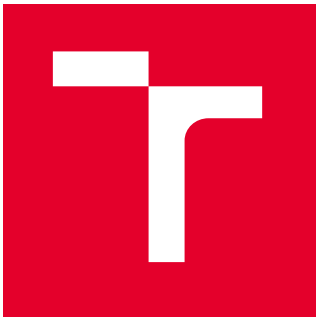


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

SOFTWARE PRO ÚPRAVU ZVUKOVÉHO SIGNÁLU PRO OZVUČOVÁNÍ VÍCE REPRODUKTOROVÝMI SOUSTAVAMI

AUDIO SIGNAL EDITING SOFTWARE FOR MULTI-SPEAKER SYSTEMS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jan Svěrák

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Petr Sysel, Ph.D.

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Bc. Jan Svěrák ***ID:*** 192818
Ročník: 2 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Software pro úpravu zvukového signálu pro ozvučování více reproduktorovými soustavami

POKYNY PRO VYPRACOVÁNÍ:

Realizujte software, který bude umožňovat v reálném čase provádět základní úpravy signálu pro ozvučování více reproduktorovými soustavami. V rámci semestrální práce se seznamte s knihovnou JUCE a vytvořte základ aplikace, která bude realizovat jednoduché zpracování např. MUTE, inverzi signálu nebo zpoždění v rozsahu jednoho až několika tisíc vzorků. V rámci diplomové práce rozšiřte počet zpracovávaných zvukových kanálů na minimálně 32. Doplňte pokročilejší zpracování parametrickým ekvalizérem a matici pro směrování a slučování signálů libovolných vstupů do libovolných výstupů s funkcí soft-fade. Doplňte podporu technologie ASIO a ovládání aplikace pomocí dotykového displeje a rozhraní MIDI.

DOPORUČENÁ LITERATURA:

[1] JUCE Developer Branch Documentation. 2018. Dostupné na URL http://docs.juce.com/develop/index.html. cite [12.9.2018]

[2] KUO, S., M.; LEE, B., H.; TIAN, W. Real-time digital signal processing: implementations and applications. 2nd ed. Hoboken, NJ: John Wiley, 2006. ISBN 0-470-01495-4.

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 24.5.2022

Vedoucí práce: doc. Ing. Petr Sysel, Ph.D.

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

Abstrakt

Tato diplomová práce je zaměřena na vytvoření aplikace pro úpravu zvukového signálu využitelnou při ozvučování více reproduktorovými soustavami, která umožňuje v reálném čase provádět jednoduché zpracování signálu jako například inverzi polarity signálu nebo zpoždění v rozsahu jednoho až několika tisíc vzorků.

Diplomová práce je rozdělena do tří hlavních kapitol. První kapitola obsahuje teoretické znalosti potřebné k vypracování praktické části, například principy číslicového zpracování signálů. Ve druhé kapitole je přehled zdrojových souborů a jejich obsahu a také princip činnosti nejdůležitějších tříd a funkcí. V poslední kapitole je aplikace popsána z pohledu uživatele a slouží jako uživatelský manuál. Je zde vysvětleno ovládání aplikace pomocí grafických prvků a nastavení zvukového zařízení.

Klíčová slova

knihovna JUCE, C++, audio aplikace, DSP, interpolace dat

Abstract

This master's thesis focuses on creating audio editing software for multi-speaker systems which can be used for real-time signal processing, such as polarity inversion or delay ranging from one to several thousand samples.

The thesis is split into three main chapters. The first chapter contains theoretical knowledge required for development of audio software, for example principles of digital signal processing. The second chapter comprises of an overview of the source files and descriptions of several significant classes and functions. The last chapter is intended as a user guide, therefore it contains instructions on how to operate the software via the user interface.

Keywords

JUCE library, C++, audio software, DSP, data interpolation

Bibliografická citace

SVĚRÁK, J. *Software pro úpravu zvukového signálu pro ozvučování více reproduktorovými soustavami*. Brno, 2022. 47 s. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/141431. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce doc. Ing. Petr Sysel, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta: *Bc. Jan Svěrák*

VUT ID studenta: *192818*

Typ práce: *Diplomová práce*

Akademický rok: *2021/22*

Téma závěrečné práce:
Software pro úpravu zvukového signálu pro ozvučování více reproduktorovými soustavami

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.
V Brně dne: 23. května 2021 ----------

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce doc. Ing. Petru Syslovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: 10. prosince 2021 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM VÝPISŮ POČÍTAČOVÝCH KÓDŮ

Úvod

Cílem této diplomové práce je seznámit se s knihovnou JUCE a vytvořit aplikaci pro úpravu zvukového signálu využitelnou při ozvučování více reproduktorovými soustavami. Tato aplikace by měla umožnit v reálném čase provádět zpracování signálu jako například inverzi signálu nebo zpoždění v rozsahu jednoho až několika tisíc vzorků.

Diplomová práce je rozdělena do tří hlavních kapitol. V první kapitole jsou shrnuty teoretické znalosti a principy potřebné k vypracování praktické části. Nejprve je vysvětlen pojem ozvučování, dále je stručně popsán signálový řetězec číslicového zpracování signálů, principy číslicového zpracování dále použité při vytváření aplikace a protokol MIDI. Také je zde stručné seznámení s programovacím jazykem C++, knihovnou JUCE a ovladači ASIO.

Ve druhé kapitole je popsána aplikace z pohledu programátora. Je zde přehled zdrojových souborů a jejich obsahu, dále také princip činnosti nejdůležitějších tříd a funkcí.

V poslední kapitole je aplikace popsána z pohledu uživatele. Je vysvětlena obsluha aplikace pomocí grafických prvků a nastavení zvukového zařízení. Také je zde stručný popis funkcionality vybraných procesů, jako je například ukládání a načítání nastavení.

1 TEORETICKÝ ÚVOD

V této části práce jsou popsány teoretické znalosti, které byly nutné k vytvoření aplikace.

1.1 Ozvučování více reproduktorovými soustavami

Ozvučování je typ reprodukce zvuku, při které je cílem vytvořit maximální úroveň akustického tlaku v místě posluchače a překrýt zesíleným signálem přímou vlnu od původního zdroje zvukového signálu. Důsledkem je překrytí přirozených akustických vlastností prostoru a lokalizace zdroje zvuku posluchačem pouze na základě signálu z reproduktorů. [1]

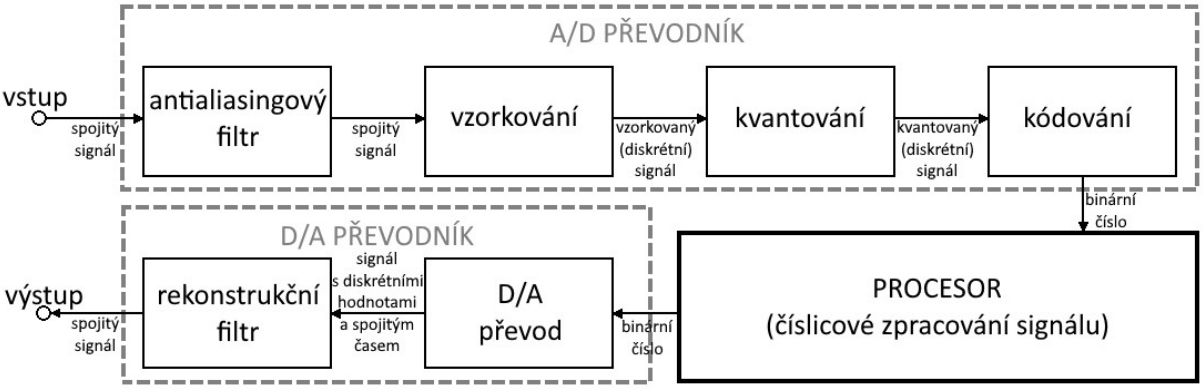
Při ozvučování více reproduktorovými soustavami vzniká potřeba upravovat signál určený pro jednotlivé reproduktory. Důvodem pro tyto úpravy mohou být například odlišnosti v parametrech použitých reproduktorů (kmitočtová charakteristika, charakteristická citlivost atd.), umístění reproduktorů v prostoru (tzn. odlišná vzdálenost od posluchačů) nebo rozdílné účely reproduktorů, ať už z hlediska kmitočtového rozsahu (např. subwoofer) či pokrytí prostoru (např. takzvaný front fill).

1.2 Číslicové zpracování signálů

V této kapitole je stručně popsán typický signálový řetězec pro číslicové zpracování signálu, jakož i metody zpracování signálu dále využité v praktické části práce.

1.2.1 Signálový řetězec

Při ozvučování je typicky vstupní i výstupní signál analogový, zatímco zpracování daného signálu probíhá v číslicové doméně v rámci audio procesoru. K převodu slouží A/D (analogově-digitální) a D/A (digitálně analogový) převodník. Popis principů obsažených v této kapitole vychází z literatury [2], [5], [6] a [7].



Obr. 1.1: Signálový řetězec číslicového zpracování signálu

A/D převod

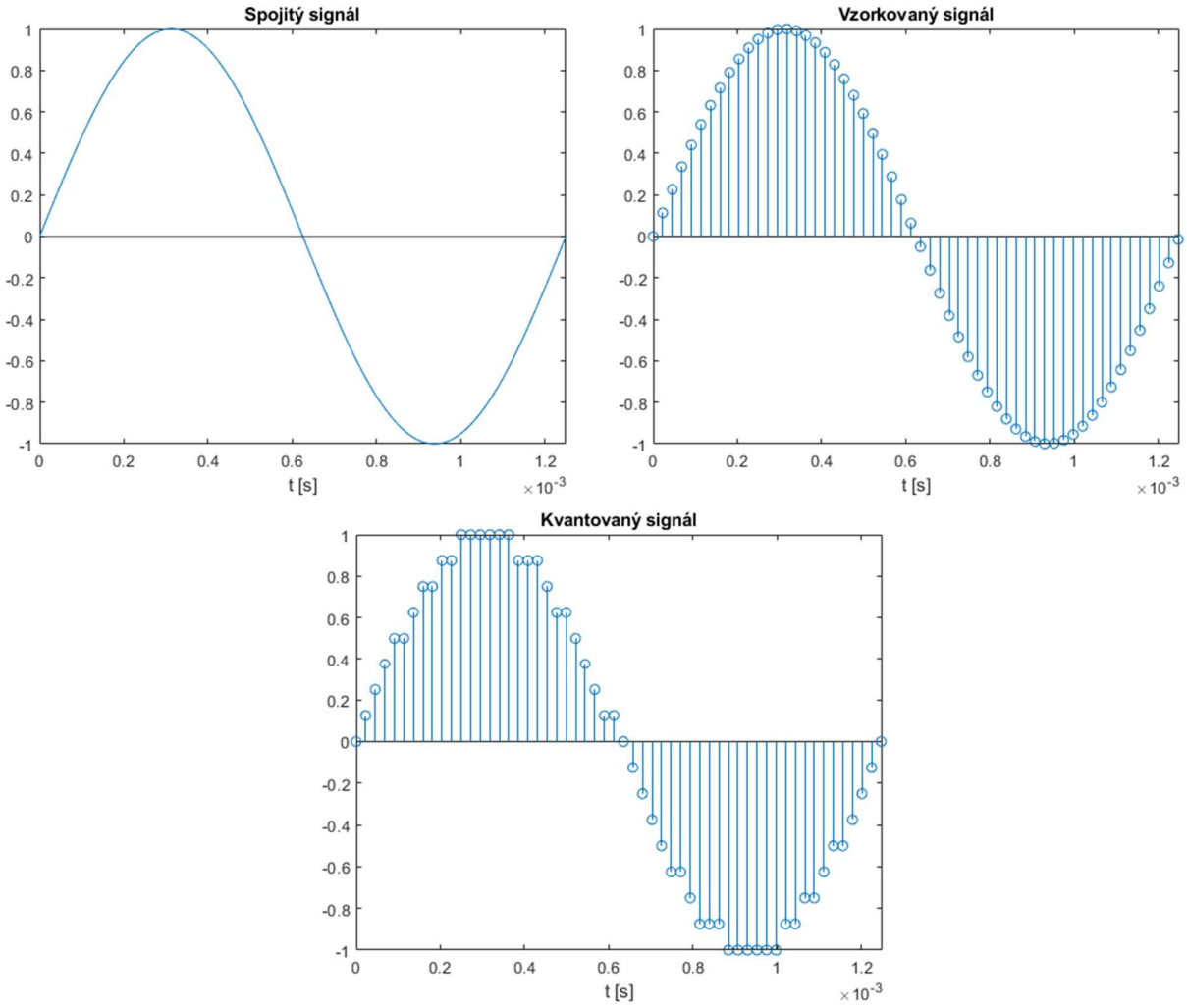
Ještě před převodem analogového signálu na digitální je třeba omezit kmitočtový rozsah. Aby nedošlo k překrývání sousedních spekter vzorkovaného signálu a tím i znemožnění bezchybného obnovení původního signálu (tzv. aliasing), je třeba dodržet vzorkovací poučku,

jinak známou například jako Nyquistův teorém. Podle ní nesmí spektrum vzorkovaného signálu obsahovat kmitočty vyšší, než je polovina vzorkovacího kmitočtu. Toho lze docílit použitím filtru typu dolní propust (tzv. antialiasingový filtr).

Prvním krokem samotného A/D převodu je tzv. vzorkování, tedy převod ze spojitého signálu na diskrétní signál. Při ideálním vzorkování je navzorkovaný signál posloupnost hodnot odpovídajících okamžitým hodnotám původního signálu v intervalech odpovídajících vzorkovací periodě.

Po vzorkování následuje kvantování. Při tomto procesu dojde k převodu vzorků ze spojitých hodnot na diskrétní hodnoty vyjádřené konečnou množinou čísel (tzv. kvantovací hladiny). O množství kvantovacích hladin rozhoduje bitová hloubka. Kvantované vzorky jsou nakonec převedeny na binární čísla v procesu zvaném kódování.

Příklad průběhu signálu v jednotlivých částech A/D převodu je zobrazen na obr. 1.2. V tomto příkladu je vzorkovací kmitočet 44100 Hz a kmitočet signálu je 800 Hz.



Obr. 1.2: Průběh signálu při A/D převodu

D/A převod

Převod z číslicového signálu na analogový je v jistém smyslu jednodušší, než je dříve popsaný A/D převod. Prvním krokem je převedení číslicových vzorků na napětí. Pro odstranění

postranních složek a vyhlazení signálu je třeba použít tzv. rekonstrukční filtr, což je filtr typu dolní propust s mezním kmitočtem odpovídajícím polovině vzorkovacího kmitočtu.

