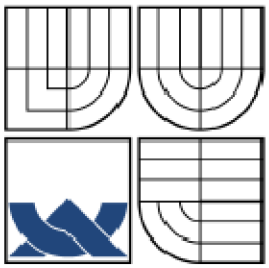
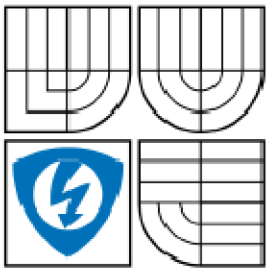


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

VST PLUG-IN PRO VODOZNAČENÍ AUDIO SIGNÁLŮ

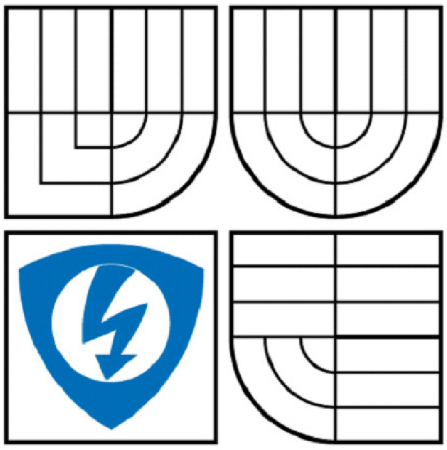
VST PLUG-IN FOR AUDIO WATERMARKING

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — BC. DAVID HENZL
AUTHOR

VEDOUCÍ PRÁCE — ING. RADEK ZEZULA, PH. D.
SUPERVISOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor

Telekomunikační a informační technika

Student: Henzl David Bc. ***ID:*** 89642

Ročník: 2 ***Akademický rok:*** 2007/2008

NÁZEV TÉMATU:

VST Plug-IN pro vodoznačení audio signálů

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte a popište vhodnou metodu vodoznačení audio signálů pro vodoznačení v reálném čase. Tuto metodu implementujte jako tzv. VST Plug-IN, který ve spolupráci s kartou FireWire 410 a ovladači ASIO umožní provádět vkládání vodoznaku do audio signálu v reálném čase.

DOPORUČENÁ LITERATURA:

[1] MILLWARD, S. Sound Synthesis with VST Instrument. 1st edition: PC Publishing, 2002, 277 p. ISBN 1870775732

[2] MILLWARD, S. Users' Guide to Sound Synthesis with VST Instruments. 1st edition: Muska & Lipman/Premier-Trade, 2002. 288 p. ISBN 1929685785

[3] ARNOLD, M; SCHMUCKER, M.; WOLTHUSEN, S. D. Techniques and Applications of Digital Watermarking and Content Protection. Artech House, inc., 2003. 296 p. ISBN 1-58053-111-3

Termín zadání: 11.2.2008 ***Termín odevzdání:*** 28.5.2008

Vedoucí práce: Ing. Radek Zezula, Ph.D.

UPOZORNĚNÍ:

Licenční smlouva

poskytovaná k výkonu práva užít školní dílo

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Bc. David Henzl

Bytem: Jevíčská 23 Letovice

Narozen/a (datum a místo): 6.3.1984/Boskovice

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií

se sídlem Údolní 244/53, 602 00, Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

prof. Ing. Kamil Vrba, CSc.....

(dále jen „nabyvatel“)

Článek. 1

Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - □ diplomová práce
   - □ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako

   .....

   (dále jen VŠKP nebo dílo)

Název VŠKP: VST Plug-IN pro vodoznačení audio signálů

Vedoucí/ školitel VŠKP: Ing. Radek Zezula, Ph.D.

Ústav: Ústav telekomunikací

Datum obhajoby VŠKP:

VŠKP odevzdal autor nabyvateli v*:

- □ tištěné formě – počet exemplářů ……1……..
- □ elektronické formě – počet exemplářů ……1……..

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2

Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - □ ihned po uzavření této smlouvy
   - □ 1 rok po uzavření této smlouvy
   - □ 3 roky po uzavření této smlouvy
   - □ 5 let po uzavření této smlouvy
   - □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3

Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: ……

…….. ……

Nabyvatel Autor

ABSTRAKT

Tato práce se zabývá digitálním zpracováním signálů, možnostem jejich zpracování a zejména pak vodoznačením audio signálů jako možnost zabezpečení autorských práv audio obsahu. V práci jsou nastíněny základní metody vodoznačení dat a možnosti jejich detekce. Pro vytvoření představy o vodoznačení je zde popsána metoda vodoznačení audio dat známá jako Echo Hiding. Tato metoda provádí vodoznačení audio obsahu v časové oblasti zatímco detekce vodoznaků je prováděna v kepstrální oblasti pomocí Fourierovy transformace a autokorelační funkce. Metoda je implementována jako VST plug-in a společně s ovladači ASIO, které minimalizují latenci signálu je možno provádět vodoznačení audio dat v reálném čase.

Cílem první části práce je seznámení se s technologií VST, ASIO ovladači a tvorbou VST plug-in modulů. Druhá část práce se zabývá implementací metod vodoznačení ve spolupráci s technologií VST.

KLÍČOVÁ SLOVA

Vodoznačení audio signálu, Vodoznak, Vodoznačení pomocí echa, VST Plug-IN, ASIO Ovladače, Reálný-čas.

ABSTRACT

This thesis deal with digital signal proccessing methods, possibilities their processing and especially audio signal watermarking like possibility safeguard author's rights of audio content. In this thesis are foreshadoweds basic audio watermarking methods and possibilities of watermark detection. To idea generation about watermarking there is described audio watermarking method known as Echo Hiding. This method embed watermarks to audio content in time-domain while watermark detection is made in kepstral-domain by using Fast Fourier Transform and correlation function. Method is implemented like VST plug - in and along with ASIO drivers that minimize signal latency provides audio signal watermarking in real - time.

Aim of the first volume of this thesis is introduction of VST technology, ASIO driver and creating VST plug–in‘s. Alternative volume of thesis deal with implementation watermarking methods in conjunction with VST technology.

KEYWORDS

Audio signal watermarking, Watermark, Echo Data Hiding, VSTPlug-IN, ASIO Driver, Real-Time.

HENZL D. VST Plug-IN pro vodoznačení audio signálů. Brno: VUT Brno, Ústav Telekomunikací, 2008.Počet stran 79, Počet stran příloh 29. Diplomová práce. Vedoucí práce byl Ing. Radek Zezula, Ph. D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma VST Plug-IN pro vodoznační audio signálů jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce 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 ....

podpis autora

Tímto bych chtěl poděkovat vedoucímu své diplomové práce panu Ing. Radku Zezulovi, Ph.D za jeho odborné vedení a vstřícnost při konzultacích. Taktéž bych mu chtěl velmi poděkovat za poskytnutí technických a literárních prostředků, které mi velmi pomohly při tvorbě této práce.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Technologie VST byla vytvořena firmou Steinberg Media Technologies, která je známá jak ve světě normálním tak ve světě hudebníků díky svým audio aplikacím a zpracování zvuku. VST Plug-in lze hostovat mnoha hostitelskými audio aplikacemi využívajících ASIO ovladače. VST plug-in samotný je zásuvný modul pro tyto aplikace, které jej využívají při DSP(Digitálním zpracování signálů) jako jednotku syntetizující zvuk či s jinou funkcí. Může obsahovat i grafické uživatelské rozhraní pro snadnější přehled a změnu parametrů.

VST plug-in lze realizovat pomocí mnoha programovacích jazyků a vývojových prostředí, které budou jmenovány dále. Plug-in popsaný v této práci byl vytvořen pomocí volně dostupného programu Microsoft Visual C++ 2005 Express Edition. Vlastní řešení plug-inu obsahuje základní rozhraní VST SDK, vlastní soubory s řešením plug-in modulu a GUI editoru. Jednotlivé prvky jsou svázány tak, aby byla zajištěna jejich funkčnost a stabilita.

Vytvořený plug-in lze pak vložit do hostitelských audio aplikací podporujících ASIO ovladače a je dále využíván jako DSP jednotka pro analýzu (analysis) či sytnézu (synthesis). Pod analýzou je možno si představit např. FFT analýzu, kdy je zobrazováno frekvenční spektrum signálů, avšak signál není žádným způsobem změněn. Pod syntézou si je potom možno představit libovolný efekt, který zpracovává vstupní signál a přeposílá jej změněný na výstup. Pokud je k tomu uzpůsoben, může provádět libovolnou funkci jakož i vodoznačení audio signálu v reálném čase ve spolupráci s ASIO ovladači.

ASIO ovladače byly vyvinuty pro minimalizaci latence signálů, které probíhají signálovou cestou počítače na výstup zvukové karty. Pokud je k dispozici odpovídající hardware je možné snížit latenci v rozmezí 0 – 10 ms. Jako zástupce zvukových karet podporujících ASIO ovladače lze jmenovat M-Audio, E-MU či Behringer. Mezi hostitelské audio aplikace podporující ASIO ovladače lze jmenovat například Sound Forge, Live Lite, či Mini Host.

Druhá část práce se potom zabývá vodoznačením audio signálů v reálném čase jako možnost ochrany vlastnických práv audio obsahu. V této práci bude popsána metoda vodoznačení v časové oblasti tzv. „Echo Data Hiding“. Detekce vodoznaku je pak prováděna pomocí autokorelace v kepstrální oblasti.

1 Potřebné součásti

Pro vytvoření VST plug-in modulu je zapotřebí mít:

- VST SDK (Software Developer Kit) tj. sada vývojových nástrojů, která obsahuje kód potřebný k vývoji plug-in modulů. Více v kap. 1.1.
- Mít k dispozici hostitelskou aplikaci (např. aplikace MiniHost) schopnou načíst VST plug-in. Názvy vhodných aplikací jsou uvedeny v kapitole 1.2.
- Kompilátor Vývojového prostředí, který je schopen pracovat s daným OS pro kompilaci zdrojových souborů. Seznam kompilátorů je uveden v kap. 1.4.

1.1 VST SDK

VST SDK je sada vývojových nástrojů vyvinutá pro vývojáře a uživatele VST plug-in modulů pro usnadnění jejich práce.

Dostupné verze VST SDK:

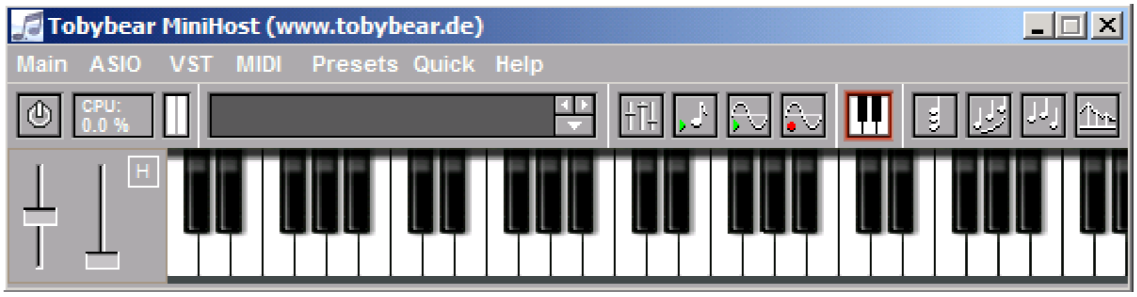
- VST 2.3 – Starší verze.
- VST 2.4 – Současná verze. Disponuje rozšířením nových funkcí o základní funkce.
- VST 3.0 – Připravovaná verze.

Tato sada nástrojů byla od počátku distribuována jako Open Source ve formě zdrojových kódů a lze je získat např. z webových stránek www.steinberg.de.

1.2 Hostitelská aplikace

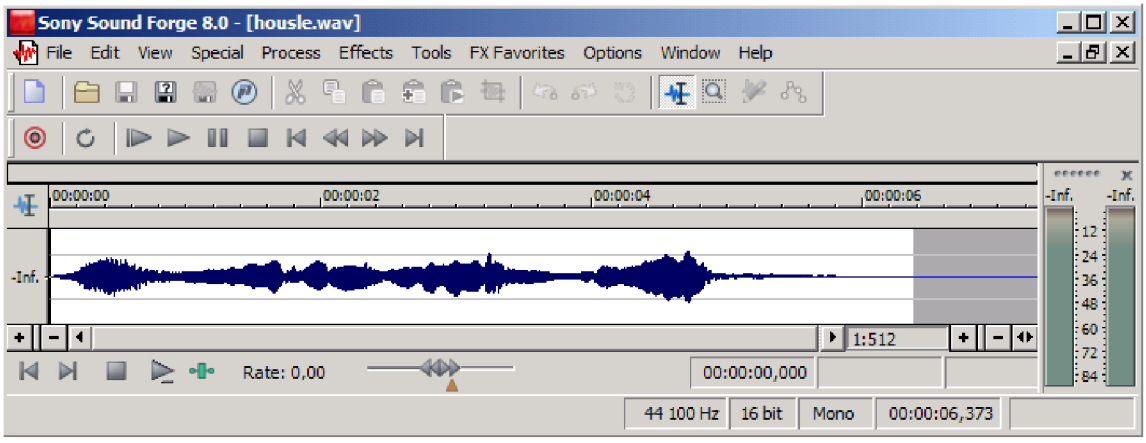
Jedná se o aplikace podporující VST technologii a většinou i ASIO ovladače. Zda tyto technologie podporují lze většinou zjistit ve specifikaci dané aplikace, jako příklad lze uvést:

MiniHost – Jednoduchá Open Source aplikace, možnost načíst VST plug-in, přehrávání wav, MIDI, podpora ASIO.



Obr. 1.1: Aplikace MiniHost pro testování VST plug-in modulů.

Sound Forge – Licencovaný audio editor, mnoho funkcí, mnoho formátů souborů, podpora VST, ASIO, MME.



Obr. 1.2: Aplikace Sound Forge pro testování VST plug-in modulů.

Všechny uvedené aplikace lze využít pro testování VST plug-in modulů. Je doporučeno nejdříve otestovat plug-in v co nejvíce aplikacích podporujících VST technologii. Pokud plug-in pracuje bez problémů ve všech aplikacích, potom je s největší pravděpodobností navržen správně a je možno jej zaregistrovat jako platný VST plug-in u společnosti Steinberg.

Pozn.: Některé hostitelské aplikace vyžadují přítomnost ASIO ovladačů v systému, z toho důvodu je nutno nainstalovat ASIO ovladače, viz. následující kapitola (1.3).

1.3 ASIO ovladače

Jejich použití má v audio aplikacích zásadní vliv na latenci signálu. Latence je zpoždění mezi vstupem a výstupem audio a MIDI signálu. Lze ji zaznamenat například při použití virtuálních MIDI nástrojů, kdy při stisku klávesy se očekávaný tón přehraje s velkým zpožděním. Tento problém může být zapříčiněn příliš dlouhými datovými či MIDI kabely, nevhodným hardwarem nebo právě špatnými ovladači. Z toho důvodu vyvinula firma Steinberg Media Technologies technologii ASIO (Audio Stream Input Output) ve formě ovladačů, které umožňují přímou komunikaci s audio rozhraním a minimalizují latenci na maximální možné minimum, které je dáno především výkonem počítače. Latence signálu je uváděna v milisekundách. Lidské ucho je schopno rozeznat zpoždění signálu větší než 10 – 15 ms, proto je nutno snížit toto nepříjemné zpoždění pod tuto hranici a poté lze subjektivně říci, že je audio signál přehráván v reálném čase s minimální latencí.

K profesionálním audio rozhraním značek M-AUDIO, EDIROL, E-MU, BEHRINGER a dalších jsou ASIO ovladače standardně k dispozici. ASIO ovladače lze provozovat i na zvukových kartách CREATIVE. Slouží k tomu speciální ovladače kxASIO vyvinutá skupinou programátorů nezávisle vyvíjející ASIO ovladače pro zvukové karty na bázi EMU10K1 a 10K2. Tyto ovladače se vyvíjejí jako Open Source projekt a lze je získat zdarma na oficiálním webu projektu http://kxproject.lugosoft.com.

Latence těchto ovladačů se pohybuje okolo 3ms. Pro integrované karty obsahující AC'97 kodek lze využít ovladače ASIO4ALL. Získat je lze také zdarma na domovské stránce projektu http://www.asio4all.com.

Tento software by měl být schopen emulovat ASIO ovladače pro jakoukoliv integrovanou zvukovou kartu, avšak výsledky nejsou až tak uspokojivé jako v případě kxASIO. Tyto ovladače mohou dosáhnout uspokojivých výsledků nebo také nemusí fungovat vůbec. To vše záleží na hardwaru počítače.

Nastavení ASIO ovladačů je prováděno přímo v prostředí audio aplikace, která je provozována (např. aplikace Sound Forge 8.0). V nastavení je nutno zvolit příslušný ovladač typu ASIO (může se jich tam vyskytovat více) a poté případně nastavit latenci signálu. Tato hodnota se nastavuje na nejnižší hodnotu, při které je počítač ještě schopen provozovat audio aplikaci bez "Drop-Out" efektu. Jedná se o nepříjemné praskání a zasekávání zvuku. Při velmi nízké hodnotě latence může aplikace do jejího restartu přestat reagovat na přehrávání zvuku vůbec. Pokud se nelze při přehrávání dostat na nějakou rozumnou hodnotu latence, může to být zapříčiněno nedostatečným výkonem počítače, nebo špatným nastavením OS.

Buffer[sample]	**64**	**128**	**256**	**384**	**512**	**768**	**1024**	**2048**
Input Latency[ms]	**2.54**	3.99	6.89	9.80	12.7	18.5	24.30	47.50
Output Latency[ms]	**4.35**	5.80	8.71	11.60	14.5	20.3	26.10	49.30
Overall Latency [ms]	**6.89**	9.80	15.60	21.40	27.2	38.8	50.40	96.90

Tab. 1.1: Latence signálu rozhraní M-Audio FireWire 410 pro ASIO ovladače.

Buffer [sample]	256	512	1024	2048
Input Latency[ms]	5.80	**11.60**	23.20	46.40
Output Latency [ms]	5.80	**11.60**	23.20	46.40
Overall Latency [ms]	11.60	**23.20**	46.40	92.90

Tab. 1.2: Latence signálu rozhraní M-Audio FireWire 410 pro WDM/MME ovladače.

Tabulka 1.1 znázorňuje velikost latence signálu v závislosti na nastavení velikosti bufferu audio rozhraní. Pro testování bylo použito audio rozhraní M-Audio Firewire 410 s ASIO 2.0 ovladači. Tabulka 1.2 pak znázorňuje latenci signálu pro klasické WDM/MME ovladače využívané OS Windows při využití stejného audio rozhraní. Nejlepší možný výsledek, kterého bylo možno dosáhnout je v tabulkách výše vyznačen tučně. Při těchto hodnotách bylo ještě možno provozovat M-Audio Firewire 410 bez „Drop-Out“ efektů. Testování proběhlo na PC Intel Pentium III, 512MB RAM s využitím software Sound Forge 8.0 při čistém přehrávání bez využití VST plug-inu. Z hodnot je zřejmé, že latence při použití ASIO ovladačů dosahuje výrazně lepších hodnot.

