GPT virtualni assistent implementiran na ESP32 platformi

Šćulac, Robin

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Prijediplomski sveučilišni studij računarstva

Završni rad

GPT virtualni asistent baziran na ESP32 platformi

Rijeka, rujan 2023.

Robin Šćulac

0069085473

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Prijediplomski sveučilišni studij računarstva

Završni rad

GPT virtualni asistent baziran na ESP32 platformi

Mentor: prof. dr. sc. Mladen Tomić

Rijeka, rujan 2023.

Robin Šćulac

0069085473

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 14. ožujka 2023.

Zavod: **Zavod za računarstvo**
Predmet: **Ugradbeni računalni sustavi**
Grana: **2.09.01 arhitektura računalnih sustava**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Robin Šćulac (0069085473)**
Studij: Sveučilišni prijediplomski studij računarstva

Zadatak: **GPT virtualni assistent implementiran na ESP32 platformi / GPT Virtual Assistant Based on ESP32 Platform**

Opis zadatka:

Realizirati "kućni asistent" ugradbeni sustav koji će odgovarati na upite koristeći usluge ChatGPT sustava. Kao ugradbenu platformu, koristiti ESP32. Procesiranje govora treba se odvijati na zasebnom poslužitelju baziranom na Django sustavu. Poslužitelj će također upravljati i korisničkim računima i podacima te pružati odgovore na korisničke upite putem ChatGPT-a. Ugradbeni sustav treba biti opremljen mikrofonom i zvučnikom za ostvarivanje audio sučelja prema korisniku. Pri programiranju ESP32 istražiti mogućnosti korištenja Rust "bare metal" prevodioca ili Zephyr-a te odabrati jednu od te dvije mogućnosti.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 20. ožujka 2023.

Mentor:

Prof. dr. sc. Mladen Tomić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2023.

_____
Robin Šćulac

Sadržaj

Popis programskog koda

Popis slika

Popis tablica

Uvod

Posljednjih godina brzi napredak tehnologije doveo je do integracije pametnih asistenata u različite aspekte naših života, od pametnih telefona i pametnih domova do vozila i industrijskih sustava. Ovi virtualni pomoćnici, opremljeni obradom prirodnog jezika i AI mogućnostima, napravili su revoluciju u interakciji između čovjeka i računala, pružajući pogodnost, učinkovitost i poboljšana korisnička iskustva. Kako potražnja za inteligentnim uređajima koji su svjesni konteksta nastavlja rasti, postoji rastuća potreba za učinkovitim i pouzdanim rješenjima za stvaranje ugradbenih virtualnih pomoćnika koji se neprimjetno integriraju u različita okruženja.

Ovaj rad predstavlja opsežno istraživanje razvoja ugradbenih virtualnog pomoćnika koji koristi programski jezik Rust. Rust je stekao priznanje zbog svoje jedinstvene kombinacije sigurnosti memorije, apstrakcija bez troškova i kontrole konkurentnosti – atributa koji su posebno korisni u kontekstu ugradbenih sustava. Iskorištavanjem mogućnosti Rusta, nastojimo se pozabaviti izazovima koji su svojstveni stvaranju učinkovitog, sigurnog i osjetljivog virtualnog pomoćnika prilagođenog za ugradbene platforme.

1 Problematika ugradbenih sustava

Ugradbeni sustavi su specijalizirani računalni sustavi dizajnirani za obavljanje određenih zadataka ili funkcija unutar većih proizvoda ili sustava. Iako nude brojne prednosti u pogledu učinkovitosti i namjenske funkcionalnosti, oni također predstavljaju nekoliko jedinstvenih izazova. Potrebno im je raditi u mjestu ograničenih resursa što izaziva potrebu za rad sa programskim jezicima niske razine. Programski jezici koji drže „monopol“ na tom tržištu su C i C++ . Programski jezici niske razine iako brzi posjeduju neke probleme koje je potrebno spomenuti [1].

1.1 Memorijska sigurnost

Pogreške pristupima memoriji kao što su:

1. Dereferenciranje (dohvaćanje vrijednosti iz memorijske adrese na koju pokazuje pokazivač) null pokazivača - kada program dereferencira null pointer dolazi do greške izvođenja.
2. Pretek stoga - dinamičnost podataka može dovesti do situacije gdje podatci prekoračuju alociranu memoriju
3. Utrke podataka - u situaciji gdje funkcije pristupaju istom podatku dolazi do grešaka u izvođenju
4. „Curenje“ memorije - repetitivno pozivanje funkcije inicijalizira nove blokove memorije koji zauzimaju prostor koji raste s vremenom izvođenja.

1.2 Nedefinirana stanja

Nedefinirano ponašanje odnosi se na ponašanje računalnog programa koje je nepredvidivo prema specifikacijama programskog jezika ili izvršnog okruženja. Kada program sadrži nedefinirano ponašanje, rezultati izvršavanja tog programa mogu se uvelike razlikovati i možda se neće pridržavati dosljednog ili logičkog uzorka. To može dovesti do rušenja, netočnog izlaza, sigurnosnih ranjivosti ili drugih neočekivanih i neželjenih ishoda.

1.3 Dugoročna održivost

Nedostatak apstrakcija: C je jezik niske razine koji pruža ograničene apstrakcije u usporedbi s modernijim jezicima. To može otežati razumijevanje i održavanje koda jer programeri moraju raditi izravno s upravljanjem memorijom i detaljima niske razine.

Ručno upravljanje memorijom: C zahtjeva ručnu dodjelu memorije i raspodjelu poslova pomoću funkcija kao što su malloc i free. To može dovesti do „curenja“ memorije, visećih pokazivača i drugih problema povezanih s memorijom ako se njima ne upravlja pažljivo.

C nema ugrađeni upravitelj paketa, što upravljanje ovisnošću i integraciju knjižnica trećih strana čini programer i potencijalno je sklono pogreškama.

Uspješno rješavanje ovih izazova zahtijeva kombinaciju znanja specifičnog za domenu, pažljivog dizajna sustava, učinkovite prakse kodiranja i upotrebe odgovarajućih razvojnih alata i metodologija.

1.4 Rust kao rješenje mnogih problema

Rust je dizajniran da jamči sigurnost i brzo izvođenje. Ugrađeni softver može imati problema, uglavnom zbog memorije. Rust je jezik u kojem prevodilac uvelike utječe na razvoj programskog koda, tako da jamči sigurnost nad memorijom već tijekom prevođenja [2].

1.4.1 Apstrakcija

Koncept apstrakcije podrazumijeva pretpostavku tipa od programskog jezika npr. u C-u se int, double i ostalo treba unaprijed odrediti te se ovdje u procesu prevađanja dešava. Automatska pretpostavka pokazivača bi programere poštedjela mnogo problema sa čitljivosti koda koji se javljaju u C-u.

Apstrakcije bez troška u Rust-u su one koje ne snose troškove vremena izvođenja u brzini izvođenja ili korištenju memorije tj. nije potrebno navoditi tip strukture koji je podatak jer će to prevoditelj prepoznati.

Kao u primjeru ispod rtc postane tip `RTC<'_>` adc1_config postane `AdcConfig<ADC2>`

Pristupanje specifičnim imenskim prostorima se radi isto kao u C++-u `ident::ident`

```
let mut rtc = Rtc::new(peripherals.RTC_CNTL);
let mut adc1_config = AdcConfig::new();
```

Programski kod 1-1 Primjer poziva konstruktora u Rust-u

Nasuprot tome, virtualne metode su dobar primjer skupe apstrakcije: u mnogim OO jezicima tip pozivatelja metode određen je tijekom izvođenja što zahtijeva održavanje tablice pretraživanja (upotreba memorije za vrijeme izvođenja) i zatim stvarno izvođenje pretraživanja (preopterećenje vremena izvođenja po metodi poziv, vjerojatno barem dodatno dereferenciranje pokazivača) s tipom vremena izvođenja kako bi se odredilo koju verziju metode pozvati.

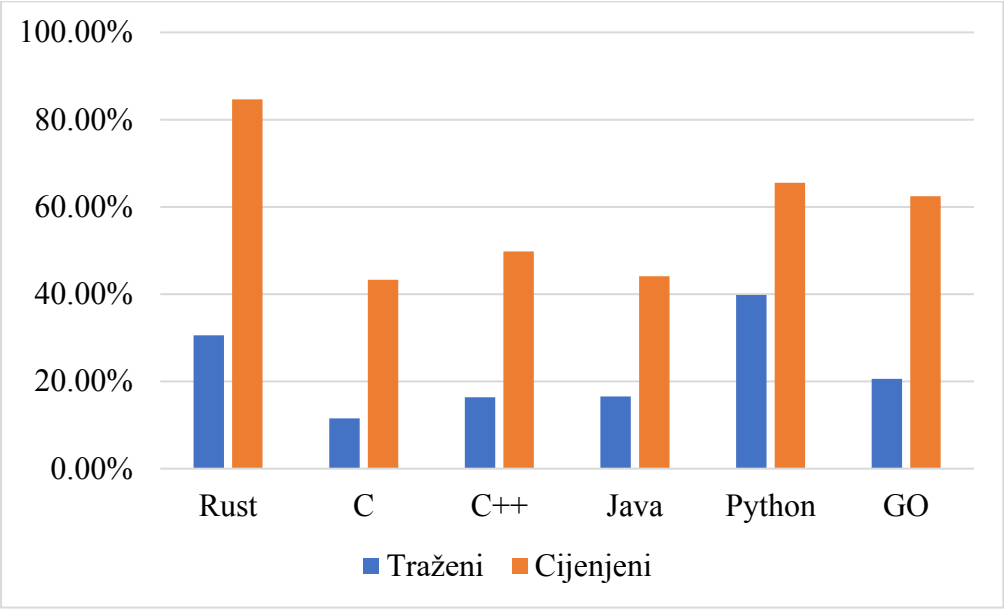
Još jedan dobar primjer bi bilo sakupljanje smeća: u zamjenu za mogućnost da se ne mora brinuti o detaljima dodjele memorije rezultira u duljim vremenima potrebnim za prevoditi sam program.

Ipak, Rust uglavnom pokušava imati apstrakcije bez troškova: one koje prevodilac može sigurno i ispravno pretvoriti u oblike koji ne podnose dodatnu indirektnost/korištenje memorije.

S apstrakcijom dobivamo kod koji je manje sklon greškama nedefiniranih stanja jer prevodilac će u svakom trenutku prepoznati potrebno stanje za određenu varijablu. Rezultat je kod koji je kvalitetan i održiv.

1.4.2 Popularnost Rust-a

U 2023. godini StackOverflow je proveo najveći upitnik među programerima te istraživanja o tome koji su programski jezici cijenjeni i traženi. Sa preko 50 programskih jezika Rust je bio smatran najcjenjenijim i petim najtraženijim programskim jezikom. [3]



Slika 1.1 Prikaz podataka 2023 StackOverflow upitnika

Dublje u istraživanju može se saznati da Rust čini 12.21% radne sile naravno i dalje u sjeni C-a (16.66%) i C++-a (20.21%).

1.4.3 Koncept posuđivanja memorije

Posuđivanje memorije je koncept u programskim jezicima koji omogućuje programu da privremeno posudi referencu na dio memorije, obično u vlasništvu drugog dijela programa, bez preuzimanja vlasništva nad tom memorijom. Ovaj mehanizam središnji je za Rust-ov sustav vlasništva i osmišljen je kako bi osigurao sigurnost memorije i spriječio uobičajene programske pogreške, poput „curenja“ memorije i utrke podataka.

U Rustu se posuđivanje memorije postiže kombinacijom referenci, trajanja i pravila vlasništva. Evo zašto je posuđivanje memorije prednost u usporedbi s tradicionalnim konceptima poput statičke alokacije memorije koji uzrokuje „curenje“ memorije ili garbage collector koji usporava sustav i čini sustav nesigurnim.

Životni ciklus varijable mora se specificirati u deklaraciji same varijable. Ako je on nepoznat, nije potrebno navesti te će prevoditelj automatski pretpostaviti životni vijek varijable. Trajno posuđivanje varijable funkcijama se odrađuje ključnom riječi move.

```
let mut add_two_to_x = move  ||   adcfunc( &mut adc, &mut adc1);
```

Programski kod 1-2 Primjer posuđivanja u Rust-u

Ključnom riječi `move` (Programski kod 1-2) izvodimo micanje varijabli *adc* i *adc1* u funkciju `adcfunc`. Unutar vertikalnih linija je moguće deklarirati varijable kojima je životni ciklus jednak izvođenju funkcije te će one biti terminirane sa prvim izvođenjem i ponovno inicijalizirane pri ponovnom izvađanju.

