VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Viktor Černý

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

SOFTWARE FOR AUDIO ADJUSTMENT IN MULTIPLE LOUDSPEAKER SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Viktor Černý

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Petr Sysel, Ph.D.

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Audio inženýrství**

Ústav telekomunikací

Student: Bc. Viktor Černý ***ID:*** 186602

Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Software pro úpravu zvukového signálu pro ozvučování více reproduktorovými soustavami

POKYNY PRO VYPRACOVÁNÍ:

Realizujte software, který bude umožňovat provádět v reálném čase základní úpravy signálu pro ozvučování více reproduktorovými soustavami: funkci MUTE, inverzi signálu, zpoždění v rozsahu alespoň 0,1 – 100 ms s interpolací signálu mezi vzorky a parametrický ekvalizér s volitelným počtem filtrů a to v alespoň 32 kanálech. Aplikace bude také disponovat maticí pro slučování signálů libovolných vstupů a v jejich směrování do jednoho nebo více libovolných výstupů s funkcí soft-fade. Doplňte podporu technologie ASIO, ovládání aplikace pomocí dotykového displeje a rozhraní MIDI a možnost ukládání a vyvolávání presetů.

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: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

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

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

První část diplomové práce se zabývá teorií číslicového zpracování signálů a popisem knihovny JUCE. Jsou zde vysvětleny operace s digitálními audio signály jako například inverze signálu, zpoždění signálu a lineární interpolace vzorků signálu. Je zde také popsána práce s knihovnou JUCE pro vytvoření audio aplikací v programovacím jazyce C++. Další části diplomové práce jsou věnovány popisu implementované aplikace, která umožňuje zmíněné základní úpravy číslicových zvukových signálů aplikovat v reálném čase. V případě vícekanálových zvukových signálů je možné jednotlivé kanály zpracovat nezávisle.

KLÍČOVÁ SLOVA

DSP, knihovna JUCE, C++, audio aplikace, inverze signálu, zpožďovací linka, lineární interpolace, MIDI

ABSTRACT

The first part of this Master's Thesis deals with the theory of digital signal processing and describes the JUCE library. In this part some basic operations are explained with digital audio signals, such as polarity inversion, delay and linear interpolation of signal samples. The creation of new audio applications using the JUCE library in the C++ programming language is explained too. The next parts of this thesis describe the implemented audio application that allows the user to provide the described basic operations with digital audio signals in real time. For multiple channel audio signals the channels can be processed independently.

KEYWORDS

DSP, JUCE library, C++, audio application, polarity inversion, delay line, linear interpolation, MIDI

ČERNÝ, Viktor. *Software pro úpravu zvukového signálu pro ozvučování více reproduktorovými soustavami*. Brno, 2020, 70 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Petr Sysel, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Software pro úpravu zvukového signálu pro ozvučování více reproduktorovými soustavami“ jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Petru Syslovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Cílem diplomové práce je realizace softwaru pro ozvučování, případně pro přizvučování více reproduktorovými soustavami v jazyce C++ využitím knihovny JUCE. Program podle zadání má umožnit uživateli v reálném čase provádět následující operace s číslicovým signálem: ztlumení signálu, tedy funkci *mute*, *inverzi signálu*, *zpoždění signálu* v rozsahu alespoň 0,1 až 100 ms s interpolací mezi vzorky a úpravu signálu *parametrickým ekvalizérem* s volitelným počtem filtrů. Aplikace má také disponovat maticí pro slučování a směrování signálů libovolných vstupů do jednoho nebo více libovolných výstupů s funkcí soft-fade. Další požadavky jsou i podpora technologie ASIO, ovládání aplikace pomocí dotykového displeje a rozhraní MIDI a možnost ukládání a vyvolávání presetů.

První část práce popisuje teoretické znalosti potřebné k vytvoření programu. Je zde vysvětleno slovní spojení DSP, pojmy ozvučování, přizvučování, inverze signálu, zpoždění signálu, lineární interpolace vzorků a filtrování signálu. Operace se signály jsou znázorněny i na grafech. Jsou také popsány postupy, jak se tyto operace provádí s digitálními audio signály. Operace jsou realizovány i v prostředí Matlab a lze je nalézt na přiloženém disku ve složce `Matlab`. Kromě znalostí zpracování číslicových signálů je potřebné znát i programovací jazyk, ve kterém byla aplikace realizována, jmenovitě jazyk C++. Další nezbytná součást programu je knihovna JUCE, která má předem implementované metody a objekty pro usnadnění práce v jazyce C++ při vytváření aplikací, které pracují s audio signály. Je zde také popsáno jak knihovnu použít, kromě toho lze zde najít i stručný popis pro nastavení podpory ASIO driverů pro knihovnu JUCE. V této části práce jsou popsány i základy protokolu MIDI, které byly také potřebné při implementaci programu.

Druhá kapitola diplomové práce se věnuje popisu uživatelského rozhraní programu, které je také znázorněno na obrázcích. Je zde také vysvětleno celkové ovládání programu a všechny funkce programu jsou podrobně popsány. Všechna okna aplikace jsou představena v jednotlivých podkapitolách. Jsou zde vysvětleny například všechny operace se stopami a se signály stop, výběr a nastavování zvukového zařízení, ovládání pomocí MIDI zařízení, matice pro směrování a slučování libovolných vstupů do libovolných výstupů a další.

Poslední část diplomové práce popisuje hlavní funkce objektů aplikace a také zajímavější programátorská řešení problémů, které nastaly při realizace programu. Některé části objektů jsou prezentovány i pomocí zdrojového kódu.

1 Teoretické předpoklady

V diplomové práci jsou nejprve vysvětleny teoretické znalosti, které byly potřebné k vytvoření programu. Tato témata jsou rozdělena do následujících podkapitol: ozvučování 1.1, přizvučování 1.2, číslicové zpracování signálů 1.3, programovací jazyk C++ 1.4, knihovna JUCE 1.5 a protokol MIDI 1.6.

1.1 Ozvučování

O ozvučování nebo anglicky *Public Address* (PA) mluvíme, když cílem procesu je vytvořit maximální úroveň akustického tlaku v pozici posluchače a přitom nejsou využita přímá vlnění od původního zdroje zvuku. Můžeme tedy říci, že zesílený signál úplně překrývá původní signál a přirozené akustické vlastnosti prostoru. Nasnímaný signál je přenášen přímo k posluchači. Ve většině případů je právě proto lokalizace zvuku jednoznačná, tzn. posluchač má dojem, že to co slyší, přichází pouze z reproduktorů nebo reproduktorových soustav. [1]

1.2 Přizvučování

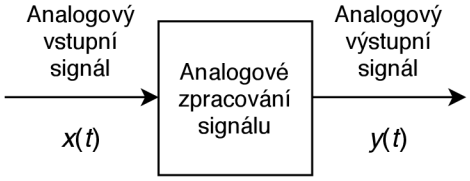
Přizvučování nebo anglickým názvem *Sound Reinforcement* má celkem jiné cíle než ozvučování. Účelem přizvučování je zvýšení úrovně akustického tlaku, přičemž jsou zachovány přímé vlny od zdrojů původních signálů a taktéž původní vjem pozic zdrojů signálů. Při tomto procesu se využívá zpoždění nasnímaných signálů, které slyšíme přes reproduktory, z toho důvodu, aby původní přímé vlny a zesílený signál byly ve fázi. Tímto způsobem lokalizace zdroje zvuku se nemění, ale vzroste subjektivní hlasitost. [1]

1.3 Číslicové zpracování signálů

Hlavní myšlenky a základy této problematiky jsou podrobněji dostupné z literatury [2], ze které budou zde v úvodu podkapitoly probírány podstatné části pro pochopení fungování aplikace. Stručnou historii signálů a signálových soustav i základní operace se signály popisuje první kapitola literatury [3].

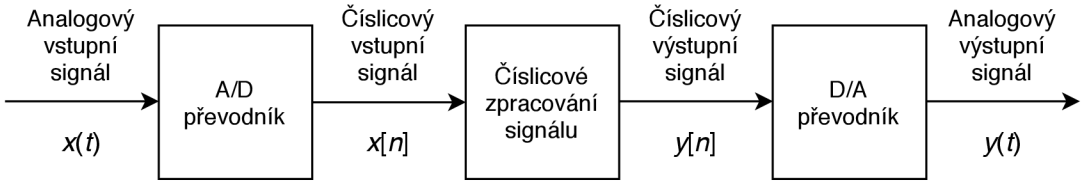
Číslicové zpracování signálu (anglicky *Digital Signal Processing*, nebo zkráceně DSP) jako slovní spojení se dá velice jednoduše vysvětlit. O číslicovém zpracováním signálu mluvíme, když pracujeme se signálem v digitální, neboli číslicové podobě, který byl převeden z analogové domény. Tento signál nemusí být primárně zvukový, i když slovní spojení se většinou používá v odvětvích profesionálního audia.

Na obrázku 1.1 můžeme vidět základní bloky zpracování analogového signálu. Ze vstupního analogového signálu $x(t)$ po zpracování analogovým systémem dostaneme výstupní analogový signál $y(t)$.



Obr. 1.1: Blokové schéma analogového zpracování signálu.

Obrázek 1.2 nám představuje základní bloky číslicového zpracování signálu. Vstupem je stejný analogový signál $x(t)$ a výstupem je taktéž analogový signál $y(t)$, ale jsou zde dva nové nezbytné bloky, které provádí převod signálu z analogové do číslicové podoby a pak i obráceně. Mezi těmito převody lze signál zpracovat číslicovým systémem.



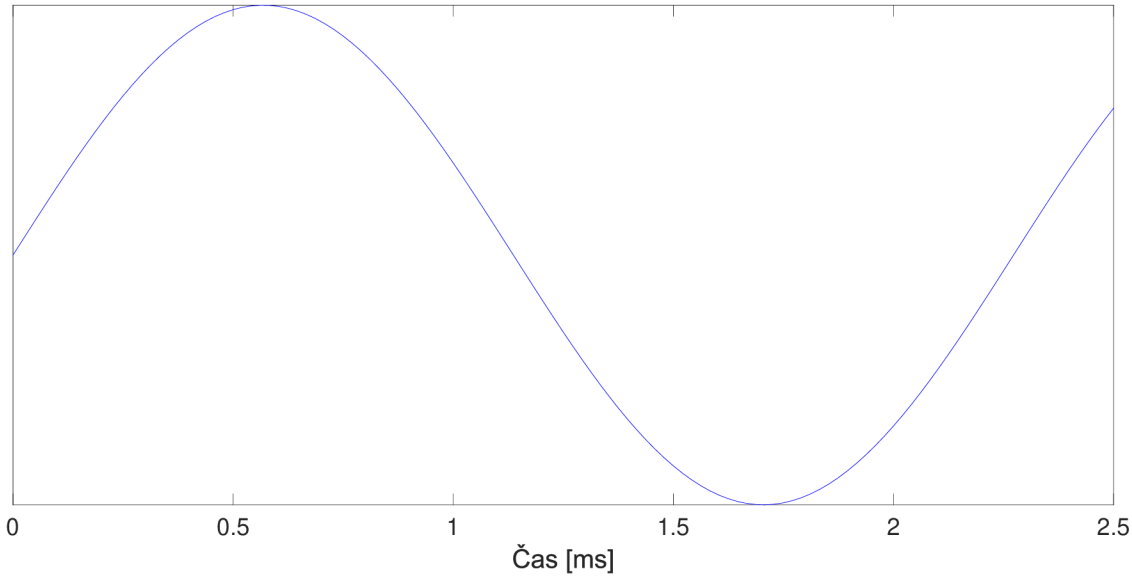
Obr. 1.2: Blokové schéma číslicového zpracování signálu.

Digitální signál tedy získáme převodem pomocí A/D převodníku (analogově digitální převodník), který ze spojitého analogového signálu vytvoří diskrétní digitální signál. Jako ukázku analogového signálu použijeme jednoduchý sinusový signál $x(t)$ o kmitočtu $f = 440\,\mathrm{Hz}$, který je popsán pomocí rovnice (1.1), kde x je samotný signál, f je frekvence a t je čas. Signál je také znázorněn na obrázku 1.3.

$$x(t) = \sin(2\pi f t) \tag{1.1}$$

A/D převodník převádí analogový signál s periodou T, kterou nazýváme jako *vzorkovací perioda*, na číselné hodnoty. To znamená, že po uplynutí času T dostaneme *vzorek* (*sample*), který reprezentuje kus analogového signálu. Vzorkovací periodu dostaneme pomocí *vzorkovacího kmitočtu* (*sampling rate*) f_{vz} podle rovnice (1.2).

$$T = \frac{1}{f_{\mathrm{vz}}} \tag{1.2}$$

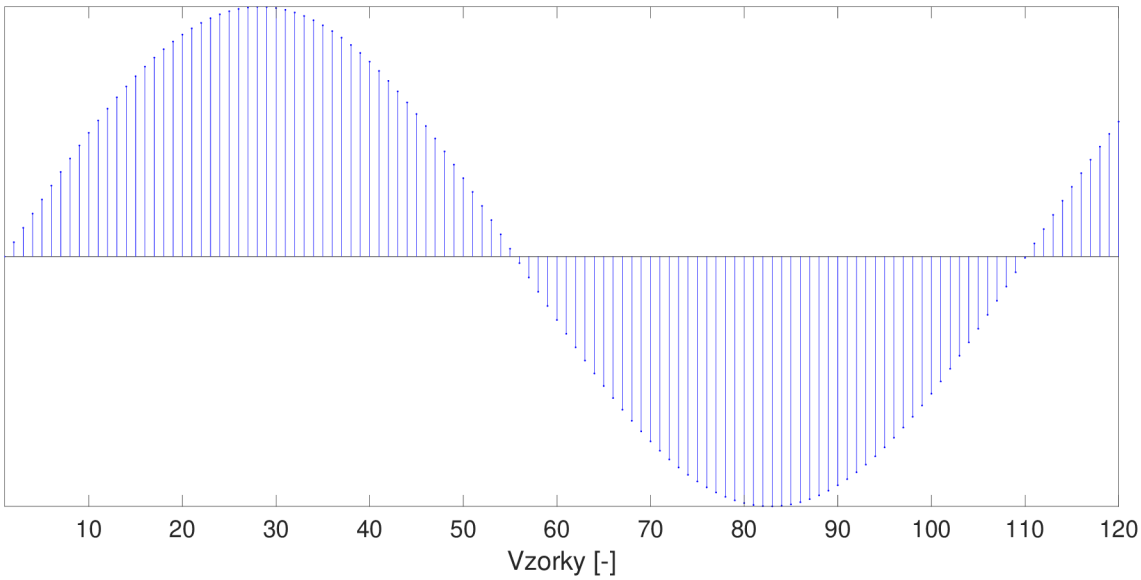


Obr. 1.3: Spojitý sinusový signál.

Po převodu můžeme popsat nový signál jako:

$$x[n] = x(nT), \tag{1.3}$$

kde $x[n]$ je digitální obraz analogového signálu $x(t)$. Je to posloupnost čísel, neboli posloupnost vzorků, kde n označuje n-tý vzorek posloupnosti. Signál $x[n]$ je znázorněn na obrázku 1.4.



Obr. 1.4: Diskrétni sinusový signál.

Pro snadné odlišování diskrétních a analogových signálů jsou používána různá značení. Analogové signály jsou většinou označovány kulatými závorkami a diskrétní signály hranatými závorkami.

Musíme ale také vysvětlit pojmy *vzorkování* (*sampling*) a *kvantování*, aby bylo jasné, jak A/D převodník pracuje. A/D převodník nejprve převádí spojitou funkci na diskrétní posloupnost čísel, to znamená, že spojitý analogový signál převádí na diskrétní signál. Tato operace se nazývá *vzorkování*, jelikož po uplynutí doby vzorkovací periody dostaneme jeden vzorek signálu. Tyto vzorky ale zatím mají nekonečně mnoho hodnot, které nelze ukládat ve dvojkové číselné soustavě. *Kvantování* tedy je proces, když tyto hodnoty jsou zaokrouhleny na konečný počet hodnot, které nazýváme jako *kvantovací hladiny*. Počet kvantovacích hladin je ovlivněn *bitovou hloubkou* (*bit depth*). Čím větší je bitová hloubka, tím více kvantovacích hladin je dostupné pro kvantování a tím pádem lze signál kvantovat s větším rozlišením. Po převodu diskrétního signálu na signál kvantovaný již můžeme signál nazvat jako číslicový (digitální) signál. Kvantování spojitého signálu na číslicový vede vždy ke ztrátě informace.

Sice na obrázku 1.2 nejsou znázorněny, ale aby blokové schéma bylo úplně správně, měli bychom přidat dvě nezbytné operace. Jedna z operací předchází převod analogového signálu na digitální a druhá operace patří až na konec blokového schematu. Jde o filtraci signálu před A/D převodem a o filtraci po D/A převodu. Oba filtry mají charakter dolní propusti. Filtr před A/D převodníkem se nazývá *anti-aliasingovým filtrem*, kterého mezní kmitočet určuje maximální frekvenci zpracovaného signálu. Druhý filtr, po rekonstrukci D/A převodníkem, vyhlazuje výstupný signál.

Důležitá věta pro vzorkování je *Nyquistův teorém*, která se také nazývá jako *vzorkovací poučka*. Tato věta je popsána rovnicí (1.4).

$$f_{\mathrm{vz}} > 2f_{\mathrm{max}}, \tag{1.4}$$

kde f_{vz} je vzorkovací kmitočet a f_{max} je nejvyšší kmitočet, který signál obsahuje. Pravě kvůli této poučce je maximální frekvence vstupního signálu omezena před A/D převodem. Kdyby signál nebyl filtrován anti-aliasingovým filtrem, došlo by k *aliasingu*. Když dojde k aliasingu už nelze obnovit původní analogový signál. Tuto problematiku popisuje podrobněji podkapitola 5.2 literatury [3].

1.3.1 Inverze signálu

Při ozvučování více reproduktory nebo více reproduktorovými soustavami může nastat situace, kde vyzařované akustické signály se navzájem částečně nebo zcela vyruší.

Jeden z důvodů, proč tento jev nastane, může být, že vyzařované signály přichází do pozice posluchače s opačnou polaritou. Když situaci zjednodušíme a začneme zkoumat obyčejný sinusový signál, snadno nalezneme řešení pro tento problém.