Při testování na výrazně složitějším softwaru Live Lite 6.0 se projevila závislost latence signálu na výkonu PC a bylo nutno zvýšit ASIO buffer na 256 vzorků a WDM/MME buffer na 1024 vzorků. Pro možnost vodoznačení signálů v reálném čase je tedy nutno vlastnit PC s výkonem odpovídajícím jeho zatížení. Při využití VST plug-inu vzroste zatížení CPU ještě vice v závislosti na složitosti algoritmu využitého v samotném plug-inu.

1.4 Kompilátor

VST plug-in lze vytvořit v poměrně mnoha programovacích jazycích a různých operačních systémech. Na platformě Windows lze využít vývojová prostředí jako Microsoft Visual C++ (dále jen VC++), Borland C++ Builder, Delphi či CodeWarrior. Oficiální vývojové prostředí pro VST SDK 2.4 na platformě Windows je Microsoft Visual C++ a Apple XCode 2.2 na platformě MacOSX. Obě vývojová prostředí jsou zdarma ke stažení na internetových stránkách:

Windows: http://www.microsoft.com/express/download/

MacOSX: http://developer.apple.com/tools/xcode/

Pro vytvoření VST plug-inu uvedeném v této práci bylo využito volně dostupné vývojové prostředí VC++ 2005 s podporou .NET. Veškerá potřebná nastavení jsou popsána v následující kapitole.

1.4.1 Instalace VC++ 2005

Nejprve je nutné stáhnout a nainstalovat VC++ [1,4]. Při volbě součástí instalace je třeba zvolit „Graphical IDE“ a „Microsoft MSDN2005“ a pro samotnou instalaci mít dostatek místa na disku. Instalace vyžaduje minimálně 350 MB volného místa na disku C pro dočasné soubory avšak místo instalace je libovolné. Celková velikost instalace je okolo 1.5 GB. Po instalaci VC++ je třeba nainstalovat service pack 1, který lze najít na internetových stránkách Microsoft update http://www.update.microsoft.com. Po instalaci SP1 je doporučeno stáhnout aktualizace „.NET Framework 2.0“, které lze najít taktéž na stránkách Microsoft update. V současné době je již k dispozici aktualizace „.NET Framework 3.5“. Předchozími kroky bylo nainstalováno vývojové prostředí. Aby bylo možno vytvářet aplikace pro OS Windows, je třeba nainstalovat „Platform SDK for Microsoft Visual C++ 2005 express“ což je specifická sada knihoven pro Windows. Tuto instalaci lze opět najít na stránkách Microsoft update, je třeba zvolit druh instalace dle procesoru:

- PSDK-amd64.exe – pro 64 bitové procesory AMD
- PSDK-ia64.exe – procesory Intel s architekturou IA-64
- PSDK-x86.exe – univerzální pro procesory s architekturou Intel

Pro instalaci je nutné mít na disku alespoň 950 MB volného prostoru. Po instalaci Platform SDK je nutno spustit Visual C++ a v menu Tools -> Options -> Project and Solutions -> VC++ Directories doplnit následující adresáře dle jejich typu:

Executable Files

C:\Program Files\Microsoft Platform SDK\Bin

Include Files

C:\Program Files\Microsoft Platform SDK\Include

Library Files

C:\Program Files\Microsoft Platform SDK\Lib

Dále je nutno v souboru “corewin_express.vsprops”, který je standardně uložen v adresáři

"*C:\Program Files\Microsoft Visual Studio 8\ VC\ VCProjectsDefaults*"

nalézt řádek s textem:

AditionalDependecies = "kernel32.lib"

a nahradit jej textem:

AdditionalDependencies="kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib".

V souboru *AppSettings.htm* v adresáři „C:\Program Files\ Microsoft Visual Studio 8\ VC \VCWizards\ AppWiz\ Generic\ Application\ html\ 1033\" zapoznámkovat pomocí „*//*" řádky 441 až 444 takto:

//WIN_APP.disabled = true; //WIN_APP_LABEL.disabled = true;
//DLL_APP.disabled = true; //DLL_APP_LABEL.disabled = true;

Tím je povoleno vytváření Win32 aplikací ve Visual C++ 2005. Nakonec je vhodné zaregistrovat Visual C++, neboť je časově omezené na 30 zkušebních dní. To lze provést z nabídky *Help -> Register Product*. Pro registraci produktu je nutno být zaregistrován na stránkách MSDN (Microsoft Developer Network) jako uživatel.

1.4.2 Vytvoření nového projektu

Po spuštění Visual C++ je třeba vytvořit projekt typu "Class Library", zadat název projektu např. *nazev_projektu*. Lze vymazat všechny soubory, které VC++ generuje automaticky kromě souboru *nazev_efektu*.rc a odstranit všechny odkazy na tyto soubory (jako např. "nazev_projektu.ico", který se nachází ve zdrojích). Dále je nutno přidat vlastní zdrojový kód a soubory VST SDK. Na obr. 1.4 je zobrazena ukázka souborů nutných pro správný běh plug-inu. Složky *vst2.x* a *vstgui* obsahují soubory z knihoven VSTSDK a VSTGUI.

Nyní je třeba přejít do nastavení projektu buď dvojitým kliknutím na název projektu *v Property Manageru* nebo z menu *Project -> Properties* a provést následující změny pro vytváření projektu [4]:

Na kartě *General* nastavit:

Character Set - **Not Set**

Common Language Runtime Support - **No Common Language Runtime Support**

Na kartě *C/C++* nastavit:

Additional Include Directories – **../vst2.x";../Interface;../vstgui;../source; ../editor** mělo by obsahovat cesty k potřebným souborům VSTSDK, pokud je použito tak i VSTGUI a všechny ostatní hlavičkové soubory použité v projektu.

Preprocessor Definitions - **WINDOWS;_WINDOWS;WIN32;_USRDLL ;_USE_MATH_DEFINES** by mělo být definováno jako minimum pro korektní funkci. Pokud je použita knihovna Libpng, je třeba přidat **USE_LIBPNG=1**.

Je také dobré definovat **_CRT_SECURE_NO_DEPRECATE** pro potlačení varovných hlášení kompileru.

V některých případech je nutno definovat **VST_FORCE_DEPRECATED=0**.

Runtime Library - **Multi-threaded**. Multi-threaded debug může být použito pro debug builds. Toto vytvoří ve VC++ common runtime knihovnu staticky do pluginu, a zvětší jeho velikost přibližně o 200Kb. Pokud je zvoleno CRL jako dynamická knihovna, je třeba distribuovat kopii CRL společně s aplikací, což komplikuje rozmístění a distribuci.

Create/Use Precompiled Header - **Not Using Precompiled Header** pokud je nastaveno takto nebude brán v úvahu předkompilovaný hlavičkový soubor, který při vytváření projektu smazán.

Compile As – **Default** další možnosti jsou C++ či C.

Runtime Library - Multi-threaded Debug DLL (/MDd)

Additional Dependencies - **zlib.lib libpng.lib** pokud je třeba. Některé knihovny nemusí být nutné v případě plug-inu bez GUI, avšak k žádným přídavným deklaracím nedojde, dokud nebudou v plug-inu linkované žádné symboly z těchto knihoven. Základní knihovny byly přidány již při instalaci VC++ (viz. kapitola 1.4.1).

Na kartě *Linker* nastavit:

Module Definition File – **nazev_projektu.def**, což je prostý textový soubor, který je nutno definovat v projektu. Soubor by měl obsahovat tento text:

„EXPORTS „VSTPluginMain“ „main=VSTPluginMain“.

Tento soubor odkazuje kompiler na hlavní metodu *main()* v napsaném kódu. Pokud je použito 2.4 SDK, pak se o to postará soubor *vstplugmain.cpp* automaticky. Pokud je použito 2.3 SDK, je třeba deklarovat hlavní metodu:

*AEffect *main_plugin (audioMasterCallback audioMaster);.*

2 VST Plug-IN

Pokud je vše nastaveno správně podle kapitoly 1.4 je možno začít psát vlastní kód pro VST plug-in. Každý VST plug-in může obsahovat jeden nebo více souborů s vlastním kódem a s jedním či více editorů GUI. Lze také kombinovat 2D editor se 3D editorem. VST plug-in je taktéž schopen přijímat příkazy z MIDI klávesnice (tzv. MIDI signály), pokud vlastní funkce pro jejich zpracování. Vlastní řešení (deklarace) je obsaženo v souborech:

nazev_efektu.cpp, nazev_editoru.cpp, nazev_efektu_main.cpp

, kde *nazev_efektu.cpp* je soubor obsahující samotný audio efekt a příslušné funkce a soubor *nazev_editoru.cpp* přiložený editor GUI, pokud je k dispozici. Soubor *nazev_efektu_main.cpp* je použit pro vytvoření nové instance efektu. Definice příslušné třídy efektu a editoru je pak obsaženo v hlavičkových souborech s ekvivalentním názvem a příponou „*h*“.

2.1 Vytvoření nové instance efektu

V souboru *nazev_efektu_main.cpp* je třeba vytvořit novou instanci třídy efektu pomocí funkce:

```
AudioEffect* createEffectInstance (audioMasterCallback audioMaster)
{
return new nazev_efektu (audioMaster);
};
```

Po zavolání této funkce je vytvořena hostující aplikací nová instance efektu. Pro definovanou funkci by měl být plug-in schopen využívat předefinovaných funkcí a sad dotazů, které musí být deklarovány ve třídě *nazev_efektu*, která se nachází v souboru *nazev_efektu.h* a definovány v souboru *nazev_efektu.cpp* a popsané v následující kapitole.

2.2 Základní funkce VST

VST SDK obsahuje velké množství užitečných funkcí [3]. V následující kapitole budou popsány potřebné a užitečné funkce nutné ke správnému chodu plug-in modulu. V souboru *nazev_efektu.h* se nachází třída, ve které je nutno tyto funkce definovat. V tab. 2.1 jsou shrnuty základní funkce potřebné pro chod plug-in modulu.

Název funkce	Definice funkce
processReplacing	virtual void processReplacing (float** inputs, float** outputs, VstInt32 sampleFrames);
subProcessReplacing	virtual void subProcessReplacing (float *in, float *out1, float *out2, float *dest, float *del, long sampleframes);
setProgram	virtual void setProgram (VstInt32 program);
setProgramName	virtual void setProgramName (char* name);
getProgramName	virtual void getProgramName (char* name);
getProgramNameIndexed	virtual bool getProgramNameIndexed (VstInt32 category, VstInt32 index, char* text);
setParameter	virtual void setParameter (VstInt32 index, float value);
getParameter	virtual float getParameter (VstInt32 index);
getParameterLabel	virtual void getParameterLabel (VstInt32 index, char* label);
getParameterDisplay	virtual void getParameterDisplay (VstInt32 index, char* text);
getParameterName	virtual void getParameterName (VstInt32 index, char* text);
resume	virtual void resume ();
suspend	virtual void suspend ();
getEffectName	virtual bool getEffectName (char* name);
getVendorString	virtual bool getVendorString (char* text);
getProductString	virtual bool getProductString (char* text);
getSampleRate	float getSampleRate (void);
getBlockSize	VstInt32 getBlockSize (void);

Tab. 2.1: Seznam základních funkcí VST.

processReplacing – Funkce je volána hostující aplikací. Tato funkce má za úkol převzít vstup o definované délce, upravit jej pomocí DSP a předat na výstup.

subProcessReplacing – Pomocná funkce, která je volána přímo plug-in modulem z předchozí funkce . Může být volána například pokud je třeba upravit signál specifickým způsobem.

setProgram – Využívá se k požadovanému nastavení jednotlivých proměnných. Parametr funkce určuje, který program se má načíst. Tuto funkci opět využívá hostující aplikace.

setProgramName – Funkce nastaví jméno aktuálního programu např. “Flanger”.

getProgramName - Funkce vrací jméno aktuálního programu např. “Chorus”.

getProgramNameIndexed – funkce vrací true nebo false dle toho, jestli do pole znaků bylo vloženo jméno programu či nikoliv.

setParametr – Důležitá funkce. Nastavuje veškeré parametry plug-inu.

getParametr – Vrací hodnotu parametru.

getParametrLabel – Vrací v proměnné *Label* název jednotky, ve kterých je daný parametr uváděn.

getParametrDisplay – Vrací v proměnné *text* honotu parametru v textové podobě.

getParametrName - Funkce vrací v proměnné *text* název parametru.

resume – Volá se pro opětovný start procesu.

suspend – Volá se pro pozastavení chodu procesu.

getEffectName – Vrací název efektu.

getVendorString – Vrací nadpis „Steinberg Media Technologies“, který je zobrazen v hostující aplikaci.

getProductString – Vrací název produktu (většinou stejné jako název efektu).

getSampleRate – Vrací aktuální vzorkovací kmitočet.

getBlockSize – Vrací aktuální maximální velikost bloku.

Pokud je třeba lze využívat mnoha dalších funkcí definovaných v prostředí VST SDK.

3 GUI (Graphic User Interface)

Při tvorbě VST plug-in modulu je možno využít knihovny pro tvorbu grafického uživatelského rozhraní, která je součástí VST SDK (Software Developer Kit). Zde je možno realizovat základní ovládací prvky potřebné pro ovládání parametrů jako třeba různé druhy tlačítek, přepínačů, jezdců, číselných hodnot či textu [3]. Pokud není GUI definováno, vhodná hostitelská aplikace vytvoří základní rozhraní pro ovládání parametrů automaticky tak, že každému parametru je přiřazen ovládací prvek.

3.1 Přidání podpory VSTGUI

Přidání VSTGUI podpory se provede jednoduše přidáním VSTGUI souborů do projektu. Pro správnou funkci by měl editor obsahovat minimálně tyto soubory:

aeffguieditor.cpp, vstcontrols.cpp, a vstgui.cpp

a dále přidáním odkazu do hlavičkového souboru editoru *editor.h* pomocí direktivy:

#include "vstgui.h".

Editor GUI je dědičnou třídou třídy *AEffGUIEditor* a *CControlListener*, kterou je nutno definovat v souboru „*editor.h*“. Třída by měla obsahovat alespoň definice základních funkcí, ovládacích prvků a bitmap. Výsledná třída může vypadat takto:

```
class editor : public AEffGUIEditor, public CControlListener
{
public:
      Editor (AudioEffect* effect);
      virtual ~Editor ();
public:
      virtual bool open (void* ptr);
      virtual void close ();
      virtual void setParameter (VstInt32 index, float value);
      virtual void valueChanged (CDrawContext* context, CControl*
control);
private:
      // Controls
      CVerticalSwitch* powerSwitch;
      COnOffButton*      button1;
      CParamDisplay* powerDisplay;
      //Text
      CTextEdit* butt1text;
      const char* text1;

      // Bitmap
      CBitmap* hBackground;
  };
```

Ve zdrojovém souboru editoru *editor.cpp* je potom zajištěna definice funkcí a funkce tlačítek.

3.2 Bitmapy

Bitmapy jsou ve Visual C++ spravovány jako tzv. „*Resources*", které jsou uloženy v souboru „*resources.rc*". Každá bitmapa je reprezentována jedním řádkem v tomto souboru a odkazuje se na ně např. pomocí výčtového typu *enum*. Příklad uložení zdrojů je zobrazen v tabulce 3.1.

ID	Typ	Použití	Název souboru
128	BITMAP	DISCARDABLE	"bmp01.bmp"
129	BITMAP	DISCARDABLE	"bmp02.bmp"

Tab. 3.1: Definice zdrojů v „Resources.rc".

Pomocí výčtového typu definovaného v editoru se lze odkazovat na jednotlivé zdroje například takto:

enum { kBackgroundId = 128, kPowerBodyId};

Pro vytvoření nové bitmapy v editoru stačí v konstruktoru třídy zadefinovat:

hBackground = new CBitmap (kBackgroundId);

V proměnné „*hBackground*", která je typu „*CBitmap**" se nachází zvolená bitmapa, kterou je možno používat i opakovaně pro různé druhy tlačítek.

Pozn.: Podle tabulky 3.1 by měly být názvy souborů uloženy ve stejném adresáři jako soubor resources.rc, v opačném případě musí být v názvu souboru uvedena také cesta k souboru.

3.3 Vytváření grafických prvků

Jednotlivé ovládací prvky, které lze využít v editoru GUI lze shrnout do následující tabulky:

Název prvku	Definice v prostředí VST SDK
CControl	CControl (CRect &size, CControlListener *listener, int tag, CBitmap *pBackground = 0);
CControlListener	virtual void valueChanged (CDrawContext *context,CControl *control);
COnOffButton	COnOffButton (CRect &size, CControlListener *listener, int tag, CBitmap *bitmap);
CParamDisplay	CParamDisplay (CRect &size, CBitmap*background = 0, int style = 0);
CTextEdit	CTextEdit (CRect &size, CControlListener *listener, int tag, const char *txt = 0, CBitmap *background = 0, int style = 0);
COptionMenu	COptionMenu (CRect &size, CControlListener *listener, int tag, CBitmap *background = 0, CBitmap *bgWhenClick = 0, int style = 0);
COptionMenuScheme	COptionMenuScheme ();
CKnob	CKnob (CRect &size, CControlListener *listener, int tag, CBitmap *background, CBitmap *handle, CPoint &offset);
CAnimKnob	CAnimKnob (CRect &size, CControlListener *listener, long tag, long subpixmaps, long heightOfOneImage, CBitmap *handle, CPoint &offset);
CVerticalSwitch	CVerticalSwitch (CRect &size, CControlListener *listener, long tag, long subpixmaps, long heightOfOneImage, long iMaxPositions, CBitmap *handle, CPoint &offset);
CHorizontalSwitch	CHorizontalSwitch (CRect &size, CControlListener *listener, long tag, long subpixmaps, long heightOfOneImage, long iMaxPositions, CBitmap *handle, CPoint &offset);
CRockerSwitch	CRockerSwitch (CRect &size, CControlListener *listener, long tag, long heightOfOneImage, CBitmap *handle, CPoint &offset, long style = kHorizontal);
CMovieBitmap	CMovieBitmap (CRect &size, CControlListener *listener, long tag, long subpixmaps, long heightOfOneImage, CBitmap *handle, CPoint &offset);
CMovieButton	CMovieButton (CRect &size, CControlListener *listener, long tag, long heightOfOneImage, CBitmap *handle, CPoint &offset);
CAutoAnimation	CAutoAnimation (CRect &size, CControlListener *listener, long tag, long subPixmaps, long heightOfOneImage, CBitmap *handle, CPoint &offset);
CSlider	CSlider (CRect &size, CControlListener *listener, long tag, int iMinPos, int iMaxPos, CBitmap *handle, CBitmap *bk, CPoint &offset, long style);
CVerticalSlider	CVerticalSlider (CRect &size, CControlListener *listener, long tag, long iMinPos, long iMaxPos, CBitmap *handle, CBitmap *bk, CPoint &offset, int style);
CHorizontalSlider	CHorizontalSlider (CRect &size, CControlListener *listener, long tag, long iMinPos, long iMaxPos, CBitmap *handle, CBitmap *bk, CPoint &offset, int style);
CSpecialDigit	CSpecialDigit (CRect size, CControlListener *listener, long tag, long dwPos, long iNumbers, long *xpos, long *ypos, int width, int height, CBitmap *background);
CKickButton	CKickButton (CRect &size, CControlListener *listener, long tag, long heightOfOneImage, CBitmap *handle, CPoint &offset);
CSplashScreen	CSplashScreen (CRect &size, CControlListener *listener, long tag, CBitmap*handle,CRect &toDisplay, CPoint& offset);
CVuMeter	CVuMeter (CRect &size,CBitmap*onBitmap, CBitmap *offBitmap, long nbLed, long style);
CFileSelector	CFileSelector (AudioEffectX *effect);

Tab. 3.2: Seznam ovládacích prvků knihovny VSTGUI.

