UPRAVLJANJE BLDC MOTOROM POMOĆU CAN-BUS KOMUNIKACIJE

Kovačević, Duje

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni diplomski studij Elektrotehnika

Duje Kovačević

ZAVRŠNI RAD

UPRAVLJANJE BLDC MOTOROM POMOĆU CAN-BUS KOMUNIKACIJE

Split, Rujan 2023.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni diplomski studij Elektrotehnika

Predmet: Senzorske mreže

ZAVRŠNI RAD

Kandidat: Duje Kovačević

Naslov rada: Upravljanje BLDC motorom pomoću CAN-BUS komunikacije

Mentor: dr.sc. Tonko Kovačević, prof. v.š.

Split, Rujan 2023.

SADRŽAJ

Sažetak

Upravljanje BLDC motorom pomoću CAN-BUS komunikacije

U ovom diplomskom radu dan je uvid u izradu sustava upravljanja BLDC motorom pomoću CAN_BUS komunikacije. Standardni istosmjerni motori oslanjaju se za mehanički način komutacije što dovodi do problema pouzdanosti i efikasnosti. BLDC motori nemaju taj problem jer je njegova komutacija ostvarena elektronički i iz tog razloga upravljanje BLDC motorom je kompleksnije. Mogućnost upravljanja motorom s većih udaljenosti zahtjeva primjenu komunikacijskog protokola koji je u ovom radu CAN_BUS komunikacijski protokol.

Ključne riječi: BLDC motor, CAN_BUS, ESP32, Trofazni H-most, primopredajnik, hall senzor

Summary

Controlling a BDLC motor using CAN-BUS communication

This Thesis gives insight into designing a control system for BDLC motor using CAN-BUS communication. Standard motors using direct current rely on mechanical way of commutation, which can cause problems in reliability and efficiency. BDLC motors don’t face such problems because their commutation is achieved electronically, which on the other hand makes controlling a BDLC motor more complex. The possibility of controlling a BDLC motor from longer distances demands the application of a communication protocol such as CAN-BUS communication protocol, which is used in this Thesis.

Keywords: BDLC motor, CAN_BUS, ESP32, 3-Phase H-Bridge, transceiver, hall sensor

1. UVOD

Zadatak ovog rad je omogućiti upravljanje istosmjernim motorom bez četkica (BLDC) preko CAN_BUS komunikacijskog protokola. U radu je opisana sva potrebna teorija za razumijevanje zadatka ovog diplomskog rada. Također opisan je postupak ožičenja svih potrebnih komponenti za sad sustava.

U prvom poglavlju dan je uvid u BLDC motore i njihov način rada kao i načini na koje se oni mogu upravljati. Opisan je CAN_BUS protokol i sve specifikacije koje treba zadovoljiti za uspješnu komunikaciju. U zadatku završnog rada opisan je postupak izrade zadatka od ožičenja i izrade elektroničkih sustava do programiranja ESP32 sustava na čipu.

2. BLDC MOTOR

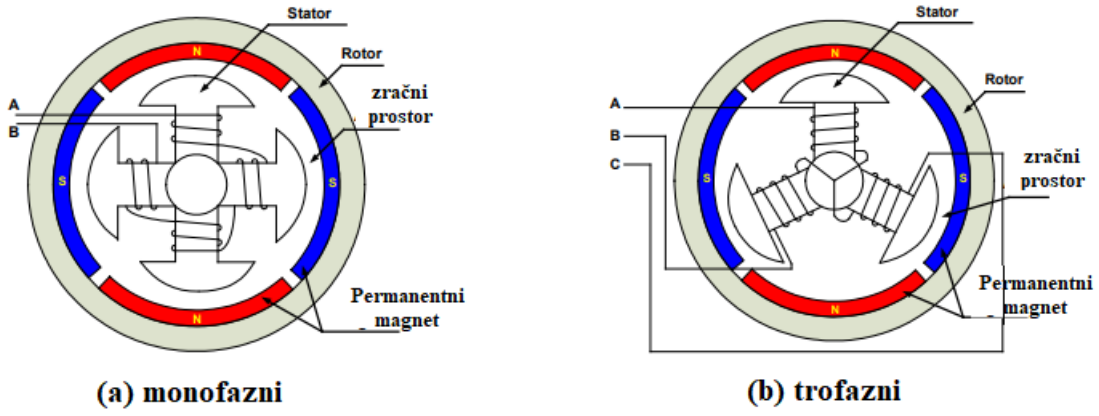
2.1. Općenito o BLDC motoru

BLDC (eng. Brushless DC motor) je elektro motor upravljan istosmjernom strujom bez korištenja četkica za kontakt s rotorom. Električni impulsi za upravljanje motor su trapezoidnog oblika no za različite izvedbe motora može se koristiti i sinusni signal. Kod istosmjernog motora s četkicama komutacija je ostvarena preko četkica koje klize po lamelama to uvodi problem jer su četkice i lamele potrošni materijal i takvi motori zahtijevaju česte servise. Kod BLDC motora komutacija je prebačena na stator motora a rotor je permanentni magnet. Za upravljanje BLDC motorom potreban je kontroler koji će puštati struju kroz zavoje statora u točno određenim trenucima. Trenuci u kojima će kontroler prebacivati smjer struje kroz zavoje i s tim magnetsko polje može biti definirano na više načina :

1. Detektiranjem povratne elektromotorene sile na slobodnom namotu
2. Primjenom hall senzora
3. Primjerom rotacijskog enkodera

Postoje tri klasifikacije BLDC motora : monofazni, dvofazni i trofazni motori. Bitno je napomenuti da svaka faza ima jednaki broj namota i polova. Najčešće korišteni BLDC motor je trofazni zbog svojih karakteristika brzine vrtnje i okretnog momenta.

Na slici 2.1 može se vidjeti pojednostavljena konstrukcija monofaznog (a) i trofaznog (b) BLDC motora.

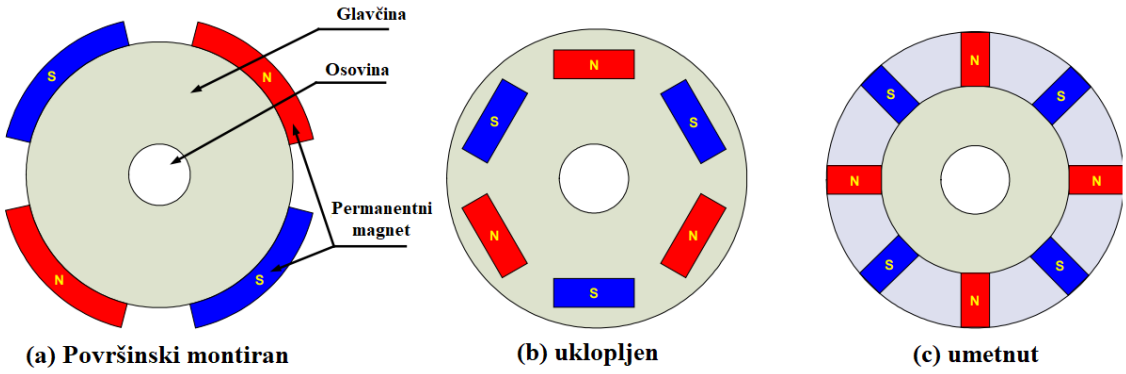


Slika 2.1 konstrukcija monofaznog (a) i trofaznog (b) BLDC motora [1]

Postoje dvije vrste namota statora: trapezni i sinusni, što se odnosi na oblik signala povratne elektromotorne sile. Oblik povratne elektromotorne sile određen je različitim međuvezama namotaja i veličinom zračnog prostora. Sinusni signal proizvodi ravnomjerniji okretni moment motora od trapezoidnog signala. Upravljanje sa sinusnim signalom je znatno složenije i skuplje nego upravljanje sa trapernim signalom [1].

Rotor BLDC motora sastoji se od osovine i glavčine s trajnim magnetima koji se nalaze između dva i osam parova polova koji su postavljeni naizmjenično. Postoji više magnetskih materijala, kao što su mješavine željeza i slitina rijetkih legura. Najčešće korišteni magneti su feritni magneti jer su relativno jeftini i dostupni, iako magneti od rijetkih legura postaju sve više korišteni zbog svoje visoke magnetske gustoće. Veća gustoća pomaže da rotor bude manji i lakši u odnosu na rotor s feritnim magnetom [1].

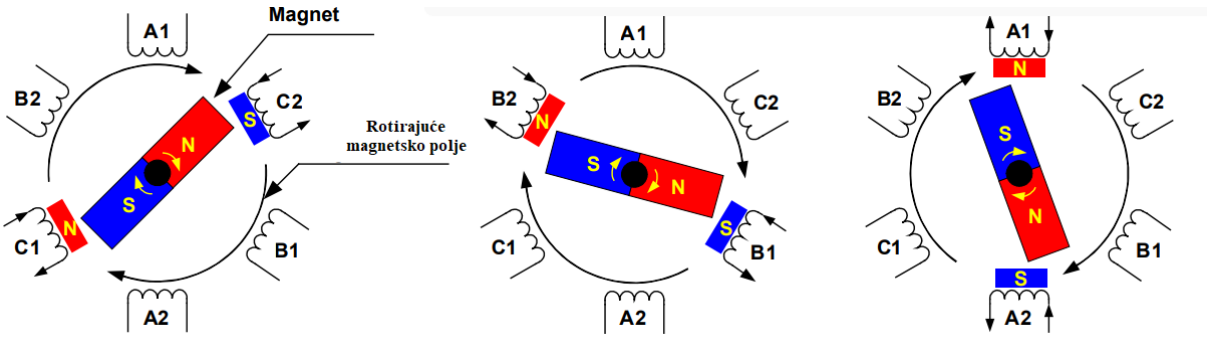
Na slici 2.2. mogu se vidjeti poprečni presjeci položaja magneta za tri vrte rotora.



Slika 2.2 Poprečni presjek magneta na rotoru [1]

2.2. Kako radi BLDC motor

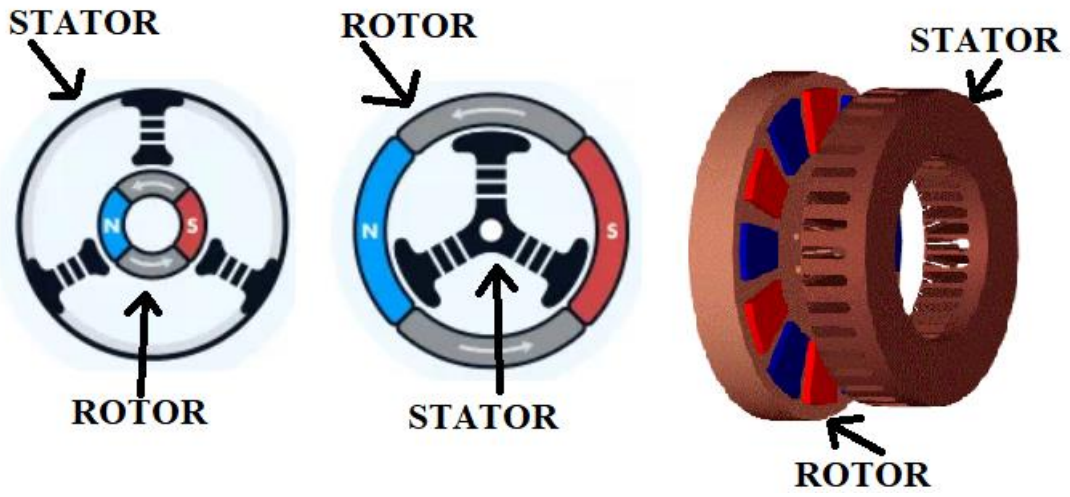
Rad motora temeljen je na privlačenju ili odbijanju između magnetskih polova permanentnog magneta i namota. Korištenjem trofaznog motora prikazanog na slici 2.3 proces rotacije započinje kada struja teče kroz jedan od tri namota statora i stvara magnetski pol koji privlači najbliži permanentni magnet suprotnog pola. Rotor će nastaviti rotaciju ako struja prijeđe na sljedeći namot. Sekvencijskim puštanjem struje kroz namote stvara se okretno magnetsko polje koje rotor prati. Okretni moment u ovom primjeru ovisi o amplitudi struje i broju zavojna na namotima statora, jakosti i veličini permanentnih magneta, zračnom prostoru između rotora i namota te duljini rotirajućeg elementa [1].



Slika 2.3 Rotacija motora [1]

2.3. Vrste BLDC motora

Tri osnovne podjele BLDC motora su: s unutrašnjim rotorom (eng. Inrunner motor), s vanjskim rotorom (eng. Outrunner) i axial flux motore. BLDC motori s unutrašnjim rotorom imaju sličan dizajn kao sinkroni motori s permanentnim magnetom gdje je stator s namotajima s vanjske strane motora, a rotor s permanentnim magnetima je u unutrašnjosti motora. Na slici 2.4. lijevo može se vidjeti presjek BLDC motora s unutrašnjim rotorom. BLDC motori s unutrašnjim rotorom su duži, tanji i mogu postići veći broj okretaja u odnosu na BLDC motor s vanjskim rotorom iste mase. Kod motora s vanjskim rotorom stator s namotima se nalazi u unutrašnjem djelu motora, a rotor s permanentnim magnetima je na vanjskog strani motora. Na slici 2.4. u sredini se može vidjeti prijesjek BLDC motora s vanjskim rotorom. Motori s vanjskim rotorom su deblji, kraći i imaju veći okretni moment u odnosu na BLDC motore s unutrašnjim rotorom iste mase. Motori s unutrašnjim rotorom imaju bolju učinkovitost od motora s vanjskim rotorom zbog manje inercije rotora no s druge strane su bučniji [6]. Motori s vanjskim rotorom se koriste u izradi ventilatora za prijenosna računala zbog svojih dobrih zvučnih karakteristika. Kod axial flux motora stator i rotor su paralelni u odnosu jedno na drugo što se može vidjeti na slici 2.4. desno. U usporedbi s prethodna dva tipa BLDC motora koju spadaju u kategoriju motora sa radijalnim tokom, axial flux BLDC motori imaju veći okretni moment [7].

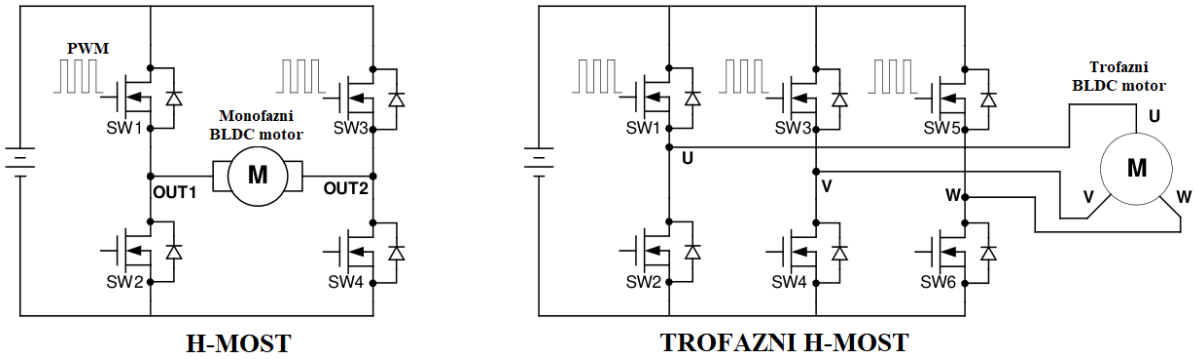


Slika 2.4 Lijevo: motor s unutrašnjim rotorom, Sredina: motor s vanjskim rotorom, Desno: axial flux motor [8][9]

2.4. Upravljanje BLDC motorom

2.4.1. Konfiguracija sklopki i PWM

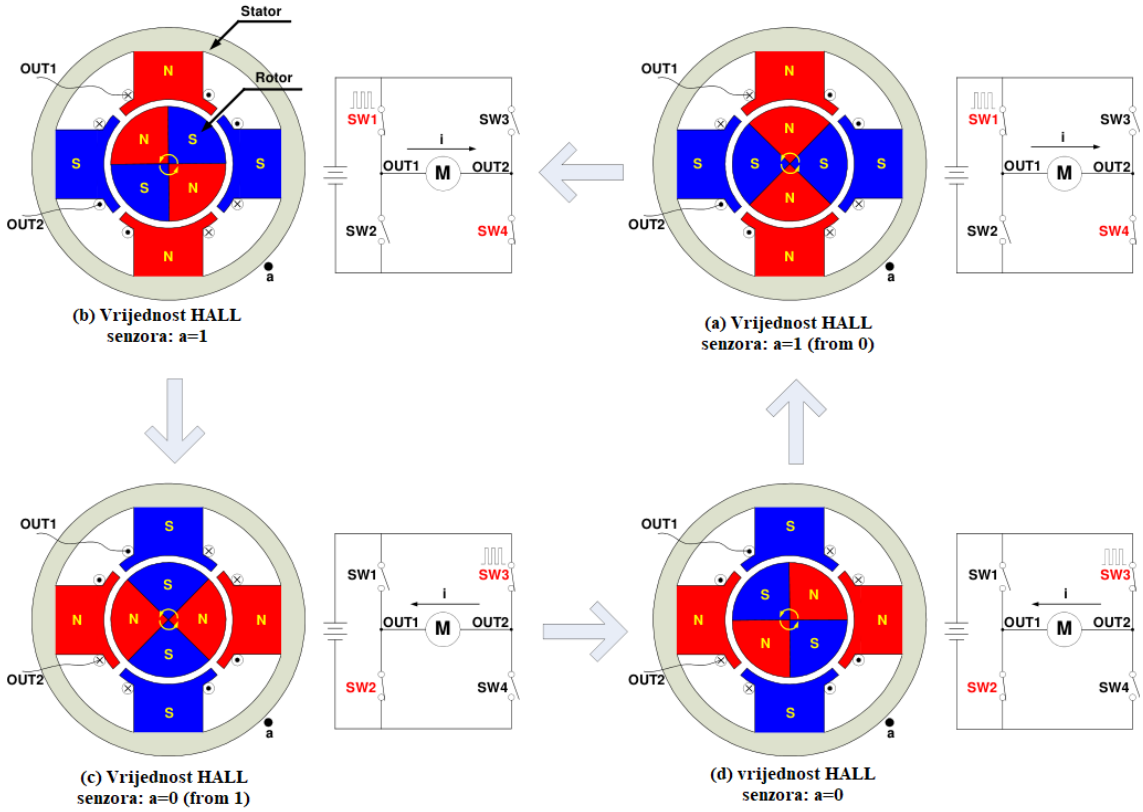
BLDC motori koriste MOSFET-e i IGBT tranzistore kao sklopke kako bi sekvencijalno puštali struju kroz namote statora i tako okretali rotor motora. Sklopke su najčešće povezane u strukturi H-mosta (slika 2.5. a) za monofazne BLDC motore i u strukturi trofaznog H-mosta (slika 2.5. b) za trofazne BLDC motore. Najčešće se s gornjim sklopkama upravlja s PWM (pulsno širinska modulacija) signalom, koja pretvara DC napon u modularni napon, koji jednostavno i učinkovito ograničava startnu struju, kontrolu brzine vrtnje rotora i snagu okretnog momenta [1].



