SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1

Bežični prijenos audio signala putem BLE sučelja razvojnog sustava STM32WB5MM-DK

Jelena Gavran

Zagreb, srpanj 2022.

SADRŽAJ

POPIS SLIKA

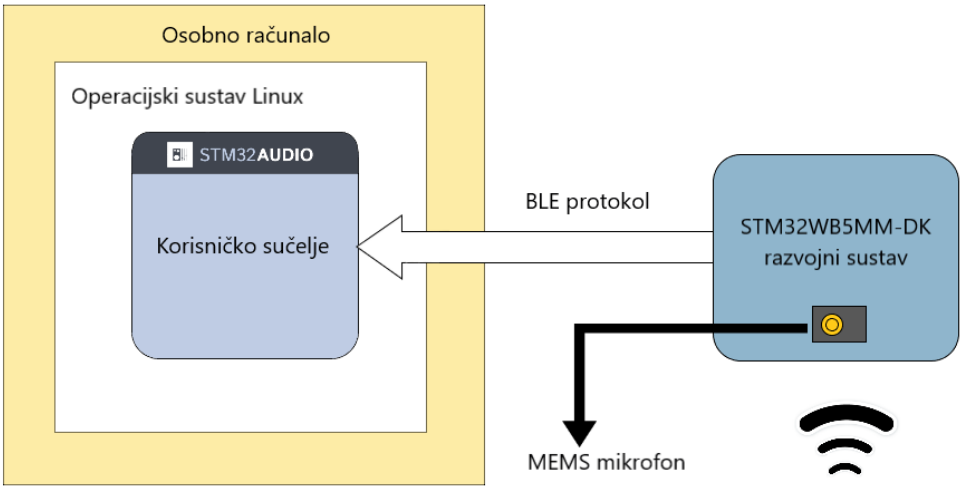
POPIS ISJEČAKA KODA

POPIS TABLICA

1. Uvod

Ovaj završni rad dio je doktorskog rada koji se bavi analizom hrkanja. Za provođenje istraživanja potrebno je sakupiti bazu zvučnih zapisa nad kojima će se provoditi obrada. Budući da se istraživanje i snimanje provode u bolnici, gdje postoje ograničenja nad uređajima koji nadziru i snimaju pacijente, potrebno je razviti neinvazivno softversko i hardversko rješenje koje će snimati i slati podatke na udaljenu lokaciju. Kao uređaj za snimanje zvuka odabran je mikrokontroler STM32WB5M koji će snimati hrkanje i slati snimljeni zvučni zapis Bluetoothom na računalo.

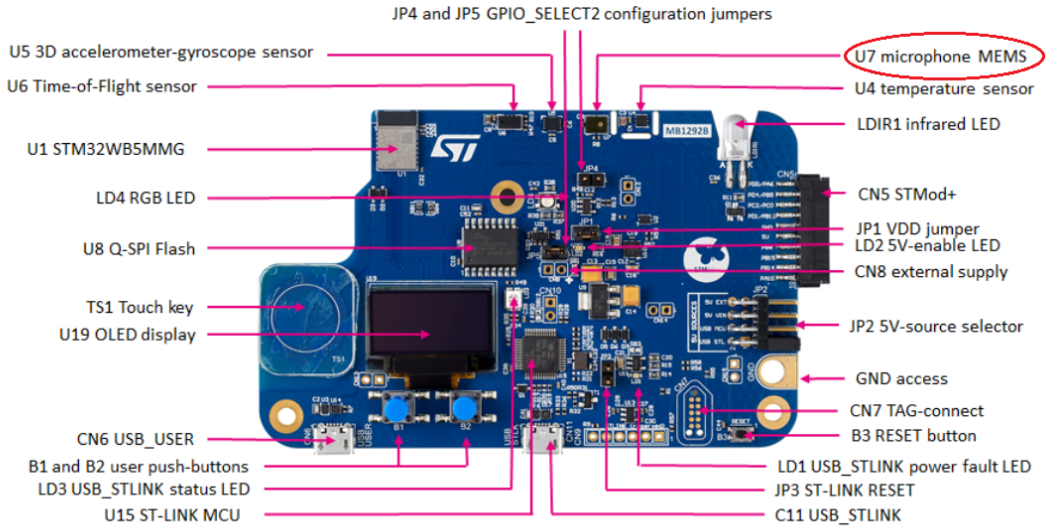
U okviru ovog završnog rada razvijena je programska potpora za mikrokontroler STM32WB5M te je uspostavljeno BLE komunikacijsko sučelje između razvojnog sustava i osobnog računala s operacijskim sustavom Linux. Sučeljem se prenosi zvučni signal sniman MEMS mikrofonom s mikrokontrolera na računalo. Također je i razvijeno korisničko sučelje za pokretanje komunikacije, prijem i pohranu signala. Blok shema sustava prikazana je na Slici 1.1.



Slika 1.1: Blok shema sustava

2. Razvojni sustav STM32WB5MM-DK

Razvojni sustav temelji se na modulu STM32WB5MMG tvrtke *STMicroelectronics*, koji je dio linije STM32WBx5 razvojnih sustava. Kao i svi mikrokontroleri iz te skupine, modul sadrži 32-bitni aplikacijski procesor ARM Cortex-M4 koji radi na frekvenciji do 64 MHz te mrežni procesor Cortex-M0+ s frekvencijom rada do 32MHz. Modul sadrži 1 MB memorije tipa *Flash* i 256 KB memorije tipa SRAM (*Static random-access memory*). Budući da modul ima funkciju RF primopredajnika, podržava protokole Bluetooth, Zigbee, Thread i konkurentne bežične standarde. Sustav također ima 0.96-inčni 128x64 zaslon, RGB LED lampice te senzore za temperaturu, dodir, *Time-of-Flight* senzor i žiroskop. Od ostale periferije najznačajniji je digitalni MEMS mikrofon. STM32WB5MMG modul je višeprotokolni, bežični uređaj niske potrošnje energije (engl. *ultra-low-power*) primarno namijenjen razvoju aplikacija koje koriste audio, USB ili *Bluetooth Low Energy* (BLE) protokol.



Slika 2.1: Konfiguracija STM32WB5MM-DK razvojnog sustava

2.1. BLE protokol

Bluetooth protokol korišten je za povezivanje razvojnog sustava s matičnim računalom i za prijenos audio signala s mikrokontrolera. BLE je vrsta bežične komunikacije namijenjena komunikaciji kratkog dometa s niskom potrošnjom energije. Razvijen je kako bi se postigao standard vrlo male snage koji radi s baterijom veličine kovanice (engl. *coin-cell batteries*) nekoliko godina. Klasična Bluetooth tehnologija razvijena je kao bežični standard, što je omogućilo razvoj bežičnih i prenosivih uređaja, no ne podržava dug život baterije zbog brze i nepredvidive komunikacije te složenih postupaka povezivanja. BLE uređaji troše samo dio energije koju troše standardni Bluetooth proizvodi te omogućavaju malenim uređajima s malim baterijama bežično povezivanje s uređajima koji koriste klasični Bluetooth.

BLE radi u istom opsegu od 2,4 GHz kao i standardni Bluetooth, no koristi različite kanale od standardnog Bluetootha. Koristi 40 kanala od 2 MHz za prijenos podataka korištenjem modulacije Gaussova pomaka frekvencije (metoda koja se koristi za glatkije prijelaze između podatkovnih impulsa), zbog čega skokovi frekvencije proizvode manje smetnji u usporedbi sa standardnom Bluetooth komunikacijom.

Arhitektura BLE tehnologije naziva se još i BLE stog zbog slojevite strukture. Stog se sastoji od dvije glavne komponente:

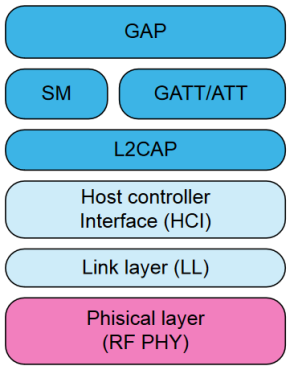
- upravljač (engl. *Controller*),
- domaćin (engl. *Host*).

Upravljač se sastoji od fizičkog sloja i sloja veze. Host uključuje protokol kontrole i prilagodbe logičke veze (*Logical Link Control and Adaptation Layer Protocol* - L2CAP), upravitelja sigurnosti (*Security Manager* - SM), protokol atributa (*Attribute Protocol* - ATT), generički profil atributa (*Generic Attribute Protocol* - GATT) i generički profil pristupa (*Generic Attribute Profile* - GAP). Sučelje između komponenti naziva se sučelje između domaćina i upravljača (*Host Controller Interface* - HCI).