Na taj se način kreira promjenjiva funkcija koja se može prilagoditi prema potrebi izvođenja programa te se varijable `adc` i `adc1` više ne mogu koristiti u glavnoj funkciji zbog premještaja u funkciju.

Posuđivanje memorije za razliku od ručnog upravljanja memorije strogo prepoznaje moguća curenja memorije i prepozna ih kao greške za razliku od C-a i C++-a kojima to rezultira u neispravnostima.

1.4.4 Rust-ov prevodioc

Poznavanje Rust-a podrazumijeva poznavanje LLVM-a jer prevodilac se prilagođava od biblioteke do biblioteke. Rust biblioteke imaju sposobnost prepoznati vlastite greške specifične za tu biblioteku(npr. SQL biblioteka prepoznaje greške vezane za SQL upit) (Slika 1.2).Prevodilac prepoznaje mnoge greške koje se javljaju kao neispravnosti u drugim programskim jezicima, kao što su null pointer reference ili krivo slanje reference, prepoznavanje i pravilno korištenje referenciranja i posuđivanja te to radi na jako jednostavan i user-friendly način.

```
    |
244 |         timer1.start(a.into_duration());
    |         ^^^^^^ `timer1` is a `&` reference, so the data it refers to cannot be borrowed as mutable
    |
help: consider specifying this binding's type
    |
241 |         let timer1: &mut esp32c3_hal::Timer<esp32c3_hal::timer::Timer0<TIMG1>> = timer1.as_ref().unwrap();
    |                     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Slika 1.2 Prikaz izlaza Rustovog prevodioca

Sve zajedno, Rustov prevodilac trenira mozak tako da izbaci loše navike iz drugih jezika te je učenje Rusta veoma teško, ali rezultira u savršenom kodu.

1.4.5 Google-ova implementacija Rust-a

Google je nedavno najavio KataOS, novi sigurni operativni sustav za ugradbene sustave koji pokreću aplikacije strojnog učenja. KataOS je gotovo u cijelosti napisan u Rust-u, programskom jeziku koji su usvojili Android Open Source Project i Linux kernel projekt. Rust pruža snažnu polaznu točku za sigurnost softvera, budući da eliminira čitave klase grešaka, kao što su off-by-one pogreške i preljevi međuspremnika. [4]

Google je odabrao seL4 kao mikrokernel za KataOS jer je matematički dokazano siguran, sa zajamčenom povjerljivošću, integritetom i dostupnošću. Putem okvira seL4 CAmkES, Google također može pružiti statički definirane komponente sustava koje se mogu analizirati. KataOS ima za cilj pružiti provjerljivo sigurnu platformu koja štiti privatnost korisnika tako što aplikacijama čini logično nemogućim probijanje hardverske sigurnosne zaštite jezgre. Google je otvorio nekoliko komponenti za KataOS na GitHubu i sklopio partnerstvo s Antimicro-m na njihovom Renode simulatoru i srodnim okvirima. KataOS se ne temelji na Linuxu i nema nikakve veze s Google-ovim Fuchsia OS-om. Referentna implementacija KataOS-a zove se Sparrow, koja kombinira KataOS sa sigurnom hardverskom platformom. [5]

1.4.6 Bare metal i RTOS

U „bare metal“ okruženju nikakav kod nije učitan prije vašeg programa. Bez softvera koji nudi OS ne možemo učitati standardnu biblioteku. Umjesto toga, program, zajedno sa bibliotekama koje koristi, može koristiti samo hardver (bare metal) za rad. Kako biste spriječili rust od standardne biblioteke koristite no_std. Platformski neovisni dijelovi standardne biblioteke dostupni su putem libcorea. libcore također isključuje stvari koje nisu uvijek poželjne u ugradbenim okruženjima. Jedna od tih stvari je alokator memorije za dinamičku dodjelu memorije. Pri upotrebi te ili neke druge funkcionalnosti, često postoje biblioteke koji ih pružaju.

RTOS ili real-time je u osnovi program koji djeluje kao sučelje između hardvera sustava i korisnika. Štoviše, upravlja svim interakcijama između softvera i hardvera.

Operativni sustavi u stvarnom vremenu koriste se u sustavima u stvarnom vremenu gdje su vremenska ograničenja fiksna i strogo se poštuju. To znači da je vrijeme za obradu i odgovor vrlo malo. Štoviše, sustav bi trebao izvršiti zadani zadatak u fiksnom vremenu, inače bi to rezultiralo kvarom sustava.

Vrijeme odziva je vrijeme unutar kojeg sustav prima unos, obrađuje podatke i daje rezultate. Štoviše, koriste se u sustavima poput robota, lansirnih projektila, zrakoplova itd.

2 Komponente koje su se koristile

2.1 ESP32-C3

Espressif je jedna od rijetkih popularnih kompanija kojima ploča dolazi s modernim IOT stogom Wi-Fi i Bluetooth5(LE) te korištenjem tih funkcionalnosti dolazi do razvojne ploče uz veliki ekosustav ambicioznih programera. [6]

Služi se niskom potrošnjom energije što je ključno za kontinuirani rad uređaja posebno za pristup koji je potreban primijeniti u slučaju da se radi o AI asistentu bez trajnog napajanja.

Koristi opće poznatu i industrijski standardnu RISC-v jezgru koja uz otvoreni kod stvara popularno i prihvaćeno rješenje za jezgru sustava.

Specifikacije su: frekvencija procesora od 160MHz, kapacitet nasumične memorije 400Kb i kapacitet flash memorije od 4Mb što je više nego dovoljno sposoban je za izgradnju sustava u projektu.

Ekonomičnost je bio veliki faktor u odabiru potrebne razvojne ploče za slučaj da sam htio probati drugi čip STM32 zahtjeva i Wi-Fi modul ESP8266 tako da je na ovaj način sve centralizirano u jednom sustavu i može se reći za cijenu od 5€ relativno ekonomično.

2.2 Senzor zvuka KY-037

Vrsta elektrostatičkog mikrofona temeljenog na kondenzatoru, koji eliminira potrebu za polarizirajućim napajanjem pomoću trajno napunjenog materijala. Radni napon mu je 5V, ima fizički potenciometar.

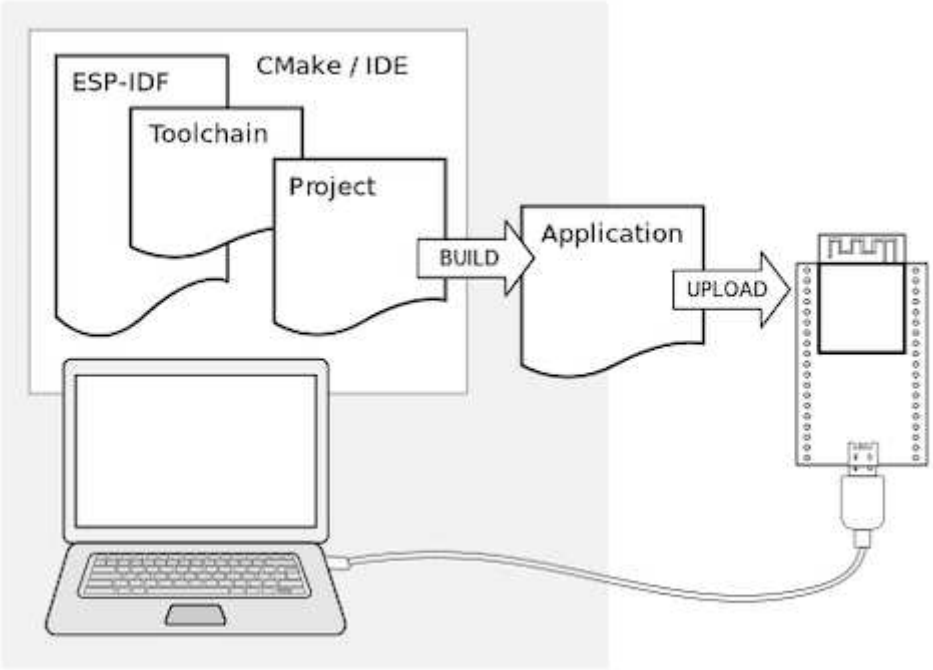
Iako je senzor zvuka u osnovi dizajniran za prepoznavanje glasnoće, moguće je razaznati zvuk u slučajevima kada se uzorci prikupljaju iznimno visokom brzinom, dosežući brzinu uzorkovanja od otprilike 20 kilo Hertza (20 KHz) [7].

3 Postavljanje radnog prostora

Postavljanje radnog prostora za Rust razvoj na ESP32-C3 uključuje konfiguriranje vašeg razvojnog okruženja za prevođenje, flashanje (premještanje programa u memoriju mikrokontrolera) i pokretanje Rust programa na ESP32-C3 mikrokontroleru (Slika 3.1).

Da bismo počeli koristiti ESP-IDF na ESP32, potrebno je instalirati sljedeći softver

- Toolchain za prevođenje koda za ESP32
- Alati za izradu - CMake i Ninja za izradu pune aplikacije za ESP32
- ESP-IDF koji u biti sadrži API (softverske biblioteke i izvorni kod) za ESP32 i skripte za rad Toolchaina



Slika 3.1 Razvoj aplikacija u Rust-u preuzeto iz [8]

3.1 Rust instalacija

Proces počinje preuzimanjem programskog jezika Rust sa službene web stranice (https://www.rust-lang.org/tools/install) i pokretanjem Windows instalacijskog programa. Instalacijski program nudi opciju uključivanja Rusta u PATH sustava, osiguravajući dostupnost naredbenog retka. Nakon toga, Visual C++ Build Tools, dostupni na (https://visualstudio.microsoft.com/visual-cpp-build-tools/), mogu se instalirati za rješavanje potencijalnih izazova kompilacije povezanih s C vezama.

Kako bi se omogućila unakrsna kompilacija, RISC-V za ESP32-C3 dodaje se u lanac alata Rust pomoću naredbe "rustup target add riscv32imc-unknown-none-elf." Pomoćni program "cargo-binutils" zatim se dobiva preko "cargo install cargo-binutils," što omogućuje konverziju prevedenih ELF binarnih datoteka u različite formate.

Za one koji slijede razvoj ESP32-C3 s okvirom ESP-IDF, upute navedene u Vodiču za početak rada s ESP-IDF (https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/get-started/index.html) preporučuju se.

3.2 Espup instalacija

Ažuriranje firmvera (softver koji omogućuje kontrolu niske razine za određeni hardver uređaja.) na mikrokontrolerima, kao što je ESP32-C3, ključni je zadatak u održavanju funkcionalnosti i sigurnosti uređaja. Alat espup nudi pojednostavljen pristup ažuriranju firmvera, omogućujući programerima da učinkovito upravljaju i implementiraju ažuriranja bežičnim putem.

