

Karakteristike doma narejenega fotovoltaičnega sistema s hranilnikom

(elektrotehnika)

Raziskovalna naloga

Mentor: Aleš Volčini, prof.

Avtor: Izak Virant, R 4. D

Ljubljana, marec 2025

Povzetek

V tej raziskovalni nalogi sem raziskoval način upravljanja DC-DC pretvornik z uporabo mikrokrmilnika, da bi deloval kot učinkovit solarni regulator, in potrebne elemente za izdelavo celotnega PV sistema. Izdelal sem PV sistem, ki vsebuje hranilnik proizvedene električne energije, solarni regulator, ločen nastavljiv DC izhod za polnenje baterij in zaslon, ki deluje kot GUV. Pri tem me je še posebej zanimalo, kako učinkovito je polnenje hranilnika in kako natančno oceniti raven napolnjenosti hranilnika. Poleg tega sem izdelal spletni vmesnik, ki deluje kot nadzorna plošča celotnega sistema in omogoča analizo podatkov o sistemu s pomočjo grafov.

Ključne besede: fotovoltaika, MPPT solarni regulator, ESP32, LiFePO₄, LT8705, digitalno krmiljenje, beleženje podatkov

Kazalo vsebine

1. UVOD	6
1.1 HIPOTEZE	6
2. PV SISTEMI	7
3. STROJNA OPREMA	7
3.1 MODUL ZK-SJ20	7
3.1.1 Sinhrona topologija	7
3.1.2 Digitalno krmiljenje	8
3.1.2.1 Delovanje povratnih zank.....	8
3.1.2.2 Krmiljenje z uporabo DAC MCP4728.....	10
3.1.2.3 Simulacija vezij	12
3.1.2.4 Izvedba in testiranje	13
3.1.3 Zaznavanje moči in energije	14
3.1.3.1 Vezje INA228	14
3.1.3.2 Nizkoprepustni filter	15
3.1.4 Hlajenje	15
3.1.4.1 Zaznavanje temperature	16
3.1.4.2 Upravljanje hitrosti ventilatorja	16
3.2 BATERIJA	16
3.2.1 Izbira celic	16
3.2.2 Izbira sistema za upravljanje z baterijo	17
3.2.3 Izdelava baterije	17
3.2.3.1 Ohišje.....	17
3.2.3.1 UART komunikacija.....	19
3.3 POLNILEC ZUNANJIH BATERIJ	19
3.3.1 DC-DC boost pretvornik	19
3.3.1.1 Filter šuma	20
3.3.1.2 Meritev izhodne napetosti polnilca	20
3.3.1.3 Nastavljanje izhodnega toka in napetosti.....	20
3.4 ZASLON IN MIKROKRMILNIK.....	21

3.4.1	Komunikacija s komponentami.....	21
3.4.2	Ohišje	21
3.5	OHIŠJE	22
3.5.1	3D oblikovanje	23
3.5.2	Izdelava	24
4.	PROGRAMSKA OPREMA	26
4.1	DIGITALNO KRMILJENJE	26
4.1.1	Branje podatkov vezji INA228.....	26
4.1.2	PID krmiljenje.....	27
4.2	POLNENJE BATERIJE	28
4.2.1	Komunikacija z baterijo	28
4.2.2	Ocena napoljenosti baterije.....	30
4.2.3	Algoritem polnjenja	31
4.3	MPPT ALGORITEM	32
4.4	BELEŽENJE PODATKOV	34
4.4.1	Statistika	34
4.4.2	Beleženje 15-minutnih podatkov.....	34
4.4.3	Beleženje dnevnih podatkov	36
4.4.4	Izvršitev beleženja podatkov	36
4.5	UPORABNIŠKI VMESNIK.....	37
4.5.1	Spletni uporabniški vmesnik.....	37
4.5.1.1	Strežniška stran	38
4.5.1.1	Odjemalčeva stran	38
4.5.1	Vmesnik na vgrajenem zaslonu	40
4.5.1.1	Začetna stran	40
4.5.1.2	Nastavitve	41
4.5.1.3	Trenutni podatki	41
5.	MERITVE	42
5.1	MIROVANJE	43

5.2 MOČNO PRAZNIENJE IN POLNENJE	44
5.3 TEST PREKOMERNEGA PRAZNIENJA.....	44
5.4 UČINKOVITOST MOČNOSTNEGA DELA SISTEMA.....	46
5.5 TEMPERATURE	46
5.6 MPPT ALGORITEM	47
5.7 MESEČNA ENERGIJA.....	48
6. ZAKLJUČEK	49
6.1 POTRJENE HIPOTEZE	49
7. VIRI IN LITERATURA	50

1. UVOD

Po izgradnji nove baterije za moj električni skuter, sem začel razmišljati, kako jo bom polnil. Po nakupu običajnega 84V polnilca za lion baterije sem začel razmišljati, ali bi lahko za polnenje moje baterije uporabil sončne panele. Namesto nakupa fotovoltaičnega (PV) sistema, sem se odločil, da ga bom naredil sam in se s tem veliko naučil. Želel sem napravo, ki bi lahko sprejmala električno energijo proizvedeno na sončnih panelih, jo shraniti ter polniti električni skuter tudi ponoči. Ob tem sem želel še shranjevati vse podatke o pretoku energije, temperaturi, tokovih, napetostih, močeh itd. in seveda tudi ustrezno vizualizirati. Zato sem začel raziskovati PV sisteme, njihove zahteve, zgradbo, učinkovitost itd.

Na tem mestu bi rad še dodal, da se v tej nalogi nisem spuščal v vsako podrobnost in da je za boljše razumevanje dobro poznati delovanje tradicionalnih PV sistemov, algoritma MPPT in še nekaterih drugih stvari, ki niso neposredno predmet te naloge.

Shema celotnega vezja je v prilogi 1.

Program in ostale datotetke si lahko ogledate v git repozitoriju:

<https://gitlab.vegova.si/ivirant/solarni-polnilec>.

1.1 HIPOTEZE

Moje začetne domneve so bile, da:

- je iz široko dostopnega DC-DC pretvornika mogoče narediti MPPT solarni polnilnik z visoko učinkovitostjo,
- lahko s pomočjo merilnika energije dobro ocenimo raven napolnjenosti hranilnika,
- je možno DC-DC pretvornik krmiliti s pomočjo D/A pretvornikov za izvajanje MPPT,
- lahko sestavim LiFePO4 baterijo z vsemi potrebnimi varnostnimi sistemi ceneje, kot pa da bi jo kupil.

2. PV SISTEMI

Fotovoltaični (PV) sistemi predstavljajo ključen del sodobnih rešitev za pridobivanje obnovljive energije, saj omogočajo neposredno pretvorbo sončne svetlobe v električno energijo. Zaradi svoje okolju prijazne narave in enostavne uporabe so PV sistemi postali izjemno priljubljeni, tako v manjših domačih aplikacijah kot tudi v obsežnih industrijskih in komercialnih projektih.

3. STROJNA OPREMA

V nadeljevanju bom predstavil strojno opremo, module in komponente, ki sem jih potreboval za končni izdelek in izvedbo meritev.

3.1 MODUL ZK-SJ20

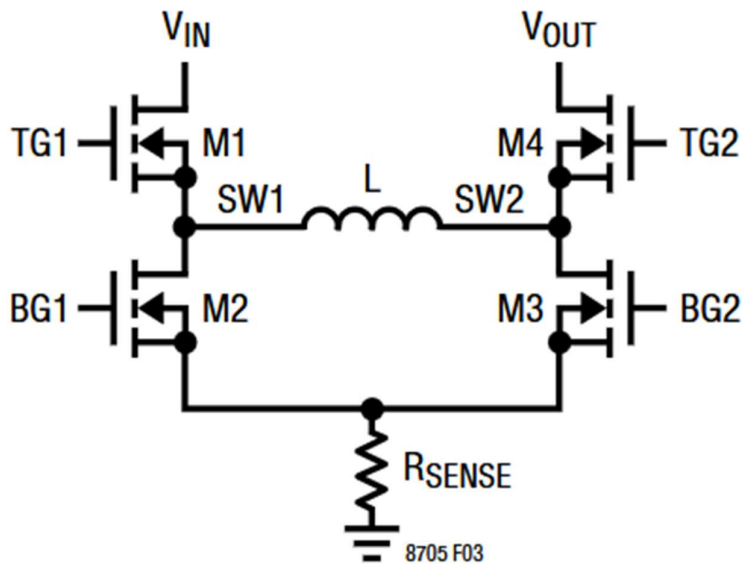
Modul ZK-SJ20 je sinhroni DC-DC »buck-boost« pretvornik, ki temelji na vezju LT8705 podjetja Analog Devices. Je najpomembnejši del mojega PV sistema, saj s pomočjo digitalnega krmiljenja in mojega prilagojenega MPPT algoritma učinkovito polni notranjo LiFePO4 baterijo.

Za ta modul sem se odločil ker ima:

- dovolj širok razpon vhodne in izhodne napetosti (od 10 V do 80 V),
- ima maksimalno tokovno omejitev 20 A,
- temelji na sinhroni topologiji s štirimi FET tranzistorji,
- ima možnost omejevanja vhodne napetosti, izhodne napetosti in izhodnega toka.

3.1.1 Sinhrona topologija

Vezje LT8705 temelji na sinhroni topologiji delovanja s pomočjo štirih FET tranzistorjev v konfiguraciji H-mostu (slika spodaj).



Slika 1: H-most v LT8705

Tu je učinkovitost višja predvsem zaradi aktivnega nadzora obeh FET tranzistorjev v vezju. Medtem ko pri asinhronih (diodnih) topologijah spodnji element deluje kot pasivna dioda, (ki ima običajno večji napetostni padec in počasnejši preklop,) sinhrona zasnova nadomesti diodo z aktivnim FET tranzistorjem. To pomeni, da se izgube zaradi napetostnih padcev zmanjšajo, kar privede do nižjih toplotnih izgub in večje pretvorbene učinkovitosti.

3.1.2 Digitalno krmiljenje

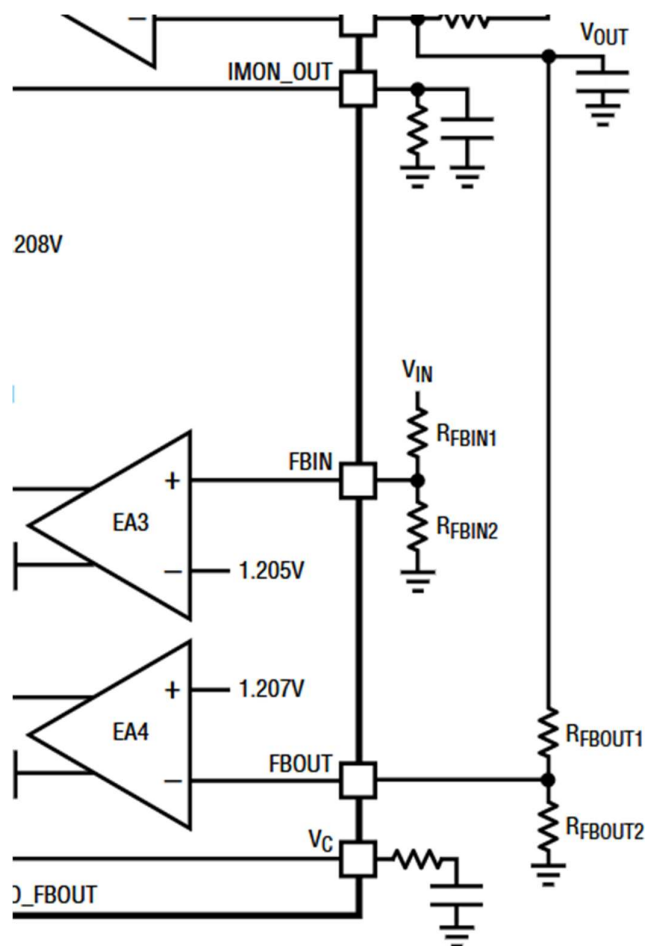
Modul je bilo prvotno možno krmiliti le preko potenciometrov, kar sem spremenil s pomočjo digitalno-analognih pretvornikov (DAC). Vezje LT8705 ima na voljo štiri povratne zanke, od katerih so na moje modulu na voljo tri.

3.1.2.1 Delovanje povratnih zank

Vezje LT8705 je potrebno krmiliti s pomočjo povratne zanke, ki vezju posreduje trenutno napetost ali tok, ki ga proizvaja. V primeru da želimo upravljati izhodno napetost, to lahko storimo preko napetostnega delilnika, ki del izhodne napetosti posreduje v povratno zanko.

Izhod napetostnega delilnika peljemo na povratni pin, ki to napetost primerja z referenčno napetostjo okoli 1,2 V. V primeru, da je ta napetost manjša od referenčne, bo vezje začelo povečevati tok čez induktor, kar bo povečalo izhodno napetost. S tem

se bo povečala napetost na povratnem pinu, kar bo izhodno napetost spravilo na željeno vrednost. S tem stabiliziramo izhodno napetost (slika spodaj).



Slika 2: Vhodna in izhodna povratna zanka

Spodnji enačbi nam povesta, kako sta v zgornjem vezju določeni vhodna in izhodna napetost.

