SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1015

Sinteza virtualnih glazbenih instrumenata upotrebom računala

Vladimir Jakšić

Zagreb, listopad 2008.

Zahvala

Prije svega, najdublje se zahvaljujem svom osobnom mentoru doc.dr.sc. Kristianu Jambrošiću na pomoći, vodstvu i savjetovanju tijekom izrade ovog rada.

Također, htio bi se zahvaliti svim članovima *#musicdsp* kanala na *Efnetovim IRC* serverima, posebice Steve Tayloru, Mike Janneyu i Karel Basselu na ukazanoj pomoći oko realizacije određenih programskih rješenja.

Također, moje zahvale idu i svim članovima *Steinbergove 3rd party developers* mailing liste te svim članovima DSP odsjeka *KVRaudio* foruma.

Sažetak

Ovaj diplomski rad predstavlja istraživanje razvojnog procesa audio aplikacije za sintezu glazbenog signala temeljene na Steinbergovom skupu razvojnih alata, zajedno s korištenim metodama te postignutim rezultatima.

Bolji uvid u razumijevanje principa današnjih tehnologija bit će dan osvrtom na povijesni pregled razvoja računalne sinteze.

Rad će se baviti glavnim principima glazbeno orijentirane računalne sinteze audio signala te danas najkorištenijim mehanizmima i sučeljima.

VST tehnologija kao odabrana tema ovog rada bit će detaljno opisana te smještena u kontekst današnjih konkurentnih tehnologija za glazbenu produkciju.

Kao posljedica ovog istraživanja nastao je polifoni sintetizator glazbenog audio signala, zajedno sa tumačenjima metoda korištenih u njegovoj realizaciji.

Sadržaj

Popis oznaka i kratica

VST	Virtual Studio Technology
DSP	digitalna obrada signala (engl. Digital Signal Processing)
D/A, A/D	oznaka za digitalno-analognu, i analogno-digitalnu konverziju
IBM	International Business Machines Corporation
PC	osobno računalo (engl. Personal Computer)
SID	čip korišten u Commodore 64/128 računalima (engl. Sound Interface Device Chip)
OPL	čip za FM sintezu zvuka (engl. FM **Op**erator Type **L**)
MIDI	interkonekcijski protokol za digitalne audio uređaje (engl. Musical Instruments Digital Interface)
LA	linearno-aritmetička sinteza (engl. Linear Arithmetic)
FM	frekvencijska modulacija
VAZ	Vurtual Analog Synthesizer
UG	generatorska jedinica (engl. Unit Generator)
ADSR	kratica za četverostupanjsku amplitudnu anvelopu (engl. Attack, Decay, Sustain, Release)
f_g	granična frekvencija filtra
Q	faktor dobrote titrajnog kruga
PhM	sinteza prema fizikalnom modelu (engl. Physical Modeling Synthesis)
DAW	radna stanica za digitalni audio (engl. Digital Audio Workstation)
VSTi	virtualni glazbeni instrument implementiran koristeći VST standard
GUI	grafičko sučelje dizajnirano za određenu aplikaciju (engl. Graphical User Interface)
SDK	razvojno sučelje za izradu programskog koda (engl. Source Development Kit)
CC	kontinuirani kontroler (engl. Continuous Controller)
IDE	integrirano razvojno okružje (engl. Integrated Development Enviroment)
SVF	State Variable Filter
IIR	filtar sa beskonačnim impulsnim odzivom (engl. Infinite Impulse Response)

Popis tablica

Popis slika

1. Uvod

Razvojem i upotrebom digitalnih uređaja te računalnih sustava, postupak glazbene kompozicije i produkcije postigao je najviši nivo isprepletenosti sa tehničkim i znanstvenim resursima današnjeg društva. Kroz široku primjenu računala u generiranju i obrađivanju glazbenog signala, kompozitori, iz kreativnih pobuda, potaknuli su čvrstu međupovezanost domena znanstvenog i muzičkog pristupa razmišljanju. Kao što se može reći da su znanost i tehnologija obogatili današnju glazbu, obrat također vrijedi: određeni glazbeni problemi u nekim slučajevima predlažu ili čak direktno nameću nove znanstvene te tehnološke probleme. Svaki sa svojim motivacijama, glazba i znanost ovise jedno o drugom te tako kreiraju jedinstveni odnos čija posljedica je obostrana korist.

Iako uporaba tehnologije u glazbi nije fenomen novijeg vremena, ona je danas, s ubrzanim razvojem računalnih sustava, dostigla novu razinu primjenjivosti. Moderni računalni sustavi obuhvaćaju mnogo šire koncepte od onih koji su svojstveni samim fizičkim uređajima. Jedan od distinktivnih atributa računarstva je programibilnost te time i programski jezici. Visoki programski jezici, kao predstavnici rezultata stoljeća razmišljanja o razmišljanju, jesu sredstva pomoću kojih računala postaju primjenjiva na širokom broju raznih disciplina.

VST sučelje za integraciju efekata i sintetizatora audio signala, kao odabrana tehnologija ovoga rada, sa svojim razvojnim sučeljem (SDK) predlaže programski jezik C++ kao primarnu paradigmu za razvoj DSP algoritama. Razvoj C++ programskog jezika počinje početkom 80-ih godina dvadesetog stoljeća, kao rezultat potrebe za novim pristupom programiranju koju dotadašnji proceduralni i strukturalni pristupi, te pripadajući programski jezici, nisu mogli zadovoljiti. C++ uvodi princip objektno-orijentiranog programiranja, te podržava četiri glavna svojstva koja čine temelj objektno orijentiranog programiranja: enkapsulaciju, sakrivanje podataka, nasljeđivanje i polimorfizam.

Za razvoj VST aplikacija potrebno je poznavanje principa i matematičke pozadine digitalne obrade signala uz solidno poznavanje nekog od objektnih jezika. Također, za razvoj VST aplikacija koje sadrže u sebi funkciju generiranja signala, takozvanih VST instrumenata, potrebno je poznavanje principa sinteze elektroničkog muzičkog signala. U tu svrhu, čitatelju će tokom ovog rada biti predstavljeni sadržaji koji daju uvid u područja koja tvore zadanu problematiku.

2. Povijesni razvoj računalne sinteze i obrade signala

U ovom poglavlju bit će prikazana povijest razvoja hardvera i softvera korištenog pri kompoziciji računalne glazbe. Računala su, tokom posljednja dva desetljeća, postala najvažnijim alatom za izradu i obradu današnje glazbe. Posljednjih godina, javlja se ubrzan razvoj raznih paketa programskih alata koji su, prije otprilike desetak godina, polučili jednu od najvažnijih posljedica svojeg postojanja, a ta je mogućnost obrade i sinteze signala u realnom vremenu.

