TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Digitální mixpult na zvukové kartě

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Daniel Louda**
Vedoucí práce: Ing. Michal Rott

Liberec 2016

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Digital sound card mixer

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Daniel Louda**
Supervisor: Ing. Michal Rott

Liberec 2016

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2015/2016**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Daniel Louda**
Osobní číslo:	**M12000362**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Digitální mixpult na zvukové kartě**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**

Zásady pro vypracování:

1. Obeznámení se s problematikou zpracování akustických dat v reálném čase na zvukové kartě.
2. Vytvoření aplikace, která umožní nahrávání dat ze zvukové karty.
3. Do aplikace implementovat funkce umožňující mixování zvukových stop a jejich základní úpravy (korekce hlasitosti, ekvalizace,...).
4. Ověřit funkčnost aplikace na standardní zvukové kartě i na kartě s nízkou odezvou např. M-AUDIO ProFire 2626.

Rozsah grafických prací:	**Dle potřeby dokumentace**
Rozsah pracovní zprávy:	**cca 30 - 40 stran**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **B. Porat, "A Course in Digital Signal Processing", John Wiley & Sons, 1997.**

[2] **Glen Ballou, "Filters and equalizers", Handbook for Sound Engineers, Fourth edition, Focal Press, 2008 ISBN 0-240-80969-6**

Vedoucí bakalářské práce:	**Ing. Michal Rott** Ústav informačních technologií a elektroniky
Konzultant bakalářské práce:	**Ing. Marek Boháč** Ústav informačních technologií a elektroniky
Datum zadání bakalářské práce:	**14. září 2015**
Termín odevzdání bakalářské práce:	**16. května 2016**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 -

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
vedoucí ústavu

V Liberci dne 14. září 2015

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 16.5.2016

Podpis:

Abstrakt

V době kdy téměř každá společnost a domácnost disponuje osobními počítači nebo laptopy přišel čas začít přemýšlet, jaká zařízení lze tímto univerzálním nástrojem nahradit. Jedna z těchto věcí je mixážní pult, což je zařízení široce využívané nejenom v hudebním průmyslu, i ale kdekoli, kde je potřeba míchání zvukových stop či jejich korekce. Pro tuto práci bylo nastudováno z odborné literatury zpracování signálů na počítači s důrazem na minimální zpoždění. Jsou zde rozebrány různé typy filtrů a návrh digitálních filtrů, vycházející z jejich analogových ekvivalentů. Výsledkem práce je program napsaný v programovacím jazyce C++ s grafickým uživatelským rozhraním, který implementuje funkce mixážních pultů.

Klíčová slova: mixážní pult, dsp, digitální filtry, portaudio, bikvadratický filtr, návrh filtru

Abstract

We live in a time when personal computers have became essential for any establishment thus we can start thinking of new ways of using this universal device. One of them could be a digital sound mixer. A device frequently used whenever sound mixing or signal filtering is required. In my thesis I have described fundamentals of digital signal processing with focus on low latency systems and covered filtering and digital filter design using bilinear transform. All of this resulting in a C++ application with a graphical user interface simulating analog sound console on a personal computer.

Keywords: mixing console, dsp, digital filters, portaudio, biquadratic filter, filter design

Poděkování

Chtěl bych poděkovat všem lidem, kteří se podíleli na vytvoření této práce, především Michalovi Rottovi za vedení a cenné rady a ústavu ITE za zázemí a pomoc při vývoji.

Seznam obrázků

Seznam zkratek

SOS	Second order series
PC	Personal computer
IIR	Infinite impulse response
FIR	Finite impulse response
HW	Hardware
DAW	Digital audio workstation
AD	Analogově digitální
DA	Digitálně analogový
DSP	Digitální signálový procesor
API	Rozhraní pro programování aplikací
JACK	JACK Audio Connection Kit
ALSA	Advanced Linux Sound Architecture
OSS	Open Sound System
PA	PortAudio
GTK+	GIMP Toolkit
LGPL	GNU Lesser General Public License
GUI	Grafické uživatelské rozhraní
IDE	Vývojové prostředí (Integrated Development Environment)
WYSIWYG	„What you see is what you get“

Obsah

1 Úvod

Tato práce simuluje analogový mixážní pult na PC a řeší tedy problém souběžného zpracování vícestopého zvukového signálu na digitální zvukové kartě. Cílem práce je vytvořit program, který uživateli umožní provést korekce vstupů jako je hlasitost, vyvážení a ekvalizace, jejich sloučení a přehrávání v reálném čase. Problém byl řešen za pomoci programovacího jazyka C++, knihovny pro vytváření grafických uživatelských rozhraních Qt a open source knihovny pro zpracování zvuku PortAudio. Jako filtr byla implementována série IIR filtrů druhého řádu (SOS). Výsledkem je funkční program pro operační systémy typu Linux, který je jednoduchý na použití, má malé hardwarové nároky a obsahuje všechny běžné funkce analogových mixážních pultů.

1.1 Cíle

Cílem této bakalářské práce je napsat program, inspirovaný analogovým zvukovým mixážním pultem, který bude schopný převést jeho vlastnosti do digitálního prostředí počítačů. Tento program by měl pracovat v reálném čase (s minimálním zpožděním), aby jím bylo možné v případě potřeby plně nahradit klasický analogový pult.

Digitální mixážní pult by měl podporovat následující funkce:

- čtení dat ze zvukové karty (vstup mikrofonu, line in),
- korekci hlasitosti, vyrovnaní stereo kanálu a ekvalizace signálu,
- sloučení vstupních zařízení,
- přehrávání výsledného signálu na vybraném výstupním zařízení,
- ovládání pomocí grafického uživatelského rozhraní.

Osobní požadavky na aplikaci jsou:

- minimální HW nároky,
- zachování jednoduchosti ovládání analogového pultu,
- multiplatformnost.

1.2 Motivace

Zpracování zvukových dat v reálném čase je zajímavé téma pro studenta informačních technologii, neboť je zde nutné skloubit znalosti signálu a informací s návrhem softwaru, algoritmizace a využití knihoven třetích stran. Konkrétně pro tuto práci, nazvanou digitální mixážní pult na zvukové kartě, jsou všechny výše uvedené dílčí části kritické.

1.3 Pozice na trhu

Množina potenciálních konkurenčních programů není tak velká vzhledem k přístupu k problematice zpracování signálů. Nejčastěji se setkáváme se základním softwarem, který je obvykle součástí operačních systémů a běžně neumožňuje nic víc než výběr vstupního zařízení a korekci hlasitosti. Druhou skupinu tvoří komplexní řešení pro hudební producenty, například Ableton Live, FL Studio nebo Adobe Audition, které jsou však pro běžné potřeby finančně a hardwarově náročné. Programy zabývající se podobnou problematikou jsou často pojaty jako DJ pulty, jež se soustředí na přehrávání a korekce dvou stop. Cílem této práce proto bude vytvoření programu, který zůstane jednoduchý a přímočarý a současně bude podporovat zpracování více vstupních zařízení najednou s důrazem na minimální zpoždění.

2 Zpracování signálů

V této kapitole je popsán převod spojitého signálu na diskrétní, některé druhy filtrů a metoda návrhu digitálního filtru pomocí bilineární transformace.