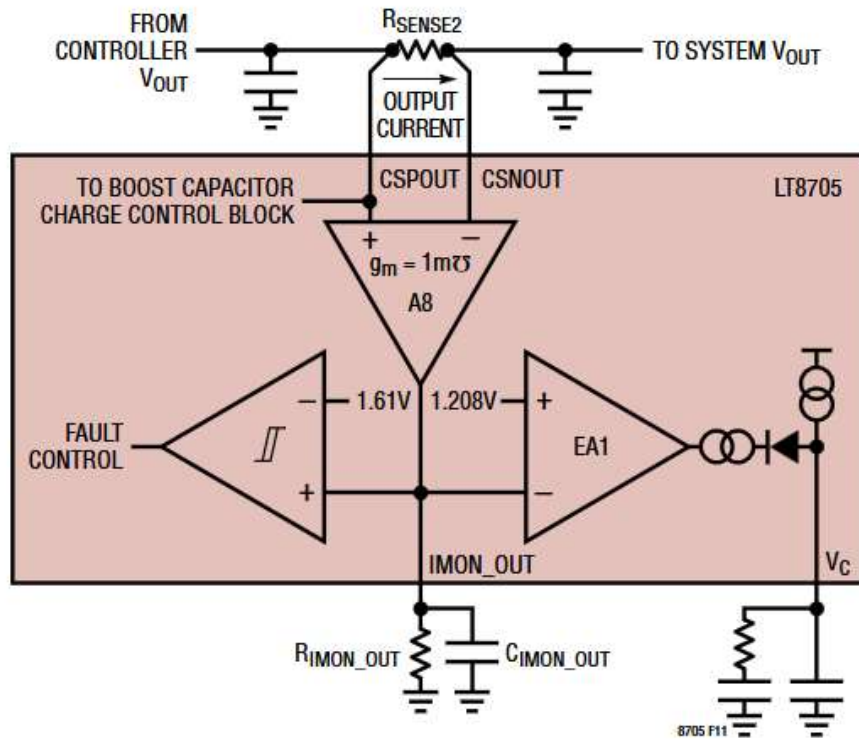
Enačba za izračun izhodne napetosti:

$$V_{OUT} = 1.207V \times \left(1 + \frac{R_{FBOUT1}}{R_{FBOUT2}}\right) V$$

Enačba za izračun omejitve vhodne napetosti:

$$V_{IN(MIN)} = 1.205V \times \left(1 + \frac{R_{FBIN1}}{R_{FBIN2}}\right) V$$

Spodnja slika prikazuje omejljnik izhodnega toka:



Slika 3: Omejljnik izhodnega toka

Omejitev izhodnega toka nastavimo preko R_{IMON_IN} upora in sicer po tej enačbi:

$$R_{IMON_OUT} = \left(\frac{1.208V}{I_{RSENSE(LIMIT)} \times 1m \frac{A}{V} \times R_{SENSE2}} \right) \Omega$$

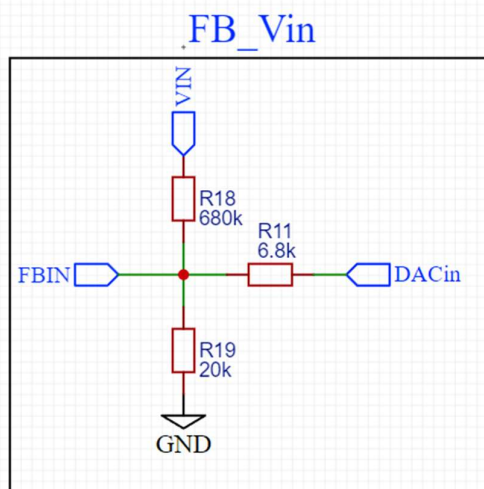
Tukaj priključek IMON_OUT proizvede tok, ki je sorazmerno odvisen od padca napetosti na uporu R_{SENSE2} v razmerju 1 mA / V. Če je napetost na IMON_OUT večja od 1.208 V bo LT8705 začel omejevati izhodni tok (slika zgoraj).

3.1.2.2 Krmiljenje z uporabo DAC MCP4728

Namesto ročnega vrtenja potenciometrov ali nastavljanja uporov, sem le-te zamenjal z digitalno analognimi pretvorniki. Slednji omogočajo programsko upravljanje DC-DC pretvornika.

Uporabil sem MCP4728, ki vsebuje štiri 12-bitne DAC. Uporabljam ga v načinu z zunanjo referenčno napetostjo, ki je enako napajalni napetosti. To pomeni da lahko

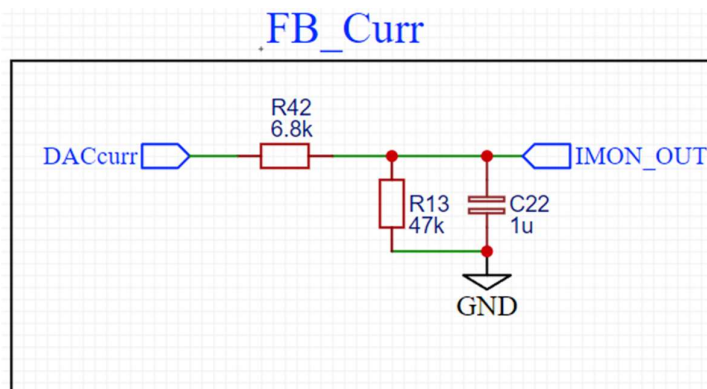
dosežem napetosti od 0 V do 3.3 V v 0.8 mV korakih, kar je ravno dovolj precizno za mojo uporabo.



Slika 4: del povratne zanke za omejevanje vhodne napetosti

Na zgornji sliki FBIN označuje priključek za povratno zanko vhodne napetosti, VIN je vhodna napetost in DACin je on od štirih izhodov DAC. Upori so del povratne zanke. S pomočjo upora R11 in napetosti iz DAC reguliramo padca napetosti skozi upora R18 in R19, kar omogoča nastavljanje omejitve vhodne napetosti.

Vezje za nastavljanje izhodne napetosti deluje na enak princip in je vidno v shemi celotnega vezja.



Slika 5: Vezje za omejevanje izhodnega toka

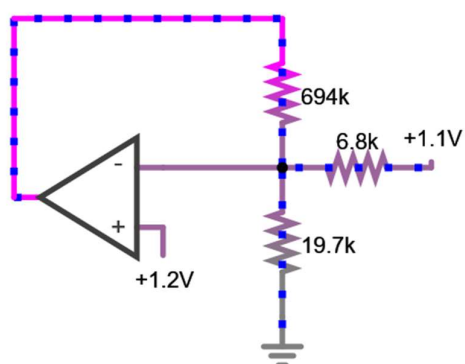
Napetost na IMON_OUT določata tako upor R42 kot tudi napetost na DACcurr, kar posledično omogoča regulacijo izhodnega toka.

3.1.2.3 Simulacija vezij

Pred praktičnim testiranjem, sem vsa vezja za digitalno krmiljenje simuliral v programu Falstad. S pomočjo simulacije sem ustrezno izbral vrednosti uporov in potrdil delovanje vezij.

Hotel sem doseči, da:

- je napetost na DAC obratno sorazmerna z izhodno,
- da izhodna napetost ne presega 80 V in napetost na DAC ne presega 2 V,
- tok ki teče skozi DAC ne presega toka ki ga je ta zmožen,
- upori so standardnih vrednosti, ki sem jih imel doma.



Slika 6: simulacija povratne zanke v programu Falstad

Za regulacijo vhodne in izhodne napetosti sem uporabil operacijski ojačevalnik z referenčno napetostjo 1,2 V povezano na neinvertirajoč vhod.

Rezultat simulacije vhodne in izhodne napetosti:

- napetost je okoli 80 V pri okoli 0,82 V napetosti na DAC,
- napetost je okoli 2 V pri okoli 1,6 V napetosti na DAC,
- ločljivost je okoli 50 mV na en korak DAC.

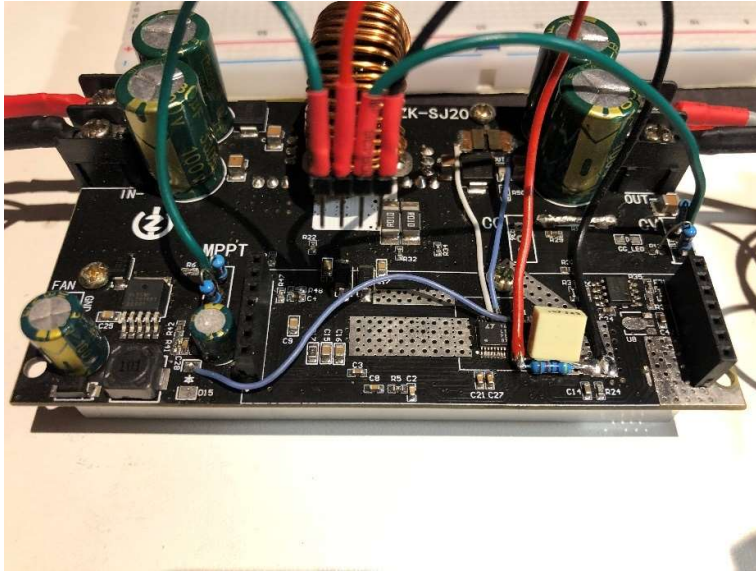
Rezultat simulacije izhodnega toka:

- tok je okoli 20 A pri okoli 0,84 V napetosti na DAC,
- tok je okoli 0,15 A pri okoli 1,38 V napetosti na DAC,

- ločljivost je okoli 18 mA na en korak DAC.

3.1.2.4 Izvedba in testiranje

Po pričakovanih rezultatih simulacije je nastopil čas, da realiziram opisano vezje tudi z elektroniko (slika spodaj).



Slika 7: DC-DC krmilnik in vezja za digitalno upravljanje brez DAC

Za določitev programskih limit pri D/A pretvorbi sem uporabil nekaj različnih testov.

Za kalibracijo izhodnega toka sem uporabil kratkostični test, pri katerem sem s pomočjo zunanjega tokovnega merilnika meril tok na izhodu. S pomočjo testnega programa sem upravljal DAC in opazoval izhodni tok. Pri tem sem si beležil pri kateri vrednosti dosežem na izhodu 20 A in pri kateri dosežem 0 A.

Izhodno napetost sem kalibriral tako, da sem med spreminjanjem vrednosti iz DAC meril napetost na izhodu in sicer tako, da sem začel z varno vrednostjo, ki sem jo pridobil v simulaciji. Medtem, ko sem zniževal napetost iz DAC, sem lahko opazoval pričakovano dvigovanje izhodne napetosti.

Za kalibracijo vhodne napetosti sem uporabil tokovno omejen vir napetosti, s katerim sem napajal vezje. Enako kot pri kalibraciji izhodne napetosti sem opazoval vhodno napetost ob spreminjanju napetosti iz DAC.

S pomočjo teh testov sem ugotovil limitne vrednosti D/A pretvornikov, ki jih ne smem preseči.

3.1.3 Zaznavanje moči in energije

Za delovanje sistema je potrebno zaznavanje moči in energije na treh različnih točkah:

- na vhodu v DC-DC krmilnika, po PV panelah,
- na izhodu DC-DC krmilnika, pred baterijo,
- na bateriji, med baterijo in izhodom.

Te točke so na shemi v prilogi 1 označene s Power_sense_IN, OUT in BAT.

3.1.3.1 Vezje INA228

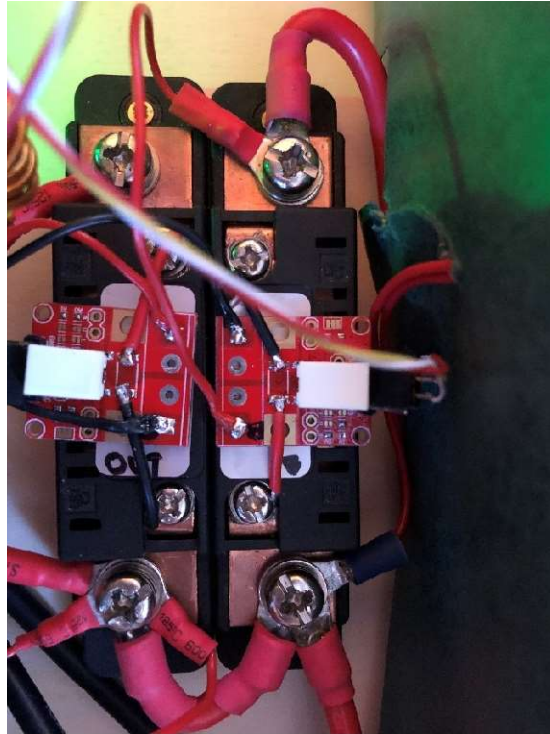
Za zaznavanja moči in energije sem izbral vezje INA228, ker:

- lahko hkrati zaznava napetost, tok, moč in energijo,
- ima dovolj visoko raven zaznavanja napetosti, do 85 V,
- je dovolj natančen z 20-bitnim A/D pretvornikom,
- ima vodilo I2C.

Vezje INA228 meri omenjene veličine preko shunt upora, ki proizvede padec napetosti (slika spodaj).

Odločil sem se za 0,75 m Ω upor, ker:

- ima pri toku 50 A samo 1,875 W izgube,
- ko je INA228 v načinu ± 40.96 mV za merjenje toka, lahko z njim merim tokove do 54,61 A.



Slika 8: Merilnika moči za izhod in baterijo

3.1.3.2 Nizkoprepustni filter

Za natančne meritve toka, je potrebno napetostni padec filtrirati z nizkoprepustnim filtrom, ki ne prepušča visokofrekvenčnega šuma. Pri merjenju tokov sem imel kar nekaj problemov zaradi šuma DC-DC pretvornikov. Brez tega filtra so bile meritve precej nezanesljive in neponovljive. Z uporabo tega filtra so ti problemi postali precej manjši.

To vezje vidimo v bloku Power_sense_IN (priloga 1), upora R50, R51 in kondenzator C25.

Formula za izračun mejne frekvence filtra:

$$f_c = \left(\frac{1}{\pi RC} \right) \text{Hz}$$

Za upor sem si izbral 100Ω za kondenzator pa $1 \mu\text{F}$, kar pomeni da je mejna frekvenca 3183Hz .

3.1.4 Hlajenje

Pri testiranju DC-DC krmilnika sem opazil, da je pasivno hlajenje premalo, da bi odvedlo vso odvečno toploto.

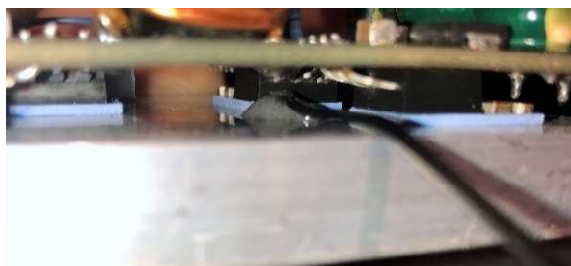
Odločil sem se uporabiti 5 V PWM nastavljen ventilator za hlajenje vezja in temperaturni senzor za zaznavanje temperature hladilnega rebra.

3.1.4.1 Zaznavanje temperature

Za zaznavanje temperature sem uporabil 10 k Ω NTC upor, v vezju z 5,1 k Ω uporom in 0,1 μ F kondenzatorjem. Te komponente vidimo v bloku Temp_sensor (priloga 1).