Razvoj metoda pomoću kojih osobno računalo može proizvesti zvuk, u skoro bilo kojoj upotrebljivoj formi, može se velikim dijelom pratiti kroz razvoj samih osobnih računala.

Opseg ovog osvrta pokriva primjenu računala kao alata za sintezu i obradu zvuka, te primjenu unutar produkcije glazbe za multimedijske aplikacije.

2.1. MUSIC – N – prvi računalno orijentirani glazbeni programi

Rani računalni sustavi, korišteni prije sedamdesetih godina prošloga stoljeća, bili su neusporedivo sporiji od današnjih računala; te kao takvi nisu imali nikakvu mogućnost generiranja zvuka u realnom vremenu.

Max Mathews stvorio je, 1957. g., prvi primjer računalnog programa za generiranje audio signala pomoću direktne sinteze koji je nazvan MUSIC I. Program je pomoću skupova naredbi, polako generirao audio podatke koji su se akumulirali na medij za pohranu podataka (tj. magnetnu traku). Nakon toga, ti podaci trebali su biti provučeni kroz D/A konvertere, kako bi se učinili spremnim za slušanje.

MUSIC I kreiran je u organizaciji Bell Laboratories, te je tamo dostigao i svoja poboljšanja u programima MUSIC II, MUSIC III te MUSIC IV.

Nakon završetka rada na MUSIC IV programu, Mathews predaje rad Princeton sveučilištu gdje su mu dodane nove funkcije među kojima su: rezonantni filtar i kontrola vremenske anvelope. Time je stvoren prvi računalno temeljeni substraktivni sustav sinteze. Alteracije u kodu su također dodane kako bi program trošio manje procesorskog vremena na IBM 7094 računalu.

Nakon toga program MUSIC imao je veliki broj nasljednika, kao naprimjer (prema [1]):

- MUSIC IV-BF (ponovno napisan u FORTRAN-u, što je omogućilo portabilnost)
- MUSIC V (posljednji u nizu nastalih unutar Bell Labsa)
- MUSIC 360 and MUSIC 11 (nasljednik MUSIC IV-BF, napisao ga je Barry Vercoe)
- Csound (nasljednik MUSIC 11, te u širokoj uporabi danas)
- CMix / Real-Time Cmix (Paul Lansky, Brad Garton, i drugi)
- CMusic (F. Richard Moore)
- SAOL, strukturirani audio orkestralni jezik, koji je dio MPEG-4 audio standarda, napisao ga je Eric Scheirer

2.2. SID chipovi unutar Commodore 64 računala

S poboljšanjima unutar tehnologije razvijene pomoću tranzistora, postalo je moguće proizvesti računala dovoljno malih dimenzija; za razliku od dotadašnjih računala koja su zauzimala velike prostore usporedive sa dimenzijama prosječne spavaće sobe, te su se mogla naći samo na sveučilištima ili istraživačkim laboratorijima.

Za razliku od današnjih dana, kada je tržište preplavljeno IBM PC kompatibilnim osobnim računalima, 80-ih godina prošlog stoljeća postojali su alternativni tipovi računala dostupni na tržištu. Za određeni period vremena (1983/84/85) Commodore 64 (Slika 1.) dominirao je tržište s, otprilike, 40

postotnim udjelom [2]. Commodore 64 i Commodore 128 sadržavali su u sebi jedan od najnaprednijih zvučnih čipova koji su se zadržali u uporabi dugi niz godina.

Slika 1. Commodore 64

Commodoreov SID čip imao je mogućnost troglasne polifonije, od koje se svaki glas sastojao od generatora signala temeljenog na 8-bitnim valnim tablicama. Jedan glas mogao je generirati 4 vrste signala (trokutasti, pilasti, niz pravokutnih impulsa sa širinskom modulacijom, te šum). Također, imao je mogućnost subtraktivne sinteze koristeći ugrađeni filtar sa padom 12dB/okt. Čip je sadržavao u sebi i audio ulaz, pomoću kojeg ih se moglo spojiti u niz, iz kojeg se generiralo složenije zvukove.

Unatoč tome što je rezultirajući zvuk djelovao vrlo „elektronično“, nijedna druga serija računala koja je bila dostupna za kućanstva nije mogla postići Commodoreovu razinu oštrine tona i točnosti frekvencijskog podešavanja.

2.3. Zvučne kartice

2.3.1. „AdLib Music Synthesizer Card“ (ALMSC)

Sljedeći veći napredak unutar računalno temeljene audio tehnologije javio se 1987. godine u obliku ISA kartice koja se je mogla priključiti na IBM PC kompatibilna računala. Koristila je Yamahin YM3526 chip, koji je bolje poznat pod nazivom OPL (FM **Op**erator Type **L**).

OPL je bio polifoničan, sa mogućnošću reprodukcije do 9 glasova u isto vrijeme. Koristio je sinusoidalni valni oblik te dva FM operatora.

Zbog svoje velike popularnosti, 1989. OPL chip dodan je i Creativovoj „SoundBlaster“ seriji zvučnih kartica.

OPL je doživio dva unapređenja pod istim nazivom. OPL2 imao je iste mogućnosti kao i original, osim što su mu dodana 4 valna oblika a ne samo sinusoidalni. OPL3 sadržavao je 4 FM operatora te 18 glasovnu polifoniju što je učinilo rezultantni zvuk mnogo realističnijim te manje „elektroničnim“.

2.3.2. Programska podrška za MIDI sekvenciranje

Razvoj OPL čipova zajedno sa razvojem Microsoftovog operativnog sustava Windows (verzije 3.0), potaknuo je nastajanje softverskih MIDI sekvencera; programske podrške koja čini preteču današnjih virtualnih studijskih okruženja.

Ova programska podrška, skupa sa OPL čipovima omogućila je korisnicima stvaranje čitavih kompozicija uporabom osobnog računala unutar kojih su mogli dodavati zvukove kao što su: udaraljke (perkusije), vodeći instrumentalni zvukovi (lead), bas i ostali timbri.

Sučelja ranih sekvencera bila su vrlo primitivna u usporedbi s današnjim; nisu imala mogućnost baratanja sa digitalnim audio signalima, te je sučelje za slaganje uzoraka bilo mnogo manje intuitivno i pristupačno.

2.3.3. Roland MT-32

Alternativa Yamahinim OPL čipovima bio je Rolandov MT-32 sintetizator (Slika 2.). To nije bila zvučna kartica, već eksterni uređaj povezan s računalom pomoću ISA kartice nazvane „Musical Processing Unit, model 401“ (MPU-401).

