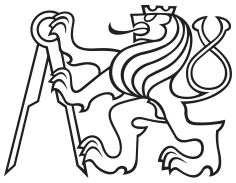
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

SW toolkit pro mikrofonní pole

Tomáš Formánek

Vedoucí: Ing. Petr Honzík, Ph.D.
Studijní program: Softwarové inženýrství a technologie
Květen 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Formánek** Jméno: **Tomáš** Osobní číslo: **483473**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

SW toolkit pro mikrofonní pole

Název bakalářské práce anglicky:

SW toolkit for microphone array

Pokyny pro vypracování:

Navrhněte a implementujte SW toolkit pro experimentální mikrofonní pole. Implementaci proveďte a otestujte na sférickém mikrofonním poli s 16 MEMS mikrofony s TDM výstupem připojenými přes adaptér na I2S sběrnici jednodeskového počítače Raspberry Pi. Navrhněte a implementujte zpracování a transport dat z těchto mikrofonů do některého standardního formátu zpracovatelného ve standardních DAW.

Seznam doporučené literatury:

Vagner D., Návrh sférického mikrofonního pole. Bakalářská práce. ČVUT FEL 2021. Dostupné z http://hdl.handle.net/10467/94677

INVENSENSE, INC., ICS-52000: Low-Noise Microphone with TDM Digital Output [online]. 2017-4-14. 20 s. Dostupné z: https://invensense.tdk.com/download-pdf/ics-52000-data-sheet/. Rev. 1.3.

Broadcom Corporation, BCM2837 ARM Peripherals, Cambridge, 2012, 205 s.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Honzík, Ph.D. UBI FD ČVUT

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **02.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Petr Honzík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Děkuji Ing. Petru Honzíkovi, Ph. D. za ochotu a trpělivost při vedení této práce, a to zejména při řešení komplikací s hardwarem. Děkuji Ing. Františku Rundovi Ph.D. za vedení semestrálního projektu, ze kterého tato práce z veliké části vychází. Děkuji také Davidovi Vagnerovi za pomoc při práci s mikrofonním polem.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 12. května 2022

Abstrakt

Práce se zabývá návrhem a implementací softwarového toolkitu pro experimentální mikrofonní pole. Řešení se skládá ze dvou částí, a to backendu pro řídicí jednotku a klienta pro uživatelský stroj. Toolkit zajišťuje sběr dat z mikrofonního pole, jejich přenos na uživatelský stroj a následné uložení ve formátu zracovatelném v běžném postprodukčním softwaru. Práce se dále zabývá testováním mikrofonního pole vzniklého v rámci bakalářské práce Davida Vagnera. Uvádí možnosti dalšího vývoje.

Klíčová slova: Raspberry Pi, Mikrofonní pole, Time Division Multiplexing, I2S

Vedoucí: Ing. Petr Honzík, Ph.D.

Abstract

The thesis describes the design and implementation of a software toolkit for an experimental microphone array. The solution consists of two parts: backend for control unit controlling the array and client for a user machine. The toolkit can read data from the microphone array, transfer the data to the user machine and store the data to a format which is suitable for a common audio postproduction software. The thesis also describes testing of the provided microphone array which was created by David Vagner. There are also suggestions for future development.

Keywords: Raspberry Pi, Microphone array, Time Division Multiplexing, I2S

Title translation: SW toolkit for microphone array

Obsah

Obrázky

Kapitola 1

Úvod

Cílem této práce je vytvořit softwarový nástroj, který umožní zaznamenat data z mikrofonního pole, přenést je na uživatelský stroj a uložit je ve formátu dále zpracovatelném v programech určených pro zvukovou postprodukci.

Ve spolupráci s Davidem Vagnerem (ČVUT FEL) a Danilem Rekhtinem (VŠUP) pracujeme na vývoji univerzálního nahrávacího zařízení pro pořizování zvukového záznamu s využitím sférického mikrofonního pole. Zařízení by mělo umožňovat například záznam ambisonické nahrávky, nebo třeba beamforming. Na trhu se již objevují podobná zařízení, která jsou však určena především pro studiové využití a jsou často velmi nákladná. Náš projekt se zaměřuje na dostupnost výroby a modularitu řešení. Našim cílem je zpřístupnit funkcionalitu sférického mikrofonního pole nejen běžnému uživateli, ale také k experimentálním účelům, to vyžaduje otevřenost systému k dalším úpravám.

Vývoj zařízení je nyní v experimentální fázi, a tomu odpovídá volba některých technologií i návrh samotného softwaru. V této práci jsou také popsány pokusy provedené se sférickým mikrofonním polem, které ukazují limity současného hardwarového řešení.

Část I

Teoretická část

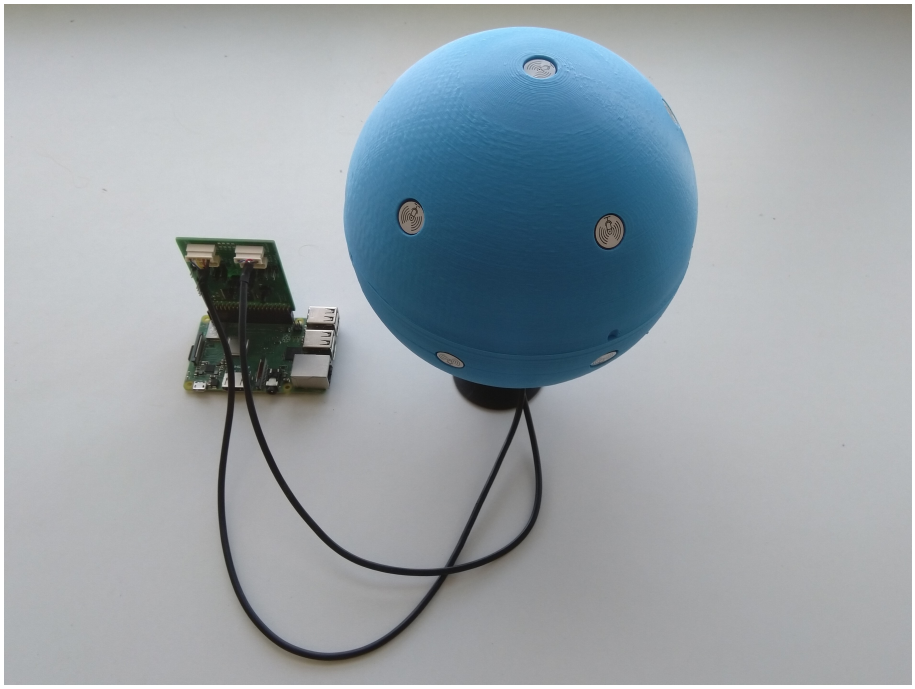
Kapitola 2

Mikrofonní pole

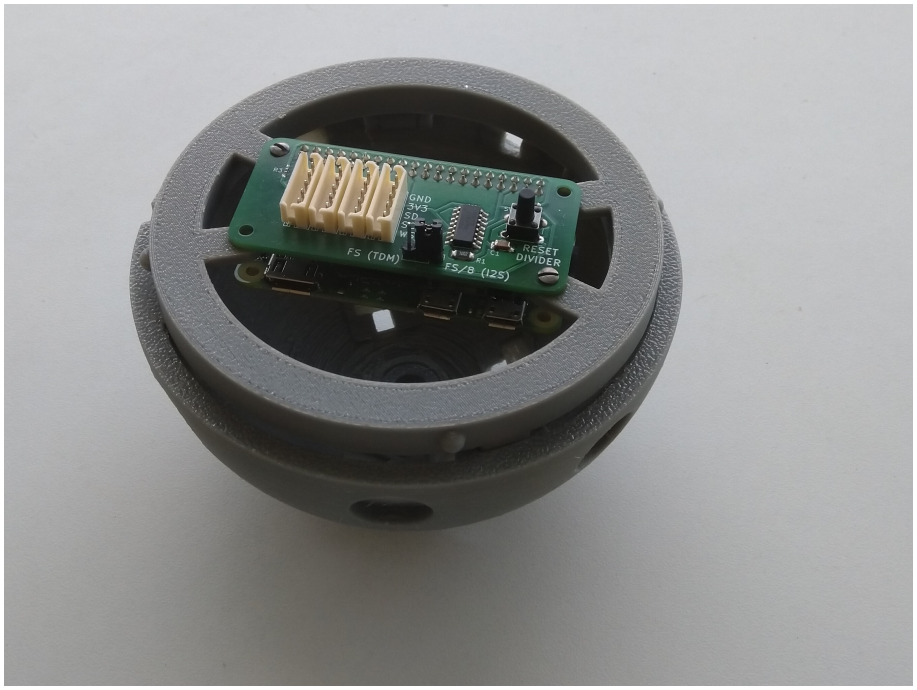
Mikrofonní pole je soustava přibližně stejných mikrofonů, které dohromady plní jednotnou funkci, jak uvádí [1]. Využití mikrofonního pole závisí na uspořádání mikrofonů v prostoru. Informace o pozicích jednotlivých mikrofonů se využívají při následném zpracování dat. Pole využívané k vývoji softwaru vzniklo v rámci práce [2]. Jde o sférické mikrofonní pole s 16 mikrofonními kapslemi.

V průběhu vývoje softwaru bylo možné využívat dvě verze sférického pole, které sestavil David Vagner. U první, starší verze, se řídicí jednotka nachází mimo sféru s mikrofony, jak je vidět na obrázku 2.1. U převodního adaptéru lze nastavit režim přizpůsobení řídicího signálu pro 2, 4, 8 a 16 mikrofonů. Disponuje obvodem pro přizpůsobení délky řídicího impulzu. Jako řídicí jednotka slouží mikorpočítač Raspberry PI 3B+.

Druhá verze má řídicí jednotku zabudovanou uvnitř sféry s mikrofony, jak je vidět na obrázku 2.2. Kvůli tomu jsou možnosti připojení periferií, nebo uživatelského stroje, omezeny. Vývoj tak směřuje spíše k bezdrátovému přístupu. Adaptér u této verze nelze nastavit na počet připojených mikrofonů. Ten je pevně nastaven na 16. Délka řídicího impulzu se nepřizpůsobuje. Jako řídicí jednotka slouží mikropočítač Raspberry Pi Zero 2. Nyní je ve fázi návrhu třetí verze, která pomocí ovládacích prvků na konstrukci zařízení umožní uživateli nahrávat i bez připojení klientské aplikace. Ve všech verzích je využíváno mikrofonů ICS-52000 [3].



