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, lipanj 2022.

SADRŽAJ

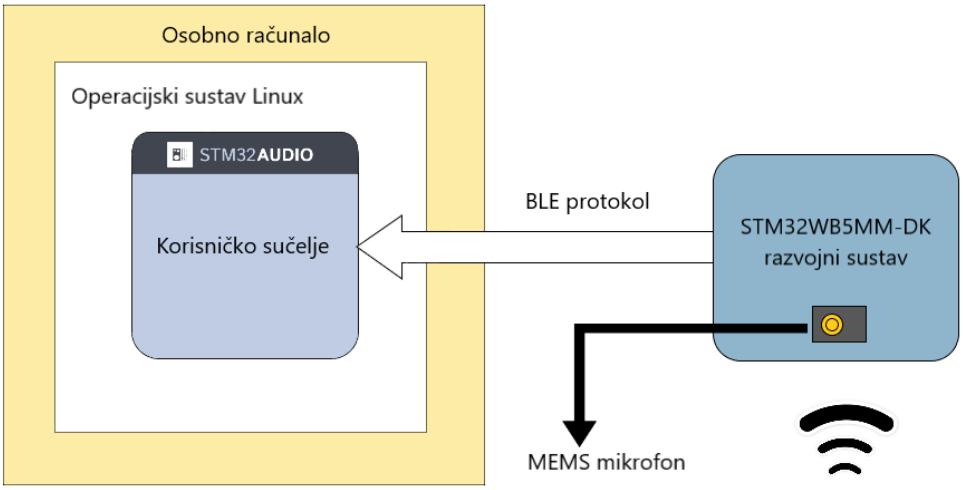
INDEKS SLIKA

INDEKS ISJEČAKA KODA

1. Uvod

Ovaj završni rad dio je doktorskog rada ... koji se bavi ....

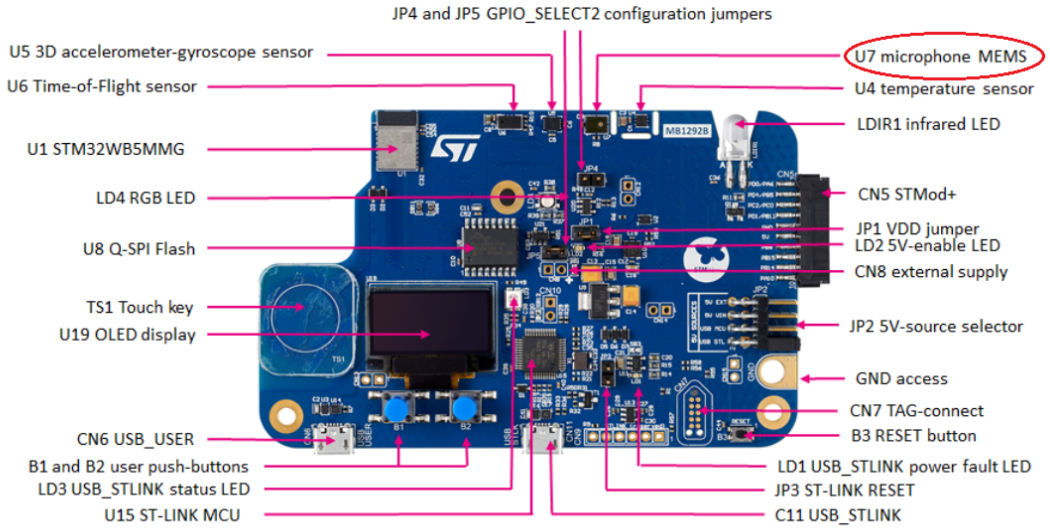
U okviru ovog završnog rada razvijena je programska potpora za mikrokontroler STM32WB5M te je uspostavljeno BLE komunikacijsko sučelje između razvojnog sustava i osobnog računala s operacijskim sustavom Linux. Sučeljem se prenosi zvučni signal sniman MEMS mikrofonom s mikrokontrolera na računalo. Također 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. STM32WB5MM-DK razvojni sustav