MPU-401

Ova kartica je bila prvo MIDI sučelje izrađeno za računalo (1984.g). Sadržavala je u sebi mnoge funkcije, kao što su metronom te konektor za sinhronizaciju sa magnetnim kasetofonom.

To sučelje je steklo iznimno visoku popularnost, da je roland odlučio postaviti standard kojeg su mnoge druge kompanije u budućnosti slijedile. MPU-401 danas je u ponudi kao čip, a ne kao kartica, te ga je još moguće naći integriranoga u današnjim zvučnim karticama.

Slika 2. Roland MT-32 sintetizator

Linearno-aritmetička (LA) sinteza

MT-32 sintetizator funkcionirao je koristivši linearno-aritmetičku sintezu. Ta vrst sinteze nastala je 1987. unutar Roland korporacije kao konkurentska metoda Yamahinoj FM sintezi.

LA sinteza je preteča sintezi pomoću valnih tablica, a stvorena je kao kompromis između realističnosti i premalih memorijskih resursa koja su tadašnja računala pružala.

Osnovni princip rada je bio sljedeći: samo kratki, početni dio zvukova (attack portion) instrumenta bio je nasnimljen u PCM obliku na memorijsku jedinicu. Nakon toga uključivao se digitalni oscilator, koji se nastavljao na početni dio (sustaining portion).

2.3.4. SoundBlaster

Razvojem računalne tehnologije, povećale su se memorijske mogućnosti osobnih računala; te zajedno sa padom cijene postalo je moguće spremati kratke dijelove audio signala na tvrde diskove, pa čak i u radnu memoriju.

Iz tih razloga, u zvučne kartice počeli su se ugrađivati D/A i A/D konverteri koji su omogućavali reprodukciju i snimanje digitalnog audia. Soundblaster 1.0 je uz Yamahin 11-glasovni FM sintetizator sadržavao dio za digitalni

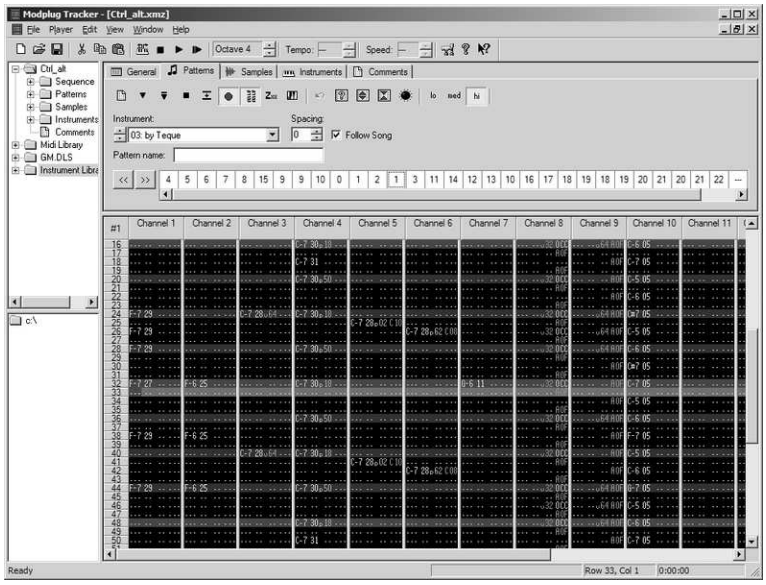
audio nazvan *DSP*. U ovom slučaju kratica DSP nije predstavlja digitalnu obradu signala, već „digitalni procesor zvuka“ (Digital Sound Processor).

Ova kartica imala je mogućnost reprodukcije monoauralnog 8-bitnog digitalnog audio signala uzorkovanog na frekvenciji do 23 kHz (otprilike FM kvaliteta); te mogućnost snimanja do 12 kHz (otprilike AM kvaliteta).

2.3.5. Pojava „tracker“ programske podrške

Tracker je generički naziv za softverske sekvencere koji omogućavaju korisniku da aranžira isječke zvukova po vremenskoj liniji unutar nekoliko monofoničnih kanala koristeći alfanumeričko sučelje (note su se unosile pomoću QWERTY tipkovnice,Slika 3.). Trackeri su, za Amiga i Commodore računala, postojali nekoliko godina ranije nego na IBM PC kompatibilnim računalima (1987.), zbog svojih naprednijih mogućnosti.

Nakon pojave SoundBlaster zvučne kartice, sa mogućnošću reprodukcije digitalnog audia, trackeri početkom 90-ih godina polako počinju gravitirati prema IBM kompatibilnim računalima.



Slika 3. ModPlug tracker

Druga zvučna kartica, koja je postala veoma popularna unutar trackerske scene je Gravisova Ultrasound kartica. Neko vrijeme, ona je nudila

neusporedivu muzičku kvalitetu. Sadržavala je 32 interna kanala te integriranu memoriju za spremanje isječaka.

2.3.6. 16-bitne zvučne kartice

Digitalni audio zapis u 16-bitnom formatu zauzima dvostruko više prostora na mediju za pohranu nego 8-bitni zapis. Za računala tog doba, takvo povećanje predstavljalo je veliki trošak. Međutim tehnologija je brzo sustizala te se javljaju komercijalne 16-bitne zvučne kartice, koje i danas, uz korištenje frekvencije uzorkovanja 44100Hz daju zadovoljavajuću kvalitetu reprodukcije za slušanje.

SoundBlaster 16 javlja se u lipnju 1992. godine. Podržavao je Yamahinu FM sintezu pomoću ugrađenog OPL3 čipa. Kartica je također imala ulaz za priključenje dodatne kartice (daughter soundcard) sa mogućnostima sinteze temeljene na snimljenim isječcima. Tracker softver prilagođava svoj format novim mogućnostima kao što su multi-sampling i 16 bitni digitalni audio.