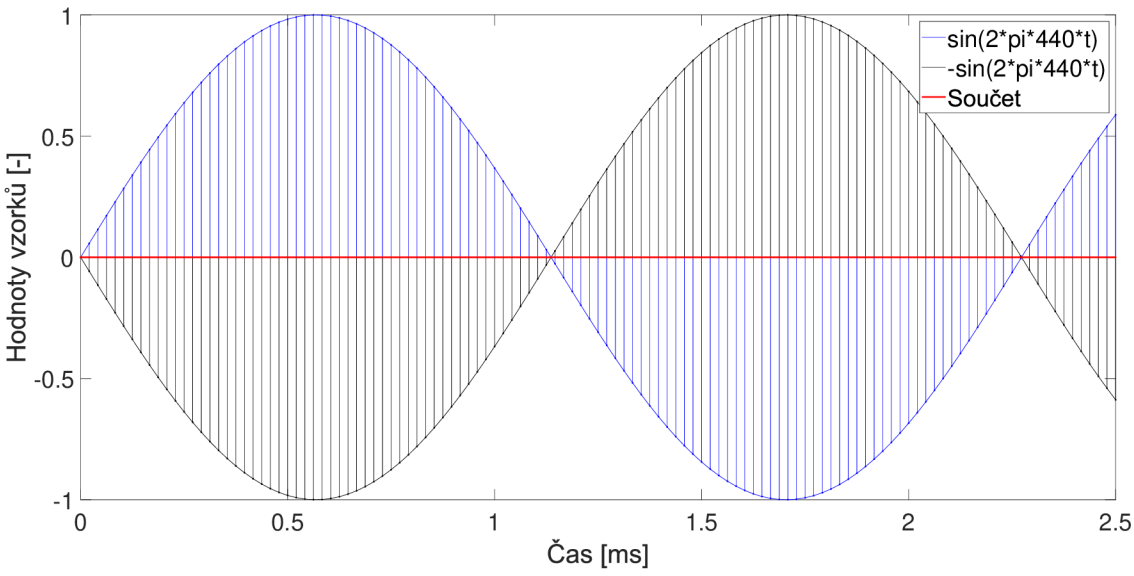
Je daný sinusový signál s frekvencí $f = 440\,\text{Hz}$, který již byl popsán rovnicí (1.1). Signál je znázorněn na obrázku 1.5 modrou barvou.

Když zmíněný signál $x(t)$ je vynásoben -1, dostaneme *inverzi* tohoto signálu. To znamená, že všechny hodnoty původního signálu budou mít v novém signálu $y_1(t)$ opačnou hodnotu. Tato jednoduchá operace je využita k převrácení polarity. Invertovaný signál je popsán rovnicí (1.5) a znázorněn na obrázku 1.5 černou barvou.

$$y_1(t) = -\sin(2\pi f t) \tag{1.5}$$

Z rovnic (1.1) a (1.5) je zřejmé, že po sečtení těchto dvou signálů, buď akusticky nebo digitálně, se signály navzájem vyruší. Tento děj je znázorněn i na obrázku 1.5 červenou barvou a je popsán pomocí rovnice (1.6).

$$z_1(t) = x(t) + y_1(t) = \sin(2\pi f t) + [-\sin(2\pi f t)] = 0 \tag{1.6}$$



Obr. 1.5: Součet signálů $x(t)$ a $y_1(t)$.

Problém opačné polarity tedy lze vyřešit vynásobením digitálního signálu konstantou -1.

Zajímavé ale je, že lidské ucho slyší signály $x(t)$ a $y_1(t)$ stejně. Když jsou tyto signály porovnávány, rozdíly mezi nimi nejsou znatelné. Právě proto se dá tento jev poznat jen tehdy, když signály hrají současně.

1.3.2 Zpoždění signálu

Další důvod, proč se signály mohou částečně vyrušit, je zapříčiněn velikostí vzdálenosti mezi zdroji zvuků. Tato vzdálenost způsobuje fázové posuny mezi některými spektrálními složkami signálů. Jev se vyskytuje díky relativně pomalé rychlosti šíření zvukových vlnění ve vzduchu.

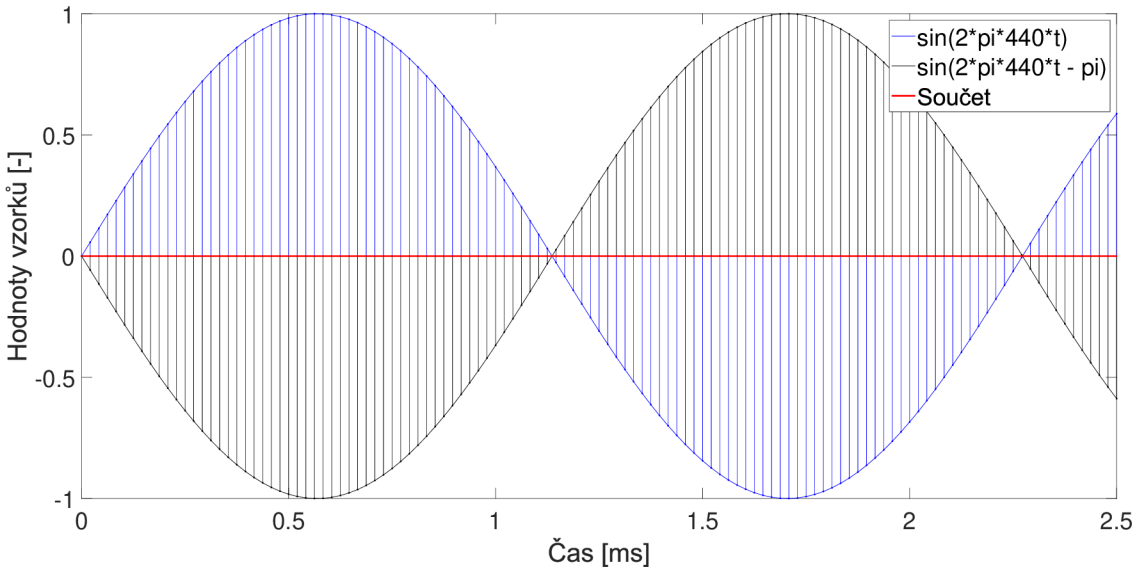
Situaci zjednodušíme, jev budeme zkoumat u sinusových signálů, stejně jako při inverzi. Jako původní signál použijeme stejný sinusový signál o kmitočtu $f = 440\,\mathrm{Hz}$, který byl popsán rovnicí (1.1) a je znázorněn na obr. 1.6 modrou barvou.

Když je tento signál $x(t)$ zpožděn o 180 stupňů (v radiánech o π), mluvíme o signálu v protifázi. Tento nový signál $y_2(t)$ lze popsat pomocí rovnice (1.7). Signál je také znázorněn na obrázku 1.6 černou barvou.

$$y_2(t) = \sin(2\pi ft - \pi) \tag{1.7}$$

Je známo, že po sečtení původního signálu se signálem v protifázi se signály vzájemně vyruší. Tento jev je znázorněn na obrázku 1.6 červenou barvou a popsán rovnicí (1.8).

$$z_2(t) = x(t) + y_2(t) = \sin(2\pi ft) + \sin(2\pi ft - \pi) = 0 \tag{1.8}$$



Obr. 1.6: Součet signálů $x(t)$ a $y_2(t)$.

Řešením tohoto problému je zpoždění původního signálu $x(t)$. Když původní signál zpozdíme o 180 stupňů (v radiánech o π), pak dostaneme nový signál, který už bude ve fázi se signálem $y_2(t)$.

U číslicových signálů se toho může docílit buď pomocí dočasného *bufferu* nebo *kruhové paměti*, kam budeme ukládat všechny vzorky signálu, které chceme zpozdit. Velikost paměti se dá vypočítat velice snadno, potřebujeme k tomu několik jednoduchých rovnic.

Vzdálenost pro další výpočty v reálné situaci známe: je to vzdálenost mezi zdroji zvuků, kterou chceme kompenzovat, aby signály byly ve fázi. U našeho teoretického příkladu ale tuto vzdálenost neznáme, právě proto musíme nejprve vypočítat vlnovou délku signálu pomocí rovnice (1.9).

$$\lambda = \frac{c}{f}, \tag{1.9}$$

kde λ je vlnová délka signálu, c je rychlost šíření zvuku ve vzduchu (použijeme hodnotu 343 m/s) a f je frekvence signálu. Když tuto rovnici vyřešíme pro $f = 440\,\text{Hz}$, dostaneme vlnovou délku našeho sinusového signálu. My ale potřebujeme kompenzovat pouze půlku vlnové délky, aby byly signály ve fázi: $s = \lambda/2 = 0{,}3898\,\text{m}$.

Vzdálenost jsme dostali pomocí řešení rovnice pro výpočet vlnové délky. Pro další výpočty potřebujeme čas, za který náš signál urazí vypočtenou vzdálenost. Pro výpočet použijeme rovnici (1.10).

$$t = \frac{s}{c}, \tag{1.10}$$

kde s je vzdálenost pro kompenzaci a c je rychlost šíření zvuku ve vzduchu. Když tuto rovnici vyřešíme podle našeho příkladu, dostaneme čas $t = 1{,}136\,\text{ms}$. To znamená, že aby signály $x(t)$ a $y_2(t)$ byly ve fázi, musíme původní signál zpozdit o 1,136 ms.

Vypočítali jsme tedy časový posun mezi signály $x(t)$ a $y_2(t)$, následuje výpočet velikosti bufferu pro zpoždění. Velikost bufferu je vlastně počet vzorků v číslicovém signálu za vypočtený čas. Když známe vzorkovací frekvenci f_{vz}, pak známe i počet vzorků N za 1 sekundu a můžeme vypočítat velikost bufferu M pomocí poměrů:

$$\frac{1}{N} = \frac{t}{M}, \tag{1.11}$$

kde N je počet vzorků signálu za 1 sekundu, t je čas v sekundách a M je velikost bufferu, kterou dostaneme úplně přesně pomocí rovnice (1.12).

$$M = tN, \tag{1.12}$$

Když tuto rovnici vyřešíme pro $f_{\text{vz}} = 48\,\text{kHz}$, takže pro $N = 48000$ vzorků, dostaneme hodnotu $M = 54{,}528$ vzorků. To znamená, že dočasný buffer pro vypočtený časový posun by měl mít délku 55 vzorků.

S tím jsme ale ještě nevyřešili problém protifáze. Nejprve musíme do bufferu ukládat vzorky signálu, když se buffer zaplní, můžeme začít číst vzorky z bufferu od nejstaršího po nejnovější. Každý přečtený vzorek smažeme z bufferu a přidáme nový vzorek signálu. Tímto způsobem jsme posunuli celý digitální signál o daný počet vzorků, to znamená, že po číslicově analogovém převodu bude analogový signál posunut v čase.

Již bylo zmíněno, že zpoždění lze realizovat i pomocí kruhové paměti. Kruhová paměť funguje obdobně jako buffer, má stejnou délku a také se do ní ukládají vzorky signálu. Rozdílem ale je, že pokud z bufferu smažeme vzorky a pak přidáváme nové, v kruhové paměti pouze přepisujeme staré hodnoty vzorků na nové. Když iterátor, pomocí něhož iterujeme v paměti, se dostane na konec, vynuluje se a začne číst a přepisovat hodnoty zase od začátku. Kvůli této skutečnosti je tento typ paměti nazývána jako kruhová. Kruhová paměť se využívá velice často v real time aplikacích.

Těmito jednoduchými způsoby můžeme vytvořit digitální *zpožďovací linku*. Vyskytuje se zde ale jeden problém: když použijeme jen celé vzorky abychom docílili zpoždění signálu, rozlišení zpožďovací linky bude závislé na vzorkovacím kmitočtu. Taková zpožďovací linka se nazývá jako *sample delay*. Nejkratší zpoždění dostaneme, když buffer bude mít délku jednoho vzorku. Čas tohoto zpoždění můžeme vypočítat pomocí rovnice (1.13).

$$t_{\text{sample}} = \frac{1}{f_{\text{vz}}}, \tag{1.13}$$

kde f_{vz} je vzorkovací kmitočet a t_{sample} je délka jednoho vzorku. Když do rovnice dosadíme $f_{\text{vz}} = 48\,\text{kHz}$ pak dostaneme, že délka jednoho vzorku je zhruba $t_{\text{sample}} = 0,0208\,\text{ms}$. Neznamená to jen to, že nejmenší zpoždění, které můžeme realizovat je $0,0208\,\text{ms}$, ale i to, že reálné hodnoty zpoždění budou vždy násobky tohoto čísla. Reálnou délku zpožďovací linky můžeme vypočítat pomocí rovnice (1.14).

$$t_{\text{delay}} = M t_{\text{sample}}, \tag{1.14}$$

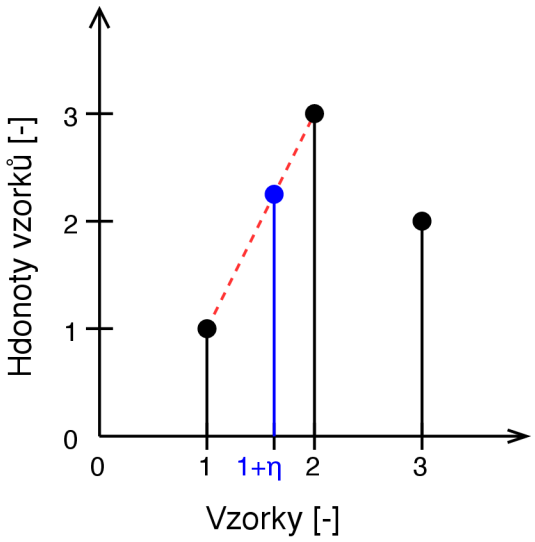
kde t_{delay} je reálná délka zpožďovací linky, M je počet vzorků zpožďovací linky a t_{sample} je délka jednoho vzorku. V našem případě pro $M = 55$ a $t_{\text{sample}} = 0,0208\,\text{ms}$ dostaneme reálnou délku zpožďovací linky $t_{\text{delay}} = 1,144\,\text{ms}$. Můžeme vidět, že výše vypočítaná požadovaná délka zpoždění t a reálná délka zpožďovací linky t_{delay} se neshodují. Zpoždění realizované pomocí posunu vzorků je tedy přesné jen pro násobky časových délek vzorků.

Pro lepší rozlišení je třeba vzorky ve zpožďovací lince interpolovat. S touto problematiku se zabývá velice podrobně literatura [4] v kapitole „Delay/Signal Interpolation“. Je zde popsáno několik metod pro interpolaci signálu, ze kterých podrobněji budeme probírat lineární interpolaci v podkapitole 1.3.3.

1.3.3 Lineární interpolace

Z mnoha metod interpolací byla vybrána lineární interpolace z důvodu rychlosti a výpočetní nenáročnosti. Po přečtení kapitoly „Delay/Signal Interpolation“ literatury [4] je zřejmé, že pro aplikace pracující v reálném čase je tento způsob interpolace výpočetně nejvýhodnější.

Když při zpoždění signálu použijeme interpolaci vzorků, vypočtenou délku paměti zaokrouhlujeme směrem nahoru na celé číslo až po uložení zlomkové části tohoto čísla. Právě proto, typy zpoždění, které nepracují jen s celými vzorky signálu, jsou nazývány jako *zlomková zpoždění* (*fractional delay*). Zlomkovou část zpoždění budeme označovat řeckým písmenem η.



Obr. 1.7: Znázornění lineární interpolace.

Na obrázku 1.7 je znázorněna lineární interpolace mezi vzorky signálu. Když známe dva vzorky signálu a potřebujeme hodnotu signálu mezi těmito vzorky, můžeme tuto hodnotu lineárně interpolovat. Grafickým řešením je spojení známých hodnot přímkou a podle zlomkové části odečtení nové hodnoty pomocí přímky. Numericky tuto hodnotu lze vypočítat pomocí rovnice (1.15).

$$y[n+\eta] = (1-\eta) \cdot y[n] + \eta \cdot y[n+1], \tag{1.15}$$

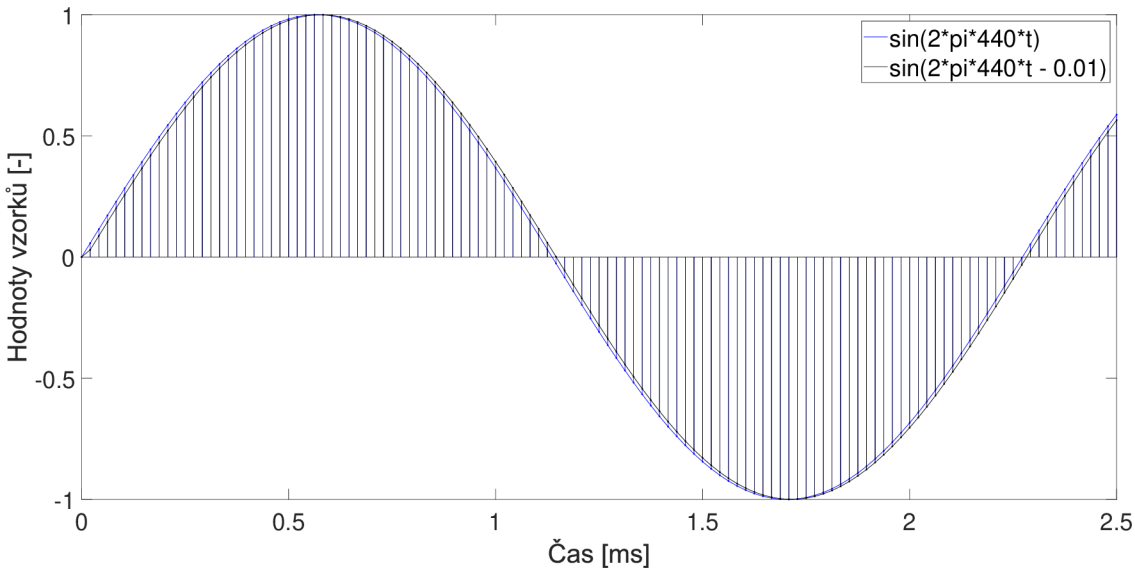
kde $y[n+\eta]$ je interpolovaná hodnota mezi vzorky $y[n]$ a $y[n+1]$. Z této rovnice vyplývá, že když $\eta = 0$, dostaneme hodnotu vzorku $y[n]$ a když $\eta = 1$, dostaneme hodnotu vzorku $y[n+1]$.

Když posloupnost čísel na obrázku 1.7 označíme jako posloupnost $x[n]$, kde vzorky jsou značeny písmenem n, pak pro $\eta = 0,6$ můžeme vypočítat interpolovanou hodnotu posloupnosti $x[n+\eta]$ podle rovnice (1.15). Když všechna známá čísla dosadíme do rovnice, dostaneme výsledek $x[n+\eta] = 2,2$. Sice na obrázku není hodnota interpolovaného vzorku vyznačena, ale můžeme vidět, že hodnoty by měly být zhruba stejné.

Výpočet lineární interpolace můžeme ještě zjednodušit. V rovnici (1.15) násobíme dvakrát, zvlášť každý vzorek. Aby výpočetní náročnost interpolace ještě klesla, můžeme tuto rovnici zjednodušit tak, aby zbylo pouze jedno násobení. Vypočítat interpolovanou hodnotu s jedním násobením lze pomocí rovnice (1.16).

$$y[n+\eta] = y[n] + \eta \cdot (y[n+1] - y[n]) \tag{1.16}$$

Zpoždění signálu $x(t)$, který byl popsán rovnicí (1.1), o $0,01$ ms pomocí lineární interpolace je znázorněno na grafu 1.8, kde původní signál je vykreslen modrou barvou a zpožděný signál černou barvou.



Obr. 1.8: Zpoždění signálu $x(t)$ o $0,01$ ms pomocí lineární interpolace.