```
cargo install espup
```

Nakon uspješne instalacije, alat espup postaje dostupan iz naredbenog retka. Programeri mogu koristiti njegove značajke za stvaranje, upravljanje i implementaciju ažuriranja firmvera za ESP32-C3 mikrokontrolere. Detaljna dokumentacija i primjeri korištenja dostupni su u službenom espup repozitoriju.

3.3 Cmake i Ninja instalacija

3.3.1 Cmake

Cmake je open-source platformski neovisan sustav za izgradnju koji pomaže u upravljanju procesom izgradnje softverskih projekata. Generira datoteke za izgradnju specifične za platformu (kao što su Makefile ili projektne datoteke za IDE) iz konfiguracijske skripte visoke razine. CMake pruža unificirani način za opisivanje procesa izrade za različite platforme i alate za izradu, omogućujući programerima da se usredotoče na pisanje koda umjesto da se bave zamršenostima izgradnje. Razlog za korištenje CMake je:

- Integracija s IDE-ovima: CMake-generirane datoteke za izgradnju mogu se neprimjetno integrirati s popularnim integriranim razvojnim okruženjima (IDE-ovima) kao što su Visual Studio, Xcode i CLion. To razvojnim programerima omogućuje rad u njihovim željenim razvojnim okruženjima, dok i dalje imaju koristi od standardiziranog procesa izrade.
- Third party knjižnice: CMake pojednostavljuje integraciju knjižnica third party knjižnica u projekte pružanjem standardiziranih metoda za lociranje i uključivanje ovisnosti. Ovo smanjuje napor potreban za postavljanje i upravljanje vanjskim knjižnicama.

1. Potrebno je posjetiti CMake stranicu za preuzimanje: https://cmake.org/download/
2. pronaći odjeljak "Windows" i preuzeti Windows instalacijski program (.msi datoteka) za svoj sustav (32-bitni ili 64-bitni).
3. Pokrenuti preuzeti instalacijski program.

4. Slijediti upute čarobnjaka za instalaciju. Obično se mogu koristiti zadane postavke, ali svakako označiti okvir koji dodaje CMake vašem sustavu PATH. To omogućuje korištenje CMake iz naredbenog retka.

3.3.2 Ninja

Ninja je brz i lagan sustav za izradu dizajniran za poboljšanu brzinu i skalabilnost u usporedbi s tradicionalnim alatima za izradu kao što je Make. Fokusiran je na brzo generiranje i izvršavanje zadataka izgradnje, što je posebno povoljno za velike projekte.

- Brzina: Ninja je dizajniran da bude brz. Njegov minimalistički dizajn i učinkovito praćenje ovisnosti rezultiraju bržim vremenom izgradnje, što je osobito korisno za velike baze koda.
- Paralelizam: Ninja može paralelno izvršavati više zadataka izrade, koristeći prednost modernih višejezgrenih procesora. Ova konkurentnost dodatno pomaže smanjiti vrijeme izgradnje.
- Smanjeni troškovi: Za razliku od složenijih sustava za izradu, Ninja izbjegava nepotrebne troškove, kao što je parsiranje velikih Makefile-ova. To dovodi do učinkovitijeg korištenja resursa tijekom procesa izgradnje.
- Skalabilnost: Kako projekti rastu, održavanje prihvatljivog vremena izgradnje postaje izazovno. Ninjina arhitektura izgrađena je za učinkovito rukovanje velikim bazama koda, osiguravajući da performanse izgradnje ostanu dosljedne čak i kada se projekt širi.
- Integracija s CMake: CMake može generirati Ninja build datoteke, omogućujući programerima da kombiniraju prednosti CMake međuplatformskih mogućnosti s Ninjinom brzinom i skalabilnošću.

Ukratko, CMake pojednostavljuje i standardizira konfiguraciju i upravljanje procesima izgradnje, čineći višeplatformski razvoj pristupačnijim, dok Ninja pruža optimizirano okruženje za izvođenje izgradnje koje smanjuje vrijeme izgradnje i dobro se skalira s veličinom projekta. Kombinacija CMakea i Ninje može dovesti do pojednostavljenog i učinkovitog tijeka razvoja.

3.4 Espflash

Espflash je alat koji pojednostavljuje flashanje i testiranje ESP-Rust aplikacija na ESP32 mikrokontrolerima. Njegova primarna uloga je pojednostaviti proces prijenosa prevedenog firmvera iz razvojnog okruženja u flash memoriju ciljnog uređaja. Automatiziranjem ovih zadataka, espflash značajno smanjuje vrijeme i trud potrebne za ručno fleširanje i testiranje, poboljšavajući cjelokupni razvojni ciklus.

Alat espflash ima značajnu važnost unutar razvojnog ciklusa ESP-Rust, djelujući kao ključni most između razvoja i implementacije. Pojednostavljivanjem i automatiziranjem procesa fleširanja prevednog firmvera na ESP32 mikrokontrolere, espflash poboljšava učinkovitost, usmjerava otklanjanje pogrešaka i testiranje te potiče pristupačnost za širi raspon programera. Njegova integracija s Cargom, upraviteljem paketa Rust, dodatno očvršćuje funkcionalnosti specifične za Rust u procesu flashanja. Utjecaj ovog alata nadilazi sferu razvoja, olakšavajući stvaranje pouzdanih IoT aplikacija i jačajući ekosustave ESP32 i Rust. U biti, espflash je ključni pomagač učinkovitih procesa kodiranja, testiranja i implementacije, pridonoseći napretku i ESP32 aplikacija i Rust programskog jezika u domeni ugradbenih sustava.

Instalirava se komandom:

```
cargo install espflash
```

3.5 Cargo Generate

Naredba za generiranje Cargo s predloškom esp-rs/esp-idf-template prikladan je način za postavljanje novog ESP-Rust projekta pomoću okvira ESP-IDF. Ovaj predložak automatizira početno postavljanje i konfiguraciju projekta za mikrokontrolere ESP32 i ESP32-C3. Evo kako proces funkcionira:

3.5.1 Instalacija Cargo Generate:

Ako je cargo generation nedostupan, možete ga instalirati koristeći sam Cargo. Otvorite terminal i pokrenite:

```
cargo install cargo-generate
```

Generirajte novi ESP-Rust projekt:

Nakon što se Cargo Generator instalira, možete ga koristiti za generiranje novog ESP-Rust projekta pomoću predloška esp-rs/esp-idf-template. Pokrenite sljedeću naredbu:

```
cargo generate --git https://github.com/esp-rs/esp-idf-template.git --
name moj-esp32-projekt
```

U ovoj naredbi, --git navodi URL spremišta predložaka, a --name navodi naziv vašeg novog projekta (zamijenite "my-esp32-project" željenim nazivom projekta).

Nakon generiranja projekta, potrebno je promijeniti poziciju terminala na novonastalu mapu

```
cd mojEsp32Projekt
```

Predložak uključuje datoteku Cargo.toml kojom je moguće modificirati za konfiguraciju postavki projekta, kao što je ciljna arhitektura (ESP32 ili ESP32-C3), ovisnosti i druge konfiguracije specifične za projekt.

Korištenjem predloška esp-rs/esp-idf-template s generiranjem tereta, moguće je brzo postaviti novi ESP-Rust projekt temeljen na ESP-IDF okviru. Predložak pruža organiziranu strukturu, skripte za izgradnju i početnu konfiguraciju, štedeći vam vrijeme i trud u početnom postavljanju projekta. Kako budete napredovali, možete dodatno prilagoditi svoj projekt, dodati značajke i izgraditi robusne aplikacije za ESP32 i ESP32-C3 mikrokontrolere.

3.6 Konfiguracija projekta

U Rust-u je datoteka Cargo.toml konfiguracijska datoteka koju koristi Cargo sustav za izgradnju i upravitelj paketa. Koristi se za određivanje raznih detalja o vašem Rust projektu, njegovim ovisnostima, metapodacima i konfiguraciji izgradnje. Datoteka Cargo.toml koristi format TOML (Tom's Obvious, Minimal Language). Evo objašnjenja različitih dijelova koje možete pronaći u tipičnoj datoteci Cargo.toml:

3.6.1 Odjeljak s informacijama o paketu

Odjeljak informacija o paketu(Programski kod 3-1) sadrži metapodatke o vašem paketu (sanduci), kao što su naziv, verzija, autori i opis. Također uključuje druga izborna polja kao što su licenca, repozitorij i izdanje.

```
[package]
name = "mojProjekt"
version = "0.1.0"
authors = ["Ime Prezime<ime.prezime@email.com>"]
description = "Opis projekta"
license = "MIT"
repository = "https://github.com/korisnickoIme/imeProjekta "
edition = "2018"
```

Programski kod 3-1 Primjer odjeljka s informacijama o paketu

3.6.2 Odjeljak ovisnosti

Ovaj se odjeljak koristi za popis ovisnosti vašeg projekta (). Ovisnosti mogu biti iz sanduka na crates.io ili iz lokalnih staza ili Git repozitorija. Ovisnosti su podijeljene u dvije kategorije: normalne ovisnosti i razvojne ovisnosti.

```
[dependencies]
```

```
esp-wifi = { path = "../esp-wifi", features = ["esp32c3"] }
embedded-hal-async = { workspace = true, optional = true }
esp-hal-common     = { version = "0.10.0",  features = ["esp32c3"]}
[dev-dependencies]
tokio = "1.0"
```

Programski kod 3-2 Primjer odjeljka ovisnosti

3.6.3 Oznake značajki

Oznake značajki ()omogućuju vam da omogućite ili onemogućite određene značajke u vašem crate-u. Ovo je korisno kada želite pružiti izbornu funkcionalnost koju korisnici mogu uključiti kada koriste neki crate.

```
[features]
default = ["std", "hal", "esp-idf-sys/native"]
pio = ["esp-idf-sys/pio"]
all = ["std", "nightly", "experimental", "embassy"]
hal = ["esp-idf-hal", "embedded-svc", "esp-idf-svc"]
```

Programski kod 3-3 Primjer oznajka značajki

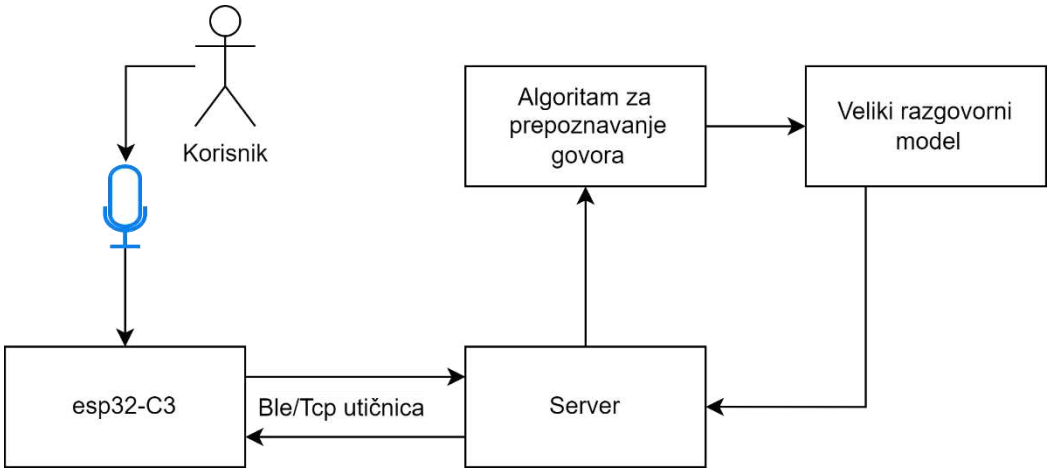
4 Implementacija sustava

Implementacija sustava ovog rada sastoji se od sustav za komunikaciju između korisnika i računalnog entiteta (Slika 4.1). Prva komponenta je mikrofon koji služi kao ulazni uređaj za korisnika, omogućujući prikupljanje audio signala iz okoline. Slijede dvije moguće opcije za komunikaciju između korisnika i sustava, ovisno o konkretnoj situaciji: Bluetooth konekcija ili TCP konekcija prema centralnom serveru.

Nakon što sustav primi audio signale putem odabrane konekcije, server preuzima ulogu centralne komponente za procesiranje tih signala. Server provodi obradu ulaznih audio podataka te ih prosljeđuje algoritmu za prepoznavanje govora. Algoritam za prepoznavanje govora koristi se kako bi se iz audio signala izdvojili korisnički unos i pretvorili u tekstualni oblik. Nakon obrade signala i pretvorbe u tekst, dobiveni podaci prosljeđuju se velikom razgovornom modelu.

Veliki razgovorni model, kao ključna komponenta sustava, ima zadatak interpretirati i razumjeti tekstualni unos korisnika te generirati adekvatan odgovor ili akciju. Ovaj model koristi duboke neuronske mreže i bogato obučene jezične modele kako bi interpretirao korisnički unos, razumio kontekst i generirao odgovarajući odgovor ili izvršio željenu akciju.

Ovaj opisani sustav predstavlja kompleksnu inačicu sučelja između korisnika i računalnog entiteta, koji omogućuje korisnicima interakciju s računalom putem govora i tekstualne komunikacije. Sustav se temelji na tehnologijama kao što su mikrofon, Bluetooth i TCP komunikacija, algoritmi za prepoznavanje govora te duboki razgovorni modeli, čime se omogućava efikasna, prirodna i intuitivna interakcija između korisnika i računalnog entiteta.



Slika 4.1 Prikaz scenarija sustava

4.1 Wi-Fi Komunikacija RTOS

Kao što je prije spomenuto ESP32-C3 ima integrirani Wi-Fi čip sposoban za 2.4ghz Wi-Fi komunikaciju, originali plan je bio koristiti TCP utičnica i da se rade web zahtjevi preko utičnice te prijam rješenja i prikaz podataka preko konzole. u Wi-Fi pristupu implementacija je napravljena s RTOS pristupom, tako da određene instrukcije su unaprijed napisane na čip te samo je potrebno upravljati tim instrukcijama. [9]

Za uspostavljanje komunikacije sa serverom potrebno je uspostaviti komunikaciju s usmjerivačem.

