DIGITALNI SUSTAV UPRAVLJANJA ELEKTROMOTORNIM POGONOM S PROCESOROM AM2634

Lesjak, Robert

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

DIGITALNI SUSTAV UPRAVLJANJA ELEKTROMOTORNIM POGONOM S PROCESOROM AM2634

Rijeka, srpanj 2023.

Robert Lesjak

0069084044

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij elektrotehnike

Diplomski rad

DIGITALNI SUSTAV UPRAVLJANJA ELEKTROMOTORNIM POGONOM S PROCESOROM AM2634

Mentor: Prof. dr. sc. Neven Bulić

Komentor: Dr. sc. Dominik Cikač

Rijeka, srpanj 2023.

Robert Lesjak
0069084044

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 5. travnja 2023.

Zavod: **Zavod za automatiku i elektroniku**
Predmet: **Sustavi digitalnog upravljanja**
Grana: **2.03.02 elektrostrojarstvo**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Robert Lesjak (0069084044)**
Studij: Sveučilišni diplomski studij elektrotehnike
Modul: Automatika

Zadatak: **Digitalni sustav upravljanja elektromotornim pogonom s procesorom AM2634 / Digital control system for electric motor drive with AM2634 processor**

Opis zadatka:

Potrebno je na laboratorijskom modelu izvesti laboratorijski sustav upravljnaja elektromotornim pogonom s algoritmom implementiranim u procesorski modul s procesorom AM2634. U radu je potrebno opisati elemente laboratorijskog postava i prezentirati eksperimantalne rezultate.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 20. ožujka 2023.

Mentor:

Prof. dr. sc. Neven Bulić

Dr. sc. Dominik Cikač (komentor)

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Dubravko Franković

IZJAVA

Izjavljujem da sam ovaj diplomski rad izradio potpuno samostalno, uvažavajući načela akademske čestitosti.

Rijeka, 12.7.2023.

Robert Lesjak

ZAHVALA

Zahvaljujem se mentoru prof. dr. sc. Nevenu Buliću na pruženoj prilici za izradu diplomskog rada na temu koja me iznimno zanima. Također se zahvaljujem mentoru i asistentima koji su mi pružili povjerenje i omogućili korištenje potrebne laboratorijske opreme za istraživanje i izradu diplomskog rada.

Sadržaj

1 UVOD

Motivacija ovoga rada istražiti je mogućnost implementacije upravljanja elektromotornim pogonom u laboratorijskom okruženju pomoću nove generacije procesora tvrtke Texas Instruments AM263X. Uz procesor tvrtka nudi razvojno okruženje koje se sastoji od uređivača programskog koda "Code Composer Studio" i generatora konfiguracije mikrokontrolera "SysConfig". Cilj je rada izraditi laboratorijski sklop mikrokontrolera i energetske elektronike te implementaciju sustava vektorskog upravljanja asinkronog elektromotora uz pristupačan način interakcije sa samim sustavom kako bi se nad sustavom upravljanja mogla izvoditi istraživanja i edukacija. Iz tog razloga na procesoru je implementiran web server koji omogućuje komunikaciju sa web sučeljem kojim se onda može u realnome vremenu upravljati sustavnom i vršiti vizualizaciju mjerenja.

Digitalni sustavi upravljanja elektromotornim pogonima postaju sve relevantniji u svijetu gdje se teži ka povećanju učinkovitosti i smanjenju potrošnje energije jer omogućuju implementaciju raznih algoritama upravljanja. Algoritmi upravljanja mogu optimizirati razne karakteristike ovisno o potrebi na mjestu primjene. Na primjer, ovisno o algoritmu generiranja PWM signala može se optimizirati iskoristivi napon napajanja ili sklopni gubitci. Također se pomoću digitalnih sustava upravljanja može algoritam upravljanja adaptirati trenutnoj situaciji u stvarnom vremenu.

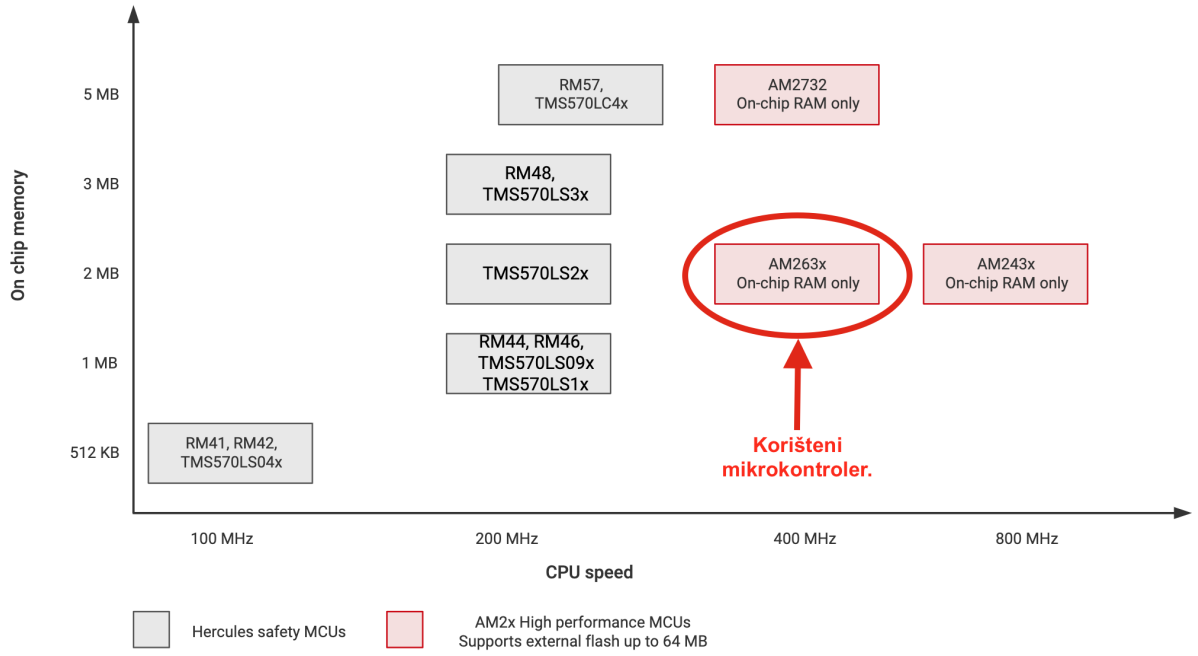
U radu će biti opisan procesor AM2634, jedan od najnovijih proizvoda Texas Instruments-a, koji je dizajniran da pruži pouzdano i robusno rješenje za različite aplikacije uključujući upravljanje elektromotornim pogonima. S obzirom na njegove napredne mogućnosti i fleksibilnost, ovaj procesor nudi značajan potencijal za poboljšanje upravljanja elektromotornim pogonima. Također za upravljanje vrlo je važna i energetska elektronika, za izradu rada također je korišten proizvod tvrtke Texas Instruments, HVMotorCtrl+PFCKit-R5. To je sklop energetske elektronike izrađen upravo za upravljanje elektromotornim pogonom, no za seriju procesora signala "C2000". Iz tog razloga bilo je potrebno izraditi adapter za povezivanje s razvojnim mikroprocesorskim sustavom "AM2634 LaunchPad".

Kao ključni dio ovog rada, razvijeno je web sučelje za interakciju s digitalnim sustavom upravljanja elektromotornim pogonom. Web sučelje dizajnirano je s namjerom da bude intuitivno i jednostavno za korištenje omogućavajući korisnicima da lako pristupe različitim funkcijama sustava. Omogućuje da se u stvarnom vremenu prilagođavaju parametri elektromotornog pogona kontrolirajući aspekte poput brzine i smjera vrtnje. Radi mogućnosti detaljne analize izmjerenih podataka web sučelje mjerene veličine vizualizira grafovima te omogućuje izvoz podataka za analizu u bilo kojem drugom programskom alatu.

2 AM 2XX GENERACIJA PROCESORA I NJIHOVE KARAKTERISTIKE

U radu je korišten mikrokontroler iz AM2xx generacije procesora koju je razvila tvrtka Texas Instruments kao odgovor tržištu na masovnu elektrifikaciju, automatizaciju, pametne tvornice i industriju 4.0. Ova područja karakterizira distribuirana obrada podataka što zahtjeva visoku razinu povezanosti pa ovi mikroprocesorski sustavi imaju naglasak na snažnim procesorima i širokom paletom dostupnih komunikacijskih periferija. Za ovaj rad vrlo je značajna ethernet periferija koja je omogućila komunikaciju upravljačkim sustavom TCP/IP protokolom koji je srž svake web tehnologije. Također iz aspekta procesorske snage za ovaj rad ključno je to da se u mikrokontroleru nalazi više procesora što omogućuje odvajanje, upravljanje i komunikaciju u dva paralelna procesa. [1]

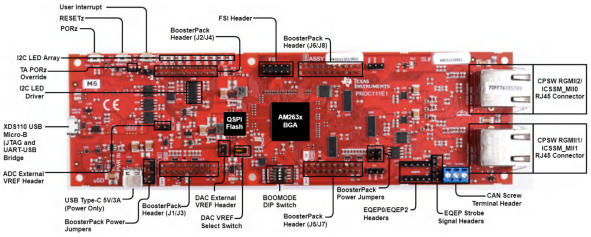
Na slici 2.1. prikazani su mikrokontroleri AM2xx generacije opremljeni Cortex-R procesorima koji su namijenjeni aplikacijama u stvarnom vremenu. U radu je korišten označeni mikrokontroler AM263x.



Slika 2.1. Arm® Cortex®-R mikrokontroleri poredani po brzini procesora i memoriji [2]

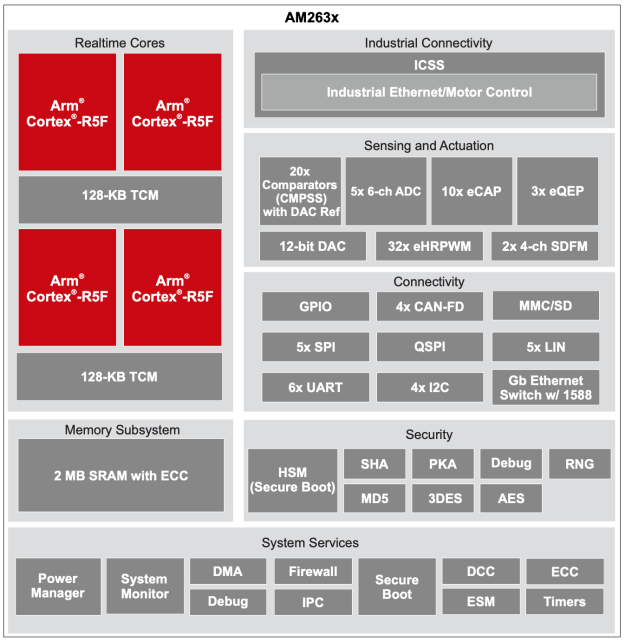
Za brži i olakšani razvoj laboratorijskih sklopova proizvođač nudi mikroprocesorske sustave koji na tiskanoj pločici pored mikrokontrolera sadrže svu potrebnu elektroniku za njihov rad te izvod periferija na praktične konektore. Tako je za ovaj rad korišten sustav "LaunchPad AM263x". Slika 2.2 prikazuje tiskanu pločicu ovoga sklopa uz naznačene izvode i konektore. U radu se koriste izvodi *"BoosterPack Header J1, J2, ..."* za ADC ulaze i PWM izlaze, *"EQEP2"* za enkoder,

"RJ45, CPSW RGMII1" za spajanje na web sučelje, *"USB Micro-B"* za programiranje mikrokontrolera, *"USB Type-C"* za napajanje mikroprocesorskog sustava.



Slika 2.2. LaunchPad AM263x [3]

Na "LaunchPad AM263x" mikroprocesorskom sustavu nalazi se mikrokontroler AM2634B za koji je blokovski dijagram dan na slici 2.3. Blok *"Realtime Cores"* prikazuje četiri mikroprocesora *"Arm Cortex-R5F"* zajedno s njihovom bliskom radnom memorijom *"128-KB TCM (engl. Tightly Coupled Memory)"*. U radu su iskorištena dva od četiri mikroprocesora i sva bliska memorija. Periferije za povezivanje s drugim sklopovima ili računalom prikazane su blokom *"Connectivity"* koji prikazuje *"Gb Ethernet Switch"* za povezivanje s računalom u svrhu prikaza web sučelja. Periferije za mjerenje fizikalnih veličina i upravljanje drugim sklopovima prikazane su blokom *"Sensing and Actuation"*. Od ovih periferija korišteni su analogno digitalni pretvornici (*"ADC"*), generatori PWM signala (*"eHRPWM"*) te periferija za obradu signala enkodera (*"eQEP"*). Programski kod koji ne stane u blisku memoriju postavljen je u SRAM memoriju koja je fizički udaljenija od mikroprocesora no znatno većeg kapaciteta. Ova memorija prikazana je blokom *"2 MB SRAM with ECC"*.



Slika 2.3. Blokovski dijagram mikrokontrolera AM2634B [4]

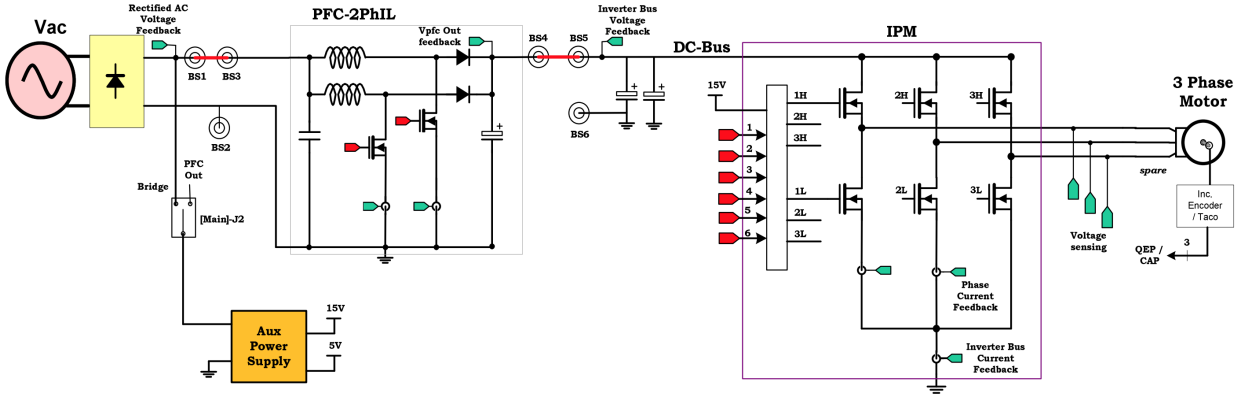
Karakteristike korištenog mikrokontrolera AM2634B [4]:

- **Mikroprocesori:** četiri Arm® Cortex®-R5F mikroprocesora
  - **Frekvencija takta:** 400 MHz
  - **l-pred memorija** 16KB po svakome mikroprocesoru s kontrolom grešaka (ECC)
  - **D-pred memorija** 16KB po svakome mikroprocesoru s kontrolom grešaka (ECC)
  - **Bliska radna memorija** 64KB po svakome mikroprocesoru s kontrolom grešaka (ECC)
- **Među procesorska komunikacija:**
  - **SPINLOCK:** Za sinkronizaciju izvršavanja programa u dva ili četiri procesora
  - **MAILBOX:** Za slanje podatka između procesora
- **Memorija:**
  - **Radna memorija:** 2MB (4x512MB) s kontrolom grešaka (ECC)
  - Podržava DMA (Direct memory access)
- **Periferije za komunikaciju:**
  - **6 kanala UART**
  - **5 kanala SPI**
  - **4 kanala I2C**
  - **Gigabitni Ethernet Switch s dva ulaza/izlaza**
- **do 140 GPIO nožica** dostupan broj ovisi o pakiranju integriranog sklopa
- **5 analogno digitalnih pretvarača preciznosti od 12 bita** brzina: 4MSPS (Mega Samples Per Second)
- **1 digitalno analogni pretvarača preciznosti od 12 bita**
- **32 modula za generiranje signala širinsko impulsne modulacije (PWM)**
- **4 modula za mjerenje signala enkodera**

3 LABORATORIJSKI POSTAV

HVMotorCtrl+PFCKit-R5 hardversko je razvojno okruženje koje na tržištu nudi tvrtka Texas Instruments. Ova razvojna elektronika namijenjena je edukaciji i ispitivanju korištenja C2000 familije mikrokontrolera u svrhu razvoja proizvoda energetske elektronike na visokom naponu za upravljanje elektromotornim pogonima. Sastoji se od ispravljača mrežnog napona, elektronike za korekciju faktora snage, trofaznog mosnog invertera, te elektronike za povezivanje s mikrokontrolerom. Sadrži sve potrebne elektroničke sklopove za rad na naponu do $400\,\mathrm{V}$ te svu potrebnu mjernu elektroniku za prilagodbu mjerenih signala mikrokontroleru. Iz tog razloga ovim proizvodom se uz korištenje raznih algoritama može upravljati asinkronim i sinkronim strojevima. [5]

Na slici 3.1 dan je blokovski dijagram energetske elektronike HVMotorCtrl+PFCKit-R5. Lijevi dio dijagrama prikazuje elektroniku za ispravljanje izmjeničnog napona u istosmjerni, a desni dio dijagrama prikazuje mosni spoj zajedno s pozicijama mjernih otpornika za mjerenje struje elektromotora.



Slika 3.1. Blokovski dijagram HVMotorCtrl+PFCKit-R5 energetske elektronike [6]

HVMotorCtrl+PFCKit-R5 prikazan je na slici 3.2 te se sastoji od plastičnog kućišta u kojem se nalazi se tiskana pločica na kojoj su postavljene elektroničke komponente. Osim energetske elektronike koja je prikazana na slici 3.1, na tiskanoj pločici nalazi se i analogno sklopovlje za prilagodbu naponskih razina mikrokontroleru te digitalno sklopovlje za programiranje mikrokontrolera. Za povezivanja mikrokontrolera s energetskom elektronikom dostupan je *DIM100* konektor za umetanje kartice mikrokontrolera familije C2000. Kako se u radu ne koristi familija mikrokontrolera C2000, već AM263x, digitalno sklopovlje i *DIM100* konektor nije moguće iskoristiti jer proizvođač ne nudi kompatibilne module mikroprocesorskih sustava iz familije AM263x. Iz tog je razloga u svrhu povezivanja AM263x serije mikrokontrolera na energetsku elektroniku HVMotorCtrl+PFCKit-R5 razvojnog okruženja izrađen adapter kojim se bilo koji mikroprocesorski sustav radne naponske razine $3.3\,\mathrm{V}$ može direktno spojiti na mosni pretvarač i na mjerne

otpornike za mjerenje struje elektromotora koji se nalaze na HVMotorCtrl+PFCKit-R5 tiskanoj pločici.

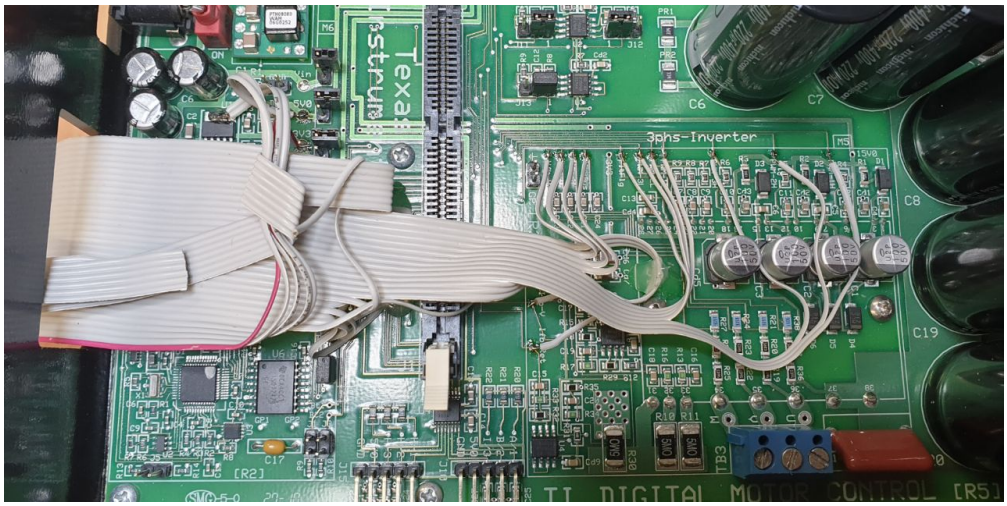
Slika 3.2. Energetska elektronika "HVMotorCtrl+PFCKit-R5"

Radi preglednosti rada u daljnjem će se tekstu tiskana pločica HVMotorCtrl+PFCKit-R5 energetske elektronike nazivati samo **"tiskana pločica energetske elektronike"**, adapter signala između mikroprocesorskog sustava LaunchPad AM263x i HVMotorCtrl+PFCKit-R5 energetske elektronike samo **"adapter"**, a LaunchPad AM263x mikroprocesorski sustav samo **"LaunchPad mikroprocesorski sustav"**. Kako je tiskana pločica energetske elektronike namijenjena ispitivanju, ispunjena je testnim točkama. Testne točke dijelovi su bakrenih vodova iznad kojih nije postavljena izolacija što omogućuje jednostavno spajanje mjerne opreme na željene točke strujnog kruga. Na ove točke moguće je zalemiti vodove pa su iskorištene u svrhu izrade adaptera. Na tiskanoj pločici energetske elektronike pored svake točke ispitivanja napisana je oznaka signala koja se koristi u shemi HVMotorCtrl+PFCKit-R5 strujnog kruga koja je dana u prilogu 10.6. Oznake signala koji su potrebni za izradu adaptera dani su u prilogu 10.2 tablicom 10.2.

Adapter se sastoji od tri dijela:

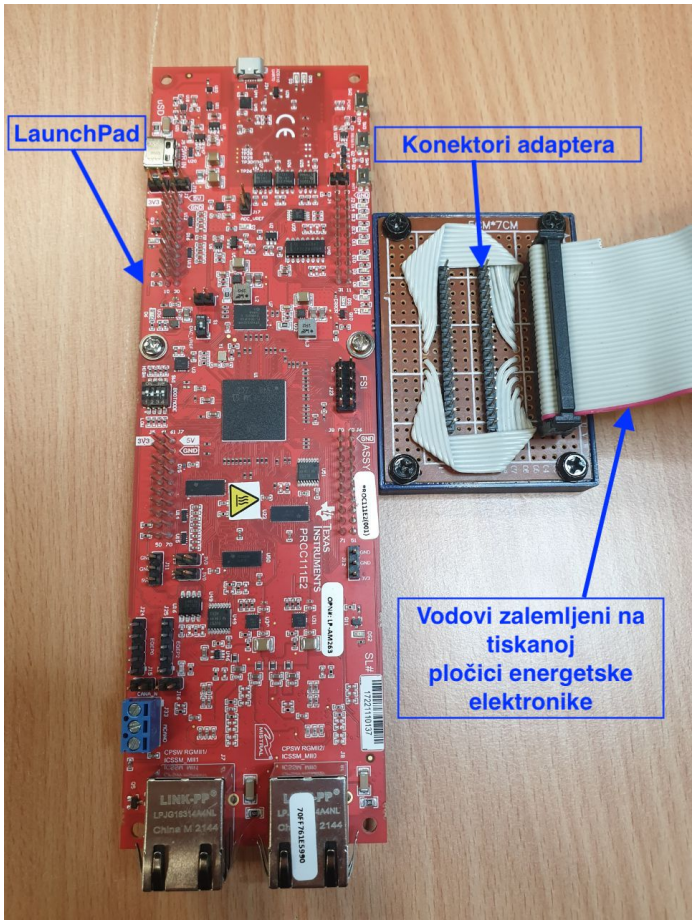
1. Skupine vodova koji su s jedne strane zalemljeni na tiskanu pločicu energetske elektronike, a s druge strane na ženski konektor
2. Tiskana pločica s dva konektora, jedan je za spajanje sa skupinom vodova zalemljenih na tiskanu pločicu energetske elektronike, a drugi za povezivanje s LaunchPad mikroprocesorskim sustavom
3. Skupine vodova koji se spajaju između konektora na adapteru i konektora na LaunchPad mikroprocesorskim sustavom

Na slici 3.3 prikazana je skupina vodova koja je zalemljena na testne točke tiskane pločice energetske elektronike.



Slika 3.3. Zalemljeni vodovi na tiskanoj pločici energetske elektronike

Adapter s vidljivim konektorima prikazan je na slici 3.4 gdje je pričvršćen na LaunchPad mikroprocesorski sustav.



Slika 3.4. Adapter pričvršćen na LaunchPad mikroprocesorski sustav

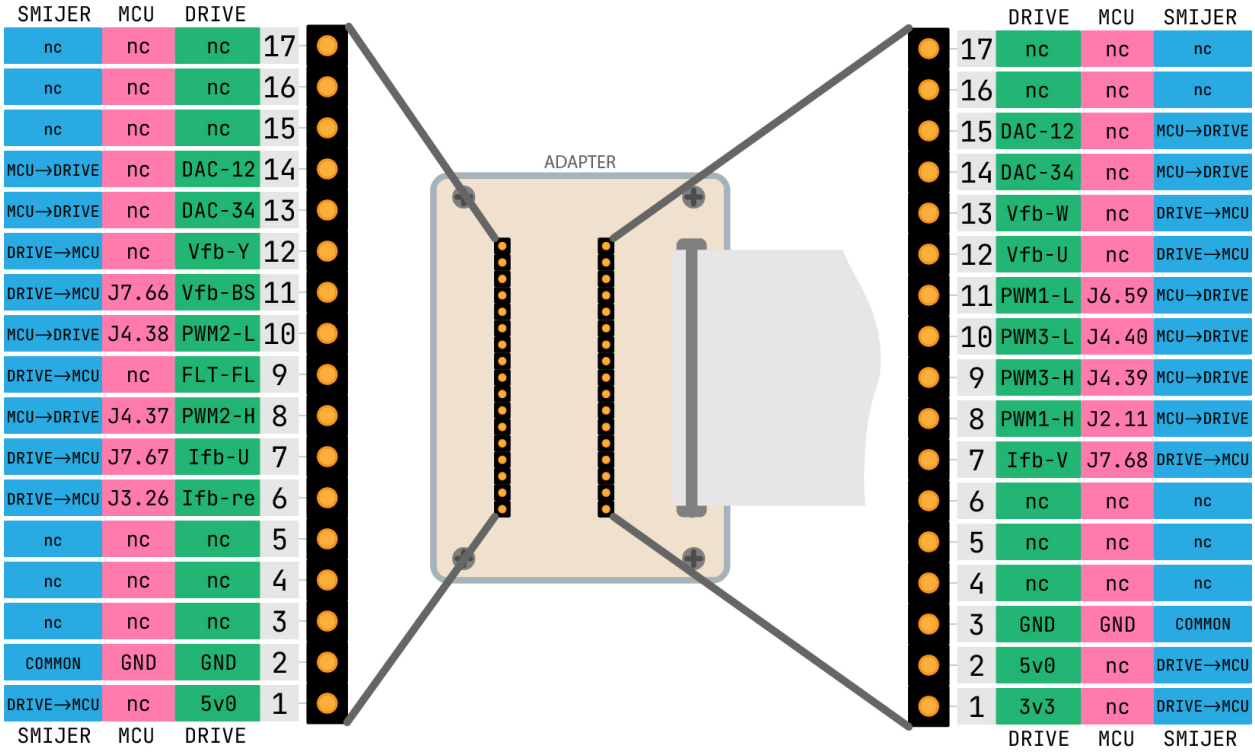
Korišteni demo projekt u C programskom jeziku korišten kao baza za implementaciju vektorskog upravljanja sadrži SysConfig konfiguraciju kojom su oznake signala u programskom kodu povezane s fizičkim pinovima mikrokontrolera. Pinovi su zatim LaunchPad mikroprocesorskim sustavom povezani s lako dostupnim konektorima. Informacije o tome kako su povezani pinovi mikrokontrolera i LaunchPad mikroprocesorskog sustava dostupne su u dokumentaciji proizvođača [3], a u prilogu 10.6 tablicama 10.3 i 10.4 prikazane su oznake signala kako su definirane u SysConfig konfiguraciji, njihova svrha i na kojem konektoru LaunchPad mikroprocesorskog sustava su dostupni.

Povezivanjem oznaka signala s tiskane pločice energetske elektronike (tablica 10.2), oznaka signala programskog koda (tablice 10.3 i 10.4) te analizom sheme dostupne u prilogu 10.6 definirana je tablica potrebnih spojnih vodova između tiskane pločice energetske elektronike i LaunchPad mikroprocesorskog sustava 3.1.

Tablica 3.1. Oznake i opis spojnih vodova između tiskane pločice energetske elektronike i LaunchPad mikroprocesorskog sustava

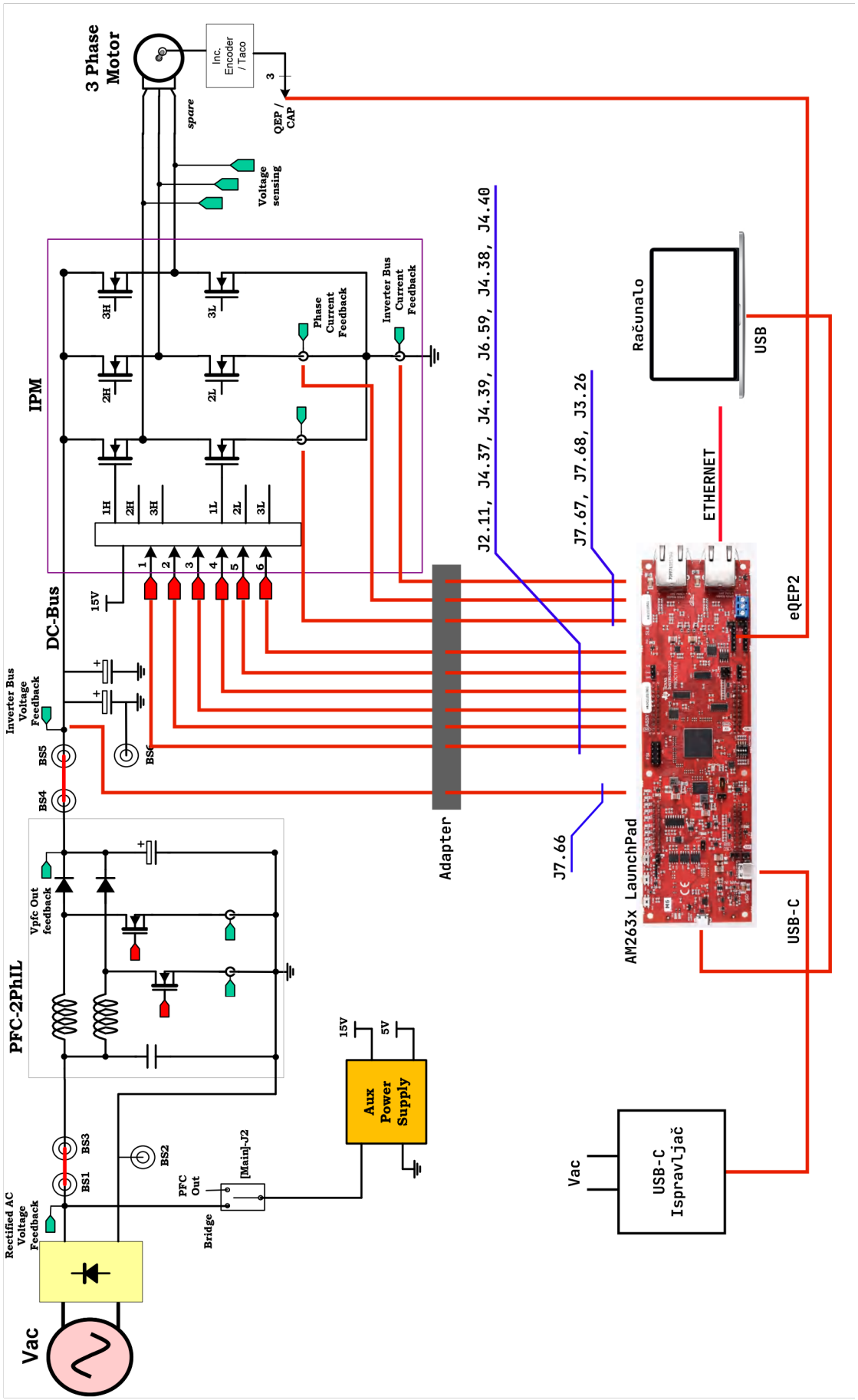
Oznaka na tiskanoj pločici	**Pin konektora na LaunchPad-u**	**Opis signala**
Vfb_BS	J7.66	Napon DC međukruga
Ifb_U	J7.67	Struja faze a elektromotora
Ifb_V	J7.68	Struja faze b elektromotora
Ifb_re	J3.26	Struja zvjezdišta elektromotora
PWM1_H	J2.11	PWM signal faze a za sklopku mosnog spoja prema napajanju
PWM1_L	J6.59	PWM signal faze a za sklopku mosnog spoja prema masi
PWM2_H	J4.37	PWM signal faze b za sklopku mosnog spoja prema napajanju
PWM2_L	J4.38	PWM signal faze b za sklopku mosnog spoja prema masi
PWM3_H	J4.39	PWM signal faze c za sklopku mosnog spoja prema napajanju
PWM3_L	J4.40	PWM signal faze c za sklopku mosnog spoja prema masi

Na slici 3.5 prikazan je raspored oznaka iz tablice 3.1 na konektoru koji se nalazi na adapteru. Slika 3.5 koristi se za spajanje vodova koji povezuju adapter i LaunchPad mikroprocesorski sustav.



