting.

2.3 Li-ion battericeller

Li-ion battericeller er en type oppladbare battericeller som ofte brukes i moderne elektronikk. Celletypen er på topplisten av energitetthet blant oppladbare celler, dette er viktig i applikasjoner hvor vekt og størrelse spiller store roller, som f.eks. i mobiltelefoner, bærbare PCer og elektriske kjøretøy. De fleste nye elbiler som produseres i dag bruker li-ion batterier blant annet av samme grunn. Det er også derfor li-ion celler brukes i batterikassene til ION Racings elektriske racerbiler.

Cellene har typisk 3 forskjellige mekaniske former, de kan være sylindriske (pakket i metallsylinder), prismatiske (pakket i rektangulært plastkasse) eller såkalte poseceller (pakket i fleksible poser). Tidligere har det vært brukt poseceller i batteriet i bilen for å oppnå lavest mulig vekt for batteriet. I praksis ble posecellene mer krevende ettersom de ikke tålte like mye som f.eks. sylindriskeceller. Posecellene hadde tendenser til å svelle opp og det var risiko for å punktere cellene når de var pakket trangt i batteriet. I årets bil er det derfor kjøpt inn cellemoduler basert på sylindriske Samsung INR18650-25R [8] celler som bedre holder form og øker sikkerheten.

2.4 Batteriovervåkning

Hovedoppgaven til et batteriovervåkning system (BMS) er å overvåke at cellene i batteriet holdes innenfor deres tillatte grenser for strøm, spenning, og temperatur (Safe Operating Area, SOA). Dersom cellene avviker fra dette området skal batteriovervåkning systemet koble fra og stoppe lading/utladning av batteriet.

Særlig for Li-ion batterier kan det være store konsekvenser hvis man mister kontroll på strømtrekk, spenning og temperatur av cellene. Da kan det oppstå "thermal runaway" når temperaturen øker og fører til økning i cellemotstanden og større effekttap, som igjen øker temperauturen på cellene. I de fleste tilfeller er det bare levetiden til batteriet som blir redusert, men i de verste tilfeller kan cellene forårsake flammer og eksplosjon.

Et batteriovervåkingssystem kan kategoriseres i forskjellige topologier basert på hvordan de er koblet til batterisystemet og hvordan de fungerer. Dette er en viktig karakteristikk ettersom den påvirker konstruksjon, pålitelighet, vedlikehold og prisen til systemet. BMS-ene kategoriseres vanligvis som [2]:

- Sentralisert
  En sentralisert BMS består av bare en modul som kobles til alle cellene i systemet. Dette har fordelen med at den er kompakt, den billigste varianten ettersom den krever færrest komponenter og kan enkelt byttes ut ved eventuelle feil.
- Modulær
  En modulær BMS ligner en sentralisert topologi, men den består av flere identiske moduler som hver kobles til noen av cellene og til hverandre. Dette har de samme fordelene som sentraliserte BMS-er i tillegg til at tilkobling er enklere pga. færre celler per modul og enkel utvidelse av systemet. Ulempen er at den ofte har en høyere pris på grunn av redundante ubrukte komponenter på modulene og det kreves flere koblinger for å koble dem sammen.
- Master-Slave
  En master-slave konfigurasjon er nokså lik den modulære varianten i at flere moduler kobles til sine celler. Forskjellig kommer fra at en mastermodul styrer systemet, håndterer beregninger og kommunikasjon, mens slavemodulene bare måler spenningene. Fordelene med dette er lik de modulære BMS-ene, slavemodulene kan optimaliseres for måling av spenning og dermed redusere kost.
- Distribuert
  Et distribuert system avviker fra de andre i at den har moduler som bare kobles til en celle hver. Dette gir små moduler, men øker kompleksiteten og kost betraktelig.

ION Racing har tidligere brukt et distribuert system fra Elithion. Dette systemet hadde et lite kretskort koblet til hver celleterminal som sløyfet seg videre til de andre kretskortene på samme segment med ledninger loddet på kortet. ION Racings erfaring er at det har vært et svært upålitelig system, der loddepunktene til ledningene mellom kretskortene ødelegges ved vibrasjon og mekanisk last under montering eller kjøring.

Fra fordelene og ulempene nevnt over og de tidligere erfaringene var det valgt at det egendesignede systemet skulle bruke en master-slave topologi. Da kan i prinsippet hver slavemodul kobles til en batteripakke, slik at batteripakkene enkelt kan byttes ved feil.

Cellebalansering

Produserte celler av samme type vil ikke alltid ha helt like egenskaper. Særlig over tid vil det merkes at ladningen (State of Charge, SOC) til cellene avviker fra hverandre. Dersom en celle i batterisystemet har ulik selvutladningskarakteristikk vil ladningen avvike fra de andre cellene over tid. Ved fullstendig opp- eller utladning av batterisystemet vil denne cellen begrense kapasiteten til batterisystemet hvis det skal holdes innenfor sikre opereringsbetingelser.

Cellebalansering løser dette problemet ved å balansere alle cellene tilbake til samme nivå. Ladning fra de høyeste cellene overføres enten til andre celler med lavere nivå eller utlades i en utladningsresistor. Hvis ladningen overføres kalles det aktiv cellebalansering og gir maksimal utnyttelse av ladningen i batterimodulen. Sistnevnte varianten hvor ladningen utlades i en utladningsresistor kalles passiv cellebalansering og er enklere å implementere, men ikke like energieffektiv som aktiv.

2.5 Estimering av ladestatus

Når man kjører i en bensin-/dieseldrevet bil er det enkelt å måle hvor mye drivstoff som er igjen i tanken og dermed hvor lang rekkevidden for bilen er. For elektriske biler er det samme ønskelig, men å måle mengden energi igjen i batteriene er mye mer komplisert enn en enkel flottør i en tank. Kompleksiteten bak estimering gjør at det er ulike metoder som kan brukes til å løse problemet.

Coulomb-teller

En enkel metode for estimering av ladestatus er å integrere strømtrekket fra batteriet for å måle hvor stor andel av batteriets ladning som har blitt brukt. En ulempe med denne metoden er akkumulering av integrasjonsfeil, der små avvik i målingen integrert over tid vil gi redusert nøyaktighet. Denne metoden tar heller ikke direkte hensyn til temperaturen i battericellene, som også påvirker kapasiteten og batterispenningen.

Kalmanfilter

En mer avansert metode er bruk av kalmanfilter, som kombinerer målinger av både strøm og spenning for mer nøyaktig estimering. Dette er mer grundig beskrevet i den tidligere masteroppgaven om estimering av ladestatus [12].

2.6 Signaloverføring

For å overføre informasjon i og mellom kretser brukes elektriske signaler. Det finnes mange former for signaloverføring, men de klassifiseres hovedsaklig i digitale og analoge signaler.

Analoge signaler

Analoge signaler er elektriske signaler der amplituden til signalet ikke kvantiseres, men kan ha et uendelig spekter av verdier. Dette gir signalene tilnærmet uendelig oppløsning, men de er følsomme for støy ettersom en hver endring i amplituden til signalet vil endre verdien. Over korte avstander og med beskyttelse mot støy kan disse brukes med høy nøyaktighet. For å redusere påvirkning fra støy kan det brukes filtrering som demper utslag i frekvensområder som ikke er ønsket.

Digitale signaler

Digitale signaler er elektriske signaler med diskrete verdier, vanligvis på og av. I praksis er alle elektriske signaler analoge, men ved å kvantisere spenningerområdet for en verdi så reduseres oppløsningen, men signalet blir sterkere og kan operere ved høyere frekvenser. I tilleg har digitale signaler høyere støymotstand fordi en større spenningspåvirkning kreves for å endre signalverdien.

PWM

Pulsbreddemodulering er en form digitale signaler der verdien som overføres enkodes i bredden på en puls med en fast frekvens. På mange måter er dette som et analogt signal i at det i teorien kan ha et uendelig spekter verdier, der oppløsningen på sender og mottaker avgjør presisjonen til signalet.

CANbus

CAN er forkortelsen for *Controlled Area Network* og er en seriell nettverksteknologi for kommunikasjon. CAN-protokollen er robust mot støy på grunn av differensiell signaloverføring og paritetsbyte i hver melding. Opprinnelig laget til biler på 80-tallet har det over tid utviklet seg med flere funksjoner og har også blitt tatt i bruk i flere andre applikasjoner som krever robuste kommunikasjonsnettverk [16]. Med sin stadige store bruk i bilindustrien for kommunikasjon i biler, er det også denne typen kommunikasjon ION Racings biler bruker til hovedkommunikasjon av det elektroniske systemet. Blant annet skal BMS-en i årets bil kommunisere med både laderen av batteriet og kontrolleren av det elektriske systemet (Electronic Control Unit, ECU) via CANbus.

I2C

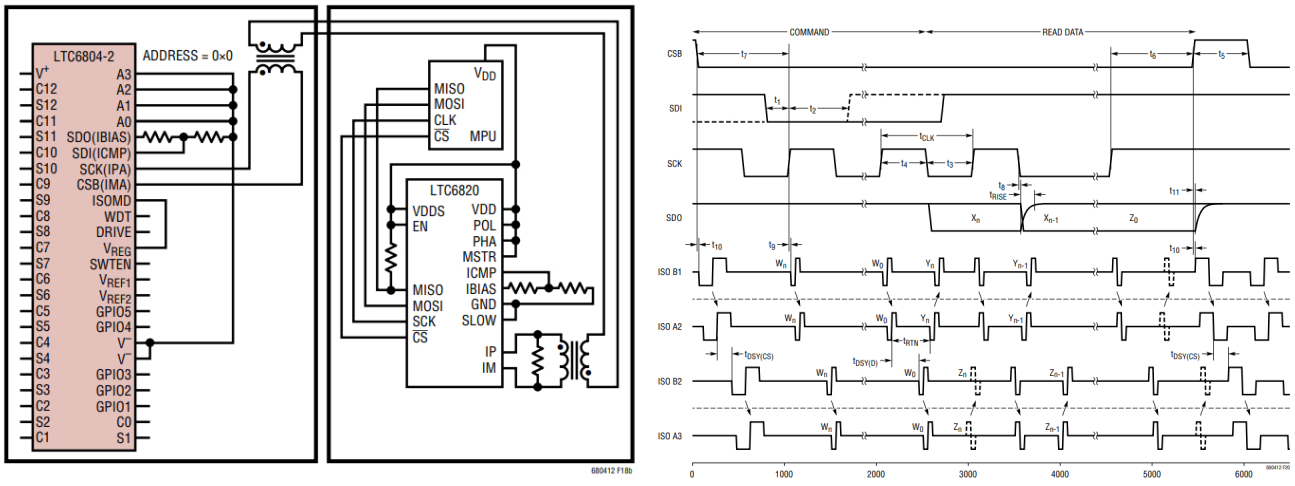
I2C (Inter-Integrated Circuit) er en protokoll for kommunikasjon mellom integrerte kretser over korte avstander. I2C bruker to signaler, SDA og SCL der SDA overfører dataverdiene og SCL overfører et klokkesignal. Vanligvis styres bussen av en master-node som adresserer slavene for å sende og motta data.

SPI

SPI (Serial Perepherial Interface) er en annen protokoll for kommunikasjon mellom integrerte kretser. SPI bruker bruker tre signaler, MISO, MOSI og SCLK, i tillegg til et chip select signal til hver slave som signaliserer når den blir adressert. MISO signalet overfører data fra slaven til masteren, MOSI overfører fra masteren til slaven og SCLK sender klokkesignalet.

isoSPI

IsoSPI er en variant av SPI-protokollen utviklet av *Linear Technologies/Analog Devices* for isolert kommunikasjon mellom kretser. En integrert krets konverterer klokke- og datasignalene fra et SPI-grensesnitt til strømpulser av varierende lengde og rekkefølge som kan overføres gjennom vekselstrømskoblinger som transformatorer og kondensatorer. Dette gjør at grensesnittet kan brukes over isolasjonsbarriærer og gir det høy støymotstand. Ved bruk av LTC6804-en [10] inkluderes også en 15-bits paritetkode i hver overføring, som også øker støymotstanden betraktelig. Figur 2.6.1 under viser til venstre oppbygging av isoSPI-grensesnittet til LTC6804-en og til høyre meldingsprotokollen til denne.



Figur 2.6.1: Figur 18b og 20 fra LTC6804-ens datablad side 40 og 42 [10].

2.7 Kretskortutlegg

Design og utlegg av kretskort kan være en komplisert prosess, særlig med store IC-er med mange pinner koblet til forskjellige kretser. I disse tilfellene kan det være vanskelig å vite hva og hvilke aspekter som bør prioriteres for å få til et godt design. Et par grunnlegende prinsipp bak kretskortdesign fra *learnemc.com* [9] som også brukes i denne oppgaven nevnes under, men prioritering av disse varierer med omstendighetene til utlegget.

- Kontinuerlige jord- og forsyningsplan.
  Dette reduserer motstand og spenningsfall eller "ground-bounce". Dette gir også en returvei med lav impedans for høyfrekvent strøm og reduserer elektromagnetisk utstråling og følsomhet.

- Minimere løkke-areal.
  En løkke der strømmen går gjennom baner på et kretskort fungerer som en induktor og gir banen økt impedans og utstråling. På samme måte som jordplan reduserer dette også elektromagnetisk utstråling.

- Samle kretser til samme funksjon.
  Dette reduserer avstanden på banene, og som et resultat løkke-arealet, og hindrer innvirking fra andre kretser. Ofte vil det også være fordelaktig å plassere spenningsregulatorene nær forbrukerene for å redusere spenningsfall og øke responsiviteten.

- Plassere avkoblingskondensatorer nærmest mulig forsyningspinner.
  Ved å minimalisere avstanden fra avkoblingskondensatorene til forsyningspinnene på en forbrukerkrets reduseres impedansen og kretsen får en mer stabil forsyning ved høyfrekvent strømtrekk.

- Støysensitive komponenter bør plasseres for seg selv, særlig borte fra støyende komponenter med høye strømmer eller frekvenser. Det kan også være gunstig å dirigere sterke eller høyfrekvente strømmer rundt støysensitive komponenter med dedikerte forsyningsbaner som kobles til jord- og forsyningsplan en annen plass.

I mange tilfeller er ikke disse prinsippene kritiske for et fungerende design, og det vil være praktisk umulig å lage et kretskort med perfekte egenskaper, men det vil forbedre robustheten til systemet og sørge for et bedre design. Det må som oftest nås en balanse mellom forskjellige aspekter ut fra formålet.

2.8 Isolasjon

For å isolere kretser fra hverandre må det være høy motsand som hindrer at strøm går mellom dem. Dette betyr at dersom strøm eller signaler skal overføres mellom dem, så må dette skje over andre medier som magnetisme i en transformator eller lys i en optokobler. Det er også viktig å tenke på fysisk separasjon mellom kretsene ettersom ingen materialer er perfekte isolatorer. På et kretskort er det flere aspekter som må tas henhold til:

- Krypstrøm
  Materialene i kretskortet har ikke uendelig motstand, og de vil lede litt strøm mellom nærliggende baner. For å redusere disse må avstanden mellom banene langs overflaten på kretskortet økes ved enten å flytte banene lenger fra hverandre eller å lage en isolasjonsspalte som krypstrømmen må rundt.

- Dielektrisk nedbrytning
  Alle materialer har ett punkt der deres evne til elektrisk isolasjon bryter ned, også kalt en lysbue. For eksempel luft har en nedbrytingsspenning på rundt 1 kV ved havnivå, men dette varierer med faktorer som fuktighet og lufttrykk. For å øke nedbrytingsspenningen må klareringen mellom utsatte ledere i luftvei økes eller dekkes med et isolasjonsbelegg som har bedre isoleringsegenskaper enn luft. Å øke avstandene mellom ledere krever mer plass på kretskortet og prosessene for et isolasjonsbelegg er kostbare og kan føre til svikt av isolasjonsevnen ved feil utførelse.

- Partikler
  Ved operasjon i et forurenset område kan partikler eller ting som spon og skruer redusere isolasjonen på et kretskort. Det beste for å unngå dette vil være å potte eller hermetisk forsegle kretskortet slik at det ikke er mulig for partikler å komme inn. Det nest beste vil være å ha større separasjon mellom banene for å øke marginen for isolasjonen.

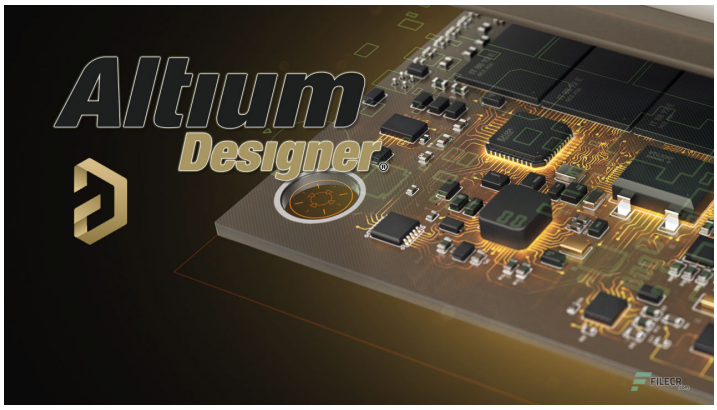
I følge reglene i Formula Student [6] skal alle kretskort med tilkoblinger til både det elektriske drivverket (Tractive System, TS) og 12 V systemet i bilen (Grounded Low Voltage, GLV) ha tydelig merket område designert til klarering mellom systemene. I følge EV 4.3.5 [6] må systemer med elektrisk drivverk (Tractice System, TS) på mer enn 300 V ha klarering på minst 12.7 mm over ubehandlet overflate, 9.5 mm over isolasjonsspalte og 4 mm med isolasjonsbelegg. En standard som ofte brukes i industrien for klarering er IPC2152. I følge denne standarden skal systemer mellom 301 til 500 V ha 2.5 mm klarering under vanlige omstendigheter, men hvis det skal være robust bør det være dobbelisolert, altså det dobbelte, som blant annet kreves for utstyr som skal kobles til stikkontakter uten jording. Standarden sier også at i et annet tilfelle, der systemet skal operere over 3 km høyde så bør klareringen være 12.5 mm på grunn av reduserte lufttrykk.

2.9 Utstyr

Programvare

Altium Designer

Programvare for kretskortdesign, tilgjengelig via sponsoravtale med ION Racing.

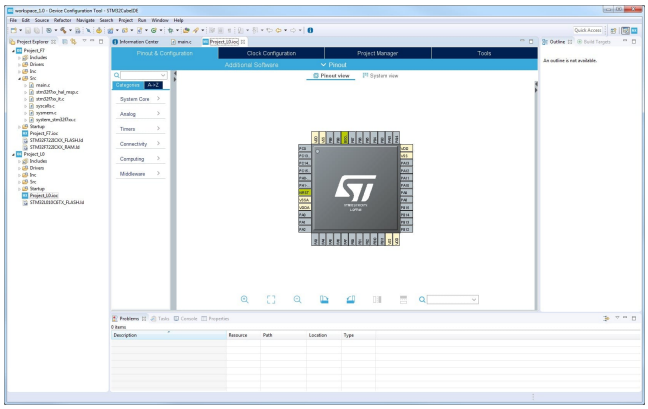


Matlab/Simulink

Programvare for simulering og testing, tilgjengelig via UiS sine studentlisenser.

STM32CubeIDE

Integrert utviklingsmiljø for C-programmering, tilgjengelig gratis via produsentens nettsider.



Spyder(Anaconda3)

Integrert utviklingsmiljø for Python-programmering, tilgjengelig gratis via produsentens nettsider.

QtDesigner

Programvare for GUI-design hvor Python kode auto-genereres fra selvlaget design, tilgjengelig gratis via produsentens nettsider.

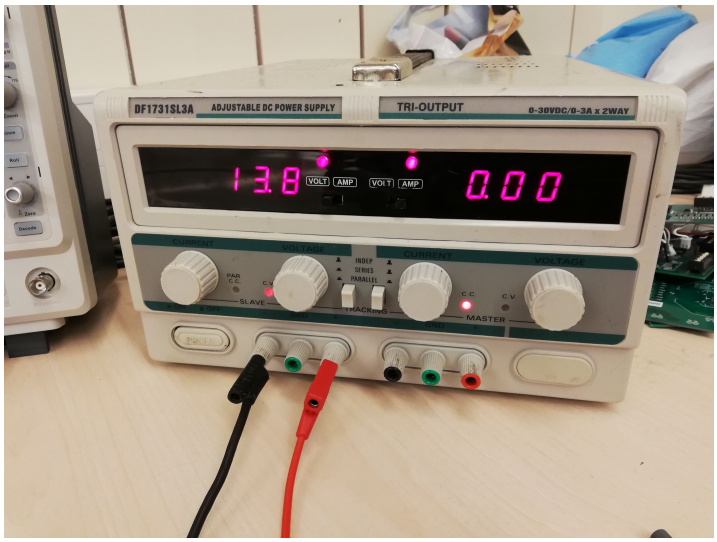
Autodesk Inventor

Programvare for design av CAD-deler til mekanisk konstruksjon, tilgjengelig gratis for studenter via produsentens nettsider.

Labutstyr

Oscilloskop, SDS2104X Plus (4-channel, 100MHz, 2GSa/s)

Strømforsyning, Protek DF1731SL3A (Dual 30V 3A)



Strømforsyning, 6268B DC POWER SUPPLY (40V 30A)

Multimeter, EEVBlog 121GW

Debugger/Programmer, ST-LINKv3

Komponenter og materialer

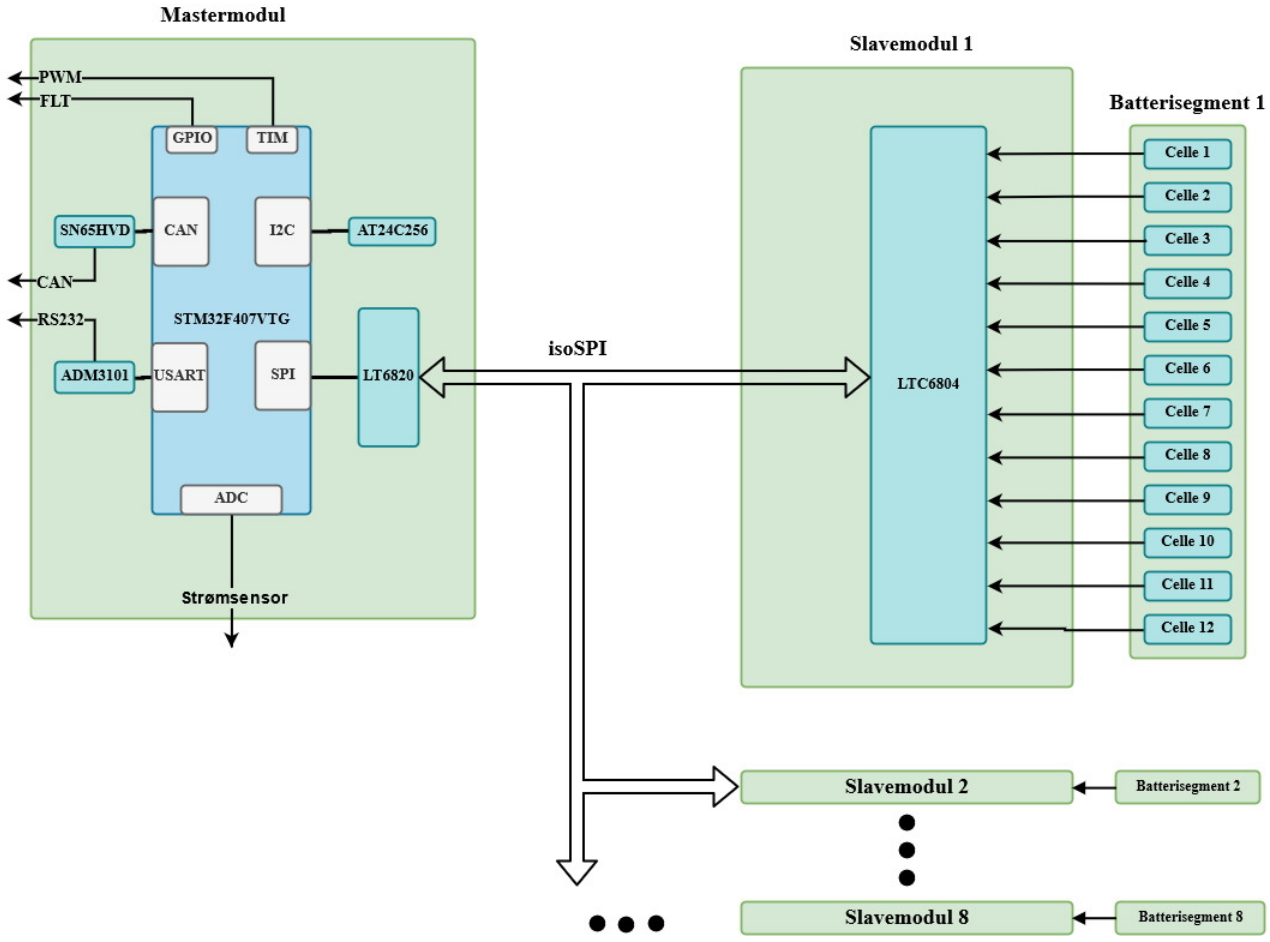
Alt av komponenter og materialer som brukes i oppgaven med unntak av 3D-printet deler og labutstyr betales av ION Racing og trekkes fra budsjettet for elektroavdelingen. Delene som 3D-printes er printet hos universitetet i Stavanger.

Kapittel 3

Design og konstruksjon

3.1 Batterisystemet

Som nevnt er det bruk for redesign av flere deler av batterisystemet for å opprettholde regelverket og de nye cellemodulene som ble valgt og kjøpt i løpet av 2020 sesongen. Blokkskjemaet under i figur 3.1.1 viser en forenklet illustrasjon av batterisystemet. Mastermodulen styrer 8 slavemoduler over isoSPI hvor hver at disse er montert på batterisegmenter med tilkoblingskort til 12 cellemoduler. Mastermodulen kommuniserer også med lader av batteriet og bilens sentrale styringsenhet (Electronic Control Unit, ECU) via CANbus. Et grafisk brukergrensesnitt kommuniserer også med mastermodulen over RS232.

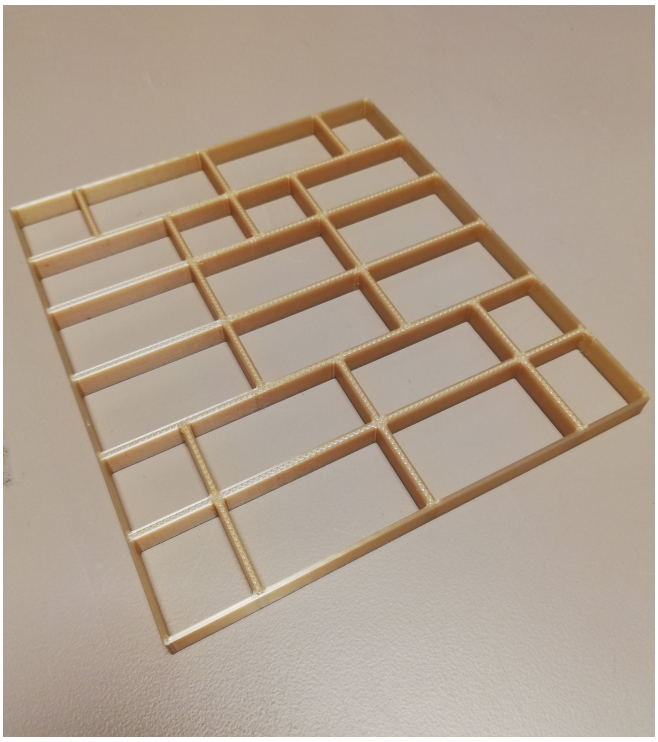


Figur 3.1.1: Overordnet blokkskjema av batterisystemet, med én mastermodul og 8 slavemoduler fordelt på 8 batterisegmenter.

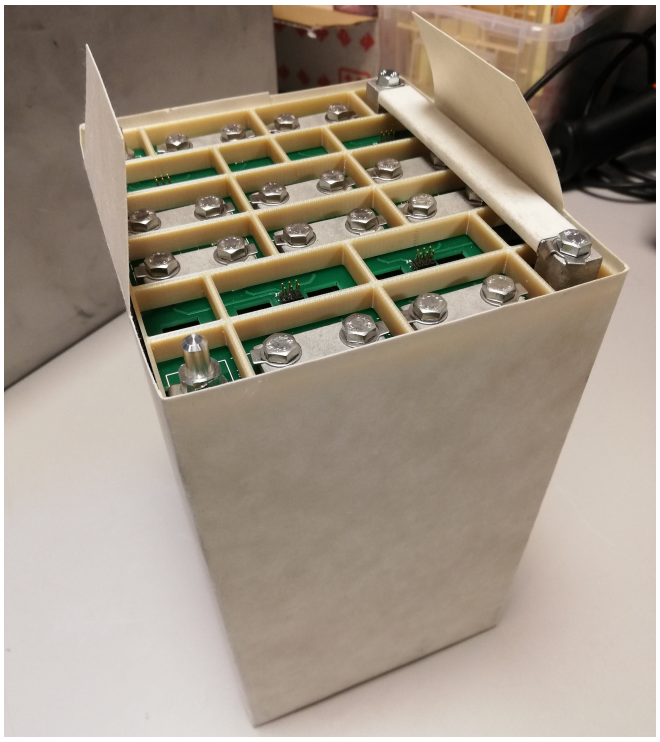
Batterisegmentene

Batterisystemet består av 8 batterisegmenter hvor hver av disse består av 12 cellemoduler fra Energus av typen *Li1x10p25RT* [5]. Batterisegmentene har ikke vært konstruert tidligere og dermed har ingen av cellene blitt testet eller kontrollert. Dette synes også i designet ettersom det blir gjort flere endringer fra det tidligere designet under prosessen for å konstruere batteriet. Blant annet er det gjort store endringer med tanke på sikkerheten i hele batterisystemet.

