VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VIZUALIZACE PARAMETRŮ MNOHAKANÁLOVÉHO ZVUKOVÉHO SYSTÉMU V INTERNETOVÉM PROHLÍŽEČI

PARAMETER VISUALIZATION OF MULTICHANNEL AUDIO SYSTEM IN WEB BROWSER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN LACH**
AUTHOR

VEDOUCÍ PRÁCE **Ing. IGOR SZŐKE, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Lach Martin, Bc.**
Program: Informační technologie Obor: Počítačové a vestavěné systémy
Název: **Vizualizace parametrů mnohakanálového zvukového systému v internetovém prohlížeči**
Parameter Visualization of Multichannel Audio System in Web Browser
Kategorie: Softwarové inženýrství
Zadání:

1. Seznamte se s dodaným HW a SW pro vícekanálové zpracování audia.
2. Navrhněte strukturu a protokoly pro přenos vizualizačních a řídicích dat v reálném čase.
3. Realizujte programové části embedded systému back endu v Linuxu pro posílání dat do front endu.
4. Realizujte vizualizační a řídicí rozhraní v internetovém prohlížeči. Otestujte stabilitu navrženého řešení.
5. Vyhodnoťte latence, časové a paměťové nároky. Zhodnoťte dosažené výsledky.
6. Vyrobte A2 plakátek a cca 30 vteřinové video prezentující výsledky vaší práce.

Literatura:

- Dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2 a část bodu 3 a 4 ze zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Szőke Igor, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Tato práce se zabývá systémem na zpracování audia od firmy Audified. Tento vestavěný systém obsahuje Arm procesor, na kterém běží operační systém Linux. V současné době je ovládání parametrů (fantom, gain) komplikované – bez zpětné vazby. V této práci je popsaná tvorba server–klient aplikace, která umožní snadné nastavení zmíněných parametrů a rovnou zobrazí jejich efekt.

Abstract

This work deals with Audified Audio Processing System. This embedded system includes an Arm processor running the Linux operating system. At present, parameter control (phantom, gain) is complicated, without feedback. In this work, the creation of server - client application, which will allow easy setting of the mentioned parameters and will show their effect, is described.

Klíčová slova

zpracování zvuku, Arm, embeded system, Linux, klient, server, javascript, c++, json, bootstrap

Keywords

audio processing, Arm, embedded system, Linux, client, server, javascript, c ++, json, bootstrap

Citace

LACH, Martin. *Vizualizace parametrů mnohakanálového zvukového systému v internetovém prohlížeči.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Igor Szőke, Ph.D.

Vizualizace parametrů mnohakanálového zvukového systému v internetovém prohlížeči

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Ing. Igora Szőkeho, Ph.D.. Další informace mi poskytli Lubor Přikryl, David Obořil a Jan Havran z firmy Audified. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Lach
21. května 2019

Poděkování

Děkuji svému vedoucímu diplomové práce za zadání na míru. Janu Havranovi za seznámení se software a Davidu Obořilovi za přípravu hardware.

Obsah

Kapitola 1

Úvod

Smyslem mojí diplomové práce je vytvořit takovou aplikaci, která umožní snadné a přehledné ovládání mnohakanálového zvukového systému. Při tvorbě této práce navazuji na svůj semestrální projekt, který měl shodné zadání.

Práce se zabývá systémem na zpracování audia od firmy Audified. Jedná se o „černou skříňku“, do které se zapojí mikrofony a počítač (obrázek 2.1). Pomocí počítače se zařízení nastaví, tak aby sbíralo data z mikrofonů a posílalo je počítači k uložení a dalšímu zpracovávání. Tento systém na naší Fakultě informačních technologií využívá například k měření impulzní odezvy uzavřených prostor. V současnosti existuje nevyhovující řešení, které má být nahrazeno.

Tato zpráva popisuje postup práce chronologicky od seznámení se s hardware a software přes návrh a implementaci po výsledný produkt.

V kapitole 2 je popsáno, co se v oné „černé skříňce“ skrývá a jak funguje. V další kapitole (3) je návrh toho, jak vylepšit současný stav ovládání. V kapitolách 4 a 6 je popsána implementace a použité technologie. V kapitole 5 popisuji, jak jsem zprovozňoval streaming dat a implementoval napojení ovládání do současného sytému. Kapitoly 7 a 8 se zabývají výslednou aplikací a testováním.

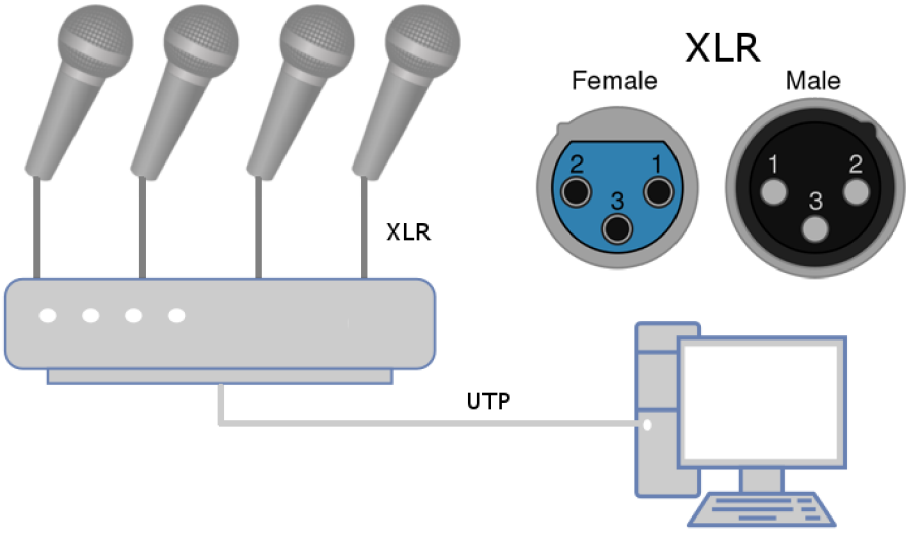
V textu hojně využívám slova, která se používají při práci se zvukem, například u mixážních pultů. Nemají dobrý český překlad, nejdou přeložit jednoslovně, případně mají české překlady nejednoznačný význam. Proto jejich definice uvádím hned v úvodu práce.

- Kanál – označuje jeden konkrétní vstup (výstup) od konektoru (XLR, jack, ...) přes ovládací prvky až do zpracování počítačem.
- Gain – zisk vstupního předzesilovače – čím je vyšší hodnota, tím citlivější je vstup. Dá se snadno přeložit jako zisk, ale v kontextu audia mi připadá vhodnější používat zažité anglické označení.
- Phantom – označení phantomového napájení. Přepínač, který ovládá přívod napětí na daný vstup.
- Úroveň – amplituda signálu daného vstupu po zesílení předzesilovačem (po aplikaci gainu)
- Stream – tomuto slovu jsem se nedokázal vyhnout už v úvodu. V této práci jako stream označuji výhradně přenos zvukových dat ze zařízení do počítače.
- Playlist – seznam skladeb.

Kapitola 2

Popis dodaného HW a SW

Zásadní součástí systému pro zpracování audia je základní deska. Ta obsahuje ARM procesor, důležité porty a sběrnice pro komunikaci s ostatními součástmi. K základní desce se připojují moduly s audio vstupy či výstupy, používáme XLR konektory.



Obrázek 2.1: Schéma zapojení systému, schématické znázorněné XLR konektoru.

Data se přenáší do počítače připojeným UTP kabelem (2.1). K zachytávání slouží aplikace v počítači. Parametry přistroje se nastavují úpravou konfiguračních souborů.

2.1 Pohled na HW

Celé zařízení je modulární. Může se měnit počet vstupů, výstupů, mohou se přidávat další komponenty. S nimi se mění vzhled celého zařízení. Dalo by se říct, že je tu velká podoba se skladbou stolního počítače. Základní deska obsahuje procesor, operační paměť a porty,

které jsou pro nás známé, například RJ-45, USB, RCA. Z těch pro tuto práci zajímavých věcí jsou zde ještě dva A/D převodníky a MikroBUS [1] sběrnice.

Obrázek 2.2: Fotografie systému.

Karty vstupů/výstupů

Kromě základní desky jsou nezbytné i vstupy a výstupy. Ty se nacházejí na menších kartách. Většinou po čtyřech. Jedná se o XLR, či JACK 6.3 konektory. Vidět jsou na obrázku 2.2.

Boot

Boot probíhá po zapnutí napájení. Používaná je metoda, kdy je systém načten z SD karty, protože pro ladění je tato možnost nejvhodnější. Karta může být naformátovaná jako *ext*, nebo *FAT* souborový systém. Je ale třeba, aby byl formát kompatibilní s bootloaderem. Průběh bootu je monžné sledovat a ovládat přes UART (Universal asynchronous receiver-transmitter) rozhraní.

UART

Univerzální asynchronní přijímač–vysílač. K přístupu z počítače je potřeba odpovídající součástka. Například USB–UART převodník. Přístup a konfiguraci umí vyřešit například program *Putty*.

2.2 Použitý procesor

V řešení, se kterým jsem pracoval, je použitý čip od firmy Analog Devices. Konkrétně model ADSP-sc589 [2].

[1] https://www.mikroe.com/mikrobus

Obrázek 2.3: Fotografie detailu systému.

Tento procesor je určen přímo pro profesionální audio, ale využívá se i v jiných oblastech. Například víceosé řízení motoru, nebo systémy distribuce energie. Hlavními součástmi (obrázek 2.4) jsou dvě SHARC jádra a ARM procesor Cortex A5. k tomuto čipu je připojená DDR paměť. Dále jsou k němu přivedeny slot micro SD karty (z ní se bootuje Linux) a flash paměť, ve které je nyní uložen pouze bootloader a program pro SHARC jádra.

2.2.1 ARM architektura

Procesor systému je založen na zmíněné ARM architektuře [7]. Procesory této architektury jsou využívány například v chytrých telefonech a vestavěných systémech. Patří do rodiny RISC (procesory s redukovanou instrukční sadou) a také se vyznačují nízkou energetickou náročností.

Architekturu vyvíjí [4][6] společnost ARM holdings. Schémata procesorů jsou pod licencí poskytována výrobcům, kteří s jejich využitím vytváří své vlastní produkty. Za zmínku také stojí, že na těchto procesorech běžně fungují operační systémy jako Android, Windows Phone, různé distribuce Linux nebo iOS.

SHARC

SHARC je digitální signálový procesor (DSP), který slouží pro operace s plovoucí řádovou čárkou v reálném čase. Proudí přes něj veškerá audio data. Každý SHARC má u sebe jednu 640 kB SRAM paměť. V této práci jsem byl od fungování signálového procesoru odstíněn rozhraním ALSA, detailnější popis proto není důležitý. Výchozí vzorkovací frekvence je 48 kHz, formát vzorku *float* (IEEE 754-1985 Single).

Cortex A5 [3]

Cortex A5 je 32 bitový ARM procesor z řady ARMv7-A, rozšířený o operace v plovoucí řádové čárce a Neon tak, aby zvládal zpracovávat data a zároveň je přes rozhraní mohl komunikovat periferními zařízeními. Využívá L1 instrukční a datovou cache (každá 32 kB) a 256 kB L2 cache.

Z rozhraní jsou pro nás nejdůležitější ethernet a slot pro SD kartu. Jsou zde i další ale u těch chybí programová podpora. Například porty USB a RCA, které by mohly být pro tento projekt velmi přínosné bohužel nejsou zprovozněné.

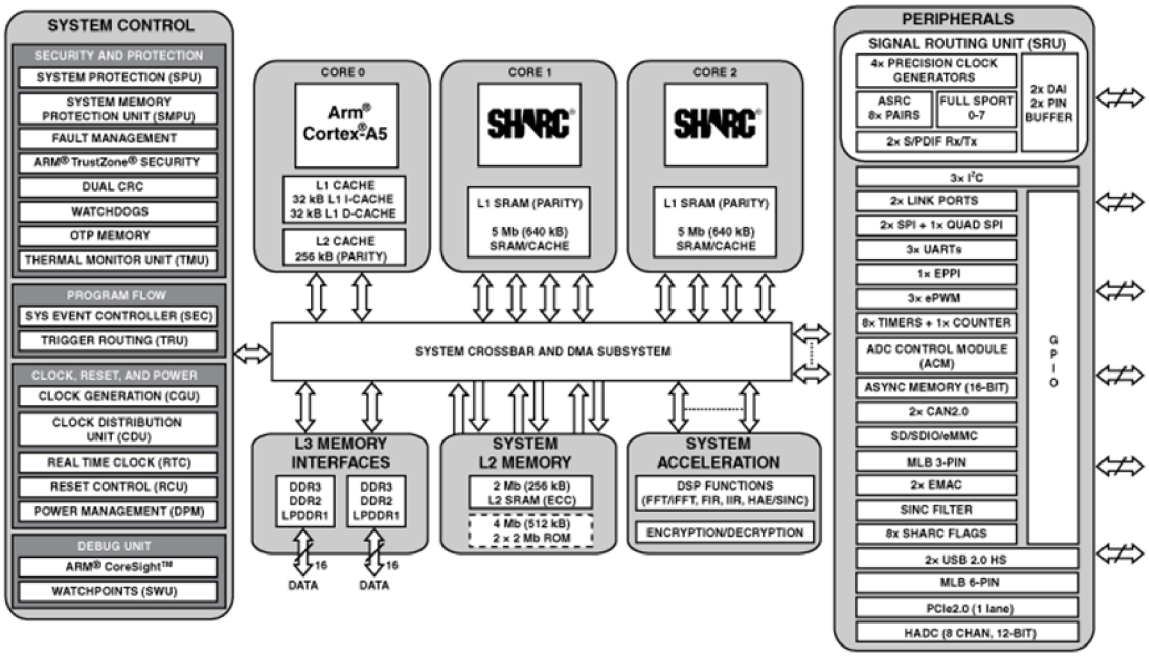
Na tomto jádře běží vestavěný Linux, vytvořený a zkompilovaný přímo pro dané použití. Pro tento Linux budu implementovat veškeré úpravy.

