Implementacija XCP protokola preko CAN protokola na Aurix platformi

Pečurlić, Dino

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU FAKULTET ELEKTOTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

IMPLEMENTACIJA XCP PROTOKOLA PREKO CAN PROTOKOLA NA AURIX PLATFORMI

Diplomski rad

Dino Pečurlić

Osijek, 2019.

Sadržaj

1. UVOD

Razvoj tehnologije i postavljanje sve većih zahtjeva za autoindustriju dovelo je do sve većeg korištenja elektroničkih uređaja u automobilima. Jedan od najvažnijih razloga povećanja broja elektroničkih upravljačkih jedinica (engl. e*lectronic control unit* - ECU) je taj što današnja vozila imaju znatno više senzora koji pribavljaju velik broj podataka koje je potrebno obraditi, a obradom tih podataka, te upravljanjem ostalim komponentama bavi se ECU. ECU je izraz koji obuhvaća sve uređaje koji kontroliraju određene podsustave u današnjim automobilima. Postoji puno vrsta ECU-a, a njihove funkcije se razlikuju. Današnji automobili mogu sadržavati čak i više od 100 ECU-a, a neke od njihovih zadaća su:

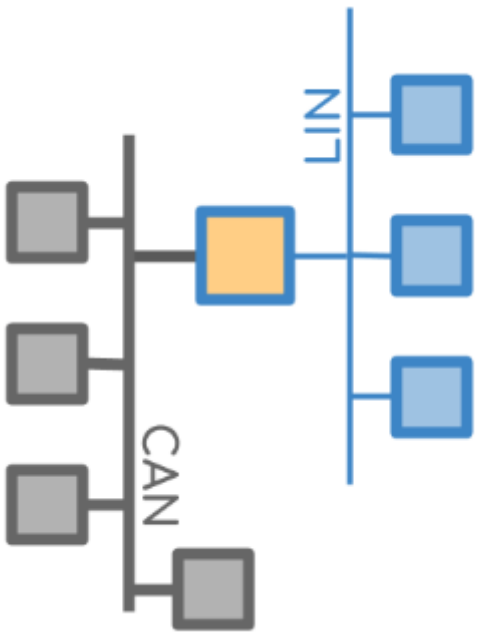
- Upravljanje motorom
- Kontrola kočenja
- Upravljanje prijenosom
- Pomoć pri parkiranju
- Kontrola vuče
- Upravljanje sustavom protiv proklizavanja

Kako postoji velik broj elektroničkih upravljačkih jedinica u automobilu, te je za ispravan rad cjelokupnog sustava potrebna pravovremena izmjena informacija između različitih ECU-a, potrebno je te upravljačke jedinice spojiti na mrežu i tako im omogućiti komuniciranje. U današnjim vozilima svaki ECU je spojen na CAN (engl. c*ontroller area network*) sabirnicu, a komunikacija se odvija putem CAN protokola. Svaki ECU spojen na CAN mrežu čini jedan mrežni čvor koji uključuje mikroupravljač, CAN upravljač i CAN primopredajnik (engl. t*ransceiver*). U svrhu postizanja željenih performansi često je potrebno promatrati i mijenjati određene parametre unutar različitih ECU-a te se zato koristi XCP (engl. u*niversal measurement and calibration protocol*) protokol. XCP protokol omogućuje fleksibilan pristup memoriji ECU-a za čitanje i pisanje i nije ograničen na CAN protokol, kao što je to bio slučaj sa CCP protokolom. (engl. *CAN calibration protocol*).

Cilj diplomskog rada je implementacija XCP protokola, tj. XCP upravljačkog programa, pritom koristeći CAN protokol za komunikaciju. XCP protokol treba omogućiti pristup različitim parametrima ECU-a, a ECU će u ovom slučaju biti AURIX platforma. Za komunikaciju putem CAN sabirnice, koristit će se *Vector* oprema i *Vector Canoe* program.

2. TRENUTNO STANJE INDUSTRIJE SA ASPEKTA UKLANJANJA POGREŠAKA I PROGRAMIRANJA ECU-A

CAN protokol je razvila tvrtka Robert Bosch GmbH 1980. godine, a 1994. godine je CAN protokol postao međunarodni standard. Protokol koji je postao konkurencija CAN protokolu je LIN protokol (engl. *local Interconnect Network*). LIN protokol se koristi kao alternativni izbor kada su prioritet jednostavnost implementacije i smanjenje troškova. Mreže koje se implementiraju korištenjem LIN protokola sastoje se od jednog glavnog računala i do 16 podređenih računala. Za implementaciju LIN protokola potreban je samo jedan kabel maksimalne duljine 40 metara gdje se podatci prenose brzinama od 1 do 20 kbit/s. Iako je LIN protokol postao konkurencija CAN protokolu zbog svoje jednostavnosti i manjih troškova prilikom implementacije, CAN protokol je i dalje najzastupljeniji u automobilskoj industriji. Pomoću LIN protokola može se realizirati komunikacija između mikroupravljača čija je zadaća upravljanje prozorima auta, upravljanje klimatizacijom itd., a svi ti mikroupravljači kojima upravlja glavno računalo LIN sabirnice mogu se spojiti na CAN sabirnicu. Način povezivana LIN sabirnice s CAN sabirnicom prikazan je na slici 2.1.



Sl. 2.1. Povezivanje LIN sabirnice na CAN sabirnicu

Otklanjanje pogrešaka u ovakvim sustavima može biti vrlo kompleksno pa se zato u automobilskoj industriji u svrhu otklanjanja pogrešaka (engl. d*ebugging*) koristi MSO (engl. *mixed signal oscilloscope*). MSO omogućuje inženjeru da istovremeno prati signale u

digitalnoj i analognoj domeni, te omogućuje postavljanje okidača (engl. *triggers*) kako bi se mogle izolirati poruke koje uzrokuju probleme. MSO omogućuje istovremeno praćenje više sabirnica, kao npr. CAN i LIN sabirnice, te omogućuje dekodiranje digitalnog signala i procjenu kvalitete analognog signala.

Kako bi se olakšao i standardizirao način otklanjanja pogrešaka prilikom razvoja programske podrške za ECU koristi se XCP protokol o kojem se više govori u poglavlju 4. XCP protokol je dizajniran kao dvoslojni protokol, tj. sastoji se od jedinstvenog sloja (XCP sloj) i transportnog sloja. Transportni sloj omogućuje podršku za različite transportne medije. Prije korištenja XCP protokola za uklanjanje pogrešaka i programiranje ECU-a korištene su različite metode koje su često bile kompliciranе iz sljedećih razloga:

- Pribavljanje podataka i uklanjanje pogrešaka bilo je odvojeno, tj. koristili su se različiti kablovi s različitim utorima za ECU.
- Stalno prebacivanje između kablova za uklanjanje pogrešaka i pribavljanje podataka predstavlja nepotreban i velik gubitak vremena.

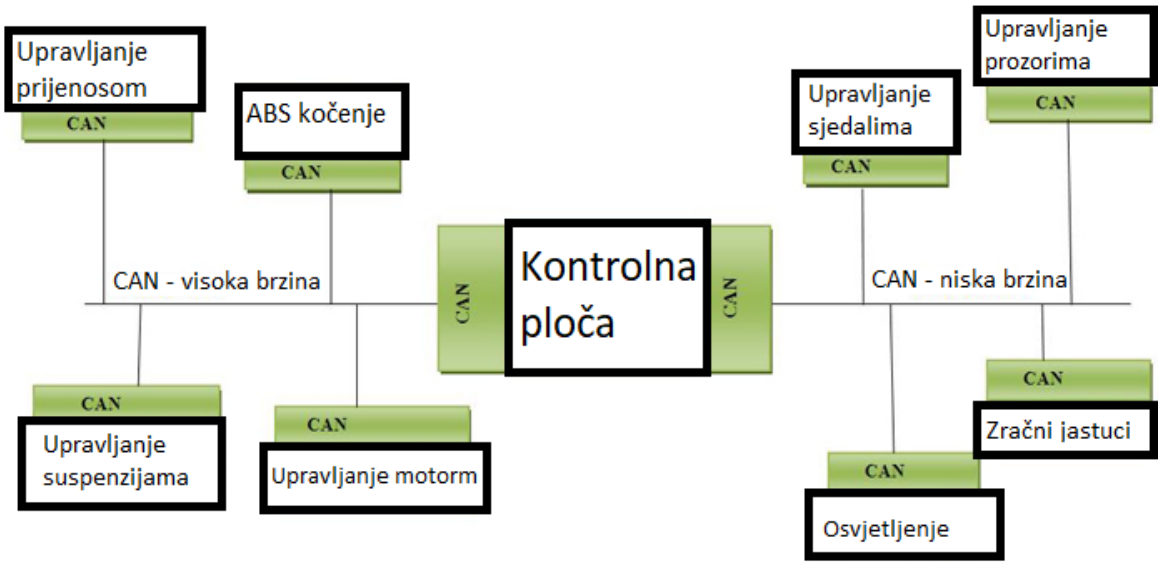
Neke od metoda za uklanjanje pogrešaka i pribavljanje podataka koje su se koristile prije korištenja XCP protokola su: Izmjena ECU signala za uklanjanje pogrešaka ECU-a (engl. *switching of* ECU *debug signals*) [1] i razdijeljeni sustav za pribavljanje podataka (engl. *partitioned MC system*) [1]. Razlog nepraktičnosti korištenja izmjene ECU signala za uklanjanje pogrešaka je preveliko opterećenje sustava, pad performansi sustava i ograničena interoperabilnost modula za uklanjanje pogrešaka i modula za dohvaćanje podataka i kalibraciju. Razlog nepraktičnosti korištenja razdijeljenog sustava za pribavljanje podataka je onemogućavanje istovremenog korištenja modula za uklanjanje pogrešaka.

XCP protokol rješava sve navedene probleme i omogućio je standardizaciju načina programiranja ECU-a i otklanjanja pogrešaka s istog te je to razlog današnje velike zastupljenosti u automobilskoj industriji. XCP protokol omogućuje korištenje različitih uređaja za uklanjanje pogrešaka i programiranje ECU-a. Pomoću XCP protokola moguće je:

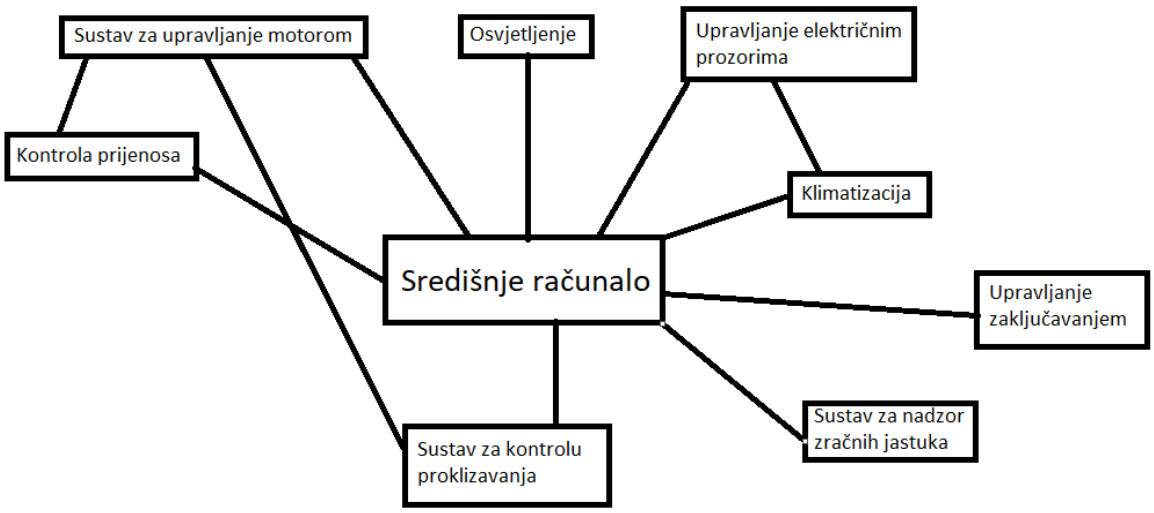
- Ubrzati i poboljšati komunikaciju između uređaja za otklanjanje pogrešaka i ECU-a
- Čitati i pisati u memoriju ECU-a
- Slati ili dohvaćati veliku količinu podataka s najmanjim mogućim opterećenjem sabirnice
- Korištenje različitih komunikacijskih protokola prilikom programiranja i uklanjanja pogrešaka s ECU-a
- Kontroliranje ulaznih i izlaznih podataka ECU-a

3. CAN PROTOKOL

CAN protokol je metoda komunikacije između različitih elektroničkih upravljačkih jedinica koji se brinu o radu pojedinih podsustava automobila kao što su ABS kočenje, upravljanje klimatizacijom i svjetlima itd. CAN protokol je razvijen u tvrtki Robert Bosch GmbH s ciljem poboljšanja sigurnosti i efikasnosti korištenja automobila. CAN protokol smanjuje kompleksnost sustava komunikacije između ECU-a unutar vozila tako što se svi ECU-i povezuju na CAN sabirnicu (prikazano na slici 3.1), a ne međusobno ili pak zajedno na neki zajednički mrežni čvor (prikazano na slici 3.2), tj. središnje računalo.



Sl. 3.1. *Upravljačke jedinice povezane CAN mrežom*

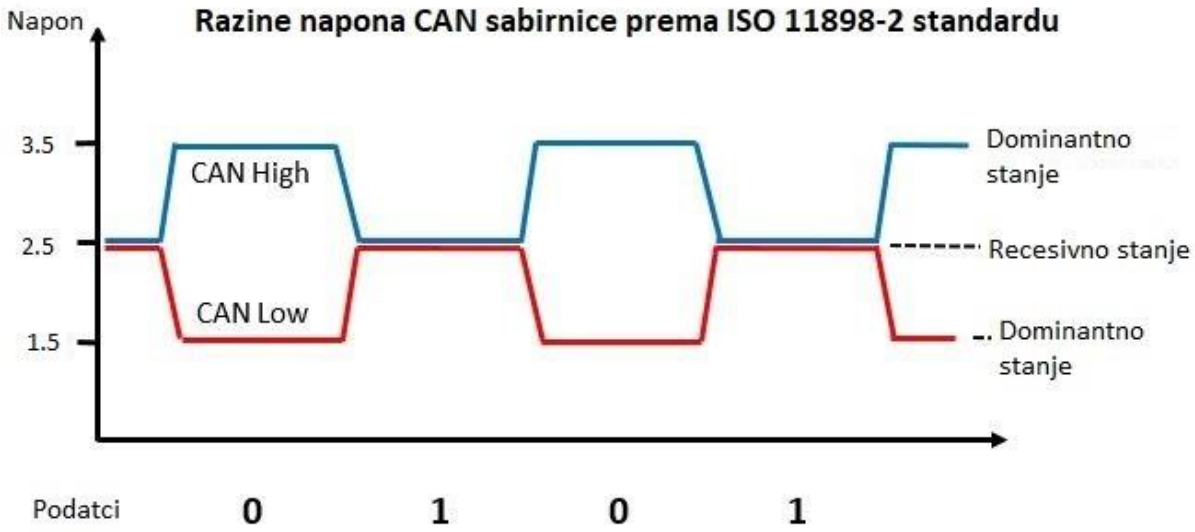


Sl. 3.2. Upravljačke jedinice povezane preko središnjeg računala

3.1. Karakteristike CAN mreže

CAN sabirnica realizirana je pomoću dva vodiča, CAN *high* (visokonaponski – od 2.5V do 3.5V) i CAN *low* (niskonaponski – od 1.5V do 2.5V) kao što to slika 3.3 prikazuje. Vodiči su upleteni kako bi otpornost na elektromagnetsko zračenje bila veća. Razlog uplitanja vodiča CAN sabirnice leži u tome što CAN mreža za slanje poruka koristi diferencijalni naponski signal, pa ako dođe do smetnji razine napona na oba vodiča će biti povećane ili smanjene za jednaki iznos, a diferencijalni napon ostaje isti. Svaki uređaj kojeg se želi spojiti na CAN sabirnicu mora imati CAN sučelje koje se sastoji od CAN upravljač i CAN primopredajnika. Primopredajnik se fizički spaja na CAN sabirnicu, te ako dođe to prevelikog napona na CAN sabirnici CAN primopredajnik može izgorjeti ali će CAN upravljač ostati ispravan. Ako se radi o CAN primopredajniku visoke brzine, brzina prijenosa doseže 1 Mbit/s, a ako se radi o CAN primopredajniku niske brzine, brzine prijenosa su do 125 Kbit/s. Na CAN mreži logička jedinica predstavlja diferencijalni napon od 0V, a logička nula predstavlja diferencijalni napon od 2V. U stanju mirovanja, tj. kada niti jedan od povezanih uređaja ne šalje podatke putem CAN sabirnice, sabirnica se nalazi u recesivnom stanju, tj. u stanju logičke jedinice. U recesivnom stanju razina napona CAN *high* i CAN *low* vodiča je 2.5V, što znači da je diferencijalni napon između CAN *high* i CAN *low* vodiča 0V. Početkom slanja podataka sabirnica prvo prelazi u dominantno stanje, tj. stanje s diferencijalnim naponom od 2V, a zatim se slijedom šalju željeni podatci. CAN protokol

funkcionira na način odašiljanja poruka, tj. svi spojeni uređaji primaju poruku i sami određuju da li im je ta poruka potrebna.



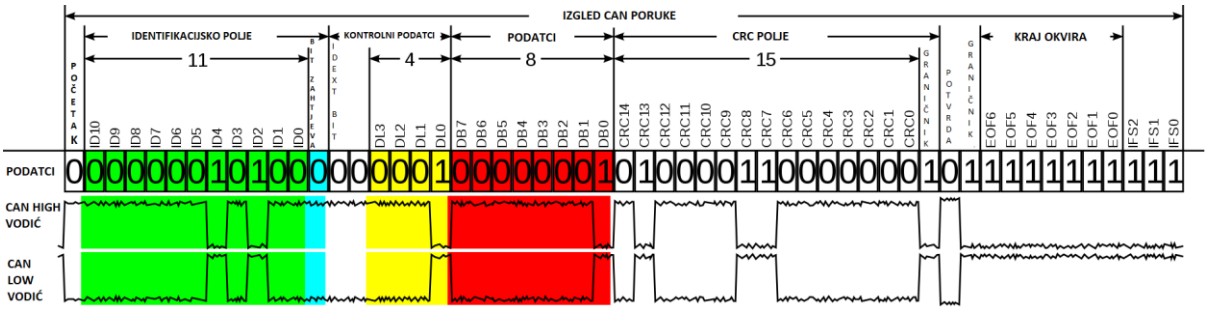
Sl. 3.3. *Razine napona CAN sabirnice*

3.2. CAN okviri

U CAN protokolu moguće je koristiti tri okvira podataka: podatkovni okvir (engl. *data frame*), daljinski okvir (engl. *remote frame*), te okvir koji služi za prijenos podataka o pogrešci (engl. *error frame*).

Korištenjem podatkovnog okvira maksimalni broj podataka koji se može prenijeti je 64 bita. Prijenos podataka započinje slanjem SOF (engl. *start of frame*) bita. Kao što je prethodno navedeno, CAN sabirnica u stanju mirovanja nalazi se u recesivnom stanju pa zato slanje svake poruke započinje slanjem dominantnog, odnosno SOF bita. Nakon SOF bita šalje se ID (engl. *identifier*) niz od 11 bitova. Pomoću ID niza bitova određuje se i prioritet poruke, što je manja vrijednost ID-a prioritet poruke je veći. Nakon ID niza bitova šalje se RTR (engl. *remote transmission request*) bit, koji se postavlja na nulu ako se radi o slanju podatkovnog okvira, odnosno na jedinicu ako se radi o slanju daljinskog okvira. Nakon slanja RTR bita, šalje se IDE (engl. *identifier extension bit*) koji mora biti postavljen na nulu ako se radi o standardnom formatu okvira s 11 bitnim ID nizom. Nakon IDE bita šalje se bit koji je rezerviran, tj. njegova vrijednost mora biti nula i ne bi se trebala mijenjati. Poslije slanja rezerviranog bita šalje se DLC (engl. *data length code*) koji je veličine četiri bajta i govori koliko će bajtova podataka biti poslano. Poslije DLC niza šalju se sami podatci, koji mogu

biti veličine od 0 do 64 bita. Nakon poslanih podataka šalje se izračunati CRC (engl. *cyclic redundancy check*), koji služi za provjeru ispravnost poslanih, odnosno primljenih podataka. CRC završava s graničnikom (engl. *delimiter*) čija vrijednost mora biti 1. Nakon CRC graničnika pošiljatelj šalje ACK (engl. *acknowledge*) bit postavljen na vrijednost 1, a primatelj može tu vrijednost promijeniti na 0 ako se radi o ispravno primljenoj poruci. Slanje podataka završava sa ACK graničnikom i EOF (engl. *end of frame*) graničnikom čija vrijednost mora biti 1. Standardna CAN poruka prikazana je na slici 3.4.



Sl. 3.4. *Standardna CAN podatkovna poruka*

Prošireni okvir podatkovne CAN poruke razlikuje se od standardnog oblika u tome što ima dodatnih 18 bita za ID poruke, a razlika je i u redoslijedu podataka nakon SOF bita i prije DLC niza bitova. U proširenom okviru nakon SOF bita šalje se 11 bitni ID, kao i kod standardnog okvira, ali se nakon toga šalje SSR (engl. *substitute remote request*) bit koji mora biti postavljen na 1, te nakon toga i IDE bit koji sada mora biti postavljen na 1 jer se radi o proširenom okviru. Nakon IDE bita šalje se drugi dio ID-a poruke koji se sastoji od 18 bitova. Nakon drugog dijela ID niza šalju se dva rezervirana bita čije bi vrijednosti trebale biti postavljene na nula. Nakon toga ostali podatci šalju se kao i kod standardnog okvira CAN poruke. Proširena CAN poruka prikazana je na slici 3.5.

Na CAN mreži podatke uglavnom šalju različiti uređaji čitajući vrijednosti sa senzora, a ECU-i odlučuju jesu li im ti podatci potrebi ili ne, međutim nekad je moguće da ECU zahtjeva podatke od senzora i za to se koristi daljinski okvir. Razlika između daljinskog i podatkovnog okvira je u tome što se kod daljinskog okvira RTR bit šalje s vrijednošću 1 i daljinski okvir nema dio za podatke, tj. nema *data field.*

Naziv polja	Duljina (biti)
POČETAK (SOF)	1
IDENTIFIKACIJSKI BROJ A	11
ZAMJENA ZA BIT ZAHTJEVA	1
BIT PRODUŽENOG IDENTIFIKACIJSKOJ BROJA	1
IDENTIFIKACIJSKI BROJ B	18
BIT ZAHTJEVA ZA PONOVNO SLANJE	1
REZERVIRANI BITOVI	2
DULJINA PODATKOVNOG DIJELA	4
PODATCI	0–64 (0-8 bytes)
CRC	15
CRC GRANIČNIK	1
ACK	1
ACK GRANIČNIK	1
KRAJ PORUKE (EOF)	7

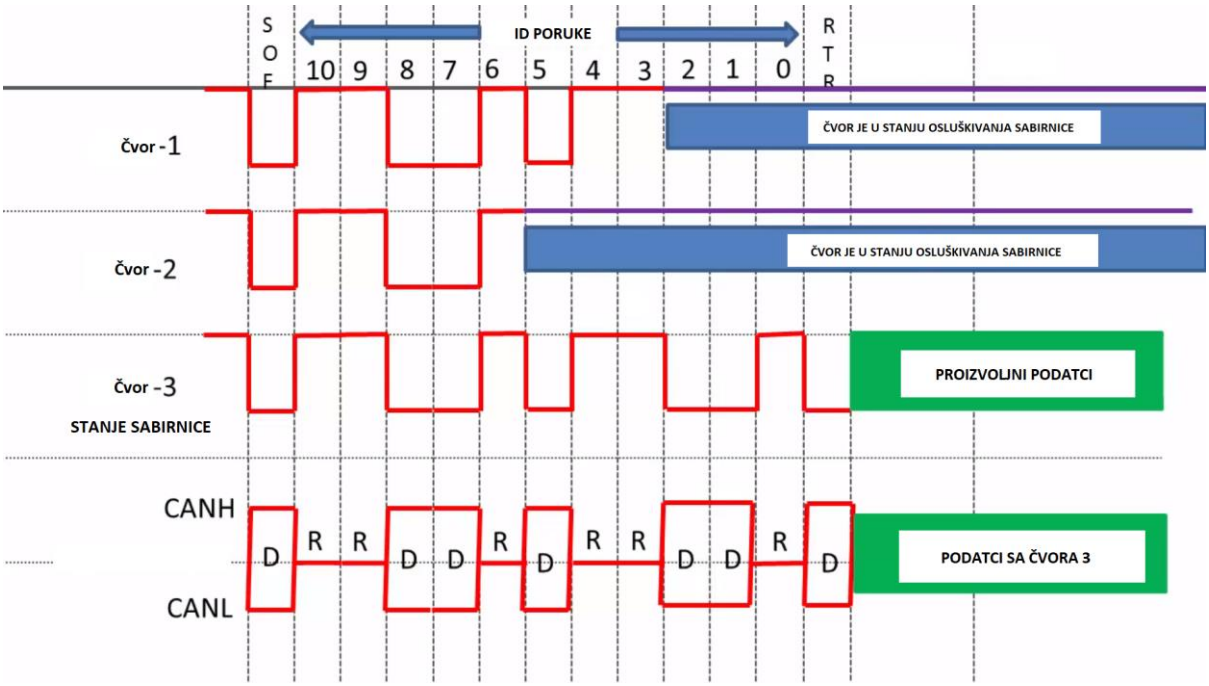
Sl. 3.5. *Proširena CAN podatkovna poruka*

Okvir za slanje podataka o pogrešci se sastoji od dva dijela. Prvi dio se sastoji od zastavica pogreške (engl. *error flags*) gdje se pomoću njihove superpozicije točno određuje gdje je došlo do pogreške. Prvi dio čini od 6 do 12 bitova. Drugi dio se sastoji od graničnika pogreške (engl. *error delimiter*) kojeg čine 8 recesivnih bitova.

3.3. Prioriteti prilikom slanja poruka

Kao što je i ranije spomenuto CAN sabirnica funkcionira tako što se na nju spaja više ECU-a, a poruke se izmjenjuju tako što neki od izvora odašilje poruke dok svi drugi čvorovi CAN mreže te poruke primaju i određuju jesu li im potrebne ili ne. Takav način rada zasniva se na *multi master* arhitekturi (engl. *multi-master architecture*). Problem kod ovakve komunikacije nastaje kada više izvora želi poslati poruku, u isto vrijeme. Taj problem se rješava CSMA/CA (engl. *carrier-sense multiple access with collision avoidance*) metodom pomoću koje se određuje prednost među čvorovima koji istovremeno žele poslati poruku. Ta metoda omogućuje pristup mreži većem broju čvorova, gdje se kolizija izbjegava tako da čvorovi koji imaju manji prioritet pokušaju ponovno poslati poruku kada uoče da je sabirnica u stanju mirovanja.

Svi čvorovi koji žele poslati poruku prvo šalju ID poruke u binarnom obliku na CAN sabirnicu. Logikom arbitraže se odlučuje hoće li neki čvor moći nastaviti slati podatke ili će morati pričekati dok se pošalje poruka s većim prioritetom. Svaki čvor prvo šalje ranije opisani SOF, a zatim bit po bit ID poruke. Svaki bit se uspoređuje s trenutnim stanjem CAN sabirnice.



Sl. 3.6. *Prikaz postupka arbitraže prilikom istovremenog slanja podataka s više čvorova*

Na slici 3.6 prikazan je slučaj kada 3 čvora pokušavaju istovremeno pristupiti sabirnici. Brojevi od 0 do 10 predstavljanju binarni zapis identifikacijskih brojeva poruka koje čvorovi pokušavaju poslati gdje se najznačajniji bit nalazi na desnoj strani. Prije nego li mogu poslati podatke svaki od čvorova mora poslati 11 bitni identifikacijski broj ako se radi o standardnom okviru, ili 29 bitni identifikacijski broj ako se radi o proširenom okviru. U ovom slučaju radi se o standardnom okviru. Čvor 1 šalje ID vrijednosti 0x65D, čvor 2 šalje ID vrijednosti 0x676, a čvor 3 0x659. Kako čvor 3 ima identifikacijski broj s najmanjom vrijednosti on će i dobiti pravo za pristup sabirnici, jer vrijednost identifikacijskog broja i prioritet poruke obrnuto proporcionalno ovise. Sve do petog bita identifikacijski brojevi poruka su jednaki tako se stanja identifikacijskih brojeva kopiraju na sabirnicu. Prilikom slanja petog bita čvor 2 prestaje sa slanjem podataka, tj. prebacuje se u stanje osluškivanja zato što pokušava poslati recesivni bit, a čvorovi 1 i 3 šalju dominantni bit, te zato imaju veći prioritet od čvora 2. Ista situacija se događa prilikom slanja drugog bita gdje čvor 1 šalje recesivni bit, a čvor 3 dominantni, te zbog „i“ logike pomoću koje je realizirana CAN

sabirnica čvor 3 dobiva prioritet, a time i arbitražu. Nakon toga čvor 1 se prebacuje u stanje osluškivanja, čvor 3 šalje preostale bitove identifikacijskog polja poruke i nakon toga šalje same podatke.