Obrázek 2.1: První verze mikrofonního pole s řídicí jednotkou.



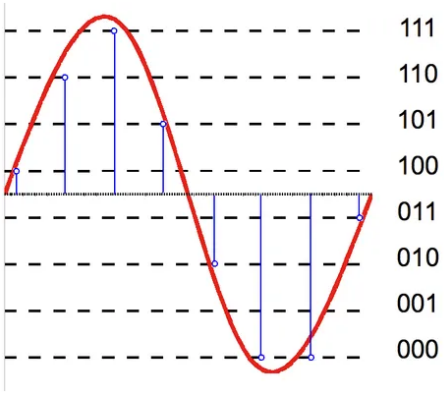
Obrázek 2.2: Polokoule druhé verze s řídicí jednotkou.

2.1 Formát dat z mikrofonního pole

Mikrofonní pole, jak je realizováno v práci [2], se skládá z digitálních mikrofonů MEMS (Micro Electro Mechanical Systems) komunikujících pomocí TDM sběrnice. Ty jsou pak pomocí adaptéru (přídavné karty) připojeny k rozhraní GPIO (General-purpose input/output) jednodeskového počítače Raspberry PI. To přímo nepodporuje rozhraní TDM, proto se ke čtení PCM dat využívá rozhraní I2S. Převod TDM na I2S pak zajišťuje samotný adaptér též vzniklý v rámci práce [2]. Na rozhraní I2S je pak nutné přizpůsobit nastavení parametrů komunikace a interpretaci načtených dat tak, aby odpovídala uspořádání TDM.

2.1.1 Pulse Code Modulation – PCM

Je to způsob digitální reprezentace analogových signálů. Jak uvádí zdroj [4], získává se čtením amplitudy analogového signálu v pravidelných intervalech. Frekvence, se kterou čtení probíhá se nazývá vzorkovací frekvence a počet bitů využitých k zapsání hodnoty amplitudy se nazývá bitová hloubka. Viz obrázek 2.3, kde je znázorněno odečítání amplitudy (svislá osa) v čase (vodorovná osa).



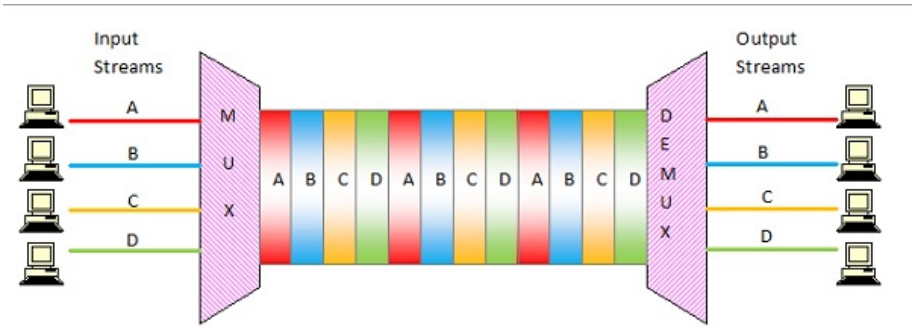
Obrázek 2.3: Nákres odečítání amplitudy.[5]

Mikrofony ICS-52000 mají zabudovaný vlastní A/D převodník, který na TDM rozhraní fixně zapisuje vzorky o šířce 32 bitů. Vzorkovací frekvenci lze na mikrofonu nastavit. Podle Nyquistova teorému musí být vzorkovací frekvence alespoň dvojnásobkem nejvyšší zaznamenávané frekvence, jak uvádí zdroj [6]. Vyšší frekvence musí být před vzorkováním odfiltrovány, jinak dochází

k aliasingu. Rozmezí slyšitelných frekvencí pro člověka je 20 Hz – 20 kHz. Standardně se pro záznam slyšitelného zvuku využívají vzorkovací frekvence 44.1 kHz, 48 kHz, 96 kHz nebo 192 kHz.

2.1.2 Time division multiplexing – TDM

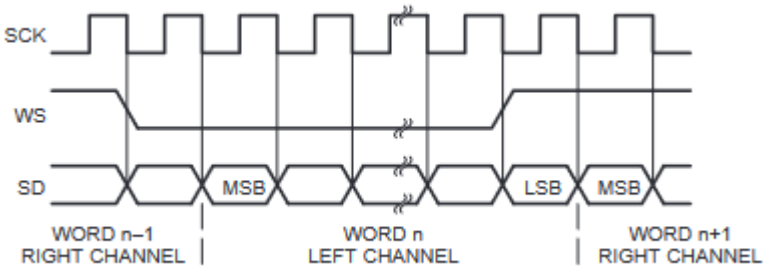
Multiplexing umožňuje přenos dat více komunikačních kanálů pomocí jedné datové linky. Time Division Multiplexing rozlišuje kanály pomocí stejně dlouhých časových slotů [7]. V rámci jednoho časového slotu probíhá zápis dat jednoho kanálu na společnou datovou linku. Pořadí, ve kterém kanály zapisují je stejné po celou dobu přenosu a je známé oběma stranám komunikace (viz obr. 2.4). Data jsou pak demultiplexována podle pořadí, ve kterém byla přijata.



Obrázek 2.4: Time Division Multiplex [8]

Rozhraní I2S

I2S je rozhraní, pomocí kterého lze přenášet dva kanály zvukových dat. Je tvořeno signály SD (data), SCK(clock) a WS (Word select). Úroveň signálu WS se střídá v pravidelných intervalech a vybírá mezi pravým a levým kanálem, jak uvádí [9] (viz obr. 2.5). Zařízení generující řídicí signály WS a SCK se nazývá master a zařízení, které tyto signály přijímá je slave.

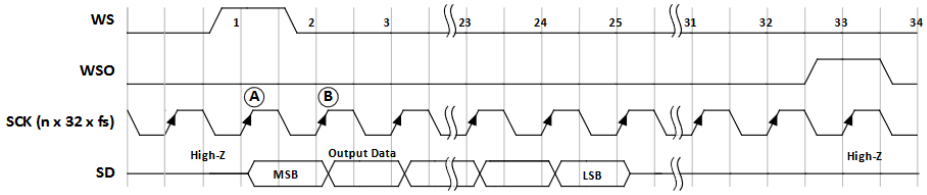


Obrázek 2.5: Časový diagram jednoho časového slotu I2S [9]

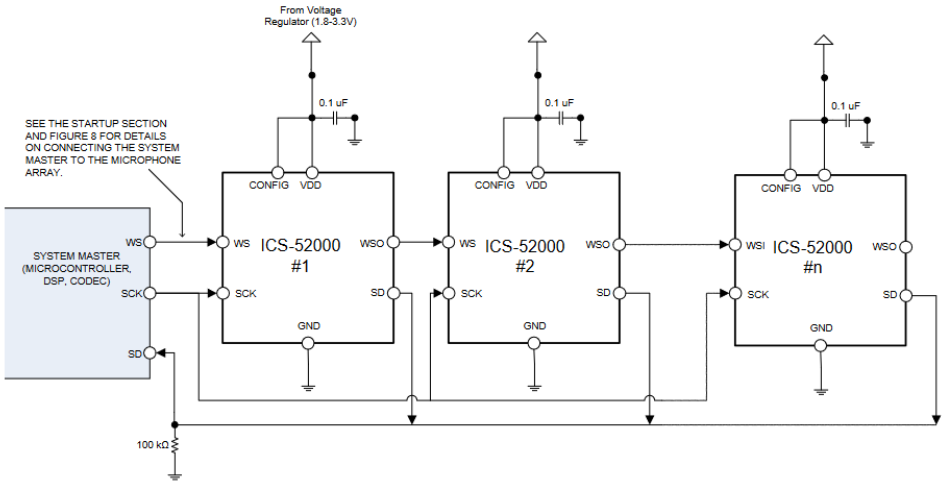
Jak uvádí [9], sériová data jsou při přenosu kódována ve dvojkovém doplňku. Most significant bit (MSB) je přenášen jako první, a to z toho důvodu, že koncové body komunikace mohou mít jinou šířku slova. V případě, že je slovo přenosového systému větší než slovo odesílajícího koncového bodu, jsou bity na místě least significant bitu (LSB) doplněny nulami. Pokud je přenášené slovo větší než, slovo přijímače, bity za LSB jsou ignorovány. Pokud přijímač přijme menší slovo, než je jeho délka slova, chybějící bity jsou doplněny nulami. Vysílač vždy odešle MSB následujícího slova pokaždé, když se změní Word Select.

Rozhraní TDM

Mikrofony ICS-52000 využívají TDM rozhraní, které je podobně jako I2S tvořeno signály SD, SCK a WS. TDM na rozdíl od I2S umožňuje přenos až 16 zvukových kanálů. Jak je popsáno v práci [2], řídicí jednotka vyšle signál po vodiči WS prvnímu mikrofonu. Mikrofon po detekci vzestupné hrany tohoto signálu začne vysílat data. Když se blíží odvysílání všech dat, mikrofon odešle signál WS po svém výstupu WSO (Word Select Output) dalšímu mikrofonu. Vzorky jsou pak při čtení uspořádány ve stejném pořadí, jako jsou odpovídající mikrofony seřazeny na komunikační lince. Datový rámec obsahuje data jedné vzorkovací periody. V jednom datovém rámci je tedy zapsáno n vzorků z n mikrofonů. Obrázek 2.6 ukazuje časový průběh jednoho rámce TDM a obrázek 2.7 ukazuje zapojení mikrofonů na lince.



Obrázek 2.6: Časový diagram jednoho časového slotu rozhraní TDM [3]



Obrázek 2.7: Nákres zapojení mikrofonů [3]

Kapitola 3

Standardy, technologie a protokoly

V následující kapitole jsou popsány standardy, technologie a protokoly, které jsou v programech vzniklých v rámci této práce využívány.

3.1 Komunikace řídicí jednotky s uživatelským strojem

Data získaná z mikrofonního pole je potřeba přenést na uživatelský stroj. Obvykle se pro zpracování nahrávaného zvuku využívá DAW (Digital Audio Workstation), se kterým zvukové zařízení nejčastěji komunikuje pomocí ASIO (Audio Stream Input/Output) driveru. Pokud je zařízení class compliant, lze u některých DAW přenos audia realizovat pomocí operačního systému. Mimo vestavěné funkce DAW lze pak zvuk zpracovávat pomocí pluginů (VST, AU, AAX apod.).

