Realizacija pametnog zvona s kamerom primjenom protokola WebRTC

Brajinović, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

REALIZACIJA PAMETNOG ZVONA S KAMEROM PRIMJENOM PROTOKOLA WEBRTC

Završni rad

Ivan Brajinović

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 15.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Ivan Brajinović
Studij, smjer:	Računalno inženjerstvo
Mat. br. Pristupnika, godina upisa:	R4471, 27.07.2020.
OIB Pristupnika:	64597910043
Mentor:	izv. prof. dr. sc. Ivan Aleksi
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Realizacija pametnog zvona s kamerom primjenom protokola WebRTC
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak završnog rad:	i.MX 8M Mini sustav audio komunikacije u realnom vremenu posredstvom WebRTC komunikacijskog protokola. Nakon aktivacije sustava od strane posjetitelja, uspostavlja se veza preko signalizacijskog servera te započinje audio komunikacija. Pored audio komunikacije, korisnik cijelo vrijeme vidi posjetitelja video prijenosom koji se također odvija u realnom vremenu.
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	15.09.2023.
Datum potvrde ocjene od strane Odbora:	24.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2023.

Ime i prezime studenta:	Ivan Brajinović
Studij:	Računalno inženjerstvo
Mat. br. studenta, godina upisa:	R4471, 27.07.2020.
Turnitin podudaranje [%]:	10

Ovom izjavom izjavljujem da je rad pod nazivom**: Realizacija pametnog zvona s kamerom primjenom protokola WebRTC**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivan Aleksi

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD **1**
1.1. Zadatak završnog rada 1
2. PREGLED PODRUČJA **2**
3. ALATI I TEHNOLOGIJE **5**
3.1. Kernel 5
3.1.1. Yocto projekt 5
3.1.2. Makefile 5
3.1.3. Device Tree 6
3.2. Zvuk 7
3.2.1. ALSA(Advanced Linux Sound Architecture) 8
3.2.2. I2S(Inter-IC Sound) 9
3.2.3. OPUS 10
3.3. WebRTC 11
3.4. i.MX 8 M Mini 12
3.5. SPH0645LM4H-B Mikrofon 13
3.6. Sublime Text 14
3.7. Shell terminal 15
4. REALIZACIJA UREĐAJA **16**
4.1. Konfiguracija SAI sučelja 16
4.2. Snimanje zvuka 19
4.3. Obrada Zvuka 20
4.4. Prijenos zvuka 21
5. ZAKLJUČAK **29**
LITERATURA **30**
SAŽETAK **31**
ABSTRACT **32**

1. UVOD

Kontrola pristupa je jedan generalan problem koji se pojavljuje od samog početka ljudske vrste. Danas se on rješava na mnoge načine i tehnologije te se kontinuirano razvijaju noviji i bolji načini rješavanja tog problema.

Pametno zvono je sigurnosni uređaj komunikacije s posjetiteljima. Aktivacijom sustava od strane posjetitelja pritiskom na gumb, omogućuje se komunikacija s posjetiteljem. Nakon toga se onda može posjetitelju dopustiti ulaz ili ne otvaranjem tj otključavanjem vanjskih vrata na kojima se nalazi pametno zvono. Ovim pristupom se značajno povećava sigurnost kako se ne mora izravno komunicirati s posjetiteljem te se omogućuje ušteda vremena udaljenim upravljanjem vrata. Izvorno ovaj sustav se razvija za firmu koja zbog veličine stambenog objekta naprosto iziskuje ovakav vid automatizacije ali se isti sustav može primijeniti i drugdje, bilo u manjim stambenim objektima ili u osobnim domovima.

U drugom poglavlju se opisuje što je to zvuk te probleme koji postoje kod prijenosa zvuka, u trećem alati i tehnologije korištene za realizaciju pametnog zvona i četvrto poglavlje koje opisuje samu realizaciju pametnog zvona.

1.1. Zadatak završnog rada

i.MX 8M Mini sustav audio komunikacije u stvarnom vremenu posredstvom WebRTC komunikacijskog protokola. Nakon aktivacije sustava od strane posjetitelja, uspostavlja se veza preko signalizacijskog servera te započinje audio komunikacija. Pored audio komunikacije, korisnik cijelo vrijeme vidi posjetitelja video prijenosom koji se također odvija u stvarnom vremenu.

2. PREGLED PODRUČJA

Interfoni su ključni dio moderne sigurnosne i komunikacijske infrastrukture u mnogim zgradama i stambenim objektima. Postoje različita rješenja koja su evoluirala kako bi se udovoljilo različitim potrebama i zahtjevima korisnika.

Jedno od najčešćih rješenja su klasični audio interfoni. Ovi uređaji omogućuju jednostavnu dvosmjernu komunikaciju između unutarnje i vanjske jedinice putem ugrađenih mikrofona i zvučnika. Koriste se u mnogim stambenim zgradama i obiteljskim kućama kao osnovno sredstvo komunikacije s posjetiteljima na ulazu. Iako su jednostavni, i dalje pružaju ključnu funkcionalnost. Izgled audio interfona je vidljiv na slici 2.1.

Slika 2.1. Audio interfon

S razvojem tehnologije, pojavili su se video interfoni. Ovi uređaji nadograđuju klasični audio interfon omogućivanjem ne samo zvučne, već i vizualne komunikacije. Korisnici mogu vidjeti tko se nalazi pred vratima putem ugrađene kamere i razgovarati s njima putem ekrana na unutarnjoj jedinici. Ovo rješenje pruža dodatnu sigurnost, jer korisnici mogu vizualno identificirati posjetitelje prije nego što odluče otvoriti vrata. Video interfoni postaju sve popularniji u stambenim i komercijalnim objektima. Izgled video interfona je vidljiv na slici 2.2.

Slika. 2.2. Video interfon

S razvojem internetske tehnologije pojavili su se IP interfoni. Ova rješenja omogućuju komunikaciju putem internetskog protokola, čime se otvaraju nove mogućnosti. Korisnici mogu pristupiti interfonskom sustavu putem pametnih telefona ili računala, čak i kada nisu kod kuće. Ovo daljinsko upravljanje i nadzor korisno je u mnogim situacijama, omogućujući prilagodbu i kontrolu interfona izvan doma. IP interfoni također omogućuju integraciju s drugim pametnim uređajima i sustavima za kućnu automatizaciju, stvarajući sveobuhvatna rješenja. Izgled IP interfona vidljiv je na slici 2.3.

Slika 2.3. IP interfon

Bežični interfoni postali su popularni zbog svoje praktičnosti i jednostavne instalacije. Eliminiraju potrebu za kablovima, što olakšava postavljanje sustava, posebno u postojećim zgradama gdje bi

provođenje kabela bilo teško ili skupo. Ova rješenja koriste bežične tehnologije poput Bluetooth-a ili Wi-Fi-ja za komunikaciju između unutarnjih i vanjskih jedinica. Izgled bežičnog interfona vidljiv je na slici 2.4.

Slika 2.4. Bežični interfon

Biometrijski interfoni predstavljaju najnoviju sigurnosnu inovaciju. Oni koriste biometrijske podatke kao što su otisak prsta ili prepoznavanje lica za autentifikaciju korisnika. Ovo dodatno povećava sigurnost Interfona, jer osigurava pristup samo ovlaštenim osobama unutarnjem prostoru. Ova rješenja su često korištena u komercijalnim objektima gdje je sigurnost ključna[1]. izgled biometrijskog interfona prikazan je na slici 2.5.

Slika 2.5. Biometrijski interfon

3. ALATI I TEHNOLOGIJE

3.1. Kernel

3.1.1. Yocto projekt

Yocto Project je inicijativa otvorenog koda koja ima za cilj olakšati razvoj ugrađenih Linux sustava za različite platforme i uređaje. Ovaj projekt pruža alate, metode i upute za izgradnju prilagođenih Linux distribucija koje su optimizirane za specifične hardverske platforme. Glavni cilj Yocto Projecta je omogućiti razvoj i održavanje visoko prilagođenih, pouzdanih i optimiziranih Linux distribucija za ugrađene sustave, uključujući različite uređaje kao što su pametni telefoni, IoT uređaji, ruteri, set-top box uređaji i mnogi drugi. Yocto Project koristi model izvornog koda, što znači da pruža sve potrebne komponente za izgradnju Linux distribucije iz samih izvornih kodova. To omogućava fleksibilnost i prilagođavanje distribucije za specifične potrebe. Projekt sadrži sustav za izgradnju zvan "OpenEmbedded", koji omogućava automatsko kreiranje i kompajliranje različitih komponenata distribucije, uključujući Linux jezgru, biblioteke, aplikacije i druge komponente. Yocto Project koristi "recepte" kao skripte koje definiraju kako se svaka komponenta izgrađuje i konfigurira. Slojevi omogućavaju organizaciju i upravljanje receptima, što olakšava dodavanje i prilagođavanje komponenti. Projekt omogućuje programerima prilagodbu distribucije putem konfiguracijskih opcija, kako bi se izdvojili samo potrebni resursi i funkcionalnosti. Yocto Project podržava širok spektar hardverskih arhitektura, što ga čini pogodnim za razvoj na različitim uređajima. Yocto Project ima aktivnu zajednicu programera, inženjera i entuzijasta koji dijele svoje znanje, iskustvo i resurse. Postoji i bogata dokumentacija i forumi gdje se može dobiti podrška i riješiti probleme. Ovaj project je posebno koristan za firme i programere koji razvijaju ugrađene uređaje sa složenim programskim potrebama. Omogućava izgradnju prilagođene Linux distribucije koje zadovoljavaju specifične zahtjeve uređaja, bez potrebe za kretanjem od nule[2].