Slika 3.5. Raspored signala na adapteru

Radi jasnog razumijevanja uloge adaptera i načina na koji su povezani LaunchPad mikroprocesorski sustav i energetska elektronika, na slici 3.6 je prikazana blokovska shema svih međusobnih veza i veze s računalom.



Slika 3.6. Shema laboratorijskog postava

4 PROGRAMSKA APLIKACIJA ZA VEKTORSKO UPRAVLJANJE

4.1 Vektorsko upravljanje asinkronim strojem

Vektorsko upravljanje naziv je koji obuhvaća algoritme upravljanja sinkronih i asinkronih strojeva te omogućuje neovisno upravljanje magnetskim tokom i momentom stroja. Koristi se u sustavima gdje je potrebna visoka točnost i brzina odziva u dinamičkim uvjetima. Cijena prednosti ovih algoritama je numerička složenost i potreba za preciznim mjerenjem električnih veličina stroja [7]. U sustavima kod kojih asinkroni stroj većinu vremena provodi u stacionarnom stanju uglavnom se koristi skalarno upravljanje koje se bazira na održavanju omjera napona i frekvencije pri promijeni brzine vrtnje. Nedostatak skalarnog upravljanja je to što i napon i frekvencija zasebno utječu i na magnetski tok i na moment stroja pa je nemoguće izvesti regulaciju visokih performansi u prijelaznim pojavama.

Cilj je vektorskog upravljanja postići razdvojeno upravljanje tokom i momentom asinkronog stroja, stoga se definira analogija s istosmjernim elektromotorom koji se sastoji od odvojenog uzbudnog i armaturnog namota. Reguliranjem istosmjerne struje uzbudnog namota upravlja se tokom magnetiziranja istosmjernog stroja, a reguliranjem istosmjerne struje armaturnog namota upravlja se momentom istosmjernog stroja. Vektorsko upravljanje svodi složene jednadžbe magnetskog toka i momenta asinkronog stroja na jednadžbe analogne jednadžbama toka i momenta istosmjernog elektromotora.

Izmjenične električne veličine prvo se transformiraju u istosmjerne kombinacijom clarke (4.1) i park (4.2) transformacije. [7]

$$i_{\alpha\beta}(t) = \frac{2}{3}\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}\begin{bmatrix} i_a(t) \\ i_b(t) \\ i_c(t) \end{bmatrix} \tag{4.1}$$

$$\begin{bmatrix} f_d \\ f_q \end{bmatrix} = \begin{bmatrix} \cos\varepsilon & \sin\varepsilon \\ -\sin\varepsilon & \cos\varepsilon \end{bmatrix} \cdot \begin{bmatrix} f_\alpha \\ f_\beta \end{bmatrix} \tag{4.2}$$

Clarke transformacija koristi se za prijelaz iz troosnog mirujućeg koordinatnog sustava u dvoosni mirujući koordinatni sustav. Zatim se park transformacijom prelazi u rotirajući koordinatni sustav čime se iz referentne pozicije rotirajućeg sustava izmjenične vrijednosti svode na istosmjerne. Primjenom clarke te zatim parke transformacije tri rotirajuća vektora faznih pomaka $120\circ$ svedena su na dva mirujuća vektora.

Prilikom park transformacije za postizanje raspregnutog upravljanja tokom i momentom ključno je za rotirajući sustav odrediti brzinu vrtnje i referentni vektor. U ovisnosti o brzini vrtnje i refe-

rentnom vektoru razlikuju se dva algoritma vektorskog upravljanja. U radu je implementirano vektorsko upravljanje asinkronim strojem u kojemu se sustav vrti sinkronom brzinom stroja, a d os dq sustava postavljena je u smjeru vektora toka rotora asinkronog stroja. Promjenom struje d osi isključivo se utječe na magnetski tok, a promjenom struje q osi isključivo na moment asinkronog stroja jer su struje d i q osi ortogonalne. Izraz za moment asinkronog stroja u dq koordinatnom sustavu (4.3) sličan je izrazu za moment istosmjernog stroja (4.4).

$$m_{em} = k_m i_{qs} \Psi_r = k'_m i_{qs} i_{ds} \tag{4.3}$$

$$m_{em} = k_m i_a \Phi_u = k'_m i_a i_u \tag{4.4}$$

Izraz (4.3) daje električni moment asinkronog stroja gdje je i_{qs} struja statora u q osi, a i_{ds} struja statora u q osi. Izraz (4.4) daje električni moment istosmjernog gdje je i_a struja armature, a i_u struja uzbude. Strujom i_{qs} odnosno i_a upravlja se momentom stroja, a strujom i_{ds} odnosno i_u upravlja se uzbudom stroja.

Osim FOC (eng. Field Oriented Control) vektorskog upravljanja koje je implementirano u radu, postoji još DTC (eng. Direct Torque control) i MPC (Model Predictive Control).

4.2 Struktura upravljanja asinkronim strojem za regulaciju brzine

U sklopu rada implementirana je struktura upravljanja za regulaciju brzine vrtnje rotora asinkronog elektromotora napajanog frekvencijskim pretvaračem s utisnutim naponom. Povratna veza brzine ostvarena je optičkim enkoderom koji je montiran na rotor elektromotora, a povratna veza struje zatvorena je direktnim mjerenjem faznih struja elektromotora.

Struktura se sastoji od jedne vanjske regulacijske petlje koja je upravo regulacija brzine vrtnje, te unutarnje petlje za struju I_Q. Također se sastoji od regulacijske petlje za struju I_D. U poglavlju 4.1 spomenuto je da se strujom I_D regulira magnetski tok stroja te se na ovu petlju dovodi konstantna referentna vrijednost. Sustavi regulacije kojima se asinkroni stroj vrti brzinom većom od nazivne brzine vrtnje stroja koristi se dodatni regulator toka.

Strujom I_Q upravlja se momentom pa je referentna vrijednost na ulazu regulatora ove struje izlaz regulatora brzine jer su brzina vrtnje rotora i moment povezani drugim Newtonovim zakonom (4.5)

$$\vec{M} = I\vec{a} \tag{4.5}$$

Gdje je: $\vec{M}$ moment sile, I moment tromosti, $\vec{a}$ akceleracija.

Pa tako, ako se rotor vrti brže od referentne vrijednosti, regulator brzine smanjit će referentnu vrijednost struje I_Q. Time će se na motor narinuti manji napon V_Q što će rezultirati smanjenjem

stvarne struje I_Q motora koja direktno smanjuje električni moment i doći će do usporavanja rotora. Isto vrijedi i za obratni slučaj.

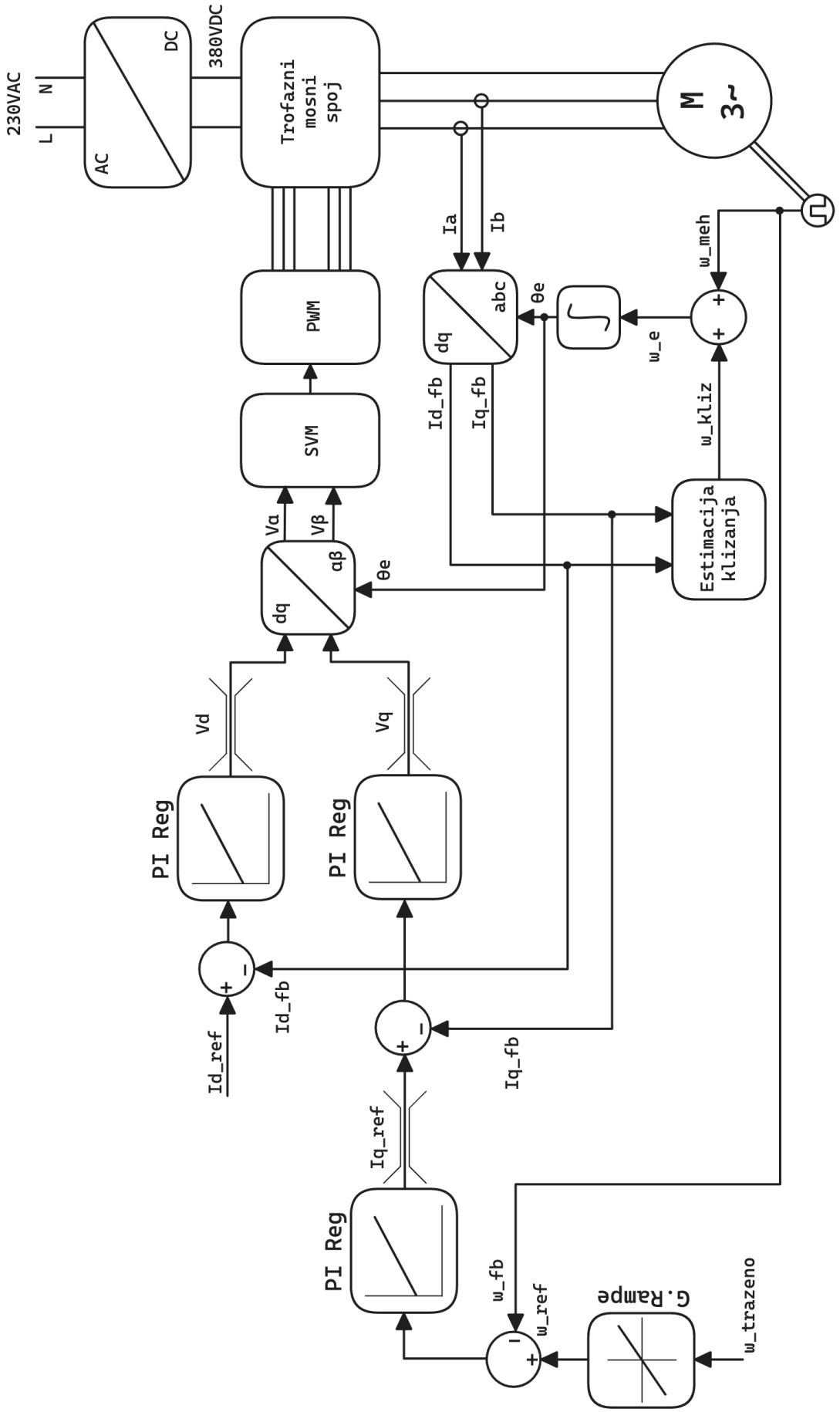
Na slici 4.1 prikazan je blokovski dijagram vektorskog upravljanja koje je implementirano na procesoru. Opis oznaka dan je u tablici 4.1.

Tablica 4.1. Opis oznaka za sliku 4.1

Oznaka	Opis
w_trazeno	Željena brzina vrtnje
w_ref	Referentna vrijednost brzine vrtnje
w_fb	Mjerena brzina vrtnje
w_meh	Mjerena brzina vrtnje
w_kliz	Estimirana brzina klizanja
w_e	Estimirana električna brzina
Iq_ref	Referentna struja regulatora struje q
Iq_fb	Mjerena struja q osi
Id_ref	referentna struja regulatora struje d

Oznaka	Opis
Id_fb	Mjerena struja osi d
Ia	Mjerena struja faze a
Ib	Mjerena struja faze b
Vd	Referentni napon d osi
Vq	Referentni napon q osi
Vα	Referentni napon u $\alpha\beta$ sustavu
Vβ	Referentni napon u $\alpha\beta$ sustavu
Θ**e**	Električni kut rotora
SVM	Vektorska modulacija
PWM	Širinsko-impulsna modulacija

Željena brzina vrtnje dovodi se na generator rampe kojim se postupno podiže referentna vrijednost željene brzine, izlaz generatora rampe referentna je vrijednost regulatora brzine vrtnje čiji se izlaz dovodi na ulaz referentne vrijednosti regulatora struje q osi. Parametar željene struje d osi dovodi se direktno na regulator. Izlazi regulatora struja Id i Iq referentni su naponi za d i q os koji se zatim transformiraju iz rotirajućeg dq sustava u mirujući $\alpha\beta$ sustav pomoću trenutne vrijednosti električnog kuta rotora (izraz (4.2)). Naponi $\alpha\beta$ sustava dovode se na ulaz bloka vektorske modulacije (engl. Space Vector Modulation - SVM) kojemu je izlaz faktor opterećenja svake faze za pulsno-širinsku modulaciju (engl. Pulse Width Modulation - PWM). Pomoću tih triju PWM signala upravlja se frekvencijskim pretvaračem tako da su dovedeni na tranzistore mosnog spoja. Vrijednost trenutnog električnog kuta rotora dobiva se mjerenjem mehaničke brzine vrtnje rotora i faznih struja motora. Pomoću faznih struja estimira se klizanje asinkronog stroja koje se dodaje mehaničkoj brzini rotora. Dobivena električna brzina zatim se integrira po vremenu čime se dobije električni kut rotora. U daljnjem tekstu biti će opisani dijelovi regulacijske strukture uz implementaciju u C programskom jeziku za AM263x mikrokontroler.



Slika 4.1. Blokovski dijagram implementiranog vektorskog upravljanja

4.3 Uvod u Code Composer Studio C projekt

Za razumijevanje dalje danih opisa implementacija u C programskom jeziku potrebno je imati na umu pomoćne strukture i konfiguracijske datoteke pa je u poglavlju 4.3 dan njihov opis. U blokovima programskog koda u prvome redu bit će napisan naziv datoteke iz koje je prikazani programski kod (npr. `// main.c`) te ako varijable ili metode nisu opisane u tekstu opis im je dan u komentarima programskog koda.

Konfiguracija svih perifernih jedinica vrši se grafičkim alatom *SysConfig* tvrtke *Texas Instruments* koji omogućuje jednostavnu konfiguraciju novijih generacija mikrokontrolera. Opcije koje alat nudi prethodno su od proizvođača prilagođene svakome mikrokontroleru, pakiranju mikrokontrolera i mikroprocesorskom sustavu. Time alat može pružati vrlo korisne povratne informacije kao što je na primjer koji izvodi mikrokontrolera su dostupni a koji nisu ovisno o pakiranju ili na kojim izvodima je povezana određena periferna jedinica. Bez ovoga alata konfiguracija bi se sastojala od detaljnog proučavanja tehničke dokumentacije mikrokontrolera koja može biti razdijeljena u više dokumenata pa je sklonost greškama veća.

Također, proizvođač je za periferne jedinice razvio driver-e koji omogućuju jednostavnije korištenje istih u programskom kodu. *SysConfig* prilikom obrade unesene konfiguracije generira definicijski programski kod te inicijalizacijski kod spomenutih driver-a. Razvojni inženjer mora u pravome trenutku pozvati generirane metode te je time zadovoljena inicijalizacija mikrokontrolera i njegovih perifernih jedinica. Ispod je dan programski kod koji prikazuje pozivanje generiranih metoda u `main()` metodi programa.

```
// main.c

#include "ti_drivers_config.h"
#include "ti_board_config.h"
void trinv_main(void *args);

int main()
{
  /* Inicijalizacija SysConfig konfiguracije */
  System_init();
  Board_init();

  // Pokretanje programa za izvođenje vektorskog
  // upravljanja
  trinv_main(NULL);

  Board_deinit();
  System_deinit();
  return 0;
}
```

Parametri regulatora i motora definirani su u dvjema header datotekama: *Motor/Trinv_param.h* i *Motor/Motor_param.h*. Ovdje se nalaze sve konstante koje se mogu mijenjati kako bi se ostatak programskog koda i algoritma prilagodio željenim uvjetima u kojima će raditi.

Ispod je dana definicija parametara motora koji se koriste za izvođenje algoritma vektorskog upravljanja. Nazivni napon motora postavljen je na 100 V umjesto 230 V kako bi se zaštitila energetska elektronika prilikom ispitivanja algoritma. Također je ovdje definirana numeracija dva stanja u kojima može biti status motora: upaljen, ugašen.

```
// Motor_param.h

// Električni parametri motora
#define RS          11.05f              // Otpor statora (ohm)
#define RR          6.11f               // Otpor rotora (ohm)
#define LS_D        0.005f              // Induktivitet statora u D osi (H)
#define LS_Q        0.005f              // Induktivitet statora u D osi (H)
#define LS          0.31642f            // Induktivitet statora
#define FLUX        0.0904f             // Back EMF (V/Hz)
#define POLES       4                   // Broj polova
#define PAIRS       2                   // Broj pari polova

// Nazivni napon je 230V, ali je zbog sigurnosti pri testiranju
// postavljeno ograničenje na 100V
#define RATED_VOLTAGE   100.0f          // Nazivni napon motora (V)

typedef enum
{
  MOTOR_STOP = 0,
  MOTOR_RUN = 1
} MotorRunStop_e;
```

Ispod su dane definicije parametara koji su povezani uz izvođenje vektorskog upravljanja i mjerenja električnih veličina. Tako je ovdje unesena frekvencija uzorkovanja što je ključna informacija za točne proračune. Ovdje se također nalaze faktori za skaliranje vrijednosti koju ADC očita u fizikalne veličine točne mjerne jedinice. Definiran je faktor skaliranja za napon (`ADC_V_REFRENCE_SCALE`) i faktor skaliranja za struju(`ADC_I_SCALE_FCT`). Definirano je i kašnjenje koje ADC periferna jedinica unosi u povratnu vezu kako bi se moglo kompenzirati pri izračunu struja u d i q osi ovisno o mjerenim strujama.

```
// Trinv_param.h

// Frekvencija takta procesora koja je odabrana u SysConfig (MHz)
#define CONTROLSS_FREQUENCY    200.0
#define SYSTEM_FREQUENCY       CONTROLSS_FREQUENCY

// Frekvencija uzorkovanja i PWM signala (KHz)
#define PWM_FREQUENCY           10.0
#define ISR_FREQUENCY           (PWM_FREQUENCY)
#define ISR_RES_RATIO           0U

// Bazne vrijednosti po kojima se normaliziraju veličine
#define BASE_VOLTAGE          340.0f  // (volt)
#define BASE_CURRENT          10.0f   // (amp)
#define BASE_FREQ             50.0f   // (Hz)

// 1/2^12
#define ADC_PU_SCALE_FACTOR 0.000244140625f

// Skaliranje adc očitanja napona za referentni napon i djelitelj napona
// (1/rezolucija adc) * (referentni napon) * (djelitelj napona)
#define ADC_V_REFRENCE_SCALE (ADC_PU_SCALE_FACTOR) * (3.22693f) * (125.0f)


```

```
// 1/2^11
#define ADC_PU_PPB_SCALE_FACTOR 0.000488281250f

// Skaliranje adc očitanja struje za vrijednost shunta i pojačanje diff pojačala
// (referentni napon) / ((pojačanje diff pojačala) * (vrijednost shunta) * (rezolucija adc))
#define ADC_I_SCALE_FCT 0.0012605195f

// Trajanje uzimanja uzorka za ADC (Sample and Hold time)
#define ADC_S_H_TIME_NS     75.0f
// Trajanje konverzije ADC-a
#define ADC_CONV_TIME_NS    175.0f
```

Kako bi programski kod bio čitljiviji grupe povezanih varijabli povezane su u C strukture. Pa tako sve varijable vezane za motor kao što su struje u troosnom mirujućem, dvoosnom mirujućem i dvoosnom rotirajućem sustavu, kut rotora, brzina vrtnje, organizirane u strukturi `Motor_t`. Slično su organizirane i varijable za PI regulator u strukturi `PI_Obj`, generator rampe, strujni model asinkronog stroja i generiranje PWM signala. Ispod je dana struktura motora `Motor_t`. Zbog čitljivosti nisu prikazane sve varijable, potpuna struktura dana je u prilogu 10.3.1.

```
// foc.h

typedef struct _Motor_t_
{
  float32_t I_abc_A[3];   // Struje faza abc [A]

  float32_t I_scale;      // ADC faktor skaliranja struje
  float32_t V_scale;      // ADC faktor skaliranja napona

  float32_t dcBus_V;           // Izmjereni napon DC linka [V]
  float32_t oneOverDcBus_invV; // 1/dcBus_V

  float32_t I_ab_A[2];     // Struja u alpha beta sustavu [A]
  float32_t I_dq_A[2];     // Struje u dq sustavu [A]

  float32_t theta_e;       // Električni kut rotora [rad]
  float32_t omega_e;       // Električna brzina rotora [rad/s]
  float32_t Sine;          // Sinus kuta rotora
  float32_t Cosine;        // Kosinus kuta rotora

  float32_t theta_e_out;   // Kompenzirani električni kut rotora [rad]
  float32_t Sine_out;      // Sinus komp. el. kuta rotora
  float32_t Cosine_out;    // Kosinus komp. el. kuta rotora

  float32_t Vff_dq_V[2];   // Naponi za rasprezanje d i q struje
  float32_t Vout_dq_V[2];  // Izlazni napon u dq sustavu [V]
  float32_t Vout_ab_V[2];  // Izlazni napon u alpha beta sustavu [V]

  PI_Obj pi_spd;  // Struktura PI regulatora brzine
  PI_Obj pi_id;   // Struktura PI regulatora struje d
  PI_Obj pi_iq;   // Struktura PI regulatora struje q

  .
  .
  .

} Motor_t;
```

4.3.1 Konfiguracija linker skripte

Linker skripta koristi se kako bi se prevodiocu programskog jezika definiralo na koje adrese memorije spremiti pojedine dijelove strojnog koda i početnih vrijednosti varijabli. Kako je spomenuto u poglavlju 2, ovaj mikrokontroler ima dvije razine memorije koje se razlikuju po veličini i fizičkoj blizini mikroprocesoru, pa vrijedi što bolje rasporediti programski kod kako bi se dobile što bolje performanse izvođenja programa. Adrese dostupne memorije uzete su iz tehničke dokumentacije [8] iz tablice 2-1.

Linker skripta sastoji se od dva bloka:

- **`MEMORY`**

  U *memory* bloku definiraju se dijelovi memorije tako da se odabere početna adresa i duljina memorije u bajtovima prikazanim heksadecimalnim brojevnim sustavom. Tako definiranom dijelu memorije dodijeli se proizvoljan naziv.

- **`SECTIONS`**

  Blok *sections* definira izgled izlazne datoteke linker-a koji je dio prevodioca programskog koda. Ovdje se pojedinim dijelovima programskog koda dodijeli mjesto u memoriji gdje će biti pohranjeni, na način da se direktno upiše adresa memorije ili da se koristi naziv memorijskih odjeljaka definiranih `MEMORY` blokom.

Za potrebe vektorskog upravljanja što veći dio programa pohranjen je u memoriji najbližoj mikroprocesoru jer se radi o obradi podataka u stvarnom vremenu. U `SECTIONS` bloku može se vidjeti da je programski kod `.tcma_code` postavljen u dio memorije `R5F_TCMA1` koji se nalazi unutar jednog para mikroprocesora kako se vidi na slici br. 2.3.

Kako `.tcma_code` nije standardni identifikator koji linker prepoznaje, u C programskom kodu može se naznačiti dio koda koji se želi spremiti u `.tcma_code` memorijski prostor kako je prikazano ispod.

```
1 // FOC_loop.c
2 __attribute__ ((section(".tcmb_code"))) void FOCrun_ISR(void *handle)
```

U svrhu međusobne komunikacije mikroprocesora rezerviran je dio zajedničke memorije u OCRAM-u `RTOS_NORTOS_IPC_SHM_MEM` te su u `RTOS_NORTOS_IPC_SHM_MEM` memorijski prostor postavljeni podatci označeni s `.bss.ipc_vring_mem`. Tu memoriju koristi IPC (engl. Inter Process Communication) driver koji se u radu koristi za komunikaciju između dva mikroprocesora. Na jednome procesoru izvršava se program vektorskog upravljanja, a na drugome TCP (engl. Transmission Control Protocol) server.

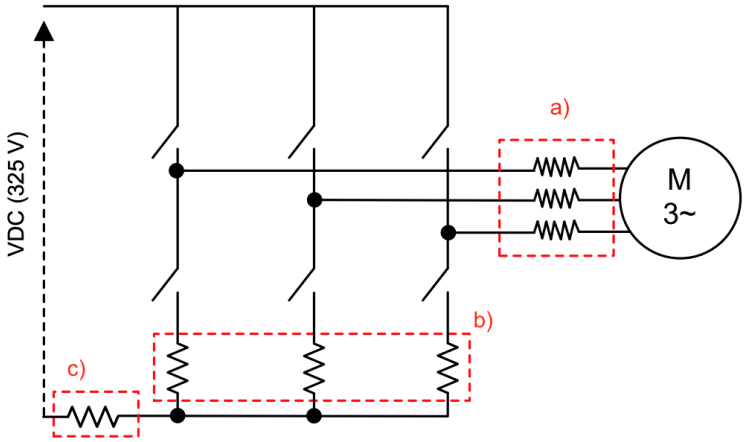
```
// linker.cmd

SECTIONS
{
  .
  .
  .

  .bss.log_shared_mem  (NOLOAD) : {} > LOG_SHM_MEM
  .bss.ipc_vring_mem   (NOLOAD) : {} > RTOS_NORTOS_IPC_SHM_MEM

  .tcma_code  : {}  > R5F_TCMA1
  .tcma_data  : {}  > R5F_TCMA2
  .tcmb_code  : {}  > R5F_TCMB1
  .tcmb_data  : {}  > R5F_TCMB2
}
MEMORY
{
  /* Adrese bliske radne memorije za svaki mikroprocesor */
  R5F_VECS  : ORIGIN = 0x00000000 , LENGTH = 0x00000040
  R5F_TCMA0  : ORIGIN = 0x00000040 , LENGTH = 0x00003FC0
  R5F_TCMA1  : ORIGIN = 0x00004000 , LENGTH = 0x00002000
  R5F_TCMA2  : ORIGIN = 0x00006000 , LENGTH = 0x00002000
  R5F_TCMB1  : ORIGIN = 0x00080000 , LENGTH = 0x00004000
  R5F_TCMB2  : ORIGIN = 0x00084000 , LENGTH = 0x00004000

  /* Blok RAM memorije koji koristi ova jezgra (R5F0-0) */
  OCRAM     : ORIGIN = 0x70000000 , LENGTH = 0x40000

  /* Zajednički blok memorije u OCRAM-u za CCS Logove*/
  LOG_SHM_MEM             : ORIGIN = 0x70040000, LENGTH = 0x00004000

  /* Zajednički blok memorije u OCRAM-u za međusobnu komunikaciju
  mikroprocesora IPC RP Message Driver-om */
  RTOS_NORTOS_IPC_SHM_MEM : ORIGIN = 0x72000000, LENGTH = 0x3E80
}
```

4.4 Mjerenje struje

Algoritam vektorskog upravljanja ovisi o vrlo točnoj vrijednosti struja faza motora jer se na bazi njih provode izračuni koji definiraju ponašanje regulatora. Iz tog razloga vrlo je bitno postići što točnije mjerenje vrijednosti i faznog pomaka struja svih faza motora. U protivnom prilikom regulacije dolazi do oscilacije momenta što dovodi do opterećenja mehaničkih komponenti i glasnog rada motora. Također netočno mjerenje struje dovodi do loše dinamike upravljanja elektromotorom, izobličenih valnih oblika i povećanih gubitaka. U poglavlju 4.4 prikazane su neke od metoda mjerenja struja elektromotora prilikom uporabe vektorskog upravljanja i mosnog spoja. [9]

Za trofazni elektromotor potrebno je znati struje svih triju faza, iste se mogu mjeriti ne nekoliko načina koji ovise o poziciji i broju mjernih otpornika u strujnome krugu mosnog spoja. Na slici br. 4.2 prikazane su tri pozicije postavljanja mjernih otpornika. Struje motora mogu se mjeriti na:

a) faznim izvodima motora
b) spoju svake grane mosnog pretvarača s masom
c) DC napajanju mosnog izmjenjivača



Slika 4.2. Metode mjerenja struje u odnosu na broj i mjesto shunt otpornika. [9]

Jedna metoda mjerenja struje je postavljanjem triju mjernih otpornika na fazne izvode motora (ili dva ako zvjezdište motora nije uzemljeno). Ova metoda smatra se najjednostavnijom jer trenutak uzimanja uzorka struje ne mora biti sinkroniziran s PWM signalom preklapanja sklopki izmjenjivača. Period uzimanja uzoraka svakako mora biti konstantan. Nedostatak ove metode je prisutnost visokog napona na mjernoj opremi pa je vrlo teško koristiti mjerne otpornike spojene na operacijska pojačala. Zato se kod ove metode koriste izolirani senzori koji su znatno skuplji. [9]

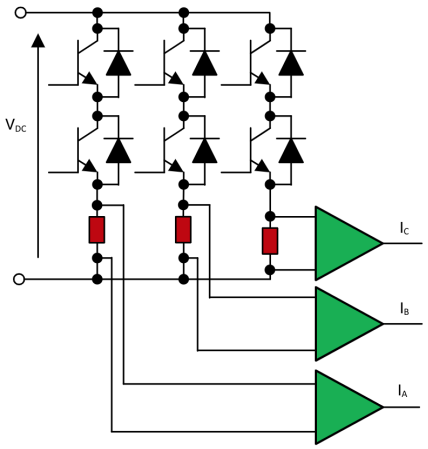
Druga metoda je postavljanje shunt otpornika na na spoju svake grane mosnog pretvarača s masom gdje su naponske prilike znatno povoljnije jer se mjerenje radi na negativnoj strani istosmjernog napona pa je moguće koristiti mjerne otpornike i operacijska pojačala. Negativna strana ove metode je potreba za sinkronizacijom uzimanja uzoraka struje i pwm signala preklapanja sklopki zbog toga što se struja u grani može mjeriti samo u trenutku kada je sklopka prema masi zatvorena i vodom teče struja faze motora. Kako se vektorsko upravljanje izvršava na digitalnim procesorima signala to nije teško izvesti pa je ovo često korištena metoda mjerenja struje. Moguće je koristiti 3 ili 2 shunt otpornika jer se struja treće grane može dobiti iz vrijednosti druge dvije.

