



UNIVERZITET U SARAJEVU
ELEKTROTEHNIČKI FAKULTET
ODSJEK ZA AUTOMATIKU I
ELEKTRONIKU

CAN SERVER

Komunikacija CAN-a i MQTT-a

ZAVRŠNI RAD
- PRVI CIKLUS STUDIJA -

Student:
Abdulah Kapić

Mentor:
Prof. dr. Samim Konjicija

Sarajevo,
septembar 2024.



UNIVERSITY IN SARAJEVO THE
FACULTY OF ELECTRICAL
ENGINEERING
DEPARTMENT OF
AUTOMATION AND
ELECTRONICS

CAN SERVER
Communication between CAN and MQTT

FINAL PAPER
- FIRST CYCLE -

Student:
Abdulah Kapić

Mentor:
Prof. dr. Samim Konjicija

Sarajevo,
september 2024

Elektrotehnički fakultet, Univerzitet u Sarajevu
Odsjek za automatiku i elektroniku
Prof. dr. Samim Konjicija
Sarajevo,.....

Postavka zadatka završnog rada I ciklusa:

Nakon odobrenja teme *CAN server* za završni rad Prvog ciklusa studija Odsjeka za automatiku i elektroniku, prema uputstvima profesora, u radu je potrebno:

- napraviti aplikaciju koja će omogućiti da se uspostavi komunikacija između CAN (Controller Area Network) mreže i neke druge mreže;
- upoznati se sa CAN standardom na hardverskom i softverskom nivou;
- upoznati se sa MQTT protokolom;

Implementaciju može biti rađena u Pythonu.

Polazna literatura:

- [1] https://en.wikipedia.org/wiki/CAN_bus,
- [2] <https://blog.ansi.org/controller-area-network-can-standards-iso-11898/>,
- [3] <https://www.ti.com/lit/an/sloa101b/sloa101b.pdf>,
- [4] <https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial>,
- [5] <https://community.nxp.com/t5/NXP-Tech-Blog/101-Controller-Area-Network-CAN-standard/ba-p/1217054>
- [6] <https://python-can.readthedocs.io/en/stable/>
- [7] <https://python-can.readthedocs.io/en/stable/interfaces.html>
- [8] Predavanja iz Ugradbenih sistema na c2 (druga godina RI, IV semestar) ili Interneta stvari (treća godina AiE, V semestar)

Univerzitet u Sarajevu
Elektrotehnički fakultet
Odsjek za

Izjava o autentičnosti radova

Završni rad I ciklusa studija

Ime i prezime: Abdulah Kapić

Naslov rada: CAN server (Komunikacija CAN-a i MQTT-a)

Vrsta rada: Završni rad Prvog ciklusa studija

Broj stranica:

Potvrđujem:

- da sam pročitao dokumente koji se odnose na plagijarizam, kako je to definirano Statutom Univerziteta u Sarajevu, Etičkim kodeksom Univerziteta u Sarajevu i pravilima studiranja koja se odnose na I i II ciklus studija, integrirani studijski program I i II ciklusa i III ciklus studija na Univerzitetu u Sarajevu, kao i uputama o plagijarizmu navedenim na web stranici Univerziteta u Sarajevu;
- da sam svjestan univerzitetskih disciplinskih pravila koja se tiču plagijarizma;
- da je rad koji predajem potpuno moj, samostalni rad, osim u dijelovima gdje je to naznačeno;
- da rad nije predat, u cjelini ili djelimično, za stjecanje zvanja na Univerzitetu u Sarajevu ili nekoj drugoj visokoškolskoj ustanovi;
- da sam jasno naznačio prisustvo citiranog ili parafraziranog materijala i da sam se referirao na sve izvore;
- da sam dosljedno naveo korištene i citirane izvore ili bibliografiju po nekom od preporučenih stilova citiranja, sa navođenjem potpune reference koja obuhvata potpuni bibliografski opis korištenog i citiranog izvora;
- da sam odgovarajuće naznačio svaku pomoć koju sam dobio pored pomoći mentora i akademskih tutora/ica.

Sarajevo, datum _____

Potpis:

Sažetak

U ovom radu će biti predložene karakteristike CAN mreže i MQTT-a, a ujedno i njihovo povezivanje kao i koristi. Cilj je da se napravi generisanje poruka preko CAN mreže, bilo to virtuelno ili hardverski, te da se zatim u Python programskom jeziku omogući komunikacija, tj. slanje poruka preko CAN-a na MQTT, i obrnuto. Primarni fokus će nam biti na kodu u Python programskom jeziku, korištenje određenih biblioteka, specifičan standard CAN biblioteka, no to ne isključuje da se nećemo nadovezati i na virtuelno generisanje CAN poruka, u našem slučaju korištenje Socketcana na Windowsu, pa tako i komunikacija u MQTT-u, te njihove karakteristike.

Ključne riječi: CAN, MQTT

Abstract

In this work we will show characteristics of CAN network and MQTT, but also we will show how to connect them and use them. Our purpose is to make message (frame) generation through CAN network, either virtually or with uses of some hardware devices, after that in Python programing language we should enable communication (sending messages) through CAN on to MQTT, and also from MQTT to CAN. Our main focus is to show how to program this connection in Python programming language, specific uses of libraries, usage of specific standard for CAN networking, but also that doesn't exclude that we wont also say something about virtual generation of CAN messages (frames), in our case usage of Socketcan on Windows, but also communication in MQTT and their characteristics.

Keywords: CAN, MQTT

Sadržaj

Sažetak.....	4
Abstract.....	4
Sadržaj.....	5
Popis slika.....	6
1. UVOD.....	7
1.1 Control Area Network.....	7
1.2 Historija CAN-a.....	7
1.3 Aplikacija CAN-a.....	8
1.4 Arhitektura CAN-a.....	8
2. CAN funkcionalnost.....	10
2.1 Prijenos podataka u CAN-u.....	10
2.2 Alokacija ID-a.....	11
2.3 Tajmiranje bitova.....	12
2.4 Slojevi CAN-a.....	12
2.5 Format podataka (poruke).....	13
2.6 Remote poruke.....	15
2.7 Poruka greške.....	16
2.8 Overload.....	16
2.9 ACK slot i izmeđuokvirno razdjeljivanje.....	17
2.10 Bit stuffing.....	17
2.11 CAN bus.....	18
3. CAN standardi i protokoli.....	20
3.1 CAN standardi za niže nivoe.....	20
3.2 CAN bazirani protokoli za više nivoe.....	20
4. MQTT.....	21
4.1 Općenito.....	21
4.2 Teme.....	22
4.3 QoS (Quality of Service).....	22
4.4 MQTT brokeri.....	23
4.5 MQTT klijenti.....	23
4.6 MQTT kao protokol za IOT.....	24
5. CAN server.....	25
5.1 Razvojno okruženje generisanja CAN poruka.....	25
5.2 WSL2/Ubuntu.....	26
5.3 Postavljanje virtualne CAN mreže u WSL2.....	27
6. CAN server u mostu sa MQTT.....	34
6.1 Python programski jezik.....	34
6.2 Python u Linuxu preko Ubuntu/WSL2.....	34
6.3 Prijenos CAN poruka na MQTT i obrnuto.....	34
6.3.1 Programski kod u Pythonu za prijenos poruka.....	35
6.4 Simulacija za prijenos poruka.....	37
7. Zaključak.....	39
8. Reference.....	40

Popis slika

Slika 1.1: Prikaz povezivanja čvorova na sabirnici visoko brzinski CAN ISO 11898-2.....	9
Slika 2.1: 11-bitna CAN mreža	11
Slika 2.2: Base frame format	13
Slika 2.3: Ekstendovani format poruke.....	15
Slika 2.4: CAN okvir prije i poslije dodavanja stuff bitova.....	18
Slika 2.5: ECU.....	19
Slika 2.6: CAN DB9 bus connector.....	20
Slika 4.1: Pokretanje MQTT-X desktop verzije.....	24
Slika 5.1: postupak za pronalaženje CAN postavki.....	29
Slika 5.2: Postupak za pokretanje podrške za CAN podsistem.....	30
Slika 5.3: Unošenje željenog interfejsa ili protokola.....	31
Slika 5.4: Postavljanje M za vcan.	32
Slika 5.5: manuelno slanje poruka.	33
Slika 5.6: manuelno primanje poruka.	33
Slika 6.1: Prikaz svih uvedenih biblioteka i postavki za MQTT	35
Slika 6.2: Funkcija za slanje CAN-a na MQTT, postavljanje poveznice.	36
Slika 6.3: Funkcija MQTT na CAN.	36
Slika 6.4: threadovi	37
Slika 6.5: Poslana poruka sa CAN na MQTT.....	37
Slika 6.6: Prikaz prenesene poruke sa MQTT-a u Pythonu	38
Slika 6.7: Poslana poruku sa MQTT	38
Slika 6.8: candump vcan 0 nakon slanja poruke sa MQTT-X na CAN.	39

1. UVOD

1.1 Control Area Network

CAN je standard za vozila koji povezuje komponente unutar vozila (auta, vozovi, avioni). Prvobitno je napravljen da smanji kompleksnost i cijenu ožičavanja u automobilima kroz multipleksiranje. Nakon toga, CAN protokol je korišten u mnogim kontekstima.

Orijentisan porukama, ovaj protokol garantuje integritet podataka i prioritizira podatke kroz proces nazvan *arbitracija*, time dozvoljavajući uređaju sa najvećim prioritetom da nastavlja prijenos ukoliko ima više uređaja da šalju podatke u isto vrijeme, dok ostali čekaju.

1.2 Historija CAN-a

Proizvodnja CAN-a je počela 1983. u Robert Bosch GmbH. Protokol je oficijalno propušten u proizvodnju i prodaju 1986. na SAE konferenciji u Detroitu. Prvi čipovi za CAN kontroler su bili uvedeni od strane Intela u 1987, i poslije kratko od strane Philipsa. Mercedes-Benz W140 je bila prva produkcija koja je isto tako predstavljala CAN bazirani multipleksirani ožičeni sistem. 1993. ISO je pustio CAN standard ISO 11898 koji se kasnije razdvojio u dva dijela ISO 11898-1 i ISO 11898-2.