Za natančno meritev temperature močnostnih komponent sem temperaturni senzor prilepil na hladilno rebro v bližino FET tranzistorjev.

Za natančen izračun temperature sem uporabil vezje ADS1115, ki je I2C ADC, s katerim zaznavam napetostni padec na upor R41 ter napetost 3,3 V vira.



Slika 9: Prilepljen temperaturni senzor

3.1.4.2 Upravljanje hitrosti ventilatorja

Namesto, da ventilator samo prižgemo ali pa ugasnemo, ko je temperatura previsoka, sem se odločil da ga bom upravljal s pomočjo PWM.

Krmilnik ESP32 proizvede PWM signal, ki je priključen na S priključek ventilatorja. Izbral sem frekvenco 10 kHz, da je upravljanje hitrosti neslišno, saj se nižje frekvence slišijo iz brnenja ventilatorja.

3.2 BATERIJA

Vsak PV sistem potrebuje baterijo za svoje delovanje. Namenjena je shranjevanju električne energije za takrat, ko sončne energije ni.

3.2.1 Izbira celic

Izbral sem si celice EVE C40 zaradi naslednjih lastnosti:

- so kemijske sestave LiFePO₄ in so veliko varnejše kot druge sestave,

- omogočajo veliko število ciklov praznenja in polnenja (5000-10000) pred izgubo 80 % svoje kapacitete,
- so majhne po velikosti, glede na njihovo rezmerje med ceno in kapaciteto,
- omogoča komunikacijo preko UART prenosa.

3.2.2 Izbira sistema za upravljanje z baterijo

Vsaka baterija potrebuje t.i. BMS (battery management system), ki lahko izklopi polnenje ali praznenje baterije in s tem zavaruje baterijske celice proti neštetim grožnjam: premajhni napetosti, preveliki napetosti, preveliki temperaturi, premajhni temperaturi, neizravnani napetosti celic...

Izbral sem JK-BD6A24S10P zaradi naslednjih lastnosti:

- omogoča dovolj velik tok praznenja in polnenja do 100 A,
- za izravnavanje napetosti celic uporablja aktivno izravnavanje, katero prazni celico z najvišjo napetostjo in s to energijo polni celico z najnižjo napetostjo, kar je bolj učinkovito kot da samo praznimo najvišjo preko upora,
- ima veliko zaščit in nastavitev s katerimi bolje zaščitimo baterijo.

3.2.3 Izdelava baterije

Vsaka baterija je sestavljena iz večih celic, ki so povezane bodisi zaporedno bodisi vzporedno ali kombinacija. Z zaporednimi povezavami pridobimo na napetosti, pri vzporednih povezavah pa na kapaciteti baterije.

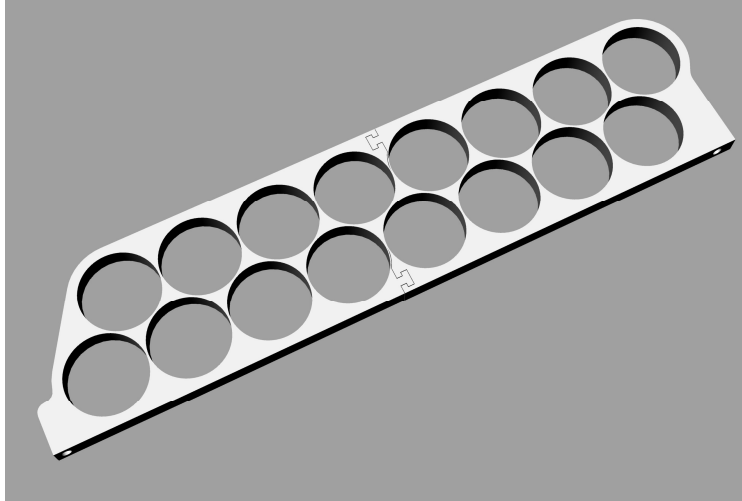
Odločil sem se za uporabo konfiguracije 16s0p, kar pomeni da je 16 celic povezano zaporedno in nič vzporedno. Za to sem se odločil ker sem želel da je nazivna napetost baterije 51,2 V in nisem želel večje kapacitete zaradi visoke cene.

Podatki baterije so:

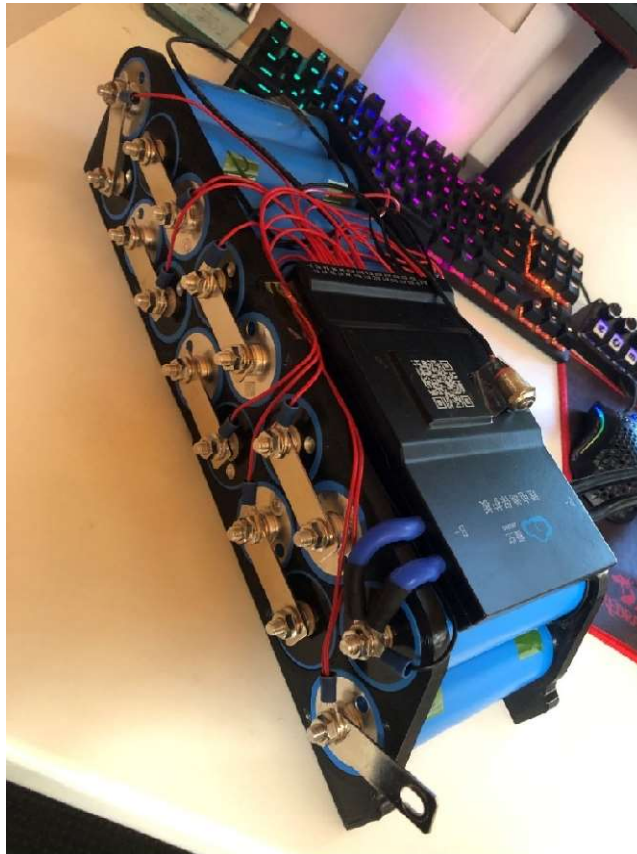
- 51,2 V nazivna napetost,
- 1024 Wh energije,
- 60 A maksimalni tok praznenja, 10 A konstantni tok praznenja.

3.2.3.1 Ohišje

Ohišje baterije sem izrisal in natisnil na 3D tiskalniku, ta drži skupaj vse celice in BMS, da se pri uporabi ne premikajo (slika spodaj).



Slika 10: 3d model enega dela ohišja

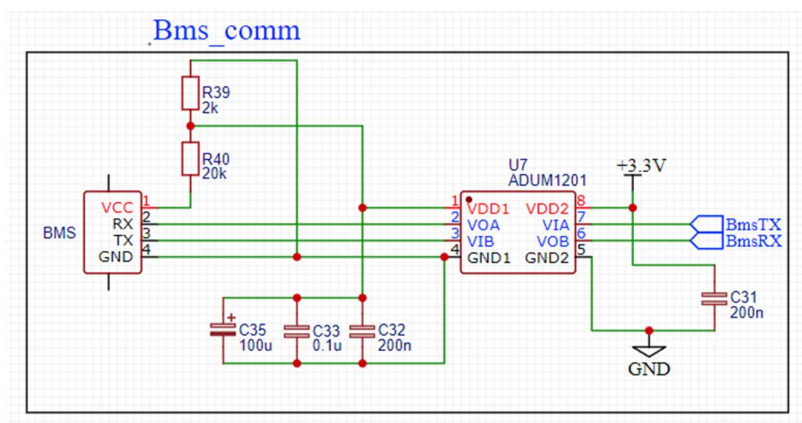


Slika 11: Končana baterija

Vse celice so vstavljene med 3d natisnjenima platnicama, povezave med celicami in žice za izravnavanje ter merjenje napetosti so privijačene, BMS pa je nalepljen z dvostranskim lepilom zgoraj (slika zgoraj).

3.2.3.1 UART komunikacija

Sistem o informacijah baterije dobi preko UART vrat. Da je povezava med baterijo in mikrokontrolerom ESP32 galvanjsko ločena, sem uporabil vezje ADUM1201 (slika spodaj).



Slika 12: Vezje za komunikacijo ločuje UART od glavnine vezja

ADUM1201 zagotovi, da negativna povezava baterije ne napaja drugih delov sistema, če je BMS izklopil praznenje. V nasprotnem primeru bi se lahko zgodilo, da bi drugi deli sistema še vedno dobili napajanje preko GND povezave na komunikacijskem konektorju, kar bi spraznilo celice pod njihovo minimalno dovoljeno napetost.

3.3 POLNILEC ZUNANJIH BATERIJ

Za zunanje porabnike, ki potrebujejo višjo napetost, kot je napetost notranje baterije, je potreben DC-DC boost pretvornik. Njegov namen je, da iz nižje napetosti naredi višjo, s katero potem lahko polnimo baterije ali napajamo porabnike.

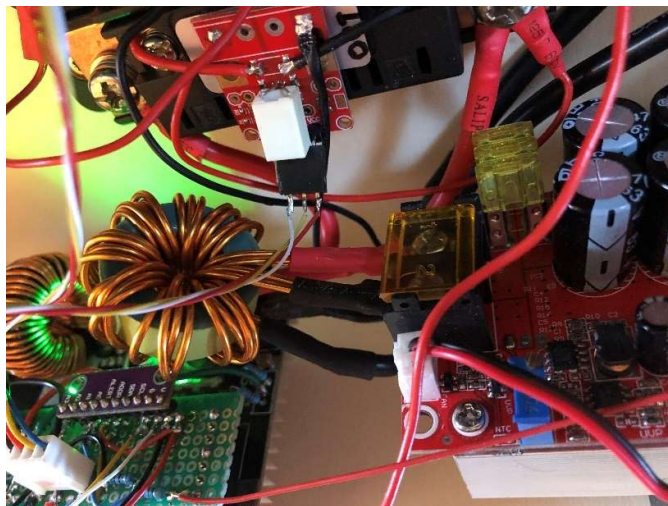
3.3.1 DC-DC boost pretvornik

Izbral sem 1500 W DC-DC boost pretvornik, ker ima zelo nizko ceno, velik razpon izhodne napetosti od 60 V - 90 V, je dovolj močan za polnjenje večjih baterij in ima možnost nastavitve tokovnega omejevanja, kar je nujno za polnjenje baterij.

Ta je povezan neposredno na LoadOUT povezavo (glej prilogo 1).

3.3.1.1 Filter šuma

Po testiranju polnilca sem ugotovil, da proizvede veliko šuma v vezju, ki vpliva na ostale elemente v sistemu. Vezja za zaznavanje moči so pri večjem toku polnilca začela kazati napačne in nestabilne meritve.



Slika 13: Dušilka na vhodu v polnilac

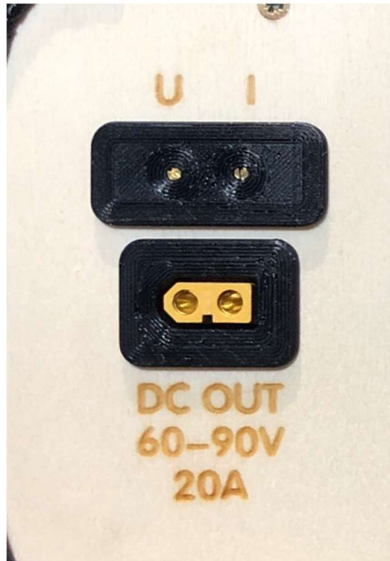
Odločil sem pred polnilac dodati dušilko, narejeno iz feritnega jedra, okoli klaterega sem zavil 7 obratov debele lakirane žice (plus in minus skupaj). S tem sem dobil nizkoprepustni filter, ki je odpravil težavo nestabilnih meritev (slika zgoraj).

3.3.1.2 Meritev izhodne napetosti polnilca

Enako kot pri meritvi temperature sem uporabil vezje ADS1115, povezano preko napetostnega delilnika na izhod polnilca (blok Ext_voltage_snese v prilogi 1).

3.3.1.3 Nastavljanje izhodnega toka in napetosti

Napetost in maksimalni tok polnilca se nastavlja preko dveh potenciometrov, ki sem jih prestavil iz vezja na zunanjo stran ohišja (slika spodaj).



Slika 14: Potenciometra za nastavljanje polnilca in izhod polnilca

3.4 ZASLON IN MIKROKRMILNIK

Celoten sistem upravlja ESP32-2432S028R (t.i. »cheap yellow display«).

Izbral sem ga zaradi:

- minimalističnega izgleda
- krmilnika ESP32,
- kapacitivnega TFT zaslona,
- bralnika SD kartic.

3.4.1 Komunikacija s komponentami

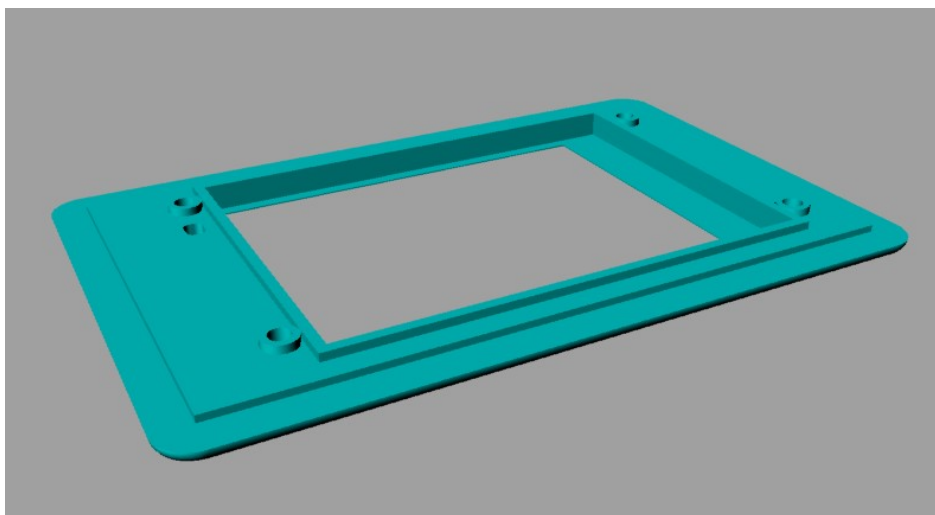
ESP32 do drugih modulov v sistemu komunicira preko dveh protokolov, I2C in UART. Za komunikacijo z zaslonom, senzorjem za dotik in SD kartico pa preko SPI protokola.

Za komunikacijo z BMS-om sem moral sprostiti nekaj priključkov, ki so bili zasedeni zaradi LED luči, ki je nisem potreboval.