Struju svih triju faza moguće je izmjeriti i jednim shunt otpornikom ako se struja mjeri na istosmjernom napajanju. Međutim, ovaj način mjerenja struje zahtijeva vrlo brzo operacijsko pojačalo i sekvencijalno uzimanje uzorka, jedna faza iza druge. Jedina prednost ove metode je korištenje samo jednog mjernog otpornika i jednog operacijskog pojačala.

4.4.1 Mjerenje struje pomoću tri mjerna otpornika u granama izmjenjivača

Mjerenje struje pomoću 3 shunt otpornika u granama izmjenjivača kako je prikazano na slici br. 4.3 ima prednost da je moguće koristiti sporija operacijska pojačala. Te je moguće od 3 mjerna uzorka odabrati 2 najpovoljnija za trenutak u kojem su uzeti uzorci. Odabiru se uzorci onih faza na

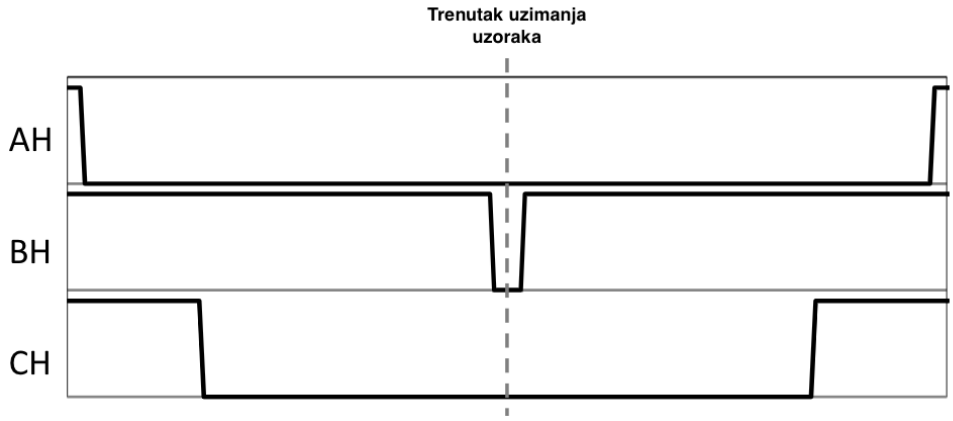
kojima je najdulje vrijeme zatvorena sklopka prema masi mosnog spoja što dozvoljava ustaljivanje prijelaznih pojava.



Slika 4.3. Mjerenje struje pomoću 3 shunt otpornika u granama izmjenjivača. [9]

Na slici 4.4 dan je primjer na kojemu je prikazan jedan period pwm signala svih triju faza za sklopke prema visokom naponu. Može se primijetiti da su sva tri signala centrirana te se uzorak struje uzima u sredini vremena kada su navedene sklopke otvorene. Signal na gornjoj i donjoj sklopci mosnog spoja komplementaran je što znači da se uzorak uzima kada su donje sklopke zatvorene.

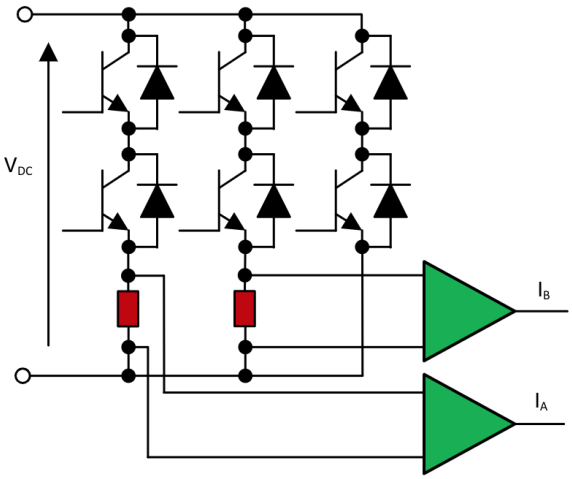
Fazama A i C sklopke su otvorene znatno dulje nego fazi B pa se kod izračuna struja faza može uzeti samo uzorke faze A i C te iz njih dobiti struju faze B, a uzorak struje faze B zanemariti jer je prozor vremena u kojemu se mogao izmjeriti vrlo kratak.



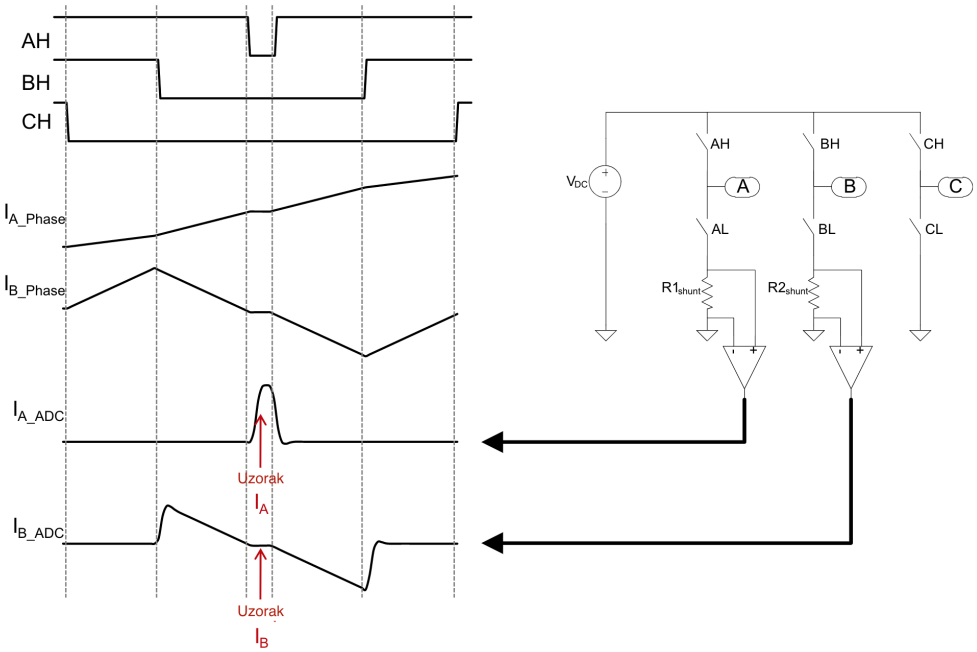
Slika 4.4. PWM signal svih triju faza s naznačenim trenutkom uzimanja uzorka struja. [9]

4.4.2 Mjerenje struje pomoću dva mjerna otpornika u granama izmjenjivača

Mjerenje struje pomoću dva mjerna otpornika moguće je, no zahtijeva znatno brža operacijska pojačala u odnosu na metodu s tri mjerna otpornika kako bi se vjerno moglo izmjeriti struju faze u vrlo kratkom prozoru vremena kada pwm radi na faktoru opterećenja blizu 1.



Slika 4.5. Mjerenje struje pomoću 2 shunt otpornika u granama izmjenjivača. [9]

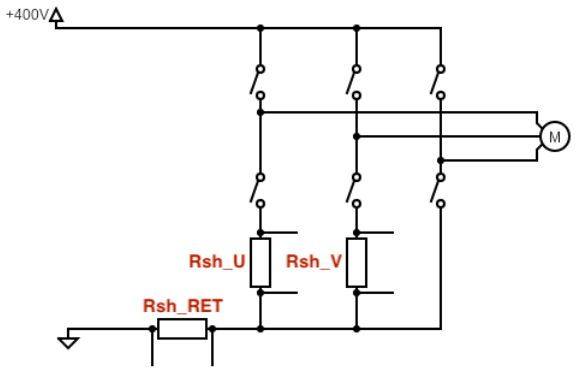


Slika 4.6. Nedovoljno vremena za mjerenje struje faze A [9]

Na slici br.4.6 može se vidjeti da ukoliko mjerna oprema nema dovoljno širok frekvencijski opseg može doći do uzimanja uzorka koji ne predstavlja točnu struju zato što što se signal nije stigao ustaliti. Mjerenje faze I_B ima dovoljno vremena, no faze I_A nema.

4.4.3 Laboratorijski rezultati mjerenja struje motora

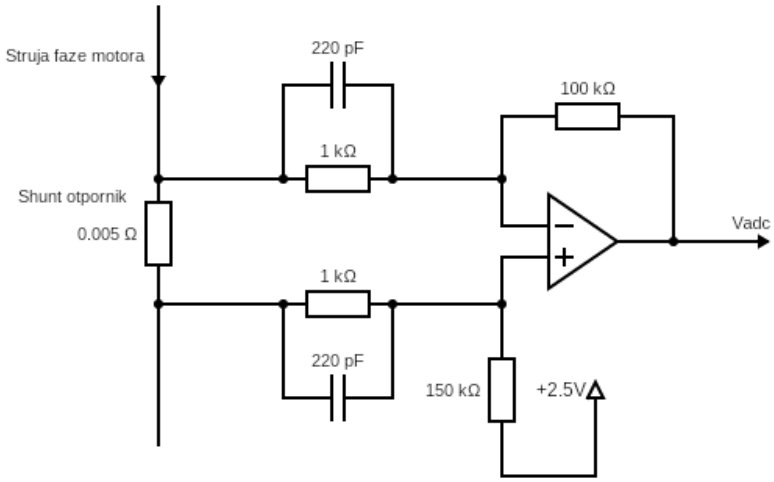
U radu korištena je metoda mjerenja struje s dva mjerna otpornika. Na energetskoj elektronici dostupan je i treći mjerni otpornik, no on je postavljen na povrat istosmjerne grane, odnosno u zvjezdište. Shema korištenog spoja mjernih otpornika prikazana je na slici 4.7.



Slika 4.7. Dva shunt otpornika u granama izmjenjivača, a jedan u grani istosmjernog napajanja

Stezaljke svakog mjernog otpornika dovedene su na dva ulaza diferencijalnih pojačala izvedenih operacijskim pojačalima. Diferencijalno pojačalo tako je izvedeno da se izlaznom naponu dodaje istosmjerna komponenta od 1.7 V jer je izlaz pojačala spojen na analogno digitalni pretvarač mikrokontrolera koji može mjeriti samo pozitivne napone dok napon na otporniku može poprimiti i negativne vrijednosti. Naknadno se u programskom kodu ta dodana istosmjerna komponenta oduzima od vrijednosti mjerenja kako bi se dobila stvarna vrijednost napona na otporniku. Izmjereni napon direktno je proporcionalan vrijednosti struje koja teče u mjerenoj grani.

Shema spoja diferencijalnog pojačala jedne faze prikazan je slikom br. 4.8.



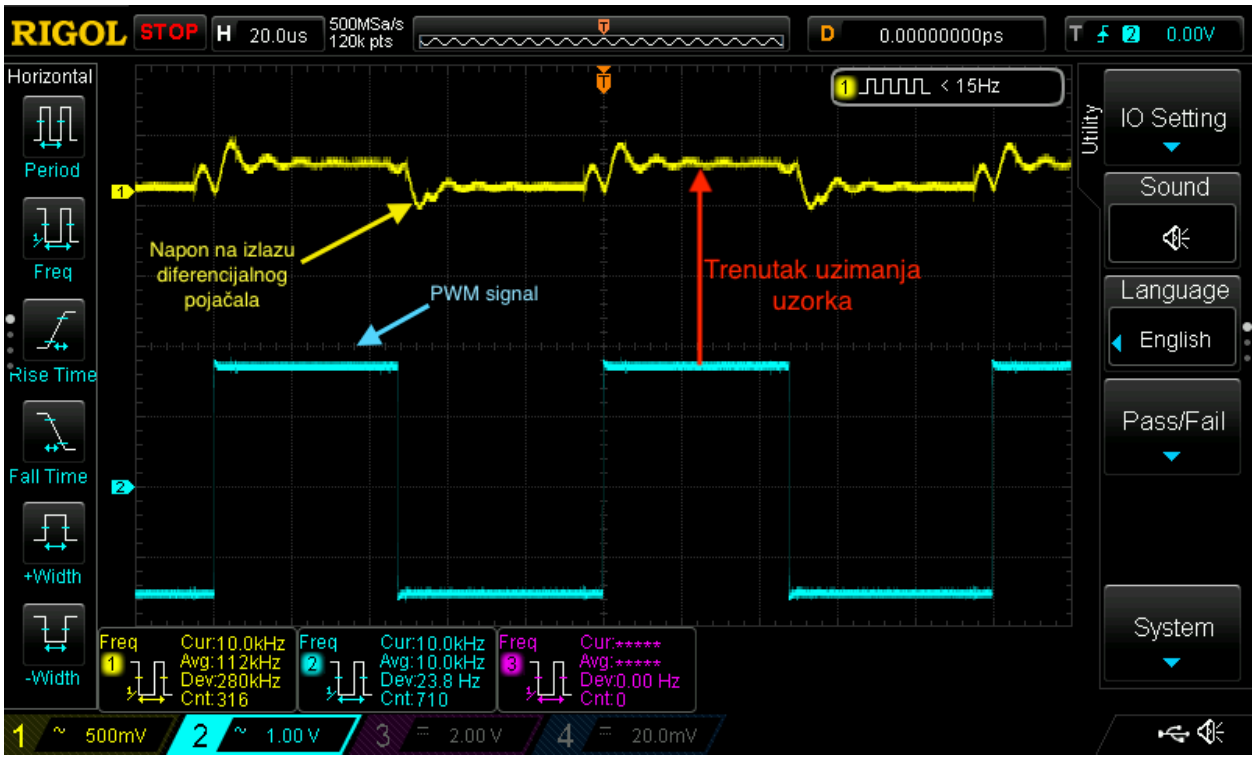
Slika 4.8. Diferencijalno pojačalo za mjerenje struje pomoću shunt otpornika.

Vrijednost mjernog otpornika je $5\,\mathrm{m\Omega}$ te je iz tog razloga pojačanje diferencijalnog pojačala postavljeno na $A = 125$ u rasponu napona na otporniku od $1\,\mathrm{mV}$ do $5\,\mathrm{mV}$ ($200\,\mathrm{mA}$ do $1\,\mathrm{A}$) kako bi se omogućilo mjerenje malih struja elektromotora koje će se pojavljivati u radu jer su ispitivanja uglavnom rađena u praznome hodu motora. Takvim pojačanjem za željeni raspon stvarnih struja ($\pm 2\,\mathrm{A}$) dobije se raspon napona na ulazu u analogno digitalni pretvornik od $0.45\,\mathrm{V}$ do $2.95\,\mathrm{V}$ što je u dozvoljenom rasponu ulaznog napona analogno digitalnog pretvornika $V_{ADC} \epsilon\, [0\,\mathrm{V}; 3.3\,\mathrm{V}]$.

Izračun razlučivosti mjerenja struje dan je izrazom (4.6).

$$\Delta I_{min} = \frac{V_{adcRef}}{A \times R_{shunt} \times N_{adc}} = \frac{3.3\,\mathrm{V}}{125 \times 0.005\,\Omega \times 4096}$$
$$\Delta I_{min} = 1.26\,\mathrm{mA} \tag{4.6}$$

Na slici 4.9 prikazana su dva signala, signal broj 1 izlaz je iz diferencijalnog pojačala koje mjeri napon na shunt otporniku, signal broj 2 PWM je signal koji generira mikrokontroler. Crvenom strelicom naznačen je trenutak u kojemu AD pretvornik mikrokontrolera uzima uzorak struje.

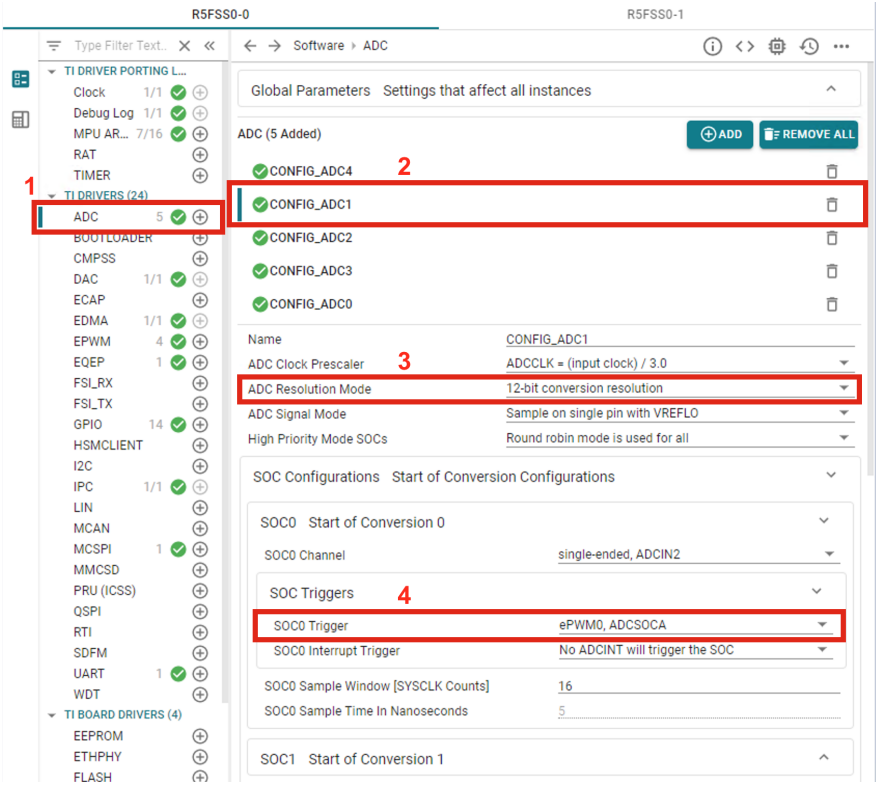


Slika 4.9. Mjerenje PWM signala i napona na izlazu diferencijalnog pojačala pri praznome hodu elektromotora.

4.4.4 Implementacija mjerenja struje u programskom kodu

Prvo je potrebno pravilno izvesti konfiguraciju ADC periferne jedinice. Za konfiguraciju je korišten alat SysConfig te je ispod dana slika br. 4.10 s odabranim postavkama za AD pretvornik

broj 1 koji se koristi za mjerenje jedne od struja faza. AD pretvornici za mjerenje ostale dvije struje i napon istosmjernog međukruga postavljene su iste postavke.



Slika 4.10. SysConfig konfiguracija ADC periferne jedinice

U listi lijevo slike br. 4.10 odabrana je opcija *ADC* (označeno brojem 1) te je odabran AD pretvornik 1 (označeno brojem 2). Za rezoluciju mjerenja odabrana je 12 bitna rezolucija (označeno brojem 3). Ključna postavka bez koje točno mjerenje struje za elektronički sklop u radu ne bi bilo moguće je u kojem vremenskom trenutku ciklusa će započinjati mjerenje struje (označeno brojem 4). Može se vidjeti da je odabrani okidač *ePWM0* što znači da je mjerenje sinkronizirano s PWM signalom i to točno onako kako je opisano u poglavlju 4.4. U poglavlju 4.7.1 opisano je kako je okidač *ePWM0* koji pokreće AD pretvorbu postavljen u centar uključenog perioda PWM signala.

Sada je u programskom kodu moguće pristupiti izmjerenim vrijednostima. Za to je radi čitljivosti definiran *C macro* kojim se blok koda može zamijeniti odabranim nazivom. Ispod su dane relevantne definicije za očitanje vrijednosti mjerenja struje.

```
// Trinv_param.h

#define IFBU \
  ADC_readResult(CSL_CONTROLSS_ADC1_RESULT_U_BASE, ADC_SOC_NUMBER0)
#define IFBV \
  ADC_readResult(CSL_CONTROLSS_ADC2_RESULT_U_BASE, ADC_SOC_NUMBER0)
#define IFBRET \
  ADC_readResult(CSL_CONTROLSS_ADC3_RESULT_U_BASE, ADC_SOC_NUMBER0)

#define IFBU_PPB \
  ADC_readPPBResult(CSL_CONTROLSS_ADC1_RESULT_U_BASE, ADC_PPB_NUMBER1)
```

```
#define IFBV_PPB \
  ADC_readPPBResult(CSL_CONTROLSS_ADC2_RESULT_U_BASE, ADC_PPB_NUMBER1)
#define IFBRET_PPB \
  ADC_readPPBResult(CSL_CONTROLSS_ADC3_RESULT_U_BASE, ADC_PPB_NUMBER1)
```

Na primjer, za vrijednost struje faze a metoda `ADC_readResult(...)` zamijenjena je nazivom `IFBU_PPB`. Definicije koje završavaju sa `_PPB` označavaju da se čita izmjerena vrijednost, ali s oduzetim DC pomakom koji je spomenut u poglavlju 4.4.3.
Metoda `ADC_readResult(ADRESA_ADCA, KANAL_ADCA)` u programskom kodu driver-a AD pretvornika čita 16 bitnu vrijednost iz registra ADC periferne jedinice, programski kod je dan ispod.

```
// adc.h

static inline uint16_t ADC_readResult(uint32_t resultBase, ADC_SOCNumber socNumber)
{
  // Pročita vrijednost adc pretvorbe iz registra za odabrani kanal
  return(HW_RD_REG16(resultBase + CSL_ADC_RESULT_ADCRESULT0 +
  socNumber * ADC_RESULT_ADCRESULTx_STEP));
}
```

Pri izvođenju vektorskog upravljanja u svakom ciklusu pročita se vrijednost s registra AD pretvornika i zapišu se u `Motor_t` strukturu za daljnje korištenje.

```
// FOC_loop.c

// Pročitaj vrijednosti AD pretvornika za mjerenje struja
motor1.I_abc_A[0] = (float32_t) IFBU_PPB;
motor1.I_abc_A[1] = (float32_t) IFBV_PPB;
// Faza W dobije se izrazom W = RET - U - V
motor1.I_abc_A[2] = (float32_t)(IFBRET_PPB - motor1.I_abc_A[0] -
  motor1.I_abc_A[1]);

.
```

Zatim se očitane vrijednosti skaliraju s faktorom definiranim u poglavlju 4.3 kako bi se dobila vrijednost u mjernoj jedinici amper. Pri inicijalizaciji strukture u varijablu `motor1.I_scale` upisana je vrijednost `ADC_I_SCALE_FCT`.

```
// FOC_loop.c

// Skaliranje u [A]
motor1.I_abc_A[0] = motor1.I_abc_A[0] * motor1.I_scale;
motor1.I_abc_A[1] = motor1.I_abc_A[1] * motor1.I_scale;
motor1.I_abc_A[2] = motor1.I_abc_A[2] * motor1.I_scale;
.
.
```

Kako bi se jednostavno moglo prikazati signal koji mikrokontroler izmjeri uz pomoć AD pretvornika osciloskopom, prilikom ispitivanja dodan je programski kod za preslikavanje uzetih uzoraka na izlaze mikrokontrolera koji ima mogućnost digitalno analogne pretvorbe. Frekvencija uzimanja uzoraka struje pa tako i frekvencija preslikavanja tih uzoraka na izlaz je $10\,\mathrm{kHz}$.

Ispod navedeni blok programskog koda izvršava se frekvencijom od $10\,\mathrm{kHz}$ te prikazuje čitanje

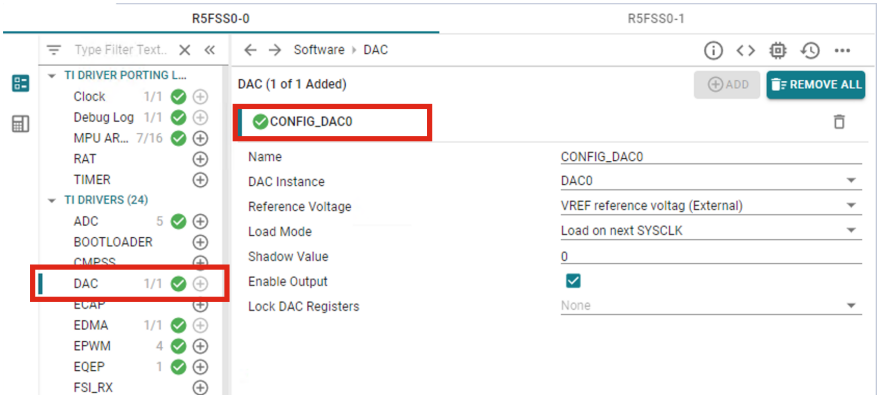
uzorka AD pretvornika te preslikavanje istoga na DA izlaz.

```
// FOC_loop.c

// Pročitaj vrijednosti AD pretvornika za mjerenje struja
motor1.I_abc_A[0] = (float32_t) IFBU_PPB;
motor1.I_abc_A[1] = (float32_t) IFBV_PPB;
// Faza W dobije se izrazom W = RET - U - V
motor1.I_abc_A[2] = (float32_t)(IFBRET_PPB - motor1.I_abc_A[0] -
motor1.I_abc_A[1]);

if (motor1.I_abc_A[0] < (0x0FFFU)) {
  // Postavljanje izmjerene struje na DAC izlaz, DEBUG
  DAC_setShadowValue(CONFIG_DAC0_BASE_ADDR, motor1.I_abc_A[0]);
}
.
.
```

Na slici br. 4.11 prikazano je mjerenje signala koji je ulaz na analogno digitalni pretvornik i signala koji je izlaz digitalno analognog pretvornika.

Slika 4.11. Signal na ulazu u AD pretvornik i signal na izlazu DA pretvornika.

Korištenje periferije za digitalno analognu pretvorbu omogućeno je alatom SysConfig. Slika br. 4.12 prikazuje konfiguraciju digitalno analogne pretvorbe.

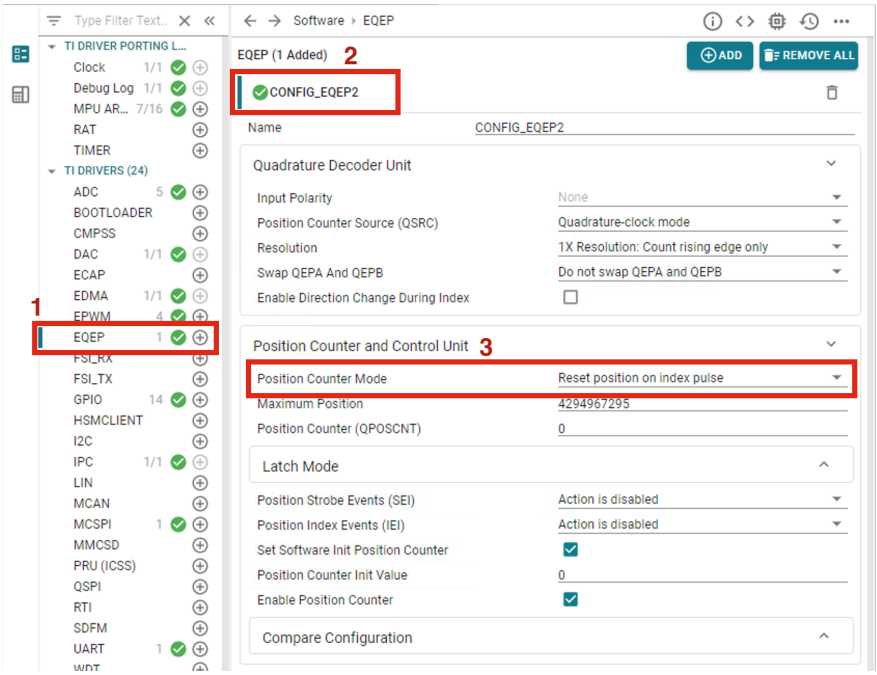


Slika 4.12. SysConfig konfiguracija DAC periferne jedinice.

4.5 Mjerenje brzine vrtnje rotora

Mjerenje brzine vrtnje rotora elektromotora izvedeno je pomoću inkrementalnog enkodera prikazanog na slici 10.2. S mikrokontrolerom povezan je pomoću konektora eQEP koji je dostupan na LaunchPad pločici. eQEP konektor povezan je na pinove mikrokontrolera koji se mogu koristiti za integriranu perifernu jedinicu *eQEP* koja olakšava korištenje enkodera tako da obavlja funkciju brojenja impulsa za određivanje brzine i pozicije te redoslijeda impulsa za mjerenje smjera vrtnje. U programskom kodu potrebno je čitati broj izbrojenih impulsa iz registra *eQEP* periferne jedinice i na osnovu broja impulsa vršiti proračun brzine vrtnje.

Na slici 4.13 prikazana je konfiguracija *eQEP* periferne jedinice u SysConfig alatu. U listi dostupnih perifernih jedinica odabrano je *EQEP* (označeno brojem 1). Na LaunchPad-u je enkoder spojen na eQEP konektor broj 2 pa je odabrana periferna jedinica 2 (označeno brojem 2) (mikrokontroler ima dvije *eQEP* periferne jedinice). Inkrementalni enkoder pored dva signala kojima broji korake prilikom vrtnje rotora daje i *engl. index* signal. Index signal daje jedan impuls za svaku potpunu rotaciju od 360 stupnjeva i uvijek se nalazi na istome fizičkome mjestu enkodera. Time se po potrebi može kalibrirati odnos mjerenog kuta rotora u odnosu na pozicije polova elektromotora. Brojem 3 označena je opcija da se brojač koraka vrati na nulu svaki put kada periferna jedinica dobije index impuls.



Slika 4.13. SysConfig konfiguracija eQEP periferne jedinice.

Izračun mehaničkog kuta rotora na osnovu vrijednosti brojača dan je izrazom (4.7).

$$\Theta_{meh} = \frac{IZBROJENO\_KORAKA}{MAX\_BROJ\_KORAKA} \times 2\pi \,[\mathrm{rad}] \tag{4.7}$$

Korišteni enkoder za svaki puni okret rotora izbroji do 8192, pa je dopunjeni izračun dan izrazom (4.8).

$$\Theta_{meh} = \frac{IZBROJENO\_KORAKA}{8192} \times 2\pi \,[\mathrm{rad}] \tag{4.8}$$

Električki kut dobije se pomoću broja pari polova (p_p) kako je prikazano izrazom (4.9).

$$\Theta_{el} = \Theta_{meh} \times p_p \,[\mathrm{rad}] \tag{4.9}$$

Električna kutna brzina rotora (ω_{el}) dobije se derivacijom električnog kuta rotora (Θ_{el}) po vremenu. Kako se radi o diskretnom vremenu, derivaciju je moguće izračunati pomoću dva uzorka mjerenja uzeta u dva različita trenutka vremena poznatog razmaka (T). Period uzorkovanja isti je periodu izvođenja petlje vektorskog upravljanja jer se programski kod za izračun izvršava u svakom ciklusu. Izračun električne brzine vrtnje na osnovu dva uzorka električnog kuta rotora dan je izrazom (4.10).

$$\omega_{el} = (\Theta_{el}\,[n] - \Theta_{el}\,[n-1]) \times \frac{1}{T} \left[\frac{\mathrm{rad}}{\mathrm{s}}\right] \tag{4.10}$$

U programskom kodu sve vrijednosti računate su u per unit mjernoj jedinici, a ne u rad (drugim riječima zanemaren je član $\times 2\pi$) pa je radi čitljivosti programskog koda kutna brzina normalizirana

na baznu frekvenciju tako da $\omega_{elpu} = 1$ predstavlja iznos sinkrone brzine $\omega_{el} = 2\pi f$. Izračun električne kutne brzine u per unit mjernoj jedinici dan je izrazom (4.11).

$$\omega_{elpu} = (\Theta_{elpu}[n] - \Theta_{elpu}[n-1]) \times \frac{1}{T \times f}[pu] \tag{4.11}$$

Izračun električne kutne brzine u mjernoj jedinici $\frac{\text{rad}}{\text{s}}$ pomoću električne kutne brzine u per unit mjernoj jedinici dan je izrazom (4.12).

$$\omega_{el} = \omega_{elpu} \times 2\pi f \left[\frac{\text{rad}}{\text{s}}\right] \tag{4.12}$$

Metoda računanja brzine izrazom (4.10) predstavlja problem prilikom prolaska kroz točku indeksa kada se brojač vraća na nulu. Neka je u jednom trenutku vrijednost brojača 8190, zatim nakon nekog vremena uzimanjem sljedećeg uzorka vrijednost je 2 jer je rotor prošao kroz index. Izrazom (4.13) dobivena je mehanička kutna brzina rotora koja nije točna.

$$\omega_{meh} = (2 - 8190) \times \frac{1}{T} = -\frac{8188}{T}\frac{\text{rad}}{\text{s}} \tag{4.13}$$

Stvarna kutna brzina je $\frac{4}{T}$. Kako bi se izbjegla greška, računanje brzine se preskače u kratkom periodu vremena dok se kut rotora nalazi blizu pozicije indeksa.