2.1.1. Upravljač

Fizički sloj

Fizički sloj signal u radiofrekvencijskom području brzine od 1 Mbps koji prenosi informacije GFSK (*Gaussian Frequency Shift Keying*) frekvencijskom modulacijom. Radi u 2,4 GHz ISM pojasu bez licence na 2400-2483,5 MHz. BLE sustav koristi 40 RF kanala (0-39), s razmakom od 2 MHz. Postoje dvije vrste kanala:



Slika 2.2: Arhitektura BLE stoga

1. Kanali za oglašavanje koji koriste tri fiksna RF kanala (37, 38 i 39) za

   (a) Pakete kanala za oglašavanje

   (b) Pakete korištene za otkrivanje ili povezivanje

   (c) Pakete korištene za odašiljanje ili skeniranje

2. Podatkovni fizički kanal, koristi ostalih 37 RF kanala za dvosmjernu komunikaciju između povezanih uređaja.

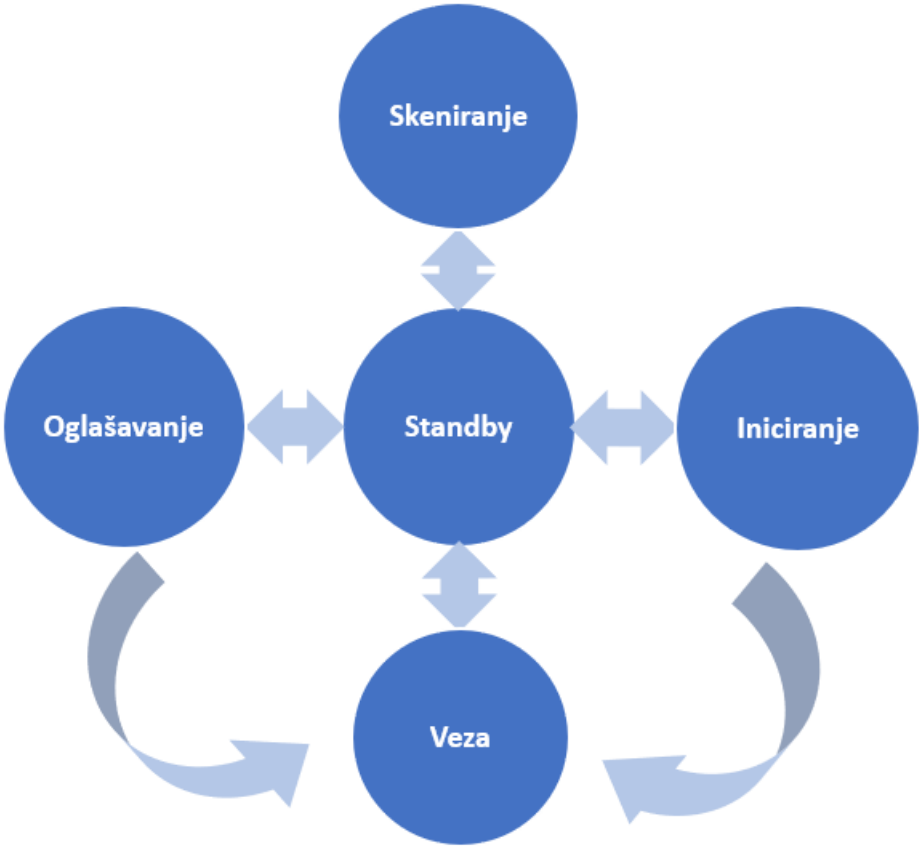
BLE je tehnologija adaptivnog skakanja frekvencije (*Adaptive frequency-hopping* - AFH) koja može koristiti samo podskup svih dostupnih frekvencija kako bi se izbjegle sve frekvencije koje koriste druge neprilagodljive tehnologije. To omogućuje prelazak s lošeg kanala na poznati dobar kanal korištenjem specifičnog algoritma za skakanje frekvencije, koji određuje sljedeći dobar kanal za korištenje.

Sloj veze

Sloj veze određuje kako dva uređaja mogu koristiti signale u radiofrekvencijskom području za međusoban prijenos informacija. Također definira automat s pet stanja:

- stanje pripravnosti (engl. *Standby*): uređaj ne šalje niti prima pakete,
- oglašavanje: uređaj šalje oglase putem kanala za oglašavanje,
- skeniranje: uređaj traži uređaje oglašivača,
- pokretanje (iniciranje): uređaj pokreće vezu s uređajem oglašivača,
- veza:
  - uređaj koji je inicirao komunikaciju je u ulozi *master*, komunicira sa *slave* uređajem i definira vrijeme prijenosa,

- uređaj oglašivača je u *slave* ulozi, komunicira s jednim *master* uređajem.



Slika 2.3: Automat sloja veze

HCI

Sloj sučelja glavnog kontrolera (HCI) pruža sredstvo komunikacije između domaćina i upravljača putem softverskog aplikacijskog programskog sučelja (*Application Programming Interface* - API) ili hardverskog sučelja kao što su: SPI, UART ili USB. Dolazi iz standardnih Bluetooth specifikacija, s novim dodatnim naredbama za funkcije specifične uz nisku potrošnju energije.

2.1.2. Domaćin

L2CAP

Protokol logičke veze i sloja prilagodbe (L2CAP) podržava multipleksiranje protokola više razine, operacije fragmentacije paketa i ponovnog sastavljanja, te prijenos informacija o kvaliteti usluga.

SM

BLE sloj veze podržava enkripciju i autentifikaciju korištenjem načina brojača s CBC-MAC algoritmom (kod za provjeru autentičnosti lančanih poruka) i 128-bitnu AES blok šifru (AES-CCM). Kada se enkripcija i autentifikacija koriste u vezi, 4-bajtna provjera integriteta poruke (MIC) dodaje se na jedinici podatkovnog protokola (PDU). Enkripcija se primjenjuje i na polja od PDU i MIC. Kada dva uređaja žele šifrirati komunikacije tijekom veze, upravitelj sigurnosti (SM) koristi postupak uparivanja. Ovaj postupak omogućuje provjeru autentičnosti dvaju uređaja razmjenom informacija o njihovu identitetu kako bi se stvorili sigurnosni ključevi koji se mogu koristiti kao osnova za pouzdani odnos ili jednu sigurnu vezu.

ATT

Protokol atributa (ATT) omogućuje uređaju da prikazuje podatke, koji se nazivaju atributima, drugom uređaju. Atributi su adresirani dijelovi informacija koji mogu sadržavati korisničke podatke ili meta-informacije o arhitekturi samih atributa. Uređaj koji prikazuje atribute naziva se poslužiteljem, a uređaj koji ih koristi naziva se klijentom. ATT definira skup metoda za otkrivanje, čitanje i pisanje atributa na drugi uređaj. Implementira *peer-to-peer* protokol između poslužitelja i klijenta tipičnom zahtjev-odgovor strukturom.

Atributi se sastoje od nekoliko parametara:

- Ručka (engl. *handle*): jedinstveni 16-bitni identifikator za svaki atribut na određenom poslužitelju; svaki atribut čini adresabilnim i zajamčeno se neće mijenjati
- Tip: 16-, 32- ili 128-bitni univerzalni jedinstveni identifikator (UUID) koji određuje vrstu podataka prisutnih u vrijednosti atributa
- Dopuštenja: meta-podaci koji opisuju dopuštenja za pristup ATT podacima, enkripciju i autorizaciju

– Vrijednost: stvarni sadržaj podataka atributa; dio atributa kojem klijent može pristupiti za čitanje i/ili pisanje

GATT

Generički atributni profil (GATT) definira okvir za korištenje ATT protokola, a koristi se za usluge, otkrivanje deskriptora, čitanje, pisanje i obavijesti. Podaci poslani putem BLE-a organizirani su ovim slojem. U GATT kontekstu, kada su dva uređaja povezana, postoje dvije uloge uređaja:

– GATT klijent: uređaj pristupa podacima na udaljenom GATT poslužitelju putem čitanja, pisanja, obavještavanja

– GATT poslužitelj: uređaj pohranjuje podatke lokalno i pruža metode pristupa podacima udaljenom GATT klijentu

Atributi GATT poslužitelja organizirani su kao niz usluga, od kojih svaka počinje atributom deklaracije usluge koji označava njen početak. Svaka usluga grupira jednu ili više karakteristika i svaka karakteristika može uključivati nula ili više deskriptora.

GAP

Bluetooth sustav definira osnovni profil koji implementiraju svi Bluetooth uređaji koji se naziva generički profil pristupa (GAP), koji definira osnovne zahtjeve Bluetooth uređaja. Programska potpora mikrokontrolera implementira komunikacijsku paradigmu temeljenu na povezivanju koja pruža trajnu vezu od točke do točke (engl. *point-to-point*) između dva uređaja kojom upravlja GAP sloj. Postoje četiri uloge GAP profila:

– emiter: šalje oglase,

– promatrač: prima oglase,

– periferija: uvijek u načinu oglašavanja i u *slave* ulozi,

– centar: nikada ne šalje oglase, uvijek u *master* ulozi.

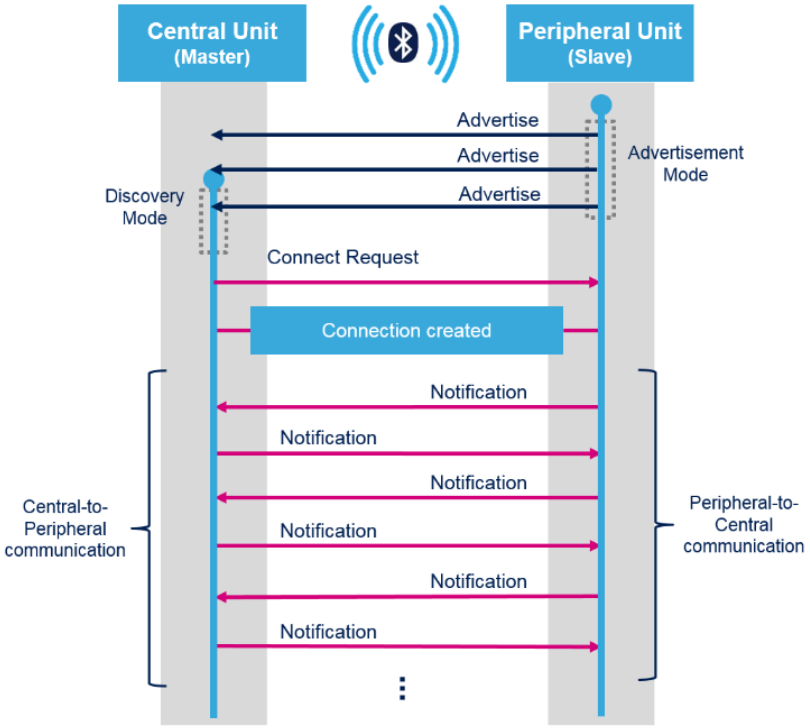
Razvijena programska potpora koristi dvije od navedenih uloga, a to su periferija i centar. Periferna uloga postavljena je mikrokontroleru jer se ta uloga postavlja uređajima koji podržavaju jednu vezu i smanjenu složenost. Ovi uređaji zahtijevaju samo upravljač koji podržava *slave* ulogu i koristi središnju frekvenciju upravljača za razmjenu podataka. S druge strane, centralna uloga pridružuje se uređaju koja podržava višestruke veze i pokretanje veza s perifernim uređajima. Ovi uređaji zahtijevaju

upravljač koji podržava glavnu ulogu sa složenijim funkcijama, što je u ovom slučaju računalo.

