VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

REALIZACE ZVUKOVÉHO EFEKTU WAVESHAPER

IMPLEMENTATION OF WAVESHAPER AUDIO EFFECT

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

David Leitgeb

VEDOUCÍ PRÁCE

SUPERVISOR

doc. Ing. Jiří Schimmel, Ph.D.

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

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

Student: David Leitgeb ***ID:*** 218925
Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Realizace zvukového efektu Waveshaper

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte možnosti Audio toolboxu pro prostředí Matlab a prostředí JUCE s jejich pomocí, případně s pomocí dalších dostupných nástrojů, realizujte nelineární zvukový efekt typu waveshaper s editovatelnou převodní charakteristikou, možností filtrace a volby stupně převzorkování signálu. Efekt bude zpracovávat signál v reálném čase, pokuste se o jeho realizaci jako plug-in modul technologie VST.

DOPORUČENÁ LITERATURA:

[1] Zölzer, U.: DAFX - Digital Audio Effects (Second Edition). John Wiley & Sons, 2011. ISBN: 978-0-470-66599-2
[2] REISS, J., D., MCPHERSON, A., P. Audio effects: theory, implementation and application. Boca Raton: CRC Press, 2014, xiii, 353 s. : il. ISBN 978-1-4665-6028-4

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 31.5.2022

Vedoucí práce: doc. Ing. Jiří Schimmel, Ph.D.

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

ABSTRAKT

Cílem této bakalářské práce je realizace nelineárního zvukového efektu typu waveshaper. Ten se skládá z těchto základních bloků: uživatelem editovatelná převodní charakteristika, různé typy kmitočtové filtrace a několik stupňů převzorkování. Prototyp tohoto efektu byl nejprve realizován pomocí softwaru Matlab v kombinaci s jeho rozšířením Audio Toolbox. Z důvodu určitých omezení tohoto prototypu způsobených použitým prostředím byl následně celý efekt od základu přepsán do jazyka C++. Pro tuto implementaci byl využit framework JUCE, který je převážně používaný pro tvorbu aplikací určených ke zpracování zvukového signálu. Přechod na toto prostředí umožnil především editaci převodní charakteristiky v reálném čase a převedení efektu do formátu VST3. Kromě stručného představení použitých typů systémů, motivace pro převzorkování a popisu implementace obou prototypů jsou v práci obsaženy i grafické ukázky demonstrující jejich správnou funkčnost. Zvukové soubory související s těmito ukázkami jsou součástí elektronické přílohy.

KLÍČOVÁ SLOVA

Nelineární efekt, zkreslení, waveshaper, filtrace, převzorkování, Matlab, Audio Toolbox, VST3, JUCE, C++

ABSTRACT

The aim of this thesis is the implementation of a non-linear audio effect called waveshaper. This type of distortion effect contains the following building blocks: user defined transfer function, several types of filters and an oversampling processor with multiple stages of oversampling. The first prototype of this audio effect was implemented using Matlab and its Audio Toolbox extension. Due to certain limitations of this prototype, the whole audio effect was later completely rewritten in C++. This new implementation uses the JUCE framework which is mainly used for audio application development. The transition to this framework allowed real time editing of the transfer function and a VST3 build of the effect. In addition to a brief introduction of the used system types, motivation for oversampling and the description of the implementation for both prototypes, this thesis also includes graphical examples demonstrating their correct functionality. Audio files related to these examples are included in the electronic attachment.

KEYWORDS

Non-linear effect, distortion, waveshaper, filtering, oversampling, Matlab, Audio Toolbox, VST3, JUCE, C++

LEITGEB, David. *Realizace efektu Waveshaper v prostředí Matlab*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2022, 78 s. Bakalářská práce. Vedoucí práce: doc. Ing. Jiří Schimmel, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	David Leitgeb
VUT ID autora:	218925
Typ práce:	Bakalářská práce
Akademický rok:	2021/22
Téma závěrečné práce:	Realizace efektu Waveshaper v prostředí Matlab

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

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/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*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu doc. Ing. Jiřímu Schimmelovi, Ph.D. za odborné vedení, konzultace, trpělivost, cenné rady a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Mezi jeden z nejobvyklejších nežádoucích jevů, který lze zpozorovat při práci se signály je bezesporu jejich zkreslení. K němu zpravidla dochází při nevhodně nastavených podmínkách přenosu daného signálu. Ne vždy však musí být zkreslení nevýhodou, v případě hudebních signálů je naopak velmi často cíleně využíváno. S jeho pomocí lze velmi snadno upravit výslednou barvu vstupní nahrávky. Využití si tak najde například při práci s nahrávkami elektrické kytary či během úprav jednotlivých stop při míchání nějaké skladby.

Právě zkreslení hudebních signálů je tématem této bakalářské práce. Zabývá se konkrétně realizací nelineárního zvukového efektu typu *waveshaper*, který se skládá z následujících druhů zpracování zvukového signálu: převzorkování, zkreslení s uživatelem volenou převodní charakteristikou a úprava kmitočtové charakteristiky signálu pomocí filtrů různých typů. Celkově byly realizovány dvě implementace tohoto efektu, první z nich byla provedena v rámci semestrální práce s využitím výpočetního softwaru Matlab v kombinaci s jeho rozšířením Audio Toolbox. Druhá z nich pak byla provedena v programovacím jazyce C++ s využitím aplikačního frameworku JUCE, který je převážně zaměřený na vývoj softwaru určeného ke zpracování zvukového signálu.

První část této práce se krátce věnuje teorii, která stojí za funkčností zmíněných typů zpracování signálu. Konkrétně je zde rozebrána problematika systémů, které jsou klíčové pro zkreslení a kmitočtovou filtraci signálu, součástí teoretického úvodu je také vysvětlení principu převzorkování.

Druhá část této práce je rozdělena do dvou kapitol, každá z nich přibližuje implementaci efektu s využitím daného prostředí a jeho nástrojů. V případě kapitoly o realizaci v Matlabu jsou jednotlivé prvky waveshaperu popsány nejprve obecně, poté je pomocí úryvků zdrojového kódu ve formě výpisů přiblížen i způsob jejich implementace. Kapitola o realizaci v programovacím jazyce C++ se již výhradně zaměřuje pouze na popis způsobů implementace daných částí efektu (ta je z důvodu větší komplexnosti podrobněji vysvětlena primárně pomocí komentářů uvedených ve zdrojovém kódu, v tomto textu je uveden pouze její náznak).

Závěr práce se věnuje výsledkům, je zde nejprve krátce rozebráno uživatelské prostředí obou realizací efektu a způsob jejich testování. Hlavním obsahem této části jsou pak ukázky zpracování různých vstupních signálů pomocí vytvořených prototypů. Tyto ukázky jsou uvedeny v grafické (tvary použitých převodních charakteristik, kmitočtová spektra vstupních a výstupních signálů) i zvukové (nahrávky vstupních a zpracovaných signálů) podobě.

1 Teoretický úvod

1.1 Systémy

Obecně můžeme systém definovat jako soustavu několika prvků, které mají za cíl splnění určité funkce. Jednotlivé prvky této soustavy se vzájemně ovlivňují. Je několik způsobů, kterými lze systémy dělit na různé typy [1]. Jedním z těchto způsobů je rozdělení podle typu zpracovávaného signálu na:

- spojité systémy,
- diskrétní systémy.

Diskrétní systémy pak zahrnují i skupinu číslicových systémů, které již pracují se signály uloženými v určité číselné soustavě. Druhým důležitým způsobem dělení je rozdělení na:

- lineární systémy,
- nelineární systémy.

Tyto elementární informace o způsobu dělení systémů zde jsou uvedeny pro udržení alespoň částečně uceleného popisu této oblasti. Zbylá část této kapitoly se již výhradně věnuje pouze těm typům systémů, které jsou pro tuto bakalářskou práci klíčové.

1.1.1 Lineární systémy

Prvním z nich jsou lineární systémy, které se využívají pro realizaci kmitočtových filtrů. Jako lineární můžeme označit takový systém, který v sobě obsahuje pouze lineární prvky. Na rozdíl od nelineárních systému zde platí princip superpozice (viz (1.2)). Ten nám říká, že dostaneme stejnou výslednou odezvu tohoto systému na určitý vstupní signál nezávisle na tom, zda tento signál na systém působí celkově, nebo zda dojde k sečtení dílčích odezev tohoto systému. V případě lineárního diskrétního systému s dílčími výstupy $y_1[n]$ a $y_2[n]$, kde

$$y_1[n] = \tau\{x_1[n]\} \quad \text{a} \quad y_2[n] = \tau\{x_2[n]\} \tag{1.1}$$

tedy v důsledku principu superpozice platí [1]:

$$y[n] = \tau\{ax_1[n] + bx_2[n]\} = a\tau\{x_1[n]\} + b\tau\{x_2[n]\} = y_1[n] + y_2[n]. \tag{1.2}$$

Zde jsou a a b konstanty, $\tau\{\}$ je jednoznačná transformace vstupního diskrétního signálu $x[n]$ na výstupní diskrétní signál $y[n]$. Pokud není tato podmínka splněna, pak se nejedná o lineární systém.

Systémy je dále možné rozdělit podle časových změn parametrů prvku či změn zapojení na [1]:

- časově invariantní systémy,
- časově proměnné systémy.

Pro nás budou důležité diskrétní časově invariantní lineární systémy, které se pro kmitočtovou filtraci v případě digitálních efektů používají [2]. O tento typ systému se bude jednat i přes občasný zásah uživatele do parametrů použitého filtru, neboť perioda změny těchto parametrů je mnohonásobně větší než perioda zpracovávaného signálu.

Lineární časově invariantní diskrétní systém

Pro tento systém, také označovaný jako LTI (Linear Time Invariant), platí toto vyjádření vstupního signálu pomocí jednotkových impulzů [1]:

$$x[n] = \sum_{m=-\infty}^{\infty} x[m]\delta[n-m]. \tag{1.3}$$

Odezvou tohoto systému na jednotkový impulz je impulzní charakteristika $h[n]$:

$$h[n] = \tau\{\delta[n]\}. \tag{1.4}$$

Odezvu diskrétního LTI systému na vstupní signál pak tedy můžeme zapsat jako [1]:

$$\begin{aligned} y[n] = \tau\{x[n]\} &= \tau\{\sum_{m=-\infty}^{\infty} x[m]\delta[n-m]\} \\ &= \sum_{m=-\infty}^{\infty} x[m]\tau\{\delta[n-m]\} \\ &= \sum_{m=-\infty}^{\infty} x[m]h[n-m] \\ &= x[n] * h[n]. \end{aligned} \tag{1.5}$$

Výstup LTI systému tedy podle rovnice (1.5) odpovídá diskrétní konvoluci vstupního signálu s impulzní charakteristikou.

Diskrétní LTI systémy můžeme dále podle délky impulzní charakteristiky rozdělit na:

- diskrétní systémy s konečnou impulzní charakteristikou (FIR – Finite Impulse Response),
- diskrétní systémy s nekonečnou impulsní charakteristikou (IIR – Infinite Impulse Response).

Pro realizaci filtrů jsou v této práci využity systémy typu IIR s těmito vlastnostmi [3]:

- Impulsní charakteristika má nekonečný počet hodnot, platí:

$$y[n] = \sum_{i=0}^{\infty} h[i] \cdot x[n-i]. \tag{1.6}$$

- Mohou se objevit problémy se stabilitou systému, především při kvantování hodnot impulsní charakteristiky.
- Fázová kmitočtová charakteristika je vždy nelineární v celém rozsahu.
- Pro jejich implementaci je v porovnání se systémy typu FIR potřeba podstatně menší paměť. Je zde i velký rozdíl ve zpoždění, které vznikne při výpočtu výstupního signálu – v případě systému typu IIR je zpoždění podstatně kratší.

Samotný návrh jednotlivých filtrů je uveden v kapitolách 2.1.3 (Matlab) a 2.2.8 (JUCE). Ten je proveden pomocí návrhových vzorců pro výpočet koeficientů IIR filtru 2. řádu, které jsou následně dosazeny do této přenosové funkce [4]:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}, \tag{1.7}$$

kde b_0, b_1, b_2, a_1 a a_2 jsou vypočtené koeficienty daného filtru.

1.1.2 Nelineární systémy

Mezi nejpoužívanější efekty pro úpravu hudebního zvukového signálu se řadí zkreslení. Ne vždy je tento jev žádoucí, v případě zpracování obecného signálu se mu naopak snažíme vyhnout, nebo jeho účinek alespoň co nejvíce potlačit. Předmětem této práce je ovšem záměrné zkreslení zpracovávaného signálu a proto bude tomuto faktu přizpůsobeno i následující zpracování teorie týkající se této problematiky.

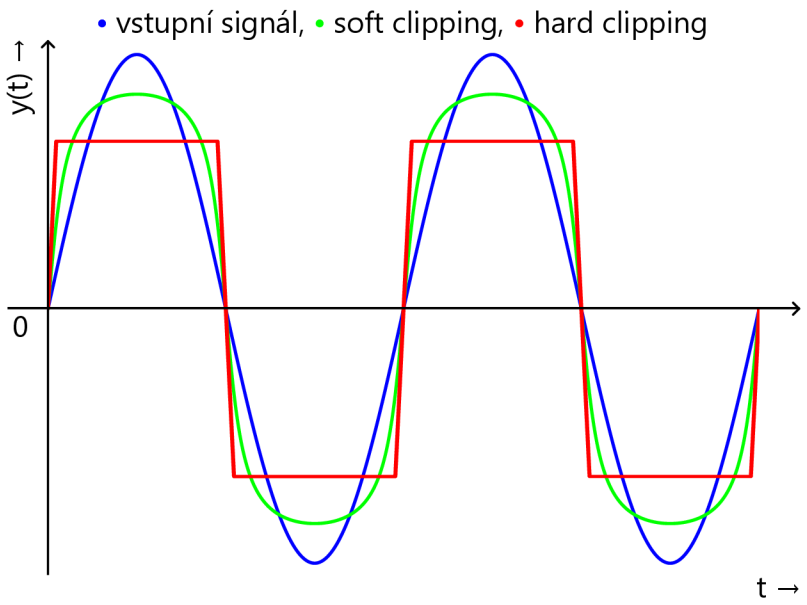
Za účelem zkreslení signálů se v analogové oblasti používají nelineární obvody, které lze obecně popsat tak, že obsahují alespoň jeden nelineární prvek. Jedním z důsledků přítomnosti takového prvku je fakt, že v nelineárních systémech nelze použít princip superpozice (neplatí tedy rovnice (1.2)). Na výstupu těchto systémů se pak v závislosti na vlastnostech vstupního signálu a použitých součástek objeví složky, které v původním signálu nebyly obsaženy – ty jsou jak harmonické, tak kombinační. Pro určení míry nelinearity systému lze v určitých případech použít výpočet celkového harmonického zkreslení (Total Harmonic Distortion, *THD*), který je vidět v rovnici (1.8) [4]:

$$\mathrm{THD} = \sqrt{\frac{A_2^2 + A_3^2 + \cdots + A_N^2}{A_1^2 + A_2^2 + \cdots + A_N^2}}, \tag{1.8}$$

kde A_1 až A_N jsou amplitudy jednotlivých harmonických složek. V čitateli jsou druhé mocniny amplitud vyšších harmonických složek, ve jmenovateli jsou pak druhé

mocniny amplitud všech harmonických složek signálu. Výsledek je udáván v procentech nebo decibelech, pro věrnou reprodukci vstupního signálu se snažíme docílit co nejmenší hodnoty THD. Vznikem vyšších harmonických složek dojde k obohacení spektra vstupního signálu, které na posluchače může v případě signálu hudebního charakteru při vhodném použití efektu působit příjemným dojmem (důvodem jsou různé hudební intervaly mezi jednotlivými harmonickými složkami, např. druhá harmonická složka tvoří společně s první oktávu). Za tímto účelem vzniká mnoho různých analogových zařízení i softwarů (např. efektové pedály k elektrickým kytarám, různé VST pluginy). Jako možné hledisko pro rozdělení těchto efektů pak může být např. barva výstupního signálu. Vzhledem k tomu, že se tato bakalářská práce týká realizace virtuálního efektu, uvedené informace z oblasti nelineárních systémů se budou v rámci celé práce týkat převážně zkreslení v digitální oblasti, jehož realizací se zabývají kapitoly 2.1 (Matlab) a 2.2 (JUCE).

Navržený efekt typu waveshaper má za úkol změnu tvaru signálu. K té dojde dosazením vstupního vzorku zpracovávaného signálu do požadované, v tomto případě uživatelem volené převodní charakteristiky. Ta může mít i poměrně komplexní tvar, což bude mít za důsledek velmi nepřirozenou barvu zpracovaného záznamu. Pokud však bude tato přenosová funkce obsahovat nějakou lineární část a okamžité hodnoty vstupního signálu budou odpovídat pouze této oblasti, ke vzniku žádných dalších složek nedojde. Z vlastností použité funkce můžeme alespoň přibližně odhadnout výslednou podobu spektra výstupního signálu, zejména zastoupení jednotlivých harmonických složek. Platí, že při použití obecné funkce získáme na výstupu všechny harmonické složky. Pro liché funkce pak můžeme říct, že po zpracování bude mít výstupní signál pouze liché harmonické složky a v případě sudé bude mít naopak pouze sudé harmonické složky [2]. Demonstrace těchto faktů je obsažena v rámci kapitol 3.1.3 a 3.2.4, kde jsou pro jejich ověření využity přímo navržené pluginy. Informace o samotném vzniku určitých alikvót však není plnohodnotným popisem signálu, důležitým faktorem ovlivňujícím výslednou barvu zvuku jsou také vzájemné poměry mezi jednotlivými harmonickými složkami. Následkem pomalých přechodů v použité funkci je nízká amplituda harmonických složek nacházejících se na vysokých kmitočtech – využití pro jemnější typy zkreslení, používá se označení *soft clipping*. Pro ostré zlomy naopak platí, že harmonické složky v této části spektra budou mít velkou amplitudu [2] – využití pro silnější zkreslení, používá se označení *hard clipping*. Oba způsoby oříznutí signálu jsou vidět na obr. 1.1 Tento obrázek má pouze ilustrativní charakter, nejedná se o přesnou analýzu.



Obr. 1.1: Rozdíl mezi soft a hard clipping.

1.2 Převzorkování

Proces převzorkování bývá velmi často součástí efektů pro zkreslení. Má to své opodstatnění, které souvisí s principem, na kterém je založeno digitální zpracování zvuku. Pro získání signálu v podobě číselných vzorků je nutné analogový signál nejprve navzorkovat a nakvantovat za dodržení určitých pravidel. Pro zachování možnosti rekonstrukce původního průběhu je nutné dodržet tzv. **Nyquistův teorém** (někdy nazývaný Shannon–Kotělnikovův), který zní [1]:

$$f_{\mathrm{VZ}} > 2f_{\max}, \tag{1.9}$$

kde f_{VZ} je vzorkovací kmitočet a $f_{\max}$ je nejvyšší kmitočet obsažený ve vzorkovaném signálu. Jeho dodržením zabráníme překrytí sousedních spekter (tzv. **aliasingu**) [1].

Důvodem pro použití převzorkování je fakt, že při zkreslení signálu dochází k vytvoření nekonečného počtu vyšších harmonických složek. Při analogové realizaci zkreslení nejsou tyto složky tak rizikové, neboť kvůli vlastnostem použitých součástek dojde k jejich útlumu. Pro digitální implementace je to však problém, protože při zkreslení dojde k antialiasingu a ve výstupním signálu se objeví spektrální složky, které už vzhledem ke vstupnímu signálu nejsou v harmonickém poměru [5]. Abychom tomuto jevu zamezili anebo jej alespoň zmírnili, provedeme ještě před samotným zkreslením nadvzorkování. Tím dojde ke zvýšení periody, se kterou se spektra opakují. Po zkreslení signálu provedeme podvzorkování na původní vzorkovací frekvenci.

Nadvzorkování provádíme pomocí tzv. interpolace. Ta spočívá ve vložení $L-1$ nových vzorků mezi již existující, přičemž hodnota vložených vzorků je vypočítána

pomocí interpolace. Rozlišujeme různé stupně nadvzorkování, které jsou určeny celočíselným činitelem nadvzorkování L. Při nadvzorkování dojde k vytvoření obrazů spektra, což vyřešíme pomocí antialiasingového filtru s mezním kmitočtem $f_{\mathrm{VZ}}/2L$ [2]. Zároveň dojde ke zvýšení vzorkovacího kmitočtu.

Podvzorkování provádíme pomocí tzv. decimace. Ta je naopak založena na výběru každého Mtého vzorku pro celočíselný činitel podvzorkování M. Pro zabránění antialiasingu je nutné zajistit splnění Nyquistova teorému také pro výsledný vzorkovací kmitočet. Toho docílíme využitím filtru typu dolní propusti s mezním kmitočtem $f_{\mathrm{VZ}}/2M$ [2]. Zároveň dojde ke snížení vzorkovacího kmitočtu.

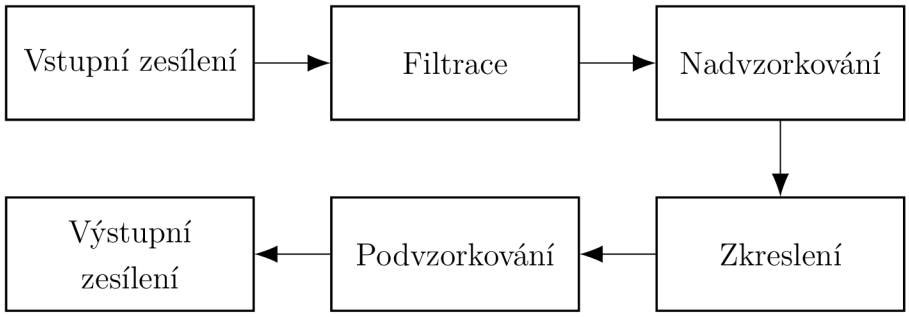
2 Implementace zvukového efektu

Tato kapitola popisuje způsoby a nástroje využité k realizaci zadaného zvukového efektu v podobě digitálního zásuvného modulu, obecně také označovaného jako tzv. plugin (část kódu obsahující algoritmus zpracování signálu, která pro svůj chod potřebuje hostitelskou aplikaci [2]). Je rozdělena do dvou hlavních částí. První z nich se zabývá implementací v prostředí Matlab, která byla provedena v rámci semestrální práce. Je rozdělena do dílčích sekcí odpovídajících jednotlivým prvkům waveshaperu. Na začátku každé této sekce je nejprve obecně zmíněn princip, na kterém daný prvek funguje, poté je zde přiblížen způsob jeho implementace s ukázkami ze zdrojového kódu. Druhá hlavní část této kapitoly popisuje realizaci v jazyce C++ s využitím frameworku JUCE. V ní jsou už popsány pouze způsoby implementace v tomto prostředí.

2.1 Implementace v prostředí Matlab

Pro realizaci zadaného zvukového efektu byl nejprve zvolen software Matlab od společnosti MathWorks. Pro práci byla rovněž použita knihovna funkcí *Audio Toolbox*, která výrazně usnadnila implementaci prvků pro zpracování zvukového signálu a rovněž umožnila provádět tento proces v reálném čase. Další vlastností tohoto toolboxu je možnost převodu vzniklého pluginu do C++ frameworku JUCE. Touto problematikou se zabývá kapitola 2.1.5. Pro vytvoření a testování této práce byl použit Matlab verze R2020b Update 4 společně s Audio Toolboxem ve verzi 2.3.

Realizovaný plugin se skládá z několika již dříve zmíněných prvků. Přesná vizualizace toku signálu mezi nimi je vidět na obr. 2.1. Jednotlivé bloky a jejich algoritmy jsou jak již bylo zmíněno nejprve rozebrány obecně, až poté je na konci každé kapitoly přiblížen způsob jejich implementace v Matlabu.



Obr. 2.1: Blokové schéma pluginu.