```
fn wifiinit (peripheral:Modem) -> EspWifi<'static>{
    let sys_loop =EspSystemEventLoop::take().unwrap();
    let nvs = EspDefaultNvsPartition::take().unwrap();
    let mut wifi_driver: EspWifi<'_>= EspWifi::new(
        peripheral,
        sys_loop,
        Some(nvs),

    ).unwrap();
```

Programski kod 4-1 Funkcija wifiinit

Prikazani kod (Programski kod 4-1) definira funkciju koja se zove wifiinit koja uzima *peripheral* parametar tipa Modem i vraća EspWifi instancu sa statičkim vijekom trajanja.

Statički životni vijek obično znači da vraćena instanca EspWifi može živjeti cijelo vrijeme trajanja programa.

Redak `let sys_loop = EspSystemEventLoop::take().unwrap();`stvara varijablu sys_loop pozivanjem metode take() na tipu EspSystemEventLoop. Petlja događaja je posebna vrsta petlje koja se koristi za sistemske događaje (na primjer, Wi-Fi događaje). Ručka za ovu petlju skrivena je od korisnika, a stvaranje, brisanje, registracija/deregistracija rukovatelja i objavljivanje događaja obavljaju se putem varijante API-ja za petlje korisničkih događaja. Metoda unwrap() koristi se za obradu svih potencijalnih pogrešaka odmotavanjem rezultata. Ako operacija take() ne uspije (vrati Ništa), unwrap() će paničariti i srušiti program.

`let nvs = EspDefaultNvsPartition::take().unwrap();`, ovo stvara varijablu nvs (non-volatile storage) uzimanjem instance `EspDefaultNvsPartition` i odmotavanjem rezultata.

`let mut wifi_driver: EspWifi<'_> = EspWifi::new(peripheral, sys_loop, Some(nvs)).unwrap();` Ovaj redak inicijalizira promjenjivu varijablu wifi_driver s instancom EspWifi. Konstruira EspWifi objekt pomoću novog konstruktora.

`peripheral` se prosljeđuje kao prvi argument EspWifi konstruktoru, jer je riječ o hardverskom modemu.

`sys_loop` prosljeđuje se kao drugi argument, što ukazuje da bi instanca EspWifi koristi sys_loop.

`Some(nvs)` prosljeđuje se kao treći argument, što ukazuje da je instanci EspWifi osigurana neka trajna pohrana.

Metoda unwrap() koristi se za rješavanje bilo kakvih pogrešaka koje se mogu pojaviti tijekom konstrukcije EspWifi instance. Ako se dogodi pogreška, uspaničit će se i srušiti program.

```
wifi_driver.set_configuration(&Configuration::Client(ClientConfigurati
on{
        ssid:"MOJ_WIFI_SSID".into(),
        password:"MOJ_WIFI_PASSWORD ".into(),
        ..Default::default()
        })).unwrap();
        wifi_driver.start().unwrap();
        wifi_driver.connect().unwrap();
        while !wifi_driver.is_connected().unwrap() {
            let _config = wifi_driver.get_configuration().unwrap();
```

```
        }
        return wifi_driver;
    });
```

Programski kod 4-2 Inicijalizacija Wi-Fi drivera

Wi-Fi driveru se konfiguracija postavlja korištenjem konstruktora za novu Wi-Fi konfiguraciju. Konstruktor za Wi-Fi konfiguraciju prima parametre:

- SSID – SSID lokalnog usmjerivača
- Password - password lokalnog usmjerivača
- Bssid – MAC adresu Wi-Fi-a (opcionalan parametar)
- Chanel – frekvencijski raspon Wi-Fi-a (opcionalan parametar)
- Auth_method – metoda autorizacije lokalnog usmjerivača (opcionalan parametar)

Opcionalni parametri se uvoze sa `..Default.default()` dok SSID i password potrebno je unesti preko konstruktora.

Nakon inicijalizacije Wi-Fi-a potrebno je spojiti se na lokalni server preko TCP utičnice koji je otvoren i sluša na portu 8080

```
fn  socketlisten()-> TcpStream
{
    loop {
        if let Ok( mut  stream) =
TcpStream::connect("192.168.1.8:8080") {
            println!("Successfully connected to server in port 8080");
            stream.set_nonblocking(true).unwrap();
            stream.write("Hello World!".as_bytes()).unwrap();
            return stream.into();

        }
        else {
            println!("Couldn't connect to server...");
        }
}
```

Programski kod 4-3 Prikaz spajanja na TCP utičnicu

Funkcija *socketlisten* pokušava se spojiti na TCP poslužitelj s IP adresom "192.168.1.8" na portu 8080. Ovdje je pregled funkcionalnosti koda:

Funkcija počinje s petljom koja će nastaviti pokušavati spojiti se na poslužitelj dok se ne uspostavi uspješna veza.

Unutar petlje, kod koristi funkciju `TcpStream::connect("192.168.1.8:8080")` za pokušaj povezivanja s poslužiteljem. IP adresa i port navedeni su u adresnom nizu.

Rezultat pokušaja povezivanja provjerava se pomoću konstrukcije if let. Ako je pokušaj povezivanja uspješan (što rezultira varijantom Ok), stvara se promjenjiva varijabla toka tipa *TcpStream*.

Kod ispisuje poruku o uspjehu koja pokazuje da je veza s poslužiteljem na portu 8080 bila uspješna.

Linija *stream.set_nonblocking(true).unwrap()* postavlja *TcpStream* na način rada bez blokiranja, što znači da čitanje i pisanje u stream neće blokirati dretvu.

Linija *stream.write("Hello World!".as_bytes()).unwrap()* ispisuje niz "Hello World!" kao bajtovi u TCP tok.

Nakon uspješnog pisanja u tok i izvođenja potrebnih operacija, funkcija vraća instancu TcpStream kao povratnu vrijednost. Metoda into() koristi se za pretvaranje TcpStream-a u traženi tip za povratnu vrijednost.

Ako pokušaj povezivanja ne uspije (što rezultira varijantom Err), kod ispisuje poruku o pogrešci koja pokazuje da se veza nije mogla uspostaviti.

Petlja se nastavlja, pokušavajući se ponovno povezati u sljedećoj iteraciji.

Za podatke koji se trebaju slati u TCP tok potrebno ih je očitati tj. potrebno je pretvoriti analogne signale od mikrofona u digitalne signale za TCP tok.

```
1.     let mut  adc = adc::AdcDriver::new(periph.adc2,
&adc::config::Config::new().calibration(true)).unwrap();
 2.     let mut adc1 =
adc::AdcChannelDriver::<_,adc::Atten11dB<adc::ADC2>>::new(periph.pins.
gpio5).unwrap();
 3.     let mut  config = config::Config::new();
 4.     config= config.auto_reload(true).divider(80).xtal(true);
 5.     let mut timer: TimerDriver<'_> =
TimerDriver::new(periph.timer00, &config).unwrap();
 7.     timer.enable_interrupt();
 8.     timer.enable_alarm(true).unwrap();
 9.     timer.enable(true).unwrap();
10.     timer.set_alarm(1000).unwrap();
12.     {
13.         let mut add_two_to_x = move  ||   adcfunc( &mut adc, &mut
adc1);
14.         unsafe { timer.subscribe(add_two_to_x).unwrap() };
15.
16.     }
```

Programski kod 4-4 Prikaz ADC-a i prekida

Kreacija Adc drivera zajedno sa timerom koji poziva frekvencijom od 1000Hz funkciju *adcfunc(&mut adc, &mut adc1)* u koju su pomaknute vrijednosti adca te kao što je prije spomenuto u konceptu posuđivanja više se ne mogu koristiti u glavnoj funkciji.

Adc funkcija je jednostavna funkcija koja sve piše u buffer što je pročitala sa ADC-a. pošto se radi o mikrofonu uzima uzorke svaku milisekundu te sprema očitane podatke sa mikrofona.

```
1. match adc.read( adc1) {
2.              Ok(value) => {
3.                  //compress the data to 8 bits
4.                  BUFFERED_DATA.push(value);
5.
6.              } ,
7.              Err(_err) => BUFFERED_DATA.push(0) ,
8.          }
9.
```

Programski kod 4-5 Prikaz match funkcije u Rustu

Bitno je napomenuti match u Rust-u čija je funkcija pokrivanje više slučaja u kojoj varijabla može biti te funkcionira kao kompleksnija verzija Switch case-a. Karakteristike koje dijeli sa switch caseom je da je moguće postupati s obzirom na variablu, ali match također podržava slučajeve errora. Globalne varijable sa statičkim životnim vijekom kao što je varijabla *BUFFERED_DATA* ne bi trebale biti korištene u Rust-u (niti u jednom drugom jeziku) u poglavlju 4.2. Pokazuje se pravilan način kreacije globalnih statičkih varijabli.

Nakon što su podatci pročitani i buffer napunjen potrebno je poslati navedeni buffer u prethodno deklariran *TcpStream*.

```
 1. unsafe{
 2.     if(BUFFERED_DATA.len()>=1000)
 3.     {
 4.             let mut data = BUFFERED_DATA.clone();
 5.             BUFFERED_DATA.clear();
 6.             println!("data {:?}",data.first());
 7.             let  res:Vec<u8> = u16_to_u8_bytes(&data);
 8.             {
 9.             stream.write(&res).unwrap();
10.             }
11.     }
12.}
```

Programski kod 4-6 Pisanje u TCP utičnicu

Kako bi se slali podatci kada se buffer napuni i kako bi koristili buffer minimalnu količinu vremena (jer buffer je u stalnom Datarace sa adcread funkcijom), kloniramo buffered data i očistimo ga te je spreman za interrupt korištenje. Zatim, ispod dobivenog buffer-a u byteovima

koji su poslani TCP Streamu. Iako možda poprilično izgleda jednostavno, mnogo stvari se treba napraviti jer je jako malo podataka oko toga kako se specifično ove situacije rade u std-u. No, to je stvar s kojima se ugradbeni sustavi susreću u bez obzira o platformi i programskom jeziku.

4.1.1 Zašto Wi-Fi pristup ne funkcionira

Pri primanju podataka na računalu i pokušaju pretvorbe te podatke u neki razumljiv zvučni signal zvuk bi izlazi nepravilan. Evidentno je da je greška u tome što zvuk koji se uzimao uzorcima frekvencijom od 1000hz i pokušaj da se poveća do 44.1KHz koristeći Rust biblioteku zvanu Rubio. No, zato što zvuk zapravo zahtjeva uzimanje uzorka frekvencijom od barem 16KHz da bi zvuk bio jasan, potrebno je promijeniti pristup. [6]

Pristup je potrebno promijeniti iz razloga jer sa Wi-Fi-em uključenim Esp32 ADC maksimalna frekvencija je 1000Hz te je potrebno prebaciti se na Bluetooth pristup koji dopušta ADC da uzima uzroke brzinom većom od 20KHz. [10]

4.2 Bluetooth Pristup Bare metal

Nakon neuspjeha Wi-Fi TCP komunikacije potrebno je promijeniti pristup te je taj pristup pronađen u Bluetooth 5 sustavu.

Bluetooth Low Energy (BLE) tehnologija može se koristiti za prijenos putem mikrofona u određenim aplikacijama, iako to možda nije najčešći slučaj upotrebe. BLE je prvenstveno dizajniran za bežičnu komunikaciju između uređaja male snage i kratkog dometa. Često se koristi za aplikacije poput IoT uređaja, nosivih uređaja, uređaja za praćenje fitnessa, pametnih kućnih uređaja i sličnih scenarija niske brzine prijenosa podataka.

Kada je pristup promijenjen potrebno je razumijeti da std:: knjižnice. esp_idf_hal koja je u prethodnom poglavlju služila za većinu apstrakcije. Knjižnica je služila kao sloj programiranja koji omogućuje računalnom OS-u interakciju s hardverskim uređajem na općoj ili apstraktnoj razini, a ne na detaljnoj hardverskoj razini.

Knjižnica je zamijenjena sa esp-hal-common koja je [no_std] knjižnica, što znači da nema koda unaprijed napisanog na čip. Esp-hal-common je više ograničena vezano za količina funkcija dostupne uz nju.

Statične globalne varijable-pravilno

Globalne varijable mogu ponuditi naizgled prikladan i jednostavan način za dijeljenje podataka u različitim dijelovima programa, nedostaci koje donose razvoju softvera ne mogu se zanemariti. Zamke globalnih varijabli, uključujući njihov negativan utjecaj na čitljivost, mogućnost održavanja, enkapsulaciju, testiranje i skalabilnost, čine ih nepoželjnim izborom u

većini scenarija programiranja. Kako zajednica razvoja softvera teži čišćem, modularnijem kodu koji se može održavati, programere se potiče da minimiziraju upotrebu globalnih varijabli u korist alternativnih pristupa koji daju prioritet enkapsulaciji, modularnosti i pravilnom opsegu. Pridržavajući se ovih načela, programeri mogu stvoriti softver koji nije samo robustan i pouzdan nego i pristupačniji kolegama programerima, utirući put učinkovitoj suradnji i praksi održivog razvoja softvera.

