



TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

IP ALAPÚ HANGINTERFÉSZ FEJLESZTÉSE

Szerző: **Maslonka Levente**
villamosmérnök BSc. szak, IV. évf.

Konzulens: **Dr. Gyányi Sándor**
egyetemi adjunktus

Köszönetnyilvánítás

Szeretnék köszönetet nyilvánítani konzulensemnek, Dr. Gyányi Sándornak, aki szakértelmével és tippjeivel támogatta munkámat, valamint Baross Márknak, aki tanácsaival segített.

Tartalomjegyzék

1. Bevezetés.....	2
2. A tervek áttekintése: Mi adta az ötletet?.....	3
2.1 A Dante hálózat.....	3
2.2 Audio over IP és az AES67 szabvány	4
2.3 Szállítási protokollok és szinkronizáció.....	5
2.4 Késleltetés (latency)	5
3. Az eszköz ismertetése	6
3.1 A hardver	7
3.2 A szoftver.....	10
3.2.1 A kernel és annak fontossága.....	10
3.2.2 A/D konverter illesztőprogram: I ² S busz kernelmodul	11
3.2.3 A Streamer Szoftver	13
3.2.4 A Raspberry Pi webfelülete	16
3.3 A késleltetés mérése és kiértékelése	18
4. Továbbfejlesztési lehetőségek	19
5. Összegzés.....	20
6. Források.....	22
6.1 Irodalomjegyzék.....	22
6.2 Ábrajegyzék	24
7. Melléklet.....	25
7.1 A prototípus panel kapcsolási rajza	25
7.2 A Streamer Szoftver forráskódja	25

1. Bevezetés

A digitalizáció megjelenésével, terjedésével már a hétköznapiakban is háttérbe szorult az analóg jelátvitel, nincs ez másképp a hangtechnikában sem. Rengeteg különböző bonyolultságú digitális audio jel továbbítási forma érhető el a piacon. Legyen szó akár a HiFi technikáról (S/PDIF) vagy a professzionális változatokról (AES3, MADI, Ravenna, stb...)

A 2023/24/1. félévben, az 58. Tudományos Diákköri Konferencián egy hallgatótársammal bemutattam egy IP alapú hangosítási rendszer elméleti tervét, mely elképzelés szerint 8 különböző hangsávot tud továbbítani egyidőben az RTP (Real-Time Transport) protokoll segítségével Ethernet hálózaton keresztül. Ennek mintáját az AES67 protokoll és a Dante hálózat adta, valamint az Audio over Ethernet és Audio over IP technológiák. Továbbá olyan eszközöket vett alapul, mint például a Raspberry Pi, vagy a DsPIC. Felhasználási területei pedig stúdiók és színháztermek lettek volna.

Az azóta eltelt időben a tervből megvalósítás lett, bár nem az eredeti elképzelés szerint. A csatornaszám kettőre redukálódott és az egyszerűség kedvéért az RTP protokoll helyett csak az egyszerű UDP-re (User Datagram Protocol) esett a választás. Gyakorlatilag egy jóval egyszerűbb eszközt sikerült megépíteni, amely két csatorna analóg hangjelét fogadja, digitalizálja, majd UDP csomagok formájában peer-to-peer összeköttetés keretein belül elküldi a fogadónak. Ilyen eszköz egyszerű felhasználók számára még nem készült, csak drágább, bonyolultabb kivitelekben terjedt el.

A dolgozat célja bemutatni az eszköz kifejlesztésének folyamatát, ismertetni hasznosságát, valamint továbbfejlesztési lehetőségeit. Mindezek mellett bemutatja egyszerűségét és alkalmazási lehetőségeit is.

2. A tervek áttekintése: Mi adta az ötletet?

A 2023/24/1-es félévben írtam egy dolgozatot *IP alapú hangosítási rendszer* címmel, amelyet egy hallgatótársam közreműködésével adtam le. Manapság nagy mértékben háttérbe szorulnak az analóg jelátviteli és jelfeldolgozási technikák, mivel a digitális eszközök az egyszerű emberek számára is egyre kedvezőbb árakon érhetők el. Sok technológia létezik már különböző jelfeldolgozási és továbbítási módokra, mi viszont egy olyan ötlettel álltunk elő, amely azt taglalja, hogyan lehet kiváltani az analóg, többeres, drága hangcsoport kábeleket viszonylag egyszerűen, hobbi körülmények között.

A sokcsatornás digitális hangátvitelre ma már többfajta megoldás érhető el, legyen szó Ethernet alapú (Ravenna), vagy egyszerűbb, soros átvitelű kommunikációról, mint például a MADI-ról. A már létező eszközök, bár jól működnek és megbízhatóak, magas árat kell fizetni értük. Az egyszerű emberek, átlagfelhasználók számára még nem készült olyan eszköz, amely digitális, többcsatornás audio jelet továbbít informatikai hálózaton keresztül, tehát úgy döntöttünk, megpróbálunk létrehozni egyet a már meglévő technológiákat alapul véve.

2.1 A Dante hálózat

Az *Audinate Ltd.* által fejlesztett, igazán népszerű *Dante* egy többcsatornás hangtovábbítási hálózat. Ez a rendszer informatikai hálózaton belül továbbítja a hangjeleket, az ISO-OSI modell 3. rétegében, magyarul IP alapú kommunikációról beszélünk. Úgy lett megtervezve, hogy az egyszerű felhasználók számára is könnyen konfigurálható legyen, valamint, hogy az említett sokeres csoportkábeleket is kiváltsa. Fast Ethernet hálózaton keresztül 48x48 csatorna volt továbbítható eredetileg, Gigabit Etherneten pedig 512x512. Mindezt értelmezve 48kHz-es mintavételi frekvencia mellett. Emellett az audio jelek forgalmának irányítására is lehetőség van egy Dante eszközön. [\[1. forrás 2. oldal\]](#)

Tulajdonképpen ennek a hálózatnak (és hasonló társainak) az az értelme -a csoportkábelek felváltása mellett-, hogy nincsen szükség külön hanginterfészekre, csupán egy szimpla hálózati kártyára. Az A/D vagy a D/A átalakítás a végeselektronikánál történik.

2.2 Audio over IP és az AES67 szabvány

A Dante tulajdonképpen az Audio over Ethernet (AoE) és az Audio over IP (AoIP) technológiákra alapszik, amelyek az AES67 szabványban vannak definiálva. Ugyanakkor fontos megjegyezni, hogy az AoIP nem keverendő össze a VoIP (Voice over IP) definíciójával. Utóbbi csak beszédátvitelre lett tervezve és a jelben megengedhető a nagyobb, akár 100ms körüli késleltetés is. AoIP esetében teljes, 20Hz-20kHz-ig terjedő spektrumra van szükség, illetve minél alacsonyabb, maximum néhány milliszekundumos késleltetésre. [\[2. forrás 54. oldal\]](#)

Egy AoIP hálózat esetében figyelmet kell fordítanunk a csomagok kapcsolási módjára. Háromféle módot különböztetünk meg, amelyek a: [\[3. forrás\]](#)

- Unicast – mely két pont összekötésére alkalmas (szerver és kliens)
- Broadcast – minek esetében a kiküldött csomagot minden eszköz megkapja
- Multicast – amelynél több forrás és több céleszköz is lehet (point-to-multipoint)

AoIP hálózatoknál általában multicast kapcsolást alkalmaznak annak érdekében, hogy egy kiküldött audio jel ne csak egy célhoz tudjon eléri, hanem többhöz is. Ez továbbá azt is jelenti, hogy D osztályú IP címeket adnak az eszközöknek, melyek nem rendelkeznek alhálózati maszkkal. Egy multicast hálózatnál a fogadó feleknek valahogyan jelezniük kell, hogy egy bizonyos adatfolyamot ők venni szeretnének. Erre használatos az Internet Group Management Protocol (IGMP), a fogadó jelzi a routernek, hogy egy bizonyos csoporthoz csatlakozni szeretne, amelyben az adatfolyam van, majd belép ebbe a csoportba. Ha váltani szeretne, átlép egy másikba. [\[2. forrás 36. oldal\]](#)

Az AES67-es szabvány magába foglalja az IP alapú hangtovábbítás folyamatát. Bár nagyon sok AoIP vagy AoE alapú rendszer ezt veszi alapul, az eredetitől jóval előbb, továbbfejlesztett verzióval találkozhatunk a különböző cégek megoldásainál.

Eredetileg azonban a szabvány a következőket definiálja: [\[4. forrás 31. oldal\]](#)

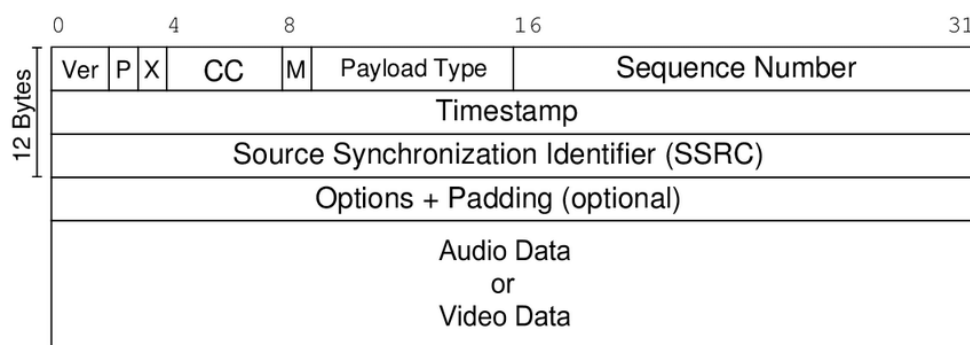
- 16 vagy 24-bites PCM minták
- 44.1, 48 vagy 96kHz-es mintavételi frekvencia
- maximum 8 hangcsatorna egy időben
- 10ms-nál alacsonyabb késleltetési idő [\[5. forrás\]](#)

2.3 Szállítási protokollok és szinkronizáció

Az AES67 szabványban az alacsony késleltetés végett a TCP (Transmission Control Protocol) nem jöhet szóba, mivel rendelkezik többek között hibajavítási mechanizmussal. Továbbá a csomagokat, ha elvesznek újraküldi, valamint a küldő és a fogadó az ún. *hármaskézfogás* procedúrájával alakítja ki a kapcsolatot. Ez nagyon sok időbe telik.