2.2.2 A/D převodníky

AK5558 [2] - 32 bitový, 8kanálový A/D převodník. Umožňuje vzorkovat až frekvencí 768 kHz. Využívá se ale frekvence 48 kHz, která je pro slyšitelné zvuky naprosto dostačující.

Podle vzorkovacího teorému tímto vzorkováním zachytíme až 24 kHz. Zvuky s frekvencí vyšší, než 20 kHz se označují jako infrazvuky. Ty by neměly být pro člověka slyšitelné, i když je to do jisté míry individuální. Použitá vzorkovací frekvence má ještě 4000 Hz rezervu.

Tato vzorkovací frekvence se zároveň standardně používá v profesionálním audiu a tak je vhodná i pro kompatibilitu s dalšími systémy.



Obrázek 2.4: Funkční schéma Analog Devices adsp-sc589.

[2] https://www.akm.com/akm/en/product/datasheet1/?partno=AK5558VN

2.3 Vestavěný operační systém

Po připojení napájení se operační systém zkopíruje z sd karty do paměti ram, kde funguje. Systém nemá přístup k perzistentní paměti. Každé vypnutí je tvrdý restart a po opětovném zapnutí je systém obnoven z sd karty. Veškerá data jsou ztracena.

To přináší následující omezení:

- velikost paměti – celkem 105 MB, systém zabírá 22 MB
- embeded systém má méně funkcí a knihoven než plnohodnotná distribuce
- pro přidání programu je třeba zkompilovat a přenést jádro systému
- systém běží v paměti ram, z čehož vyplývá, že nelze provádět perzistentní změny, data je třeba ukládat na externí úložiště

Velikost paměti v tomto případě, kdy se neukládají žádná data, není omezující. Pro testování je možné nakopírovat potřebné soubory pomocí vzdáleného přístupu.

Vzdálený přístup

K přípravku je možné se vzdáleně připojit pomocí LAN portu, přes SSH protokol. Přístup dovolí vše, co je potřeba. Především kopírování souborů a spouštění programů.

2.4 Nastavované parametry

Pro každý kanál (vstup) je možné nastavit dva parametry – a phantom (phantomové napájení) a gain (zisk).

Phantom

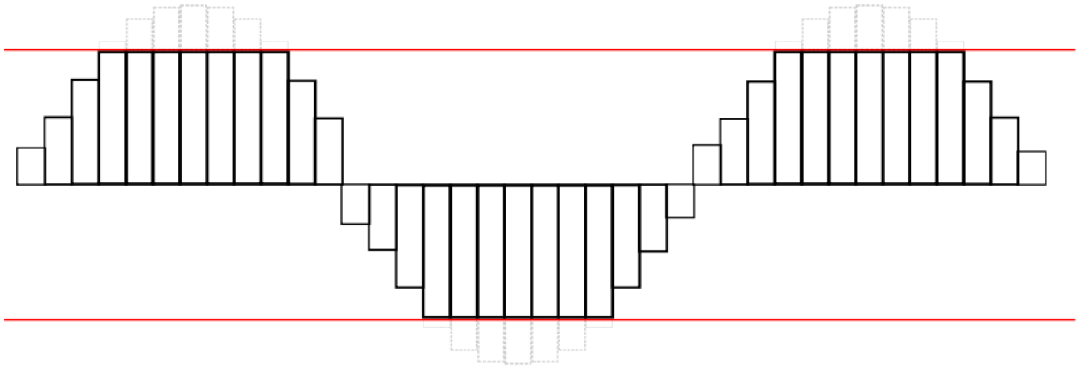
Phantomové napájení má polohy zapnuto a vypnuto. Jde o přídavné napájení XLR konektorem. Velikost napětí záleží na výrobci a daném produktu, obvykle bývá +48V.

Využívá se především pro kondenzátorové mikrofony, ale i jiná zařízení, která potřebují napájení. Výhodou tohoto řešení je, že se pro data i napájení používá jeden kabel. Nevýhodou je potom riziko poničení přístrojů, které napájení nepotřebují. Toto riziko je však při správném zapojení velmi malé a pravděpodobnost poničení přístroje tudíž minimální.

Potenciální problém plyne z nesymetrického napájení, způsobeného například odpojováním a připojováním konektoru, poničeným kabelem, nebo opotřebovanými konektory. Nemá cenu spekulovat o tom, jestli by nestačilo mít přivedené napájení po celou dobu. Dobrou praxí je zkrátka zapínání phantomového napájení pouze na těch vstupech, kde je potřeba.

Gain

Gain je upravení zisku předzesilovače tak, aby úroveň vstupního signálu byla optimální pro AD převodník. Při nedostatečném vybuzení je signál nerozlišitelný od šumu. Při přebuzení dochází k takzvanému „clippingu“ (2.5) – zkreslení, kdy největší amplitudy jsou ořezané, protože pro každou hodnotu větší než maximální je zpracována právě maximální hodnota.



Obrázek 2.5: Příklad „clippingu“ a PCM.

2.5 Současné SW řešení

Tuto sekci rozepisuji detailněji, protože v této práci bylo nutné pochopení všech částí systému, které jsem následně upravoval. K software není existující dokumentace, veškeré informace jsem byl nucen čerpat ze zdrojových kódů a jejich komentářů. V řešení, které má fakulta k dispozici se k předávání dat používá konzolová aplikace. K nastavení parametrů slouží konfigurační soubor. Průběh práce není přívětivý ani intuitivní. Je třeba udělat následující:

- spustit nahrávání
- otevřít nahraná data a zobrazit úrovně
- přepsat konfigurační soubor – odhadnout vhodné hodnoty
- opakovat

Je třeba dodat, že současné řešení funguje se starší verzí přístroje. Tato práce se zabývá především novějším, aktuálním přístrojem Audified. Komunikaci tak, jak v tuto chvíli funguje jsem znázornil na obrázku 2.6.

2.5.1 Audio streaming

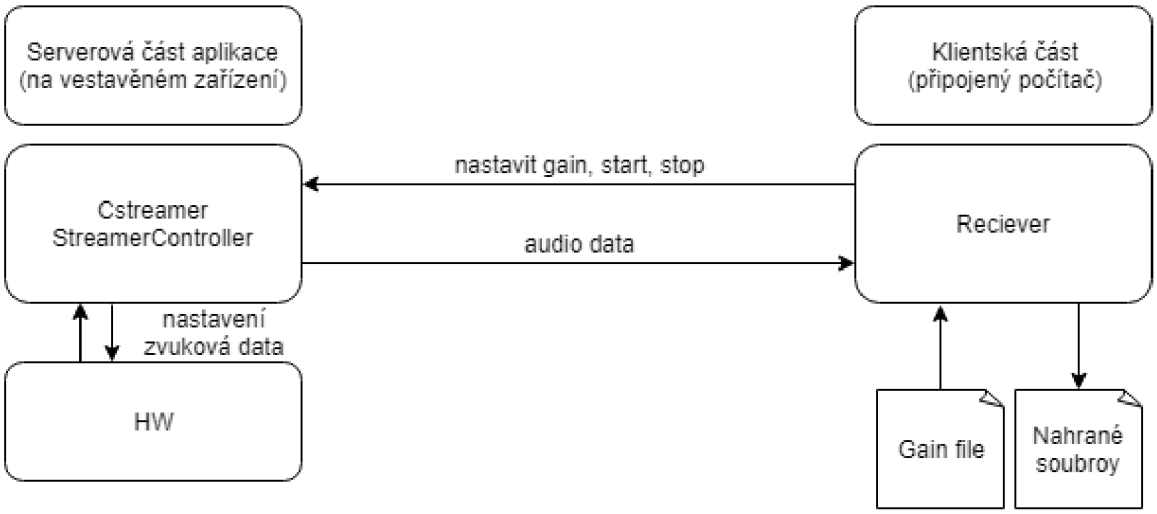
Vícevláknový program běžící na zařízení. Čeká na připojení Recieveru, komunikuje s Recieverem, čte audio data z SHARC jádra, odesílá audio data.

Streamer

Streamer je třída sloužící pro čtení dat z hardware a jejich odesílání přijímači. Je připravena na různé implementace získání dat. V současnosti využívá MCAPI (Multicore Communications API).

Vyčítá data z SHARC jádra. Data jsou získána přímo rozhraním MCAPI, jsou uložena do kruhového bufferu. Kruhový buffer je pomocí mutexů synchronizován s vláknem sloužícím pro odesílání. Jakmile je buffer naplněný neodeslanými daty, dochází k blokování aplikace – to se nesmí stávat.

Používá se nestandardní 3 bytový formát vzorků. Pro kompatibilitu s různým množstvím kanálů se používá prokládání.



Obrázek 2.6: Schéma komunikace jednotlivých částí současného systému.

StreamerController

Inicializuje Streamer a síťové připojení, vyřizuje řídící zprávy. Implementuje rozhraní pro přístup ke Streameru.

Při běhu spouští vlákno pro komunikaci. Přijímá řídící pakety pro spuštění streamu dat, jeho ukončení a nastavení ztlumení kanálů.

2.5.2 Síťová komunikace

Další samostatnou třídou je protokol pro síťovou komunikaci. Zde popíši formát paketů, způsob uložení dat a celé odesílání.

Formát paketů

Každý paket je uveden označením typu (výčtem). Nejpoužívanější typy jsou datový paket (kNetMessageData), konfigurační (kNetMessageCfg) a informační (kNetMessageInform) paket. Jsou připravené i další ale jejich využití je okrajové, nebo se nepoužívají vůbec. Proto se o nich nezmiňuji. Využití paměti se liší podle typu paketu, viz. obrázek 2.7.

Datový paket

Používá se ke streamování dat. Kromě vyjádření typu, který obsahují všechny pakety, obsahuje časovou známku (timestamp), která určuje pořadí paketu ve streamu.

Následuje počet vzorků, který vyjadřuje velikost posílaných dat. Zbytek zabírají samotná data.

Konfigurační paket

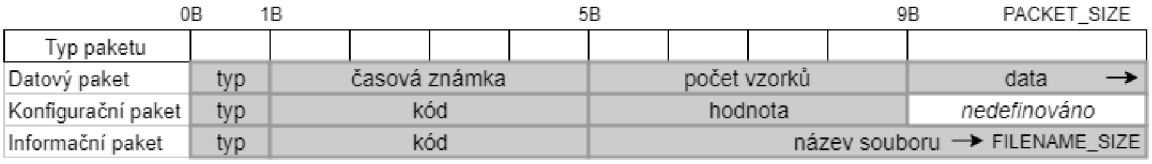
Tímto paketem se například zahajuje nahrávání. Historicky sloužilo k nastavování parametrů.

Obsahuje dvě číselné informace. Kód zprávy a hodnotu. Kód je opět vyjádřený výčtem. Hodnota dostává smysl až podle použitého kódu.

Informační paket

Jak název napovídá, obsahuje informace. Kód slouží obdobně jako v předchozím případě k bližší identifikaci zprávy. Například „spuštění přehrávání“.

Název souboru primárně přenáší jména souboru až do velikosti `FILENAME_SIZE`. Omezení délky souboru slouží k tomu, aby nebylo třeba posílat informace o délce názvu.



Obrázek 2.7: Typy paketů s využitím paměti.

Odesílání dat (streaming)

Odesíláni dat musí být rychlejší, než jejich nahrávání. S přihlédnutím k rychlosti sběru dat probíhá nahrávání velmi dlouho. Proto není přípustné žádné hromadění dat. Za minutu vznikne na jednom kanálu cca 10 MB dat. Délka nahrávání se počítá na hodiny, případně dny.

Je třeba zachovat časově synchronizovaná data. Odesílání musí být spolehlivé. Proto je každý paket označen časovou známkou a kontroluje se jeho doručení.

Čím menší velikost paketů, tím větší nároky na režii (přenášení hlaviček a dalších metadat), větší pakety zase zdržují a je problém při jejich znovu zasílání. Je proto nutné najít vhodný kompromis.

2.5.3 Reciever

Jak název napovídá, tato programová část je opakem Streameru. Spouští se na počítači. IP adresou a portem se připojí k přístroji. Činností Recieveru je ovládat stream (spustit, zastavit), přijímat data a ukládat je do souborů.

Zde se také využívají kruhové buffery. Jednak je výhodnější ukládat do souboru až větší množství dat, ale také se zde rozděluje prokládaný formát dat na jednotlivé kanály – pro každý kanál je jeden buffer. Zde se také mění formát vzorků na standardnější velikost, ze 3 bytových na 4 bytové.

Kapitola 3

Přenos vizualizačních a řídicích dat – návrh

Přístroj nemá na sobě žádná tlačítka, ani display. Cílem je vytvořit aplikaci, která usnadní ovládání systému a dá uživateli zpětnou vazbu o tom, co se zrovna děje. Usnadnění spočívá ve zlepšení přehlednosti – grafické zobrazení úrovní s nepostřehnutelným zpožděním, možností okamžité úpravy parametrů, správa seznamů skladeb a přehrávání audio souborů.