Číslicové zpracování signálu

Jelikož by převod a zpracování každého vzorku zvlášť bylo velice výpočetně náročné, vzorky jsou ukládány do tzv. vyrovnávací paměti (anglicky buffer) a zpracovány po skupinách. Počet vzorků ve vyrovnávací paměti obvykle odpovídá mocnině dvou, ale můžeme se setkat i s jinými hodnotami. [6]

Jak již bylo zmíněno, A/D převodník předává procesoru vzorky jako binární čísla. Procesor s těmito čísly může pracovat ve dvou formátech – v případě zpracování víceúčelovým zařízením jako například počítačem jde o čísla s pohyblivou desetinnou čárku, které jsou používány z důvodu většího dynamického rozsahu a tím i všestrannosti. V případě specializovaných zařízení se však můžeme setkat i s procesory pracujícími s pevnou desetinnou čárkou, které bývají levnější z hlediska výroby i provozu vzhledem k výpočetnímu výkonu, jsou však náročnější na vývoj kvůli nutnosti škálování (je nutno zamezit přetékání hodnot). [5][6]

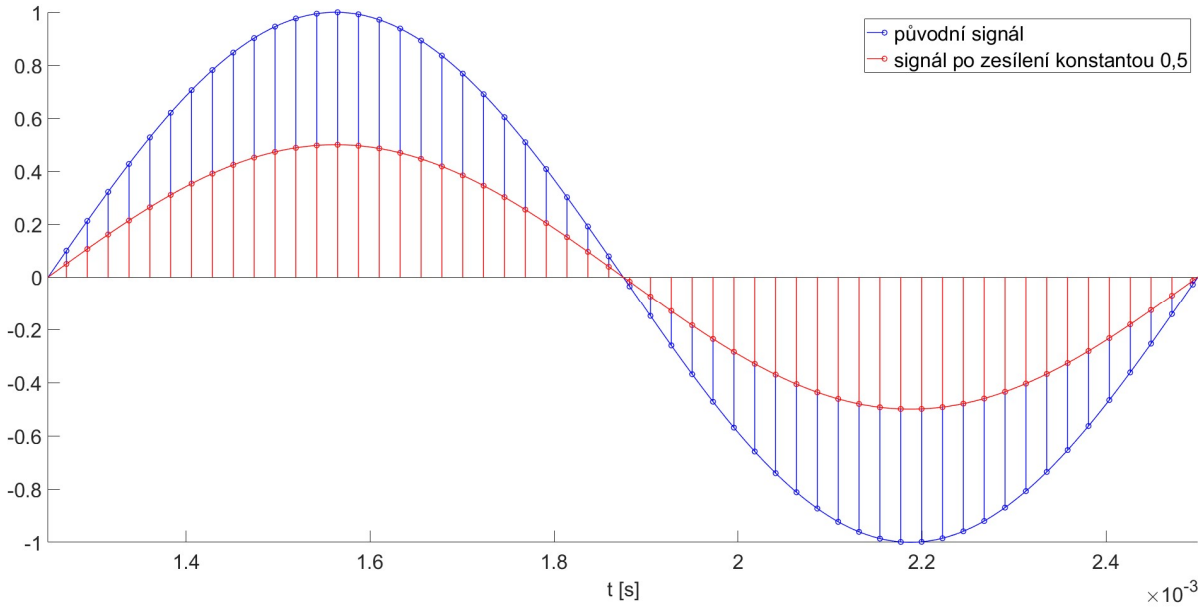
1.2.2 Zesílení signálu

Zesílení je jednou ze základních úprav signálu. Hodnota zesílení odpovídá poměru amplitud výstupního a vstupního signálu. Často se setkáme s vyjádřením zesílení v decibelech (dB), které výsledek poměru převádí do logaritmického měřítka, viz rovnice (1.1) a (1.2) pro zpětný převod. Pokud je hodnota zesílení menší než jedna, případně nižší než nula decibelů, můžeme hovořit o zeslabení. [6]

$$A_{\mathrm{dB}} = 20 \log A \quad [\mathrm{dB}] \tag{1.1}$$

$$A = 10^{A_{\mathrm{dB}}/20} \quad [-] \tag{1.2}$$

V číslicovém zpracování signálů je možné implementovat zesílení pouhým vynásobením hodnoty každého vzorku hodnotou zesílení. Při změnách zesílení je vhodné zajistit, aby nedošlo k prudkým změnám v signálu, což může způsobit slyšitelné prasknutí. Řešením tohoto problému je aplikovat změny postupně, běžně postačí délka přechodu v řádu milisekund.

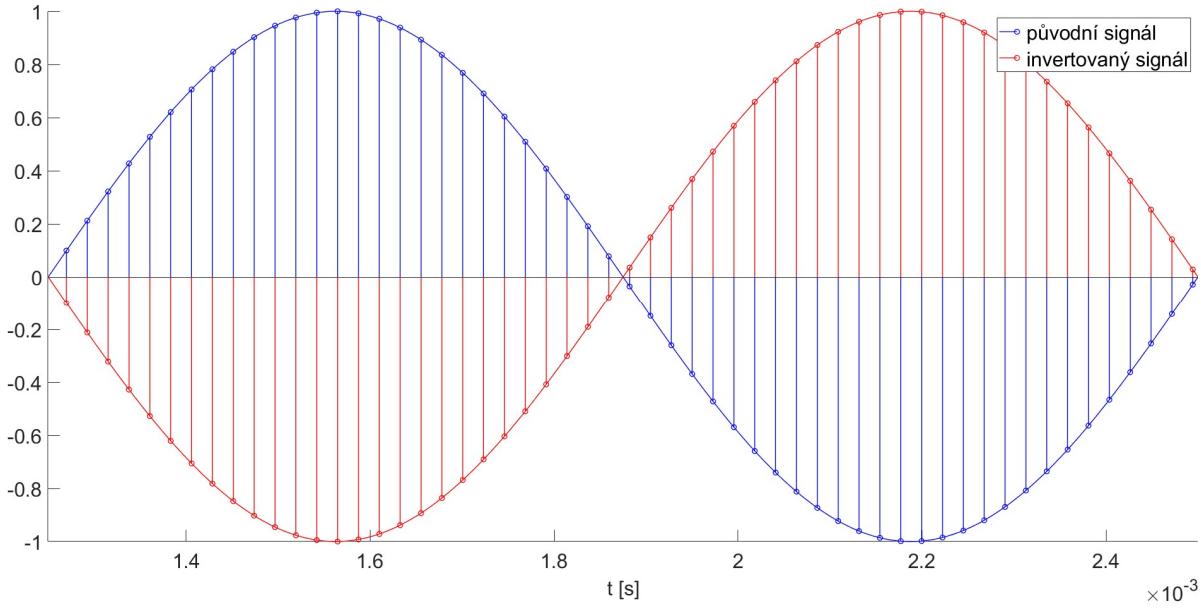


Obr. 1.3: Zesílení signálu

1.2.3 Inverze signálu

Inverze signálu odpovídá posunutí fáze signálu o 180° či π radiánů. Tato úprava signálu se běžně používá například při snímání zvukového zdroje dvěma mikrofony, které jsou na opačných stranách zdroje a signál k nim přichází s opačnou polaritou (např. snímání hrací a rezonanční blány malého bubnu).

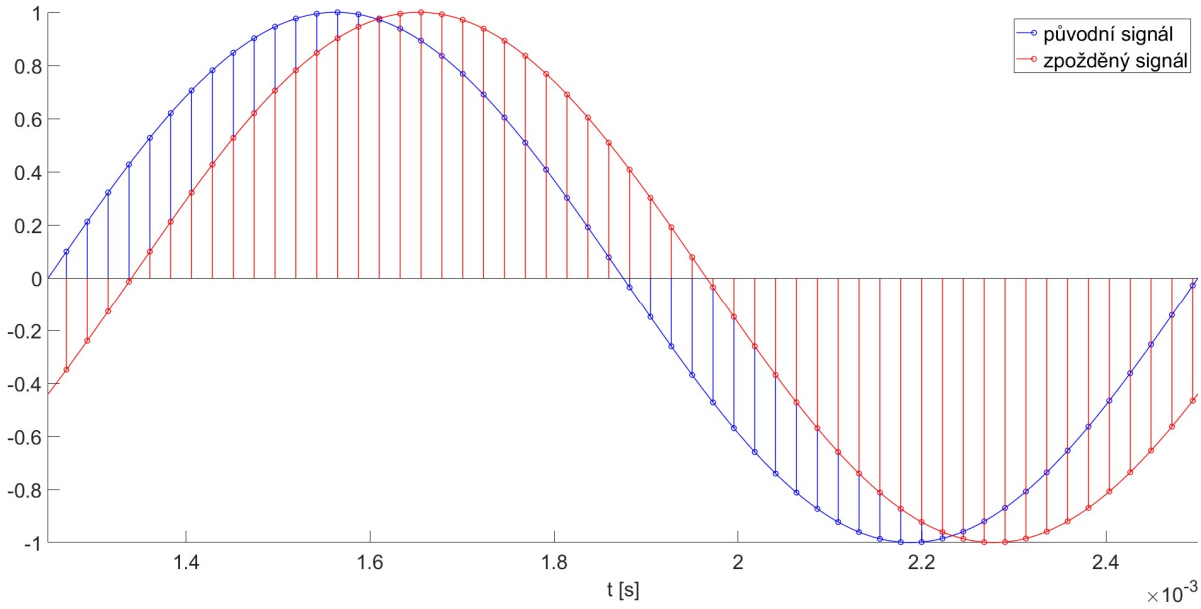
Při číslicovém zpracování signálu je inverze velice jednoduchá – stačí pouze otočit polaritu signálu, čehož lze docílit vynásobením vzorků hodnotou −1. Prakticky se tedy jedná o změnu zesílení v podobě otočení znaménka, tudíž je vhodné inverzi aplikovat v rámci alespoň několika milisekund, viz kapitola 2.2.2.



Obr. 1.4: Inverze signálu

1.2.4 Zpoždění signálu

Rozdílnou vzdálenost reproduktorů od posluchačů lze kompenzovat zpožděním signálu bližšího reproduktoru. V číslicovém zpracování signálů lze zpoždění implementovat pomocí kruhové vyrovnávací paměti. Do této paměti se ukládají vzorky vstupního signálu (tzv. zápis) a poté se kopírují do výstupního signálu, ovšem s posunutým indexem (tzv. čtení). Když indexy čtení či zápisu dojdou na konec paměti, vrátí se zpět na začátek – odtud pojem kruhová paměť. V případě zápisu jsou pak nejstarší vzorky přepisovány novějšími. [6]



Obr. 1.5: Zpoždění signálu

1.2.5 Zlomkové zpoždění

Implementace zpoždění číslicového signálu s konečným vzorkovacím kmitočtem f_s pomocí kruhové paměti má omezené časové rozlišení určené vzorkovací periodou T_s. Kvůli zaokrouhlení požadovaného zpoždění na celé násobky vzorkovací periody dochází k odchylce rovné až polovině vzorkovací periody. Tato odchylka je obvykle zanedbatelná (při f_s = 44,1 kHz odpovídá nanejvýš cca 11,3 µs), při použití v laboratorních podmínkách tomu tak ovšem být nemusí. V těchto případech je třeba provádět zpoždění o neceločíselný násobek vzorkovací periody, tzv. *zlomkové zpoždění*.

Lineární interpolace

Nejjednodušší metodou zlomkového zpoždění je lineární interpolace, která vychází z předpokladu, že průběh signálu mezi dvěma body odpovídá průběhu lineární funkce, viz rovnice (1.3).

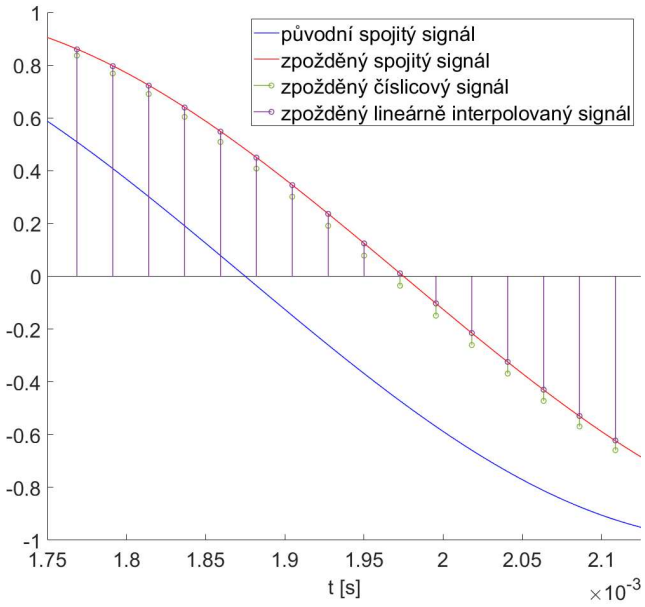
$$y(t) = at + b \quad [-] \tag{1.3}$$

Budeme-li předpokládat, že zpoždění o celý počet vzorků je realizováno kruhovou pamětí a interpolací signál zpožďujeme pouze o zlomek vzorku, můžeme do této rovnice dosadit hodnoty nejbližších vzorků a časy 0 a 1. Získáme tak soustavu dvou rovnic o dvou neznámých,

z jejíhož řešení můžeme sestavit rovnici (1.4), kde y_0 a y_1 jsou hodnoty nejbližších vzorků, d je zlomkové zpoždění v rozsahu 0 až 1 a $y[d]$ je výsledná hodnota signálu zpožděného o d vzorků.

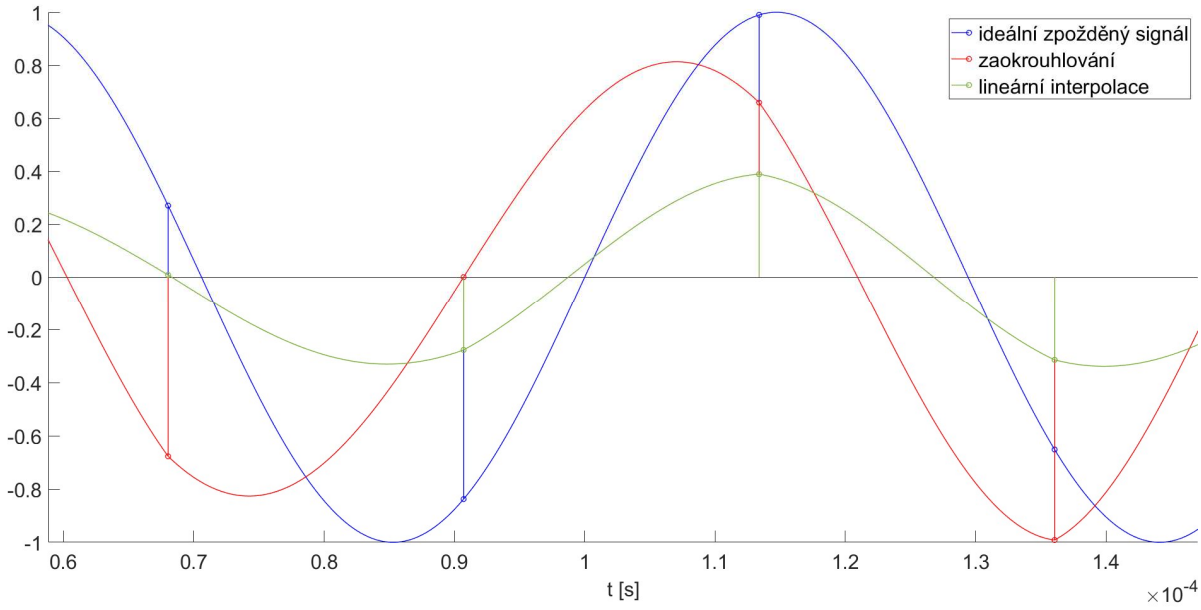
$$y[d] = (y_1 - y_0) \cdot d + y_0 \qquad [-] \qquad (1.4)$$

Na obrázku 1.6 je srovnání zpožděného signálu s lineární interpolací a zaokrouhlením na celé vzorky.



Obr. 1.6: Lineární interpolace

Jelikož je k výpočtu hodnoty použit následující vzorek y_1, při zpracování signálu v reálném čase zavádí lineární interpolace zpoždění o délce jednoho vzorku, tudíž signál nelze zpozdit méně než o jednu vzorkovací periodu. Výhodou lineární interpolace je nízká výpočetní náročnost a nezávislost náročnosti na hodnotě zpoždění (za předpokladu použití stejných datových typů), nevýhodou je ovšem nepřesnost aproximace harmonických signálů lineární funkcí, což vede ke zkreslení signálu projevujícímu se zejména na vysokých kmitočtech, viz obr. 1.7.



Obr. 1.7: Zkreslení při použití lineární interpolace

Kvadratická interpolace

Kvadratická interpolace využívá stejných principů jako lineární interpolace, používá ovšem kvadratickou funkci, viz rovnice (1.5), a tři nejbližší vzorky.

