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

IMPLEMENTACE ZVUKOVÉHO PROCESORU FIR S MINIMÁLNÍM PROCESNÍM ZPOŽDĚNÍM

IMPLEMETATION OF FIR SOUND PROCESSOR WITH MINIMAL PROCESSING LATENCY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Tomáš Brhel**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Miroslav Balík, Ph.D.**
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Audio inženýrství**
Ústav telekomunikací

Student: Tomáš Brhel ***ID:*** 164590
Ročník: 3 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Implementace zvukového procesoru FIR s minimálním procesním zpožděním

POKYNY PRO VYPRACOVÁNÍ:

Metodu rychlé konvoluce s přičtením přesahu s minimálním procesním zpožděním simulujte v Matlabu jako univerzální FIR procesor pro zpracování velmi dlouhých posloupností vzorků. Metodu pak realizujte jako zásuvný modul VST pro jeho použití v reálném čase a s uživatelským rozhraním, které umožní nastavení potřebných parametrů metody. Rozhraní bude umožňovat použití různých impulsových odezev pro jednotlivé kanály univerzálního FIR procesoru.

DOPORUČENÁ LITERATURA:

[1] M. Balík, Číslicové zpracování akustických signálů. Skriptum, FEKT VUT v Brně, 2014.

[2] O. Julius, III. Overlap-Add STFT Processing. Center for Computer Research in Music and Acoustics (CCRMA) Department of Music, Stanford University, 2014 [online]. Dostupné z https://ccrma.stanford.edu/~jos/sasp/index.html

[3] W. G. Gardner, Efficient convolution without input/output delay. J. Audio Eng. Soc. 43 (3), 127-136, 1995, ISSN 1549-4950

Termín zadání: 1.2.2016 ***Termín odevzdání:*** 1.6.2016

Vedoucí práce: Ing. Miroslav Balík, Ph.D.
Konzultant bakalářské práce:

doc. Ing. Jiří Mišurec, CSc., *předseda oborové rady*

ABSTRAKT

Tato práce se zabývá návrhem univerzálního FIR procesoru pro zpracování velmi dlouhých posloupností. Práce dále řeší optimalizaci pro souběžné zpracování dvoukanálové vstupní posloupnosti a potlačení procesního zpoždění. Realizace algoritmů využívá prostředí Matlab a frameworku JUCE.

KLÍČOVÁ SLOVA

FIR systém, impulsová odezva, kmitočtová charakteristika, FFT, VST.

ABSTRACT

This thesis deals with design of the universal FIR processor for long signals processing. Subsequently, it focuses on processing of the two-channel input signal and the reduction of the process delay. Matlab and framework JUCE are used as an interactive environment for algorithm development.

KEYWORDS

FIR system, impulse response, frequency response, FFT, VST.

BRHEL, Tomáš *Implementace zvukového procesoru FIR s minimáním procesním zpožděním*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2016. 60 s. Vedoucí práce byl Ing. Miroslav Balík, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Implementace zvukového procesoru FIR s minimáním procesním zpožděním“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Miroslavu Balíkovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

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

Brno . . . . . .

. . . . . .

podpis autora

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Tato práce se věnuje návrhu univerzálního zvukového FIR procesoru. Takový procesor je schopen realizovat systém definovaný svou impulsovou odezvou. Jako příklad může sloužit simulace kytarových reproboxů, poslechových prostor nebo filtrů. Pomocí dobře změřené impulsové odezvy, lze tak simulovat hru přes kytarové aparáty *MESA Bogie* nebo zpěv ve studiu *Abbey Road*.

Při realizaci systémů s velmi dlouhou impulsovou odezvou však dochází k výskytu vysokého procesního zpoždění, které je nežádoucí např. při zpracování hudebních signálů během živého hraní. Jedna z metod, jak procesní zpoždění potlačit je popsána v teoretické části práce a následně je implementována.

Procesor by dále měl být navržen pro souběžnou filtraci dvou signálů. Pomocí algoritmu dvoukanálové rychlé konvoluce s přičtením přesahu lze zrealizovat procesor zpracovávající dva kanály současně s minimálním přírůstkem výpočetní náročnosti. Návrhy algoritmu jsou nejdříve realizovány a odzkoušeny v prostředí Matlab.

Možnosti implementace zásuvných modulů VST se za poslední léta podstatně rozšířily. Je potřeba prozkoumat dostupné možnosti a následně vybrat optimální metodu pro implementaci zásuvných modulů. V práci budou shrnuty kroky pro přípravu vývojových prostředí, která dále budou porovnána.

Ve zvoleném vývojovém prostředí následně proběhne realizace VST zásuvného modulu v jazyce C++, založeného na algoritmech popsaných v této práci. Výsledný VST modul bude také testován s běžně dostupnými odezvami a vyzkouší se při hře na elektrickou kytaru.

1 DISKRÉTNÍ SYSTÉMY S KONEČNOU IMPULSOVOU ODEZVOU A JEJICH REALIZACE

Diskrétní systémy rozdělujeme podle délky impulsové odezvy na systémy s nekonečnou impulsovou odezvou a systémy s konečnou impulsovou odezvou. Oba systémy jsou obecně popsány přenosovou funkcí

$$\boldsymbol{H(z)} = \frac{\boldsymbol{P(z)}}{\boldsymbol{Q(z)}}. \tag{1.1}$$

Následující text vychází z [1]. Systémy s nekonečnou impulsovou odezvou IIR (*Infinite Impulse Response*) jsou definované racionální lomenou přenosovu funkcí. Pro FIR systémy (*Finite Impulse Response*) je však charakteristická přenosová funkce, kde je jmenovatel $\boldsymbol{Q(z)}$ roven 1. Rovnice (1.2) je převzata z [1] a popisuje přenosovou funkci obecného FIR systému.

$$\boldsymbol{H(z)} = \boldsymbol{P(z)} = \sum_{i=0}^{N} b_i \boldsymbol{z}^{-i} = b_0 + b_1 \boldsymbol{z}^{-1} + b_2 \boldsymbol{z}^{-2} + \cdots + b_N \boldsymbol{z}^{-N} = k \prod_{i=1}^{N} (\boldsymbol{z} - \boldsymbol{n}_i). \tag{1.2}$$

Tento systém je definován koeficienty $b_0, b_1, b_2 \ldots b_N$ polynomu $\boldsymbol{P(z)}$. Řešením charakteristické rovnice jsou nuly $\boldsymbol{n}_i$. Následující rovnice (1.3) je převzata z [1]. Impulsovou odezvu $h(n)$ definují koeficienty polynomu $\boldsymbol{P(z)}$

$$h(n) = [b_0 \quad b_1 \quad b_2 \quad \ldots \quad b_N]_M, \tag{1.3}$$

kde M označuje celkový počet koeficientů systému nebo délku jeho impulsové odezvy v počtu vzorků.

Podle [1] a [3] mají FIR systémy tyto vlastnosti:

- Můžou vykazovat lineární fázovou kmitočtovou charakteristiku.
- Systémy jsou vždy stabilní.
- Při implementaci systému typu FIR je nezbytný vyšší řád systému FIR pro dosažení podobných vlastností, které vykazuje systém IIR podstatně nižšího řádu.
- Pro realizaci FIR systému v časové oblasti je charakteristická vysoká výpočetní náročnost, která narůstá exponenciálně s řádem filtru.
- Vysoký počet koeficientů v čitateli přenosové funkce způsobuje vyšší nároky na paměť.
- Realizace FIR systému v kmitočtové oblasti je doprovázena procesním zpožděním.

1.1 Možnosti implementace v časové a kmitočtové oblasti

Zpracování obecného signálu systémem FIR je uskutečněno pomocí konvoluce. Výpočet konvoluce může proběhnout buďto přímo v časové oblasti nebo pomocí diskrétní Fourierovy transformace (DFT) v kmitočtové oblasti. Následující diferenční rovnice převzata z [1], obecně popisuje zpracování vstupní posloupnosti $x(n)$ systémem s impulsovou odezvou $h(n)$

$$y(n) = \sum_{m=-\infty}^{\infty} h(m)x(n-m) \quad \leftrightarrow \quad y(n) = h(n) * x(n), \tag{1.4}$$

kde $*$ značí přímou konvoluci. Pro impulsovou odezvu krátkých délek je výhodnější konvoluce přímá v časové oblasti, ovšem od jisté délky impulsové odezvy je výhodnější použít pro výpočet konvoluce DFT, respektive její realizaci pomocí algoritmu FFT (Fast Fourirer Transform) v kmitočtové oblasti. Poté se stane výpočet efektivnější v kmitočtové oblasti než v časové oblasti. Podle [2] odpovídá takové délce 32 nebo 64 vzorků.

1.2 Procesní zpoždění algoritmu rychlé konvoluce

Lidské ucho je schopno rozeznávat časové změny zvukového signálu i pro velmi malé hodnoty času. Prakticky bylo změřeno, že při reprodukci dvou signálů zpožděných o více než 10 ms jsou tyto signály vnímané jako signály oddělené.

Podle [2] výpočet konvoluce pomocí FFT vykazuje procesní zpoždění, ktere je způsobeno následujícímy kroky. Procesor čeká na segment vstupních vzorků, které mají být načteny a poté vykonává několik náročných výpočtů předtím, než je výstupní segment zrekonstruován. Odezva na výpočet prvního vzorku, respektive N vzorků výstupního segmentu je označována jako procesní zpoždění. Celkové vstupní a výstupní zpoždění se tak rovná délce segmentu N násobené vzorkovací periodou T_{vz} a době, za kterou bude spočítána odezva pro tento segment. Procesní zpoždění je závislé na délce impulsové odezvy a tak pro velmi dlouhé impulsové odezvy nabývá zpoždění vysokých hodnot, které je nežádoucí např. při zpracování zvuku vznikajícího při tzv. živé reprodukci hudby nebo přímém přenosu vystoupení.

V kapitole 1.4 bude popsána efektivní metoda potlačení procesního zpoždění pomocí vhodné segmentace impulsové odezvy.

1.3 Implementace FIR filtrace pomocí rychlé konvoluce

Jelikož je řád FIR systému často podstatně vyšší než u IIR systému, je výhodné realizovat systém pomocí rychlé diskrétní konvoluce v kmitočtové oblasti. Výpočetní náročnost takového systému pak narůstá lineárně s řádem systému v důsledku využití algoritmu FFT.

Následující rovnice a text jsou převzaty z [1]. Zjednodušený zápis (1.5) obrazu diskrétní Fourierovy transformace definuje jen indexované diskrétní složky spektra.

$$\boldsymbol{X}(k) = \sum_{n=0}^{N-1} x(n) \mathrm{e}^{-jk\frac{2\pi}{N}n}. \tag{1.5}$$

Vzor takového obrazu je definován pomocí inverzní diskrétní Furierovy transformace

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} \boldsymbol{X}(k) \mathrm{e}^{+jk\frac{2\pi}{N}n} = \frac{1}{N} \sum_{n=0}^{N-1} \boldsymbol{X}^*(k) \mathrm{e}^{-jk\frac{2\pi}{N}n}, \tag{1.6}$$

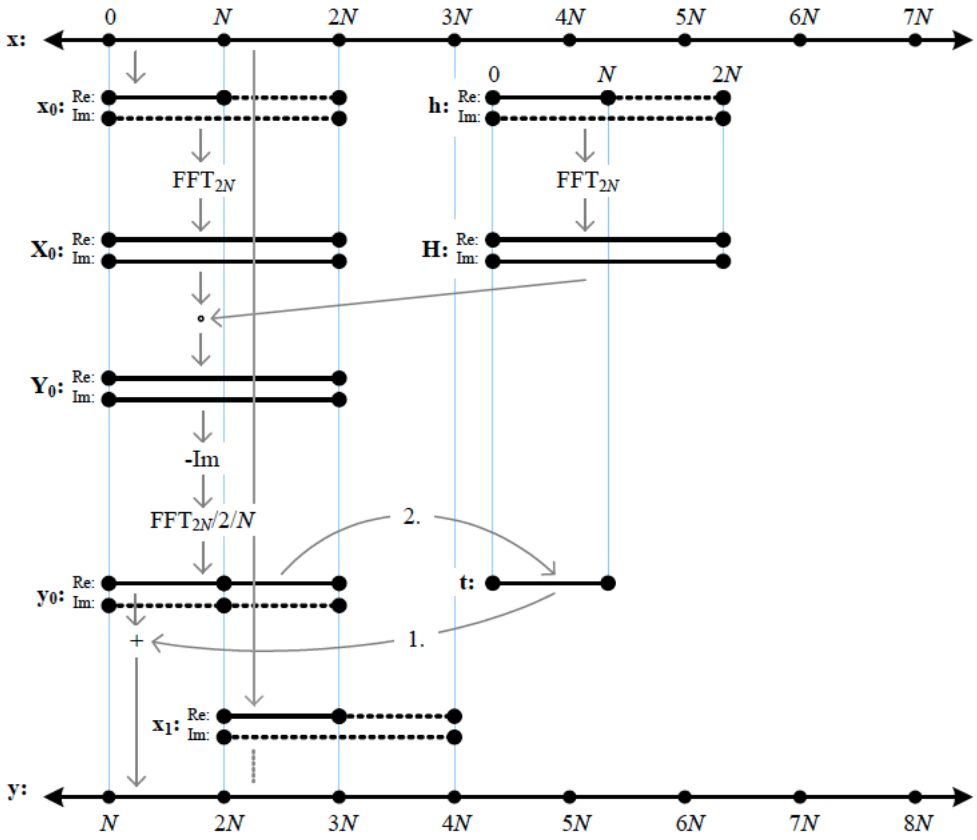
kde * označuje komplexně sdružený obraz dopředné Fourierovy transformace. Dopředná a zpětná transformace se liší pouze ve znaménku jádra transformace a ve váhovací konstantě $1/N$. Zpětnou (inverzní) Fourierovu transformaci lze tedy realizovat pomocí dopředné diskrétní Fourierovy transformace komplexně sdruženého obrazu a následného váhování výsledku konstantou $1/N$. Podle výše uvedených rovnic lze popsat obraz diskrétní Fourierovy transformace impulsové odezvy. Tento obraz popisuje rovnice (1.7). Obraz $\boldsymbol{H}$ odpovídá kmitočtové charakteristice systému. Rovnice je použita z [1].

$$\boldsymbol{H}(k) = \sum_{n=0}^{N-1} h(n) \mathrm{e}^{-jk\frac{2\pi}{N}n}. \tag{1.7}$$

1.3.1 Optimalizovaný návrh jednokanálové rychlé konvoluce s přičtením přesahu

Následující text vychází z [1]. Obrázek 1.1 popisuje realizaci jednokanálové rychlé konvoluce s přičtením přesahu. Pro optimální realizaci FIR systému vysokého řádu v kmitočtové oblasti je zapotřebí dvou dvojitých zásobníků, které se postupně naplňují a vyprazdňují daty v algoritmu skládajícím se z vnějšího a vnitřního cyklu. První dvojitý zásobník realizuje segmentaci vstupního signálu, zatímco druhý dvojitý zásobník má na starost desegmentaci výstupního signálu. Proces naplňování a vyprazdňování zásobníků probíha ve vnějším cyklu algoritmu.

V rámci vnějšího cyklu tvoří vstupní dvojitý zásobník délky N segment vstupního signálu $\boldsymbol{x_i}$ pro každý časový interval NT_{vz}. Na obrázku 1.1 má vstupní signál délku $7N$. Délka segmentu pro naplnění zásobníku je rovna N.



Obr. 1.1: Realizace jednokanálové rychlé konvoluce s přičtením přesahu. Převzato z [1].

Segment $\boldsymbol{x_i}$ se ve vnitřním cyklu algoritmu nejdříve doplní nulami na délku $2N$. Předpokládáme implementaci výpočtu FFT, která pracuje na vstupu s komplexní posloupností. Proto je vstupní segment $\boldsymbol{x_i}$ načten do komplexního zásobníku. V jednokanálové realizaci je realná část zásobníku naplněna vstupní posloupností a imaginární čast zásobníku je naplněna nulami. Po naplnění se následně spočítá FFT segmentu délky $2N$ vstupní posloupnosti $\boldsymbol{x_i}$. Výsledkem je kmitočtové spektrum[1] (obraz) aktuálního segmentu $\boldsymbol{X_i}$. Obdobným způsobem je naplněn komplexní zásobník impulsové odezvy konečné délky. Pomocí FFT se spočítá také kmitočtová charakteristika $\boldsymbol{H}$ délky $2N$. Pro časově invariantní FIR systém lze tento výpočet uskutečnit ještě před začátkem vnějšího cyklu.

Následuje výpočet komplexního Hadamardova součinu[2] kmitočtového spektra aktuálního segmentu $\boldsymbol{X_i}$ a kmitočtové charakteristiky $\boldsymbol{H}$. Výpočet znázorňuje rov-

[1]Pro obrazy vstupní a výstupní posloupnost se bude používat pojem kmitočtové spektrum a pro obraz impulsové odezvy pojem kmitočtová charakteristika

[2]Jedná se násobení dvou vektorů nebo matic `A` a `B` o stejném rozměru po složkách.

nice (1.8) převzata z [1].

$$\boldsymbol{Y}(k) = \boldsymbol{X}(k) \circ \boldsymbol{H}(k). \tag{1.8}$$

Výsledkem součinu je aktuální spektrum konvoluce $\boldsymbol{Y_i}$. Pro realizaci efektivního výpočtu zpětné FFT následuje komplexní sdružení aktuálního kmitočtového spektra konvoluce – imaginární část je vynásobena -1. Poté je zpětné FFT vypočítáno pomocí dopředného FFT a následně váhováno $1/(2N)$, čímž se získa výsledek konvoluce $\boldsymbol{y_i}$ aktuálního segmentu $\boldsymbol{x_i}$ a impulsové odezvy $\boldsymbol{h}$.