2.1 Analogový mixážní pult

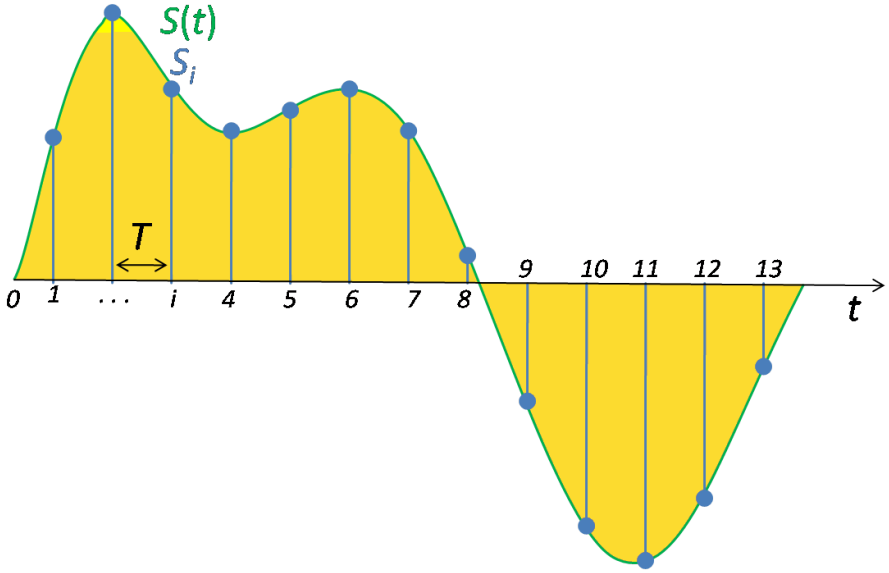
Analogový mixážní pult je zařízení, které vzniklo v 60. letech 20. století v reakci na vytvoření prvního vícevstupého kazetového nahrávače a umožňovalo korekce jednotlivých stop po nahrání (na počátku typicky čtyřstopý pult pro bicí, kytaru, basu a zpěv). Od té doby bylo hojně využíváno v hudebním průmyslu a postupem času se funkce mixážního pultu rozšířily podle potřeb muzikantů a producentů. Přibyl například počet vstupů, ekvalizace a smyčka pro efekty. V 90. letech se začaly používat digitálni verze mixážních pultů, ve kterých analogové obvody nahradily mikropočítače. V dnešní době lidé, kteří hledající retro zvuk, stále využívají analogové mixážní pulty či jejich kombinace s digitálními audio stanicemi, takzvané DAW.

2.2 Kritérium reálného času

Reálný čas zde chápeme jako zpoždění programu dostatečně malé tak, aby jej lidské ucho nezaznamenalo. Je dán zpožděním analogově digitálního (A/D) převodníku, výpočty procesoru a digitálně analogového (D/A) převodníku. Zpoždění převodníku je dáno hardwarem (HW) a nelze jej do větší míry ovlivnit, na rozdíl od zpoždění výpočtu, které lze minimalizovat efektivním kódem. Celkové zpoždění systému se tedy rovná součtu zpoždění třech výše zmíněných složek.

2.3 A/D a D/A převodníky

A/D a D/A převodníky jsou elektronické součástky, které dokáží převést spojitý analogový signál na diskrétní a naopak. Tyto signály jsou číslicové počítače schopny zpracovat. Převodníky bývají zpravidla součástí zvukové karty, v případě integrované verze jsou přímo součástí základní desky. Převod probíhá ve dvou krocích, jsou jimi vzorkování a kvantování.



Obrázek 2.1: Vzorkování spojitého signálu[12].

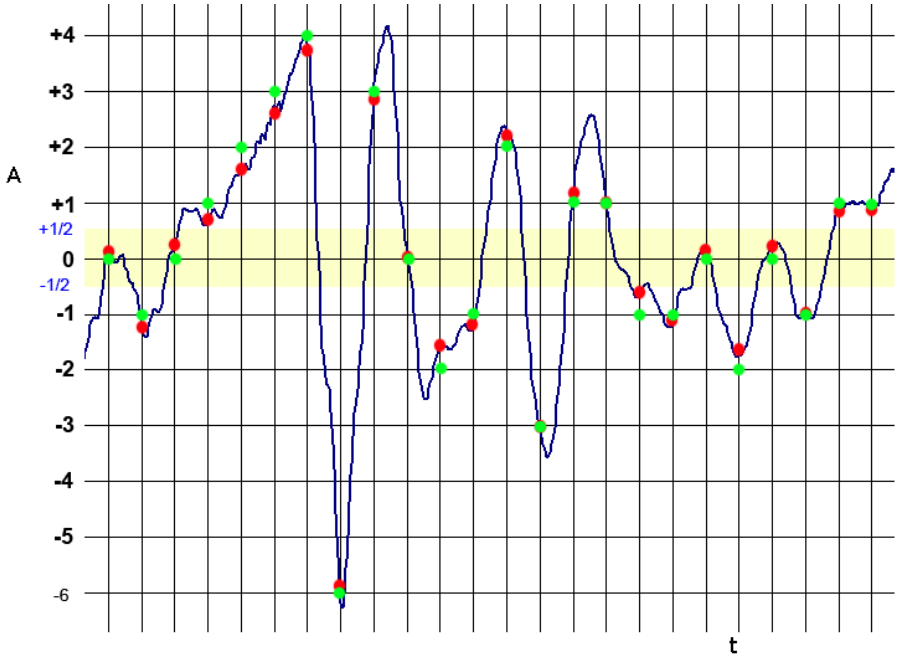
2.3.1 Vzorkování

Vzhledem k tomu, že mikročipy mají omezený výpočetní výkon a paměť, je nejprve nutno spojitý signál převést na konečný počet digitálních informací, se kterými dokážeme pracovat. Vzorkování je pravidelné odebírání vzorků spojitého signálu, dané vzorkovací frekvencí. Na obrázku 2.1 můžeme vidět modré vzorky ze zeleného spojitého signálu.

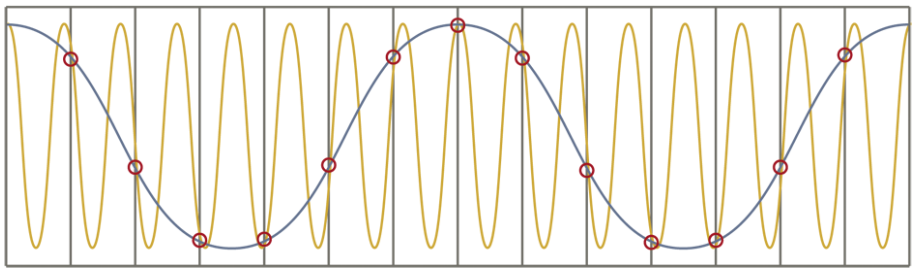
Aliasing je chyba vycházející ze Shanonova teorému, kterou způsobuje výskyt frekvencí vyšších než je 1/2 vzorkovací frekvence. V případě existence vyšších frekvencí dochází k překrytí frekvenčních spekter vzorkovaného signálu, aliasingu (viz obrázek 2.3). Například pro 44100Hz vzorkovací frekvenci by mohlo dojít k nenávratnému zkreslení signálu při přítomnosti frekvence vyšší než 22050Hz. Běžně se proto využívá dolních propustí, aby se aliasingu předešlo.

2.3.2 Kvantování

Další fází převodu je kvantování. Jelikož počítače nemají neomezenou paměť, musíme pro zpracování spojitých signálů pracovat s omezenou přesností. Reprezentovat amplitudu vzorku signálu lze pouze konečnými čísly. Počet hodnot, kterých může vzorek nabývat, je dán počtem kvantizačních úrovní, typicky N–tá mocnina čísla 2. Tento postup může ovšem vést k chybě jejíž velikost je $-1/2$ až $1/2$ kvantizační úrovně. Na obrázku 2.2 můžeme vidět průběh kvantování a jeho nepřesnost ilustrovanou rozdílem zelených a červených bodů.



