

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 438

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

Jelena Gavran

Zagreb, srpanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 438

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

Jelena Gavran

Zagreb, srpanj 2022.

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 438

Pristupnica: **Jelena Gavran (0036522308)**

Studij: Elektrotehnika i informacijska tehnologija i Računarstvo

Modul: Računarstvo

Mentor: prof. dr. sc. Hrvoje Džapo

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

Opis zadatka:

Proučiti specifikacije modula STM32W5MMG, razvojnog sustava STM32WB5MM-DK i značajke Bluetooth Low Energy (BLE) komunikacijskog sučelja. Razviti programsku potporu za mikrokontroler STM32WB55 na STM32W5MMG modulu razvojnog sustava STM32WB5MM-DK, upravljački program za MEMS mikrofoni i uspostaviti BLE komunikacijsko sučelje između razvojnog sustava i osobnog računala. Omogućiti prijenos audio signala s MEMS mikrofona na osobno računalo u stvarnom vremenu. Razviti korisničko sučelje za pokretanje komunikacije, prijem i pohranu signala. Istražiti ograničenja na frekvenciju uzorkovanja signala kako bi se osigurao dugoročni pouzdani prijenos signala. Istražiti osjetljivost mikrofona u ovisnosti o udaljenosti od izvora signala. Za detaljnije informacije obratiti se Karli Salamun, mag. ing.

Rok za predaju rada: 10. lipnja 2022.

SADRŽAJ

Popis slika	vi
Popis isječaka koda	vi
Popis tablica	viii
1. Uvod	1
2. Razvojni sustav STM32WB5MM-DK	2
2.1. BLE protokol	2
2.1.1. Upravljač	4
2.1.2. Domaćin	5
2.1.3. BLE komunikacija	8
2.2. MEMS mikrofoni	8
2.2.1. MEMS tehnologija	9
2.2.2. Način rada MEMS mikrofona	10
3. Povezivanje razvojnog sustava i računala	12
3.1. Programska potpora za mikrokontroler	12
3.1.1. Arhitektura programske potpore za mikrokontroler	12
3.1.2. <i>Middleware</i> za prijenos zvuka	14
3.1.3. Opus	17
3.2. Programska potpora za računalo	17
3.2.1. Opis razvojnog okvira <i>BlueST-SDK</i>	18
3.2.2. Povezivanje s mikrokontrolerom	22
4. Programska potpora za korisničko sučelje	24
4.1. Razvojni alat PyQt	24
4.2. Implementacija korisničkog sučelja	25

4.3. Višedretvenost sučelja i programske potpore	28
5. Obrada audio signala	31
5.1. Odnos intenziteta zvuka i udaljenosti	32
5.2. Obrada zvučnih zapisa hrkanja	33
5.2.1. Anatomija hrkanja	33
5.2.2. Analiza zvučnih zapisa	35
6. Zaključak	37
Literatura	38

POPIS SLIKA

1.1. Blok shema sustava	1
2.1. Konfiguracija razvojnog sustava STM32WB5MM-DK [18]	3
2.2. Arhitektura BLE stoga [17]	4
2.3. Automat sloja veze [17]	6
2.4. Uspostava BLE veze [19]	8
2.5. Poprečni presjek MEMS mikrofona [9]	11
3.1. Arhitektura softvera FP-AUD-BVLINKWB1 [19]	13
3.2. Lanac obrade odašiljača u <i>FP-AUD-BVLINKWB</i> [19]	14
3.3. Arhitektura aplikacije s modulom <i>BlueST-SDK</i> [16]	18
3.4. Struktura modula <i>blue_st_sdk</i>	20
3.5. Lanac obrade prijamnika u aplikaciji [19]	23
4.1. Uvodni izbornik	25
4.2. Sučelje za podešavanje parametara	26
4.3. Prije pokretanja snimanja	27
4.4. Skeniranje uređaja	27
4.5. Snimanje zvuka	27
4.6. Završetak snimanja	27
5.1. Primjer prikaza analize zvučnog zapisa	32
5.2. Anatomija gornjih dišnih puteva [14]	34
5.3. Analiza bez <i>notch</i> filtra	36
5.4. Analiza s <i>notch</i> filtrom	36

POPIS ISJEČAKA KODA

3.1. Parametri za Opus koder	15
3.2. Pristup instanci <i>Manager</i> i skeniranje Bluetooth uređaja	22
3.3. Postavljanje parametara za dekodiranje audio signala	23
4.1. Definicija klase <i>Form</i> i njezin konstruktor	27
4.2. Naredbe za pokretanje korisničkog sučelja	28
4.3. Klasa <i>Worker</i>	29
4.4. Povezivanje glavne dretve s dretvom tipa <i>worker</i>	30
5.1. Primjena <i>notch</i> filtra i Fourierove transformacije	36

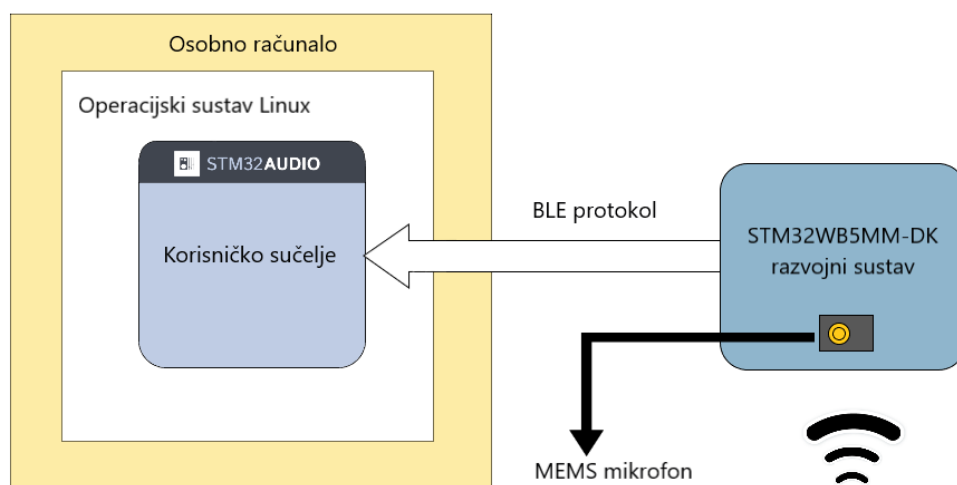
POPIS TABLICA

3.1.	Oblikovanje polja specifično za dobavljača modula <i>Blue-SDK</i> [16]	. .	19
3.2.	Maske bitova i pripadne karakteristike u modulu <i>BlueST-SDK</i> [16]	. .	19
3.3.	Karakterističan format podataka u modulu <i>BlueST-SDK</i> [16]	19

1. Uvod

Ovaj završni rad dio je istraživanja koji se bavi analizom spavanja. U okviru ovog rada, razvijen je sustav za prikupljanje zvučnih zapisa hrkanja koji će se koristiti pri analizi. Budući da se istraživanje i snimanje provode u kliničkom okruženju, 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 razvojni sustav STM32WB5MM-DK koji će snimati zvučne zapise hrkanja i slati ih putem Bluetooth sučelja 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, prijam i pohranu signala. Blok shema sustava prikazana je na Slici 1.1.



Slika 1.1: Blok shema sustava

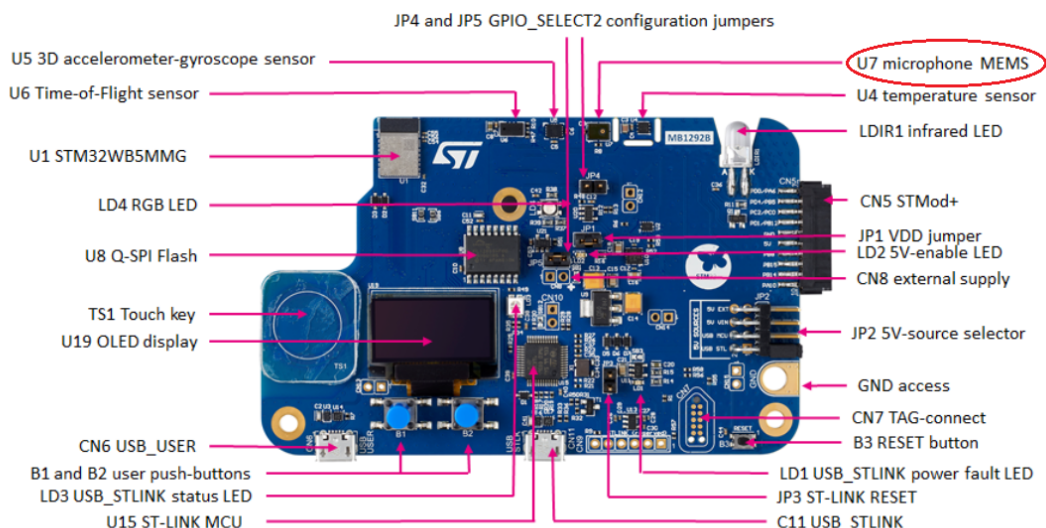
2. Razvojni sustav

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 (engl. *Static random-access memory*).[18] Budući da modul ima funkciju RF (engl. *radio frequency*) primopredajnika, podržava protokole Bluetooth, Zigbee, Thread i konkurentne bežične standarde. Sustav također ima 0.96-inčni 128x64 zaslon, RGB LED diode te senzore za temperaturu, dodir, *Time-of-Flight* senzor i žiroskop. Od ostale periferije najznačajniji je digitalni MEMS mikrofoni. Modul STM32WB5MMG 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.