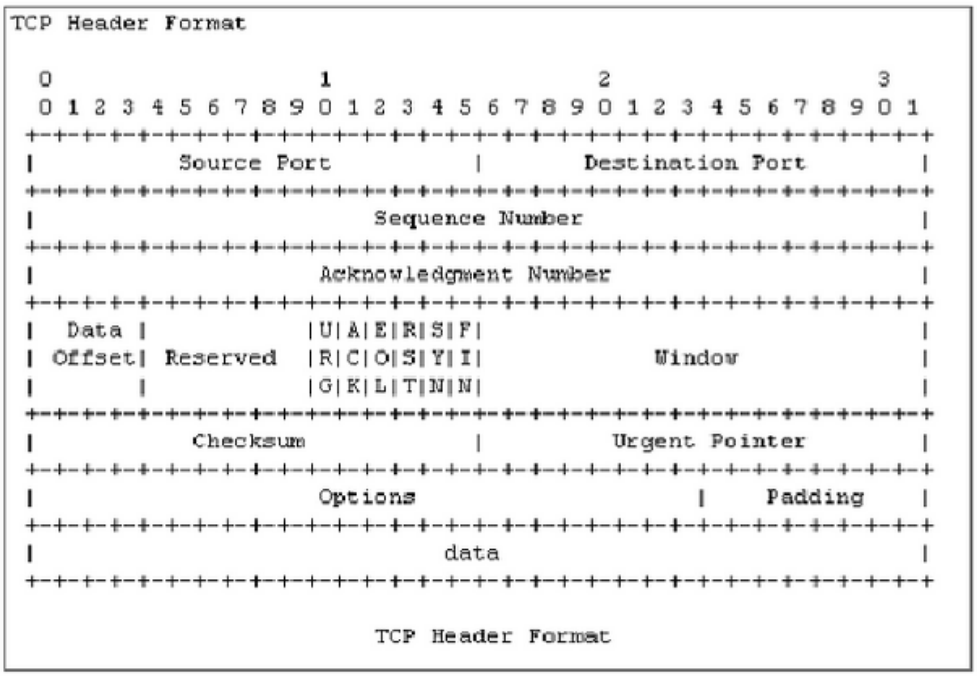
V našem případě bude zařízení komunikovat se samostatnou klientskou aplikací, která bude data ukládat přímo do lokálního uložiště uživatelského stroje. Pro zpracování a monitoring v reálném čase bude možné využít vlastních přídavných modulů. Uživatel pak bude mít možnost nahrané audio do DAW importovat a následně provádět postprodukci. Toto řešení dovoluje mimo jiné větší svobodu při vývoji zařízení a zároveň nevyžaduje k chodu software třetí strany.

3.1.1 Transportní protokol

Komunikace mezi řídicí jednotkou a uživatelským strojem bude realizována pomocí standardních protokolů z rodiny TCP/IP. Důležitou roli v návrhu komunikace hraje volba transportního protokolu.

TCP

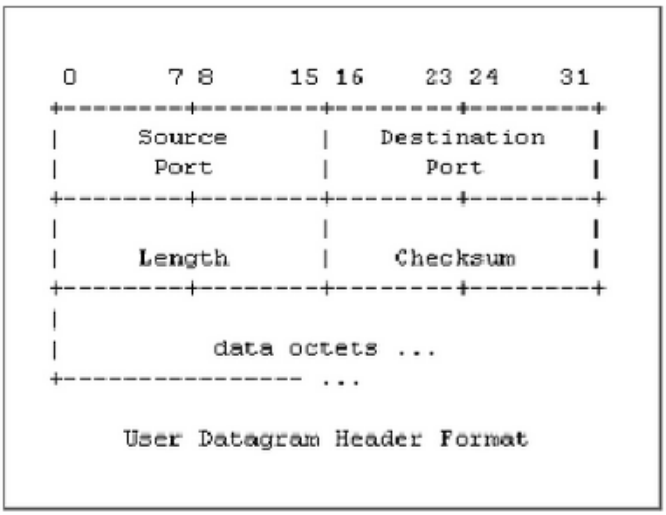
Jak uvádí zdroj [10], Trasmission Control Protocol (TCP) je spolehlivý, spojově orientovaný protokol, který udržuje spojení po celou dobu výměny dat. Spojení je navázáno pomocí takzvaného three-way handshaku. Protokol je zodpovědný za rozdělení dat do paketů na straně odesílatele a zpětné sestavení na straně příjemce. Strukturu datového paketu si lze prohlédnout na 3.1. Zaručuje spolehlivé doručení zprávy tím, že při obdržení datového paketu příjemce odešle potvrzení odesílateli. Pokud odesílatel potvrzení neobdrží, odešle paket znovu. Pakety jsou očíslované pořadovými čísly, pomocí kterých jsou na straně příjemce seřazeny. Poškození paketu je detekováno pomocí kontrolního součtu, jak uvádí zdroj [11].



Obrázek 3.1: Struktura hlavičky TCP paketu [10]

UDP

User Datagram Protocol (UDP) je nespolehlivý nespojový protokol. Na rozdíl od TCP neodesílá potvrzení a nenavazuje spojení. Strukturu datagramu si lze prohlédnout na 3.2. Příjemce nemusí některé datagramy obdržet, nebo mohou přijít v nesprávném pořadí. Spolehlivost přenosu je tak třeba zajišťovat na vyšších vrstvách. UDP díky tomu komunikuje méně dat a je rychlejší než TCP. Na rozdíl od UDP se podle [10] TCP typicky využívá k přenosu velkých objemů dat, kde je vhodné zajistit, aby se nemusel přenos opakovat. Jak uvádí [12], UDP se typicky využívá v aplikacích s vysokou tolerancí chyb, jako jsou například živé streamy, online hry nebo VoIP služby.



Obrázek 3.2: Struktura hlavičky UDP datagramu [10]

STCP

Jak uvádí [13], Stream Control Transmission Protocol (SCTP) je spolehlivý transportní protokol. Na rozdíl od TCP, který navazuje spojení mezi dvěma koncovými body, SCTP vytváří asociaci, která může obsahovat více adres. To umožňuje potenciálně odesílat a přijímat data pomocí více síťových rozhraní najednou. Oproti TCP je SCTP odolný proti útokům typu Denial of Service. SCTP umožňuje tzv. částečnou spolehlivost, díky které lze omezit potvrzovací mechanismus ve prospěch časové náročnosti za cenu nižší spolehlivosti. V současnosti SCTP stále není podporován některými operačními systémy.

Volba protokolu

Mikrofonní pole se často používají pro přesné měření a jakékoliv nedetekované poškození dat může mít fatální dopad na výsledek. Je proto důležité upřednostnit spolehlivost nad rychlost komunikace, a proto UDP není vhodný protokol pro takové účely. SCTP nabízí řadu výhod oproti TCP, nicméně je poměrně komplikovaný a málo podporovaný. Z těchto důvodů se ke komunikaci řídicí jednotky s uživatelským strojem využívá protokol TCP.

3.2 Řídicí jednotka

Na řídicí jednotce je potřeba zajistit čtení z I2S rozhraní, ukládání a odesílání dat. Volba technologií je ovlivněna ve velké míře tím, že software se spouští na operačním systému Linux.

3.2.1 ALSA

Advanced Linux Sound Architecture (ALSA) je nízkoúrovňové rozhraní operačních systémů Linux sloužící ke komunikaci se zvukovými zařízeními. Pro obsluhu zvukového zařízení je potřeba ALSA driver, který umožní komunikaci programů se zvukovým hardwarem, jak uvádí [14]. V našem případě není potřeba vlastní driver vytvářet, pro I2S na Raspberry PI lze využít existujících řešení. K obsluze zařízení z programu řídicí jednotky využijeme knihovny libasound 2. Zařízení lze také ovládat pomocí utilit arecord a aplay.

3.2.2 POSIX

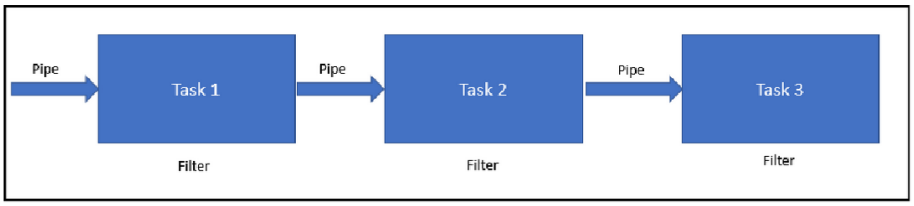
Portable Operating System Interface (POSIX) je rodina standardů, které představují přenositelné rozhraní pro operační systémy vycházející z UNIX [15]. Zapouzdřuje různé funkcionality systému jako je práce se soubory, pokročilou práci s terminálem nebo třeba komunikaci pomocí síťových soketů. Systémy typu Linux i některé systémy typu RTOS, jako je například [16] implementují funkce POSIX. Využitím těchto standardů můžeme dosáhnout snažší přenositelnosti zdrojového kódu mezi systémy.

3.3 Uživatelský stroj

Na uživatelském stroji je potřeba zajistit příjem, základní zpracování a uložení dat ve formátu, se kterým lze pracovat v běžném postprodukčním softwaru.