Obrázek 2.2: Kvantování vzorků signálu[11].



Obrázek 2.3: Zkreslení signálu způsobené aliasingem

2.4 Filtry

Filtrování je druh zpracování signálů. Filtr jako takový je zařízení nebo proces, který toto umožňuje. Jsou používané především k potlačení určité části signálu, například k odstranění nežádoucích frekvencí bez vlivu na ostatní frekvence. Mají využití i mimo frekvenční doménu, třeba pro zpracování obrazu. Filtry můžeme dělit různými způsoby – například podle linearity, obsažení zpětné vazby nebo zda pracují na spojitém či diskrétním signálu.

2.4.1 Lineární filtr

Aby byl filtr lineární musí platit aditivita a homogenita. Součet výstupů dvou signálů se tedy **musí** rovnat výstupu součtu těchto dvou signálů.

Za předpokladu:

$$x_1(t) \rightarrow y_1(t), x_2(t) \rightarrow y_2(t) \tag{2.1}$$

platí[7]:

$$ax_1(t) + bx_2(t) \rightarrow ay_1(t) + by_2(t) \tag{2.2}$$

Výstup nerekurzivního lineárního filtru se rovná konvoluci vstupního signálu s impulzní odezvovu filtru. Účelem lineární filtrace bývá obvykle potlačení nebo zvýraznění určitých spektrálních složek. Filtry lze popsat níže uvedenými způsoby.

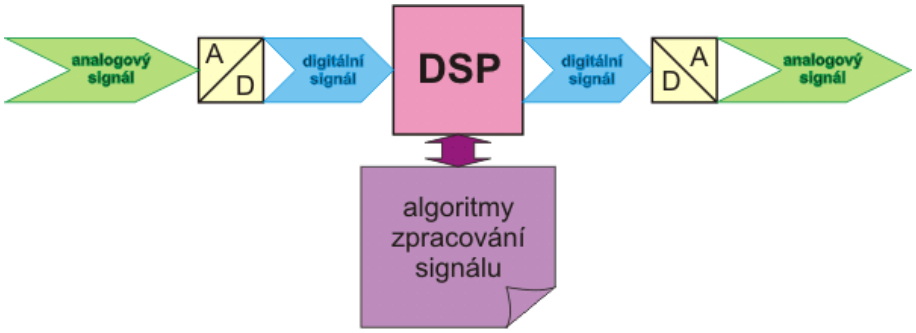
- Diferenční rovnicí,
- impulzní odezvou,
- frekvenční charakteristikou,
- přenosovou funkcí.

2.4.2 Digitální filtry

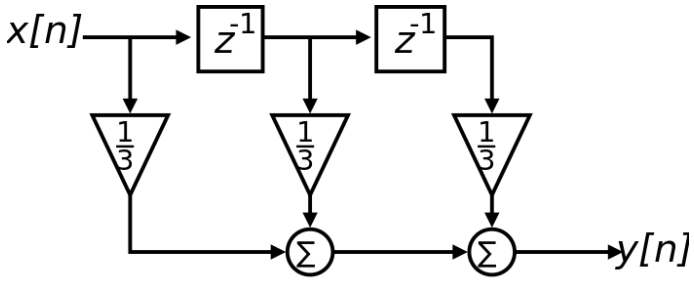
Digitální filtr je matematický algoritmus implementovaný hardwarem, firmwarem anebo softwarem, který provádí výpočty na diskrétním vstupním signálu tak, aby vytvořil diskrétní výstup vyhovující požadavkům[1]. Velkou výhodou softwarově realizovaných filtrů je možnost modifikovat jejich funkci pouhou úpravou kódu bez nutnosti změny hardwaru. Typický řetězec zpracování signálu je vidět na obrázku 2.4.

2.4.3 Filtr s konečnou impulzní odezvou

Filtry s konečnou impulzní odezvou, označovány také jako FIR (finite impulse response), jsou filtry, jejichž impulzní odezva je po konečném čase nulová. FIR filtry mohou být analogové nebo digitální. Ukázku FIR filtru druhého řádu je možné vidět na obrázku 2.5.



Obrázek 2.4: Typický řetězec pro zpracování signálu[10].



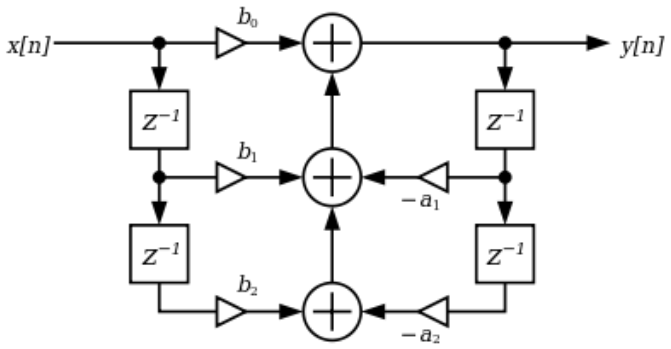
Obrázek 2.5: Blokový diagram FIR filtru 2. řádu[9].

Vlastnosti FIR filtrů[1]:

- jsou stabilní, pokud jsou póly v jednotkové kružnici z–roviny,
- neobsahují zpětnou vazbu,
- je potřeba vyšší řád FIR filtru k získání stejné charakteristiky v porovnání s IIR filtrem,
- delší odezva filtru je způsobená velikostí řádu,
- chyby způsobené konečnou přesností reprezentace dat nejsou tak závažné jako u IIR filtrů,
- širší přechodové pásmo ve srovnání s IIR stejného řádu,
- záruka možnosti návrhu filtru s lineární fází.

2.4.4 Filtr s nekonečnou impulzní odezvou

IIR (infinite impulse response) filtry, které mohou být analogové i diskrétní jsou charakterizované svou impulzní odezvou, která nikdy nebude rovna nule po konečném čase. Ukázka IIR filtru druhého řádu je vidět na obrázku 2.6.



Obrázek 2.6: Blokový diagram IIR filtru 2. řádu[8].

Vlastnosti IIR filtrů:

- stačí relativně nižší řád přenosové funkce k získání stejné charakteristiky v porovnání s FIR filtrem,
- menší zpoždění dané relativně menším řádem,
- kratší přechodové pásmo ve srovnání s FIR stejného řádu,
- rovné propustné a závěrné pásmo.

2.4.5 Návrh filtru použitím bilineární transformace (BLT)

BLT je metoda využívaná pro převod spojitého systému $H(s)$ na diskrétní systém $H(z)$ a naopak[1]. Této metody lze využít při návrhu digitálního filtru a to transformací přenosové funkce jeho analogového ekvivalentu podle vztahu:

$$s = \frac{2}{T}(\frac{z-1}{z+1}) = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} \tag{2.3}$$

Z přenosové funkce $H(s)$ analogového filtru je možné vytvořit přenosovou funkci odpovídajícího digitálního filtru následující substitucí kde T je vzorkovací perioda.

$$H(z) = H(s)\Big|_{s=\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}} \tag{2.4}$$

Představme si jednoduchý analogový filtr dolní propusti s následující přenosovou funkcí, na který aplikujeme BLT.

$$H(s) = \frac{1}{1+s} \tag{2.5}$$

$$H(z) = \frac{1}{1+\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}} \tag{2.6}$$

Po úpravě dostáváme požadovanou přenosovou funkci $H(z)$, kterou můžeme využít k implementaci digitálního filtru.