Slika 2.5 Struktura H-mosta i trofaznog H-mosta

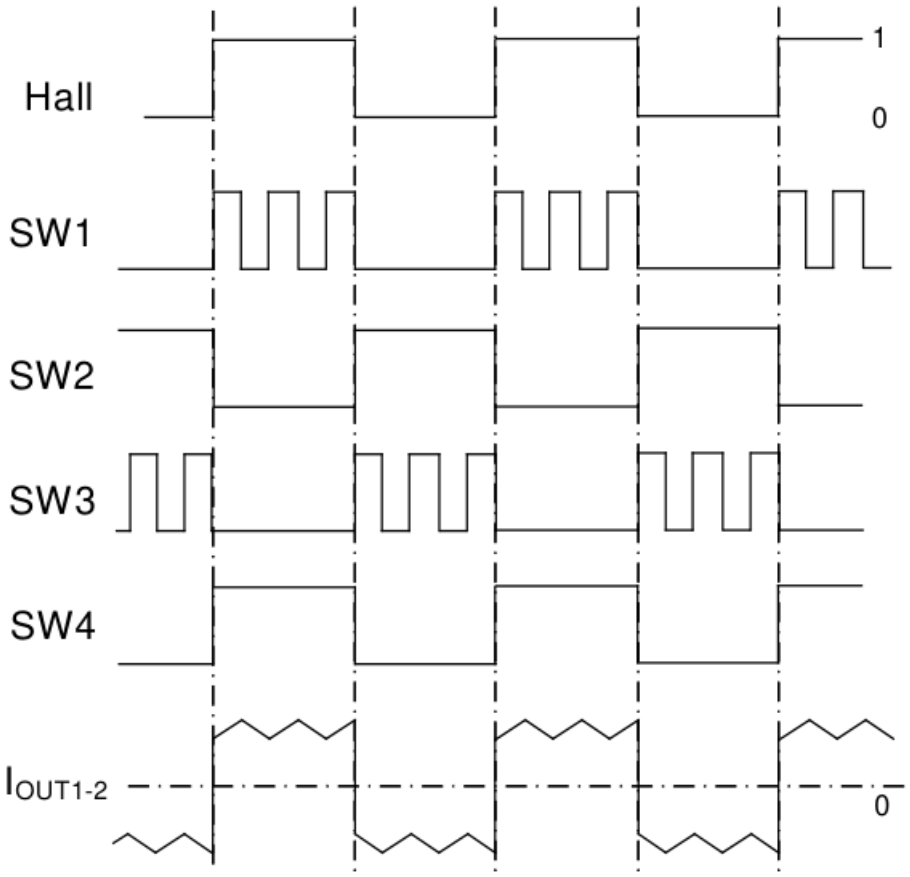
2.4.2. Upravljanje monofaznim BLDC motorom

Komutacija BLDC motora oslanja se na povratnu informaciju o položaju rotora kako bi aktivirala odgovarajuće sklopke za stvaranje najvećeg okretnog momenta. Najlakši način za točno definiranje pozicije rotora je s korištenjem hall senzora ugrađenih u stator motora [1]. SW1 i SW4 prekidači se uključuju kada je izlaz HALL senzora u logičkom stanju HIGH, kao što je prikazana na slici 2.6. (a) i (b). U ovoj fazi struja teće kroz namote statora od OUT1 do OUT2 i inducira elektromagnetske polove na statoru. Magnetska sila permanentnih magneta na rotoru i proizvedena magnetska sila na namotima uzrokuju rotaciju rotora. Nakon što rotor prijeđe 180° izlaz HALL senzora mijenja stanje zbog nailazećeg južnog pola rotora. SW2 i SW3 zatim se uključuju i tako promjenu tok struje kroz namote koja ide prema OUT2 i izlazi na OUT3 kao što je prikazano na slici 2.6. (c) i (d). Suprotni magnetski polovi potiču rotor da se nastavi rotirati u istom smjeru [1].



Slika 2.6 Sekvenca komutacije monofaznog BLDC motora [1]

Na slici 2.7. prikazan je primjer signala s HALL senzora u usporedbi s pogonskim signalima sklopki i struji koja teće kroz namote statora. Struja namota ima pilasti oblik signala koji je uzrokovan zbog PWM kontrole gornjih sklopki. Napon napajanja namota statora, frekvencija prekidanja i radni ciklus PWM signala su tri ključna parametra za određivanje brzine vrtnje rotora i okretnog momenta [1].

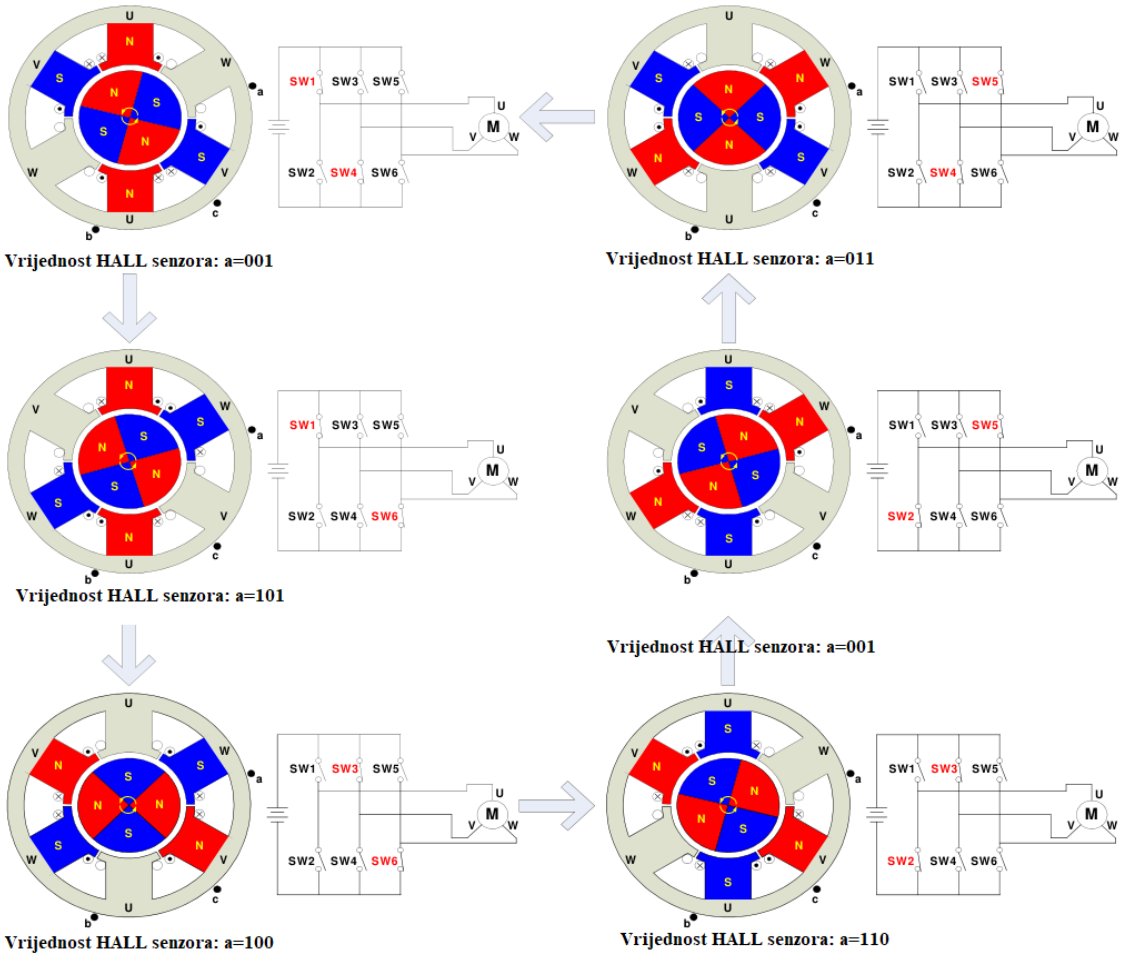


Slika 2.7 Valni oblici ulaznih i izlaznih signala [1]

2.4.3. Upravljanje trofaznim BLDC motorom

Trofazni BLDC motor zahtjeva tri HALL senzora za detekciju pozicije rotora. Prema fizičkim pozicijama HALL senzora imamo dvi vrte izlaza: sa 60° faznim pomakom i sa 120° faznim pomakom. Kombiniranjem izlaza HALL senzora može se odrediti točan komunikacijski slijed puštanja struja kroz namote statora. Slika 1.8. prikazuje komutacijsku sekvencu BLDC motora za smjer obrnut kazaljke na satu. Tri HALL senzora (a) , (b) , (c) su montirana na stator s pomakom od 120° dok su namoti statora povezani u spoj zvijezde. Za svaku rotaciju rotora od 60° jedan od HALL senzora mijenja svoje stanje. Potrebno je 6 koraka tj promjena stanja HALL senzora da se završi cijeli električni ciklus. Za svaki korak u sekvenci jedan kontakt motora je u HIGH stanju, drugi je u LOW stanju, a treći kontakt motora je u beskontaktnom stanju (float) [1]. Kako se sekvenca komutacije mijenja u krug tako se i kontakti motora nalaze u različitim logičkim stanjima. Namot koji nije priključen na nikakvo logičko stanje služi kao generator povratne elektromotorne sile koja služi kao detekcija pozicije rotora u BLDC motorima bez HALL senzora.

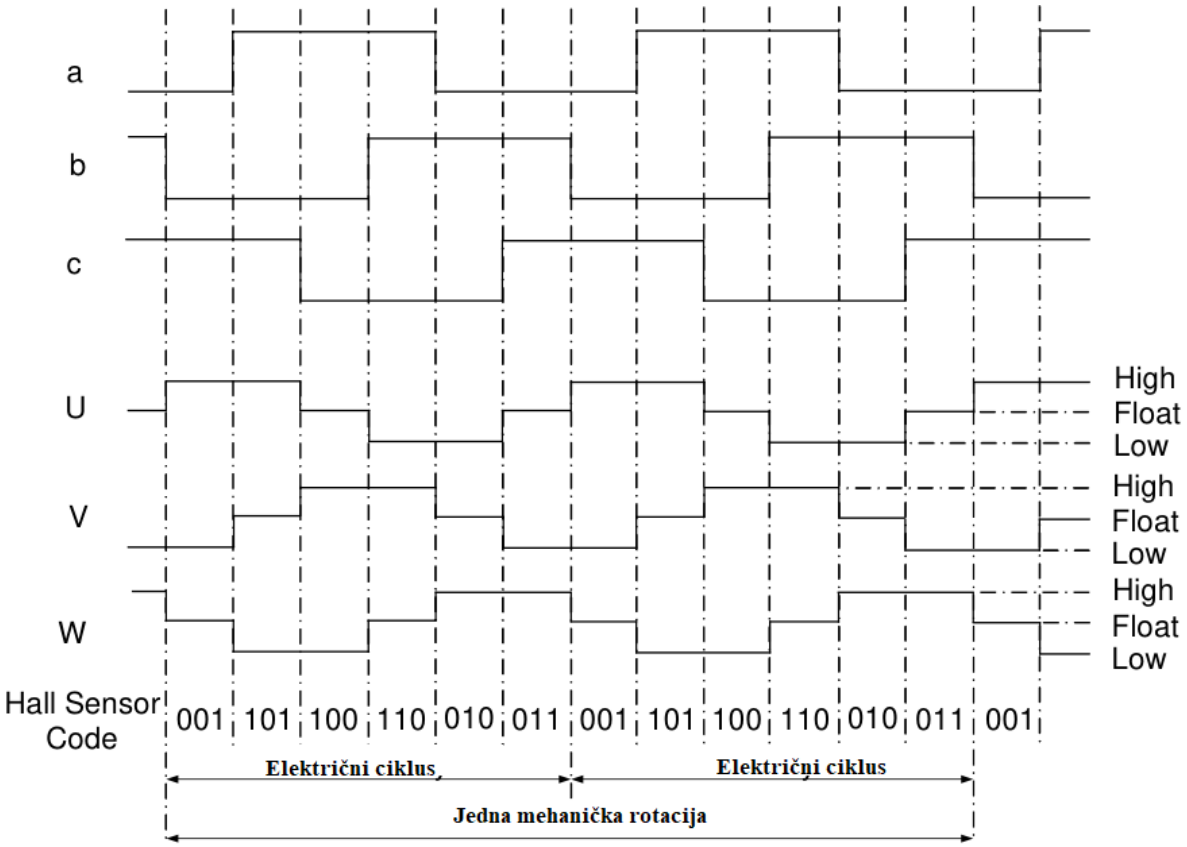
Jedan električni ciklus ne mora odgovarati potpunoj mehaničkoj rotaciji rotora. Broj električnih ciklusa potrebnih za potpunu mehaničku rotaciju rotora određen je brojem pari polova rotora. Svaki par polova rotor zahtjeva jedan signalni ciklus u jednoj mehaničkoj rotaciji. Broj ciklusa signala je jednak broju parova polova rotora [1].



Slika 2.8 Sekvenca komutacije trofaznog BLDC motora [1]

Da bi mijenjali brzinu rotacije motora potrebno je koristiti PWM signal znatno veće frekvencije nego što je frekvencija vrtnje motora. Općenito frekvencija PWM signala trebala bi biti najmanje 10 puta veća od maksimalne frekvencije vrtnje motora. Prednost korištenja PWM signala je što se radni ciklus PWM signala može ograničiti tako da se napon napajanja izjednači s nazivnim naponom motora [1].

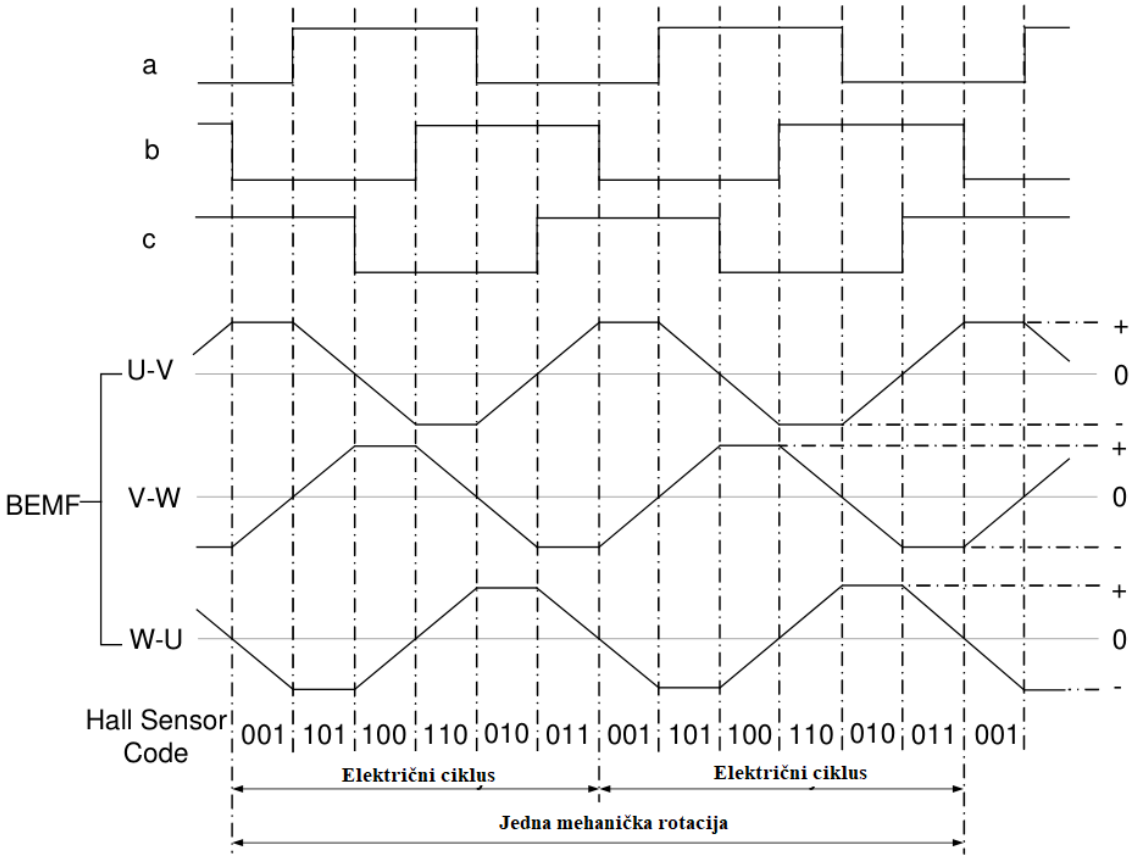
Slika 2.9. prikaziva vremenske dijagrame u kojima se može vidjeti stanje svakog namotaja i stanje svakog Hall senzora. Također u primjeru se može vidjeti da su signali HALL senzora razmaknuti za 120° jedan od drugoga.



Slika 2.9 Vremenski dijagram stanja namotaja i izlaza HALL senzora [1]

2.4.4. Upravljanje trofaznim BLDC motorom bez senzora

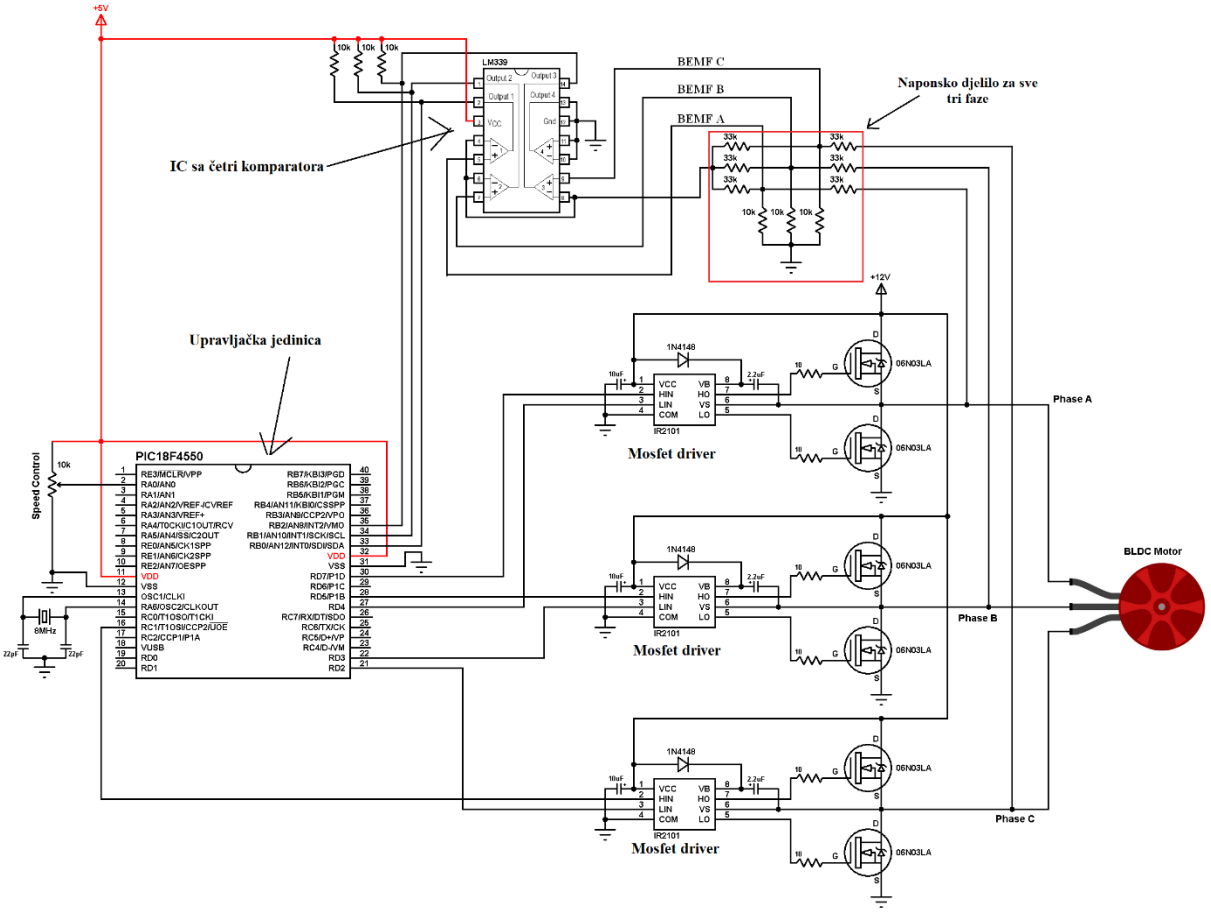
Prednost upravljanja BLDC motorom bez korištenja senzora su: veća pouzdanost, manja osjetljivost na smetnje, kompaktniji motor, manje ožičenje motora itd. Upravljanje BLDC motorom bez senzora se zasniva na detektiranju prelaza povratne elektromotorne sile kroz nulu (eng. Zero-crossing), drugim riječima prijelaz ih HIGH stanja u LOW stanje i obratno. Do povratne elektromotorne sile (BEMF) dolazi kada je namot statora u beskontaktnom float stanju, a pokraj njega prijeđe permanentni magnet rotora i tako inducira elektromotornu silu. BEMF je proporcionalan brzini rotora što znači da se rotor mora rotirati minimalnom brzinom kako bi generirao BEMF koji bi se mogao koristiti za upravljanje motorom. Minimalna brzina rotacije rotora ovisi o konstrukciji motora, snagi permanentnih magneta na rotoru, zračnom prostoru između rotora i statora itd [2]. Ako je signal BEMF-a jako slab mogu se koristiti pojačala kako bi ga upravljački sklop mogao detektirati



Slika 2.10 Usporedba izlaza Hall senzora i BEMF signala[1]

Na slici 2.10. može se vidjeti kako usporedba BEMF signala i stanja HALL senzora. Također se može vidjeti da za svaki uzlazni i silazni brid HALL senzora, BEMF signal prolazi kroz nulu [1].