Valós idejű audio (vagy video) továbbítására az UDP-t (User Datagram Protocol) használják. Egyszerű szállítási szolgáltatás, amely nem garantálja önmagában, hogy minden csomag megérkezik, vagy, hogy egyáltalán sorrendben érkeznek. Ezeknek a felügyeletét más rétegekre bízva, de emiatt is alacsony az UDP csomagok továbbításának ideje.

Ennek tetejébe érkezik még az RTP (Real-time Transport Protocol), amely időbélyeggel látja el az UDP csomag fejlécét, valamint az RTCP (Real-time Control Protocol), ami egy visszírányú kommunikációt is lehetővé tesz, hogy a küldő információt kapjon az esetleges csomagvesztésekről vagy késésekről. A szinkronizációért emellett a PTP (Precision Time Protocol) is felel. [\[4. forrás\]](#)



1. ábra Az RTP csomagok felépítése

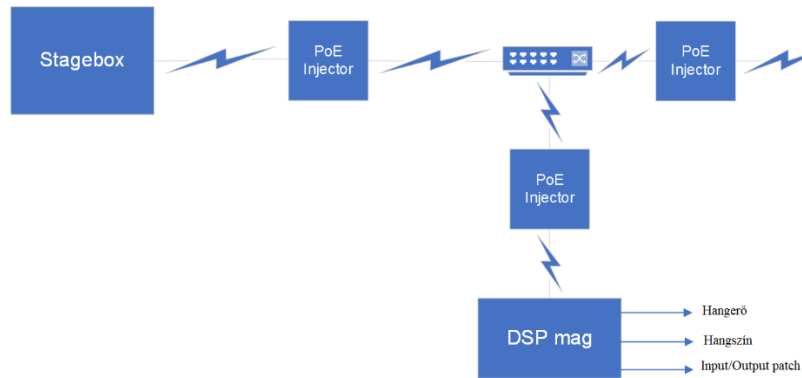
2.4 Késleltetés (latency)

Természetesen az egyik fontos -ha nem a legfontosabb- téma a digitális rendszerekben a késleltetés. Annak érdekében, hogy egy AoIP hálózat eleget tegyen a szabványban leírt követelménynek és a lehető legalacsonyabb legyen a latency, különböző megoldásokat lehet alkalmazni, például:

VLAN (Virtuális helyi hálózat): Erre azért van szükség, hogy el lehessen különíteni egy meglévő hálózaton belül az audio forgalmat a más, egyéb csomagok forgalmától. [\[2. forrás 35. oldal\]](#)

QoS (Quality of Service): Szolgáltatási minőség. Segítségével felállítható a csomagok fontossági sorrendje. Egy audio csomagnak a lehető leghamarabb kell célba érnie, tehát a prioritása magasabb lesz. [\[6. forrás 7. oldal\]](#)

Összegezve tehát, ahhoz, hogy egy IP alapú hangeszközt tudjunk építeni, figyelembe kell vennünk az imént felsorolt és részletezett szolgáltatásokat, szabványokat és protokollokat. Ezek alapján készült el az a rendszerterv, amely az AES67 szabványban előírtaknak is eleget tesz. Felhasználási területei pedig az épülethangosítás, vagy a színháztermek, stúdiótermek lennének.



2. ábra A hálózat általános blokkvázlata

Az egésznek a központjában egy digitális jelfeldolgozó processzor áll, melyen a forgalomirányítás és alapszintű módosítások (hangerő, hangszín) testreszabása történik. Ez tulajdonképpen egy Linux alapú számítógép, mivel ennek a rendszernek a hardverkezelése barátságosabb, mint Windows esetében. Futhat rajta egy audio adat fogadó, IP alapon továbbító szoftver, valamint egy web szerver. Utóbbira azért van szükség, hogy a felhasználók könnyen el tudják érni a beállítási paramétereket egy HTML weboldalon keresztül.

3. Az eszköz ismertetése

Az előző fejezetben ismertetett elképzeléseket alapul véve fel kellett állítani egy megvalósítási sorrendet:

- Szükség van olyan platformra, amin az egyes részek futhatnak majd. Ebben adott volt, hogy Linux alapú gép legyen, szóval egy Raspberry Pi 3B-re esett a választás a sok, különböző interfész támogatás miatt.
- Ehhez a platformhoz illeszteni kell egy A/D konvertert, hogy valahogyan belejuttassuk az analóg hangjeleket.
- A digitalizált PCM audio-t valamilyen formában, lehetőleg tehát RTP segítségével ki kell küldeni a hálózatra.

Így egy „küldő eszköz” megszülethet, ennek kell megépíteni a fordítottját, hogy legyen vevő is. Ebből kétfajta kell, egy céleszköz, amelyre például hangszóró van kötve, és egy Core eszköz, ami a DSP mag része lenne.

A fejlesztés során realizálódott az a tény, hogy egy ilyen rendszer megépítése nagyon komplex és összetett feladat. Az egyes részek több különböző dologból tevődnek össze -főleg szoftveresen- és ezek kiismerése hosszú idő. Emiatt az egész terven módosítani kellett.

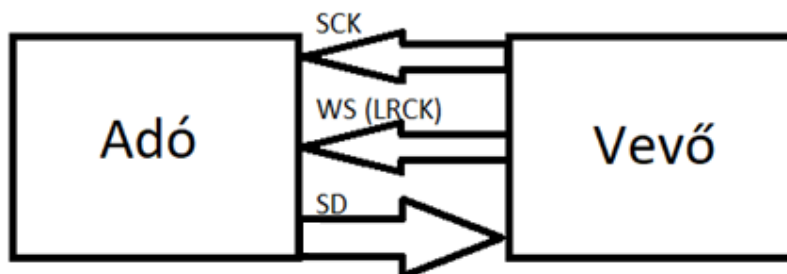
Az ötletet nem elvetem, csak átformálom és később ezt fejlesztem tovább:

- 8 bemenet helyett csak 2, hogy sztereó jelet mindenféleképpen digitalizálni lehessen.
- Az egyszerűség kedvéért először RTP helyett legyen UDP a szállító protokoll „nyersen”, kiegészítő protokoll vagy prioritáskezelés nélkül.
- Az AES67 szabvány a mérvadó, tehát cél a 10ms körüli maximális késleltetés, valamint a megfelelő mintavétel és bitráta.
- A vevő oldalon egyelőre bármi lehet, ami tud nyers, PCM audio adatot fogadni UDP-n keresztül.

3.1 A hardver

Legyen tehát az egész rendszer alapja egy Raspberry Pi 3B és kössünk hozzá egy A/D konvertert. A szabványokat követve olyan konverterre lesz szükség, amely 48kHz-es mintavételre képes 24-bites felbontással, mivel a legtöbb esetben a stúdiókban is ezt a párost használják. Opcionálisan képes legyen a mintavétel kétszeresére, 96kHz-re is. [\[7. forrás\]](#)

A választás a Cirrus Logic CS5343 típusú konverterére esett. Kommunikációs interfésze I²S (Inter-Ic Sound), amely egy háromvezetésű, kifejezetten áramköri részek közötti audio-kommunikációra kifejlesztett buszrendszer.



3. ábra I2S busz topológia

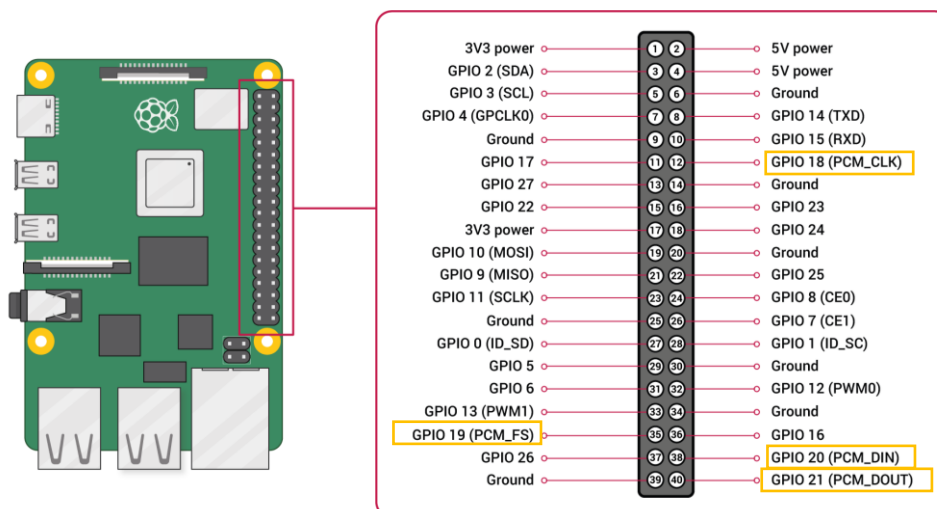
A Serial Clock (SCK) a működéshez szükséges órajel, a Word Select (WS), -amelyet olykor Left Right Clock (LRCK)-nak is hívnak- a bemeneti csatornák közötti váltásért felel és a mintavételi frekvenciával megegyező értéken dolgozik.