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 [13].



Slika 2.1: Konfiguracija razvojnog sustava STM32WB5MM-DK [18]

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 glatke 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:

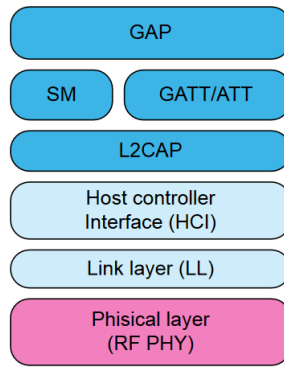
- upravljač (engl. *controller*),
- domaćin (engl. *host*).

Upravljač se sastoji od fizičkog sloja i sloja veze. Host uključuje protokol kontrole i prilagodbe logičke veze (engl. *Logical Link Control and Adaptation Layer Protocol* - L2CAP), upravitelja sigurnosti (engl. *Security Manager* - SM), protokol atributa (*Attribute Protocol* - ATT), generički profil atributa (engl. *Generic Attribute Protocol* - GATT) i generički profil pristupa (engl. *Generic Attribute Profile* - GAP). Sučelje između komponenti naziva se sučelje između domaćina i upravljača (engl. *Host Controller Interface* - HCI) [17].

Atributi su adresirani dijelovi informacija koji mogu sadržavati korisničke podatke ili meta-informacije o arhitekturi samih atributa. Uređaj koji prikazuje attribute naziva se poslužiteljem, a uređaj koji ih koristi naziva se klijentom.

Atributi se sastoje od nekoliko parametara:

- parametar *handle*: jedinstveni 16-bitni identifikator za svaki atribut na određe-



Slika 2.2: Arhitektura BLE stoga [17]

nom 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.

2.1.1. Upravljač

Fizički sloj

Fizički sloj signal u radiofrekvencijskom području brzine od 1 Mbps koji prenosi informacije GFSK (engl. *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:

1. Kanali za oglašavanje koji koriste tri fiksna RF kanala (37, 38 i 39) za sljedeće tipove paketa:
 - (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 (engl. *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 (engl. *advertisement*): uređaj šalje oglase putem kanala za oglašavanje,
- skeniranje (engl. *scaanning*): uređaj traži uređaje oglašivača,
- pokretanje (engl. *initiating*): uređaj pokreće vezu s uređajem oglašivača,
- veza (engl. *connection*):
 - uređaj koji je inicirao komunikaciju je u ulozi *master*, komunicira sa *slave* uređajem i definira vrijeme prijenosa,
 - uređaj oglašivača je u ulozi *slave*, komunicira s jednim *master* uređajem.

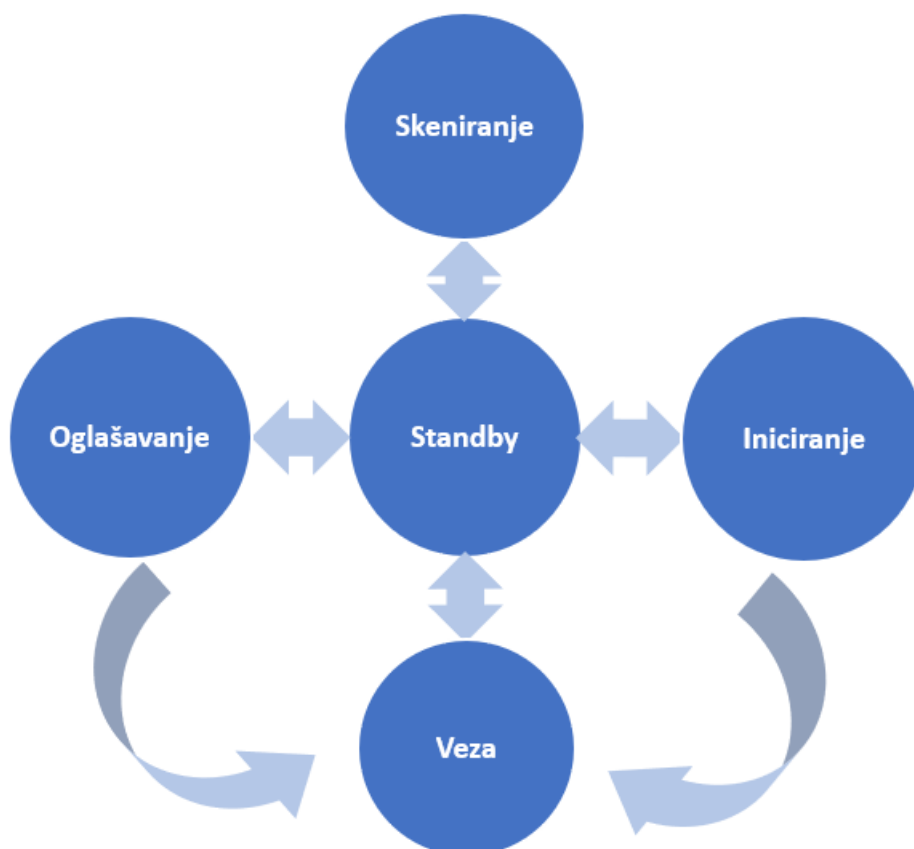
HCI

Sučelje između domaćina i upravljača (HCI) pruža sredstvo komunikacije putem softverskog aplikacijskog programskog sučelja (engl. *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 koje osiguravaju 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.



Slika 2.3: Automat sloja veze [17]

SM

BLE sloj veze (SM) 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 podatke tijekom veze, upravitelj sigurnosti (SM) koristi postupak uparivanja. Ovaj postupak omogućuje provjeru autentičnosti dvaju uređaja razmjenom informacija o njihovoj identitetu kako bi se stvorili sigurnosni ključevi koji se mogu koristiti kao osnova za pouzdani odnos ili jednu sigurnu vezu.

ATT

Protokol atributa (ATT) definira skup metoda za otkrivanje, čitanje i pisanje atributa na drugi uređaj. Implementira *peer-to-peer* protokol između poslužitelja i klijenta ti-

pičnom zahtjev-odgovor strukturom.

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. Ovaj profil naziva se 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:

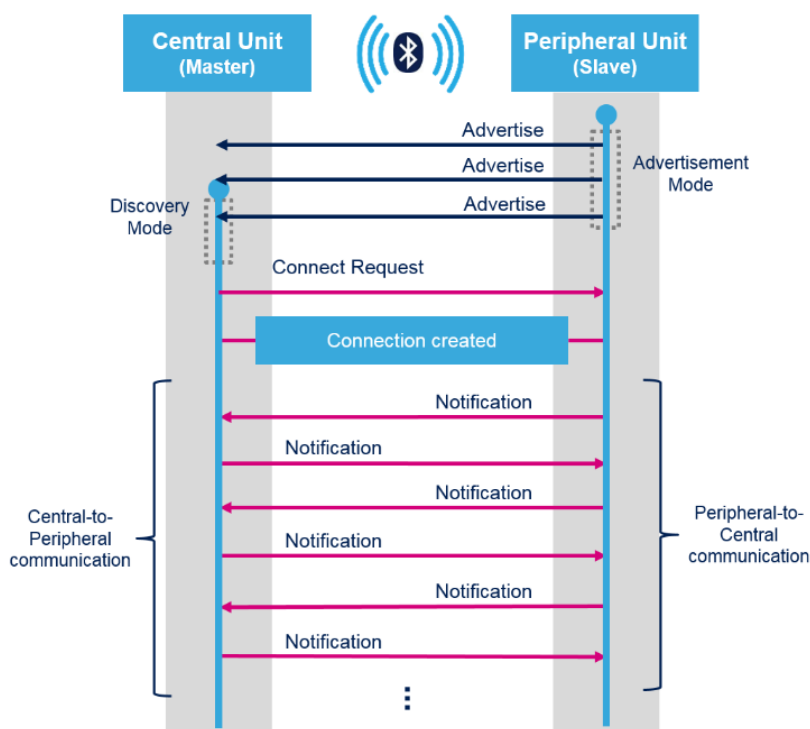
- emiter (engl. *broadcaster*): šalje oglase,
- promatrač (engl. *observer*): prima oglase,
- periferija (engl. *peripheral*): uvijek u načinu oglašavanja i u ulozi *slave*,
- centar (engl. *central*): nikada ne šalje oglase, uvijek u ulozi master.