Pro využití lineární interpolaci ve zpožďovací lince můžeme vzorky nejprve interpolovat a následně je ukládat do paměti, nebo můžeme do paměti ukládat původní vzorky signálu a pak na výstupu počítat interpolaci vzorků.

1.3.4 Filtry

Kmitočtovou analýzou signálů a číslicovými filtry se podrobně zabývá čtvrtá kapitola literatury [2]. Kromě čtvrté kapitoly zmíněné literatury byly pro základy této podkapitoly použity i znalosti z druhé kapitoly. Literatura popisuje problematiku velice rozsáhle.

Číslicové filtry jsou diskrétní systémy, přesněji *lineární časově invariantní* diskrétní systémy, zkráceně LTI (*Linear Time-Invariant*). U LTI systémů pro vstupní signál jednotkového impulsu dostaneme na výstupu systému takzvanou *impulsní charakteristiku (odezvu)* $h[n]$. Pro systémy LTI platí, že po diskrétní konvoluci vstupního signálu $x[n]$ a impulsní charakteristiky $h[n]$ získáme odezvu systému na vstupní signál, tedy výstupní signál $y[n]$. Tato vlastnost systému je popsána rovnicí (1.17).

$$y[n] = h[n] * x[n] \tag{1.17}$$

Podle délky impulsní charakteristiky se dá tyto systémy rozdělit do dvou kategorií: FIR a IIR systémy.

FIR (*Finite Impulse Response*) systémy, jak název uvádí, jsou systémy s *konečnou* impulsní charakteristikou. Impulsní charakteristika takového systému má konečný počet hodnot. Při implementaci systému FIR je potřebná velká paměť pro uložení všech hodnot impulsní charakteristiky, což způsobí zpoždění při výpočtu výstupního signálu.

IIR (*Infinite Impulse Response*) systémy naopak jsou systémy s *nekonečnou* impulsní charakteristikou. Impulsní charakteristika IIR systému má nekonečný počet hodnot. Implementace takového systému vyžaduje menší paměť pro uložení koeficientů filtru, tím pádem i zpoždění při výpočtu výstupního signálu je menší. Kvůli těmto vlastnostem jsou využívány v aplikaci filtry typu IIR.

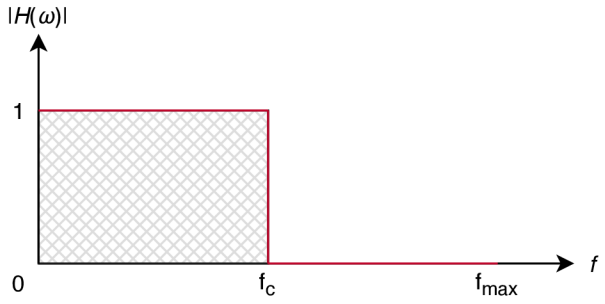
Po Fourierově transformaci impulsní charakteristiky $h[n]$ LTI diskrétního systému dostaneme *kmitočtovou charakteristiku* $H(\mathrm{e}^{\mathrm{j}\omega})$ systému. Kmitočtová charakteristika se skládá ze dvou členů: z *modulové kmitočtové charakteristiky* $M(\omega)$ a z *fázové kmitočtové charakteristiky* $\Theta(\omega)$. *Spektrum* signálu dostaneme taktéž pomocí Fourierovy transformace.

Jak již bylo zmíněno, číslicové filtry jsou lineární časově invariantní systémy. Tyto systémy ovlivňují spektrum $X(\mathrm{e}^{\mathrm{j}\omega})$ vstupního signál $x[n]$ vlastní kmitočtovou charakteristikou $H(\mathrm{e}^{\mathrm{j}\omega})$. Spektrum $Y(\mathrm{e}^{\mathrm{j}\omega})$ výstupního signálu $y[n]$ lze vypočítat pomocí rovnice (1.18).

$$Y(\mathrm{e}^{\mathrm{j}\omega}) = H(\mathrm{e}^{\mathrm{j}\omega}) \cdot X(\mathrm{e}^{\mathrm{j}\omega}) \tag{1.18}$$

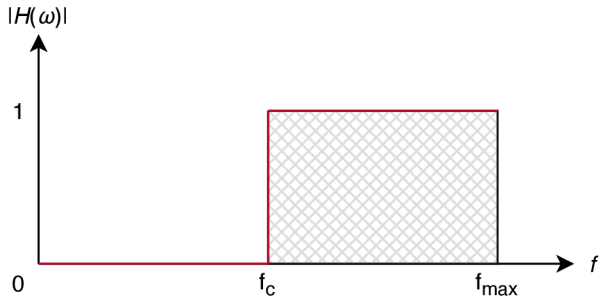
Kmitočtová charakteristika LTI systému tedy upravuje spektrum vstupního signálu. Podle toho, kterou část spektra vstupního signálu filtr propouští, a kterou potlačuje, jsou i jednotlivé filtry pojmenované.

Dolní propust (DP), anglicky *low-pass filter* (LPF), je typ filtru, který propouští dolní část spektra vstupního signálu a horní část spektra potlačuje. Jednostranná modulová kmitočtová charakteristika ideální dolní propusti je znázorněna na obrázku 1.9.



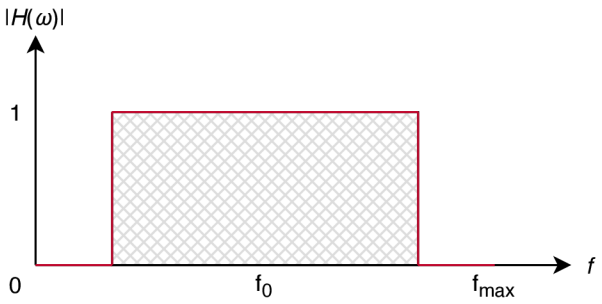
Obr. 1.9: Jednostranná modulová kmitočtová charakteristika ideální dolní propusti.

Horní propust (HP), anglicky *high-pass filter* (HPF), propouští horní část spektra vstupního signálu a dolní část spektra potlačuje. Jednostranná modulová kmitočtová charakteristika ideální horní propusti je znázorněna na obrázku 1.10.



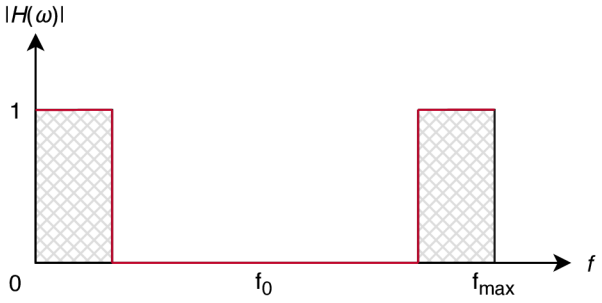
Obr. 1.10: Jednostranná modulová kmitočtová charakteristika ideální horní propusti.

Pásmová propust (PP), anglicky *band-pass filter* (BPF), propouští jen vybrané kmitočtové pásmo spektra vstupního signálu a ostatní části spektra potlačuje. Jednostranná modulová kmitočtová charakteristika ideální pásmové propusti je znázorněna na obrázku 1.11.



Obr. 1.11: Jednostranná modulová kmitočtová charakteristika ideální pásmové propusti.

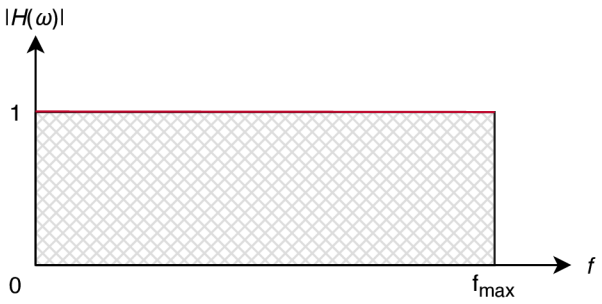
Pásmová zádrž (PZ), anglicky *band-stop filter* (BSF), funguje opačným způsobem jako pásmová propust, potlačuje pouze vybrané kmitočtové pásmo spektra vstupního signálu a ostatní části spektra propouští. Jednostranná kmitočtová charakteristika ideální pásmové zádrže je znázorněna na obrázku 1.11.



Obr. 1.12: Jednostranná modulová kmitočtová charakteristika ideální pásmové zádrže.

Zajímavý případ filtrů je *fázovací článek*, který má konstantní modulovou kmitočtovou charakteristiku a používá se hlavně pro úpravu fázové kmitočtové charakteristiky ostatních typů filtrů. Jednostranná kmitočtová charakteristika ideálního fázovacího článku je znázorněna na obrázku 1.13.

Pro pochopení fungování aplikace jsou základní znalosti, popsané v této podkapitole, dostačující. Pro podrobný a rozsáhlejší popis problematiky lze využít již zmíněnou literaturu [2], hlavně kapitolu čtvrtou, která popisuje kmitočtovou analýzu signálů a kmitočtové filtry.



Obr. 1.13: Jednostranná modulová kmitočtová charakteristika ideálního fázovacího článku.

1.4 Jazyk C++

Programovací jazyk C++ byl vyvinut v roce 1985 autorem Bjarnem Stroustrupem. C++ vychází z programovacího jazyka C, odtud pochází i název. Symbol „++“ je operátor inkrementace v jazyce C a je symbolem evoluce programovacího jazyka. Jazyk C je ve většině případů kompatibilní s jazykem C++.

Programovací jazyk C++ podporuje několik programovacích stylů. Jeden z nejčastěji používaných stylů, když se mluví o programování v tomto jazyku, je objektově orientované programování. Nelze ale říci, že jazyk je čistě objektový.

Jazyk je standardizován a v současnosti se používá C++17. Tento programovací jazyk patří mezi nejrozšířenější programovací jazyky na světě. Více informací o programovacím jazyce lze čerpat z literatury [5], ze kterého byly již nejpodstatnější informace shrnuty v této podkapitole.

Základy a použití programovacího jazyka i pokročilejší koncepty jsou podrobně popsány v literatuře [6]. Kniha byla napsána autorem tohoto programovacího jazyka a v současnosti již čtvrté vydání knihy popisuje i moderní techniky využití jazyka C++.

Webová stránka dostupná z literatury [7] nabízí také velice podrobný popis programovacího jazyka, základní informace o jazyku, stručnou historii a kromě toho jsou zde dostupné i návody pro snazší porozumění struktury C++.

1.5 Knihovna JUCE

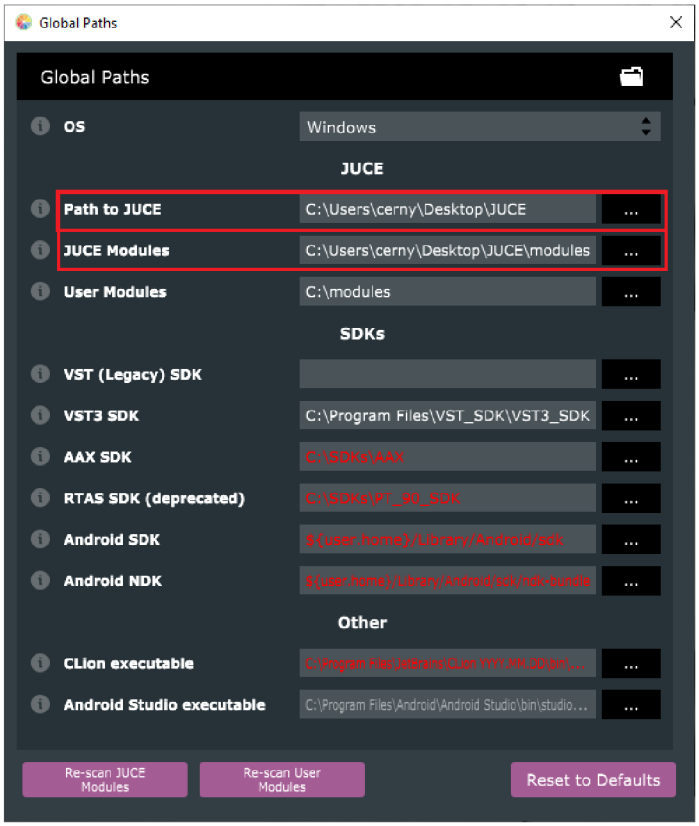
JUCE (Jules' Utility Class Extensions) je knihovna napsaná v programovacím jazyku C++, která byla vytvořena autorem Julesem Strorerem v roce 2004 a v současnosti patří k firmě ROLI. JUCE se používá pro vývoj aplikací nejen pro počítače, ale i pro mobilní telefony a tablety. Hlavním cílem knihovny je, aby zdrojové kódy

aplikací implementované knihovnou se daly sestavit stejně na všech platformách. Právě proto knihovna podporuje různá vývojová prostředí a překladače. [8]

Pro použití knihovny JUCE je třeba znát alespoň základy programovacího jazyka C++. Při práci můžeme také využít dokumentaci knihovny, která je napsaná velice srozumitelně a je dostupná z [9]. Na oficiálních stránkách lze nalézt i tutoriály pro řešení daných problémů pomocí knihovny. Tyto tutoriály jsou dostupné z [10].

1.5.1 Instalace JUCE

Knihovna JUCE je dostupná na oficiálních stránkách [11]. JUCE disponuje i vlastním vývojovým prostředím, které dostalo název *Projucer*. Po stažení knihovny musíme vytvořit účet, abychom ji mohli používat. Registrovat se je možné po spuštění aplikace. Po úspěšné registraci musíme ještě nastavit přístupové cesty pro potřebné adresáře knihovny. Nastavení těchto cest najdeme, když z nabídky menu vybereme `File` a pak `Global paths...`. Pro naši aplikaci je teď důležitá hlavně cesta adresáře JUCE a cesta pro JUCE moduly (složka `modules` v adresáři JUCE), které jsou červenou barvou zvýrazněny na obrázku 1.14. Po nastavení těchto cest je knihovna připravena k použití.



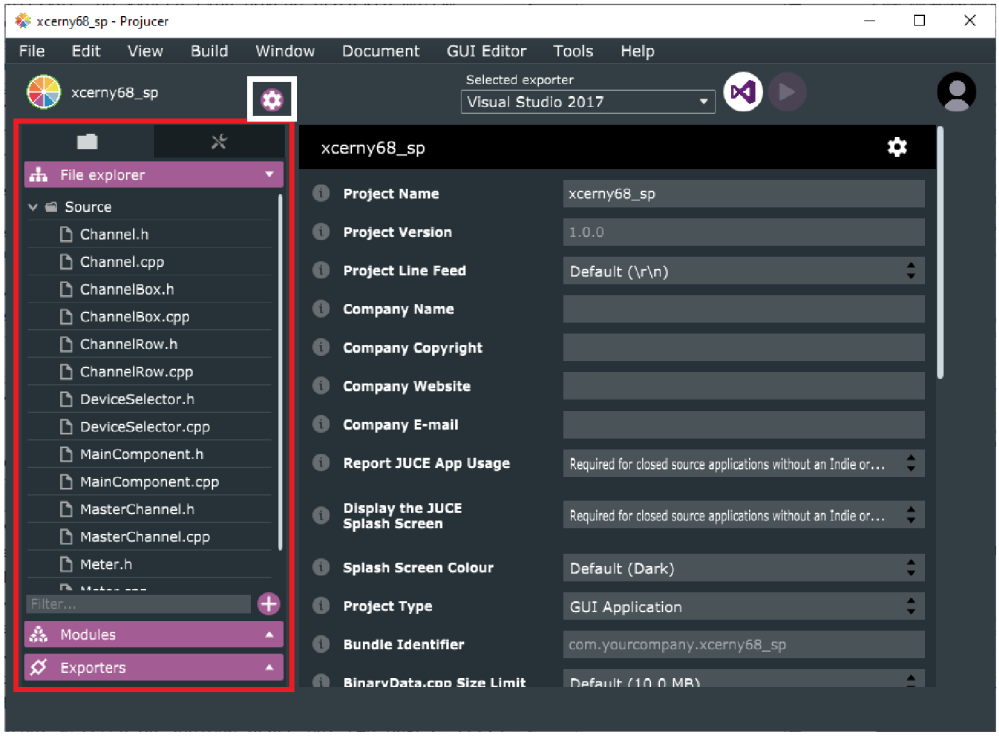
Obr. 1.14: Příklad nastavení cest knihovny.

V adresáři knihovny JUCE můžeme nalézt i aplikaci `DemoRunner`, která prezentuje jednodušší ukázky knihovny se zdrojovými kódy. Kromě toho ve složce `Extras` jsou další aplikace, které jsou implementovány pomocí knihovny JUCE. Zajímavé je, že aplikace *Projucer* byla také realizována pomocí knihovny. Můžeme zde najít zdrojové soubory všech aplikací.

1.5.2 Práce s JUCE

Pro vytvoření nového projektu, který využívá knihovnu, můžeme použít vlastní vývojové prostředí knihovny, zmíněné v podkapitole 1.5.1, názvem *Projucer*. Aplikace je velice intuitivní, po výběru typu nového projektu můžeme projekt pojmenovat, vybrat adresář, do kterého bude uložen a nastavit vývojové prostředí, které bude použito pro sestavování projektu. Na výběr jsou např.: Visual Studio, Code::Blocks, Xcode a další. Kromě toho má *Projucer* při vytváření nových projektů funkci pro automatickou generaci souborů s několika možnostmi. Po zmáčknutí tlačítka `Create...` aplikace vytvoří nový projekt.

Po vytvoření nového projektu se dostaneme do vývojového prostředí aplikace. Vedle názvu projektu je ikonka pro úpravu všech nastavení projektu (znázorněna na obr. 1.15 bílou barvou), můžeme zde projekt přejmenovat, změnit vzhled, přidat poznámky a mnoho dalšího.



Obr. 1.15: Projekt v aplikaci *Projucer*.

Menu na levé straně aplikace (znázorněno na obr. 1.15 červenou barvou) je rozděleno na tři důležité části: `File Explorer`, `Modules` a `Exporters`.

V části `File explorer` jsou všechny zdrojové soubory projektu, které můžeme rovnou v aplikaci upravovat. Můžeme zde také přidat nové soubory do projektu. Zajímavou funkcí je přidávání GUI (Graphical User Interface – grafické uživatelské rozhraní) komponentů do projektu, pro které vývojové prostředí disponuje velice snadno použitelným editorem.

Druhá část menu (`Modules`) je navržen pro práci s moduly knihovny. Tyto moduly jsou popsány v už zmíněné literatuře [9]. Můžeme zde přidat nové nebo odebrat nepoužívané moduly z projektu. Změnit cestu pro dané moduly je také možné, nebo je zkopírovat do adresáře projektu.

Poslední část menu (`Exporters`) slouží pro nastavování překladačů projektu. Knihovna podporuje operační systémy Windows, Linux, MacOSX, iOS a také Android. Můžeme zde přidat nebo odebrat překladače a upravovat všechna nastavení pro sestavování projektu.

V aplikaci *Projucer* se dá pracovat se zdrojovými kódy projektu a následně ho sestavit, ale doporučuji použít IDE (Integrated Development Environment – vývojové prostředí), které je navrženo pro práci v programovacím jazyce C++. Knihovna podporuje práci např. ve Visual Studio, Xcode, Code::Blocks a další. Velice příjemnou funkcí aplikace *Projucer* je vytvoření projektu ve vybraném vývojovém prostředí, které lze otevřít přímo z aplikace. Tato funkce nám ušetří spoustu času při práci s projekty.