Další proces vnitřního cyklu spočívá v přičtení přesahu a je rozdělen do dvou kroků. V prvním kroku se provede součet první poloviny délky zásobníku $\boldsymbol{y_i}$ s dočasným zásobníkem $\boldsymbol{t}$. Pro nulové počáteční podmínky je tento dočasný zásobník před začátkem prvního cyklu naplněn nulami. Součet dočasného zásobníku a první poloviny zásobníku $\mathbf{y}_i$ definuje aktuální výstupní segment realizace, který je poté ve vnějším cyklu zařazen do výstupní posloupnosti. Ve druhém kroku probíhá naplnění dočasného zásobníku $\boldsymbol{t}$ druhou polovinou zásobníku $\boldsymbol{y_i}$ délky N. Dočasný zásobník je tak naplněn novým segmentem, vnitřní cyklus se ukončí a pokračuje se novým cyklem segmentace vstupního signálu.

Ve skutečnosti realizaci provází procesní zpoždění $2NT_{\text{vz}}$, které je dáno časem, za který se naplní vstupní dvojitý zásobník vzorky vstupního signálu a také odezvou na počáteční podmínky. Pro nulové počáteční podmínky je vstupní zásobník naplněn nulami. Algoritmus počítá s nulami, než je načten první segment vstupního signálu.

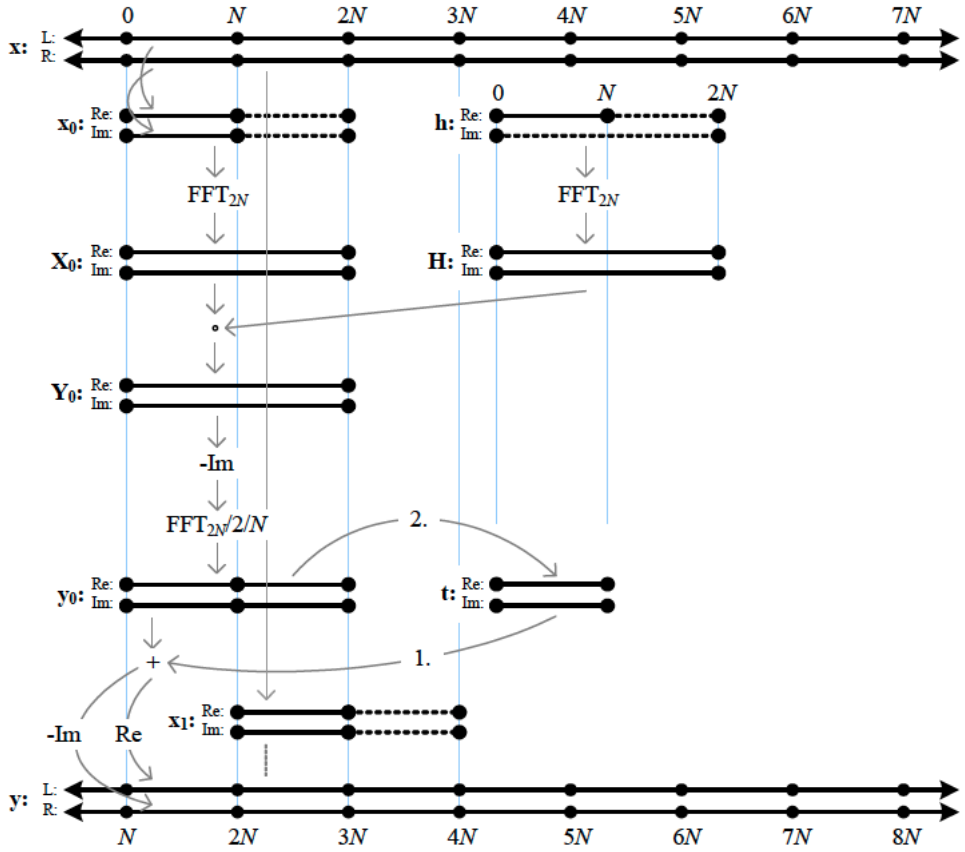
1.3.2 Optimalizovaný návrh dvoukanálové rychlé konvoluce s přičtením přesahu

Pro následující vícekanálovou realizaci bude uvažován systém s nejmenším možným počtem vstupů a výstupů 2×2. Systém je tedy připraven pro zpracování klasického stereofonního signálu. Implementovaný systém může být skokově variantní, kdy ke změně dochází s každým novým segmentem.

Podle [1] vychází optimalizovaný návrh z návrhu jednokanálové rychlé konvoluce s přičtením přesahu. Na rozdíl od jednokanálové realizace se zde využívá imaginární části komplexního počtu. Využití komplexního počtu je pro algoritmus zpracovávající dva kanály současně nezbytné. Dvoukanálový signál načteme do zásobníku, který se skládá z realné a imaginární části. Pravý kanál do reálné časti a levý do imaginární nebo obráceně. Obrázek 1.2 detailně popisuje celý algoritmus.

Na obrázku je levý kanál vstupního signálu segmentován do reálné části vstupního segmentu a pravý kanál do imaginární části. Veškeré následující operace jsou totožné s operacemi u jednokanálové realizace. Rozdíl oproti jednokanálové realizaci spočívá v části desegmentace, kde je přesah uložen do dočasného komplexního

zásobníku $\boldsymbol{t}$. Reálná i imaginární složka tohoto zásobníku je pro nulové počáteční podmínky naplněna nulami.



Obr. 1.2: Realizace dvoukanálové rychlé konvoluce s přičtením přesahu. Převzato z [1].

Výstupní posloupnost představuje dvoukanálový signál. Desegmentace výstupní posloupnosti spočívá v postupném naplňování reálné a imaginární části výsledku konvoluce $\boldsymbol{y_i}$ do pravého a levého kanálu výstupního signálu. Před desegmentací pravého výstupního kanálu je imaginární část nejdříve vynásobena -1 a až poté desegmentována.

Ve výsledku poté realizace zpracovává dvě vstupní posloupnosti s minimálním přírůstkem výpočetní náročnosti.

1.4 Potlačení procesního zpoždění

Algoritmus potlačení procesního zpoždění výchází z [2]. Pro další úvahy je vhodné zavést zápis (1.9). Zápis značí N-dlouhý výsledek konvoluce začínající na vzorku n_0. Rovnice vychází ze vztahu (1.4).

$$y_0(n) = x(0, N) * h. \tag{1.9}$$

Celkovou výstupní posloupnost $y(n)$ lze pak zapsat jako zřetězení dílčích výsledků konvolucí vstupních segmentů a impulsové odezvy. Zřetězením se myslí sečtení jednotlivých posloupností s potřebným zpožděním. Výsledek $y(n)$ se podle [2] zapíše jako

$$y(n) = x(0, N) * h|x(N, N) * h|x(2N, N) * h \ldots, \tag{1.10}$$

kde $*$ značí konvoluci a $|$ značí zřetězení dílčích výsledků do $y(n)$.

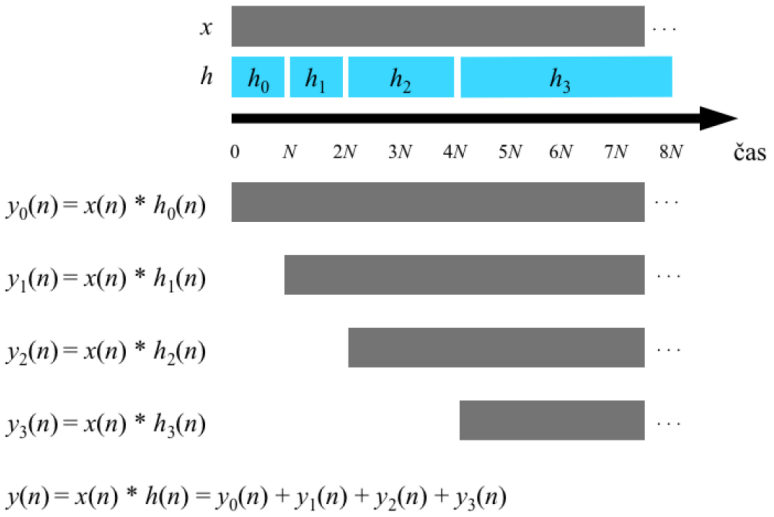
Potlačení procesního zpoždení spočívá v rozdělení impulsové odezvy do segmentů různých délek. Linearita konvoluce umožňuje spočítat konvoluci každého z těchto segmentů s vstupní posloupností a poté se dílčí výsledky konvoluce sčítají a synchronizují do výstupní posloupnosti. Doba procesního zpoždění se tak zkrátí pouze na dobu, za kterou se načte a spočítá odezva na **první segment** délky N.

Na obrázku 1.3 je tato segmentace znázorněna, kde délka impulsové odezvy je rovna $8N$. Impulsová odezva je rozdělena do segmentů h_0, h_1, h_2 a h_3 s délkami N, N, $2N$, a $4N$. Následně je spočítána konvoluce vstupního signálu a aktuálního segmentu impulsové odezvy totožné délky, kde $y_i(n)$ značí aktuální výsledek konvoluce.

Využitím zápisu z rovnice (1.10) lze pak celkovou výstupní posloupnost zapsat jako

$$\begin{aligned} y(n) =& x(0, N) * h(0, N)|x(0, N) * h(1, N)|x(N, N) * h(0, N)|x(N, N) * h(N, N)| \\ & x(0, 2N) * h(2N, 2N)|x(2N, N) * h(0, N)|x(2N, N) * h(1, N)| \\ & x(3N, N) * h(0, N)|x(3N, N) * h(1, N)|x(4N, N) * h(0, N)| \\ & x(4N, N) * h(1, N)|x(3N, 2N) * h(2N, 2N)|x(0, 4N) * h(4N, 4N) \ldots, \end{aligned} \tag{1.11}$$

kde jednotlivé segmenty impulsové odezvy h_0 až h_3 musí odpovídat celkové délce impulsové odezvy h před rozdělením. Z rovnice však nelze vypozorovat synchronizaci aktuálních výsledků konvolucí. Na obrázku 1.3 je princip zřetězení (synchronizace) naznačen. Velikost N odpovídá délce, kdy je přímá forma konvoluce v časové oblasti efektivnější než konvoluce v kmitočtové oblasti. Odezva segmentu délky N musí být spočítána za dobu trvání jednoho vzorku. Z výše popsaného algoritmu vyplývá, že odezva segmentu délky N začíná N vzorků před impulsovou



Obr. 1.3: Rozdělení impulsové odezvy a následný součet výsledků konvolucí. Vychází z [2].

odezvou dalšího segmentu. Každý N-dlouhý segment odezvy musí čekat N vzorkovacích period na načtení N-dlouhých segmentů vstupní posloupnosti $x_i(n)$. Výsledek konvoluce musí být spočítán ještě před další vzorkovací periodou. To má za následek čekání procesoru na výsledek konvoluce před načtením nového segmentu. Tento algoritmus není zcela optimální a neřeší potlačení procesního zpoždění, je ovšem výchozím pro následující funkční algoritmus.

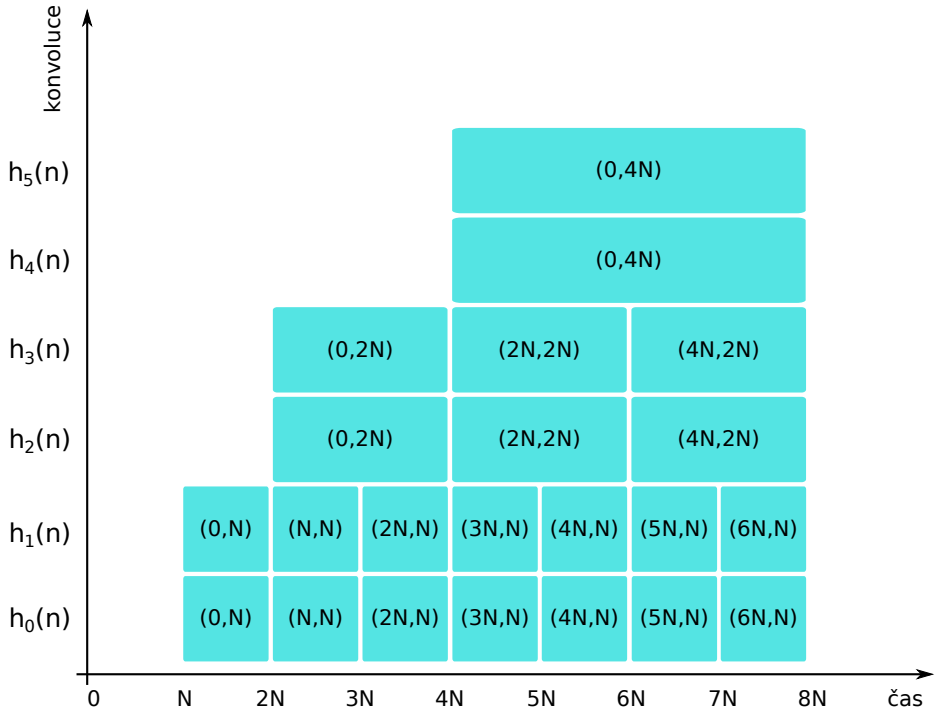
Následující text výchází z [2]. Funkční algoritmus spočívá v rozdělení impusové odezvy tak, aby měl procesor dostatek času pro výpočet konvoluce a nemusel čekat na načtení vstupních dat. Tabulka 1.1 demonstruje vhodné rozdělení impulsové odezvy. Odezva segmentu délky N musí začít $2N$ vzorků od začátku impulsové odezvy dalšího segmentu. Poté má procesor na výpočet segmentu délky N čas odpovídající N vzorků. V nadcházejících algoritmech budou všechny výpočty realizovány pomocí FFT. Výpočet nejmenšího segmentu pomocí přímé konvoluce nebude využíván, proto v tabulce není uveden.

Tab. 1.1: Vhodná segmentace impulsové odezvy pro algoritmus FFT. Použito z [2].

	N	N	$2N$	$2N$	$4N$	$4N$
h	h_0	h_1	h_2	h_3	h_4	h_5

Realizaci funkčního algoritmu lze pochopit z obrázku 1.4, který demonstruje rozdělení impulsové odezvy v čase a dobu, za kterou se odezvy různých délek vypo-

čítají. V čase N začíná výpočet konvoluce segmentu impulsové odezvy h_0 délky N se segmentem vstupní posloupnosti délky N začínajícím v čase 0. Výpočet konvoluce musí být spočítán do doby $2N$. Souběžně s tímto výpočtem probíhá výpočet konvoluce stejného vstupního segmentu s dalším segmentem impulsové odezvy h_1 stejné délky. Po výpočtu se načte další segment vstupní posloupnosti začínající v čase N, který v čase $2N$ vstupuje do výpočtu se segmenty impulsové odezvy h_0 a h_1. V tomto čase se také spouští výpočet vstupního segmentu délky $2N$ (začínajícího v čase 0) se segmenty impulsové odezvy h_2 a h_3. Pro impulsové odezvy h_2 a h_3 délek $2N$ začíná výpočet konvoluce v čase $2N$ a musí být spočítan v čase $4N$.

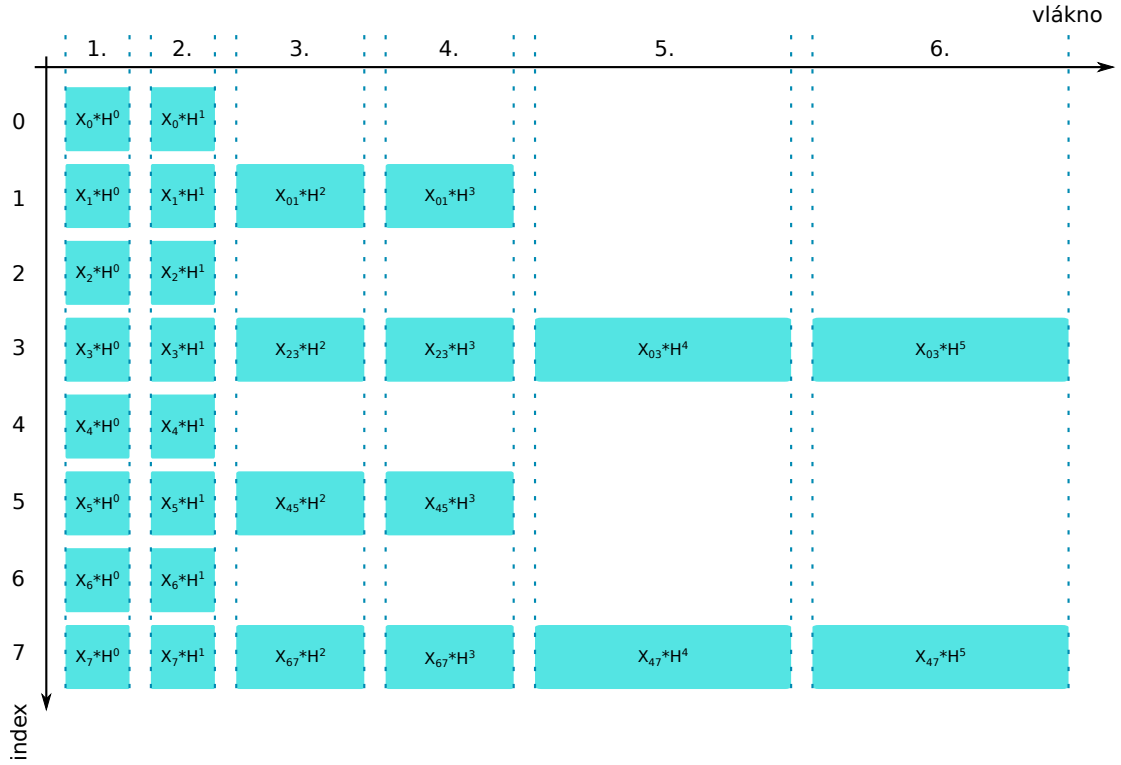


Obr. 1.4: Rozdělení impulsové odezvy a doba výpočtů konvolucí v časové oblasti. Výchází z [2].