3.1.2. Makefile

Makefile je tekstualna datoteka koja se koristi za automatizaciju procesa kompajliranja i izgradnje softverskog projekta. On sadrži niz instrukcija koje omogućavaju programerima definiranje kako se izvorni kodovi prevode u izvršni program ili biblioteku. Makefile je posebno koristan za projekte s više izvornih datoteka ili složenim zavisnostima. Ciljevi(targets) predstavljaju krajnje rezultate koji se žele postići kompajliranjem projekta. Na primjer, to može biti izvršna datoteka vaše aplikacije ili biblioteka. Zavisnosti definiraju od kojih izvornih datoteka ili drugih ciljeva

zavisi trenutni cilj. Ako se promijene izvorne datoteke ili druge zavisnosti, Makefile će automatski odrediti koje ciljeve treba ponovo kompajlirati. Naredbe su naredbe koje se izvršavaju kako bi se postigao određeni cilj. To mogu biti naredbe za kompajliranje izvornog koda, povezivanje objektnih datoteka ili izgradnju biblioteka. Varijable se koriste za skladištenje vrijednosti koje se često koriste u Makefile-u, kao što su putanje do direktorija ili opcije kompajlera. Makefile pruža specijalne promjenljive koje omogućavaju pristup informacijama kao što su ime kompajlera, izlazne datoteke, trenutni direktorij i drugo. Pravila definiraju kako se kompajlira izvorni kod za dobivanje određenog izlaza. Svako pravilo ima cilj, zavisnosti i naredbe. Makefile-ovi olakšavaju proces razvoja tako što omogućavaju programerima lako i dosljedno kompiliranje projekata s mnogo datoteka. Kada se izmijeni samo jedna izvorna datoteka, Makefile će automatski prepoznati koje komponente projekta treba ponovo kompajlirati kako bi se osigurala izraženost promjena u izvršnom programu ili biblioteci. Makefile-ovi su posebno česti u projektnim okruženjima koja koriste C, C++, ali se mogu koristiti i za druge programske jezike i tehnologije. Mnogi projekti zajednice otvorenog koda koriste Makefile-ove kao osnovni alat za izgradnju.

3.1.3. Device Tree

Linux Device Tree (DT) je podatkovna struktura koja se koristi u Linux jezgri kako bi opisala hardverske komponente sustava, posebno u ugrađenim i ARM baziranim sustavima. On pruža standardiziran način opisivanja hardverske konfiguracije sustava jezgru, omogućavajući mu razumijevanje različitih uređaja, njihove karakteristike i kako su povezani.

U tradicionalnim računarskim sustavima, jezgra može izravno pristupiti registrima hardvera kako bi konfigurirali uređaje. Međutim, u ugrađenim sustavima, posebno onima s raznolikim hardverskim konfiguracijama, ovakav pristup postaje manje izvodljiv zbog raznovrsnosti hardverskih komponenata i njihovih veza. Device Tree rješava ovaj problem pružanjem čitljivog za čovjeka, a strojno pretraživog opisa hardverske konfiguracije. Napisan je u specijaliziranom jeziku koji opisuje magistrale sustava, mostove, CPU jezgre, memorijske dijelove, kontrolere prekida, periferne uređaje i njihove veze. Linux Device Tree se obično predstavlja kao "spljoštena" struktura podataka, što znači čuvanje informacija u jednostavnoj hijerarhiji u memoriji, omogućavajući brzu i efikasnu obradu. Organizira se u obliku drveta koje se sastoji od čvorova. Svaki čvor predstavlja jedan hardverski element, poput CPU jezgre, memorijskog bloka, periferne komponente itd. Svaki čvor ima svojstva koja opisuju karakteristike, tog elementa, kao što su adrese, registri, interrupt linije, tip uređaja itd. Device Tree kompilacija uključuje alate kao što su

dtc (Device Tree Compiler) koji prevode ljudski čitljive Device Tree datoteke u binarne strukture koje kernel može koristiti. Linux kernel mora biti posebno konfiguriran za podršku Device Tree-u. Prilikom pokretanja sustava, bootloader osigurava binarnu Device Tree datoteku kernelu kako bi mu omogućio prepoznavanje i konfiguraciju hardvera na osnovu opisa iz Device Tree-a. Device Tree je posebno koristan u ugrađenim sustavima, gdje se konfiguracija hardvera može značajno razlikovati od sustava do sustava. Omogućava da se isto Linux jezgro koristi za različite platforme, samo uz primjenu odgovarajućeg Device Tree datoteke[3].

3.2. Zvuk

Zvuk je fizički fenomen koji se doživljava kao vibracija molekula ili čestica koje se šire u mediju kao što su zrak, voda ili čvrsti materijali. To je rezultat oscilacija ili vibracija objekata koji uzrokuju promjene u tlaku i gustoći medija u kojem se šire. Ovaj mehanički prijenos energije omogućuje zvuku dopiranje do naših ušiju i interpretaciju od strane mozga.

Zvuk se općenito opisuje s nekoliko osnovnih karakteristika:

1. Frekvencija: ovo je broj vibracija ili ciklusa zvuka u sekundi, izražen u hercima (Hz). Viša frekvencija daje viši ton, dok niža frekvencija daje niži ton.
2. Amplituda: predstavlja maksimalno odstupanje okolnih čestica od ravnotežnog položaja tijekom titranja.To je povezano s glasnoćom zvuka: veća amplituda znači glasniji zvuk.
3. Valna duljina: Ovo je udaljenost između dvije točke u istoj fazi vibracije.Zvuk visoke frekvencije općenito ima kraću valnu duljinu.
4. Brzina zvuka: ovisi o svojstvima medija u kojem se kreće. Na primjer, zvuk putuje brže kroz čvrste materijale poput metala nego kroz zrak.

Zvuk igra važnu ulogu u svakodnevnom životu i koristi se za komunikaciju, glazbu, otkrivanje prijetnji i više. Ako se razumiju fizička svojstva zvuka, može se bolje njime upravljati i koristiti ga u različite svrhe.

Prijenos zvuka, iako često pouzdan, može se suočiti s raznim problemima koji mogu uticati na kvalitetu i točnost prijenosa zvuka. Evo nekoliko uobičajenih problema prijenosa zvuka:

1. Gubitak signala**:** Prilikom prijenosa zvuka putem žica ili bežičnih kanala, signal može biti oslabljen ili izgubljen usred električnih smetnji, interferencije ili slabih veza. Ovo može rezultirati slabim ili isprekidanim zvukom.
2. Eho**:** Eho se javlja kada odjek zvuka stvoren od refleksije vala uzrokuje da se čuje isti zvuk dva puta sa zakašnjenjem. Ovaj problem može nastati u prostorima s mnogo ravnih površina koje reflektiraju zvuk.
3. Latencija**:** Latencija se odnosi na vrijeme kašnjenja između stvaranja zvuka i njegovog prijema. U digitalnim komunikacijskim sustavima, kao što su video pozivi ili streaming, može doći do latencije koja uzrokuje značajno kašnjenje između usmenog izraza i stvarnog slušanja zvuka.
4. Komprimirani audio**:** Pri kompresiji audio signala radi lakšeg prijenosa, može doći do gubitka detalja i kvaliteta zvuka. Popularni audio formati kao što su MP3 koriste kompresiju koja smanjuje veličinu datoteke, ali ponekad uz gubitak audio kvalitete.
5. Interferencija**:** Električni uređaji, elektronska oprema i drugi izvori elektromagnetne interferencije mogu utjecati na zvuk stvaranjem dodatnih šumova ili iskrivljenja u prijenosu.
6. Akustičke karakteristike prostora**:** Zvuk se ponaša drugačije u različitim akustičkim okruženjima. Loše akustične karakteristike prostora, kao što su velike površine koje reflektiraju zvuk ili naglašeni odjeci, mogu rezultirati lošim razumijevanjem govora ili izobličenim zvukom.
7. Dopplerov efekt**:** Ako izvor zvuka ili slušalac putuju brzo u odnosu na izvor zvuka, dolazi do promjene u frekvenciji zvuka koju čuje slušalac. Ovaj efekt je poznat kao Dopplerov efekt i često se primjenjuje u situacijama kao što su prolazak voznih sredstava.

Rješavanje svih problema zahtjeva kombinaciju tehničkih rješenja, pažljivog planiranja i kvalitetnih komunikacijskih sustava kako bi se osigurao pouzdan i kvalitetan prijenos zvuka.

3.2.1. ALSA (Advanced Linux Sound Architecture)

Advanced Linux Sound Architecture (ALSA) je softverski sloj unutar Linux jezgra koji pruža podršku za upravljanje zvukom i audio uređajima. On omogućava korisnicima i programima komunikaciju s audio hardverom i reprodukciju, snimaju i manipuliraju zvukom na Linux operativnim sustavima. Koristi se za upravljanje zvučnim karticama, mikrofonima, zvučnicima i drugim audio uređajima. Pruža apstrakciju za komunikaciju s različitim hardverskim uređajima i

omogućava aplikacijama pristup zvuku putem jednostavnog sučelja. Pruža audio drivere za različite zvučne kartice i uređaje. Ovi drajveri omogućavaju operativnom sustavu komunikaciju s konkretnim hardverom i iskorištavanje njegove funkcionalnosti. Sastoji se od više komponenti, uključujući ALSA kernel drivere (koji su dio Linux jezgre), ALSA biblioteke (koje pružaju API za pristup zvuku) i ALSA aplikacije (koje omogućavaju korisnicima interakciju sa zvukom). ALSA kernel drajveri omogućavaju jezgri Linux-a komunikaciju izravno sa zvučnim sklopovljem. Oni pružaju podršku za osnovne operacije kao što su reprodukcija i snimanje zvuka, upravljanje glasnoćom i kontrola uređaja. Biblioteke kao što su "libasound" pružaju programerima API za pristup zvučnim funkcijama. Ove biblioteke omogućavaju programima rad s audio uređajima bez potrebe znanja detalja o konkretnom sklopovlju. Dolazi s raznim alatkama koje omogućavaju dijagnostiku, testiranje i konfiguraciju audio uređaja. Na primjer, "alsamixer" omogućava korisnicima prilagođavanje glasnoće i druge postavke. ALSA je ključna komponenta za pružanje zvučne funkcionalnosti u Linux operativnom sustavu. Omogućava korisnicima uživanje u bogatom audio iskustvu i programerima integraciju audio funkcionalnost u svoje aplikacije[4].