Kasnije Bosch proizvodi CAN FD 1.0 2012. godine. Ova specifikacija koristi drukčiji format poruka koji dozvoljavava da se različite dužine podataka šalju a isto tako da se može opcionalno promijeniti na brži prijenos bitova, nakon što je proces arbitracije završen.¹

¹ https://en.wikipedia.org/wiki/CAN_bus

1.3 Aplikacija CAN-a

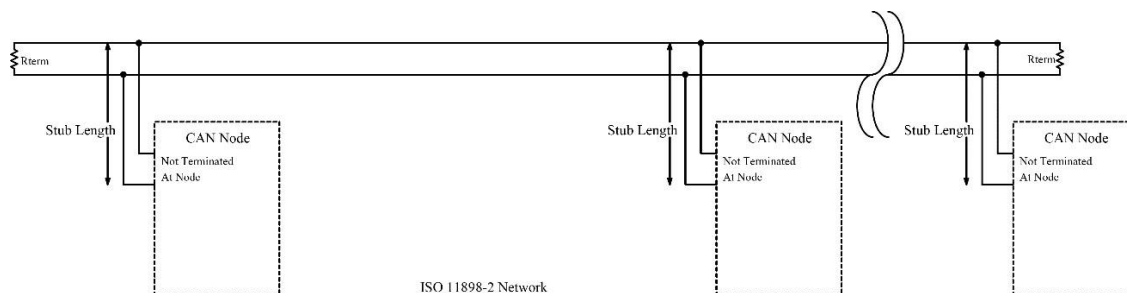
CAN se koristi u mnogim primjenama kao što su:

- a) putnička vozila, kamioni, autobusi;
- b) poljoprivredni alati;
- c) elektronički alati za avijaciju i navigaciju;
- d) električni generatori;
- e) industrijska automacija i mehanička kontrola;
- f) liftovi;
- g) medicinski instrumenti;
- h) brodovi;
- i) 3D printeri.

1.4 Arhitektura CAN-a

Fizička organizacija CAN-a je u suštini standard za povezivanje (ECUs), elektroničke kontrolne jedinice koje su poznate kao čvorovi. Dva ili više čvora su potrebni na CAN sabirnici da bi se ostvarila komunikacija. Čvor može imati jednostavnu digitalnu logiku sve do nekog ugrađenog računara koji pokreće ekstenzovani softver. Svi čvorovi su povezani preko dvožične sabirnice. Ove žice su parovi od 120 ohma karakteristične impedanse.

ISO0 11898-2 se naziva visoko-brzinski CAN (brzina može ići do 1 Mbit/s na CAN-u dok na CAN-FD ide i do 5 Mbit/s). Ovaj tip koristi linearne sabirnice koje su terminirane na svakom kraju sa 120 ohma otpornicima.



Slika 1.1: Prikaz povezivanja čvorova na sabirnici visoko brzinski CAN ISO 11898-2.²

Električne karakteristike su takve da brzina tranzicije je brža kad se dešava recesivna ka dominantnoj tranziciji jer su CAN žice onda aktivno nošene, a u suprotnom brzina ovisi od dužine CAN mreže i kapaciteta žice koja je korištena. Visoko brzinski CAN je tipično korišten u automatizacijskim i industrijskim aplikacijama, gdje sabirnica ide sa jednog kraja okruženja do drugog.

CAN koji je tolerantan na greške se inače koristi gdje grupa čvorova treba da bude povezana.

Čvorovi su osnovni dio jedne CAN mreže. Svaki čvor zahtijeva:

- a) centralnu procesorsku jedinicu;
- b) mikroprocesor;
- c) host procesor.

Svaki host procesor odlučuje šta znače dostavljene poruke i šta poruke trebaju prenijeti. Senzori, aktuatori i kontrolni uređaji mogu se povezati na host procesor.

CAN kontroler je integralni dio mikrokontrolera. Transiver dok prima podatke konvertuje strim podataka od CAN sabirnice na nekim nivoima do nivoa gdje CAN

² https://en.wikipedia.org/wiki/CAN_bus

kontroler može da koristi, ima zaštićene krugove da zaštiti CAN kontroler.

A dok prenosi podatke, onda ih prenosi od CAN kontrolera pa sve do nivoa CAN sabirnice.

Svaki čvor može da šalje i prima poruke, ali ne i u isto vrijeme. Poruka se sastoji primarno od ID (identifikatora) koji reprezentuje prioritet prouke i ide do 8 bajtova podataka, zatim od CRC-a, ACK acknowledge slot, itd.

Dodatno CAN FD produžuje dužinu sekcije podataka čak i do 64 bajta po poruci.

Uređaji koji su povezani sa CAN mrežom su inače senzori, aktuatori. Ovi uređaji su povezani na sabirnicu kroz host procesor, CAN kontroler i CAN transiver.

2. CAN funkcionalnost

2.1 Prijenos podataka u CAN-u

CAN prenos podataka koristi arbitraciju. Ova metoda zahtijeva da svi čvorovi na CAN mreži budu sinhronizirani tako da mogu testirati svaki bit na mreži u isto vrijeme. To je i jedan od razloga zašto se CAN naziva sinhronizovanim. Međutim, to nije i najtačnije, jer podaci su prenošeni u asinhronizovanom formatu, bez clock signala.

CAN specifikacija koristi nazive dominantni i recesivni bitovi. Dominantni je logička 0 (aktivno nošeni prema voltaži preko transmitera), dok su recesivni logička 1 (pasivno vraćeni voltaži preko otpornika). Stanje mirovanja je reprezentovano recesivnim nivoom (logička 1). Ako jedan čvor prenosi dominantni bit, a drugi čvor prenosi recesivni bit, onda nastaje kolozijska i dominantni bit pobjeđuje.

To znači da neće biti kašnjenja kod prouke koja je visokog prioriteta i čvor koji prenosi poruku manjeg prioriteta, automatski pokušava da je prenese opet preko šest bitnog clocka nakon kraja dominantne poruke. Tačna voltaža za logičke 0 i 1 ovisi od fizičkog sloja koji je korišten, ali osnovni princip CAN-a kaže da svaki čvor sluša podatke na mreži uključujući i same sebe. Ako je logička 1 prenošena sa svim čvorovima u isto vrijeme, onda je logička 1 viđena od strane svih čvorova, uključujući i one što se prenose i šalju, a ako se logička 0

prenosi u isto vrijeme od svih čvorova, onda je 0 viđena od strane svih čvorova.

Ima i slučaj gdje se logička 0 prenosi sa jednim ili više čvorova i logička 1 se prenosi sa jednim ili više čvorova. Onda je logička 0 viđena od strane svih čvorova uključujući čvorove koji prenose logičku 1. Kada čvor prenosi logičku 1, a vidi 0, onda shvata da ima kontestovanje, te gubi arbitraciju i odustaje. Nakon što izgubi arbitraciju, određeni čvor ponovo ulazi u red i pokušava svoju poruku slati kasnije opet i stream CAN poruka nastavlja bez greški sve dok jedan čvor ne preostane u prijenosu.

Na sljedećoj slici ćemo vidjeti primjer gdje imamo 11-bitnu ID CAN mrežu, sa dva čvora od dva ID-a od 15 i 16, gdje se dva čvora prenose u isto vrijeme.

	Start bit	ID bits											The rest of the frame
		10	9	8	7	6	5	4	3	2	1	0	
Node 15	0	0	0	0	0	0	0	0	1	1	1	1	
Node 16	0	0	0	0	0	0	0	1	Stopped Transmitting				
CAN data	0	0	0	0	0	0	0	0	1	1	1	1	

Slika 1.1: 11-bitna CAN mreža³

Na slici 2.1 se vidi da se ID bit 4 prenosi, čvor sa ID 16 prenosi 1 (recesivni) i čvor sa ID 15 prenosi 0 (dominantni). Kad se ovo desi čvor ID 16 zna da je prenio 1, ali vidi 0 i shvati da je kolizija i da je izgubio arbitraciju. Čvor sa ID 16 prestane prenositi, što dozvoljava čvoru ID 15 da nastavi prenos bez ikakvog gubitka podataka. Čvor sa najmanjim ID-em će uvijek pobijediti proces arbitracije i ima najveći prioritet.

2.2 Alokacija ID-a

³ https://en.wikipedia.org/wiki/CAN_bus

ID poruka mora biti unikatna na jednoj CAN sabirnici, u drugom slučaju dva čvora moraju nastaviti prijenos preko arbitracijskog polja uzrokujući grešku. U ranim 1990-im izbor ID-ova za poruke je bio jednostavno na bazi identificiranja tipa podataka i šaljući čvorova. Naravno, kako je ID korišten kao prioritet poruke, onda je ovo dovodilo do slabih real-time performansi.

2.3 Tajmiranje bitova

Svi čvorovi na CAN mreži moraju da rade na istoj nominalnoj brzini prijenosa, prijenosu faze, toleranciji oscilatora i driftu oscilatora, što nam kaže da stvarna brzina prijenosa ne mora biti i nominalna brzina prijenosa.

2.4 Slojevi CAN-a

CAN protokol može biti rastavljen na sljedeće apstrakcijske slojeve:

- a) aplikacijski sloj;
- b) objektni sloj;
- c) prijenosni sloj.

Prijenosni sloj je apliciran na većinu CAN standarda. Prijenosni sloj prima poruke od strane fizičkog sloja i prenosi te poruke na objektni sloj.

Prijenosni sloj je zadužen za:

- a) ograničavanje greški;
- b) detektovanje greški;
- c) validacija poruke;
- d) ACK;
- e) arbitracija;
- f) poruke;

- g) brzina prijenosa;
- h) kruženje informacija.

Fizički sloj se sastoji inače od pinova, koji su inače pin2 što predstavlja CAN-Low (CAN-), pin 3 predstavlja GND (ground), pin 7 predstavlja CAN-High (CAN+) i pin9 CAN V+ (napajanje).

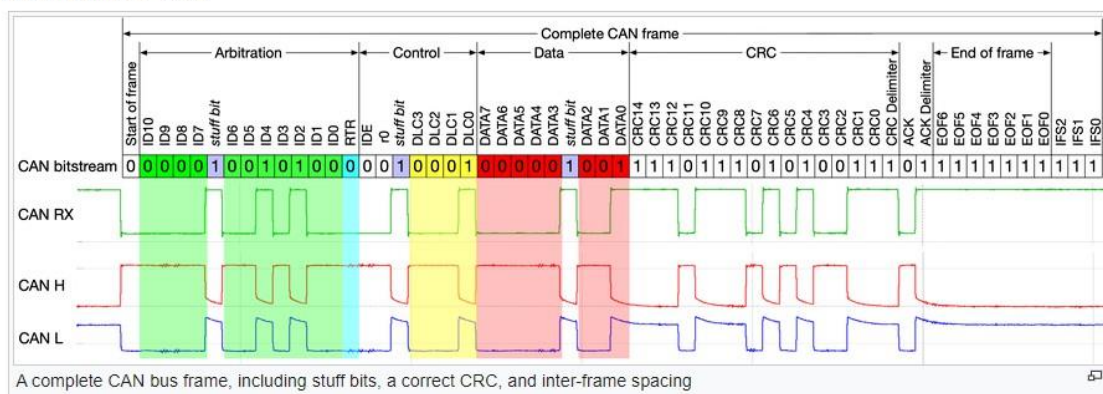
2.5 Format podataka (poruke)

Ovo je jedini format za stvarni prenos podataka. Imamo dva formata poruka:

- a) osnovni format sa 11 bitova identifikatora;
- b) ekstenzovani format sa 29 bitova identifikatora.