Batteriets sikkerhet har vært en stor prioritet under hele design- og planleggingsprosessen. Konkurransens regelverk stiller selv store krav om sikkerheten, men for å gjøre det enda enklere og sikkert for arbeid utføres det et par ekstra tiltak. Blant tiltakene pakkes batterisegmentene inn i 0.4 mm Nomex 410 [3], som vist i figur 3.1.2(b) under. Nomex 410 er et slags papir som er laget av både flammehemmede og elektrisk isolerende materiale. I tillegg til å isolere beskytter også innpakningen cellemodulene for eventuelle skader som kan forårsake fysisk skade på cellene. I den samme figuren viser også (a) en skillevegg som designes for å gi god isolering mellom cellepunktene på segmentene. Skilleveggen hindrer kortslutning under montering av cellene til slavemodulene og også ved eventuelle ledende gjenstander som mistes ned i batteriet.



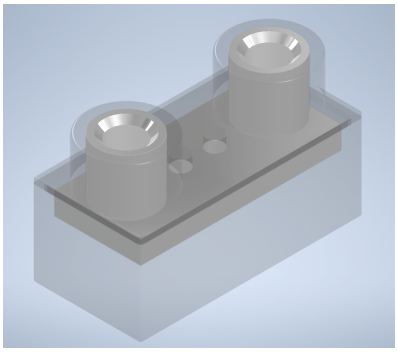
(a) Batterisegmentets skillevegg mellom cellene, 3D-printet i flammehemmende UL94-V-0 materiale.



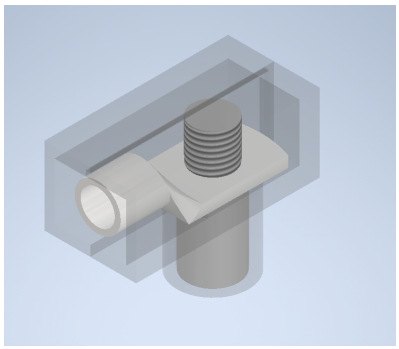
(b) Fullstendig batterisegment med montert skillevegg og innpakket av isoleringsmaterialet 0.4 mm Nomex 410 [3].

Figur 3.1.2: Isolering av batterisegmentene.

For å forbinde alle batterisegmentene elektrisk sammen til hverandre og systemet i bilen brukes det vedlikeholdsplugger. De gjør det enkelt å koble ned spenningen over batteriet ved eventuell vedlikehold, slik at man kan arbeide mer sikkert på batteriet når det er under 50 V som er grensen for høyspenning ifølge EV 4.2.1 i regelverket [6]. Designet fra de tidligere pluggene passer ikke til det nye batterisystemet og redesignes dermed til årets design. Tre forskjellige typer vedlikeholdsplugger har blitt designet og er fremstilt under i figur 3.1.3. Mer detaljerte arbeidstegninger for delene er vedlagt i vedlegg C.2.



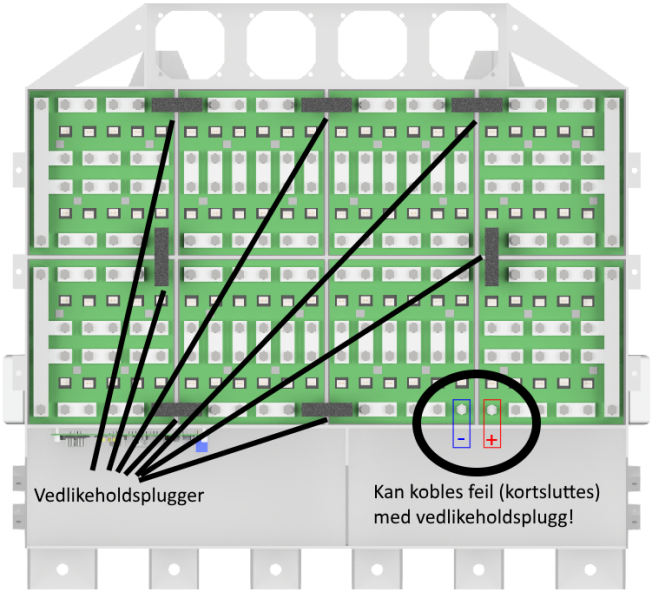
(a) Vedlikeholdsplugg av den vanligste typen som forbinder de ulike batterisegmentene. Typen har to ulike lengder ettersom vertikal og horisontal distanse mellom forbindelse punktene på segmentene er ulike.



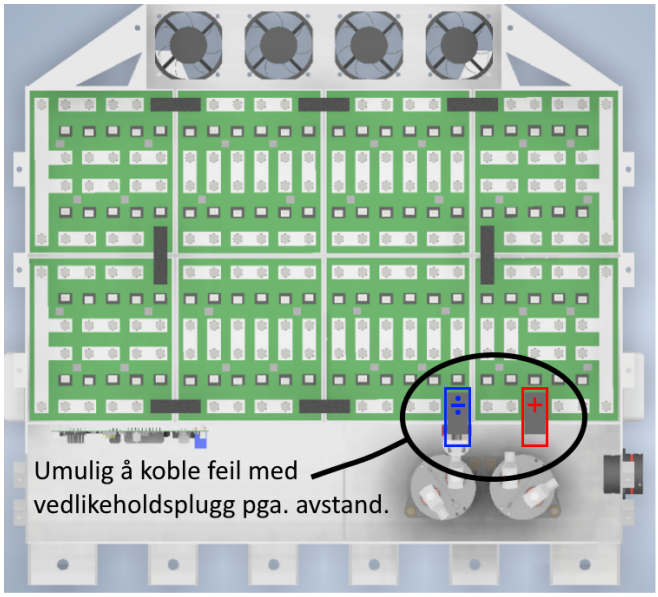
(b) Vedlikeholdsplugg som forbinder batterisegmentenes hovedpoler til kabler med kabelsko som føres ut av batterisystemet.

Figur 3.1.3: Illustrasjon av vedlikeholdspluggene designet i Autodesk Inventor. Det ledende materialet i aluminium er maskinert på UiS og dekselet rundt er 3D-printet i isolasjonsmaterialet ULTEM 9085 [13] også på UiS.

Samtidig for å forsikre at det er ingen mulighet til å koble vedlikeholdspluggene feil på batteriet redesignes cellekonfigurasjonen i batterisegmentene. I tidligere design var det mulig å koble en vedlikeholdsplugg over hovedpolene til batteriet. Med ny ruting i batteriet gjør cellekonfigurasjonen at dette ikke mulig, løsningen er illustrert i figur 3.1.4 under.



(a) Illustrasjon av batterisystemet med vedlikeholdsplugger merket. Sirkelen illustrerer området hvor det i tidligere design var mulig å kortslutte batteriet med vedlikeholdsplugg.



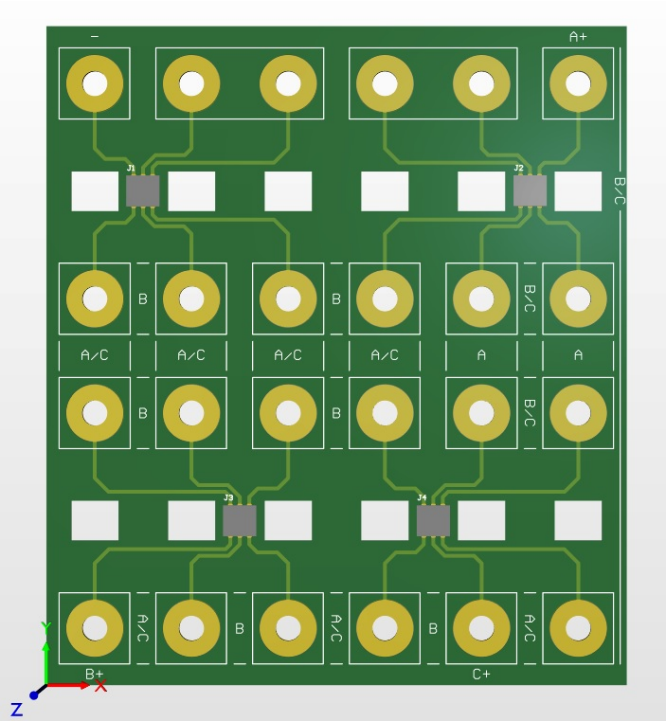
(b) Illustrasjon av batterisystemet med ny cellekonfigurasjon som gjør det umulig å kortslutte batteriet med vedlikeholdsplugg (se svart sirkel).

Figur 3.1.4: Endring av koblingsmønsteret i batteriet. Illustrasjonene er fugleperspektiv tatt fra CAD-filene for batterisystemet laget i Autodesk Inventor.

Celletilkobling

Et av problemene med det tidligere designet av celletilkoblingen i batterisegmentene var muligheten for å kortslutte forbindelseplatene til banene på kretskortet de monteres på. Utlegget av kretskortet var gjort slik for at man skal kunne snu kretskortet for å bruke det samme kretskortet for to forskjellige cellekonfigurasjoner vist i figur 3.2.5. Hadde designet brukt et 4-lags kretskort og banene blitt rutet på mellomlagene kunne dette problemet vært unngått. Men etter innføring av en tredje konfigurasjon av batterisegmentene, er heller ikke dette en gunstig løsning. Dermed ble det valgt å endre konfigureringen til å ha flere plasser å montere sikringene på slavekortet og heller rute flere baner på slavekortet i stedet for på tilkoblingskortet. På tilkoblingskortet i figur 3.1.5 rutes banene med kortest mulig avstand og også rundt forbindelseplatene for å unngå kortsluttningsproblemene. I tillegg forskyves kontaktene til slavekortet for å hindre baklengs montering.

I første omgang var planen å koble temperatursensorene på cellemodulene til kontakter på tilkoblingskortet, men det var ikke plass til dette. Med kontaktene i hull i kretskortet og med korte ledningsstubber til loddepunkter kunne dette fortsatt fungert, men det ville trolig økt muligheten for kortslutninger igjen. Dermed blir kontaktene til temperatursensorene til slavekortet koblet med korte ledninger som loddes på. Dette øker kompleksiteten av monteringen, men virker å være en bedre løsning med henhold til risiko for kortslutning.



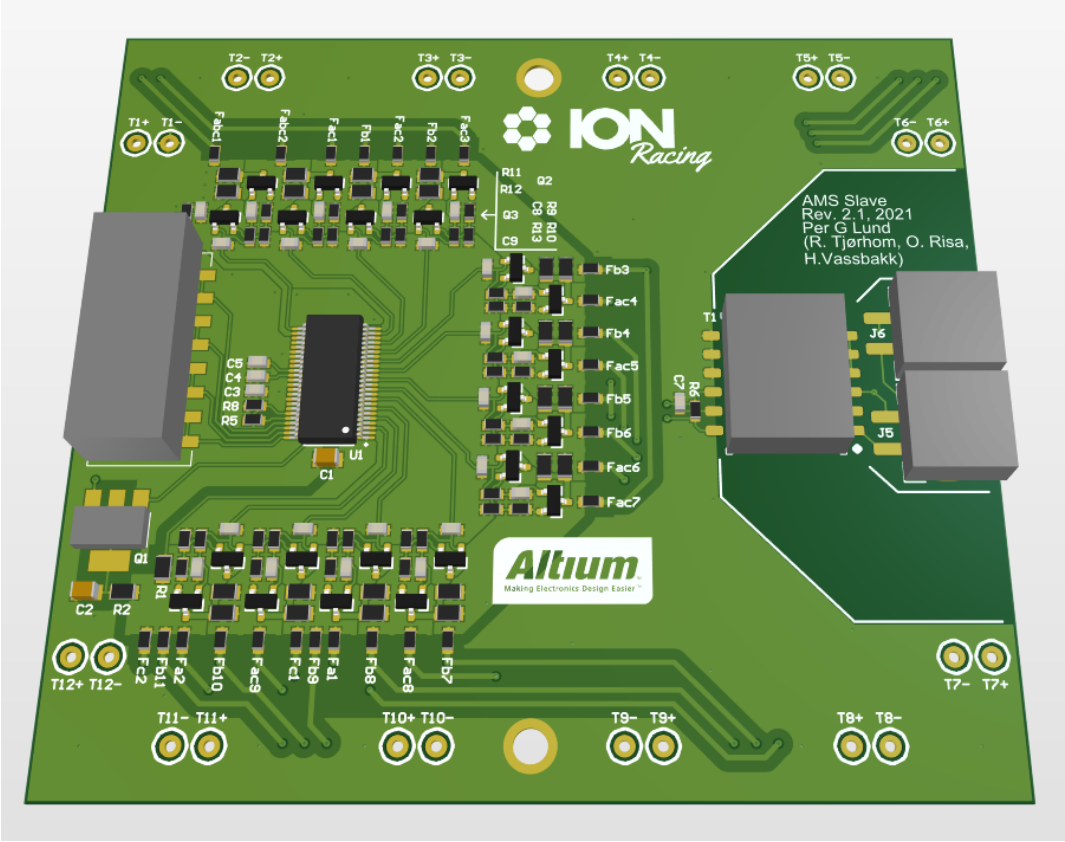
Figur 3.1.5: Tilkoblingskort til slavemodulene. De hvite silketrykkene indikerer plassering av busbarene.

3.2 Slavekort

Slavekortene er kretskort som monteres på batterisegmentene for å måle spenning og temperatur av cellene. Det generelle designet til slavekortene ble utført i den tidligere masteroppgaven [12] i 2020. Dette designet har et par feil og mangler som må rettes og forbedres. Det nye slavekortet som designes fremstilles i figur 3.2.1 under.

Den galvaniske isolasjonen til isoSPI-grensensittet oppfylte ikke kravene i regelverket [6] og endres. Ettersom det er utført endring med at konfigurasjonen av cellene i batterisegmentene ikke gjøres på tilkoblingskortet implementeres dette på slavekortene. Måling av temperatur og spenning er kombineres for å redusere antall batteriovervåkningskretser.

Klarering mellom banene med høye spenninger har blitt økt til IPC-2152 nivå som nevnt under isolasjon i bakgrunnskapittelet 2.8. Flere underdimensjonerte komponenter er byttet ut til å tåle riktig spenning og strøm. Det er også lagt til mulighet for mekanisk festing av slavekortet ettersom det må tåle mye vibrasjon og bevegelser under kjøring.



Figur 3.2.1: 3D-visning av slavekortet i Altium Designer.

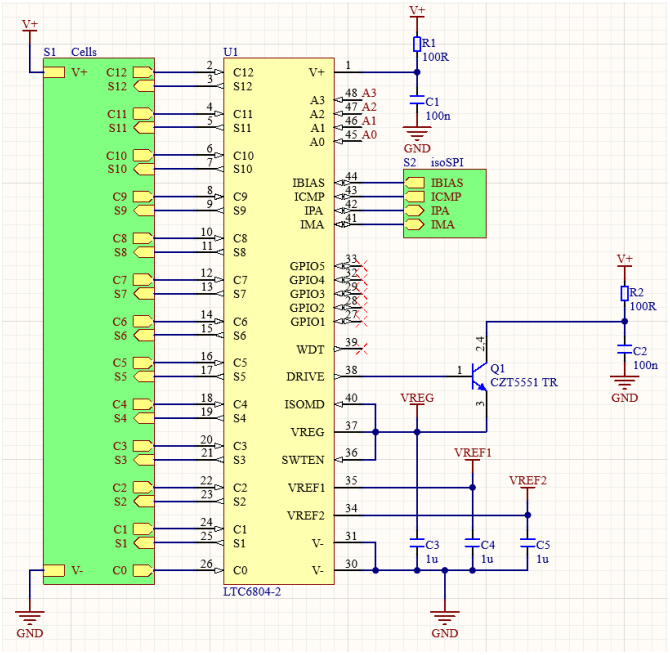
LTC6804

Batterioverv˚akingskretsen som brukes p˚a slavekortene er en LTC6804 [10] fra *Linear Technologies/Analog Devices*. Den kan overv˚ake opp til 12 seriekoblede battericeller med 1.2 mV nøyaktighet og passiv cellebalansering. Den kommer med integrert isoSPI-grensesnitt, som brukes til kommunikasjon med mastermodulen. Dette forenkler designet betraktelig og reduserer størrelsen p˚a kretsene ettersom denne inneholder funksjonalitet for multiplexing, AD-omforming og kommunikasjon, som da ikke trenger ˚a implementeres i tillegg.

Kretsen forsynes direkte fra batteripakken den monteres p˚a. Med dette unng˚as det en isolert strømforsyning utenfra mot et lite strømtrekk fra batterisegmentet. Dette strømtrekket vil i følge databladet være typisk 4 µA i dvalemodus og maks 20 mA når aktiv. Mesteparten av det aktive strømtrekket kommer fra 5 V forsyningsnivået som driver AD-omformeren og isoSPI-grensesnittet. For ˚a regulere batteripakkens spenning til dette spenningsnivået kan det brukes enten en lineær regulator som allerede er inkludert i LTC6804-en eller en ekstern buck-omformer. Ved bruk av en lineær regulator vil effekttapet økes med opp til 10 ganger p˚a grunn av spenningstapet i regulatoren, mens en buck-omformer optimalt vil ha tilnærmet null ekstra tap.

Allikevel tas det i bruk en lineær regulator fordi strømtrekket er nok˚a lavt, en buck-omformer vil ogs˚a ha en betydelig høy hvilestrøm, den vil ogs˚a generere støy som kan p˚avirke m˚alingen av cellespenningen og den vil kreve flere komponenter som m˚a monteres p˚a kretskortet.

Etter oppfordring i databladet brukes det RC-kretser dannet av R1 og C1, og R2 og C2 for ekstra filtrering og avkobling til spenningsforsyningene, disse vises i figur 3.2.2 under. Til disse kondensatorene og motstandene brukes det større 0805 (2 x 1,3 mm) komponenter som skal t˚ale 50 V fra batteripakken og 250 mW effekttap. Under disse komponentene og mellom kobberfyll med store spenningsforskjeller p˚a kretskortet brukes 1 mm klarering for ˚a opprettholde IPC-2152 standarden.

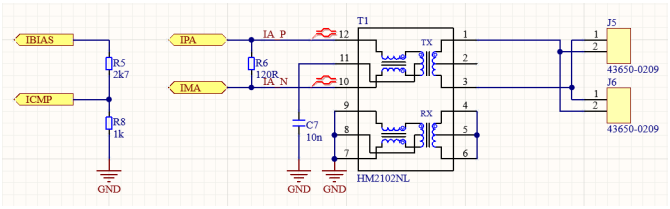


Figur 3.2.2: Skjematikk for tilkobling til LTC6804.

Konfigurasjonen av isoSPI-grensesnittet styres av motstandene R5 og R8 i figur 3.2.3 og 4 brytere koblet til A0 til A3 i figur 3.2.2. Valget av motstandene avgjør amplituden på strømpulsene som sendes over grensesnittet med likning 3.1 og spenningen fra spenningsdeleren avgjør terskelspenningen for mottagelse av signaler som i likning 3.2. Motstandsverdier er valgt for nokså stor signalstyrke og lav terskel ettersom overføringsegenskapene til grensesnittet er ubestemt, men disse kan endres senere etter marginen på signaloverføringen har blitt målt.

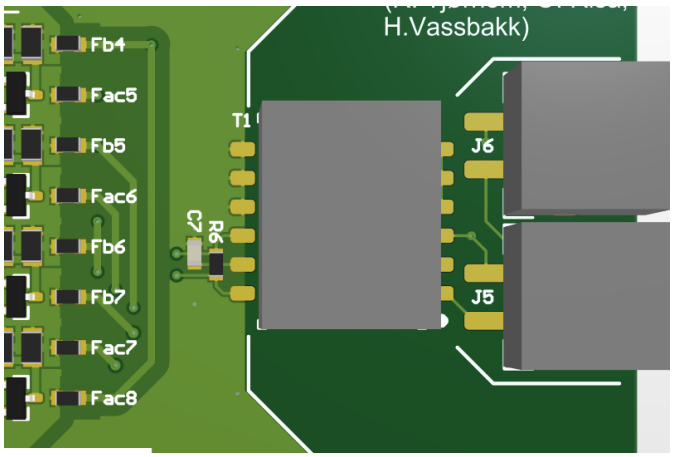
$$I_{DRV} = \frac{20 \cdot 2V}{R_5 \cdot R_8} = \frac{40V}{2.7k\Omega + 1k\Omega} = 10.8mA \tag{3.1}$$

$$V_{TCMP} = \frac{0.5 \cdot 2V \cdot R_8}{R_5 + R_8} = \frac{1V \cdot 1k\Omega}{2.7k\Omega + 1k\Omega} = 270mV \tag{3.2}$$



Figur 3.2.3: Skjematikk for isoSPI-grensesnittet på slavemodulen.

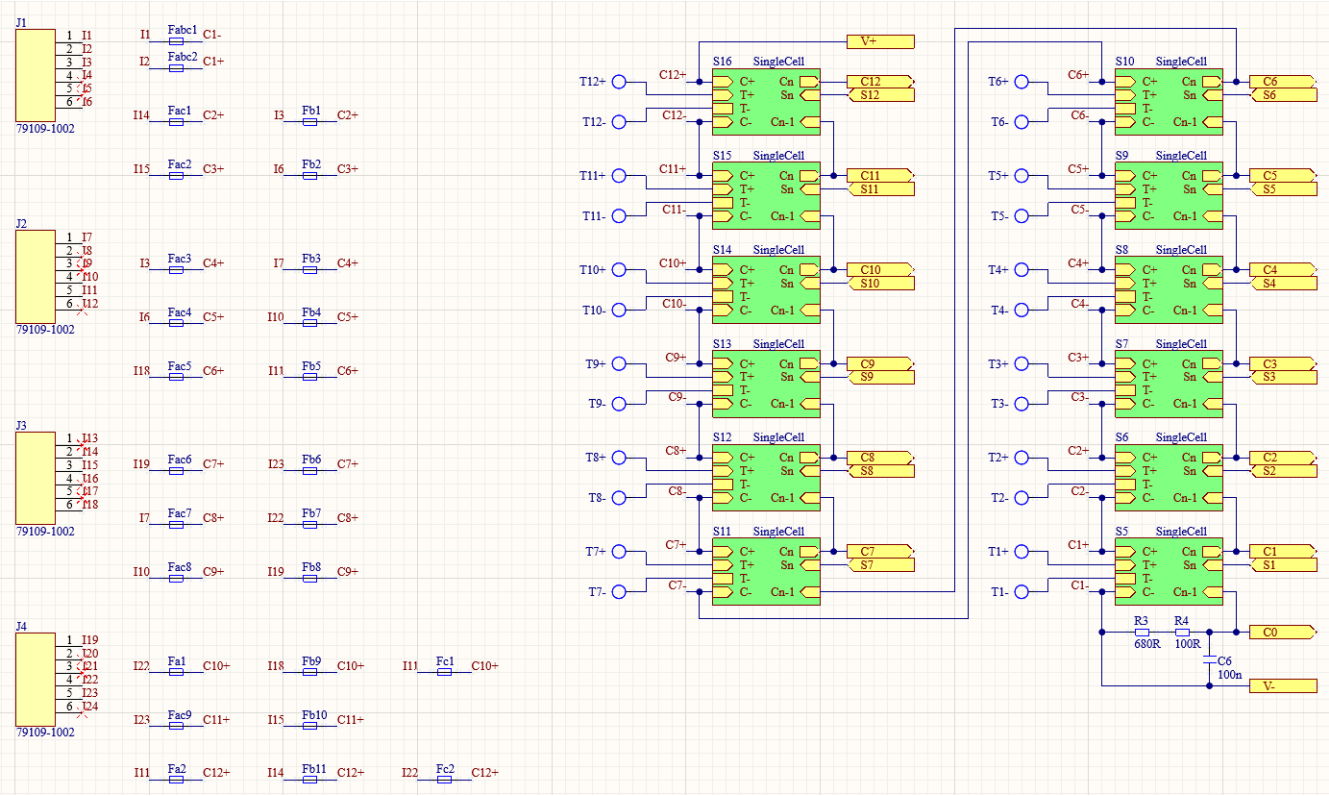
For å oppfylle regel EV 4.3.5 som nevnt i kapittel 2.8 brukes 13 mm klarering under transformatoren og mellom kontaktene og kobberfyllene på kretskortet vist i figur 3.2.4. Dette øker plassbehovet på kretskortet og fører til banene til isoSPI-signalene er rutet under de analoge signalene på kretskortet, men dette kommer sannsynligvis til å ha liten betydning ettersom strømmene er nokså lave og frekvensen er mye høyere enn knekkfrekvensen på filterene.



Figur 3.2.4: Klarering til isoSPI grensesnitt.

Inngangsbeskyttelse

For å beskytte elektronikken og cellene for kortslutning brukes sikringer som loddes på kretskortet. Disse brukes også til konfigurasjon av rekkefølgen på cellene fra celletilkoblingen vist tidligere i figur 3.1.5 og skjema for rekkefølgen vises under i figur 3.2.5. Banene fra celleterminalene før sikringene har 0.5 mm klarering for enkel isolasjon fra hverandre og 1 mm klarering for dobbel isolasjon til andre baner på kretskortet.



Figur 3.2.5: Skjematikk for cellekonfigurasjon på slavekortet.

Måling og balansering

Et differensielt RC-lavpassfilter reduserer transienter som kan skade LTC6804-en. Dette filteret kan ikke forhindre nedfolding på grunn av målefeil som oppstår fra LTC6804-en ved større filterverdier. Det vil ikke være mulig å hindre nedfolding fullstendig, men digital filtrering kan redusere frekvensbåndene som kan forårsake det. Bruken av systemet vil heller ikke forårsake store frekvenser i dette frekvensområdet ettersom motorkontrolleren begrenser frekvensen til strømtrekket til lavere frekvenser utenom selve inverterfrekvensen som er på 10 kHz og blir dempet av filteret.

Når cellespenningene måles vil filteret i følge likning 3.3 ha en knekkfrekvens på 2 kHz. En lavere knekkfrekvens vil være gunstig for å unngå aliasing og for enda bedre støyreduksjon, men databladet for LTC6804 [10] fraråder bruk av større motstand og kapasitans ettersom dette innfører systematiske målefeil.

$$f_{c,RCv} = \frac{1}{2\pi(R_9 + R_{10})C_9} = \frac{1}{2\pi \cdot (680\Omega + 100\Omega) \cdot 100nF} = 2.04\ kHz \tag{3.3}$$

For å redusere antall LTC6804-er på kretskortene kombineres spenning- og temperaturmålingene. Dette gjøres ved å parallellkoble temperatursensoren med spenningsmålingen gjennom transistoren Q3 som styres av balanseringsutgangen fra LT6804-en i figur 3.2.6. Når Q3 er deaktivert vil utgangsspenningen være lik cellespenningen. Når den er aktivert fungerer temperatursensoren som en variabel zenerdiode som gir et spenningssignal avhengig av temperaturen. Når temperaturen måles vil ikke R9 inkluderes i filteret og det vil da ifølge likning 3.4 ha en knekkfrekvens på 16 kHz. Den raskere responsen er valgt for å kunne gjøre raskere temperaturmålinger ettersom signalet ikke trenger å være like nøyaktig som cellespenningen.