Razvojni sustav temelji se na STM32WB5MMG modulu tvrtke *STMicroelectronics*, koji je dio linije STM32WBx5 razvojnih sustava. Kao i svi mikrokontroleri iz te skupine, modul sadrži 32-bitni Arm Cortex-M4, aplikacijski procesor koji radi na frekvenciji do 64 MHz, te Cortex-M0+, mrežni procesor s frekvencijom rada do 32 MHz. Modul sadrži 1 MB *Flash* memorije i 256 KB SRAM-a. Budući da je modul RF primopredajnik, 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, I2C, *Time-of-Flight* senzor i žiroskop. Od ostale periferije najznačajniji je digitalni MEMS mikrofon. STM32 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

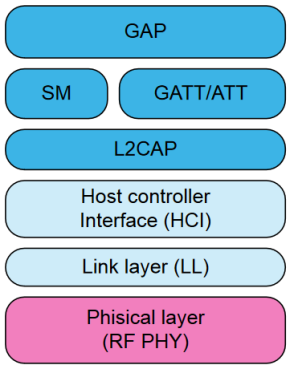
Bluetooth protokol korišten je za povezivanje razvojnog sustava s matičnim računalom i za prijenos audio signala s mikrokontrolera. BLE je vrsta bežične komunikacije namijenjena komunikaciji kratkog dometa s niskom potrošnjom energije. Razvijen je kako bi se postigao standard vrlo male snage koji radi 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 (L2CAP), upravitelja sigurnosti (SM), protokol atributa (ATT), generički profil atributa (GATT) i generički profil pristupa (GAP). Sučelje između komponenti naziva se sučelje host kontrolera (HCI).



Slika 2.2: BLE arhitektura stoga

2.1.1. Upravljač

Fizički sloj

Fizički sloj je radio 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:

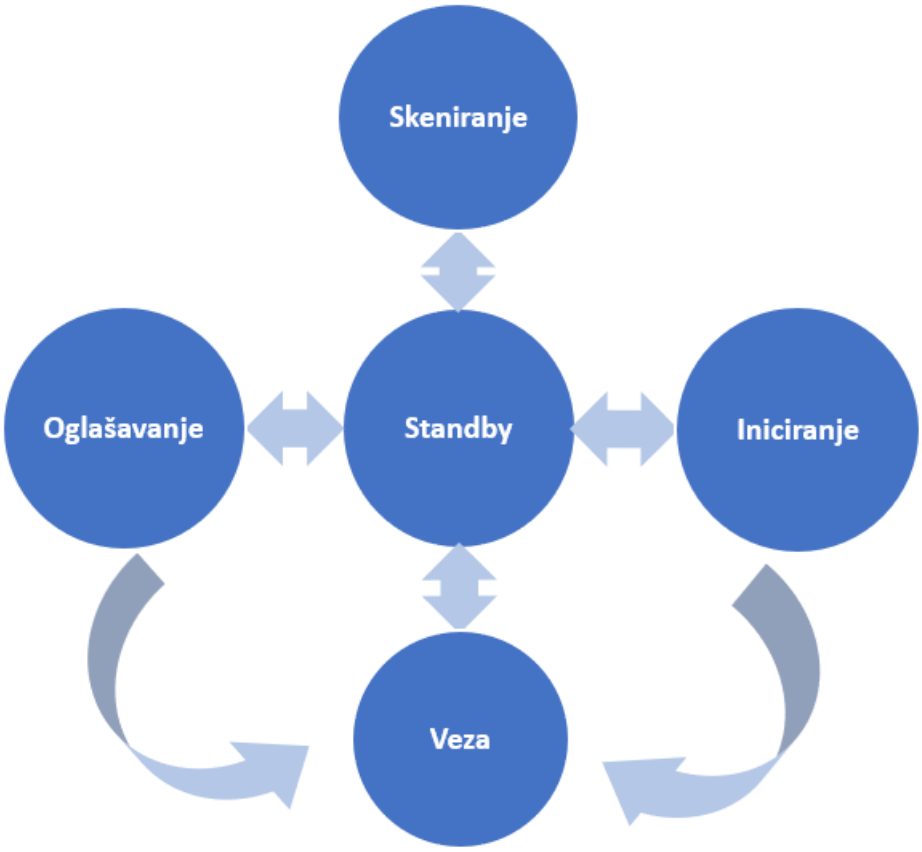
1. Kanali za oglašavanje koji koriste tri fiksna RF kanala (37, 38 i 39) za
   (a) Pakete kanala za oglašavanje
   (b) Pakete korištene za otkrivanje ili povezivanje
   (c) Pakete korištene za odašiljanje ili skeniranje
2. Podatkovni fizički kanal, koristi ostalih 37 RF kanala za dvosmjernu komunikaciju između povezanih uređaja.