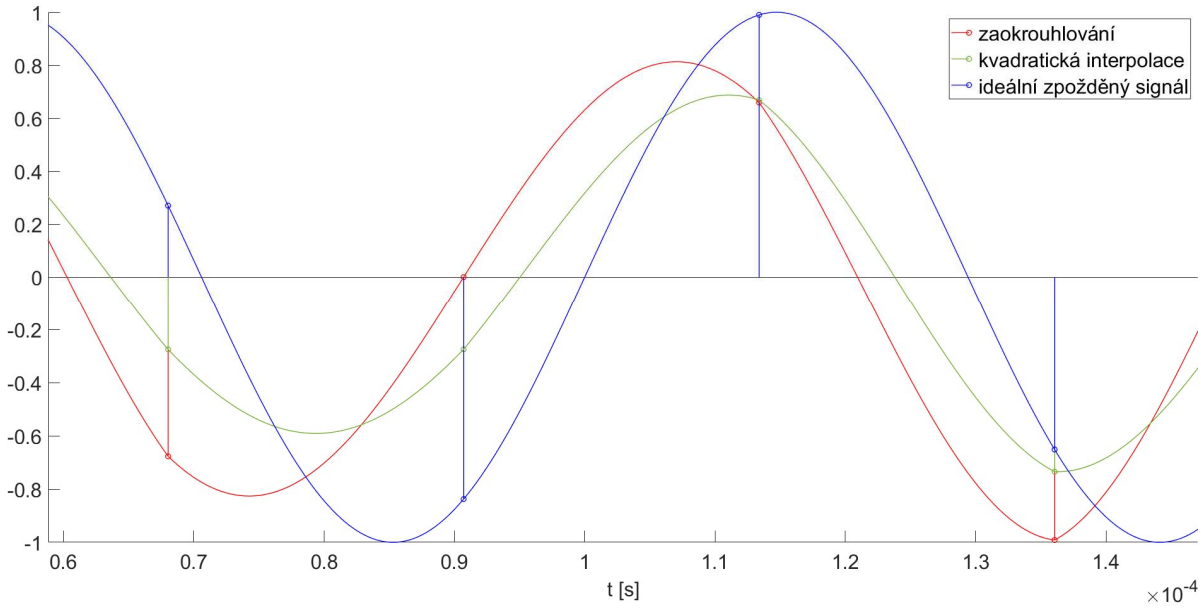
$$y(t) = at^2 + bt + c \qquad [-] \tag{1.5}$$

Budeme-li opět předpokládat, že interpolací signál zpožďujeme pouze o zlomek vzorku, můžeme sestavit a vyřešit soustavu tří rovnic o třech neznámých, jejíž řešením získáme rovnici pro určení výsledku interpolace. Jelikož kvadratická interpolace využívá lichý počet vzorků, má výsledná rovnice dvě podoby v závislosti na tom, které vzorky jsou nejbližší, tedy zda je zlomkové zpoždění menší či větší než ½ (pokud je hodnota přesně ½, lze použít kteroukoliv z podob). Řešením pro vzorky s indexy {-1, 0, 1} je rovnice (1.6), pro vzorky s indexy {0, 1, 2} je jím rovnice (1.7).

$$y[d] = \left(\frac{y_{-1}}{2} - y_0 + \frac{y_1}{2}\right) \cdot d^2 + \left(-\frac{y_{-1}}{2} + \frac{y_1}{2}\right) \cdot d + y_0 \quad [-] \tag{1.6}$$

$$y[d] = \left(\frac{y_0}{2} - y_1 + \frac{y_2}{2}\right) \cdot d^2 + \left(-\frac{3y_0}{2} + 2y_1 + \frac{y_2}{2}\right) \cdot d + y_0 \qquad [-] \tag{1.7}$$

Kvadratická interpolace je výpočetně náročnější než lineární, aproximace harmonických signálů kvadratickou funkcí je však přesnější, což vede k méně výraznému zkreslení signálu. Stejně jako u lineární interpolace je zavedeno zpoždění – pro zlomkové zpoždění menší než ½ se jedná o jeden vzorek, pro zlomkové zpoždění větší než ½ jsou to dva vzorky.



Obr. 1.8: Zkreslení při použití kvadratické interpolace

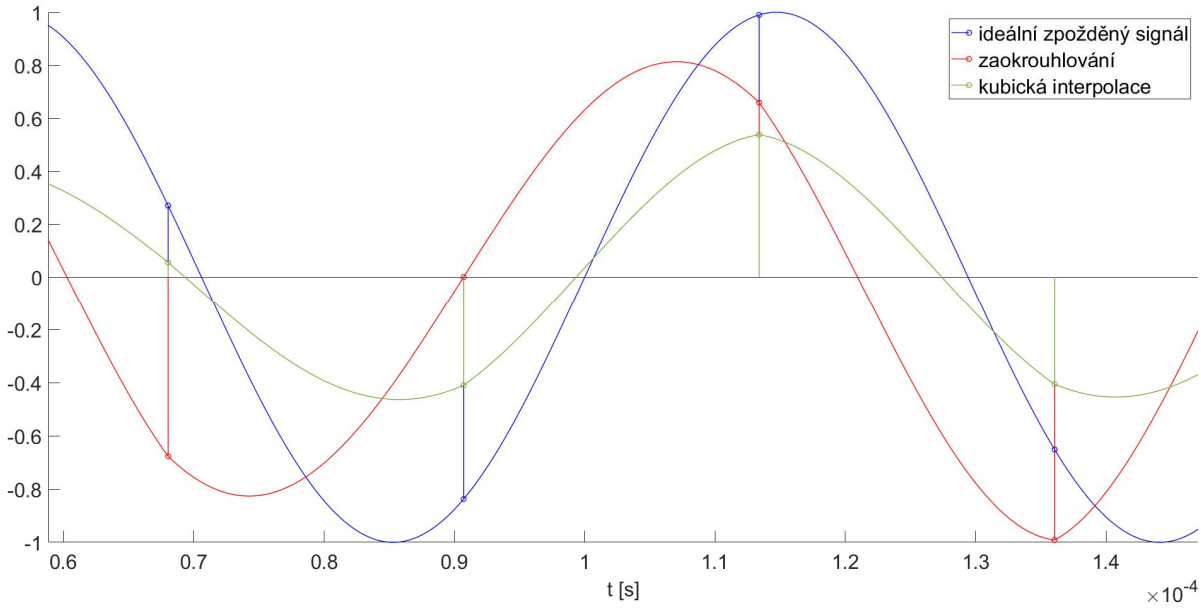
Kubická interpolace

Jak je již patrné z názvu, tato metoda využívá aproximaci kubickou funkcí, viz rovnice (1.8). Obdobně jako u předchozích metod vzorec pro výpočet zpožděného signálu získáme sestavením a řešením soustavy čtyř rovnic o čtyřech neznámých, viz rovnice (1.9).

$$y(t) = at^3 + bt^2 + ct + d \qquad [-] \tag{1.8}$$

$$y[d] = \left(-\frac{y_{-1}}{6} + \frac{y_0}{2} - \frac{y_1}{2} + \frac{y_2}{6}\right) \cdot d^3 + \left(\frac{y_{-1}}{2} - y_0 + \frac{y_1}{2}\right) \cdot d^2 + \\ + \left(-\frac{y_{-1}}{3} - \frac{y_0}{2} + y_1 + \frac{y_2}{2}\right) \cdot d + y_0 \qquad [-] \tag{1.9}$$

Důsledkem použití funkce vyššího stupně je kubická interpolace výpočetně náročnější než předchozí metody, výhodou je však další zlepšení přesnosti. Stejně jako u kvadratické interpolace pro zlomkové zpoždění větší než ½ je při zpracování v reálném čase zavedeno zpoždění dva vzorky.



Obr. 1.9: Zkreslení při použití kubické interpolace

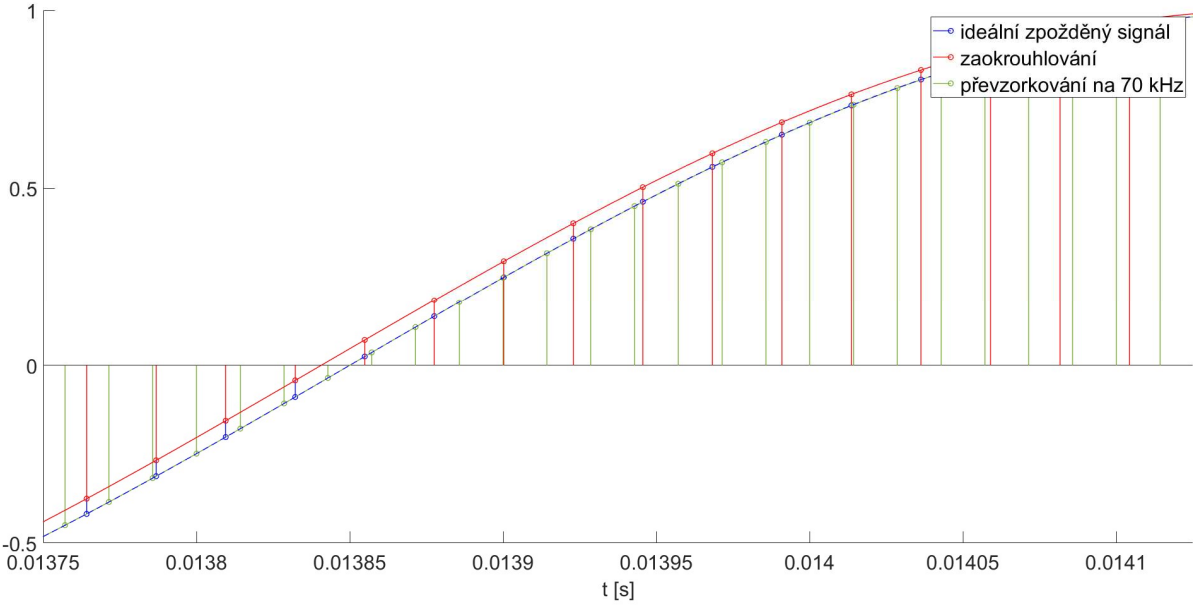
Převzorkování

Další metodou zpracování zlomkového zpoždění je převzorkování na takový vzorkovací kmitočet, pro který lze požadované zpoždění vyjádřit jako celočíselný násobek vzorkovací periody. Po tomto převodu je provedeno zpoždění pomocí kruhové paměti a signál je následně převzorkován zpět na původní kmitočet. Pro zachování původního kmitočtového rozsahu je třeba převzorkování provádět tak, aby nový vzorkovací kmitočet nikdy nebyl nižší než původní.

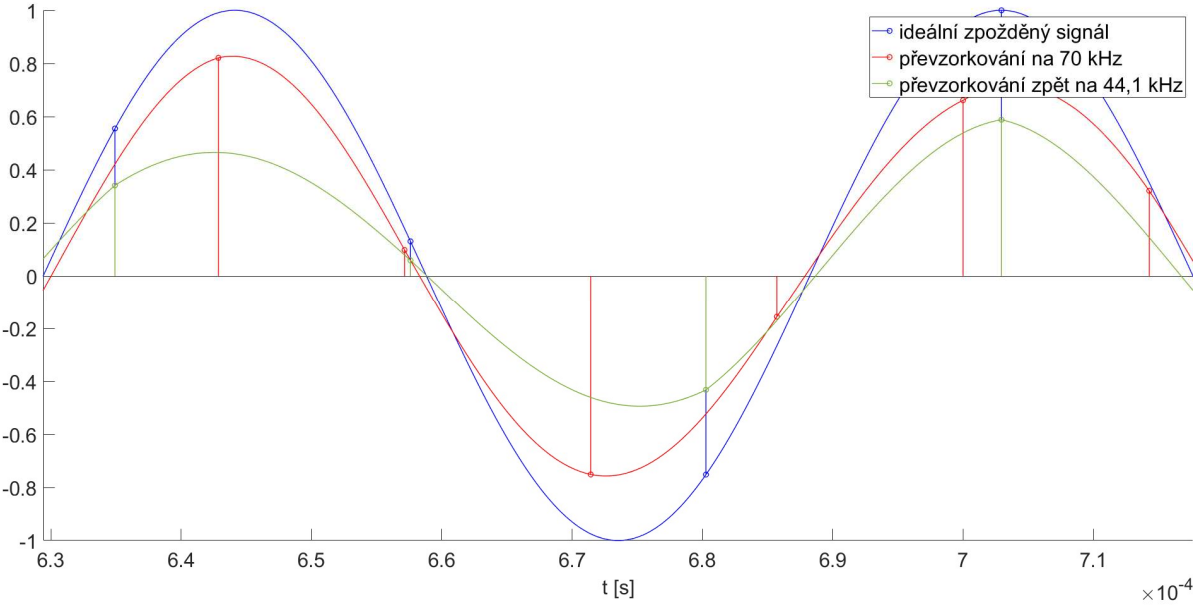
Mějme např. požadované zpoždění D = 0,1 ms a vzorkovací kmitočet f_s = 44,1 kHz. Vzorkovací perioda je tedy $T_s = 1/f_s \approx 0{,}022676$ ms. Odečteme-li od D nejbližší nižší celočíselný násobek $4 \cdot T_s$ (toto zpoždění lze realizovat kruhovou pamětí) a výsledek vydělíme vzorkovací periodou, získáme tak zlomkové zpoždění $d = \frac{D - 4T_s}{T_s} = \frac{0{,}1 - 4 \cdot 0{,}022676}{0{,}022676} \approx 0{,}41$ vzorků. Je tedy třeba signál převzorkovat.

Aby D bylo celočíselným násobkem T_s, je třeba převzorkovat na vzorkovací kmitočet, který je celočíselným násobkem $1/D = 10$ kHz. Nabízí se možnost signál nadvzorkovat s faktorem L = 100, čímž bychom získali signál se vzorkovacím kmitočtem f_{s1} = 4410 kHz. To by však zahrnovalo použití antialiasingového filtru (více je o číslicových filtrech pojednáno v kapitole 1.2.6) s normalizovaným mezním kmitočtem nižším než $\frac{1}{100}$ na velice dlouhou vyrovnávací paměť a kruhové posunutí této paměti, což by bylo výpočetně velice náročné. Namísto toho nejdříve nadvzorkujeme s faktorem L_1 = 10, čímž získáme f_{s1} = 441 kHz. Následně signál podvzorkujeme s faktorem L_2 = 7, tudíž f_{s2} bude 63 kHz. Dále nadvzorkujeme s faktorem L_3 = 10 (f_{s3} = 630 kHz) a nakonec podvzorkujeme s faktorem L_4 = 9, čímž získáme konečný kmitočet f_{s4} = 70 kHz. Provedeme posunutí o $D \cdot f_{s4} = 0{,}1 \cdot 70 = 7$ vzorků, čímž dosáhneme požadovaného zpoždění (viz obr. 1.10). Nakonec musíme signál převzorkovat zpět na f_s = 44,1 kHz. Jak je patrné, tento proces je velice výpočetně náročný a pro zpracování signálu v reálném čase nepraktický. Kvůli několika procesům nadvzorkování a podvzorkování, tedy

i filtracím antialiasingovým filtrem, navíc dochází ke značnému zkreslení signálu zejména na vysokých frekvencích podobně jako u předchozích metod (viz obr. 1.11).



Obr: 1.10: Zpoždění po převzorkování signálu



Obr. 1.11 Zkreslení signálu po převzorkování

Polyfázová implementace převzorkování

Výpočetní náročnost převzorkování i zkreslení na vysokých kmitočtech lze značně snížit využitím polyfázové implementace, která umožňuje nahradit celý postup aplikací jediného filtru typu dolní propust.