4.5.1 Implementacija mjerenja brzine vrtnje rotora u programskom kodu

Definirana je struktura koja se koristi za radnju mjerenja kuta rotora i kutne brzine rotora `PosSpeed_Object` koja se u programskom kodu koristi za pristupanje potrebnim vrijednostima.

```
1  // Encoder.h
2
3  typedef struct
4  {
5      float thetaElec;         // IZLAZ: Električki kut [rad]
6      float thetaMech;         // IZLAZ: Mehanički kut [rad]
7      int16_t directionQEP;    // IZLAZ: Smjer vrtnje
8      int16_t thetaRaw;        // Brojač enkodera
9
10     // KONSTANTA: Skaliranje brojača enkodera u električne radijane
11     float mechScaler;
12
13     int16_t polePairs;       // KONSTANTA: Broj parova polova motora
14     int16_t calAngle;        // Kalibracijski kut [impulsa enkodera]
15     float speedPR;           // IZLAZ: Kutna brzina [pu]
16     uint32_t K1;             // KONSTANTA: deltaTheta -> omega
17 } PosSpeed_Object;
```

Ispod je dan programski kod kojim je inicijalizirana `PosSpeed_Object` struktura. Tu je vidljivo kako su postavljene konstante koje se koriste u proračunu. Važno je obratiti pozornost na liniju koda br. 7 gdje je definirana konstanta kao u izrazu (4.8) i na liniju koda br. 11 gdje je definirana konstanta kao u izrazu (4.11).

```
// FOC_loop.c

posSpeed.thetaElec = 0;
posSpeed.thetaMech = 0;
posSpeed.directionQEP = 0;
posSpeed.thetaRaw = 0;
posSpeed.mechScaler = 0.0001220703125f; //1/(2048*4)
posSpeed.polePairs = PAIRS;
posSpeed.calAngle = 0;
posSpeed.speedPR = 0;
posSpeed.K1 = 1/(BASE_FREQ*T); // Skaliranje razlike kuta u brzinu
posSpeed.oldPos = 0;
```

Pored strukture implementirana je metoda `PosSpeed_calculate(...)` koja kao argument uzima pokazivač na varijablu tipa strukture `PosSpeed_Object` i adresu registra *eQEP* periferne jedinice. Prilikom izvođenja prvo se očita smjer vrtnje rotora enkodera pomoću metode `EQEP_getDirection()` koja je dostupna u sklopu driver-a periferne jedinice. Zatim se pomoću metode `EQEP_getPosition()` očita trenutno stanje brojača koraka koje je periferna jedinica izbrojila. Na vrijednost brojača pribroji se kalibracijski kut te se sve skupa pomnoži s konstantom skaliranja u per unit mjernu jedinicu. Time je dobiven mehanički kut rotora. Zatim se mehanički kut rotora pomnoži s brojem pari polova čime se dobije električki kut rotora.

Ispod proračuna mehaničkog i električkog kuta nalazi se provjera ako se enkoder nalazi u neposrednoj blizini indeksa. Ako se ne nalazi u blizini indexa, izvršava se proračun kutne brzine rotora. Na kraju se u strukturu sprema kopija vrijednosti električkog kuta rotora kako bi se u sljedećem ciklusu izvršavanja mogla računati brzina. Jer, kako je već spomenuto za izračun brzine, potreban je trenutni i prethodni uzorak kuta rotora (izraz (4.11)).

```
// Encoder.c

void PosSpeed_calculate(PosSpeed_Object *p, uint32_t gEqepBaseAddr)
{
    uint16_t posCount;
    // Pročitaj smjer vrtnje
    p->directionQEP = EQEP_getDirection(gEqepBaseAddr);
    // Pročitaj brojač
    posCount = (uint16_t)EQEP_getPosition(gEqepBaseAddr);
    // Dodaj kalibracijski kut izmjerenome
    p->thetaRaw = posCount + p->calAngle;
    // Skaliraj broj impulsa u meh radijane
    p->thetaMech = (float)p->thetaRaw * p->mechScaler;
    // Kaliraj meh. kut u el. kut
    p->thetaElec = p->polePairs * p->thetaMech;

    // Ne mjeri brzinu u trenutku indeksa
    // Izbjegavanje naglog skoka brzine koji nije stvaran
    if (
        ((p->thetaMech < 0.95) & (p->thetaMech > 0.05)) ||
        ((p->thetaMech > -0.95) & (p->thetaMech < -0.05))
    ) {
        Tmp_fr = p->K1 * (p->thetaElec - p->oldPos);
    }
    else {
        Tmp_fr = p->speedPR;
    }

    // Limitiranje iznosa brzine na raspon od -1 do 1
```

```
    Tmp_fr=_SATURATE(Tmp_fr,1,-1);
    p->speedPR = Tmp_fr;
    p->oldPos = p->thetaElec;
}
```

Dalje je prikazano kako se implementirana funkcija `PosSpeed_calculate()` koristi u petlji koja izvršava algoritam vektorskog upravljanja. Ispod navedeni programski kod izvršava se neposredno poslije prethodno prikazanog koda u poglavlju mjerenja struje 4.4.4, u datoteci `FOC_loop.c`.

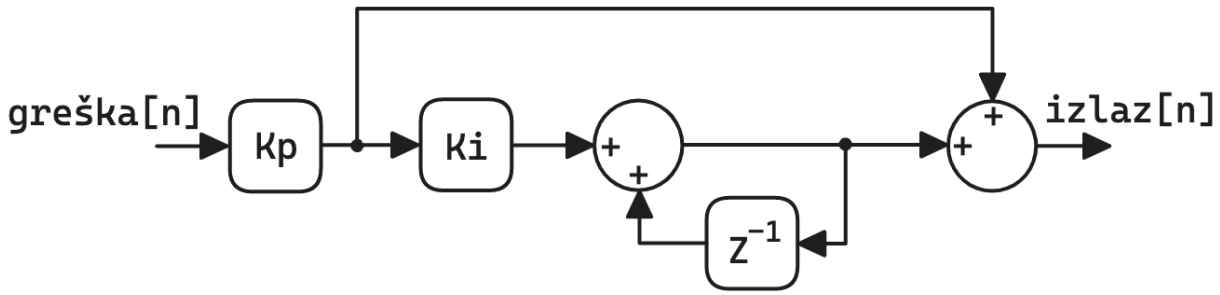
Prvo se poziva prethodno opisana metoda `PosSpeed_calculate()`. Zatim se dobivena brzina vrtnje preračuna u $\frac{rad}{s}$ izrazom (4.12), upiše u strukturu PI regulatora brzine kao vrijednost povratne veze te se pozove metoda za izvršavanje PI regulatora. Regulator brzine, kako je spomenuto u poglavlju 4.2, daje referentnu vrijednost struje I_q.

Nakon toga vrši se estimacija klizanja asinkronog stroja kako bi se moglo odrediti električnu kutnu brzinu statora, a time i električki kut statora. Kutnoj brzini rotora pribroji se estimirana kutna brzina klizanja te se kutna brzina statora integrira za dobivanje kuta statora. Proces integracije u diskretnome sustavu jednostavno je zbrajanje s vrijednošću kuta prethodnog ciklusa.

```
// FOC_loop.c

// Pročitaj enkoder
PosSpeed_calculate(&posSpeed, CONFIG_EQEP2_BASE_ADDR);
// Prijelaz iz pu u rad/s
encoder_omega = posSpeed.speedPR * BASE_FREQ * TWO_PI;

// Postavljanje vrijednosti kutne brzine
// na povratnu vezu regulatora brzine
motor1.pi_spd.fbackValue = encoder_omega;
// Izvršavanje regulatora brzine
motor1.pi_iq.refValue = PI_run_series(&(motor1.pi_spd));

// Aproksimacija klizanja
aci1.IMDs += aci1.Kr * (motor1.I_dq_A[0] - aci1.IMDs);

// Limitiranje na +/- 0.001, da se izbjegne dijeljenje s 0
aci1.IMDs = ((aci1.IMDs <  0.001) && (aci1.IMDs >= 0)) ? 0.001 : aci1.IMDs;
aci1.IMDs = ((aci1.IMDs > -0.001) && (aci1.IMDs <  0)) ? -0.001 : aci1.IMDs;

aci1.Wslip = aci1.Kt * motor1.I_dq_A[1] / aci1.IMDs;

// Računanje električne brzine i kuta STATORA
motor1.omega_e = encoder_omega + aci1.Wslip;
motor1.theta_e += motor1.sampleTime * motor1.omega_e;

// Preračunavanje kuta u +-2*pi
theta_limiter(&(motor1.theta_e));
```

4.6 Implementacija PI regulatora u programskom kodu

Regulatori struja I_d i I_q te regulator brzine vrtnje serijski su PI regulatori. Blokovski dijagram u Z domeni prikazan je na slici 4.14.



Slika 4.14. Blokovski dijagram serijskog PI regulatora.

Struktura sa slike br. 4.14 može se opisati izrazom (4.14).

$$u\left[n\right] = K_p e\left[n\right] + K_p K_i \sum_{i=0}^{n} e\left[i\right] \tag{4.14}$$

Gdje je $u\left[n\right]$ izlaz regulatora, $e\left[n\right]$ ulaz regulatora, odnosno greška, K_p proporcionalno pojačanje, i K_i integralno pojačanje. U svrhu implementacije u programskom kodu izraz (4.14) pogodnije je promatrati zapisan u rekurzivnom obliku koji je dan izrazom (4.15).

$$\begin{aligned} I\left[n\right] &= I\left[n-1\right] + K_p K_i e\left[n\right] \\ u\left[n\right] &= K_p e\left[n\right] + I\left[n\right] \end{aligned} \tag{4.15}$$

$I\left[n\right]$ rekurzivna je funkcija koja u izrazu (4.14) zamjenjuje sumu $K_p K_i \sum_{i=0}^{n} e\left[i\right]$. Dakle, za implementaciju integracijskog dijela regulatora potrebno je pamtiti jedan prethodni uzorak proračuna.

Definirana je struktura PI regulatora `PI_Obj` koja sadrži sve potrebne varijable za proračun i pohranu vrijednosti u memoriji.

```
// foc.h

typedef struct _PI_Obj_
{
    float32_t Kp;           // Proporcionalno pojačanje
    float32_t Ki;           // Integralno pojačanje
    float32_t Ui;           // I[n] u izrazima
    float32_t refValue;     // Referentna vrijednost
    float32_t fbackValue;   // Vrijednost povratne veze
    float32_t error;        // Greška
    float32_t up;           // Kp*e[n] u izrazima
    float32_t outValue;     // Izlaz
    float32_t outMin;       // Maksimalna vrijednost izlaza
    float32_t outMax;       // Minimalna vrijendost izlaza
    float32_t ffwdValue;    // Direktan prijenos na izlaz
} PI_Obj;
.
.
```

Izraz (4.15) implementiran je u metodi `PI_run_series` čiji je programski kod dan ispod. Metoda kao argument uzima pokazivač na varijablu tipa `PI_Obj`.

```
// foc.h

__attribute__ ((section(".tcmb_code"))) static inline float32_t
PI_run_series(PI_Obj* pi)
{
    // Računanje greške
    pi->error = pi->refValue - pi->fbackValue;

    // Računanje proporcionalnog dijela
    pi->up = pi->Kp * pi->error;

    // Računanje integralnog dijela
    pi->Ui = MATH_sat(
        pi->Ui + (pi->Ki * pi->up), pi->outMax, pi->outMin
    );

    // Limitiranje izlaza na odabrane granice
    pi->outValue = MATH_sat(
        pi->up + pi->Ui + pi->ffwdValue, pi->outMax, pi->outMin
    );

    return(pi->outValue);
}
```

4.7 Generiranje PWM signala vektorskom modulacijom

Vektorska modulacija jedna je od metoda upravljanja trofaznim mosnim spojem s ciljem dobivanja željenog vektora napona odnosno struje statora elektromotora. Također, vektorska modulacija pruža bolju iskoristivost napona DC kruga u odnosu na sinusnu modulaciju kod koje je maksimalna amplituda napon na izlazu izmjenjivača $\frac{1}{2}V_{DC}$ dok je kod vektorske modulacije $\frac{1}{\sqrt{3}}V_{DC}$.

Ulazna varijabla vektorske modulacije željeni je vektor napona statora, a rezultat je faktor opterećenja PWM signala svake faze koji se dovodi na sklopke mosnoga spoja. U svakoj grani trofaznog mosnog spoja nalaze se dvije sklopke koje su uvijek komplementarno pobuđene pa je za tri faze, odnosno 6 sklopki potrebno 3 PWM signala. Detaljan opis vektorske modulacije dostupan je u knjizi *Vector Control of Three-Phase AC Machines* [10] u kojoj je opisan jedan od algoritama provođenja vektorske modulacije koji se bazira na sektorima u koordinatnom sustavu. Pri računanju ovom metodom za traženi vektor je prvo potrebno odrediti sektor u kojemu se nalazi. Na osnovu sektora se iz tablice izraza odabire točan izraz za proračun faktora opterećenja. Međutim, postoji druga metoda izvedbe algoritma vektorske modulacije koju je jednostavnije implementirati u računalnom programu koji se zove *engl. Zero Sequence Modulation*. Izvodi se tako da se željeni vektor napona definira u troosnom mirujućem koordinatnom sustavu, na signal svake faze pribroji se trokutasti signal frekvencije trećeg harmonika te se dobiveni signal svake faze koristi kao referenca za generiranje PWM signala.[11]

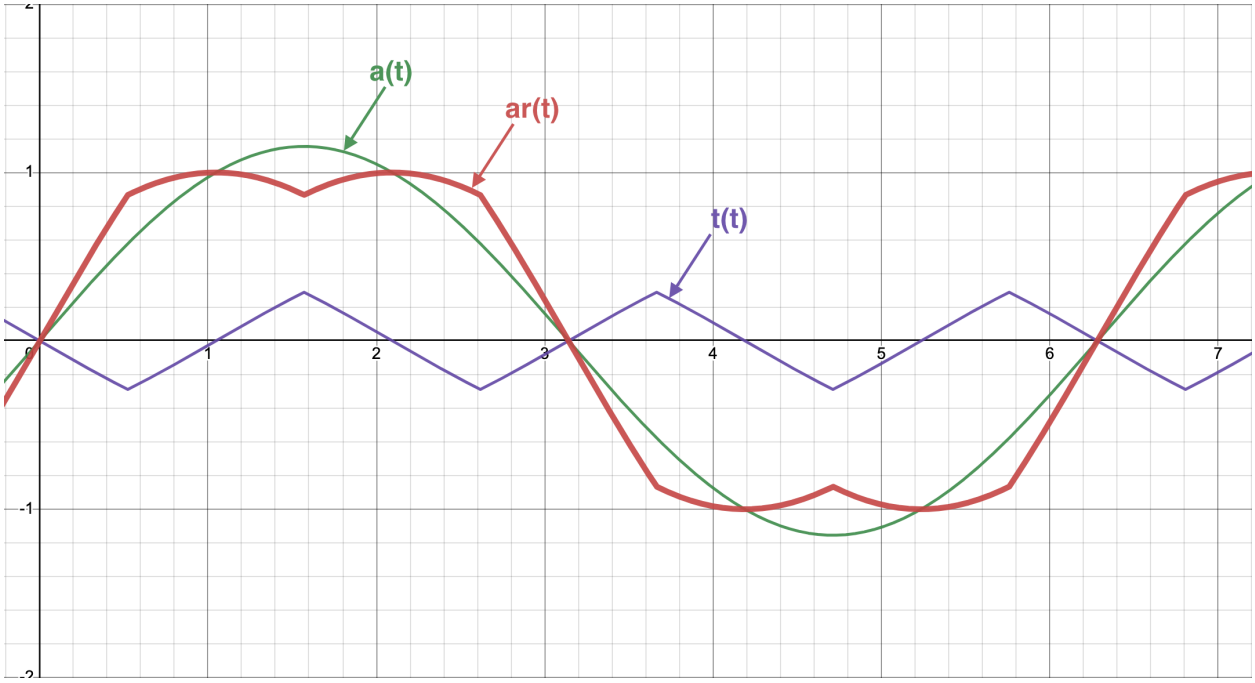
Trokutasti napon T dobije se pomoću već dostupnih vrijednosti svake faze sinusnog signala $a\,(t)$, $b\,(t)$, $c\,(t)$. Izraz za izračun zajedničkog trokutastog napona dan je ispod:

$$t\,(t) = -\frac{\min(a\,(t)\,,b\,(t)\,,c\,(t)) + \max(a\,(t)\,,b\,(t)\,,c\,(t))}{2} \tag{4.16}$$

$a\,(t)$, $b\,(t)$, $c\,(t)$ sinusni su signali svake faze napona vektora koji je dobiven iz izlaza strujnih regulatora, njima se pribroji trokutasti signal (4.17).

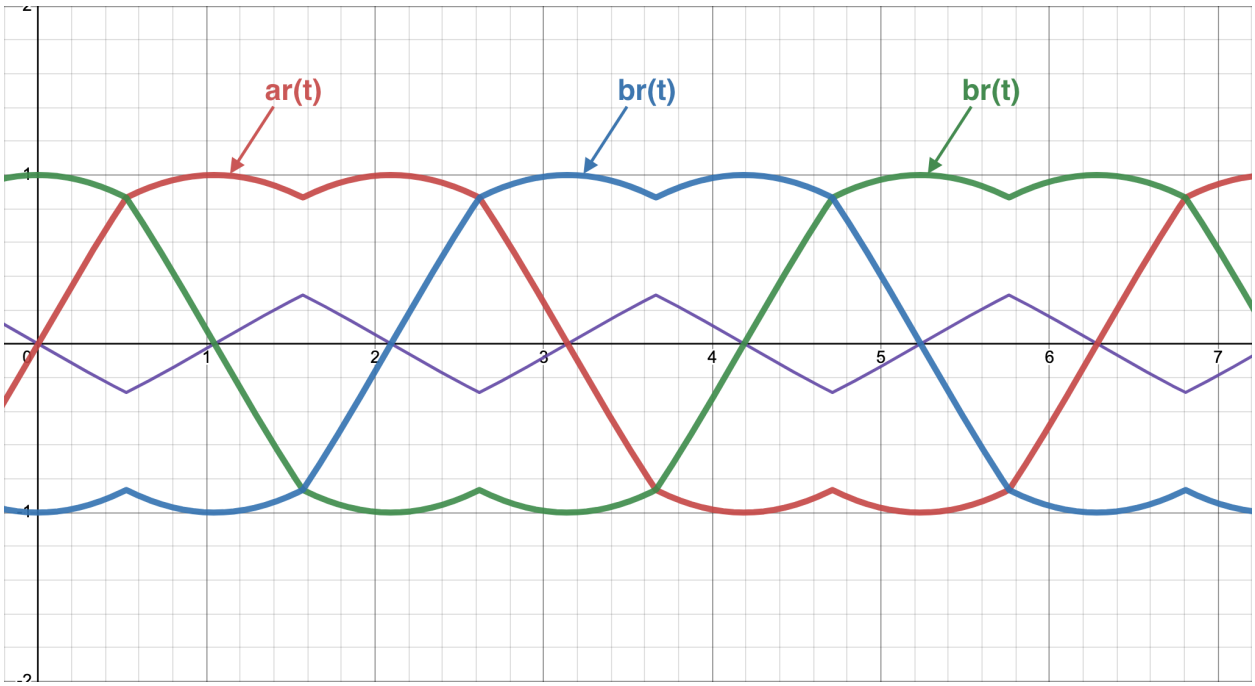
$$\begin{aligned} a_r\,(t) &= a\,(t) + t\,(t) \\ b_r\,(t) &= b\,(t) + t\,(t) \\ c_r\,(t) &= b\,(t) + t\,(t) \end{aligned} \tag{4.17}$$

Na slici 4.15 prikazani su dobiveni valni oblici za jednu fazu.



Slika 4.15. Vektorska modulacija jedne faze.

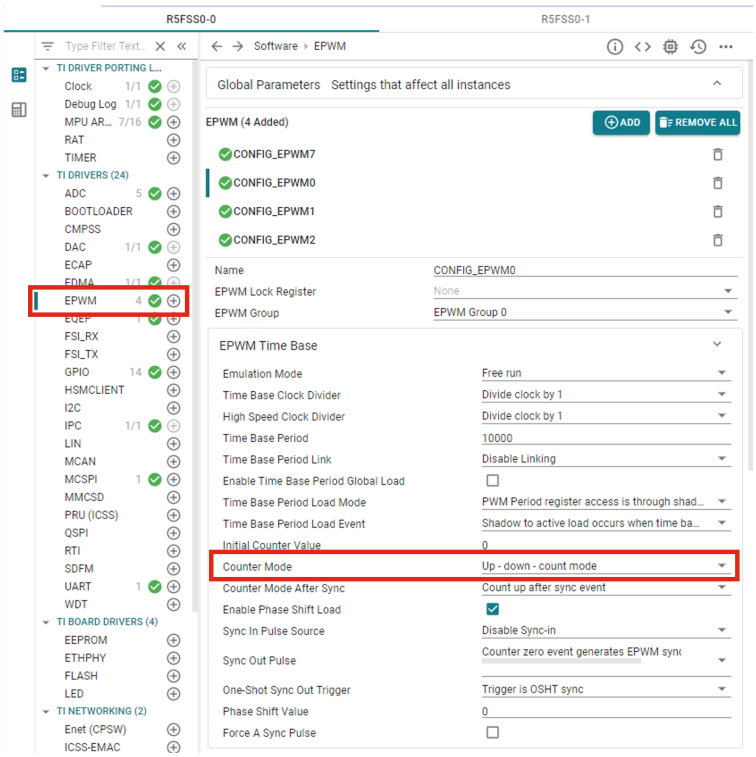
Na slici 4.16 prikazani su referentni signali sve tri faze $ar\,(t)$, $br\,(t)$, $cr\,(t)$ na osnovu kojih se generira PWM signal.



Slika 4.16. Vektorska modulacija svih faza.

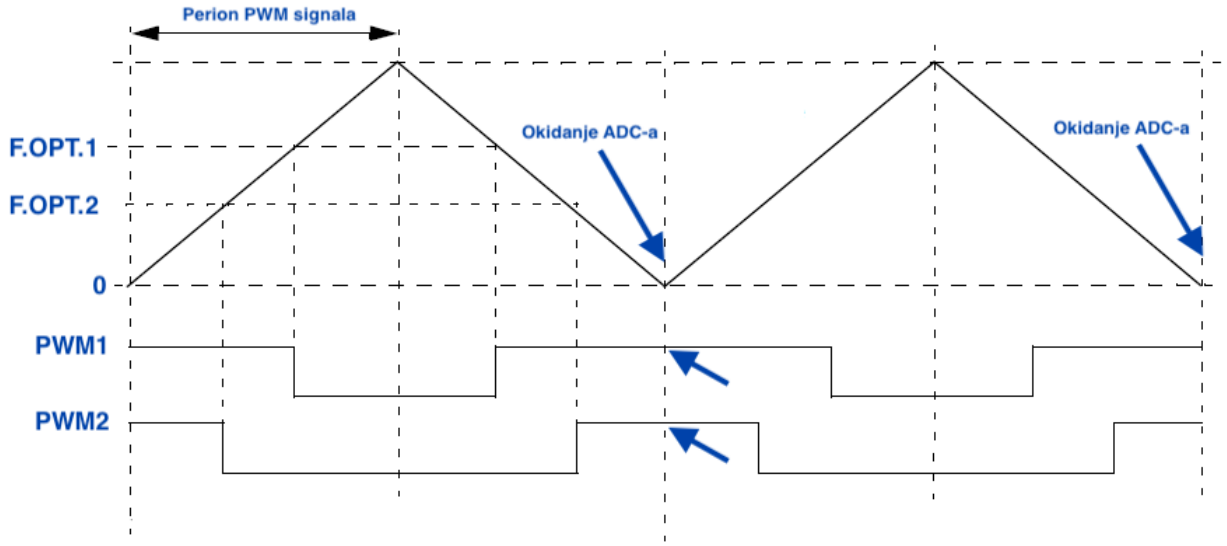
4.7.1 Implementacija vektorske modulacije u programskom kodu

Kako je za izvedbu vektorske modulacije potrebna periferna jedinica za generiranje PWM signala, prvo ju je potrebno konfigurirati pomoću SysConfig alata (slika br. 4.17).



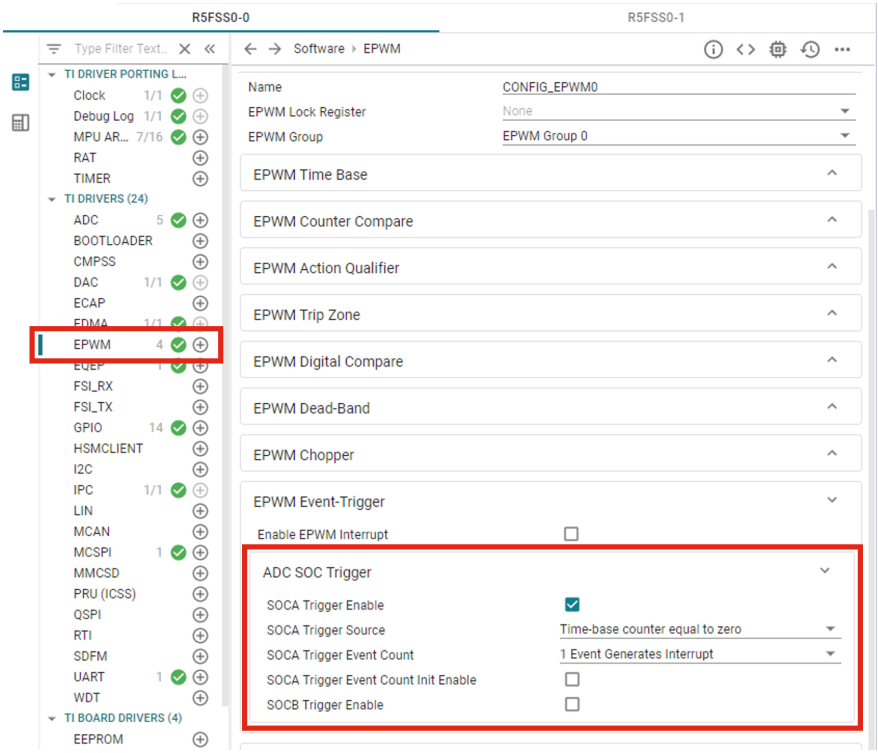
Slika 4.17. SysConfig konfiguracija PWM periferije.

Na slici 4.17 označena je postavka *"Counter Mode"* koja je postavljena na *"Up-down - count mode"* za sve PWM kanale. Brojač (*engl. Counter*) generira signal nosioc koji se koristi za generiranje PWM signala, *"Up-down - count mode"* postavkom signal nosioc trokutastog je valnoga oblika što centrira PWM signale svih triju faza (slika 4.18). Ovako poravnati PWM signal važan je iz dva razloga: jedan razlog već je spomenut u poglavlju 4.4.4 i radi se o mjerenju struje. Drugi je razlog odabrani algoritam vektorske modulacije *engl. Zero Sequence Modulation* koji pretpostavlja centralno poravnate PWM signale triju faza.



Slika 4.18. Centriranje PWM signala

Slika 4.18 prikazuje PWM brojač u obliku trokutastog valnog oblika koji broji od nule do polovice perioda PWM signala, zatim pada nazad do nule i ciklus se ponavlja. Važno je primijetiti da je centar visoke razine svih triju PWM signala u trenutku kada je brojač na broju 0. Upravo je to trenutak kada je potrebno izvršiti mjerenje struje. Stoga je u SysConfig alatu postavljeno da se generira prekid koji pokreće ADC pretvorbu svaki put kada brojač dođe na broj 0. Konfiguracija je prikazana na slici 4.19.



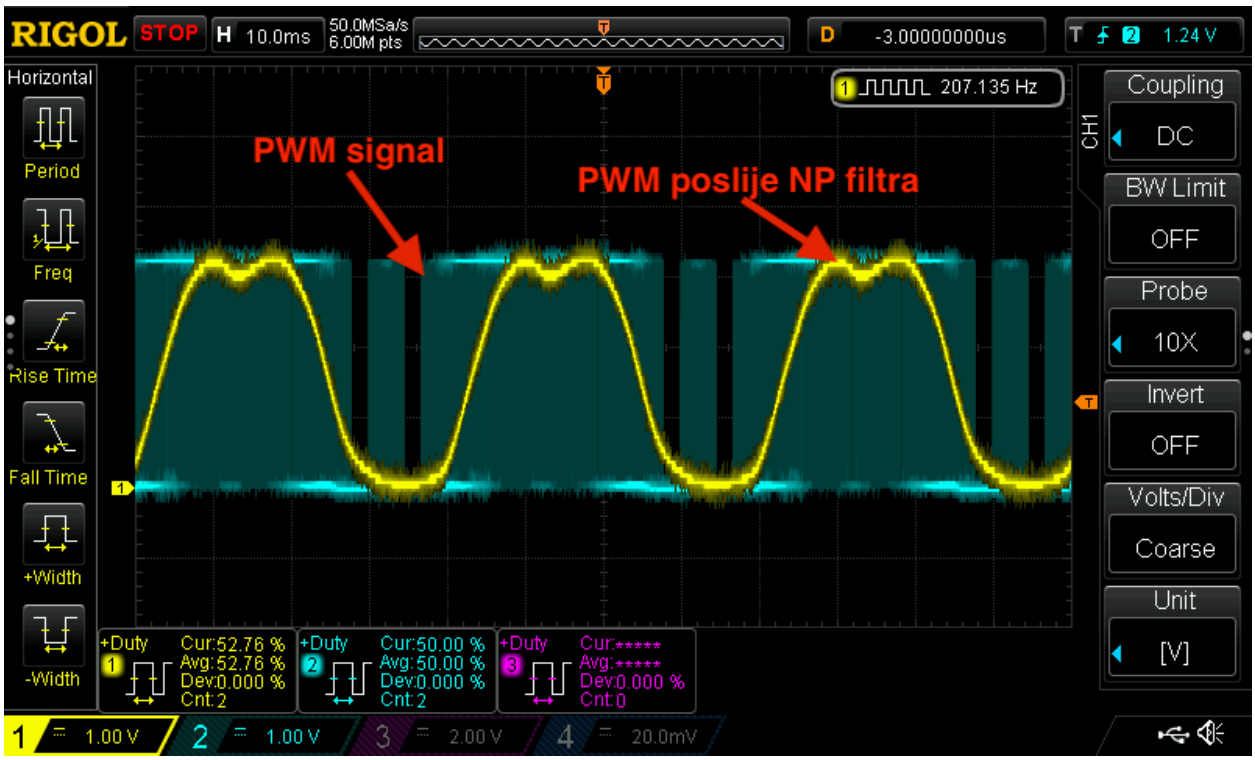
Slika 4.19. SysConfig konfiguracija pokretanja ADC pretvorbe pomoću PWM brojača.