3.4. Umetanje kontrolnog bita

Sinkronizacija u prijenosu poruka na CAN sabirnici se odvija pomoću detekcije padajućeg brida, tj. pomoću detekcije prijelaza iz dominantnog u recesivno stanje. Ovakva vrsta sinkronizacije se koristi zato što se CAN sabirnica u stanju mirovanja nalazi u recesivnom stanju pa se na taj način detektira kada je poslan SOF bit. Za zadržavanje sinkronizacije koristi se detekcija rastućeg brida, tj. detekcija prijelaza iz recesivnog u dominantno stanje te se u tu svrhu koristi umetanje kontrolnog bita, tj. bita popune (engl. *bit stuffing*).

Umetanje bita popune započinje od SOF bita i završava sa slanjem zadnjeg bita CRC niza. Umetanje bita popune odvija se tako da se na svakih 5 bita iste vrijednosti poslanih u nizu šalje jedan bit popune koji je komplementarne vrijednosti. Ako se npr. pokušava poslati binarni broj 11111100, taj broj će prilikom slanja na sabirnicu imati sljedeću vrijednost 11111**0**100. Umetanje bita popune izvršava pošiljatelj poruke, a na prijemnoj strani će se prepoznati umetnuti bit i biti će zanemaren.

3.5. Detekcija pogrešaka

Detekcija pogrešaka prilikom slanja podataka putem CAN sabirnice odvija se pomoću sljedećih mehanizama:

Promatranje bitova (engl. *bit monitoring*) - Promatranje bitova obavlja svaki primopredajnik spojen na CAN sabirnicu. Svaki primopredajnik prati razinu poslanog signala i razinu signala koji se nalazi na sabirnici (engl. *reads back*), te ako su uspoređene razine različite signalizira se da je došlo do pogreške.

Promatranje formata poruke (engl. *frame check*) - Određeni dijelovi CAN poruke uvijek imaju isti format, tj. točno je definirano koji bit mora imati određenu vrijednost u nekom trenutku. Ti dijelovi su CRC graničnik, ACK graničnik, EOF i bitovi prekida. Ako CAN upravljač nekog čvora detektira neispravnu vrijednost u jednom od tih dijelova signalizira se da je došlo do pogreške.

Promatranje umetnutih bitova (engl. *bit stuffing*) - Ako se prilikom prijenosa podataka pojavi niz od 5 bitova iste vrijednosti čvor će to detektirati i umetnuti bit suprotne vrijednosti,

te nastaviti s prijenosom podataka. Također prilikom primanja podataka čvorovi će detektirati umetnute bitove i ukloniti ih prilikom obrade primljenih podataka. Umetanje ovih bitova omogućuje uočavanje pogrešaka prilikom prijenosa podataka, tj. signalizirati će se pogreška ako neki čvor primi više od 5 uzastopnih bitova iste vrijednosti.

Promatranje bitova potvrde (engl. *acknowledge check*) - Svi čvorovi koji su spojeni na CAN sabirnicu i koji ispravno prime poruku, bez obzira na to da li je ta poruka relevantna za taj čvor ili ne, u poruci umeću bit dominante vrijednosti na mjestu bita potvrde (engl. *Acknowledgement slot*). Primopredajnik prilikom slanja poruke na mjestu bita potvrde zapisuje bit recesivne vrijednosti i ako nakon slanja poruke primopredajnik ne detektira dominantnu vrijednost na mjestu bita potvrde to znači da poruka nije ispravno primljena od strane drugih čvorova, te se signalizira pogreška.

Provjera ispravnosti pomoću kontrolnog zbroja (engl. *verifying checksum*) - Svaka poruka sadrži svoj kontrolni zbroj koji izračunava primopredajnik. Nakon primanja poruke svaki čvor ponovno računa kontrolni zbroj te ga uspoređuje sa zapisanim kontrolnim zbrojem u poruci. Ako se zapisani i izračunati kontrolni zbrojevi razlikuju signalizira se pogreška.

3.6 Mehanizmi praćenja pogrešaka čvorova

Svaki čvor spojen na CAN sabirnicu može detektirati pogreške prilikom slanja ili primanja podataka i poslati informaciju o pronađenoj pogrešci. Ako neki čvor detektira pogrešku, on će poslati zastavicu pogreške (engl. *error flag*) i time uzrokovati uništavanje svog trenutnog prometa na sabirnici. Ostali čvorovi će primiti informaciju o pogrešci koja se dogodila i primijeniti odgovarajuću radnju, npr. odbaciti primljenu poruku.

Svaki čvor ima dva brojača pogrešaka, jedan koji bilježi pogreške koje su se dogodile prilikom odašiljanja podataka, a drugi bilježi pogreške koje su se dogodile prilikom primanja podataka. Navedeni brojači svoje vrijednost smanjuju i uvećavaju prema određenim pravilima, npr. primopredajnik koji uoči pogreške prilikom slanja podataka brže će povećavati svoj broj brojač pogrešaka nego li čvorovi koji uoče pogreške prilikom primanja neispravnih podataka. Ova tehnika se provodi jer su greškama najčešće uzroci neispravni upravljači koji kontroliraju slanje podataka, a ne upravljači koji kontroliraju primanje podataka.

Svaki čvor započinje u aktivnom načinu rada (engl. *error active),* a kada se vrijednost bilo kojeg od dva navedena brojača pogrešaka poveća preko 127 čvor prelazi u pasivni način rada (engl. *error passive*). Kada se brojač pogrešaka koji bilježi probleme prilikom slanja

poruka poveća preko 255 čvor se isključuje s CAN sabirnice. Čvor u aktivnom načinu rada odašilje aktivne zastavice pogreške (engl. *active error flags*) kad detektira pogrešku. Čvor u pasivnom načinu rada odašilje pasivne zastavice pogreške (engl. *passive error flags*) kad detektira pogrešku. Razlika između tih zastavica je u tome što prilikom odašiljanja aktivne zastavice pogreške sav trenutni promet na sabirnici se uništava, tj. svi čvorovi odbacuju primljenu poruku, a slanjem pasivne zastavice pogreške promet na sabirnici se nastavlja normalno.

3.7 CAN FD

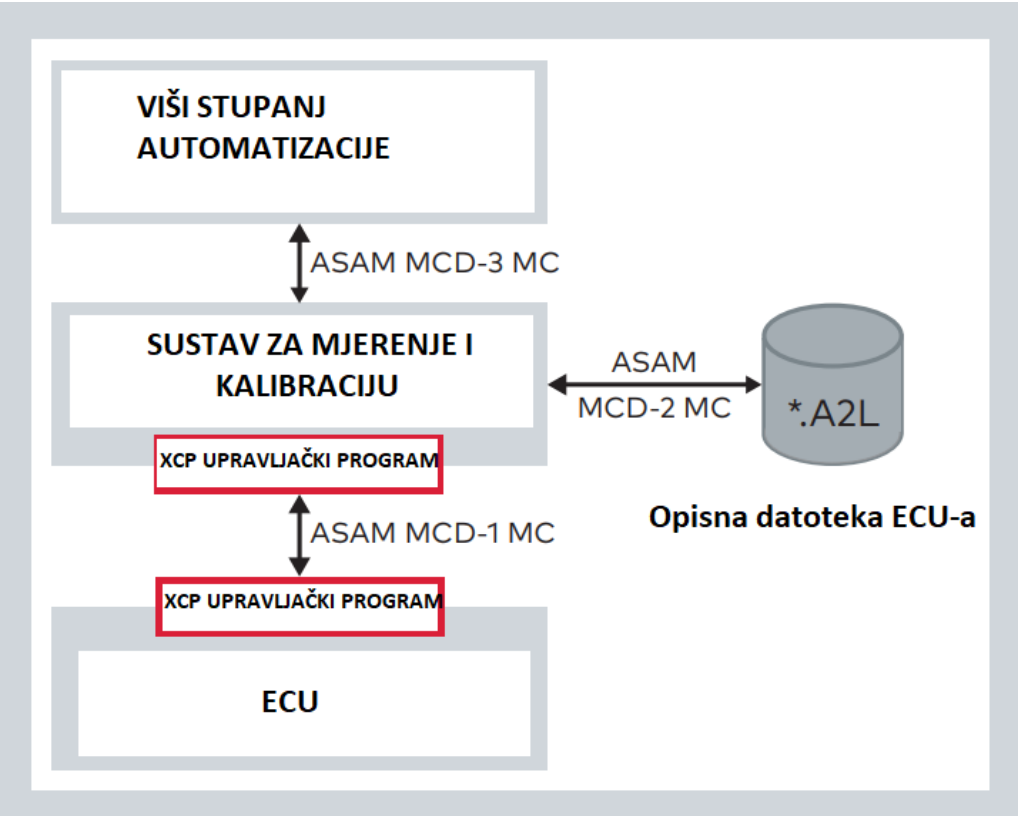
CAN FD (engl. *can with flexible data-rate*) je proširenje originalnog CAN protokola. CAN FD protokol je napravljen kako bi se zadovoljili sve veći suvremeni zahtjevi automobilske industrije, tj. kako bi se omogućilo povećanje propusnosti sabirnice. Korištenjem CAN FD protokola softver i obrada podataka je približena realnom vremenu kroz smanjenje kašnjenja između slanja naredbe i slanja samih podataka. U standardnom CAN protokolu unutar jednog podatkovnog okvira moguće je poslati 8 bajta podataka, a korištenjem CAN FD protokola broj podataka koji je moguće poslati je povećan na 64 bajta. Poboljšanjem algoritma za računanje kontrolnog zbroja povećana je i mogućnost detekcije pogrešaka, tj. CAN FD protokol smanjuje broj neotkrivenih pogrešaka. CAN FD protokol zasniva se na ideji da kada samo jedan čvor šalje podatke brzina prijenosa može biti i veća od 1 Mbit/s, dok za vrijeme istovremenog slanja podataka od stane više čvorova, npr. za vrijeme određivanja prioriteta između čvorova, brzina slanja je do 1 Mbit/s.

Kako bi se mogli razlikovati CAN FD podatkovni okrivi od klasičnih okvira koristi se FDF bit (engl. *FD frame bit*). Ako je FDF bit postavljen na recesivnu vrijednost radi se o CAN FD okviru. Novo uvedeni bit je i BRS bit (engl. *bit rate switch bit*), pomoću kojeg se određuje brzina prijenosa podataka. Ako je BRS bit postavljen na recesivnu vrijednost onda se koristi veća brzina prijenosa podataka, a ako je BRS bit postavljen na dominantnu vrijednost ista, tj. manja brzina prijenosa koristi se prilikom faze određivanja prioriteta i prilikom slanja podataka. U ovom diplomskom radu korišten je standardni CAN protokol.

4. XCP PROTOKOL

Kalibracija ECU-a je proces određivanja optimalnih vrijednosti za različite parametre unutar ECU-a kako bi se postigle željene performanse. Kalibracija ECU-a je složen postupak koji uključuje dizajniranje testova, prikupljanje podataka, analizu podataka i kalibriranje istih. Željeni rezultati kalibracije su identificiranje i postavljanje ravnoteže između performansi motora, emisije plinova i potrošnje goriva. Fizička veza između razvojnog alata i ECU-a je protokol koji omogućuje mjerenja i kalibraciju, a XCP protokol je postao standard koji se u te svrhe koristi.

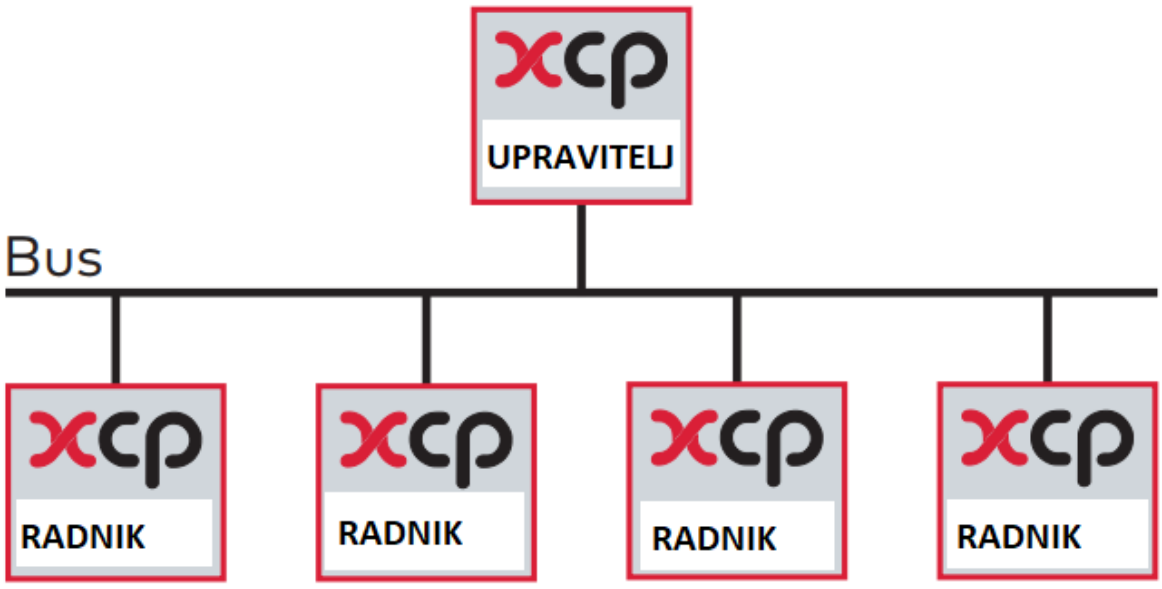
XCP protokol je standardizirao ASAM (engl. *association for standardisation of automation and measuring systems*), a ASAM je organizacija proizvođača automobilske opreme. XCP protokol je nasljednik CCP (engl. *CAN calibration protocol*) protokola. Svrha CCP protokola je bila omogućavanje čitanja i pisanja u memoriju ECU-a korištenjem CAN protokola. Razvijanjem tehnologija i povećanjem zahtjeva na automobilsku industriju pojavili su se i novi protokoli koji se koriste za razmjenu podataka, a ti protokoli su LIN, ETHERNET i FlexRay. XCP protokol omogućuje kalibriranje ECU-a preko bilo kojeg od tih protokola. U ovom diplomskom radu XCP protokol je implementiran preko CAN protokola.



Sl. 4.1. *ASAM Model sučelja[3].*

Na slici 4.1. je prikazan ASAM model sučelja. ASAM MCD-1 MC odnosi se na sučelje između ECU-a i sustava za mjerenje i kalibriranje. Ovo sučelje sadrži fizičke dijelove i dijelove vezane uz sam protokol. ASAM MCD-1 MC sučelje zamijenilo je 2 sučelja, a to su ASAP1a i ASAP1b sučelje. ASAP1a sučelje se koristilo kod CCP protokola, a služilo je za povezivanje ECU-a i sustava za kalibraciju i mjerenja, a ASAP1b je predstavljao upravo taj sustav za kalibraciju i mjerenja kojeg je ASAP1a povezivao sa ECU-om. ASAP1b više nije potrebno jer se koristio za CCP protokol, a ASAP1a sučelje nije bilo opće prihvaćeno pa danas više nema primjenu. ASAM MCD-2 MC sa slike 4.1. odnosi se na način upravljana s memorijom ECU-a. Kako se XCP protokol zasniva na korištenju memorijskih adresa bilo bi vrlo teško kada bi korisnik morao ručno unositi memorijske adrese kojima želi upravljati, pa se zato koristi A2L datoteka. Korištenje A2L datoteke omogućuje korištenje smislenijih i opisnih imena umjesto adresa. ASAM MCD-3 MC sučelje sa slike 4.1. koristi se za povezivanje drugih sustava sa sustavima za mjerenje i kalibraciju.

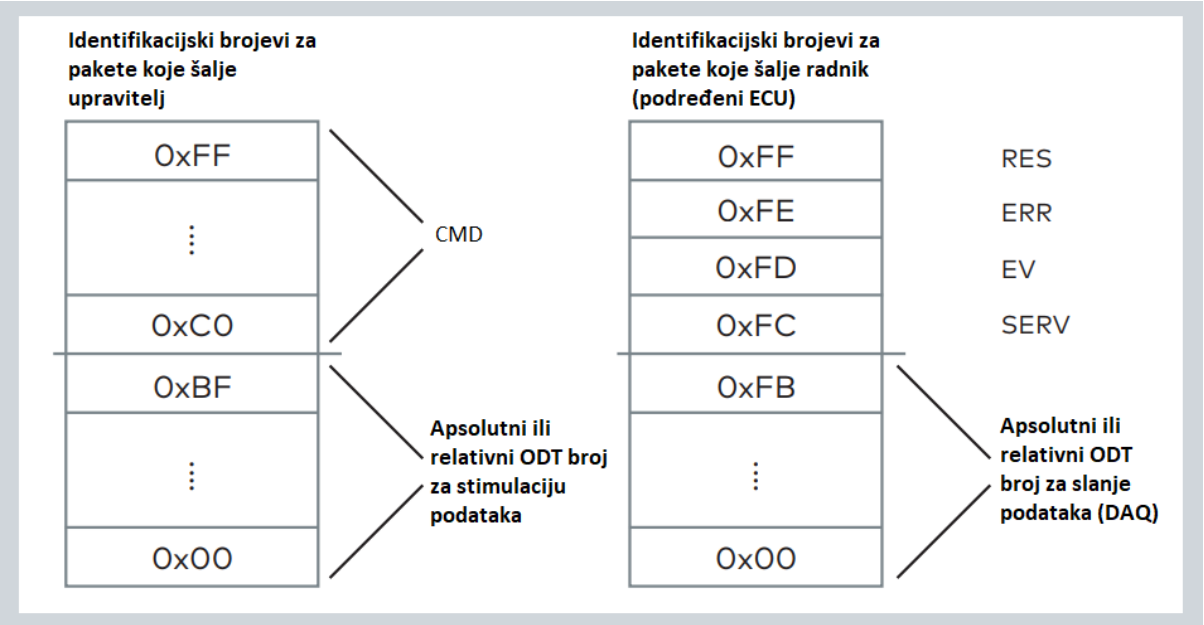
XCP protokol se zasniva na upravitelj-radnik principu (engl. *master-slave principle*). ECU predstavlja radnika, a alat za kalibraciju je upravitelj. Radnik može komunicirati samo s jednim upraviteljem u određenom trenutnu, a upravitelj može istovremeno komunicirati s više radnika. Slika 4.2 prikazuje princip rada XCP protokola.



Sl. 4.2. *Princip rada XCP protokola [3].*

4.1. Izgled XCP poruka

XCP podatci se razmjenjuju između upravitelja i radnika putem poruka, a cijeli XCP podatkovni okvir poruke se ugrađuje u okvir transportnog sloja, što je CAN u ovom slučaju. XCP paket se sastoji od 3 komponente: identifikacijskog polja, polja za vremensku oznaku i polja za podatke. Svako identifikacijsko polje počinje s identifikacijskim brojem.



Sl. 4.3. *Izgled XCP paketa [3].*

Prilikom razmjene poruka između upravitelja i radnika, oba sudionika u komunikaciji moraju moći odrediti tko je pošiljatelj poruke pa se zato koristi identifikacijsko polje i zato svaka poruka započinje s PID-om (engl. *packet identifier*). Prilikom razmjene određenog tipa poruka nekad je potrebno slati i vremensku oznaku uz same podatke. To je potrebno kada se šalju podatci o mjerenjima parametara, jer je osim samih vrijednosti nekih varijabli potrebno znati i kada točno su te varijable imale tu vrijednost. Primjer XCP paketa prikazan je na slici 4.3.

4.2. Vrste XCP poruka

U XCP protokolu postoje dvije vrste poruka, a to su CTO (engl. *command transfer object*) i DTO (engl. *data transfer object*) poruke.

CTO se koristi za slanje naredbi od upravitelja prema radniku i obrnuto. To se koristi za izvršavanje naredbi vezanih za sam protokol (CMD), za slanje odgovora na poslanu

naredbu ili zahtjev (RES), za slanje pogrešaka (ERR), za slanje događaja (EV) i za slanje servisnih zahtjeva (SERV).

DTO se koristi za sinkroni prijenos podataka između upravitelja i ECU-a. DAQ (engl. *synchronous dana acquisition*) podatci se prenose s ECU-a upravitelju, a STIM (engl. *synchronous dana stimulation*) se prenose od XCP upravitelja prema ECU-u.

4.3. Vrste XCP naredbi

XCP protokol je vrlo skalabilan i u svojoj implementaciji što znači da nije potrebno implementirati svaku naredbu. U A2L datoteci navedene su dostupne naredbe, a ako postoji razlike između navedenih naredbi u A2L datoteci i stvarnoj implementaciji na uređaju XCP upravitelj će to primijetiti prilikom komunikacije s ECU-om. Ako upravitelj pošalje naredbu koja nije implementirana na ECU-u, XCP upravitelj će to doznati po odgovoru ECU-a.

Naredbe su organizirane u sljedeće skupine: standardne naredbe, naredbe za kalibraciju, naredbe za straničenje (engl. *page commands*), naredbe za programiranje i naredbe za DAQ mjerenja. Standardne naredbe prikazane su na slici 4.4, naredbe za kalibraciju prikazane su na slici 4.5, naredbe za straničenje prikazane su na slici 4.6, naredbe za programiranje prikazane su na slici 4.9, a naredbe korištene za DAQ liste prikazane su na slikama 4.7 i 4.8. Ako neka od ovih grupa nije potreba onda ju nije obavezno niti implementirati, a ako je grupa potrebna onda postoje neke naredbe koje je obavezno implementirati dok su ostali iz grupe neobavezni.

NAREDBA	PID NAREDBE	NAREDBA OBAVEZNA
CONNECT	0xFF	DA
DISCONNECT	0xFE	DA
GET_STATUS	0xFD	DA
SYNCH	0xFC	DA
GET_COMM_MODE_INFO	0xFB	NE
GET_ID	0xFA	NE
SET_REQUEST	0xF9	NE
GET_SEED	0xF8	NE
UNLOCK	0xF7	NE
SET_MTA	0xF6	NE
UPLOAD	0xF5	NE
SHORT_UPLOAD	0xF4	NE
BUILD_CHECKSUM	0xF3	NE
TRANSPORT_LAYER_CMD	0xF2	NE
USER_CMD	0xF1	NE

Sl. 4.4. *Standardne XCP naredbe [3].*

NAREDBA	PID NAREDBE	NAREDBA OBAVEZNA
DOWNLOAD	0xF0	DA
DOWNLOAD_NEXT	0xEF	NE
DOWNLOAD_MAX	0xEE	NE
SHORT_DOWNLOAD	0xED	NE
MODIFY_BITS	0xEC	NE

Sl. 4.5. *XCP naredbe za kalibraciju [3].*

NAREDBA	PID NAREDBE	NAREDBA OBAVEZNA
SET_CAL_PAGE	0xEB	DA
GET_CAL_PAGE	0xEA	DA
GET_PAG_PROCESSOR_INFO	0xE9	NE
GET_SEGMENT_INFO	0xE8	NE
GET_PAGE_INFO	0xE7	NE
SET_SEGMENT_MODE	0xE6	NE
GET_SEGMENT_MODE	0xE5	NE
COPY_CAL_PAGE	0xE4	NE

Sl. 4.6. *XCP naredbe straničenja[3].*

NAREDBA	PID NAREDBE	NAREDBA OBAVEZNA
SET_DAQ_PTR	0xE2	DA
WRITE_DAQ	0xE1	DA
SET_DAQ_LIST_MODE	0xE0	DA
START_STOP_DAQ_LIST	0xDE	DA
START_STOP_SYNCH	0xDD	DA
WRITE_DAQ_MULTIPLE	0xC7	NE
READ_DAQ	0xDB	NE
GET_DAQ_CLOCK	0xDC	NE
GET_DAQ_PROCESSOR_INFO	0xDA	NE
GET_DAQ_RESOLUTION_INFO	0xD9	NE
GET_DAQ_LIST_INFO	0xD8	NE
GET_DAQ_EVENT_INFO	0xD7	NE

Sl. 4.7. *XCP* DAQ *naredbe[3].*

NAREDBA	PID NAREDBE	NAREDBA OBAVEZNA
FREE_DAQ	0xD6	DA
ALLOC_DAQ	0xD5	DA
ALLOC_ODT	0xD4	DA
ALLOC_ODT_ENTRY	0xD3	DA

Sl. 4.8. *XCP* DAQ *naredbe za dinamičke liste[3].*

NAREDBA	PID NAREDBE	NAREDBA OBAVEZNA
PROGRAM_START	0xD2	DA
PROGRAM_CLEAR	0xD1	DA
PROGRAM	0xD0	DA
PROGRAM_RESET	0xCF	DA
GET_PGM_PROCESSOR_INFO	0xCE	NE
GET_SECTOR_INFO	0xCD	NE
PROGRAM_PREPARE	0xCC	NE
PROGRAM_FORMAT	0xCB	NE
PROGRAM_NEXT	0xCA	NE
PROGRAM_MAX	0xC9	NE
PROGRAM_VERIFY	0xC8	NE

Sl. 4.9. *XCP naredbe za programiranje [3].*

4.4. Razmjena CTO naredbi

CTO naredbe koriste se prilikom slanja podataka od upravitelja prema radniku, tj. ECU-u, i obrnuto. Kada ECU primi poruku od upravitelja on na nju mora odgovoriti pozitivnim ili negativnim odgovorom. Svaka naredba ima jedinstveni identifikacijski broj, te se uz naredbu mogu poslati i drugi parametri specifični za tu naredbu.

POZICIJA	VRSTA PODATKA	ZNAČENJE PODATKA
0	BYTE	IDENTIFIKACIJSKI BROJ NAREDBE
1..MAX_CTO-1	BYTE	DODATNI PARAMETRI

Sl. 4.10. *Izgled XCP CTO naredbe [3].*

MAX_CTO označava maksimalnu duljinu CTO paketa u bajtovima, a maksimalna duljina CTO paketa ovisi o samom ECU-u te zato upravitelj od radnika, tj. ECU-a može zatražiti tu informaciju. Način formiranja CTO naredbe prikazan je na slici 4.10.

POZICIJA	VRSTA PODATKA	ZNAČENJE PODATKA
0	BAJT	IDENTIFIKACIJSKI BROJ POZIVITNOG ODGOVORA = 0XFF
1..MAX_CTO-1	BAJT	DODATNI PARAMETRI

Sl. 4.11. *Izgled pozitivnog odgovora na CTO naredbu [3].*

POZICIJA	VRSTA PODATKA	ZNAČENJE PODATKA
0	BAJT	IDENTIFIKACIJSKI BROJ NEGATIVNOG ODGOVORA = 0XFE
1	BAJT	KOD POGREŠKE
2..MAX_CTO-1	BAJT	DODATNI PARAMETRI

Sl. 4.12. *Izgled negativnog odgovora na CTO naredbu [3].*

Kao što je vidljivo na slikama 4.11 i 4.12 parametri se mogu prenositi prilikom slanja i pozitivnog i negativnog odgovora. Osim što se podrazumijeva da se prilikom slanja pozitivnog odgovora pošalju i neke dodatne informacije vezane uz izvršavanje same naredbe, dodatne informacije je potrebno slati i prilikom negativnog odgovora jer je poželjno točnije znati zašto se određena pogreška dogodila.

4.5. Načini izmjena naredbi

XCP protokol pruža tri načina za razmjenu naredbi i poruka između XCP upravitelja i XCP radnika, a to su: Standardni način, Blokovski način (engl. *block mode*) i naizmjenični način (engl. *interleaved mode*).