2.1.1 Audio Toolbox

Rozšíření Audio Toolbox je určeno převážně pro zpracování zvukových signálů, analýzu řeči a měření akustických veličin. Hlavní výhodou tohoto rozšíření, pro kterou bylo zvoleno pro realizaci původního prototypu efektu, je rychlá implementace algoritmů pro zpracování signálů. O základní funkčnost (např. čtení bloků zvukových vzorků, jejich zápis, či navázání ovládacích prvků pluginu na správné parametry) se stará samotné rozšíření. To však nemusí být vždy výhodou, viz kapitola 2.1.5. Stejně tak není potřeba manuálně naprogramovat vizuální stránku navrhovaného pluginu, stačí provést výběr z několika dostupných ovládacích prvků a následné nastavení jejich parametrů. Díky těmto faktorům se tak může uživatel tohoto rozšíření zaměřit hlavně na samotné algoritmy.

Pro lepší orientaci čtenáře ve vzniklém zdrojovém kódu je zde uvedena základní struktura zápisu audio pluginu, která se používá v rozšíření Audio Toolbox.

Výpis 2.1: Základní struktura zdrojového kódu audio pluginu.

```
classdef Waveshaper < audioPlugin
    properties
        % Zde jsou definovány proměnné použité dále.
    end
    properties (Constant)
        PluginInterface = audioPluginInterface( ...
        % Zde jsou tyto proměnné namapovány
        % na jednotlivé ovládací prvky.
        );
    end
    methods
        function out = process(plugin,in)
        % Toto je hlavní funkce se samotnými algoritmy.
        end
    end
end
```

Tato struktura zůstává neměnná, uživatel pouze přidává potřebný obsah ve formě proměnných, ovládacích prvků a algoritmů. Pro úplnost jsou zde ještě zmíněny ovládací prvky, které jsou v rámci práce použity. Jejich definice je provedena vložením `audioPluginParameter()` do argumentu `audioPluginInterface()`. V tomto místě jsou definovány všechny ovládací prvky mezi sebou oddělené čárkou. Jejich nastavení je provedeno pomocí několika parametrů uvedených v argumentu `audioPluginParameter()`. Mezi tyto parametry patří především typ prvku, jeho

název, umístění v mřížce uživatelského prostředí, rozsah a použitá veličina (např. dB). V této práci jsou použity jejich následující typy:

- `rotaryknob` – otočný potenciometr,
- `dropdown` – rozbalovací menu,
- `vrocker` – přepínač.

Výčet všech dostupných typů uživatelských prvků společně s jejich parametry je možné najít v [6]. Za jednu z nevýhod Audio Toolboxu by mohla být označena omezená možnost zásahu do vzhledu pluginu, který je zde poměrně dost základní. Pokročilejší úpravy jsou teoreticky možné až po převedení do JUCE, pro účely semestrální práce byla ovšem vizuální stránka dostačující.

2.1.2 Zkreslení zvukového signálu

Princip waveshaperu je založen na průchodu signálu na vstupu nějakou nelineární funkcí, což způsobí vznik vyšších harmonických složek na výstupu. Jak již bylo zmíněno, tvary této funkce mohou být různé a od nich se budou odvíjet i vlastnosti výstupního signálu. Ve vzniklém pluginu je editace přenosové funkce provedena formou přidávání a odebírání jednotlivých bodů. Jejich maximální možný počet je u implementace v Matlabu omezen na 20 bodů. Uživatel má na výběr ze dvou typů proložení – lineární a proložení polynomem. Obě implementace jsou více přiblíženy v následujících odstavcích.

Pro editaci přenosové funkce slouží funkce `draw.m`. V argumentu přijímá pouze objekt pluginu, což umožňuje přístup k parametrům uloženým v hlavní třídě. Výstupem této funkce jsou zvolené body uložené v proměnné xy, jejich počet v proměnné n a také vypočtené strmosti k a hodnoty q pro úseky mezi jednotlivými body. Tyto dvě proměnné jsou dále použity pro výpočet zkreslení s lineárním proložením bodů. Jejich výpočet je v této funkci umístěn z toho důvodu, aby k němu došlo pouze při změně přenosové funkce. K jejímu návrhu je použit příkaz pro její vykreslení `plot` společně s funkcí `ginput`, která umožňuje detekci pozice, na které došlo ke stlačení tlačítka myši. Hlavní nevýhodou využití kombinace těchto dvou funkcí je problém s převodem pluginu na technologii VST, viz kapitola 2.1.5.

Pro zkreslení zpracovávaného signálu slouží funkce `distort.m`, která při jejím zavolání v argumentu přijímá objekt pluginu a vstupní signál. V závislosti na zvoleném typu proložení bodů je na jejím výstupu signál zkreslený příslušným způsobem. Její součástí je také implementace symetrického zkreslení, které je realizováno převodem záporných vzorků na kladné, aplikací požadované přenosové funkce a následným převodem těchto vzorků zpět.

Lineární proložení – přenosová funkce s tímto typem proložení bude obsahovat ostré zlomy, které jak již bylo zmíněno způsobí velké množství energie na vysokých kmitočtech. Pro definici jednotlivých částí převodní funkce je použito rovnice přímky ve směrnicovém tvaru $y = kx + q$. Zkreslení využívající tohoto proložení je v pluginu implementováno manuálně za pomocí výpočtu strmostí jednotlivých úseků podle následujícího postupu:

1. nejprve je vypočtena strmost k pro dva po sobě jdoucí body:
$$k = \frac{y(x_2) - y(x_1)}{x_2 - x_1}, \tag{2.1}$$
2. poté je vypočteno q:
$$q = y(i) - kx(i), \tag{2.2}$$
3. posledním krokem je dosazení do směrnicového tvaru přímky:
$$y = kx + q. \tag{2.3}$$

První dva kroky jsou provedeny ještě uvnitř funkce `draw.m` – v cyklu `for` dojde k výpočtu hodnot k a q pro dvojice po sobě jdoucích bodů. Samotné dosazení vzorků zpracovávaného signálu do rovnice (2.3) je součástí funkce `distort.m`. Implementace obsažená v této funkci je vidět na výpisu 2.2.

Výpis 2.2: Implementace lineárního zkreslení.

```
for sampleIndex = 1:length(audioDistorted)
  for pointIndex = 1:plugin.n - 1
    if audioDistorted(sampleIndex,:) >= plugin.Points(
       pointIndex,1) & audioDistorted(sampleIndex,:) <=
       plugin.Points(pointIndex+1,1)
        audioDistorted(sampleIndex,:) = plugin.kq(
           pointIndex,1) * audioDistorted(sampleIndex,:)
           + plugin.kq(pointIndex,2);
    end
  end
end
```

V něm vidíme dva cykly `for` – první prochází všemi vzorky v aktuálně zpracovávaném bloku a druhý pak slouží ke zjištění intervalu, do kterého aktuální vzorek patří. Po zjištění tohoto úseku dojde k dosazení vstupní hodnoty signálu do rovnice (2.3) společně s hodnotami k a q příslušnými k tomuto intervalu, které byly vypočteny již při ukončení režimu editace přenosové funkce.

Proložení polynomem – tento typ proložení způsobí pozvolnější přechody, které se ve spektru výstupního signálu projeví nižším množstvím energie na vysokých kmitočtech, než tomu bylo u lineárního proložení. Pro aproximaci pomocí

polynomu také platí, že na výstupu vzniknou pouze harmonické složky, jejichž pořadové číslo odpovídá nejvýše řádu použitého polynomu [2]. Pro implementaci byly použity funkce Matlabu `polyfit` a `polyval`. Výstupem funkce `polyfit` jsou koeficienty polynomu, které jsou poté společně se vstupním signálem použity ve funkci `polyval`, která provede proložení vstupního signálu daným polynomem. Funkce polyfit používá pro proložení bodů metodu nejmenších čtverců [7]. Implementace těchto funkcí je vidět na výpisu 2.3. V prvním sloupci matice *Points* jsou uloženy x-ové souřadnice a ve druhém pak y-ové.

Výpis 2.3: Implementace zkreslení s proložením bodů polynomem.

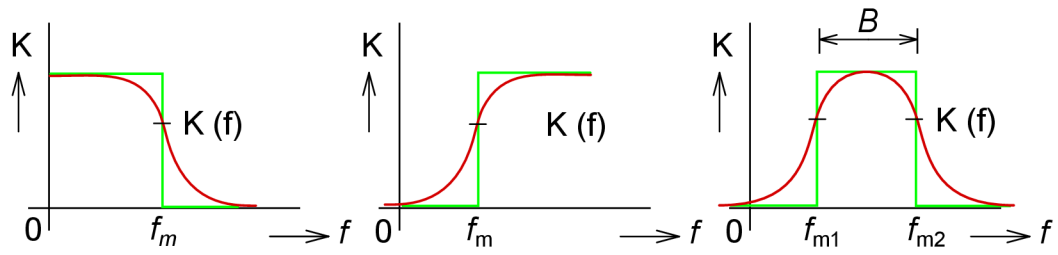
```
degree = plugin.n - 1;
pCoeff = polyfit(plugin.Points(:,1), plugin.Points(:,2),
   degree);
audioDistorted = polyval(pCoeff, audioDistorted);
```

2.1.3 Filtrace zvukového signálu

Další nedílnou součástí efektu typu waveshaper je kmitočtová filtrace signálu. Ta je v případě realizovaného pluginu umístěna ještě před samotné zkreslení za účelem zmírnění amplitud vyšších harmonických složek s vysokým pořadovým číslem, které na posluchače mohou působit příliš ostře, občas až nepříjemně. Jak již bylo zmíněno v kapitole 1.1.1, jedná se o lineární systémy, které upravují kmitočtovou charakteristiku signálu v závislosti na jejich typu a parametrech. Mezi základní typy filtrů patří:

- **horní propust** (high-pass) – propouští pouze nad mezním kmitočtem f_{m},
- **dolní propust** (low-pass) – propouští pouze pod mezním kmitočtem f_{m},
- **pásmová propust** (bandpass) – propouští pouze mezi dolním f_{m1} a horním f_{m2} mezním kmitočtem,
- **pásmová zádrž** (band-reject) – potlačuje oblast mezi dolním f_{m1} a horním f_{m2} mezním kmitočtem,
- **shelving** – podobný s horní a dolní propustí, u tohoto typu ovšem nedochází k úplnému potlačení spektra, nýbrž k úpravě zisku nad nebo pod mezním kmitočtem f_{m},
- **peak** – úprava zisku v okolí požadovaného kmitočtu,
- **fázovací článek** (allpass) – speciální případ, u kterého nedojde k úpravě kmitočtové charakteristiky, ale změní se fáze filtrovaného signálu.

Ilustrace modulových charakteristik prvních tří uvedených filtrů můžeme vidět na obr. 2.2 [8].



(a) Dolní propust (b) Horní propust (c) Pásmová propust

Obr. 2.2: Skutečné (červené) a ideální (zelené) modulové charakteristiky filtrů.

Různých typů filtrů je samozřejmě ještě více, stejně jako dalších hledisek, podle kterých lze filtry rozdělit. Jedním z dalších způsobů dělení je také podle jejich realizace na [8]:

- analogové filtry, které lze rozdělit na:
  - pasivní filtry,
  - aktivní filtry,
- digitální filtry, které lze rozdělit na:
  - filtry s konečnou impulsní odezvou – Finite Impulse Response (FIR),
  - filtry s nekonečnou impulsní odezvou – Infinite Impulse Response (IIR).

Pro naše účely je důležitá skupina digitálních filtrů s nekonečnou impulsní charakteristikou, které jsou pro zvukové efekty v digitální oblasti nejpoužívanější [2]. V navrženém pluginu je na výběr ze tří různých typů filtrů (tzv. parametrických): dolní propust, horní propust a pásmová propust. Jejich realizace je provedena pomocí IIR filtru 2. řádu podle [4]. Návrhové vzorce pro výpočet koeficientů tohoto typu filtru jsou uvedeny v tab. 2.1,

Tab. 2.1: Návrhové vzorce pro výpočet koeficientů IIR filtru 2. řádu [4].

	b_0	b_1	b_2	a_1	a_2
Lowpass	$\frac{K^2Q}{K^2Q+K+Q}$	$\frac{2K^2Q}{K^2Q+K+Q}$	$\frac{K^2Q}{K^2Q+K+Q}$	$\frac{2Q\cdot(K^2-1)}{K^2Q+K+Q}$	$\frac{K^2Q-K+Q}{K^2Q+K+Q}$
Highpass	$\frac{Q}{K^2Q+K+Q}$	$-\frac{2Q}{K^2Q+K+Q}$	$\frac{Q}{K^2Q+K+Q}$	$\frac{2Q\cdot(K^2-1)}{K^2Q+K+Q}$	$\frac{K^2Q-K+Q}{K^2Q+K+Q}$
Bandpass	$\frac{K}{K^2Q+K+Q}$	0	$-\frac{K}{K^2Q+K+Q}$	$\frac{2Q\cdot(K^2-1)}{K^2Q+K+Q}$	$\frac{K^2Q-K+Q}{K^2Q+K+Q}$

kde $K = \tan(\pi f_C/f_S)$, Q je činitel jakosti filtru, f_C je mezní/střední kmitočet filtru a f_S je vzorkovací kmitočet.

Tyto návrhové vzorce jsou implementovány v rámci funkce `filters.m`, která jako vstupní parametry přijímá objekt pluginu a vstupní signál. Hodnoty mezního kmitočtu f_C a činitele jakosti Q tato funkce získá z otočných potenciometrů v uživatelském prostředí. Vzorkovací kmitočet je získán pomocí funkce `getSampleRate()`, která ji vyčte ze samotného pluginu. Koeficienty tedy budou nabývat správných hodnot pro zpracovávané signály s různými vzorkovacími kmitočty. Koeficienty a_i a b_i, vypočtené podle tab. 2.1, jsou poté společně se vzorky vstupního signálu dosazeny do Matlab funkce `filter` (viz výpis 2.4), která provede samotnou filtraci. Z té zároveň získáme kromě filtrovaného signálu také vnitřní stav filtru, který je následně dosazen zpět do této funkce při přechodu na další blok zpracovávaného signálu. Důvodem jsou artefakty, které by se při neuchování vnitřních stavů filtru objevily na přechodech mezi navazujícími bloky signálu. Výstupem funkce `filters.m` je tedy již filtrovaný signál, který odpovídá výstupu filtru zvoleného přepínačem. V rámci semestrální práce bylo možné použít pouze jeden typ filtru současně. Další typy vzniklé především jejich spojením byly i s možností jejich kombinace přidány v rámci bakalářské práce.

Výpis 2.4: Implementace filtrů s využitím funkce `filter`.

```
function out = filters(plugin,in)
    fs = getSampleRate(plugin); % Vzorkovací kmitočet
    fc = plugin.Frequency; % Mezní kmitočet
    Q = plugin.Q; % Činitel
    % zde jsou umístěny návrhové vzorce z tab. 2.1
    [out_hp, plugin.hp_state] = filter(b_hp, a_hp, in
       , plugin.hp_state)
    % na tomto místě je switch pro výběr filtru
end
```

Výpis 2.4 je zde zjednodušen za účelem udržení přehlednosti textu, jelikož obsahem funkce `filters.m` jsou především návrhové vzorce pro výpočet koeficientů filtrů, které zde již byly zmíněny v rámci tab. 2.1. Dosazení do Matlab funkce `filter` je zde také uvedeno pouze jednou, ve zdrojovém kódu je její volání provedeno pro každý filtr zvlášť. Výběr výstupu navržené funkce je pak jak již bylo zmíněno provedeno pomocí jednoduchého přepínače.

2.1.4 Převzorkování zvukového signálu

Za účelem převzorkování signálů se používají principy popsané v kapitole 1.2. Pro realizaci **nadvzorkování** v Matlabu byla použita funkce `interp(x,L)`, která jako argumenty přijímá vstupní signál x určený k nadvzorkování a činitel nadvzorkování

L. Tato funkce funguje na již zmíněném principu vložení vzorků s nulovou hodnotou mezi původní vzorky a následném použití filtrace [9]. Pro účel **podvzorkování** byla použita funkce `decimate(x,M)`, která jako argumenty přijímá vstupní signál x a činitel podvzorkování M. Tato funkce v základním nastavení používá filtr typu dolní propusti 8. řádu s aproximací dle Čebyševa [10].

Implementace nadvzorkování a podvzorkování je ve vypracovaném pluginu provedena v rámci funkce `oversample.m`, která je vidět na výpisu 2.5. Ta přijímá tři vstupní parametry v uvedeném pořadí – vstupní signál, směr převzorkování a celočíselný činitel převzorkování. Funkce `interp` a `decimate` jsou tak volány v závislosti na vyhodnocení přepínače, který se řídí směrem převzorkování (možnosti `'Upsample'` a `'Downsample'`). Obě funkce používají základní nastavení zmíněné v předešlé části této kapitoly. Výstupem `oversample.m` je pak již převzorkovaný signál.

Výpis 2.5: Implementace převzorkování.

```
function out = oversample(in, direction, factor)
    switch direction
        case ('Upsample')
            out = [interp(in(:,1), factor), interp(in
               (:,2), factor)];
        case ('Downsample')
            out = [decimate(in(:,1),factor), decimate(in
               (:,2),factor)];
    end
end
```

2.1.5 Převod na VST

Poslední krok, který nám brání v použití navrženého pluginu v námi zvolené hostitelské aplikaci, je převod na technologii VST, přesněji nejprve převod zdrojového kódu Matlabu do jazyka C++. Pro tento účel se používá rozšíření *Matlab Coder*. Před vygenerováním převedeného zdrojového kódu je potřeba provést kontrolu pomocí příkazu `validateAudioPlugin`. Dojde tak k otestování přítomnosti chyb, které by mohly mít za následek chybu běhu kódu uživatelem používané hostitelské aplikace. Poté už stačí pomocí příkazu `generateAudioPlugin` zahájit převod na VST. Určité chyby se mohou projevit až při použití tohoto příkazu. Za volání obou příkazů je nutné napsat název hlavní třídy převáděného pluginu. Obě funkce mají ještě možnost dodatečného nastavení, které lze dohledat v jejich dokumentaci. Pro úplnost je zde pouze zmíněn způsob, jakým by byl zahájen převod vytvořeného pluginu do frameworku JUCE: `generateAudioPlugin -juceproject Waveshaper`.

Jak již bylo zmíněno v předchozím odstavci, pro převod do jazyka C++ se používá rozšíření Matlab Coder, který má určitá omezení. Nejvýraznější omezení v kontextu prvního prototypu efektu se týkalo především funkcí Matlabu, které jsou a nejsou podporované při převodu do jiných formátů. Mezi problémové funkce patří `plot` a `ginput`, které jsou v realizaci pluginu použity pro editaci převodní charakteristiky. Použity byly z toho důvodu, že samotný Audio Toolbox pro ně nemá žádnou adekvátní náhradu. Možným řešením by bylo zadávání převodové funkce formou textu, tabulky, nebo poté kompletní přeprogramování editačního módu v jazyce C++. Jako nejsmysluplnější možnost však byl celkový přechod do jazyka C++, který je blíže popsán v následující části.

2.2 Implementace v programovacím jazyce C++

Za účelem splnění cílů stanovených po dokončení semestrální práce byl celý plugin od základu přepracován do programovacího jazyka C++. K tomu byl využit framework JUCE, který je krátce představen v následující kapitole. Hlavním důvodem byl především jednodušší převod do formátu VST, který je možné použít v kombinaci s jakoukoliv uživatelem zvolenou hostitelskou aplikací, která tento formát podporuje. Tento cíl nebylo možné s použitím Matlabu v kombinaci s Audio Toolboxem splnit, viz kapitola 2.1.5. Zdrojový kód prototypu zpracovaného v rámci semestrální práce již proto nebyl dále upravován. Díky přechodu do C++ je nyní také možné provádět změny převodní charakteristiky v reálném čase. Dalším přínosem byly nesrovnatelně větší možnosti úpravy uživatelského prostředí, což umožnilo alespoň částečné přiblížení se ke kvalitě komerčního softwaru. Pro vytvoření a testování této verze pluginu byl použit JUCE v6.1.6 a Microsoft Visual Studio Community 2019. V rámci této kapitoly jsou důležité části zdrojového kódu z důvodu větší komplexnosti už popsány pouze slovně.

2.2.1 JUCE (Jules' Utility Class Extensions)

JUCE je aplikační framework používaný k vývoji softwaru určeného pro zpracování zvuku. Je napsán v programovacím jazyce C++. Jeho hlavní výhodou je umožnění vývoje pro více platforem najednou a také přítomnost rozsáhlých modulů, které nesmírně ulehčují práci při vytváření jak programové, tak vizuální části vyvíjeného softwaru. Autorem tohoto frameworku je Julian Storer, který dříve pracoval na vývoji DAW s názvem Tracktion (dnes již Waveform). V roce 2004 z částí zdrojového kódu tohoto DAW vytvořil a vydal právě JUCE, který je v dnešní době velmi často používán [11]. Používají jej pro vývoj jak jednotlivci (v nabídce je verze

zdarma pro studenty a soukromé použití [12]), tak i velké společnosti (např. Korg, M-Audio, Cycling 74', a další) [13].

2.2.2 Projucer

Projucer je nástroj, který se používá pro organizaci projektů vyvíjených s použitím frameworku JUCE. Po vytvoření nového projektu vygeneruje šablonu zdrojového kódu, který je dále upravován samotným vývojářem. Tato šablona má určitou základní strukturu, která je stručně popsána v následující kapitole. Dále je možné v Projuceru měnit různá nastavení daného projektu – např. základní informace o vyvíjeném softwaru, vytvářet nové třídy, přidávat další moduly obsažené v JUCE, přidávat knihovny funkcí pro jazyk C++, změnit použité vývojové prostředí (např. Visual Studio (Microsoft), Xcode (Apple), a další), nebo zvolit formáty, do kterých bude proveden export pluginu (např. VST3, AU, AAX). Po otevření projektu vytvořeného v rámci této bakalářské práce (Projucer používá formát *.jucer*) je nutné zkontrolovat, že je u použitého vývojového prostředí (sekce *Exporters*) u obou módů (*debug* a *release*) v kolonce *Header Search Paths* uvedena cesta k externí knihovně *Eigen*, která byla použita pro práci s maticemi a je blíže představena v kapitole o implementaci zkreslení. Složka se soubory této knihovny je obsažena v adresáři se zdrojovým kódem, který je součástí přílohy této bakalářské práce.

2.2.3 Základní struktura pluginů vytvořených pomocí JUCE

Princip plugin modulu je obecně založen na zpracování určitého množství zvukových vzorků (tzv. *buffer*), které mu dodává použitá hostitelská aplikace. Spolu s tímto blokem vstupního signálu je pluginu také dodána informace o délce této vyrovnávací paměti, hodnota vzorkovacího kmitočtu, počet vstupních a výstupních kanálů a samozřejmě i údaj o místě v paměti, kde se blok vzorků určených ke zpracování nachází. Zpracovaný signál by teoreticky mohl být ukládán na jiné místo v paměti, v případě frameworku JUCE je však přepisována původní vyrovnávací paměť [5]. Výhodou tohoto principu je, že plugin se stará pouze o samotné zpracování zvukových vzorků dodaných použitou hostitelskou aplikací. Pro tento účel slouží tzv. *callback funkce*, která je opakovaně volána při požadavku na zpracování následujícího bloku zvukového signálu. Jeho zpracování se ovšem bude ve většině případů lišit v závislosti na uživatelem nastavených parametrech. Tento fakt se odráží i ve struktuře šablony, kterou při vytvoření nového projektu vygeneruje Projucer.

V připraveném zdrojovém kódu jsou dvě hlavní třídy – *PluginProcessor* a *PluginEditor*. Jak jejich názvy napovídají, každá z nich se stará o odlišnou část činnosti pluginu, *PluginProcessor* má na starost zpracování zvukových dat, zatímco *PluginEditor* se stará o grafické prostředí, kterým je uživatel schopný měnit nastavení

daného efektu. Každá z těchto tříd obsahuje svůj konstruktor, dekonstruktor a několik metod určených pro různé účely (viz tab. 2.2). Ty nejdůležitější (označené tučným písmem) zde budou stručně vysvětleny. Zbylé metody napsané pro účel této práce jsou zmíněny v rámci kapitol popisujících implementaci daných prvků.

Tab. 2.2: Základní struktura projektu vygenerovaného pomocí Projuceru.