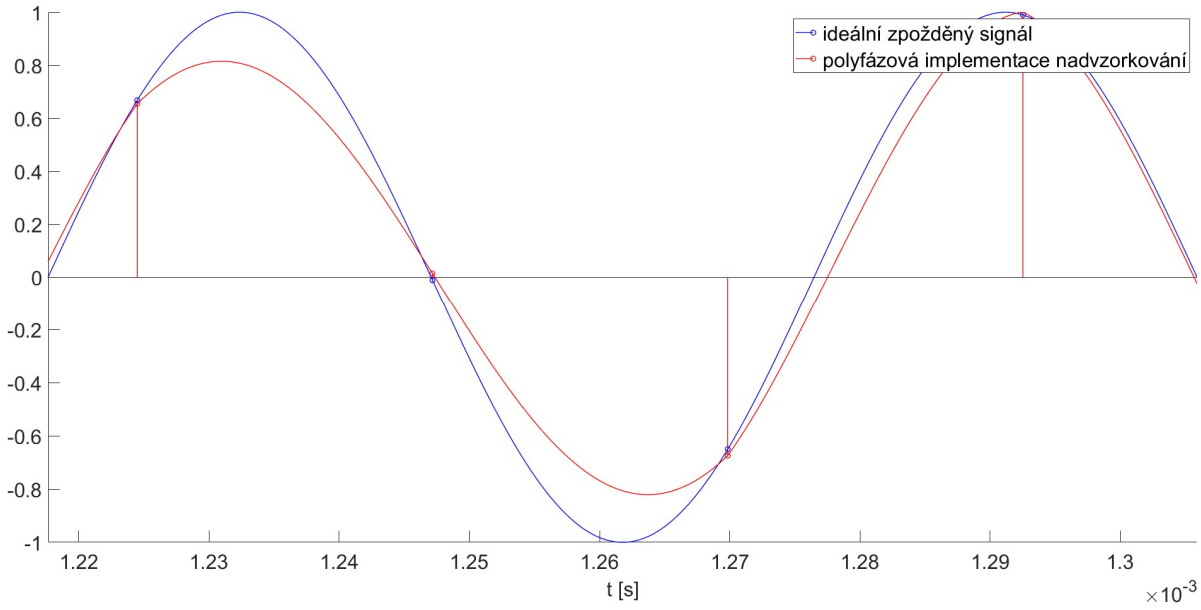
Mějme stejné parametry jako v předchozím případě, tj. D = 0,1 ms a f_s = 44,1 kHz. Opět musíme nalézt vzorkovací kmitočet, pro který je D celočíselným násobkem vzorkovací periody. Narozdíl od běžného převzorkování je však výhodné použít takový kmitočet, kterého můžeme

dosáhnout jediným procesem nadvzorkování, tzn. hledaný kmitočet je celočíselný násobek původního vzorkovacího kmitočtu. Nabízí se tedy nadvzorkování s faktorem L = 100 a vzorkovací kmitočet f_{s1} = 4410 kHz. Dalším krokem je pro toto nadvzorkování navrhnout antialiasingový filtr s normalizovaným mezním kmitočtem $\frac{1}{100}$. Abychom zajistili lineární fázi, tzn. konstantní skupinové zpoždění nezávislé na frekvenci, musí jít o symetrický či antisymetrický filtr. Zpoždění takového filtru je $D_\mathrm{f} = \frac{M-1}{2}$, kde M je délka impulzní charakteristiky, což znamená, že aby D_f bylo celý počet vzorků, M musí být liché číslo. Zároveň musí být délka M celočíselným násobkem faktoru nadvzorkování L. Návrh takového filtru je prakticky neřešitelný, lze si však pomoci vložením nulových vzorků na začátek impulzní charakteristiky, které neovlivní frekvenční odezvu filtru, pouze zajistí potřebnou délku a tím i zpoždění filtru.

Máme-li impulzní odezvu filtru IR, dalším krokem je její rozdělení na L dílčích filtrů podle vztahu:

$$\mathrm{IR}_i = \{\mathrm{IR}[i + nL]\},\ i \in \mathrm{N} \mid \mathrm{i} \leq L, n \in \mathrm{Z} \mid 0 \leq n < \frac{M}{L} - 1 \tag{1.10}$$

Tyto filtry jsou dolní propusti se stejným mezním kmitočtem jako původní filtr, ale pro vzorkovací kmitočet f_s. Jejich zpoždění odpovídá $D_i = \frac{D_\mathrm{f}}{L} + \frac{L-i}{L}$ vzorků (první člen vyjadřuje počet celých vzorků a druhý člen vyjadřuje zlomkové zpoždění). Následně vybereme potřebný filtr podle vztahu $i = (1 - d) \cdot \mathrm{L} = (1 - 0{,}41) \cdot 100 = 59$ a provedeme filtraci. Posledním krokem je zpoždění pomocí kruhové paměti o $\mathrm{floor}\left(\frac{D}{\mathrm{T_s}}\right) - \frac{D_\mathrm{f}}{L} = \mathrm{floor}\left(\frac{0{,}1}{0{,}022676}\right) - \frac{D_\mathrm{f}}{100}$ vzorků. V závislosti na délce filtru se tedy může stát, že minimální aplikovatelné zpoždění bude větší než požadované zpoždění, tím pádem tuto metodu nebude možno použít. Pokud je ovšem důležitější přesnost zpoždění signálu oproti ostatním zpracovávaným signálům než celková latence systému, tento problém lze kompenzovat zpožděním ostatních signálů o rozdíl minimálního a požadovaného zpoždění.



Obr. 1.12: Polyfázová implementace nadvzorkování

1.2.6 Číslicové filtry

Tato kapitola čerpá především z literatury [3].

Číslicové filtry jsou lineární časově invariantní diskrétní systémy (LTI, *linear time-invariant*) běžně používané pro úpravu kmitočtového spektra (nejen) zvukových číslicových signálů. Lineární diskrétní systém je takový systém, pro který platí princip superpozice, tzn. výstup pro součet signálů je roven součtu výstupů pro jednotlivé signály a výstup pro násobek vstupního signálu je roven stejnému násobku výstupu pro tento vstupní signál. Časová invariantnost (neměnnost) znamená, že systém nemění své chování v čase.

Vlastnosti LTI systému lze popsat tzv. impulzní odezvou (IR, *impulse response*), která odpovídá odezvě systému na jednotkový impulz. Výstupní signál $y[n]$ získáme konvolucí vstupního signálu $x[n]$ a impulzní odezvy $h[n]$, viz rovnice (1.11).

$$y[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m] = x[n] * h[n] \tag{1.11}$$

LTI diskrétní systémy lze rozdělit na dva typy podle délky impulzní odezvy:

- systémy s nekonečnou impulzní odezvou (IIR, *infinite impulse response*)
- systémy s konečnou impulzní odezvou (FIR, *finite impulse response*)

Mějme vstupní signál $x[n]$, který můžeme popsat rovnicí (1.12), kde A je amplituda a ω je úhlový kmitočet. Dosazením do rovnice (1.11) získáme vztah (1.13), kde $H(e^{j\omega})$ je kmitočtová charakteristika systému, kterou lze dále vyjádřit rovnicí (1.14), kde $M(e^{j\omega})$ je tzv. modulová kmitočtová charakteristika a $\varphi(e^{j\omega})$ je tzv. fázová kmitočtová charakteristika. Jelikož je $H(e^{j\omega})$ Fourierova transformace impulzní charakteristiky $h[n]$, jde o periodickou funkci s periodou 2π.

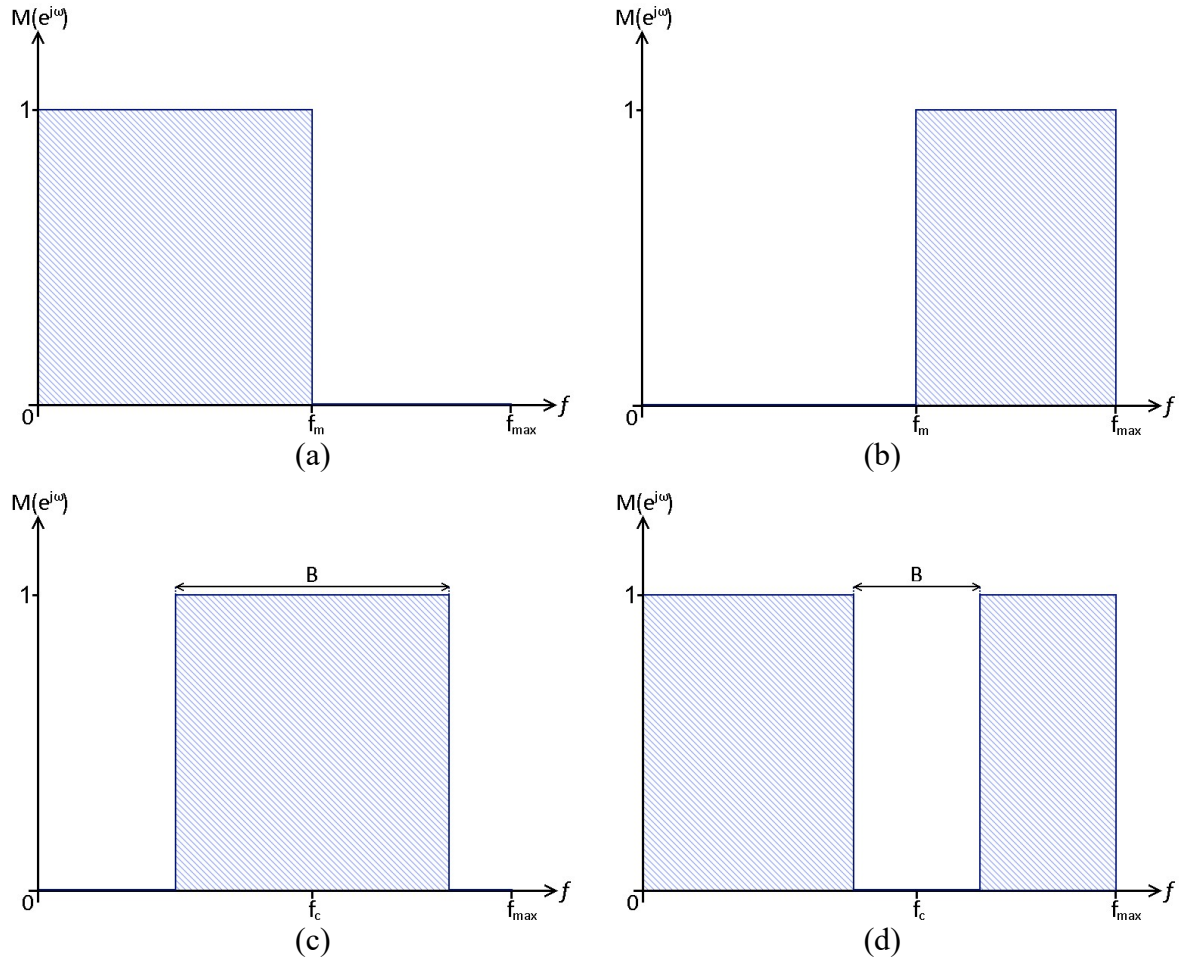
$$x[n] = Ae^{j\omega n} = A\cos(\omega n) + jA\sin(\omega n) \tag{1.12}$$

$$y[n] = H\left(e^{j\omega n}\right)x[n] \tag{1.13}$$

$$H\left(e^{\mathrm{j}\omega n}\right) = M\left(\mathrm{e}^{\mathrm{j}\omega}\ \right)e^{\mathrm{j}\varphi(e^{\mathrm{j}\omega})} \tag{1.14}$$

Kmitočtové filtry lze rozdělit na několik typů podle průběhu modulové kmitočtové charakteristiky (modulové kmitočtové charakteristiky ideálních filtrů jsou znázorněny na obr. 1.13):

- dolní propust (DP, anglicky *low-pass filter* – LPF) je takový filtr, který propouští složky vstupního signálu s nižším kmitočtem než mezní kmitočet filtru f_m
- horní propust (HP, anglicky *high-pass filter* – HPF) je filtr opačný, tzn. propouští složky s kmitočtem vyšším než mezní kmitočet f_m
- pásmová propust (PP, anglicky *band-pass filter* – BPF) je filtr, který propouští pouze složky okolo středního kmitočtu f_c s šířkou pásma B
- pásmová zádrž (PZ, anglicky *band-stop filter* – BSF) je naopak takový filtr, který složky okolo středního kmitočtu f_c s šířkou pásma B potlačuje
- fázovací článek (*all-pass filter*) je zvláštním případem filtru, který nijak neupravuje modulovou kmitočtovou charakteristiku, ale pouze fázovou kmitočtovou charakteristiku



Obr. 1.13: Modulové charakteristiky ideálních filtrů: dolní propust (a), horní propust (b), pásmová propust (c), pásmová zádrž (d)

Obecně se pásmo kmitočtů, které filtr propouští, označuje jako *propustné pásmo* (*pass band*). Pásmo, které filtr nepropouští, se pak nazývá *nepropustné pásmo* (*stop band*).

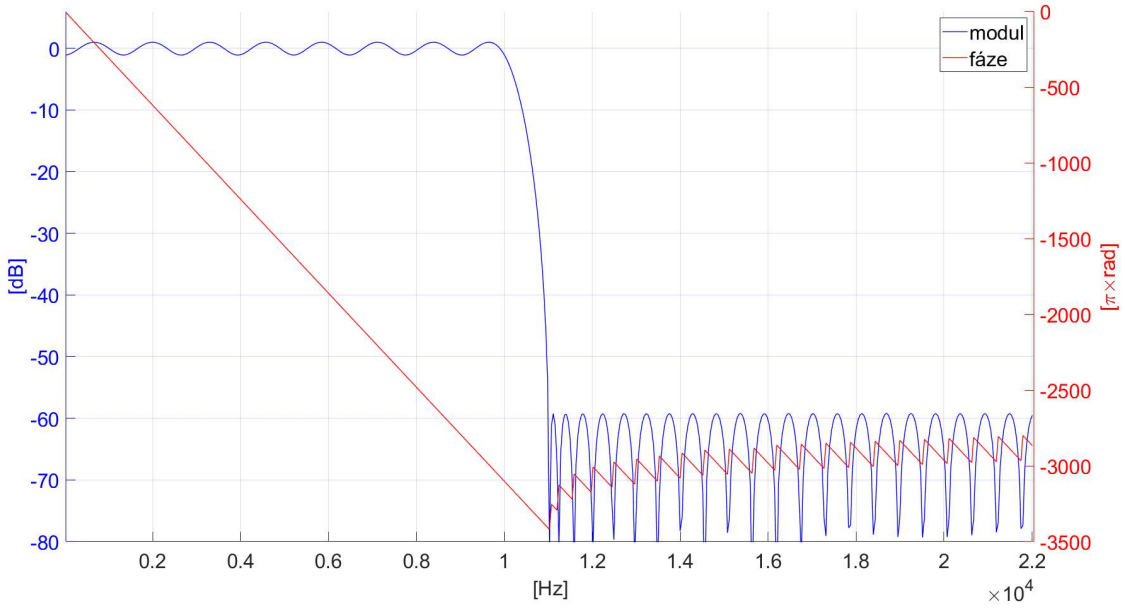
U reálných kmitočtových filtrů není přechod mezi pásmy okamžitý, zavádí se tedy ještě termín *přechodové pásmo*. Průběh propustného pásma reálných filtrů nebývá lineární, ale objevuje se *zvlnění přechodového pásma*. Zároveň *útlum nepropustného pásma* reálných filtrů není nekonečný.

IIR filtry

Filtry s nekonečnou impulzní odezvou vyžadují menší paměť pro uložení koeficientů a mezivýsledků matematických operací oproti filtrům typu FIR. Fázová kmitočtová charakteristika IIR filtrů je vždy nelineární v celém kmitočtovém rozsahu a proto dochází k fázovému zkreslení, tzn. složky signálu s různým kmitočtem jsou zpožděny o různý čas. Vzhledem k zaměření aplikace na laboratorní použití je různé zpoždění složek zcela nežádoucí a tudíž IIR filtry nebudou použity.

FIR filtry

Filtry s konečnou impulzní charakteristikou vyžadují větší paměť, než IIR filtry, což vede k většímu zpoždění při výpočtu výstupního signálu. Narozdíl od IIR filtrů lze dosáhnout lineární fázové kmitočtové charakteristiky použitím symetrického či antisymetrického filtru, díky čemuž jsou kmitočtové složky zpožděny o stejný čas, což je požadavek plánovaného použití aplikace. Na obr. 1.13 je vykreslena modulová a fázová kmitočtová charakteristika reálného FIR filtru typu dolní propust s následujícími parametry: f_s = 44,1 kHz, f_m = 10 kHz, šířka přechodového pásma 1 kHz, zvlnění propustného pásma ±1 dB a útlum nepropustného pásma 60 dB.



Obr. 1.14: Kmitočtová charakteristika reálného FIR filtru (dolní propust)

1.3 MIDI

Tato kapitola čerpá především z literatury [4].

Protokol MIDI (Musical Instruments Digital Interface) byl vytvořen ve snaze umožnit komunikaci mezi hudebními nástroji a počítačem, což by umožnilo např. automatizovanou hru, zpracování hudebních dat mimo reálný čas či snadný převod na standardní notaci. Fyzická MIDI sběrnice používá asynchronní datový přenos s rychlostí 31,25 kBit/s. Rámec se skládá ze startbitu (logická nula), osmi datových bitů a stopbitu (logická jednička).

