CAN Packet Sniffer

Josip Nigojević

Petar Kaselj

Sadržaj

Uvod

Naglim razvojem tehnologije, elektronika postaje sve više i više zastupljena u našim svakodnevnim životima, od rasvjete preko kućanskih aparata pa sve do samih zgrada se proizvodi i/ili planira imajući na umu sve sofisticiranije tehnologije kao što su senzori i upravljači. Kako takvi uređaji postaju sve napredniji, precizniji i brži, potrebno je osigurati i zadovoljavajuću komunikacijsku infrastrukturu na kojoj su izgrađeni.

Jedan od takvih sustava čine i osobna vozila. Posebnost osobnih vozila je u tome što komunikacijsku infrastrukturu tjeraju do granica iskorištenosti s obzirom da računalni sustavi u vozilima moraju biti što brži i pouzdaniji da bi se spriječila velika materijalna ali i u nekim slučajevima ljudska šteta. Da bi se projektirali ovakvi sustavi realiziran je jedan od najraširenijih i najrobusnijih automobilskih komunikacijskih infrastruktura koji se temelji na *Bosch*-ovom CAN protokolu.

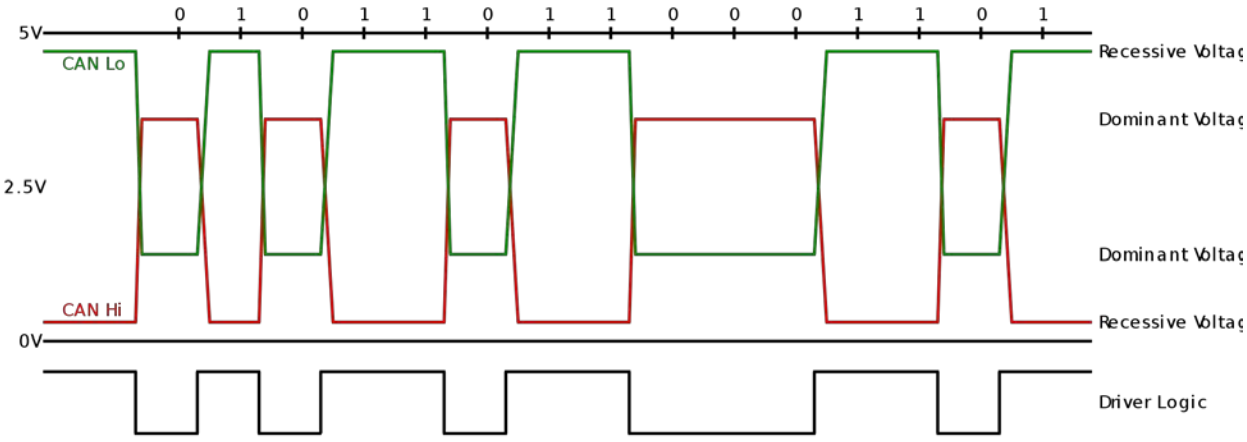
CAN protokol je danas jedan on najraširenijih komunikacijskih protokola svoje razine uz Ethernet, te je našao primjene i izvan automobilske industrije.

CAN protokol

CAN protokol komunikacijski protokol osmišljen kako bi mikrokontroleri i uređaji mogli međusobno komunicirati bez glavnog računala.Princip rada se zasniva na 2 žice(CAN high i CAN low) koje se nalaze na istom potencijalu kada nema prijenosa podataka.Prilikom prijenosa jedna se nalazi na višoj naponskoj razini a druga na nižoj te između njih postoji razlika potencijala.Okvire podataka primaju svi uređaji u mreži uključujući predajnički CAN uređaj.

Podaci iz okvira se prenose sekvencijalno al na način da ako više uređaja u isto vrijeme emitira podatke,tada ih šalje samo onaj sa najvećim prioritetom.

(Herres, 2017)



Slika 1 Signali na CAN sabirnici

Suvremeni automobil može imati čak 70 elektroničkih upravljačkih jedinica (ECU) za razne podsustave.Tipično najveći ECU upravljačka jedinica motora. Ostali se koriste za mjenjač, zračne jastuke, ABS, tempomat, električnu snagu upravljanje, audio sustavi, vrata, podešavanje ogledala, sustavi baterija i punjenja za hibridne / električne automobile itd.

Neki od njih čine neovisne podsustave, ali komunikacija između ostalih je ključna. Podsustav će možda trebati upravljati pogonima ili primati povratne informacije, CAN standard osmišljen je kako bi ispunio tu potrebu. Jedna od ključnih prednosti je što međusobno povezivanje različitih sustava vozila može omogućiti da se širok raspon sigurnosnih, ekonomičnih i praktičnih značajki implementira samo pomoću softverske funkcionalnosti koja bi dodala trošak i složenost da su izvedene na klasični hardverski način.

Primjeri korištenja CAN protkola

Senzori se mogu postaviti na najprikladnije mjesto, a njihove podatke koristi nekoliko ECU-a. Na primjer, senzori vanjske temperature (tradicionalno postavljeni sprijeda) mogu se postaviti u vanjska ogledala, izbjegavajući zagrijavanje motora, a podatke koristi motor, kontrola klime i zaslon vozača.

- Automatsko pokretanje/zaustavljanje(start/stop): razni ulazi senzora iz okoline vozila objedinjuju se preko CAN sabirnice kako bi se utvrdilo može li se motor isključiti kada miruje radi poboljšane potrošnje goriva i emisije.

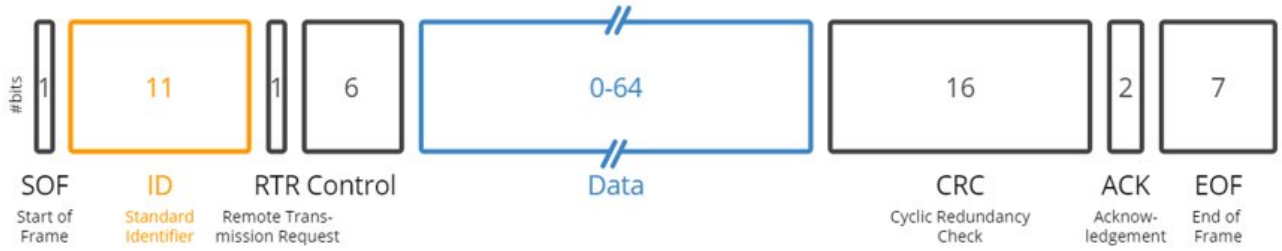
- Električne parkirne kočnice: Funkcija "zadržavanja na uzbrdici" uzima ulaz putem senzora za nagib vozila (također ga koristi protuprovalni alarm) i senzora brzine na cesti (također ih koriste ABS, kontrola motora i kontrola proklizavanja) putem CAN sabirnice kako bi utvrdila je li vozilo se zaustavlja na nagibu. Slično tome, ulazi senzora sigurnosnih pojaseva (dio kontrola zračnog jastuka) napajaju se iz CAN sabirnice kako bi se utvrdilo jesu li sigurnosni pojasevi pričvršćeni, tako da će se ručna kočnica automatski otpustiti pri kretanju.