Ezt úgy kell elképzelni, hogy 48kHz-es mintavétel esetén azt határozza meg, hogy $\frac{1}{48000}$ másodpercig a bal, majd $\frac{1}{48000}$ másodpercig a jobb csatorna bitjei jelenjenek meg a Serial Data (SD) kimeneten. [\[8. forrás 2. oldal\]](#)

Az I²S eszközök között megkülönböztetünk *Master* és *Slave* változatot. Előbbi generálja az SCK és a WS órajeleket, hogy utóbbi ennek ütemében visszaküldhesse az SD jelet.

Nincs ez másképp a CS5343 esetében sem, habár ennél az IC-nél választható, hogy milyen üzemmódban dolgozzon.

A konverter Raspberry-hez történő kötése nem egy bonyolult dolog, mivel az I²S busznak vannak dedikált lábai, amelyek a 4. ábrán, sárgával jelölve láthatók.



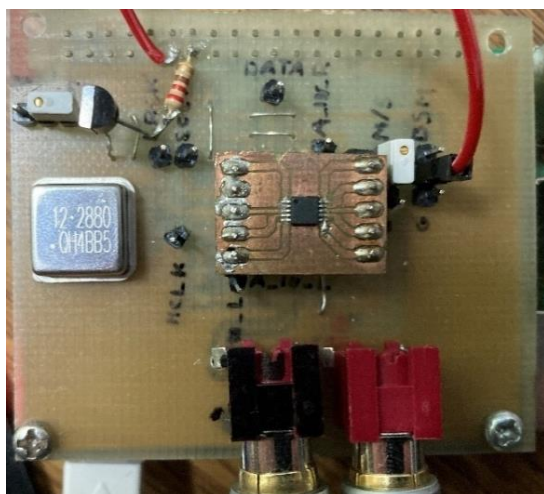
4. ábra Raspberry Pi 3 lábkiosztása

A PCM_CLK a Serial Clock, PCM_FS felel meg a WS-nek, valamint a PCM_DIN az adat bemenet, PCM_DOUT pedig az adat kimenet.

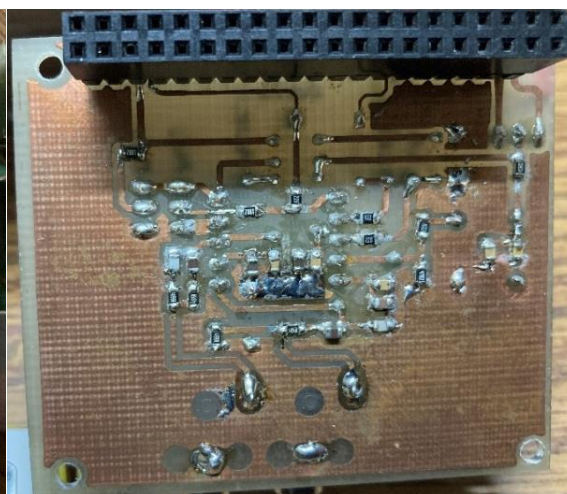
Természetesen ezen lábak összekötése a konverterrel önmagában nem elég, szükség van pár kiegészítő elemre is, nem beszélve a tápellátásról. A konverter adatlapján található egy ajánlás, amelyen a szükséges kiegészítő alkatrészek fel vannak tüntetve. Itt ábrázolják a tápfeszültséget beállító R és C tagokat, a funkciók kiválasztásához szükséges bekötéseket, valamint az analóg bemenet példahálózatát is. [\[9. forrás 11. és 15. oldal\]](#)

Az alkatrészek pontos kiválasztásához segítséget nyújt továbbá a *Cirrus Logic CDB5343: Evaluation Board for cs5343* dokumentációja. Ez egy A/D átalakító panel, mely az analóg hangot S/PDIF digitális bitfolyammá alakítja és a bemeneti oldalon a cs5343 konvertert használja. Megtalálható a dokumentumban a teljes kapcsolási rajz és az alkatrész lista, rajta a kondenzátorok és ellenállások pontos típusaival, valamint értékeivel. [\[10. forrás\]](#)

A fentebb említett ötletek és források alapján készült el egy prototípus panel a Raspberry Pi-re, melynek kapcsolási rajza a [7.1 mellékletben](#), a megvalósított változat pedig az 5. és a 6. ábrán látható.



5. ábra Prototípus panel felső oldala



6. ábra Prototípus panel alsó oldala

Az IC órajel lábai és a Serial Data lába ki vannak vezetve tesztponthoz, valamint maga a konverter is egy cserélhető adapteren helyezhető a panelen lévő foglalatba. Helyet kapott még kettő, jumperekkel ellátott háromlábú érintkezősor, amelyekkel a Master/Slave üzemmódot és a Double Speed/Single Speed módot¹ állíthatjuk be. A Master/Szolga üzemmódok közötti váltást úgy érhetjük el, ha az SDOUT lábat 10k Ω -os ellenállással felhúzzuk tápfeszültségre, vagy letesteljük. A Double és Single Speed módok közötti váltást pedig úgy, hogy az LRCK lábat 10k Ω -os ellenállással pozitív tápfeszültségre kötjük. [\[9. forrás 12. és 13. oldal\]](#)

Helyet kapott még egy 12.288MHz-es kristályoscillátor is a panelen, amely a fő órajelet (Master Clock) szolgáltatja, valamint választható, hogy a tápfeszültség 3.3V vagy 5V legyen, mivel a konverter mindkettőt támogatja. A kiegészítő alkatrészek zöme SMD alkatrész. A bemeneteket két darab RCA csatlakozón keresztül lehet fogadni.

¹ Single Speed módban 12.288MHz-es órajel mellett a mintavételi frekvencia 48kHz, Double Speed módban pedig 96kHz.

Az 5. ábrán látható utólagos kiegészítés a panelon. Egy vezeték és egy tranzisztor-ellenállás páros. Előbbire azért lett szükség, hogy szoftverből lehessen kapcsolni a váltást Double Speed Mode és Single Speed Mode között. A Raspberry 17-es GPIO lábára (fizikai 11-es láb) lett kötve egy vezeték, melyet a módváltó tükkesor középső érintkezőjére kell csatlakoztatni.

Mivel a konverternek nincsen külön RESET lába és az adatlap is azt állítja, hogy az ezen módok közötti váltás a bekapcsolási folyamat során történik, a tranzisztor egy kapcsolóként lett alkalmazva az áramkörben. A fő tápot szakítja meg a Raspberry 27-es GPIO lábáról (fizikai 13-as láb) kiadott jelhez mérten. Az egyszerű utólagos beépítés végett a tranzisztor a GND vonalat kapcsolgatja.

3.2 A szoftver

Miután eldőlt, hogy milyen konverter lesz alkalmazva és megépült a prototípus panel, következett a szoftvernek a fejlesztése. Ezzel kapcsolatban az elvárások a következők lettek:

- Mivel Real Time eszközről beszélünk, érdemes módosítani a Raspberry Pi kernelét és rátölteni az ún. PREEMPT-RT kernel patch-et. Ennek megléte mellett a mikrogép bizonyos feladatokat gyorsabban tud ellátni, mert -többek között- a folyamatok kezelése is másképp zajlik.
- Szükség lesz egy illesztőprogramra, amely lehetővé teszi, hogy az I²S buszra csatlakoztatott A/D konvertert bemeneti hangeszközként használhassuk.
- A fő szoftvert, amely majd veszi a hangot a konverter felől és továbbítja IP-n keresztül, a lehető legegyszerűbben kell megírni, hogy a késleltetés a lehető legalacsonyabb legyen. Emiatt, bár a Raspberry Pi-n népszerűbb a Python nyelv használata, a programot érdemes C-ben megírni.
- Ha mindez sikerült és a szoftver is hibátlanul működik, akkor kreálni kell egy webfelületet, ahonnan indítható lesz a program. Erre azért van szükség, hogy átlag felhasználás mellett ne kelljen közvetlen a Raspberry Pi termináljába SSH kapcsolaton keresztül belépni, majd hosszú parancsokat beírni.

3.2.1 A kernel és annak fontossága

A kernel az alapja egy számítógépes rendszernek, ami mondhatni hidat képez a hardver és az operációs rendszer között. Feladata többek között a memóriakezelés, folyamatok közötti kommunikáció, eszköz- és hálózatkezelés. Indításkor ez az alrendszer töltődik be először, tehát ha ez nem sikerül valami oknál fogva, akkor a számítógép nem fog elindulni. [\[11. forrás\]](#)

Ehhez képest egy Real Time Kernel, vagy akár egy Real Time Operációs Rendszer úgy lett kifejlesztve, hogy képes legyen időkritikus feladatokat kezelni, ahol nagyon fontos a gyors reakcióidő és a megbízhatóság, a stabilitás. Ebben az esetben milli- vagy mikroszekundumokról beszélünk, mint megengedhető késleltetés. A Linux PREEMPT-RT kernel csak annyiban különbözik az idáig felsoroltaktól, hogy nem külön rendszerként működik, hanem csak egy javítócsomag, amelyet egy már meglévő operációs rendszerbe tudunk integrálni, majd aktiválni. [\[12. forrás\]](#)

A Raspberry Pi-n a 4.14.91-es kernel van használatban. Nem volt szükség a legfrissebb kernelverzióra -ahogyan a legújabb Raspberry-re sem-, mivel viszonylag egyszerű feladatot kell végrehajtani, nem kötelező a konstans online kapcsolat és a frissítések sem. Így tehát az egyetlen szempont, ami figyelembe lett véve az a stabilitás. A 4.14.x verziójú kernelek elég megbízhatóak voltak, nem mellesleg sok olyan dokumentáció érhető el hozzájuk, ami elősegíti a PREEMPT-RT patch feltelepítését.