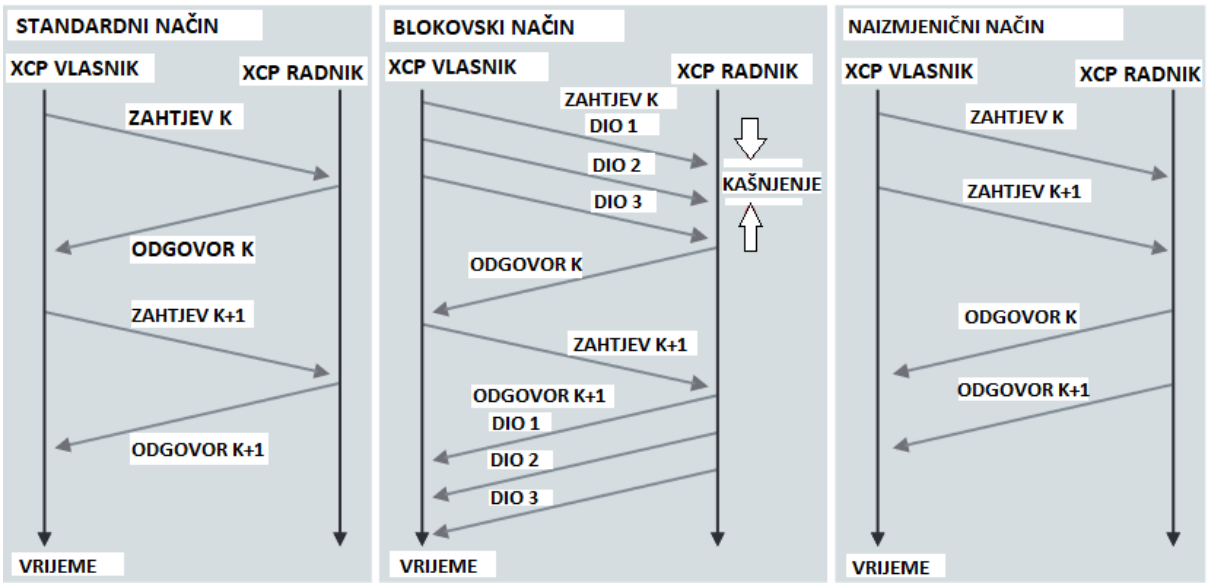
Korištenjem standardnog načina radnik, tj. ECU odgovara odmah nakon primljene naredbe od XCP upravitelja. Osim kod XCP protokola koji se koristi preko CAN protokola za komunikaciju, nije dopušteno da istovremeno više XCP radnika odgovara na zahtjev ili naredbu od upravitelja. Ovo je standardan način komunikacije i njime je omogućeno da se svaka poruka prati do odgovarajućeg radnika.

Korištenje blokovskog načina nije obavezno ali doprinosi uštedi vremena prilikom prijenosa velikog broja podataka, kao npr. pri stalnim slanjem zahtjeva za dohvaćanje podataka s određene memorijske adrese ECU-u i stalnim odgovorima ECU-a na te zahtjeve. Prilikom korištenja blokovskog načina prijenosa podataka potrebno je paziti na performanse što se tiče ECU-a, te se zato mora održavati određeno minimalno vrijeme prilikom slanja između dvije poruke, te maksimalan broj naredbi mora biti poštivan. Navedena ograničenja upravitelj može dobiti ako pošalje radnik *GET_COMM_MODE_INFO* naredbu. Upravitelj ne

mora paziti na ova ograničenja što se tiče samog računala jer su performanse i resursi računala gotovo uvijek dovoljni za prihvaćanje i obradu podataka koje šalje mikroupravljač.

U naizmjeničnom načinu rada XCP upravitelj šalje više naredbi za redom, a ECU odgovara na te naredbe istim redoslijedom kojim ih je primao. Naizmjenični način rada također nije obavezan za korištenje ali se implementira zbog performansi. U praksi se vrlo rijetko koristi jer se zbog istih potreba češće upotrebljava blokovski način rada.

Prikazi standardnog, blokovskog i naizmjeničnog načina rada vidljivi su na slici 4.13, a u ovom diplomskom radu korišten je standardni način prijenosa podataka.



Sl. 4.13. *Izgled standardnog, blokovskog i naizmjeničnog načina prijenosa poruka [3].*

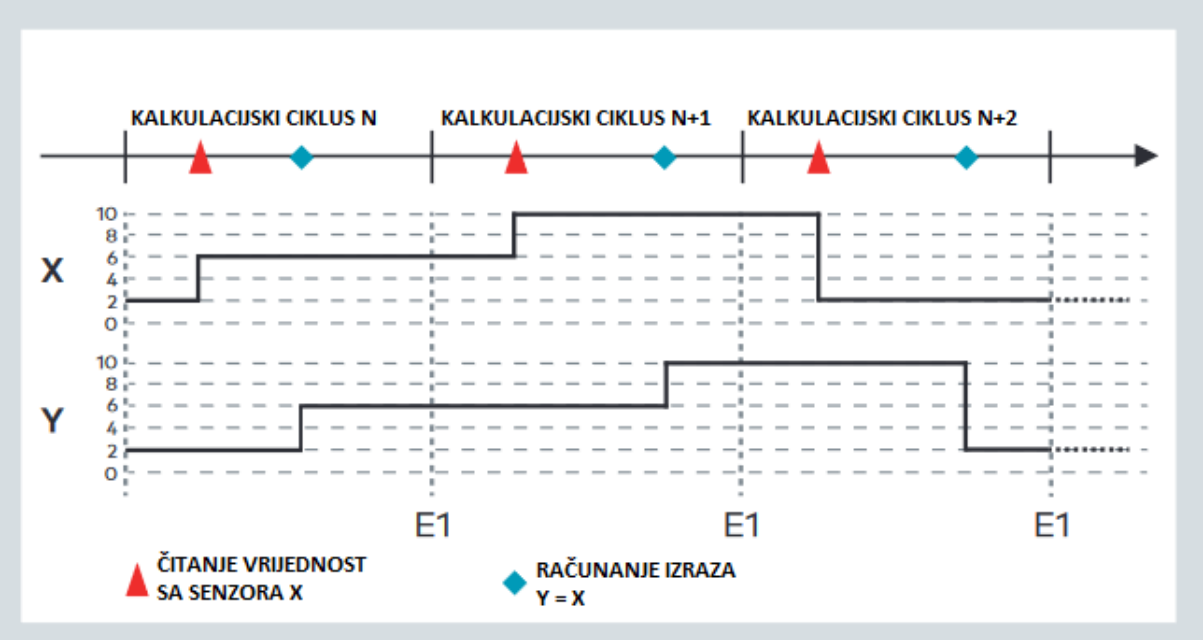
4.6. Razmjena DTO naredbi

DTO razmjena podataka koristi se za izvođenje sinkronih mjerenja i kalibracije. Podatci od radnika, tj. ECU-a šalju se sinkrono s nekim internim događajem. Taj događaj može biti promjena stanja neke varijable, protok vremena i slično. Ovakav način komunikacije podijeljen je u dvije faze. U fazi inicijalizacije upravitelj određuje radniku koje podatke treba slati vezano uz koji događaj. Poslije ove faze upravitelj pokreće mjerenja u radniku i faza mjerenja započinje. Od ovog trenutka radnik šalje podatke sve dok upravitelj ne pošalje naredbu da više nije potrebno slati podatke.

Osim mjerenja putem DTO razmjene podataka moguće je obavljati i kalibriranje, tj. slanje podataka (STIM). Ovakva komunikacija također se odvija u dvije faze. U fazi inicijalizacije upravitelj određuje radniku koje podatke će mu slati. Nakon te faze upravitelj

šalje podatke a STIM procesor, tj. procesor zadužen za obrađivanje i spremate primljenih podataka, sprema podatke. Čim se odviju događaji vezani za stimulaciju podataka primljeni podatci se spremaju u memoriju ECU-a na odgovarajuće mjesto.

Prilikom obavljanja intenzivnijih mjerenja puno je efikasnije koristiti DTO razmjenu podataka s DAQ metodom iz više razloga. Korištenjem standardnog načina, tj. izmjenom CTO naredbi i poruka, upravitelj šalje naredbu vezanu za podatak koji želi pročitati, a radnik na tu naredbu odgovara s porukom u kojoj sprema podatke iz željene memorijske lokacije. Ovim načinom veliko je opterećene sabirnice jer za svako čitanje podataka potrebno je izmijeniti dvije poruke, a često je potrebno čitanje podataka obaviti i više puta u sekundi. Mjerenja korištenjem CTO naredbi mogu biti nepraktična i zbog teže korelacije između primljenih podataka jer podatci primljeni ovim putem ne moraju biti povezani jedan s drugim, tj. moguće je da nisu generirani tijekom istog računalnog ciklusa ECU-a.



Sl. 4.14. *Prikaz kalkulacijskih ciklusa unutar* ECU-*a [3].*

Na slici 4.14 vidljivi su kalkulacijski ciklusi unutar ECU-a, gdje se unutar svakog ciklusa čita vrijednost sa nekog senzora X, pa se zatim određuje vrijednost varijable Y po izrazu Y = X. E1 označava kraj kalkulacijskog ciklusa. Ovi ciklusi se ne moraju periodično ponavljati, tj. mogu ovisiti o drugim događajima unutar ECU-a, ili cjelokupnog vozila, kao što su npr. promjena kuta kotača, promjena brzine obrtaja motora, promjena vrijednost neke varijable itd. Kako se zahtjev za čitanjem varijable Y može poslati u bilo kojem trenutku, moguće je da se to dogodi i u sredini jednog od kalkulacijskog ciklusa, pa tako dolazi do

slanja vrijednosti koja je pročitana sa senzora X, te slanja vrijednosti varijable Y koja je izračunata u prošlom kalkulacijskom ciklusu i tu nastaje problem jer je vidljivo da ti podatci nisu korelirani. Upravo zbog takvih problema poželjno je izbjegavati mjerenja putem izmjene CTO podataka.

Navedene probleme rješava DTO izmjena podataka s DAQ metodom. Opterećenje sabirnice se smanjuje jer upravitelj samo jednom šalje koje podatke želi čitati, a nakon toga ECU šalje željene podatke sve dok od upravitelja ne primi naredbu za prestanak, a kako će ECU željene podatke slati vezano uz neki događaj (protok vremena, promjena vrijednosti neke varijable i sl.) time je osigurano da su podatci korelirani.

4.7. DAQ metoda mjerenja

Kao što je i ranije navedeno DAQ metoda se koristi jer rješava dva glavna problema koja se pojavljuju prilikom mjerenja putem stalnog naređivanja ECU-u da pročita i pošalje podatke, a ti problemi su:

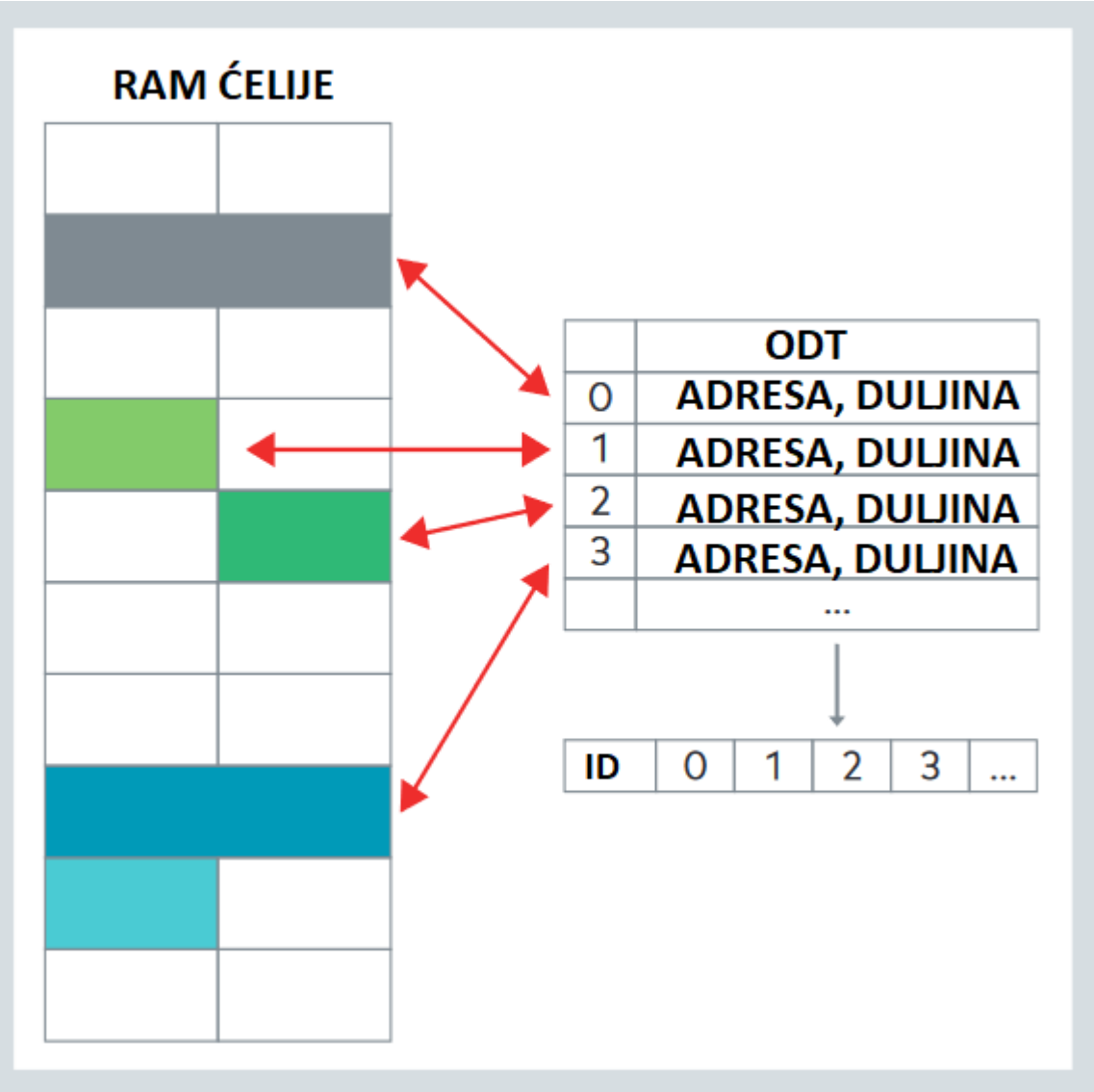
- Korelacija izmjerenih vrijednosti koja se postiže spajanjem prikupljenih vrijednosti s događajem unutar ECU-a, tj. svi željeni podatci se šalju tek kad su svi izračuni završeni.
- Veliko opterećenje sabirnice koje se smanjuje dijeljenjem procesa mjerenja u dvije faze, tj. u fazu inicijalizacije i u fazu samog mjerenja.

Prilikom stvaranja DAQ lista korisnik bira signale čije vrijednosti želi bilježiti. Osim željenog signala korisnik mora odabrati i jedan od dostupnih događaja s kojim želi povezati signal. Popis svih signala i događaja korisnik može pronaći u opis A2L datoteci ECU-a. Nakon konfiguriranja signala koje korisnik želi mjeriti započinje i samo mjerenje. Upravitelj željene signale stavlja u takozvane DAQ liste. Svaki od signala u DAQ listama povezan je s određenim događajem. Sve navedene postavke o signalima i događajima šalju se ECU-u prije mjerenja kako ECU mogao znati kada će poslati određenu vrijednost.

Nakon što je korisnik odabrao signale i započeo mjerenja, ECU mora pročitati vrijednosti odabranih signala i te podatke formirati u pakete poruka. ECU ne formira poruke proizvoljno već po određenim pravilima koja mora znati i upravitelj kako mi mogao interpretirati primljene vrijednosti.

Redoslijed kojim ECU treba spremiti bajtove definiran je u tablicama opisa objekta (engl. *object description tables* - ODT). Adresa i duljina objekta su važni kako bi se svaka

vrijednost koja se mjeri mogla jedinstveno identificirati. Tablica opisa objekata pruža na uvid raspodjelu RAM sadržaja iz ECU-a.

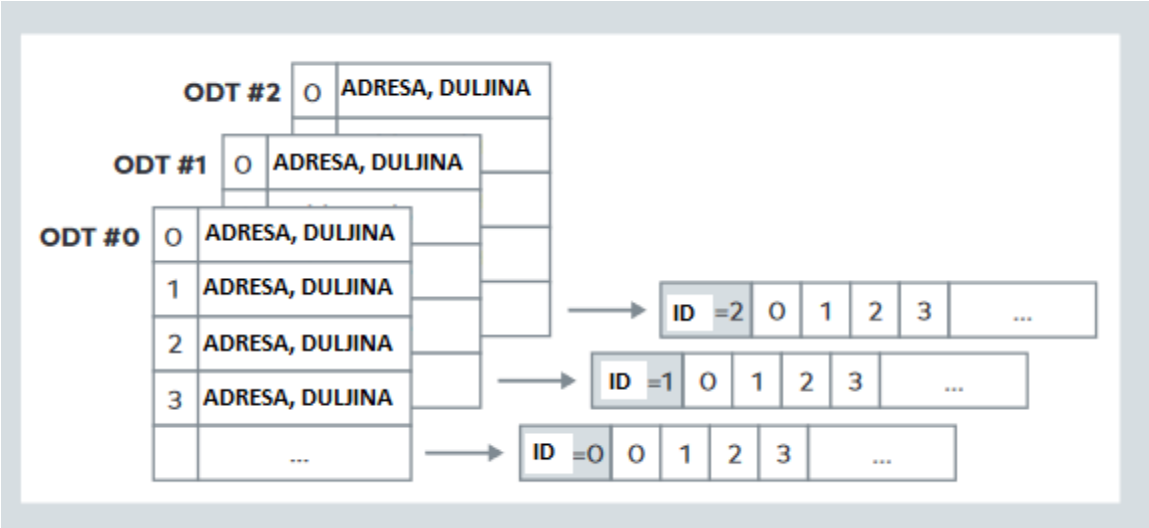


Sl. 4.15. *Prikaz lokacije RAM memorije za* ODT *liste unutar* ECU*-a [3].*

Unos u ODT listi referencira memorijski blok u RAM memoriji ECU-a pomoću adrese i duljine objekta kao što je i vidljivo na slici 4.15. Kada se u nekom trenutku izvrši događaj za koji su vezani zapisi unutar ODT liste, ECU započinje s prikupljanjem svih vrijednosti unutar te ODT liste, prikupljanje vrijednosti formira u pakete i šalje ih na sabirnicu. Svaki paket ima maksimalni broj bajtova koji se može iskoristiti što ovisi o protokolu koji se koristi za prijenos poruka. U ovom slučaju koristi se CAN protokol pa je tako maksimalni broj bajtova sedam. Ako se šalje više vrijednosti, pa zapis ne stane unutar jednog ODT zapisa, ECU će koristiti više ODT zapisa kako bi mogao spremiti i poslati

željene vrijednosti. Prilikom slanja više ODT zapisa ECU mora pratiti određena pravila za formiranje poruke i slanje iste kako bi i upravitelj mogao identificirati o kojem točno ODT zapisu se radi.

U XCP protokolu ODT liste se kombiniraju i spremaju u DAQ liste, tj. svaka DAQ lista se veže uz neki događaj i sadrži određen broj ODT lista, a svaka ODT lista ima reference na memorijske adrese unutar ECU-a. Na primjer, ako korisnik želi koristiti dva intervala prilikom mjerenja, svaki interval predstavlja poseban događaj pa je zato potrebno koristiti dvije DAQ liste jer se jedna DAQ lista može vezati samo uz jedan događaj. Slika 4.16 prikazuje primjer DAQ liste koja sadrži 3 ODT liste.



Sl. 4.16. *Prikaz izgleda* DAQ *liste sa 3* ODT *liste [3].*

4.8. Vrste DAQ lista

Postoji tri vrste DAQ lista i to su: statičke DAQ liste, predefinirane DAQ liste i dinamičke DAQ liste.

Ako se radi o DAQ i ODT listama koje su trajno definirane u ECU-u onda se radi o statičkim DAQ listama. Statičke DAQ liste mogu se promatrati kao okvir koji je potrebno popuniti ali njegove dimenzije nije moguće mijenjati. Dimenzije statičkih DAQ lista su postavljenje unutar ECU-a i opisane su u A2L datoteci.

Predefinirane DAQ liste gotovo se nikada ne koriste zbog nedostatka fleksibilnosti jer korištenje predefiniranih DAQ lista podrazumijeva unaprijed poznate vrijednosti, tj. parametre koje se želi mjeriti. Predefinirane DAQ liste imaju primjenu kod analognih sustava

jer fleksibilnost tu nije potrebna zato što fizička struktura mjernog sustava ostaje ista tijekom cijelog životnog ciklusa sustava.

Dinamičke DAQ liste su poseban aspekt XCP protokola jer su vrlo fleksibilne i ostavljaju korisniku puno prostora za organiziranje sustava mjerenja na način na koji želi. Kod dinamičkih DAQ lista, njihov broj, kao i broj ODT lista, nije unaprijed određen već korisnik može odrediti koliko DAQ i ODT lista je potrebno alocirati. Prednost ovog tipa DAQ liste je u tome što alat za mjerenje ima veće mogućnosti prilikom sastavljanja DAQ lista i upravljanja strukturom istih. Dinamičke DAQ liste su korištene u ovom diplomskom radu.

5. AURIX PLATFORMA

Za implementaciju XCP protokola preko CAN protokola u ovom diplomskom radu korištena je ploča s Infineon AURIX tc27xT[2] mikroupravljačem koja je prikazana na slici 5.1. Tc27xT mikroupravljač ima 32-bitni procesor s tri jezgre. Radni takt procesora je 200MHz, potrebno napajanje je napona 3.3V, te sadrži moćni generički vremenski modul (engl. *generic timer module*). Glavne karakteristike ovog mikroupravljača su smanjena kompleksnost, energetska učinkovitost i manja cijena. Dodatne karakteristike tc27xT mikroupravljača:

- TriCore™ trojezgreni procesor radnog takta 200MHz s DSP funkcionalnostima
- 4MB flash memorije s ECC (engl. *error-correcting code*) zaštitom
- 348KB EEProm (engl. e*lectrically erasable programmable read-only memory*) memorije
- 472KB RAM memorije s ECC zaštitom
- 64x DMA (engl. *direct memory access*) kanali
- Povezivost preko Ethernet, FlexRay, CAN, CAN FD, LIN i SPI protokola
- Izmjenjiv HSM (engl. *hardware security module*)
- Radne temperature od -40 do 150 Celzijevih stupnjeva

Sl. 5.1. *Aurix platforma.*

6. VECTOR I LAUTERBACH OPREMA

Prilikom izrade ovog diplomskog rada bila je potrebna oprema za uklanjanje pogrešaka prilikom razvoja softverskog rješenja i oprema za spajanje razvojne ploče na CAN sabirnicu.

Za uklanjanje pogrešaka korišten je *lauterbach* uređaj zajedno sa *Trace32* programom. *Lauterbach* uređaj spaja se pomoću JTAG kabla na razvojnu ploču, a pomoću *Trace32* programa moguće je vidjeti spojeni uređaj, trenutno stanje uređaja, programski kod koji se trenutno izvršava, popis varijabli i funkcija i slično. *Lauterbach* uređaj bio je vrlo koristan jer je olakšao pronalazak i uklanjanje pogrešaka promjenom varijabli tijekom izvođena programa, zaustavljanje izvođena programa u bilo kojem trenutku i slično. Slika 6.1 prikazuje izgled *lauterbach* uređaja za uklanjanje pogrešaka.

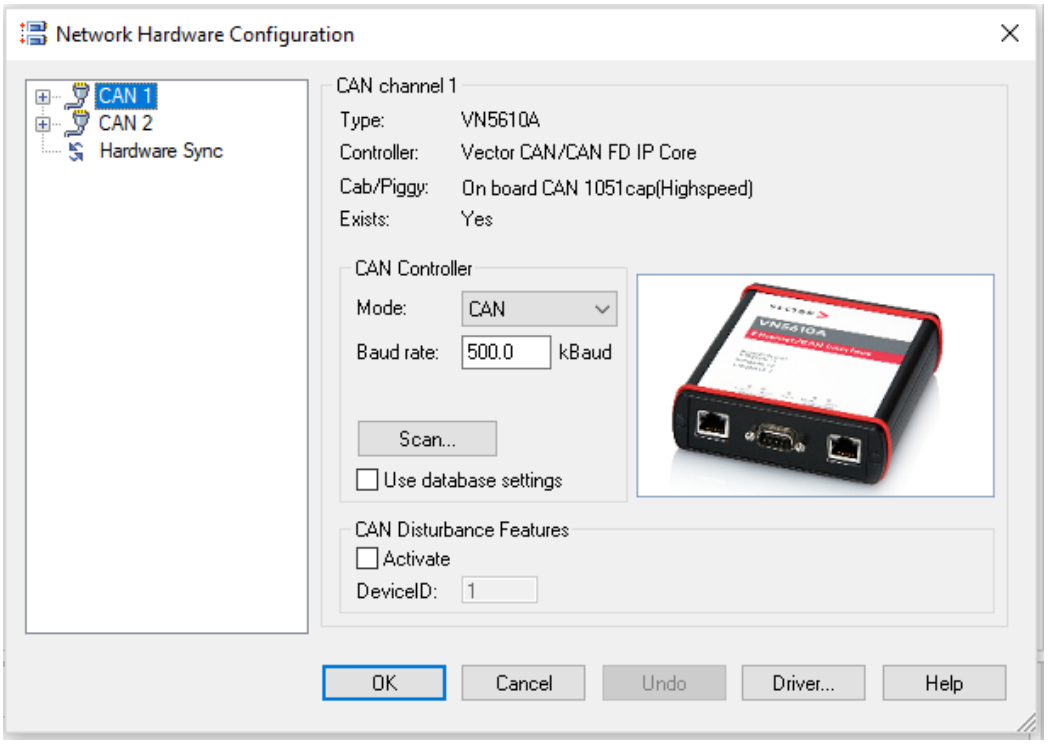
Sl. 6.1. *Lauterbach uređaj za uklananje pogrešaka.*

Za spajanje razvojne ploče na CAN sabirnicu korišteno je *Vector VN5610A* mrežno sučelje. *VN5610A* sučelje povezuje se USB priključkom na računalo, a omogućuje pristup Ethernet i CAN sabirnici. Postavke *VN5610A* uređaja moguće je mijenjati u CANoe programu, kao što je i vidljivo na slici 6.2. Izgled *VN5610A* uređaja vidljiv je na slici 6.3.

Karakteristike *VN5610A* mrežnog sučelja:

- Dva neovisna Ethernet kanala za IEEE standarde
- Dva CAN kanala visoke brzine (omogućuju CAN FD)
- Ethernet nadzor između dva čvora

- Dodatno IO sučelje za podešavanje (npr. DoIP aktivacijska linija) ili uzorkovanje digitalnih vrijednosti
- Spajanje s računalom putem USB 2.0 i / ili USB 3.0 sučelja
- Napajanje s USB sučelja
- Sinkronizacija između više uređaja i drugih sustava sabirnica (CAN, LIN, FlexRay, MOST)



Sl. 6.2. *Postavke VN5610A uređaja u CANoe programu.*

Sl. 6.3. *Vector VN5610A mrežno sučelje.*

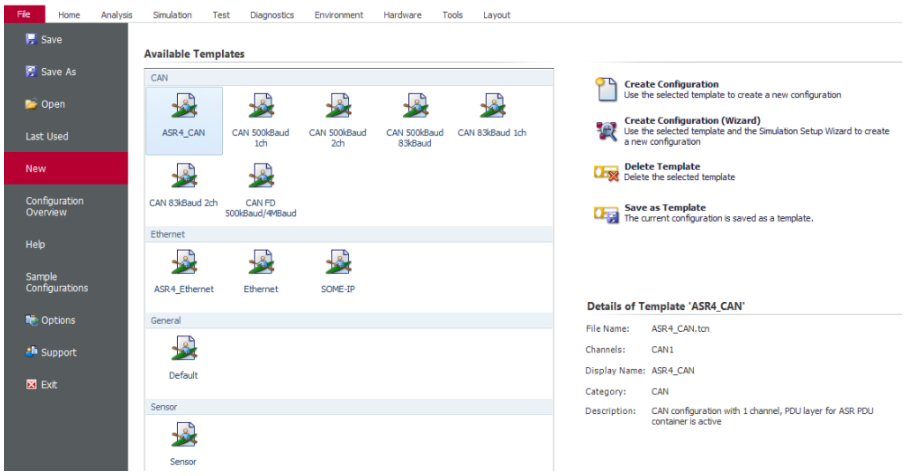
7. CANoe PROGRAM

CANoe je programski alat za mjerenja, razvoj i testiranja kojeg je razvila tvrtka *Vector Informatik GmbH*. CANoe alat se najviše koristi u automobilskoj industriji, tj. koriste ga proizvođači automobila i razvojni inženjeri elektroničkih kontrolnih jedinica. Glavna primjena CANoe alata je za razvoj, analizu, testiranje, simulaciju i dijagnostiku ECU uređaja. CANoe podržava korištenje CAN, LIN, *FlexRay* i *Ethernet* protokola. U ovom diplomskom radu CANoe alat je korišten za primanje i slanje CAN poruka i za interpretaciju primljenih podataka.

