SUSTAV UPRAVLJANJA BATERIJAMA NA RASPBERRY PI PLATFORMI ZA E-BUGGY

Livajić, Matej

S V E U Č I L I Š T E U S P L I T U

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Elektronika

MATEJ LIVAJIĆ

Z A V R Š N I R A D

SUSTAV UPRAVLJANJA BATERIJAMA NA RASPBERRY PI PLATFORMI ZA E-BUGGY

Split, srpanj 2022.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Elektronika

Predmet: Energetski elektronički pretvarači

Z A V R Š N I R A D

Kandidat: Matej Livajić

Naslov rada: Sustav upravljanja baterijama na Raspberry Pi platformi za e-buggy

Mentor: dr. sc. Marko Vukšić

Split, srpanj 2022.

SADRŽAJ

Sažetak

Sustav upravljanja baterijama na Raspberry Pi platformi za e-buggy

Rad opisuje predmetni problem, razvoj sustava upravljanja (praćenja) baterijama za e-buggy, vozila maksimalne snage 20 kW. Prvo se daje šira slika problematike i teorijska pozadina mjerenja. Potom se obrazlaže odabrani smjer rješenja u obliku modularnih senzorskih jedinica umreženih pomoću CAN-a. Centralni modul dalje prenosi podatke BeagleBone Black-u, robusnijoj (industrijskoj) zamjeni za Raspberry Pi. Operacijski sustav Debian na BeagleBone Black, malom računalu, omogućava instalaciju ekrana za grafičko sučelje s vozačem kao i prikaz podataka na udaljenom računalu preko VNC sustava.

Summary

Battery management system on Raspberry Pi platform for e-buggy

This thesis describes the problem (task), the development of a battery management (monitoring) system for e-buggy vehicle, with a maximum power of 20 kW. First, a broader picture of the problem and the theoretical background of the measurement is given. Then, the chosen direction of the solution in the form of modular sensor units networked using CAN is explained. The central module transfers the data to the BeagleBone Black, a more robust (industrial) replacement for the Raspberry Pi. The Debian operating system on the BeagleBone Black, a small computer, enables the installation of a screen for a graphical interface with a driver, as well as the display of data on a remote computer via the VNC system.

1. UVOD

Sustavi upravljanja baterijama (engl. *Battery management systems*, u nastavku BMS) su sustavi praćenja (i upravljanja) koji prate i reguliraju punjenje, pražnjenje baterija te njihovo skladištenje. Nekada se pod kraticom BMS smatra samo sustav nadziranja (praćenja) baterija. Zadatak BMS-a je optimalno korištenje baterija.

Elektrifikacijom i digitalizacijom društva sve je veća potreba za skladištenjem električne energije. Razni su načini spremanja energije, a baterije su dominantne kad je u pitanju mobilni spremnik energije. Razvojem Litij-ionskih i nasljednica, Litij-polimerskih baterija koje imaju znatno manju specifičnu masu naspram ostalih stvorili su se uvjeti za promjenu raznih koncepata u društvu. Energetska elektronika i njen razvoj drugi je značajni doprinos koji je pridonio postavljanju baterija kao mobilnog spremnika energije u sve više uređaja i strojeva.

Osnovno što svaki BMS sustav mora imati je mjerenje pojedinih fizikalnih veličina na baterijama koje čine baterijski sustav za neko vozilo i to je tema ovog rada.

Rad opisuje predmetni problem, razvoj sustava upravljanja (praćenja) baterijama za e-buggy, vozila maksimalne snage 20 kW. Prvo se daje šira slika problematike i teorijska pozadina mjerenja. Potom se obrazlaže odabrani smjer rješenja u obliku modularnih senzorskih jedinica umreženih pomoću CAN-a i daljnjeg toka informacija do grafičkog sučelja s korisnikom. Rješenje je prvo objašnjeno u hardverskom, a potom u softverskom smislu.

2. SUSTAV E-BUGGY

2.1. E-buggy

Opisivanje sustava *e-buggya* se radi da bi se dobio uvid u sve okolnosti odnosno uvjete i moguća ograničenja koja su potrebna za dizajniranje BMS-a. *Buggy* je izraz koji se koristi za lagano vozilo prilagođeno necestovnoj (engl. *off road*) vožnji sa često rešetkasto izvedenom karoserijom. U početku se izraz koristio za lagana vozila koja su vukli konji, a kasnije je primijenjen za lagane automobile [1]. Prefiks e sugerira da se radi o električno pogonjenom vozilu.

E-buggy je projekt na Sveučilišnom odjelu za stručne studije, Sveučilišta u Splitu koji se koristi kao baza za stručni i praktični rad studenata. Sustav kao što je vozilo, u ovom slučaju e-buggy, jedan je od kompleksnijih i izazovnijih sustava za projektiranje i konstruiranje. Na tržištu se za takve sustave, pogotovo ako se radi o komercijalnim proizvodima, natječu najbolji timovi inženjera iza kojih stoje godine iskustva i potpora tradicije koju poduzeće za kojeg projektiraju eventualno ima. Ovdje se radi o svojevrsnom prototipu i bazi za istraživanja te će se BMS projektirati kao blok sustava za koji su poznate neke ulazne i izlazne vrijednosti koje su u ovom trenutku poznate (baterijski sustav, naponi, pogon i sl.).

Slika 2.1. prikazuje nedavno stanje *e-buggya* gdje se mogu uočiti osnovne karakteristike vozila:

- 4 kotača od kojih su 2 stražnja pogonska, a prednja 2 se koriste za upravljanje
- 2 sjedala (vozač i suvozač)
- Lagana rešetkasta šasija

Slika 2.1. E-buggy u nedavnom nezavršenom stanju

2.2. Baterijski sustav

Prethodno odabrane baterije su LFP60150190 proizvođača Sunder battery. Baterijski sustav (u nastavku BS) čine 24 serijski spojene baterije. U tablici 2.1. su prikazane dostupne karakteristike:

Tablica 2.1. Podaci o baterijama [2]

Dimenzije	61,5 x 151,5 x 208,5 mm
Masa	Približno 3,2 kg
Broj punjenja (DOD 80%)	Više od 2000
Primjena	Solarni sustavi, solarne svjetiljke…
Kapacitet (nominalni)	100 Ah
Kapacitet (minimalni)	95Ah
Napon (nominalni)	Prosječno 3,2 V
Napon ispražnjenosti (engl. *cut-off voltage*)	2 V
Struja punjenja (standardna)	20 A
Napon punjenja (maksimalni)	3,65 V
Vrijeme punjenja	6 h približno
Maksimalna struja punjenja	100 A
Kontinuirana struja pražnjenja	100 A
Maksimalna struja pražnjenja	200 A
Temperatura punjenja	0~45℃
Temperatura pražnjenja	-20~60℃
Temperatura skladištenja (< 1 mjeseca)	-10~45℃
Temperatura skladištenja (< 6 mjeseci)	-10~25℃

Baterije su $LiFePO_4$ (litij željezo fosfat) što se odnosi na kemijski sastav katode (+) dok je anoda (-) grafitna s metalnom oblogom. Kao najvažnija karakteristika, a vezana za BMS najčešće se navodi osiguravanje da baterija ne padne ispod i iznad dozvoljenog napona te iznad dozvoljenih struja punjenja i pražnjenja. [2]

Najvažnije ulazno-izlazne veličine baterijskog sustava za BMS su:

- 24 serijski spojene LFP60150190 baterije ukupnog maksimalnog napona

  $$3{,}65 \cdot 24 = 87{,}6\ \text{V}$$

  odnosno nominalnog napona

  $$3{,}2 \cdot 24 = 76{,}8\ \text{V}$$

- Maksimalna struja punjenja od 100 A
- Maksimalna struja pražnjenja od 200 A

- Ostale karakteristike dane od proizvođača baterija

Slika 2.2. Prikaz baterijskog sustava u montaži

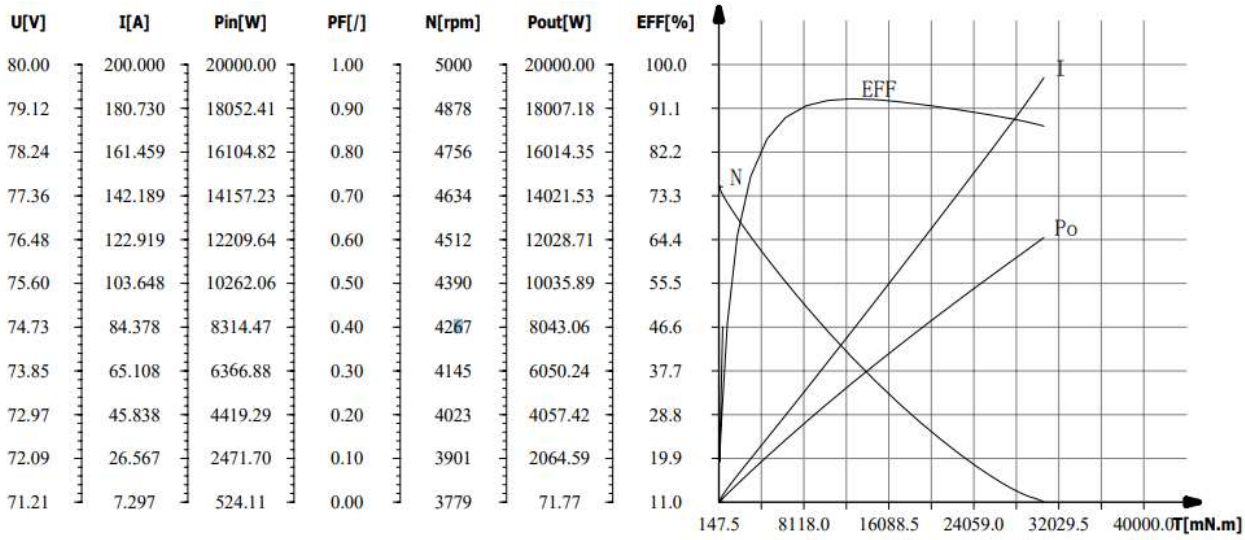
2.3. Pogonski sustav

2.3.1. Elektromotor

Pogonski sustav čine dva istosmjerna motora bez četkica (BLDC – engl. *Brushless direct current*) nominalnog napona 72 V i nominalne snage 10 kW. To je varijanta elektromotornog pogona koja se najčešće koristi u električnim motorima zbog dobrog omjera snaga/masa koji je u odnosu na klasične (s četkicama) istosmjerne motore postignut preko elektronički realiziranog komutatora. Osnovni podaci o motoru su prikazani u tablici 2.2. i na slici 2.3.

Tablica 2.2. Podaci o motoru [3]

Napon nominalni	72 V
Snaga nominalna	10 kW
Efikasnost nominalna	91 %
Brzina vrtnje	2000-6000 o/min
Masa	17,7 kg s hlađenje
Visina	170 mm
Promjer	206 mm



Slika 2.3. Prikaz najvažnijih karakteristika motora [3]

Najvažniji zaključci koji se mogu donijeti s obzirom na karakteristike sa slike 2.3.:

- Radi se o serijski uzbuđenom motoru
- Motor efikasno (>80% efikasnosti) u području tereta preko 3000 Nm
- U efikasnom području struje su veće od 25 A
- Maksimalna struja koju jedan motor može povući je 200 A što odgovara maksimalnoj struji pražnjenja baterije
- Maksimalni napon motora je 80 V što je više od 76,8 V (nominalni napon BS), a manje od 87,6 V (maksimalni napon BS)

Slika 2.4. Prikaz pločice elektromotora

2.3.2. Kontroler pogonskog sustava

Kontroler pogonskog sustava je VECTOR 500 Series BLDC Motor Controller istog proizvođača kao i motor (Golden motor). Kontroler je prikazan na slici 2.5.

