VAN protokol i njegova primjena u automobilima

Milinković, Mislav

Sveučilište Josipa Jurja Strossmayera u Osijeku

Fakultet primijenjene matematike i informatike

Sveučilišni prijediplomski studij Matematika i računarstvo

VAN protokol i njegova primjena u automobilima

Završni rad

Mentor:

izv. prof. dr. sc. Domagoj Matijević

Komentor:

Jurica Maltar

Kandidat:

Mislav Milinković

Osijek, 2024

Sadržaj

1 | Uvod

1.1 Uvod u rad i motivacija

"Kako radio zna kojom brzinom automobil vozi?", "Kako ekran i radio mogu biti odvojeni?", "Kako ekran funkcionira?", i "Mogu li mijenjati što piše na ekranu?" samo su neka od pitanja koja su služila kao motivacija za istraživanje komunikacije u Peugeot 206 automobilu.

Peugeot 206 koristi VAN protokol za komunikaciju između upravljačkih jedinica. To je protokol koji je korišten u ograničenom broju Peugeot i Citroën automobila, te je jako slabo dokumentiran. Jedna od motivacija za pisanje ovog rada jest dokumentacija VAN protokola. Sva postojeća dokumentacija je na francuskom jeziku, razbacana po povjerljivim dokumentima. Jedini dokument u kojem je detaljnije objašnjen protokol je datasheet integriranog kruga Atmel TSS463C [6] koji služi za konstruiranje i slanje poruka sa VAN protokolom. Ovaj rad je napravljen kako bi koncentrirao sve tehničke informacije, poznatu povijest protokola i otkrića hobi zajednice u jedan dokument.

Druga motivacija za istraživanje i dokumentiranje protokola jest izrada vlastitog proizvoda. Naime, kako u automobilu već postoji pregršt senzora i računala, ideja je bila napraviti infotainment sustav. Naravno, moguće je kupiti gotov sustav, ali plan je bio integrirati sustav u automobil kao originalni radio, te da komunicira sa originalnim ekranom ili ga čak zamjeni bez gubitka funkcionalnosti.

Glavni izazovi VAN protokol je razvijen u Francuskoj od strane PSA grupe (Peugeot, Citroën) i korišten je u kratkom periodu samo u njihovim automobilima. Zbog toga nije puno poznato o njemu i velika većina dokumentacije koja postoji na internetu je na francuskom jeziku.

Izazovi s VAN protokolom su bili pronalazak te dokumentacije i prevođenje, traženje hardvera koji može komunicirati s VAN protokolom i dekodiranje paketa koji se šalju u automobilu.

Drugi izazovi su sa samim infotainment sustavom. Plan je bio koristiti Raspberry Pi 3 računalo i 7″ ekran na dodir za prikaz sučelja, i koristiti neki mikroupravljač za pristup VAN mreži i slanje informacija na Raspberry Pi. Izazov je također bio softver. Koju Linux distribuciju koristiti? Kako komunicirati sa mikroupravljačem? Kako to sve objediniti?

U ovom dokumentu će biti pokrivene sve bitne informacije o VAN protokolu. Dokument će biti više tehničke naravi jer će služiti kao površna dokumentacija za VAN protokol i infotainment sustav.

1.2 Struktura rada

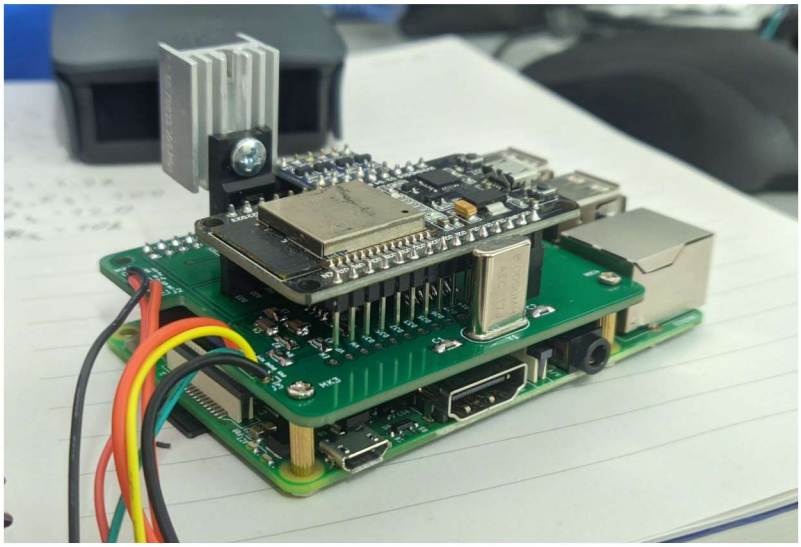
Rad je podijeljen na dva velika poglavlja.

U poglavlju VAN protokol uvodimo VAN protokol i uspoređujemo ga sa CAN protokolom. Prikazujemo oblik VAN poruka, i objašnjavamo sve vrste poruka i kada se koriste.

U poglavlju Peugeot Infotainment Project prikazujemo kako smo izradili infotainment sustav koji koristi VAN protokol kako bi komunicirao s automobilom i prikazao te informacije korisniku. Također šalje poruke kako bi se predstavio kao CD izmjenjivač u automobilu. Koristimo ESP32 kako bismo primali i slali VAN poruke, te ih prosljeđujemo Raspberry Pi računalu pomoću UART sučelja koristeći vlastiti protokol, libPSA. Za Raspberry Pi smo napravili prilagođenu Linux distribuciju pomoću Yocto[1] projekta, te na njemu radi naše grafičko sučelje napravljeno u Qt[2] programskom okviru. Sučelje se prikazuje na 7-inčnom ekranu na dodir koji se spojen sa DSI konektorom. Kako bismo sve objedinili, napravili smo PCB na kojem se nalazi sav potreban hardver za komunikaciju sa automobilom koji se spaja kao HAT na Raspberry Pi.

Slika 1.1: Prikaz sučelja infotainment sustava. Detaljnije u poglavlju Peugeot Infotainment Project.

[1] https://www.yoctoproject.org/
[2] https://www.qt.io/



Slika 1.2: Raspberry Pi, ESP32 i PCB korišteni u izradi projekta. Detaljnije u poglavlju Peugeot Infotainment Project.

Cijeli projekt je otvorenog koda i možete sa sljedećim linkovima pristupiti repozitorijima za sve komponente:

- Repozitorij za grafičku aplikaciju: https://github.com/Mive82/psa-app
- Repozitorij za ESP32 firmware: https://github.com/Mive82/psa_esp32
- Repozitorij za PCB shemu: https://github.com/Mive82/psa-pcb

Sva dokumentacija gore navedenih komponenata, libPSA protokola i uputsva za spajanje su dostupna na sljedećem linku: https://github.com/Mive82/psa-docs.

2 | VAN protokol

2.1 Uvod u komunikacijske protokole

Kada se grupa od dva ili više čvorova međusobno poveže u mrežu kako bi mogli razmjenjivati poruke, takvu mrežu zovemo komunikacijskom mrežom. Kako bi čvorovi mogli nesmetano komunicirati, potrebno je definirati zajednički komunikacijski protokol. Pomoću protokola definiramo na koji način će se poruke prenositi. Možemo definirati medij prenošenja poruka i oblik poruka.

U računalnim sustavima pronalazimo mnoštvo različitih komunikacijskih protokola. Svakodnevno se susrećemo sa internetom. Za rad interneta su zaduženi mnogi protokoli, a neki od najpoznatijih su Ethernet, odnosno „IEEE 802.3“ protokol ili WiFi, odnosno „IEEE 802.11“ protokol. Ta dva protokola opisuju različite medije za slanje podataka - Ethernet šalje podatke bakrenom žicom, dok WiFi šalje podatke pomoću radio valova.

Ovi protokoli doduše ne definiraju podatke koji se šalju. Za oblikovanje poruka zaduženo je mnoštvo drugih protokola, na primjer, IPv4 (Internet Protocol verzije 4) definira adresiranje čvorova na mreži, TCP definira način i oblikovanje podataka za slanje, HTTPS definira vrstu podataka koji se šalju (u ovom slučaju web-stranicu).

Možemo vidjeti da se u svakodnevnoj računalnoj mrežnoj komunikaciji koristi više protokola odjednom: HTTP web stranica je poslana u TCP paketu koji sadrži IP informacije o izvoru i odredištu koji putuju Ethernet kablovima.

Ovakav „sendvič“ model je vrlo čest u kompleksnim sustavima kao što je Internet, dok u manje kompleksnim sustavima nije potreban kompleksan model.

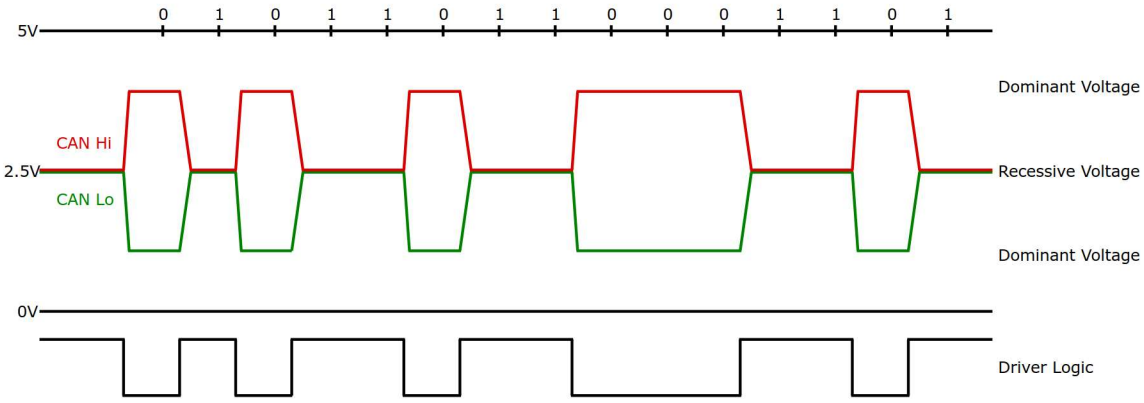
2.2 Uvod u CAN (Controller Area Network)

Originalno osmišljen 1983. u tvrtki Bosch, pušten je u javnost 1986. za korištenje u automobilskim sustavima kako bi se smanjila količina bakra, ali danas se može naći i u raznim industrijskim okruženjima. Dizajniran je za omogućivanje komunikacije između raznih uređaja i mikroupravljača bez glavnog (host) računala i to preko samo jednog para žica.