`PluginProcessor`	`PluginEditor`
prepareToPlay()	**paint()**
releaseResources()	**resized()**
isBusesLayoutSupported()	
processBlock()	
createEditor()	
hasEditor()	
getName()	
acceptsMidi()	
producesMidi()	
gettailLengthSeconds()	
getNumPrograms()	
getCurrentProgram()	
getProgramName()	
changeProgramName()	
getStateInformation()	
setStateInformation()	

PluginEditor:

- **`paint()`** – má na starost vykreslení uživatelského prostředí,
- **`resized()`** – slouží k umístění ovládacích prvků editoru.

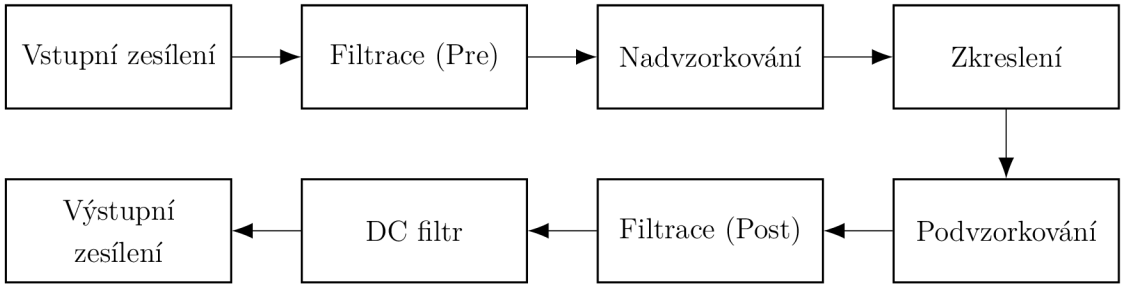
PluginProcessor:

- **`prepareToPlay()`** – zavolána před spuštěním zpracování zvukových dat, je v ní možné provést inicializaci parametrů,
- **`releaseResources()`** – zavolána po tom, co hostitelská aplikace ukončí zpracovávání dat, slouží např. k uvolnění paměti,
- **`processBlock()`** – opakovaně volána při obdržení nové vyrovnávací paměti, obsahuje algoritmy pro zpracování zvukových dat,
- **`createEditor()`** – vytvoří editor (okno s uživatelským prostředím pluginu nemusí být otevřené od momentu vložení efektu až do jeho odstranění, editor tak na rozdíl od procesoru nemusí vždy existovat – podle toho je potřeba přizpůsobit i uživatelem prováděnou manipulaci s parametry, viz. kapitola 2.2.6),

- `hasEditor()` – určuje, jestli má vyvíjený plugin nějaký editor vůbec obsahovat,
- `getStateInformation()` – ukládání parametrů do paměti,
- `setStateInformation()` – načtení parametrů z paměti.

2.2.4 Aktualizované blokové schéma waveshaperu

Po dokončení semestrální práce bylo rozhodnuto o určitých změnách, které se projevily i na základním blokovém schématu waveshaperu. Jak je vidět na obr. 2.3, v rámci bakalářské práce byly přidány další možnosti filtrace. Filtraci je teď možné provést jak ještě před samotným zkreslením, tak nově i až po něm. Dále byl také přidán filtr stejnosměrné složky, který je umístěn před výstupním zesílením pluginu. Nové možnosti filtrace jsou více přiblíženy v kapitole o jejich implementaci.



Obr. 2.3: Aktualizované blokové schéma pluginu.

2.2.5 Třída ShaperWindow

Ještě před popisem realizace waveshaperu je nutné zmínit další důležitou třídu, která vznikla za účelem udržení určité přehlednosti, což na druhé straně při ohlédnutí zpět přineslo i určité komplikace. Jedná se o třídu s názvem `ShaperWindow`, která se týká výhradně pole editoru určeného k návrhu převodní charakteristiky. Tato třída dědí z JUCE třídy `Component` zaměřené na uživatelské prostředí. Vytvořená třída obsahuje předefinování (tzv. `override`) několika zděděných metod. Vzhledem k tomu, že byla tato třída implementována za účelem realizace uživatelského prostředí pro zkreslení, tyto metody budou blíže specifikovány až v kapitole, která jej popisuje (viz kapitola 2.2.7).

2.2.6 Synchronizace parametrů procesoru s editorem

Tato kapitola je rozdělena na dvě části, první se zabývá obecnou synchronizací parametrů s prvky uživatelského prostředí (např. tlačítka, otočné potenciometry,

přepínače, a další), ve druhé části je pak přiblížen systém, na kterém funguje editace převodní charakteristiky.

Za účelem synchronizace parametrů procesoru s ovládacími prvky byla použita třída **`AudioProcessorValueTreeState`**, která je pro toto použití určena. Její instance je v rámci třídy `PluginProcessor` propojena s hodnotami jednotlivých parametrů procesoru, které jsou dále použity pro zpracování zvukových dat. Uvnitř třídy `PluginEditor` jsou tyto parametry propojeny s ovládacími prvky pluginu. Podrobnější popis funkčnosti a samotného nastavení této třídy je možné nalézt v její dokumentaci, viz [14].

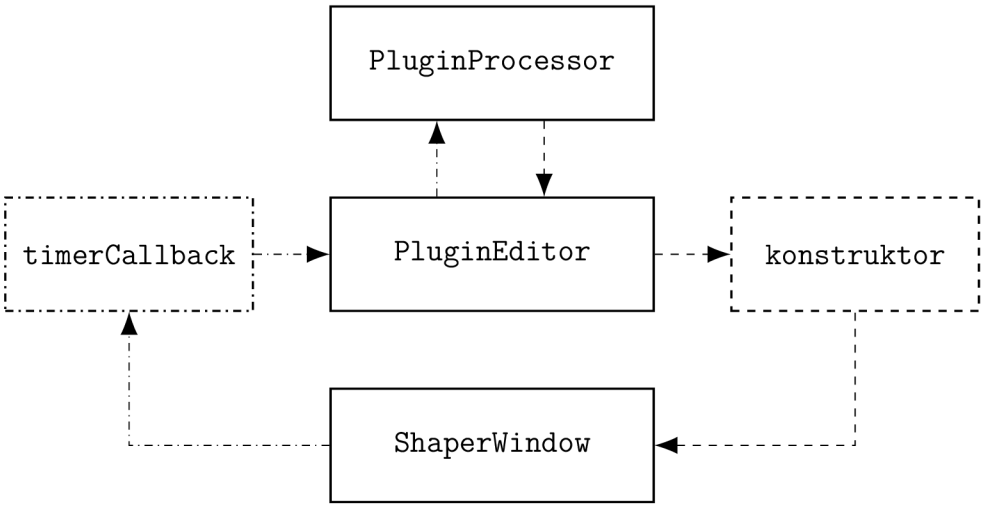
Patrně největším problémem této práce bylo nalezení vhodného řešení realizace návrhu převodní charakteristiky. Ten nově probíhá v reálném čase, což přineslo určité komplikace. Tou největší byla synchronizace uživatelského prostředí s procesorem. Jelikož je přenosová funkce definována předem neznámým počtem bodů, bylo nutné tuto synchronizaci provést jinou cestou, než tomu bylo u běžných ovládacích prvků. V následujících dvou odstavcích budou popsány použité proměnné a způsoby jejich aktualizace.

Tab. 2.3: Seznam nejdůležitějších proměnných použitých pro implementaci zkreslení.

`PluginProcessor`	`ShaperWindow`
shaperPointsProcess	
shaperPointsBuffer	shaperPoints
processorNeedsUpdate	shaperNeedsUpdate
polynomialCoefficientsReady	
polynomialCoefficients	polynomialCoefficientsShaper
shaperRangeMin	shaperRangeMin

Proměnné v prvních dvou řádcích jsou používány jako kontejnery bodů zvolených uživatelem. Ty jsou pomocí šablonové třídy `Point` s datovým typem `int` (pro `shaperPointsProcess` je použit datový typ `float`) zapsány do datového kontejneru `vector` z prostoru jmen `std`. Proměnné v dalších dvou řádcích jsou typu `bool` a slouží k řízení celé synchronizace. Na předposledním řádku jsou uvedeny datové kontejnery `vector` (tentokrát s datovým typem float), do kterých jsou zapsány koeficienty polynomu, viz odstavec o proložení polynomem v kapitole 2.2.7. Na posledním řádku jsou pak uvedeny proměnné datového typu `float` uchovávající informaci o dolní hranici rozsahu převodní charakteristiky – pro symetrické zkreslení bude nabývat hodnoty 0, pro nesymetrické pak hodnoty -1.

Synchronizace je provedena podle následujícího principu (její zjednodušená vizuální podoba je vidět na obr. 2.4):



Obr. 2.4: Princip synchronizace návrhu převodní charakteristiky.

1. Nejprve dojde v rámci třídy `PluginProcessor` k inicializaci datových kontejnerů `shaperPointsProcess` a `shaperPointsBuffer`. Každý z nich je naplněn dvěma body, jejichž spojením vznikne lineární převodní charakteristika (nebude docházet k žádnému zkreslení).
2. Obsah `shaperPointsBuffer` je při vytvoření editoru (jak již bylo zmíněno, nemusí vždy existovat) v rámci konstruktoru uložen do proměnné `shaperPoints` uvnitř třídy `ShaperWindow`. Ta obsahuje předefinování JUCE metody `paint()`, která provede vykreslení převodní charakteristiky, viz obr. 2.5.
3. Uvnitř třídy `PluginEditor` je umístěno předefinování metody `timerCallback` zděděné z JUCE třídy `Timer` (pro více informací viz [15]). Tato metoda je periodicky volána každých 30 milisekund. Při jejím provolání je provedeno několik úkonů, hlavním z nich je kontrola proměnné `shaperNeedsUpdate` – pokud je její hodnota `true` (je tomu tak po přidání, odstranění, či změně pozice bodu), dojde k předání obsahu `shaperPoints` do `shaperPointsBuffer` a proměnná `processorNeedsUpdate` je nastavena na `true`, což má za následek další manipulaci s body uvnitř procesoru. Také zde například dochází ke kontrole proměnné `polynomialCoefficientsReady` (jak její název napovídá, indikuje stav, kdy jsou koeficienty polynomu vypočteny a připraveny k použití), pokud je její hodnota `true`, dojde k jejich předání do datového kontejneru `polynomialCoefficientsShaper` a následně je provedeno překreslení přenosové funkce.

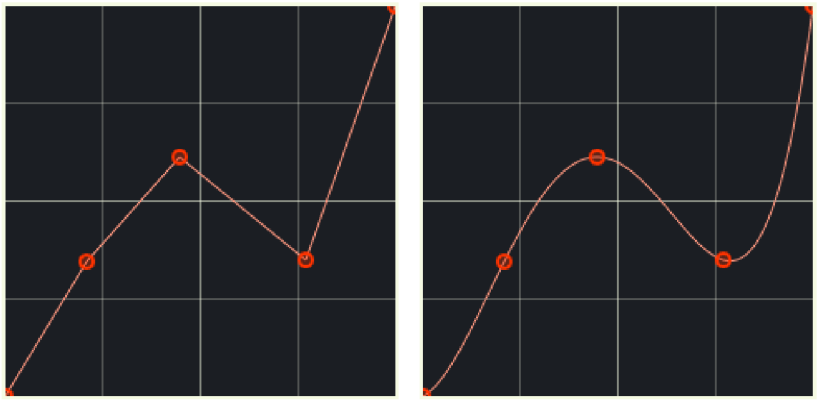
2.2.7 Zkreslení zvukového signálu

Díky přechodu na JUCE bylo možné odstranit některé nevýhody implementace zkreslení z verze realizované v Matlabu. Její hlavní nevýhodou bylo zastavení zpracování zvukových dat během editace převodní charakteristiky – v rámci prototypu to nebyl až takový problém, ovšem při použití v praxi by to značně narušovalo plynulost práce. V nové verzi není žádné zastavení nutné, editace probíhá v reálném čase. Stejně tak byl vyřešen problém s vymazáním zvolené převodní charakteristiky při přechodu mezi symetrickým a nesymetrickým zkreslením – při přechodu z nesymetrické přenosové funkce na symetrickou je nyní celý její tvar aplikován stejně na obě půlvlny a naopak. Mechanika pro manipulaci s jednotlivými body byla částečně inspirována tímto projektem [16], samotný princip je však velmi podobný tomu, který byl použit pro původní prototyp z Matlabu. Dalším vylepšením je vykreslení přenosové funkce i pro aproximaci polynomem (v předchozí verzi bylo vše vykresleno s lineárním proložením). Proložení je opět provedeno s využitím metody nejmenších čtverců, kterou však bylo potřeba implementovat manuálně, neboť moduly obsažené v JUCE metodu pro výpočet koeficientů polynomu neobsahují. Tato implementace je popsána v další části této kapitoly. Odstraněn byl i limit na maximální počet bodů určujících tvar převodní charakteristiky. Ten však nebyl v porovnání s ostatními zmíněnými nevýhodami Matlabové verze tak omezující, původních 20 bodů je dostatečný počet i pro návrh komplexnější přenosové funkce.

Zkreslení nové verze waveshaperu opět obsahuje dvě možnosti proložení uživatelem zvolených bodů, jako tomu bylo u semestrální práce. Jejich obecný princip byl vysvětlen v rámci kapitoly 2.1.2. Z tohoto důvodu zde proto bude pouze přiblížen způsob implementace v rámci jazyka C++, který se z většiny velmi podobá původní realizaci efektu.

Lineární proložení – pro realizaci tohoto typu proložení bodů byla opět směrnicová rovnice přímky, výpočet viz rovnice (2.1), (2.2) a (2.3). Metoda podílející se na realizaci zkreslení signálu je umístěna v hlavní třídě *PluginProcessor* pod názvem `updateShaper()`. Ta je volána uvnitř metody `processBlock()` ještě před zahájením zpracování zvukových dat a to pouze v případě, kdy je proměnná `processorNeedsUpdate` nastavena na `true`. Pokud tomu tak je, při jejím provolání dojde nejprve k přemapování pozic všech bodů z původního rozsahu odpovídajícího jejich umístění v okně editoru (hodnoty v pixelech) do nového rozsahu od -1 do 1 (případně od 0 do 1, pokud je aktivováno symetrické zkreslení. Pro tento účel byla použita metoda `jmap`, která v argumentu přijímá hodnotu určenou k přemapování, původní rozsah a požadovaný rozsah [17]. Body s nyní již správnými souřadnicemi jsou následně pomocí šablonové třídy `Point` s datovým typem `float` zapsány do datového kontejneru `vector` z prostoru jmen `std` s názvem `shaperPointsProcess`.

Šablonová třída `Point` obsahuje řadu metod, které ulehčují následnou manipulaci s body – v rámci této práce byly použity převážně metody vracející souřadnice daných bodů [18]. Stejně jako u realizace v Matlabu je následně v rámci cyklu `for` dle rovnic (2.1) a (2.2) proveden výpočet hodnot k a q směrnicového tvaru přímky pro dvojice po sobě jdoucích bodů. Tyto hodnoty jsou také zapsány do vlastních datových kontejnerů `vector` s názvy `shaperK` a `shaperQ`. Jejich kalkulace je tedy provedena pouze v případě úpravy bodů převodní charakteristiky. Samotné dosazení do směrnicové rovnice přímky je provedeno uvnitř metody `processBlock()`. Implementace je provedena stejným způsobem, jako tomu bylo u původní verze waveshaperu (viz výpis 2.2) – tentokrát jsou pouze použity tři cykly `for`, první prochází jednotlivé vzorky zpracovávané vyrovnávací paměti, druhý zvukové kanály a třetí hledá interval, do kterého aktuální vzorek spadá. Po jeho nalezení je provedeno dosazení do rovnice (2.3). S využitím již zmíněného principu synchronizace je také navržená převodní charakteristika vykreslena, viz obr. 2.5(a).



(a) Lineární proložení (b) Proložení polynomem

Obr. 2.5: Vykreslení převodní charakteristiky pro oba typy proložení bodů.

Proložení polynomem – jak již bylo nastíněno v úvodu této kapitoly, implementace aproximace polynomem byla tentokrát trochu složitější. Důvodem je absence nějaké obdoby funkce `polyfit` z Matlabu. Výpočet koeficientů tedy bylo nutné implementovat zvlášť. Za tímto účelem vznikla metoda, která je definována ve třídě `PluginProcessor` a má název `calculatePolynomialCoefficients()`. Pro výpočet koeficientů polynomu byla použita *metoda nejmenších čtverců* (jelikož není hlavním předmětem práce, popsána je zde pouze její část klíčová pro implementaci),

kterou lze maticově zapsat následovně [19]:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \dots x_1^m \\ 1 & x_2 & x_2^2 \dots x_2^m \\ 1 & x_3 & x_3^2 \dots x_3^m \\ \vdots & \vdots & \vdots \ddots \vdots \\ 1 & x_n & x_n^2 \dots x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \epsilon_n \end{bmatrix}, \tag{2.4}$$

kde y_1 až y_n a x_1 až x_n jsou souřadnice zvolených bodů, β_1 až β_m jsou hledané koeficienty a ϵ_1 až ϵ_n jsou odchylky.
Zjednodušený zápis pak je:

$$\vec{y} = \mathbf{X}\vec{\beta} + \vec{\epsilon}. \tag{2.5}$$

Jedná se o soustavu lineárních rovnic, jejíž řešením jsou hledané koeficienty β_m:

$$\widehat{\vec{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \vec{y}. \tag{2.6}$$

Pro práci s maticemi byla použita matematická knihovna *Eigen*, která tuto implementaci značně ulehčila. Její soubory jsou umístěny uvnitř složky se zdrojovým kódem waveshaperu, tento adresář je nutné specifikovat v nastavení projektu, viz kapitola 2.2.2. Pro účely této práce byla použita verze knihovny 3.4.0. Obsahem metody `calculatePolynomialCoefficients()` je tedy nejprve naplnění dvou matic souřadnicemi bodů uložených v `shaperPointsProcess` podle (2.4). Následně je proveden výpočet dle (2.6), jehož výsledkem jsou hledané koeficienty polynomu. V tomto výpočtu jsou použity metody z knihovny Eigen sloužící pro transpozici matice a také nalezení inverzní matice. Tyto operace byly hlavním důvodem pro použití externí knihovny, jelikož třída `Matrix` obsažená přímo v JUCE metody k nim určené postrádá [20]. Výsledné koeficienty jsou poté přepsány do datového kontejneru `polynomialCoefficients` s datovým typem `float`. Důvodem pro tento přepis je kompatibilita s konstruktorem JUCE třídy `Polynomial`, do kterého jsou koeficienty předány jako argument. Po ukončení výpočtu také dojde k nastavení proměnné `polynomialCoefficientsReady` na hodnotu `true`, což umožní přepis koeficientů uvnitř třídy `ShaperWindow` (viz kapitola 2.2.6), ve které následně dojde k vykreslení převodní charakteristiky. Jak již bylo zmíněno, pro aproximaci polynomem je použita třída `Polynomial`, jejíž instance je vytvořena, pokud je zvolen tento typ proložení. V argumentu jsou předány vypočtené koeficienty uložené v `polynomialCoefficients` a jejich počet, který odpovídá řádu vytvořeného polynomu. Jeho řád se mění v závislosti na počtu uživatelem zvolených bodů, stejně tomu bylo i v případě prvního prototypu efektu. V cyklu určeném ke zpracování vzorků uvnitř `processBlock()` je pak použita metoda `operator()()`, která jako

argument přijímá daný vstupní vzorek a její návratovou hodnotou je výsledek po dosazení do daného polynomu [21]. Vykreslení výsledného polynomu v rámci třídy `ShaperWindow` je provedeno s použitím cyklu `for` – pro dva po sobě jdoucí pixely na obrazovce jsou vždy pomocí již zmíněné metody vypočteny funkční hodnoty, které jsou následně spojeny přímkou, viz obr. 2.5(b).

2.2.8 Filtrace zvukového signálu

I filtrace se dočkala v rámci bakalářské práce několika vylepšení. Obecné informace týkající se tohoto typu úpravy signálu byly uvedeny v rámci kapitoly 2.1.3, následuje proto pouze popis změn a přiblížení implementace s využitím frameworku JUCE. Hlavní výhodou této verze waveshaperu je možnost použití více typů filtrů současně. Dostupné jsou tyto typy:

- horní propust (high-pass),
- low-shelving,
- pásmová propust (bandpass),
- peak,
- high-shelving,
- dolní propust (low-pass),
- filtr stejnosměrné složky.

Filtrace probíhá postupně v právě uvedeném pořadí. Jednotlivé filtry je možné podle potřeby aktivovat či deaktivovat. V případě horní a dolní propusti je také možné volit strmost (na výběr je 12, 24, 36 a 48 dB/okt). Vyšších strmostí je docíleno kaskádním zapojením dvou až čtyř stejných filtrů. Nevýhodou tohoto provedení je, že při zvolení vyšší strmosti mezní kmitočet f_C a činitel jakosti filtru Q přesně neodpovídá hodnotám, které jsou zobrazeny uživateli (např. při kaskádním zapojení dvou stejných filtrů nebude na zvoleném mezním kmitočtu pokles o 3 dB, nýbrž 6 dB – skutečný mezní kmitočet bude posunutý). V případě nastavení vyššího činitele jakosti Q by také při zapojení více filtrů za sebou vznikla ve výsledném signálu velká rezonance, jeho hodnota je proto ještě před dosazením do návrhových vzorců vydělena počtem použitých filtrů. Tyto nepřesnosti by bylo možné odstranit přizpůsobením výpočtu koeficientů v závislosti na aktuálně zvolené strmosti (úpravou činitele jakosti podle řádu filtru se zabývá například tento příspěvek [22]), jelikož však není účelem exaktní filtrace (uživatel parametry nastavuje sluchem), nýbrž pouze možnost ovlivnění barvy, jednoduchost implementace zde převažuje zmíněnou chybu v zobrazených hodnotách. Dalším vylepšením je volba umístění filtrů v rámci signálového toku – kmitočtová úprava může být provedena buď ještě před zkreslením (možnost *Pre*), nebo až po něm (*Post*), viz obr. 2.3. Byly také přidány nové typy filtrů, konkrétně jde o tyto čtyři: low-shelving, peak, high-shelving a filtr stejnosměrné složky.