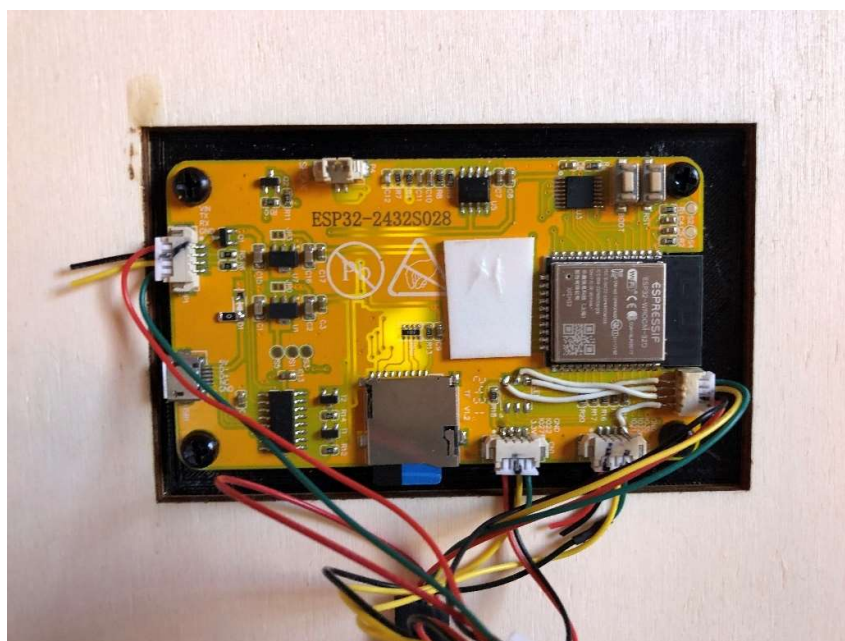
Za komunikacijo s senzorjem dotika sem moral uporabiti »bitbang« knjižnico, saj sta obe SPI vodili že zasedeni z SD kartico in TFT zaslonom.

3.4.2 Ohišje zaslona

Ohišje za zaslon sem oblikoval in 3D natisnil (sliki spodaj).



Slika 15: 3D model ohišja za zaslon



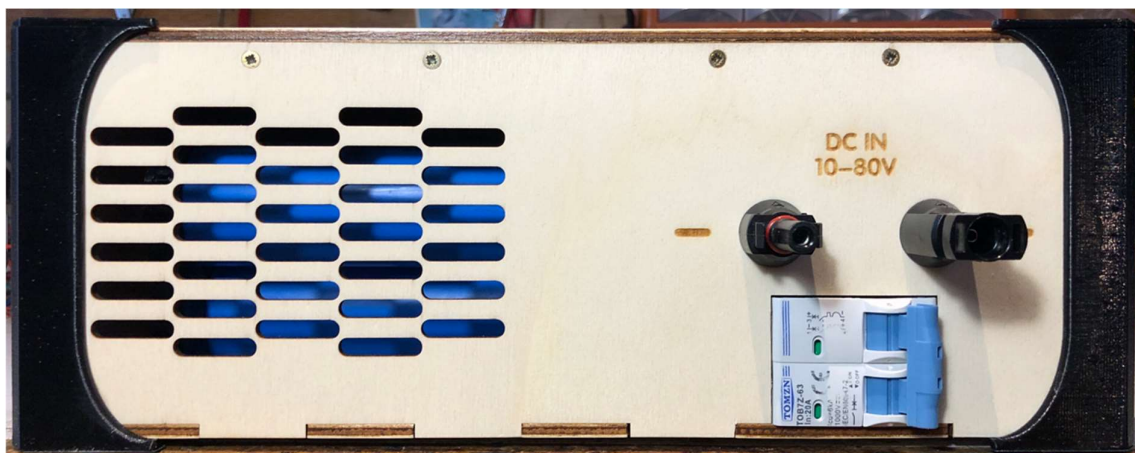
Slika 16: 3D natisnjeno ohišje za zaslon

3.5 OHIŠJE

Ohišje sistema sem izdelal iz vezane plošče debeline 4 mm in 3D natisnjenih komponent. Vezano ploščo sem izbral tako iz okoljskega kot tudi estetskega vidika. Skupaj s črnimi 3D natisnjenimi deli daje lesu kontrast in izboljša izgled.

3.5.2 Izdelava

Lesene dele sem s 3D natisnjenimi kotniki privijačil in zalepil skupaj. Baterija je zaradi svoje teže z večjimi vijaki privijačena od spodaj.



Slika 19: Leva stran ohišja

Na zgornji sliki vidimo levo stran ohišja, kjer je vhod za sončne panele z MC4 konektorjema. Pod njima je 20A varovalka, ki služi tudi kot stikalo, saj izklopi oba pola.



Slika 20: Desna stran ohišja

Na nasprotni strani ohišja (slika zgoraj) je izhod za porabnike in izhod polnilca baterij. Nad izhodom za polnilec sta še dva trimerja za nastavljanje najvišje izhodne napetosti in toka. Desno od izhoda za porabnike je stikalo za vklop sistema.

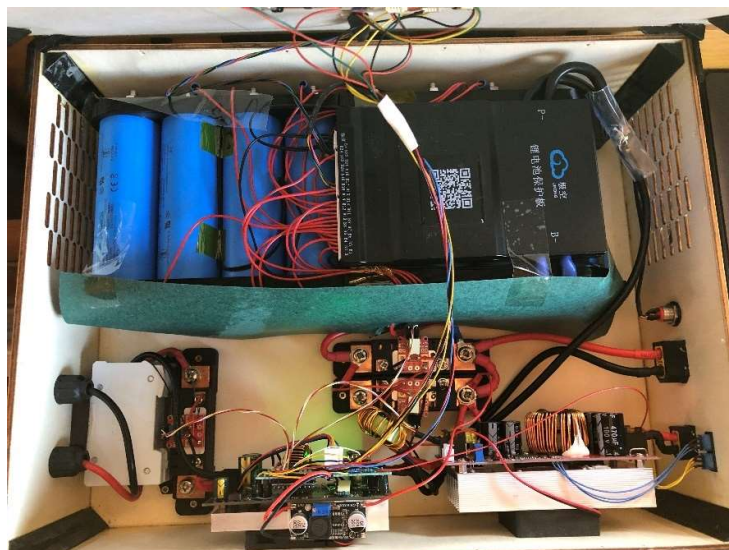
Spodaj sledijo slike sprednje in zgornje strani ohišja ter notranjosti.



Slika 21: Sprednja stran ohišja



Slika 22: Zgornja stran ohišja



Slika 23: Razpored komponent v ohišju

4. PROGRAMSKA OPREMA

Program je napisan za krmilnik ESP32. Pri programiranju sem uporabil programsko okolje Arduino IDE. Naprava ima dva uporabniška vmesnika, enega na preko spletne strani in drugega na vgrajenem TFT zaslonu na dotik. Slednji je sestavljen in vizualno sprogramiran v programu EEZ Studio. Spletni vmesnik v programskem jeziku javascript, html in css.

V nadaljevanju bom predstavil samo pomembne dele kode, ker bi predstavitev celotne kode vzela preveč prostora.

Vso kodo si je možno ogledati v git repozitoriju <https://gitlab.vegova.si/ivirant/solarni-polnilec>.

4.1 DIGITALNO KRMILJENJE

Na kratko bom predstavil programski del digitalnega krmiljenja DC-DC pretvornika (ZK-SJ20).

4.1.1 Branje podatkov vezji INA228

Podatke vezja INA228 berem s pomočjo knjižnice INA228.h od Roba Tillaarta. Za hitro odzivnost sistema, vse podatke iz senzorjev berem vsakih 10ms. Preden podatek shranim v spremenljivko preverim, če je v območju meritev, sicer ga zavržem. Včasih se namreč zgodi, da prispejo vrednosti za katere zgloda kot ,da z dejanskim stanjem nimajo nobenega opravka. Razloga nisem iskal, vrjetno je problem v knjižnici.

Za pridobivanje podatkov o trenutnem pretoku energije, ob vsakem branju preverim razliko s prejšnjim, in jo seštejem ali odštejem celoti glede na predznak toka, kot vidimo na izseku kode spodaj:

```
double rawenergy;
double dif;

float temp;
temp = inaDataIn.volt;
inaDataIn.volt=inaIn.getBusVoltage();
if(inaDataIn.volt > 100 || inaDataIn.volt < -10)
    inaDataIn.volt = temp;

temp = inaDataIn.curr;
inaDataIn.curr=inaIn.getCurrent();
if(inaDataIn.curr > 100 || inaDataIn.curr < -100)
    inaDataIn.curr = temp;

temp = inaDataIn.pow;
inaDataIn.pow=inaIn.getPower();
if(inaDataIn.pow > 5000 || inaDataIn.pow < -100)
```

```

    inaDataIn.pow = temp;

    if(inaDataIn.curr < 0)
        inaDataIn.pow*=-1;

    rawenergy = inaIn.getEnergy()/3600;
    dif = rawenergy - prevrawenergyIn;
    if(dif > 0 && dif < 1){
        if(inaDataIn.curr>0){
            inaDataIn.energy += dif;
        }else
            inaDataIn.energy -= dif;
        prevrawenergyIn = rawenergy;
    }
}

```

Izsek kode 1: Merjenje pretoka energije.

4.1.2 PID krmiljenje

Ko sem testiral krmiljenje DC-DC pretvornika z D/A pretvorniki, sem ugotovil, da preprosta kalibracija z minimumom in maksimumom ni natančna. Odločil sem se, da bom za bolj natančno krmiljenje in bolj točno sledenje želeni vrednosti uporabil PID krmilnik s podatki s senzorjev v povratni zanki.

```

void computePID(){
    static unsigned long lastTime = 0;
    static float lastErrorVin = 0;
    static float lastErrorCurr = 0;
    static float lastErrorVout = 0;

    static float integralVin = 0;
    static float integralCurr = 0;
    static float integralVout = 0;

    unsigned long now = micros();

    float timeChange = now - lastTime;
    timeChange/=1000.0;
    if (timeChange >= pid_sample_rate){
        float errorVin = (vin_setpoint/100.0)-inaDataIn.volt;
        float errorCurr = (curr_setpoint/100.0)-inaDataBat.curr;
        float errorVout = (vout_setpoint/100.0)-inaDataOut.volt;

        if(avgErrReset){
            errVin.reset();
            errVout.reset();
            errCurr.reset();
            avgErrReset = false;
        }

        errVin.add(errorVin);
        errVout.add(errorCurr);
        errCurr.add(errorVout);

        integralVin += errorVin * timeChange;
        integralCurr += errorCurr * timeChange;
        integralVout += errorVout * timeChange;

        integralVin=constrain(integralVin, 300, 3200);
        integralCurr=constrain(integralCurr, 0, 600);
        integralVout=constrain(integralVout, 300, 3000);

        float derivativeVin = (errorVin - lastErrorVin) / timeChange;
        float derivativeCurr = (errorCurr - lastErrorCurr) / timeChange;
    }
}

```

```

float derivativeVout = (errorVout - lastErrorVout) / timeChange;

dac_vin_value = (p_vin * errorVin) + (i_vin * integralVin) + (d_vin * derivativeVin);
dac_curr_value = (p_curr * errorCurr) + (i_curr * integralCurr) + (d_curr *
derivativeCurr);
dac_vout_value = (p_vout * errorVout) + (i_vout * integralVout) + (d_vout *
derivativeVout);

dac_vin_value=fmap(constrain(dac_vin_value, 10, 69), 10, 69, dac_vin_max_value,
dac_vin_min_value);
dac_curr_value=fmap(constrain(dac_curr_value, 1, 20), 1, 20, dac_curr_max_value,
dac_curr_min_value);
dac_vout_value=fmap(constrain(dac_vout_value, 10, 69), 10, 69, dac_vout_max_value,
dac_vout_min_value);

if(!stop){
mcp.setChannelValue(MCP4728_CHANNEL_A, int(dac_vin_value));
mcp.setChannelValue(MCP4728_CHANNEL_B, int(dac_curr_value));
mcp.setChannelValue(MCP4728_CHANNEL_C, int(dac_vout_value));
}
else{
mcp.setChannelValue(MCP4728_CHANNEL_A, 2500);
mcp.setChannelValue(MCP4728_CHANNEL_B, 2500);
mcp.setChannelValue(MCP4728_CHANNEL_C, 2500);
}

lastErrorVin = errorVin;
lastErrorCurr = errorCurr;
lastErrorVout = errorVout;

lastTime = now;
}
}

```

Izsek kode 2: Funkcija, ki regulira povratno zanko.

Program kliče zgornjo funkcijo ob vsaki ponovitvo glavne zanke, vendar pa se ta zares izvede le na vsaki 2 ms. Funkcija izračuna vrednosti za vse tri povratne zanke in jih sporoči D/A pretvornikom z `mcp.setChannelValue()`.

V primeru, ko zanka ni v uporabi in je razlika med željeno in pravo vrednostjo prevelika, poskrbimo, da integralna vrednost ne narašča ali pada v neskončnost (ang. integral windup):

```
integralVin=constrain(integralVin, 300, 3200);
```

Izsek kode 3: Omejitev integralne vrednosti.

4.2 POLNENJE BATERIJE

V temu poglavju bom predstavil polnenja baterije (hranilnika) s programskega vidika.

4.2.1 Komunikacija z baterijo

Za BMS, ki sem ga uporabil, sem po kratkem iskanju na spletu obupal in del programa za komunikacijo napisal kar sam. Koda ni popolna in omogoča branje samo tistih

podatkov, ki sem jih potreboval za moj sistem. Pri pisanju sem si pomagal s podatkovnim listom protokola, ki ga BMS uporablja.