Ispod je dana implementacija vektorske modulacije kako je prikazano izrazima (4.16) i (4.17). Proračun se vrši u per unit mjernoj jedinici tako da su naponi normalizirani na trenutno izmjereni napon DC međukruga čime se izbjegava propagiranje varijabilnosti napona napajanja na elektromotor. Kroz strukturu motora `Motor_t` implementiranoj metodi dan je parametar željenog napona u $\alpha\beta$ koordinatnom sustavu, no *engl. Zero Sequence Modulation* algoritam vektorske modulacije za proračun koristi trofazni zapis napona pa se vrši inverzna clarke transformacija. Rezultat pozivanja metode `SVGEN_run` upisivanje je željenog napona svake faze u per unit mjernoj jedinici u `PWMData_t` strukturu koja je metodi proslijeđena putem pokazivača. `PWMData_t` struktura kasnije se koristi za upisivanje odabranog faktora opterećenja u registre komparatora PWM periferne jedinice. Na slici 4.18 primjer vrijednosti faktora opterećenja označen je "F.OPT.1", "F.OPT.2".

```
// foc.h
static inline void SVGEN_run(Motor_t * m, PWMData_t * pwm)
{
    pwm->vmax_pu = 0;
    pwm->vmin_pu = 0;
    // Definiranje napona u per unit mjernoj jedinici
    pwm->va_pu = m->Vout_ab_V[0] * m->oneOverDcBus_invV;
    pwm->vbeta_pu = m->Vout_ab_V[1] * m->oneOverDcBus_invV;
    // Pomoćni izrazi za inverznu clarke transformaciju
    pwm->va_tmp = (float32_t)(0.5) * (-pwm->va_pu);
    pwm->vb_tmp = SVGEN_SQRT3_OVER_2 * pwm->vbeta_pu;
    // Inverzna clarke transformacija
    pwm->vb_pu = pwm->va_tmp + pwm->vb_tmp;
    pwm->vc_pu = pwm->va_tmp - pwm->vb_tmp;
    // Računanje zajedničkog napona
    pwm->vmax_pu = MATH_max(MATH_max(pwm->va_pu, pwm->vb_pu),
      pwm->vc_pu);
```

```
18      pwm->vmin_pu = MATH_min(MATH_min(pwm->va_pu, pwm->vb_pu),
19        pwm->vc_pu);
20      pwm->vcom_pu = (float32_t)0.5 * (pwm->vmax_pu + pwm->vmin_pu);
21      // Pribrajanje zajedničkog napona svakoj fazi
22      pwm->Vabc_pu[0] = (pwm->va_pu - pwm->vcom_pu);
23      pwm->Vabc_pu[1] = (pwm->vb_pu - pwm->vcom_pu);
24      pwm->Vabc_pu[2] = (pwm->vc_pu - pwm->vcom_pu);
25  }
```

Slika 4.20 prikazuje je izmjereni PWM signal, i filtrirani PWM signal. PWM signal filtriran je NP filtrom kako bi se uklonila frekvencija nosioca PWM signala.



Slika 4.20. Mjerenje PWM signala vektorske modulacije.

4.8 Zaključak implementacije vektorskog upravljanja na AM263x mikrokontroleru

Algoritam vektorskog upravljanja izvršava se u jednoj metodi (`FOCrun_ISR`) kojoj se izvršavanje pokreće hardverskim prekidom pomoću brojila i komparatora. Komparator je konfiguriran tako da da izvrši prekid svakih 100 µs odnosno frekvencijom od 10 kHz. Prekid je također sinkroniziran s PWM signalom i ADC pretvorbom što je spomenuto u poglavljima 4.4 i 4.7.

Prije pokretanja algoritma vektorskog upravljanja potrebno je izvršiti inicijalizaciju struktura te kalibraciju ADC pretvornika pozivom metoda `FOC_init()` i `FOC_cal()` u `trinv_main()` metodi koja se poziva pri pokretanju mikroprocesora u `main()` metodi.

```c
// trinv_main.c

void trinv_main(void *args)
{
    /* Pokreni driver-e konfigurirane SysConfig alatom */
    Drivers_open();
    Board_driversOpen();

    /* Inicijalizacija koda za vektorsko upravljanje */
    FOC_init();
    DebugP_log("foc init done \r\n");
    /* Kalibracija ADC-a */
    FOC_cal();
    DebugP_log("foc cal done \r\n");
    /**
     * Registracija prekida za izvođenje
     * vektorskog upravljanja
    */
    FOC_run();
    DebugP_log("foc run done \r\n");

    /* Inicijalizacija međuprocesorske komunikacije */
    setup_IPC();

    while (TRUE){
        // Izvršavanje komunikacije sa jezgrom za tcp komunikaciju
        processIPC();
    }

    Board_driversClose();
    Drivers_close();
}
```

Pozivom metode `FOC_init()` sve potrebne strukture za izvođenje vektorskog upravljanja inicijaliziraju se s početnim vrijednostima i konstantama. Pozivom metode `FOC_cal()`, osim izvršavanja kalibracije ADC periferne jedinice, izmjeri se i DC pomak na izvodima frekvencijskog pretvarača za mjerenje struje. DC pomak kasnije se pri izvođenju vektorskog upravljanja oduzima od svakoga mjerenja struje. Ovo ponašanje detaljnije je pojašnjeno u poglavlju 4.4.3.

Na kraju se pozove `FOC_run()` čime se pokazivač metode `FOCrun_ISR` zapiše kao adresa na kojoj se izvršava prekid čime se pokreće proces izvršavanja vektorskog upravljanja. Implementacija spomenutih inicijalizacijskih metoda dana je u prilogu 10.3.2.

Sve metode implementirane u poglavlju 4 se pozivaju u metodi `FOCrun_ISR`. Time se izvršava proces koji je blokovski prikazan na slici 4.1. Ispod je dan programski kod metode `FOCrun_ISR` čiji je opis dan u komentarima programskog koda radi preglednosti. Određeni blokovi koda već su prethodno prikazani, no cilj je poglavlja 4.8 prikazati potpunu sliku. Na kraju se nalazi poziv metode `raiseIPCTransmissionFlag(gIsrCnt)` koji je vezan uz temu poglavlja 5, a to je komunikacija među mikroprocesorima i tcp server.

```c
// FOC_loop.c

__attribute__ ((section(".tcmb_code"))) void FOCrun_ISR(void *handle)
{

    // Provjera ako je omogućen rad motora
```

```
    if(runMotor == MOTOR_RUN)
    {
        // Tražena referentna brzina postavlja se na ulaz
        // generatora rampe
        rc1.TargetValue = SpdRef;
        rampControl(&rc1);
        rg1.Freq = rc1.SetpointValue;
        rampGen(&rg1);
        // Te se izlaz generatora rampe postavlja
        // na ulaz PI regulatora brzine
        motor1.pi_spd.refValue = rg1.Freq * BASE_FREQ * TWO_PI;

    }
    // Ukoliko rad motora nije omogućen
    // Tj. ako je u bilo kojem trenutku onemogućen
    // Referentna brzina je 0 !
    else
    {
        // Resetira se ulaz u sustav, nakon gašenja motora
        // ponovo se mora poslati željena referentna brzina
        // za ponovno pokretanje motora
        SpdRef = 0;
        motor1.pi_spd.refValue = 0;
    }

    // Čitanje vrijednosti AD pretvornika za mjerenje struja
    motor1.I_abc_A[0] = (float32_t) IFBU_PPB;
    motor1.I_abc_A[1] = (float32_t) IFBV_PPB;
    // Faza W dobije se izrazom W = RET - U - V
    motor1.I_abc_A[2] = (float32_t)(IFBRET_PPB - motor1.I_abc_A[0] -
      motor1.I_abc_A[1]);

    // Skaliranje izmjerenih struja u ampere [A]
    motor1.I_abc_A[0] = motor1.I_abc_A[0] * motor1.I_scale;
    motor1.I_abc_A[1] = motor1.I_abc_A[1] * motor1.I_scale;
    motor1.I_abc_A[2] = motor1.I_abc_A[2] * motor1.I_scale;

    // Čitanje vrijednosti AD pretvornika za mjerenje
    // napona DC međukruga
    motor1.dcBus_V = (float32_t)VDC_EVT;

    // Filtriranje izmjerenog napona DC međukruga
    motor1.dcBus_V = motor1.dcBus_V * motor1.V_scale;
    motor1.dcBus_V = FILTER_FO_run(&filterVdc, motor1.dcBus_V);
    motor1.dcBus_V = (motor1.dcBus_V > 1.0) ? motor1.dcBus_V : 1.0;
    motor1.oneOverDcBus_invV = 1.0 / motor1.dcBus_V;

    // Čitanje enkodera i izračun brzine vrtnje
    PosSpeed_calculate(&posSpeed, CONFIG_EQEP2_BASE_ADDR);
    encoder_omega = posSpeed.speedPR * BASE_FREQ * TWO_PI;

    // Postavljanje izmjerene brine u povratnu vezu
    // PI regulatora brzine
    motor1.pi_spd.fbackValue = encoder_omega;
    // Izvršavanje PI regulatora brzine,
    // i postavljanje rezultata na ulaz PI regulatora Q struje
    motor1.pi_iq.refValue = PI_run_series(&(motor1.pi_spd));

    // Estimiranje brzine klizanja rotora AS motora
    aci1.IMDs += aci1.Kr * (motor1.I_dq_A[0] - aci1.IMDs);

    // Limitiranje na +/- 0.001, da se izbjegne dijeljenje s 0
    aci1.IMDs = ((aci1.IMDs <  0.001) && (aci1.IMDs >= 0)) ?
      0.001 : aci1.IMDs;
    aci1.IMDs = ((aci1.IMDs > -0.001) && (aci1.IMDs <  0)) ?
      -0.001 : aci1.IMDs;

    // Estimirana brzina klizanja
```

```
    aci1.Wslip = aci1.Kt * motor1.I_dq_A[1] / aci1.IMDs;

    // Estimirana brzina klizanja dodaje se izmjerenoj
    // brzini rotora, kako bi se dobila sinkrona kutna brzina
    motor1.omega_e = encoder_omega + aci1.Wslip;
    motor1.theta_e += motor1.sampleTime * motor1.omega_e;

    // Svođenje kuta na prvu rotaciju
    // odnosno na raspon [-2pi do 2pi]
    theta_limiter(&(motor1.theta_e));

    // Kompenziranje kašnjenja na izmjerenom kutu rotora
    // koje je uneseno duljinom trajanja izvršavanja ADC pretvorbe
    motor1.theta_e_out = motor1.theta_e +
                         (motor1.outputTimeCompDelay * encoder_omega);
    theta_limiter(&(motor1.theta_e_out));


    // Računanje sinusa i kosinusa kuta rotora
    // (priprema za park transformaciju)
    motor1.Sine = sinf(motor1.theta_e);
    motor1.Cosine = cosf(motor1.theta_e);
    motor1.Sine_out = sinf(motor1.theta_e_out);
    motor1.Cosine_out = cosf(motor1.theta_e_out);

    // Prelazak iz abc u dq sustav
    clarke_run(&motor1);
    park_run(&motor1);

    // Prilagodba za način spajanja motora
    motor1.I_dq_A[0] = -motor1.I_dq_A[0];
    motor1.I_dq_A[1] = -motor1.I_dq_A[1];

    // Ako je rad motora omogućen postavljanje referentne vrijednosti
    // na regulator struje D, ova vrijednost parametar je strukture
    motor1.pi_id.refValue = (runMotor == MOTOR_STOP) ? 0 : IdRef;

    // Definiranje limita za regulator D struje
    motor1.pi_id.outMax = motor1.vqLimit * motor1.dcBus_V;
    motor1.pi_id.outMin = - motor1.pi_id.outMax;
    motor1.pi_iq.outMax = motor1.pi_id.outMax;
    motor1.pi_iq.outMin = motor1.pi_id.outMin;
    motor1.Vout_max = motor1.pi_id.outMax;

    // Postavljanje povratne veze oba regulatora struje
    motor1.pi_id.fbackValue = motor1.I_dq_A[0];
    motor1.pi_iq.fbackValue = motor1.I_dq_A[1];
    // Izvršavanje oba regulatora struje
    // rezultat je traženi napon u dq sustavu
    motor1.Vout_dq_V[0] = PI_run_series(&(motor1.pi_id));
    motor1.Vout_dq_V[1] = PI_run_series(&(motor1.pi_iq));

    // Limitiranje izlaza regulatora struje na vrijednosti
    // konfigurirane u Motor_param.h i Trivn_param.h datotekama
    dq_limiter_run(&motor1);

    // Povratak u mirujući sustav
    ipark_run(&motor1);
    // Vektorska modulacija
    SVGEN_run(&motor1, &pwm1);
    PWM_clamp(&pwm1);

    // Zapisivanje rezultata vektorske modulacije
    // u registre PWM periferne jedinice
    // Ovim korakom traženi napon se narine na motor
    TRINV_HAL_setPwmOutput(&pwm1);

    gIsrCnt+=1;
```

```
143     if (gIsrCnt >= ULONG_MAX - 1)
144     {
145         gIsrCnt = 0;
146     }
147     // Postavi flag da je izvršen jedan period
148     // u svrhu slanja mjerenja na tcp server jezgru
149     raiseIPCTransmissionFlag(gIsrCnt);
150 
151     ADC_clearInterruptStatus(CONFIG_ADC4_BASE_ADDR,ADC_INT_NUMBER1);
152 }
```

5 INTEGRIRANO WEB SUČELJE

Kako bi se olakšalo upravljanje elektromotorom, izrađena je web aplikacija koja nudi mogućnost detaljnog očitanja i vizualizacije svih parametara i mjerenja koje vrši program izvođenja vektorskog upravljanja. Web sučelje također nudi mogućnost mijenjanja parametara regulatora u stvarnom vremenu te upravljanje brzinom i pobudom elektromotora.

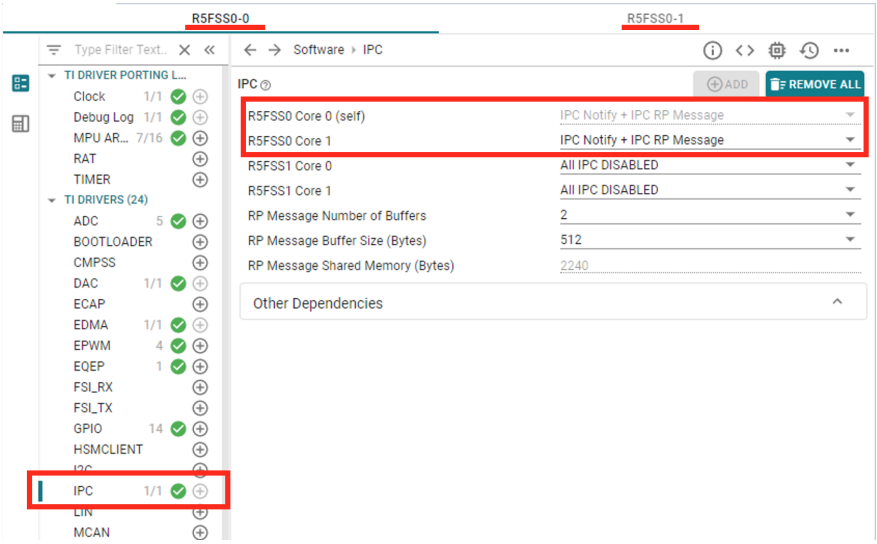
Za rad web aplikacije potrebna je dvosmjerna komunikacija s mikrokontrolerom te kako se radi o web aplikaciji, najpogodnija metoda komunikacije je putem Ethernet periferne jedinice. Ethernet periferna jedinica nudi rješenje za samo prva dva sloja OSI (*eng. Open Systems Interconnection*) modela, odnosno pvi sloj koji se još naziva *engl. Physical Layer* i drugi sloj koji se još naziva *engl. Data Link Layer* u kojem se adresiranje vrši putem MAC (*engl. Media Access Control*) adrese. Sve slojeve iznad ova dva kao što je *engl. Internet Protocol* IP i *engl. Transmission Control Protocol* TCPpotrebno je implementirati u programu. U radu se ne ulazi u duboku analizu ovih protokola jer njihovo detaljno poznavanje nije potrebno, već se koriste po principu crne kutije.

5.1 Među procesorska komunikacija

Mikrokontroler AM2634 korišten u radu sastoji se od četiri mikroprocesora. Iz tog razloga je program za komunikaciju s web aplikacijom implementiran na zasebnom mikroprocesoru kako bi onaj na kojemu se izvodi algoritam vektorskog upravljanja mogao nesmetano obavljati svoj zadatak. Ova tehnika sa sobom nosi potrebu za razmjenom podataka između dva paralelna procesa izvođenja programa. Komunikacija je izvedena pisanjem i čitanjem podataka iz zajedničke memorije. Kako postupak međuprocesorske komunikacije nije trivijalno implementirati na korektan način Texas Instruments nudi knjižnicu s već implementiranim sustavom međuprocesorske komunikacije *IPC* (*engl. Inter Process Communication*).

IPC knjižnica na raspolaganje daje metode kojima se na svakome procesoru definira jedan ili više *engl. endpoint* na koji se može slati tj. zaprimati podate. *Endpoint* je ključ koji se mora poslati uz poruku kako bi s naznačilo u kojem dijelu koda tu poruku treba konzumirati. Implementacija web sučelja koristi samo po jedan endpoint na svakome mikroprocesoru, a onda se u slučaju slanja komandi prema jezgri koja izvršava vektorsko upravljanje neovisno o IPC knjižnici delegira radnja ovisno o sadržaju poruke.

Korištenje ove knjižnice odnosno driver-a potrebno je konfigurirati SysConfig alatom što je prikazano slikom 5.1.



Slika 5.1. SysConfig konfiguracija međuprocesorske komunikacije.

Na vrhu slike može se primijetiti da su dostupna dva mikroprocesora. Kako bi se koristila dva mikroprocesora u CCS-u (Code Composer Studio), potrebno je za svaki mikroprocesor izraditi zaseban projekt te ih povezati pomoću trećeg sistemskog projekta. Detaljne upute kako povezati projekte može se pronaći na ovoj poveznici: https://dev.ti.com/tirex/content/AM26x_Academy_2_00_00_00/ _build/source/sdk_fundamentals/multicore_application.html.

Kao način komunikacije na slici 5.1 odabrana je opcija *"IPC Notify + IPC RPC Message"* čime se omogućuje slanje poruka veličine do 512bita. Za jednostavnu komunikaciju dovoljno je koristiti samo *IPC Notify* čime je jednom porukom moguće poslati 16bita.

Dalje će u tekstu na početku blokova koda u komentaru pisati iz kojega je projekta mikroprocesora (Mikroprocesor za vektorsko upravljanje ili Mikroprocesor za komunikaciju). Dalje je dan inicijalizacijski kod za mikroprocesor na kojemu se izvršava vektorsko upravljanje.

```
// Mikroprocesor za vektorsko upravljanje
// IpcComm/IPC_RPC_Comm.c

// ID endpoint-a na kojemu će se slušati za poruke
#define RECEVE_ENDPOINT 10
// ID endpoint-a TCP server jezgre, tu šaljemo podatke
#define TCP_STREAM_ENDPOINT 12
// ID mikroprocesora na koji se šalju poruke
#define TCP_SERVER_CORE_ID CSL_CORE_ID_R5FSS0_1

.
.
.

void setup_IPC()
{
    // Inicijalizacija međuprocesorske komunikacije
    RPMessage_CreateParams createParams;
    RPMessage_CreateParams_init(&createParams);
    createParams.localEndPt = RECEVE_ENDPOINT;
    RPMessage_construct(&gRecvMsgObject, &createParams);

```

```
    // Zastavica za iniciranje slanja paketa
    SendNewIPCRPPacket = 0;
    // Vrijem paketa koji se šalje
    packetTime = 0;
}
```

Ispod je konfiguracijski kod za komunikacijski mikroprocesor.

```
// Mikroprocesor za komunikaciju
// Tcp_Server_IPC.c

// ID Endpoint-a na koji se dobivaju poruke
#define RECEVE_ENDPOINT 12

void setup_IPC()
{
    RPMessage_CreateParams createParams;
    RPMessage_CreateParams_init(&createParams);
    createParams.localEndPt = RECEVE_ENDPOINT;
    RPMessage_construct(&gAckReplyMsgObject, &createParams);
}
```

Podatke mjerenja prije slanja potrebno je sakupiti u pravim trenutcima, zatim serijalizirati i pohraniti na memorijsku lokaciju dostupnu mikroprocesoru za komunikaciju. Serijalizacija je proces kojim se više varijabli u memoriju spreme jedna iza druge poznatim redoslijedom kako bi se isti podatci mogli na drugom mikroprocesoru pročitati iz memorije uz očuvanu informaciju o svrsi pojedine varijable. Za sve odabrane podatke koji će se slati na web sučelje definirana je C struktura `InverterStreamPacket_t` kako bi odabrani podatci i njihova struktura bila jasno čitljiva u programskom kodu.

```
// Mikroprocesor za vektorsko upravljanje
// Tcp_Server_IPC.h
typedef struct __InverterStreamPacket {
    // Trenutak vremena u kojemu su očitane vrijednosti
    uint32_t IsrTick;
    // Mjerene struje
    float32_t Ia;
    float32_t Ib;
    float32_t Ic;
    // Napon napajanja
    float32_t DcBus;
    // D i Q komponente mjerene struje
    float32_t Id;
    float32_t Iq;
    // Mjereni kut i brzina rotora
    float32_t theta_e;
    float32_t omega_e;
    // Izlazni napon u dq sustavu
    float32_t Out_Vd;
    float32_t Out_Vq;
    // Regulator brzine
    float32_t RegSpeed_Fback;
    float32_t RegSpeed_Output;
    // Regulator struje Id
    float32_t RegId_Fback;
    float32_t RegId_Output;
    // Regulator struje Iq
    float32_t RegIq_Fback;
    float32_t RegIq_Output;
    // Mjerenja s enkodera
    float32_t EncoderTheta;
```

```
    float32_t EncoderOmega;
    // Referentna brzina normalizirana na sinkronu brzinu
    float32_t SpeedRef;
    // Status motora
    unsigned short MotorRunStop;
    // Regulator brzine
    float32_t RegSpeed_RefVal;
    float32_t RegSpeed_Kp;
    float32_t RegSpeed_Ki;
    // Regulator struje Id
    float32_t RegId_RefVal;
    float32_t RegId_Kp;
    float32_t RegId_Ki;
    // Regulator struje Iq
    float32_t RegIq_RefVal;
    float32_t RegIq_Kp;
    float32_t RegIq_Ki;
} InverterStreamPacket_t;
```

Zatim su implementirane metode kojima se dohvaćaju pokazivači na strukture koje sadrže željene podatke. Način pristupa podataka nije potrebno implementirati pomoću pomoćnih metoda, već se moglo definicije struktura postaviti u .h datoteku te ih time učiniti dostupnima u svim datotekama programskog koda. Strukture s podatcima usko su povezane uz vektorsko upravljanje. Modificiranjem njihovih vrijednosti može u potpunosti onesposobiti izvođenje algoritma pa je odabrana metoda pristupa sa *engl. getter* metodama kako bi se do pokazivača došlo eksplicitno. Na primjer, interakcija sa statusom motora i referentne brzine izvedena je isključivo preko *engl. getter* i *engl. setter* metoda koje sadrže zaštitni programski kod. Ispod je dan programski kod *engl. getter* i *engl. setter* metoda.

```
// Mikroprocesor za vektorsko upravljanje
// FOC_loop.c

Motor_t* FOC_DANGER_getMotorStructPointer() {
    return &motor1;
}
PosSpeed_Object* FOC_DANGER_getPosSpeedStructPointer() {
    return &posSpeed;
}
float32_t FOC_getSpeedRef(void) {
    return SpdRef;
}
unsigned int FOC_getMotorRunState() {
    return runMotor;
}
void FOC_setMotorRunState(MotorRunStop_e state) {
    if (state == MOTOR_RUN)
    {
        runMotor = MOTOR_RUN;
        return;
    }
    runMotor = MOTOR_STOP;
}
void FOC_setSpeedRef(float32_t SpeedSetpoint) {
    // Limit na raspon od 0 do 1
    SpdRef = fminf(1.0, fmaxf(SpeedSetpoint, -1.0));
}
```

Slanje podatka sinkronizirano je s njihovom akvizicijom tako što se na kraju svake iteracije izvođenja vektorskog upravljanja obavijesti dio programskog koda koji je zadužen za pripremu i slanje podataka da je akvizicija završila i da su podatci spremni za slanje. Također, kako bi podatci koji se šalju bili korisni, potrebno im je dodijeliti vremensku os. Podatak o vremenu mjerenja dobiva se brojačem ciklusa izvođenja vektorskog upravljanja. Kako se proces izvođenja vektorskog upravljanja odvija poznatom brzinom, moguće je u odnosu na vrijednost brojača raditi frekvencijsku i vremensku analizu nad podatcima. Brojač i signalizacija završenog ciklusa nalazi se na kraju metode `FOCrun_ISR` čiji je relevantan dio programskog koda dan je ispod.

```
// Mikroprocesor za vektorsko upravljanje
// FOC_loop.c

__attribute__ ((section(".tcmb_code"))) void FOCrun_ISR(void *handle)
{
  .
  .

    gIsrCnt+=1;
    if (gIsrCnt >= ULONG_MAX - 1)
    {
        gIsrCnt = 0;
    }
    // Postavi flag da je izvršen jedan period
    // u svrhu slanja mjerenja na tcp server jezgru
    raiseIPCTransmissionFlag(gIsrCnt);
}
```

Slanje paketa podataka za svaki akvizicijski ciklus ne mora biti potrebno pa je implementiran jednostavan mehanizam preskakanja fiksnog broja ciklusa. Implementacija metode `raiseIPCTransmissionFlag` u kojoj se broji koliko puta je ista pozvana dana je ispod. Ako je broj poziva metode premašio konfigurirani broj, brojač se postavlja na nulu i signalizira slanje trenutnog paketa.

```
// Mikroprocesor za vektorsko upravljanje
// IPC_RPC_Comm.c

void raiseIPCTransmissionFlag(uint32_t isrTick)
{
    skipPacketsCounter++;
    if (skipPacketsCounter >= PACKETS_TO_SKIP) {
        skipPacketsCounter = 0;
        SendNewIPCRPPacket = 1;
        packetTime = isrTick;
    }
}
```

Programski kod koji je zadužen za pripremu i slanje podataka izvršava se maksimalnom preostalom brzinom procesora pored izvođenja vektorskog upravljanja. Izvršava se u beskonačnoj petlji glavne grane programa koja je pokrenuta nakon inicijalizacijskog procesa, izvođenje vektorskog upravljanja ciklično se pokreće prekidima. Ispod je dan programski kod u kojemu je prikazana beskonačna petlja koja izvršava program za komunikaciju.

```
// Mikroprocesor za vektorsko upravljanje
// trinv_main.c

void trinv_main(void *args)
{
  ...

    /* Inicijalizacija međuprocesorske komunikacije */
    setup_IPC();

    while (TRUE){
        // Izvršavanje komunikacije sa jezgrom za tcp komunikaciju
        processIPC();

    }
  ...
}
```

Metoda `setup_IPC()` već je prethodno spomenuta, a relevantni dio metode `processIPC()` dan je ispod. Ostatak programskog koda metode vezan je uz primanje poruka pa je prikazan u poglavlju 5.2. U metodi `processIPC()` vrši se provjera zastavice treba li poslati paket. Ukoliko treba, poziva se metoda `IPC_SendInverterDataPacket()` koja sadrži logiku slanja paketa.

```
// Mikroprocesor za vektorsko upravljanje
// IPC_RPC_Comm.c

void processIPC()
{
    if (SendNewIPCRPPacket == 1) {
        // Resetiranje zastavice za slanje paketa
        SendNewIPCRPPacket = 0;
        // Priprema podataka i slanje paketa
        IPC_SendInverterDataPacket();
    }

  ...
}
```

Metoda `IPC_SendInverterDataPacket()` u strukturu paketa `InverterStreamPacket_t` upiše sve vrijednosti pomoćnom metodom `PopulateInverterStreamPacket()` koja je zbog preglednosti dana u prilogu 10.3.3. U memoriji se zatim alocira lista bajtova dovoljne veličine za pohranu svih podataka. Pomoću metode `SerialiseInverterStreamPacket()` na alocirani memorijski prostor zapišu se podatci u serijaliziranom obliku. Zatim se poziva metoda koja paket podataka pošalje mikroprocesoru za komunikaciju. Slanje se izvodi tako da se metodi kao argument daje pokazivač na početak liste u kojoj je serijalizirani paket podataka te duljina liste u bajtovima. Memorijski blok koji sadrži podatke zatim se kopira u zajednički dio memorije kojemu drugi mikroprocesor ima pristup.

```
// Mikroprocesor za vektorsko upravljanje
// IPC_RPC_Comm.c

void IPC_SendInverterDataPacket()
{
    InverterStreamPacket_t packet;
    // Zapisivanje vrijednosti u strukturu paketa
    PopulateInverterStreamPacket(&packet, packetTime);

```

```
    // Serijaliziranje paketa
    char serialisedPacket[sizeof(packet)] = {0};
    SerialiseInverterStreamPacket(&packet, serialisedPacket);

    // Slanje paketa mikroprocesoru za komunikaciju
    IPC_SendMessage(serialisedPacket, sizeof(serialisedPacket));
}
```

Prvih nekoliko linija koda metode `SerialiseInverterStreamPacket()` dano je ispod, ostatak linija koda se ponavlja za ostala polja strukture. Kopiranje svakoga polja strukture u listu bajtova izvedeno je pomoću *engl. macro* definicije `SERIALIZE_FIELD()`.

```
// Mikroprocesor za vektorsko upravljanje
// IPC_RPC_Comm.c

#define SERIALIZE_FIELD(bufferPtr, structPtr, field) \
    memcpy(bufferPtr, &(structPtr->field), sizeof(structPtr->field)); \
    buffer += sizeof(structPtr->field);

void SerialiseInverterStreamPacket
(const InverterStreamPacket_t* packet, char buffer[])
{
  SERIALIZE_FIELD(buffer, packet, IsrTick);

  SERIALIZE_FIELD(buffer, packet, Ia);
  SERIALIZE_FIELD(buffer, packet, Ib);
  SERIALIZE_FIELD(buffer, packet, Ic);
  . . .
}
```

5.2 TCP server

Texas instruments nudi na raspolaganju primjer projekta u kojemu je implementiran TCP server pomoću *LwIP* biblioteke. *LwIP* biblioteka najkorištenija je implementacija IP i TCP protokola u sustavima mikrokontrolera. Više informacija o ovoj biblioteci dostupno je na sljedećoj adresi: https://savannah.nongnu.org/projects/lwip. U projektu je TCP server implementiran na *FreeRTOS* operativnom sustavu kako bi bilo jednostavnije i jasnije organizirati procese. Implementacija web sučelja za rad zahtijeva najmanje dvije paralelne TCP konekcije. Operativni sustav razvojnom inženjeru apstrahira proces odvajanja vremena za više procesa koji se izvršavaju na jednom mikroprocesoru pa pri razvoju može razmišljati na način kao da se procesi zaista izvršavaju paralelno. Kada se govori o izvođenju paralelnih radnji u računalstvu često se koristi pojam *engl. thread* ili na hrvatskom nit. Dalje u tekstu radi jasnoće koristit će se engleski naziv "thread".