Pro pochopení základů a fungování knihovny JUCE doporučuji literaturu [12], která podrobně popisuje základní a nejdůležitější části knihovny. Kniha sice nepracuje s nejnovější verzí knihovny, ale i tak je velice užitečná pomůcka.

1.5.3 Podpora ASIO

Knihovna JUCE sice má implementovanou podporu ASIO ovladačů (Audio Stream Input/Output), ale ve výchozím nastavení knihovny je tato funkce vypnutá. Je to tak kvůli licenčním podmínkám firmy Steinberg Media Technologies GmbH, která tento protokol navrhla. Abychom mohli používat ASIO drivery v knihovně, musíme nejprve stáhnout ASIO SDK (Software Development Kit) z oficiálních stránek firmy Steinberg [13]. Po stažení rozbalíme archiv a uložíme soubory `asio.h`, `asiosys.h` a `iasiodrv.h` ze složky `common` do složky `juce_audio_devices`, kterou najdeme v adresáři `modules` knihovny JUCE. Ve složce `juce_audio_devices` můžeme najít soubory `juce_audio_devices.h` a `juce_audio_devices.cpp`, ve kterých je ještě třeba nastavit dvě věci pro úplnou podporu ASIO.

Začneme se souborem `juce_audio_devices.h`, kde nejprve najdeme definici makra `JUCE_ASIO` a pak změníme předem definovanou hodnotu 0 na 1. Tímto krokem jsme aktivovali globální proměnnou `JUCE_ASIO`. Po úpravě by zdrojový kód měl vypadat následovně:

Výpis 1.1: Aktivace `JUCE_ASIO` v souboru `juce_audio_devices.h`

```
86 #ifndef JUCE_ASIO
87  #define JUCE_ASIO 1
88 #endif
```

Podporu ASIO jsme již aktivovali, musíme ale ještě definovat cestu k souborům z ASIO SDK. Po otevření souboru `juce_audio_devices.cpp` najdeme část `JUCE_ASIO` a definujeme cestu pro soubor `iasiodrv.h`. Část zdrojového kódu souboru po úpravě je následovné:

Výpis 1.2: Nastavení cesty v souboru `juce_audio_devices.cpp`

```
118   #include "iasiodrv.h"
119 #endif
```

Jak můžeme vidět, nastavení podpory ASIO knihovny JUCE není vůbec složité. Po těchto jednoduchých krocích naše aplikace budou moci pracovat s nainstalovanými ASIO ovladači.

1.6 MIDI

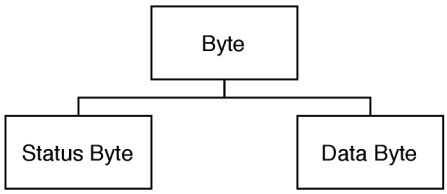
Tato podkapitola popisuje základy protokolu MIDI podle oficiálních specifikací, které jsou dostupné z literatury [14].

Protokol MIDI (Musical Instrument Digital Interface) byl vynalezen pro propojování hardwaru a softwaru, který měl umožnit snadnou výměnu informací například mezi hudebními nástroji, sekvencery, počítači, ovladači pro osvětlování a další. Informace může být kupříkladu hudební nota, změna programu, ovládání výrazu a i mnoho jiné. Protokol byl původně vymyšlen pro využití u živých vystoupeních, ale následující vývoj protokolu měl dopad i na práci v nahrávacích studiích, v audio a video tvorbě a také v kompozičních aplikacích.

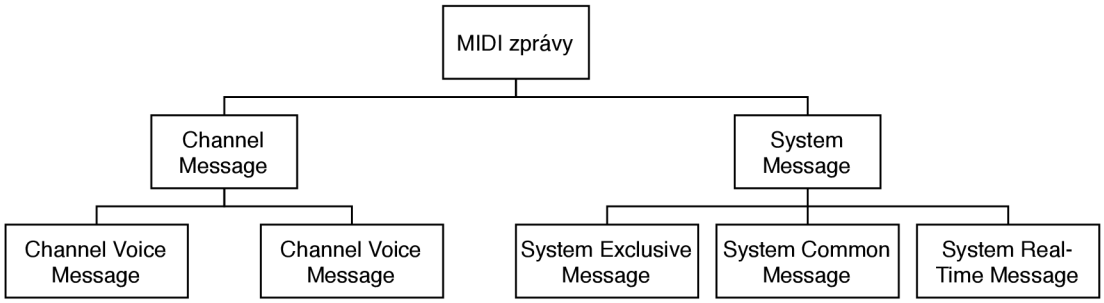
Hardwarové rozhraní MIDI funguje s rychlostí $31,25\,\text{kbit/s}$ a je asynchronní, právě proto první bit je *start bit* (česky rozběhový prvek), který má hodnotu logické nuly a poslední bit je *stop bit* (česky závěrný prvek), který má hodnotu logické jednotky. Mezi tyto bity jsou přenášena MIDI data pomocí osmi bitů. Jako konektor protokolu MIDI je většinou použit konektor typu DIN 5 pin.

MIDI komunikace se uskuteční pomocí MIDI zpráv, které se skládají z více bytů. První byte je *Status byte* a následuje ho jeden nebo dva *Data byty*. Výjimkou jsou *Real-Time* a *Exclusive* zprávy (struktura znázorněna na obrázku 1.19). Typy MIDI bytů jsou znázorněny na obrázku 1.16 a struktury MIDI zpráv jsou znázorněny na obrázku 1.18.

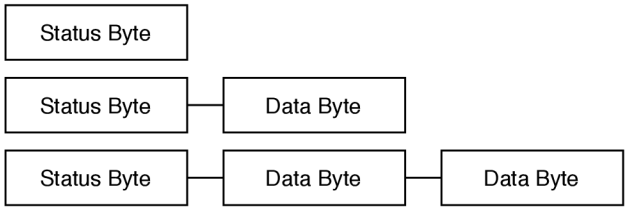
Nástroje, které mají MIDI rozhraní, většinou obsahují přijímač i vysílač zpráv, není ale nutné, aby daný nástroj měl obě součástky. Některé nástroje MIDI zprávy jen vysílají a některé je jen přijímají. Přijímač po přijmutí MIDI zprávy provede MIDI příkazy. MIDI zprávy jsou vysílány v šestnácti kanálech. Nejvíce používané zprávy jsou: *Channel Voice*, *Channel Mode*, *System Common*, *System Real-Time* a *System Exclusive*. Typy MIDI zpráv jsou znázorněny na obrázku 1.17.



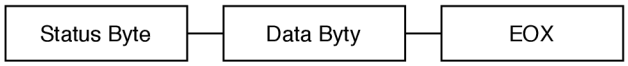
Obr. 1.16: Typy MIDI bytů.



Obr. 1.17: Typy MIDI zpráv.



Obr. 1.18: Struktury MIDI zpráv.



Obr. 1.19: Struktura MIDI zprávy System Exclusive.

1.6.1 MIDI zprávy

MIDI zprávy jsou rozdělené do dvou hlavních kategorií: *Channel* a *System*.

Channel messages

Channel zprávy používají čtyři bity ve Status bytu pro uložení MIDI kanálu a další čtyři bity pro definici zprávy. Právě proto je tento typ zpráv určen pro daný MIDI kanál, který je kódován ve zprávě. Nástroje mohou přijímat MIDI zprávy i pro více kanálů. Kanál, ve kterém nástroj přijímá hlavní instrukce jako například číslo programu a režim, které jsou právě použité, se nazývá jako *Basic Channel*. Nástroj ale může přijímat i data k provedení na více kanálech, tyto kanály se nazývají jako *Voice Channels*.

Channel zprávy jsou rozděleny na dva typy zpráv: *Voice* a *Mode*.

Voice zprávy se využívají pro ovládání zvuku nástroje a jsou posílány přes Voice Channel.

Mode zprávy definují režim nástroje, tedy způsob, jak má nástroj reagovat na Voice zprávy. Tyto zprávy jsou posílány přes Basic Channel.

Pro účely aplikace budou využívány Voice zprávy, jmenovitě zpráva *Control Change*, který je využit pro posílání zpráv po změně kontroleru (např. potenciometr, přepínač a další). Tento typ zprávy obsahuje číslo ovladače a novou hodnotu po změně. Jednoduchá zpráva Control Change může mít hodnotu od 0 po 127. Zprávu se dá využít i jako přepínač, kde hodnota kontroleru 0 znamená vypnutý a hodnota kontroleru 127 znamená zapnutý stav.

System Messages

Systémové zprávy jsou kódovány bez čísla kanálu. Systémové zprávy mohou být *Common*, *Real-Time* a *Exclusive*.

Common systémové zprávy jsou přijímány všemi přijímači v systému bez ohledu na číslo kanálu.

Real-Time systémové zprávy jsou používány pro synchronizaci a jsou určeny pro všechny přijímače v systému, které tyto zprávy dokáží zpracovat. Real-Time zprávy obsahují jen Status byte, jsou posílány bez Data bytu.

Exclusive systémové zprávy (SysEx) mohou obsahovat libovolný počet Data bytů a mohou být ukončeny dalším Status bytem (kromě Real-Time zpráv) nebo EOX-em (*End of Exclusive*). EOX by měl být vždy na konci System Exclusive zprávy.

Pro pochopení fungování aplikace jsou vědomosti z této podkapitoly o protokolu MIDI dostačující. Podrobný a velice rozsáhlý rozpis všech parametrů protokolu MIDI je dostupné z již zmíněné literatury [14], která byla použita jako základ této podkapitoly.

2 Práce s aplikací

V této kapitole diplomové práce je podrobně popsáno fungování celé aplikace. Grafické uživatelské rozhraní programu je také znázorněno na obrázcích. Aplikace byla implementována pomocí knihovny JUCE (verze 5.4.7) ve vývojovém prostředí Visual Studio 2019 (verze 16.5.3).

Logika fungování a některé zajímavější řešení ve zdrojovém kódu programu jsou prezentovány v kapitole 3.

Program byl pojmenován podle oficiálního anglického názvu práce jako „Software for Audio Adjustment“, zkráceně „SoFAA“. Aby program působil co nejvíce profesionálně, celé grafické uživatelské rozhraní bylo velice pečlivě promyšleno, bylo navrženo logo pro aplikaci a byl také vytvořen instalační balíček pro usnadnění manipulaci s programem.

2.1 Instalace

Instalační balíček byl vytvořen pomocí doplňku *Microsoft Visual Studio Installer Projects* pro vývojové prostředí Visual Studio 2019, který je dostupný ze stránek literatury [15]. Tento doplněk umožňuje vytvářet projekt ve vývojovém prostředí Visual Studio, kde můžeme nastavovat veškeré parametry instalačního balíčku. Po sestavení projektu získáme soubor `.msi`, pomocí něhož můžeme program nainstalovat. Tento projekt lze najít i na přiloženém CD ve složce `SoFAA_Setup`. Výstup projektu, `SoFAA_Setup.msi`, je také přiložen na disku.

Při instalaci programu je možné vybrat složku, kam má balíček program nainstalovat. Výchozí cesta pro instalaci je nastavená tak, aby balíček vytvořil novou složku `SoFAA` v adresáři `Program Files`.

Balíček také nastavuje aplikaci jako výchozí program pro otevírání souborů s příponou `.sofaa`.

Po instalaci aplikace ukládá veškerá nastavení do složky `SoFAA`, která se nachází ve složce `AppData/Roaming` daného uživatele.

Instalační balíček také umožňuje jednoduché odinstalování aplikace buď pomocí stejného `.msi` souboru, nebo v části „Aplikace a funkce“ nastavení systému Windows. Balíček soubory nastavení aplikace ze složky `AppData/Roaming` ale neodstraňuje. Bylo třeba napsat další pomocnou konzolovou aplikaci pro tuto úlohu, kterou balíček volá při odinstalování. Projekt tohoto jednoduchého programu je také na přiloženém disku v adresáři `SoFAA_Uninstall`.

Po úspěšné instalaci je také vytvořen odkaz na pracovní ploše pro snadné otevírání aplikace.

2.2 Uvítací okno

Po spuštění aplikace se otevře uvítací okno, které můžeme vidět na obrázku 2.1

Obr. 2.1: Uvítací okno programu.

Uvítací okno je velice jednoduché, je zde logo programu, oficiální název a tři tlačítka: `New project...`, `Open project...` a `Quit`.

Okno lze přemísťovat jednoduchým kliknutím a následujícím tahem s myší. Nové místo okna je pak uloženo v nastaveních aplikace v souboru `Windows.settings` a při dalším otevření jsou souřadnice okna z téhož souboru načteny. Tato nastavení je možné resetovat v aplikaci. Velikost tohoto uvítacího okna nelze změnit.

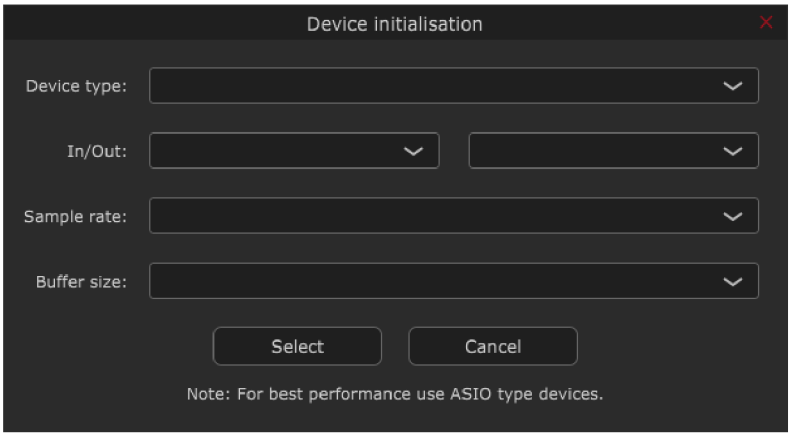
Pomocí tlačítka `New project...` můžeme vytvořit nový projekt. Po kliknutí na tlačítko se zobrazí nastavení zvukového zařízení pro nový projekt. Okno pro výběr zvukového zařízení je podrobněji popsáno v podkapitole 2.3. Tato nastavení pak lze v aplikaci změnit.

Otevřít již existující projekty aplikace lze dvěma způsoby. Pro otevírání projektů můžeme použít tlačítko `Open project...`, které po kliknutí otevře okno pro výběr souboru typu `.sofaa`. Další způsob, jak otevřít projekt v programu, je ještě rychlejší: soubor s příponou `.sofaa` můžeme jednoduše přetáhnout nad uvítací okno a pustit. Po otevření projektu se dostaneme do hlavního okna aplikace, které je popsáno v podkapitole 2.4.

Tlačítko `Quit` má velice jednoduchou úlohu, ukončí program. Po kliknutí na tlačítko se zobrazí dialogové okno a po potvrzení, aplikace se ukončí.

2.3 Výběr zvukového zařízení

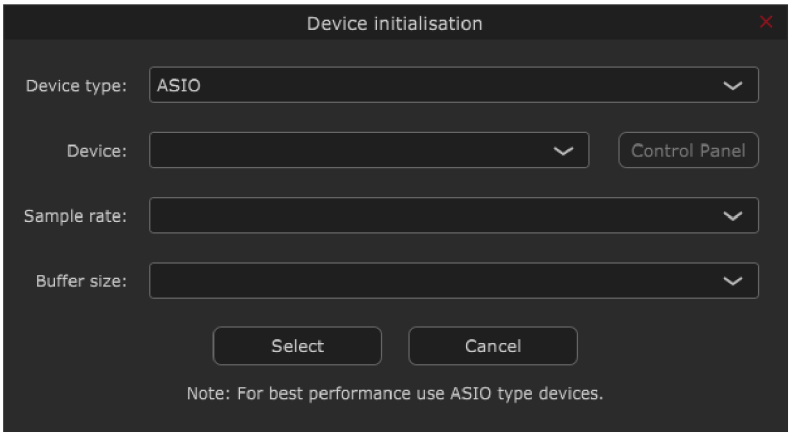
Okno pro výběr zvukového zařízení můžeme vidět na obrázku 2.2.



Obr. 2.2: Okno pro výběr zvukového zařízení.

Fungování této části aplikace je jednoduché. Nejprve vybereme jeden z dostupných typů zvukových ovladačů. Po výběru jsou aktualizována dostupná zařízení, která můžeme kombinovat pro vstupy a výstupy aplikace. Po výběru těchto zařízení je automaticky vybrána hodnota vzorkovacího kmitočtu a velikosti vyrovnávací paměti. Tyto hodnoty lze ještě před výběrem zařízení změnit na dostupné hodnoty.

U typu ovladačů ASIO funguje objekt trochu jinak, neboť ASIO nepodporuje kombinaci více zařízení. Když vybereme typ ASIO, v okně na místě `In/Out:` bude napsáno `Device:`, jeden z rozbalovacích seznamů bude odstraněn a přidá se tlačítko `Control Panel` pro otevření ovládacího panelu zařízení. Všechny tyto změny jsou viditelné na obrázku 2.3.

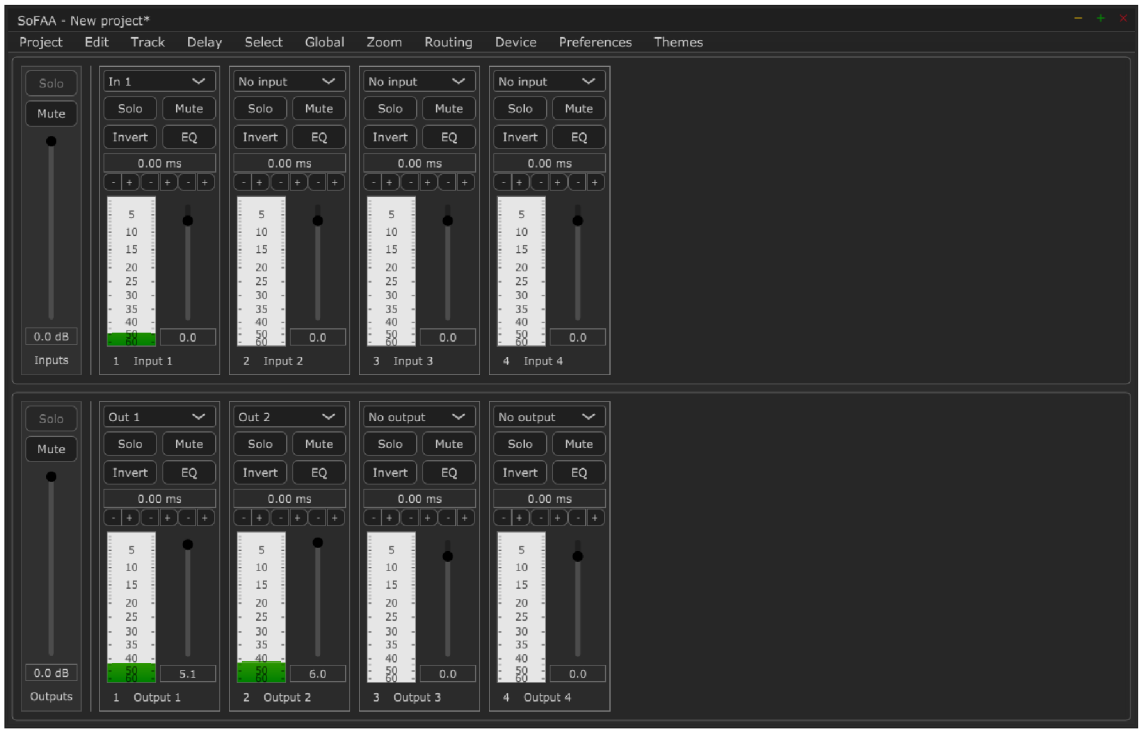


Obr. 2.3: Okno pro výběr zvukového zařízení typu ASIO.