CAN je serijska sabirnica za povezivanje više elektroničkih upravljačkih jedinica (Electronic Control Unit; ECU) ili čvorova. Čvorovi mogu biti bilo kakvi uređaji od najjednostavnijeg logičkog uređaja, do cijelog računala. Takvo računalo inače služi kao pristup („gateway“) mreži izvana, na primjer za dijagnostiku.

Na sabirnici se koristi jedan par diferencijalnih signala, CAN High ($CANH$) i CAN Low ($CANL$). Signali mogu biti u dva stanja: Dominantnom, gdje je $CANH > CANL$ i recesivnom, gdje je $CANH \leq CANL$. Dominantno stanje se interpretira kao logički 0, a recesivno kao logički 1.

Nominalni naponski raspon CAN sabirnice je između 0 i 5 V. Kada nema komunikacije na mreži, sabirnica se nalazi u recesivnom stanju, odnosno CANH i CANL su na naponima od 2.5 V. U dominantnom stanju, nominalan napon CANH je 3.5 V, a CANL 1.5 V. Razlika u naponu veća od 0.5 V se interpretira kao dominantno stanje. Nominalna razlika za dominantno stanje je 2 V.



Slika 2.1: Prikaz dominantnog i recesivnog stanja na CAN sabirnici. Slika preuzeta sa Wikipedije [2].

Podaci na CAN mreži su organizirani kao okviri (frame) koji se sastoje od početka (Start of Frame; SOF), identifikatora (ID), kontrolnih bitova (Control), samih podataka (Data), provjere (Cyclic Redundancy Check; CRC), potvrdnog (acknowledge; ACK) bita i kraja okvira (End of Frame; EOF). Nakon svakog okvira, mora proći 3 recesivna bita (IFS) prije slanja idućeg okvira.

Početak	1 bit
ID	11 bitova
Control	7 bitova
Data	0 do 8 bajtova (0-64 bita)
CRC	15 bitova + 1 bit
ACK	1 bit + 1 bit
EOF	7 bit
IFS	3 bit

Tablica 2.1: Struktura običnog CAN okvira. Postoji i prošireni CAN okvir, ali to nije tema ovog rada. Slika preuzeta sa Wikipedije [2].

Arbitraža S obzirom da je na jednoj sabirnici povezano više uređaja, postoji mogućnost da neki uređaji šalju podatke u isto vrijeme. Prvo rješenje tog problema je pravilo da uređaj ne smije početi prijenos ako drugi uređaj trenutno prenosi podatke. Drugo je da uređaji moraju čekati barem 3 vremenske jedinice (time slots;

TS), odnosno 3 bita, nakon kraja zadnje poruke prije nego što mogu započeti slanje. Treće rješenje je arbitraža.

Jedini trenutak kada se prijenos sa više uređaja može dogoditi je na početku slanja, stoga ID polje služi kako bi se odredio prioritet. U arbitraži, dominantno stanje, odnosno logička nula, ima prioritet. Dakle, ako uređaj pošalje logičku jedinicu, ali sabirnica je i dalje u dominantnom stanju, taj uređaj prestaje sa slanjem i čeka iduću priliku. Kako manji binarni brojevi imaju više 0, manji identifikatori imaju veći prioritet od većih identifikatora.

	Početni bit	Bitovi identifikatora										Ostatak okvira
		10	9	8	7	6	5	4	3	2	1	0
Uređaj 1	0	0	0	0	0	0	0	0	1	1	0	1	
Uređaj 2	0	0	0	0	0	0	0	1	Prekid prijenosa				
Stanje na sabirnici	0	0	0	0	0	0	0	0	1	1	0	1	

Tablica 2.2: Primjer arbitraže. Uređaj koji prvi pošalje 1 dok drugi šalje 0 odmah prestaje sa slanjem.

Ovakav sustav arbitraže funkcionira jer se mreža pasivno drži u recesivnom stanju, što znači da uređaji moraju aktivno staviti sabirnicu u dominantno stanje. Zato dominantno stanje ima prioritet i pobjeđuje u arbitraži.

2.3 Uvod u VAN protokol (Vehicle Area Network)

VAN je serijska sabirnica dizajnirana za automobile razvijena od strane PSA grupe (Peugeot, Citroën) i Renault-a. Ispunjava istu svrhu kao i CAN, ali se razlikuje u načinu oblikovanja podataka. Kao i CAN, VAN sabirnica koristi isti fizički sloj (diferencijalni par) i sustav arbitraže, ali podatkovni sloj je drugačiji.

Kao i CAN, VAN protokol razlikuje poruke pomoću identifikatora, te pomoću njega radi arbitražu. Ali uz to nudi mogućnost odgovora na poruku u istom okviru, mogućnost slanja više podataka (28 bajtova umjesto 8) i mogućnost direktne komunikacije s drugim uređajem.

Podaci su na VAN mreži organizirani u okvire. Okviri se sastoje od nekoliko dijelova: početak okvira (start of frame; SOF), identifikator (IDEN), naredbe (COM), podaci (DATA), provjera (frame check sum; FCS), signala za kraj podataka (end of data; EOD), signala za potvrdu (ACK) i signala za kraj okvira (end of frame; EOF). Kao i CAN, poslije svakog okvira postoji prazan prostor nakon kojeg se može slati idući okvir. Na slici 2.3 vidimo ilustraciju formata.

SOF 10 TS	IDEN 12 bit 15 TS	Command (4 bit, 5 TS)				DATA max. 28 byte 280 TS	CRC 15 bit 18 TS	EOD 2 TS	ACK 2 TS	EOF 4 TS
		EXT	RAK	R/W	RTR					

Tablica 2.3: Struktura VAN okvira. Informacije preuzete iz članka Info Tech [4] i TSS463C dokumentacije [6].

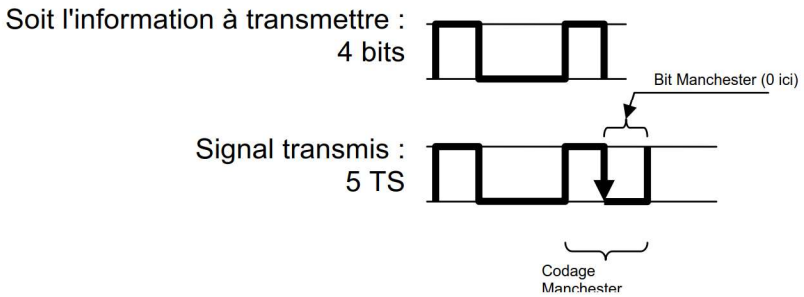
U VAN protokolu razlikujemo četiri vrste poruka, odnosno okvira: normalna poruka (normal frame), zahtjev sa odgođenim odgovorom (reply request frame with deferred reply), odgođeni odgovor (deferred reply frame) i zahtjev sa trenutnim odgovorom (reply request frame with immediate reply). Vrsta poruke je određena sa „Command" bitovima.

2.3.1 Enhanced Manchester kodiranje

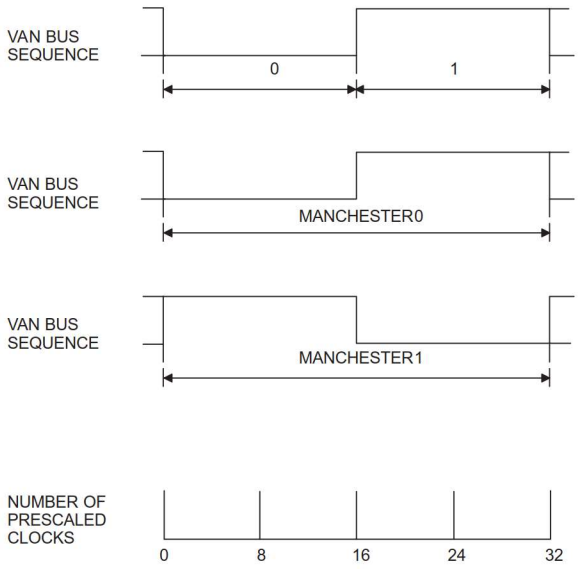
Sve VAN poruke su kodirane pomoću „enhanced Manchester" kodiranja, odnosno e-Manchester. Sve između SOF i EOD niza je kodirano sa e-Manchesterom.

U e-Manchester kodiranju, nakon svaka tri NRZ (Non-Return to Zero), odnosno normalno prenesena bita, četvrti bit je Manchester bit. Ovakvo kodiranje omogućuje lakšu sinkronizaciju svih uređaja na sabirnici, te rudimentarni oblik provjere integriteta poruke.

Tablica 2.3 prikazuje VAN okvir i duljinu svakog dijela u TS jedinicama. 1 TS = 1 preneseni bit, ali s obzirom da koristimo e-Manchester, za svaki 4. bit dodajemo još jedan TS. Vidi slike 2.2 i 2.3.