En alternativ metode for temperaturmåling som kan vurderes er å multiplexe temperatursensorene til GPIO pinnene på LTC6804-en, men dette velges ikke her på grunn av at kun 3 målinger kan gjøres samtidig og det ville redusert samplingfrekvensen betraktelig.

$$f_{c,RCt} = \frac{1}{2\pi R_{10} C_9} = \frac{1}{2\pi \cdot 100\Omega \cdot 100nF} = 15.82\ kHz \tag{3.4}$$

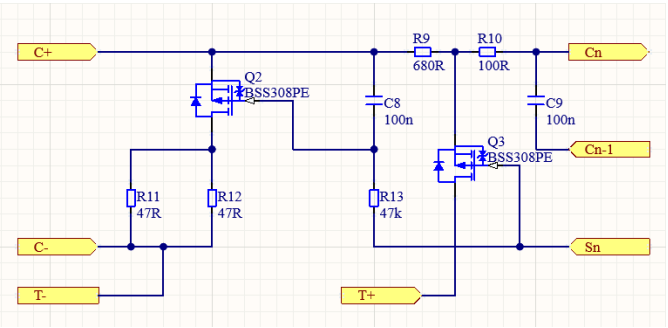
Til balansering av cellene brukes eksterne transistorer og motstander på grunn av effektbegrensningene i LTC6804-ene. Balanseringsutgangen fra LTC6804-en er koblet til *gaten* på transistoren gjennom en RC-krets for å hindre aktivering ved temperaturmåling. Det er usikkert hvor mye det vil påvirke batteritiden å aktivere balanseringsresistorene når temperaturene måles ettersom balanseringsstrømmen er nokså lav og temperaturmålingene skjer i korte pulser. For å hindre at transistorene aktiveres ved en 1 ms puls regnes minimum motstand i likning 3.5 ut. Kondensatoren C8 velges til å være 100 nF for å være lik C9.

$$R_{13,min} = -\frac{t_{RC}}{C_8 \ln(1 - \frac{V_{GS,th}}{V_{cell}})} = -\frac{1ms}{100nF \cdot \ln(1 - \frac{1V}{4.2V})} = 36.8k\ \Omega \tag{3.5}$$

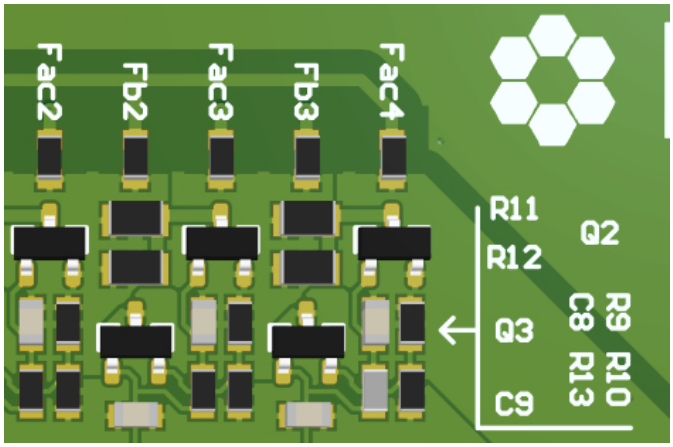
Cellebalansering forventes ikke å være en regelmessig handling, det ble derfor valgt en nokså svak balanseringskrets fordi dette reduserer størrelsen på komponentene som kreves for å håndtere den reduserte effekten. Ved å parallellkoble to 0805 (2 x 1,3 mm) motstander som skal tåle opp til 0.5 W, med 25 % derating blir motstanden for å oppnå denne effekten 47 Ω i likning 3.6. Dette gir den følgende balanseringsstrømmen i likning 3.7.

$$R_{min} = \frac{V_{max}^2}{P_{max} \cdot derating} = \frac{(4.2V)^2}{0.5W \cdot 0.75} = 47.04\Omega \tag{3.6}$$

$$I_{b,max} = 2 \cdot \frac{V_{max}}{R} = 2 \cdot \frac{4.2V}{47\Omega} = 178.7mA \tag{3.7}$$



Figur 3.2.6: Skjematikk for kretsene til individuelle celler på slavekortet.

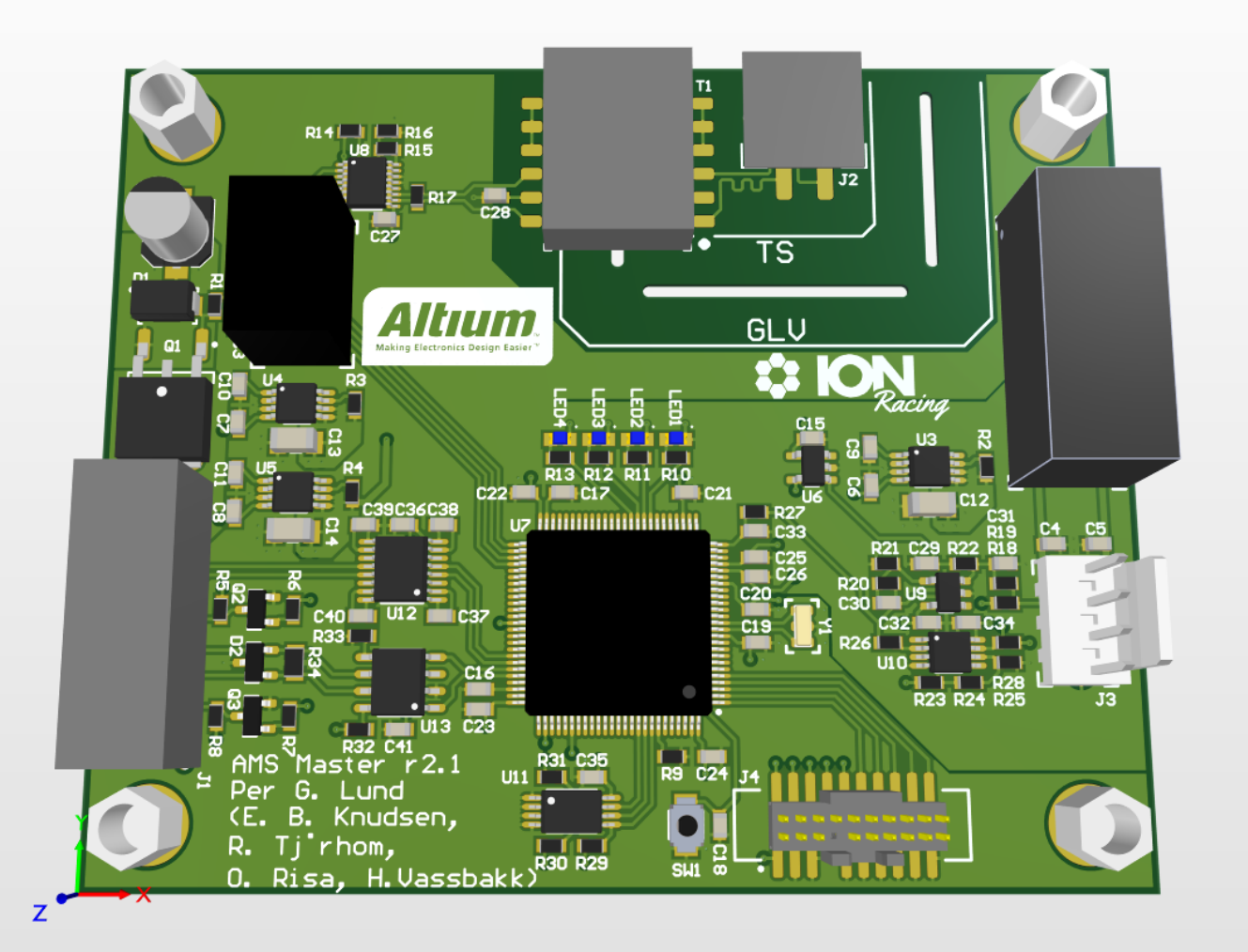


Figur 3.2.7: Kretskortutlegg for kretsene i figur 3.2.6.

3.3 Masterkortet

Masterkortet består i hovedsak av mikrokontrolleren som styrer slavene i batteriovervåkingssystemet, måler strømmen og kommuniserer med andre systemer i bilen. Blant annet over CANbus kommuniserer den med bilens sentral styringenhet (Electronic Control Unit, ECU) og laderen som lader batteriet. Den kommuniserer også over RS232 med datamaskin.

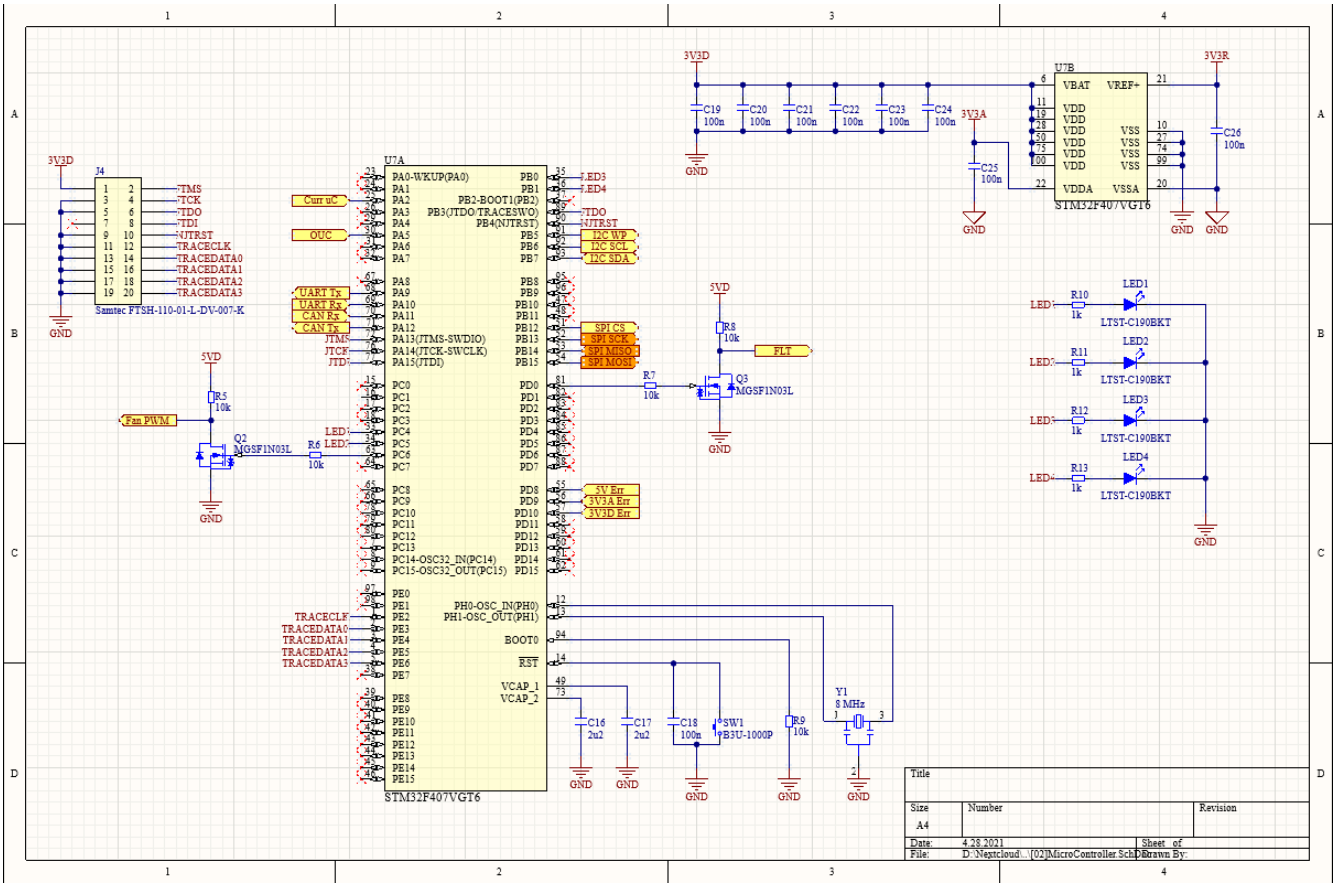
Likt som de andre kretskortene hadde masterkortet et generelt design i 2020 [12] som også hadde feil og mangler. Flere baner var koblet feil, og komponentplasseringene var lite gunstige. Likt som ble nevnt for slavekortene, hadde ikke isoSPI-grensesnittet tilstrekkelig isolasjon og klarering. Redesignet ble også tatt i bruk til å redusere kompleksiteten og størrelsen på kretskortet drastisk, samt å forsøke å forbedre signalintegritet og støymotstand for kretsene.



Figur 3.3.1: Kretskortet til mastermodulen i Altium Designer.

Mikrokontroller

Mikrokontrolleren som brukes til å styre systemet er en 32-bit ARM mikrokontroller av typen STM32F407VTG som erfaringsmessig egner seg godt til bruksområdet og har god margin på prosessorhastighet og minne. Tilkoblingsskjema til mikrokontrolleren vises under i figur 3.3.2 og viser de brukte pinnene og hva de går til.



Figur 3.3.2: Skjematikk for tilkobling til mikrokontrolleren.

Integrerte moduler for kommunikasjonsgrensesnitt er koblet til transceivere på kretskortet som forsterker signalene ut fra kretskoret. Mikrokontrolleren styrer slavene gjennom et SPI-grensesnitt fra pinne B12 til B15 koblet til en LTC6820 isoSPI transceiver. For tilkobling til CAN- og RS232-grensesnittene er pinnene A9 til A12 koblet til transceivere som driver signalene med spenningsnivåene som kreves.

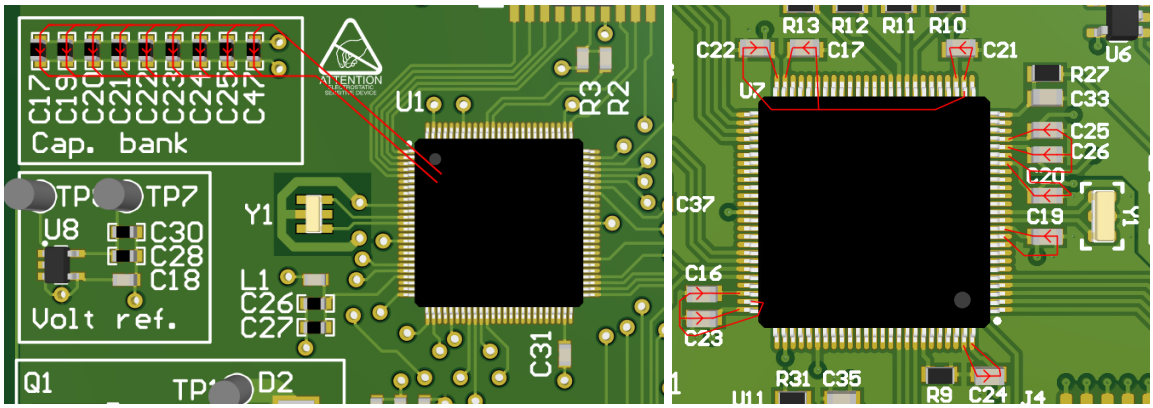
Masteren signaliserer tilstanden til batterisystemet til sikkerhetskretsene som styrer reléene på polene på batteriet. Hvis en feil oppdages av batteriovervåkingssystemet deaktiveres dette signalet og lasten kobles fra batteriet. På mikrokontrolleren kommer dette signalet fra pinne D0 til transistoren Q3, som ved et aktivt feilsignal vil deaktivere utgangen til sikkerhetskretsen.

For å måle signalet fra strømsensoren er strømsignalet koblet gjennom en forsterkerkrets til pinne PA2 på mikrokontrolleren som er konfigurert for AD-konversjon med en intern 12-bit AD-omformer.

En EEPROM-krets er koblet til mikrokontrolleren med et I2C-grensesnitt på pinne B5 til B7. Denne skal kunne brukes til lagring av konfigurasjonsdata til batteriovervåkingssystemet.

Til styring av viftene i kjølesystemet brukes et PWM signal. Dette genereres fra en intern timer-modul på pinne C6, som er koblet til transistor Q2 som driver signalet med 5 V spenningsnivåutgangen til viftene.

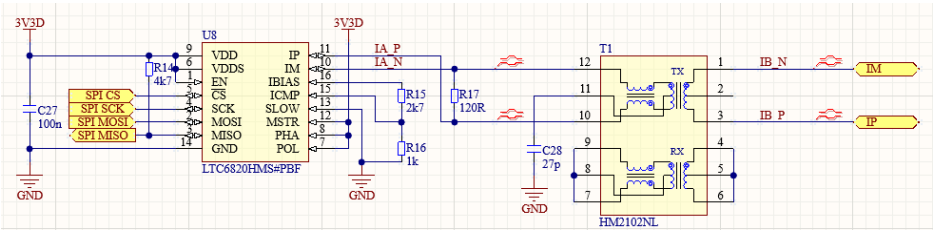
Avkoblingskondensatorene var tidligere plassert i en gruppe borte fra mikrokontrolleren. Dette er ikke optimal plassering for best avkobling og kortest strømsløyfe som forklart i kapittel 2.7 om kretskortutlegg. I det nye utlegget er avkoblingskondensatorene plassert så nærme som mulig forsyningspinnene på mikrokontrolleren for å forbedre dette, figur 3.3.3 under illustrerer dette.



Figur 3.3.3: Plassering av avkoblingskondensatorer til mikrokontrolleren i gammelt og nytt kretskortutlegg, lederbanen er fremvist i rødt.

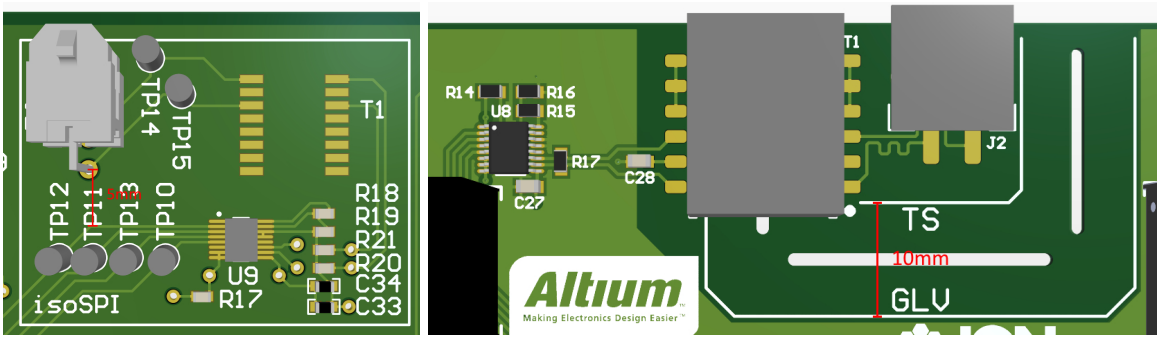
IsoSPI

En LTC6820 brukes til å konvertere SPI-grensesnittet fra mikrokontrolleren til isoSPI som går til slavemodulene. IsoSPI-grensesnittet skal operere ved høye frekvenser og relativt sterke strømpulser. Derfor er LT6820-en plassert borte fra andre kretser som er følsomme for støy. Skjematikken vises i figur 3.2.3 under og viser blant annet MISO-signalet fra LT6804-en som er drevet av en open drain transistor inne i chip-en og må ha en ekstern motstand til å drive signalet høyt. Mikrokontrollerens pinner kan konfigureres med pull-up eller pull-down motstander, men disse har typisk en motstand rundt 40 $k\Omega$ og kan ikke drive linjen ved høye frekvenser. For å ha rask nok stigning til å operere opp til 10 MHz, brukes det en pull-up motstand R14 på 4.7 $k\Omega$. R15 og R16 konfigurerer amplituden på strømpulsene og grensenivået for signalene, disse er konfigurert likt som på slavemodulene.



Figur 3.3.4: Skjematikk for isoSPI.

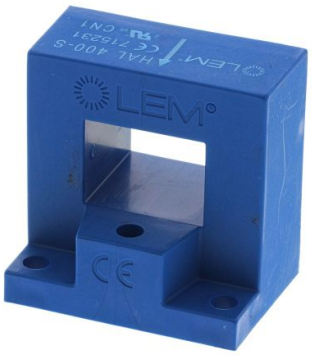
Det tidligere kretskortutlegget hadde ikke skille til isolerte signaler og oppfylte ikke regel EV 4.3.5 [6] om isolasjonsskille mellom kretsene. I det nye utlegget er dette endret med 10 mm skille og isolasjonsspalter i kretskortet, endring i avstand vises i figur 3.2.4 under. Transformatoren er flyttet litt lenger bort fra LT6820-en for å følge anbefaling i databladet [10] om minimum 1 cm avstand. Kobberplanene er fjernet under transformatoren for å unngå eddystrømmer som dannes av magnetfeltene.



Figur 3.3.5: Gammelt og nytt kretskortutlegg for isoSPI, rød merking viser den nye avstanden.

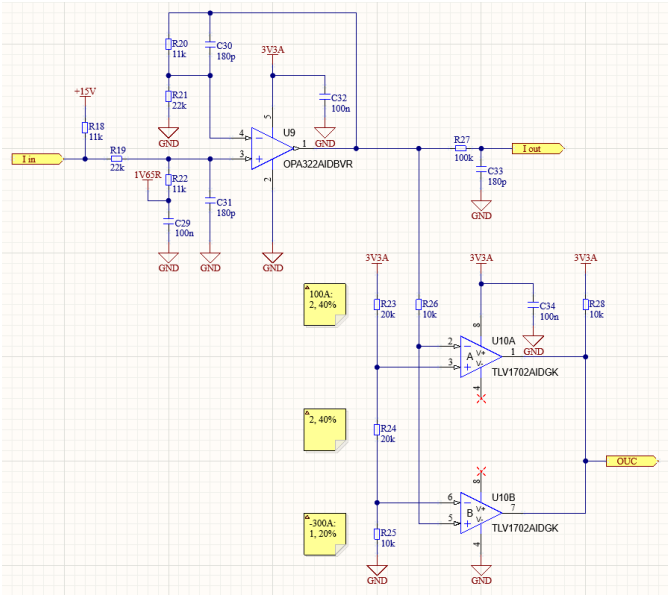
Strømmåling

Ettersom batterisegmentene er seiekoblet trenger bare strømmen måles på ett punkt. For å unngå å øke kompleksiteten til slavekortene er det naturlig at dette skjer på masterkortet. Strømsensoren som brukes er en LEM HAL400-S, som allerede blir brukt andre systemer i bilen som motorkontrolleren og BSPD-en og er et naturlig valg å gjenbruke. Strømsensoren vises i figur 3.3.6 under og har et måleområde på $I_{PN} = \pm 400\ A$. Utgangssignalet er et analogt spenningssignal med funksjonen $V_{i,out}(t) = 4V \cdot \frac{I(t)}{I_{PN}}$, med maksimum ± 1 % avvik i det nominelle måleområdet og $\pm 10\ mV$ offset. Strømsensoren trenger forsyningsspenning på $\pm 15\ V$ som genereres av en DCDC-omformer som nevnes senere i kapittel 3.3.



Figur 3.3.6: LEM HAL400-S strømsensor.

Strømsensoren sender et analogt signal med utslagsverdi $V_{i,out} = \pm 4\ V$. For å konvertere signalet til et spenningsnivå som mikrokontrolleren kan lese brukes en operasjonsforsterker med ikke-inverterende tilbakekobling. Forsterkeren som brukes er en OPA322, som også er brukt på tidligere utlegg. Skjematikken av sensorens forsterkerkrets er vist under i figur 3.3.7.



Figur 3.3.7: Skjematikk for analog forsterkerkrets

For at spenningen fra sensoren skal konverteres til AD-omformerens måleområde på 0 V til 3.3 V forsterkes spenningen med $\frac{1}{3}$ og forskyves med 1.65 V. Dette gir signalområdet for AD-omformeren på 0.32 V til 2.98 V i likning 3.8. For å oppnå denne forsterkningen og forskyvningen brukes spenningsdelerene dannet av R20 og R21 og R19 og R22.

$$V_{ADC} \in V_{in} \cdot G + V_{offset} = [-4, +4]\ V \cdot \frac{1}{3} + 1.65\ V = [0.32, 2.98]\ V \tag{3.8}$$

Kondensatoren C31 er lagt til før inngangen til forsterkeren og danner et RC-lavpassfilter med R19 for forhåndsfiltrering. Dette filteret vil ha knekkfrekvensen i likning 3.9 og kan sees som H_RC1 i Bode-diagrammet under i figur 3.3.8.

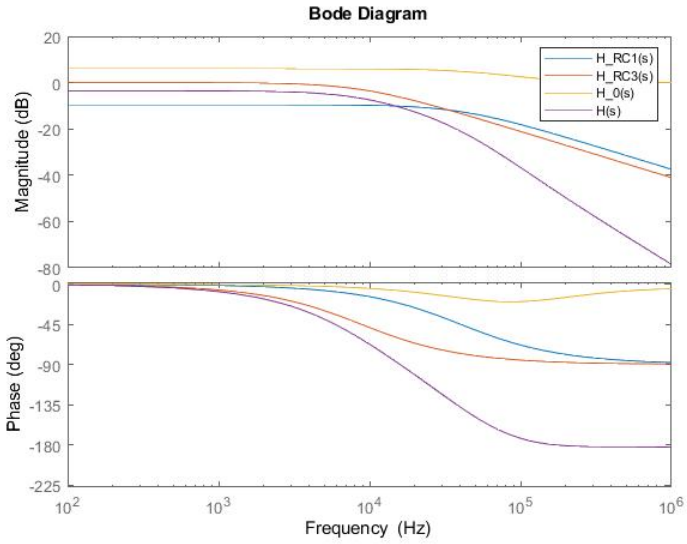
$$f_{c,RC1} = \frac{1}{2\pi R_{22} C_{31}} = \frac{1}{2\pi \cdot 22\ k\Omega \cdot 180\ pF} = 40.19\ kHz \tag{3.9}$$

I tilbakekoblingen til forsterkeren hindrer RC-filteret dannet av R20 og C30 faseforskyvningen å nå 180 °slik at oscillasjoner ikke kan oppstå og forsterkerkretsen er stabil ved alle frekvenser. Det kan sees som H_0 i figur 3.3.8 under.

For å hindre nedfolding ved sampling og ytterligere filtrere støy introdusert etter forsterkeren brukes et RC-lavpassfilter plassert nærme AD-omformeren på mikrokontrolleren. Dette filteret dannes av R27 og C33 og vil ha knekkfrekvensen i likning 3.10 og frekvensresponsen merket H_RC3 i figur 3.3.8.

En simulering av forsterkerkretsens frekvensrespons ble utført i skriptet inkludert i vedlegg B.1 og kan vises i figur 3.3.8.

$$f_{c,RC3} = \frac{1}{2\pi R_{27} C_{33}} = \frac{1}{2\pi \cdot 100\ k\Omega \cdot 180\ pF = 8.842\ kHz} \tag{3.10}$$

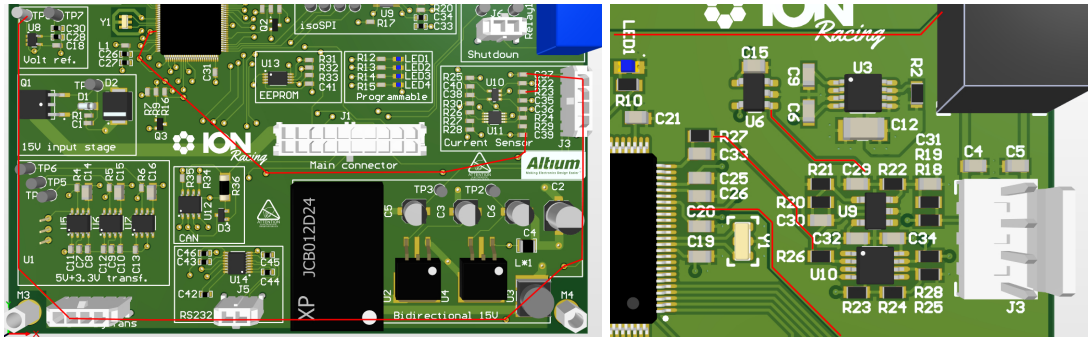


Figur 3.3.8: Simulering av forsterkerkretsens frekvensrespons i MATLAB. Skriptet kan finnes i vedlegg B.1

Etter konkurransens regel T 11.9 [6] må det være mulig å oppdage frakobling av strømsensoren. For å oppdage dette er R18 koblet til inngangen og vil dra signalet høyt hvis sensoren kobles fra. Signalet vil da tilsvare en strøm på 308 A og aktivere overstrømsbeskyttelsen til overvåkingssystemet.

Vinduskomparatoren dannet av U10 kan brukes til et redundant overstrømsvern, men er ikke nødvendig og blir ikke brukt.

Som vist på kretskortet i figur 3.3.9 plasseres de analoge komponentene tett sammen, og med korte baner til inngangen mikrokontrolleren, spenningsregulator og spenningsreferanse for å forbedre støymotstand. Dette har vært en tydelig mangel på den tidligere versjonen. Jord- og forsyningsplanene over og under det analoge området skilles også for å hindre innvirkning fra strømmen til DCDC-omformeren til strømsensoren og de høyfrekvente digitale signalene til mikrokontrolleren.

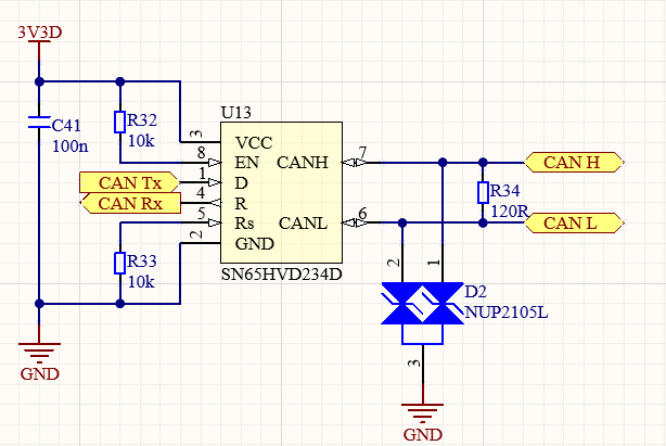


Figur 3.3.9: Eldre og nytt kretskortutlegg for analogt strømsignal merket i rødt.