3.2.2 A/D konverter illesztőprogram: I²S busz kernelmodul

Egy kernelmodult -esetünkben- úgy lehet elképzelni, mint egy illesztőprogramot. Egy apró program, amely segít interakcióba lépni egy külső hardverrel. A Windows illesztőprogramokkal ellentétben egy ilyen modul betöltését követően nincsen szükség a rendszer újraindítására, a hardver azonnal használható lesz. Emellett tetszés szerint megszüntethető a futása is. Egy ilyen illesztőprogramnak az elkészítése, lefordítása hasonlít az egyszerű programokéhoz, azonban nagy különbség, hogy szükség van hozzá a kernel forráskódjára, mert a modulnak kapcsolatban kell állnia a kernel minden részével. [\[13. forrás\]](#)

Szerencsére a Raspberry Pi-hez léteznek olyan kernelmodulok, amelyek az I²S buszra csatlakozható eszközök használatát teszik lehetővé. Bár elsősorban nem A/D konverterhez valók, hanem ún. MEMS (micro-elektromechanical systems) mikrofonokhoz.

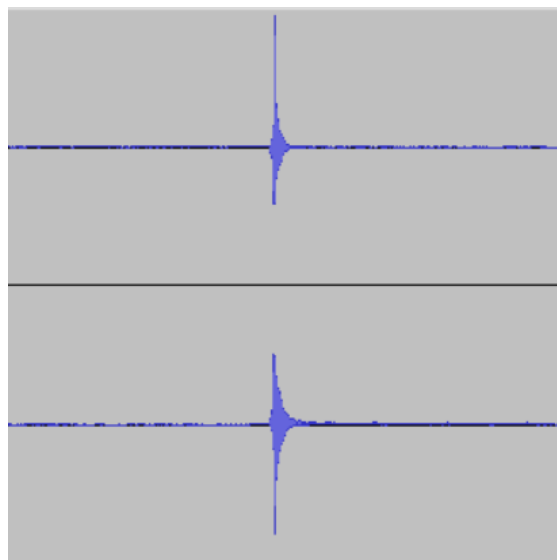
Ezek az apró mikrofonok szintúgy I²S buszt használnak a kommunikációhoz, azonban mivel a három vezetéken és a rajtuk futó órajeleken kívül semmi másra nincsen szükség (tehát nem eszközszerkezet specifikus kommunikációról beszélünk), egy ilyen kernelmodul az A/D konverterhez is megfelelő lesz.

Nicholas Johnson Github oldalán található egy leírás, amely alapján el lehet indulni a népszerű, Paul Creaser által 2016-ban írt kernelmodul kialakítása felé. Johnson elmagyarázza a modul fordításának menetét: [\[14. forrás\]](#)

- Első lépésként engedélyezzük az I²S buszt a Raspberry-n.
- Ezután beszerezzük a kernelnek a forráskódját és lefordítjuk.
- Lefordítjuk a kernelmodult is, majd betöltjük a kernelbe.

A modul elkészítése -a kernelforrás fordításának időigénye mellett is- egyszerű feladat, az illesztőprogram azonnal használható. Az ALSA² *arecord* parancsával el lehetett indítani egy tesztörögzítést és a hardver láthatóan működött is. Vagy mégsem?

Csatlakoztatva a konvertert a Raspberry-hez az első tapasztalat az, hogy bár működik az IC, Audacity-ben vizsgálva a rögzített anyagot a kimeneten véletlen időközönként pattogás jelentkezik.



7. ábra A pattogás "csendes" bemenet mellett is jelentkezik mindkét csatornán

Ez valamelyik órajel rendellenességére utal és bár oszcilloszkópon mérve minden rendben volt, sokáig megakadt a projekt ennél a pontnál.

Kicsit hosszabb idő elteltével, mikor egyszer a kernelmodul forráskódját olvastam, felfigyeltem egy parancsra, amely azt határozza meg, hogy az illesztőprogramot futtató eszköz Masterként, vagy Slave-ként üzemeljen. Idáig Master üzemmódban volt és így a Raspberry generálta az órajeleket is. Mivel a CS5343 képes üzemelni mind Slave mind Master módban, megcseréltem a két eszköz szerepét.

² Advanced *Linux* Sound Architecture: Ez felel a hangeszközökért Linux rendszerekben.

Így az A/D konverter generálta az órajeleket, a Raspberry pedig fogadta őket. Szerencsére ezen módosítás után megszűnt a pattogás és tiszta lett a digitalizált hang.

3.2.3 A Streamer Szoftver

Miután az illesztőprogram már tökéletesen működött és a konverter, mint bemeneti hangeszköz szabadon használhatóvá vált, következett a Streamer Szoftver megírása. Ezt két főbb részre kellett bontani:

- Írni kell egy olyan kódot, amely megnyitja ALSA-n keresztül az I²S eszközt és fogadni tudja a jelét.
- A konverterből érkező PCM jelet UDP fejléccel kell ellátni, majd a hálózaton továbbítani.

Első lépésként egy olyan kód született, amely megnyitja az A/D konvertert, fogadja az abból jövő mintákat és elmenti .RAW kiterjesztésű fájlba. Egy ilyen állomány nem rendelkezik fejléccel, mint például a .WAV, csak a nyers adatot tartalmazza, tehát visszajátszáskor pontosan meg kell adni a mintavételezési gyakoriságot és a bitmélységet.

Erre a kódra azért volt szükség, hogy meg lehessen győződni arról, hogy fogadható az IC-ből érkező PCM adatfolyam és visszajátszható mindenféle minőségromlás nélkül. A program megírásában első sorban az *Advanced Linux Sound Architecture (ALSA) project – the C library Reference* segített, melynek weboldalán dokumentálva van az összes ALSA függvény, amelyet C környezetben használni lehet.

Mihelyst a szoftvert sikerült megírni és Audacity-ben visszahallgatni a rögzített anyagot, következett a hálózati szekció megírása. A feladat az volt, hogy „csomagoljuk be UDP dobozokba” a PCM adatfolyam kereteket a pufferből és küldjük el egy IP címre egy adott porton keresztül, amit majd egy távoli számítógép fogad, kicsomagol és visszajátszik.

Kezdeképpen nagy segítséget nyújtott Brian Hall *Beej's Guide to Network Programming Using Internet Sockets (2024) című könyve*.

Ebben rengeteg információ lelhető fel a hálózati programozásról, sőt, sokkal kevesebb sorból is meg lehetett írni a Streamer Szoftverhez szükséges kiegészítést, mint amire számítottam.

A teljes kód a [7.2 mellékletben](#) található, ebben a fejezetben csak a fontosabb részei lesznek ismertetve. A program struktúrája a következő szekciókból épül fel (melyek a kódban jól elkülönítve láthatók):

- Bemeneti argumentumok kezelése: Felhasználó által megadott IP- és portcím, audio hardver azonosító, valamint mintavételi frekvencia és pufferméret fogadása. (**Input arguments**)
- AD konverter beállítása: GPIO portok inicializálása és a CS5343 sebességének, valamint feszültségének beállítása. (**AD converter setup**)
- Általános változók, mint az ALSA inicializáláshoz szükséges pointerok és memórafoglalás a puffereknek. (**Variables**)
- Hálózati inicializálás: UDP socket létrehozása, a felhasználótól bekért IP és portcím konvertálása, tárolása. (**Networking setup**)
- ALSA inicializálása: memórafoglalás, hozzáférés, bitráta, mintavételezési gyakoriság, pufferméret és csatornaszám beállítása, végül a hardver megnyitása. (**ALSA SETUP**)
- PCM audio pufferbe rögzítése majd hálózaton továbbítása. (**Capturing and sending audio data**)
- Memória takarítás és az AD konverter kikapcsolása. (**Cleanup and AD poweroff**)

Miután sikerült a kódot hibák nélkül lefordíttatani a *gcc compiler* segítségével, máris futtathatóvá vált a következő parancs beírásával:

```
./udp_streamer [cél IP cím][cél port cím] [hardver ID] [mintavételi frek.] [pufferméret]
```

A bemeneti argumentumokat a fentebb említett sorrendben megadva elindul a program és a megadott paramétereket vissza is jelzi a felhasználó felé. A Wireshark szoftverben a cél számítógépen a hálózati forgalmat vizsgálva megfigyelhetővé vált, mennyi UDP csomaggal árasztotta el a LAN-t a Raspberry.