Jejich implementace byla opět provedena s použitím návrhových vzorců pro výpočet koeficientů IIR filtru 2. řádu podle [4]. Tyto vzorce jsou uvedeny v tab. 2.4,

Tab. 2.4: Výpočet koeficientů IIR filtru 2. řádu [4].

	b_0	b_1	b_2	a_1	a_2
a)	$\frac{1+\sqrt{2V_0}K+V_0K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(V_0K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2V_0}K+V_0K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$
b)	$\frac{V_0(1+\sqrt{2}K+K^2)}{V_0+\sqrt{2V_0}K+K^2}$	$\frac{2V_0(K^2-1)}{V_0+\sqrt{2V_0}K+K^2}$	$\frac{V_0(1-\sqrt{2}K+K^2)}{V_0+\sqrt{2V_0}K+K^2}$	$\frac{2(K^2-V_0)}{V_0+\sqrt{2V_0}K+K^2}$	$\frac{V_0-\sqrt{2V_0}K+K^2}{V_0+\sqrt{2V_0}K+K^2}$
c)	$\frac{V_0+\sqrt{2V_0}K+K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-V_0)}{1+\sqrt{2}K+K^2}$	$\frac{V_0-\sqrt{2V_0}K+K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$
d)	$\frac{V_0(1+\sqrt{2}K+K^2)}{1+\sqrt{2V_0}K+V_0K^2}$	$\frac{2V_0(K^2-1)}{1+\sqrt{2V_0}K+V_0K^2}$	$\frac{V_0(1-\sqrt{2}K+K^2)}{1+\sqrt{2V_0}K+V_0K^2}$	$\frac{2(V_0K^2-1)}{1+\sqrt{2V_0}K+V_0K^2}$	$\frac{1-\sqrt{2V_0}K+V_0K^2}{1+\sqrt{2V_0}K+V_0K^2}$
e)	$\frac{1+\frac{V_0}{Q}K+K^2}{1+\frac{1}{Q}K+K^2}$	$\frac{2(K^2-1)}{1+\frac{1}{Q}K+K^2}$	$\frac{1-\frac{V_0}{Q}K+K^2}{1+\frac{1}{Q}K+K^2}$	$\frac{2(K^2-1)}{1+\frac{1}{Q}K+K^2}$	$\frac{1-\frac{1}{Q}K+K^2}{1+\frac{1}{Q}K+K^2}$
f)	$\frac{1+\frac{1}{Q}K+K^2}{1+\frac{1}{V_0Q}K+K^2}$	$\frac{2(K^2-1)}{1+\frac{1}{V_0Q}K+K^2}$	$\frac{1-\frac{1}{Q}K+K^2}{1+\frac{1}{V_0Q}K+K^2}$	$\frac{2(K^2-1)}{1+\frac{1}{V_0Q}K+K^2}$	$\frac{1-\frac{1}{V_0Q}K+K^2}{1+\frac{1}{V_0Q}K+K^2}$

Poznámka k tab. 2.4:
a) LF boost, b) LF cut, c) HF boost, d) HF cut, e) Peak boost, f) Peak cut

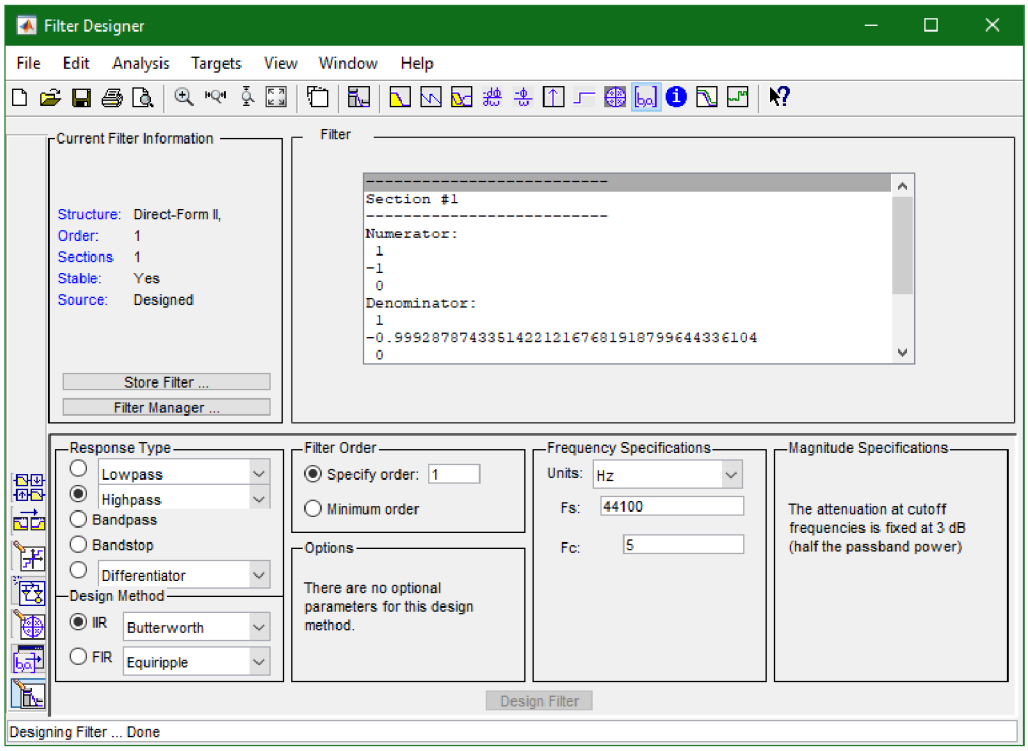
kde $K = \tan(\pi f_C/f_S)$, Q je činitel jakosti filtru, f_C je mezní/střední kmitočet filtru, f_S je vzorkovací kmitočet, $V_0 = 10^{G/20}$ a G je zesílení daného filtru. Samotná implementace kmitočtové filtrace je provedena uvnitř třídy `PluginProcessor`. Byla k ní použita JUCE třída `dsp::IIR::Filter`, která je určena přímo pro filtry s nekonečnou impulsní odezvou [23]. Za účelem minimalizace opakování kódu a tím i zlepšení jeho přehlednosti byly použity šablony funkcí, které byly zapsány převážně podle následujícího návodu [24]. Daný kód byl následně upraven a rozšířen, jelikož ve zmíněném návodu jsou realizovány pouze tři typy filtrů a jejich koeficienty jsou vypočteny pomocí předdefinovaných metod obsažených přímo v JUCE. Výpočty koeficientů filtrů implementovaných v této bakalářské práci jsou provedeny v rámci metod pojmenovaných jako `updateTyp_filtru` (např. `updateHighpassFilter`) – použité návrhové vzorce (viz tab. 2.1 a 2.4) jsou upraveny tak, aby byly výpočty opakujících se částí provedeny pouze jednou (např. mocniny, odmocniny, totožné jmenovatele zlomků). Vypočtené koeficienty jsou následně uvnitř struktury `dsp::IIR:Coefficients` (viz její dokumentace [25]) předány danému filtru. Každá ze zmíněných metod je poté volána z metody `updateFilters()`, která všechny z nich

sdružuje dohromady a každé předá v argumentu nastavení daného filtru (např. hodnota mezního/středního kmitočtu f_C, činitele jakosti Q, a další). Kontrolou aktuálního stavu tlačítka *Pre/Post* je poté v rámci metody `processBlock()` rozhodnuto, zda bude filtrace provedena před, nebo až po zkreslení.

Filtr stejnosměrné složky – jak je vidět na obr. 2.3, téměř na konci signálového toku waveshaperu je umístěn DC filtr. Jeho účelem je odstranění složek s velmi nízkými kmitočty, které mohly vzniknout v důsledku zkreslení zvukového signálu. Jeho implementace je provedena stejným způsobem, jako tomu bylo u všech předchozích typů, tentokrát se však jedná o filtr prvního řádu (ten má sice nižší strmost, ale také menší míru zvlnění fázové charakteristiky). K návrhu jeho koeficientů byl použit nástroj *Filter Designer*, který je součástí Matlabu, viz obr. 2.6. Jak je na tomto obrázku vidět, jako mezní kmitočet f_C filtru typu horní propusti byla zvolena hodnota 5 Hz. Použité koeficienty tedy jsou rovny:

- $a_0 = 1$,
- $a_1 = -0,999$,
- $b_0 = 1$,
- $b_1 = -1$.

Platí, že čím je koeficient a_1 bližší hodnotě -1, tím je mezní kmitočet filtru f_C nižší. Tyto koeficienty jsou opět dosazeny do rovnice (1.7), neboť při dosazení $a_2 = b_2 = 0$ vznikne filtr prvního řádu [4].



Obr. 2.6: Určení koeficientů filtru prvního řádu pomocí nástroje Filter Designer.

2.2.9 Převzorkování zvukového signálu

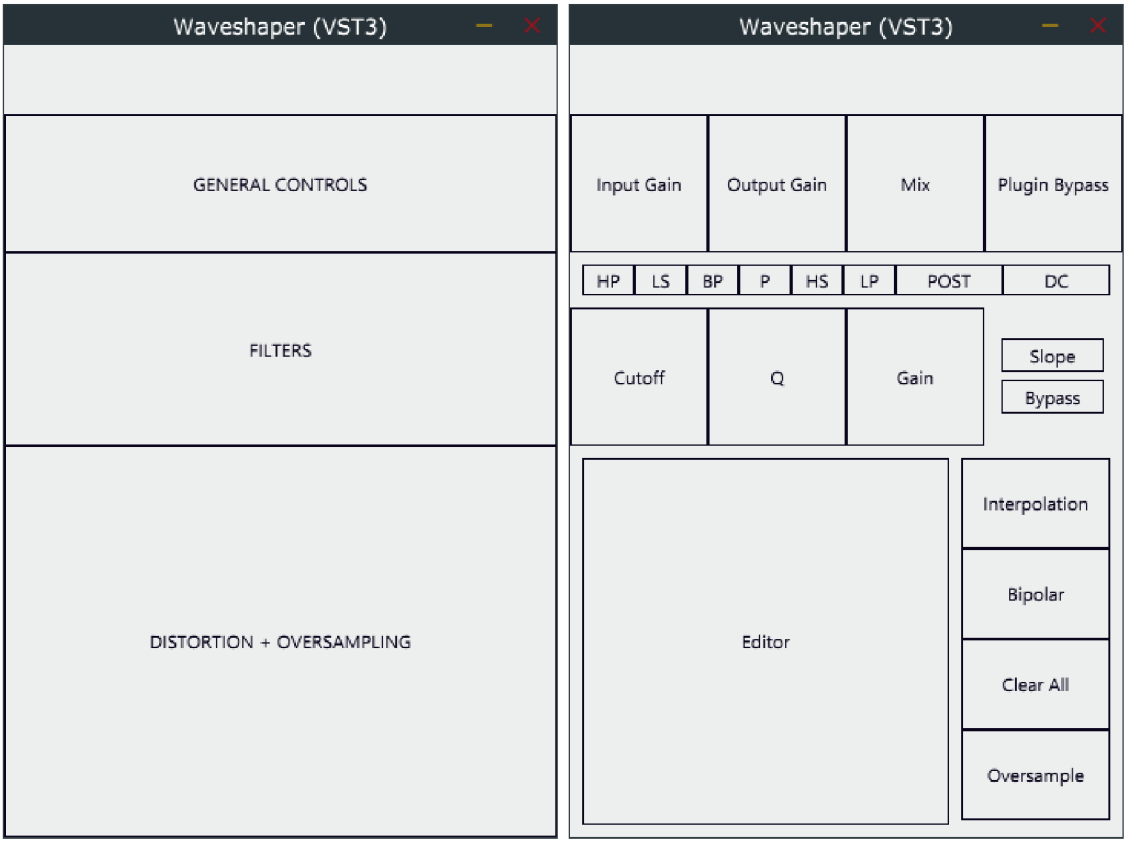
Posledním funkčním prvkem pluginu, který byl implementován, bylo převzorkování signálu. Jeho účel a princip byl popsán v rámci teoretického úvodu, viz kapitola 1.2. Pro realizaci převzorkování byla použita JUCE třída `Oversampling`, která podporuje následující stupně převzorkování: 2, 4, 8 a 16 (viz její dokumentace [26]). Pro každý z nich byl uvnitř konstruktoru třídy `PluginProcessor` vytvořen nový objekt této třídy. Jako argument je jejímu konstruktoru předán počet vstupních kanálů (pomocí metody `getTotalNumInputChannels()`), činitel převzorkování a typ filtru, který bude během procesu převzorkování použit (v tomto případě byl zvolen typ `filterHalfBandFIREquiripple`, který má sice vyšší latenci, ovšem při jeho použití u realizovaného nelineárního efektu dojde k většímu potlačení aliasingu [26]). Uvnitř metody `prepareToPlay()` je následně pro každý z vytvořených objektů zavolána metoda `initProcessing()`, která je připraví na zpracování signálu (dojde k nastavení velikosti vnitřních vyrovnávacích pamětí daného objektu). V rámci metody `processBlock()` je pomocí přepínače (`switch`) v závislosti na uživatelem nastaveném stupni převzorkování pro zvolený objekt zavolána metoda `processSamplesUp()`, která provede nadvzorkování aktuálně zpracovávané vyrovnávací paměti. Tento signál je následně podle postupu popsaného v kapitole 2.2.7 zkreslen a poté pomocí metody `processSamplesDown()` zpět podvzorkován. Podvzorkování je opět provedeno s použitím přepínače.

2.2.10 Uživatelské prostředí

Na rozdíl od realizace v Matlabu, kde bylo uživatelské prostředí vytvořeno umístěním všech prvků do jednoduché mřížky definované pouze šířkou sloupců a výškou řádků, jsou možnosti frameworku JUCE mnohem rozsáhlejší (např. je zde větší výběr připravených ovládacích prvků, animace, práce s 3D modely, apod.).

Rozložení ovládacích prvků – byl použit princip postupného dělení celého okna na jednotlivé obdélníky (třída `Rectangle`), do kterých jsou následně v rámci metody `resized()` uvnitř třídy `PluginEditor` umístěny jednotlivé prvky. Výhodou tohoto systému je, že uživateli umožňuje manipulaci s rozměry okna (stačí tuto možnost pouze povolit). Vytvořený waveshaper tuto funkcionalitu bohužel nemá, jelikož by tomu bylo nutné přizpůsobit systém editace převodní charakteristiky, u ostatních prvků pluginu by žádné komplikace nebyly, jelikož jsou vykresleny vektorově. Proces návrhu rozložení pluginu je znázorněn na obr. 2.7 – nejprve vznikly hlavní sekce okna, které byly poté rozděleny na dílčí oblasti.

Vlastní vzhled ovládacích prvků – JUCE také umožňuje přizpůsobení podoby předdefinovaných ovladačů podle potřeb vývojáře. K tomuto účelu slouží JUCE třída `LookAndFeel_V4`, pomocí které je možné předefinovat vykreslení zvoleného



(a) Hlavní sekce (b) Jednotlivé prvky

Obr. 2.7: Rozvržení uživatelského prostředí pomocí postupného dělení okna.

prvku [27]. Z důvodu poměrně velké rozsáhlosti JUCE tříd zaměřených na úpravu uživatelského prostředí vyvíjeného softwaru byly úpravy provedené pro potřeby této bakalářské práce inspirovány převážně tímto návodem [28] zabývajícím se úpravou vykreslení otočného potenciometru.

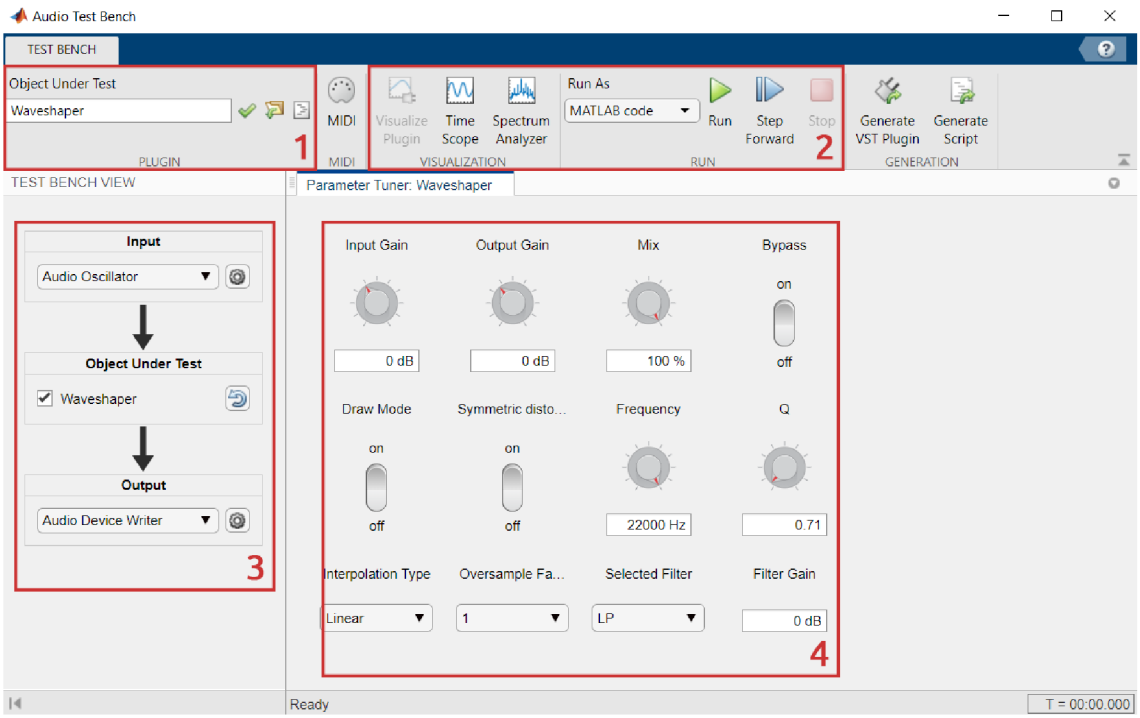
3 Testování realizovaných modulů

V této kapitole je nejprve uvedeno, jakým způsobem bylo provedeno testování obou realizovaných plugin modulů. Dále je zde popis uživatelského prostředí každého z nich, po kterém následují grafické ukázky. Poslední částí této kapitoly je porovnání obou realizací efektu.

3.1 Matlab verze pluginu

3.1.1 Audio Test Bench

Pro testování navrženého prototypu pluginu slouží rozhraní Audio Test Bench. Jeho spuštění lze provést pomocí příkazu `audioTestBench()`, v jehož argumentu uvedeme název třídy se zdrojovým kódem pluginu. Pro účely testování navrženého pluginu je tedy potřeba použít příkaz: `audioTestBench(Waveshaper)`. Následně dojde k otevření okna, které je vidět na obr. 3.1.



Obr. 3.1: Rozhraní Audio Test Bench pro testování prototypu pluginu.

Jsou zde vyznačeny čtyři důležité oblasti:

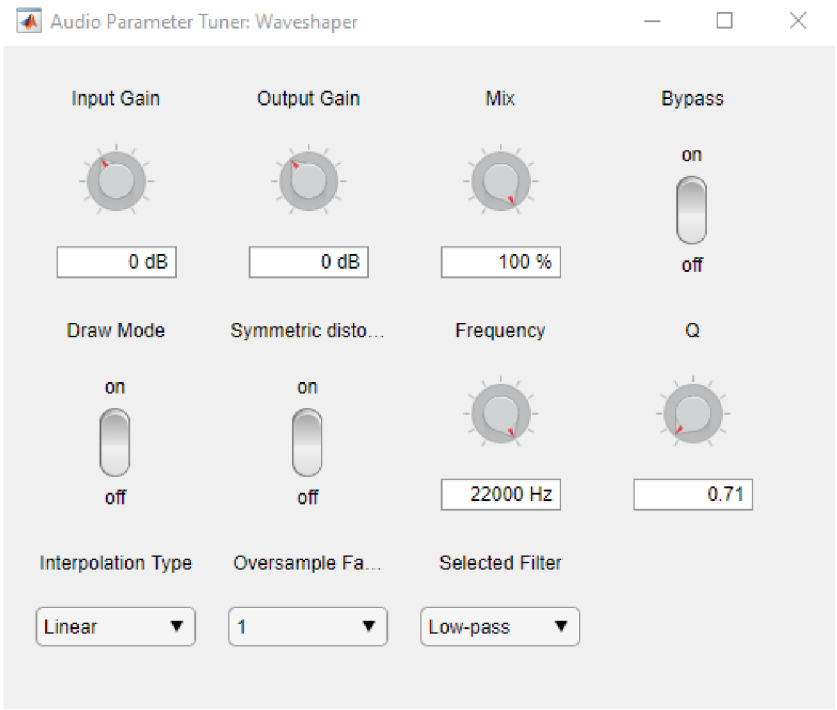
1. Zvolení objektu, který má být testován. Nemusí se však jednat pouze o pluginy naprogramované přímo v Matlabu, importovat lze i klasické VST efekty ve

formátech .dll a .vst3. V takovém případě jsou i jejich parametry zobrazeny v základním vzhledu (viz obr. 3.14).

2. Možnosti spuštění a vizualizace pluginu. Pro vlastní prototypy lze volit ze dvou možností spuštění, v případě *MATLAB code* poběží testované zařízení přímo uvnitř Matlabu, u možnosti *VST plugin* dojde nejprve k převedení na technologii VST. Dále si můžeme pomocí tlačítek z nabídky *Visualization* zobrazit časový průběh signálu a jeho modulovou kmitočtovou charakteristiku. Obě možnosti vizualizace obsahují další funkce pro podrobnější analýzu signálu (např. kurzory, měření harmonického a intermodulačního zkreslení).
3. Nastavení vstupu a výstupu. Jako zdroj zvukového signálu lze vybrat z několika možností. Nejdůležitějšími vstupy jsou: zvukový soubor, signál ze zvukové karty (využití např. pro mikrofon) a generátor signálu s výběrem požadovaného průběhu (sinus, obdélník a pila).
4. Uživatelské prostředí navrhovaného prototypu.

3.1.2 Uživatelské prostředí

Rozložení ovládacích prvků v navrženém pluginu je vidět na obr. 3.2. Je zde vidět rozdělení do tří hlavních skupin, z nichž každá je více přiblížena v následujících podkapitolách.



Obr. 3.2: Uživatelské prostředí navrženého prototypu (Matlab).

Základní ovládací prvky

První řádek obsahuje ovládací prvky týkající se pluginu jako takového. Je tady tedy vstupní a výstupní zesílení, ovládání mixu pro případné smíchání zpracovaného i čistého signálu a také možnost celkového potlačení efektu.

Zkreslení

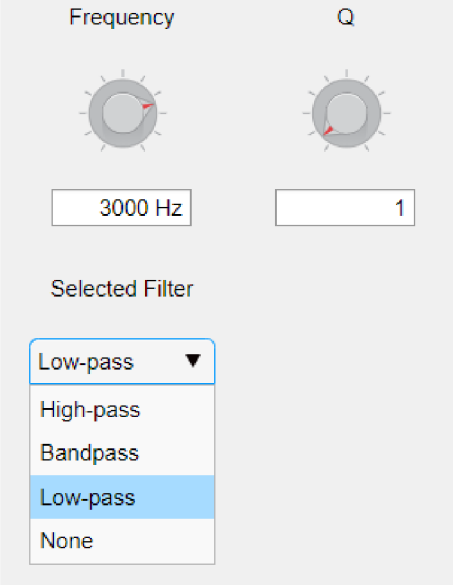
V levé dolní části se pak nachází skupina ovládacích prvků týkajících se zkreslení zvukového signálu. Je zde aktivace módu pro editaci přenosové funkce, která probíhá v offline režimu – zpracování signálu se tedy obnoví až po ukončení tohoto módu (stlačením tlačítka ESC). Nové body převodní charakteristiky lze vytvářet stiskem levého tlačítka, odstranění bodu je provedeno stisknutím pravého tlačítka v jeho blízkosti. Přesun bodu na jinou pozici není možný, je nutné jej odstranit a vytvořit nový. Další možností je změna mezi zkreslením pomocí symetrické/nesymetrické přenosové funkce. Její průběh je po stisknutí tohoto tlačítka resetován, je tedy nutné po přepnutí mezi jednotlivými typy zkreslení funkci navrhnout znovu. Dalším prvkem je výběr mezi dvěma typy interpolace. Zvolené body přenosové funkce mohou být proloženy buď lineárně, nebo polynomem. Poslední možností týkající se zkreslení, v tomto případě tedy nepřímo, je volba stupně převzorkování signálu. Na výběr je z několika stupňů: 1, 2, 4 a 8. Stupeň 1 zde odpovídá původnímu nepřevzorkovanému signálu.

Filtrace

V pravém dolním rohu se nachází poslední skupina ovládacích prvků, které se týkají filtrace signálu. Zpracování signálu pomocí filtrů probíhá ještě před nadvzorkováním (viz obr. 2.1). Pro jejich ovládání slouží dva otočné potenciometry, jeden mění mezní kmitočet f_C a druhý pak nastavuje činitel jakosti filtru Q. Posledním prvkem je rozbalovací seznam dostupných typu filtrů, který je vidět na obr. 3.3.

3.1.3 Zvukové ukázky

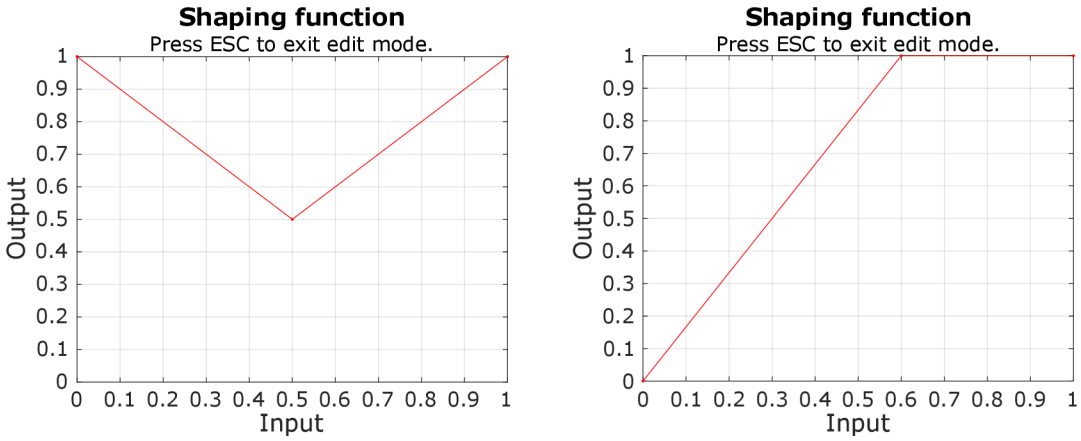
Poslední částí této kapitoly je demonstrace funkčnosti navrženého pluginu s přiloženými zvukovými ukázkami. Pro záznam všech nahrávek byl využit pouze Audio Test Bench, žádné další úpravy na nich nebyly provedeny. Jako zdroj vstupního signálu byl využit zvukový soubor (nahrávka bicích) a generátor signálu také přímo z Audio Test Bench (viz kapitola 3.1.1). Nastavení jednotlivých parametrů je následně uvedeno zvlášť pro každou ukázku.