Základní datový blok, tzv. *MIDI zpráva*, se skládá z jednoho stavového (nejvýznamnější bit MSB = 1) a několika datových MIDI bytů (MSB = 0). Stavový byte se skládá z 1bitového identifikátoru stavového bytu (1), 3bitového identifikátoru typu zprávy a 4bitového identifikátoru MIDI kanálu. Datový byte se skládá z 1bitového identifikátoru datového bytu (0) a 7bitové hodnoty. Z těchto vlastností vyplývá, že MIDI rozhraní je omezeno na 16 kanálů a MIDI zprávy mohou podle počtu datových bytů mít rozsah 0-127 či 0-16383 (více než 2 datové byty může obsahovat pouze *SysEx* zpráva).

1.3.1 MIDI zprávy

MIDI zprávy jsou rozděleny na tzv. *kanálové zprávy* (Channel Message) a *systémové zprávy* (System Message). Systémové zprávy jsou společné pro všechny kanály a nepřenáší tak identifikátor kanálu, dolní čtyři bity jsou namísto toho určeny pro identifikaci typu systémové zprávy.

Kanálové zprávy

Jak již bylo zmíněno, kanálové zprávy obsahují identifikátor MIDI kanálu a pro identifikátor typu zprávy tak zbývají tři bity, tzn. sedm možných identifikátorů (osmý je vyhrazený pro systémové zprávy) – *Note Off*, *Note On*, *Polyphonic Key Pressure*, *Control Change*, *Program Change*, *Channel Pressure* a *Pitch Bend Change*. Aplikace bude pracovat pouze se zprávami typu *Control Change* (změna kontroleru, identifikátor = 3), ostatní typy tedy není třeba dále popisovat.

Zprávu typu *Control Change*, zkráceně CC, se přenášejí informace o stavu kontrolních prvků, např. tlačítek či potenciometrů. Tento typ zprávy se skládá z jednoho stavového a dvou datových bytů. První datový byte určuje číslo kontroleru a druhý datový byte určuje jeho novou hodnotu. Některé MIDI kontrolery mají vyhrazeny dvě MIDI čísla, jedno pro nejvýznamnější a jedno pro nejméně významný byte, mohou tudíž nabývat hodnot 0 až 16383.

Spojité MIDI kontrolery mohou nabývat všech hodnot v jejich rozsahu a používají se např. jako ovladače hlasitosti či pozice ve stereu.

Spínače mají pouze dva stavy – „vypnuto“ (0-63) a „zapnuto“ (64-127). Jde například o tlačítko *Mute*.

U inkrementačních/dekrementačních zpráv a tzv. povelů nezáleží na hodnotě přenášené zprávy, jejich funkce je definována pouze přijetím čísla kontroleru. Výjimkou je kontroler č. 122 (*Local Control*), kde 0 je vypnuto a 127 je zapnuto.

1.4 Jazyk C++

C++ je programovací jazyk vycházející z jazyka C. První publikovaná verze tohoto jazyka byla vytvořena Bjarnem Stroustrupem v letech 1979 až 1985, od té doby se však jazyk dále vyvíjel a bylo vydáno několik dalších verzí. Aktuální verzí je C++20 vydaná v prosinci 2020. [8]

C++ podporuje řadu programovacích stylů jako například objektově orientované či funkcionální programování. Oficiální dokumentace je obsažena v mezinárodním standardu jazyka C++, který je nutno zakoupit. Vzhledem k popularitě jazyka (podle TIOBE indexu se jedná o čtvrtý nejpopulárnější programovací jazyk) ovšem existuje i velké množství neoficiální dokumentace, tutoriálů i komunit věnujících se jeho využití.

1.5 Knihovna JUCE

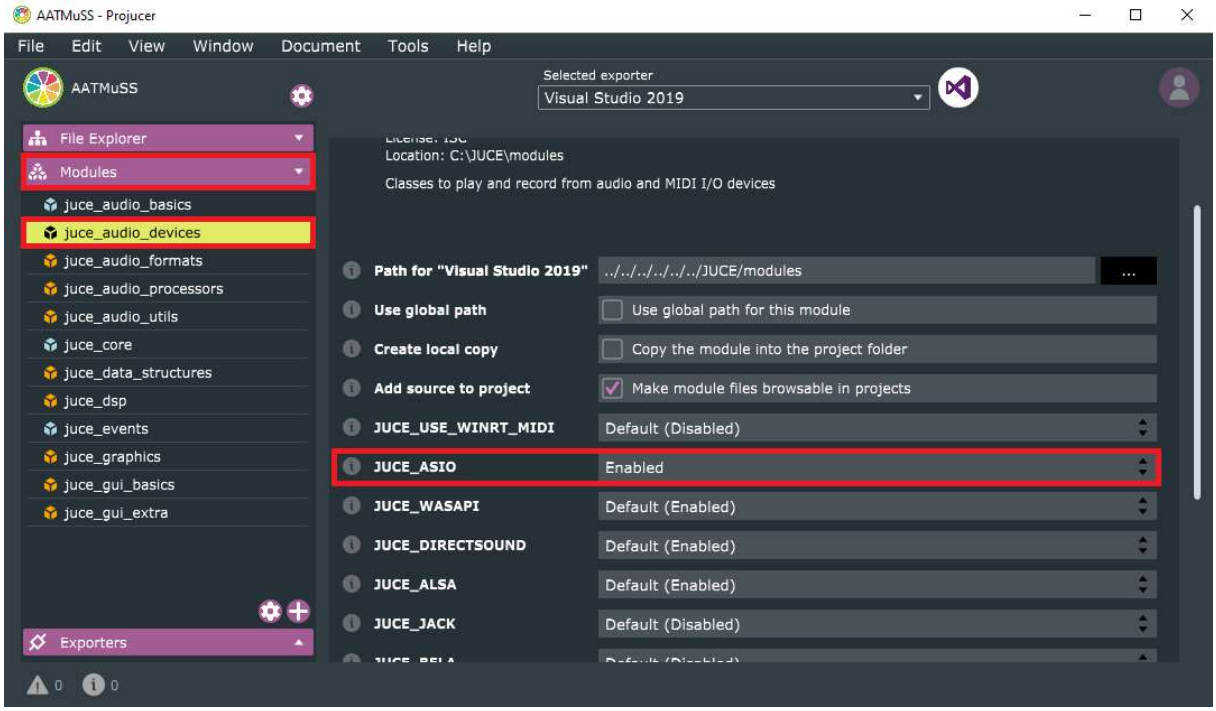
JUCE je knihovna napsaná v jazyce C++ používaná pro vývoj aplikací pro řadu operačních systémů, zahrnují nejen OS pro počítače (např. Windows, macOS), ale i pro mobilní zařízení (Android, iOS). Dále je knihovna určená i pro vývoj audio pluginů různých formátů (např. VST, AAX). Velkou výhodou JUCE je možnost sestavovat aplikace pro různé operační systémy ze stejného zdrojového kódu.

Knihovna JUCE byla vytvořena Julianem Storerem a byla poprvé vydána v roce 2004. Aktuální stabilní verzí je JUCE 6.1 vydaná v srpnu 2021. [9] Při práci s touto knihovnou je možné využívat nástroj Projucer, který usnadňuje vytváření a správu projektů v řadě vývojových prostředí (např. Visual Studio, XCode). Na oficiálních stránkách je také k dispozici kompletní dokumentace [10] všech modulů, tříd a funkcí JUCE, jakož i tutoriály [11], které mohou značně usnadnit osvojení si základů práce s JUCE. Aktivní komunita sídlí především na oficiálním fóru, kde na dotazy, diskuse či hlášení o problémech často odpovídají i tvůrci JUCE, což mimo jiné vede i k rychlým opravám bugů v rámci vývojové větve knihovny.

1.6 Ovladače ASIO

Protokol pro ovladače zvukových karet ASIO (Audio Stream Input/Output) je vytvořený společností Steinberg. Je k dispozici pro Windows a v experimentální verzi také pro Linux. Výhodou těchto ovladačů oproti např. DirectSound je výrazně nižší zpoždění, tzv. latence (obecně v řádu milisekund), což je nezbytné pro zpracování zvuku v reálném čase.

Vzhledem k licenční smlouvě se společností Steinberg je ve výchozím stavu knihovny JUCE podpora ovladačů ASIO vypnutá. Pro její zapnutí a využití ovladačů je třeba stáhnout ASIO SDK (Software Development Kit) z webových stránek Steinberg [12]. Následně je třeba přidat umístění podsložky `common` obsahující hlavičkové soubory do `Header Search Paths` v nastavení projektu v Projucer. Posledním krokem je povolit JUCE_ASIO v modulu juce_audio_devices (viz obr. 1.15).



Obr. 1.15: Povolení ASIO v juce_audio_devices

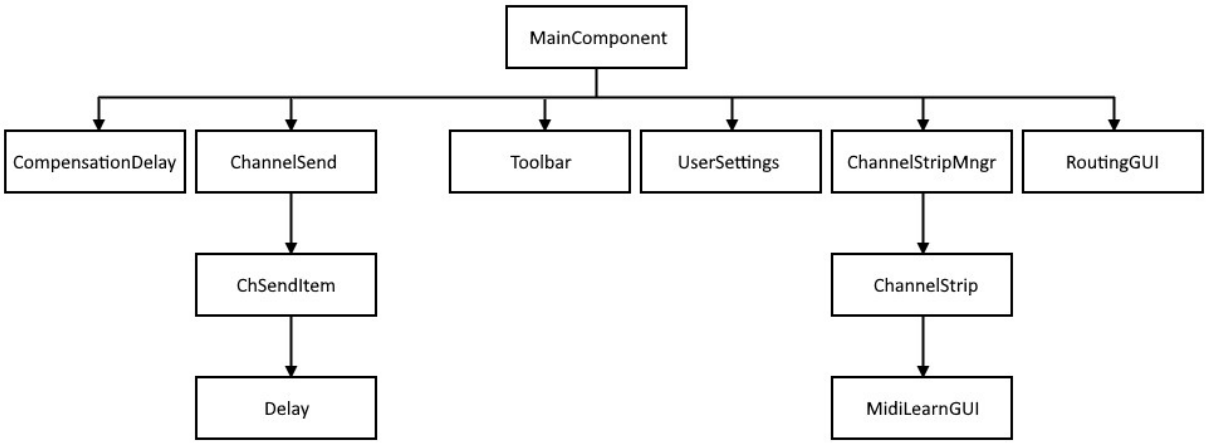
2 POPIS APLIKACE

V této části aplikace je popsána vnitřní struktura aplikace, funkce a hierarchie vytvořených tříd a vlastní zpracování zvukového signálu. Složka projektu je k dispozici na tomto odkaze.

Program byl vytvořen na základě JUCE šablony Audio Application, která se skládá z jednoho okna a audio procesoru. Umožňuje také ovládání parametrů pomocí MIDI zpráv. Aplikace byla pojmenována *Audio Adjustment Tool for Multi-Speaker Systems*, zkráceně *AATMuSS*.

2.1 Vnitřní struktura aplikace

Zdrojové soubory jsou rozděleny do několika skupin – *Audio*, *GUI* a *MainComponent*. Výjimkou jsou soubory `Main.cpp` a `Constants.h`. `Main.cpp` obsahuje instrukce a parametry, podle nichž JUCE generuje funkci `main()`. Ta slouží k inicializaci a spuštění programu. `Constants.h` obsahuje konstanty a enumerátory použité v aplikaci. Jsou zde uloženy například rozměry grafických prvků či koeficienty FIR filtrů používaných pro polyfázovou interpolaci.



Obr. 2.1: Hierarchie tříd aplikace

2.1.1 Audio

Skupina *Audio* obsahuje třídy, které mají na starosti zpracování audio signálu. Třídy `ChannelSend`, `ChSendItem`, `Delay` a `CompensationDelay` obsahují pouze několik proměnných a funkcí, jejich deklarace a definice tudíž nebyly rozděleny do samostatných souborů. Aby zdrojový kód při kompilaci nebyl obsažen více než jednou, je v každé třídě umístěn příkaz `#pragma once`. Samotný proces zpracování signálu je blíže popsán v kapitole 2.2.

ChSendItem

Objekty `ChSendItem` by se daly označit za základní stavební prvky z hlediska zpracování zvukového signálu, neboť v nich probíhá k aplikaci zesílení, inverze polarity a prostřednictvím objektu `Delay` i zpoždění. Každý objekt obsahuje indexy vstupního a výstupního kanálu, ke kterým je přiřazen. Není tedy třeba, aby byla objektu předávána vstupní data a aby tento objekt

upravená data vracel, což by vedlo k alokaci potenciálně velkých úseků paměti při každém cyklu zpracování signálu. Namísto toho je mu předáván pouze ukazatel na vstupní, resp. výstupní vyrovnávací paměť.

Výpis kódu 2.1: `ChSendItem` – funkce `readInput()`

```
void ChSendItem::readInput(juce::AudioBuffer<float>* inputBuffer)
{
  if (delayObj == nullptr) buffer.copyFrom(0, 0, *inputBuffer, inputIndex,
                                           0, bufferSize);
  else delayObj->processDelay(inputBuffer, &buffer);
  processGain();
}
```

Členské proměnné této třídy zahrnují vyrovnávací paměť pro zpracování signálu, zesílení v rozsahu −1 až 1, objekt `Delay` a proměnné související se změnami stavu. Funkce této třídy slouží ke čtení ze vstupních a zapisování do výstupních kanálů vyrovnávací paměti, zahájení deaktivace a zjištění stavu objektu.

ChannelSend

`ChannelSend` lze považovat za nadřazenou třídu `ChSendItem`, neboť tyto objekty spravuje a slouží k jejich komunikaci s procesorem ve formě předávání pokynů ke čtení či zápisu s ukazatelem na právě zpracovávanou vyrovnávací paměť. Správa objektů `ChSendItem` zahrnuje vyhodnocování, zda je nutné měnit parametry zpracování signálu, a s tím související předávání pokynů k deaktivaci, odstraňování a vytváření nových objektů s aktuálními parametry.

Objekty této třídy obsahují indexy přiřazených kanálů a délku přechodu na nové hodnoty (crossfade), délku vyrovnávací paměti, vzorkovací kmitočet a časovač pro změny hodnot. Dále také obsahují všechny potřebné hodnoty související se zpracováním zvukového signálu, tj. zesílení, ztlumení (*mute*), invertování signálu, zpoždění a mód zpoždění. Každý z těchto parametrů je uložen ve dvou proměnných reprezentujících aktuální a budoucí hodnoty – viz kapitola 2.2.2.

Delay

Třída `Delay` má na starosti zpoždění signálu pomocí kruhové vyrovnávací paměti a metod popsaných v kapitole 1.2.5. Obsahuje index přiřazeného vstupního kanálu, mód zpoždění, proměnné související se zápisem a čtení z kruhové paměti a případný FIR filtr používaný při polyfázové implementaci nadvzorkování.

CompensationDelay

Třída `CompensationDelay` je v podstatě značně zjednodušená kombinace tříd `ChSendItem` a `Delay`, jejímž úkolem je pouze zajistit kompenzaci zpoždění FIR filtru použitého pro realizaci zlomkového zpoždění. Objekty této třídy obsahují indexy přiřazeného vstupního a výstupního kanálu, kruhovou vyrovnávací paměť a proměnné související se změnami stavu a čtením a zápisem ze/do zpracovávané vyrovnávací paměti.

2.1.2 GUI

Skupina *GUI* obsahuje soubory, které definují a zpracovávají uživatelské rozhraní. Jelikož jsou mnou vytvořené grafické třídy poměrně jednoduché, deklarace a definice jejich funkcí a proměnných jsou sjednoceny do souborů `ChannelStrip.h`, `ChannelStripMngr.h`, `MidiLearnGUI.h`, `Toolbar.h`, `UserSettings.h` a `RoutingGUI.h` (obdobně jako třídy v kapitole 2.1.1).

ChannelStrip

Třída `ChannelStrip` obsahuje grafické prvky pro ovládání parametrů zpracování signálu jednoho kanálu. Konkrétně se jedná o posuvné kontroléry pro zesílení a zpoždění signálu, dále také tlačítka pro ztlumení kanálu, invertování polarity signálu, skrytí daného `ChannelStripu`, odstranění kanálu a zobrazení prvku pro přiřazování MIDI zpráv k jednotlivým parametrům zpracování signálu.

