VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

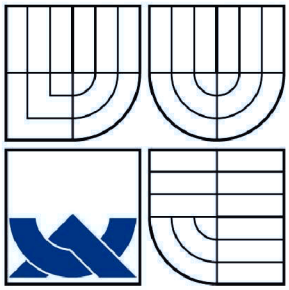
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE DIGITÁLNÍHO METRONOMU S PŘEVZORKOVÁNÍM SIGNÁLU V TECHNOLOGII VST

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

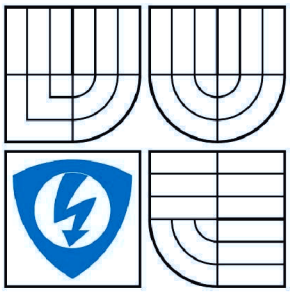
AUTOR PRÁCE — ALEŠ KŘUPKA
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

Implementace digitálního metronomu s převzorkováním signálu v technologii VST

Implementation of digital metronome with oversampling in VST technology

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

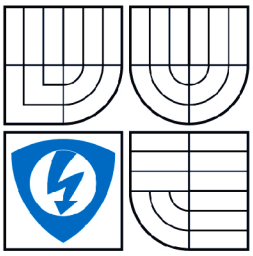
AUTOR PRÁCE
AUTHOR

Aleš Křupka

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Jaromír Mačák

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Aleš Křupka ***ID:*** 98175
Ročník: 3 ***Akademický rok:*** 2008/2009

NÁZEV TÉMATU:

Implementace digitálního metronomu s převzorkováním signálu v technologii VST

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte a pomocí technologie VST pro zpracování zvukových signálů v reálném čase implementujte digitální metronom s možností nezávislé volby zvuku pro první dobu a ostatní doby taktu. Metronom bude při přehrávání zvuků využívat převzorkování signálu, aby při změně vzorkovacího kmitočtu zůstala zachována výška zvuku. Informace o rytmu a tempu skladby bude zadávána ručně pomocí grafického uživatelského rozhraní nebo načítána z hostitelské aplikace pomocí VST událostí.

DOPORUČENÁ LITERATURA:

[1] PROAKIS, J. G., MANOLAKIS D. G. Digital Signal Processing - Principles, Algorithms and Applications. Third Edition. Prentice Hall, New Jersey 1996. ISBN 0-13-373762-4.
[2] FLIEGE, N. J. Multirate Digital Signal Processing (Multirate Systems, Filter Banks, Wavelets). John Willy & Sons, Chichester 1994. ISBN 0-471-93976-5.
[3] SCHIMMEL, J. Formáty zvukových souborů na PC. Elektrorevue [online], 2001, č. 7. Dosupný z http://www.elektrorevue.cz/clanky/01007/. ISSN 1213-1539.

Termín zadání: 9.2.2009 ***Termín odevzdání:*** 2.6.2009

Vedoucí práce: Ing. Jaromír Mačák

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

Abstrakt

Tato bakalářská práce je vypracováním požadavku na implementaci digitálního metronomu v technologii VST. V úvodu jsou rozebrány základní vlastnosti digitálního metronomu a na jejich základě je navržen algoritmus, jenž bude digitální metronom realizovat. Tento algoritmus je zapouzdřen do třídy C++ s patřičně definovaným rozhraním, která je tak vhodná pro využití v různých technologiích pro zpracování digitálního zvukového signálu. Takto zapouzdřený metronom je implementován v technologii VST ve formě plug-in modulu. Pro tyto účely je v práci stručné seznámení s vývojovou sadou VST, definicemi jejích základních metod a způsobem, jakým lze pomocí ní vytvořit vlastní grafické uživatelské rozhraní VST plug-in modulu. Pro realizaci funkce volby libovolného zvuku jeho načtením ze souboru typu *wav* je v práci popsán formát RIFF, jež specifikuje formát ukládání multimediálních dat do souborů na disk. Na základě znalosti tohoto formátu je funkce načítání z *wav* souboru implementována. Pro potřebu přizpůsobení vzorkovací frekvence načteného zvuku je v práci pojednáno o problematice převzorkování a následně tyto poznatky využity při implementaci funkce převzorkování.

Abstract

This bachelor's thesis presents solution to requirement of implementation of digital metronome in VST technology. In the beginning, there is analysis of basic characteristics of digital metronome, that helps to suggest algorithm realizing digital metronome. Algorithm is encapsulated into C++ class with appropriate interface, which is suitable for usage in different technologies of digital sound signal processing. This way encapsulated metronome is implemented in VST technology as plug-in. For this reason, thesis contains a brief introduction to VST software development kit, definitions of its basic methods and way how to do graphical user interface of VST plug-in. For realizing of function enabling choice of any sound loaded from *wav* file, there is a description of RIFF format, that specifies the format of saving multimedia data into drive. Using this knowledge, the function of loading from *wav* file is implemented. In order to adjust sampling frequency of loaded sound, thesis deals with problems of sample rate conversion and implements resampling function.

Klíčová slova

Metronom, *wav* soubor, převzorkování, implementace, grafické uřivatelské rozhraní, technologie VST, plug-in modul.

Key words

Metronome, *wav* file, resampling, implementation, graphical user interface, VST technology, plug-in

Bibliografická citace

KŘUPKA, A. *Implementace digitálního metronomu s převzorkováním signálu v technologii VST*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 36 s. Vedoucí bakalářské práce Ing. Jaromír Mačák.

Prohlášení

Prohlašuji, že svoji bakalářskou práci na téma Implementace digitálního metronomu s převzorkováním signálu v technologii VST jsem vypracoval samostatně pod vedením vedoucího semestrálního projektu 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 uvedeného semestrálního projektu dále prohlašuji, že v souvislosti s vytvořením tohoto projektu jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 16. 12. 2008 .....

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Jaromírovi Mačákovi za odborné vedení a pomoc poskytnutou při řešení mé bakalářské práce.

Obsah

Seznam obrázků

1 Úvod

Bakalářská práce na téma Implementace digitálního metronomu s převzorkováním signálu v technologii VST je zaměřena na realizaci digitálního metronomu ve formě plug-in modulu. Práce je rozdělena do dílčích kapitol, kde jsou jednotlivé problémy teoreticky rozebrány a poté je navržen postup jejich řešení.

Metronom je zařízení, jež pomocí periodicky opakujících se zvuků udává tempo. První kapitola tedy obsahuje jeho obecný popis a rozbor jeho základních vlastností z hlediska digitálního zvukového signálu. Je zde uveden základní algoritmus, jehož obsah je podstatou funkce digitálního metronomu, a dále také rozšířená forma základního algoritmu beroucí v potaz i nutnost rozlišení první a ostatních dob v taktu pomocí různých zvuků. Tento rozšířený algoritmus bude v metronomu použit.

Druhá kapitola se zabývá načítáním zvukových signálů ze souboru typu *wav*. Proto je zde nejdřív popsán formát, který soubory typu *wav* používají a poté je zde objasněna jejich struktura. Tyto poznatky jsou použity pro realizaci funkce metronomu, pomocí níž si uživatel může zvuk metronomu volit jeho načtením ze souboru.

Bude-li metronom zajišťovat funkci načítání libovolných zvuků ze souborů, je nutno zajistit přizpůsobení vzorkovací frekvence načteného zvuku vzorkovací frekvenci, se kterou metronom pracuje, pokud si tyto frekvence nejsou rovny. Z toho důvodu třetí kapitola popisuje problematiku převzorkování, vliv operace nadvzorkování a podvzorkování na spektrum diskrétního signálu a způsob úpravy spektra převzorkovávaného signálu pomocí číslicového filtru. Pro operaci převzorkování je zde navržen filtr typu IIR.

Ve čtvrté kapitole je pojednáváno o způsobu zapouzdření metronomu a definice jeho rozhraní. Účelem je, aby takto vytvořený metronom mohl být využitelný i v různých technologiích, nejen VST.

Pátá kapitola se v úvodu obecně zabývá standardem VST. Dále uvádí, jakým způsobem lze vytvořit plug-in modul v technologii VST a obsahuje seznámení s vývojovým prostředím VST SDK, tj. vysvětlení jeho nejdůležitějších tříd a popis funkce nejdůležitějších metod, jež je třeba definovat pro správnou funkci výsledného plug-in modulu. Také je zde uveden způsob zajištění synchronizace tempa a rytmu s hostující aplikací.

Náplní šesté kapitoly je objasnit, jakým způsobem je realizováno vlastní grafické uživatelské rozhraní v technologii VST. Vymezuje, které komponenty z VST SDK pro tvorbu grafiky bude třeba použít pro jednotlivé ovládací prvky a v jaké formě je pro ně nutné připravit rastrové obrázky. Součástí je rovněž popis způsobu, jakým lze pomocí VST SDK realizovat dialogové okno pro výběr souborů.

2 Rozbor základní funkce digitálního metronomu

Metronom je zařízení sloužící k periodickému opakování určitého zvuku. Může existovat jako samostatný výrobek ve formě mechanické či elektronické, nebo jako software. Využití nachází především v hudbě, kde slouží jako pevný udavač tempa, což může například sloužit jako pomůcka při skladbách, u kterých není žádoucí kolísání tempa v průběhu skladby. Základním nastavitelným parametrem metronomu je tedy tempo, které se udává v BPM - beats per minute, což značí počet úderů za minutu. Běžně mívají metronomy rozsah od 35 do 250 BPM. Doplňkovým parametrem metronomu bývá nastavení taktu, kde první doba taktu může být charakterizována zvukem odlišným od zvuků ostatních dob.

2.1 Popis základního algoritmu

Pro realizaci metronomu je nutné vytvořit algoritmus generující zvuk s periodou, jejíž hodnota je určena uživatelem zvoleným tempem *BPM*. Znamená to tedy, že pro docílení opakování zvuku při konkrétně zvoleném tempu je třeba znát vzorkovací frekvenci f_{vz}, aby bylo možné určit počet vzorků n tvořících periodu metronomu. Délku n bloku vzorků, který bude periodicky posílán na výstup, lze zjistit podle vztahu

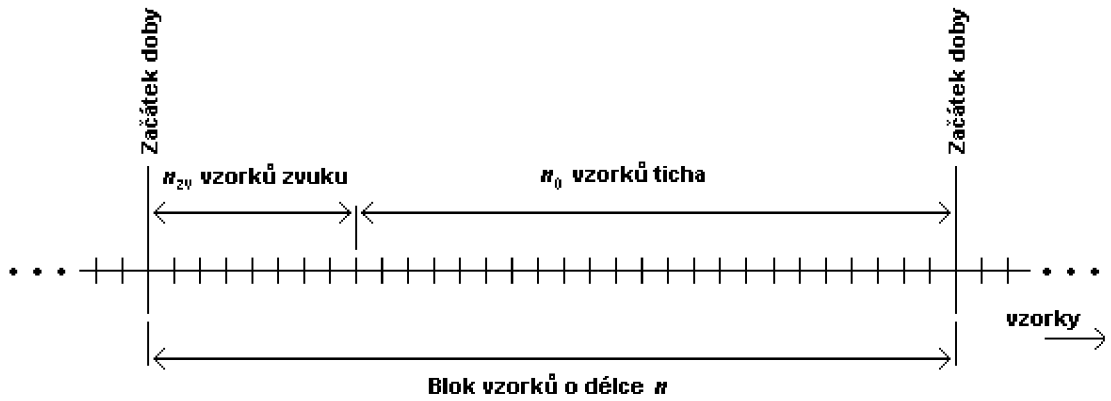
$$n = \frac{60.f_{vz}}{BPM}. \qquad (2.\ 1)$$

Dále je třeba znát počet vzorků zvuku n_{zv}, který bude přehráván. K nim bude přidán počet nulových vzorků n_0 (vzorky ticha) podle vztahu

$$n_0 = n - n_{zv}, \qquad (2.\ 2)$$

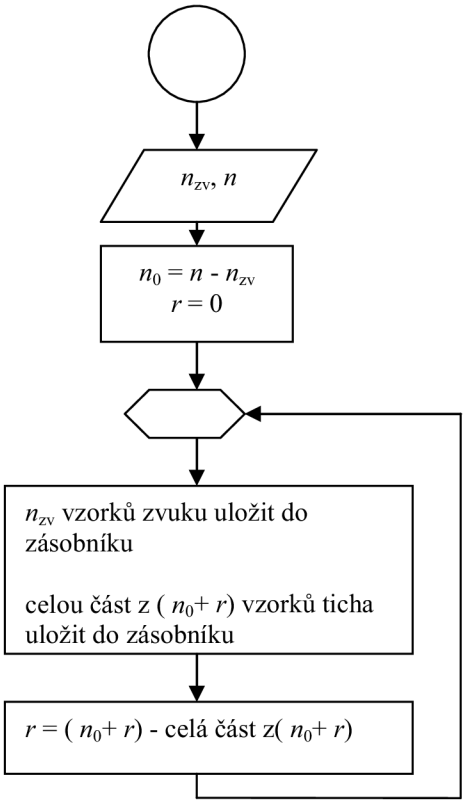
abychom dostali blok vzorků o délce n.

Tento blok o délce n pak bude opakovaně posílán na výstup, což bude mít za následek žádaný periodický sled zvuků. Situace je znázorněna na Obr. 2. 1.



Obr. 2. 1: Sekvence vzorků utvářející zvuk metronomu

Při výpočtu délky bloku vzorků n může obecně vyjít desetinné číslo, a tudíž i počet nulových vzorků n_0 může být desetinné číslo. Jedná-li se však o počet vzorků, nemá smysl uvažovat tuto délku n_0 v desetinném číslu. Avšak zanedbání desetinné části čísla n_0 nám způsobí mírnou odchylku požadovaného tempa a metronom se tak začne nepatrně předbíhat. Způsob, jakým se dá tato odchylka eliminovat, je zobrazen na Obr. 2. 2.



Obr. 2. 2: Eliminace předbíhání zvuku za použití proměnné *r*

Nejprve je zavedena pomocná proměnná *r*, která slouží k ukládání desetinných zbytků. V jednotlivých cyklech na výstup posíláme vždy n_{zv} vzorků zvuku a celou část z hodnoty součtu $n_0 + r$ nulových vzorků. V každém průchodu cyklem se hodnota proměnné *r* zvyšuje o desetinnou část hodnoty n_0. Jakmile proměnná *r* dosáhne takové hodnoty, že její součet s desetinnou částí hodnoty n_0 je větší nebo roven jedné, je v následujícím cyklu na výstup vysláno o jeden vzorek více než v předchozích cyklech (tj. celá část ze součtu $n_0 + r$) a zároveň je hodnota proměnné *r* snížena o jedničku. Tímto je eliminováno předbíhání metronomu.

2.2 Implementace základního algoritmu

Na obrázku v příloze A je zobrazen vývojový diagram znázorňující základní algoritmus metronomu, jenž zajišťuje přehrávání dvou různých zvuků, tzn. první doba taktu je odlišena od ostatních dob. Na začátku algoritmu jsou předány parametry nz_{v1} a nz_{v2}, což jsou počty vzorků zvuků, které máme k dispozici. Počet vzorků zvuku pro první dobu taktu je vyjádřen nz_{v1} (tyto vzorky jsou uloženy v zásobníku $buff_1$), nz_{v2} je počet vzorků zvuku pro ostatní doby taktu (tyto vzorky jsou uloženy v zásobníku $buff_2$). Na základě zvoleného tempa v BPM podle vztahu 2. 1 lze spočítat délku *n* bloku vzorků, z níž se poté ze vztahu 2. 2 určí počty nulových vzorků n_{01} a n_{02}, jež jsou algoritmu také předány. Počet nulových vzorků n_{01} se přidá ke vzorkům prvního zvuku, počet nulových vzorků n_{02} pak ke vzorkům zvuku druhého. V obou případech nám tak vzniknou bloky vzorků o stejné délce *n*.

Dále na začátku algoritmu proběhne inicializace pomocných proměnných ct_{zv}, ct_0, ct_d, *sled*, *doba* a *r*. Proměnné ct_{zv}, ct_0 a ct_d slouží jako čítače. Proměnná *sled* slouží jako příznak, který určuje, jestli do výstupního zásobníku *outBuff* ukládáme vzorky zvuku nebo vzorky

ticha. Proměnná *doba* slouží jako příznak určující dobu taktu, se kterou pracujeme. Proměnná *r* slouží k odkládání desetinných zbytků hodnot proměnných n_{01} a n_{02}.