CAN

Signalene fra mikrokontrollerens CAN-modul er ikke direkte kompatibel med CANbus-protokollen og må konverteres med en transceiver. Kretsen som brukes til dette er en SN65HVD232 og tilkoblingen vises i figur 3.3.10 under. Denne kretsen implementerer all funksjonalitet som kreves for mikrokontrolleren å koble til CANbussen. Det kreves ingen eksterne komponenter, men det er anbefalt med avkoblingskondensatoren C41 og TVS-dioden D2 til beskyttelse mot høyere spenningstransienter. Det er også lagt til fotavtrykk til resistorer for konfigurasjon av andre varianter fra SN65HVD23X-serien.

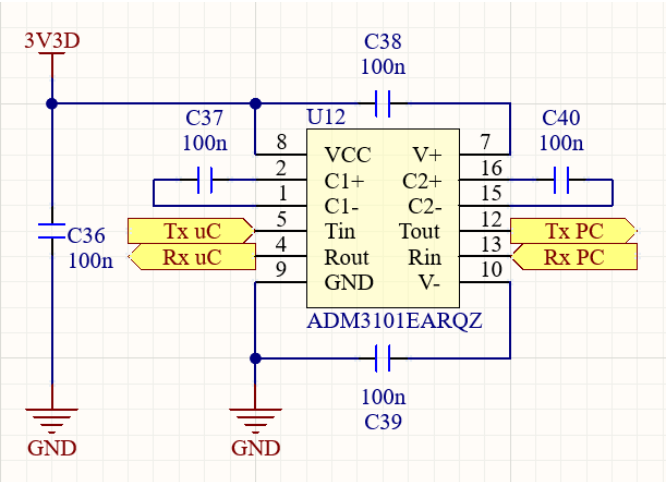
Transceiverkretsen er plassert nærme konnektoren for å redusere støy fra CAN-bussen og minimere avstanden fra kontakten til inngangsbeskyttelsen. Dette kan sees i tidligere figur 3.3.1.



Figur 3.3.10: Skjematikk for CAN-transceiver på masterkortet.

RS232

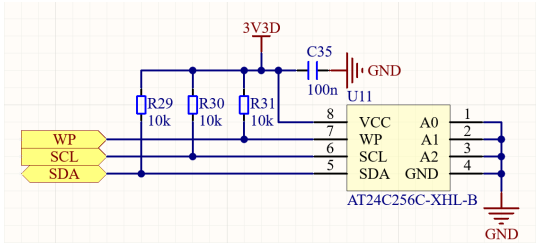
Som CANbus-signalene er heller ikke RS232-signalene fra mikrokontrolleren kompatible med RS232-protokollen. Her brukes en ADM3101 transceiverkrets. Denne implementerer også de fleste funksjonene som kreves, men må ha de eksterne kondensatorene C37, C38, C39 og C40 i figur 3.3.11 til en intern charge-pump spenningsforsyning som genererer spenningsnivåene som brukes til RS232. Denne kretsen er også plassert nærme konnektoren av samme grunn som CAN-transceiveren i kapittel 3.3.



Figur 3.3.11: Skjematikk for RS232 transceiver på masterkortet.

EEPROM

Flash-minnet i mikrokontrolleren er ikke særlig egnet til lagring av data som ofte overskrives. Dette er på grunn av kort levetid på 10 000 skrivesykluser og at man må overskrive hele sektorer på 16 kB om gangen. For å unngå dette brukes en AT24C256 EEPROM-krets med 32 kB lagringsplass og tilkoblings skjematikk er vist under i figur 3.3.12. Denne kretsen har levetid på 1 000 000 skrivesykluser, skrivebeskyttelse og kan man skrive et vilkårlig antall byte om gangen. Mikrokontrolleren kommuniserer med denne over et I2C-grensesnitt.



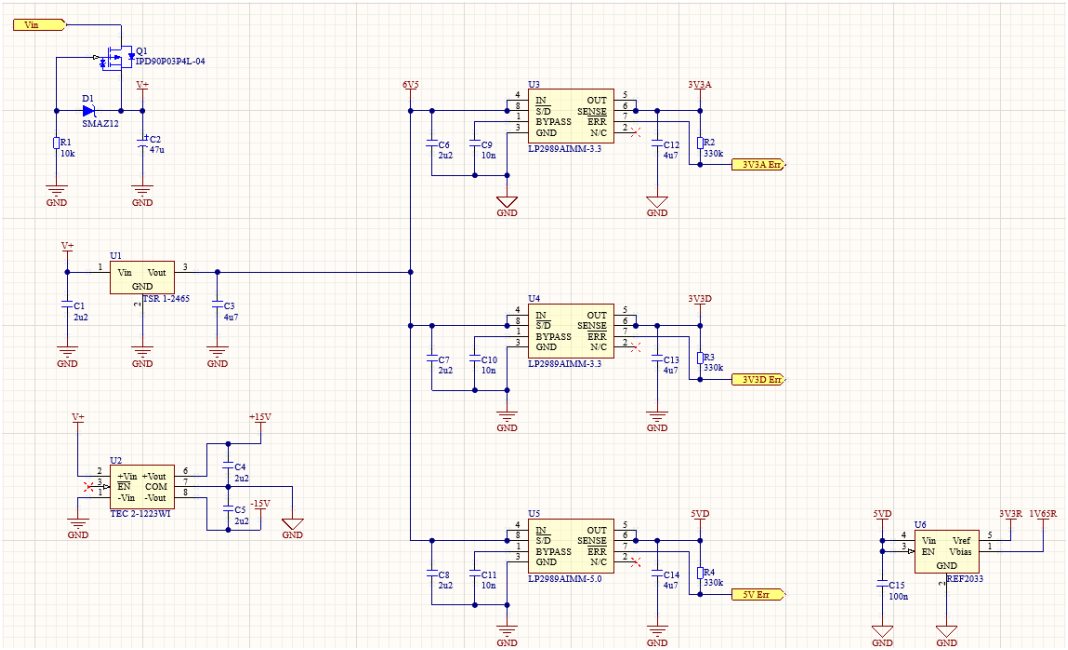
Figur 3.3.12: Skjematikk for EEPROM-krets på masterkortet.

Spenningsforsyning

Valg av spenningsregulatorer gjøres hovedsaklig med grunnlag i kretser som kan gjenbrukes fra tidligere design. En endring som gjøres er på forsyningskretsen til strømsensoren. Det tidligere designet brukte en DCDC-omformer og etterregulering med lineære spenningsregulatorer. Disse kretsene var rimelig store og tok mye plass på det gamle kretskortet. Med endring til en nyere, mer kompakt variant DCDC-omformer som har mindre krav til filtrering, kan de lineære regulatorene fjernes og mye areal blir spart. Skjematikken for de endelige strømforsyningskretsene vises under i figur 3.3.13.

Annet enn fjerningen av de lineære spenningsregulatorene er de viktigste endringene flyttingen av forsyningskretsene næremere forbrukerkretsene og flytting av kretsen for inngangsbeskyttelse nærmere konnektoren.

Etter design av kretskortet ble det innsett at en annen variant av den brukte spenningsomformeren TSR 1-2465 kunne vært tatt i bruk, ettersom den hadde en utgangsspenning på 5 V i stedet for 6.5 V. Hvis denne endres kan U5 fjernes for å redusere antall komponenter og redusere tap fra spenningsregulering.



Figur 3.3.13: Skjematikk for strømforsyningskretser.

Kapittel 4

Programvare

Programvare for systemet deles inn i to hoveddeler, et C-prosjekt for mikrokontrolleren på mastermodulen som kjører dens oppgaver og en datamaskin som kontinuerlig kjører et Python-script for fremvisning gjennom et grafisk brukergrensesnitt.
Mastermodulens oppgaver er:

- Styring av slavene
  Kommunisere med slavene over isoSPI og sende kommandoer for måling av temperatur og cellespenning og aktivere cellebalansering.
- Behandling av data fra slavene
  Omformatere og filtrere dataverdiene til videre bruk.
- Beskytting av batteri
  Beskytte batterisystemet ved å koble fra last utenfor sikre betingelser.
- Kommunikasjon med andre systemer
  Kommunisere ECU, motorkontroller og lader over CANbus og datamaskin over RS232.
- Styring av lader
  Kontrollere strøm og spenning fra laderen og balansere battericeller.

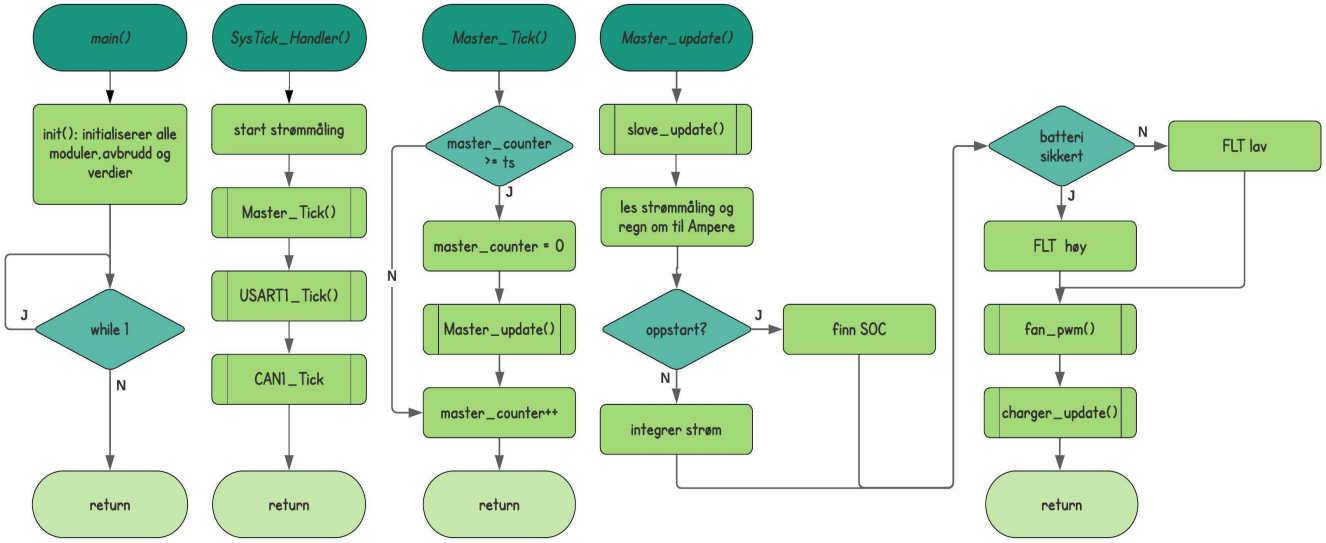
Det ble valgt å hovedsaklig bruke avbruddsbasert programflyt ettersom dette prosjektet krever rask utføring av asynkrone oppgaver. Subrutiner kunne vært brukt, men det ville resultert i mange blokkerende funksjoner som ville hindret effektiv prosessorutnyttelse. Et sanntids operativsystem kunne også vært brukt, men det ville økt kompleksiteten uten særlig vinning for dette prosjektet. Hvis andre oppgaver som ikke er like tidskritiske skulle implementeres hadde det muligens vært gunstig å implementere disse som subrutiner som kjører kontinuerlig, men blir avbrutt av mer tidskritiske oppgaver.

Koden til mastermodulen er implementert i STM32CubeIDE som er kort beskrevet i bakgrunns kapittelet 2.9. Det er valgt å implementere direkte i C-kode for å ha mer oversikt og kontroll over programflyten og derfor kunne optimalisere koden ved bruk av avbrudd og DMA-overføringer fra starten.

Et annet kodeverktøy som har vært vurdert er Simulink Coder som genererer C-kode til mikrokontrolleren fra Simulink blokkskjema. Den genererte koden konstrueres av kodeblokker som må modifiseres kraftig for å være gunstige til bruk i ikke-blokkerende avbruddsfunksjoner, men kunne vært brukt til Subrutiner eller sanntids operativsystem.

Overordnet programflyt

Den overordnede programflyten er fremvist under i figur 4.0.1. Ved oppstart initialiseres alle funksjoner og variabler. Etter dette kjøres en uendelig løkke som blir avbrutt av avbruddsfunksjoner. *SysTick*-avbruddet kjøres hvert ms og kjører funksjonene for overvåking og kommunikasjon med intervaller som kan konfigureres.



Figur 4.0.1: Flytdiagram for overordnet programflyt.

4.1 Datastrukturer

For å holde orden på og enkelt kunne behandle data brukes datastrukturer. En datastruktur er en forhåndsdefinert rekke med ulike dataverdier som plasseres i faste adresser i forhold til starten på strukturen i minnet. I datastrukturene som er laget samles verdier som brukes av flere funksjoner og som skal være tilgjengelige for overføring til brukergrensesnittet på datamaskinen.

Hovedstruktur

Masterstrukturen som vises under i figur 4.1.1 er hovedstrukturen som inneholder alle strukturene og verdiene som blir brukt til den overordnede funksjonaliteten i mastermodulen. Den inneholder vektorene med spenning, temperatur og ladestatus for alle battericellene og minimum, maksimum og gjennomsnitt av dise, samt noen vektorer til funksjonalitet som ikke er implementert. Den inneholder også strøm og overordnet ladestatus for batteriet.

Expression	Type	Value
master	master_struct	{...}
conf	config_struct	{...}
status	status_struct	{...}
charger	charger_struct	{...}
n_slaves	uint8_t	1 '\001'
slave	slave_struct [8]	0x20000764 <master+444>
i_fs	int32_t	-5
i_fc	int32_t	-4
i_fc_hp	int32_t	-1
v	uint16_t [96]	0x20000d10 <master+1896>
ocv	uint16_t [96]	0x20000dd0 <master+2088>
v_min	uint16_t	40594
v_max	uint16_t	40794
v_avg	uint16_t	40725
v_min_id	uint8_t	6 '\006'
v_max_id	uint8_t	1 '\001'
v_tot	uint16_t	4887
t	int8_t [96]	0x20000e9a <master+2290>
t_min	int8_t	-8 'ø'
t_max	int8_t	7 '\a'
t_avg	int8_t	-2 'þ'
t_min_id	uint8_t	1 '\001'
t_max_id	uint8_t	5 '\005'
dod	uint32_t	24079
c	uint16_t [96]	0x20000f04 <master+2396>
c_min	uint16_t	2500
c_max	uint16_t	0
c_avg	uint16_t	0
c_min_id	uint16_t	0
c_max_id	uint16_t	0
r_fs	uint16_t [96]	0x20000fce <master+2598>
r_fc	uint16_t [96]	0x2000108e <master+2790>
soc	uint16_t	9991

Figur 4.1.1: Oppbygging av masterstrukturen.

Slaver

Slavestrukturen som vises under i figur 4.1.2 brukes til å holde orden på konfigurasjon og dataverdier fra en slave og gjenspeiler i stor del registerene i LTC6804-ene. Når dataverdiene fra slaven blir mottatt lagres de her til signalbehandlingen utføres. Statusverdien indikerer om slavemodulen er aktiv eller om den er frakoblet. En vektor med tidsstempel indikerer når verdiene i vektoren sist var oppdatert.

Expression	Type	Value
master.slave[0]	slave_struct	{...}
status	uint8_t	1 '\001'
time	uint32_t [18]	0x20000768 <master+448>
cfgr	uint8_t [6]	0x200007b0 <master+520>
cvr	uint16_t [12]	0x200007b6 <master+526>
avr	uint16_t [3]	0x200007ce <master+550>
str	uint16_t [6]	0x200007d4 <master+556>
com	uint16_t [3]	0x200007e0 <master+568>
temp	uint16_t [24]	0x200007e6 <master+574>

Figur 4.1.2: Oppbygging av slavestrukturen.

Konfigurasjon

I konfigurasjonsregisteret som vises under i figur 4.1.3 samles alle verdier som skal konfigureres i batterisystemet, dette gjør at endring av konfigurasjonsdata kan gjøres ved å overskrive denne strukturen. Verdiene i denne strukturen bestemmer oppførselen til forskjellige funksjoner. Blant disse er grensene for sikker operasjon av battericellene, samplingfrekvensen til systemet, viftekurven for PWM-signalet til kjølesystemet og meldinger som skal leses og sendes på CANbussen.

Expression	Type	Value
master.conf	config_struct	{...}
ts	uint16_t	100
v_cell_min	uint16_t	25000
v_cell_max	uint16_t	42000
v_cell_chrg	uint16_t	41500
t_cell_min	int8_t	-15 'ñ'
t_cell_max	int8_t	45 '-'
i_offset	uint16_t	20490
i_gain	uint16_t	12120
i_disc_min	int16_t	-10000
i_disc_max	int16_t	20000
i_chrg_min	int16_t	0
i_chrg_max	int16_t	1200
c_avg	int16_t	2500
r_fs_avg	int16_t	210
r_fc_avg	int16_t	0
fc_avg	int16_t	10
fan_ramp_start	int8_t	20 '\024'
fan_ramp_stop	int8_t	50 '2'
fan_ramp_min	uint8_t	32 ' '
fan_ramp_max	uint8_t	224 'à'
balance_threshold	uint16_t	250
v_alpha	uint16_t	500
can_brp	uint8_t	7 '\a'
can_tx_msg_en_startup	uint8_t	2 '\002'
can_tx_msg_id	uint32_t [8]	0x200005d0 <master+40>
can_tx_msg_timing	uint16_t [8]	0x200005f0 <master+72>
can_tx_msg_len	uint8_t [8]	0x20000600 <master+88>
can_tx_msg_data	uint16_t [8][8]	0x20000608 <master+96>
can_rx_msg_en_startup	uint8_t	1 '\001'
can_rx_msg_id	uint32_t [8]	0x2000068c <master+228>
can_rx_msg_timeout	uint16_t [8]	0x200006ac <master+260>
can_rx_msg_len	uint8_t [8]	0x200006bc <master+276>
can_rx_msg_response	uint8_t [8]	0x200006c4 <master+284>
can_rx_msg_data	uint16_t [8][8]	0x200006cc <master+292>

Figur 4.1.3: Oppbygging av konfigurasjonsstrukturen.

Status

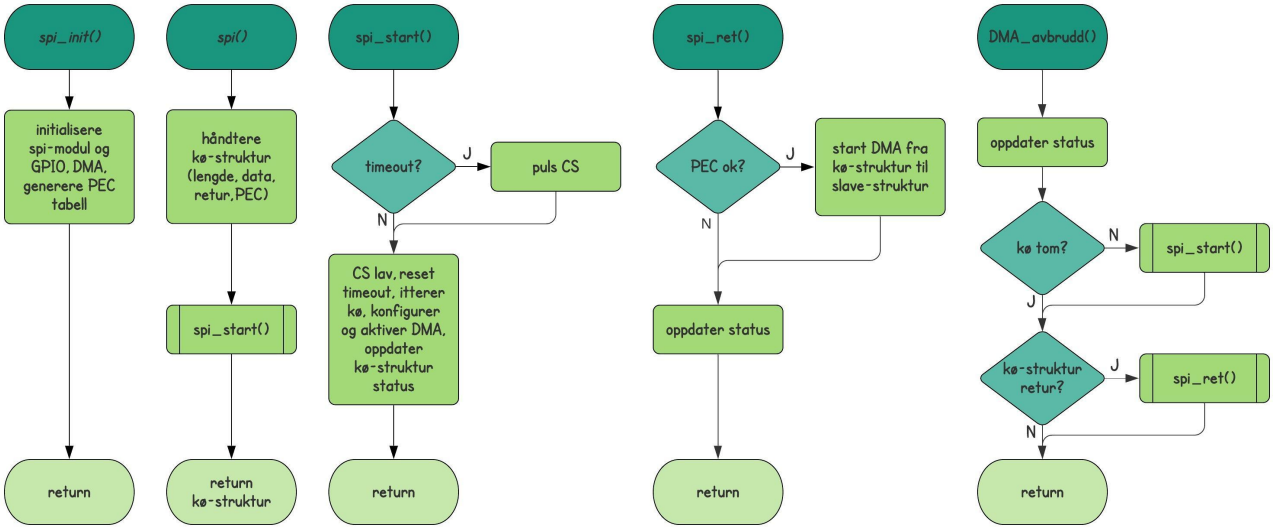
Statusstrukturen vist under i figur 4.1.4 indikerer tilstanden til forskjellige systemer. Ved hjelp av disse tilstandene er det enkelt å overvåke hvilke funksjoner/program som kjører og legge til eventuelle begrensninger med tanke på dette.

Expression	Type	Value
master.status	status_struct	{...}
flt	uint32_t	0
usb	uint32_t	0
i2c1	uint32_t	1
usart1	uint32_t	1
can1	uint32_t	0
charger	uint32_t	0
charging	uint32_t	0
balancing	uint32_t	0
load	uint32_t	0
can_tx_msg_en	uint8_t	2 '\002'
can_rx_msg_en	uint8_t	1 '\001'
can_rx_msg_active	uint8_t	0 '\0'
charger_status	uint8_t	0 '\0'

Figur 4.1.4: Oppbygging av statusstrukturen.

4.2 Kommunikasjon med slavemodulene

Mastermodulen kommuniserer med slavemodulene over isoSPI-grensesnittet. For tilkobling til LT6820 isoSPI-transceiveren brukes et SPI-grensesnitt fra mikrokontrollerens SPI-modul. SPI funksjonene i mikrokontrolleren håndterer SPI-grensesnittet og overføring av kommandoer og data til slavemodulene. Programflyten for programmet vises i flytdiagrammet i figur 4.2.1 under.



Figur 4.2.1: SPI programflyt

Når en melding skal sendes til en slavemodul brukes *spi*(). Denne funksjonen tar argumenter for adresse, kommando, data som skal sendes og data som skal mottas, håndterer oppsettet og formatering av meldingene og returnerer adressen til status for overføringen. Meldingen blir så sendt med DMA-overføring til SPI-modulen i mikrokontrolleren og håndtert med avbrudd når DMA-overføringen er ferdig. Hvis data som blir mottatt skal lagres, kopieres det fra kø-strukturen til returadressen spesifisert ved kjøring av *spi*() (kodesnutten for dette finnes i vedlegg B.1).

For å initialisere SPI-grensesnittet brukes *spi_init*() som konfigurerer det med 10 MHz klokkefrekvens og aktiverer DMA-strømmene som brukes (kodesnutten for dette finnes i vedlegg B.2).

Kø-håndtering

For å håndtere at flere meldinger blir lagt til samtidig brukes en vektor *spi_queue* med kø-strukturer, som inneholder dataverdiene som skal overføres og instruksjon om hvordan returdata skal håndteres (kodesnutt finnes i vedlegg B.3). Dette blir brukt for å unngå å hoppe over meldinger eller stoppe programflyten for å vente på at SPI-modulen skal være ledig. Lengden på vektoren *spi_queue* (kodesnutten for funksjonen finnes i vedlegg B.4) avgjør hvor mange meldinger som kan håndteres samtidig. Ettersom mastermodulen skal kommunisere med 8 slavemoduler og skal sende en melding om gangen til hver, vil den miste lengen være 8, men for margin er det valgt å bruke det dobbelte av dette.

Paritet

IsoSPI protokollen til LTC6804 ICen krever at alle meldinger har tilføyet en 15-bit paritetskode. Denne paritetskoden genereres med en CRC funksjon med startverdien $0x0010$ og polynomet $x^{15}+x^{14}+x^{10}+x^8+x^7+x^4+x^3+1$. Dette er en funksjon som blir utført flere ganger per melding og opp til mange tusen ganger i sekundet. Den bør derfor implementeres så effektivt som mulig for å unngå unødvendig prosessorforbruk. Først ble det gjort et forsøk på en egen implementasjon av algoritmen, men etter at det kjørte for tregt ble dette endret til en algoritme anbefalt av produsenten av LTC6804-en. Den anbefalte algoritmen forenkler oppgaven ved å regne med hele byte i stedet for bit og bruke paritetsverdier fra en forhåndslaget tabell som genereres ved initialisering. Dette reduserte utføringstiden 10 ganger. Denne algoritmen er implementert i funksjonen *spi_pec*() (kodesnutt finnes i vedlegg B.5).

Dataoverføring

Når en melding skal sendes kjøres $spi\_start()$(kodesnutt finnes i vedlegg B.6), som konfigurerer $DMA1\_Stream4$ til å overføre meldingen til DR-registeret i SPI-modulen og $DMA1\_Stream3$ til å overføre mottatt data fra DR-registeret til kø-strukturen. Hvis grensesnittet har vært inaktivt i over 2 ms må det sendes en puls på CS-pinnen for å vekke slavene. Når meldingen sendes oppdateres overføringsstatusen, og når den er ferdig kjøres et avbrudd som håndterer data som returneres og neste overføring.

Avbrudd

Når overføringen er ferdig aktiveres avbruddet $DMA1\_Stream3\_IRQHandler()$ (kodesnutt finnes i vedlegg B.7) som oppdaterer status og sender neste melding i køen. Hvis en returadresse er lagt til og den mottatte meldingen har gyldig paritetsverdier, kopieres mottatt data til returadressen.

4.3 Styring av slavemodulene

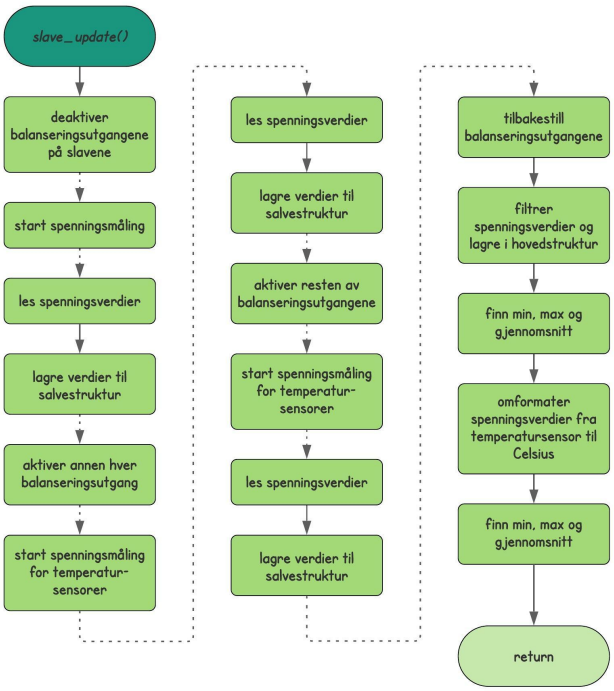
Masteren styrer slavemodulene ved å sende kommandoer og skrive til dataregistrene i slavemodulene over isoSPI. Ved å alternere mellom aktive balanseringsutganger på slavene kan den lese signalene fra temperatursensorene.

Oppstart av slavene

Ved oppstart kjøres *slave_init*() (kodesnutt finnes i vedlegg B.8) som initialiserer SPI-grensesnittet og en timer til generering av avbrudd. Deretter nullstilles og konfigurer den slavene. Hvis slavene ikke responderer returnerer funksjonen en feilmelding.

Måling av cellenes temperatur og spenning

Programflyten for programmet under måling av celledata vises i flytdiagrammet under i figur 4.3.1. For å måle temperatur og spenning kjøres *slave_update*() som sender en serie med kommandoer til slavene med timing bestemt med avbrudd fra $TIM6$ (kodesnutt finnes i vedlegg B.9). Kommandoene måler først spenningsverdiene ettersom disse har tregest filter som nevnt i kapittel 3.2 og krever høyest nøyaktighet. Deretter aktiveres og måles temperatursensorene, og slaven blir tilbakestilt til konfigurasjonen fra slavestrukturen. Når slavene har returnert alle verdiene filtreres spenningsverdiene og temperaturverdiene blir omregnet til celsius med interpolering og overført til vektorene i masterstrukturen (kodesnutt finnes i vedlegg B.10). Samtidig som verdiene overføres beregnes også minimum, maksimum og gjennomsnitt og summene som også blir lagret i masterstrukturen. Resten av C-koden for måling av temperatur og spenning kan finnes i vedlegg B.



Figur 4.3.1: Programflyt for måling med slaver (stiplede linjer er tidsforsinket avbrudd).

Balansering av celler

Når battericellene skal balanseres endres konfigurasjonen i slavestrukturene som nevnt i kapittel 4.3 og blir skrevet til slaven ved neste måling.

4.4 Strømmåling

Signalet fra strømsensoren samples av AD-omformeren hvert ms og overføres med DMA til en vektor. Når funksjonen *master_update*() (kodesnutt finnes i vedlegg B.11) kjøres beregnes gjennomsnittet av verdiene i vektoren samlet etter forrige kjøring. For å konvertere signalet trekkes forskyvning fra gjennomsnittsverdien og dette multipliseres med forsterkningsfaktoren fra konfigurasjonsstrukturen.

4.5 Funksjoner

De overordnede funksjonene under kjøres for hver gang mastermodulen oppdateres.