CANoe program se u ovom diplomskom radu koristio slanje i primanje CAN poruka, te za validaciju istih.

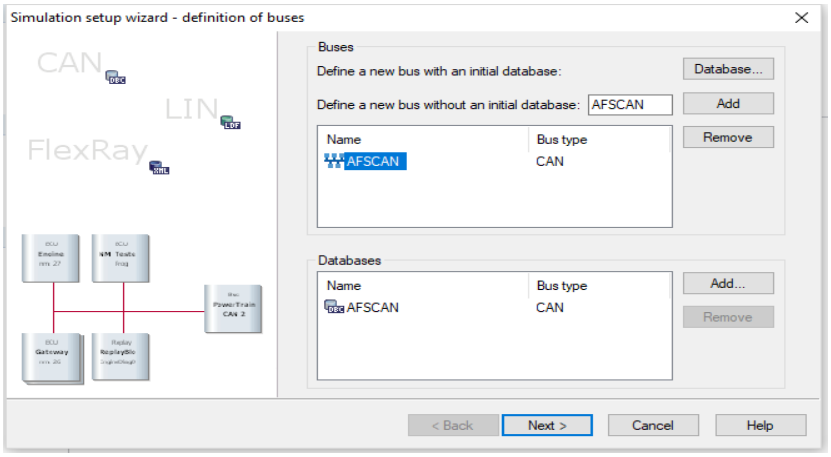
7.1. Korištenje CANoe programa

Prilikom pokretanja CANoe programa potrebno je odabrati *File* i zatim *CAN 500kBaud 2ch* predložak s popisa CAN predložaka kao što je i vidljivo na slici 7.1. Odabirom tog predloška pokreće se postupak postavljanja okruženja gdje je moguće slati CAN poruke na CAN1 ili CAN2 kanalu brzinom 500kBaud-a.



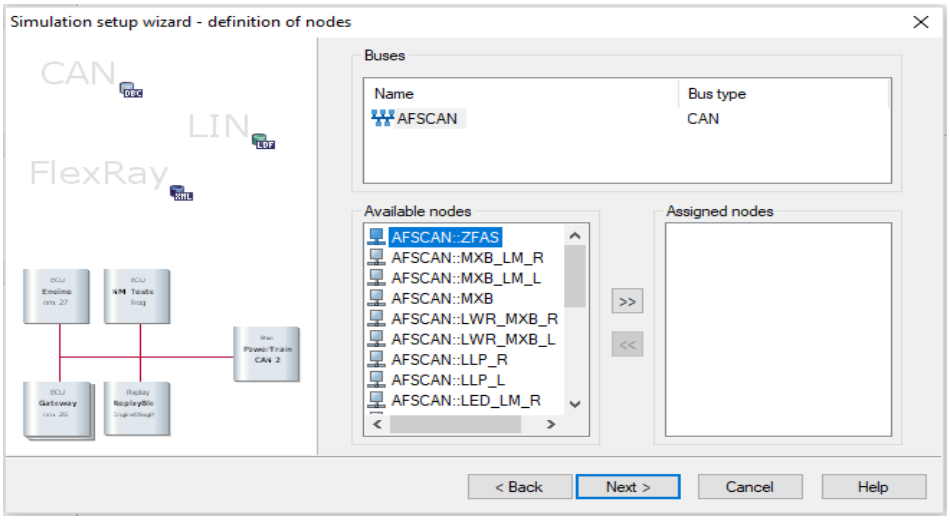
Sl. 7.1. *Prikaz odabira CAN predloška u CANoe programu.*

Nakon odabira predloška pojavljuje se prozor prikazan na slici 7.2 gdje je potrebno odabrati bazu podataka s CAN porukama koje će biti moguće slati.



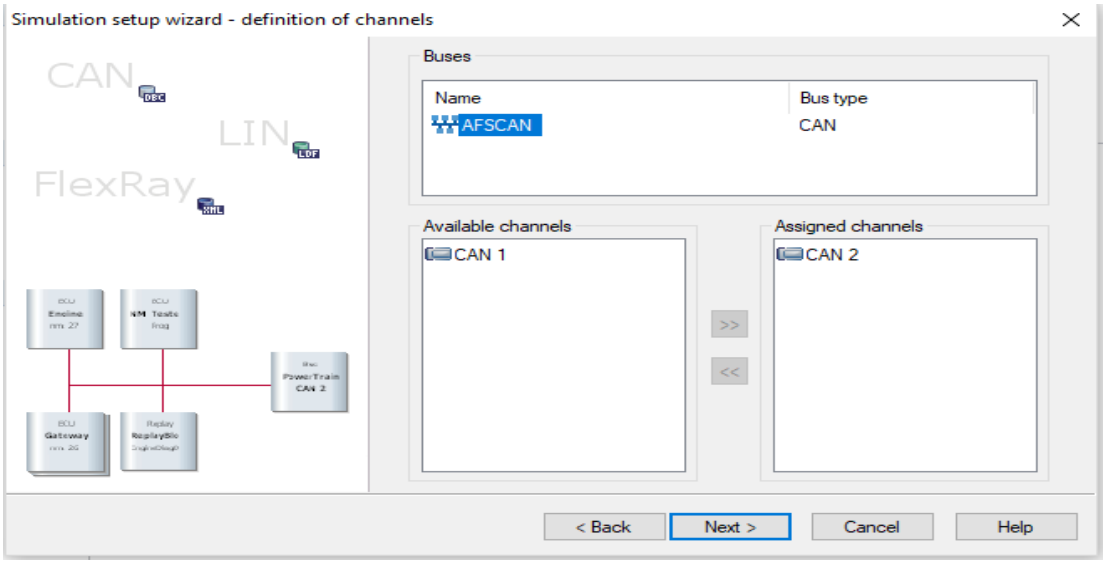
Sl. 7.2. *Prikaz odabira baze podataka s CAN porukama u CANoe programu.*

Nakon odabira baze podataka s CAN porukama potrebno je odabrati i broj čvorova koji će biti spojeni na CAN sabirnicu, u ovom slučaju radi se o samo jednom čvoru jer je na CAN sabirnicu spojen samo jedan ECU. Odabir čvorova vidljiv je na slici 7.3.

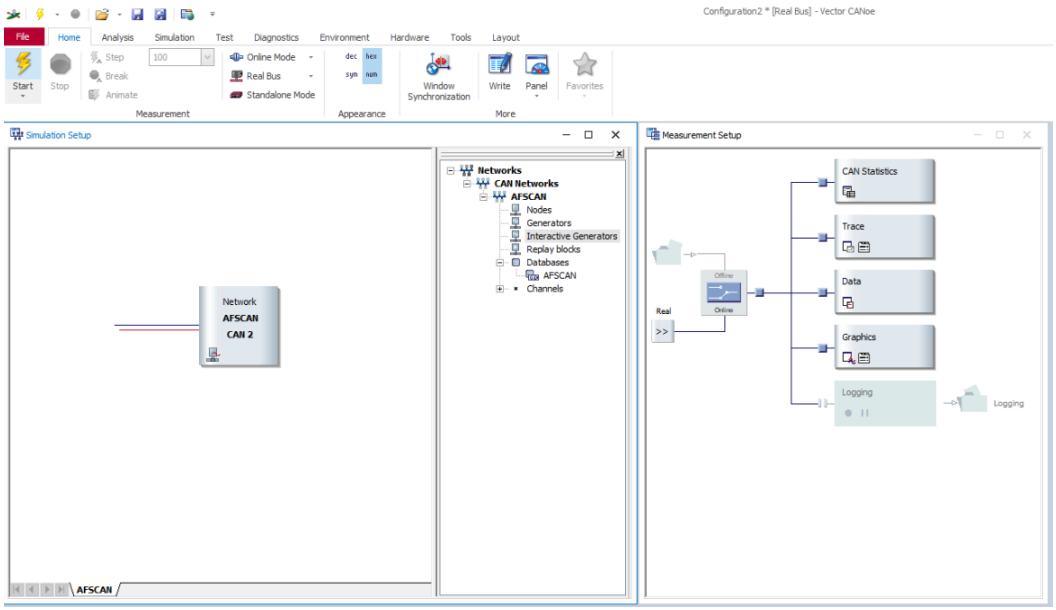


Sl. 7.3. *Prikaz odabira baze podataka s CAN porukama u CANoe programu.*

Nakon odabira broja čvorova potrebno je odabrati koja CAN sabirnica će se koristiti, a u ovom slučaju moguće je koristiti CAN1 i CAN2 sabirnicu. Odabir sabirnice vidljiv je na slici 7.4. Poslije odabira sabirnice otvara se glavno sučelje CANoe programa koje je prikazano na slici 7.5. U glavnom sučelju, s lijeve strane prikazani su čvorovi koji su spojeni na CAN sabirnicu, a s desne strane vidljive su postavke mjerenja.

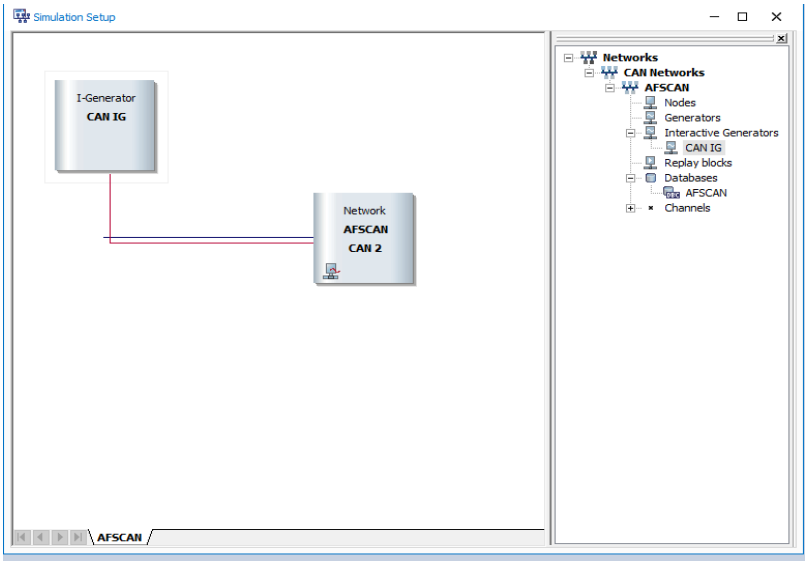


Sl. 7.4. *Prikaz odabira CAN sabirnice u CANoe programu.*



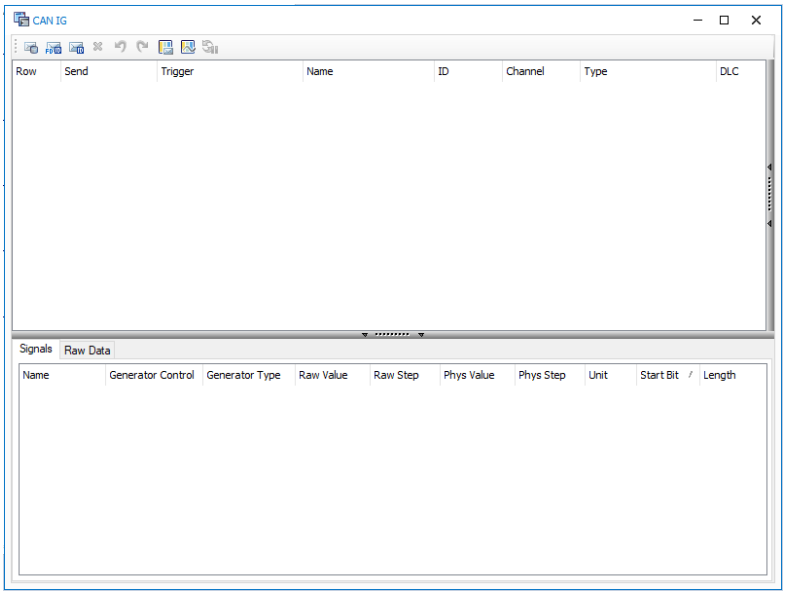
Sl. 7.5. *Prikaz glavnog sučelja CANoe programa.*

Kako bi se mogle slati poruke iz CANoe programa potrebno je dodati interaktivni generator CAN poruka, a taj generator se dodaje dvoklikom na *Interactive Generators* opciju vidljivu na slici 7.5. Nakon dvoklika na *Interactive Generators* potrebno je odabrati *CAN generator*. Dodani CAN generator vidljiv je na slici 7.6.



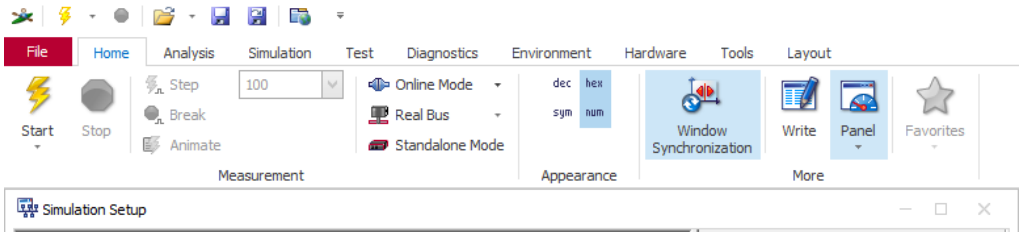
Sl. 7.6. *Prikaz CAN interaktivnog generatora.*

Kako bi se dodale poruke, na popis poruka koje je moguće poslati, potrebno je dvokliknuti na CAN generator nakon čega se otvara sučelje u kojem je moguće dodati CAN poruke, izmjenjivati njihov sadržaj i sl. Slika 7.7 prikazuje izgled sučelja generatora CAN poruka.



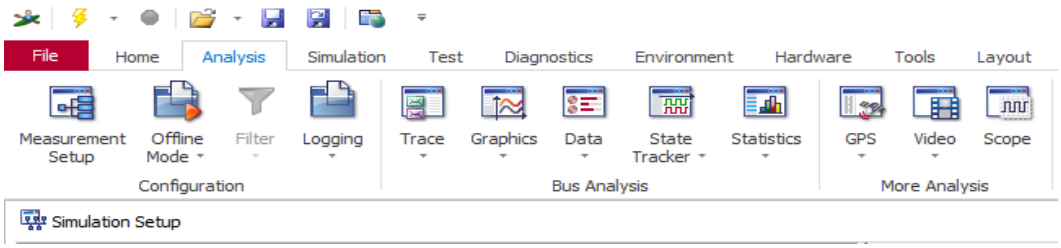
Sl. 7.7. *Prikaz sučelja CAN interaktivnog generatora.*

Za pokretanje mjerenja potrebno je kliknuti na ikonu munje u lijevom gornjem kutu, kao što je i vidljivo na slici 7.8.



Sl. 7.8. *Prikaz ikone za pokretanje mjerenja.*

Kako bi bilo moguće pratiti promet podataka na CAN sabirnici poželjno je upaliti *trace* prozor u kojem se vide svi podatci koje se izmjenjuju na CAN sabirnici. *Trace* prozor može otvoriti odabirom *Analysis* izbornika, te odabirom *Trace* opcije, kao što je i prikazano na slici 7.9.



Sl. 7.9. *Prikaz odabira trace prozora.*

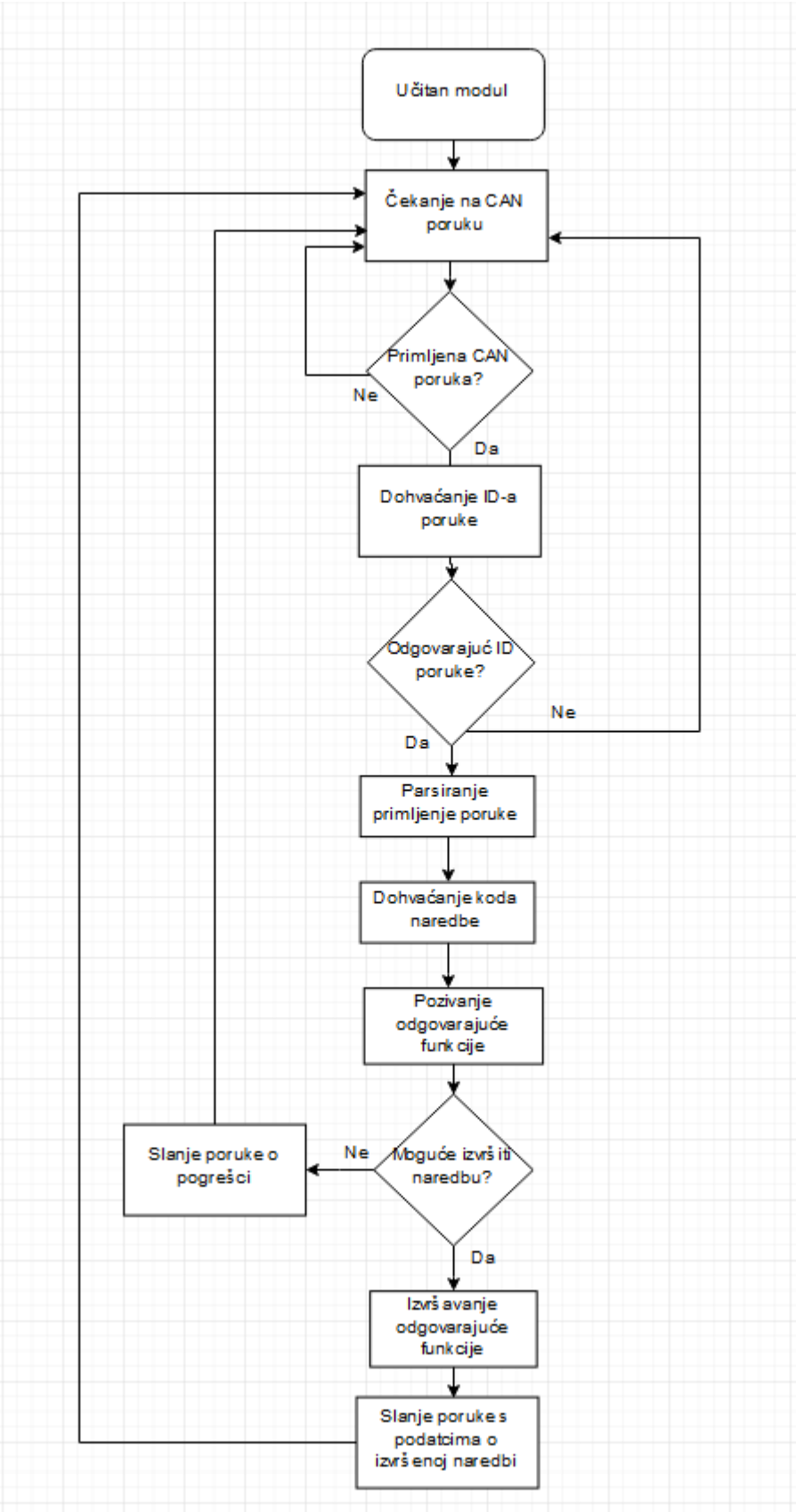
8. OPIS IMPLEMENTIRANOG RJEŠENJA

Ovaj diplomski rad je zamišljen i implementiran kao modul koji se može uključiti u postojeći projekt dodavanjem *Xcp.c* i *Xcp.h* datoteka. U *Xcp.h* datoteci se nalaze se definicije svih kodova naredbi koji se mogu koristiti, te deklaracije funkcija koje se koriste za primanje poruka. Izgled *Xcp.h* datoteke vidljiv je na slici 8.1. U *Xcp.c* datoteci nalaze se deklaracije i definicije funkcija koje se koriste pri samom radu algoritma, kao i definicije potrebnih konstanti, struktura itd. Modul funkcionira tako da prima svaku poruku koja se šalje putem CAN protokola, te na temelju identifikacijskog broja određuje da li je primljenu poruku potrebno obrađivati i na nju odgovarati ili ne. Ako je primljena poruka važna, poruka se parsira, te se dohvaća kod naredbe iz primljene poruke i na temelju primljenog koda se pozivaju odgovarajuće funkcije. Za svaku naredbu postoji funkcija koja stvara odgovor na tu naredbu dohvaćanjem potrebnih informacija, pozivanjem za to vezanih funkcija i sl. Dijagram toka implementiranog modula vidljiv je na slici 8.2.

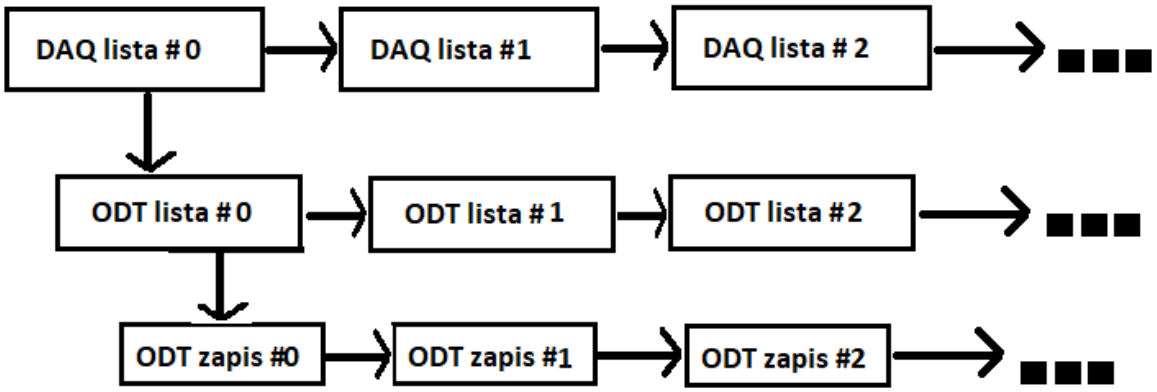
Dok je za većinu naredbi potrebno zapisati ili dohvatiti određene vrijednosti iz struktura ili memorijskih adresa, primijeniti bit maske na određene vrijednosti i sl., kompleksniji dio modula je onaj koji se bavi upravljanjem DAQ listama. DAQ liste, zajedno sa ODT listama i ODT zapisima, implementirane su korištenje povezanih popisa. Za svaku DAQ listu alocira se jedan element povezanog popisa koji ima informaciju o broju ODT lista koji sadrži i pokazivač na iduću DAQ listu ako postoji. Svaka ODT lista ima informaciju o broju ODT zapisa koji sadrži, pokazivač na prvi ODT zapis koji ta ODT lista sadrži i pokazivač na iduću ODT listu koja se nalazi u danoj DAQ listi. Svaki ODT zapis koji se nalazi unutar neke ODT liste, a time i unutar neke DAQ liste, sadrži pokazivač na memorijsku adresu na kojoj se nalazi vrijednost koju se želi pratiti, sadrži adresnu ekstenziju te adrese kao i veličinu podatka koji je tu zapisan, te na posljetku sadrži i pokazivač na idući ODT zapis, ako postoji. Struktura ovog rješenja može se predočiti kao drvo gdje se krajnjim informacijama pristupa pretraživanjem drveta po dubini, a svim ovim informacijama moguće je pristupiti pomoću petlji i samo jedne globalne varijable koja je pokazivač na prvu DAQ listu. Slika 8.3 prikazuje strukturu implementirano rješenja DAQ lista.

```
#ifndef OS_XCP_H_
#define OS_XCP_H_

#include <Multican/Std/IfxMultican.h>

#define XCP_VERSION_PROTOCOL_LAYER  (0x0001)

/* XCP transport layer version number (16-bit) */
#define XCP_VERSION_TRANSPORT_LAYER (0x0001)

/* XCP packet identifiers */
#define XCP_PID_RES                 (0xff)          /* command response packet        */
#define XCP_PID_ERR                 (0xfe)          /* error packet                   */

/* XCP error codes */
#define XCP_ERR_CMD_SYNCH           (0x00)          /* cmd processor synchronization */
#define XCP_ERR_CMD_BUSY            (0x10)          /* command was not executed       */
#define XCP_ERR_CMD_UNKNOWN         (0x20)          /* unknown or unsupported command*/
#define XCP_ERR_OUT_OF_RANGE        (0x22)          /* parameter out of range         */
#define XCP_ERR_ACCESS_LOCKED       (0x25)          /* protected. seed/key required  */
#define XCP_ERR_PAGE_NOT_VALID      (0x26)          /* cal page not valid             */
#define XCP_ERR_SEQUENCE            (0x29)          /* sequence error                 */
#define XCP_ERR_GENERIC             (0x31)          /* generic error                  */
#define XCP_ERR_MEMORY_OVERFLOW     (0x30)

/* XCP command codes */
#define XCP_CMD_CONNECT             (0xff)          /* CONNECT command code           */
#define XCP_CMD_DISCONNECT          (0xfe)          /* DISCONNECT command code        */
#define XCP_CMD_GET_STATUS          (0xfd)          /* GET_STATUS command code        */
#define XCP_CMD_SYNCH               (0xfc)          /* SYNCH command code             */
#define XCP_CMD_GET_ID              (0xf2)          /* GET_ID command code            */
#define XCP_CMD_GET_SEED            (0xf8)          /* GET_SEED command code          */
#define XCP_CMD_UNLOCK              (0xf7)          /* UNLOCK command code            */
#define XCP_CMD_SET_MTA             (0xf6)          /* SET_MTA command code           */
#define XCP_CMD_UPLOAD              (0xf5)          /* UPLOAD command code            */
#define XCP_CMD_SHORT_UPLOAD        (0xf4)          /* SHORT_UPLOAD command code      */
#define XCP_CMD_BUILD_CHECKSUM      (0xf3)          /* BUILD_CHECKSUM command code    */
#define XCP_CMD_DOWNLOAD            (0xf0)          /* DOWNLOAD command code          */
#define XCP_CMD_DOWLOAD_MAX         (0xee)          /* DOWNLOAD_MAX command code      */
#define XCP_CMD_SET_CAL_PAGE        (0xeb)          /* SET_CALPAGE command code       */
#define XCP_CMD_GET_CAL_PAGE        (0xea)          /* GET_CALPAGE command code       */
#define XCP_CMD_PROGRAM_START       (0xd2)          /* PROGRAM_START command code     */
#define XCP_CMD_PROGRAM_CLEAR       (0xd1)          /* PROGRAM_CLEAR command code     */
#define XCP_CMD_PROGRAM             (0xd0)          /* PROGRAM command code           */
#define XCP_CMD_PROGRAM_RESET       (0xcf)          /* PROGRAM_RESET command code     */
#define XCP_CMD_PROGRAM_PREPARE     (0xcc)          /* PROGRAM_PREPARE command code   */
#define XCP_CMD_PROGRAM_MAX         (0xc9)          /* PROGRAM_MAX command code       */

#define XCP_CMD_SET_DAQ_PTR         (0xe2)
#define XCP_CMD_WRITE_DAQ           (0xe1)
#define XCP_CMD_SET_DAQ_LIST_MODE   (0xe0)
#define XCP_CMD_FREE_DAQ            (0xd6)
#define XCP_CMD_ALLOC_DAQ           (0xd5)
#define XCP_CMD_ALLOC_ODT           (0xd4)
#define XCP_CMD_ALLOC_ODT_ENTRY     (0xd3)
#define XCP_CMD_START_STOP_DAQ_LIST (0xde)
#define XCP_CMD_START_STOP_SYNCH    (0xdd)

#endif /* OS_XCP_H_ */
```

Sl. 8.1 *Izgled Xcp.h datoteke*



Sl. 8.2 *Dijagram toka implementiranog rješenja*



Sl. 8.2 *Struktura* DAQ *lista*

9. IMPLEMENTIRANE XCP FUNKCIJE

Kao što je i ranije navedeno, jedna od najvećih prednosti *XCP* protokola leži u njegovoj fleksibilnosti, tj. postoji više grupa funkcija koje nije potrebno implementirati ako za njima nema potrebe. Ako je neka grupa funkcija potrebna postoji nekoliko funkcija koje je obavezno implementirati dok se implementiranje ostalih funkcija ostavlja na izbor razvojnom programeru.

Iz grupe standardnih funkcija implementirane su funkcije za: spajanje (engl. *connect*), odspajanje (engl. *disconnect*), dobavljanje statusa (engl. *get status*), dobavljanje identifikacije (engl. *get id*) i sinkroniziranje (engl. *synch*).