Při postupu dále algoritmem je testován příznak *sled*. Na základě jeho hodnoty se rozhoduje, zda-li bude výstupní zásobník *outBuff* plněn vzorky zvuku, či vzorky ticha. Jelikož byly pomocné proměnné nainicializovány na výchozí hodnoty (což odpovídá události zapnutí metronomu), bude metronom počítat od první doby. Příznak *sled* tedy nabývá hodnoty určující přehrávání zvuku a postoupíme do levé větve přehrávání zvuku. Zde je čítač ct_{zv}, jehož úkolem je monitorování počtu vzorků zvuku již uložených do výstupního zásobníku *outBuff*, inkrementován o jedničku. Dále proběhne testovaní příznaku *doba*, což určuje, zdali bude výstupní zásobník *outBuff* plněn zvukem první doby nebo zvukem ostatních dob. Jeho výchozí hodnota určuje plnění zásobníku *outBuff* vzorky prvního zvuku. Následuje testování, zda hodnota čítače již nepřesáhla počet vzorků nz_{v1} prvního zvuku. Konečně je tedy první vzorek zvuku první doby uložen do zásobníku *outBuff*. Odtud proběhne přesun téměř na začátek algoritmu a danou větví prvního zvuku bude iterováno tolikrát, kolik vzorků nz_{v1} zvuk obsahuje. Při posledním průchodu větví bude čítač ct_{zv} vynulován a příznak *sled* nastaven na hodnotu, jež určuje plnění zásobníku *outBuff* nulovými vzorky ticha. Při testování příznaku *sled* proběhne přesměrování do pravé větve algoritmu.

Při průchodu touto větví je vždy inkrementován o jedničku čítač ct_0 udávající počet vzorků ticha uložených do zásobníku *outBuff*. Dále je testováním příznaku *doba* určeno, zdali budeme do výstupního zásobníku *outBuff* ukládat počet nulových vzorků odpovídající celým částem hodnot součtů $n_{01}+r$ či $n_{02}+r$. Hodnota proměnné *r* je s každou dobou taktu zvýšena o desetinný zbytek hodnoty n_{01} či n_{02}, což zabraňuje předbíhání metronomu oproti stanovenému tempu. Jakmile bude hodnota součtu desetinné části n_{01} a *r*, respektive desetinné části n_{02} a *r*, větší nebo rovna jedné, bude do výstupního zásobníku *outBuff* uloženo o jeden nulový vzorek více než v předchozích dobách a zároveň bude hodnota proměnné *r* snížena o jedničku. Jakmile je do výstupního zásobníku *outBuff* uložen cílový počet nulových vzorků, čítač ct_0 je vynulován a o jedničku je inkrementován čítač ct_d čítající jednotlivé doby. Hodnota příznaku *sled* je nastavena na hodnotu, jež nás při dalším průchodu nasměruje do levé větve. V závislosti na tom, jestli hodnota čítače přesáhla počet dob v taktu n_d, je určeno, který zvuk bude vložen do zásobníku *outBuff* v další době, což je určeno příznakem *doba*.

3 Volba libovolného zvuku

Jedním z požadavků je, aby si uživatel mohl zvolit libovolný zvuk, kterým bude charakterizována první doba taktu a libovolný zvuk, kterým budou charakterizovány ostatní doby. Je tudíž potřeba vytvořit funkci, jež dokáže ze souboru typu *wav* načítat zvukové vzorky. Pro realizaci takové funkce je nutno vědět, jakým způsobem jsou data v souborech typu *wav* uložených na disku organizována.

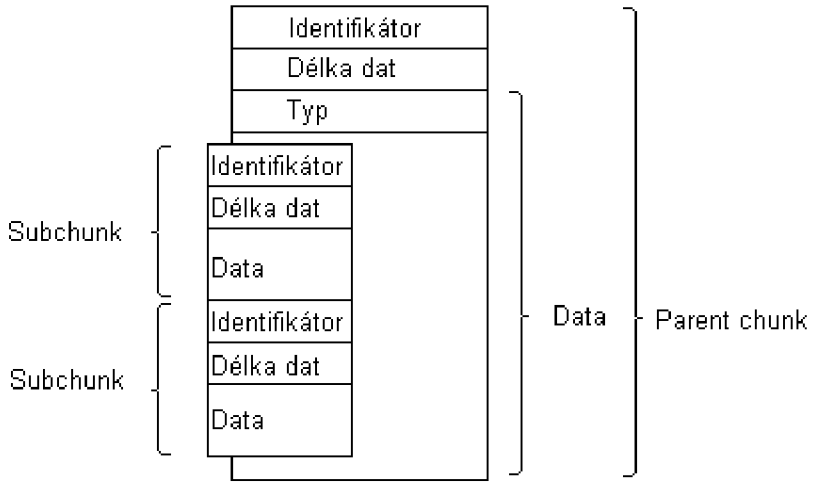
3.1 Resource interchange file format (RIFF)

Data bývají na disk ukládána v jistém standardizovaném formátu. Jeden z takových nejpoužívanějších standardizovaných formátem popisujících strukturu ukládání dat do souboru se nazývá RIFF (viz [1][2]). Tento formát je používán především u multimediálních souborů.

Základním prvkem struktury definované formátem RIFF je blok dat nazývaný *chunk* (v překladu kus). Ten sestává ze záhlaví (obsahující čtyřznakový identifikátor

o velikosti 4 bajtů a délky dat o velikosti také 4 bajty) a části obsahující data. Délka dat takového musí být sudá, je-li velikost obsažených dat lichá, jsou tato data doplněna nulovým bajtem.

Chunky do sebe mohou být i vnořeny, tzn., že je jeden obsažen v datové části druhého. Pak mluvíme o podřízeném *chunku*, nazývaném *subchunk*, a nadřazeném *chunku*, nazývaném *parent chunk*. *Parent chunky* mohou být pouze ty, které mají identifikátor typu RIFF nebo LIST, a za jejich záhlaví je pak ještě přidána čtyřbajtová informace o jejich typu. RIFF obvykle obsahuje samostatné datové struktury a LIST seznamy položek. Základní *chunk* souboru formátu RIFF musí mít identifikátor typu RIFF. Pro lepší představu je RIFF struktura zobrazená na Obr. 3.1.



Obr. 3. 1: Ilustrace RIFF struktury

3.2 Formát souboru wav

Soubory typu *wav* používají pro uložení dat výše uvedenou strukturu RIFF. Nejnadřazenější *chunk* má identifikátor typu RIFF a aby bylo zřejmé, že se jedná o soubor typu *wav*, v poli nesoucím informaci o typu musí být uloženo WAVE. Pro soubor typu *wav* je pak definováno několik *subchunků*, které nesou informace o daném souboru. V následujícím textu jsou vyjmenovány nejdůležitější z nich a stručně popsán jejich obsah :

- *subchunk Format*

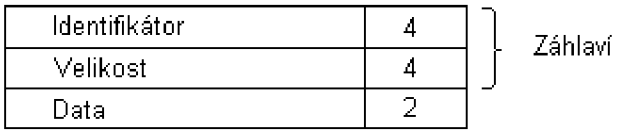
Jeho struktura je zobrazena na Obr. 3. 2 a lze vidět, jaké informace obsahuje a také velikost těchto informací v bajtech. Typ *Format* je určen čtyřznakovým identifikátorem *fmt* obsaženým v záhlaví. Položka formát kódování může obsahovat určité číselné hodnoty, z nichž každá odpovídá jistému způsobu kódování audio dat. Obsahuje-li hodnotu 1, pak jsou audio data v podobě nekomprimované PCM. V aplikaci pro metronom bude uvažováno pouze kódování audio dat ve formě nekomprimované PCM.

- *subchunk Data*

Jeho strukturu lze vidět na Obr. 3. 3. Typ *Data* je určen čtyřznakovým identifikátorem *data*. Po záhlaví následují již data obsahující jednotlivé vzorky audio signálu. Jedná-li se

o vícekanálové audio, jsou kanály sloučeny, tj. po jednom celém vzorku prvního kanálu následuje jeden celý vzorek druhého kanálu atd. Teprve po prvním vzorku posledního kanálu následuje druhý vzorek prvního kanálu. Jsou-li vzorky uloženy maximálně v osmi bitech, pak jsou uloženy jako bezznaménkové hodnoty. Jsou-li vzorky uloženy ve více než osmi bitech, pak se využívá reprezentace čísla pomocí dvojkového doplňku. Pokud počet bitů na vzorek není celistvým násobkem osmi, pak je daný vzorek doplněn nulovými bity, aby se dal uložit do celých bajtů.

Obr. 3. 2: Ilustrace struktury *format chunku*

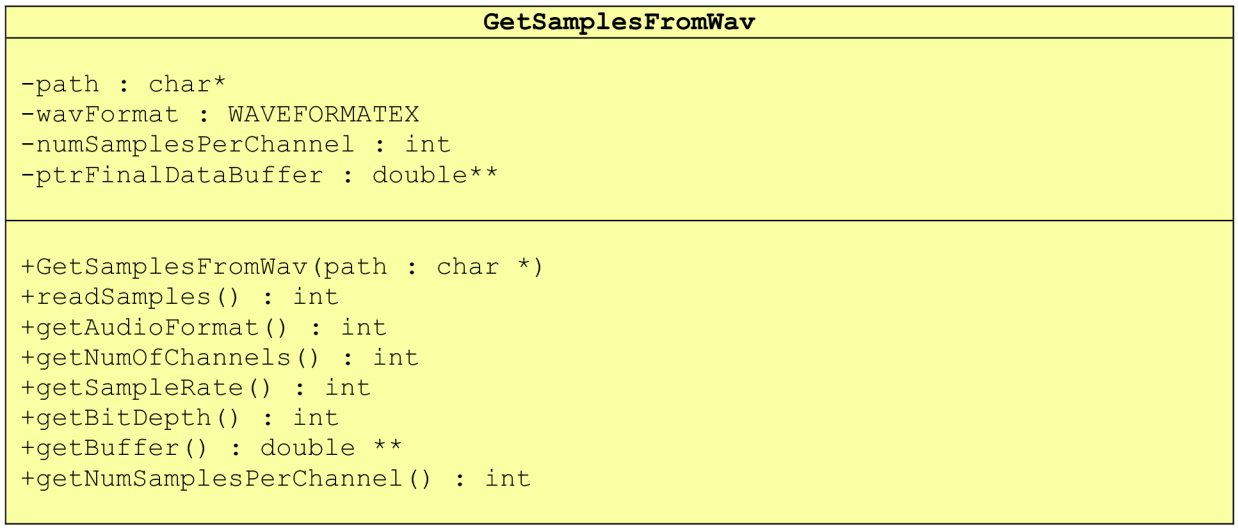


Obr. 3. 3: Ilustrace struktury *data chunku*

Dále může soubor typu *wav* obsahovat ještě další typy *chunků*, jako jsou například *Factual length, Sampler, Cue, PAD*, které mohou obsahovat informace pro hudební nástroje. Pro aplikaci metronomu však není nutné se těmito typy *chunků* zabývat.

3.3 Implementace načítání zvuků z wav souboru

Pro účel získání vzorků zvuku, jenž bude použit do metronomu, je vytvořena třída v jazyce C++, jejímž úkolem bude otevření souboru typu *wav*, načtení dat, ze kterých se získají hodnoty jednotlivých vzorků a uložení těchto vzorků do vyhrazeného zásobníku v paměti. Třída bude také schopna poskytovat informace o formátu zvukového souboru jako jsou vzorkovací frekvence, počet bitů na vzorek, počet kanálů atd. Deklaraci této třídy lze vidět na Obr. 3. 4. Třída obsahuje několik veřejných metod a několik soukromých členů. Při vytvoření této třídy je zavolán konstruktor `GetSamplesFromWav`, kterému se jako parametr předá řetězec obsahující cestu k souboru typu *wav*. Tento řetězec je následně zkopírován do vyhrazené paměti, na kterou odkazuje ukazatel `path`. Třída dále obsahuje strukturu `wavFormat`, jejíž typ `WAVEFORMATEX` je deklarován v hlavičkovém souboru `MMSystem.h`. Tato struktura obsahuje jednotlivé položky *subchunku Format* uvedeného v odstavci 3. 2. Proměnná `numSamplesPerChannel` nese údaj, kolik vzorků obsahuje jeden audio kanál.



Obr. 3. 4: Deklarace třídy pro načítání wav souborů

Dále je zde ukazatel `ptrFinalDataBuffer` na zásobník, do kterého se ukládají zvukové vzorky. Veřejné metody slouží k získaní ukazatele na zásobník se vzorky nebo pro získání jednotlivých informací ze struktury `wavFormat`. Zavoláním metody `readSamples()` se načtou vzorky zvuku ze souboru a uloží do zásobníku.

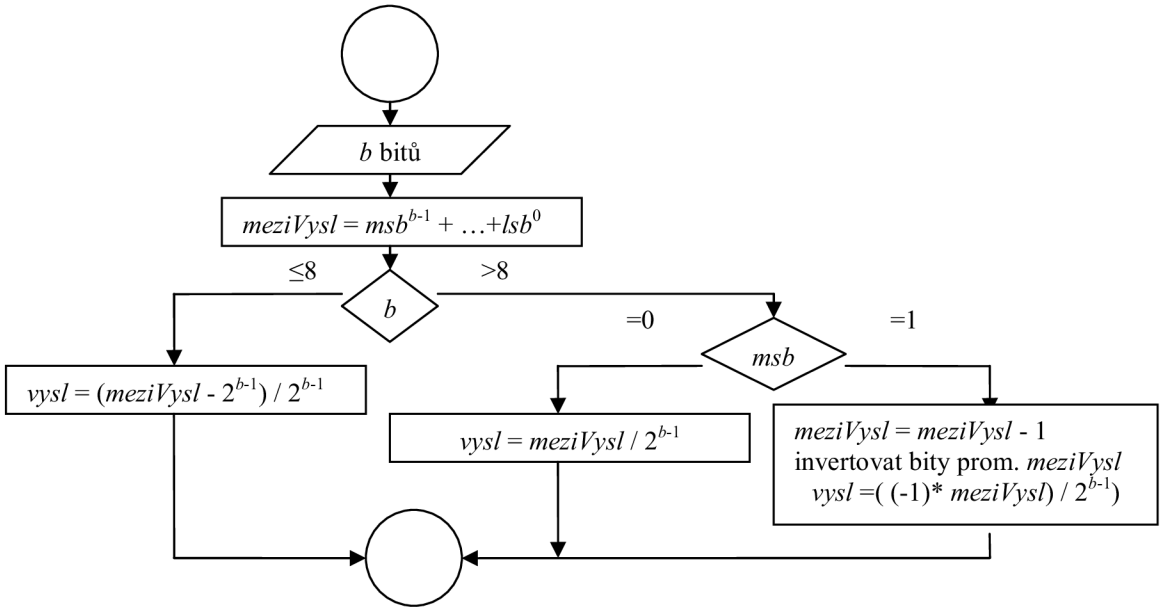
Metoda `readSamples` používá funkce `mmioOpen`, `mmioRead`, `mmioClose`, `mmioDescend` a `mmioAscend`, což jsou funkce usnadňující manipulaci s multimediálními soubory (viz. [3]). Funkce `mmioOpen` a `mmioClose` slouží k otevření či zavření multimediálních souborů, se kterými potom lze manipulovat. Funkce `mmioDescend` a `mmioAscend` umožňují zanořovat se a vynořovat se na pozice jednotlivých *chunků* obsažených v souboru s formátem RIFF. Metoda `readSamples` je znázorněna na vývojovém diagramu v příloze B. Načtení tedy proběhne tak, že metoda `readSamples` má k dispozici cestu k souboru, který chceme načíst, ve formě řetězce `path`. Proto tento soubor můžeme otevřít. V případě splnění podmínky správného načtení souboru je zjištěno, zdali se jedná o soubor typu *wav*, tzn. informace o typu hlavního RIFF chunku je WAVE. Pokud se jedná o soubor typu *wav*, je nalezen *subchunk Format*, který se načte a uloží do struktury `wavFormat`. Tím jsou známy základní informace o uloženém zvuku nutné ke správné interpetaci zvuku. Nyní je nalezen *subchunk Data* obsahující zvukové vzorky a tyto vzorky jsou načteny do paměti. Na základě znalosti informací přečtených ze *subchunku Format* a na základě informací uvedených v odstavci 3. 2 je možné převést načtená data na hodnoty jednotlivých vzorků a tyto výsledné hodnoty uložit do zásobníku, na který se odkazuje ukazatel `ptrFinalDataBuffer`.