Batteribeskyttelse

Hver gang verdiene for strøm, spenning og temperatur oppdateres sammenlignes de med grenseverdiene i konfigurasjonsstrukturen. Hvis grensene overskrides deaktiveres sikkerhetssignalet til låsekretsen som kobler fra lasten på batteriet.

Kjølesystemet

Kjølesystemet består av vifter av typen FD1260-AP581E2AL, som styres med et PWM-signal mellom 20 til 30 kHz. Disse reguleres med en viftekurve som styrer viftehastigheten fra den høyeste celletemperaturen. Funksjonen *fan_pwm*() regner viftepådraget med interpolering av viftekurven og temperatur. Så blir pådraget skrevet til CCR-registeret i TIM3 som er koblet til PWM-utgangen.

Estimering av ladestatus

For å estimere ladestatus for batterisystemet brukes Columb-telling som er beskrevet i kapittel 2.5. Strømtrekket integreres og trekkes fra den totale ladningen til systemet. For å redusere påvirkningen for integrasjonsfeil interpoleres ladestatusen fra batterispenningen ved oppstart når det ikke er noen last.

Det hadde vært gunstig å implementere en variant av Kalmanfilter, men dette er nedprioritert på grunn av den korte operasjonstiden mellom lading og utlading.

Styring av lader

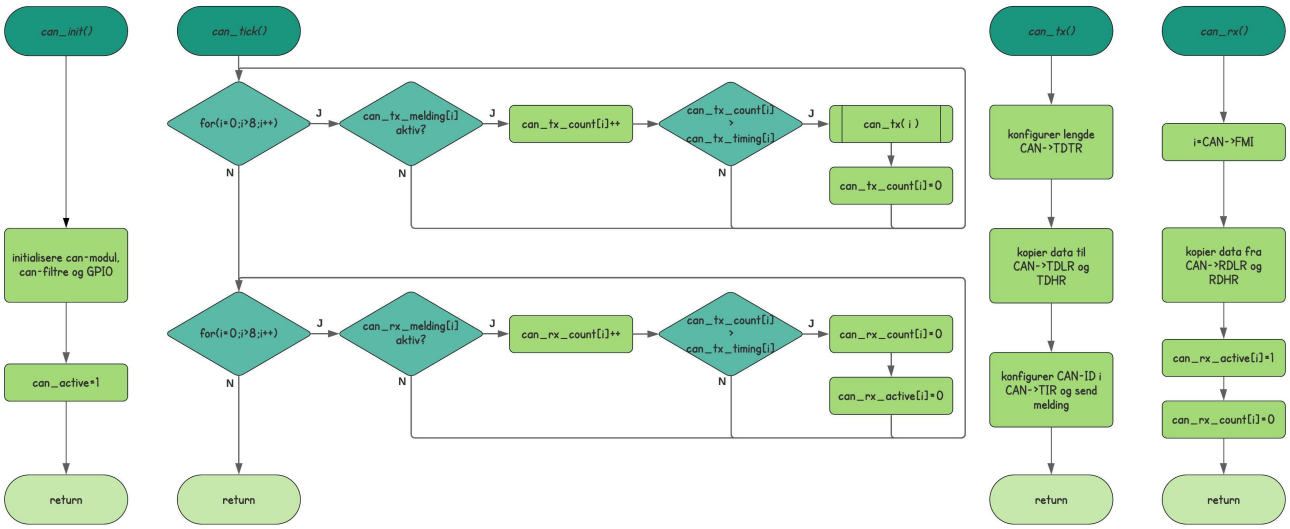
Ladefunksjonen aktiveres når laderen kobles til og sender en melding på CANbussen. Når ladefunksjonen er aktivert sender den et styresignal over CANbus til laderen hvert sekund med grenser for strøm og spenning. Hvis CANbussen kobles fra blir ikke meldingene mottatt og systemet deaktiveres automatisk.

Når den første cellen når den øvre grensen for spenning, stoppes ladingen mens alle cellene blir balansert til den laveste cellespenningen. Etter dette fortsettes ladingen i konstant spenning modus til strømmen fra laderen underskrider 1 A.

Under hele ladeprosessen kjøres overvåking og beskyttelsesfunksjonene som vil deaktivere laderen hvis battericellene opereres utenfor trygge grenser konfigurert i konfigurasjonsstrukturen.

Kommunikasjon med bil og lader

For å kommunisere med den senstrale styringsenheten i bilen (Electronic Control Unit, ECU), motorkontroller og lader brukes CANbus. I flytdiagrammet under i figur 4.5.1 vises programflyten for CAN-kommunikasjonen. Meldinger som skal bli sendt og mottatt blir lastet fra konfigurasjonsstrukturen når CAN-modulen initialiseres. Når en melding som skal sendes aktiveres vil den bli sendt ved neste kjøring av *can_tick*(). Hvis meldingen har en konfigurert verdi for repitisjonsintervall vil en teller inkrementeres hvert ms ved kjøring av *can_tick*() til denne verdien overskrides og meldingen blir sendt på ny.

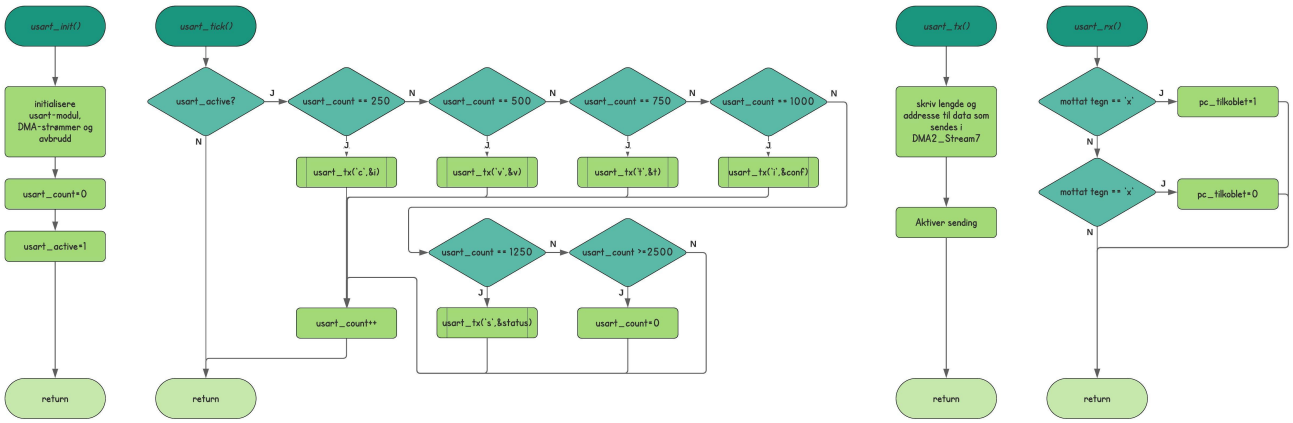


Figur 4.5.1: CAN programflyt.

For meldinger som er programmert for mottakelse lastes meldings-ID-en inn i filterregisterene til CAN-modulen ved initialisering. Hvis en melding blir mottatt aktiveres avbruddet $CAN1\_RX0\_IRQHandler()$ som lagrer verdiene til adressene konfigurert for meldingen, nullstiller timeout-telleren, aktiverer sending av en konfigurert melding og oppdaterer *can_rx_msg_active* i statusstrukturen for å indikere at meldingen har blitt mottatt. Hvis timeout-telleren overskrider timeout-verdien nullstilles statusverdien og svarmeldingen blir deaktivert.

Kommunikasjon med datamaskin

Masteren kommuniserer med datamaskinen over RS232-grensesnittet som er koblet til USART-modulen på mikrokontrolleren. Programflyten for denne kommunikasjonen vises i flytdiagrammet under i figur 4.5.2. Etter konfigurasjon vil USART-modulen lytte konstant etter meldinger og en DMA-overføring vil overføre den til en buffer-vektor. Hvis noe blir mottatt aktiveres avbruddet $USART1\_IRHHandler()$ som sjekker den mottatte verdien. Hvis den mottar ascii-tegnet 'x' aktiveres statusverdien for datamaskintilkobling og masteren vil sende verdiene for temperatur, strøm og spenning til datamaskinen. Hvis ascii-tegnet 'y' blir mottatt deaktiveres sendingen.



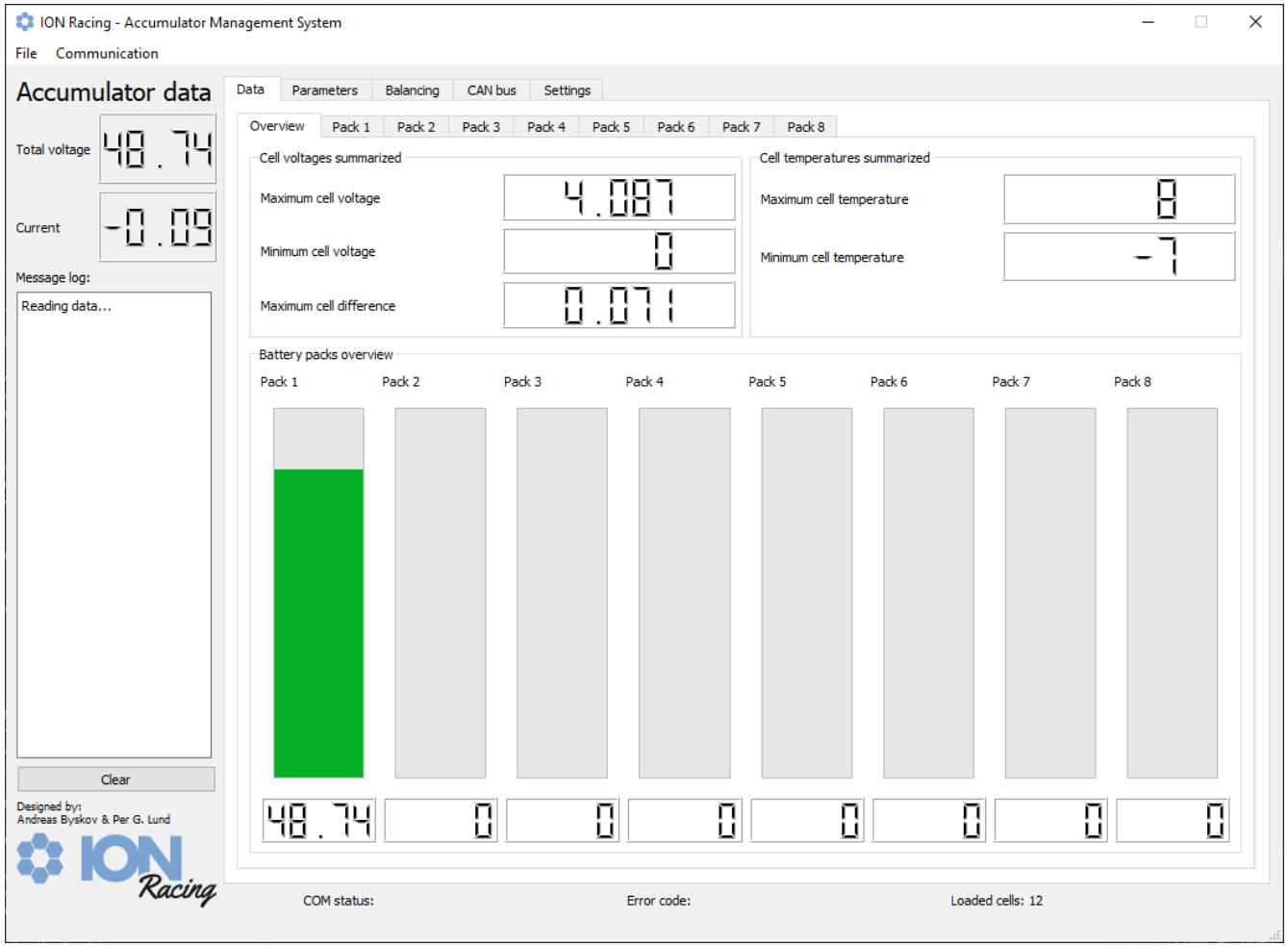
Figur 4.5.2: USART programflyt.

En annen funksjon som var ønskelig å implementere er å kunne oppdatere konfigurasjonen av mastermodulen fra datamaskinen over dette grensesnittet. For å implementere dette kan konfigurasjonen sendes fra datamaskinen i samme format som konfigurasjonsstrukturen og overskrive de lagrede verdiene. Dette ble ikke implementert på grunn av mangel på funksjonalitet i brukergrensesnittet.

4.6 Brukergrensesnitt

Som nevnt tidligere i mastermodulens oppgaver foregår sending og mottakelse mellom BMS-en og grensesnittet over RS232 som også er overgangen fra systemets C-kode til Python kode. For å kunne fremvise og overvåke all data som sendes fra batterisystemet designes det et eget grafisk brukergrensesnitt (Graphical User Interface, GUI) som oppfyller dette. Grensesnittet bygges i Python ved hjelp av Qt Designer [15] og ulike moduler fra PyQt5 [14]. Den ønskede grafikken er først laget i Qt Designer og deretter kjøres en konverteringsfunksjon som auto-genererer Python kodelinjer for grafikken. Kodelinjene kopieres i eget Python-script som styrer det overordnede programmet for brukergrensesnittet. Det spares mye tid med denne prosessen, men koden som genereres er ikke veldig effektiv og kompakt, men ettersom det en en datamskin som skal kjøre programmet og det er relativt lite som skal prosesseres begrunnes dette som godt nok. Dermed er Python-scriptet vedlagt i B.1 på over 5200 linjer hvor rundt 500 av disse kodelinjene er selvlaget.

Selv om mye av koden er auto-generert er det kun den enkle delen altså grafikken som auto-genereres, det er utført endel selvstendig arbeid med kommunikasjon, databehandling og visualisering av data. Blant annet for å kunne oppdatere grafikken med data fra batteriet i brukergrensesnittet samtidig som sending og mottakelse skal kjøre ubrutt tas det i bruk trådprogrammering. Dette forsikrer at begge delene kan kjøre samtidig, det settes opp en egen tråd for oppdatering av grafikken i brukergrensesnittet og en annen tråd for kontinuerlig lesing av COM-porten. I samme tråd blir dataene tolket og validert.



Figur 4.6.1: Brukergrensesnittets meny for fremvisning av batterioversikten. Bildet er tatt med kun én fullstendig pakke (batterisegment) tilkoblet BMS-en som forklarer at kun én pakke registreres i systemet. Samtidig viser det grader fra temperatursensorene før de ble riktig innstilt og kalibrert.

Mottakelse og sending

For mottakelse av batteristatus og sending av parametere/innstilling til batteriet opprettes det som nevnt en egen tråd for seriekommunikasjon vha. pySerial-pakken [1]. Dette starter opp sending og mottaking. Oppstarten av serieporten er fremstilt som kodesnutt B.13 i vedlegg B.3.

Mastermodulen sender all celledata over RS232 og datameldingen dekodes i funksjonen *seriekomm*. Funksjonen *seriekomm* er fremstilt som kodesnutt B.14 i vedlegg B.3.

Databehandling

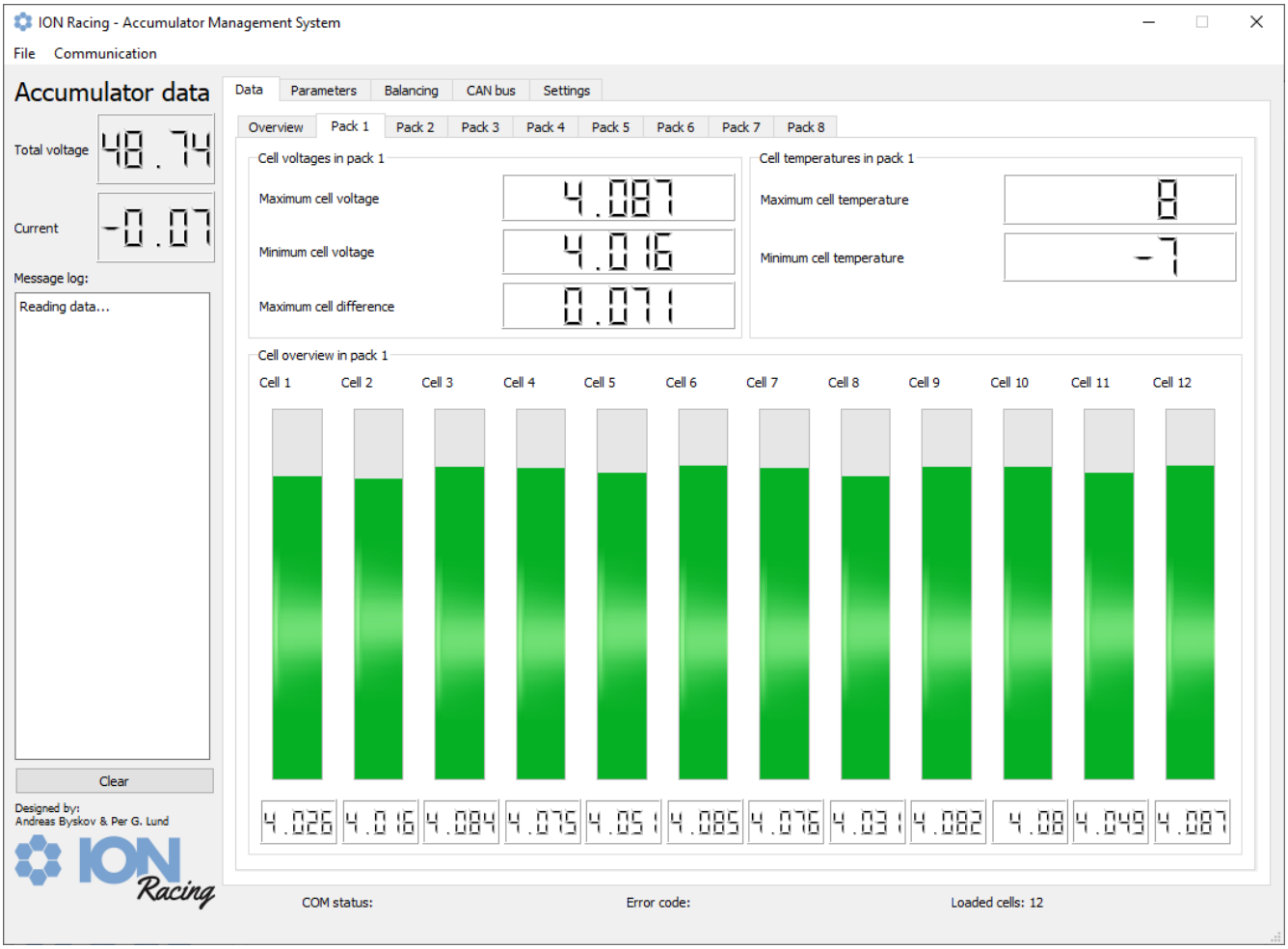
For hver gang målingene fra batteriet er fullstendig mottatt kjører det enkel databehandling av verdiene. Hovedsakelig for å gjøre verdiene kompatible med grafikken til brukergrensesnittet, men også for å bedre fremvise nyttig data. En egen funksjon definert som *max_min_verdier* sjekker all data fra cellene og finner de høyeste og laveste spenningsverdiene for hver cellemodul i hver pakke. Tilsvarende finner den også høyeste og laveste temperaturverdier. Den kjører også gjennom alle spenningsverdiene og finner største differanse mellom cellene. Funksjonen *max_min_verdier* er fremstilt som kodesnutt B.15 i vedlegg B.3.

En annen funksjon *sum_verdier* summerer verdier som brukes til de samlede spenningsnivåene i segmentene og over hele batteriet. Funksjonen *sum_verdier* er fremstilt som kodesnutt B.16 i vedlegg B.3. Resultatene fra de to øvrige funksjonene lagres i lister som er argumenter til oppdateringsfunksjonen for grafikken til brukergrensesnittet.

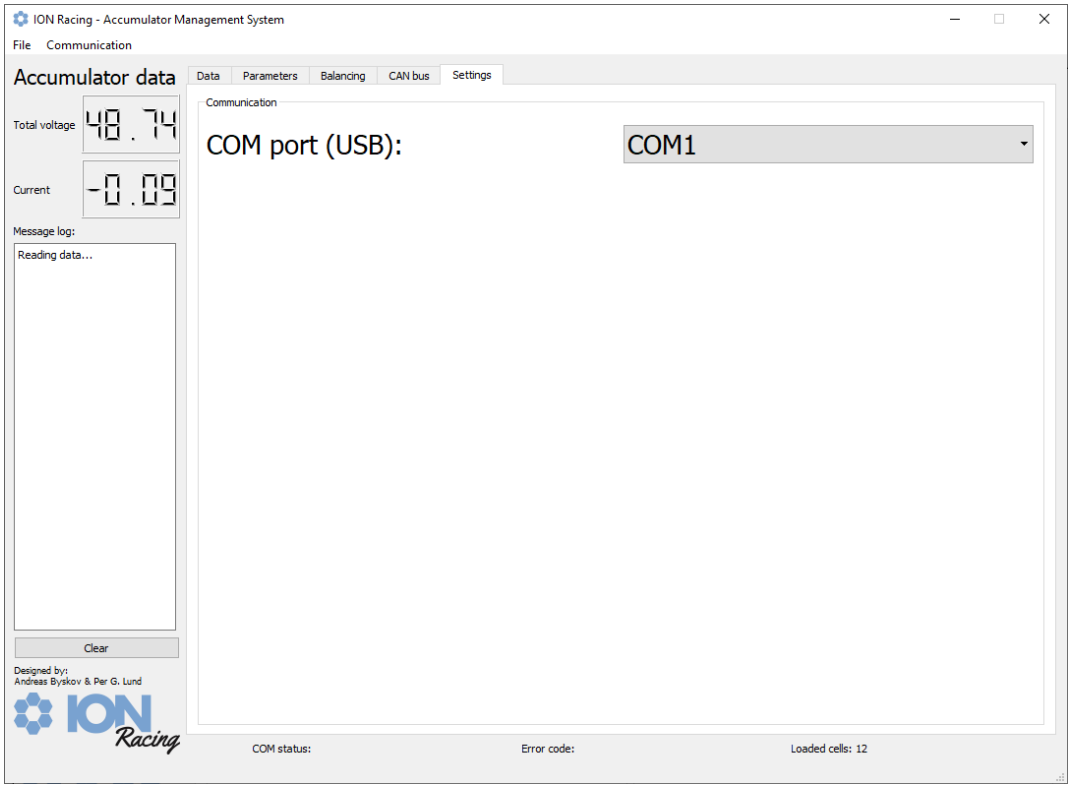
Visualisering av data

Grafikken til brukergrensesnittet oppdaters i en egen tråd og venter med å oppdatere seg etter hver fullført melding over RS232 (ca. hvert sekund). Alle verdier og bargrafer oppdateres da utifra innholdet fra meldingen over RS232. Oversiktsfanen (Overview) i figur 4.6.1 fra tidligere og bildene under tatt fra programmet under kjøring viser alle verdier som oppdateres og hvilke andre funksjoner som kan endres. Den spesifikke oppdateringsfunksjonen for brukergrensesnittet *oppdaterGUI* er fremstilt som kodesnutt B.17 i vedlegg B.3.

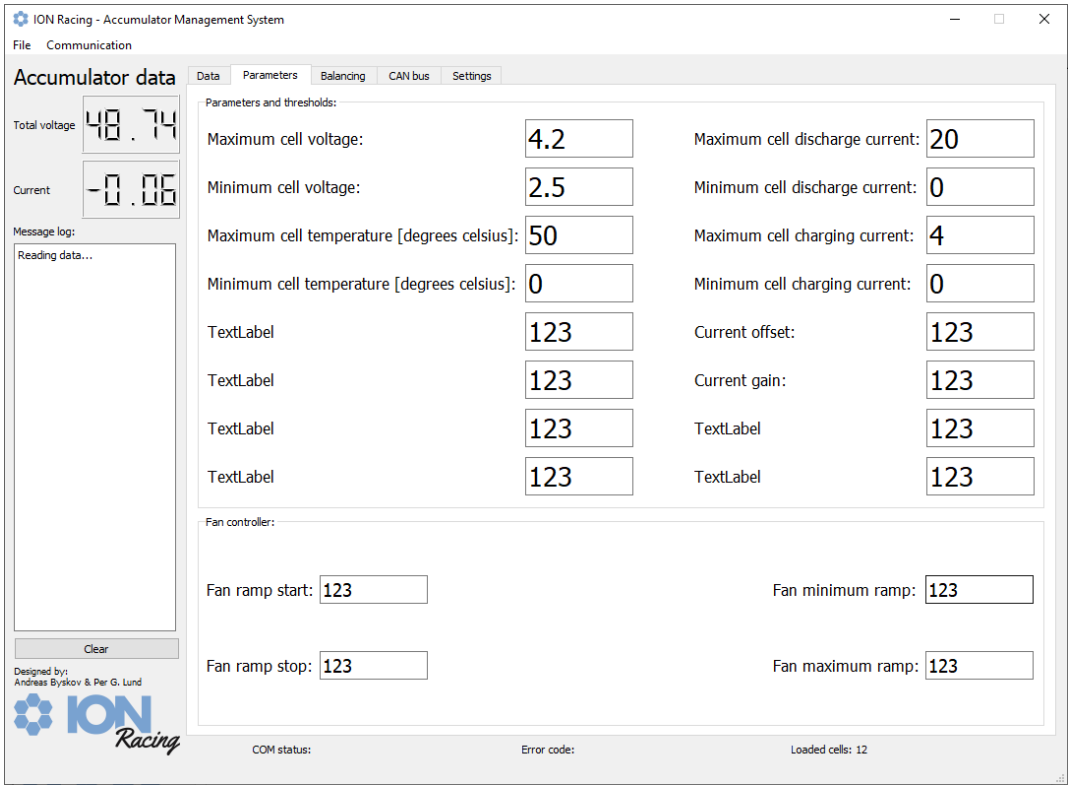
Ulike små tilleggs funksjoner er også implementert som blant annet meldinglogg til venstre og generell info nederst med kommunikasjonsstatus, feilkoder og antall celler tilkoblet.



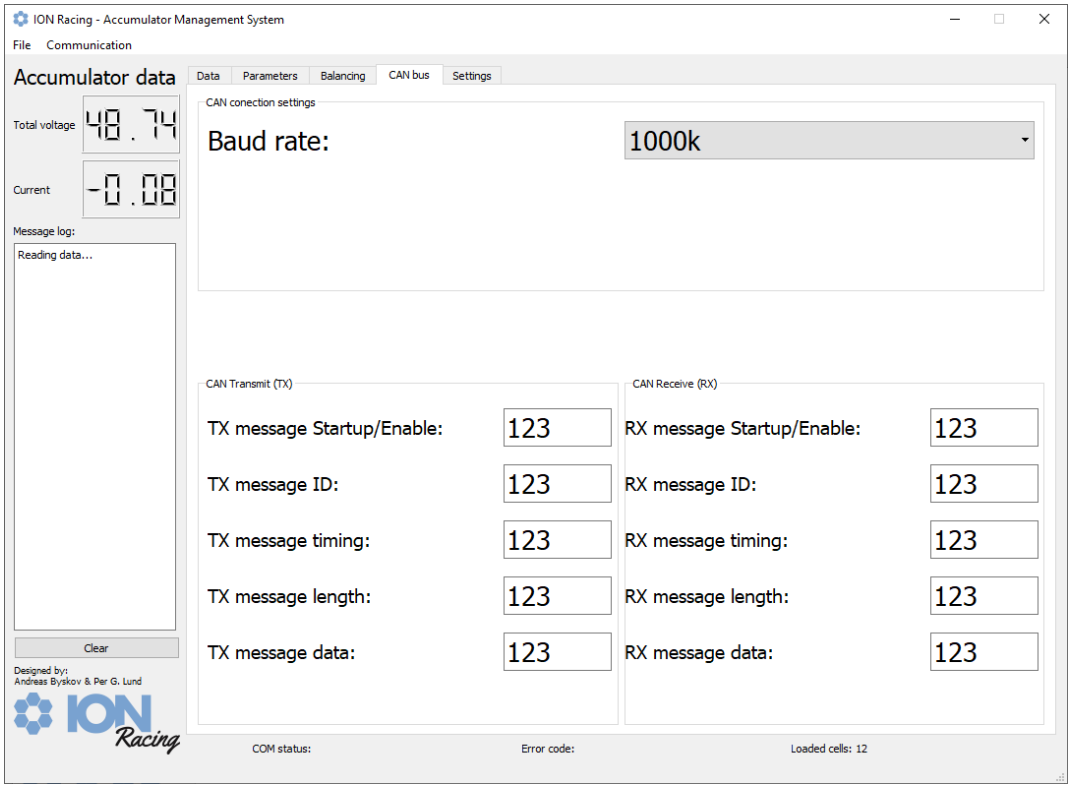
Figur 4.6.2: Brukergrensesnittets meny for fremvisning av celledata til hver av de 8 individuelle pakkene i batteriet. Bildet er tatt med kun én fullstendig pakke (batterisegment) tilkoblet BMS-en som forklarer at kun én pakke registreres i systemet. Samtidig viser det grader fra temperatursensorene før de ble riktig innstilt og kalibrert.



