VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE PARAMETRICKÉHO EKVALIZÉRU DO VST3 ZÁSUVNÉHO MODULU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

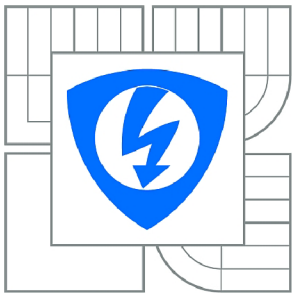
AUTOR PRÁCE
AUTHOR

JINDŘICH PEVNÝ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE PARAMETRICKÉHO EKVALIZÉRU DO VST3 ZÁSUVNÉHO MODULU

IMPLEMENTATION OF THE PARAMETRIC EQUALIZER INTO VST3 PLUG-IN

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

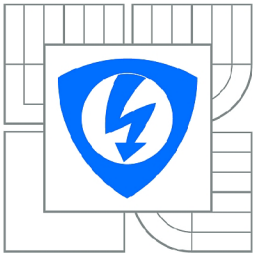
AUTOR PRÁCE — JINDŘICH PEVNÝ

AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR FRENŠTÁTSKÝ

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Jindřich Pevný ***ID:*** 155217
Ročník: 3 ***Akademický rok:*** 2014/2015

NÁZEV TÉMATU:

Implementace parametrického ekvalizéru do VST3 zásuvného modulu

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je seznámit se s technologií zásuvných modulů VST3, které se využívají pro implementaci algoritmů pro číslicové zpracování signálů ve zvukové technice.
V rámci bakalářské práce student provede návrh parametrického ekvalizéru využívající dekompozici fázovacího článku a daný algoritmus implementuje do zasuvného modulu technologie VST3 pomocí jazyka C++.

DOPORUČENÁ LITERATURA:

ZÖLZER, U. Digital Audio Signal Processing, První vydání. New York: McGraw-Hill, Inc., 1997, 290 stran. ISBN 0-47-197226-6.
SMÉKAL, Zdeněk. Systémy a signály: 1D a 2D diskrétní a číslicové zpracování. 1. vyd. Praha: nakladatelství Sdělovací technika, 2013, 254 s. ISBN 978-80-86645-23-0.
Steinberg VST SDK 3.0 Overview [online]. Hamburg : Steinberg Media Technologies, 2008. Dostupné z WWW: <http://www.steinberg.net/en/company/developer.html>.

Termín zadání: 9.2.2015 ***Termín odevzdání:*** 2.6.2015

Vedoucí práce: Ing. Petr Frenštátský
Konzultanti bakalářské práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce se věnuje oblasti DSP (číslicové zpracování signálů – Digital Signal Processing), konkrétně parametrickým strukturám digitálních shelving filtrů využitelných jakožto stavebních bloků parametrického ekvalizéru. Cílem práce je nastudovat možnosti nepřímého návrhu parametrických filtrů využitím dekompozice přenosové funkce shelving filtrů na funkci fázovacího článku a následná implementace těchto filtrů do zásuvného modulu pomocí Steinberg VST3 SDK.

KLÍČOVÁ SLOVA

parametrický filtr, fázovací článek, VST, nepřímý návrh filtrů, bilineární transformace, ekvalizér

ABSTRACT

This thesis deals with the field of DSP (Digital Signal Processing), specifically with structures of digital parametric shelving filters, which can be used as building blocks of parametric equalizer. The objective of this thesis is to learn the options of indirect parametric filter design using decomposition of transfer functions into the function of an all-pass filter and implement these using Steinberg's VST3 SDK.

KEYWORDS

parametric filter, all-pass filter, VST, indirect filter design, billinear transform, equalizer

PEVNÝ, Jindřich *Implementace parametrického ekvalizéru do VST3 zásuvného modulu*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015. 53 s. Vedoucí práce byl Ing. Petr Frenštátský

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Implementace parametrického ekvalizéru do VST3 zásuvného modulu“ jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské 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 práce panu Ing. Petru Frenštátskému za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Místo ....

podpis autora(-ky)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Parametrické ekvalizéry umožňují, oproti klasickým ekvalizérům, které mají pevně nastavené mezní pracovní frekvence, přesnější modelování frekvenčního spektra ekvalizovaného signálu. Nastavením jednotlivých parametrů je možné upravovat hodnotu mezních popř. středních frekvencí, šířku pásma a zesílení daného pásma, v ideálním případě nezávisle na sobě. Výhodou parametrických ekvalizéru je možnost větší kontroly nad spektrem zpracovávaného signálu. Zvláštím druhem parametrického ekvalizéru je ekvalizér semi-parametrický, který se hojně vyskytuje např.: v levnějších analogových mixpultech. Takovýto ekvalizér umožňuje úpravu pouze vybraných parametrů, konkrétně střední frekvence a zesílení.

V současné době, kdy už je ve většině případech post-procesingová práce přesunuta z hardwarových konzolí do softwarového prostředí aplikací DAW, se velké popularitě těší parametrické ekvalizéry v podobě VST zásuvného modulu. Takovýto modul v kombinaci se spektrálním analyzérem se v rukou zvukového inženýra stává mocným nástrojem a značně zefektivňuje práci v hudebním studiu, ať už co se týče větší kontroly na celým spektrem signálu, ale i celkově rychlejší práci, obzvláště při mixu skladeb s velkým počtem hudebních stop.

Hlavním cílem práce je nastudovat způsob návrhu filtrů nepřímou metodou, s tím související transformace přenosových funkcí ze spojité časové oblasti do oblasti diskrétní, zjistit způsob realizace elementárních filtrů pomocí fázovacího článku a získané poznatky zúročit při návrhu konkrétních filtrů a jejich následné implementace do podoby VST3 zásuvného modulu. Součástí projektu je i implementace navržených filtrů v podobě programu realizovaného v prostředí Matlab, zhodnocení jejich vlastností a použité metody.

1 PARAMETRICKÉ EKVALIZÉRY

1.1 Využití parametrických ekvalizérů pro úpravy audio signálů

Ekvalizace je základním elementem v obecné úpravě zvukových nahrávek. Pomocí vhodné úpravy spektra daného zvukového signálu je možné docílit např.: silnějšího dojmu daného zvuku, odstranění nedokonalostí či umístění daného hudebního prvku skladby při mixu do frekvenčního spektra tak, aby nekolidoval s jinými prvky. Pro lepší ilustraci toho, jakým způsobem ekvalizace daných pásem ovlivňuje celkový charakter zvuku je vhodné pohlédnout na strukturu spektra hudebních skladeb očima studiového inženýra. Tabulka 1.1 obsahuje některé důležité důsledky úprav konkretních pásem [11]. S možností přesně upravovat frekvenční spektrum jednotlivých zvukových elementů jdou ruku v ruce specifické možnosti využití těchto ekvalizérů pro práci s audio signály. Pro příklad uveďme některé z nich.

1.1.1 Eliminace nežádoucích zvuků v nahrávce

Zvuky zabírající úzké frekvenční pásmo (např.: zvuk trsátka v nahrávce akustické kytary) je poměrně snadné eliminovat zavedením parametrické pásmové zádrže se střední frekvencí rovné dominantní frekvenci parazitního zvuku, požadovanou šířkou pásma (volíme co nejužší abychom co nejméně zasáhli do zbytku nahrávky) a přiměřenou hodnotou záporného zesílení.

1.1.2 Eliminace zpětné vazby

Zpětná vazba jako extrémní případ rezonance je častým a nepříjemným problémem, zejména co se týče živých vystoupení. Vyrušení zpětné vazby je možné, podobně jako v předchozím případě, pomocí pásmové zádrže se střední frekvencí totožnou s danou rezonanční frekvencí.

1.1.3 Korekce charakteristik reproduktorů

Poměrně časté je i použití parametrických filtrů za účelem korekce nedokonalé frekvenční charakteristiky reproduktorů. Vhodně zvolenými filtry lze dosáhnout více ploché a konstantní charakteristiky, důsledkem čehož pak daný reproduktor dosahuje přesnějšího, nezkresleného podání zvuku.

1.1.4 Korekce hlasitosti určitých tónů

Je poměrně běžné, že při nahrávání hudebního nástroje jsou některé tóny hlasitější než ostatní, což může pramenit už ze samotné konstrukce daného nástroje. Obvykle se tyto rozdíly v hlasitosti vyrovnávají pomocí kompresoru, nicméně pokud jej z nějakého důvodu nemůžeme použít, je jednou z možností, jak tuto nedokonalost napravit právě použití parametrického ekvalizéru, kdy je možné upravit hlasitost tónů s přesně stanovenou frekvencí.

Tab. 1.1: Ekvalizace z pohledu postprodukce hudebních nahrávek.

Sub-bass	16-60 HZ	Obecně přijímané tvrzení je, že tyto frekvence je spíše „cítit“ než slyšet, proto úpravy tohoto pásma ovlivňují celkovou sílu a „fyzický dopad“ skladby na posluchače.
Bass	60-250 HZ	Obsahuje základní tóny rytmické sekce (bicí nástroje, baskytara...). Má podobně jako sub-basové pásmo vliv na sílu a energii skladby, ovšem ve více „muzikálním“ smyslu slova.
Low mids	250-2500 HZ	V tomto pásmu lze nalézt dominantní frekvence většiny hudebních nástrojů. Přehnané zesílení tohoto pásma vede k „telefonnímu“ charakteru zvuku a může způsobit celkovou nečitelnost výsledné skladby.
High mids	2-4 kHZ	Obsahuje frekvence důležité pro srozumitelnost zaznamenaného lidského hlasu. Přílišné zesílení může vést k nečitelnosti některých hlásek, zejména těch tvořenými rty – m,b,v. Citlivé zesílení vokální stopy kolem 3 kHZ v kombinaci s adekvátním zeslabením instrumentálních stop ve stejném frekvenčním pásmu vede k vyniknutí vokální stopy v mixu bez nutnosti celkového zeslabení instrumentálního elementu.
Brilliance	4-6 kHZ	Zesílení tohoto pásma vede k dojmu, že se daný zvuk nachází blíže k posluchači a naopak zeslabení k dojmu, že zdroj zvuku je od posluchače vzdálenější.
Presence	6-16 kHZ	Pásmo ovlivňující čistotu a zřetelnost zvuku. Přílišné zesílení však může zdůraznit sykavky.

1.2 Shelving filtry jako stavební bloky parametrických ekvalizérů

Pro modelování frekvenčního spektra signálů se používají tzv. shelving filtry, na které tedy můžeme nahlížet jako na základní stavební bloky parametrických ekvalizérů. Tyto filtry jsou založeny na principu paralelního spojení elementárního filtru a vše propustného systému, kdy se propustné pásmo daného elementárního filtru s určitým zesílením (zeslabením) přičítá k původnímu nezměněnému spektru ekvalizovaného signálu.