- Sustavi za automatsku pomoć u izbjegavanju sudara / izbjegavanja sudara: ulaze senzora za parkiranje također koristi CAN sabirnica za unos podataka iz neposredne blizine u sustave za pomoć vozaču kao što je upozorenje na odlazak s trake, od nedavno ti signali putuju kroz CAN sabirnicu za pokretanje kočnice žicom u aktivnim sustavima za izbjegavanje sudara

CAN okvir

CAN okvir emitira poruku, stvarne podatke, na CAN sabirnicu, bilo zbog promjene događaja (na primjer, promjene ulaznog signala, vremenskog događaja itd.) Ili kao odgovor na zahtjev za porukom. Okvir podataka, identificiran jedinstvenim ID-jem poruke, može prihvatiti bilo koji broj čvorova u mreži u skladu s individualnim potrebama aplikacije, ali ga može prenijeti samo (jedan i jedini) čvor povezan s podatkovnom porukom.

(CSS Electronics, 2021)



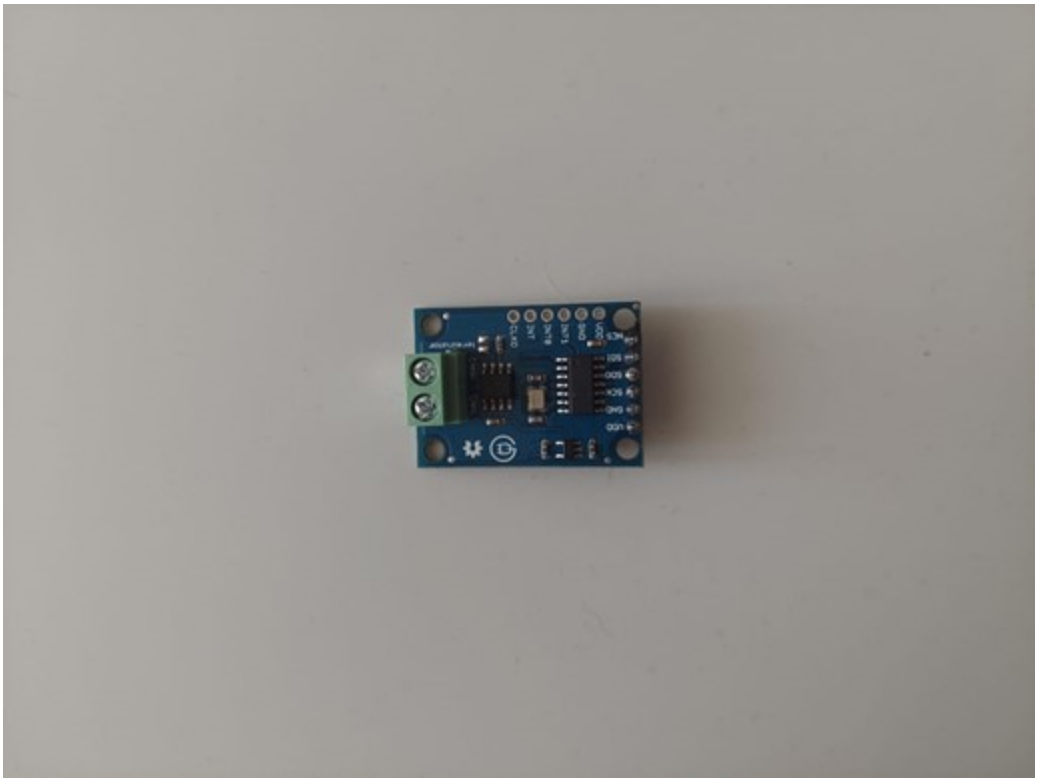
Slika 2 Prikaz dijelova CAN paketa

- SOF: Početak okvira je „dominantna 0“ da bi se ostalim čvorovima reklo da CAN čvor namjerava slati poruke
- ID: ID je identifikator okvira - niže vrijednosti imaju veći prioritet, moraju biti jedinstvene
- RTR: Zahtjev za daljinski prijenos ukazuje na to šalje li čvor podatke ili zahtijeva posebne podatke s drugog čvora
- Control: Kontrola sadrži identifikacijski produžni bit (IDE-identifier extension bit) koji je za 11-bitni „dominantna 0“. Sadrži i 4-bitni kod duljine podataka (DLC) koji određuje duljinu bajtova podataka koji se prenose (0 do 8 bajtova)
- Data: Data sadrže bajtove podataka, odnosno korisni teret, koji uključuje CAN signale koji se mogu izvući i dekodirati radi informacija
- CRC: Koristi se kako bi se osigurala cjelovitost podataka
- ACK: pokazuje je li čvor ispravno priznao i primio podatke
- EOF: označava kraj CAN okvira