2.1.3. BLE komunikacija

Na Slici 2.4 prikazana je komunikacija između dva uređaja, odnosno računala i mikrokontrolera. Prema BLE specifikaciji, periferija ulazi u način oglašavanja (engl. *Advertisment Mode*) pri pokretanju i šalje pakete oglasa u relativno dugim intervalima. Središnja jedinica ulazi u način otkrivanja (engl. *discovery mode*) i šalje zahtjev za povezivanjem nakon primitka reklamnog paketa od *slave* uređaja. Nakon što je veza uspostavljena, obavijesti koje nose audio podatke periodično šalju od poslužitelja do klijenta, prema odabranom smjeru: periferija-centar, centar-periferija ili istovremeno na oba načina. Dok je mikrokontroler u načinu oglašavanja sve do uspostave veze, računalo je u načinu otkrivanja samo kraći vremenski period te prekida sa skeniranjem dostupnih uređaja nakon zadanog vremenskog intervala.

Mikrokontroler ima *slave* ulogu, dok računalo ima *master* ulogu.



Slika 2.4: Uspostava BLE veze

2.2. MEMS mikrofon

MEMS (*Micro-Electro-Mechanical Systems*) mikrofon je elektroakustični pretvornik koji sadrži MEMS senzor i aplikacijski specifičan integrirani sklop (ASIC). MEMS mikrofoni se uglavnom temelje na elektretskim kapsulama i obično imaju ugrađena pretpojačala i analogno-digitalne pretvornike. MEMS mikrofoni su također poznati kao mikrofonski čipovi ili silikonski mikrofoni.

Svi mikrofoni detektiraju akustične valove pomoću fleksibilne membrane, odnosno dijafragme. Membrana se pomiče pod pritiskom induciranih akustičnih valova. Danas većina MEMS mikrofona na tržištu koristi kapacitivnu tehnologiju za mjerenje zvuka. Kapacitivni MEMS mikrofoni mjere kapacitet između fleksibilne mikromembrane i fiksne stražnje ploče. Promjene tlaka zraka koje stvaraju zvučni valovi uzrokuju pomicanje membrane. Stražnja ploča je perforirana kako bi kroz nju mogao strujati zrak i dizajnirana je da ostane kruta budući da zrak prolazi kroz njezine perforacije. Kako se membrana pomiče, kapacitet se mijenja između pokretne membrane i fiksne stražnje ploče (budući da se udaljenost između njih mijenja), a ta se promjena može analizirati i zabilježiti.

Dizajn digitalnog MEMS mikrofona obično ima dodatni CMOS čip kao analogno-digitalni pretvornik. Ovi čipovi učinkovito preuzimaju pojačane analogne audio signale i pretvaraju ih u digitalne podatke. Također omogućuju lakšu integraciju digitalnih MEMS mikrofona s digitalnim proizvodima.

Najčešći format za digitalno kodiranje unutar MEMS mikrofona je modulacija trajanja impulsa (*pulse-duration modulation* - PDM). PDM omogućuje komunikaciju jednom podatkovnom linijom i satom. Prijamnici PDM signala, kao i sami MEMS mikrofoni, jeftini su i lako dostupni.

2.2.1. MEMS tehnologija

Mikroelektromehanički sustavi ili MEMS je tehnologija koja se definira kao sustav minijaturiziranih mehaničkih i elektromehaničkih elemenata (tj. uređaja i struktura) koji su izrađeni mikrotvorničkim tehnikama. Fizičke dimenzije MEMS uređaja mogu varirati od znatno ispod jednog mikrometra pa sve do nekoliko milimetara. Isto tako, tipovi MEMS uređaja mogu varirati od relativno jednostavnih struktura bez pokretnih elemenata, do iznimno složenih elektromehaničkih sustava s više pokretnih elemenata pod kontrolom integrirane mikroelektronike. Jedan glavni kriterij MEMS-a je da postoje barem neki elementi koji imaju neku vrstu mehaničke funkcionalnosti bez obzira

na to mogu li se ti elementi kretati ili ne.

Dok su funkcionalni elementi MEMS-a minijaturizirane strukture, senzori, aktuatori i mikroelektronika, najznačajniji elementi su mikrosenzori i mikroaktuatori. Oni su kategorizirani kao pretvornici energije iz jednog oblika u drugi - primjerice mikrosenzor, koji obično pretvara izmjereni mehanički signal u električni.

Stvarni potencijal MEMS-a ostvaruje se kada se minijaturizirani senzori, aktuatori i strukture spoje na silicijsku podlogu zajedno s integriranim krugovima, odnosno mikroelektronikom. Dok se elektronika proizvodi pomoću sekvenci procesa integriranog kruga (npr. CMOS, bipolarni ili BICMOS procesi), mikromehaničke komponente proizvode se korištenjem kompatibilnih *micromachining* procesa koji selektivno urezuju dijelove silikonske pločice ili dodaju nove strukturne slojeve kako bi formirali mehaničke i elektromehaničke uređaje. Još je kompleksnije ako se MEMS može spojiti ne samo s mikroelektronikom, već i s drugim tehnologijama kao što su fotonika, nanotehnologija itd. To se ponekad naziva heterogenom integracijom. Dok su složenije razine integracije budući trend MEMS tehnologije, sadašnja je tehnologija skromnija i obično uključuje jedan diskretni mikrosenzor, jedan diskretni mikroaktuator, jedan mikrosenzor integriran s elektronikom, mnoštvo identičnih mikrosenzora integriranih s elektronikom, jedan mikroaktuator integriran s elektronikom ili mnoštvo identičnih mikroaktuatora integriranih s elektronikom.

2.2.2. Rad MEMS mikrofona

MEMS mikrofon sadrži sljedeće komponente:

- MEMS pretvornik: sastoji se od membrane, perforirane ploče i kućišta,
- tiskana pločica (*printed circuit board* - PCB): uključuje ASIC polarizacijsku jedinicu, mikrofonsko pretpojačalo i AD pretvornik,
- mehanički poklopac.

Zvučni valovi ulaze u MEMS mikrofon kroz poklopac i prolaze kroz perforirano kućište i ploču prije nego dođu do membrane. Valovi uzrokuju različiti zvučni tlak na membrani i razliku u tlaku između prednje i stražnje strane membrane. Ova razlika tlaka uzrokuje njeno pomicanje sukladno zvučnim valovima. Međutim, mikrofonski signal se stvara samo ako postoji naboj između vodljive membrane i nepokretne ploče. Ploča i membrana skupa djeluju kao kondenzator koji je potrebno napuniti za ispravan rad. ASIC osigurava ovo punjenje.

Jednom napunjene, ploča i membrana mogu proizvesti napon. Budući da djeluju kao kondenzator, svaka promjena kapaciteta prouzročit će obrnuto proporcionalnu pro-

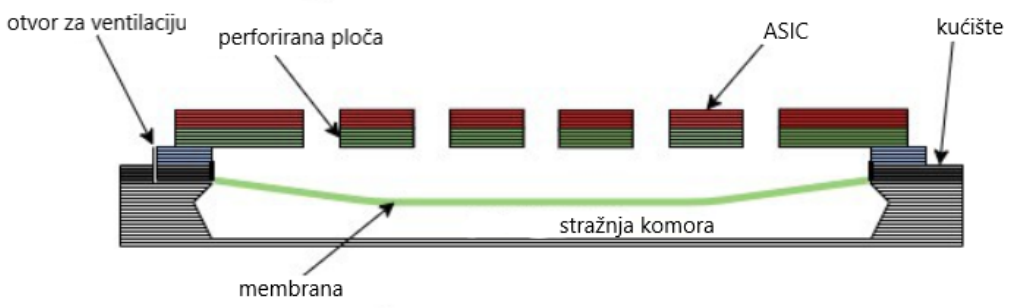
mjenu napona. Kapacitet je funkcija udaljenosti između ploče i membrane, stoga dok membrana oscilira, stvara se izmjenični napon odnosno mikrofonski signal. Ovaj napon treba pojačati da bi bio koristan kao audio signal, stoga odvojeni integrirani krug (poluvodička matrica), uključen u PCB, pojačava signal.

Ako se koristi analogni MEMS mikrofon, pojačani audio signal bi se u ovom obliku doveo na izlaz MEMS mikrofona. Međutim, u digitalnom MEMS mikrofonu postoji dodatni proces u kojem ADC pretvara analogni signal PDM metodom prije nego što emitira digitalni audio signal.

Kao što se vidi na Slici 2.4., stacionarna ploča je perforirana, što omogućava zraku prolaz do membrane. Na ovom je prikazu ASIC čip pričvršćen na ploču, no to nije slučaj kod svakog MEMS mikrofona.

Stražnja je komora u ovom primjeru zatvorena, što znači da je MEMS mikrofon tlačni mikrofon - membrana je otvorena samo za zvučne valove s jedne strane, što znači da prima zvuk iz svih smjerova. Stražnja komora također djeluje kao akustični rezonator i tako pomaže pri pravilnom podešavanju mikrofona.

Također je potreban i otvor za ventilaciju kako bi stražnja komora bila pod tlakom okoline.



Slika 2.5: Poprečni presjek MEMS mikrofona

3. Povezivanje razvojnog sustava i računala

Računalo i STM32 razvojni sustav dva su odvojena sustava koja moraju međusobno komunicirati i razmjenjivati podatke. Za ostvarenje njihove veze razvijena su dva programska rješenja:

1. programska potpora za mikrokontroler, koja će omogućiti pokretanje i snimanje zvučnog zapisa te njegov prijenos BLE sučeljem,
2. programska potpora za računalo, koja će ostvariti Bluetooth vezu između računala i mikrokontrolera te omogućiti prijem i pohranu primljenog audio signala.