Inicijalizacija servera sastoji se od registriranja glavnoga procesa programa `freertos_main()` te pokretanja *FreeRTOS* planera procesa kojim se započinje izvršavati `freertos_main()` metoda. Programski kod inicijalizacije dan je ispod.

```
// Mikroprocesor za komunikaciju
// main.c

void freertos_main(void *args)
{
    appMain(NULL);

    vTaskDelete(NULL);
}


int main(void)
{
  // Inicijalizacija SysConfig konfiguracije
  System_init();
  Board_init();

  // Registriranje metode freertos_main
  gMainTask = xTaskCreateStatic( freertos_main,
                  "freertos_main",
                  MAIN_TASK_SIZE,
                  NULL,
                  MAIN_TASK_PRI,
                  gMainTaskStack,
                  &gMainTaskObj );
  configASSERT(gMainTask != NULL);

  // Pokretanje planera procesa
  vTaskStartScheduler();

.
.
.
```

`freertos_main()` izvodi se u glavnome thread-u te pozivom metode `appMain()` inicijalizira Ethernet perifernu jedinicu i LwIP biblioteku. Nakon inicijalizacije i uspješno uspostavljene veze s mrežom poziva se metoda `AppTcp_startServer()` kojoj je jedina zadaća pokrenuti novi thread koji izvršava funkciju tcp servera. Programski kod `AppTcp_startServer()` metode dan je ispod.

```
// Mikroprocesor za komunikaciju
// app_tcpserver.c

void AppTcp_startServer()
{
    sys_thread_new("AppTcp_ServerTask", AppTcp_ServerTask, NULL,
    DEFAULT_THREAD_STACKSIZE, DEFAULT_THREAD_PRIO);
}
```

Pokrenuti thread izvršava metodu `AppTcp_ServerTask()` koja vrši zadaću prihvaćanja TCP konekcija od klijenata, spremanje stanja konekcije u memoriju te pokretanje novih thread-ova davajući im pokazivač na strukturu konekcije u memoriji. Novi thread onda čita podatke poslane unutar te konekcije i vrši zadaću slanja ili primanja podataka sa web sučelja. Programski kod koji pokreće nove thread-ove za obradu konekcija dan je ispod, a detaljan opis nalazi se u komentarima programskog koda.

```
// Mikroprocesor za komunikaciju
// app_tcpserver.c

static void AppTcp_ServerTask(void *arg)
{
    // Inicijaliziranje strukture za spremanje stanja konekcije
    // ova struktura koristi se za slušanje na nove konekcije
    struct netconn *pConn = NULL;
    err_t err;
    LWIP_UNUSED_ARG(arg);

    // Konekciji se dodjeljuje satična IP adresa i PORT
    pConn = netconn_new(NETCONN_TCP);
    netconn_bind(pConn, IP_ADDR_ANY, APP_SERVER_PORT);
    LWIP_ERROR("tcpecho: invalid conn\r\n", (pConn != NULL), return;);

    // Konekcija se postavlja u stanje slušanja
    netconn_listen(pConn);

    while (1)
    {
        // Alociranje memorije za novu konekciju
        struct netconn *newClientConnection = (struct netconn*)
      pvPortMalloc(sizeof(struct netconn));

        // Čekanje nove konekcije
        // Ova metoda "blokira" izvođenje koda
        // dok se ne pojavi nova konekcija
        err = netconn_accept(pConn, &newClientConnection);
        printf("accepted new connection %p\r\n", newClientConnection);


        if (err < ERR_OK)
        {
            DebugP_log(
              "Unable to accept connection: errno %d\r\n", err
          );

            // Ako je došlo do greške oslobađa se
            // alocirana memorija konekcije
            vPortFree(newClientConnection);
            continue;
        }

        // Pokretanje novog threada koji obradi konekciju
        // Pokazivač na stanje konekcije šalje se kao argument
        // Thread ima zadaću osloboditi memoriju konekcije kada ju
        // zatvori!
        sys_thread_new("hTcp", HandleTcpConnection,
        newClientConnection,
        DEFAULT_THREAD_STACKSIZE,
        DEFAULT_THREAD_PRIO);
    }

    printf("Closed listen connection %p\r\n", pConn);

    netconn_close(pConn);
    netconn_delete(pConn);

}
```

Thread koji `AppTcp_ServerTask()` metoda pokrene izvršava metodu *HandleTcpConnection()*. Kako je cilj servera komunikacija s internetskim preglednikom koji izvršava klijentsku aplikaciju, potrebno je implementirati HTTP *engl. Hypertext Transfer Protocol*. HTTP se imple-

mentira na TCP konekcijama te je u potpunosti baziran na slanje podataka u obliku teksta. Za željeni rad web sučelja potrebno je implementirati mali dio HTTP protokola koji je dovoljan za ostvarenje potrebne komunikacije. Dovoljno je čitati samo prvi redak HTTP paketa u kojemu se nalazi naziv metode HTTP upita i adresa HTTP upita. Metoda *HandleTcpConnection()* vrši upravo radnju čitanja prvog retka HTTP paketa te, ovisno o adresi upita, poziva metodu za slanje podataka mjerenja ili metodu za izmjenu parametara upravljačke strukture. Ovaj proces često se naziva *engl. Routing*, a metoda koja vrši tu radnju *engl. Router*.

Ispod je dan programski kod metode *HandleTcpConnection()*. Tekst upita uspoređuje se s dva stringa. Ako HTTP paket počinje tekstom `"GET /stream"` pozove se metoda koja će započeti slanje podataka s invertera, a ako paket počinje tekstom `"GET "` poziva se metoda koja dalje nastavlja čitati tekst upita i na osnovu njega vrši izmjenu parametara regulacijske strukture.

```
// Mikroprocesor za komunikaciju
// app_tcpserver.c

static void HandleTcpConnection(void *arg)
{
    struct netconn *clientConnection = (struct netconn*) arg;
    struct netbuf *buf;
    void *data;
    u16_t len;
    err_t err;

    err = netconn_recv(clientConnection, &buf);
    if (err == ERR_OK) {
        // Čitanje poslanih podataka
        netbuf_data(buf, &data, &len);
        char *reqString = (char *)data;

        // Odabir metoda koja će odgovoriti na HTTP upit
        if (strncmp(reqString, "GET /stream", 11) == 0) {
            StreamInverterData(clientConnection);
        } else if (strncmp(reqString, "GET ", 4) == 0) {
            SendCommandToInverter(clientConnection, reqString);
        }
    }

    printf("HandleTcpConnection: closed client connection %p\r\n", clientConnection);
    // Po završetku izvođenja zatvaranje konekcije
    netconn_close(clientConnection);
    netconn_delete(clientConnection);

    // Oslobađanje memorije u kojoj je bila spremljena konekcija
    vPortFree(clientConnection);
    // Brisanje threada iz upravljača procesima
    vTaskDelete(NULL);
}
```

Metoda `StreamInverterData()` vrši kontinuirano slanje podataka sa invertera. Radi kompatibilnosti s već dostupnim komunikacijskim protokolima koje podržava browser, podatci se šalju protokolom *engl. Server Sent Events* (dalje u tekstu SSE) o kojemu se može naći više informacija na slijedećoj poveznici: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent _events. SSE protokol osmišljen je za jednosmjernu postojanu komunikaciju od servera prema klijentu te

se implementira na HTTP protokolu. Kada internetski preglednik pošalje upit serveru za podatcima koji se šalju SSE protokolom, očekuje točno specificirano zaglavlje HTTP paketa u odgovoru. Specifikacija zaglavlja definirana je SSE protokolom i njom se internetskom pregledniku signalizira da je server u mogućnosti uspostaviti traženu vezu. Internetski preglednik nakon prvoga odgovora servera ne zatvara konekciju, već čeka pravilno formatirane poruke. Ispod je dan programski kod metode `StreamInverterData()`.

```
// Mikroprocesor za komunikaciju
// app_tcpserver.c

static void StreamInverterData(struct netconn *pClientConn)
{
    err_t err;

    // Slanje HTTP odgovora sa zaglavljem da je ovo "text/event-stream"
    err = netconn_write(pClientConn, APP_CLIENT_TX_HEADER,
      sizeof(APP_CLIENT_TX_HEADER)-1, NETCONN_COPY);
    if (err != ERR_OK)
    {
        printf("tcpecho: netconn_write: error \"%s\"\r\n",
        lwip_strerr(err));
        return;
    }

    do
    {
        char messageBuffer[sizeof(InverterStreamPacket_t)] = {0};
        uint16_t replyMsgSize = sizeof(InverterStreamPacket_t);
        uint16_t remoteCoreId, remoteCoreEndPt;

        // Prihvati poruku drugoga mikroprocesora
        int32_t status = RPMessage_recv(
            &gAckReplyMsgObject,
            messageBuffer,
            &replyMsgSize,
            &remoteCoreId,
            &remoteCoreEndPt,
            SystemP_WAIT_FOREVER);

        if (status != SystemP_SUCCESS) continue;

        // Pretvorba podataka u tekst odnosno CSV
        char stringMessage[512];
        int stringLen = stringifyInverterPacket(messageBuffer,
        stringMessage);

        // Slanje poruke
        err = netconn_write(pClientConn, stringMessage, stringLen,
        NETCONN_COPY);
        if (err != ERR_OK)
        {
            printf("tcpecho: netconn_write: error \"%s\"\r\n",
          lwip_strerr(err));
            break;
        }

    } while (1);
}
```

Za uspostavu veze prvo se šalje HTTP zaglavlje u kojemu je postavljena vrijednost ključa `Content-Type` na *"text/event-stream"*. To je ključno za pravilno započinjanje SSE prijenosa po-

dataka. Zatim se pokreće beskonačna petlja unutar koje se poziva metoda kojom se primaju poruke poslane s drugoga mikroprocesora `RPMessage_recv()`. Poziv metode `RPMessage_recv()` zaustavlja izvršavanje `while` petlje dok drugi mikroprocesor ne pošalje poruku što znači da je brzina slanja podataka definirana mikroprocesorom za vektorsko upravljanje. Primljena poruka sadrži serijaliziranu strukturu podataka `InverterStreamPacket_t` koji se pozivom metode *stringifyInverterPacket()* pretvaraju u tekstualni format gdje je svaka vrijednost varijable odvojena zarezom (*engl. Comma Separated Values (CSV)*). Također se na početak dodaje tekst "*data:* " te na kraj dva prazna retka čime se ispunjava zahtjev formata poruke koji nalaže SSE protokol. Na kraju se formatirani tekst pošalje klijentu pozivom metode `netconn_write()`.

Ispod je dan primjer jedne poruke koja se šalje SSE protokolom.

```
"data: 3631740,0.166389,0.247062,-0.428577,329.744324,...\n\n"
```

Sada će se razmotriti rad metode `SendCommandToInverter()`.

Promjena parametara regulacije kao što je struja magnetiziranja ili željena brzina implementirana je metodom `SendCommandToInverter()`. Proizvoljno je odabrano da se prilikom slanja komandi odabere HTTP metoda "GET". Format komande proizvoljno je odabran da počinje sa znakom "/", nakon kojega dolazi naziv komande koji završava znakom "?" poslije kojega se nalazi vrijednost komande zapisana u obliku decimalnog broja. Ispod je dan primjer prvog reda HTTP paketa za postavljanje željene brzine vrtnje motora na $0{,}75$:

```
"GET /N_ref?0.75"
```

Ispod je dana pomoćna metoda `getEndpointID()` koja tekstualni identifikator komande pretvara u brojčani kako bi se informacija efikasnije poslala na mikroprocesor za vektorsko upravljanje.

```
// Mikroprocesor za komunikaciju
// app_tcpserver.c

// Helper za mapiranje string endpointa u broj radi efikasnijeg prijenosa na drugu jezgru
#define MAP_ENDPOINT_TO_ID(LEN, ENDP, ID) else if (strncmp(requestString, ENDP, LEN) == 0) return ID;
int getEndpointID(char *requestString)
{
    if (strncmp(requestString, "GET /MotEn", 10) == 0) return 1;
    MAP_ENDPOINT_TO_ID(11, "GET /Id_ref", 2)
    MAP_ENDPOINT_TO_ID(10, "GET /N_ref", 3)
    MAP_ENDPOINT_TO_ID(10, "GET /SpdKp", 4)
    MAP_ENDPOINT_TO_ID(10, "GET /SpdKi", 5)
    MAP_ENDPOINT_TO_ID(9, "GET /IdKp", 6)
    MAP_ENDPOINT_TO_ID(9, "GET /IdKi", 7)
    MAP_ENDPOINT_TO_ID(9, "GET /IqKp", 8)
    MAP_ENDPOINT_TO_ID(9, "GET /IqKi", 9)
    MAP_ENDPOINT_TO_ID(11, "GET /ackPer", 10)
    return -1;
}
```

Ispod je dana metoda `extractFloat()` koja iz teksta upita očita brojčanu vrijednost komande.

```
// Mikroprocesor za komunikaciju
// app_tcpserver.c

err_t extractFloat(const char* http_packet, float32_t *result) {
    const char* start = strchr(http_packet, '?');

    if (start != NULL) {
        // Stavi pointer na prvi znak poslije '?'
        start++;

        char* end;
        *result = strtod(start, &end);

        if (start == end) {
            // Neuspiješna konverzija floata
            return -1;
        } else {
            return 0;
        }
    } else {
        // Nije pronađena lokacija znaka '?'
        return -1;
    }
}
```

Ispod je dan programski kod metode `SendCommandToInverter()` u kojoj se pozivaju metoda za prijevod tekstualne komande u brojčanu i pretvorbu tekstualnog zapisa decimalnog broja u tip varijable `float32_t`. Brojčana komanda i vrijednost komande zatim se šalju na mikroprocesor za vektorsko upravljanje.

```
// Mikroprocesor za komunikaciju
// app_tcpserver.c

static void SendCommandToInverter(struct netconn *clientConnection, char *
    requestString)
{
    err_t err;
    float32_t requestValue;

  // Čitanje brojčane vrijednosti upita
    err = extractFloat(requestString, &requestValue);
    if ( err == -1 ) {
      // Odgovori klijentu da je upit neuspješan
        netconn_write(clientConnection, HTTP_BADREQ_RESPONSE,
        sizeof(HTTP_BADREQ_RESPONSE)-1, NETCONN_COPY);
        return;
    }

    // Čitanje traženog parametra upita
    int requestEndpoint = getEndpointID(requestString);
    if ( requestEndpoint == -1 ) {
      // Odgovori klijentu da je upit neuspješan
        netconn_write(clientConnection, HTTP_BADREQ_RESPONSE,
        sizeof(HTTP_BADREQ_RESPONSE)-1, NETCONN_COPY);
        return;
    }

    // Formiranje i serijaliziranje komandnog paketa
    CommandPacket commandPacket = { requestEndpoint, requestValue };
    char serialisedPacket[sizeof(commandPacket)];
    char *bufferPtr = serialisedPacket;
```

```
    SERIALIZE_FIELD(bufferPtr, commandPacket, endpoint);
    SERIALIZE_FIELD(bufferPtr, commandPacket, value);

    // Slanje paketa jezgri za vektorsko upravljanje
    RPMessage_send(
            &serialisedPacket, sizeof(serialisedPacket),
            INVERTER_CORE_ID, INVERTER_ENDPOINT,
            RPMessage_getLocalEndPt(&gAckReplyMsgObject),
            50);

    // Odgovor klijentu da je proces uspješan
    netconn_write(clientConnection, HTTP_OK_RESPONSE,
      sizeof(HTTP_OK_RESPONSE)-1, NETCONN_COPY);
}
```

Ispod je prikazan dio programskog koda koji zahtjev za izmjenom parametra obradi na mikroprocesoru za vektorsko upravljanje. Poruka se zaprimi, te zatim deserijalizira nazad u `CommandPacket` strukturu te se provjerom broja komande izvrši izmjena traženog parametra.

```
// Mikroprocesor za vektorsko upravljanje
// IPC_RPC_Comm.c

void processIPC()
{
  .
  .
  .

    // Primanje paketa
    int32_t status = RPMessage_recv(
        &gRecvMsgObject,
        receveBuffer,
        &replyMsgSize,
        &remoteCoreId,
        &remoteCoreEndPt,
        1);

    // Deserijalizacija paketa
    memcpy(&(commandPacket.endpoint), receveBuffer,
      sizeof(commandPacket.endpoint));
    memcpy(&(commandPacket.value), receveBuffer +
      sizeof(commandPacket.endpoint), sizeof(commandPacket.value));

    // Ne radi ništa ako nije prepoznat endpoint
    if ( commandPacket.endpoint == -1) {
        return;
    }

    // MotorEnable
    if ( commandPacket.endpoint == 1) {
        if (commandPacket.value > 0.98 && commandPacket.value < 1.02) {
            FOC_setMotorRunState(1);
        }
        else {
            FOC_setMotorRunState(0);
        }
    }
    // Id_ref
    else if ( commandPacket.endpoint == 2) {
        FOC_setIdref(commandPacket.value);
    }
}
```

5.3 Typescript web aplikacija

Upravljačka konzola izrađena je u obliku web aplikacije kako bi se u budućnosti ista aplikacija mogla ugraditi u program mikrokontrolera. To bi omogućilo da mikroprocesor koji vrši dužnost TCP servera ujedno može klijentu poslati cijeli paket HTML, CSS i JavaScript programskog koda od kojega se sastoji web aplikacija koju internetski preglednik pokreće. U radu je web aplikacija zapakirana u *Electron* okruženje koje omogućuje distribuciju web aplikacija u istome obliku kako se distribuiraju tradicionalne računalne aplikacije. To znači da se web sučelje poslužuje s računala koje ga pokreće, a ne sa servera ili mikrokontrolera.

Za izradu web sučelja korišten je programski jezik TypeScript koji je nadskup JavaScript programskog jezika što znači da je svaki JavaScript kod valjan TypeScript kod, no ne i obrnuto. Kako internetski preglednici podržavaju isključivo JavaScript programski jezik, TypeScript kod prevodi se u JavaScript kod pomoću prevodioca. Također je za razvoj odabran *Vue* aplikacijski okvir koji daje korisne alate za izradu web aplikacija koje se ponašaju kao standardne računalne aplikacije. *Vue* okvir omogućuje razvoj web aplikacija u obliku komponenti koje se mogu ponovno koristiti te se na taj način može izbjeći ponavljanje koda, također omogućuje jednostavno povezivanje podataka između komponenti zbog čega je odabran za izradu web sučelja.

Više o spomenutim tehnologijama može se pronaći na slijedećim poveznicama:

- **TypeScript** https://www.typescriptlang.org/
- **Vue** https://vuejs.org/
- **Electron** https://www.electronjs.org/

Kako ideja rada nije razvoj dizajna web sučelja, dalje će biti prikazani samo dijelovi programskog koda koji su zaslužni za komunikaciju s TCP serverom i obradu podataka. Potpuni programski kod dostupan je na sljedećoj poveznici: https://github.com/Rlesjak/diplomski-inverter-frontend.

Na mikrokontroleru je slanje mjerenja implementirano SSE protokolom za koji svi popularni internetski preglednici imaju podršku i time eliminiraju potrebu za implementacijom protokola. Upravo je to razlog odabira SSE protokola. U TypeScript kodu je potrebno izraditi instancu klase `EventSource` koja će pokrenuti proces povezivanja na server. Status o greškama ili niz poruka koje server šalje dobiva se putem funkcija povratnog poziva (*engl. callback function*).

Ispod je dan programski kod metode *setupEventSource()* koja vrši inicijalizaciju i povezivanje na server. Metoda *setupEventSource()* poziva se klikom na gumb "SPOJI SE" koji se nalazi na alatnoj traci web sučelja (slika 5.2) i kao argument uzima URL adresu invertera koju korisnik može upisati u tekstualno polje za unos.

```
1  // inverterConnection.ts
2
3  export function setupEventSource(address: string) {
4      return new Promise<void>((resolve, reject) => {
5
6          // Pokušaj uspostaviti konekciju sa inverterom
7          inverterEventSource = new EventSource(
8              `${address}${STREAM_ENDPOINT}`, {});
9
10         // Resolvaj promise kada se konekcija uspostavi
11         inverterEventSource.onopen = () => {
12             inverterAddress = address;
13             resolve();
14         }
15
16         // Slušaj na poruke sa invertera
17         inverterEventSource.onmessage = processInverterStreamEvent;
18
19         // Rejectaj promise ako se ne uspostavi konekcija
20         // Ovaj callback također moze biti pozvan ako se konekcija izgubi
21         inverterEventSource.onerror = () => {
22             // Zatvori konekciju ako već nije zatvorena
23             inverterEventSource?.close();
24
25             // Ažuriraj globalni status konekcije na inverter
26             inverterConnected.value = false;
27             inverterAddress = null;
28
29             // Ako je još u toku uspostavljanja konekcije, rejectaj promise
30             reject();
31         }
32     });
33 }
```

Slika 5.2. Gumb za spajanje na inverter.

Pri pozivu metode *setupEventSource()* prvo se konstruira `EventSource` klasa te se registriraju funkcije povratnog poziva za detekciju uspješnog spajanja na inverter `onopen()` i za detekciju grešaka `onerror()` kako bi se korisniku moglo signalizirati da je došlo do greške. Također je registrirana funkcija povratnog poziva `onmessage()` koja se poziva svaki put kada se zaprimi poruka. U TypeScript programskom jeziku funkcije se može tretirati kao varijable pa je u objekt `inverterEventSource` upisana metoda `processInverterStreamEvent` (red 17) u kojoj se poruka obradi i proslijedi svim ostalim dijelovima programskog koda koji se registriraju kao korisnici podataka invertera.

Ispod je dana implementacija metode `processInverterStreamEvent()` kojom se izdvoje sve vrijednosti koje su odvojene zarezom. Zatim se u petlji pozovu svi pretplatnici na podatke invertera kojima se kao argument pošalje trenutni paket podataka. Varijabla `streamSubscribers` lista je pokazivača na metode koje se nazivaju pretplatnicima na podatke. Kasnije će se vidjeti da se u pojedinim dijelovima programskog koda tu listu dodaju metode kojima se konzumiraju podatci.

```
// inverterConnection.ts

function processInverterStreamEvent(event: MessageEvent) {
    const packetData: InverterPacket = parseInverterStreamPacket(event.data);

    // Pokreni sve pretplatnike na stream
    streamSubscribers.forEach((subscriber) => {
        subscriber(packetData);
    });
}
```

Jedan od glavnih dijelova web aplikacije vizualizacija je mjerenja u obliku grafova. U programskom je kodu u kojemu se konfigurira izgled grafova i izvor podataka registriran jedan pretplatnik na podatke invertera. Programski je kod dan ispod:

```
// scope.ts

// Pretplati osi grafova na stream podataka
subscribeToInverterStream("scope", (data) => {
    // Brzina vrtnje motora
    measuredSpeedSeries.add({ x: data[0], y: data[18] });
    referenceSpeedSeries.add({ x: data[0], y: data[21] });
    wantedSpeedSeries.add({ x: data[0], y: data[19]*twoPiF });
    electricalSpeedSeries.add({ x: data[0], y: data[8] });

    // DQ struje motora
    idSeries.add({ x: data[0], y: data[5] });
    iqSeries.add({ x: data[0], y: data[6] });

    // ABC struje motora
    iaSeries.add({ x: data[0], y: data[1] });
    ibSeries.add({ x: data[0], y: data[2] });
    icSeries.add({ x: data[0], y: data[3] });

    // DQ naponi motora
    vdSeries.add({ x: data[0], y: data[9] });
    vqSeries.add({ x: data[0], y: data[10] });

    // Mehanički i električni kut rotora
    electricalTheta.add({ x: data[0], y: data[7] });
    mechanicalTheta.add({ x: data[0], y: data[17] });
});
```

Argument metodi pretplatnika lista je svih vrijednosti koje je inverter poslao u jednoj poruci (`data`). Svaki put kada se metoda pretplatnika pozove svim grafovima dodaje se jedna točka kojoj je koordinata x osi vrijednost brojača izvršavanja petlje vektorskog upravljanja, a koordinata y osi vrijednost mjerenja ili parametra u tom trenutku vremena. Pozicija vrijednosti u listi definirana je u programskom kodu TCP servera i poklapa se sa strukturom `InverterStreamPacket_t`.

Web aplikacija ima mogućnost snimanja i spremanja dobivenih mjerenja u .csv datoteku, od-

nosno u datoteku tabličnog oblika u kojoj su u svakome redu upisani podatci odvojeni zarezom (*engl. Comma Separated Values*). Mogućnost snimanja i spremanja također je implementirana pomoću pretplatnika na podatke. Klikom na gumb za snimanje registrira se pretplatnik koji počne dobivati podatke i spremati ih u listu. Implementacija te metode koja se pozove klikom na gumb za početak snimanja dana je ispod.

```
// dataExport.ts

export function startRecording() {
    // Isprazni buffer
    recordingBuffer = [];

    // Dodaj header .csv datoteke
    recordingBuffer.push(csvHeader);

    // Spremanje podataka u buffer
    subscribeToInverterStream("dataExport", (data) => {
        recordingBuffer.push(data.join(","));
    });
}
```

U trenutku kada korisnik želi prekinuti snimanje pozove se metoda `stopRecordingAndExport()` kojom se prekine pretplata na podatke i korisniku otvori prozor za odabir lokacije na koju želi spremiti datoteku. Implementacija je dana ispod.

```
// dataExport.ts

export function stopRecordingAndExport() {
    // Prekid pretplate na nove podatke invertera
    unsubscribeFromInverterStream("dataExport");

    // Ako je u buffer dodano barem jedno mjerenje otvori save prozor
    if (recordingBuffer.length > 1) {
        const csvContent = recordingBuffer.join("\n");
        const blob = new Blob(
            [csvContent], { type: "text/csv;charset=utf-8;" }
        );
        . . .
}
```

Za potrebe slanja komandi inverteru koriste se HTTP GET upiti kojima je format opisan u poglavlju 5.2. Za slanje HTTP upita iz internetskog preglednika pomoću TypeScript programskog jezika koristi se dostupna metoda `fetch()` kojom je za potrebe slanja komandi inverteru implementirana je pomoćna metoda koja kao argument uzima naziv komande i vrijednost komande te složi potreban format upita i pošalje ga inverteru. Implementacija metode za slanje komandi inverteru `sendCommand()` dana je ispod.

```
// inverterConnection.ts

async function sendCommand(endpoint: string, value: number) {
    if (!inverterAddress) throw new Error("Inverter nije spojen");

    await fetch(`${inverterAddress}${endpoint}?${value}`, {
        mode: "no-cors"
    });
}
```

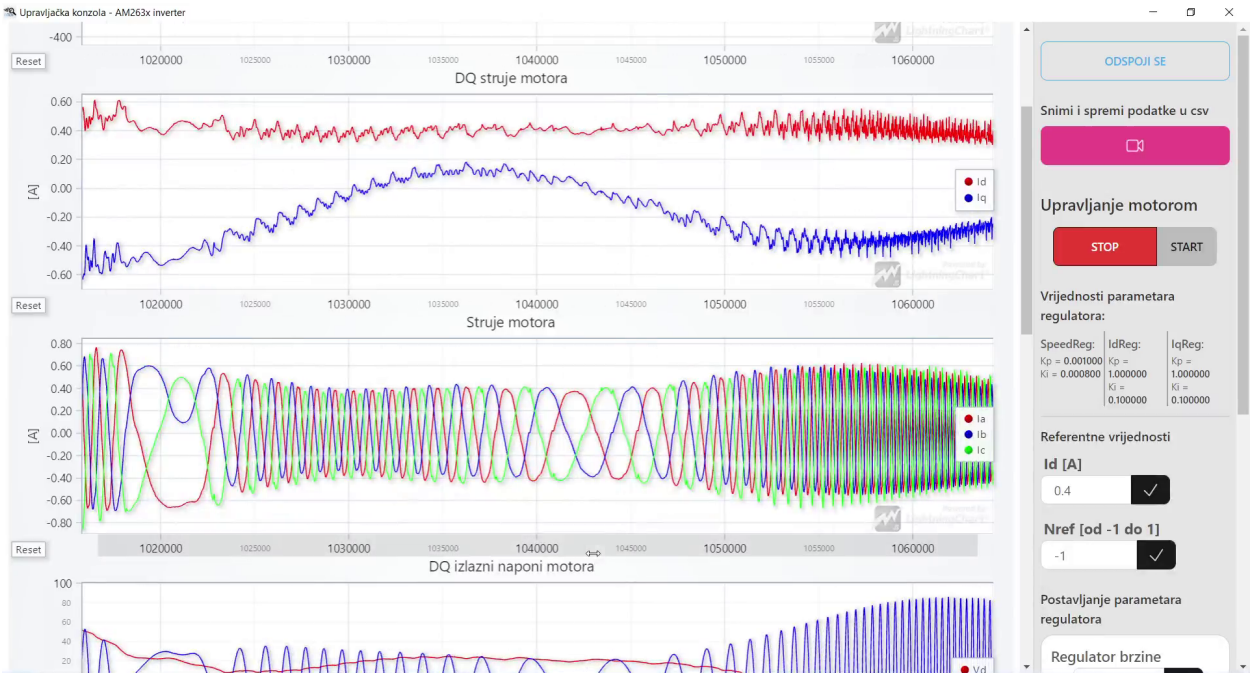
Slanje komandi koristi se kada korisnik odabere opciju za paljenje i gašenje motora, davanje referentne brzine vrtnje i struje magnetiziranja te za postavljanje parametara regulatora brzine i struje. Nekoliko primjera korištenja metode `sendCommand()` dano je ispod.

```
// inverterConnection.ts

export const InverterCommand = {
    // Pokreni motor
    motorEnable: async () => {
        await sendCommand("/MotEn", 1);
    },
    // Zaustavi motor
    motorDisable: async () => {
        await sendCommand("/MotEn", 0);
    },
    // Postavi struju magnetiziranja
    setId: async (value: number) => {
        await sendCommand("/Id_ref", value);
    },
.
.
.
```

6 EKSPERIMENTALNI REZULTATI

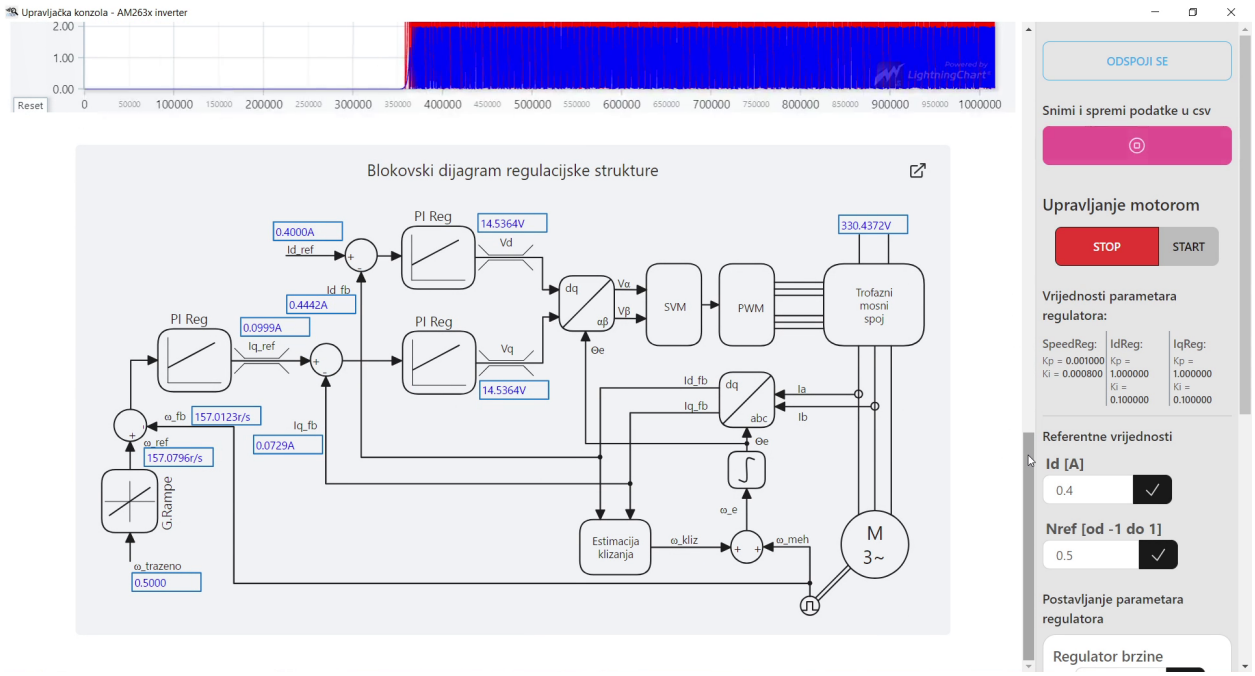
Na slici 6.1 prikazano je web sučelje za upravljanje motorom i vizualizaciju signala čija je implementacija prikazana u poglavlju 5.