4 Vytváření efektů

Efekty jako flanger,chorus, phaser a echo lze popsat podle blokového schématu uvedeného na obr. 1. Jednotlivé efekty se od sebe většinou liší pouze odlišnými parametry, které je možno v prostředí VST SDK snadno přednastavit. Základem je přímá a efektová cesta. Efektovou cestu tvoří zásobník s proměnnou délkou obsahující ZV.

4.1 Nastavení parametrů

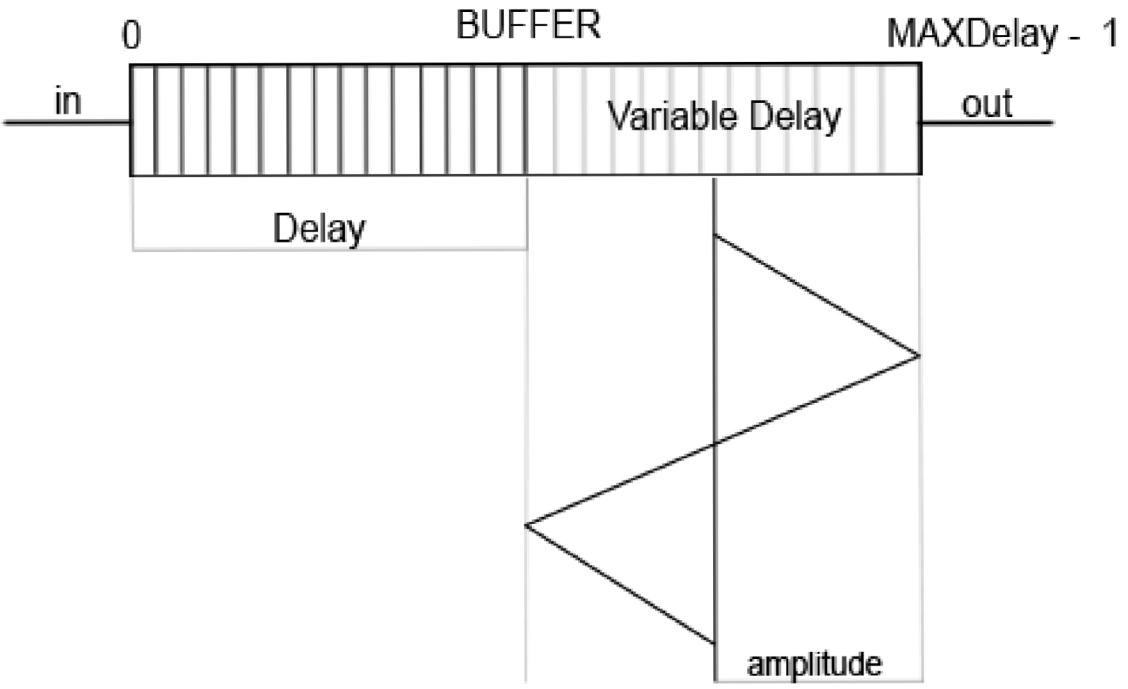
V následující tabulce lze pro přehled vyčíst základní parametry efektů, které lze možno aplikovat v tomto projektu pomocí zpožďovacího bufferu [6].

Efekt	Dry	Wet	Feedback	Zpoždění [ms]	Změna [Hz]	Hloubka [ms]
Chorus	0.7	1	0	1-30	0.1-30	1-30
Flanger	0.7	0.7	0.7	0-15	0.1-1	0-2
Vibrato	0	1	0	0-15	0.1-5	0-3
Phaser	0.7	1	0	1-10 vzorků	0.1-5	1-10 vzorků

Tab.4.1: Parametry efektů.

4.2 Princip zásobníku s proměnnou délkou

Metody zpracování zvuku založené na proměnné délce zpožďovacího zásobníku lze využít u mnoha efektů. Jedná se o buffer s pevnou a proměnnou částí. Pevnou část lze měnit pomocí parametru Delay a proměnnou část pomocí parametru VariableDelay. VariableDelay je obvykle řízena pomocí periodického signálu jako je např. sinus, triangle, square či sawtooht. Na obrázku 4.1 je uveden příklad zásobníku řízeného trojúhelníkovou periodickou funkcí.



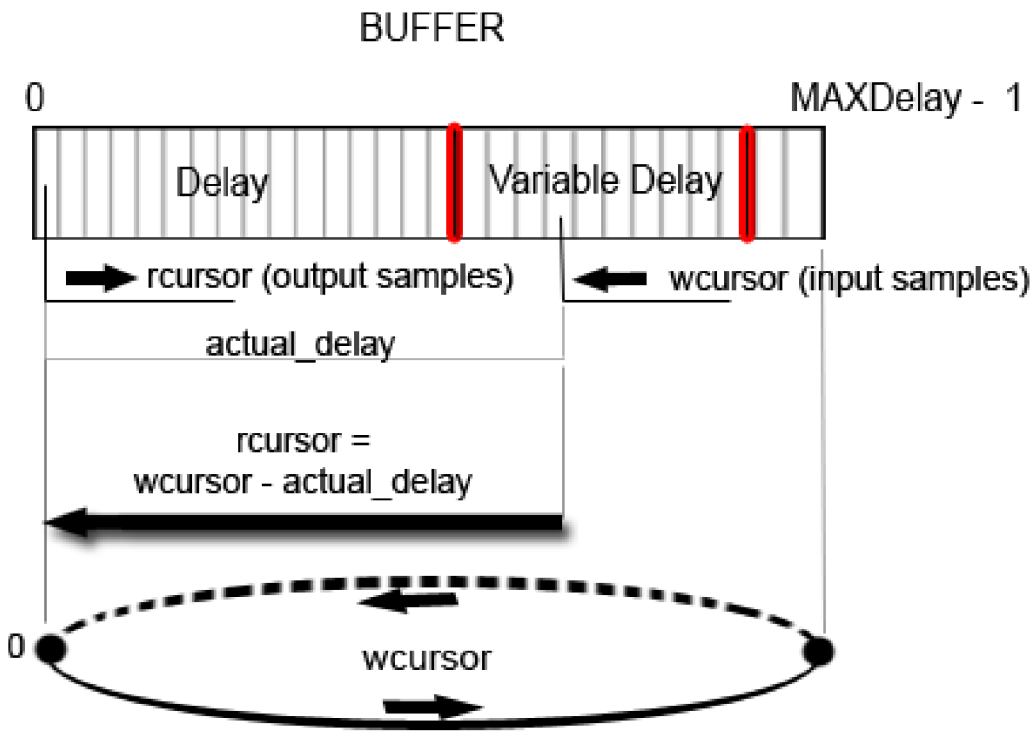
Obr. 4.1: Zásobník s proměnnou délkou.

Celkové zpoždění je rovno $\mathrm{Z}^{-\mathrm{M}} = \mathrm{Z}^{-\,\mathrm{MAXDelay}\,-1}$. Změna zpoždění je dána funkcí $\mathrm{M(n)}$,což je zadaná periodická funkce.

Druhy periodických funkcí :

- Harmonický signál (sin)
- Trojúhelník
- Obdelník
- Pila
- Exp.
- Log.

Pro implementaci tohoto bufferu do prostředí VSTSDK lze postupovat dle obrázku 4.2. Nejprve je nutno definovat buffer o požadované velikosti MAXDelay a kurzory pro čtení (rcursor) a zápis (wcursor) dat.

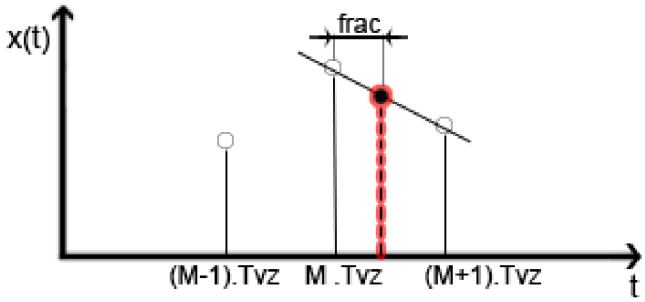


Obr. 4.2: Zásobník s proměnnou délkou – implementace.

Zápis dat provádí *wcursor* a to v rozmezí 0 ÷ MAXDelay–1. Jakmile dorazí *wcursor* na konec bufferu skočí opět na jeho počátek. Čtení dat provádí *rcursor* tak, že se odečte aktuální hodnota zpoždění od *wcursor*u. Výsledkem je pozice kurzoru na aktuální hodnotě zpoždění. Tato hodnota je pak přivedena na výstup bufferu. Po sečtení přímé a efektované cesty je docíleno vložení zpožděného prvku do aktuálně přehrávaného prvku.

4.3 Interpolace vzorků

Jelikož je zpoždění bufferu proměnné podle periodické funkce, hodnoty zpoždění nejsou celá čísla ale reálná. Proto je nutno dopočítat hodnotu prvku pro dané zpoždění pomocí interpolace. Pokud není prováděn výpočet interpolace lze očekávat výskyt různých druhů praskání v signálu známé jako „Zipper noise“.



Obr. 4.3: Výpočet hodnoty zpožděného vzorku.

Interpolace	Výpočet
lineární	$y[n] = x[n - M + 1].frac + x[n - M].(1 - frac)$
all-pass	$y[n] = x[n - M + 1].frac + x[n - M].(1 - frac) - \mathrm{y[n-1].(1-frac)}$
spline	$y[n] = x[n - M + 1].\frac{frac^3}{6} + x[n - M].\frac{-4.frac^3}{6}$ $+ \ x[n - M - 1].\frac{(1 - frac)^3 - 2.frac^3}{6}$

Tab. 4.2: Výpočet interpolace.

Tabulka 4.2 dle [7]. Po výpočtu hodnoty zpožděného prvku tento přičte s přímou cestou a výsledek je přiveden na výstup zvukové karty. Zvuk, který je pak slyšet na výstupu zvukové karty je dán poměrem Dry/Wet přímé a zpožděné cesty.

5 Vodoznačení audio signálů

Při vodoznačení audio signálu [8] je do užitečného (slyšitelného) signálu vkládána tajná informace (neslyšitelný, informační signál), která může později sloužit k jeho identifikaci například z důvodu určení jeho původu či vlastníka. Tato tajná informace vkládaná ve formě vodoznaku. Tvar a povaha vodoznaku závisí na typu použité metody vodoznačení a u některých metod i na tvaru originální audio nahrávky, do které je vodoznak vkládán. Při návrhu systému vodoznačení jsou hlavními kritérii robustnost a neslyšitelnost vodoznaku.

Robustnost vodoznaku popisuje jeho schopnost odolávat okolním rušivým vlivům, které vložený vodoznak modifikují a tím ztěžují jeho obnovení při detekci, zatímco neslyšitelnost vyjadřuje výkonovou úroveň signálu, při které není vložený vodoznak možno zaznamenat lidským uchem. Proto je nutno zvolit kompromis mezi výkonovou úrovní vkládaného vodoznaku a jeho robustností.

Pro vodoznačení audio signálů se využívá mnoho technik. Jako základní rozdělení lze označit metody v časové a frekvenční oblasti. Každá z metod má své výhody a nevýhody a pro seznámení lze jmenovat několik metod vodoznačení audio signalů.

- LSB (added noise can be heard)

Metoda LSB [8] využívá nejméně významného bitu každého vzorku signálu pro vložení vodoznaku. Výhoda je vysoká bitová rychlost kodéru kde při $Fvz = 44100\,Hz$ lze vložit $44{,}1\,kb/s$. Nevýhoda je, že přidaný šum lze při přehrávání zaznamenat a tím se snižuje zvuková kvalita nahrávky.

- Phase Coding

Metoda fázového kódování [8] využívá změny fáze signálu ve frekvenční oblasti. Využívá toho, že lidské ucho je necitlivé k fázovým změnám audio signálu vyjma například přechodných dějů v signálu, kde lze změnu fáze zaznamenat. Pro vodoznačení touto metodou je vodoznak vložen pouze do prvního bloku signálu. Proto je tato metoda velmi náchylná na změnu délky signálu zejména na oříznutí počátku signálu, kde je vodoznak vložen.

- Spread Spectrum

Spread Spectrum je metoda frekvenčního vodoznačení, kde je vodoznak vkládán jako pseudonáhodný šum ve frekvenční oblasti. Pro zvýšení

robustnosti této metody se využívají psychoakustické modely, které přizpůsobují vodoznak tak, aby nebyl slyšitelný lidským uchem a zároveň nebyl ovlivněn kompresí do jiných datových formátů jako např. mp3. Tento formát využívá psychoakustického modelu k odstranění frekvenčních složek , které jsou lidským uchem nezaznamenatelné a tím snižuje objem kódovaných dat.

- Echo Data Hiding

Metoda echo hiding [7,8,9] patří do kategorie tzv. samoznačících (samoreplikujících) metod. Samoznačící metody vkládají vodoznak zanecháním zcela evidentního znaku do signálu. Tyto metody vkládájí speciální signál do audio obsahu nebo mění signální křivky v časové či frekvenční oblasti. Modifikace časové osy a spousta jiných schémat založených na těchto charakteristických rysech patří do této kategorie. Clumsyho samo-značící metoda například vkládá špičky ve frekvenční oblasti. Tyto špičky jsou lehce postřehnutelné, a proto je tato metoda náchylná k útokům. U metody echo hiding je vodoznak vkládán jako ozvěna (echo) signálu s požadovaným zpožděním. Pro neslyšitelnost vodoznaku se využívá faktu, že lidské ucho není schopno zachytit krátkodobé signály kratší jak 10-20 ms. Toto opět souvisí s psychoakustickým modelem lidského slyšení, kdy lidské ucho potřebuje určitý tón či zvuk slyšet po určitou dobu, aby si uvědomil jeho tón. Pokud je tento tón kratší než určitá hranice, lidské ucho jej nezaznamená. Výhoda této metody je její robustnost a neslyšitelnost. Nevýhodou je snadná detekce vodoznaku druhou osobou.

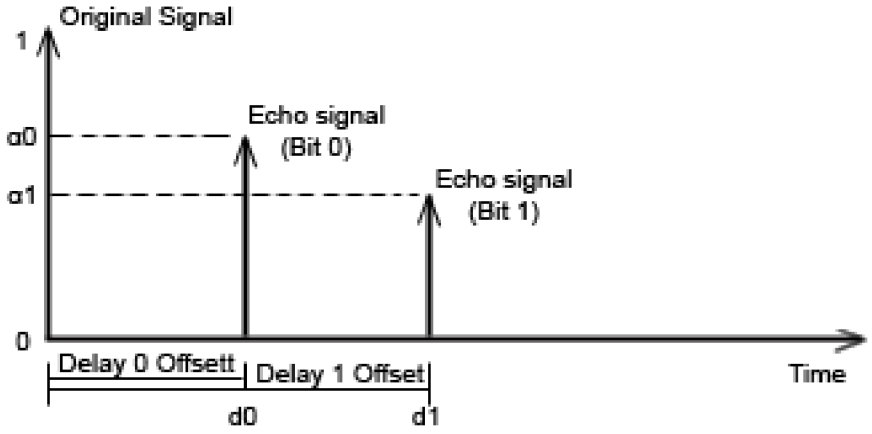
5.1 Metoda Echo Hiding

Echo hiding [7,8,9] využívá postmasking efektu z důvodu ovládání neslyšitelnosti vloženého vodoznaku. Časy zpoždění Δt_k a útlumový koeficient $\propto_k$, $k = 0,1$ musí být přizpůsobeny při procesu vkládání vodoznaku podle prahu vnímání lidského sluchového systému k zajištění neslyšitelnosti ozvěn. Jedná se o tzv. slepé vodoznačení, které moduluje vkládané bity jako ozvěnu(echo) signálu a tyto bity jsou pak vkládané do individuálních bloků audio signálu. Vkládání a detekce jsou prováděny ve dvou různých oblastech (doménách), a to v časové a kepstrální oblasti.

Tato metoda vkládá data do originálního audio signálu jako ozvěnu(echo) v časové oblasti pomocí vzorce:

$$x(n) = s(n) + \alpha s(n - d) \quad (1)$$

Jednotlivá echa jsou přidány do originálního signálu dle obrázku níže.



Obr. 5.1: Vložení echo signálů do originálního signálu.

Binární zpráva je zakódována do originálního signálu pomocí dvou echo signálů s různým zpožděním dle obrázku 5.1. Například pro bit 0 je zvoleno zpoždění signálu(Delay Offset) 1ms a pro bit 1 2ms. V diskrétní oblasti lze pak zapsat tyto zpoždění jako $d0$ a $d1$ pro bity 0 a 1 vyjadřující počet vzorků signálu. Extrakce vloženého vodoznaku pak spočívá v detekci zpoždění d. Toto lze realizovat pomocí autokorelace signálu v kepstrální oblasti, kdy je cílem nalézt v signálu špičku detekující velikost zpoždění a tím odhalit vložený bit.

Dvojité echo může snížit percentuální deformace signálu a zvýšit robustnost např. dle následujícího vzorce:

$$x(n) = s(n) + \alpha s(n - d) - \alpha s(n - d - \Delta) \quad (2)$$

,kde $x(n)$ je vodoznačný signál, $s(n)$ originální signál a d velikost zpoždění. Typická hodnota Δ se pohybuje v rozmezí menším jak 3 až 4 vzorky. Echo hiding je obvykle nepostřehnutelné a občas vytváří hlasitý zvuk. Synchronizační metody často adaptují tuto metodu jako hrubou synchronizaci. Výhoda této metody je jeho robustnost a neslyšitelnost.

Nevýhoda metody echo hiding je jeho složitost díky složitému výpočtu kepstrálních koeficientů během detekce. Další nevýhoda detekce vodoznaku druhou osobou. Kdokoliv může detekovat echo bez předchozí znalosti. Jinými slovy poskytuje vodítko k úmyslnému útoku či detekci vodoznaku. Další velká nevýhoda je zranitelnost algoritmu k záludným útokům, neboť vložená informace může být detekována kýmkoliv bez použití tajného klíče. Útočník

může objevit vložený vodoznak, pokud zná algoritmus, kterým byl vodoznak vložen. Možná náprava proti lehkému výpočtu časů zpoždění δt při detekci vodoznaku je rozprostření(spreading) ozvěny(echa) přes časovou osu. Toto je provedeno substitucí Diracova impulzu v odezvě PN sekvencí. Namísto výpočtu autokorelace v kepstrální doméně je provedeno zhuštění(despreading) ozvěny(echa) křížovou korelací kepstrálního signálu s PN sekvencí generovanou z tajného klíče. Vzorec pro výpočet dvojitého echa je uveden zde:

$$x(n) = s(n) + \alpha s(n-d) + \alpha s(n+d) \quad (3)$$

, kde vitruální echo s(n+d) porušuje kauzalitu. Mimo jiné je možné vložit virtuální echo zpožděním vkládacího procesu o d vzorků. Tato dvojice echa vytváří špičky a v kepstrální oblasti vyšší než obyčejné echo se stejnou úrovní α. Takto lze zvýšit pravděpodobnost detekce správného echa díky vyšším špičkám či snížit úroveň vodoznaku úměrným snížením parametru α.