3.1. Programska potpora za mikrokontroler

Dvije glavne funkcionalnosti koje mikrokontroler mora sadržavati su snimanje zvuka i njegov prijenos BLE komunikacijskim sučeljem. Za rad mikrokontrolera odabran je paket funkcija *FP-AUD-BVLINKWB1* iz alata *STM32Cube* koji je razvila tvrtka *ST-Microeletronics*. Ovaj *firmware* omogućava potpuni dvosmjerni prijenos zvuka koji se prenosi BLE sučeljem koristeći Opus algoritam za kompresiju. Aplikacija sadrži upravljačke programe i posrednički softver (engl. *middleware*) za BLE i digitalne MEMS mikrofone. Također uključuje kompletan Opus audio kodek kao *middleware* za izvođenje dvosmjernog i simultanog prijenosa zvuka između dva STM32WB mikrokontrolera.

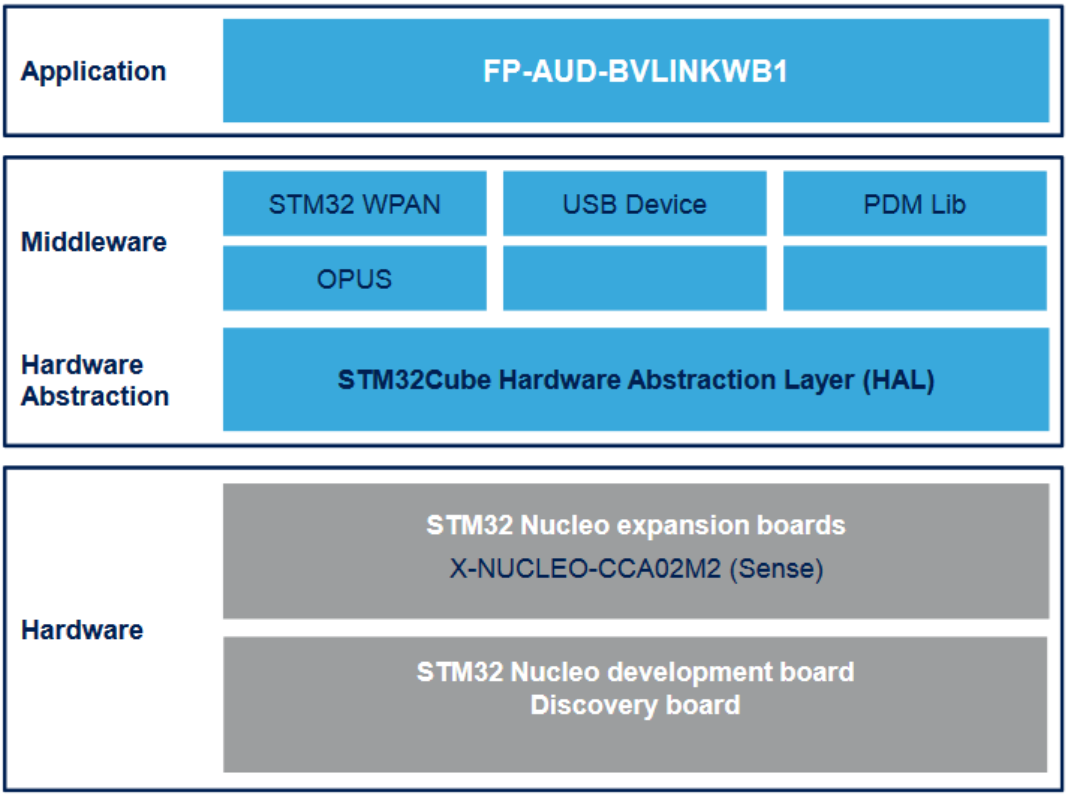
3.1.1. Arhitektura programske potpore za mikrokontroler

Softver se temelji na sloju apstrakcije hardvera STM32CubeHAL za STM32 mikrokontroler. Paket funkcija opremljen je skupom *middleware* komponenti za audio prijem, kompresiju i dekompresiju, prijenos podataka preko BLE sučelja i USB-a.

Aplikacija se sastoji od sljedećih slojeva softvera:

– STM32Cube HAL sloj: pruža jednostavan i modularan skup generičkih i proširenih API-ja za interakciju s gornjim slojevima aplikacije i bibliotekama. Ovi su API-ji izgrađeni na zajedničkoj arhitekturi te je moguće na njih dodavati slojeve (primjerice specifični *middleware*) bez obzira na sklopovske značajke mikrokontrolera.

– Sloj paketa podrške za razvojni sustav (BSP): skup API-ja koji pruža programsko sučelje za periferne uređaje specifične za razvojni sustav kao što su SPI, ADC, LED i korisnički gumbi.



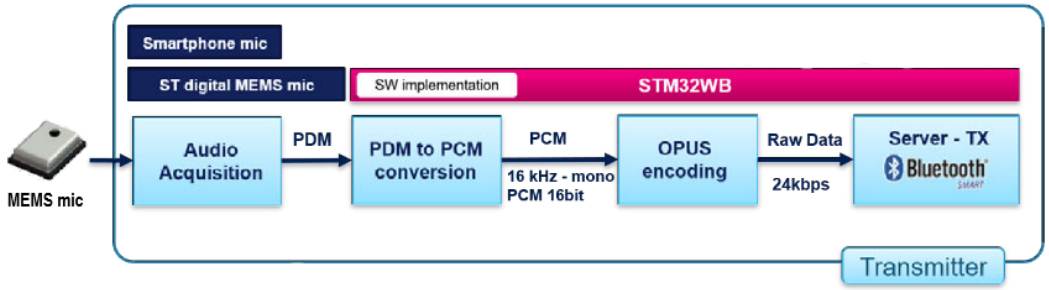
Slika 3.1: Arhitektura softvera FP-AUD-BVLINKWB1

Komponente za obradu funkcijskog paketa *FP-AUD-BVLINKWB1* dizajnirane su za stvaranje bežične audio veze između modula odašiljača (Tx) i prijamnika (Rx), gdje mikrokontroler služi kao odašiljač, a računalo kao prijamnik. Cijeli lanac obrade zvuka počinje prijemom MEMS digitalnim mikrofonom i kulminira reprodukcijom zvuka na računalu.

BLE je konfiguriran za slanje paketa s maksimalnom veličinom od 150 bajtova. Ovisno o aplikaciji, kodirani bajtovi mogu biti iznad ovog praga, stoga komprimirani

međuspremnik (engl. *buffer*) mora biti podijeljen u više BLE paketa. Štoviše, veličina kodiranog međuspremnika može promijeniti svaki audio okvir i prijamnik mora znati njegovu duljinu da bi ga obnovio; za ovaj opseg implementiran je jednostavan protokol BLE prijenosa.

Na strani odašiljača, zvuk se dobiva digitalnim MEMS mikrofonom kao 1-bitni PDM signal i pretvara se pomoću filtra za pretvorbu PDM-u-PCM u 16-bitni PCM (pulsno-kodna modulacija). Svaki put kad je audio okvir spreman, prenosi se u algoritam kompresije: veličina kodiranog međuspremnika koju vraća Opus koder može se značajno promijeniti u skladu s parametrima Opus kodera.



Slika 3.2: Lanac obrade odašiljača u *FP-AUD-BVLINKWB*

3.1.2. *Middleware* za prijenos zvuka

Budući da *streaming* zvuka nije dio predefiniranog skupa profila mikrokontrolera, *FP-AUD-BVLINKWB1* definira uslugu specifičnu za dobavljača pod nazivom *BlueVoice-OPUS* koja je posrednik između korisničkog zvuka i klijentskog uređaja. Ovisno o pokrenutoj aplikaciji, karakteristika se mijenja između audio ili glazbene karakteristike. Budući da *streaming* glazbe nije implementiran u ovoj aplikaciji, opisane su samo audio karakteristike.

Audio karakteristike sadrže sljedeće atribute:

- *Att1* - sadrži deklaraciju audio karakteristika,
  - UUID: standardni 16-bitni UUID za karakterističnu deklaraciju
  - Dozvole: R
  - Vrijednost: svojstva za ovu karakteristiku su "*notify only*", a UUID je za audio podatke
- *Att2* - sadrži audio podatke,
  - UUID: isti UUID u zadnjih 16 bajtova vrijednosti atributa definicije karakteristike

- Dozvole: nema
- Vrijednost: stvarni audio sadržaj

– *Att3* - sadrži konfiguraciju karakteristika klijenta.
  - UUID: standardni 16-bitni UUID za karakterističnu konfiguraciju klijenta
  - Dozvole: R/W
  - Vrijednost: prvi bit označava omogućenost obavijesti (0 ili 1), drugi bit omogućenost indikacija

Usluga *BlueVoiceOPUS* može implementirati odašiljač, prijamnik ili oboje u slučaju *full-duplex* komunikacije. Za ovu aplikaciju potrebno je implementirati odašiljač odnosno transmiter. Za prijenos zvuka, usluga i karakteristike moraju se kreirati pozivanjem inicijalizacijske funkcije `BVOPUS_STM_Init()`, što uključuje funkcije `BluevoiceOPUS_AddService()` i `BluevoiceOPUS_AddChar()`; UUID-ovi su definirani u datoteci `bvopus_service_stm.c`.

Karakteristike se mogu dodati već postojećoj usluzi pozivanjem funkcije `BluevoiceOPUS_AddChar()` i prosljeđivanjem oznake te određene usluge kao parametra. Ako funkcija vrati `BV_OPUS_SUCCESS`, BLE profil je ispravno kreiran.

Također, potrebno je konfigurirati Opus koder. U skladu sa traženim funkcijama, koder se može kreirati ispunjavanjem relevantne strukture: `OPUS_IF_ENC_ConfigTypeDef`.

Koder se može inicijalizirati pozivom pripadne funkcije, odnosno pozivom `BVOPUS_CodecEncInit(&EncConfigOpus)`. Ako je *BlueVoiceOPUS* profil ispravno konfiguriran, funkcija će vratiti `BV_OPUS_SUCCESS`. Ako vrati neuspjeh odnosno `BV_OPUS_INVALID_PARAM`, neki od parametara nisu ispravni. Ovisno o odabranim parametrima, inicijalizacijska funkcija dodjeljuje količinu memorije koju relevantni API vraća interno.