1.2.1 Low-shelving filtr

Pro modelování nižších kmitočtů frekvenčního spektra je používáný tzv. low-shelving filtr, který je možné získat paralelním spojením dolní propusti prvního řádu s přenosovou funkcí $H_{\mathrm{DP}}(p)$) a vše propustného systému s přenosovou funkcí rovné jedné [15].

$$H_{\mathrm{LS}}(p) = 1 + H_{\mathrm{DP}}(p) = 1 + \frac{H_0}{p+1} \tag{1.1}$$

Hodnoty zesílení v krajních bodech jsou tedy:

$$H_{\mathrm{LS}}(0) = 1 + H_0 = V_0 \tag{1.2}$$

$$H_{\mathrm{LS}}(\infty) = 1 \tag{1.3}$$

Výslednou přenosovou funkci můžeme zapsat jako:

$$H_{\mathrm{LS}}(p) = \frac{p + V_0}{p+1} \tag{1.4}$$

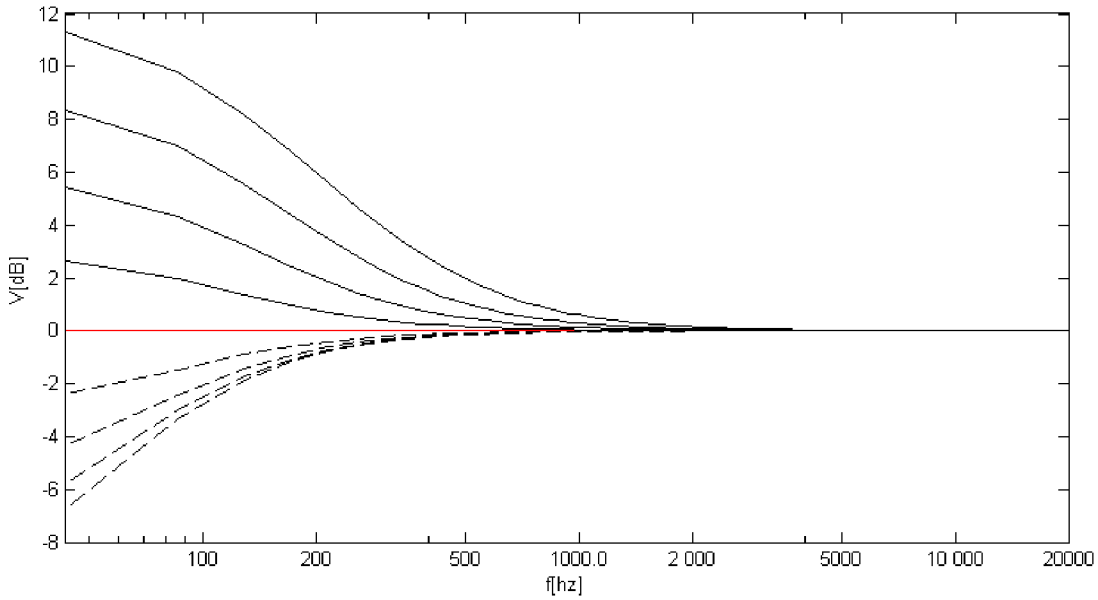
Můžeme pozorovat že tvar charakteristiky na Obr. 1.1 pro případ zesílení (boost) neodpovídá tvaru pro případ zeslabení (cut). Pro zachování symetrie je tedy nutné zavést samostatnou přenosovou funkci pro případ cut [15].

$$H_{\mathrm{LS}}(p) = \frac{p+1}{p + V_0} \tag{1.5}$$

1.2.2 High-shelving filtry

Jak už je z názvu patrné, jedná se o filtry určené pro modelování horních kmitočtů frekvenčního spektra. Jeho návrh probíhá analogicky k návrhu low-shelving filtru s tím rozdílem, že přenosovou funkci dolní propusti nahradí přenosová funkce horní propusti prvního řádu.

$$H_{\mathrm{HS}}(p) = 1 + H_{\mathrm{HP}}(p) = 1 + \frac{H_0 p}{p+1} \tag{1.6}$$



Obr. 1.1: Frekvenční odezva filtru bez úpravy přenosové funkce pro případ cut.

Krajní hodnoty frekvenční odezvy jsou tedy:

$$H_{\mathrm{HS}}(0) = 1 \tag{1.7}$$

$$H_{\mathrm{HS}}(\infty) = 1 + H_0 = V_0 \tag{1.8}$$

Výslednou přenosovou funkci je možné zapsat jako:

$$H_{\mathrm{HS}}(p) = \frac{pV_0 + 1}{p + 1} \tag{1.9}$$

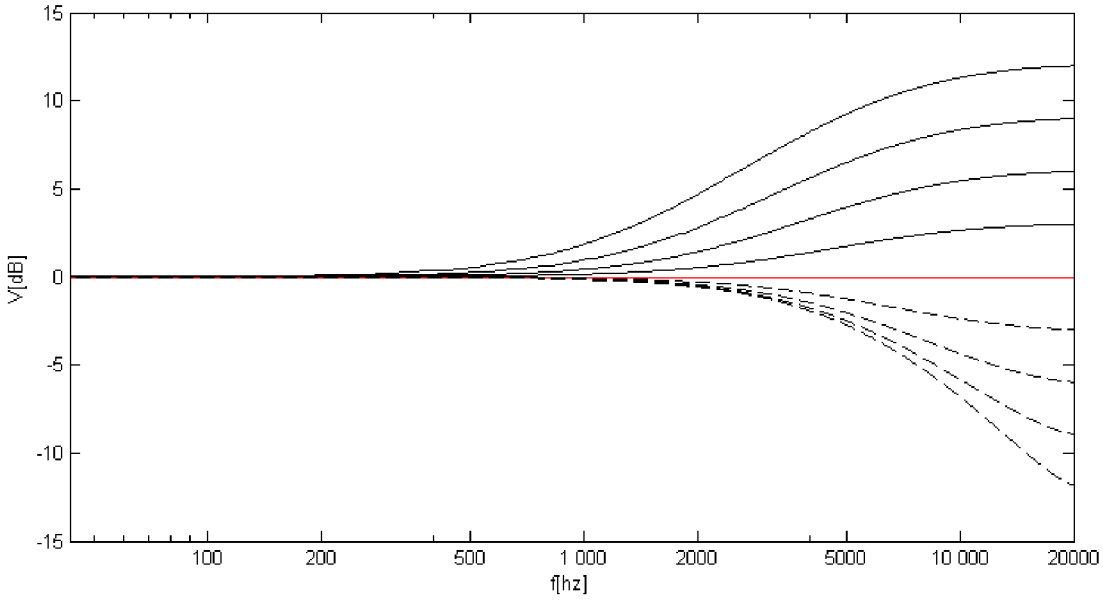
Frekvenční odezvu tohoto filtru lze pozorovat na Obr. 1.2 Pro zachování symetrické frekvenční odezvy je opět nutné zavést novou přenosovou funkci pro případ zeslabení [15]:

$$H_{\mathrm{HS}}(p) = \frac{p + 1}{pV_0 + 1} \tag{1.10}$$

1.2.3 Peak filtr

Peak filtr umožňuje úpravy pásma frekvenčního spektra přesně určeného střední frekvencí a šířkou. Samotná realizace spočívá v kombinaci pásmové propusti druhého řádu a vše propustného systému s jednotkovou přenosovou funkcí.

$$H_P(p) = 1 + H_{PP}(p) \tag{1.11}$$



Obr. 1.2: Frekvenční odezva filtru bez úpravy přenosové funkce pro případ cut.

1.3 Fázovací články a jejich využití pro návrh parametrických filtrů

1.3.1 Fázovací článek

Fázovací článek, neboli all-pass filtr, je takový filtr, který neupravuje zesílení jednotlivých frekvenčních složek zpracovávaného signálu, ale mění jejich fázi.

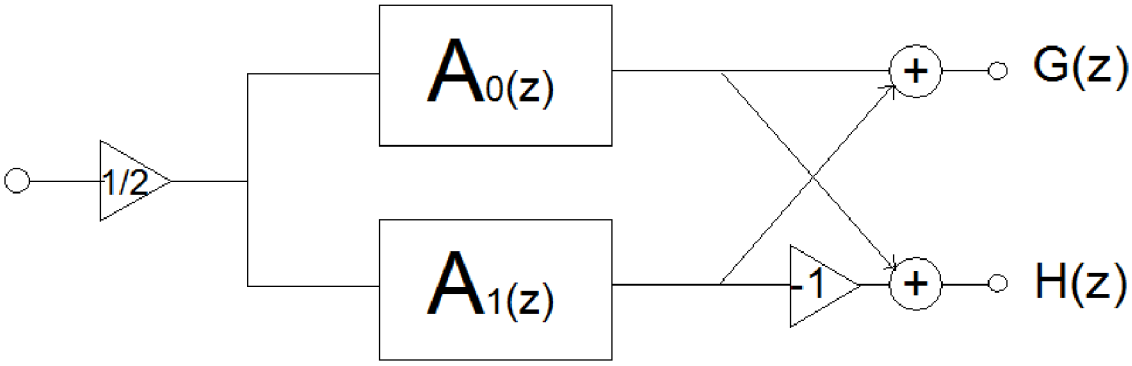
$$|A(\mathrm{e}^{\mathrm{j}\omega})| = 1 \tag{1.12}$$

Frekvenční modulová charakteristika 1.12 je tedy konstantní v celém frekvenčním spektru nezávisle na hodnotě koeficientů přenosové funkce fázovacího článku [12].

$$\phi A(\omega) = arg[A(\mathrm{e}^{\mathrm{j}\omega})] \tag{1.13}$$

Fázovou frekvenční charakteristiku lze popsat jako argument frekvenční odezvy fázovacího článku vztahem 1.13 [12]. Požadovaného fázového posunu lze dosáhnout pomocí vhodného výběru koeficientů přenosové funkce.
Skutečné zpoždění daných frekvenčních složek signálu při průchodu fázovacím článkem nazýváme skupinové zpoždění, které je dáno derivací fáze podle úhlové frekvence [13] vztahem 1.14.

$$\tau_G = \frac{\mathrm{d}\phi}{\mathrm{d}\omega} \tag{1.14}$$



Obr. 1.3: Struktura složeného filtru realizovaného pomocí fázovacího článku.

1.3.2 Realizace filtrů pomocí fázovacích článků

Realizovat jakýkoliv signálový filtr (DP,HP,PP) je možné pomocí paralelního zapojení dvou fázovacích článků tak, aby v propustném pásmu dosahovaly oba články stejného fázového posuvu a naopak v nepropustném pásmu byly jednotlivé frekvenční složky v protifázi [16].

Mějme komplementární dvojici filtrů $G(z)$ a $H(z)$ řádu n. Filtr tvořený tímto párem je možné realizovat jako paralelní zapojení dvou fázovacích článků A_N a A_M kde $n = N + M$. Pro tento příklad uvažujme řád $M = 1$ tedy $N = 0$ a nepočítejme se zesílením propustného pásma ani jednoho z filtrů. A_0 je tedy přenosová funkce „nultého řádu“ $\rightarrow A_0 = 1$. Filtry lze tedy realizovat jako: [12]

$$G(z) = \frac{1}{2}(A_0 + A_1) = \frac{1}{2}(1 + A_1) \tag{1.15}$$

$$H(z) = \frac{1}{2}(A_0 - A_1) = \frac{1}{2}(1 - A_1) \tag{1.16}$$

Výsledná struktura filtru $F(z) = G(z) + H(z)$ realizovaná pomocí fázovacího článku má podobu Obr. 1.3 [12]. Při návrhu parametrických filtrů v rámci této práce je využita matematická úprava přenosových funkcí shelving filtrů právě do podoby vztahů 1.15 a 1.16.

2 TRANSFORMACE PŘENOSOVÉ FUNKCE

2.1 Laplaceova transformace

Tato transformace převádí funkce reálné proměnné na funkce proměnné komplexní, díky čemuž výrazně zjednodušuje vztahy mezi původními funkcemi. Právě kvůli této vlastnosti se Laplaceova transformace používá pro řešení diferenciálních rovnic, což z ní dělá vhodný nástroj pro popis systémů pracujících ve spojitém čase.
Převod z roviny r do komplexní roviny p je realizován pomocí následujícího vztahu: [1]

$$F(p) = \int_0^\infty f(t)\, exp(-pt)\mathrm{d}t \tag{2.1}$$

Funkce $F(p)$ se nazývá Laplaceův obraz.

2.2 Z-transformace

Z-transformace je diskrétní variantou Laplaceovy transformace. Transformace je realizována podle vztahu: [1]

$$F(z) = \sum_{n=0}^{\infty} \frac{f_n}{z^n} \tag{2.2}$$

2.3 Analogově číslicová transformace

V této práci je pro návrh jednotlivých filtrů použita nepřímá neboli transformační metoda. Tento způsob návrhu spočívá v použití vhodných transformačních vztahů pro převod spojité analogové přenosové funkce definované v rovině p na diskrétní číslicovou funkci v rovině z. Na transformaci $p \to z$ klademe dvě podmínky:

1. Imaginární osa roviny $p = \mathrm{j}w$ se musí transformovat na jednotkou kružnici v rovině $z = \mathrm{e}^{\mathrm{j}}w$, což má za následek zachování kmitočtové charakteristiky modelového filtru u číslicového filtru vzniklého transformací[14].
2. Stabilní analogový filtr se musí transformovat na stabilní digitální filtr, tedy levá polorovina $R(p) < 0$ roviny p se musí transformovat dovnitř jednotkové kružnice $|z| < 1$ roviny z [14].

V praxi používanými metodami jsou:

- Invariantní impulzní odezva – spočívá v hledání takové impulzní odezvy číslicového filtru, která se shoduje s analogovou impulzní odezvou modelového filtru v okamžicích vzorkování $t = Tn$.
- Diskrétní aproximace derivace – založena na diskrétní aproximaci derivace spojité funkce.

- Bilineární transformace – založena na numerické integraci diferenciální rovnice prvního řádu

Pro účely této práce byla zvolena metoda bilineární transformace, která je vhodná pro transformaci přenosových funkcí filtrů s po částech konstantní modulovou kmitočtovou charakteristikou (filtry typu DP,HP,PP a PZ).