Naposledy použitá nastavení zvukového zařízení jsou uložena ve složce nastavení aplikace v souboru `Audio.settings` a při vytváření nových projektů jsou tato nastavení automaticky načtena pro urychlení práce.

2.4 Hlavní okno

Po výběru zvukového zařízení aplikace otevře hlavní okno programu, které můžeme vidět na obrázku 2.4.



Obr. 2.4: Hlavní okno aplikace.

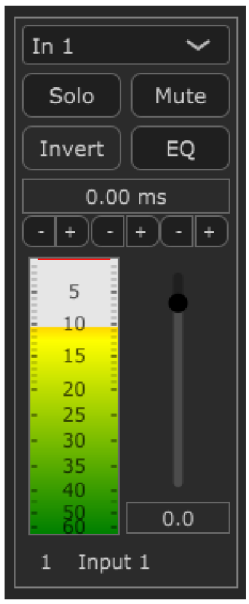
Hlavní okno aplikace je rozděleno na tři části. Je zde menu, kde jsou podle klíčových slov uspořádány všechny funkce aplikace. Kromě toho jsou zde dvě skupiny stop, jedna skupina pro vstupy a druhá pro výstupy. Maximální počet stop v hlavním okně aplikace je omezen na počet aktivních vstupů/výstupů zvukového zařízení. Všechny funkce z menu (2.4.3), všechny ovládací prvky stop (2.4.1) a všechny ovládací prvky hlavních stop (2.4.2) budou podrobněji popsány.

V záhlaví hlavního okna můžeme vidět název otevřeného projektu. Neuložené změny označuje symbol „*“ po názvu projektu v záhlaví.

Hlavní okno aplikace lze přemísťovat a lze také změnit jeho velikost. Změny jsou ukládány v souboru `Windows.settings` a lze je resetovat v aplikaci.

2.4.1 Stopa

Výstřižek jedné stopy z hlavního okna můžeme vidět na obrázku 2.5. Ovládací prvky stopy budou popisovány z hora dolů.



Obr. 2.5: Stopa v hlavním okně.

První ovládací prvek stopy je rozbalovací seznam pro výběr vstupu nebo výstupu zvukového zařízení pro danou stopu. Jeden vstup/výstup je možné přiřadit jen jedné stopě, pro ostatní stopy tento prvek v seznamu již nebude přístupný.

Dalším prvkem je tlačítko `Solo`. Po kliknutí na toto tlačítko, je tlačítko překresleno na červenou barvu, čímž je znázorněno, že stopa je v režimu *solo*. Kliknutím na tlačítko `Solo` u jiné stopy můžeme přidat více stop do režimu. Při držení klávesy CTRL na klávesnici a kliknutím na tlačítko `Solo` je přidána do režimu *solo* jen stopa, kde jsme tlačítko zmáčkli, ostatní stopy jsou ztlumeny. Pomocí klávesy SHIFT na klávesnici a kliknutím na tlačítko `Solo` můžeme přepnout všechny vybrané stopy do režimu *solo*.

Dalším ovládacím prvkem je tlačítko `Mute`. Režim *mute* funguje podobně jako režim *solo*. Po kliknutí na tlačítko, je stopa ztlumena a tlačítko překresleno na zelenou barvu, čímž je znázorněno, že stopa je v režimu *mute*. Do režimu můžeme přidat další stopy kliknutím na tlačítko `Mute` u jiných stop. Pomocí klávesy SHIFT na klávesnici můžeme ztlumit všechny vybrané stopy.

Pomocí tlačítka `Invert` můžeme invertovat signál dané stopy. Pro inverzi signálu všech vybraných stop lze použít tlačítka SHIFT při kliknutí na tlačítko.

Tlačítko `EQ` má několik funkcí. Při normálním kliknutí na tlačítko se otevře okno ekvalizéru, které bude popsáno v podkapitole 2.5. Ekvalizér můžeme vypnout a zapnout kliknutím se stisknutým tlačítkem CTRL. Pro vypnutí nebo zapnutí všech ekvalizérů vybraných stop je třeba při kliknutí zároveň použít tlačítko SHIFT.

Další ovládací prvek je horizontální slider pro nastavení zpoždění signálu v milisekundách. Pro nastavování hodnot lze použít i samotný slider, nebo po kliknutí lze zadat požadovanou hodnotu zpoždění úplně přesně. Je možné také použít tlačítko SHIFT při manipulaci se sliderem pro nastavování hodnot všech vybraných stop. Tlačítkem CTRL a kliknutím na slider lze hodnotu vynulovat. Pod sliderem jsou malá tlačítka pro přesnější nastavování zpoždění. Jsou tam tři skupiny tlačítek `-` a `+`. První skupina je pro rozlišení 1 ms, druhá pro 0,1 ms a třetí pro 0,01 ms. Minimální hodnota zpoždění je 0 ms a maximální nastavitelná hodnota je 500 ms.

Pod sliderem pro nastavování zpoždění signálu se nachází měřič pro vykreslování úrovně digitálního audio signálu. Měřič byl navržen podle normy IEC 60268-18 a doporučení EBU R.68. Kromě špiček audio signálu měřič vykresluje také globální špičku signálu, od začátku měření nebo od doby vynulování hodnoty, pomocí horizontální čáry. Tuto hodnotu lze vynulovat kliknutím na měřič. Globální špičky všech měřičů v dané skupině stop lze vynulovat pomocí tlačítka ALT na klávesnici a kliknutím na měřič.

Vedle měřiče je vertikální slider pro nastavování zesílení/zeslabení signálu v decibelech. Zesílení je možné také zadat pomocí textového pole pod sliderem. Maximální nastavitelná hodnota zesílení je 6 dB a minimální -60 dB. Zesílení je možné nastavovat pro všechny vybrané stopy pomocí tlačítka SHIFT. Vynulovat zesílení lze pomocí tlačítka CTRL a kliknutím na slider nebo dvojklikem.

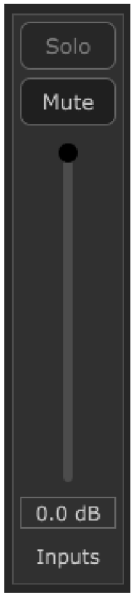
Pod měřičem je malé textové pole, kde je napsáno identifikační číslo stopy. Toto číslo přiřazuje program automaticky pro každou stopu a nelze ho ručně editovat.

Poslední prvek stopy je textové pole, kde je napsán název stopy. Název stopy přiřazuje buď program automaticky, nebo ho můžeme definovat při přidávání nových stop. Název je možné později změnit.

Vybírat stopy můžeme kliknutím na textové pole identifikačního čísla nebo kliknutím na prázdná místa stopy. Ve výchozím režimu po kliknutí je vybrána vždy jen jedna stopa. Vybraná stopa je překreslena na jinou barvu. Pro výběr více stop můžeme použít klávesu CTRL na klávesnici při klikání na stopy. Klávesu SHIFT na klávesnici můžeme použít pro výběr intervalu stop.

2.4.2 Hlavní stopa

Hlavní stopa je o mnoho jednodušší než normální stopy pro vstupy a výstupy. Jednu z hlavních stop hlavního okna můžeme vidět na obrázku 2.6. Popis ovládacích prvků bude také probíhat z hora dolů.



Obr. 2.6: Hlavní stopa skupiny v hlavním okně.

Tlačítko `Solo` je v normálním režimu deaktivované. Když je aktivní režim *solo* alespoň u jedné stopy, je toto tlačítko aktivováno, které po kliknutí ukončí režim *solo* u všech stop dané skupiny.

Tlačítko `Mute` v normálním režimu lze využít pro ztlumení všech stop v skupině. Když je alespoň jedna stopa v režimu *mute*, je toto tlačítko překresleno na zelenou barvu, které po kliknutí odstraní všechny stopy dané skupiny z režimu *mute*.

Další ovládací prvek hlavní stopy je vertikální slider pro nastavení zesílení/zeslabení všech stop v skupině. Slider umožňuje nastavit hodnoty od -20 dB po 0 dB. Zesílení je také možné nastavovat pomocí textového pole pod sliderem.

Poslední prvek hlavní stopy je textové pole, kde je napsáno název skupiny stop. Název nelze změnit, je to buď `Inputs` pro vstupy anebo `Outputs` pro výstupy.

2.4.3 Menu

Menu aplikace bude popisováno podle jednotlivých kategorií. Součástí hlavního menu jsou kategorie `Project`, `Edit`, `Track`, `Delay`, `Select`, `Global`, `Zoom`, `Routing`, `Device`, `Preferences` a `Theme`.

Ke každé funkci, která je součástí menu, lze přiřadit klávesové zkratky. Výchozí klávesové zkratky lze v programu změnit v okně `Key mapping editor`, které bude popisováno podrobněji v podkapitole 2.6. Klávesové zkratky jsou také uvedeny vedle názvů funkcí v menu.

Položky v menu jsou aktivovány a deaktivovány podle podmínek.

Project

Podnabídku kategorie `Project` můžeme vidět na obrázku 2.7.

Obr. 2.7: Podnabídka kategorie `Project` hlavního menu.

Pomocí položky `New` v podnabídce můžeme vytvořit nový projekt. Před zavřením aktivního projektu aplikace nás varuje, když projekt obsahuje neuložené změny. Vytvoření nového projektu je zahájeno výběrem zvukového zařízení, který již byl popsán v části 2.3.

Položka `Open...` funguje stejně jako tlačítko `Open project...` uvítacího okna. Aplikace nás varuje před zavřením projektu s neuloženými změnami. Otevřít projekt lze také stejným „drag and drop“ způsobem jako u uvítacího okna.

Funkce `Save...` uloží projekt do souboru s příponou `.sofaa`. Když projekt ještě nebyl uložen, otevře se dialogové okno pro umístění nového souboru.

Pomocí funkci `Save As...` můžeme projekt uložit do nového souboru `.sofaa`.

Položka `Save New Version` byla navržena pro rychlé ukládání projektu do nového souboru. Po kliknutí na tuto položku nebo po použiti klávesové zkratky se neobjeví žádné dialogové okno. Projekt je uložen do nového souboru s názvem původního souboru, ale na konec názvu je přidáno číslo, které je inkrementováno po každém použití této funkce. Jako příklad si vezmeme projekt s názvem `Sample project`. Po prvním použití funkce bude vytvořen nový projekt s názvem `Sample project-01`, po dalším použití `Sample project-02` a tak dále.

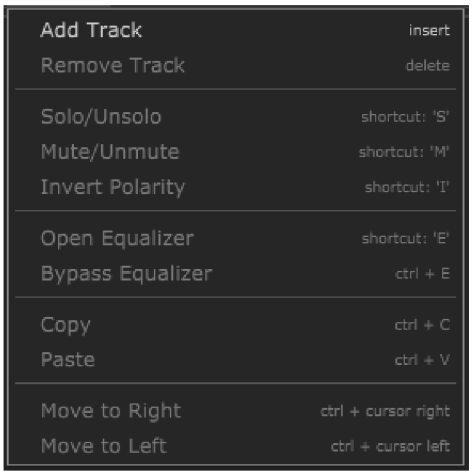
Položka `Exit` v podnabídce kategorie `Project` slouží na ukončení programu. Před zavřením programu nás aplikace upozorní na neuložené změny v projektu.

Edit

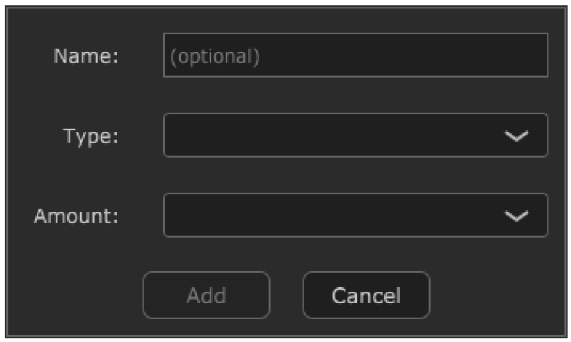
Podnabídka kategorie `Edit` obsahuje pouze dvě položky: `Undo` a `Redo`. Skoro všechny změny v aplikaci jsou ukládány a lze je pomocí funkce `Undo` vrátit nebo pomocí funkce `Redo` znovu provést. Položka `Undo` je aktivována, když nastanou změny, které lze vrátit. Položka `Redo` je aktivována po použití funkce `Undo`.

Track

Do podnabídky kategorie `Track` jsou umístěny funkce aplikace, které provádí změny se stopami. Podnabídku můžeme vidět na obrázku 2.8.



Obr. 2.8: Podnabídka kategorie `Track` hlavního menu.



Obr. 2.9: Okno pro přidávání nových stop.

Pomocí položky `Add Track` můžeme přidat nové stopy do projektu. Stopy lze také přidávat dvojklikem v panelu pro stopy. Funkce je deaktivována, když v projektu je dosažen maximální počet vstupů a výstupů vybraného zvukového zařízení.

Po kliknutí na položku `Add Track` se otevře okno pro přidávání nových stop, které můžeme vidět na obrázku 2.9. V okně musíme nejprve vybrat typ nové stopy a pak definovat počet stop. Stopu lze zde také pojmenovat.

Funkce `Remove Track` odstraní všechny vybrané stopy z projektu. Podmínkou aktivace funkce je výběr minimálně jedné stopy.

Podmínkou aktivace pro skupinu dalších třech funkcí je také výběr alespoň jedné stopy. Funkce `Solo/Unsolo` přidá nebo odebere vybrané stopy z režimu *solo*. Funkce `Mute/Unmute` funguje obdobně pro režim *mute*. Položka `Invert Polarity` invertuje signál vybraných stop.

Položka `Open Equalizer` otevře okno ekvalizéru pro vybranou stopu. Podmínkou aktivace je výběr jedné stopy. Okno ekvalizéru bude podrobněji popsáno v části 2.5.

Funkce `Bypass Equalizer` slouží pro přepínáni ekvalizéru vybraných stop mezi *bypass* a normálním režimem, neboli pro vypnutí a zapnutí ekvalizéru. Pro aktivaci položky musí být vybrána alespoň jedna stopa.

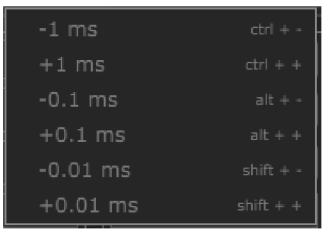
Pomocí funkce `Copy` můžeme kopírovat všechna nastavení vybrané stopy. Podmínkou aktivace funkce je výběr jedné stopy.

Funkce `Paste` přidělí již zkopírovaná nastavení pro vybranou stopu. Před použitím funkce `Paste` musíme použít funkci `Copy` a vybrat jednu stopu.

Další dvě položky podnabídky fungují skoro stejným způsobem. Položka `Move to Right` přesune vybranou stopu o jedno místo doprava mezi ostatními stopami. Položka `Move to Left` přesune vybranou stopu o jedno místo doleva mezi ostatními stopami. Tyto položky jsou aktivovány, když je vybrána jedna stopa. Použití těchto funkcí je jediný způsob pro změnu identifikačního čísla kanálu.

Delay

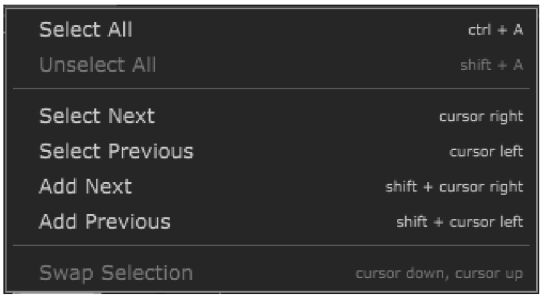
Položky této podnabídky jsou navrženy pro přesnější nastavování zpoždění vybraných stop a můžeme je vidět na obrázku 2.10. Položky jsou aktivovány po výběru alespoň jedné stopy.



Obr. 2.10: Podnabídka kategorie `Delay` hlavního menu.

Select

Podnabídku kategorie `Select` můžeme vidět na obrázku 2.11.



Obr. 2.11: Podnabídka kategorie `Select` hlavního menu.

Položka `Select All` vybere všechny stopy dané skupiny.

Položka `Unselect All` odstraní všechny stopy z výběru. Tuto funkci lze zavolat i kliknutím v panelu pro stopy. Tato funkce je aktivována, když je vybrána alespoň jedna stopa.

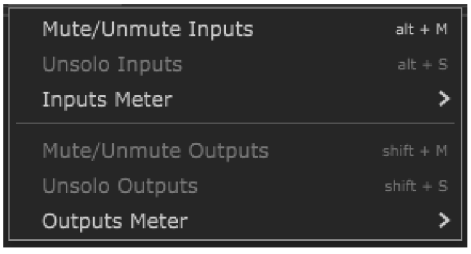
Funkce `Select Next` vybere další stopu a funkce `Select Previous` vybere předchozí stopu ve skupině stop.

Pomocí funkce `Add Next` můžeme do výběru přidat další stopu ve skupině a pomocí funkce `Add Previous` můžeme přidat předchozí stopu do výběru.

Funkce `Swap Selection` byla navržena pro přehození výběru mezi skupinami stop, neboli pro snadné procházení mezi vstupy a výstupy.

Global

Položky podnabídky kategorie `Global` můžeme vidět na obrázku 2.11. Položky jsou rozděleny do dvou skupin: jedna skupina pro vstupy a druhá skupiny pro výstupy.



Obr. 2.12: Podnabídka kategorie `Global` hlavního menu.

Skupina prvních třech položek podnabídky nastavuje vlastnosti vstupů. Položky fungují obdobně jako tlačítka hlavní stopy skupiny.

Když ani jedna stopa ze vstupů není v režimu *mute*, funkce `Mute/Unmute Inputs` přepne všechny stopy do tohoto režimu. Když je v režimu *mute* alespoň jedna stopa ze skupiny, tato funkce vypne režim *mute* pro všechny vstupy.

Položka `Unsolo Inputs` je aktivována, když je v režimu *solo* alespoň jedna stopa ze skupiny vstupů. Funkce vypne režim *solo* pro všechny stopy dané skupiny.

Pomocí položky `Inputs Meter` můžeme přepínat režim měřičů pro stopy ve skupině mezi „pre fader“ a „post fader“ režimy. Režim „pre fader“ posílá do digitálního měřiče hodnoty signálu před zesilováním/zeslabováním signálu a režim „post fader“ posílá do digitálního měřiče hodnoty signálu po zesílení/zeslabení signálu.