Aplikace by měla být integrovaná v zařízení. Ovládání se zobrazí jako webová stránka, po zadání IP adresy do prohlížeče, podobně jako při nastavování běžného domácího routeru. To vyžaduje vytvoření http serveru a webové stránky. Serverová část bude mít následující funkce:

- naslouchání na přidělené IP adrese a portu
- přenos klientské aplikace na připojený počítač
- vyřizování požadavků klientské aplikace
- řízení hardware a získání vizualizačních dat

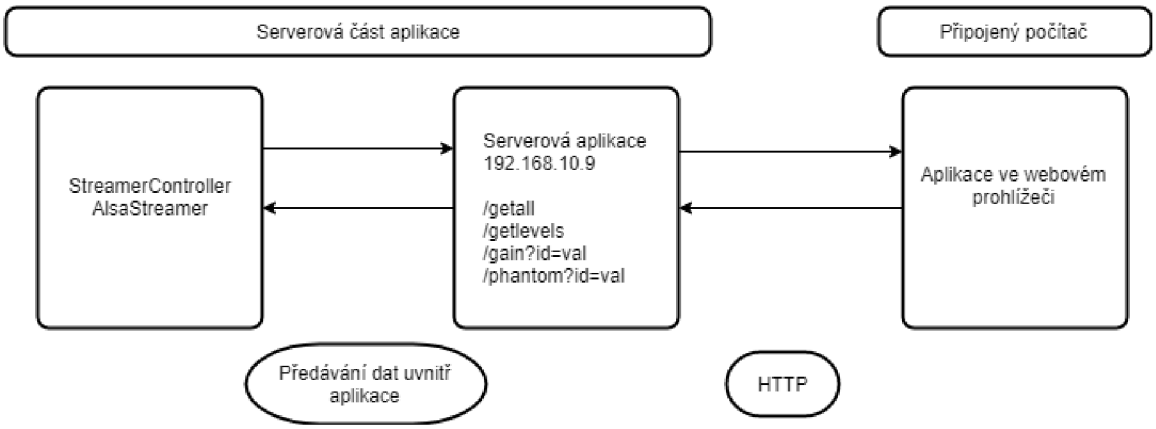
Klientská webová aplikace bude:

- použitelná s většinou webových prohlížečů
- mít přehledný vzhled a intuitivní ovládání
- posílat požadavky serveru a přijímat odpovědi

Vzniká potřeba několika komunikačních rozhraní. Webové, je mezi serverem a klientem na připojeném počítači. Přes toto rozhraní se nastavují parametry hardware. Další rozhraní potom slouží pro samotné získání dat (úrovní).

Data jsou „živá“. Neustále přibývají nové vzorky, které je třeba zpracovat. Rychlost přibývání dat závisí na vzorkovací frekvenci, počtu kanálů a velikosti vzorku. Počet kanálů je proměnný, vzorkovací frekvenci používáme 48 KHz, velikost vzorku je 4 byty. Ke získání stavu hardware a jeho ovládání využiji rozhraní třídy StreamerController. To bude rozšířeno tak, aby bylo možné získat všechna potřebná data.

Standart pro posílání dat webové aplikaci je formát JSON. Jedná se o textový formát. Není tedy příliš podstatné jak jsou parametry uložené, pokud se dají převést na řetězec.



Obrázek 3.1: Schéma komunikace

3.1 Řídící data

Pro inicializaci a úvodní nastavení je třeba získat počet kanálů, jejich gain a phantom. Dále stav přístroje (je aktivní nahrávání, je inicializované síťové připojení, ...), po rozšíření také seznamy skladeb, zvukové soubory a právě přehrávanou skladbu. V neustálých aktualizacích je třeba získávat úrovně jednotlivých kanálů a stav přístroje. Zpracovávat a odesílat je třeba změny v parametrech, ovládací příkazy a ukládání seznamů skladeb.

Počet kanálů je definovaná konstanta. Pro nastavení phantomu stačí jeden bit. Je to hodnota boolean – zapnuto, vypnuto. Pro gain je třeba více bitů, využívá se sedmi různých úrovní. Tyto dva parametry je možné vyčíst z příslušných souborů. Pro hodnotu úrovně vyjádřenou procenty aplikaci rovněž stačí omezený rozsah. Hodnota úrovně se musí připravit z několika vzorků, aby při požadavku mohla být odeslána aktuální odpověď. Stav přístroje se dá vyčíst ze stavu síťového připojení a stavu Streameru.

3.1.1 Úrovně

Jak už jsem zmiňoval, zvukové vlny se kvantifikují na vzorky. Pro optimální funkci A/D převodníku musí být přivedený signál ve správném rozsahu. Síla signálu se nastavuje ziskem – parametrem gain. Pro informaci uživatele slouží zobrazení úrovně signálu. Vypočtení hodnoty není triviální a je možné přistupovat k němu několika způsoby. Při zpracování zvuku se používají hlavně dva přístupy a to efektivní hodnota (RMS) a maximální hodnota (peak).

RMS

RMS ukazuje hodnotu výkonu signálu, jakou by měl odpovídající stejnosměrný signál. Počítá se z více vzorků, smysl dává výpočet ze vzorků sesbíraných za více než půl sekundy. Ukazuje sílu signálu, ale nezobrazí se zde ořezané vzorky (2.5).

Peak

Peak slouží pro zobrazení maximální hodnoty. Na tomto měření je vidět především clipping. Pro danou aplikaci, kdy je důležité přesné nahráváni, mi tato metoda přijde důležitější. Proto jsem se rozhodl právě pro ni.

3.1.2 Zvukové soubory a jejich seznamy - návrh

Zvukové soubory mohou být ukládány přímo na SD kartu, konkrétně na část, kde není systém. To je spíše bezpečnostní opatření, než že by to způsobení jiným omezením.

Soubory se dají na kartu kopírovat přímo fyzicky, pomocí čtečky. Je ale nutné se ke kartě dostat. K tomu je potřeba rozebrat zařízení a určitě to není nejrozumnější řešení.

Naštěstí je tu i druhá možnost a to využít SSH, případně SFTP. Linuxové systémy podporují připojení serveru přímo do souborového systému. Potom se s ním dá pracovat podobně jako s jakýmkoliv jiným adresářem. Samozřejmě lze využít scp.

U Windows je třeba použít program podporující tyto protokoly. Například Putty, nebo Bitvise SSH Client.

Adresářová struktura

Na příslušném oddíle SD karty jsou dva adresáře. `Audio` slouží právě pro uložení souborů. `Playlist` potom pro uložení playlistů.

Formát souboru

Podporovány budou pouze wav soubory. Vzorkovací frekvence musí být shodná s frekvencí HW. Počet kanálů může být různý - využije se jich tolik, kolik je maximálně možné. Formát vzorků může být téměř libovolný. Nejvhodnější je `float`, ostatní formáty se musejí převádět. Převádění je ale automatické.

Formát playlistu

Playlist je textový soubor, který obsahuje názvy zvukových souborů oddělené čárkami. Kvůli tomu názvy soubory nesmí obsahovat čárku.

3.2 Rozhraní pro komunikaci s front end aplikací

Tvorba tohoto rozhraní byla přímočařejší. Z funkčního hlediska implementuje totéž. Server tvoří REST rozhraní, které volá aplikace podle potřeby a volby uživatele. Data směrem od uživatele k serveru jsou předána v GET http parametru. Odpovědi jsou ve formátu JSON, který je snadno zpracovatelný JavaScriptem. Rozhraní se skládá z následujících zdrojů, které mají své URI:

/getall

Bez parametrů. Vrací nastavení gain a phantom pro všechny kanály. JSON obsahuje dvě pole – gains a phantoms. Číslo kanálu se shoduje s indexem pole.

/getlevels

Funguje podobně jako *getall*, vrací úroveň signálu pro všechny kanály.

/gain?id=value

Parametr x je číslo kanálu, y nastavená hodnota gain. Odpověď je shodná s *getall*.

/phant?id=value

Stejně jako *gain*, y je nastavená hodnota phantom.

/getaudiofiles

Vrací názvy souborů wav v příslušném adresáři.

/getplaylist?name

Vrátí obsah daného playlistu.

/save?name=values

Přiřadí audio soubory `values` do playlistu `name`.

3.3 Porovnání existujících řešení pro embeded web server

Samotné tvorbě serveru předcházel výzkum možností. Studování různých řešení, která by usnadnila práci na již mnohokrát vyřešeném problému. Z možností jsem sáhl po střední cestě, která je mezi programem v čistém C++ a frameworkem.

C++ REST SDK [1]

Projekt k tvorbě server-klient aplikací s nesynchronními operacemi od Microsoftu. Obsahuje nástroje pro připojení a komunikaci s REST rozhraními.
Klady:

- Je součástí operačního systému, používá se v Audified
- Vývoj Microsoftem s podporou Linuxu

Zápory:

- Stručná dokumentace (přechod z verze pojmenované Casablanca)
- Málo příkladů
- Není přímá podpora funkce pro předávání souborů
- Příliš robustní
- Je určen na připojování se ke službám, ne jejich vytváření

Minihttp server [2]

Minimalistický http server v C++, popsaný v článku na root.cz [3].
Klady:

- Používá čisté C++, bez dalších knihoven

[1] https://github.com/Microsoft/cpprestsdk
[2] https://github.com/ondra-novak/minihttp
[3] https://blog.root.cz/novacisko/minimalisticky-http-server-v-cpp/

- Je opravdu miniaturní (pouze jeden soubor)
- Je určený právě pro daný účel, podporuje zpracování jednotlivých žádostí pomocí skriptu.
- Pěkné výpisy stavu

Zápory:

- Používá C++17, křížový překladač ho nepodporuje
- Není přehledný - pouze jeden soubor

Projektů jsem porovnával více, všechny jsou si velmi podobné a tak mi nepřijde důležité se o nich rozepisovat. Jak jsem psal v úvodu, zvolil jsem kompromis mezi psaním v čistém C (C++) a obsáhlým projektem. Tím je knihovna Boost. Využil jsem a upravoval jeden z poskytnutých příkladů – http server.

Knihovna Boost [1], http server [5]

Boost je velmi rozsáhlá knihovna, která usnadňuje práci v mnoha oblastech. Řadí se mezi standardní knihovny a její součásti postupně přecházejí do norem C++.
Klady:

- Knihovny jsou součástí operačního systému, používají se v Audified
- Velmi dobrá dokumentace a podpora na fórech
- Hotový http server mezi příklady
- Přehledně psaný, snadno rozšiřitelný kód

Zápory:

- Jednotlivé verze knihovny nejsou plně kompatibilní

Kapitola 4

Programová část pro posílání dat klientské aplikaci

Tato kapitola popisuje serverovou část. Jedná se o aplikaci psanou jazykem C++, určenou k běhu na vestavěném systému. Její tvorba přinesla několik výzev. Zejména synchronizaci mnoha vláken, kompilování pro cílový systém, bezchybné vyčítání a posílání dat z ALSA zařízení, funkční a smysluplné propojení celého systému při zachování kompatibility. To vše za používání pouze knihoven, které jsou dostupné v systému – toto se nakonec ukázalo jako slabé omezení. C++11 se dá využívat plnohodnotně.

4.1 Křížová kompilace (cross compilation)

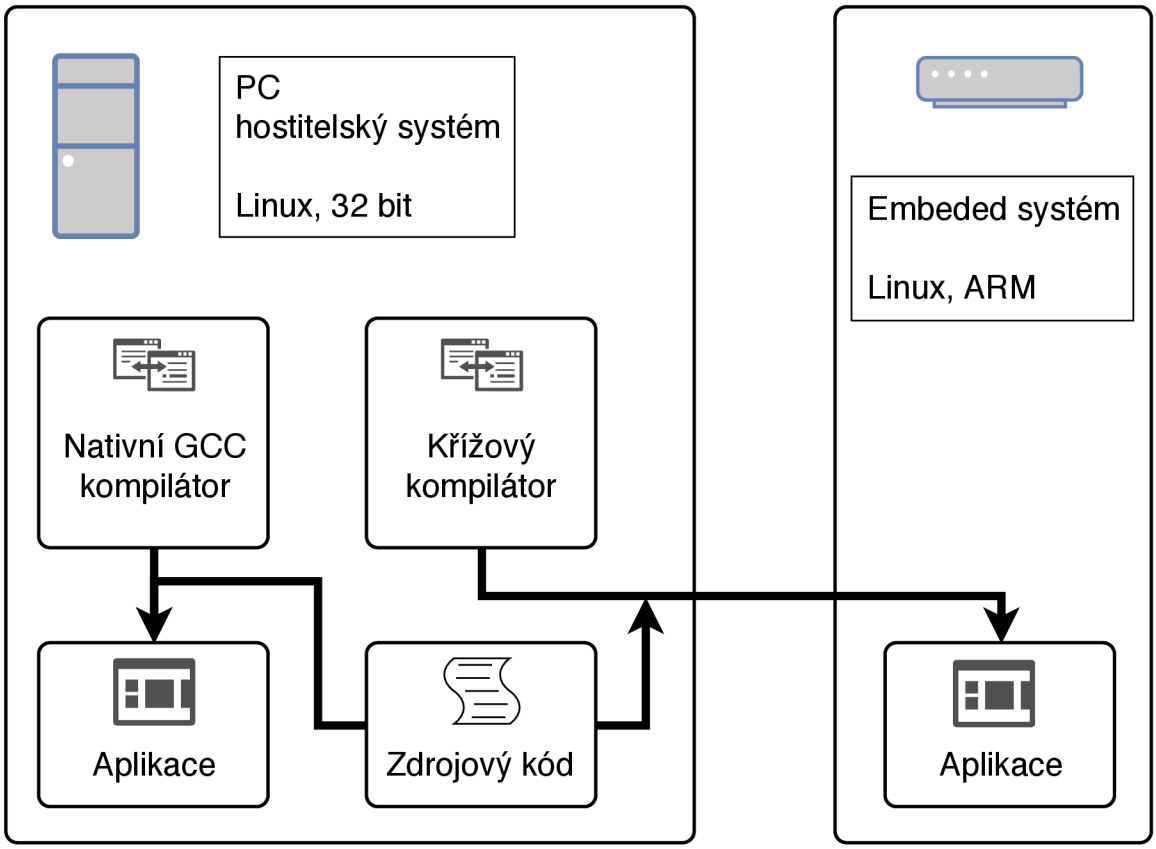
Pro správnou funkci aplikace je ideální ji zkompilovat právě na systému, kde má běžet. Vestavěné systémy mají velmi omezené prostředky. Ať už jde o výkon, přítomnost kompilátoru, nebo dokonce operační systém. Proto se přistupuje ke křížové kompilaci, kde *hostitelský systém* dokáže zkompilovat kód pro cílovou platformu 4.1.

Křížová kompilace je velmi citlivá na nastavení. Je možné stáhnout různé balíčky rozšiřující gcc o překlad pro jiné platformy. Jejich nastavení je bez detailní znalosti cílové platformy a operačního systému téměř nemožné. Při testování jsem úspěšně překládal programy se statickým linkováním knihoven. To samozřejmě není ideální přístup. Jako jediné vhodné řešení se ukázalo použít křížový překladač, který poskytuje přímo Buildroot - nástroj pro generování vestavěného linuxu. Byl nakonfigurovaný Audified pro daný operační systém. Balíček obsahuje gcc (g++), překladač assembleru i hlavičkové soubory knihoven.