3437...	582.386394163	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.386724852	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.387055430	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.387379085	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.387705595	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.391726914	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.392054948	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.392379114	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.397061409	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.397385284	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.397723957	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.398055758	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384
3437...	582.398372029	192.168.1.108	192.168.1.107	UDP	426 59625 → 8000	Len=384

Frame 343723: 426 bytes on wire (3408 bits), 426 bytes captured (3408 bits) on interface eno1, id 0	
Ethernet II, Src: Raspberr_6e:be:e1 (b8:27:eb:6e:be:e1), Dst: d0:ad:08:d2:e5:fa (d0:ad:08:d2:e5:fa)	
Internet Protocol Version 4, Src: 192.168.1.108, Dst: 192.168.1.107	
User Datagram Protocol, Src Port: 59625, Dst Port: 8000	
Data (384 bytes)	

0000	d0 ad 08 d2 e5 fa b8 27 eb 6e be e1 08 00 45 00	n.....E..
0010	01 9c 22 bc 40 00 11 92 73 c0 a0 01 0c c0 a0	...@...s...L...
0020	01 01 e9 1f 40 01 88 08 7b 56 56 20 49 d0 21	...@...h{V I..!
0030	21 44 20 ae 2f 20 02 3a 20 3e a9 1e 19 74 20 6f	ID / : : >...t o
0040	38 1f aa aa 22 dc 7b 1f c2 68 24 db b4 1e 21 d7	8..."-{...h\$...!
0050	28 8c de 1c 9b 46 2b 6d 32 1e 41 ed 2c 04 72 20	(...-F+m 2 A...r
0060	9b 0a 2e d9 1e 21 25 d4 2e f7 f2 21 8c 8b 2e e3	...-!%...!...!
0070	09 22 09 c4 2c 73 b8 21 4d 4c 2b 72 93 20 e3 4a	...s...! ML+r...J
0080	2a de 00 23 61 4c 2b 66 0a 23 e7 18 2b c7 ab 22	*...#aL+f...#...+...
0090	02 a1 29 7f 64 22 18 10 2a b7 df 24 a9 bb 2a 69	...)..d"...*...\$...i
00a0	c9 27 d4 8b 2b c6 55 28 29 d0 2a 40 6c 25 8f 46	)...+ U(...)*@!%F
00b0	29 e1 c9 24 54 74 27 87 41 25 39 99 26 03 f6 23	)...STt'...A%9...&...#
00c0	cf 8f 25 51 2e 23 b3 87 24 2a e5 20 dc e8 23 17	...%Q...#...\$*...-...#
00d0	ad 1f ff ba 22 5c 39 1e 34 56 1e 79 a4 1c 29 05	...-...'\9...4V-y...)
00e0	1d 2f 7f 1c 07 69 1e 5e 21 1e 04 40 1e fb f0 1c	/...-i-A !...@...-

8. ábra UDP csomagok vizsgálata Wireshark-ban

Láthatjuk a 8. ábrán, hogy a 192.168.1.108 IP címről a 8000-es porton keresztül érkeztek az UDP csomagok a 192.168.1.107-es IP című számítógépre. Méretüket a pufferméret határozza meg:

$$\text{Len} = [\text{puffer keretszám}] * [\text{bitek száma}] * [\text{csatornaszám}]$$

A 8. ábra készítésekor a pufferméret 64 keret volt. Tudjuk, hogy 24-bites (3 bájtos) mintákat továbbítunk két csatornán, tehát a csomag mérete:

$$\text{Len} = 64 * 3 * 2 = 384 \text{ bájt.}$$

Végezetül egy dolog maradt hátra, valahogyan vissza kell alakítani a csomagokat PCM adatfolyammá, majd le kell játszani azt a céleszköz hangszóróin. Erre a feladatra a leghasznosabb megoldás egy saját fogadóprogram elkészítése volna, de a dolgozat célja bemutatni a küldőt, ezért már előre elkészített szoftverek is megfelelők lesznek. Ettől függetlenül nem kizárt, hogy a jövőben vevőszoftver is lesz hozzá írva.

A választott szoftver a *Socat* és a *Sox* (*Sound eXchange*) Linux alapú programok pipe módban.

A futtató parancs:

```
socat -u UDP-LISTEN:8000,reuseaddr,fork - | sox -t raw -r 48000 -e signed -b 24 -L -c 2 - -d --buffer=128
```

A *Socat* hallgatja a 8000-es portot, valamint, ha megszűnik rajta az adatfolyam, majd újraindul, akkor újracsatlakozik. Első próbálkozásakor *Socat* helyett a *Netcat* szoftvert használva feltűnt, hogy az nem képes az újra felépült kapcsolathoz visszacsatlakozni.

A *Sox* fogadja a *Socat* által dekódolt csomagokat, és ahogyan a parancsból is látható, előre meg kellett adni bizonyos paramétereket:

A program nyers (raw), 48kHz-es mintavételű, S24LE³ bitmélységű, kétcsatornás adatfolyamot fog kapni. A vevőoldali pufferméret 128 keret, mely semmiképpen sem lehet kisebb, mint a küldő oldali puffer mérete.

A 9. és 10. ábrákon látható a küldő és a fogadó oldali Terminál ablak:

³ Signed 24-bit Little Endian (előjeles 24-bites növekvő bájtrendű)

```

raspi@Raspi:~/i2s/Software $ ./udp_streamer 192.168.1.107 8000 hw:1,0 48000 64
IP address: 192.168.1.107
Destination port: 8000
Capture device: hw:1,0
Sample rate: 48000 Hz
Buffer size: 64 frames

Streaming...

```

9. ábra Terminál a Raspberry Pi oldalon

```

levi@levi-HP-255:/media/levi/Logic/!MEGA/!Kandós/Szakdolgozat/Software/Receive$
socat -u UDP-LISTEN:8000,reuseaddr,fork - | sox -t raw -r 48000 -e signed -b 24
-L -c 2 - -d --buffer=128
sox WARN alsa: can't encode 0-bit Unknown or not applicable

-: (raw)

File Size: 0
Encoding: Signed PCM
Channels: 2 @ 24-bit
Samplerate: 48000Hz
Replaygain: off
Duration: unknown

In:0.00% 00:00:20.11 [00:00:00.00] Out:965k [====|=====] Hd:1.9 Clip:0 so
x WARN alsa: under-run
In:0.00% 00:00:20.82 [00:00:00.00] Out:999k [ -====|===== ] Hd:5.8 Clip:0 so
x WARN alsa: under-run

```

10. ábra Terminál a vevőoldalon

Futás közben a hang tisztán hallható, nem torz. Az *under-run* üzenetek, amelyek néha megjelennek azok a puffer nem megfelelően beállított méretéből, a csomagok elvesztéséből, vagy sorrendjeik felcserélődéséből adódhatnak. Legtöbb esetben a küldő és fogadó oldali puffer méretének módosításával kiküszöbölhető a hibajelenség, de gyakoriságát nagyban meghatározza az is, milyen forgalmak vannak egy időben a hálózaton. A küldés-fogadás kábeles kapcsolaton stabil, ugyanakkor vezeték nélküli összeköttetés esetén a késés megnövekszik, valamint a csomagvesztés is gyakoribb.

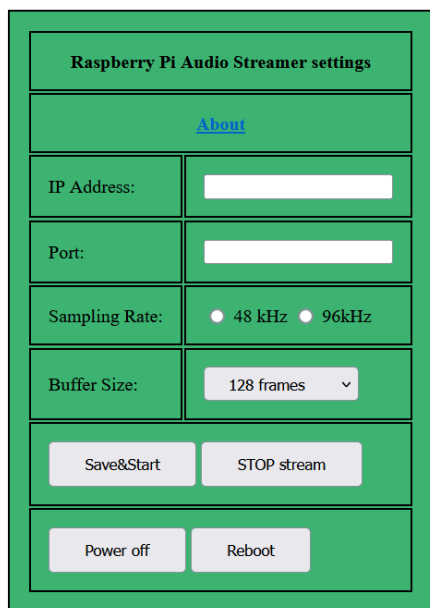
3.2.4 A Raspberry Pi webfelülete

Ahogy az már pár fejezettel korábban említésre került, egy weboldalra azért van szükség, hogy a Streamer Szoftver beállításait egy felhasználóbarát felületről el lehessen érni. A Raspberry-n fut egy *Apache* webservert, ezen keresztül érhető el az oldal, ha webböngészőbe beírjuk a mikrogép IP címét.

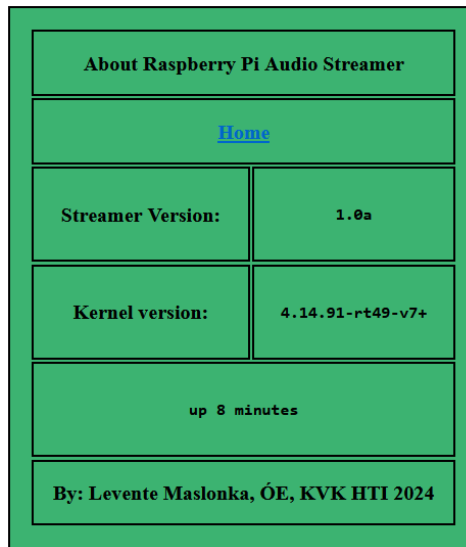
A weboldallal szemben támasztott követelmények:

- Egyszerű felület, nincsen szükség komoly dizájnra.
- Mobileszközön is jól látható legyen.
- Képes legyen külső programokat, parancsokat futtatni.

Egy egyszerű HTML weboldal nem lesz elég, ha külső programokat akarunk megnyitni, viszont a PHP nyelvben van rá függvény, nevezetesen az *exec()* és a *shell_exec()* függvények. Így tehát az oldal nagyrészt PHP nyelven íródott.



11. ábra A webfelület nyitóoldala



12. ábra A webfelület About oldala

A 11. ábrán láthatjuk a nyitóoldalt (*Home*), amelyen meg lehet adni a vevő eszköz IP címét, a portot, amelyen az UDP csomagok küldve lesznek, valamint a mintavételi frekvenciát és a pufferméretet is itt választhatjuk ki. A *Save&Start* és *STOP* stream gombok a program elindításáért, még a *Power off* és *Reboot* gombok a Raspberry Pi leállításáért és újraindításáért felelnek.

A lap tetején kiválaszthatjuk az *About* menüpontot is, ilyenkor átkerülünk egy másik oldalra, amelyen megnézhetjük a Streamer Szoftver és a kernel verzióját, valamint a bekapcsolás óta eltelt időt.

A weblapnak a formáját egy táblázat adja, így mobileszközről nézve sem esik szét a tartalom. A funkciógombok mögött PHP szkriptek vannak, amelyek a gomb lenyomására aktiválódnak.