Vse podatke pridobljene iz BMS-ja, shranjujem v strukturo `BMS_Data`:

```
struct Cell{
    byte number;
    int voltage;
};
struct BMS_Data{
    Cell cell[24];
    int numCells;
    int MOStemp;
    int BATtemp1;
    int BATtemp2;
    float TotalVoltage;
    float Current;
    int SOC;
};

class JK_BMS{
    HardwareSerial *serialPort = NULL;
    void sendData();
    bool processData(byte [], int &);

public:
    BMS_Data bmsData;
    void setSerialPort(HardwareSerial *);
    bool getData();
};

void JK_BMS::sendData(){
    byte request[] = {0x4E, 0x57, 0x00, 0x13, 0x00, 0x00, 0x00, 0x00, 0x06, 0x03, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x68, 0x00, 0x00, 0x01, 0x29};
    unsigned int requestSize = sizeof(request);
    for(int i = 0; i < requestSize; i++){ serialPort->write(request[i]); }
}
void JK_BMS::setSerialPort(HardwareSerial *port){ serialPort = port; }
bool JK_BMS::getData(){
    byte receivedData[350];
    int receivedDataSize = 0;
    while(serialPort->available()){
        // beri iz serijca
        byte incomingByte = serialPort->read();
        // Store the received byte in the buffer
        receivedData[receivedDataSize] = incomingByte;
        receivedDataSize++;
    }
    sendData();
    if(receivedDataSize and processData(receivedData, receivedDataSize)){ return true; }
    return false;
}
bool JK_BMS::processData(byte data[], int &dataSize){
    //preverimo če so podatki pravilno prišli
    if (dataSize >= 18 && data[0] == 0x4E && data[dataSize - 5] == 0x68 && data[11] ==
0x79) {
        //zapolnimo število celic
        bmsData.numCells = data[12] / 3; //delimo s 3 ker vsaka zavzema 3 bajte
        if (dataSize >= 13 + bmsData.numCells * 3 + 5) {
            int batteryIndex = 0;
            //pridobivanje podatkov o posameznih celicah
            for (int i = 0; i < bmsData.numCells * 3; i += 3) {
```

```

        bmsData.cell[batteryIndex].number = data[13 + i];
        bmsData.cell[batteryIndex].voltage = (data[13 + i + 1] << 8) | data[13 + i + 2];
        batteryIndex++;
    }
    //pridobivanje drugih podatkov
    bmsData.MOStemp = (data[13 + bmsData.numCells * 3+1] << 8) | data[13 +
bmsData.numCells * 3 + 2];
    //računanje prave temperature
    if(bmsData.MOStemp>=100) bmsData.MOStemp=100-bmsData.MOStemp;
    bmsData.BATtemp1 = (data[13 + bmsData.numCells * 3 + 4] << 8) | data[13 +
bmsData.numCells * 3 + 5];
    if(bmsData.BATtemp1>=100) bmsData.BATtemp1=100-bmsData.BATtemp1;
    bmsData.BATtemp2 = (data[13 + bmsData.numCells * 3 + 7] << 8) | data[13 +
bmsData.numCells * 3 + 8];
    if(bmsData.BATtemp2>=100) bmsData.BATtemp2=100-bmsData.BATtemp2;
    bmsData.TotalVoltage= (data[13 + bmsData.numCells * 3 + 10] << 8) | data[13 +
bmsData.numCells * 3 + 11];
    bmsData.TotalVoltage*=0.01;
    //pridobivanje toka
    uint16_t currraw = (data[13 + bmsData.numCells * 3 + 13] << 8) | data[13 +
bmsData.numCells * 3 + 14];
    //pretvorba toka
    if ((currraw & 0x8000) == 0x8000) {
        bmsData.Current = (float)(currraw & 0x7FFF) * 0.01;
    } else { bmsData.Current = (float)(currraw & 0x7FFF) * -1 * 0.01; }
    bmsData.SOC= data[13 + bmsData.numCells * 3 + 16];
    return true;
} else { WebSerial("Error: Incomplete battery data received."); }
} else { WebSerial("Error: Invalid data received."); }
return false;
}
}

```

Izsek kode 4: Del programa za komunikacijo z BMS.

4.2.2 Ocena napolnjenosti baterije

Ocena napolnjenosti baterije samo na podlagi njene napetosti (še posebej pri LiFePO4 baterijah) ni natančna. Napetost takih baterij je zelo konstantna skoraj do konca praznenja ali polnenja, po tem pa začne hitro padati ali naraščati. Zato napolnjenost baterije ocenjujem s pomočjo podatkov o energiji z senzorja INA228, postavljenega takoj za baterijo (ang. battery fuel gauge).

```

rawenergy = inaBat.getEnergy()/3600;
dif = rawenergy - prevrawenergyBat;
if(dif > 0 && dif < 1){
    if(inaDataBat.curr>0){
        inaDataBat.energy += dif;
        batEnergyAdj += dif;
    }else{
        inaDataBat.energy -= dif;
        batEnergyAdj -= dif;
    }
    prevrawenergyBat = rawenergy;
}

```

Izsek kode 5: Pridobivanje podatkov o energijski bilanci.

Zgornji kode vsakih 10 ms pridobi podatke o energiji in jih shranjuje v spremenljivko batEnergyAdj. Ob praznjenju baterije se bo njena vrednost zmanjševala, ob polnjenju pa povečevala. Tako lahko spremljamo energijo v bateriji. Vsakič, ko se baterija

popolnoma napolni, se ta spremenljivka ponastavi na nek maksimum. V mojem primeru je to 1024, ker ima baterija 1024 Wh.

```
float fmap(float x, float in_min, float in_max, float out_min, float out_max){
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
socBat = fmap(batEnergyAdj, 0, settings.bat_cap, 0, 100);
```

Izsek kode 6: Izračun SOC

Stanje napolnenosti baterije (ang. state of charge (SOC)) izračunamo tako, da obseg vrednosti od 0 do maksimuma (1024) pretvorimo v obseg od 0 do 100.

4.2.3 Algoritem polnenja

LiFePO4 baterije je potrebno polniti s konstantnim tokom in konstantno napetostjo. Baterija je polna takrat, ko tok v območju konstantne napetosti pade pod neko majhno vrednost, v mojem primeru 100 mA.

Večina ljudi polni LiFePO4 baterije do 3,65 V na celico ali maksimalno napetost polnenja, ki je zapisana v podatkovnem listu celice. Vendar vsakdanje polnenje do 3,65V na celico lahko škodi bateriji skozi leta uporabe. Zato sem se odločil polniti moje celice do 3,45 V saj s tem skoraj ne izgubljam na kapaciteti, hkrati pa pridobim na življenski dobi baterije.

BMS sem nastavil tako, da izravnava napetosti celic baterije samo nad napetostjo 3,4 V, ker je pri LiFePO4 baterijah izravnanje pred tem nepotrebno in neželjeno.

Po napolnenosti baterije moj algoritem spusti napetost celic na okoli 3,34 V, saj je to njena mirovna napetost (ang. float voltage). Na tej napetosti jo drži, vse dokler je dovolj sončne energije oziroma dokler napetost in s tem napolnjenost ne padeta pod določen nivo.

Čeprav je napetost celic manjša za 0,2 V, je SOC baterije še vedno okoli 99,7 %. Tako ne zgubimo bistveno na kapaciteti, pridobimo pa na življenski dobi baterije, saj manj časa preživi na visokih napetostih.

```
if(BMS.bmsData.TotalVoltage > settings.chrg_volt - 0.05 && avg_full_curr <
settings.chrg_ctf_curr / 1000.0 && avg_full_curr > 0 && (millis()-eeprom_full_millis >=
eeprom_full_update)){
    batEnergyAdj = settings.bat_cap;
    DataEnergy.whIn = inaDataIn.energy;
    DataEnergy.whOut = inaDataOut.energy;
    DataEnergy.whBat = inaDataBat.energy;
    DataEnergy.whBatAdj = batEnergyAdj;
    vout_setpoint = round(settings.float_volt * 100.0);
```

```

EEPROM.put(0, DataEnergy);
EEPROM.commit();

    eeprom_full_millis = millis();
}
if((batEnergyAdj < settings.float_exit_soc * 10.0 || BMS.bmsData.TotalVoltage <
settings.float_exit_volt) && (vout_setpoint == round(settings.float_volt * 100.0))){
vout_setpoint = round(settings.chrg_volt * 100.0);
}

```

Izsek kode 7: Koda za preklon med režimom polnenja in režimom ohranjevanja življenske dobe baterije.

Zgornja koda v prvem pogoju se izvrši, ko je baterija polna in njen tok polnenja pade pod 100 mA. Pri tem se ponastavi batEnergyAdj na maksimalno energijo baterije, napetost polnenja se zmanjša in vrednosti se shranijo v dolgotrajni pomnilnik v primeru ponovnega zagona sistema.

4.3 MPPT ALGORITEM

Vsak fotovoltaični panel ima svoj MPP (max power point) napetosti, pri kateri je mogoče iz njega pridobiti najvišjo moč. Težava je, da se ta napetost spreminja zaradi veliko dejavnikov kot so temperatura, intenzivnost svetlobe itd.

Da bi MPPT solarni regulator deloval najučinkoviteje, je točki MPP potrebno nenehno slediti. Za sledenje spreminjam najnižjo dovoljeno napetost na vhodu DC-DC pretvornika (ZK-SJ20), kar je tudi napetost na solarnih panelih.

Ko se sistem zažene, začne iskati MPP tako, da iz najvišje napetosti preide na najnižjo napetost, ki je nastavljiva, in si zapomne točko v kateri je bila moč največja:

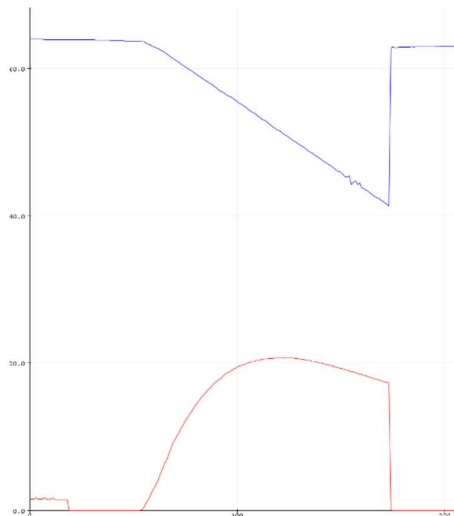
```

if(vin_setpoint <= round(settings.mppt_min_volt * 100)){
    vin_setpoint=mpptv;
    start=1;
}
if(inaDataIn.pow > maxp){
    maxp=inaDataIn.pow;
    mpptv=inaDataIn.volt*100.0;
}
vin_setpoint-=10;

```

Izsek kode 8: Koda za iskanje MPP.

Ko pride do najnižje napatosti si v spremenljivko mpptv shrani napetost pri kateri se je to zgodilo.



Slika 24: Vhodna napetost in vhodna moč

Zgornja slika prikazuje podatke iz enega od praktičnih testov. Modra linija prikazuje vhodno napetost, rdeča pa vhodno moč. Na sliki viden MPP v takratnjih pogojih. Slika je namenjena prikazu oblike krivulje in ne dobesednih vrednosti moči in napetosti, zato enote tu niso pomembne.

Po začetni proceduri sistem preide v drug režim delovanja. Tu preverja moč pri napetosti, ki je za 50 mV višja od trenutne, nato še pri napetosti, ki je za 50 mV nižja. Nato ugotovi kdaj je moč višja in nastavi PID krmilniku nove parametre:

```
switch(stateMppt){
    case 0:
        vin_setpoint+=5;
        stateMppt=1;
        break;
    case 1:
        powup=inaDataIn.pow;
        vin_setpoint-=10;
        stateMppt=2;
        break;
    case 2:
        powdown=inaDataIn.pow;
        vin_setpoint+=5;
        if(powdown>powup){
            vin_setpoint-=5;
        }else{
            vin_setpoint+=5;
        }
        stateMppt=0;
        break;
}
```

Izsek kode 9: Koda za sledenje MPP

S tem algoritmom dosežemo najvišjo točko moči in ji nenehno sledimo.

4.4 BELEŽENJE PODATKOV

Moj sistem beleži povprečne podatke senzorjev vsakih 15 minut in na koncu vsakega dneva. Podatke shranjujem na SD kartico v .csv datotekah.

4.4.1 Statistika

Povprečje moči, napetosti, temperatur itd. izračunavam v razredu Average:

```
class Average {
private:
    volatile double sum;
    volatile int count;

public:
    Average(){
        sum = 0;
        count = 0;
    }

    void add(float value){
        sum += value;
        count++;
    }

    float get(){
        if (count == 0){
            return 0;
        }
        return sum / count;
    }
    void reset() {
        sum = 0;
        count = 0;
    }
};
```

Izsek kode 10: Razred Average.

Razred ima metode za dodajanje, izračun in ponastavitev povprečja. Metodo `add()` kličem vsakič, ko dobim podatek iz senzorja, `get()` ko podatek zapišem in `reset()` vsakih 15 minut po zapisu podatkov.

4.4.2 Beleženje 15-minutnih podatkov

Za beleženje podatkov je zadolžen spodnji izsek programa:

```
char fileName[30];

struct tm adjTime;
adjTime = timeinfo;
adjTime.tm_min -= 15;
time_t normalizedTime = mktime(&adjTime);
localtime_r(&normalizedTime, &adjTime);

strftime(fileName, sizeof(fileName), "/data/%d-%m-%Y.csv", &adjTime);
// Ali obstaja
bool fileExists = SD.exists(fileName);
// Odpri datoteko
File file = SD.open(fileName, FILE_APPEND);
if (!file) {
    WebSerial("Failed to open file: %s\n", fileName);
    return;
}
```


4.4.3 Beleženje dnevnih podatkov

Rezultate posameznega dneva beležim s spodnjim delom programa:

```
// Naredi naslov datoteke
char fileName[30];

struct tm adjTime;
adjTime = timeinfo;
adjTime.tm_min -= 15;
time_t normalizedTime = mktime(&adjTime);
localtime_r(&normalizedTime, &adjTime);

strftime(fileName, sizeof(fileName), "/data/sum_%d-%m-%Y.csv", &adjTime);
// Odpri datoteko v append načinu
File file = SD.open(fileName, FILE_APPEND);
if (!file) {
    WebSerial("Failed to open file: %s\n", fileName);
    return;
}

file.println("End_SOC,Energy_in,Energy_out,Energy_bat,Energy_load");

// Zapiši podatke
file.printf("%.2f,%.2f,%.2f,%.2f,%.2f\n", socBat, inaDataIn.energy, inaDataOut.energy, inaDataBat.energy, dataLoad.energy);

file.close();
```

Izsek kode 12: Beleženje rezultatov posameznega dneva.