Tímto způsobem probíhá načtení a výpočet konvolucí všech vstupních segmentů a impulsových odezev.

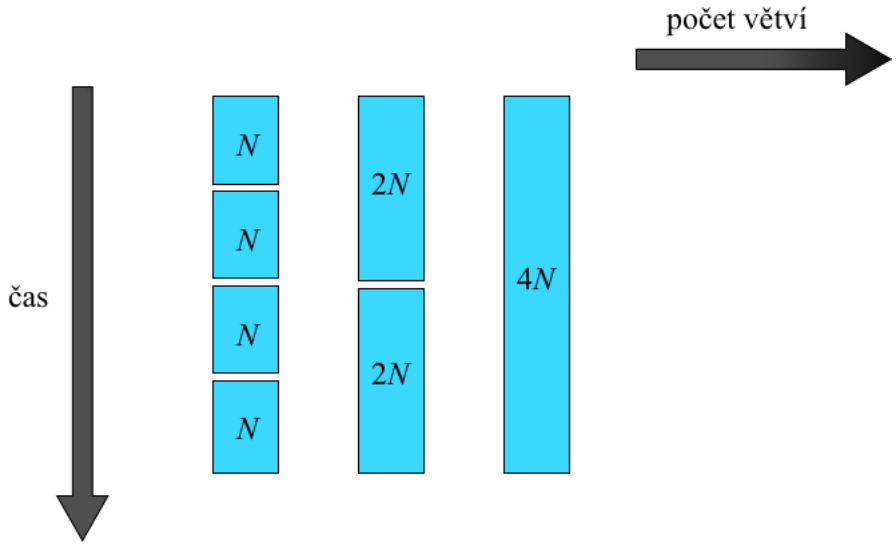
Potlačení procesního zpoždění dále spočívá ve výpočtu konvolucí segmentů menších délek během výpočtu konvoluce segmentu vetší délky a následné synchronizace výsledků. Takový výpočet lze realizovat v paralelních větvích, kdy lze během výpočtu segmentu délky $4N$ souběžně počítat 4-krát segment délky N. Zjednodušené schéma 1.6 popisuje výpočet v paralelních větvích. Efektivní realizaci takového algoritmu využívající plně výkon dnešních procesorů, lze docílit vícevláknovým programováním.

Při realizaci algoritmu využívající jedno vlákno procesoru výpočty konvolucí



Obr. 1.5: Výpočty dílčích konvolucí jednotlivých kroků algoritmu v kmitočtové oblasti.

začínají podle schématu 1.5. Pro označení výpočtu vstupního segmentu s ruznými segmenty impulsové odezvy bude ponechán pojem výpočetní *vlákno*, ačkoliv se v této chvíli nejedná o fyzické vlákno (jádro) procesoru. Index ve schématu představuje nový segment vstupní posloupnosti (krok algoritmu). Ve schématu lze vidět dílčí výpočty konvolucí vstupních segmentů se segmenty odpovídající délky a také začátky výpočtů jednotlivých vláken s krokem algoritmu. Dokud nebudou všechny výpočty provedeny, hodnota indexu se nezvýší – nenačte se další segment vstupní posloupnosti. Zápis ve schématu odpovídá výpočtům v kmitočtové oblasti. Je zde také zavedeno nové značení demonstrující přesně výpočty segmentů. Horní index identifikátorů značí pořadí segmentu kmitočtové charakteristiky impulsové odezvy a dolní index označuje pořadí vstupního segmentu. Víceciferný dolní index odpovídá složením dvou a více segmentů ze vstupní posloupnosti. Například zápis X_{01} představuje segment délky $2N$ skládající se z nultého a prvního vstupního segmentu. Tento segment vstupuje do výpočtu se segmentem impulsové odezvy H^2. Značení bude dále využito v nadcházející teoretické části.



Obr. 1.6: Praktická segmentace a paralelní výpočet dílčích konvolucí.

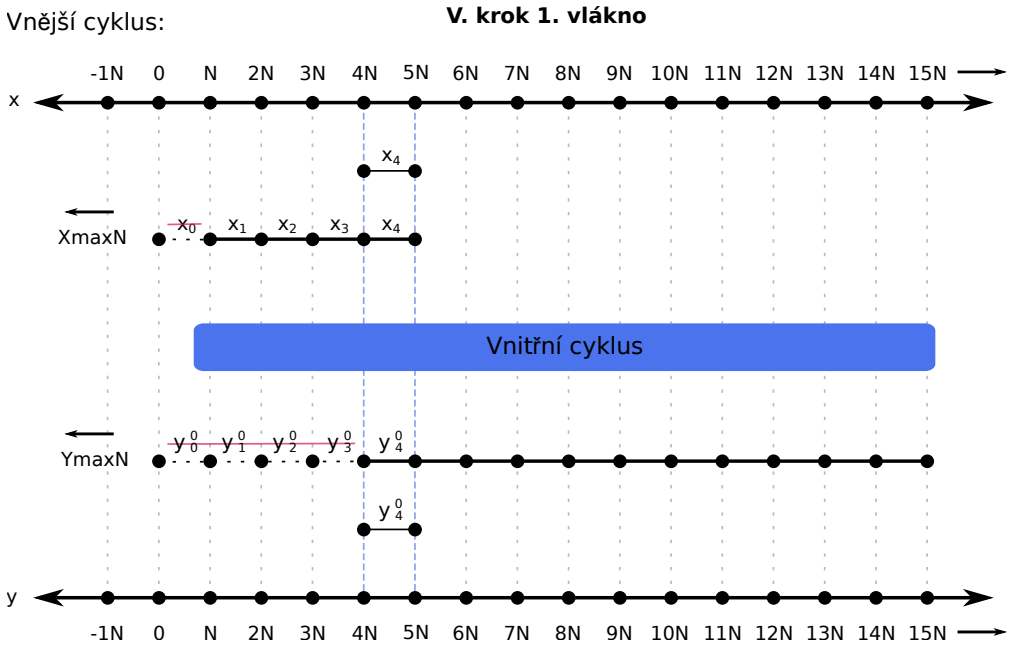
1.5 Optimalizovaný návrh jednokanálové rychlé konvoluce s přičtením přesahu a s potlačeným procesním zpožděním

V této části popíšu vnější a vnitřní cyklus algoritmu rychlé konvoluce s minimalizovaným procesním zpožděním a také synchronizaci dílčích výsledků konvoluce. Algoritmus nepředpokládá výpočty ve více vláknech procesoru. Pro snadné krokování algoritmu použiju vlastní schámata čerpající z 1.1. Indexování segmentů začíná od nuly. Ve schématech není podrobně znázorněno zpracování dvoukanálové vstupní posloupnosti pro zachování větší přehlednosti. V případě popisu zpracování dvoukanálového vstupu je potřeba schémata doplnit podle 1.2.

Algoritmus bude popsán na příkladě s impulsovou odezvou délky $14N$, která je rozdělena na segmenty délek $N\ N\ 2N\ 2N\ 4N\ 4N$. Podle počtu segmentů impulsové odezvy se také určí počet výpočetních vláken. V ukázkovém příkladu je tedy zapotřebí 6 výpočetních vláken. Přesahy z výsledků konvolucí jsou ukládány do dočasného zásobníku t stejné délky jako impulsová odezva. Podle [2] lze pro zvýšení efektivnosti algoritmu vypočítat kmitočtovou charakteristiku impulsové odezvy dopředu, ještě před začátkem segmentace vstupní posloupnosti.

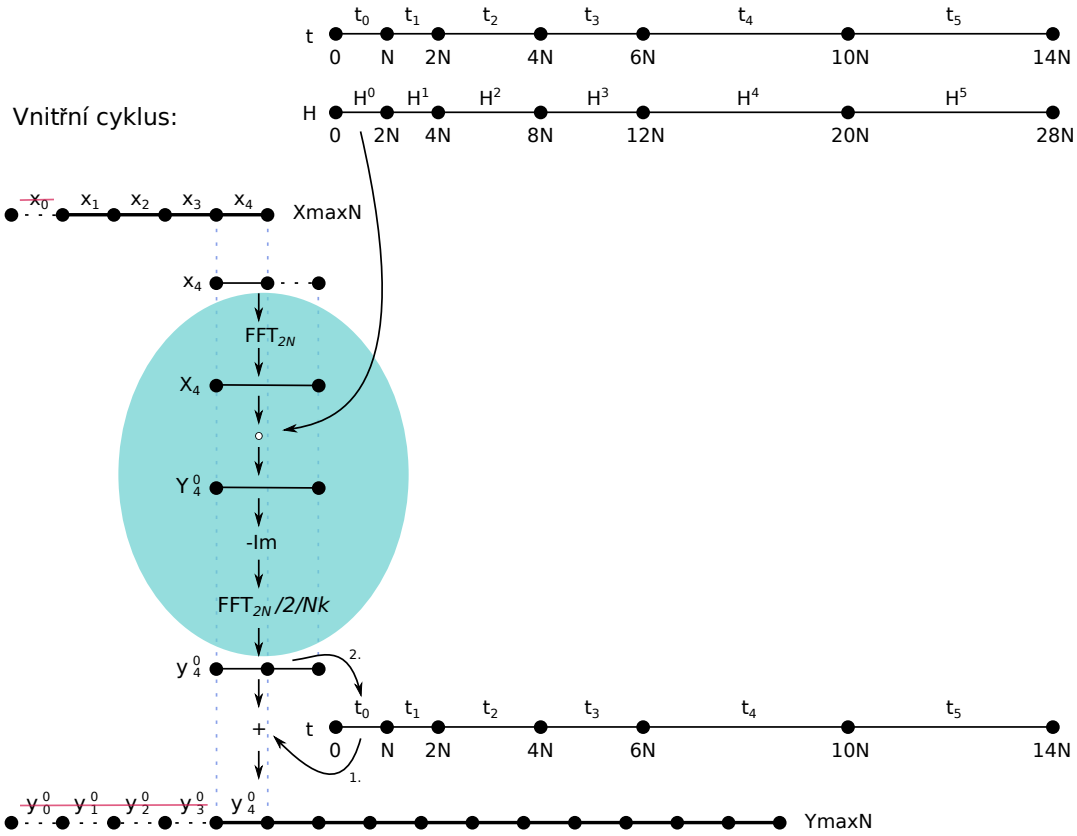
Ve schématu 1.7 je uveden výpočet prvního vlákna pátého kroku algoritmu s detailně popsaným výpočtem konvoluce. Vyšší krok algoritmu lépe přibližuje celý proces zpracování vstupní posloupnosti. V dalších schématech bude výpočet konvoluce znázorněn jednodušeji, jelikož se jeho princip po celou dobu algoritmu nemění a také bude algoritmus krokován od začátku.

Segmentaci a desegmentaci vstupní a výstupní posloupnosti ve vnějším cyklu algoritmu obstarávají dva zdvojené komplexní zásobníky. Vstupní zásobník $XmaxN$ svou délkou musí odpovídat nejdelšímu segmentu rozdělené impulsové odezvy, tedy 4 N. Do tohoto zásobníku jsou postupně ukládány nové segmenty vstupní posloupnosti s každým krokem algoritmu. Výstupní zásobník $YmaxN$ je dlouhý $11N$. Tato délka je spočítána jako polovina délky impulsové odezvy plus délka jejího nejdelšího segmentu. Ve vnějším cyklu algoritmu se načte nový segment x_4 délky N na konec zásobníku $XmaxN$. Pro první vlákno výpočtu tento segment o stejné délce vstupuje do vnitřního cyklu algoritmu.



Obr. 1.7: Schéma vnějšího cyklu pátého kroku algoritmu - první vlákno

Ve vnitřním cyklu 1.8 se segment doplní nulami na dvojnásobnou délku a je spočítáno kmitočtové spektrum vstupního segmentu pomocí FFT. Spektrum je vynásobeno s nultým segmentem kmitočtové charakteristiky H^0. Výsledek je dále komplexně sdružen a pomocí FFT a váhovací konstanty N_k je převeden zpět do časové oblasti. Váhovací konstanta N_k odpovídá délce aktuálního segmentu. K první polovině výsledku konvoluce y_4^0 je přičten obsah nultého segmentu t_0 dočasného zásobníku a výsledek součtu je přičten a uložen na nultou pozici ve výstupním zásobníku. Na této pozici se nachází hodnoty z předešlých kroků, proto se musí výsledek y_4^0 s těmito hodnotami sečíst. Druhá polovina aktuálního výsledku je uložena na pozici t_0 dočasného zásobníku.