A weboldal elkészítésében a legtöbb fejtörést igénylő feladat nem maga a felület megalkotása volt, hanem rájönni, hogy miért nem lehet PHP szkriptekből külső parancsokat vagy programokat indítani. A megoldás az volt, hogy a Raspberry rendszerében hozzáférést kellett adni a *www-data*⁴ felhasználónak bizonyos parancsokhoz, hogy ezek az Apache szerveren keresztül futtathatóak legyenek. Amint ez megtörtént, minden parancs hibátlanul futtathatóvá vált.

⁴ Az Apache szerver ezen a felhasználón keresztül lép kapcsolatba a Linux rendszerrel

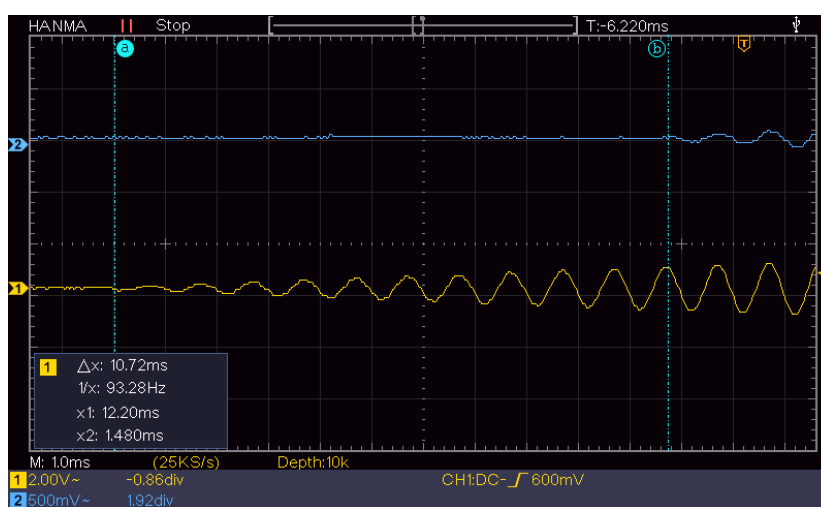
3.3 A késleltetés mérése és kiértékelése

Az eszköz megépítését és beállítását követően az első és legfontosabb kérdés, hogy mennyi a késleltetés?

A mérést úgy állítottam össze, hogy a Yamaha 02R keverőpultból egy -12dB értékű, 1kHz-es, szinuszos mérőjelet kapcsoltam a Raspberry-n lévő konverterre, melynek UDP csomagjait a HP 255 g9 laptopomon futó *SOCAT* és *SOX* programok fogadtak. Oszilloszkópon (Hanmatek DOS1102) figyeltem a CH1(keverőpult kimenet) és CH2(vevő laptop audio kimenet) csatornákat először 500ms/DIV horizontális eltérítéssel mérve. Bekapcsoltam a mérőjelet és amint megjelent a műszeren, megállítottam a futását. Ezután belenagyítva a jel kezdetéhez, összevetettem a két csatorna legelső jelváltozásai között eltelt időt.

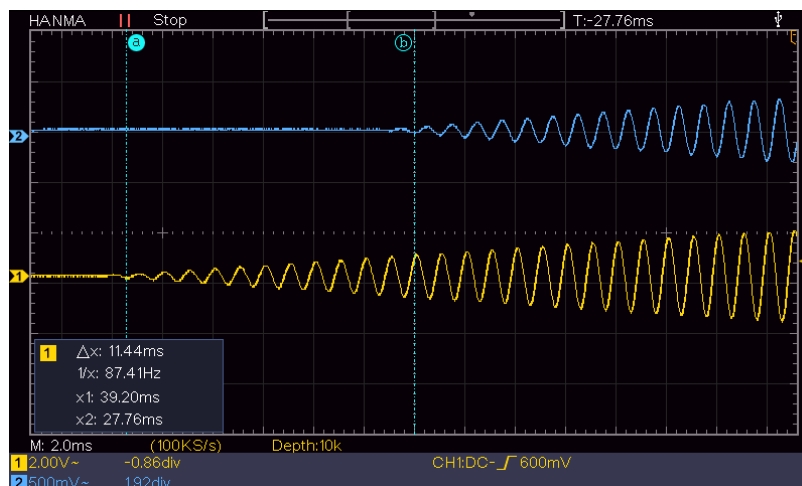
Különböző paraméterekkel indítva a Streamer Szoftvert a következő eredmények születtek:

- A legrosszabb esetben 30ms körüli késleltetést lehetett mérni.
- A legjobb esetben pedig 10.64ms-ot sikerült, amely annyira jónak bizonyult, hogy még egyszer megismétltem. Ekkor 10.72ms lett.



13. ábra Az egyik legalacsonyabb mért késleltetés

Ilyen alacsony késleltetést 64 frame nagyságú puffer mellett sikerült elérni, de ilyen kicsi méretűnél már előfordul egy-két csomagvesztés. A legstabilabb akkor a rendszer, hogyha 128, vagy 256 frame a puffer nagysága. Ilyenkor a késés leginkább 11ms és 13ms között mozog. A 14. ábrán egy olyan mérésnek az eredményét látjuk, ahol a pufferméret 256 frame.



14. ábra 256 frame nagyságú puffernél a késleltetés 11.44ms

A mintavételi frekvenciától egyébként független a késleltetés. A mérések 48kHz-es mintavételezési gyakoriságnál lettek elvégezve, de 96kHz-en is hasonló eredmények születnek. Ebben az esetben csak az adatsebesség növekszik meg.

A késleltetésről tehát elmondható, hogy sajnos nem sikerült 10ms alatti értéket tartani, magyarul a szabvány szerint nem tekinthető valós idejűnek az átvitel. Ezzel szemben elég közel van hozzá, tehát meglehet, hogy a jövőben a továbbfejlesztések során sikerül még rövidíteni a latency idejét.

4. Továbbfejlesztési lehetőségek

Az eszköz megépítésének előrehaladtával egyre több és több bővítési lehetőség realizálódott. Amint bekerül a hangforrás jele a digitális világba, gyakorlatilag bármit csinálhatunk vele, legyen szó akár a jel feldolgozásáról (hangerő, hangszín, dinamika módosítása) akár a továbbítás módjáról. A felmerült ötleteket két csoportba lehet sorolni. Egy egyszerűen kivitelezhető, és egy „nagyobb kihívás” kategóriába.

Az egyszerűbb bővítések olyanok, mint például a:

- Digitális bemenet – Létezik a Cirrus Logic-nak egy olyan konvertere (CS8416), amely képes fogadni AES3 vagy S/PDIF digitális hangforrásokat, majd I²S buszon továbbítani. Bár ez egy összetettebb IC, de standalone⁵ módban is képes üzemelni, szimpla jelfordítóként. Egy ilyen konverter illesztésével digitális bemenetet lehet adni az eszköznek és innentől kezdve keverőpultok AES/EBU kimeneteit is rá lehet kötni.

⁵ Magától működő

- Szoftver optimalizálás – A Raspberry Pi rendszerét érdemes lehet átdolgozni úgy, hogy a lehető legkevesebb háttérfolyamat fusson rajta, erőforrást felszabadítva így a Streamer Szoftver számára. Plusz segítség lenne írni egy apró programot, ami figyeli az egyik GPIO lábat és ha állapotváltozás megy végbe rajta, indítsa újra a rendszert. Mindezt azért, hogyha valami baj történik menet közben, azonnal újra lehessen indítani.
- Webfelület optimalizálás – Az előző fejezetben említett statikus IP megadásához létre lehetne hozni egy külön fület, valamint az oldal a megnyitásakor elsőként kérhetne felhasználónevet és jelszót a látogatótól. Bár egy ilyen egyszerű eszköz nincsen kitéve a támadásoknak, főleg egy olyan elkülönített hálózaton, ahol csak a Raspberry és egy fogadóeszköz van, a felhasználónév és jelszó páros sosem árt.

A bonyolultabb bővítési ötletek szoftveres oldalról:

- Az UDP-t kibővíteni RTP-re és megoldani egy pontos szinkronizációt az RTCP és PTP protokollok segítségével.
- Quality of Service (QoS) implementálása.
- Multicast, hogy ne csak pont-pont kapcsolatot lehessen létesíteni.

Ezek segítségével a késleltetés is csökkenthető lenne. Bár az a legjobb, minél alacsonyabb a latency, ha már 10ms alatt lehet tartani, akkor kiindulva az AES67 szabványból, valós idejűnek lehetne tekinteni.

Hardveres oldalról a nagyobb kihívást jelentő bővítés az a több konverter illesztése volna. Bár az I²S busz alapján véve két csatornát tud kezelni, lehetséges több I²S eszköz egy rendszerbe kötése Time Division Multiplex (TDM) segítségével. Ez azt jelenti, hogy minden eszköz minden órajelet egy helyről kap meg, de frekvenciáik más arányban vannak elosztva. [\[17. forrás\]](#)

5. Összegzés

Összegzésképp elmondható, hogy az IP alapú hangátvitel, bár már elterjedt technológia, hobbiszinten még nem olyan népszerű, mint például egy Raspberry Pi alapú CNC eszterga vezérlő. Ettől függetlenül érdekes téma, rengeteg lehetőséggel és kihívással a háta mögött.

Az általam megépített eszköz, bár késleltetési értékét figyelve épphogy nem mondható valós idejűnek, jelenlegi formájában is jól működik. A felsorolt bővítési lehetőségeket is számításba véve egy jó alap, melyre a későbbiekben is érdemes építkezni.

A meglévő küldő eszköz mellé egy fogadó megépítése is tervben van, szintén Raspberry Pi alapon, I²S buszra csatlakoztatható D/A konverterrel kiegészülve. Itt pedig természetesen megint minden eddig felsorolt ötletet számításba lehet venni, csak „visszafelé”. Analóg kimenet mellé digitális, egy barátságos webfelület, kettőnél több kimeneti csatorna stb.

