ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra řídící techniky

Bakalářská práce

Digitální syntéza výstražných zvukových signálů

Rok: 2007
Vypracoval: Tomáš Sosna
Vedoucí práce: Ing. Richard Šusta, PhD.

České vysoké učení technické v Praze, fakulta elektrotechnická

Katedra řídicí techniky Školní rok: 2005/2006

Zadání bakalářské práce

Student: Tomáš Sosna

Obor: Kybernetika a měření

Název tématu: Digitální syntéza výstražných zvukových signálů

Zásady pro vypracování:

1. Prostudujte metody zvukové syntézy pro tvorbu výstražných signálů.
2. Navrhnete modul pro zvukovou syntézu určený pro prostředí jazyka C#.NET.
3. Zvolený modul naprogramujte.

Seznam odborné literatury: Dodá vedoucí práce

Vedoucí bakalářské práce: Ing. Richard Šusta, Ph.D.

Datum zadání bakalářské práce: letní semestr 2005/06

Termín odevzdání bakalářské práce: 19. 1. 2007

L.S.

Prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

v zast.

Prof. Ing. Zbyněk Škvor, CSc.
děkan

V Praze, dne 27. 9. 2006

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW, atd.) uvedené v přiloženém seznamu.

Nemám žádný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č.121/2000 sb., o právu autorském, o právech souvisejících s právem autorským a změně některých zákonů (autorský zákon).

V Praze dne: 17.1.2007 Podpis: ……

Abstrakt

Cílem této bakalářské práce je vytvoření programového modulu určeného k digitální syntéze výstražných zvukových signálů v prostředí jazyka C#.NET. Nejprve bude mou snahou přiblížit čtenáři jednotlivé metody zvukové syntézy a funkci syntezátoru. Poté vhodně navrhnout programový modul, který bude umožňovat, jak vlastní návrh algoritmu syntézy, tak bude obsahovat již vytvořené algoritmy vhodné právě pro tvorbu výstražných signálů.

Abstract

The aim of this bachelor's thesis is to create programming module for digital alarm sound synthesis in C# language for .NET platform. At first work describes sound synthesis methods and function of synthesizer and the task is to develop suitable programming module, which will be able to design own algorithm of synthesis and include presets for making alarm sounds.

Obsah

Úvod ... 1
1 Seznámení s problematikou ... 2
1.1 Použité termíny ... 2
1.2 Historie elektronického tvoření zvuků ... 2
2 Základy zvukové syntézy ... 4
2.1 Klasické metody syntézy ... 4
2.1.1 FM syntéza ... 4
2.1.2 Aditivní syntéza ... 5
2.1.3 Filtrační syntéza ... 5
2.1.4 Tvarová syntéza ... 5
2.1.5 Vektorová syntéza ... 5
2.2 Digitální syntéza ... 6
2.2.1 Vzorkování ... 7
2.2.2 Kvantování ... 7
3 Syntezátor a jeho bloky ... 10
3.1 VCO - Napěťově řízený oscilátor ... 10
3.2 VCF - Napěťově řízený filtr ... 13
3.3 VCA - Napěťově řízený zesilovač ... 15
3.4 EG - Generátor obálky ... 15
3.5 LFO – Nízkofrekvenční oscilátor ... 16
4 Popis návrhu aplikace ... 18
4.1 Syntezátor poplašných a varovných signálů ... 18
4.2 Návrh struktury aplikace ... 19
4.3 Grafické uživatelské rozhraní ... 19
4.4 Výběr a popis zvukových ovladačů ... 20
4.5 Práce s komponentou DirectSound v C# ... 21
4.6 Třída Schema ... 22
4.7 Způsob zpracování ... 25
4.8 Využití prostředků jazyka C# ... 27
5 Manuál ... 28
5.1 Hlavní formulář ... 28
5.2 Formulář Zapojeni ... 28
5.3 Formulář Panel ... 29
5.4 Nahrávání ... 30
5.5 Popis konektorů ... 30
5.6 Popis zařízení ... 31
5.6.1 Ovládací prvky ... 31
5.6.2 Zařízení signálové cesty ... 33
5.6.3 Matematické operace ... 34
5.6.4 Oscilátory ... 34
5.6.5 Filtry ... 34
5.6.6 Modulátory ... 35
6 Srovnání ... 37
7 Závěr ... 38

A. Slovník pojmů a zkratek ... 39
B. Literatura ... 40
C. Obsah CD-ROM ... 41

Seznam obrázků

1. **Princip frekvenční modulace** 4
2. **Princip vektorové syntézy** 6
3. **Blokové schéma vzorkovače** 7
4. **Kvantování** 8
5. **Blokové schéma subtraktivního syntezátoru** 10
6. **Ukázka periodického signálu** 11
7. **Základní princip filtru** 13
8. **Funkce řízeného zesilovače signálu** 15
9. **Princip generátoru obálky** 16
10. **Příklad funkčního a ovládacího bloku** 18
11. **Struktura přístupu aplikace k hardwaru** 20
12. **Princip funkce DirectSound** 21
13. **Princip funkce cyklického zásobníku** 22
14. **Shema a jeho vztah k dílčím třídám** 23
15. **Obecné schéma zapojení** 24
16. **Schéma zapojení s barevným vyznačením úrovní** 24
17. **Interpretace zapojení v poli prehravani** 25
18. **Ukázka vstupů a výstupů zesilovače** 26
19. **Hlavní aplikační formulář** 28
20. **Formulář zapojení a hlavní kontextové menu** 28
21. **Smazání zařízení** 29
22. **Formulář panel s ovládacími prvky** 29
23. **Demonstrace způsobu nahrávání** 30

Seznam tabulek

1. **Odstup signál šum pro dané kvantovací kroky** 9
2. **Popis jednotlivý konektorů** 31

Úvod

Tato bakalářská práce si klade za cíl vytvořit modul v prostředí jazyka C#.NET, určený k digitální syntéze výstražných zvukových signálů. Problémem syntézy zvuku na počítači se zabývám již delší dobu, proto mi přišlo jako nejvhodnější zvolit modulární přístup k architektuře syntézy. Z toho vychází i vlastní návrh aplikace, která člení jednotlivá zařízeni do bloků které je možno vzájemně propojovat a vytvářet tak architekturu syntézy.

V první kapitole jsou vysvětleny základní termíny související se syntézou. Další část je věnována historickému vývoji syntézy zvuku a to od středověku až po současnost. Jsou zde představeny stěžejní a přelomové vynálezy z dějin syntézy.

Druhá kapitola popisuje metody syntézy používané jak v digitálních tak analogových syntezátorech. Představeno bude pět nejvýznamnějších metod: aditivní, filtrační, FM, vektorová a tvarová. Poslední podkapitoly jsou věnovány digitální syntéze a teoretickému ohlédnutí za procesy vzorkování a kvantování.

Následující kapitola obsahuje popis syntezátoru a jeho funkčních bloků. Nejprve rozeberu proč je vhodné dělit syntetizér do bloků. V druhé části podrobně rozeberu funkci a princip bloků využívaných subtraktivním syntezátorem.

Čtvrtá kapitola popisuje návrh a vývoj vlastní aplikace. Pozornost je také věnována výběru zvukového ovladače a práci s komponentou DirectSound v C#. Nakonec je rozebrána funkce třídy `Schema` a způsob generování a zpracování signálu z programátorského hlediska. V této kapitole je k popisu kódu a názvů tříd použito stejného stylu jako v prostředí C#.

Poslední kapitola obsahuje manuál, který je vytvořen s důrazem na uživatelský přístup k aplikaci. Představuje jednotlivé bloky, možnosti a způsob ovládání. Nezabývám se kódem ani způsobem naprogramování.

1 Seznámení s problematikou

1.1 Použité termíny

Současný matematický aparát umožňuje dokonale analyzovat existující zvuky. K takovéto analýze jsou zpravidla použity metody na základě Fourierovy transformace. Výsledkem analýzy zvuku je jeho základní kmitočet, jeho veškeré harmonické, obálka amplitudy a obálka harmonických.

Syntézou nazýváme proces, při kterém skládáme zvuky z elementů získaných při analýze. Existuje několik způsobu, jak k syntéze výsledného zvuku přistupovat. Jednotlivé druhy syntéz se navzájem liší, jak z pohledu dokonalosti, tak efektivnosti, která souvisí s náročností na vybavení. Zařízení provádějící syntézu nazýváme syntezátor.

Syntezátor je čistě technická záležitost. Jde o zařízení, které používá jednotlivé stavební prvky k vytváření zvuku na základě zvolené metody. Jeho vstupem jsou řídící signály a výstupem výsledný zvuk. Řídící signály jsou buď zadávány uživatelem přes vstupní rozhraní, nebo jsou generovány sekvencerem.

Sekvencer vytváří lineární posloupnost řídících signálů v závislosti na čase a předem nastavených podmínkách. Z obecného pohledu lze říct, že jde o programovatelné zařízení, které je schopno uchovat a reprodukovat informaci.

1.2 Historie elektronického tvoření zvuků

Jak uvádějí odborné prameny (viz [1]), objevily se první pokusy o využití elektřiny při tvorbě tonů hudebních nástrojů již v osmnáctém století, jejich předmětem byly klavír a cembalo.

Prvním zvukovým zařízením pracujícím na principu syntézy sinusového průběhu byl tzv. dynamophon, sestrojený dr. Thaddeusem Cahillem kolem roku 1900. Nástroj používal pro tvorbu zvuků dvanáct výchozích signálů z generátorů s ozubenými koly. Pomocí odporů byly zajištěny požadované poměry amplitud jednotlivých výchozích signálů tak, aby výsledný průběh byl co nejpodobnější požadovanému. Tento princip byl převzat z běžných kostelních varhan a na tomto principu byly později vybudovány varhany typu Hammond.

Ve dvacátých letech vyvinul berlínský elektrotechnik a hudebník prof. Friderich Trautwein nástroj, který je po něm nazýván trautonium. Výchozími signály pro tvorbu tónu byly nízkofrekvenční průběhy pilového tvaru.

V roce 1934 byly v Chicagu Laurensem Hammondem vynalezeny varhany, pracující na principu Cahillových generátorů. Nástroj dostal jméno po svém vynálezci Hammondovy varhany a dosáhl značné obliby zejména pro svůj charakteristický ušlechtilý zvuk.

Snahy o automatizaci hudby začaly kolem 16. století u akustických nástrojů (zvonohry, flašinety, orchestriony, hrací strojky pianoly a jiné automatofony řízené děrnými štítky, pásy, nebo kolíčkovým bubnem ve funkci „paměti“). V 17. století vzniklo zařízení Arca Musaritmica, které sloužilo k mechanickému komponování čtyřhlasých kompozic za pomoci dřevěných šoupátek.

První předchůdci syntezátorů, se objevily v padesátých letech. V laboratořích Radio Corporation of America byl vyvinut nástroj zvaný Electronic Music Syntesizer, k řízení parametrů obálek, oscilátorů, filtrů se používaly informace zakódované do kombinací otvorů na děrné pásce. Podobnými systémy se zabýval vývojový tým firmy Siemens. Sověti na oplátku uvedly ANS, programovatelný na základě graficko-optického principu. Data pro řízení změn parametrů byla dodávána na filmovém pásu, z nějž byla čtena fotobuňkami.

V šedesátých letech se objevil první klasický analogový syntezátor, jehož konstruktérem byl R.A.Moog. V tomto syntezátoru se poprvé použily obvody VCO, VCF, VCA (viz. kapitola 3) a byla založena norma mezi řídícím napětím a výškou tónu 1V/Okt. Následující vývoj patří převážně vývojovým skupinám firem. EMS, Oberheim, Arp, Korg, Roland atd.

Konec sedmdesátých a osmdesátá léta jsou ve znamení prvních vícehlasých nástrojů s mikroprocesorovým řízením. Dále s rozvojem vyplynula nutnost jednotného komunikačního systému, jehož základy byly položeny roku 1982 kdy se domluvilo 15 japonských a amerických firem na jednotném komunikačním rozhraní MIDI (musical instruments digital interface) více viz [2]. První MIDI norma vyšla 5.8.1983.

Devadesátá léta jsou význačné rozvojem výpočetní techniky a obrovským tempem jejího zpřístupňování široké veřejnosti. S tím je spojen rozvoj digitálního zpracování signálu, objevily se první softwarové produkty, ekvivalentní jejich hardwarovým předlohám. Nevýhodou počítačového zpracování bylo ovládání a rychlost, takže nebylo možné signál zpracovávat v reálném čase. Na poli hardwaru se objevili virtuální syntezátory, u kterých lze přes jedno uživatelské rozhraní přistupovat k několika zdrojům zvuku najednou. Tento nástroj je schopen hrát rozličné zvuky přes sebe.

V dnešní době jsou možnosti výpočetní techniky pro digitální zpracování zvukových signálů téměř neomezené. Zvukový software se používá na profesionální úrovni již zcela běžně, softwarové syntezátory mají lepší editační možnosti. Osobní počítače obsahují dosti velké pevné paměti, které lze využívat pro vícekanálové digitální nahrávání v reálném čase.

2 Základy zvukové syntézy

2.1 Klasické metody syntézy

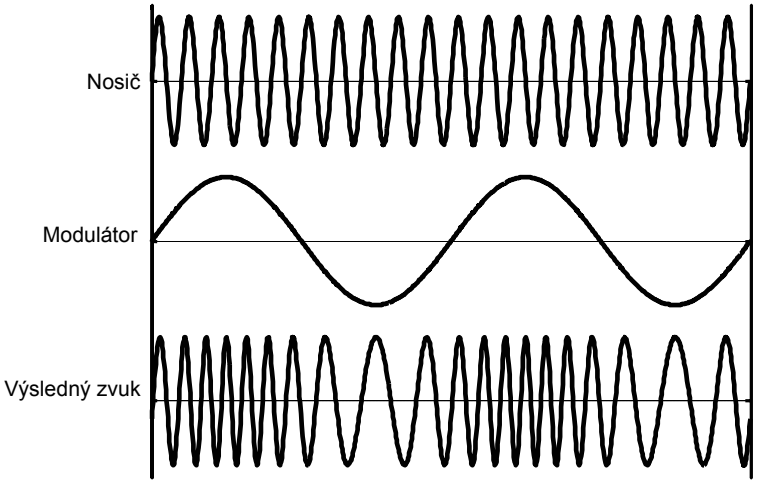
V této kapitole chci čtenáři stručně přiblížit nejpoužívanější metody syntézy. První tři patří mezi nejrozšířenější; další dvě jsou používány pro specifické účely, hlavně v hudebních nástrojích. Pokud má být produktem syntézy opravdu věrný zvuk, přináší použití pouze jedné metody dosti velké komplikace, proto je výhodnější jednotlivé metody vzájemně kombinovat.