6 Vložení vodoznaku

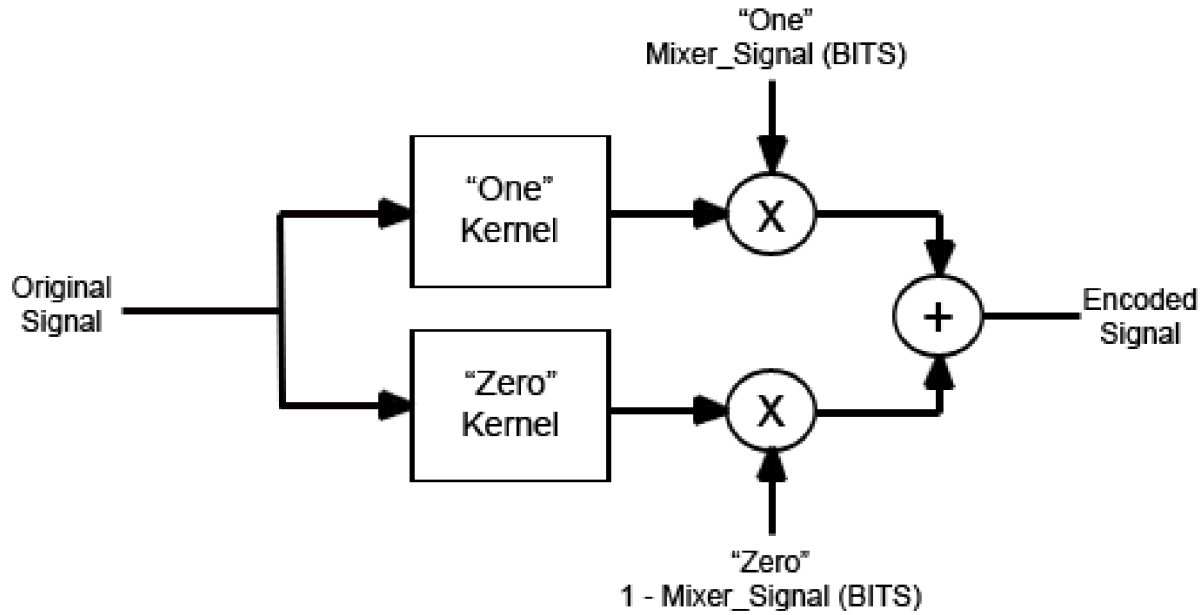
Tato kapitola popisuje techniku vkládání vodoznaků pomocí metody echo hiding. Vkládání vodoznaku je prováděno v časové oblasti vložením signálu s aktuálním zpožděním do právě zpracovávaného bloku. Zpoždění, které je právě vkládáno závisí na modulačním signálu, který je generován v závislosti na vkládaném vodoznaku. Pokud je třeba vložit vodoznak $W_c = "ahoj"$, pak je nutno tento vodoznak nejprve rozložit na bitovou posloupnost např. dle následujícího kódu.

```
int AEcho_Hiding_Wtm::Wtm2Binary (char* Wtm)
{
      int znaku=((int)strlen(Wtm));
      int bitu=Wtmlength*8;
      for(int i=0;i<Wtmlength;i++){
            int num = (int)Wtm[i-1];

            for(int j=0;j<8;j++){
                  if(num%2==0)
                  bits[j+i*8]='0';
                  else bits[j+i*8]='1';
                  num = num >> 1;
            }
      }
      bits[znaku*8]='\0';
      return bitu;
  }
```

Proměnná *bits* obsahující posloupnost bitů je pak využita při tvorbě modulačního signálu, který je popsán v následující kapitole.

6.1 Princip vložení vodoznaku

Při vkládání vodoznaku [7] do audio obsahu je možno použít blokového schématu uvedeném na obrázku níže.



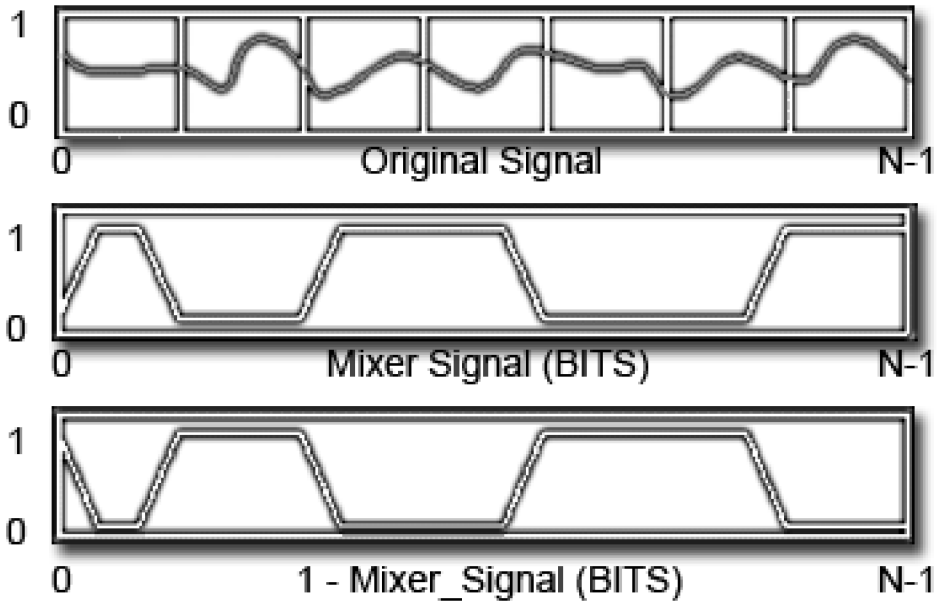
Obr. 6.1: Blokové schéma kodéru metody Echo Hiding.

Blok „One Kernel" a „Zero Kernel" mají za úkol zpozdit orignální signál o určitý čas(delay) a snížit jeho energii vhodnou konstantou k zajištění jeho neslyšitelnosti. Zpoždění je možno provézt využitím FIR filtru s přenosovou funkcí:

$$H(z) = 1 + g * z - d \tag{4}$$

,kde g je velikost amplitudy signálu a d velikost zpoždění. Každý blok zpožďuje signál s jinou hodnotou a na základě těchto hodnot lze při detekci zjistit, o který bit se jedná. Signály „One Mixer" a „Zero Mixer" zajišťují časové proložení originálního signálu dle bitů, které se za sebou vkládají v požadovaném sledu dle obrázku 6.2. Pokud je právě vkládaný bit 0 či 1, pak „One Mixer" bude mít hodnotu x a „Zero Mixer" hodnotu $1 - x$. Takto je zajištěno sečtení originálního signálu a pouze jednoho echa s určitým zpožděním reprezentujícím jeden bit 0 či 1. Každý blok pak nese 1 bit informace. Bitová rychlost kodéru záleží na velikosti bloku a vzorkovací frekvenci. Pro $Fvz = 44.1\ KHz$ a $N = 512$ bude bitová rychlost kodéru rovna:

$$Br = \frac{Fvz}{N} = \frac{44100}{512} \doteq 86\ b/s. \quad (5)$$



Obr. 6.2: Vkládání jednotlivých bitů pomocí modulačního signálu.

Výsledný signál z kodéru je pak přičten k originálnímu signálu a tak vznikne vodoznačný signál.

6.2 Realizace Algoritmu

V následující podkapitole bude popsána realizace algoritmu, která implementuje metodu echo hiding jako VST Plugin, který umožňuje vkládat vodoznak do audio signálu v reálném čase nebo jinými slovy blok po bloku. Část tohoto algoritmu je uvedena v následujícím kódu.

```
void AEcho_Hiding_Wtm::processReplacing (float** inputs, float**
outputs, VstInt32 sampleFrames)
{
//Deklarace proměnných
//Konec deklarace proměnných
while(--sampleFrames >= 0) {//0:1:512
bufferL[position] = *inL;
bufferL[position+N] = *inL;
if(blockCounter==0){//Inicializační(první) blok
  *outL++ = (*inL++);//Výpočet výstupního vzorku pro 1. Blok signálu
} else{
  causalityL = params[kAlfa]*bufferL[N-delay0-2+position]*(1.0f-
mixer_signal(a,b,c,d,e,f)) + params[kAlfa]*bufferL[N-delay1-
2+position]* mixer_signal(a,b,c,d,e,f); //vypocet zpožděného vzorku

*outL++ =(*inL++)+params[kAlfa]*bufferL[position+3*delay0]*(1.0f-
mixer_signal(a,b,c,d,e,f) + params[kAlfa]*bufferL[position+delay1]*
mixer_signal(a,b,c,d,e,f)+causalityL; //výpočet výstupního vzorku
```

```
}
position++; //pozice 0:blocksize
}//konec cyklu while
if(blockCounter>0) BitsCounter++;//Pocitadlo bitu
if(BitsCounter>NumberOfBits-1) BitsCounter=0;
position=0; //vždy pro každý blok opakující se index od 0 do
blocksize(N)
blockCounter++;
}//konec funkce ProcessReplacing
```

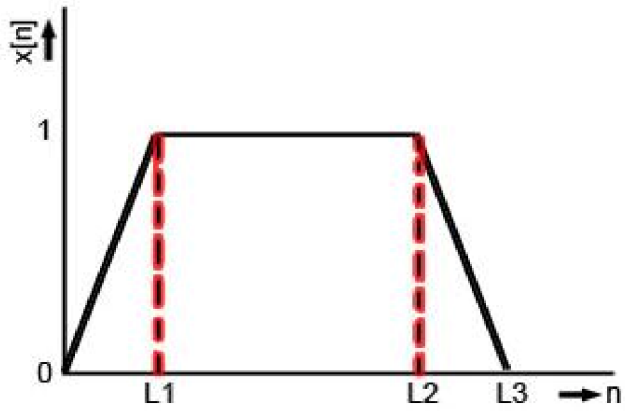
Samotná implementace kódu se týká funkce „*processReplacing*“, kde se provádí výpočet výstupních vzorků blok po bloku. Výpočet v kódu výše je pro názornost omezen pouze pro levý zvukový kanál. Funkce převezme 1. blok o *N* vzorků a tyto hodnoty uloží do bufferu. Při zpracování dalších bloků je již prováděno vodoznačení. Na výstup outL jsou postupně ukládány vypočtené vzorky s různým zpožděním. Toto zpoždění je určováno pomocí funkce *“mixer_signal(a,b,c,d,e,f)”*, která zajistí přičtení zpožděných vzorků k současným vzorkům na základě bitů, které se mají v danou chvíli vložit do signálu. Pro bit 0 platí zpoždění *delay0* a pro bit 1 zpoždění *delay1*. Proměnné *BitsCounter* a *blockCounter* značí pozici bitu a bloku. Proměnná *position* pak značí pozici v bloku 0,1,2,…,N-1.

```
float AEcho_Hiding_Wtm::mixer_signal(int position,char* bits,int
BitsCounter,int L1,int L2,int L3)
{
      float y;
      int counter1=counter-1;int counter2=counter;int
counter3=counter+1;

      if (counter == 0){
      counter1 = (int)strlen(bits)-1;
      counter3 = counter+1; }

      if (counter == ((int)strlen(bits))-1){
      counter1 = ((int)strlen(bits))-2;
      counter3 = 0; }

if(((int)bits[counter1]==(int)'1') & ((int)bits[counter2]==(int)'1') &
((int)bits[counter3]==(int)'1'))
  if (position <= L3)
    return y = 1.0f;
if(((int)bits[counter1]==(int)'1') & ((int)bits[counter2]==(int)'1') &
((int)bits[counter3]==(int)'0')){
  if (position <= L2)
    return y = 1.0f;
if ((position > L2) & (position <= L3))
    return y = (1.0f/L1)*(L3-position); }
if(((int)bits[counter1]==(int)'1') & ((int)bits[counter2]==(int)'0') &
((int)bits[counter3]==(int)'1')){
  if (position <= L2)
    return y = 0.0f;
  if ((position > L2) & (position <= L3))
return y = (1.0f/L1)*(position-L2); }//Ostatní část kódu viz. příloha
```

Kód uvedený výše reprezentuje mixážní signál z obrázku 6.2. Prototyp funkce obsahuje 6 parametrů a může být zapsán takto:

float mixer_signal(int position,char bits,int BitsCounter,int L1,int L2,int L3);*

Výstup funkce závisí na parametru *position*, který určuje danou pozici v bloku viz. text výše. Parametr bits obsahuje pole bitů(vodoznak), které se mají vkládat a BitsCounter pozici daného bitu. Parametry *L1,L2 a L3* určují tvar impulsu respektive jeho sklon tak, aby nedocházelo k nechtěnému praskání zvuku při navázání echa na originální signál. Funkce na základě těchto parametrů vytváří periodicky impulsy a tvar impulsu pro každý blok závisí na předchozím a následujícím bitu tak, aby na sebe tento signál správně navazoval. Obrázek 6.3 demonstruje nastavení těchto parametrů na jednoduchém impulsu z mixážního signálu pro bit 1, kde předchozí i následující bit je roven 0.



Obr. 6.3: Modulační impuls s příslušnými hladinami.

$L1, L2$ a $L3$ definují požadované strmosti impulsu a $L3$ zároveň určuje konec bloku jinými slovy $L3 = N$. Dle obrázku 6.3 potom platí pro modulační signál:

$$m_0(n) + m_1(n) = 1 \forall n, \text{ kde } m_1[n] = x[n] \text{ a } m_0[n] = 1 - x[n]. \quad (6)$$

Výsledný vodoznačný signál lze potom zapsat pomocí následujícího vzorce takto:

$$y[n] = x[n] + \propto w_0[n] m_0(n) + \propto w_1[n] m_1[n] \quad (7)$$

, kde $w_1[n]$ a $w_0[n]$ jsou vkládané vodoznaky, které jsou výstupem bloků „One Kernel“ a „Zero Kernel“ z obrázku 6.1 neboli výstupem FIR filtru (4) s odpovídajícím zpožděním.

7 Extrahování vodoznaku

Extrahování vodoznaku metodou Echo Hiding je prováděno pomocí komplexního kepstra definovaného jako inverzní Fourierova transformace logaritmu modulu Fourierovy transformace vodoznačného signálu. Aby bylo možno vodoznak správně extrahovat, je nutná synchronizační procedura k zarovnání vodoznačených bloků.

7.1 Synchronizace signálu

Detekce vodoznaku začíná zarovnáním bloku vodoznaku s detektorem. Ztráta synchronizace způsobí falešnou detekci vodoznaku. Modifikace časového režimu nebo kmitočtové stupnice způsobuje ztrátu synchronizace detektoru. Proto nejzávažnější a nejzáludnější útok je pravděpodobně desynchronizace. Všechny algoritmy vodoznačení předpokládají, že každý detektor je synchronizován před etekcí vodoznaku. Hrubé(silové) hledání vodoznaku je výpočtově neproveditelné. Je třeba rychlý a přesný algoritmus synchronizace. Některé schémata vodoznačení jako modulace repliky nebo metoda echo hiding jsou dost robustní proti jistým útokům na synchronizaci(desynchronizace) a proto je lze také využít pro synchronizaci. Tyto metody jsou užívané jako základní metody na hrubou synchronizaci. Synchronizační kód je proto použit pro nalezení počátku vodoznaku. Návrh kvalitního schématu synchronizace není jednoduché. Chytří útočníci se taktéž pokusí vymyslet sofistikované metody pro desynchronizaci. Proto by algoritmy synchronizace měly být odolné proti těmto útokům.

Existují dva problémy při synchronizaci. První je zarovnání startovního bodu bloku. Toto se využívá zejména při modifikaci časové osy. Například některé kodéry mp3 přidávají na začátek audio souboru okolo 50 ms, což může způsobit nechtěnou desynchronizaci. Druhým problémem je změna časového režimu nebo kmitočtové stupnice úmyslně záludnými útočníky a neúmyslně audio systémy. V každém případě je velmi obtížné nalézt vhodný algoritmus, který by tento problém odstranil.

Modifikace časového režimu je útok v časové oblasti, kdy jsou periodicky vkládány vzorky do zvukového cíle či jejich mazání. Tímto je změněna délka zpracovávaného audio signálu. Naopak změna frekvenční stupnice změní frekvenční složky signálu a poté použije časový režim k zajištění stejné délky

signálu. Tento útok může být implementovaný sofistikovanými technikami zvukového zpracování signálu. Aperiodická modifikace má pak na detekci ještě vyšší vliv. Existuje mnoho charakteristických rysů signálů jako například krátkodobá energie signálu, počet průchodů nulou, tempo, frekvenční centrování a podobně. Některé z nich mohou být použity pro synchronizaci, dokud takovéto rysy jsou invarianty pod útoky.

Přesné zarovnání je konečný cíl synchronizace. V každém případě takové zarovnání není jednoduché. Hrubá synchronizace je potřebná k rychlé a efektivní lokalizaci možné pozice. Po hrubé metodě identifikující takové pozice následují mechanismy synchronizace užívané pro přesnou synchronizaci. Je možno tedy říct, že hrubé schéma zarovnání(synchronizace) by mělo být jednoduché a rychlé. Kombinace krátkodobé energie a počtu průchodů nulou je dobrý příklad pro hrubé schéma zarovnání. Tato metoda byla použita i v této práci pro možnou lokalizaci vodoznaku.

Krátkodobou energii signálu lze vypočítat pomocí následujícího vzorce:

$$E_n = \frac{1}{\mathrm{N}} \sum_{n=0}^{N-1} [x(n)]^2 \tag{8}$$

, kde $x(n)$ je vstupní signál a $\mathrm{n} = 0,1,2,\ldots,\mathrm{N}-1$, kde N je délka rámce. V prostředí C++ lze pak tento vzorec realizovat například pomocí následujícího kódu:

```
float E = 0.0f;
      for(int n=0;n<N;n++){
            E=E + x[n]*x[n];
            }
  E=E/N;
```

Krátkodobou funkci středního počtu průchodu signálu nulou lze potom vyjádřit takto:

$$Z_n = \frac{1}{2} \sum_{n=0}^{N-1} |sign\ x[\mathrm{n}] - sign\ \mathrm{x}[\mathrm{n}-1]| \tag{9}$$

, kde

$$\mathrm{sign\ x[n]} = \begin{cases} 1 \text{ pro } & \mathrm{x[n]} \geq 0 \\ -1 \text{ pro } & \mathrm{x[n]} < 0 \end{cases}.$$

V prostředí C++ lze pak tento vzorec opět realizovat například pomocí následujícího kódu:

```
float Zn = 0.0f;
int x1,x2;
```

```
for (int n = 1;n<N;n++){
    if(x[n]>=0) x1=1;
    else x1=-1;
    if(x[n-1]>=0) x2=1;
    else x2=-1;
    Zn = Zn + abs(x1-x2);
}
   Zn = Zn/2;
```

7.2 Princip Detekce vodoznaku

Po synchronizaci signálu je možno použít následující postup pro detekci vodoznaku, kde každý blok o délce N nese informaci o jednom bitu 0 či 1.