3.2.2. I2S (Inter-IC Sound)

Inter-IC Sound (I2S) je komunikacijski protokol koji se često koristi za prijenos digitalnog audio signala između različitih uređaja. I2S je posebno optimiziran za prijenos audio podataka, što ga čini popularnim izborom za komunikaciju između digitalnih audio čipova, mikrokontrolera, DSP-ova (*Digital Signal Processors*) i drugih audio uređaja.

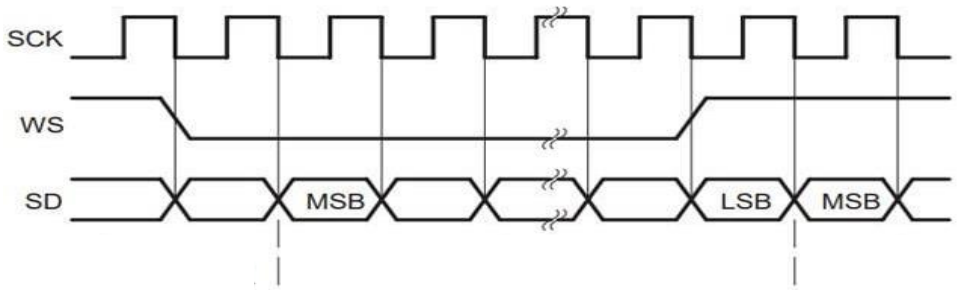
Glavne karakteristike I2S protokola:

- Vremenski sinkroniziran: I2S koristi vremenski sinkronizirane signale za prijenos podataka. Ovo omogućava precizno usklađivanje audio uzoraka između uređaja i osigurava točnu reprodukciju zvuka.
- Linije za podatke: I2S koristi tri glavne linije za prijenos podataka
- *Master* i *Slave* režim: I2S može raditi u *Master* ili *Slave* režimu. *Master* uređaj generira taktove i kontrolira sinkronizaciju, dok *Slave* uređaj prihvata taktove od *Master* uređaja.
- Bitova po uzorku: I2S omogućava prenos različitog broja bitova po uzorku, često 16 ili 24 bita. To omogućava visokokvalitetan prenos audio podataka.
- Lijevo-desni signal: I2S koristi lijevo-desni signalizaciju kako bi označio da li se radi o lijevom ili desnom audio kanalu. Ovo je važno za *stereo* reprodukciju.

I2S je posebno koristan u audio uređajima kao što su DAC-ovi (*Digital-to-Analog Converters*) i ADC-ovi (*Analog-to-Digital Converters*), gdje su preciznost i sinkronizacija bitna za pravilnu reprodukciju zvuka. Osim toga, I2S se također može koristiti za komunikaciju između mikrokontrolera i audio čipova u raznim aplikacijama. Prijenos podataka u I2S (*Inter-IC Sound*) protokolu se odvija koristeći tri glavne linije: *Serial Data Line* (SD), *Bit Clock Line* (BCLK) i *Word Clock Line* (LRCLK/WS) [6]. Grafički prikaz protokola je prikazan na slici 3.2.2.

Redoslijed događaja:

1. *Bit Clock* (BCLK) generira taktove za svaki bit podataka.
2. *Word Clock* (LRCLK/WS) generira signal koji označava početak svakog uzorka (lijevog i desnog kanala).
3. Na padajućoj ivici *Word Clock*-a (LRCLK/WS), prvi bit (najmanje značajni bit) uzorka se postavlja na liniju *Serial Data* (SD-L).
4. Svaka uzastopna padajuća ivica *Bit Clock*-a (BCLK) prenosi sljedeći bit uzorka na liniju *Serial Data* (SD-L).
5. Kada se prenesu svi bitovi uzorka, *Word Clock* (LRCLK/WS) signalom označava kraj uzorka.
6. Proces se ponavlja za svaki uzorak u audio signalu.



Slika 3.2. I2S komunikacijski protokol

3.2.3. OPUS

Opus je moderni audio format kodiranja i dekodiranja zvuka koji je razvijen kao rezultat zajedničkog napora između Internet Engineering Task Force (IETF) i Xiph.Org Foundation. On je posebno prilagođen za prijenos zvuka u stvarnom vremenu preko Interneta, uključujući VoIP (Voice over IP) komunikacije i striming sadržaj.

Opus je poznat po svojoj sposobnosti pružanja visokokvalitetnog zvuka s niskom latencijom, što ga čini odličnim izborom za komunikaciju uživo i interaktivne aplikacije kao što su videokonferencije i online igre.Adaptivni kodek koji prilagođava stopu bitova u stvarnom vremenu u zavisnosti od sadržaja koji se enkodira. Ovo omogućava optimalnu iskorištenost propusnog opsega i osigura dobru kvalitetu zvuka čak i pri promjenjivim uvjetima mreže. Opus podržava širok spektar brzina bitova, od veoma niskih do visokih brzina. Ovo omogućava prilagođavanje kompromisa između kvaliteta i veličine datoteke prema potrebama. Dizajniran je za obradu različitih vrsta audio sadržaja, uključujući govorni i muzički materijal. Ovo ga čini pogodnim za različite vrste aplikacija, od razgovora preko interneta do streaminga muzike. Slobodan i open-source format. To znači da se može koristiti besplatno i da je dostupan s otvorenim izvornim kodom. Podržava stereo i višekanalni zvuk, što omogućava reprodukciju bogatog audio iskustva u različitim okruženjima[7].

Opus se često koristi za komunikaciju aplikacije putem interneta, streaming muzike, VoIP, videokonferencije i druge slične scenarije. Zahvaljujući svojoj visokoj kvaliteti, niskoj latenciji i adaptivnom bitrateu, Opus je postao popularan izbor za prijenos zvuka u stvarnom vremenu preko mreža.

3.3. WebRTC

WebRTC (Web Real-Time Communication) je open-source tehnologija i standard koji omogućava komunikaciju u stvarnom vremenu (audio, video, i data) između web preglednika i drugih aplikacija. WebRTC je razvijen kako bi olakšao integraciju audio i video komunikacije izravno u web aplikacije bez potrebe za dodatnim dodacima ili plugin-ima.

Evo nekoliko ključnih aspekata vezanih za WebRTC:

1. Audio i Video Komunikacija: WebRTC omogućava direktnu komunikaciju između korisnika putem audio i video poziva. Korisnici mogu uspostaviti vezu bez potrebe za instaliranjem posebnog softvera.
2. P2P (Peer-to-Peer) Komunikacija: WebRTC omogućava direktnu komunikaciju između uređaja koristeći P2P arhitekturu, što znači da podaci idu izravno između uređaja, umjesto da prolaze kroz centralni server. To doprinosi smanjenju latencije i poboljšava kvalitetu komunikacije.

3. Real-Time Data Prenos: Pored audio i video komunikacije, WebRTC podržava i prijenos podataka u stvarnom vremenu. Ovo omogućava razmjenu informacija između korisnika tijekom komunikacije, kao što su čet poruke ili dijeljenje datoteka.
4. Open-Source i Standardizacija: WebRTC je open-source projekt koji je razvijen u okviru Web Real-Time Communications (WebRTC) Working Group unutar W3C (World Wide Web Consortium) i RTCWEB Research Group u IETF (Internet Engineering Task Force). Ovo osigurava standardizaciju i interoperabilnost između različitih implementacija.
5. JavaScript API: WebRTC pruža JavaScript API koji omogućava programerima integraciju audio funkcionalnosti, video i data komunikacije u svoje web aplikacije. Ovo uključuje funkcije za uspostavljanje veza, upravljanje medijima i kontrolu nad komunikacijom.
6. Sigurnost: WebRTC podržava enkripciju podataka putem Secure Real-time Transport Protocol (SRTP) i Datagram Transport Layer Security (DTLS), što osigurava siguran prijenos informacija između korisnika.

WebRTC je postao ključni alat za razvoj aplikacija koje zahtijevaju real-time komunikaciju i interakciju između korisnika. To uključuje video konferencije, audio pozive, webinare, online igre, i mnoge druge aplikacije koje zahtijevaju mogućnost trenutnog prijenosa podataka između korisnika.