S desne strane nalazi se alatna traka koja omogućuje spajanje na mikrokontroler, snimanje dobivenih podataka, pokretanje i zaustavljanje motora, prikaz trenutnih vrijednosti regulatora brzine i struje. Lijevi dio ekrana sastoji se od grafova koji prikazuju zaprimljene podatke mjerenja u stvarnom vremenu.



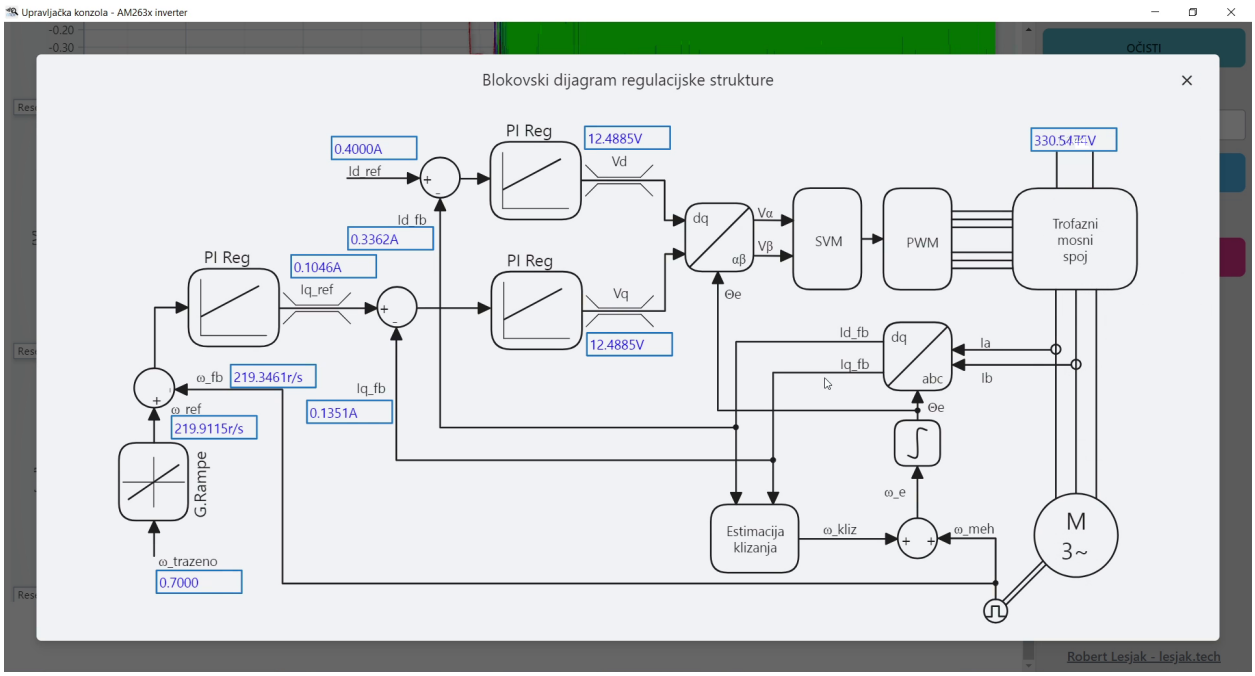
Slika 6.1. Web sučelje za upravljanje motorom i vizualizaciju signala.

Ispod grafičkih prikaza mjerenja nalazi se blokovski prikaz regulacijske strukture na kojemu se u stvarnom vremenu ispisuju trenutne vrijednosti signala na pojedinim točkama regulacijske strukture (slika 6.2).



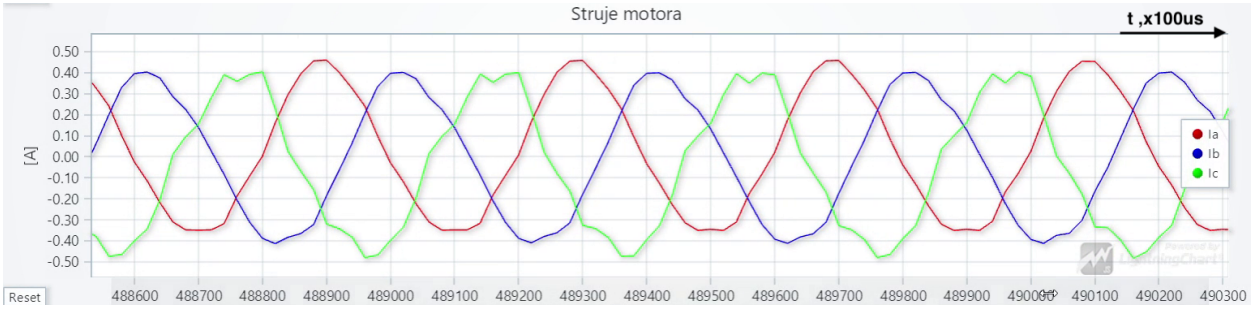
Slika 6.2. Blokovski dijagram regulacijske strukture prikazan pomoću web sučelja.

Blokovski prikaz moguće je povećati na cijeli ekran (slika 6.3).



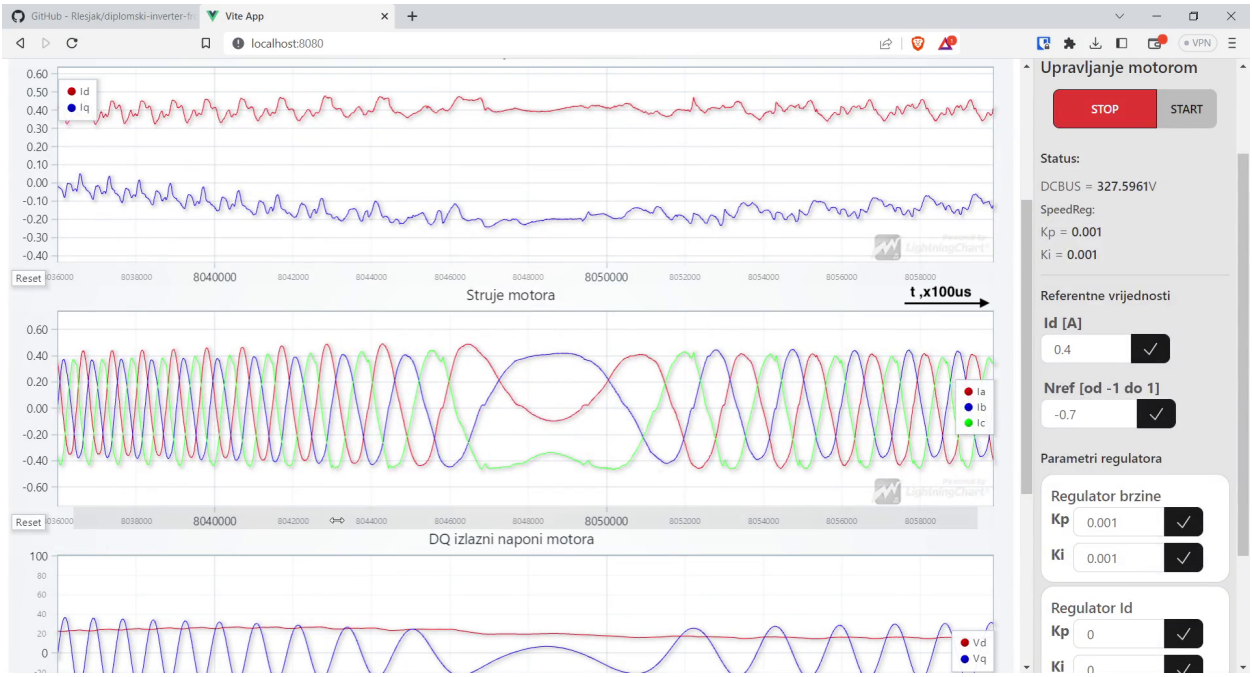
Slika 6.3. Blokovski dijagram regulacijske strukture prikazan pomoću web sučelja.

Slike 6.4 i 6.5 prikazuju valne oblike izmjerenih struja što pokazuje da je period slanja podataka web sučelju dovoljno malen ($2\,\mathrm{ms}$) da se mogu prikazivati valni oblici sinusnih signala koji se u radu kreću do frekvencije od 50 Hz.



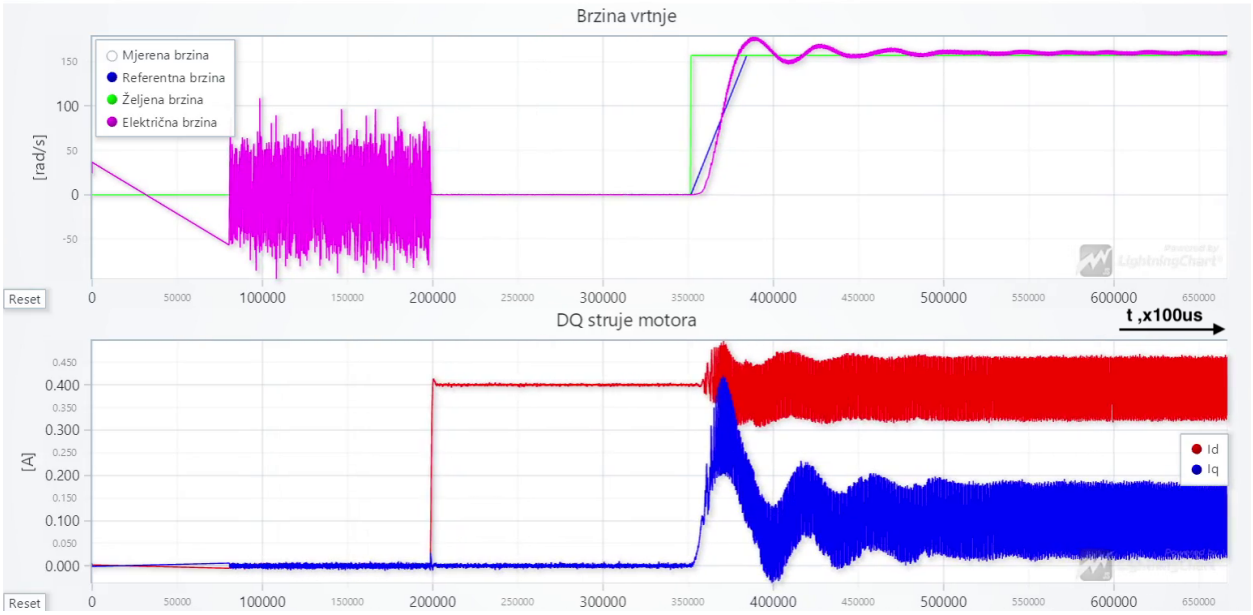
Slika 6.4. Prikaz valnih oblika mjerenih struja faza motora.

Slika 6.5 pokazuje valne oblike struja faza motora prilikom promjene smjera vrtnje.



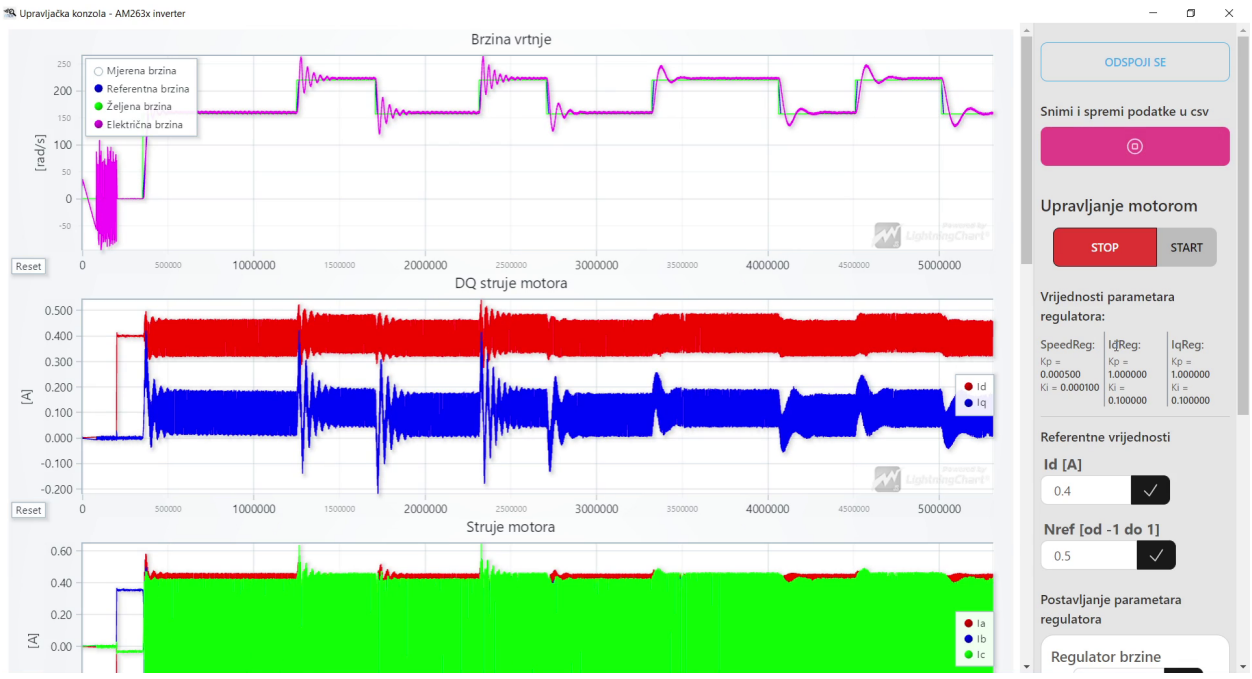
Slika 6.5. Prikaz valnih oblika struja faza prilikom promjene smjera vrtnje.

Slika 6.6 prikazuje prigušeno oscilatorni odziv brzine vrtnje motora na rampu što je posljedica loše podešenog regulatora brzine vrtnje motora. Pogodnost ovog web sučelja je da omogućuje mijenjanje parametara regulatora u realnom vremenu bez zaustavljanja motora ili ponovnog programiranja mikrokontrolera.



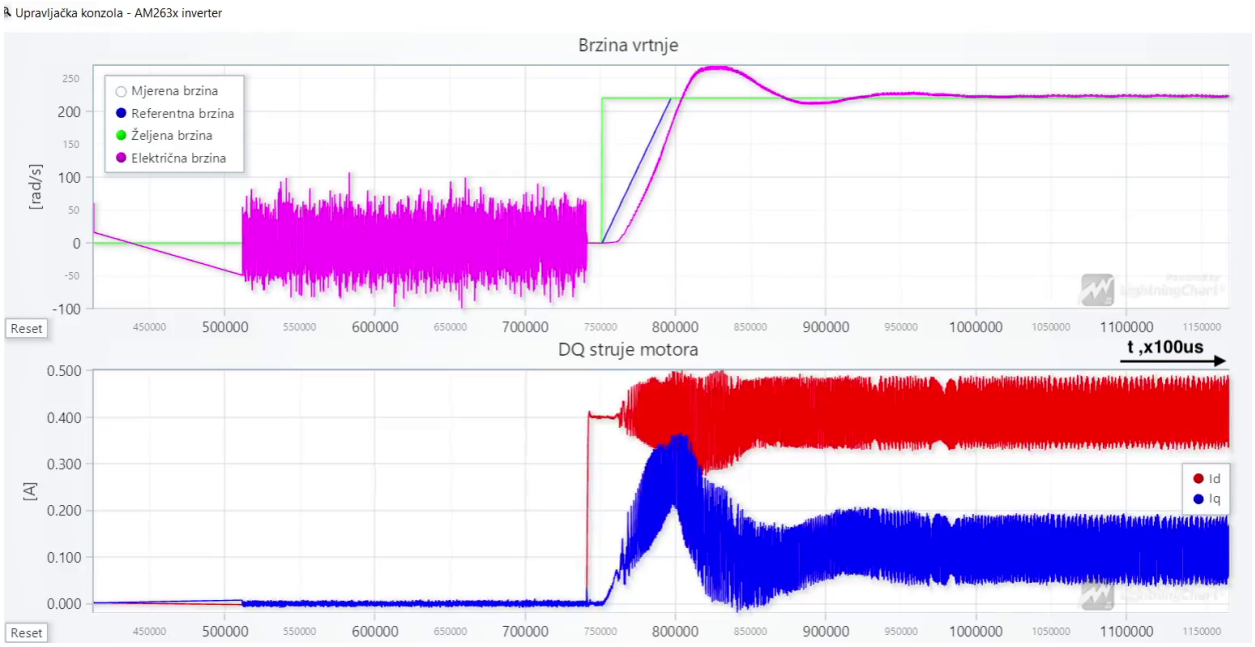
Slika 6.6. Prigušeni oscilatorni odziv brzine vrtnje.

Uzastopno podešavanje proporcionalnog i integralnog pojačanja PI regulatora mijenjajući referentnu brzinu prikazano je na slici 6.7. Prvi graf prikazuje ljubičastom bojom odziv brzine rotora asinkronog stroja na promjenu refenetne brzine. Drugi graf prikazije d i q komponente struje motora.



Slika 6.7. Primjer izmjene parametra regulatora brzine vrtnje motora pomoću web sučelja.

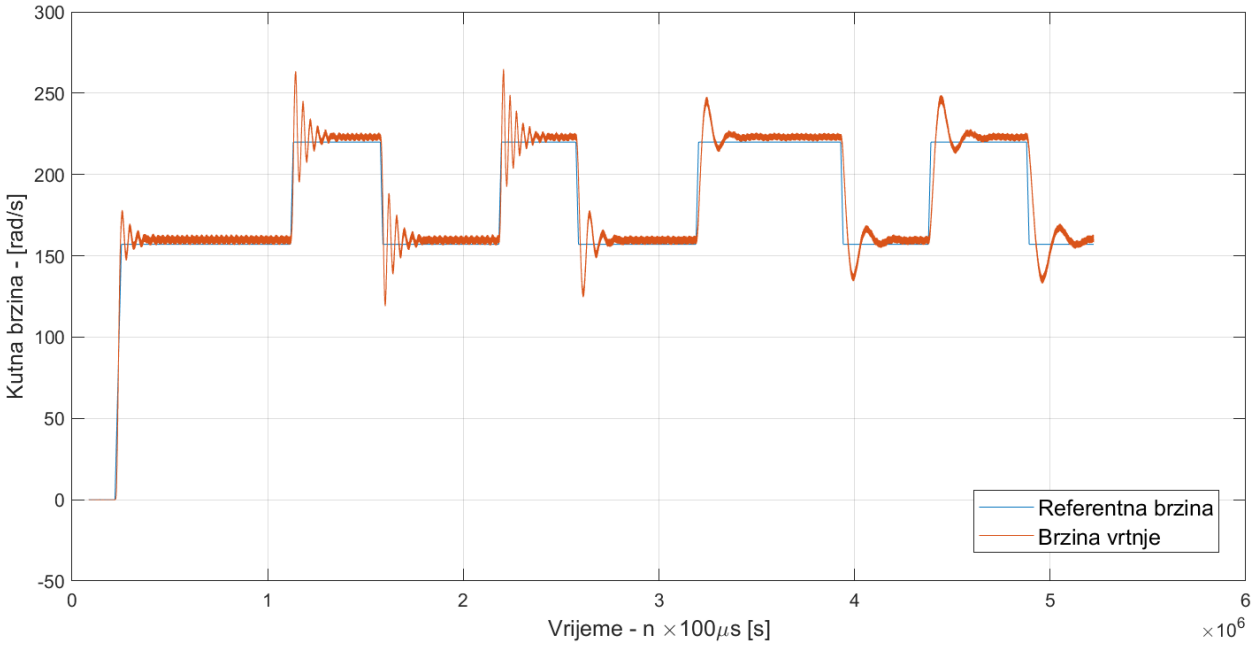
Na kraju podešavanja postignut je odziv prikazan na slici 6.8.



Slika 6.8. Odziv brzine vrtnje motora nakon grubog podešenja regulatora brzine.

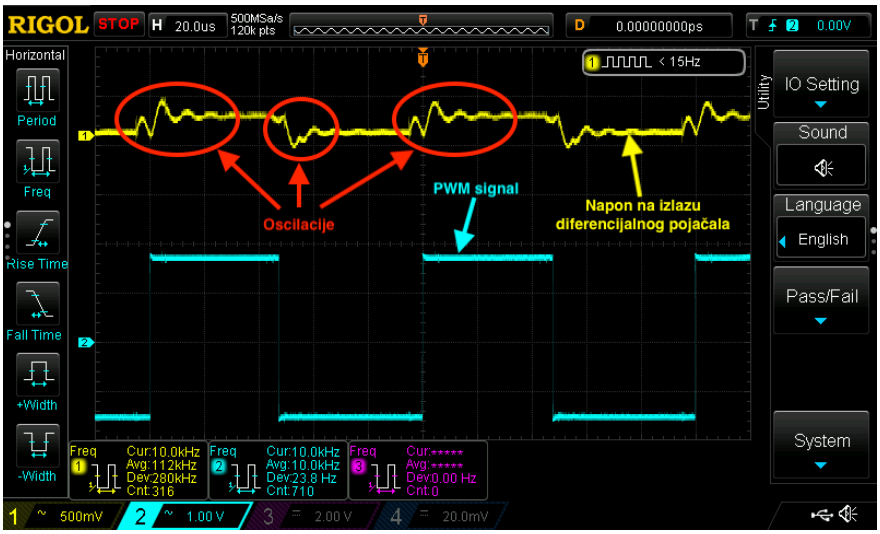
Prilikom uzastopnog podešavanja regulatora brzine vrtnje motora, snimljeni su podatci odziva brzine vrtnje motora i spremljeni u CSV datoteku. Spremljena datoteka zatim je uvezena u programski alat MatLab te je za primjer iscrtan graf odziva brzine vrtnje isti kao što ga prikazuje web sučelje na slici 6.7. Odziv prikazan pomoću MatLab alata prikazan je na slici 6.9, a MatLab programski kod za uvoz podataka dan je ispod.

```
rawData = readtable("data.csv");
timeToStep = 20;
beginPadding = 86540/timeToStep;

time = table2array(rawData(:, "IsrTick")) - 132520;
time = time(beginPadding:end);
spdRef = table2array(rawData(:,"RegSpeed_RefVal"));
spdRef = spdRef(beginPadding:end);
spdMotor = table2array(rawData(:, "omega_e"));
spdMotor = spdMotor(beginPadding:end);


figure()
plot(time, spdRef);
hold on;
plot(time, spdMotor);
xlabel("Vrijeme - n \times100\mus [s]", "FontSize", 12);
ylabel("Kutna brzina - [rad/s]", "FontSize", 12);
legend("Referentna brzina", "Brzina vrtnje", "FontSize", 12);
grid on;
```



Slika 6.9. Odziv brzine vrtnje motora prikazan pomoću izvezenih podataka iz web sučelja.

Prilikom prilagodbe HVMotorCtrl+PFCKit-R5 energetske elektronike za korištenje uz LaunchPad AM263x pokazalo se da je za mala opterećenja motora nemoguće pouzdano mjeriti struju faza motora. Pa je modificiran elektronički sklop tako što je povećano pojačanje diferencijalnih pojačala kojima se mjeri pad napona na mjernim otpornicima. Također, kako na energetskoj elektronici nisu korektno odvojeni analogni i digitalni strujni krugovi, na mjerenim signalima struje dolazi do dugotrajnih oscilacija prilikom promjena stanja PWM signala. Što znatno utječe na mogućnost točnog mjerenja vrijednosti struje. Oscilacije na signalu mjerene struje prikazane su na slici 6.10.



Slika 6.10. Oscilacije signala mjerene struje faze elektromotora.

7 ZAKLJUČAK

Implementacija vektorskog upravljanja i TCP servera za ostvarenje komunikacije s web sučeljem koristi veliki dio dostupne radne memorije mikrokontrolera AM2634B zato što su obje funkcionalnosti kompleksne. TCP komunikacijski protokol vrlo je opsežan i robustan pa zahtijeva znatnu procesorsku snagu i znatnu radnu memoriju koja je na mikroprocesorskim sustavima limitirana. Za implementaciju TCP servera u radu rezervirano je $1.5\,\mathrm{MB}$ od $2\,\mathrm{MB}$ dostupne radne memorije. Algoritam za vektorsko upravljanje numerički je složen jer se koristi trigonometrijskim i eksponencijalnim funkcijama. Zato zahtjeva moćne mikroprocesore kako bi se potrebne računske operacije mogle izvoditi u stvarnom vremenu. Često se iz tog razloga za implementaciju upravljačkih algoritama koriste digitalni procesori signala (*engl. DSP - Digital Signal Processor*) kojima su u procesor ugrađene instrukcije za obavljanje trigonometrijskih i eksponencijalnih funkcija čime se reducira potreban broj ciklusa za izračun u odnosu na implementaciju iste funkcije u programskom kodu.

Ključan dio rada implementacija je web sučelja za upravljanje i vizualizaciju mjerenih signala. Kako bi bilo moguće implementirati web sučelje, bilo je ključno koristiti mikrokontroler koji sadrži barem dva mikroprocesora zbog spomenute kompleksnosti TCP servera. Algoritam za vektorsko upravljanje i TCP server nebi se mogli izvoditi na istom mikroprocesoru. Potrebu za izvođenjem složenih aplikacija na mikrokontrolerima potvrđuje trend industrije k sve moćnijim mikrokontrolerima i tranziciji obrade podataka bliže izvoru podataka (*engl. on the edge*). Radi izvođenja dva paralelna procesa koji međusobno moraju komunicirati, korištena je biblioteka za među procesnu komunikaciju (*engl. IPC - Inter Process Communication*) koju nudi proizvođač mikrokontrolera Texas Instruments.

Implementirano web sučelje pokazalo se kao vrlo koristan alat za vizualizaciju više mjerenih veličina upravljačke strukture od jednom što može ubrzati pronalazak grešaka algoritma i mjesta za poboljšanje. U radu je kao primjer prikazano mijenjanje parametara PI regulatora brzine vrtnje u stvarnom vremenu za vrijeme vrtnje motora što korisniku web sučelja omogućuje brzi razvoj i ispitivanje nad sustavom.

Bibliografija

[1] „Revolutionizing Performance in Real-Time Control, Networking and Analytics With Sitara™ AM2x MCUs," adresa: https://www.ti.com/lit/wp/spry342/spry342.pdf (pogledano 20. 5. 2023.).

[2] Adresa: https://www.ti.com/microcontrollers-mcus-processors/arm-based-microcontrollers/arm-cortex-r-mcus/overview.html (pogledano 20. 5. 2023.).

[3] *LP-AM263*. adresa: https://www.ti.com/lit/pdf/spruj10 (pogledano 18. 2. 2023.).

[4] *AM263x Sitara*. adresa: https://www.ti.com/lit/ds/symlink/am2634.pdf (pogledano 18. 2. 2023.).

[5] *TMDSHVMTRPFCKIT*. adresa: https://www.ti.com/tool/TMDSHVMTRPFCKIT (pogledano 18. 2. 2023.).

[6] *High Voltage Motor Control and PFC Kit Hardware Reference Guide*. adresa: https://engineering.purdue.edu/~dionysis/EE452/Lab6/Lab6_Supporting_Materials/HVMotorCtrl+PFC_HWGuide.pdf (pogledano 22. 5. 2023.).

[7] B. K. Bose, *Modern Power Electronics and AC Drives*. Prentice Hall, siječanj 2002.

[8] *AM263x Sitara Technical Reference Manual*. adresa: https://www.ti.com/lit/ug/spruj17c/spruj17c.pdf (pogledano 22. 5. 2023.).

[9] „Current Sensing With <1-µs Settling for 1-, 2-, and 3-Shunt FOC Inverter Reference Design TI Designs Current Sensing With <1-µs Settling for 1-, 2-, and 3-Shunt FOC Inverter Reference Design," adresa: www.ti.com (pogledano 29. 4. 2023.).

[10] N. Quang i J. Dittrich, *Vector Control of Three-Phase AC Machines: System Development in the Practice* (Power Systems). Springer Berlin Heidelberg, 2008.

[11] *Zero Sequence Modulation Tutorial*. adresa: https://microchipdeveloper.com/mct5001:which-zsm-is-best (pogledano 3. 6. 2023.).

8 SAŽETAK

U radu je prikazan proces implementacije algoritma vektorskog upravljanja asinkronim strojem i proces implementacije web sučelja za upravljanje upravljačkim sustavom na procesoru AM2634. Pri razvoju su korišteni razvojni alati koje nudi proizvođač mikrokontrolera (Texas Instruments) Code Composer Studio i SysConfig. Code Composer Studio razvojno je okruženje za uređivanje programskog koda te za otkrivanje i rješavanje problema na mikroprocesoru. SysConfig je konfiguracijski alat kojime se za pojedine procesore proizvođača Texas Instruments može grafičkim sučeljem generirati inicijalizacijski programski kod. Web sučelje razvijeno je s ciljem jednostavnog upravljanja upravljačim sustavom, vizualizacijom mjerenja i jednostavnim prikupljanjem podataka mjerenja. Implementirano je pomoću programskog jezika TypeScript i biblioteke Vue.js u NPM (Node Package Manager) projektu. Komunikacija između web sučelja i mikrokontrolera izvedena je na *ETHERNET* komunikacijskom kanalu pomoću *TCP* i *HTTP* komunikacijskih protokola.

Programska aplikacija za vektorsko upravljanje i TCP/HTTP server za komunikaciju s web sučeljem nije moguće istovremeno izvoditi na jednom mikroprocesoru AM2634B mikrokontrolera, no korišteni mikrokontroler AM2634B sastoji se od četiri mikroprocesora pa su dvije aplikacije implementirane na dva zasebna mikroprocesora. Obije aplikacije međusobno komuniciraju pomoću međuprocesorske komunikacije kako bi se omogućilo slanje mjerenih podataka na web sučelje i slanje komandi sa web sučelja programskoj aplikaciji za vektorsko upravljanje.