CAN *breakout*

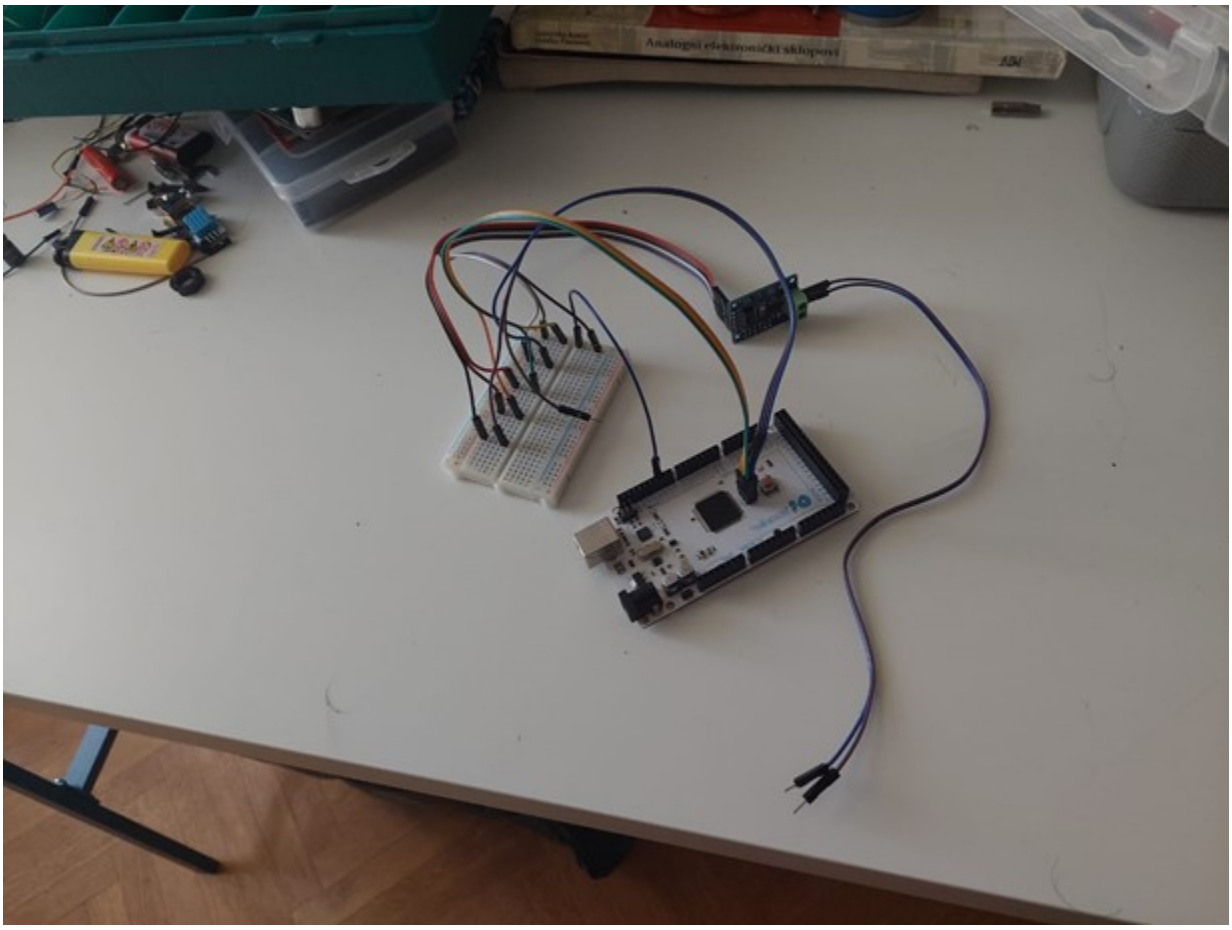
U ovom radu je za hardverski dio rješenja korištena *Arduino Mega* 2560 pločicu,

no zbog određenih ograničenja, da bih omogućili primanje i obradu CAN poruka potrebno je bilo koristiti CAN *breakout*. CAN *breakout* koristi čipove MCP2515 kao CAN-BUS kontroler i MCP2551 kao CAN primopredajnik.



Slika 3 CAN breakout

CAN *breakout* je sučelje koje omogućava arduinu da prima CAN poruke preko SPI(serial peripheral interface), spaja se na SPI (SCK, MISO, MOSI...) pinove arduina kao na slici ispod.



Slika 4 Način spajanja CAN breakout-a na Arduino

CAN *Arduino* biblioteka

Za korištenje ovog shielda napisana je posebna biblioteka“arduino-CAN” koju pozivamo u okruženje pomoću `#include <CAN.h>` . Ova bibilioteka sadrži gotove funkcije pomoću kojih nam je omogućeno lakše korištenje funkcionalnosti shielda s *Arduino*-m.

Neki primjeri su:

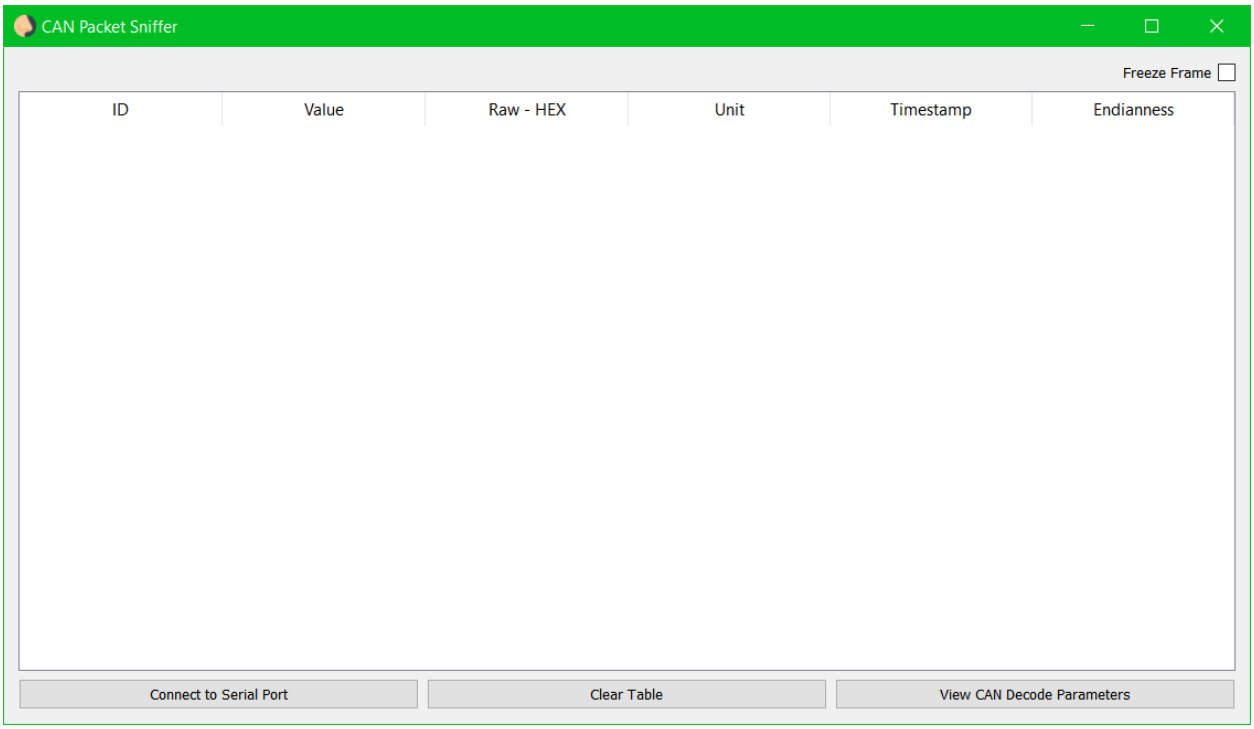
- `CAN.parsePacket()` - Vraća veličinu paketa u bajtovima ili 0 ako nije primljen paket.
- `CAN.packetId()` - Vraća ID (11-bitni ili 29-bitni) primljenog paketa. Standardni paketi imaju 11-bitni ID dok *extended* paketi imaju 29-bitni ID.
- `CAN.available()` - Vraća broj bajtova koji su dostupni za čitanje.
- `CAN.read()` - Vraća sljedeći bajt u paketu ili -1 ako nema dostupnih bajtova.
- `CAN.loopback()` - Postavi CAN kontroler u loopback mode odnosno primit će se i svi odlazni paketi.