3.4. i.MX 8 M Mini

NXP i.MX 8M Mini Evaluation Kit (i.MX 8M Mini EVK) je razvojna platforma koju je razvila kompanija NXP Semiconductors. Ova platforma je dizajnirana da olakša razvoj aplikacija baziranih na i.MX 8M Mini procesoru, koji pripada i.MX 8M porodici procesora. i.MX 8M Mini je ARM baziran sustav-na-čip (SoC) koji se često koristi u raznim aplikacijama, uključujući uređaje za multimediju, industrijske uređaje, pametne uređaje i druge.

i.MX 8M Mini je ARM Cortex-A53 baziran procesor koji dolazi s raznim ugrađenim perifernim funkcijama i podrškom za razne multimedijalne formate. Razvojna Platforma: i.MX 8M Mini EVK je razvojna platforma koja omogućava programerima brzo razvijanje i testiranje svoje aplikacije bazirane na i.MX 8M Mini procesoru. Ova platforma sadrži sve potrebne komponente za pokretanje i razvoj softverskih i hardverskih rješenja. Funkcionalnosti i Povezivanje: EVK sadrži različite ulazno/izlazne (I/O) portove, senzore, ekrane, i druge komponente koje omogućavaju programerima testiranje različitih aspekata svog projekta. Također, EVK često

dolazi s Ethernet, USB, HDMI, i drugim interfejsima za povezivanje sa vanjskim uređajima. Operativni sustav: i.MX 8M Mini EVK obično dolazi s podrškom za različite operativne sustave, uključujući Linux i Android. Programeri mogu odabrati operativni sustav koji najbolje odgovara njihovim potrebama. Primjeri Aplikacija: i.MX 8M Mini EVK se često koristi za razvoj različitih aplikacija kao što su pametni uređaji, industrijske automatizacije, multimedijalni uređaji i drugi. NXP Semiconductors je globalna firma koja se bavi razvojem poluprovodničkih komponenti i rješenja. i.MX 8M Mini EVK je samo jedan od mnogih proizvoda koje NXP nudi programerima za razvoj različitih aplikacija[8]. Fizički prikaz je prikazan na slici 3.4.

Slika 3.4. i.MX 8M Mini – EVK razvojna pločica

3.5. SPH0645LM4H-B Mikrofon

SPH0645LM4H-B je MEMS (Micro-Electro-Mechanical Systems) mikrofon koji je proizveo Adafruit. Ovaj mikrofon je popularan izbor za različite aplikacije koje zahtijevaju detekciju zvuka, kao što su audio snimanje, analiza zvuka, prepoznavanje glasa i druge slične primjene.

MEMS mikrofoni koriste tehnologiju koja kombinira elektroniku i mehaničke komponente na veoma malom nivou. To omogućava izuzetno malu veličinu mikrofona uz visoku osjetljivost i preciznost. SPH0645LM4H-B generira analogni izlazni signal koji varira u skladu s dolaznim zvukom. Ovaj signal može se dalje obraditi ili digitalizirati za daljnju analizu ili reprodukciju. Ovaj mikrofon omogućava izravno digitalno povezivanje preko I2S (Inter-IC Sound) sučelja. To

olakšava integraciju s mikrokontrolerima i drugim digitalnim uređajima. Ima visoku osjetljivost, što omogućuje precizno detektiranje širokog spektra zvukova u okolini. MEMS tehnologija omogućava nisku potrošnju energije, što je pogodno za uređaje s ograničenim napajanjem. Često se koristi u raznim projektima za audio snimanje, analizu zvuka, prepoznavanje glasa, aktivaciju uređaja putem glasa (voice activation), i druge slične primjene.

Ukratko, SPH0645LM4H-B je jako osjetljiv MEMS mikrofon s digitalnim sučeljem koji je pogodan za različite projekte koji zahtijevaju detekciju zvuka. Adafruit i drugi proizvođači često ga koriste za svoje razvojne platforme i DIY projekte koji uključuju audio funkcionalnost.

3.6. Sublime Text

Sublime Text je popularan tekstualni uređivač koda koji je široko korišten u razvoju softvera. On je poznat po svojoj brzini, jednostavnosti upotrebe i bogatom ekosustavu dodataka (plugins) koji pružaju dodatne funkcionalnosti programerima. Sublime Text je dostupan za Windows, mac OS i Linux operativne sustave. Sublime Text je brz i veoma reaktivan. Njegova brzina otvaranja, pretrage i promjene sadržaja čini ga efikasnim alatom za svakodnevni rad. Korisničko sučelje je jednostavno i intuitivno. Pogodan je kako za početnike tako i za iskusne programere. Podržava mnoge jezike programiranja i formate kao što su C, C++, Python, JavaScript, HTML, CSS, Ruby, PHP i mnogi drugi. Korisnici mogu prilagoditi izgled, boje, fontove i druge aspekte kako bi se stvorila lična radna okolina. Sublime Text podržava razne dodatke koji proširuju funkcionalnosti uređivača. Ovi dodaci omogućavaju integraciju sa sustavima kontrole verzija, automatsko zatvaranje tagova, podršku za različite programerske okoline i mnoge druge funkcionalnosti. Snippet-ovi su kratki komadi koda koji se mogu brzo umetnuti u editor, često uz mogućnost automatskog dopunjavanja i prilagođavanja parametara. Omogućava korištenje više kursora istovremeno kako bi se brže izvršavale masovne izmjene u kodu. Podržava podijeljeno prikazivanje više fajlova istovremeno, što olakšava uspoređivanje i rad s više dijelova koda. Sublime Text omogućava kreiranje projekata za bolje organiziranje fajlova i resursa. Ima veliku zajednicu korisnika i razvijača koji stvaraju i dijeljenju dodatke, teme i resurse.

Sublime Text je popularan izbor među programerima koji traže brz i funkcionalan tekstualni uređivač koda s mnogo dodataka. Iako se razvijanje Sublime Text-a smanjilo u korist drugih uređivača, on i dalje ostaje popularan alat za mnoge programere.

3.7. *Shell* terminal

Shell terminal u Linux operativnom sustavu je interaktivna komandna linija koja omogućava korisnicima komuniciranje s operacijskim sustavom putem tekstualnih naredbi. To je sučelje koje omogućava izvršavanje sustavskih naredbi, manipulaciju datotekama i direktorijima, instaliranje i upravljanje aplikacijama, rad s procesima i još mnogo toga. Shell terminal je alat koji omogućava napredno upravljanje Linux sustavom i automatizaciju različitih zadataka. Linux koristi različite tipove shell-ova, pri čemu su najčešći Bash (Bourne Again Shell) i Zsh (Z Shell). Shell je program koji interpretira korisničke komande i izvršava odgovarajuće akcije. Korisnici mogu unositi naredbe izravno u terminal i vidjeti odgovor sustava u stvarnom vremenu. Ovo omogućava interaktivno izvršavanje naredbi i dobivanje trenutnih rezultata. Komande se unose putem komandne sintakse. Svaka komanda obično ima format "naredba opcije argumenti", gde su opcije i argumenti dodatne informacije koje specificiraju kako se naredba izvršava. Terminal omogućava navigaciju kroz direktorije i datoteke putem naredbi kao što su cd (change directory), ls (list), pwd (print working directory) i druge. Korisnici mogu kreirati, kopirati, premještati i brisati datoteke i direktorije putem naredbi kao što su mkdir (make directory), cp (copy), mv (move), rm (remove) i druge. Terminal omogućava pokretanje i upravljanje procesa. Naredbe kao što su ps (process status), kill (terminate process), top (view running processes) su korisne za rad s procesima. Shell skripte omogućavaju korisnicima automatsko izvršavanje naredbi, čime se olakšavaju ponavljajući zadaci. Terminal omogućava pristup sustavskim resursima kao što su konfiguracijske datoteke, sustavski zapisi, i druge informacije. Korisnici s odgovarajućim privilegijama mogu dobiti korijenski pristup (superuser) koji im omogućava potpunu kontrolu nad sustavom. Ovo zahtijeva posebne naredbe kao što su sudo i su. Linux omogućava korištenje virtualnih terminala (TTY) koji omogućavaju više instanci terminala na istom sustavu.

Shell terminal je neophodan alat za administraciju i razvoj u Linux operativnom sustavu. Iako može djelovati zastrašujuće korisnicima koji nisu upoznati s komandnom linijom, ovladavanje osnovama terminala može značajno poboljšati produktivnost i omogućiti dublje razumijevanje Linux sustava.

4. REALIZACIJA UREĐAJA

4.1. Konfiguracija SAI sučelja

Prvi korak realizacije uređaja je konfiguracija samog Linux operacijskog sustava i omogućavanje korištenja vanjskih izvoda razvojne pločice i.MX 8M Mini EVK spojenih na SAI5 komunikacijsko sučelje mikroprocesora i.MX 8M Mini. Način na koji se to postiže je uređivanje stablaste konfiguracije uređaja(DeviceTree ili DT) operacijskog sustava Linux na način se doda jedan čvor koji opisuje sučelje.

DT se nalazi u datoteci naziva "imx8mm-evk.dts", a izmjene i nadopune se odrađuju u datoteci "imx8mm-evk.dtsi" koja je nadopuna DT-u. Svaki čvor namijenjen za opisivanje audio sučelja, između ostalih parametara očekuje informacije i o codec-u odnosno o ADC-u koji pretvara analogne zvučne valove u digitalni oblik pogodan za daljnju obradu. Međutim, kako je SPH0645 digitalni, on ne zahtijeva codec nego se koristi codec koji ne radi ništa, tzv. dummy codec koji koristi samo kako bi se ispunio taj parametar. Njegov kod se nalazi u nastavku.

4.1.1. Programski kod za dummy codec

```
codec_dummy: codec_dummy {
          compatible = "asoc,snd-soc-dummy";
          #sound-dai-cells = <0>;
          frame-master;
          bitclock-master;
          status = "okay";
          };
```

Nakon što je definiran codec tj. njegov placeholder, definira se sučelje određivanjem drivera kojeg je potrebno koristiti, osnovne podatke o tom sučelju koji će se poslije prikazati prilikom korištenja sučelja iz operacijskog sustava, povezuju sa SAI sučeljem te se povezuje codec. Kod se nalazi u nastavku.