$$H(z) = \frac{1 + z^{-1}}{(1 + \frac{2}{T}) + (1 - \frac{2}{T})} = \frac{1 + z^{-1}}{2} \tag{2.7}$$

Frekvenční warping

Vzhledem k tomu, že $j\Omega$ osa s-roviny se přenese na jednotkovou kružnici z-roviny ($|z| = 1$) existuje zde přímý vztah mezi frekvencí Ω s-roviny a frekvencí w z-roviny[1]. Substitucí $s = j\Omega$ a $z = e^{jw}$ do rovnice 2.3, kde w je kritická frekvence $2\pi * f_0 * T$, dostáváme:

$$j\Omega = \frac{2}{T}(\frac{e^{jw} - 1}{e^{jw} + 1}) \tag{2.8}$$

Z této rovnice lze vyjádřit Ω, případně w.

$$\Omega = \frac{2}{T}\tan(w/2) \tag{2.9}$$

$$w = 2\tan^{-1}(\frac{\Omega T}{2}) \tag{2.10}$$

Můžeme tedy říct, že digitální filtr se chová k frekvenci w stejně jako analogový ekvivalent k frekvenci $\frac{2}{T}\tan(w/2)$. Tento nelineární vztah Ω k w označujeme jako takzvaný frekvenční warping, který musíme brát v potaz při návrhu digitálních filtrů metodou BLT. Pro návrh filtru metodou BLT s ohledem na frekvenční warping je potřeba provést následující tři kroky[1].

1. Upravíme kritickou frekvenci w_c digitálního filtru použitím vzorce 2.9 k získání odpovídající frekvence Ω_c analogového filtru.

2. Pomocí Ω_c naškálujem přenosovou funkci analogového filtru $H(s)$ abychom dostali
$$\hat{H}(s) = \hat{H}(s)\Big|_{s=s/\Omega_c} = H(\frac{s}{\Omega_c}) \tag{2.11}$$
kde $\hat{H}(s)$ je škálovaná přenosová funkce $H(s)$.

3. Nahrazením s v $\hat{H}(s)$ výrazem $2(z-1)/(z+1)T$ získáváme požadovanou přenosovou funkci digitální filtru $H(z)$.
$$H(z) = \hat{H}(s)\Big|_{s=\frac{2}{T}(\frac{z-1}{z+1})} \tag{2.12}$$

Tento postup lze zjednodušit do jednoho kroku dosazením

$$s \leftarrow \frac{1}{\tan(w_c/2)}(\frac{1 - z^{-1}}{1 + z^{-1}}) \tag{2.13}$$

kde $w_c = 2\pi f_0 T$ a f_0 je rohová nebo středová frekvence filtru.

V této kapitole byly některé výpočty zjednodušeny či zkráceny [1, 3, 6]. Tato metoda je podrobněji rozebrána v knize *Real-Time Digital Signal Processing* od Sena M. Kua a Boba H. Lee.

2.4.6 Bikvadratický filtr

Je digitální IIR filtr druhého řádu. Název je odvozen od jeho přenosové funkce v z-rovině, která je tvořena podílem dvou kvadratických funkcí. Při zachování pólů uvnitř levé poloviny jednotkové kružnice z-roviny je stabilní[1]. K realizaci IIR filtrů vyšších řádu se běžně používá série bikvadratických filtrů označovaná jako SOS. Jedná se o velmi flexibilní filtr, ze kterého je možné vyrobit různé typy filtrů pouhou změnou jeho koeficientů.

Přenosová funkce s normalizovanými koeficienty pro $a_0 = 1$.

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{2.14}$$

Implementace bikvadratického filtru

Z přenosové funkce nám vychází tato diferenční rovnice. Koeficienty b_0, b_1, b_2 určují pozici nul a a_1, a_2 určují pozici pólů.

$$y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] - a_1 y[n-1] - a_2 y[n-2] \tag{2.15}$$

Vzhledem k tomu, že většina analogových filtrů je typu IIR a byly již popsány v literatuře, můžeme využít těchto znalostí k návrhu digitálního filtru pomocí BLT[6, 2].

Přenosové funkce analogových filtrů druhého řádu

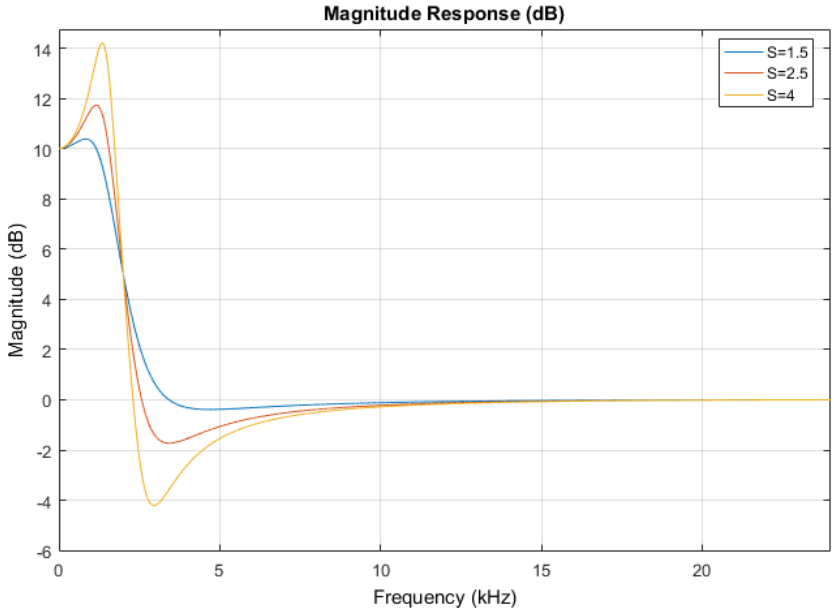
Níže uvedené funkce byly převzaty od Roberta Bristow-Johnsona[2].

Horní propust propouští vysoké frekvence a je využívaná pro oříznutí nízkých.

$$H(s) = \frac{s^2}{s^2 + \frac{s}{Q} + 1} \tag{2.16}$$

Dolní propust propouští nízké frekvence a je využívaná pro oříznutí vysokých.

$$H(s) = \frac{1}{s^2 + \frac{s}{Q} + 1} \tag{2.17}$$



Obrázek 2.7: Magnitudová odezva lowshelf filtru druhého řádu v závislosti na parametru S.

Highshelf propouští všechny frekvence, ale zesiluje nebo zeslabuje vše pod danou frekvencí o určité množství.

$$H(s) = A * \frac{A * s^2 + \sqrt[2]{\frac{A}{Q}} * s + 1}{s^2 + \sqrt[2]{\frac{A}{Q}} * s + A} \tag{2.18}$$

Lowshelf propouští všechny frekvence, ale zesiluje nebo zeslabuje vše nad danou frekvencí o určité množství.

$$H(s) = A * \frac{s^2 + \sqrt[2]{\frac{A}{Q}} * s + A}{A * s^2 + \sqrt[2]{\frac{A}{Q}} * s + 1} \tag{2.19}$$

Uživatelem definované parametry

- Fs – vzorkovací frekvence,
- f_0 – centrální frekvence,
- dBgain – zesílení/zeslabení shelf filtrů,
- Q – Q(quality) faktor, např. pro Butterworth filtr dolní propusti druhého řádu $Q = \frac{1}{\sqrt{2}}$,
- S – shelf slope – strmost přechodového pásma shelf filtrů.