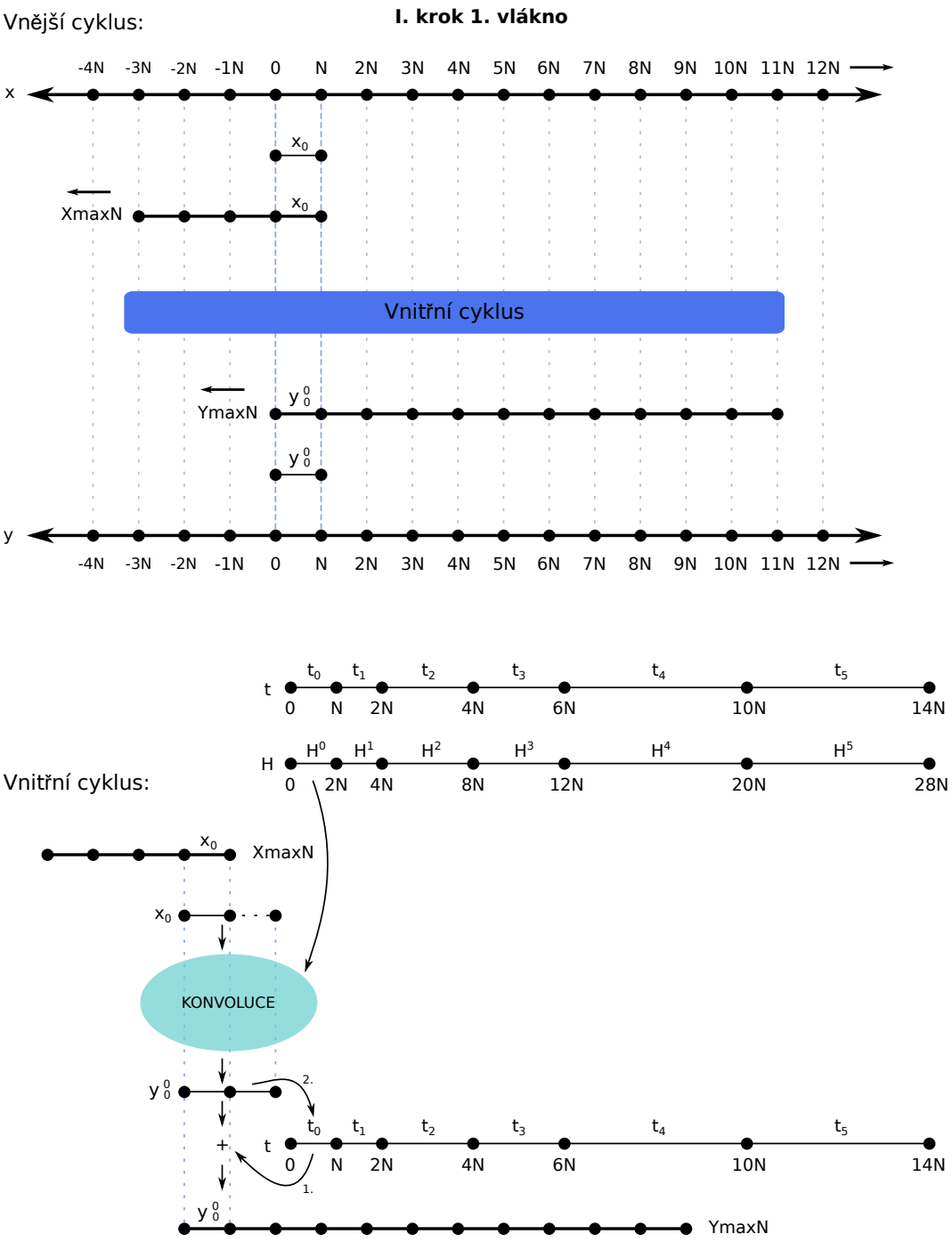


Obr. 1.8: Schéma vnitřního cyklu pátého kroku algoritmu - první vlákno

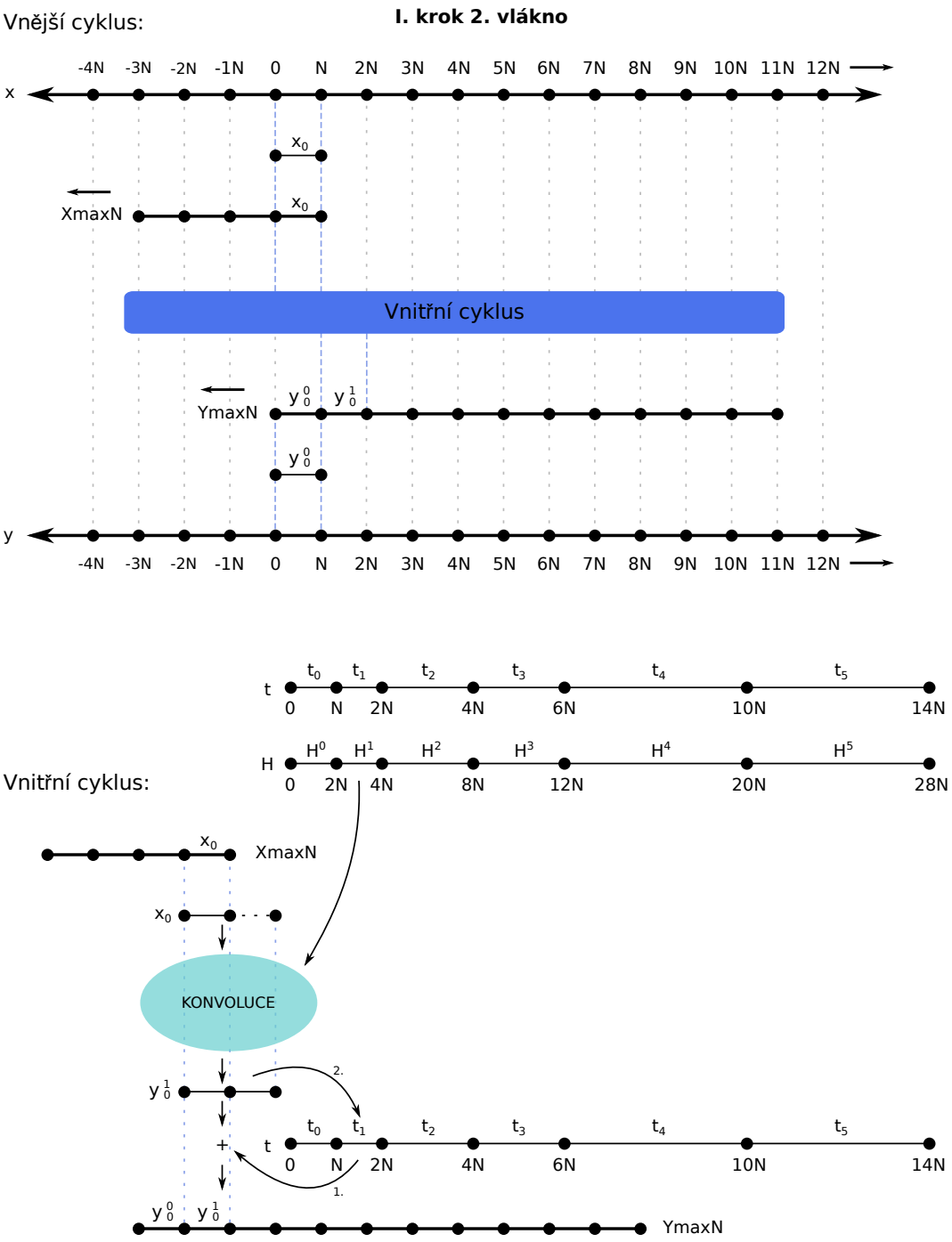
Do výstupní posloupnosti y se vždy vyčítá prvních N vzorků z výstupního zásobníku. Po výčtu je vstupní i výstupní zásobník posunut o N vzorků doleva. Na schématu 1.7 lze jasně vidět, jak se posunují jednotlivé segmenty a také zahození již nepotřebných segmentů.

Schéma 1.9 zobrazuje výpočet prvního vlákna prvního kroku algoritmu. Pro nulové počáteční podminky je vstupní, výstupní i dočasný zásobník vynulován. Veškeré operace prvního vlákna již byly popsány výše. Výpočet prvního vlákna je pro každý krok algoritmu vždy přičten a uložen na pozici 0 až N výstupního zásobníku.

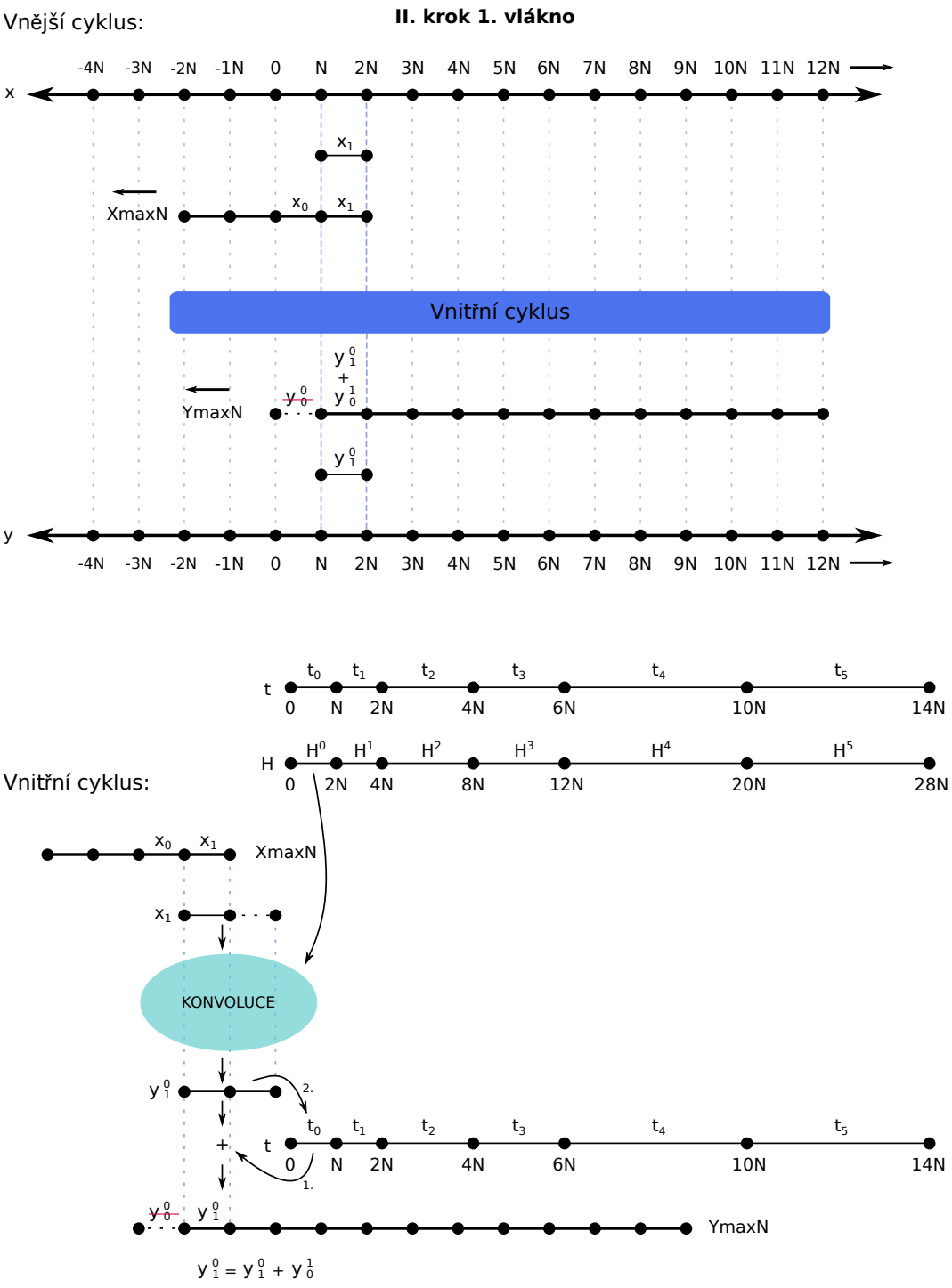
Druhé výpočetní vlákno probíhá obdobně jako první. Do vnitřního cyklu vstupuje stejný vstupní segment. Změna spočívá v Hadamardově součinu spektra vstupního segmentu se segmentem kmitočtové charakteristiky H^1 a přičtení obsahu pozice t_1 k první polovině výsledku konvoluce y_0^1. Po součtu je druhá polovina výsledku uložena na pozici t_1. Výsledek druhého vlákna je poté přičten a uložen do výstupního zásobníku na pozici N až $2N$. Tento krok popisuje schéma 1.10. Pro další kroky algoritmu probíhají výpočty a synchronizace prvního a druhého vlákna již analogicky.



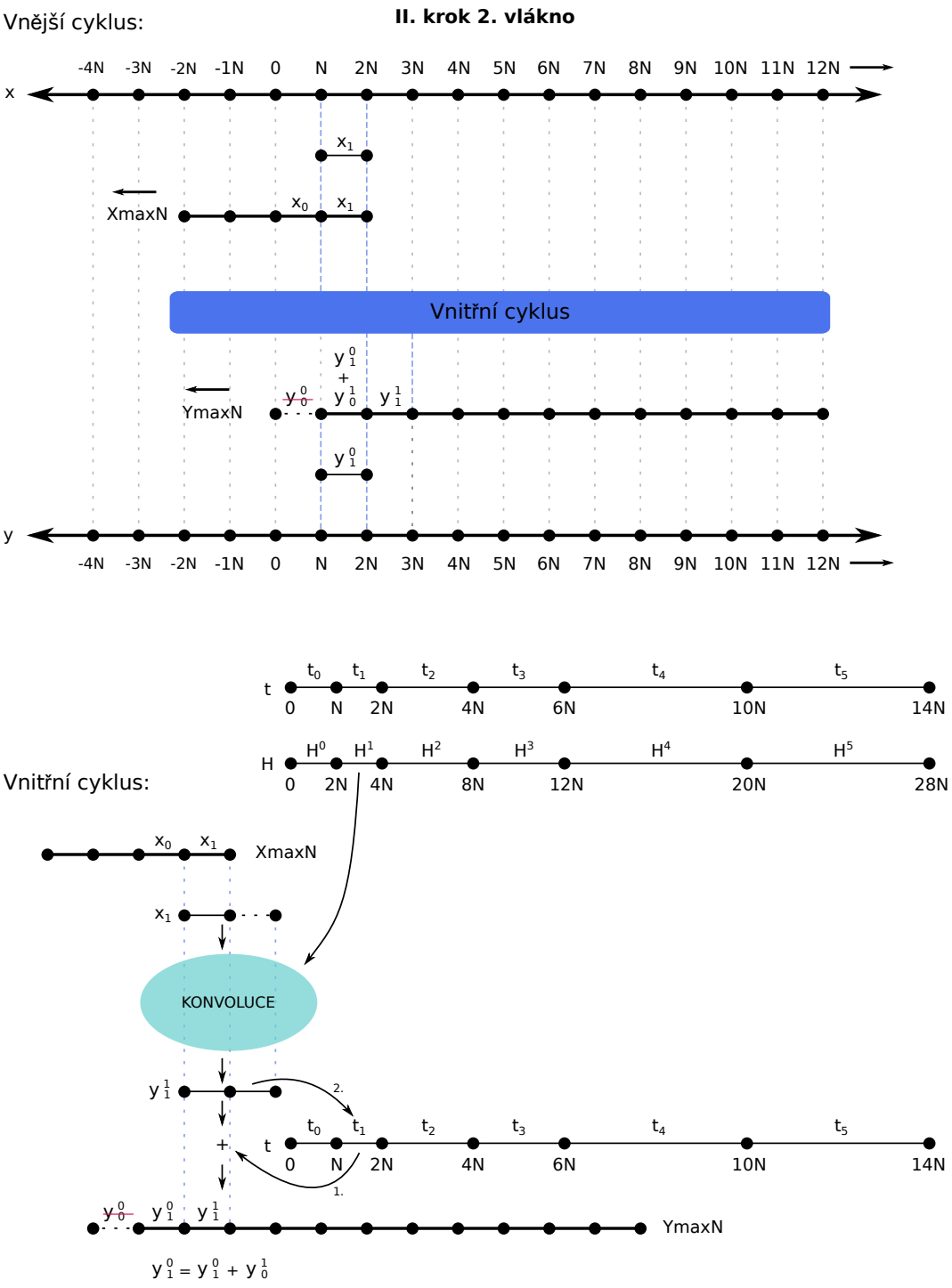
Obr. 1.9: Schéma vnějšího a vnitřního cyklu prvního kroku algoritmu - první vlákno



Obr. 1.10: Schéma vnějšího a vnitřního cyklu prvního kroku algoritmu - druhé vlákno

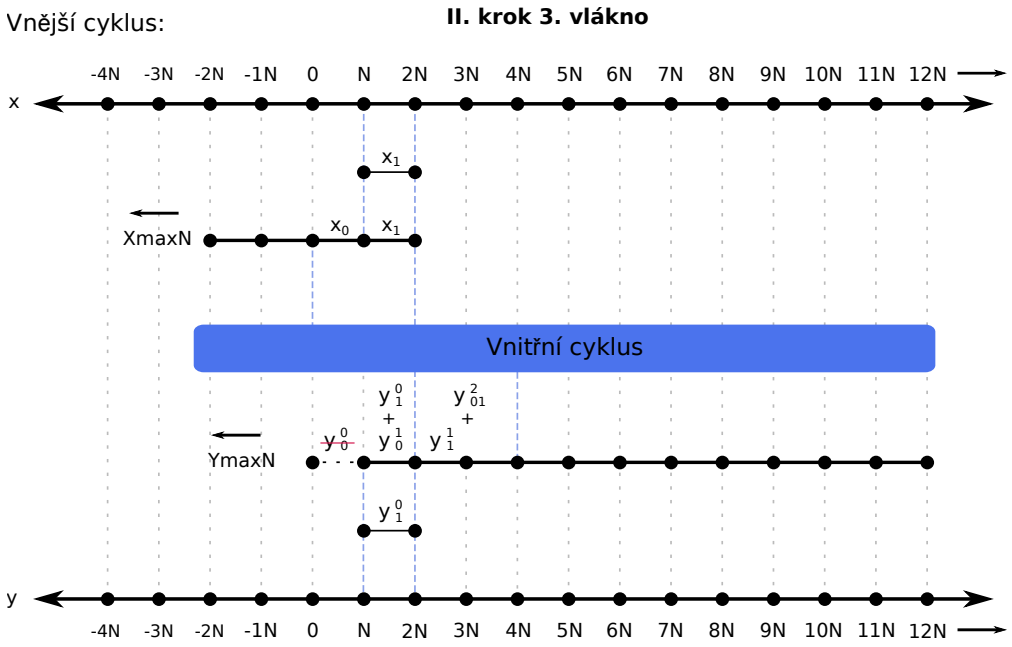


Obr. 1.11: Schéma vnějšího a vnitřního cyklu druhého kroku algoritmu - první vlákno



Obr. 1.12: Schéma vnějšího a vnitřního cyklu druhého kroku algoritmu - druhé vlákno

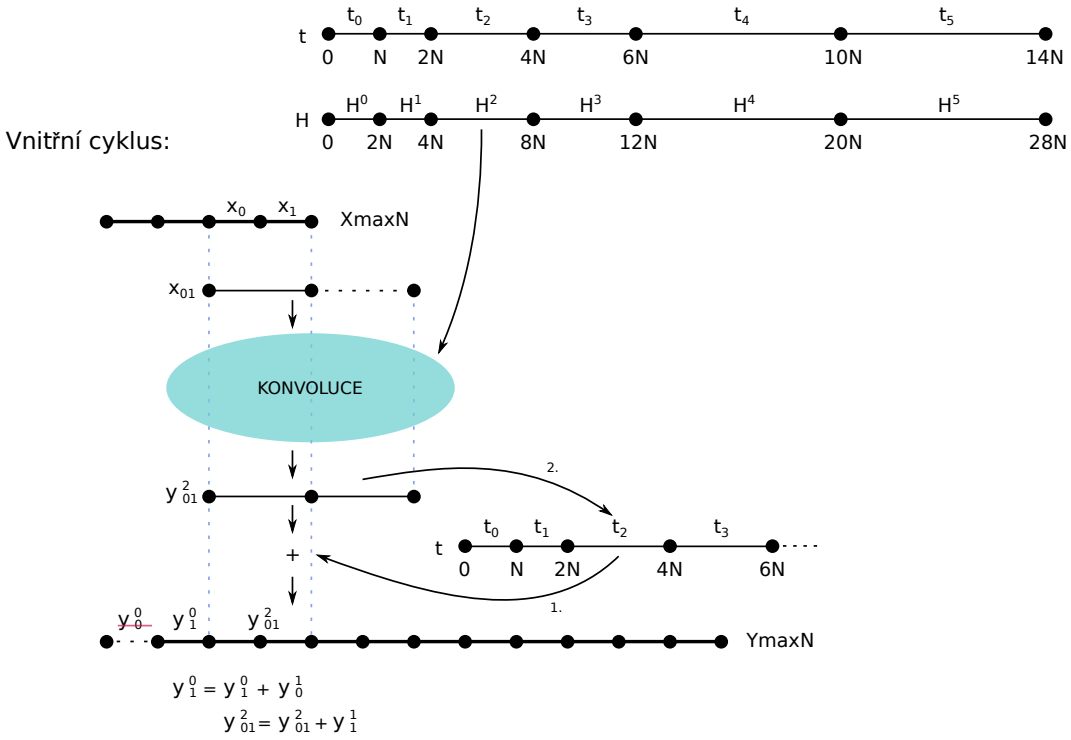
S každým druhým krokem začíná výpočet třetího a čtvrtého vlákna. Do vnitřního cyklu algoritmu vstupuje aktuální segment spolu s předešlým vstupním segmentem. Schéma 1.13 ukazuje načtení segmentu $x1$ do vstupního zásobníku vedle segmentu x_0 z předcházejícího kroku. Tyto dva segmenty pak vstupují do vnitřního cyklu jako jeden segment délky $2N$. Následuje výpočet spektra a Hadamardův součin se segmentem kmitočtové charakteristiky H^2. Pro výsledek konvoluce y_{01}^2 je k potřebným operacím přichystána pozice t_2 dočasného zásobníku. Synchronizace výsledku spočívá v přičtení a uložení na pozici N až $3N$ výstupního zásobníku. Princip výpočtu čtvrtého vlákna 1.15 již vyplývá z popisu předešlých vláken. Výsledek čtvrtého vlákna je přičten a uložen na pozici $3N$ až $5N$ výstupního zásobníku.