Pokud se vše podařilo, otevřený soubor lze zavřít. Nepodařilo-li se během procedury některý z dílčích úkolů vykonat, vrátí metoda readSamples hodnotu různou od nuly, což značí chybu.

Algoritmus metronomu pracuje se vzorky v rozsahu <-1;1>, proto je potřeba načtená data ze souboru do tohoto formátu převést. Tento převod je zobrazen na Obr. 3. 5. Ze struktury `wavFormat` je známo, kolika bity *b* jsou vyjádřeny jednotlivé vzorky zvuku. Binární hodnota daných *b* bitů je převedena na celé číslo vyjádřené dekadicky a uložena do proměnné *meziVysl*.

- Je-li počet bitů *b* menší nebo roven osmi, pak je tato hodnota uložena jako bezznaménková. Proto je nejdříve od proměnné *meziVysl* odečteno číslo 2^{b-1}, což je polovina maximální hodnoty, která lze uložit do *b* bitů. Číslem 2^{b-1} je tak daný rozdíl vydělen a je získána hodnota vzorku v rozmezí <-1;1>.

- Je-li počet bitů *b* větší než osm, pak je hodnota uložena jako dvojkový doplněk. Rovná-li se nejvýznamější bit *msb* nule, jedná se o kladné číslo a hodnota proměnné *meziVysl* je podělena pouze 2^{b-1}. Tím je získán výsledek v rozmezí <0;1>. Je-li nejvýznamější bit *msb* roven jedné, jde o záporné číslo. Od hodnoty proměnné *meziVysl* se odečte jednička a bitově je tato hodnota invertována. Následně je hodnota proměnné *meziVysl* vydělena 2^{b-1} a obráceno její znaménko. Tím je získán výsledek v rozmezí <-1;0).



Obr. 3. 5: Převedení dat na hodnoty vzorků

4 Převzorkování digitálního zvukového signálu

Po metronomu je požadována ta vlastnost, aby byl schopen pracovat i při různých, ideálně libovolných vzorkovacích frekvencích. V souvislosti s předchozí požadovanou vlastností metronomu, tj. načítání libovolného zvuku, se může stát, že je načten zvuk, který byl navzorkován určitou frekvencí, ale frekvence, s jakou bude výsledný zvuk přehráván, je odlišná. Proto je nutno opatřit metronom funkcí, která provede převzorkování signálu, aby při různých vzorkovacích frekvencích byla zachována výška a délka zvuku.

4.1 Teoretický rozbor převzorkování

Převzorkování signálu v poměru racionálního čísla se provádí tak, že se signál nejprve v poměru nějakého celého čísla nadvzorkuje a poté se v poměru jiného celého čísla podvzorkuje, viz.[4].

- **podvzorkování signálu v poměru celého čísla *M***

Podvzorkováním signálu je chápán postup snižování jeho vzorkovací frekvence, což je prováděno výběrem každého M - tého vzorku z původního signálu. U této operace musí být dodržen vzorkovací teorém, tzn. nejvyšší harmonická složka původního signálu nesmí být větší než polovina vzorkovací frekvence, na kterou je signál podvzorkováván. Při nedodržení této podmínky se projeví aliasing, což má za následek degradaci signálu.

Blokové schéma nadvzorkování je na Obr. 4. 1. Jak se mění spektrum signálu při podvzorkování, lze vidět na Obr. 4. 2. Jedná se o případ, kdy je signál podvzorkováván na poloviční vzorkovací frekvenci, tedy $M = 2$. Nejprve je třeba spektrum signálu $x[n]$ omezit pomocí filtru typu dolní propust tak, aby nejvyšší harmonická složka byla rovna maximálně hodnotě $f_{vzx}/2M$. Modulová frekvenční charakteristika filtru $|H(e^{j2\pi f})|$ je zobrazena na Obr. 4. 2. Pokud je každý M-tý vzorek (v případě na obrázku druhý) vzorek signálu $x[n]$ vynulován, vznikne signál $x'[n]$, jehož spektrum je tvořeno spektrem filtrovaného signálu $x[n]$ a spektry signálu $x[n]$ posunutých o celé násobky f_{vzx}/M. Spektrum signálu $y[n]$ bude stejné jako spektrum signálu $x'[n]$, pouze se změní poloha vzorkovací frekvence.

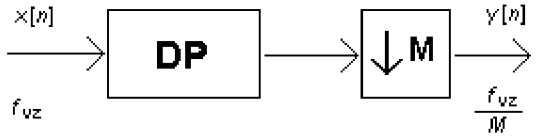
- **nadvzorkování signálu v poměru celého čísla *L***

Nadvzorkováním signálu se rozumí postup zvyšování jeho vzorkovací frekvence, což je prováděno vložením L - 1 nových aproximovaných hodnot vzorků mezi každé dva sousední vzorky původního signálu. Prakticky je to realizováno tak, že mezi každé dva sousední vzorky původního signálu je vloženo L - 1 nových nulových hodnot vzorků. Tyto nulové hodnoty jsou pak aproximovány pomocí filtru typu dolní propust. Blokové schéma pro nadvzorkování je na Obr. 4. 3, situace nadvzorkování signálu je zobrazena na Obr. 4. 4. Signál je zde nadvzorkován činitelem $L = 3$. Mezi každé dva sousední vzorky signálu $x[n]$ jsou vloženy dvě nulové hodnoty vzorků. Takto vzniklý signál $x'[n]$ má spektrum obohaceno o vyšší harmonické složky. Tyto vyšší harmonické složky jsou vlastně posunutím spektra původního signálu $x[n]$ o celé násobky původní vzorkovací frekvence f_{vzx}. K odstranění těchto vyšších harmonických složek, které v původním signálu $x[n]$ nebyly, je použit filtr typu dolní propust s nepropustným pásmem od $f_{vzx}/2$. Použitím tohoto filtru je docílena v časové oblasti aproximace nulových hodnot vzorků na hodnoty, jež přibližně odpovídají hodnotám vzorků původního signálu $x[n]$. Je tak získán výsledný signál $y[n]$ se vzorkovací frekvencí f_{vzy}.

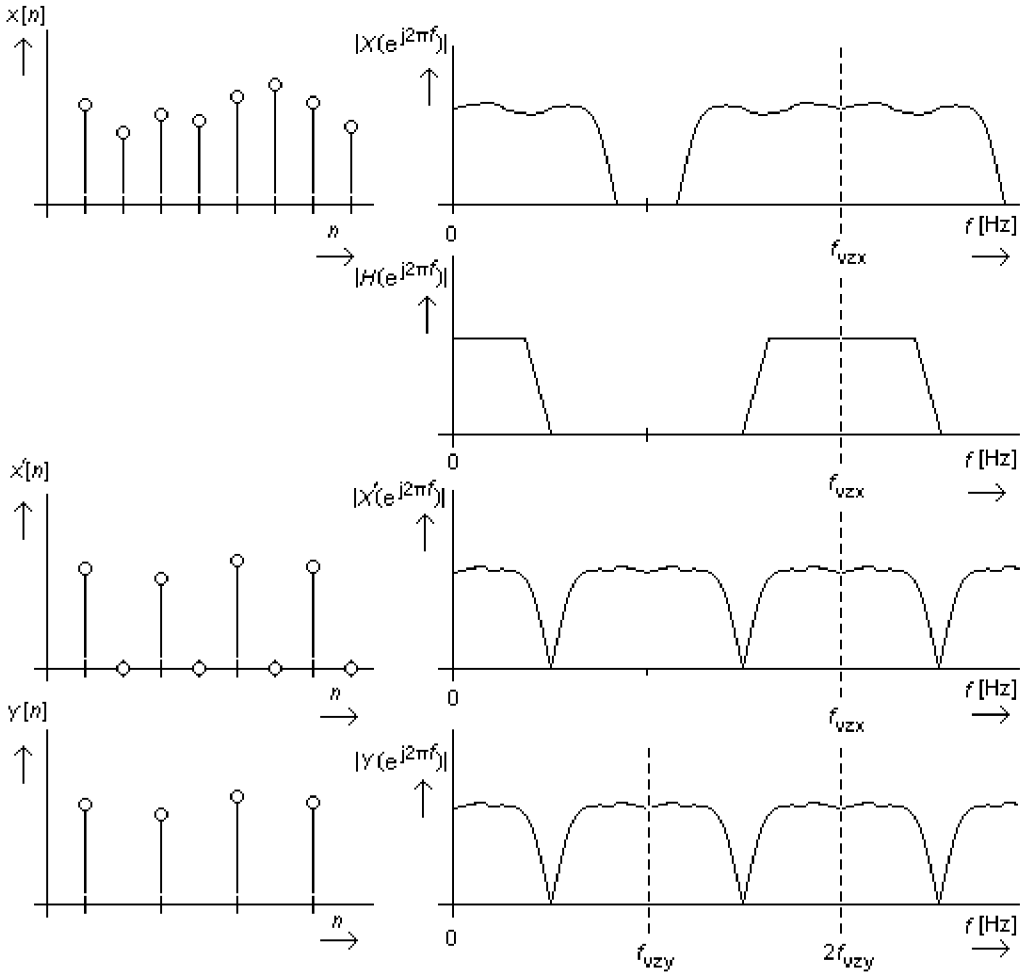
Bude-li tedy třeba převzorkovat číslicový signál o vzorkovací frekvenci f_{vzx} na signál o vzorkovací frekvenci f_{vzy}, je nejprve nalezen nejmenší společný násobek *NSN* hodnot frekvencí f_{vzx} a f_{vzy}. Na tento násobek *NSN* vzorkovacích frekvencí je nejprve signál nadvzorkován činitelem rovnému podílu NSN/f_{vzx}. Z této vzorkovací frekvence pak signál podvzorkován činitelem NSN/f_{vzy} na signál o vzorkovací frekvenci f_{vzy}.

Aproximaci hodnot při nadvzorkování a antialiasingové omezení spektra při podvzorkování je možné při převzorkování číslicového signálu zajistit jedním číslicovým filtrem. Pokud bude platit $f_{vzx} > f_{vzy}$, pak filtr musí mít mezní kmitočet nastaven maximálně na $f_{vzy}/2$ Hz. Pokud bude $f_{vzy} > f_{vzx}$, pak filtr musí mít mezní kmitočet nastaven maximálně na $f_{vzx}/2$ Hz.

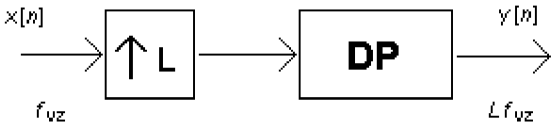
Proces převzorkování digitálního signálu ilustruje blokové schéma na Obr. 4.5.



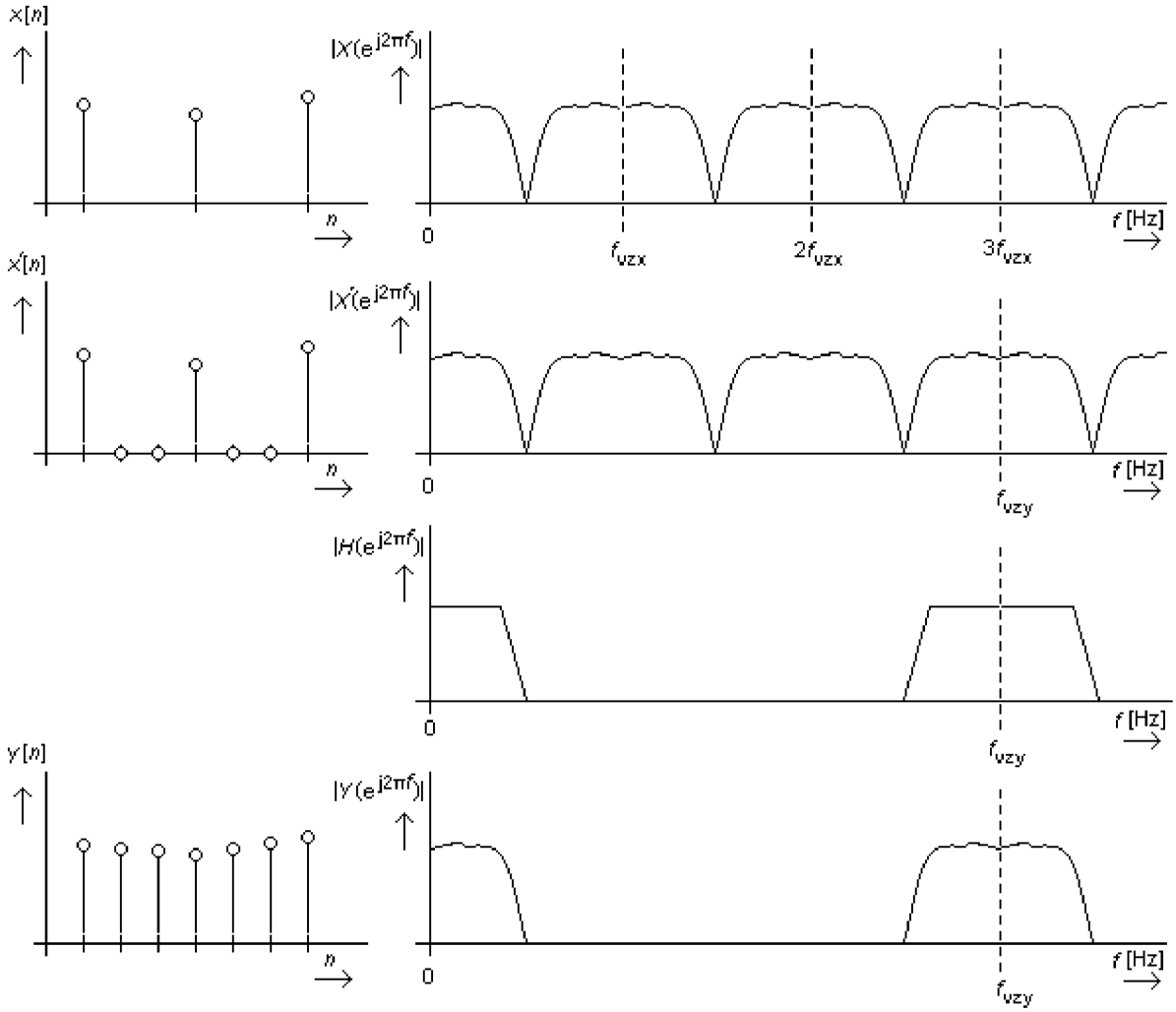
Obr. 4. 1: Podvzorkování digitálního signálu v poměru M



Obr. 4. 2: Ukázka spekter signálu při podvzorkování v poměru $M = 2$



Obr. 4. 3: Nadvzorkování digitálního signálu v poměru L



Obr. 4. 4: Ukázka spekter signálu při nadvzorkování v poměru $L = 3$

Obr. 4. 5: Převzorkování digitálního signálu v poměru L /M

Je nutné navrhnout takový filtr, jehož proměnným parametrem bude hodnota mezního kmitočtu. Ta je zadávána na základě znalosti původní vzorkovací frekvence f_{vzx} a cílové vzorkovací frekvence f_{vzy} podle výše uvedených pravidel. Jelikož v případě digitálního metronomu při filtrování digitálního zvukového signálu nezáleží na fázovém zkreslení signálu, je možné využít číslicového filtru typu IIR, který pro dosažení daného výsledku postačí nižšího řádu.

Číslicové filtry vyšších řádů se často realizují formou kaskádního řazení dílčích číslicových filtrů 2. řádu, jelikož filtry vyšších řádů jsou mnohem citlivější na zaokrouhlování stavových proměnných. V případě digitálního metronomu by bylo vhodné navrhnout filtr alespoň 8. až 10. řádu, aby však nebylo třeba navrhovat více filtrů, bude návrh omezen na dva filtry 4. řádu.

Jednou z možných metod návrhu číslicového filtru typu IIR je metoda bilineární transformace (viz. lit. [5]). Je založena na prvotním návrhu analogového filtru požadovaných vlastností a jeho následného převedení na číslicový filtr pomocí vztahu bilineární transformace. Nyní následuje stručný popis tohoto návrhu.