Pri inicijalizaciji podržani su sljedeći parametri:

– *application*: `OPUS_APPLICATION_VOIP, OPUS_APPLICATION_AUDIO , OPUS_APPLICATION_RESTRICTED_LOWDELAY`,

– *bitrate* [bps]: od 6000 do 510000,

– *channels*: od 1 do 255,

– *complexity*: od 0 do 10,

– *ms_frame* [ms]: 2.5, 5, 10, 20, 40, 60,

– *sample_freq* [Hz]: 8000, 12000, 16000, 24000, 48000.

```
EncConfigOpus.application = OPUS_APPLICATION_VOIP;
/* bps */
EncConfigOpus.bitrate = 24000;
/* 1 channel, mono*/
EncConfigOpus.channels = AUDIO_CHANNELS_IN;
EncConfigOpus.complexity = 0;
/* 20 ms */
EncConfigOpus.ms_frame = AUDIO_IN_MS;
/* 16000 Hz */
EncConfigOpus.sample_freq = AUDIO_IN_SAMPLING_FREQUENCY
 ;
```

Isječak koda 3.1: Parametri za Opus koder

Nakon postavljanja veze, modul koji je otkrio *BlueVoiceOPUS* profil drugog modula mora omogućiti kontrolnu obavijest pozivanjem API-ja `BluevoiceOPUS_EnableCtrl_Notif(void)`. Kontrolna se obavijest zatim koristi za zahtjev za pokretanje i zaustavljanje prijenosa.

Za početak audio prijenosa, modul odašiljača mora zatražiti od prijamnika da omogući njegovu audio obavijest pozivom `BluevoiceOPUS_SendEnableNotifReq()`. Ovaj API šalje obavijest putem kontrolne karakteristike koja sadrži dva bajta (`{BV_OPUS_CONTROL, BV_OPUS_ENABLE_NOTIF_REQ}`). Čim čvor primi zahtjev može omogućiti audio obavijest podnositelju zahtjeva pozivom funkcije `BluevoiceOPUS_EnableAudio_Notif(void)`. Ako je obavijest ispravno omogućena, modul može započeti prijenos zvuka.

BlueVoiceOPUS profil na ulaz prihvaća količinu PCM uzoraka jednaku veličini audio okvira postavljenoj tijekom Opus konfiguracije. Svaki put kada je audio okvir spreman, treba pozvati API `BluevoiceOPUS_SendAudioData()` i on automatski sažima, fragmentira i šalje pakete audio podataka.

Za svaku primljenu zvučnu obavijest potrebno je pozvati funkciju `BluevoiceOPUS_ParseData()` i provjeriti vraćeni status. U slučaju uspjeha, parametar `pcm_samples` pokazuje je li spreman kompletan audio okvir.

Prema zadanim postavkama, Opus koder je konfiguriran s promjenjivom brzinom prijenosa: svaki kodirani okvir ima duljinu prilagođenu brzini prijenosa postavljenoj tijekom faze inicijalizacije. Maksimalna veličina BLE paketa postavljena je na 150

bajtova, a broj BLE paketa može varirati među različitim audio okvirima ili ovisno o konfiguraciji Opusa.

Protokol prijenosa *BlueVoiceOPUS* modula pokazuje kada kodirani podaci počinju i završavaju tako da prijamnik može ponovno izgraditi komprimirani međuspremnik i dekodirati ga: jedan bajt se dodaje kao prvi bajt svakog BLE paketa, preostalih 19 bajtova ili više, ovisno o odabranom MTU, popunjeni su podacima kodiranim Opusom.

Sljedeće vrijednosti mogu biti bajt zaglavlja:

- `BV_OPUS_TP_START_PACKET = 0x00`
- `BV_OPUS_TP_START_END_PACKET = 0x20`
- `BV_OPUS_TP_MIDDLE_PACKET = 0x40`
- `BV_OPUS_TP_END_PACKET = 0x80`

Protokol prijenosa u potpunosti je obrađen u *BlueVoiceOPUS* usluzi.

3.1.3. Opus

Opus je otvoren i svestran audio kodek koji se može koristiti za različite vrste aplikacija kao što su *streaming* govora i glazbe ili komprimirana pohrana zvuka. Skalabilnost, od uskopojasnog govora niske brzine prijenosa pri 6 kbit/s do stereo glazbe pri 510 kbit/s niske složenosti, čini ga pogodnim za širok raspon interaktivnih aplikacija.

Sastoji se od dva sloja: jedan se temelji na linearnom predviđanju (LP), a drugi se temelji na modificiranoj diskretnoj kosinusnoj transformaciji (MDCT). Opus kombinira rezultate s gubitcima i bez gubitaka. Primjerice, u govornim aplikacijama, LP tehnike poput CELP-a (engl. *Code-excited linear prediction*) učinkovitije kodiraju niske frekvencije nego u tehnikama transformacijske domene kao što je MDCT.

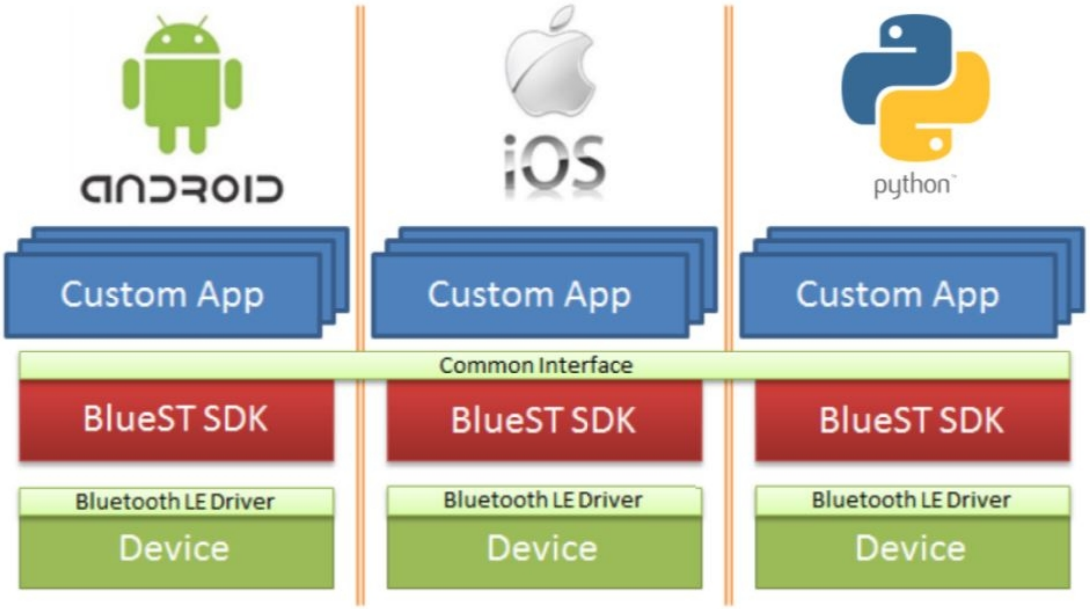
Opus kodek se sastoji od SILK i CELT tehnologija kodiranja. Prvi koristi model temeljen na predviđanju (LPC), dok je drugi u potpunosti modeliran na MDCT transformaciji. Ova svestranost omogućuje Opusu rad u tri načina rada (SILK, CELT ili hibridni način) i osigurava višestruke konfiguracije za različite aplikacije.

3.2. Programska potpora za računalo

Glavna zadaća aplikacije na računalu je Bluetoothom primiti audio signal, prikladno ga obraditi te izravno reproducirati i pohraniti. Za računalnu programsku potporu odabrana je *BlueST-SDK* biblioteka koji omogućuje jednostavan pristup podacima koje izvozi BLE uređaj s implementiranim *BlueST* protokolom. *BlueST* protokol lako je

proširiv za podršku korisnički definiranih podataka. On također prima podatke koji dolaze iz različitih senzora kao što su inercijski senzori, senzori okoliša, informacije o bateriji, te DC i motori. Protokol implementira i serijsku konzolu preko Bluetootha koja omogućuje funkcionalnosti standardnog izlaza i standardnog ulaza te definira konfiguracijski servis za kontrolu postavki povezanih ploča.

Korištenjem zajedničkog modela programiranja za podržane platforme, *BlueST-SDK* olakšava razvoj aplikacija na Android, iOS i Linux (s instaliranim Python) sustavima i uključuje primjere aplikacija koji demonstriraju korištenje paketa za razvoj programa (*Software development kit* - SDK). Paket za razvoj aplikacija na Linuxu *BlueST-SDK* biblioteke koristi *bluepy* Python biblioteku dostupnu na Linuxu za povezivanje s BLE uređajima.



Slika 3.3: Arhitektura aplikacije s *BlueST-SDK* modulom

Za razvoj aplikacije odabran je programski jezik Python na operacijskom sustavu Linux za lakše povezivanje s grafičkim korisničkim sučeljem, koje je također razvijeno u Pythonu.

3.2.1. Rad paketa za razvoj programa

BlueST-SDK biblioteka prikazuje samo uređaje s poljem specifičnim za dobavljača formatiranim kao što je prikazano u Tablici 3.1. Polje duljine mora biti veličine 7 ili 13 bajtova. ID uređaja je broj koji identificira tip uređaja te brojevi između 0x80 i 0xFF označavaju STM32 Nucleo razvojne sustave. Maska značajke polje bitova koje

daje informaciju o značajkama koje emitira uređaj. Polje je veličine 4 bajta i svaki bit označava jednu značajku. Svaki je bit postavljen u 0 ili 1, ovisno o tome je li značajka emitirana.