2.4. Virtualni instrumenti

2.4.1. Offline virtualna sinteza

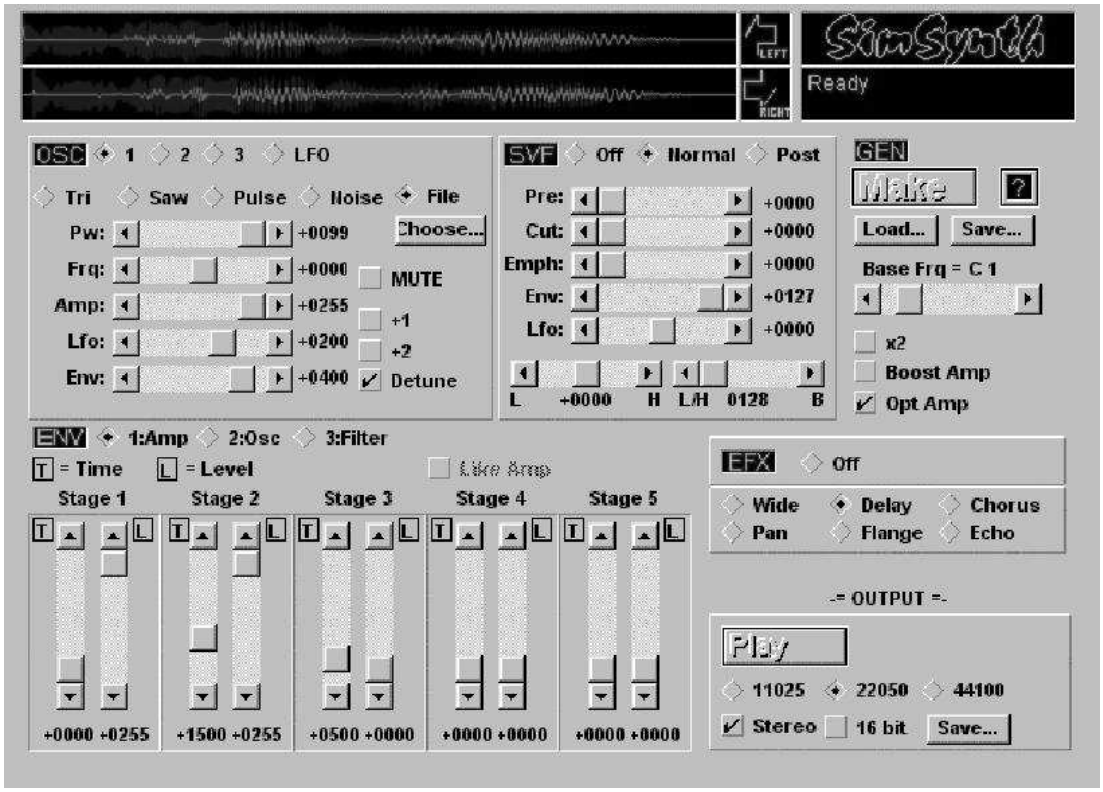
Postojanje programske podrške za emulaciju sintetizatora temeljenih na snimljenim isječcima programere je dovelo do ideje za emulacijom hardverskih sintetizatora koristeći mogućnosti novijih računalnih središnjih procesorskih jedinica.

U početku takvi programi nisu radili u realnom vremenu, već u tzv. „offline“ načinu rada. Korisniku je mogao unositi parametre pomoću potenciometara i kliznika, te u polja za unos alfanumeričkih znakova. Ti parametri su mijenjali određene varijable koje su definirale karakteristike zvuka (npr. valni oblici na izlazu iz oscilatora, granične frekvencije filtara, visina tona za pojedini oscilator i oblikovanje anvelope).

Korisnik, nakon što je bio zadovoljan postignutim rezultatom mogao je nasnimiti generirani zvuk u obliku .WAV datoteke. Ta datoteka sadržavala je

najčešće jednu glazbenu notu, te se je nakon toga učitavala u sekvencer gdje se je njome moglo manipulirati.

Među prvim izdanim komercijalnim softverom za offline sintezu nalazi se SimSynth koji je proizveden 1994. godine. Pružao je mogućnost generiranja 16-bitnih audio datoteka. Sučelje mu je bilo podijeljeno na nekoliko glavnih dijelova (Slika 4.): podešavanje oscilatora, podešavanje filtra, podešavanje vremenske anvelope za amplitudu i filtar te dodavanje efekta.



Slika 4. Simsynth 1.0 [3]

2.4.2. Virtualni instrumenti u realnom vremenu

S povećanjem procesorske brzine, postaje moguće generirati audio signal „on the fly“. Pojavljuje se programska podrška koja svoje izlazne podatke šalje direktno na zvučnu karticu bez prethodnog spremanja na memorijsku jedinicu. Ovaj napredak u brzini je veoma mnogo revolucionarizirao računalno temeljenu digitalnu obradu signala.

1996. godine javlja se jedan od prvih sintetizatora u realnom vremenu, naziva „Vurtual Analog Synthesizer“ (VAZ). Imao je slično sučelje kao SimSynth, s razlikom da se je rezultat mogao podešavati za vrijeme slušanja izlaznih podataka. VAZ je radio na IBM kompatibilnim platformama sa brzinama procesora 66 MHz ili više.

Ista godine, unutar Steinberg korporacije, VST tehnologija je integrirana unutar Cubase sekvencera, što mu daje mogućnosti studijskog okruženja u realnom vremenu sa opcijama kao što su ekvilizatori, efekti, miješanje i automatizacija. U to vrijeme, Cubase aplikacija postojala je samo za Apple Machintosh platformu.

2.5. Programska podrška bazirana na aplikacijama-domaćinima („host-based systems“)

Trenutno, kućna računala za opću primjenu dovoljno su snažna da se mogu nositi sa mnogim aspektima glazbene sinteze i obrade signala. Više-manje sva današnja IBM PC kompatibilna računala imaju u sebi ugrađenu mogućnost reprodukcije audio signala visoke kvalitete kao standard, u obliku PCI kartica ili integriranih kartica na matičnoj ploči.

Razvoj softvera za takve strojeve se kontinuirano razvijao, te se danas standard za izradu računalne glazbe pomoću kućnih računala pronalazi u sustavima aplikacija-domaćina.

Računalni sintetizatori, kao npr. VAZ, su bile samostalne aplikacije koje su se morale pokretati usporedo sa softverom za sekvenciranje. Takav postupak predstavljao je prezauzeće za procesor, pošto je istovremeno morao pokretati dvije odvojene aplikacije, koje nisu dijelile ikakve resurse. Probitak u računalno temeljenom audio softveru stvoren je mogućnošću pokretanja spomenutih aplikacija unutar softvera za sekvenciranje.