4.2 Ukládání dat

Linux se při spuštění bootuje z SD karty, nebo flash paměti. Následně běží v nevolatilní paměti. Omezení, které to způsobuje je, že veškerá uložená data jsou po restartu ztracena. To je zároveň velká výhoda, protože tímto způsobem je systém ochráněn před hrozbou poškození uživatelským zásahem.

Potřeboval jsem vyřešit, jak ukládat audio soubory a seznamy skladeb. Nedává smysl mít je v připojeném počítači. Nahrávání může probíhat pokaždé s jiným počítačem a přehrávaná data by bylo třeba streamovat na přípravek. Tím by se ztratila výhoda integrace této funkce, protože by bylo možné použít jakoukoliv jinou zvukovou kartu, vestavěný přehrávač počítače a vše vyřešit jednodušeji.



Obrázek 4.1: Schéma klasické a křížové kompilace.

Řešení, které jsem vymyslel využívá přiloženou SD kartu. Tu jsem naformátoval tak, aby měla dva oddíly. Jeden na operační systém, druhý právě na data, která bychom mohli využívat. Tato paměť bude připojena příkazem *mount* do souborového systému již při bootování a uživatel ani nepozná, že je tato paměť na jiném disku. Rozdíl oproti zbytku souborového systému bude v tom, že data zapsaná na tomto svazku zůstanou dostupná i po restartu. Paměť je omezena velikostí SD karty.

4.3 Web server

Server obsluhuje jeden počítač připojený na přímo kabelem. Zařízení není připojeno k internetu. Tím odpadá potřeba paralelního vyřizování požadavků a zabezpečení. Předpoklad je, že pokud má uživatel fyzický přístup k zařízení, nemá ani význam server zabezpečovat. Aplikace potřebuje přístup k http portu (80), spouští se tedy s právy root.
Server má tři funkční bloky.

1. File server – pokud je specifikován soubor, odešle jeho obsah, nebo zápornou odpověď
2. Práce s daty – pokud není v URI dotaz na soubor, ale na zdroj, vykoná danou činnost a odpoví JSON daty
3. Předávání dat z, nebo do zbytku systému

Aplikace vychází z [5]. Upravil jsem třídy `request_handler` a `mime_types`, které řeší http požadavky a přidal novou třídu `api_handler`, pro požadavky na aplikační rozhraní. Funkce `main` a pár dalších jsem se zbavil a nahradil tím, co bylo potřeba.

Request handler

Do této třídy přichází adresa požadavku, ta je zpracována. Požadavky se dělí na žádosti o soubory a data.

Nejdříve se zkontroluje, zda je cesta validní – přece jen by nebylo správné poskytnout ke stažení přes toto rozhraní celý souborový systém. Adresa nesmí obsahovat například „..“ pro přechod do nadřazeného adresáře.

Pokud adresa obsahuje „*?*“, přeposílá se požadavek na `api_handler`. Pokud se v adrese otazník nenachází, zpracuje požadavek přímo `request_handler`. Podle přípony se určí typ souboru a příslušný typ internetového média (MIME - Multipurpose Internet Mail Extensions). Pokud není daná přípona nalezena, použije se *text/plain*. Výjimkou je zakončení adresy lomítkem. V tom případě se požadavek před zpracováním doplní o `index.html`.

Pokud je soubor nalezen, jsou naplněny hlavičky, stav zprávy a společně s daty je vše odesláno. Pokud soubor nebyl nalezen, nebo se vyskytla jiná chyba, Je použita knihovní zpráva `boost::asio::buffer(bad_request)`.

Api handler

Pro posílání dat do frontendu jsem se rozhodl využít bezstavové rozhraní rozhraní. Server čeká na požadavek klienta. Neukládá stav komunikace ani jiné informace. Vyřídí požadavek a čeká na další. Server šetří prostředky tím, že odesílá data jen pokud o ně někdo požádá a klientská aplikace může být škálovatelná. Například když se uživatel rozhodne, že chce dostávat aktualizace o stavu zařízení častěji. Stačí změnit příslušný kód JavaScriptu. Protokol je univerzální a může ho využít i jakákoliv jiná aplikace. A to bez jakýchkoliv úprav na serveru. Samozřejmě to přináší i nevýhody. Tou největší je zřejmě to, že pro každou část dat se vytváří nové spojení. Na server tak přichází relativně velké množství požadavků i ve chvíli, kdy pouze zobrazuje úrovně.

Pro zachování jednoduchosti jsou všechny požadavky typu GET. K odeslání dat na server by bylo možná správnější využít metodu POST. Nejedná se ale o citlivá ani velká data. Tak si myslím, že je toto zjednodušení v pořádku. Samotný protokol je popsán výše.

Mime types

Víceúčelová rozšíření internetové pošty (Multipurpose Internet Mail Extensions) je internetový protokol vytvořený pro email, ale používaný i dalšími protokoly. Mezi nimi i HTTP. Sloužil k rozšíření elektronické pošty o specifikaci kódování textu (diakritika), přenos binárních dat, příloh a podobně.

Hlavička obsahuje verzi protokolu, typ obsahu, kódování a několik dalších parametrů. Nás nejvíce zajímá použití s HTTP, konkrétně typ obsahu. Bez správně nastaveného typu obsahu webový prohlížeč interpretuje vše jako text. Například css soubory bez správného příznaku odmítá použít.

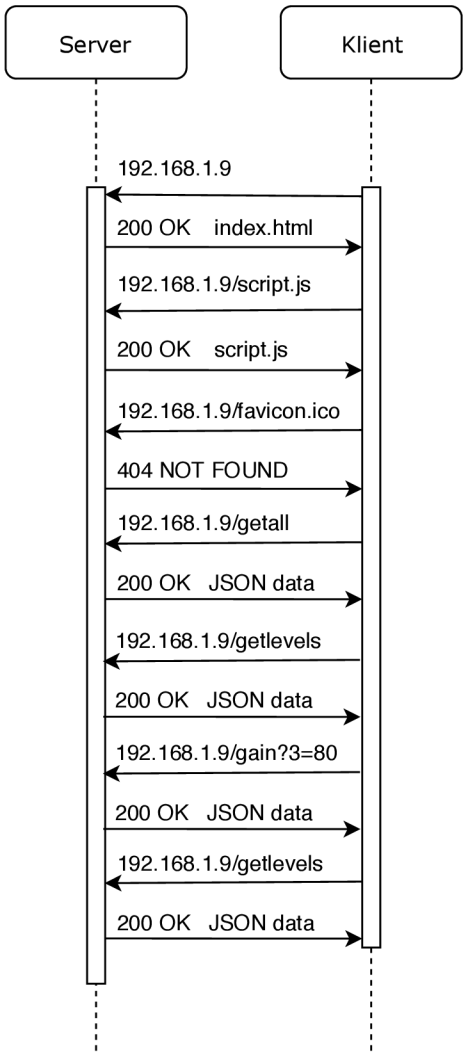
Server rozlišuje typy podle přípony. V základní implementace zvládala pouze text a jpeg obrázky. Bylo třeba doplnit typy pro posílání kaskádových stylů, javascriptu, dalších typů obrázků.

4.4 Playlisty a soubory - implementace

Seznam playlistů a a souborů se odesílá při prvním požadavku klienta. Názvy souborů se získávají přímo ze složky. Jsou omezeny délkou `FILENAME_SIZE`.

Při ukládání pošle klient název playlistu a názvy souborů, které obsahuje. Vše s uloží do souboru. Pokud existuje, přepíše se. Pokud neexistuje, je vytvořen.

Aktuálně vybrané skladby se také ukládají do playlistu. Slouží k tomu `.actualplaylist`, který je před uživatelem ukrytý.



Obrázek 4.2: Sekvenční diagram průběhu komunikace.

Kapitola 5

ALSA Streamer

Tato část programu propojuje HW se zbytkem systému, vytváří pakety a odesílá je. Rozšiřuje třídu *Streamer*. Součástí software, který jsem dostal byl i soubor *CAlsaStreamer*. Nedopátral jsem se k informaci, k čemu sloužil. Po jeho prostudování jsem usoudil, že to byl testovací prototyp, který nemohl fungovat. Z tohoto souboru jsem převzal pouze strukturu. To je zhruba to samé, jako kdybych se rozhodl přizpůsobit fungující *CStreamer*.

Ve verzi, kterou jsem dostal k dispozici například nastavené parametry gain pouze vynásobily získaný signál, mute potom bylo násobení nulou, přičemž na získávání dat se nic neměnilo. Nastavení phantomu nebylo implementované vůbec.

5.1 Inicializace ovladače a technické parametry

Ke zpřístupnění HW je třeba dodržet protokol, včetně netriviálního nastavení struktur a parametrů hardware. Na začátku potřebujeme *handle* pro PCM zařízení. Poté zvolit směr streamu, a to buď přehrávání (playback), nebo nahrávání (capture). Také musíme nakonfigurovat další parametry jako velikost bufferu, vzorkovací frekvence, formát vzorků.

```
// handle pro PCM
snd_pcm_t *pcm_handle;
// stream - nahravani
snd_pcm_stream_t stream = SND_PCM_STREAM_CAPTURE;
// Struktura s informacemi o hardware
//vyuziva se ke konfiguraci PCM streamu.
snd_pcm_hw_params_t *hwparams;
```

Nejdůležitějšími rozhraními jsou PCM zařízení *plughw* a *hw*. Při používání rozhraní *plughw* nezáleží tolik na hardware. Pokud hardware nepodporuje zadanou konfiguraci, například vzorkovací frekvenci nebo formát vzorku, tak je provedena konverze. Vše funguje a uživatel nic nepozná. Při použití *hw* je třeba používat podporované parametry. Uživatel je buď musí znát, nebo předem zjistit.

Následuje otevření PCM zařízení. Je dostupný blokující a neblokující mód. V neblokujícím módu je volání zápisu (čtení) okamžitě vráceno. Zpracována jsou data dostupná ve chvíli volání. Blokující volání blokuje, dokud nezpracuje daný počet vzorků. Před inicializací a použitím zařízení je třeba inicializovat a naplnit strukturu s parametry.

```
// Nazev zarizeni: hw:0,0
// Prvni cislo oznacuje zvukovou kartu, druhe je cislo zarizeni
```

```
char *pcm_name;
// Alokovani struktury snd_pcm_hw_params_t na zasobniku
snd_pcm_hw_params_alloca(&hwparams);

//Otevreni zaarizeni v blokujicim modu
if (snd_pcm_open(&pcm_handle, pcm_name, stream, 0) < 0) {
  fprintf(stderr, "Error opening PCM device %s\n", pcm_name);
  return(-1);

// Init hwparams
if (snd_pcm_hw_params_any(pcm_handle, hwparams) < 0) {
  fprintf(stderr, "Can not configure this PCM device.\n");
  return(-1);
}
```

Parametry se nastavují sadou funkcí `snd_pcm_hw_params_set_<parameter>`. Kromě výše zmíněných je důležitý ještě typ přístupu (access type). Určuje, jakým způsobem jsou data z více kanálů ukládána do bufferu. Bez prokládání `(NONINTERLEAVED)` se postupně ukládají jednotlivé kanály. V případě sterea by to znamenalo, že první půlku bufferu zaplní levý kanál a druhou pravý. V našem případě, u N-kanálového streamu dává větší smysl druhá možnost – prokládání `(INTERLEAVED)`. Nejdříve je uložen první vzorek z každého kanálu, následuje další (obrázek 5.1).

```
if (snd_pcm_hw_params_set_access(pcm_handle, hwparams,
                                 SND_PCM_ACCESS_RW_INTERLEAVED) < 0) {
  fprintf(stderr, "Error setting access.\n");
  return(-1);
}

// Aplikace nastaveni HW parametru
// PCM zarizeni a priprava zarizeni
if (snd_pcm_hw_params(pcm_handle, hwparams) < 0) {
  fprintf(stderr, "Error setting HW params.\n");
  return(-1);
}
```

Nyní je konečně možné vyčítat data ze zařízení. Využívá se ukazatel `pcm_handle`.

```
snd_pcm_readi(pcm_handle, data, num_frames);
```

Používané parametry jsou:

- vzorkovací frekvence: 48000 Hz
- formát vzorků: float little endian – float_LE
- počet vstupních kanálů: 0 – 16
- počet výstupních kanálů: 0 – 2

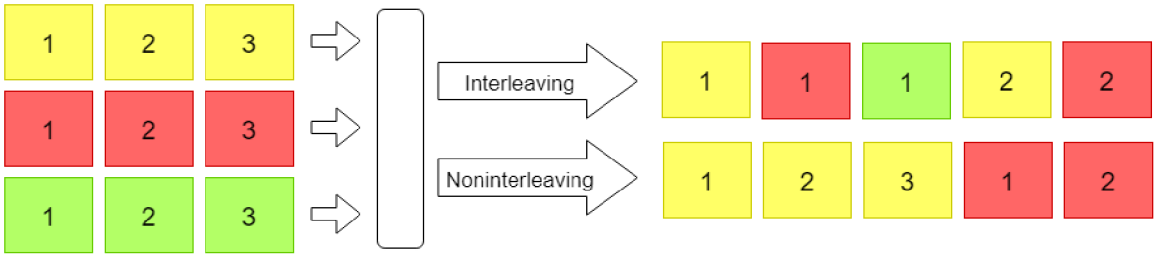
5.2 Čtení, způsob uložení dat

Ihned po inicializaci je zahájen sběr vzorků a jejich analýza. Výsledky analýzy (peak) jsou předávány pro zobrazení. Když neběží nahrávání, nemusí se sítí posílat všechna data. Stačí tyto agregované informace. Proto se ani nevyužívá dalších bufferů. Data se v cyklu pouze čtou a analyzují. Pro tuto činnost bohatě stačí vestavěné buffery ovladače.

Při streamování se všechna data posílají do sítě. To přináší zpomalení a určitou míru nejistoty - mohlo by se stát, že kvůli zpoždění přeteče buffer a nějaké vzorky se ztratí. Taky by nebylo praktické, aby velikost čtených dat byla závislá na velikosti odesílaných paketů, nebo naopak. Proto je mezi čtením a odesláním kruhový buffer, který může uložit několik vzorků před odesláním, tím případně vyrovnat výkyv sítě.