Za detektiranje prijelaza BEMF signala kroz nulu može se koristiti metoda sempliranja koja se provodi tako da se semplira signal namota kada je namot u float stanju. Zbog većeg napona napajanja motora od upravljačke jedinice kao što je npr. mikroprocesor potrebno je BEMF signal skalirati kako se ne bi oštetila upravljačka jedinica. Skaliranje BEMF signala se najčešće provodi s naponskim djeliteljem realiziranim s dva otpornika. Druga metoda za detektiranje prijelaza BEMF signala kroz nulu je s korištenjem komparatora. Na taj način dobiva se pravokutni signal gotovo jednak signalu HALL senzora. Prednost korištenja komparatora je u tome što direktnim sempliranjem BEMF signala može dovest do resetiranja upravljačke jedinice zbog naglih skokova napona. Zbog tog razloga na najčešće se koristi kombinacija obje metode čija je shema prikazana sna slici 2.11.

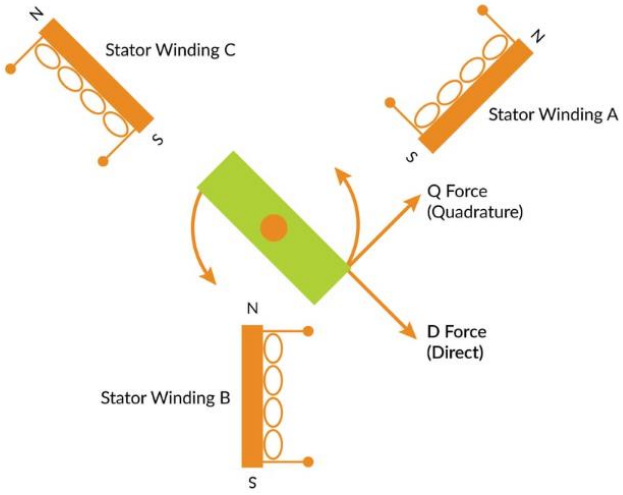


Slika 2.11 Shema upravljanja BLDC motorom bez senzora[3]

2.4.5. FOC upravljanje BLDC motorom

FOC (eng. Field Oriented Control) također poznata kao vektorska kontrola pruža bolju učinkovitost, veći okretni moment i veću brzinu rotacije od ostalih metoda upravljanja. FOC upravljanje je najsloženiji način upravljanja i zbog toga zahtjeva najviše procesorske snage. FOC upravljanje pruža bolje performanse kod dinamičkih promjena opterećenja u usporedbi s drugim metodama [4].

Ovisno o tome kako se pokreću pojedinačni namoti, oni mogu međusobno djelovati stvarajući silu koja ne stvara rotacijski moment rotora ili mogu stvarati silu koja stvara rotaciju rotora. Ove dvi različite sile poznate su kao kvadraturne (Q) i direktne (D). Za generiranje rotacije potrebno je maksimizirati kvadraturnu silu (Q) i minimiziranje direktne sile (D) [5].



Slika 2.12 Kvadraturna (Q) i direktna (D) sila

FOC način upravljanja povećava iskoristivost motora na 97% što je znatno bitno kada se motor napaja iz baterija. Kontrolerima koji koriste FOC nisu nužno potrebni HALl senzori jer oni konstantno nadziru struju i napon na svakoj pojedinoj fazi motora i tako točno izračunavaju poziciju rotora.

3. CAN BUS PROTOKOL

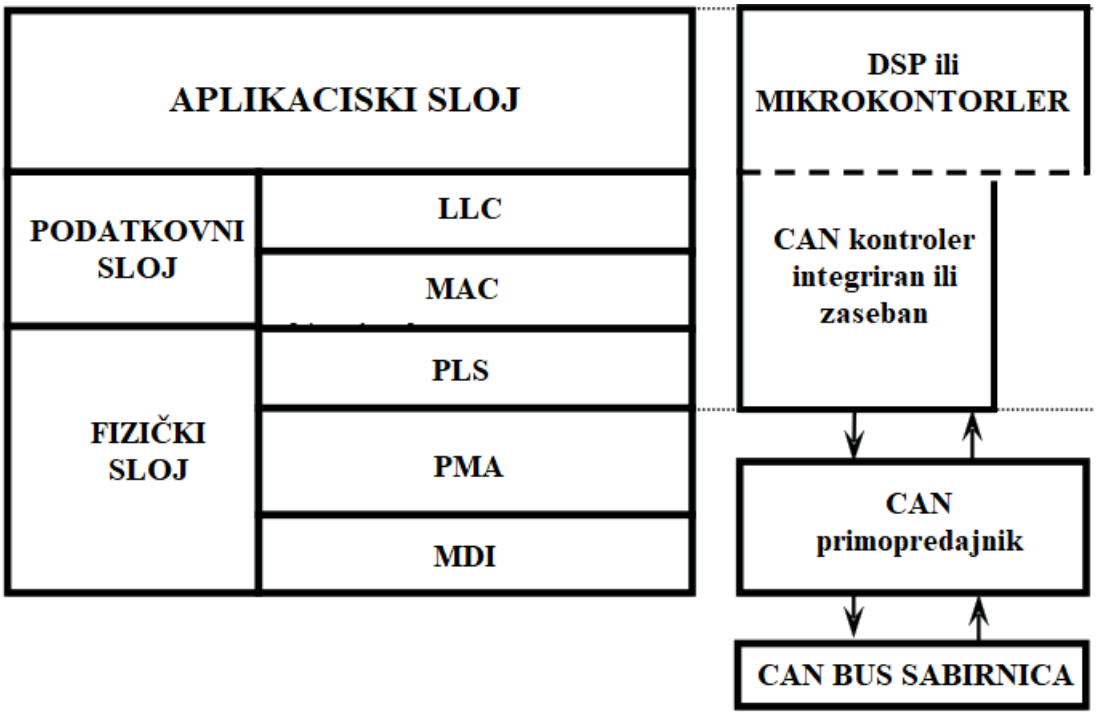
3.1. Općenito o CAN Bus protokolu

CAN Bus protokol razvila je kompanija BOSCH 1986. godine, a protokol je standardiziran 1993. godine pod oznakom ISO 11898. Izvorno je CAN Bus protokol namijenjen za automobilsku industriju da zamjeni složeni sustav ožičenja sa dvožilnom sabirnicom. Za razliku od tradicionalnih mreža kao što su USB ili Ethernet, CAN Bus ne šalje velike pakete podataka od jednog uređaja direktno ka drugom (point to point) već kratke poruke kao na primjer vrijednost temperature ili broj okreta u minuti (RPM) šalje svim uređajima spojenim na sabirnicu. Takav način komunikacije osigurava konzistenciju podataka na svakom priključenom uređaju. CAN Bus rad u multi-master broadcast načinu što znači da vi uređaji spojeni na sabirnicu mogu primati i slati podatke, ali ne istovremeno [11]. Poruke se ne razliku prema odredišnoj adresi već prema identifikatoru poruke koji ujedno i definira prioritet poruke. Neke poruke imaju veći prioritet od drugih na primjer poruke ispravnosti sigurnosnih sustava automobila imaju veću važnost od poruka za potrošenu količinu goriva u automobilu. Kod slučaja istovremenog slanja poruka s više uređaja dolazi kolizije koja se rješava postupkom arbitraže. Arbitraža osigurava da uređaj koji šalje podatke većeg prioriteta nastavi sa slanjem podataka. CAN protokol je "carrier-sanse", multiple-access with collision detection and i arbitražom po prioritetu poruke (CSMA/CD+AMP). CSMA znači da svaki uređaj spojen na sabirnicu mora sačekati propisano vrijeme neaktivnosti prije pokušaja slanja poruke. CD+AMP znači da se kolizija rješava arbitražom, na temelju unaprijed programiranog prioriteta svake poruke u indetifikaciskom polju poruke [11]. Svaki uređaj spojen na sabirnicu u postupku slanja podataka na sabirnicu u isto vrijeme i čita logičko stanje na sabirnici. U slučaju da uređaj na sabirnicu šalje recesivni bit odnosno logičko stanje high, a sabirnica je i dalje u dominantom stanju odnosno logičkom stanju low tada taj uređaj gubi u arbitraži i prestaje sa slanjem.

CAN standard ISO-11898:2003 ima standardni 11-bitni identifikator i osigurava brzinu prijenosa od 125 kbps do 1Mbps. Standardni identifikator je kasnije dopunjen s 29 bitnim identifikatorom. Standardni identifikator ima mogućnost za 2048 različitih poruka dok dopunjeni standard ima mogućnost za 537 milijuna različitih poruka.

3.2. CAN Bus i OSI model

CAN komunikacijski protokol, ISO-11898:2003, opisuje kako se informacije prenose između uređaja spojenih na sabirnicu i u skladu je s OSI (eng. Open Systems Interconnection) modelom koji je definiran u slojevima. Stvarna komunikacija između uređaja povezanih fizičkim medijem definirana je fizičkim slojem OSI modela. CAN protokol koristi najniža dva sloja OSI/ISO modela koji su podatkovni sloj i fizički sloj kao što je može vidjeti na slici 3.1. [11].



Slika 3.1 CAN Bus i OSI model [11]

- **Aplikacijski sloj** – predstavlja poveznicu CAN protokola sa operacijskim sustavom ili aplikacijom CAN uređaja [10].
- **Podatkovni sloj** – je zadužen za slanje i primanje podataka te provjeru ispravnosti podataka. Podatkovni sloj se sastoji od dva podsloja: LLC (eng. Logic Link Control) koji vrši funkcije kao što su izdvajanje podataka iz okvira, popravak greški i signalizira preopterećenje u komunikaciji. Drugi podsloj je MAC (eng. Media

Access Control) vrši kreiranje okvira poruke i potvrdu njihova prijema, vrši arbitražu pristupa sabirnici, te detekciju i signalizaciju greške [10].

- **Fizički sloj** – definira sklopovlje (eng. Hardwere), karakteristike električnog signala, način kodiranja i sinkronizacije, prijenosni medij itd. Podijeljen je na ti podsloja: PLS(eng.Physical Signaling) vrši kodiranje i sinkronizaciju, PMA (eng. Physical Medium Attachment) i MDI (eng. Medium Dependent Interface) definira prijenosni medij, konektore itd [10].

3.2.1. FIZIČKI SLOJ

3.2.1.1. Standardi fizičkog sloja

1. **Standard ISO 11898-2 high speed**

   High speed standard je najčešće korišteni standard CAN_BUS protokola. Podržava brzine do 1Mbit/s uz maksimalnu duljinu sabirnice od 40m. Linija je simetrična, a broj uređaja koji se mogu spojiti ovisi o električnom opterećenju sabirnice. Nominalno specificirano propagacijsko kašnjenje na liniji je 5ns/m [10].

2. **Standard ISO 11898-3 fault tolerant**

   Ovaj standard se koristi za elektroniku karoserije u automobilskoj industriji. Elektronika karoserije može uključivati razne mogule za upravljanje sustavima automobila kao što je npr. elektronički modul za multimediju, instrument ploču, navigaciju itd. Kod ovakvih mreža CAN_BUS sabirnica je kratka i utjecaj refleksije je gotovo zanemariv tako da nije potrebno zaključivati na krajevima. Moguće je koristiti i ostale topologije te ostvariti asimetričan prijenos podataka preko samo jednog vodiča. Ovaj standard podržava brzine od 125kbp/s i maksimalno 32 CAN_BUS uređaja.

3. **SEA J411 single wire**

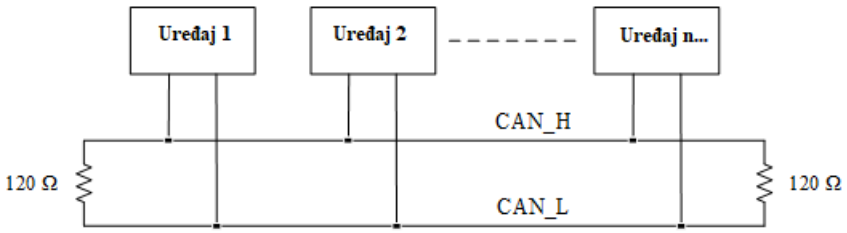
   Standard podržava brzine od 33.3 kbit/s uz maksimalno 32 čvora. Brzine od 83.3 kbit/s moguće je koristiti samo u dijagnostičkom modu. Kao prijenosni medij koristi se jedan vodič, a moguće je koristiti i druge topologije.

4. **STANDARD ISO 11992 point to popint (PtP)**

   Standard se koristiti za point to point (direktne) veze između dva CAN_BUS čvora. Podržava je brzina od 125kbit/s uz maksimalnu dužinu sabirnice od 40m. Ovaj protokol se često koristi za povezivanje prikolica na CAN_BUS liniju automobila.

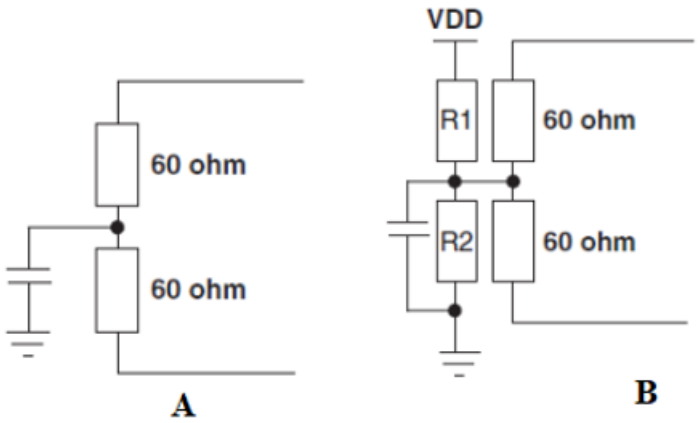
3.2.1.2. CAN Bus topologija i zaključenje linije

Uređaji su u CAN_BUS mreži povezani preko dvožične linije (sabirnice) koja je na svojim krajevima zaključena otpornicima tipične vrijednosti 120 Ω. Zaključni otpornici se koriste da smanje utjecaj refleksije signala. Prema standardu ISO 11898-2 zaključni otpornici trebaju imati vrijednosti od 100 do 130 Ω i minimalne disipacije snage od 220mW [10].



Slika 3.2. CAN_BUS topologija mreže[10]

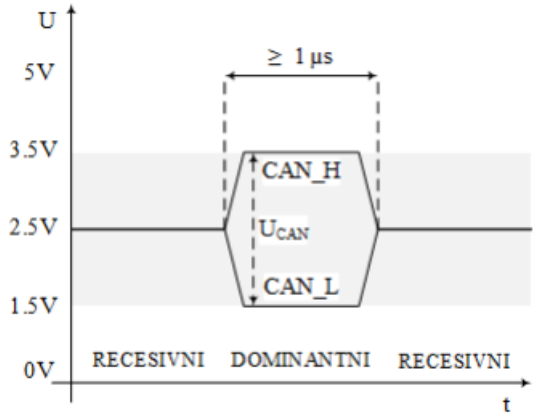
Osim standardnog zaključenja linije koristi se i razdjelno zaključenje (eng. Split termination) i prednaponsko razdjelno zaključenje (eng. Bias split termination). Razdjelno zaključenje je prikazano na slici 3.3 (a) i u odnosu na standardno zaključenje ima manje zračenje. Prednaponsko razdjelno zaključenje, slika 3.3 (b), je slično razdjelnom zaključenju osim što se naponsko djelilo i kondenzator koriste na oba kraja linije. Prednaponsko razdjelno zaključenje povećava performanse elektromagnetske kompatibilnosti linije, smanjiva osjetljivost na utjecaje elektromagnetskog zračenja [13].



Slika 3.3 Zaključenje CAN Bus linije (A) razdjelno, (B) Prednaponsko razdjelno zaključenje

3.2.1.3. CAN Bus signal

Signal CAN protokola je binaran tj predstavljen je s dva logička stanja, dominantnim stanjem koji odgovara logičkom stanju „0“ i recesivnim stanjem koji odgovara logičkom stanju „1“. Sabirnica CAN Bus protokola je simetrična što znači da su logička stanja predstavljena s razlikom napona oba vodiča, UCAN=UCAN_H – UCAN_L. Za vrijeme dominantnog bita naponska razlika signala CAN_H vodiča u odnosu na referenti napon od 3.5V, a na CAN_L vodiču iznosi 1.5V. Naponska razlika između CAN_H i CAN_L vodiča za vrijeme recesivnog bita jednaka je 0V no u usporedbeni s referentnim naponom razlika oba vodiča je 2.5V [10].



Slika 3.4 CAN signal [10]

Zbog simetričnosti linije signal je manje osjetljiv na smetnje budući da smetnje utječu jednako na oba signala tako da razlika napona ostaje nepromijenjena. Minimalni napon detekcije dominantnog bita je 0.9V, a detekcije recesivnog bita je određen naponom linije manjim od 0.5V [10].