3.3.1 Pipe and filter

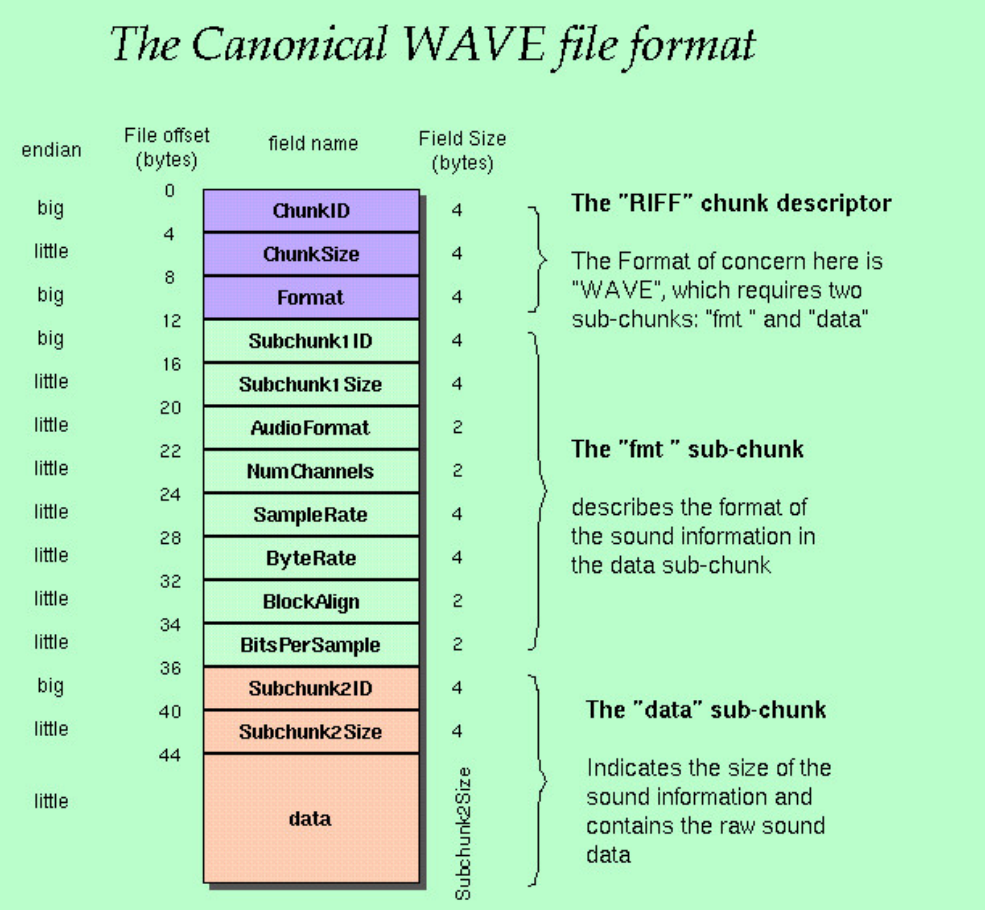
Pipe and filter, jak uvádí zdroj [17], je návrhový vzor užívaný v situacích, kdy jedna událost spouští sérii akcí. Každá z akcí plní svoji specifickou funkci. Umožňuje tak komplexní zpracování příchozích dat se zachováním flexibility a nezávislosti jednotlivých částí řetězce. Celky provádějící samotné zpracování se nazývají „filters“ a jsou propojeny pomocí kanálů zvaných „pipes“. Každý z filtrů má vstupní kanál pro příjem dat a výstupní kanál pro zápis zpracovaných dat (viz obr. 3.3). Systém tak zajišťuje tok mezi zdrojovou komponentou (pump) a cílovou komponentou (sink).



Obrázek 3.3: Architektura pipe and filter [17]

3.3.2 WAV

Aby uživatel mohl data přijatá z mikrofonního pole zpracovávat jako audiozáznam, je potřeba je ukládat ve vhodném formátu. V této práci se pro ukládání dat bude využívat formátu WAV. WAV, známý jako WAVE (Wavefrom Audio File Format) je podmnožinou RIFF (Resource Interchange File Format) od společnosti Microsoft, specifikaci pro ukládání digitálních zvukových souborů. Tento formát umožňuje ukládat zvukové záznamy s různými vzorkovacími frekvencemi a šířkami vzorků přímo v PCM. Je jedním ze standardních formátů pro CD. Soubory uložené ve formátu WAV jsou větší v porovnání se ztrátově komprimovanými formáty jako je například MP3 (MPEG-2 layer 3), jak je uvedeno v [18]. Strukturu formátu si lze prohlédnout na obrázku 3.4.



Obrázek 3.4: Struktura souboru WAV [19]

3.3.3 Python

Software je na uživatelském stroji implementován pomocí programovacího jazyka Python 3. To je multiplatformní interpretovaný jazyk, který je oblíbený zejména pro svou jednoduchost. Python je podle [20] nejpoužívanějším programovacím jazykem na světě. Je vyvíjen pod OSI-approved open-source licencí. To znamená, že je volně použitelný a šiřitelný i pro komerční účely. Licenci spravuje organizace Python Software Foundation [21]. Například oproti jazyku C je, jak ukazuje zdroj [22], výrazně pomalejší. Python nicméně umožňuje kritická místa v jazyce C implementovat. Díky tomu existují knihovny, které díky nízkoúrovňovému backendu umožňují velmi efektivní zpracování dat. Mezi takové patří například SciPy [23] nebo NumPy [24]. Pro dosažení lepšího výkonu lze Python i JIT (Just In Time) kompilovat např. pomocí kompilátoru Numba [25].

Část II

Praktická část

Kapitola 4

Popis aplikace

Aplikace zajišťuje čtení dat z mikrofonního pole a jejich následný přenos na uživatelský stroj. Pro řídicí jednotku byl vytvořen backend, který načítá data ze zvukového rozhraní a odesílá je připojenému klientovi. Klient ovládá řídicí jednotku pomocí řídicích zpráv. Backend může obsluhovat v jednu chvíli pouze jednoho klienta, a to proto, že má k dispozici vždy pouze jedno zvukové zařízení. Aplikace je navržena tak, aby bylo možné využívát více různých klientů pro různé potřeby. V rámci práce vznikly dvě varianty klienta, jeden pro režim IMMEDIATE a druhý pro režim LAZY. Obě varianty jsou implementovány v jazyce Python, aby si je mohl uživatel jednoduše upravit. Klient pro IMMEDIATE režim je o něco složitější, je však rozčleněn do znovupoužitelných modulů, které lze snadno řetězit. Uživatel si tak může do řetězce zapojit moduly vlastní.

4.1 Režimy chodu

V první verzi aplikace program řídicí jednotky data ze zvukového rozhraní rovnou odesílal na uživatelský stroj, kde jej v reálném čase zpracovával klient. Tento režim se v současné verzi nazývá IMMEDIATE. Při testováni se ale zjistilo, že takový režim klade vysoké nároky na kvalitu sítě. Z důvodů uvedených v teoretické části se ke komunikaci dat používá blokující TCP protokol, a to v praxi způsobí čekání programu na potvrzení přijetí paketu z uživatelského stroje. Pokud potvrzení trvá příliš dlouho, program nestihne přečíst data ze zvukového rozhraní, jehož buffer se přeplní a dojde k „buffer

overrun“ erroru. V takovém případě je třeba resetovat zvukové rozhraní, a to způsobí chvilkový výpadek čtení a v důsledku toho ztrátu dat.

Proto současná verze nabízí také tzv. LAZY režim (název je odvozen z tzv. lazy loadingu, kdy program načítá data až ve chvíli, kdy jsou potřeba), ve kterém se nahrávané audio ukládá do souboru, který se až po skončení nahrávání odešle na uživatelský stroj. Minimalizují se tak nároky na kvalitu sítě, nicméně uživatel ztrácí kontrolu nad signálem během nahrávání.

4.2 Systém zpráv

Klient s řídicí jednotkou komunikuje pomocí zpráv. Zprávy jsou textové ASCII řetězce oddělené konci řádků. Zpráva může mít podobu příkazu, oznámení chyby nebo konfigurace. Samotný přenos dat z mikrofonního pole je realizován mimo tento systém zpráv, odesílají se pouze samotná data. Program řídicí jednotky umožňuje jak blokující tak neblokující příjem zpráv, je proto možné s jednotkou komunikovat i v případě, že zrovna odesílá zvuková data.

Implementované zprávy:

1. Počáteční konfigurační zpráva, kterou klient odesílá bezprostředně po připojení k řídicí jednotce. Zpráva má podobu seznamu parametrů oddělených mezerami v tomto pořádí: „<sample rate> <frames to buffer> <audio device> <number of channels> <output file name / immediate mode>”
   a. <sample rate> představuje vzorkovací frekvenci TDM rozhraní v Hz.
   b. <frames to buffer> nastavuje velikost bufferu zvukového rozhraní v TDM rámcích
   c. <audio device> říká, ze kterého zvukového zařízení má program řídicí jednotky data číst. Očekávaná hodnota je `hw`:<číslo zařízení>. Lze však použít i třeba `plughw`, jak popisuje [26].
   d. <number of channels> uvádí počet čtených kanálů, tedy počet mikrofonů v poli.
   e. <output file name/immediate mode> Pokud je hodnota tohoto parametru “IMMEDIATE”, program se přepne do režimu IMMEDIATE. Jinak je program v režimu LAZY a tento parametr je považován za platný název souboru, který se na řídicí jednotce vytvoří a do kterého se uloží nahraná data.

2. Zpráva “record” spustí samotné nahrávání. Během nahrávání program přijímá zprávu, která nahrávání ukončí.
3. Zpráva “stop” ukončí nahrávání.
4. Zpráva “disconnect” ukončí spojení s řídicí jednotkou. Dále vyčkává na připojení dalšího klienta.
5. Zpráva “exit” ukončí spojení s řídicí jednotkou a ukončí obslužný program.
6. Zpráva “fetch” přenese soubor vzniklý v rámci LAZY záznamu s názvem specifikovaným v následující zprávě na klientský stroj.
7. Řídicí jednotka v případě, že dojde k chybě, odesílá textový popis klientské aplikaci.