Filtr bude určen parametrem ω_c, což bude značit mezní úhlový kmitočet propustného pásma filtru. Aby bylo možné navrhnout odpovídající analogový filtr, je nutné parametr ω_c převést na odpovídající parametr úhlového kmitočtu analogového filtru ω_p

$$\omega_p = \frac{2}{T}\operatorname{tg}\frac{\omega_c T}{2}, \qquad (4.\ 1)$$

což lze označit jako předzkreslení kmitočtové osy.

Člen T odpovídá vzorkovací periodě, pro kterou je daný číslicový filtr navrhován. Následuje převod na tzv. normovanou dolní propust. Jelikož číslicový filtr, který je navrhován, je také dolní propust, bude převáděna dolní propust s parametrem mezního úhlového kmitočtu ω_p na normovanou dolní propust s parametrem mezního úhlového kmitočtu rovnému hodnotě 1. Pro takovou normovanou dolní propust 4. řádu bude přenosová funkce ve tvaru

$$H(p) = \frac{c_0}{c_0 + c_1 p + c_2 p^2 + c_3 p^3 + c_4 p^4}, \qquad (4.\ 2)$$

kde c_0 - c_4 lze zjistit z tabulky pro daný typ aproximace (např. Butterworthova) v příloze [6] . Dalším krokem je převod této přenosové funkce do roviny Z. To se provede pomocí vztahu pro bilineární transformaci

$$p = \frac{2}{T}\frac{z-1}{z+1}. \qquad (4.\ 3)$$

Je-li převáděna analogová normovaná dolní propust na číslicovou dolní propust, za operátor p se dosadí

$$p = \frac{2}{\omega_p T}\frac{z-1}{z+1} \quad . \qquad (4.\ 4)$$

Úpravami je získána transformovaná přenosová funkce $H(z)$ ve tvaru

$$H(z) = \frac{b_4 z^4 + b_3 z^3 + b_2 z^2 + b_1 z^1 + b_0}{a_4 z^4 + a_3 z^3 + a_2 z^2 + a_1 z^1 + a_0}. \qquad (4.\ 5)$$

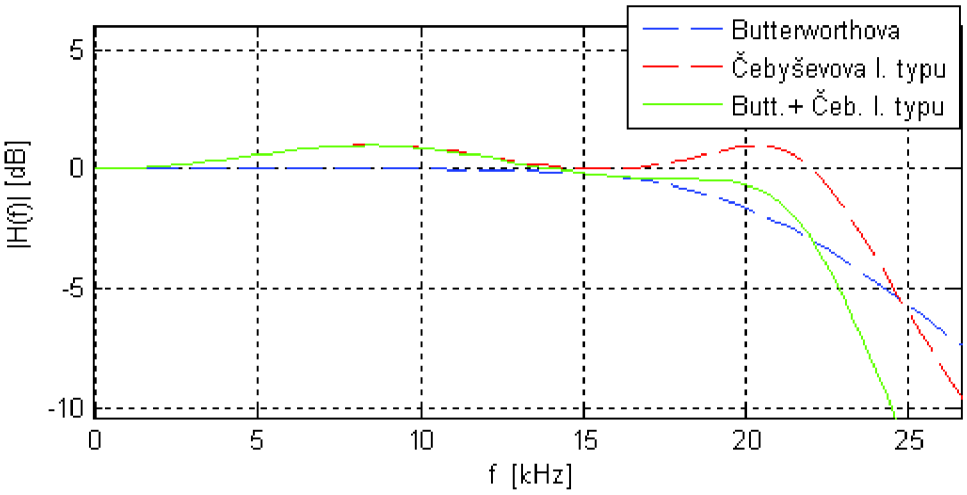
Po úpravách jsou pro jednotlivé koeficienty přenosové funkce $H(z)$ tyto vztahy

$$\begin{aligned}
b_4 &= \omega_p{}^4 T^4 c_0 \\
b_3 &= 4\omega_p{}^4 T^4 c_0 \\
b_2 &= 6\omega_p{}^4 T^4 c_0 \\
b_1 &= 4\omega_p{}^4 T^4 c_0 \\
b_0 &= \omega_p{}^4 T^4 c_0
\end{aligned}$$

$$\begin{aligned}
a_4 &= \omega_p{}^4 T^4 c_0 + 2\omega_p{}^3 T^3 c_1 + 4\omega_p{}^2 T^2 c_2 + 8\omega_p T c_3 + 16 c_4 \\
a_3 &= 4\omega_p{}^4 T^4 c_0 + 4\omega_p{}^3 T^3 c_1 - 16\omega_p T c_3 - 64 c_4 \\
a_2 &= 6\omega_p{}^4 T^4 c_0 - 8\omega_p{}^2 T^2 c_2 + 96 c_4 \\
a_1 &= 4\omega_p{}^4 T^4 c_0 - 4\omega_p{}^3 T^3 c_1 + 16\omega_p T c_3 - 64 c_4 \\
a_0 &= \omega_p{}^4 T^4 c_0 - 2\omega_p{}^3 T^3 c_1 + 4\omega_p{}^2 T^2 c_2 - 8\omega_p T c_3 + 16 c_4
\end{aligned} \qquad (4.\ 6)$$

Nyní jsou známe vztahy pro výpočet jednotlivých koeficientů filtru typu IIR. Jelikož byl požadován filtr 8. řádu, bude realizován pomocí dvou filtrů 4. řádu. V v příloze lit. [5]. jsou k dispozici koeficienty c_0 - c_4 pro dva základní typy aproximací. Jsou to aproximace

Butterworthova a Čebyševova. Pro Butterworthovu aproximaci je typická monotónně klesající modulová kmitočtová charakteristika v celém rozsahu, avšak klesaní v nepropustném pásmu je méně strmé. Pro Čebyševovu aproximaci I. typu je charakteristické zvlnění modulové kmitočtové charakteristiky v propustném pásmu a monotónní klesání charakteristiky v pásmu nepropustném s větší strmostí než v aproximaci Butterworthově. Ze simulací v Matlabu vyplývá, že by bylo vhodné koeficienty přenosové funkce $H(z)$ pro jeden



Obr. 4. 6: Zobrazení modulových kmitočtových charakteristik

filtr spočítat dosazením koeficientů pro Butterworthovu aproximaci s maximálním poklesem o 3dB v propustném pásmu a pro druhý spočítat dosazením koeficientů pro Čebyševovu aproximaci I. typu pro maximální zvlnění 1 dB v propustném pásmu. Na Obr. 4. 6. je tato situace znázorněna. Červená čárkovaná křivka odpovídá Čebyševově aproximaci a modrá čárkovaná křivka odpovídá Butterworthově aproximaci. Jejich součtem je pak zelená křivka. Daný filtr má mezní frekvenci nastavenu na 22,05 kHz. Je zřejmé, že zvlnění na konci přenosového pásma částečně kompenzováno.

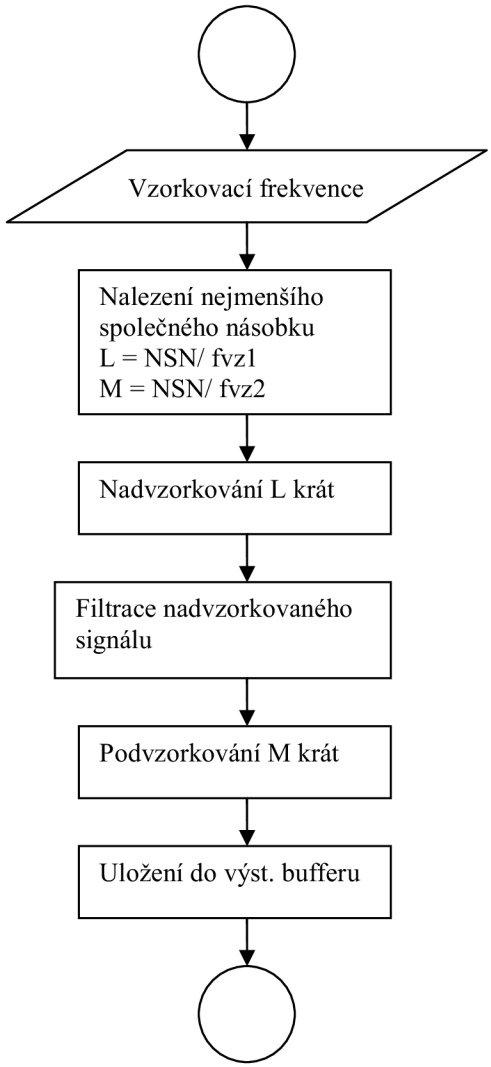
Při poslechových testech, kdy byly převzorkovávány krátké zvuky mezi frekvencemi typickými pro vzorkování audia (44,1 kHz, 48 kHz, 88 kHz atd.) se jevilo jako dostačující použití dvou až tří sekcí filtrů 4. řádu. Viz. skript v Matlabu na přiloženém CD.

4.2 Implementace převzorkování

Výše zmíněný rozbor je nyní nutno implementovat do třídy, která bude používána v případě nutnosti převzorkování načteného zvuku. Deklaraci takové třídy je vidět na Obr. 4. 7. Objekt této třídy bude vytvořen tehdy, bude-li objekt třídy metronomu potřebovat převzorkovat zvuk, jenž načetl a jehož vzorkovací frekvence neodpovídá výstupní vzorkovací frekvenci, se kterou pracuje. V tomto případě bude vytvořenému objektu třídy `Resample` předán zásobník obsahující vzorky zvukového signálu, který je třeba převzorkovat, dále počet vzorků jednoho kanálu, vzorkovací frekvenci načteného zvuku a počet kanálů. Tuto akci zajišťuje metoda `loadInputBuffer`. Vstupní data se zapouzdří do objektu `InDataBuffer` typu `CSoundSampleBufferPCM`. Tato třída byla deklarována za účelem zjednodušení předávání zvukových zásobníků mezi jednotlivými objekty. Následným zavoláním metody `resample` je provedeno převzorkování načteného zvuku, vzorky

```
Resample
-ptrOutDataBuffer : double**
-ptrWorkDataBuffer : double**
-numSamplesPerChannelOut : int
-InDataBuffer : CSoundSampleBufferPCM
-OutDataBuffer : CSoundSampleBufferPCM
+Resample()
+getBufferOfSamples() : CSoundSampleBufferPCM &
+loadInputBuffer(InputBuffer : CSoundSampleBufferPCM)
+loadInputBuffer(ptrBuffer : double **, numSamplesPerChannel : int,
    sampleRate : float, numOfChannels : int)
+resample(outSampleRate : float)
-lcm(num1 : float, num2 : float) : float
-filter(cutoffFreq : double, T : double, lengthOfWorkBuffer : int,
    approx : int)
```

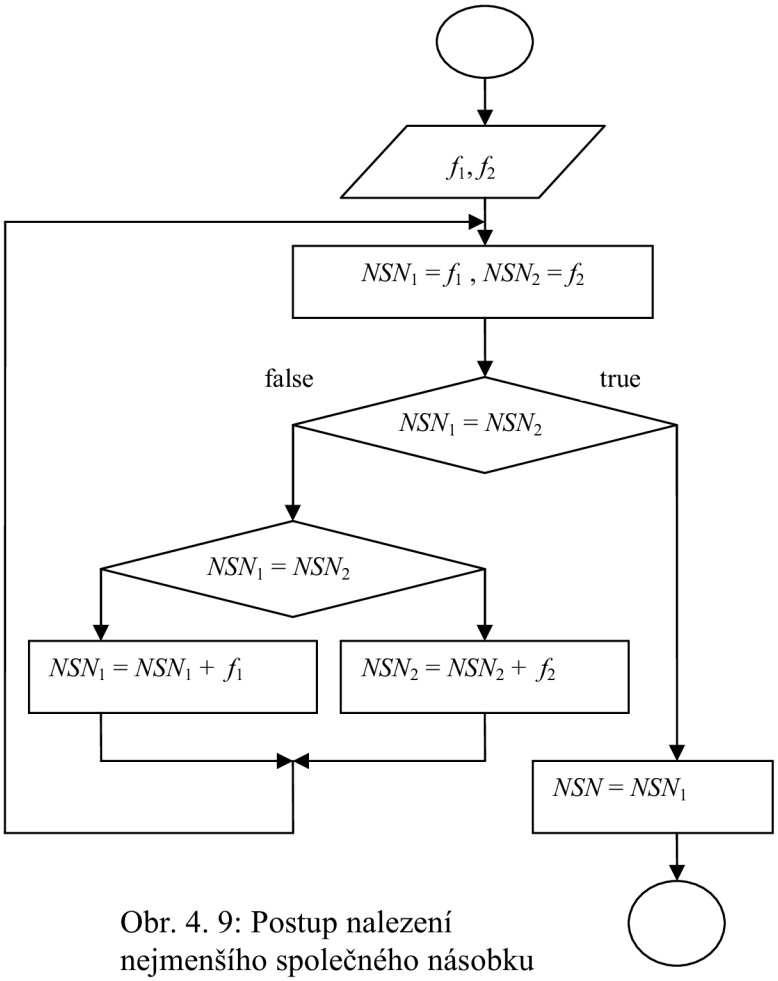
Obr. 4. 7: Deklarace třídy pro převzorkování signálu



Obr. 4. 8: Průběh převzorkování

převzorkovaného zvuku se uloží do objektu `OutDataBuffer`. V deklaraci třídy je také metoda `lcm` sloužící k nalezení nejmenšího společného násobku dvou čísel a metoda `filter`, která slouží k interpolaci nadvzorkovaného signálu. Parametry této metody jsou mezní frekvence, vzorkovací perioda, délka signálu a zvolená aproximace (Čebyševova nebo Butterworthova).

Jednotlivé kroky, které jsou vykonány po zavolání metody `resample` jsou zobrazeny ve vývojovém diagramu na Obr. 4. 8. Metodě je předána hodnota vzorkovací frekvence, na kterou bude převzorkován vstupní zvukový signál. Je zavolána metoda `lcm`, které je předána hodnota požadované vzorkovací frekvence a hodnota vzorkovací frekvence vstupního zvuku uložená v objetku `InDataBuffer`. Z těchto hodnot je určena hodnota nejmenšího společného násobku, což bude hodnota vzorkovací frekvence, na níž je v následujícím kroku nadvzorkován vstupní zvukový signál vložením L - 1 nulových vzorků mezi dva sousední vzorky vstupního signálu. Jak pracuje metoda `lcm`, lze vidět na vývojovém diagramu na Obr. 4.9.