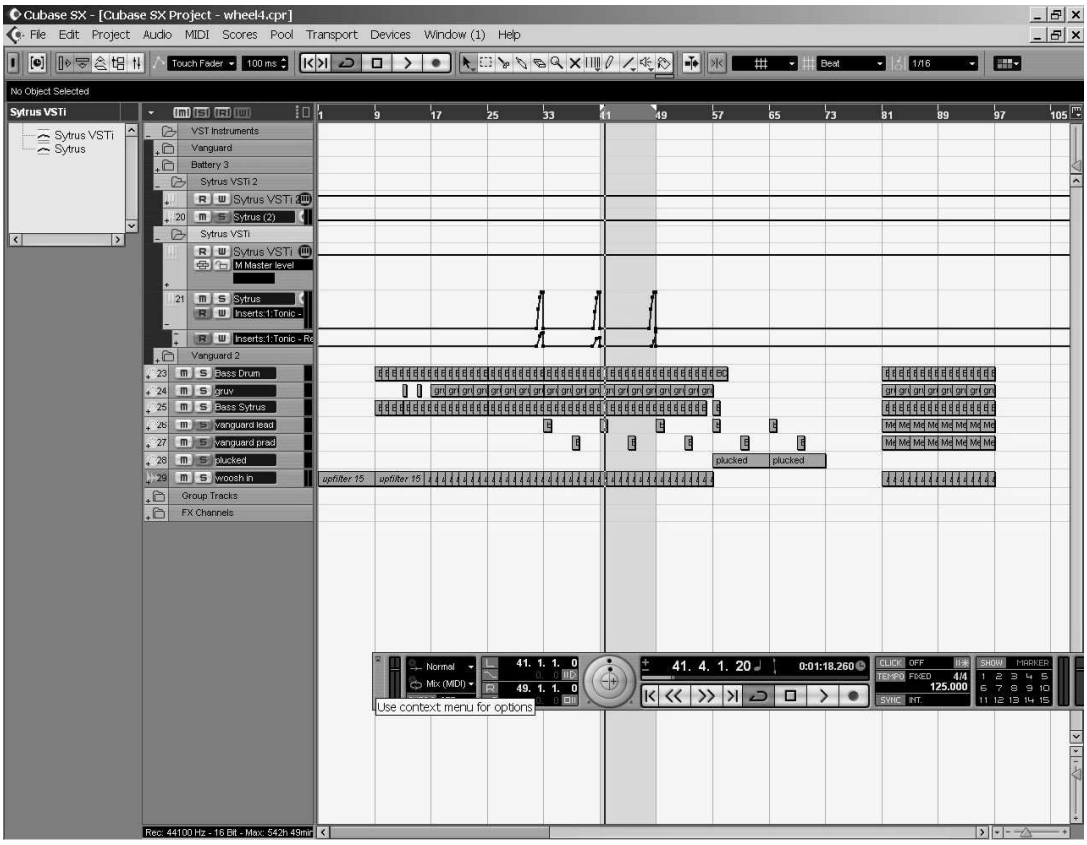
Sustav se postavlja instalacijom jedne od mnogih *host* aplikacija dostupnih na tržištu danas. Te aplikacije, svojim dizajnom, nalikuju MIDI i audio sekvencerima 90-ih godina (slika 5.); ali pružaju korisniku mogućnost da

instalira dodatke (engl. *plugin*) , razvijene od strane drugih kompanija i programera, kojima mogu povećati mogućnosti domaćina.

Mnogi dostupni dodaci su modelirani prema tradicionalnim analognim efektima kao što su ekvilizatori, vokoderi itd.; dok mnogi dodaci uvode nove digitalne efekte kao što su mijenjanje frekvencijske visine zvuka (engl. pitch-shifting).

Softverski sintetizatori također bivaju prilagođeni na sustave s aplikacijama-domaćinima. Kompanija Native Instruments izdala je FM7, softversku verziju Yamahinog DX-7 FM sintetizatora, koji dijeli svoj generator zvuka sa Yamahinim OPL chipovima opisanim u poglavlju 2.3.1.

Kao što se može vidjeti, mnogi dodaci nude simulaciju tradicionalnih hardverskih metoda koristeći softversku emulaciju; što spaja mogućnosti originalnih sustava sa pogodnostima integracije sa sekvencerima bez uvođenja potrebe za dodatnim hardverom.



Slika 5. Cubase SX audio aplikacija

3. Temeljni principi digitalne audio sinteze

U ovom poglavlju bit će dan pregled temeljnih principa koji tvore digitalni sintetizator glazbenog signala. Sve navedene metode moguće je implementirati koristeći VST razvojno sučelje te su one obilato korištene u većini današnjih komercijalnih proizvoda.

3.1. Koncept generatorskih jedinica

Jedna od najznačajnijih značajki u razvoju jezika za sintezu digitalnog zvuka je koncept generatorskih jedinica (UG, engl. Unit Generator). Generatorske jedinice su DSP moduli kao što su oscilatori, filtri, pojačala koji se mogu povezati u sintetičke instrumente, ili se povezuju unutar samog procesa podešavanja instrumenta, koji generiraju zvučne signale. Prvi sintetizatorski jezik koji je uporabio UG koncept je MUSIC III, a napisao ga je Mathews i njegov kolega Joan Miller 1960. godine. MUSIC III dopušta korisnicima dizajniranje sintetizatorskih mreža pomoću UG modula.

Propuštajući audio signal kroz seriju generatorskih jedinica, moguće je relativno lako implementirati veliki broj algoritama za sintezu.

3.2. Digitalni oscilator

Digitalni oscilator je fundamentalna jedinica sintetiziranja. Kao što je poznato, digitalni audio signal se sastoji od niza brojeva koji predstavljaju uzorke valnog oblika, koje je moguće čuti nakon prolaska tog signala kroz D/A konverter koji pretvara vrijednosti uzoraka u kontinuirani varijabilni napon koji se može pojačati te spojiti na zvučnik.

Prema toj činjenici, bilo bi moguće realizirati oscilator tako da računalo izračunava vrijednosti uzoraka prema matematičkoj formuli, te ih uzastopno šalje na D/A konverter. Takav proces funkcionira dobro, ali nije vrlo efikasan za digitalnu sintezu, posebice u realnom vremenu.

Mnogo efikasnija tehnika je tzv. *potraživanje u valnoj tablici* koja koristi unaprijed izračunat te spremljen niz vrijednosti uzoraka koji čine jednu periodu željenog valnog oblika.

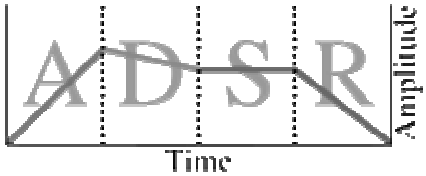
Nakon toga se pomoću sistemske frekvencije uzorkovanja te željene izlazne frekvencije oscilatora i broja uzoraka spremljenih u valnoj tablici određuje koji uzorci iz niza će biti upotrebljeni na izlazu. Detaljniji opis ovog principa demonstriran na primjeru nalazi se opisan u poglavlju 5.2.1.