Při inicializaci jsou objektu této třídy předány indexy příslušného vstupního a výstupního kanálu a všechny hodnoty, které grafické prvky této třídy ovládají (tzn. hodnoty zesílení a zpoždění a stav ztlumení kanálu a invertování polarity), jakož i parametr `maxDelay`, který určuje horní hranici hodnot kontroléru zpoždění. Dalšími parametry konstruktoru jsou ukazatele na objekty `Listener`, které umožňují aplikaci zaznamenávat a reagovat na změny polohy posuvníků a kliknutí na tlačítka, viz kapitola 2.1.3, část *MC_Listeners.cpp*.

ChannelStripMngr

`ChannelStripMngr` má na starosti vytváření, správu, zobrazení a odstraňování objektů `ChannelStrip`. Ukazatele na tyto objekty jsou ukládány ve třech maticích, jejichž rozměry odpovídají maximálnímu počtu kanálů, tudíž je možný snadný přístup k libovolnému objektu na základě indexů příslušných kanálů. Matice jsou tři kvůli rozdělení na pre-processing, processing a post-processing.

Hlavní objekt aplikace `MainComponent` obsahuje právě jeden objekt `ChannelStripMngr` umístěný do tzv. *viewportu*. Pokud jsou celkové rozměry větší než rozměry viewportu, automaticky se vpravo a/nebo dole zobrazí posuvník umožňující posouvání zobrazení. `ChannelStripMngr` aktualizuje svou velikost, jakož i zobrazení či skrytí ovládacích prvků jednotlivých kanálů a jejich umístění, pomocí metody `updateAll()`, která je volána časovačem každých 200 ms. Je tak zajištěna pravidelná aktualizace grafických prvků nezávislá na jiných procesech aplikace, což je výhodné např. při změnách parametrů pomocí MIDI zpráv. Pokud by totiž byly požadavky na změny volány přímo z funkce zpracovávající MIDI zprávy, může dojít ke konfliktu mezi jednotlivými vlákny.

Výpis kódu 2.2: `ChannelStripMngr` – funkce `updateAll()`

```
void updateAll(juce::Slider::Listener* sliderListener,
               juce::Button::Listener* buttonListener,
               juce::ValueTree valueTreeRoot)
{
  valTreeRoot = valueTreeRoot;
  ChannelStrip* curStrip;
  juce::ValueTree curTree;
  bool show = false;
  for (int in = -1; in < constants::MAX_NUM_INPUTS; in++)
  {
    for (int out = -1; out < constants::MAX_NUM_OUTPUTS; out++)
    {
      if (in == -1 && out == -1) continue;
      curTree = findValueTreeSend(in, out);
      if (out == -1) curStrip = inputStrips[in].get();
      else if (in == -1) curStrip = outputStrips[out].get();
      else curStrip = channelStrips[in][out].get();

      show = (curTree.hasType("send") || curTree.hasType("input") ||
              curTree.hasType("output")) &&
              curTree.getProperty(ValTreeSendProperty::visible);

      if (curStrip == nullptr && show) addStrip(in, out,
                                                sliderListener, buttonListener);
      else if (curStrip != nullptr && !show) removeStrip(in, out);
      else if (curStrip != nullptr && show) updateStrip(in, out);
    }
  }
}
```

MidiLearnGUI

`MidiLearnGUI` je jednoduché menu sloužící k přiřazování MIDI zpráv k parametrům zpracování jednotlivých kanálu jako např. zesílení či zpoždění. Pro každý parametr je zobrazen jeho název, tlačítko pro zahájení/dokončení procesu přiřazování, číslo přiřazeného MIDI kanálu a MIDI zprávy (např. c7n64, tzn. kanál 7, číslo 64) a tlačítko pro odstranění daného přiřazení. Objekty této třídy jsou vytvářeny a spravovány jednotlivými objekty `ChannelStrip`.

Toolbar

Objekt třídy `Toolbar` je jediným grafickým prvkem aplikace, který je viditelný ve všech situacích. Slouží k přepínání zobrazeného obsahu mezi channel stripy, uživatelským nastavením, nastavením zařízení a směrovací maticí pomocí několika tlačítek. Také obsahuje tlačítka pro uložení a načtení presetu a dva objekty třídy `FilenameComponent` pro výběr XML souboru s nastavením.

UserSettings

V `UserSettings` jsou umístěny posuvníky pro konfiguraci délky přechodu na nové hodnoty ve zpracování audio signálu (crossfade) a maximálního nastavitelného zpoždění. Dále je zde také výběr módu zpoždění pomocí tlačítek, jakož i popis všech parametrů nastavení. V aplikaci je přítomen vždy jen jeden objekt této třídy a podobně jako `ChannelStripMngr` je umístěn do viewportu.

RoutingGUI

Poslední grafickou třídou je `RoutingGUI`. Stejně jako `ChannelStripMngr` a `UserSettings` je v aplikaci pouze jeden objekt této třídy a je umístěn do viewportu. Tato třída obsahuje zaškrtávací políčka graficky uspořádaná do tří matic – směrovací matice, matice pro správu zobrazení `ChannelStripů` a kombinovaná matice pro pre- a post-processing (viz kapitola 2.2). Umístění těchto políček a popisných textů, jakož i velikost samotné komponenty, se mění na základě počtu kanálů vybraného zvukového zařízení.

2.1.3 MainComponent

Poslední skupina obsahuje všechny soubory náležící ke třídě `MainComponent`. Deklarace všech proměnných a funkcí jsou v hlavičkovém souboru `MainComponent.h`, definice jsou však rozděleny do několika `.cpp` souborů kvůli přehlednosti.

MainComponent.cpp

`MainComponent.cpp` obsahuje konstruktor a destruktor třídy. Konstruktor volá funkce, které inicializují grafické rozhraní a načtou parametry pro zpracování signálu jednotlivých kanálů. Dále také připraví používané audio zařízení a v případě potřeby požádá platformu o povolení otevřít vstupní kanály.

MC_GUI.cpp

V souboru `MC_GUI.cpp` jsou funkce, které mají na starosti vytvoření grafického rozhraní a jeho změny (např. zobrazení uživatelského nastavení). K těmto změnám slouží funkce `showComponent()` – za zmínku stojí, že funkce volá samu sebe, ale díky podmínkám vyhodnocujícím vstupní parametry nemůže dojít k zacyklení.

Výpis kódu 2.3: `MC_GUI.cpp` - část funkce `showComponent()`

```
void MainComponent::showComponent(View type, bool show)
{
    if (show)
    {
        if (type == View::channelStrips)
        {
            showComponent(View::deviceSettings, false);
            showComponent(View::userSettings, false);
            showComponent(View::routing, false);
            chStripViewport.setVisible(true);
        }
    ...
    }
    else
    if (type == View::channelStrips)
    {
        chStripViewport.setVisible(false);
    }
```

MC_Listeners.cpp

`MC_Listeners.cpp` obsahuje funkce určené ke zpracování událostí, konkrétně změny hodnot posuvných kontrolérů a stisknutí tlačítek. Vstupním argumentem těchto funkcí je

ukazatel na komponentu, které se událost týká. Pomocí podmínkového větvení `if-else` je vyhodnoceno, jaké kroky je třeba podniknout a jsou volány příslušné funkce, ať už se týká změn parametrů zpracování signálu nebo úpravy grafického rozhraní.

Výpis kódu 2.4: `MC_Listeners.cpp` – část funkce `sliderValueChanged()`

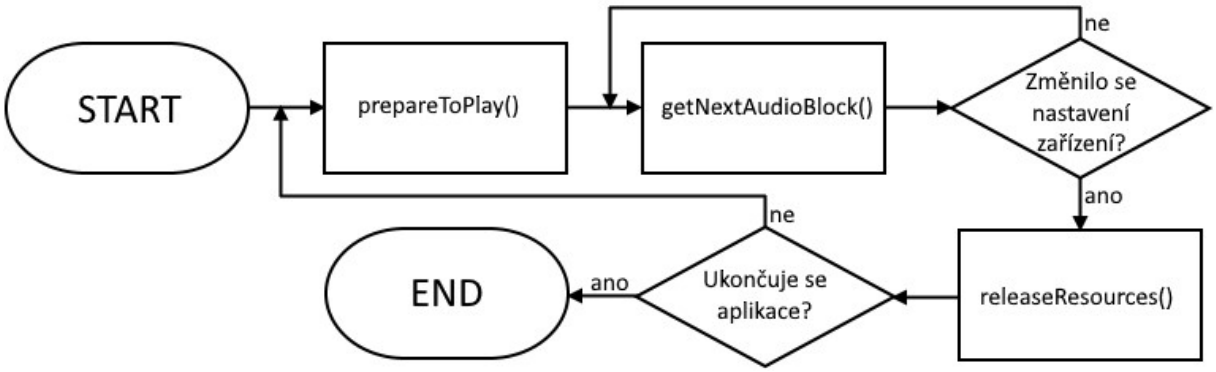
```
/*Listener for slider events.*/
void MainComponent::sliderValueChanged(juce::Slider* slider)
{
  // ChannelStrip
  if (slider->findParentComponentOfClass<ChannelStrip>() ==
      slider->getParentComponent())
  {
    ChannelStrip* parent = slider->findParentComponentOfClass<ChannelStrip>();
    int in = parent->getInputIndex();
    int out = parent->getOutputIndex();

    if (slider == &parent->gainSlider)
    {
      changeSendValue(in, out, ValTreeSendProperty::gain,
          (float)juce::Decibels::decibelsToGain(slider->getValue()));
      changeTreeValue(in, out, ValTreeSendProperty::gain,
          (float)juce::Decibels::decibelsToGain(slider->getValue()));
    }
    else if (slider == &parent->delaySlider)
    {
      changeSendValue(in, out, ValTreeSendProperty::delay,
          (float)slider->getValue() / 1000.0f);
      changeTreeValue(in, out, ValTreeSendProperty::delay,
          (float)slider->getValue());
    }
}
```

MC_ValueTree.cpp

Soubor `MC_ValueTree.cpp` obsahuje všechny funkce, které souvisí s objekty třídy `ValueTree`. Jedná se především o ukládání a načítání nastavení do/z XML souborů a kontroly, případně i vytvoření výchozích souborů při spuštění aplikace. Také jsou zde i funkce pro vyhledávání a změny hodnot obsažených v hlavní instanci objektu `ValueTree`.

MC_Audio.cpp

Dalším souborem je `MC_Audio.cpp`, ve kterém jsou umístěny funkce související se zpracováním audio signálu a směrovací maticí. Funkce `prepareToPlay()` je volána po spuštění audio zařízení či změně jeho nastavení. Vytváří objekty třídy `ChannelSend` s parametry získanými z `ValueTree` a přiřazuje je do směrovací matice a vektorů vstupních a výstupních kanálů (viz kapitola 2.2) na základě indexů příslušného vstupního a/nebo výstupního kanálu. Funkce `releaseResources()` je volána před zastavením audio zařízení nebo změně jeho nastavení a všechny objekty třídy `ChannelSend` vymaže. Funkce `getNextAudioBlock()` má již na starosti zpracování audio signálu a také průběžnou správu objektů `ChannelSend`.



Obr. 2.2: Vývojový diagram hlavní činnosti procesoru

MC_MIDI.cpp

V souboru `MC_MIDI.cpp` jsou funkce, které mají na starosti příjem MIDI zpráv a jejich zpracování, jakož i odstranění MIDI mapy z hlavní instance objektu `ValueTree`.

2.2 Zpracování signálu

V této kapitole je popsán proces zpracování audio signálu, který zahrnuje objekty tříd zběžně popsaných v kapitole 2.1.1 (`ChannelSend`, `ChSendInstance`, `Delay` a `CompensationDelay`) a funkcí zmíněných v kapitole 2.1.3 (část `MC_Audio.cpp`). Nejprve je charakterizován průběh zpracování signálu v rámci třídy `MainComponent`, dále pak v rámci podřazených tříd `ChannelSend`, `ChSendInstance` a `Delay`.

2.2.1 MainComponent

Vstupní a výstupní vyrovnávací paměť je sdílená, je tedy vždy třeba nejprve zkopírovat vstupní audio data, následně vyrovnávací paměť vyčistit, a nakonec do ní zapsat výstupní data. Pokud by tento postup nebyl dodržen, mohlo by dojít k přepisování dat nebo přítomnosti nežádoucího obsahu při zpracování více kanálů.

Jak již bylo zmíněno, funkce `prepareToPlay()` vytváří objekty třídy `ChannelSend` a přiřazuje je do směrovací matice `sendMatrix`. Kromě směrovací matice jsou také používány vektory `inputMatrix` a `outputMatrix`, do nichž jsou přiřazovány objekty stejné třídy, ale určené pro zpracování před, resp. po směšování signálů.

Funkci `getNextAudioBlock()` můžeme rozdělit do tří částí s velmi podobnou funkcí – *pre-processing*, *processing* a *post-processing*.

Kvůli přehlednosti je nejprve popsán průběh druhé části, tedy zpracování vstupních signálů a jejich směšování do výstupních kanálů pomocí objektů v matici `sendMatrix`. Nejdříve proběhne aktualizace objektů `ChannelSend` a případné přidání nových či odstranění již nepotřebných objektů ze směrovací matice. Následně je u všech těchto objektů zavolána členská funkce `readInputs()` s argumentem obsahujícím ukazatel na vyrovnávací paměť. Dále je vyrovnávací paměť vyčištěna. Nakonec je u všech objektů `ChannelSend` ve směrovací matici zavolána členská funkce `writeOutputs()` s ukazatelem na vyrovnávací paměť stejně jako v případě čtení.

První a třetí část fungují podobně. Hlavní rozdíl je v tom, že pokud objekt `ChannelSend` pro daný vstup či výstup neexistuje, signál je ponechán beze změny. Výjimka nastává, pokud uživatel zvolí nadvzorkování jako mód zpoždění. V takovém případě je třeba kompenzovat zpoždění polyfázového filtru v kanálech, které jej nepoužívají (jejich zpoždění je nulové nebo odpovídá celočíselnému násobku vzorkovací periody). Je tedy také třeba zpozdit vstupní a výstupní kanály, k nimž není žádný objekt `ChannelSend` přiřazen (v rámci `inputMatrix` či `outputMatrix`, nikoliv `sendMatrix`). Pro tento případ byla vytvořena třída `CompensationDelay`.

Výpis kódu 2.5: `MC_Audio.cpp` – část funkce `getNextAudioBuffer()`

```
// read & write
for (int in = 0; in < maxInputChannels; in++)
{
  // leave signal as is
  if (!inputMatrix[in].has_value() && !inCompensationDelay[in].has_value())
       { continue; }

  // read signal
  if (inputMatrix[in].has_value() && activeInputChannels[in])
       { inputMatrix[in]->readInputs(bufferToFill.buffer); }

  if (inCompensationDelay[in].has_value() && activeInputChannels[in])
       { inCompensationDelay[in]->readInput(bufferToFill.buffer); }

  // clear
  bufferToFill.buffer->clear(in, 0, bufferSize);

  // write signal
  if (inputMatrix[in].has_value() && activeInputChannels[in])
       { inputMatrix[in]->writeOutputs(bufferToFill.buffer); }

  if (inCompensationDelay[in].has_value() && activeInputChannels[in])
       { inCompensationDelay[in]->writeOutput(bufferToFill.buffer); }
}
```

2.2.2 ChannelSend

Členská funkce `update()` nejprve projde pole objektů `ChSendInstance`, odstraní nepotřebné objekty a předá všem pokyn k deaktivaci, pokud je sám objekt `ChannelSend` určen k odstranění. Dále vyhodnotí, zda je připraven ke změně parametrů, tzn. zda se některý z parametrů změnil a odpočet odpovídající cca 200 ms došel k nule – odpočet probíhá v počtu vzorků namísto milisekund a změny jsou prováděny pouze po přijetí nové vyrovnávací paměti, konkrétní hodnota tedy závisí na vzorkovacím kmitočtu a délce vyrovnávací paměti. Toto vyhodnocení neprobíhá v případě, že je kanál ztišen (*mute*). Nové hodnoty parametrů jsou objektu `ChannelSend` předávány funkcemi typu `Listener` při změně k nim přiřazených grafických prvků (viz kapitola 2.1.2). Pokud je objekt připraven ke změně parametrů, předá existujícím objektům `ChSendInstance` pokyn k deaktivaci a vytvoří nový objekt s novými hodnotami.