Výpočet proměnných

Zesílení/zeslabení shelf filtrů,

$$A = \sqrt[2]{10^{dBgain/20}} \tag{2.20}$$

kritická frekvence

$$w_0 = \frac{2 * \pi * f_0}{Fs} \tag{2.21}$$

Bilineární transformace z s-roviny do z-roviny

Přenosové funkce digitálních filtrů jsou reprezentovány v z-rovině, zatímco funkce analogových filtrů v s-rovině. Je tedy třeba použít bilineární transformaci k převodu. Využity byli následující substituce.

$$\begin{gathered} 1 \leftarrow (1 + 2z^{-1} + z^{-2}) * (1 - cos(w_0)) \\ s \leftarrow (1 - z^{-2}) * sin(w_0) \\ s^2 \leftarrow (1 - 2z^{-1} + z^{-2}) * (1 + cos(w_0)) \\ 1 + s^2 \leftarrow 2 * (1 - 2cos(w_0) * z^{-1} + z^{-2}) \end{gathered}$$

Dosazením těchto výrazů do přenosové funkce analogových filtrů je možné postupně vypočítat koeficienty $b_0, b_1, b_2, a_0, a_1, a_2$ pro digitální bikvadratický filtr[2].

Koeficienty jednotlivých filtrů

$$\alpha = \frac{sin(w0)}{(2 * Q)} = \frac{sin(w_0)}{2} * \sqrt{(A + \frac{1}{A}) * (\frac{1}{S} - 1) + 2} \tag{2.22}$$

Horní propust

$$\begin{aligned} b_0 &= \frac{1 + cos(w_0)}{2} \\ b_1 &= -(1 + cos(w_0)) \\ b_2 &= \frac{1 + cos(w_0)}{2} \\ a_0 &= 1 + \alpha \\ a_1 &= -2 * cos(w_0) \\ a_2 &= 1 - \alpha \end{aligned} \tag{2.23}$$

Dolní propust

$$\begin{aligned} b_0 &= \frac{1 - cos(w_0)}{2} \\ b_1 &= 1 - cos(w_0) \\ b_2 &= \frac{1 - cos(w_0)}{2} \\ a_0 &= 1 + \alpha \\ a_1 &= -2 * cos(w_0) \\ a_2 &= 1 - \alpha \end{aligned} \tag{2.24}$$

Lowshelf

$$\begin{aligned} b_0 &= A * ((A+1) - (A-1) * cos(w_0) + 2 * \sqrt{A} * \alpha) \\ b_1 &= 2 * A * ((A-1) - (A+1) * cos(w_0)) \\ b_2 &= A * ((A+1) - (A-1) * cos(w_0) - 2 * \sqrt{A} * \alpha) \\ a_0 &= (A+1) + (A-1) * cos(w_0) + 2 * \sqrt{A} * \alpha \\ a_1 &= -2 * ((A-1) + (A+1) * cos(w_0)) \\ a_2 &= (A+1) + (A-1) * cos(w_0) - 2 * \sqrt{A} * \alpha \end{aligned} \tag{2.25}$$

Highshelf

$$\begin{aligned} b_0 &= A * ((A+1) + (A-1) * cos(w_0) + 2 * \sqrt{A} * \alpha) \\ b_1 &= -2 * A * ((A-1) + (A+1) * cos(w_0)) \\ b_2 &= A * ((A+1) + (A-1) * cos(w_0) - 2 * \sqrt{A} * \alpha) \\ a_0 &= (A+1) - (A-1) * cos(w_0) + 2 * \sqrt{A} * \alpha \\ a_1 &= 2 * ((A-1) - (A+1) * cos(w_0)) \\ a_2 &= (A+1) - (A-1) * cos(w_0) - 2 * \sqrt{A} * \alpha \end{aligned} \tag{2.26}$$

2.4.7 Hlasitostní filtr

Korekce hlasitosti vstupu $y[n]$ v závislosti na konstantě K.

$$y[n] = Kx[n]$$

K > 1 filtr zesiluje signál.
$0 <$ K < 1 filtr zeslabuje signál.

2.4.8 Vyvažovací filtr

Filtr sloužící pro vyvážení hlasitosti kanálů stereo vstupu.
$y_L[n]$ levý kanál vstupu
$y_R[n]$ pravý kanál vstupu
$x_L[n]$ levý kanál výstupu

$x_R[n]$ pravý kanál výstupu

$$K \in (0, 1)$$

$$y_L[n] = \begin{cases} (1 - (2K - 1))x_L[n] & K > 0.5 \\ x_L[n] & K <= 0.5 \end{cases}$$

$$y_R[n] = \begin{cases} 2K * x_R[n] & K < 0.5 \\ x_R[n] & K >= 0.5 \end{cases}$$

3 Knihovny

Ke splnění požadavků aplikace bylo potřeba vybrat knihovnu pro vytvoření grafického rozhraní a audio knihovnu, usnadňující práci se zvukem.

3.1 Zvukové knihovny

Existují různá řešení pro různé potřeby, ze kterých lze vybírat. V této práci byl důraz kladen především na nízkou latenci a možnost programování na nízké úrovni.

3.1.1 Qt Multimedia

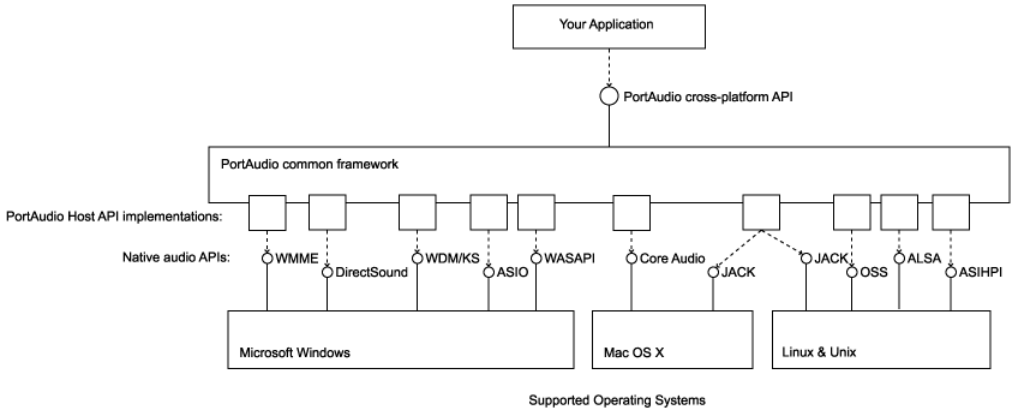
Tato knihovna je součástí Qt balíčku, který obsahuje mnoho tříd pro práci se zvukem. Lze zde například bez problému přehrát kompresovaný soubor nebo pracovat s hudebními soubory na síti. Jednoduchost použití je zde ovšem vykoupena vetší latencí.

3.1.2 PortAudio

PortAudio, součást projektu PortMedia (dříve PortMusic), je multiplatformní zvuková knihovna distribuovaná pod MIT licencí. Je podporována na nejrozšířenějších operačních systémech tj. Windows, Mac OS X a Linux. Podporuje tedy mnohé nativní API operačních systémů jako jsou DirectSound pro Windows, Core Audio a JACK pro Mac OS X, či ALSA, nebo OSS pro Linux.