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 ulogu *slave* 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. *advertisement 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 paketa oglasa od *slave* uređaja. Nakon što je veza uspostavljena, obavijesti koje nose audio podatke periodično se šalju od poslužitelja do klijenta u 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 ulogu *slave*, dok računalo ima ulogu *master*.



Slika 2.4: Uspostava BLE veze [19]

2.2. MEMS mikrofon

MEMS (engl. *Micro-Electro-Mechanical Systems*) mikrofoni su elektroakustični pretvornici koji sadrže 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 [11].

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 prikupljanje zvučnih signala. 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 (engl. *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 mikro-

senzor, 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. 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. Ovakve konfiguracije ponekad se nazivaju 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 [8].

2.2.2. Način rada MEMS mikrofona

MEMS mikrofonski sadrži sljedeće komponente:

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

Zvučni valovi ulaze u MEMS mikrofonski kroz poklopac i prolaze kroz perforirano kućište i ploču prije nego dođu do membrane. Valovi uzrokuju razliku u tlaku između prednje i stražnje strane membrane. Ova razlika tlaka uzrokuje njeno pomicanje skladno zvučnim valovima. Međutim, mikrofonski signal se stvara samo ako postoji naboj između vodljive membrane i stražnje ploče. Ploča i membrana zajedno djeluju kao kondenzator koji je potrebno napuniti za ispravan rad. Pripadajući integrirani sklop (ASIC) osigurava ovo punjenje.

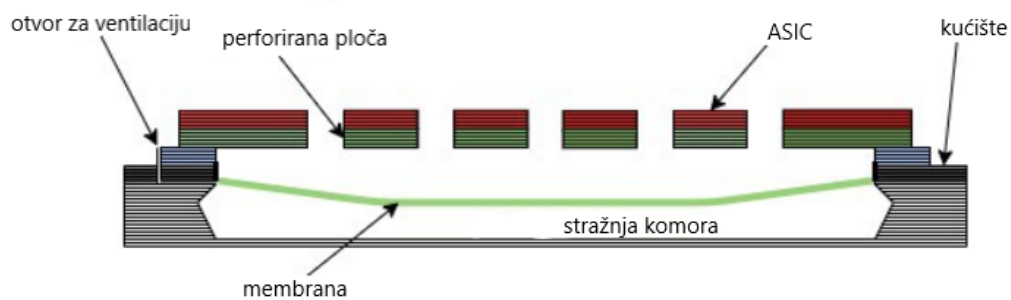
Jednom napunjene, ploča i membrana mogu proizvesti napon. Budući da djeluju kao kondenzator, svaka promjena kapaciteta prouzročit će obrnuto proporcionalnu promjenu napona. Kapacitet je funkcija udaljenosti između ploče i membrane, stoga dok membrana oscilira, stvara se izmjenični napon odnosno mikrofonski 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 mikrofoni, pojačani audio signal bi se u ovom obliku doveo na izlaz MEMS mikrofona. Međutim, u digitalnom MEMS mikrofoni postoji dodatni proces u kojem ADC pretvara analogni signal PDM metodom prije nego što emitira digitalni audio signal.

Kao što se vidi na Slici 2.5, 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 mikrofoni tlačni mikrofoni - 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 [9].



Slika 2.5: Poprečni presjek MEMS mikrofona [9]

3. Povezivanje razvojnog sustava i računala

Računalo i STM32WB5MM-DK 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-Microelectronics*. 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

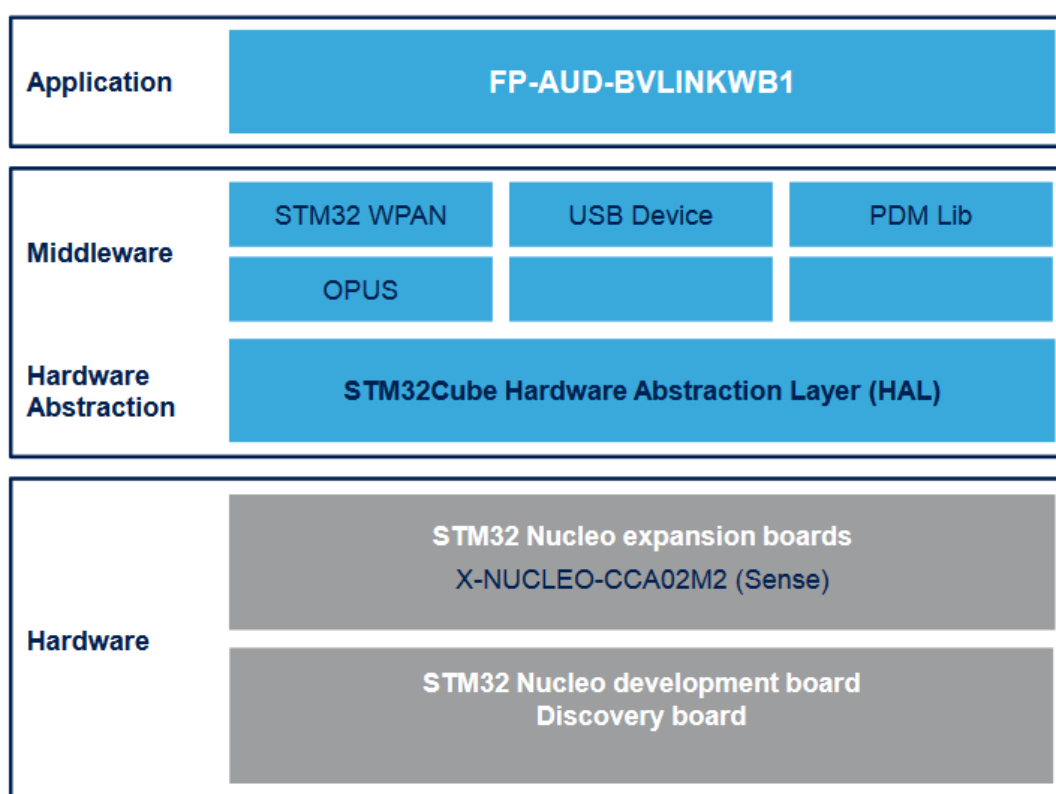
Programska potpora za mikrokontroler temelji se na sloju apstrakcije hardvera HAL za STM32 mikrokontroler. Paket funkcija opremljen je skupom *middleware* komponenti

za prijam audio signala, kompresiju i dekompresiju, prijenos podataka preko BLE sučelja i USB-a.

Aplikacija sadrži dva sloja softvera - STM32Cube HAL sloj i sloj paketa podrške za razvojni sustav (BSP).

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) je skup API-ja koji pruža programsko sučelje za periferne uređaje specifične za razvojni sustav kao što su SPI, ADC, LED i korisnički gumbi [19].

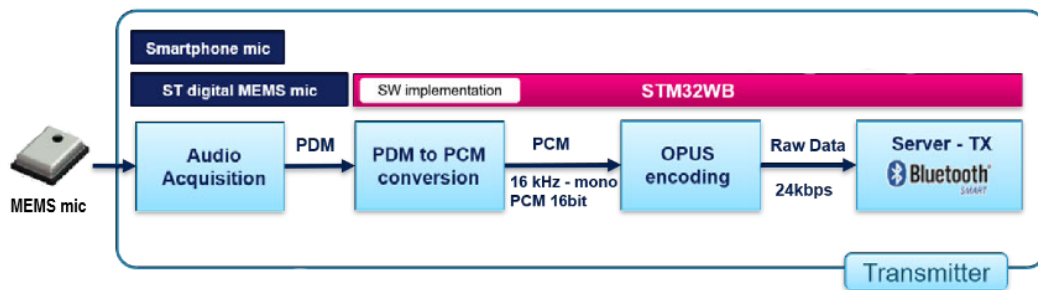


Slika 3.1: Arhitektura softvera FP-AUD-BVLINKWB1 [19]

Komponente za obradu funkcijskog paketa *FP-AUD-BVLINKWB1* dizajnirane su za stvaranje bežične audio veze između modula odašiljača (Tx) i prijarnika (Rx), gdje mikrokontroler služi kao odašiljač, a računalo kao prijarnik. Cijeli lanac obrade zvuka počinje snimanjem signala 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 u 16-bitni PCM signal (engl. *pulse-code modulation*). 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 koda.