1. Rozdělení signálu do M bloků od místa synchronizace.
2. Transformace jednotlivých bloků do kepstrální oblasti pomocí vzorce:

$$\boldsymbol{C_w} = \mathrm{F}^{-1}\{\log(|\mathrm{F}\{\boldsymbol{c_w}\}|)\} \tag{10}$$

3. Autokorelace signálu $\boldsymbol{C_w}$ v kepstrální oblasti.
4. Měření vzdálenosti zpoždění δt přes špičky autokorelační funkce $\boldsymbol{C_w}$.
5. Stanovení vloženého bitu porovnáním δt s Δt_k, $k = 0\ nebo\ 1$.

Mějme signál $x(n)$ reprezentující vstupní audio signál. Tento signál je nejdříve nutno rozdělit do M rámců jako například:

$$M = \frac{l(x)}{N} \tag{11}$$

Následujícím krokem je transformovat jednotlivé rámce pomocí logaritmu diskrétní Fourierovy transformace následovně:

$$\hat{X}[k] = log10 \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} \tag{12}$$

Potom definici komplexního kepstra je možno určit jako zpětnou diskrétní Fourierovu transformaci modulu signálu $\hat{X}[k]$ takto:

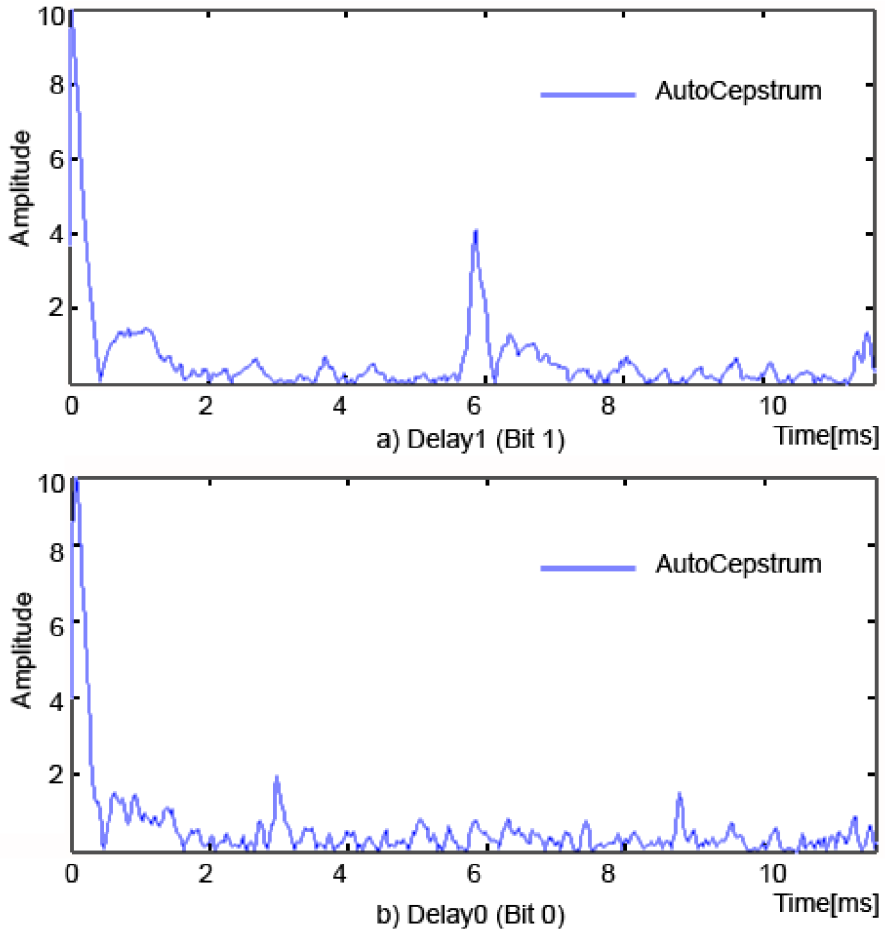
$$\hat{x}[n] = \frac{1}{N}\sum_{k=0}^{N-1}\left|\hat{X}[k]\right| e^{j\frac{2\pi}{N}kn} \tag{13}$$

Po převodu do kepstrální oblasti následuje výpočet autokorelační funkce kepstrálních koeficientů dle vzorce:

$$\widehat{\boldsymbol{y_{corr}}}[\boldsymbol{n}] = \sum_{n=0}^{N-1-m} \hat{x}[n]\hat{x}[n+m] \tag{14}$$

Autokorelace měří podobnost signálů a pokud je zde existence ozvěny (echa), je tato podobnost zobrazena jako špička v místě, kde má tato podobnost počátek jinými slovy zpoždění δt, které odpovídá předdefinovanému

zpoždění Δt_k pro $k = 0\ nebo\ 1$. Na základě pozice špičky je pak stanoven vložený bit v daném bloku. Danou situaci znázorňuje obrázek 7.1.



Obr. 7.1: Detekce zpoždění signálu korelací kepstra (AutoCepstrum). a) Detekce bitu 1 se zpožděním $\Delta t_1 = 6\ ms$. b) Detekce bitu 0 se zpožděním $\Delta t_0 = 3\ ms$.

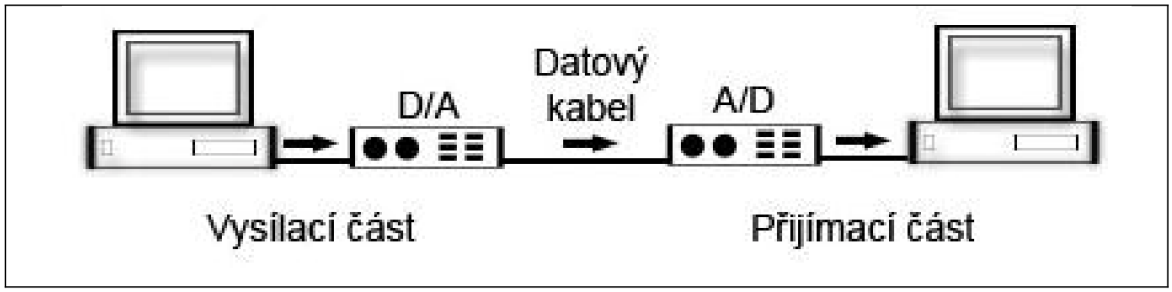
8 Přenosový systém

Předchozí kapitoly popisovaly systém vodoznačení z hlediska implementace algoritmu a jeho funkce. Následující kapitola popisuje vodoznačení signálu z hlediska jeho přenosu. Tento přenos může být realizován datovou cestou jako přenos audio dat sítí, datovým kabelem nebo přenos audio signálu vzduchem (prostředím) pomocí akustických vln. Každý přenosový systém zavádí cestou signálu určité ztráty. Tyto ztráty lze rozdělit na ztráty způsobené přenosem signálu datovou cestou a ztráty způsobené vlivem prostředí.

8.1 Přenos vodoznačného signálu datovou cestou

Při přenosu signálu datovou cestu dle obrázku 8.1 je signál konvertován pomocí DA/AD převodníku. Během DA/AD převodu trpí digitální signál zejména těmito vlivy:

- Šum produkovaný zvukovou kartou během D/A konverze.
- Změna energie audio signálu a šumu.
- Šum v analogovém kanálu (Datový kabel).
- Šum produkovaný zvukovou kartou během A/D konverze včetně kvantizačního šumu.



Obr. 8.1: Základní schéma pro přenos vodoznačného audio signálu pomocí přenosového prostředí.

Při přenosu signálu analogovou cestou dochází ke zpoždění (posunu) přenášených vzorků a při zpětné A/D konverzi jsou jednotlivé vzorky zaznamenány nepřesně. Velikost změny časového měřítka je různá pro různé zvukové karty. Jinými slovy během DA/AD konverze různé nastavení zvukové karty způsobí různé rozostření signálu.

Změna měřítka je také závislá na vzorkovací frekvenci audio nahrávek. Pro stejnou zvukovou kartu a různé vzorkovací frekvence audio nahrávek platí různé rozostření časové osy. Odstranění tohoto problému se provádí pomocí

metod interpolace signálu. Základní metody pro interpolaci lze nalézt v kapitole 4.3. Kromě změny časového měřítka způsobuje AD/DA převod rozostření amplitudy vln což je v podstatě změna energie signálu a jeho ovlivnění přídavným šumem. Je pozorováno, že amplitudy signálu se mění během DA/AD konverze a množství změn je závislé na hlasitosti přehrávaného signálu. Při porovnání přehrávaného a zaznamenaného signálu je pak možno pozorovat snížení amplitudy zaznamenaného signálu. Změna amplitudy závisí na konkrétní zvukové kartě.

Pro matematické vyjádření se používá výpočet odstupu signálu od šumu *SNR (Signal to Noise Ratio), který lze vypočíst jako:*

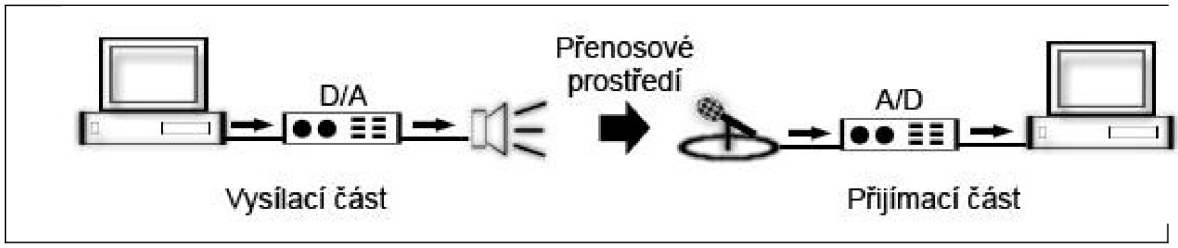
$$SNR = -10log_{10}\left(\frac{\sum_{n=1}^{N}[x(n)-x'(n)]^2}{\sum_{n=1}^{N}[x(n)]^2}\right)\ [\mathrm{dB}] \tag{15}$$

, kde $x(n)$ je diskrétní signál vysílací části a $x'(n)$ je diskrétní signál přenesený datovým kabelem a uložen přijímací části dle obrázku 8.1. Jestliže je k dispozici originální signál, lze provézt normalizaci signálu $x'(n)$ podle:

$$x''(n) = x'(n)\frac{\sum_{n=0}^{N-1}|x(n)|}{\sum_{n=0}^{N-1}|x'(n)|}. \tag{16}$$

, kde $x(n)$ je originální signál, $x'(n)$ zaznamenaný signál a $x''(n)$ je normalizovaný signál. Zaznamenaný signál by měl být nejdříve normalizován podle (16), poté provedena synchronizační procedura podle kapitoly 7.1 a nakonec výpočet SNR.

Pro přenos vodoznačného signálu pomocí přenosového prostředí a záznam pomocí externího mikrofonu platí stejná pravidla s tím rozdílem, že audio signál je při přenosu prostředím navíc degradován ztrátami prostředí a případným aditivním hlukem či šumem. Detekce vodoznaku je pak obtížnější. Tuto situaci znázorňuje obrázek 8.2.



Obr. 8.2: Základní schéma pro přenos vodoznačného audio signálu pomocí přenosového prostředí.

9 Robustnost Algoritmu

Jak bylo popsáno v kapitole 5, hlavním kritériem vkládaného vodoznaku do audio signálu je jeho robustnost a neslyšitelnost, proto je nedílnou součástí návrhu systému vodoznačení testování jeho robustnosti pro zhodnocení jeho efektivity.

Pro vyjádření robustnosti vodoznaku je možno vzorec pro bitovou chybovost BER(Bit Error Rate) takto:

$$BER = \frac{Be}{N} \tag{17}$$

, kde Be je počet chybně přijatých bitů a N celkový počet bitů. Vyjádření neslyšitelnosti vodoznaku je pak možno vyjádřit pomocí pomocí vzorce pro výpočet odstupu signálu od šumu SNR(Signal to Noise Ratio) dle vzorce (15).

Pro testování robustnosti je možno použít řadu technik, které se snaží vložený vodoznak poškodit či znehodnotit jako například:

- Mp3 komprese signálu
- Filtrace DP, HP, PP
- Převzorkování signálu (Decimace->Interpolace)

Testovaná nahrávka	Fvz [Hz]	BER [%]	SNR [dB]
Pop	44100	15.24	9.8
Jazz	44100	23.24	11.2
Klasika	44100	25.60	12.4
Rock	44100	16.56	8.1

Tab. 9.1: Výsledky testování nekomprimovaných nahrávek

Testovaná nahrávka	Fvz [Hz]	BER [%]	SNR [dB]
Pop	44100	40.15	10.00
Jazz	44100	40.20	10.00
Klasika	44100	45.86	10.00
Rock	44100	27.05	10.00

Tab. 9.2: Výsledky testování mp3 komprimovaných nahrávek

Testovaná nahrávka	Fvz [Hz]	BER [%]	SNR [dB]
Pop	44100	55.90	10.00
Jazz	44100	50.10	10.00
Klasika	44100	35.20	10.00
Rock	44100	32	10.00

Tab. 9.3: Výsledky testování decimovaných (podvzorkovaných) nahrávek

Testovaná nahrávka	Fvz [Hz]	BER [%]	SNR [dB]
Pop	44100	35.10	10.00
Jazz	44100	60.50	10.00
Klasika	44100	45.60	10.00
Rock	44100	29.60	10.00

Tab. 9.4: Výsledky testování nahrávek filtrovaných DP

Testovaná nahrávka	Fvz [Hz]	BER [%]	SNR [dB]
Pop	44100	65.23	10.00
Jazz	44100	60.40	10.00
Klasika	44100	41.80	10.00
Rock	44100	49.60	10.00

Tab. 9.5: Výsledky testování nahrávek filtrovaných HP

10 Závěr

Cílem této práce bylo vytvoření algoritmu pro vodoznačení audio signálů, který by za pomoci technologie VST vkládal vodoznak do audio signálu v reálném čase.

První část práce se zabývala testováním a popisem technologie VST (Virtual Steinberg Technology), ASIO ovladačů a tvorbou VST Plug-IN modulů, které slouží jako zásuvné moduly pro audio aplikace podporující ASIO ovladače a technologii VST. Za pomoci této technologie ve spolupráci s ASIO ovladači bylo možné provádět zpracování audio obsahu v reálném čase s latencí signálu v rozmezí 0-20 ms. Tyto hodnoty závisely na použitém algoritmu VST Plug-in modulu a výkonu PC, neboť mají ASIO ovladače vyšší nároky na CPU, než klasické WDM/MME ovladače využívané operačním systémem Windows.

V druhé části byly pak popsány techniky vodoznačení audio signálů za pomoci vytvořeného VST Plug-IN modulu, zejména pak metoda Echo Hiding, která vkládá vodoznaky v časové oblasti a detekci provádí v oblasti kepstrální.

Tato metoda byla nakonec implementována jako zmíněný VST Plug-IN.

Z testování lze vyvodit, že se jedná o robustní metodu, která neklade velké nároky na využití strojového času při procesu vkládání vodoznaku, a proto bylo možno provádět vodoznačení dat v reálném čase. Při testování robustnosti vodoznak odolalával modifikacím jako je mp3 komprese využívající psychoakustického modelu, filtrací DP, HP, PP a převzorkování audio signálu. Efektivita detekce pak závisela hlavně na velikosti vkládaného vodoznaku a druhu audio nahrávky. Typická hodnota BER (Bitová chybovost) se pohybovala v rozmezí 10-50 % při odstupu výkonu originálního signálu k výkonu vkládaného vodoznaku přibližně 15 dB.

Literatura

[1] Jiří Schimmel, Implementace algoritmů číslicového zpracování signálů pomocí technologie VST.

[2] MILLWARD, S. Sound Synthesis with VST Instrument. 1st edition: PC Publishing, 2002, 277 p. ISBN 1870775732

[3] VST Plug-Ins SDK documentation. Dostupné z WWW: <http://ygrabit.steinberg.de/~ygrabit/public_html/vstgui/V2.2/doc/>

[4] How to build VST plugins on Windows with Visual Studio Express. Dostupné z WWW: <http://www.teragon.org/code/visual-studio-vst-howto.html>

[5] KVR Audio Plug-in News. (Informační server zabývající se VST, DirectX a audio plug-in moduly a DSP). Dostupné z WWW:< http://www.kvraudio.com/forum/>

[6] Předmět MCAS, VUT BRNO

[7] W. Bender, D. Gruhl, N. Morimoto, A. Lu, “Techniques for data hiding,” http://www.research.ibm.com/journal/sj/mit/sectiona/bender.html, 1996.

[8] Arnold M; Schmucker M; Wolthusen S; Techniques and applicatios of digital watermarking and content protection artech. House, inc, 2003. ISBN 1-58053-111-3.

[9] Idea Group - Multimedia Security - Steganography And Digital Watermarking Techniques For Protection Of Intellectual Property – 2005.