Druhá skupina položek v podnabídce podrobněji nebude popisována, jelikož provádí stejné operace pro výstupy jako první skupina pro vstupy.

Zoom

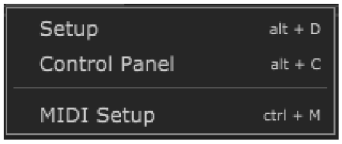
Podnabídka kategorie `Zoom` obsahuje dvě položky. Pomocí položky `Zoom In` můžeme zvětšit šířku stop a pomocí položky `Zoom Out` je možné šířku stop zmenšit.

Routing

Podnabídka kategorie `Routing` obsahuje jen jednu položku `Open Matrix`. Položka je aktivována, když je v projektu alespoň jeden vstup a jeden výstup. Kliknutím na tuto položku lze otevřít matici pro slučování a směrování signálů vstupních stop do libovolných výstupních stop. Matice bude podrobněji popsána v části 2.7.

Device

Podnabídka kategorie `Device` obsahuje položky pro nastavení zvukového a MIDI zařízení. Tuto podnabídku můžeme vidět na obrázku 2.13.



Obr. 2.13: Podnabídka kategorie `Device` hlavního menu.

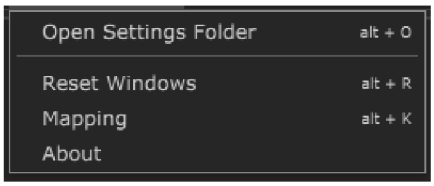
Položka `Setup` otevře okno pro výběr zvukového zařízení, které již bylo popsáno v podkapitole 2.3.

Položka `Control Panel` je aktivována pro zařízení, které mají ovládací panel a po kliknutí na tuto položku se ovládací panel zvukového zařízení otevře.

Položka `MIDI Setup` slouží pro nastavování MIDI zařízení a MIDI zpráv pro ovládání aplikace pomocí protokolu MIDI. Po kliknutí na tuto položku je otevřeno okno pro upravování nastavení MIDI, které je podrobněji popsáno v části 2.8.

Preferences

Podnabídka kategorie `Preferences` obsahuje další položky pro úpravu nastavení aplikace. Podnabídku můžeme vidět na obrázku 2.14.



Obr. 2.14: Podnabídka kategorie `Preferences` hlavního menu.

Položka `Open Settings Folder` této podnabídky otevře složku, kde jsou uložena všechna nastavení programu. Tato nastavení jsou uložena do souborů s příponami `.settings`.

Funkce `Reset Windows` nastaví velikosti a umístění všech oken aplikace na výchozí hodnoty v souboru `Windows.settings`.

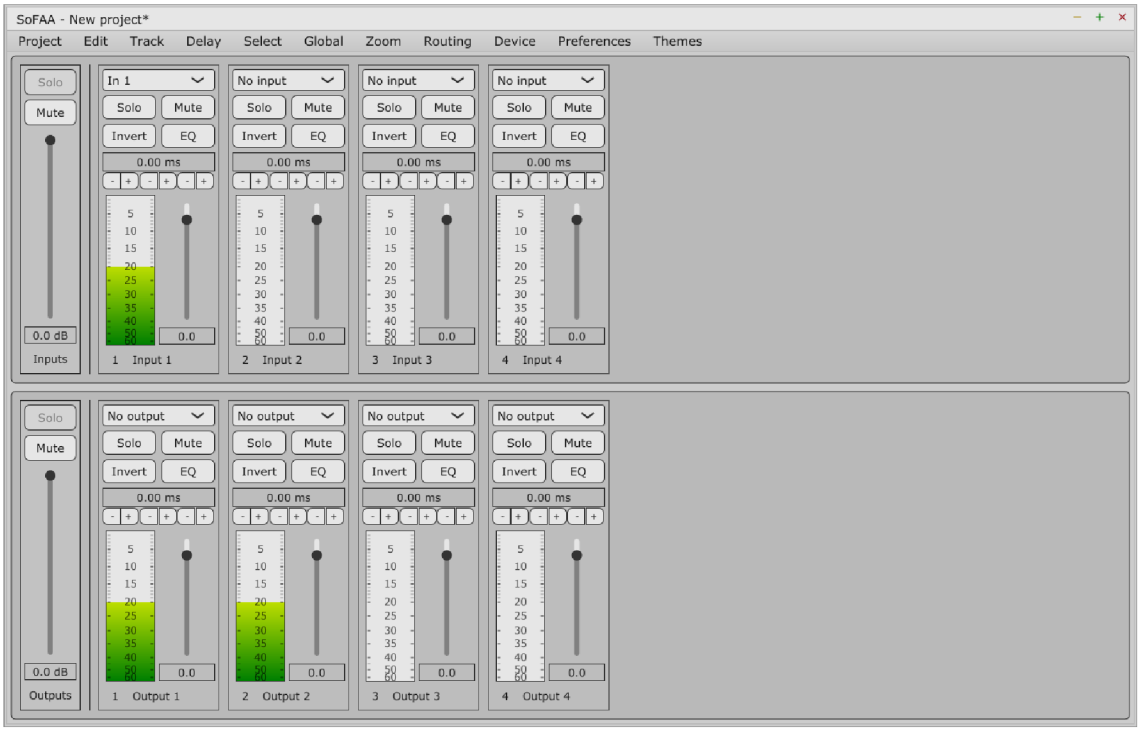
Položka `Mapping` otevře okno pro editaci klávesových zkratek, které bude popsáno v části 2.6.

Poslední položka podnabídky `About` otevře okno s informacemi o aplikaci (obrázek 2.15).

Obr. 2.15: Okno `About` – „O aplikaci“.

Theme

Poslední kategorie hlavního menu slouží pro změnu vzhledu aplikace. Aplikace je dostupná, jak v tmavém provedení, které je výchozí, tak i ve světlém provedení. Světlý režim aplikace můžeme vidět na obrázku 2.16. Režim aplikace je uložen do souboru `Theme.settings` ve složce nastavení programu a je načten při každém spuštění.



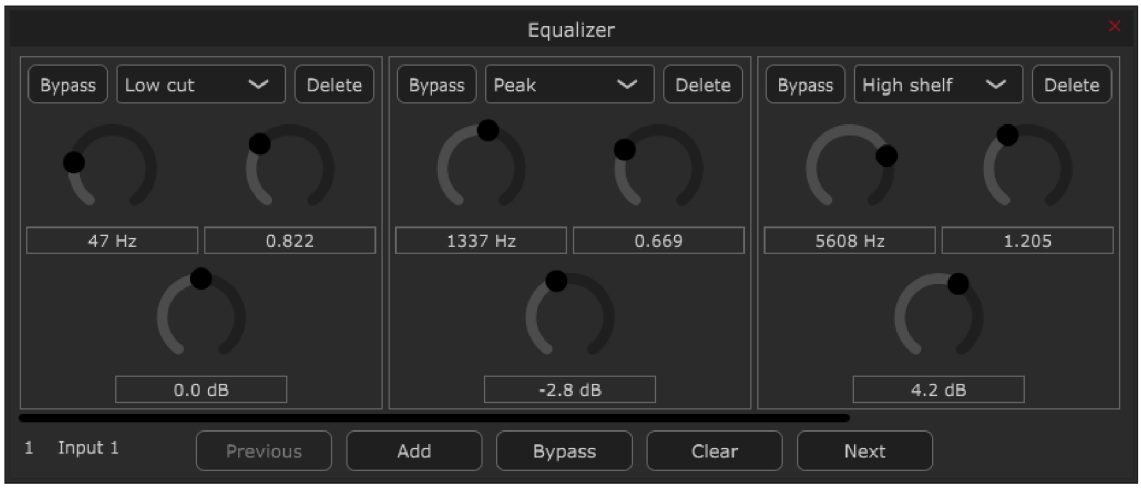
Obr. 2.16: Hlavní okno aplikace ve světlém režimu.

2.5 Ekvalizér

V této podkapitole budou popisovány ovládací prvky okna ekvalizéru a fungování ekvalizéru. Okno ekvalizéru můžeme vidět na obrázku 2.17.

Každá stopa disponuje parametrickým ekvalizérem, který umožňuje signál filtrovat maximálně šesti filtry. Maximální počet filtrů v ekvalizéru byl omezen, aby program mohl fungovat v reálném čase i pro větší počet stop.

Okno ekvalizéru lze přemísťovat a také lze změnit jeho velikost. Změny jsou ukládány do souboru `Windows.settings` a lze je v aplikaci resetovat.



Obr. 2.17: Okno ekvalizéru.

Na obrázku 2.17 můžeme vidět ekvalizér se čtyřmi filtry. Filtry můžeme procházet pomocí posuvníku.

Filtr můžeme pomocí tlačítka `Bypass` vypínat a zapínat. Tlačítko `Delete` slouží k odstranění jednoho filtru. Kromě toho každý filtr má rozbalovací seznam pro výběr typu filtru. Aplikace umožňuje použití pěti typů filtrů: `Low cut`, `Low shelf`, `Peak`, `High shelf` a `High cut`. Každý filtr má tři rotační slidery pro nastavování parametrů. Tyto parametry jsou: `Frequency`, `Q` a `Gain`. Hodnoty parametrů lze také zadat pomocí textových polí pod slidery.

Okno ekvalizéru dole v levém rohu ukazuje identifikační číslo a název stopy, kterou právě upravujeme.

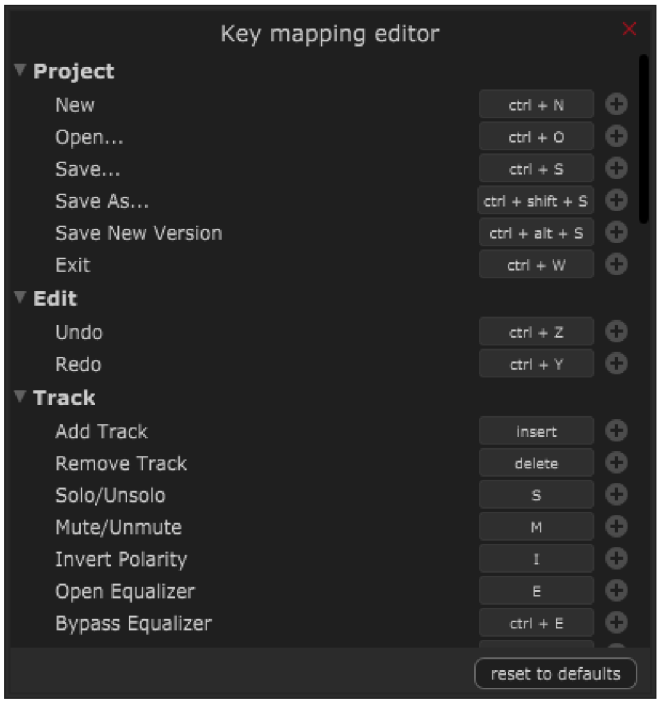
Tlačítko `Previous` a tlačítko `Next` umožňují rychlé otevírání oken ekvalizéru pro předchozí a pro následující stopy. Tyto operace lze také docílit pomocí šipek doleva a doprava na klávesnici.

Přidat nový filtr je možné pomocí tlačítka `Add`. Tlačítko `Clear` slouží k odstranění všech filtrů ekvalizéru. Pomocí tlačítka `Bypass` lze vypínat a zapínat ekvalizér.

2.6 Klávesové zkratky

Celá aplikace byla navržena tak, aby se ji dalo používat i pomocí klávesových zkratek. Klávesové zkratky jsou ukládány v souboru `Mapping.settings`. Tyto zkratky lze změnit v okně `Key mapping editor`, které můžeme vidět na obrázku 2.18. Je zde seznam všech funkcí aplikace, kterým je možné přiřadit klávesové zkratky. Můžeme zde přidat nové, odstranit nebo změnit již nastavené klávesové zkratky pro dané funkce. Pomocí tlačítka `reset to defaults` je možné všechny zkratky vrátit do výchozího stavu.

Velikost okna lze změnit a okno přemísťovat. Změny jsou ukládány do souboru `Windows.settings` a lze je v aplikaci resetovat.

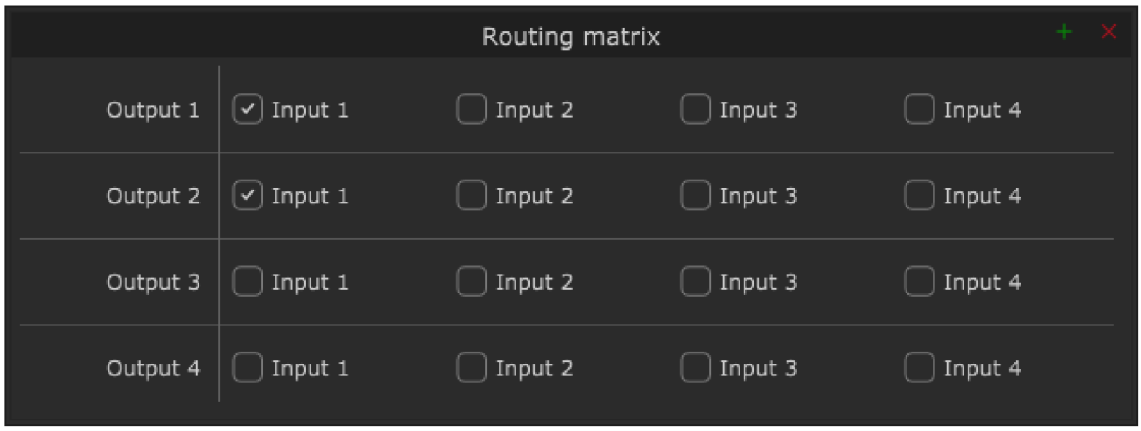


Obr. 2.18: Okno pro editaci klávesových zkratek.

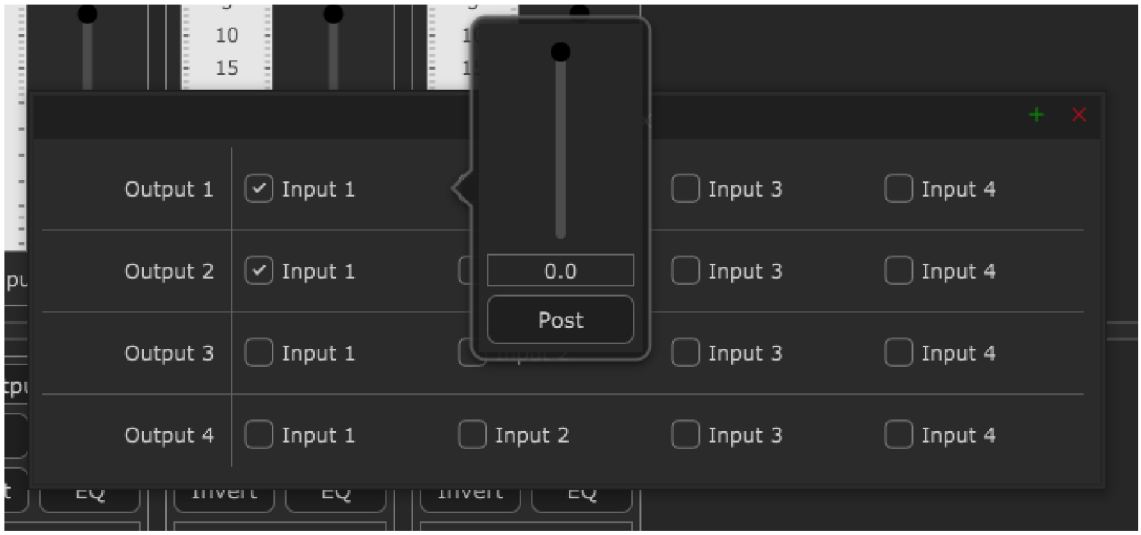
2.7 Matice

Matici pro slučování a směrování signálů libovolných vstupů do libovolných výstupů můžeme vidět na obrázku 2.19. Změny ve velikosti a umístění okna jsou taktéž ukládány v souboru `Windows.settings` a lze je v aplikaci resetovat.

Matice je navržena pro jednoduché a intuitivní použití. Zaškrtnutím zaškrtávacího pole před názvem vstupu můžeme signál této stopy směrovat do výstupu, kterého název je na začátku řádku. Po zaškrtnutí zaškrtávacího pole lze otevřít další možnosti směrování signálu jednoduchým způsobem: kurzor myši necháme nad zaškrtávacím polem a malé okno se automaticky objeví po uplynutí doby jedné sekundy. Toto okénko můžeme vidět na obrázku 2.20. Můžeme zde nastavit zesílení signálu vstupu pomocí vertikálního slideru anebo textového pole pod sliderem pro daný výstup a také můžeme přepínat mezi režimy „pre fader“ a „post fader“. Režim „pre fader“ posílá signál ze vstupu před sliderem pro zesilování signálu stopy a „post fader“ posílá signál ze vstupu až po slideru pro zesilování signálu stopy. Když kurzor myši opustí toto okénko a následně uplyne doba jedné sekundy, okénko se zavře.



Obr. 2.19: Matice pro slučování a směrování vstupních signálů do libovolných výstupů.

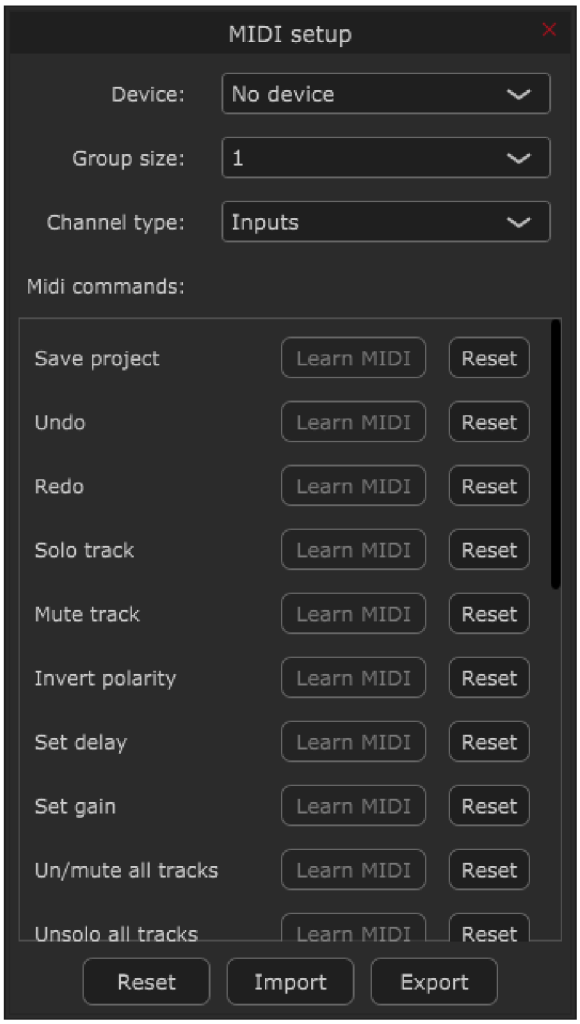


Obr. 2.20: Okénko pro nastavování vlastností vstupního signálu.