Tablica 3.1: Oblikovanje polja specifično za dobavljača *Blue-SDK* modula

Duljina	Naziv	Vrijednost
1	Duljina	0x07/0x0D
1	Tip polja	0xFF
1	Verzija protokola	0x01
1	ID uređaja	0xX
4	Maska značajke	0xXXXXXX
6	Adresa kontrole pristupa (MAC) uređaja	0xXXXXXXXXX

U Tablici 3.2 nalazi se popis ključnih značajki za ovu aplikaciju i njihove maske bitova. ADPCM označava prilagodljivu diferencijalnu impulsnu kodnu modulaciju, što je varijanta diferencijalne impulsne kodne modulacije (DPCM) koja mijenja veličinu koraka kvantizacije kako bi se omogućilo daljnje smanjenje potrebne širine opsega podataka za dani omjer signal-šum.

Tablica 3.2: Maske bitova i pripadne karakteristike u modulu *BlueST-SDK*

Duljina	Naziv
26	Razina mikrofona
27	ADPCM Audio
28	Smjer dolaska
29	*Switch*
30	ADPCM sinkronizacija

Karakteristike kojima upravlja SDK moraju imati navedeni UUID:
`XXXXXXXX-0001-11e1-ac36-0002a5d5c51`.

SDK skenira sve usluge, tražeći karakteristike koje odgovaraju uzorku. Prvi dio UUID-a ima bitove postavljene na 1 za svaku značajku koju karakteristika izvozi. U slučaju više značajki mapiranih u jednu karakteristiku, podaci moraju biti u istom redoslijedu kao maska bitova. Podaci se trebaju formatirati kao što je prikazano u Tablici 3.3.

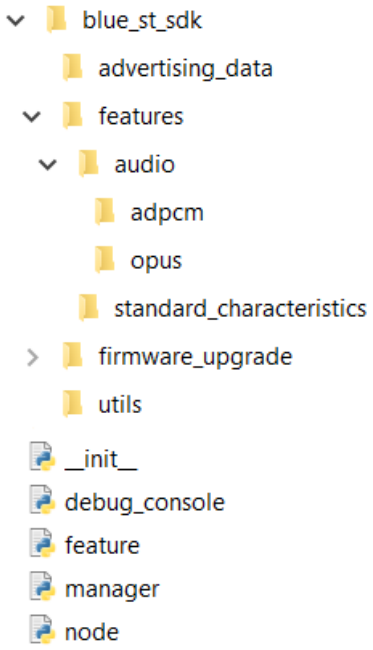
Prva dva bajta koriste se za slanje vremenske oznake. Ovo je osobito korisno za prepoznavanje bilo kakvog gubitka podataka. Budući da je maksimalna veličina BLE

paketa 20 bajtova, maksimalna veličina polja podataka značajke je 18 bajtova.

Tablica 3.3: Karakterističan format podataka u modulu *BlueST-SDK*

Duljina	Naziv
2	Vremenska oznaka
>1	Podatak prve značajke
>1	Podatak druge značajke
...	...

Modul *BlueST-SDK*, odnosno *blue_st_sdk*, koristi *bluepy* biblioteku za BLE povezivanje na Linuxu. Također, koristi modul *concurrent.futures* za pokretanje skupova dretvi (engl. *pools*) u pozadini, koje poslužuju povratne pozive promatrača. Međutim, zbog ograničenja *bluepy* biblioteke, pri korištenju *BlueST-SDK* modula nije moguće paralelno korištenje starih i otkrivanje novih uređaja. Isto tako, neočekivani prekid veze nije moguće odmah detektirati, nego se otkriva i obavještava putem promatrača pri izvođenju operacija čitanja i pisanja.



Slika 3.4: Struktura modula *blue_st_sdk*

Od važnijih direktorija unutar modula izdvajaju se *features* i *advertising_data*. Direktorij *features* sadrži klase koje implementiraju bazno sučelje *Feature* iz datoteke *feature.py* te svaka označava pojedinačnu značajku koju uređaj nudi, primjerice senzori za temperaturu i vlagu. Tu se također nalaze i klase koje implementiraju značajke

vezane za prijenos audio signala. Direktorij *advertising_data* sadrži datoteke u kojima se nalaze klase za obradu i pohranu podataka oglasa koje šalje uređaj.

Manager klasa

Manager je jedinstveni objekt koji pokreće i zaustavlja proces otkrivanja uređaja i pohranjuje dohvaćene čvorove. *Manager* obavještava novootkriveni čvor putem sučelja *ManagerListener*. Svaki povratni poziv (engl. *callback*) izvodi se asinkrono u pozadinskoj dretvi. Pojam jedinstvenog objekta označava da postoji samo jedna, globalna instanca klase kojoj se ne može pristupiti izravno, nego isključivo putem statičke metode `instance()`.

Također, objekt koji implementira sučelje *ManagerListener* promatrač je *Manager* klase, te izvršava određen skup naredbi pri svakoj promjeni *Manager* objekta. Budući da bi na promjenu stanja *ManagerListener* čekao u beskonačnoj petlji, odnosno ne bi obavljao nikakav koristan rad, *Manager* objekt pri svakoj vlastitoj promjeni obavještava sve promatrače koji promatraju njegove promjene, odnosno poziva njihove `update()` metode. Time se izbjegava beskonačna petlja u promatračima te izvođenje na zahtjev naredbi vezanih uz promjenu *Manager* objekta.

Node klasa

Node klasa predstavlja udaljeni uređaj. Prihvaća značajke koje čvor odnosno uređaj emitira te omogućuje čitanje poslanih podataka, kao i njihovo slanje na uređaj. Čvor emitira sve značajke čiji je odgovarajući bit postavljen na 1 unutar poruke oglasa. Nakon što se uređaj poveže, moguće je skenirati i omogućiti dostupne karakteristike te razmjenjivati podatke vezane uz te karakteristike. Isto tako, svi promatrači zainteresirani za promjene čvora registriraju se putem *NodeListener* sučelja.

Čvor može biti u jednom od sljedećih stanja:

- *init*: početni status,
- *idle*: čvor čeka vezu i šalje oglasne poruke,
- spajanje: uspostavlja se veza sa čvorom te čvor otkriva karakteristike i usluge uređaja,
- spojen: veza sa čvorom je uspješno uspostavljena,
- odspajanje: prekidanje veze sa čvorom koji se zatim vraća u *idle* status,
- izgubljen: uređaj je poslao oglašivačke podatke koji su nedohvatljivi,

– nedohvatljiv: veza sa čvorom je uspostavljena, no ne može se dohvatiti,

– mrtav: finalni status.

Feature klasa

Feature klasa predstavlja podatke koje čvor emitira, odnosno jednu značajku. Svaka značajka ima niz objekata polja koji opisuju izvezene podatke. Podaci se primaju iz BLE karakteristike i pohranjuju se u objekt *Sample* klase. *Feature* objekti također obavještavu korisnike o promjeni podataka putem sučelja *FeatureListener*.

3.2.2. Povezivanje s mikrokontrolerom

Najprije je potrebno pristupiti globalnoj instanci klase *Manager* za kontrolu i skeniranje Bluetooth uređaja. Također je potrebno kreirati promatrač *Manager* objekta, odnosno stvoriti instancu *ManagerListener* sučelja. Korištenje bilo kojeg sučelja promatrača nije moguće bez prethodnog kreiranja klase koje nasljeđuje sučelje.

```
manager = Manager.instance()
manager_listener = MyManagerListener()
manager.add_listener(manager_listener)
manager.discover(globals.SCANNING_TIME_s)
devices = manager.get_nodes()
```

Isječak koda 3.2: Pristup *Manager* instanci i skeniranje Bluetooth uređaja

Nakon definiranja objekata pokreće se vremenski ograničeno skeniranje dostupnih Bluetooth uređaja, koji se nakon otkrivanja pohrane u listu otkrivenih čvorova. Budući da je u ovoj aplikaciji potreban samo jedan uređaj, odabire se prvi iz liste uređaja s kojim se *Manager* uređaj spaja. Nakon uspješnog spajanja i dohvata dostupnih značajki započinje snimanje zvuka. Za snimanje se mogu koristiti Opus ili AD značajke, ovisno o tome koju značajku uređaj podržava. Kao što je već ranije opisano, STM32WB5MM-DK modul podržava Opus audio kodek.

Programska podrška za snimanje zvuka je *alsaaudio* modul. *Advanced Linux Sound Architecture*, odnosno ALSA, pruža audio i MIDI (*Musical Instrument Digital Interface*) funkcionalnost za Linux operacijski sustav. Biblioteka sadrži klase omotače (engl. *wrappers*) za pristup ALSA API-ju iz Pythona.

ALSA se sastoji od sljedećih komponenti:

– skup jezgrinih upravljačkih programa koji upravljaju hardveru za zvuk iz Linux jezgre,

– API na razini jezgre za upravljanje ALSA uređajima,

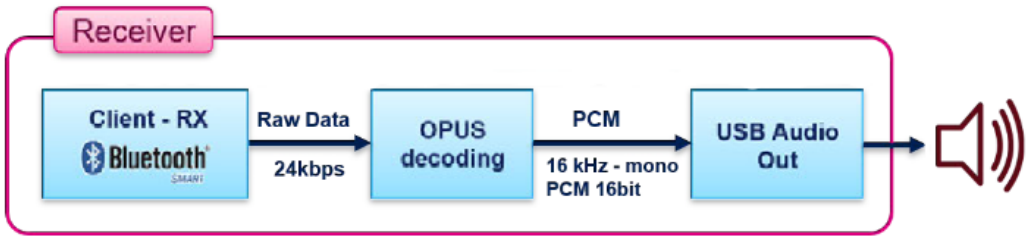
– C biblioteka za pojednostavljeni pristup hardveru za zvuk iz korisničkih aplikacija.