Seznam symbolů, veličin a zkratek

Echo Hiding – metoda vodoznačení

ASIO - Audio Stream Input Output

DSP – Digital Signal Processing

GUI – Graphic User Interface

VST - Virtual Studio Technology

VSTGUI – VST Graphic User Interface

VSTSDK – VST Software Development Kit

VC++ - Visual C++

OS – Operating System (Operační systém)

FFT – Fast Fourier transform (Rychlá Fourierova transformace)

PC – Personal Computer

PN – Pseudo-Noise Sequence

dB – Decibel

ms - Milisecond

sample – Vzorek

Fvz – Vzorkovací frekvence

Cw – Kepstrální koeficienty

Sync – Synchronizace

HP, PP, DP – Horní, pásmová a dolní propust

BER – Bit Error Rate

SNR – Signal to Noise Ratio

Seznam příloh

A Zdrojový kód VST Plug-IN

```
//-------------------------------------------------------------------------
-----
// VST Plug-Ins SDK
// Version 2.4          $Date: 2006/11/13 09:08:27 $
//
// Category     : VST 2.x SDK Samples
// Filename     : AEcho_Hiding_Wtm.cpp
// Created by   : Steinberg Media Technologies
// Description  : Simple Delay plugin (Mono->Stereo)
//
// © 2006, Steinberg Media Technologies, All Rights Reserved
//-------------------------------------------------------------------------
-----
//-------------------------------------------------------------------------
-----
// Date: 2008/05/10
//
// Category     : VST 2.x SDK Samples
// Filename     : AEcho_Hiding_Wtm.h
// Created by   : David Henzl
// Description  : Audio Watermarking(Echo Hiding) (Mono->Stereo)
//-------------------------------------------------------------------------
-----------------------------------


#ifndef __AEcho_Hiding_Wtm__
#define __AEcho_Hiding_Wtm__

#include "../vst2.x/audioeffectx.h"
#include "../editor/VSTGLEditor.h"

#define PI 3.14159265358979323846
#define M_PI 3.14159265358979323846

class AEchoHidingProgram;

//-------------------------------------------------------------------------
----
class AEcho_Hiding_Wtm : public AudioEffectX
{
public:
      AEcho_Hiding_Wtm (audioMasterCallback audioMaster);
      ~AEcho_Hiding_Wtm ();

      //---from AudioEffect-----------------------
      virtual void processReplacing (float** inputs, float** outputs,
VstInt32 sampleFrames);
      //Funkce volaná funkcí processReplacing. Slouží ke vložení
vodoznaku do audio signálu.
      virtual void subProcessInserting (float *inL,float *inR, float
*outL, float *outR, long sampleFrames);
      //Funkce volaná funkcí processReplacing. Slouží k detekci
vodoznaku v audio signálu.
      virtual void subProcessDecoding (float *inL,float *inR, float
*outL, float *outR, long sampleFrames);
      //Rychlá Fourierova transformace FFT. Využita k detekci
vodoznaku.
```

```
	virtual void fft(float *x,float *y,int order,int param);
	//Funkce pro mixování signálu, která generuje modulační signál.
	//Slouží ke složení originálního signálu a směsi zpoždění.
	virtual float mixer_signal(float f,int position,char* bits,int
counter,int L1,int L2,int L3);
	//Převod vodoznaku(String) na posloupnost bitů.(Inserting Mode)
	virtual int Wtm2Bin(char *Wtm);
	//Převod posloupnosti bitů na vodoznak(String).(Decoding Mode)
	virtual int Bin2Wtm(int DecBits[],int i);
	//Funkce volána funkcí subProcessDecoding. Slouží k dekódování
vložených vodoznaků
	virtual void Decode(int DecBits[],int BitsCounter);
	VstIntPtr test_buff;
	//Nastavení vodoznaku
	virtual void setWatermark (char wattermark[]);
	/*******Funkce nutné pro běh VST plug-in modulu***************/
	virtual void setProgram (VstInt32 program);
	virtual void setProgramName (char* name);
	virtual void getProgramName (char* name);
	virtual bool getProgramNameIndexed (VstInt32 category, VstInt32
index, char* text);

	virtual void setParameter (VstInt32 index, float value);
	virtual float getParameter (VstInt32 index);
	virtual void getParameterLabel (VstInt32 index, char* label);
	virtual void getParameterDisplay (VstInt32 index, char* text);
	virtual void getParameterName (VstInt32 index, char* text);

	virtual void resume ();
	virtual void suspend ();
	virtual bool getEffectName (char* name);
	virtual bool getVendorString (char* text);
	virtual bool getProductString (char* text);
	virtual VstInt32 getVendorVersion () { return 1000; };

	virtual VstPlugCategory getPlugCategory () { return
kPlugCategEffect; };
	/*************************************************************/
	//-------------------------------------------------------------
	enum
	{
		// Global
		kNumPrograms = 16,

		// Parameters Tags
		kDelay0 = 0,
		kDelay1,
		kAlfa,
		kMode,
		kVSTblok,
		kWtmblok,
		kWitchBlok,
		kWatermark,
		kNumParams,
		kWavDisp,
		kWavDispSize,
		kCharCounter,

	};

protected:
```

```
      float triangle (float f, float position);
      AEchoHidingProgram* programs;

      float params[kNumParams+4];

      int* my_index;
      float* bufferL;
      float* bufferR;
      float* fftbuffer;
      float* ifftbuffer;
      float* corr;
      float* decodedbits;
      float* y2_1;
      float* y2_2;
      //float* x;
      char Wtm[20];
      char bits[200];
      int DecBits[20000];
      int RecBits[250][80];
      int* SumWtm;//rekonstruovany vodoznak
      float xx[512];
      int* one;
      int* zer;
      int partWtm;
      int syncposition;
      int StartDec,StopDec;
      int Wtmlength;
      long delay;
      long size;
      int pSize;
      int fftsize;
      long cursor;
      long rcursor;
      long wcursor;
      float VolumeUnit;
      VSTGLEditor* GLeditor;

      int delay0;
      int delay1;
      float Alfa;
      int position;
      float stepDec;
      float stepSum;
      int stepRecBits;
      int stepSingleWtm;
      float Energy;
      float ZeroCross;
      bool  canWatermark,canDetect,sync;
      int   N;
      int   N2;
      int   blockCounter;
      int   BitsCounter;
      int   NumberOfBits;
      int L1,L2,L3;  //indexy pro mixer_signal

};
//-------------------------------------------------------------------------
----
// Nastavení programů(parametrů) plug-inu
class AEchoHidingProgram
```

```
{
	friend class AEchoHidingProgram;
public:
	AEchoHidingProgram ();
	~AEchoHidingProgram () {};

	//private:
	float params[AEcho_Hiding_Wtm::kNumParams];
	char name[24];
	char Wtm[16];

};
#endif
```

```
//-------------------------------------------------------------------------
-----
// VST Plug-Ins SDK
// Version 2.4          $Date: 2006/11/13 09:08:27 $
//
// Category     : VST 2.x SDK Samples
// Filename     : AEcho_Hiding_Wtm.cpp
// Created by   : Steinberg Media Technologies
// Description  : Simple Delay plugin (Mono->Stereo)
//
// © 2006, Steinberg Media Technologies, All Rights Reserved
//-------------------------------------------------------------------------
-----
//-------------------------------------------------------------------------
-----
// Date: 2008/05/10
//
// Category     : VST 2.x SDK Samples
// Filename     : AEcho_Hiding_Wtm.cpp
// Created by   : David Henzl
// Description  : Audio Watermarking(Echo Hiding) (Mono->Stereo)
//-------------------------------------------------------------------------
-----
#include <stdio.h>
#include <string.h>
#include <vstcontrols.h>
#include <complex>
#include <math.h>

#ifndef __AEcho_Hiding_Wtm__
#include "echo_hiding_wtm.h"
#endif

#ifndef __Editor__
#include "../editor/my_editor.h"
#endif
//-------------------------------------------------------------------------
---------
AEchoHidingProgram::AEchoHidingProgram ()
{
	// default Program Values
	params[AEcho_Hiding_Wtm::kDelay0] = 128.0f;
	params[AEcho_Hiding_Wtm::kDelay1] = 256.0f;
	params[AEcho_Hiding_Wtm::kAlfa] = 0.5f;
	params[AEcho_Hiding_Wtm::kMode] = 0.0f;
	params[AEcho_Hiding_Wtm::kVSTblok] = 0.0f;
	params[AEcho_Hiding_Wtm::kWtmblok] = 0.0f;
```

```
	params[AEcho_Hiding_Wtm::kWitchBlok] = 0.0f;
	params[AEcho_Hiding_Wtm::kWatermark] = 0.0f;
	strcpy (name, "Init");
}
//-------------------------------------------------------------------------
---------
AEcho_Hiding_Wtm::AEcho_Hiding_Wtm (audioMasterCallback audioMaster)
: AudioEffectX (audioMaster, kNumPrograms, kNumParams)
{
	// init
	size = 44100;
	Wtmlength=10;//pevna delka vodoznaku
	stepDec=(Wtmlength+1)*8;
	stepSum=200;
	stepRecBits=0;//max. 10*8bitu
	stepSingleWtm=0;//max. 250 vodoznaku
	pSize = 882;  //Info pro WaveDisplay
	fftsize = 2048;
	cursor = 0;
	rcursor = 0;
	wcursor = 0;
	blockCounter = 0;
	BitsCounter  = 0;
	syncposition=0;
	StartDec = 0;
	StopDec = 0;

	L1=0;L2=0;L3=0;
	delay0=0;
	delay1=0;
	Energy=0.0f;
	N=0;
	N2=0;
	ZeroCross=0.0f;
	canWatermark = false;
	canDetect = false;
	sync = false;
	partWtm=0;
	//delay = 100;
	VolumeUnit = 1;
	bufferL = new float[size];
	bufferR = new float[size];
	SumWtm = new int[(Wtmlength+1)*8];
	one = new int[20000];
	zer = new int[20000];
	fftbuffer = new float[fftsize];
	ifftbuffer = new float[fftsize];
	corr = new float[fftsize];
	decodedbits = new float[size];
	test_buff = 0;
	y2_1 = new float[size];   //
	y2_2 = new float[size];
	position = 0;
	my_index = 0;
	programs = new AEchoHidingProgram[numPrograms];
	programs[0].params[kDelay0] = 128.0f;
	programs[0].params[kDelay1] = 256.0f;
	programs[0].params[kAlfa] = 0.5f;
	programs[0].params[kMode] = 0.0f;
	programs[0].params[kVSTblok] = getBlockSize();
	programs[0].params[kWtmblok] = 512.0f;
```

```
	programs[0].params[kWitchBlok] = 0.0f;
	programs[0].params[kWatermark] = 0.0f;
	strcpy (programs[0].name,"Echo Hiding Wtm");
	strcpy (programs[0].Wtm,"WATERMARK!");

	strcpy (Wtm,"WATERMARK!");
	NumberOfBits=Wtm2Bin(Wtm);//Vytvoření bitové posloupnosti
(výsledek uložen v proměnné "bits", funkce vrací počet bitů)

	if (programs)
		setProgram (0);

	setNumInputs (2); // stereo input
	setNumOutputs (2);	// stereo output
	canProcessReplacing ();
	setUniqueID ('AWtm');	// this should be unique, use the
Steinberg web page for plugin Id registration

	// create the editor
	editor = new Editor(this,pSize,Wtm);
	resume ();		// flush buffer
}

//-------------------------------------------------------------------------
----
AEcho_Hiding_Wtm::~AEcho_Hiding_Wtm ()
{
	if (bufferL)
		delete[] bufferL;
	if (bufferR)
		delete[] bufferR;
	if (fftbuffer)
		delete[] fftbuffer;
	if (ifftbuffer)
		delete[] ifftbuffer;
	if (decodedbits)
		delete[] decodedbits;
	if (y2_1)
		delete[] y2_1;
	if (y2_2)
		delete[] y2_2;
	if (programs)
		delete[] programs;

	GLeditor->close();
}
//-------------------------------------------------------------------------
----
void AEcho_Hiding_Wtm::setProgram (VstInt32 program)
{
	int i;
	AEchoHidingProgram* prog = &programs[program];
	curProgram = program;
	for(i=0;i<numParams;i++){
		setParameter(i, prog->params[i]);
		if (editor)
			((AEffGUIEditor*)editor)->setParameter (i, prog-
>params[i]);
	}
}
//-------------------------------------------------------------------------
```

```
----
void AEcho_Hiding_Wtm::setWatermark (char wattermark[])
{
      strcpy(programs[curProgram].Wtm,wattermark);
}
//------------------------------------------------------------------------
----
void AEcho_Hiding_Wtm::setProgramName (char *name)
{
      strcpy (programs[curProgram].name, name);
}
//------------------------------------------------------------------------
----
void AEcho_Hiding_Wtm::getProgramName (char *name)
{
      /*if (!strcmp (programs[curProgram].name, "Init"))
      sprintf (name, "%s %d", programs[curProgram].name, curProgram +
1);
      else*/
      strcpy (name, programs[curProgram].name);
}

//------------------------------------------------------------------------
----------------------
bool AEcho_Hiding_Wtm::getProgramNameIndexed (VstInt32 category,
VstInt32 index, char* text)
{
      if (index < kNumPrograms)
      {
            strcpy (text, programs[index].name);
            return true;
      }
      return false;
}
//------------------------------------------------------------------------
----
void AEcho_Hiding_Wtm::resume ()
{
      memset (bufferL, 0, size * sizeof (float));
      memset (bufferR, 0, size * sizeof (float));
      memset (fftbuffer, 0, fftsize * sizeof (float));
      memset (ifftbuffer, 0, fftsize * sizeof (float));
      memset (one, 0, 10000 * sizeof (float));
      memset (zer, 0, 10000 * sizeof (float));
      memset (corr, 0, fftsize * sizeof (float));
      memset (decodedbits, 0, size * sizeof (float));
      memset (SumWtm, 0, Wtmlength*8 * sizeof (int));
      memset (y2_1, 0, size * sizeof (float));
      memset (y2_2, 0, size * sizeof (float));

      AudioEffectX::resume ();
}
//------------------------------------------------------------------------
----
void AEcho_Hiding_Wtm::suspend ()
{
      memset (bufferL, 0, size * sizeof (float));
      memset (bufferR, 0, size * sizeof (float));
      AudioEffectX::suspend ();
}
//------------------------------------------------------------------------
```

```
----
void AEcho_Hiding_Wtm::setParameter (VstInt32 index, float value)
{
      AEchoHidingProgram* prog = &programs[curProgram];
      params[index] = value;
      prog->params[index] = value;
}
//------------------------------------------------------------------------
----
float AEcho_Hiding_Wtm::getParameter (VstInt32 index)
{
      return params[index];
}
//------------------------------------------------------------------------
----
void AEcho_Hiding_Wtm::getParameterName (VstInt32 index, char *label)
{
      switch (index)
      {
      case kDelay0 :          strcpy (label, "Delay0");     break;
      case kDelay1 :          strcpy (label, "Delay1");     break;
      case kAlfa :            strcpy (label, "Alfa");       break;
      case kWatermark : strcpy (label, "Watermark");break;
      }
}
//------------------------------------------------------------------------
----
void AEcho_Hiding_Wtm::getParameterDisplay (VstInt32 index, char
*text)
{
      /*if (index == kOut)
      {
      dB2string (params[index], text, kVstMaxParamStrLen);
      } else*/
      float2string(params[index], text, kVstMaxParamStrLen);
}

//------------------------------------------------------------------------
----
void AEcho_Hiding_Wtm::getParameterLabel (VstInt32 index, char *label)
{
      switch (index)
      {
      case kDelay0 :          strcpy (label, "Sample");     break;
      case kDelay1 :          strcpy (label, "Sample");     break;
      case kAlfa :            strcpy (label, "Amplitude");break;
      case kWatermark : strcpy (label, "Bits");       break;

      }
}
//------------------------------------------------------------------------
----
bool AEcho_Hiding_Wtm::getEffectName (char* name)
{
      strcpy (name, "Echo_Hiding_Wtm");
      return true;
}
//------------------------------------------------------------------------
----
bool AEcho_Hiding_Wtm::getProductString (char* text)
{
```

```
	strcpy (text, "Echo_Hiding_Wtm");
	return true;
}
//------------------------------------------------------------------------
----
bool AEcho_Hiding_Wtm::getVendorString (char* text)
{
	strcpy (text, "Steinberg Media Technologies");
	return true;
}
//------------------------------------------------------------------------
-------
void AEcho_Hiding_Wtm::processReplacing (float** inputs, float**
outputs, VstInt32 sampleFrames)
{
	float* inL = inputs[0];
	float* inR = inputs[1];
	float* outL = outputs[0];
	float* outR = outputs[1];

	if(params[kWitchBlok]== 0.0f){
		N = (int)sampleFrames;
	}
	else if(params[kWitchBlok]==1.0f){
		N = (int)params[kWtmblok];
	}
	//refresh pri stisku tlacitka Inserting,Decoding nebo při změně
parametů(velikost bloku,vodoznak)
	if(params[kWatermark]==1.0f){
		setWatermark(Wtm);
		NumberOfBits=Wtm2Bin(Wtm); //Vytvoření bitové posloupnosti
vodoznaku a získání počtu bitů
		blockCounter=0;
		BitsCounter=0;
		position =0;
		Energy=0.0f;
		canWatermark=false;
		stepDec=(Wtmlength+1)*8;
		syncposition = 0;
		params[kWatermark]=0.0f;
	}

	//params[kVSTblok]=sampleFrames;
	if (editor){
		((AEffGUIEditor*)editor)->setParameter(kVSTblok,N);

		((AEffGUIEditor*)editor)->setParameter (kWavDispSize, N);
	}
	//Mód Plug-inu (Inserting,Decoding)
	if(params[kMode]==0.0f) //Proces vkládání vodoznaku
		subProcessInserting(inL, inR, outL, outR, sampleFrames);
	else  subProcessDecoding (inL, inR, outL, outR, sampleFrames);

}
//------------------------------------------------------------------------
---------------------------------------
void AEcho_Hiding_Wtm::subProcessInserting (float *inL,float *inR,
float *outL, float *outR, long sampleFrames)
{
	L1=(int)ceil(N/3.0);
	L2=N-L1;
```

```
	L3=N;
	delay0=(int)params[kDelay0];
	delay1=(int)params[kDelay1];
	float causalityL,causalityR;
	float Te=0.00001f;

	//for(int i = 0;i < sampleFrames ;i++)
	while(--sampleFrames >= 0) //0:1:512
	{
		//Naplnění bufferu(slouží k výběru zpožděných vzorků)
		bufferL[position] = *inL;
		bufferR[position] = *inR;
		bufferL[position+N] = *inL;
		bufferR[position+N] = *inR;
		//Výpočet krátkodobé energie(slouží k určení charakteru
zvuku, pokud je větší než T je možno vložit vodoznak)
		Energy = Energy + (*inL)*(*inL);
		//Výpočet funkce počtu průchodu nulou(slouží k určení char.
zvuku[vložení vodoznaku])

		if(blockCounter == 0 || canWatermark == false){
			if(outL)
				*outL++ = (*inL++);
			if(outR)
				*outR++ = (*inR++);
		} else{

		causalityL = 0.0f;
		causalityR = 0.0f;
		//causalityL =params[kAlfa]*bufferL[N-delay0-
2+position]*(1.0f-
mixer_signal(1/N,position,bits,BitsCounter,L1,L2,L3)) +
params[kAlfa]*bufferL[N-delay1-
2+position]*mixer_signal(1.0f/N,position,bits,BitsCounter,L1,L2,L3);
		//causalityR = params[kAlfa]*bufferL[N-delay0-
2+position]*(1.0f-
mixer_signal(1/N,position,bits,BitsCounter,L1,L2,L3)) +
params[kAlfa]*bufferR[N-delay1-
2+position]*mixer_signal(1/N,position,bits,BitsCounter,L1,L2,L3);

		if(outL)
			*outL++ =(*inL++) + params[kAlfa]*bufferL[N-
delay0+position]*(1.0f-
mixer_signal(1/N,position,bits,BitsCounter,L1,L2,L3)) +
params[kAlfa]*bufferL[N-
delay1+position]*mixer_signal(1.0f/N,position,bits,BitsCounter,L1,L2,L
3)+causalityL;
		if(outR)
			*outR++ =(*inR++) + params[kAlfa]*bufferL[N-
delay0+position]*(1.0f-
mixer_signal(1/N,position,bits,BitsCounter,L1,L2,L3)) +
params[kAlfa]*bufferR[N-
delay1+position]*mixer_signal(1/N,position,bits,BitsCounter,L1,L2,L3)+
causalityR;
		}
		if(++position>N){ //pozice 0:blocksize
			if(blockCounter>0 & canWatermark==true)
				BitsCounter++;
			if(BitsCounter>NumberOfBits-1)
				BitsCounter=0;
```

```
                  if(Energy/N > Te){
                        canWatermark=true;
                  }

                  Energy=0.0f;
                  position = 0;
                  blockCounter++;
            }
      }

      if((int)params[kVSTblok]==N){
            if(blockCounter>0 & canWatermark==true)
                  BitsCounter++;
            if(BitsCounter>NumberOfBits-1)
                  BitsCounter=0;

            if(Energy/N > Te){
                  canWatermark=true;
            }

            Energy=0.0f;
            position=0; //vždy pro každý blok opakující se index od 0
do blocksize(512)
            blockCounter++;
      }
}
//-------------------------------------------------------------------------
----------------------------------------
void AEcho_Hiding_Wtm::subProcessDecoding (float *inL,float *inR,
float *outL, float *outR, long sampleFrames)
{
      delay0=(int)params[kDelay0];
      delay1=(int)params[kDelay1];
      float Te=0.00001f;
      float x0=0;
      float x1=0;

      while(--sampleFrames >= 0) //0:1:512
      {
            bufferL[position] = (*inL);
            if(outL)
                  *outL++ = (*inL++);
            if(outR)
                  *outR++ = (*inR++);

            Energy = Energy + (*inL)*(*inL);

            if (editor)
                  ((AEffGUIEditor*)editor)->setParameter (kWavDisp,
corr[position]);

            if(++position>N){

                  if(canDetect == true){
                        memset (fftbuffer, 0, N * sizeof (float));
                        memset (ifftbuffer, 0, N * sizeof (float));
                        memset (corr, 0, N * sizeof (float));
                        fft(bufferL,fftbuffer,9,0);
                        for(int k=0;k<N;k++)
                        {
```

```
	fftbuffer[k]=log(sqrt(bufferL[k]*bufferL[k]*bufferL[k]*bufferL[k]
]
                                +
fftbuffer[k]*fftbuffer[k]*fftbuffer[k]*fftbuffer[k]));
                        }
                        fft(fftbuffer,ifftbuffer,9,1);
                        for(int m=0;m<N;m++){
                                for(int n=0;n<N-m;n++){


	corr[m]=fftbuffer[n]*fftbuffer[n+m];

                                }
                                corr[m]= abs(corr[m]);
                        }
                        x1=corr[delay1-1]+corr[delay1];
                        x0=corr[delay0-1]+corr[delay0];
                        if(x1 > x0){

                                one[BitsCounter]=x1;
                                DecBits[BitsCounter]=1;
                        }
                        else{
                                zer[BitsCounter]=x0;
                                DecBits[BitsCounter]=0;
                        }

                        //Dekódování jednotlivých bitů a převod na
znaky
                        if(BitsCounter>=stepDec){ //>=99
                                Decode(DecBits,BitsCounter);
                                stepDec+=(Wtmlength+1)*8;
                        }


                        if(++BitsCounter>=20000){
                                BitsCounter=0;
                                syncposition=0;
                        }

                }
                if(Energy/N > Te)
                        canDetect = true;

                position = 0;
                blockCounter++;
                Energy=0.0f;
        }
	}//konec cyklu while
}
//-------------------------------------------------------------------
--------------------------------
float AEcho_Hiding_Wtm::mixer_signal(float f,int position,char*
bits,int counter,int L1,int L2,int L3)
{
	float y;
	int counter1=counter-1;
	int counter2=counter;
	int counter3=counter+1;

	if (counter == 0){
```

```
            counter1 = (int)strlen(bits)-1;
            counter3 = counter+1;
      }
      if (counter == ((int)strlen(bits))-1){
            counter1 = ((int)strlen(bits))-2;
            counter3 = 0;
      }

      if(((int)bits[counter1]==(int)'1') &
((int)bits[counter2]==(int)'1') & ((int)bits[counter3]==(int)'1'))
            if (position <= L3)
                  return y = 1.0f;


      if(((int)bits[counter1]==(int)'1') &
((int)bits[counter2]==(int)'1') & ((int)bits[counter3]==(int)'0')){
            if (position <= L2)
                  return y = 1.0f;

            if ((position > L2) & (position <= L3))
                  return y = (1.0f/L1)*(L3-position);
      }

      if(((int)bits[counter1]==(int)'1') &
((int)bits[counter2]==(int)'0') & ((int)bits[counter3]==(int)'1')){
            if (position <= L2)
                  return y = 0.0f;

            if ((position > L2) & (position <= L3))
                  return y = (1.0f/L1)*(position-L2);
      }

      if(((int)bits[counter1]==(int)'1') &
((int)bits[counter2]==(int)'0') & ((int)bits[counter3]==(int)'0')){
            if (position <= L3)
                  return y = 0.0f;
      }

      if(((int)bits[counter1]==(int)'0') &
((int)bits[counter2]==(int)'1') & ((int)bits[counter3]==(int)'0')){
            if(position <= L2)
                  return y = 1.0f;
            if((position > L2) & (position <= L3))
                  return y = (1.0f/L1)*(L3-position);
      }

      if(((int)bits[counter1]==(int)'0') &
((int)bits[counter2]==(int)'1') & ((int)bits[counter3]==(int)'1')){
            if (position <= L3)
                  return y = 1.0f;
      }

      if(((int)bits[counter1]==(int)'0') &
((int)bits[counter2]==(int)'0') & ((int)bits[counter3]==(int)'1')){
            if (position <= L2)
                  return y = 0.0f;
            if ((position > L2) & (position <= L3))