Felhasználási területe pedig lehet bármi az egyszerű háztartásbeli audio-átviteltől a komolyabb stúdió célokig. Amint bekerül az IP hálózatba a hanganyag, bárhová elküldhető. Jó alapja lehet akár egy olyan internet rádióknak is, ahol a bejátszások, keverések hardveren történnek és nem adáslebonyolító szoftveren keresztül. Itt ráadásul nincs is szükség annyira alacsony késleltetésre.

Kiegészülve a digitális bemeneti IC-vel akár egy modern bővítése is lehet a Kandó Stúdióban található Studer D950 típusú keverőpultnak. Ennek konverter keretébe kártya formájában beépítve és párosítva egy AESO kártyával (melyről megkaphatja a digitális audio-t) hálózatra lehetne kötni a keverőpultot és esetleg a majdan épülő videóstúdióban lévő rendszerre is csatlakoztatni lehetne.

A mai világban már minden hálózathoz köthető, így jogosan merül fel a kérdés az olvasóban, hogy mennyire lesz biztonságos a többi eszköz mellett szállítani a hangcsomagokat. Gyakorlatilag, ha elkülönített vonalat használunk, külön alhálózatot, melyen csak a küldő és a fogadó eszköz kommunikálnak és ezen végpontokról az Internet nem érhető el, akkor nagy problémákba nem ütközhetünk.

Az pedig, hogy mennyire lesz időtálló, arra a legegyszerűbb válasz az, hogy az eszköz digitális. Amint egyesekről és nullákról beszélünk, azok később is manipulálhatóak lesznek a már megalkotott szabványok alapján. Ezek pedig nem változnak, legfeljebb átalakulnak. Be- és kimeneti eszközökre viszont még akkor is szükség lesz, ha már mindent felvált az IP és mikor egy keverőpulton már nem is látunk mást, csak egy magányos RJ45-ös csatlakozót.

A fejlesztés természetesen nem marad félbe, a jövőben szeretném a felsorolt bővítéseket hozzáépíteni az eszközhöz.

6. Források

6.1 Irodalomjegyzék

1. Cristopher Holder (2009): *Dante is the new audio networking name on everybody's lips. But why is Dante so hot?* -Letöltve: 2024.10.28.
https://web.archive.org/web/20090915233935/http://www.av.net.au/contents/issue_6/audinate_dante.pdf
2. Steve Church & Skip Pizzi (2010): *Audio over IP: Building Pro AoIP Systems with LiveWire* Oxford, Focal Press, ISBN: 978-0-240-81244-1, Letöltve: 2024.10.28.
http://pws.npru.ac.th/sarthong/data/files/Audio_Over_IP_Building_Pro_AoIP_Systems_with_Livewire.pdf
3. Lukovszki Tamás (2007): *Számítógépes hálózatok* -Letöltve: 2023.10.29.
<https://people.inf.elte.hu/lukovszki/Courses/0708NWI/nw110s4.pdf>
4. Andreas Hildebrand (2021): *AES67 & SMPTE ST 2110: Transport & Synchronization* -Letöltve: 2024.10.28.
<https://www.ravenna-network.com/wp-content/uploads/2021/11/AoIP-Transport-Synchronization-Fundamentals-in-a-Nutshell.pdf>
5. ALC NetworX -Ravenna: AES67 AND RAVENNA IN A NUTSHELL – Letöltve: 2024.10.28.
<https://ravenna-network.com/wp-content/uploads/2020/02/AES67-and-RAVENNA-in-a-nutshell.pdf>
6. Clear-Com (2023): *AoIP Network Recommendations* -Letöltve: 2024.10.28.
https://clearcom.com/DownloadCenter/technicaldocs/AES67SwitchRecommendations/AoIP-AES67_Network_Recommendations.pdf
7. Griffin Brown (2021): *Digital Audio Basics: Audio Sample Rate and Bit Depth* -Letöltve: 2024.11.01.
<https://www.izotope.com/en/learn/digital-audio-basics-sample-rate-and-bit-depth.html>

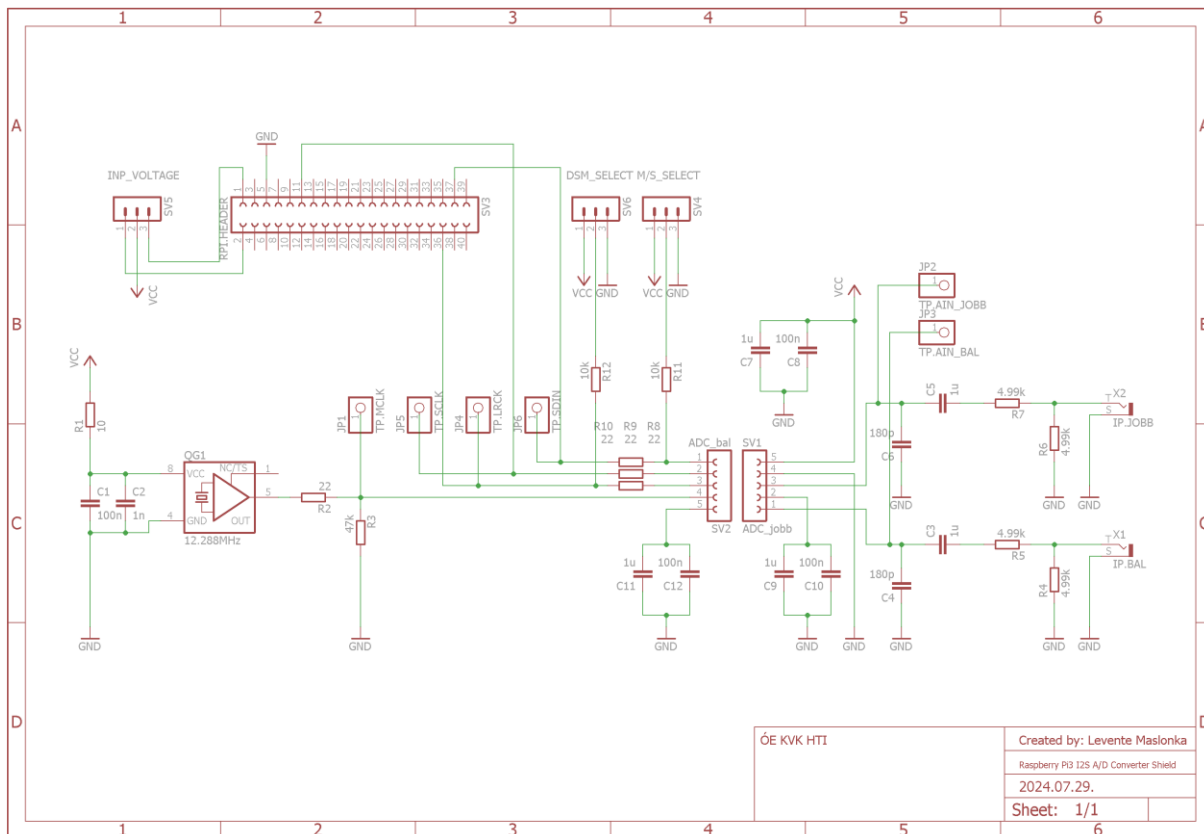
8. Cypress Semiconductor Corporation (2012): Inter-IC Sound Bus (I2S) 2.30 – *Letöltve:* 2024.11.01.
https://www.infineon.com/dgdl/Infineon-Component_I2S_V2.30-Software%20Module%20Datasheets-v02_07-EN.pdf?fileId=8ac78c8c7d0d8da4017d0ea35b2125c8
9. Cirrus Logic (2015): CS5343/5344 datasheet -*Letöltve:* 2024.11.01.
https://www.mouser.com/datasheet/2/76/CS5343-44_F5-1142054.pdf
10. Cirrus Logic (2006): CDB5343: Evaluation Board for cs5343 -*Letöltve:* 2024.11.01.
https://statics.cirrus.com/pubs/rdDatasheet/CS5343_DB2.pdf
11. Alexander S. Gillis, Stephen J. Bigelow, Jessica Lulka (2024): What is a Kernel? -
Letöltve: 2024.11.02.
<https://www.techtarget.com/searchdatacenter/definition/kernel>
12. Steven Vaughan-Nichols (2024): 20 years later, real-time Linux makes it to the kernel – really -*Letöltve:* 2024.11.02.
<https://www.zdnet.com/article/20-years-later-real-time-linux-makes-it-to-the-kernel-really/>
13. Dinesh Achari (2022): Using Loadable Kernel Modules in Linux – *Letöltve:* 2024.11.05.
<https://www.linuxfordevices.com/tutorials/linux/loadable-kernel-modules-linux>
14. Nicholas Johnson, Drew Gates, Glenn Fernandes (2016): Using the ICS43432 MEMS microphone on a Raspberry Pi with i2s – *Letöltve:* 2024.11.05.
<https://github.com/nejohnson2/rpi-i2s>
15. Advanced Linux Sound Architecture (ALSA) project – the C library reference -
Letöltve: 2024.09.27.
<https://www.alsa-project.org/alsa-doc/alsa-lib/index.html>
16. Brian “Beej Jorgensen” Hall (2024): Beej’s Guide to Network Programming v3.1.12 -
Letöltve: 2024.10.06
https://beej.us/guide/bgnet/pdf/bgnet_a4_c_1.pdf
17. Jerad Lewis, Analog Devices, Inc. (2012): Common Inter-IC Digital Interfaces for Audio Data Transfer – *Letöltve:* 2024.11.04.
<https://www.analog.com/media/en/technical-documentation/technical-articles/ms-2275.pdf>

6.2 Ábrajegyzék

1. https://www.researchgate.net/figure/RTP-packet-internal-structure-f2_fig2_307589012
2. A hálózat általános blokkvázlata
3. I2S busz topológia
4. <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#raspberry-pi-3-model-b>
5. A prototípus panel felső oldala
6. A prototípus panel alsó oldala
7. A pattogás "csendes" bemenet mellett is jelentkezik mindkét csatornán (Vizsgálva Audacity 3.5.1-ben)
8. UDP csomagok vizsgálata Wireshark-ban
9. Terminál a Raspberry Pi oldalon
10. Terminál a vevőoldalon
11. A webfelület nyitóoldala
12. A webfelület About oldala
13. Az egyik legalacsonyabb mért késleltetés
14. 256 frame nagyságú puffernél a késleltetés 11.44ms

7. Melléklet

7.1 A prototípus panel kapcsolási rajza



7.2 A Streamer Szoftver forráskódja

/*

File name: udp_streamer.c

Date: 2024.10.26.