Putem ALSA biblioteke definira se novi audio tok koji koristi PCM način pretvaranja binarnog zapisa u zvuk kako bi bio kompatibilan s Opus kodekom. Podaci su u 16-bitni s predznakom u *little-endian* obliku, a frekvencija uzorkovanja je 16 kHz. Broj okvira koji će se upisati u pri svakoj iteraciji postavljen je na 160.

```
stream = alsaaudio.PCM(alsaaudio.PCM_PLAYBACK,
 alsaaudio.PCM_NORMAL,'default')
stream.setformat(alsaaudio.PCM_FORMAT_S16_LE)
stream.setchannels(globals.CHANNELS)
stream.setrate(globals.SAMPLING_FREQ_OPUS)
stream.setperiodsize(160)
```

Isječak koda 3.3: Postavljanje parametara za dekodiranje audio signala

Pri svakoj promjeni *Feature* objekta *FeatureListener* promatrač je obaviješten i *Feature* poziva *update()* metodu promatrača. Ta funkcija prima značajku i *Sample* objekt te, ovisno o načinu obrade zvuka (ADPCM ili Opus), prikladno se obrađuje primljeni uzorak iz dobivenog *Sample* objekta. Budući da je u ovoj aplikaciji korišten Opus kodek, potrebno je samo dohvatiti bajt podatka iz *Sample* objekta i pohraniti ga u datoteku i/ili ga poslati na audio tok koji preusmjerava zvuk na zvučnike računala.



Slika 3.5: Lanac obrade prijamnika u aplikaciji

Po završetku snimanja zvuka, svi promatrači otkazuju pretplatu na subjekte koje su promatrali te se zatvore audio tokovi. *Manager* objekt se odspaja od čvora i postavlja se u početno stanje.

4. Programska potpora za korisničko sučelje

Za bolje korisničko iskustvo kreirano je grafičko korisničko sučelje (engl. *Graphic User Interface* - GUI) koje se izvodi na korisničkom računalu. U GUI aplikaciji moguće je pokrenuti snimanje novog audiozapisa te grafički prikazati obradu signala postojećeg zvuka na računalu.

4.1. Razvojni alat PyQt

Aplikacija je izrađena korištenjem razvojnog alata PyQt temeljenog na programskom jeziku Python i pripadnih biblioteka za razvoj grafičkih korisničkih sučelja. PyQt je priključak (engl. *plug-in*) za Python - mostna biblioteka između Pythona i razvojnog alata Qt, koji podržava programski jezik C++. Korištena je inačica *PyQt5*, koja je kompatibilna s Python 3 verzijom.

Osnova *Qt* aplikacija je objektni model koji, koristeći sustav *Meta Object* i klasu *QObject*, proširuje funkcionalnost standardnog programskog jezika C++ i time omogućuje razvoj grafičkih korisničkih sučelja. *PyQt* enkapsulira funkcionalnosti *Qt* radnog okvira te ih prilagođava programskom jeziku Python, odnosno kombinira kompleksnost alata za razvoj grafičkog sučelja i jednostavnost programskog jezika.

Osnovna klasa je *QObject* koja pruža sljedeće funkcionalnosti:

- definiranje objekata jedinstvenim imenom,
- hijerarhijska organizacija objekata,
- komunikacija između objekata,
- upravljanje događajima.

Komunikacija između *Qt* objekata odvija se mehanizmom signala i priključaka (engl. *signals and slots*). Signal se emitira pri promjeni stanja objekta, primjerice

pritiskom na gumb unutar korisničkog sučelja. Pri emisiji signala poziva se funkcija priključka s kojom je taj signal povezan te se obrađuje događaj koji je izazvao emisiju.

Stvaranje i uređivanje grafičkih elemenata (engl. *widgets*) omogućeno je klasom *QWidget*. Grafički elementi organizirani su hijerarhijski, pri čemu je glavni prozor „roditelj“ ostalih elemenata.

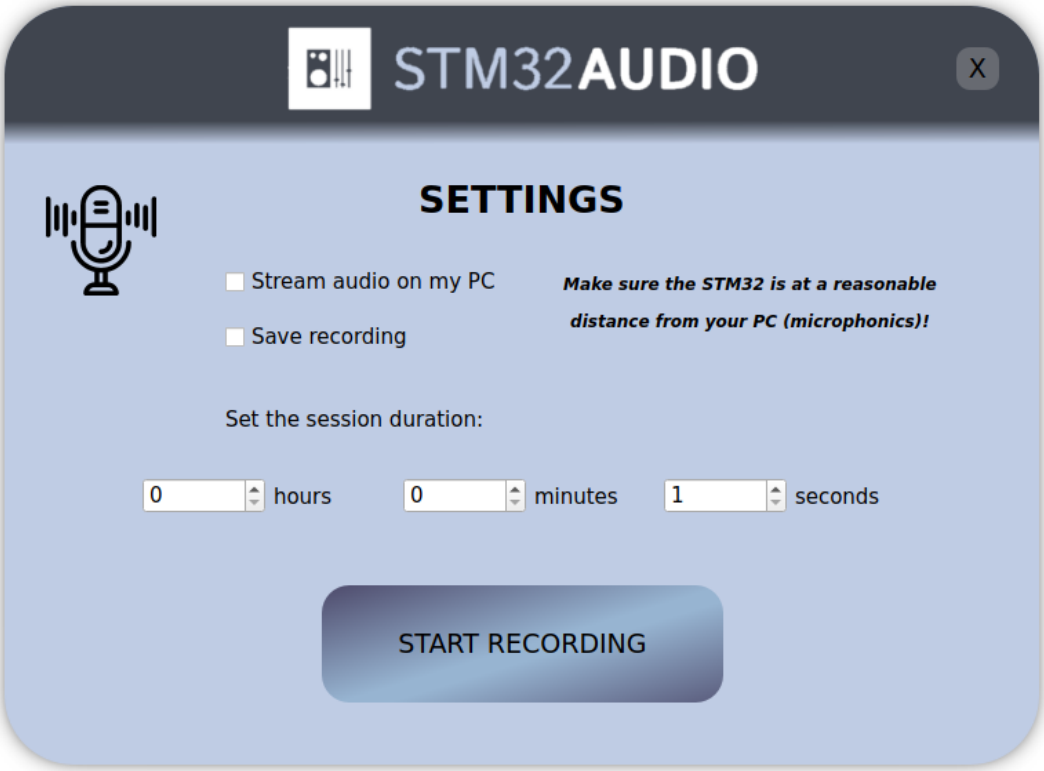
4.2. Implementacija korisničkog sučelja

Pri pokretanju aplikacije, u glavnom prozoru prikazuje se izbornik s gumbima *Record* i *Analyse Audio*. *Record* gumb vodi na sučelje za podešavanje parametara snimanja, dok *Analyse Audio* otvara izbornik za odabir audio datoteke nad kojom će se provesti obrada signala.

Slika 4.1: Uvodni izbornik

U sučelju za postavljanje parametara snimanja korisnik postavlja trajanje snimanja zvuka koje je ograničeno na minimalno 1 sekundu te maksimalno 24 sata. Korisnik također može odabrati opciju pohrane zvučnog zapisa na računalu, kao i trenutnu reprodukciju snimanog zvuka. Trenutna reprodukcija zvuka nije preporučljiva ako se

mikrokontroler i računalo nalaze u istoj prostoriji jer može doći do mikrofonije, koja se događa kada mikrofon prima zvuk iz uređaja za reprodukciju zvuka. Klikom na gumb *Start recording* otvara se novo sučelje u kojem se pokreće Bluetooth skeniranje i snimanje zvuka.



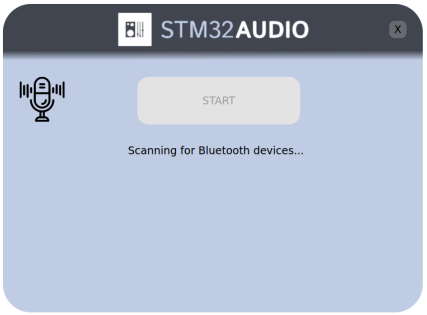
Slika 4.2: Sučelje za podešavanje parametara

Gumb *Start* pokreće Bluetooth skeniranje uređaja na računalu. Ako nije pronađen uređaj s kojim se računalo može upariti, sustav obavještava korisnika te omogućuje ponovno skeniranje uređaja.

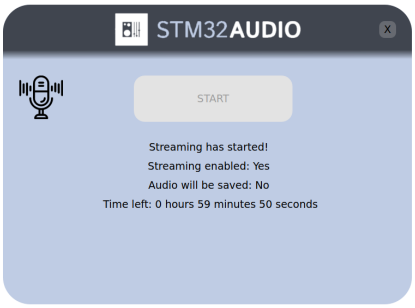
Nakon uparivanja s pronađenim uređajem, pokreće se snimanje zvuka na mikrokontroleru. Korisničko sučelje ispisuje na ekranu prethodno postavljene parametre snimanja i preostalo vrijeme do kraja snimanja. Po završetku primljeni se zvučni zapis pohranjuje obliku *RAW* datoteke, koja se zatim sprema u direktorij *audioDumps*. Svaki zvučni zapis u svom nazivu sadrži vremensku oznaku početka snimanja.

Snimanje je moguće pokrenuti ponovno pritiskom na gumb *Start*, ali s ranije definiranim parametrima.

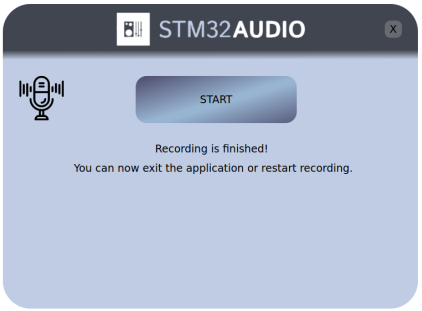
Slika 4.3: Prije pokretanja snimanja



Slika 4.4: Skeniranje uređaja



Slika 4.5: Snimanje zvuka



Slika 4.6: Završetak snimanja

Glavni prozor i ostali elementi korisničkog sučelja konfiguriraju se u klasi *Ui_Form*, odnosno u njezinoj funkciji `setupUi()`, koja kao parametar prima *Form* objekt. Taj objekt nasljeđuje dvije klase - *QWidget*, kao glavni prozor s grafičkim komponentama, i samu *Ui_Form* klasu, kako bi se glavni prozor mogao pomicati povlačenjem miša.

```
class Form(QtWidgets.QWidget, Ui_Form):
  def __init__(self, parent=None):
    super(Form, self).__init__(parent)
    self.setupUi(self)
    self.setMouseTracking(True)
```

Isječak koda 4.1: Definicija klase *Form* i njezin konstruktor

Također, definiraju se dva objekta za praćenje vremena:

1. *QTimer*: pokreće se u trenutku početka snimanja zvuka i trajanje mu je određeno na prethodnom sučelju. Služi za prikaz preostalog vremena i signalima je povezan s metodom `finished()` koja ispisuje poruku o završetku snimanja.