Iz grupe funkcija za kalibraciju implementirane su funkcije za: skidanje podataka (engl. *download*), prijenos podataka (engl. *short upload*) i naredba za postavljanje pokazivača (engl. *set mta*).

Iz grupe funkcija za mjerenja implementirane su funkcije koje se odnose na korištenje dinamičkih DAQ lista, a to su funkcije za: postavljanje pokazivača na element liste (engl. *set daq ptr*), upisivanje podataka unutar nekog elementa DAQ liste (engl. *write daq*), pokretanje ili zaustavljanje DAQ liste (engl. *start stop daq list*), sinkronizirano pokretanje ili zaustavljanje DAQ lista (engl. *start stop synch*), alociranje DAQ liste (engl. *alloc daq*), alociranje ODT liste (engl. *alloc odt*) i alociranje pojedinog ODT elementa (engl. *alloc odt entry*).

Tablica 9.1 *Legenda tipova podataka*

NAZIV TIPA PODATKA	ZNAČENJE
BYTE	8 bitni cjelobrojni podatak bez predznaka
WORD	16 bitni cjelobrojni podatak bez predznaka
DWORD	32 bitni cjelobrojni podatak bez predznaka
DLONG	64 bitni cjelobrojni podatak bez predznaka

Prilikom implementiranja funkcija gotovo uvijek je bilo potrebno raditi s varijablama različitih veličina, a nazivi tipova podataka različitih veličina opisani su u tablici 9.1.

9.1 Naredba za spajanje

Pomoću ove naredbe [4] upravitelj postavlja vezu s podređenim uređajem, tj. ECU-om. Nakon poslane ove naredbe ECU prelazi u spojeno stanjem (engl. *connected state*) i prije toga ne odgovara na niti jednu naredbu sve dok nije u spojenom stanju. Uz slanje koda za spajanje, upravitelj šalje i dodatni parametar koji govori podređenom uređaju da li je u normalnom načinu rada, ili posebnom korisnički definiranom načinu rada (engl. *user defined*). ECU odgovor na naredbu za spajanje formira dohvaćanjem različitih informacija, a izgled poruke koja se šalje kao odgovor prikazan je na tablici 9.2.

Tablica 9.2 *Izgled podatka koji podređeni uređaj šalje kao odgovor*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Id paketa: 0xFF
1	*BYTE*	*Resource*
2	*BYTE*	*Comm_mode_basic*
3	*BYTE*	*Max_CTO*
4	*BYTE*	*Max_DTO*
6	*BYTE*	*XCP protocol layer vesion number*
7	*BYTE*	*XCP transport layer vesion number*

Neki od parametara koji se šalju formiraju se na specifičan način, tj. formiraju se dohvaćanjem vrijednosti bitova iz više varijabli te se zajedno spajaju u bajtove. Parametri koji se formiraju za specifičan način su *resource* i *comm_mode_basic* parametri. *Resource* parametar se formira na način opisan u tablici 9.3. gdje na mjestu bita koji je označen slovom X nije važno koja je vrijednost zapisana, a *PGM, STIM,* DAQ i *CAL/PAG* vrijednosti se određuju na način opisan u tablici 9.4. *Comm_mode_basic* parametar se formira na način opisan u tablici 9.5, a vrijednost pojedinih bitova *Comm_mode_basic* parametra određuju se na način opisan u tablici 9.6.

Tablica 9.3 *Bit maska Resource parametra*

X	X	X	*PGM*	*STIM*	DAQ	X	*CAL/PAG*

Tablica 9.4 *Način formiranja pojedinih bitova Resource parametra*

ZASTAVICA	**OPIS**
CAL/PAG	Kalibracija i straničenje: 0: nedostupno 1: dostupno
DAQ	Dostupnost DAQ liste: 0: nedostupno 1: dostupno
STIM	Stimulacija podataka: 0: nedostupno 1: dostupno
PGM	Programiranje *flash* memorije: 0: nedostupno 1:dostupno

Tablica 9.5 *Bit maska Comm_mode_basic parametra*

Neobavezno	*Slave_block_mode*	X	X	X	*Address_ granularity1*	*Address_ Granularity0*	*Byte_order*

Tablica 9.6 *Način formiranja pojedinih bitova Comm_mode_basic parametra*

ZASTAVICA	**OPIS**
Byte_order	Redoslijed bitova: 0: Intel format 1: Motorola format (najznačajniji bit na najnižoj adresi)
Address_granularity	[*address_granularity1:address_granularity0*]: 0:0 -> *BYTE* 0:1 -> *WORD* 1:0 -> *DWORD* 1:1 Rezervirano
Slave_block_mode	Blokovski način slanja podataka dostupan:

	0: nije 1: je

Max_CTO parametar označava maksimalnu veličinu *CTO* paketa u bajtovima, a *MAX_DTO* parametar označava maksimalnu veličinu *DTO* paketa u bajtovima. *XCP protocol layer version number* daje informaciju o glavnoj inačici protokola, a *XCP transport layer version number* označava glavnu inačicu trenutnog transportnog sloja.

9.2 Naredba za odspajanje

Naredba za odspajanje[4] koristi se kako bi se ECU-u javilo da upravitelj više neće slati naredbe u trenutnoj sesiji. Ako trenutno nije moguće odspajanje ECU će poslati *ERR_CMD_BUSY* pogrešku. Tablica 9.7 prikazuje izgled naredbe koja se šalje ECU-u kako bi se izvršilo odspajanje.

Tablica 9.7 *Prikaz naredbe za odspajanje*

POZICIJA	**TIP PODATKA**	**OPIS**
0	*BYTE*	Kod naredbe: 0xFE

9.3 Naredba za dohvaćanje statusa

Ova naredba[4] koristi se za dohvaćanje informacija o trenutnom statusu podređenog uređaja. To uključuje status zaštite resursa, naredbe koje tek čekaju da budu izvršene, te opće informacije o prikupljanju i stimulaciji podataka. Tablica 9.8 prikazuje izgled naredbe za dohvaćanje statusa podređenog uređaja koju prima ECU. Izgled poruke koju ECU šalje kao odgovor na naredbu za dohvaćanje statusa prikazan je na tablici 9.9. Parametar koji se nalazi na poziciji prvog bajta, tj. trenutni status sesije, određuje se na način prikazan na tablici 9.10 gdje se vrijednosti *DAQ_RUNNING, CLEAR_DAQ_REQ, STORE_DAQ_REQ* i *STORE_CAL_REQ* bitova određuju na način prikazan na tablici 9.11. Parametar statusa zaštite resursa određuje se na način prikazan na tablici 9.12, a vrijednosti *PGM, STIM,* DAQ i *CAL/PAG* bitova određuju se određuju se na način prikazan na tablici 9.13.

Tablica 9.8 *Prikaz naredbe za dohvaćanje statusa podređenog uređaja*

POZICIJA	**TIP PODATKA**	**OPIS**
0	*BYTE*	Kod naredbe: 0xFD

Tablica 9.9 *Izgled podatka koji podređeni uređaj šalje kao odgovor*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Id paketa: 0xFF
1	*BYTE*	Trenutni status sesije
2	*BYTE*	Trenutni status zaštite resursa
3	*BYTE*	Rezervirano
4	*WORD*	Identifikacijski broj sesije

Tablica 9.10 *Bit maska parametra trenutnog statusa sesije*

Resume	*DAQ_RUNNING*	X	X	*CLEAR_DAQ_REQ*	*STORE_DAQ_REQ*	X	*STORE_CAL_REQ*

Tablica 9.11 *Način formiranja pojedinih bitova parametra trenutnog statusa sesije*

ZASTAVICA	**OPIS**
STORE_CAL_REQ	Zahtjev za spremanjem podataka za kalibraciju poslan: 0: ne 1: da
STORE_DAQ_REQ	Zahtjev za spremanjem DAQ lista poslan: 0: ne 1: da
CLEAR_DAQ_REQ	Zahtjev za čišćenjem DAQ lista poslan: 0: ne 1: da
DAQ_RUNNING	Prijenos podataka korištenjem DAQ lista je u tijeku: 0: ne 1: da
RESUME	Podređeni uređaj je u načinu za nastavljanje rada: 0: ne 1: da

STORE_CAL_REQ zastavica označava da je zahtjev za spremanje kalibracijskih podataka na čekanju. Čim ECU ispuni zahtjev *STORE_CAL_REQ* zastavica će biti postavljena na odgovarajuću vrijednost.

STORE_DAQ_REQ zastavica označava da je zahtjev za spremanje podataka o konfiguraciji DAQ lista na čekanju. Čim ECU ispuni zahtjev *STORE_* DAQ *_REQ* zastavica će biti postavljena na odgovarajuću vrijednost.

CLEAR_DAQ_REQ zastavica označava da je zahtjev za čišćenjem svih DAQ lista na čekanju. Sve vrijednosti svakog ODT zapisa se resetiraju, tj. veličina i ekstenzija se postavljaju na 0, a pomak bita se postavlja na vrijednost 0xFF.

DAQ_RUNNING zastavica označava da je barem jedna DAQ lista aktivna, tj. da se podatci iz barem jedne DAQ liste trenutno šalju.

Tablica 9.12 *Bit maska parametra trenutnog statusa zaštite resursa*

X	X	X	*PGM*	*STIM*	*DAQ*	X	*CAL/PAG*

Tablica 9.13 *Način formiranja pojedinih bitova parametra trenutnog statusa zaštite resursa*

ZASTAVICA	**OPIS**
CAL/PAG	Postavke naredbi straničenja i kalibracije: 0 = naredbe nisu zaštićene *SEED&KEY* mehanizmom 1 = naredbe su zaštićene *SEED&KEY* mehanizmom
DAQ	Postavke DAQ naredbi (slanje podataka u smijeru od podređenog uređaja prema upravitelju): 0 = naredbe nisu zaštićene *SEED&KEY* mehanizmom 1 = naredbe su zaštićene *SEED&KEY* mehanizmom
STIM	Postavke DAQ naredbi (slanje podataka u smijeru od upravitelja prema podređenom

	uređaju) : 0 = naredbe nisu zaštićene *SEED&KEY* mehanizmom 1 = naredbe su zaštićene *SEED&KEY* mehanizmom
PGM	Postavke narebi za programiranje: 0 = naredbe nisu zaštićene *SEED&KEY* mehanizmom 1 = naredbe su zaštićene *SEED&KEY* mehanizmom

9.4 Naredba za sinkronizaciju

Ova naredba[4] se koristi za sinkroniziranje prilikom izvršavanja naredbi nakon isteka određenih vremenskih perioda. *SYNCH* naredba će uvijek imati negativan odgovor koji koristi *ERR_CMD_SYNCH* kod pogreške. Niti jedna druga naredba ne koristi ovaj kod pa je zato jednostavno razlikovati odgovor podređenog uređaja na *SYNCH* naredbu od odgovora na bilo koju drugu naredbu. Tablica 9.14 prikazuje izgled naredbe za sinkroniziranje koju prima ECU, a tablica 9.15 prikazuje odgovor koji ECU šalje na naredbu za sinkronizaciju.

Tablica 9.14 *Prikaz naredbe za sinkroniziranje*

POZICIJA	**TIP PODATKA**	**OPIS**
0	*BYTE*	Kod naredbe: 0xFC

Tablica 9.15 *Prikaz odgovora na naredbu za sinkroniziranje*

POZICIJA	**TIP PODATKA**	**OPIS**
0	*BYTE*	Id paketa: 0xFE
1	*BYTE*	Kod pogreške: *ERR_CMD_SYNCH*

9.5 Naredba za dohvaćanje identifikacije podređenog uređaja

Ova naredba[4] je definirana u specifikacijama transportnog sloja zato što se koristi za izvođenje radnji specifičnih za transportni sloj. Tablica 9.16 prikazuje izgled naredbe za dohvaćanje identifikacije podređenog uređaja koja se šalje ECU-u. Tablica 9.17 prikazuje primjer kako naredba za dohvaćanje identifikacije podređenog uređaja može izgledati.

Tablica 9.16 *Prikaz naredbe za dohvaćanje identifikacije podređenog uređaja, tj.* ECU-*a*

POZICIJA	**TIP PODATKA**	**OPIS**
0	*BYTE*	Kod naredbe: 0xF2
1	*BYTE*	Dodatni kod naredbe
2...	*BYTE*	Dodatni parametri

Tablica 9.17 *Primjer naredbe za dohvaćanje identifikacije podređenog uređaja*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Id paketa: 0xF2
1	*BYTE*	Dodatni kod naredbe: 0xFF
2	*BYTE*	0x58 (*ASCII* = X)
3	*BYTE*	0x43 (*ASCII* = C)
4	*BYTE*	0x50 (*ASCII* = P)
5	*BYTE*	Način potvrde ECU-a: 0 = Idenitikacija slanjem istih dodatnih parametara 1 = Identifikacija slanjem invertiranih dodatnih parametara

9.6 Naredba za postavljanje pokazivača na memorijsku adresu

Ova naredba[4] koristi se za inicijaliziranje pokazivača na 32 bitnu memorijsku adresu s mogućim korištenjem 8 bita ekstenzije. Naredba za postavljanje pokazivača koristi se i za sljedeće naredbe: *BUILD_CHECKSUM, UPLOAD, DOWNLOAD, DOWNLOAD_NEXT, DOWNLOAD_MAX, MODIFY_BITS, PROGRAM_CLEAR, PROGRAM, PROGRAM_NEXT* i *PROG*. U ovom diplomskom radu naredba za postavljanje

pokazivača na memorijsku adresu implementirana je zato što je implementirana i naredba za promjenu podataka neke varijable, tj. implementirana je i *DOWNLOAD* naredba. Tablica 9.18 prikazuje izgled naredbe za postavljanje pokazivača na memorijsku adresu koja se šalje ECU-u.

Tablica 9.18 *Prikaz naredbe za postavljanje pokazivača na memorijsku adresu*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Kod naredbe: 0xF6
1	*WORD*	Rezervirano
3	*BYTE*	Adresna ekstenzija
4	*DWORD*	Adresa

9.7 Naredba za pisanje podataka s upravitelja na podređeni uređaj

Naredba za pisanje podataka na podređeni uređaj[4], tj. *DOWNLOAD* naredba može se koristiti tek nakon što se prethodno koristila naredba za postavljanje pokazivača na memorijsku adresu. Naredbom za postavljanje pokazivača odredi se koju varijablu je potrebno izmijeniti, tj. na koju memorijsku adresu ECU-a će biti zapisani novi podatci, a naredbom za slanje podataka izmjenjuju se vrijednosti na željenoj adresi. Tablica 9.19 prikazuje izgled naredbe za pisanje podataka na određenu memorijsku adresu koja se šalje ECU-u.

Tablica 9.19 *Izgled podatka koji podređeni uređaj šalje kao odgovor*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Kod naredbe: 0xF0
1	*BYTE*	n = Broj podataka[*AG*] [1..(*MAX_CTO*-2)/AG] za standarni način razmjene podataka
..		
AG=1:2 *AG*>1:*AG*	*Element 1*	Prvi element podataka

..	..	..
AG=1: n+1 *AG*>1: n**AG*	*Element n*	*n-ti element podataka*

Kao što je i ranije navedeno, a i kao što je prikazano na tablici 9.19, *AG* se odnosi na veličinu podataka, tj. na granularnost memorijske adrese (engl. *address granularity*), a potrebno je razlikovati kada je granularnost 1 ili više od 1 zbog načina postavljanja elemenata podataka, tj. zbog poravnanja bajtova. Prije spremanja primljenih podataka prvo je potrebno provjeriti da li je uopće postavljena adresa na koju je primljene podatke potrebno spremiti.

9.8 Naredba za čitanje podataka s podređenog uređaja

Naredba za čitanje podataka s podređenog uređaja[4], tj. *SHORT_UPLOAD* naredba koristi se kako bi se željeni podatci pročitali s dane memorijske adrese ECU-a. Osim podataka koje je potrebno pročitati u istoj poruci primaju se i podatci o adresi s koje se podatci čitaju. Tablica 9.20 prikazuje izgled naredbe za čitanje podataka s određene memorijske adrese ECU-a.

Tablica 9.20 *Prikaz naredbe za čitanje podataka sa* ECU-*a*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Kod naredbe: 0xF4
1	*BYTE*	Broj podataka koji će se pročitati
2	*BYTE*	Rezervirano
3	*BYTE*	Adresna ekstenzija
4	*DWORD*	Adresa

9.9 Naredba za alokaciju DAQ liste

Ova naredba, tj. *ALLOC*_DAQ naredba[4], koristi za alociranje određenog broja DAQ lista. Prije same alokacije provjerava se ispunjavaju li se određeni uvjeti i ima li dovoljno memorije za alociranje željenog broja listi. Tablica 9.21 prikazuje izgled naredbe za alociranje DAQ lista koja se šalje ECU-u.

Tablica 9.21 *Prikaz naredbe za alociranje* DAQ *liste*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Kod naredbe: 0xD5
1	*BYTE*	Rezervirano
2	*WORD*	Broj DAQ lista za alokaciju

9.10 Naredba za alokaciju ODT liste

Ova naredba, tj. *ALLOC_ODT* naredba[4], koristi se za alociranje određenog broja ODT lista i pridruživanje tih ODT lista nekoj od DAQ lista. Tablica 9.22 prikazuje izgled naredbe za alociranje ODT listi koja se šalje ECU-u.

Tablica 8.22 *Prikaz naredbe za alociranje* ODT *liste*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Kod naredbe: 0xD4
1	*BYTE*	Rezervirano
2	*WORD*	Redni broj DAQ liste
4	*BYTE*	Broj ODT lista za alokaciju

9.11 Naredba za alokaciju ODT zapisa

Ova naredba, tj. *ALLOC_ODT_ENTRY* naredba[4], koristi se za alociranje željenog broja ODT zapisa koji će se dodijeliti željenoj DAQ i ODT listi. Tablica 9.23 prikazuje izgled naredbe za alociranje ODT zapisa koja se šalje ECU-u.

Tablica 9.23 *Prikaz naredbe za alociranje* ODT *zapisa*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Kod naredbe: 0xD3
1	*BYTE*	Rezervirano
2	*WORD*	Redni broj DAQ liste
4	*BYTE*	Redni broj ODT liste
5	*BYTE*	Broj ODT zapisa za alokaciju

9.12 Naredba za postavljanje pokazivača na određeni element DAQ i ODT liste

Ova naredba, tj. *SET_DAQ_PTR* naredba[4], koristi se kako bi se postavio pokazivač na određeni ODT zapis neke DAQ i ODT liste. Ova naredba se koristi kako bi se kasnije mogla pozvati naredba za upisivanje dodatnih podataka u ODT zapis na koji ovaj pokazivač pokazuje. Tablica 9.24 prikazuje izgled naredbe za postavljanje pokazivača na željeni ODT zapis.

Tablica 9.24 *Prikaz naredbe za postavljanje pokazivača na određeni* ODT *zapis*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Kod naredbe: 0xD3
1	*BYTE*	Rezervirano
2	*WORD*	Redni broj DAQ liste
4	*BYTE*	Redni broj ODT liste
5	*BYTE*	Redni broj ODT elementa

9.13 Naredba za upisivanje podataka u određeni ODT element

Ova naredba, tj. *WRITE_DAQ* naredba[4], koristi se kako bi se upisali podatci na neki ODT element, a taj ODT element je određen prethodnim slanjem naredbe za postavljanje pokazivača na željeni ODT element. Tablica 9.25 prikazuje izgled naredbe za upisivanje podataka u neki ODT element.

Tablica 9.25 *Prikaz naredbe za upisivanje podataka u* ODT *element*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Kod naredbe: 0xE1
1	*BYTE*	Pomak bita
2	*BYTE*	Veličina zapisa
3	*BYTE*	Adresna ekstenzija elementa koji će se upisati
4	*DWORD*	Adresa elementa koji će se upisati

Pojedini ODT elementi su zapravo memorijske lokacije s kojima će se izvoditi operacije, tj. na njihovo mjesto će se upisivati podatci u slučaju stimulacije (slanja) podataka ili će se s tih memorijskih lokacija čitati vrijednosti ako se radi o mjerenju. Često nije potrebno čitati cijelu vrijednost s neke memorijske adrese već je potrebno pročitati samo stanje određenog bita, te se zato koristi i pomak bita (engl. *bit offset*) koji će se primijeniti kako bi se došlo do željene vrijednosti. U ovom diplomskom radu implementirana je opcija samo za čitanje podataka, tj. samo opcija za mjerenje.

9.14 Naredba za oslobađanje svih resursa zauzetih za DAQ liste

Naredba za oslobađanje svih resursa zauzetim za DAQ liste, tj. *FREE_DAQ* naredba[4], koristi se kako bi se oslobodila sva memorija koja je zauzeta alociranje DAQ lista, ODT lista i ODT zapisa. Tablica 9.26 prikazuje izgled naredbe za oslobađanje svih resursa koji su zauzeti zbog realizacije DAQ lista.

Tablica 9.26 *Prikaz naredbe za oslobađanje svih resursa zauzetih za* DAQ *liste*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Kod naredbe: 0xD6

9.15 Naredba za pokretanje ili zaustavljanje DAQ liste

Naredba za pokretanje ili zaustavljanje DAQ liste, tj. *START_STOP_DAQ_LIST* naredba[4] koristi se kako bi se pokrenula mjerenja ili stimulacija podataka željene DAQ liste. Ova naredba, osim zaustavljanja ili pokretanja neke DAQ liste, omogućuje i odabiranje neke DAQ liste kako bi se stvorio niz odabranih DAQ lista i omogućilo istovremeno pokretanje ili zaustavljanje mjerenja više DAQ lista. Tablica 9.27 prikazuje izgled naredbe za pokretanje,za zaustavljanje ili odabiranje DAQ liste koja se šalje ECU-u.

Tablica 9.27 *Prikaz naredbe za pokretanje ili zaustavljanje* DAQ *liste*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Kod naredbe: 0xDE
1	*BYTE*	Željeni način rada: 00 = zaustavljanje 01 = pokretanje 02 = odabiranje
2	*WORD*	Redni broj DAQ liste

9.16 Naredba za sinkronizirano pokretanje ili zaustavljanje DAQ lista

Naredba za sinkronizirano pokretanje ili zaustavljanje DAQ lista, tj. *START_STOP_SYNCH* naredba[4], koristi se kako bi se zaustavila ili pokrenula mjerenja svih DAQ lista koje su zabilježene (postavljene kao odabrane) korištenjem *START_STOP_DAQ_LIST* naredbe. Tablica 9.28 prikazuje izgled naredbe za sinkroniziranje pokretanje ili zaustavljanje DAQ lista koja se šalje ECU-u.

Tablica 9.28 *Prikaz naredbe za sinkronizirano pokretanje ili zaustavljanje* DAQ *lista*

POZICIJA	**TIP PODATKA**	**Parametri**
0	*BYTE*	Kod naredbe: 0xDD
1	*BYTE*	Željeni način rada: 00 = zaustavljanje svih DAQ lista 01 = pokretanje odabranih DAQ lista 02 = zaustavljanje odabranih DAQ lista

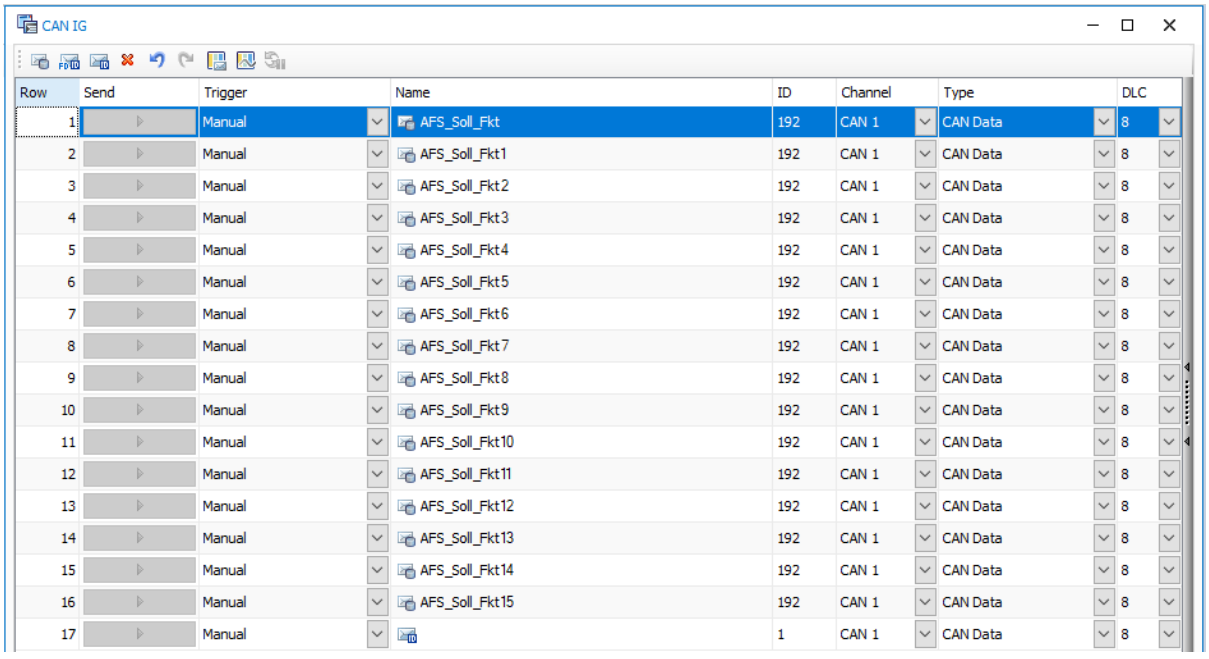
10.MJERENJA I VALIDACIJA IMPLEMENTIRANOG RJEŠENJA

Za sva mjerenja i testiranje ispravnosti implementiranog algoritma koristio se CANoe program. Pomoću CANoe programa šalju se CAN poruke s odgovarajućim kodovima naredbi i dodatnim parametrima te se u posebnom prozoru promatraju odgovori koji su formirani i poslanu na AURIX platformi. Kako bi se razvojna ploča mogla spojiti sa CANoe programom za to je potreban hardver, a u ovom slučaju korišten je *Vector VN5610A* uređaj s dva CAN kanala, tj. s dvije CAN sabirnice: CAN1 i CAN2.

Na sve CAN poruke poslane na kanalu 1 iz CANoe programa AURIX platforma odgovara na kanalu 2 s identifikacijskim brojem 100 ako se radi o klasičnim naredbama, a ako se radi o mjerenjima putem DAQ lista CAN poruke koje će biti poslane kao odgovor imaju identifikacijske brojeve od 64.

Prvo testiranje implementiranog rješenja fokusira se na funkcionalnosti sustava, tj. testiranje ispravnosti svih implementiranih funkcija i naredbi XCP protokola. Drugo testiranje fokusira se na performanse implementiranog rješenja tako što sabirnicu dovodi u veliko opterećenje slanjem velikog broja poruka u kratkom vremenskom intervalu.

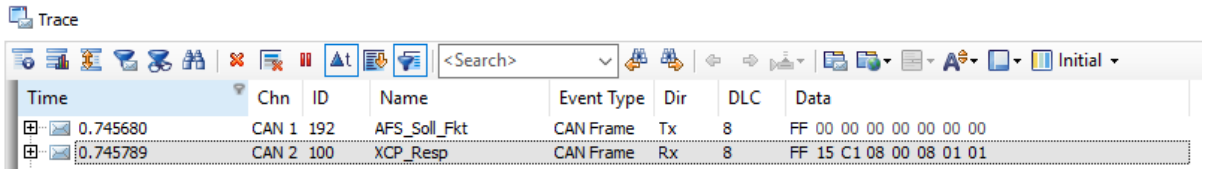
10.1. Testiranje funkcionalnosti implementiranog rješenja



Row	Send	Trigger	Name	ID	Channel	Type	DLC
1		Manual	AFS_Soll_Fkt	192	CAN 1	CAN Data	8
2		Manual	AFS_Soll_Fkt1	192	CAN 1	CAN Data	8
3		Manual	AFS_Soll_Fkt2	192	CAN 1	CAN Data	8
4		Manual	AFS_Soll_Fkt3	192	CAN 1	CAN Data	8
5		Manual	AFS_Soll_Fkt4	192	CAN 1	CAN Data	8
6		Manual	AFS_Soll_Fkt5	192	CAN 1	CAN Data	8
7		Manual	AFS_Soll_Fkt6	192	CAN 1	CAN Data	8
8		Manual	AFS_Soll_Fkt7	192	CAN 1	CAN Data	8
9		Manual	AFS_Soll_Fkt8	192	CAN 1	CAN Data	8
10		Manual	AFS_Soll_Fkt9	192	CAN 1	CAN Data	8
11		Manual	AFS_Soll_Fkt10	192	CAN 1	CAN Data	8
12		Manual	AFS_Soll_Fkt11	192	CAN 1	CAN Data	8
13		Manual	AFS_Soll_Fkt12	192	CAN 1	CAN Data	8
14		Manual	AFS_Soll_Fkt13	192	CAN 1	CAN Data	8
15		Manual	AFS_Soll_Fkt14	192	CAN 1	CAN Data	8
16		Manual	AFS_Soll_Fkt15	192	CAN 1	CAN Data	8
17		Manual		1	CAN 1	CAN Data	8

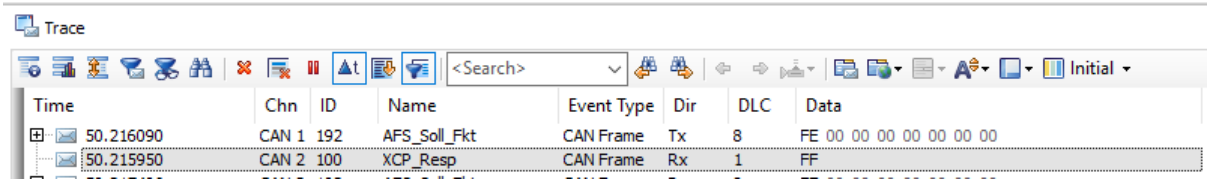
Sl. 10.1. *Prikaz* CAN *poruka koje se mogu poslati pomoću CANoe programa*

Slika 10.1 prikazuje popis svih CAN poruka koje su se koristile prilikom testiranja sustava.