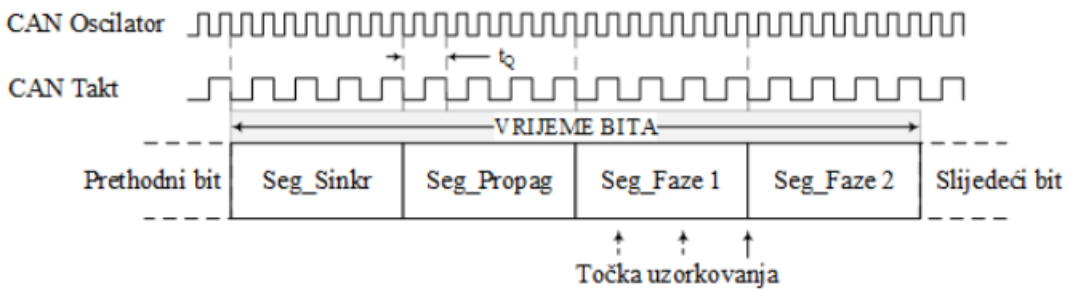
Kada je na CAN_BUS sabirnicu povezano više uređaja napon sabirnice je definiran logičkim „I“ (eng.AND) stanjem svim izlaza spojenih uređaja. Dovoljan je jedan uređaj da šalje dominanti bit i linija će biti u dominantom stanju, a da bi sabirnica bila u recesivnom stanju svi izlazi spojenih uređaja moraju biti također u recesivnom stanju.

U CAN_BUS protokolu se koristi NRZ (eng.Non-Return-To-Zero). NRZ kodiranje je način predstavljanja logičkih stanja s promjenama u razini signala, promjena signala može predstavljati logičko stanje „1“ dok ne promjena signala može predstavljati logičko stanje

„0“. Kod NRZ načina kodiranja dolazi do problema sinkronizacije kod dužih nizova istih logičkih stanja. Prema specifikaciji CAN_BUS protokola svaka pojava više 5 uzastopnih istih logičkih stanja predstavlja grešku. Kako bi se osigurala sinkronizacija kao i mogućnost slanja dužih nizova istih logičkih stanja koristi se postupak umetanja bita (eng. Bit Suffing). Svakih 5 uzastopnih istih logičkih stanja umeće se suprotno logičko stanje. Umetnuti bitovi se pri primitku uklanjaju i cisti podaci se procesuiraju dalje.

3.2.1.4. Struktura bit i sinkronizacija

Svaki uređaj spojen na CAN sabirnicu uspoređuje poslani bit sa stanjem signala na sabirnici. Da bi to bilo moguće nužno je da vrijeme propagacije signala od uređaja koji šalje do najudaljenijeg uređaja i nazad bude unutar trajanja jednog bita. Svi uređaji spojeni na sabirnicu moraju vršiti stalno sinkronizaciju internog takta u odnosu na poslani niz. Takt se sinkronizira stalnim podešavanjem točke uzorkovanja unutar svakog bita. Da bi se ostvarila sinkronizacija svaki je bit podijeljen u 4 segmenta, kao što je prikazano na slici 3.5. [10].



Slika 3.5 Struktura bita u CAN Bus protokolu

- **Segment sinkronizacija (Seg_Sinkr)** - Ovaj segment se koristi za sinkroniziranje uređaja spojenih na CAN Bus mrežu. Sinkronizacija je ostvarena u odnosu na početni brid impulsa, koji se mora pojaviti unutar ovog segmenta. Svako odstupanje brida se detektira od svih uređaja i svaki uređaj u skladu s odstupanjem podešava svoje fazne segmente [10].
- **Propagaciski segment (Seg_Propag)** – koristi se za kompenzaciju kašnjenja signala na sabirnici. Vrijeme ovog segmenta mora biti dvostruko veće od trajanja kašnjenja kako bi se uzelo u obzir kašnjenje signala u oba smjera [10].

- **Segment faze 1 i faze 2 (Seg_Faze 1, Seg_Faze 2) -** služi za kompenzaciju faznih odstupanja bridova signala. Uzorkovanje bita ostvaruje se na kraju segmenta faze 1 i u toj točki se određuje naponska razina bita[10].

Vrijeme bita je određeno brzinom prijenosa signala prema izrazu $t_{BIT}=1/V_{BAUD}$. Veća brzina prijenosa rezultira kraćim trajanjem bita i obrnuto. Svi uređaji spojeni na CAN sabirnicu moraju koristiti isti takt neovisno o internoj frekvenciji oscilatora.

Sinkronizacija je ostvarena na dva načina, metodom sinkronizacije i resinkronizacije bita. Kod sinkronizacije interni takt prijemnog uređaja se postavlja na početnu vrijednost na silaznom bridu početnog bit okvira (SOF bit). Resinkronizacija bita je ostvarena podešavanjem duljine segmenta faze 1 i segmenta faze 2 čime se korigira mjesto uzorkovanja [10].

3.2.2. PODATKOVNI SLOJ

3.2.2.1. Struktura CAN_BUS okriva

Postoje dva formata okvira za slanje poruka:

1. CAN 2.0 A standardni ili osnovni format (engl. Base frame)
2. CAN 2.0 B proširenI format (engl. Extended frame)

Glavna razlika između standardnog i proširenog CAN_BUS okvira je u polju identifikatora. Osnovni format koristi 11-bitni identifikator, a prošireni 29-bitni identifikator koji je razdvojen na dva dijela: 11 bita od standardnog identifikatora plus dodatnih 18 bita. Razlikovanje ova dva formata ostvareno je korištenjem IDE bita koje je opisano u daljnjem tekstu [10].

3.2.2.2. Standardni okvir

	Arbitražno polje		Kontrolno polje							
SOF	11-bitni identifikator	RTR	IDE	r0	DLC	Podaci	CRC	ACK	EOF	IFS
1 bit	11 bita	1 bit	1 bit	1 bit	4 bita	0 – 8 bajta	16 bita	2 bita	7 bita	3 bita

Slika 3.6. Standardni okvir – 11 bitni identifikator [10]

SOF – Dominanti bit označava početak slanja okvira, a ujedno se koristi i za sinkronizaciju čvorova. CAN_BUS čvor može početi slati poruke tek nakon što detektira neaktivnost linije u vremenu trajanja 11 recesivnih bita u koje su uključeni ACK bit razdvajanja, 7 bita EOF i 3 bita IFS.

Arbitražno polje:

Određiva u postupku arbitraže koja će se poruka prije poslati i sastoji se od dva dijela:

- **Identifikator** – sastoji se od 11 bita. Predstavlja identifikator poruke i definira njezin prioritet. Prioritet je veći što je binarna vrijednost identifikatora manja.
- **RTR** (engl. Remote Transmission Request) – sastoji se od 1 bit-a, a koristi se za razlikovanje okvira podataka od okvira upita. Za označavanje okvira s podacima koristi se dominanti bit.

Kontrolno polje:

Određuje format okvira i duljinu polja podataka.

- **IDE** (engl. Identifier Extension) – sastoji se od 1 bit-a, a i koristi se za razlikovanje standardnog i proširenog formata identifikatora. Za standardni format se koristi dominanti bit.
- **r0** – sastoji se od 1 bita. Bit rezerve za buduće namjene.
- **DLC** (engl. Data Length Code) – sastoji se od 4 bita i predstavlja broj bajta u polju podataka.

Polje podataka – sastoji se od 0-8 bajta. Podaci koji se žele poslat.

CRC (engl. Cyclic Redundancy Check) – sastoji se od 16bit-a od kojih je 15 bita CRC niza i 1 CRC bit razdvajanja (CRC delimiter). Provjera greške u prijenosu ostvarena je CRC proračunom nad svim prethodnim bitovima okvira (od SOF do polja podataka). Proračun se vrši na strani predaje i prijema gdje se uspoređiva sa primijenim CRC nizom. CRC bit razdvajanja je uvijek recesivan i ima ulogu da omogući dodatno vrijeme potrebno za proračun.

ACK (engl. Acknowledge) – sastoji se od 2 bita i predstavlja ispravno primjenu poruku. Sastoji se od ACK bita i ACK bita razdvajanja. ACK bit se uvijek šalje recesivan i svaki čvor koji ispravno primi okvir zamijenit će ga u okviru dominantnim i time potvrditi da je okvir primljen bez greške. Ako je okvir primljen s greškom ACK bit će ostati nepromijenjen i okvir se neće prihvatiti. U slučaju greške čvor će ponoviti slanje okvira. Bit razdvajanja je uvijek recesivan i neophodan je kako bi se naznačila razlika između potvrde prijema i eventualno generiranog okvira greške.

EOF (engl. End of frame) – sastoji se od 7-bita i označava kraj okvira predstavljen nizom od 7 recesivnih bita.

IFS (engl. Intermission Frame Space) – sastoji se od 3 bita i predstavljeno je nizom od 3 recesivna bita i određuje minimalni broj bita kojim su razdvojena dva uzastopna okvira od kojih je drugi okvir podataka ili okvir upita. Unutar ova tri bita dozvoljena je samo signalizacija statusa preopterećenja.

3.2.2.3. Prošireni okvir

U odnosu na standardni okvir prošireni okvir je nadopunjen s 18 bit-a identifikatora i dva dodatna polja.

	Arbitražno polje					Kontrolno polje						
SOF	11-bitni identifikator	SRR	IDE	18-bitni identifikator	RTR	r1	r0	DLC	Podaci	CRC	ACK	EOF	IFS
1 bit	11 bita	1 bit	1 bit	18 bita	1 bit	1 bit	1 bit	4 bita	0 – 8 bajta	16 bita	2 bita	7 bita	3 bita

Slika 3.7. Prošireni okvir, 29 bitni identifikator [10]

SRR (engl. substitute remote request) – sastoji se od 1 bita. Zamijenio je RTR bit u proširenom formatu i uvijek je u recesivnom stanju. Zajedno s IDE bitom koji je recesivan za prošireni format okvira osigurava da je poruka koja se šalje standardnim okvirom uvijek većeg prioriteta.

r1 – sastoji se od 1 bita i predstavlja dodatni rezervni bit za buduće namjene. Uvijek je postavljen kao dominantan uz mogućnost da se može koristiti i kao recesivan.

Arbitražno polje je prošireno na 32 bita u skladu s 29 bitnim identifikatorom, te dodatno uključuje SRR, IDE i RTR bit.

Na CAN_BUS liniji mogu postojati poruke različitih formata stoga uvijek veći prioritet moraju imati poruke standardnog formata aku su im osnovni identifikatori jednaki. CAN 2.0B kontroleri koji podržavaju prošireni format okvira mogu slati i primati poruke standardnog formata. CAN 2.0A kontroleri koji podržavaju standardni format ne mogu prepoznati poruke proširenog formata. Pored ovih postoje i CAN 2.0B pasivni kontroleri koji podržavaju poruke standardnog formata, a također mogu prepoznati i one proširenog formata ali ih ignoriraju [10].

3.2.3. Vrste okvira

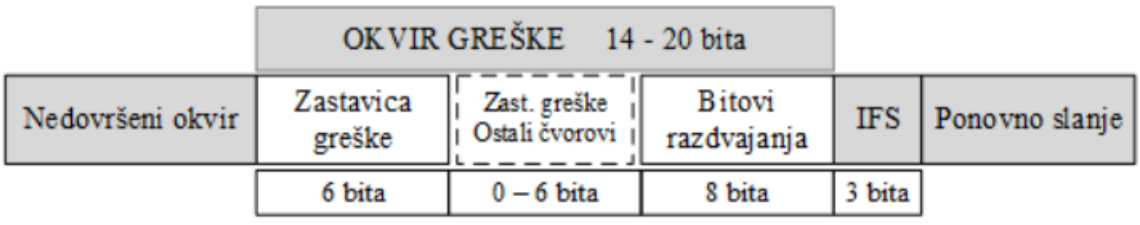
CAN_BUS protokol koristi četiri vrte okvira koji se mogu koristiti u komunikaciji.

Okvir podataka (engl. data frame) - okvir podataka omogućuje razmjenu podataka između čvorova. Okvir podataka označava se dominantnim RTR bitom.

Okvir upita (engl. Remote Frame) - okvir upita omogućuje čvoru da pošalje zahtjev za određenu poruku drugom čvoru. U odgovoru podaci se šalju preko okvira podataka. Okvir upita i okvir odgovora u tom slučaju moraju imati jednaku vrijednost identifikacijskog i DLC polja. Okvir upita za razliku od okvira podataka ne sadrži polje podataka a RTR bit je recesivan. Budući je RTR bit uključen u polje arbitraže kod istovremenog slanja okvira upita i okvira podataka zbog dominantnog RTR bita okvir podataka ima veći prioritet [10].

Okvir greške (engl. Error Frame) – okvir greške šalje čvor koji detektira grešku u prijenosu podataka. Okvir se šalje odmah nakon detekcije greške što dovodi do prekida trenutnog prijenosa podataka. Slanje okvira greške na sabirnicu uzorkuje i detekciju greške svih ostalih čvorova koji radi toga šalju svoje okvire greške. Okviri grešaka ostalih čvorova mogu biti poslani u isto vrijeme ili nakon detekcije prvog okvira greške što može dovesti do toga da polje zastavice greške bude prošireno do 12 bita. U slučaju greške pošiljatelj ponavlja slanje poruke.

Svaki CAN kontroler sadrži dva brojača grešaka (brojač grešaka slanja i brojač grešaka prijema) koji osiguravaju da čvor ne zauzme sabirnicu stalnim slanjem okvira grešaka. Ako brojač grešaka slanja pređe vrijednost 255 čvoru se nadalje dozvoljava samo prijem poruka bez mogućnosti slanja. Čvor se ponovno može osposobiti za slanje tek ponovnim pokretanjem ili ako primi 128 puta niz od 11 uzastopnih recesivnih bita.



Slika 3.8. Okvir greške [10]

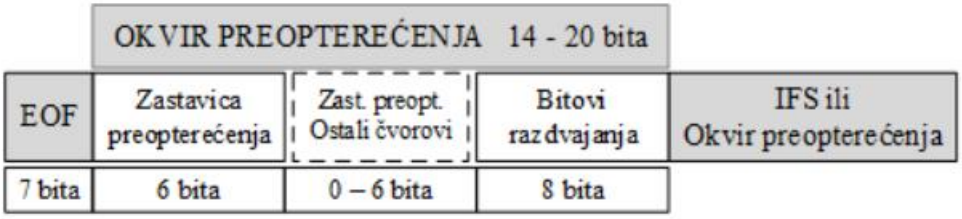
Okvir preopterećenja (engl. overload frame)

Okvir preopterećenja šalje čvor koji nije spreman primiti podatke i slanjem okvira preopterećenja signalizira da mu se okvir pošalje sa zakašnjenjem. Kao i kod okvira greške slanje okvira preopterećenja na sabirnicu uzorkovat će slanje okvira preopterećenja svih ostalih čvorova. Slanje okvira preopterećenja ostalih čvorova uzorkovat će da polje zastavica bude prošireno na 12 bita. Maksimalno je dozvoljeno slanje dva uzastopna okvira preopterećenja [10].

Struktura okvira preopterećenja jednaka je kao i kod okvira greške i sastoji se od:

- **Polje zastavice preopterećenja** (engl. Overload flag) – sastoji se od 6 do 12 bita prvih 6 bita su u dominantom stanju, a preostalih 6 predstavljaju proširenje okvira preopterećenja zbog suspenzije okvira preopterećenja ostalih čvorova.

- **Polje razdvajanja preopterećenja** (engl overload delimiter) sastoji se od 8 uzastopnih recesivnih bita, a omogućuju sinkronizaciju čvorova prema prvom bitu ovog polja.



Slika 3.9. Okvir preopterećenja [10]

Slanje okvira preopterećenja može nastati kao posljedica dva slučaja. Prvi slučaj je zbog internih uvjeta u prijemniku zbog kojih čvor ne može trenutno primiti podatke. U ovom slučaju okvir preopterećenja šalje se odmah nakon završetka okvira umjesto IFS polja budući dominanti bitovi zastavice preopterećenja imaju prioritet nad bitima IFS. Drugi slučaj je kada se detektira dominanti bit za vrijeme IFS pauze ili za vrijeme završetka okvira [10].

3.2.4. Detekcija i signalizacija greške

CAN_BUS protokol sadrži 5 metoda provjere greške. Tri su na razini podatkovnog sloja tj na razini okvira, a ostale dvi na razini fizičkog sloja tj razini bita. Ako se detektira bilo koja greška poruka neće biti prihvaćena te će se generirati okvir greške. Čvor koji je poslao poruku će tada ponoviti slanje poruke sve dok se ona ne pošalje bez greške. Ako čvor zauzme sabirnicu stalnim slanjem okvira greške slanje će mu biti onemogućeno kada brojač grešaka poslanih okvira prijeđe vrijednost od 255.

Na razini podatkovnog sloja koriste se tri metode provjere greške:

CRC metoda (engl cyclic redundancy check) – koristi se CRC 15 bitni proračuna nad svim prethodnim bitovima okvira, a za proračun se koristi BCH (Bose Chaudhuri Hocquenghen code) algoritam i vrši se na strani prijema i predaje. CRC proračun se na prijemu izračuna i uspoređuje s poslanim ako nisu isti generira se okvir greške.

Greška potvrde prijema – čvor koji primi paket podataka mora to potvrditi ACK bitom u slučaju da nije pošiljatelj će generirati grešku

Greška formata okvira – ova se greška odnosi na bitove koji uvijek moraju biti u recesivnom stanju, a to su ACK, CRC bit razdvajanja, bitovi u polju završetka okvira EOF i bitovi u polju razdvajanja okvira IFS. Ako je jedan od navedenih bitova u dominantom stanju generirat će se greška. Izuzeće u ovoj provjeri se vrši za uvjete :

1. Ako je zadnji bit EOF dominatan
2. Ako je zadnji bit polja bitova razdvajanja okvira greške dominatan
3. Ako je zadnji bit polja bitova razdvajanja okvira preopterećenja dominantan

Na razini fizičkog sloja koriste se dvi metode za provjeru greške:

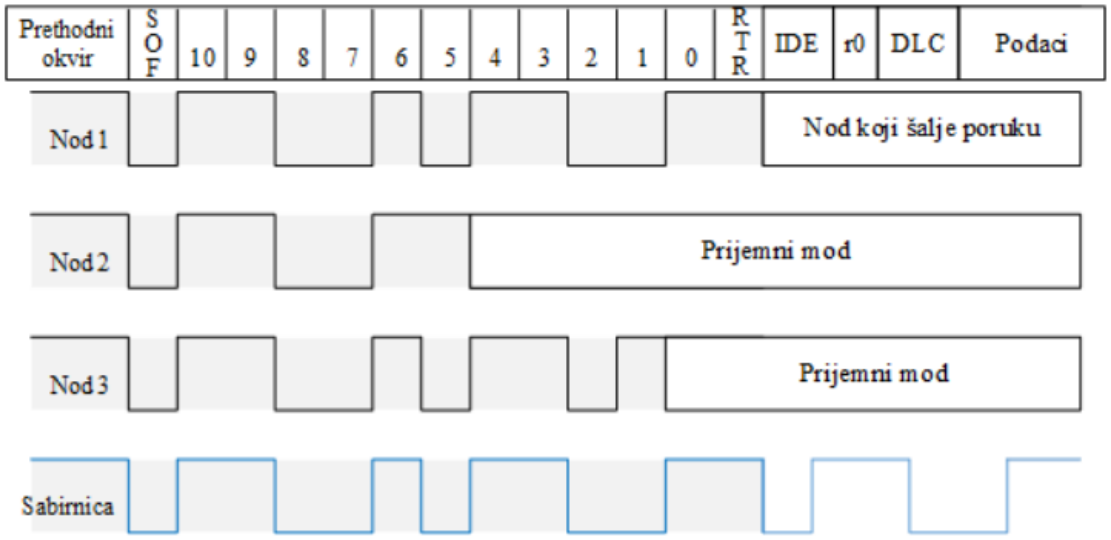
Nadzor bita (engl. Bit monitoring) – svaki čvor kod slanja bita provjerava razinu signala na sabirnici te je usporediva s poslanim bit-on. Ako se detektira razlika generirat će se greška. Izuzetak u ovoj provjeri su bitovi kod kojih je promjena bita uobičajena u postupku prijenosa, a to su bitovi koji se koriste u arbitraži, ACK bit i bitovi pasivne zastavice greške [10].