Obr. 3.3: Nabídka filtrů.

Ukázky zkreslení

Zkreslení vstupního signálu je zde předvedeno v několika různých případech. První ukázkou je zkreslení s **lineárním proložením bodů**. Vstupním signálem byl sinusový průběh s kmitočtem 300 Hz. Nebylo použito převzorkování ani žádný filtr. Pro tento vstup byla využita sudá i lichá přenosová funkce (obr. 3.4) za účelem potvrzení výroku z kapitoly 2.1.2.

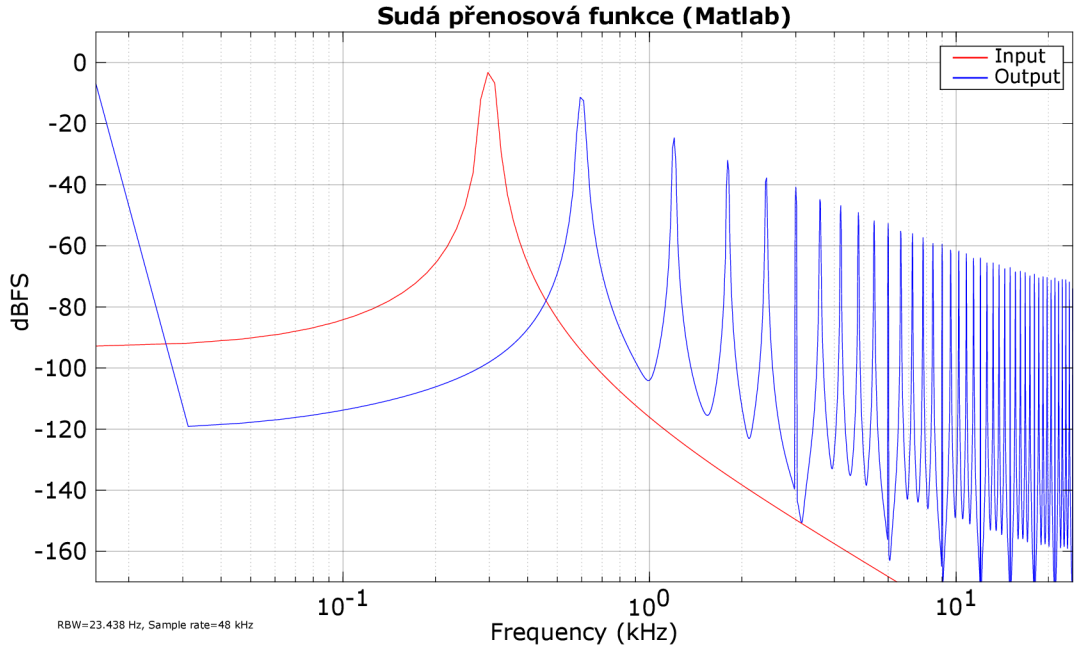


(a) Sudá přenosová funkce

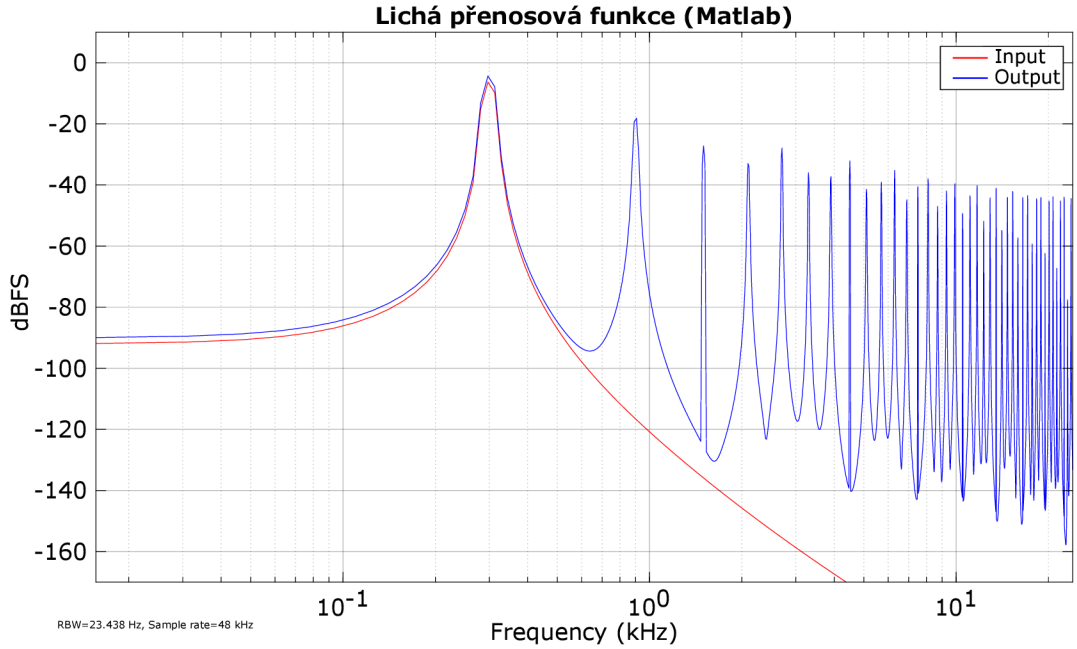
(b) Lichá přenosová funkce

Obr. 3.4: Převodní charakteristiky pro první demonstraci zkreslení.

Spektra zkreslených signálů jsou vidět na obr. 3.5. Zde můžeme vidět, že v případě sudé funkce opravdu vzniknou pouze sudé harmonické složky, pro lichou funkci naopak pouze liché.



(a) Sudá přenosová funkce



(b) Lichá přenosová funkce

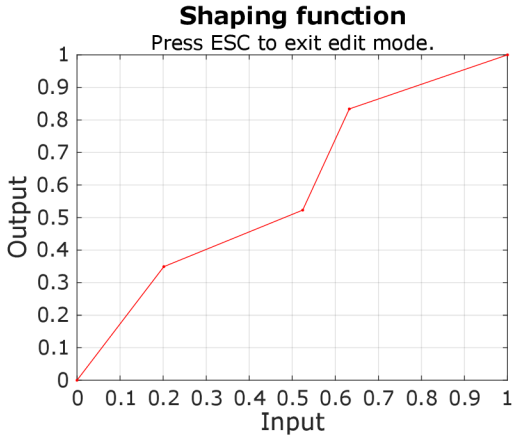
Obr. 3.5: Spektra výstupních signálů po zkreslení pomocí sudé a liché přenosové funkce.

Zmíněných ukázek zkreslení se týkají následující tři zvukové soubory obsažené v elektronické příloze: `sine_300Hz.wav`, `sine_300Hz_suda_funkce_matlab.wav` a `sine_300Hz_licha_funkce_matlab.wav`.

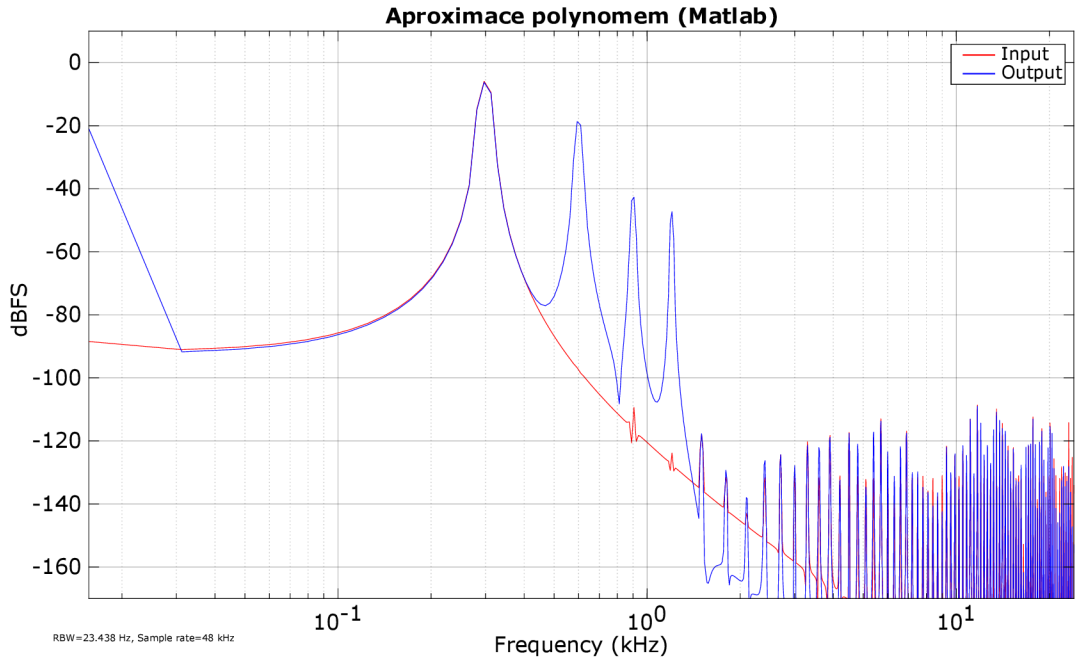
Další ukázkou (v tomto případě pouze zvukovou) je zkreslení nahrávky bicích.

Byla opět využita stejná lichá přenosová funkce (viz obr. 3.4), aktivován nebyl žádný filtr a byl nastaven osmý stupeň převzorkování. Vstupní a zkreslený signál je uložen v souborech: `bici.wav` a `bici_zkreslene_matlab.wav`.

Poslední ukázka se týká zkreslení s **body proloženými polynomem**. Vstupním signálem byl opět sinusový průběh s kmitočtem 300 Hz. Zvolená převodní charakteristika je vidět na obr. 3.6. Nebylo zde použito žádné převzorkování ani filtr. Na kmitočtovém spektru (viz obr. 3.7) si můžeme všimnout, že na výstupu jsou celkově pouze čtyři harmonické složky.



Obr. 3.6: Převodní charakteristika s body proloženými polynomem.



Obr. 3.7: Kmitočtové spektrum výstupního signálu po zkreslení (aproximace polynomem).

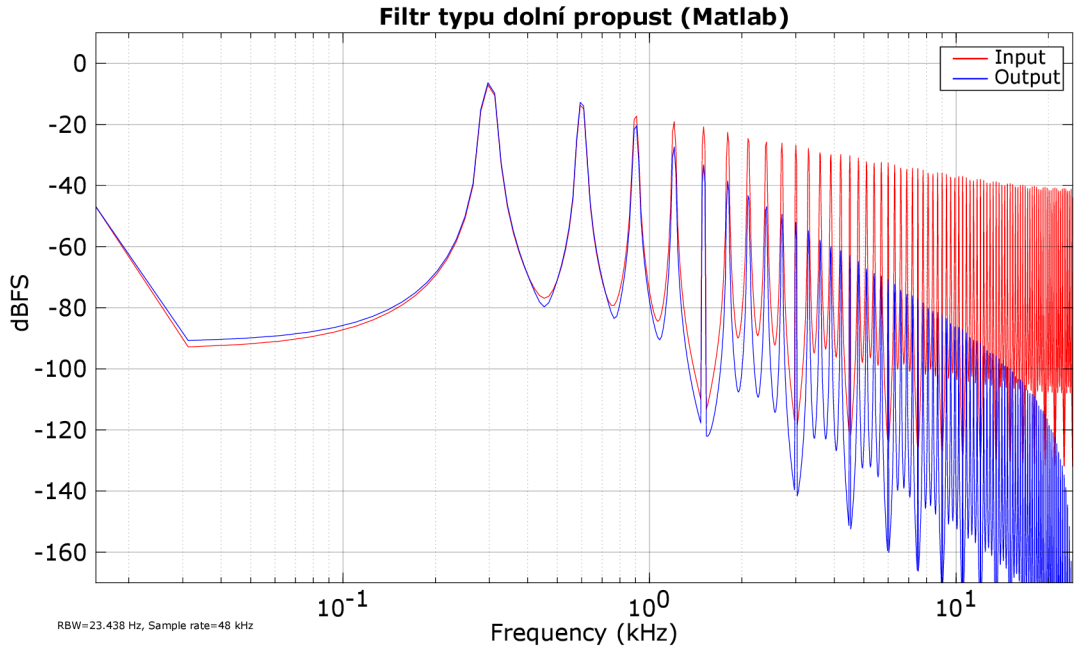
Příčinou je použitá přenosová funkce, která obsahuje pět zvolených bodů – dojde tedy k proložení polynomem čtvrtého řádu. Jak již bylo zmíněno v kapitole 2.1.2, pro aproximaci pomocí polynomu platí, že na výstupu vzniknou pouze harmonické složky, jejichž pořadové číslo odpovídá nejvýše řádu polynomu [2]. Tato ukázka je uložena v souboru `sine_300Hz_polynom_matlab.wav`.

Ukázky filtrace

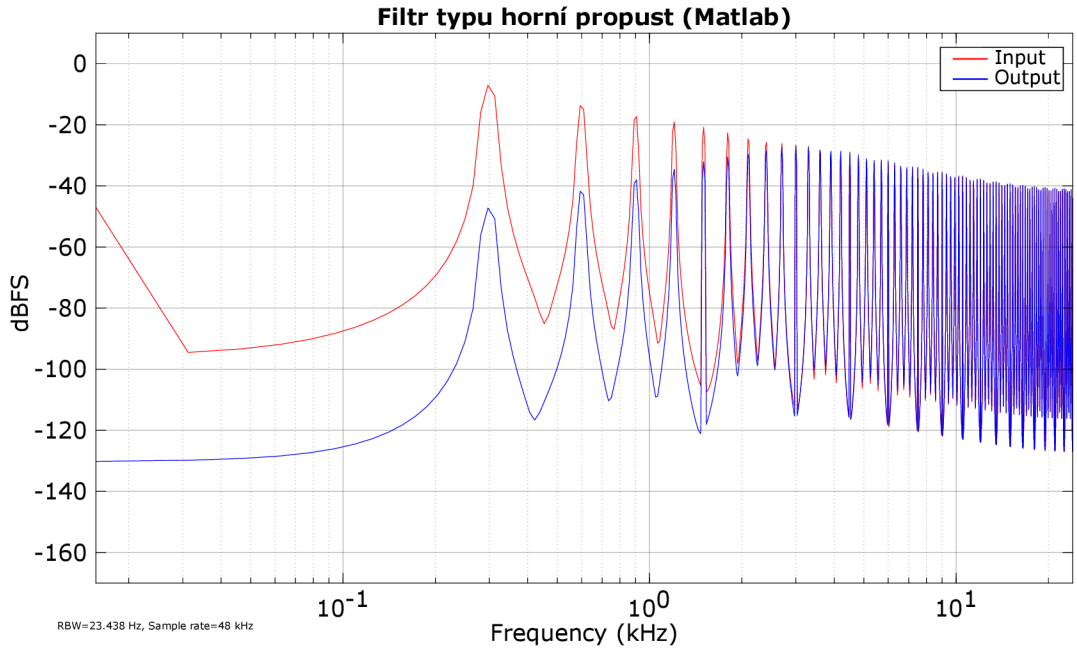
Následují tři ukázky týkající se jednotlivých typů filtrů. Jako vstupní signál byl použit pilový průběh s kmitočtem 300 Hz. Na něj byly postupně aplikovány všechny tři implementované typy filtrů s následujícím nastavením:

- **Dolní propust** (obr. 3.8) – $f_C = 700\,\text{Hz}$, $Q = 1$. (`300Hz_sawtooth_700Hz_lowpass_matlab`)
- **Horní propust** (obr. 3.9) – $f_C = 3\,\text{kHz}$, $Q = 1$. (`300Hz_sawtooth_3kHz_highpass_matlab`)
- **Pásmová propust** (obr. 3.10) – $f_C = 3\,\text{kHz}$, $Q = 5$, výst. zesílení $= 6\,\text{dB}$. (`300Hz_sawtooth_3kHz_bandpass_matlab`)

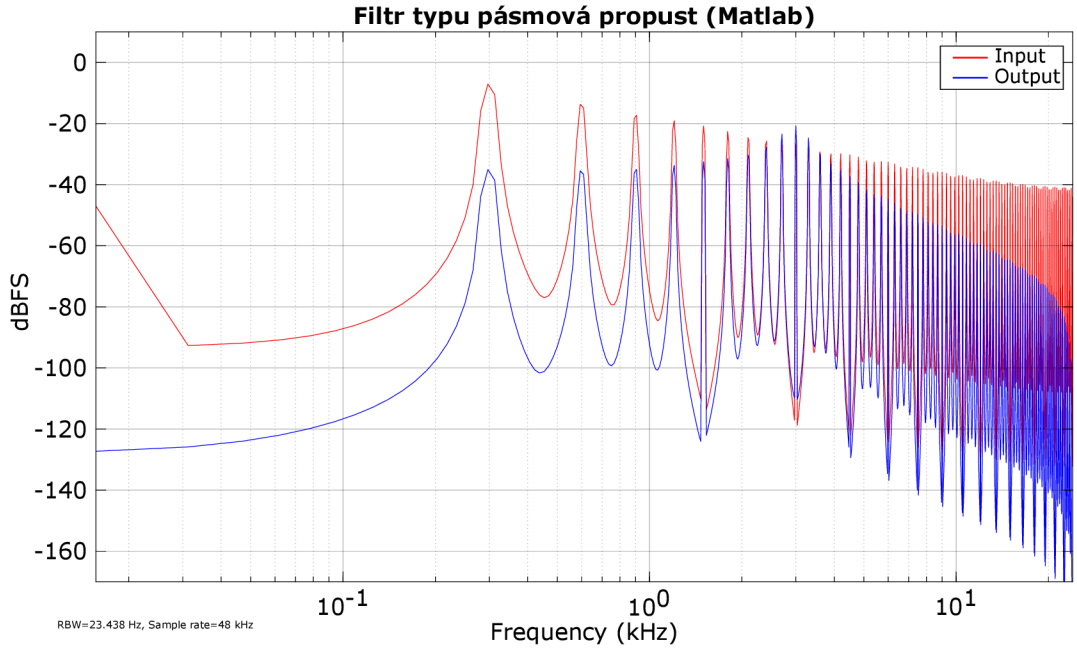
Zvukové soubory jsou obsaženy v rámci elektronické přílohy.



Obr. 3.8: Kmitočtové spektrum výstupního signálu po filtraci (dolní propust).



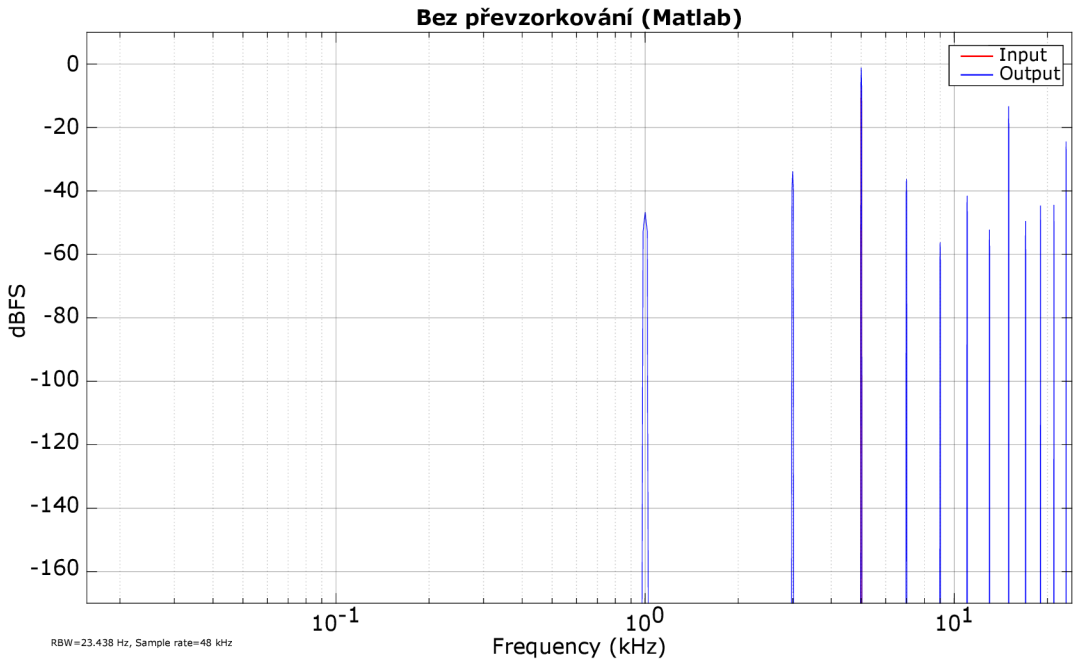
Obr. 3.9: Kmitočtové spektrum výstupního signálu po filtraci (horní propust).



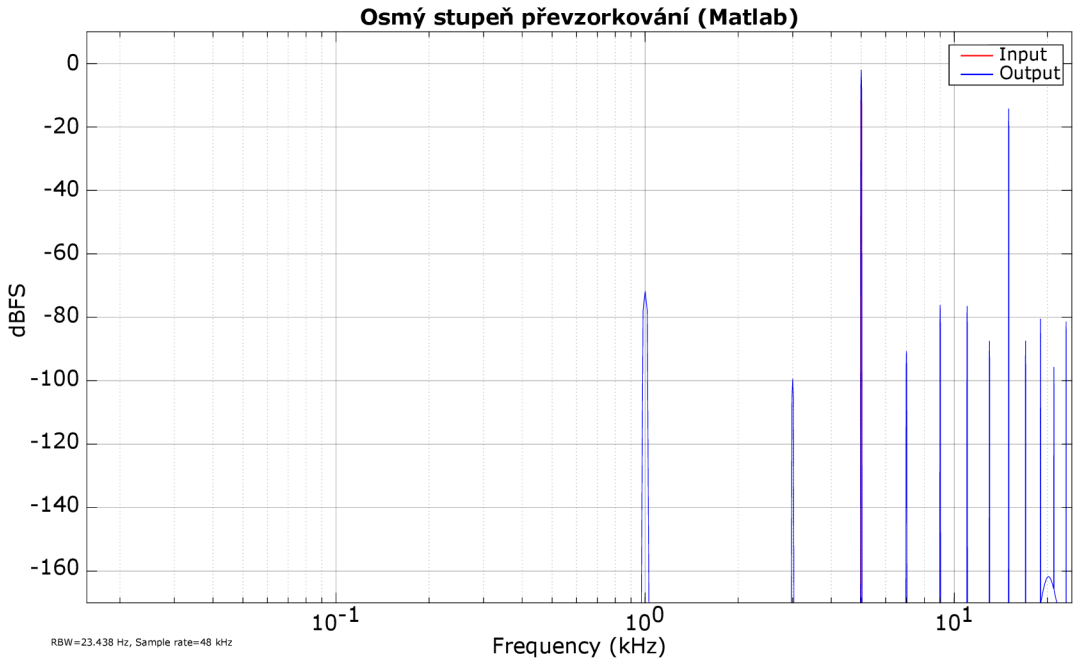
Obr. 3.10: Kmitočtové spektrum výstupního signálu po filtraci (pásmová propust).

Ukázka převzorkování

Poslední ukázkou je demonstrace vlivu převzorkování na zkreslený výstupní signál, viz obr. 3.11 a 3.12.



Obr. 3.11: Kmitočtové spektrum výstupního signálu bez použití převzorkování.