Slika 2.5. VEC500 BLDC kontroler [4]

Ovaj kontroler je dizajniran za korišteni BLDC motor. Koristi FOC (engl. *Field Oriented Control*) algoritam u kojem se SVPWM (engl. *Sine Wave Pulse Width Modulation*) koristi kako bi se uspostavila sinusoidna struja trofaznom motoru. Kontroler preko nekoliko zatvorenih petlji kontrolira moment, tok (fluks) i brzinu. Proizvođač kao glavne karakteristike navodi:

- Fina regulacija
    - Kontrola momenta

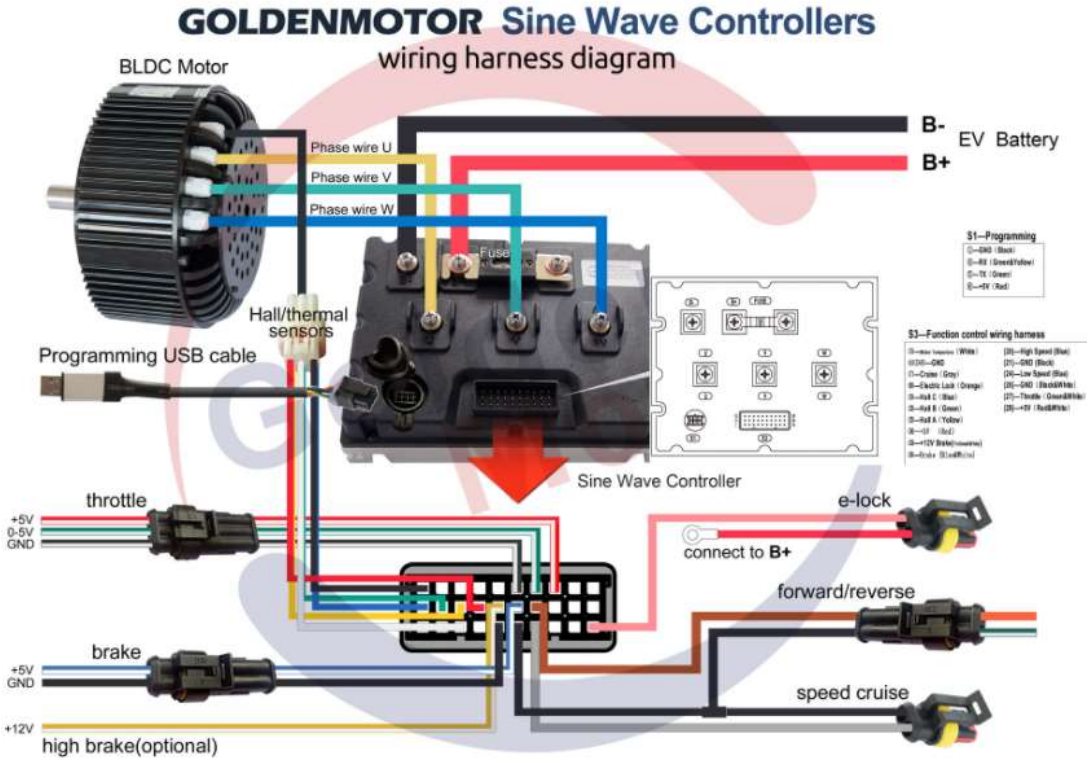
    - Fino startno ubrzanje
- Niska razina buke
    - Vektorska kontrola sinusoidne struje i glatki izlazni moment koji suzbija niskofrekventnu buku uzrokovanu fluktuacijama momenta
- PC programibilnost
- Odlične zaštitne funkcije
    - Kontrola signala
    - Prekostrujna zaštita
    - Naponska zaštita
    - Temperaturna zaštita

Tablica 2.3. prikazuje tehničke podatke i karakteristike.

Tablica 2.3. Tehnički parametri i karakteristike [4]

Nominalna područje napona	48V～96V
Nominalno područje struja	30A～200A
Nominalna snaga	1000～10000W
Kontrola motora	FOC
Struja mirovanja	20～40mA
Ograničenje brzine	Kontrolirano motorom i konfiguracijom
Metoda upravljanja	Direktno upravljanje momentom

Slika 2.6. prikazuje način spajanja kontrolera i ostalih dijelova sustava.



Slika 2.6. Shema spajanja [4]

3. TEORIJSKA POZADINA RJEŠENJA

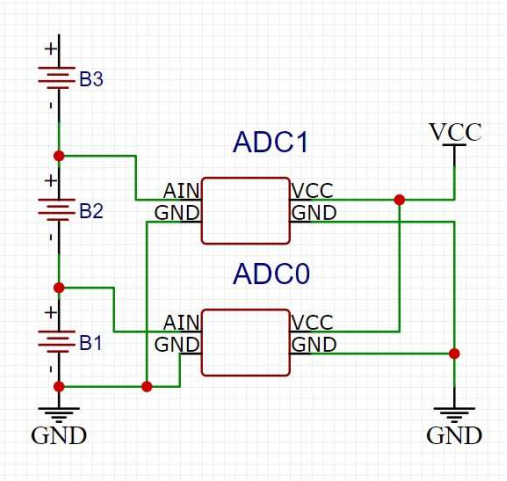
3.1. Mjerne metode

Kao što je u uvodu rečeno, za baterijske sustave kao varijable s najvećim utjecajem na stanje baterija, što se često poistovjećuje sa SOC (engl. *state of charge*), odnosno stanjem napunjenosti, su napon, temperatura i struja. U ovom poglavlju će se dati pregled općenitih, a i najčešće korištenih metoda mjerenja u baterijskim sustavima navedenih fizikalnih veličina te neke njihove prednosti i razlike. [5, 6]

3.1.1. Mjerenje napona

- Varijanta sa zajedničkim uzemljenjem

  Ova opcija je najjednostavnija. Mane su joj što je povoljna za konfiguracije manje baterija u seriji. Za razmatrani baterijski sustav koji je opisan u idućem poglavlju bilo bi potrebno da posljednji ADC ima mogućnost mjerenja napona od skoro 90 V. Takvo mjerno područje nepotrebno „troši“ razlučivost, a i nije toliko često što znatno povećava cijenu ADC.



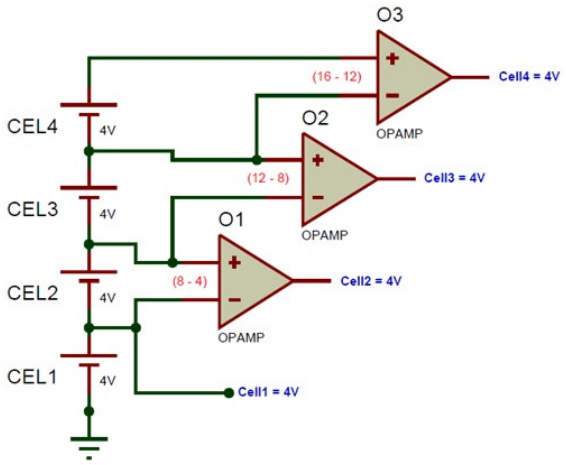
Slika 3.1. Varijanta sa zajedničkim uzemljenjem

- Varijanta sa uzastopno spojenim ADC

Takva varijanta nema problema s velikim mjernim područjem, ali ima potrebu za galvanskim odvajanjem kontakta spojenih na bateriju s ulaznim u ADC. Da nema galvanskog odvajanja negativni ulaz (GND) primjerice drugog ADC u nizu bi bio kratko spojen s negativnim ulazom prvog, a oni imaju naponsku razliku napona jedne baterije što bi rezultiralo kratkim spojem. Od raznih načina galvanskog odvajanja (trafo, opto, kapacitivno…) za ovu situaciju najpogodnija se čini opto-izolacija.

- Ostale metode (topologije)

Prikazane su osnovne varijante podijeljene s obzirom na uzemljenje svake mjerne jedinice. Pretraživanjem interneta mogu se pronaći još neke, manje pogodne varijante. Prva predlaže snižavanje mjernog napona dijeljenjem napona pomoću reaktivnih ili kapacitivnih djelitelja napona. Jedno od rješenja su uzastopno spojena operacijska pojačala prema slici 3.2.



Slika 3.2. Uzastopno spojena OP [5]

Kako se u ovom radu želi izraditi šire primjenjivo modularno rješenje, konačno rješenje je nešto na tragu ovog rješenja.

3.1.2. Mjerenje temperature

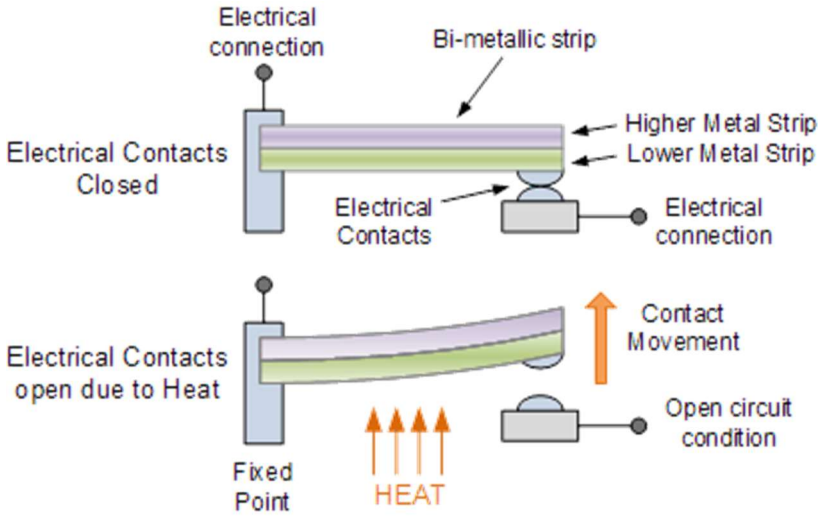
Mjerenje temperature [6] u baterijskim sustavima nema posebnih zahtjeva naspram standardnog mjerenja temperature objekata. Izvor [6] navodi osnovnu podjelu s obzirom na je li se radi o kontaktnom ili beskontaktnom mjerenju:

- Beskontaktno mjerenje

Ovakav tip mjerenja karakterizira nepostojanje fizičkog kontakta između mjernog senzora i objekta mjerenja. Najčešće se radi o senzorima koji mjere elektromagnetsko zračenje u infracrvenom području. Ovakvi sustavi su skuplji i s obzirom na prostorne uvjete nemaju smisao u primjeni kod baterijskih sustava.

- Kontaktno mjerenje
    - Bimetalni termostat

      Ovi senzori se zasnivaju na fizikalnom fenomenu različite promjene dimenzije s promjenom temperature odnosno na različitim koeficijentima toplinske dilatacije dvaju materijala. Takvi materijali, kada su međusobno povezani u traku i sl. uzrokuju savijanje trake u jednom smjeru kao na slici.

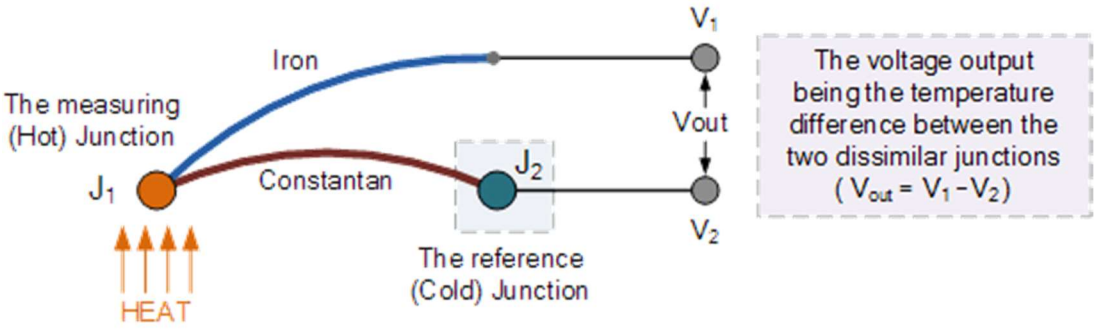


Slika 3.3. Bimetalni termostat [6]

Ova pojava je često korištena u električnim uređajima, ali češće za zaštite (upali – ugasi) nego za samo mjerenje jer ih karakterizira velika histereza. Također, potrebno je mjeriti pomak kako bi se u konačnici dobila temperatura što dodatno komplicira problematiku.

    - Termopar

Mjerenje se zasniva na fenomenu različitih napona na stezaljkama vodiča od različitih materijala na različitim temperaturama što se još naziva termoelektrični efekt.



Slika 3.4. Termopar [6]

Mana ovakvih sustava su male promjene napona od par mV/(10 °C) što zahtjeva pojačanje signala. Primjenjivi su za baterijske sustave, ali zahtijevaju veći utrošak vremena na umjeravanje.

- Termistor

Termistori koriste fenomen promjene otpora s promjenom temperature. Svi otpornici mijenjaju otpor s promjenom temperature, ali kod termistora to je izraženije, od kojih se najviše koriste NTC (engl. *negative temperature coefficient*) odnosno otpornici s negativnim temperaturnim koeficijentom. To znači da takvi otpornici s povećanjem temperature smanjuju otpor. Postoje razni načini spajanja NTC otpornika pri mjerenju, ali najčešće se izvode u serijskom spoju s nekim drugim otpornikom gdje je podijeljeni napon izlazni i predstavlja ulaznu vrijednost u matematičku funkciju koja na izlazu daje temperaturu. Osim NTC otpornika za preciznije mjerenje se koriste PTC otpornici zbog linearnije naponsko – otporne karakteristike.