2.3.1 Bilineární transformace

Jak již bylo zmíněno, metoda bilineární transformace je založena na numerické integraci diferenciální rovnice 1. řádu. Mějme tedy diferenciální rovnici

$$\frac{\mathrm{d}}{\mathrm{d}t}y(t) = x(t) \tag{2.3}$$

Po aplikaci numerické integrace lichoběžníkovou metodou pro $t \in < t_1, t_2 >$ na tuto rovnici dostáváme vztah: [2]

$$y(t_2) - y(t_1) = \int_{t_1}^{t_2} f(x) \approx (t_2 - t_1)\frac{x(t_1) + x(t_2)}{2} \tag{2.4}$$

Následně dosadíme vzorkovací okamžik $(n-1)T$ za t_1 a nT za t_2: [10]

$$y(nT) - y((n-1)T) = \frac{T}{2}(x(n-1)T) + x(nT)) \tag{2.5}$$

Na výsledný vztah aplikujeme Z-transformaci vedoucí na: [10]

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

a po úpravě dostáváme rovnici:

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

Dále je pro získání transformačních vztahů nutné aplikovat Laplaceovu transformaci na výchozí diferenciální rovnici 2.1 , což vede ke vztahu:

$$pY(p) = X(p) \tag{2.8}$$

Spojením 2.5 a 2.6 získáváme finální transformační vztah[10].

$$p = \frac{2}{T}\frac{z-1}{z+1} \tag{2.9}$$

Pokud do vztahu 2.9 dosadíme za z jednotkovou kružnici $e^{\mathrm{j}\omega}$ dostaneme vztah 2.10 ilustrující způsob transformace frekvenční osy. [9]

$$p = \frac{2}{T}\frac{e^{\mathrm{j}\omega} - 1}{e^{\mathrm{j}\omega} + 1} = \frac{2}{T}\mathrm{j}\frac{\sin(\omega\frac{T}{2})}{\cos(\omega\frac{T}{2})} = \frac{2}{T}\mathrm{j}\tan(\omega\frac{T}{2}) \tag{2.10}$$

Z vztahu 2.10 je možné pozorovat, že dochází k frekvenčnímu zkreslení určenému vztahem [9]:

$$\omega = \frac{2}{T}\arctan(\frac{\omega_d}{2}) \tag{2.11}$$

Vliv tohoto zkreslení je možné eliminovat předzkreslením analogové kmitočtové osy vztahem [14].

$$\omega = \frac{2}{T}\tan(\frac{\omega_d}{2}) \tag{2.12}$$

3 NÁVRH PARAMETRICKÝCH FILTRŮ

V této části přistoupíme k samotnému návrhu modelů parametrických filtrů pomocí dekompozice přenosové funkce shelving filtrů na funkci fázovacího článku a výpočtu jejích koeficientů. Pro návrh těchto filtrů byla zvolena tzv. nepřímá metoda, při které nejprve pracujeme s modelem filtru pracujícím ve spojité časové oblasti a až poté jej transformujeme pomocí bilineární transformace na jeho digitální podobu pracující v diskrétním čase.

3.1 Parametrický low-shelving filtr

Parametrický filtr tohoto typu umožňuje úpravy spektra v oblasti nízkých kmitočtů v závislosti na dvou parametrech – mezní frekvence f_{m}[Hz] a zesílení daného pásma V_0[dB].

3.1.1 Návrh přenosové funkce

Vycházejme z přenosové funkce analogového low-shelving filtru 1.4 a její denormalizované podoby pro případ boost($V_0 > 0$): [15]

$$H_{\mathrm{LS}}(p) = \frac{p + V_0\omega_c}{p + \omega_c} \tag{3.1}$$

Funkci lze pro lepší přehled zapsat jako součet dvou subfunkcí:

$$H_{\mathrm{LS}}(p) = \frac{p}{p + \omega_c} + V_0\frac{\omega_c}{p + \omega_c} \tag{3.2}$$

Tyto dvě funkce je možné matematickou úpravou vyjádřit v jejich all-pass podobě jako:

$$\frac{p}{p + \omega_c} = \frac{1}{2}\left[1 + \frac{p - \omega_c}{p + \omega_c}\right] \tag{3.3}$$

$$\frac{V_0\omega_c}{p + \omega_c} = \frac{V_0}{2}\left[1 - \frac{p - \omega_c}{p + \omega_c}\right] \tag{3.4}$$

S přenosovou funkcí fázovacího článku: [15]

$$A_b(p) = \frac{p - \omega_c}{p + \omega_c} \tag{3.5}$$

Výslednou přenosovou funkci low-shelving filtru v all-pass formě lze tedy zapsat jako:

$$H_{\mathrm{LS}}(p) = \frac{1}{2}\left[1 + A_b(p)\right] + \frac{V_0}{2}\left[1 - A_b(p)\right] \tag{3.6}$$

Pro převod modelu ze spojitého do diskretního času nyní aplikujeme bilineární transformaci dle vzorce 2.7:

$$H_{\mathrm{LS}}(z) = \frac{1}{2}\left[1 + A_b(z)\right] + \frac{V_0}{2}\left[1 - A_b(z)\right] \tag{3.7}$$

Pro zachování symetrie frekvenční odezvy je nutné zavést zvláštní přenosovou funkci pro případ útlumu (viz. Kapitola 1). Její denormalizovaná podoba odpovídá vztahu: [15]

$$H_{\mathrm{LS}}(p) = \frac{p + \omega_c}{p + \frac{\omega_c}{V_0}} \tag{3.8}$$

Postupujeme-li stejným způsobem jako pro případ zesílení, dostaneme all-pass vyjádření přenosové funkce ve tvaru:

$$H_{\mathrm{LS}}(p) = \frac{1}{2}\left[1 + A_c(p)\right] + \frac{V_0}{2}\left[1 - A_c(p)\right] \tag{3.9}$$

s přenosovou funkcí fázovacího článku A_c:

$$A_c(p) = \frac{p - \frac{\omega_c}{V_0}}{p + \frac{\omega_c}{V_0}} \tag{3.10}$$

Lze pozorovat, že vzorec 3.6 a 3.9 se liší pouze v přenosové funkci použitého fázovacího článku. Z toho plyne, že zesílení i útlum lze realizovat pomocí stejné filtrové struktury, při čemž rozdíl nastává až při výpočtu koeficientů fázovacího článku pro jednotlivé případy [15].

3.1.2 Výpočet koeficientů přenosové funkce

Přenosová funkce $A_b(z)$ fázovacího článku pracujícího v diskrétní oblasti pro případ zesílení po substituci dle transformačního vztahu odpovídá:

$$A_b(z) = -\frac{\frac{\omega_c T - 2}{\omega_c T + 2} + z^{-1}}{1 + \frac{\omega_c T - 2}{\omega_c T + 2} z^{-1}} \tag{3.11}$$

kde $\frac{\omega_c T - 2}{\omega_c T + 2}$ označujeme jako frekvenční parametr fázovacího článku. Pro eliminaci frekvenčního zkreslení vyplývajícího z bilineární transformace je třeba provést předzkreslení dle vztahu 2.12.

$$\omega_c = \frac{2}{T}\tan\left(\frac{\omega_c T}{2}\right)$$

$$\frac{\omega_c T - 2}{\omega_c T + 2} = \frac{2(\omega_c \frac{T}{2} - 1)}{2(\omega_c \frac{T}{2} + 1)} = \frac{\omega_c \frac{T}{2} - 1}{\omega_c \frac{T}{2} + 1}$$

$$a_b = \frac{\frac{2}{T}\tan\left(\frac{\omega_c T}{2}\right)\frac{T}{2} - 1}{\frac{2}{T}\tan\left(\frac{\omega_c T}{2}\right)\frac{T}{2} + 1} = \frac{\tan(\frac{\omega_c T}{2}) - 1}{\tan(\frac{\omega_c T}{2}) + 1} \tag{3.12}$$

Po předzkreslení střední úhlové frekvence ω_c byl zjištěn frekvenční parametr a_b:

$$a_b = \frac{\tan(\frac{\omega_c T}{2}) - 1}{\tan(\frac{\omega_c T}{2}) + 1} \tag{3.13}$$

Analogickým způsobem byl zjištěn i parametr pro případ cut:

$$a_c = \frac{\tan(\frac{\omega_c T}{2}) - V_0}{\tan(\frac{\omega_c T}{2}) + V_0} \tag{3.14}$$

3.2 Parametrický high-shelving filtr

Stejný způsob byl zvolen i pro návrh parametrického high-shelving filtru, který slouží k úpravě spektra s maximálním zesílením v jeho horním okraji opět v závislosti na mezní frekvenci f_{m}[Hz] a zesílení V_0[dB].

3.2.1 Návrh přenosové funkce

Vycházejme z přenosové funkce analogového high-shelving filtru 1.9 a její denormalizované podoby pro případ boost($V_0 > 0$): [15]

$$H_{\mathrm{HS}}(p) = \frac{pV_0 + \omega_c}{p + \omega_c} \tag{3.15}$$

Dekompozice této přenosové funkce vede na vztah:

$$H_{\mathrm{HS}}(p) = \frac{1}{2}\left[1 - A_b(p)\right] + \frac{V_0}{2}\left[1 + A_b(p)\right] \tag{3.16}$$

S přenosovou funkcí fázovacího článku: [15]

$$A_b(p) = \frac{p - \omega_c}{p + \omega_c} \tag{3.17}$$

Pro převod modelu do diskretního času aplikujeme bilineární transformaci :

$$H_{\mathrm{HS}}(z) = \frac{1}{2}\left[1 - A_b(z)\right] + \frac{V_0}{2}\left[1 + A_b(z)\right] \tag{3.18}$$

Pro zachování symetrie frekvenční odezvy je opět nutné zavést zvláštní přenosovou funkci pro případ útlumu (viz. Kapitola 1). Její denormalizovaná podoba odpovídá vztahu: [15]

$$H_{\mathrm{HS}}(p) = \frac{p + \omega_c}{\frac{p}{V_0} + \omega_c} \tag{3.19}$$

Filtr realizujeme pomocí stejné struktury jako v předchozím případě:

$$H_{\mathrm{HS}}(p) = \frac{1}{2}\left[1 - A_c(p)\right] + \frac{V_0}{2}\left[1 + A_c(p)\right] \tag{3.20}$$

S přenosovou funkcí fázovacího článku: [15]

$$A_c(p) = \frac{p - \omega_c * V_0}{p + \omega_c * V_0} \tag{3.21}$$

3.2.2 Výpočet koeficientů přenosové funkce

Přenosová funkce fázovacího článku pro případ zesílení je stejná jako u low-shelving filtru, tedy i frekvenční parametr a_b bude stejný.

$$a_b = \frac{\tan(\frac{\omega_c T}{2}) - 1}{tan(\frac{\omega_c T}{2}) + 1} \tag{3.22}$$

Pro $V_0 < 0$ odpovídá přenosová funkce použitého fázovacího článku po bilineární transformaci vztahu:

$$A_c(z) = -\frac{\frac{2-V_0\omega_c T}{2+V_0\omega_c T} + z^{-1}}{1 + \frac{2-V_0\omega_c T}{2+V_0\omega_c T} z^{-1}} \tag{3.23}$$

kde $\frac{2-V_0\omega_c T}{2+V_0\omega_c T}$ je frekvenční parametr fázovacího článku. Po aplikaci předzkreslení získáváme finální vztah:

$$a_c = \frac{V_0 \tan(\frac{\omega_c T}{2}) - 1}{1 + V_0 \tan(\frac{\omega_c T}{2})} \tag{3.24}$$

3.3 Parametrický peak filtr

Parametrickou reprezentaci peak filtru můžeme získat opět obdobně jako v předchozích případech pomocí kombinace pásmové propusti a vše-propustného systému a nebo využitím spektrální transformace dolní propust → pásmová propust. Druhá varianta nabízí poměrně rychlý přístup k návrhu digitálních filtrů, neboť nám stačí navrhnout pouze modelovou dolní propust a poté vhodnou transformací získat požadovaný filtr.

3.3.1 Návrh přenosové funkce

Vycházejme z all-pass podoby přenosové funkce low shelving filtru 3.7.

$$H_\mathrm{LS}(z) = \frac{1}{2}\left[1 + A_b(z)\right] + \frac{V_0}{2}\left[1 - A_b(z)\right]$$

Kde A(z) odpovídá přenosové funkci fázovacího článku:

$$A_b(z) = -\frac{a_b + z^{-1}}{1 + a_b z^{-1}} \tag{3.25}$$

Transformace DP-PP pro tento případ probíhá podle vztahu: [12]

$$z^{-1} \to -z^{-1}\frac{z^{-1} + d}{dz^{-1} + 1} \tag{3.26}$$

Po transformaci přenosové funkce 3.7 získáváme přenosovou funkci peak filtru:

$$H_p(z) = \frac{1}{2}\left[1 + A_{bc}(z)\right] + \frac{V_0}{2}\left[1 - A_{bc}(z)\right] \tag{3.27}$$

kde A(z) odpovídá přenosové funkci fázovacího článku druhého řádu [15].

$$A(z) = \frac{z^{-2} + d(1 - a_{bc})z^{-1} - a_{bc}}{1 + d(1 - a_{bc})z^{-1} - a_{bc}z^{-2}} \tag{3.28}$$

Parametry a_{b},a_{c} ovlivňující šířku pásma odpovídají stejně označeným frekvenčním parametrům low-shelving filtru 3.13 3.14.