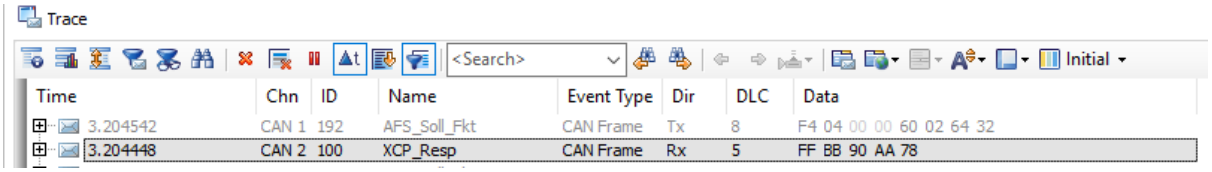


Sl. 10.2. *Prikaz poslane CAN poruke s naredbom za spajanje (connect) i primljenog odgovora*

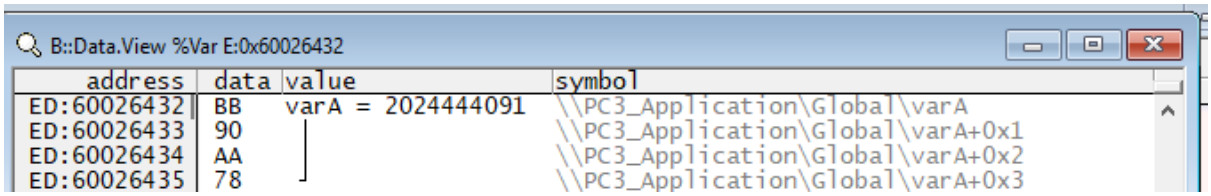
Kao što je i vidljivo na slici 10.2, pomoću CANoe programa šalje se CAN poruka s naredbom za spajanje (0xFF000000) s identifikacijskim brojem 192. AURIX platforma odgovara na poslanu naredbu s CAN porukom identifikacijskog broja 100, te s podatcima koji su formirani na način opisan u poglavlju šest. Ako se pošalje bilo koja druga naredba, prije nego li se pošalje naredba za spajanje s AURIX platformom, AURIX platforma neće odgovoriti na tu naredbu. Slika 10.3 prikazuje odspajanje AURIX platforme i CANoe alata, tj. putem CANoe alata šalje se naredba 0xFE000000, a AURIX platforma odgovara sa 0xFF000000 što označava uspješno odspajanje, a to znači da AURIX platforma neće odgovarati na bilo kakve druge naredbe prije ponovnog spajanja.



Sl. 10.3. *Prikaz poslane CAN poruke s naredbom za odspajanje (disconnect naredba) i primljenog odgovora*

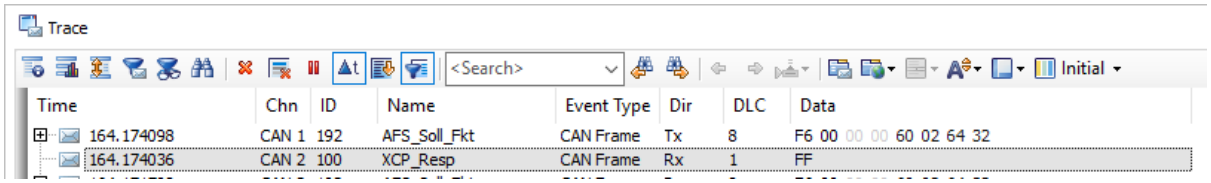


Sl. 10.4. *Prikaz poslane CAN poruke s naredbom za dohvaćanje podataka sa željene memorijske adrese (short upload naredba) i primljenog odgovora*

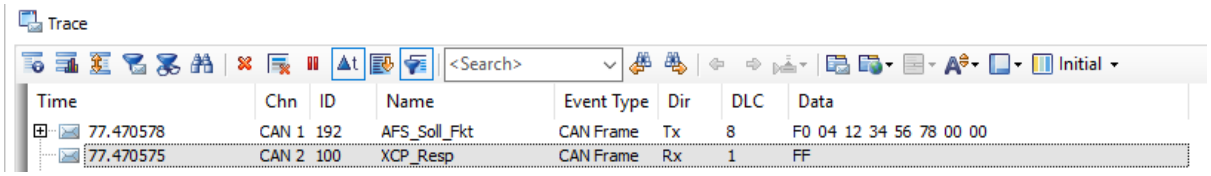


Sl. 10.5. *Prikaz memorijske adrese i vrijednosti spremljene na istoj pomoću lauterbach uređaja*

Kao što je i vidljivo na slici 10.4 AURIX platformi je poslana naredba za čitanje podataka spremljenoj na memorijskoj adresi 0x60026432. AURIX platforma odgovara s porukom 0xFFBB90AA78 što znači da je na memorijskoj adresi 0x60026432 spremljena heksadecimalna vrijednost BB90AA78. Memorijska adresa s koje su pročitani podatci pronađena je pomoću *lauterbach*[5] uređaja za otklanjanje pogrešaka koji je bio spojen na AURIX platformu, te su ti podatci vidljivi na slici 10.5.



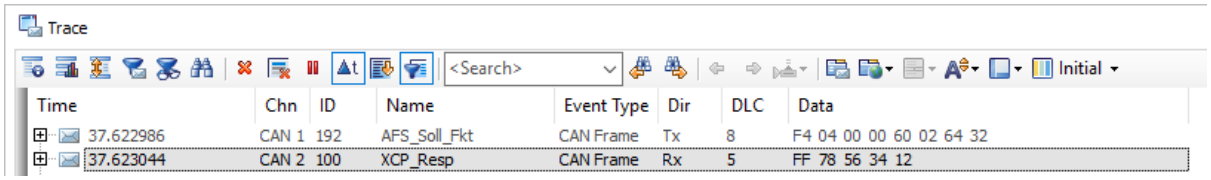
Sl. 10.6. *Prikaz poslane CAN poruke s naredbom za postavljanje pokazivača na određenu memorijsku adresu (set mta naredba) i primljenog odgovora*



Sl. 10.7. *Prikaz poslane CAN poruke s naredbom za promjenom vrijednosti memorijske adrese(download naredba) i primljenog odgovora*

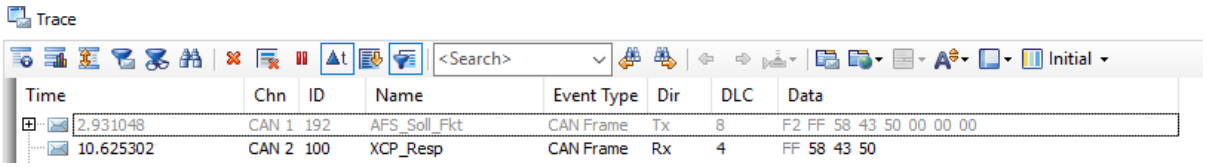
Ako se želi promijeniti vrijednost neke varijable na AURIX platformi potrebno je koristiti dvije naredbe. Prvo je potrebno koristiti naredbu za postavljanje pokazivača kako bi se postavio pokazivač na memorijsku adresu varijable čija se vrijednost želi promijeniti. Nakon toga potrebno je koristiti naredbu za slanje podataka gdje se šalju samo podatci koji se žele spremiti na AURIX platformi, a ti podatci se spremaju na lokaciju koja je prethodno određena korištenjem naredbe za postavljanjem pokazivača. Kao što je i u poglavlju šest opisano, kako bi se koristila naredba za promjenu vrijednosti spremljene na određenoj

memorijskoj lokaciji potrebno je prvo postaviti pokazivač na tu lokaciju, a to je vidljivo na slici 10.6. AURIX platformi se šalje naredba 0xF600000060026432 što znači da se pokazivač postavlja na memorijsku adresu 0x60026432, a AURIX platforma odgovara s 0xFF što znači da je pokazivač uspješno postavljen. Nakon uspješnog postavljanja pokazivača na memorijsku adresu koristi se naredba za postavljanje novih vrijednosti na tu adresu, a to je vidljivo na slici 10.7. AURIX platformi se šalje naredba s vrijednosti 0xF004123456780000 što znači da se pomoću prethodno postavljenog pokazivača na željenu memorijsku adresu upiše vrijednost 0x12345678.

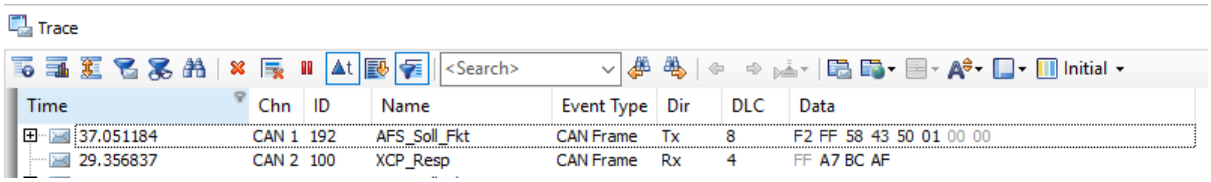


Sl. 10.8. *Prikaz poslane CAN poruke s naredbom za dohvaćanje podataka sa željene memorijske adrese (short upload naredba) i primljenog odgovora*

Kako bi se provjerilo da li je AURIX platforma promijenila vrijednost na željenoj memorijskoj adresi potrebno je ponovno pročitati vrijednost na istoj. Ponovnim slanjem naredbe za čitanje s memorijske adrese AURIX platformi vidljivo je da je vrijednost promijenjena tako što AURIX platforma odgovara s vrijednosti 0xFF78563412. Rezultati promjene vidljivi su na slici 10.8.



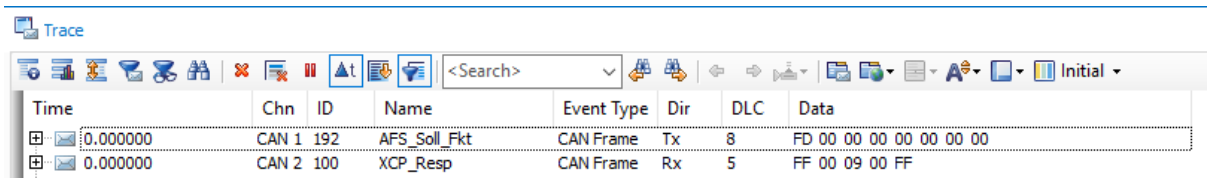
Sl. 10.9. *Prikaz poslane CAN poruke s naredbom za identificiranje podređenog uređaja i primljenog odgovora*



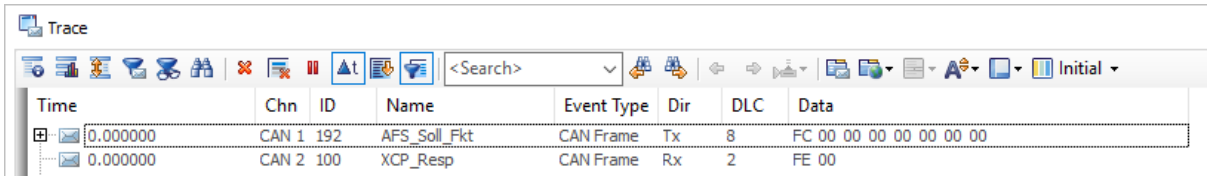
Sl. 10.10. *Prikaz poslane CAN poruke s naredbom za identificiranje podređenog uređaja i primljenog odgovora*

Na slici 10.9 vidljivo je da je poslana naredba za dohvaćanje identifikacije podređenog uređaja, a kako je vrijednost šestog bajta postavljena na 0x00 podređeni uređaj na tu poruku mora odgovoriti sa istim parametrima koje je primio, tj. sa 0x584350. Ispravan odgovor također je vidljiv na slici 10.9 pomoću poruke koju šalje AURIX platforma sa identifikacijskim brojem 100 i vrijednosti 0xFF584350.

Ponovnim slanjem naredbe za dohvaćanje identifikacije podređenog uređaja, ali ovaj puta s vrijednosti šestog bajta postavljenim na 0x01, podređeni uređaj mora odgovoriti s porukom koja sadrži invertirane vrijednosti trećeg, četvrtog i petog bita primljene poruke. Poslana naredba i ispravan odgovor vidljiv je na slici 10.10, tj. AURIX platformi se šalje poruka s vrijednosti 0xF258435001, a AURIX platforma odgovara s porukom 0xFFA7BCAF.

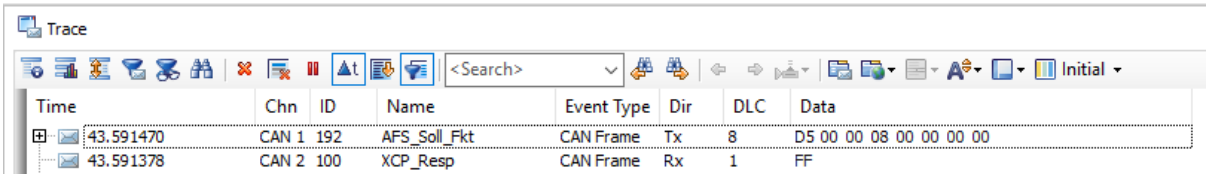


Sl. 10.11. *Prikaz poslane CAN poruke s naredbom za dohvaćanjem trenutnog statusa podređenog uređaja i primljenog odgovora*

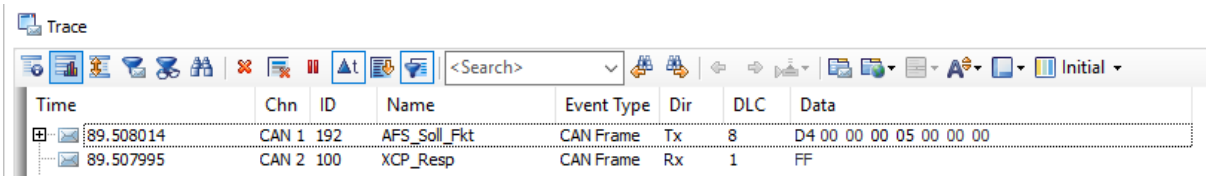


Sl. 10.12. *Prikaz poslane CAN poruke s naredbom za sinkroniziranje i primljenog odgovora*

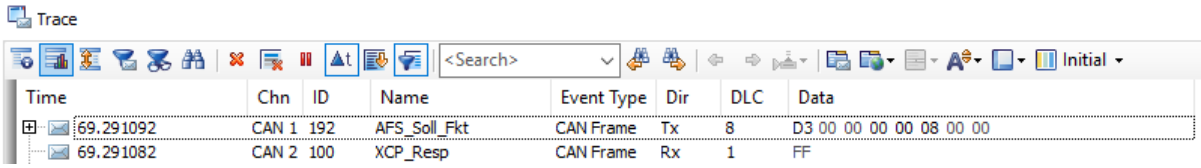
Na slikama 10.11 i 10.12 vidljive su naredbe za dohvaćanje trenutnog statusa podređenog uređaja i sinkroniziranje, a značenje primljenih odgovora objašnjeno je u poglavlju devet. Slika 10.11 prikazuje da je AURIX platformi poslana naredba za dohvaćanjem trenutno status na što AURIX platforma odgovara s vrijednosti 0x009000FF. Trenutni status ukazuje na to da li se trenutno izvršavaju DAQ mjerenja, u kojem načinu rada se uređaj nalazi i slično, a točan način za interpretaciju primljenih podataka opisan je u poglavlju devet. Slika 10.12 prikazuje slanje naredbe za sinkronizaciju AURIX platformi. AURIX platformi šalje se vrijednost 0xFC, a AURIX platforma odgovara sa 0xFE00.



Sl. 10.13. *Prikaz poslane CAN poruke s naredbom za alociranje 8* DAQ *lista i primljenog odgovora*

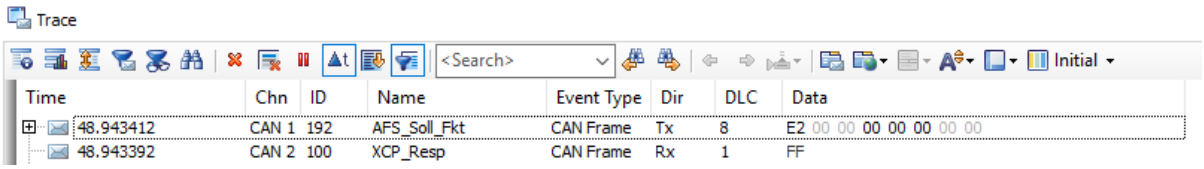


Sl. 10.14. *Prikaz poslane CAN poruke s naredbom za alociranje 5* ODT *lista u nultoj* DAQ *listi i primljenog odgovora*

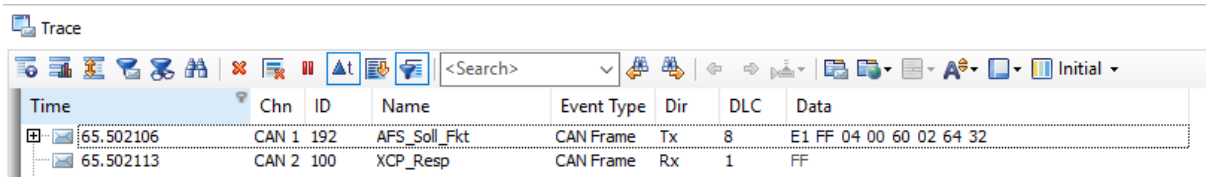


Sl. 10.15. *Prikaz poslane CAN poruke s naredbom za alociranje 8* ODT *zapisa u nultoj* ODT *listi unutar nulte* DAQ *listi i primljenog odgovora*

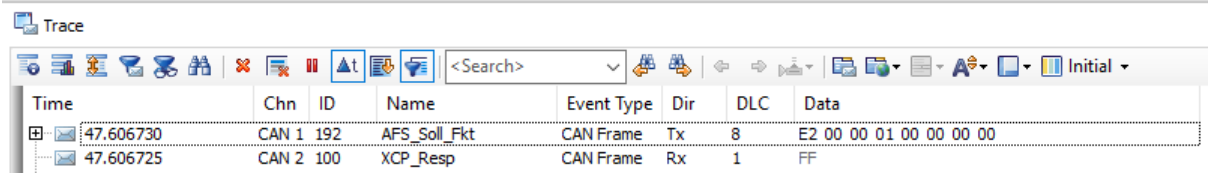
Prije bilo kakvog rada s dinamičkim DAQ listama potrebno je prvo alocirati iste. Alociranje 8 DAQ lista prikazano je na slici 10.13. Kako bi se ODT liste dodale nekoj DAQ listi potrebno je i za to poslati naredbu , što je vidljivo na slici 10.14 gdje je nultoj DAQ listi dodijeljeno 5 ODT lista. Na posljetku potrebno je alocirati i ODT zapise, a na slici 10.15 vidljivo je da je alocirano 8 ODT zapisa, te su isti dodijeljeni nultoj ODT listi u nultoj DAQ listi. Na sve ove naredbe AURIX platforma je odgovorila s porukom 0xFF što znači da su zahtjevi uspješno obrađeni.



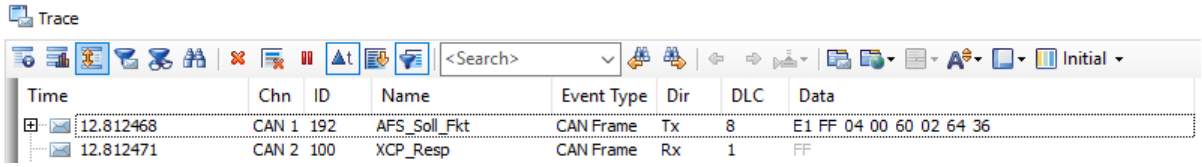
Sl. 10.16. *Prikaz poslane CAN poruke s naredbom za postavljanje pokazivača na nulti* ODT *zapis unutar nulte* ODT *liste koja pripada nultoj* DAQ *listi i primljenog odgovora*



Sl. 10.17. *Prikaz poslane CAN poruke s naredbom za dodjeljivanje vrijednosti, tj. postavljanje memorijske adrese na ranije postavljen* DAQ *pokazivač i primljenog odgovora*

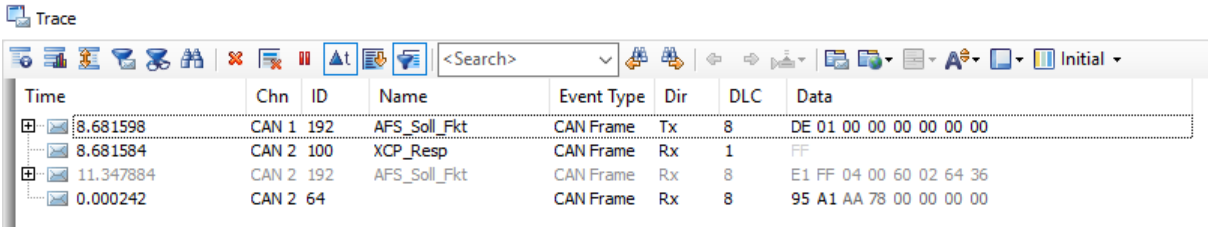


Sl. 10.18. *Prikaz poslane CAN poruke s naredbom za postavljanje pokazivača na nulti* ODT *zapis unutar nulte* ODT *liste koja pripada prvoj* DAQ *listi i primljenog odgovora*



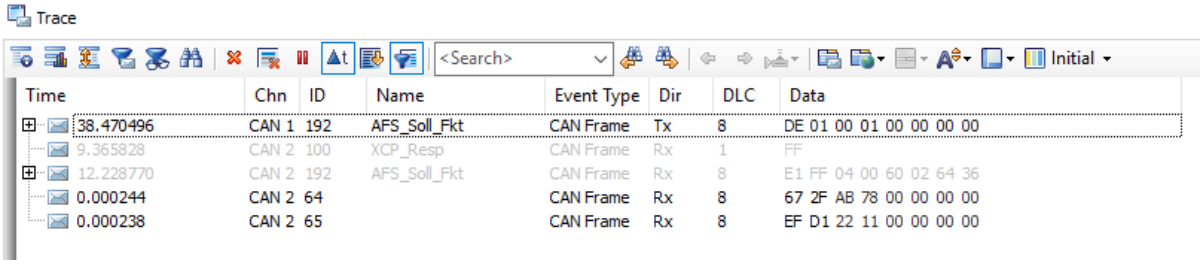
Sl. 10.19. *Prikaz poslane CAN poruke s naredbom za dodijeljivanje vrijednosti, tj. postavljanje memorijske adrese na ranije postavljen* DAQ *pokazivač i primljenog odgovora*

Kako bi se koristile DAQ liste za mjerenja, nakon alokacije, potrebno je dodati same varijable, tj. memorijske lokacije na DAQ liste. Na slici 10.16 prikazano je postavljanje pokazivača na nulti ODT zapis koji je član nulte ODT liste unutar nulte DAQ liste. Tek nakon postavljanja pokazivača moguće je dodati varijablu na DAQ listu, a na slici 10.17 prikazano je dodjeljivanje memorijske adrese 0x60026432. Isti postupak ponovljen je još jednom uz razliku u ODT elementu i dodijeljenoj memorijskoj adresi. Pokazivač se postavlja na nulti ODT zapis unutar prve ODT liste koja je član nulte DAQ liste, a to je vidljivo na slici 10.18. Nakon toga potrebno je dodati i vrijednost memorijskoj adresi na koju pokazivač pokazuje, a to je vidljivo na slici 10.19. Slika 10.19 prikazuje kako je memorijska adresa 0x60026436 dodana u ODT zapis na koji je prethodno postavljen pokazivač.



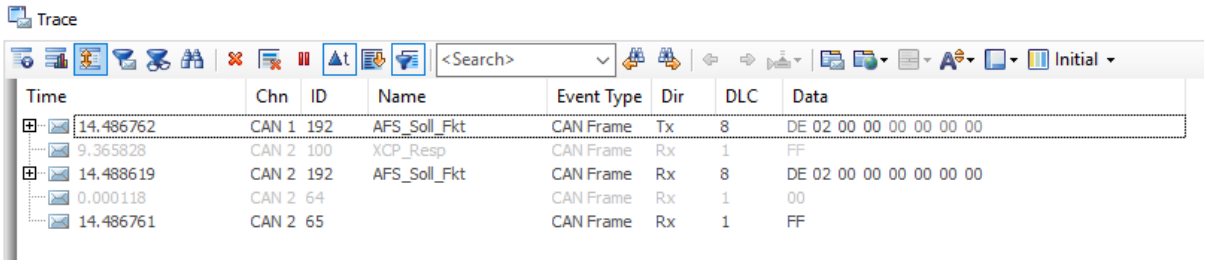
Sl. 10.20. *Prikaz poslane CAN poruke s naredbom za pokretanje nulte* DAQ *liste i primljenog odgovora*

Kako bi mjerenje započelo potrebno je poslati naredbu za pokretanje željene DAQ liste. Na slici 10.20 prikazano je pokretanje prve DAQ liste, a kako prva DAQ lista ima jedan element, samo jedan poruka, s identifikacijskim brojem 64, se šalje od strane AURIX platforme svakih 100 milisekundi.

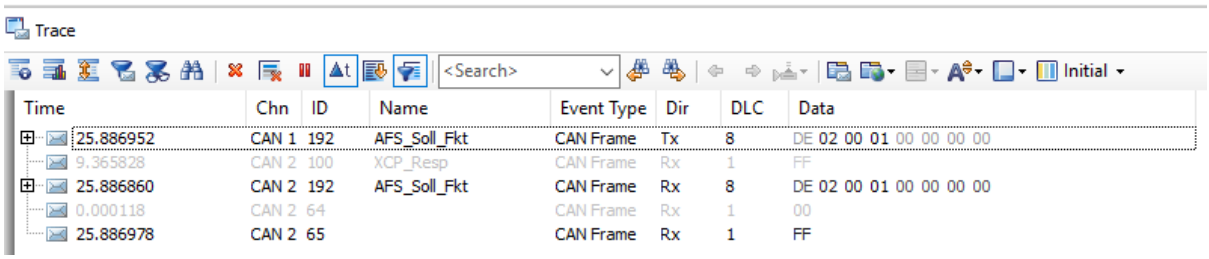


Sl. 10.21. *Prikaz poslane CAN poruke s naredbom za pokretanje prve* DAQ *liste i primljenog odgovora*

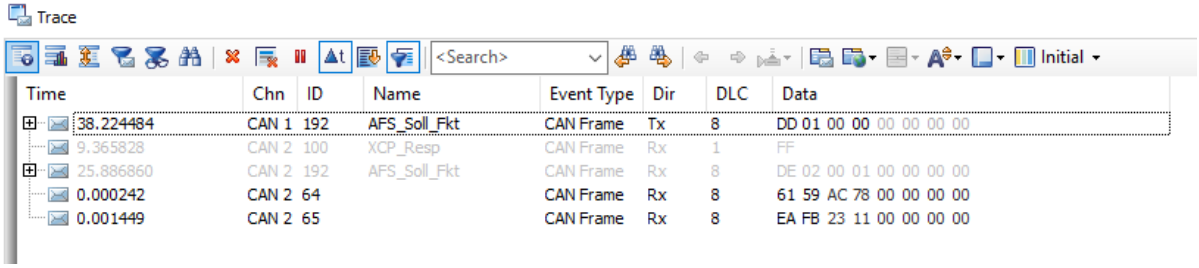
Iako je mjerenje već bilo u tijeku, moguće je ponovno poslati naredbu za pokretanje mjerenja i neke druge DAQ liste, u ovom slučaju prve DAQ liste. Na slici 10.21 vidljivo je da se vrijednosti mjerenja nulte DAQ liste šalju s identifikacijskim brojem poruke 64, a vrijednosti mjerenja prve DAQ liste šalju s identifikacijskim brojem poruke 65. Ova mjerenja šalju se sve dok se ne pošalje naredba za zaustavljanjem mjerenja jedne ili svih DAQ lista.