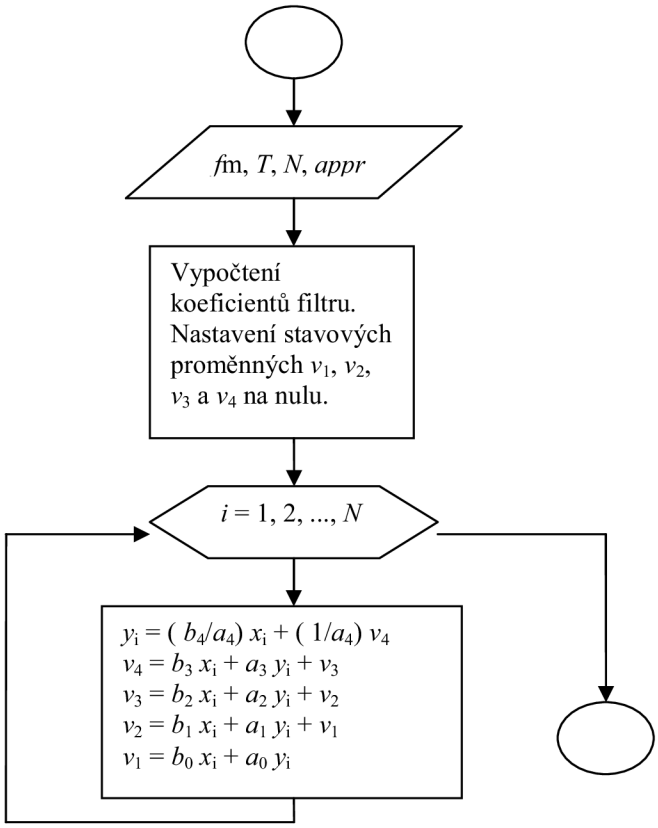


Obr. 4. 9: Postup nalezení nejmenšího společného násobku

Na začátku jsou jí předány hodnoty čísel *f1* a *f2*. Hodnoty proměnných *NSN1* a *NSN2* jsou nainicializovány na hodnoty *f1* a *f2*. Pokud si hodnoty *NSN1* a *NSN2* nejsou rovny, je vždy k menší z nich přičtena hodnota proměnné *f1*, respektive *f2*. Tento postup je opakován tak

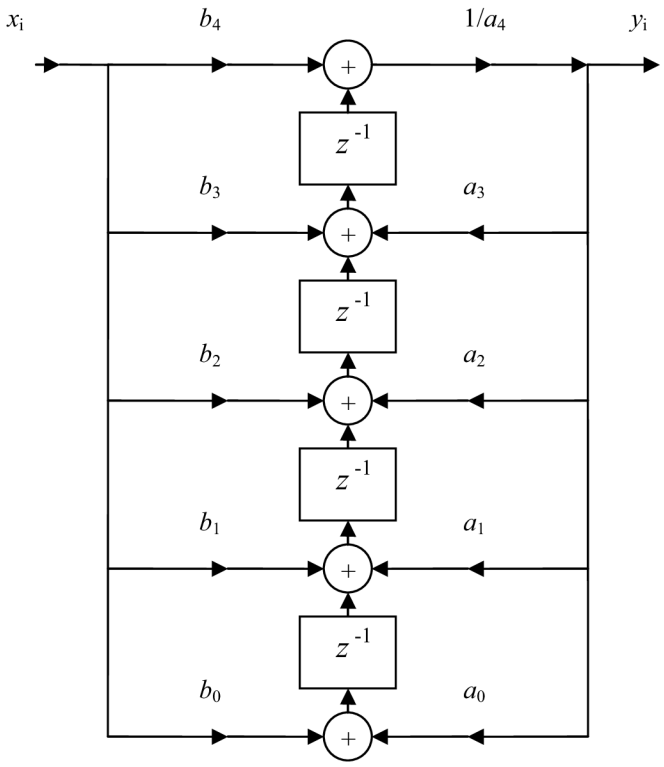
dlouho, dokud si hodnoty *NSN1* a *NSN2* nejsou rovny. Dosažením rovnosti hodnot *NSN1* a *NSN2* je nalezen nejmenší společný násobek *NSN*.

Po nadvzorkování signálu *L*-krát je nutno jej přefiltrovat přes filtr typu dolní propust, což je zajištěno zavoláním metody `filter`. Její struktura je zobrazena na vývojovém diagramu na obr. 4. 10.



Obr. 4. 10: Filtrace signálu

Tato metoda na základě parametrů mezní frekvence, vzorkovací periody a aproximace, které jsou jí předány při jejím zavolání, spočítá podle vztahů 4. 6 koeficienty filtru. Filtr je relizován pomocí první kanonické struktury, jejíž graf signálových toků je zobrazen na Obr. 4. 11., viz [5]. Před filtrací je nutné stavové proměnné nainicializovat na nulové hodnoty. Nadvzorkovaný signál je přefiltrován průchodem jednotlivých vzorků první kanonickou strukturou. Metoda `filter` zajistí filtrování dolní propustí 4. řádu. Tuto metodu je nutno zavolat tolikrát, aby zkreslení vzniklé nadvzorkováním a podvzorkováním bylo potlačeno do takové míry, kdy bude maskováno užitečným zvukovým signálem a pro lidský sluch tak bude neslyšitelné. Pro zvuky, které jsou převzorkovávány mezi frekvencemi typické pro vzrokování audia (22,05 kHz, 44,1 kHz, 48 kHz, atd.) postačí průchod třemi sekcemi filtrů 4. řádu, metoda `filter` bude tudíž zavolána třikrát. Po nadvzorkování a omezení spektra nadvzorkovaného signálu je signál *M*-krát podvzorkován, tj. ze signálu vybereme každý *M*-tý vzorek. Výsledný podvzorkovaný signál je uložen do zásobníku `OutDataBuffer`, kde se uloží i informace o vzorkovací frekvenci, počtu kanálů a počtu vzorků.



Obr. 4. 11: První kanonická struktura

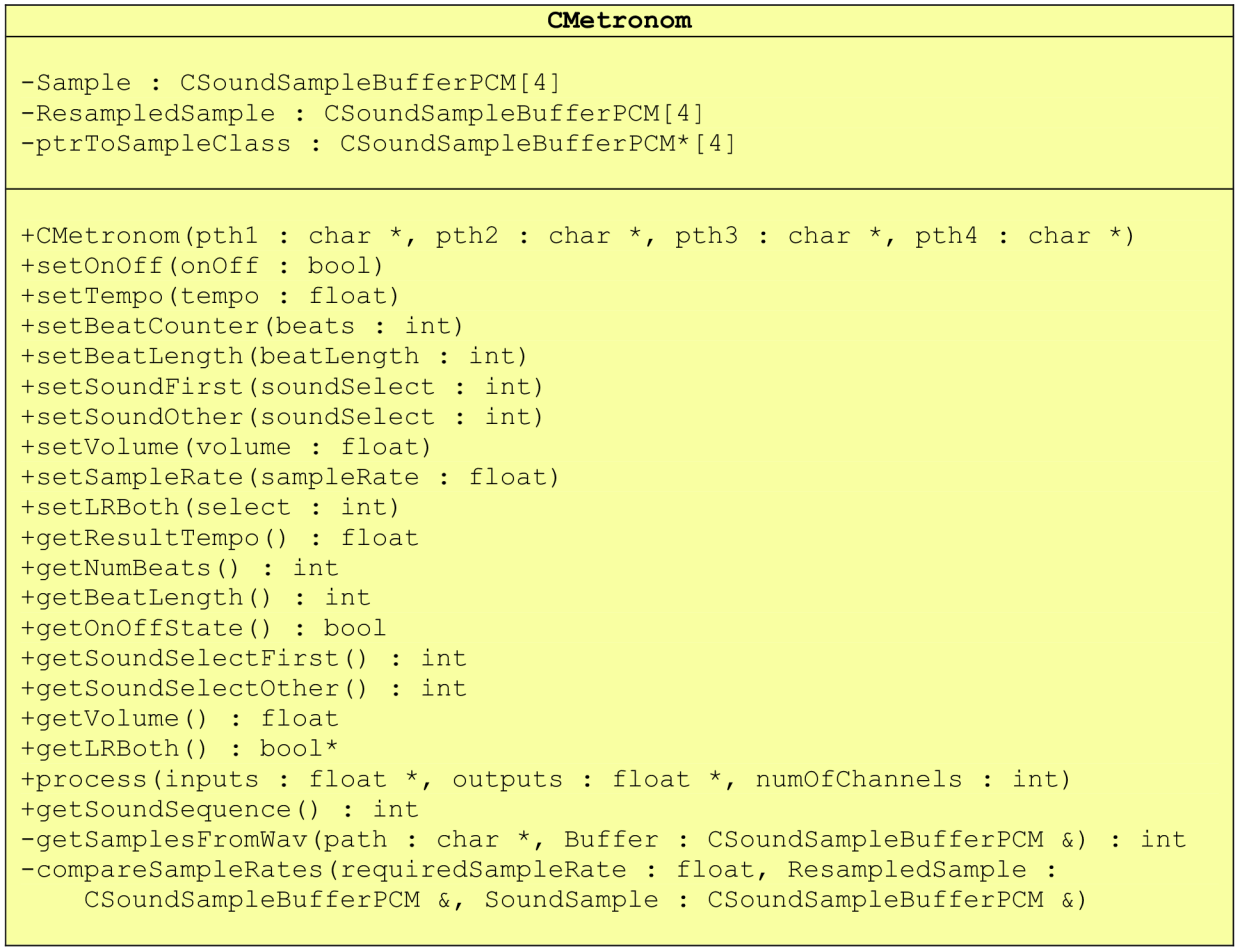
5 Implementace metronomu

Jako nejvhodnější se jeví zapouzdření metronomu do třídy, která obsahuje metodu vykonávající základní algoritmus a také metody, pomocí nichž bude možno nastavovat jednotlivé parametry metronomu, jako jsou tempo, počet dob v taktu atd. Taková třída bude nezávislá na použité technologii plug-in modulů.

Na Obr 5.1 je zobrazena deklarace třídy metronomu `CMetronom`, ve které jsou uvedeny nejdůležitější metody. Metody `setOnOff`, `setTempo`, `setBeatCounter`, `setBeatLength`, `setSoundFirst`, `setSoundOther`, `setVolume`, `setSampleRate` a `setLRBoth` slouží k nastavení jednotlivých parametrů metronomu a jsou volány jednak při inicializaci objektu třídy `CMetronom`, kdy dochází k nastavení parametrů metronomu na výchozí hodnoty, tak i například za běhu metronomu, kdy si uživatel přeje pozměnit některý z parametrů.

Metody `getResultTempo`, `getNumBeats`, `getBeatLength`, `getOnOffState`, `getSoundSelectFirst`, `getSoundSelectOther`, `getVolume` a `getLRBoth` slouží naopak ke zjištění hodnot jednotlivých parametrů, což bývá využíváno hostující aplikací, aby byla schopna zobrazit aktuální hodnoty parametrů.

Nejdůležitější metodou je metoda `process`. Této metodě je předáno pole `inputs` obsahující tolik vstupních vzorků, kolik je používaných kanálů. Tyto vzorky jsou zpracovány a uloženy do pole `outputs`. Metronom je vhodné realizovat ve formě efektu, tzn. k procházejícímu zdigitalizovanému zvukovému signálu (např. z hudebního nástroje) se budou přidávat vzorky zvuku metronomu. Tak vznikne výsledný zvukový signál, jenž se bude přehrávat výstupním zvukovým zařízením. Metoda `process` vstupní zvukové vzorky



Obr. 5. 1: Deklarace třídy metronomu

zpracovává tak, že k nim přičítá vzorky zvuku metronomu. Při jednom zavolání je pro každý kanál zpracován jeden vzorek. Algoritmus metody `process` je popsán v kapitole 2. 2. Další metodou je metoda `loadSample`, která slouží k načtení zvuku ze souboru typu *wav*, který bude metronom využívat. Metoda přebírá parametr `path`, což je cesta k danému souboru a `index`, který udává, na kterou pozici se načtený zvuk uloží (metronom umožňuje načtení čtyř zvuků). Po zavolání metoda vytvoří objekt třídy `GetSamplesFromWav` (popsánov kapitole 3. 3.), jehož konstruktoru bude předán řetězec `path`. Tento objekt pomocí svojí metody `readSamples` načte ze souboru vzorky zvukového signálu, které se uloží společně s údaji o počtu vzorků, počtu kanálů a vzorkovací frekvence do pole `Sample` typu `CSoundSampleBufferPCM` na pozici určenou parametrem `index`. Dále dojde zavoláním metody `compareSampleRates` k porovnání, zda-li je vzorkovací frekvence načteného zvuku shodná se vzorkovací frekvencí, se kterou pracuje dané výstupní zvukové zařízení. Jsou-li shodné, do pole `ptrToSampleClass` se uloží ukazatel na danou pozici v poli `Sample`, jež obsahuje načtený zvuk. Pokud vzorkovací frekvence shodné nejsou, je vytvořen objekt typu `Resample`, jenž si pomocí metody `loadInputBuffer` do svého vstupního zásobníku `InDataBuffer` zkopíruje z pole `Sample` načtený zvuk. Ten je následně převzorkován pomocí členské metody `resample` a uložen do výstupního zásobníku `OutDataBuffer`, který je

pomocí metody `getBufferOfSamples` uložen do pole `ResampledSample` na pozici určenou parametrem `index` a jeho adresa je uložena do pole `ptrToSampleClass` na příslušnou pozici. Pole ukazatelů `ptrToSampleClass` je využíváno metodou `process`, kde má význam ukazatele na zásobník se vzorky signálu vzorkovaného s požadovanou frekvencí.

6 Technologie VST

Technologie VST (Virtual Studio Technology)(viz [6]) , která se prvně objevila roku 1996, byla vyvinuta firmou Steinberg. Jedná se o technologii umožňující realizovat softwarový digitální hudební nástroj či efekt ve formě tzv. VST plug-in modulu. Pod tímto pojmem si lze představit přídavný softwarový modul, který lze použít jako rozšíření hostující audio aplikace podporující technologii VST. Tímto lze dosáhnout například obohacení hostující aplikace o nový algoritmus digitálního zpracování zvukového signálu.

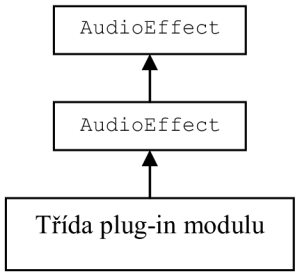
VST technologie poskytuje rozhraní, nezbytné pro integraci VST plug-in modulu do řetězce digitálního zpracování audio signálu. Hostující aplikace zde plní funkci zprostředkovatele, kdy VST plug-in modulu předává vstupní data (audio signál kódovaný pomocí PCM, jehož vzorky jsou normalizovány v rozmezí hodnot < -1;1 > nebo MIDI data). Tato data mohou být například načítána ze souboru uloženém na pevném disku, nebo mohou být přímo odebírána ze vstupu zvukové karty. Tato vstupní data jsou algoritmem implementovaným ve VST plug-in modulu zpracována a již zpracovaná výstupní data jsou předána opět hostující aplikaci, která je může poslat na výstup zvukové karty. Jednotlivé vzorky jsou reprezentovány jako čísla s plovoucí desetinnou čárkou buď o velikosti 32 bitů (jednoduchá přesnost) nebo o velikosti 64 bitů (dvojitá přesnost). Hostující aplikace může také poskytovat jednoduché grafické uživatelské rozhraní (GUI – graphical user interface) sloužící pro editaci hodnot parametrů zpracování audio signálu, nebo VST plug-in modul může obsahovat svoje vlastní GUI. Zdrojový kód VST plug-in modulu je nezávislý na platformě, ale forma, v jaké se vyskytuje, na platformě závisí. V operačním systému Microsoft Windows je realizován jako dynamická knihovna (DLL).

Dynamická knihovna DLL (dynamic link library) je knihovna (viz [7]), jež obsahuje programový kód a data, která mohou být použita více než jedním programem ve stejnou dobu, což umožňuje znovupoužití kódu a efektivnější využití operační paměti. Používáním dynamických knihoven lze dosáhnout modularizace daného programu do dílčích komponent. Spustitelná aplikace může například obsahovat jen základní program, ostatní moduly ve formě dynamických knihoven mohou být nahrávány až za běhu aplikace. Taková modularizace přináší výhodu ve formě rychlejšího spuštění aplikace, dané moduly jsou nahrávány pouze tehdy, je-li vyžadována jejich funkce, čímž je šetřena kapacita operační paměti. Je-li třeba aktualizovat nějakou funkci, stačí pouze přepsat daný modul, v němž je tato funkce implementována.

6.1 Vytvoření plug-in modulu v technologii VST

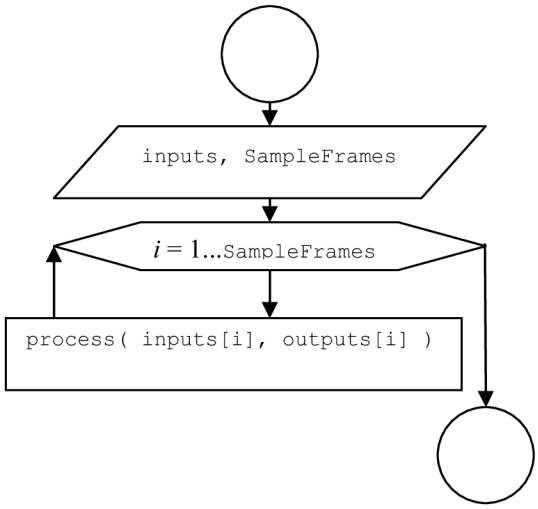
Pro vytvoření plug-in modulu v technologii VST je nutné mít k dispozici sadu vývojových nástrojů VST SDK. V případě technologie VST je to sada zdrojových kódů obsahujících definice struktur a funkcí definovaných standardem VST, jež tvoří rozhraní mezi VST plug-in modulem a VST hostující aplikací. Základním objektem ve VST SDK je třída s názvem `AudioEffect`, viz také [6], v níž jsou deklarovány metody tvořící rozhraní VST. Z této třídy je odvozena třída `AudioEffectX` rozšiřující možnosti třídy `AudioEffect`.