Slika 3.5. Česti izgled termistora [6]

3.1.3. Mjerenje struje

Osnovna podjela [7] se vrši s obzirom na fizikalnu pojavu prema kojoj pojedina vrsta vrši pretvaranje struje u analogni signal. Tako razlikujemo *shunt* senzore i senzore koji se zasnivaju na Hallovom efektu. Postoje još razni principi mjerenja struje, ali ovi u nastavku imaju ozbiljniju primjenu u industriji električnih vozila:

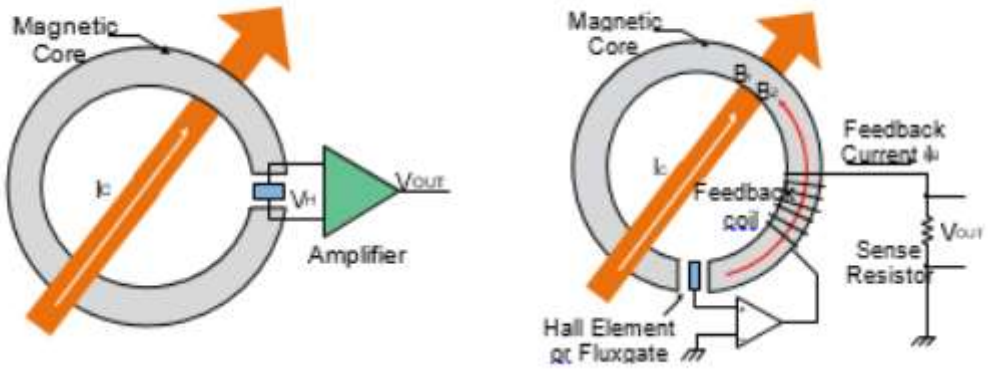
- *Shunt* senzori

  Fizikalna osnova im je Ohmov zakon. To su zapravo precizni otpornici čiji otpor pomnožen s padom napona na njima daje struju. Iz navedenih karakteristika se može zaključiti da se češće koriste za manje struje (<50A) jer bi inače rasipanje energije bilo veliko. Sa sve većim napretkom energetske elektronike mane ovakvih senzora su sve manje i sve im je veća primjena.

- Hallov efekt senzori [8]

  Ime su dobili po pojavi razlike napona duž električnog vodiča koji je okomit na električnu struju u vodiču i na magnetsko polje okomito na struju. Osnovna podjela im je na senzore otvorene (engl. *open-loop*) i zatvorene (engl. *closed-loop*) petlje. Zapravo senzor zatvorene petlje ima povratnu vezu u sustav realiziranu tako da se pusti struja kroz sekundarnu zavojnicu koja stvara magnetsko polje suprotnog smjera od onog kojeg je stvorila mjerena struja. Ove razlike rezultiraju određenim karakteristikama pa tako senzori zatvorene petlje se koriste u mjerenjima gdje je

potrebna veća preciznost, gdje ima više šumova i temperaturnih razlika. Senzori otvorene petlje se češće koriste ponajviše zbog povoljne cijene, manjih dimenzija itd.



Slika 3.6. Senzor otvorene (lijevo) i zatvorene (desno) petlje [8]

3.2. Digitalizacija signala

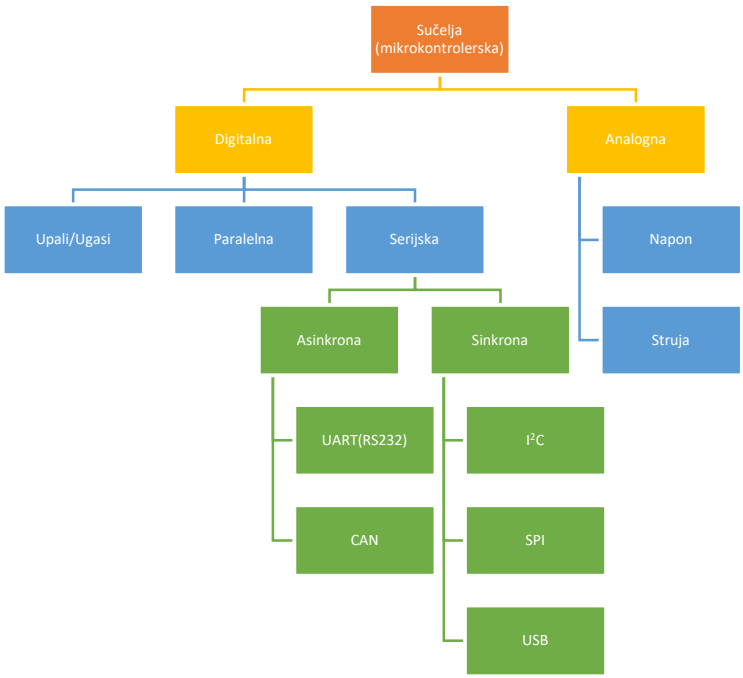
Pod digitalizacijom signala smatra se pretvaranje analogne veličine (signala) u digitalni oblik. Proces se naziva ADC (nekada A/D) kao i pretvarači koji ga vrše. Na tržištu postoje razni IC oblici pogodni za ADC. Digitalizacija analognog signala se vrši kroz tri postupka [9]:

- Uzorkovanje (uzimanje uzorka analognog signala)
- Kvantiziranje (pridruživanje diskretnih vrijednosti uzorcima signala)
- Kodiranje (izražavanje vrijednosti kvantiziranih uzoraka u binarnom br. sustavu)

Slika 3.7. Proces analogno digitalna pretvorba [9]

3.3. Mikrokontrolerska sučelja i komunikacija

Sučelje (engl. *interface*) označava fizičku i funkcionalnu vezu koja se uspostavlja između dva uređaja, uređaja ili sustava koji djeluju neovisno jedan o drugom. Postoje razne vrste mikrokontrolerskih sučelja, a dijele se prema stablu na slici 3.8. Slika prikazuje neka sučelja koja se mogu koristiti između mikrokontrolera, ali najčešće se koriste serijska, dok se ostala više koriste kod sučelja između mikrokontrolera i senzora, mikrokontrolera i uređaja i sl.



Slika 3.8. Mikrokontrolerska sučelja [10]

Analogna sučelja su najčešće kod senzora određenih fizikalnih veličina. Naponski signal (sučelje) može biti izlaz senzora struje koji radi na principu *Hallovog* efekta, strujni signal recimo izlaz fototranzistora koji mjeri razinu svjetlosti.

Digitalna sučelja se dijele na paralelna i serijska. Posebnu vrstu predstavljaju Upali/ugasi koja zapravo mogu biti i paralelna i serijska, ali kako predstavljaju posebnu skupinu u kojoj je informacijska razmjena mala razvrstani su na ovaj način. Jednostavni oblik digitalnog sučelja upali/ugasi je oblik gdje se recimo očitava veličina s mikroprekidača (visoko i nisko stanje), Paralelna komunikacija danas još postoji, ali se sve manje koristi zbog potrebe za većim brojem vodiča, a s manjkom nekih prednosti nad serijskom komunikacijom. Radi se

zapravo o dva koncepta u kojem se za paralelnu prostorna dimenzija „cijepa“ , a u serijskoj se to radi vremenskoj dimenziji. To je malo apstraktniji pogled na razlike.

Asinkrone [11] komunikacije karakterizira nepostojanje konstantnog prijenosa podataka po jedinici vremena što znači da neki blok podataka ne određuje vrijeme nego je blok upisan u informacijski niz, najčešće se koriste start i stop bitovi i sl.

UART (engl. *universal asynchronous receiver-transmitter*) je skraćenica za univerzalni asinkroni prijemnik-odašiljač. To je oznaka za uređaje koji komuniciraju po RS-232 standardu za 12 V sustave i RS-485 za 5 V sustave. Ti protokoli su se u početku nametnuli i danas se koriste u većini uređaja i situacija za koje su karakteristike takvog prijenosa pogodne. Ovaj tip komunikacije je dvosmjeran, asinkron i koristi dvije žice uz uzemljenje. Brzina prijenosa je relativno spora, od 300 bps to 115200 bps.

CAN (engl. *controller area network*) je sučelje za povezivanje u male mreže. Najčešće se koristi u automobilskoj i sličnim industrijama gdje je postalo standard. Tipično koristi upletenu paricu i uzemljenje. Brzine prijenosa su veće nego kod UARTA i kreću se od 10 kbps do 1 Mbps. Oblik podataka je također propisan kao i kod UARTA, a uz start i stop bitove koristi bitove za za otkrivanje pogreške i korekciju mogućnosti tako da je jako pouzdana što je bitno kod primjena za koje se koristi.

I2C ili I^2C je skraćenica od inter-integrirani krug (engl. *interintegrated circuit*). To je serijski sinkroni protokol za manje brzine prijenosa. Jednostavan je, dvosmjeran i koristi samo dvije žice. Koristan je za situacije gdje je potrebna komunikacija sa većim brojem (do 128) uređaja odnosno sa brojem adresa koje se mogu pridijeliti unutar 7 bitova. Na brzinu utječu kvaliteta žice, vanjski šumovi itd.

SPI (engl. *Serial peripheral interface*) je skraćenica za serijsko periferno sučelje. To je protokol za veće brzine prijenosa (do 20 Mbps) koji koristi četiri žice, dvije za dva smjera podataka, treća za sinkronizaciju (*clock*) i četvrta za kontrolu *slave*-a.

4. HARDVERSKO RJEŠENJE

Prije objašnjenja oblika realizacije postavljenih ciljeva (zadataka) bitno je dati pregled rješenja s obzirom na tržište elektroničkih komponenti i najčešćih oblika realizacija istih sustava. Automobilska industrija najčešće koristi cilindrične 18650 (18 mm promjer, 65 dužina) i 21700 (21 mm promjer, 70 dužina) ćelije posložene u odgovarajuću konfiguraciju. To im omogućava prostornu prilagodbu koja je zahtjevnija kod vozila serijske proizvodnje koja moraju imati bolje iskorišten prostor. Također omogućava bolju prilagodbu potrebama. Spomenute baterije se nekada razlikuju dvostruko ili trostruko u smislu kapaciteta i maksimalne struje za isti volumen. Takav sustav ima redundantnost odnosno kada jedna od ćelija u paraleli izgubi kapacitet ne gubi ga toliko cijela paralela. Te izvedbe u konačnici i ne definiraju toliko sami BMS jer se ćelije u paraleli svakako u većini slučajeva same balansiraju i čine relativno stabilan podsustav.

Neki od IC-a na tržištu koji se nude za ispunjenje navedenih zahtjeva su primjerice BQ79654 od Texas Instruments-a. Radi se o 14 serijskom baterijskom čipu za balansiranje i praćenje napona svake ćelije, ukupne struje i temperature (jedno mjesto). Čip ima mogućnost paralelnog vezivanja što bi za 2 čipa dalo mogućnost spajanja na sustav od 28 baterija u seriji što bi bilo dovoljno. Na tržištu nema puno alternativa ni proizvođača ovakvih čipova osim spomenutog TI i Maxim Integrated-a. Veliki problem u trenutku pisanja rada predstavlja nemogućnost nabave ovih čipova i/ili čipova koji su kompatibilni s njima kao njihovi kontroleri i slično. Prema Forbesu 8,1 milion automobila u razdoblju od 2021. do 2023. neće biti proizvedeno zbog manjka čipova. [12]. Takva situacija traži alternativu koja u konačnici traži više rada i istraživanja, ali omogućava veću konfigurabilnost BMS-a.