Funkce `readInputs()` u všech příslušných objektů `ChSendInstance` zavolá členskou funkci `readInput()`, jejímž argumentem je ukazatel na právě zpracovávanou vyrovnávací paměť. Stejně tak funguje i funkce `writeOutputs()`.

2.2.3 ChSendInstance

V objektech třídy `ChSendInstance` dochází k samotné práci se zvukovým signálem. K tomuto účelu slouží vyrovnávací paměť s jedním kanálem, jejíž délka odpovídá délce vyrovnávací paměti určené zvukovým zařízením.

Členská funkce `readInput()` buď předá ukazatel na právě zpracovávanou vyrovnávací paměť a členskou vyrovnávací paměť přiřazenému objektu `Delay` (pokud existuje), nebo zkopíruje data z přiřazeného kanálu zpracovávané vyrovnávací paměti přímo do členské vyrovnávací paměti. Nakonec je aplikováno zesílení, k čemuž slouží členská funkce `processGain()`. Ta má na starosti i postupné zesílení po vytvoření objektu (*fade-in*) a zeslabení po deaktivaci (*fade-out*), tedy tzv. *crossfade*. Ke crossfade však nedochází vždy ihned po aktivaci či deaktivaci – pokud je nastaveno nenulové zpoždění, deaktivovaný objekt po dobu odpovídající délce zpoždění ještě pracuje v normálním režimu, zatímco z nového objektu nejsou v rámci funkce `writeOutput()` kopírována data. Důvodem k tomuto postupu je nutnost zaplnění kruhové paměti nového objektu `Delay`, aby nedošlo k prudkému skoku z nulové hodnoty prázdného vzorku na nenulovou hodnotu, a tudíž slyšitelnému praskání.

Funkce `writeOutput()` kopíruje data z vyrovnávací paměti objektu do příslušného výstupního kanálu právě zpracovávané vyrovnávací paměti, přičemž nedochází k přepsání existujících hodnot kopírovanými, nýbrž k jejich součtu.

2.2.4 Delay

Jak již název napovídá, tato třída má na starosti zpoždění signálu.

Při vytvoření objektu této třídy je na základě vstupních parametrů vyhodnoceno, jaký mód zpoždění bude použit, např. pokud je požadován kterýkoliv typ interpolace a požadované zpoždění odpovídá celému počtu vzorků, stačí zpoždění zpracovávat pouhým posunem v kruhové paměti. Následně jsou přizpůsobeny některé parametry, jako např. pozice čtení či délka kruhové paměti.

V rámci samotného zpracování signálu je, jak již bylo zmíněno, volána funkce `processDelay()` s ukazateli na vstupní a výstupní vyrovnávací paměť. V kontextu použití tohoto objektu jde o právě zpracovávanou vyrovnávací paměť a vyrovnávací paměť objektu `ChSendItem`. Nejprve je vstupní signál zkopírován do členské kruhové vyrovnávací paměti, což zajistí zpoždění o požadovaný celý počet vzorků. V případě módů zaokrouhlování a nadvzorkování je následně posunutý signál zkopírován do výstupní vyrovnávací paměti. Při nadvzorkování je na tuto paměť následně aplikován FIR filtr, který signál zpozdí o zlomek vzorku – zpoždění o celé vzorky je v kanálech, které tento filtr nepoužívají, kompenzováno delší kruhovou pamětí, tudíž jednotlivé kanály zachovávají správné vzájemné zpoždění. V případě lineární, kvadratické či kubické interpolace je posunutý signál taktéž kopírován do výstupní vyrovnávací paměti, zároveň je však prováděn přepočet hodnot signálu podle příslušné interpolační metody.

2.2.5 CompensationDelay

Třída `CompensationDelay` je určena pro kompenzaci zpoždění výše zmíněného FIR filtru ve vstupních a výstupních kanálech (tzn. pouze pre- a post-processing), které nejsou jinak upravovány. V podstatě jde o kombinaci vybraných funkcí a proměnných objektů `ChSendItem` a `Delay` se schopností komunikovat přímo s procesorem, není tedy třeba spravující objekt `ChannelSend`.

Při vytvoření objektu je na základě vzorkovacího kmitočtu (a tedy délky použitého filtru) vypočteno potřebné kompenzační zpoždění a délka kruhové vyrovnávací paměti.

V rámci samotného zpracování signálu je nejdříve volána funkce readInput() s ukazatelem na právě zpracovávanou vyrovnávací paměť, která obdobně jako u objektu `ChSendItem` zkopíruje signál příslušného kanálu do kruhové vyrovnávací paměti. Dále je členskou funkcí `processGain()` zpracováno zesílení, v rámci tohoto objektu je však zpracován pouze crossfade, neboť účelem tohoto objektu není jakkoliv upravovat signál kromě zpoždění. Poté je v objektu `MainComponent` daný kanál zpracovávané vyrovnávací paměti vyčištěn.

Nakonec je zavolána funkce `writeOutput()`, která zkopíruje zpožděný signál do příslušného kanálu právě zpracovávané vyrovnávací paměti, přičemž nejsou přepsána již přítomná data. To zajišťuje, že v případě přechodu mezi zpracováním vstupu či výstupu objekty `ChannelSend` a `CompensationDelay` proběhne plynulý crossfade.

3 UŽIVATELSKÁ PŘÍRUČKA

V této části práce je popsáno fungování aplikace z pohledu uživatele a její ovládání. Aplikace byla otestována na laptopu Dell Inspiron 15 s integrovanou zvukovou kartou a externí zvukovou kartou Focusrite Scarlett 2i4 2nd Gen. Ovládání pomocí dotykové obrazovky bylo otestováno na notebooku Microsoft Surface Pro 2017. Oba počítače používají Windows 10.

3.1 Spuštění

Jelikož velikost aplikace přesahuje maximální velikost příloh, soubory aplikace jsou k dispozici na tomto odkaze. Aplikace se spouští pomocí souboru `AATMuSS.exe`.

Po spuštění aplikace proběhne kontrola, zda adresář `Dokumenty` obsahuje podsložku `AATMuSS Presets`, a také zda se v této složce nachází soubory „továrních“ přednastavení (*def_stereo*, *def_flipped_stereo* a *def_mono*). V případě neexistující složky nebo jednotlivých souborů jsou všechny potřebné soubory vytvořeny.
Následně je načteno nastavení, se kterým byla aplikace naposledy ukončena. Pokud se jedná o první spuštění, případně pokud byl soubor s nastavením smazán, je načteno přednastavení *def_stereo*, které ve výchozím stavu přiřadí první vstup k prvnímu výstupu a druhý vstup ke druhému výstupu. Není zde nastavena inverze signálu ani zpoždění. Jelikož aplikace při spuštění kontroluje pouze existenci základních nastavení, uživatel si je může libovolně přizpůsobit.

3.2 Okno aplikace

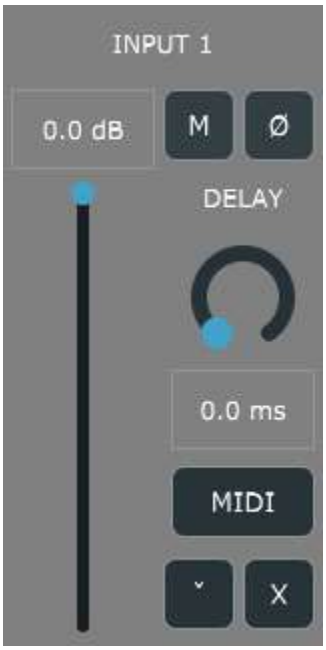
Ve vrchní části aplikace se nachází několik tlačítek (viz obr. 3.1). *Load Preset* a *Save Preset* slouží k načítání, resp. ukládání nastavení (viz dále). Pomocí tlačítka *User Settings* se zobrazuje uživatelské nastavení, zatímco *Device Settings* zobrazí nastavení zvukového zařízení. Tlačítko *Routing Matrix* zobrazí směrovací matici.

Pod těmito tlačítky jsou ve výchozím pohledu umístěny ovládací prvky zpracovávaných zvukových kanálů sdružené do „channel stripů“. Channel stripy jsou seskupeny podle přiřazeného výstupu. Pokud k výstupnímu kanálu není přiřazen žádný vstup, je tento výstupní kanál přeskočen. Pokud uživatel umožní zpracování samotných vstupů či výstupů (*pre-* a *post-processing*), příslušné stripy jsou umístěny v první, resp. poslední řadě. Pokud se všechny stripy nevejdou do okna aplikace, objeví se vpravo a/nebo dole posuvníky, s jejichž pomocí lze zobrazení vertikálně a/nebo horizontálně posouvat.

Obr. 3.1: Okno aplikace

3.3 Channel Strip

V horní části channel stripu je označení kanálu, ke kterému náleží. Pod tímto titulkem jsou pak samotné ovládací prvky.



Obr. 3.2: Channel strip

Na levé straně se nachází posuvný kontrolér zesílení s rozsahem −60 až 0 dB. Hodnotu v decibelech lze rovněž zadat do textového pole.

Na pravé straně je umístěno tlačítko pro inverzi polarity signálu (symbol Ø), které se při aktivaci zbarví do světle tyrkysového odstínu. Také je zde umístěno tlačítko pro ztišení kanálu označené písmenem M (*mute*), které se po aktivaci zbarví do červena. Pokud je kanál ztišený, nijak nezpracovává příchozí signál a má tak velice malý vliv na vytížení procesoru a operační paměti. Parametry daného kanálu lze však i nadále měnit jak pomocí grafických rozhraní, tak i pomocí MIDI ovladače, nové hodnoty budou aplikovány po zrušení ztišení.

Pod těmito tlačítky se nachází posuvný kontrolér ovládající zpoždění – hodnotu v milisekundách lze stejně jako u zesílení zadat do textového pole. Rozsah tohoto kontroléru lze změnit v uživatelském nastavení, výchozí maximální hodnota je 100 ms. Nejmenší krok hodnoty zpoždění je 0,1 ms.

Pod tímto posuvníkem je umístěno tlačítko, které slouží ke zobrazení menu pro přiřazování MIDI zpráv k jednotlivým parametrům zpracování daného kanálu (viz kapitola 3.3.1). Pro skrytí tohoto menu stačí znovu stisknout tlačítko *MIDI*.

Posledními ovládacími prvky jsou pak tlačítka pro skrytí channel stripu (opětovně jej lze zobrazit povolením v menu směrovací matice) a kompletní odstranění kanálu.

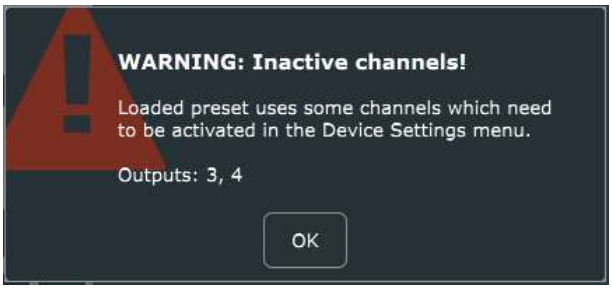
3.3.1 MIDI ovládání

Po kliknutí na tlačítko MIDI se zobrazí jednoduché menu pro přiřazování MIDI zpráv k ovládání parametrů daného kanálu. Tlačítko *L* zahájí proces mapování (rozsvítí se žlutě), následně stačí pohnout vybraným ovládacím prvkem na MIDI kontroléru a dalším kliknutím na *L* se potvrdí přiřazení poslední zprávy k parametru. Tlačítkem *X* je pak možné toto přiřazení odstranit. MIDI ovládání rozlišuje jak mezi kanály, tak i mezi jednotlivými zařízeními, a to na základě identifikátoru. Je tedy možné používat několik totožných zařízení najednou.

Obr. 3.3: Menu přiřazování MIDI zpráv k parametrům

3.4 Ukládání a načítání nastavení

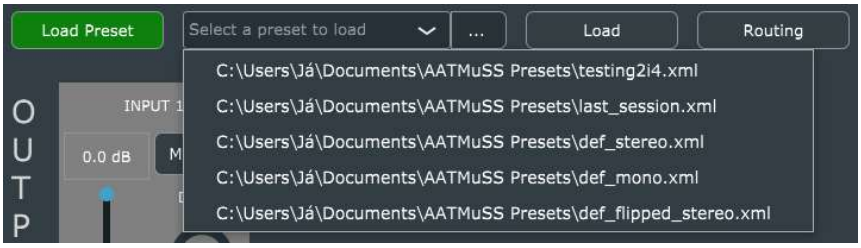
Tlačítko *Load Preset* v horní části okna po kliknutí zobrazí výběr přednastavení (viz obr. 3.4a). Kliknutím na tři tečky vedle výběru se otevře okno pro procházení souborů, které slouží například k načtení presetu uloženého mimo výchozí adresář. Tlačítko *Load* pak slouží k potvrzení výběru. Pokud preset obsahuje nastavení pro kanály, které nejsou aktivní, zobrazí se upozornění se seznamem indexů těchto kanálů. Stačí je následně aktivovat v nastavení zařízení.



Obr. 3.4: Upozornění na neaktivované kanály

Obdobně funguje i ukládání nastavení tlačítkem *Save Preset* (viz obr. 3.4b). Při výběru (a potvrzení) existujícího nastavení je soubor přepsán. Pro uložení nového presetu je třeba zvolit procházení (taktéž symbol tří teček) a zadat název. Potvrzením pomocí tlačítka *Save* se pak soubor uloží.

Načítání i ukládání nastavení je možné zrušit opětovným kliknutím na tlačítko *Load Preset*, resp. *Save Preset*.



(a)

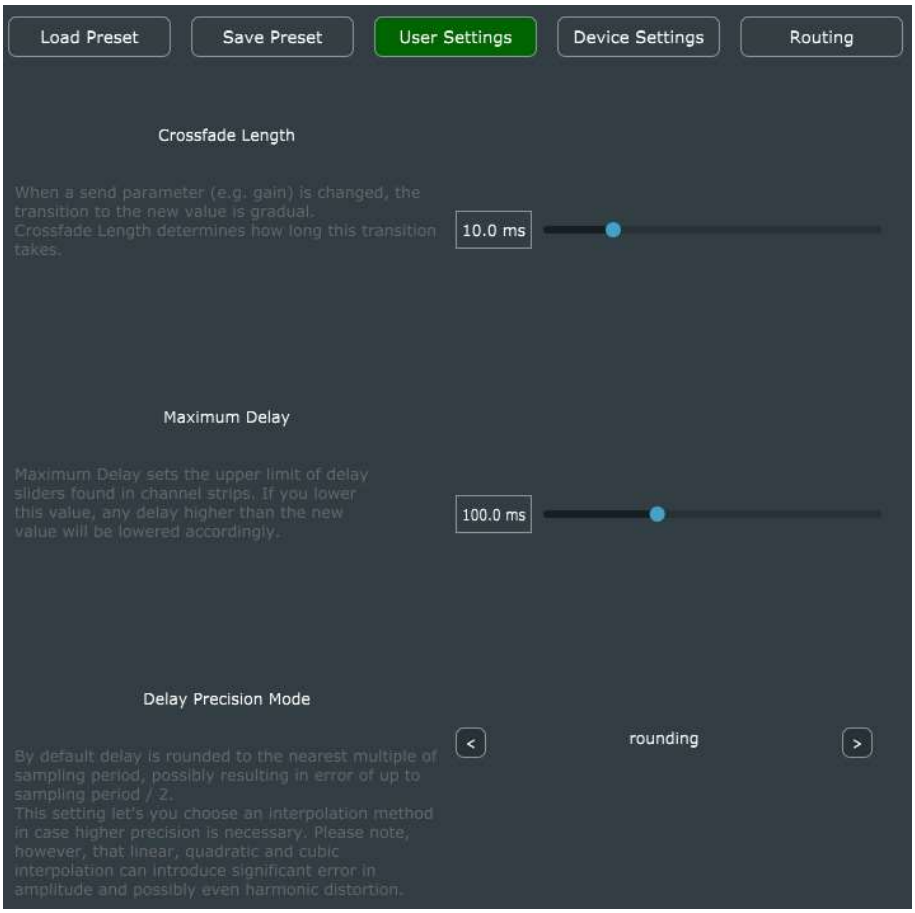
(b)

Obr. 3.5: Načítání (a) a ukládání presetu (b)

Soubory jsou ukládány ve formátu XML, lze je tedy upravovat v textovém editoru. Jelikož však aplikace nekontroluje dodržení formátu XML ani platnost uložených hodnot, tato činnost může způsobit nežádoucí chování aplikace.