Mým cílem je naprogramovat modul, využívající k tvorbě výsledného zvuku první tři metody. Pomocí aditivní syntézy lze generovat zvuky, které v čase nemění svou frekvenci, např. nejrůznější pískoty a bzuky. FM-syntézou lze vytvářet zvuky v čase měnící svou frekvenci, např. zvuky houkaček.

Nyní přejdu k vysvětlení jednotlivých metod.

2.1.1 FM syntéza

FM syntéza využívá k tvoření zvuku frekvenční modulace. Základem je periodický signál, jehož kmitočet řídíme jiným průběhem, většinou sinusovým. První průběh nazýváme nosičem, druhý modulátorem. V okamžicích, kdy se amplituda modulátoru zvyšuje, roste také frekvence výsledného signálu, a naopak. Princip frekvenční modulace je zobrazen na obr.1.



obr.1: Princip frekvenční modulace

Lze matematicky dokázat, že výsledný zvuk obsahuje značné množství harmonických. Možnosti této metody lze rozvinout použitím většího počtu modulátorů, nebo nosičů.

Nosiče a modulátory lze vzájemně propojovat, např. modulátory mohou být modulovány, nebo lze použít zpětnovazební zapojení.

2.1.2 Aditivní syntéza

Aditivní syntéza vytváří výsledný tón z elementárních stavebních prvků, ty reprezentují sinusové zdroje - oscilátory. Výsledný zvuk je v každém okamžiku součtem dílčích amplitud, což znamená, že k vytvoření složitějšího zvuku musíme přistupovat individuálně a je zapotřebí většího počtu oscilátorů, u kterých je se řídí časové změny zabarvení amplitudy. Z toho vyplývá, že tento způsob je těžkopádný, chceme-li vytvořit zvuk bohatý na harmonické, ale použitelný v případě potřeby jednoduchého strohého zvuku.

2.1.3 Filtrační syntéza

Filtrační neboli subtraktivní, syntéza na rozdíl od metody aditivní využívá periodických průběhů, které jsou samy o sobě velmi bohaté na harmonické. Pomocí se ze spektra takovéhoto průběhu oříznou harmonické tak, aby se spektra zamýšleného a vytvářeného zvuku co nejméně lišily. Mezi frekvenčně bohaté průběhy řadíme kmity tvaru obdélníku, nebo pily a trojúhelníku.

2.1.4 Tvarová syntéza

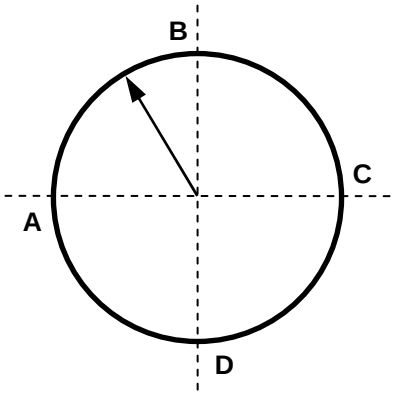
Tvarová syntéza (jinak PD-syntéza) využívá tvarovacího obvodu, který má nelineární převodní charakteristiku. Při průchodu většinou sinusového signálu tvarovačem dojde k jeho amplitudovému zkreslení, které se projeví ve frekvenční oblasti obohacením spektra.

2.1.5 Vektorová syntéza

Vektor je ve zjednodušeném pojetí orientovaná úsečka na ploše, která má definovanou velikost a směr. K přesnému určení pak stačí buď úhel a velikost nebo dvě souřadnice.

Představme si kruh, jehož hraniční kružnici protínají dva navzájem kolmé průměty v bodech A,B,C,D. Body A,B,C,D jsou zdroje signálů, které chceme spolu mixovat. Každý bod kruhu je dán určitým vektorem, odpovídá zcela jednoznačnému způsobu smíchání těchto čtyřech signálů. Vektor má počátek ve středu kruhové plochy, jak ukazuje obr. 2.

Výhodou této metody je snadné ovládání poměru amplitud čtyř zvukových zdrojů. Polohu vektoru mixu lze např. ovládat XY-kontrolorem (joystickem).



obr. 2: Princip vektorové syntézy

2.2 Digitální syntéza

Digitální syntéza zpracovává a tvoří zvuky výhradně digitální cestou. Význam této metody vzrostl v době, kdy začalo být digitální zpracování signálu stejně drahé jako analogové. Příchod této metody sebou přinesl značnou řadu výhod, jako např. tvoření zvuku z již v paměti uloženého signálu, jakéhokoliv průběhu a délky, ale i své překážky. Největší je rychlost ta v systému Windows souvisí se zvukovým ovladačem a s rychlostí zpracování událostí od periférií. Obecně je rychlost zpracování uživatelské interakce tak 100x pomalejší než rychlost zpracovávaní signálu. Takže např. pokud uživatel pohne rychle ovládacím prvkem změní se hodnota na jeho výstupu skokově. K odstranění tohoto vlivu bývá zpravidla použita interpolace. Další podstatnou roli hrají úrovně vzorkování a kvantování vlastního audio signálu.

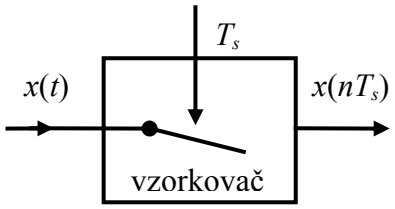
2.2.1 Vzorkování

Vzorkování je proces, ve kterém je ze spojitého signálu $x(t)$ transformací vytvořena posloupnost diskrétních vzorků $x(nT_s)$ angl. *sampling*. Vzorkovaní se týká pouze časové osy, jde vlastně o přidělení určitého intervalu v oblasti spojitého času jedné hodnotě času diskrétního.

T_s - Perioda vzorkování.

$f_s = \frac{1}{T_s}$ - Vzorkovací frekvence.

Problém jakou frekvenci f_s zvolit řeší Shanonova věta.



obr. 3: Blokové schéma

Vzorkovací (Shanonova) věta: Pokud je signál $x(t)$ spojitý v čase a obsahuje pouze složky s frekvencí menší než $f_{\max}$ (signál je frekvenčně omezený), pak může být rekonstruován z posloupnosti diskrétních vzorků $x(nT)$. Pro vzorkovací frekvence f_s=1/T platí:

$$f_s \geq 2 f_{\max}, \tag{1}$$

$\frac{f_s}{2}$ - Nyquistova frekvence.

Frekvenční spektrum vzorkovaného signálu se periodicky opakuje s frekvencí f_s kdy při nesplnění (1) dojde k překrytí opakujících se částí. U spektra v intervalu <0 ÷ f_s/2> dojde k deformaci a není možné z něj získat původní signál. Tato chyba se nazývá anglicky *aliasing*.

Pro zvukové spektrum lze říci, že čím vyšší vzorkovací frekvence, tím vyšší bývá kvalita získané diskrétní podoby daného signálu, ale tím vyšší je i potřebný datový tok. U systémů, které zpracovávají vstupní signál nemá smysl nastavovat vyšší vzorkovací frekvenci výstupu, nežli je vzorkování vstupu. Pro zachycení mluveného slova stačí vzorkování 8-16 kHz, pro CD kvalitu se užívá 44.1 kHz. Ve své práci jsem se rozhodl vzorkovat na 44.1kHz, protože v dnešní době je to standardní vzorkovací frekvence používaná v softwarových syntezátorech na běžné úrovni.

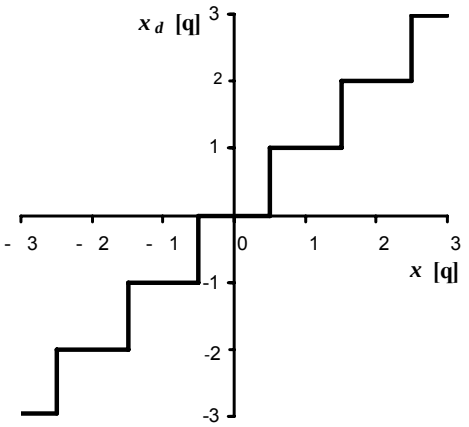
2.2.2 Kvantování

Při diskrétním zpracování zvukového signálu dochází nejprve ke kmitočtovým omezením při vzorkování. Poté je signál kvantován, kde dochází k přimíchávání chyby dané zaokrouhlo-

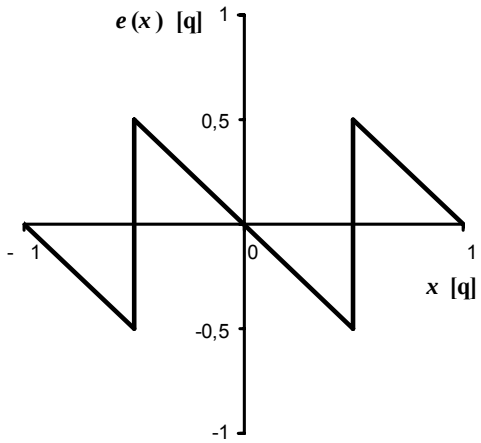
váním. Tato chyba e[x(t)] tvoří rozdíl mezi vstupní analogovou úrovní x(t) a kvantovanou úrovní signálu na výstupu $x_d(t)$. Signál na výstupu kvantizačního procesu můžeme psát ve tvaru

$$x_d(t) = x(t) + e[x(t)]\,. \tag{2}$$

Závislost úrovně kvantovaného signálu na vstupním signálu je znázorněna na obr. 4a. Průběh kvantizační chyby znázorňuje obr. 4b. Jednotka 1q je velikost jednoho kvantovacího kroku.



obr. 4a: Převodní charakteristika



obr. 4b: Závislost kvantizační chyby

Počet bitů převodníku udává jeho rozlišovací schopnost, tj. dynamiku, takže u běžných signálů, které mají dostatečně velkou amplitudu oproti kvantizačnímu šumu, lze předpokládat rovnoměrné rozdělení kvantizační chyby a její statistickou nezávislost oproti chybě v následujícím vzorkovacím kroku. Odstup signálu od kvantizačního šumu je dán vztahem

$$\frac{S}{N} = 20\log\frac{x_{ef}}{\xi_{ef}} \quad [dB], \tag{3}$$

kde x_{ef} je efektivní hodnota stochastického kvantovaného signálu

$$x_{ef} = \frac{2^{n-1}q}{2}\,, \tag{4}$$

a ξ_{ef} je efektivní hodnota kvantizačního šumu

$$\xi_{ef} = \frac{q}{\sqrt{12}}\,, \tag{5}$$

proměnná q určuje velikost kvantovacího kroku a n určuje počet bytů.

Dosazením vztahů (5) a (4) do vztahu (3) dostaneme vyjádření vztahu poměr signál šum (6) pro proměnou *n*

$$\frac{S}{N} = 20\log\left(\frac{2^{n-1}q}{2}\frac{\sqrt{12}}{q}\right) = 6,02n - 1,25 \quad [dB]. \tag{6}$$

Výraz udává teoretický odstup maximální možné úrovně, kterou je schopen převodník zpracovat, od kvantizačního šumu. Tabulka 1 ukazuje teoretickou závislost odstupu signál/šum pro daný počet bitů převodníku. O vzorkování a kvantování signálů pojednává na hlubší úrovni např. literatura [3], nebo [4].

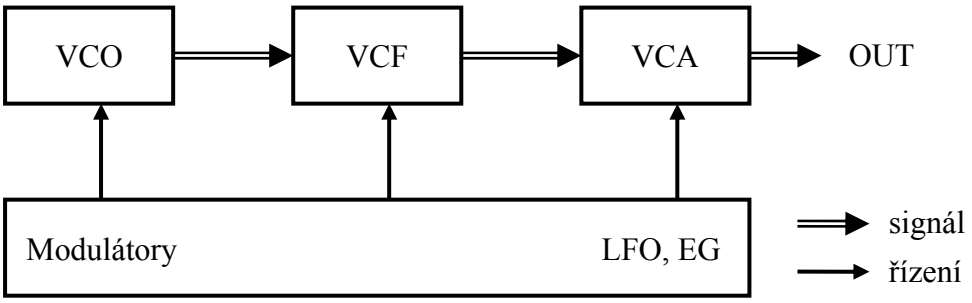
Počet bitů	**Kvantovací krok**	**Teoretický odstup**
8	1 / 256	47 dB
12	1 / 4096	71 dB
16	1 / 65536	95 dB
24	1 / 16777216	143 dB

tabulka 1: Odstup signál šum pro dané kvantovací kroky

3 Syntezátor a jeho bloky

Koncepce syntezátoru do bloků vychází z jeho modulární podoby, ta se skládala z různých funkčních bloků, které lze libovolně propojovat a utvářet tak architekturu syntézy. Při vývoji nového druhu syntezátoru byl postup následující. Nejprve byl utvořen určitý způsob zapojení, který byl následně fixován a přenesen na plošný spoj. Jednotlivé bloky samozřejmě vycházely z modulárního syntetizéru, akorát měly pevné propojení a počet. Toto řešení sice omezilo možnosti tvorby zvuku, ale na druhou stranu podstatně zjednodušilo a urychlilo jeho programování.