Greška umetanja bita (engl. Bit stuffing error) – ova greška se detektira ako se na sabirnici pojavio niz od 5 uzastopnih bita istog polariteta. Greška umetanja bita odnosi se samo na bitove okvira podataka i upita i to od početka SOF bita do uključujući bitova CRC polja. Greška se ne primjenjuj kod okvira greške i preopterećenja [10].

3.2.5. Arbitraža na sabirnici

Prijenos podataka u CAN_BUS protokolu je ostvaren kao poludupleksi prijenos što znači da u svakom trenutku može slati samo jedan čvor. Svi drugi čvorovi mogu te podatke primiti. Ako se desi da dva ili više čvora počnu slati podatke u istom trenutku dolazi do kolizije koja se rješava postupkom arbitraže. Pravo pristupa sabirnici određeno je prioritetom poruke koji je definiran identifikatorom poruke. Prioritet je veći što je binarna vrijednost identifikatora manja. Na taj će se način poruke većeg prioriteta prije poslati. CAN_BUS protokol je određeno da kod svakog slanja bita čvor provjerava razinu signala na sabirnici u odnosu na razinu signala kojeg je poslao. Budući su izlazni stupnjevi CAN primopredajnika spojeni u soju otvorenog kolektora dominanti bit bilo kojeg čvora na sabirnici će prevladati nad recesivnim bitom. Recesivno stanje sabirnice jedino je moguće kada su svi izlazi čvorova u recesivnom stanju.

Arbitraža se provodi za svaki bit posebno unutar bitova arbitražnog polja. Svi čvorovi u postupku arbitraže će slati svoju poruku sve dok su im međusobno bitovi identični. U trenutku kada jedan od čvorova pošalje recesivni bit, a u istom trenutku postoji jedan ili više čvorova koji šalju dominantni bit, čvorovi s recesivnim bitom će prestati slati. Čvor koji je prestao slati prelazi u prijemni mod, a poruku može poslati već u sljedećem ciklusu slanja okvira. Postupak arbitraže nastavlja se sve dok na sabirnici ne prevlada samo jedan čvor s porukom najvećeg prioriteta [10]. Postupak arbitraže se može vidjeti na slici 2.10.



Slika 3.10. Arbitraža na CAN_BUS sabirnici [10]

4. ESP32

4.1. Općenito o ESP32 SOC

ESP32 je „tzv“ sustav na čipu (engl. System on Chip) što bi značilo da sadrži sve potrebne računalne periferije potrebne za rad. Neke od tih periferija su CPU, memorija, ulazno/izlazni moduli itd. ESP32 je razvijen od kompanije Espressif Systems koja je zaslužna za proizvodnju popularnog ESP8266 SoC. ESP32 dolazi u varijantama jedno jezgrenog i dvo jezgrenog Tensilica 32-bit Xtansa LX6 mikroprocesora s integriranim WI-FI i bluetooth periferijama. Dobra stvar kod ESP32 je što su integrirane RF komponente kao što su pojačalo snage, niskošumno prijamno pojačalo, antenski razdjelnik, filtri i RF balun. To čini dizajniranje hardvera oko ESP32 vrlo lakim jer je potrebno vrlo malo vanjskih komponenti[16].

Jedna od zanimljivih periferija ESP32 SoC je hardverski kripto akcelerator. Njegova glavna uloga je ubrzavanje obrade kriptografskih podatka i upravljanje s kriptografskim ključevima. Periferija podržava rad s SHA(engl. Secure Hash Algorithms), RSA(engl. Rivest Shamir Adleman) i AES (engl. Advanced Encryption Standard) algoritam. Također sadrži generator slučajnih brojeva.

ESP32 je jako brz SoC i njegov radni takt je 240MHz što ga u kombinaciji s dvojezgrenim procesorom čini prikladnim za implementaciju sa RTOS (engl. Real time operating system) sustavom.

ESP32 podržava većinu komunikacijskih protokola koji se koriste u mikroprocesorima i elektroničkim modulima , a to su SPI, I2C, I2S, UART i TWAI. Za neke od komunikacijskih protokola postoji više periferija što omogućava bržu komunikaciju i lakšu implementaciju. Također podržava WIFI i bluetooth tehnologiju što ESP32 čini prikladnim za izradu IoT (engl. Internet of thing) uređaja.

4.2. Karakteristike i prikaz izvoda ESP32 SoC

Neke od karakterisktika ESP32 SoC:

CPU i memorija:

- Xtensa single-/dual-core 32-bit LX6 microprocessor(s)
- 448 KB ROM
- 520 KB SRAM
- 16 KB SRAM in RTC
- QSPI podrška za više flash/SRAM čipova

Oscilatori i brojači:

- Unutrašnji 8 MHz ocilator sa kalibracijom
- Unutrašnji RC oscilator sa kalibracijom
- Vanjski 2Mhz – 60Mhz kristalni ocilator (40Mhz samo za WI-FI/Bluetooth)
- Vansjki 32kHz kristalni oscilator za RTC sa kalibracijom
- Dvi grupe timer-a, 2 x 64 bit rimer i 1x glavni „watchdog“ u svakoj grupi
- RTC timer
- RTC watchdog

Napredna periferna sučelja:

- 34 x programabilna GPIO pina
- 18 x 12 bit SAR ADC
- 2 x 8 bit DAC
- 10 x senzor dodira
- 4 x SPI
- 2 x I2S
- 3 x UART
- 1 host (SD/eMMC/SDIO)
- 1 slave (SDIO/SPI)
- Ethernet MAC s podrškom za DMA i IEEE 1588
- TWAI kompatibilan sa ISO 11898-1 (CAN Specifikacijom 2.0)
- 16 x PWM kanala

Upravljanje napajanjem:

- Ko procesori ultra male potrošnje (ULP)
- Potrošnja energije u „Deep-sleep“ modu 10 μA
- RTC memorija ostaje na napajanju u „Deep-sleep“ modu
- Pet modova potrošnje: Aktivni, Modem-sleep, Light-sleep, Deep-sleep, Hibernacija

Sigurnost:

- Sigurni boot
- Flash širfiranje (engl. Encryption)
- Kripto hardverski akcelerator: AES, Hash (SHA-2), RSA, ECC, Generator nasumičnih brojeva (RNG)

WIFI:

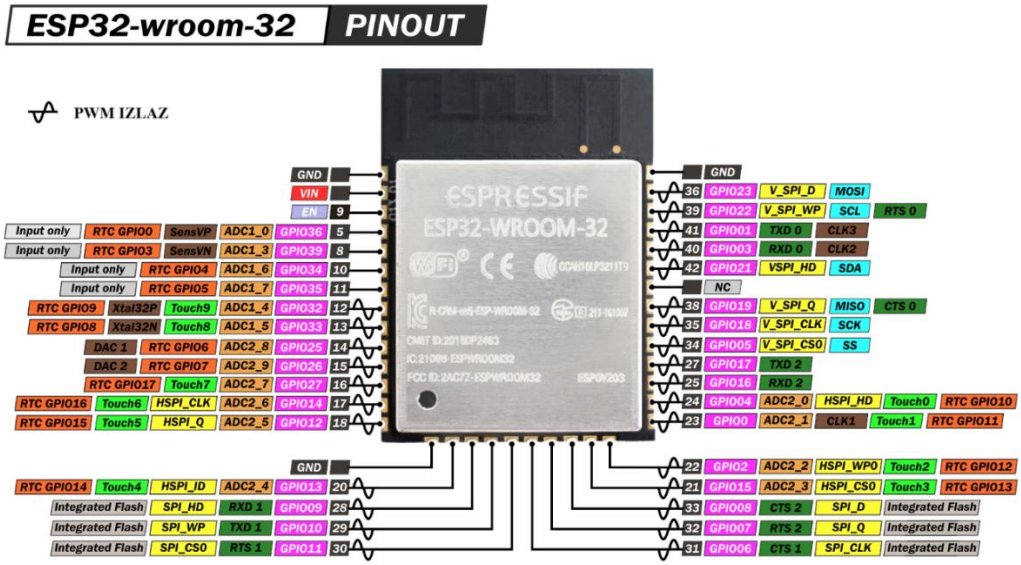
- 802.11b/g/n
- 802.11n (2.4 GHz), do 150 Mbps
- WMM
- TX/RX A-MPDU, RX A-MSDU
- Defregmentacija
- 4 × virtual Wi-Fi interfaces

Bluetooth:

- Sukladno sa Bluetooth v4.2 BR/EDR i Bluetooth LE specifikacijama
- klasa-1, klasa-2 i klasa-3 odašiljača bez vanjskih pojačala snage
- +9 dBm odašiljačke snage
- NZIF risiver sa –94 dBm Bluetooth LE osljetljivosti
- High-speed UART HCI, up to 4 Mbps

Ostale karakteristike i specifikacije ESP32 SoC mogu se pronaći u tehničkoj dokumentaciji na [14]

Na slici 4.1. može se vidjeti funkcija svakog pojedinog izlaznog pita. Većina pinova ima više funkcionalnosti stoga je pri programiranju potrebno specificirati koju od funkcionalno se koristi.



Slika 4.1 Prikaz izlaza ESP32 SoC [16]

ESP32 SoC s dvojezgrenim procesorom koji podržava do 32 prekida po jezgri, a jednojezgreni procesor podržava 32 prekida. Prekidi se dijele na softverske i hardverske prekide. Softverski prekidi su interni prekidi koji se javljaju kao odgovor na izvršavanje softverske instrukcije kao npr. mjerač vremena se može koristiti za generiranje prekida. Hardverski prekidi su vanjski prekidi i njih uzrokuje neki vanjski događaj kao npr. tipkalo, senzor pokreta itd. Hardverski prekidi mogu reagirati na različite vrste signala kao što su rastući brid, padajući brid, logičko stanje 1, logičko stanje 0 i na promjenu iz jednog logičkog stanja u drugo. Hardverski prekid mora biti specificiran u programskom kodu i to na način da se definira pin koji se koristi i na koju vrstu signala će reagirati [19].

ISR (engl. Interrupt Service Routine) - kada dođe do prekida prilikom normalnog izvršavanja programa poziva se ISR ili upravljač prekidima. Normalno izvođenje programa bit će zaustavljeno, a prekid će se izvršiti na temelju razine prioriteta prekida. Nakon što se izvrši prekid program nastavlja s radom gdje je prekid nasta[19].

Kod ESP32 IRAM_ATTR atribut treba definirati za rukovanje s prekidima jer bi se usluge prekida trebale izvoditi unutar RAM memorije budući da je RAM memorija brža za izvršavanje koda od FLASH memorije [19].

4.3. Programiranje ESP32 SoC

Programiranje ESP32 SoC je moguće s više programskih jezika kao što su C/C++, MicroPython, Lua. Kao programsko okruženje moguće je koristit Espressif IoT Development Framework (IDF), VSCode s PlatformIO i Arduino iDE. Espressif IoT Development Framework je službeno programsko okuženje za programiranje ESP32 SoC i u njemu je moguće koristiti se svim funkcijama koje ESP32 pruža. Arduino IDE ne podržava sve vrste ESP32 SoC ali njegova je prednost jednostavnije implementiranje funkcija koje ESP32 pruža. Arduino IDE iako pruža podršku za veliku većinu funkcija nekima se ipak ne može koristiti kao što je npr. direktno pristupanje registrima ESP32 što je nekad potrebno zbog brzine promjene izlaznih signala.

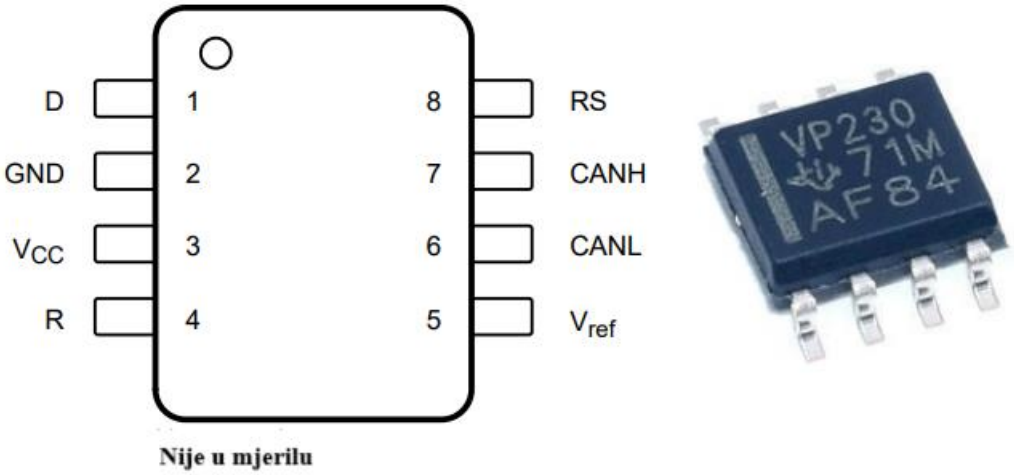
Tablica 4-1 Prednosti i manje ESP IDF u odnosu na ARDUINO IDE [21]

ESP IDF	**ARDUINO IDE**
Izvorna podrška za FreeROTS	Limitirana podrška za FreeRTOS
Aplikacije temeljene na zadacima	Setup() i loop() funkcije
Više jezgi postavljeno prema zadanim postavkama	Jedna jezgra postavljena prema zadanim postavkama
Podrška za nove ESP32 SoC	Limitirana podrška za nove ESP32 SoC
Manje prilagođen početnicima	Prilagođen početnicima
Manja zajednica korisnika	Veća zajednica korisnika

U tablici 3.1. mogu se vidjeti neke od prednosti i mana između trenutno dva najpopularnija programska sučelja za ESP32 SoC. U ovom radu će se koristiti Arduino IDE programsko sučelje zbog jednostavnijeg korištenja i veće dostupnosti primjera.

5. SN65HVD230 CAN_BUS primopredajnik čip

SN65HVD230 CAN_BUS primopredajnik je kompatibilan sa specifikacijama ISO 11898-2 High Speed standardom. Primopredajnik povezuje procesorsku jedinicu sa diferencijalnom CAN_BUS sabirnicom što bi značilo da SN65HVD230 radi na fizičkoj razini. SN65HVD230 je dizajniran da radi u teškim uvjetima i ima implementirane zaštite kao što su zaštita od prenapona, zaštita od previsoke temperature, zaštitu od kratkog spoja sabirnice itd. Dozvoljeni napon na CAN_BUS sabirnici je od -2V do 7V [22].



Slika 5.1 SN65HVD230 prikaz izlaza (označen s VP230) [22]

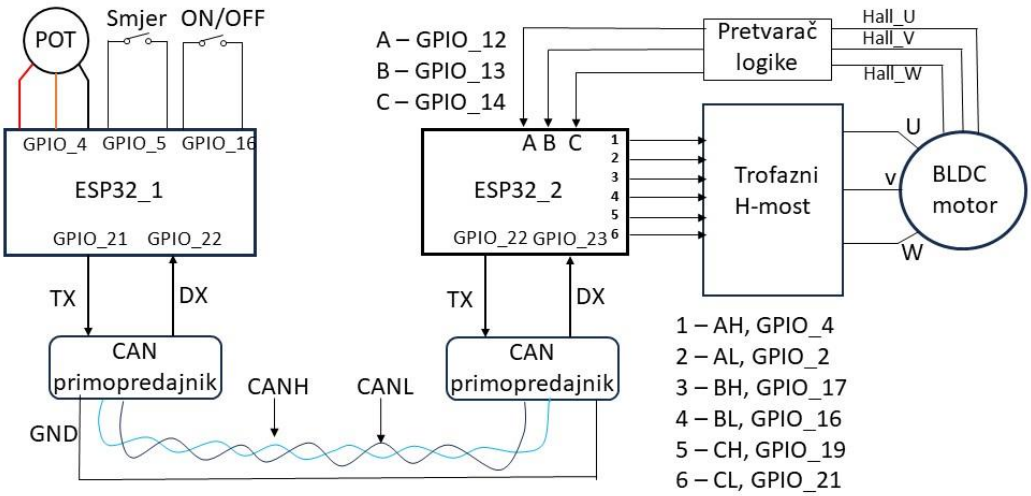
Opis pinova:

- D (1) – CAN predajni ulaz, logičko stanje 0 za slanje dominatnog bita, a logičko stanje 1 za slanje recesivnog bita. Ovaj pin se u praki jos naziva i TXD.
- GND(2) – priključak uzemljenja (minus pol izvora)
- Vcc (3) – 3.3V napajanje (plus pol izvora)
- R (4) – CAN prijemni izlaz, logičko stanje 0 za prijem dominatnog bita, a logičko stanje 1 za prijem recesivnog bita. Ovaj pin se u praki jos naziva i RXD.
- Vref (5) - Vcc/2 V
- CANL (6) – linija niske razine CAN sabirnice
- CANH (7) – linija visoke razine CAN sabirnice
- Rs (8) – Odabir moda rada (Rs na GND = high speed, Rs na Vcc = mode niske potrošnje, Rs pritegnut na GND s otpornikon vrijednosti 10kΩ do 100kΩ = slope mod)

6. PRAKTIČNI DIO

6.1. Zadatak završnog rada

Zadatak završnog rada je upravljanje BLDC motorom preko CAN_BUS protokola. Zadatak je izvršen uz pomoć dva ESP32 SoC mikrokontrolera tako da jedan ESP32 šalje podatke o brzini vrtnje motora, smjeru vrtnje, pokretanju i stajanju motora preko CAN_BUS protokola. Drugi ESP32 mikrokontroler će odrađivati komutaciju BLDC motora i obrađuje primjene podatke s CAN_BUS sabirnice. Brzina vrtnje motora se regulira s potenciometrom, a smjer vrtnje, pokretanje i zaustavljanje s prekidačem. Blok shema je prikazna na slici 6.1.