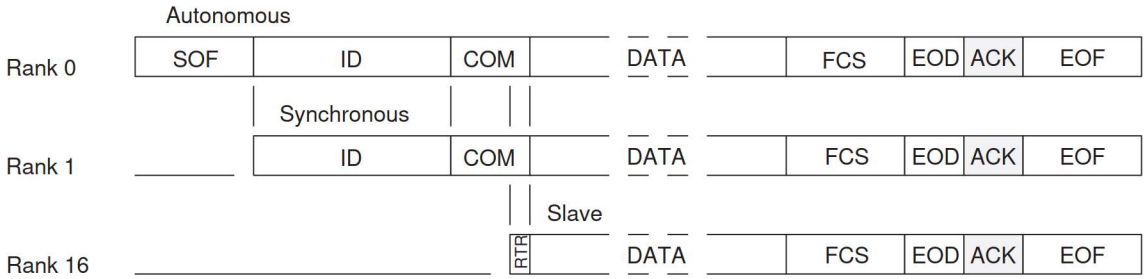
Slika 2.2: TS i bitovi u e-Manchesteru, slika preuzeta iz članka Info Tech. [3]



Slika 2.3: Pregled običnih i Manchester bitova. Slika preuzeta iz TSS463C dokumentacije. [6]

2.4 VAN poruke

Kako bi razumjeli vrste poruka, moramo razlikovati vrste uređaja, odnosno modula koji se mogu naći na sabirnici: autonomni modul, sinkroni modul i sporedni modul. Autonomni modul, odnosno „master“ je glavni uređaj na sabirnici. On može slati Start Of Frame (SOF) niz, može započeti prijenos podataka i može primati podatke. Sinkroni modul ne može slati SOF niz, ali može započeti prijenos podataka i može primati podatke. Sporedni, odnosno slave modul može prenositi podatke samo kao odgovor u već postojećem okviru i može primati podatke.



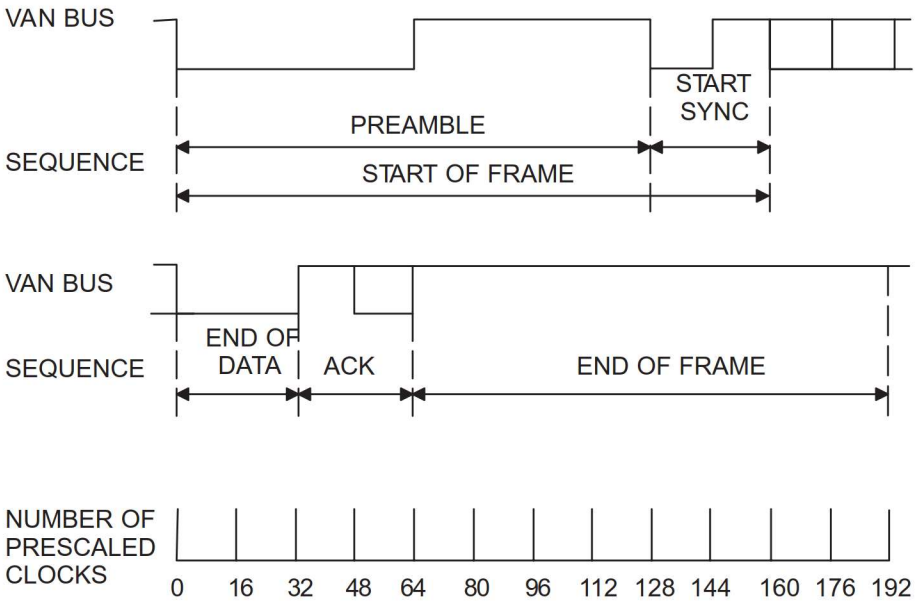
Slika 2.4: Ilustracija i vizualizacija svih vrsta modula. Slika preuzeta iz TSS463C dokumentacije. [6]

Dodatno, moramo razlikovati TS (Time slot) i bitove. Kao što vidimo u slici 2.2, u jednom TS-u možemo prijeneti jedno recesivno, odnosno dominantno stanje, što znači da jedan TS odgovara jednom bitu. Kako koristimo e-Manchester, nakon svakog trećeg bita od jednog TS-a, šaljemo manchester bit od dva TS-a. To znači da nam je za slanje četiri bita potrebno pet TS-a.

U ostatku dokumenta, ako nije naznačeno drugačije, manchester bitovi će se tretirati kao normalni bitovi.

SOF i EOF Tablica 2.3 prikazuje normalni VAN okvir koji započinje sa SOF nizom. Jedino autonomni modul može slati SOF nizove. SOF niz se sastoji od preambule (00001111 u binarnom) i sinkronizacije (01), vidi 2.5. Uređaji na sabirnici koriste SOF niz kako bi sinkronizirali prijenos podataka.

EOF niz se sastoji od barem 5 TS-a u recesivnom stanju [4].



Slika 2.5: Prikaz SOF, EOD, ACK i EOF niza. Slika preuzeta iz TSS463C dokumentacije. [6]

Identifikator Nakon prijenosa cijelog SOF niza, započinje prijenos identifikatora. Sav daljnji prijenos je kodiran pomoću e-Manchestera. Tijekom prijenosa identifikatora, događa se arbitraža. S obzirom da je logička 0 dominantno stanje, a logička 1 recesivno stanje, prvi uređaj koji pošalje 1 gubi arbitražu i čeka sljedeću priliku za slanje poruke.

S obzirom da se u prijenosu najznačajniji bit prenosi prvi, onaj identifikator sa manjim brojem ima prioritet jer sadrži više logičkih 0.

Ako dva paketa imaju isti identifikator, arbitraža se nastavlja i dalje. Na taj način se može produljiti identifikator. Na primjer, može se koristiti kako bi se označila adresa uređaja primatelja, ili adresa izvora u prvim bajtovima podataka.

Identifikator je uvijek 12 bitova i popraćen je sa dodatnih 4 „command“ bita koji glase:

- EXT - uvijek je 1
- RAK - „Request Acknowledge“. Ako je 1, primatelj mora potvrditi primitak poruke, ako je 0, na sabirnici ne smije biti potvrda poruke.
- R/W - „Read/Write“. Označava smjer podataka. Ako je 0, onda je „write“ poruka, odnosno jedan uređaj šalje podatke kako bi ih neki drugi uređaj primio. Ako je 1, onda „read“ poruka. Ona implicira zahtjev za podatke i očekuje odgovor.
- RTR - „Remote Transmission Request“. Ako je 0, u poruci se nalaze podaci. Ako je 1, u poruci nema podataka i implicira zahtjev za podatke s trenutnim odgovorom. Nije dozvoljena poruka sa kombinacijom R/W = 0; RTR = 1.

Podaci Nakon identifikatora i command bitova slijede podaci. To je samo niz podataka kodiranih u e- Manchesteru najveće duljine od 28 bajtova. Podaci su poredani tako da je najznačajniji bit prvi (MSB first).

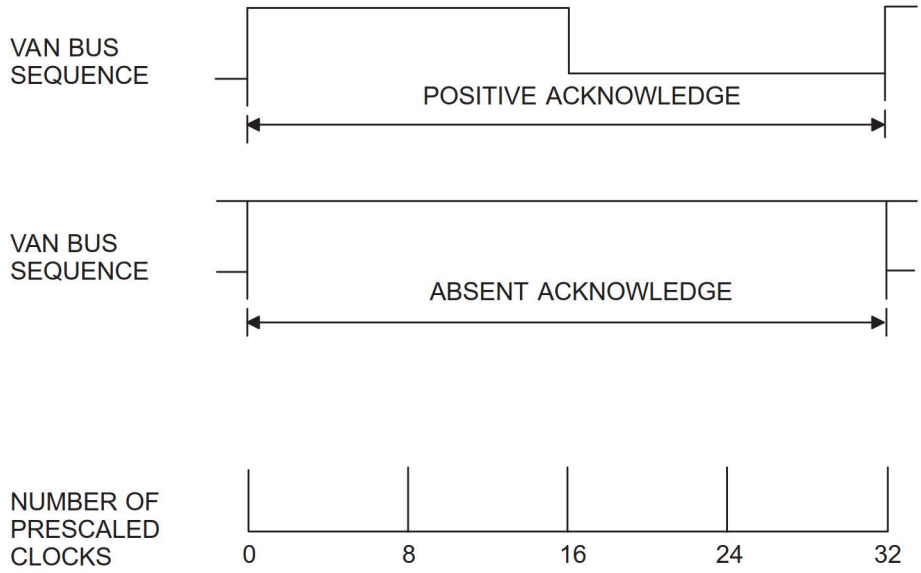
Provjera (FCS) Nakon podataka slijedi provjera podataka. Za to se koristi 15 bitna CRC provjera definirana sa sljedećim polinomom:

$$g(x) = x^{15} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^4 + x^3 + x^2 + 1, \tag{2.1}$$

gdje eksponenti pretstavljaju indeks bita u broju.

Nakon izračuna se izvrši logička I (AND) operacija rezultata sa 0x7FFF i CRC bitovi se invertiraju prije slanja.

ACK Uređaj koji prihvaća podatke mora, ovisno o tome kako je postavljen RAK bit, nakon primitka FCS-a poslati Acknowledge niz, vidi 2.6.