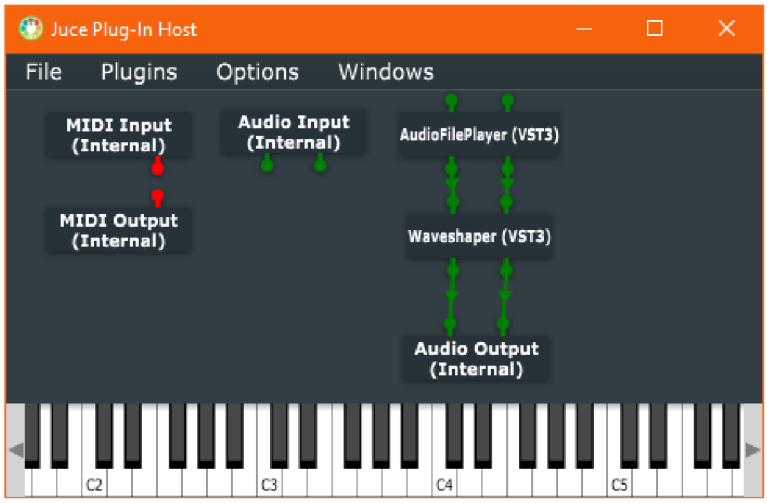
Obr. 3.12: Kmitočtové spektrum výstupního signálu s použitím osmého stupně převzorkování.

Pro tuto ukázku byl využit vstupní signál sinusového průběhu o kmitočtu 5 kHz, u kterého bylo provedeno tvrdé ořezání (tzv. hard clipping). To bylo realizováno pouhým zvýšením vstupního zesílení o 10 dB. Výsledný úbytek aliasingových složek je vidět na obr. 3.11 a 3.12, obsahující dvě výsledná kmitočtová spektra – jedno před použitím převzorkování a druhé po nastavení osmého stupně převzorkování. Zvukové soubory této demonstrace jsou umístěny v elektronické příloze.

3.2 Juce verze pluginu

3.2.1 Juce Plug-In Host

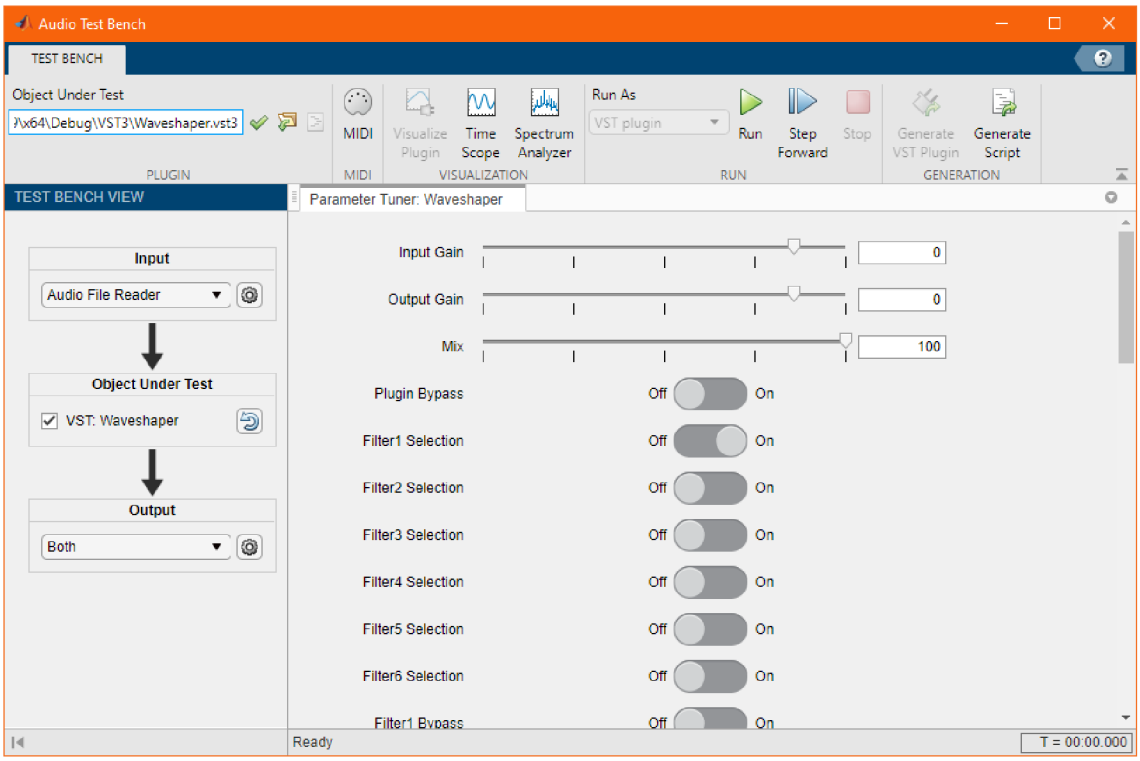
Pro testování waveshaperu v průběhu práce na něm byl použit nástroj Juce Plug-In Host, ve kterém je možné různě kombinovat zvukové efekty, viz obr. 3.13. Na vstup testovaného prototypu byl přiveden AudioFilePlayer (jedná se konverzi dema obsaženého v JUCE do formátu VST, viz [29]), který přehrával požadovaný vstupní signál.



Obr. 3.13: Grafické rozhraní nástroje Juce Plug-In Host.

3.2.2 Audio Test Bench

K vytvoření grafických ukázek byl opět použit Audio Test Bench a jeho nástroj pro zobrazení kmitočtového spektra. Jak již bylo zmíněno v kapitole 3.1.1, i při testování externích pluginů ve formátu VST je zobrazeno pouze jednoduché uživatelské prostředí se všemi dostupnými parametry, viz obr. 3.14. V případě této práce tedy bylo nutné mít požadovanou přenosovou charakteristiku definovanou již při spuštění waveshaperu, jelikož okno s její editací není uvnitř Audio Test Bench dostupné.

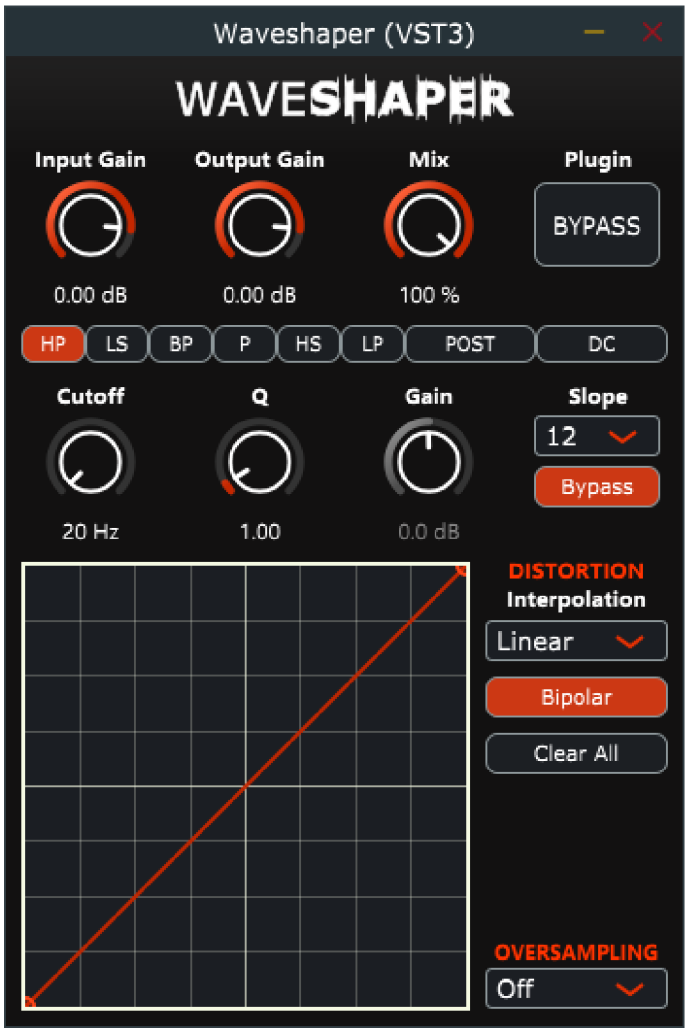


Obr. 3.14: Zobrazení JUCE verze pluginu uvnitř nástroje Audio Test Bench.

3.2.3 Uživatelské prostředí

Rozložení ovládacích prvků v této nové implementaci waveshaperu je vidět na obr. 3.15. Jak již bylo zmíněno, framework JUCE umožňuje poměrně rozsáhlý zásah do vizuální stránky vyvíjené aplikace. V porovnání s prototypem z Matlabu je zde velký rozdíl i přes to, že nebylo provedeno až tak velké množství úprav. Hlavním zlepšením tohoto prototypu je především větší plynulost způsobená umožněním práce v reálném čase a absencí druhého okna s editorem přenosové funkce. Plocha pluginu je opět pomyslně rozdělena do tří hlavních částí, které jsou blíže představeny ve zbytku této podkapitoly.

Podobu uživatelského prostředí je možné částečně měnit úpravou barev definovaných v prostoru jmen `ColorPalette`, ve kterém jsou uvedeny nejdůležitější barevné odstíny použité při vykreslování waveshaperu. Změnou jednoho řádku zdrojového kódu tak lze dosáhnout např. těchto výsledků, viz obr. 3.16. Úpravou dalších barev ve zmíněném prostoru jmen je také možné jednoduše vytvořit například světlou verzi uživatelského prostředí, viz obr. B.1 umístěný v příloze. Tyto úpravy je však v tuto chvíli možné provést pouze před kompilací pluginu do formátu VST, pro případnou změnu odstínů použitých k vykreslení uživatelského prostředí během chodu by bylo nutné zdrojový kód ještě částečně přepracovat.



Obr. 3.15: Uživatelské prostředí waveshaperu (JUCE).

Základní ovládací prvky

V prvním řádku jsou umístěny ovladače týkající se pluginu jako celku, je zde úprava vstupního a výstupního zesílení, možnost smíchání zpracovaného signálu s čistým signálem na vstupu a také tlačítko pro celkovou deaktivaci zpracování signálu waveshaperem.

Filtrace

Hned pod právě zmíněnými prvky se nachází skupina ovladačů spojená s kmitočtovou filtrací signálu. Jsou vždy zobrazeny parametry pro právě zvolený filtr. K dispozici je potenciometr pro úpravu mezního/středního kmitočtu, činitele jakosti, zesílení, tlačítko pro deaktivaci filtru a rozbalovací seznam s výběrem jeho strmosti. Zesílení a strmost je možné měnit pouze u některých typů filtrů. Vedle

(a) (b) (c)

Obr. 3.16: Úprava barevného odstínu uživatelského prostředí.

výběru filtru je umístěno tlačítko pro přepnutí pozice kmitočtové filtrace v rámci blokového schématu pluginu (Post) a tlačítko pro aktivaci filtru stejnosměrné složky.

Zkreslení

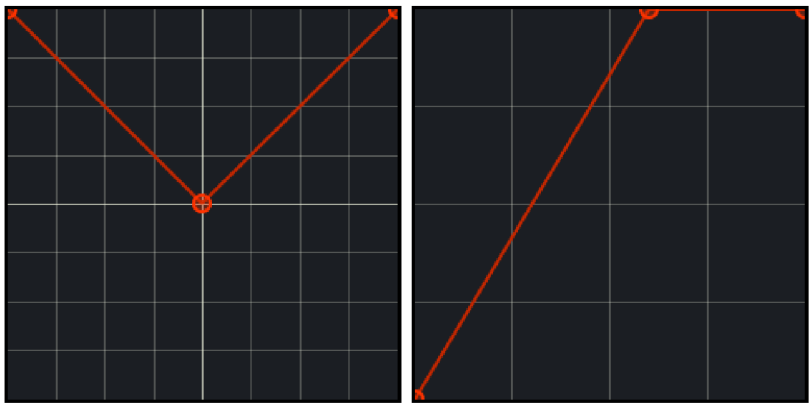
Sekce zaměřená na zkreslení signálu je umístěna ve spodní částí okna. Největší plochu zabírá pole určené k návrhu převodní charakteristiky. Kliknutím do prázdného místa tohoto pole dojde k vytvoření nového bodu, kliknutím a následným držením tlačítka myši v blízkosti nějakého bodu dochází v kombinaci s pohybem myši k jeho přesunu. K odstranění nějakého bodu dojde po dvojitém kliknutí v jeho bezprostřední blízkosti. Napravo od editoru přenosové funkce jsou zbylé ovládací prvky, konkrétně jde o rozbalovací seznam určený k výběru typu proložení bodů, tlačítko k resetování tvaru převodní charakteristiky, tlačítko k přepnutí mezi symetrickým a nesymetrickým zkreslením a úplně dole je rozbalovací seznam s výběrem stupně převzorkování.

3.2.4 Zvukové ukázky

Stejně jako u protypu z Matlabu (viz kapitola 3.1.3) bylo provedeno několik demonstrací použití waveshaperu realizovaného pomocí JUCE. Pro porovnání obou z nich byly u některých ukázek použity stejné parametry.

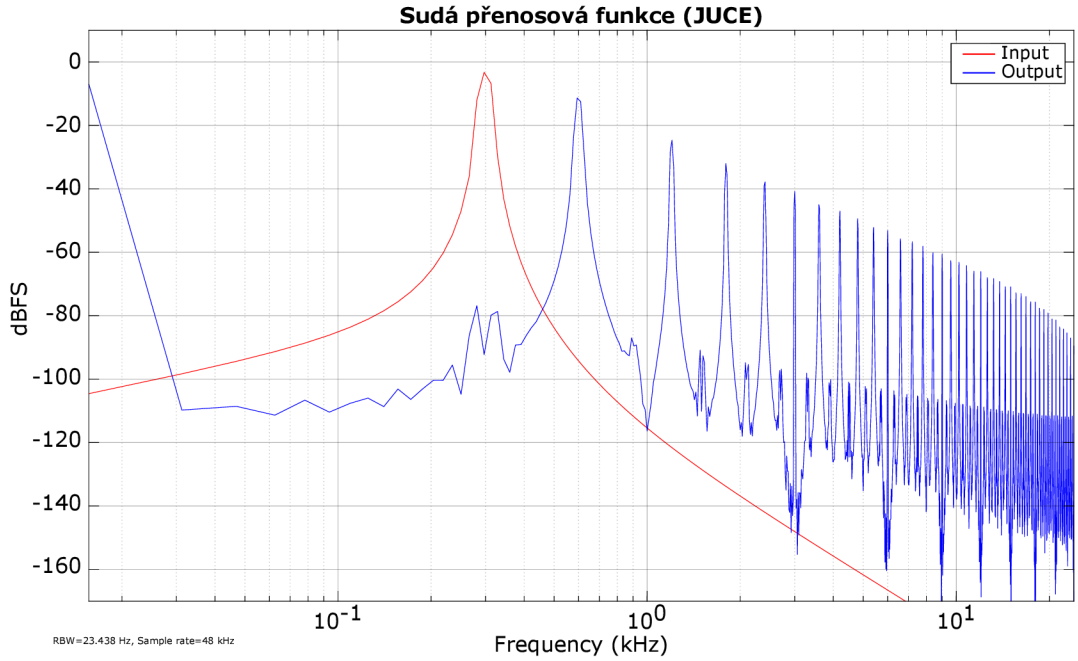
Ukázky zkreslení

Demonstrace byla provedena při použití převážně stejných parametrů, jako tomu bylo v případě prototypu zhotoveného v rámci semestrální práce. Jejím účelem je převážně ověření správnosti implementace zkreslení signálu v novém programovacím jazyce. Jako první zde je uvedena ukázka zkreslení pomocí sudé a liché přenosové funkce s **lineárním proložením bodů**, viz obr. 3.17. Opět byl použit signál sinusového průběhu s kmitočtem 300 Hz. Kmitočtová spektra výstupního signálu jsou vidět na obr. 3.18 a 3.19.

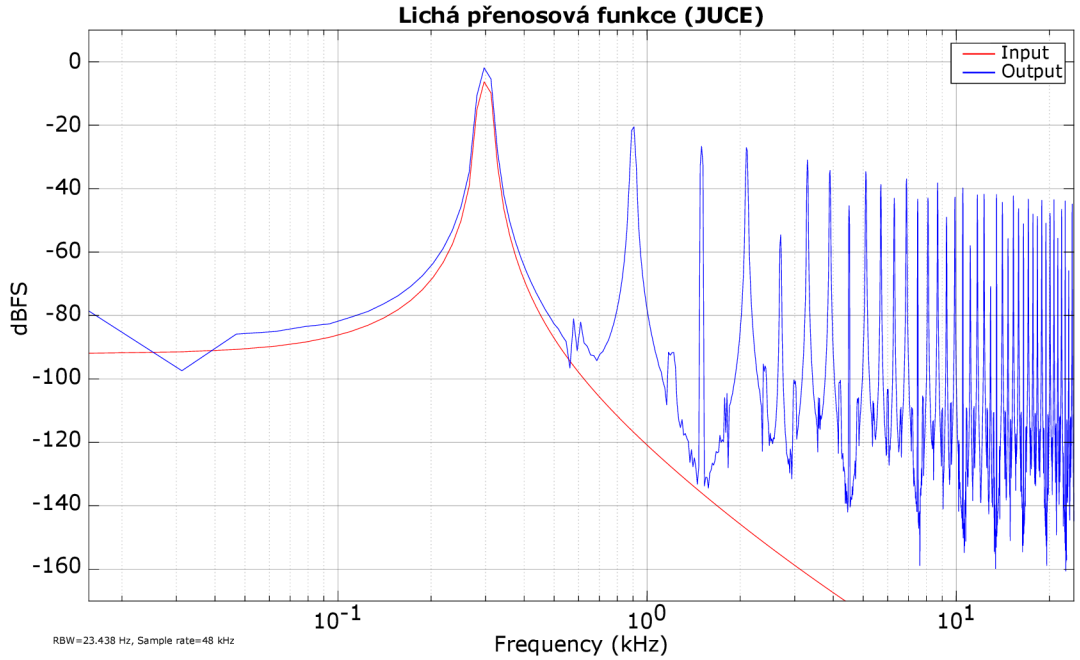


(a) Sudá přenosová funkce (b) Lichá přenosová funkce

Obr. 3.17: Převodní charakteristiky pro první demonstraci zkreslení.



Obr. 3.18: Kmitočtové spektrum výstupního signálu po zkreslení (sudá funkce).

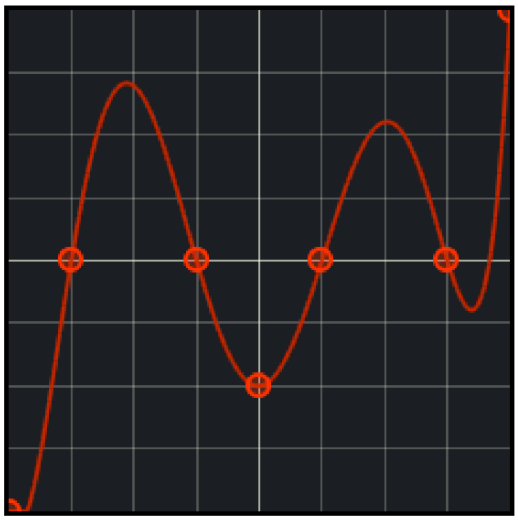


Obr. 3.19: Kmitočtové spektrum výstupního signálu po zkreslení (lichá funkce).

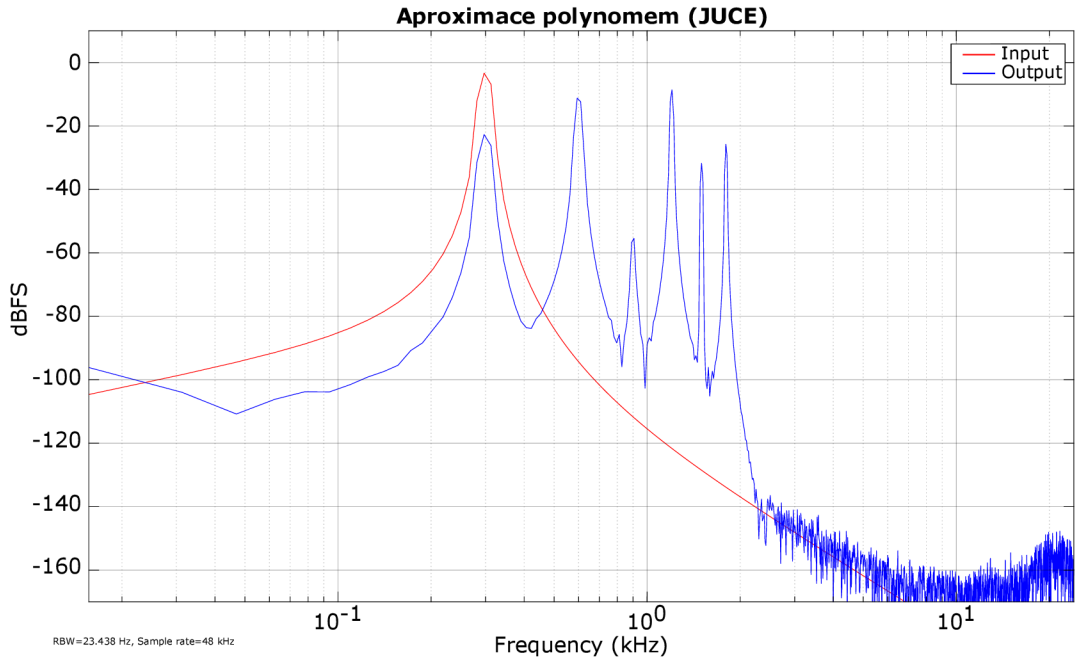
Na nich si můžeme všimnout, že v případě sudé funkce jsou na výstupu pouze sudé vyšší harmonické složky, u liché přenosové funkce pak pouze liché. Byl tedy opět potvrzen výrok z kapitoly 2.1.2. Zvukové soubory této ukázky jsou uvedeny v elektronické příloze pod názvy `sine_300Hz.wav`, `sine_300Hz_suda_funkce_juce.wav` a `sine_300Hz_licha_funkce_juce.wav`.

Pro druhou ukázku zkreslení byla použita převodní charakteristika s **body proloženými polynomem**, viz obr. 3.20. Jako vstupní signál byl opět použit sinusový průběh s kmitočtem 300 Hz. Při této ukázce byl aktivován filtr stejnosměrné složky – demonstrace jeho funkčnosti je uvedena v rámci ukázek filtrace signálu. Na kmitočtovém spektru výstupního signálu (viz obr. 3.21) je opět vidět souvislost mezi řádem použitého polynomu a počtem harmonických složek na výstupu – zvolená přenosová funkce je definována pomocí sedmi bodů, došlo proto k jejich proložení polynomem šestého řádu a na kmitočtovém spektru se objevilo šest harmonických složek. Tato ukázka je ve zvukové podobě dostupná pod názvem `sine_300Hz_polynom_juce.wav`.

Vstupním signálem pro poslední ukázku zkreslení byla nahrávka bicích. Opět byla použita lichá přenosová funkce z prvního příkladu (viz obr. 3.17), aktivován nebyl žádný filtr a byl nastaven osmý stupeň převzorkování. Zvukové soubory vstupního a zkresleného signálu jsou uloženy v elektronické příloze pod názvy: `bici.wav` a `bici_zkreslene_juce.wav`.



Obr. 3.20: Převodní charakteristika s body proloženými polynomem.

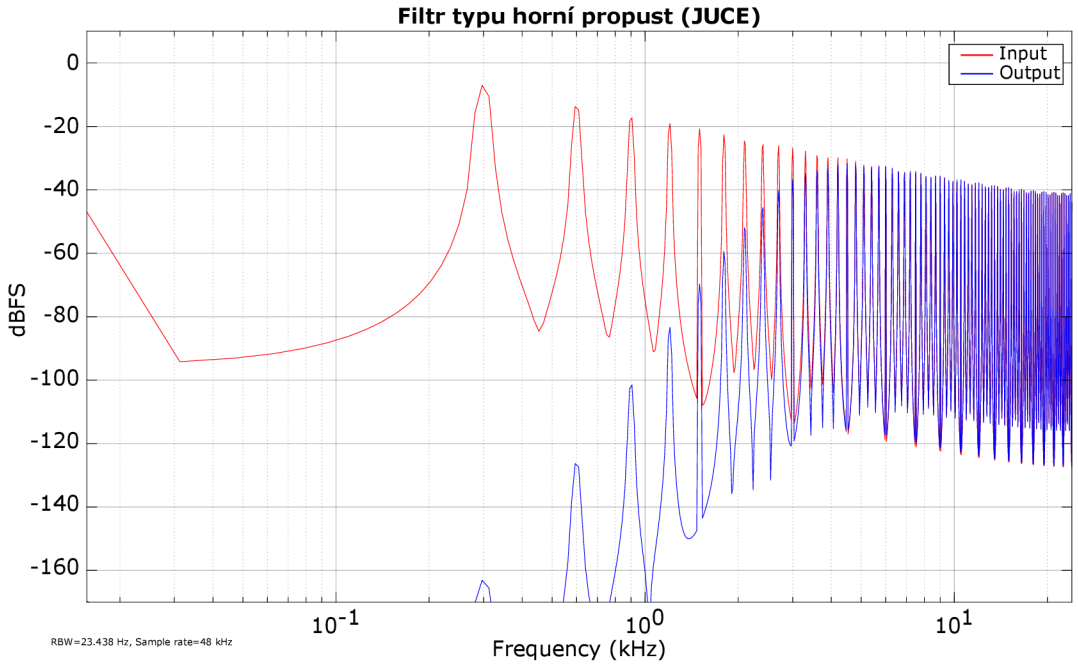


Obr. 3.21: Kmitočtové spektrum výstupního signálu po zkreslení (aproximace polynomem).