                  return y = (1.0f/L1)*(position-L2);
      }

      if(((int)bits[counter1]==(int)'0') &
```

```
((int)bits[counter2]==(int)'0') & ((int)bits[counter3]==(int)'0')){
            if (position <= L3)
                  return y = 0.0f;
      }
      return y=0.0f;
}
//-------------------------------------
int AEcho_Hiding_Wtm::Wtm2Bin (char* Wtm)
{
      int znaku=((int)strlen(Wtm));
      int bitu=Wtmlength*8;
      strcpy(bits,"11111111");
      for(int i=1;i<Wtmlength+1;i++){
            int num = (int)Wtm[i-1];

            for(int j=0;j<8;j++){
                  if(num%2==0)
                        bits[j+i*8]='0';
                  else bits[j+i*8]='1';
                  num = num >> 1;
            }

      }
      bits[(znaku+1)*8]='\0';

      return bitu+8;
}
//-----------------------------------------------
int AEcho_Hiding_Wtm::Bin2Wtm (int DecBits[],int i)
{

      int num = 0;
      int nas = 1;
      int bit = 0;

      for(int n=0;n<8;n++){

            if(DecBits[n+i] == 1)
                  bit=1;
            else bit = 0;
            num = num + bit*nas;
            nas = nas*2;

      }
      if(num == 0)
            return 48;

      return num;
}
//-----------------------------------------------------------------
void AEcho_Hiding_Wtm::fft(float *x,float *y,int order,int param)
{
      unsigned int n,l,e,f,i,j,o,o1,j1,i1,k;
      double u,v,z,c,s,p,q,r,t,w,a;
      int FFT=0;

      n=1u<<order;

      for(l=1;l<=order;l++)
      {
            u=1.0;
```

```
            v=0.0;
            e=1u<<(order-l+1);
            f=e/2;

            z=M_PI/f;

            c=cos(z);
            s=sin(z);

            if(param==FFT) s=-s;
            //if(param==IFFT) s=s;

            for(j=1;j<=f;j++)
            {
                  for(i=j;i<=n;i+=e)
                  {
                        o=i+f-1;
                        o1=i-1;
                        p=x[o1]+x[o];
                        r=x[o1]-x[o];
                        q=y[o1]+y[o];
                        t=y[o1]-y[o];
                        x[o]=r*u-t*v;
                        y[o]=t*u+r*v;
                        x[o1]=p;
                        y[o1]=q;
                  }
                  w=u*c-v*s;
                  v=v*c+u*s;
                  u=w;
            }
      }
      j=1;
      for(i=1;i<n;i++)
      {
            if(i<j)
            {
                  j1=j-1;
                  i1=i-1;
                  p=x[j1];
                  q=y[j1];
                  x[j1]=x[i1];
                  y[j1]=y[i1];
                  x[i1]=p;
                  y[i1]=q;
            }

            k=n/2;

            while(k<j)
            {
                  j=j-k;
                  k=k/2;
            }

            j+=k;
      }

      if(param==FFT) return;

      a=1.0/n;
```

```
	for(k=0;k<n;k++)
	{
		x[k]*=a;
		y[k]*=a;
	}
	return;
}
//-------------------------------------------------------------
void AEcho_Hiding_Wtm::Decode(int DecBits[],int BitsCounter)
{
	int znak=0;
	StartDec = 0;
	StopDec = 0;
	//nalezení synchronizační sekvence
	for(int i = 0+syncposition;i<BitsCounter;i++){

		znak = Bin2Wtm(DecBits,i);
		if(znak == 255){ //sync
			sync = true;
			syncposition=i;
			StartDec=syncposition+8;
			StopDec = StartDec+Wtmlength*8;

			//if (editor)
			//((AEffGUIEditor*)editor)->setParameter
(kCharCounter,(float)znak);
			break;
		}

	}
	if(sync){
		for(int n = StartDec;n<StopDec;n=n+8){

			znak = Bin2Wtm(DecBits,n);
			if(znak == 255){
				syncposition = n;
				break;
			}
			if (editor)
				((AEffGUIEditor*)editor)->setParameter
(kCharCounter,(float)znak);
			syncposition = n+8;
		}
	}
	sync = false;
	return;
}
//-------------------------------------------------------------
float AEcho_Hiding_Wtm::triangle (float f, float position)
{
	float maxpos =  f/2;
	float triang = 0.0f;

	if(position <= (maxpos))
		triang = (1/maxpos)*position;

	if(position > (maxpos) && position <= 2*(maxpos))
		triang = (1/maxpos)*(2*maxpos - position);

	/*if(position > 2*(maxpos) && position <= (3)*(maxpos))
	triang = (1/maxpos)*position;
```

```
	//triang = -(1/maxpos)*(position - 2*maxpos);

	if(position > (3)*(maxpos) && position <= 4*(maxpos))
	triang = (1/maxpos)*(2*maxpos - position);
	//triang = -(1/maxpos)*(4 * maxpos - position);

	*/
	return triang;
}
```

```
//-------------------------------------------------------------------------
---------------------------------
// VST Plug-Ins SDK
// Version 2.4          $Date: 2006/11/13 09:08:28 $
//
// Category     : VST 2.x SDK Samples
// Filename     : Editor.h
// Created by   : Steinberg Media Technologies
// Description  : Simple Surround Delay plugin with Editor using
VSTGUI
//
// © 2006, Steinberg Media Technologies, All Rights Reserved
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
// Date: 2008/05/10
//
// Category     : VST 2.x SDK Samples
// Filename     : my_editor.h
// Created by   : David Henzl
// Description  : GUI Editor for AWtm(Echo Hiding) (Mono->Stereo)
//-------------------------------------------------------------------------

#ifndef __my_editor__
#define __my_editor__

// include VSTGUI
#ifndef __vstgui__
#include "../vstgui/vstgui.h"
#endif
//Zobrazení kepstrální oblasti
#ifndef __waveform__
#include "../editor/waveform.h"
#endif
#ifndef __AEcho_Hiding_Wtm__
#include "../source/echo_hiding_wtm.h"
#endif
//-------------------------------------------------------------------------

class Editor : public AEffGUIEditor, public CControlListener
{
public:
	Editor (AudioEffect *effect,int pSizeWaveDisp, char* Wtm);
	virtual ~Editor ();

public:
	virtual bool open (void* ptr);
	virtual void close ();
	virtual void setParameter (VstInt32 index, float value);
	virtual void valueChanged (CDrawContext* context, CControl*
control);
	virtual char* getwatermark(void);
```

```
private:
	// Controls
	COnOffButton*	button1;
	COnOffButton*	button2;
	CAnimKnob*  Delay0Val;
	CAnimKnob*  Delay1Val;
	CAnimKnob*  AlfaVal;
	CParamDisplay* Delay0Display;
	CParamDisplay* Delay1Display;
	CParamDisplay* AlfaDisplay;
	CParamDisplay* VSTBlokDisplay;

	CWaveform*  WaveDisplay;
	//Text
	CTextEdit* Delay0Txt;
	CTextEdit* Delay1Txt;
	CTextEdit* AlfaTxt;
	CTextEdit* VSTBlokTxt;
	CTextEdit* WatermarkTxt;
	CTextEdit* Watermark2Txt;
	CTextEdit* ModeTxt;
	CTextEdit* BlocksizeinfoTxt;
	CTextEdit* InsertingTxt;
	CTextEdit* DecodingTxt;
	CTextEdit* InsertWtmTxt;
	CTextEdit* DecodedWtmTxt;

	const char* Delay0;
	const char* Delay1;
	const char* Alfa;
	const char* VSTblok;
	const char* Watermark;
	const char* Watermark2;
	const char* Mode;
	const char* Inserting;
	const char* Decoding;
	const char* InsertWtm;
	const char* DecodedWtm;
	const char* WtmInfo;
	char Info[4];
	char ReceivedString[10];
	int Wtmblok;
	int CharCounter;

	float* buffer;
	int Wavesize;
	int _index;
	int Wtmlength;
	char* Wtm2Insert;

	// Bitmap
	CBitmap* hBackground;
	CDrawContext* my_context;
};
#endif
```

```
//-------------------------------------------------------------------------
// VST Plug-Ins SDK
// Version 2.4		$Date: 2006/11/13 09:08:28 $
//
```

```
// Category     : VST 2.x SDK Samples
// Filename     : Editor.cpp
// Created by   : Steinberg Media Technologies
// Description  : Simple Surround Delay plugin with Editor using
VSTGUI
//
// © 2006, Steinberg Media Technologies, All Rights Reserved
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
// Date: 2008/05/10
//
// Category     : VST 2.x SDK Samples
// Filename     : my_editor.cpp
// Created by   : David Henzl
// Description  : GUI Editor for AWtm(Echo Hiding) (Mono->Stereo)
//-------------------------------------------------------------------------

#ifndef __my_editor__
#include "my_editor.h"
#endif

#include <stdio.h>
#include <math.h>
#include <vstcontrols.h>

//-------------------------------------------------------------------------
// resource id's
enum
{
      kDelay0Txt=20,
      kDelay1Txt,
      kAlfaTxt,
      kVSTblokTxt,
      kWatermarkTxt,
      kWatermark2Txt,
      kBlocksizeinfoTxt, //Alternativní velikost bloku
      kModeTxt,
      kText,

};
enum {
      // bitmaps
      kBackgroundId = 128,
      kPowerBodyId,
      kFaderHandleId,
      kMaster,
      kPowerPixmapId,
      kButtonPixmapId,
      kBigKnobId,
      kWaveWinID,
      kMinMax,

      // positions

      kModeX = 20, //Tablex. 34.8pix)
      kModeY = 100, //(Tabley. 85.8pix)
      kDelay0X =33,
      kDelay0Y =100,

      kInfoTxtX = 350,
      kInfoTxtY = 80,
```

```
	kDisplayX = 157,
	kDisplayY = 30,
	kDisplayXWidth = 30,
	kDisplayHeight = 14,
	kWaveDisplayX = 364,
	kWaveDisplayY = 13,
	kDispParamsX = 157,
	kDispParamsY = 20,
};
enum {
	//button on,off
	kOff=0,
	kOn=1,
	kButt1=10,
	kButt2=11,

};
//-----------------------------------------------------------------------
// prototype string convert float -> percent
void percentStringConvert (float value, char* string);
void percentStringConvert (float value, char* string)
{
	sprintf (string, "%d%%", (int)(100 * value + 0.5f));

}
//-----------------------------------------------------------------------
// Editor class implementation
//-----------------------------------------------------------------------
Editor::Editor (AudioEffect *effect,int pSizeWaveDisp, char* Wtm)
: AEffGUIEditor (effect)
{

	Delay0Display = 0;
	Delay1Display = 0;
	AlfaDisplay = 0;
	Delay0Val = 0;
	Delay1Val = 0;
	AlfaVal = 0;
	WaveDisplay = 0;
	Wtmblok=0;
	CharCounter=0;
	Wtmlength=10;
	Delay0 = "Delay0";
	Delay1 = "Delay1";
	Alfa   = "Alfa";
	VSTblok = "VST Blok";
	Mode = "MODE";
	Inserting = "Inserting";
	Decoding = "Decoding";
	InsertWtm = "Insert Watermark Here";
	DecodedWtm = "Decoded Watermark";

	Watermark2 = "xxxxxxxxxx";
	WtmInfo="Insert Alternative Block Size!";
	strcpy(Info,"xxxx");
	//pBuff = new float[100];
	Wavesize = pSizeWaveDisp;
	_index = 0;
	buffer = new float[Wavesize];
	Wtm2Insert=Wtm;
```

```
	Watermark=Wtm2Insert;
	strcpy(ReceivedString,"x");

	// load the background bitmap
	// we don't need to load all bitmaps, this could be done when
open is called
	hBackground = new CBitmap (kBackgroundId);

	// init the size of the plugin
	rect.left   = 0;
	rect.top    = 0;
	rect.right  = (short)hBackground->getWidth ();
	rect.bottom = (short)hBackground->getHeight ();
}
//-----------------------------------------------------------------------
-----
Editor::~Editor ()
{
	// free the background bitmap
	if (hBackground)
		hBackground->forget ();
	hBackground = 0;
}
//-----------------------------------------------------------------------
-----
bool Editor::open (void *ptr)
{
	// !!! always call this !!!
	AEffGUIEditor::open (ptr);

	//--load some bitmaps
	CBitmap* hPowerPixmap   = new CBitmap (kPowerPixmapId);
	CBitmap* hButtonPixmap   = new CBitmap (kButtonPixmapId);
	CBitmap* hBigKnob   = new CBitmap (kBigKnobId);
	CBitmap* hWaveWindow   = new CBitmap (kWaveWinID);

//--init background frame-----------------------------------
// We use a local CFrame object so that calls to setParameter won't
call into objects which may not exist yet.
// If all GUI objects are created we assign our class member to this
one. See bottom of this method.
	CRect size (0, 0, hBackground->getWidth (), hBackground-
>getHeight ());
	CFrame* lFrame = new CFrame (size, ptr, this);
	lFrame->setBackground (hBackground);

	//--init the faders------------------------------------------

	CPoint point (0, 0);
	CPoint offset (1,0);
	//text
	size (kDispParamsX,kDispParamsY,kDispParamsX + 30,kDispParamsY +
10);
	Delay0Txt = new CTextEdit (size,this,kDelay0Txt,Delay0,0,0);
	Delay0Txt->setFont(kNormalFontVerySmall);
	Delay0Txt->setFontColor(kYellowCColor);
	Delay0Txt->setTxtFace(kNormalFace);
	Delay0Txt->setMouseEnabled(false);
	Delay0Txt->setTransparency(true);
	lFrame->addView (Delay0Txt);
```

```
      size.offset (40, 0);
      Delay1Txt = new CTextEdit (size,this,kDelay1Txt,Delay1,0,0);
      Delay1Txt->setFont(kNormalFontVerySmall);
      Delay1Txt->setFontColor(kYellowCColor);
      Delay1Txt->setTxtFace(kNormalFace);
      Delay1Txt->setTransparency(true);
      Delay1Txt->setMouseEnabled(false);
      lFrame->addView (Delay1Txt);

      size.offset (40, 0);
      AlfaTxt = new CTextEdit (size,this,kAlfaTxt,Alfa,0,0);
      AlfaTxt->setFont(kNormalFontVerySmall);
      AlfaTxt->setFontColor(kYellowCColor);
      AlfaTxt->setTxtFace(kNormalFace);
      AlfaTxt->setMouseEnabled(false);
      AlfaTxt->setTransparency(true);
      lFrame->addView (AlfaTxt);

      size.setWidth(kDisplayXWidth + 8);
      size.offset (40, 0);
      VSTBlokTxt = new CTextEdit (size,this,kVSTblokTxt,VSTblok,0,0);
      VSTBlokTxt->setFont(kNormalFontVerySmall);
      VSTBlokTxt->setFontColor(kYellowCColor);
      VSTBlokTxt->setTxtFace(kNormalFace);
      VSTBlokTxt->setMouseEnabled(false);
      VSTBlokTxt->setTransparency(true);
      lFrame->addView (VSTBlokTxt);

      size (kInfoTxtX, kInfoTxtY,kInfoTxtX + 100, kInfoTxtY + 10);
      InsertWtmTxt = new CTextEdit (size,this,kText,InsertWtm,0,0);
      InsertWtmTxt->setFont(kNormalFontVerySmall);
      InsertWtmTxt->setFontColor(kYellowCColor);
      InsertWtmTxt->setTxtFace(kNormalFace);
      InsertWtmTxt->setTransparency(true);
      InsertWtmTxt->setMouseEnabled(false);
      lFrame->addView (InsertWtmTxt);

      size.offset (0, 10);
      WatermarkTxt = new CTextEdit
(size,this,kWatermarkTxt,Watermark,0,0);
      WatermarkTxt->setFont(kNormalFontVerySmall);
      WatermarkTxt->setFontColor(kYellowCColor);
      WatermarkTxt->setTxtFace(kNormalFace);
      WatermarkTxt->setTransparency(true);
      lFrame->addView (WatermarkTxt);

      size.offset (0, 15);
      DecodedWtmTxt = new CTextEdit (size,this,kText,DecodedWtm,0,0);
      DecodedWtmTxt->setFont(kNormalFontVerySmall);
      DecodedWtmTxt->setFontColor(kYellowCColor);
      DecodedWtmTxt->setTxtFace(kNormalFace);
      DecodedWtmTxt->setMouseEnabled(false);
      DecodedWtmTxt->setTransparency(true);
      lFrame->addView (DecodedWtmTxt);

      size.offset (0, 10);
      Watermark2Txt = new CTextEdit
(size,this,kWatermark2Txt,Watermark2,0,0);
      Watermark2Txt->setFont(kNormalFontVerySmall);
      Watermark2Txt->setFontColor(kYellowCColor);
      Watermark2Txt->setTxtFace(kNormalFace);
```

```
      Watermark2Txt->setTransparency(true);
      lFrame->addView (Watermark2Txt);

      size.setWidth(150);
      size.offset(0,15);
      BlocksizeinfoTxt = new CTextEdit
(size,this,kBlocksizeinfoTxt,WtmInfo,0,0);
      BlocksizeinfoTxt->setFont(kNormalFontSmall);
      BlocksizeinfoTxt->setFontColor(kYellowCColor);
      BlocksizeinfoTxt->setTxtFace(kNormalFace);
      //BlocksizeinfoTxt->setMouseEnabled(false);
      BlocksizeinfoTxt->setTransparency(true);
      lFrame->addView (BlocksizeinfoTxt);

      size(kDispParamsX+17,kDispParamsY+78,kDispParamsX+20 + hBigKnob-
>getWidth(), kDispParamsY+78 + hBigKnob->getHeight()/58);

      Delay0Val = new CAnimKnob(size, this, AEcho_Hiding_Wtm::kDelay0,
57, 35 , hBigKnob, point);
      Delay0Val->setValue(effect-
>getParameter(AEcho_Hiding_Wtm::kDelay0)/512);
      Delay0Val->setTransparency(false);
      lFrame->addView(Delay0Val);

      size.offset(hBigKnob->getWidth()+22,0);
      Delay1Val = new CAnimKnob(size, this, AEcho_Hiding_Wtm::kDelay1,
57, 35 , hBigKnob, point);
      Delay1Val->setValue(effect-
>getParameter(AEcho_Hiding_Wtm::kDelay1)/512);
      Delay1Val->setTransparency(false);
      lFrame->addView(Delay1Val);

      size.offset(hBigKnob->getWidth()+22,0);
      AlfaVal = new CAnimKnob(size, this, AEcho_Hiding_Wtm::kAlfa, 57,
35 , hBigKnob, point);
      AlfaVal->setValue(effect-
>getParameter(AEcho_Hiding_Wtm::kAlfa));
      AlfaVal->setTransparency(false);
      lFrame->addView(AlfaVal);

      //--init the display-----------------------------------------
      // Delay1
      size (kDisplayX, kDisplayY,kDisplayX + kDisplayXWidth, kDisplayY
+ kDisplayHeight);
      Delay0Display = new CParamDisplay (size, 0, kCenterText);
      Delay0Display->setFont (kNormalFontSmall);
      Delay0Display->setFontColor (kWhiteCColor);
      Delay0Display->setBackColor (kBlackCColor);
      Delay0Display->setFrameColor (kBlueCColor);
      Delay0Display->setValue (effect->getParameter
(AEcho_Hiding_Wtm::kDelay0));
      lFrame->addView (Delay0Display);
      // Delay1
      size.offset (40,0);
      Delay1Display = new CParamDisplay (size, 0, kCenterText);
      Delay1Display->setFont (kNormalFontSmall);
      Delay1Display->setFontColor (kWhiteCColor);
      Delay1Display->setBackColor (kBlackCColor);
      Delay1Display->setFrameColor (kBlueCColor);
      Delay1Display->setValue (effect->getParameter
(AEcho_Hiding_Wtm::kDelay1));
```

```
	lFrame->addView (Delay1Display);
	// Alfa
	size.setWidth(kDisplayXWidth + 3);
	size.offset (40,0);
	AlfaDisplay = new CParamDisplay (size, 0, kRightText);
	AlfaDisplay->setFont (kNormalFontSmall);
	AlfaDisplay->setFontColor (kWhiteCColor);
	AlfaDisplay->setBackColor (kBlackCColor);
	AlfaDisplay->setFrameColor (kBlueCColor);
	AlfaDisplay->setValue (effect->getParameter
(AEcho_Hiding_Wtm::kAlfa));
	lFrame->addView (AlfaDisplay);
	// Sizo of VST blok
	size.setWidth(kDisplayXWidth + 3);
	size.offset (40,0);
	VSTBlokDisplay = new CParamDisplay (size, 0, kRightText);
	VSTBlokDisplay->setFont (kNormalFontSmall);
	VSTBlokDisplay->setFontColor (kWhiteCColor);
	VSTBlokDisplay->setBackColor (kBlackCColor);
	VSTBlokDisplay->setFrameColor (kBlueCColor);
	VSTBlokDisplay->setValue (effect->getParameter
(AEcho_Hiding_Wtm::kVSTblok));
	lFrame->addView (VSTBlokDisplay);
	//Mode Area
	size (kModeX, kModeY,kModeX + 30, kModeY + 15);
	ModeTxt = new CTextEdit (size,this,kModeTxt,Mode,0,0);
	ModeTxt->setFont(kNormalFontSmall);
	ModeTxt->setFontColor(kYellowCColor);
	ModeTxt->setTxtFace(kNormalFace);
	ModeTxt->setMouseEnabled(false);
	ModeTxt->setTransparency(true);
	lFrame->addView (ModeTxt);
	//Buttons and labels
	size (kModeX, kModeY+15 ,kModeX + hButtonPixmap->getWidth(),
kModeY + 15 + hButtonPixmap->getHeight()/2);
	button1 = new COnOffButton (size,this,kButt1,hButtonPixmap);
	button1->setValue (1.0f-effect->getParameter
(AEcho_Hiding_Wtm::kMode));
	button1->setTransparency(false);
	lFrame->addView (button1);

	size.offset(15,-2);
	size.setWidth(50);
	size.setHeight(12);
	InsertingTxt = new CTextEdit (size,this,kModeTxt,Inserting,0,0);
	InsertingTxt->setFont(kNormalFontSmall);
	InsertingTxt->setFontColor(kYellowCColor);
	InsertingTxt->setTxtFace(kNormalFace);
	InsertingTxt->setMouseEnabled(false);
	InsertingTxt->setTransparency(true);
	lFrame->addView (InsertingTxt);

	size (kModeX, kModeY+30 ,kModeX + hButtonPixmap->getWidth(),
kModeY + 30 + hButtonPixmap->getHeight()/2);
	button2 = new COnOffButton (size,this,kButt2,hButtonPixmap);
	button2->setValue (effect->getParameter
(AEcho_Hiding_Wtm::kMode));
	button2->setTransparency(false);
	lFrame->addView (button2);

	size.offset(15,-2);
```

```
	size.setWidth(50);
	size.setHeight(12);
	DecodingTxt = new CTextEdit (size,this,kModeTxt,Decoding,0,0);
	DecodingTxt->setFont(kNormalFontSmall);
	DecodingTxt->setFontColor(kYellowCColor);
	DecodingTxt->setTxtFace(kNormalFace);
	DecodingTxt->setMouseEnabled(false);
	DecodingTxt->setTransparency(true);
	lFrame->addView (DecodingTxt);

	//Wavedisp Area for Cepstral Display
size (kWaveDisplayX, kWaveDisplayY,kWaveDisplayX +120,kWaveDisplayY +
60);
	WaveDisplay = new CWaveform
(size,this,AEcho_Hiding_Wtm::kWavDisp,hWaveWindow);
	my_context = new CDrawContext(lFrame,NULL,systemWindow);
	WaveDisplay->setValue (buffer,Wavesize);
	WaveDisplay->setMin(0.0f);
	WaveDisplay->setWheelInc(10.0f);
	WaveDisplay->setMax(1.0f);
	WaveDisplay->setColor (kYellowCColor);
	WaveDisplay->setTransparency(true);
	WaveDisplay->setWantsFocus(true);
	lFrame->addView (WaveDisplay);

	frame = lFrame;
	return true;
}
//-----------------------------------------------------------------------
void Editor::close ()
{
	delete frame;
	frame = 0;

}
//-----------------------------------------------------------------------
void Editor::setParameter (VstInt32 index, float value)
{
	if (frame == 0)
		return;

	switch (index)
	{
	case AEcho_Hiding_Wtm::kDelay0:
		Delay0Val->setValue (effect->getParameter (index)/512);
		if (Delay0Display)
			Delay0Display->setValue (effect->getParameter
(index));
		break;
	case AEcho_Hiding_Wtm::kDelay1:
		Delay1Val->setValue (effect->getParameter (index)/512);
		if (Delay1Display)
			Delay1Display->setValue (effect->getParameter
(index));	break;
	case AEcho_Hiding_Wtm::kAlfa:
		if (AlfaDisplay)
			AlfaVal->setValue (effect->getParameter (index));
		AlfaDisplay->setValue(effect->getParameter (index));
	break;
	case AEcho_Hiding_Wtm::kVSTblok:
```

```
            //sprintf(info,"Block size is %d%",(int)value);
            VSTBlokDisplay->setValue(value);
                  break;

      case AEcho_Hiding_Wtm::kWavDisp:
            if(_index > Wavesize)
            {
                  _index = 0;
                  WaveDisplay->setValue(buffer,Wavesize);

                  WaveDisplay->draw(my_context);
            }
            buffer[_index++] =value;
                        break;

      case AEcho_Hiding_Wtm::kWavDispSize:
            Wavesize=(int)value;
            memset (buffer, 0, Wavesize * sizeof (float));
            WaveDisplay->setValue(buffer,(int)value);

            _index = 0;
                              break;
      case AEcho_Hiding_Wtm::kCharCounter:
            ReceivedString[CharCounter]=(int)value;
            Watermark2Txt->setText(ReceivedString);
            if(++CharCounter > Wtmlength){
                  CharCounter=0;
            }
            break;
      }
}
//------------------------------------------------------------------------
void Editor::valueChanged (CDrawContext* context, CControl* control)
//rewritte display parameters
{
      long tag = control->getTag ();

      switch (tag)
      {
      case AEcho_Hiding_Wtm::kDelay0:
            Delay0Display->setValue (floor(control->getValue ()*512));
            effect-
>setParameterAutomated(AEcho_Hiding_Wtm::kDelay0,(float)(control-
>getValue ())*512);     break;
      case AEcho_Hiding_Wtm::kDelay1:
            Delay1Display->setValue (floor(control->getValue ()*512));
            effect-
>setParameterAutomated(AEcho_Hiding_Wtm::kDelay1,(float)(control-
>getValue ())*512);     break;
      case AEcho_Hiding_Wtm::kAlfa:
            AlfaDisplay->setValue (control->getValue ());

            effect-
>setParameterAutomated(AEcho_Hiding_Wtm::kAlfa,control->getValue ());
                        break;

      case kWatermarkTxt:
            WatermarkTxt->getText(Wtm2Insert);
            effect-
>setParameterAutomated(AEcho_Hiding_Wtm::kWatermark,1.0f);
                              break;
```

```
	case kBlocksizeinfoTxt:
		BlocksizeinfoTxt->getText(Info);
		Wtmblok = 100*((int)Info[0]-48)+10*((int)Info[1]-
48)+((int)Info[2]-48);
		effect-
>setParameterAutomated(AEcho_Hiding_Wtm::kWtmblok,(float)Wtmblok);
		effect-
>setParameterAutomated(AEcho_Hiding_Wtm::kWitchBlok,1.0f);//Změna
velikosti bloku
		effect-
>setParameterAutomated(AEcho_Hiding_Wtm::kWatermark,1.0f);//opakovana
inicializace algoritmu
		//sprintf(Info,"Alternative Block size is
%d%",(int)Wtmblok);
		//BlocksizeinfoTxt->setText(Info);break;

	case kButt1:
		if(button1->getValue() == kOn)
		{
			button2->setValue(kOff);
			effect-
>setParameterAutomated(AEcho_Hiding_Wtm::kMode, 0.0f);
			effect-
>setParameterAutomated(AEcho_Hiding_Wtm::kWatermark,1.0f);//opakovana
inicializace algoritmu
		}else button1->setValue(kOn); break;

	case kButt2:
		if(button2->getValue() == kOn)
		{
			button1->setValue(kOff);
			effect-
>setParameterAutomated(AEcho_Hiding_Wtm::kMode, 1.0f);
			effect-
>setParameterAutomated(AEcho_Hiding_Wtm::kWatermark,1.0f);//opakovana
inicializace algoritmu
		}button2->setValue(kOn);	break;

	}
	control->setDirty ();
}
//------------------------------------------------------------------------
char* Editor::getwatermark ()
{
	return Wtm2Insert;
}
```