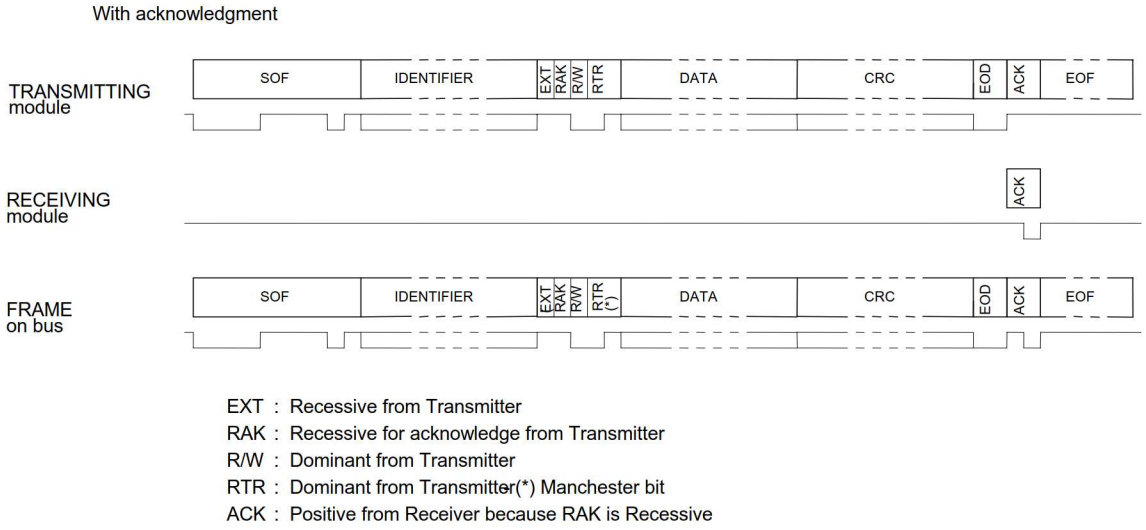
Slika 2.6: Prikaz ACK nizova. Slika preuzeta iz TSS463C dokumentacije [6]

2.4.1 Vrste VAN poruka

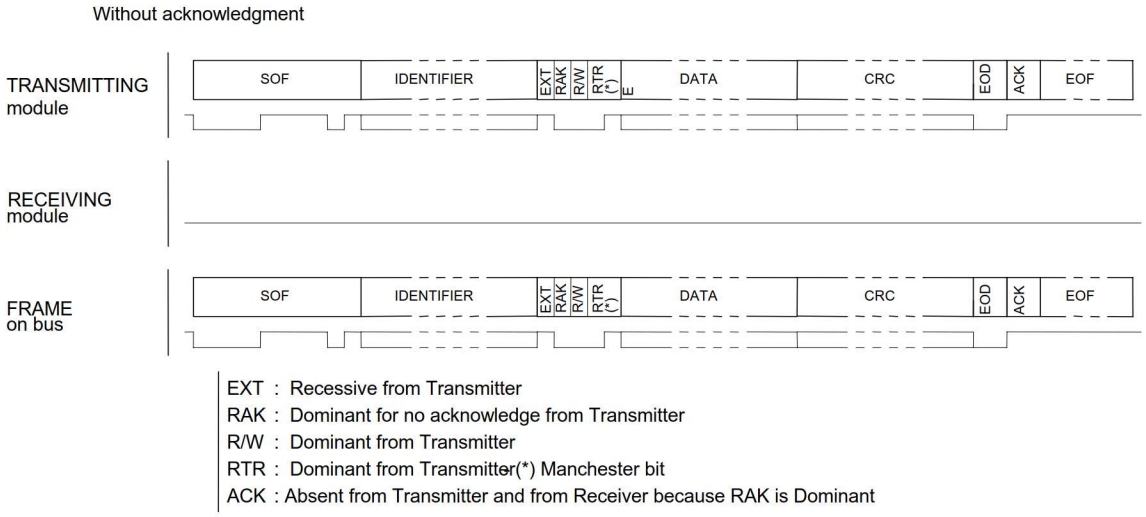
Poruke na VAN sabirnici su generirane pomoću više modula, gdje svaki modul generira svoj dio poruke. Sljedeći primjeri će prikazati sve četiri vrste poruka u VAN standardu. Sve ilustracije su preuzete iz TSS463C dokumentacije [6].

Normalna poruka (Normal frame) Ova vrsta poruke je najjednostavniji oblik poruke. Ovu poruku manje-više prenosi samo jedan modul, dok modul primatelj može poslati ACK. Vidi sliku 2.7.

U slučaju da ACK nije potreban, ni jedan uređaj neće poslati ACK niz. Tada se poruka smatra broadcast porukom. Vidi sliku 2.8.

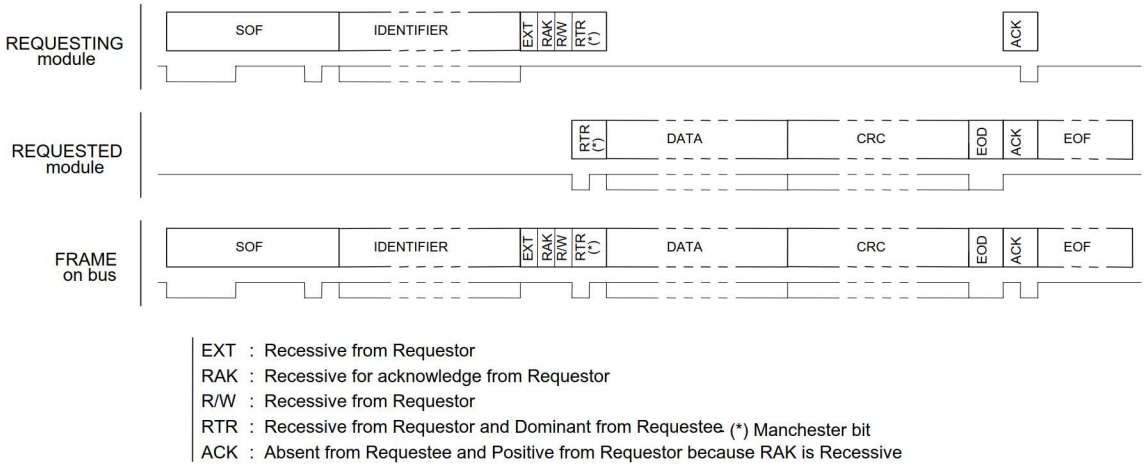


Slika 2.7: Ilustracija prijenosa normalne poruke sa ACK nizom.



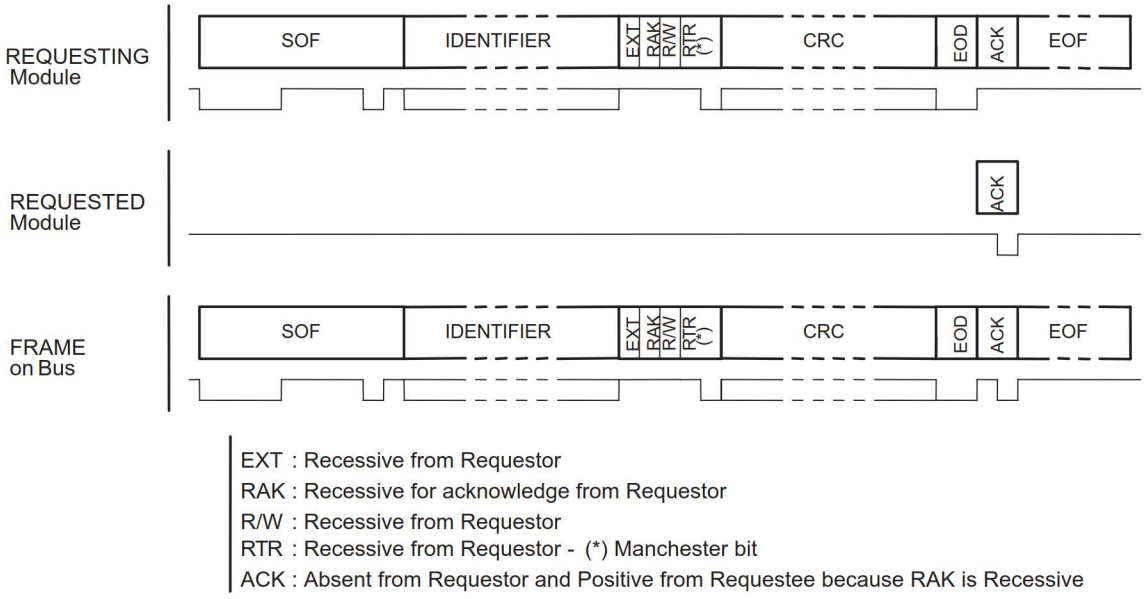
Slika 2.8: Ilustracija prijenosa normalne poruke bez ACK niza.

Zahtjev sa trenutnim odgovorom (Reply Request Frame with Immediate Reply) Ovo je jedina vrsta poruke u kojoj slave modul može slati podatke. Jedina razlika između normalne poruke je R/W bit. Ovo je poruka u čijem slanju aktivno sudjeluju dva, najviše tri modula. Master pošalje SOF niz, modul inicijator (može biti isti modul kao i master) pošalje identifikator, command i na kraju ACK. Sve ostalo dopunjuje uređaj koji odgovara na zahtjev. Vidi sliku 2.9.