Beleženje teh podatkov deluje na zelo podoben način. Koda se izvede vsak dan ob 0:00, kar pomeni da bo naslov datoteke imel datum prejšnjega dne. V to datoteko zapisujem samo podatke o energijah skozi cel dan, saj drugi podatki niso bistveni.

4.4.4 Izvšitev beleženja podatkov

Spodaj vidimo v kakšnih okoliščinah kličem funkciji `logData()` in `logDataDay()` (zgoraj razloženi funkciji), ki shranjujeta podatke na SD kartico.

```
if (getLocalTime(&timeinfo)) {
    char buffer[10];
    strftime(buffer, sizeof(buffer), "%H:%M", &timeinfo);
    set_var_timem(buffer);
    if (timeinfo.tm_min % 15 == 0 && prevMin != timeinfo.tm_min){
        inaDataOut.energyDiff = inaDataOut.energy - inaDataOut.energyLast;
        inaDataIn.energyDiff = inaDataIn.energy - inaDataIn.energyLast;
        inaDataBat.energyDiff = inaDataBat.energy - inaDataBat.energyLast;
        dataLoad.energyDiff = dataLoad.energy - dataLoad.energyLast;
        logData();
        avg_full_curr = inaDataBat.avgCurr.get();
        resAvgFlg = true;
        resAvgMisc = true;
        inaDataOut.energyLast = inaDataOut.energy;
        inaDataIn.energyLast = inaDataIn.energy;
        inaDataBat.energyLast = inaDataBat.energy;
        dataLoad.energyLast = dataLoad.energy;

        prevMin = timeinfo.tm_min;
        ws.broadcastTXT("15minUpd");
    }
    if (timeinfo.tm_hour == 0 && timeinfo.tm_min == 0 && !dayReset) {
        logDataDay();
    }
}
```

```

DataEnergy.whBatAdj = batEnergyAdj;
inaDataIn.energy = 0;
inaDataOut.energy = 0;
inaDataBat.energy = 0;
dataLoad.energy = 0;

inaDataOut.energyLast = inaDataOut.energy;
inaDataIn.energyLast = inaDataIn.energy;
inaDataBat.energyLast = inaDataBat.energy;
dataLoad.energyLast = dataLoad.energy;

DataEnergy.whIn = inaDataIn.energy;
DataEnergy.whOut = inaDataOut.energy;
DataEnergy.whBat = inaDataBat.energy;

EEPROM.put(0, DataEnergy);
EEPROM.commit();
inaIn.setAccumulation(1);
inaOut.setAccumulation(1);
inaBat.setAccumulation(1);
prevrawenergyIn = 0;
prevrawenergyOut = 0;
prevrawenergyBat = 0;

    dayReset = true;
}

if (timeinfo.tm_hour != 0) {
    dayReset = false;
}
} else {
    WebSerial("Failed to obtain time.");
}
}

```

Izsek kode 13: Klic funkcij za shranjevanje podatkov.

Tu pridobim uro preko NTP protokola, potem preverim, ali je ura 00, 15, 30, 45 in da prejšna minuta ni enaka. Koda v pogoju se bo izvedla enkrat vsakih 15 min. V pogoju izračunam razliko energij, podatki pa zapisujem v 15 minutnih intervalih in ponastavim spremenljivke za naslednjo beleženje.

Drugi if stavek deluje podobno, vendar se izvede vsako polnoč. Po zapisu podatkov podatke o energiji nastavim na 0 in ponastavim registre za energijo v vezjih.

4.5 UPORABNIŠKI VMESNIK

Sistem ima dva različna uporabniška vmesnika, katerih namen je uporabniku prikazati stanje in različne podatke o sistemu.

V nadaljevanju bom predstavil samo proces delovanja in elemente uporabniških vmesnikov (UV).

4.5.1 Spletni uporabniški vmesnik

Namen spletnega vmesnika je nadzorovanje trenutnega stanja naprave, vizualizacija beleženih podatkov in izvrševanje ukazov za napredne uporabnike.

4.5.1.1 Strežniška stran

V mojem sistemu ESP32 deluje tudi kot strežnik za spletno stran, ki je ob emem uporabniški vmesnik. Spletna stran je dostopna preko internetnega naslova sistema. Da ta funkcija deluje, mora biti sistem povezan na WiFi omrežje, v katerem bo spletna stran dostopna. Strežniške datoteke javascript, html in css so zaradi svoje velikosti shranjene na SD kartici.

Na strežniku sem napisal preprost programski vmesnik (ang. application programming interface oz. API), ki odjemalcu preko metode HTTP GET pošlje podatke senzorjev.

Koda na odjemalčevi strani do podatkov dostopa preko dveh končnih točk (ang. endpoint), kot vidimo spodaj:

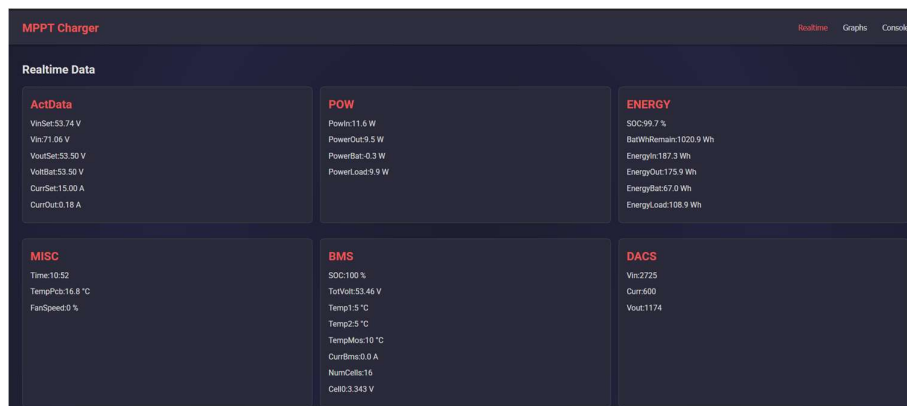
[/data/day?date=20-02-2025](#) za dostop do 15-minutnih podatkov za določen dan,
[/data/month?date=02-2025](#) za dostop do podatkov za cel mesec.

Za pošiljanje trenutnih podatkov uporabljam protokol websocket. Odjemalcu pošiljam trenutne podatke vsakih 500 ms. Podatki so v tem intervalu tudi povprečeni z istim razredom, ki sem ga opisal v poglavju o statistiki. Prav tako lahko strežnik sprejema sporočila od odjemalca in nanja tudi odgovarja.

4.5.1.1 Odjemalčeva stran

Na odjemalčevi strani so podatki iz strežnika pridobljeni s pomočjo programa napisanega v javascriptu. Ta podatke obdela in jih vizualizira. Odjemalec ima tudi možnost pošiljanja različnih ukazov, ki so bili primarno namenjeni razvoju, a so se izkazali za uporabne in sem jih pustil tudi v končni različici.

V nadaljevanju bom predstavil spletni UV.



Slika 25: začetna stran spletnega UV

Na prejšnji sliki vidimo trenutne podatke o sistemu, ki se osvežujejo na vsakih 500 ms. Tu spremljamo podatke o napetostih, močeh, energijh, bateriji in DAC.

Do drugih strani je možno priti preko navigacijskih gumbov desno zgoraj.



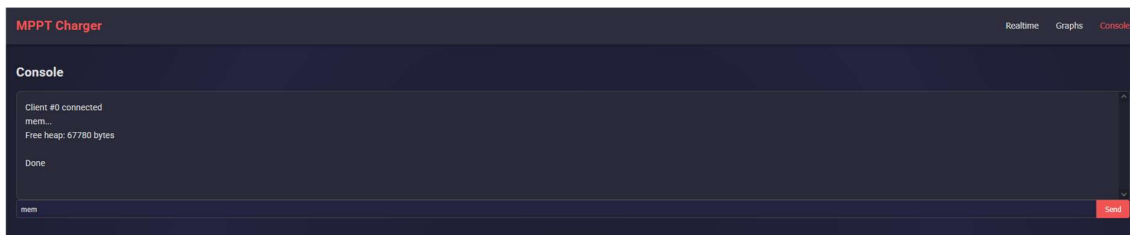
Slika 26: Dnevna vizualizacija podatkov

Zgornja slika prikazuje uporabniški vmesnik za vizualizacijo dnevnih podatkov. Uporabnik lahko spreminja datum, število dni in vrsto podatkov, ki bodo prikazani na grafu. Izbira lahko tudi med energijo, močjo in drugimi podatki, pri katerih je vedno na desni skali vidna napolnjenost hranilnika.



Slika 27: Mesečna vizualizacija podatkov

Zgornja slika prikazuje podatke o energijah in SOC za celoten mesec po dnevih.



Slika 28: Ukazna vrstica za komunikacijo z napravo

Na tej sliki vidimo vmesnik, kjer lahko napredni uporabniki komunicirajo z napravo s pomočjo v naprej določenih ukazov. Uporabi lahko ukaze kot so npr.: mem (izpiše količino še ne zapolnjenega RAM-a), ls (izpiše vse datoteke na sistemu), rm (izbriše določeno datoteko), restart (ponovno zažene napravo), reset (ponastavi vse meritve energij), full (nastavi soc na željeno vrednost).

4.5.1 Vmesnik na vgrajenem zaslonu

Ta vmesnik je namenjen prikazu podatkov in stanja sistema ter pomembnim nastavitvam. Uporabniški vmesnik uporablja knjižnice lvgl.h, TFT_eSPI.h in XPT2046_Bitbang.h. Sestavil sem ga v programu EEZ Studio.

V nadaljevanju bom predstavil njegovo obliko, podatki na slikah pa so simbolični.

4.5.1.1 Začetna stran

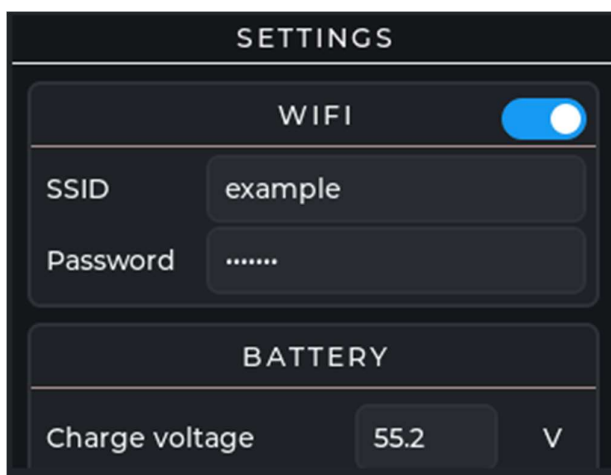


Slika 29: Privzeta stran

Na zgornji sliki je prikazana privzeta stran vmesnika. Spremljamo lahko stanje WiFi povezave, uro, napolnjenost baterije, stanje polnenja (Idle, MPPT, Const Voltage, Const Current, Float), stanje sistema, temperaturo močnostnega dela, hitrost

ventilatorja, obremenjenost sistema in čas delovanja. Ima tudi gumba za ponovni zagon ter zasilno ustavitev.

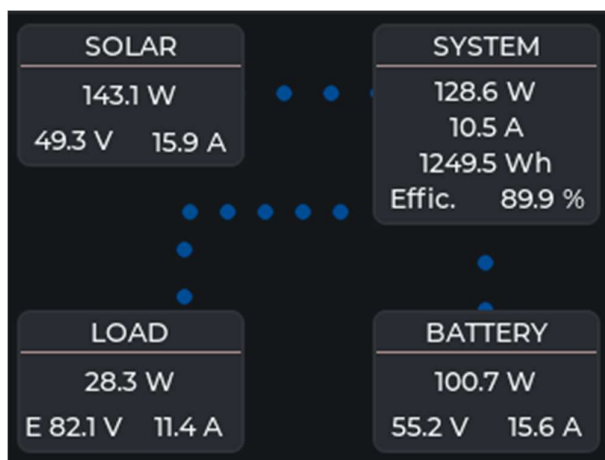
4.5.1.2 Nastavitve



Slika 30: Nastavitve sistema

Slika prikazuje stran za nastavitve sistema, do katere lahko pridemo s premikom prsta v levo na zaslonu. Da bo sistem pravilno deloval mora uporabnik nastaviti vse parametre na tej strani. Podatke se shrani s pritiskom na gumb »Save and Restart«.

4.5.1.3 Trenutni podatki



Slika 31: Trenutni podatki sistema

Zgornja slika prikazuje stran za vizualizacijo trenutnega stanja in ogled trenutnih podatkov sistema. Do nje lahko pridemo s potegom privzete strani navzgor. Vsak razdelek prikazuje podatke o delu sistema (njegovo moč, napetost in tok), le pri obremenitvi (LOAD) imamo napetost na polnilcu, pri sistemu (SYSTEM) pa vidimo tudi

pretvorjeno energijo dneva in učinkovitost. Modri krogi prikazujejo smer in velikost moči, ki teče skozi sistem.

S pritiskom na SYSTEM ali BATTERY, vidimo še več informacij o tem delu sistema (sliki 32 in 33).

SYSTEM INFO			
ENERGY		SETPOINTS	
Solar:	567.5 Wh	Vin:	50.15 V
Converted:	1249.5 Wh	Vout:	55.26 V
Battery:	490.3 Wh	Curr:	9.55 A
Load:	120.6 Wh		
MISC		DACS	
IP add:	192.168.1.24	Vin:	4326
	AA:BB:CC:DD:EE:FF	Vout:	8732
SD free:	4013 MB	Curr:	2678
Fw date:	Feb 25 2025		

Slika 32: Informacije o sistemu

BATTERY INFO		SOC 95.5 %	
Cells dev:	14 mV	Cells avg:	3.325 V
Temp 1:	6.2 °C	Temp 2:	7.5 °C
Temp MOS:	17.3 °C	Voltage:	53.4 V
Ene rem:	1006.2 Wh	Current:	12.4 A
01:	3.456 V	02:	3.456 V
04:	3.372 V	05:	3.372 V
07:	3.123 V	08:	3.123 V
10:	3.123 V	11:	3.123 V
13:	3.123 V	14:	3.123 V
16:	3.123 V	15:	3.123 V

Slika 33: Informacije o bateriji

5. MERITVE

V tem poglavju bom predstavil, kako se sistem obnaša v različnih pogojih.