Figur 4.6.3: Brukergrensesnittets meny for generelle innstillinger



Figur 4.6.4: Brukergrensesnittets meny for innstilling av ulike parametere.



Figur 4.6.5: Brukergrensesnittets meny for oppsett av CANbus.

Kapittel 5

Tester og resultater

Etter ferdig lodding og produksjon av egendesignet kretskort funksjonstestes de først hver for seg. Deretter er det utført flere tester på sammenstilling av alle tre kretskortene med et enkelt batterisegment. Dette inkluderer altså mastermodulens kort, slavemodulens kort og slavemodulens tilkoblingskort. Bilder fra de ferdig produserte kretskortene er vedlagt i vedlegg C.3.

5.1 Signaler

For å teste at signalverdiene stemmer og er nøyaktige sammenlignes de med eksterne målinger med multimeter og oscilloskop.

Spenningsmåling

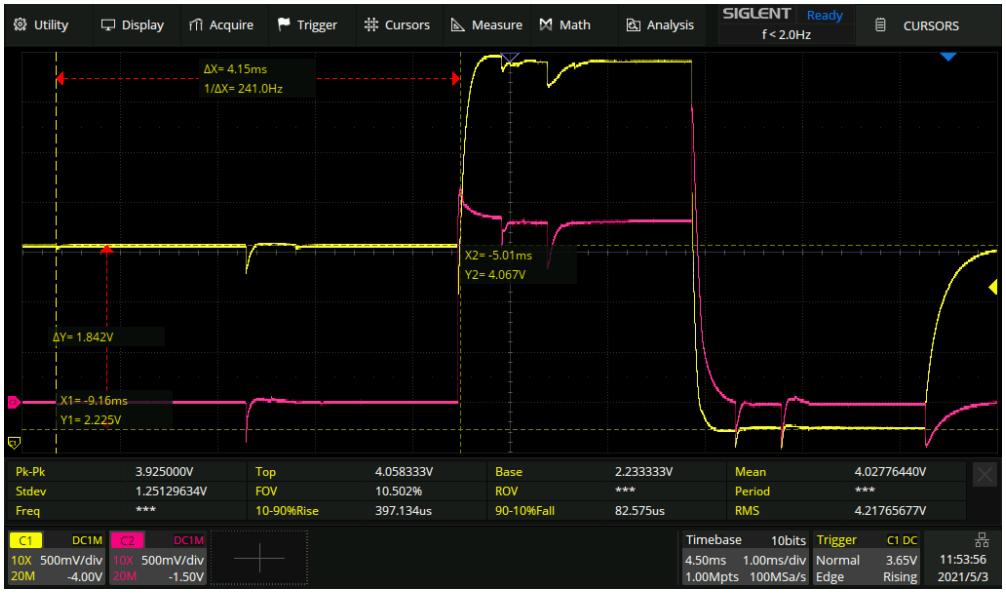
Måling av spenning med slavekortet testes ved å koble den på batterisegmentet og sende kommando for måling fra masteren. Når målingen er utført leser masteren verdiene og de kan vises i sanntid i STM32CubeIDE. Verdiene sammenlignes med målinger fra oscilloskop og multimeter. Ettersom en spenningsforsyning med like høy oppløsning som AD-omformerene i slavene ikke er tilgjengelig testes det bare med spenningen på batterisegmentet.

Ved kjøring av spenningsmåling med 10 Hz oppdaterignsfrekvens og et IIR-filter viser verdien for den laveste cellen i mikrokontrolleren 4.0798 V, mens det ble målt til 4.067 V med multimeter. Avviket i dette tilfellet er 128 mV som tilsvarer 0.3 %.

Temperaturmåling

Målingen av temeratursignalene er ikke nøyaktige på grunn av bruk av transistorer med en høyere gate-source terskelspenning enn signalet fra temperatursensorene. Dette begrenser utgangsverdien til dette og hindrer det egentlige signalet fra å måles.

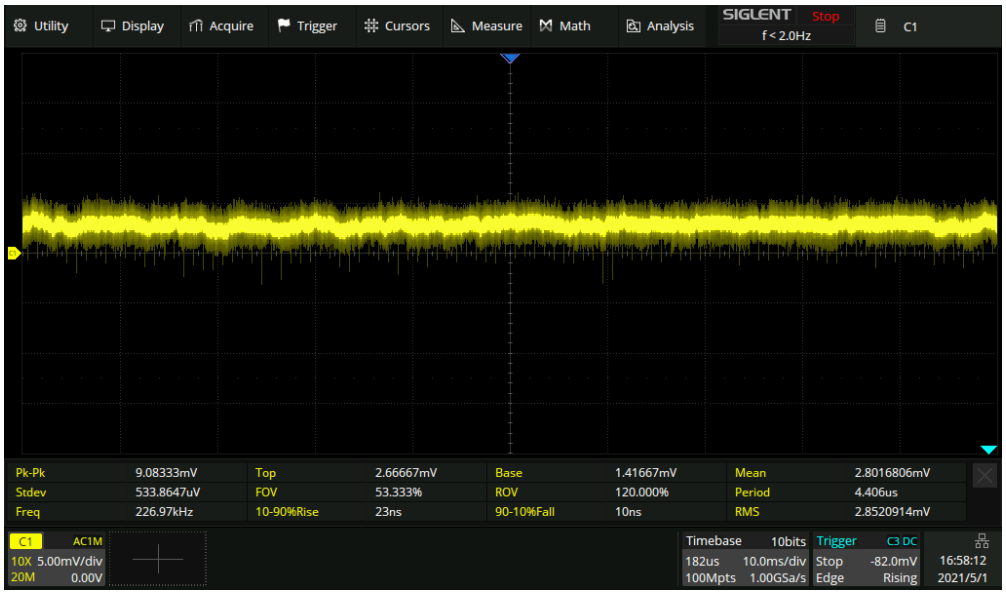
Den målte spenningen ender på 2.225 V som vist i figur 5.1.1. I følge databladet til cellemodulene tilsvarer dette en temperatur på omtrent 4 °C.



Figur 5.1.1: Skjermbilde av måling av temperatursignal. (CH1: spenning over C9, CH2: spenning over temperatursensor)

Strømmåling

Strømsignalet er mer eller mindre det eneste analoge signalet på masterkortet og er derfor mest utsatt for støy. I figur 5.1.2 måles derfor støy på signalet ved inngangen til AD-omformeren. Målingen foretas med 0 A strøm i strømsensoren. Målingen resulterer i 9.08 mV_{pk-pk} og 534 µV std. deviasjon, noe som tilsvarer et maksimalt avvik på 1.2 A og vil kunne filtreres ut digitalt.



Figur 5.1.2: Skjermbilde av støy på strømsignal ved inngang til adc på mikrokontrolleren.

Viftestyring

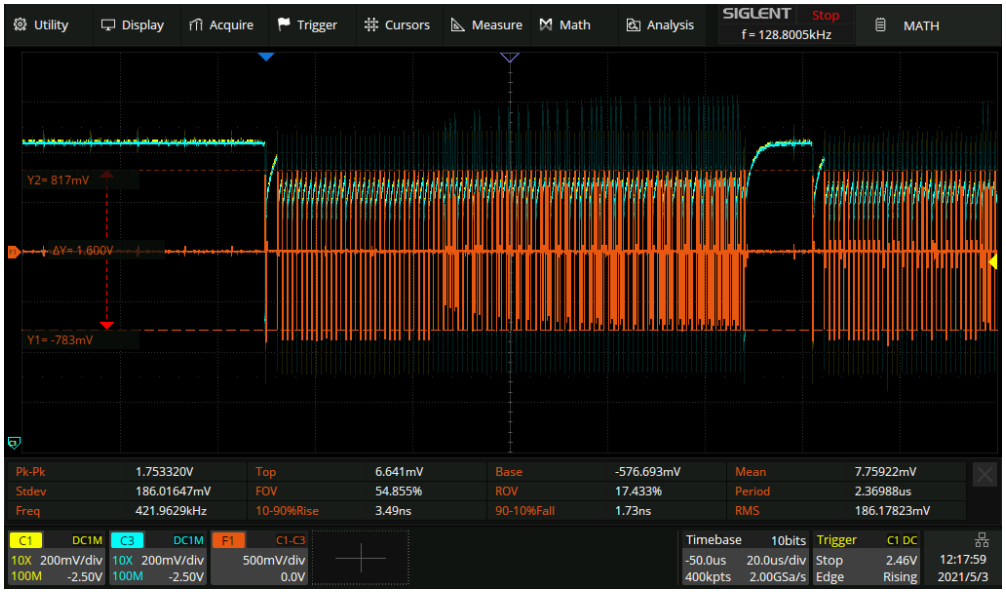
Viftesignalet måles for å verifisere at PWM-signalet genereres og at det stemmer med viftekurven. I figur 5.1.3 vises PWM-signalet målt fra konnektoren på masterkortet. Amplituden og frekvensen er tilstrekkelige, men overgangsraten er lavere enn foretrukket. Tregheten i overgangen forårsakes av RC-kretsen som dannes av R7 og kapasitansen i gaten til Q3 i figur 3.3.2. Tregheten vil ikke påvirke signalet betraktelig, men vil forskyve det litt ved korte pulsbredder. Det vil derfor være foretrukket å redusere motstanden til R7.

Figur 5.1.3: Skjermbilde av PWM-signal til styring av vifter.

isoSPI

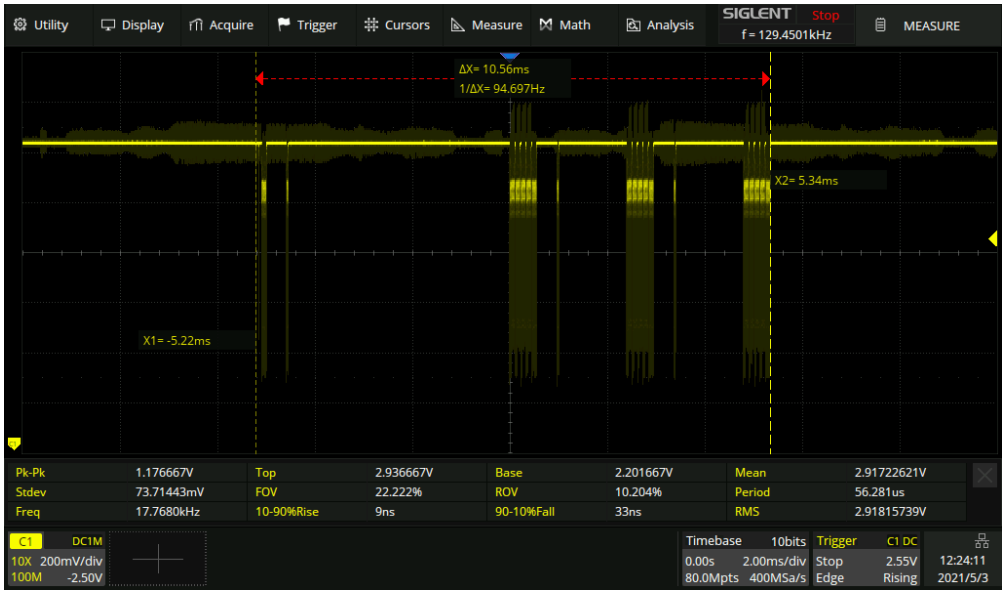
Masteren og slavene er koblet sammen med isoSPI-grensesnittet. For sikre signalintegritet og for å unngå at grensesnittet overbelastes er det viktig å måle signalstyrken og aktiviteten.

Konfigurasjonsresistorene til grensesnittet er konfigurert til strømpulser på 10 mA i likning 3.1 og terskelspenning på 370 mV i likning 3.2. For at grensesnittet skal fungere må amplituden på de mottatte signalene være over terskelspenningen. I figur 5.1.4 måles amplituden til å være mellom 783 mV og 817 mV. Dette er over 2 ganger terskelverdien, men denne målingen er bare mellom en slave og masteren. Med flere slaver vil amplituden reduseres. Hvis signalstyrken er liknende med flere slaver kan enten terskelspenningen økes eller strømpulsene reduseres avhengig om lavere strømtrekk og utstråling av støy eller mer støymotstand er ønskelig.



Figur 5.1.4: Skjermbilde av isoSPI-signaler på masterkort. (CH1:IP på LTC6820, CH2: IM på LTC6820, F1: differanse mellom IP og IM)

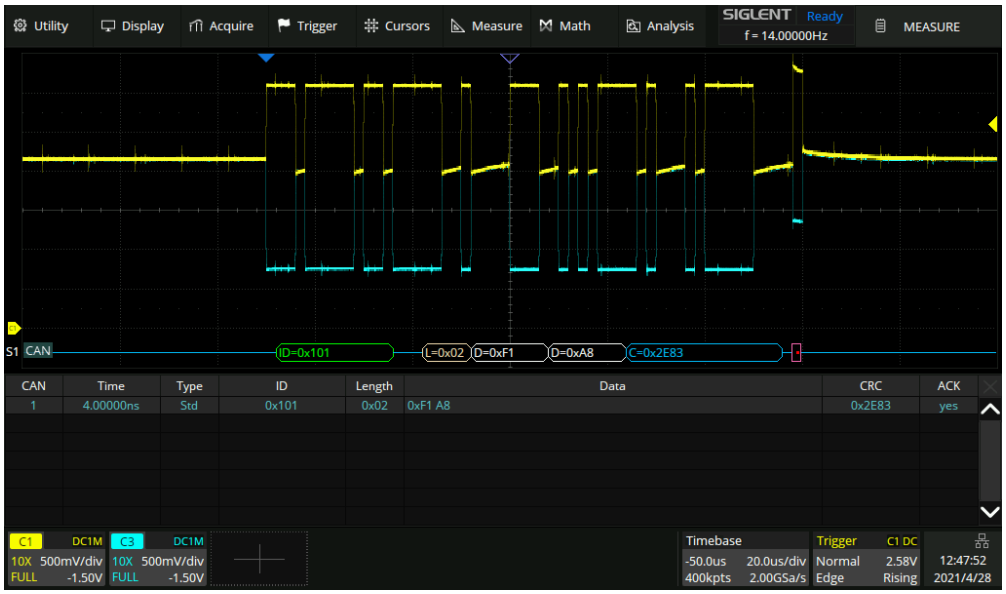
For å finne den maksimale samplingfrekvensen for slavene må kapasiteten på grensesnittet tas til henhold. I figur 5.1.5 måles aktiviteten på grensesnittet for en målesyklus. Med èn slave tar det 10 ms å fullføre målingen, men med flere slaver vil dette sannsynligvis øke til rundt 20 ms. Fra dette vil den maksimale samplingfrekvensen med denne konfigurasjonen være 50 Hz



Figur 5.1.5: Skopbilde av SPI SDA mellom mikrokontroller og LTC6820.

CANbus

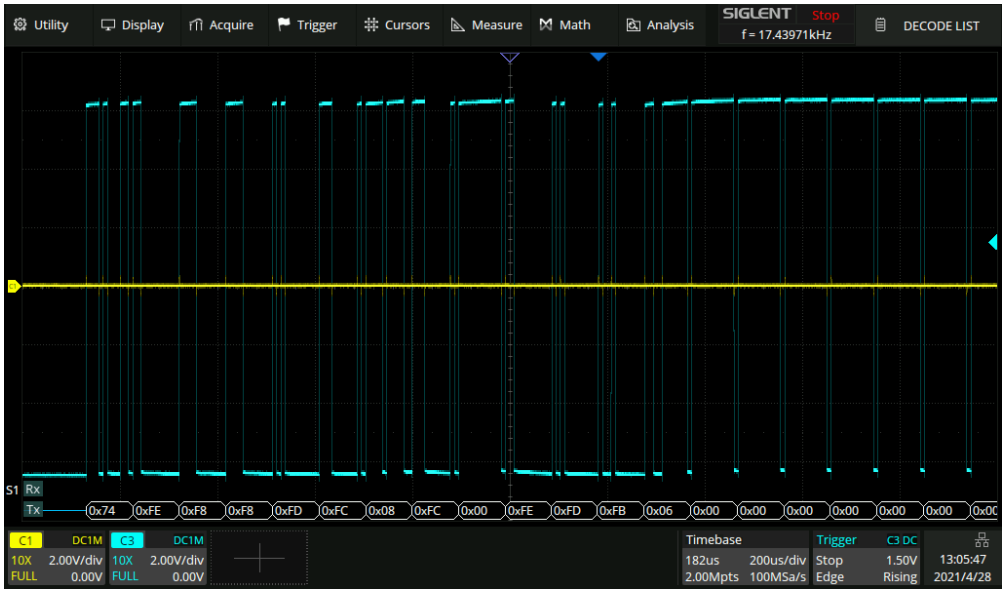
Under i figur 5.1.6 måles signal fra CANbussen ved sending fra masteren til en CAN til USB adapter.



Figur 5.1.6: Skjermbilde av CANbus-melding. (CH1: CAN:H, CH3: CAN:L)

RS232

Under i figur 5.1.7 måles signal fra RS232 mellom master og pc.



Figur 5.1.7: Skjermbilde av datapakke sendt over RS232. (CH1: Rx, CH3: Tx)

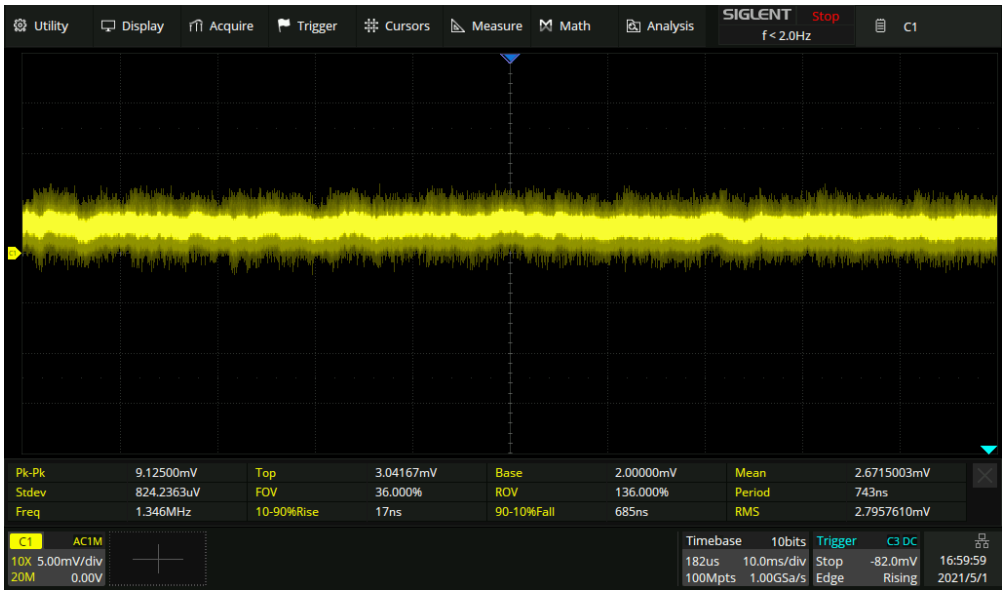
Forsyningsspenninger

For å forsikre at kretsene har stabil spennignsforsyning måles DCDC-konverterene for rippel og støy.

I figur 5.1.8 måles rippelspenningen 6.5 V forsyningen til kretsene på masterkortet. Spenningsforsyningen har litt rippel med 35.5 mV_{pk-pk} og 2.06 mV std. deviasjon. Dette er akseptabelt for digital elektronikk og blir i tillegg regulert med lineære spenningsregulatorer til kretsenes forsyningsspenning.

Figur 5.1.8: Skjermbilde av støy på 6.5 V forsyningsspenning.

I figur 5.1.9 måles rippelspenningen på 15 V forsyningen til strømsensoren med 0A strøm. Spenningsnivået er nesten helt fritt for rippel med 9.125 mV_{pk-pk} og 824µV std. deviasjon. Det er usikkert om høyfrekvent strøm gjennom sensoren vil påvirke stabiliteten, men dette tyder på at reduksjonen i kompleksitet i kapittel 3.3



Figur 5.1.9: Skjermbilde av rippel på 15 V forsyningsspenning.

I figur 5.1.10 vises oppstartsstrømmen til masterkortet. De to første strømpulsene trekkes til 15 V forsyningen og de neste strømpulsene trekkes til 6.5 V forsyningen. De første strømpulsene er relativt høye og burde muligens dempes med et RC- eller LC-filter.

Figur 5.1.10: Skjermbilde av spenning og strømtrekk ved oppstart av masterkortet. (CH1: 12 V spenningstilførsel, CH2: 15 V forsyning til strømsensor, CH3: 6.5 V forsyning, F1: strømtrekk fra spenningstilførselen)

Slavenes strømtrekk

Under i figur 5.1.11 måles slavens strømmtrekk over resistorene R1 og R2 vist i tidligere figur 3.2.2. Den gjennomsnittlige spenningen som måles er 1.286 mV, som gir en gjennomsnittsstrøm på 12.86 µA.

Figur 5.1.11: Skjermbilde av spenning over R1, R2 og summen av disse på slavemodul. (CH1: spenning over R1, CH2: spenning over R2, F1: sum av spenningene)

5.2 Prossessor- og minneforbruk

For å finne ut hvor mye prosessoren i mikrokontrolleren blir brukt og om det er mulighet for utvidelse brukes PC-sampling funksjonen i STM32CubeIDE. PC-sampling leser med jevne mellomrom verdieren til programpekeren i prosessoren for å kunne analysere hvilken funksjon som kjører. Over tid kan den da representere statistikk for prossessorforbruket til ulike funksjoner. Dette vises i figur 5.2.1. Her kan det sees at while-løkken i *main*() bruker 98.6 % av prosessorforbruket, da er prosessoren ledig og ingen avbrudd kjører.

Med flere slaver vil prosessorforbuket øke. Hvis vi antar at prosessorforbruket skalerer proporsjonalt med antall slaver vil det være 11.2 % og det vil fortsatt være god margin. Denne prosessortiden kan utnyttes til mer avanserte funksjoner som implementeres senere.

Function	% in use	Samples	Start address	Size
main()	98,58%	628990	0x8003045	0x10
EXTI1_IRQHandler()	0,79%	5051	0x8002cad	0x84
spi_start()	0,19%	1215	0x8004745	0x148
Unknown function	0,13%	840	0x0	0x0
spi()	0,13%	816	0x800488d	0x164
eeprom_load_conf()	0,08%	505	0x8002509	0x214
slave_update4()	0,06%	365	0x8003fad	0x4cc
master_update()	0,03%	210	0x8003145	0x248
v2t()	0,00%	29	0x80036c9	0xcc
EXTI3_IRQHandler()	0,00%	14	0x8002049	0x18c
eeprom_handler()	0,00%	5	0x800271d	0x504
I2C1_EV_IRQHandler()	0,00%	1	0x8002d31	0xc

Figur 5.2.1: Prosessorforbruk for funksjoner i mikrokontrolleren.

I figur 5.2.2 vises en analyse av minneforbruk i mikrokontrolleren fra STM32CubeIDE. Det totale minneforbruket for denne implementasjonen er 5.91 % RAM og 1.97 % Flash. Minnet som er til overs kan brukes til lagring av mer data til mer omfattende filtrering og analyse.

Memory Regions | Memory Details

Region	Start address	End address	Size	Free	Used	Usage (%)
RAM	0x20000000	0x20020000	128 KB	120,43 KB	7,57 KB	5,91%
FLASH	0x08000000	0x08100000	1024 KB	1003,86 KB	20,14 KB	1,97%

Figur 5.2.2: Minneforbruk av koden til mikrokontrolleren.

Kapittel 6

Diskusjon og konklusjon

For å tilfredsstille regelverket og de andre eksisterende systemene på bilen som var et viktig punkt fra oppgaveteksten i kapittel 1.3 er det utført mer omfattende endringer på tidligere design enn først planlagt. Under gjennomgangen av det tidligere designet var det tydelige feil på både oppbyggingen av batterisystemet og design av kretskortene. Implementeringen av overvåkingssystemet krever at hele batterisystemet fungerer feilfritt og oppgaven har derfor dekket en del mer av det overordnede batterisystemet enn kun batteriovervåkingen.

Batterisystemet

For den fysiske oppbyggingen av batteriet er det gjort endringer for å bedre sikkerheten ved montering og bruk av batteriet. 3D-printet skillevegg, vedlikeholdsplugger og innpakningen batterisegment i Nomex-papir er tiltak som er gjort til å forbedre dette. Koblingsmønsteret for celletilkobling i batteriet er også endret for å oppfylle regelverket med tanke på vedlikeholdspluggene.

På kretskortene er det utført både feilretting og effektivisering av de tidligere kretskortene. Blant annet er størrelsen av mastermodulen redusert til 26 % av det opprinnelig designet fra 2017 og 38.5 % av fjorårets design se figur C.3.1 vedlagt i vedlegg C.3 som fremviser endringen.

For implementeringen av batteriovervåkingssystemet er det utført programmering i C-kode for alle de nødvendige funksjonene som er nevnt i oppgaveteksten. Blant annet overvåking av cellenes temperatur, spenning og temperatur med styring av et regulert kjølesystem og lading av batteriet med cellebalansering. De nevnte funksjonene er helt egenprodusert kode som ikke var utført i det tidligere designet fra fjorårets masteroppgave [12].

Verifisering av at overvåkingssystemet kjører og fungerer fremvises i det grafiske brukergrensesnittet laget i Python som presenterer celledata fra batteriet som verdier og bargrafer. Brukergrensesnittet kjører i sanntid sammen med overvåkingssystemet og har funksjoner for å eventuelt overstyre mastermodulen med kommandoer eller endring av innstillinger.

For å kunne ferdigstille batterisystemet til det endelige målet må ytterlige tester gjennomføres på en fullstendig sammenstilling med alle systemene på bilen som påvirker og er påvirket av batterisystemet. Tidspunktet for denne rapporten samsvarer beklageligvis ikke helt med ION Racings fremdriftsplan, men systemet vil antageligvis ferdigstilles utover sommeren før konkurransen i juli.

Sammenstilling og videre arbeid

Mer spesifikt skal flere tester utføres på batterisystemet når racerbilen ferdigstilles. Da kan det testes integrasjon med andre systemer. Med last fra motoren i racerbilen kan da også estimering av ladestatus testes.

Oppgaver som videre kan og burde implementeres er blant annet enklere og flere konfigurasjonsmuligheter ved hjelp av brukergrensesnittet inkludert funksjoner for diagnostikk og kalibrering av slaver. Mikrokontrolleren har en del prosessorforbruk og minneplass til overs og man kan dermed se på implementasjon av mer nøyaktighet på målingene. Ved å ta i bruk flyttall vil målingene ta mer plass og være mer krevende å prosessere, men vil gi mye bedre

nøyaktighet. For estimering av ladestatus er det kun tatt i bruk en elementær metode. Fjorårets masteroppgave [12] hadde en løsning på estimering av ladestatus ved bruk av kalmanfilter og kan mulig implementeres for bedre resultat.

Konklusjon

Målet med oppgaven har vært å implementere et fungerende batteriovervåkingssystem til denne sesongens racerbil. Fra oppgaveteksten i kapittel 1.3 er alle punkter for design og konstruksjon ferdig implementert og tester som har latt seg gjøre innen tidspunktet til denne rapporten er utført med gode resultater. Ettersom sesongen i Formula Student fortsetter utover sommeren, er det fortsatt rimelig tid til å utføre resterende tester som mangler når nødvendige andre systemer på bilen er ferdigstilt. På bakgrunn av dette anses implementeringen til å være utført på en velykket måte og gir ION Racing gode muligheter for veien videre.

Bibliografi

[1] Chris Liechti. *pySerial's documentation*, 2017. URL `https://pyserial.readthedocs.io/en/latest/index.html`. [Lest: 12.05.21].

[2] Davide Andrea. *Battery Management Systems for Large Lithium-Ion Battery Packs*. ARTECH HOUSE, 2010.

[3] DuPont. *DuPont™ Nomex® 410 Technical Data Sheet*, 2016. URL `https://www.dupont.com/content/dam/dupont/amer/us/en/safety/public/documents/en/DPT16_21668_Nomex_410_Tech_Data_Sheet_me03_REFERENCE.pdf`. [Lest: 12.05.21].

[4] ElCon. *HK-J Series 6.6KW IP 67 Sealed UHF CAN Bus Charger, VER A*, november 2016. URL `https://www.elconchargers.com/f/6.6KW_User_Manual.pdf`. [Lest: 12.05.21].

[5] Energus Power Solutions. *Li-ion building block Li8P25RT*, oktober 2016. Generelt datablad for Energus sine cellemoduler, mottatt fra Energus via. epost.

[6] Formula Student Germany. *Formula Student Rules 2020*, 2020. URL `https://www.formulastudent.de/fileadmin/user_upload/all/2020/rules/FS-Rules_2020_V1.0.pdf`. [Lest: 12.05.21].