Slika 3.2: Lanac obrade odašiljača u *FP-AUD-BVLINKWB* [19]

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 snimanog zvuka i klijentskog uređaja. Usluga *BlueVoiceOPUS* može implementirati odašiljač, prijamnik ili oboje u slučaju *full-duplex* komunikacije. Za ovu aplikaciju potrebno je implementirati odašiljač odnosno transmitter.

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 proslijeđivanjem 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 uz pomoć strukture

OPUS_IF_ENC_ConfigTypeDef.

Koder se može inicijalizirati pozivom pripadne funkcije, odnosno pozivom `BVOPUS_CodecEncInit(&EncConfigOpus)`. Ako je profil *BlueVoiceOPUS* 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 su podržani sljedeći parametri:

- *application* - vrsta aplikacije:
 - `OPUS_APPLICATION_VOIP`,
 - `OPUS_APPLICATION_AUDIO`,
 - `OPUS_APPLICATION_RESTRICTED_LOWDELAY`
- *bitrate* - brzina prijenosa [bps]: od 6000 do 510000,
- *channels* - broj kanala: od 1 do 255,
- *complexity* - složenost: od 0 do 10,
- *ms_frame* - trajanje okvira [ms]: 2.5, 5, 10, 20, 40, 60,
- *sample_freq* - frekvencija očitavanja [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 profil *BlueVoiceOPUS* drugog modula mora omogućiti kontrolnu obavijest pozivanjem funkcije

`BluevoiceOPUS_EnableCtrl_Notif()`. Kontrolna se obavijest zatim koristi za zahtjev za pokretanje i zaustavljanje prijenosa.

Za početak audio prijenosa, modul odašiljača mora zatražiti od prijatelja da omogućiti njegovu audio obavijest pozivom

`BluevoiceOPUS_SendEnableNotifReq()`. Ova funkcija šalje obavijest putem kontrolne karakteristike koja sadrži dva bajta (`{BV_OPUS_CONTROL, BV_OPUS_ENABLE_NOTIF}`). Čim čvor primi zahtjev, može omogućiti audio obavijest podnositelju zahtjeva pozivom funkcije

`BluevoiceOPUS_EnableAudio_Notif()`. Ako je obavijest ispravno omogućena, modul može započeti prijenos zvuka.

Profil *BlueVoiceOPUS* 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 modula *BlueVoiceOPUS* pokazuje kada kodirani podaci počinju i završavaju tako da prijatelj 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.

Bajt zaglavlja može uključivati jednu od sljedećih vrijednosti:

- `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 usluzi *BlueVoiceOPUS*.

3.1.3. Opus

Opus je programski okvir otvorenog koda i audio kodek koji se može koristiti za različite vrste aplikacija kao što su *streaming* govora i glazbe ili komprimiranje i pohrana audio signala. 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 ima mogućnost komprimiranja zvučnih signala sa gubitcima i bez njih. Kompresijom zvučnog signala bez gubitaka u potpunosti je očuvan izvorni oblik signala, no novodobiveni zapis zauzima veću količinu memorije. S druge strane, kompresija s gubitcima ograničava frekvencijski spektar, čime se gubi značajna količina izvornog signala, no dobiveni zvučni zapis zauzima značajno manju količinu memorije. Opus kodek kombinira rezultate obje kompresije kako bi postigao balans između zauzeća memorije i gubitka izvornih informacija [4].

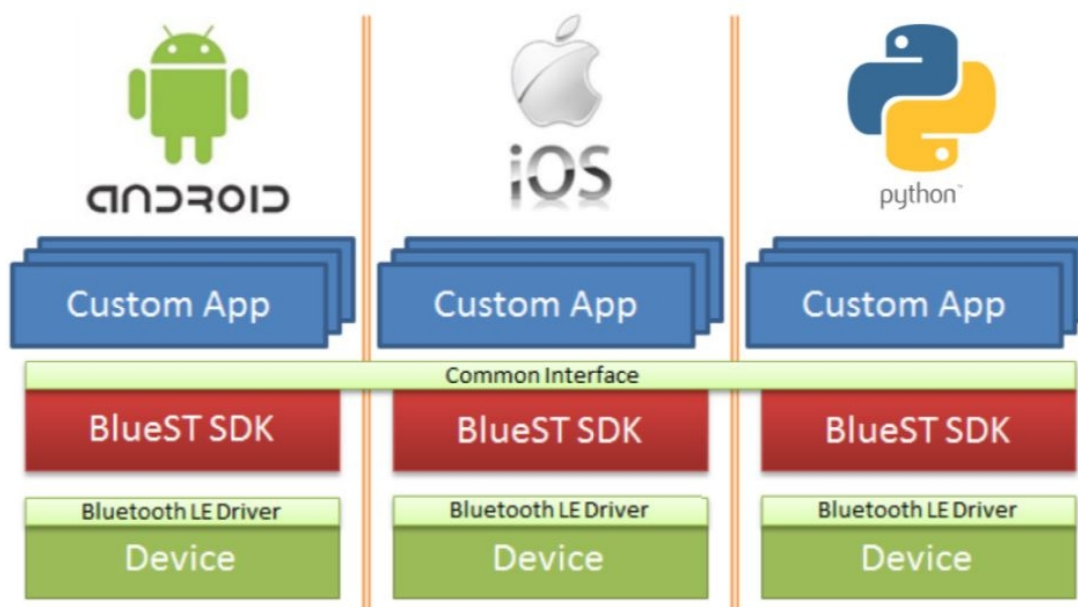
Opus kodek se sastoji od SILK i CELT tehnologija kodiranja. Prvi koristi model temeljen na predviđanju (LPC), dok je drugi u potpunosti modeliran na MDCT transformaciji. Ova svestranost omogućuje Opusu rad u tri načina rada (SILK, CELT ili hibridni način) i osigurava višestruke konfiguracije za različite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

3.2. Programska potpora za računalu

Glavna zadaća aplikacije na računalu je primiti audio signal putem sučelja Bluetooth, prikladno ga obraditi te izravno reproducirati i pohraniti. Za računalnu programsku potporu odabrana je biblioteka *BlueST-SDK* koja omogućuje jednostavan pristup podacima dobivenih od BLE uređaja s implementiranim protokolom *BlueST*. Protokol *BlueST* može se na jednostavan način proširiti kako bi se osigurala podrška za korisnički definirane podatke. BLE protokol već sadrži podršku za različite senzore 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 (engl. *Software development kit* - SDK). Paket za razvoj aplikacija na Linuxu biblioteke *BlueST-SDK* koristi modul *bluepy* dostupan na Linuxu za povezivanje s BLE uređajima [16].



Slika 3.3: Arhitektura aplikacije s modulom *BlueST-SDK* [16]

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. Opis razvojnog okvira *BlueST-SDK*

Biblioteka *BlueST-SDK* prikazuje samo uređaje s poljem specifičnim za dobavljača formatiranim kao što je prikazano u Tablici 3.1. Polje *Duljina* 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. Polje bitova gdje je svaki bit maska neke značajke 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.

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 signala i šuma (SNR).

Tablica 3.1: Oblikovanje polja specifično za dobavljača modula *Blue-SDK* [16]

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

Tablica 3.2: Maske bitova i pripadne karakteristike u modulu *BlueST-SDK* [16]

Duljina	Naziv
26	Razina mikrofona
27	ADPCM Audio
28	Smjer dolaska
29	<i>Switch</i>
30	ADPCM sinkronizacija

Karakteristike kojima upravlja SDK moraju imati navedeni UUID:

XXXXXXXX-0001-11e1-ac36-0002a5d5c51.

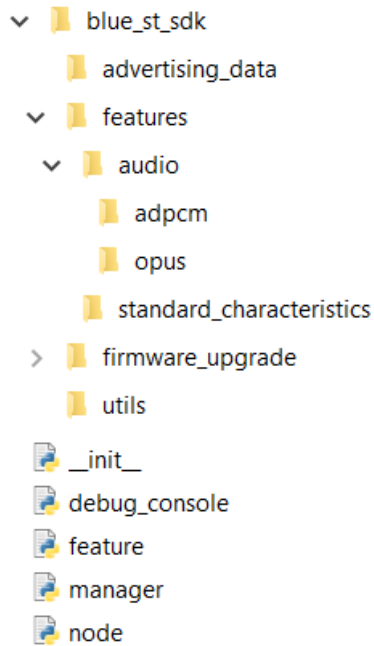
SDK skenira sve usluge, tražeći karakteristike koje odgovaraju uzorku. Prvi dio UUID-a ima bitove postavljene na 1 za svaku značajku koju karakteristika definira. U slučaju više značajki mapiranih u jednu karakteristiku, podaci moraju biti u istom redoslijedu kao maska bitova. Podaci se trebaju formatirati 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* [16]

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

Modul *BlueST-SDK*, odnosno *blue_st_sdk*, koristi biblioteku *bluepy* za povezivanje putem BLE sučelja na operacijskom sustavu Linux. Također, koristi modul *concurrent.futures* za pokretanje skupova dretvi (engl. *pools*) u pozadini, koje poslužuju povratne pozive promatrača. Međutim, zbog ograničenja biblioteke *bluepy*, pri korištenju modula *BlueST-SDK* 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 [2].