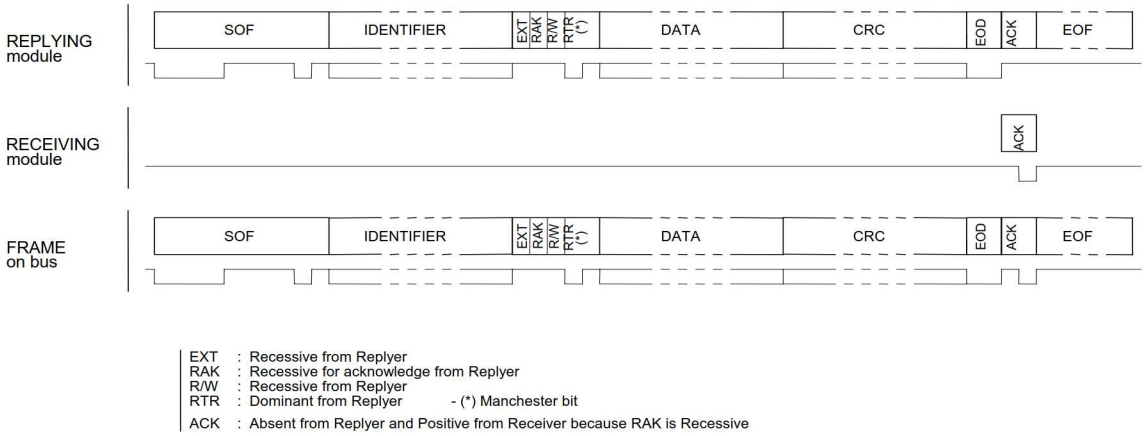
Slika 2.9: Ilustracija prijenosa zahtjeva sa trenutnim odgovorom.

Zahtjev sa odgođenim odgovorom (Reply Request Frame with Deferred Reply) Ova poruka je u osnovi ista kao kod zahtjeva sa trenutnim odgovorom, ali s obzirom da traženi modul ne generira RTR bit, tražeći modul nastavlja sa FCS postupkom i EOF nizom. Traženi modul će generirati ACK ako je tražeći modul postavio RAK bit. Vidi sliku 2.10.



Slika 2.10: Ilustracija prijenosa zahtjeva sa odgođenim odgovorom.

Odgođeni odgovor (Deferred Reply Frame) Ovu poruku šalje modul na kojeg se odnosio zahtjev sa odgođenim odgovorom nakon što je odgovor pripremljen. Oblik poruke je jako sličan normalnoj poruci, razlika je u promjeni R/W bita. Vidi sliku 2.11.



Slika 2.11: Ilustracija prijenosa odgođenog odgovora na zahtjev prikazan u slici 2.10

2.5 Primjena VAN protokola

VAN protokol se koristio samo u automobilima PSA grupe, to jest u Peugeot i Citroën automobilima početkom 2000-ih godina. Oko 2005. godine su počeli prelaziti na CAN protokol, ali neki automobili su koristili VAN čak do 2009. godine.

Informacije o korištenosti protokola u pojedinim automobilima nisu javno dostupne, pa je sljedeći popis sastavljen iz različitih izvora gdje su ljudi eksperimentirali na svojim automobilima:

- Peugeot 206 (proizvedeni poslije 9/2001) i 206+
- Peugeot 207 (modeli do 2005)
- Peugeot 406 (od 2000.)
- Peugeot 307 (2001 - 4/2005)
- Peugeot 1007 (2005 - 2007)
- Peugeot Partner
- Citroën Xsara/Picasso
- Citroën Berlingo
- Citroën C2, C3, C5, C8 (2001 - 2005)

Izvore za ove informacije možemo naći u dokumentacijama više projekata koji se bave VAN protokolom. Prvi je Peter Pinterova Arduino biblioteka za TSS463C [12]. Drugi je Peter Pinterov projekt prevođenja VAN paketa u CAN pakete za novije automobile [15]. Treći izvor je Erik Trompova Arduino biblioteka za čitanje VAN paketa [17].

VAN protokol je korišten za komunikaciju između modula za udobnost, dok je CAN protokol korišten za komunikaciju između modula za upravljanje motorom i dijagnostiku, kao što možemo vidjeti na slici 2.12.

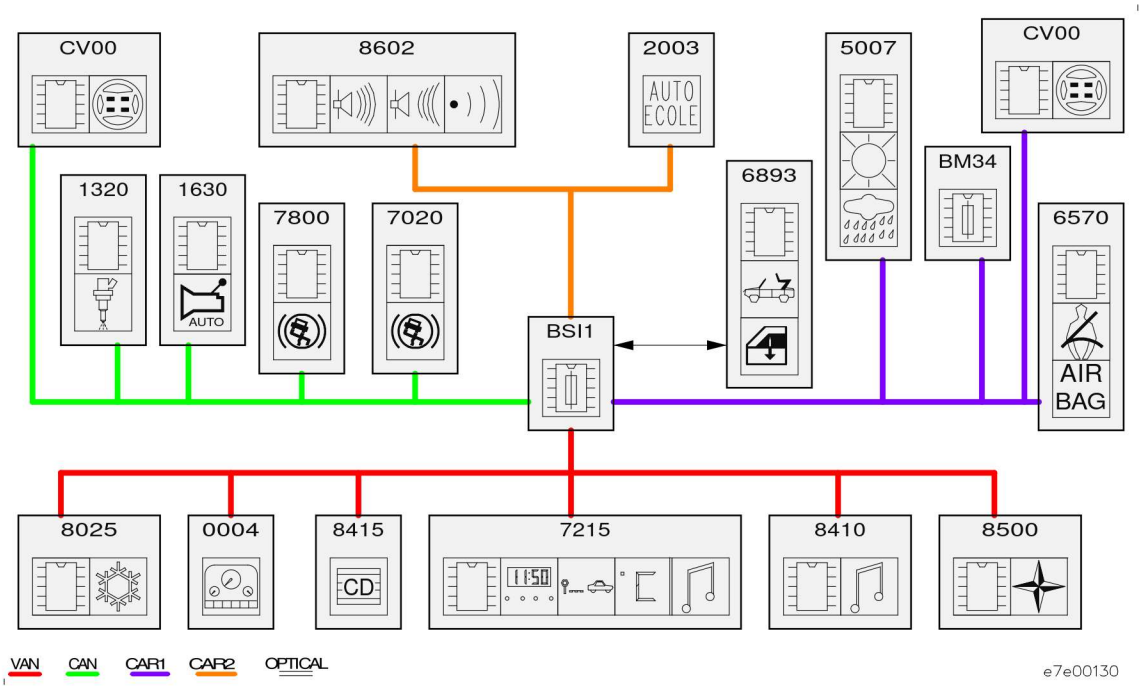
U svakom modelu automobila je topologija mreže drugačija, pa će se u sljedećim primjerima koristiti samo jedan model - Peugeot 206.

VAN mreže

Općenito u bilo kakvim povezanim sustavima, moduli su povezani u mreže. U jednom sustavu može biti više mreža. Tako i u automobilima postoji više mreža koje rade na različitim brzinama i povezane su sa različitim komponentama. U Peugeot 206 automobilu imamo jednu CAN mrežu *CAN Engine* i tri VAN mreže: *VAN Comfort*, *VAN Body 1* i *VAN Body 2*. Vidi sliku 2.12.

Za pristup tim mrežama je zadužen BSI (Built-in Systems Interface) modul. To je glavni modul u automobilu koji upravlja svim drugim modulima i služi kao pristupni (gateway) modul za dijagnostiku.

U Peugeotu 206, mreže rade na različitim brzinama prijenosa.[1] *CAN Engine* radi na 1 Mbit/s, *VAN Comfort* na 125 kbit/s i *VAN Body* mreže na 62.5 kbit/s.



Slika 2.12: Topologija VAN i CAN mreža u Peugeot 206[1]

[1]Informacija preuzeta iz Peugeot SEDRE dokumentacije [5]

3 | Peugeot Infotainment Project

Peugeot Infotainment Project je infotainment rješenje za Peugeot i Citroën automobile koji koriste VAN sabirnicu za komunikaciju. Sustav je još uvijek u procesu izrade i otvorenog je koda. Svi repozitoriji koji su objavljeni u trenutku pisanja ovog rada su izlistani u bibliografiji na stranici 33.

3.1 Dizajniranje prototipa

Plan i motivacija

Cilj projekta je bio ugraditi multifunkcionalni ekran u Peugeot 206 automobil koji bi komunicirao sa automobilom, te čak proširio mogućnosti postojećeg infotainment, odnosno radio sustava.

Motivacija je bila reprodukcija vlastite glazbe na već postojećoj opremi. Jedina opcija je bila korištenje CD medija, ali to nije pouzdano jer je CD čitač u radio uređaju star i osjetljiv na vibracije. Radio uređaj nije imao nikakvu opciju vanjskog ulaza zvuka (AUX) ili funkcionalnost reprodukcije glazbenih datoteka sa USB uređaja. Jedina "vanjska" opcija je ugradnja CD izmjenjivača, ali to ima isti problem kao i ugrađeni CD čitač. Međutim, CD izmjenjivač se spaja u radio, te na tom konektoru postoji vanjski ulaz za zvuk (AUX) koji je ignoriran dok god CD izmjenjivač nije odabran kao izvor zvuka.

U tom trenutku je plan bio poznat - napravit ćemo uređaj koji će se predstavljati kao CD izmjenjivač.

Istraživanje

Kako bi se uspjeli predstaviti kao CD izmjenjivač trebalo je otkriti kako radio i CD izmjenjivač komuniciraju. To istraživanje nas je dovelo do jedne objave na EEVblog forumu[1] u kojoj je aktivan korisnik morcibacsi (Peter Pinter). Na njegovom GitHub-u pronalazimo razne Arduino biblioteke za čitanje [13] i pisanje [12] VAN paketa.