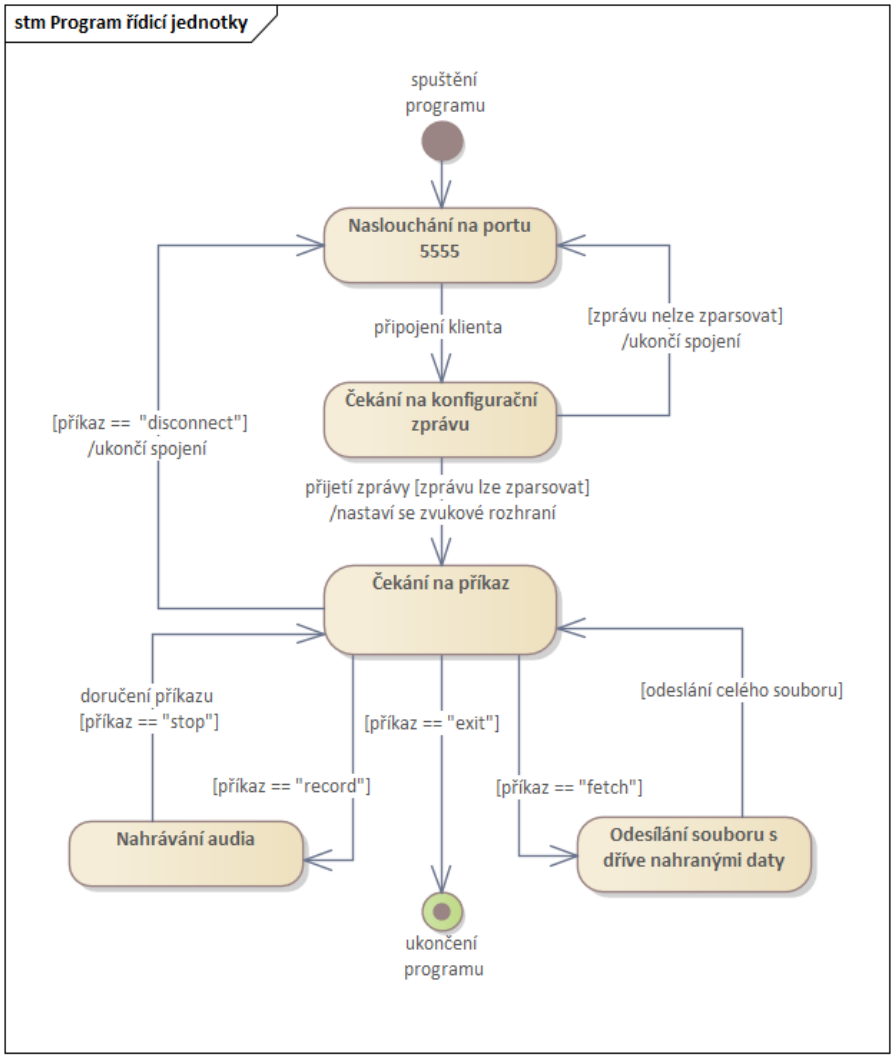
Kapitola 5

Program řídicí jednotky

Program řídicí jednotky je implementován v jazyce C, který byl vybrán pro svou rychlost a snadnou integraci se systémem Linux. Imlementace využívá především standardních knihoven jazyka C pro POSIX systémy. I2S rozhraní je obsluhováno ALSA driverem třetí strany [27]. Pro přístup k rozhraní ALSA se využívá knihovny libasound2 [28]. Program je kompilován překladačem GCC s flagy: `-Wall -Werror -pedantic -lasound`. První tři upravují hlášení warningů a errorů, poslední informuje kompilátor, že má využít knihovny libasound2. Kód je členěn na moduly, které budou dále v textu označovány pomocí hlavičkových souborů, a to proto, že některé moduly obsahují stejnojmenné struktury, od kterých je potřeba název modulu odlišit. Například dvojice `message.h` a `message.c` bude zjednodušeně označena jako modul `message.h`. Naproti tomu `message` je pojmenování struktury, která je v modulu `message.h` definována. Diagram na obrázku 5.1 ukazuje průchod stavy programu. Kód programu se nachází v příloze.

5.1 Síťová komunikace

Aby bylo možné se k řídicí jednotce připojit klientem, je potřeba spustit naslouchání programu na síťovém portu. O to se stará funkce `setup_TCP_server` z modulu `network.h`. Konkrétně v našem případě aplikace naslouchá na portu 5555. Přijetí spojení se provádí funkcí `accept_TCP_client`. Jakmile se klient připojí, přeruší se naslouchání pro další spojení a je očekávaná konfigurační zpráva s nastavením zvukového rozhraní. Poté klient může spustit nahrávání,



Obrázek 5.1: Stavový diagram programu řídicí jednotky

nebo přenést soubor s dříve pořízenou nahrávkou na uživatelský stroj. Po ukončení spojení se může připojit další klient.

Data odeslaná z jednoho stroje v jeden moment nemusí na druhý stroj dorazit najednou. Je tedy možné, že část zprávy přijde v následujícím bufferu, nebo že již část dorazila v bufferu předchozím. Je proto potřeba zajistit kompletaci zpráv. O to se stará modul `message.h`, který obsahuje strukturu `message` a procedury pro práci s ní. Ukládání dat do struktury `message` zajišťuje procedura `add_to_message`. Struktura `message` uchovává informace o přijímané zprávě včetně pravdivostní hodnoty, zda je kompletní. Tu nastaví procedura `add_to_message` na `true`, když narazí na konec řádku, který je oddělovačem zpráv.

Zprávy lze tak přijímat i neblokujícím způsobem. To nám umožňuje přijímat zprávy i v moment kdy probíhá nahrávání a blokování není možné. Zpráva se zpracuje až ve chvíli kdy je označena jako kompletní. V takovém případě je však nutné pro zápis další zprávy strukturu `message`, na rozdíl od blokující varianty čtecí funkce, manuálně resetovat. Pro blokující příjem zprávy slouží funkce `receive_message` a `receive_partial_message_nonblock` pro příjem neblokující.

5.2 Obsluha zvukového rozhraní

Podle přijaté konfigurační zprávy je potřeba nastavit zvukové rozhraní. Parametry jsou přepočítány aby odpovídaly parametrům I2S. Zvukové rozhraní I2S má šířku rámce fixně dva vzorky. Počet vzorků v jednom rámci TDM se rovná maximálnímu počtu mikrofonů v poli. Pokud je v poli N mikrofonů, lze načtení jednoho TDM rámce realizovat načtením $N/2$ rámců I2S. Aby čtení TDM dat probíhalo se správnou vzorkovací frekvencí, je potřeba na I2S nastavit vzorkovací frekvenci $f_{I2S} = f_{tdm} * N/2$. Šířka vzorků zůstává zachována. O přepočty se starají přímo parsovací funkce v modulu `config.h`.

Samotné nastavení zvukového rozhraní realizuje funkce `setup_pcm_interface`, která se nachází v modulu `audio.h`. Ta také z předaných parametrů spočítá velikost čtecího bufferu a alokuje ho. Velikost čtecího bufferu musí být násobkem velikosti TDM framu, aby se kanály nepromíchaly. Čtení pak zajišťuje funkce `read_pcm`, která mimo volání `snd_pcm_readi` ošetřuje případ přetečení vnitřního bufferu rozhraní ALSA, ke kterému může dojít v případě, že se čtení z rozhraní neprovádí dostatečně rychle.

Kapitola 6

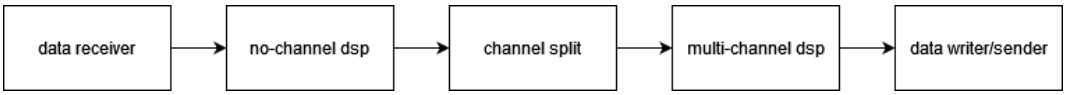
Klient

Od klientské strany se očekává příjem dat, možnost základního zpracování signálu a roztřídění dat do souborů podle kanálů. Program sloužící jako klient pro IMMEDATE režim je možné snadno rozšiřovat implementací vlastních modulů. Moduly mohou sloužit například pro monitorování přijatého signálu a základní zpracování. Monitorovat signál v průběhu nahrávání je vhodné pro včasnou identifikaci defektů způsobených vnějšími vlivy, jako je například saturace mikrofonu. V rámci této práce budou však realizovány pouze moduly pro příjem a správné uložení dat. Uložená data pak budou připravena pro následnou postprodukci. Vzhledem k povaze aplikace je vhodné vycházet z architektury pipe and filter.

Ke spuštění programu je třeba mít nainstalován interpreter jazyka Python 3.x. V modulu `Settings.py` se nachází třída `NetworkSettings`, kde je potřeba nastavit aktuální IP adresu řídicí jednotky mikrofonu. Parametry třídy `AudioSettings` nastavují zvukové rozhraní. Řídicí jednotka musí být připojena k síti tak, aby byla z uživatelského stroje dostupná. Před spuštěním klienta je nutné spustit na řídicí jednotce backend. Klient se pak spustí příkazem `python3 MainProgram.py`. Po spuštění se klient připojí k řídicí jednotce. Uživatel pak stiskem tlačítka `enter` spustí nahrávání. Opětovným stiskem tlačítka `enter` nahrávání vypne. Ve složce `outputs` je poté uložen výstup v podobě šestnácti souborů WAV. Kód programu se nachází v příloze.

6.1 Filtry

Filtry jsou uspořádány do filter chainu. Filter chain dodržuje strukturu naznačenou na obrázku 6.1. Každý box na obrázku představuje pouze abstrakci, která reprezentuje jeden, nebo více reálných modulů.



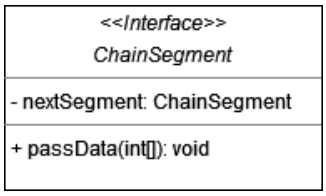
Obrázek 6.1: Obecná organizace filter chainu

- Data receiver - Přijímá data, která netřídí ani nijak nezpracovává. Zajišťuje integritu přijatých dat. Dále předává pole vzorků definované šířky. Délka výstupního pole musí být násobek počtu kanálů. Jako data receiver v této práci slouží modul `TCPClient`.
- No-channel DSP - Řetězec modulů, které provádí operace bez rozlišení kanálu. Zde jsou data uspořádána pro tento úsek neznámým způsobem. Proto operace v této části musí být proveditelné nad každým jednotlivým vzorkem zvlášť bez ohledu na zbytek signálu. V rámci této práce nebyl žádný takový modul implementován, není třeba pro přenos a uložení dat.
- Channel split - Rozdělí data do kanálů. Jedná se o první prvek v cestě, který má o uspořádání dat nějakou informaci. Dále předává data ve 2D poli, kde každý řádek reprezentuje jeden kanál. Jako channel split v této práci slouží `TDMChannelSplit`.
- Multi-channel DSP - Řetězec modulů zpracovávajících signál. Vstupem i výstupem jsou 2D pole. Zde je vhodné řešit vyrovnání charakteristiky jednotlivých mikrofonů, eliminovat přeslechy apod. Jako příklad takového modulu může být `DataLogger`.
- Data writer - Přebírá 2D pole a buď jej uloží, nebo přepošle na další stanici. Tento stupeň zajišťuje správný formát výstupních dat. Může například zapisovat jednotlivé kanály do WAV souborů. V jiném případě se může tvářit jako virtuální audio zařízení komunikující přes ASIO s DAW. Jako data writer v této práci slouží `WavOutput`.

6.1.1 Implementace filtrů

V této sekci je popsána implementace jednotlivých filtrů podle abstraktního modelu, který je popsán v sekci předchozí. Všechny filtry musí implementovat

rozhraní `ChainSegment`. Pro názornost obrázek 6.2 ukazuje `ChainSegment` v notaci UML.