Sl. 10.22. *Prikaz poslane CAN poruke s naredbom za postavljanje nulte* DAQ *liste na popis odabranih i primljenog odgovora*

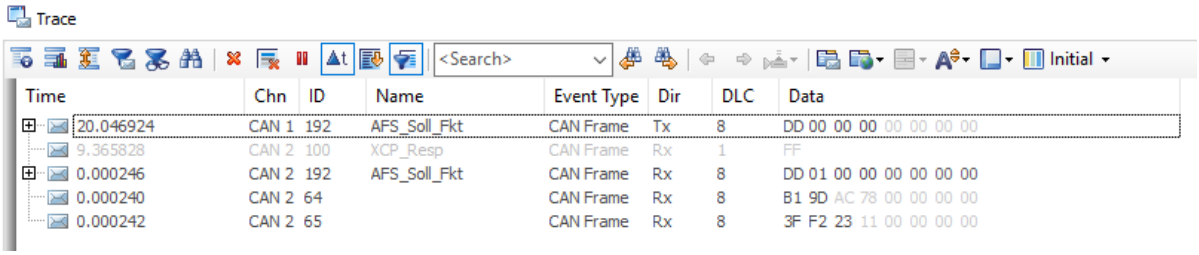


Sl. 10.23. *Prikaz poslane CAN poruke s naredbom za postavljanje prve* DAQ *liste na popis odabranih i primljenog odgovora*



Sl. 10.24. *Prikaz poslane CAN poruke s naredbom za postavljanje prve* DAQ *liste na popis odabranih i primljenog odgovora*

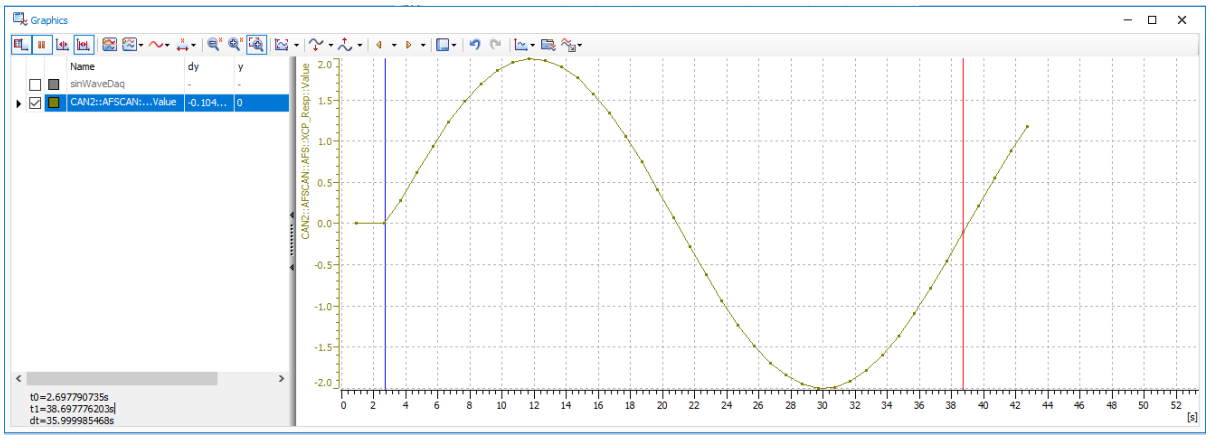
Zaustavljanje/pokretanje DAQ lista može se napraviti na više načina, tj. moguće je svaku listu zasebno zaustaviti/pokrenuti, moguće je sve liste od jednom zaustaviti/pokrenuti ili pak odabrati željene DAQ liste i onda zaustaviti/pokrenuti samo odabrane DAQ liste. Na slikama 10.22 i 10.23 vidljivo je da su nulta i prva DAQ lista postavljene na listu odabranih, te kako bi ih se zaustavilo ili ponovno pokrenulo potrebno je poslati naredbu za sinkronizirano pokretanje/zaustavljanje. Slika 10.22 prikazuje kako se AURIX platformi šalje naredba s vrijednosti 0xDE020000 što znači da se nulta DAQ lista dodala na listu trenutno odabranih lista. Slika 10.23 prikazuje slanje AURIX platformi poruke s vrijednost 0xDE020001 što označava dodavanje prve DAQ liste na listu trenutno odabranih DAQ lista. Na slici 10.24 vidljivo je da su sinkronizirano, tj. istovremeno pokrenute nulta i prva DAQ liste koje su bile dodane na popis odabranih lista slanjem naredbe s vrijednosti 0xDD010000. AURIX platforma na to odgovara stalnim slanjem podataka s memorijskih adresa koje su dodane u DAQ liste, a to je vidljivo na slici 10.24 s porukama koje imaju ID 64 i 65.



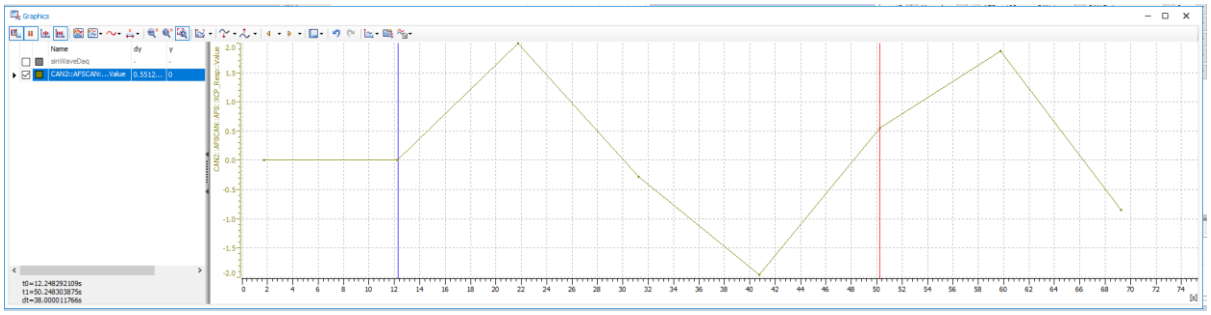
Sl. 10.25. *Prikaz poslane CAN poruke s naredbom za zaustavljanje svih* DAQ *lista i primljenog odgovora*

Zaustavljanje pokrenutih DAQ lista obavljeno je pomoću naredbe za zaustavljanje svih pokrenutih DAQ lista, što je i vidljivo na slici 10.25.

Sl. 10.26. *Prikaz čitanja vrijednosti varijable pomoću koje je računat vrijednost sinus funkcije s lauterbach uređajem za uklanjanje pogrešaka*



Sl. 10.27. *Prikaz čitanja vrijednosti varijable pomoću koje je računat vrijednost sinus funkcije s CANoe programom*



Sl. 10.28. *Prikaz čitanja vrijednosti varijable pomoću koje je računat vrijednost sinus funkcije s CANoe programom*

Radi testiranja ispravnost implementirani XCP funkcija implementirana je i funkcija koja računa vrijednost sinus funkcije svakih 100 milisekundi po formuli $f = r * \sin(\alpha)$ gdje je r = 2, a α kut koji se svakih 100 milisekundi mjenjao za 1 u intervalu od 0 do 360. Korištenjem *lauterbach* uređaja i *trace32* programa vrijednost izračunate varijable čitala se direktno iz razvojne ploče te je iscrtan graf prikazan na slici 10.26 a putem CANoe programa periodično je slana CAN poruka s naredbom za čitanje vrijednosti iste varijable te je iscrtan graf prikazan na slici 10.27. Usporedbom grafova sa slika 10.26 i 10.27 vidljivo je da algoritam radi ispravno, te da su primljeni podatci valjani. Na slici 10.28 prikazan je graf koji je nacrtan pomoću CANoe programa gdje se povećao period između slanja naredbi za čitanjem vrijednosti. Na taj način krši se Nyquistovog pravilo uzorkovanja, tj. frekvencija uzorkovanja je bila znatno manja od frekvencije signala pa je to rezultiralo grafom koji ne izgleda kao graf na slici 10.26. Vidljivo je da su pročitani podatci i dalje ispravni jer otprilike 9 sekundi nakon početka mjerenja, a početak mjerenja je označen plavom crtom na slikama 10.26 i 10.27, grafovi prikazani na obje slike imaju istu vrijednost na y osi (vrijednost 2).

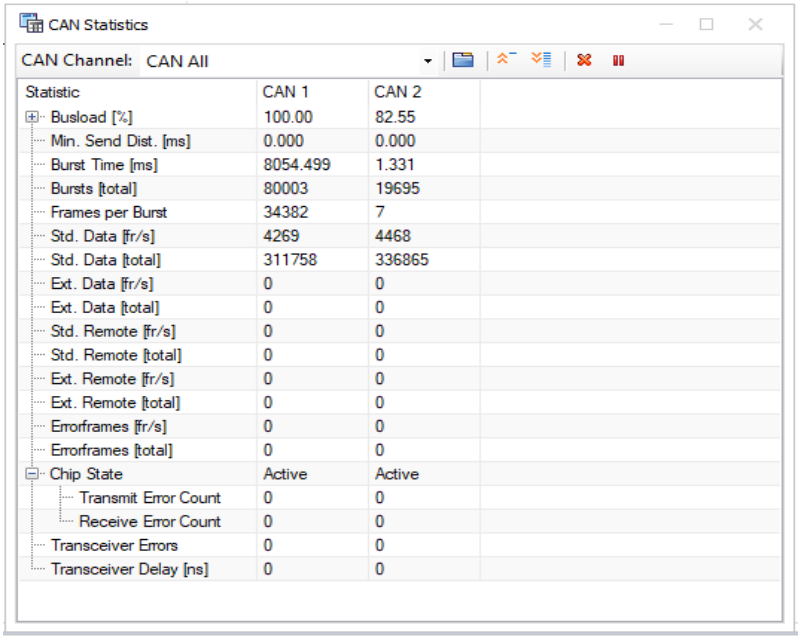
10.2. Testiranje performansi implementiranog rješenja

Performanse implementiranog rješenja testirane su pomoću CANoe programa tako što se sustav, tj. sabirnica i modul, dovode u veliko opterećenje slanjem velikog broja poruka u kratkom vremenskom intervalu. Pomoću CANoe programa periodično, tj. svaku milisekundu, slane su naredbe za čitanje vrijednosti s određenih memorijskih lokacija ECU-a, te je postupno povećavan broj naredbi koji se šalje svake milisekunde sve dok opterećenje sabirnice ne dođe do 100%. Performanse modula su testirane i pomoću DAQ lista tako što se postupno povećavao broj elemenata DAQ lista sve dok opterećenje sabirnice nije postalo približno 100%.

Tablica 10.1. *Prikaz opterećenosti sabirnica prilikom istovremenog slanja više naredbi u vremenskom intervalu od 1 milisekunde*

Broj naredbi koje se istovremeno šalju	**Opterećenje CAN sabirnice 1 [%]**	**Opterećenje CAN sabirnice 2 [%]**
1 naredba	23.42	23.07
2 naredbe	46.85	41.04
3 naredbe	70.27	59.25
4 naredbe	100	82.55

Tablica 10.1 prikazuje opterećenost sabirnica prilikom izvođenja testiranja pomoću CTO naredbi. Testiranje je izvedeno tako što sto se postupno povećavao broj varijabli ECU-a čija se vrijednost čita svake milisekunde. Naredbe se šalju ECU-u na CAN1 sabirnici, a ECU odgovara na CAN2 sabirnici. Nakon što je postavljeno periodično slanje 4 naredbe istovremeno za dohvaćanje vrijednosti određenih varijabli ECU-a opterećenost sabirnice CAN1 je dosegla 100%. Opterećenje od 100% označava da dolazi do gubitaka poruka i da se ne može slati niti jedan podataka više nego što se trenutno šalje. Na slici 10.29 vidljivo i pomoću CANoe programa opterećenje sabirnice CAN1 i CAN2 prilikom slanja 4 naredbe istovremeno svake milisekunde. Slika 10.30 prikazuje 4 naredbe koje su postavljenje na periodično slanje svake milisekunde u CANoe programu.



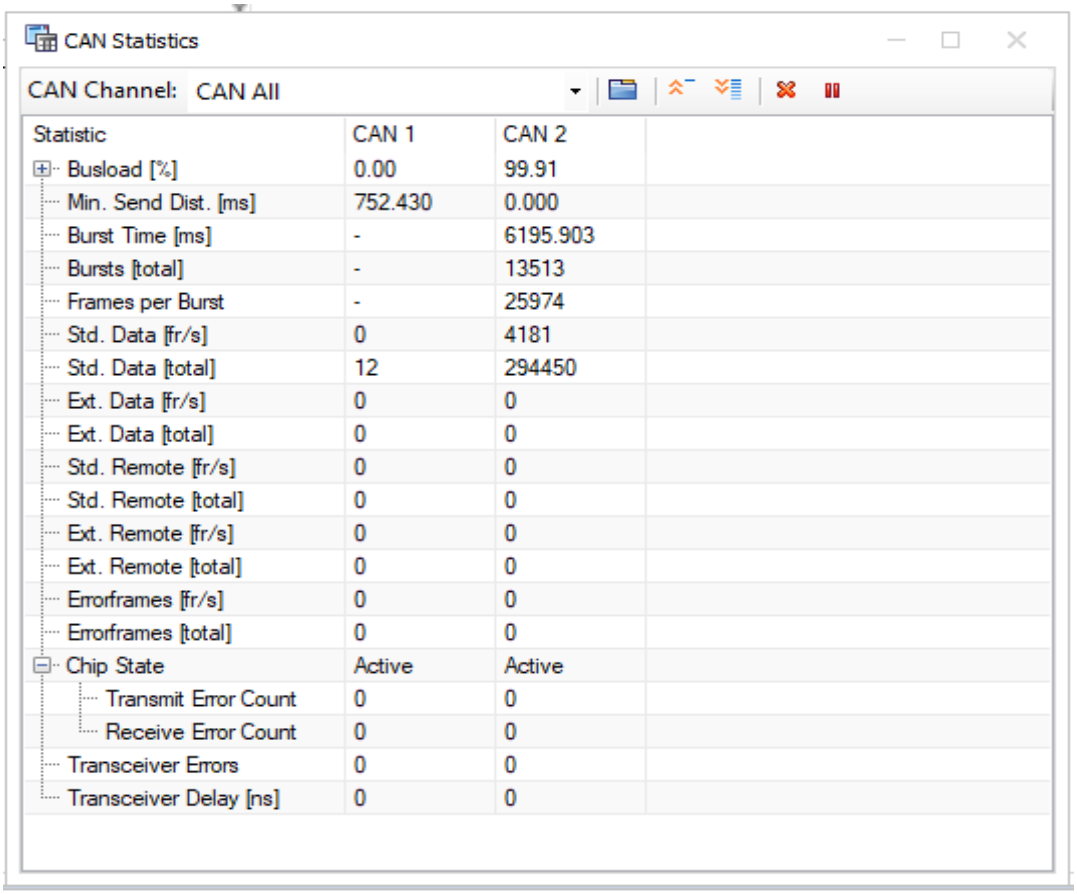
Sl. 10.29. *Prikaz opterećenosti CAN sabirnica u CANoe programu koristeći CTO naredbe u vremenskom intervalu od 1 milisekunde*

Sl. 10.30. *Prikaz naredbi koje su slane svake milisekunde pomoći CANoe programa*

Tablica 10.2. *Prikaz opterećenosti sabirnica prilikom čitanja vrijednosti više varijabli koristeći* DAQ *liste u vremenskom intervalu od 1 milisekunde*

Broj varijabli čije se vrijednosti istovremeno šalju	**Opterećenje CAN sabirnice 1 [%]**	**Opterećenje CAN sabirnice 2 [%]**
1 varijabla	0	23.80
2 varijable	0	47.80
3 varijable	0	71.39
4 varijable	0	99.91

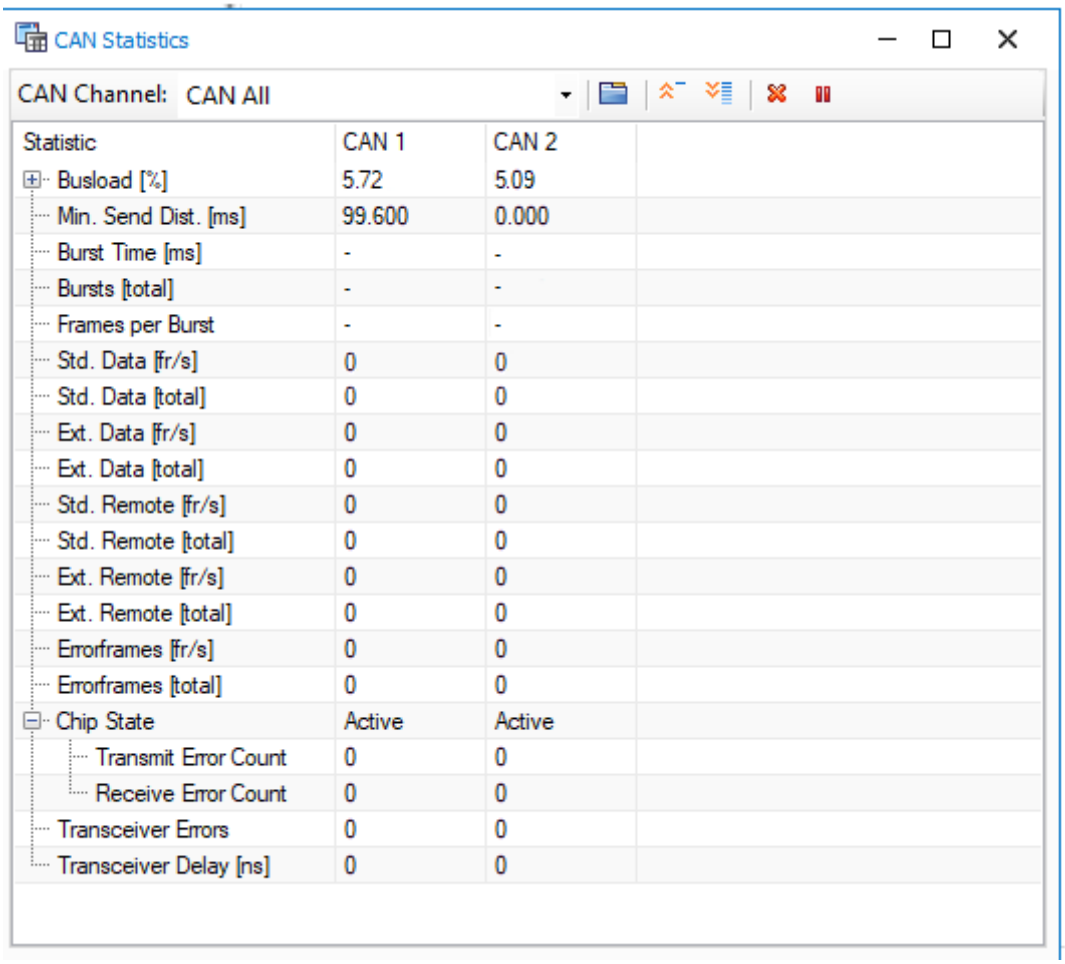
Kao što je i ranije napomenuto jedan od ciljeva korištenja DAQ lista je smanjenje opterećenosti sabirnica koje se koriste što je i vidljivo u tablici 10.2 gdje je opterećenje sabirnice CAN1 0%. Razlog tome što je opterećenje CAN1 sabirnice 0% je to što se CAN1 sabirnica koristi za slanje naredbi ECU-u, a korištenjem DAQ lista potrebno je samo odrediti broj elemenata lista, kao i njihov sadržaj, nakon čega ECU periodično šalje podatke sve dok ne primi naredbu za zaustavljanje. Kao i u slučaju testiranja korištenjem slanja CTO naredbi, i u ovom slučaju, tj. u slučaju korištenja DAQ lista, kada se istovremeno promatraju vrijednosti 4 varijable svake milisekunde opterećenje sabirnice dosegne 99.91 %. Slika 10.31 prikazuje opterećene sabirnica u CANoe programu prilikom korištenja DAQ lista s 4 elementa.



Sl. 10.31. *Prikaz opterećenosti CAN sabirnica u CANoe programu koristeći* DAQ *liste u vremenskom intervalu od 1 milisekunde*

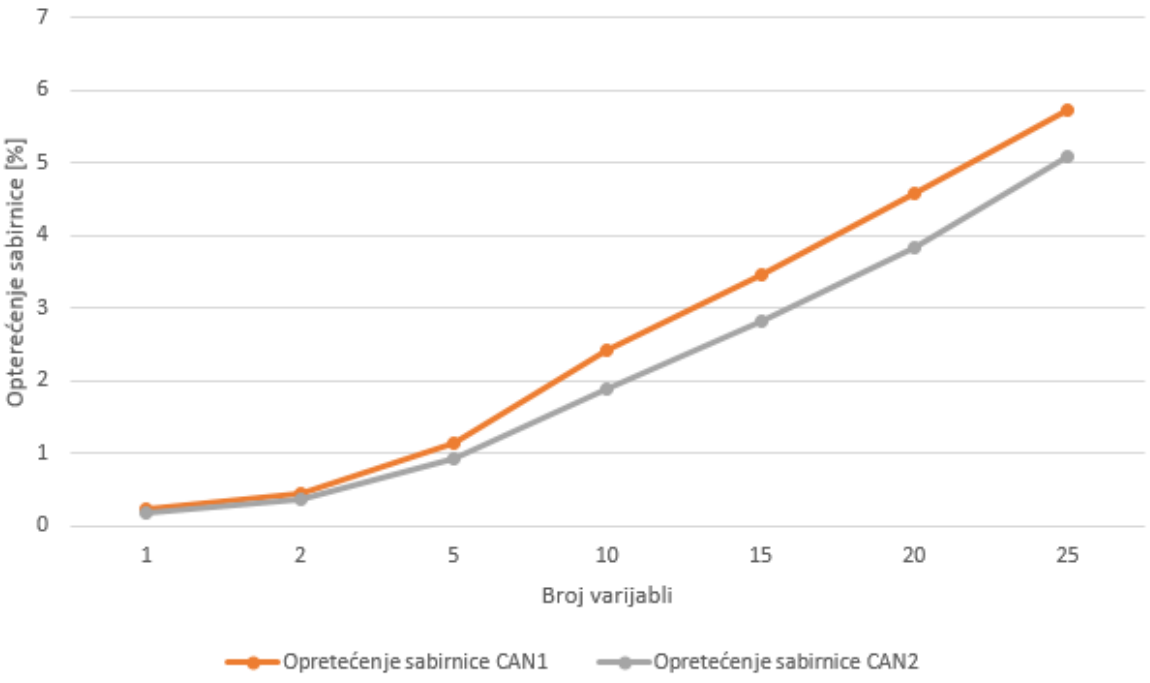
Tablica 10.3. *Prikaz opterećenosti sabirnica prilikom istovremenog slanja više naredbi u vremenskom intervalu od 100 milisekundi*

Broj varijabli čije se vrijednosti istovremeno šalju	**Opterećenje CAN sabirnice 1 [%]**	**Opterećenje CAN sabirnice 2 [%]**
1 varijabla	0.23	0.19
2 varijable	0.44	0.37
5 varijable	1.13	0.94
10 varijabli	2.41	1.89
15 varijabli	3.45	2.82
20 varijabli	4.57	3.83
25 varijabli	5.72	5.09



Statistic	CAN 1	CAN 2
Busload [%]	5.72	5.09
Min. Send Dist. [ms]	99.600	0.000
Burst Time [ms]	-	-
Bursts [total]	-	-
Frames per Burst	-	-
Std. Data [fr/s]	0	0
Std. Data [total]	0	0
Ext. Data [fr/s]	0	0
Ext. Data [total]	0	0
Std. Remote [fr/s]	0	0
Std. Remote [total]	0	0
Ext. Remote [fr/s]	0	0
Ext. Remote [total]	0	0
Errorframes [fr/s]	0	0
Errorframes [total]	0	0
Chip State	Active	Active
Transmit Error Count	0	0
Receive Error Count	0	0
Transceiver Errors	0	0
Transceiver Delay [ns]	0	0

Sl. 10.32. *Prikaz opterećenosti CAN sabirnica u CANoe programu koristeći CTO naredbe u vremenskom intervalu od 100 milisekundi*

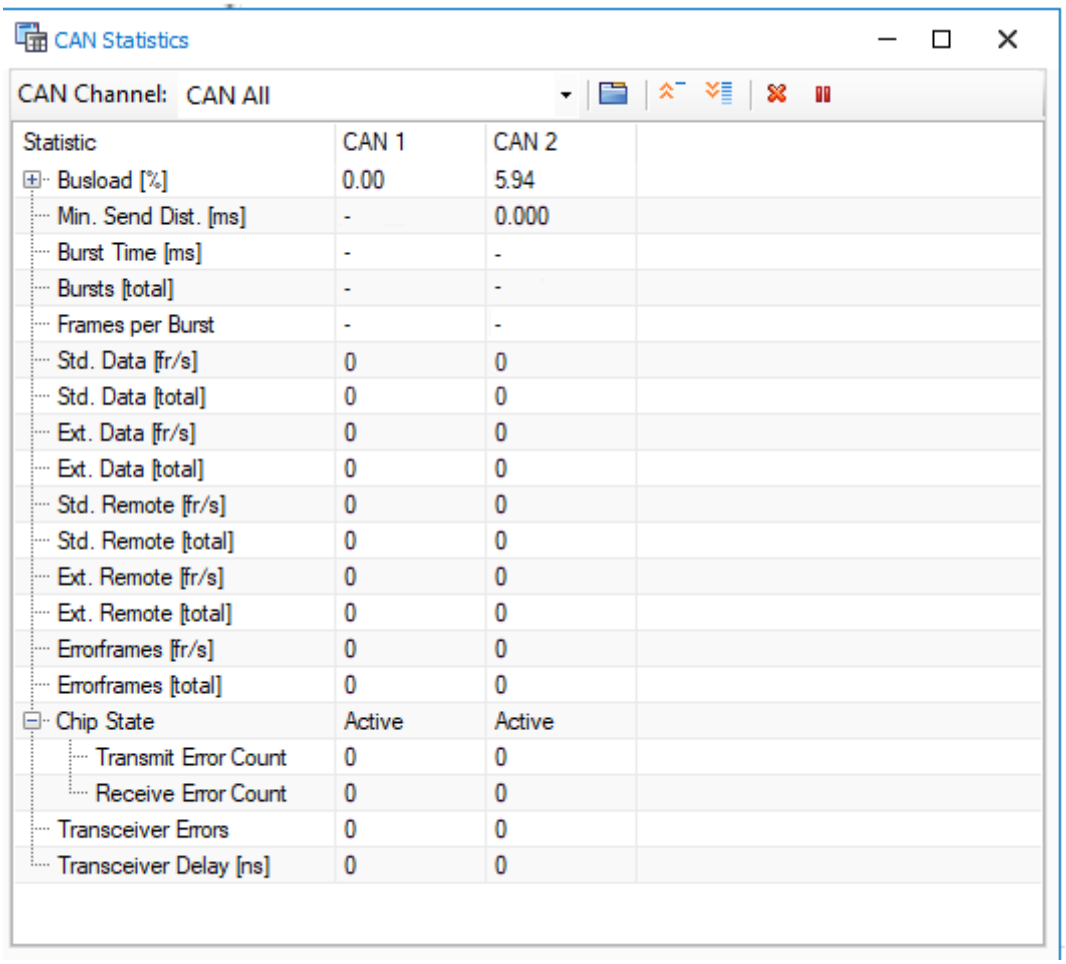


Sl. 10.33. *Grafički prikaz opterećenosti CAN sabirnica u intervalu od 100 milisekundi*

Tablica 10.3 prikazuje opterećenost sabirnica prilikom izvođenja testiranja pomoću CTO naredbi. Testiranje je izvedeno tako što se povećavao broj varijabli čija se vrijednost istovremeno čita svakih 100 milisekundi. Kao i u prethodnim testiranjima naredbe se šalju ECU-u na CAN1 sabirnici, a ECU odgovara na sabirnici CAN2. Uspoređivanjem tablica 10.1 i 10.3 uočava se da se opterećenje sabirnica značajno smanjilo čak i uz praćenje puno većeg broja varijabli. Razlog smanjenju opterećenja sabirnica je povećanje vremenskog intervala između slanja naredbi za dohvaćanje vrijednosti varijabli s ECU-a. Vremenski interval između svake naredbe povećao se s 1 milisekunde na 100 milisekundi, te se može zaključiti da se uz vremenski interval od 100 milisekundi sustav nalazi u optimalnom stanju jer je opterećenost CAN1 sabirnice 5.72%, a CAN2 sabirnice 5.09% uz praćenje vrijednosti čak 25 varijabli. Opterećenost sabirnica prilikom slanja 25 naredbi svakih 100 milisekundi vidljiva je na slici 10.32 pomoću CANoe programa. Na slici 10.33 je grafički prikaz opterećenosti sabirnica CAN1 i CAN2 u vremenskom intervalu od 100 milisekundi, a podatci koji su prikazani nalaze se u tablici 10.3.