Za čitanje VAN paketa nam ne treba puno posebnog hardvera, samo CAN transciever i ESP32, ali za slanje paketa, zbog različitih oblika poruka koje smo objasnili u poglavlju 2.4 i malo kompleksnijeg oblika arbitraže, nam je potreban poseban IC (*Integrated Circuit*). IC u pitanju je Atmel TSS463C kojeg je još nekako

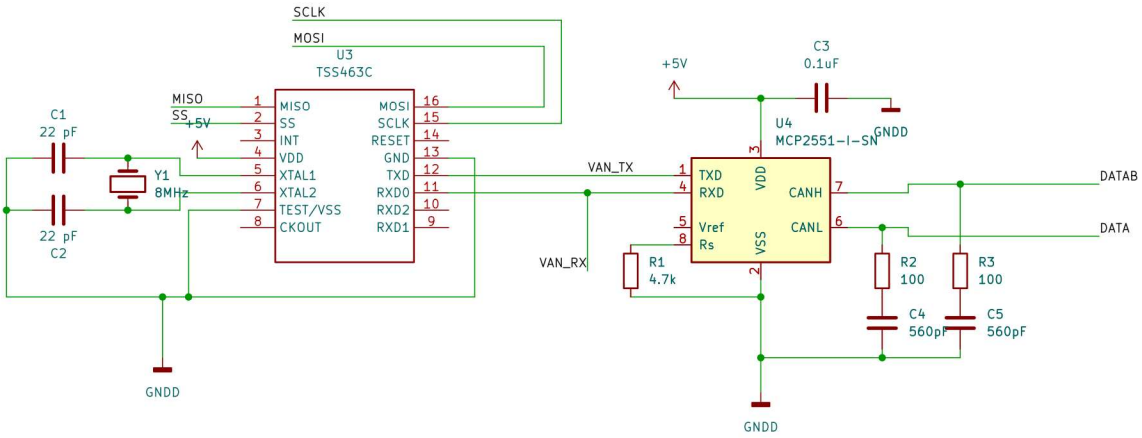
[1] https://www.eevblog.com/forum/projects/van-bus-interfacing-(to-read-car-speed-and-engine-rpm-peugeot-206-1-4-hdi-2002)/

moguće pronaći na AliExpress stranici, vjerojatno jer se još proizvode rezervni dijelovi za starije Peugeot i Citroen automobile. Spajanje i korištenje navedenih komponenti biti će pokriveno u poglavlju Hardver.

Nakon što smo napravili program za čitanje VAN paketa slijedi najbitniji dio - dekodiranje informacija iz paketa. Naime, sada možemo primati podatke sa sabirnice, ali ne znamo što znače. Vidimo puno paketa sa različitim identifikatorima i Command bitovima, ali ne znamo koji uređaj šalje koji paket, koji uređaji odgovaraju i koje informacije se šalju. Srećom neki ljudi su se već bavili sa time. Peter Pinter[2] ima bazu podataka sa paketima koji su djelomice ili u potpunosti dekodirani [14] i Erik Tromp[3] radi projekt gdje prikazuje sve informacije sa VAN sabirnice uživo u web pregledniku i ima dekodirano puno više paketa [16].

Hardver

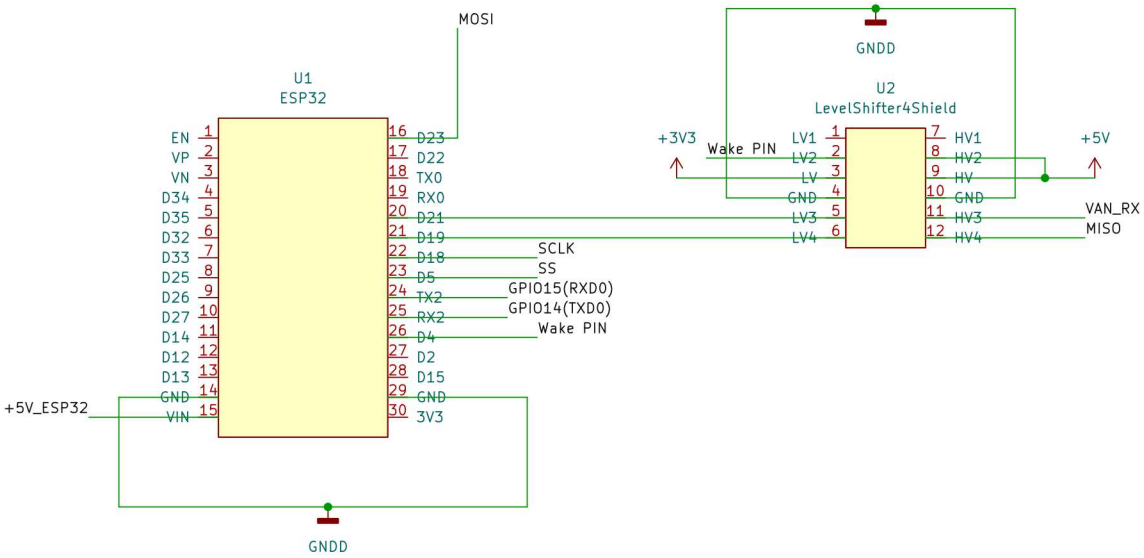
Kako bismo uspješno čitali i slali pakete na VAN sabirnicu, morali smo složiti komponente na breadboard[4]. Koristili smo shemu sa slike 3.1, gdje je *U4* CAN transciever, *U3* TSS463C. *DATA* i *DATAB* se spajaju na VAN sabirnicu.



Slika 3.1: Shema spajanja CAN transcievera i TSS463C IC-a.

TSS463C se na ESP32 spaja preko SPI protokola, te za to koristimo *SCLK*, *MOSI*, *MISO* i *SS* pinove. Za čitanje VAN paketa možemo isto koristiti TSS463C, ali jednostavnije je koristiti biblioteku od Petera Pintera za softversko čitanje. Kako bi to napravili, moramo spojiti *VAN_RX* pin na ESP32. S obzirom da TSS463C radi na 5V, a ESP32 na 3.3V, potreban nam je level shifter koji pretvara logičke signale sa 3.3V na 5V i obrnuto. Na slici 3.2 možemo vidjeti kako smo spojili TSS463C i *VAN_RX* na ESP32 pomoću level shiftera.

[2]Peter Pinter - https://github.com/morcibacsi/
[3]Erik Tromp - https://github.com/0xCAFEDECAF/
[4]Pločica za prototipiranje



Slika 3.2: Shema spajanja ESP32 sa TSS463C i CAN transcieverom pomoću level shiftera.

U programu na ESP32 čitamo VAN pakete, parsiramo ih koristeći informacije iz Peter Pinterove baze i Erik Trompovog projekta i pripremamo za slanje preko UART-a na računalo ili Raspberry Pi.

Softver

Kako znamo koje informacije se nalaze u nekim paketima, možemo koristiti C struct za lakši pristup tim informacijama. Na primjer, znamo da VAN paket sa identifikatorom `0x824` šalje informacije o okretaijma motora, brzini i prijeđenoj udaljenosti. Strukturu tog paketa vidimo u kodu 3.1. Preko biblioteke za primanje VAN paketa dobijemo niz bajtova koji predstavljaju cijelu poruku u obliku kao na slici 2.3.

Kod 3.1 Struktura VAN paketa sa identifikatorom `0x824`

```c
#include <stdint.h>

/**
 * @brief VAN rpm and speed data struct. Iden 0x824
 *
 */
struct psa_van_rpm_speed_struct
{

    uint16_t rpm;           // RPM in big endian. Divide by 8 to get
     ↪ actual rpm.
    uint16_t speed;         // Speed in big endian. Divide by 100 to
     ↪ get actual speed.
    uint16_t distance;      // Distance in big endian. Increments
     ↪ with each passed decimeter.
    uint8_t packet_changed; // Increments each time information
     ↪ significantly changes.
} __attribute__((packed));
```

CD izmjenjivač Kako bismo se predstavili kao CD izmjenjivač moramo otkriti koji paket se odnosi na njega. Na EEVblog forumu doznajemo da treba poslati paket sa identifikatorom `0x4EC`. Iz baze [14] doznajemo format podataka i da je to zapravo zahtjev sa trenutnim odgovorom (vidi stranicu 17) koji MFD (*multifunkcionalni ekran*) šalje prema CD izmjenjivaču i očekuje odgovor. Ako CD izmjenjivač ne odgovori na zahtjev, na sabirnici samo ostaje zaglavlje poruke koje možemo pročitati sa ESP32 VAN bibliotekom [13]. Međutim, kada smo spojili prototip u naš automobil, otkrili smo da se poruka sa tim identifikatorom uopće ne šalje. Zašto?

S obzirom da je CD izmjenjivač dodatna oprema, on se mora omogučiti u glavnom računalu automobila i MFD-u, što je moguće samo sa originalnim Peugeot programom za dijagnostiku, *Peugeot Planet 2000*. Naš auto nije došao sa CD izmjenjivačem, pa je ta opcija isključena i paket `0x4EC` se ne šalje. Nakon što smo nabavili softver za dijagnostiku i omogučili CD izmjenjivač, paket `0x4EC` se pojavio na sabirnici.

Format paketa `0x4EC` možete vidjeti u kodu 3.2. Bitno je slati brojeve u BCD (Binary-Coded decimal) formatu, gdje jedan bajt ne predstavlja jedan broj od 0 do 255, nego dvije znamenke. Podijelimo bajt na dva dijela od 4 bita, gdje lijevi dio predstavlja lijevu znamenku, a desni desnu znamenku. Na primjer, kako bi poslali broj 17, umjesto `0x11`, pošaljemo `0x17`. Periodičnim slanjem ovog paketa sa izmjenjenim header i footer poljem, radio nam dozvoli prebacivanje na CD izmjenjivač i reprodukciju zvuka preko AUX konektora.