Třída VST plug-in modulu, který si přejeme vytvořit, je pak odvozena ze třídy `AudioEffectX`, viz Obr. 6. 1. Metody ve třídách `AudioEffect` a `AudioEffectX` jsou deklarovány jako virtuální, některé nejsou ani definovány. Proto je lze přetížit a ve třídě plug-in modulu dodefinovat, případně podle potřeby poupravit.



Obr. 6. 1. Odvození třídy plug-in modulu

Nejdůležitější metodou třídy `AudioEffect` je metoda `processReplacing`, v níž je implemetován algoritmus digitálního zpracování signálu. Tato metoda je deklarována jako čistě virtuální, a je tedy nutné ji definovat v odvozené třídě plug-in modulu. Metoda je deklarována jako `void processReplacing (float** inputs, float** outputs, VstInt32 sampleFrames)`, jako parametr je jí předáno pole `inputs` obsahující vzorky vstupního signálu, pole `outputs`, do něhož jsou ukládány vzorky zpracovaného signálu a počet vzorků `sampleFrames` jednoho kanálu, které jsou metodě processReplacing předány na zpracování. Algoritmus metronomu byl definován v metodě `process` třídy `CMetronom`, proto je ve třídě plug-in modulu deklarován objekt `Metronom` třídy `CMetronom`, jehož metody bude třída plug-in modulu využívat. V metodě `processReplacing` pak bude pro každý vstupní vzorek volána metoda `Process` objektu `Metronom`, jak je zobrazeno na Obr 6. 2. V každém průchodu cyklem je předán metodě `process` od každého kanálu jeden vzorek z dané pozice v poli `inputs`, a po zpracování je výstupní vzorek od každého kanálu uložen do pole `outputs` na příslušnou pozici.



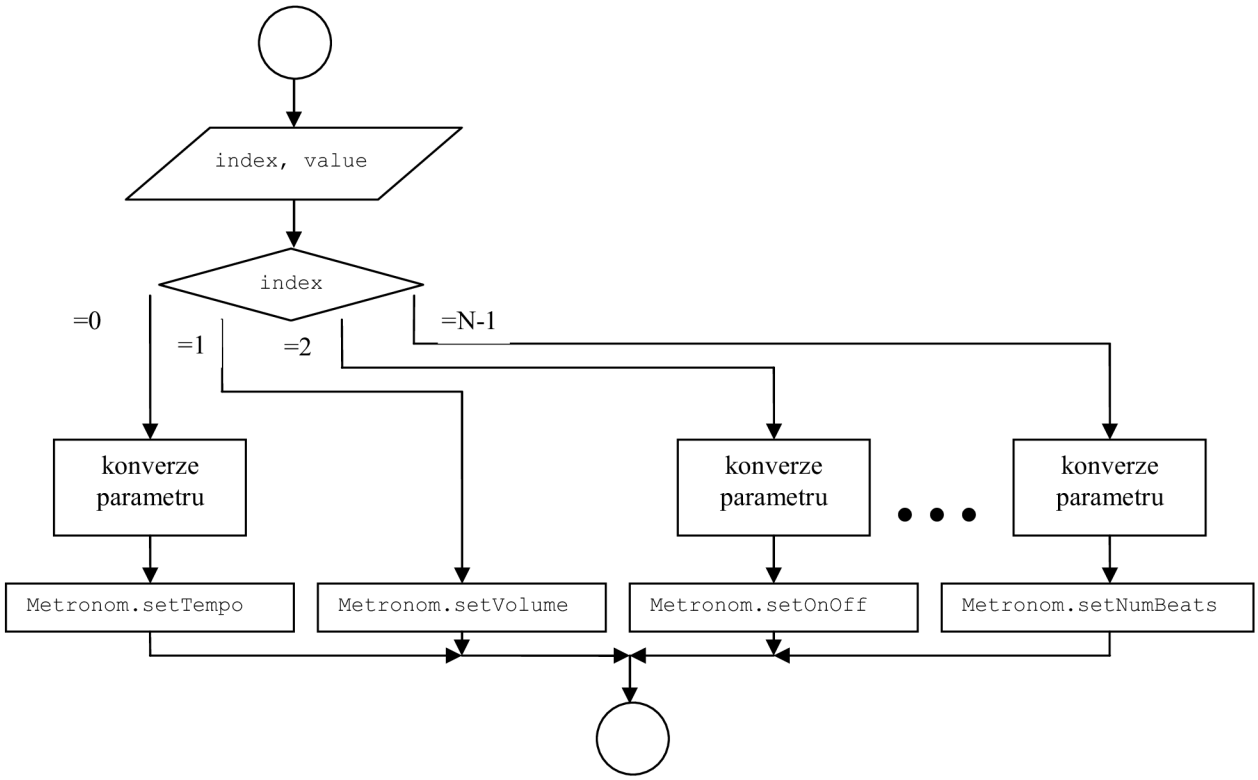
Obr. 6. 2: Definice metody `processReplacing`

Metoda `processReplacing` je volána průběžně hostující aplikací tak, aby zpracování vstupního signálu probíhalo v reálném čase, přičemž hostující aplikace si určuje počet vzorků `SampleFrames` na jeden kanál, jež metodě `processReplacing` předá na zpracování.

6.2 Nastavování parametrů

Pro nastavování parametrů algoritmu zapsaného v metodě `ProcessReplacing` je ve standardu VST deklarována metoda `void setParameter (VstInt32 index, float value)`. Změní-li uživatel v hostující aplikaci hodnotu některého z parametrů, je tato metoda zavolána. Je jí předán celočíselný identifikátor `index` určující parametr algoritmu, který uživatel právě změnil. Hodnota `value` je desetinné číslo v rozmezí < 0 ; -1 > a nese informaci o nastavení editovaného parametru. Definice metody `setParameter` pak obsahuje předpis, podle kterého je docíleno změny algoritmu.

V případě metronomu se po zavolání metody `setParameter` na základě identifikátoru parametru `index` (pomocí funkce `switch`) zavolá příslušná členská metoda objektu `Metronom` třídy `CMetronom`. Ta slouží k nastavení daného parametru (viz kapitola 5). Tato situace je znázorněna na Obr 6. 3.



Obr. 6. 3. Metoda `setParameter`

Informace o nastavení příslušného parametru je vždy reprezentována hodnotou desetinného čísla v rozmezí < 0 ; -1 > . Bude-li například zapínán a vypínán metronom, tak hodnota parametru v rozmezí < 0 ; 0.5> bude reprezentovat stav vypnuto a hodnota parametru (0.5 ; 1> stav zapnuto. Členská metoda třídy `CMetronom` však pro zapínání

a vypínání metronomu akceptuje parametry `true` nebo `false`. Je proto vhodné si nadefinovat pomocné funkce, jež budou tvořit mezivrstvu pro převod parametrů v různých vyjádřeních. Pro převod hodnot v rozmezí < 0 ; 1 > na hodnoty v požadovaném formátu byly deklarovány následující funkce:

- `bool convertSwitchParam (float onOff)`
- `float convertTempoParam (float tempo)`
- `int convertNumPositionsParam (float position, int numParams )`

Funkce `convertSwitchParam` slouží pro dvoustavové přepínaní. Je-li hodnota `onOff` větší než 0,5, funkce vrátí hodnotu `true`, v opačném případě hodnotu `false`. Je používána pro konverzi parametru předávánému metodě `setOnOff` třídy `CMetronom`. Funkce `convertTempoParam` slouží ke konverzi na hodnotu v BPM. Bude-li rozsah nastavitelných temp metronomu v rozmezí < 50 ; 250> BPM, pak převod proběhne na základě jednoduchého vztahu 50 + 250 . `onOff`. Tento převod je využíván pro metodu `setTempo`. Funkce `convertNumPositionParam` slouží k vícestavovému přepínání. Počet stavů je určen hodnotou `numParams`. Interval < 0 ; 1 > je tedy rozdělen na hodnotou `numParams` daný počet intervalů, podle hodnoty `position` funkce vrátí číslo intervalu, do kterého hodnota `position`n patří. Tato hodnota se pak použije jako parametr pro metody `setBeatCounter, setBeatLength, setSoundFirst, setSoundOther, setLRBoth.`

Dále třída plug-in modulu v technogii VST obsahuje definici metody `float getParameter (VstInt32 index)` mající opačný účel než metoda `setParameter`. Slouží tedy jako dotaz směrující k objektu plug-in modulu na aktuální hodnotu parametru určeného indentifikátorem `index`. Podle hodnoty identifikátoru `index` je pomocí funkce `switch` zavolána příslušná metoda objektu `Metronom` třídy `CMetronom`, obdobně jako tomu bylo u metody `setParameter`. Aktuální hodnota parametru je vrácena v rozmezí hodnot < 0; 1>.

S předáváním parametrů souvisí ještě některé metody třídy plug-in modulu definované ve standardu VST využívané především hostující aplikací, která pro případy, kdy VST plug-in modul nemá vytvořen svoje vlastní grafické uživatelské rozhraní pro ovládání parametrů, poskytuje uživateli svůj výchozí editor. Pro vyžádání informací, jež se ve výchozím editoru zobrazí, slouží následující metody:

- `void getParameterDisplay (index index, char* text)`
- `void getParameterName (index index, char* text)`
- `void getParameterLabel (index index, char* text)`

Předávání informací se děje prostřednictvím ukazatele `text` na pole znaků, do kterého je metodami uložen znakový řetězec, jenž se zobrazí uživateli. Toto se využívá pro zobrazení názvu parametru, zobrazení aktuálního nastavení parametru v požadované formě a pro zobrazení jednotek.

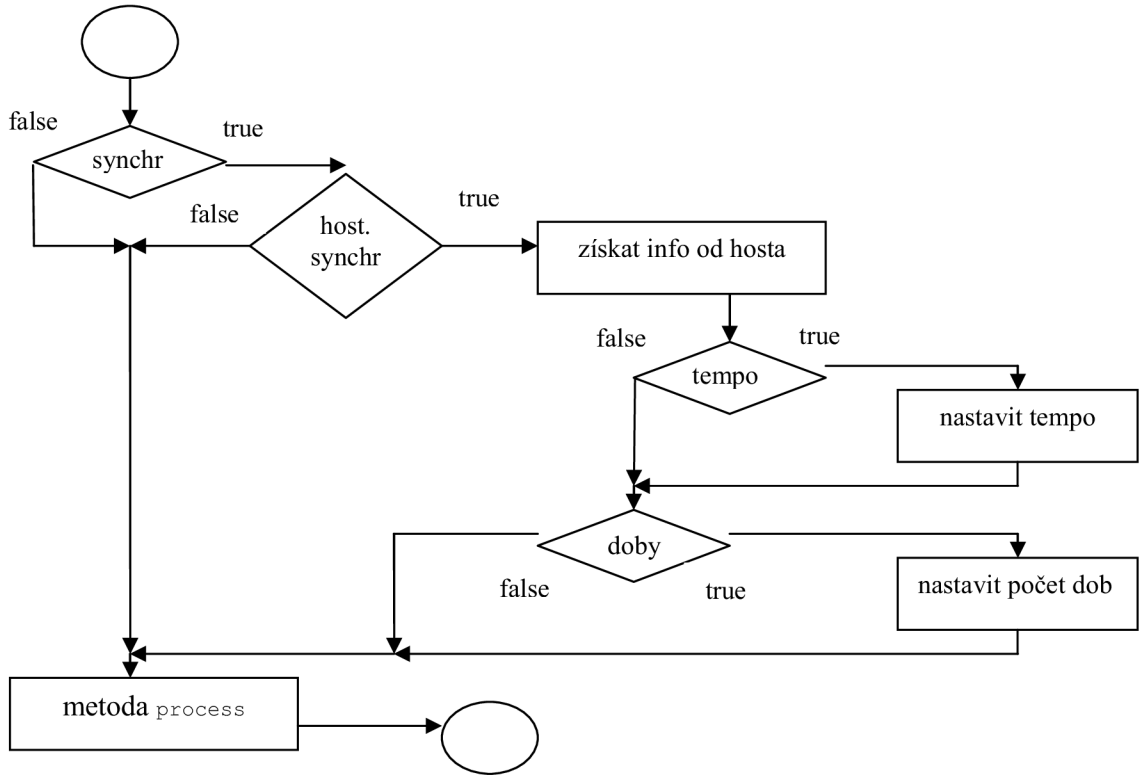
6.3 Synchronizace tempa a rytmu s hostující aplikací

Jedním z požadavků na vlastnosti metronomu je také schopnost přijímat informace o tempu a počtu dob v taktu, aby pak tyto informace použil pro nastavení svých parametrů. Tento přístup má výhodu v tom, že stačí upravovat nastavení tempa a rytmu v hostující aplikaci a tato nastavení se pak automaticky přenesou ke všem modulům. Například pokud hudebník pracuje s více plug-in moduly pro zpracování zvukového signálu, pro které je stejně jako u metronomu třeba zadat informaci o aktuálním tempu a rytmu. Podmínkou je však podpora této vlastnosti jak ze strany hostující aplikace, tak i ze strany jednotlivých modulů.

Ve standardu VST je pro účely časové synchronizace mezi hostující aplikací a plug-in moduly deklarována struktura `VstTimeInfo`, viz [6] . Jednotlivé položky této struktury slouží

k ukládání potřebných údajů pro synchronizaci. Mezi tyto položky patří i proměnné `tempo` a `timeSigNumerator`. Strukturu `VstTimeInfo` si může plug-in modul vyžádat od hostující aplikace a podle hodnot jejích jednotlivých proměnných upravit nastavení svých parametrů. Zda-li je tato možnost podporována ze strany hostující aplikace, lze zjistit pomocí metody `VstInt32 canHostDo (char* text)`. Jako parametr se jí předá textový řetězec `"sendVstTimeInfo"` a je-li vrácena hodnota jedna, pak je tato funkce hostující aplikace podporována. Pro předávání struktury `VstTimeInfo` slouží metoda `VstTimeInfo* getTimeInfo (VstInt32 filter)`. Jako parametr jsou jí předány příznaky určující, které informace ze struktury `VstTimeInfo` si plug-in modul přeje zjistit. Pro určení tempa a počtu dob v taktu to jsou příznaky `kVstTempoValid` a `kVstTimeSigValid`. Metoda `getTimeInfo` pak vrátí ukazatel na strukturu `VstTimeInfo`. Informace obsažené v této struktuře nemusí být platné a je tedy jejich platnost nutné otestovat porovnáním příznaků `kVstTempoValid` a `kVstTimeSigValid` s příznakem `flags` obsaženým ve struktuře `VstTimeInfo`. Jsou-li tyto informace platné, využije se jejich hodnota pro nastavení parametrů tempa a počtu dob v taktu.

Procedura synchronizace tempa a rytmu se zapíše do definice metody `processReplacing`, ve které bude prováděna, což zajistí synchronizaci metronomu za běhu zpracování. Je zobrazena ve vývojovém diagramu na obr. 6. 4.



Obr. 6. 4: Provedení synchronizace uvnitř metody `processReplacing`

Nejprve je zjištěno testováním první podmínky, zda-li je zapnuta synchronizace s hostující aplikací. Je-li synchronizace zapnuta, je v druhé podmínce za pomoci metody `canHostDo` ověřeno, podporuje-li hostující aplikace předávání informací pro synchronizaci prostřednictvím struktury `VstTimeInfo`. Pokud je toto hostující aplikací poskytováno, je pomocí metody `getTimeInfo` získána struktura `VstTimeInfo`. Jsou-li informace o tempu a rytmu uložené ve struktuře `VstTimeInfo` platné, což se ověří testováním příznaku `flags`, je

zavolána metoda `setParameter`. Té jsou předány hodnoty parametrů za pomoci funkcí `convertTempoParam` a `convertNumPositionsParam` ve správném tvaru. Po vykonání této akce je pak zavolána metoda `process` třídy `CMetronom` .

Za zmínku stojí, jakým způsobem jsou ve VST SDK realizovány příznaky. Jsou realizovány pomocí výčtových typů, viz na příkladu:

```
enum VstTimeInfoFlags
{
      kVstTempoValid            = 1 << 10,
      kVstTimeSigValid          = 1 << 13,
}
```

Číselná hodnota určitého příznaku je vytvořena bitovým posuvem jedničky o daný počet míst. Jednotlivým příznakům ve výčtovém typu tudíž odpovídají hodnoty rovny daných mocnin dvou, v binárním vyjádření je to jednička na dané pozici. Je-li nutné předat více příznaků jako parametr, je proveden jejich součet, což v binárním vyjádření bude odpovídat jedničkám na daných pozicích. Probíhá-li pak kontrola, zda jistá hodnota obsahuje nastavení určitého příznaku na jedničku či nulu, je realizována pomocí bitového operátoru `&`, kdy je prováděn vždy logický součin dvou bitů na odpovídajících si pozicích. Podle výsledku jedna či nula je určeno, zda-li je daný příznak nastaven či nikoliv.