B Zdrojový kód MATLAB

```
%Vložení vodoznaku
clc;clear all;close all;
[x,Fs,nbits]=WAVREAD('housle.wav');
N=512;
q=ceil((size(x,1))/N)-1;
matrix=zeros(N,q);
energy=zeros(q,1);
y=zeros(q*N,channels);
a0=0.5;
a1=0.5;
delay0=128;
delay1=256;
L1=10;
L2=502;
L3=N;
n=1;
vyst=1;
aa=[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1];
bits=[1 1 1 0 1 0 1 0 1 1 1 1 0 0 0 1];

for i=1:q;
 for j = 1:N
  matrix(j,i) = x(j+N*(i-1),1);
  energy(i)=energy(i)+ matrix(j,i)^2;
 end
   energy(i)=energy(i)/N;
end
one_mixer=zeros(16,512);
zero_mixer=zeros(16,512);

 for j=1:16;
  for i = 1:N
one_mixer(j,i)=mixer_signal(1/512,i,bits,j,L1,L2,L3);
zero_mixer(j,i)=1-mixer_signal(1/512,i,bits,j,L1,L2,L3);
end
end

  for i=10:size(matrix,2) %sloupce
       a=a1;
       n=aa(n);

       if(bits(n)==1)
       delay=delay1;
       else delay=delay0;
       end

    for j=1:N %radky
    matrix(j,i) = matrix(j,i) + a*x(j+N*(i-1)-
delay0,1)*zero_mixer(n,j)+a*x(j+N*(i-1)-delay1,1)*one_mixer(n,j);
    end

   n=n+1;
end
for i=1:size(matrix,2) %sloupce
 for j=1:size(matrix,1) %radky
 y(j+N*(i-1),1)=matrix(j,i)*vyst;
 end
 end
```

```
wavwrite(y,Fs,'output.wav');
plot(y);
sound(y,Fs);
```

```
%Detekce
clc;clear all;close all;
[x,Fs,nbits]=WAVREAD('output.wav');
delay0=128;
delay1=256;
nbits=16;
N=882;
T=0.5;
Nram = 1+floor((size(x,1)-N/2)/N); %pocet ramcu
q=ceil(size(x,1)/N)-1;
matrix=zeros(N,Nram);
energy=zeros(q,1);
zer=zeros(nbits,1);
one=zeros(nbits,1);
energy2=zeros(q,1);
y=zeros(N+(Nram*N),2);
x0=zeros(1,Nram);
x1=zeros(1,Nram);
ffty = zeros(N,1);
ifftlogy = zeros(1,N);
sync=zeros(1,8*N);
recbits = zeros(1,Nram);
xcorrel0 = zeros(1,N);
n=1;
counter =1;
startblok=0;
stopblok=0;
fs1=1024;
fs2=1/100;
Q0=1;
Q1=1;
fb1=fs1/Q0;
fb2=fs2/Q1;
Fvz=44100;
NN=2048;
c=(tan(pi*fb1/Fvz)-1)/(tan(pi*fb1/Fvz)+1);
d=-cos(2*pi*fs1/Fvz);
Ab=[-c,d*(1-c),1];
Aa=[1,d*(1-c),-c];
[Hb1,Ha1]=parallel([1 0],[1 0],-Ab,Aa);
Hb1=0.5*Hb1;
Ha1=0.5*Ha1;
L1=10;
L2=502;
L3=N;
d=0;%syncindex;
for i=1:(q-1)
 for j = 1:N
  matrix(j,i) = x(j+d+N*(i-1));
  energy(i)=energy(i)+ matrix(j,i)^2;
 end
   energy(i)=energy(i)/N;
   if(energy(i)>1e-5) & (startblok==0)
```

```
        startblok=i+1+1;
    end
    if(energy(i)<1e-5) & (stopblok==0) & (i>q-100)
    stopblok=i-1;
    end
end
a=1;
for i=startblok:stopblok %sloupce
        ffty=log(abs(fft(matrix(:,i),N).^2));
        ifftlogy = real(ifft(ffty));
        xcorrel0 = abs(xcorr2(ifftlogy));
        xcorrel0(N-2:N+2)=0;
        xcorrel0 = abs(xcorr2(ifftlogy));
        a=a+1;
        str=0;
        ee=0;
     x0(counter)=sum(xcorrel1(str+delay0-ee:1:str+delay0+ee));
     x1(counter)=sum(xcorrel1(str+delay1-ee:1:str+delay1+ee));

     if(x0(counter)<x1(counter))

         recbits(1,counter)=1;

     end
     counter = counter +1;
end

for i = 1:floor(stopblok/nbits)
   for j=1:nbits
if(recbits((i-1)*nbits+j)==1)
     if(x1((i-1)*nbits+j)>T)
     one(j) = one(j)+1;
     end
 else
     if(x0((i-1)*nbits+j)>T)
         zer(j) = zer(j)+1;
     end
end
   end
end
watermark = zeros(size(zer,1),1);
for j=1:size(zer,1)
if(one(j)>zer(j))
watermark(j)=1;
end
end
recbits
watermark'
 bits=[1 1 1 0 1 0 1 0 1 1 1 1 0 0 0 1];
%BER%%%%%%%%%%
BER=0;
for i = 1:floor(counter/nbits)
   for j=1:nbits
    if(recbits((i-1)*nbits+j)~=bits(j))
    BER=BER+1;
    end
   end
end
BER=(BER/(floor(q/nbits)*nbits))*100
```