Aplikacija "CAN Packet Sniffer"

Za potrebe prikazivanja presretenih CAN paketa sa CAN sabirnice, razvijena je aplikacija *CAN Packet Sniffer* napisana u C++ programskom jeziku koristeći Qt razvojni okvir. Aplikacija na minimalistički i *user-friendly* način omogućava jednostavnu obradu i prikazivanje CAN paketa na računalu.

Slika 5 Prečac na radnoj površini

Na slici Slika 3 može se vidjeti početni zaslon vidljiv odmah pri pokretanu aplikacije.

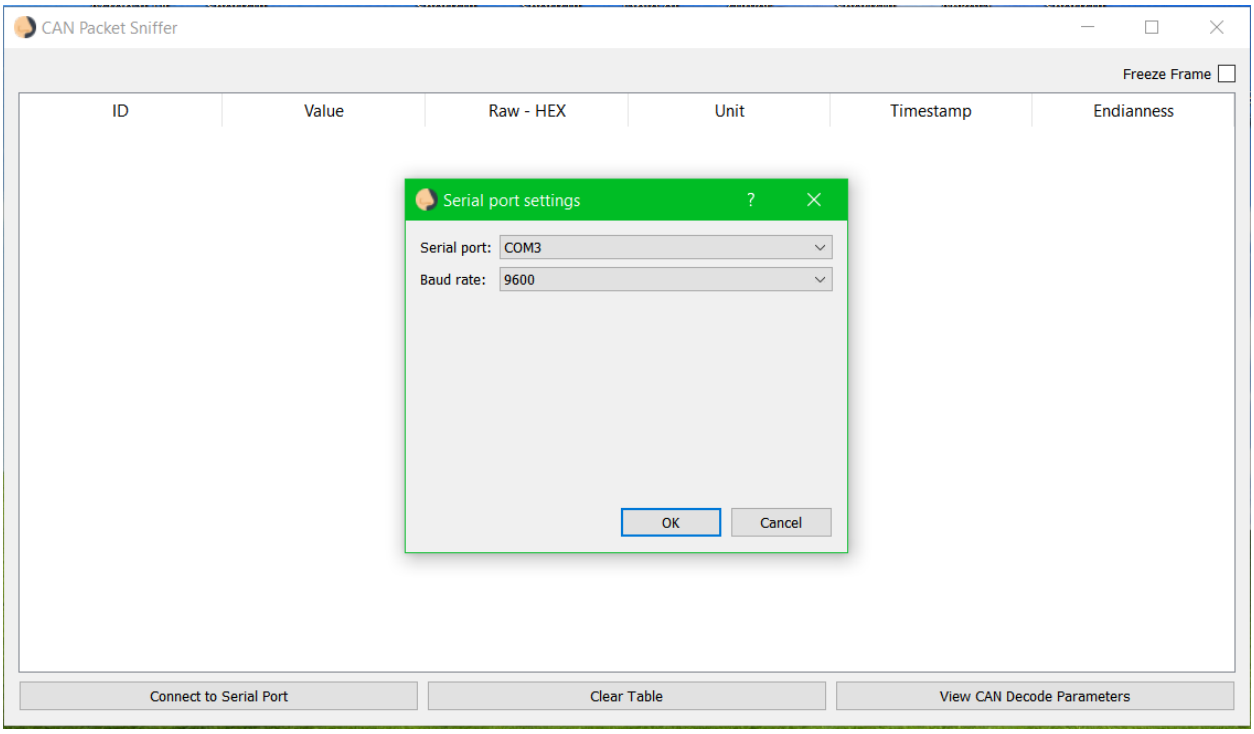


Slika 6 Početni zaslon aplikacije

Spajanje s *Arduino* pločicom

Prvi korak pri postavljanju aplikacije bi bio spajanje na *Arudino* razvojnu pločicu preko serijskog porta.

Da bi se to izvelo potrebno je pritisnuti gumb "Connect to Serial Port" koji pokreće novi dijalog prikazan na Slici 3.



Slika 7 Dijalog spajanja na serijski port

Pri pokretanju dijalog pruža izbor slobodnih serijskih portova, u ovom slučaju COM3, i brzina prijenosa podataka *Baud rate* koji je postavljen na početnu vrijednost 9600 Bauda.

Komunikacija *Arduino* – računalo

U ovom poglavlju biti će ukratko predstavljeni tehnički podaci vezani za serijsku vezu između *Arudino* razvojne pločice i računala.

Serijska komunikacija se odvija brzinom 9600 Bauda, bez kontrole toka, bez paritetnih bitova, s 8 podatkovnih bitova i 1 STOP bitom.

Većina podataka je ugrađena (engl. *hardcoded*) u izvorne kodove programske podrške te krajnji korisnik o njima ne treba brinuti. Ono na što krajnji korisnik mora paziti jesu serijski port na kojem će se odvijati komunikacija i brzina prijenosa. Neusklađenost navedenih parametara na prijemnoj (računalo) i odašiljačkoj (*Arduino*) strani skoro sigurno će rezultirati krivim očitanjima, ako očitanja uopće bude.

Sinkronizacija serijske veze

Potreba za `uint8_t sync_sequence` proizlazi iz jednostavne (asinkrone) prirode serijske komunikacije. Kako je serijski protokol asinkron, te se u ovom projektu koristi bez neke nadogradnje javljaju se pojedini problemi.

Ako pretpostavimo da *Arudino* preko serijske veze šalje paket definiran nizom bajtova (simbolički prikaz, ne prikazuje vjerno strukturu korištenog paketa):

0x11	0x00	0x00	0x00	0x12	0x13	0x14	0x15	0x20	0x21
ID				VALUE					

Veličina paketa iznosi 10 bajtova.