```
1. static TIMER1: Mutex<RefCell<Option<Timer<Timer0<TIMG1>>>>> =
Mutex::new(RefCell::new(None));
2. static ARRAY: Mutex<RefCell<Option<Vec<u8, 128>>>> =
Mutex::new(RefCell::new(None));
3. static PIN:
Mutex<RefCell<Option<esp_hal_common::adc::AdcPin<esp_hal_common::gpio:
:GpioPin<esp_hal_common::gpio::Analog, 0>,
esp_hal_common::analog::ADC1>>>> = Mutex::new(RefCell::new(None));
4. static ADCRE1:Mutex<RefCell<Option<ADC<'_, ADC1>>>>  =
Mutex::new(RefCell::new(None));
5.
```

Rust nameće stroga pravila vlasništva i posuđivanja kako bi spriječio utrku podataka i nesigurnost memorije. Korištenje Mutexa i RefCell-a u ovim globalnim varijablama osigurava siguran istovremeni pristup promjenjivim podacima. Mutex nameće međusobno isključivanje, dopuštajući samo jednoj niti da pristupi podacima u isto vrijeme, dok RefCell omogućuje unutarnju promjenjivost, dopuštajući mutaciju podataka čak i kada je vanjska referenca posuđena.

U Rust-u se općenito izbjegavaju promjenjive globalne varijable zbog potencijalnih problema s paralelnošću. Međutim, kada trebate upravljati globalnim promjenjivim stanjem, upotreba konstrukcija kao što su Mutex i RefCell je preporučeni pristup jer oni provode Rust-ova pravila o vlasništvu i posuđivanju tijekom izvođenja. To pomaže u sprječavanju utrkivanja podataka, zastoja i drugih grešaka povezanih s paralelnošću.

Imajte na umu da, iako ove konstrukcije pružaju sigurnost, bitno ih je koristiti razumno i promišljeno jer pretjerano zaključavanje i sinkronizacija mogu dovesti do uskih grla u izvedbi. Osim toga, trebalo bi postojati odgovarajuće rukovanje pogreškama kada se radi o zaključavanjima i promjenjivom dijeljenom stanju.. [2]

4.2.1 Mutex

Mutexi su primitivan tip podataka koji služi za sigurnu izmjenu podataka. Mutex će blokirati dretve koje čekaju da zaključavanje postane dostupno. Svaki mutex ima parametar tipa koji predstavlja podatke koje štiti. Podacima se može pristupiti samo putem RAII čuvara vraćenih iz lock i try_lock, koji jamče da se podacima može pristupiti samo kada je mutex zaključan

Mutex se smatra otrovanim kad god dretva paničari dok drži mutex. Nakon što je mutex otrovan, sve ostale dretve ne mogu pristupiti podacima prema zadanim postavkama jer su vjerojatno zaražene

4.2.2 RefCell

`RefCell<T>` koristi Rust-ov životni vijek za implementaciju "dinamičkog posuđivanja", procesa u kojem se može tražiti privremeni, ekskluzivni, promjenjivi pristup unutarnjoj vrijednosti. Posudbe za *`RefCell<T>`*s prate se tijekom izvođenja, za razliku od izvornih tipova referenci Rust-a koji se u potpunosti prate statički, tijekom prevođenja.

Nepromjenjiva referenca na unutarnju vrijednost RefCell-a (&T) može se dobiti s borrow, a promjenjiva posudbe (&mut T) može se dobiti s borrow_mut. Kada se te funkcije pozovu, one prvo provjeravaju jesu li zadovoljena Rustova pravila posuđivanja: dopušten je bilo koji broj nepromjenjivih posuđivanja ili je dopuštena jedna nepromjenjiva posudbe, ali nikada oboje. Ako se pokuša posuditi koji bi prekršio ova pravila, dretva će se uspaničiti.

4.2.3 Critical_section

Tokom pisanja softvera za ugradbene sustave, uobičajeno je koristiti "kritični odjeljak" kao osnovnu primitivnu za kontrolu sinkronizacije programa. Kritični odjeljak je zapravo Mutex koji je globalan za cijeli proces, koji može dobiti samo jedna dretva u isto vrijeme. To se može koristiti za zaštitu podataka iza mutexa, za emulaciju atoma u ciljevima koji ih ne podržavaju itd.

Za bare-metal jedno-jezgreni sustav, onemogućavanje prekida u trenutnoj (jedinoj) jezgri.

4.3 ADC i Prekidi

ADC i prekidi drugačije funkcioniraju kada se radi o „bare metal“ pristupu . Glavna razlika koja se mogla prepoznati je da je puno sličniji C kodu.

```
    let peripherals = Peripherals::take();
    let mut rtc = Rtc::new(peripherals.RTC_CNTL);
    let mut adc1_config = AdcConfig::new();
    adc1_config.resolution=Resolution::Resolution12Bit;
    let io = IO::new(peripherals.GPIO, peripherals.IO_MUX);
    let mut system = examples_util::system!(peripherals);
    let clocks = examples_util::clocks!(system);
    let analog = examples_util::analog!(peripherals);
    let mut pin:
esp_hal_common::adc::AdcPin<esp_hal_common::gpio::GpioPin<esp_hal_comm
on::gpio::Analog, 0>, esp_hal_common::analog::ADC1> =
adc1_config.enable_pin(io.pins.gpio0.into_analog(),
Attenuation::Attenuation11dB);
    let mut adc1: ADC<''_, ADC1> = ADC::<ADC1>::adc(
```

```
        &mut system.peripheral_clock_control,
        analog.adc1,
        adc1_config,
    )
    .unwrap();
```

Programski kod 4-7 Inicijalizacija početnih varijabli

Inicjalizacija u „bare metal“ pristupu zahtjeva dublju konfiguraciju opcija. Razne knjižnice za pristup određenim značajkama. Za ADC je potrebno konfigurirati razlučivost i prigušenije. Za pristup ADC pinu potrebno je prvo deklarirati periferije te u knjižnici *esp_hal_common::gpio::GpioPin* potrebno je konfigurirati novi ADC pin.

Pin deklariramo kao novi ADC pin u kojem preko prethodno definirane io varijable koristimo *into_analog()* metodu da iz pretvorimo strukturu GpioPin u AdcPin i postavimo potrebne ADC zastavice.

```
let timer_group1 = TimerGroup::new(
        peripherals.TIMG1,
        &clocks,
        &mut system.peripheral_clock_control,
    );
    let mut timer1 = timer_group1.timer0;
    let mut wdt1 = timer_group1.wdt;
    rtc.swd.disable();
    rtc.rwdt.disable();
    wdt1.disable();
    interrupt::enable(
        esp_hal_common::peripherals::Interrupt::TG1_T0_LEVEL,
        interrupt::Priority::Priority1,
    )
    .unwrap();
    let hz= fugit::HertzU64::Hz(30000);
    timer1.start(hz.into_duration());
    let mut arraytemp: Vec<u8, 128>= Vec::new();
    timer1.listen();
        critical_section::with(|cs| {
        TIMER1.borrow_ref_mut(cs).replace(timer1);
        PIN.borrow_ref_mut(cs).replace(pin);
        ADCRE1.borrow_ref_mut(cs).replace(adc1);
        ARRAY.borrow_ref_mut(cs).replace(arraytemp);
    });
```

Programski kod 4-8 Primjer Kritičnog odjeljaka

Kritični odjeljak unosi se korištenjem *critical_section::with(|cs| { ... });*.

Razne globalne varijable (*TIMER1, PIN, ADCRE1, ARRAY*) popunjavaju se vrijednostima (*timer1, pin, adc1, arraytemp*) unutar kritičnog odjeljka.

Ovo osigurava siguran istovremeni pristup ovim varijablama.

Ovaj blok koda u biti inicijalizira i konfigurira periferiju ADC1 za analogno-digitalnu pretvorbu. Također postavlja mjerač vremena, inicijalizira globalne varijable i osigurava da se sve to radi na način koji je siguran za istovremenost korištenjem kritičnih odjeljaka. To je dio veće postavke sustava, vjerojatno unutar ugradbenog okruženja, gdje su sinkronizacija i pažljiva inicijalizacija ključni za sprječavanje problema poput utrke podataka i netočne konfiguracije.Poziv interrupta se dešava u odvojenoj funkciji

```
#[interrupt]
fn TG1_T0_LEVEL() {
    critical_section::with(|cs| {
        let mut timer1 = TIMER1.borrow_ref_mut(cs);
        let timer1 = timer1.as_mut().unwrap();
        timer1.clear_interrupt();
        let a = fugit::HertzU64::Hz(30000);
        timer1.start(a.into_duration());

        let mut array = ARRAY.borrow_ref_mut(cs);
        let mut arrays = array.as_mut().unwrap();
            if arrays.len()>=127 {
            arrays.clear();
        }


        let mut adc1 = ADCRE1.borrow_ref_mut(cs);
        let mut adc1 = adc1.as_mut().unwrap();
        let mut pin = PIN.borrow_ref_mut(cs);
        let mut pin = pin.as_mut().unwrap();
        let pin_value:u16 = nb::block!(adc1.read(&mut
pin)).unwrap();

        let atten = Attenuation::Attenuation11dB;
        let pin_value_mv = pin_value as u32 * atten.ref_mv() as
u32 / 4096;
        arrays.push(pin_value_mv as u8);

    });

```

Programski kod 4-9 Primjer prekidnog potprograma s ADC-om

Kao što je napomenuto, prije critical section onesposobi sve moguće prekide te onesposobljava mogući data race za globalne. Za slučaj da niz naraste previše, moram ga počistiti zbog toga što korištenje clear funkcije javlja samo da se taj memorijski blok oslobodio i nestati će kada program shvati da nije više potreban te će spremiti vrijednost očitanu sa pina u globalni ***ARRAY*** tj. u posuđenu referencu globalne varijable.

4.4 Bluetooth oglašavanje

U ovom principu sam odlučio raditi asinkrono Bluetooth oglašavanje. Rust-ov async/await omogućuje jednostavno i učinkovito obavljanje više zadataka bez presedana u ugradbenim sustavima. Zadaci se u vrijeme prevođenja pretvaraju u automate stanja koji se pokreću kooperativno. Ne zahtijeva dinamičku dodjelu memorije i radi na jednom stogu, tako da nije potrebno podešavanje veličine stoga po zadatku. Potpuno je unaprijedio i zanemario potrebu za tradicionalnim RTOS-om s promjenom konteksta jezgre, te ju ubrzao i smanjio.

```
1.      embassy::init(&clocks, timer_group0.timer0);
2.      let executor = EXECUTOR.init(Executor::new());
3.      executor.run(|spawner| {
4.          spawner.spawn(run(init, bluetooth,button)).ok();
5.
6.      });
```

Programski kod 4-10 Inicijalizacija asinkronog BLE servisa

Ovaj async/await izvršitelj dizajniran je za ugradbene sustave, s naglaskom na minimalnu upotrebu resursa i učinkovito upravljanje zadacima. Prethodno dodjeljuje „buduće“ zadatke, ne zahtijeva fiksne postavke kapaciteta i uključuje integrirani sustav odbrojavanja za zadatke u mirovanju. Izvršitelj štedi CPU izbjegavajući zauzeto prozivanje i koristi ciljano prozivanje za probuđene zadatke. Provodi pravednost, sprječava bilo koji zadatak da monopolizira CPU vrijeme. Dodatno, podržava više instanci, omogućujući različite razine prioriteta zadataka za preventivno planiranje.

```
1.      let connector = BleConnector::new(&init, &mut bluetooth);
2.      let mut ble = Ble::new(connector, esp_wifi::current_millis);
```

Programski kod 4-11 Inicijalizacija Bluetooth drivera

Za inicijalizaciju i početak GATT reklamiranja potrebno je uzeti BleConnector i povezati ga sa timerom kako bi se pravilno vršilo asinkrona komunikacija sa Bluetooth servisima.

```
 1. println!("{:?}", ble.init().await);
 2.         println!("{:?}",
ble.cmd_set_le_advertising_parameters().await);
 3.         println!(
 4.             "{:?}",
 5.             ble.cmd_set_le_advertising_data(
 6.                 create_advertising_data(&[
 7.                     AdStructure::Flags(LE_GENERAL_DISCOVERABLE |
BR_EDR_NOT_SUPPORTED),
 8.
AdStructure::ServiceUuids16(&[Uuid::Uuid16(0x1809)]),
 9.
AdStructure::CompleteLocalName(examples_util::SOC_NAME),
```

```
                ])
                .unwrap()
            )
            .await
        );
        println!("{:?}",
ble.cmd_set_le_advertise_enable(true).await);
```

Programski kod 4-12 Postavljanje osnovnih parametra za BLE oglašavanje

Ble instanca se koristi za inicijalizaciju BLE periferije metodom init, a rezultat se ispisuje pomoću println!.

Metoda *cmd_set_le_advertising_parameters* poziva se na ble instanci za postavljanje LE parametara oglašavanja, a rezultat se ispisuje.

Metoda *cmd_set_le_advertising_data* poziva se na ble instanci, konfigurirajući podatke o oglašavanju sa zastavicama, UUID-ovima usluga i lokalnim nazivom. Rezultat se ispisuje.

Metoda *cmd_set_le_advertise_enable* poziva se na ble instanci da omogući LE oglašavanje, a rezultat se ispisuje.

Ovaj kod iskorištava Rust-ove asinkrone mogućnosti (async/await) za upravljanje konfiguracijom i aktivacijom BLE periferije. Println! makronaredbe se koriste za prikaz ishoda ovih asinkronih operacija dok se one dovršavaju. Sveukupna svrha je postaviti i aktivirati BLE oglašivačku funkcionalnost za ugradbeni sustav, omogućujući mu interakciju s drugim uređajima putem Bluetooth Low Energy.