2. *QBasicTimer*: služi za osvježavanje grafičkog prikaza svake sekunde. Zaustavlja se istekom vremena postavljenog objektom *QTimer*, odnosno aktiviranjem me-

tode `finished()`. Povezan je signalima s funkcijom `timerEvent()` koja poziva metodu `update_gui()` za ponovno iscrtavanje sučelja.

Za pokretanje korisničkog sučelja potrebno je najprije napraviti instancu objekta *QApplication* koja upravlja glavnim postavkama i tokom kontrole GUI aplikacije. *QApplication* sadrži glavnu petlju događaja, gdje se obrađuju i šalju svi događaji iz prozorskog sustava i drugih izvora. Također upravlja inicijalizacijom, finalizacijom aplikacije i pruža upravljanje sesijom. Osim toga, *QApplication* obrađuje većinu postavki za cijeli sustav i aplikaciju. Za bilo koju GUI aplikaciju koja koristi *Qt*, postoji točno jedan *QApplication* objekt, bez obzira na broj prozora.

Metoda `exec_()` pokreće glavnu petlju događaja i čeka dok se ne pozove `exit` funkcija. Potrebno je pozvati ovu funkciju za početak rukovanja događajima. Glavna petlja događaja prima događaje iz prozorskog sustava i šalje ih widgetima aplikacije. Glavna petlja događaja prima događaje iz prozorskog sustava i šalje ih elementima aplikacije.

```
import sys
app = QtWidgets.QApplication(sys.argv)
w = Form()
w.show()
sys.exit(app.exec_())
```

Isječak koda 4.2: Naredbe za pokretanje korisničkog sučelja

4.3. Višedretvenost sučelja i programske potpore

PyQt aplikacije s grafičkim korisničkim sučeljem imaju glavnu dretvu koja pokreće glavnu petlju događaja i GUI. Pokrene li se dugotrajni zadatak u ovoj dretvi, GUI će se zamrznuti sve dok se zadatak ne izvrši. Za to vrijeme korisnik ne može komunicirati s aplikacijom, niti se prikaz sučelja može mijenjati tijekom izvođenja zadatka. Stoga je izvođenje dugotrajnih zadataka potrebno odvojiti od rada korisničkog sučelja.

U ovoj aplikaciji prikaz sučelja i programska potpora koja obavlja povezivanje Bluetoothom i snimanje zvuka izvršavaju se paralelno, stoga ih je potrebno izvoditi simultano u dvjema dretvama koje međusobno komuniciraju. Iako Python u svojoj biblioteci nudi module za rad s dretvama, u ovoj je aplikaciji korištena klasa *QtThread* koja se nalazi u okviru PyQt radi povezivanja rada dretvi sa signalima i događajima.

PyQt aplikacije imaju dvije vrste dretvi:

– Glavna dretva

– *Worker* dretve

Glavna dretva aplikacije uvijek postoji te se još naziva i GUI dretva. S druge strane, *worker* dretve ovise o potrebama aplikacije i može ih biti proizvoljno mnogo. One su sekundarne dretve koje se mogu koristiti za rasterećivanje glavnog programa i odvajanje dugotrajnih zadataka iz glavne dretve, čime se sprječava smrzavanje korisničkog sučelja.

Svaki *QThread* objekt upravlja jednom dretvom unutar programa i njihov rad započinje metodom `run`. Ta metoda pokreće petlju događaja unutar dretve pozivanjem funkcije `exec`. Važno je naglasiti da *QThread* objekt nije dretva sam po sebi, nego je omotač oko dretve operacijskog sustava. Prava dretva kreira se pozivom funkcije `QThread.start()`.

Worker klasa nasljeđuje *QObject* klasu i u nju je smještena programska potpora za povezivanje računala s mikrokontrolerom. Također su definirana tri signala koja komuniciraju s glavnom dretvom - jedan koji signalizira kraj izvođenja procesa, drugi za ažuriranje tekstualnog prikaza, i treći koji signalizira početak snimanja zvuka. Svaki od signala pozivom metode `emit()` emitira signal glavnoj petlji o vlastitoj promjeni.

```
class Worker(QObject):
  # Class for running BLE
  finished = pyqtSignal()
  textlabel = pyqtSignal(str)
  stream = pyqtSignal()

  def run(self):
    self.textlabel.emit("Scanning for devices...")
    # ... code for BLE connection ...
    return
```

Isječak koda 4.3: *Worker* klasa

Isječak koda 4.4 prikazuje princip povezivanja glavne, GUI dretve s *worker* dretvom koja povezuje rad mikrokontrolera i računala.

Za početak je potrebno stvoriti instance *QThread* i *Worker* klase. Nakon toga, funkcijom `moveToThread()` funkcionalnost *Worker* klase prebacuje se u *QThread* objekt koji će se izvršavati neovisno o GUI dretvi. Zatim je potrebno povezati signale dretve s funkcijama koje će se izvršiti pri emisiji signala. Naposljetku, izvršavanje

dretve započinje pokretanjem metode `start()` nad *QThread* objektom, što ujedno i emitira `started` signal koji je povezan s metodom `run()` objekta *Worker*. Time je započet rad te klase u odvojenoj dretvi bez ikakve ovisnosti o glavnoj dretvi s korisničkim sučeljem.

```
# Step 1: Create a QThread object
self.thread = QThread()
# Step 2: Create a worker object
self.worker = Worker()
# Step 3: Move worker to the thread
self.worker.moveToThread(self.thread)
# Step 4: Connect signals and slots
self.thread.started.connect(self.worker.run)
self.worker.finished.connect(self.thread.quit)
...
# Step 5: Start the thread
self.thread.start()
```

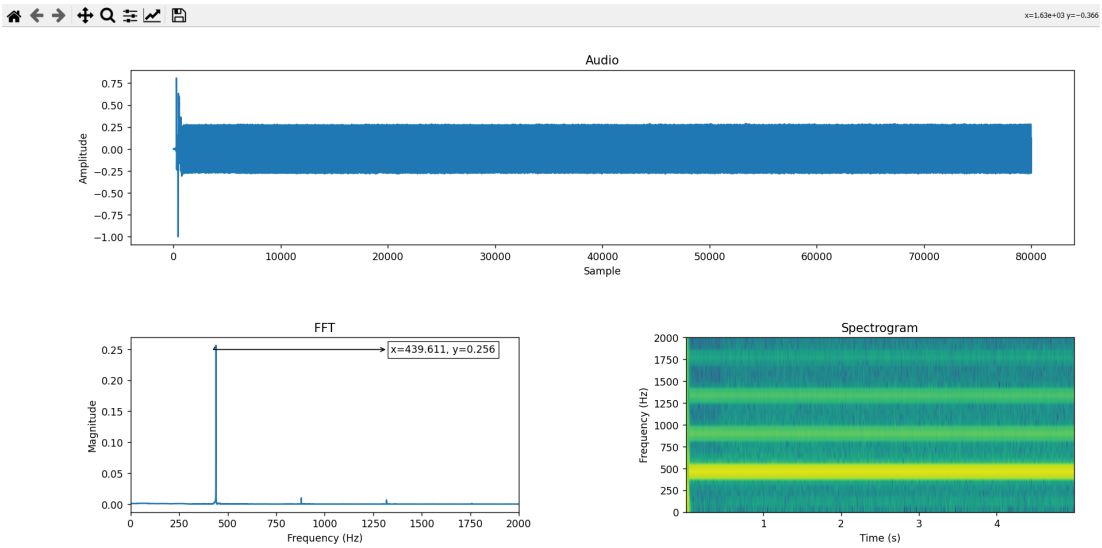
Isječak koda 4.4: Povezivanje glavne dretve s *worker* dretvom

5. Obrada audio signala

Nakon otvaranja audio datoteke započinje proces obrade zvučnog signala. Audio biblioteka *SoundFile* koristi se za učitavanje audio datoteke te, pozivom `sf.read(file_path)` dobivaju se matrica amplituda i frekvencija uzorkovanja. Dobivena matrica reprezentacija je audio signala u vremenskoj domeni, odnosno prikazuje glasnoću (amplitudu) zvuka dok se mijenja u vremenu. Amplituda jednaka nuli označava tišinu.

Za analizu odnosa amplitude i frekvencije signala potrebno transformirati signal u frekvencijsku domenu kako bi se prikazalo koje se frekvencije nalaze u signalu. Fourierovom transformacijom signal se dekomponira u odgovarajuće frekvencije. *Scipy* biblioteka sadrži ugrađenu funkciju za brzu Fourierovu transformaciju.

Dobivene matrice iscrtavaju se grafički pomoću biblioteke *matplotlib*. Ograničen je prikaz frekvencija na 2000 Hz...



Slika 5.1: Primjer prikaza analize audiozapisa

5.1. Odnos intenziteta zvuka i udaljenosti

5.2. Obrada zvučnih zapisa hrkanja

6. Zaključak

LITERATURA

Bežični prijenos audio signala putem BLE sučelja razvojnog sustava STM32WB5MM-DK

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: STM32WB5MM-DK, BLE, MEMS mikrofon, korisničko sučelje, obrada audio signala

Audio Signal Transmission Using BLE Interface of STM32WB5MM-DK Development Kit

Abstract

English abstract.

Keywords: STM32WB5MM-DK, BLE, MEMS microphone, user interface, audio signal processing