[7] Formula Student UK. *FORMULASTUDENT UK 2021 SUPPLEMENTARY RULES*, 2021. URL `https://www.imeche.org/docs/default-source/1-oscar/formula-student/2021/rules/fsuk-2021-supplementary-rules---v1-04e94f48d54216d0c8310ff0100d05193.pdf?sfvrsn=2`. [Lest: 12.05.21].

[8] In-Young Jang. *Lithium-ion rechargeable cell for power tools*. Samsung, mars 2014. URL `http://dalincom.ru/datasheet/SAMSUNG%20INR18650-25R.pdf`. [Lest: 12.05.21].

[9] LearnEMC. *EMC Design Guideline Collection*, 2020. URL `https://learnemc.com/emc-design-guidelines`. [Lest: 12.05.21].

[10] Linear Technology Corporation. *LTC6804-1/ LTC6804-2, Multicell Battery Monitors*, 2013. URL `https://www.analog.com/media/en/technical-documentation/data-sheets/680412fc.pdf`. [Lest: 12.05.21].

[11] Python Software Foundation. *The Python Language Reference*, 2021. URL `https://docs.python.org/3/reference/index.html#reference-index`. [Lest: 12.05.21].

[12] Raymond Tjørhom. *Estimering av ladestatus i en elektrisk racerbil med bruk av Kalmanfilter*. Masteroppgave, Universitetet i Stavanger, juni 2020.

[13] Stratasys. *ULTEM™ 9085 Resin*, 2020. URL `https://www.stratasys.com/materials/search/ultem9085`. [Lest: 12.05.21].

[14] The Qt Company. *Qt for Python Documentation*, 2021. URL `https://doc.qt.io/qtforpython/`. [Lest: 12.05.21].

[15] The Qt Company. *Qt Designer Manual*, 2021. URL `https://doc.qt.io/qt-5/qtdesigner-manual.html`. [Lest: 12.05.21].

[16] Wilfried Voss. *A Comprehensible Guide to Controller Area Network*. Copperhill Media Corporation, 2008.

Vedlegg A

Skjematikk og utlegg

I dette vedlegget viser all aktuell skjematikk og utlegg som er brukt i oppgaven.
**Vedleggene kan åpnes eller lastes ned ved å dobbeltklikke på vedleggene under (fungerer dersom oppgaven er åpnet i nettleserne Firefox, Edge eller Safari).*

Skjematikk og kretskortutlegg

Vedlegg B

Programvarefiler

I dette vedlegget viser aktuelt C-prosjekt, Matlab-script, Python-script og QtDesigner GUI-prosjekt som er brukt i oppgaven.

B.1 Fullstendige prosjekt og filer

**Vedleggene kan åpnes eller lastes ned ved å dobbeltklikke på vedleggene under (fungerer dersom oppgaven er åpnet i nettleserne Firefox, Edge eller Safari).*

C-prosjekt

Matlab-script for simuleringer

Python-script GUI m/logo

QtDesigner GUI-prosjekt

B.2 Kodesnutter fra mastermodulen (C-kode)

SPI

```
133 int8_t *spi( uint8_t addr, uint16_t cmd, const uint8_t * const write_data,
134         uint8_t write_len, uint8_t * const read_data, uint8_t read_len , uint32_t * time ){
135     spi_queue_struct *queue = spi_get_queue();
136     if( !queue ) return NULL;
137     queue->status = 1;
138     queue->len = 4;
139     queue->ret_data = read_data;
140     queue->ret_time = time;
141     queue->tx[0] =(uint8_t)( cmd >> 8 );
142     if( addr ) queue->tx[0] |= 0x80 | ( addr << 3 );
143     queue->tx[1] = (uint8_t)cmd;
144 
145     uint16_t cmd_pec = spi_pec( queue->tx, 2 );
146     queue->tx[2] = (uint8_t)( cmd_pec >> 8 );
147     queue->tx[3] = (uint8_t)cmd_pec;
148 
149     if( write_len ){
150         for( uint8_t i = 0; i < write_len; i++ ) queue->tx[queue->len++] = write_data[i];
151         uint16_t data_pec = spi_pec( &queue->tx[4], write_len );
152         queue->tx[queue->len++] = (uint8_t)( data_pec >> 8 );
153         queue->tx[queue->len++] = (uint8_t)data_pec;
154     }
155 
156     if( read_len ){
157         for( uint8_t i = 0; i < read_len; i++ ) queue->tx[queue->len++] = 0xFF;
158     }
159 
160     if( !( SPI2->SR & SPI_SR_BSY ) ) spi_start();
161 
162     return &queue->status;
163 }
```

Kodesnutt B.1: Funksjon for sending av melding til slaver. (spi.c)

```
14 void spi_init( void ){
15 
16     // PIN setup
17     RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;
18     GPIOB->MODER |= 0b01 << GPIO_MODER_MODE12_Pos | 0b10 << GPIO_MODER_MODE13_Pos |
19             0b10 << GPIO_MODER_MODE14_Pos | 0b10 << GPIO_MODER_MODE15_Pos;
20     GPIOB->AFR[1] |= GPIO_AF5_SPI2 << GPIO_AFRH_AFSEL13_Pos | GPIO_AF5_SPI2 << GPIO_AFRH_AFSEL14_Pos |
21             GPIO_AF5_SPI2 << GPIO_AFRH_AFSEL15_Pos;
22     GPIOB->BSRR = GPIO_BSRR_BS_12;
23 
24     // SPI setup
25     RCC->APB1ENR |= RCC_APB1ENR_SPI2EN;
26     SPI2->CR2 = SPI_CR2_ERRIE | SPI_CR2_TXDMAEN | SPI_CR2_RXDMAEN;
27     SPI2->CR1 = SPI_CR1_SSM | SPI_CR1_SSI | SPI_CR1_SPE | 0x4 << SPI_CR1_BR_Pos |
28             SPI_CR1_MSTR | SPI_CR1_CPOL | SPI_CR1_CPHA;
29 
30     // DMA setup
31     RCC->AHB1ENR |= RCC_AHB1ENR_DMA1EN | RCC_AHB1ENR_DMA2EN;
32 
33     // SPI2 RX DMA
34     DMA1_Stream3->CR = 0b10 << DMA_SxCR_PL_Pos | DMA_SxCR_MINC | 0b00 << DMA_SxCR_DIR_Pos |
35             DMA_SxCR_TCIE | DMA_SxCR_TEIE;
36     DMA1_Stream3->PAR = (uint32_t)&(SPI2->DR);
37     NVIC_EnableIRQ(DMA1_Stream3_IRQn);
38 
39     // SPI2 TX DMA
40     DMA1_Stream4->CR = 0b10 << DMA_SxCR_PL_Pos | DMA_SxCR_MINC | 0b01 << DMA_SxCR_DIR_Pos |
41             DMA_SxCR_TEIE;
42     DMA1_Stream4->PAR = (uint32_t)&(SPI2->DR);
43     NVIC_EnableIRQ(DMA1_Stream4_IRQn);
44 
45     // SPI idle timeout counter
```

```
46      RCC->APB1ENR |= RCC_APB1ENR_TIM7EN;
47      TIM7->DIER = TIM_DIER_UIE;
48      TIM7->PSC = SystemCoreClock / 1e6;
49      TIM7->ARR = 4000;
50      TIM7->CR1 = TIM_CR1_OPM | TIM_CR1_URS;
51      NVIC_EnableIRQ(TIM7_IRQn);
52
53      // generate pec table
54      for( uint16_t i = 0; i < 256; i++ ){
55          uint16_t rem = i << 7;
56          for( uint8_t j = 8; j; --j ){
57              rem <<= 1;
58              if( rem & 0x8000 ) rem ^= 0x4599;
59          }
60          pec_table[i] = rem;
61      }
62
63  }
```

Kodesnutt B.2: Funksjon for initialisering av SPI-grensesnittet. (spi.c)

Kø-håndtering

```
13 typedef struct {
14     int8_t status;
15     uint8_t len;
16     uint32_t time;
17     uint8_t tx[16];
18     uint8_t rx[16];
19     uint8_t * ret_data;
20     uint32_t * ret_time;
```

Kodesnutt B.3: Deklarasjon av kø-struktur. (spi.h)

```
83 uint8_t spi_queue_index[2]; // 0: queue, 1: current
84 spi_queue_struct spi_queue[spi_queue_len];
85
86 spi_queue_struct *spi_get_queue( void ){
87     spi_queue_struct *queue = &spi_queue[spi_queue_index[0]];
88 //  if( queue->status ) return NULL;
89     spi_queue_index[0] = (spi_queue_index[0] + 1 ) % spi_queue_len;
90     return queue;
91 }
```

Kodesnutt B.4: Funksjon for håndtering av kø-struktur. (spi.c)

Paritet

```
65 uint16_t spi_pec( uint8_t const * const data, uint8_t len ){
66 //  // Bør optimaliseres
67 //  // 9 us
68 //  uint16_t pec = 0x0010;
69 //  for( uint8_t i = 0; i < len; i++ )
70 //          for( int8_t j = 7; j >= 0; j-- )
71 //          pec = ( pec << 1 ) ^ ( 0x4599 * ( !!( pec & 0x4000 ) != ( 0x1 & ( data[i] >> j ) ) ) );
72 //  GPIOC->BSRR = GPIO_BSRR_BR_4;
73 //  return pec << 1;
74
75 // 900 ns
76     uint16_t pec = 0x10;
77     for( uint8_t i = 0; i < len; i++ ) pec = ( pec << 8 ) ^
78             pec_table[(uint8_t)((pec>>7)^data[i])];
79
80     return pec << 1;
81 }
```

Kodesnutt B.5: Funksjon for regning av paritetsverdier. (spi.c)

Dataoverføring

```
95 uint8_t spi_start( void ){
96     uint32_t prim = __get_PRIMASK();
97     __disable_irq();
98 
99     uint8_t ret = 0;
100    current_queue = &spi_queue[spi_queue_index[1]]; //load next queue
101    if( current_queue->status != 1 ) ret = 1;
102    else {
103        if( timeout ){
104            timeout = 0; // timer (4 ms)
105            GPIOB->BSRR = GPIO_BSRR_BR_12;
106            for( uint16_t i = 0; i < 600; i++ ) asm (""); // 600?
107            GPIOB->BSRR = GPIO_BSRR_BS_12;
108            for( uint16_t i = 0; i < 600; i++ ) asm ("");
109        }
110
111        GPIOB->BSRR = GPIO_BSRR_BR_12;
112
113        TIM7->EGR = TIM_EGR_UG;
114        TIM7->CR1 |= TIM_CR1_CEN;
115
116        spi_queue_index[1] = ( spi_queue_index[1] + 1 ) % spi_queue_len;
117
118        DMA1_Stream4->M0AR = (uint32_t)current_queue->tx;
119        DMA1_Stream4->NDTR = current_queue->len;
120        DMA1_Stream3->M0AR = (uint32_t)current_queue->rx;
121        DMA1_Stream3->NDTR = current_queue->len;
122
123        current_queue->status = 2;
124        current_queue->time = HAL_GetTick();
125
126        DMA1_Stream3->CR |= DMA_SxCR_EN;
127        DMA1_Stream4->CR |= DMA_SxCR_EN;
128    }
129    if( !prim ) __enable_irq();
130    return ret;
131 }
```

Kodesnutt B.6: Funksjon for sending av meldinger fra kø. (spi.c)

Avbrudd

```
165 spi_queue_struct *ret_queue;
166 void spi_ret( spi_queue_struct *queue ){
167     ret_queue = queue;
168     uint16_t temp1 = ( (uint16_t)( ret_queue->rx[10] ) << 8 ) | ret_queue->rx[11];
169     uint16_t temp2 = spi_pec( &ret_queue->rx[4] , 6 );
170     if( temp1 == temp2 ){
171         *ret_queue->ret_time = ret_queue->time;
172         for( uint8_t i = 0; i < 6; i++ )
173             *(ret_queue->ret_data + i) = ret_queue->rx[4+i];
174 
175         ret_queue->status = 4;
176     } else ret_queue->status = -1;
177 }
178 
179 // SPI Rx DMA
180 void DMA1_Stream3_IRQHandler( void ){
181     if( DMA1->LISR & DMA_LISR_TCIF3 ){
182         GPIOB->BSRR = GPIO_BSRR_BS_12;
183 
184         DMA1->LIFCR = DMA_LIFCR_CTCIF3 | DMA_LIFCR_CHTIF3;
185         DMA1->HIFCR = DMA_HIFCR_CTCIF4 | DMA_HIFCR_CHTIF4;
186 
187         spi_queue_struct *queue = current_queue;
188         queue->status = 3;
189 
190         spi_start();
191         if( queue->ret_data ) spi_ret( queue );
192     } else {
193         // kjøre håndtering
194     }
195 }
```

Kodesnutt B.7: DMA-avbrudd og returfunksjon (spi.c)

Oppstart av slavene

```
14 uint8_t cfgr_default[6] = { 0x05, 0x98, 0x46, 0xA4, 0x00, 0x10 };
15 uint16_t slave_update_timing[4] = { 500, 4500, 1400, 100 };
16 int8_t *ret[8];
17
18 uint8_t slave_init( uint8_t n ){
19     master.n_slaves = n;
20
21     spi_init();
22
23     // SPI2 1us variable time interrupt
24     RCC->APB1ENR |= RCC_APB1ENR_TIM6EN;
25     TIM6->DIER = TIM_DIER_UIE;
26     TIM6->PSC = SystemCoreClock / 2e6;
27     TIM6->CR1 = TIM_CR1_OPM | TIM_CR1_CEN;
28     NVIC_EnableIRQ(TIM6_DAC_IRQn);
29
30     ret[0]= spi( 0, 0x711, NULL, 0, NULL, 0, NULL ); // CLRCELL
31     while( *ret[0] < 3 );
32     ret[0] = spi( 0, 0x001, cfgr_default, 6, NULL, 0, NULL );
33     while( *ret[0] < 3 );
34     for( uint8_t i = 0; i < master.n_slaves; i++ ){
35         ret[i] = spi( i+1, 0x002, NULL, 0, &master.slave[i].cfgr[0], 8, &master.slave[i].time[0] );
36     }
37     while( *ret[0] < 3 );
38     uint8_t error;
39     for( uint8_t i = 0; i < master.n_slaves; i++ ){
40         for( uint8_t j = 1; j < 6; j++ ){
41             if( master.slave[i].cfgr[j] != cfgr_default[j] ){
42                 master.slave[i].status = 255;
43                 error = 1;
44                 break;
45             }
46         }
47         if( master.slave[i].status != 255 ) master.slave[i].status = 1;
48     }
49
50     return error;
51 }
```

Kodesnutt B.8: Funksjon for initialisering av slavemodulene. (slave.c)

Måling av cellenes temperatur og spenning

```
268 void TIM6_DAC_IRQHandler( void ){
269     TIM6->SR = 0;
270     switch( slave_update_stage ){
271     case 1:
272         slave_update2();
273         break;
274     case 2:
275         slave_update3();
276         break;
277     case 3:
278         slave_update1();
279         break;
280     case 4:
281         slave_update2();
282         break;
283     case 5:
284         slave_update3();
285         break;
286     case 6:
287         slave_update1();
288         break;
289     case 7:
290         slave_update2();
291         break;
292     case 8:
293         slave_update3();
294         break;
295     case 9:
296         slave_update4();
297         break;
298     }
299
300 }
```

Kodesnutt B.9: Avbruddsfunksjon for TIM6. (slave.c)

```
172     uint16_t v_min = UINT16_MAX, v_max = 0;
173     uint8_t v_min_id = 0, v_max_id = 0;
174     int32_t v_sum = 0;
175 
176     for( uint8_t i = 0; i < master.n_slaves; i++ ){
177         for( uint8_t j = 0; j < 12; j++ ){
178 
179             master.v[i*12+j] = (uint32_t)( master.v[i*12+j] * ( 1000 - master.conf.v_alpha ) + ( master.sl
180             if( master.v[i*12+j] < v_min ){
181                 v_min = master.v[i*12+j];
182                 v_min_id = i*12+j;
183             }
184             if( master.v[i*12+j] > v_max ){
185                 v_max = master.v[i*12+j];
186                 v_max_id = i*12+j;
187             }
188             v_sum += master.v[i*12+j];
189 
190 //          if( master.status.balancing ){
191 //              uint32_t cfgr = ( master.slave[i].cfgr[5] << 10 ) | ( master.slave[i].cfgr[4] << 2 );
192 //              cfgr >>= j;
193 //              offset[i][j] = 2200*!!( cfgr & (1<<2) ) - 1100*!!( cfgr & (1<<1) ) - 1100*!!( cfgr & (1<<3
194 //          } else offset[i][j] = 0;
195         }
196     }
197 
198     master.v_min = v_min;
199     master.v_min_id = v_min_id;
200     master.v_max = v_max;
201     master.v_max_id = v_max_id;
202     master.v_avg = v_sum / ( master.n_slaves * 12 );
203     master.v_tot = v_sum/100;
204 
205     int8_t t_min = INT8_MAX, t_max = INT8_MIN;
206     uint8_t t_min_id = 0, t_max_id = 0;
207     int16_t t_sum = 0;
208 
209     for( uint8_t i = 0; i < master.n_slaves; i++ ){
210         uint8_t k = 0;
211         for( uint8_t j = 0; j < 12; j++ ){
212             GPIOB->BSRR = GPIO_BSRR_BS_1;
213             master.t[i*12+j] = v2t( master.slave[i].temp[k] );
214             GPIOB->BSRR = GPIO_BSRR_BR_1;
215             if( master.t[i*12+j] < t_min ){
216                 t_min = master.t[i*12+j];
217                 t_min_id = i*12+j;
218             }
219             if( master.t[i*12+j] > t_max ){
220                 t_max = master.t[i*12+j];
221                 t_max_id = i*12+j;
222             }
223             t_sum += master.t[i*12+j];
224 
225             k+=2;
226             if( k == 12 ) k++;
227         }
228     }
229 
230     master.t_min = t_min;
231     master.t_min_id = t_min_id;
232     master.t_max = t_max;
233     master.t_max_id = t_max_id;
234     master.t_avg = t_sum / ( master.n_slaves * 12 );
```

Kodesnutt B.10: Signalbehandling for spennings- og temperaturverdier. (slave.c)

Strømmåling

```
79      uint32_t sum = 0;
80      for( uint8_t i = 0; i < master.conf.ts; i++ ) sum += i_unfiltered[ct][i] & 0xFFF;
81      i_fs = ( ( (float)sum / 10 - (float)master.conf.i_offset ) * (float)master.conf.i_gain ) / 100 / (floa
82      master.i_fs = i_fs;
```

Kodesnutt B.11: Signalbehandling for strømverdier. (master.c)

```
12 int16_t i_unfiltered[2][200];
13
14 void adc1_init( void ){
15     RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
16     GPIO_InitTypeDef GPIO_InitStruct;
17     GPIO_InitStruct.Pin = GPIO_PIN_2;
18     GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
19     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
20
21     RCC->AHB1ENR |= RCC_AHB1ENR_DMA2EN;
22     DMA2_Stream4->PAR = (uint32_t)&ADC1->DR;
23     DMA2_Stream4->M0AR = (uint32_t)i_unfiltered[0];
24     DMA2_Stream4->M1AR = (uint32_t)i_unfiltered[1];
25     DMA2_Stream4->NDTR = master.conf.ts;
26     DMA2_Stream4->CR = DMA_SxCR_DBM | 0b1 << DMA_SxCR_PSIZE_Pos | 0b1 << DMA_SxCR_MSIZE_Pos | DMA_SxCR_MIN
27             DMA_SxCR_TEIE | DMA_SxCR_EN;
28     NVIC_EnableIRQ(DMA2_Stream4_IRQn);
29
30     RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;
31     ADC->CCR = 0b11 << ADC_CCR_ADCPRE_Pos; // ( APB2CLK = 64M ) / 8 / 480 = 16.7 kHz
32     ADC1->CR2 = ADC_CR2_DDS | ADC_CR2_DMA | /*ADC_CR2_CONT | */ADC_CR2_ADON;
33     ADC1->SMPR1 = 0b111 << ADC_SMPR1_SMP11_Pos;
34     ADC1->SQR3 = 2 << ADC_SQR3_SQ1_Pos;
35 //  ADC1->CR2 |= ADC_CR2_SWSTART;
36 }
37
38 uint8_t ct;
39
40 void DMA2_Stream4_IRQHandler( void ){
41     if( DMA2->HISR & DMA_HISR_TCIF4 ){
42         DMA2->HIFCR = DMA_HIFCR_CTCIF4 | DMA_HIFCR_CHTIF4;
43         ct = !(DMA2_Stream4->CR & DMA_SxCR_CT);
44     }
45 }
```

Kodesnutt B.12: Konfigurasjon av AD-omformer og DMA til strømmåling. (adc1.c)

B.3 Kodesnutter fra GUI (Python)

Mottakelse og sending

```
4959 # KOMMUNIKASJON, oppstart
4960 try:
4961     port = 'COM6'
4962     baud = 115200 #9600
4963     self.tilstand = 's'
4964     self.update_num = 5
4965     self.serieport = serial.Serial(port=port, baudrate=baud, timeout=0.1, \
4966                      parity=serial.PARITY_EVEN, stopbits=serial.STOPBITS_ONE)
4967     if self.serieport.isOpen():
4968         print(self.serieport.name, 'er open')
4969     else:
4970         self.serieport.open()
4971     self.serie_traad = threading.Thread(target=self.seriekomm)
4972     self.serie_traad.start()
4973 except:
4974     print("Exception, no master connected to the assigned COM port.")
```

Kodesnutt B.13: Oppstart av kommunikasjon via serieport.

```
#KOMMUNIKASJON, avlesning av serieport
def seriekomm(self):
    try:
        tegn = [];
        data_c = [];
        data_v = [];
        data_t = [];
        while self.tilstand != 'x':
            tegn.append(self.serieport.read(1));
            x = 0;
            if tegn[0]==b'':
                tegn.pop(0)
            else:
                x = 0;
                for i in range(0,len(tegn)):
                    if tegn[i]==b'':
                        x = i;
                        break;
                if x:
                    if tegn[0] == b'c': # Avlesninig av strømverdi
                        data_c.append(int.from_bytes(b''.join(tegn[1:3]), \
                                      'little', signed=True));
                    if tegn[0] == b'v': # Avlesninig av spenningsverdi
                        temp_data = [];
                        for j in range(0,96):
                            if j >= x/2-1:
                                break;
                            temp_data.append(int.from_bytes(b''.join(tegn[(j*2+1):(j*2+3)]), \
                                      'little', signed=False));
                        data_v.append(temp_data);
                    if tegn[0] == b't': # Avlesninig av temperaturverdi
                        temp_data = [];
                        for j in range(0,96):
                            if j >= x-1:
                                break;
                            temp_data.append(int.from_bytes(tegn[j+1], 'little', signed=True));
                        data_t.append(temp_data);
                    tegn.clear();
                    if len(data_c) and len(data_v) and len(data_t):
                        celldata= [[[],[]],[[],[]],[[],[]],[[],[]],[[],[]],[[],[]],[[],[]],[[],[]]]
                        max_min_summarized = []
                        total_sum = []
                        for i in range(0,8): # Lagrer avlesning for videre bruk
                            celldata[i][0].append(data_v[-1][i*12:i*12+12])
                            celldata[i][1].append(data_t[-1][i*12:i*12+12])
                        # Kjører databehandling
                        max_min_summarized = self.max_min_verdier(celldata)
                        total_sum = self.sum_verdier(celldata)
                        # Oppdaterer visualisering i GUI
                        self.oppdaterGUI(data_c[-1], celldata, max_min_summarized, total_sum)
                        celldata.clear() # Tømmer data før ny måling mottas
                        data_c.clear()
                        data_v.clear()
                        data_t.clear()
    except:
        self.serieport.write(b'y')
        self.serieport.close();  # Steng ned
        print(self.serieport.name, 'er stengt. Exception!')
```

Kodesnutt B.14: Avlesning av data over serieport.

Databehandling

```
5075     # DATABEHANDLING, max og min verdier
5076     def max_min_verdier(self, celldata):
5077         max_cell_voltages = []
5078         min_cell_voltages = []
5079         max_cell_temp =[]
5080         min_cell_temp = []
5081         #cell_temps = []
5082         max_min_summarized = []
5083         diff_summarized = []
5084 
5085         for i in range(0,8):
5086              max_cell_voltages.append(max(celldata[i][0][0]))
5087              min_cell_voltages.append(min(celldata[i][0][0]))
5088              max_cell_temp.append(max(celldata[i][1][0]))
5089              min_cell_temp.append(min(celldata[i][1][0]))
5090              diff_summarized.append(max_cell_voltages[i]-min_cell_voltages[i])
5091         max_min_summarized.append(max_cell_voltages)
5092         max_min_summarized.append(min_cell_voltages)
5093         max_min_summarized.append(diff_summarized)
5094         max_min_summarized.append(max_cell_temp)
5095         max_min_summarized.append(min_cell_temp)
5096         max_min_summarized.append([])
5097         max_min_summarized[-1].append(max(max_cell_voltages))
5098         max_min_summarized[-1].append(min(min_cell_voltages))
5099         max_min_summarized[-1].append(max(diff_summarized))
5100         max_min_summarized[-1].append(max(max_cell_temp))
5101         max_min_summarized[-1].append(min(min_cell_temp))
5102         return max_min_summarized
```

Kodesnutt B.15: Databehandlingsfunksjonen *max_ min_ verdier*.

```
5104     # DATABEHANDLING, sum
5105     def sum_verdier(self, celldata):
5106         total_sum = []
5107         for i in range(0,8):
5108              total_sum.append(sum(celldata[i][0][0]))
5109         total_sum.append(sum(total_sum))
5110         return total_sum
```

Kodesnutt B.16: Databehandlingsfunksjonen *sum_ verdier*.

Visualisering av data

```
5112     # VISUALISERING, oppdatering av brukergrensesnitt
5113     def oppdaterGUI(self, data_c, celldata, max_min_summarized, total_sum):
5114         # Total voltage
5115         self.total_voltage_num.display(total_sum[-1]/10000) # Total voltage
5116         # Current
5117         self.current_num.display(data_c/100)
5118         # Pack voltage - OVERVIEW
5119         j=0
5120         for i in self.packLCDnum:
5121              i.display(total_sum[j]/10000)
5122              j+=1
5123         # Cell voltage - PACKS
5124         for i in range(0,8):
5125              k=0
5126              for j in self.cellLCDnum[i]:
5127                  j.display(celldata[i][0][0][k]/10000)
5128                  k+=1
5129         # Max/Min/diff - PACKS
5130         j=-5
5131         for i in self.max_min_LCDnum[0]:
5132              if j <= -3:
5133                  i.display(max_min_summarized[-1][j]/10000)
5134              else:
5135                  i.display(max_min_summarized[-1][j])
5136              j+=1
5137         # Max/min/diff voltage - CELLS
5138         for i in range(1,9):
5139              k=0
5140              for j in self.max_min_LCDnum[i]:
5141                  if k <= 2:
5142                       j.display(max_min_summarized[k][i-1]/10000)
5143                  else:
5144                       j.display(max_min_summarized[k][i-1])
5145                  k+=1
5146         # Bar display
5147         if self.update_num >= 5:
5148              print('Bar update.')
5149              self.update_num = 0
5150              j=0
5151              for i in self.pack_voltage_bars:
5152                   i.setValue(total_sum[j])
5153                   j+=1
5154              for i in range(0,8):
5155                   k=0
5156                   for j in self.cell_voltage_bars[i]:
5157                        j.setValue(celldata[i][0][0][k])
5158                        k+=1
5159         self.update_num +=1
5160         #Loaded cells, sjekker kun voltage
5161         antall_celler = [] # Uleste celler
5162         for i in range(0,8):
5163              antall_celler.append(celldata[i][0][0].count(0))
5164         self.label_5.setText('Loaded cells: '+ str(96-sum(antall_celler)))
5165         return
```

Kodesnutt B.17: GUI oppdateringsfunksjon *oppdaterGUI*.

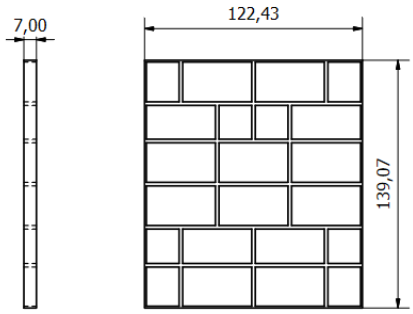
Vedlegg C

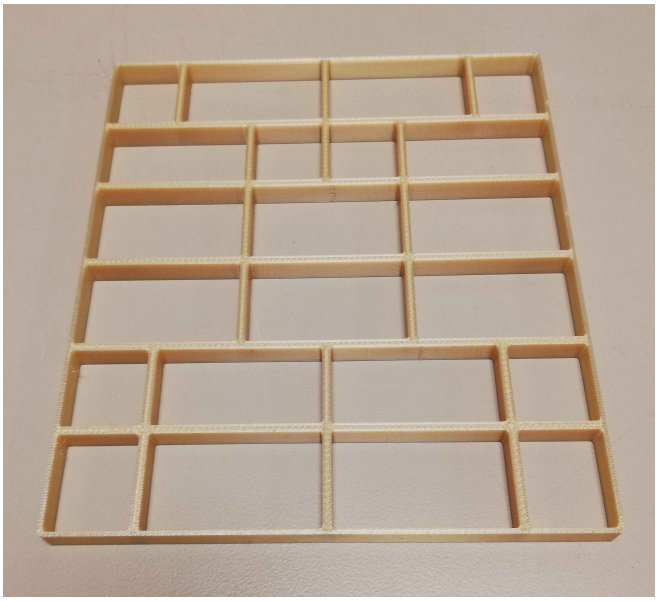
Konstruerte deler

I dette vedlegget viser aktuelle arbeidstegninger som er designet til konstruksjon av batteriet.