4.1.2. Programski kod za audio sučelje

```
sound-sai5 {
        compatible = "simple-audio-card";

        simple-audio-card,name="sound | sai5";
        simple-audio-card,format="i2s";

        simple-audio-card,frame-master=<&sai5>;
        simple-audio-card,bitclock-master=<&sai5>;
        status="okay";

        cpu_dai:simple-audio-card,cpu{
```

```
                sound-dai=<&sai5>;
                system-clock-frequency=<&clk IMX8MM_CLK_SAI5_ROOT>;
        };

        codec_dai: simple-audio-card,codec{
                sound-dai=<&codec_dummy>;
                system-clock-frequency=<&clk IMX8MM_CLK_SAI5_ROOT>;
        };
    };
```

Sljedeća stvar je konfiguracija SAI5 čvora u kojem se određuju parametri kao što su frekvencija, *interrupt* pinovi, dodijeljeni generatori signala takta, smjer komunikacije, način rada te status.

4.1.3. Programski kod za SAI5 čvor

```
&sai5 {
     compatible="fsl,imx8mm-sai";
     pinctrl-names = "default";
     pinctrl-0 = <&pinctrl_sai5>;
     assigned-clocks = <&clk IMX8MM_CLK_SAI5>;
     assigned-clock-parents = <&clk IMX8MM_AUDIO_PLL1_OUT>;
     assigned-clock-rates = <24576000>;

     fsl,sai-mclk-direction-output;
     fsl,txmasterflag = <0>;
     fsl,mode="i2s-master";
     status="okay";
   };
```

I zadnja stvar za konfigurirati su adrese pinova koji se koriste. NXP je istu riješio implementiranjem *pin multiplexing* odnosno jedan pin može koristiti za više stvari te je na programeru određivanje konkretne uloge pina. U ovom slučaju, potrebno je samo definirati točan naziv uloge nakon čega se automatski određuje točna adresa pina(objašnjavanje načina automatskog određivanja izlazi iz okvira ovog završnog rada). Pripadajući kod se nalazi u nastavku.

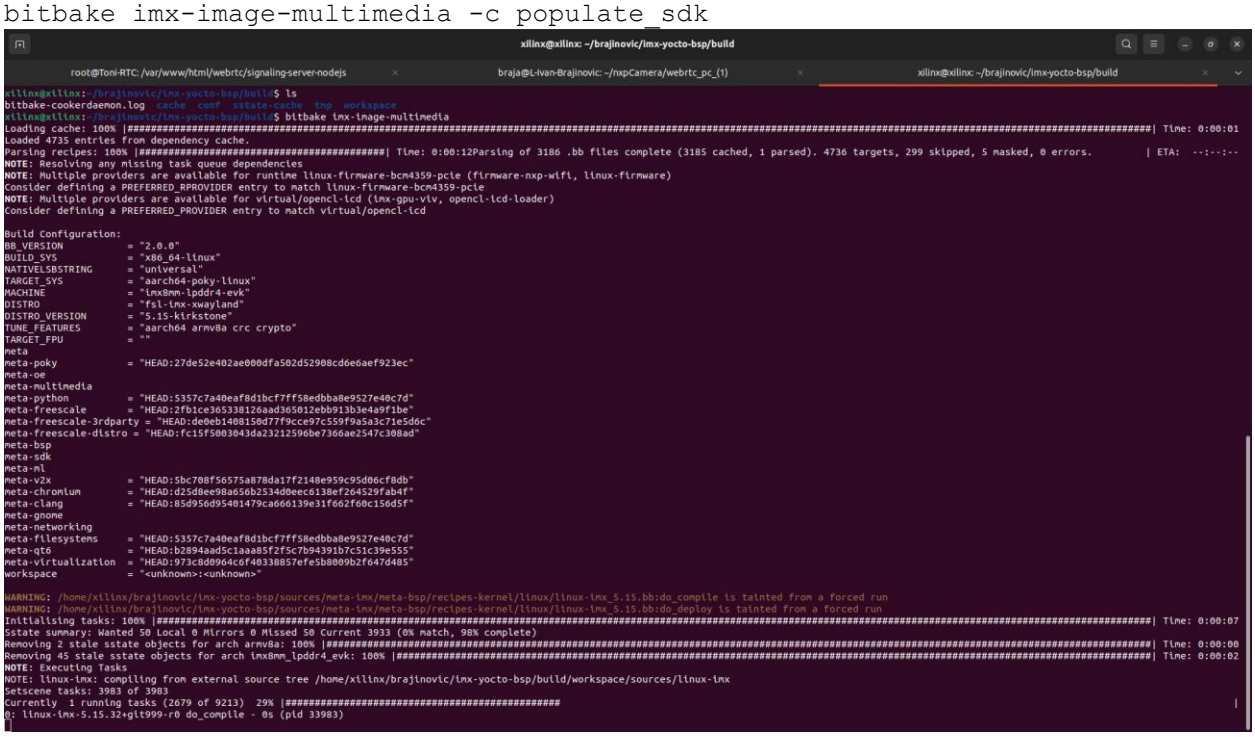
4.1.4. Programski kod za adrese pinova

```
pinctrl_sai5: sai5grp {
        fsl,pins = <
          MX8MM_IOMUXC_SAI5_MCLK_SAI5_MCLK 0xd6
          MX8MM_IOMUXC_SAI5_RXD0_SAI5_RX_DATA0   0xd6
          MX8MM_IOMUXC_SAI5_RXD1_SAI5_TX_SYNC    0xd6
          MX8MM_IOMUXC_SAI5_RXD2_SAI5_TX_BCLK    0xd6
        >;
     };
```

Nakon svih potrebnih izmjena, slijedi kompajliranje našeg *custom* Linux operativnog sustava pomoću Yocto-a pokretanjem sljedećih naredbi:

```
source setup-environment build-braja
```

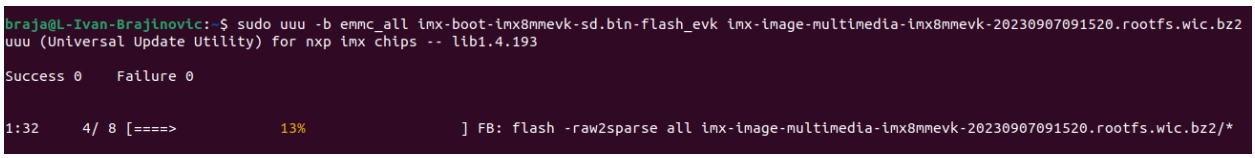
```
bitbake imx-image-multimedia
bitbake imx-image-multimedia -c populate_sdk
```



Slika 4.1. Terminal tijekom kompajliranja Linux operacijskog sustava korištenjem Yocto alata

Nakon završetka kompajliranja, sustav se flash-uje na i.MX korištenjem UUU(*universal update utility*-a):

```
sudo uuu -b emmc_all imx-boot-imx8mmevk-sd.bin-flash_evk imx-image-
multimedia-imx8mmevk-20230629135313.rootfs.wic.bz2
```



Slika 4.2. Temrinal tijekom programiranja i.MX mikrokontrolera

Ovom naredbom se pokreće postupak programiranja i.MX mikrokontrolera upisujući mu naš uređeni Linux operacijski sustav.

Nakon pokretanja sustava, na i.MX pokretanjem naredbe arecord -l vidimo naše sučelje te nakon pokretanja osnovne naredbe za snimanje zvuka možemo utvrditi uspješnost naše konfiguracije.

```
**** List of CAPTURE Hardware Devices ****
card 0: PCH  [HDA Intel PCH], device 0: ALC3204 Analog [ALC3204 Analog]
     Subdevices: 1/1
     Subdevice #0: subdevice #0
```

4.2. Snimanje zvuka

Prethodno se koristio ALSA izravno iz terminala kako bi testirali ispravnost našeg sučelja. Sljedeći kod koristi ALSA library kako bi se odabrao uređaj, postavili parametri te snimio zvuk pomoću spojenog mikrofona. Važni parametri koji se koriste prilikom snimanja zvuka su:

1. brzina uzorkovanja(eng. *sample rate)*

   Brzina uzorkovanja određuje koliko uzoraka se uzima svake sekunde. Obično se mjeri u Hertz (Hz). Na primjer, CD audio kvaliteta koristi brzinu uzorkovanja od 44.1 kHz, što znači da se 44,100 uzoraka uzima svake sekunde.

2. Format

   Format zvuka se odnosi na način na koji su zvučni podaci organizirani i kodirani, kako bi se sačuvali ili prenijeli. Format igra ključnu ulogu u snimanju, reprodukciji, skladištenju i razmjeni zvuka. Postoji mnogo različitih formata zvuka, a svaki od njih ima svoje karakteristike i namjene. Mi koristimo format Signed 16 bit Little-Endian.

3. Broj kanala

   Broj kanala u zvuku se odnosi na broj odvojenih audio putanja ili izvora zvuka koji se koriste u audio snimanju, reprodukciji ili reprodukciji. Broj kanala definira koliko različitih audio signala može biti prisutno u audio zapisu ili reprodukciji i ima značajan utjecaj na prostorni i stereo efekt zvuka.

4. Broj uzoraka po okviru(*frame size*)

   Broj uzoraka koji se koriste u jednom okviru audio zapisa. Na primjer, u stereofonskom audio zapisu, svaki okvir može sadržavati uzorke za lijevu i desnu stvarnu, gdje svaka stvar koristi isti broj uzoraka. Ova veličina okvira može varirati ovisno o formatu audio zapisa, ali se obično definira tako da omogući odgovarajuću reprodukciju zvuka bez gubitka kvalitete.