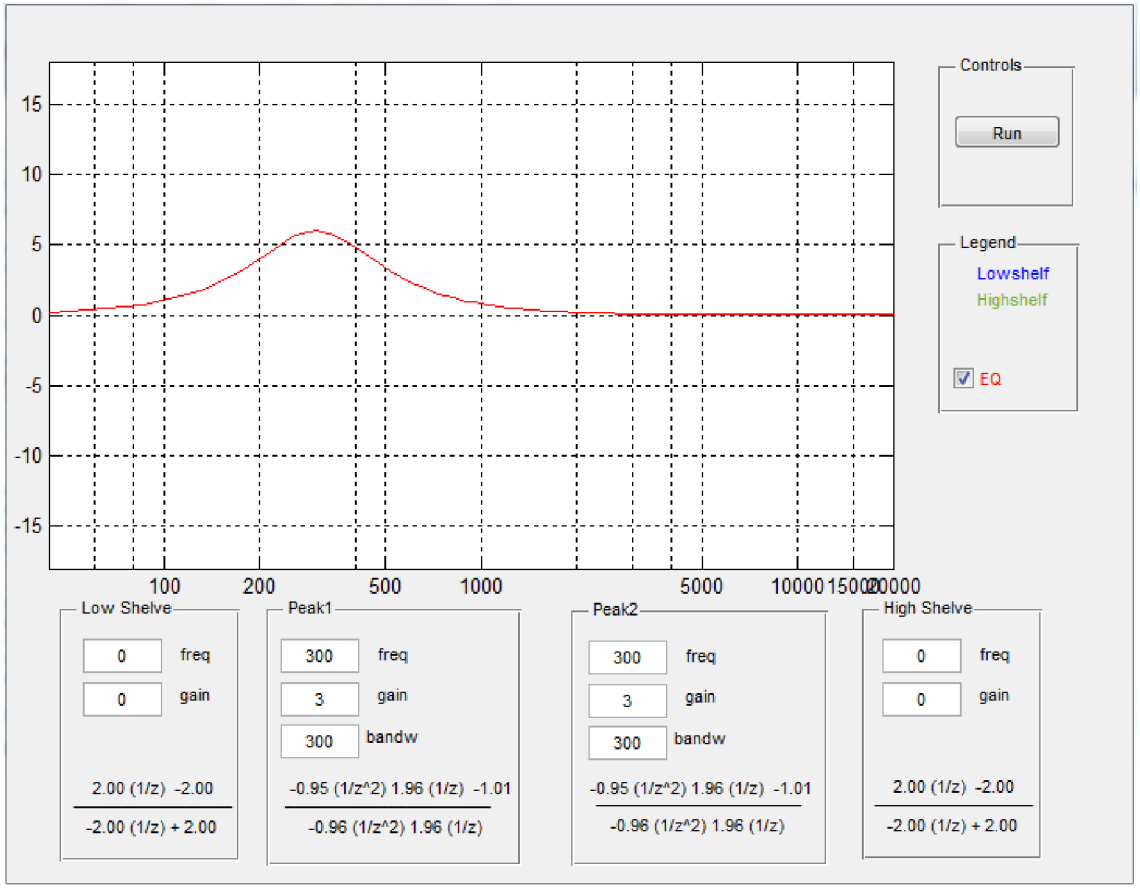
$$a_b = \frac{\tan(\frac{\omega_c T}{2}) - 1}{\tan(\frac{\omega_c T}{2}) + 1}$$

$$a_c = \frac{\tan(\frac{\omega_c T}{2}) - V_0}{\tan(\frac{\omega_c T}{2}) + V_0}$$

Parametr d určující střední frekvenci navrženého filtru odpovídá vztahu: [12, 15]

$$d = -\cos(\omega_d) \tag{3.29}$$

kde ω_d je digitální frekvence $\omega_c T$.



Obr. 4.1: Výsledný program v prostředí Matlab

4 IMPLEMENTACE NAVRŽENÝCH FILTRŮ PARAMETRICKÉHO EKVALIZÉRU V PROSTŘEDÍ MATLAB

Součástí této práce je i program sloužící k vizualizaci kmitočtových charakteristik filtrů navržených v předchozí kapitole realizovaný pomocí prostředí Matlab . Tato část popisuje způsob realizace tohoto programu a jeho funkce.

4.1 Popis

Finální vzhled programu pro vizualizaci kmitočtových charakteristik je možné pozorovat na obrázku 4.1. Ovládání je intuitivní – po stisknutí tlačítka *Run* se provede výpočet koeficientů přenosových funkcí a zobrazení příslušných grafů. Dále je možné pomocí zaškrtávacího pole *Eq* v panelu *Legend* přepínat mezi zobrazením odezev jed-

notlivých filtrů a zobrazením celkové frekvenční odezvy parametrického ekvalizéru tvořeného těmito filtry. Přenosové funkce jsou zobrazeny na příslušných panelech.

4.2 Realizace

4.2.1 Implementace přenosových funkcí jednotlivých filtrů

Samotné filtry jsou implementovány pomocí jednoduchých funkcí vracejících koeficienty přenosové funkce daného filtru v závislosti na vstupních parametrech. Program obsahuje vlastní funkci pro každý typ filtru v souborech *lowshelve.m*, *highshelve.m* a *peak.m* .

4.2.2 Frekvenční odezva filtrů

Kmitočtové odezvy těchto filtrů jsou získány prostřednictvím funkce freqz:

```
[h,f] = freqz(b,a,n,fs)
```

Tato funkce přejímá jako vstupní parametr koeficienty přenosové funkce, délku návratových vektorů n (počet vzorků) a vzorkovací frekvenci f_s. Na základě těchto vstupů vrací vektor h kmitočtové odezvy a odpovídající frekvenční vektor f.

4.2.3 GUI

Za účelem snadného pozorování kmitočtových odezev a vlastností navržených filtrů bylo pro tento program vytvořeno jednoduché grafické uživatelské rozhraní pomocí editoru GUIDE. Jedná se o *WYSIWYG* editor fungující na principu *Drag and Drop* tedy prostém přetahování jednotlivých prvků z palety komponent do pracovního okna aplikace. Poté co je grafický návrh hotov, GUIDE sám vygeneruje všechny potřebné metody vratného volání obsluhující události v reakci na uživatelskou akci na dané komponentě.

K vlastnostem instancí jednotlivých stavebních prvků (tlačítka, zaškrtávací pole...) lze z MATLAB kódu přistupovat pomocí struktur nazvaných *handles*. Struktury handles je v práci užíváno zejména pro získání vstupních parametrů z editovatelných textových polí pro výpočet koeficientů přenosových funkcí.

Kód 1 Ukázka kódu pro výpočet koeficientů přenosové funkce low-shelving filtru.

```
function [N1,N2,D1,D2]= lowshelve(y,G,f)
%-------------------------------------------------------------------------
%funkce vrací koeficienty lowshelving filtru vypočtené na základě vstupních
%parametrů
%-------------------------------------------------------------------------
V=10^(G/20); %výpočet zesílení pásma propuštěného lowpass filtrem
T=1/44100;   %vzorkovací perioda
wc=2*pi*f;   %mezní uhlová frekvence
%-------------------------------------------------------------------------
if V>=1      %pro případ BOOST

ab=(tan(wc*T/2)-1)/(tan(wc*T/2)+1); %frekvenční parametr allpass filtru

N1=ab-1+V*ab + V;  %koeficienty výsledne přenosove funkce
N2=1-ab+V+V*ab;
D1=2;
D2=2*ab;
%-------------------------------------------------------------------------
elseif V<1   %pro případ CUT - je třeba upravit přenosovou funkci
                  %pro zachování symetrické kmitočtové charakteristiky
ac=(tan(wc*T/2)-V)/(tan(wc*T/2)+V); %parametr allpass filtru
N1=ac-1+V*ac + V;  %koeficienty vysledne prenosove funkce
N2=1-ac+V+V*ac;
D1=2;
D2=2*ac;

end
```

Kód 2 Použití funkce freqz.

```
[H,f] = freqz([b1,b2],[a1,a2],512,44100); %získání vektorů H a f
plot(handles.axes2,f,20*log10(abs(H))); %zobraz graf se zesílením v dB
```

5 TECHNOLOGIE VST 3

5.1 Virtual Studio Technology

Technologie VST(Virtual Studio Technology) společnosti Steinberg umožňuje integraci externích nástrojů či efektů v podobě zásuvného pluginu do programů pro editaci zvukových souborů(DAW). Díky tomu uživatel není závislý pouze na vnitřním vybavení jeho DAW, ale může využívat obrovské množství pluginů vytvářených různými vývojáři nezávisle na sobě. VST zásuvné moduly jsou vybaveny grafickým uživatelským rozhraním většinou emulujícím panely hardwarových efektů či nástrojů a není možné je využívat mimo hostitelskou aplikaci.

5.2 Struktura VST3 pluginu

Programová struktura VST3 pluginu se poměrně markantně liší od svého předchůdce – standardu VST2. Zatímco u VST2 byl celý námi programovaný plugin řešen pomocí jednoho hlavního objektu dědícího vlastnosti třídy *AudioEffectX*, VST3 přichází s řešením, při kterém je použito dvou vzájemně nezávislých komponent pro zpracování audio signálu a pro uživatelskou interakci označovaných jako Processor a Controller. Tyto dvě části jsou od sebe naprosto separovány, což přináší výhodu v tom, že hostitelská aplikace je může využívat v odlišných souvislostech. Tato struktura je realizována pomocí VST-MA (VST Module Architecture), což je technologie vycházející z Microsoft COM (Component Object Model), kdy je komunikace mezi jednotlivými softwarovými komponentami zprostředkována pomocí tzv. rozhraní. Poměrně dobře demonstruje výhodu tohoto uspořádání příklad související s automatizací parametrů pluginu, kdy processingová část musí upravovat parametry s frekvencí totožnou s vzorkovací frekvencí, ale úpravy ovládacích prvků GUI je možné provádět se zdaleka menší frekvencí což má za následek přesnější synchronizaci[5].

5.2.1 Component Object Model

Detailní popis architektury COM je nad rámec této práce, nicméně pro pochopení základních principů programování VST3 pluginů je nutné se jí alespoň zběžně zabývat. COM je dílem společnosti Microsoft a navazuje na technologii OLE (Object Linking and Embedding). S průběhem času, kdy se software stával čím dál tím větším a komplexnějším, již nebylo vhodné jej vyvíjet jako jednu monolitickou aplikaci, ale bylo nutné jej rozkládat na jednotlivé funkční bloky - komponenty, které mezi sebou nezávisle komunikují. COM architekturu je tedy možné definovat jako způsob