Subtraktivní syntéza jako první umožnila programovat zvuky na syntezátorech i běžným uživatelům a stále patří mezi nejpoužívanější. Na následující obrázek ukazuje obecné blokové zapojení subtraktivního syntezátoru.



obr. 5: Blokové schéma subtraktivního syntezátoru

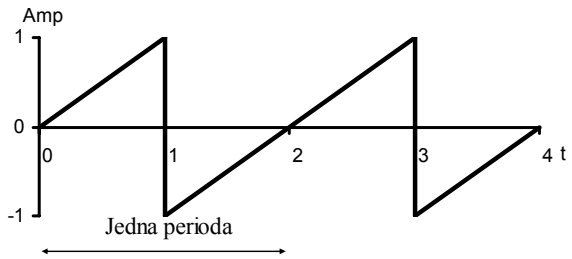
3.1 VCO - Napěťově řízený oscilátor

VCO, z anglického *voltage controlled oscilator* je zdroj periodického signálu, u kterého je možné měnit dvě základní vlastnosti, frekvence (*frequenci, pitch*) a průběh (*waveform*).

Frekvence oscilátoru

Oscilátor generuje cyklický průběh, který má určitou délku periody viz. obr.6. Tato délka závisí na výšce tónu oscilátoru (*pitch*); čím je tato perioda kratší (vyšší frekvence opakování), tím je produkovaný tón vyšší a naopak obr.6. Výšku tónu tedy udává frekvence oscilátoru. Řízení kmitočtu lze provádět dvěmi způsoby, a to buď ze vstupního zařízení (kláves /

klávesnice), kde jednotlivé klávesy reprezentují různé hodnoty frekvence. Po stlačení klávesy dojde k přímému řízení oscilátoru. Dále modulací, která umožňuje měnit frekvenci oscilátoru automaticky, nejběžnějším modulátorem je LFO.



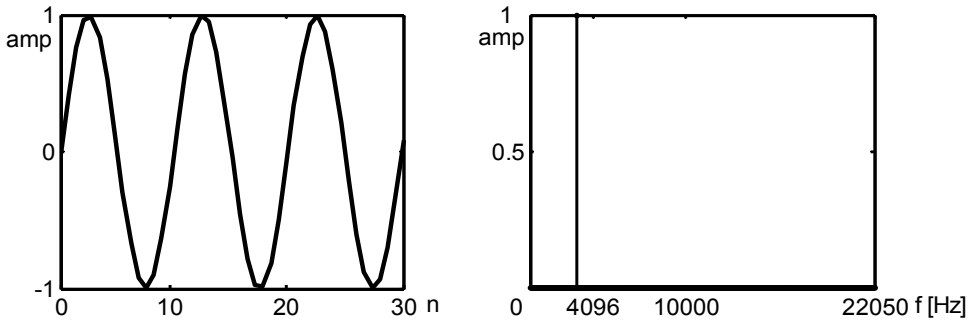
obr. 6: Ukázka periodického signálu

Systémy řízení závislosti výšky tónu na velikosti řídícího signálu jsou dva. První je lineární, ten budu ve své práci používat, vychází z klasické definice linearity, což je pro generování houkaček a sirén ideální. Druhý způsob je nelineární, logaritmický (pitch), ten se používá v elektronických hudebních nástrojích, kde je k popsání tonů použito not, které mají vzhledem k frekvenci logaritmickou závislost.

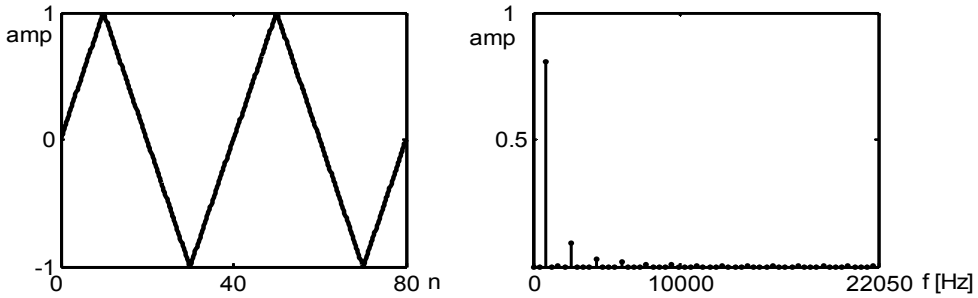
Vlna oscilátoru (waveform)

Vlna (průběh) oscilátoru má přímý vliv na harmonický obsah a barvu zvuku. Každý průběh je složen z určitého počtu harmonických vln. Každá harmonická obsahuje vlnu sinus jde o nejčistší a nejjednodušší průběh, jaký v přírodě existuje. Pokud složíme více těchto vln o určité frekvenci a amplitudě, můžeme vytvořit jakýkoliv průběh. Klasický oscilátor syntetizéru obsahuje tyto nejběžnější tvary: sinus (*sine*), pila (*sawtooth*), obdélník (*square*), trojúhelník (*triangle*) u kterých nyní připomenu základní vlastnosti v amplitudové a frekvenční oblasti.

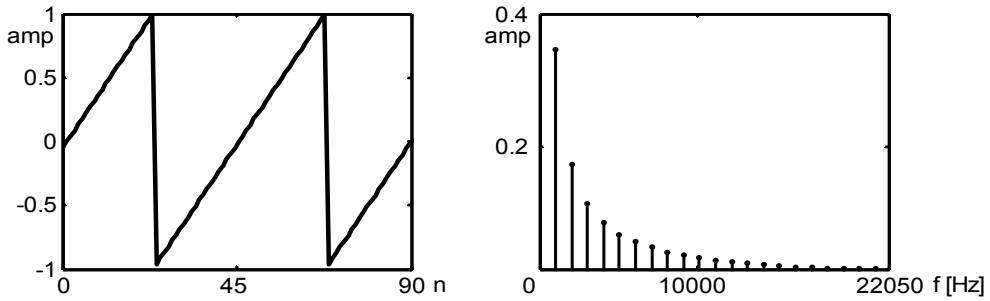
Sinus - nejčistší harmonický průběh, jeho spektrum obsahuje pouze jednu čáru.



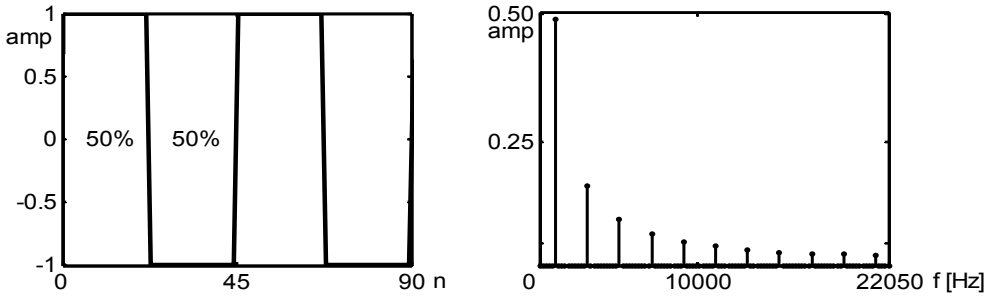
Trojúhelník - spektrum obsahuje liché harmonické, amplituda značně klesá s frekvencí.



Pila - její spektrum obsahuje všechny harmonické, jejich amplituda klesá s frekvencí.



Obdélník - obsahuje liché harmonické, jejich amplituda klesá s frekvencí.

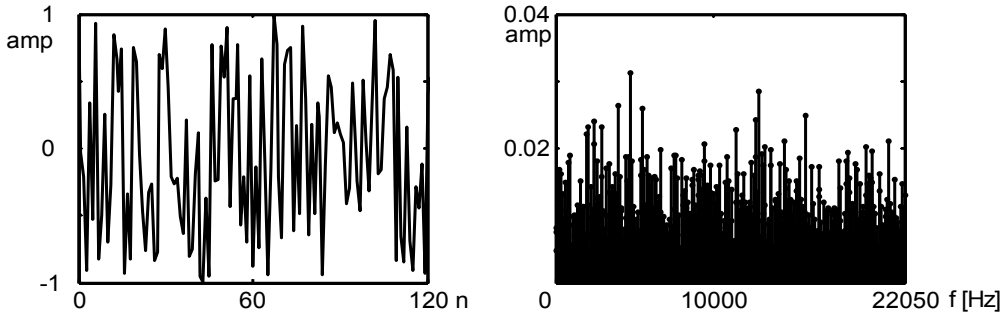


U obdélníkového průběhu lze použít parametr PW (*pulse width*) - změna poměru obdélníku. Tato změna ovlivňuje průběh a obsah harmonických. Parametr PW je většinou modulovatelný, např. pomocí LFO (PWM - *pulse width modulation*).

Neharmonické spektrum

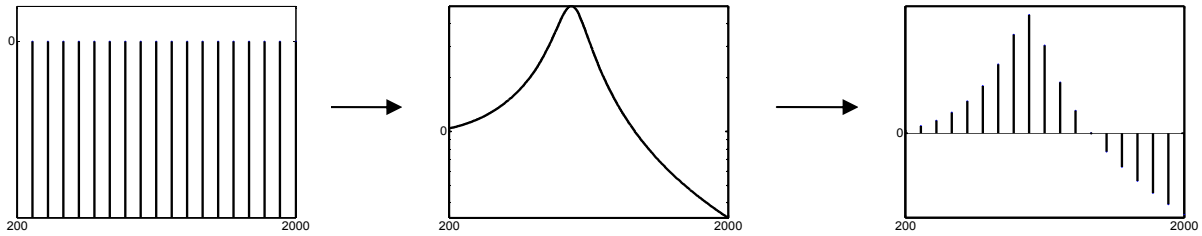
Některé oscilátory (nazýváme je spíše generátory) produkují i neharmonické spektrum. Typickým příkladem je šum (*noise*). Spektrum šumu je závislé na druhu šumu a počtu nevzorkovaných period. Obrázek ukazuje spektrum bílého šumu. Spektrum na obrázku není

vyrovnané z důvodu nízkého počtu vzorků vstupujících do transformace. Při použití tak 1000 vzorků už bude spektrum skoro konstantní.



3.2 VCF - Napěťově řízený filtr

VCF, anglicky *voltage controlled filter.* Je nejdůležitějším barvotvorným obvodem syntezátoru. Slouží ke korekci harmonických složek spektra, protože v původním (většinou z VCO) průběhu mohou být i takové harmonické, které ve výsledném průběhu nepotřebujeme, filtrem můžeme dosáhnout jejich potlačení. Nebo naopak, je-li amplituda pro nás potřebných harmonických malá oproti ostatním, filtr sníží amplitudu ostatních, čímž vzroste jejich relativní amplituda. Schematicky je princip filtru naznačen na následujícím obrázku.



vstup filtr výsledný signál

obr. 7: Základní princip filtru

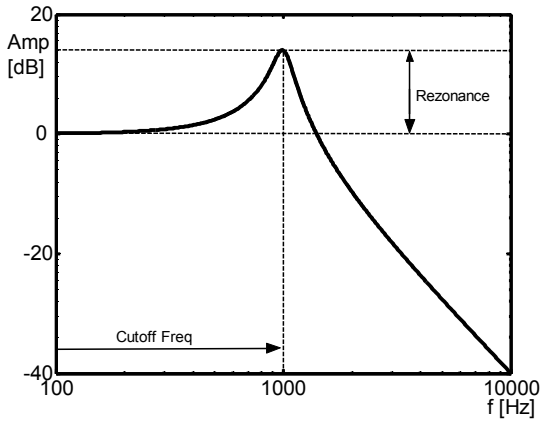
Parametry:

Cutoff frequency - zlomová frekvence, frekvence, od které se pásmo ořezává nebo propouští; kmitočty, které jsou vyšší než tato frekvence, jsou postupně utlumovány v závislosti na řádu filtru, kdežto kmitočty pod zlomovou frekvencí jsou přenášeny beze změny.

Resonance (jinak Q, Emphasis) - zvýraznění zlomové frekvence (některé filtry samooscilují při maximálním nastavení rezonance). Chceme-li, aby se filtr dostal do rezonance zavádíme zpětnou vazbu. Velikost rezonance je pak řízena podle úrovně zpětnovazebního signálu.

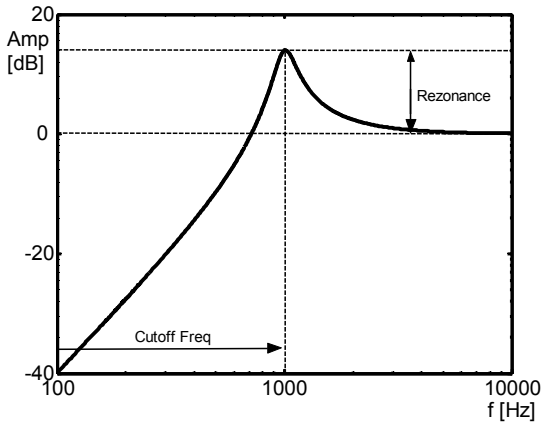
Druhy filtrů:

Nejběžnější typy digitálních filtrů pro zpracování zvuku jsou odvozeny od jejich analogových prototypů, jejichž amplitudové frekvenční charakteristiky ukazují následující čtyři obrázky. A jsou to dolní propust (*lowpass*), horní propust (*highpass*), pásmová propust (*bandpass*), a pásmová zádrž (*notch, bandstop*). U každého filtru se uvádí jeho spojitá přenosová funkce. V mé práci budou všechny výše zmíněné filtry realizovány, u každého půjde měnit parametr *Cutoff* a *Resonance*.