4.2.1. Programski kod za snimanje zvuka

```
orqa_mic_config_t *sph0645_mic;
sph0645_mic = (orqa_mic_config_t*)malloc(sizeof(orqa_mic_config_t));

strcpy(sph0645_mic->device, "hw:0,0");
sph0645_mic->stream = SND_PCM_STREAM_CAPTURE;
```

```
sph0645_mic->access = SND_PCM_ACCESS_RW_INTERLEAVED;
sph0645_mic->format = SND_PCM_FORMAT_S16_LE;
sph0645_mic->number_of_channels = 2;
unsigned int sample_rate = 48000;
sph0645_mic->sample_rate = sample_rate;
sph0645_mic->number_of_frames = 960;

orqa_create_mic(sph0645_mic);
orqa_record(sph0645_mic, buffer);
orqa_destroy_mic(sph0645_mic->handle);
```

Promatrajući ovaj izlistaj koda stvari bi nam trebale biti poznate iz ranije priče, sve je samo pretočeno u programski kod C. Postoji struktura tipa orqa_mic_config_t koja sadržava sve parametre potrebne za inicijalizaciju mikrofona: format, broj kanala, brzina uzorkovanja, broj uzoraka po okviru. Međutim, postoji i naziv sučelja(sph0645_mic->device), vrsta toka(sph0645->stream) i pravo pristupa(sph0645_mic->access). Prisjetivši se konfiguracije SAI sučelja i naredbe arecord -l mogu se vidjeti brojevi koji označavaju redni broj sučelja(card: 0 u našem slučaju) i device koji označava redni broj uređaja koji koristi to sučelje. Koristeći ta dva redna broja, određuje se uređaj za snimanje zvuka odnosno u našem slučaju je to hw:0,0. Pod pojmom vrsta toka se misli na smjer, snima li se ili reproducira multimedijski sadržaj(u našem slučaju snimanje) te pravo pristupa koje je u ovom slučaju za čitanje i pisanje. Nakon što je struktura popunnjena, poziva se funkcija orqa_create_mic s pokazivačem na ranije spomenutu strukturu te se inicijalizira mikrofon. Nakon toga se poziva funkcija orqa_record s argumentima naše strukture te spremnika za audio podatke nakon čega se mikrofon deinicijalizira.

4.3. Obrada Zvuka

Koristi se OPUS enkoder koji je standardiziran za prijenos zvuka u WebRTC protokolu te se kod kojim se postiže željeno enkodiranje nalazi u nastavku.

4.3.1. Programski kod za enkodiranje zvuka

```
orqa_encoder_config_t *opus_encoder;
opus_encoder = (orqa_encoder_config_t*)malloc(sizeof(orqa_encoder_config_t));
opus_encoder->sample_rate = sph0645_mic->sample_rate;
opus_encoder->number_of_channels = sph0645_mic->number_of_channels;
opus_encoder->bitrate = 64000;
opus_encoder->frame_size = sph0645_mic->number_of_frames;

if(orqa_create_encoder(opus_encoder)){
      fprintf(stderr, "Error creating encoder!\n");
      return -1;
}
orqa_encode(opus_encoder, buffer, opus_buffer);
```

Opet se nalazi na strukturu koja sadrži podatke koji su potrebni kod konfiguracije enkodera. Struktura se sastoji od brzine uzorkovanja, broja kanala, broja uzoraka po okviru te gornje granice broja podataka u jedinici vremena. Nakon toga se inicijalizira enkoder nakon čega se koristi kako bi enkodirali zvuk snimljen mikrofonom te se sad zvuk može pohraniti u datoteku ili dalje s njim manipulirati i prenositi.

Sad, ako se kombiniraju dijelovi za pohranu zvuka kao i za njegovo enkodiranje, dobiva se jedna skripta pomoću koje se pretvara zvuk iz stvarnog svijeta u digitalni, OPUS format koji se može reproducirati.

4.4. Prijenos zvuka

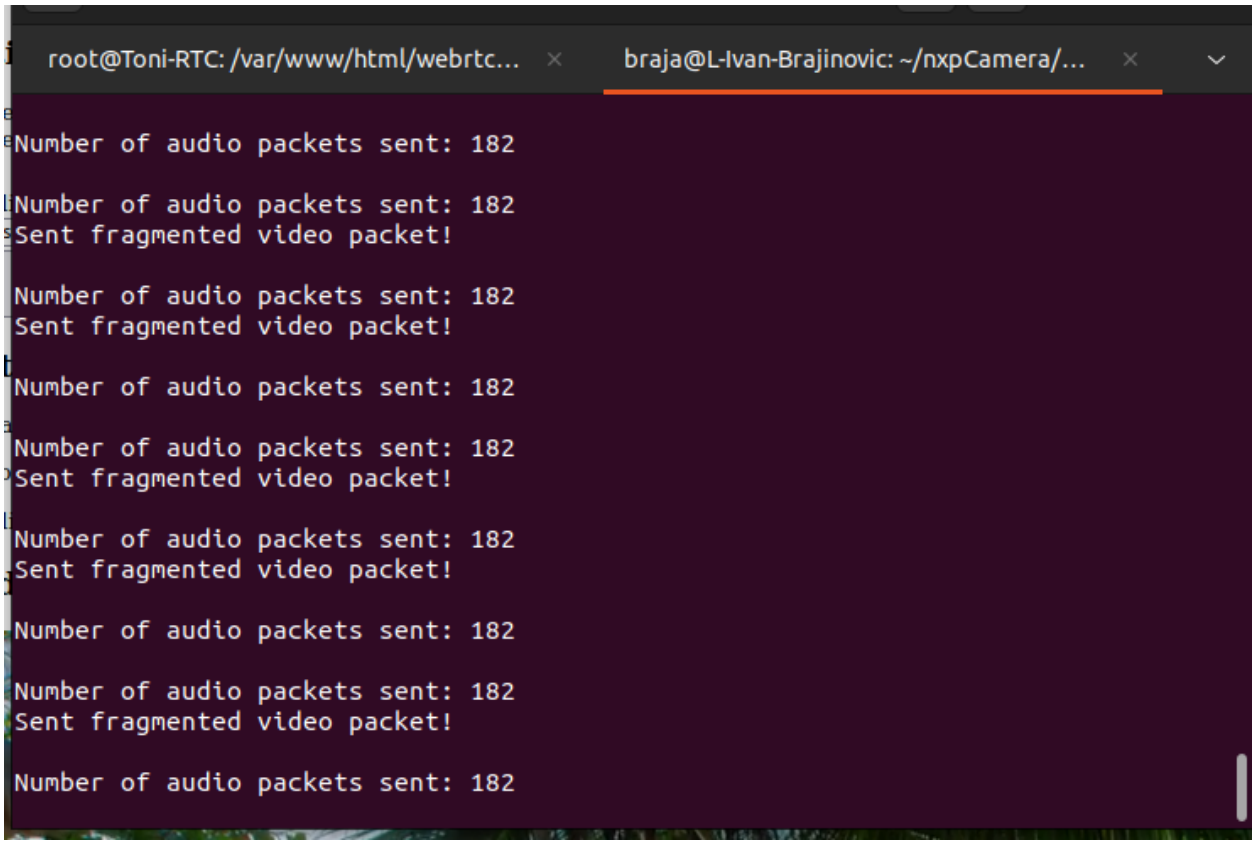
Prijenos zvuka se vrši putem WebRTC komunikacijskog protokola. Nakon uspostavljanja veze između dva klijenta i razmjene postavki komunikacije korištenjem SDP protokola, slijedi prijenos zvuka pakiranjem u RTP pakete te se nakon toga zaštiti SRTP protokolom nakon čega slijedi slanje paketa do drugog klijenta. Ovaj proces komunikacije je jednosmjeran kako nema potrebe za dvosmjernom komunikacijom.

Kod za pametno zvono koji je ujedno i klijent koji šalje podatke prikazan je u nastavku(kako se radi o sigurnosnom uređaju koji će se koristiti u firmi Orqa d.o.o, prikazan je samo neophodan dio koda koji obavlja slanje zvuka).

4.4.1. Programski kod za prijenos zvuka

```
 orqa_mutex_lock(context->mtx);
 int start;
 int end = 0;

 while(context->audio_frame_number > 0){
   start = end;
   end = start + 160;
   orqa_RTP_make_header(audio_packetizer.rtp, audio_packetizer.packet, 0, 0);
   memcpy(audio_packetizer.packet + 12, context->opus_audio_buffer + start,
end - start);

   fprintf(stderr, "\nNumber of audio packets sent: %d\n",
send_packet(audio_packetizer, audio_packetizer.packet, end - start + 12));

    context->audio_frame_number--;
 }

  memset(context->raw_audio_buffer, 0, 960 * 5);
  memset(context->opus_audio_buffer, 0, 960 * 30);

  orqa_mutex_unlock(context->mtx);
```

Prvo se zauzima kritični odsječak kako ne bi u isto vrijeme pristupali istom spremniku. U ovom izlistaju koda se vidi slanje zvuka tako da se spremnik podataka nalazi u strukturi naziva context. Ona se puni u posebnoj niti(kao što se moglo naslutiti iz funkcija orqa_mutex_lock i orqa_mutex_unlock). Stoga postoji i brojač audio_frame_number koji drži podatak o broju okvira koji se nalaze u spremniku kako bi se sav spremnik ispraznio jednom kad dođe vrijeme za slanje podataka. Nakon svih slanja, resetiraju se vrijednosti spremnika te se oslobađa kritični odsječak. Slika 4.4. prikazuje izgled programa tijekom izvršavanja.