Jedna od ideja ovog rada je sva mjerenja izvoditi modularno tako da ovaj sustav bude primjenjiv i za različite konfuguracije spoja baterija odnosno veći ili manji broj baterija spojenih u seriju. Sustav je realiziran na način da svaka baterija ima svoj modul ukupnog BMS sustava. Svaki modul mjeri napon i temperaturu ćelije, a struja se mjeri na glavnom (*master*) modulu. Mjerenje temperature na svakoj ćeliji je pogodno u ovoj situaciji jer nema više baterija u paraleli koje se samobalansiraju na način da kada se jedna ugrije poraste joj i unutarnji otpor i na taj način se izbalansira s drugima. Svaki modul ima po jedan digitalni izlaz i jedan analogni ulaz. Digitalni izlaz se može iskoristiti za balansiranje, a analogni ulaz

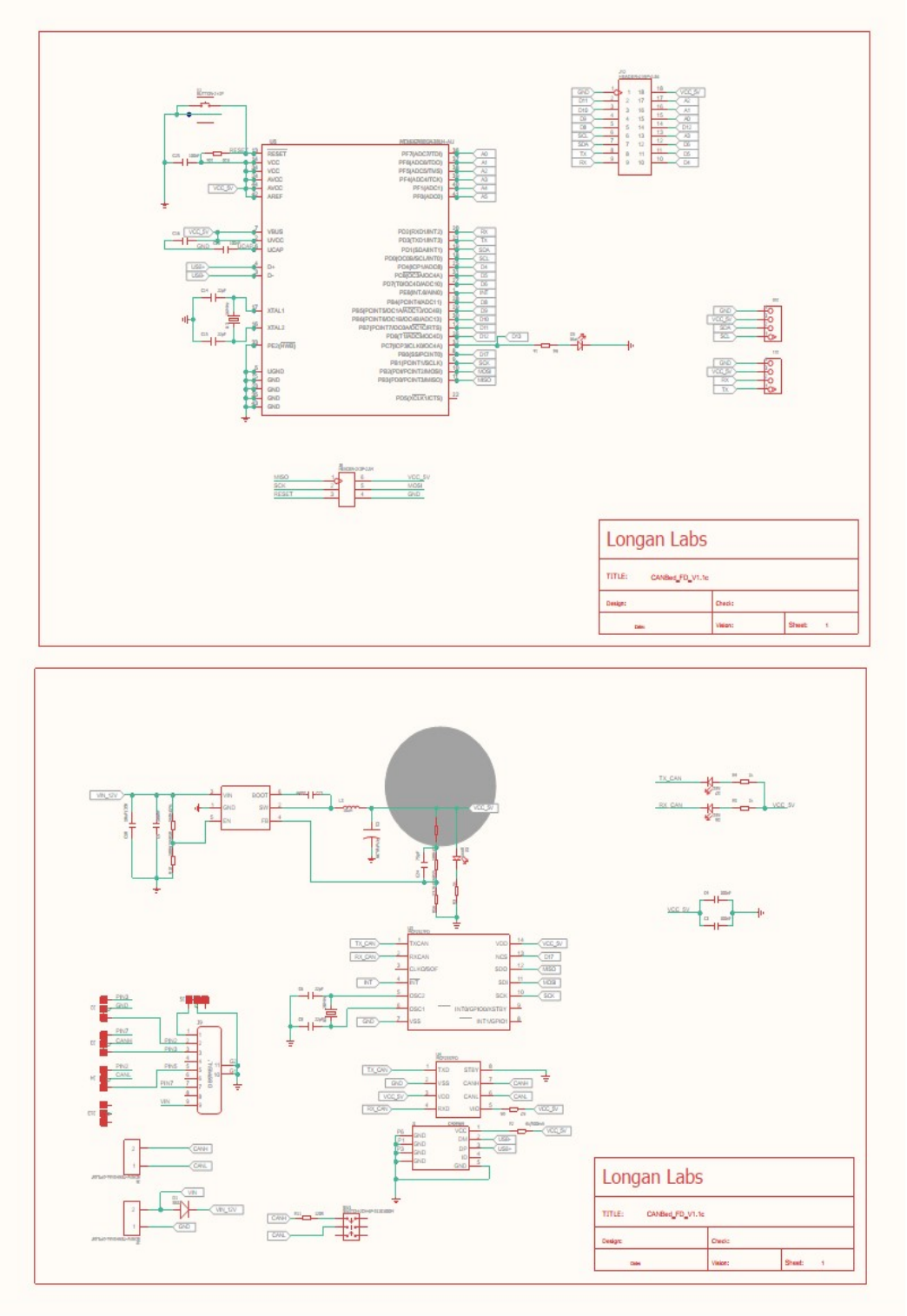
za primjerice mjerenje tlaka ili neke druge veličine na bateriji. Jedan od planova za ovaj baterijski sustav je razvoj brzog punjača pa je ovakva konfigurabilnost povoljna. Svi moduli komuniciraju kao virutalni *slave* uređaji s centralnim virtualnim *master* kontrolerom. CAN komunikacija nema pravi oblik *master-slave* odnosa već se on realizira prozivanjem (softverski). Centralni *master* kontroler dalje preko UART komunikacije podatke šalje BeagleBone Black SBC (engl. *Single-board computer*) koji te podatke prikazuje bežično na virtualnom ekranu (VNC sustav). Dan je kratki uvod u funkcioniranje sustava, a u idućim potpoglavljima je detaljnije objašnjeno po dijelovima.

4.1. Ćelijski modul

Ćelijski modul sastoji se od dvije međusobno povezane pločice čije su sheme prikazane na slikama 4.2. i 4.5.

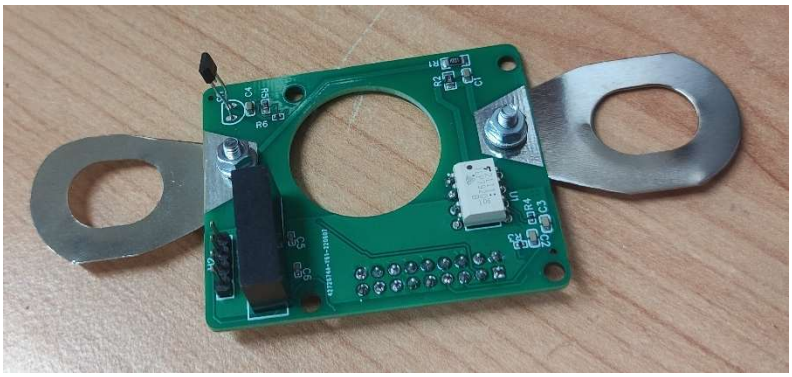
Slika 4.2. prikazuje shemu CANBED FD [12] razvojne pločice. Pločica ima Atmega32U4 mikrokontroler (kompatibilan s Arduino Leonardom) i MCP2517FD kao CAN kontroler i MCP2542 kao CAN *reciever*. Slika 4.1. prikazuje kontroler u fizičkom obliku.

Slika 4.1. CANBED FD [13]



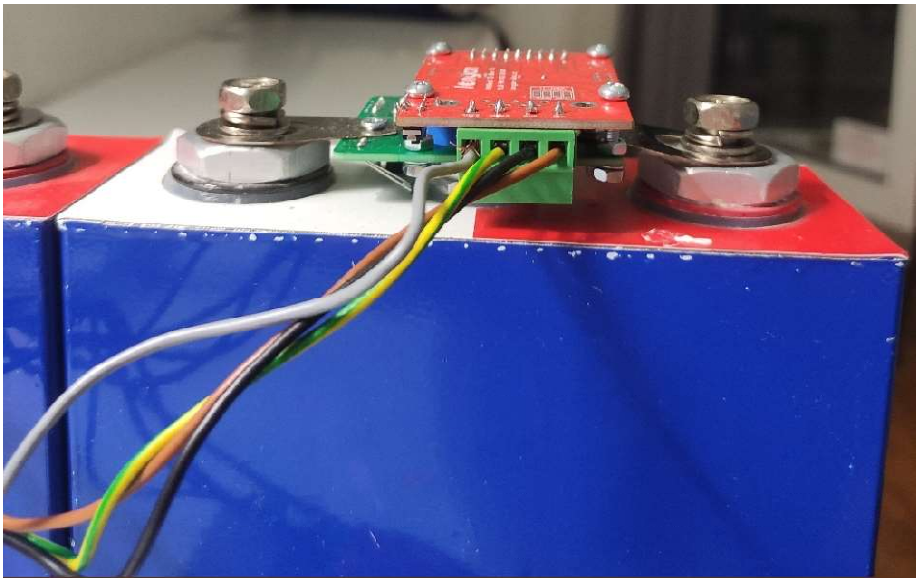
Slika 4.2. CANBED FD Shema [13]

Slika 4.3. prikazuje fizički izgled izrađene pločice na kojoj se nalaze elementi prema shemi na slici 4.5.

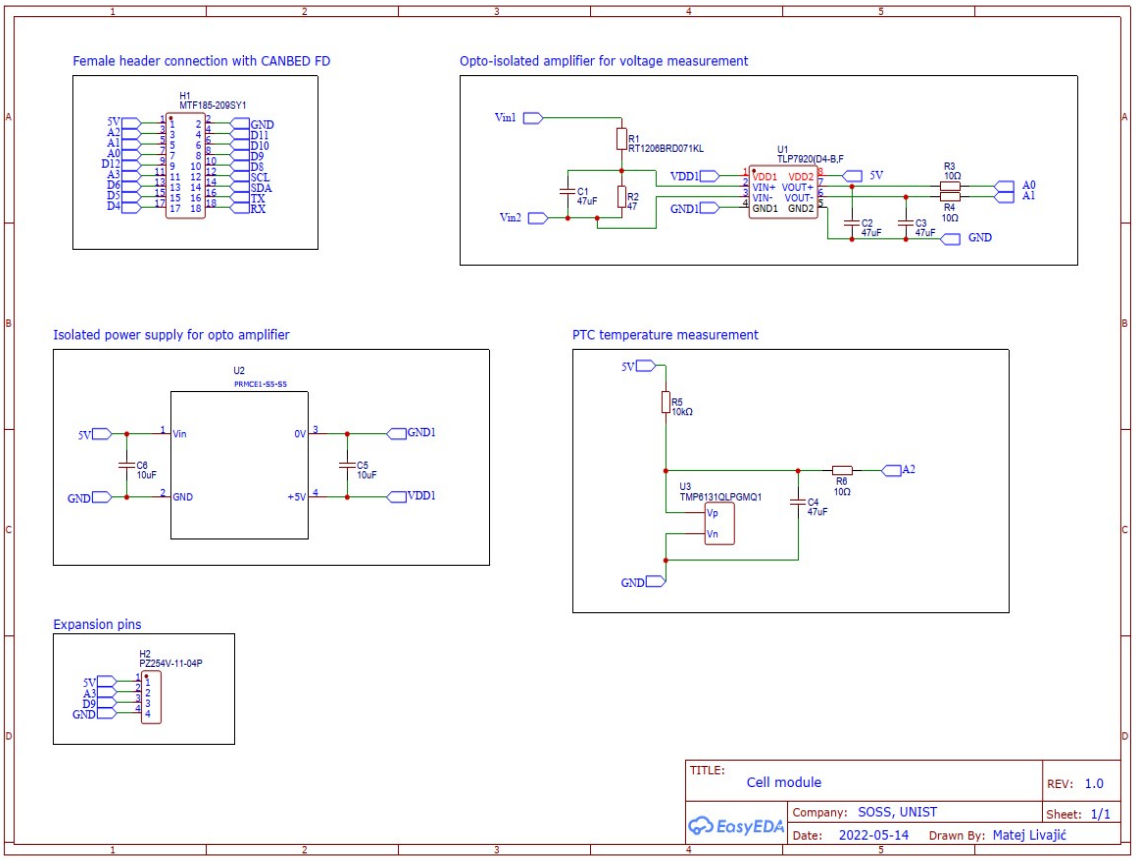


Slika 4.3. Izgled druge pločice ćelijskog modula

Dizajniranje i izrada pločice je bila izrazito zahtjevna jer je bilo potrebno izvesti mehaničko električni prihvat i montažu tako da se dodatno ne povećavaju gabariti baterije.



Slika 4.4. Ćelijski modul nakon montaže

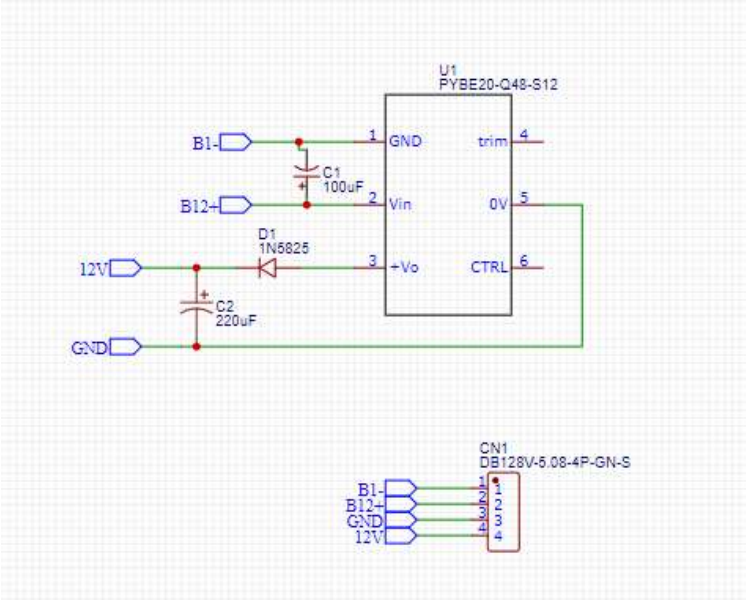


Slika 4.5. Shema druge pločice ćelijskog modula

Matematičke funkcije koje povezuju vrijednosti fizikalnih veličina i odgovarajućih pretvorbi pri digitalizaciji su izvedene kroz programski kod u idućem poglavlju. Varijable (njihova imena) u funkcijama su prilagođene vrijednostima komponenti.