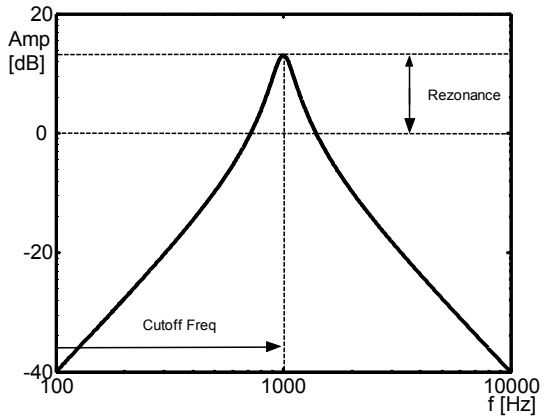
Dolní propust:

$$H(s) = \frac{1}{s^2 + \omega \cdot q \cdot s + \omega^2}$$



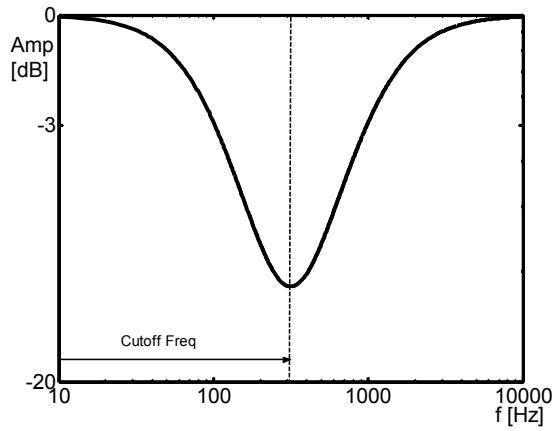
Horní propust:

$$H(s) = \frac{s^2}{s^2 + \omega \cdot q \cdot s + \omega^2}$$



Pásmová propust:

$$H(s) = \frac{s}{s^2 + \omega \cdot q \cdot s + \omega^2}$$



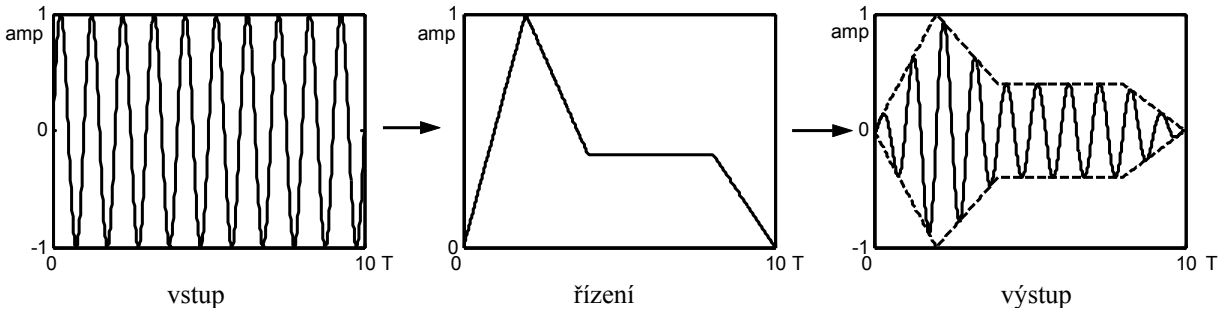
Pásmová zádrž:

$$H(s) = \frac{s^2 + \omega^2}{s^2 + \omega \cdot q \cdot s + \omega^2}$$

Průběh filtru má také určitou strmost, kterou udáváme v decibelech na oktávu. Nejběžnější strmosti jsou 6/12/18/24 dB/okt. (to odpovídá podle řádu filtru 20/40/60/80dB na dekádu).

3.3 VCA - Napěťově řízený zesilovač

VCA, *voltage controlled amplifier*, je klasický zesilovač, který ovládá výslednou hlasitost signálu v závislosti na velikosti řídícího vstupu. Většina syntezátorů používá zesilovače lineární. Řízení zesílení provádíme buď ručně, čímž pevně nastavíme výstupní úroveň, nebo z modulátorů (LFO, EG). Pokud úroveň zesilovače modulujeme, např. generátorem obálky (EG), můžete změnit tvar amplitudy v závislosti na čase, např. krátký razantní, nebo tichý s dlouhým náběhem. Typický výstupní průběh VCA řízeného generátorem obálky je znázorněn na obrázku 8.



obr. 8: Funkce řízeného zesilovače signálu

3.4 EG - Generátor obálky

EG zkratka znamená generátor obálky, anglicky *envelope generator*. Ta se používá pro řízení parametrů jako je frekvence, amplituda, zlomová frekvence filtru (*cutoff*) a dalších. Nastavený průběh spouštíme stiskem klávesy (přesněji s náběžnou hranou řídícího signálu) a proběhne pouze jednou. Pro zjednodušení editace průběhu se obálka skládá ze segmentů, nejběžnějším typem je **ADSR**, který má čtyři segmenty - **attack** (náběh), **decay** (pokles), sustain (držení) a release (doznění). Každý segment může mít dva parametry - **time** (čas) a **level** (úroveň), většinou se používá pouze jeden (nejběžnější přiřazení - attack time, decay time, sustain level, release time). Obalové křivky jsou dvojího druhu nelineární (log, exp) a lineární. V mé práci budu používat nejběžnější lineární ADSR obálku.

A nyní popíšu funkci obálky, pro bližší přiblížení po časových segmentech, které jsou vyznačeny na obr. 9. Se sestupnou hranou řídícího signálu je obálka spuštěna (trigrována). V aplikaci je použit tzv. Level Trigger to znamená spouštění hladinou. Zařízení (EG) je spuštěno po přechodu vstupního trigger signálu přes spouštěcí úroveň (v aplikaci 0) směrem

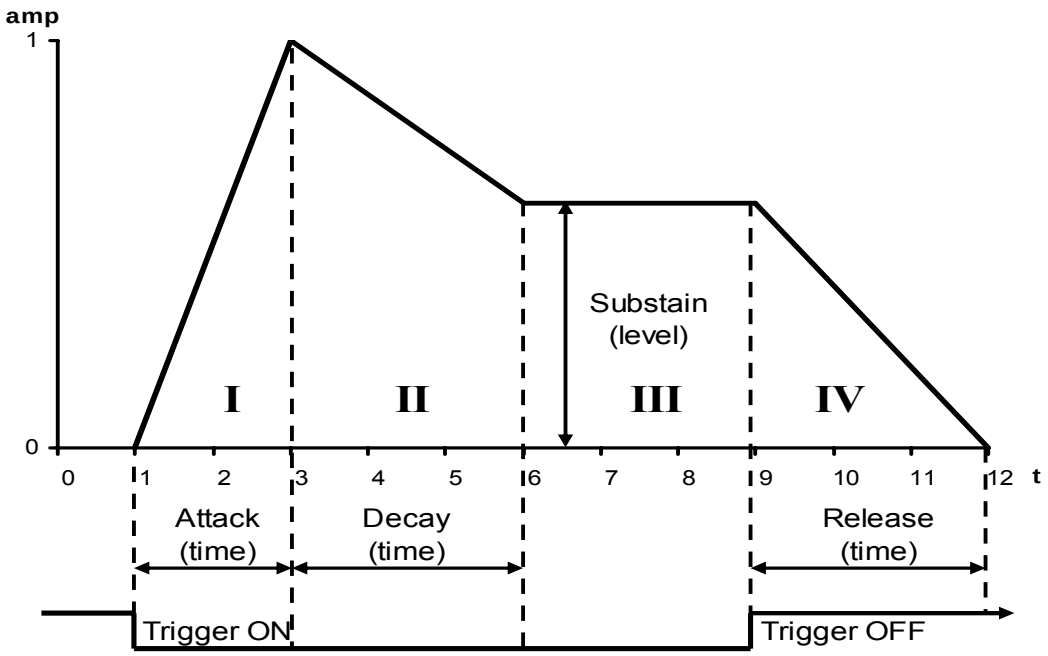
nahoru a vypnuto přechodem dolu. U dvoustavových signálu odpovídá náběžné hraně spuštění a sestupné vypnutí.

I. Průběh obálky startuje z nulové hodnoty do maximální. Čas dosažení maximální hodnoty je závislý na nastavení **Attack**. Pokud je hodnota náběhu nulová, tak obálka startuje téměř okamžitě z maximální hodnoty a naopak.

II. Až obálka dosáhne maximální hodnoty, začne její úroveň klesat. Rychlost (čas) poklesu závisí na nastavení parametru **Decay**.

III. Parametr **Sustain** určuje, na jakou úroveň obálka klesne. Obálka zůstává na této hodnotě až do uvolnění stisku klávesy.

IV. Po uvolnění stisku klávesy se startuje segment **Release** - doznění. Je to pokles z hodnoty **sustain** na nulovou hodnotu, nastavuje se jeho čas.



obr. 9: Princip generátoru obálky

3.5 LFO – Nízkofrekvenční oscilátor

LFO, anglicky *low frequency oscillator*, pracuje na podobném principu jako klasický oscilátor, ovšem s těmito hlavními rozdíly:

- LFO se nepoužívá jako zdroj zvuku, ale jako řídící zdroj pro modulace parametrů.
- Frekvence LFO je velmi nízká (0,01Hz až 20Hz), pro naše uši neslyšitelná.

Jak již bylo řečeno, LFO se používá jako modulační zdroj. Produkuje cyklické hodnoty, které řídí hodnotu zadaného parametru. Pokud např. aplikujete LFO na výšku tónu (pitch) dostanete vibráto.

Parametry LFO:

- Vlna (*waveform*) - Tvar průběhu oscilátoru.
- Frekvence (*rate*) - Rychlost opakování.
- Hloubka (*amount*) - Úroveň modulace daného parametru.

Frekvenci LFO lze většinou také synchronizovat - buď pomocí druhého LFO nebo pomocí MIDI hodin (ve standartu MIDI je pro hodiny vyhrazen jeden vodič a data jsou po něm přenášena v reálném čase).

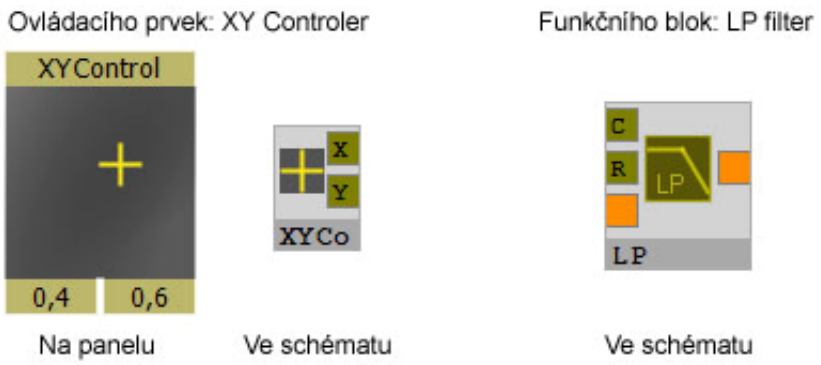
4 Popis návrhu aplikace

Dva hlavní aspekty návrhu aplikace, tedy její vývoj a srozumitelný popis, jsou problémy stejně podstatné. Dobře napsaná funkční aplikace vyžaduje též vysvětlení všech stěžejních principů užívání. Přistupme nyní k podrobnému popisu.

Začínáme zvolením vhodného přístupu k vlastnímu návrhu, zamyšlením nad počtem a potřebou vláken. Následovat bude výběr vhodného zvukového rozhraní a popis principu kontinuálního doplňování vzorků do zásobníku ovladače. Poté bude pozornost zaměřena na graficky uživatelské rozhraní tzv. GUI a způsob ovládání. A nakonec přijde vysvětlení funkce třídy, která v sobě nese ty nejcennější informace, třídy `Schema`.

4.1 Syntezátor poplašných a varovných signálů

Syntezátor poplašných a varovných signálů dále *popsynth* je program, jenž dovede na základě zadaného schématu vytvořit algoritmus syntézy a podle něj vygeneruje výstupní zvukový signál. Schéma lze navrhovat přímo v programu a to z bloků popsaných v dokumentaci k aplikaci (poslední část této práce). Obecně jsou navrženy dva druhy bloků. První jsou funkční bloky, ty obsahují vstupy a výstupy, s tím že výstupy jsou závislé na vstupech. Druhý druh zastupují ovládací prvky, přes které může uživatel zasahovat do průběhu syntézy. Na následujícím obrázku 10 je příklad výše zmiňovaných bloků.



obr. 10: Příklad funkčního a ovládacího bloku

Za zmínku ještě stojí existence bloku, který zastupuje výstup na zvukovou kartu, v programu značen jako OUTPUT. Protože zpracování signálu je jednokanálové (mono) může existovat pouze jeden.

4.2 Návrh struktury aplikace

Nejprve rozeberu základní strukturu aplikace. Jak již bylo výše zmíněno, návrh bude směřovat k možnosti přímo zasahovat do procesu syntézy při jejím běhu s tím, že uživatelský zásah na vstupu bude na výstupu patrný za co nejkratší dobu. Z toho vyplývá nutnost navrhnout aplikaci asynchronně (proces bude obsahovat více než jedno vlákno). V tomto případě postačí 2 vlákna. První je hlavní aplikační vlákno, a to druhé zabezpečuje doplňování vzorků do zásobníku (angl. *buffer)* na zvukové kartě.

4.3 Grafické uživatelské rozhraní

GUI vychází z představy, že aplikace dává uživateli možnost zasahovat do běhu a formy zvuku ve třech vrstvách (úrovních). K realizaci jednotlivých úrovní lze s výhodou využít formulářů (oken). Následuje stručný popis možností a funkcí jednotlivých vrstev:

[1] Nejvyšší vrstvou je hlavní aplikační formulář. Obsahuje dvě menu, jedno hlavní s obvyklým obsahem a funkcemi, které obsahuje položky pro práci se souborem, ve smyslu celého zapojení, dále základní nastavení ovladače a nápovědu. Druhé menu slouží, jak k zobrazování podřízených formulářů (zapojení a panel), tak pro ovládání běhu syntézy na úrovni: **Rec**/**Play**/**Pause**/**Stop**.