Ukázky filtrace

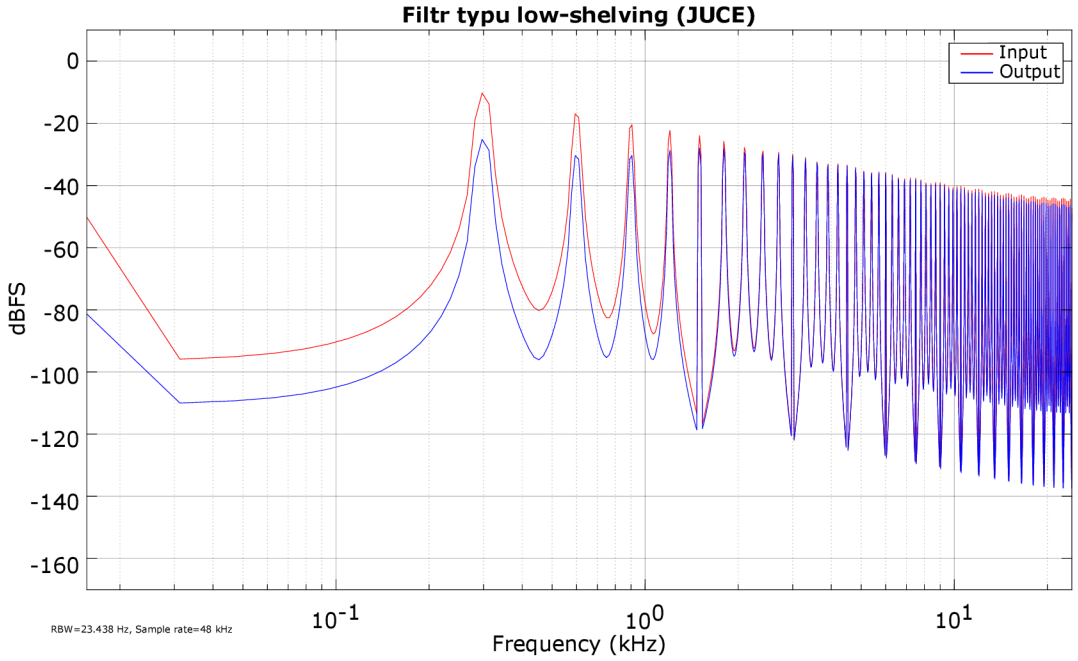
Následující ukázky jsou zaměřené na kmitočtovou filtraci. Vstupním signálem pro každou z nich byl pilový průběh s kmitočtem 300 Hz. Parametry použité pro demonstraci jednotlivých typů filtrů jsou napsány nad každým kmitočtovým spektrem uvedeným v následující části. Zvukové soubory k příslušným ukázkám jsou uvedeny v elektronické příloze, jejich názvy viz C.2.

- **Horní propust** (obr. 3.22) – $f_C = 3\,\text{kHz}$, $Q = 3$, strmost byla 48 dB/okt.



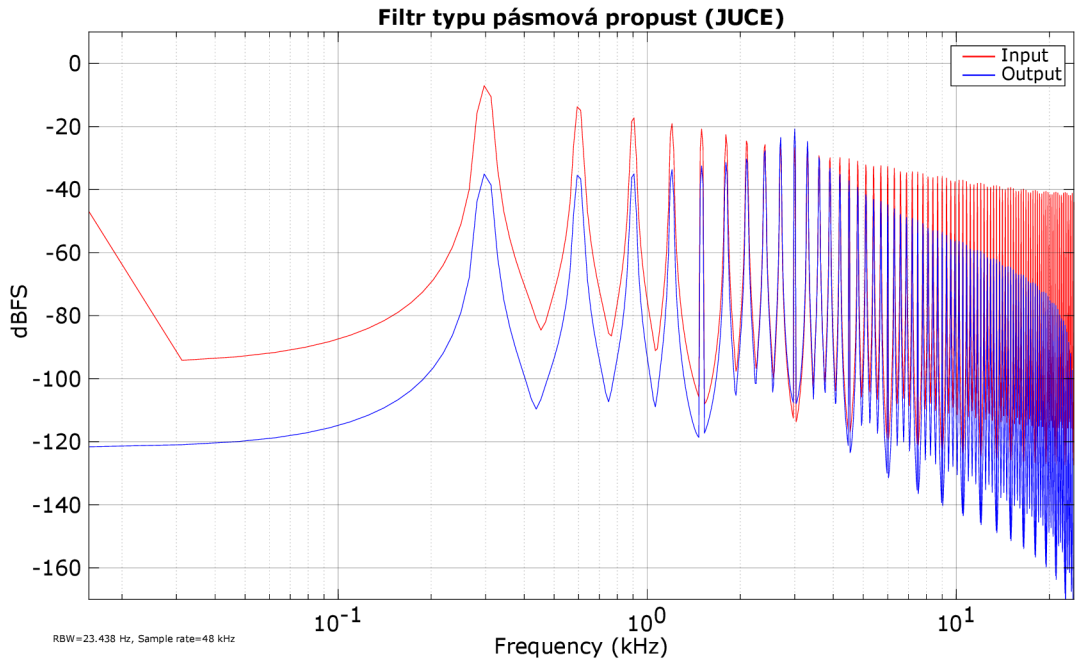
Obr. 3.22: Kmitočtové spektrum výstupního signálu po filtraci (horní propust).

- **Low-shelving** (obr. 3.23) – $f_C = 700\,\text{Hz}$, zesílení = -15 dB.



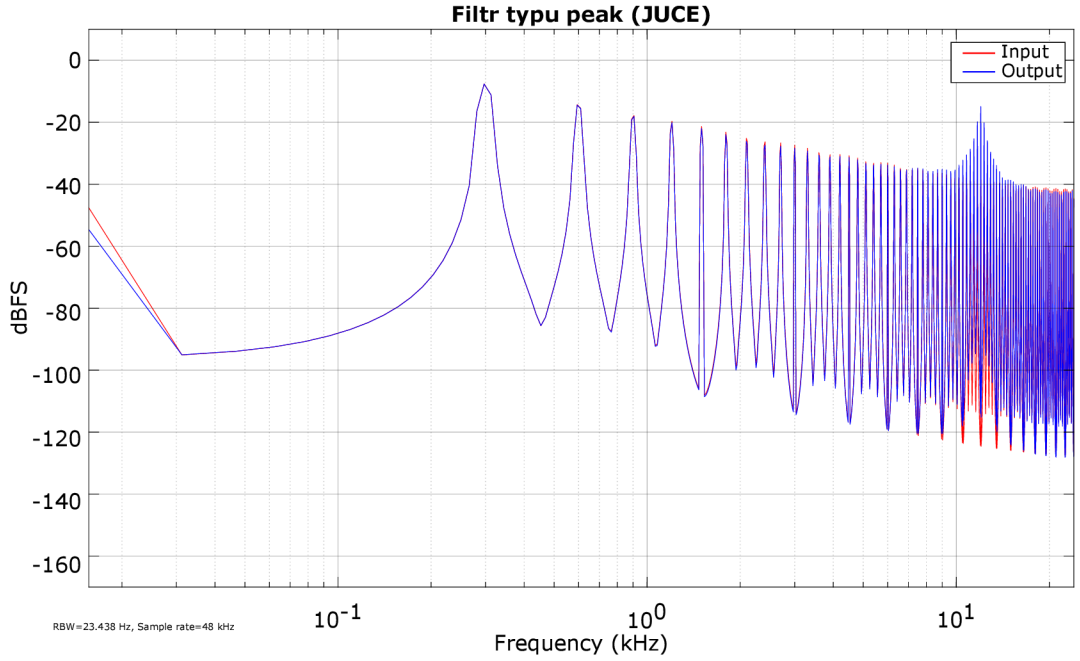
Obr. 3.23: Kmitočtové spektrum výstupního signálu po filtraci (low-shelving).

- **Pásmová propust** (obr. 3.24) – $f_C = 3\,\text{kHz}$, $Q = 5$, výst. zesílení $= 6\,\text{dB}$.



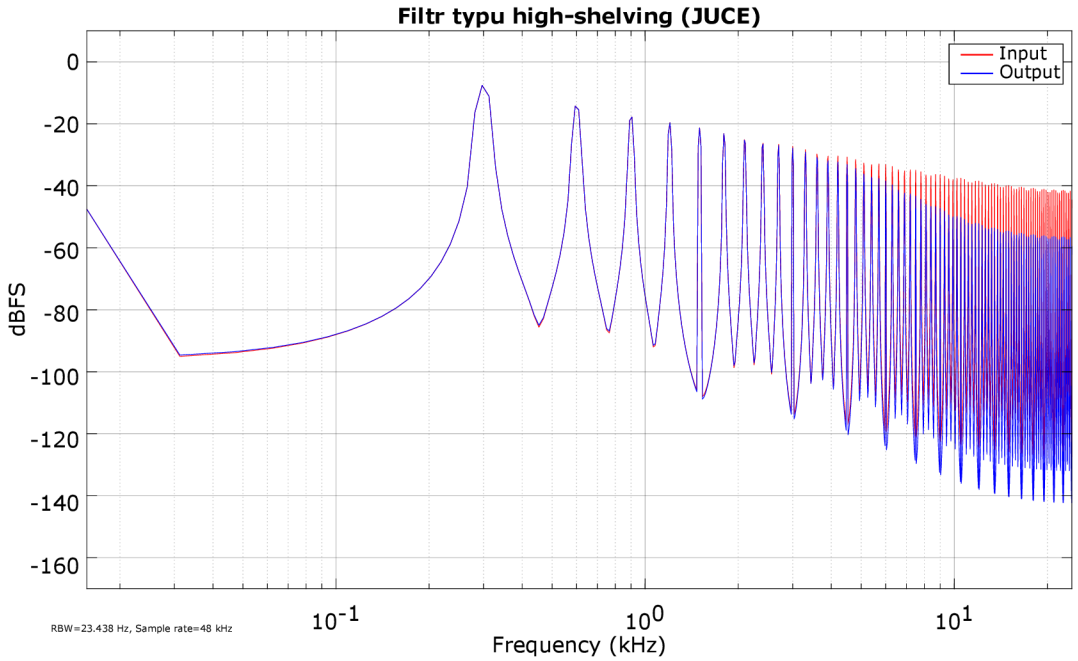
Obr. 3.24: Kmitočtové spektrum výstupního signálu po filtraci (pásmová propust).

- **Peak** (obr. 3.25) – $f_C = 12$ kHz, $Q = 18$, zesílení $= 24\,\text{dB}$.



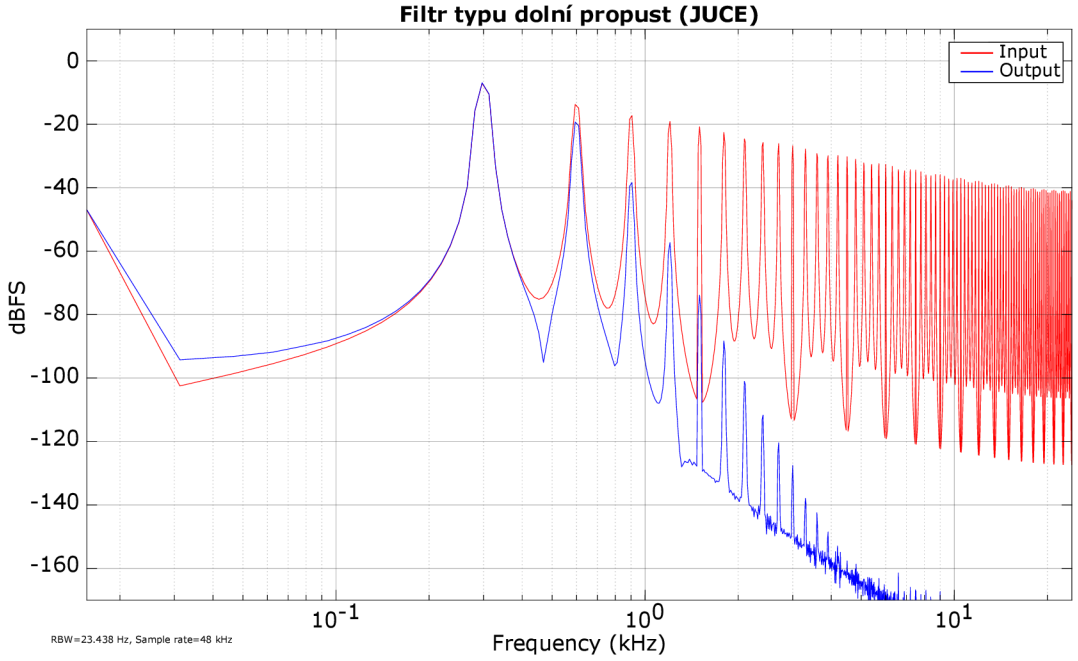
Obr. 3.25: Kmitočtové spektrum výstupního signálu po filtraci (peak).

- **High-shelving** (obr. 3.26) – $f_C = 10\,\text{kHz}$, zesílení = -15 dB.



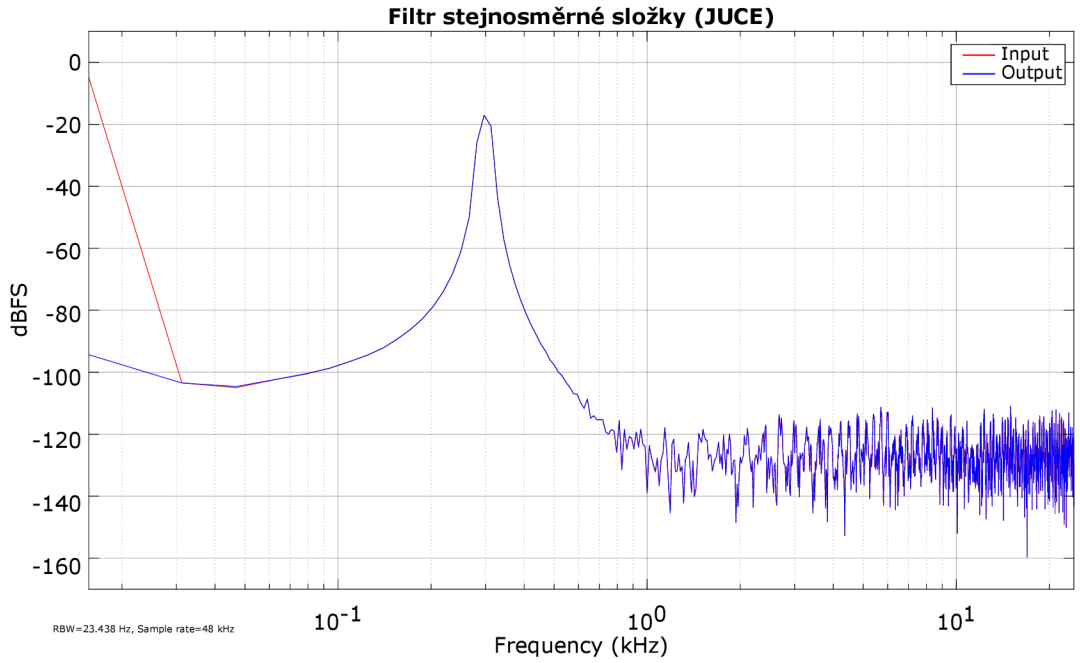
Obr. 3.26: Kmitočtové spektrum výstupního signálu po filtraci (high-shelving).

- **Dolní propust** (obr. 3.27) – $f_C = 700\,\text{Hz}$, $Q = 3$, strmost byla 48 dB/okt.



Obr. 3.27: Kmitočtové spektrum výstupního signálu po filtraci (dolní propust).

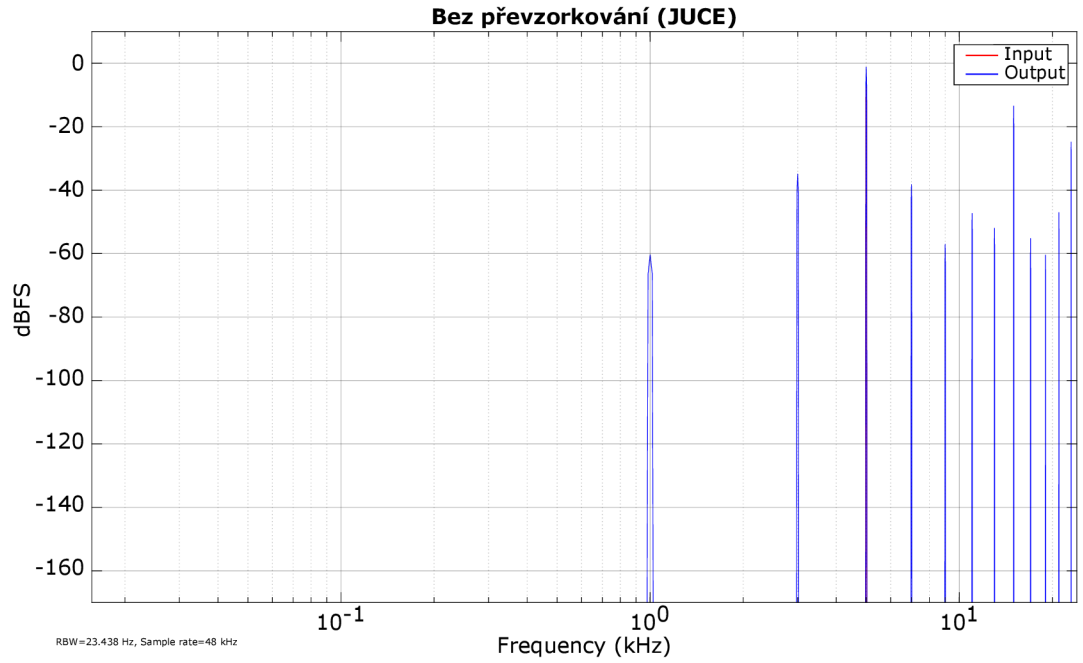
- **Filtr stejnosměrné složky** (obr. 3.28) – parametry viz kapitola 2.2.8.



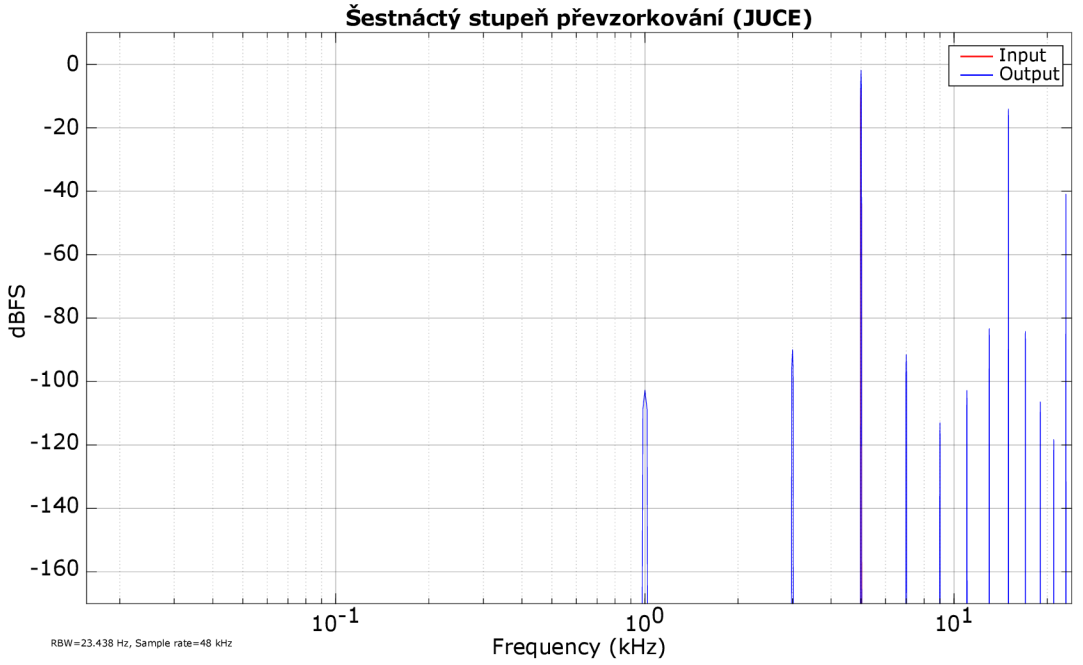
Obr. 3.28: Kmitočtové spektrum výstupního signálu po použití DC filtru.

Ukázka převzorkování

Poslední ukázkou je demonstrace funkčnosti převzorkování. Opět byl použit vstupní signál se sinusovým průběhem o kmitočtu 5 kHz, který byl tvrdě ořezán zvýšením vstupního zesílení o 10 dB. Na obr. 3.29 pak je vidět porovnání výsledných kmitočtových spekter pro zkreslený výstupní signál bez použití převzorkování a s použitím šestnáctého stupně převzorkování, který byl nově implementován do JUCE verze efektu. Při porovnání obr. 3.29(b) s obr. 3.12 si můžeme všimnout, že díky vyššímu stupni převzorkování došlo ještě k většímu potlačení aliasingových složek, než tomu bylo v případě prototypu z Matlabu. Zvukové soubory této demonstrace jsou umístěny v elektronické příloze.



(a) Žádné převzorkování



(b) Šestnáctý stupeň převzorkování

Obr. 3.29: Porovnání kmitočtových spekter výstupního signálu bez použití převzorkování a s použitím šestnáctého stupně převzorkování.

3.3 Porovnání obou realizací waveshaperu

V této části jsou rozebrány klady a zápory obou realizací.

Matlab (Audio Toolbox)

- Zdrojový kód není příliš rozsáhlý.
- Základní vzhled uživatelského prostředí, který lze jen minimálně měnit.
- Offline editace přenosové funkce.
- Není možné měnit pozici bodů jejich tažením.
- Přepnutím mezi symetrickým/nesymetrickým zkreslením dojde k vymazání navržené přenosové funkce.
- Pouze jeden typ filtru současně.
- Dostupné pouze tři typy filtrů.
- Neobsahuje DC filtr.
- Filtrace vždy před zkreslením.
- Převod do formátu VST není možný.

C++ (JUCE)

- Rozsáhlý zdrojový kód.
- Vykreslení ovládacích prvků je možné od základu přepsat.
- Editace přenosové funkce je prováděna v reálném čase.
- Možnost přesouvání bodů tažením myši.
- Přepnutí mezi symetrickým/nesymetrickým zkreslením nezpůsobí vymazání navržené přenosové funkce.
- Všechny obsažené filtry mohou být použity současně.
- K dispozici je celkově sedm typů filtrů (včetně DC filtru).
- Možnost umístěním kmitočtové filtrace před i po zkreslení.
- Možnost kompilace do různých formátů (např. VST3, AU, AAX).

Závěr

Tato práce se zabývala realizací nelineárního efektu typu waveshaper. Jejím cílem byla implementace zkreslení s návrhem vlastní převodní charakteristiky, kmitočtová filtrace a převzorkování signálu. Za tímto účelem měl být původně použit pouze výpočetní software Matlab s rozšířením Audio Toolbox. První prototyp obsahoval všechny požadované funkce, kvůli různým omezením použitého prostředí však práce s ním nebyla dostatečně plynulá. Z tohoto důvodu bylo následně rozhodnuto o jeho celkovém přepracování v programovacím jazyce C++ s využitím frameworku JUCE. Tato implementace obsahuje oproti té původní řadu vylepšení, které značně ulehčují práci s efektem. Hlavní výhodou pak je možnost kompilace do různých formátů používaných různými hostitelskými aplikacemi (v příloze práce je efekt uložen ve formátu VST3).