3.5 Uživatelské nastavení

Tlačítko *User Settings* skryje channel stripy (případně nastavení zařízení či směrovací matici) a zobrazí uživatelské nastavení. Zde se nachází dva posuvné kontroléry, jejichž funkce je krátce popsána i v aplikaci. Také je zde umístěn výběr módu zpoždění.



Obr. 3.6: Uživatelské nastavení

Při změně kteréhokoliv parametru kanálu, případně jeho přidání či odstranění, se na novou hodnotu přechází postupně, aby bylo zamezeno prudkým skokům v průběhu signálu, které mohou způsobit praskání. Kontrolér *Crossfade Length* nastavuje délku tohoto přechodu v milisekundách. Zde je vhodno poznamenat, že přechod na novou hodnotu může trvat výrazně delší dobu, pokud je nastaveno nenulové zpoždění. Důvod pro tuto prodlevu je taktéž zamezení praskání (blíže popsáno v kapitole 2.2.3).

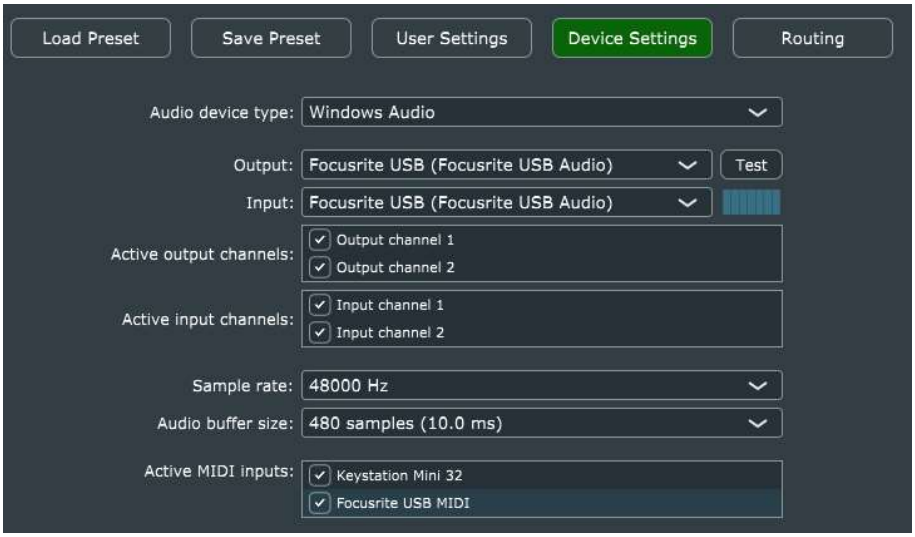
Druhým kontrolérem je *Maximum Delay*, který nastavuje horní mez rozsahu zpoždění v channel stripech. Pokud je nově nastavená hodnota nižší než parametr zpoždění kteréhokoliv kanálu, zpoždění daného kanálu se sníží na novou maximální hodnotu.

Výchozím módem zpoždění je zaokrouhlování požadované hodnoty zpoždění na násobky vzorkovací periody, další možnosti jsou pak lineární, kvadratická a kubická interpolace a nadvzorkování. Nadvzorkování je podporováno pouze pro vzorkovací kmitočty 44,1 kHz, 48 kHz, 88,2 kHz a 96 kHz, při výběru jiného kmitočtu bude aplikace zpracovávat zpoždění s využitím kubické interpolace.

Uživatelské nastavení lze skrýt opětovným kliknutím na *User Settings*, případně kliknutím na *Load Preset*, *Save Preset*, *Device Settings* nebo *Routing Matrix*.

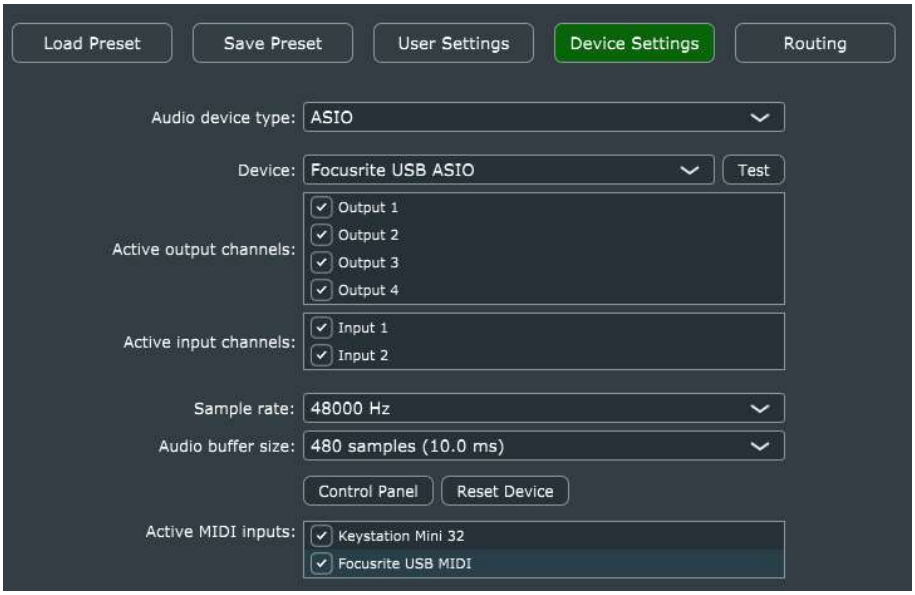
3.6 Nastavení zařízení

Tlačítko *Device Settings* zobrazí nastavení zvukového zařízení.



Obr. 3.7: Nastavení zařízení

Prvním parametrem *Audio device type* se vybírají ovladače zařízení. Další je volba vstupního a výstupního zařízení. Při volbě ASIO ovladačů se místo nich zobrazí parametr *Device*, který určí zařízení pro vstup i výstup, protože ASIO nepodporuje kombinaci různých zařízení. Dále je možné vybrat vstupní a výstupní kanály, jejichž počet je omezen na 32.



Obr. 3.8: Nastavení zařízení – ASIO

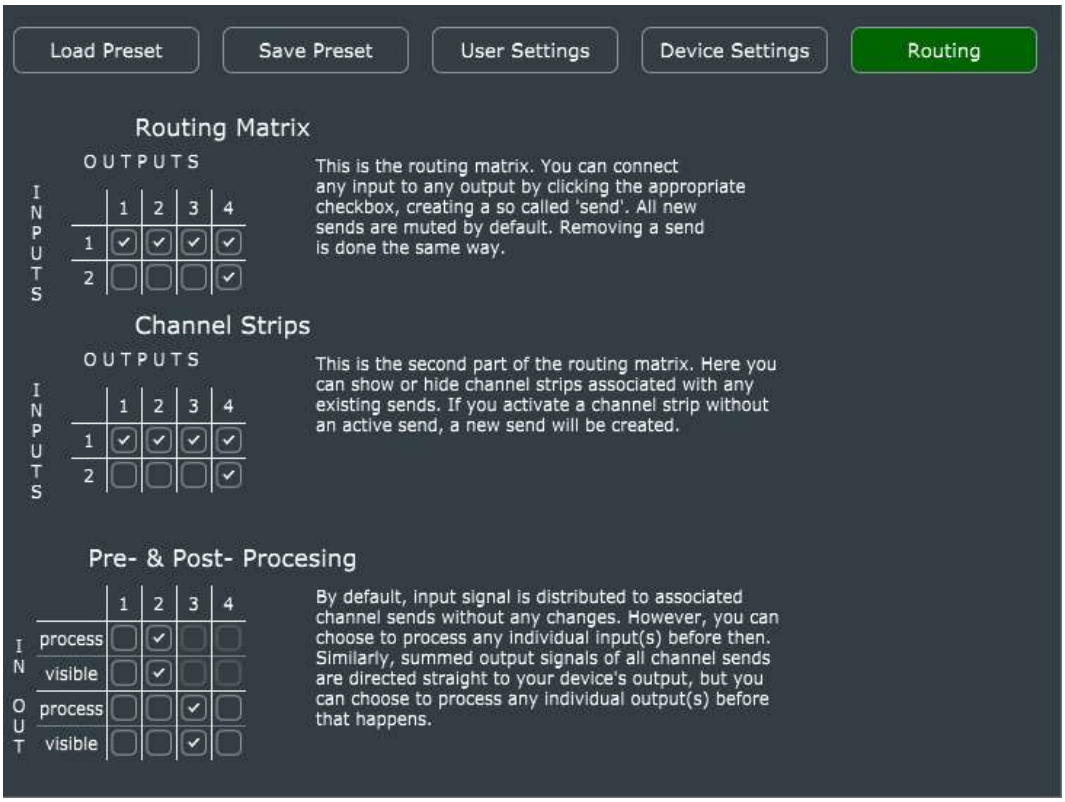
Dále si uživatel může zvolit vzorkovací kmitočet a velikost vyrovnávací paměti. V případě ASIO ovladačů se ještě zobrazí tlačítka pro otevření kontrolního panelu zařízení a resetování zařízení.

Nakonec je možné aktivovat libovolné množství vstupních MIDI zařízení, která lze použít k ovládání vybraných prvků aplikace.

Nastavení zařízení je možné skrýt stejným způsobem jako uživatelské nastavení, tzn. opětovně kliknout na *Device Settings* nebo kliknout na *Load Preset*, *Save Preset*, *User Settings* či *RoutingMatrix*.

3.7 Směrovací matice

Kliknutím na tlačítko Routing Matrix se zobrazí směrovací matice. Ta je rozdělena do tří dílčích matic.



Obr. 3.9: Směrovací matice

První matice reprezentuje směšování signálů z libovolných vstupních kanálů do libovolných výstupních kanálů. Pro každou kombinaci vstupu a výstupu je zobrazeno zaškrtávací pole. Pokud jej uživatel kliknutím aktivuje, je přidán příslušný „send“ s výchozími parametry, tj. zesílení 0 dB, bez inverze signálu a zpoždění a aktivním ztišením, tudíž uživatel může upravit hlasitost signálu před jeho aktivací. Zároveň je ve výchozím pohledu zobrazen příslušný channel strip.

Druhá matice reprezentuje právě zobrazení channel stripů. Pokud uživatel deaktivuje libovolné (aktivní) zaškrtávací pole, je příslušný channel strip skryt, ovšem zpracování signálu probíhá i nadále. Pro opětovné zobrazení skrytého channel stripu stačí aktivovat příslušné zaškrtávací pole. Pokud k aktivovanému channel stripu nenáleží send (tedy stejné pole v první matici není zaškrnuté), je tento send vytvořen s výchozími parametry, viz výše.

Třetí matice reprezentuje kombinaci prvních dvou matic, ovšem namísto směšování vstupů do výstupních kanálů je určena pro zpracování signálu vstupních a výstupních kanálů před, resp. po směšování signálů. Řádky *process* odpovídají funkci první matice, tzn. zapnutí či

vypnutí zpracování daného kanálu. Pokud je zpracování vypnuto, signál prochází beze změny. Řádky *visible* pak slouží k zobrazení či skrytí příslušných channel stripů.

Všechny tři matice automaticky přizpůsobují své rozměry počtu vstupních a výstupních kanálů zvoleného zvukového zařízení, přičemž maximální počet je 32. Pokud je nějaký vstup či výstup neaktivní, příslušná zaškrtávací pole jsou vykreslena jinou barvou a nelze je aktivovat.

Vedle každé matice je zobrazen popis její funkce, při použití tedy není třeba znovu pročítat uživatelský manuál.

4 ZÁVĚR

V první kapitole diplomové práce byly shrnuty teoretické znalosti a principy, které byly dále využity k vypracování praktické části. Byl zde vysvětlen pojem ozvučování a využití více reproduktorových soustav, také byl stručně popsán signálový řetězec číslicového zpracování signálů, základní principy číslicového zpracování zvukového signálu a protokol MIDI. Kapitolu uzavřelo stručné seznámení s programovacím jazykem C++, knihovnou JUCE a ovladači ASIO.

Druhá kapitola se zabývala popisem aplikace z pohledu programátora. Byl zde popsán obsah zdrojových souborů a princip činnosti nejdůležitějších tříd a funkcí.

Ve třetí kapitole aplikace byla popsána z pohledu uživatele. Byla vysvětlena obsluha aplikace pomocí grafických prvků a nastavení zvukového zařízení. Také zde byla stručně popsána funkcionalita vybraných procesů.

Cílem této diplomové práce bylo seznámit se s knihovnou JUCE a vytvořit aplikaci pro úpravu zvukového signálu. Tato aplikace umožňuje v reálném čase provádět jednoduché zpracování signálu, konkrétně zesílení, inverzi signálu a zpoždění v rozsahu 0 až 200 milisekund s krokem 0,1 ms, jehož přesnost může být zajištěna použitím několika interpolačních metod. Také byla implementována podpora ASIO ovladačů, nastavitelné ovládání pomocí jednoho či více MIDI zařízení a směrovací matice pro přiřazování až 32 vstupních a výstupních kanálů. Nakonec bylo otestováno funkční ovládání pomocí dotykové obrazovky. Vytvořením této aplikace bylo zadání diplomové práce částečně splněno.

Požadavek zadání na možnost zpracování parametrickým ekvalizérem nebyl splněn z časových důvodů.

LITERATURA

[1] SCHIMMEL, Jiří. *Elektroakustika*. Brno: VUT v Brně. Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací 2016, 200 s. ISBN 978-80-214-4716-5.

[2] SMÉKAL, Zdeněk. *Analýza signálu a soustav – BASS*. Brno: VUT v Brně. Fakulta elektrotechniky a komunikačních technologií, Ústav komunikací 2012, 252 s. ISBN 978-80-214-4453-9.

[3] SMÉKAL, Zdeněk. *Číslicové zpracování signálů*. Brno: VUT v Brně. Fakulta elektrotechniky a komunikačních technologií, Ústav komunikací 2012, 246 s. ISBN 978-80-214-4639-7.

[4] SCHIMMEL, Jiří. *Studiová a hudební technika*. Brno: VUT v Brně. Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací 2015, 197 s. ISBN 978-80-214-4452-2.

[5] KUO, Sen M. y GAN, Woon-Seng. *Digital Signal Processors: Architectures, Implementations, and Applications*. Hardcover ed. Prentice Hall, 2004. p. 601. ISBN 978-0130352149

[6] KUO, Sen M., LEE, Bob H. y TIAN, Wenshun. *Real-Time Digital Signal Processing: Implementations and Applications*. Hardcover ed. Wiley, 2006. p. 664. ISBN 978-0470014950

[7] LYONS, Richard G. *Understanding Digital Signal Processing*. Kindle Edition ed. Pearson, 2010. p. 966. ISBN 978-0137027415

[8] *ISO/IEC 14882:2020.* ISO – International Organization for Standardization [online]. International Organization for Standardization, 2020 [cit. 2021-12-05]. Dostupné z: https://www.iso.org/standard/79358.html

[9] *6.1.0*. GitHub [online]. [cit. 2021-12-06]. Dostupné z: https://github.com/juce-framework/JUCE/releases/tag/6.1.0

[10] *Documentation*. JUCE [online]. [cit. 2021-12-06]. Dostupné z: https://juce.com/learn/documentation

[11] *Tutorials*. JUCE [online]. [cit. 2021-12-06]. Dostupné z: https://juce.com/learn/tutorials

[12] *3rd-party developers support & SDKs*. Steinberg [online]. [cit. 2021-12-06]. Dostupné z: https://www.steinberg.net/developers/