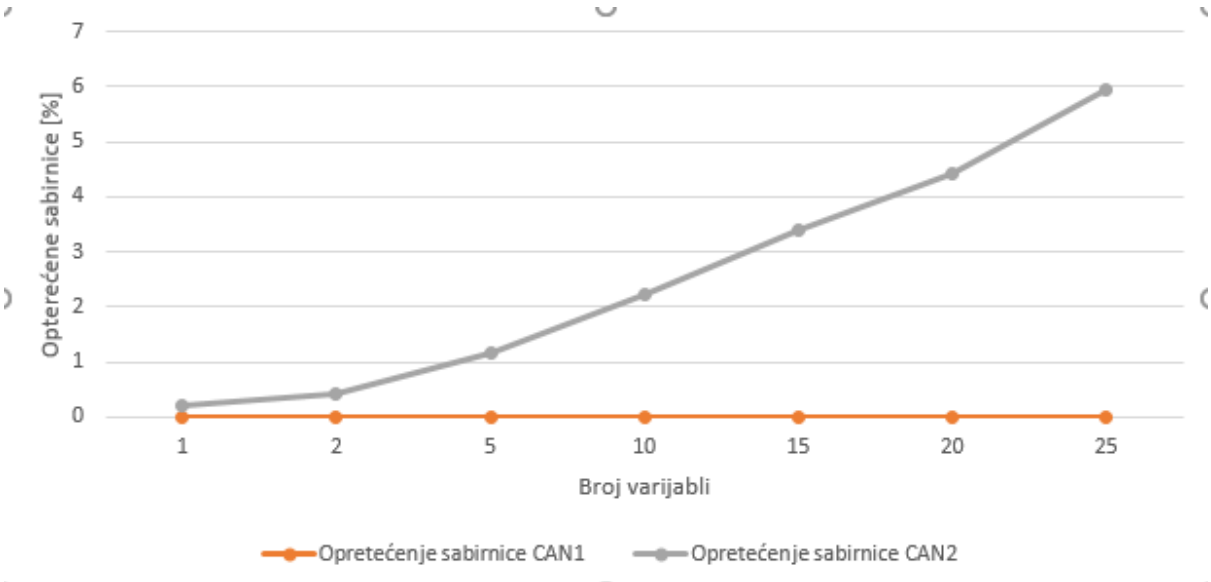
Tablica 10.4. *Prikaz opterećenosti sabirnica prilikom istovremenog slanja više naredbi u vremenskom intervalu od 100 milisekundi koristeći* DAQ *liste*

Broj varijabli čije se vrijednosti istovremeno šalju	**Opterećenje CAN sabirnice 1 [%]**	**Opterećenje CAN sabirnice 2 [%]**
1 naredba	0	0.22
2 naredbe	0	0.43
5 naredbi	0	1.17
10 naredbi	0	2.21
15 naredbi	0	3.39
20 naredbi	0	4.42
25 naredbi	0	5.94



Statistic	CAN 1	CAN 2
Busload [%]	0.00	5.94
Min. Send Dist. [ms]	-	0.000
Burst Time [ms]	-	-
Bursts [total]	-	-
Frames per Burst	-	-
Std. Data [fr/s]	0	0
Std. Data [total]	0	0
Ext. Data [fr/s]	0	0
Ext. Data [total]	0	0
Std. Remote [fr/s]	0	0
Std. Remote [total]	0	0
Ext. Remote [fr/s]	0	0
Ext. Remote [total]	0	0
Errorframes [fr/s]	0	0
Errorframes [total]	0	0
Chip State	Active	Active
Transmit Error Count	0	0
Receive Error Count	0	0
Transceiver Errors	0	0
Transceiver Delay [ns]	0	0

Sl. 10.34. *opterećenosti CAN sabirnica u CANoe programu u vremenskom intervalu od 100 milisekundi koriteći* DAQ *liste*

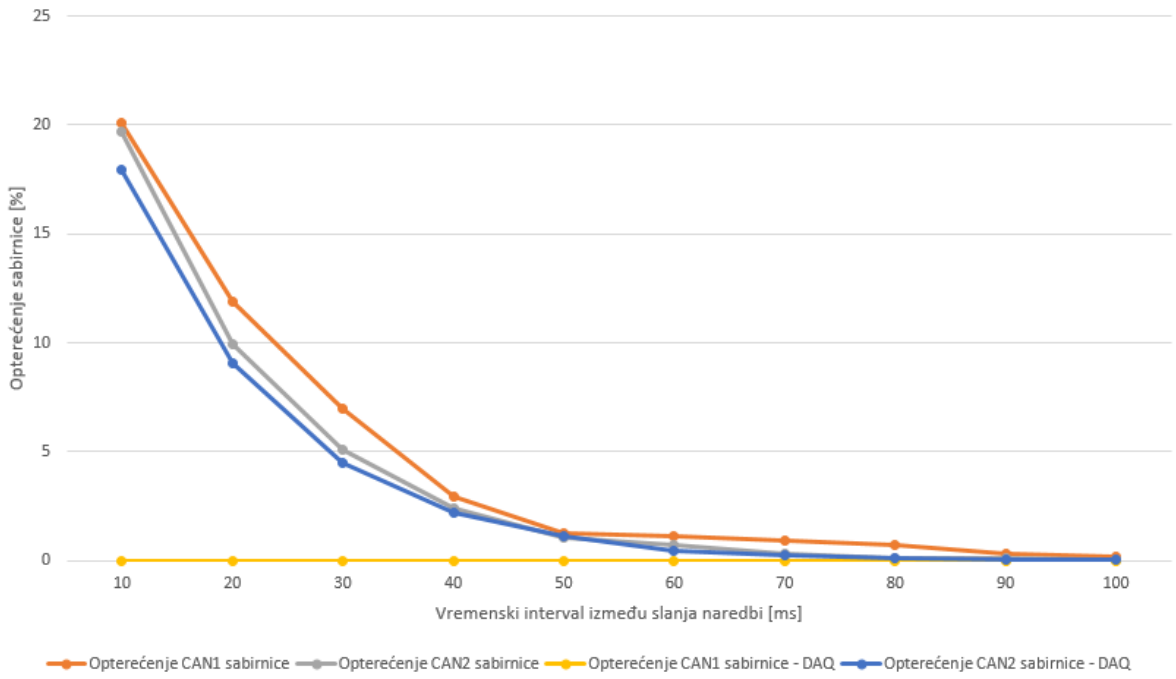


Sl. 10.35. *Grafički prikaz opterećenosti CAN sabirnica u intervalu od 100 milisekundi koristeći DAQ metodu*

Tablica 10.4 prikazuje opterećenost sabirnica prilikom izvođenja testiranja pomoću DAQ lista. Događaj s kojim je bila povezana DAQ lista je vremenski interval koji iznosi 100 milisekundi, što znači da je ECU svakih 100 milisekundi slao CAN poruke s vrijednostima pročitanih s postavljenih varijabli. Na slici 10.34 prikazana je opterećenost sabirnica prilikom slanja vrijednosti 25 varijabli sa ECU-a. Može se zaključiti da se sustav u ovakvim uvjetima nalazi u optimalnom stanju jer je opterećenje CAN2 sabirnice, tj. sabirnice na kojoj ECU šalje podatke samo 5.94%, a CAN1 sabirnice 0%. Nisko opterećene sabirnice CAN2 znači da je moguće dodati puno veći broj varijabli u DAQ liste i istovremeno pratiti njihove vrijednosti, te pritom slati još bilo kakve potrebne dodatne naredbe ECU-u jer je opterećenje i CAN1 sabirnice vrlo nisko, tj. 0%. Na slici 10.35 je grafički prikaz opterećenosti sabirnica CAN1 i CAN2 prilikom korištenja DAQ metode u vremenskom intervalu od 100 milisekundi, a podatci koji su prikazani nalaze se u tablici 10.4.

Tablica 10.5. *Prikaz opterećenosti sabirnica prilikom slanja jedne naredbe u različitim vremenskim intervalima*

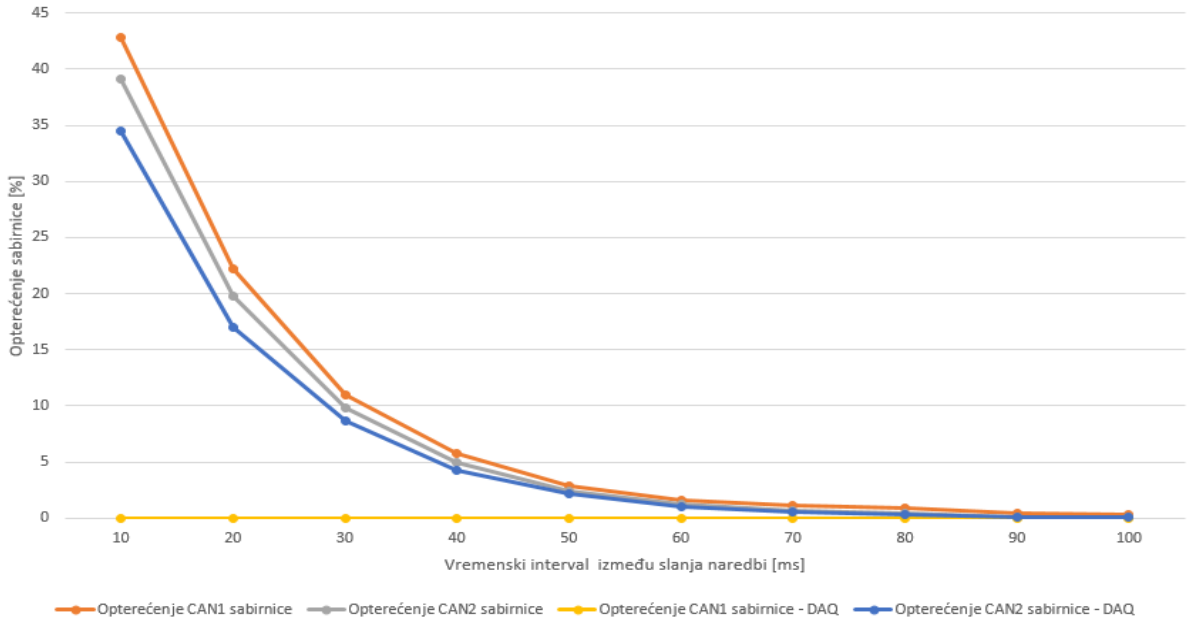
Vremenski interval između slanja naredbi	**Opterećenje CAN sabirnice 1 [%]**	**Opterećenje CAN sabirnice 2 [%]**	**Opterećenje CAN sabirnice 1 korištenjem DAQ metode[%]**	**Opterećenje CAN sabirnice 2 korištenjem DAQ metode [%]**
10 milisekundi	20.07	19.67	0	17.98
20 milisekundi	11.89	9.92	0	9.07
30 milisekundi	7.01	5.07	0	4.49
40 milisekundi	2.92	2.40	0	2.20
50 milisekundi	1.25	1.08	0	1.12
60 milisekundi	1.12	0.71	0	0.49
70 milisekundi	0.96	0.31	0	0.27
80 milisekundi	0.72	0.15	0	0.14
90 milisekundi	0.30	0.09	0	0.06
100 milisekundi	0.18	0.02	0	0.03



Sl. 10.36. *Grafički prikaz opterećenosti CAN sabirnica u različitim vremenskim intervalima prilikom slanja jedne naredbe*

Tablica 10.6. *Prikaz opterećenosti sabirnica prilikom slanja dvije naredbe u različitim vremenskim intervalima*

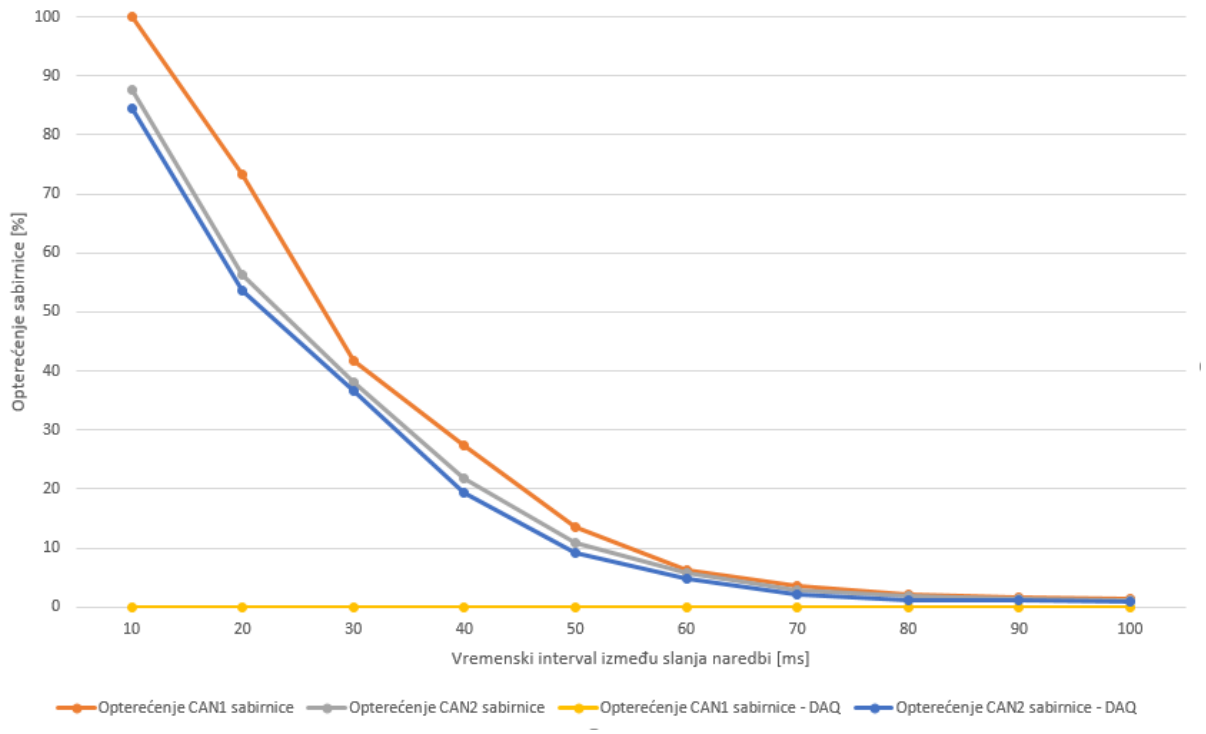
Vremenski interval između slanja naredbi	**Opterećenje CAN sabirnice 1 [%]**	**Opterećenje CAN sabirnice 2 [%]**	**Opterećenje CAN sabirnice 1 korištenjem DAQ metode[%]**	**Opterećenje CAN sabirnice 2 korištenjem DAQ metode [%]**
10 milisekundi	42.85	39.12	0	34.53
20 milisekundi	22.17	19.81	0	17.03
30 milisekundi	11.03	9.80	0	8.63
40 milisekundi	5.77	4.97	0	4.27
50 milisekundi	2.84	2.45	0	2.15
60 milisekundi	1.57	1.20	0	1.06
70 milisekundi	1.09	0.64	0	0.53
80 milisekundi	0.85	0.39	0	0.31
90 milisekundi	0.47	0.15	0	0.13
100 milisekundi	0.27	0.08	0	0.06



Sl. 10.37. *Grafički prikaz opterećenosti CAN sabirnica u različitim vremenskim intervalima prilikom slanja dvije naredbe*

Tablica 10.7. *Prikaz opterećenosti sabirnica prilikom slanja pet naredbi u različitim vremenskim intervalima*

Vremenski interval između slanja naredbi	**Opterećenje CAN sabirnice 1 [%]**	**Opterećenje CAN sabirnice 2 [%]**	**Opterećenje CAN sabirnice 1 korištenjem DAQ metode[%]**	**Opterećenje CAN sabirnice 2 korištenjem DAQ metode [%]**
10 milisekundi	100.00	87.55	0	84.54
20 milisekundi	73.27	56.25	0	53.69
30 milisekundi	41.63	38.04	0	36.62
40 milisekundi	27.31	21.83	0	19.41
50 milisekundi	13.59	10.97	0	9.27
60 milisekundi	6.20	5.66	0	4.71
70 milisekundi	3.49	2.94	0	2.13
80 milisekundi	2.05	1.96	0	1.25
90 milisekundi	1.76	1.42	0	1.07
100 milisekundi	1.32	1.07	0	0.92



Sl. 10.38. *Grafički prikaz opterećenosti CAN sabirnica u različitim vremenskim intervalima prilikom slanja pet naredbi*

Prethodno su testirane performanse sustava prilikom rada pod velikim opterećenjem (tablice 10.1 i 10.2) i prilikom rada pod optimalnim opterećenjem(tablice 10.3 i 10.4), no kako bi se preciznije odredili uvjeti u kojima sustav optimalno radi provedeno je testiranje u kojem se postepeno povećavao interval između slanja naredbi. Početni interval iznosi 10 milisekundi, te se povećava do 100 milisekundi s korakom od 10 milisekundi. Tablica 10.5 prikazuje slanje jedne, tablica 10.6 dvije, a tablica 10.7 pet naredbi u različitim vremenskim intervalima koristeći CTO naredbe (2. i 3. stupac u tablicama 10.5, 10.6 i 10.7) i DAQ metodu (4. i 5. stupac u tablicama 10.5, 10.6 i 10.7). Uočljivo je da se povećanjem vremenskog intervala između slanja naredbi smanjuje opterećenost sabirnice. Najmanja opterećenost u intervalu od 10 milisekundi vidljiva je u tablici 10.5 zato što se koristila samo jedna naredba za testiranje. Povećanjem vremenskog intervala s 10 na 30 milisekundi smanjila se opterećenost CAN1 sabirnice sa 20.07% na 7.01% i CAN2 sabirnice sa 19.67% na 5.07%, tj. CAN2 sabirnice sa 17.98% na 9.07% korištenjem DAQ metode. Značajni pad opterećenja sabirnice prilikom slanja 2 naredbe povećanjem vremenskog intervala s 10 na 30 milisekundi vidljiv je u tablici 10.6. gdje se opterećenost sabirnice CAN1 smanjila sa 42.85% na 11.03%, a CAN2 sabirnice sa 39.12% na 9.80%. Opterećenost CAN2 sabirnice smanjila se

sa 34.53% na 17.03% prilikom korištenja DAQ metode. Tablica 10.7 prikazuje opterećenja CAN1 i CAN2 sabirnica prilikom slanja 5 naredbi u različitim vremenskim intervalima. U tablici 10.7 vidljivo je da je opterećenje sustava vrlo visoko, tj. opterećenje sabirnice CAN1 iznosi 100% prilikom korištenja vremenskog intervala od 10 milisekundi što znači da se ne može poslati niti jedna naredba više nego što se trenutno šalje. Opterećenja CAN1 i CAN2 sabirnica smanjila su se do otprilike 20% tek nakon što se povećao vremenski interval na 50 milisekundi. Tablica 10.7 najbolje ukazuje na performanse sustava jer za razliku od tablica 10.5 i 10.6, koje pokazuju opterećenost sabirnica prilikom slanja jedne ili dvije naredbe, tablica 10.7 pokazuje opterećenost sabirnica prilikom slanja 5 naredbi u različitim vremenskim intervalima, a prilikom izvođenja mjerenja najčešće se prati veći broj varijabli. Na slici 10.36 je grafički prikaz opterećenosti CAN sabirnica u različitim vremenskim intervalima prilikom slanja jedne naredbe, sa i bez korištenja DAQ metode, a prikazani podatci nalaze se u tablici 10.5. Na slikama 10.37 i 10.38 nalaze se grafički prikazi opterećenosti CAN sabirnica prilikom slanja dvije, odnosno pet naredbi u različitim vremenskim intervalima. Podatci prikazani na slici 10.37 nalaze se u tablici 10.6, a podatci prikazani na slici 10.38 nalaze se u tablici 10.7.

11.ZAKLJUČAK

U sklopu ovog diplomskog rada bilo je potrebno implementirati XCP protokol preko CAN protokola na AURIX platformi korištenjem C programskog jezika. Radi same validacije implementiranog rješenja bilo je potrebno koristiti neki alat za mjerenje, a u ovom slučaju to je bio CANoe alat. Prije same izrade rješenja bilo je potrebno detaljno proučiti način rada CAN protokola, a nakon toga i samog XCP protokola. Kako je XCP fleksibilan protokol, nije obavezna implementacija svih grupa naredbi XCP protokola, već je moguće implementirati samo potrebne grupe naredbi. Implementirane su naredbe iz sljedećih grupa: standardna funkcije, funkcije za kalibraciju i funkcije za mjerenje korištenjem dinamičkih DAQ lista. Implementacija standardnih funkcija i funkcija za kalibraciju bila je relativno jednostavna jer nije zahtijevala puno više od zapisivanja ili čitanja vrijednosti s raznih adresa, dohvaćanja stanja određenih bitova i slično. Implementacija funkcija za mjerenje korištenjem dinamičkih DAQ lista bila je kompleksnija jer je zahtijevala dobro poznavanje programskog jezika i izbor odgovarajuće strukture podataka za realizaciju istih. Verifikacija je provedena korištenjem CANoe programa putem kojeg su se testirale performanse i funkcionalnost sustava. Za testiranje funkcionalnosti sustava slale su se razne naredbe koristeći CAN poruke kako bi se provjerila ispravnost implementiranog protokola. Performanse sustava testirane su mijenjanjem uvjeta u kojima sustav radi, tj. mijenjao se vremenski interval između slanja naredbi. Zaključeno je da prilikom korištenja vremenskog intervala od jedne milisekunde sustav vrlo brzo dolazi do stanja maksimalne opterećenosti, dok se korištenjem intervala od 100 milisekundi sustav nalazi u optimalnom stanju. Svi podatci primljeni koristeći CANoe program bili su uspoređivani s podatcima koji se nalazili na razvojnoj ploči, za vrijeme izvođenja programa, koristeći *lauterbach* uređaj za otklanjanje pogrešaka. Uspoređivanjem podataka dobivenih putem CANoe programa i *lauterbach* uređaja zaključeno je da implementirane funkcije XCP protokola rade ispravno, a time je ispunjen zadatak diplomskog rada.

LITERATURA

[1] Software Debugging over XCP: Effectively Debugging ECUs in the Field https://www.asam.net/index.php?eID=dumpFile&t=f&f=1786&token=d7614e0640dffb78c35d45a85fada99823ae55e6 , 12.08.2019

[2] Infineon (2017), AURIX™ Family – TC27xT: https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc2xx/aurix-family-tc27xt/ , 12.08.2019.

[3] Andreas Patzer, Rainer Zaiser: XCP – The Standard Protocol for ECU Development: https://assets.vector.com/cms/content/application-areas/ecu-calibration/xcp/XCP_ReferenceBook_V3.0_EN.pdf , 18.08.2019

[4] Universal Measurement and Calibration Protocol Family - Protocol Layer Specification: http://read.pudn.com/downloads192/doc/comm/903802/XCP%20-Part%202-%20Protocol%20Layer%20Specification%20-1.0.pdf, 12.08.2019.

[5] Power Debug Interface USB 3: https://www.lauterbach.com/frames.html?powerdebugusb3.html, 12.08.2019

[6] VECTOR E-learning: CAN https://elearning.vector.com/index.php?wbt_ls_kapitel_id=1329975&root=378422&seit=vl_ can_introduction_en, 12.08.2019.

[7] Measurement and Calibration Protocol XCP – Fundamentals : https://www.vector.com/int/en/know-how/technologies/protocols/xcp-measurement-and-calibration-protocol/ , 12.08.2019.

[8] Kvaser CAN E-learning: https://www.kvaser.com/e-learning/ , 12.08.2019

[9] CCP/XCP: https://www.kvaser.com/about-can/higher-layer-protocols/ccpxcp/ , 12.08.2019.

[10] Kvaser CAN Protocol Tutorial: https://www.kvaser.com/course/can-protocol-tutorial/, 12.08.2019.

[11] A CAN Protocol for Calibration and Measurement Data Acquisition: http://read.pudn.com/downloads559/sourcecode/embedded/2302871/CCP(Vector)/DOC/CCPINTRO.PDF , 12.08.2019.

[12] S. Corrigan (2002): Introducion to the Controller Area Network (CAN)

[13] CAN training : https://vector-academy.com/vi_class_can_fundamentals_en.html, 12.08.2019

SAŽETAK

Tema ovog diplomskog rada je implementacija XCP protokola preko CAN protokola na AURIX platformi korištenjem C programskog jezika. XCP protokol zbog svoje fleksibilnosti ne zahtijeva implementaciju svih mogućih naredbi, već samo onih koje su potrebne te su zato implementirane naredbe iz sljedećih grupa: standardna funkcije, funkcije za kalibraciju i funkcije za mjerenje korištenjem dinamičkih DAQ lista. Standardne naredbe i naredbe za kalibraciju implementirane su na manje kompleksan način za razliku od DAQ lista. Zbog što boljih performansi, i jednostavnosti pristupa, podatci DAQ lista su realizirani korištenjem povezanih popisa. Prilikom razvoja algoritma korišten je *lauterbach* uređaj za uklanjanje pogrešaka i CANoe program za testiranje implementiranih funkcionalnosti. CANoe alat za mjerenje korišten je za slanje CAN poruka, te za validaciju implementiranog protokola i vizualizaciju prikupljenih podataka. Rezultati mjerenja su pokazali da modul funkcionira kako je i očekivano, te da je moguće slati i više od 25 naredbi u vremenskom intervalu od 100 milisekundi uz vrlo nisku opterećenost CAN sabirnica što znači da se u tim uvjetima sustav nalazi u optimalnom stanju.

Ključne riječi: CAN protokol, XCP protokol, CANoe program, DAQ liste

IMPLEMENTATION OF THE XCP PROTOCOL THROUGH THE CAN PROTOCOL ON THE AURIX PLATFORM

ABSTRACT

The theme of this graduate thesis is the implementation of the XCP protocol over the CAN protocol on the AURIX platform, using the C programming language. Because of its flexibility, the XCP protocol does not require all the possible commands to be implemented, but only the ones that are needed, and therefore the commands from the following groups are implemented: standard functions, calibration functions, and measurement functions using dynamic DAQ lists. Standard commands and calibration commands are implemented in a less complex way unlike the DAQ list. Because of better performance and simplicity, the DAQ data is stored using linked lists. During the development of the algorithm, the lauterbach debugging device and the CANoe program were used. The CANoe measuring tool was used to send the CAN messages, validate the implemented protocol and visualize the collected data. The measurement results showed that the module was working as expected, and that it was possible to send more than 25 commands in a 100 millisecond time interval with a very low CAN bus load, which means that under these conditions the system was in optimal state.

Key words: CAN protocol, XCP protocol, CANoe software, DAQ lists

ŽIVOTOPIS

Dino Pečurlić je rođen 20. studenog 1995. godine u Osijeku. Završio je osnovnu školu Retfala u Osijeku, nakon čega upisuje Opću gimnaziju u Osijeku. Tijekom srednje škole sudjelovao je na natjecanjima iz informatike. 2014. godine, nakon maturiranja s vrlo dobrim uspjehom, upisuje Elektrotehnički fakultet u Osijeku, smjer računarstvo. Završetkom preddiplomskog studija računarstva 2017. godine stekao je zvanje: prvostupnik (baccalareus) inžinjer računarstva. 2017. godine upisuje diplomski studij računarstva, smjer programskog inžinjerstva, i postaje stipendist tvrtke Institut RT-RK.