3.3. Generator anvelope

Ako izlazne vrijednosti oscilatora množimo sa konstantom (npr. 1.0), onda će izlazni signal imati konstantu amplitudu preko cjelokupnog događaja (gdje je događaj, naprimjer, trenutak kad je nota na klavijaturi stisnuta i vrijeme dok je pritisnuta). Protivno tome, uhu zanimljiviji zvukovi imaju vremenski promjenjivu amplitudnu anvelopu. Tipično, nota počinje sa amplitudom 0, te raste do određene maksimalne vrijednosti (najčešće normalizirane na 1.0), te nakon toga brže ili sporije pada natrag na nulu. Početni dio anvelope naziva se *attack*, dok se krajnji dio anvelope naziva *release.*

Komercijalni analogni sintetizatori definirali su amplitudne anvelope pomoću četiri stadija: *attack, decay, sustain* (period, koji zavisi, na primjer, o vremenu dok je tipka na klavijaturi pritisnuta) i *release.* Uobičajena kratica za takvu anvelopu sa četiri stadija je ADSR (Slika 6.). ADSR koncept je koristan za verbalno opisivanje cjelokupne forme anvelope, na primjer: „malo pooštri attack“. Ali za specificiranje glazbene anvelope, ograničenje na linearnost i na četiri stadija je pomalo anakronistično. Oblikovanje anvelope je delikatna operacija, te zato fleksibilniji alati omogućuju muzičarima da stvaraju proizvoljne krivulje.

Na slici prikazana linearna krivulja se često nadomješta eksponencijalnom množeći odziv oscilatora odzivom jednopolnog niskopropusnog filtra na step funkciju.



Slika 6. graf jednostavne ADSR amplitudne anvelope

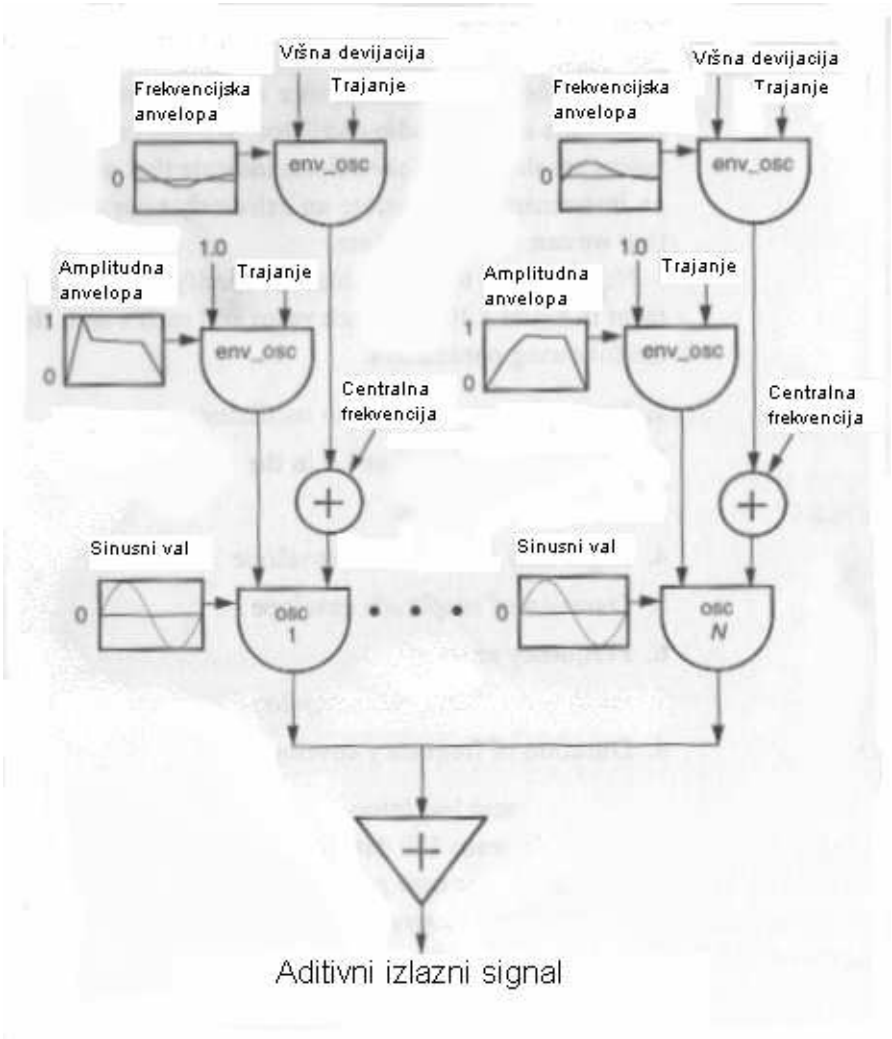
3.4. Aditivna sinteza

Aditivna sinteza je kategorija tehnika za sintezu zvuka temeljenih na sumaciji elementarnih valnih oblika u svrnu kreiranja složenijih signala. Aditivna sinteza je jedna od najstarijih i najviše istraživanih tehnika za sintezu.

Slika 7. predstavlja digitalni sintetizator za aditivnu sintezu. Instrument uključuje frekvencijsku anvelopu kao i amplitudnu anvelopu za svaki oscilator. Frekvencijska anvelopa je vremenski promjenjiva funkcija sa rasponom u intervalu [-1.0, 1.0]. Ova anvelopa vrednuje vrijednost *vršne devijacije* specificirane kao jedan ulaz generatora anvelope (*env_osc*). Uzmimo, za primjer, da vršna devijacija iznosi 100 i da frekvencijska anvelopa ima svoju minimalnu vrijednost -0.1. Vrijednost na izlazu generatora anvelope u tom trenutku iznosi -10. Sumator nakon toga zbraja tu vrijednost sa centralnom frekvencijom donjeg oscilatora, što rezultira padom izlazne frekvencije iz njene središnje točke. Ako bi središnja frekvencija bila postavljena na 440 Hz, frekvencijska anvelopa bi prouzročila pad na 430 Hz u određenom trenutku.

Iz slike se može primijetiti da svaki vertikalni odsječak uključuje dva generatora anvelope i audio oscilator. Ova jedinica se naziva *glas.* Samo dva glasa su prikazana, ali naznačuje se prisustvo još glasova. Takav instrument može generirati ekstremno širok raspon zvukova – pod uvjetom da možemo specificirati podatke. Za svaki glas ovog instrumenta potrebno je specificirati 8 parametara (centralna frekvencija, vršna devijacija, amplitudna anvelopa, početno vrijeme i trajanje amplitudne anvelope, frekvencijska anvelopa, početno vrijeme i trajanje frekvencijske anvelope). Ako bi pretpostavili da se

instrument sastoji od 15 glasova, za jednu postavku sintetizatora bilo bi potrebno namjestiti 120 parametara (!!) što predstavlja *problem specificiranja kontrolnih podataka.*



Slika 7. Blok shema pojednostavljenog instrumenta za aditivnu sintezu [4]

3.5. Subtraktivna sinteza

Subtraktivna sinteza je metoda oduzimanja harmoničkog sadržaja, karakterizirana primjenom filtra na audio signalu. Na primjer, dodavajući niskopropusni filtar na generator pilastog signala dobije se prirodnija aproksimacija gudačkih žičanih instrumenata nego koristeći samo generator signala.