4.2. Napajanje

Napajanje cijelog sustava je izvedeno pomoću dva paralelno spojena izolirana DC-DC pretvarača prema shemi na slici 4.6. Pretvarači su spojeni paralelno na izlazu, a ulazi su spojeni tako da Vin ulaz prvog predstavlja GND drugog što predstavlja pozitivni kraj 12. ćelije iz serije. Na taj način su sve ćelije jednako opterećene, a izbjegnuta je problematika velike razlike ulaznog i izlaznog napona. Napajanje tako isporučuje maksimalno 3334 mA struje na 12 VDC što je dovoljno za 24 ćelijska modula prosječne ukupne potrošnje oko 1 A i BBB čija je maksimalna potrošnja oko 2 A (prosječna je red veličine manja).

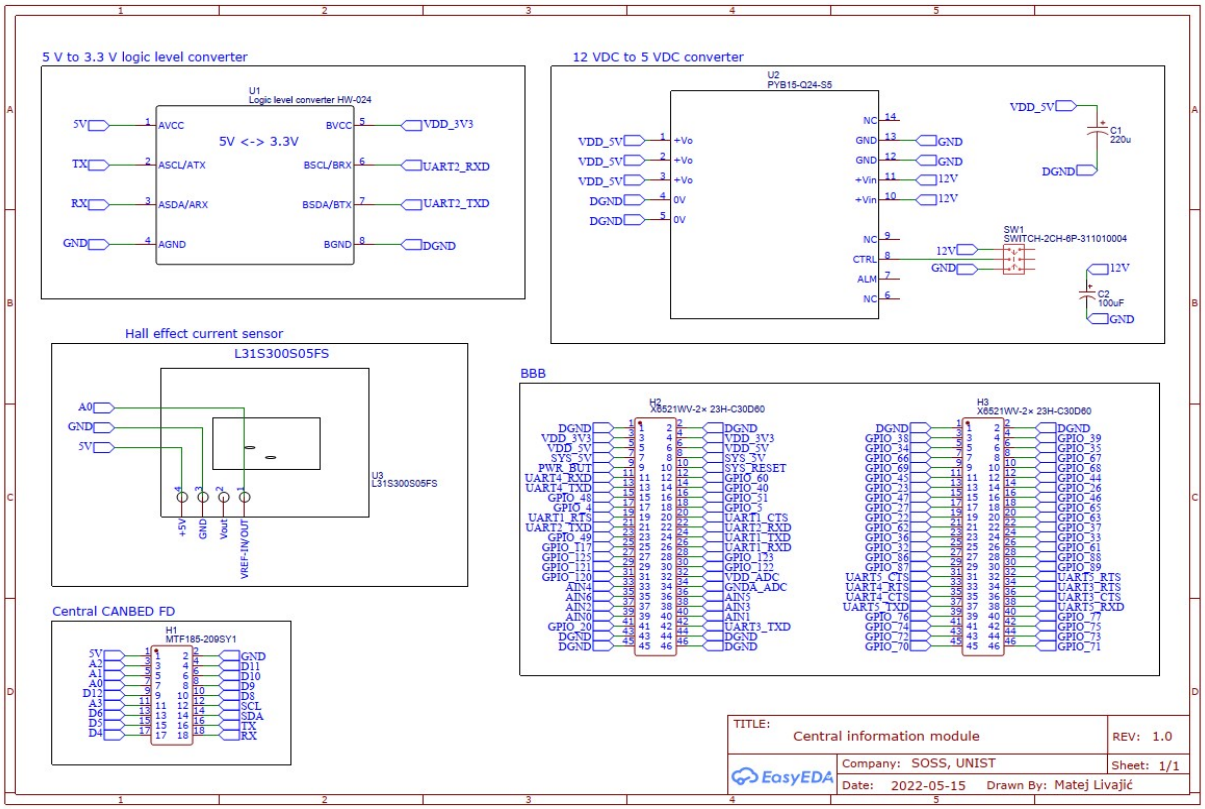


Slika 4.6. Shema prvog DC-DC pretvarača iz paralele

Slika 4.7. Pločice napajanja nakon izrade

4.3. Centralni informacijski modul

Slika 4.8. prikazuje shemu centralnog informacijskog modula kojeg čine jedan BBB, CANBED FD, pripadni pretvarač sa 12 VDC na 5 VDC, pretvarač razine logičkih signala i senzor struje.



Slika 4.8. Shema centralnog informacijskog modula

Centralni informacijski sustav je jasniji kada se pogledaju programski kodovi i rezultati koji su objašnjeni u idućem poglavlju.

5. SOFTVERSKA IMPLEMENTACIJA

Softverski zahtjevi su programiranje svakog ADC-a – *slave-a* (CANBED FD) i centralnog *master* CANBED FD-a i s njime povezanog BeagleBone Black, malog računala. Neki dijelovi sustava su u početku testirani pomoću Raspberry Pi-a , ali zbog robusnosti je zamijenjen sa BeagleBone Black Wireless-om. Radi se o mini računalu prikazanom na slici 5.1. s nekim od karakteristika:

- Processor Octavo Systems OSD3358 1GHz ARM® Cortex-A8
- 512MB DDR3 RAM
- 4GB 8-bit eMMC *on-board flash* memorija
- 3D *graphics accelerator*
- HDMI izlaz
- Bežična WIFI i Bluetooth povezivost
- Mogućnost pokretanja Linux, Android i ostalih operacijskih sustava

Slika 5.1. BeagleBone Black Wireless [14]

Za programiranje CANBED FD-a korišteno je Arduino razvojno okruženje, a za BBB skripte su pisane u Python programskom jeziku i automatski se pokreću sa operacijskim sustavom (pri *boot*-u). Svi programski kodovi su komentirani na engleskom jeziku što je česta praksa u programiranju.

5.1. Ćelijski moduli

U nastavku je programski kod ćelijskih modula.

```
/*
 * Cell module slave code for BMS
 * Final thesis at SOSS, UNIST
 * Created by Matej Livajić
 *
 * Code is not memory optimized
 * It's written for better understanding of working principle
 */



    /*
      MCP2517/8 send a CAN frame

      CAN Bus baudrare for CAN2.0 as below,

      CAN20_5KBPS
      CAN20_10KBPS
      CAN20_20KBPS
      CAN20_25KBPS
      CAN20_31K25BPS
      CAN20_33KBPS
      CAN20_40KBPS
      CAN20_50KBPS
      CAN20_80KBPS
      CAN20_83K3BPS
      CAN20_95KBPS
      CAN20_100KBPS
      CAN20_125KBPS
      CAN20_200KBPS
      CAN20_250KBPS
      CAN20_500KBPS
      CAN20_666KBPS
      CAN20_800KBPS
      CAN20_1000KBPS

      CAN FD Shield - https://www.longan-labs.cc/1030012.html
      CANBed FD - https://www.longan-labs.cc/1030009.html
```

```
39.      */
40.
41.      #include <SPI.h>
42.      #include "mcp2518fd_can.h"
43.
44.
45.      // Variables for communication
46.      const int SPI_CS_PIN = 17;            // CS pin for
   SPI
47.      const int CAN_INT_PIN = 7;            // Interrupt
   pin for CAN
48.      const unsigned int myId = 3;          // Cell module
   ID (address). From 1 to 24
49.      char cellVoltage [4];                 // Voltage
   value string
50.      char cellTemperature[4];              // Temperature
   value string
51.      unsigned char dataFrame[8];           // Dataframe to
   send via CAN
52.      unsigned char dataBuf [2];            // Buffer for
   CAN
53.      unsigned char len = 0;                // Pointer for
   CAN
54.      unsigned int flagRecv = 0;            // Interrupt
   flag
55.      mcp2518fd CAN(SPI_CS_PIN);            // attach CS
   pin
56.
57.      // Variables for sensor readings:
58.
59.      //        Voltage
60.      const int vOutPlusPin = A0;       // First contact
   pin of optoisolated amplifier output
61.      const int vOutMinusPin = A1;      // Second contact
   pin of optoisolated amplifier output
62.      const int R3 = 1000;              // R3 resistor
   value in ohms
63.      const int R2 = 47;                // R2 resistor
   value in ohms
64.      int vOutPlus;                     // Analog read
   value of first cell contact
65.      int vOutMinus;                    // Analog read
   value of second cell contact
66.      float vOutPlusVal;                // Read value of
   first contact in Volts
```

```
67.     float vOutMinusVal;                 // Read value of
   second contact in Volts
68.     float vCellPlus;                    // Voltage value of
   first contact converted considering schematics and
   components datasheet
69.     float vCellMinus;                   // Voltage value of
   second contact converted considering schematics and
   components datasheet
70.     float vCellAvg;                     // Average cell
   value
71.
72.     //        Temperature
73.     const int temperaturePin = A2;      // Voltage output
   pin from PTC resistor
74.     const int Rbias = 10;               // Bias resistor in
   kohms
75.     int vTemp;                          // Analog read
   value
76.     float vTempVal;                     // Analog read
   value in volts
77.     float Rtemp;                        // PTC resistance
   value
78.     float tempVal;                      // Temperature in
   Celsius
79.
80.
81.     void CAN_ISR();                     // CAN interrupt
   service routine (interrupt handler)
82.     void readFromMaster();              // Read from master
   which slave is called
83.     void sendToMaster();                // Send data to
   master
84.     void voltageRead();                 // Read cell
   voltage
85.     void temperatureRead();             // Read cell
   temperature
86.     void adjustFormat();                // Adjust format
   and value for CAN (converts float to string)
87.
88.     void setup()
89.     {
90.         Serial.begin(115200);
   // Serial for debug
91.         while (CAN_OK != CAN.begin(CAN20_10KBPS))
```

```
92.         {
   // Init can bus : baudrate = 10 kbps
93.           Serial.println("CAN init fail, retry...");
94.           delay(100);
95.         }
96.
   attachInterrupt(digitalPinToInterrupt(CAN_INT_PIN),
   CAN_ISR, FALLING); // Start interrupt
97.         Serial.println("CAN init ok!");
98.     }
99.
100.    void loop ()
101.    {
102.      if (flagRecv)        // Check for interrupt
103.      {
104.        flagRecv = 0;      // Put down interrupt flag
105.        readFromMaster(); // Read from master which slave
   is called
106.      }
107.    }
108.
109.    void CAN_ISR()
110.    {
111.      flagRecv = 1;        // interrupt flag (raise) to 1
112.    }
113.
114.    void readFromMaster()
115.    {
116.      CAN.readMsgBuf(&len, dataBuf);          // Read CAN
   data
117.      unsigned long id = CAN.getCanId();     // ID
   (address) of called slave
118.      if (id == myId)                         // Check if
   is this slave called
119.        sendToMaster();                       // Send data
   to master
120.      else
121.      {
122.        Serial.println("\nI am not called. ");
123.        Serial.print("Called is: ");
124.        Serial.println(id);
125.        voltageRead();
126.        temperatureRead();
127.        adjustFormat();
128.      }
```

```
129.     }
130.
131.
132.     void sendToMaster ()
133.     {
134.       CAN.sendMsgBuf(myId, 1, 8, dataFrame);
135.       }
136.
137.     void voltageRead()
138.     {
139.       // uncomment for debug and coeff. adjust
140.       //Serial.println("___________");
141.       vOutPlus = analogRead(vOutPlusPin);
142.       //Serial.print("vOutPlus: ");
143.       //Serial.println(vOutPlus);
144.       vOutMinus = analogRead(vOutMinusPin);
145.       //Serial.print("vOutMinus: ");
146.       //Serial.println(vOutMinus);
147.       vOutPlusVal = vOutPlus / 1023.0 * 5;
148.       //Serial.print("vOutPlusVal: ");
149.       //Serial.println(vOutPlusVal - 1.23, 3);
150.       vOutMinusVal = vOutMinus / 1023.0 * 5;
151.       //Serial.print("vOutMinusVal: ");
152.       //Serial.println(vOutMinusVal - 1.23, 3);
153.       vCellPlus = (vOutPlusVal - 1.23) / (1.23 * R2) *
   0.3 * (R2 + R3);
154.       //Serial.print("vCellPlus: ");
155.       //Serial.println(vCellPlus, 3);
156.       vCellMinus = (vOutMinusVal - 1.23) / (1.23 * R2) *
   (-0.3) * (R2 + R3);
157.       //Serial.print("vCellMinus: ");
158.       //Serial.println(vCellMinus, 3);
159.       vCellAvg = abs((vCellPlus + vCellMinus) / 2);
160.       Serial.print("Cell voltage: ");
161.       Serial.print(vCellAvg, 3);
162.       Serial.println(" V");
163.       }
164.
165.     void temperatureRead()
166.     {
167.       // functions from TMP6131QLPGMQ1 datasheet
168.       vTemp = analogRead(temperaturePin);
169.       vTempVal = vTemp * 5.0 / 1023.0;
170.       Rtemp = Rbias * vTempVal / (5.0 - vTempVal);
171.       tempVal = 14.1 * Rtemp - 115;
```

```
172.       Serial.print("Cell temperature: ");
173.       Serial.print(tempVal, 3);
174.       Serial.println(" °C");
175.       }
176.
177.
178.     void adjustFormat()
179.     {
180.       // shrinking data to standard CAN data frame - 8
   bytes
181.       dtostrf(vCellAvg, 4, 2, cellVoltage);
182.       dtostrf(tempVal, 4, 1, cellTemperature);
183.       for (int i = 0; i < 4; i++)
184.       {
185.         dataFrame[i] = (unsigned char) cellVoltage[i];
186.         dataFrame[i+4] = (unsigned char)
   cellTemperature[i];
187.         }
188.
189.       }
190.
```