```
let mut rf = |_offset: usize, data: &mut [u8]| {
            critical_section::with(|cs|{
            let mut array = ARRAY.borrow(cs).borrow_mut();
            let mut arrays = array.as_mut().unwrap();
            data[..arrays.len()].copy_from_slice(&arrays[..]);
            arrays.clear();
            let mut timer1 = TIMER1.borrow_ref_mut(cs);
            let timer1 = timer1.as_mut().unwrap();
            timer1.clear_interrupt();
            let a = fugit::HertzU64::Hz(30000);
            timer1.start(a.into_duration());
            }
            );
            data.len()-9
        };
        let mut wf = |offset: usize, data: &[u8]| {
```

```
17.             println!("RECEIVED: {} {:x?}", offset,
str::as_ascii(data));
18.         };
19.    gatt!([service {
20.             uuid: "937312e0-2354-11eb-9f10-fbc30a62cf38",
21.             characteristics: [
22.                 characteristic {
23.                     uuid: "937312e0-2354-11eb-9f10-fbc30a62cf38",
24.                     read: rf,
25.                     write: wf,
26.                 },
27.             ],
28.         },]);
```

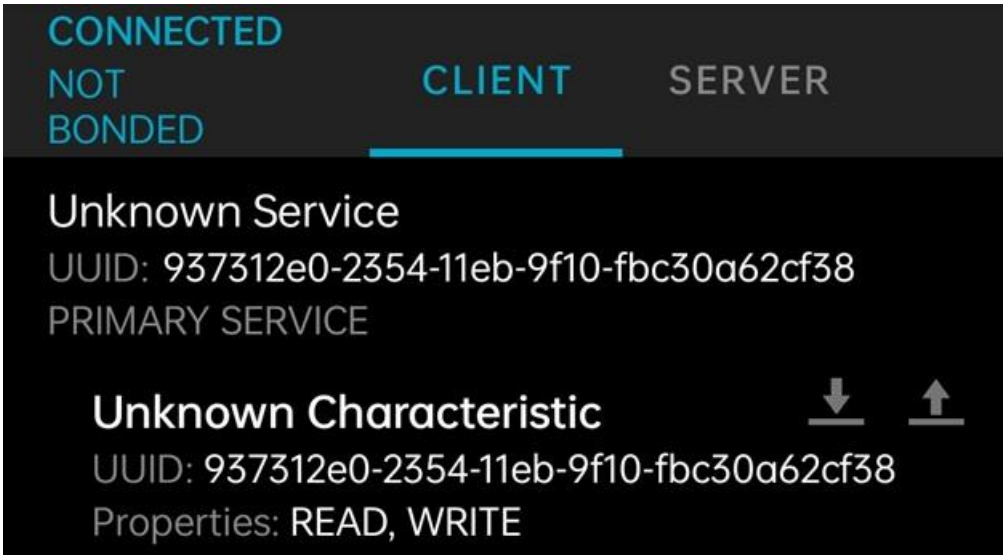
Programski kod 4-13 Postavljanje GATT karakteristika i servisa

Funkcija *rf* se poziva tokom čitanja Bluetooth karakteristike, dok *wf* se koristi za slučajeve kada se primaju podatci od poslužitelja. Za pozivanje *rf* potrebno je opet koristiti globalnu varijablu i zato je potrebno ponovno pokrenuti prekidni tajmer jer *critical_section* zaustavlja prekide.

```
1 let mut srv = AttributeServer::new(&mut ble, &mut gatt_attributes);
2 let counter = RefCell::new(0u8);
3 srv.run(&mut notifier).await.unwrap();
```

Programski kod 4-14 Oglašavanje GATT servisa

Nakon toga potrebno je samo pokrenuti attribute server. To je to sa GATT oglašavanjem. Koristeći mobilnu aplikaciju NRF connect provjeravamo stanje GATT-a.



Slika 4.2 Prikaz GATT karakteristike u aplikaciji NRF connect

Gatt karakteristika je vidljiva (Slika 4.2) i prijenos je uspješan te je vrijeme da se krene na poslužiteljsku stranu projekta.

4.5 Primanje GATT podataka

BlueZ je službeni skup Linux Bluetooth protokola i nudi alate i biblioteke za rad s Bluetooth uređajima. Potrebno je prvo instalirati BlueZ

Otkrivanje BLE uređaja:

Koristite BlueZ alate ili biblioteke za otkrivanje BLE uređaja u blizini. Možete koristiti alat naredbenog retka hcitool ili biblioteku pybluez u Pythonu da biste to postigli.

Povezivanje s BLE uređajima:

Nakon što ste otkrili ciljni BLE uređaj, trebate uspostaviti vezu s njim. BlueZ pruža naredbe i API-je za pokretanje veze.

Pregled usluga i karakteristike GATT-a:

Nakon uspostavljanja veze, možete pregledavati GATT usluge i karakteristike koje nudi uređaj. To uključuje čitanje GATT baze podataka uređaja, koja uključuje informacije o dostupnim uslugama i njihovim atributima.

Čitanje podatke karakteristike GATT-a:

Nakon što ste identificirali specifično GATT obilježje koje vas zanima, možete čitati podatke iz njega. Svaka karakteristika ima UUID (Universally Unique Identifier) koji ćete koristiti za identifikaciju.

4.6 Pretvaranje podataka u text

Kao opcije za odabir modela za prepoznavanje govora tri glavna modela otvorenog izvora

Razlog za korištenje Whispera je zato što Whisper pruža preciznost znatno veću od alternativa poput wav2vec2 i Kaldi

Whisper, koji je razvio Facebook AI, učinkovit je sustav otvorenog koda za automatsko prepoznavanje govora (ASR) poznat po svojoj arhitekturi neuronske mreže i snažnim performansama kada se fino podešava na određenim domenama. Wav2Vec, iz Hugging Face-a, fokusiran je na samo nadzirano učenje i koristi arhitekture temeljene na transformatorima, čineći ga prilagodljivim raznim ASR zadacima kroz fino podešavanje. Kaldi, jedan od najstarijih i naširoko korištenih alata za ASR, nudi modularnost i opsežne mogućnosti prilagodbe za istraživače i programere, što ga čini popularnim izborom za izradu prilagođenih ASR(Automatic Speech Recognition) i skaliranje na velike skupove podataka. Odabir između ovih biblioteka ovisi

o zahtjevima projekta, pri čemu Whisper i Wav2Vec naglašavaju jednostavnost korištenja i preobuku, dok Kaldi pruža robusnu osnovu za prilagođena ASR rješenja i istraživačke napore.

Istraživanje podupire korištenje Whispera testiranjem na 5 drugačijih domena.

Tablica 4.1 metrike za usporedbu modela preuzeto iz [11]

Domena	**Opis**	**Prosječno trajanje [min]**
razgovorni AI	Spontani govor iz interakcije ljudi/chat bota, uključujući različite slučajeve upotrebe naručivanja brze hrane. Izazovan zvuk s šumom i pozadinskim govorom. Mnogo kratkih datoteka.	0.65
Telefonski poziv	Spontani razgovorni govor iz telefonskih poziva. Uglavnom s dva zvučnika. Uključuje slučajeve upotrebe pozivnog centra. Općenito niske kvalitete, zvuk od 8 kHz s pozadinskim govorom.	2.2
Sastanak	Spontani razgovorni govor sa sastanaka s više osoba, uključujući i osobne sastanke i videokonferencije. Datoteke su obično duge s velikim brojem govornika.	36.1
Pozivi zarade	Javni pozivi zarade s općenito visokokvalitetnim zvukom. Uglavnom uvježban, čitan govor. Puno financijskog žargona i izgovorenih brojeva. Datoteke su obično duge	2.9
Video	Videozapisi s YouTubea i drugih izvora koji pokrivaju različite sadržaje. Uglavnom spontani govor. Općenito visokokvalitetni zvuk visoke brzine uzorkovanja.	39.5

Bitan fokus ovdje je 8kHz telefonski poziv jer primjena je slična niskokvalitetnom audio izvoru.

Također, bitno je za napomenuti da je mjera preciznosti utvrđena korištenjem WER sustava. Stopa pogreške riječi (WER)(4.1) uobičajena je metrika performansi sustava za prepoznavanje govora ili strojnog prevođenja.

Opća poteškoća u mjerenju uspješnosti leži u činjenici da prepoznati niz riječi može imati različitu duljinu od referentnog niza riječi (navodno ispravnog). WER je vrijedan alat za usporedbu različitih sustava kao i za procjenu poboljšanja unutar jednog sustava. Ova vrsta mjerenja,

međutim, ne daje pojedinosti o prirodi pogrešaka u prijevodu i stoga je potreban daljnji rad kako bi se identificirali glavni izvor(i) pogreške i usredotočili svi istraživački napori.

Ovaj se problem rješava tako da se prepoznati slijed riječi prvo poravna s referentnim (izgovorenim) nizom riječi pomoću dinamičkog poravnanja niza. Ispitivanje ovog pitanja vidi se kroz teoriju nazvanu zakon potencije koja navodi korelaciju između zbunjenosti i stope pogrešaka riječi.

$$WER = \frac{S + D + I}{N} \quad (4.1)$$

- **S** je broj zamjena,
- **D** je broj brisanja,
- **I** je broj umetanja,
- **C** je broj točnih riječi,
- **N** je broj riječi u referenci (N=S+D+C)

Primjenjujući WER na spomenute modele rezultati su jasni.

Tablica 4.2 WER usporedba različitih modela preuzeto iz [11]

Skup podataka	Kaldi	wav2vec 2.0	Whisper
Razgovorni AI	65.9	33.3	5.7
Telefonski poziv	78.1	40.9	20.1
Sastanak	54.3	38.7	18.6
Pozivi zarade	68.8	26.4	8.9
Video	35.4	17.2	5.2

Word Error Rate (WER) za tri različite biblioteke za prepoznavanje govora, Kaldi, Wav2Vec 2.0 i Whisper, u različitim domenama. WER mjeri točnost ovih ASR sustava, s nižim vrijednostima koje ukazuju na bolje performanse.

Razgovorna umjetna inteligencija: U domeni razgovorne umjetne inteligencije, Whisper ima najniži WER od 5,7, što ukazuje na najveću točnost među tri biblioteke. Kaldi ima znatno veći WER od 65,9, dok je Wav2Vec 2.0 između s WER-om od 33,3.

Telefonski pozivi: Za telefonske pozive, Whisper još uvijek ima najniži WER od 20,1, što ga čini najpreciznijim izborom. Wav2Vec 2.0 ima WER od 40,9, a Kaldi ima najveći WER od 78,1.

Sastanci: U domeni Sastanci, Whisper održava svoju poziciju najtočnije knjižnice s WER-om od 18,6. Slijedi Wav2Vec 2.0 s WER-om od 38,7, dok Kaldi ima WER od 54,3.

Videozapisi: za videozapise, Whisper ostaje najtočnija opcija s WER-om od 35.4. Wav2Vec 2.0 je druga najbolja izvedba s WER-om od 17.2 a Kaldi zaostaje s WER-om od 35.4.

Pozivi o zaradi: U domeni poziva o zaradi, Whisper ima najniži WER od 8,9, što ukazuje na najveću točnost, a slijedi ga Wav2Vec 26.4 s WER-om od 68.4. Kaldi ima najviši WER od 35,4.

Ukratko, na temelju dostavljenih podataka, Whisper dosljedno nadmašuje Kaldi i Wav2Vec 2.0 u smislu stope pogrešaka riječi u različitim domenama. Postiže najniže vrijednosti WER, što označava najveću točnost u transkripciji govornog jezika u tekst u ovim specifičnim kontekstima.

Za pretvaranje podataka u text koristio sam knjižnicu Whisper od OpenAi-a.

Whisper nudi prikladan i jednostavan način integracije mogućnosti automatskog prepoznavanja govora u male projekte koji zahtijevaju točnu transkripciju govornog jezika.

Implementacija Whispera s obzirom na ostalo je prilično jednostavna (Programski kod 4-15). [11]