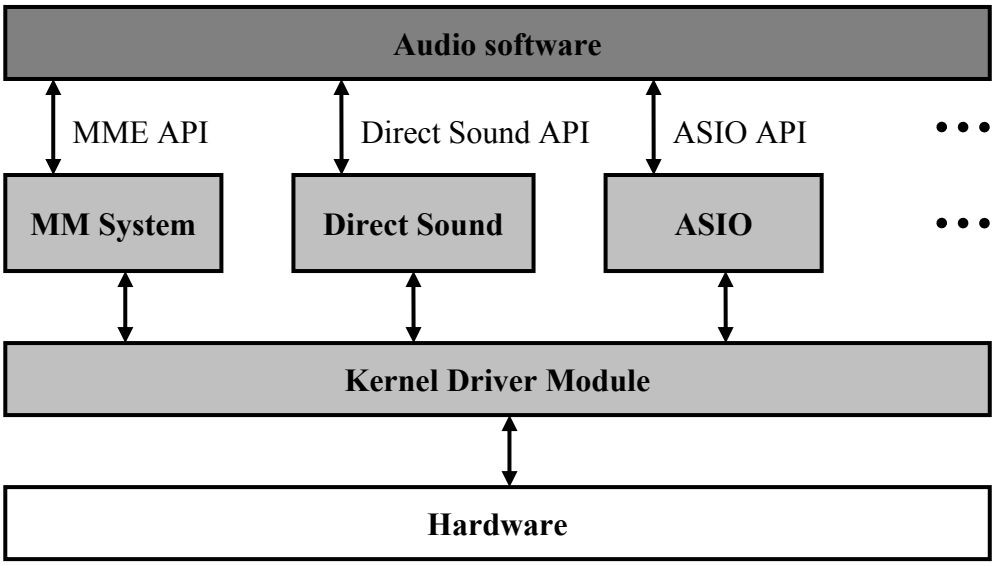
[2] Střední vrstva definována ve třídě `Form_Panel` umožňuje kontrolu nad parametry jednotlivých zařízení pomocí zobrazovaných ovládacích prvků jako Fader, XYcontroller, Button. Tyto prvky inicializuje jejich vytvoření ve formuláři zapojení (viz kapitola 5.3). Lze s nimi pohybovat, měnit hodnotu kterou nesou, podle stavu zámku v menu formuláře. U těchto prvků lze měnit rozsah, protože zařízení mají různé rozsahy řídících vstupů.

[3] Nejnižší vrstva je reprezentována třídou `Form_Zapojeni`, která poskytuje možnost tvořit schéma syntézy, nebo upravovat schémata již vytvořená. Další zařízení lze přidávat přes kontextové menu vyvolávané po pravém kliku na pracovní ploše formuláře. Informace o jednotlivých zařízeních jsou zobrazována přes tzv. tooltip. Jde o popisek který vyskočí pokud se kurzor myši vznáší nad daným objektem (asi 1.5s).

Přechod mezi grafickou podobou zapojení a jeho reprodukcí zajišťuje třída `Schema`.

4.4 Výběr a popis zvukových ovladačů

Následující obrázek ukazuje základní strukturu přístupu ke zvukové kartě v systému Windows. Oblast vyznačenou světle šedou barvou nazýváme *driver* (ovladač). To znamená, že ovladač je zprostředkovatel komunikace mezi softwarem a zvukovou kartou. Obrázek 11 znázorňuje přístup aplikace systému Windows ke zvukové kartě.



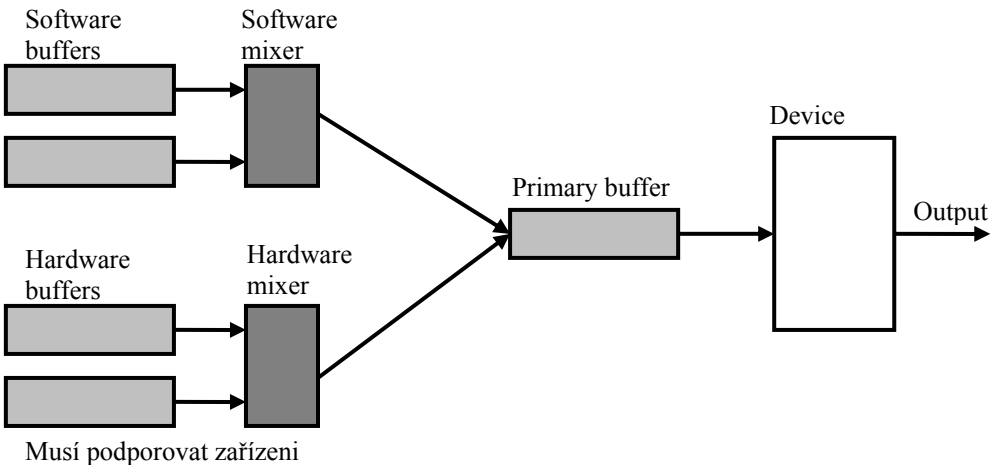
obr. 11: Struktura přístupu aplikace k hardwaru

Jak je vidět základní ovladače jsou 3. Z toho pouze *MM System* je součástí každých Windows, následuje popis jednotlivých zvukových API a výběr toho nejvhodnějšího.

MME Znamená *MultiMedia Extension*, jde o API s možností přístupu ke zvukové kartě, které je zabudované přímo do Windows od dob neoperační verze 3.0 a v malých modifikacích je v nich obsaženo až do dneška. Jde o pomalejší variantu, protože je pod přímou kontrolou systému

DirectSound Přišel s Windows 95, je komponentou rozhraní DirectX, má schopnost přistupovat k zařízení rychleji než MME, tzn. že má nižší latenci (čas mezi zapsáním vzorku do softwarového bufferu a jeho přítomností na výstupu zařízení). Princip funkce DirectSound je obrázku č.12. Jako všechny DirectX komponenty i DirectSound zkouší jak moc dané zařízení podporuje hardwarovou akceleraci (přehrávání několika bufferů najednou, mixování)

pokud akceleraci nepodporuje emuluje její průběh softwarově. Schématicky je funkce naznačena funkce DirectSound takto:



obr. 12: Princip funkce DirectSound

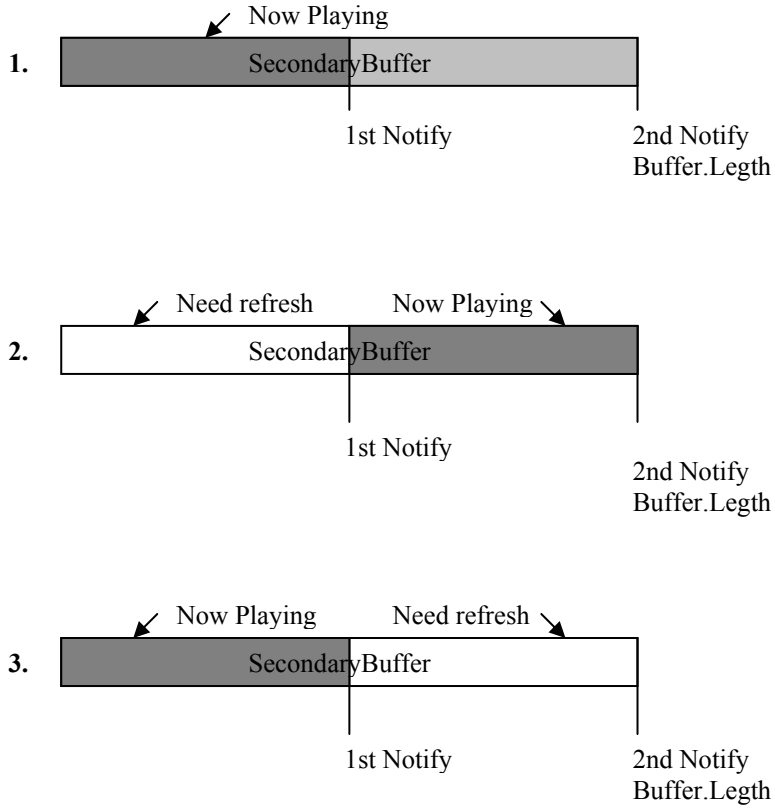
ASIO znamená zkráceně *Audio Streaming Input/Output* je vyvíjen předním výrobcem audio softwaru firmou Steinberg a je určen pro minimalizaci latence a spolupráci s různými druhy bufferů. Většina profesionálních zvukových aplikací pracuje nejlépe právě s tímto ovladačem.

Navrhovaná aplikace bude spolupracovat s ovladači DirectSound, z důvodu jejich dostupnosti a rychlosti, při přehrávání a s ovladači MME při záznamu, protože DirectSound nepodporuje funkce spojené se vstupem. ASIO ovladače nebudou implementovány.

4.5 Práce s komponentou DirectSound v C#

DirectX není zakomponováno ve Windows již od výrobce, ale je potřeba jej doinstalovat před použitím aplikace, která využívá některou z jeho komponent. Tak je tomu i s při jeho použití v prostředí C#. Nejprve je nutno nainstalovat tzv. DirectX SDK které obsahuje knihovny pro práci s jeho komponentami z prostředí .NET. Pokud chceme tyto knihovny využívat, stačí v prostředí .NET vytvořit, nebo otevřít projekt. Následně zvolíme v okně **Solution Explorer** pravým tlačítkem položku **Reference,** poté **Add Reference,** a na následujícím okně v záložce **.NET** zvolíme požadovanou komponentu, v našem případě **Microsoft.DirectX.DirectSound**. Poté je nutno ji zavést do třídy, která bude využívat jeho možností (`using Microsoft.DirectX.DirectSound;`).

Navrhovaná aplikace komunikuje s DirectSound pomocí třídy **SoundPlayer** v souboru **DS_SoundStream.cs**. Hlavním úkolem této třídy je vytvoření tzv. SecondaryBufferu (hw/sw buffers z obrázku 11.), který je cyklicky přehráván. V něm nastavíme dva body ověření (*position notify*), při jejichž průchodu dojde k probuzení čekajícího vlákna, které spí ve smyčce metody `SoundPlayback()`. Více napoví následující obrázek 13 a popis:



Při prvním volání metody `SoundPlayback()` dojde k zaplnění bufferu daty a spustí se jeho přehrávání. `SoundPlayback()` čeká.

Poté co je přehrán vzorek na notifikační pozici, je vlákno probuzeno a vyvolá vzdálenou metodu, která generuje nové vzorky.

Po dalším průchodu notifykační zónou je doplněna zase druhá část bufferu. Fáze **2,3** se nyní již opakují a buffer je doplňován nepřetržitě.

obr. 13: Princip funkce cyklického zásobníku

Zmiňovaný delegát `PullAudio()`, má přiřazenou vzdálenou metodu `Player.Vzorky()`, jejím úkolem je počítat vzorky podle zjednodušeného schématu syntézy. Jak to probíhá bude vysvětleno v následující kapitole.

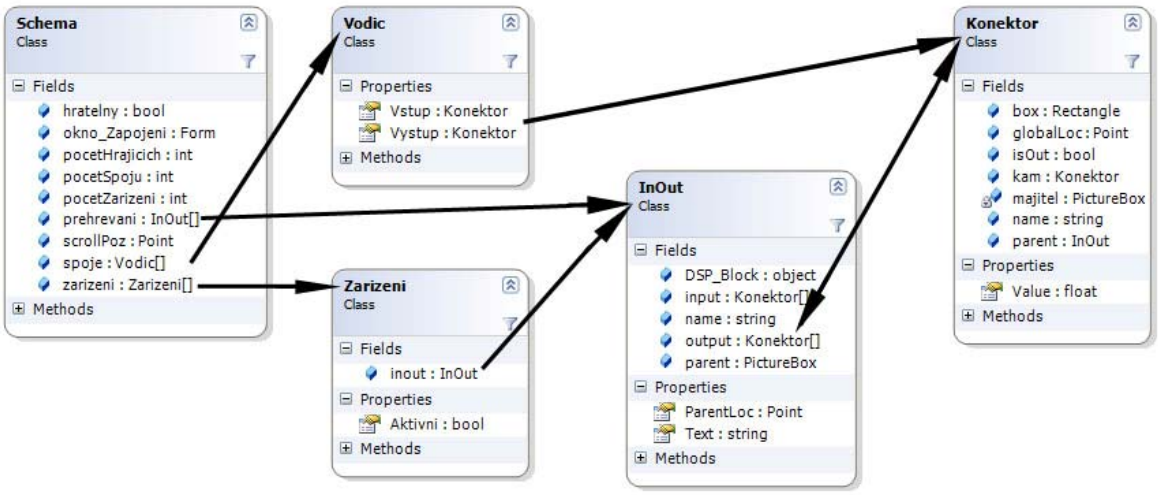
4.6 Třída Schema

`Schema` je nejdůležitější třídou celé aplikace. Nese informace o všech využívaných blocích, spojích a převádí navržené schéma do podoby vhodné pro další zpracování. K tomu je zapotřebí zvolit vhodný přístup. Ten vychází z předpokladu, že schéma je tvořeno zařízeními, spoji, konektory a vždy má pouze jeden výstup. Proto je vhodné takto rozčlenit i dílčí třídy, které budou zastupovat jednotlivé úrovně abstrakce.

Následuje popis nejvýznamnějších tříd:

1. Nejníže je třída `Konektor`, obsahuje informace potřebné k určení polohy konektoru v prostoru rodičovského okna, dále nese hodnotu a referenci na rodičovský objekt. Také rozlišuje zda-li je konektor výstupem či vstupem a nese informace o aktivní/neaktivní barvě. Pokud je konektor připojen obsahuje proměnná `kam` odkaz na připojený konektor.
2. Funkční blok je na obecné úrovni definován třídou `InOut`, ta obsahuje pole `Konektor`ů (vstupy/výstupy), jméno bloku a dva rodičovské objekty, z nichž první je vizuální `Parent`, který odkazuje na `PictureBox` (obrázek zařízení v zapojení) a druhý, referuje na třídu s konkrétní funkcí daného zařízení a je nazván `DSP_Block`.
3. Dále je zavedena třída `Zarizeni` doplňující `InOut` o vlastnost `aktivni`. Ta určuje je-li zařízení zahrnuto v uživatelském výběru (lze přesouvat a mazat více zařízení). Také obsahuje kolekci `PictureBox`ů vytvořených a používaných daným zařízením.
4. A nakonec třída `Vodic`, jejímž jediným obsahem jsou dva `Konektor`y. Vodič je znázorněn žlutou čárou, a konektory jsou rozlišeny na vstupní a výstupní, aby nešlo spojit vstup se vstupem, nebo naopak.

Vše by mělo být patrné z následujícího obrázku, kde jsou zobrazeny výše popisované třídy s nejvýznamnějšími proměnnými a vlastnostmi. Šipkami jsou naznačeny vztahy mezi jednotlivými třídami.