5.2. Centralni master modul

U nastavku je programski kod centralnog master modula.

```
1. /*
2.  * Central information module master code for BMS
3.  * Final thesis at SOSS, UNIST
4.  * Created by Matej Livajić
5.  *
6.  * Code is not memory optimized
7.  * It's written for better understanding of working
   principle
8.  */
9.
10.     /*
11.       MCP2517/8 send a CAN frame
```

```
12.
13.       CAN Bus baudrare for CAN2.0 as below,
14.
15.       CAN20_5KBPS
16.       CAN20_10KBPS
17.       CAN20_20KBPS
18.       CAN20_25KBPS
19.       CAN20_31K25BPS
20.       CAN20_33KBPS
21.       CAN20_40KBPS
22.       CAN20_50KBPS
23.       CAN20_80KBPS
24.       CAN20_83K3BPS
25.       CAN20_95KBPS
26.       CAN20_100KBPS
27.       CAN20_125KBPS
28.       CAN20_200KBPS
29.       CAN20_250KBPS
30.       CAN20_500KBPS
31.       CAN20_666KBPS
32.       CAN20_800KBPS
33.       CAN20_1000KBPS
34.
35.       CAN FD Shield - https://www.longan-
   labs.cc/1030012.html
36.       CANBed FD - https://www.longan-labs.cc/1030009.html
37.     */
38.
39.     #include <SPI.h>
40.     #include "mcp2518fd_can.h"
41.
42.     // Variables for communication
43.
44.     const int SPI_CS_PIN = 17;                  // CS pin
   for SPI
45.     const int CAN_INT_PIN = 7;                  //
   Interrupt pin for CAN
46.     mcp2518fd CAN(SPI_CS_PIN);                  // attach
   CS pin
47.     const unsigned int myId = 0;                // Master
   module ID (adress). "Master" is zero.
48.     unsigned char cellVoltage[4][24] = {};      // Two-
   dimensional array for cell voltage storage
49.     unsigned char cellTemperature[4][24] = {}; // Two-
   dimensional array for cell voltage storage
```

```
50.     unsigned int currentSlave = 23;            // Starting
   slave adress
51.     unsigned int lastSlave = 22;               // Starting
   last slave adress
52.     unsigned int flagRecv = 0;                 //
   Interrupt flag
53.
54.
55.     // Variables for analog read
56.
57.     const int currentPin = A0;                 // Analog
   read of current sensor pin
58.     const int currOffsetPin = A1;              // Analog
   read of current sensor offset pin
59.     int vCurr;                                 // Analog
   read of current sensor voltage
60.     float vCurrVal;                            // Analog
   read of current sensor voltage in volts
61.     int vOffset;                               // Analog
   read of current sensor offset voltage
62.     float vOffsetVal;                          // Analog
   read of current sensor offset voltage in volts
63.     float currVal;                             // Analog
   read of current sensor converted to current
64.     float currValBuf [10];                     // Buffer
   for current reading filtering
65.     float avgcurr = 0;                         // Filtered
   current value
66.
67.
68.     void CAN_ISR();                 // CAN interrupt
   service routine (interrupt handler)
69.     void readFromSlave();           // Read data from
   slave
70.     void callNextSlave ();          // Calling next slave
71.     void printValues();             // Printing values
   (debug)
72.     void currentRead();             // Read current from
   sensor
73.     void currentReadFiltered();     // Filter current
   read values
74.     void printValues();             // Send values to BBB
75.
76.
77.     void setup() {
```

```
78.         Serial.begin(9600);                    // Serial
   for debug
79.         Serial1.begin(115200);                 // Serial
   for communication with BBB
80.         while (CAN_OK != CAN.begin(CAN20_10KBPS))
81.         {                                      // init can
   bus : baudrate = 10 kbps
82.             Serial.println("CAN init fail, retry...");
83.             delay(10);
84.         }
85.
   attachInterrupt(digitalPinToInterrupt(CAN_INT_PIN),
   CAN_ISR, FALLING); // Start interrupt
86.         Serial.println("CAN init ok!");
87.     }
88.
89.     void loop ()
90.     {
91.       delay(20);          // Adjusting speed, increase
   delay if slaves do not follow
92.       if (flagRecv)       // Check for interrupt
93.       {
94.         flagRecv = 0;     // Put interrupt flag down
95.         readFromSlave();  // Read value from slave
96.       }
97.       callNextSlave ();   // Calling next slave
98.       delay(5);
99.       currentReadFiltered();    // Read filtered current
   sensor value
100.
101.      }
102.
103.    void CAN_ISR()
104.    {
105.        flagRecv = 1;                    // Raise
   interrupt flag
106.    }
107.
108.    void readFromSlave()
109.    {
110.      unsigned long id = 0;              // Assign ID
   (address) 0 to master
111.      unsigned char len = 0;             // Pointer for
   CAN
```

```
112.      unsigned char databuf [8];          // CAN incoming
   data buffer
113.      CAN.readMsgBuf(&len, databuf);      // Read CAN data
114.      id = CAN.getCanId();                // Get address of
   response slave
115.      // check if slaves are following:
116.      Serial.println("Current slave is " +
   String(currentSlave) +" and response is from " +
   String(id));
117.      for (int i = 0; i < 4; i++)
118.      {
119.        cellVoltage[i][(int)id-1] = databuf[i];
   // store voltage value
120.        cellTemperature[i][(int)id-1] = databuf [i+4];
   // store temperature value
121.        }
122.        lastSlave = currentSlave;
   // reading finished, current slave becomes last
123.    }
124.
125.    void callNextSlave ()
126.    {
127.      if (currentSlave < 24)
128.      currentSlave++;
   // next slave
129.      else
130.      {
131.      currentSlave = 1;
   // go to first slave
132.      printValues();
   // comment if not debug
133.      }
134.      CAN.sendMsgBuf(currentSlave, 0, 1, 0);
   // call current slave with 0 data
135.      Serial.println("Called slave: ");
   // debug
136.      Serial.println (currentSlave);
137.      }
138.
139.
140.    void currentRead()
141.    {
142.      // from L31S300S05FS datasheet
143.      vCurr = analogRead(currentPin);
144.      vCurrVal = vCurr * 5.0 / 1023.0;
```

```
145.      vOffset = analogRead(currOffsetPin);
146.      vOffsetVal = vOffset * 5.0 / 1023.0;
147.      currVal = (vCurrVal - vOffsetVal) / 0.625 * 300 +
   5;
148.
149.      }
150.
151.    void currentReadFiltered()
152.    {
153.      currentRead();
154.      for (int i = 9; i > 0 ; i--)
155.      {
156.      currValBuf[i] = currValBuf[i-1];
157.      avgcurr += currValBuf [i];
158.      }
159.      currValBuf[0] = currVal;
160.      avgcurr += currValBuf[0];
161.      avgcurr /= 10;
162.      if(isnan(avgcurr))
163.      avgcurr = 0;
164.      }
165.
166.    void printValues()
167.    {
168.      Serial1.println("Dataframe");
169.      for (int j = 0; j < 24; j++)
170.      {
171.        //Serial1.println("V" + String(j+1));
172.        for (int i = 0; i < 4; i++)
173.        {
174.        Serial1.print((char)cellVoltage[i][j]);
175.        cellVoltage[i][j] = 0;
176.        }
177.        Serial1.println("");
178.      }
179.      for (int j = 0; j < 24; j++)
180.      {
181.        //Serial1.println("T" + String(j+1));
182.        for (int i = 0; i < 4; i++)
183.        {
184.        Serial1.print((char)cellTemperature[i][j]);
185.        cellTemperature[i][j] = 0;
186.        }
187.        Serial1.println("");
188.      }
```

```
189.        //Serial1.println("I");
190.        Serial1.println(avgcurr);
191.        avgcurr = 'x';
192.        Serial1.println("");
193.        }
194.
```