Kod 3.2 Struktura VAN paketa sa identifikatorom `0x4EC`

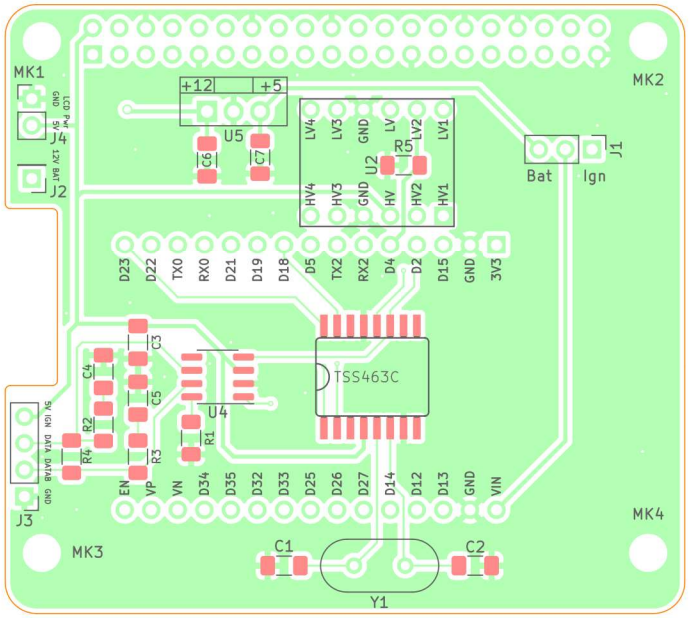
```
#include <stdint.h>

enum van_cd_changer_status
{
    VAN_CDC_OFF = 0x41,
    VAN_CDC_ON_PAUSED = 0xC1,
    VAN_CDC_ON_BUSY = 0xD3,
    VAN_CDC_ON_PLAYING = 0xC3,
    VAN_CDC_ON_FAST_FORWARD = 0xC4,
    VAN_CDC_ON_FAST_REWIND = 0xC5,
};

enum van_cd_changer_cd_presence
{
    VAN_CDC_CD_PRESENT = 0x16,
    VAN_CDC_CD_NOT_PRESENT = 0x06,
};

struct van_cd_changer_packet
{
    uint8_t header;       // 0x80 to 0x87. Same as footer
    uint8_t shuffle;      // 1 if tracks are shuffled
    uint8_t status;       // enum van_cd_changer_status
    uint8_t cd_present;   // enum van_cd_changer_cd_presence
    uint8_t minutes;      // Track minute progress in bcd
    uint8_t seconds;      // Track seconds progress in the current
     ↪ minute in bcd
    uint8_t track_num;    // Number of the current track in bcd
    uint8_t cd_num;       // Number of the current CD in bcd
    uint8_t total_tracks; // Total number of tracks in bcd
    uint8_t unknown;
    uint8_t cd_flag; // bitwise representation of present CDs
    uint8_t footer;  // Same as header
} __attribute__((packed));
```

3.2 Sastavljanje uređaja

PCB i ugradnja

Kako bismo mogli lakše ugraditi naš uređaj, dizajniramo PCB, odnosno Printed Circuit Board na kojoj se nalaze sve komponente potrebne za rad. Kako ćemo taj PCB ugraditi na Raspberry Pi, možemo ga dizajnirati kao HAT dodatak. Na slici 3.3 vidimo konačni dizajn PCB-a. Kako planiramo napajati Raspberry Pi i ekran u isto vrijeme, dodajemo izlaze za napajanje ekrana. Želimo ESP i Raspberry napajati odvojeno, te spajamo ESP na stalno napajanje, a Raspberry na kontakt napajanje.

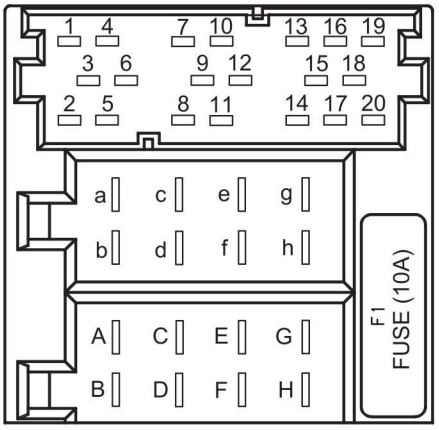
Ako želite sami isprobati ovaj PCB, na GitHub-u se nalaze sheme[5] i dokumentacija[6] za pločicu.



Slika 3.3: Konačni dizajn PCB-a koji se koristi u projektu.

[5] https://github.com/Mive82/psa-pcb
[6] https://github.com/Mive82/psa-docs

No.	Description
1	N.C.
2	N.C.
3	N.C
4	N.C.
5	N.C.
6	N.C.
7	VOICE SYN
8	VOICE SYN-GND
9	N.C.
10	N.C.
11	AUX(+)
12	AUX(-)
13	VAN DATA
14	VAN DATA/
15	CD A/C GND
16	CD A/C B/U
17	+VAN
18	CD A/C S-GND
19	CD A/C L
20	CD A/C R



Slika 3.4: Pinout radio konektora. Nas zanimaju pinovi 13 do 20. Slika preuzeta iz priručnika za servisiranje RD3 radia [7].

Također, kako bismo spojili PCB u radio, moramo napraviti konektor. Na slici 3.4 vidimo pinout radio konektora na kojem nas zanimaju pinovi za CD changer, što su pinovi 13 do 20. Sa slike 3.1 imamo *DATA* i *DATAB* koje spajamo na pinove 13 i 14. Na pin 15 spajamo uzemljenje napajanja. Pin 16 nudi stalno napajanje, te na njega spajamo 5V regulator sa kojim napajamo ESP32. Na pinu 17 se nalazi kontakt napajanje, ono je pod naponom samo kada je auto aktivan. Na njega spajamo vanjski, jači 5V regulator i napajamo Raspberry Pi i ekran. Pinove 18, 19 i 20 spajamo na 3.5mm AUX konektor i to je naš ulaz za zvuk.

Raspberry Pi

Raspberry Pi 3 je malo računalo idealno za ovakve projekte. Na njemu radi Linux operativni sustav i ima dobre grafičke sposobnosti.

Kako bismo ga iskoristiti što više moguće, napravit ćemo prilagođenu Linux distribuciju na kojoj će se nalaziti samo stvari koje su nam potrebne. Koristili smo Yocto[7] kao build system i napravili prilagođeni layer (*sloj*), kojeg smo nazvali `meta-psa`, na kojem se nalazi popis svih programa koji nam trebaju, dodatna prilagođavanja za Qt, gstreamer i pulseaudio, te splash slika pri uključivanju. U budućnosti, kada program bude više spreman, i on će biti uključen u `meta-psa` layer.

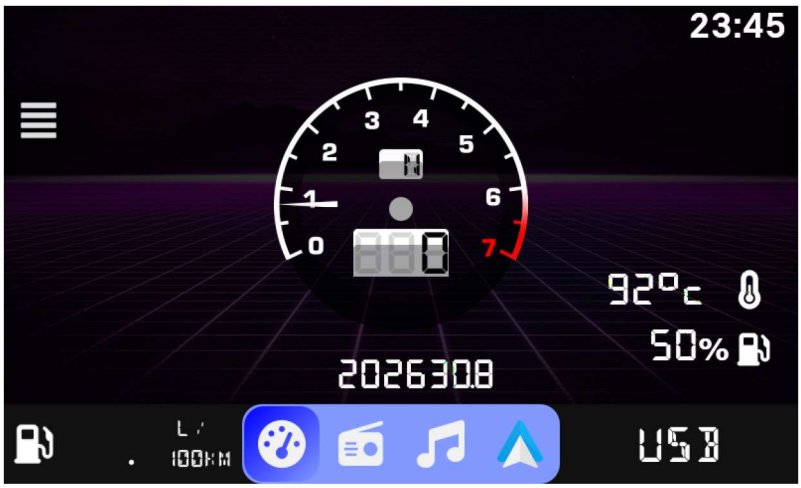
Grafičko sučelje

Grafičko sučelje za sustav je napravljeno u Qt programskom okviru. Qt nudi jednostavno pravljenje grafičkih aplikacija, te nudi pokretanje istih bez windowing

[7] https://www.yoctoproject.org/

systema[8] na Raspberry Pi, što dodatno pojednostavljuje našu Linux distribuciju.

Za izgled i raspored sučelja smo se vodili po dizajnu i ponašanju postojećeg ekrana (MFD) u automobilu. Prikazani izbornik se automatski mijenja ovisno o odabranom izvoru zvuka, stanju radio uređaja i stanju automobila. Slijede slike i kratki opis svakog izbornika. Sav izvorni kod se nalazi na GitHub-u[9].



Slika 3.5: Prikaz informacija o motoru. Ovaj izbornik prikazuje okretaje motora, brzinu, temperaturu motora, stanje goriva i trenutnu kilometražu. Također može prikazati informacije sa putnog računala i čak ih resetirati.

Slika 3.6: Varijacija slike 3.5 sa prikazanim informacijama sa putnog računala.

[8]Window system - Sustav koji se koristi za prikazivanje grafičkih aplikacija. Na Linuxu se koriste X11 i Wayland.