BLE je tehnologija adaptivnog skakanja frekvencije (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 radio 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 inicijatora u glavnoj je ulozi (engl. *master*) - komunicira s uređajem u podređenoj (engl. *slave*) ulozi i definira vrijeme prijenosa
- Uređaj oglašivača je u ulozi podređenog - komunicira s jednim uređajem u glavnoj ulozi



Slika 2.3: Automat sloja veze

HCI

Sloj sučelja glavnog kontrolera (HCI) pruža sredstvo komunikacije između hosta i upravljača putem softverskog API-ja 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.

ATT

Protokol atributa (ATT) omogućuje uređaju da prikazuje podatke, koji se nazivaju atributima, drugom uređaju. 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. Poslužitelj i klijent imaju sljedeće uloge:

- Poslužitelj
    - Sadrži sve atribute (baza atributa)
    - Prima zahtjeve, šalje odgovore, izvršava naredbe
    - Obavijesti o promjeni vrijednosti atributa
- Klijent
    - Komunicira s poslužiteljem
    - Šalje zahtjeve, čeka na odgovor
    - Može čitati i mijenjati podatke uz dopuštenje poslužitelja

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 korisnom učitavanju podatkovnog kanala 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.

GATT

Generički atributni profil (GATT) definira okvir za korištenje ATT protokola, a koristi se za usluge, otkrivanje deskriptora, čitanje, pisanje i obavijesti. 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

Uređaj istovremeno može biti i GATT poslužitelj i GATT klijent.

GAP

Bluetooth sustav definira osnovni profil koji implementiraju svi Bluetooth uređaji koji se naziva generički profil pristupa (GAP), koji definira osnovne zahtjeve Bluetooth uređaja. 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

U kontekstu GAP-a definirana su dva temeljna koncepta:

– GAP načini rada (engl. *modes*): konfigurira uređaj da djeluje na određeni način duži vremenski period. Postoje četiri tipa GAP načina rada: emitiranje, otkrivanje, povezivanje i vezanje

– GAP procedure: konfigurira uređaj da izvrši jednu radnju u ograničenom vremenskom periodu. Postoje četiri tipa GAP postupaka: promatrač, otkrivanje, povezivanje, postupci povezivanja

Istovremeno se mogu koristiti različiti načini otkrivanja i povezivanja.

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 (PDM). PDM omogućuje komunikaciju jednom podatkovnom linijom i satom. Prijemnici 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, nano-

tehnologija 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
- Isprintana matična ploča (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 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.