Kako se podaci šalju bajt po bajt (v. Komunikacija *Arduino* – računalo), a računalo i *Arduino* prethodno se ne sinkroniziraju, moguće je da računalo počne primati bajtove od sredine nekog od paketa pa se dogodi sljedeća situacija na prijemnoj strani (N.B. u primjeru se pretpostavlja da *Arduino* šalje jedan te isti paket u beskonačnoj petlji, bez prekida):

0x12	0x13	0x14	0x15	0x20	0x21	0x11	0x00	0x00	0x00
ID (računalo)				VALUE (računalo)					

Odnosno računalo krivo pročita poslani paket, što uzrokuje lančanu reakciju odn. računalo krivo čita i svaki paket nakon njega jer uzima idućih 10 bajtova, bez provjere ispravnosti paketa.

Da bi se to izbjeglo implementiran je jednostavni protokol tako da odašiljač prethodi paket sa sinkronizirajućom sekvencom (naprimjer bajt označen sa SS):

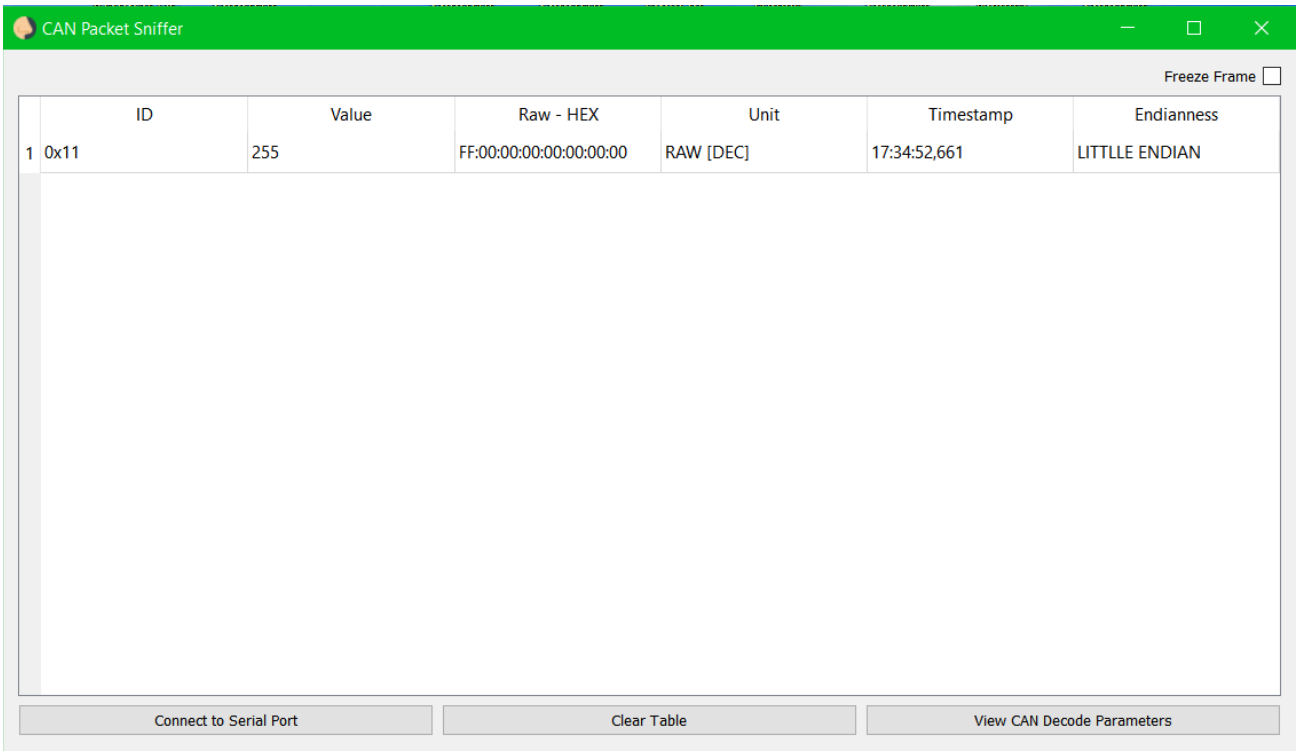
SS	0x11	0x00	0x00	0x00	0x12	0x13	0x14	0x15	0x20	0x21
	ID				VALUE					

dok računalo na prijemnoj strani odbacuje sve primljenje podatke dok ne primi sinkronizirajuću sekevncu SS, nakon koje pročita 10 bajtova (širina paketa):

XX	XX	SS	0x11	0x00	0x00	0x00	0x12	0x13	0x14	0x15	0x20	0x21
Odbačeno			ID				VALUE					

Grafički prikaz CAN paketa

Nakon uspješne uspostave veze, aplikacija počinje ispisivati primljene pakete u sljedećem obliku:

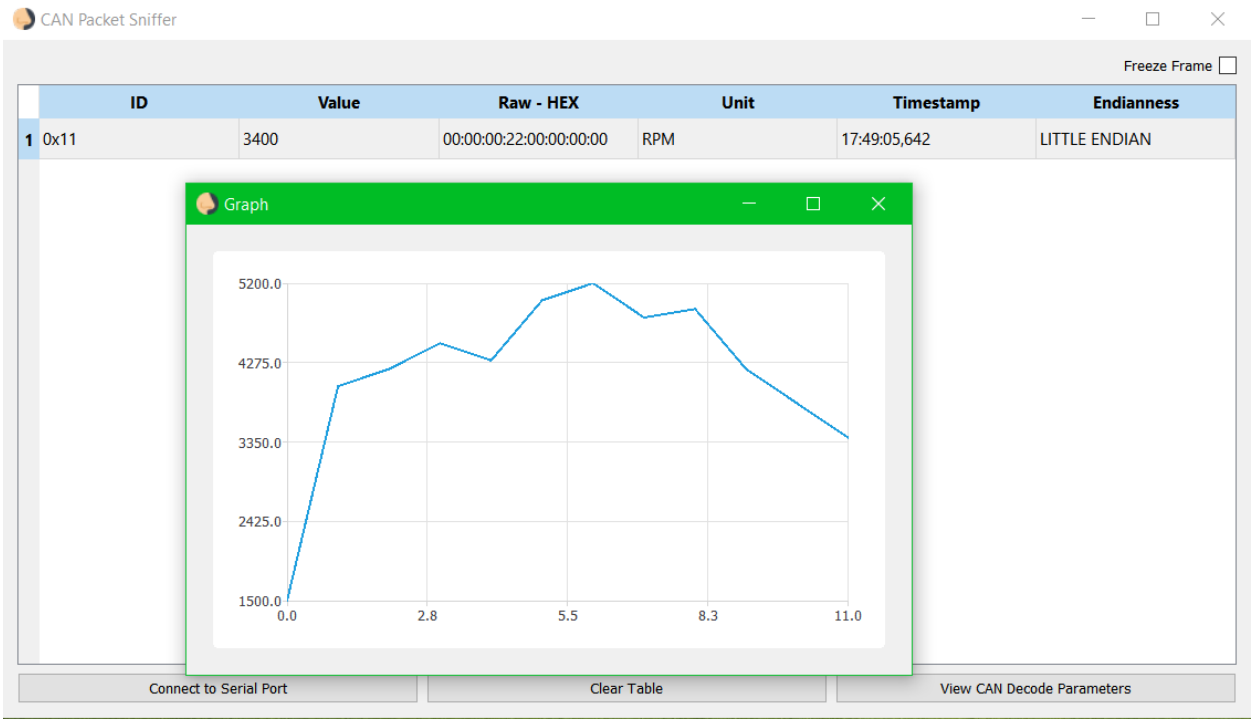


Slika 8 Tablica primljenih CAN paketa

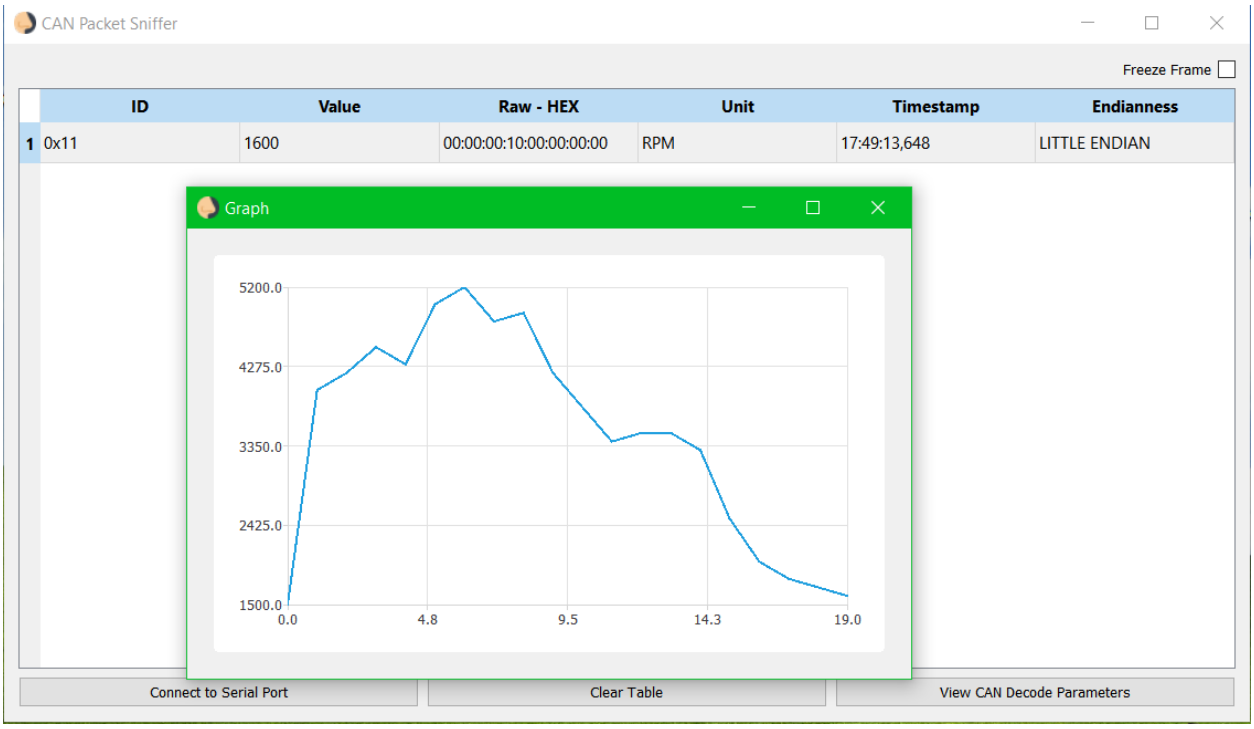
Gdje *Value* predstavlja dekadsku vrijednost polja *Value* CAN paketa (prikazan u HEX obliku pod Raw - HEX), *Unit* predstavlja jedinicu mjere, a *Endianness* način na koji aplikacija tumači heksadekadski zapis polja *Value* primljenog CAN paketa.

U slučaju primitka nepoznatog CAN paketa, aplikacija dekodira svih 8 bajtova u polje *Value* te ih tretira kao *Little endian* podatke.

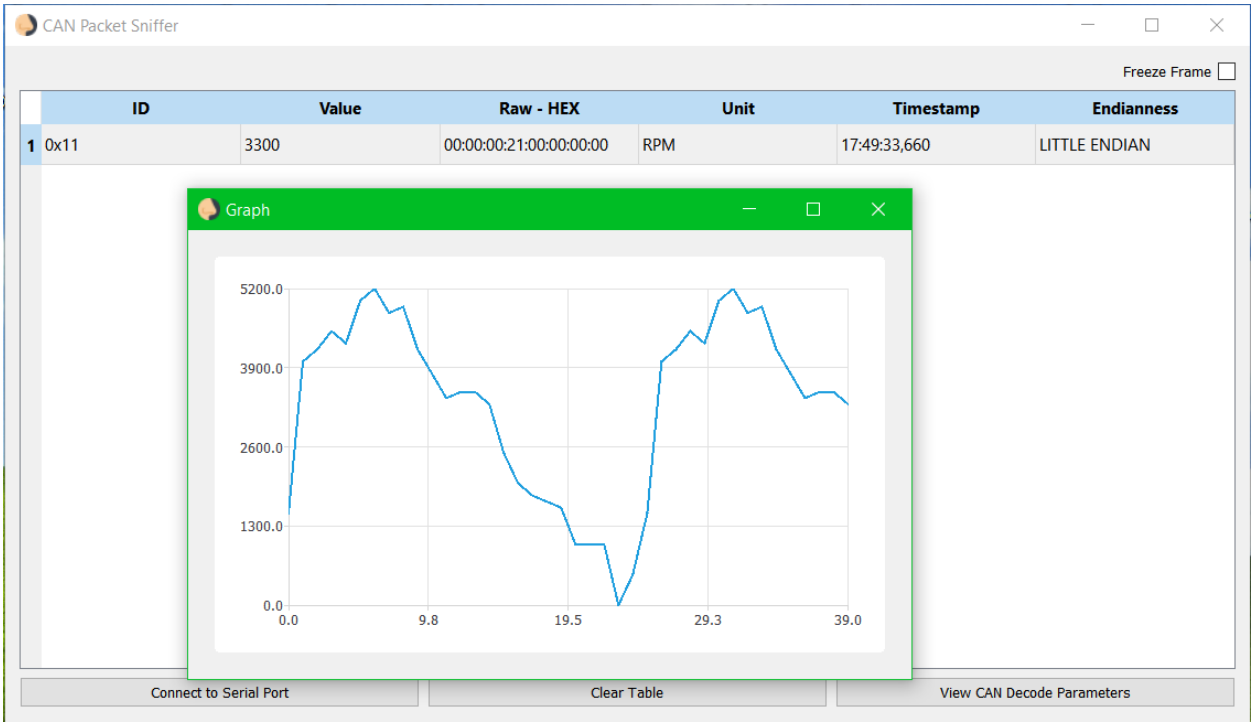
Dvostrukim klikom na polje *ID* nekog zapisa otvara se grafički prikaz vrijednosti *Value* odabranog ID-ja u realnom vremenu od zadnjih nekoliko očitanja, gdje vrijednost na x osi definira redni broj uzorka (ukupna veličina spremnika grafičkog prikaza definirana je u izvornom kodu kao 100 uzoraka v. varijabla *BufferSize*), dok vrijednost na y osi predstavlja vrijednost *Value* u mjernoj jedinici *Unit* (prema standardnim postavkama *RAW [DEC]* je bezdimenzionalna jedinica).