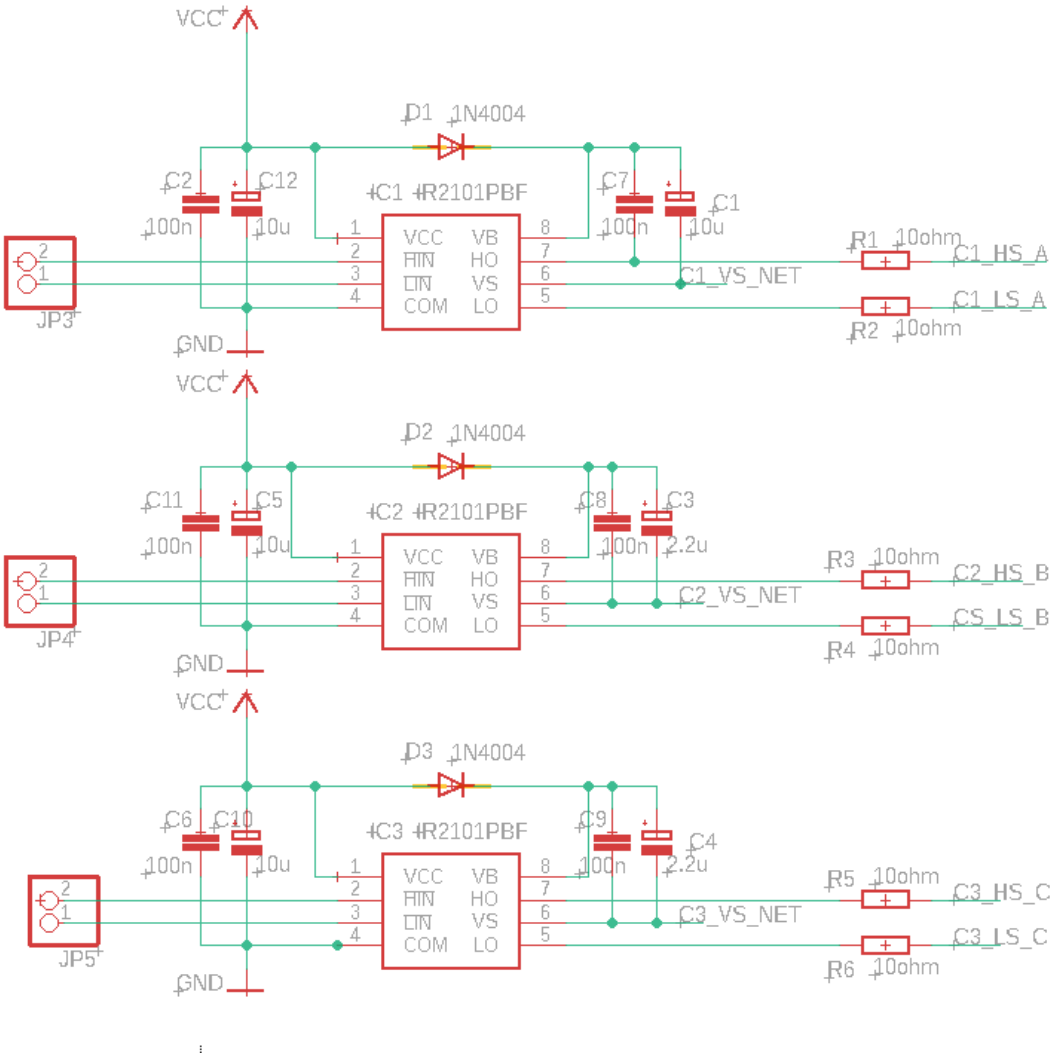
Slika 6.1 Blok shema sustava upravljanja BLDC motorom preko CAN_BUS protokola

Za CAN_BUS komunikaciju korišteni su SN65HVD230 CAN_BUS primopredajnici čiji je radni napon 3.3V i napajaju se direktno s ESP32 SoC razvojne pločice. Kao komunikacijska linija korištena je upletena parica s RJ45 konektorima na krajevima. Brzina prijenosa je podešena na 1000kbps na obje strane. Potenciometar je nazivne vrijednosti 1kΩ, a prekidači su pritegnuti na 3.3V preko inertnog otpornika u ESP32 mikrokontroleru i zatvaranjem prekidača šalje se logička 0 na GPIO pin.

ESP32_2 upravlja komutacijom BLDC motora uz pomoć tri hall senzora koja služe kao povratna veza pozicije rotora. Pretvarač logike je nužan jer hall senzori zahtijevaju 5V napajanje, a ESP32 radi na 3.3V.

6.2. Trofazni H-most

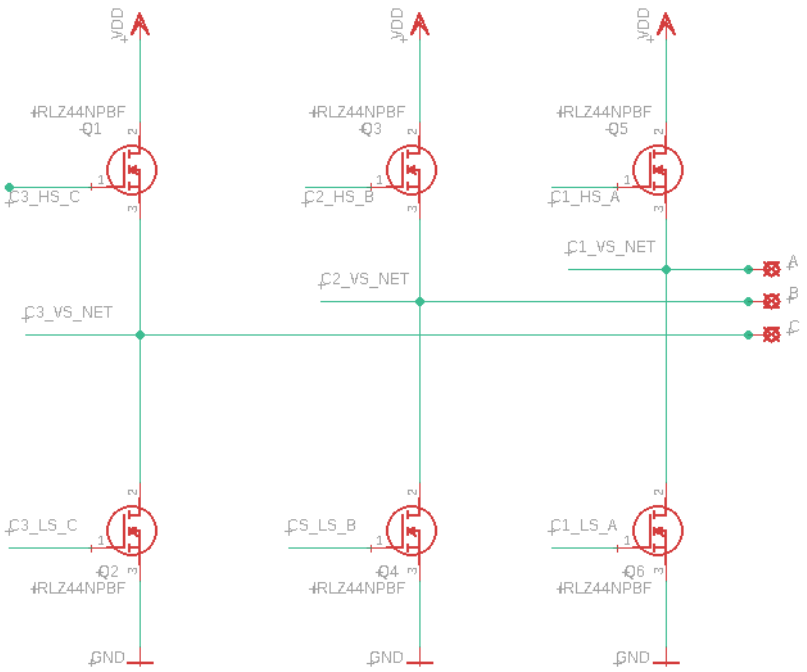
Trofazni H-most služi za upravljanje komutacijom BLDC motora. Uloga mu je omogućiti izmjenični protok struje kroz namotaje BLDC motora. Kod dizajniranja Trofaznog H-mosta treba voditi računa o upravljanju s mosfetima iako nije nužno da se moraju koristiti mosfet-ti moguće je koristiti i BJT tranzistore. Mosfet-ti imaju prednost kod upravljanja nad BJT tranzistorima jer mosfet-ti se upravljaju naponom dok se BJT tranzistori upravljaju strujom. Trofazni H-most se najčešće upravlja s mikrokontrolerom kao što je to slučaj u ovom radu. Izlaz mikrokontrolera nije u mogućnosti dati dovoljno struje za optimalan rad BJT tranzistora stoga bi to zahtijevalo dodatnih komponenti za upravljanje. Iako se mosfet-ti upravljaju naponom to ne znači da nema protoka struje naprotiv protok struje u GATE mosfeta je ključan za precizno upravljanje. GATE mosfet-a ima kapacitivnost u odnosu na SOURCE i zbog te kapacitivnosti potrebno je neko vrijeme da se kanal mosfet-a otvori. Kod isključenja mosfeta dolazi do obrnute situacije, GATE mosfeta se prazni i kanal se zatvara. Kako se ne bi dogodilo da gornji i donji mosfet istog reda (slika 5.2 npr. Q1 i Q2) uključeni u isto vrijeme zbog sporog punjena ili pražnjena kapacitivnosti potrebno je koristiti mosfet upravljač (engl. Mosfet driver). Upravljač omogućava brzo punjenje i pražnjene kapacitivnost GATE-a i tim ubrazava vrijeme otvaranja mosfet-a. N-kanalani mosfet nije moguće koristiti u prekidanju strujnog kruga prije potrošača iz razloga što se s istim izvorom napajanje ne može postići dovoljan napon između GATE-a i SOURCE-a mosfeta, kako bi se zaobišao ovaj problem koristi se „bootstrap“ sklop koji omogućuje da se dobije adekvatan napon između GATA-a i SOURCE-a za otvaranje kanala mosfeta. U ovom radu koristi se IR2101 upravljač mosfetima koji ima ugrađen bootstrap mehanizam.



Slika 6.2 Shema spajanja IR2101 mosfet upravljača

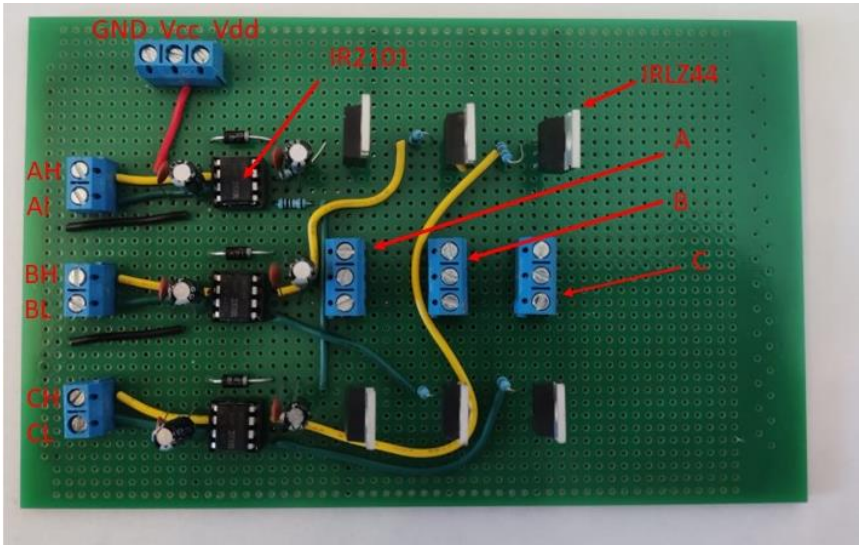
Na slikama 6.2. i 6.3. prikazana je shema spajanja mosfet upravljača i mosfeta. Konektori JP3, JP4 i JP5 služe za spajanje upravljačkih linija s mikrokontrolera u ovom slučaju s ESP32 SoC. Dovođenjem logičkog stanja 1 na pin 2 konektora JP3 otvara se kanal Q1 mosfeta, a dovođenjem logičkog stanja 0 zatvara se. Pin 1 JP3 konektora ima istu logiku upravljanja za Q2 mosfet. Isto vrijedi za ostale mosfet-e. Korišteni su IRLZ44 mosfet-ti koji su N-kanalog tipa iako su se za upravljanje gornjih sklopki mogli koristiti P-kanalni bolja je opcija korištenje N-kanalnih jer N-kanalni mosfeti imaju manje otpor između DRAN-a i SOURCE-a, npr. IRLZ44 ima otpor RDS 28mΩ pri naponu UGS=5V [23] dok IRF9510 P-kanalni

mosfet ima otpor RDS 1.2Ω pri naponu UGS= -10V [24]. Manji otpor RDS znači manji pad napona što rezultiranja manjih grijanjem i boljom učinkovitošću.



Slika 6.3 Trofazni H-most

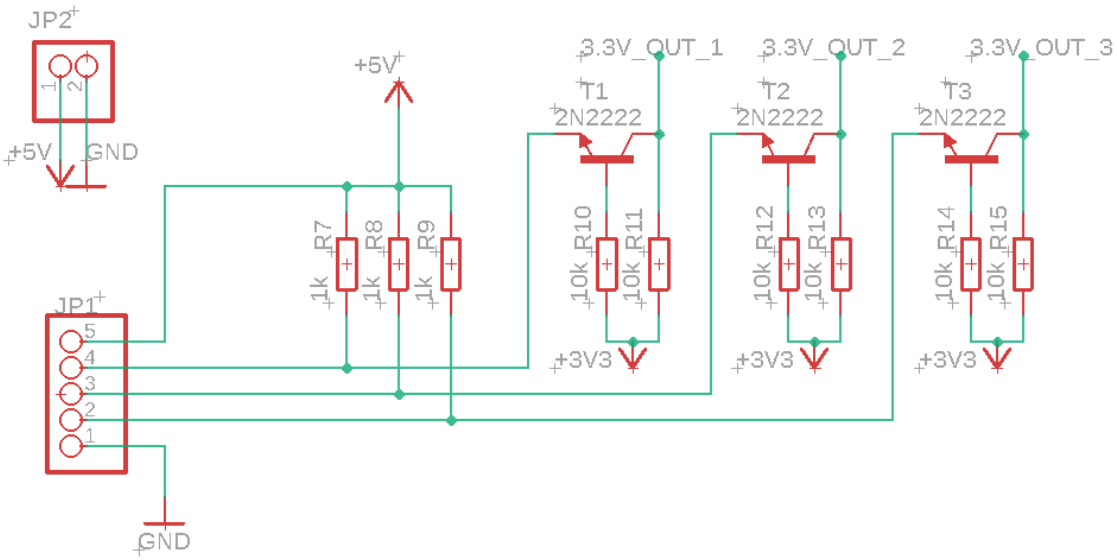
Na slici 6.4 se može vidjeti fizički napravljeni trofazni H-most s upravljačkim djelom. Kod izrade potrebno je voditi računa o presjeku vodiča protivnom može doći do izgaranja vodiča.



Slika 6.4 Trofazni H-most s upravljačkim dijelom

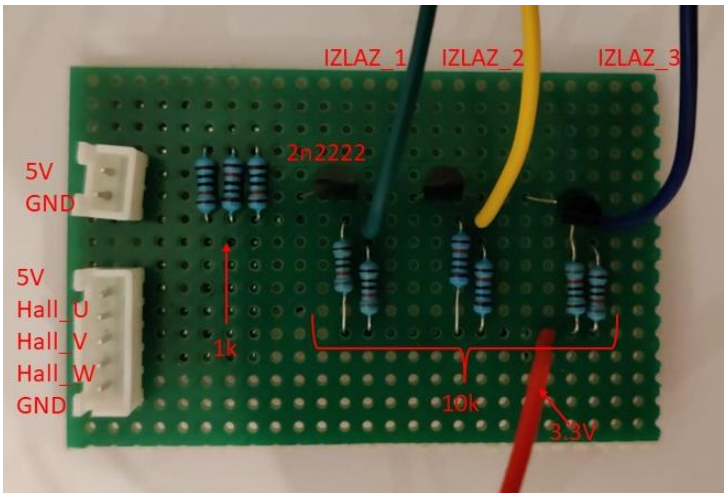
6.3. Pretvarač logike za 3 hall senzora

Pretvarač logike je potreban zato što hall senzori rade na 5V, a ESP32 mikrokontroler radi na 3.3V. Izlazi hall senzora da bi ispravno radili trebaju biti pretegnuti na 5V preko otpornika od 1kΩ. Na Slici 6.5 može se vidjeti shema spajanja pretvarača logike za 3 hall senzora.



Slika 6.5 Shema pretvarača logike za 3 hall senzora

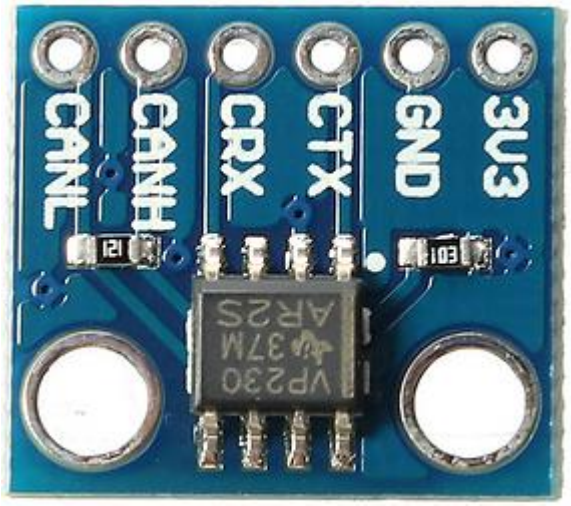
Dizajnirani pretvarač logike čija je shema prikazana na slici 6.5 radi samo u jednom smjeru. Na slici 6.6 se može vidjeti napravljeni pretvarač.



Slika 6.6 Pretvarač logike za 3 hall senzora

6.4. Spajanje CAN_BUS primopredajnika

Na slici je prikazan CAN_BUS primopredajnik koji je u ovom radu korišten za povezivanje ESP32 mikrokontrolera na CAN sabirnicu. Radni napon primopredajnika je 3.3V. Primopredajnik radi na način da pri dovođenju logičkog stanja 1 na CTX na CAN sabirnicu se šalje recesivno stanje, a slanjem logičkog stanja 0 na sabirnicu se šalje dominanti bit. Isto vrijedi za CRX samo što je CRX primanje sa sabirnice. Napaja se direktno s ESP32 razvojne pločice.



Slika 6.7. SN65HVD230 primopredajnik

CANL i CANH su spojeni na RJ45 ženski konektor i na strani ESP32_1 i ESP32_2. Konektori su povezani s mrežnim kabelom.

6.5. Opis koda za ESP32_1

Za programiranje ESP32 mikrokontrolera korišteno je Arduino IDE programsko okruženje.

Za CAN_BUS komunikaciju korištena je biblioteka CAN.h koja sadrži sve potrebne funkcije za uspravljanje CAN_BUS sabirnicom.

```
#include <CAN.h>
#define TX_GPIO_NUM   21  // Spojit na CTX
#define RX_GPIO_NUM   22  // Spojit na CRX
#define pot 4             // potenciometar
#define sm 5              // prekidač smjera
#define on_off 16         // prekidač ON/OFF


int data = 0;
int identifikator = 0x01;


int k = 0;
int last_data = 0;

int smjer = 1;
int last_smjer = 0;
int ON = 0;
int last_ON = 0;
```

GPIO pinovi ESP32 mikrokontrolera su definirani s #define, a pomoćne globalne varijable su definirane prema vrsti podataka.

U void setup() je funkcija koja se izvodi samo jednom pri pokretanju mikrokontrolera i njena funkcija je definirati sve GPIO pinove kao ulaze, izlaze, interapte itd. U void setup funkciji se također inicijaliziraju komunikacijski protokoli koji se koriste. U ovom radu se koriste UART komunikacijski protokol kao komunikacija između računala i mikrokontrolera. Primarna funkcija je dijagnosticiranje programskog koda. Također inicijalizira se CAN_BUS komunikacija te se ispisiva povratna informacija o uspješnosti njene inicijalizacije na serial monitoru Arduino IDE programskog okruženja.

```
void setup() {
  Serial.begin (115200);  //inicijalizacija UART komunikacije

  pinMode(pot,INPUT);               //postavljanje GPIO_4 kao ulaz
  pinMode(sm,INPUT_PULLUP);         //postavljanje GPIO_5 kao ulaz
  pinMode(on_off,INPUT_PULLUP);     //postavljanje GPIO_16 kao ulaz
  while (!Serial);
  delay (1000);
  Serial.println ("CAN Receiver/Receiver");
  CAN.setPins (RX_GPIO_NUM, TX_GPIO_NUM);          // Postavljanje pinova

  if (!CAN.begin (1000E3)) {                      // Postavljanje brzine
                                                     prijenosa na 1000kbps
    Serial.println ("Starting CAN failed!");      // Greška u inicijalizaciji
    while (1);
  }
  else {
    Serial.println ("CAN Initialized");          //Uspješno inicijaliziran
                                                    CAN_BUS
  }
}
```

Funkcijom pinMode definira se režim rada GPIO pina. Za potenciometar se definira kao ulaz dok se za prekidače postavlja kao ulaz s priteznim otpornik koji ESP32 ima ugrađene na 32 GPIO pina.

Pozivanjem CAN.setPins postavljaju se definirani pinovi koji se koriste u CAN_BUS komunikaciji, a pozivanjem CAN.begin inicijalizira se komunikacija. Ulazni parametar CAN.begin funkcije je brzina prijenosa koja je u ovom radu postavljena na 1000kbps.