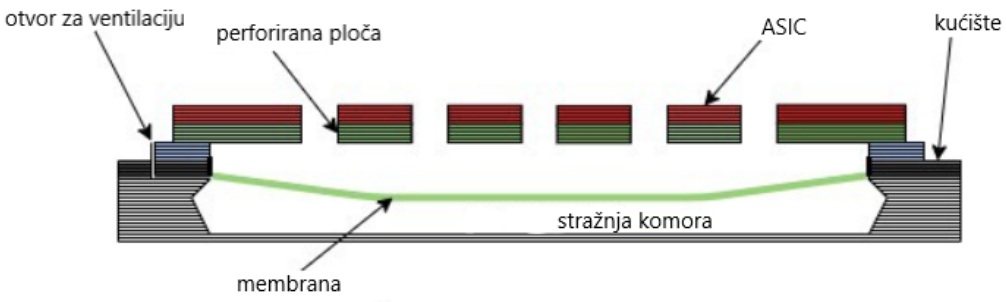
U analognom MEMS mikrofonu, pojačani signal bi se zatim izveo iz MEMS mikrofona i poslao kamo treba ići. Međutim, u digitalnom MEMS mikrofonu postoji dodatni proces u kojem ADC pretvara analogni signal (putem PDM) prije nego što emitira digitalni audio signal.

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 2.4: 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 *drivere* 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.

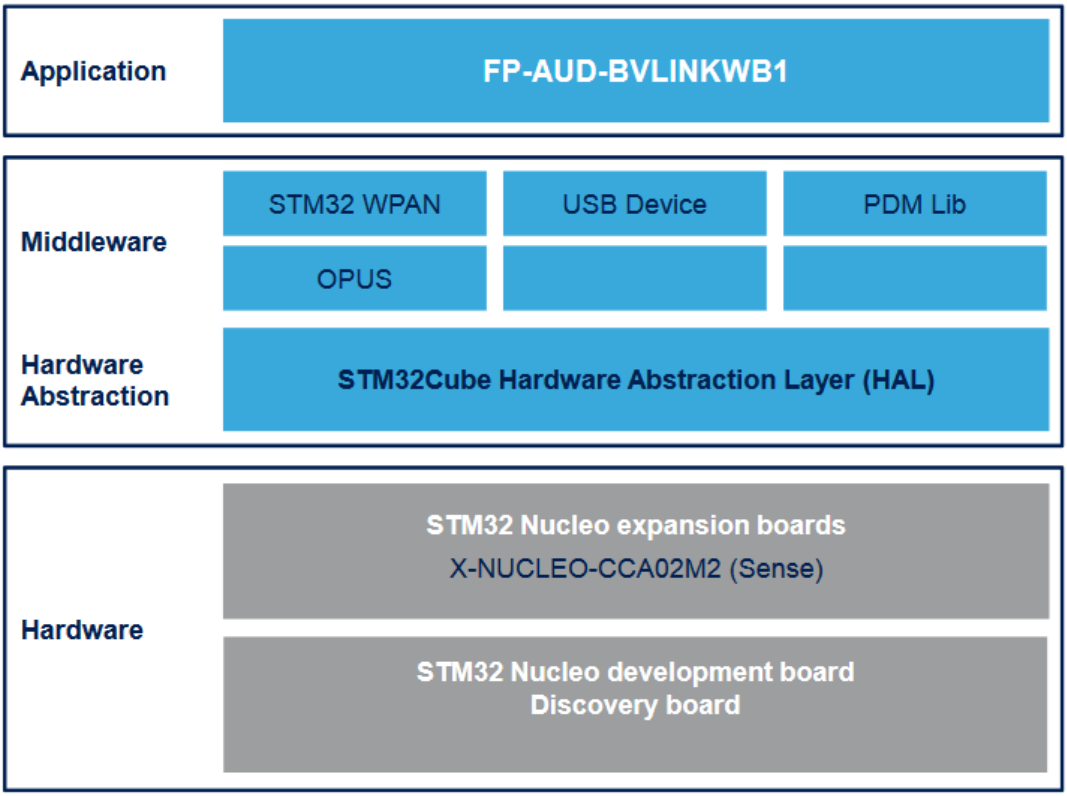
3.1.1. Arhitektura programske potpore za mikrokontroler

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 generički skup generičkih i proširenih API-ja (sučelja za programiranje aplikacije) 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 potrebe za specifičnim hardverskim informacijama mikrokontrolera.

– Sloj paketa podrške za ploču (BSP): skup API-ja koji pruža programsko sučelje za periferne uređaje specifične za ploču 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 prijemnika (Rx), gdje mikrokontroler služi kao odašiljač, a računalo kao prijemnik. 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 prijemnik 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. 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.