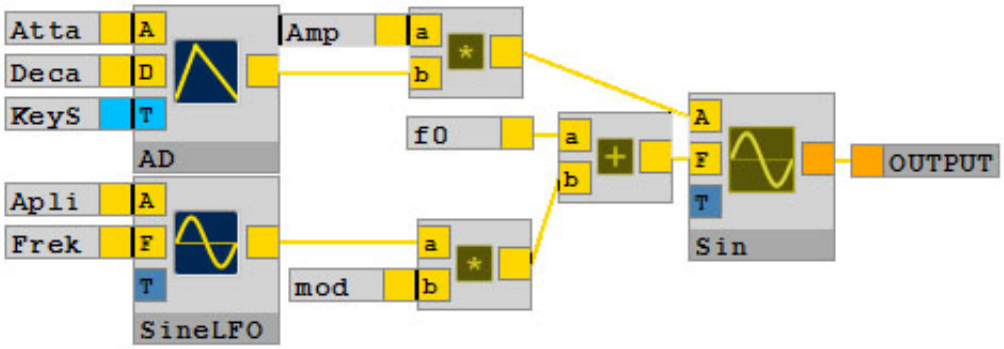


obr. 14: Shema a jeho vztah k dílčím třídám

Jak je vidět, třída `Schema` obsahuje tři pole. První, `spoje`, v sobě zahrnuje všechny vytvořené vodiče a je vytvořeno z důvodu nutnosti překreslování čar při pohybu oknem.

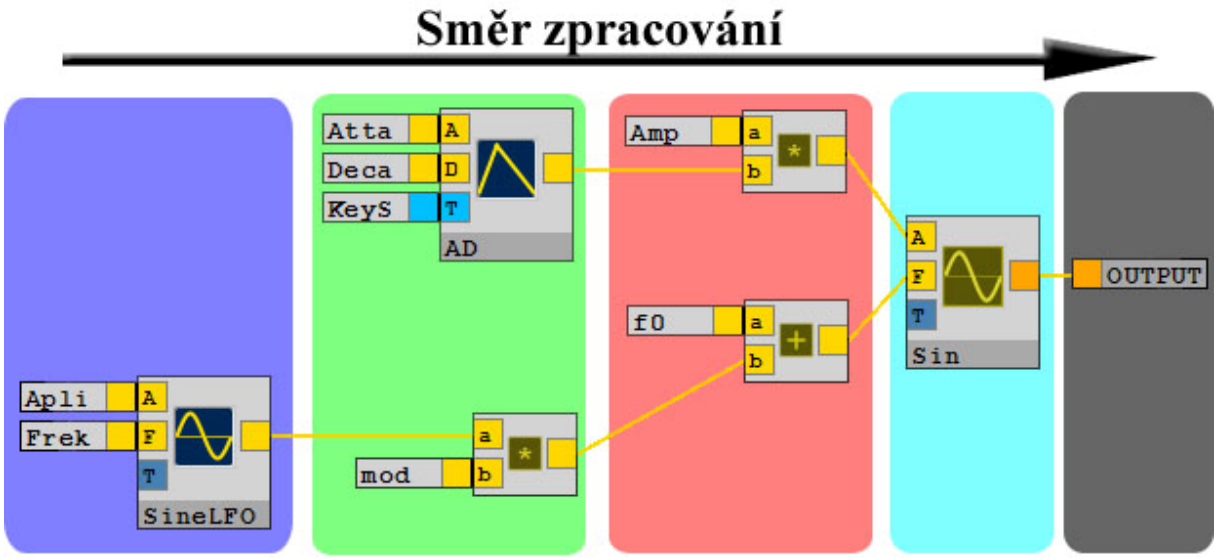
Druhé pole, `zarizeni` obsahuje všechna zařízení, které jsou na ploše okna panel. Poslední pole, `prehravani` obsahuje seřazený seznam zařízení, ten je sdílený metodou `Player.Vzorky()`, která podle něj generuje výstupní signál.

Nyní přejdu k vysvětlení funkce nejdůležitější metody `Schema.SortToPlay()`. Má za úkol setřídit jednotlivá zařízení do pole `prehravani`. Algoritmus vychází z poznatku, že schéma zapojení jde na obecné úrovni reprezentovat acyklickým orientovaným grafem se známým počátečním vrcholem. Obrázek 15 je toho důkazem.



obr. 15: Obecné schéma zapojení

Popis problému vyjádřený grafem je správný, ale zbytečně komplikovaný. Bližší zobecnění nabízí struktura stromu, kde output bude kořenem a zařízení listy i pro případ úplně jiného zapojení. V tomto kroku je ještě dobré si uvědomit existenci úrovní (pater) stromu, protože to přímo souvisí s dalším zjednodušením, které bude provedeno. U zařízení na stejných úrovních nezáleží na pořadí zpracování. Obrázek 16 názorně ukazuje tuto skutečnost pro případ zapojení z obr.16.



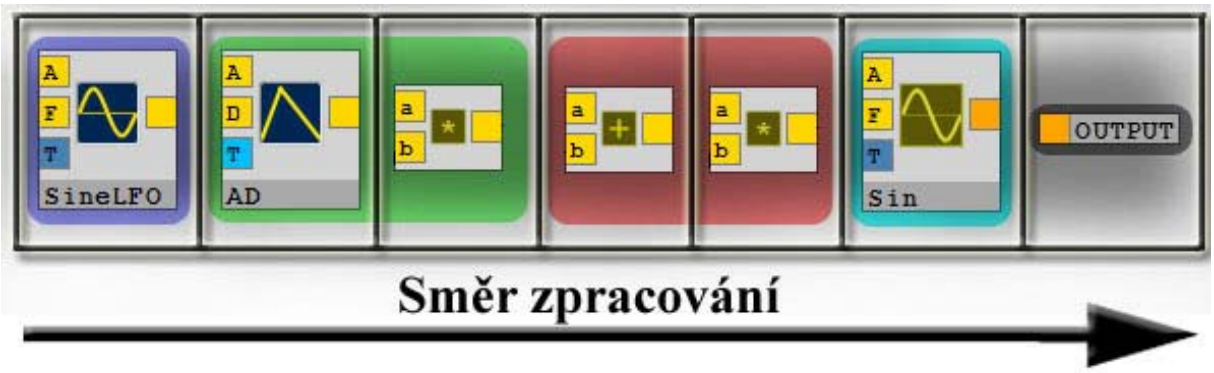
obr. 16: Schéma zapojení s barevným vyznačením úrovní

Jednotlivé úrovně zpracování jsou zde odlišeny barevnými obdélníky. Jelikož ovládací prvky neobsahují vstup, mohou být zařazeny na úroveň zařízení které řídí. Koncepce třídy `Konektor` v sobě zahrnuje odkaz na připojený konektor `kam` a vlastnost `Value`, která nese hodnotu na konektoru. Ta při svém nastavování vykonává tuto podmínku:

```
if ((this.kam != null)&( this.isOut)) this.kam.value = this.value;
```

Funkce podmínky je následující. V případě, že konektor je zapojen a je výstupem, změní hodnotu i na konektoru `kam`, s kterým je propojen. Následkem toho lze přenášet hodnoty zapojením, takže v tuto chvíli již vodiče nehrají žádnou roli.

Zjednodušení celého problému lze zakončit vyjádřením stromu pomocí setříděného pole objektů `InOut`, to je v poli `prehravani`. Následující obrázek 17 zobrazuje schéma zapojení z předchozích obrázků (č 15,16), setříděné ke zpracování,



obr. 17: Interpretace zapojení v poli `prehravani`

kdy jednotlivé barvy a schéma korespondují s obr.16, černou mřížkou je naznačeno umístění prvků v poli. Nyní se pozastavím nad problémem, jak z takto setříděného seznamu, přejít ke generování a zpracování signálu.

4.7 Způsob zpracování

Způsobu zpracování z takto připraveného seznamu zařízení vychází ze skutečnosti, že zařízení mají odlišnou vnitřní strukturu. Každé je implementováno vlastní třídou, což není špatné do doby, dokud nenarazíme na problém polymorfizmu. Jde o nutnost vytvoření přístupového bodu vlastního všem zařízením. Jednou z možností jak na to v jazyce C# je využití rozhraní. Každá třída může totiž dědit pouze od jedné mateřské třídy a několika rozhraní. Rozhraní obsahuje pouze prototyp metody, která musí být implementována ve všech třídách, které od daného rozhraní dědí.

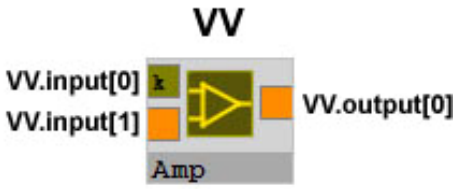
Implementace rozhraní v programu je následující:

```
…
public interface IDavaVzorky  // Deklarace rozhraní
{
      void Vzorek(); // Prototyp metody
}

…
```

Příklad třídy, provádějící zesílení signálu je následující:

```
public class VCA : IDavaVzorky
{
      ... // Konstruktor a deklarace proměnných
      public void Vzorek() // Vrací zesílený vstupní vzorek
      {
            zesileni = VV.input[0].Value;
            VV.output[0].Value = VV.input[1].Value * zesileni;
      }
}
```



obr. 18: Ukázka vstupů a výstupů zesilovače

Jak bylo výše zmíněno o doplňování vzorků zásobníku zvukového ovladače se stará metoda `Player.Vzorky()`. Jejím úkolem je lineárně projít schéma syntézy v poli `prehravani` ve směru zpracování, v každém kroku vykonat metodu `IDavaVzorky.Vzorek()`, která ze vstupů vypočte výstup. Z následující části kódu je patrný průběh výpočtu jednoho vzorku:

```
…
for (int z = pocetHrajicich - 1; z > 0; z--)
{
      IDavaVzorky zarizeni = prehrevani[z].DSP_Block as IDavaVzorky;
      if (zarizeni != null) zarizeni.Vzorek();
}
mixBuffer[n] = vystup.Kvantize(OUTPUT.Value);

…
```

Vlastní zpracování probíhá na urovni čísel s pohyblivou řádovou čárkou, audiosignál je normován na rozsah (**-1**…**1**), přepočet na 16-ti bitový rozsah je proveden až při přiřazování výstupní hodnoty. Přetypování operátorem `as` je voleno z důvodu vyšší efektivnosti kódu, protože k vytvoření `zarizeni` dojde jen v případě, když `DSP_Block` disponuje rozhraním `IDavaVzorky`, jinak `zarizeni = null`.

4.8 Využití prostředků jazyka C#

V aplikaci je využito i některých dalších možností jazyka C#:

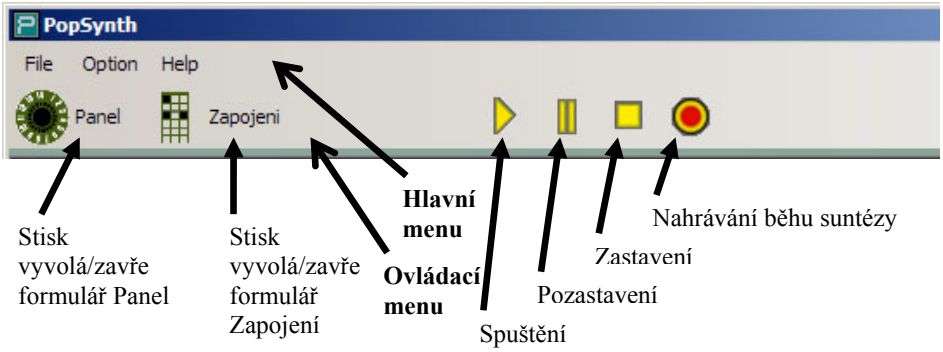
- Událostí - hlavně při řešení interakce s uživatelem. Obsluha klávesnice, myši, překreslování.
- Serializace - při ukládání struktury zapojení do souboru, formát dat je volen XML.
- Atributů - při kreslení transparentních obrázků.
- Synchronizačních primitiv - k řízení přístupu ke sdílené části kódu.

5 Manuál

Je vytvořen s důrazem na uživatelský přístup k aplikaci. Nezabývám se zde kódem ani způsobem naprogramování. Jde o představení jednotlivých bloků, možností a způsobu ovládání.

5.1 Hlavní formulář

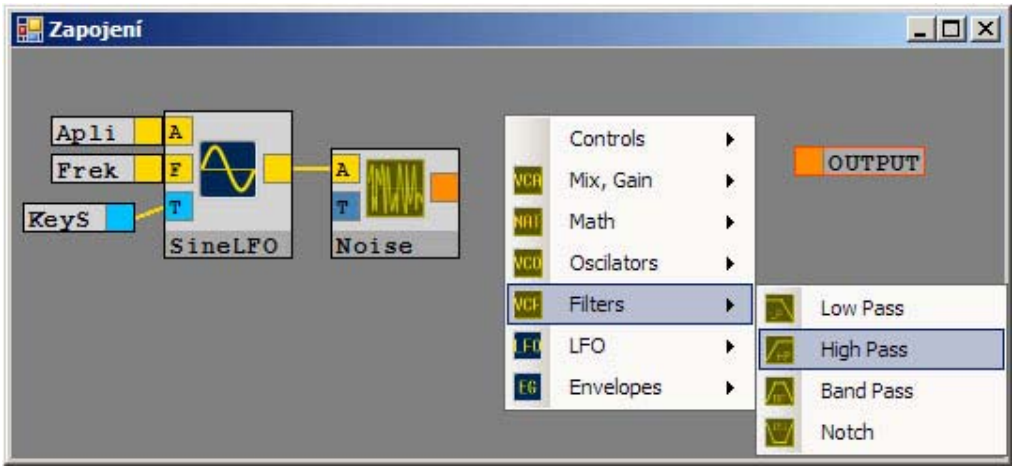
Hlavní formulář obsahuje dvě menu: 1) Hlavní menu s obvyklým obsahem. 2) Ovládací menu jehož položky jsou popsány na následujícím obrázku.



obr. 19: Hlavní aplikační formulář

5.2 Formulář Zapojení

Formulář Zapojeni slouží k tvorbě schémat syntézy, kde je nejdůležitějším prvkem zařízení. Všechny možnosti spojené se zařízením lze ovládat přes kontextová menu.