Data získáváme v prokládaném formátu (obrázek 5.1). To je velmi praktické z hlediska škálovatelnosti, protože se nemusí čekat na větší bloky. Je víceméně jedno, kolik kanálů využíváme. Samozřejmě ale není vhodné rozdělit data kdekoliv, ale zarovnat je tak, ať mají všechny kanály stejný počet vzorků.

Při tomto přístupu je díky prokládání zajištěná i časová synchronizace vzorků. Nestane se, že by byl některý kanál zpožděný.



Obrázek 5.1: Prokládání vzorků.

Kruhový buffer je tedy plněn zvukovými daty. Ta čte vlákno pro odesílání. Činnost těchto dvou vláken je třeba synchronizovat tak, nebyla přepsána data čekající na odeslání a aby se neodesílala stará, dříve využitá data. Nahrávací vlákno musí mít zkrátka vždy náskok ale nesmí se předběhnout o celé kolo.

Při odvozování velikosti bufferu je třeba zvážit následující myšlenky:

- Data přibývají konstantní rychlostí pro každý kanál – rychlost roste lineárně s počtem kanálů
- Vzorky budou zarovnané, tzn. dělitelné počtem kanálů
- Buffer by měl být vytvořen pro obecný typ, ale počítat s velikostí vzorku – kolik bytů paměti zabere

Velikost bufferu jsem zvolil konstantu vynásobenou velikostí vzorků a počtem kanálů. Vhodnou velikost konstanty jsem zjišťoval během testování. Aby nebylo třeba synchronizovat každý vzorek, je buffer rozdělen do bloků. Je implementován jako 2D pole, kde první dimenze určuje blok, druhá vzorek v bloku. Velikost bloku je shodná s velikostí čtených dat. Počet bloků je zmíněná konstanta. Každý blok má dva mutexy. Jeden určuje zapsaná data, druhý přečtená. Vlákna si takto vzájemně vylučují přístup k daným blokům. Struktura bufferu již byla vytvořena.

5.3 Odesílání dat

Odesílání dat je přímočaré. Řídil jsem se poučkou, že co funguje není třeba měnit. Při streamování se spustí odesílací vlákno. To kontroluje mutex na kruhovém bufferu. Pokud se podaří uzamknout, znamená to, že jsou data připravena k odeslání. Zamkne se druhý mutex na kruhovém bufferu, inkrementuje `timestamp` a data jsou odeslána po síti. Příslušná část bufferu se označí se jako odeslaná. Mutexy jsou uvolněny.

Při dodávání analyzovaných dat nedochází k odeslání v pravém smyslu. Výpočet probíhá bez ustání na všech příchozích datech. Ukládá se do sdílené paměti. Data se odesílají ve chvíli, kdy o ně někdo požádá.

V tuto chvíli dojde také k vynulování příslušných dat. Nestane se tedy, že by například některé peaky byly ignorovány. Na druhou stranu dochází k mírnému omezení uživatele. Data se změní každým přístupem. Neměl by přistupovat k datům ze dvou různých míst. Například mít otevřená dvě okna prohlížeče. Data v tom případě budou zkreslená a do každého půjde půl analýzy.

5.4 Streamer Controller

Jak napovídá název, touto třídou se ovládá Streamer. Je to takový střed celého systému, kde se potkávají data všech součástí systému.

Je zde spravováno síťové připojení, spouštěn hlavní cyklus. Zpracovávají se požadavky Recieveru. Tato třída zároveň poskytuje informace Webserveru. A to jak o stavu Streameru, tak o připojení.

Výzvu, kterou jsem tu řešil bylo, jak zpřístupnit ovládání z Recieveru a Web serveru zároveň. Nejvýhodnější řešení které mě napadlo je vytvořit něco jako handshake. Veškeré příkazy půjdou přes Reciever. Do něj jsem přidal možnost zpracování zpráv.

Průběh je následující. Uživatel zvolí ve webovém prohlížeči možnost nahrávání. Na port 80 se odešle požadavek pro zahájení nahrávání. Webserver pošle zprávu Recieveru „začni nahrávat“. Reciever zprávu zpracuje tak, že pošle Streamer Controlleru požadavek o začátek streamování dat. Analogicky funguje i zastavení, s tím rozdílem, že ukončení Recieveru a odpojení je detekováno a stream zastaven. Není třeba ukončovat zprávou. Při využití možnosti přehrávání je Controlleru předán seznam skladeb. V tomto případě nahrávání spouští i zastavuje Controller (zprávami přes Reciever).

Přehrávání zvuku

Před zahájením přehrávání je aktualizován soubor `.actualplaylist`. V něm je uložen seznam souborů. Ukládání v souboru jsem zvolil, aby si systém pamatoval poslední přehraný playlist. Ten se použije při spuštění přehrávání z Recieveru.

Jsou načteny názvy souborů. Soubory jsou postupně přehrávány příkazem `aplay`. Je vynucena vzorkovací frekvence 48000 Hz. Pokud náhodou soubor neexistuje, je přeskočen.

Při zahájení a ukončení přehrávání je odeslána zpráva s názvem přehrávaného souboru.

5.5 Reciever

Reciever nemá vlastní kapitolu, protože na něm nebylo třeba měnit mnoho věcí. Hlavní komplikaci jsem měl s překladem vložené knihovny, která vyžadovala standardní závislosti

pomocí nestandardní cesty k souborům. Byla to absolutní cesta z jiného systému. Instalací příslušné verze překladače, přepsáním makefile a přeskočením testů jsem vše zprovoznil.

Co se úprav týče, tak zásadní bylo zbavit se části kódu, která se starala o bitové posuny a zpracovávala 24 bitové vzorky. Nově není potřeba a využíváme přímo `float`, který má 32 bitů.

Další odstraňování se týkalo zastaralého nastavení gainů. Program vyžadoval cestu k souboru gainfile jako povinný parametr a jeho obsah odesílal ihned po inicializaci. To nově není potřeba. Rozhraní pro příjem nastavení tímto způsobem už není aktuální.

Abych nezůstal jen u odstraňování, přidal jsem nové rozhraní pro vzdálené ovládání, tak aby příjem dat mohl být řízený webovou aplikací. Ta může odesílat pokyny ke spuštění a zastavení nahrávání. Také se přenáší název audio souboru při spuštění a ukončení přehrávání jeho přehrávání.

Kapitola 6

Vizualizační a řídicí aplikace v internetovém prohlížeči.

Jde o webovou aplikaci. Není zpřístupněná přes WWW, ale pouze lokálně. Aplikace je načtena po zadání IP adresy a portu. Jsou přeneseny všechny soubory, o které webový prohlížeč požádá.

Styly a JavaScript knihovny jsou uloženy přímo ve vestavěném serveru. Pro správnou funkci nemusí být počítač připojený k internetu.

6.1 Bootstrap, JavaScript, jQuery, JSON

Vyjmenované technologie jsou ve webových aplikacích standardem a pravděpodobně jsou všem známé. Přesto bych rád v krátkosti vysvětlil jejich použití v projektu.

Bootstrap

Je zde kvůli vzhledu. Pro rozložení využívám mřížkový systém (grid layout). Pro zobrazení úrovní jsou použité *progress bary*.

JavaScript

Je využitý pro generování stránky i veškerou funkčnost. Část stránky se skládá z několika řádků, které se od sebe liší pouze identifikátory prvků. Proto jsou vytvořeny cyklem v JavaScriptu. Je tak možné vždy vygenerovat přesně počet audio vstupů, se kterými se pracuje.

jQuery

Tato knihovna je připojena kvůli Bootstrapu. Zajišťuje ale i zpracování veškerých uživatelských vstupů a pomocí *setTimeout* pravidelně získává nové hodnoty úrovní.

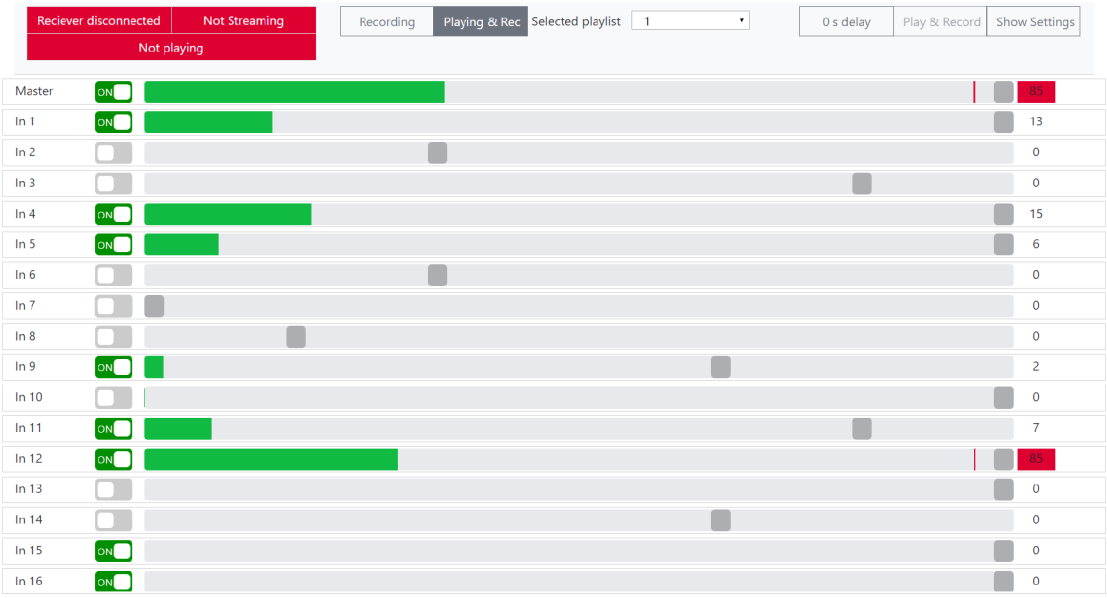
JSON

I JSON používám k tomu, na co byl vytvořen. V tomto formátu posílá server data. Používám jej, protože je úsporný a zpracování dat do proměnných v JavaScriptu je velmi snadné.

6.2 Vzhled a ovládání

Vzhled (6.2) je inspirovaný zvukařským mixážním pultem. Funkčnost totiž je velmi podobná. Převzal jsem myšlenku rozdělení jednotlivých vstupů. Každý vstup má své ovládací prvky. To je dobré pro rychlost ovládání a přehlednost. Vizualizace úrovní je také známá z různých aplikací pro práci se zvukem.

Zaměřil jsem se i na praktická hlediska s ohledem na tvorbu webových stránek. Celé zobrazení je například otočené o devadesát stupňů, aby bylo možné využít html *input range* bez transformací.



Obrázek 6.1: Ukázka webové aplikace.

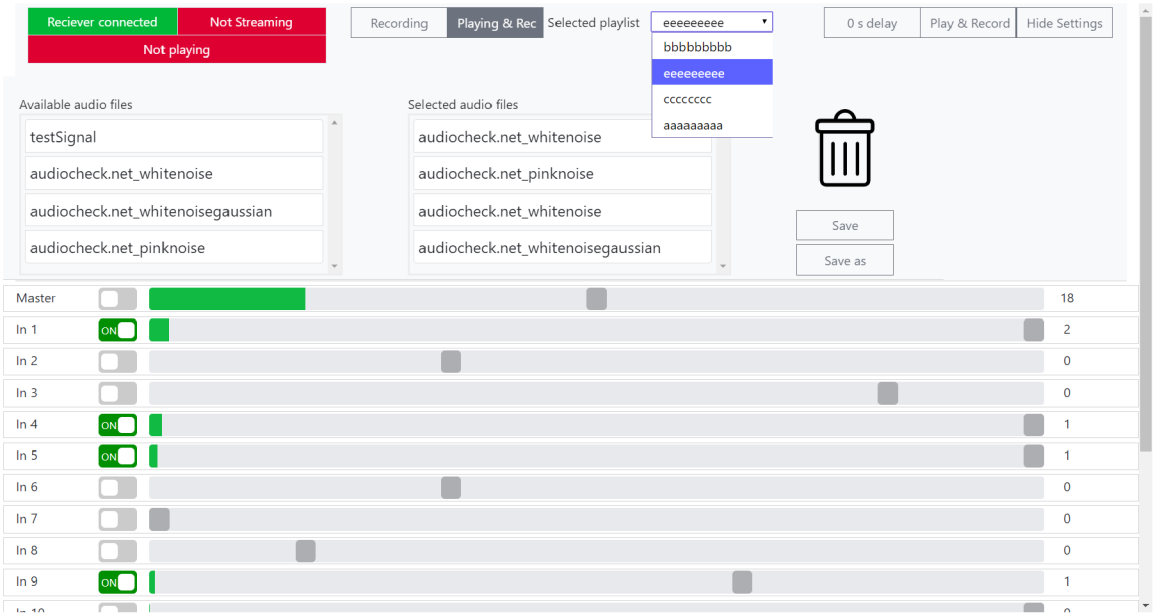
Nahoře se nachází stavové informace, ovládání a nastavení. V informacích se zobrazuje aktuální stav zařízení. Zda je připojený přijímač, jestli je aktivní streamování, případně jaký soubor se přehrává. Je zde přepínač pro výběr mezi samotným nahráváním a nahráváním včetně přehrávání. Je možné omezit dobu nahrávání. Je zde i tlačítko pro nastavení zpoždění. Po kliknutí iteruje přes hodnoty 0, 5, 10, 15 a 20 sekund. Po kliknutí na tlačítko "show settings"se zobrazí správa playlistů.

Hlavní část aplikace tvoří jednotlivé zvukové vstupy. Každý vstup je na jednom řádku. Je vidět název, přepínač phantomového napájení, zobrazení úrovně s „faderem“ pro nastavení gainu a vybuzení vstupu. Gain se udává v decibelech, hodnoty jsou 0-6 (0 = 0dB, 1 = 10dB, ... 6 = 60dB).

Ovládání je uzpůsobené pro myš a dotyk. Omezeně je možné využít i klávesnici (tabulátor pro pohyb, mezerník pro phantom a kurzorovými šipkami nastavení gainu).