Slika 9 Grafički prikaz broja okretaja motora (RPM) - 1



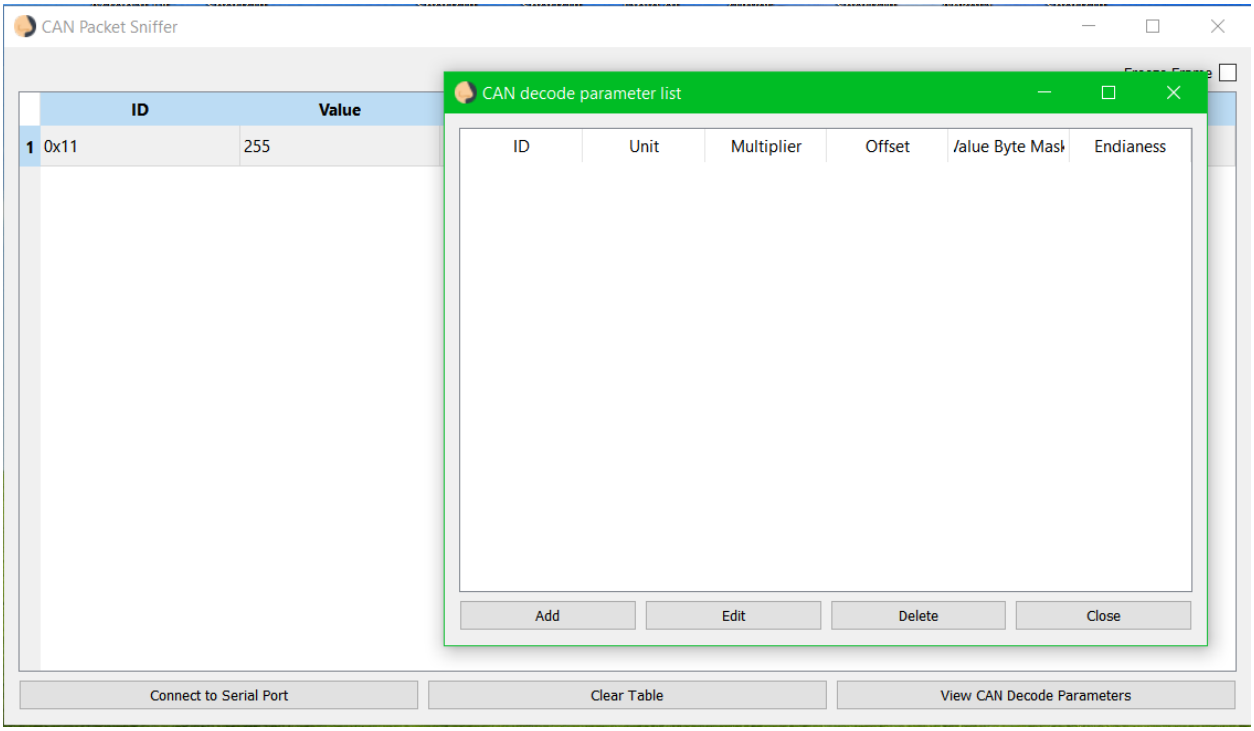
Slika 10 Grafički prikaz broja okretaja motora (RPM) – 2



Slika 11 Grafički prikaz broja okretaja motora (RPM) - 3

Direktive za dekodiranje CAN paketa

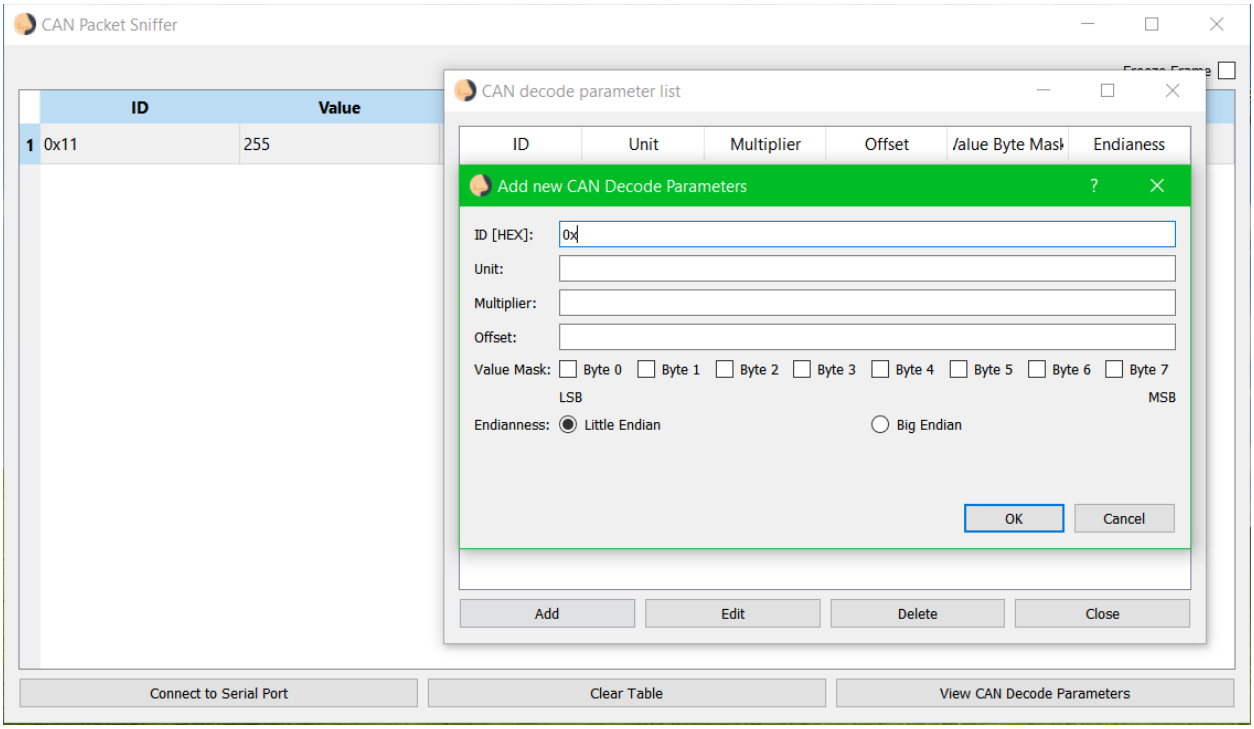
Da bi se podaci prikazivali u jedinicama čitljivim korisniku potrebno je dodati direktive za prevođenje CAN paketa pritiskom na gumb *View CAN Decode Parameters* čime se otvara sljedeći prozor:



Slika 12 Prikaz svih direktiva za prevođenje CAN paketa

U ovom prozoru mogu se vidjeti sve definirane direktive za prevođenje CAN paketa.

Da bi se dodala direktiva potrebno je pritisnuti gumb *Add* koji otvara sljedeći dijalog:



Slika 13 Dijalog za dodavanje CAN direktive

U kojem se definira heksadekadska vrijednost polja *ID* CAN paketa za koje se želi dodati direktiva.

Direktiva definira pravila prema kojima se *Value* polje CAN paketa prevodi u broj s jedinicom *Unit* tako da se unesu dekadske vrijednost parametara *Offset* i *Multiplier,* te se odaberu koji bajtovi će biti uzeti u obzir pri računanju vrijednosti *Value*.

Aplikacija pri prevođenju *Value* polja CAN paketa u *Value* polje u tablici koristi sljedeću formulu:

$$Value_{Tablica} = Value_{CANPaket} \cdot Multiplier + Offset$$

Value(CAN paket) dobije se tako da se uzme polje *Value* primljenog CAN paketa. Polja koja nisu u direktivi označena kvačicom (v. *Value mask* na Slici 9) se ignoriraju tj. postavljaju se u 0x00. Za najmanje značajan bajt uzima se onaj koji je prvi označen pod *Value mask* s kvačicom gledajući iz smjera LSB (ovisno o načinu zapisa tj. *Little endian / Big endian*).

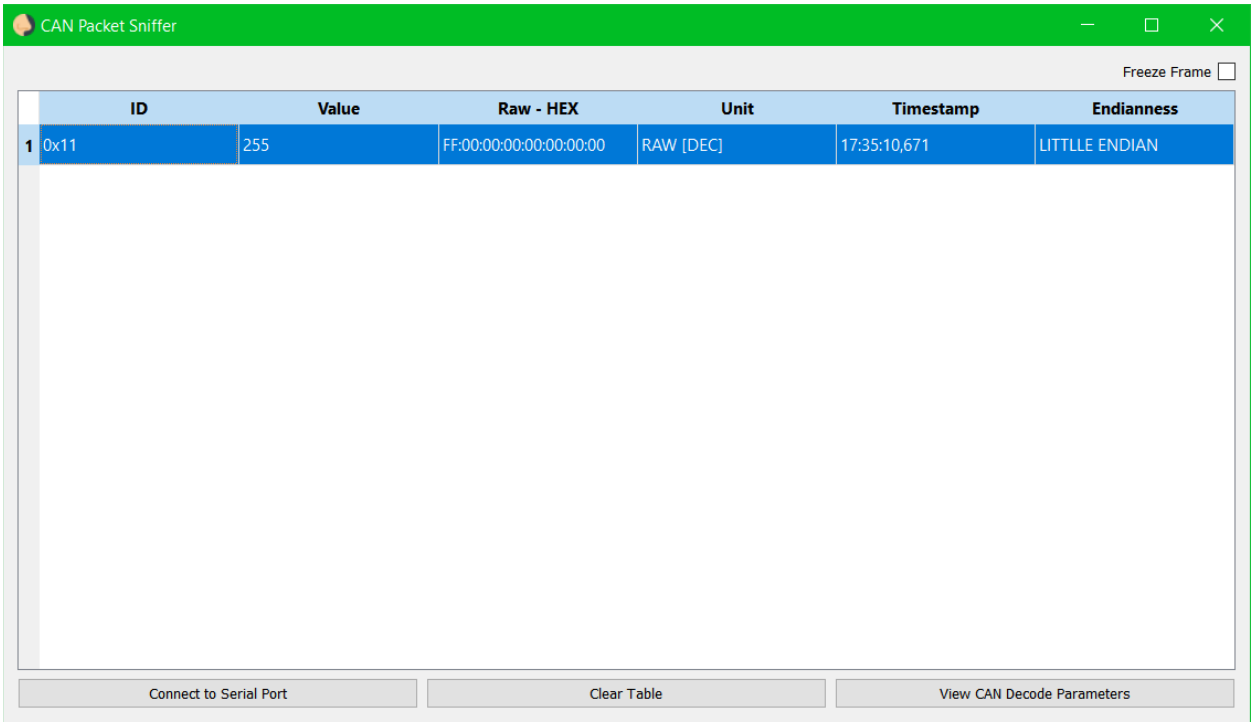
Primjer:

Ako računalo primi CAN paket sa sljedećom vrijednosti *Value* polja, ako je u aplikaciji postavljen *Value mask* za primljeni paket kao u tablici tada se dobije:

Value(CAN Paket)	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF
Value Mask	+	-	-	-	-	-	-	-
Rezultat	0xFF	0x00	0x00	0x00	0x00	0x00	0x00	0x00

U sljedećim proračunima uzima se vrijednost "Rezultat", ovisno o postavci *Endianness* on se tumači kao

0x00000000000000FF za *Endianess* postavljen kao *Little endian* (Kao na slici 10) odnosno 0xFF00000000000000 za *Big endian.*



Slika 14 Primljeni paket uz postavljen Value mask

Aplikacija podržava i brisanje i uređivanje već postojećih CAN direktiva u prozoru *View CAN Decode Parameters*

Popis slika

Reference

CSS Electronics. (2021). Retrieved from CSS Electronics: https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en

Herres, D. (2017, January 31). *Test and Measurement Tips*. Retrieved from Test and Measurement Tips: https://www.testandmeasurementtips.com/exploring-canbus-oscilloscope/