3.1.2. 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. Na primjer, 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

BlueST SDK

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* omata 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 utora (engl. *signals and slots*). Signal se emitira pri promjeni stanja objekta, primjerice pritiskom

na gumb unutar korisničkog sučelja. Pri emisiji signala poziva se funkcija utora 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.

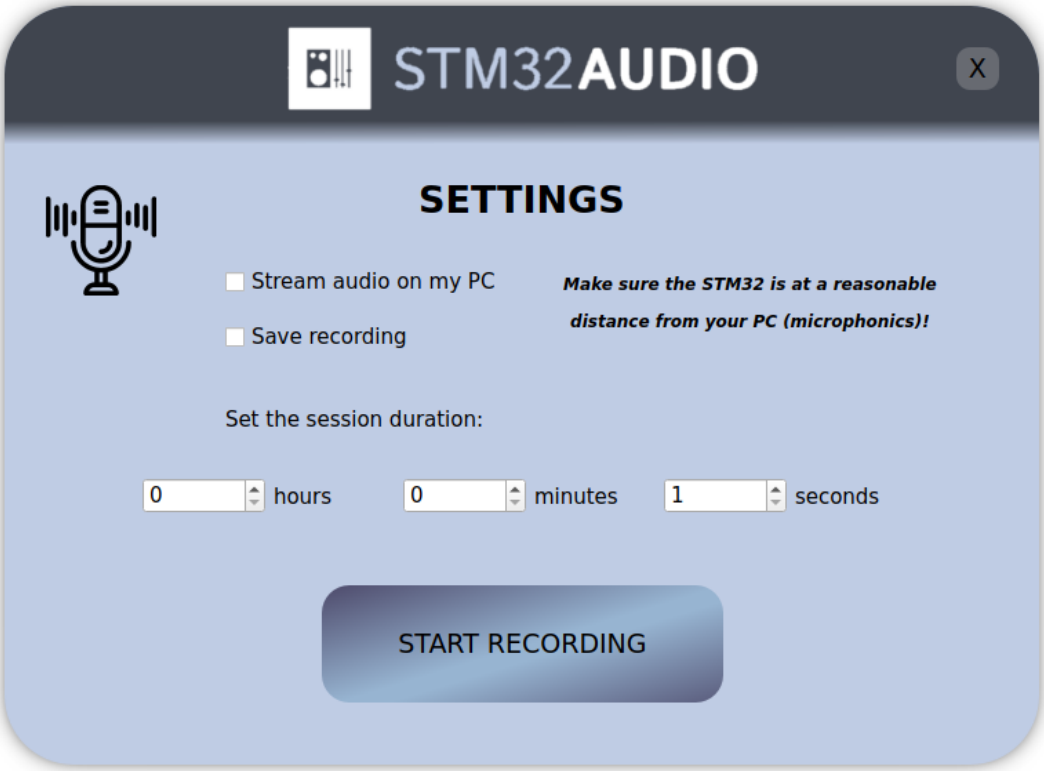
4.2. Implementacija korisničkog sučelja

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 meni 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 računa, kao i trenutnu reprodukciju snimanog zvuka na računalu. 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.



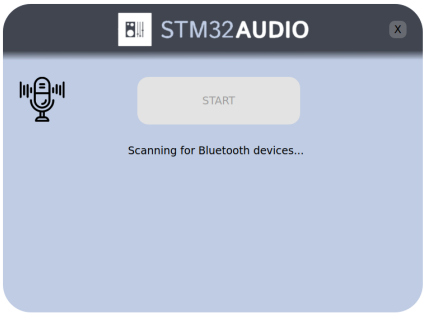
Slika 4.2: Sučelje za podešavanje parametara

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.