```
1. model = whisper.load_model("base")
2. result = model.transcribe("filename-1.wav")
3. print(result["text"])
```

Programski kod 4-15 Pretvaranje Zvuka u tekst

4.7 Slanje teksta jezičnom modelu

Kao dio istraživanja obrade prirodnog jezika, potreba za LLM-om (Large language model) otvorenog koda koji se može lako replicirati i prilagoditi za razne aplikacije. Konkretno, potreba je za model koji se može fino podesiti za različite slučajeve upotrebe, budući da će sustav morati prihvatiti niz jezičnih varijacija i nijansi. Sposobnost modificiranja i prilagođavanja modela specifičnim kontekstima presudna je za postizanje optimalnih rezultata i osiguravanje svestranosti sustava

Meta je objavila svoj LLAMA-2 LLM 18. srpnja 2023. Najnovija verzija LLAMA koje je dostupna pojedincima, kreatorima, istraživačima i tvrtkama svih veličina kako bi mogli eksperimentirati, inovirati i odgovorno skalirati svoje ideje.

To izdanje uključuje težine modela i početni kod za unaprijed obučene i fino podešene modele jezika LLAMA — u rasponu od 7×10^9 do 70×10^9 parametara.

Istraživanje modeli na standardnim akademskim mjerilima. sve javno objavljene rezultate.

Tablica 4.3 Usporedba LLM-ova preuzeto iz [12].

Model	Veličina	Kod	zaključivanje	Poznavanje svijeta	čitanje s razumijevanjem	Matematika
MPT	7B	20.5	57.4	41.0	57.5	4.9
	30B	28.9	64.9	50.0	64.7	9.1
Falcon	7B	5.6	56.1	42.8	36.0	4.6
	40B	15.2	69.2	56.7	65.7	12.6
LLAMA 1	7B	14.1	60.8	46.2	58.5	6.95
	13B	18.9	66.1	52.6	62.3	10.9
	33B	26.0	70.0	58.4	67.6	21.4
	65B	30.7	70.7	60.5	68.6	30.8
LLAMA 2	7B	16.8	63.9	48.9	61.3	14.6
	13B	24.5	66.9	55.4	65.8	28.7
	34B	27.8	69.9	58.7	68.0	24.2
	70B	37.5	71.9	63.6	69.4	35.2

Model: Ovaj stupac označava naziv modela ili sustava koji se ocjenjuje. Spominju se tri modela: MPT, Falcon i LLAMA 1 & 2.

Veličina: Ovaj stupac predstavlja veličinu ili kapacitet modela, koji se može mjeriti u milijardama parametara (B). Na primjer, "7B" znači 7 milijardi parametara, a "30B" znači 30 milijardi parametara.

Kod: Čini se da je ovaj stupac numerički rezultat ili ocjena povezana s kvalitetom "koda" modela. Više vrijednosti vjerojatno ukazuju na bolju kvalitetu koda [13].

Zaključivanje: Ovaj stupac predstavlja rezultat ili ocjenu koja se odnosi na sposobnost modela da izvrši zadatke zdravorazumskog rasuđivanja. Više vrijednosti sugeriraju bolju izvedbu u ovoj kategoriji [14].

Poznavanje svijeta: čini se da ovaj stupac predstavlja znanje modela o svijetu. Više vrijednosti vjerojatno ukazuju na veću sposobnost razumijevanja i razmišljanja o svijetu [15].

Čitanje s razumijevanjem: Ovaj stupac predstavlja izvedbu modela u zadacima razumijevanja pročitanog teksta. Viši rezultat ukazuje na bolje sposobnosti razumijevanja pročitanog [14].

Matematika: Ovaj stupac označava izvedbu modela u zadacima povezanim s matematikom. Više vrijednosti sugeriraju bolje matematičke sposobnosti [16].

Na izbor modela LLAMA 2 možda su utjecale njegove superiorne performanse u više kategorija u usporedbi s drugim modelima. Konkretno, čini se da LLAMA 2 briljira u

zdravorazumskom razmišljanju, poznavanju svijeta, razumijevanju pročitanog i matematičkim zadacima, o čemu svjedoče njeni dosljedno visoki rezultati u različitim veličinama (npr. 7B, 13B, 34B i 70B).

Model LLAMA 2 nadmašuje konkurente kao što su MPT i Falcon u raznim aspektima, što ga čini atraktivnim izborom za zadatke koji zahtijevaju dobro razumijevanje prirodnog jezika i sposobnosti zaključivanja. Dodatno, njegova skalabilnost na veće veličine (npr. 70B) omogućuje još snažniju izvedbu pri rješavanju složenih jezičnih zadataka. U konačnici, odluka o odabiru LLAMA 2 vjerojatno proizlazi iz njegovih dobro zaokruženih i visokoučinkovitih mogućnosti u različitim dimenzijama razumijevanja jezika. Te LLM koji je očit odabir je LLAMA2.

Za korištenje LLAMA2 modela potrebne ja instalacija dvije knjižnice: PyTorch i CUDA

PyTorch je optimizirana biblioteka tenzora za duboko učenje pomoću GPU-a i CPU-a. Nužna je za daljnji rad sustava.

NVIDIA® CUDA® Toolkit pruža razvojno okruženje za stvaranje visokoučinkovitih GPU-ubrzanih aplikacija. Uz CUDA Toolkit moguće je razvijati, optimizirati i implementirati aplikacije na GPU-ubrzanim ugradbenim sustavima.

Da biste instalirali PyTorch, možete slijediti ove općenite korake. Imajte na umu da se određene naredbe i opcije mogu razlikovati ovisno o vašem operativnom sustavu, konfiguraciji hardvera i verziji PyTorcha koju želite instalirati. Od mog posljednjeg ažuriranja znanja u rujnu 2021., pružit ću vam uobičajene metode za instaliranje PyTorcha pomoću pipa i conde. Metoda kojom ovaj papir pokriva je pip (pythonov upravitelj paketa).

Instalacija PyTorch:

Za instalaciju samo na CPU (bez podrške za GPU):

```
pip install torch
```

Za GPU podršku (zahtijeva kompatibilni NVIDIA GPU i CUDA toolkit):

Instalacija CUDA za Python uključuje nekoliko koraka jer zahtijeva instaliranje CUDA alata i GPU drajvera na vašem sustavu. CUDA je paralelna računalna platforma i API koje je razvila NVIDIA, a omogućuje Python programerima da iskoriste snagu NVIDIA GPU-a za ubrzano računalstvo. Ovdje je opći vodič o tome kako instalirati CUDA za Python:

Provjera kompatibilnost GPU-a:

Potrebno je prvo provjeriti je li GPU kompatibilan sa CUDA na službenoj web stranici NVIDIA-e(https://developer.nvidia.com/cuda-gpus#compute) kako biste provjerili podržava li GPU CUDA i koja je verzija CUDA kompatibilna s njim.

Instalacija upravljačkih programe za GPU:

Prije instaliranja CUDA-e, potrebno je provjeriti verziju instaliranih GPU upravljačke programe za serverovu NVIDIA grafičku karticu. Upravljački programi se mogu preuzeti s NVIDIA-ine web stranice.

Preuzimanje CUDA Toolkit:

Posjet stranici za preuzimanje NVIDIA CUDA Toolkita (https://developer.nvidia.com/cuda-toolkit-archive). Potrebno je odabrati verziju CUDA Toolkita koja je kompatibilna s serverovim GPU-om i operativnim sustavom i preuzeti instalacijski program.

Instalacija CUDA Toolkit:

Pri pokretanju instalacijskog programa CUDA Toolkit trebalo bi slijedite upute na zaslonu. Preporučeno je odabrati prilagođenu instalaciju kako bi odabrali komponente koje su potrebne. Tipična instalacija uključuje CUDA driver, biblioteke za izvađanje i razvojne alate. Nakon toga možemo započeti sa instalacijom razgovornog modela.

Instalacija LLAMA2:

Potrebno je napraviti zahtjev za pristup modelu na stranici (https://ai.meta.com/resources/models-and-libraries/llama-downloads/). Nakon primljenog odobrenja potrebno je klonirati repozitorij (https://github.com/facebookresearch/llama). Zatim, potrebno je pokrenuti download.sh i pratiti upute kojih konzola vodi. U repozitoriju nalaze se dva primjera uporabe modela example.py.

Uz minimalnu prilagodbu moguće je koristiti vlastite prompt-ove. Za prijenos natrag na Bluetooth uređaj potrebno je samo ponoviti GATT prijenos spomenut u poglavlju 4.5.

4.8 Razlika ugradbenim računala

U usporednoj analizi između dvaju radova, od kojih se jedan fokusira na korištenje STM32 mikrokontrolera, a drugi naglašava implementaciju pomoću Espressif tehnologije, pojavljuju se intrigantne razlike u njihovim pristupima. Unutar područja ugradbenih sustava i aplikacija

mikrokontrolera, i STM32 tvrtke STMicroelectronics i Espressif nude različite razine implementacije, od kojih svaka zadovoljava specifične filozofije dizajna i prioritete.

Rad koji istražuje STM32 mikrokontrolere otkriva područje koje karakterizira pedantna pažnja na zamršenosti na razini hardvera. Uz STMicroelectronics STM32 seriju, predstavljen je raznolik spektar opcija mikrokontrolera, od kojih je svaka fino podešena da zadovolji specifične domene primjene. Rad razjašnjava zamršene detalje značajki STM32, uključujući napredne periferne uređaje, konfiguracije memorije i mogućnosti rada u stvarnom vremenu. Naglasak je stavljen na robusnost hardverskog ekosustava STM32, što ga čini posebno pogodnim za scenarije koji zahtijevaju zahtjevno računanje, preciznost i deterministično ponašanje.

S druge strane, rad koji se fokusira na Espressifovu tehnologiju oslikava portret krajolika kojeg više pokreću povezanost i integracija ekosustava. Espressifova ponuda, osobito serija ESP32, poznata je po besprijekornoj integraciji Wi-Fi i Bluetooth mogućnosti, služeći se aplikacijama koje zahtijevaju bežičnu komunikaciju. Rad se bavi Espressifovim programskim paketom, koji uključuje biblioteke i alate posebno dizajnirane za omogućavanje brzog razvoja i povezivanja. Ovdje je naglasak na omogućavanju programerima da brzo i učinkovito kreiraju uređaje s omogućenim internetom, koristeći Espressifov sveobuhvatni softverski okvir.

Razlika vrijedna pažnje leži u razinama apstrakcije kojoj svaka tehnologija daje prioritet. STM32, kao što je istaknuto u radu, gravitira hardverskim zamršenostima, što ga čini idealnim izborom za projekte u kojima su fino podešavanje i optimizacija najvažniji. Espressif se, kao što je objašnjeno, fokusira na pružanje ekosustava više razine, usmjeravajući razvojni proces za projekte koji daju prioritet brzoj izradi prototipova i povezivosti.

4.9 Razlike

Bez sumnje, usporedba između primjera iz 2018. i trenutnog stanja tehnologije naglašava značajnu evoluciju u području programiranja mikrokontrolera. Oštra razlika u pristupu između ova dva razdoblja naglašava transformativni napredak koji je u posljednjih nekoliko godina učinio manipulaciju bitovima niske razine manje bitnom.

U primjeru iz 2018.g., potreba za pristupom niske razine bila je potaknuta ograničenjima dostupnih hardverskih i softverskih alata. Tada je programiranje mikrokontrolera često zahtijevalo izravnu manipulaciju pojedinačnim bitovima i registrima kako bi se postigla željena funkcionalnost. Ovaj pristup zahtijevao je dubinsko poznavanje detalja hardvera, kao i razumijevanje arhitekture mikrokontrolera. Iako učinkovit, bio je zamršen, sklon pogreškama i

dugotrajan, posebno za složene zadatke. Zahtijevanje direktnom pristupu prekidnom sustavu NVIC(Programski kod 4-16) [7]

```
p.device.AFIO.exticr1.write(|w| unsafe { w.exti0().bits(0)
});
let _int0 = gpioa.pa0.into_floating_input(&mut gpioa.crl);
unsafe {
p.core.NVIC.set_priority(
hal::stm32f103xx::interrupt::Interrupt::EXTI0, 1);
}
p.core.NVIC
.enable(hal::stm32f103xx::interrupt::Interrupt::EXTI0);
p.device.EXTI.imr.write(|w| w.mr0().set_bit());
// unmask the interrupt (EXTI)
p.device.EXTI.emr.write(|w| w.mr0().set_bit());
p.device.EXTI.rtsr.write(|w| w.tr0().set_bit());
// trigger interrupt on falling edge
rtfm::set_pending(Interrupt::EXTI0);
```

Programski kod 4-16 Primjer Pristupa Prekidima u 2018 preuzeto iz [7]