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.

Klasa *Manager*

Manager je jedinstveni objekt koji pokreće i zaustavlja proces otkrivanja uređaja i pohranjuje dohvaćene čvorove. *Manager* obavještava novootkriveni čvor putem sučelja *ManagerListener*. Svaka *callback* funkcija 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 klase *Manager*, te izvršava određen skup naredbi pri svakoj promjeni objekta *Manager*. Budući da bi na promjenu stanja *ManagerListener* čekao u beskonačnoj petlji, odnosno ne bi obavljao nikakav koristan rad, objekt *Manager* 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 objekta *Manager*.

Klasa *Node*

Klasa *Node* 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 sučelja *NodeListener*.

Čvor može biti u jednom od sljedećih stanja:

- *init*: početni status,
- *idle*: čvor čeka vezu i šalje oglasne poruke,
- *connecting*: uspostavlja se veza sa čvorom te čvor otkriva karakteristike i usluge uređaja,
- *connected*: veza sa čvorom je uspješno uspostavljena,
- *disconnecting*: prekidanje veze sa čvorom koji se zatim vraća u *idle* status,
- *lost*: uređaj je poslao oglašivačke podatke koji su nedohvatljivi,
- *unreachable*: veza sa čvorom je uspostavljena, no ne može se dohvatiti,
- *dead*: finalni status.

Klasa *Feature*

Klasa *Feature* 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 klase *Sample*. *Feature* objekti također obavještavaju korisnike o promjeni podataka putem sučelja *FeatureListener*.

3.2.2. Povezivanje s mikrokontrolerom

Prvi korak kod povezivanja s mikrokontrolerom je pristupanje globalnoj instanci klase *Manager* za kontrolu i skeniranje Bluetooth uređaja. Također je potrebno kreirati promatrač objekta *Manager*, odnosno stvoriti instancu sučelja *ManagerListener*. Korištenje bilo kojeg sučelja promatrača nije moguće bez prethodnog kreiranja vlastite klase koje nasljeđuje sučelje, što je za promatrač objekta *Manager* korisnički definira klasa *MyManagerListener*.

```
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 instanci *Manager* 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 modul *alsaaudio*. *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 [1].

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 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.

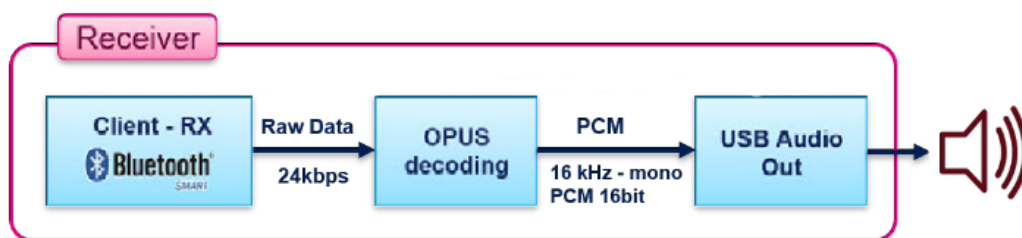
```

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



Slika 3.5: Lanac obrade prijmnika u aplikaciji [19]

Po završetku snimanja zvuka, svi promatrači otkazuju pretplatu na subjekte koje su promatrali te se zatvore audio tokovi. Objekt *Manager* odspaja se 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 [5].

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 [6].

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

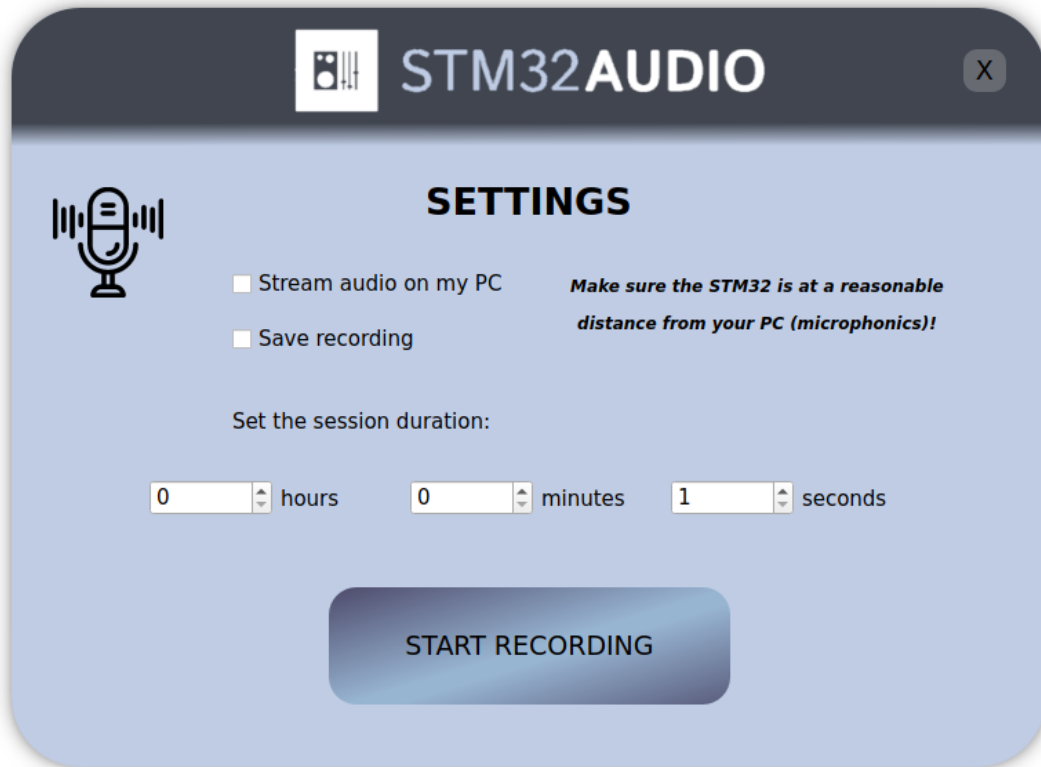
Pri pokretanju aplikacije, u glavnom prozoru prikazuje se izbornik s gumbima *Record* i *Analyse Audio*. *Record* gumb vodi na sučelje za podešavanje parametara snimanja, dok *Analyse Audio* otvara izbornik za odabir audio datoteke nad kojom će se provesti obrada signala.



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 mikrofonijske, koja se događa kada mikrofona 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

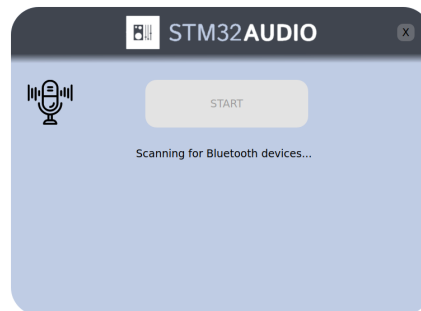
Gumb *Start* pokreće Bluetooth skeniranje uređaja na računalo. Ako nije pronađen uređaj s kojim se računalo može upariti, sustav obavještava korisnika te omogućuje ponovno skeniranje uređaja.

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.

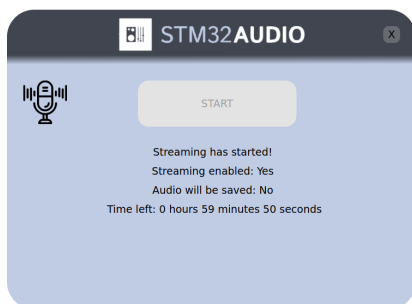
Snimanje je moguće pokrenuti ponovno pritiskom na gumb *Start*, ali s ranije definiranim parametrima.