Textová část práce se nejprve krátce věnovala teorii spojené s tímto druhem zvukového efektu. V kapitolách o praktické části pak byly přiblíženy způsoby a nástroje použité k implementaci zadaného efektu v obou prostředích. V případě Matlabu text také obsahuje části zdrojového kódu v podobě výpisů. Při vytváření struktury tohoto kódu byla projevena především snaha o jeho přehlednost. Vznikla proto jedna hlavní třída, která strukturou odpovídá požadavkům rozšíření Audio Toolbox a obsahuje pouze nezbytné části. Algoritmy pro zpracování signálu a editaci převodní charakteristiky jsou zapsány do čtyř samostatných funkcí, které jsou z hlavní třídy pouze volány. V části o prototypu realizovaném v jazyce C++ je implementace už popsána pouze pomocí textu a schémat, další komentáře jsou uvedeny přímo ve zdrojovém kódu. I v tomto případě bylo cílem udržení co nejlepší čitelnosti, vzhledem k velkému počtu metod obsažených již v šabloně frameworku JUCE to však bylo možné pouze do určité míry.

V závěru práce byla pomocí řady ukázek provedena demonstrace funkčnosti jednotlivých způsobů zpracování signálu obsažených v realizovaném efektu. Pro každou z nich je uvedeno použité nastavení jednotlivých parametrů, lze je tedy zpětně rekonstruovat. Grafické ukázky zahrnují tvary použitých převodních charakteristik a porovnání kmitočtových spekter signálů na vstupu a na výstupu efektu. Všechny ukázky uvedené v této části jsou obsahem elektronické přílohy. Ta dále obsahuje zvukové soubory k těmto ukázkám a zdrojové kódy pro obě provedené implementace zvukového efektu.

Literatura

[1] SMÉKAL, Z. *Analýza signálu a soustav – BASS* [online]. 1. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav komunikací, 2012 [cit. 2021-12-02]. ISBN 978-80-214-4453-9. Dostupné z URL: < `https://moodle-archiv-2019-2020.ro.vutbr.cz/mod/resource/view.php?id=139758`> [Přístupné pouze pro studenty BPC-ASI].

[2] SCHIMMEL, J.: *Studiová a hudební elektronika*. [online]. Druhé. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015 [cit. 2021-12-02]. ISBN 978-80-214-4452-2. Dostupné z URL: <`https://www.vut.cz/www_base/priloha_fs.php?dpid=61387&skupina=dokument_priloha`> [Dostupné pouze po přihlášení].

[3] MIŠUREC, J., SMÉKAL, Z.: *Číslicové zpracování signálů*. Skriptum. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. 2011.

[4] ZÖLZER, U.: *DAFX: digital audio effects*. 2nd ed. Chichester: John Wiley & Sons, 2011. ISBN 978-0-470-66599-2.

[5] REISS, J., D., MCPHERSON, A., P.: *Audio effects: theory, implementation and application*. Boca Raton: CRC Press, 2014. ISBN 978-1-4665-6028-4.

[6] *Dokumentace Audio Toolboxu*. [online]. [cit. 2021-12-07]. Dostupné z URL: <`https://www.mathworks.com/help/audio/ref/audiopluginparameter.html`.

[7] *Dokumentace k Matlab funkci polyfit*. [online]. [cit. 2021-12-12]. Dostupné z URL: <`https://www.mathworks.com/help/matlab/ref/polyfit.html`.

[8] VRBA, K. *Analogová technika* [online]. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2020 [cit. 2021-12-02]. Dostupné z URL: <`https://moodle-archiv-2019-2020.ro.vutbr.cz/mod/folder/view.php?id=190033`> [Dostupné pouze pro studenty BPC-ANA].

[9] *Dokumentace k Matlab funkci interp*. [online]. [cit. 2021-12-11]. Dostupné z URL: <`https://www.mathworks.com/help/signal/ref/interp.html`.

[10] *Dokumentace k Matlab funkci decimate*. [online]. [cit. 2021-12-11]. Dostupné z URL: <`https://www.mathworks.com/help/signal/ref/decimate.html`.

[11] *JUCE*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001, 22.4.2022 [cit. 2022-05-15]. Dostupné z URL: <https://en.wikipedia.org/wiki/JUCE

[12] *Get JUCE*. [online]. [cit. 2022-05-15]. Dostupné z URL: <https://juce.com/get-juce.

[13] *Made with JUCE*. [online]. [cit. 2022-05-15]. Dostupné z URL: <https://juce.com/discover/made-with-juce.

[14] *Dokumentace k JUCE třídě AudioProcessorValueTreeState*. [online]. [cit. 2022-05-17]. Dostupné z URL: <https://docs.juce.com/master/classAudioProcessorValueTreeState.html

[15] *Dokumentace k JUCE třídě Time*. [online]. [cit. 2022-05-17]. Dostupné z URL: <https://docs.juce.com/master/classTimer.html

[16] FERGUSON, Sam. *JuceBezierGraph*. In: GitHub [online]. [cit. 2022-05-22]. Dostupné z URL: <https://github.com/DrCheph/JuceBezierGraph

[17] *Dokumentace k JUCE metodě jmap*. [online]. [cit. 2022-05-16]. Dostupné z URL: <https://docs.juce.com/master/group__juce__core-maths.html#ga8acdd3d518517bd5e3c0bd1922218bf9

[18] *Dokumentace k JUCE třídě Point*. [online]. [cit. 2022-05-16]. Dostupné z URL: <https://docs.juce.com/master/classPoint.html

[19] *Polynomial regression*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001, 6.7.2021 [cit. 2022-05-24]. Dostupné z URL: <https://en.wikipedia.org/wiki/Polynomial_regression

[20] *Dokumentace k JUCE třídě Matrix*. [online]. [cit. 2022-05-18]. Dostupné z URL: <https://docs.juce.com/master/classdsp_1_1Matrix.html

[21] *Dokumentace k JUCE třídě Polynomial*. [online]. [cit. 2022-05-18]. Dostupné z URL: <https://docs.juce.com/master/classdsp_1_1Polynomial.html

[22] REDMON, Nigel. *Cascading filters*. In: EarLevel Engineering [online]. [cit. 2022-05-19]. Dostupné z URL: <http://www.earlevel.com/main/2016/09/29/cascading-filters/

[23] *Dokumentace k JUCE třídě dsp::IIR:Filter*. [online]. [cit. 2022-05-18]. Dostupné z URL: <https://docs.juce.com/master/classdsp_1_1IIR_1_1Filter.html

[24] SCHIERMEYER, Charles. *Learn Modern C++ by Building an Audio Plugin (w/ JUCE Framework) - Full Course*. In: YouTube [online]. 20.5.2021 [cit. 2022-05-18]. Dostupné z URL: <https://www.youtube.com/watch?v=i_Iq4_Kd7Rc

[25] *Dokumentace k JUCE struktuře dsp::IIR::Coefficients*. [online]. [cit. 2022-05-18]. Dostupné z URL: <https://docs.juce.com/master/structdsp_1_1IIR_1_1Coefficients.html

[26] *Dokumentace k JUCE třídě Oversampling*. [online]. [cit. 2022-05-19]. Dostupné z URL: <https://docs.juce.com/master/classdsp_1_1Oversampling.html

[27] *Dokumentace k JUCE metodě LookAndFeel_V4*. [online]. [cit. 2022-05-20]. Dostupné z URL: <https://docs.juce.com/master/classLookAndFeel__V4.html

[28] SUZUKI, Kengo. *Dial Customization*. In: KENGO [online]. [cit. 2022-05-20]. Dostupné z URL: <https://suzuki-kengo.dev/posts/dial-customization

[29] SCHIERMEYER, Charles. *AudioFilePlayer*. In: GitHub [online]. [cit. 2022-05-22]. Dostupné z URL: <https://github.com/matkatmusic/AudioFilePlayer

Seznam symbolů a zkratek

AAX	Avid Audio eXtension
AU	Audio Units
DAW	Digital Audio Workstation
GUI	Graphical user interface
IIR	Infinite Impulse Response
THD	Total Harmonic Distortion
VST	Virtual Studio Technology

Seznam příloh

A Implementace v Matlabu

Pro implementaci a následné testování této práce byl použit Matlab verze R2020b Update 4 společně s Audio Toolboxem ve verzi 2.3. Grafické i zvukové ukázky byly provedeny pomocí nástroje Audio Test Bench. Struktura zdrojového kódu této implementace je následující:

`draw.m`

Funkce, ve které je implementována editace převodní charakteristiky.

`distort.m`

Funkce obsahující implementaci zkreslení.

`filters.m`

Funkce obsahující implementaci kmitočtových filtrů.

`oversample.m`

Funkce obsahující implementaci převzorkování.

`Waveshaper.m`

Hlavní třída, přes kterou je efekt spouštěn (viz 3.1.1).

B Implementace v C++

Druhá verze efektu byla realizována s použitím vývojového prostředí Microsoft Visual Studio Community 2019 v kombinaci s aplikačním frameworkem JUCE v6.1.6. Pro implementaci aproximace polynomem byla dále použita externí matematická knihovna Eigen ve verzi 3.4.0. Pro testování byl použit Juce Plug-In Host společně s přehrávačem AudioFilePlayer [29]. K vytvoření grafických i zvukových ukázek byl opět využit nástroj Audio Test Bench z Matlabu. Zdrojový kód se skládá z následujících souborů:

`eigen`
Složka obsahující soubory matematické knihovny Eigen.

`ColorPalette`
Obsahuje definice hlavních barev používaných v uživatelském prostředí.

`ComboBoxLNF`
Třída obsahující úpravu vzhledu rozbalovacího seznamu.

`CustomLNF`
Třída sloužící k nastavení vlastního vzhledu některých ovládacích prvků.

`NameLabel`
Nastavení vzhledu textu s názvy ovládacích prvků.

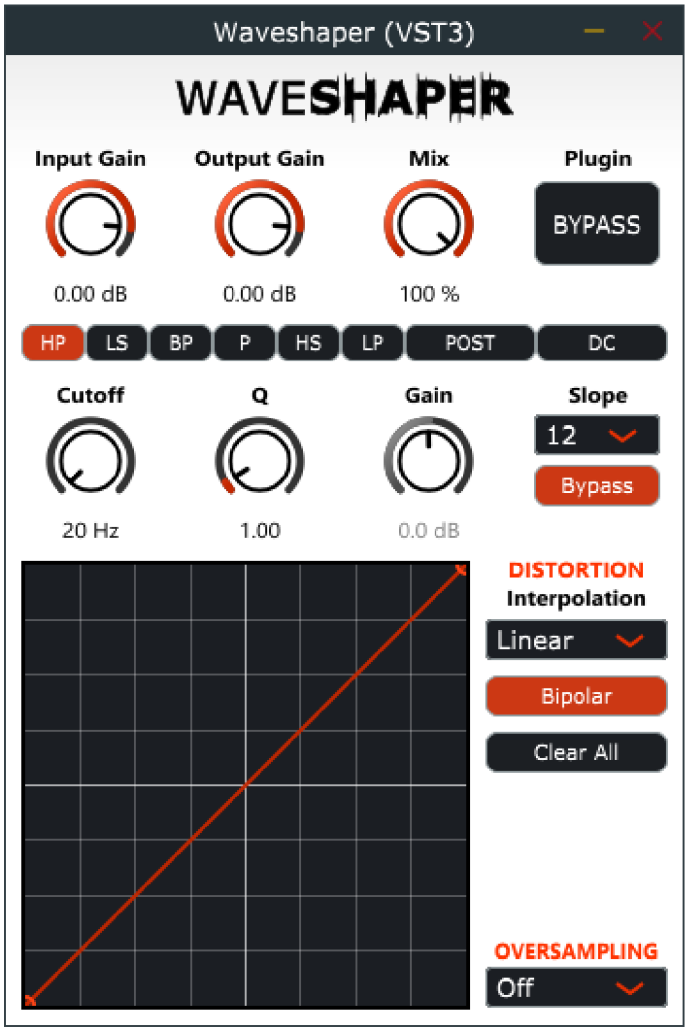
`PluginEditor`
Jedna ze dvou hlavních tříd, která byla vygenerována Projucerem. Je zaměřena na okno s uživatelským prostředím efektu.

`PluginProcessor`
Druhá hlavní třída vygenerovaná pomocí Projuceru. Tato třída je zaměřena na zpracování zvukového signálu.

`ShaperWindow`
Třída s implementací pole určeného k editaci převodní charakteristiky.

`SliderLNF`
Třída obsahující úpravu vzhledu otočného potenciometru.

`TextButtonLNF`
Třída obsahující úpravu vzhledu tlačítka.



Obr. B.1: Světlá verze uživatelského prostředí waveshaperu (JUCE).

C Seznam zvukových ukázek

Společné zvukové soubory pro obě realizace:

`300Hz_sawtooth.wav`

Signál s pilovým průběhem a kmitočtem 300 Hz.

`bici.wav`

Nahrávka bicí použitá ke zkreslení, pochází z Audio Test Benche.

`sine_300Hz.wav`

Signál se sinusovým průběhem a kmitočtem 300 Hz.

C.1 Matlab

`300Hz_sawtooth_3kHz_bandpass_matlab.wav`

Filtrace pilového průběhu s kmitočtem 300 Hz pásmovou propustí s mezním kmitočtem rovným 3 kHz, činitelem jakosti rovným 5 a výstupním zesílením +6 dB.

`300Hz_sawtooth_3kHz_highpass_matlab.wav`

Filtrace pilového průběhu s kmitočtem 300 Hz horní propustí s mezním kmitočtem rovným 3 kHz a činitelem jakosti rovnému 1.

`300Hz_sawtooth_700Hz_lowpass_matlab.wav`

Filtrace pilového průběhu s kmitočtem 300 Hz dolní propustí s mezním kmitočtem rovným 700 Hz a činitelem jakosti rovnému 1.

`bici_zkreslene_matlab.wav`

Nahrávka bicích zkreslena pomocí liché přenosové funkce (viz obr. 3.4) a osmého stupně převzorkování.

`sine_5kHz_bez_prevzorkovani_matlab.wav`

Tvrdě ořezaný signál sinusového průběhu s kmitočtem 5 kHz bez použití převzorkování.

`sine_5kHz_prevzorkovani_8x_matlab.wav`

Tvrdě ořezaný signál sinusového průběhu s kmitočtem 5 kHz s použitím osmého stupně převzorkování.

`sine_300Hz_licha_funkce_matlab.wav`

Zkreslení signálu pomocí liché přenosové funkce (viz obr. 3.4).

`sine_300Hz_polynom_matlab.wav`

Zkreslení s body přenosové funkce proloženými polynomem (viz obr. 3.6).

`sine_300Hz_suda_funkce_matlab.wav`

Zkreslení signálu pomocí sudé přenosové funkce (viz obr. 3.4).

C.2 JUCE

300Hz_sawtooth_3kHz_bandpass_juce.wav

Filtrace pilového průběhu s kmitočtem 300 Hz pásmovou propustí s mezním kmitočtem rovným 3 kHz, činitelem jakosti rovným 5 a výstupním zesílením +6 dB.

300Hz_sawtooth_3kHz_highpass_slope_48_juce.wav

Filtrace pilového průběhu s kmitočtem 300 Hz horní propustí s mezním kmitočtem rovným 3 kHz, činitelem jakosti rovným 3 a strmostí 48 dB/okt.

300Hz_sawtooth_10kHz_highshelving_juce.wav

Filtrace pilového průběhu s kmitočtem 300 Hz filtrem typu high-shelving s mezním kmitočtem rovným 10 kHz a zesílením -15 dB.

300Hz_sawtooth_12kHz_peak_juce.wav

Filtrace pilového průběhu s kmitočtem 300 Hz filtrem typu peak s mezním kmitočtem rovným 12 kHz, činitelem jakosti rovným 18 a zesílením 24 dB.

300Hz_sawtooth_700Hz_lowpass_slope_48_juce.wav

Filtrace pilového průběhu s kmitočtem 300 Hz dolní propustí s mezním kmitočtem rovným 700 Hz, činitelem jakosti rovným 3 a strmostí 48 dB/okt.

300Hz_sawtooth_700Hz_lowshelving_juce.wav

Filtrace pilového průběhu s kmitočtem 300 Hz filtrem typu low-shelving s mezním kmitočtem rovným 700 Hz a zesílením -15 dB.

300Hz_sine_DC_filtr_juce.wav

Odfiltrování stejnosměrné složky.

bici_zkreslene_juce.wav

Nahrávka bicích zkreslena pomocí liché přenosové funkce (viz obr. 3.17) a osmého stupně převzorkování.

sine_5kHz_bez_prevzorkovani_juce.wav

Tvrdě ořezaný signál sinusového průběhu s kmitočtem 5 kHz bez použití převzorkování.

sine_5kHz_prevzorkovani_16x_juce.wav

Tvrdě ořezaný signál sinusového průběhu s kmitočtem 5 kHz s použitím šestnáctého stupně převzorkování.

sine_300Hz_DC_slozka.wav

Nahrávka obsahující signál se sinusovým průběhem o kmitočtu 300 Hz a přidanou stejnosměrnou složku.

sine_300Hz_licha_funkce_juce.wav

Zkreslení signálu pomocí liché přenosové funkce (viz obr. 3.17).

sine_300Hz_polynom_juce.wav

Zkreslení s body přenosové funkce proloženými polynomem (viz obr. 3.20).

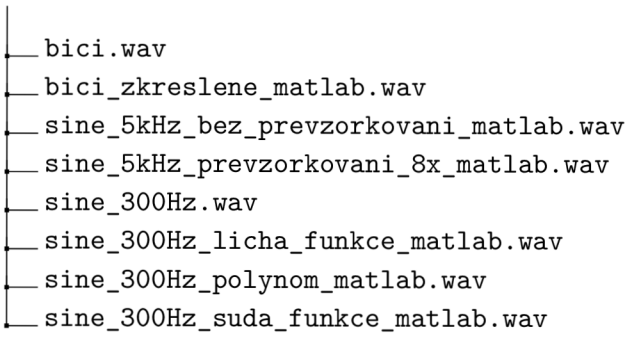
`sine_300Hz_suda_funkce_juce.wav`

Zkreslení signálu pomocí sudé přenosové funkce (viz obr. 3.17).

D Obsah elektronické přílohy

```
/
└── JUCE implementace
    ├── Grafické ukázky
    │   ├── obr_5kHz_sine_bez_prevzorkovani_juce.pdf
    │   ├── obr_5kHz_sine_prevzorkovani_16x_juce.pdf
    │   ├── obr_300Hz_sawtooth_3kHz_bandpass_juce.pdf
    │   ├── obr_300Hz_sawtooth_3kHz_highpass_slope_48_juce.pdf
    │   ├── obr_300Hz_sawtooth_10kHz_highshelving_juce.pdf
    │   ├── obr_300Hz_sawtooth_12kHz_peak_juce.pdf
    │   ├── obr_300Hz_sawtooth_700Hz_lowpass_slope_48_juce.pdf
    │   ├── obr_300Hz_sawtooth_700Hz_lowshelving_juce.pdf
    │   ├── obr_300Hz_sine_DC_filtr_juce.pdf
    │   ├── obr_uzivatelske_prostredi_rozdeleni_prvky.png
    │   ├── obr_uzivatelske_prostredi_rozdeleni_sekce.png
    │   ├── obr_uzivatelske_prostredi_waveshaperu_juce.png
    │   ├── obr_uzivatelske_prostredi_waveshaperu_juce_fialove.png
    │   ├── obr_uzivatelske_prostredi_waveshaperu_juce_modre.png
    │   ├── obr_uzivatelske_prostredi_waveshaperu_juce_svetle.png
    │   ├── obr_uzivatelske_prostredi_waveshaperu_juce_zelene.png
    │   ├── obr_zkresleni_licha_funkce_prenos_juce.png
    │   ├── obr_zkresleni_licha_funkce_spektrum_juce.pdf
    │   ├── obr_zkresleni_polynom_prenos_juce.png
    │   ├── obr_zkresleni_polynom_spektrum_juce.pdf
    │   ├── obr_zkresleni_suda_funkce_prenos_juce.png
    │   └── obr_zkresleni_suda_funkce_spektrum_juce.pdf
    └── Waveshaper ..... zdrojové soubory implementace v C++
        ├── JuceLibraryCode ..... použité JUCE moduly
        └── Source
            ├── eigen
            ├── ColorPalette.h
            ├── ComboBoxLNF.cpp
            ├── ComboBoxLNF.h
            ├── CustomLNF.cpp
            ├── CustomLNF.h
            ├── NameLabel.h
            ├── PluginEditor.cpp
            ├── PluginEditor.h
            ├── PluginProcessor.cpp
            ├── PluginProcessor.h
            ├── ShaperWindow.cpp
            ├── ShaperWindow.h
            ├── SliderLNF.cpp
            ├── SliderLNF.h
            └── TextButtonLNF.cpp
```

```
        ├── TextButtonLNF.h
        └── waveshaper_logo.svg
    ├── Waveshaper.jucer
    └── Waveshaper.vst3
├── Zvukové ukázky
    ├── 300Hz_sawtooth.wav
    ├── 300Hz_sawtooth_3kHz_bandpass_juce.wav
    ├── 300Hz_sawtooth_3kHz_highpass_slope_48_juce.wav
    ├── 300Hz_sawtooth_10kHz_highshelving_juce.wav
    ├── 300Hz_sawtooth_12kHz_peak_juce.wav
    ├── 300Hz_sawtooth_700Hz_lowpass_slope_48_juce.wav
    ├── 300Hz_sawtooth_700Hz_lowshelving_juce.wav
    ├── 300Hz_sine_DC_filtr_juce.wav
    ├── bici.wav
    ├── bici_zkreslene_juce.wav
    ├── sine_5kHz_bez_prevzorkovani_juce.wav
    ├── sine_5kHz_prevzorkovani_16x_juce.wav
    ├── sine_300Hz.wav
    ├── sine_300Hz_DC_slozka.wav
    ├── sine_300Hz_licha_funkce_juce.wav
    ├── sine_300Hz_polynom_juce.wav
    └── sine_300Hz_suda_funkce_juce.wav
└── Matlab implementace
    ├── Grafické ukázky
        ├── obr_5kHz_sine_bez_prevzorkovani_matlab.pdf
        ├── obr_5kHz_sine_prevzorkovani_8x_matlab.pdf
        ├── obr_300Hz_sawtooth_3kHz_bandpass_matlab.pdf
        ├── obr_300Hz_sawtooth_3kHz_highpass_matlab.pdf
        ├── obr_300Hz_sawtooth_700Hz_lowpass_matlab.pdf
        ├── obr_zkresleni_licha_funkce_prenos_matlab.pdf
        ├── obr_zkresleni_licha_funkce_spektrum_matlab.pdf
        ├── obr_zkresleni_polynom_prenos_matlab.pdf
        ├── obr_zkresleni_polynom_spektrum_matlab.pdf
        ├── obr_zkresleni_suda_funkce_prenos_matlab.pdf
        └── obr_zkresleni_suda_funkce_spektrum_matlab.pdf
    ├── Waveshaper ..... zdrojové soubory implementace v Matlabu
        ├── distort.m
        ├── draw.m
        ├── filters.m
        ├── oversample.m
        └── Waveshaper.m
    └── Zvukové ukázky
        ├── 300Hz_sawtooth.wav
        ├── 300Hz_sawtooth_3kHz_bandpass_matlab.wav
        ├── 300Hz_sawtooth_3kHz_highpass_matlab.wav
        └── 300Hz_sawtooth_700Hz_lowpass_matlab.wav
```


bici.wav
bici_zkreslene_matlab.wav
sine_5kHz_bez_prevzorkovani_matlab.wav
sine_5kHz_prevzorkovani_8x_matlab.wav
sine_300Hz.wav
sine_300Hz_licha_funkce_matlab.wav
sine_300Hz_polynom_matlab.wav
sine_300Hz_suda_funkce_matlab.wav