Jednostavan primjer koji može poslužiti za razumijevanje principa subtraktivne sinteze je promatranje ljudskog glasa kao *sintetizatora.* Kad govorimo, pjevamo ili proizvodimo ostale zvukove, naše glasnice se ponašaju kao oscilator, dok usna šupljina i grlo preuzimaju ulogu filtra. Primijetimo razliku između pjevanja „uuuu“ i „aaaa“ na istoj visini tona. Glasnice generiraju, više ili manje, jednaki, harmonički bogat, zvuk u oba slučaja. Razlika dolazi iz filtriranja koje primjenjujemo sa ustima i grlom. Mijenjajući oblik usne šupljine mijenjamo *graničnu frekvenciju* te time oduzimamo neke harmonike. Polako mijenjajući zvuk iz „uuuu“ u „aaaa“ te nazad, dobijemo efekt „metućeg filtra“ (engl. *sweeping filter)* koji se široko upotrebljava u elektronskoj glazbi i koji je temelj „wahwah“ gitarskog efekta (koji je dobio ime po sličnosti sa ovim „vokalnim“ filtrom.)

3.5.1. Vremenski promjenjiva subtraktivna sinteza

Filtri mogu biti fiksni ili vremenski promjenjivi. Unutar fiksnog filtra, sva njegova svojstva su predodređena i ne mijenjaju se tokom vremena. Ova situacija je tipična za konvencionalno snimanje glazbe kada ton majstor postavlja ekvilizaciju svakog kanala na početku dionice.

Vremenski promjenjivi filtri imaju mnoge muzičke primjene, posebice u elektroničkoj i računalnoj glazbi gdje je cilj prevazići ograničenja tradicionalnih instrumenata. Pojasno propusni filtar čiji je faktor dobrote (Q), središnja frekvencija i prigušenje mijenjaju tokom vremena može definirati veliku raznolikost koloracija zvuka, posebice ako je i signal na koji se djeluje filtrom vremenski promjenjiv. Primjer vremenski-promjenjivog filtra može se naći u ekvilizacijskoj sekciji miješačke konzole. Ton majstor može mijenjati Q, centralnu frekvenciju te vrijednost pojačanja ili gušenja u bilo koje vrijeme tijekom procesa miješanja ili se ti parametri mogu programirati da se mijenjaju automatski.

3.6. FM sinteza

FM sinteza je forma u kojoj se timbar jednostavnog valnog oblika mijenja frekvencijskom modulacijom sa modulacijskim signalom čija je frekvencija

također unutar čujnog područja; što rezultira nastajanjem kompleksnih valnih oblika. Vrijednost FM signala u vremenu, $u_{FM}(t)$, određuje se pomoću formule:

$$u_{FM}(t) = A\sin\left[C_t + (I * \sin(M_t))\right] \quad (1)$$

Gdje su:

A – amplituda nosioca

I – indeks modulacije

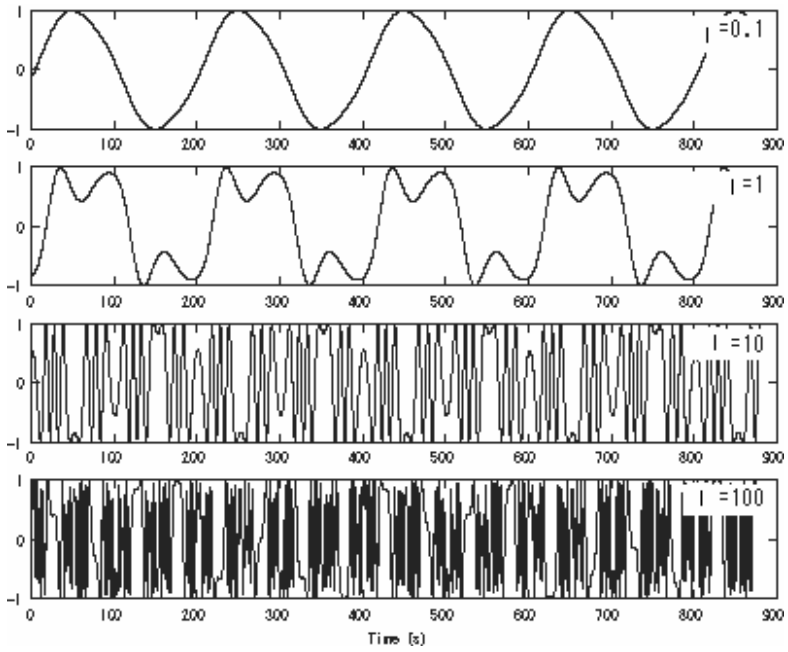
C_t, M_t – frekvencije nosioca, odnosno modulacijskog signala

Iz izraza (1) vidi se da je jednostavni FM veoma efikasan za softversku implementaciju, jer je zahtjeva samo dvije operacije množenja, jedno zbrajanje te dva potraživanja u valnoj tablici. Potraživanje u valnoj tablici referencira sinusne valove spremljene u memoriji. Zbog ove prednosti, FM sintezu je puno lakše izvesti upotrebom digitalnog sklopovlja ili računalnog jezika.