Void loop () funckija se izvodi svaki put iznova i u njoj se nalazi upravljački dio programa. Prvi korak je čitanje analogne vrijednosti potenciometra, čitanje se izvodi 5 puta uz pomoć for petlje i u svakoj iteraciji for petlje vrijednost potenciometra se pridodaje samoj sebi u konačnici se dobije pet zbrojenih vrijednosti potenciometra koje treba podijeliti s brojem iteracija da bi se dobila prava vrijednost pozicije potenciometra. Ovaj korak nije nužan za rad ali povećava preciznost i smanjiva oscilacije vrijednosti potenciometra. Nakon čitanja vrijednosti potenciometra ona se skalira linearno na vrijednosti od 0-100. Varijabla „k“ se resetira na 0 kako bi se u sljedećem čitanju stanja potenciometra krenulo od nule.

```
void loop() {

    for (int i = 0; i<5; i++){                  // 5 čitanja vrijednosti
                                                   potenciometra
      k = k + analogRead(pot);
    }
    k = k/5;

    data = map(k,0,4095,0,100);      // skaliranje vrijednosti potenciometra
    k = 0;
```

Ako je vrijednost podataka koji su u ovom slučaju brzina motora različit od prošlog slanja istih podataka, pozvat će se funkcija camSender i podaci će se poslati na CAN_BUS sabirnicu. Također se u varijablu identifikator upisiva vrijednost identifikatora s kojom se želi poslati poruka.

```
   if(data != last_data){           // slanje brzine motora ako je došlo do
                                       promjene vrijednosti potenciometra
      last_data = data;
      identifikator = 0x13;         //postavljanje indetifikatora za brzinu
                                       motora
      canSender();                  //slanje podataka o brzini vrtnje motora
    }
```

Ako su podaci o smjeru vrtnje motora različiti od prošlih poslati će se na sabirnicu.

```
if (smjer != last_smjer){       // slanje smjera motora ako je došlo do
                                   promjene vrijednosti
      last_smjer = smjer;
      identifikator = 0x14;       //postavljanje indetifikatora za smjer
                                     poruku
      data = smjer;
      canSender();                //slanje podataka o smjeru vrtnje motora

    }
```

Ako su podaci o pokretanju motora različiti od prošlih poslat će se na sabirnicu. Ovaj korak se koristi iz razloga što bi bilo nepotrebno zatrpavati CAN_BUS sabirnicu istim podacima.

```
  if (ON != last_ON){            // slanje ON/OFF motora ako je došlo do
                                    promjene vrijednosti
      last_ON = ON;
      identifikator = 0x11;        //postavljanje indetifikatora za ON/OFF
                                     poruku
      data = ON;
      canSender();                 //slanje podataka ON/OFF

    }
```

Stanja GPIO_4 i GPIO_16 se čitaju i upisuju u varijable. Ako je stanje na GPIO_4 pinu logičko stanje 1 tada je smjer vrtnje motora 1, a ako je logičko stanje 0 tada je smjer vrtnje također 0. Isto vrijedi za GPIO_16 samo što se upisiva u varijablu za pokretanje i gašenje motora.

```
if (digitalRead(sm)==HIGH){          // čitanje stanja prekidača za smjer
                                        vrtnje
      smjer = 1;
    }
    else{
      smjer = 0;
    }

    if (digitalRead(on_off)==HIGH){      // čitanje stanja prekidača za ON/OFF
      ON = 1;
    }
    else{
      ON = 0;
    }

}
```

```
void canSender() {
  // slanje paketa: id je 11bita, a paket može sadržavati do 8 byta podataka
  Serial.print ("Slanje paketa ... ");    //ispisivanje poruke na Serial
                                            monitor
  CAN.beginPacket (identifikator);        //postavlja se ID i čisti se buffer
  // CAN.beginExtendedPacket(0xabcdef);   //uključivanje ove funkcije koristi
                                           se prošireni standard CAN protokola
  CAN.write (data);                       //podaci se upisuju u buffer ali se
                                            ne šalju sve dok se ne pozove
                                            endPacket()

  Serial.println(data);
  CAN.endPacket();                           //slanje podataka
  CAN.beginPacket (identifikator, 3, true);    //postavljanje indetifikator za
                                                 poruku upita
  CAN.endPacket();
  Serial.println ("done");
  delay (1000);
}
```

Funkcija camSender() priprema paket za slanje na CAN_BUS sabirnicu. CAN.beginPacket označava s kojim će se identifikatorom poruka poslati, a CAN.write podatke upisiva u buffer. Prilikom pozivanja CAN.endPacket() podaci se iz buffera šalju na CAN_BUS sabirnicu.

6.6. Opis koda za ESP32_2

Za programiranje je korišteno Arduino Ide programsko sučelje. Za CAN_BUS komunikaciju je korištena CAN.h biblioteka ista kao za ESP32_1.

```
#include <CAN.h>

#define TX_GPIO_NUM   22  // Connects to CTX
#define RX_GPIO_NUM   23  // Connects to CRX

#define AH 4              // izlazni GPIO za trofazni h-most ...
#define AL 2
#define BH 17
#define BL 16
#define CH 19
#define CL 21             // ...izlazni GPIO za trofazni h-most

#define Senzor_1 12       // Ulazni GPIO za hall senzor
#define Senzor_2 13       // Ulazni GPIO za hall senzor
#define Senzor_3 14       // Ulazni GPIO za hall senzor
```

U ovom dijelu koda se definiraju GPIO pinovi.

```
int Hall_A = 0;
int Hall_B = 0;
int Hall_C = 0;
int HAll_A_last  = 0;
int HAll_B_last  = 0;
int HAll_C_last  = 0;

int Delay = 7000;
int faza = 1;


int R_identifikator = 0x01;
int data = 0;
int smjer = 1;
int ON = 0;

unsigned long trenutnoVrijeme = 0;
unsigned long prosloVrijeme = 0;
```

Definiranje globalnih varijabli prema vrti podataka.

```
void IRAM_ATTR isr1() {                  // interapt funkcija za HALL senzor_1
    Hall_A = digitalRead(Senzor_1);      // čitanje stanja hall senzora...
    Hall_B = digitalRead(Senzor_2);
    Hall_C = digitalRead(Senzor_3);      //...čitanje stanja hall senzora

    if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 1){   // podešavanje faze
                                                         komutacije s izlazima
                                                         3 hall senzora...
      faza = 3;
      }
    if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 0){
      faza = 2;
      }
    if (Hall_A == 1 && Hall_B == 1 &&  Hall_C == 0){
      faza = 1;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 0){
      faza = 6;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 1){
      faza = 5;
      }
    if (Hall_A == 0 && Hall_B == 0 &&  Hall_C == 1){   //...podešavanje
                                                       faze komutacije s
                                                       izlazima 3 hall senzora

      faza = 4;
      }
}
```

Kreira se ISR interapt funkcija koja se poziva pri promjeni stanja hall senora_1 iz logičkog stanja 1 u logičko stanje 0 i obratno. Kada se ova funkcija pozove provjeravaju se stanja svih senzora i njihova vrijednost određiva fazu komutacije. Ako je npr. stanje senzora 1 u logičkom stanju 1, stanje senzora 2 u logičkom stanju 0 i stanje senzora 3 u logičkom stanju 0 tada je faza komutacije 2 što bi značilo da treba uključiti mosfet Q2 i Q3, a ostale ugasiti.

Dodavanjem atributa IRAM_ATTR ubrzava se postupak izračuna jer se izračun izvršava s upotrebom RAM memorije koja je znatno brža of flash memorije.

```
void IRAM_ATTR isr2() {           // interapt funkcija za HALL senzor_2
    Hall_A = digitalRead(Senzor_1);      // čitanje stanja hall senzora...
    Hall_B = digitalRead(Senzor_2);
    Hall_C = digitalRead(Senzor_3);     //...čitanje stanja hall senzora

    if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 1){ // podešavanje faze
                                                        komutacije s izlazima
                                                         3 hall senzora...
      faza = 3;
      }
    if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 0){
      faza = 2;
      }
    if (Hall_A == 1 && Hall_B == 1 &&  Hall_C == 0){
      faza = 1;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 0){
      faza = 6;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 1){
      faza = 5;
      }
    if (Hall_A == 0 && Hall_B == 0 &&  Hall_C == 1){ //...podešavanje faze
                                                        komutacije s izlazima
                                                        3 hall senzora

      faza = 4;
      }
}
```

Kreiranje ISR interapt funkcije za senzor 2. Ima istu funkciju kao kod senzora 1.

```
void IRAM_ATTR isr3() {           // interapt funkcija za HALL senzor_3
    Hall_A = digitalRead(Senzor_1);    // čitanje stanja hall senzora...
    Hall_B = digitalRead(Senzor_2);
    Hall_C = digitalRead(Senzor_3);    //...čitanje stanja hall senzora
    if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 1){  // podešavanje faze
                                                        komutacije s izlazima
                                                         3 hall senzora...
      faza = 3;
      }
    if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 0){
      faza = 2;
      }
    if (Hall_A == 1 && Hall_B == 1 &&  Hall_C == 0){
      faza = 1;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 0){
      faza = 6;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 1){
      faza = 5;
      }
    if (Hall_A == 0 && Hall_B == 0 &&  Hall_C == 1){  //...podešavanje faze
                                                        komutacije s izlazima
                                                        3 hall senzora

      faza = 4;
      }
}
```

Kreiranje ISR interapt funkcije za senzor 3 i ima istu funkciju kao kod senzora 1 i senzora 2.

U void setup () funkciji inicijalizira se UART komunikacija, postavlja se CAN_BUS brzina prijenosa na 1000kbps i definiraju se GPIO pinovi kao ulazni izlazni ili interapti.

```
void setup() {
  Serial.begin (115200);           // inicijalizacija UART komunikacije
  while (!Serial);
  delay (1000);
  Serial.println ("CAN Receiver/Receiver");
  CAN.setPins (RX_GPIO_NUM, TX_GPIO_NUM);  // postavljanje pinova za CAN_BUS
  if (!CAN.begin (1000E3)) {           //Postavljanje brzine prijenosa na
                                        1000kbps
    Serial.println ("Starting CAN failed!");    // greška u inicijalizaciji
    while (1);
  }
  else {
    Serial.println ("CAN Initialized"); //uspješna inicijalizacija CAN
                                        protokola
  }

   pinMode(AH,OUTPUT);                 //postavljanje GPIO pina kao izlaz...
   pinMode(AL,OUTPUT);
   pinMode(BH,OUTPUT);
   pinMode(BL,OUTPUT);
   pinMode(CH,OUTPUT);
   pinMode(CL,OUTPUT);          //...postavljanje GPIO pina kao izlaz

   pinMode(Senzor_1,INPUT);     //postavljanje GPIO pina senzora kao ulaz ...
   pinMode(Senzor_2,INPUT);
   pinMode(Senzor_3,INPUT);     //...postavljanje GPIO pina senzora kao ulaz


   attachInterrupt(Senzor_1, isr1, CHANGE);    //postavljanje GPIO pina
                                                senzora kao "izmjenični"
                                                interapt
   attachInterrupt(Senzor_2, isr2, CHANGE);    //postavljanje GPIO pina
                                                senzora kao "izmjenični"
                                                interapt
   attachInterrupt(Senzor_3, isr3, CHANGE);    //postavljanje GPIO pina
                                                senzora kao "izmjenični"
                                                interapt


 prosloVrijeme = micros();
}
```

```
void loop() {

  canReceiver();          //pozivanje funkcije za provjeru dostupnosti
                            podataka na CAN_BUS sabirnici

  if (R_identifikator == 0x13){          // ako je primljena poruka s
                                            indetifikatorom 0x13 radi se o
                                             podacima brzine vrtnje motora
    Delay = map(data,0,100,7000,1);     //mapiranje podataka na vjerdnosti
                                          7000-1
  }
  if (R_identifikator == 0x14){        // ako je primljena poruka s
                                          indetifikatorom 0x14 radi se o
                                          podacima smjera vrtnje motora
    smjer = data;
  }
  if (R_identifikator == 0x11){       // ako je primljena poruka s
                                          indetifikatorom 0x11 radi se o
                                           podacima za paljenje i gašenje
    ON = data;
  }
```

Pri početku void loop funkcije() poziva se canReciver() funkcija s kojom se provjerava ima li novih podataka na CAN_BUS sabirnici, ako ima spremaju se u data varijablu. IF petlje služe za usporedbu identifikatora poruke, ako je identifikator 0x13 tada se radi o podacima brzine vrtnje motora. Identifikator 0x14 je za smjer okretanja motora, a 0x11 je za paljenje i gašenje motora.

```
trenutnoVrijeme = micros();             //pokretanje brojanja vremena
if (ON == 1 ){ //ako je ON = 1 motor se može pokrenuti, a ako je ON = 0 motor
                se ne može pokrenut
  if(trenutnoVrijeme - prosloVrijeme >= Delay){
        prosloVrijeme += Delay;         //dodavanje delay-a na vrijednost
                                          prošlog vremena
```

U ovom dijelu koda počinje brojanje vremena što je potrebno za komutaciju motora. Prva if petlja služi za omogućavanje ili onemogućavanje druge if petlja. Ovaj dio je potreban kako bi se motor zaustavia. Ako je ON varijabla u logičkom stanju 1 druga if petlja je onemogućena, a ako je ON = 0 tada je onemogućena druga if petlja i svi MOSFET-ti se postavljaju u stanje isključeno. Druga if petlja se izvršava nakon što prođe vrijeme definirano s varijablom Delay. Vrijeme je izraženo u mikrosekundama.

```
switch(faza){                    // upravljanje fazama komutacije
          case 1:
          if (smjer == 1){      // ako je smjer 1 poziva se funckija step1()
            step1();
            }
            else{              //ako je smjer 0 poziva se step4()
            step4();
              }
          break;

          case 2:
          if (smjer == 1){
            step2();
            }
            else{
            step5();
              }
          break;

          case 3:
          if (smjer == 1){
            step3();
            }
            else{
            step6();
              }
          break;
          case 4:
          if (smjer == 1){
            step4();
            }
            else{
            step1();
              }
          break;
          case 5:
          if (smjer == 1){
            step5();
            }
            else{
            step2();
              }
```

```
          case 6:
          if (smjer == 1){
            step6();
            }
            else{
            step3();
              }

          break;

        }

  }
}
else{                                   //ako je ON 0 poziva se step7()
  step7();
  }
}
```

Switch-case petlja služi za povezivanje faza komutacije s uključivanjem odgovarajućih MOSFET-a. Kada se definira faza komutacije preko hall senzora tada se u switch-case petlji poziva funkcija koja uključiva odgovarajuće mosfete. If petlja ugniježđena u fazi komutacije služi za promjenu smjera vrtnje. Ako je varijabla smjer u logičkom stanju 1 pozivat će se koraci redom 1,2,3,4,5,6, a ako je smjer obrnut tada su koraci 4,5,6,1,2,3. Pozivanjem funkcije step7() svi mosfet-ti se gase i motor prestaje s okretanjem.

Tablica 6-1. sekvenca paljenja mosfeta za smjer = 1

Senzor1	Senzor2	Senzor3	AH(Q1)	AL(Q2)	BH(Q3)	BL(Q4)	CH(Q5)	CL(Q6)
1	0	1	0	0	1	0	0	1
1	0	0	0	1	1	1	0	0
1	1	0	0	1	0	0	1	0
0	1	0	0	0	0	1	1	0
0	1	1	1	0	0	1	0	0
0	0	1	1	0	0	0	0	1

U tablici 6.1 može se vidjeti sekvenca paljenja pojedinih mosfeta u odnosu na izlazna stanja hall senzora. Ako se želi postići promjena smjera vrte motora mora se prominiti sekvenca paljenja.

Promjena smjera se izvodi na način da se za istu kombinaciju hall senzora magnetizira namoti motora u drugom smjeru nego što je prikazano u tablici 6.1. Okretanje smjera struje kroz namot se provodi tako da se obrnu logička stanja mosfeta za istu kombinaciju hall senzora. Za istu kombinaciju hall senzora mosfeti koji u jednom smjeru nisu bili upaljeni ni u drugom smjeru neće biti.

Tablica 6-2 sekvenca paljenja mosfeta za smjer = 0

Senzor 1	Senzor 2	Senzor 3	AH(Q1)	AL(Q2)	BH(Q3)	BL(Q4)	CH(Q5)	CL(Q6)
1	0	1	0	0	0	1	1	0
1	0	0	1	0	0	1	0	0
1	1	0	1	0	0	0	0	1
0	1	0	0	0	1	0	0	1
0	1	1	0	1	1	0	0	0
0	0	1	0	1	0	0	1	0

```
void step1(){                    //postavljanje logičkih stanja na GPIO pinove
    digitalWrite(AH,0); //0                    // MOSFET Q1 "isključen"
    digitalWrite(AL,1); //1                    // MOSFET Q2 "uključen"
    digitalWrite(BH,0); //0                    // MOSFET Q3 "isključen"
    digitalWrite(BL,0); //0                    // MOSFET Q4 "isključen"
    digitalWrite(CH,1); //1                    // MOSFET Q5 "uključen"
    digitalWrite(CL,0); //0                    // MOSFET Q6 "isključen"

  }

void step2(){
    digitalWrite(AH,0); //0
    digitalWrite(AL,1); //1
    digitalWrite(BH,1); //1
    digitalWrite(BL,0); //0
    digitalWrite(CH,0); //0
    digitalWrite(CL,0); //0

  }
```

```
void step3(){
    digitalWrite(AH,0); //0
    digitalWrite(AL,0); //0
    digitalWrite(BH,1); //1
    digitalWrite(BL,0); //0
    digitalWrite(CH,0); //0
    digitalWrite(CL,1); //1
  }

void step4(){
    digitalWrite(AH,1); //1
    digitalWrite(AL,0); //0
    digitalWrite(BH,0); //0
    digitalWrite(BL,0); //0
    digitalWrite(CH,0); //0
    digitalWrite(CL,1); //1
  }

void step5(){
    digitalWrite(AH,1); //1
    digitalWrite(AL,0); //0
    digitalWrite(BH,0); //0
    digitalWrite(BL,1); //1
    digitalWrite(CH,0); //0
    digitalWrite(CL,0); //0
  }
void step6(){                   //postavljanje logičkih stanja na GPIO pinove
    digitalWrite(AH,0); //0           // MOSFET Q1 "isključen"
    digitalWrite(AL,0); //0           // MOSFET Q2 "isključen"
    digitalWrite(BH,0); //0           // MOSFET Q3 "isključen"
    digitalWrite(BL,1); //1           // MOSFET Q4 "uključen"
    digitalWrite(CH,1); //1           // MOSFET Q5 "uključen"
    digitalWrite(CL,0); //0           // MOSFET Q6 "isključen"
  }
void step7(){                        //svi MOSFETi su "isključeni"
    digitalWrite(AH,0); //0
    digitalWrite(AL,0); //1
    digitalWrite(BH,0); //0
    digitalWrite(BL,0); //0
    digitalWrite(CH,0); //1
    digitalWrite(CL,0); //0

  }
```

Funkcije step 1 do step 7 šalju logička stanja na GPIO pinove koji upravljaju mosfetima.

```
void canReceiver() {      //funkcija za primanje podataka s CAN_BUS sabirnice
  // razčlanjivanje paketa
  int packetSize = CAN.parsePacket();
  if (packetSize) {
    // paket primljen
    Serial.print ("primljen ");
    if (CAN.packetExtended()) {         //provjera vrste indetifikatora
      Serial.print ("extended ");
    }
    if (CAN.packetRtr()) {              //provjera RTR bita tj jel se radi o
                                          paketu upita ili paketa s podacima
      // Remote transmission request, paket nema podataka
      Serial.print ("RTR ");
    }
    Serial.print ("paket sa id 0x");
    Serial.print (CAN.packetId(), HEX);    //ispisivanje indetifikator na
                                             Serial monitor
    R_identifikator = CAN.packetId();     //zapisivanje indetifikatora u
                                            varijablu za kasniju usporedbu
    if (CAN.packetRtr()) {
      Serial.print (" and requested length ");
      Serial.println (CAN.packetDlc());
    } else {                           //Ako se radi o običnom paketu s podacima
      Serial.print (" and length ");
      Serial.println (packetSize);    //ispisiva se veličina podataka na
                                        serial monitoru
      while (CAN.available()) {
        data = CAN.read();         // upisivanje CAN_BUS podataka u varijablu
                                      za kasniju usporedbu i obradu
        Serial.println (data)
      }
      Serial.println();
    }
    Serial.println();
  }
}
```

Funkcija canReceiver služi za prijem podataka s CAN_BUS sabirnice. U prvom dijelu funkcije provjerava se dostupnost paketa, a zatim se provjerava vrsta identifikatora. Ako se radi o proširenom identifikatoru ispisivat će se poruka na serijal monitoru. Vrijednosnost identifikatora se sprema u globalnu varijablu za kasnije korištenje. Nakon provjere identifikatora provjerava se jeli se radi o paketu s podacima ili upitu. Ako se radi o paketu s podacima ti se podaci spremaju u globalnu varijablu i ispisivanju se na serial monitoru.