PortAudio je napsáno v programovacím jazyce C, ale dá se využít i v jiných jako je Python (PyAudio) či PureBasic. PortAudio je založeno na callback paradigmatu, v němž každá audio I/O akce zavolá požadovanou callback funkci. Například pokud jsou vyžádána data na výstup nebo načtena data ze vstupu. Od verze V19 je navíc možné využít vedle callbacků i blokující read/write funkce, která sice bývá jednodušší na pochopení a implementaci, avšak ztrácí na rychlosti a přenositelnosti napříč API. Následuje ukázka callback funkce s komentářem z dokumentace PA[14], který generuje dvě vlny pro levý a pravý kanál s rozdílnou frekvencí.

```
typedef struct
{
    float left_phase;
    float right_phase;
}
```



Obrázek 3.1: Diagram vztahů mezi PA a zvukovými API[13].

```
paTestData;

static int patestCallback( const void *inputBuffer, void *outputBuffer,
                           unsigned long framesPerBuffer,
                           const PaStreamCallbackTimeInfo* timeInfo,
                           PaStreamCallbackFlags statusFlags,
                           void *userData )
{
    /* Cast data passed through stream to our structure. */
    paTestData *data = (paTestData*)userData;
    float *out = (float*)outputBuffer;
    unsigned int i;
    (void) inputBuffer; /* Prevent unused variable warning. */

    for( i=0; i<framesPerBuffer; i++ )
    {
        *out++ = data->left_phase; /* left */
        *out++ = data->right_phase; /* right */
        /* Generate simple sawtooth phaser that ranges between -1.0 and
            1.0. */
        data->left_phase += 0.01f;
        /* When signal reaches top, drop back down. */
        if( data->left_phase >= 1.0f ) data->left_phase -= 2.0f;
        /* higher pitch so we can distinguish left and right. */
        data->right_phase += 0.03f;
        if( data->right_phase >= 1.0f ) data->right_phase -= 2.0f;
    }
    return 0;
}
```

PortAudio pracuje se zvuky jako s proudy dat. Pro úspěšné otevření proudu je třeba definovat několik parametrů. Komentovaný exemplární kód níže z dokumentace PA[14] demonstruje otevření proudu bez vstupního zařízení a defaultním stereo vý-

stupem. Vzorky jsou typu **paFloat32** (+1.0, -1.0), vzorkovací frekvence je 44100Hz a buffer obsahuje 256 vzorků.

```
\label{code:pa_open}
#define SAMPLE_RATE (44100)
static paTestData data;
.....
    PaStream *stream;
    PaError err;
    /* Open an audio I/O stream. */
    err = Pa_OpenDefaultStream( &stream,
                                0,          /* no input channels */
                                2,          /* stereo output */
                                paFloat32,  /* 32 bit floating point output */
                                SAMPLE_RATE,
                                256,        /* frames per buffer, i.e. the
                                    number
                                                of sample frames that
                                                    PortAudio will
                                                request from the callback.
                                                    Many apps
                                                may want to use
                                                paFramesPerBufferUnspecified,
                                                    which
                                                tells PortAudio to pick the
                                                    best,
                                                possibly changing, buffer
                                                    size.*/
                                patestCallback, /* this is your callback
                                    function */
                                &data ); /*This is a pointer that will be
                                    passed to
                                                your callback*/
    if( err != paNoError ) goto error;
```

3.2 Knihovny grafických rozhraní

Existuje několik možných knihoven pro grafická rozhraní. V této práci jsou zmíněna pouze dvě největší, použitelné pro operační systémy (nejen) Linuxového typu. Oba toolkity jsou dobře otestované a mají obdobné nároky na paměť a výpočetní výkon.

3.2.1 GTK+

GTK+ je knihovna pro grafická rozhraní, napsaná v jazyce C, původně vznikla v roce 1997 pro vývoj aplikace na tvorbu a úpravu rastrové grafiky GIMP. Od té doby pokračovala ve vývoji, nabyla na popularitě a stala se tak jednou z nejpoužívanějších

toolkitů především pro operační systémy typu Linux díky cílení na X server a otevřené licenci LGPL. Je využívaná desktopovým prostředím GNOME a podporuje díky wrapperům i mnoho jiných programovacích jazyků například C++, Javascript, Perl či Python.

3.2.2 Qt

Qt je jeden z nejpopulárnějších GUI frameworků napříč operačními systémy. Původní projekt společnosti Trolltech koupila v roce 2008 firma Nokia a od té doby se stará o jeho vývoj (poslední verze 5.5). Toolkit je napsán v programovacím jazyce C++, ale je možné ho využívat i pro jiné jazyky jako Python (PyQt), Ruby (QtRuby) nebo C. Na Qt je například postavené Linuxové desktopové prostředí KDE včetně velkého množství jeho nativních aplikací, multiplatformní programy jako Skype nebo Google Earth. Qt převyšuje GTK+ svým rozsahem dokumentace a počtem aktivních vývojářů.

Qt Creator

Qt Creator je multiplatformní IDE (vývojové prostředí) pro vývoj softwaru v C++ nebo Javascriptu. Umožňuje využití různých kompilerů, zvýraznění syntexe nebo verzovacích systémů. Obsahuje debugger a integrovaný nástroj pro vývoj WYSIWYG grafických rozhraních Qt Designer.

QObject

QObject je základní třída všech Qt objektů. Pokud ji objekty dědí, je možné využít funkci **connect** pro vytvoření vazeb mezi nimi.

Signály a sloty

Systém signálů a slotů je mechanismus pro komunikaci mezi objekty, který nahrazuje dříve používané callback funkce. Signál je vyslán, pokud dojde k určité události. Qt Widgety mívají několik předdefinovaných signálů (triggered, closed, pushed,…) a umožňují přidat vlastní. Slot je funkce, která je zavolána při reakci na konkrétní signál s danými parametry.
Ukázka připojení objektů[15].

```
connect(sender, SIGNAL (valueChanged(QString,QString)),
 receiver, SLOT (updateValue(QString)) );
```

Signály a slot v Qt 5

Ve verzi Qt 5 přibyla nová syntaxe propojení objektů předáním nestringových parametrů. Následující ukázky pocházejí z dokumentace Qt[15].

```
connect(sender, &Sender::valueChanged,
 receiver, &Receiver::updateValue );
```

Dále je možné nově spojit signál nejenom s QObjektem, ale také s funkcí.

```
connect(sender, &Sender::valueChanged, someFunction);
```

Vzhledem k podpoře standardu C++11 lze použít dokonce lambda výraz místo funkce.

```
connect(sender, &Sender::valueChanged, [=](const QString &newValue) {
 receiver->updateValue("senderValue", newValue);
 } );
```

QThread

QThread poskytuje nástroje pro práci s vlákny nezávisle na platformě. Mezi možnosti patří nastavení priority, startování, ukončení, uspání či přesun vláken. Nemělo by se zapomínat, že QThread je stále QObject a je tedy možné ho propojovat s jinými objekty mechanismem signálů a slotů.

4 Implementace aplikace

Pro vytvoření programu byl použit programovací jazyk C++ s využitím knihoven PortAudio a Qt. Jedná se o kombinaci, která je nezávislá na operačním systému a tudíž lze docílit multiplatformnosti. Byla vyvíjena v programu QtCreator (viz sekce 3.2.2), za použití verzovací služby GitLab. Aplikace byla následně testována na operačním systému Arch Linux.

C++11 je nejnovější standard schválený v roce 2011, který přidává prvky a částečně upravuje jádro kvůli zvýšení výkonu při zachování zpětné kompatibility. Jedna z novinek využita v této práci je lambda funkce v mechanismu signálů a slotů.

4.1 Třídy

Program byl strukturovaný do následujících tříd.