5.3. Centralno BBB računalo

U nastavku je programski kod centralnog BBB računala.

```
1. # Central information module master code for BMS (BBB)
2.  # Final thesis at SOSS, UNIST
3.  # Created by Matej Livajic
4.
5.  # Code is not memory optimized
6.  # It's written for better understanding of working
   principle
7.
8. # Necessary libary
9.
10.      from PyQt5.QtGui import *
11.      from PyQt5.QtWidgets import *
12.      from PyQt5.QtCore import *
13.      from PyQt5 import QtCore, QtGui, QtWidgets
14.      from pyqtgraph import PlotWidget
15.      import random
16.      import numpy as np
17.      import math
18.      import Adafruit_BBIO.UART as UART
19.      import serial
20.      import time
21.      import traceback, sys
22.
23.      # Configure UART with CANBED FD (Master)
24.      UART.setup("UART2")
25.      ser = serial.Serial(port = "/dev/ttyO2", baudrate =
   115200)
26.
27.      # Styles for progress bars
```

```
28.
29.     AMPERAGE_DISCHARGE_STYLE = """
30.     QProgressBar{
31.         border: 2px solid grey;
32.         border-radius: 5px;
33.         text-align: center
34.     }
35.
36.     QProgressBar::chunk {
37.         background-color: red;
38.         width: 10px;
39.         margin: 1px;
40.     }
41.     """
42.
43.     AMPERAGE_CHARGE_STYLE = """
44.     QProgressBar{
45.         border: 2px solid grey;
46.         border-radius: 5px;
47.         text-align: center
48.     }
49.
50.     QProgressBar::chunk {
51.         background-color: green;
52.         width: 10px;
53.         margin: 1px;
54.     }
55.     """
56.
57.     VOLTAGE_OK_STYLE = """
58.     QProgressBar{
59.         border: 2px solid grey;
60.         border-radius: 3px;
61.         text-align: center
62.     }
63.
64.     QProgressBar::chunk {
65.         background-color: green;
66.         width: 5px;
67.         margin: 1px;
68.     }
69.     """
70.
71.     VOLTAGE_NOTOK_STYLE = """
72.     QProgressBar{
```

```
73.             border: 2px solid grey;
74.             border-radius: 3px;
75.             text-align: center
76.         }
77.
78.         QProgressBar::chunk {
79.             background-color: red;
80.             width: 5px;
81.             margin: 1px;
82.         }
83.         """
84.
85.         VOLTAGE_ERR_STYLE = """
86.         QProgressBar{
87.             border: 2px solid red;
88.             border-radius: 1px;
89.             text-align: center
90.         }
91.         """
92.
93.         backgroundColor = "background-color: rgb(229, 229,
    229);"
94.
95.         # Classes for multithreading GUI and data
96.
97.         class WorkerSignals(QObject):
98.
99.             finished = pyqtSignal()
100.            error = pyqtSignal(tuple)
101.            result = pyqtSignal(object)
102.            progress = pyqtSignal(int)
103.
104.
105.        class Worker(QRunnable):
106.
107.            def __init__(self, fn, *args, **kwargs):
108.                super(Worker, self).__init__()
109.
110.                # Store constructor arguments (re-used for
    processing)
111.                self.fn = fn
112.                self.args = args
113.                self.kwargs = kwargs
114.                self.signals = WorkerSignals()
115.
```

```
116.             # Add the callback to our kwargs
117.             self.kwargs['progress_callback'] =
   self.signals.progress
118.
119.         @pyqtSlot()
120.         def run(self):
121.
122.             try:
123.                 result = self.fn(*self.args,
   **self.kwargs)
124.             except:
125.                 traceback.print_exc()
126.                 exctype, value = sys.exc_info()[:2]
127.                 self.signals.error.emit((exctype, value,
   traceback.format_exc()))
128.             else:
129.                 self.signals.result.emit(result)  #
   Return the result of the processing
130.             finally:
131.                 self.signals.finished.emit()  # Done
132.
133.     # GUI class
134.
135.     class MainWindow(QMainWindow):
136.
137.         def __init__(self, *args, **kwargs):
138.             super(MainWindow, self).__init__(*args,
   **kwargs)
139.
140.             _translate =
   QtCore.QCoreApplication.translate
141.             self.setWindowTitle(_translate("eBuggy",
   "eBuggy"))
142.             self.resize(1280, 720)
        # Window size
143.             self.voltageVals = []                    #
   Array for voltage values
144.             self.temperatureVals = []                #
   Array for temperature values
145.             for i in range (24):
        # Numpy create 24 elements
146.                 self.voltageVals.append("")
147.                 self.temperatureVals.append("")
148.             self.currentVal = ""
149.             self.sumVolt = 0.0
```

```
150.            self.avgTemp = 0.0
151.            self.corrRead = 0
152.
153.            #GUI elements variables
154.
155.            self.battery_label = []
156.               self.voltage_progress_bar = []
157.               self.voltage_label = []
158.               self.temperature_progress_bar = []
159.               self.temperature_label = []
160.               batt_font = QtGui.QFont()                    #
   Font for cells
161.               batt_font.setBold(True)
162.               batt_font.setWeight(75)
163.               self.main_toolbar_pushbutton = []
164.               self.main_toolbar_icon = []
165.               self.main_toolbar_icon_name = ["buggy.png",
   "navigation.png", "media.png", "internet_browser.png",
   "phone.png", "settings.png"]
166.               self.main_toolbar_label = []
167.               self.main_toolbar_text = ["Vehicle", "Navi",
   "Media", "Browser", "Phone", "Settings"]
168.               toolbar_font = QtGui.QFont()                 #
   Font for main toolbar
169.               toolbar_font.setPointSize(8)
170.               toolbar_font.setBold(True)
171.               toolbar_font.setWeight(75)
172.               mini_tittle_font = QtGui.QFont()             #
   Font for secondary toolbar
173.               mini_tittle_font.setBold(True)
174.               mini_tittle_font.setWeight(75)
175.               self.second_toolbar_pushbutton = []
176.               self.second_toolbar_icon = []
177.               self.second_toolbar_icon_name =
   ["performance.png", "bms.png", "diagnostics.png",
   "dynamics.png"]
178.               self.second_toolbar_label = []
179.               self.second_toolbar_text = ["Performance",
   "BMS", "Diagnostics", "Dynamics"]
180.
181.               _translate =
   QtCore.QCoreApplication.translate  # translate tool
182.
183.               for i in range(24):      # creating GUI
   elements for 24 cells
```

```
184.
185.
    self.battery_label.append(QtWidgets.QLabel(self))
186.
    self.battery_label[i].setGeometry(QtCore.QRect(240, 60 +
    i*20, 30, 15))
187.
    self.battery_label[i].setText(_translate("MainWindow",str(i
    +1)))
188.                 self.battery_label[i].setFont(batt_font)
189.
    self.battery_label[i].setAlignment(QtCore.Qt.AlignRight|QtC
    ore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
190.
191.
    self.voltage_progress_bar.append(QtWidgets.QProgressBar(sel
    f))
192.
    self.voltage_progress_bar[i].setGeometry(QtCore.QRect(310,
    60 + i*20, 100, 15))
193.
    self.voltage_progress_bar[i].setProperty("value", 58)
194.
    self.voltage_progress_bar[i].setTextVisible(False)
195.
196.
    self.voltage_label.append(QtWidgets.QLabel(self))
197.
    self.voltage_label[i].setGeometry(QtCore.QRect(420, 60 +
    i*20, 61, 15))
198.
    self.voltage_label[i].setText(_translate("MainWindow",
    "22.222"))
199.
200.
    self.temperature_progress_bar.append(QtWidgets.QProgressBar
    (self))
201.
    self.temperature_progress_bar[i].setGeometry(QtCore.QRect(5
    10, 60 + i*20, 100, 15))
202.
    self.temperature_progress_bar[i].setProperty("value", 58)
203.
    self.temperature_progress_bar[i].setTextVisible(False)
204.
```

```
205.
    self.temperature_label.append(QtWidgets.QLabel(self))
206.
    self.temperature_label[i].setGeometry(QtCore.QRect(630, 60
    + i*20, 61, 15))
207.
    self.temperature_label[i].setText(_translate("MainWindow",
    "35,335"))
208.
209.
210.             for i in range(6):              # Creating main
    toolbar GUI elements
211.
    self.main_toolbar_pushbutton.append(QtWidgets.QPushButton(s
    elf))
212.
    self.main_toolbar_pushbutton[i].setGeometry(QtCore.QRect(0,
    10 + i*110, 100, 110))
213.
    self.main_toolbar_pushbutton[i].setLayoutDirection(QtCore.Q
    t.LeftToRight)
214.                 self.main_toolbar_pushbutton[i].raise_()
215.
216.
    self.main_toolbar_icon.append(QtWidgets.QLabel(self))
217.
    self.main_toolbar_icon[i].setGeometry(QtCore.QRect(15, 40 +
    i*110, 65, 65))
218.
    self.main_toolbar_icon[i].setPixmap(QtGui.QPixmap("/home/de
    bian/Desktop/startUpApp/res/img/icons/main_toolbar/"+
    str(self.main_toolbar_icon_name[i])))
219.
    self.main_toolbar_icon[i].setScaledContents(True)
220.                 self.main_toolbar_icon[i].raise_()
221.
222.
    self.main_toolbar_label.append(QtWidgets.QLabel(self))
223.
    self.main_toolbar_label[i].setGeometry(QtCore.QRect(10, 20
    + i*110, 80, 20))
224.
    self.main_toolbar_label[i].setFont(toolbar_font)
```

```
225.
   self.main_toolbar_label[i].setAlignment(QtCore.Qt.AlignCent
   er)
226.
   self.main_toolbar_label[i].setText(_translate("MainWindow",
   str(self.main_toolbar_text[i])))
227.                 self.main_toolbar_label[i].raise_()
228.
229.
230.             for i in range(4):              # Creating
   second toolbar GUI elements
231.
   self.second_toolbar_pushbutton.append(QtWidgets.QPushButton
   (self))
232.
   self.second_toolbar_pushbutton[i].setGeometry(QtCore.QRect(
   110, 10 + i*110, 100, 110))
233.
   self.second_toolbar_pushbutton[i].setLayoutDirection(QtCore
   .Qt.LeftToRight)
234.
   self.second_toolbar_pushbutton[i].raise_()
235.
236.
   self.second_toolbar_icon.append(QtWidgets.QLabel(self))
237.
   self.second_toolbar_icon[i].setGeometry(QtCore.QRect(125,
   40 + i*110, 65, 65))
238.
   self.second_toolbar_icon[i].setPixmap(QtGui.QPixmap("/home/
   debian/Desktop/startUpApp/res/img/icons/main_toolbar/"+
   str(self.second_toolbar_icon_name[i])))
239.
   self.second_toolbar_icon[i].setScaledContents(True)
240.                 self.second_toolbar_icon[i].raise_()
241.
242.
   self.second_toolbar_label.append(QtWidgets.QLabel(self))
243.
   self.second_toolbar_label[i].setGeometry(QtCore.QRect(115,
   20 + i*110, 90, 20))
244.
   self.second_toolbar_label[i].setFont(toolbar_font)
```

```
245.
   self.second_toolbar_label[i].setAlignment(QtCore.Qt.AlignCe
   nter)
246.
   self.second_toolbar_label[i].setText(_translate("MainWindow
   ", str(self.second_toolbar_text[i])))
247.                 self.second_toolbar_label[i].raise_()
248.
249.
250.          # Creation of lines for borders between toolbar
   elements
251.             self.line_definition = QtGui.QFont()
252.             self.line_definition.setPointSize(9)
253.             self.line_definition.setWeight(50)
254.
255.             self.V_line_1 = QtWidgets.QFrame(self)
256.             self.V_line_1.setGeometry(QtCore.QRect(100,
   5, 10, 690))
257.             self.V_line_1.setFont(self.line_definition)
258.
   self.V_line_1.setFrameShape(QtWidgets.QFrame.VLine)
259.
   self.V_line_1.setFrameShadow(QtWidgets.QFrame.Sunken)
260.             self.V_line_2 = QtWidgets.QFrame(self)
261.             self.V_line_2.setGeometry(QtCore.QRect(210,
   5, 10, 690))
262.             self.V_line_2.setFont(self.line_definition)
263.
   self.V_line_2.setFrameShape(QtWidgets.QFrame.VLine)
264.
   self.V_line_2.setFrameShadow(QtWidgets.QFrame.Sunken)
265.
266.          # Creation of first graph
267.             self.graphWidget = PlotWidget(self)
268.
   self.graphWidget.setGeometry(QtCore.QRect(750, 70, 450,
   150))
269.             sizePolicy =
   QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
   QtWidgets.QSizePolicy.Preferred)
270.             sizePolicy.setHorizontalStretch(0)
271.             sizePolicy.setVerticalStretch(0)
272.
   sizePolicy.setHeightForWidth(self.graphWidget.sizePolicy().
   hasHeightForWidth())
```

```
273.            self.graphWidget.setSizePolicy(sizePolicy)
274.            self.graphWidget.setAutoFillBackground(False)
275.            self.graphWidget.setStyleSheet("background-
   color: rgb(255, 255, 255);")
276.            self.graphWidget.setObjectName("graphWidget")
277.          # Creation of second graph
278.            self.graphWidget_2 = PlotWidget(self)
279.
   self.graphWidget_2.setGeometry(QtCore.QRect(750, 490, 450,
   150))
280.            sizePolicy =
   QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
   QtWidgets.QSizePolicy.Preferred)
281.            sizePolicy.setHorizontalStretch(0)
282.            sizePolicy.setVerticalStretch(0)
283.
   sizePolicy.setHeightForWidth(self.graphWidget_2.sizePolicy(
   ).hasHeightForWidth())
284.            self.graphWidget_2.setSizePolicy(sizePolicy)
285.
   self.graphWidget_2.setAutoFillBackground(False)
286.            self.graphWidget_2.setStyleSheet("background-
   color: rgb(255, 255, 255);")
287.
   self.graphWidget_2.setObjectName("graphWidget_2")
288.          # Creation of third graph
289.            self.graphWidget_3 = PlotWidget(self)
290.
   self.graphWidget_3.setGeometry(QtCore.QRect(750, 280, 450,
   150))
291.            sizePolicy =
   QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
   QtWidgets.QSizePolicy.Preferred)
292.            sizePolicy.setHorizontalStretch(0)
293.            sizePolicy.setVerticalStretch(0)
294.
   sizePolicy.setHeightForWidth(self.graphWidget_3.sizePolicy(
   ).hasHeightForWidth())
295.            self.graphWidget_3.setSizePolicy(sizePolicy)
296.
   self.graphWidget_3.setAutoFillBackground(False)
297.            self.graphWidget_3.setStyleSheet("background-
   color: rgb(255, 255, 255);")
298.
   self.graphWidget_3.setObjectName("graphWidget_3")
```

```
299.
300.          # Creation of other GUI elements
301.             self.label_1 = QtWidgets.QLabel(self)
302.             self.label_1.setGeometry(QtCore.QRect(230,
   20, 61, 21))
303.             self.label_1.setFont(mini_tittle_font)
304.
   self.label_1.setAlignment(QtCore.Qt.AlignCenter)
305.             self.label_1.setObjectName("label_1")
306.
307.             self.label_2 = QtWidgets.QLabel(self)
308.             self.label_2.setGeometry(QtCore.QRect(290,
   20, 171, 21))
309.             self.label_2.setFont(mini_tittle_font)
310.
   self.label_2.setAlignment(QtCore.Qt.AlignCenter)
311.             self.label_2.setObjectName("label_2")
312.
313.             self.label_3 = QtWidgets.QLabel(self)
314.             self.label_3.setGeometry(QtCore.QRect(500,
   20, 171, 21))
315.             self.label_3.setFont(mini_tittle_font)
316.
   self.label_3.setAlignment(QtCore.Qt.AlignCenter)
317.             self.label_3.setObjectName("label_3")
318.
319.             self.label_4 = QtWidgets.QLabel(self)
320.             self.label_4.setGeometry(QtCore.QRect(370,
   560, 171, 21))
321.             self.label_4.setFont(mini_tittle_font)
322.
   self.label_4.setAlignment(QtCore.Qt.AlignCenter)
323.             self.label_4.setObjectName("label_4")
324.
325.             self.label_5 = QtWidgets.QLabel(self)
326.             self.label_5.setGeometry(QtCore.QRect(570,
   590, 61, 31))
327.             self.label_5.setFont(mini_tittle_font)
328.             self.label_5.setObjectName("label_5")
329.
330.             self.label_6 = QtWidgets.QLabel(self)
331.             self.label_6.setGeometry(QtCore.QRect(870,
   40, 171, 21))
332.             self.label_6.setFont(mini_tittle_font)
```

```
333.
   self.label_6.setAlignment(QtCore.Qt.AlignCenter)
334.             self.label_6.setObjectName("label_6")
335.
336.             self.label_7 = QtWidgets.QLabel(self)
337.             self.label_7.setGeometry(QtCore.QRect(830,
   250, 261, 21))
338.             self.label_7.setFont(mini_tittle_font)
339.
   self.label_7.setAlignment(QtCore.Qt.AlignCenter)
340.             self.label_7.setObjectName("label_7")
341.
342.             self.label_8 = QtWidgets.QLabel(self)
343.             self.label_8.setGeometry(QtCore.QRect(830,
   460, 261, 21))
344.             self.label_8.setFont(mini_tittle_font)
345.
   self.label_8.setAlignment(QtCore.Qt.AlignCenter)
346.             self.label_8.setObjectName("label_8")
347.
348.             self.amperage_progress_bar =
   QtWidgets.QProgressBar(self)
349.
   self.amperage_progress_bar.setGeometry(QtCore.QRect(250,
   590, 301, 31))
350.
   self.amperage_progress_bar.setProperty("value", 77)
351.
   self.amperage_progress_bar.setTextVisible(False)
352.
   self.amperage_progress_bar.setObjectName("amperage_progress
   _bar")
353.
354.             self.label_1.setText(_translate("MainWindow",
   "NO Cell"))
355.             self.label_2.setText(_translate("MainWindow",
   "Voltage / V"))
356.             self.label_3.setText(_translate("MainWindow",
   "Temperature / C"))
357.             self.label_4.setText(_translate("MainWindow",
   "Current / A"))
358.
359.             self.label_6.setText(_translate("MainWindow",
   "Voltage / V,   Battery pack"))
```

```
360.             self.label_7.setText(_translate("MainWindow",
   "Current / A ,   Battery pack"))
361.             self.label_8.setText(_translate("MainWindow",
   "Average temperature / C ,   Battery pack"))
362.
363.
364.             self.plotTimeInterval = 10
365.             self.voltagePlot = []
366.             self.temperaturePlot = []
367.             self.amperagePlot = []
368.             self.xPlot = []
369.             self.counter = 0.0
370.
371.             for i in range(self.plotTimeInterval):
372.                 self.xPlot.append(-(self.plotTimeInterval
   - i))
373.                 self.voltagePlot.append(0.0)
374.                 self.temperaturePlot.append(0.0)
375.                 self.amperagePlot.append(0.0)
376.
377.             self.show()    #GUI show
378.
379.             self.threadpool = QThreadPool()
380.             print("Multithreading with maximum %d
   threads" % self.threadpool.maxThreadCount())
381.
382.             self.timer = QTimer()
383.             self.timer.setInterval(1000)    # Interval for
   GUI update
384.             self.timer.timeout.connect(self.dataUpdate)
385.             self.timer.start()
386.
387.         def oh_no(self):
388.             # Pass the function to execute
389.             worker = Worker(self.execute_this_fn) # Any
   other args, kwargs are passed to the run function
390.
   worker.signals.result.connect(self.print_output)
391.
   worker.signals.finished.connect(self.thread_complete)
392.
   worker.signals.progress.connect(self.progress_fn)
393.
394.             # Execute
395.             self.threadpool.start(worker)
```

```
396.
397.
398.        def dataUpdate(self):
399.
400.            while True:
401.                ser.close()        # Clearing of SERAL
    buffer
402.                ser.open()
403.                if ser.readline().strip() == "Dataframe":
404.                      for i in range (24):
405.                            try:
406.                                  self.voltageVals[i] =
    str(float(ser.readline().strip()))
407.                            except ValueError:
408.                                  self.voltageVals[i] =
    "ERR"
409.                      for i in range (24):
410.                            try:
411.                                  self.temperatureVals[i] =
    str(float(ser.readline().strip()))
412.                            except ValueError:
413.                                  self.temperatureVals[i] =
    "ERR"
414.                      try:
415.                            self.currentVal =
    str(float(ser.readline().strip()))
416.                      except ValueError:
417.                            self.currentVal = "ERR"
418.                break
419.
420.          for i in range(24):
421.
        self.voltage_label[i].setText(self.voltageVals[i])
422.
        self.temperature_label[i].setText(self.temperatureVal
    s[i])
423.                try:
424.
    self.voltage_progress_bar[i].setProperty("value",
    float(self.voltageVals[i])/3.75*100)
425.                            if (float(self.voltageVals[i])
    > 2.0) and (float(self.voltageVals[i]) < 3.65):
426.
        self.voltage_progress_bar[i].setStyleSheet(VOLTAGE_OK
    _STYLE)
```

```
427.                         else:
428.
        self.voltage_progress_bar[i].setStyleSheet(VOLTAGE_NO
   TOK_STYLE)
429.                 except ValueError:
430.
        self.voltage_progress_bar[i].setProperty("value",
   float(0))
431.
        self.voltage_progress_bar[i].setStyleSheet(VOLTAGE_ER
   R_STYLE)
432.
433.                         try:
434.
        self.temperature_progress_bar[i].setProperty("value",
   float(self.temperatureVals[i]))
435.                             if
   (float(self.temperatureVals[i]) > 0.0) and
   (float(self.temperatureVals[i]) < 50):
436.
        self.temperature_progress_bar[i].setStyleSheet(VOLTAG
   E_OK_STYLE)
437.                             else:
438.
        self.temperature_progress_bar[i].setStyleSheet(VOLTAG
   E_NOTOK_STYLE)
439.                 except ValueError:
440.
        self.temperature_progress_bar[i].setProperty("value",
   float(0))
441.
        self.temperature_progress_bar[i].setStyleSheet(VOLTAG
   E_ERR_STYLE)
442.             self.label_5.setText(self.currentVal)
443.
   self.amperage_progress_bar.setProperty("value",
   float(self.currentVal))
444.
445.
446.
447.
448.
449.             if(float(self.currentVal) > 0):
```

```
450.
   self.amperage_progress_bar.setStyleSheet(AMPERAGE_DISCHARGE
   _STYLE)
451.             else:
452.
   self.amperage_progress_bar.setStyleSheet(AMPERAGE_CHARGE_ST
   YLE)
453.
454.
455.
456.          # Calculate total voltage, average temperature,
   and current for graph display
457.          self.corrRead = 0
458.          self.sumVolt = 0.0
459.          self.avgTemp = 0.0
460.          for i in range(24):
461.               try:
462.                    self.sumVolt +=
   float(self.voltageVals[i])
463.               except:
464.                    self.sumVolt += 0.0
465.               try:
466.                    self.avgTemp +=
   float(self.temperatureVals[i])
467.                    self.corrRead += 1
468.               except:
469.                    self.avgTemp += 0.0
470.          if self.corrRead!=0:
471.               self.avgTemp = self.avgTemp / self.corrRead
472.
473.            self.counter = self.counter + 0.3
474.            self.voltagePlot = np.roll(self.voltagePlot,
   -1)
475.            self.voltagePlot[self.plotTimeInterval - 1] =
   self.sumVolt
476.            self.graphWidget.clear()
477.            self.graphWidget.plot(self.xPlot,
   self.voltagePlot)
478.
   self.graphWidget.setYRange(0.8*np.mean(self.voltagePlot),1.
   2*np.mean(self.sumVolt) , padding=0)
479.          self.graphWidget.showGrid(x=True, y=True)
480.
481.            self.temperaturePlot
   =np.roll(self.temperaturePlot, -1)
```

```
482.             self.temperaturePlot[self.plotTimeInterval -
   1] = self.avgTemp
483.             self.graphWidget_2.clear()
484.             self.graphWidget_2.plot(self.xPlot,
   self.temperaturePlot)
485.             self.graphWidget_2.setYRange(self.avgTemp -
   10.0, self.avgTemp + 10.0, padding=0)
486.          self.graphWidget_2.showGrid(x=True, y=True)
487.
488.             self.amperagePlot =np.roll(self.amperagePlot,
   -1)
489.             self.amperagePlot[self.plotTimeInterval - 1]
   = self.currentVal
490.             self.graphWidget_3.clear()
491.             self.graphWidget_3.plot(self.xPlot,
   self.amperagePlot)
492.             self.graphWidget_3.setYRange(-100 , 200,
   padding=0)
493.          self.graphWidget_3.showGrid(x=True, y= True)
494.
495.     app = QApplication([])
496.     window = MainWindow()
497.     app.exec_()
498.
```