Slika 4.4. Terminal tijekom izvođenja programa slanja podataka

Kod za web klijet koji prikazuje i reproducira zvuk je prikazan u nastavku.

4.4.2. Programski kod HTML dijela web klijenta

```
<!DOCTYPE html>
<html lang="en">
<link rel="icon"
href="data:;base64,iVBORw0KGgo=">
<link rel="manifest" id="my-manifest-
placeholder">

<head>
    <meta charset="UTF-8">
    <title>ORQA WebRTC</title>
    <style>
        button {
            padding: 8px 16px;
        }

        pre {
            overflow-x: hidden;
            overflow-y: auto;
```

```
        }

        video {
            width: 1280px;
            height: 720px;
        }

        .option {
            margin-bottom: 8px;
        }

        #media {
            max-width: 1280px;
        }
    </style>
</head>

<body>
    <script
src="https://webrtc.github.io/adapter
/adapter-latest.js"></script>

    <h2>Options (Video Only)</h2>

    <div class="option">
        <input id="use-stun"
type="checkbox" checked />
        <label for="use-stun">Use
STUN Server</label>
        </br>
        <input id="use-wss"
type="checkbox" unchecked />
        <label for="use-wss">Use
Secure WebSocket</label>
    </div>

    </br><label
for="signaling_server_ip">Signaling
Server IP : Signaling Server PORT /
DEVICE_ID</label></br>
    <input id="signaling_server_ip"
value="95.217.13.82" />
    <input id="signaling_server_port"
value="9091" />
    <input id="device_id"
value="server_zvone" /><br>
    <button id="ss_connect"
onclick="changeSignalingServer()"
        style="width: 150px; height:
50px; margin-top: 2px">SS</button>
    <button id="start"
onclick="start()" disabled
style="width: 400px; height: 50px;
margin-top: 2px">Start</button>
    <button id="stop" style="display:
none; width: 400px; height: 50px;
margin-top: 2px"
onclick="stop()">Stop</button>

    <h2>State</h2>
    <p>
        ICE gathering state: <span
id="ice-gathering-state"></span>
    </p>
    <p>
        ICE connection state: <span
id="ice-connection-state"></span>
    </p>
    <p>
        Signaling state: <span
id="signaling-state"></span>
    </p>

    <div id="media" style="display:
none">
        <h2>Media</h2>
        <video id="video" autoplay
playsinline></video>
</div>

    <h2>SDP</h2>

    <h3>Offer</h3>
    <pre id="offer-sdp"></pre>

    <h3>Answer</h3>
    <pre id="answer-sdp"
style="width: 1200px"></pre>

    <script src="client.js"></script>

</body>

</html>
```

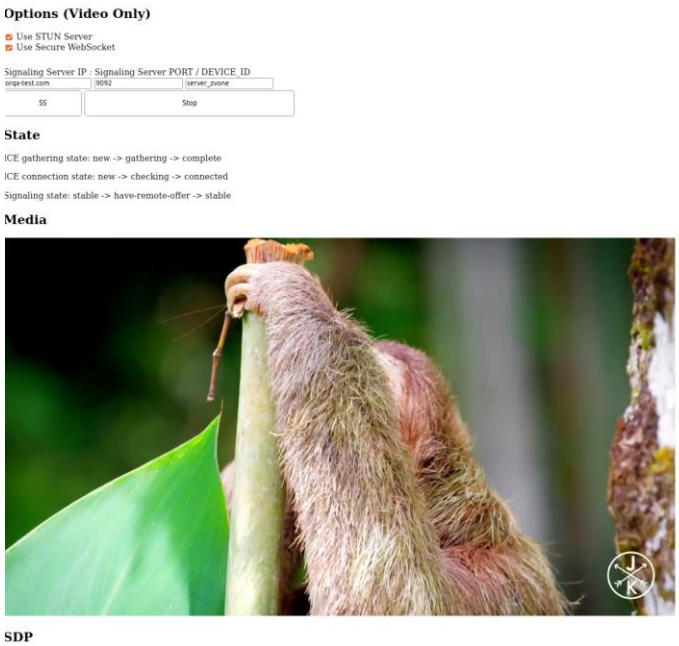
4.4.3. Programski kod JavaScript dijela web klijenta

```
/** @type {RTCPeerConnection} */
let rtc;
const iceConnectionLog =
document.getElementById('ice-
connection-state'),
    iceGatheringLog =
document.getElementById('ice-
gathering-state'),
    signalingLog =
document.getElementById('signaling-
state'),
```

```
    wss_checkbox =
document.getElementById('use-wss'),
    signaling_server_ip =
document.getElementById('signaling_se
rver_ip'),
    signaling_server_port =
document.getElementById('signaling_se
rver_port'),
    device_id =
document.getElementById('device_id');
//dataChannelLog =
document.getElementById('data-
channel');

function randomString(len) {
    const charSet =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghij
klmnopqrstuvwxyz0123456789';
    let randomString = '';
    for (let i = 0; i < len; i++) {
        const randomPoz =
Math.floor(Math.random() *
charSet.length);
        randomString +=
charSet.substring(randomPoz,
randomPoz + 1);
    }
    return randomString;
}

wss_checkbox.addEventListener('change
', function () {
    if (wss_checkbox.checked == true)
{
        signaling_server_ip.value =
"orqa-test.com";
        signaling_server_port.value =
"9092";
    } else {
        signaling_server_ip.value =
"95.217.13.82";
        signaling_server_port.value =
"9091";
    }
});

let receiveID;
let websocket;
function changeSignalingServer() {
    is_wss_enabled = "ws";
    if (wss_checkbox.checked == true)
{
        is_wss_enabled = "wss";
    }

    receiveID = randomString(10);
    websocket = new
WebSocket(is_wss_enabled + '://' +
signaling_server_ip.value + ':' +
signaling_server_port.value + '/' +
receiveID);
    websocket.onopen = function () {

document.getElementById('start').disa
bled = false;
    }

    websocket.onmessage = async
function (evt) {
        const received_msg =
evt.data;
        const object =
JSON.parse(received_msg);
        if (object.type == "offer") {

document.getElementById('offer-
sdp').textContent = object.sdp;
            await handleOffer(object)
        }
    }
}

// data channel
let dc = null, dcTimeout = null;

function createPeerConnection() {
    const config = {
        bundlePolicy: "max-bundle",
    };

    if (document.getElementById('use-
stun').checked) {
        config.iceServers = [{ urls:
['turn:95.217.13.82:3478'],
credential: '1N3eDTuRn', username:
'orqa' }];
        //config.iceTransportPolicy =
'relay'; //Using for testing
purposes, forcing relay connection
    }

    let pc = new
RTCPeerConnection(config);

    // register some listeners to
help debugging

pc.addEventListener('icegatheringstat
echange', function () {
        iceGatheringLog.textContent
+= ' -> ' + pc.iceGatheringState;
    }, false);
    iceGatheringLog.textContent =
pc.iceGatheringState;


pc.addEventListener('iceconnectionsta
techange', function () {
```

```
        iceConnectionLog.textContent
+= ' -> ' + pc.iceConnectionState;
    }, false);
    iceConnectionLog.textContent =
pc.iceConnectionState;


pc.addEventListener('signalingstatech
ange', function () {
        signalingLog.textContent += '
-> ' + pc.signalingState;
    }, false);
    signalingLog.textContent =
pc.signalingState;

    // connect audio / video
    pc.addEventListener('track',
function (evt) {

document.getElementById('media').styl
e.display = 'block';
        const videoTag =
document.getElementById('video');
        if (!videoTag.srcObject) {
            videoTag.srcObject =
evt.streams[0]; // The stream groups
audio and video tracks
            videoTag.play();
        }

    });

    let time_start = null;

    function current_stamp() {
        if (time_start === null) {
            time_start = new
Date().getTime();
            return 0;
        } else {
            return new
Date().getTime() - time_start;
        }
    }

    return pc;
}

function sendAnswer(pc) {
    return pc.createAnswer()
        .then((answer) =>
rtc.setLocalDescription(answer))
        .then(function () {
            // wait for ICE gathering
to complete
            return new
Promise(function (resolve) {
                if
(pc.iceGatheringState === 'complete')
{
                    resolve();
                } else {
                    function
checkState() {
                        if
(pc.iceGatheringState === 'complete')
{

pc.removeEventListener('icegatherings
tatechange', checkState);

resolve();
                        }
                    }


pc.addEventListener('icegatheringstat
echange', checkState);
                }
            });
        }).then(function () {
            const answer =
pc.localDescription;


document.getElementById('answer-
sdp').textContent = answer.sdp;

            return
websocket.send(JSON.stringify(
                {
                    id:
device_id.value,
                    type:
answer.type,
                    sdp: answer.sdp,
                }));
        }).catch(function (e) {
            alert(e);
        });
}

function handleOffer(offer) {
    rtc = createPeerConnection();
    return
rtc.setRemoteDescription(offer)
        .then(() => sendAnswer(rtc));
}

function sendStreamRequest() {
    websocket.send(JSON.stringify(
        {
            id: device_id.value,
            type: "streamRequest",
            receiver: receiveID,
        }));
}

async function start() {
```

```
document.getElementById('start').styl
e.display = 'none';

document.getElementById('stop').style
.display = 'inline-block';

document.getElementById('media').styl
e.display = 'block';
    sendStreamRequest();
}

function stop() {

document.getElementById('stop').style
.display = 'none';

document.getElementById('media').styl
e.display = 'none';

document.getElementById('start').styl
e.display = 'inline-block';

    iceConnectionLog.textContent += '
-> stopped';

    // close data channel
    if (dc) {
        dc.close();
        dc = null;
    }

    // close transceivers
    if (rtc.getTransceivers) {

rtc.getTransceivers().forEach(functio
n (transceiver) {
            if (transceiver.stop) {
                transceiver.stop();
            }
        });
    }

    // close local audio / video
    rtc.getSenders().forEach(function
(sender) {
        const track = sender.track;
        if (track !== null) {
            sender.track.stop();
        }
    });

    // close peer connection
    setTimeout(function () {
        rtc.close();
        rtc = null;
    }, 500);
}
```