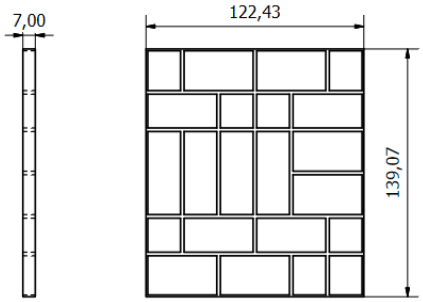
C.1 3D-printet skillevegg

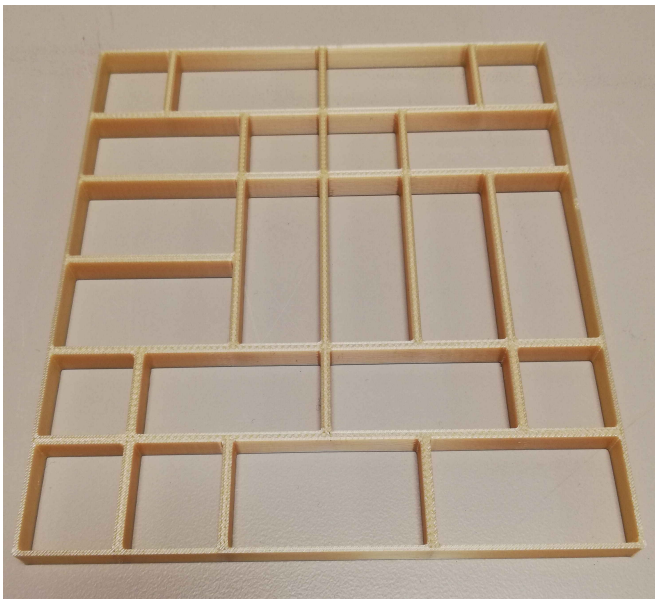
Batterisegment A-type (layout A)





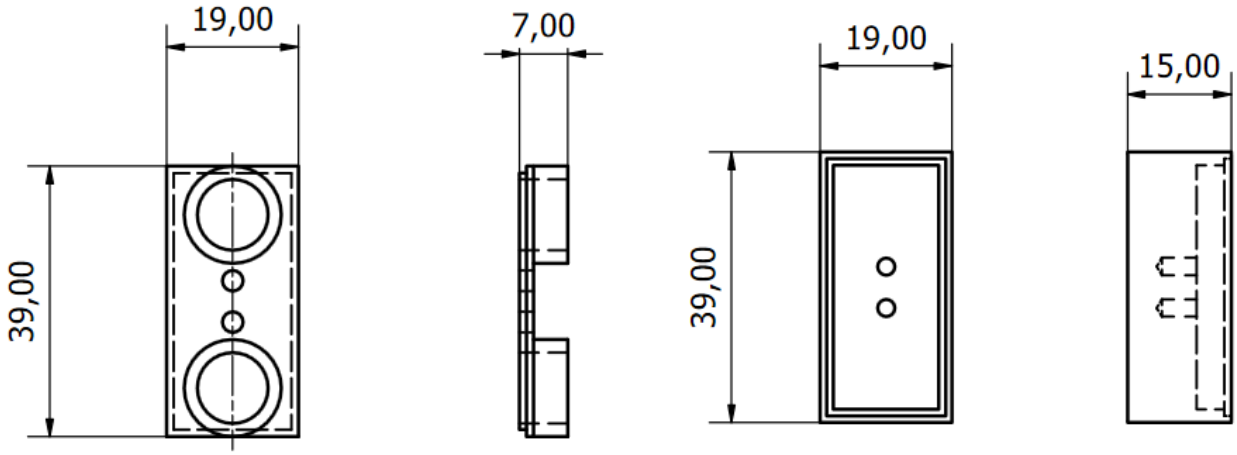
Batterisegment C-type (layout C)

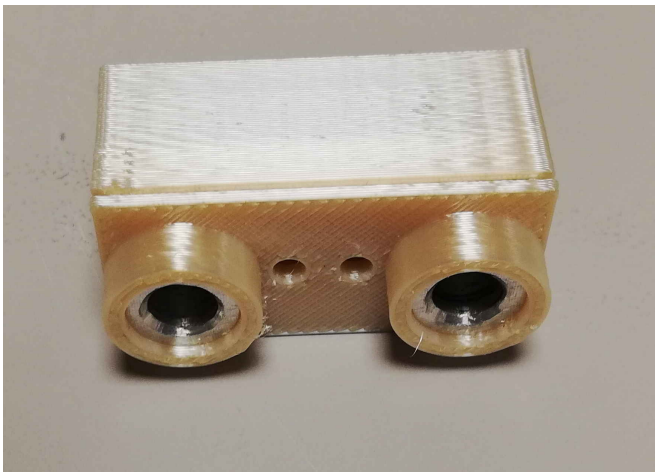


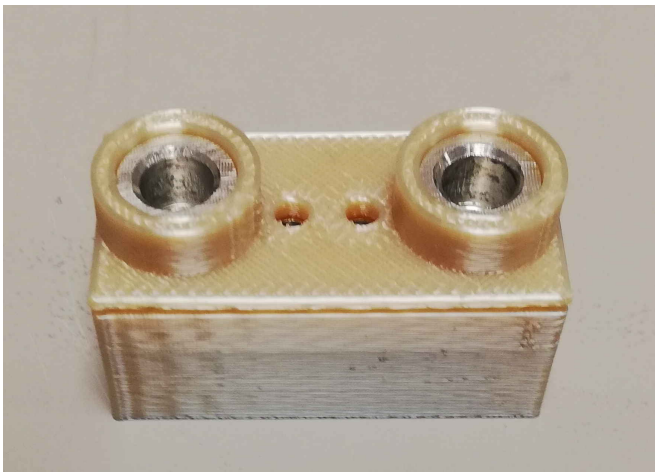


C.2 3D-printet vedlikeholdsplugger

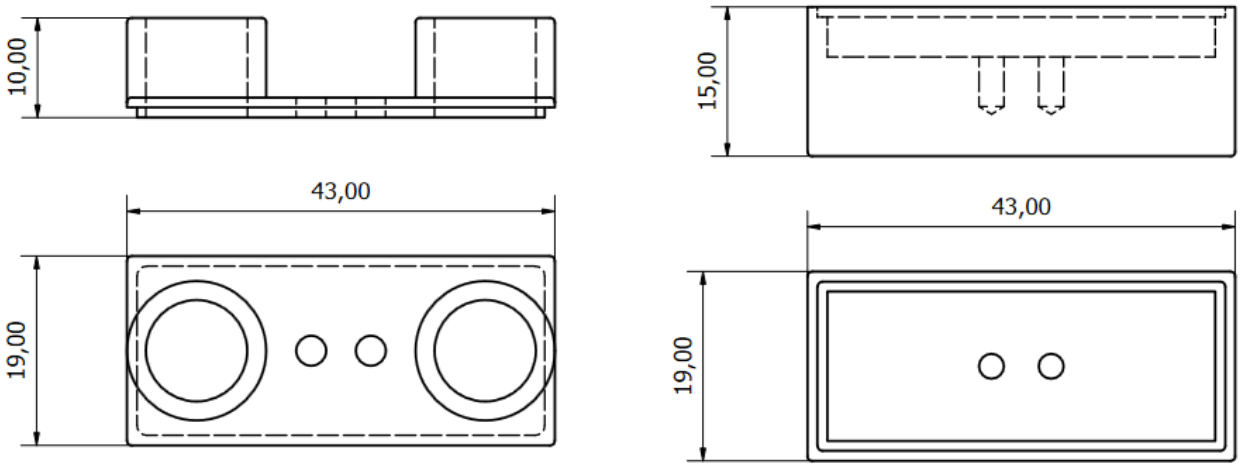
Korte vedlikeholdsplugger

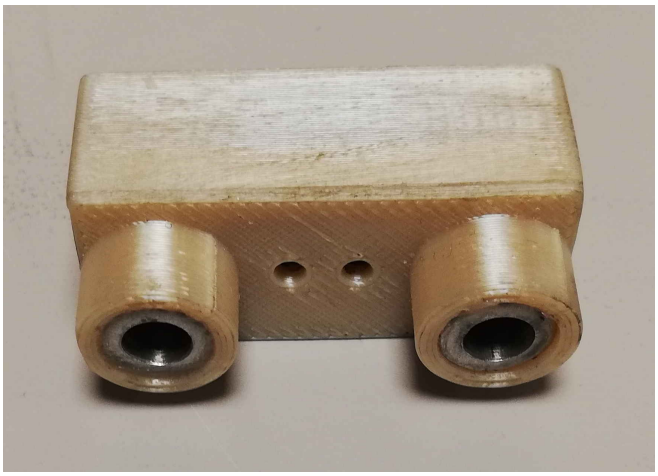


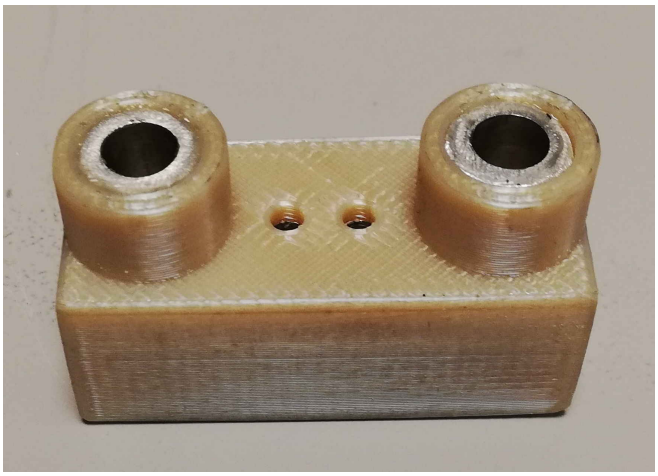




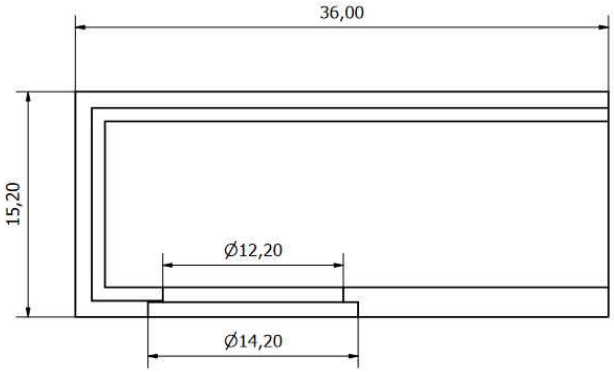
Lange vedlikeholdsplugger

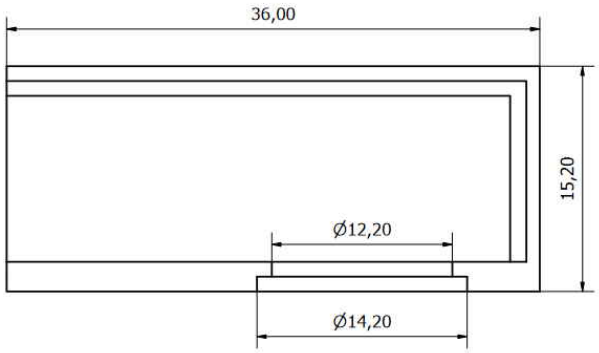


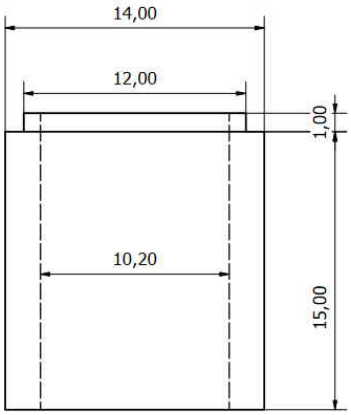


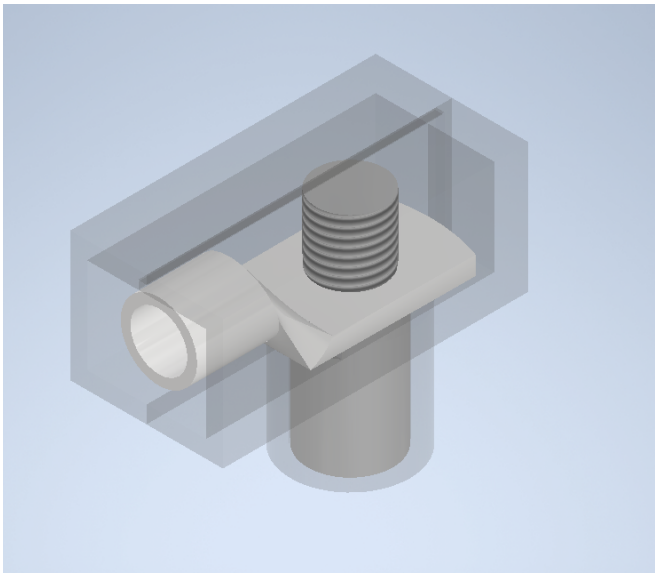


Kabel vedlikeholdsdplugg



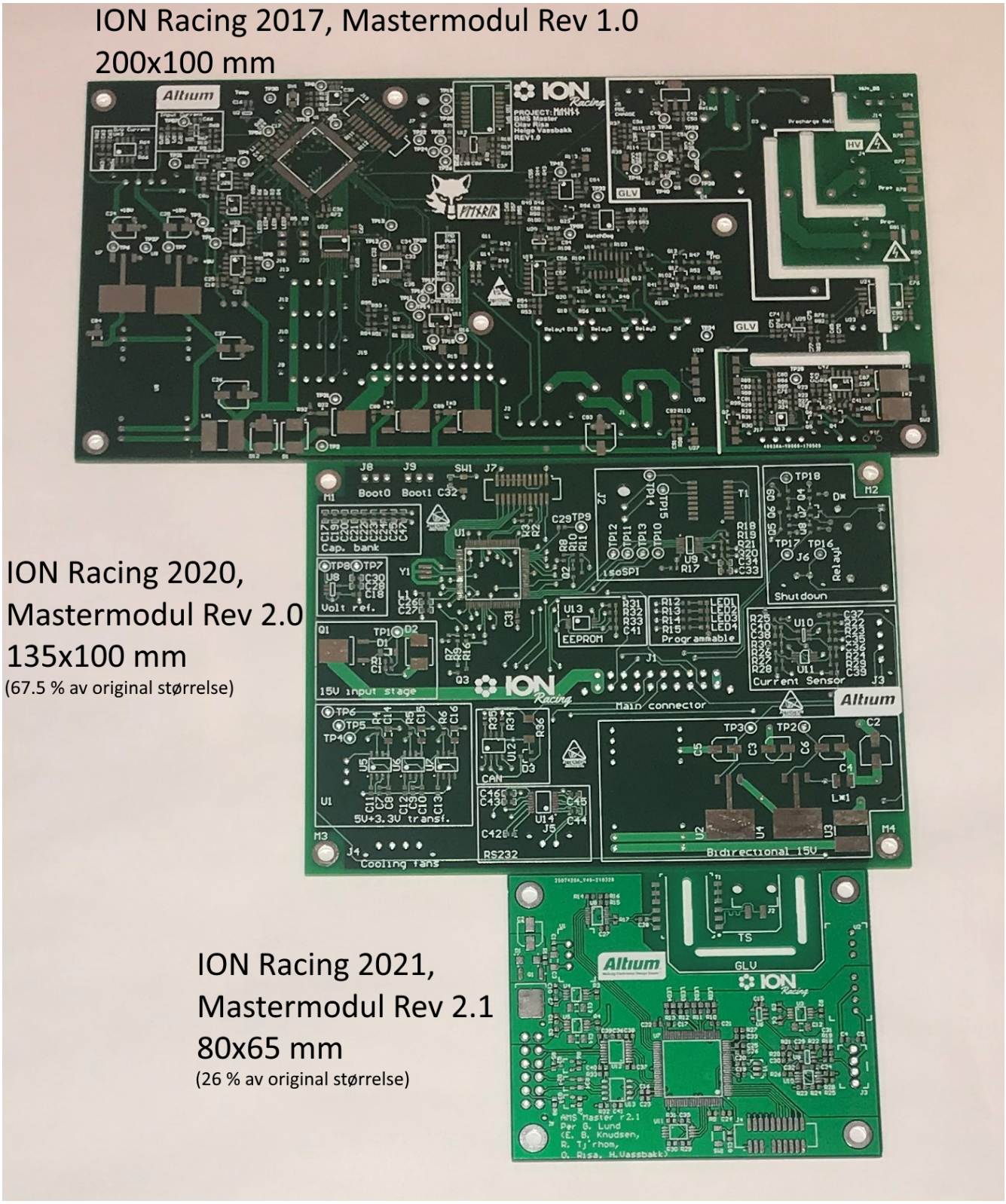




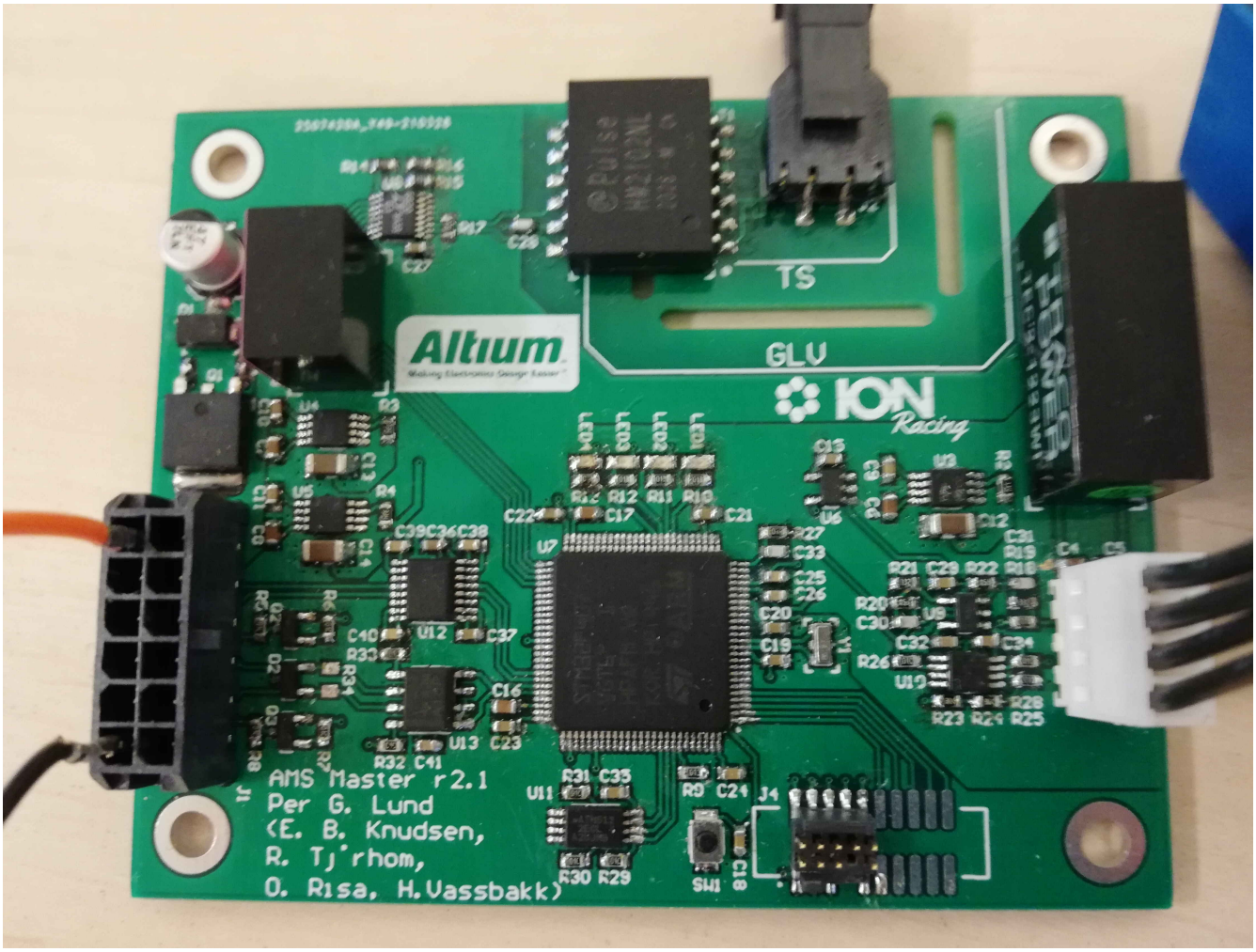


C.3 Kretskort

Mastermodulens kretskort

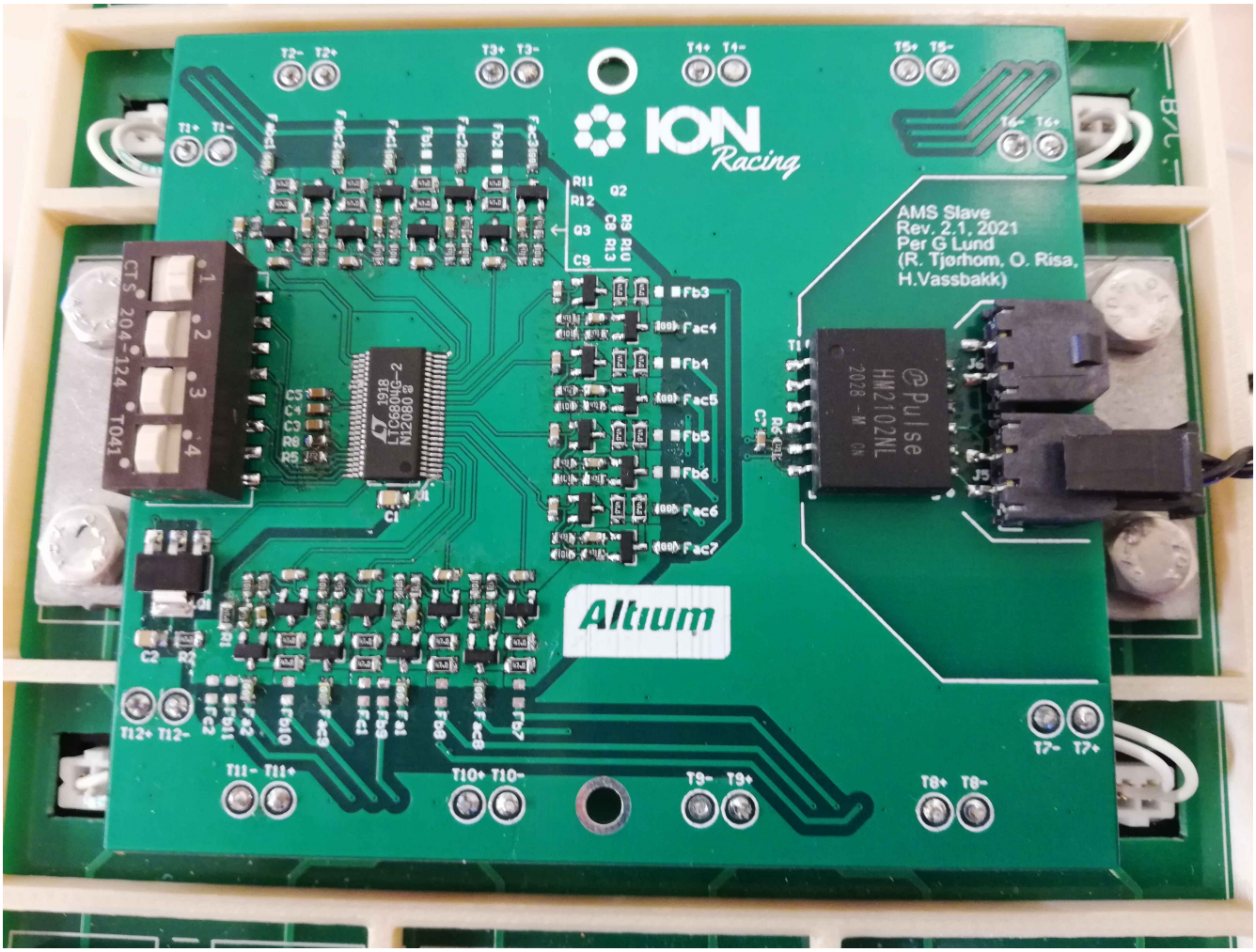


Figur C.3.1: Mastermodulens kretskort sammenlignet med tidligere design fra 2017 og 2020.



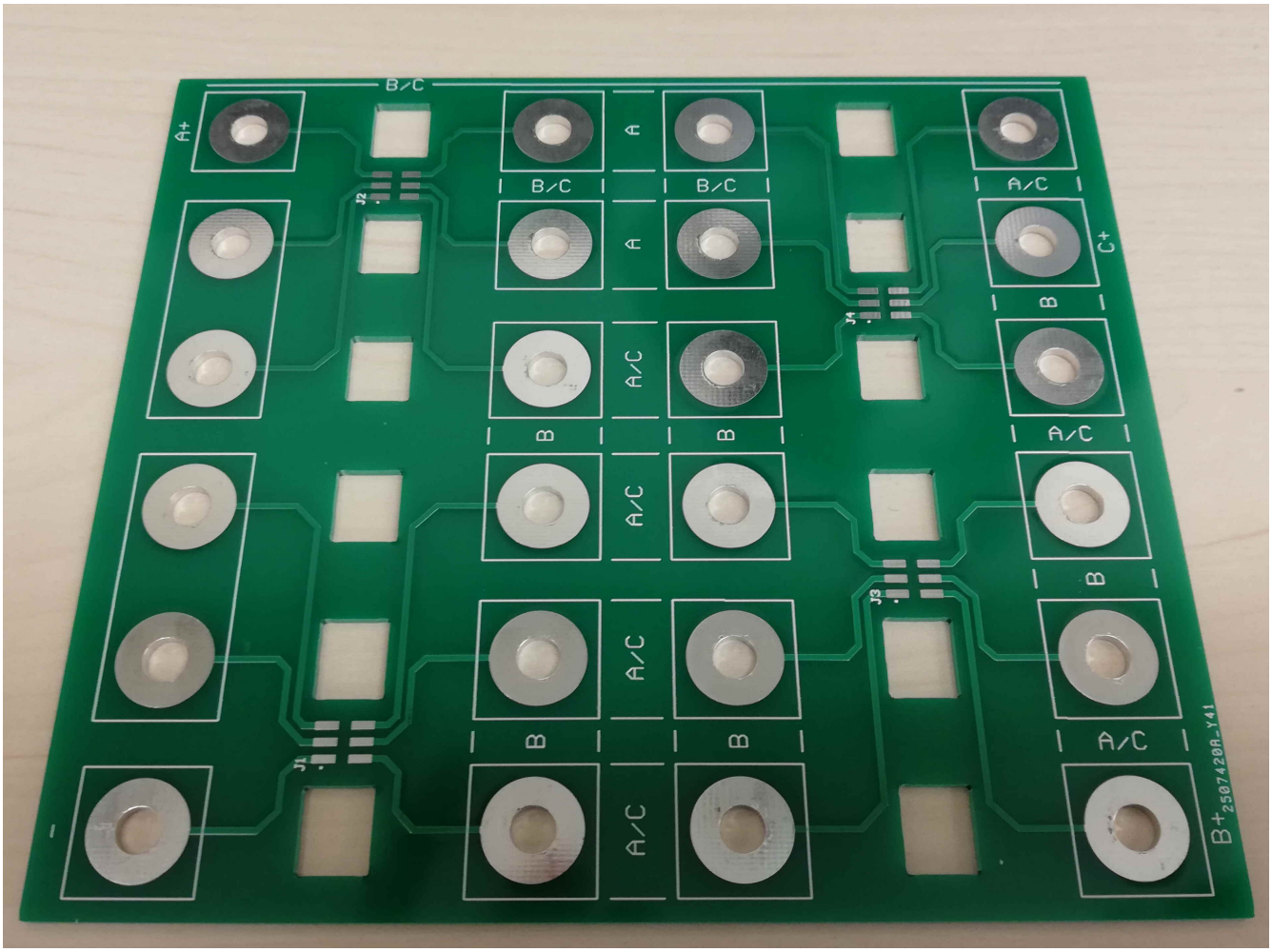
Figur C.3.2: Mastermodulens kretskort ferdig loddet.

Slavemodulenes kretskort



Figur C.3.3: Slavemodulens kretskort ferdig loddet og montert på batterisegment.

Tilkoblingskort til slavemodulene



Figur C.3.4: Slavemodulens tilkoblingskort.