Implementacija CAN standarda zahtijeva da se mora prihvatiti osnovni izgled poruke, a onda može naknadno da prihvati ekstenzovani format, ali mora tolerisati ekstenzovani okvir poruke.

Base frame format [edit]



Slika 2.2: Base frame format⁴

⁴ https://en.wikipedia.org/wiki/CAN_bus

Na slici 2.2 možemo da vidimo različite dijelove okvira poruke:

1. Start of frame: sastoji se od 1 bita te pokazuje početak prenosa poruke
2. Identifier: sastoji se od 11 bita , i pokazuje na prioritet poruke
3. Stuff bit : 1 bit i to je bit koji je suprotnog polariteta da bi se održala sinhronizacija
4. RTR : 1 bit mora biti (0) za podatke u poruci a recesivni (1) da bi se remote zahtjevi poruke održali
5. IDE: mora biti: 1 bit i ovo je ekstenzovani bit i mora biti (0) dominantni za osnovni format poruke sa 11- bit identifikatore
6. R0: reserved bit mora biti (0) ali može biti prihvaćen kao (0) ili (1)
7. DLC: dužina podataka (0 – 8 bajtova)
8. Data field: podaci od 0 – 64 ili 0-8 bajtova
9. CRC: 15 bitova i ovo je ciklično provjeravanje redundancije
10. CRC delimiter: mora biti recesivna (1)
11. ACK slot: Transmitter šalje (1) i bilo koji primatelj može prihvatiti (0)
12. ACK delimiter: mora biti (1)
13. EOF: kraj okvira i mora biti recesivna (1)
14. Inter-frame spacing: isto mora biti recesivna (1)

Uz sve navedeno imamo i ekstenzovani format poruke.

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier A (green)	11	First part of the (unique) identifier which also represents the message priority
Substitute remote request (SRR)	1	Must be recessive (1)
Identifier extension bit (IDE)	1	Must be recessive (1) for extended frame format with 29-bit identifiers
Identifier B (green)	18	Second part of the (unique) identifier which also represents the message priority
Remote transmission request (RTR) (blue)	1	Must be dominant (0) for data frames and recessive (1) for remote request frames (see Remote Frame , below)
Reserved bits (r1, r0)	2	Reserved bits which must be set dominant (0), but accepted as either dominant or recessive
Data length code (DLC) (yellow)	4	Number of bytes of data (0–8 bytes) ^[a]
Data field (red)	0–64 (0–8 bytes)	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)
Inter-frame spacing (IFS)	3	Must be recessive (1)

Slika 2.3: Ekstendovani format poruke.⁵

Na slici 2.3 možemo da vidimo male razlike kod ekstendovanog okvira poruke, a dodatni slotovi su SSR (substitute remote request) koji mora biti recisivna 1 i umjesto jednog rezervisanog bita u ovom dijelu imamo 2 r1 i r0. Još jedna od razlika jeste da je identifikator 18 bitova umjesto 11 bitova.

2.6 Remote poruke

U generalnom smislu, prijenos je izvršavan na autonomnoj bazi sa izvornim čvorom (naprimjer - senzor) šaljući podatke, no isto tako je moguće zahtijevati podatke sa izvora putem remote okvira. Dvije glavne razlike su da je RTR bit prenošen kao dominantni (0) bit u običnom okviru podataka, a druga je da u remote okviru nema polja za podatke. DLC polje pokazuje dužinu podatka zahtijevane poruke preko remotea, ali ne i prenošene poruke. RTR je 0 u okviru za podatke dok u remote okviru RTR je 1 recisivni.

⁵ https://en.wikipedia.org/wiki/CAN_bus

2.7 Poruka greške

Poruka za grešku se sastoji od dva polja:

- a) flagovi za grešku (6 – 12 dominantni/recesivni bitovi) dodani sa različitih stanica;
- b) delimiter za grešku (recesivnih bitovi).

Aktivni flag za grešku ima šest dominantnih bitova, prenošen je sa čvorom koji detektuje grešku na mreži da je u stanju greške.

Pasivni flag za grešku ima šest recesivnih bitova i prenošen je sa čvorom koji detektuje aktivnu grešku koja je u stanju pasivne greške.

Imamo i 2 brojača za brojanje greški u CAN-u:

- a) TEC (transmit error counter);
- b) REC (receive error counter).

Same riječi i kažu - greške u prenosu i greške u primanju poruka.

Ako je TEC ili REC između 127 i 255 onda se šalje pasivni okvir greške. Ukoliko su TEC i REC manji od 128, onda će se aktivni okvir greške prenijeti na bus (sabirnicu), te ukoliko TEC je veći od 255 onda čvor ulazi u Bus Off stanje gdje se neće ništa slati.

2.8 Overload

Ovaj okvir se sastoji od 2 polja:

- a) overload flag;
- b) overload delimiter.

Ovo je vrlo slično kao i što je kod greške, između ostalog, imamo dva uslova koja će dovesti do prijenosa overload flaga, a to su:

- a) unutrašnji uslov resivera, koji zahtijeva zastoј sljedeće poruke;
- b) detekcija dominantnog bita za vrijeme prekida.

Početak overload okvira, zbog slučaja pod broj jedan, jedino je dozvoljen da bude započet na prvom bitu očekivanog prekida, dok u slučaju 2 počinje jedan bit nakon detektovanja dominantnog bita.

2.9 ACK slot i izmeđuokvirno razdjeljivanje

Acknowledge slot je korišten da se prihvati validna CAN poruka. Svaki čvor koji prihvati poruku bez pronalaženja greške prenosi dominantni nivo u ACK i prepisuje recesivni nivo transmitera. Ako transmitter pronađe recesivni nivo u ACK, onda zna da nijedan resiver nije pronašao validnu poruku. Čvor koji prihvata poruke može prenijeti recesivni bit da pokaže da nije prihvatio validnu poruku, ali isto tako druga poruka koja je primila validnu poruku može da prepíše ovo sa dominantnim bitom (1). Rješenje inače bude da se ponovo pošalje poruka nakon što je pokazano da nije validna. Ovo može dovesti do ulaza u pasivno stanje greške.

Što se tiče izmeđuokvirnog razdjeljivanja, podaci koji su standardni i remote podaci su razdvojeni sa prethodnim okvirima sa poljem nazvanim izmeđuokvirno polje (interframe space). On se sastoji od najmanje tri uzastopna recesivna (1) bita. Ako je dominantan bit detektovan, onda će biti označen kao Start of frame bit za sljedeću poruku.

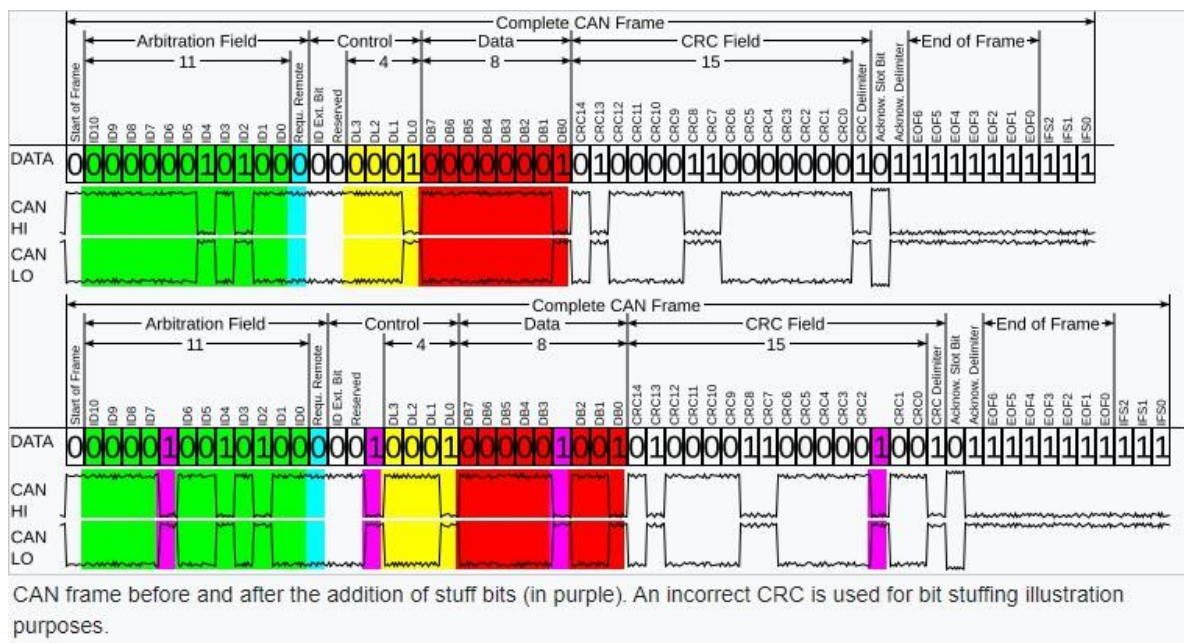
2.10 Bit stuffing

Da bi se ustanovilo dovoljno prijenosa da bi se održala sinhronizacija, bit koji je suprotne polarizacije, tj. bit specifičan za to je unesen nakon 5 uzastopnih bita iste polarnosti. Ova praksa se inače i zove „bit stuffing“ i inače je potrebno zbog NRZ (non return to zero) kodiranja koji je korišten od strane CAN-a. Sva polja u poruci su „stuffed“, osim izuzetaka kao što su CRC delimiter, ACK polje i EOF polje. Oni su fiksirane veličine i nisu „stuffed“. Aktivno stanje greške ima 6 uzastopnih dominantnih bitova i krši pravilo navedeno gore.

Bit stuffing inače znači da podaci mogu biti veći nego što bi neko očekivao.

Veličina osnovne poruke je ograničena sa:

$$8n + 44 + \left\lfloor \frac{34+8n-1}{4} \right\rfloor$$



Slika 2.4: CAN okvir prije i poslije dodavanja stuff bitova.⁶

Na slici 2.4 možemo vidjeti kako u suštini nakon bit stuffing u rozim poljima poruka izgleda. U suštini ubacujemo dodatne bitove da bi se ostvarila sinhronizacija, a i sa prethodnom jednačinom dajemo sebi za pravo proširenja poruke koja može ići do određene granice.

2.11 CAN bus

Šta je ECU?