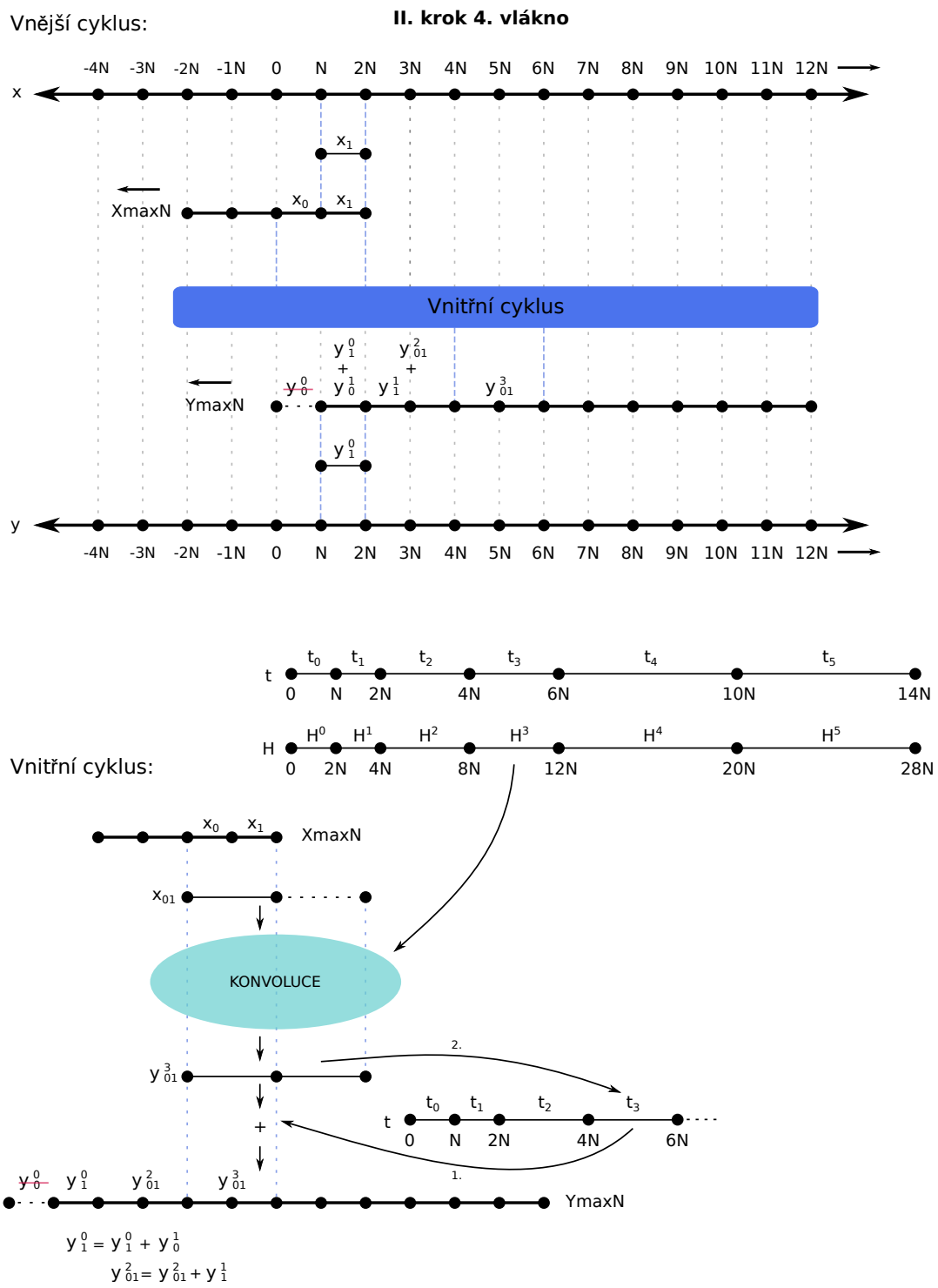
Obr. 1.13: Schéma vnějšího cyklu druhého kroku algoritmu - třetí vlákno

Pro výpočet konvoluce s nejdelšími segmenty impulsové odezvy H^4 a H^5 je každým čtvrtým krokem algoritmu vybrán celý obsah zásobníku, který vstupuje do vnitřního výpočetního cyklu. Výsledky pátého a šestého vlákna jsou délky $4N$ (již po součtu s příslušným obsahem dočasného zásobníku) a jejich synchronizace je následující. Výsledek pátého vlákna je přičten a uložen na pozici $3N$ až $7N$ a výsledek šestého vlákna na pozici $7N$ až $11N$ výstupního zásobníku. Proces pátého a šestého vlákna není již ve schématech zobrazen.

V případě více výpočetních vláken by výpočty probíhaly obdobně a synchronizace by vycházela analogicky z ukázkového příkladu.



Obr. 1.14: Schéma vnitřního cyklu druhého kroku algoritmu - třetí vlákno



Obr. 1.15: Schéma vnějšího a vnitřního cyklu druhého kroku algoritmu - čtvrté vlákno

2 OPTIMALIZOVANÁ REALIZACE FIR PROCESORU V MATLABU

Následující kapitola se bude zabývat vlastní implementací FIR procesoru v prostředí *Matlab*®. Konkrétně byla použita verze R2014a (8.3.0.532). Takovým procesorem může být například FIR filtr vysokého řádu nebo efekt reverb. Procesor je schopen zpracovávat impulsové odezvy délek větších než 500 000 vzorků. Tato implementace však nepracuje s více vlákny procesoru. Nejdříve bude uveden seznam důležitých proměnných vyskytujících se v hlavní smyčce algoritmu. Hlavní smyčkou je myšleno zpracování vstupní nekonečné posloupnosti posloupností konečné délky. Dále bude popsána implementace jednokanálové rychlé konvoluce s přičtením přesahu, dvoukanálové rychlé konvoluce s přičtením přesahu a implementace rychlé konvoluce s minimalizovaným procesním zpožděním. Nakonec proběhne několik testů pro ověření správnosti algoritmu.

Pro přehlednost a snadnou orientaci při krokování algoritmu v textu bude použit pro proměnné a funkce text psaný strojopisem. Znak `*` značí násobení a `.*` značí Hadamardův součin. Z technických důvodů budou komentáře k řádkům programu psané bez diakritiky.

Seznam důležitých proměnných:

- `xL, xR` vstupní posloupnosti
- `x_max` vstupní komplexní zásobník
- `h` impulsová odezva
- `H` obraz impulsové odezvy
- `xL_n` aktualní vstupní segment kanálu 1
- `xR_n` aktualní vstupní segment kanálu 2
- `X_n` obraz vstupního segmentu
- `Y_n` výsledek po Hadamardově součinu
- `y_n` aktuální výstupní segment v časové oblasti
- `yL_n` aktuální výstupní segment prvního kanálu
- `yR_n` aktuální výstupní segment druhého kanálu
- `tL` dočasný zásobník pro kanál 1
- `tR` dočasný zásobník pro kanál 2
- `y_maxL` výstupní zásobník pro kanál 1
- `y_maxR` výstupní zásobník pro kanál 2
- `yL, xR` výstupní posloupnosti

2.1 Jednokanálová rychlá konvoluce s přičtením přesahu

Jednokanálová rychlá konvoluce očekává jako vstup pouze monofonní signál. Ten je uložen v proměnné `xL`. Do proměnné `H` je uložen obraz impulsové odezvy vypočítaný funkcí `fft`. Parametrem této funkce je impulsová odezva `h`, vstupující do funkce, doplněna nulami na délku `2N`. Před započetím hlavní smyčky je ještě vytvořen a vynulován dočasný zásobník `tL`. Jádrem algoritmu zpracovávající vstupní posloupnost je pak následující hlavní smyčka. Algoritmus vychází z 1.1.

```
for i=1:I                              % I-krat se opakuje
    xL_n = xL((i-1)*N+1:i*N);          % segmentace vstupu
    % vypocet kmitoctoveho spektra
    X_n = fft(xL_n,2*N);
    Y_n = X_n.*H;                      % Hadamarduv soucin
    y_n = fft(conj(Y_n),2*N)/2/N;      % zpetne FFT
    yL_n = real(y_n(1:N))+tL;          % pricteni presahu
    % naplneni docasneho zasobniku presahem
    tL = real(y_n(N+1:2*N))
    yL((i-1)*N+1:i*N) = yL_n;          % desegmentace
end
```

Smyčku lze rozdělit do tří částí. Nejdříve se provede segmentace vstupní posloupnosti, dále proběhne několik výpočtů a poté dochází k desegmentaci nové výstupní posloupnosti.

V prvním kroku cyklu se do proměnné `xL_n` načítá vstupní segment o velikosti `N`. Následně se spočítá rychlá Fourierova transformace segmentu vstupní posloupnosti. Výsledkem je obraz `X_n`. Proměnná `Y_n` představuje výsledek Hadamardova součinu obrazu `X_n` s kmitočtovou charakteristikou impulsové odezvy `H`. V další fázi dochází k výpočtu zpětné Fourierovy transformace pomocí dopředné Fourierovy transformace komplexně sdruženého aktuálního kmitočtového spektra konvoluce s následným váhováním `2/N`. Výstup po Fourierově transformaci `y_n` je v komplexním tvaru a délky `2N`. V jednokanálové implementaci se využívá pouze reálná část komplexního výstupu. Přičtení přesahu, tj. obsahu dočasného zásobníku `tL` k první polovině reálné složky aktuálního segmentu `y_n` je uloženo do proměnné `yL_n`. Následně dochází k přepsání dočasného zásobníku `tL` přesahem novým. Tímto přesahem je myšlena druhá polovina reálné části výstupního segmentu `y_n`. Dočasný zásobník naplněný aktuálním přesahem se v dalších krocích cyklu zapojuje do výpočtu.

V poslední části se desegmentuje výstupní posloupnost. S každým dalším kro-

kem cyklu se aktuální výstupní segment `yL_n` o velikosti `N` ukládá do výstupní posloupnosti `yL`. Po `I` krocích se zrekonstruje kompletní výstupní posloupnost.

2.2 Dvoukanálová rychlá konvoluce s přičtením přesahu

Implementace dvoukanálové rychlé konvoluce s přičtením přesahu vychází z 1.2 a podobá se implementaci rychlé konvoluce jednokanálové. Před začátkem hlavní smyčky je potřeba vytvořit dva dočasné zásobníky `tL`, `tR` a vynulovat je. Každý z těchto zásobníků slouží k uložení přesahů realné a imaginární části komplexního výstupu `y_n`.

V hlavní smyčce algoritmu je segment prvního kanálu vstupní posloupnosti uložen do `xL_n` a druhého kanálu do `xR_n`. Dále je vytvořen vstupní komplexní zásobník `x_max`, kde je do reálné části uložen první kanál a do imaginární části druhý kanál vstupní posloupnosti. Obsah tohoto zásobníku pak vstupuje do funkce `fft`. Přičtení přesahu k první polovině reálné a imaginární složky aktuálního segmentu `y_n` je uloženo do proměnných `yL` a `yR_n`. Dočasné zásobníky `tL` a `tR` jsou naplněny druhou polovinou reálné a imaginární složky segmentu `y_n`. Proměnné `yL` a `yR` představují výstupní posloupnosti prvního a druhého kanálu.

```
for i=1:I
    xL_n = xL((i-1)*N + 1:i*N);  % segmentace kanalu 1
    xR_n = xR((i-1)*N + 1:i*N);  % segmentace kanalu 2
    % vstupni komplexni zasobnik
    x_max = complex(xL_n, xR_n);
    % vypocet kmitoctoveho spektra
    X_n = fft(x_max, 2*N);
    Y_n = X_n.*H;                % Hadamarduv soucin
    y_n = fft(conj(Y_n),2*N)/2/N;% zpetne FFT
    yL_n = real(y_n(1:N)) + tL;  % pricteni presahu kanalu 1
    yR_n = imag(y_n(1:N)) + tR;  % pricteni presahu kanalu 2
    tL = real(y_n(N + 1:2*N));   % ulozeni noveho presahu
    tR = imag(y_n(N + 1:2*N));   % ulozeni noveho presahu
    % vystupni posloupnost pro 1. kanal
    yL((i-1)*N + 1:i*N) = yL_n;
    % vystupni posloupnost pro 2. kanal
    yR((i-1)*N + 1:i*N) = -yR_n;
end
```

Implementace v Matlabu využívá dvou zásobníků (levý a pravý kanál) pro výstupní a dočasný zásobník namísto zásobníků komplexních. Tyto zásobníky budou využívány i pro následující algoritmus.

2.3 Implementace rychlé konvoluce s minimalizovaným procesním zpožděním

V této části bude představena implementace rychlé konvoluce s minimalizovaným procesním zpožděním. V algoritmu je nezbytné definovat některé nové proměnné.

- `Ni` vektor délek segmentů impulsové odezvy
- `mez` hraniční délka impulsové odezvy
- `PocetVlaken` počet vláken potřebných k výpočtu
- `PoradiVlakna` index aktuálního vlákna

Proměnné `tL`, `tR`, `hi` a `H` jsou dynamicky tvořeny a proto jsou v Matlabu definované jako buňky (*cell array*).

Pro správné rozdělení impulsové odezvy podle tabulky 1.1, je potřeba délku impulsové odezvy vždy doplnit nulami na délku $2^n - 2$. Součin proměnných `mez` a `N` představuje hraniční délku, která je výše zmíněné velikosti. Pro délku impulsové odezvy menší než `N` je nová délka doplněna pouze do velikosti `N`. Pokud je délka impulsové odezvy menší než hraniční délka, doplní se nulami do hraniční délky. Pokud je délka větší, je impulsová odezva doplněna nulami do další vhodné mezní hraniční délky.

```
% vypocet hranice pro spravne doplneni delky impulsove odezvy
ex = nextpow2(Nn)
mez = 2^ex - 2
if mez <= Nn
    mez = 2^(ex + 1) - 2
end
if ex == 0
    mez = 2
end
% pro odezvy kratsi nez N
if Nn == 0
    mez = 1
end
```

```
% osetreni ruzne delky impulsove odezvy
% pokud je delka odezvy vetsi nez hranicni delka odezvy
if NN_n > mez*N
% I_N novy celociselny nasobek realne delky odezvy pro
   doplneni nulami
    I_N= fix(NN_n/N) + (mez + 2);
elseif NN_n <= mez*N  % kratsi mezni hranice
     I_N= fix(NN_n/N) + (mez - Nn);
end
```

Po načtení odezvy je nutné připravit několik dalších proměnných, které vstupují do hlavní smyčky algoritmu. Do vektoru `Ni` jsou uloženy délky segmentů impulsové odezvy. Dále je nutné dynamicky vytvořit zásobníky `tL`, `tR` k pojmutí přesahů po výpočtech konvolucí.

V hlavní smyčce algoritmu je pak stěžejní funkce `fzerofft`, která se dynamicky volá pro různý počet výpočetních vláken. Pro dvoukanálovou realizaci funkce `fzerofft2ch` vrací naplněné výstupní zásobníky pravého a levého kanálu aktuálním výpočtem vlákna a také zásobníky uchovávající přesahy konvolucí.

Vstupními parametry funkce jsou komplexní vstupní zásobník `x_max`, výstupní zásobníky `y_maxL`, `y_maxR`, délka nejmenšího segmentu `N`, délka aktuálního segmentu výpočtu `Nk`, aktuální obsah dočasných zásobníků `tL_n`, `tR_n`, aktuální segment kmitočtové charakteristiky impulsové odezvy `Hk`, počet vláken algoritmu, pořadí aktuálního vlákna a vektor délek segmentů impulsové odezvy `Ni`. V ukázce hlavní smyčky algoritmu jsou namísto návratových hodnot a vstupních parametrů použité pomocné proměnné `outs` a `ins`. Kompletní zdrojové kódy a funkce jsou k dispozici na přiloženém médiu.