```
let mut timer: TimerDriver<'_> =  TimerDriver::new(periph.timer00,
&config).unwrap();
    let mut x = 0;
    timer.enable_interrupt();
    timer.enable_alarm(true).unwrap();
    timer.enable(true).unwrap();
    timer.set_alarm(1000).unwrap();
    {
        let mut add_two_to_x = move || adcfunc( &mut adc, &mut adc1);
        unsafe { timer.subscribe(add_two_to_x).unwrap() };
    }
```

Programski kod 4-17 Primjer prekida u 2022

Brzo premotavanje u sadašnjost i krajolik se značajno promijenio (Programski kod 4-17). Proliferacija snažnih razvojnih okruženja, bogatih biblioteka i apstrakcija više razine drastično je promijenila programsku paradigmu za mikrokontrolere. Suvremene platforme mikrokontrolera,

poput STM32 i Espressifovih ponuda, opremljene su sveobuhvatnim softverskim okvirima koji apstrahiraju mnoge zamršenosti niske razine.

Moderni razvojni alati, integrirana razvojna okruženja (IDE) i biblioteke nude API-je koji pojednostavljuju složene zadatke. Značajke poput rukovanja prekidima, periferne konfiguracije i komunikacijskih protokola sada su apstrahirane u funkcije jednostavne za korištenje. Ovo omogućuje programerima da se više usredotoče na logiku aplikacije, a manje na manipulaciju bitovima niske razine.

Nadalje, porast slojeva hardverske apstrakcije (HAL) i međuprograma premostio je jaz između hardverske interakcije niske razine i aplikacijske logike visoke razine. Ovi slojevi pružaju standardizirana sučelja za interakciju s hardverskim komponentama, štiteći programere od temeljnih zamršenosti. Ova promjena u pristupu dramatično je ubrzala proces razvoja i smanjila vjerojatnost bugova i pogrešaka.

Dok je primjer iz 2018.g. zahtijevao duboko razumijevanje unutarnjih dijelova mikrokontrolera, napredak u proteklih pet godina omogućio je programerima stvaranje sofisticiranih aplikacija s manje oslanjanja na manipulaciju bitovima niske razine. Fokus se pomaknuo prema korištenju programskih konstrukcija više razine, korištenju biblioteka i iskorištavanju prednosti integriranih lanaca alata koje pružaju proizvođači mikrokontrolera.

Naime, evolucija u programiranju mikrokontrolera odražava širi trend u tehnologiji, gdje slojevi apstrakcije i apstrakcije više razine omogućuju brz razvoj bez žrtvovanja kontrole ili funkcionalnosti. Kao rezultat toga, razvojni programeri danas su ovlašteni stvarati zamršene aplikacije bogate značajkama, dok se mogu više usredotočiti na željene rezultate umjesto da se bore sa zamršenostima niske razine.

Zaključak

Kroz ovaj projekt, uspješno je izrađen inovativan i učinkovit ugradbeni sustav, koristeći mogućnosti čipa ESP32-C3 uz snagu programiranja Bare Metal Rust. Ovo postignuće ne samo da naglašava potencijal ugradbenih sustava, već također ističe prednosti novog pristupa interakciji hardvera i AI integraciji.

Korištenje ESP32-C3 BLE GATT (Bluetooth Low Energy Generic Attribute Profile) za komunikaciju postavlja snažne temelje za učinkovitu razmjenu podataka s vanjskim uređajima. Ovaj protokol osigurava besprijekornu povezanost, omogućavajući ugradbenom sustavu besprijekornu interakciju s poslužiteljima i drugim uređajima, poboljšavajući njegove mogućnosti i svestranost.

Usvajanje programiranja Bare Metal Rust u odnosu na tradicionalne operativne sustave u stvarnom vremenu (RTOS) pojavljuje se kao značajka koja definira ovaj projekt. Izravnim sučeljem s hardverom, Bare Metal Rust omogućuje programerima preciznu kontrolu, minimizirane troškove i učinkovitu alokaciju resursa. Ovaj pristup optimizira performanse i odziv sustava, eliminirajući složenosti povezane s RTOS raspoređivanjem i upravljanjem.

U kontekstu programskih jezika, sklonost Rustu u odnosu na C i C++ očita je u uklanjanju uobičajenih zamki i ranjivosti. Rustova stroga sigurnost memorije, model vlasništva i provjera posudbe osiguravaju viši stupanj pouzdanosti i sigurnosti softvera u usporedbi s prirodom C i C++ koja je sklonija greškama. Ovaj izbor smanjuje vjerojatnost curenja memorije, dereferenciranja nultog pokazivača i utrke podataka, jačajući robusnost ugradbenih sustava.

U usporedbi Rust-a za ugradbene sustave u 2018.g. gdje se uvelike oslanjalo na manipulaciju bitovima niske razine u 2023.g. napredak na bibliotekama znatno je pojednostavio programiranje mikrokontrolera, smanjujući potrebu za takvim zamršenim pristupima.

Integracija Whispera za pretvaranje govora u tekst i LLAMA2 za jezični model dodaje sloj sofisticiranosti projektu. Točnost Whispera u transkripciji govora poboljšava sposobnost virtualnog pomoćnika da učinkovito tumači korisnički unos. U međuvremenu, jezični model LLAMA2 koji je svjestan konteksta poboljšava razumijevanje prirodnog jezika, omogućujući sustavu da inteligentnije shvati korisničku namjeru i pruži kontekstualno relevantne odgovore.

Zaključno, uspješna realizacija ovog projekta ugradbenih sustava koji pokreće ESP32-C3 čip, zajedno s programiranjem Bare Metal Rust, pojačava potencijal inovacija na raskrižju hardvera, komunikacijskih protokola i naprednih AI tehnologija. Naglasak projekta na Rustu u odnosu na C i C++, zajedno s integracijom Whispera i LLAMA2, predstavlja primjer holističkog pristupa

razvoju sustava koji daje prednost performansama, sigurnosti i korisničkom iskustvu. Kako se tehnologija nastavlja razvijati, ovaj projekt služi kao dokaz izvanrednih mogućnosti koje proizlaze iz sinergije vrhunskih komponenti i naprednih metodologija, vodeći nas prema budućnosti pametnijih ugradbenih sustava s boljim odzivom i usmjerenih na korisnika.

5 Bibliografija

[1] Karsai i ostali, »Evolving Embedded Systems,« *Computer,* svez. 43, pp. 34-40, 2010.

[2] N. D. Matsakis i F. S. Klock, »The rust language,« *ACM SIGAda Ada Letters,* svez. 34, p. 103–104, listopad 2014.

[3] StackOverflow, »StackOverflow,«, S interneta https://survey.stackoverflow.co/2023/#section-admired-and-desired-programming-scripting-and-markup-languages , kolovoz 2023.

[4] L. Tung, »zdnet,« zdnet, s interneta https://www.zdnet.com/article/google-shows-off-kataos-a-secure-operating-system-written-in-rust/ , kolovoz 2023.

[5] Sam, Scott i June, »googleblog,« Google, S interneta https://opensource.googleblog.com/2022/10/announcing-kataos-and-sparrow.html , kolovoz 2023.

[6] E. Systems, ESP32-C3 Technical Reference Manual, Shanghai: Espressif Systems, 2023.

[7] WatElectronics, »watelectronics,«, S interneta https://www.watelectronics.com/sound-sensor/ , rujan 2023.

[8] Paul Osborne i ostali, »Rust embedded,« S interneta https://docs.rust-embedded.org/book/ , kolovoz 2023.

[9] Shahzad i ostali, »Challenges in Transition of Software Defined Radio Implementation from Bare Metal to Real-Time Operating System,« u *2021 International Conference on Robotics and Automation in Industry (ICRAI)*, 2021.

[10] D. Lavry, »The optimal sample rate for quality audio,« *Lavry Engineering Inc,* 2012.

[11] Radford i ostali, »Robust speech recognition via large-scale weak supervision,« u *International Conference on Machine Learning*, 2023.

[12] Touvron i ostali, »Llama 2: Open foundation and fine-tuned chat models,« *arXiv preprint arXiv:2307.09288,* 2023.

[13] Austin i ostali, *Program Synthesis with Large Language Models,* arXiv, 2021.

[14] Clark i ostali, u *Proceedings of the 2019 Conference of the North*, 2019.

[15] Kwiatkowski i ostali, »Natural Questions: A Benchmark for Question Answering Research,« *Transactions of the Association for Computational Linguistics,* svez. 7, p. 453–466, studeni 2019.

[16] Cobbe i ostali, *Training Verifiers to Solve Math Word Problems,* arXiv, 2021.

[17] Touvron i ostali, *Llama 2: Open Foundation and Fine-Tuned Chat Models,* arXiv, 2023.

[18] Bisk i ostali, *PIQA: Reasoning about Physical Commonsense in Natural Language,* arXiv, 2019.

[19] Qin i ostali, »Understanding memory and thread safety practices and issues in real-world Rust programs,« u *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020.

[20] P. Schmid, »Llama 2 is here,« Hugging Face, s interneta https://huggingface.co/blog/llama2 , kolovoz 2023.

Pojmovnik

ADC – Analog to digital converter

Bare metal - softver koji je dizajniran za izravnu interakciju s temeljnim hardverom.

BLE – Bluetooth low energy

GATT - Generic Attribute Profile

LLM – Large language model

no_std - softver koji je dizajniran za izravnu interakciju s temeljnim hardverom

NVIC - Nested Vectored Interrupt Controller

NVM- Non-volatile memory

OO – Object oriented

RAII - Resource Acquisition is Initialisation,

RTC – real time clock

RTOS – Real time operating system

WiFi - Wireless fidelity

Sažetak

Ovaj rad istražuje simbiozu ugradbenih sustava i napredne umjetne inteligencije pomoću čipa ESP32-C3 za stvaranje inovativnog virtualnog pomoćnika. Koristi Bare Metal Rust za učinkovitu kontrolu hardvera i uključuje Whisper i LLAMA 2 AI za visokokvalitetnu konverziju govora u tekst i razumijevanje prirodnog jezika.

Bežične značajke i značajke niske potrošnje energije ESP32-C3 čine ga idealnim. Bare Metal Rust optimizira performanse, memoriju i vrijeme odziva, što je ključno u postavkama s ograničenim resursima. Studija uspoređuje Rust s etabliranim jezicima, ocjenjujući njihove značajke i sigurnost memorije.

Komunikacija se oslanja na GATT za razmjenu podataka, poboljšavajući prepoznavanje govora putem Whispera. LLAMA 2 dodaje svijest o kontekstu, poboljšavajući prepoznavanje namjere korisnika i relevantnost odgovora.

Dvosmjerni protok podataka poboljšava interakciju. Korisnički unos neprimjetno dolazi do poslužitelja za informirane odgovore. Dokument detaljno opisuje implementaciju, pokrivajući hardver, Rust programiranje, postavljanje GATT-a, integraciju Whispera, uključivanje LLAMA 2 i uspostavljanje dvosmjerne komunikacije.

Procjena učinka naglašava učinkovitost, osjetljivost, preciznost i korištenje resursa. Koristeći ESP32-C3 i Rust, ovaj rad prikazuje integrirani pristup i transformativni potencijal kombiniranja hardverske ekspertize s umjetnom inteligencijom.

***Ključne riječi* —Whisper, Rust, Ugradbeni sustavi, LLAMA2, ESP32, Bluetooth**

Abstract

This work explores the fusion of embedded systems and advanced AI using the ESP32-C3 chip to create an innovative virtual assistant. It employs Bare Metal Rust for efficient hardware control and incorporates Whisper and LLAMA 2 AI engines for high-quality speech-to-text conversion and natural language understanding.

The ESP32-C3's wireless and low-power features make it ideal. Bare Metal Rust optimizes performance, memory, and response time, crucial in resource-constrained setups. The study compares Rust with established languages, evaluating their features and memory security.

Communication relies on GATT for data exchange, enhancing speech recognition via Whisper. LLAMA 2 adds context-awareness, improving user intent recognition and response relevance.

The two-way data flow enhances interaction. User input seamlessly reaches the server for informed responses. The paper details implementation, covering hardware, Rust programming, GATT setup, Whisper integration, LLAMA 2 inclusion, and establishing bidirectional communication.

Performance evaluation underscores efficiency, sensitivity, precision, and resource use. Using ESP32-C3 and Rust, this work showcases an integrated approach and the transformative potential of combining hardware expertise with AI.

***Keywords* —Whisper, Rust, Embedded systems, LLAMA2, ESP32, Bluetooth**