Electronic

Control

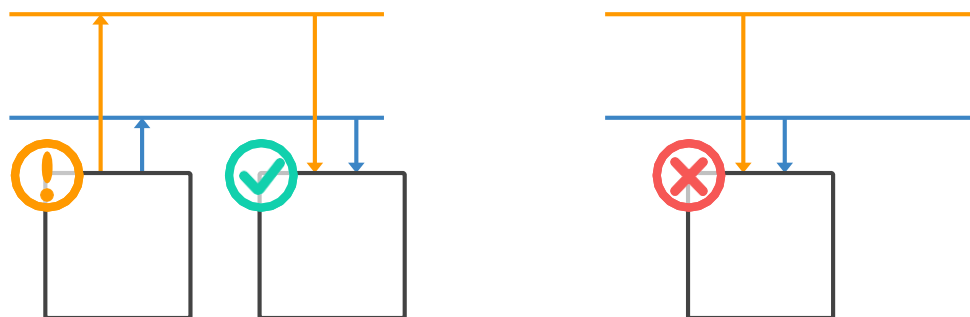
Units

(ECU) su komponente koje kontrolišu određene funkcije - naprimjer, upravljačke jedinice motora, mjenjače, kočnice, upravljanje, temperature itd. Moderan automobil lahko može imati više od 70 ECU-a - svaki dijeli informacije s drugim ECU-ima u autobusu.

Svaki uređaj na CAN sabirnici može pripremiti i prenijeti informacije (npr. podaci senzora).

⁶ https://en.wikipedia.org/wiki/CAN_bus

Poslane podatke primaju I svi ostali uređaji na mreži - i svaki uređaj može pregledati podatke I odlučiti hoće li ih primiti ili zanemariti.



Slika 2.5: ECU⁷

CAN bus DB9 konektor

Kako se spaja na CAN sabirnicu?

Ne postoji standardni konektor za aplikacije CAN sabirnice. Kao što ćemo kasnije pokazati, to implicira da različita vozila/mašine mogu koristiti različite konektore.

Međutim, bliski kandidat je CAN DB9 (D-sub9) konektor (CANopen CiA 303-1), koji je postao standard za mnoge aplikacije uključujući CAN bus zapisivače podataka.

Varijante CAN sabirnice

Prije nego što nastavimo, korisno je znati da postoji više varijanti CAN-a:

- c) CAN male brzine: CAN otporan na greške je jeftina opcija kada je tolerancija kvara kritična - ali se sve više zamjenjuje LIN sabirnicom;
- d) Brzi CAN: Klasični CAN je najčešća varijanta danas u automobilske industriji/mašinama (i fokus ovog članka);
- e) CAN FD: Nudi dužu nosivost i veću brzinu, iako je usvajanje i dalje ograničeno - saznajte više u našem CAN FD uvodu;
- f) CAN XL: Nudi još dužu nosivost i veću brzinu za spajanje između CAN-a i Automotive Ethernet (100BASE-T1).

⁷ <https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial>



Slika 2.6: CAN DB9 bus connector⁸

3. CAN standardi i protokoli

3.1 CAN standardi za niže nivoe

ISO 11898 serija: specificira fizički sloj i link podataka sloj serijske komunikacije.

ISO 11898-1:2015: specificira DLL (data link layer) i fizičko signaliziranje CAN-a.

ISO 11898-2:2016: specificira visoko brzinske transmisije.

ISO 11898-3:2006: niske brzine, ali je tolerantan na greške.

ISO 11898-4:2004: specificira komunikaciju tipa okidač na vrijeme.

ISO 11898-5:2007: veće brzine fizičkog sloja.

ISO 11898-6:2013: za putna vozila veće brzine fizičkog sloja.

ISO 16845-1:2016: metodologija za provjerovanje implementacije CAN-a.

ISO 16845-2:2018: dodaje uslove za testiranje i zahtjeve da se realizuje plan za CAN transiver.

DBC: ovi fajlovi su ASCII fajlovi koji definišu poruke poslane preko CAN-a.

3.2 CAN bazirani protokoli za više nivoe

⁸ <https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial>

Can standard inače ne uključuje osnovne osobine komunikacije. Pa kao jedan od primjera je za putnička vozila gdje svaki proizvođač ima svoj zaseban standard.

CAN in Automation (CiA) je organizacija koja proizvodi i podržava CAN bazirane protokole za više brzine.

Standardni pristupi (navest ćemo neke od njih):

- a) ARINC 812 (avijacijska industrija);
- b) CANopen (industrijska automatizacija);
- c) ISOBUS (poljoprivreda);
- d) ISO-TP (transport protokol);
- e) SAE J1939 (za autobuse i traktore);
- f) SAE J2284 (za putnička vozila);
- g) UAVCAN (avioni i robotika);
- h) CSP (protokol za svemirske alate).

4. MQTT

4.1 Općenito

MQTT nosi naziv kao Message Queuing Telemetry Transport, te služi za prijenos poruka, tačnije, to je protokol. Koristi se publish/subscribe mehanizam. Taj mehanizam se zasniva da se može subscribeovati na neku temu ili da se objavi neka tema. Što se tiče standarda imamo ISO/IEC PRF 20922. Ovaj standard je korišten za IOT i home automation.

MQTT se sastoji od dva elementa sistema:

- a) klijent je čvor koji se povezuje na brokera, te je zadužen da se prijavi ne temu ili da šalje podatke;
- b) broker je server koji vodi evidenciju o svim klijentima.

4.2 Teme

Poruke koje se razmjenjuju korištenjem MQTT-a se objave na neku temu. Nije potrebno konfigurisati teme, nego se navode prilikom objavljivanja poruke. Teme se imenuju hijerarhijski te se koristi slash kao razdvajač „/“ .

Neki od primjera: can/poruke ili mqtt/poruke (što će biti korišteno kasnije u aplikaciji).

Neki od specijalnih znakova:

- a) # zamjenski znak za sve niže nivoe;
- b) # znak za sve vrijednosti na određenom nivou.

Neki od primjera su:

- c) Canserver1/+/poruke → sve poruke na prvom can serveru;
- d) Canserver1/# → svi senzori za Canserver1.

4.3 QoS (Quality of Service)

MQTT ima tri nivoa QoS:

- a) **1 (at least once)** → poruka se nastavlja slati sve dok se nije primila potvrda od strane subscribeovanih klijenata;
- b) **2 (exactly once)** → poruka se garantirano isporučuje pretplaćenom klijentu tačno jedanput;
- c) **0 (at most once)** → poruka se šalje samo jednom, bez potvrde prije od subscribeovanih klijenata.

Bitno je za napomenuti da sve implementacije klijenata i brokera ne podržavaju sva tri nivoa QoS, ali su 0 i 1 uvijek podržani.

Postavljanjem flaga *retained* je moguće da se specificira da broker zadrži poruku na datu temu, te tako klijent koji se pretplati može dobiti aktuelnu vrijednost, tj. čim se pretplati. Isto tako prilikom povezivanja na broker klijent postavlja ili resetuje clean flag.

Ako je clean 0 onda će sve poruke sa QoS 1 i QoS2 da budu pohranjene na brokeru i isporučene nakon što se uspostavi konekcija, dok ako je clean 1 onda će sve poruke biti odbačene. Isto tako klijent može definirati i will. To je poruka koja se šalje na datu temu ukoliko dođe do prekida konekcije.

4.4 MQTT brokeri

Neke od implementacija MQTT brokera su:

- a) Mosquitto;
- b) ActiveMQ;
- c) Eclipse IoT;
- d) HiveMQ.

Broker korišten u našoj aplikaciji bit će HiveMQ.

4.5 MQTT klijenti

Što se tiče klijenata za browsere, imamo:

- a) HiveMQ Websocket Client (online websocket klijent);
- b) MQTTLens (za google chrome).

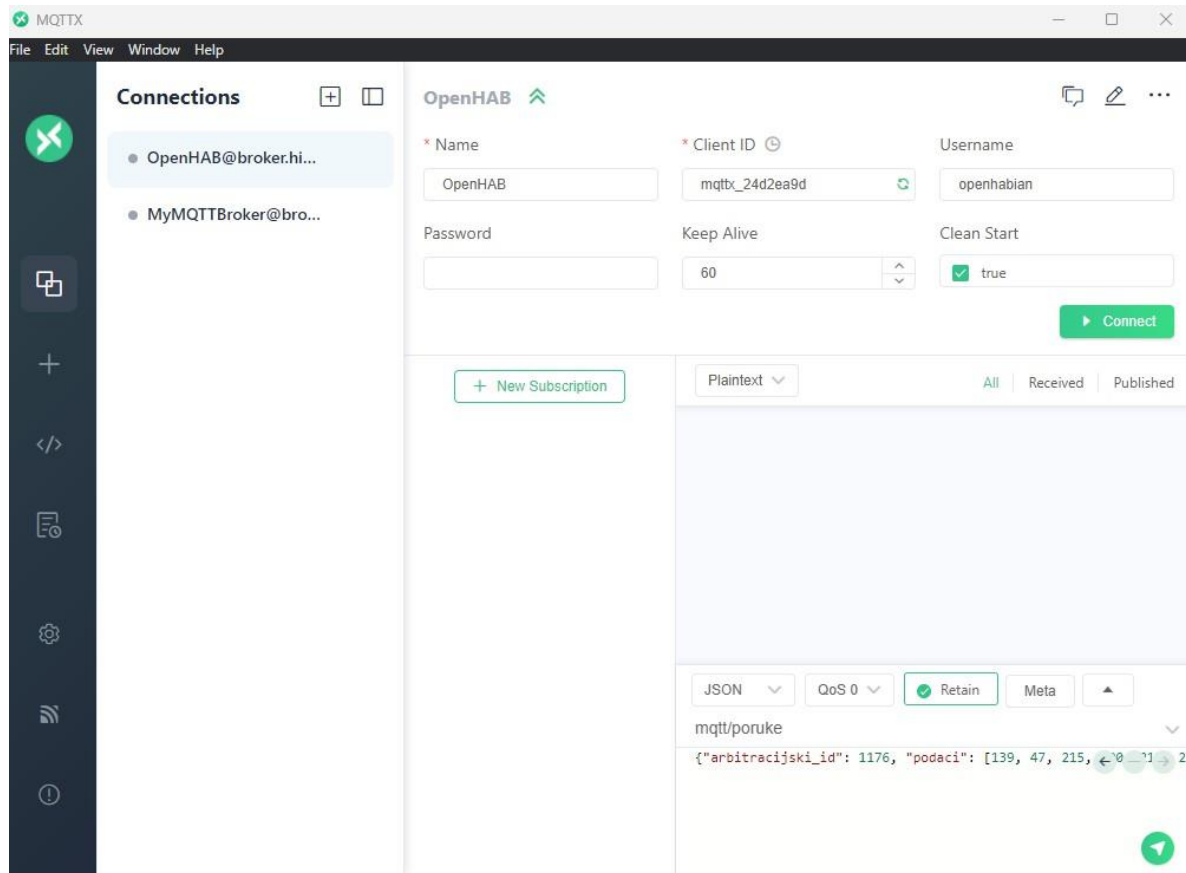
Što se tiče klijenata za desktop:

- c) MQTT -X;
- d) MQTT Explorer.

Te što se tiče klijenata za Android i IOS:

- e) MQTT Dash;

- f) MQTT Client;
- g) MyMQTT.



Slika 4.1: Pokretanje MQTT-X desktop verzije.