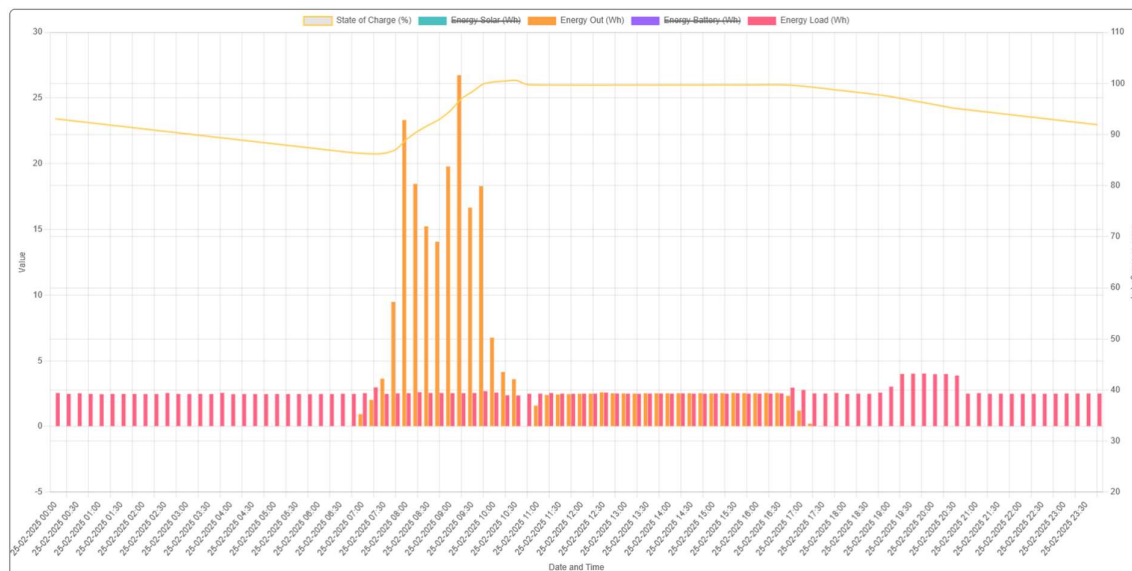
Grafi prikazujejo podatke, pridobljene s sistema med dejanskim obratovanjem in niso simulirani.

Ker je količina zajetih podatkov kar velika, sem se odločil predstaviti samo določene podatke in primere.

Vsi podatki so iz obdobja med januarjem in marcem 2025.

5.1 MIROVANJE

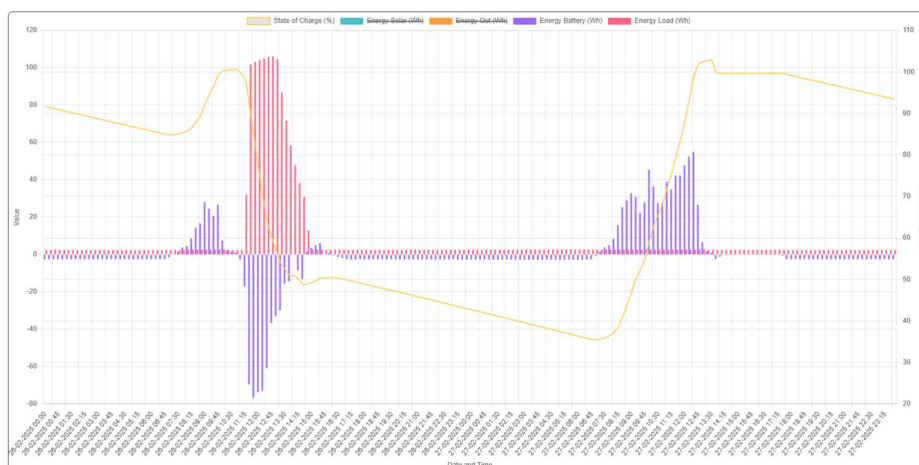
Kar nekaj dni v tednu je sistem v mirovanju, ker mi baterije električnega skuterja ni potrebno polnit vsak dan. Ob takih dneh je edini porabnik majhen računalnik, ki ga uporabljam kot strežnik za igre in še za nekatere druge storitve. Njegova konstantna moč je okoli 10 W, najvišja pa okoli 20 W.



Slika 34: Graf energije, ko je sistem v mirovanju

Graf na zgornji sliki prikazuje energijo skozi dan, ko ni večjih porabnikov. Energy Out prikazuje energijo, ki zapusti MPPT vezje (ZK-SJ20), Energy Load pa energijo, ki jo porabnik porabi b intervalu 15-minut. Z grafa je razvidno polnjenje baterije od 86 % do 100 %. Ko se baterija popolnoma je na grafu razvidna majhna stopnica, ki se zgodi ob ponastavitvi SOC na 100 %. S tem sem zagotovil, da bo SOC kazal pravilno, ker sem na tej točki prepričan, da je baterija popolnoma napolnjena. Po tem gre sistem v stanje napolnjenosti (float), pri katerem spusti napetost baterije. V tem režimu vidimo, da se SOC ne zmanjšuje, kar pomeni da ne izgubljam na kapaciteti. Energija porabnika je skozi cel dan konstantna z izjemo med urami 19:00 in 21:00. V tem obdobju je bil nekdo priključen na igralni strežnik zato je ta porabljal nekoliko več energije.

5.2 MOČNO PRAZNIENJE IN POLNENJE

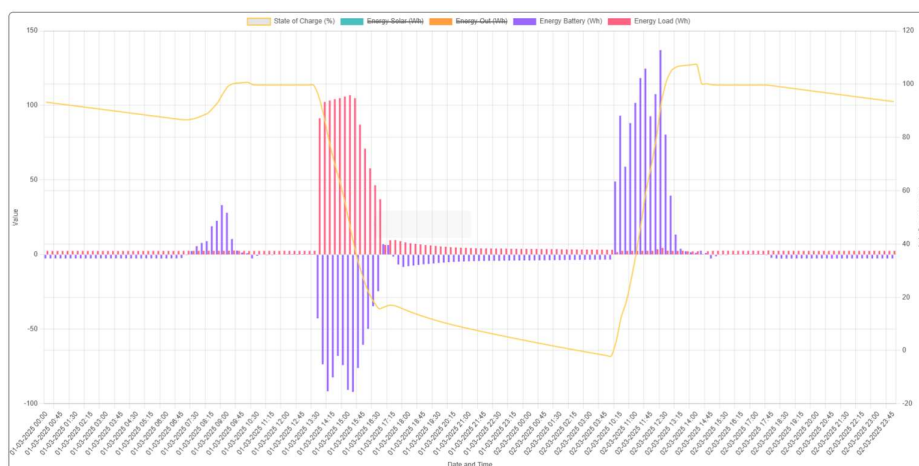


Slika 35: Graf energije ob močnem praznanju in polnjenju

Na grafu zgoraj je razvidno močno praznjenje in polnjenje hranilnika. Na levi strani grafa je razvidna priključitev velikega porabnika, v tem primeru sem dal polniti električni skuter in moč nastavil na 400 W. Energija začne teči iz hranilnika v polnilca in v baterijo polnilca. Energija hranilnika je vidna v modri barvi pod osjo grafa. Razlog, da je količina celotne energije iz baterije manjša, kot jo je porabnik porabil je, da je energija sonca istočasno polnila hranilnik in pomagala med praznjenjem.

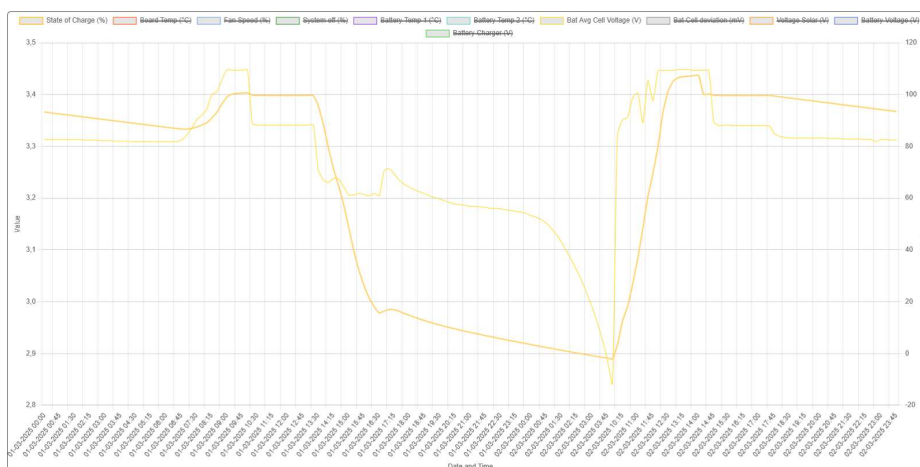
Naslednji dan vidimo, da se je hranilnik popolnoma napolnil. Stopnica SOC na grafu pri polnjenju je tu okoli 2,7 %. Razloga, zakaj se je ta odstotek tako velik, še nisem ugotovil. Vendar pa se sistem ob vsakem polnjenju kalibrira in to ni problem.

5.3 TEST PREKOMERNEGA PRAZNIENJA



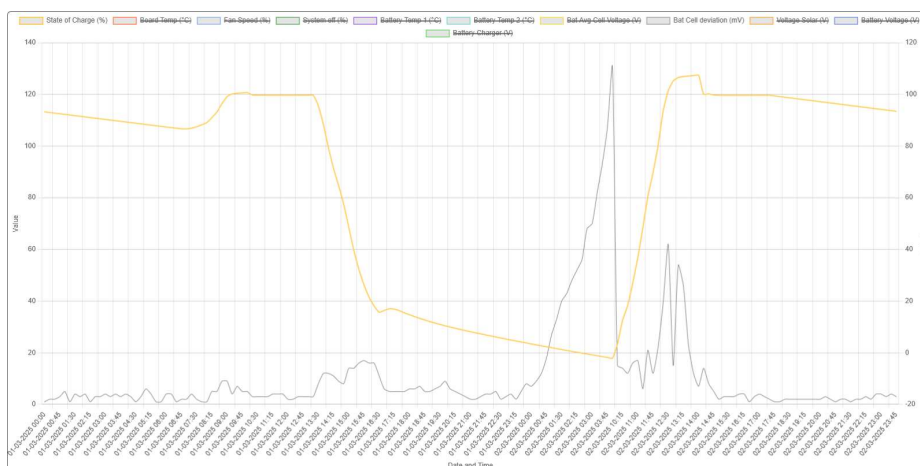
Slika 36: Graf popolne izpraznitve hranilnika

Na zgornjem grafu vidimo primer, ko hranilnik ni imel dovolj energije, ki jo je porabnik želel. Ko SOC pride okoli 0 %, se sistem ugasne, ker BMS zavaruje hranilnik pred prenizko napetostjo (ang. under voltage protection (UVP)). To se zgodi, ko napetost ene od celic hranilnika pride pod 2,8 V. Takrat BMS izklopi sistem.



Slika 37: Povprečna napetost celic hranilnika ob popolni izpraznitvi

Na zgornji sliki vidimo isto situacijo vendar je zdaj graf prikazuje SOC in povprečno napetost posamezne celice v hranilniku. Tukaj se dobro vidi tipično krivuljo praznenja LiFePO4 baterij. Napetost je zelo stabilna pri SOC večjih od okoli 5 %, po tem pa začne hitro in agresivno padati, kljub temu da je takrat moč iz baterije samo okoli 12 W.

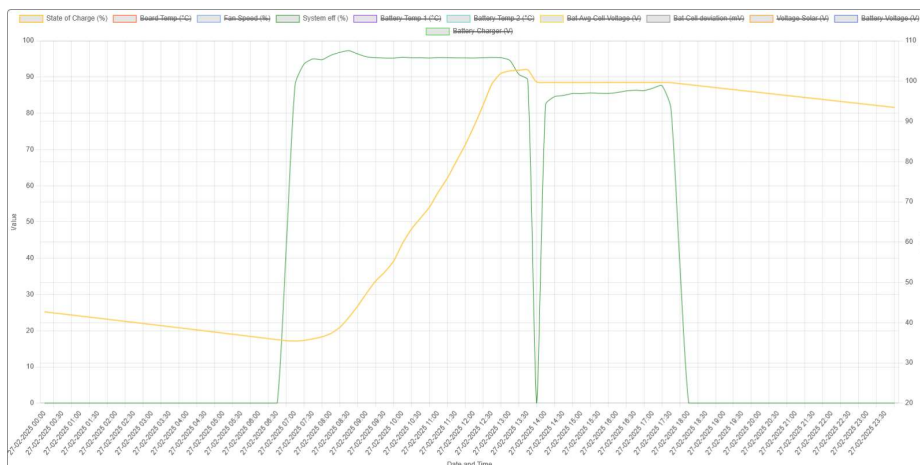


Slika 38: Razlika med napetostjo celic

Na zgornji sliki lahko vidimo, kako se razlika med napetostjo celic hranilnika poveča, ko je njihova napetost manjša od 3 V.

5.4 UČINKOVITOST MOČNOSTNEGA DELA SISTEMA

Učinkovitost močnostnega dela sistema (modula ZK-SJ20), je razmerje med vhodno in izhodno močjo modula.

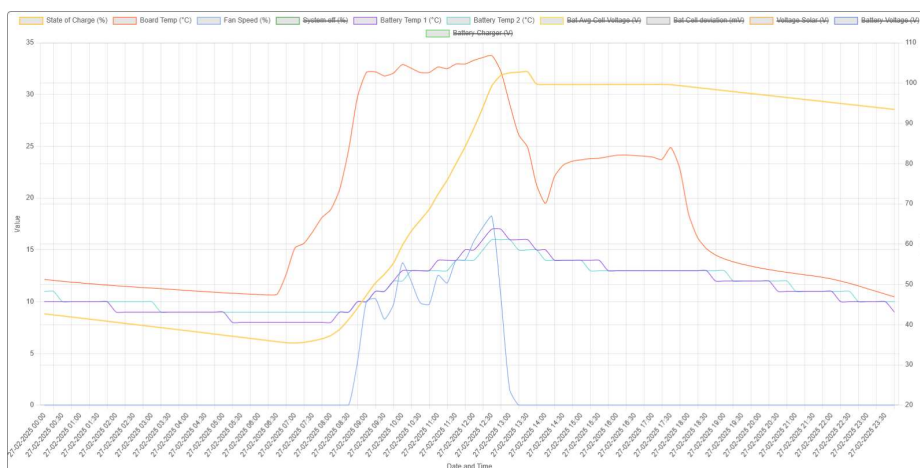


Slika 39: Učinkovitost močnostnega dela sistema, pri polnjenju hranilnika z 400 W moči

Na zgornji sliki je z zeleno barvo vidna omenjena učinkovitost. Vidimo, da skozi celoten proces polnjenja giblje okoli 95 %. Ko pa gre sistem v stanje napolnjenosti (float) način, učinkovitost pade na okoli 85 %, saj je takrat moč samo okoli 11 W.