Uz digitalnu elektroniku izvedenu mikrokontrolerom AM2634 i LaunchPad AM263x mikroprocesorskim sustavom korištena je energetska elektronika HVMotorCtrl+PFCKit-R5 proizvođača TexasInstruments. HVMotorCtrl+PFCKit-R5 energetska elektronika namjenjena je korištenju uz procesore serije C2000, pa je u radu prikazan proces prilagodbe energetske elektronike kako bi se koristila uz LaunchPad AM263x mikroprocesorski sustav.

Ključne riječi: AM263x Sitara ™ mikrokontroleri; vektorsko upravljanje; web aplikacija; programiranje u jeziku c; programiranje u jeziku TypeScript; HTTP; TCP

9 ABSTRACT

The thesis presents the process of implementing field oriented controll of an induction machine and the web interface for managing the control system on the AM2634 processor. The development tools offered by the microcontroller manufacturer (Texas Instruments) Code Composer Studio and SysConfig were used during the development. Code Composer Studio is an integrated development environment for code editing, and debuging the microprocessor. SysConfig is a configuration tool that can be used to generate initialization code for Texas Instruments processors through a graphical interface. The web interface was developed with the aim of simple management of the control system, visualization of measurements and simple collection of measurement data. It is implemented using the TypeScript programming language and the Vue.js framework in the NPM (Node Package Manager) project. Communication between the web interface and the microcontroller is performed on the *ETHERNET* communication channel using *TCP* and *HTTP* communication protocols.

Since Field-oriented control and the TCP/HTTP server cannot be run simultaneously on one core of the AM2634B microcontroller which has four cores the two applications run on two separate cores. Both applications communicate with each other using inter-processor communication to allow sending measurement data to the web interface and commands from the web interface to the vector control software application.

In addition to the digital electronics made with the AM2634 microcontroller and the LaunchPad AM263x development board, the power electronics HVMotorCtrl+PFCKit-R5 manufactured by TexasInstruments were used. HVMotorCtrl+PFCKit-R5 power electronics is intended for use with processors of the C2000 series, so the thesis shows the process of adapting the power electronics to be used with the LaunchPad AM263x development board.

Keywords: AM263x Sitara ™ microcontrollers; field oriented control; web application; programming in c language; programming in TypeScript language; HTTP; TCP

10 PRILOZI

10.1 Korišteni elektromotor i enkoder

Tablicom 10.1 dani su parametri asinkronog stroja (slike 10.1) korištenog tokom ispitivanja energetske elektronike i implementacije algoritma vektorskog upravljanja.

Tablica 10.1. Parametri asinkronog elektromotora "Marathon electric 5K33GN2A"

Parametar	Vrijednost
Nazivna snaga	$P_{meh} = 186\,\mathrm{W}$
Nazivni broj okretaja	$N = 1725\,\mathrm{omin}^{-1}$
Broj pari polova	$p = 2$
Otpor statora	$R_S = 11.05\,\Omega$
Otpor rotora	$R_R = 6.11\,\Omega$
Rasipni induktivitet statora	$L_{sl} = 0.022\,48\,\mathrm{H}$
Rasipni induktivitet rotora	$L_{rl} = 0.022\,48\,\mathrm{H}$
Induktivitet magnetiziranja	$L_m = 0.293\,94\,\mathrm{H}$
Induktivitet statora	$L_s = L_m + L_{sl} = 0.316\,42\,\mathrm{H}$
Induktivitet rotora	$L_s = L_m + L_{rl} = 0.316\,42\,\mathrm{H}$

Slika 10.1. Marathon electric 5K33GN2A asinkroni elektromotor

Za mjerenje brzine vrtnje i pozicije rotora korišten je optički enkoder prikazan na slici 10.2. Jedna puna rotacija enkodera podijeljena je na 8192 koraka, dakle kutna rezolucija je $0{,}044°$.

Slika 10.2. Quantum Devices QPhase Optički inkrementalni enkoder

10.2 Oznake signala adaptera za povezivanje "LaunchPad AM263x" na "HVMotorCtrl+PFCKit-R5" energetsku elektroniku

Tablica 10.2. Oznake signala na tiskanoj pločici energetske elektronike HVMotorCtrl+PFCKit-R5

OPIS	SMJER	SILKSCREEN	SHEMA TAG	NOZICA NA MCU	DIM100 Broj Nozice
pwm faza U	ULAZ NA INVERTER	**PWM-1H**	PWM-1A	GPIO-00	23
pwm faza V	ULAZ NA INVERTER	**PWM-2H**	PWM-2A	GPIO-02	24
pwm faza W	ULAZ NA INVERTER	**PWM-3H**	PWM-3A	GPIO-04	25
pwm faza U	ULAZ NA INVERTER	**PWM-1L**	PWM-1B	GPIO-01	73
pwm faza V	ULAZ NA INVERTER	**PWM-2L**	PWM-2B	GPIO-03	74
pwm faza W	ULAZ NA INVERTER	**PWM-3L**	PWM-3B	GPIO-05	75
mjerena strija faze U	IZLAZ IZ INVERTERA	**Ifb-U**	Ifb-U	ADC-B3	

Tablica 10.3. Oznake signala iz SysConfig konfiguracije i njihova pozicija na LaunchPad AM263x konektorima

	Input Pin(Signals)	HW Pin	**LaunchPad Konektor Pin**	Opis	Mjesta korištenja
CONFIG_ADC4				Resolver	FOC_loop.c { linija 455-458 }
	ADC4_AIN0	U6	**J3.27**	Sinus 1	
	ADC4_AIN1	V5	**J7.65**	Cosinus 1, Cosinus 2	
	ADC4_AIN3	U5	**Nema**	Sinus 2	
CONFIG_ADC1				Struja A	FOC_loop.c
	ADC1_AIN2	Any(T12)	**J7.67**		
CONFIG_ADC2				Struja B	FOC_loop.c
	ADC2_AIN0	Any(R10)	**J3.25**		
	ADC2_AIN2	Any(U10)	**J7.68**		
CONFIG_ADC3				Struja C	FOC_loop.c
	ADC3_AIN0	Any(U7)	**J3.26**		
	ADC3_AIN3	Any(R7)	**J5.46**		
CONFIG_ADC0				DC bus napon	OC_loop.c { linija 460 }
	ADC0_AIN2	Any(T14)	**J7.66**		
	ADC0_AIN3	Any(U14)	**J1.2**		

Tablica 10.4. Oznake signala iz SysConfig konfiguracije i njihova pozicija na LaunchPad AM263x konektorima

	Signals	HW Pin	**LaunchPad Konektor Pin**	Opis	Mjesta korištenja
CONFIG_EPWM7				Trigger DMA for resolver excitation	
	EPWM7_A	F4	**Nema**		
CONFIG_EPWM0				PWM za fazu A	FOC_loop.c, ucc5870.c
	EPWM0_A	Any(B2)	**J2.11**		
	EPWM0_B	Any(B1)	**J6.59**		
CONFIG_EPWM1				PWM za fazu B	FOC_loop.c, ucc5870.c
	EPWM1_A	Any(D3)	**J4.37**		
	EPWM1_B	Any(D2)	**J4.38**		
CONFIG_EPWM2				PWM za fazu C	FOC_loop.c, ucc5870.c
	EPWM2_A	Any(C2)	**J4.39**		
	EPWM2_B	Any(C1)	**J4.40**		

10.3 Programski kod projekta za vektorsko upravljanje

Inicijalni projekt preuzet je s internetskih stranica Texas Instruments te je modificiran i nadograđen za potrebe ovoga rada.

Poveznica projekta na datum 10.6.2023.:

https://dev.ti.com/tirex/explore/node?node=A__ALE2ZhNMRrGUETGIo9ioyA__AM263x-Traction-Inverter-Demo__TgMm0pe__LATEST

Programski kod priloga 10.3 dostupan je u obliku Code Composer Studio projekta na sljedećoj internetskoj adresi:

https://github.com/Rlesjak/am263_tractionDemo

10.3.1 Motor_t struktura

```
// foc.h

typedef struct _Motor_t_
{
    // Parametri motora
    float32_t Ls_d;
    float32_t Ls_q;
    float32_t Rs;
    float32_t Phi_e;

    float32_t I_abc_A[3];   // Struje faza abc [A]
```

```
12
13      float32_t I_scale;      // ADC faktor skaliranja struje
14      float32_t V_scale;      // ADC faktor skaliranja napona
15
16      float32_t dcBus_V;           // Izmjereni napon DC linka [V]
17      float32_t oneOverDcBus_invV; // 1/dcBus_V
18
19      float32_t I_ab_A[2];         // Struja u alpha beta sustavu [A]
20      float32_t I_dq_A[2];         // Struje u dq sustavu [A]
21
22      float32_t theta_e;           // Električni kut rotora [rad]
23      float32_t omega_e;           // Električna brzina rotora [rad/s]
24      float32_t Sine;              // Sinus kuta rotora
25      float32_t Cosine;            // Kosinus kuta rotora
26
27      float32_t theta_e_out;       // Kompenzirani električni kut rotora [rad]
28      float32_t Sine_out;          // Sinus komp. el. kuta rotora
29      float32_t Cosine_out;        // Kosinus komp. el. kuta rotora
30
31      float32_t Vff_dq_V[2];       // Naponi za rasprezanje d i q struje
32      float32_t Vout_dq_V[2];      // Izlazni napon u dq sustavu [V]
33      float32_t Vout_ab_V[2];      // Izlazni napon u alpha beta sustavu [V]
34
35      float32_t Vout_dq_out_norm;   // Pomocna varijabla za ogranicenje izlaznog napona
36      float32_t Vout_dq_sat_gain;   // Pomocna varijabla za ogranicenje izlaznog napona
37
38      float32_t Vff_max;
39      float32_t Vff_min;
40
41      PI_Obj pi_spd;  // Struktura PI regulatora brzine
42      PI_Obj pi_id;   // Struktura PI regulatora struje d
43      PI_Obj pi_iq;   // Struktura PI regulatora struje q
44
45      float32_t vqLimit;
46      float32_t modulationLimitSquare;
47
48      float32_t Vout_max;
49
50      // Varijable za računanje kompenzacije
51      float32_t sampleTime;
52      float32_t outputTimeCompDelay;
53      float32_t resolverCompDelay;
54      uint16_t resolverCompIdx;
55      uint16_t isrResRatio;
56  } Motor_t;
```

10.3.2 Inicijalizacijski programski kod vektorskog upravljanja

```
1  // FOC_loop.c
2
3  void FOC_init(void);
4  void FOC_cal(void);
5  void FOC_run(void);
6  void FOCcal_ISR(void *handle);
7  __attribute__ ((section(".tcmb_code"))) void FOCrun_ISR(void *handle);
8  static inline void TRINV_HAL_setPwmOutput(PWMData_t *pwm);
9
10 /* Brojač izvođenja interrupta vektorskog upravljanja */
11 __attribute__ ((section(".tcmb_data"))) uint32_t gIsrCnt = 0;
12
13 /* Upravljanje upaljeno/ugaseno */
14 volatile MotorRunStop_e runMotor = MOTOR_STOP;
15
16 /* Id referentan vrijednost */
```

```
volatile float32_t IdRef     = 0.0;
/* Iq referentna vrijednost */
volatile float32_t IqRef     = 0.0;
/* Željena brzina, normalizirana na sinkronu [-1 do 1] */
volatile float32_t SpdRef  = 0.0;

//
// Interrupt routine
//
/* Hardware interrupt instance */
static HwiP_Object gAdcHwiObject;

//
// Offset calibration routine is run to calibrate for any offsets on the opamps
//
/* Calibration routine */
volatile uint32_t gFlag_AdcCal = FALSE;
volatile uint32_t offsetCalCounter = 0;
/* Current offset */
volatile float32_t offset_curU = 0.0f;   // offset in AMC1302 current U fbk channel @ 0A
volatile float32_t offset_curV = 0.0f;   // offset in AMC1302 current V fbk channel @ 0A
volatile float32_t offset_curW = 0.0f;   // offset in AMC1302 current W fbk channel @ 0A
/* Offset filter coefficient*/
float32_t K1 = 0.9980001f;      // Offset filter coefficient K1: 0.05/(T+0.05);
float32_t K2 = 0.0019999f;      // Offset filter coefficient K2: T/(T+0.05);
/* DC bus voltage filter */
__attribute__ ((section(".tcmb_data"))) Filter_t filterVdc;


//
// Field Oriented Control
//
/* Samping period (sec), see parameter.h */
float32_t T = 0.001 / ISR_FREQUENCY;
/* Instance a ramp controller to smoothly ramp the frequency */
__attribute__ ((section(".tcmb_data"))) RMPCNTL rc1;
/* Instance a ramp generator */
__attribute__ ((section(".tcmb_data"))) RAMPGEN rg1;
/* Variables for Motor control */
__attribute__ ((section(".tcmb_data"))) Motor_t motor1;
/* Variables for PWM update */
__attribute__ ((section(".tcmb_data"))) PWMData_t pwm1;
/* Variables for induction machine current model */
__attribute__ ((section(".tcmb_data"))) ACI_Model_t aci1;


//
// ENCODER
//
__attribute__ ((section(".tcmb_data"))) PosSpeed_Object posSpeed;
__attribute__ ((section(".tcmb_data"))) float32_t encoder_omega;


void FOC_init(void){

    //
    // Init encoder struct
    //
    posSpeed.thetaElec = 0;
    posSpeed.thetaMech = 0;
    posSpeed.directionQEP = 0;
    posSpeed.thetaRaw = 0;
    posSpeed.mechScaler = 0.0001220703125f; //1/(2048*4)
    posSpeed.polePairs = PAIRS;
    posSpeed.calAngle = 0;
```

```
    posSpeed.speedPR = 0;
    posSpeed.K1 = 1/(BASE_FREQ*T); // Skaliranje razlike kuta u brzinu
    posSpeed.K2 = 1/(1+T*2*PI*5); // Low Pass filter za mjerenje brzine
    posSpeed.K3 = 1-posSpeed.K2;
    posSpeed.oldPos = 0;

    //
    // Initialize the calibration module
    //
    offsetCalCounter = 0;
    offset_curU = 0;
    offset_curV = 0;
    offset_curW = 0;

    //
    // Initialize the RAMPGEN module
    //
    initRampGen(&rg1);
    rg1.StepAngleMax = BASE_FREQ * T;
    rg1.Angle = 0.0f;
    rg1.Out = 0.0f;
    rg1.Gain = 1.0f;
    rg1.Offset = 1.0f;

    //
    // Initialize the RAMP control module
    //
    initRampControl(&rc1);

    //
    // Parametri motora
    //
    motor_init(&motor1);
    motor1.Ls_d = LS_D;
    motor1.Ls_q = LS_Q;
    motor1.Rs = RS;
    motor1.Phi_e = FLUX / (2.0f * PI);
    motor1.I_scale = ADC_I_SCALE_FCT;
    motor1.V_scale = ADC_V_REFRENCE_SCALE;


    //
    // Parametara regulatora brzine
    //
    motor1.pi_spd.Kp = 0.001;
    motor1.pi_spd.Ki = 0.0008;
    motor1.pi_spd.outMax = 2.0f;
    motor1.pi_spd.outMin = -motor1.pi_spd.outMax;

    //
    // Setup PI parameter for current loop
    //
    motor1.vqLimit = VQ_LIMIT;
    motor1.modulationLimitSquare = MODULATION_LIMIT * MODULATION_LIMIT;

    // Parametri regulatora D struje
    motor1.pi_id.Kp = 1;
    motor1.pi_id.Ki = 0.1;
    motor1.pi_id.outMax = RATED_VOLTAGE * motor1.vqLimit;
    motor1.pi_id.outMin = -motor1.pi_id.outMax;

    // Parametri regulatora Q struje
    motor1.pi_iq.Kp = 1;
    motor1.pi_iq.Ki = 0.1;
    motor1.pi_iq.outMax = RATED_VOLTAGE * motor1.vqLimit;
    motor1.pi_iq.outMin = -motor1.pi_iq.outMax;

    // Granice izlaza regulatora struje
```

```
    motor1.Vout_max = motor1.pi_id.outMax;
    motor1.Vff_max = 0.05f * RATED_VOLTAGE;
    motor1.Vff_min = -0.05f * RATED_VOLTAGE;

    motor1.sampleTime = 1.0f / (ISR_FREQUENCY * 1000.0f);
    motor1.outputTimeCompDelay = motor1.sampleTime -
        ADC_S_H_TIME_NS * NS_TO_S / 2.0f;
    motor1.resolverCompDelay = motor1.sampleTime -
        ((ADC_S_H_TIME_NS * 4.0f + ADC_CONV_TIME_NS * 4.0f) / 2.0f) * NS_TO_S;
    motor1.resolverCompIdx = 0U;
    motor1.isrResRatio = ISR_RES_RATIO;

    //
    // Initialize Induction Machine Current Model
    //
    aci1.Kt = 10;
    aci1.Kr = 0.1;
    aci1.IMDs = 0;
    aci1.Wslip = 0;

    //
    // Initialize LPF for VDC
    //
    FILTER_FO_init(&filterVdc, (VDC_FLT_BW_HZ * TWO_PI),
                   (ISR_FREQUENCY * 1000.0f));

    /* Inicijalizacija PWM strukture */
    pwm1.modulationLimit = 1.0f;
    pwm1.inv_half_prd = INV_PWM_HALF_TBPRD;


    //
    // Init FLAGS
    //
    runMotor = MOTOR_STOP;

    gLog_ptr[0] = &motor1.omega_e;
    gLog_ptr[1] = &motor1.theta_e;
    gLog_ptr[2] = &motor1.Sine;
    gLog_ptr[3] = &motor1.Cosine;
    gLog_ptr[4] = &pwm1.Vabc_pu[0];
    gLog_ptr[5] = &pwm1.Vabc_pu[1];
    gLog_ptr[6] = &pwm1.Vabc_pu[2];
    gLog_ptr[7] = &motor1.I_abc_A[0];
    gLog_ptr[8] = &motor1.I_abc_A[1];
    gLog_ptr[9] = &motor1.I_abc_A[2];
    gLog_ptr[10] = &motor1.I_dq_A[0];
    gLog_ptr[11] = &motor1.I_dq_A[1];
    gLog_ptr[12] = NULL;
    gLog_ptr[13] = &encoder_omega;
    gLog_ptr[14] = NULL;
    gLog_ptr[15] = NULL;

    LoopLog_init();
}

void FOC_cal(void){

    HwiP_Params hwiPrms;
    int32_t status;

    /* Register & enable interrupt */
    HwiP_Params_init(&hwiPrms);
    hwiPrms.intNum = CSLR_R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_0;
    hwiPrms.callback = &FOCcal_ISR;
    hwiPrms.priority = 1U;
    hwiPrms.isPulse = 1U;
    status = HwiP_construct(&gAdcHwiObject, &hwiPrms);
```

```
218     DebugP_assert(SystemP_SUCCESS == status);
219 
220     ADC_clearInterruptStatus(CONFIG_ADC4_BASE_ADDR,ADC_INT_NUMBER1);
221 
222     while(gFlag_AdcCal == FALSE)
223     {
224         ClockP_usleep(200000L);
225     }
226 
227     HwiP_destruct(&gAdcHwiObject);
228 
229 }
230 
231 void FOC_run(void){
232 
233     HwiP_Params hwiPrms;
234     int32_t status;
235 
236     /* Register & enable interrupt */
237     HwiP_Params_init(&hwiPrms);
238     hwiPrms.intNum = CSLR_R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_0;
239     hwiPrms.callback = &FOCrun_ISR;
240     hwiPrms.priority = 1U;
241     hwiPrms.isPulse = 1U;
242     status = HwiP_construct(&gAdcHwiObject, &hwiPrms);
243     DebugP_assert(SystemP_SUCCESS == status);
244 
245     ADC_clearInterruptStatus(CONFIG_ADC4_BASE_ADDR,ADC_INT_NUMBER1);
246 }
```

10.3.3 Pomoćna metoda PopulateInverterStreamPacket()

```
1  // SerialiseDeserialise.c
2 
3  void PopulateInverterStreamPacket(InverterStreamPacket_t* packetHandle, uint32_t
       isrTick)
4  {
5      Motor_t* motorStructHandle = FOC_DANGER_getMotorStructPointer();
6      PosSpeed_Object* posSpeedStructHandle = FOC_DANGER_getPosSpeedStructPointer();
7 
8      packetHandle->IsrTick = isrTick;
9      packetHandle->Ia = motorStructHandle->I_abc_A[0];
10     packetHandle->Ib = motorStructHandle->I_abc_A[1];
11     packetHandle->Ic = motorStructHandle->I_abc_A[2];
12     packetHandle->DcBus = motorStructHandle->dcBus_V;
13     packetHandle->Id = motorStructHandle->I_dq_A[0];
14     packetHandle->Iq = motorStructHandle->I_dq_A[1];
15     packetHandle->theta_e = motorStructHandle->theta_e;
16     packetHandle->omega_e = motorStructHandle->omega_e;
17     packetHandle->Out_Vd = motorStructHandle->Vout_dq_V[0];
18     packetHandle->Out_Vq = motorStructHandle->Vout_dq_V[2];
19     packetHandle->RegSpeed_Fback = motorStructHandle->pi_spd.fbackValue;
20     packetHandle->RegSpeed_Output = motorStructHandle->pi_spd.outValue;
21     packetHandle->RegId_Fback = motorStructHandle->pi_id.fbackValue;
22     packetHandle->RegId_Output = motorStructHandle->pi_id.outValue;
23     packetHandle->RegIq_Fback = motorStructHandle->pi_iq.fbackValue;
24     packetHandle->RegIq_Output = motorStructHandle->pi_id.outValue;
25     packetHandle->EncoderTheta = posSpeedStructHandle->thetaElec;
26     packetHandle->EncoderOmega = posSpeedStructHandle->speedPR;
27     packetHandle->SpeedRef = FOC_getSpeedRef();
28     packetHandle->MotorRunStop = (float32_t) FOC_getMotorRunState();
29     packetHandle->RegSpeed_RefVal = motorStructHandle->pi_spd.refValue;
30     packetHandle->RegSpeed_Kp = motorStructHandle->pi_spd.Kp;
31     packetHandle->RegSpeed_Ki = motorStructHandle->pi_spd.Ki;
```

```
32      packetHandle->RegId_RefVal = motorStructHandle->pi_id.refValue;
33      packetHandle->RegId_Kp = motorStructHandle->pi_id.Kp;
34      packetHandle->RegId_Ki = motorStructHandle->pi_id.Ki;
35      packetHandle->RegIq_RefVal = motorStructHandle->pi_iq.refValue;
36      packetHandle->RegIq_Kp = motorStructHandle->pi_iq.Kp;
37      packetHandle->RegIq_Ki = motorStructHandle->pi_iq.Ki;
38  }
```

10.4 Programski kod projekta za TCP server

Inicijalni projekt preuzet je s internetskih stranica Texas Instruments te je modificiran i nadograđen za potrebe ovoga rada.

Poveznica projekta na datum 10.6.2023.:

https://dev.ti.com/tirex/explore/node?node=A__AA6iLI.knzRdCyrrAz7i8Q__com.ti.MCU_PLUS_SDK_AM263X__aBmeCqF__LATEST

Programski kod dostupan je u obliku Code Composer Studio projekta na sljedećoj internetskoj adresi:

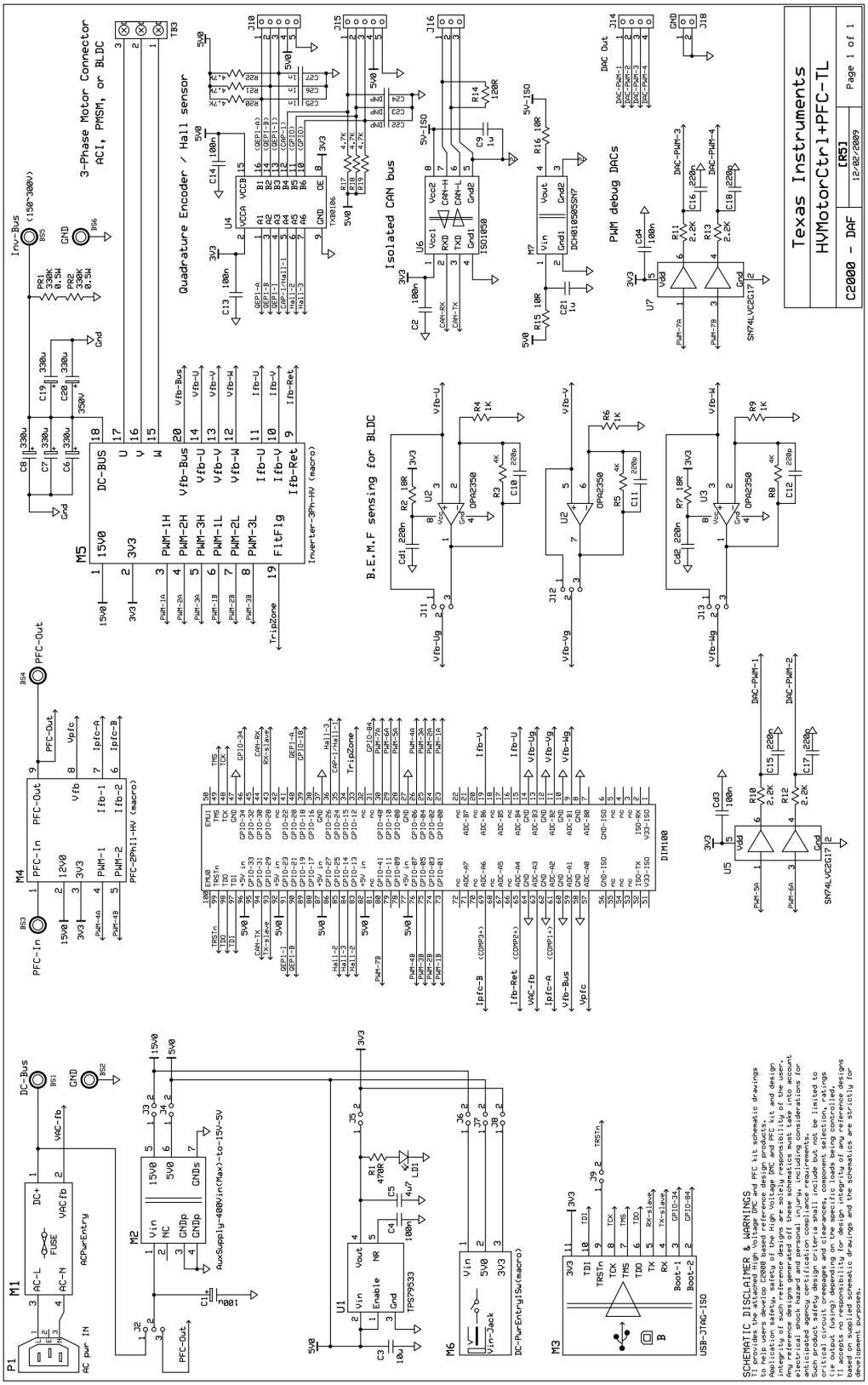
https://github.com/Rlesjak/am263_traction_tcp_server

10.5 Programski kod projekta web aplikacije

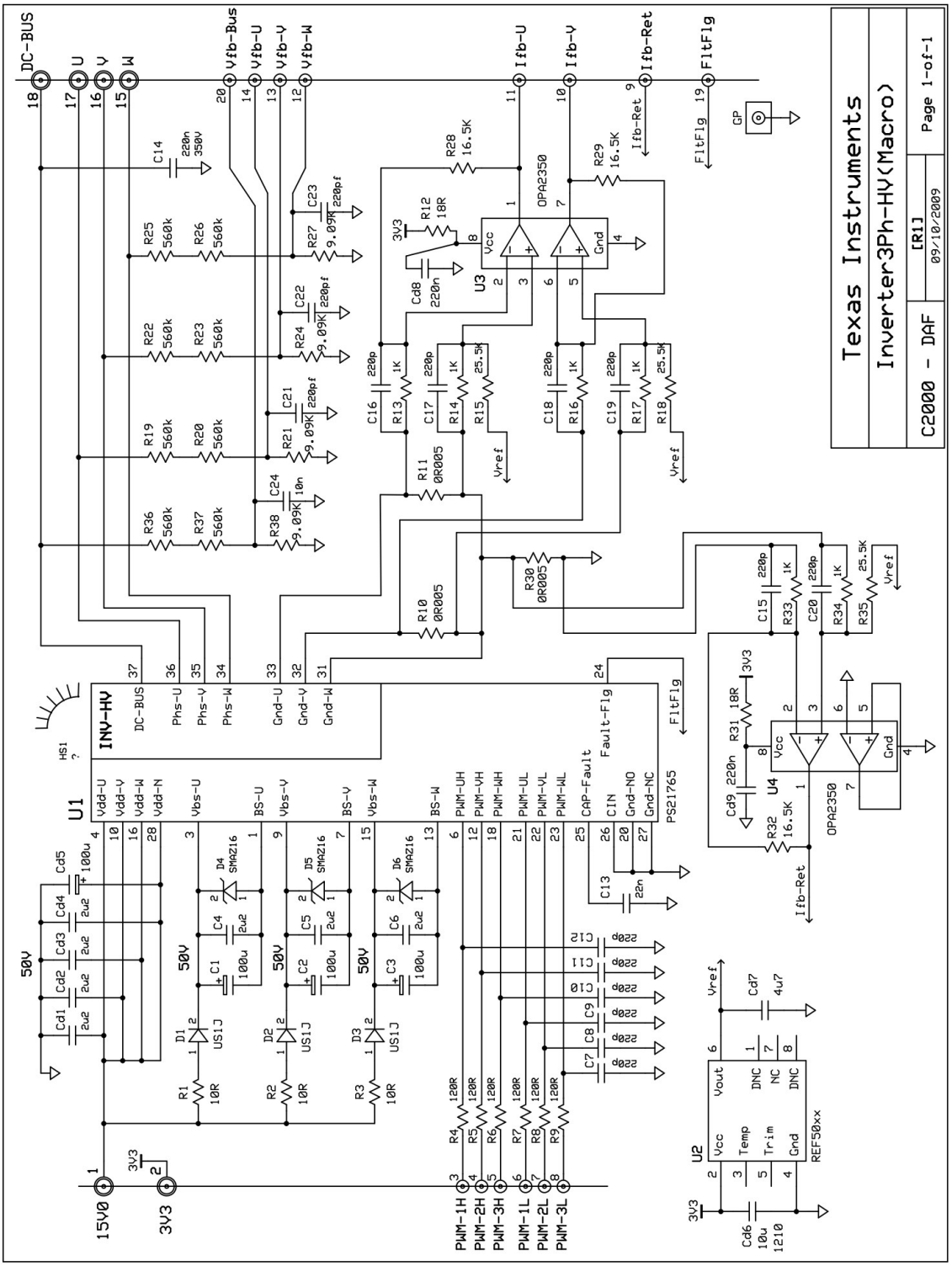
Programski kod web sučelja u potpunosti je razvijen za diplomski rad u obliku NPM (*Node Package Manager*) projekta te se nalazi na sljedećoj internetskoj adresi:

https://github.com/Rlesjak/diplomski-inverter-frontend

10.6 SHEMA HVMotorCtrl+PFCKit-R5 ENERGETSKE ELEKTRONIKE



Slika 10.3. Shema HVMotorCtrl+PFCKit-R5 energetske elektronike - dio 1/2 [5]



Slika 10.4. Shema HVMotorCtrl+PFCKit-R5 energetske elektronike - dio 2/2 [5]