Na slici 4.1 može se vidjeti pokretanje MQTT-X desktop verzije, koju ćemo i mi ujedno koristiti za aplikaciju. Kao što se vidi i na slici imamo različite postavke i za QoS i za subskripcije, teme i šta se nalazi u poruci.

4.6 MQTT kao protokol za IOT

Organizacija komunikacije se gradi u više slojeva, ovakva se organizacija naziva mrežnom arhitekturom.

ISO je predložio apstraktni opis arhitekture mreže u formi ISO OSI

open system interconnection). Model koji ima 7 slojeva. Funkcije slojeva mogu biti implementirane ili hardverski ili softverski.

U datoj aplikaciji se implementiraju samo oni slojevi za kojima postoji potreba.

Svaki sloj pruža usluge višem sloju i koristi usluge nižeg sloja.

Komunikacija se realizira između entiteta na istom nivou, korištenjem komunikacijskih protokola. Kako se realizira komunikacija između PC računara i modula sa mikrokontrolerom putem serijskog porta? Komunikacija će biti realizirana po principu master-slav. Master adresira, određišni uređaj prosljeđuje mu komandu i parametre komande. Slave izvršava traženu komandu i šalje masteru odgovor. Postoje samo dvije komande, a svaka je dužine 4 bajta

Također postoji i komanda za postavljanje vrijednosti na port, te komanda za očitavanje vrijednosti sa jednog od analognih ulaza.

#ASA – ispisuje 0x41 na port

#AR0 – očitava vrijednost sa AN0

Prijemnik za opisani protokol možemo opisati konačnim automatom:

Stanja:

S1 – čekanje početka poruke

S2 – čekanje adrese

S3 – čekanje komande

S4,S5- čekanje parametra

ULAZI

B1 – početak poruke(„#“)

B3 adresa modula (0 – 255)

B6 ispis na PORTB(„S“)

B7 čitanje A/D ulaza („R“)

B8 – bajt za ispit

B9 broj ad ulaza

5. CAN server

5.1 Razvojno okruženje generisanja CAN poruka

Što se tiče generisanja poruka, CAN inače radi preko hardvera, jer se sve okruženje u suštini se zasniva na porukama koje se dobijaju sa servera koji se nalaze u automobilima i ostalim okruženjima. Python programski jezik podržava različita okruženja, a između i ostalog mi ćemo navesti neka:

- a) CANalyst-2 je USB na CAN Analyzer uređaj koji je proizveden od strane Chuangxin Technology, instalirava se komandom: `pip install "python-can[canalystii]"`. Ovaj uređaj ima limitacije i koristi, jedna od njih je da USB protokol prenosi poruke grupisane preko kanala. Sve poruke primljene od strane kanala 0 i kanala 1 mogu se vratiti od strane softvera bez pažnje na red između dva kanala;
- b) Kvaser je objekat sa fizičkim CAN Busom koji može biti operisan u 2 moda:
 - a. `Single_handle` mod sa jednim zajedničkim busom korišten za čitanje i pisanje na CAN bus;
 - b. `Two_separate_bus_handles` su potrebni za primanje i slanje poruka u različitim threadovima;
 - c. Kvaser se koristi za Windows operativni sistem, ali isto tako se može i koristiti za Linux;
 - c) CAN over Serial je interfejs baziran na tekstu. Naprimjer, koriste se kao `/dev/ttyS1` ili `dev/ttyUSB0` na Linuxu ili `COM1` na Windows. Isto tako, remote portovi se mogu koristiti putem specijalnog URL-a;
 - d) SocketCAN se koristi specifično za Linux mašine. Međutim, u našem slučaju mi ćemo pokušati da primijenimo SocketCAN na Windows mašini korištenjem WSL2 da prikazemo virtuelno generisanje poruka kao `vcan0`.

5.2 WSL2/Ubuntu

Da bismo koristili SocketCan preko Windowsa potrebno nam je da koristim WSL2/Ubuntu. Zbog nemogućnosti virtuelnog generisanja poruka preko Windowsa, ovo nam je jedan od načina koji je vrlo koristan zbog toga što nekad ne možemo biti u stanju da imamo hardver pri ruci, te ukoliko pratimo korake i instalacije može nam znatno olakšati posao. Zbog ovog problema, dok ne instaliramo sljedeće korake, ne možemo koristiti alate poput `canutils` na Windows mašinama, i ovo može biti vrlo frustrirajuće.

WSL je Windows podsistem za Linux. Ovo je kompatibilni sloj dodan od strane Microsofta da se pokrene Linux executable preko Windowsa. U suštini, ovo je kompletno virtuelna mašina i može se pristupiti Linux terminalu preko jednog klika.

WSL2 je dosta bolji od originalnog WSL-a jer ima manje bugova, brži je, više versatilan i koristi u suštini pravi Linux kernel.

Linux kernel updateovi se ispuštaju od strane Windows 10 softvera i na dalje.
Naravno, što se tiče performansi isto je bolje, i stvara se puna kompatibilnost.

Neki od koraka kako instalirati WSL2 na Windowsu su:

- a) da iskoristimo command prompt u Windowsu i ukucamo `wsl.exe --install`;
- b) isto tako `apt update && apt upgrade`.

Nakon toga se pokrene ponovo sistem, te imamo instaliran WSL2, naravno tu ima još različitih pristupa kako instalirati, međutim nama će ovo biti sasvim dovoljno.

5.3 Postavljanje virtuelne CAN mreže u WSL2

Prije svega moramo znati da WSL2 ne podržava CAN networking odmah po defaultu, jer mi ne možemo odmah kreirati virtuelni CAN interfejs na prvu. Da bismo ovo izbjegli, mi trebamo pokrenuti naš customizirani kernel u WSL2.

Koraci za kreiranje customiziranog kernela:

1. `sudo apt-get update -y`
2. `git clone https://github.com/microsoft/WSL2-Linux-Kernel`

```
3. cat /proc/config.gz | gunzip > .config
4. make prepare modules_prepare
5. make menuconfig
6. make -j4
7. sudo make modules_install
8. cp vmlinux /mnt/c/
9. cat >> /mnt/c/.wslconfig << "ENDL"
   [wsl2]
   kernel=C:\\vmlinux
   ENDL
10. wsl -shutdown
11. sudo modprobe can
12. sudo modprobe can-raw
13. sudo modprobe vcan
14. sudo ip link add dev vcan0 type vcan
15. sudo ip link set up vcan0
16. sudo apt install can-utils
```

Neki od koraka će biti u nastavku objašnjeni.

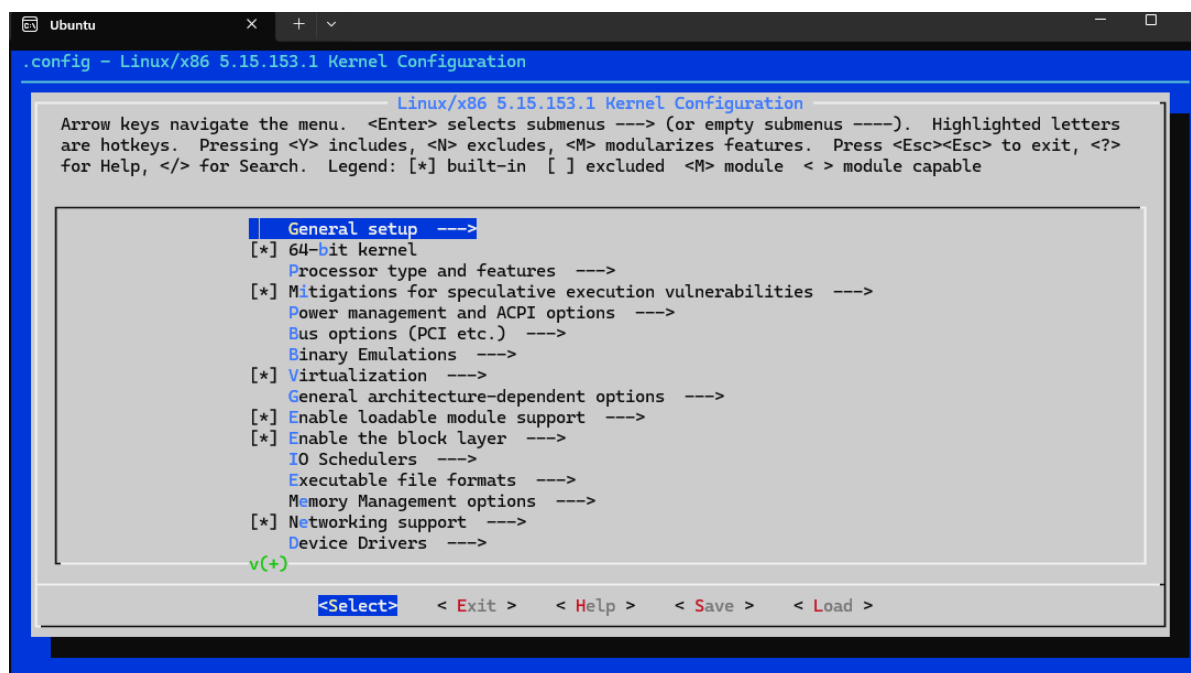
Korak 1. nam služi za update našeg okruženja.

Korak 2. nam služi da kloniramo WSL2 project na Githubu na naše lokalno okruženje. Moramo voditi računa da nam je ista verzija našeg WSL2 Linux Kernela i WSL2 ista. Možemo je dobiti tako što kucamo u terminalu (Ubuntu), kucamo `uname-r`, te će nam izbaciti trenutačnu verziju WSL-a. Tako svrsishodno instaliramo isto i Linux kernel

Korak 3. nam vraća informaciju kernel konfiguracije iz jednog fajla i sačuva je u drugi fajl naziva `.config`. Na osnovu ove informacije, mi ćemo izgraditi naš custom kernel.