Ve funkci `fzerofft2ch` je nejzajímavější částí synchronizace výsledků. Pomocí proměnných `PoradiVlakna`, `PocetVlaken` a délky výstupního zásobníku `Ni` v násobních `N` se v cyklu počítá pozice k uložení výsledku konvoluce.

```
for i=1:I
    % segmentace vstupu
    x_maxL(xMax-Ni(1)+1:xMax) = xL((i-1)*N+1:i*N);
    x_maxR(xMax-Ni(1)+1:xMax) = xR((i-1)*N+1:i*N);
    x_max = complex(x_maxL, x_maxR);

    if PocetVlaken ~= 1
                    [outs] = fzerofft2ch(ins);      % 1 vlakno
                    [outs] = fzerofft2ch(ins);      % 2 vlakno
        % n-te vlakno
        if Nhmax ~= 512
            for kk=1: expoend %expoend=nextpow2(Ni(end)/N);
                n = kk * 2;
                k = 2^kk;
                if mod(i, k) == 0
                    [outs] = fzerofft2ch(ins);
                    [outs] = fzerofft2ch(ins);
                end
            end
        end
    else
       [outs] = fzerofft2ch(ins);      % 1 vlakno
    end
    % vysledny synchronizovany vystup
    % vybiram N vzorku z vystupniho zasobniku
    yL((i-1) * N + 1 : i * N) = y_maxL(1 : N);
    yR((i-1) * N + 1 : i * N) = y_maxR(1 : N);
    % posunuti vstupnich a vystupnich zasobniku o N vzorku s
       kazdym i
    x_maxL(1 : xMax - N) = x_maxL(N + 1 : xMax);
    y_maxL(1 : yMax - N) = y_maxL(N + 1 : yMax);
    % zamezeni pricitani predesleho segmentu
    y_maxL(yMax - N + 1 : yMax) = zeros(1, N);
    x_maxR(1 : xMax - N) = x_maxR(N + 1 : xMax);
    y_maxR(1 : yMax - N) = y_maxR(N + 1 : yMax);
    % zamezeni pricitani predesleho segmentu
    y_maxR(yMax - N + 1 : yMax) = zeros(1, N);
end;
```

```
function[y_maxL,y_maxR,tL_n,tR_n]=fzerofft2ch(x_max,y_maxL,y_maxR,N,Nk,tL_n,tR_n,Hk,PocetVlaken,
   PoradiVlakna,Ni)
xMax = length(x_max);
yMax = length(y_maxL);
Nn = yMax/N          % delka vystupniho zasobniku v nasobcich N
X_n = fft((x_max(xMax - Nk + 1 : xMax)), 2 * Nk);
Y_n = X_n .* Hk;
y_n = fft(conj(Y_n), 2 * Nk)/2/Nk;
yL_n = real(y_n(1 : Nk)) + tL_n;
yR_n = imag(y_n(1 : Nk)) + tR_n;
tL_n = real(y_n(Nk + 1 : 2 * Nk));
tR_n = imag(y_n(Nk + 1 : 2 * Nk));
B = 1;
segmentvector = Ni;
for j=1:PocetVlaken
    % synchronizace
    if j == PoradiVlakna
        y_maxL(yMax-(Nn-(B-Nk/N))*N+1:yMax-(Nn-B)*N)=y_maxL(yMax-(Nn-(B-Nk/N))*N+1:yMax-(Nn-B)*N)
           +yL_n;
        y_maxR(yMax-(Nn-(B-Nk/N))*N+1:yMax-(Nn-B)*N)=y_maxR(yMax-(Nn-(B-Nk/N))*N+1:yMax-(Nn-B)*N)
           -yR_n;
    end
    % priprav pozici ulozeni
    B = B + (segmentvector(j)/N);
end
```

3 VÝVOJOVÁ PROSTŘEDÍ PRO IMPLEMENTACI VST MODULŮ

Číslicové zpracování signálů se v neposlední řadě také hojně využívá v audiotechnice, kde původně akustické signály jsou převedeny do číslicové podoby a následně zpracovávány. V praxi se pro zpracování signálů používá počítač s vhodným softwarem. Ten má především nahradit analogové audio procesory, používané ke zpracování akustických signálů. Takový software, který známe například jako reverb, kompresor nebo ekvalizér, je pojmenovaný jako zásuvný modul a vyskytuje se v několika formátech (VST, AU, LADSPA, RTAS, AAX). V této kapitole bude nastíněno několik možností implementace VST zásuvných modulů na operačním systému Windows pomocí programovacího jazyka C++.

3.1 Rozhraní VST a zásuvné moduly

Zásuvný modul neboli plugin je software, který pro svůj chod využívá primárně hostitelskou aplikaci např. DAW.[1] Tato skutečnost přináší hned několik zásadních výhod. Co se týče hudebních aplikací, hostitelská aplikace se stará o čtení vstupních dat a odebírání dat výstupních hudebního signálu. Zásuvný modul se tedy pouze zaměřuje na zpracování tohoto signálu.

VST (Virtual Studio Technology) je jedním z výše uvedených formátů, ve kterých zásuvné moduly existují. Jedná se tedy o softwarové rozhraní, pomocí něhož komunikuje hostitelská aplikace se zásuvnými moduly. Formát VST je kompatibilní jak s operačním systémem Windows tak se systémy vycházejících z Unixu. Formát VST byl poprvé představen společností Steinberg. Zásuvné moduly jsou dostupné v placených nebo volně dostupných verzích. Čerpáno z [7]. Mezi klíčové vlastnosti VST modulů patří:

- široká podpora hostitelských aplikací
- nízká náročnost na CPU
- nízká časová prodleva při přepnutí mezi bankami, nástroji nebo presety
- možnost dosažení nízkého procesního zpoždění
- dobrá zvuková kvalita
- rychlé načtení modulu

[1] Digital Audio Workstation je program pro nahrávání, editaci, mixáž a mastering audio signálů.

3.2 Výběr vhodného vývojového prostředí

3.2.1 Standardní vývojový kit VST

Na oficiálních webových stránkách `http://www.steinberg.net` poskytuje společnost Steinberg volně dostupný vývojářský kit (VSTSDK[2]) pro tvorbu VST zásuvných modulů. Původně ve verzi 2.0, aktuální verzí je nyní verze 3.6.5.. Velké množství zásuvných modulů vyskytujících se na trhu je vytvořeno právě tímto vývojářským kitem.

Pro vývoj zásuvných modulů pomocí standardního vývojového kitu je nezbytné:

- zvolit vhodné vývojové prostředí (např. Microsoft Visual Studio)
- stáhnout potřebné SDK soubory volně dostupné z: `http://www.steinberg.net/en/company/developers.html`
- ve vývojovém prostředí zadat správnou cestu k SDK souborům

Práce vývojáře spočívá v načtení SDK do vhodného vývojového prostředí a volání funkcí, které získávají vstupní a výstupní data. Dále pak vývojář sestaví kód zpracovávající audio signál. Pro jednodušší moduly je možné se zcela vyhnout implementaci grafického uživatelského rozhraní. Finální zásuvný modul pak využívá uživatelské prostředí hostitelské aplikace. Čerpáno z [6].

Testování a přeložení demo zásuvného modulu proběhlo úspěšně na operačním systému Windows 7 ve vývojovém prostředí *Microsoft Visual Studio 2015*. Po přeložení byl vytvořen zásuvný modul VST verze 2.4 jako soubor s příponou `.dll`. Ten byl dále načten a testován v hostitelské aplikaci Reaper a Sound Forge Pro 10.

3.2.2 Využití *frameworku* JUCE

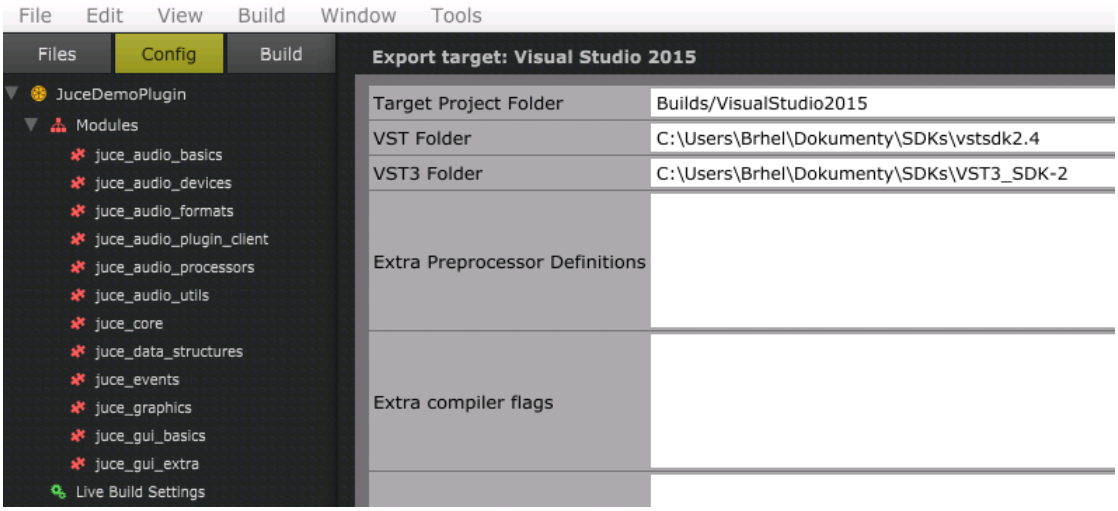
Následující text vychází ze zdrojů [9] a [10]. Pro vývoj VST pluginů můžeme také využít *frameworku* JUCE od společnosti ROLI. Tento multiplatformní *framework*, napsaný jako svobodný kód vznikl v roce 2004 a jeho hlavním cílem je umožňovat vývojářům napsat takový zdrojový kód, který bude spuštěn na všech operačních systémech. Mezi hlavní výhody *frameworku* JUCE patří:

- napsání jednoho společného kódu až pro čtyři různé formáty zásuvných modulů (VST, AU, AAX, RTAS)
- snadný a intuitivní vývoj GUI[3] ve srovnání s předešlou metodou
- dostupnost knihoven
- spustitelnost na všech operačních systémech

[2] VST Software Developement Kit

[3] Graphical User Interface - Grafické uživatelské rozhraní

Po založení nového projektu se vývojáři zobrazí nastavení prostředí. Zde je potřeba zadat jméno zásuvného modulu, jméno vývojáře případně společnosti a nastavit počet vstupních a výstupních kanálů vyvíjeného zásuvného modulu. Pro správnou kompilaci se musí správně zadat cesta k adresáři SDK. Juce umožňuje zvolit cestu jak k novější verzi VST-3 tak ke starší verzi VST-2.



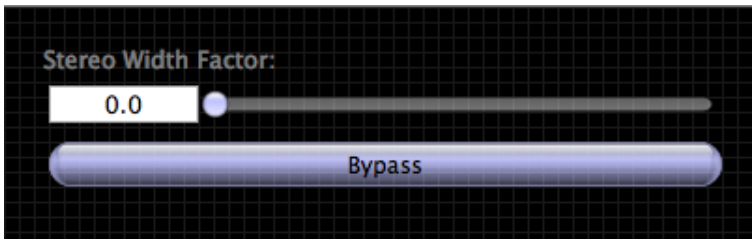
Obr. 3.1: Nastavení cesty k souborům VST ve *frameworku* JUCE

JUCE není schopen samostatně překládat kód, proto využívá překladače IDE (Integrated Development Environment), ve kterém následně kód přeloží a spustí. Další výhodou JUCE je spolupráce s IDE platformy Windows, OS X či Linux. Pro operační systém Windows bylo jako IDE zvoleno Microsoft Visual Studio 2015.

Je doporučeno psát kód v editoru JUCE a poté vždy přeložit a spustit v IDE. Pro náročnější aplikace je vhodné psát uživatelský kód do separátních souborů. Je například efektivní, pokud se funkce, zpracovávající audio signál, vyskytuje ve zvláštním `cpp` a `h` souboru. Vývojář tak má lepší přehled o funkcích, starajících se o zpracování signálu, čtení vstupnich dat a grafiku.

Jak již bylo zmíněno, jednou z dalších výhod použití *frameworku* JUCE spočívá ve vývoji GUI. Ve frameworku je možnost přidat nový GUI komponent, kde se jeho třídě přiřadí třídy bázové. V záložce *subcomponents* se dá využít několik přednastavených komponent, které jsou pro vývoj uživatelského prostředí dostačující. Pro pokročilejší grafiku je možné použít vlastní obrázky a ikony.

V hlavičkovém souboru dědí hlavní třída vlastnosti bázových tříd. Také jsou zde deklarovány metody a proměnné uživatele. Pokud tvoříme grafiku pomocí myši v záložce subcomponents, proměnné se ve hlavičkovém souboru vytvoří automaticky s defaultním názvem. Podobně se také vytvoří v cpp souboru, kde se dále připojí každemu komponentu jeho vlastnosti a funkce.



Obr. 3.2: Příklad GUI demo pluginu vytvořeného přednastavenými komponenty v JUCE. Převzato z [10]

Autoři označují původní *framework* jako *The Introjucer*. V říjnu roku 2015 autoři představili novou verzi JUCE *The Projucer*. Mezi hlavní funkci tohoto dospělejšího nástupce patří vlastní překladač, který okamžitě a automaticky zobrazuje uživatelský kód v reálném čase. Tato funkce má tedy velký potenciál při budoucím vývoji zásuvných modulů, zejména pro efektivní testování nových modulů.

Vlastní testování proběhlo na operačním systému Windows 7. Jako IDE bylo použito *Mcrosoft Visual Studio*. Demo projekt byl správně přeložen a výsledný modul otestován v aplikaci Reaper a Sound Forge Pro 10.

3.2.3 Java jako vývojové prostředí

Při tvorbě následujícího textu bylo čerpáno z [8]. Pro implementaci zásuvných modulů lze také využít jVstwRapper a programovacího jazyku Java. Díky němu lze vytvářet zásuvné moduly formátu VST (2.4), Audio Unit (AU) a LADSPA na všech operačních systémech. Zásuvný modul vytvořený pomocí jVstwRapperu lze také spustit na všech operačních systémech a jeden napsaný kód je identický pro všechny výše zmíněné formáty zásuvného modulu. O vstupní a výstupní data se stará hostitelská aplikace, vývojář pouze implementuje prostředí *wrapperu* a algoritmus zpracovávající audio signál. Vývoj uživatelského prostředí je založen pouze na Javě.

Wrapper je sám o sobě zkompilovaný binární kód (.so na operačním systému Linux, .vst a .component na operačním systému OSX a .dll na operačním systému Windows). Hostitelská aplikace předává vzorky *wrapperu*, který volá JVM (Java Virtual Machine) a také VST zásuvný modul napsaný v Javě. Z toho plyne, že pro spuštění zásuvného modulu je nutná instalace JVM. Jednou napsáný kód lze použít bez nutnosti kompilace na více operačních systémech.[11].

Instalace jVstwRapperu:

- pro instalaci je zapotřebí stáhnout Javu z `http://www.java.com`
- stáhnout a nainstalovat Java Virtual Machine
- stáhnout a nainstalovat nejnovější verzi jVSTwRapperu dostupnou z `http://sourceforge.net/projects/jvstwrapper/files/`

Pro tvorbu grafického rozhraní lze použít knihovny Swing nebo JavaFX.

Výkon

Rychlost a výkonnost zásuvných modulu implementovaných v Javě jsou srovnatelné s implementací v C++. Zdroj [8] uvádí příklad srovnání stejného modulu napsaného v jazyce C++ a Java. Modul napsaný v jazyce Java vykazoval stejnou výpočetní naročnost, jako modul napsaný v jazyce C++.

3.2.4 Porovnání vývojových prostředí

Tabulka 3.1 poskytuje přehled a srovnává diskutovaná vývojová prostředí. Pro vývoj zásuvných modulů na operačním systému Windows lze využít kterékoliv prostředí. Rychlost zásuvného modulu by neměla být ovlivněna zvoleným programovacím jazykem, proto může vývojář zvolit programovací jazyk podle svých preferencí. Zvolený jazyk však určuje vývojové prostředí. Pro vývoj složitějšího GUI je vhodné využít *frameworku* JUCE. Tato volba dále nese pro vývojáře výhody jako přehledná a široká dostupnost knihoven a funkcí nebo velká komunita uživatelů. S dospělejší verzí JUCE *The Projucer* lze efektivně testovat navržené moduly, bez nutnosti spuštění v hostitelské aplikaci.

Pokud by implementace modulu vyžadovala zvláštní požadavky na grafické uživatelské prostředí, bude nejvhodnější volbou *framework* JUCE. Pro jednoduché moduly bez nutnosti implementace grafického uživatelského prostředí je vhodné využít standardní vývojový kit VST, popř. vývojové prostředí v Javě.

Tab. 3.1: Srovnání jednotlivých prostředí

Tabulka

	standartní kit VST	JUCE	jVSTwRapper
programovací jazyk	C++	C++	JAVA
podporované formáty	VST	VST,AU,AAX,RTAS	VST,AU,LADSPA
podpora GUI	standardní	pokročilý editor	standardní
podporované OS	Windows,OSX,Linux	Windows,OSX,Linux	Windows,OSX,Linux
knihovny	standardní	rozšířené	standardní
návody, tutoriály	průměrné,základní	průměrné,vzrůstající	průměrné
komunita	průměrná	široká	menšinová

4 REALIZACE FIR PROCESORU JAKO VST ZÁSUVNÝ MODUL

Vývoj zásuvného modulu probíhal ve *frameworku* JUCE. Pro tohle vývojové prostředí jsem se rozhodl díky dostupnosti tutoriálů a široké API. [1] Práce s *frameworkem* se postupem času projevila jako velmi intuitivní. Zásuvný modul jsem pojmenoval *FastFIR*.

Implementace zásuvného modulu je v podstatě založena na dvou hlavních třídách `FastFirAudioProcessorEditor` a `FastFirAudioProcessor`. První z nich obstarává GUI a jeho aktualizaci. Třídu lze najít v souborech `PluginEditor.cpp` a `PluginEditor.h` v projektu. JUCE umožňuje ve třídě snadno přidávat nové ovládací prvky a vytvořit jednoduché GUI. Druhá hlavní třída `FastFirAudioProcessor` pracuje se vstupními daty a hlavní výpočty jsou implementovány právě v této třídě. Tato třída se vyskytuje v souborech `PluginProcessor.cpp` a `PluginProcessor.h`. Obě třídy jsou vzájemně propojeny. Stěžejní funkcí třídy `FastFirAudioProcessor` je funkce `FastFirAudioProcessor::processBlock`, ve které dochází k segmentaci a desegmentaci zpracovávaného signálu. Tato funkce je neustále volána ihned po načtení modulu v hostitelské aplikaci.