7 Grafické uživatelské rozhraní

Je-li vytvořen VST plug-in modul, kdy jsou v jeho odvozené třídě nadefinovány pouze metody popsané v předešlém textu, tj. metoda `processReplacing`, metody pro správu parametrů a výměny informací o nich, tj. `setParameter`, `getParameter`, `getParameterName`, `getParameterDisplay`, `getParameterLabel` atd., bude možné jednotlivé parametry plug-in modulu měnit pouze tak, že hostující aplikace poskytne své vlastní rozhraní, pomocí nichž budou jednotlivé metody pro ovládání volány. Toto umožňuje se soustředit při vývoji plug-in modulu nejdříve na správnou funkci algoritmu pro zpracování signálu, jeho testování a ladění. Všechny hostující aplikace však funkci implicitního editoru pro správu parametrů plug-in modulu nemají. V takovém případě je možnost definice vlastního grafického uživatelského rozhraní - GUI , jež funkci spravování parametrů zajistí. Pro účel tvorby takového rozhraní jsou ve VST SDK zahrnuty zdrojové kódy obsahující objekty, ze kterých bude GUI složeno, viz [6] .

GUI je definováno jako třída, nazvaná například `SDEditor`, jejíž instance je vytvořena v konstruktoru objektu plug-in modulu. Je-li definováno GUI, je zahrnuto do objektu plug-in modulu, kde je reprezentováno ukazatelem `editor` deklarovaným ve třídě `AudioEffect`. Třída `SDEditor` je odvozena ze tříd `AEffGUIEditor` a `CControlListener` a tímto už tedy obsahuje základní funkčnost, kterou musí GUI mít. Do třídy `SDEditor` už pak stačí dodefinovat objekty jednotlivých grafických prvků, které jsou potřebné a metody pro jejich ovládání. Jak bude vypadat deklarace třídy `SDEditor` je ukázáno na Obr. 7. 1. V deklaraci se nachází deklarace ukazatelů na objekty ovládacích prvků, jež budou v GUI obsaženy. Nainicializování těchto prvků probíhá spuštěním metody `open`, která je ve třídě `SDEditor` také deklarována. Dále se zde nachází deklarace metod `setParameter` a `valueChanged` sloužící pro přenášení změn v nastavení prvků GUI do nastavení parametrů objektu plug-in modulu a naopak.

Za účelem vytvoření GUI pro modul metronomu je vhodné si nejprve navrhnout, jak by mělo GUI vypadat, jaké typy ovládacích prvků by mělo obsahovat a také jejich rozmístění. Na Obr 7. 2. je návrh GUI metronomu.

SDEditor

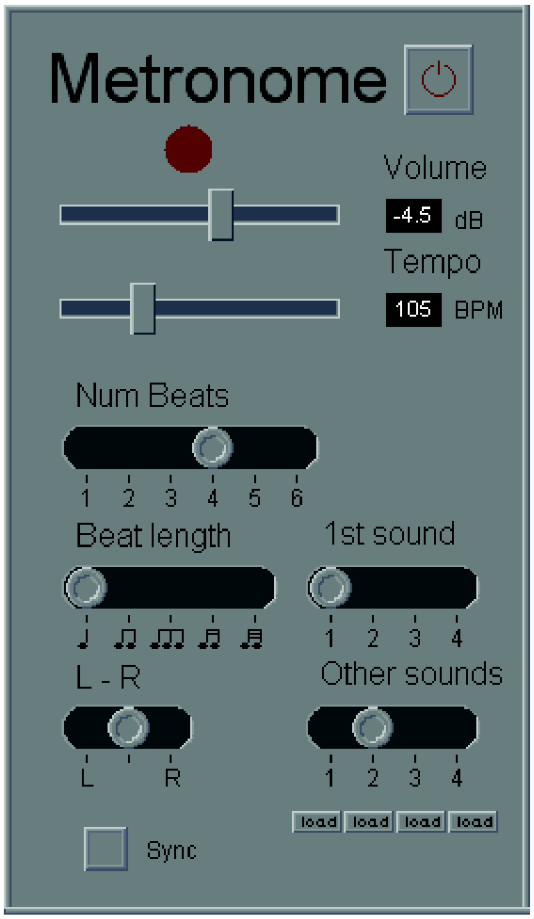
```
-hBackground : CBitmap*
-LRSwitch : CHorizontalSwitch*
-volumeFader : CHorizontalSlider*
-tempoFader : CHorizontalSlider*
-numBeatsSwitch : CHorizontalSwitch*
-beatLenSwicth : CHorizontalSwitch*
-soundSelect1stSwitch : CHorizontalSwitch*
-soundSelectOthSwitch : CHorizontalSwitch*
-volumeDisplay : CParamDisplay*
-tempoDisplay : CParamDisplay*
-LED : CMovieBitmap*
-onOffButton : COnOffButton*
-syncButton : COnOffButton*
-openButton1 : COnOffButton*
-openButton2 : COnOffButton*
-openButton3 : COnOffButton*
-openButton4 : COnOffButton*
```

```
+SDEditor(effect : AudioEffect *)
+open(ptr : void *)
+close()
+setParameter(index : VstInt32, value : float)
+valueChanged(context : CDrawContext *, control : CControl *)
+openDialogForSound(indexSound : int)
```

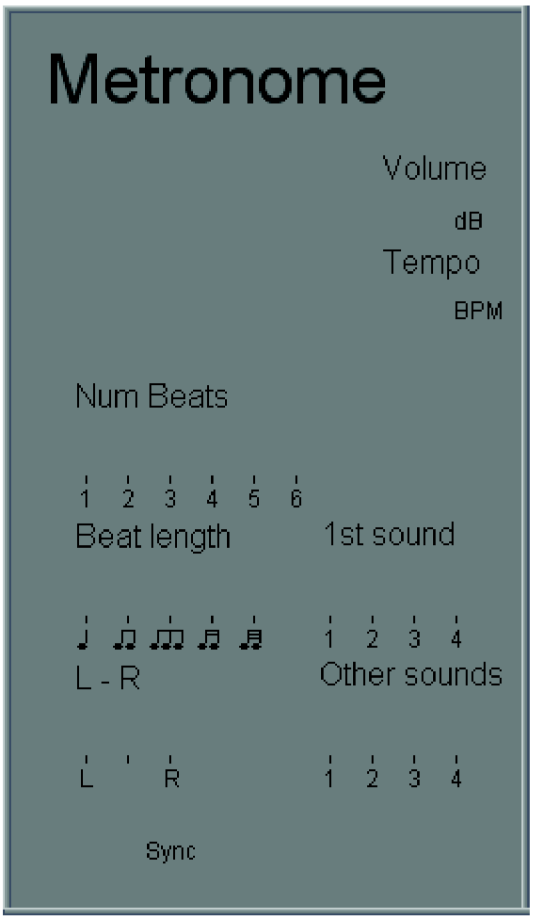
Obr. 7. 1: Deklarace třídy grafického uživatelského rozhraní

Obsahuje několik dvoustavových tlačítek. Je to tlačítko pro zapnutí/vypnutí metronomu, tlačítko pro zapnutí/vypnutí synchronizace a čtyři tlačítka sloužící pro vyvolání dialogu pro výběr souboru. Dvoustavovým tlačítkům ve VST SDK odpovídá třída `COnOffButton`, ukazatele na objekty tohoto typu jsou zahrnuty v deklaraci třídy `SDEditor`. V návrhu GUI jsou dále dva spojité horizontální posuvníky sloužící pro nastavování hlasitosti a tempa metronomu. Ve třídě `SDEditor` jim odpovídají ukazatele na objekty typu `CHorizontalSlider`. Pro zobrazení aktuální hodnoty nastaveného parametru pomocí vertikálních posuvníků jsou vedle nich dva displeje typu `CParamDisplay`. Pro volbu počtu dob v taktu, pro délky jedné noty, pro určení zvuku pro první a ostatní doby taktu a pro přepínání kanálů se využijí horizontální vícepolohové přepínače typu `CHorizontalSwitch`. Zvuk metronomu může být doprovázen blikáním prvku imitujícího LED diodu, v editoru bude tedy použit objekt typu `CMovieBitmap`.

Je nutné si připravit rastrové obrázky, ze kterých bude výsledné GUI složeno. Hlavní složku představuje pozadí, jehož rozměry určují rozměr vlastního GUI. Obrázek bude tedy téměř shodný s návrhem , nebude ovšem obsahovat jednotlivé ovládací prvky, viz Obr. 7. 3. Pro reprezentaci rastrových obrázků slouží ve VST GUI třída `CBitmap`. Umožňuje na základě číselného identifikátoru rastrového obrázku tento obrázek načíst a poskytovat metody pro práci s ním, například metody pro zjištění jeho sířky či výšky. Ukazatel na objekt `CBitmap`. obsahující načtený obrázek pozadí je také deklarován ve třídě `SDEditor`.

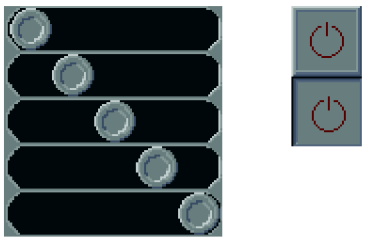


Obr. 7. 2: Návrh grafického uživatelského rozhraní



Obr. 7. 3: Pozadí pro grafické uživatelské rozhraní

Formu rastrových obrázků pro ovládací prvky je třeba si připravit v závislosti na typu ovládacího prvku. Pro dvoustavová tlačítka, vícestavové přepínače a LED diodu je nutno složit rastrový obrázek z jednotlivých stavů, kterých může daný prvek nabývat, jak je znázorněno na obrázku 7. 4. Podle toho, kolik stavů ovládací prvek obsahuje, je obrázek svisle rozdělen na daný počet dílčích obrázků. Na základě aktuální hodnoty parametru je zobrazen odpovídající dílčí obrázek. Pro spojité ovládací prvky, v případě metronomu je to spojitý horizontální posuvník, se bude výsledný prvek skládat z obrázku představujícího nepohyblivou dráhu a z obrázku představující pohyblivý jezdec, viz Obr. 7. 5.



Obr. 7. 4: Rastrové obrázky pro dvoustavové a vícestavové ovládací prvky

Obr. 7. 5: Rastrové obrázky pro spojitý horizontální posuvník

Takto připravené obrázky jsou uvnitř metody `open` načteny do objektů typu `CBitmap` a v této formě jsou předány objektům ovládacích prvků při jejich inicializaci. Zároveň se při inicializaci prvků určí pozice, kterou budou mít na ploše pozadí. Metoda `open` musí také obsahovat inicializaci objektů zobrazovacích displejů. Tomu odpovídá určení jejich rozměrů, jejich pozice na ploše pozadí, barvy pozadí a písma. Poloha či stav ovládacího prvku, který je zobrazen, závisí na hodnotě parametru, jenž byl objektu ovládacího prvku předán. K předávání hodnoty objektům ovládacích prvků slouží jejich členská metoda `void setValue (float val)` přijímající parametr `val` ve formě desetinného čísla v intervalu < 0 ; -1 >. Interval je pak rozdělen na stejný počet dílčích intervalů, jako je počet dílčích obrázků daného ovládacího prvku. Podle toho, do kterého intervalu získaná hodnota náleží, se zobrazí odpovídající dílčí obrázek. Stejně tak je displeji předávána hodnota parametru v rozmezí < 0 ; -1 >, a není-li určeno jinak, tato hodnota se v displeji objeví. Zobrazování hodnoty hlasitosti a tempa v tomto tvaru však není pro uživatele přívětivá, a proto objekt displeje obsahuje metodu `void setStringConvert (void (*convert) (float value, char *string))`. Tato metoda přebírá ukazatel na funkci prototypu `void funkce (float value, char *string)`. Proto je třeba si vytvořit funkci daného prototypu, jejímž účelem bude převést předanou hodnotu value v rozmezí < 0 ; -1 > na hodnotu v požadovaném formátu. Použitím definovaného vztahu se tato hodnota převede na textový řetězec, který se uloží do textového pole `string`. Je-li ukazatel na takovou funkci objektu displeje předán, objekt pak vždy při zobrazování hodnoty parametru provede s pomocí určené funkce konverzi do požadovaného formátu.

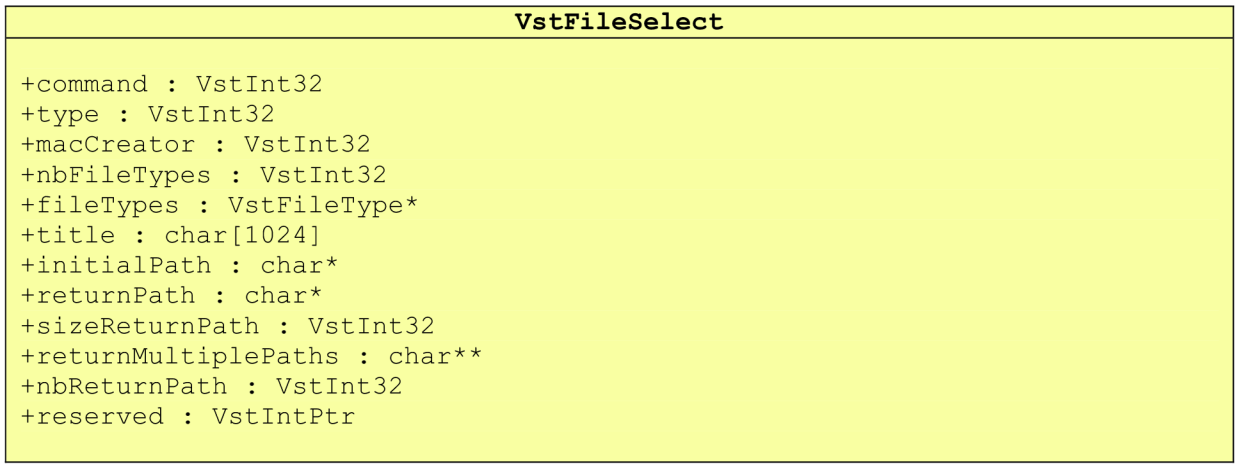
Je-li například v implicitním editoru poskytovaném hostující aplikací změněna hodnota některého z parametrů, je zavolána metoda `setParameter` objektu plug-in modulu. Ta provede příslušné změny v objektu třídy `CMetronom`. Zároveň, je-li pro plug-in modul definováno GUI, je také uvnitř metody `setParameter` zavolána metoda `setParameter` objektu třídy `SDEditor`. Ta na základě číselného identifikátoru `index` zvolí odpovídající hodnotu, a stav ovládacího prvku nastaví pomocí metody `setValue`. Tímto způsobem se změna parametru provedená v implicitním editoru projeví i ve vlastním GUI plug-in modulu.

Je-li naopak změněna hodnota některého z parametrů ve vlastním GUI plug-in modulu, je zavolána metoda `valueChanged`. Metoda zjistí, o který parametr se jedná, a zavolá metodu `setParameter` objektu plug-in modulu, který nastavení přenese do objektu třídy `CMetronom` a také se tyto změny projeví i v implicitním grafickém editoru.

7.1 Dialogové okno pro volbu souborů

Požadavkem na metronom je, aby si uživatel mohl zvolit zvuky, kterými bude metronom udávat rytmus. Je tudíž nutné implementovat do metronomu mechanismus, který uživateli otevře dialogové okno umožňující procházení adresářové strukury pevného disku a volbu souboru. Tímto postupem je možno získat absolutní cestu k souboru, jež bude předána jako parametr objektu třídy `GetSamplesFromWav`.

Ve VST SDK je pro účely volby souboru prostřednictvím dialogového okna k dispozici třída `CFileSelector`. Tato třída obsahuje metodu `long run (VstFileSelect *vstFileSelect)`, jíž je jako parametr předána struktura typu `VstFileSelect` také deklarovaná ve VST SDK. Před zavoláním metody `run` je nutné si připravit strukturu typu `VstFileSelect` a její proměnné naplnit informacemi, podle kterých bude nastaveno dialogové okno následně spuštěné metodou `run`. Na Obr. 7. 6 je deklarace struktury `VstFileSelect`.