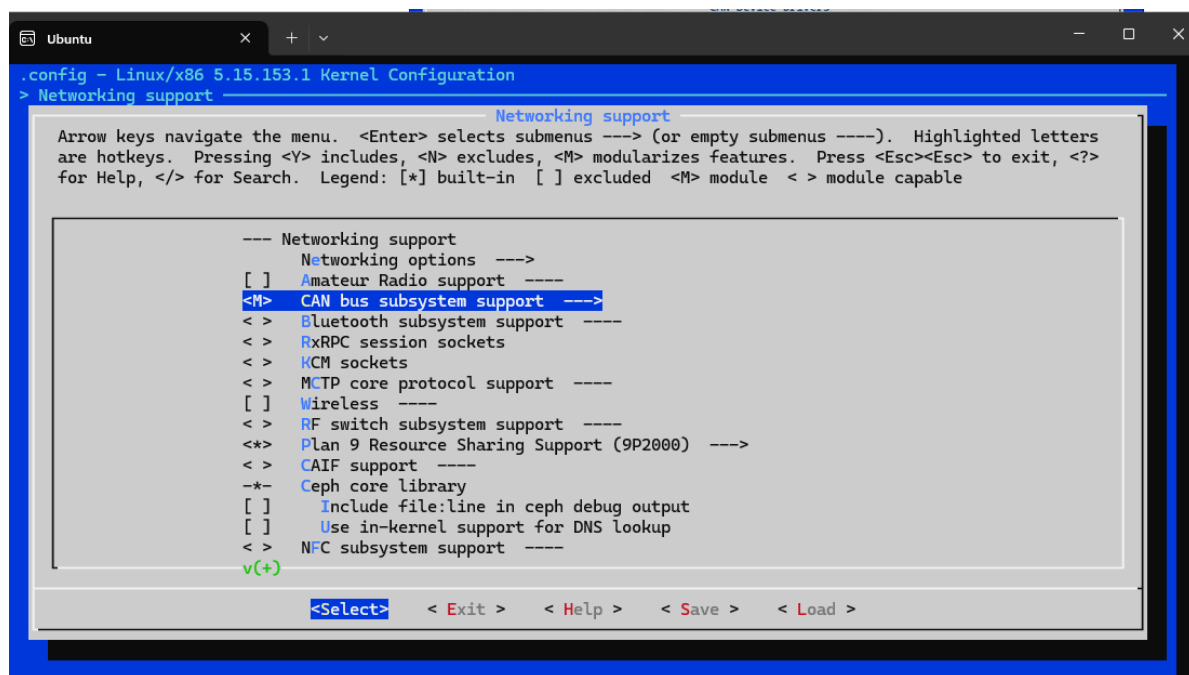
Korak 4. Pripremamo naš izvor kernela da bude u odgovarajućem stanju da se može pokrenuti.

Korak 5. je malo zahtjevniji, te će nam se prikazati konfiguracija gdje moramo pronaći CAN postavke, što ćemo vidjeti na sljedećih par slika.



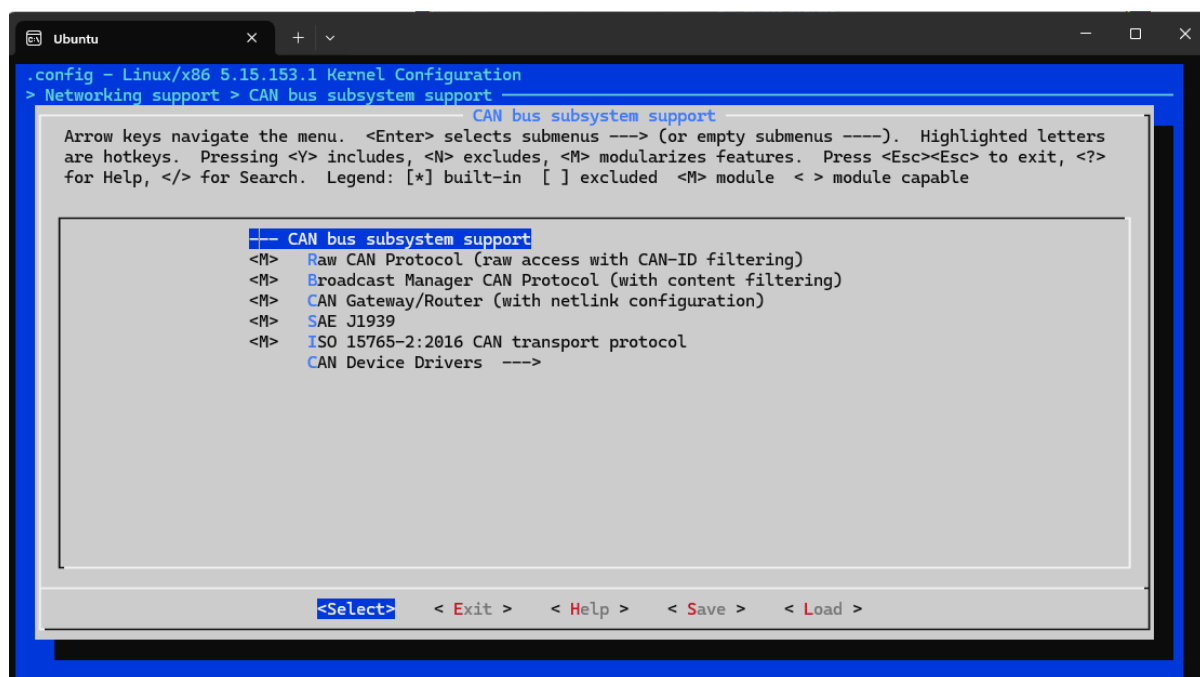
Slika 5.1: postupak za pronalaženje CAN postavki.

Ovdje na slici 5.1 moramo pronaći Networking support, gdje ćemo pronaći CAN.



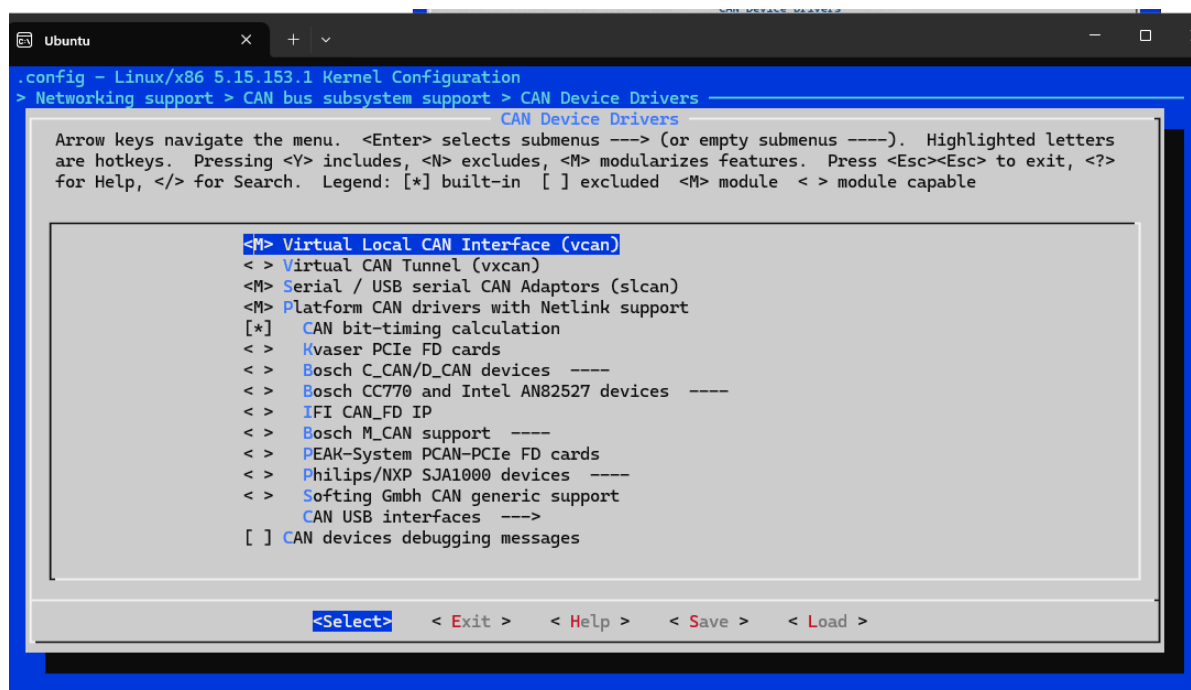
Slika 5.2: Postupak za pokretanje podrške za CAN podsistem.

Zatim kliknemo na podsistem podršku za CAN na slici 5.2. Ukoliko ne stoji M (u našem slučaju nije bilo, moramo unijeti M da bi se otvorila podrška za CAN podsistem).



Slika 5.3: Unošenje željenog interfejsa ili protokola.

Na slici 5.3 moramo unijeti M gdje želimo da imamo podršku za određeni željeni interfejs ili protokol koji nam je potreban. U našem slučaju ovaj korak nije potreban, ali možemo neke CAN-ove odabrati ukoliko želimo i imamo hardver.



Slika 5.4: Postavljanje M za vcan.

Na slici 5.4 potrebno je uključiti (postaviti M) za vcan. Ostali nisu potrebni. Ovo je vrlo potreban korak.

Koraci 6. i 7. će komapjlirati izvorne Linux fajlove.

Korak 8. je kopiranje našem customiziranog kernela u top direktoriju.

Korak 9. nam daje put do WSL2 za naš kernel.

Korak 10. ugasimo i restartujemo WSL2.

Koraci 11. , 12. i 13. je manuelno dodavanje modula koje smo ugradili na WSL2.

Koraci 14. i 15. nam služe da kreiramo interfejs nazvan vcan0, možemo i koristiti ifconfig komandu da vidimo konfiguraciju.

Korak 16. instaliramo can utils i možemo ih koristiti, bitne komande su cangen vcan0 što nam generiše can poruke, dok candump vcan0 prima can poruke i možemo vidjeti koje poruke dobijamo na vcan.

Sad možemo testirati manuelno generisanje poruka korištenjem cangen i candump komandi.

```
abikralj123@DESKTOP-EIHQK3C:~$ cangen vcan0
socket: Address family not supported by protocol
abikralj123@DESKTOP-EIHQK3C:~$ sudo modprobe can
[sudo] password for abikralj123:
Sorry, try again.
[sudo] password for abikralj123:
Sorry, try again.
[sudo] password for abikralj123:
abikralj123@DESKTOP-EIHQK3C:~$ sudo modprobe can-raw
abikralj123@DESKTOP-EIHQK3C:~$ sudo modprobe vcan
abikralj123@DESKTOP-EIHQK3C:~$ sudo ip link add dev vcan0 type vcan
abikralj123@DESKTOP-EIHQK3C:~$ sudo ip link set up vcan0
abikralj123@DESKTOP-EIHQK3C:~$ cangen vcan0
|
```

Slika 5.5: manuelno slanje poruka.

```
abikralj123@DESKTOP-EIHQK3C:~$ candump vcan0
vcan0 753 [7] 26 E8 DC 51 73 68 58
vcan0 5D0 [1] EF
vcan0 2EC [6] 41 06 17 54 1A C9
vcan0 06A [7] 88 D2 BF 32 9A 9F 1A
vcan0 6F9 [8] 30 9D 6B 30 98 D6 02 4E
vcan0 789 [2] 1D 00
vcan0 6F5 [8] 75 F8 A5 44 EE 89 9D 40
vcan0 448 [8] A6 9B D1 45 9B FB 2E 21
vcan0 586 [8] 0E 64 87 6F 57 53 BB 47
vcan0 7CE [8] BC A1 CB 2A BA CA BB 0B
vcan0 123 [8] D5 93 A7 09 8D 29 A3 09
vcan0 135 [8] 27 C9 BD 58 2F 60 35 62
vcan0 0A8 [7] C7 36 38 30 31 38 58
```

Slika 5.6: manuelno primanje poruka.

Na slikama 5.5 i 5.6 možemo vidjeti manuelno slanje i primanje poruka preko vcan0, što nam je omogućilo da generišemo te iste poruke na Windows operativnom sistemu.