6.3 Funkce

Po načtení stránky je odeslán požadavek *getall*. Tím se načte počet kanálů, seznam playlistů a souborů. Inicializují hodnoty phantom a gain. Aktualizace úrovní je prováděna periodicky ve zvoleném intervalu. Funkce pro nastavení hodnot je spuštěna vždy s uživatelským vstupem – kliknutí na phantom, nastavení gain, nebo uložení playlistu.



Obrázek 6.2: Ukázka webové aplikace – nastavení.

Interakce uživatele je zachycena událostmi *onchange* a *onclick*, které volají přislušnou funkci s parametry *id* a *hodnota*. Volaná funkce odesílá hodnoty na server.

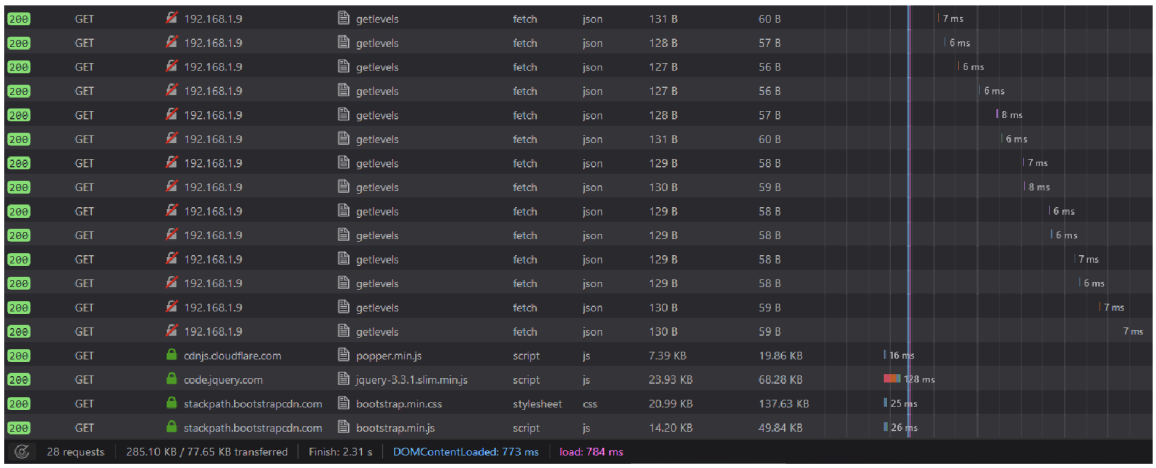
6.3.1 Správa seznamu stop

Byla přidána možnost prohlížet a upravovat seznamy stop. Po rozkliknutí nastavení se zobrazí možnost výběru playlistu. Zvolený playlist je zobrazen a je možné jej upravovat.

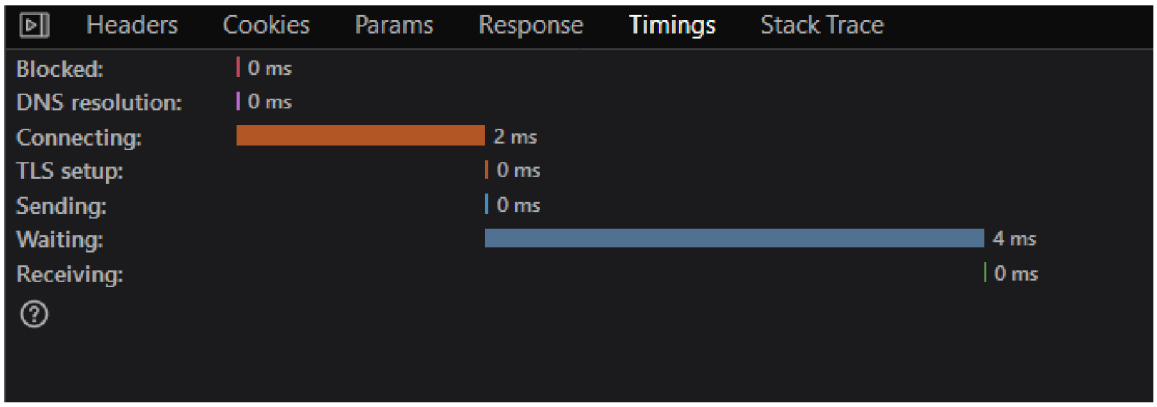
Možné úpravy jsou přidat a odebrat soubor, měnit pořadí. Je možné použít jeden soubor vícekrát.

Úpravy probíhají přetahováním položek. Přetáhnutím ze seznamu souborů na playlist je soubor přidán do seznamu. Přetáhnutím na ikonu koše je odebrán.

Palylist je možné uložit, případně uložit jako. Při zahájení nahrávání neuloženého seznamu se přehrají soubory aktuálně přiřazené ve webovém rozhraní. Ne ty, které jsou uložené v aktuálním playlistu.



Obrázek 6.3: Doba zpracování jednoho požadavku.



Obrázek 6.4: Požadavky webové aplikace.

6.4 Stabilita a rychlost

Očekává se zobrazení v reálném čase. Definovat reálný čas není snadné a elektronické zpracování vždy přinese nějaké zpoždění. Aplikace by neměla být nejpomalejším článkem řetězce. Rozhodl jsem se porovnávat její rychlost s obnovovací frekvencí monitoru. Zvolil jsem standardní 60hz monitor. Zobrazuje 60 snímků za sekundu. Snímek potřebuje zhruba každých 16 ms. Zpoždění aplikace měří snadno vývojářský režim webového prohlížeče. Na výpisu sítě 6.3 je vidět, že vyřízení požadavku od odeslání do přijetí trvá mezi šesti a osmi milisekundami. Časování průměrného požadavku je vidět na obrázku 6.4. Dvě milisekundy trvá připojování. Tento čas je možné zkrátit navrženou optimalizací. Prozatím je výkon dostatečný a není třeba aplikaci optimalizovat. Pokud se ukáže být zpracování dat serverem náročnější a tato doba se neúměrně prodlouží, bude muset předávání informací o úrovních probíhat jiným způsobem.

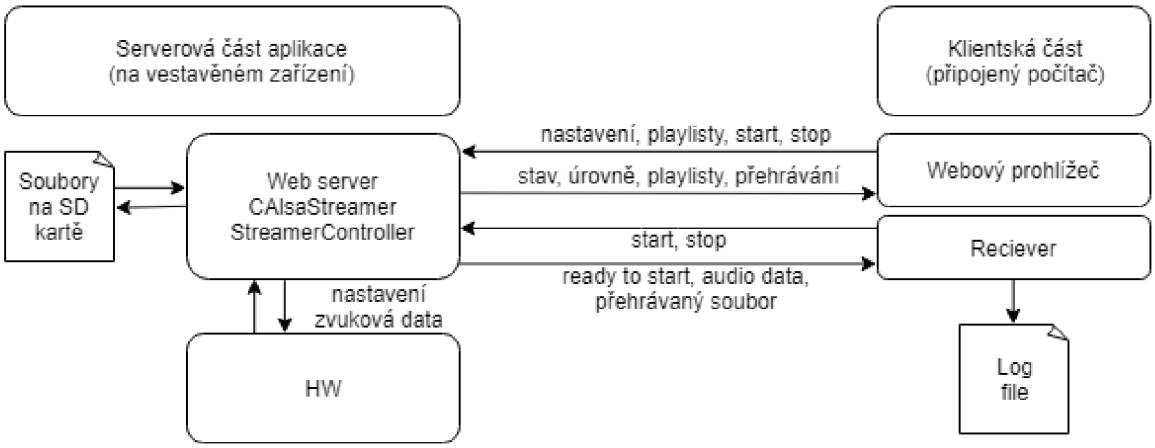
Se stabilitou jsem žádné problémy nezaznamenal. Pokud je server odpojen, přestanou se posílat data. Při opětovném připojení je třeba stránku aktualizovat a pokračuje jako by k výpadku nedošlo.

Po aktualizaci aplikace získá aktuální stav HW a tak nedojde k žádné inkonzistenci mezi reálným stavem a stavem aplikace.

Kapitola 7

Popis výsledné aplikace a práce se systémem

V kapitole 2.5 jsem popisoval stav systému ve chvíli, kdy jsem začínal pracovat. Později jsem zjistil, že současný stav je horší, než popisovaný. K fungující verzi totiž chyběla dokumentace a funkční byla na starší revizi zařízení, kde se používal jiný přístup k datům. Zde budu popisovat výsledný stav, použití, ovládání a některá specifika řešení. Na obrázku 7.1 je znázorněná komunikace jednotlivých částí systému po úpravách.



Obrázek 7.1: Scháma komunikace po úpravách.

7.1 Inicializace systému

Při připojení napájení se nabootuje systém, a ihned se spustí server na IP adrese 192.168.10.9. Počítač musí pro připojení mít na daném síťovém rozhraní IP adresu stejné sítě.

Systém je v tuto chvíli ve stavu před inicializací – ta se dokončí až po připojení přijímače. V tuto chvíli je možné ve webovém rozhraní spravovat playlisty a nastavovat parametry. Zbytek funkcí je zpřístupněn právě až po připojení přijímače. Celý systém slouží k nahrávání, proto používání bez přijímače nedává smysl. Hodnoty úrovní se aktualizují každých 100 milisekund, tedy desetkrát za sekundu.

7.2 Nastavení parametrů

Po inicializaci jsou parametry nastavené tak, aby s nimi bylo co nejméně práce. Používají se mikrofony, které využívají phantomové napájení. Phantom je tak (nestandardně) aktivován už při zapnutí. Gain je nastaven na střední hodnotu, 30 decibelů.

Tato nastavení je možné měnit v reálném čase. Změna nastavení se okamžitě projeví na zobrazené úrovni. K tomu je zde ještě master - ten upravuje všechny kanály. Jeho změna gainu se relativně projeví na všech vstupech (například pokud se master gain zvedl o dva, ke gainu všech kanálů se přičte 2 – pokud nepřekročí maximum. Analogicky funguje i snižování.)

7.3 Nahrávání a přehrávání

Princip nahrávání funguje tak, jak je vysvětleno výše. Spustit i zastavit nahrávání je možné jak z Recieveru, tak z webového rozhraní. Ve webovém rozhraní je i možnost samospouště – spuštění s časovým odkladem a zvolit délku nahrávání – čas, po kterém bude nahrávání zastaveno. Při zvolení funkce nahrávání s přehráváním je činnost automaticky zastavena po přehrání posledního souboru v playlistu. Zároveň jsou do Recieveru posílány informace o aktuálně přehrávaném souboru. Ty jsou ukládány do souboru zvoleného parametrem *logfile*. Když je aktivní nahrávání, obnovovací frekvence aplikace je snížena, aby nezabírala přenosové pásmo a výkon serveru. Ty jsou vytíženy přenosem dat.

7.4 Nahraná data – raw formát

Tok dat je opět rozdělen na jednotlivé kanály. Každý kanál se ukládá do jednoho binárního souboru. Jsou to surová data (raw formát), vzorky jsou formátu float. Kanály jsou synchronizované. Všechny soubory jsou stejně velké – obsahují shodný počet vzorků. N-té vzorky všech kanálů byly získány v jeden okamžik.

Tento formát dat neukládá metadata. Není možné soubor jen tak otevřít a přehrát. Přehrávači chybí informace o způsobu uložení dat a vzorkovací frekvenci. Pro rekonstrukci nahrávaného signálu je třeba tyto informace dodat.

Pro přehrání je možné využít například open-source program Audacity. Jeho funkce „nahrát - původní data“ je vytvořena přesně pro náš případ. Po označení souboru je nutné vyplnit i zmíněné dodatečné informace. Data jsou ihned graficky zobrazena a je možné si poslechnout nahraný zvuk.

Kapitola 8

Testování a dosažené výsledky

Tento systém je považován za profesionální nástroj ke zpracování zvuku. Testování bylo velmi důležitou částí práce. Kvůli novému způsobu přístupu k datům je třeba ověřit, že se jsou zpracovány opravdu všechny vzorky a data se nikde neztrácejí. Nejvíce jsem se zaměřil fázi, kdy jsou data na zpracována systémem Linux. Prakticky to je jediná část, kterou jsem mohl ovlivnit. Zároveň zbytek systému dělají opravdoví profesionálové v Audified, tak mám důvěru v bezchybnou funkčnost zbylých částí.

8.1 Nástroje a způsob testování

Vzorků je takové množství, že není možné kontrolovat je manuálně. Musel jsem připravit několik způsobů, jak zjistit že nahráváme správná data. Pro účely testování jsem rozdělil zpracování na několik fází.

1. Získaná data – vyčítání dat z Alsa zařízení
2. Odesílaná data – správnost uložení do kruhového bufferu, ovlivnění posíláním
3. Přijatá data – výsledek a ověření celé funkčnosti.

Postupně jsem se propracovával dál v řetězci zpracování dat.

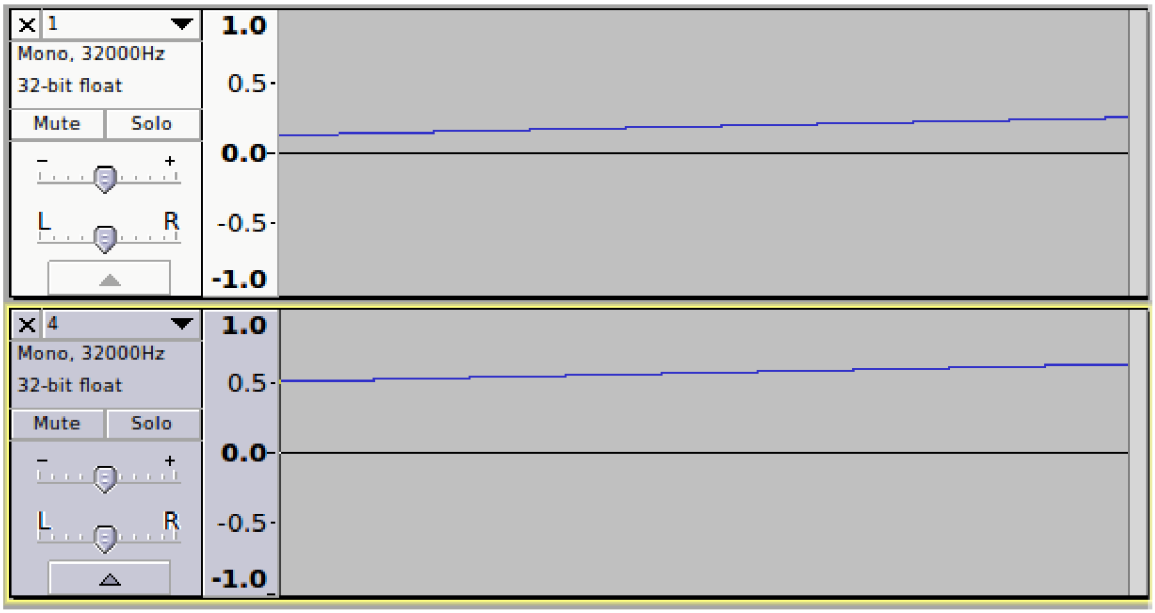
8.1.1 Testovací programy v C

Vytvořil jsem dvě aplikace, které jsem využil k testování. Nezabýval jsem se přívětivostí pro uživatele. Veškerá nastavení jsou přímo ve zdrojovém kódu, využívají se konstanty. Celé použití je velmi specifické a jednoúčelové.