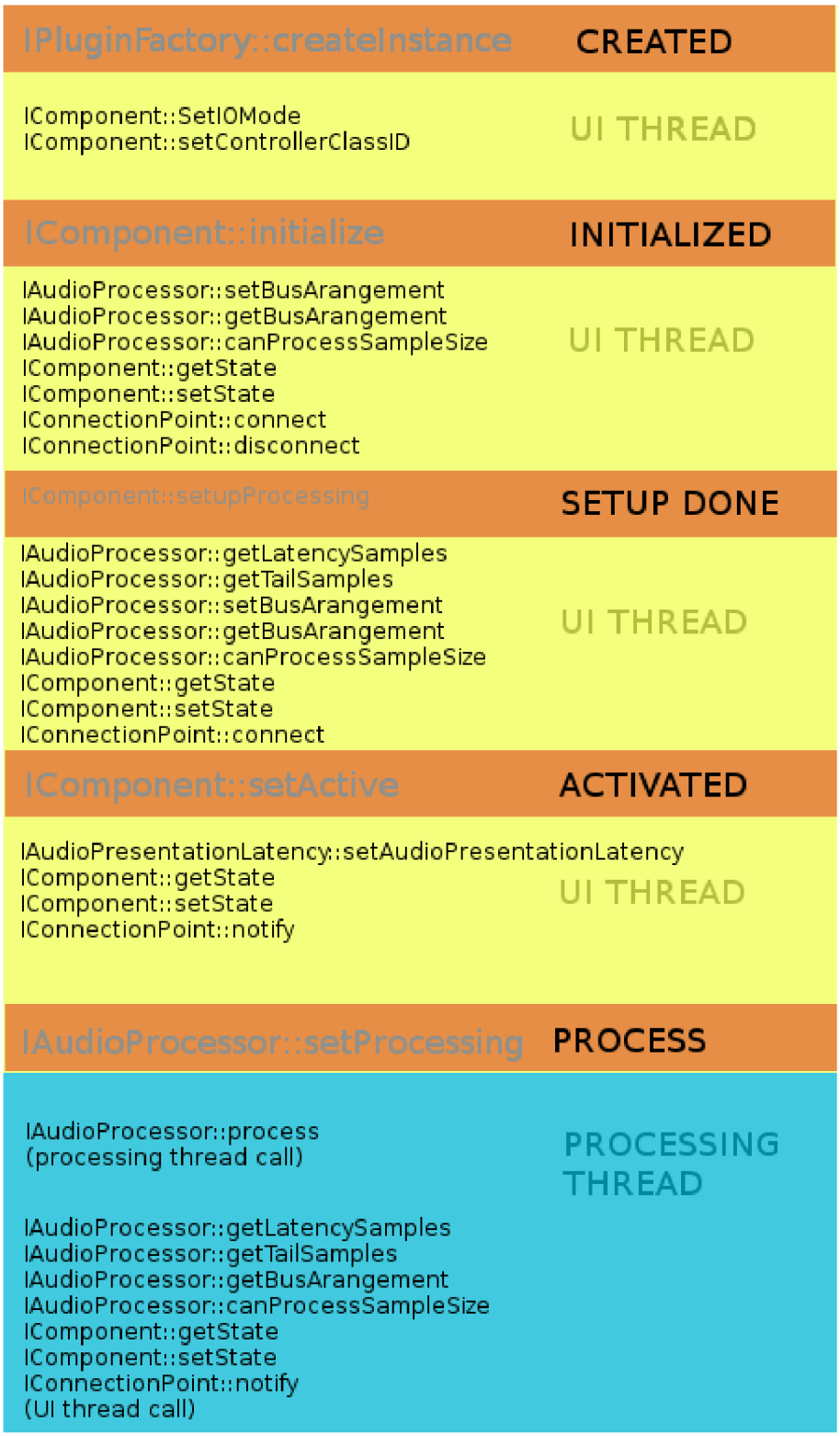
jak takovéto aplikace vytvářet. Jak již bylo zmíněno, jednotlivé komponenty mezi sebou komunikují pomocí rozhraní. COM rozhraní tedy deklaruje soubor funkcí, bez toho aby diktovalo způsob jejich implementace. V prostředí jazyka C++ probíhá implementace rozhraní zděděním jeho abstraktních metod do našeho objektu a jejich následnou definicí. Instance COM objektu je vytvořena zvláštní metodou, která je ve VST3 SDK nazvána *CreateInstance()*. Tato metoda vytváří nový objekt, ale nevrací ukazatel na něj, nýbrž ukazatel na jeho základní COM rozhraní. Každá instance takového objektu musí být jednoznačně identifikována – to se v COM děje pomocí 128 bitového unikátního identifikátoru GUID (ve VST3 SDK fUID), který je získán například pomocí programu Microsoft GUID Generator. [3] [6]

5.2.2 Processor

Mimo samotné zpracování signálu zodpovídá *processor* za odpovědi na požadavky hostitelské aplikace týkajících se počtu kanálů a audio formátu, deklaraci vstupů pluginu, reakce na změny parametrů přímo ovlivňujících processing (změna parametru v GUI; reakce na MIDI příkaz) a ukládání/načítání stavu parametrů pluginu – tzv. serializace (parametry jsou ukládány do souboru za sebe, tedy do série). V naší aplikaci dědí Processor vlastnosti třídy *AudioEffect* (jediná souvislost s třídou *AudioEffectX* u VST2 je podobnost jmen). [3] Komponenta *processor* je realizována dvěma základními rozhraními: *IComponent*, rozhraní společné oběma komponentám a zodpovědné mimo jiné za popis sběrnic a jejich směrování a *IAudioProcessor* provádějící samotné zpracování zvukového signálu a úkony s tím spojené. Důvodem tohoto rozdělení je snaha vývojářů SDK o případnou možnost v budoucnu snadno implementovat i zpracování jiných druhů signálů např.: videa [5].

Diagram aktivit

Průběh existence komponenty Processor od vytvoření po jeho samotnou funkci znázorňuje workflow diagram na Obr.8. VST modul musí obsahovat „výrobní třídu“ (class factory) obsahující definici metadat a metod určených k instantizaci komponent pluginu. K této třídě má host přístup přes rozhraní *IpluginFactory*. Následuje inicializace komponenty (*IComponent::initialize*) a offline nastavení processingu, které probíhá v případě, že plugin je ještě v neaktivním stavu(*setupProcessing*). Přechod k samotné realtime processingové metodě (process call) probíhá prostřednictvím *IAudioProcessor::setProcessing*. [5]



Obr. 5.1: Workflow diagram processingové komponenty

5.2.3 Controller

Tato komponenta zodpovídá především za inicializaci a nastavení GUI a implementaci komunikačního mechanismu pro posílání či příjem informací z/do GUI. Mimo to se také stará o automatizaci parametrů, serializaci (pouze čtení) a nastavení MIDI kontrolerů. Dědí vlastnosti nadřazené třídy *EditController*. Hlavní rozhraní nese název iEditController. *IComponentHandler* je druhé důležité rozhraní sloužící ať už pro komunikaci s hostitelskou aplikací nebo pro komunikaci s procesorem [3].

Grafické uživatelské rozhraní

Pro vytváření uživatelských rozhraní obsahuje VST SDK soubor tříd a metod VSTGUI, které obsahuje definice základních nejčastěji používaných ovládacích prvků. V případě potřeby jejich rozšíření je samozřejmě možné vytvořit vlastní objekt dědící z některé ze základních tříd těchto prvků a podle potřeb jej doplnit či upravit. Základní třídou uživatelského rozhraní je třída VSTGUIeditor. Pokud VSTGUI nevyhovuje předpokladům pro vyvíjený plugin (například při požadavku na detailnější vykreslování grafických primitiv) je možné použít i externí grafický framework jako např.: JUCE library, který mimo jiné umožňuje i jednoduché vytvoření OpenGL zobrazovače [7].

5.2.4 Komunikace mezi komponentami

Vzhledem k tomu, že VST3 plugin je tvořen dvěma nezávislými komponentami, je nutné, aby tyto komponenty měly možnost mezi sebou sebou komunikovat. VST3 platforma rozlišuje dva druhy takovéto komunikace – standardní (přenos parametrů) a privátní (přenos zpráv neznámých hostitelské aplikaci). Je úlohou hostitelské aplikace tuto komunikaci zprostředkovat.

Standardní komunikace - VST parametry

VST parametry jsou základním způsobem předávání uživatelem definovaných hodnot parametrů processingu. Host přijímá a předává tyto hodnoty parametrů mezi komponentami v normalizované podobě (rozsah 0.0 – 1.0) a přepočítání na vlastní parametry daného DSP systému probíhá až v příslušných komponentách. Ve VST modulech jsou parametry reprezentovány jako instance třídy *Steinberg::VST::Parameter*, které je nutné následně přidat do objektu typu *ParameterContainer*. Každý objekt parametru je určený unikátním identifikátorem přiřazeným již v konstruktoru a pro správnou funkci je nutné aby se toto ID již dále neměnilo. Je možné nastavit vlastnosti parametrů jako například krok (*StepCount*) či přiřadit daný parametr k určité jednotce (*Unit*). [5]

Přenos parametrů mezi komponentami je základním druhem komunikace, bez které se neobejde žádný plugin umožňující uživateli nastavovat parametry zpracování signálu. Každá uživatelská změna parametru je kontrolerem oznámena hostitelské aplikaci prostřednictvím rozhraní IcomponentHandler. Host samotný je nadále zodpovědný za předání této změny komponentě Processor pomocí rozhraní IParameterChanges, které je používáno k přenášení všech změn parametrů, které mají být aplikovány na právě zpracovávaný blok vzorků. [5]

Privátní komunikace

Přenos dat, která jsou hostitelské aplikaci neznámá, je možný pomocí zpráv přenášených mezi komponentami. Komunikace samotná je zprostředkována hostem pomocí rozhraní *Steinberg::VST:IConnectionPoint*. Každá zpráva (*Steinberg::VST::IMessage*) obsahuje také seznam atributů (*Steinberg::VST::IAttributeList*). Tento způsob komunikace by však na straně processoru neměl být používán během samotného zpracovávání signálu, neboť by mohl mít za následek zpomalení a neschopnost pluginu zpracovávat vstupní signál v reálném čase. [5]

5.3 Nové funkce ve VST3

Díky novému konceptu programové struktury VST3 aplikací se objevují i nové možnosti, které tento standard přináší oproti předchozím verzím. Díky odlišnostem v přistupování k jednotlivým prvkům daného zásuvného modulu je tak možné vyvíjet pluginy, které jsou modernější, ekonomičtější a v neposlední řadě také pluginy, které nabízejí uživatelům mnohem komfortnější a efektivnější způsob jejich používání.

5.3.1 Dynamické přiřazování vstupů a výstupů

VST3 pluginy již nejsou limitovány pevně nastaveným počtem vstupů a výstupů a můžou se dynamicky adaptovat podle potřeb uživatele. Pokud tedy např.: vložíme daný zásuvný modul na stopu v DAW, sám upraví počet svých vstupů a výstupů podle typu této stopy (mono,stereo,5.1). [8]

5.3.2 Správa sběrnic

U softwarových nástrojů s velkým množstvím výstupů (samplery, simulátory bicích sestav) vzniká problém s nevyužitými výstupy, které zbytečně zatěžují hostitelskou aplikaci a zabírají místo. S VST3 je možné tyto sběrnice deaktivovat a a použít je až tehdy, kdy jich je zapotřebí. [8]

5.3.3 Logické seskupování parametrů pluginu

Parametry pluginů je možné seskupovat do skupin odpovídajících jejich využití. Například parametry filtru jako mezní frekvence či Q faktor je možné sjednotit do skupiny „Filter“ což má za následek lepší orientaci při práci s modulem. Určité skupině je možné přiřadit i konkrétní MIDI kanál. [8]

5.3.4 Midi

Nové funkce se uplatňují i co se týče spolupráce modulu s protokolem MIDI. Pluginy verze 3 mohou na rozdíl od předchozí verze využívat více než jeden MIDI vstup či výstup najednou. Za zmínku stojí také technologie VSTXML využitelná pro zobrazování parametrů VST modulu na displeji MIDI kontroleru. [8]

5.4 Konvence VST3 programování

Před tím než přejdeme k samotné implementaci navržených filtrů pomocí VST3 SDK je vhodné uvést některé zažité konvence vztahující se k tomuto SDK.

Prostory jmen

Každá komponenta zapouzdřuje svůj obsah pomocí tří příkazů namespace – Steinberg, VST a jménem daného modulu, tím je ve zdrojových souborech VST3 modulů předcházeno kolizím jmen tříd a jejich metod. [3]

Názvy konstant

Konstanty jsou ve VST3 SDK definované jak vlastní hodnotou, tak jménem s přidaným písmenem k na začátku (kGain, kFreq), samozřejmě není nutné tuto konvenci následovat, nicméně pro lepší orientaci je nutné ji brát na vědomí. [3]

Návratové metody typu tResult

Většina VST metod má návratovou hodnotu tResult. Jedná se jednoduše o 32-bitovou proměnnou typu integer. Běžnými konkrétními hodnotami jsou kResultTrue a kResultFalse, tedy návratové hodnoty signalizující úspěšné či neúspěšné proběhnutí dané metody. [3]

5.5 Implementace navržených filtrů do VST3 zásuvného modulu

Pro lepší přehlednost se následující podkapitola omezuje pouze na stěžejní body implementace navržených filtrů. Podrobnější náhled na problematiku lze získat nahlédnutím do přiloženého komentovaného zdrojového kódu výsledného modulu.