NETWORK AUDIO STREAMER SOFTWARE for Raspberry Pi and CS5343 A/D converter

CREATED BY: Levente Maslonka, ÖE KVK HTI 2024

This code takes the input of an audio device found in the ALSA hardware list and organizes them into UDP packets, then sends it to a destination IP address on a specified port.

Additional functions of the program:

-Input arguments: on startup, user must specify the destination IP and port address, the id of the hardware wanted to be used, and the sample rate and buffer size. The last two are arbitrary, if not specified, default values of 48kHz and 512 frames will be used.

If the only input argument is "version", the software prints out it's version and exits

-GPIO controls: The power of the A/D converter is controlled via GPIO 27 through a transistor
The speed mode of the A/D is controlled via GPIO 17.

-The device used in this code is "hw:1,0" but it's arbitrary.

-As so the sampling rate, bitrate and buffer size.

reference: Alsa library documentation -Downloaded: 2024.09.27.

-https://www.alsa-project.org/alsa-doc/alsa-lib/group__p_c_m__h_w__params.html
-https://www.alsa-project.org/alsa-doc/alsa-lib/group__p_c_m.html

2024.10.06. Brian Hall(2024): Beej's Guide to Network Programming v3.1.12 -Downloaded:
https://beej.us/guide/bgnet/pdf/bgnet_a4_c_1.pdf

```
*/
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>

#include <wiringPi.h>

#define DEFAULT_SR 48000                       //default sample rate
#define CHANNELS 2                            // channel count [FIX!!!]
#define FORMAT_SND_PCM_FORMAT_S32_LE        // bit depth (32 bit signed, little endian) [FIX!!!]
#define DEFAULT_BS 512                        // default buffer size (frames)

#define SPEED_MODE 17                         //       pin number 11 on the Pi3
#define AD_POWER 27                           //       pin number 13 on the Pi3

int main(int argc, char* argv[]) {

//-----input arguments-----

if(strcmp(argv[1],"version") == 0) {printf("1.0a\n"); return 0;}

if (argc < 4 || argc > 7)
{
    printf("Usage: ./udp_streamer [DESTINATION IP] [DESTINATION PORT] [SOURCE AUDIO DEVICE\n(hw:x,x)]\n\n");
    printf("Other options:\n[Sampling rate] (48000 / 96000) --> sets the sampling rate to either\n48kHz or 96kHz\n");
    printf("[Buffer size] (64-1024) sets the buffer size in frames\n\n");

    printf("Example: ./udp_capture 192.168.1.100 80 hw:0,0 48000 512\n");
    printf("Sends the stream to 192.168.1.100:80\nwith 48kHz sampling rate and 512 frames of\nbuffer.\n\n");
    printf("Sampling rate and buffer is optional,\nif not specified, the default is 48Khz and 512\nframes!\n");

    return 1;
}

unsigned int SAMPLE_RATE = (argc >= 5) ? atoi(argv[4]) : DEFAULT_SR;
int BUFFER_SIZE = (argc == 6) ? atoi(argv[5]) : DEFAULT_BS;

//-----AD converter Setup-----

    //Initializing the wiringPi library and so the GPIO ports
wiringPiSetupGpio();
pinMode(SPEED_MODE, OUTPUT);
pinMode(AD_POWER, OUTPUT);

    //Setting the sample rate pin high or low according to the smple rate itself
if(SAMPLE_RATE == 48000) {digitalWrite(SPEED_MODE, LOW);}
else if (SAMPLE_RATE == 96000) {digitalWrite(SPEED_MODE, HIGH);}

    //First powering the AD converter off, waiting a few seconds, then powering the AD converter on
//It's because of the AD converter only swithches the sample rate if it is restarted. It has no
reset pin.
digitalWrite(AD_POWER, LOW);
sleep(2); //wait for 2 seconds
digitalWrite(AD_POWER, HIGH);
sleep(1); //wait for 1 second afer powering the AD on
```

```
//-----Variables-----

uint32_t rc;
snd_pcm_t *capture_handle;
snd_pcm_hw_params_t *hw_params;
size_t bufferSize = BUFFER_SIZE * CHANNELS * 4; //buffer size in bytes [buffer * channel count * 4]
(32-bits = 4 bytes)
unsigned char *buffer = malloc(bufferSize); //dynamically specify memory allocation

//memory allocation for 24-bit samples. Conversion from 32-bit to 24-bit is needed to keep the
foundations of the AES67 standard
unsigned char *buffer_24bit = malloc(BUFFER_SIZE * CHANNELS * 3); //buffer * channel count * 3 (24-
bits = 3 bytes)

//-----Networking setup-----

int udp_socket;
struct sockaddr_in dest_addr; //structure to hold the destination address

//Check the format of the IP address
if (inet_pton(AF_INET, argv[1], &dest_addr) != 1) {
    printf("Invalid IP address format\n");
    return 1;
}

//Check if the port is in the correct range (the port number is 16-bits wide)
if (atoi(argv[2]) <= 0 || atoi(argv[2]) > 65535) {
    printf("Invalid port number\n");
    return 1;
}

//Destination address structure
memset(&dest_addr, 0, sizeof(dest_addr));
dest_addr.sin_family = AF_INET; //The destination address will be IPv4
dest_addr.sin_port = htons(atoi(argv[2])); //Port number from host byte order to network byte order
inet_pton(AF_INET, argv[1], &dest_addr.sin_addr); //Storing the IP address in the structure

//UDP socket
udp_socket = socket(AF_INET, SOCK_DGRAM, 0); //Creating a socket: IPv4 address, DATAGRAM, Internet
Protocol
if (udp_socket < 0) {
    perror("Error creating socket");
    return EXIT_FAILURE;
}

//----- ALSA SETUP-----

// Opening the device given as the 3rd argument for capture
rc = snd_pcm_open(&capture_handle, argv[3], SND_PCM_STREAM_CAPTURE, 0);
if (rc < 0) {
    fprintf(stderr, "Could not open audio device!\n"); //standard error: can't open the device
    return EXIT_FAILURE;
}

// ALSA setup: hardware parameters
snd_pcm_hw_params_alloca(&hw_params); //allocate memory for hardware parameters
snd_pcm_hw_params_any(capture_handle, hw_params); //initialize the parameters
snd_pcm_hw_params_set_access(capture_handle, hw_params, SND_PCM_ACCESS_RW_INTERLEAVED); //access
type interleaved: one L channel frame, one R channel frame
snd_pcm_hw_params_set_format(capture_handle, hw_params, FORMAT); //setting up the bitrate
snd_pcm_hw_params_set_rate(capture_handle, hw_params, SAMPLE_RATE, 0); //setting up the sample rate
snd_pcm_hw_params_set_channels(capture_handle, hw_params, CHANNELS); //setting up the channels

// Apply hardware parameters
rc = snd_pcm_hw_params(capture_handle, hw_params);
if (rc < 0) {
    fprintf(stderr, "Unable to set hardware parameters!\n");
    return EXIT_FAILURE;
}
}
```

```
//-----Capturing and sending audio data-----

    printf("IP address: %s\nDestination port: %s\nCapture device: %s\nSample rate: %d Hz\nBuffer size: %d frames\n",argv[1],argv[2],argv[3],SAMPLE_RATE,BUFFER_SIZE);
    printf("\nStreaming...\n");

    while (1) {

        rc = snd_pcm_readi(capture_handle, buffer, BUFFER_SIZE); //reading from the audio device to the buffer

        if (rc == -EPIPE) {
            fprintf(stderr, "Overrun!!!\n");
            snd_pcm_prepare(capture_handle);
        }

        else if (rc < 0) {
            fprintf(stderr, "Audio device fault!\n");
        }

        else if (rc != BUFFER_SIZE) {
            fprintf(stderr, "Short read!\n");
        }

        //chopping off the upper 8-bits from the 32-bit samples and making them 24-bit pnes
        for (int i = 0; i < BUFFER_SIZE * CHANNELS; i++) {
            memcpy(&buffer_24bit[i * 3], &buffer[i * 4 + 1], 3);
        }

        //send audio data: the udp socket, from the 24-bit buffer, how many bytes to send in the packet (BUFFER_SIZE * CHANNELS * 3), no special flags (0), address of the target, how many bytes to read from memory
        ssize_t sent_bytes = sendto(udp_socket, buffer_24bit, BUFFER_SIZE * CHANNELS * 3, 0, (struct sockaddr*)&dest_addr, sizeof(dest_addr));

        if (sent_bytes < 0) {
            perror("Error sending data!");
        }
        usleep(10); //delay
    }

//-----Cleanup and AD poweroff-----

    snd_pcm_close(capture_handle);
    free(buffer);
    free(buffer_24bit);
    close(udp_socket);
    digitalWrite(AD_POWER, LOW);
    return 0;
}
```