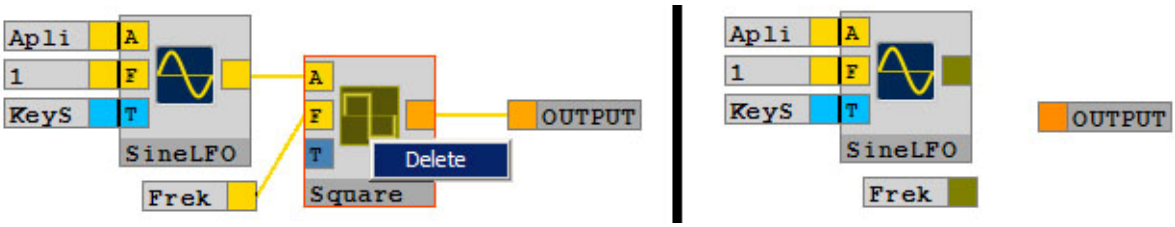


obr. 20: Formulář zapojení a hlavní kontextové menu

Přidání nového zařízení provedeme stiskem pravého tlačítka na pracovní ploše, tím vyvoláme hlavní kontextové menu, kde lze provádět výběr ze všech zařízení. Vybrané zařízení je pak umístěno na pozici, kde bylo menu vyvoláno (levý horní roh menu). Vše nejlépe demonstruje obrázek 20.

Mazání zařízení je umožněno dvěma způsoby:

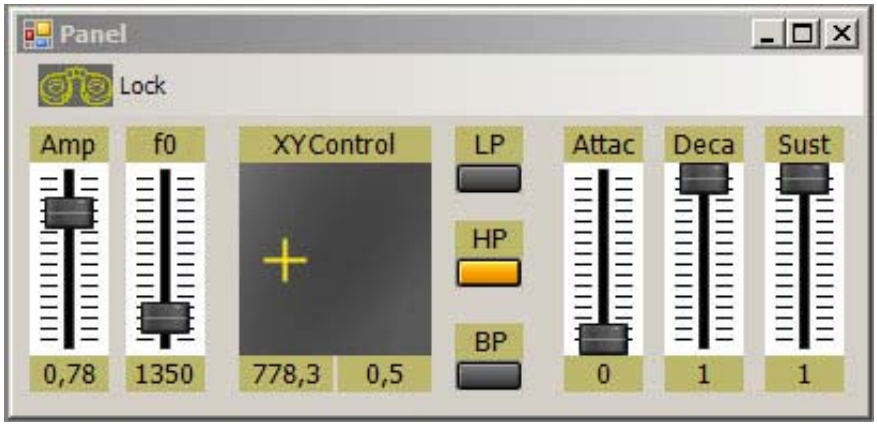
- Na zařízení pravý klik a z menu položka smazat, nebo klávesa **Del**. Použití prvního způsobu je na obrázku 21.
- Vybrat skupinu zařízení stiskem pravého tlačítka na ploše a následným tahem a smazat je klávesou **Del**.



obr. 21: Smazání zařízení

5.3 Formulář Panel

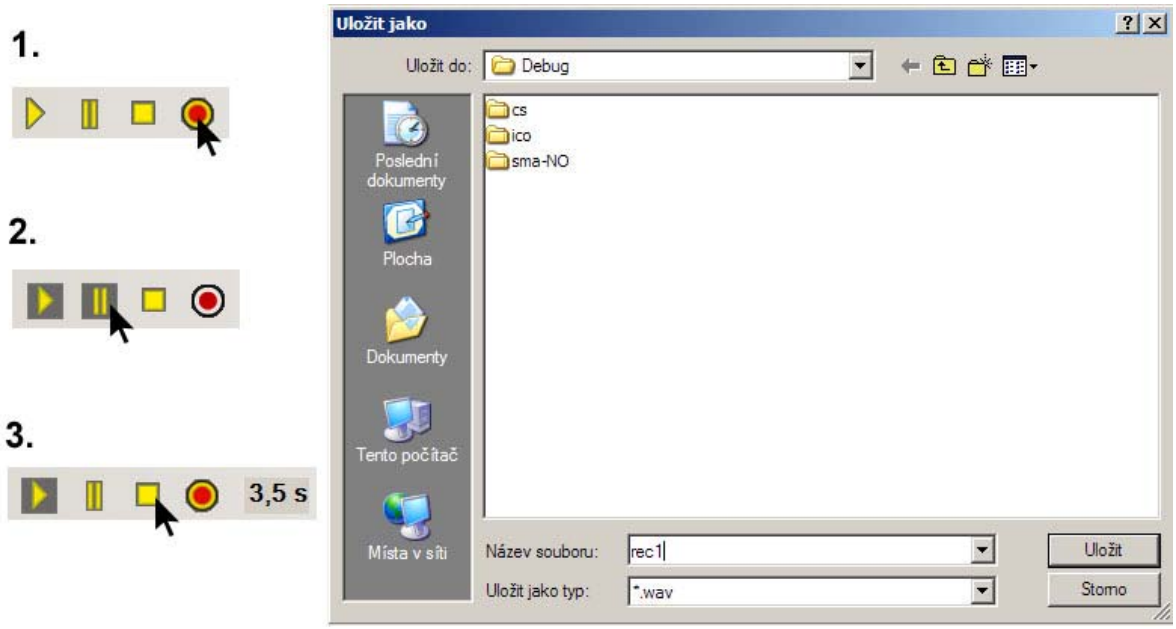
Na formuláři Panel jsou zobrazovány ovládací prvky. Umísťovány jsou automaticky, aby se nepřekrývaly. Polohu všech prvků lze měnit při odemknutém zámku. To je naznačeno tečkovaným pozadím. Při zamknutém zámku můžeme jednotlivé prvky ovládat. Buď stiskem a tahem v případě Faderu, XY Controlleru, nebo jenom stiskem pro případ tlačítka. Obrázek 22 ukazuje formulář Panel s několika ovládacími prvky.



obr. 22: Formulář panel s ovládacími prvky

5.4 Nahrávání

Nahrávání generovaného zvuku je možné následujícím způsobem. Kliknutím na tlačítko nahrávaní z ovládacího menu, tím vyvoláme formulář pro uložení souboru. (Bod 1. z obrázku 23) Ve formuláři je nutné zadat jméno souboru, do kterého chceme nahrávat. Běh syntézy se poté přepne do stavu Pause (druhý bod obrázku 23). Pro spuštění nahrávání je nutno kliknout na tlačítko pozastavení (pause) a spustit běh nahrávání. Doba nahrávání je poté ukazována v textboxu napravo od tlačítka nahrávání. Konec nahrávání provedeme stiskem tlačítka stop.



obr. 23: Demonstrace způsobu nahrávání.

Formát výstupního zvukového souboru je *.wav. Má stejné parametry jako vlastní syntéza, takže: Mono / 44.1Khz / 16Bit / Nekompresovaný.

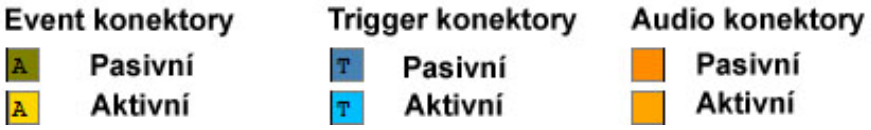
5.5 Popis konektorů

Schéma syntézy je tvořeno zařízeními, u kterých jsou rozlišeny tři typy konektorů. Pomocí konektorů je možné propojování zařízení mezi sebou. A to stiskem tlačítka na konektoru a tahem. Pokud je tlačítko uvolněno nad konektorem jiného zařízení dojde ke spojení.

Mazaní spoje je možné provést z kontextového menu vyvolaného stiskem pravého tlačítka nad konektorem. Následuje tabulka 2 s popisem možností a použití konektorů:

Konektor	Použití	Rozsah	Doporučené zapojení
Event	Přenos pomalých a řídících dat.	Podle popisu zařízení	Řízení oscilátorů, filtrů, zesilovačů.
Trigger	Spouští funkci zařízení	Trigger ≤ 0 = OFF Trigger > 0 = ON	dvoustavovému ovládaní klávesnice, tlačítko
Audio	Pro přenos audio signálu	(-1…1), výstup je limitován max= |1|	Řetězec zařízení musí končit výstupem. OUTPUT

tabulka 2: Popis jednotlivý konektorů



Barevné odlišení jednotlivých konektorů.

Aplikace je navržena tak, že nebrání mezi sebou propojovat jednotlivé typy konektorů, hlavně z experimentálních důvodů, ale musí být brána zřetel na skutečnost, že některé Event vstupy jsou pomalejší a nestíhají zpracovávat audio signál. Jediné co není možné je zavedení zpětné vazby.

5.6 Popis zařízení

Použitá zařízení lze obecně rozdělit do několika skupin:

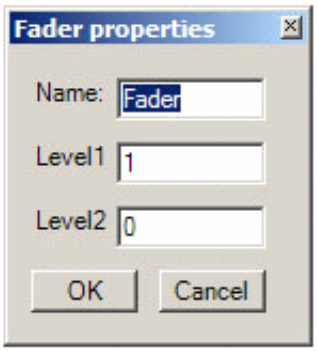
- Ovládací prvky
- Zdroje signálu
- Zařízení signálové cesty
- Modulátory

5.6.1 Ovládací prvky

Slouží k provedení uživatelského zásahu do průběhu syntézy. Popis všech prvků dokumentují obrázky ze kterých je patrný způsob zobrazení a jejich možnosti. Formulář vlastností je možné vyvolat poklikáním na prvku.

Fader

Na panelu Formulář V zapojení



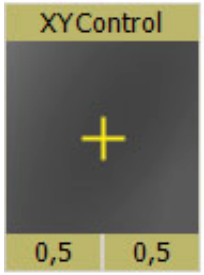
Fader je zástupce klasického potenciometru s posuvným jezdcem.

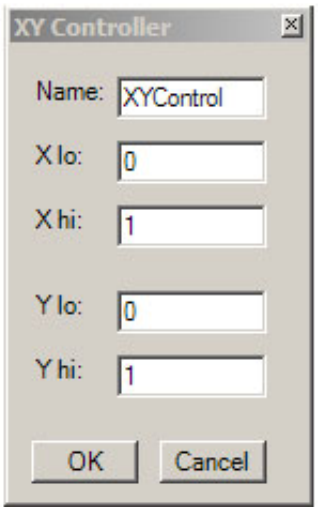
Nese hodnotu, kterou je možné měnit posunem jezdce. Posun je umožněn při zamknutém panelu a to jeho chycením (levý stisk) a táhnutím.

Formulář nastavení umožňuje měnit jméno a rozsah. Fader je vhodný k řízení událostí spojených se změnou zesílení, frekvence,...

XY Controller

Na panelu Formulář V zapojení



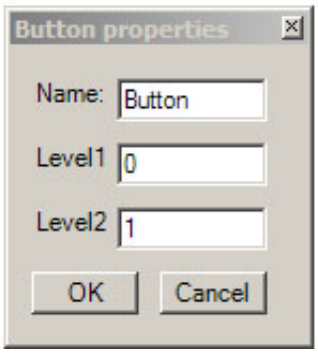


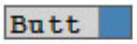
XY Controller Vychází z myšlenky dvourozměrného ovládání. Křížem je možno nastavit polohu vektoru na ploše a výstupem jsou jeho souřadnice. Počátek je v pravém dolním rohu. Oba výstupní konektory (X, Y) přísluší jedné souřadnici a nesou její hodnotu.

Využití XY Controlleru je podobné jako u Faderu. Je dobré ho použít k řízení vstupům, které spolu nějak souvisí, např. s ním můžeme řídit frekvenci dvou oscilátorů, kdy pohybem po diagonále zachováváme poměr frekvencí obou oscilátorů 1:1.

Button

Na panelu Formulář V zapojení



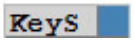


Buton (Tlačítko), slouží k dvoustavovému řízení. Přepínání stavů je možné při zamknutém panelu klikem na tlačítko. Vypnuté tlačítko nabývá hodnoty **Level1** a je tmavé. Zapnuté nese hodnotu **Level2** a má žlutou barvu.

Je vhodné k řízení trigger vstupů.

KeyControl

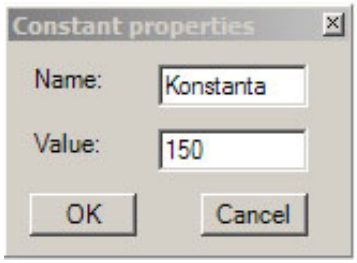
V zapojení



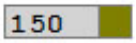
KeyControl je uživatelský vstup z klávesnice, tudíž nemá obdobu na panelu. Vstupní klávesy nejsou nijak rozlišovány, KeyControl reaguje pouze na přítomnost jejich stisku. Funguje na podobném principu jako tlačítko. Pokud není klávesa stisknuta nese hodnotu 0, po stisku klávesy má hodnotu 1. Vhodný k řízení trigger vstupů.

Constant

Formulář



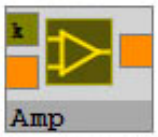
Ve schématu



Konstantu je vhodné použít všude tam kde není potřeba vstup přímo řídit. Nese jméno a hodnotu, zobrazuje se pouze ve schématu. Popisek ukazuje její hodnotu.

5.6.2 Zařízení signálové cesty

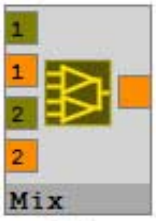
Zesilovač



Zesilovač

Zesilovač (**Amp**) zesiluje signál v závislosti na vstupu **k**, který je lineární. Doporučovaný rozsah je (**0**…**1**), může být řízen i audio signálem.

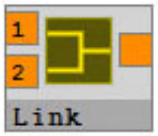
Mixér



Mix

Mixér (**Mix**) je tvořen dvěma zesilovači a sčítacím členem. Výstup je roven součtu obou zesílených vstupních signálů. Vhodný rozsah zesílení (zelené vstupy 1,2) je (**0**…**1**). Použití je vhodné všude tam, kde nestačí signál sloučit nebo sečíst.

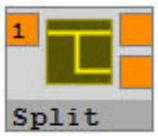
Slučovač



Slučovač

Slučovač (**Link**) slouží ke spojení dvou audio signálů. Vnitřně funguje jako součet s limitací. Pokud je součet jinde než v (**-1**…**1**) tak vrací nejbližší hodnotu tohoto rozsahu.