7. ZAKLJUČAK

Upravljanjem BLDC motorom pomoću CAN_BUS protokola omogućilo je upravljanje motorom s veće udaljenosti. Također je učinilo sustav upravljanjem pouzdanijim i otpornijim na vanjske smetnje. Komutacija BLDC motora je ostvarena uz pomoć trofaznog H-mosta koji omogućava izmjenični protok struje kroz namotaje BLDC motora. Trofazni H-most realiziran je s primjenom mosfeta i adekvatnih upravljačkih sklopova za njihovo upravljanje. Povratna veza pozicije rotora je realizira s primjenom 3 hall senzora čiji izlazi određuju fazu komutacije. Mogućnost pristupa ESP32 mikrokontrolera na CAN_BUS sabirnicu je ostvareno korištenjem vanjskog primopredajnika koji CAN_BUS signal pretvara u LVTTL logiku.

LITEARATURA

[1] Brushless DC Motor Fundamentals Application Note Prepared by Jian Zhao/Yangwei Yu July 2011,

[2] AN1946 APPLICATION NOTE SENSORLESS BLDC MOTOR CONTROL AND BEMF SAMPLING METHODS WITH ST7MC

[3] https://simple-circuit.com/pic18f4550-esc-sensorless-bldc-motor-controller/

[4] Design of FOC Brushless DC (BLDC) Motor Controller 1Rohita Lokhande, 2Prof. Priyanka Dukre

[5] https://www.pmdcorp.com/resources/type/articles/get/field-oriented-control-foc-a-deep-dive-article

[6] https://www.ato.com/differences-between-outrunner-and-inrunner-brushless-motors

[7] Design of 6S8P axial flux permanent magnet brushless DC motor with double-sided rotor Satryo Budi Utomo1 , Januar Fery Irawan2 , Widyono Hadi1 , BA Sastiko1

[8] https://www.integrasources.com/blog/bldc-motor-controller-design-principles/

[9] Design of axial-flux permanent-magnet low-speed machines and performance comparison between radial-flux and axial-flux machines

[10] Sveučilište u Splitu, Industrijske Računalne Mreže,autor: Silvano Jenčić

[11] Introduction to the Controller Area Network (CAN) SLOA101B–August 2002–Revised May 2016

[12] CAN Bus Explained - A Simple Intro [2023] – CSS Electronics

[13] Home Appliances Management System using Controller Area Network (CAN)

[14] https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[15] https://dronebotworkshop.com/esp32-intro/

[16] https://mischianti.org/2021/05/26/esp32-wroom-32-high-resolution-pinout-and-specs/

[17] https://limitedresults.com/2019/08/pwn-the-esp32-crypto-core/

[18] https://lastminuteengineers.com/handling-esp32-gpio-interrupts-tutorial/#google_vignette

[19] https://www.theengineeringprojects.com/2021/12/esp32-interrupts.html

[20] https://makeabilitylab.github.io/physcomp/esp32/esp32.html#programming-environment

[21] https://www.espboards.dev/blog/esp-idf-vs-arduino-core/

[22] Texas Instruments: SN65HVD23x 3.3-V CAN Bus Transceivers datasheet

[23] Vishay Siliconix IRLR44 datasheet

[24] Vishay Siliconix IRF9510 datasheet

POPIS SLIKA

POPIS TABLICA

PRILOZI

Programski kod za ESP32_1

```
#include <CAN.h>
#define TX_GPIO_NUM   21  // Spojit na CTX
#define RX_GPIO_NUM   22  // Spojit na CRX
#define pot 4             // potenciometar
#define sm 5              // prekidač smjera
#define on_off 16        // prekidač ON/OFF
int data = 0;
int identifikator = 0x01;
int k = 0;
int last_data = 0;
int smjer = 1;
int last_smjer = 0;
int ON = 0;
int last_ON = 0;
int y = 0;
int last_y = 1;

void setup() {

  Serial.begin (115200);  //inicijalizacija UART komunikacije
    pinMode(pot,INPUT);               //postavljanje GPIO_4 kao ulaz
  pinMode(sm,INPUT_PULLUP);         //postavljanje GPIO_5 kao ulaz
  pinMode(on_off,INPUT_PULLUP);     //postavljanje GPIO_16 kao ulaz
  while (!Serial);
  delay (1000);
  Serial.println ("CAN Receiver/Receiver");
  CAN.setPins (RX_GPIO_NUM, TX_GPIO_NUM);          // Postavljanje pinova
  // start the CAN bus at 500 kbps
  if (!CAN.begin (1000E3)) {                      // Postavljanje brzine
prijenosa
    Serial.println ("Starting CAN failed!");      // Greška u inicijalizaciji
    while (1);
  }
  else {
    Serial.println ("CAN Initialized");          //Uspješno inicijaliziran
CAN_BUS
  }
}
```

```
void loop() {
    for (int i = 0; i<5; i++){                // 5 čitanja vrijednosti
potenciometra
      k = k + analogRead(pot);
    }
    k = k/5;
    data = map(k,0,4095,0,100);      // skaliranje vrijednosti potenciometra
    k = 0;
    if(data != last_data){           // slanje brzine motora ako je došlo do
promjene vrijednosti potenciometra
      last_data = data;
      identifikator = 0x13;         //postavljanje indetifikatora za brzinu
motora
      canSender();                  //slanje podataka o brzini vrtnje motora
    }
    if (smjer != last_smjer){       // slanje smjera motora ako je došlo do
promjene vrijednosti
      last_smjer = smjer;
      identifikator = 0x14;         //postavljanje indetifikatora za smjer
poruku
      data = smjer;
      canSender();                  //slanje podataka o smjeru vrtnje motora
    }
    if (ON != last_ON){            // slanje ON/OFF motora ako je došlo do
promjene vrijednosti
      last_ON = ON;
      identifikator = 0x11;        //postavljanje indetifikatora za ON/OFF
poruku
      data = ON;
      canSender();                 //slanje podataka ON/OFF
    }
    if (digitalRead(sm)==HIGH){          // čitanje stanja prekidača za smjer
vrtnje
      smjer = 1;
    }
    else{
      smjer = 0;
    }
    if (digitalRead(on_off)==HIGH){      // čitanje stanja prekidača za ON/OFF
      ON = 1;
    }
    else{
      ON = 0;
    }

}
```

```
void canSender() {
  // slanje paketa: id je 11bita, a paket može sadržavati do 8 byta podataka
  Serial.print ("Slanje paketa ... ");    //ispisivanje poruke na Serial
monitor
  CAN.beginPacket (identifikator);        //postavlja se ID i čisti se buffer
  // CAN.beginExtendedPacket(0xabcdef);   //uključivanje ove funkcije koristi
se proširenI standard CAN protokola
  CAN.write (data);                       //podaci se upisuju u buffer ali se
ne šalju sve dok se ne pozove endPacket()
  Serial.println(data);
  CAN.endPacket();                              //slanje podataka
  CAN.beginPacket (identifikator, 3, true);    //postavljanje indetifikator za
poruku upita
  CAN.endPacket();
  Serial.println ("done");
  delay (1000);
}
```

Programski kod za ESP32_2

```
#include <CAN.h>

#define TX_GPIO_NUM   22  // Connects to CTX
#define RX_GPIO_NUM   23  // Connects to CRX

#define AH 4              // izlazni GPIO za trofazni h-most ...
#define AL 2
#define BH 17
#define BL 16
#define CH 19
#define CL 21             // ...izlazni GPIO za trofazni h-most

#define Senzor_1 12       // Ulazni GPIO za hall senzor
#define Senzor_2 13       // Ulazni GPIO za hall senzor
#define Senzor_3 14       // Ulazni GPIO za hall senzor


int Hall_A = 0;
int Hall_B = 0;
int Hall_C = 0;
int HAll_A_last  = 0;
int HAll_B_last  = 0;
int HAll_C_last  = 0;

int Delay = 7000;
int faza = 1;


int R_identifikator = 0x01;
int data = 0;
int smjer = 1;
int ON = 0;

unsigned long trenutnoVrijeme = 0;
unsigned long prosloVrijeme = 0;
```

```
void IRAM_ATTR isr1() {                                                 //
interapt funkcija za HALL senzor_1
    Hall_A = digitalRead(Senzor_1);                                     //
čitanje stanja hall senzora...
    Hall_B = digitalRead(Senzor_2);
    Hall_C =
digitalRead(Senzor_3);                                  //...čitanje stanja
hall senzora

    if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 1){                //
podešavanje faze komutacije s izlazima 3 hall senzora...
      faza = 3;
      }
    if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 0){
      faza = 2;
      }
    if (Hall_A == 1 && Hall_B == 1 &&  Hall_C == 0){
      faza = 1;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 0){
      faza = 6;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 1){
      faza = 5;
      }
    if (Hall_A == 0 && Hall_B == 0 &&  Hall_C ==
1){              //...podešavanje faze komutacije s izlazima 3 hall senzora
      faza = 4;
      }
}
void IRAM_ATTR isr2() {                                                 // interapt
funkcija za HALL senzor_2
    Hall_A = digitalRead(Senzor_1);                                     // čitanje
stanja hall senzora...
    Hall_B = digitalRead(Senzor_2);
    Hall_C = digitalRead(Senzor_3);                                     //...čitanje
stanja hall senzora
```

```
if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 1){              // podešavanje
faze komutacije s izlazima 3 hall senzora...
      faza = 3;
      }
    if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 0){
      faza = 2;
      }
    if (Hall_A == 1 && Hall_B == 1 &&  Hall_C == 0){
      faza = 1;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 0){
      faza = 6;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 1){
      faza = 5;
      }
    if (Hall_A == 0 && Hall_B == 0 &&  Hall_C == 1){          //...podešavanje
faze komutacije s izlazima 3 hall senzora
      faza = 4;
      }
}
void IRAM_ATTR isr3() {                                          // interapt
funkcija za HALL senzor_3
    Hall_A = digitalRead(Senzor_1);                              // čitanje
stanja hall senzora...
    Hall_B = digitalRead(Senzor_2);
    Hall_C = digitalRead(Senzor_3);                              //...čitanje
stanja hall senzora
    if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 1){             //
podešavanje faze komutacije s izlazima 3 hall senzora...
      faza = 3;
      }
    if (Hall_A == 1 && Hall_B == 0 &&  Hall_C == 0){
      faza = 2;
      }
    if (Hall_A == 1 && Hall_B == 1 &&  Hall_C == 0){
      faza = 1;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 0){
      faza = 6;
      }
    if (Hall_A == 0 && Hall_B == 1 &&  Hall_C == 1){
      faza = 5;
      }
    if (Hall_A == 0 && Hall_B == 0 &&  Hall_C == 1){          //...podešavanje
faze komutacije s izlazima 3 hall senzora
      faza = 4;
      }}
```

```
void setup() {
  Serial.begin (115200);                                          //
inicijalizacija UART komunikacije
  while (!Serial);
  delay (1000);
  Serial.println ("CAN Receiver/Receiver");
  CAN.setPins (RX_GPIO_NUM, TX_GPIO_NUM);                        // postavljanje
pinova za CAN_BUS
  if (!CAN.begin (1000E3)) {                                      //Postavljanje
brzine prijenosa na 1000kbps
    Serial.println ("Starting CAN failed!");                     // greška u
inicijalizaciji
    while (1);
  }
  else {
    Serial.println ("CAN Initialized");                          //uspješna
inicijalizacija CAN protokola
  }

   pinMode(AH,OUTPUT);                  //postavljanje GPIO pina kao izlaz...
   pinMode(AL,OUTPUT);
   pinMode(BH,OUTPUT);
   pinMode(BL,OUTPUT);
   pinMode(CH,OUTPUT);
   pinMode(CL,OUTPUT);                 //...postavljanje GPIO pina kao izlaz

   pinMode(Senzor_1,INPUT);            //postavljanje GPIO pina senzora kao
ulaz ...
   pinMode(Senzor_2,INPUT);
   pinMode(Senzor_3,INPUT);           //...postavljanje GPIO pina senzora kao
ulaz


   attachInterrupt(Senzor_1, isr1, CHANGE);    //postavljanje GPIO pina
senzora kao "izmjenični" interapt
   attachInterrupt(Senzor_2, isr2, CHANGE);    //postavljanje GPIO pina
senzora kao "izmjenični" interapt
   attachInterrupt(Senzor_3, isr3, CHANGE);    //postavljanje GPIO pina
senzora kao "izmjenični" interapt


 prosloVrijeme = micros();
}
```

```
void loop() {

  canReceiver();                                    //pozivanje funkcije za
provjeru dostupnosti podataka na CAN_BUS sabirnici

  if (R_identifikator == 0x13){                    // ako je primljena poruka
s indetifikatorom 0x13 radi se o podacima brzine vrtnje motora
    Delay = map(data,0,100,7000,1);                //mapiranje podataka na
vjerdnosti 7000-1
  }
  if (R_identifikator == 0x14){                   // ako je primljena poruka s
indetifikatorom 0x14 radi se o podacima smjera vrtnje motora
    smjer = data;
  }
  if (R_identifikator == 0x11){                  //// ako je primljena poruka
s indetifikatorom 0x11 radi se o podacima za paljenje i gašenje
    ON = data;
  }



  trenutnoVrijeme = micros();                                          //pokretanje
brojanja vremena
if (ON == 1 ){                                                         //ako je ON
= 1 motor se može pokrenuti, a ako je ON = 0 motor se ne može pokrenut
  if(trenutnoVrijeme - prosloVrijeme >= Delay){
        prosloVrijeme += Delay;                                        //dodavanje
delay-a na vrijednost prošlog vremena
        switch(faza){                                                  //
upravljanje fazama komutacije
          case 1:
          if (smjer == 1){                                             // ako je
smjer 1 poziva se funckija step1()
            step1();
            }
            else{                                                      //ako je
smjer 0 poziva se step4()
            step4();
              }
          break;

          case 2:
          if (smjer == 1){
            step2();
            }
            else{
            step5();
              }
          break;
```

```
case 3:
          if (smjer == 1){
            step3();
            }
            else{
            step6();
              }
          break;

          case 4:
          if (smjer == 1){
            step4();
            }
            else{
            step1();
              }
          break;

          case 5:
          if (smjer == 1){
            step5();
            }
            else{
            step2();
              }

          case 6:
          if (smjer == 1){
            step6();
            }
            else{
            step3();
              }

          break;

        }

  }
}
else{                                          //ako je ON 0 poziva se
step7()
  step7();
  }


}
```

```
void step1(){                                   //postavljanje logičkih
stanja na GPIO pinove
    digitalWrite(AH,0); //0                     // MOSFET Q1 "isključen"
    digitalWrite(AL,1); //1                     // MOSFET Q2 "uključen"
    digitalWrite(BH,0); //0                     // MOSFET Q3 "isključen"
    digitalWrite(BL,0); //0                     // MOSFET Q4 "isključen"
    digitalWrite(CH,1); //1                     // MOSFET Q5 "uključen"
    digitalWrite(CL,0); //0                     // MOSFET Q6 "isključen"

  }

void step2(){
    digitalWrite(AH,0); //0
    digitalWrite(AL,1); //1
    digitalWrite(BH,1); //1
    digitalWrite(BL,0); //0
    digitalWrite(CH,0); //0
    digitalWrite(CL,0); //0

  }

void step3(){
    digitalWrite(AH,0); //0
    digitalWrite(AL,0); //0
    digitalWrite(BH,1); //1
    digitalWrite(BL,0); //0
    digitalWrite(CH,0); //0
    digitalWrite(CL,1); //1

  }

void step4(){
    digitalWrite(AH,1); //1
    digitalWrite(AL,0); //0
    digitalWrite(BH,0); //0
    digitalWrite(BL,0); //0
    digitalWrite(CH,0); //0
    digitalWrite(CL,1); //1

  }

void step5(){
    digitalWrite(AH,1); //1
    digitalWrite(AL,0); //0
    digitalWrite(BH,0); //0
    digitalWrite(BL,1); //1
    digitalWrite(CH,0); //0
    digitalWrite(CL,0); //0
  }
```

```
void step6(){                           //postavljanje logičkih stanja na GPIO
pinove
    digitalWrite(AH,0); //0           // MOSFET Q1 "isključen"
    digitalWrite(AL,0); //0           // MOSFET Q2 "isključen"
    digitalWrite(BH,0); //0           // MOSFET Q3 "isključen"
    digitalWrite(BL,1); //1           // MOSFET Q4 "uključen"
    digitalWrite(CH,1); //1           // MOSFET Q5
"uključen"
    digitalWrite(CL,0); //0           // MOSFET Q6 "isključen"

  }
  void step7(){                         //svi MOSFETi su "isključeni"
    digitalWrite(AH,0); //0
    digitalWrite(AL,0); //1
    digitalWrite(BH,0); //0
    digitalWrite(BL,0); //0
    digitalWrite(CH,0); //1
    digitalWrite(CL,0); //0

  }


void canReceiver() {                                          //funkcija za
primanje podataka s CAN_BUS sabirnice
  // razčlanjivanje paketa
  int packetSize = CAN.parsePacket();
  if (packetSize) {
    // paket primljen
    Serial.print ("primljen ");
    if (CAN.packetExtended()) {                                //provjera
vrste indetifikatora
      Serial.print ("extended ");
    }
    if (CAN.packetRtr()) {                                     //provjera RTR
bita tj jel se radi o paketu upita ili paketa s podacima
      // Remote transmission request, paket nema podataka
      Serial.print ("RTR ");
    }
    Serial.print ("paket sa id 0x");
    Serial.print (CAN.packetId(), HEX);                       //ispisivanje
indetifikator na Serial monitor
```

```
    R_identifikator = CAN.packetId();                              //zapisivanje
indetifikatora u varijablu za kasniju usporedbu
    if (CAN.packetRtr()) {
      Serial.print (" and requested length ");
      Serial.println (CAN.packetDlc());
    } else {                                                        //Ako se radi o
običnom paketu s podacima
      Serial.print (" and length ");
      Serial.println (packetSize);                                 //ispisiva se
veličina podataka na serial monitoru
      while (CAN.available()) {
        data = CAN.read();                                         // upisivanje
CAN_BUS podataka u varijablu za kasniju usporedbu i obradu
        Serial.println (data);



      }
      Serial.println();
    }
    Serial.println();
  }
}
```