4.1.1 MainWindow

Je třída implementující QMainWindow, ve které je veškeré nastavení grafického rozhraní a zároveň jediná třída, která ho může měnit. Nejprve se vytvoří menu aplikace (obrázek 4.2), následuje vytvoření horizontálního grafického rozložení, které je poté naplněno čtyřmi panely (obrázek 4.1), jež jsou tvořeny mřížovým rozložením obsahující jednotlivé ovládací prvky programu. Dále se vytvoří panel výstupu a dojde k připojení všech přítomných ovládacích prvků mechanismem signálu a slotů (3.2.2) k požadovaným funkcím. Poté dochází k inicializaci PortAudia třídou PaHandler (4.1.2) a zapnutí časovače pro pravidelné aktulizace stavu jednotlivých vstupů. Na závěr dochází k nastavení defaultní velikosti okna, jeho pojmenování a vykreslení.

4.1.2 PaHandler

Tato třída se stará především o správnou inicializaci a ukončení PortAudia. Dále se zde nachází funkce pro překlad chybových hlášek a získávání seznamů vstupních i výstupních zařízení a jejich podrobnostech.

4.1.3 MyThread

Aby mohla část programu, která se stará o zvuk, pracovat ve vlastním vlákně, je zde implementovaná třída QThread (viz sekce 3.2.2). Ta má na starost vypínání a zapínaní jednotlivých proudů a jejich správu. Běží pouze v případě, že existuje alespoň jeden otevřený datový proud. Slouží jako komunikační most mezi grafickým rozhraním a samotnou logikou pro zpracování zvuku.

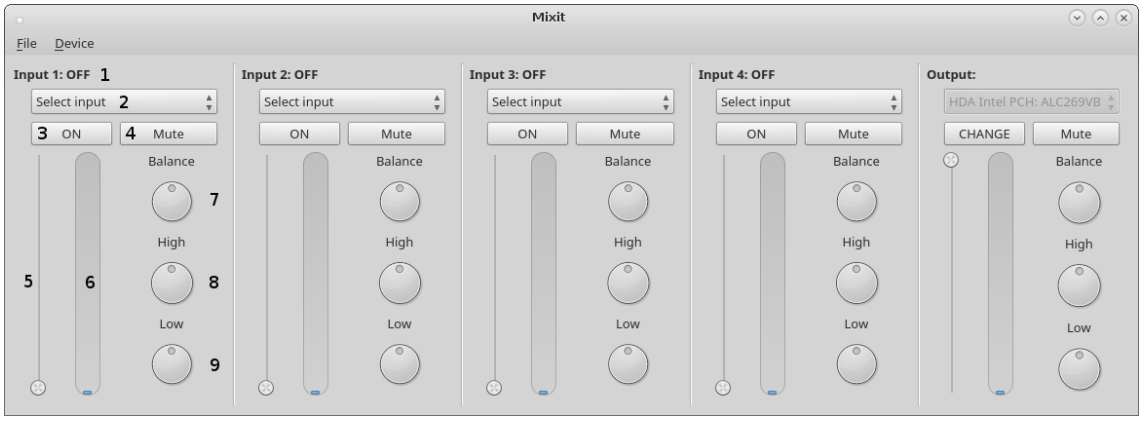
4.1.4 Wire

(ang. *wire* – drát, štěnice) Wire je třída sloužící pro přenesení zvuku ze vstupu na výstup a jeho korekce. Jsou zde funkce pro otevření (ukázka v sekci 3.1.2) a zavření proudu. Tyto funkce berou jako vstupní parametr index vstupního a výstupního zařízení. Nejprve zjistí, zda zařízení existují a jestli jsou kompatibilní se vzorkovací frekvencí. Pokud ano, dojde k nastavení nízkého vstupního a výstupního zpoždění a otevření proudu s danými parametry. Se samotným zvukem pracuje (zde přítomná) callback funkce (ukázka v sekci 3.1.2). Ta odesílá informace o hlasitosti do GUI a v případě, že není nastaveno mute, provede hlasitostní filtr (viz 2.4.7), vyvažovací filtr (viz 2.4.8) a ekvalizační filtr na vstupních datech a následně je přehraje. Hlasitostní a vyvažovací filtry jsou funkce, provádějící požadované matematické výpočty. Ekvalizační filtr je datový kontejner typu **std::vector<BiQuadFilter>**, realizující sérii bikvadratikých filtrů.

4.1.5 BiQuadFilter

BiQuadFilter je třída implementující bikvadratický filtr (popsán v sekci 2.4.6). Pro vytvoření této třídy je potřeba zadat několik parametrů: typ filtru (třída momentálně podporuje dolní propust, horní propust, lowshelf a highshelf), střední frekvenci, vzorkovací frekvenci, Q a zesílení v dB. Po nastavení uživatelských proměnných dojde k vypočtení několika parametrů, které jsou následně využity pro výpočet koeficientů bikvadratického filtru zadaného typu. Filtrace každého vzorku je pak realizovaná rovnicí 2.15. V aplikaci je využit highshelf a lowshelf filtr, jehož parametry jsou upraveny změnou polohy potenciometrů v grafickém rozhraní. Třída podporuje přidání vlastních parametrů bikvadratického filtru navrhnutých například v MATLABu, a tím tvorbu uživatelsky definovaného filtru.
Ukázka implementace rovnice 2.15 v jazyce C++.

```
float BiQuadFilter::filter(float x)
{
    y = b0 * x + b1 * x1 + b2 * x2 - a1 * y1 - a2 * y2;
    x2 = x1;
    x1 = x;
    y2 = y1;
    y1 = y;
    return (y);
}
```



Obrázek 4.1: Hlavní okno aplikace

4.2 Popis vlastností

4.2.1 GUI

Grafické rozhraní je vytvořeno pouze kódem a nikoliv WYSIWYG editorem. Je plně responzivní – při zvětšování a zmenšování se mění velikost okna i velikost objektů, avšak poměry zůstávají zachovány.

Hlavní okno

V hlavním okně se nachází všechny ovládací prvky aplikace. Jsou setříděny jako jednotlivé panely do horizontálního layoutu, přičemž vpravo je vždy výstup. Vstupní panely jsou naprogramovány **dynamicky** a jejich počet lze měnit za běhu aplikace podle potřeby.

Panel vstupu

1. Označení a stav – obsahuje informace o současném stavu panelu (on/off/mute) a označení vstupu,
2. select input – výběrové pole umožňující zvolení vstupu, obsahující položky všech dostupných, vstupních zařízeních.
3. on/off – zapnutí a vypnutí proudu,
4. mute – zapnutí a vypnutí hlasitosti v případě, že proud stále běží,
5. šoupátko – hlasitostní filtr,
6. indikátor hlasitosti – vizualizace hlasitosti vstupu,
7. balance – vyvažovací filtr,
8. high – vvládání parametru highshelf filtru,

9. low – ovládání parametru lowshelf filtru.

Menu

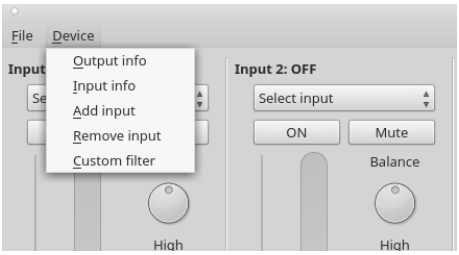
Položky menu reagují na klávesové zkratky, typicky první písmeno.