Obrázek 6.2: ChainSegment interface

Chain segment

`ChainSegment` je základním stavebním kamenem filter chainu. Obsahuje odkaz na instanci následujícího prvku v řetězci. Ten je nastaven na `None` právě v případě jedná-li se o sink. Dále obsahuje abstraktní metodu `passData(data)`, která zajišťuje příjem dat od předchozího segmentu. Pokud není implementována, jedná se o zdroj (pump nebo source). Zdroj získává data a cyklicky volá na následujícím segmentu metodu `passData(data)`, dokud jsou dostupná data, nebo dokud není program uživatelem přerušen. Parametr `data` obsahuje pole hodnot jednotlivých vzorků uspořádaných do kanálů podle TDM. Pokud se však jedná o již roztříděná data, obsahuje dvourozměrné pole, kde první index vybírá kanál a druhý index vzorek na tomto kanálu. Pokud segment není sink, je potřeba, aby se v metodě `passData(data)` volala metoda `passData` následujícího segmentu.

- `TCPClient` - Tento segment přijímá data z řídicí jednotky. Chová se jako TCP klient, se kterým komunikuje řídicí jednotka jako server. Přijímá data ze sítě ve formě streamu bajtů, které převádí na čtyřbajtová čísla. Data jsou při přijmu ve formátu little-endian kvůli I2S. Čísla poté ukládá do bufferu předem určené velikosti, která musí být opět násobek velikosti framu TDM, v našem případě tedy násobek čísla 64. Poté buffer předá jako parametr data následujícímu chain segmentu. Mimo příjem dat zajišťuje komunikaci s řídicí jednotkou pomocí řídicích zpráv.
- `TDMChannelSplit` - Data je po obdržení potřeba roztřídit do kanálů. K tomu je určen modul `TDMChannelSplit`, který přijatý buffer převede na pole bufferů, kde každý představuje data pro separátní kanál. Třídění probíhá tak, že i-tý prvek vstupního pole se zařadí na konec pole j-tého z kanálu z n kanálů. $j = i \bmod n$.
- `DataLogger` - Tento segment umožňuje zapisovat data konkrétního kanálu do souboru v čitelné podobě. To je dobré pro diagnostiku chyb

vzniklých při přenosu i záznamu. Data musí být již v roztříděné formě. V případě, že není kanál specifikován, budou se zapisovat data všech kanálů do jednoho souboru.

- `WavOutput` - V našem případě představuje sink. Ukládá data do nekomprimovaných WAV souborů vybraných parametrů. Parametry jsou specifikovány v modulu `Settings`.

6.1.2 Ostatní moduly

- `Settings` - Modul `Settings` obsahuje třídy zapouzdřující nastavení aplikace. Všechna nastavení jsou tak přístupná všem modulům z jednoho místa. V současné verzi jsou dostupné třídy `NetworkSettings` a `AudioSettings`, které upravují parametry síťového připojení a formátu zvukových dat.
- `MainProgram` - `MainProgram` je samotný program klienta, který vytvoří filter chain a spustí běhovou smyčku segmentu `TCPClient`.

Kapitola 7

Pokusy s mikrofonním polem

Kontrolní poslech nahrávek pořízených první verzí toolkitu ze všech 16. mikrofonů v poli odhalil, že nahraný zvuk je velmi zkreslen a překryt velmi hlasitým šumem. Při pokusech pořídit nahrávku z jednoho I2S mikrofonu bez použití přídavného modulu pro převod TDM na I2S byl však signál podle kontrolního poslechu v pořádku. Bylo proto potřeba najít způsob, jak zjistit, kde přesně k problému dochází. Vzhledem k tomu, že data z mikrofonního pole jsou víceméně náhodná, bylo vhodné je pro některé testy nahradit kontrolovaným signálem, který na TDM rozhraní nastavoval předem určené hodnoty. K tomu byl využit signálový procesor ADAU 1452 kontrolovaný softwarem SigmaStudio.

Aby byl minimalizován prostor pro případné chyby na straně softwaru, byla využita zjednodušená varianta aplikace. Pro záznam zvuku z I2S byl na Raspberry PI použit jednoduchý program rovněž realizovaný v jazyce C s využitím technologie ALSA, který data ukládá neroztříděná do binárního souboru na SD kartu příslušného Raspberry PI. Následně byl tento soubor přenášel pomocí scp [29] na klientský stroj. Zde se prováděla kontrola pomocí md5 hashe, aby bylo jisté, že byla data v pořádku přenesena. Následně byla data přetříděna na kanály pomocí skriptu v jazyce Python, který je uložil do oddělených souborů WAV. Ve všech pokusech, kde byl záznam pořizován pomocí mikrofonů, se nahrávala lidská řeč. V takových případech byla nahrávka podrobena kontrolnímu poslechu.

Pokud záznam nevykazoval výrazné známky poškození, je označován dále v textu jako v pořádku, a to i v případech, kdy kvůli nízké vzorkovací frekvenci docházelo k slyšitelnému ořezu vyšších frekvencí. Jde totiž o předpokládané

chování, které vychází z Nyquisova theoremu. Všechny záznamy byly vzhledem k vlastnostem testovaného pole pořizovány se stejnou šířkou vzorku 32 bitů.

Pokus č.1

První pokus měl ověřit funkčnost I2S periferie na Raspebrry PI pomocí jednoduchého záznamu I2S mikrofonu bez přídavné karty. Jako řídicí jednotka bylo využito Raspberry PI Zero 2. Nahrávala se lidská řeč. Při kontrolním poslechu bylo podle očekávání v jednom kanálu úplné ticho a v druhém čistá nahrávka řeči. Nahrávka byla pořízena se vzorkovacím kmitočtem 48 kHz. Datový tok na I2S periferii byl tedy dle výpočtu $sample\_rate * bit\_depth * channels = 48000 * 32 * 2 = 3,072,000$ bitů za sekundu. Vzhledem k tomu, že došlo ke správnému uložení datového souboru i roztřídění dat na kanály, je ověřena i funkčnost testovacího softwaru.

Pokus č.2

Druhý pokus měl ověřit funkčnost přídavné karty v režimu neděleného WS s jedním TDM mikrofonem. Záznam byl stejně jako v prvním případě v pořádku. Nahrávka byla pořízena se vzorkovacím kmitočtem 48 kHz. Datový tok na I2S byl tedy stejný jako v předchozím případě, a to 3,072,000 bitů za sekundu.

Pokus č.3

Třetí pokus byl proveden se dvěma TDM mikrofony v poli. Jako v předchozím případě byl záznam pořízen pomocí Raspberry PI Zero 2 s přídavnou kartou stále v režimu neděleného WS. Záznam byl též v pořádku. Vzorkovací kmitočet, tedy i datový tok, zůstává stejný jako v předchozích případech. Tím je potvrzeno, že správně fungují TDM mikrofony, I2S periferie i přídavná karta v režimu, kdy nedělí WS.

Pokus č.4

Ve čtvrtém pokusu bylo z důvodů zvyšování nároků na procesor využito výkonnější Rasperry PI 4. Testovány jsou 4 mikrofony na vzorkovacím kmitočtu 48 kHz. Přídavná karta byla nastavena do režimu dělení WS pro 4 kanály. Na I2S periferii je tedy nastaven vzorkovací kmitočet 96 kHz. Zvuk ze všech mikrofonů je v pořádku. Je tedy ověřeno, že přídavná karta funguje v režimu dělení WS pro 4 kanály správně a I2S periferie pracuje s datovým tokem 6,144,000 b/s bez problému.

Pokus č.5

Pátý pokus opět využívá Raspberry PI 4 tentokrát s osmi mikrofony v poli. Aby se zaznamenal zvuk s vzorkovacím kmitočtem 48 kHz, na I2S rozhraní bylo potřeba nastavit 192 kHz. To odpovídá datovému toku 12,288,000 b/s. Přídavná karta je uvedena do režimu dělení WS pro osm kanálů. Nahraný zvuk byl hlasitý zkreslený a zašuměný. Dochází tedy k chybě, která byla pozorována se při záznamu 16. mikrofonů. To napovídá, že by mohl být problém s rychlostí čtení z I2S periferie.

Pokus č.6

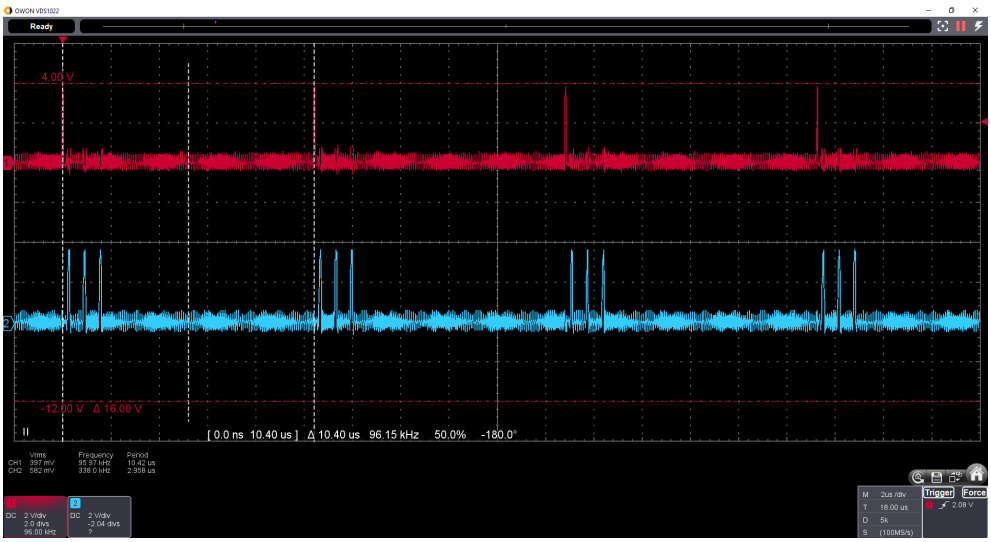
V šestém pokusu bylo zopakováno předchozí měření, akorát s tím rozdílem, že bylo mikrofonní pole nahrazeno signálovým procesorem ADAU 1452, který posílal stále stejné číslo, aby bylo zřejmé, jak se data při čtení oproti zápisu změní. Signálový procesor byl připojen k přídavné kartě stejně, jako by bylo připojeno mikrofonní pole. Kontrola nahraného neroztříděného souboru odhalila, že je načtené číslo oproti číslu posílanému o jeden bit vždy posunuté doprava.