Zaradi visoke učinkovitosti ima močnostni del pri moči 400 W izgube samo 20 W, kar za ventilator in hladilno rebro ne predstavlja noben problem.

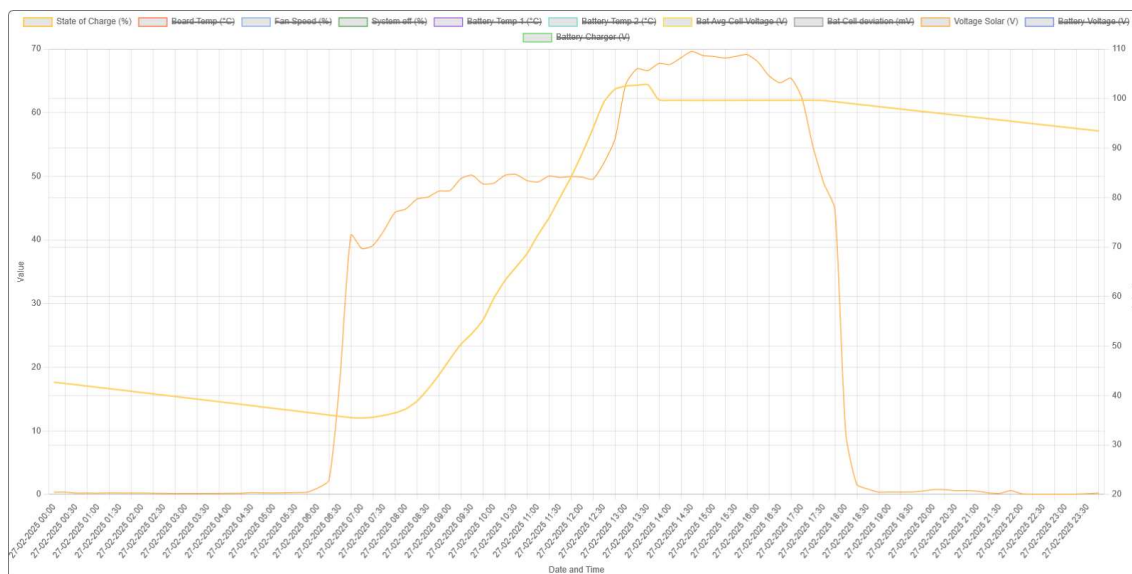
5.5 TEMPERATURE



Slika 40: Meritev temperatur ob polnjenju hranilnika

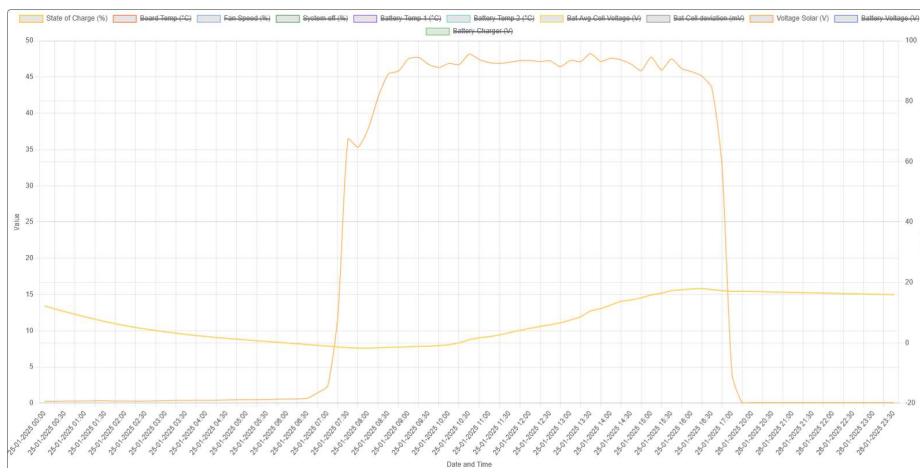
Zgornji graf prikazuje temperaturi hranilnika, močnostnega vezja in hitrost ventilatorja. Ventilator deluje tako, da večja kot je temperatura na vezju, višja bo hitrost ventilatorja. Iz grafa se lepo vidi, kako temperatura narašča počasneje kot narašča hitrost ventilatorja.

5.6 MPPT ALGORITEM



Slika 41: Vhodna napetost ob polnjenju hranilnika

Zgornji graf prikazuje vhodno napetost v razponu enega dneva. Iz grafa vidimo, da je pri nižjih močeh sonca (med uro 6:30 in 9:00) tudi napetost na panelah manjša. Po tem je napetost konstantna okoli 49 V, kar je zelo blizu MPP napetosti, zapisane na podatkovnem listu panel. Ob koncu polnjenja hranilnika, napetost polja panelov naraste, saj začne polnilni tok baterije padati zaradi režima konstantne napetosti.



Slika 42: Vhodna napetost ob polnjenju hranilnika ob oblačnem dnevu

Na zgornji sliki spet vidimo vhodno napetost, vendar ta dan ni bilo dovolj sonca, da bi hranilnik popolnoma napolnilo. Zaradi tega je vhodna napetost malenkost nižja in konstantna skozi cel dan, ker je sistem ves dan v MPPT načinu.

5.7 MESEČNA ENERGIJA



Slika 43: Energije po dnevih za februar 2025

Zgornji diagram prikazuje vse pretoke energij v sistemu po dnevih za mesec februar. Iz grafa lahko vidimo vse dni, ko sem polnil električni skuter.

6. ZAKLJUČEK

Pri raziskovanju in izdelavi svojega PV sistema, sem se veliko naučil o elektrotehniko in računalništvu. S predelavo modula ZK-SJ20 sem spoznal, kako pomembno je branje podatkovnih listov vezji za razumevanje delovanja modula. Velikokrat sem moral reševati nepričakovane težave. Pri reševanju problemov, mi je veliko pomagalo predznanje pridobljeno v šoli in znanje iz drugih projektov, ki jih izvajam v prostem času.

Izdelal sem PV sistem, ki je učinkovit in ima vse funkcije, ki sem hotel. Že več kot pol leta polni moj električni skuter in mi s tem omogoča neodvisnost od električnega omrežja. V prihodnosti imam namen povečati hranilnik, da bi lahko napolnil baterijo skuterja v celoti.

6.1 POTRJEENE HIPOTEZE

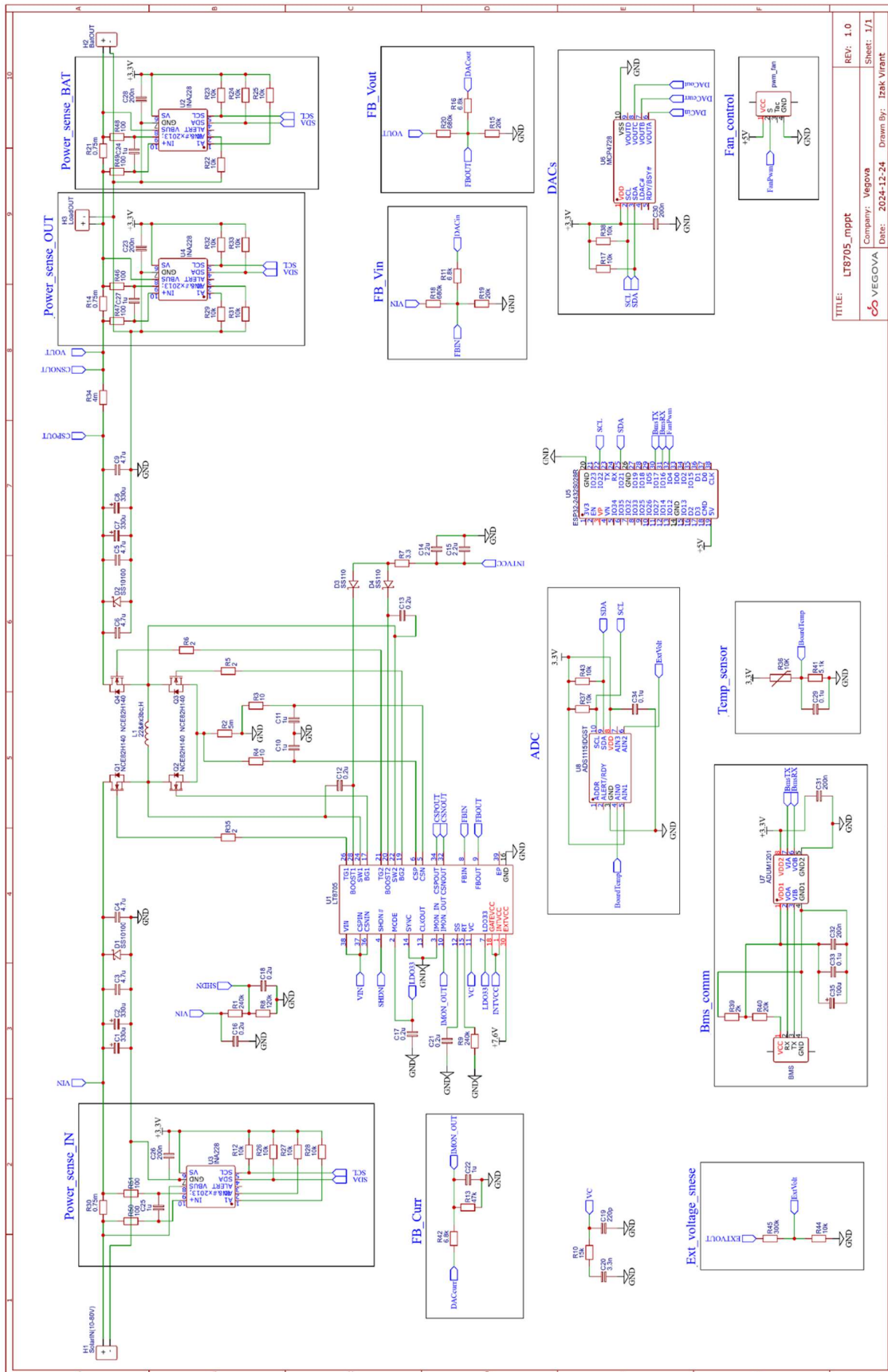
V raziskovalni nalogi sem potrdil vse hipoteze:

- iz široko dostopnega DC-DC pretvornika (ZK-SJ20) sem izdelal učinkovit MPPT solarni polnilnik, ki že nekaj mesecev brez težav polni hranilnik,
- z vezjem INA228 sistem dobro predvide raven napnjenosti baterije in se kalibrira ob vsaki popolni napolnjenosti baterije,
- modul ZK-SJ20 uspešno krmilim s pomočjo DAC in ga uporabljam z algoritmom MPPT, da sledim MPP solarnih panelov,
- strošek nakupov vseh komponent baterije in doma izdelanega ohišja, je primerljiv ali nižji od stroška sestavljenega sistema na trgu. Obenem se se izognil tveganju nakupa nekvalitetnih ali netestiranih komponent, za nameček pa vem tudi da sem sistem dobro sestavil.

7. VIRI IN LITERATURA

- [1] Ghislain REMY, Olivier BETHOUX, Claude MARCHAND, Hussein DOGAN, **Review of MPPT Techniques for Photovoltaic Systems**, November 2009, dostopno na: https://www.researchgate.net/publication/280849388_Review_of_MPPT_Techniques_for_Photovoltaic_Systems (6.3.2025)
- [2] Linear Technology Corporation, **80V VIN and VOUT Synchronous 4-Switch Buck-Boost DC/DC Controller**, dostopno na: <https://www.analog.com/media/en/technical-documentation/data-sheets/8705ff.pdf> (6.3.2025)
- [3] Analog Devices, Inc., **Dual-Channel Digital Isolators ADuM1200/ADuM1201**, dostopno na: https://www.analog.com/media/en/technical-documentation/data-sheets/adum1200_1201.pdf (6.3.2025)
- [4] Texas Instruments, **INA228 85-V, 20-Bit, Ultra-Precise Power/Energy/Charge Monitor With I2C Interface**, dostopno na: <https://www.ti.com/lit/ds/symlink/ina228.pdf> (6.3.2025)
- [5] **Communication protocol between monitoring platform and BMS**, dostopno na: <http://www.jk-bms.com/Upload/2022-05-19/1621104621.pdf> (6.3.2025)
- [6] **Intelligent protection board for lithium battery Operation and maintenance instructions**, dostopno na: <https://de.gobelpower.com/download/JK-Smart-BMS-with-Active-Balancer-Instruction-EN.pdf> (6.3.2025)
- [7] EVE Power CO., LTD, **IRF40135**, dostopno na: https://www.evebatteryusa.com/files/ugd/b4afc3_5f83846234374b7fa995bb3a913d3447.pdf (6.3.2025)
- [8] AEET Energy Group, **Solar module AEET-SE235M-33/D — AEE-SE265M-33/D**, dostopno na: https://reenergyhub.com/files/hersteller/AEET-Energy/pdf/AEET_SE_232-265M-33_D_EN.pdf (6.3.2025)
- [9] Microchip Technology Inc., **12-Bit, Quad Digital-to-Analog Converter with EEPROM Memory**, dostopno na: <http://ww1.microchip.com/downloads/en/devicedoc/22187e.pdf> (6.3.2025)
- [10] Texas Instruments, **ADS111x Ultra-Small, Low-Power, I2CCompatible, 860SPS, 16-Bit ADCs with Internal Reference, Oscillator, and Programmable Comparator**, dostopno na: https://www.ti.com/lit/ds/symlink/ads1114.pdf?ts=1741073808700&ref_url=https%253A%252F%252Fwww.google.com%252F (6.3.2025)
- [11] **EEZ Studio documentation**, dostopno na: <https://www.envox.eu/eez-studio-docs/> (6.3.2025)

PRILOGA 1 – ELEKTRIČNA SCHEMA



TITLE: LT8705_mppt
 Company: Vegova
 Date: 2024-12-24
 Drawn By: Izak Vrant
 REV: 1.0
 Sheet: 1/1