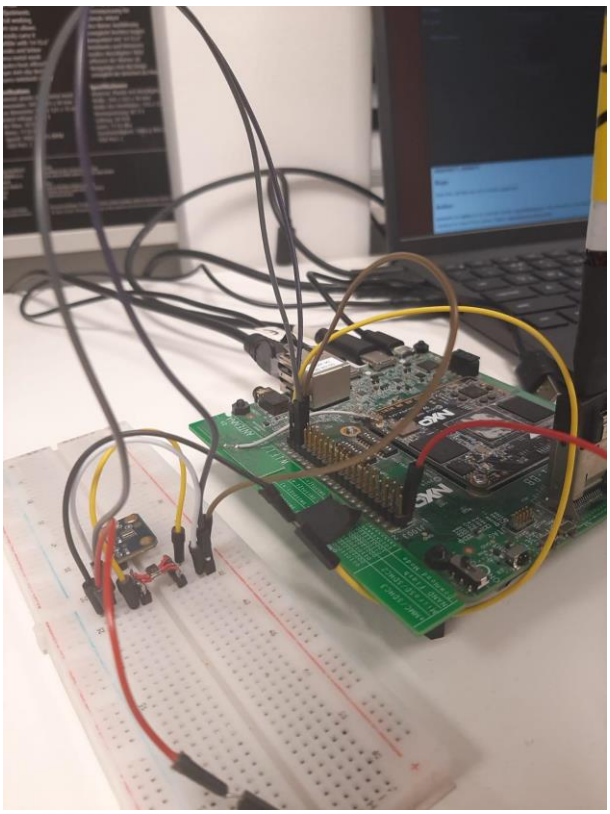
Slika 4.5. Web klijent koji prima i prikazuje podatke

Slika 4.5. prikazuje izgled web klijenta uslijed uspješne komunikacije.

4.4.4. Programski kod MakeFile-a

```
CC = gcc
LD = gcc

CFLAGS = -Isrc/header/ # Main Libraries, Non-custom ones
CFLAGS += -Ivendor/libsrtp/include -Lvendor/libsrtp/lib # Cisco(lsrtp2)
Library

SRCS = $(wildcard src/*.c)
OBJS = $(patsubst %c, %o, $(SRCS))

ORQA_SOURCES_DIR := src
ORQA_SRC_C_DIR := $(ORQA_SOURCES_DIR)
ORQA_SRC_H_DIR := $(ORQA_SOURCES_DIR)/include

ifeq ($(build), static)
LFLAGS = -Wl,-Bstatic -l:libssl.so -l:libcrypto.so -lsrtp2 -Wl,-Bdynamic -
lpthread -ldl
else
LFLAGS = -l:libssl.so -l:libcrypto.so -lpthread -lsrtp2
endif

TARGET = webrtc

.PHONY:all clean

all: $(TARGET)
$(TARGET): $(OBJS)
   @$(CC) $(CFLAGS) $^ $(LFLAGS) -o $@
   $(info Done Building Application: $@)

%.o: %.c
   $(info Building object $@)
   @$(CC) $(CFLAGS) $< -c -o $@

clean:
   @rm -f $(OBJS) $(TARGET)
   $(info Application Cleanup Done)
```

Programski kod iznad opisuje lokaciju datoteka od kojih se cijeli projekt sastoji, koje dodatne biblioteke se koriste te njihove lokacije. Također opisuje i način, redoslijed i pravila prevođenja datoteka u izvršnu datoteku.Slika 4.6. koja se nalazi u nastavku prikazuje fizički izgled razvojnog uređaja zajedno sa mikrofonom koji su spojeni žicama.



Slika 4.6. Fizički izgled i.MX mikroupravljača spojenog na mikrofon preko *header*-a te spojenog na laptop

5. ZAKLJUČAK

U ovom završnom radu napravljen je uređaj koji snima zvuk, obrađuje te šalje na udaljeni web klijent. Opisani su ključni parametri kod rada sa zvukom, proces stvaranja uređaja kao i njegov način rada. Unaprijeđeni sustav pametnih zvona predstavlja inovativno rješenje koje kombinira visokokvalitetno snimanje zvuka i video sadržaja kako bi pružio sveobuhvatan i učinkovit sigurnosni sustav. Integracija Sph0645 mikrofona na i.MX 8M Mini platformi osigurava jasno i precizno snimanje zvukova u okolini zvona, pružajući korisnicima visokokvalitetni audio sadržaj. Ključna prednost ovog sustava je potpuna sinkronizacija video i audio komponenti. Integracija kamere s mikrofonom omogućuje korisnicima istovremeno gledanje i slušanje posjetitelja pred vratima, stvarajući tako cjelovitu sliku događaja. Ova funkcionalnost omogućuje bolje razumijevanje situacije i brže reagiranje na potrebe korisnika. Pametna analitika predstavlja dodatni sloj inteligencije u sustavu. Napredni algoritmi omogućuju prepoznavanje lica i identifikaciju različitih zvukova, pružajući korisnicima dodatne informacije o tome što se događa ispred njihovog zvona.Daljinsko upravljanje je ključni aspekt ovog sustava, koji omogućuje korisnicima upravljanje zvonom putem mobilnih aplikacija ili web sučelja. Ova funkcionalnost pruža fleksibilnost i praktičnost, čak i kada korisnici nisu fizički prisutni kod zvona.Sigurnost i privatnost su od iznimne važnosti, stoga su svi video i audio prijenosi zaštićeni snažnom enkripcijom kako bi se spriječio neovlašten pristup. Također, sustav nudi autentifikaciju korisnika kako bi osigurao pristup samo ovlaštenih osoba. Učinkovito napajanje i mogućnost nadzora potrošnje energije osiguravaju neprekidnu operaciju sustava, produžujući vijek trajanja baterije i smanjujući potrošnju struje.Naposljetku, redovita ažuriranja softverskih komponenti održavaju sustav uvijek u optimalnom stanju, osiguravajući najnovije performanse i sigurnosne standarde.Sve ove komponente čine unaprijeđeni sustav pametnih zvona iznimno funkcionalnim, sigurnim i praktičnim, što korisnicima pruža visokokvalitetno iskustvo i povećava razinu sigurnosti u njihovim domovima ili poslovnim prostorima.

LITERATURA

[1] Types of Intercom Systems: Your Ultimate Guide to Choosing the Right One , dostupno na: https://bas-ip.com/articles/types-of-intercom-systems/[pristupljeno: 1.3.2023.]

[2] What I wish I'd known about Yocto Project — The Yocto Project ® 4.2.999 documentation, dostupno na: https://docs.yoctoproject.org/what-i-wish-id-known.html [pristupljeno: 1.3.2023.]

[3] Device Tree Reference, dostupno na: https://elinux.org/Device_Tree_Reference [pristupljeno: 1.3.2023.]

[4] Advanced Linux Sound Architecture (ALSA) project, dostupno na: https://www.alsa-project.org/wiki/Main_Page [pristupljeno: 15.4.2023.]

[5] Introduction to the I2S Interface, dostupno na: https://www.allaboutcircuits.com/technical-articles/introduction-to-the-i2s-interface/ [pristupljeno: 3.4.2023.]

[6] I2S bus specification, dostupno na: https://www.nxp.com/docs/en/user-manual/UM11732.pdf [pristupljeno: 16.4.2023.]

[7] Opus Interactive Audio Codec , dostupno na: https://opus-codec.org/ [pristupljeno: 15.4.2023.]

[8] i.MX 8M Mini Applications Processor Reference Manual, dostupno na: https://www.nxp.com/webapp/sps/download/preDownload.jsp?render=true [pristupljeno: 16.4.2023]

SAŽETAK

U ovom završnom radu konfiguriran je Linux operacijski sustav izmjenom DeviceTree-a te je omogućen rad sa SAI5 komunikacijskim sučeljem posredstvom I2S komunikacijskog protokola. Uspješno je napravljen program koji komunicira s mikrofonom SPH0645 koji je spojen na i.MX 8M Mini – EVK mikroupravljač te se snimljeni zvuk uspješno enkodira u OPUS format. Od tehnologija korišteni su Sublime Text i Nano za pisanje koda, kompajliranje je odrađeno pisanjem Makefile-ova.

Ključne riječi: DeviceTree, i.MX, I2S, Linux, WebRTC

ABSTRACT

Title: Realization of a smart doorbell camera via WebRTC

In this thesis the Linux operating system was configured in a way that the SAI5 interface was enabled through editing the DeviceTree and can now be used with the I2S protocol. Communication with the SPH0645 microphone which is connected to the i.MX 8M Mini – EVK microcontroller was established via a script and the recorded audio was successfully encoded with the OPUS encoder. Software used to create this thesis are SublimeText and Nano for writing code, the compiling was done by using Makefiles.

Keywords: DeviceTree, i.MX, I2S, Linux, WebRTC