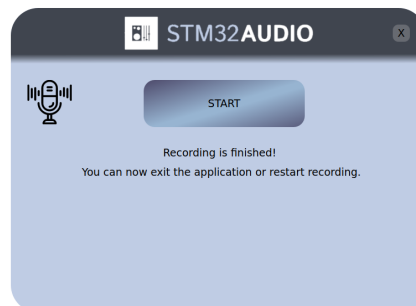
Slika 4.3: Prije pokretanja snimanja



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 objekt *Form*. Taj objekt nasljeđuje dvije klase - *QWidget*, kao glavni prozor s grafičkim komponentama, i samu klasu *Ui_Form*, 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 objekt *QApplication*, 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
- dretve *Worker*

Glavna dretva aplikacije uvijek postoji te se još naziva i GUI dretva. S druge strane, dretve tipa *worker* 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 objekt *QThread* 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 objekt *QThread* nije dretva sam po sebi, nego je omotač oko dretve operacijskog sustava. Prava dretva kreira se pozivom funkcije `QThread.start()`.

Klasa *Worker* nasljeđuje klasu *QObject* 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: Klasa *Worker*

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

Za početak je potrebno stvoriti instance klasa *QThread* i *Worker*. Nakon toga, funkcijom `moveToThread()` funkcionalnost klase *Worker* prebacuje se u objekt *QThread* 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 objektom *QThread*, 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 dretvom tipa *worker*

5. Obrada audio signala

Nakon otvaranja audio datoteke započinje proces obrade zvučnog signala. Audio biblioteka *SoundFile* koristi se za učitavanje audio datoteke i 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.

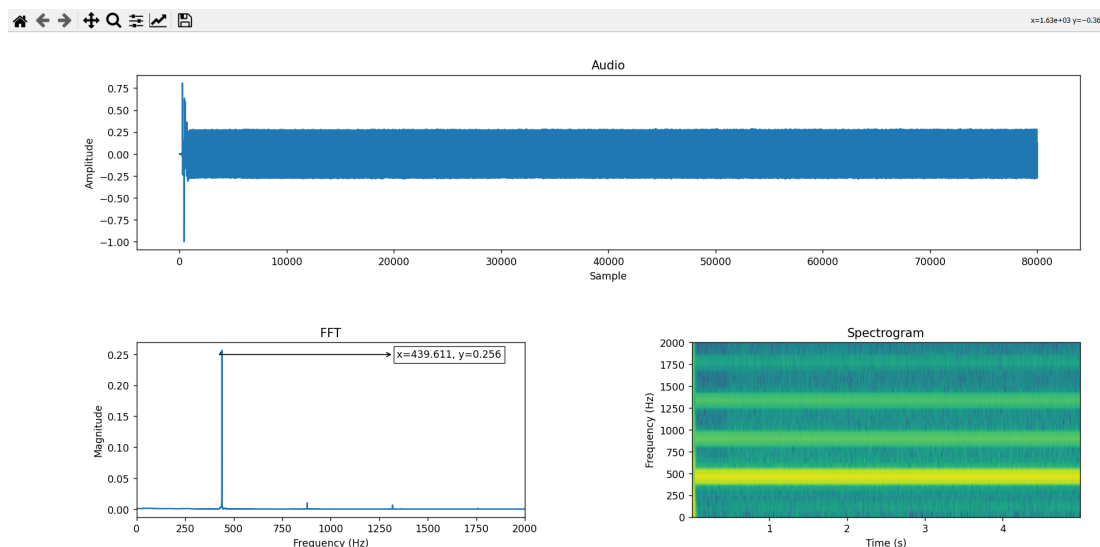
Za analizu odnosa amplitude i frekvencije signala potrebno je transformirati signal u frekvencijsku domenu za prikaz frekvencija koje se nalaze u signalu. Fourierovom transformacijom signal se dekomponira u odgovarajuće frekvencije. Biblioteka *Scipy* sadrži ugrađenu funkciju za brzu Fourierovu transformaciju.

Dobivene matrice iscrtavaju se grafički pomoću biblioteke *matplotlib*. Koristeći funkciju `matplotlib.plot()` prikazuju se tri grafa:

- *Audio*: prikaz samog zvučnog zapisa u vremenskoj domeni,
- *FFT*: prikaz zvučnog zapisa u frekvencijskoj domeni dobiven Fourierovom transformacijom,
- *Spectrogram*: prikaz spektra frekvencija signala koji se mijenja s vremenom.

Prema teoremu Nyquist-Shannon, valni oblik mora se uzorkovati frekvencijom barem dva puta većom od frekvencije signala, odnosno barem 16 kHz. Grafovi stoga prikazuju frekvencije do 8 kHz. Međutim, prikaz frekvencija na grafovima ograničen je na 2 kHz jer zvuk koji čovjek može proizvesti rijetko prelazi tu frekvenciju, stoga su frekvencije iznad 2 kHz redundantne i mogu se ukloniti.

Na Slici 5.1 prikazan je primjer zvučnog zapisa koji je učitao u aplikaciju pomoću funkcije `filedialog.askopenfilename()` iz standardne Python biblioteke *tkinter*. Na grafu zvučnog zapisa u frekvencijskoj domeni također je istaknuta frekvencija na kojoj je vrijednost funkcije maksimalna. Ovaj je podatak koristan pri određivanju dominantne frekvencije u zvučnom zapisu.



Slika 5.1: Primjer prikaza analize zvučnog zapisa

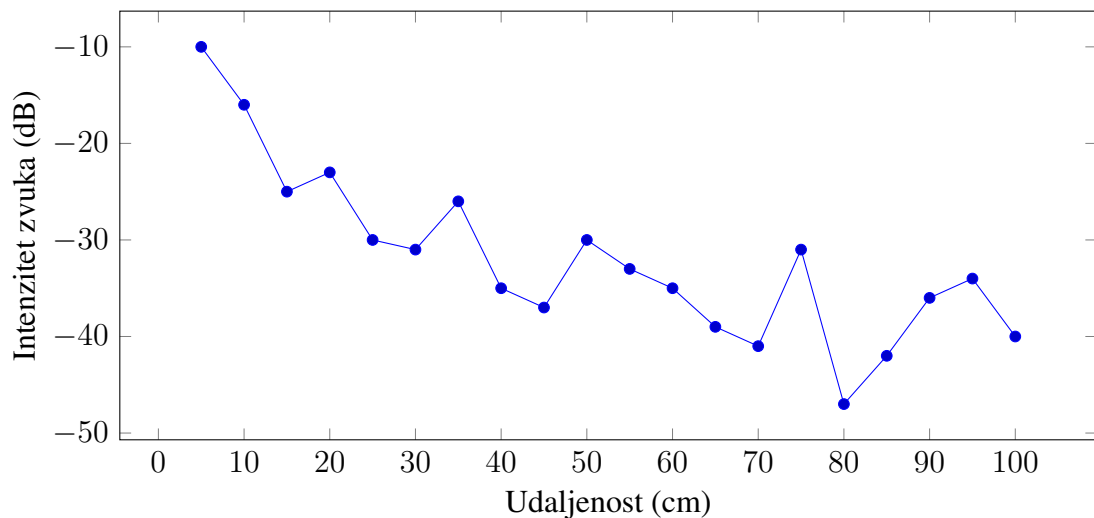
5.1. Odnos intenziteta zvuka i udaljenosti

Za analizu ovisnosti intenziteta zvuka o udaljenosti potrebno je pri konstantnoj frekvenciji periodično mijenjati udaljenost i mjeriti intenzitet zvuka u svakoj iteraciji. Za zvuk je odabran ton A4, odnosno zvuk s konstantnom frekvencijom od 440 Hz. Početna udaljenost izvora zvuka od mikrofona je 5 centimetara, a povećavana je za 5 centimetara u svakoj iteraciji sve do udaljenosti od 100 centimetara, odnosno 1 metra.

Za dobivanje intenziteta svakog zvučnog zapisa korišten je *Audacity*, programski paket otvorenog koda za snimanje i uređivanje audio signala. Program nudi mogućnost analize frekvencije zvučnog zapisa i iscrtavanja grafa ovisnosti intenziteta zvuka o frekvenciji. Za dobivanje željene vrijednosti intenziteta potrebno je iz iscrtanog grafa u programu pročitati tu vrijednost pri frekvenciji od 440 Hz. Nula je postavljena za maksimalnu glasnoću, stoga su vrijednosti intenziteta zvuka prikazani u negativnoj skali.

Provedena je analiza nad 20 zvučnih zapisa i dobivene su vrijednosti grafički prikazane u donjoj tablici.