Writer

První z programů vytváří vzorky. V této fázi testování jsem ještě pracoval na vzorcích, které byly 16 bit integery. Program slouží ke generování umělých dat, na kterých lze kontrolovat správné přiřazení kanálů, pořadí vzorků, případné vynechání nějakého vzorku i synchronizace kanálů.

16 bitů jsou čtyři hexadecimální číslice, každá kóduje 4 bity. Při otevření souboru hexadecimálně, například programem *hexdump*, jsou přehledně vidět tyto čtveřice. Každá čtveřice čísel reprezentuje jeden vzorek.



Obrázek 8.1: Zobrazení vybraných testovacích stop.

Vzorky jsou generovány tak, že první 4 bity (podle MSB) označují číslo kanálu. To dává možnost rozlišit od sebe 16 různých kanálů. Zbytek čísla je využitý jako čítač. Od vzorku `X000`, po `Xfff`, kde X je číslo kanálu. Celkem 4096 unikátních vzorků pro kanál. Program ukládá vygenerované vzorky do souboru (obrázek 8.1).

Reader

Má obrácenou funkci. Čte soubor s daty a kontroluje, jestli obsahuje správné (očekávané) hodnoty. Čtený soubor nemusí začínat prvním vzorkem. Při nahrávání může dojít k určitým zpozdením. Prvním vzorkem se Reader nastaví. Potom kontroluje, že každý následující je o jedna větší (případně nula po dosažení hodnoty max), než vzorek předchozí.

K ověření funkčnosti těchto aplikací jsem využil vizuální kontrolu vygenerovaných souborů programem *hexdump*. Následně vzájemnou kontrolu: Výstup Writeru připojený na vstup Readeru musí být označen jako validní.

8.1.2 Audacity®

Audacity[1] je editor a nahrávací program pro práci se zvukem. Podporuje více stop (kanálů), je otevřený, distribuovaný zdarma, open-source – pod licencí GPL. Autoři nabádají uživatele ke zkoumání funkcí a vylepšování programu. Jsou vytvořeny distribuce pro nejpoužívanější operační systémy.

V této práci jsem využíval program k

- vizuální reprezentaci dat
- přehrávání
- vytváření wav a raw souborů

[1] Audacity® software is copyright © 1999-2019 Audacity Team. The name Audacity® is a registered trademark of Dominic Mazzoni.

- odečítání stop

Vizuální reprezentace

Při nahrání dat je zobrazen jejich časový průběh. Je to rychlá metoda, jak poznat jestli je výsledek podobný jako zdroj, nebo je něco úplně špatně a data se někde poničila. Celou dobu jsem se potýkal se zarovnáním dat tak, aby bylo možné je porovnat. V testovacích souborech je to snadné a stačí najít shodné číslo. U reálných zvuků je ale velmi náročné je spolehlivě automaticky zarovnat. Nejvýhodněji se mi jeví nejjednodušší metoda a to podívat se na průběh signálu a zarovnat data manuálně. Ideální je, když se v nahrávce vyskytují nějaké přechody mezi tichem a vysokou hlasitostí. Podle nich je zarovnání po troše tréningu velmi snadné.

Přehrávání

Je pěkné vědět, že za čísly se schovává opravdový zvuk. Porovnání poslechem sice nedává žádné objektivní výsledky, ale při odhalování chyb mi tato metoda velmi pomohla. Velmi dobře jsou poznat, kdy je při nahrání vynecháno několik vzorků. Nahrávka „přeskočí“, nebo hlasitě lupne. Krásně rozeznatelný byl i problém s buffery, kdy se část kopírovala stále dokola a z řeči se stalo koktání.

Vytváření wav a raw souborů

O surových datech jsem se již rozepisoval v podkapitole 7.4. Formát wav snad není nutné představovat tak detailně.

Známý je hlavně pro použití na CD. Je odvozen z RIFF, data mohou být ve více formátech. Drtivě nejpoužívanější jsou ale nekomprimovaná data pulzní modulace. Je to bezeztrátový formát, jeho zpracování je snadné a výpočetně nenáročné. Jeho hlavní použití je v oblastech pro zpracování zvuku, jako výchozí formát před další konverzí či pro archivaci či přenos zvukových dat v nejvyšší kvalitě. Díky rozšíření a jednoduché vnitřní struktuře je formát WAV používán pro přenos zvukových dat mezi různými systémy.

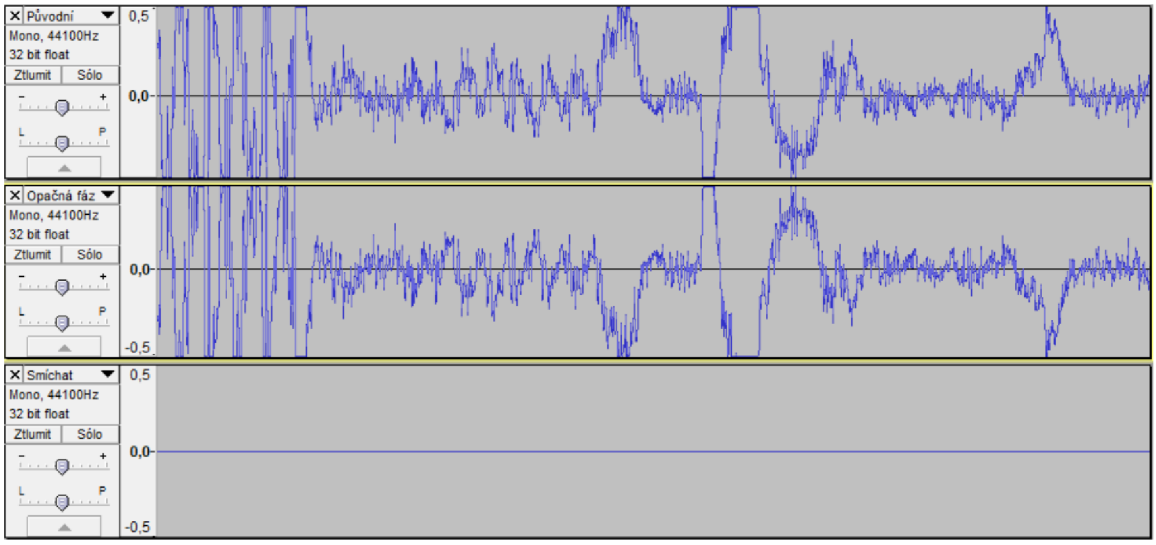
Z výše popsaného se dá odvodit, že formáty jsou si velmi podobné. Obrovská výhoda wav formátu je to, že obsahuje i metadata a tak je snadno přenositelný. Příkladem může být třeba, že po otevření tohoto formátu v Audacity® se automaticky načtou jednotlivé stopy, ostatní přehrávače jej dokáží přehrát bez dalších informací. Ze všech přehrávačů bych vyzdvihl *aplay*, který je vestavěný v linuxovém systém a zvládá přehrávat oba zmíněné formáty.

Audacity® používám pro převod mezi zmíněnými formáty. Zároveň jej využívám k vytváření vlastních souborů, ať už poskládaných z jednotlivých vzorků, nebo převodem z jiného formátu.

Odečítání stop

Zde jsem se přiblížil fyzikální stránce věci. Zvuk je vlnění, vnímáme ho díky rozkmitaným molekulám vzduchu. Kmitání má určitou fázi. Mluvíme o počáteční fázi, fázovém posunu, opačné fázi. Zde by mohl být velmi technický popis, chci se dostat pouze k malé části a tou je právě opačná fáze.

Když mají dva signály opačnou fázi, tak se navzájem vyruší. V praxi je možné to pozorovat u reproduktorů. Když jeden reproduktor je zapojený správně a druhý obráceně,



Obrázek 8.2: Kombinace stop s opačnou fází.

hrají s opačnou frekvencí a jejich zvuk je utlumený. Tato technika se používá i ve sluchátkách s potlačením hluku. Poslouchají okolí a snaží se generovat právě signál s opačnou fází. Proto nejlépe fungují na konstantní zvuky, jako například motor letadla – zvuk se nemění a sluchátka jej mohou spolehlivě vyrušit.

Já jsem tuto techniku používal na zjištění, zda jsou stopy shodné. V Audacity® je funkce pro otočení fáze i sloučení stop. Pokud dám pod sebe přehranou a nahranou stopu a časově je synchronizuji, tak po otočení fáze jedné z nich a jejich sloučení vznikne nulový výsledek. viz obrázek 8.2.

8.1.3 Wireshark

I tento program vytvářejí dobrovolníci a je pod licencí GPL. Slouží k analýze sítí. Podobně jako Audacity® je multiplatformní a nejpoužívanější ve svém oboru. Používal jsem ho na kontrolu připojení, posílaných paketů a dat v nich. Také na kontrolu zahlcení sítě a přenosové rychlosti.

8.1.4 Feeder

Feeder je jednoduchá aplikace napsaná v C++. Pro účely ladění byl reálný vstup ze zvukových vstupů nahrazen souborem. Všechna data zapsaná do tohoto souboru fungují v systému stejně, jako by byla přímo z mikrofonu.

Program Feeder simuluje vstupy tak, že čte připravené audio soubory a ve smyčce je zapisuje do `/dev/sharc_0`.

Pro ladění je velmi vhodné, že můžeme ovlivnit vstup a poslat na něj přesně data, která potřebujeme. Tím simulovat laboratorní podmínky.

Potenciální problém může nastat, pokud se vykytuje chyba, kterou by zamaskovalo právě periodické opakování signálu. Například buffer stejné velikosti jako přehrávaný soubor, který by se naplnil jednou a potom odesílal stále stejná data. Tomu lze předejít použitím různě dlouhých souborů, aby se chyba projevila alespoň v některém. Zrádné je v tomto případě používání mocnin dvou. Doporučuji náhodnou délku.

8.1.5 Soubory a formát dat

O různých formátech jsem se rozepisoval v předchozích kapitolách. Stejně mi to nedá a ještě bych zde rád shrnul formáty, které jsem v průběhu práce a ladění používal.

1. Feeder pracuje s 16 bit raw soubory. Pro každý kanál jeden soubor.
2. Z Alsy jsou nahrávána data v prokládaném raw formátu. V počátečních fázích 16 bitový fixed point, potom jsem přešel na float 32 bit.
3. V tomto formátu s nimi pracuje celý server, kontrolní výstupy do souborů též zachovávají tento formát. Pro přenos po síti se data rozdělí do paketů, ale jsou stále stejná.
4. Reciever je nakonec přijme a rozdělí na jednotlivé kanály. Výsledkem je tedy jeden raw soubor pro každý kanál.
5. Pro přehrávání se využívají wav soubory. Tím pádem není potřeba další programová podpora. Soubor si uživatel upraví podle potřeb a při přehrávání se použije konfigurace uložená v souboru. Je potřeba dodržet vzorkovací frekvenci, ale například počet kanálů je univerzální.

8.2 Získaná data

První zastávka dat. Výpis do souboru ihned po získání dat. Data se získávají příkazem `snd_pcm_readi()`, který by měl být robustní a bezchybný. Takže jde hlavně o kontrolu konfigurace a zjištění, jestli je cyklus dost rychlý a nevynechává data (funkce na čtení je blokující, tedy synchronizuje provádění). Kdyby se ale volala příliš pomalu, mohl by přetékat zabudovaný buffer.

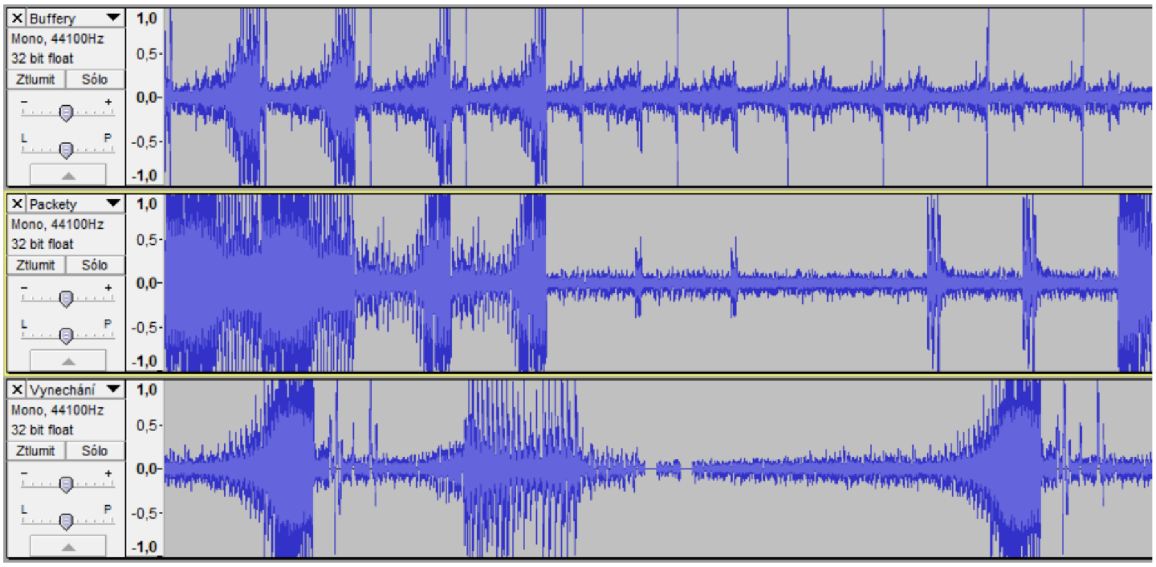
Používají se umělá data vygenerovaná Writerem, data jsem porovnával Readerem, které jsem popsal v předchozí kapitole.