Rozdělovač



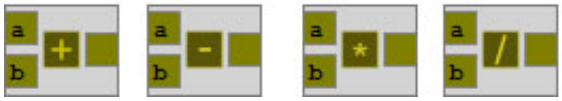
Rozdělovač

Rozdělovač (**Split**) rozdělí vstupní signál na dva. Vhodné použití je např. distribuce šumového signálu do různých filtrů.

5.6.3 Matematické operace

Zavedeny jsou čtyři základní matematické operace: sčítání, odčítání, násobení a podíl, který při nulovém vstupu **b** vrací hodnotu 1. Schematicky jsou vyobrazeny vlevo. Jejich použití je neomezené fungují v propojení se všemi typy konektorů. Také rychlost zpracování je vysoká, takže přítomnost vyššího počtu těchto zařízení ve schématu nedělá problémy.

Schématické značky matematických operací



5.6.4 Oscilátory

Oscilátory jsou zdroje signálů:

Sin…… Sinus

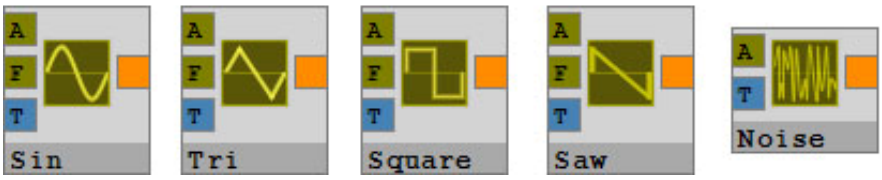
Tri…… Trojúhelník

Square…. Obdélník

Saw……...Pila

Noise…… Šum

Schématické značky oscilátorů



Aplikace nabízí pět variant. První čtyři podle obrázku nahoře mají tři vstupy, šumový má pouze dva. Následuje popis jednotlivých konektorů:

A. Vstup zesílení amplitudy. Rozsah (**-1**…**1**).

F. Vstup frekvence. Rozsah (**20**..**15000**)Hz.

T. Trigger vstup (**OFF** ≤ **0** < **ON**).

Výstupem je audio signál (**-A**…**A**).

5.6.5 Filtry

Slouží k úpravě spektra signálu:

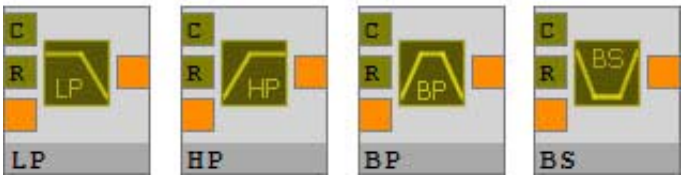
LP….. Dolní propust

HP…..Horní propust

BP….. Pásmová propust

BS…...Pásmová zádrž

Schématické značky filtrů



Všechny typy filtrů mají shodný počet konektorů jak je patrné z obrázku nahoře, oranžové jsou audio vstup/výstup. Přiváděný audio signál musí být v mezích (-1…1).

Popis vstupů pro události:

C. Ořezávací frekvence filtru (F_c). Rozsah (**20**…**15000**)Hz.

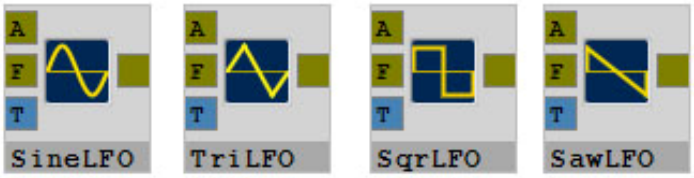
R. Velikost rezonace (zesílení na F_c). Rozsah (**0**…**10**).

Výstup je audio signál v rozsahu (**-1**…**1**).

5.6.6 Modulátory

LFO

Schématické značky pomalých oscilátorů



Jsou to pomalé oscilátory, které generují události. Jak ukazuje obrázek nalevo navrhnuty jsou čtyři varianty o různém výstupním průběhu.

Vhodné použití je např. řízení frekvence, zesílení, rezonance. Pokud řídíme frekvenci je zapotřebí k výstupu přičíst střední hodnotu frekvence (Nosnou), protože výstup osciluje kolem 0.

Popis konektorů:

A. Výstupní zesílení. Rozsah není doporučen.

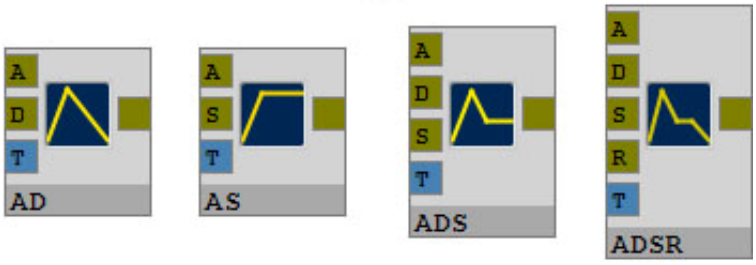
F. Vstup frekvence. Rozsah (**0,1**…**10**)Hz.

T. Trigger vstup (**OFF** ≤ **0** < **ON**).

Výstupem je událost (-**A**…**A**).

Obálky

Schématické značky generátorů obálek



Princip generátoru obálky je vysvětlen v kapitole 3.4. Obálku je vhodné použít tam, kde potřebujeme zdůraznit dynamiku v závislosti na čase. Takže např. pro různé údery, pomalé náběhy, dlouhé dozvuky. Výstup je normován (0…1).

Popis konektorů generátorů obálek:

A. Attack, náběh (**0**…**1**)s.

D. Decay, pokles (**0**…**1**)s.

S. Sustain, držení (**0**…**1**).

R. Release, doznění (**0**…1)s.

T. Trigger vstup (**OFF** ≤ **0** < **ON**).

Výstupem je událost (**0**…**1**).

Popsáním vstupů a výstupů generátoru obálky zakončuji část práce věnovaný uživatelskému manuálu. Je dosti možné, do aplikace budou ještě pře odevzdáním zakomponovány bloky sloužící k vizualizaci vytvářeného průběhu tzv. LevelMeter a Scope. Budou koncipovány jako průchozí audio IN/OUT bloky s volbou rozlišeni v amplitudové oblasti. LeverMetr bude zobrazovat aktuální hodnotu pomocí obdélníkového grafu a Scope bude zobrazovat generovaný průběh v čase, kdy rozlišení časové základny bude volitelný parametr.

6 Srovnání

Na profesionální úrovni existuje nástroj Reaktor vyvíjen německou firmou Native Instruments GmBH. Jde taktéž o software pracující na principu modulární syntézy, určený pro tvorbu virtuálních hudebních nástrojů pro systém Mac OS a Windows XP. Protože tato aplikace oslavila letos své desáté narozeniny, disponuje řadou nezměrných výhod. Např. zařízení jsou dělena do třech úrovní: modul, makro, nástroj. Tak, že z modulů se skládá makro a z maker nástroj. Z tohoto pojetí lze říci, že mnou navrhnutá aplikace generuje zvuk na úrovni modulů. Další výhodou Reaktoru je spolupráce s MIDI a VST technologiemi. Zjednodušeně lze říci, že MIDI je komunikační rozhraní (mezi nástroji) na hardwarové úrovni a VST na softwarové. Popis produktu je možné najít domovské stránce Native Instuments http://www.native-instruments.com/index.php?id=reaktor5new_us.

Na nekomerční scéně existuje několik volně dostupných aplikací fungujících na principu modulární syntézy. První je syntezátor nazvaný *OM*, je navržený pro operační systém Linux a může se pyšnit pěkným uživatelským rozhraním a spoluprácí s MIDI. Popis s možností stažení se nachází na adrese http://www.nongnu.org/om-synth/. Jinou na modulárním základě postavenou aplikaci je možno najít na http://www.pawfal.org/Software/SSM/ jmenuje se *SpiralSynthModular*. Je navrhnuta pro sytém Mac Os, mezi jeho přednosti patří originální a velmi popisná vizualizace generovaného průběhu na oscilografu ve tvaru mezikruží. Poslední aplikace je určena pro systém Windows XP a je nazvaná *Syd*, disponuje vizualizací průběhu. Dále přímo sčítá vstupy na konektoru, takže na jeden konektoru může být zapojeno několik vodičů, nejlépe je to patrné z domovské stánky produktu http://www.jbum.com/syd/ tam je možno také stáhnou zdrojový kód aplikace.

Z akademického pohledu je velice zajímavý vývoj italské University of Genova a to hlavně na katedře Počítačů Komunikace a Systémových věd (Department of Communication Computer and System Sciences), kde vyvíjí nástroj zvaný EyesWeb, určený ke spojení obrazových a zvukových signálů. Většinou pro použití v případech, kdy je z hudby generován obraz, nebo naopak. Podrobný popis tohoto projektu s možností stáhnutí prostředí je na http://www.infomus.dist.unige.it/eywindex.html.

7 Závěr

V závěru se chci věnovat shrnutí práce a ukázat další možnosti využití navrženého modulu.

V práci jsem nastínil způsob, jakým se dají vytvářet zvukové signály vhodné pro syntézu různých varovných nebo upozorňujících signálů, jejichž parametry lze měnit podle potřeby. Základem bylo teoretické přiblížení pojmů z oblasti syntézy zvuku. Zabýval jsem se jednotlivými metodami syntézy, následoval popis bloků využívaných zvukovými syntezátory a volba modulární koncepce navrhovaného syntezátoru.

V praktické části bakalářské práce jsem se zaměřil na návrh programového modulu, který disponuje jak možností tvorby vlastního algoritmu syntézy, tak sadou již vytvořených schémat vhodných k syntéze poplašných a varovných signálů.

Tímto způsobem se dají vytvářet nejen výstrahy, ale provádět i syntézu pro virtuální modely výrobních procesů, které se používají při výuce řízení, a tak obohatit vizualizaci a přiblížit jí realitě. Další možnost využití je při programování DSP pro syntézu zvuku, kdy lze v modulu pohodlně navrhnout (odladit) schéma syntézy. Tvorba programu pro DSP sice nebude jednoduší, ale programátor dostane jistý přehled nad bloky a parametry, které pro svůj účel potřebuje naimplementovat.

Z pohledu dalšího vývoje práce se nabízí některá z těchto možností. Jelikož jde o modul navržený pro platformu .NET může být poměrně jednoduše rozšířen o síťové prostředky, např. může být navržen serverový modul, který by zpracovával požadavky od klientů a podle těch by generoval výsledný zvuk. Tato možnost přímo souvisí s výše zmiňovaným použitím k syntéze zvuků pro virtuální modely. V tomto případě by bylo vhodné doplnit modul ještě o sekvencer (viz kap.1.1). Budoucnost modulu se také může ubírat cestou spojení se světem HW, např. implementovat řízení jednotlivých parametrů syntézy z vnějšku, k tomuto účelu je výhodné využít MIDI rozhraní.

Software a část této páce jsou volně publikovány (od 22.1.06) na internetové adrese http://www.popsynth.wz.cz.

Závěrem mohu konstatovat, že jsem splnil zadání práce, která mi zároveň vytyčila určitý prostor pro další práci v oblasti.

A. Slovník pojmů a zkratek

API……...Je zkratka **Application Programming Interface**, což znamená rozhraní pro programování aplikací. Jde o sbírku procedur, funkcí či tříd nějaké knihovny, které může programátor využívat.

Latenci….Slouží se k popisu časových parametrů zvukového driveru. Je to čas mezi zapsáním vzorku do softwarového bufferu a jeho přítomností na výstupu zvukové karty.

SDK……. Je zkratkou **Software Development Kit**, jde o balíček vývojových nástrojů, které umožňují vývoj aplikace pro specifickou platformu, počítačový systém, nebo operační systém.

GUI……..Znamená **Graphical User Interface**, který je částí uživatelského rozhraní. K interakci uživatele s aplikací používá grafických prvků.

MIDI……Je zkratka pro **Musical Instruments Digital Interface**, umožňuje vzájemné propojení syntetizérů, sekvencerů, osobních počítačů, atd. pomocí standardního rozhraní. Je to ve své podstatě upravené komunikační rozhraní RS-232.

VST……..Je zkratka pro **Virtual Studio Technology**. VST je standardní rozhraní pro propojování audio efektových a syntezátorových pluginů s audio editory a HD nahrávacími systémy. Tato technologie se snaží vystřídat běžné studiové HW zařízení za jeho SW protějšky, které mají v dnešní době značnou řadu výhod.

B. Literatura

[1] Kotrubenko: „*Tajemství syntezátorů*“, Suprafon, Praha, 1988.

[2] Forró: „*MIDI - Komunikace v hudbě*“, Grada, Praha, 1993.

[3] F. Kadlec: „*Zpracování akustických signálů*“, Vydavatelství ČVUT, Praha, 2002.

[4] V. Hlaváč, M. Sedláček: „*Zpracování signálů a obrazů*“, Vydavatelství ČVUT, Praha, 2005.

[5] T. Archer, „*Myslíme v jazyku C#*“, Grada, Praha 2002.

[6] J. Hanák, „*C# praktické příklady*“, Grada, Praha 2006.

[7] J. Pirkl, „*Řešené příklady v C#*“, Kopp, České Budějovice 2005.

[8] Music DSP [online] 20.12.2006, [cit. 2006-12-20] <http://www.musicdsp.org/archive.php>

[9] DirectX 9.0 SDK [online] 15.11.2006, [cit. 2006-11-25] <http://msdn.microsoft.com/directx/sdk/>

[10] Windows driver API basics, [online] 28.12.2006, [cit. 2006-12-28] <http://www.staudio.de/kb/english/drivers/index.html>

[11] The Code Project, [online] 17.1.2006, [cit. 2006-12-08] <http://www.codeproject.com>

C. Obsah CD-ROM

K této práci je přiložen CD-ROM. Obsahuje text bakalářské práce, projekt vytvořený v Microsoft Visual Studiu 2005 a přeložený kód aplikace.

CD-ROM je členěn do těchto adresářů:

TextBP – obsahuje text bakalářské práce

Project – obsahuje projekt PopSynth pro Visual Studio 2005

PopSynth – obsahuje spustitelnou verzi programu.

backup – obsahuje druhou kopii celého cd pro případ poškození

Další informace jsou dostupné v textovém souboru v kořenovém adresáři CD-ROMu.