4.1 Realizace vstupních a výstupních zásobníků

Vstupní a výstupní data jsou zpracovávána po blocích. Nová data se nejdříve načtou do pole[2] (bloku) a jsou ve funkci `processBlock` zpracována. Tato data jsou poté stejným polem odeslaná na výstup. V prostředí JUCE je takové pole realizováno pomocí třídy `AudioSampleBuffer`. Následující část kódu představuje, jak lze na vstupní data přistupovat.

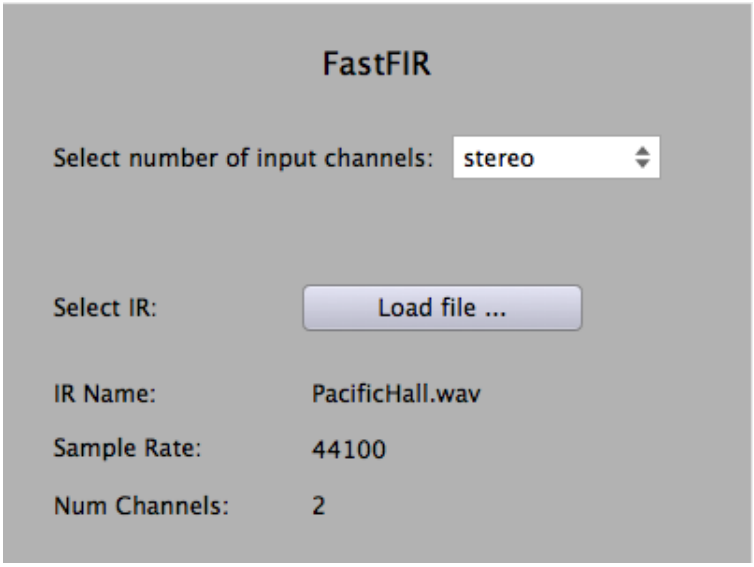
```
void FastFirAudioProcessor::processBlock (AudioSampleBuffer&
   buffer)
{
    float* ld = buffer.getWritePointer(0); // levy kanal
    float* rd = buffer.getWritePointer(1); // pravy kanal
}
```

[1] Application Programming Interface - jedná se o rozhraní pro vývoj aplikací. Tento termín označuje soubor tříd, funkcí a protokolů, které může vývojář používat.

[2] Pojem pole představuje v jazyce C zásobník.

4.2 Příprava a ovládání GUI

Jednoduché GUI lze snadno vytvořit ve třídě `FastFirAudioProcessorEditor` pomocí tříd představující ovládací prvky, které *framework* JUCE poskytuje. Například popisky jsou zobrazeny pomocí třídy `Label` a pro tlačítko sloužící k otevření impulsové odezvy je použita třída `TextButton`.



Obr. 4.1: GUI zásuvného modulu FastFIR

Při otevření zásuvného modulu v hostitelské aplikaci uživatel načte impulsovou odezvu tlačítkem *Load file* a zvolí počet kanálů vstupního signálu. Po načtení impulsové odezvy se zobrazí její základní informace. Název souboru, vzorkovací kmitočet a počet kanálů impulsové odezvy.

4.3 Implementace algoritmu ve vývojovém prostředí

Vývoj aplikace v jazyce C++ vychází z velké části z implemenace v Matlabu. Díky možnostem, které jazyk C++ nabízí a snaze minimalizovat nároky na paměť jsou některé funkce a operace s pamětí nepatrně odlišné.

Jedná se především o vytvoření polí maximální délky, dočasného pole a pole pro impulsovou odezvu. Tato pole jsou realizována jako zdvojená (komplexní) cyklická pole. Práce s daty probíhá následujícím způsobem. Hodnoty se v polích ukládají/čtou zleva s tím, že index ukazuje na pozici zápisu/čtení. Pokud hodnota indexu přesáhne maxima, je vynulována a index tak ukazuje opět na začátek pole. Pro vytvoření komplexních polí lze použít strukturu `Complex`, která je definována ve

třídě `FFT` z *frameworku* JUCE. Vstupní a výstupní komplexní pole maximální délky vytváří následující funkce.

```
void FastFirAudioProcessor::createInputOutputBuffers()
{
    counter = 0;
    delete m_X;
    // velikost vstupniho pole v nasobcich N
    m_xSize = pow(2, m_lastSliceIdx);
    // absolutni velikost vstupniho pole
    int xSize = m_xSize * m_sizeBlock;
    m_X = new FFT::Complex[xSize];
    // pocatecni podminky - vynuluj
    zeromem(m_X, xSize*sizeof(FFT::Complex));

    delete m_Y;
    // absolutni velikost vystupniho pole
    m_ySize = m_NMax/2 + pow(2, m_lastSliceIdx)*m_sizeBlock;
    m_Y = new FFT::Complex[m_ySize];
    // pocatecni podminky - vynuluj
    zeromem(m_Y, m_ySize*sizeof(FFT::Complex));
}
```

Protože se vstupní data z funkce `processBlock` nacházejí v poli třídy `AudioSampleBuffer`, je potřeba je před výpočtem FFT převést do komplexních polí. Tento proces obstarává funkce `FastFirAudioProcessor::ASBuffer2Complex`.

4.3.1 Načtení impulsové odezvy

Impulsovou odezvu je potřeba načíst ve formátu *wav*. Následující část kódu popisuje načtení hodnot odezvy do pole typu `AudioSampleBuffer`.

```
// vytvoreni pole
AudioSampleBuffer buffer(numChannels, bufSize);
buffer.clear(); // vynuluj
// zapis do pole
wavReader->read(&buffer, 0, bufSize, 0, true, true);
// zpracovani odezvy
m_processor.setWav(buffer);
delete wavReader;
```

Po načtení hodnot se také spustí funkce `FastFirAudioProcessor::setWav(const AudioSampleBuffer& buffer)`. Tím dochází ke komunikaci mezi třídou `FastFirAudioProcessorEditor` a `FastFirAudioProcessor`. Ve funkci jsou hodnoty odezvy převedeny do komplexního pole a také je vytvořeno pole dočasné. Výpočet obrazu impulsové odezvy pomocí FFT je proveden ve funkci `FastFirAudioProcessor::calcH(const FFT::Complex* h)`, která se volá ve funkci `setWav()`. Hodnoty impulsové odezvy jsou nejdříve po segmentech načteny do pole `FFT::Complex* inp`, které následovně vstupuje do funkce FFT.

```
fft.perform(inp, out);  // dopredna FFT
// ulozeni vysledku do m_H
memcpy(m_H+2*offset, out, sizeof(out));
```

Obraz impulsové odezvy je obsažen v `FFT::Complex* out` a je uložen do pole `m_H`, kde je nachystán k Hadamardově součinu.

4.3.2 Zpracování signálů

Hlavní výpočet algoritmu představuje funkce FastFirAudioProcessor::calcOutput(const int idxX, const int idxH, const int idxY, const int k), kde jsou vstupními parametry aktuálně nastavené indexy polí. Ve funkci je výpočítán obraz vstupního segmentu a proveden Hadamardův součin spolu s dalšími potřebnými výpočty. Výsledky jsou přičteny a uloženy do pole `m_Y`, ze kterého jsou pak ve funkci `processBlock()` přehrány na výstupu.

```
for (int i = 0; i < buffer.getNumSamples(); ++i)
        {
            ld[i] = m_Y[idxPlay+i].r;    // levy kanal
            rd[i] = m_Y[idxPlay+i].i;    // pravy kanal
        }
```

Na základě zvoleného počtu vstupních kanálů uživatelem v GUI jsou vzorky rozdílně vyčítány na výstupu. Monofonní výstup je tvořen vyčtením reálných částí komplexního pole `m_Y` do levého i pravého kanálu výstupu. Část kódu výše ukazuje stereofonní výstup.

5 TESTOVÁNÍ ZÁSUVNÉHO MODULU

Algoritmus, na kterém je zásuvný modul založen, byl opakovaně testován již při vývoji v prostředí *Matlab*. Další testování probíhalo jen s výsledným zásuvným modulem. Modul byl vytvořen jak pro operační systém Windows tak pro OS X. Z časových důvodů však testování na systému Windows neproběhlo. Hostitelskou aplikací pro testování je aplikace *Reaper*.

Kromě následujících testů také probíhalo průběžné testování mezi výpočty algoritmu z *Matlabu* a algoritmu vyvíjeného v C++. Výsledky funkcí byly ukládány do textových souborů a ty byly porovnávány. Odchylky se pohybovaly v řádech 10^{-7}, což bylo způsobené chybou vzniklou zaukrouhlováním nebo rozdílnými funkcemi pro čtení a zápis do formátu *wav*.

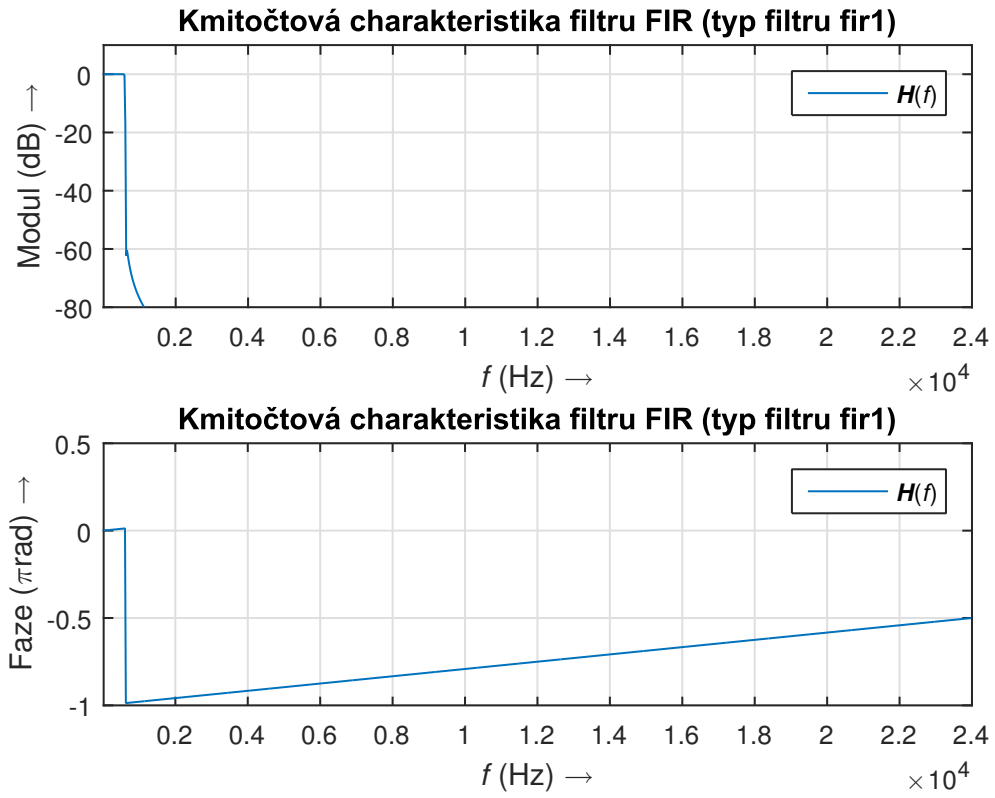
5.1 Příprava testovacích signálů

Pro testování byl použit vstupní signál vygenerován jako *wav* soubor v prostředí *Matlab*. Jedná se o dvoukanálový harmonický signál s těmito parametry:

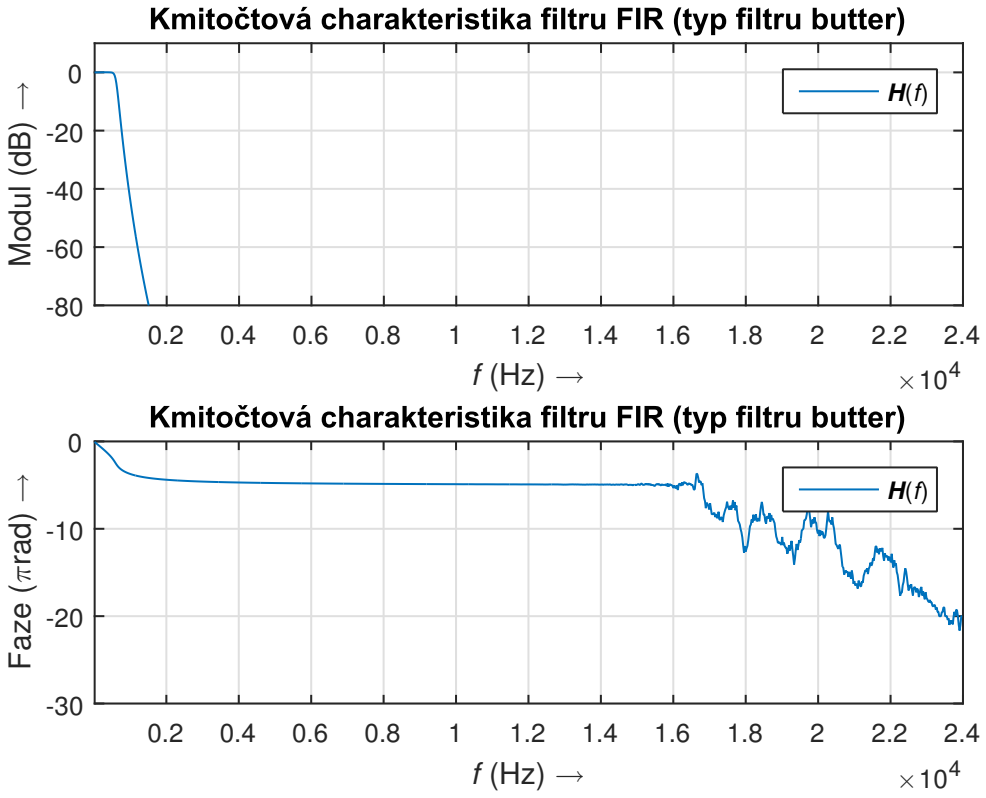
```
Fvz = 48000; % vzorkovaci frekvence
fm = 600;        % mezni kmitocet
fmr = 2*fm/Fvz;
f1 = 100;
f2 = 800;
td = 70/f1;
%vstupni  signal x
Nt = td*Fvz;
n = linspace(0, Nt-1, Nt);
A = 0.5;
B = 0.3;

x= B*cos(2*pi*2*f1*n/Fvz+3*pi/4);
xv=A*cos(2*pi*2*f1*n/Fvz);
x1=xv+A*cos(2*pi*3*f2*n/Fvz);        % kanal1
x2=x+B*cos(2*pi*3*f2*n/Fvz);         % kanal2
% typ impulsove odezvy
h = fir1(4095, fmr);     % dolni propust maximalni delky
%[b,a]=butter(10,fmr);
%h=impz(b,a,4096);
%h = h';                 % otoceni radku se sloupci
```

Impulsová odezva je definována buďto jako symetrická (*fir1*) nebo nesymetrická (*butter*) délky 4096 vzorků. Odezva byla rovněž vygenerována jako *wav* soubor pro načtení v modulu. Obrázky 5.1 a 5.2 zobrazují kmitočtovou charakteristiku obou filtrů.