File

- Exit – vypnutí programu

Device

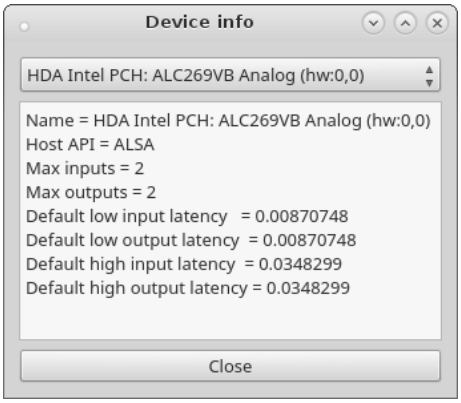
- Output info – okno pro informace o výstupních zařízení,
- input info – okno pro informace o vstupních zařízení,
- add input – přidání panelu vstupu,
- remove input – odebrání panelu vstupu,
- custom filter – okno pro zadání parametrů vlastního filtru (obrázek 4.5).



Obrázek 4.2: Ukázka menu zařízení

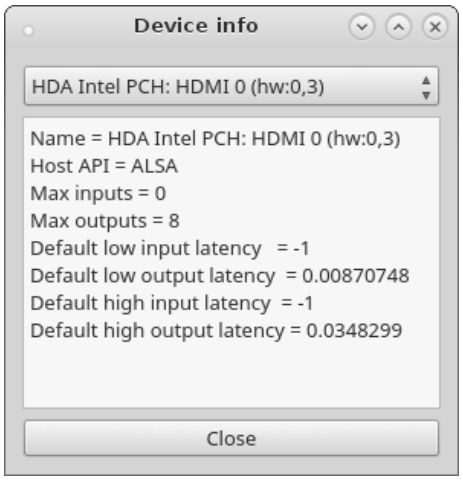
Informace o zařízeních

Vstupní zařízení Nástroj pro přehled vlastností vstupních zařízeních.



Obrázek 4.3: Okno s informacemi o vstupním zařízení

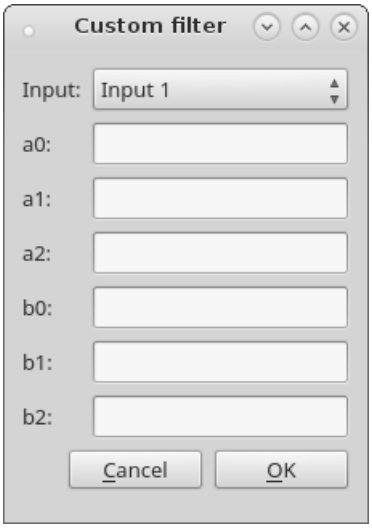
Výstupní zařízení Nástroj pro přehled vlastností výstupních zařízeních.



Obrázek 4.4: Okno s informacemi o výstupních zařízeních

Vlastní filtr

Možnost vytvoření vlastního filtru. Obsahuje výběrové pole pro zvolení vstupu, na který se filtr aplikuje, a textové pole pro zadání jeho parametrů.



Obrázek 4.5: Okno přídání vlastního filtru

4.2.2 Latence

Zpoždění aplikace je minimální vzhledem k využití série bikvadratických filtrů. Teoretické zpoždění těchto filtrů je $2 * n$ vzorků kde n je počet filtrů v sérii. Program volí automaticky nejmenší možné zpoždění vstupů a výstupu zvukové karty. Závisí zde tedy především na výpočetním výkonu procesoru. Na testovaném počítači s

procesorem Intel i3 nebyla empiricky zaznamenána prodleva při používání programu pro ekvalizaci a přehrání kytarového vstupu.

5 Závěr

Práce umožnila seznámení se zpracováním zvuku v reálném čase a vytvoření aplikace, která využívá zvukovou knihovnu PortAudio a grafickou knihovnu Qt. Obě použité knihovny jsou multiplatformní a distribuované pod svobodnými licencemi. Aplikace byla napsána v jazyce C++ a otestována na operačním systému Arch Linux na počítači s integrovanou zvukovou kartou Intel a externí zvukovou kartou M-Audio Fast Track C400. Vytvořený program je možné ovládat pomocí responzivního GUI, podporující dynamické přidávání a odebírání vstupů za běhu. Lze v něm přehrát vstupní signály s minimální latencí. Mezi jeho funkce patří úprava hlasitosti vstupních a výstupních zařízení, ovládání rovnováhy stereo vstupu, ekvalizace signálu realizovaná sérii bikvaratických filtrů a signalizace hlasitostní úrovně vstupu.

5.1 Uplatnění aplikace

Program může být využit kdekoli, kde je potřeba míchat zvukové stopy nebo provést korekce signálu. Nabízí široké možnosti využití, protože jím lze nahradit analogový mixážní pult, například k ozvučení sálů, kaváren nebo učeben. Navíc může být použit i jako ekvalizér pro hudební nástroje.

5.2 Možnost rozšíření

Je zde stále prostor pro pokračování ve vývoji a vylepšování programu. Mezi funkce, které by mohly být v budoucnu implementovány, patří například podpora (VTS) pluginů, grafický ekvalizér, nahrávání vybraných stop, možnost více výstupů, efekty (reverb, distortion,..), přehrávání ze souborů nebo metronom.

Literatura

[1] Sen M. Kuo, Bob H. Lee, Wenshun Tian, *Real-Time Digital Signal Processing Implementations and Applications* 2nd Edition, 2006.

[2] Robert Bristow-Johnson, *Cookbook formulae for audio EQ biquad filter coefficients* [online]. [cit. 2016-05-15]. Dostupné z: `http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt`

[3] Texas Instruments, *Configure the Coefficients for Digital Biquad Filters in TLV320AIC3xxx Family* [online], 2010. Dostupné z: `http://www.ti.com/lit/an/slaa447/slaa447.pdf`

[4] B. Porat, *A Course in Digital Signal Processing*, John Wiley & Sons, 1997.

[5] Glen Ballou, *Filters and Equalizers*, Handbook for Sound Engineers, 4th edition, Focal Press, 2008.

[6] J. Fessler, *Design of Digital Filters* [online], 2004. Dostupné z: `http://web.eecs.umich.edu/~fessler/course/451/l/pdf/c8.pdf`

[7] Jiří Jan, *Číslicová filtrace, analýza a restaurace signálů. 2.*. 2002. ISBN 80-214-1558-4.

[8] Wikimedia Commons *Flowchart of a digital biquad filter* [online]. 2010 [cit. 2016-05-15]. Dostupné z: `https://commons.wikimedia.org/wiki/File:Biquad_filter_DF-I.svg`

[9] Wikimedia Commons *A diagram of an FIR filter* [online]. 2009 [cit. 2016-05-15]. Dostupné z: `https://commons.wikimedia.org/wiki/File:FIR_Filter_(Moving_Average).svg`

[10] Wikimedia Commons *Typický řetězec s DSP* [online]. 2006 [cit. 2016-05-15]. Dostupné z: `https://commons.wikimedia.org/wiki/File:DSP_%C5%99et%C4%9Bzec.png`

[11] Wikimedia Commons *Kvantování vzorků při A/D převodu* [online]. 2006 [cit. 2016-05-15]. Dostupné z: `https://commons.wikimedia.org/wiki/File:Kvantov%C3%A1n%C3%AD.png`

[12] Wikimedia Commons *Sampling demonstration used in signal processing* [online]. 2009 [cit. 2016-05-15]. Dostupné z: `https://commons.wikimedia.org/wiki/File:Signal_Sampling.png`

[13] *PortAudio API Overview* [online]. [cit. 2016-05-15]. Dostupné z: `http://portaudio.com/docs/v19-doxydocs/api_overview.html`.

[14] *PortAudio Tutorials* [online]. [cit. 2016-05-15]. Dostupné z: `http://portaudio.com/docs/v19-doxydocs/tutorial_start.html`.

[15] *Qt Documentation* [online]. [cit. 2016-05-15]. Dostupné z: `http://doc.qt.io/`.