Pokus č.7

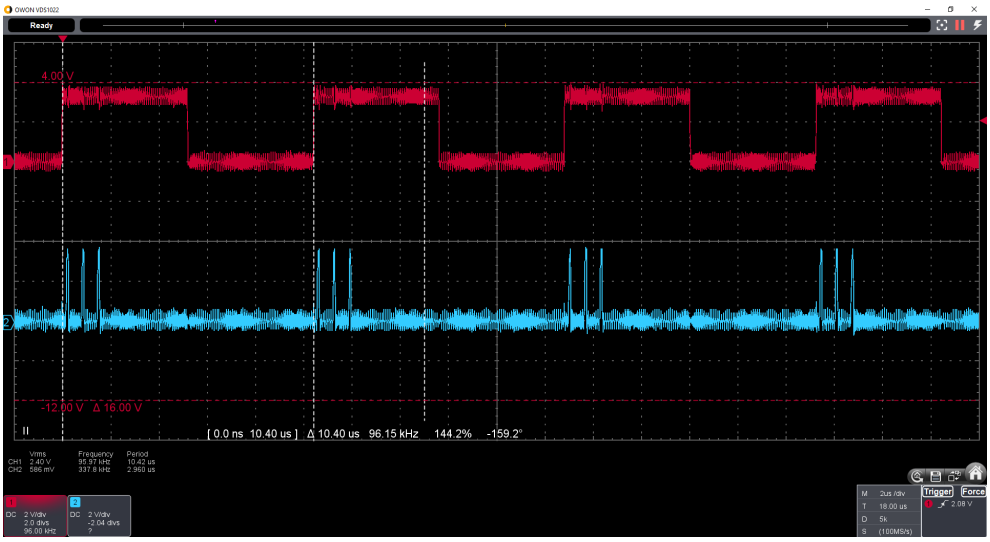
Nyní je zřejmé, že k chybě dochází při vyšších vzorkovacích kmitočtech. Pokus č.6 je tedy zopakován opět se zapojeným mikrofonním polem s osmi mikrofony a dělením WS. Oproti pokusu č.6 byla snížena vzorkovací frekvence na 44.1 kHz. To znamená, že na I2S periferii byl nastaven vzorkovací kmitočet na 176.4 kHz. Mikrofony za těchto podmínek nevysílaly žádná zvuková data. Na osciloskopu bylo možné pozorovat, že WS je nastavován správně, nicméně na datové lince se nic nedělo. Do souboru se uložily jen samé nuly.

Pokus č.8

Přídavná karta zkracuje WS tak, aby se podobal impulzu. Pro účel tohoto testu byl obvod pro zkracování WS přemostěn, aby ke zkracování nedocházelo. Pokus č.8 měl tedy ukázat, jakým způsobem delší WS ovlivní záznam. Mimo prodloužení WS byl pokus č.8 proveden stejně jako pokus č.6. Data byla zaznamenána se vzorkovacím kmitočtem 192 kHz. K zápisu dat na linku byl opět využit ADAU 1452. Data byla, na rozdíl od pokusu č.6, přijata v pořádku. To ukazuje, že délka WS zřejmě nebyla dostatečná pro takto vysoký vzorkovací kmitočet. Zkrácený impulz ukazuje obrázek 7.1 a nezkrácený 7.2. Červeně je znázorněn signál WS a modře SD.



Obrázek 7.1: Ukázka zkráceného WS



Obrázek 7.2: Ukázka nezkráceného WS

Pokus č.9

V předchozím pokusu byl vyřešen problém čtení, v pokusu č.9. bylo potřeba ověřit, zda se vlivem změny délky WS neobjeví problém při zápisu. Pokus č.8 byl proto zopakován tentokrát se zapojeným mikrofonním polem o osmi mikrofonech. Zvuk byl ze všech osmi mikrofonů při kontrolním poslechu v pořádku.

Pokus č.10

Pokus č.10 byl realizován pomocí Raspberry PI 4 se zapojenými všemi šestnácti mikrofony. Přídavná karta byla uvedena do režimu dělení WS pro 16 kanálů. Záznam byl pořízen se vzorkovacím kmitočtem 48 kHz. Na I2S periferii byl tedy nastaven vzorkovací kmitočet 384 kHz. To odpovídá datovému toku 24,576,000 b/s. Nahraný zvuk byl hlasitý zkreslený a zašuměný.

Pokus č.11

Pokus č.10 byl zopakován tentokrát s ADAU 1452. Na zobrazených datech záznamu bylo vidět, že oproti testu č.6 se číslo posouvá tentokrát nepravidelně o jeden bit doprava nebo doleva.

Pokus č.12

Pokus č.10 byl zopakován se vzorkovacím kmitočtem 24 kHz. Na I2S se tedy nastavil vzorkovací kmitočet na 192 kHz. Datový tok byl tedy stejný jako v případě pokusu č.5, tedy 12,288,000 b/s. Záznam byl dle kontrolního poslechu v pořádku. Při takto nízké vzorkovací frekvenci dochází k výraznému ořezu výšek. Záznam s takovou vzorkovací frekvencí může dle Nyquistova theoremu obsahovat zaznamenané frekvence nejvýše do 12 kHz, přitom slyšitelné spektrum je u člověka do cca 20 kHz.

Pokus č.13

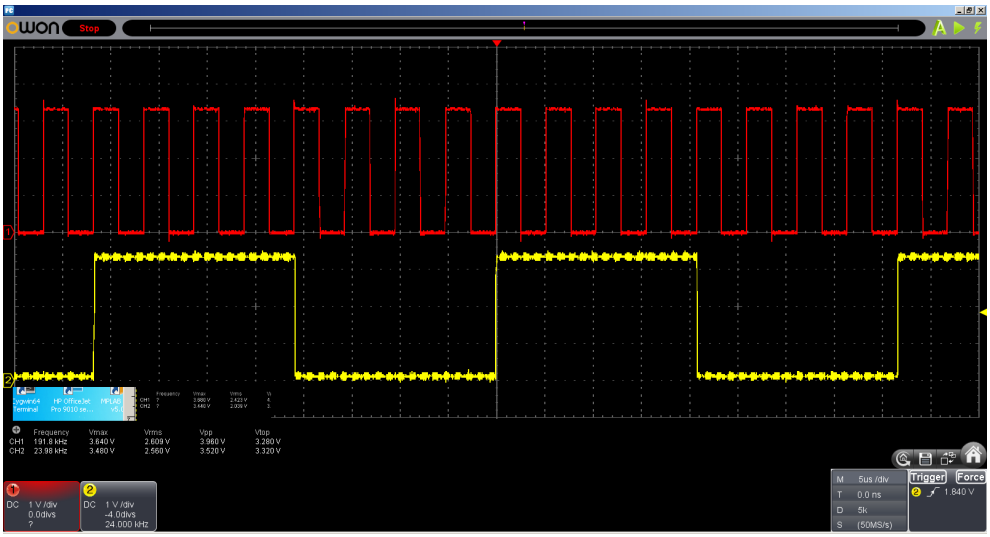
Pokus č.12 byl zopakován tentokrát se vzorkovacím kmitočtem 32 kHz. Na I2S se měl nastavit vzorkovací kmitočet 256 kHz. Pomocí osciloskopu bylo však zjištěno, že se 256 kHz nenastaví. místo toho se vzorkuje stále na 24 kHz, tedy na I2S je vzorkovací kmitočet 192 kHz. Výsledek je tedy stejný jako v předchozím případě.

Pokus č.14

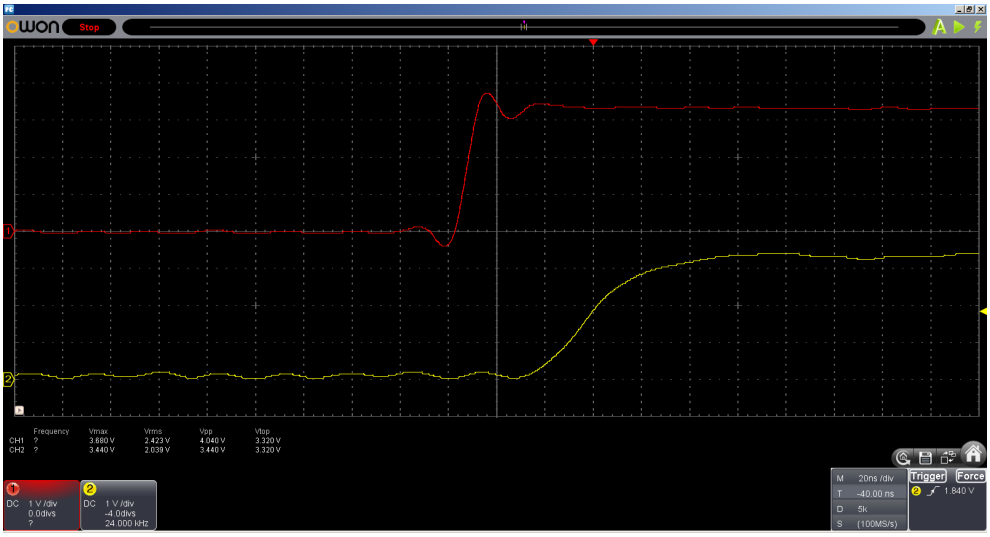
Aby se ukázalo, zda k problémům dochází na přídavné kartě, nebo na samotném Raspberry PI, byl proveden test s ADAU 1452 připojeným přímo ke GPIO pinům Raspberry bez přídavné karty. ADAU 1452 byl nastaven jako I2S slave, který zapisoval se vzorkovací frekvencí 384 kHz konstantní číslo. Po kontrole datového souboru se zjistilo, že na Raspberry k problému nedochází.

Zpoždění signálu WS

Bližší zkoumání děleného signálu WS odhalilo, že při dělení dochází ke zpoždění. Obrázek 7.3 ukazuje porovnání červeného nevyděleného a žlutého vyděleného signálu, kde vypadá signál v pořádku. Obrázek 7.4 ukazuje detail pro porovnání náběžných hran obou signálů. Zpoždění odpovídá cca jedné periodě clocku, data jsou tedy zpožděná cca o jeden bit. Při pokusu opravit signál bitovým posunem se část kanálů opravila, některé však zůstaly poškozené. To může být způsobeno nepravidelností této odchylky.



Obrázek 7.3: Porovnání děleného a neděleného WS



Obrázek 7.4: Detail porovnání děleného a neděleného WS

Shrnutí

Pokusy ukázaly, že mikrofonní pole se zapojenými šestnácti mikrofony je v současnosti funkční pro záznam s datovým tokem 12,288,000 b/s. Na rozhraní I2S to odpovídá nastavení vzorkovací frekvence 192 kHz, což na TDM odpovídá 24 kHz. Při vyšších vzorkovacích frekvencích dochází ke zkreslení a zašumění záznamu. K problému dochází při dělení signálu WS, kde vzniká příliš veliké zpoždění, které se na vyšších vzorkovacích frekvencích projeví jako bitový posun. K posunu nedochází pravidelně, proto jej nelze softwarově opravit.