6. CAN server u mostu sa MQTT

6.1 Python programski jezik

Da bismo spojili CAN sa MQTT-om, mi trebamo da napravimo kod koji će nam to isto omogućiti. Naš korišteni kod će koristiti Python programski jezik, koji će se moći kompajlirati u python3, ali na Linux podsistemu, što je vrlo bitno za napomenuti. Inače, što se tiče programiranja u Python programskom jeziku, mi vrlo lahko možemo da kod programiramo bilo gdje, što je i ovdje urađeno zbog jednostavnosti. Korišten je Anaconda navigator, te Spyder okruženje gdje je napravljen kod.

Međutim implementacija tog istog koda je pravljena nakon što se instalirao python3 kompajler u Linux Kernelu za WSL , kojem su pridružene sve biblioteke za MQTT te isto tako i CAN, kao što je CAN-utils i paho biblioteka za MQTT.

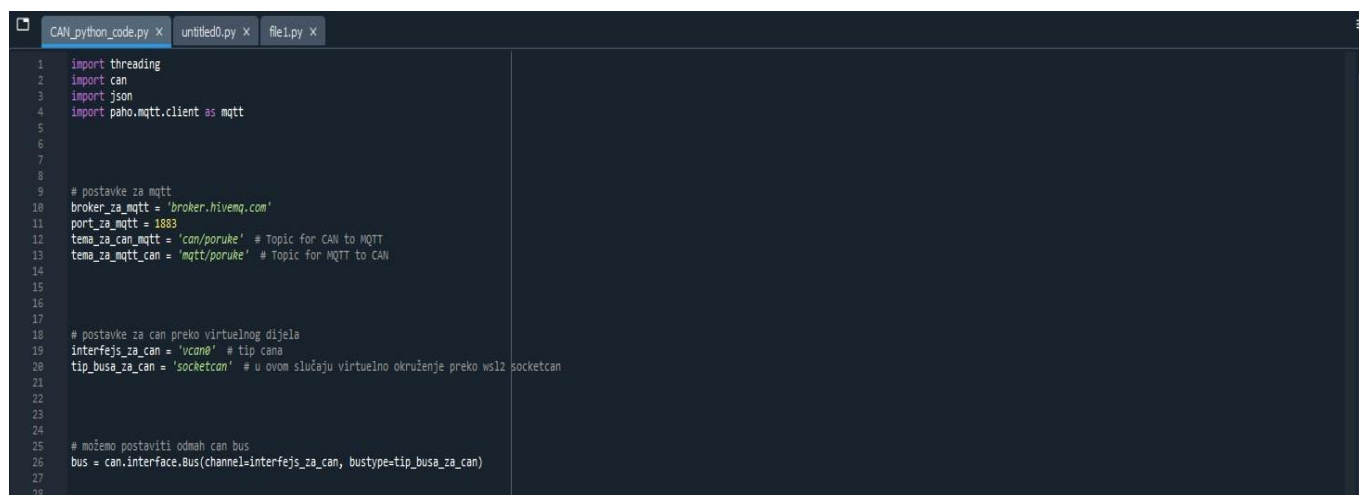
6.2 Python u Linuxu preko Ubuntu/WSL2

Kako bismo pozvali editor, koristimo nano can_program.py, te nakon što iz drugog editora napravimo kod koristeći ctrl + v, dodamo ga na mjesto gdje se nalazi, npr. can_program.py, a u našem slučaju to će biti u Linux Kernel WSL2 direktoriji, zbog kompajliranja.

Na samom kraju samo pokrenemo program komandom python3 can_program.py.

6.3 Prijenos CAN poruka na MQTT i obrnuto

6.3.1 Programski kod u Pythonu za prijenos poruka



```
1 import threading
2 import can
3 import json
4 import paho.mqtt.client as mqtt
5
6
7
8
9 # postavke za mqtt
10 broker_za_mqtt = 'broker.hivemq.com'
11 port_za_mqtt = 1883
12 tema_za_can_mqtt = 'can/poruke' # Topic for CAN to MQTT
13 tema_za_mqtt_can = 'mqtt/poruke' # Topic for MQTT to CAN
14
15
16
17
18 # postavke za can preko virtuelnog dijela
19 interfejs_za_can = 'vcan0' # tip cana
20 tip_busa_za_can = 'socketcan' # u ovom slučaju virtuelno okruženje preko ws12 socketcan
21
22
23
24
25 # možemo postaviti odmah can bus
26 bus = can.interface.Bus(channel=interfejs_za_can, bustype=tip_busa_za_can)
27
28
```

Slika 6.1: Prikaz svih uvedenih biblioteka i postavki za MQTT.

Slika 6.1 nam pokazuje sve uvedene biblioteke, te postavke za MQTT, gdje smo postavili brokera koji je besplatan, određen port koji inače je većinom 1883, a 8883 se koristi za sigurne konekcije koje može značiti ukoliko je sadržaj poruka vrlo bitan, te nakon toga vidimo dvije teme isto kreirane, jedna je za prenos sa CAN-a na MQTT, a druga će biti obrnuto. Te na kraju imamo i interfejs za CAN, u našem slučaju mi koristimo virtuelni interfejs SocketCan, te se postavi bus sa tim postavkama.

```

31
32 def can_na_mqtt():
33     # thread funkcija koja će služiti za poruke preko cana na mqtt
34     klijent_na_mqtt = mqtt.Client()
35
36     def kad_se_poveze_na_mqtt(client, userdata, flags, rc):
37         print(f"Povezana sa brokerom za MQTT sa kodom {rc}")
38         print("Ovaj prijenos će biti za slanje poruka sa CAN na MQTT")
39
40     # postavi mqtt klijenta za slanje poruka za CAN na MQTT
41     klijent_na_mqtt.on_connect = kad_se_poveze_na_mqtt
42     klijent_na_mqtt.connect(broker_na_mqtt, port_na_mqtt)
43     klijent_na_mqtt.loop_start() # započet petlju za klijenta
44
45     while 1:
46         poruka = bus.recv() # primi can poruku sa bus.recv()
47         if poruka:
48             poruka_rjecnik = {
49                 'arbitracijski_id': poruka.arbitration_id,
50                 'podaci': list(poruka.data),
51                 'da_li_je_id_extendovan': poruka.is_extended_id,
52                 'da_li_je_can_fd': poruka.is_fd,
53                 'da_li_je_greska': poruka.is_error_frame,
54                 'vremenska_oznaka': poruka.timestamp
55             }
56
57             poruka_u_json = json.dumps(poruka_rjecnik) # konvertovati u poruku jsona
58
59             klijent_na_mqtt.publish(tema_na_mqtt, poruka_u_json)
60             print(f"Poruka poslana sa CAN-a na MQTT: {poruka_u_json}")
61             print(f"Duzina podataka: {len(poruka.data)}")
62
63
64
65
66
67
68
69

```

Slika 6.2: Funkcija za slanje CAN-a na MQTT, postavljanje poveznice.

Na slici 6.2 imamo funkciju koja šalje CAN na MQTT, te iz ovoga postavimo poveznicu sa CAN-a na MQTT, te zatim uzmemo poruku sa CAN-a i onda pretvorimo tu poruku u rječnik i taj rječnik convertujemo u JSON, te zatim pokažemo te iste poruke, a ujedno su i poslane te poruke na MQTT (u našem slučaju MQTTx).

```

72
73 def mqtt_na_can():
74
75     # threadovana funkcija koja će slati poruke sa MQTTa na CAN
76     def kad_se_poveze_na_can(client, userdata, flags, rc):
77         print(f"Povezana sa brokerom za MQTT sa kodom {rc}")
78         print("Ovaj prijenos će biti za slanje poruka sa MQTT na CAN")
79         client.subscribe(tema_na_mqtt_can) # subskribuj se na temu za slanje poruka mqtt na can
80
81
82
83
84     def kad_se_dobije_poruka(client, userdata, msg):
85         poruka = msg.payload.decode()
86         print(f"Poslana MQTT poruka na CAN: {poruka}")
87
88         # sad rastaviti poruku iz jsona
89         poruka_iz_jsona = json.loads(poruka)
90         id_na_can = poruka_iz_jsona.get('arbitracijski_id')
91         podatak_na_can = poruka_iz_jsona.get('podaci')
92         da_li_je_id_extendovan = poruka_iz_jsona.get('da_li_je_id_extendovan', False)
93         da_li_je_can_fd = poruka_iz_jsona.get('da_li_je_can_fd')
94         da_li_je_greska = poruka_iz_jsona.get('da_li_je_greska')
95         vremenska_oznaka = poruka_iz_jsona.get('vremenska_oznaka')
96
97
98         # postavi i poslati can poruku
99         poruka_na_can = can.Message(
100             arbitration_id=int(id_na_can),
101             data=bytes(podatak_na_can),
102             is_extended_id=da_li_je_id_extendovan,
103             is_fd=da_li_je_can_fd,
104             is_error_frame=da_li_je_greska,
105             timestamp=vremenska_oznaka
106         )
107
108         bus.send(poruka_na_can)
109         print(f"Poslana poruka je: ID={id_na_can}, Podatak={bytes(podatak_na_can).hex()}, Ekstendovan={da_li_je_id_extendovan}, FD={da_li_je_can_fd}, GRESKA={da_li_je_greska}, VREMENSKA_OZNAKA={vremenska_oznaka}")
110
111
112     # postavi klijenta za mqtt
113     klijent_na_mqtt = mqtt.Client()
114     klijent_na_mqtt.on_connect = kad_se_poveze_na_can
115     klijent_na_mqtt.on_message = kad_se_dobije_poruka
116
117     # pozovi se brokerom
118     klijent_na_mqtt.connect(broker_na_mqtt, port_na_mqtt)
119
120
121
122
123
124
125     klijent_na_mqtt.loop_forever() # pocni petlju za mqtt klijenta
126

```

Slika 6.3: Funkcija MQTT na CAN.

Na slici 6.3 funkcija MQTT na CAN, naša funkcija će prvo da se poveze na CAN, zatim pri

slanju poruka sa MQTT-a, subscribeujemo se na specificiranu temu, te zatim nakon što dobijemo poruku decodiramo je za CAN, ali prvo se mora rastaviti iz JSON formata u određene dijelove. Nakon tih koraka pozovemo odgovarajuće funkcije, te povežemo klijenta sa Brokerom i počnemo petlju za MQTT klijenta.

```
126
127
128
129 # zapoceti ova 2 threada
130 thread_ra_can_na_mqtt = threading.Thread(target=can_na_mqtt, daemon=True)
131 thread_ra_mqtt_na_can = threading.Thread(target=mqtt_na_can, daemon=True)
132
133
134 thread_ra_can_na_mqtt.start()
135 thread_ra_mqtt_na_can.start()
136
137
138 def main():
139     thread_ra_can_na_mqtt.join() # dok ovaj thread završi treba sačekati
140     thread_ra_mqtt_na_can.join() # zatim ovaj
141     bus.shutdown() # mora se na kraju ugasiti bus za CAN, inace pravi problem
142
143 if __name__ == '__main__':
144     main()
145
146
```

Slika 6.4: threadovi

Slika 6.4 nam pokazuje da postoje 2 threada za slanje MQTT na CAN i obrnuto, da bi moglo biti istovremeno slanje poruka, držimo main thread i čekamo da se završe oba threada zatim ugasimo bus.