Obr. 5.1: Kmitočtová charakteristika filtru typu fir1



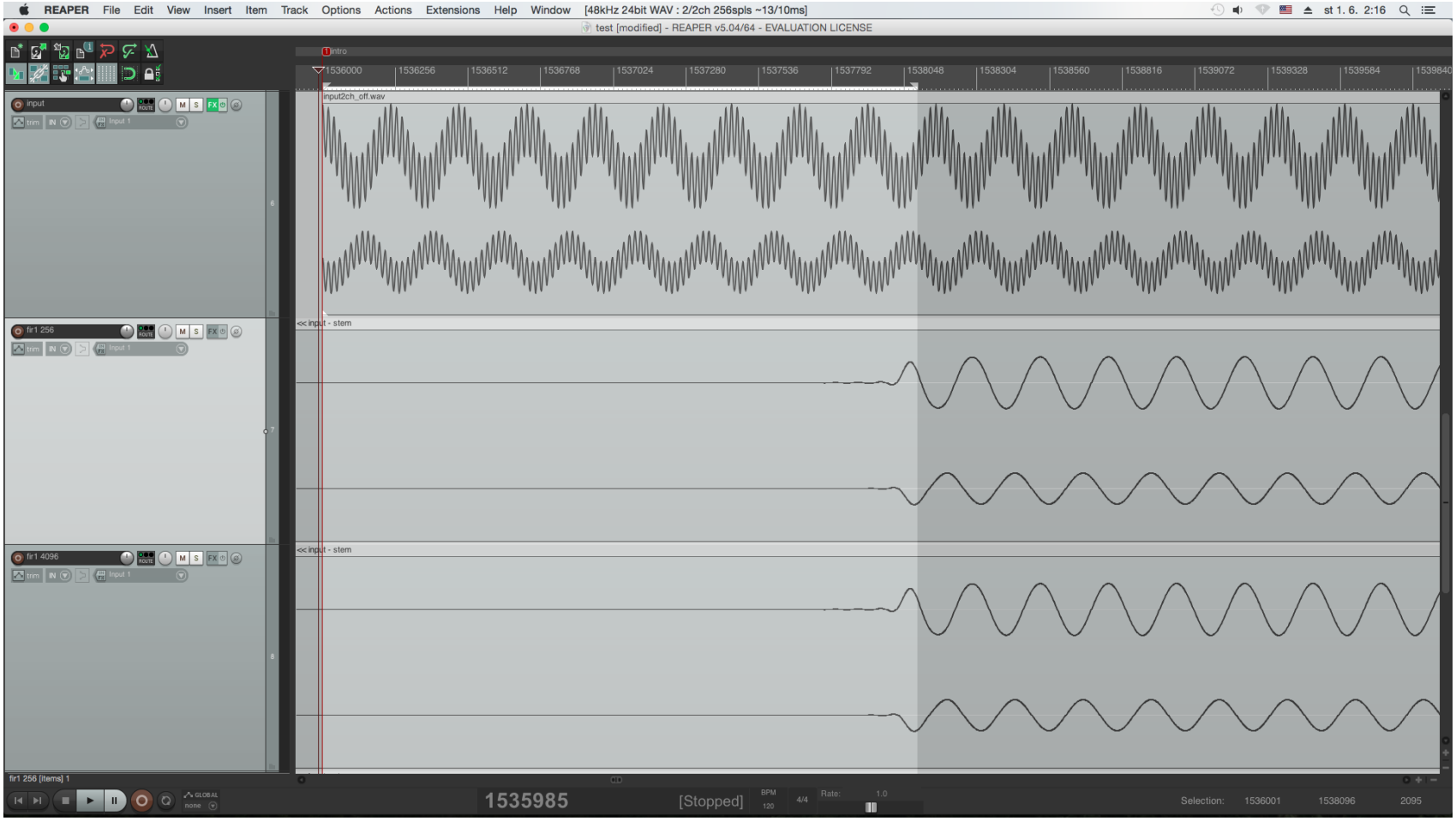
Obr. 5.2: Kmitočtová charakteristika filtru typu butter

5.2 Testování v hostitelské aplikaci

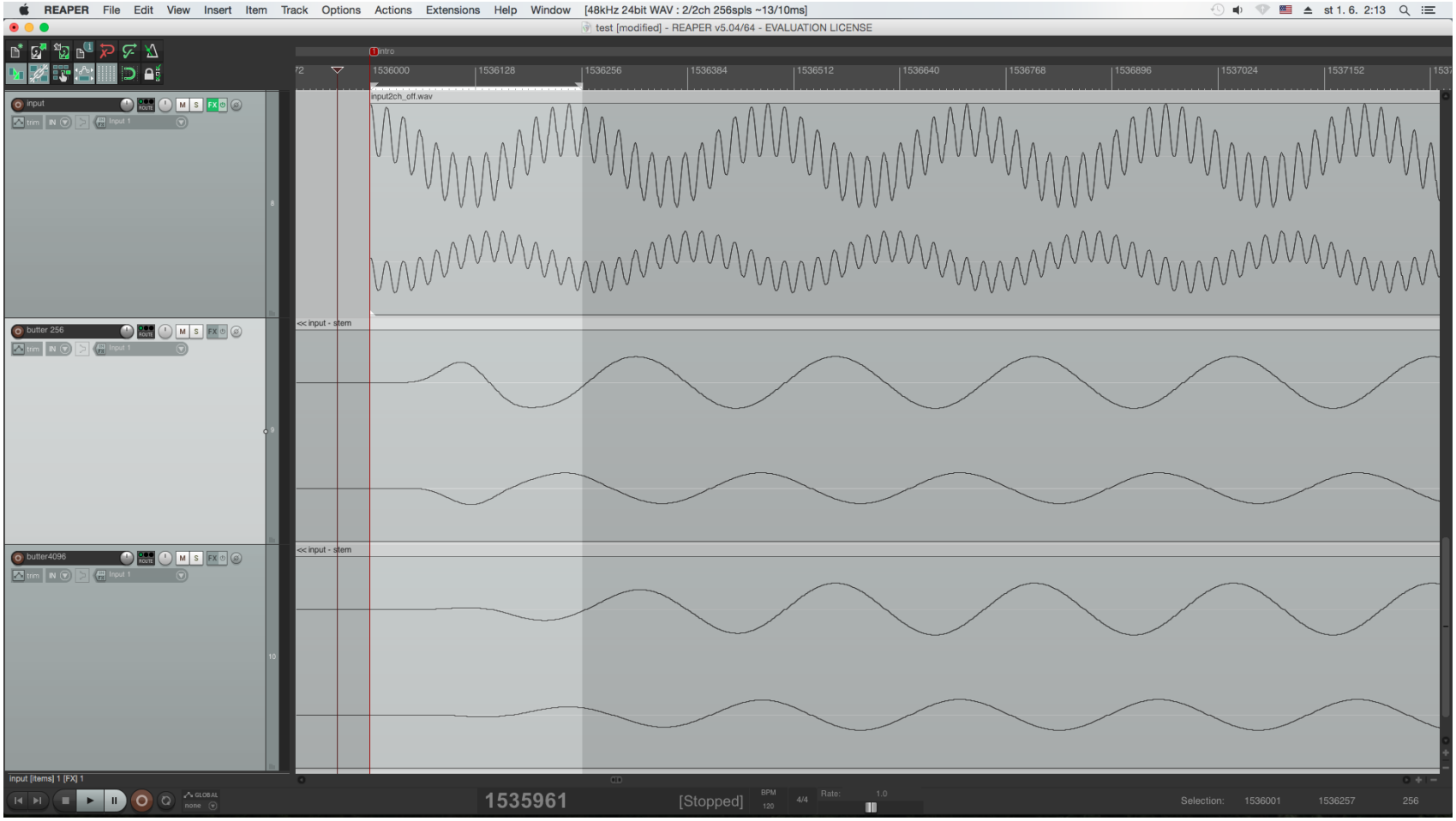
Zásuvný modul byl načten v nové stopě v *DAW Reaper*. Do stopy byl přidán *wav* soubor *input2ch_off.wav* se vstupním signálem. Ve stopě byl otevřen zásuvný modul *FastFIR* a byla vybrána požadovaná impulsová odezva. Pro každou odezvu a velikost nejmenšího segmentu (audio buffer) byla stopa s modulem tzv. *vyrendrována*. Z křivek nebo poslechem lze úspěšně usoudit, že odfiltrovaný signál neobsahuje vysoké kmitočty. Nově vzniklá stopa byla následovně časově porovnána s originálem. Zobrazení osy ve vzorcích napomohlo k lepšímu porovnání signálů. Funkce *rendrování* však probíhá offline, proto je testování procesního zpoždění při zpracování signálu vhodnější při živé reprodukci.

Na obrázku 5.5 je zobrazen časový průběh nahrávání dvou stop, kde stopa *guitar_clean* představuje přímý signál z kytarového multiefektu a stopa *guitar_plugin* představuje stejný přímý signál nahráván současně, ovšem s modulem *FastFIR* přidaným do nahrávací cesty.

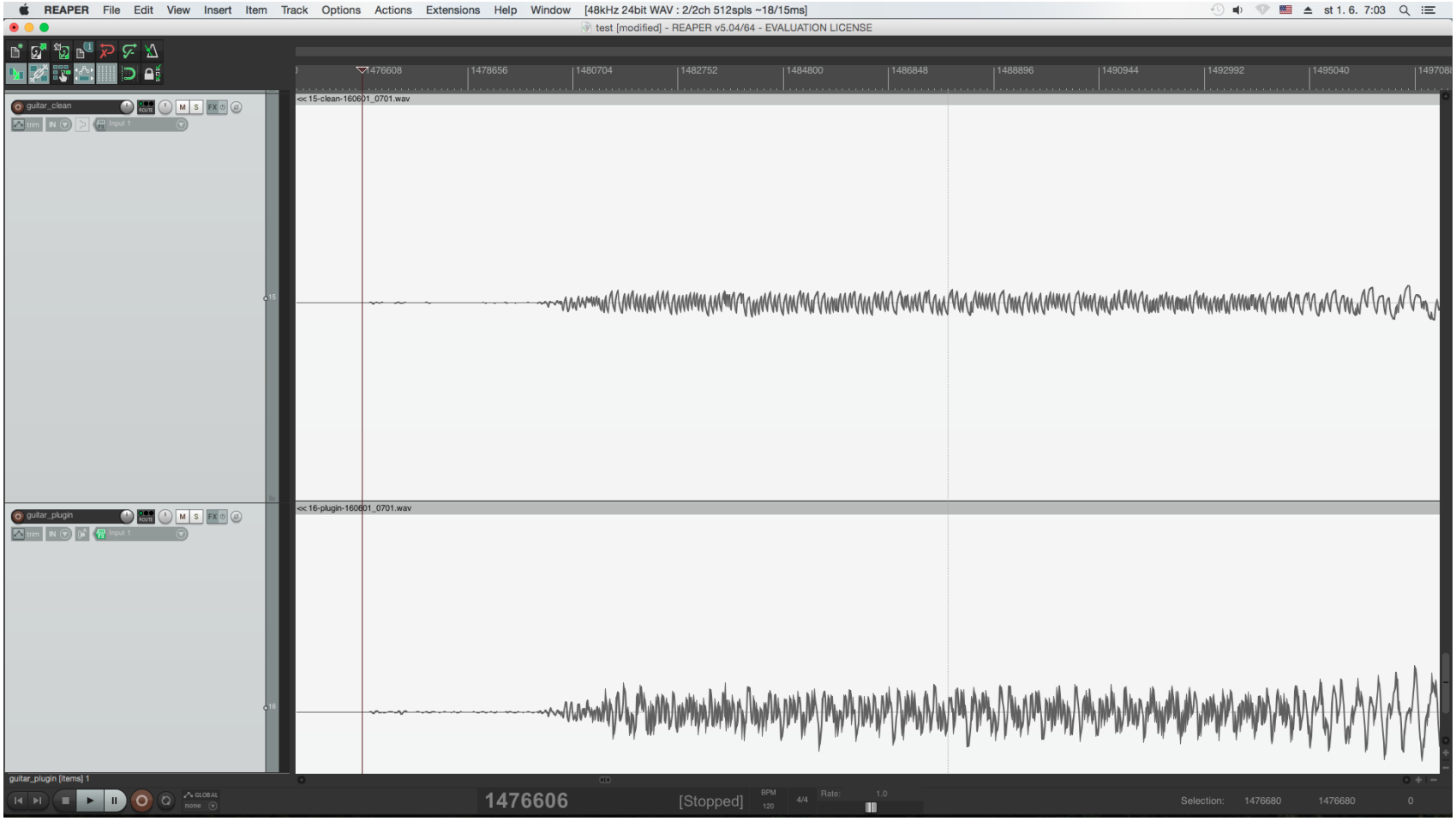
Testování běžně dostupných impulsových odezev vykazuje tato fakta. Pro krátké impulsové odezvy asi do 1000 vzorků lze pro správnou funkčnost nastavit velikost nejkratšího segmentu na hodnoty menší, například 512. Pro dlouhé impulsové odezvy poslechových prostorů je nutné nastavit velikost nejkratšího segmentu alespoň na 1024 vzorků.



Obr. 5.3: Filtrace signálu filtrem typu fir1 pro velikost nejmenšího segmentu 256 a 4096 vzorků



Obr. 5.4: Filtrace signálu filtrem typu butter pro velikost nejmenšího segmentu 256 a 4096 vzorků



Obr. 5.5: Souběžné nahrávání přímé cesty z kytarového multiefektu a cesty o přidaný zásuvný modul

6 ZÁVĚR

Náplní této práce byla realizace univerzálního FIR procesoru pro souběžné zpracování dvou signálů s minimalizovaným procesním zpožděním. V teoretické části byly nejdříve popsány principy FIR systémů a jejich realizace. Pro praktickou realizaci FIR procesoru s delší impulsovou odezvou je nezbytné využít rychlé Fourierovy transformace v kmitočtové oblasti.

Algoritmus pro souběžnou filtraci dvou signálů je založen na rychlé konvoluci s přičtením přesahu. Zpracování dvou signálu současně využívá komplexní vstupní posloupnosti pro výpočet algoritmu FFT. V praktické části se podařilo v Matlabu zrealizovat FIR procesor pro zpracování dvoukanálových posloupností. Funkčnost byla otestována na filtraci dvoukanálového signálu dolní propustí. Chyba výpočtu algoritmu, pohybující se v hodnotách 10^{-15}, byla způsobena přesností interního výpočtu FFT v Matlabu.

Dále byla teoreticky popsána metoda potlačení procesního zpoždění. Tato metoda vychází z rozdělení dlouhé impulsové odezvy do segmentů vhodných délek. Při rozdělení impulsové odezvy na menší segmenty algoritmus nečeká na výpočet konvoluce dlouhé posloupnosti, ale je schopen rychleji vypočítat odezvu nejkratšího segmentu. Dílčí výpočty segmentů různých délek je však potřeba správně synchronizovat. Během výpočtu konvoluce delších segmentů je možné současně vypočítat odpovídající segmenty kratší ve stejném čase v paralelních vláknech. Tato práce však souběžné výpočty neuvažuje. Efektivní realizaci takové úvahy lze docílit vícevláknovým programováním, které by bylo vhodným rozšířením této práce.

Třetí kapitola pojednává o vývoji VST zásuvných modulů. Bylo představeno několik vývojových prostředí pro implementaci VST zásuvných modulů, která byla také odzkoušena. Vývojová prostředí byla porovnávána a doporučena v rámci možností, vlastností a požadavků vývojáře. Jako optimální volbou pro vývoj zásuvných modulů se projevil *framework* JUCE díky široké komunitě uživatelů, dostupnosti knihoven a funkcí a snadné tvorbě grafického uživatelského prostředí.

Pomoci tohoto *frameworku* byla následně provedena realizace funkčního VST zásuvného modulu. Ve výsledku si uživatel tento modul načte v hostitelské aplikaci, načte požadovanou impulsovou odezvu a modul aproximuje systém, charakteristický touto impulsovou odezvou.

LITERATURA

[1] BALÍK, M. *Diskrétní systémy s konečnou impulsovou charakteristikou. Číslicové zpracování akustických signálů.* Brno: 2010. 21 s.

[2] GARDNER, W, G. *Efficient Convolution without Input/Output Delay.* J. Audio Eng. Soc., 1995. [cit. 25. 11. 2015]. Dostupné z: `http://www.cs.ust.hk/mjg_lib/bibs/DPSu/DPSu.Files/Ga95.PDF`.

[3] MIŠUREC, J., SMÉKAL, Z. *Číslicové zpracování signálů.* Brno: Vysoké učení technické v Brně Fakulta elektrotechniky a komunikačních technologií Ústav telekomunikací, 2012. 187 s. ISBN 978-80-214-4448-5.

[4] ZOLZER, U. *DAFX - Digital Audio Effects.* John Willey & Sons Ltd., 2005, ISBN 0-470-84604-6.

[5] INGLE, K, V., PROAKIS, G, J. *Digital Signal Processing Using MATLAB®.* 3rd ed. Stamford, Conn.: Cengage Learning, c2012, xv, 652 s. ISBN 1111427372.

[6] STEINBERG. *VST Plug-in Specification 2.0 SDK.* 1999. 84 s.

[7] ESKEROOT, J. *Impementing a parametric EQ plug-in in C++ using the multi-platform VST specification* : Esej. Luleå : Luleå University of Technology, The School of Music in Piteå, 2003. 43 s. Vedoucí práce Jan Berg.

[8] *Java-Based Audio Plug-Ins* [online]. jvstwrapper.sourceforge.net, [cit. 11.11.2015]. Dostupné z: `http://jvstwrapper.sourceforge.net`.

[9] *Tutorial: Create a basic Audio/MIDI plugin* [online]. juce.com, [cit. 12. 12. 2015]. Dostupné z: `http://www.juce.com/doc/tutorial_code_basic_plugin`.

[10] *JUCE for VST Plugin Development* [online]. redwoodaudio.com, [cit. 6. 10. 2015]. Dostupné z: `http://www.redwoodaudio.net/Tutorials/juce_for_vst_development__intro.html`.

[11] Havel, A. *VST implementace strukturálního modelu HRTF* : Bakalářská práce. Praha : Fakulta elektrotechnická, Katedra počíačové grafiky a interakce, České vysoké učení technické v Praze, 2011. 46 s. Vedoucí práce Ing. František Rund Ph.D..

[12] Zwicker, E., Fastl, H. *Psychoacoustics, Facts and Models.* 2nd edition. Springer-Verlag Berlin, Heidelberg, New York, 1999. ISBN 3-540-65063-6.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

FIR	Finite Impulse Response – konečná impulsová odezva
FIR	Infinite Impulse Response – nekonečná impulsová odezva
DFT	Discrete Fourier Transform – diskrétní Fourierova transformace
FFT	Fast Fourier Transform – rychlá Fourierova transformace
IDE	Integrated Development Environment – integrované vývojové prostředí
GUI	Graphical User Interface – grafické uživatelské prostředí
f_{vz}	vzorkovací kmitočet
SDK	Software Development Kit – sada vývojových nástrojů
VST	Virtual Studio Technology – formát zásuvných modulů
DAW	Digital Audio Workstation – zařízení nebo program pro zpracování audio signálů
CPU	Central Processing Unit – centrální procesorová jednotka

OBSAH PŘILOŽENÉHO MÉDIA

`cti_me.txt` - popis jednotlivých souborů
`Matlab` - scripty z Matlabu
`VST` - VST plugin, zdrojové kódy
`BP_Tomas_Brhel.pdf` - text práce