Graf ovisnosti intenziteta zvuka o udaljenosti



Analizom grafa može se uočiti da postoji obrnuto proporcionalna ovisnost između intenziteta zvuka i udaljenosti izvora od mikrofona - što je izvor dalji od mikrofona, to je intenzitet slabiji. Dobiveni rezultati u skladu su sa zakonom inverznog kvadrata, koji tvrdi da je određena fizikalna veličina obrnuto proporcionalna kvadratu udaljenosti od izvora te fizikalne veličine:

$$intenzitet \propto \frac{1}{udaljenost^2} \quad (5.1)$$

5.2. Obrada zvučnih zapisa hrkanja

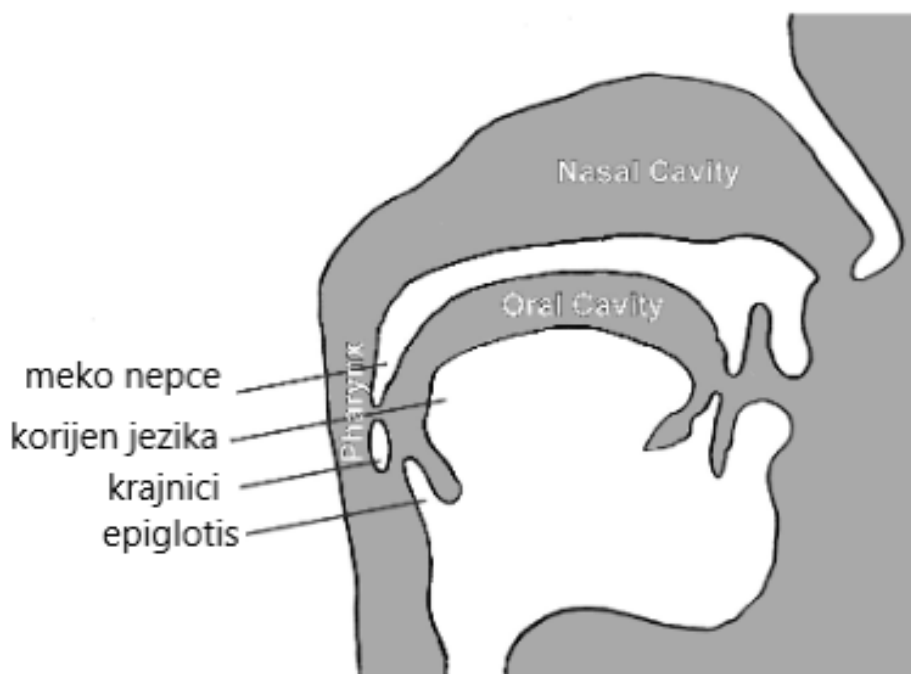
Hrkanje je čest poremećaj koji pogađa 20-40% opće populacije. Ono nastaje kao posljedica vibracije anatomskih struktura u dišnom putu. Treperenje mekog nepca odgovorno je za grubi aspekt zvuka hrkanja. Na hrkanje utječu mnogi faktori kao što su položaj tijela, faza spavanja, dišni putevi i prisutnost ili odsutnost poremećaja disanja tijekom spavanja. Pojava hrkanja može varirati tijekom noći [12].

Iako se hrkanje općenito doživljava kao društvena smetnja, ocjena njegove bučnosti subjektivna je i stoga nedosljedna. Objektivna procjena hrkanja važna je za procjenu učinka terapijskih intervencija. Štoviše, hrkanje nosi informacije koje se odnose na mjesto i stupanj opstrukcije gornjih dišnih putova.

5.2.1. Anatomija hrkanja

Gornji dišni putevi su područje od nosnica i usana do glasnica. Sastoje se od nosne i usne šupljine, ždrijela i gornjeg dijela grkljana. Ždrijelo je stražnji dio glave i sadrži

nekoliko anatomskih orijentira, kao što su meko nepce (velum), nepčani krajnici, korijen jezika i nepčana resica (epiglotis). Epiglotis odvaja ždrijelo od jednjaka i grkljana, koji sadrži glasnice [14].



Slika 5.2: Anatomija gornjih dišnih puteva [14]

Hrkanje je uzrokovano vibracijama struktura mekog tkiva u gornjim dišnim putovima, osobito na fiziološkim suženjima. Tijekom sna, tonus mišića se smanjuje, a meko tkivo opušta, povećavajući njegovu sklonost vibriranju. Brzina protoka zraka pri udisaju povećava se u uskim dijelovima gornjih dišnih puteva, izazivajući vibracije tkiva i strujanja zraka, što zauzvrat uzrokuje hrkanje.

Tipična područja koja pridonose stvaranju zvuka hrkanja su meko nepce i njegov vrh, uvula, koja može vibrirati u anteriorno-posteriornom smjeru, nepčani krajnici koji obično vibriraju u lateralnom smjeru, korijen jezika koja može pasti natrag i ograničiti prolaz prema stražnjoj stijenci ždrijela i epiglotis, koji se može spustiti zbog smanjene strukturne krutosti ili zbog stražnjeg pomaka prema stražnjoj stijenci ždrijela.

Za ciljano liječenje hrkanja i srodnih poremećaja disanja povezanih sa spavanjem, ključno je identificirati mehanizme i mjesta koja doprinose sužavanju dišnih puteva i uzrokuju hrkanje ili respiratorne opstrukcije kod pojedinog subjekta.

Široko korištena definicija raznih mehanizama hrkanja je VOTE klasifikacija, koja razlikuje četiri razine u kojima se mogu pojaviti hrkanje i sužavanje dišnih puteva:

- V - *Velum*: razina mekog nepca,
- O - *Oropharynx*: razina usnog dijela ždrijela,
- T - *Tongue*: razina korijena jezika,
- E - *Epiglottis*: razina epiglotisa.

U radu [15] uspoređeni su zvukovi hrkanja izazvani tijekom nazendoskopije s onima u prirodnom snu pomoću spektra zvučnih frekvencija. Pacijenti s palatinalnim hrkanjem tijekom nazendoskopije u snu imali su srednju vršnu frekvenciju od 137 Hz. Najveća frekvencija hrkanja u korijenu jezika bila je 1243 Hz, a kombinirano hrkanje nepca i jezika 190 Hz. Središnje frekvencije bile su 371, 1094 i 404 Hz. Epiglотиčno hrkanje imalo je vršnu frekvenciju od 490 Hz. Iako rezultati pokazuju da inducirano hrkanje sadrži komponentu više frekvencije zvuka koja nije vidljiva tijekom prirodnog hrkanja, korelacija je dovoljno dobra za kategoriziranje zvučnih zapisa hrkanja.

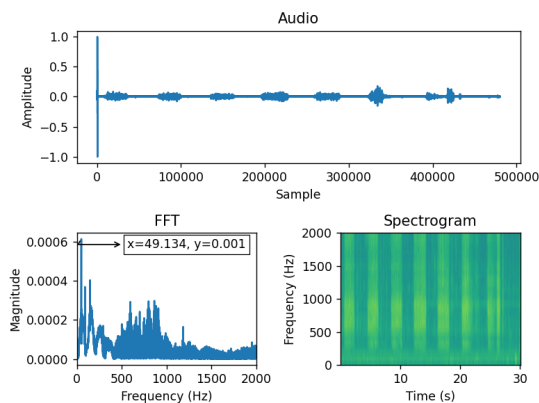
Također, visina zvuka hrkanja u niskofrekventnom rasponu (<500 Hz) odgovara osnovnoj frekvenciji s pripadajućim harmonicima. Visina hrkanja određena je vibracijom mekog nepca, dok je nepalatalno hrkanje više „nalik buci“ i ima raspršeniju energiju u višim spektralnim podpojasevima (>500 Hz).

5.2.2. Analiza zvučnih zapisa

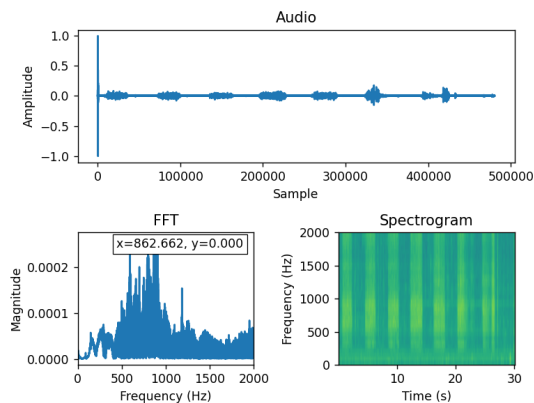
Budući da pri snimanju zvuka mikrofonom snimi i neželjene frekvencije, potrebno ih je filtrirati. Za filtriranje odabrana je frekvencija od 50 Hz, što je frekvencija gradske mreže u Hrvatskoj i uzročnik najviše smetnji pri obradi zvučnog signala. Slike 5.4 i 5.3 prikazuju bitne razlike u očitanim frekvencijama. Na Slici 5.3, koja prikazuje graf bez primjene filtra, vidljivo je da je maksimalna vrijednost na frekvenciji oko 50 Hz, što je upravo frekvencija koja je odabrana za filtriranje. Slika 5.4 prikazuje graf s primjenom *notch* filtra (pojasna brana), i na njoj se jasno vidi koja je stvarna frekvencija za maksimalnu vrijednost grafa.