2.8 MIDI ovládání

Některé funkce aplikace lze ovládat i pomocí MIDI zařízení. Upravovat potřebná nastavení pro tuto funkci programu lze v okně `MIDI Setup`, které můžeme vidět na obrázku 2.21. Tato nastavení jsou ukládána v souboru `Midi.settings` ve složce nastavení programu. Okno lze přemísťovat a také lze změnit jeho velikost. Změny jsou ukládány do souboru `Windows.settings` a lze je v aplikaci resetovat.

První ovládací prvek okna je rozbalovací seznam pro výběr vstupního MIDI zařízení. Po nastavení zařízení, aplikace přijímá všechny MIDI zprávy, které jsou vysílány vybraným zařízením.

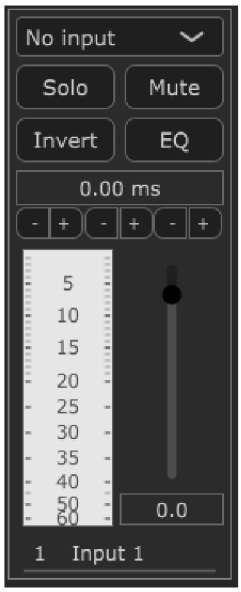


Obr. 2.21: Okno MIDI nastavení.

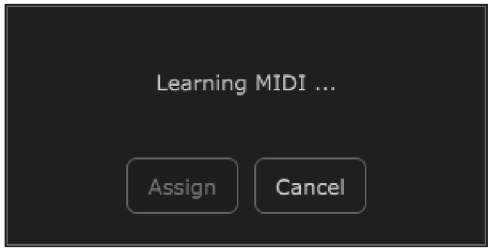
Druhý ovládací prvek je také rozbalovací seznam pro nastavení velikosti skupiny stop, které lze pomocí zařízení ovládat. Maximální hodnota velikosti skupiny stop je 16. Každá stopa ve skupině přijímá MIDI zprávy z jednoho MIDI kanálu. Když velikost skupiny je například 8, pak první vybraná stopa přijímá MIDI zprávy MIDI kanálu 1 a poslední přijímá MIDI zprávy MIDI kanálu 8. Tímto způsobem lze jedním zařízením ovládat více stop programu. Vybrané stopy jsou označeny horizontální čárkou pod názvem stopy (znázorněno na obrázku 2.22).

Dalším prvkem okna je také rozbalovací seznam, kde lze nastavit typ vybraných stop. Toto nastavení lze změnit i pomocí MIDI zprávy pro rychlé a efektivní využití MIDI zařízení.

V další části okna je seznam všech funkcí, které lze pomocí MIDI zpráv ovládat. Pro přiřazení kontroleru lze použít tlačítko `Learn MIDI`, které otevře dialogové okno. Toto okno můžeme vidět na obrázku 2.23.



Obr. 2.22: Stopa vybrána aktivním MIDI zařízením.



Obr. 2.23: Okno `MIDI Learn`.

Okno `MIDI Learn` čeká, dokud nepřijme první MIDI zprávu. Z MIDI zprávy extrahuje číslo kontroleru, které následně vypíše do okénka. Když číslo ještě není použito, lze ho pomocí tlačítka `Assign` přiřadit k dané funkci.

Pomocí tlačítka `Reset` vedle názvu funkce lze přiřazené číslo MIDI kontroleru resetovat.

Tlačítko `Reset` na spodku okna resetuje čísla MIDI kontrolerů u všech funkcí v seznamu.

Přiřazená čísla MIDI kontrolerů k funkcím aplikace můžeme pomocí tlačítka `Export` uložit do souboru s příponou `.midimap`. Ze souborů se stejnou příponou pak můžeme načítat všechna přiřazená čísla MIDI kontrolerů, buď pomocí tlačítka `Import`, nebo pomocí gesta „drag and drop“.

Funkce `Save Project` slouží k uložení projektu.

Funkce `Undo` a `Redo` mají stejné úlohy jako v podnabídce kategorie `Edit` hlavního menu.

Funkce `Solo track` přepíná vybranou stopu ve skupině mezi *solo* režimem a normálním režimem.

Funkce `Mute track` slouží k přepínání vybrané stopy ze skupiny mezi *mute* režimem a normálním režimem.

Funkce `Invert polarity` invertuje signál vybrané stopy.

Pomocí funkce `Set delay` lze nastavovat zpoždění vybraných stop.

Pomocí funkce `Set gain` lze nastavovat zesílení/zeslabení vybraných stop.

Funkce `Un/mute all tracks` slouží k přepínání všech stop v dané skupině stop mezi *mute* režimem a normálním režimem.

Funkce `Unsolo all tracks` slouží k přepnutí všech stop dané skupiny stop do normálního režimu z režimu *solo*.

Stopy lze procházet pomocí MIDI zpráv. Pomocí funkce `Go to previous track` můžeme výběr posunout o stopu zpátky a pomocí funkce `Go to next track` lze výběr posunout o jednu stopu dopředu. Funkce `Go to previous group` slouží k posunu MIDI výběru stop zpátky o předem definovanou velikost skupiny. Funkce `Go to next group` slouží k posunu MIDI výběru stop dopředu o stejnou hodnotu.

Ovládání ekvalizéru bylo navrženo tak, že první filtr přijímá MIDI zprávy z MIDI kanálu 1, druhý z MIDI kanálu 2 a tak dále. Aplikace tak byla navržena z důvodu, aby bylo možné upravovat všechny parametry všech filtrů ekvalizéru dané stopy.

Funkce `Change filter type` změní typ filtru vybrané stopy.

Funkce `Change filter frequency` slouží ke změně frekvence filtru vybrané stopy.

Funkce `Change filter Q` slouží ke změně parametru „Q“ filtru vybrané stopy.

Funkce `Change filter gain` slouží ke změně zesílení/zeslabení filtru vybrané stopy.

Pomocí funkce `Bypass filter` lze filtr jednoduše vypínat a zapínat.

Pomocí funkce `Set master gain` lze nastavovat zesílení/zeslabení všech stop dané skupiny stop.

Funkce `Change channel type` slouží k prohození typu vybraných stop.

3 Objekty a návrh aplikace

Tato část diplomové práce popisuje všechny objekty aplikace a jejich funkci. Jsou zde také vysvětlena řešení některých problematických oblastí a uvedeny části zdrojového kódu.

Jelikož aplikace má skoro deset tisíc řádků a 70 zdrojových souborů, není možné úplně všechno popisovat do detailů v této práci. Ze stejného důvodu a kvůli skutečnosti, že jsem program implementoval úplně sám bez vnější pomoci, je možné že v aplikaci ještě zůstaly neošetřené případy, které jsem nebyl schopen sám najít při testování.

Celý JUCE projekt aplikace lze najít na přiloženém disku ve složce `SoFAA`. Zdrojový kód aplikace byl okomentován, aby byl co nejpřehlednější.

Zdrojové soubory aplikace jsou rozděleny do skupin `BinaryData`, `Channels`, `Midi`, `Source` a `Windows`.

Skoro všechny objekty aplikace jsou rozděleny do dvou souborů. V hlavičkovém souboru `.h` jsou deklarace všech funkcí a proměnných objektu a v souboru `.cpp` jsou definice deklarovaných funkcí. Implementace objektů je ve většině případů dost složitá, takže samotná implementace objektů nebude popisována, jen hlavní funkce objektů a zajímavá programátorská řešení.

Základem programu je šablona `Audio Application` knihovny JUCE, která byla navržena pro aplikace pracující s audio signály a tyto signály umožňuje zpracovávat i v reálném čase.

Všechna data projektu jsou ukládána do jedné instance `ValueTree`, což umožňuje aplikaci jednoduché načítání dat a rychlé reagování na změnu dat.

3.1 BinaryData

Skupina `BinaryData` obsahuje pouze dva soubory. Jsou to SVG (*Scalable Vector Graphics*) soubory vektorových obrázků. Je zde logo (`sofaa_logo.svg`) a takzvaný banner (`sofaa_baner.svg`) aplikace. Knihovna JUCE převádí tyto soubory na binární data, což pak lze v aplikaci použít pro vykreslování obrázků.

3.2 Channels

Skupina `Channels` obsahuje všechny zdrojové soubory, které mají nějakou spojitost se stopy aplikace. Kromě souborů je zde ještě jedna skupina `Equalizer`, která obsahuje soubory, které tvoří ekvalizér: `EqualizerBand.cpp`, `EqualizerBand.h`, `Equalizer.cpp` a `Equalizer.h`.

Skupina `Channels` obsahuje ještě objekty `Channel`, `ChannelBox`, `ChannelRow`, `Delay`, `MasterChannel`, `Meter`, `NewTrack` a `Routing`.

3.2.1 Equalizer

Objekt `Equalizer` tvoří základ okna ekvalizéru. Objekt obsahuje funkce pro smazání filtrů, pro přidávání filtrů, pro zapnutí a vypnutí režimu *bypass* ekvalizéru, pro filtrování signálu pomocí objektů `EqualizerBand` a mnoho dalších.

Výpis 3.1: Funkce `run` nového vlákna objektu `Equalizer`.

```
348 if (fadeBypass) {
349   std::atomic<bool> smoothing;
350   smoothing.store(gain.isSmoothing());
351   while (smoothing.load()) {
352     smoothing.exchange(gain.isSmoothing());
353   }
354
355   this->bypassed = true;
356   fadeBypass = false;
357 }
358 if (remove.size() > -1) {
359   for (int i = 0; i < remove.size(); i++) {
360     int index = remove[i];
361     eqBands[index]->getGain().setGainLinear(0);
362     eqBands[index]->getBypassGain().setGainLinear(1);
363
364     std::atomic<bool> smoothing;
365     smoothing.store(eqBands[index]->getGain().isSmoothing());
366     while (smoothing.load()) {
367       smoothing.exchange(eqBands[index]->getGain().
            isSmoothing());
368     };
369
370     MessageManagerLock lock;
371     eqBands.remove(index);
372     resized();
373     if (eqBands.size() < EQ_MAX_BAND) addButton->setEnabled(
          true);
374   }
375   remove.clear();
376 }
```

Zajímavé řešení v ekvalizéru je vytvoření nového vlákna pro „fadeování“ signálu při smazání všech filtrů a při přepnutí ekvalizéru do režimu *bypass*, aby vlákno pro vykreslování aplikace mohlo fungovat dále i během tohoto časového intervalu. Kromě toho bylo třeba použít pro tuto funkci typ `std::atomic` pro některé proměnné, aby výměna dat mezi více vlákny byla bezpečná. Funkci `run` nového vlákna můžeme vidět ve výpisu 3.1.

Po vytvoření nového vlákna je funkce `run` volána až do doby přerušení vlákna. První část funkce provádí „fade out“ při zapnutí režimu *bypass* ekvalizéru. Druhá část funkce slouží pro prolnutí originálního signálu a filtrovaného signálu po odstranění filtru, aby nedocházelo k digitálním audio artefaktům. Zajímavá část funkce je vytvoření nové instance objektu `MessageManagerLock`, která propojí nové vlákno s výchozím vláknem knihovny JUCE jen pro následující příkazy. Tímto způsobem je práce mezi více vlákny bezpečná.

3.2.2 EqualizerBand

Objekt `EqualizerBand` představuje jeden filtr ekvalizéru.

„Fade out“ signálu při přepnutí do režime *bypass* je vyřešeno také vytvořením nového vlákna.

Hlavní částí objektu jsou filtry, které jsou součástí knihovny JUCE. Objekt obsahuje 2 filtry typu IIR a to z toho důvodu, že při změně typu filtru objektu je vytvořeno prolnutí mezi signálem přes starý filtr a signálem přes nový filtr. Tímto způsobem jsou eliminovány náhlé změny v signálu.

Všechny parametry filtru využívají třídu `SmoothedValue` knihovny JUCE, aby při manipulaci s filtrem nebyly slyšitelné žádné lupance v signálu. Pro parametry `Gain` a `Q` byl použit typ vyhlazování hodnot `ValueSmoothingTypes::Linear` a pro frekvenci `ValueSmoothingTypes::Multiplicative`.

Hlavní funkcí objektu je filtrování signálu podle zadaných parametrů. Knihovna JUCE disponuje funkcemi pro vytvoření filtrů, v tomto případě pro vytváření IIR filtrů.

Ve jmenném prostoru `DSP` knihovny JUCE jsou k dispozici třídy pro návrh filtrů. Funkce třídy `IIR::Coefficients` vrací koeficienty IIR filtrů, které lze ukládat a následně používat pomocí instancí třídy `IIR::Filter`. Výpočet koeficientů lze velice snadno najít ve zdrojovém kódu knihovny. Pro objekt `EqualizerBand` byla vytvořena funkce `updateCoefficients`, která podle argumentů aktualizuje parametry filtru v objektu. Část této funkce lze vidět ve výpisu 3.2, která provádí změnu typu filtru podle nových parametrů.

Výpis 3.2: Část funkce `updateCoefficients` objektu `EqualizerBand`.

```
206 switch (type) {
207     //Low cut
208   case 1:
209     filter.coefficients = IIR::Coefficients<float>::
            makeHighPass(sampleRate, freq, q);
210     break;
211     //Low shelf
212   case 2:
213     filter.coefficients = IIR::Coefficients<float>::
            makeLowShelf(sampleRate, freq, q, gain);
214     break;
215     //Peak
216   case 3:
217     filter.coefficients = IIR::Coefficients<float>::
            makePeakFilter(sampleRate, freq, q, gain);
218     break;
219     //High shelf
220   case 4:
221     filter.coefficients = IIR::Coefficients<float>::
            makeHighShelf(sampleRate, freq, q, gain);
222     break;
223     //High cut
224   case 5:
225     filter.coefficients = IIR::Coefficients<float>::
            makeLowPass(sampleRate, freq, q);
226     break;
227   }
```

3.2.3 Channel

Jeden z nejsložitějších objektů aplikace je objekt `Channel`, který tvoří základ stop v aplikaci. Je zde logika výběru stop, logika režimu *mute*, logika režimu *solo*, instance všech objektů stopy a mnoho dalších funkcí. Fungování vnitřních funkcí objektu je velice komplexní.

Fungování stop aplikace již bylo popsáno v části 2.4.1.

Asi nejzajímavější funkcí objektu je funkce `setBlock`, která zpracovává signál stopy. Funkce bude popisována po částech (výpis 3.3 a výpis 3.4).

Funkce nejprve uloží `AudioBlock`, neboli vzorky bufferu signálu a následně je zpracovává.

Výpis 3.3: První část funkce `setBlock` objektu `Channel`.

```
260 this->preBlock = block;
261
262   //Delete
263   if (shouldBeDeleted) {
264     gainToApply.setGainLinear(0);
265   }
266   //Solo
267   else if (soloGroup && !soloBool) {
268     gainToApply.setGainLinear(0);
269   }
270   //Delay change
271   else if (delay.getDelayTime() != delay.getDelayTarget()) {
272     gainToApply.setGainLinear(0);
273     if (!gainToApply.isSmoothing() && delay.
          getDelayTimeTarget() != delay.getDelayTarget()) {
274       delay.updateDelay();
275     }
276   }
277   else {
278     //Mute
279     if (muteBool) {
280       gainToApply.setGainLinear(0);
281     }
282     //IO change
283     else if (io != ioTarget) {
284       gainToApply.setGainLinear(0);
285       if (!gainToApply.isSmoothing()) {
286         io = ioTarget;
287       }
288     }
289     //Invert
290     else if (invert) {
291       gainToApply.setGainLinear(-1);
292     }
293     //Play
294     else {
295       gainToApply.setGainLinear(1);
296     }
297   }
```

Výpis 3.4: Druhá část funkce `setBlock` objektu `Channel`.

```
299   //Pre block
300   ProcessContextReplacing<float> preContext(preBlock);
301   if(!equalizer->isBypassed() && equalizer->getEqBands().size
          () > 0) equalizer->filter(preContext);
302   delay.process(preContext);
303   gainToApply.process(preContext);
304
305   //Post block
306   postBuffer = AudioBuffer<float>::AudioBuffer(1, preBlock.
          getNumSamples());
307   postBlock = AudioBlock<float>::AudioBlock(postBuffer);
308   postBlock.copyFrom(preBlock);
309   ProcessContextReplacing<float> postContext(postBlock);
310   postGain.setGainLinear(gain);
311   postGain.process(postContext);
312
313   //Pre metering
314   if (meterType == "pre") {
315     Range<float> range = preBlock.findMinAndMax();
316     meter->setSample(std::abs(range.getStart()) > range.
          getEnd() ? std::abs(range.getStart()) : range.getEnd()
          );
317   }
318   //Post metering
319   else if (meterType == "post") {
320     Range<float> range = postBlock.findMinAndMax();
321     meter->setSample(std::abs(range.getStart()) > range.
          getEnd() ? std::abs(range.getStart()) : range.getEnd()
          );
322   }
```

První část funkce sice vypadá složitě, ale vlastně jen nastavuje zesílení signálu pomocí podmínek. Pro zesílení a zeslabení signálu je použita instance třídy `Gain` knihovny JUCE, která vyhlazuje změny hodnot zesílení. Když stopa má být odstraněna, zesílení je nastaveno na nulu, čím je docíleno „fade out“ signálu. Při změně zpoždění stopy nebo změně vstupu/výstupu stopy je signál nejprve ztlumen a následně zesílen. Když signál je v režimu *mute*, je zesílení nastaveno na 0. Když je režim *solo* aktivní, ale stopa v režimu *solo* není, zesílení je také nastaveno na 0. Pro invertování signálu je zesílení nastaveno na hodnotu -1.

Druhá část funkce rozděluje signál do dvou bufferů. Instance `preBlock` třídy `AudioBlock` je buffer vzorků stopy před použitím zesilování/zeslabování pomocí hodnoty ze slideru stopy a druhý buffer `postBlock` obsahuje vzorky, které již byly zesíleny nebo zeslabeny. Signál stopy je zde také filtrován pomocí instance třídy `Equalizer` a zpožděn pomocí instance třídy `Delay`. Poslední část funkce posílá vzorky digitálního signálu do instance objektu `Meter`.

3.2.4 ChannelBox

Objekt `ChannelBox` je velice jednoduchý. Obsahuje instanci třídy `ChannelRow` a instanci třídy `MasterChannel`. Objekt tedy obsahuje stopy a hlavní stopu jedné skupiny stop.

3.2.5 ChannelRow

Objekt `ChannelRow` je také velice jednoduchý. Obsahuje pouze instance třídy `Channel`. Objekt `ChannelRow` je vlastně pomocný objekt pro uspořádání stop.

3.2.6 Delay

Objekt `Delay` provádí zpoždění signálu stopy pomocí kruhové paměti. Objekt také využívá lineární interpolaci pro lepší rozlišení zpožďovací linky. Funkci pro výpočet lineární interpolace můžeme vidět ve výpisu 3.5. Funkce funguje přesně podle popisu lineární interpolace z části 1.3.3 této práce.

Výpis 3.5: Funkce lineární interpolace v objektu `Delay`.

```
113 float Delay::linearInterpolation(float actual, float previous
    ) {
114   float difference = previous - actual;
115   return actual + delayFraction * difference;
116 }
```

Hlavní funkce objektu `Delay` je funkce `delayWithInterpolation` (výpis 3.6), která provádí zpoždění signálu podle popisu v části 1.3.3. Argument funkce je buffer vzorků signálu stopy a číslo aktuálního vzorku v bufferu. Ostatní potřebné hodnoty jsou ukládány v objektu jako globální proměnné.