5.5.1 Třída Filter

Pro programovou implementaci výše navržených filtrů byla vytvořena třída Filter umožňující vytvářet instance jednotlivých objektů filtrů a obsahující metody pro jejich následnou obsluhu. Následující podkapitola se věnuje stručnému popisu této třídy a jí přidružených metod. Deklarace samotné třídy a jejích metod, struktur a proměnných je možné nálezt v hlavičkovém souboru *filter.h*, zatímco přímé definice jsou obsaženy v hlavním souboru třídy *filter.cpp*. Třída byla koncipována tak, aby její instance kompletně pokryly obsluhu daných filtrů, od inicializace, přes přepočítávání koeficientů až po samotné zpracování signálu. Stejně tak bylo cílem, aby v případě potřeby bylo možné měnit základní vlastnosti objektu, jako např.: typ daného filtru, přímo za běhu programu bez nutnosti vytvářet novou instanci objektu. Výsledná třída je zapouzdřená ve vlastních souborech a neobsahuje žádné reference na VST SDK a je tedy snadno přenositelná a použitelná i v jiných projektech, které nutně nemusí mít charakter zásuvného VST modulu.

Reference třídy Filter

Filter() – defaultní konstruktor, vytváří nový objekt typu Filter
float G,f,q – proměnné základních parametrů daného objektu filtru
registr reg – pomocná struktura sloužící pro uchování obsahu zpožďovacích členů filtru
char filtertype – určuje typ filtru
void Filter::Recalculate(double Samplerate) – metoda sloužící pro přepočítání koeficientů v závislosti na daných parametrech filtru
float Filter::Process(float input) – metoda sloužící pro vlastní zpracování vstupního signálu, na základě vstupu vrací odpovídající přefiltrovaný vzorek
void Filter::setLowshelf() – pomocná metoda sloužící ke změně typu filtru, stejným způsobem pracují i metody setHighshelf() a setPeak()

Filtrační algoritmus

Samotné filtrování vstupu zajišťuje již zmíněná metoda Process. Vzhledem k tomu, že řád navržených filtrů není vyšší než druhý, je možné filtry implementovat přímo v jejich první přímé formě. Výsledný signál je poté tedy určen přímo dosazením do diferenční rovnice daného filtru. Pro filtr řádu n je nutné znát n předchozích hodnot vzorku, tedy implementovat n paměťových členů. Pro tyto účely byla vytvořena jednoduchá struktura *registr*, ve které jsou při zpracovávání signálu uchovávány hodnoty předchozích výstupních a vstupních vzorků. Pokud filtr není aktivní nebo je nastaveno nulové zesílení v rámci úspory výpočetního výkonu metoda rovnou vrací vstupní hodnotu.

Řešení kritických přístupů

Není vhodné, aby byla volána metoda Filter::Recalculate v průběhu metody Filter::Process a naopak. Důvod je zřejmý – pokud dojde ke změně koeficientů v průběhu filtrování daného vzorku výstup bude zkreslený. Stejný efekt bude mít i volání filtrační funkce v momentě, kdy ještě nebyly přepočítány všechny koeficienty. Je tedy vhodné zamezit tomu, aby více vláken mohlo sdílet dané prostředky zároveň. K tomuto účelu jsou použity mutexové zámky z volně šiřitelné knihovny TinyThread++. Tato knihovna byla vybrána kvůli přenositelnosti a snadné implementaci (nejsou potřeba žádné externí knihovny). Při používání těchto zámků u audio aplikací je však nutné dbát na to, aby nedošlo k nežádoucímu blokování metody pro zpracování signálu, což by mělo za následek narušení funkce dané aplikace v reálném čase. [5]

5.5.2 Vytvoření zásuvného modulu

Jako šablona sloužící jako výchozí bod pro implementaci výše navržených filtrů do VST3 zásuvného modulu byl zvolen příkladový plugin Again, který je přímo součástí VST3 SDK. Rozdělení komponent je patrné z názvu jednotlivých souborů –PQ.cpp (komponenta Processor), PQcontroller.cpp (komponenta Controller) a PQeditor.cpp (GUI).

5.5.3 Processor *(PQprocessor.cpp)*

Mimo samotného zpracování audiosignálu obdrženého z hostitelské aplikace zodpovídá i za prvotní vytvoření a inicializaci banky filtrů a přepočítávání koeficientů filtrů v závislosti na parametrech obdržených z kontroleru.

Získání parametrů z kontroleru

Způsob komunikace mezi kontrolerem a processorem byl nastíněn v kapitole výše. Realizace samotné aktualizace změněných parametrů daného filtru probíhá přímo v metodě PQ::Process a tedy je prováděna vždy pro daný blok audio dat. Parametry jsou postupně získávány z fronty – objektu typu *IParamValueQueue*, která je obdržena pomocí metody *IparameterChanges::getParameterData*. Po získání VST parametru je podle jeho ID rozhodnuto (*IParamValueQueue::getParameterID*) jakým způsobem se přepočítá na konkrétní parametr filtru (mezní frekvence, zesílení apod...) a je zavolána funkce *Filter::Recalculate(double Samplerate)*. Vzorkovací frekvence potřebná pro správné přepočítání koeficientů filtru je získána ze struktury *ProcessContext*, která obsahuje informace o právě zpracovávaném bloku vzorků.

Serializace

Pro obsluhu serializace slouží metody *GetState* a *SetState* na straně procesoru a metoda *GetCompononetState* na straně kontroleru. Data jsou při načítání/ukládání presetů řazena za sebe do streamu typu *IBstream*. Pokud host zavolá funkci *SetState* v processingové části pluginu, automaticky volá i metodu GetCompononentState v kontroleru, čímž zajišťuje synchronizaci mezi oběma komponentami [5] .

Zpracování signálu

Samotné zpracování signálu, tedy hlavní funkce procesorové komponenty, probíhá opět v metodě PQ::Process. Vstupní data jsou zpracována postupně v cyklu for, tedy výstup daného filtru kaskády je přiváděn na vstup následujícího filtru až do dosažení posledního filtru. Podle počtu kanálů na vstupu (*data.inputs.numchannels*) je rozhodnuto, zda je třeba zpracovat MONO signál či STEREO signál, tedy zda je nutné získat jeden vstupní buffer či dva (levý a pravý). Zpracování vstupního vzorku je přenecháno danému objektu typu filter (*Filter::Process*) a vrácená hodnota je přiřazena do odpovídajícího výstupního bufferu.

5.5.4 Controller *(PQcontroller.cpp)*

V tomto konkrétním případu tato komponenta zodpovídá hlavně za vytvoření a inicializaci VST parametrů a zpracování uživatelských požadavků z GUI. V hlavním souboru třídy probíhá pouze deklarace a definice parametrů, aktualizace parametrů probíhá taktéž prostřednictvím controlleru, nicméně se k němu přistupuje z objektu editoru. Jednotlivé hodnoty unikátních identifikátorů použitých parametrů jsou definovány v souboru *PQparamids.h*.

5.5.5 Editor *(PQEditorView.cpp)*

Uživatelské rozhraní neboli editor slouží k uživatelské interakci, v tomto případě tedy k nastavování parametrů jednotlivých filtrů.

Ovládací prvky

Jako ovládací prvky byly zvoleny modely otočných potenciometrů – instance třídy CanimKnob. Animace pohybu prvků je řešena pomocí stripů obrázků ve formátu PNG. Tento strip byl vygenerován zdarma dostupným programem KnobMan. Druhým ovládacím prvkem a zároveň indikátorem nastavované hodnoty jsou textová pole (CTextEdit). Pro získání normalizované či denormalizované hodnoty parametru slouží metody fromString a toString definované pro různé typy parametrů v kontroleru.

Přiřazení hodnot parametrům controlleru

Aktualizace hodnoty parametrů probíhá v metodě PQEditorView::ValueChanged. Na základě tagu, který je unikátní pro danou instanci ovládacího prvku je pomocí metod setParamNormalized a performEdit aktualizována hodnota daného parametru v controlleru, který ji následně může předat hostitelské aplikaci a ta následně komponentě processoru. Aktualizace hodnot ovládacích prvků v případě, že editor obdrží nový soubor parametrů (například při načtení presetu nebo při použití automatizační stopy v hostitelské aplikaci), probíhá v metodě *PQEditorView::update*.

6 ZHODNOCENÍ VÝSLEDKŮ PRÁCE

6.1 Zhodnocení použité metody pro návrh parametrických filtrů

Metoda dekompozice přenosové funkce filtru na fázovací článek při návrhu parametrických filtrů s sebou nese řadu výhod. Nejdůležitější vlastností takto navržených filtrů je vysoká efektivnost co se týče implementace, kdy je pro realizaci IIR filtru N-tého řádu potřeba pouze N koeficientů přenosové funkce oproti 2N+1 koeficientům při přímém návrhu. [12] Díky tomu jsou filtry navržené touto metodou méně náročné na výpočetní výkon. S tím souvisí i fakt, že jednotlivé filtry navržené v této práci je možné realizovat vždy za použití jedné konkrétní přenosové funkce all-pass filtru.

Značnou výhodou je i modularita výsledných struktur, kde je možné fázovací články používat jako stavební bloky – nahrazovat, řadit do kaskád, a poměrně rychle a efektivně škálovat vlastnosti filtru tak, aby odpovídaly požadavkům návrhu. [16]

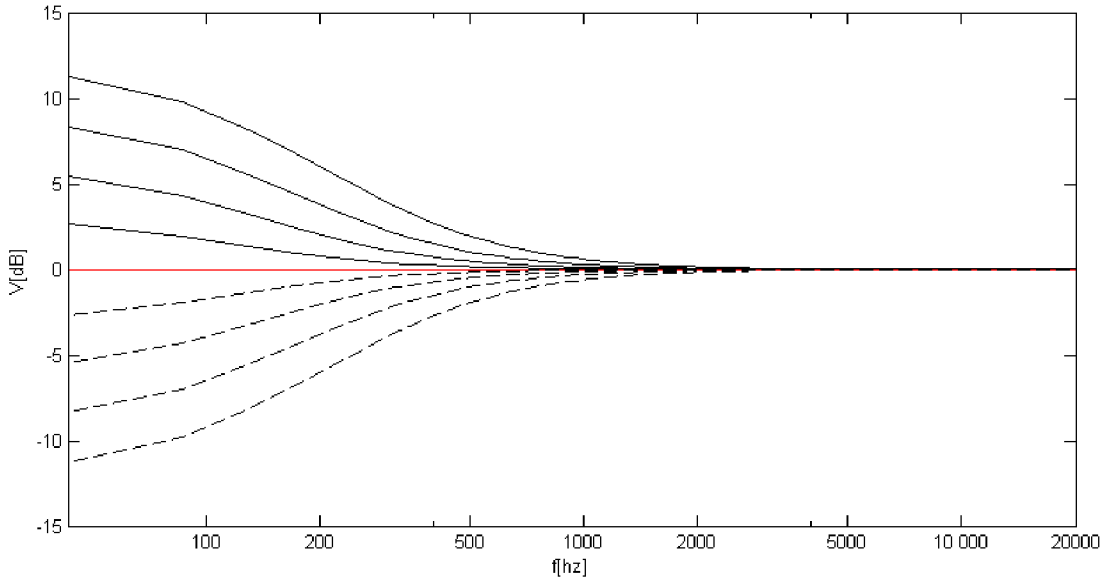
Jak je možné si všimnout, jednotlivé filtry jsou realizované stejnou strukturou jak pro případ cut ($V_0 < 0$), tak pro případ boost ($V_0 > 0$). To s sebou nese zřejmou výhodu ve snažší finální implementaci, nicméně na druhou stranu tento fakt znemožňuje vypočítat parametry dané přenosové funkce nezávisle na sobě. [15] Pro příklad uveďme rovnici 3.14 znázorňující vztah pro výpočet frekvenčního parametru low-shelving filtru pro případ cut, kde je možné pozorovat že parametr a_c závisí na velikosti zesílení V_0.

$$a_c = \frac{tan(\frac{\omega_c T}{2}) - V_0}{tan(\frac{\omega_c T}{2}) + V_0}$$