Pojasna brana je pojasni filter koji kroz većinu frekvencija ostane nepromijenjen, ali prigušuje one u određenom rasponu. Suprotan je propusnom filteru koji ima cilj propuštanja željenih frekvencija. Pojasni filteri odbijaju odnosno prigušuju signale u određenom frekvencijskom pojasu koji se naziva frekvencijski raspon zaustavnog pojasa i propuštaju signale iznad i ispod tog pojasa [3]. U nastavku je prikazana prijenosna funkcija filtra:

$$H(s) = \frac{s^2 + \omega_0^2}{s^2 + \omega_c s + \omega_0^2} \quad (5.2)$$



Slika 5.3: Analiza bez *notch* filtra



Slika 5.4: Analiza s *notch* filtrom

gdje je ω_0 središnja frekvencija koju se želi ukloniti, a ω_c je širina odbijenog frekvencijskog pojasa. Funkcija `iirnotch()` prima središnju frekvenciju za ukloniti, faktor kvalitete i frekvenciju uzorkovanja. Funkcija vraća dva koeficijenta koje prima funkcija `filtfilt()`. Ova funkcija primjenjuje linearni digitalni filter na ulazni signal. Dobivenim koeficijentima koji služe kao brojnik i nazivnik prijenosne funkcije filtrira ulazni signal i vraća ga u filtriranom obliku nad kojim se dalje može pozvati funkcija za Fourierovu transformaciju [7].

```
b, a = iirnotch(50, 50/len(samples), sampling_rate)
notch = firlfilt(b, a, samples)
yf = fft(notch)
xf = np.linspace(start=0.0, stop=1.0/(2.0*T), num=n//2)
```

Isječak koda 5.1: Primjena *notch* filtra i Fourierove transformacije

Razvojnim sustavom STM32WB5MM-DK snimljeno je 27 zvučnih zapisa hrkanja različitih osoba. Budući da je vrlo malen uzorak, zvučni zapisi podijelit će se isključivo na palatalno i nepalatalno hrkanje. Od snimljenih zvučnih zapisa samo su 4 imala frekvenciju hrkanja 500 Hz ili niže, što znači da nepalatalno hrkanje unutar odabranog uzorka daleko učestalije od palatalnog.

Uzme li se u obzir istraživanje vezano uz nazoendoskopiju, 4 zvučna zapisa su kombinirano hrkanje nepca i jezika, dok njih 11 pripada kategoriji hrkanja u korijenu jezika. Ostali zapisi nemaju frekvencije koje su usporedive s rezultatima ovog istraživanja.

6. Zaključak

Na razvojnom sustavu STM32WB5MM-DK implementiran je sustav za prijam i prijenos audio signala. Audio signal snima se MEMS mikrofonom ugrađenim u razvojni sustav te se snimljeni signal prenosi putem BLE sučelja na računalo. Na računalo je razvijena aplikacija koja povezuje razvojni sustav i računalo. Korištenje aplikacije pojednostavljeno je kreiranjem korisničkog grafičkog sučelja. Aplikacija i korisničko sučelje izvode se u operacijskom sustavu Linux. Neovisan rad korisničkog sučelja i dijela aplikacije koje komunicira s razvojnim sustavom STM32WB5MM-DK omogućen je višedretvenim izvođenjem procesa. Korisničko sučelje za interakciju s korisnikom i snimanje zvuka razvijeno je kao *PyQt* aplikacija. Razvojni alat *PyQt* pruža mogućnosti za razvoj korisničkih sučelja namijenjenih za različite platforme, uključujući operacijski sustav Linux. Detaljnom analizom zvučnih zapisa hrkanja ispitana je povezanost frekvencije i mjesta nastajanja hrkanja, kao i ovisnost intenziteta zvuka o udaljenosti izvora zvuka.

Konfiguracija sustava opisana ovim završnim radom predstavlja alat za snimanje zvuka koji je moguće koristiti u kliničkom okruženju i ostalim ograničenim okruženjima gdje mikrofoni i korisnik aplikacije ne mogu biti u istom prostoru. Također, njime je omogućena jednostavna analiza zvučnih zapisa.

LITERATURA

- [1] Alsaudio documentation. URL <https://larsimisch.github.io/pyalsaudio/index.html>.
- [2] Bluest-sdk python. URL https://github.com/STMicroelectronics/BlueSTSDK_Python.
- [3] Notch filter. URL <https://www.electrical4u.com/band-stop-notch-filter/>.
- [4] Opus interactive audio codec. URL <https://opus-codec.org/>.
- [5] *PyQt* documentation. URL <https://riverbankcomputing.com/software/pyqt/>.
- [6] *Qt* documentation. URL <https://doc.qt.io/>.
- [7] Scipy docs. URL <https://docs.scipy.org/doc/>.
- [8] What is mems technology? URL <https://www.mems-exchange.org/MEMS/what-is.html>.
- [9] Arthur Fox. What is a mems microphone. URL <https://mynewmicrophone.com/what-is-a-mems-micro-electro-mechanical-systems-microphone/>.
- [10] S. Miyazaki, Y. Itasaka, K. Ishikawa, , i K. Togawa. Acoustic analysis of snoring and the site of airway obstruction in sleep related respiratory disorders. *Acta Oto-Laryngol.*, 1998.
- [11] Arnaud Parent. An explanation of new mems microphone technology and design. 2020. URL <https://www.coventor.com/blog/explanation-new-mems-microphone-technology-design/>.

- [12] Dirk Pevernagie, Ronald M. Aarts, i Micheline De Meyer. The acoustics of snoring. *Sleep Medicine Reviews*, 2010. URL <https://www.sciencedirect.com/science/article/pii/S1087079209000495>.
- [13] Bob Proctor. Bluetooth vs. bluetooth low energy. 2021. URL <https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy>.
- [14] Kun Qian, Christoph Janott, Maximilian Schmitt, Zixing Zhang, Clemens Heiser, Werner Hemmert, Yoshiharu Yamamoto, i Björn W. Schuller. Can machine learning assist locating the excitation of snore sound? a review. *IEEE Journal of Biomedical and Health Informatics*, 2021.
- [15] Agrawal S, Stone P, McGuinness K, Morris J, i Camilleri AE. Sound frequency analysis and the site of snoring in natural and induced sleep. 2002, Clin Otolaryngol Allied Sci.
- [16] *BlueST protocol and SDK*. STMicroelectronics, 2020. URL https://www.st.com/resource/en/user_manual/um2496-getting-started-with-the-bluest-protocol-and-sdk-stmicroelectronics.pdf.
- [17] *STM32WB BLE stack programming guidelines*. STMicroelectronics, 2021. URL https://www.st.com/resource/en/programming_manual/pm0271-stm32wb-ble-stack-programming-guidelines-stmicroelectronics.pdf.
- [18] *Discovery kit with STM32WB5MMG module user manual*. STMicroelectronics, 2021. URL https://www.st.com/resource/en/user_manual/dm00766322-discovery-kit-with-stm32wb5mmg-module-stmicroelectronics.pdf.
- [19] *STM32Cube function pack featuring advanced audio streaming over Bluetooth 5.0 using Opus codec*. STMicroelectronics, 2021. URL https://www.st.com/resource/en/user_manual/um2614-getting-started-with-the-stm32cube-function-pack-for-stm32wb-mcu-featuring-advanced-audio-streaming-over-bluetooth-50-using-opus-codec-stmicroelectronics.pdf.

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

Sažetak

U ovom radu implementiran je sustav za prijam, prikaz i obradu audio signala korištenjem razvojnog sustava STM32WB5MM-DK. Korištene su biblioteke koje omogućavaju snimanje zvuka MEMS mikrofonom na razvojnom sustavu. Korišteno je BLE sučelje za prijenos audio signala s razvojnog sustava na računalo. Razvijeno je grafičko korisničko sučelje za snimanje zvuka i vizualizaciju ranije snimljenih podataka korištenjem razvojnog alata *PyQt*. Aplikacija se izvodi na operacijskom sustavu Linux. Provedena je analiza zvučnih zapisa hrkanja snimljenih razvijenim sustavom.

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

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

Abstract

This thesis describes an implementation of a system for receiving, displaying and analysis of audio signal using STM32WB5MM-DK development board. Libraries that enable audio recording with a MEMS microphone on the development board have been used. BLE interface is used for signal transfer from the STM32WB5MM-DK to the computer. A graphical user interface has been developed for audio recording and audio signal visualization, using *PyQt* development tools. The application runs on the Linux operating system. Analysis of snoring audio recorded with the developed system has been performed.

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