Funkce nejprve vypočítá interpolovanou hodnotu aktuálního vzorku zpožďovací linky a uloží ji do dočasné proměnné. Po uložení aktuální hodnoty vzorku stopy do kruhové paměti zpožďovací linky, přepíše hodnotu v bufferu stopy na vypočítanou hodnotu, která byla uložena v dočasné proměnné.

Výpis 3.6: Funkce `delayWithInterpolation` objektu `Delay`.

```
89  void Delay::delayWithInterpolation(AudioBlock<float>& block,
        size_t& i) noexcept {
90    float out = 0;
91    if (delayFraction == 0) out = delay[iterator];
92    else {
93      if (iterator == delaySize - 1) {
94        if (delaySize == 1)
95          out = linearInterpolation(block.getSample(0, i),
                delay[0]);
96        else
97          out = linearInterpolation(delay[0], delay[delaySize -
                1]);
98      }
99      else
100       out = linearInterpolation(delay[iterator + 1], delay[
              iterator]);
101   }
102
103   delay[iterator] = block.getSample(0, i);
104
105   block.setSample(0, i, out);
106
107   iterator++;
108   if (iterator == delaySize) iterator = 0;
109
110   if (delayTimeTarget != delayTime) delayTime =
          delayTimeTarget;
111 }
```

3.2.7 MasterChannel

Objekt `MasterChannel` představuje hlavní stopu skupiny. Fungování hlavní stopy již bylo popsáno v části 2.4.2.

3.2.8 Meter

Objekt `Meter` představuje digitální měřič. Fungování měřiče již bylo v části 2.4.1 stručně popsáno.

Hlavní funkcí objektu je zobrazit úroveň digitálního signálu v grafické podobě. Využívá se k tomu třída `Timer`, která funguje jako časovač. `Timer` v tomto objektu je spuštěn s frekvencí 30 Hz a po vypršení časovače se volá funkce `timerCallback`. Tato funkce překresluje objekt, v závislosti na tom, jaké měly hodnoty vzorky v předchozím časovém intervalu. Když v intervalu od posledního volání funkce je největší hodnota vzorků větší, než hodnota, která je právě vykreslena, pak se tato hodnota aktualizuje. Když v intervalu jsou jen menší hodnoty vzorků, tak vykreslená hodnota se zmenší o konstantu při každém vykreslení. Tím bylo docíleno nepřetržité vykreslování hodnot.

Další zajímavá funkce objektu je funkce `scale` (výpis 3.7), která byla implementována pro škálování hodnot vzorků v dBFS na pixely a využívá se v celém objektu při vykreslování.

Výpis 3.7: Funkce pro škálování v objektu `Meter`.

```
159 //Scaling algorithm
160 float Meter::scale(float x, float minX, float maxX, float
       minY, float maxY) {
161   return (x - minX) / (maxX - minX) * (maxY - minY) + minY;
162 }
```

Škálování je řešeno pomocí poměrů popsaných rovnicí (3.1). Vstupní parametry funkce podle rovnice jsou hodnota x, kterou chceme škálovat, meze x_{min} a x_{max}, které vymezí interval starých hodnot a nové meze intervalu pro škálování y_{min} a y_{max}. Funkce vrací škálovanou hodnotu y.

$$\frac{y - y_{min}}{y_{max} - y_{min}} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

Po řešení této rovnice dostaneme y. Rovnice (3.2) byla použita ve funkci pro výpočet vrácené hodnoty.

$$y = \frac{(x - x_{min})(y_{max} - y_{min})}{x_{max} - x_{min}} + y_{min} \tag{3.2}$$

Objekt `Meter` disponuje také značením úrovně digitálního signálu v dBFS. Objekt byl navržen tak, aby při vyšších úrovních signálu bylo větší i rozlišení vykreslených hodnot.

3.2.9 NewTrack

Objekt `NewTrack` tvoří základ okna pro přidávání nových stop. Toto okno můžeme vidět na obrázku 2.9. Hlavní funkcí objektu je inicializace proměnných nových stop, které jsou ukládány v již zmíněné instanci `ValueTree`.

3.2.10 Routing

Objekt `Routing` slouží pro ukládání parametrů matice pro směrování a slučování signálu. Pouze výstupní stopy obsahují instanci tohoto objektu. Objekt ukládá identifikační čísla všech vstupů, které jsou směrovány do daného výstupu. Kromě toho je zde také uloženo, jestli aplikace má použít režim „post fader“ nebo režim „pre fader“ a zesílení signálu vstupu pro daný výstup. Pro odstranění vstupu nebo pro změnu režimu je vytvořeno nové vlákno pro „fadeování“ signálu, stejným způsobem jako již bylo popsáno u objektu `Equalizer` v podkapitole 3.2.1.

3.3 Midi

Skupina objektů `Midi` provádí všechny funkce aplikace, které jsou spjaty s protokolem MIDI.

3.3.1 MidiCommandManager

Hlavní funkce objektu `MidiCommandManager` je uložení aktivního vstupního MIDI zařízení a přijímání všech zpráv, které toto zařízení vyšle. Objekt obsahuje všechny funkce aplikace, které lze pomocí MIDI zpráv ovládat. Také obsahuje čísla MIDI kontrolerů přiřazená k těmto funkcím.

Pomocí vnořené třídy `IncomingMessageCallback` typu `CallbackMessage`, objekt `MidiCommandManager` přeposílá všechny přijaté MIDI zprávy objektu `MidiPerform`, který tyto zprávy zpracuje.

3.3.2 MidiLearn

Objekt `MidiLearn` tvoří základ okénka `MIDI Learn`. Fungování okna již bylo popsáno v části 2.8 této práce.

3.3.3 MidiPerform

Nejsložitější objekt této skupiny objektů je `MidiPerfom`, který zpracovává všechny MIDI zprávy aktivního vstupního MIDI zařízení.

Hlavní funkce aplikace je funkce `perform`, která vyhodnocuje MIDI zprávy. Celou funkci není možné prezentovat ve výpisu, jelikož má více než sto řádků, ale jako příklad si vezmeme část funkce, která volá další funkci `delay` podle MIDI zprávy. Tuto část funkce můžeme vidět ve výpisu 3.8. Na začátku funkce `perform` je z MIDI zprávy uloženo číslo kontroleru a následně podle tohoto čísla pomocí příkazu `switch` jsou ošetřeny jednotlivé případy.

Výpis 3.8: Část funkce `perform` objektu `MidiPerform`.

```
70    case Delay:
71      delay(message.getChannel(), message.getControllerValue()
          );
72      break;
```

Tato část funkce `perform` posílá funkci `delay` přes argumenty MIDI kanál, ze kterého dostaneme identifikační číslo stopy a hodnotu MIDI kontroleru.

Výpis 3.9: Funkce `delay` objektu `MidiPerform`.

```
220 void MidiPerform::delay(int id, int value) {
221   id += trackOffset - 1;
222   float normalised = linScale(value, 0, 127, 0, 1);
223   NormalisableRange<float> delayRange(DELAY_MIN_VALUE,
        DELAY_MAX_VALUE);
224   delayRange.skew = DELAY_SKEW_VALUE;
225   float delay = delayRange.convertFrom0to1(normalised);
226   if (isId(id)) {
227     String name = tree.getChildWithName(getType()).getChild(
          id).getProperty("name");
228     undoManager->beginNewTransaction("- Delay " + name);
229     tree.getChildWithName(getType()).getChild(id).setProperty
          ("delay", delay, undoManager);
230   }
231 }
```

Na začátku funkce `delay` po úpravách čísla MIDI kanálu dostaneme identifikační číslo stopy. Hodnota kontroleru je škálována 0 až 1 pomocí algoritmu, který již byl popsán rovnicí (3.2) v části 3.2.8. Následně je vytvořena instance třídy `NormalisableRange` knihovny JUCE, pomocí které lze získat hodnotu zpoždění pro škálovanou hodnotu MIDI kontroleru. Funkce přepíše hodnotu zpoždění stopy v instanci objektu `ValueTree`. Ostatní objekty aplikace pak reagují na změnu hodnoty zpoždění.

3.3.4 MidiSetup

Objekt `MidiSetup` představuje základ okna pro upravování MIDI nastavení. Fungování tohoto okna již bylo popsáno v části 2.8.

3.4 Source

Do skupiny objektů `Source` patří všechny objekty aplikace, které tvoří hlavní část programu a objekty, které nespadají do jiné skupiny s přesnějším zaměřením.

3.4.1 About

Objekt `About` tvoří základ okna `About`, které jsme mohli vidět na obrázku 2.15. Objekt vykresluje logo a banner aplikace pomocí binárních dat, popsaných v části 3.1 této práce.

3.4.2 Commands

Objekt `Commands` obsahuje všechny funkce aplikace, které lze volat pomocí klávesových zkratek. Po zadání příkazu v aplikaci funkce `perform` tohoto objektu provede volaný příkaz. Jsou zde také uloženy názvy všech funkcí, které pak jsou vypisovány v hlavním menu aplikace. Objekt je velice rozsáhlý a složitý.

3.4.3 Constants.h

Soubor `Constants.h` obsahuje všechny globální proměnné aplikace.

3.4.4 DeviceSelector

Objekt `DeviceSelector` je základem okna pro výběr zvukového zařízení, které již bylo popsáno v podkapitole 2.3.

3.4.5 Main

Objekt `Main` je základním objektem aplikace. Tento objekt je vytvořen ze třídy `JUCEApplication` a provádí základní operace aplikace. Kromě toho objekt provádí inicializaci objektu `ValueTree`, který slouží pro ukládání všech dat aplikace. Objekt je také využit pro vytvoření nových a otevírání vybraných projektů nebo pro změnu režimu aplikace z tmavé na světlé a obráceně. Jsou zde inicializována a načtena všechna potřebná data pro fungování aplikace.

3.4.6 Matrix

Objekt `Matrix` je základem matice pro směrování a slučování vstupních signálů do výstupů. Fungování matice již bylo popsáno v části 2.7.

Zajímavé řešení v objektu je pro zaškrtávací políčka. Pro tato políčka byla vytvořena nová vnitřní třída `MatrixButton` ze třídy `ToggleButton` knihovny JUCE. Třída také využívá třídu časovače knihovny `Timer`. Část třídy můžeme vidět ve výpisu 3.10.

Výpis 3.10: Část třídy `MatrixButton` objektu `Matrix`.

```
47 //Mouse Listener
48 void mouseEnter(const MouseEvent& event) override {
49   if(!showed) startTimer(MATRIX_BOX_TIMER_MS);
50 }
51
52 void mouseExit(const MouseEvent& event) override {
53   showed = false;
54   stopTimer();
55 }
56
57 //Timer Callback
58 void timerCallback() override {
59   Point<int> xy = getMouseXYRelative();
60   if (getToggleState() && box->getLocalBounds().contains(xy))
         {
61     box->showBox(getScreenBounds());
62     showed = true;
63     stopTimer();
64   }
65 }
```

Časovač třídy je spuštěn, když ukazatel myši vstoupí mezi souřadnice políčka. Funkce zpětného volání časovače zkoumá jestli políčko je zaškrtnuté a jestli ukazatel myši ještě stále je nad políčkem, a když obě podmínky jsou pravdivé, otevře se malé okénko pro nastavování dalších parametrů, které bylo popsáno v části 2.7 a mohli jsme ho vidět na obrázku 2.20. Časovač je vypnut, když se objeví malé okénko nebo když ukazatel myši opustí políčko.

3.4.7 MatrixBox

Objekt `MatrixBox` tvoří okénko pro nastavení dalších parametrů směrování vstupních signálů, které bylo popsáno v podkapitole 2.7. Objekt také využívá časovač knihovny JUCE. Funkci zpětného volání časovače můžeme vidět ve výpisu 3.11. Časovač je spuštěn po zobrazení malého okénka.

Výpis 3.11: Funke `timerCallback` objektu `MatrixBox`.

```
89 void MatrixBox::timerCallback() {
90   Point<int> xy = getMouseXYRelative();
91   if (!box->getLocalBounds().contains(xy)) {
92     box->dismiss();
93     stopTimer();
94   }
95 }
```

Funkce `timerCallback` objektu `MatrixBox` zkoumá jestli ukazatel myši je stále nad objektem, a když ukazatel objekt opustí, okénko je zavřeno a časovač zastaven.

3.4.8 Menu

Objekt `Menu` tvoří základ hlavního menu aplikace, které bylo velice podrobně popsáno v podkapitole 2.4.3.

3.4.9 Project

Hlavním objektem aplikace je objekt `Project`, který tvoří základ hlavního okna aplikace. Fungování hlavního okna aplikace již bylo popsáno v podkapitole 2.4. Objekt obsahuje všechny proměnné a instance objektů pro daný projekt. Fungování objektu je velice složité a implementace objektu je velice rozsáhlá. Pro porozumění fungování objektu je možné se podívat na zdrojový kód v projektu aplikace na přiloženém disku. Objekt kvůli komplexitě není možné prezentovat pomocí výpisů.

3.4.10 Startup

Objekt `Startup` tvoří základ uvítacího okna, které bylo popsáno v části 2.2 diplomové práce.

3.5 Windows

Poslední skupina objektů je skupina `Windows`, která obsahuje všechny objekty pro samotná okna aplikace. Tyto objekty jsou `AboutWindow`, `DeviceSelectorWindow`, `EqualizerWindow`, `KeyMappingWindow`, `MatrixWindow`, `MidiLearnWindow`, `MidiSetupWindow`, `NewTrackWindow`, `ProjectWindow` a `StartupWindow`. Objekty oken nebudou podrobněji popisovány. Skupina také obsahuje soubor `Windows.h`, který má implementované funkce pro ukládání a načítání velikostí a umístění oken ze souboru `Windows.settings`.

Závěr

Diplomová práce v první kapitole popisuje teoretické znalosti, které byly potřebné k vytvoření aplikace podle zadání. Jsou zde popsány hlavně základy číslicového zpracování signálů, práce s knihovnou JUCE, která byla použita při implementaci aplikace, a základy protokolu MIDI. Byly zde také popsány postupy operací pro digitální audio signály, jako například inverze signálu, zpoždění signálu a lineární interpolace vzorků signálu.

Těžiště práce je v druhé kapitole, která velice detailně popisuje implementovanou aplikaci. Jsou zde popisy všech oken aplikace a také popis všech položek hlavního menu. Tato kapitola diplomové práce slouží vlastně jako návod pro porozumění všech funkcí aplikace a také pro porozumění obsluhy aplikace. Všechny důležité části grafického uživatelského rozhraní programu můžeme najít i na obrázcích.

Poslední část diplomové práce popisuje všechny objekty aplikace a některá zajímavější programátorská řešení problematických částí programu. Některé části programu jsou také představeny pomocí výpisů. Implementace a vnitřní funkce všech objektů nebylo možné do diplomové práci napsat kvůli komplexitě a délce zdrojového kódu aplikace. Celý projekt lze ale najít na přiloženém disku ve složce `SoFAA`.

Cílem diplomové práce byla implementace hotového programu podle zadání, který umožňuje provádět v reálném čase základní úpravy signálu (funkci *mute*, *inverzi* signálu, *zpoždění* signálu), disponuje parametrickým ekvalizérem, maticí pro slučování signálů libovolných vstupů a jejich směrování do jednoho nebo více libovolných výstupů s funkcí soft-fade, podporuje technologii ASIO, lze ho ovládat pomocí MIDI zařízení a umožňuje ukládání a vyvolání projektů. Aplikace, která byla implementována, splňuje všechny zadáním stanovené požadavky. Cíl diplomové práce byl splněn.

Literatura

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

[2] MIŠUREC, Jiří a Zdeněk SMÉKAL. *Číslicové zpracování signálů*. Brno: VUT v Brně. Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací 2012, 187 s. ISBN 978-80-214-4448-5.

[3] 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.

[4] SMITH, Julius O. *Physical Audio Signal Processing: for Virtual Musical Instruments and Digital Audio Effects.* W3K Publishing, 2010, xxii, 803 s. ISBN 978-0-9745607-2-4.

[5] *C++* [online]. [cit. 2019-11-21].
Dostupné z: <`https://cs.wikipedia.org/wiki/C%2B%2B`>.

[6] STROUSTRUP, Bjarne. *C++ programming language.* Massachusetts: Addison-Wesley, 1997, 910 s. ISBN 0-201-88954-4.

[7] *cplusplus.com – The C++ Resources Network* [online]. [cit. 2020-04-17].
Dostupné z: <`https://www.cplusplus.com/`>.

[8] *JUCE* [online]. [cit. 2019-11-21].
Dostupné z: <`https://en.wikipedia.org/wiki/JUCE`>.

[9] *Documentation* | JUCE [online]. [cit. 2019-11-21].
Dostupné z: <`https://juce.com/learn/documentation`>.

[10] *Tutorials* | JUCE [online]. [cit. 2019-11-21].
Dostupné z: <`https://juce.com/learn/tutorials`>.

[11] *Get JUCE* | JUCE [online]. [cit. 2019-11-21].
Dostupné z: <`https://shop.juce.com/get-juce`>.

[12] ROBINSON, Martin. *Getting started with JUCE.* Packt Publishing Ltd, 2013, 145 s. ISBN 978-1-78328-331-6.

[13] *Developers* | Steinberg [online]. [cit. 2019-11-22].
Dostupné z: <`https://www.steinberg.net/en/company/developers.html`>.

[14] The MIDI Manufacturers Association. *The Complete MIDI 1.0 Detailed Specification* – document version 96.1, third edition. Los Angeles, CA, 1996, 334 s.
Dostupné z: <https://www.midi.org/specifications-old/item/the-midi-1-0-specification>.

[15] *Microsoft Visual Studio Installer Projects* – Visual Studio Marketplace [online]. [cit. 2020-04-18].
Dostupné z: <https://marketplace.visualstudio.com/items?itemName=VisualStudioClient.MicrosoftVisualStudio2017InstallerProjects>.

Seznam symbolů, veličin a zkratek

ASIO	Audio Stream Input/Output
BPF	Band-pass filter
BPF	Band-stop filter
DP	Dolní propust
DSP	Digital Signal Processing – číslicové zpracování signálů
EOX	End of Exclusive (Midi Message)
FIR	Finite Impulse Response - konečná impulsní charakteristika
GUI	Graphical User Interface – grafické uživatelské rozhraní
HP	Horní propust
HPF	High-pass filter
IDE	Integrated Development Environment – vývojové prostředí
IIR	Infinite Impulse Response - nekonečná impulsní charakteristika
JUCE	Jules' Utility Class Extensions
LTI	Linear Time-Invariant – lineární časově invariantní
LPF	Low-pass filter
MIDI	Musical Instrument Digital Interface
PA	Public Address – ozvučování
PP	Pásmová propust
PZ	Pásmová zádrž
SDK	Software Development Kit
SVG	Scalable Vector Graphics
SysEx	System Exclusive (Midi Message)