Obr. 7. 6: Informace předávané metodě `run`

Mezi její důležité položky, které je nutno nainicializovat, patří položka `command`. Slouží k určení funkce dialogového okna. V případě metronomu bude obsahovat identifikátor `kVstFileLoad` výčtového typu určující, že dialogové okno bude sloužit pro výběr jednoho souboru pro nahrátí. Položka `nbFileTypes` udává, kolik typů souborů bude dialogové okno načítat. Jelikož si přejeme pouze výběr souborů typu *wav*, bude obsahovat hodnotu 1. Položka `fileTypes` obsahuje ukazatel na strukturu typu `VstFileType` obsahující řetězce, v nichž jsou uloženy název a přípona požadovaného typu souboru. Do znakového pole `title` se uloží věta, jež se zobrazí na liště otevřeného dialogového okna. Ukazatel `returnPath` bude ukazovat na znakové pole, jenž bude alokováno v paměti a jenž bude sloužit pro uložení absolutní cesty k uživatelem zvolenému souboru. Položka `sizeReturnPath` pak udává, jak je velké pole, na něhož ukazatel `returnPath` ukazuje.

Je-li v editoru stisknuto tlačítko pro volbu zvuku, bude vytvořena struktura typu `VstFileSelect`, do které ze zapíší patřičné údaje, a tato struktura se předá metodě `run` objektu typu `CFileSelector`, který zprostředkuje otevření dialogového okna. Je-li uživatelem zvolen soubor, je nahlédnuto do položky `returnPath` struktury `VstFileSelect`, ve které je zapsaná cesta k souboru, kterou si uživatel zvolil. Poté je zavolána metoda `loadSample` objektu třídy `CMetronom`, která zajistí načtení vzorků ze souboru na disku. Je-li třeba, je zvukový signál převzorkováním přizpůsoben vzorkovací frekvenci výstupního zařízení. Po dokončení metody `run` je ještě nutné zavolat metodu `closeFileSelector` definovanou ve třídě `AudioEffectX`, která slouží pro uvolnění paměti pro znakové pole `returnPath` alokované hostující aplikací. Tímto postupem je zajištěno načtení zvuku ze souboru.

7.2 Nastavení LED diody

Grafické uživatelské rozhraní obsahuje také prvek imitující LED diodu. Aby dioda blikala zároveň se zvukem, který metronom vydává, je nutno ji ovládat uvnitř metody `processReplacing` objektu plug-in modulu. Dioda bude svítit tehdy, budou-li do výstupního zásobníku ukládány nenulové vzorky, tj. bude-li metronom vydávat zvuk. V době, kdy budou do výstupního zásobníku ukládány nulové vzorky, tj. metronom nebude vydávat zvuk, bude dioda zhasnuta. Zda jsou v daném průchodu smyčkou uvnitř metody `processReplacing` do výstupního zásobníku ukládány nulové či nenulové vzorky, zjistíme zavoláním metody `getSoundSequence` objektu třídy `CMetronom`. Na základě vrácené hodnoty metodou

`getSoundSequence` se zavolá metoda `setParameter` objektu editoru, která zavolá metodu `setValue` objektu LED diody a nastaví ji na hodnotu buď jedna, nebo nula, tedy stav rozsvícena, nebo zhasnuta.

8 Závěr

Na závěr je uvedeno zhodnocení bakalářské práce. Kapitoly představují řešení jednotlivých problémů implementace digitálního metronomu.

Prvotním úkolem bylo vytvořit algoritmus metronomu, který byl zapouzdřen do třídy `CMetronom` napsané v jazyce C++, jejíž nejdůležitější metodou je metoda `process`. Ta má v sobě zapsán algoritmus zajišťující, aby se ke každému vstupnímu vzorku přičetl vždy takový vzorek zvuku, který obohatí výslednou zvukovou sekvenci o periodický zvuk metronomu udávajícího tempo. Nastavování parametrů metronomu, jako je například tempo, hlasitost atd., je zajišťováno pomocí metod k tomu určených.

K zpřístupnění možnosti volby zvuku jeho načtením ze souboru typu *wav* byla implementována pomocná třída `GetSamplesFromWav`, která je schopna data ze souboru typu wav načíst a převést je do požadovaného formátu. Takto připravená data jsou k dispozici pro využívání metronomem. V případě, že je nutné načtený zvukový signál převzorkovat, byla implementována další pomocná třída `Resample`, jež tuto funkci provádí. Uvedené pomocné třídy jsou využívány hlavní třídou metronomu `CMetronom`.

Takto vytvořený metronom byl implementován v technologii VST ve formě plug-in modulu. Po dosažení základní funkčnosti metronomu bylo pro VST plug-in modul vytvořeno grafické uživatelské rozhraní. Obsahuje ovládací prvky potřebné pro jednoduché nastavení všech parametrů metronomu. Zvuk vydávaný metronomem je indikován blikáním prvku imitujícího LED diodu. Volba zvukového souboru pro načtení je uskutečněna pomocí dialogového okna.

Takto realizovaný metronom ve formě VST plug-in modulu je funkčním přípravkem použitelným v hostující aplikaci podporující technologii VST.

Literatura

[1] SCHIMMEL, J. *Formáty zvukových souborů na PC*. *Elektrorevue* [online]. 2001, č. 7 Dostupný z WWW: <http://www.elektrorevue.cz/clanky/01007/index.html>. ISSN 1213-1539.

[2] *Resource Interchange File Format Services* [online]. c2008 . Dostupný z WWW: <http://msdn.microsoft.com/cs-cz/library/ms713231>.

[3] *Multimedia Functions* [online]. c2008. Dostupný z WWW: <http://msdn.microsoft.com/cs-cz/library/ms712636>.

[4] SMÉKAL, Z. *Číslicové zpracování signálů.* Elektronická skripta, VUT v Brně, 2006

[5] SMÉKAL, Z., SYSEL, P. *Číslicové filtry.* Elektronická skripta, VUT v Brně, 2004

[6] *VSTSDK 2.4 Documentation* [online]. Steinberg Media Technologies, c2006 Dostupný z WWW: <http://www.steinberg.net/de/company/3rd_party_developer.html>.

[7] *What is a DLL?*. Microsoft Help and Support [online]. 2007, no. 815065 Dostupný z WWW: <http://support.microsoft.com/kb/815065>.

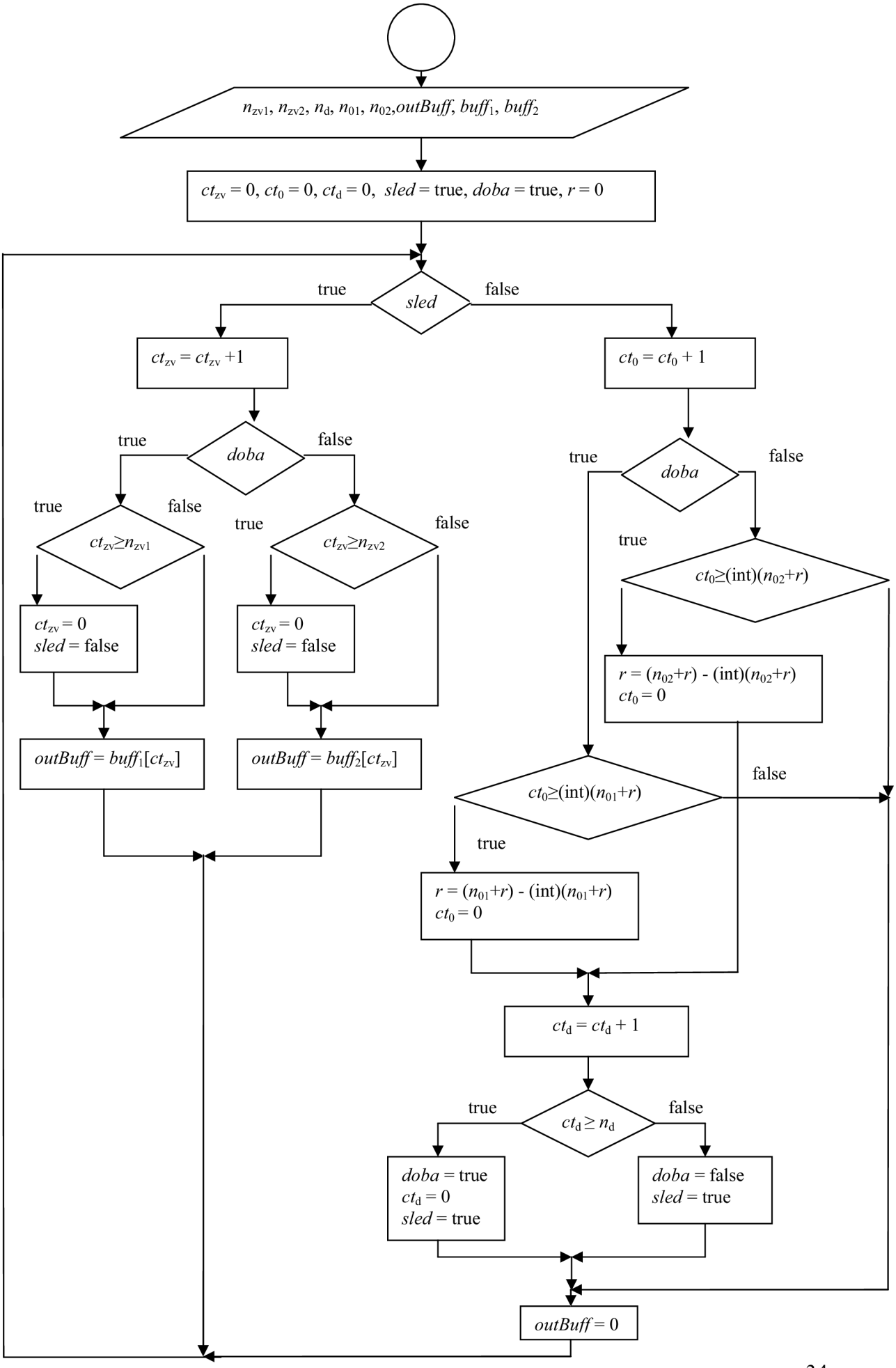
Seznam použitých zkratek

BPM	Beats Per Minute - *počet úderů za minutu*
DLL	Dynamic Link Library - *dynamicky připojovaná knihovna*
GUI	Graphical User Interface - *grafické uživatelské rozhraní*
IIR	Infinite Impulse Response - *nekonečná impulsní odezva*
NSN	*nejmenší společný násobek*
PCM	Pulse-Code Modulation - *pulsně-kódová modulace*
SDK	Software Development Kit - *sada pro vývoj softwaru*

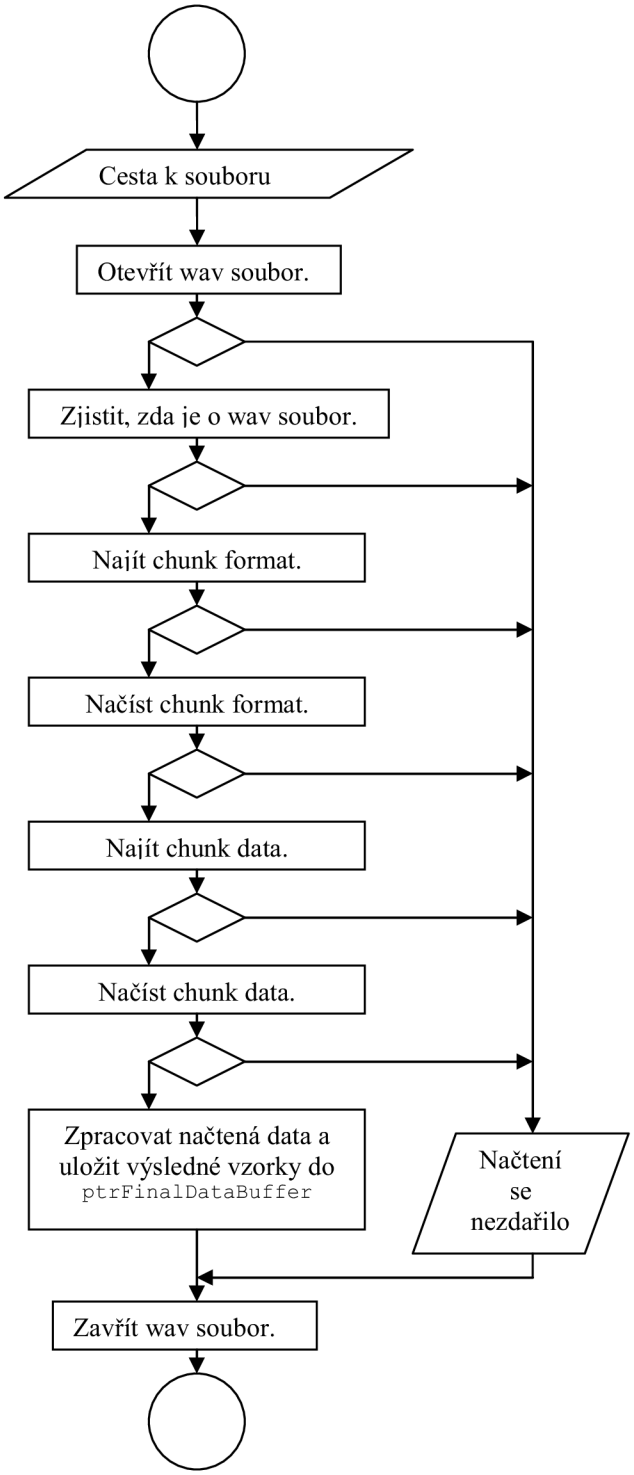
Seznam příloh

Příloha A:	Vývojový diagram zobrazující algoritmus metronomu
Příloha B:	Algoritmus metody `readSamples`

Příloha A: Vývojový diagram zobrazující algoritmus metronomu



Příloha B: Algoritmus metody `readSamples`



Obsah přiloženého CD

- **Metronom** - adresář obsahuje soubory se zdrojovými kódy, tvořícími vývojovou sadu VST, a zdrojovými kódy metronomu.
  - **editor -** v adresáři jsou soubory *editor.h* a *editor.cpp* se zdrojovými kódy grafického editoru pro ovládání metronomu, v podadresáři **resources** jsou obsaženy rastrové obrázky tvořící ovládací prvky .
  - **vstgui** - obsahuje soubory zdrojových kódů využívaných pro tvorbu GUI.
  - **source -** obsahuje tyto soubory:
    - *Metronom.h a Metronom.cpp* - zdrojové kódy třídy, v níž je implementován algoritmus metronomu.
    - *CSoundSampleBufferPCM.h a CSoundSampleBufferPCM.cpp* - zdrojové kódy pomocné třídy sloužící pro manipulaci se zásobníky vzorků.
    - *GetSamplesFromWav*.h a *GetSamplesFromWav.cpp* - zdrojové kódy pomocné třídy pro načítání souborů typu *wav*.
    - *resam.h* a *resam.cpp* - zdrojové kódy pomocné třídy sloužící pro převzorkování signálu.
    - *vst_temp.h*, *vst_temp.cpp*, *vst_temp_defs.h* a *user_variables.h* - zdrojové kódy šablony VST usnadňující implementaci VST plug-in modulu.
  - **vst2.x** - obsahuje soubory se zdrojovými kódy potřebnými pro tvorbu VST plug-in modulu.
  - **win** - obsahuje soubory projektu vytvořeného ve Visual Studiu. V tomto projektu je implementován metronom v technogii VST. V podadresáři **Debug** se nachází již zkompilovaný projekt metronomu ve formě souboru *metronom.dll*.

- **Matlab - IIR filtr -** v adresáři se nachází soubor *IIR.m* obsahující skript, jenž simuluje převzorkování zvukového signálu. Během skriptu jsou nejprve načteny vzorky zvukového signálu ze souboru typu *wav*. Podle požadované cílové vzorkovací frekvence a vzorkovací frekvence načteného zvuku jsou spočítány koeficienty dvou filtrů typu IIR 4. řádu. Načtený zvukový signál je převzorkován a na závěr jsou pro srovnání přehrány oba zvuky, jak původní, tak i převzorkovaný. V adresáři je také k dispozici několik souborů typu *wav* obsahující krátké zvuky, které je možné pro simulaci převzorkování použít.

- **Bakalářská práce -** text bakalářské práce v elektronické formě