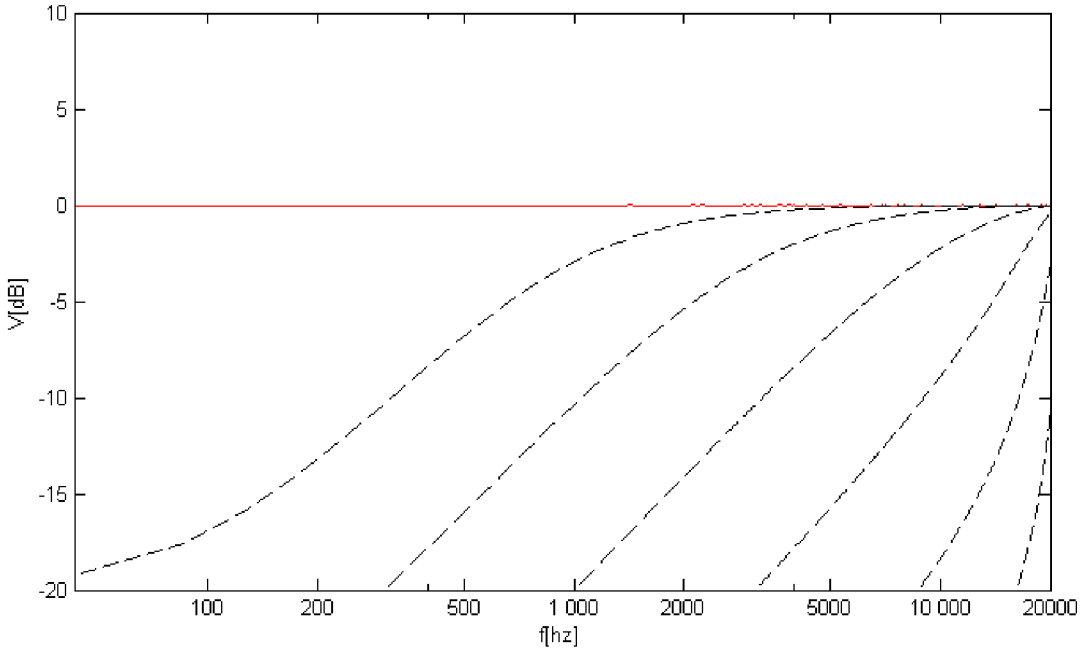
6.2 Frekvenční odezvy navržených filtrů

6.2.1 Low-shelving filtr

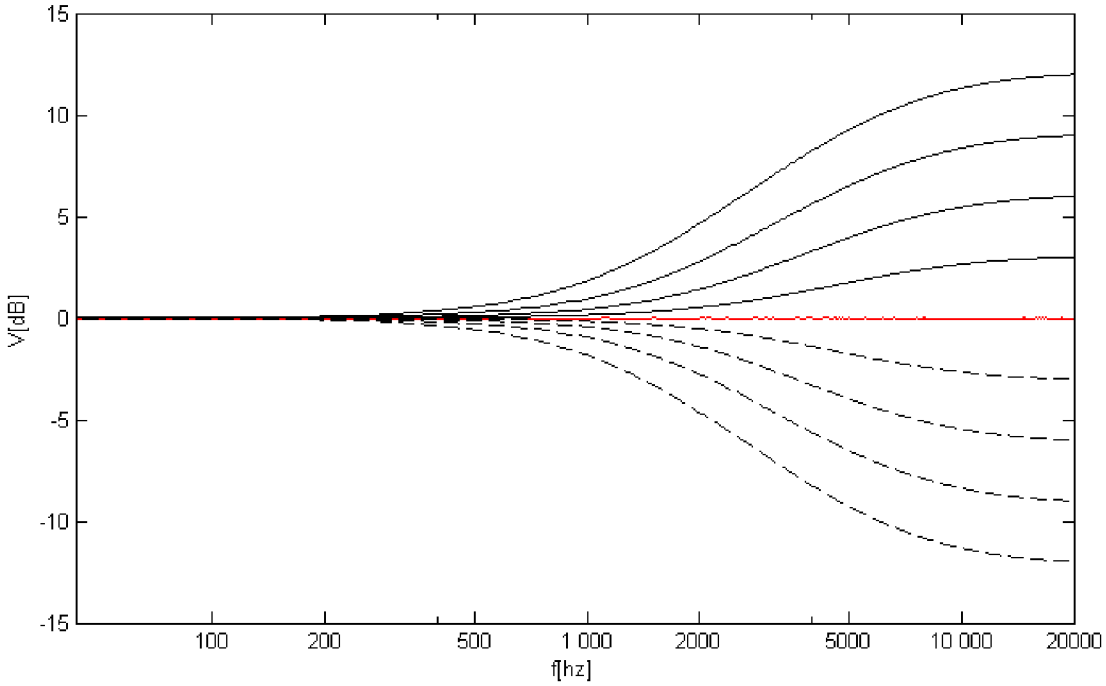
Frekvenční odezvu low-shelving navrženého v kapitole 3 můžeme pozorovat na Obr. 6.1. Je patrné, že tvarování kmitočtového spektra zasahuje poměrně hluboko za nastavený mezní kmitočet filtru. Pro dosažení strmější modulové charakteristiky by bylo možné použít filtr vyššího řádu, v ideálním případě uživateli nabídnout možnost volby mezi více řády pro větší kontrolu nad nízkými kmitočty. Funkce filtru při nastavení větších hodnot útlumu je ilustrována na Obr. 6.7. Při hodnotě $V_0 = -75$ již dosahuje filtr útlumu přes $20dB$ v celém frekvenčním spektru.



Obr. 6.1: Frekvenční odezva navrženého low-shelving filtru.



Obr. 6.2: Frekvenční odezva navrženého low-shelving filtru pro vysoké hodnoty útlumu V=20,30,40,50,60,70 [dB]



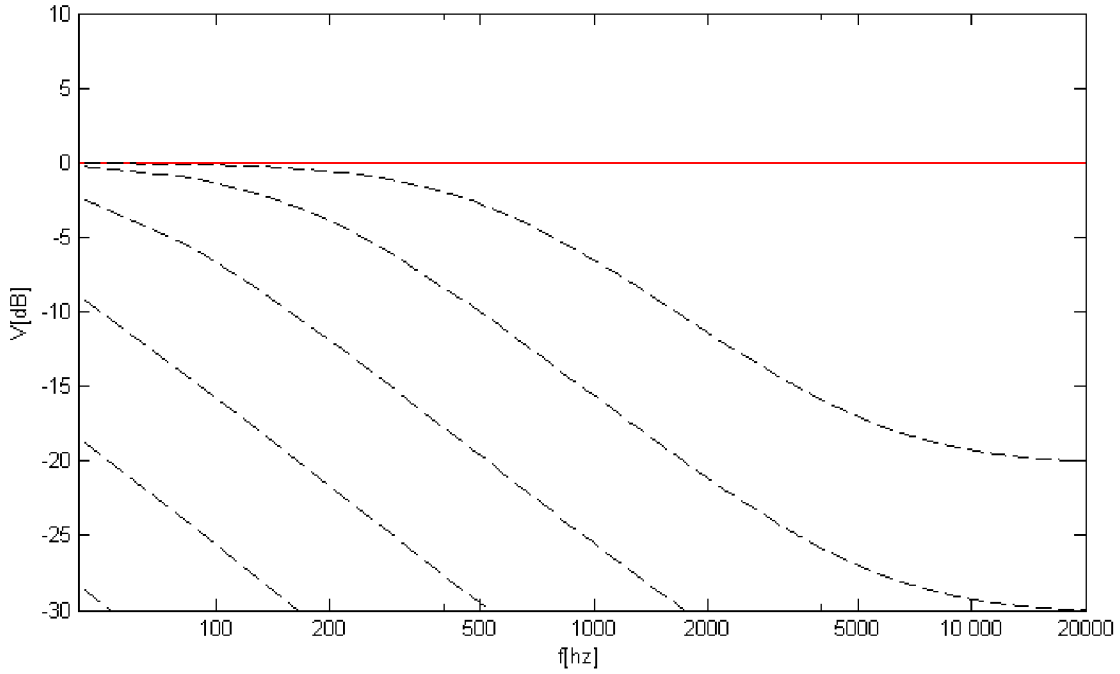
Obr. 6.3: Frekvenční odezva navrženého high-shelving filtru.

6.2.2 High-shelving filtr

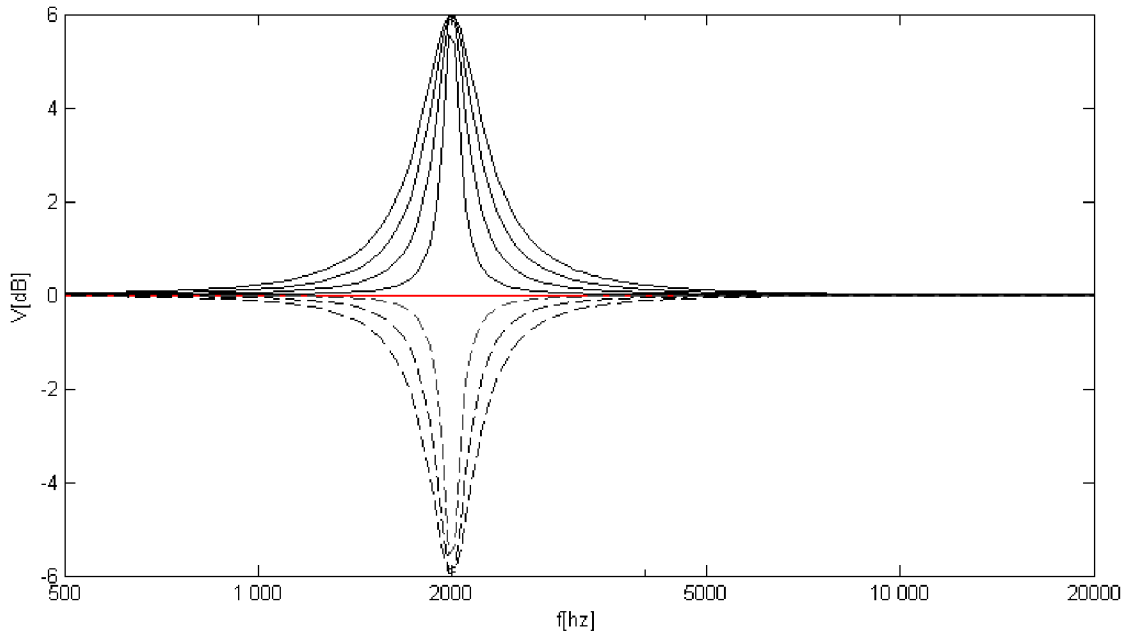
Pro high shelf filtr platí v podstatě stejné poznatky, které byly uvedeny v souvislosti s filtrem typu low-shelving. Příslušné charakteristky lze pozorovat na Obr. 6.3 a 6.4.

6.2.3 Peak filtr

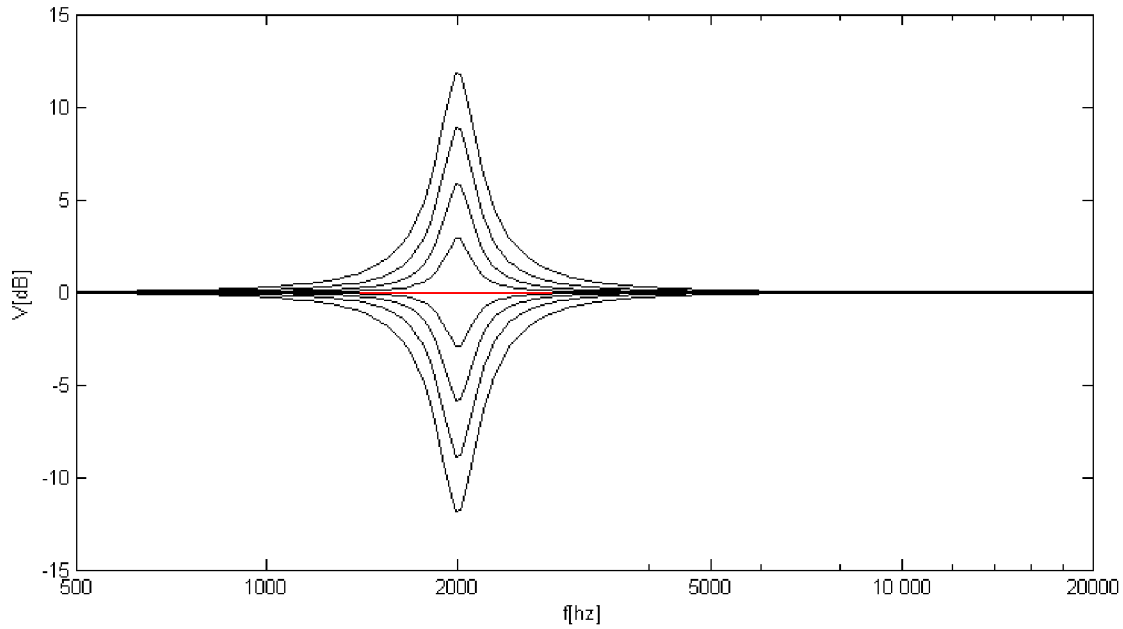
Frekvenční odezvu peak filtru navrženého v kapitole 3 můžeme pozorovat na Obr. 6.5 a Obr. 6.6. Možným vylepšením pro finální implementaci do zásuvného modulu by bylo zavedení parametru Q jako poměru střední frekvence f_c a šířky pásma f_b. [15] Funkce filtru při nastavení větších hodnot útlumu je ilustrována na Obr. 6.1. Při hodnotě $V_0 = -75$ dosahuje filtr útlumu přes $20dB$ v celém frekvenčním spektru.