6.4 Simulacija za prijenos poruka

The image shows a terminal window on the left and a MQTT client interface on the right. The terminal displays a series of messages being sent from CAN to MQTT, each containing an arbitration ID, a list of data points, and a timestamp. The MQTT client interface on the right shows a subscription to the 'can/poruke' topic and displays the received messages in a structured format, including the arbitration ID, data points, and timestamp.

Terminal output (left):

```
Poruka poslana sa CAN-a na MQTT: {"arbitracijski_id": 1423, "podaci": [90, 157, 243, 114, 106, 60, 109, 73], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725287616.856912}
Duzina podataka: 8
Poruka poslana sa CAN-a na MQTT: {"arbitracijski_id": 732, "podaci": [156, 238, 183, 78, 42, 214, 0, 37], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725287617.8570338}
Duzina podataka: 8
Poruka poslana sa CAN-a na MQTT: {"arbitracijski_id": 1583, "podaci": [79, 55, 114, 48, 169, 168, 56], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725287617.2571855}
Duzina podataka: 7
Poruka poslana sa CAN-a na MQTT: {"arbitracijski_id": 1254, "podaci": [99, 192, 205, 98, 180, 173, 286, 18], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725287617.4573944}
Duzina podataka: 8
Poruka poslana sa CAN-a na MQTT: {"arbitracijski_id": 1277, "podaci": [281, 245, 117, 59, 9, 119, 77], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725287617.657597}
Duzina podataka: 7
Poruka poslana sa CAN-a na MQTT: {"arbitracijski_id": 635, "podaci": [1, 80, 228, 83], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725287617.857792}
Duzina podataka: 4
Poruka poslana sa CAN-a na MQTT: {"arbitracijski_id": 1975, "podaci": [161, 39, 69, 125, 178, 17, 73], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725287618.0579946}
Duzina podataka: 7
Poruka poslana sa CAN-a na MQTT: {"arbitracijski_id": 1570, "podaci": [5, 74, 161, 127, 177, 3, 237, 74], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725287618.2582011}
Duzina podataka: 8
Poruka poslana sa CAN-a na MQTT: {"arbitracijski_id": 1095, "podaci": [28, 64, 90, 20, 35, 215, 226, 109], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725287618.458364}
Duzina podataka: 8
^Z
[2]+  Stopped                  python3 can_program.py
abikralj123@DESKTOP-EIHQK3C:~$
```

MQTT Client interface (right):

- Subscriptions: can/poruke (QoS 0), mqtt/poruke (QoS 0)
- Received messages (Topic: can/poruke, QoS: 0):
 - Message 1: {"arbitracijski_id": 1570, "podaci": [5, 74, 161, 127, 177, 3, 237, 74], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725287618.2582011} (2024-09-02 16:33:38.354)
 - Message 2: {"arbitracijski_id": 1095, "podaci": [28, 64, 90, 20, 35, 215, 226, 109], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725287618.458364} (2024-09-02 16:33:38.916)

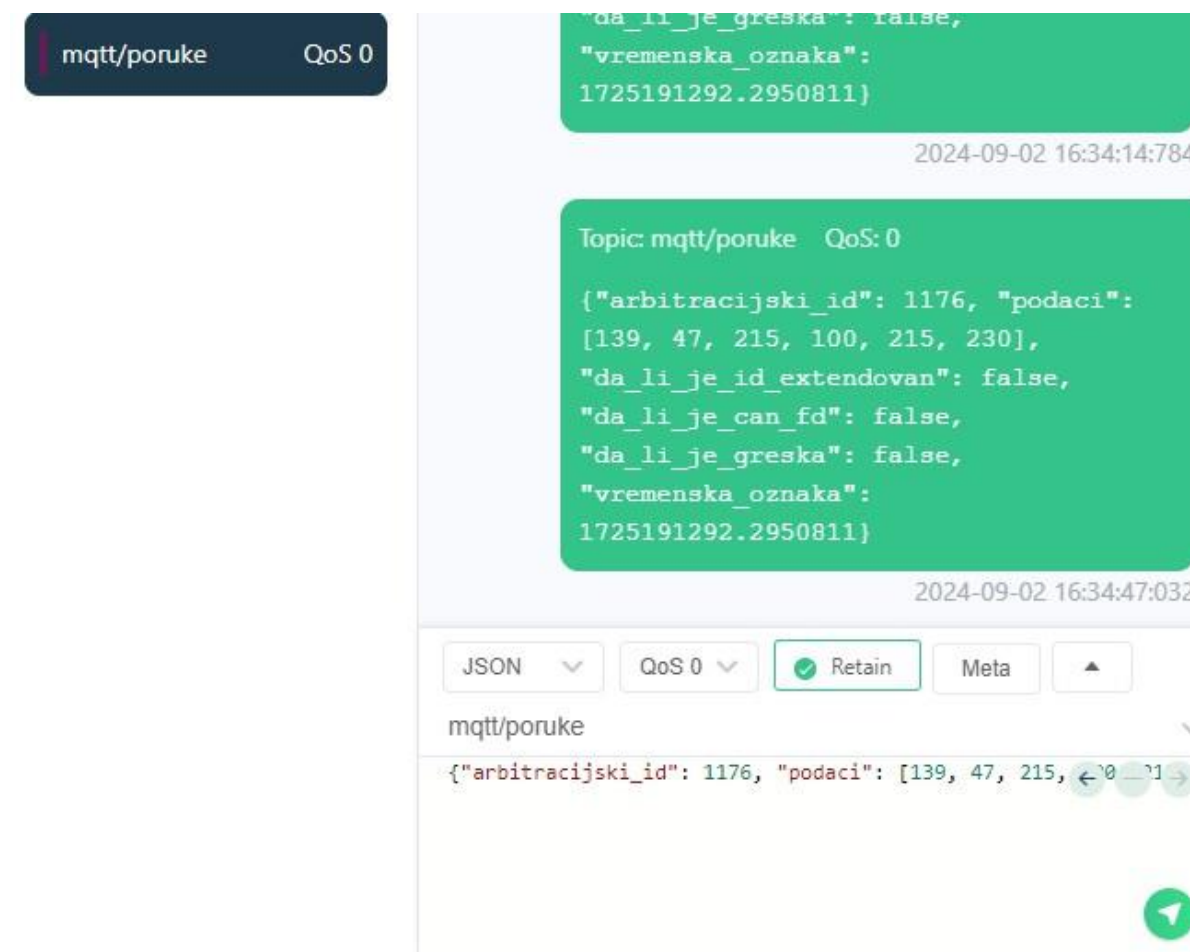
Slika 6.5: Poslana poruka sa CAN na MQTT

Slika 6.5 nam pokazuje da nakon što se pokrene Python program, te kad se šalju poruke preko cangen vcan0, ujedno printamo u Pythonu primljene poruke, te zatim nam se iste pojavljuju na temi can/poruke, koje nakon konektovanja na MQTT-X konstantno dolaze dokle god se pokreće cangen vcan0.

```
Poslana MQTT poruka na CAN: {"arbitracijski_id": 1176, "podaci": [139, 47, 215, 100, 215, 230], "da_li_je_id_extendovan": false, "da_li_je_can_fd": false, "da_li_je_greska": false, "vremenska_oznaka": 1725191292.2950811}
Poslana poruka je: : ID=1176, Podatak=8b2fd764d7e6, Ekstendovan=False, FD=False, GRESKA=False, VREMENSKA_OZNAKA=1725191292.2950811
```

Slika 6.6: Prikaz prenesene poruke sa MQTT-a u Pythonu

Slika 6.6 je predstavila kako u Pythonu prikazujemo prenesenu prouku sa MQTT-a, koja je rastavljena i isprintana.



Slika 6.7: Poslana poruku sa MQTT

Slika 6.7 nam predstavlja poslanu poruku sa teme mqtt/poruke koja je poslana u sličnom formatu kao što je i poslano sa CAN-a na MQTT.



```
abikralj123@DESKTOP-EIHQK3C:~$ candump vcan0
vcan0  498    [6]  8B 2F D7 64 D7 E6
```

Slika 6.8: candump vcan 0 nakon slanja poruke sa MQTT-X na CAN.

Slika 6.8 nam prikazuje kako funkcioniše candump vcan0 nakon slanja poruke sa MQTT-X na CAN, te se dobije slična kao i izvorna poruka u CAN-u.

7. Zaključak

CAN prijenos poruka na MQTT nam omogućuje lakše čitanje poruka, editovanje i dodavanje podataka. Obrnuto prenošenje poruka nam isto omogućava lakše upravljanje nekim vozilima, uređajima itd. Ovakvi prijenosi definitivno olakšavaju komunikaciju sa nekih mašina/vozila koje nisu baš tako lahko pristupačne u generalnom smislu, a isto tako i čitljivije poruke koje će biti u MQTT-u, dok su na CAN-u vrlo nečitke poruke, te potrebna su dekodiranja, zato je i MQTT okruženje vrlo dobro i jednostavno za koristiti.

Mi možemo spojiti više MQTT brokera da omogućimo uređaju na različitim brokerima da komuniciraju jednih sa drugima.

CAN na MQTT mogu biti vrlo korisni i u automotivnoj telemetriji da se monitorišu stvarne metrike vozila kao što su brzina, nivo goriva itd.

Također se koristi u industrijskoj automatizaciji, smart transportaciji...

Sve u svemu, zaključak je da je ovo vrlo moćan alat koji nam može omogućiti lakšu i efikasniju upotrebu navedenih sistema.

8. Reference

1. US (ugradbeni sistemi predavanje Komunikacija 2. dio)
https://c2.etf.unsa.ba/pluginfile.php/52248/mod_resource/content/9/Predavanja_2017/US-Komunikacija%20%282.%20dio%29.pdf
2. Creating a Virtual CAN network in WSL2
<https://berkerturk.medium.com/creating-a-virtual-can-network-in-wsl2-7ccdf166367c>
3. CAN BUS wikipedia
https://en.wikipedia.org/wiki/CAN_bus
4. Python-Can 4.4.2 dokumentacija
<https://python-can.readthedocs.io/en/stable/index.html>
5. WSL/2 instalacija
<https://www.omgubuntu.co.uk/how-to-install-wsl2-on-windows-10>
6. CAN standards
<https://blog.ansi.org/controller-area-network-can-standards-iso-11898/>
7. Introduction to CAN
<https://www.ti.com/lit/an/sloa101b/sloa101b.pdf>