Prvotní problém byl, že to nevycházelo. Této fázi se snad ani nedá říkat testování, myslet si, že to vyjde napoprvé bylo naivní. A tak jsem se pustil do ladění. V této fázi jsem se se software seznamoval a přistupoval jsem k tomu spíše chaoticky, než podle jasného plánu. Po pročtení dokumentace k Alse, upravení parametrů a změně velikosti bufferů, čtených dat a odpojení Recieveru začalo vše dávat větší smysl. Nakonec začala data opravdu procházet i mým testem. Dalo se tedy usoudit, že byla nahrávána správně.

Dříve jsem naznačil, že tato testovací metoda je chybná. Přijít jsem na to musel vlastní zkušeností, kdy syntetická data velikosti 4096 vzorků pro každý kanál fungovala spolehlivě, ale jakákoliv jiné testovací nahrávky byly poškozené, deformované. Problém byl v implementaci bufferů.

Používání bufferů bylo vůbec zpracováno velmi nepřehledně. Opět začalo dělat problémy, když jsem začal posílat data do sítě. V tuto chvíli jsem se rozhodl, že konstanty jsou sice pěkné, ale celek je zbytečně složitý tím se znepřehledňuje. Buffer jsem přepsal tak, aby měl stejnou velikost jako nahrávaná data. Paket obsahuje stejná data. To se mi ze začátku vůbec nezdálo jako rozumné řešení, ale praxe ukázala, že to funguje výborně.

Dalším krokem byla možnost vynechat buffery, sloužit odesílací a čtecí vlákno. Když už je nahrávání a odesílání 1:1, tak mi to přišlo jako dobrý nápad. Bohužel se zanedlouho ukázal jako nefunkční. Nevím přesně, jestli to způsobovaly chybějící buffery (nic tomu nenasvědčuje, podle výpisů se nepoužívá víc než jedno pole a Alsa buffery by to měly



Obrázek 8.3: Ukázka různě poškozených dat.

pokrývat), nebo problémy s posíláním do sítě. Každopádně se toto řešení neukázalo jako praktické a tak jsem od něj ustoupil.

Největší problém se ukázal po zapracování všech částí systému. Data začaly vynechávat. A to tak, že chvilku jsou nahrávány pouze nulové vzorky. Po důkladném testování jsem nedostal úplně jednoznačnou odpověď na to, co to způsobuje. Respektive neumím spolehlivě otestovat své domněnky, ale problém jsem odstranil.

Fakta jsou, že při změně velikosti paketů se mění délka intervalu, kdy se nenahrává. Hluchá místa vznikají na místě, kdy se začne přehrávat audio soubor. Když byl připojený špatně implementovaný Reciever, který posílal příliš mnoho požadavků, problém byl mnohem větší (nemuselo se přehrávat). Problém se projevoval i bez spuštěného Recieveru.

Podle mého vše ukazuje na to, že byl problém s výkonem. Program `top` neukazoval vytížení procesoru větší, než 80%. Síť se nedostávala do problematických čísel.

Každopádně po mírné optimalizaci a omezení rychlosti aktualizace zobrazovaných dat se problém přestal projevovat.

Důležité je podotknout, že byly nahrány nulové vzorky. Proto jsem dospěl k závěru, že byl omezený výkon programu Feeder, který nestíhal dodávat data do souboru, ze kterého se vyčítají. A tak byly nahrazeny nulami.

K tomu mě vede jednak to, že Alsa nedávala žádné chybové údaje (při jakékoliv chybě se ukončí program, aby se zamezilo chybné načítání dat). A také to, že se chyba nikdy neobjevila s reálnými vstupy.

8.3 Odeslaná data

Neplánovaně jsem problémy s odesílanými daty popsal v předchozí části. Výpis do souboru mám ve stejném cyklu, jako odesílání. Při debugování jsem tímto výpisem buď simuloval odesílání, nebo porovnával odeslaná data s těmi přijatými Recieverem.

Zajímavé je také porovnání velikostí souborů nahraných dat. Samozřejmě není to úplně stoprocentní metoda. Na debugovací soubory není použitý `flush`, tak mohou nějaká data

zůstávat nezapsaná. I přes určitou nepřesnost je pěkné zjistit, jaké zpoždění získávají data mezi těmito dvěma vlákny.

Další metoda, která není úplně přesná je porovnání velikosti dat a času nahrávání. Vstupuje chyba měření, která se dá velmi dobře minimalizovat opakovaným měřením a delšími úseky měření. Samozřejmě nikdy nebude metoda přesná na počet vzorků. Můžeme se dostat na přesnost na sekundy. Podle velikosti vzorku a vzorkovací frekvence jde vypočítat přesná očekávaná velikost souboru za určitý čas. Že se neztrácejí jednotlivé vzorky potvrdily syntetické testy. Zde se testuje, zda nedochází ke ztrátě paketů, nebo jiným větším chybám.

8.4 Přijatá data

Kritické u přijatých dat je opět kontrolovat, zda se náhodou někde neztrácejí pakety. Stejně tak je potenciální problém správné rozdělení kanálů a bezchybné ukládání do souboru. Porovnáváním dat před posláním a po poslání usnadňuje Audacity. Kde při importu raw dat můžeme zvolit, že importovaná data obsahují až 16 kanálů. Jednotlivé kanály už potom není problém porovnávat s jednotlivými soubory výsledných přijatých dat.

Ve výsledku není důležité, co se děje mezi nahráním a přijetím dat. Důležité je, aby výsledek odpovídal očekávání. Proto naprosto zásadní fáze testování byla vrácení se na začátek a porovnání přijatých dat s daty vytvořenými pro testování a poslanými na zvukový vstup.

Konečné testování s mikrofony už byla jen odměna, kdy se ukázalo, že vše funguje a otestované principy platí i pro reálná data.

8.5 Přehrávání

Tuto část jsem otestoval pouze v software. Hardware, který mám k dispozici bohužel nemá kartu pro vstupy. Hardware jsem se snažil získat, ale v Audified se nám nepodařilo jej zprovoznit. Není otestovaná přímo na hardware. Vše ale nasvědčuje tomu, že přehrávání funguje.

1. Audified má otestováno, že hardware zvládá propustnost 16 + 16 kanálů. Víc nebylo testováno.
2. Software se chová, jako by přehrával – posílá do ovladače data. Více výkonu potřebovat nebude.
3. Při testování s ovladačem přesměrovaným do souboru vše fungovalo.

Možná jen dodám, že posílání informačních paketů způsobuje chyby ve výpisu Recieveru - ukazuje je jako `Drop rate`. Vždy při ukončení přehrávání a opětovném spuštění se odešlou dva pakety a Reciever vypíše `Drop rate: -2`.

8.6 Výkon aplikace

Na výkon z pohledu webové aplikace jsem se zaměřil v kapitole 6. Je tam popsaná především latence. U části aplikace, která běží na ARM procesoru má smysl zjišťovat, kolik zabírá prostředků. Zkoumal jsem se na využití paměti a procesoru. Souhrn je v tabulce 8.1.

Vysoké paměťové nároky má streaming také na paměť připojeného zařízení. Každá minuta záznamu znamená přibližně 175 MB. Terabytový disk pojme cca 12 hodin záznamu.

Stav zařízení	využití paměti [%]	využití procesoru [%]
Po spuštění	0	5
Reciever připojen	11	15
Stream	14	85 – 95

Tabulka 8.1: Využití prostředků pro 16 kanálů

8.7 Shrnutí výsledků

Na obrázku 8.3 je ukázka několika různých poškození dat, se kterými jsem se setkal. V první stopě je ono popisované koktání. Je vidět, že část se periodicky opakuje, postupně se blíží nule. Ve druhém vzorku je podobný problém všimněme si shodných dvojic. Ve třetí stopě jsou vidět vynechaná místa.

Dále se pokusím popsat, jak vše dopadlo. Už se vyvaruji rozboru jednotlivých částí a pokusím se hodnotit systém z pohledu uživatelských zkušeností.

Většinu testování jsem prováděl na základní desce. Po odladění jsem otestoval vše i s připojenými hardwarovými vstupy. K dispozici bylo zařízení se šestnácti vstupy. Proto to bylo i maximum, které jsem používal.

Testování potvrdilo, že systém funguje téměř tak, jak má.

- Z Alsy jsou vzorky vyčítány spolehlivě.
- Vzorky jsou správě odesílané po síti.
- Přenosová rychlost se pro 16 kanálů pohybuje v rozmezí 2,9 – 2,95 MBs^{-1}. Očekávaná průměrná hodnota je 2,93 MBs^{-1}.
- Nahrané soubory jsou kompletní.
- Frekvenci aktualizace zobrazování úrovní jsem omezil na 10Hz. Větší rychlost je možná, ale subjektivně jsem u rychlejší aktualizace nepoznal rozdíl. Není potřeba více zatěžovat počítač.
- Nastavení parametrů funguje spolehlivě.
- Přehrávání funguje na základní desce. Zřejmě nic nebrání tomu, aby fungovalo i na hardware.
- Fungují pouze tři ze čtyř desek se vstupy. Zvuk reálně jde pouze z 12ti vstupů.

Kapitola 9

Závěr

V práci jsem se seznámil s dostupným HW a SW. Navrhl možné řešení a ukázal, že funguje. V rámci semestrálního projektu byla vytvořena část serverové aplikace a většina klientská aplikace. V průběhu dalšího semestru jsem práci dokončil.

9.1 Vlastní přínos

Svůj přínos vidím především v tom, že jsem vytvořil použitelnější ovládání systému a tím snad ušetřil čas všem, kteří jej budou využívat.

Dále jsem se seznámil se SW, který se používá, ale není podporovaný na novějším typu zařízení. Zároveň nebyl nikdo z autorů dostupný pro úpravy. Tento kód jsem dokázal pochopit a aktualizovat pro současné požadavky.

Webová aplikace

Vytvořil jsem kompletní webovou aplikaci pro zobrazení informací, nastaveni parametrů a práci s playlisty. K této aplikaci jsem navrhl a implementoval komunikační rozhraní. Zobrazení úrovní se aktualizuje každých 100 milisekund. Zpoždění je kolem 10 milisekund. Dá se říct, že jde o zobrazení v reálném čase.

Web server

Webová aplikace komunikuje se serverem, který běží přímo na přiděleném HW. Tento server jsem vytvořil z příkladu příkladu uvedeného u Boost knihovny. Bylo třeba jej kompilovat pro vestavěný Linux.

Přidal jsem třídu na vyřizování požadavků, napojil jsem ovládání gain a phantom. Pro podporu zobrazení úrovní a stavových informací jsem připojil i současný systém na získávání dat a streamování.

Streamer a Reciever

Tady jsem narazil na situaci, kdy současné řešení už nebylo aktuální. Rozhodně nejsem autorem celého kódu. Všude jsem ale musel upravovat kritické části. U Streameru to bylo vyčítání dat z ALSY a jejich balení do paketů. Reciever bylo třeba upravit pro aktuální formát dat. Celý tento SW jsem potom upravoval pro příjem konfiguračních zpráv a předávání dat web serveru.

Main

Celý systém je propojený do jednoho programu. Vymýšlel postup spouštění vláken a správné předání parametrů jednotlivým objektům.

9.2 Budoucnost

Celý systém je živý a stále se vyvíjí. Proto je už nyní jasné, že jde mnoho věcí vylepšit. Primárně by to bylo zkompletování hardware tak, aby měl dostatek vstupů i výstupů. Během implementace jsem se nemohl ubránit pocitu, že je streamování přes síť do počítače krok navíc.

Osobně by mi přišlo zajímavé zprovoznit USB port a nahrávat data přímo na připojený externí disk. V tom případě by nemusel být počítač neustále připojený. Stačilo by jej používat ke zjišťování stavu. Už v tuto chvíli je možné nahrávat data přímo na SD kartu (pravda není to moc praktické, protože se k ní složitě dostává).

Literatura

[1] ABRAHAMS, D.: *Boost.org*. [Online; navštíveno 16.12.2018].
URL `https://www.boost.org/`

[2] ANALOG DEVICES: *ADSP-SC589*. [Online; navštíveno 10.12.2018].
URL `https://www.analog.com/en/products/adsp-sc589.html`

[3] ARM Developer: *Cortex-A5*. [Online; navštíveno 10.12.2018].
URL `https://developer.arm.com/products/processors/cortex-a/cortex-a5`

[4] KACHRŇÁK, D.: *Procesory ARM: Základ nové éry*. [Online; 08.06.2010].
URL `https://www.zive.cz/clanky/procesory-arm-zaklad-nove-ery/sc-3-a-164061/default.aspx`

[5] KOHLHOFF, C. M.: *HTTP Server*. [Online; navštíveno 04.11.2018].
URL `https://www.boost.org/doc/libs/1_69_0/doc/html/boost_asio/examples/cpp11_examples.html#boost_asio.examples.cpp11_examples.http_server`

[6] TIŠNOVSKÝ, P.: *Mikroprocesory s architekturou ARM*. [Online; 06.03.2012].
URL `https://www.root.cz/clanky/mikroprocesory-s-architekturou-arm/`

[7] VÁŇA, V.: *ARM pro začátečníky*. Praha: BEN - technická literatura, 2009, ISBN 978-80-7300-246-6.

Příloha A

Obsah přiloženého paměťového média

A.1 Software

A.1.1 Zdrojové soubory k implementovanému systému

A.1.2 Programy a knihovny pro křížovou kompilaci

A.1.3 Readme

A.2 Videoprezentace práce

A.3 Plakát