[9]https://github.com/Mive82/psa-app

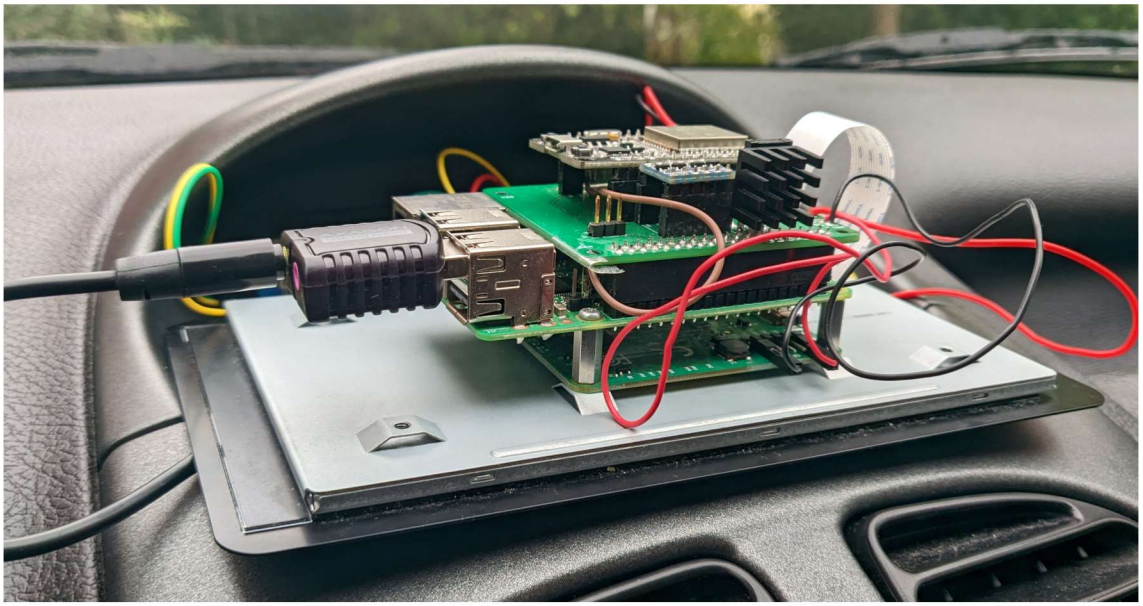
Slika 3.7: Prikaz informacija o radio prijemniku. Ovaj izbornik prikazuje informacije o trenutnoj radio stanici i popis svih zapamćenih stanica. Također prikazuje informacije o ugrađenom CD reproduktoru kada se reproducira.

Slika 3.8: Prikaz informacija o satu. Ovaj izbornik se prikazuje kada se radio isključen, a automobil uključen. Prikazuje trenutno vrijeme i datum.

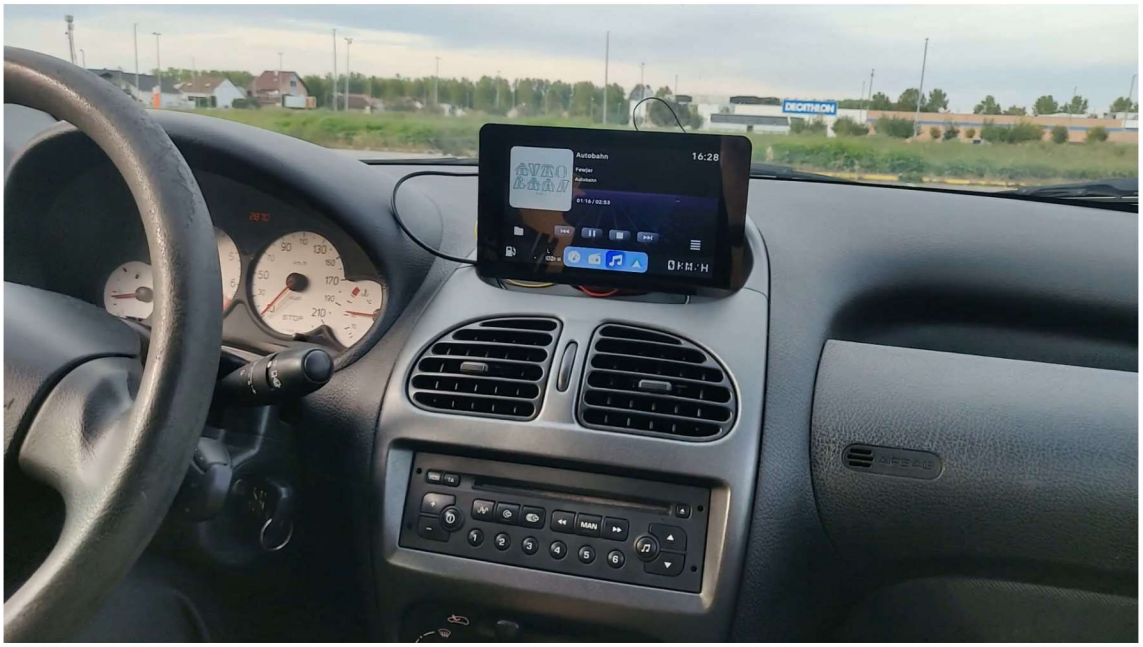
Slika 3.9: Reproduktor glazbe. Kada je odabran CD izmjenjivač na radio uređaju, otvori se ovaj izbornik. Reproduktor čita glazbu sa USB uređaja u svim poznatim formatima i reproducira ju kroz odgovarajuće pinove sa slike 3.4. Reproduktor također podržava meta-podatke poput imena pjesme, albuma, izvođača i slike albuma.

Prikaz konačnog uređaja

Nakon što smo napravili prototip i pretvorili ga u gotov uređaj, vrijeme je da ga stavimo u pogon. U ovom dijelu će biti slike uređaja u pravom automobilu.



Slika 3.10: Slika cijelog uređaja. Ovdje se vidi Raspberry Pi, naš PCB i ESP32 kako izgledaju uživo.



Slika 3.11: Slika uređaja u pogonu. Ovdje se vidi sučelje i originalni RD3 radio uređaj. Na sučelju je prikazan reproduktor glazbe.

3.3 Zaključak

Ovaj projekt je daleko od kraja. U budućnosti planiramo u potpunosti zamijeniti postojeći MFD sa našom implementacijom. Međutim da to napravimo, potrebno je detaljnije istraživanje VAN paketa. Ovaj rad će, nadamo se, služiti kao motivacija nekome drugome da se upusti u istraživanje ovog čudnog i nepoznatog francuskog protokola koji se koristi u nekoliko stotina tisuća automobila.

Za pregled već postojećih informacija, pregledajte sve repozitorije i dokumente navedene u bibliografiji ovog rada i nadamo se da ćemo uspjeti zainteresirati nekoga od vas u daljnje istraživanje VAN paketa.

Bibliografija

[1] Graham Auld. *VAN Intro*. 2001. URL: http://graham.auld.me.uk/projects/vanbus/index.html.

[2] *CAN bus*. URL: https://en.wikipedia.org/wiki/CAN_bus.

[3] Alain Chautar. „Info Tech n°6". *Educauto.org* (2003.). URL: http://graham.auld.me.uk/projects/vanbus/datasheets/Mux1.pdf.

[4] Alain Chautar. „Info Tech n°11". *Educauto.org* (2004.). URL: http://graham.auld.me.uk/projects/vanbus/datasheets/Mux3.pdf.

[5] PSA Peugeot Citroën. *Peugeot Service Box backup - SEDRE*.

[6] Atmel Corporation. *VAN Data Link Controller with Serial Interface, TSS463C*. 2006.

[7] Clarion Co. Ltd. *RD3 Service Manual*.

[8] Mislav Milinković. *Documentation for the Infotainment Project*. URL: https://github.com/Mive82/psa-docs.

[9] Mislav Milinković. *ESP32 Firmware for the Infotainment Project*. URL: https://github.com/Mive82/psa_esp32.

[10] Mislav Milinković. *Peugeot Infotainment Project PCB*. URL: https://github.com/Mive82/psa-pcb.

[11] Mislav Milinković. *Qt GUI for the Infotainment Project*. URL: https://github.com/Mive82/psa-app.

[12] Peter Pinter. *Arduino TSS463/TSS461 VAN interface library*. URL: https://github.com/morcibacsi/arduino_tss463_van.

[13] Peter Pinter. *ESP32 RMT peripheral Vehicle Area Network (VAN bus) reader*. URL: https://github.com/morcibacsi/esp32_rmt_van_rx.

[14] Peter Pinter. *PSA VAN bus, All data related to Peugeot 206 307 406 S2, Citroen C3 2004*. URL: http://pinterpeti.hu/psavanbus/PSA-VAN.html.

[15] Peter Pinter. *PSA VAN-CAN protocol bridge*. URL: https://github.com/morcibacsi/PSAVanCanBridge.

[16] Erik Tromp. *PSA (Peugeot, Citroën) VAN bus Live Connect*. URL: https://github.com/0xCAFEDECAF/VanLiveConnect.

[17] Erik Tromp. *VAN bus reader/writer for Arduino-ESP8266*. URL: https://github.com/0xCAFEDECAF/VanBus.

Sažetak

U ovom radu ćemo se upoznati sa komunikacijskim protokolima u automobilima i njihovom primjenom. Pobliže ćemo pogledati VAN protokol i upoznati se sa vrstama poruka koje on podržava. Osvrnuti ćemo se i na njegovu primjenu u konkretnim automobilima. Koristeći ESP32 mikrokontroler i Raspberry Pi 3 računalo ćemo napraviti vlastiti infotainment uređaj koji koristi VAN protokol kako bi komunicirao sa računalima unutar automobila.

Ključne riječi

Ugrađeni sustavi, Linux, Mikroupravljači, ESP32, ESP, Arduino, Raspberry Pi, VAN, VAN protokol, Peugeot, PSA, Qt, C, C++, Automobil, PCB

The VAN Protocol and its application in automobiles

Summary

In this paper we will look at network communication protocols in automobiles, and their application. We will focus on the VAN protocol, and the message types it supports. Additionally, we will cover it's usage in a specific vehicle model. Using an ESP32 microcontroller and a Raspberry Pi 3 computer, we will be making an infotainment system that uses the VAN protocol to communicate with computers inside the car.

Keywords

Embedded systems, Linux, Microcontrollers, ESP32, ESP, Arduino, Raspberry Pi, VAN, VAN protocol, Peugeot, Qt, C, C++, Automobile, PCB