Kapitola 8

Vize budoucího vývoje

8.1 Hardware

V rámci této práce jsme zjistili, že převod TDM na I2S není v současné verzi funkční pro vysoké vzorkovací kmitočty. Není proto možné pořizovat zvuk z mikrofonního pole ve studiové kvalitě. Mezi diskutovaná řešení patří:

- Upravit adaptér pro převod TDM na I2S tak, aby při dělení frekvence řídicího signálu WS docházelo k menšímu zpoždění.
- Použít více synchronizovaných jednotek, které by sbíraly data z mikrofonního pole po méně kanálech. Synchronizace by probíhala tak, že by na rozhraní I2S jedna z jednotek běžela v módu master a ostatní v módu slave. To bohužel nejde realizovat pomocí Raspberry PI, které mód slave nepodporuje. Bylo by proto vhodné využít například mikrokontroléru ESP32, který oba módy podporuje. Řešení by vyžadovalo osm těchto mikrokontrolérů, které by mikrofony pomocí I2S rozhraní četly po dvou.
- Vyvinout vlastní řídicí jednotku pomocí programovatelného hradlového pole FPGA. Ta by pak s uživatelským strojem komunikovala pravděpodobně pomocí USB, což by vyloučilo bezdrátové využití. Jedná se také o zdaleka nejnáročnější z uvedených řešení.

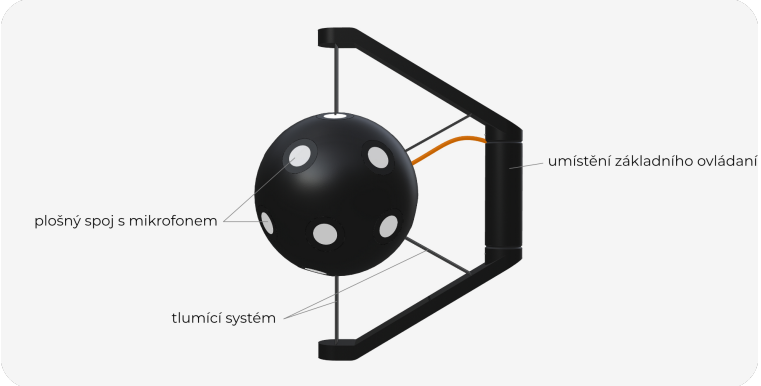
8.2 Software

Podle toho, kterým směrem se bude dále ubírat vývoj hardwaru, vzniknou nové požadavky na software. Samozřejmě řešení v podobě ESP32 nebo FPGA bude vyžadovat kompletně jiný přístup, než řešení s Raspberry PI. Pokud se nadále bude pokračovat v řešení s Raspberry PI, vývoj softwaru si klade následující cíle:

- Vytvořit klienta s GUI.
- Umožnit integraci s DAW pomocí ASIO.
- Vytvořit mobilního klienta pro vzdálenou obsluhu řídicí jednotky.
- Implementovat ovládání (nikoliv přenos dat) řídicí jednotky pomocí Bluetooth.
- Implementovat ovládání pomocí MIDI a OSC.
- Vytvořit backend pro vhodný RTOS který by šlo případně využít jako alternativu k systému Linux.

8.3 Design

Design v současné době zpracovává Danil Rekhtin z Ateliéru průmyslového designu na VŠUP v Praze. Návrh se ubírá směrem přenosného zařízení, které lze využít pro záznam ve venkovních podmínkách. Obrázek 8.1 ukazuje vizi vývoje zařízení.



Obrázek 8.1: Vize vývoje zařízení. Autor: Danil Rekhtin

Kapitola 9

Závěr

V této práci byl navržen a implementován software pro záznam dat z mikrofonního pole a jejich přenos na uživatelský stroj. Návrh kladl důraz na experimentální využití pole a přizpůsobitelnost systému. Aplikace je tvořena backendem běžícím na řídicí jednotce a klientem spouštěným na uživatelském stroji. Aplikace podporuje dva režimy přenosu dat. Jeden přenáší data v reálném čase a druhý až po skončení záznamu. To umožňuje běh přizpůsobit momentální kvalitě dostupné sítě. Klient je snadno rozšiřitelný pomocí vlastních modulů. S využitím první verze aplikace bylo zjištěno, že data z mikrofonního pole přichází poškozena. Provedením několika pokusů se zjistilo, že k poškození dochází při vyšších vzorkovacích frekvencích. Nakonec se ukázalo, že problém působí zpozdění řídicího signálu WS způsobené dělením jeho frekvence. Práce uvádí některá možná řešení a vizi budoucího vývoje.

Přílohy

Příloha A

Odkaz na zdrojový kód

Zdrojový kód je dostupný na repozitáři GitLab.
`https://gitlab.fel.cvut.cz/formato1/sw-toolkit-pro-mikrofonni-pole`.

Příloha B

Seznam zkratek

AAX Avid Audio Extension

ALSA Advanced Linux Sound Architecture

ASIO Audio Stream Input Output

AU Audio Units

CD Compact Disc

DAW Digital Audio Workstation

DSP Digital Signal Processing

FPGA Field Programmable Gate Array

GPIO General Purpose Input Output

GUI Graphical User Interface

I2S Inter-IC Sound

IP Internet Prototcol

JIT Just In Time (compilation)

LSB Least Significant Bit

MEMS Micro Electro Mechanical Systems

MIDI Musical Instruments Digital Interface

MP3 MPEG 2 layer 3

MPEG Moving Picture Experts Group

MSB Most Significant Bit

OSC Open Sound Control

OSI Open Systems Interconnection

PCM Pulse Code Modulation

POSIX Portable Operating System Interface

RIFF Resource Interchange File Format

RTOS Real Time Operating System

SCK Serial Clock

SCTP Stream Control Transmission Protocol

SD Serial Data

TCP Transmission Control Protocol

TDM Time Division Multiplexing

UDP User Datagram Protocol

UML Unified Modeling Language

UNIX UNiplexed Information Computing System

USB Universal Serial Bus

VoIP Voice Over IP

VST Virtual Studio Technology

WS Word Select

WSO Word Select Output

Příloha C

Literatura

[1] Jingdong Chen Jacob Benesty a Chao Pan. *Fundamentals of Differential Beamforming*. Springer Singapore Pte. Limited, květ. 2016. ISBN: 9789811010460.

[2] Vagner David. *Návrh sférického mikrofonního pole*. 2021.

[3] *ICS-52000 Datasheet)*. `https://invensense.tdk.com/download-pdf/ics-52000-datasheet/`. [cit. 2022-05-16].

[4] Stephen Horane. *Introduction to PCM Telemetering Systems*. Taylor & Francis Group, srp. 2017. ISBN: 9781315298481.

[5] *Pulse code modulation technique*. `https://lambdageeks.com/pulse-code-modulation-pcm/`. [cit. 2022-05-16].

[6] Alan P. Kefauver a David Patschke. *Fundamentals of Digital Audio*. A-R Editions, Inc, led. 2007. ISBN: 9780895796110.

[7] Jan A. Audestad. *Technologies and Systems for Access and Transport Networks*. Artech House, říj. 2007. ISBN: 9781596933002.

[8] `https://www.tutorialspoint.com/assets/questions/media/11386/time_division_multiplexing.jpg`. [cit. 2022-5-18].

[9] *I2S bus specification*. `https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf`. [cit. 2022-1-16]. 1986.

[10] Andrew G. Blank. *TCP/IP Foundations*. John Wiley & Sons, Incorporated, zář. 2004. ISBN: 9780782151138.

[11] *The TCP/IP Checksum*. `https://locklessinc.com/articles/tcp_checksum/`. [cit. 2022-05-16].

[12] Wowza Media Systems. *UDP vs. TCP and Which One to Use for Video Streaming (Update)*. `https://www.wowza.com/blog/udp-vs-tcp`. [cit. 2022-05-16]. Břez. 2022.

[13] Victor C. M. Leung et al. *Multihomed Communication with SCTP (Stream Control Transmission Protocol)*. Taylor & Francis Group, pros. 2012. ISBN: 9781466566996.

[14] Jan Newmarch. *Linux Sound Programming*. Apress L. P., led. 2017. ISBN: 9781484224960.

[15] *POSIX (The Portable Operating System Interface)*. `https://www.gnu.org/software/libc/manual/html_node/POSIX.html`. [cit. 2022-05-16].

[16] *eMCOS POSIX: POSIX-compliant RTOS*. `https://www.esol.com/embedded/emcos_posix2.html`. [cit. 2022-05-16].

[17] Chelliah Pethuru Raj, Subramanian Harihara a Murali Anupama. *Architectural Patterns : Uncover Essential Patterns in the Most Indispensable Realm of Enterprise Architecture*. Packt Publishing, Limited, pros. 2017. ISBN: 9781787288348.

[18] *Wav file format*. `https://docs.fileformat.com/audio/wav/`. [cit. 2022-1-16].

[19] *Detailed explanation of WAV file format*. `https://www.fatalerrors.org/a/detailed-explanation-of-wav-file-format.html`. [cit. 2022-1-16].

[20] *TIOBE Index for January 2022*. `https://www.tiobe.com/tiobe-index/`. [cit. 2022-1-16]. 2022.

[21] PSF. *Python*. `https://www.python.org/about/`. [cit. 2022-1-16]. 2022.

[22] Peter Xie. *How Slow is Python Compared to C*. `https://peter-jp-xie.medium.com/how-slow-is-python-compared-to-c-3795071ce82a`. [cit. 2022-05-16]. Čvc. 2020.

[23] *SciPy*. `https://scipy.org/`. [cit. 2022-5-18].

[24] *NumPy*. `https://numpy.org/`. [cit. 2022-5-18].

[25] *Numba*. `https://numba.pydata.org/`. [cit. 2022-5-18].

[26] *ALSA Device Names*. `https://www.alsa-project.org/wiki/DeviceNames`. [cit. 2022-5-18].

[27] *Raspberry Pi Wiring and Test*. `https://learn.adafruit.com/adafruit-i2s-mems-microphone-breakout/raspberry-pi-wiring-test`. [cit. 2022-5-18].

[28] *libasound2*. `https://developer.puri.sm/licenses/Librem5/Birch/libasound2.html`. [cit. 2022-5-18].

[29] *SCP*. `https://man.openbsd.org/scp.1`. [cit. 2022-5-18].