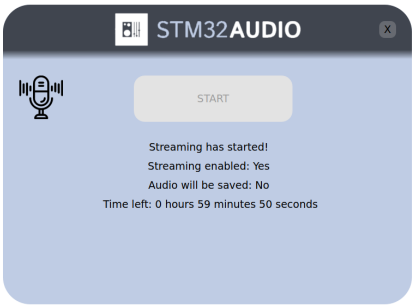
Nakon uparivanja s pronađenim uređajem, pokreće se snimanje zvuka na mikrokontroleru. Korisničko sučelje ispisuje na ekranu prethodno postavljene parametre snimanja i preostalo vrijeme do kraja snimanja. Na završetku primljeni se zvučni zapis pohranjuje obliku *RAW* datoteke, koja se zatim sprema u direktorij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

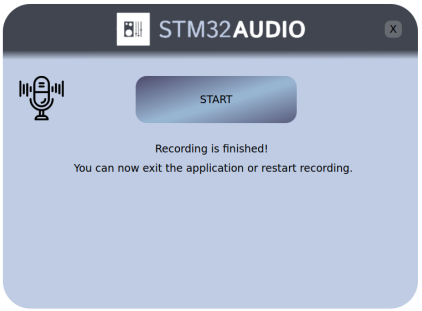
Slika 4.3: Prije pokretanja snimanja



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 tijekom kontrole GUI aplikacije i glavnim postavkama. *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. Potrebno je pozvati ovu funkciju za početak rukovanja događajima. 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, vrši se obrada 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.

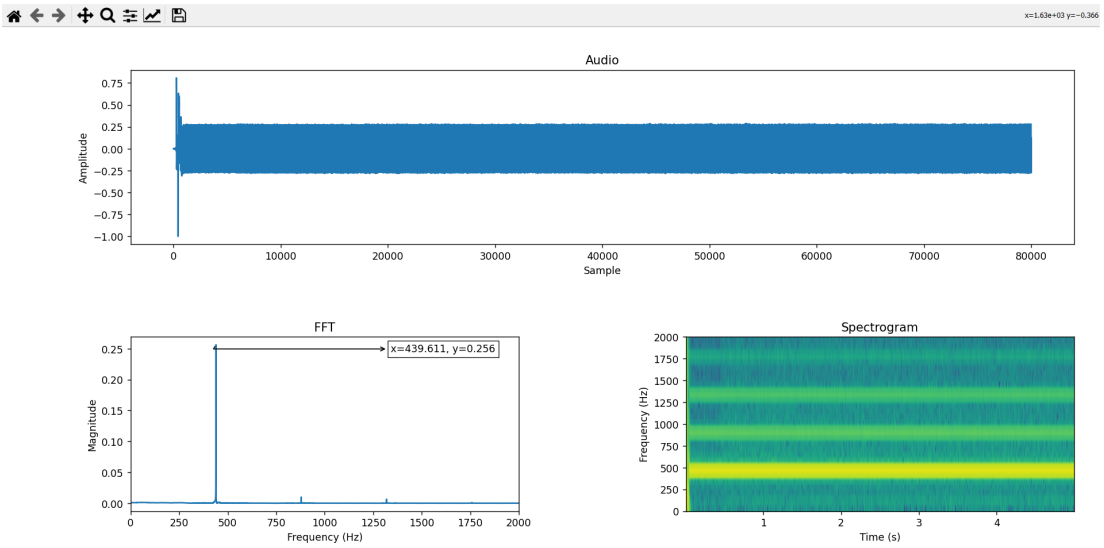
Za analizu odnosa amplitude i frekvencije signala potrebno transformirati signal u frekvencijsku domenu kako bi se prikazalo koje frekvencije se nalaze u signalu. Fourierovom transformacijom signal se dekomponira u odgovarajuće frekvencije. *Scipy* biblioteka sadrži ugrađenu funkciju za brzu Fourierovu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

```
n = len(samples)
T = 1/sampling_rate

b, a = iirnotch(50, 50/len(samples), sampling_rate)
```

Isječak koda 5.1: Fourierova transformacija

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



Slika 5.1: Primjer prikaza analize audiozapisa

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