5.4. Grafičko sučelje

Konačni rezultat je grafičko sučelje s korisnikom koji može biti vozač koji gleda u ekran spojen preko HDMI ili istraživač u laboratoriju ili čak dalje. Slika 5.2. prikazuje primjer sučelja (GUI) koji je izrađen pomoću PyQt5 biblioteke za Python.

Slika 5.2. Grafičko sučelje BMS-a

Slika prikazuje testni, ali reprezentativni primjer rezultata gdje su spojeni moduli adresa 3 i 7. Prikazuju se naponi i temperature svake od ćelija koje su uspješno spojene, kao i ukupna struja. Grafovi (desno) prikazuju promjenu ukupnog napona spojenih baterija, promjenu prosječne temperature i struje kroz vrijeme. Na slici se vidi da je GUI-u pristupljeno s udaljenog računala pomoću VNC preglednika gdje je BBB podijelio svoj zaslon.

6. ZAKLJUČAK

BMS je sustav koji u poduzećima razvijaju timovi desetina inženjera i u velikom je fokusu danas jer je bitno imati pouzdan baterijski sustav. Otegotna okolnost nedostatka čipova je dovela do situacije da se pribjegava neklasičnim metodama dizajniranja i topologijama ovakvih sustava, a prvenstveno se misli na modularno rješenje povezano preko CAN-a. Takvo rješenje je pokazalo neke prednosti kao što je skalabilnost, konfigurabilnost, razumljivost i sl. Neki od nedostataka su količina vremena koje je potrebno uložiti u razvoj, nešto veća potrošnja energije i za serijsku proizvodnju vjerojatno veća cijena proizvodnje naspram integriranog rješenja. Kao nastavak rada nameće se kontroliranje određenih izlaznih veličina baterijskog sustava s obzirom na praćene informacije, kao i finija kalibriranja senzora te razvoj vlastite CANBED FD pločice koja bi smanjila potrošnju energije po modulu dvostruko (za minimalno 20 mA).

LITERATURA

[1] Wikipedia: Buggy (automobile), https://en.wikipedia.org/wiki/Buggy_(automobile), [14.8.2021.]

[2] Sunder battery: LiFePO4 battery 3.2 V 100Ah, https://www.sunderbattery.com/product/lifepo4-battery-3-2v-100ah/, [18.8.2021.]

[3] Golden motor: HPM-10KW - High Power BLDC, https://www.goldenmotor.com/frame-bldcmotor.htm, [19.8.2021.]

[4] Golden motor: VECTOR 500 Series Brushless Motor Controller, https://goldenmotor.bike/product/vector-500-series-72-volt-brushless-motor-controller/, [20.8.2021.]

[5] CircuitDigest: *Multicell voltage monitoring for lithium battery pack in electric vehicles*, https://circuitdigest.com/microcontroller-projects/multi-cell-voltage-monitoring-for-lithium-battery-pack-in-electric-vehicles, [16.8.2021.]

[6] Electronics tutorials: *Temperature sensors*, https://www.electronics-tutorials.ws/io/io_3.html, [16.8.2021.]

[7] Eletimes: *Comparing shunt and hall-based isolated current-sensing solutions in HEV/EV*, http://www.eletimes.com/comparing-shunt-and-hall-based-isolated-current-sensing-solutions-in-hev-ev, [17.8.2021.]

[8] Digi-Key: *The basics of current sensors*, https://www.digikey.com/en/articles/the-basics-of-current-sensors, [17.8.2021.]

[9] Afrić, W. *Osnovi telekomunikacija.* Split, Sveučilišni odjel za stručne studije Sveučilišta u Splitu, 2007.

[10] PIJA Education: *Serial Communication Methods - Synchronous & Asynchronous*, https://pijaeducation.com/communication/serial-communication-methods-synchronous-asynchronous/ , [20.8.2021.]

[11] Louis E. Frenzel, Jr. *Electronics Explained.* 2018.

[12] Forbes: *Battery Scarcity Could Dwarf Chip Shortage Impact On Global Auto Sales*, *https://www.forbes.com/sites/neilwinton/2021/07/27/battery-scarcity-will-dwarf-chip-shortage-impact-on-global-auto-sales-report/?sh=28e50c84363e,* [22.8.2021.]

[13] Longan labs: CANBED FD, *https://docs.longan-labs.cc/1030009/* , [20.9.2021.]

[14] BeagleBone Black Wireless, https://beagleboard.org/black-wireless, [22.9.2021.]

POPIS SLIKA

POPIS TABLICA