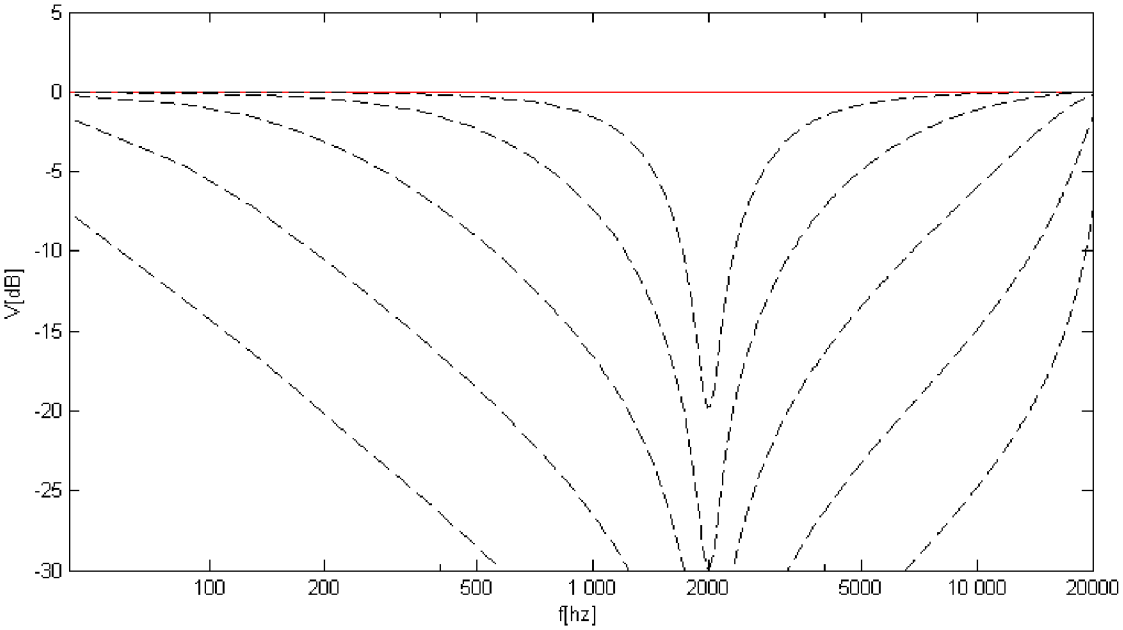
Obr. 6.4: Frekvenční odezva navrženého high-shelving filtru pro vysoké hodnoty útlumu V=20,30,40,50,60,70 [dB]



Obr. 6.5: Frekvenční odezva navrženého peak filtru pro střední frekvenci f_c=2000 [hz] , útlum $V = +-6[dB]$ a šířku pásma f_b=100,200,300,400 [hz]



Obr. 6.6: Frekvenční odezva navrženého peak filtru pro střední frekvenci f_c=2000 [hz] , šířku pásma f_b=200 [hz] a útlum $V = +-3, 6, 9, 12[dB]$



Obr. 6.7: Frekvenční odezva navrženého peak filtru pro vysoké hodnoty útlumu V=20,30,40,50,60,70 [dB]

Obr. 6.8: Výsledná podoba vytvořeného pluginu.

6.3 Vytvoření VST3 zásuvného modulu

Implementace navržených filtrů využitím technologie VST3 byla úspešná. Protože cílem práce nebyla jen implementace algoritmu pro zpracování signálu, ale i seznámení s principy VST3 programování, výsledný plugin implementuje i sice pro processing nedůležité, avšak pro v praxi použitelný zásuvný modul nezbytné funkce jako serializaci či možnost automatizace parametrů. Díky koncepci přenositelné třídy Filter je výsledný projekt připravený na případná rozšíření.

6.3.1 Možná vylepšení

Zobrazení frekvenční odezvy filtrů

VSTGUI sice nabízí možnost vykreslování geometrických tvarů a čar () nicméně neumožňuje vytvářet vyhlazené linky, což jej činí poměrně nevhodným pro vykreslování grafů frekvenčních odezev. Možným řešením by bylo použití externího frameworku pro uživatelská rozhraní, například framework JUCE++, který mimo jiné umožňuje vytvoření objektu view používajícího pro vykreslování knihovnu OpenGL. [7] Přenos jednotlivých hodnot mezi komponentou procesoru a kontroleru je možné realizovat buď opět použitím VST parametrů a nebo lépe pomocí privátní komunikace(*IMessage*) s tím, že jednotlivé hodnoty by byly zasílány v určitých časových parametrech pomocí časovače realizovaného vně metody*IAudioProcessor::process*. [5]

Dezippering

Pro eliminaci zvukových artefaktů, které můžou vznikat při skokové změně parametru filtru by bylo možné implementovat algoritmus pro tzv. Dezippering – zjemnění přechodu postupnou aproximací parametru. [5]

Podpora filtrů vyšších řádů

Jelikož navržené filtry nedosahují vyššího řádu než druhého, je možné je implementovat přímo prostřednictvím diferenční rovnice a paměťových (zpožďovacích) členů. V případě implementace vyšších řádů filtrů je tento přístup z hlediska zpracování signálu v reálném čase nevyhovující. Proto je nejpoužívanějším způsobem implementace vytvoření několika fitrů druhého řádu tzv. Biquadů jejichž kaskádováním při správném přepočítání koeficientů vznikne filtr požadovaného řádu. [4]

7 ZÁVĚR

V první části této práce bylo cílem seznámit se s návrhem parametrických shelving filtrů a poté tyto znalosti a poznatky aplikovat při návrhu jednotlivých typů filtrů parametrického ekvalizéru. Z důvodu volby nepřímého návrhu těchto filtrů bylo třeba nastudovat transformační vztahy bilineární transformace mezi spojitou a diskrétní časovou oblastí a následně se seznámit s možností realizace elementárních filtrů pomocí fázovacího článku. Zvolená metoda dekompozice přenosové funkce na fázovací článek byla shledána efektivní, jak z hlediska návrhu, tak i pozdější implementace (blíže popsáno v kapitole 6).

Pro ověření funkce filtrů a snažší pozorování jejich vlastností byl vytvořen program v prostředí Matlab umožňující sledovat modulovou frekvenční charakteristiku a jednotlivé přenosové funkce parametrického ekvalizéru v závislosti na zadaných parametrech jednotlivých filtrů.

V poslední fázi práce proběhlo seznámení s programovou strukturou VST3 zásuvných modulů a následná programová implementace navržených filtrů do podoby VST3 pluginu, který je možné nalézt v příloze B společně se zdrojovými soubory. Pro vytvoření modulu byl použit program Microsoft Visual Studio 2013, ladění a testování probíhalo v prostředí hostitelského DAW Steinberg Cubase 5. Implementace navržených filtrů byla úspěšná, výsledný plugin mimo samotného zpracování signálu implementuje i funkce nezbytné pro jeho možné využití v praxi. Možná vylepšení jako eliminace zvukových fragmentů postupnou aproximací parametrů nebo zobrazení frekvenčních odezev byla nastíněna v předchozí kapitole. V souladu s prezentovanými výsledky považuji stanovený cíl bakalářské práce za splněný.

LITERATURA

[1] SVOBODA, Zdeněk. VÍTOVEC, Jiří. *Matematika 2* [online]. 2014 Dostupné z URL: <http://www.umat.feec.vutbr.cz/~svobodaz/BMA2/bma2.pdf>.

[2] FAJMON, Břetislav. RŮŽIČKOVÁ, Irena. *Matematika 3* [online].

Dostupné z URL: <http://www.umat.feec.vutbr.cz/~hlavicka/skripta/matematika3.pdf>.

[3] PIRKLE, William C. Designing software synthesizer plug-ins in C: for RackAFX, VST3, and Audio Units. pages cm. ISBN 978-113-8787-070.

[4] DELIYANNIS, T, Yichuang SUN a J FIDLER. Continuous-time active filter design. CRC Press, 1999, ISBN 0849325730-.

[5] STEINBERG MEDIA TECHNOLOGIES *VST SDK Documentation* [online]. 2008, poslední aktualizace 22. 11. 2013 [cit. 10. 12. 2014]. Dostupné z URL: <http://www.steinberg.net/en/company/developers.html>.

[6] Dr. Dobbs Journal *The Component Object Model: Technical Overview* [online]. 1994, [cit. 10. 2. 2015]. Dostupné z URL: <http://www.cs.umd.edu/~pugh/com/>.

[7] *JUCE documentation* [online]. [cit. 10. 5. 2015]. Dostupné z URL: <https://www.juce.com/api/index.html>.

[8] STEINBERG MEDIA TECHNOLOGIES *VST3: New Standard for Virtual Studio Technology* [online]. [cit. 10. 12. 2014]. Dostupné z URL: <https://www.steinberg.net/en/company/technologies/vst3.html>.

[9] SKALICKÝ, Petr. *Digitální filtrace a signálové procesory.* Praha: ČVUT, 1995. ISBN 8001012441.

[10] ANTONIOU, Andreas. *Digital signal processing: signals, systems and filters.* New York : McGraw-Hill, 2005. ISBN 978-007-1454-247.

[11] OWSINSKI, Bobby. *The mixing engineer's handbook.* 2nd ed. Boston: Thomson Course Technology, 2006, xxi, 285 p. ISBN 15-986-3251-5.

[12] MITRA, Sanjit Kumar. *Digital signal processing: a computer-based approach.* 2th ed. New York: McGraw-Hill, 2011, xx, 940 s. ISBN 978-0072513783.

[13] CLAERBOUT, Jon F. Earth soundings analysis: processing versus inversion. Boston: Blackwell Scientific Publications, c1992, xvi, 304 p. ISBN 08-654-2210-9.

[14] SMÉKAL, Zdeněk. Číslicové filtry. 1. vyd. Brno: VUT, 1993, 136 s. ISBN 80-214-0500-7.

[15] ZÖLZER, Udo. Digital audio signal processing. 2. vyd. Chichester: John Wiley, 1997, x, 279 s. ISBN 04-719-7226-6.

[16] SARAMÄKI. IIR FILTERS: ADDITIONAL MATERIAL: Design of IIR filters using allpass filters as basic building blocks: Basic lecture notes. In: [online]. [cit. 2014-12-14].

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

Hz	Hertz - jednotka
dB	Decibel - jednotka
f_{vz}	vzorkovací kmitočet
V_0	Zesílení pásma
ω_{c}	Střední úhlová frekvence
ω_{d}	Digitální úhlová frekvence
a_{b}	Frekvenční parametr filtru pro případ boost
a_{c}	Frekvenční parametr filtru pro případ cut
d	Frekvenční parametr peak filtru
T	Vzorkovací perioda T
DSP	číslicové zpracování signálů – Digital Signal Processing
DAW	Digital Audio Workstation
VST-MA	VST - Module Architecture
COM	Component Object Model
VST	Virtual Studio Technology
MIDI	Musical Instrument Digital Interface
GUI	Graphical User Interface
WYSIWYG	What You See Is What You Get

SEZNAM PŘÍLOH

A PARAMETRICKÝ EKVALIZÉR V PROSTŘEDÍ MATLAB

V přiloženém archivu (elektronická verze) popřípadě na přiloženém CD (tištěná verze) se nachází adresář *MatlabEQ* obsahující soubory potřebné pro spuštění simulace parametrického ekvalizéru. Program lze spustit pomocí souboru GUI.m popřípadě zadáním příkazu GUI při současném nastavení pracovního adresáře Matlabu na adresář této přílohy. Dále lze v podadresáři Generators nalézt pomocné funkce využité pro generování frekvenčních odezev v kapitole 6. Program byl vytvořen ve verzi Matlabu R2014a.

B VÝSLEDNÝ MODUL A JEHO ZDROJOVÉ SOUBORY

V přiloženém archivu (elektronická verze) popřípadě na přiloženém CD (tištěná verze) se nachází adresář *VST3 implementace* obsahující zdrojové soubory (adresář *Zdrojové soubory*) a projekt vytvořeného pluginu (PQ). Součastí je i značně zredukovaná (ve smyslu redukce velikosti výsledneho archivu; byly odstraněny příklady a dokumentace) verze VST3 SDK pro usnadnění případného buildu projektu. Možné příčiny případných chyb kompilování projektu řeší soubor *readme.txt*. Mimo tyto soubory obsahuje tato příloha i výsledný soubor vytvořeného pluginu.