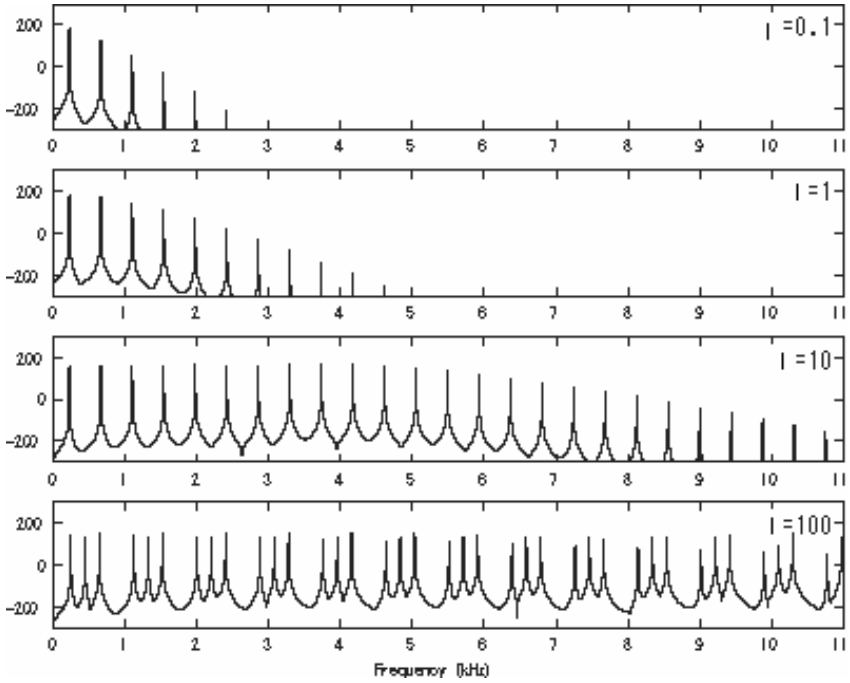
Za sintetiziranje harmoničkih zvukova, modulacijski signal treba imati harmonični odnos sa nosiocem. Kako količina frekvencijske modulacije raste, tako zvuk progresivno postaje kompleksniji. Korištenjem modulatora sa frekvencijama koje nisu cjelobrojni višekratnik frekvencije nosioca, mogu se dobiti različiti perkusivni, te razno razni „zvonki“ zvukovi.

Tehniku FM sinteze otkrio je John Chowning na Stanfordskom sveučilištu 1967/68. godine, a patentirala ju je Yamaha 1975. godine. Kao posljedica, FM sinteza je bila jedna od temeljnih tehnika ranih digitalnih sintetizatora iz Yamahe, sa njezinim vodećim DX7 sintetizatorom koji je bio sveprisutan tokom 80-ih godina. Casio je razvio sličnu metodu sinteze (sa suptilnim razlikama) sa nazivom sinteza distorzijom faze (engl. *phase distorsion synthesis*) koju je koristila u svojoj CZ seriji sintetizatora.

Sa istekom FM patenta Standfordskog sveučilišta 1995. g. FM sinteza je postala uobičajeni dio repertoara modernih sintetizatora; najčešće u kombinaciji sa ostalim tehnikama kao što su aditivna ili subtraktivna.



Slika 8. prikaz frekvencijski moduliranog signala sa različitim indeksom modulacije u vremenskoj domeni



Slika 9. prikaz frekvencijski moduliranog signala sa različitim indeksom modulacije u frekvencijskoj domeni

3.7. Granularna sinteza

Granularna sinteza je temeljni princip audio sinteze koji svoju operaciju vrši na razini *mikrozvuka.* Mikrozvuk uključuje svaki zvuk kraći od glazbene note (glazbenog objekta), a dulji od trajanja jednog uzorka signala. Često je temeljno na sličnim principima kao *sampling* (metoda sinteze koristeći isječke zvukova) te se često koristi u kombinaciji sa analognom opremom.

Rezultirajući signal nije jedan ton, već cijeli *zvučni pejzaž*, koji se može manipulirati na načine koji nisu slični nijednom prirodnom, pa čak i elektroničkom načinu manipulacije zvuka. Različiti zvukovi mogu biti postignuti variranjem valnog oblika, anvelope, prostornog položaja i gustoće granula.

Generirani zvukovi se mogu upotrebljavati u glazbi, zvučnim efektima ili kao sirovi materijal za daljnji proces sinteze ili DSP efekte. Obim efekata koji se mogu proizvesti uključuju amplitudnu modulaciju, vremensko rastezanje, stereo i multikanalno raspršenje, nasumično preuređivanje te mijenjanje oblika (*morphing*).

Granularna sinteza može se implementirati pomoću jednostavnog instrumenta: oscilatorom sinusnog vala kontroliranog sa generatorom anvelope. Takav instrument se lako može proširiti sa izborom među više valnih tablica.

Unatoč jednostavnosti instrumenta, za generiranje jednostavnog, nekompliciranog zvuka potrebna je velika količina kontrolnih podataka – do tisuću parametara po sekundi. Ti parametri opisuju svaku granulu: početno vrijeme, amplitudu itd. Pošto je nepoželjno specificirati tolike količine podataka ručno, potrebno je uvesti organizacijsku jedinicu više razine. Takva jedinica bi trebala automatski generirati tisuće pojedinih granularnih specifikacija. Postojeće metode za granularnu sintezu mogu se klasificirati, prema načinu organizacije granula, unutar pet tipova: 1. Fourierove i wavelet mreže, 2. preklapajući tokovi sinkroni sa visinom tona, 3. kvazi-sinkroni tokovi, 4. asinkroni oblaci, 5. vremenski granulirani tokovi ili tokovi isječaka sa preklapajućom, kvazi-sinkronom ili asinkronom reprodukcijom [4].

3.8. Sinteza prema fizikalnom modelu (PhM)

U području audio sinteze, PhM sinteza se odnosi na metode unutar kojih se generirani valni oblik signala izračunava koristeći matematički model, koji predstavlja skup jednadžbi i algoritama koji simuliraju fizički izvor zvuka, najčešće glazbeni instrument ili ljudski glas. Takav model sastoji se od (najčešće pojednostavljenih) zakona fizike koji upravljaju proizvodnjom zvuka, te tipično sadrži nekolicinu parametara od kojih su neki konstantni te opisuju karakteristike materijala i dimenzija instrumenata, a drugi su vremenski promjenjivi te opisuju glazbenikovu interakciju sa instrumentom (kao npr. okidanje žice, ili pokrivanje rupe na flauti).

Za moduliranje, naprimjer, bubnja, potrebna je formula koja opisuje kako udarac palice bubnja unosi energiju u dvodimenzionalnu membranu. Nakon toga, svojstva membrane (gustoća, tvrdoća...), njena sprega sa rezonancijom cilindričnog oblika bubnja, te rubni uvjeti opisuju njenu kretnju tokom vremena što za rezultat daje zvuk.

Slične faze u modeliranju se mogu naći i na primjeru violine, u čijem slučaju se pobuda energije vrši pomoću trenja između gudala i žica, ovisi o širini gudala, rezonantnim i prigušnim svojstvima žica, prijenosu vibracija kroz most, te konačno rezonanciji trbuščića.

Iako fizikalno modeliranje nije nov koncept u akustici i sintezi (implementacije su postojale metodama aproksimacija valne jednadžbe[1]), komercijalne implementacije nisu bile izvedive sve do razvoja Karplus-Strong algoritma, te njegovog unapređenja u efikasnu valovodnu sintezu, skup sa napredovanjem mogućnosti DSP tehnologija 80ih godina.

Prvi komercijalno dostupni sintetizator prema fizikalno modelu nastao je 1994. godine u Yamahi, a radio je na principu valovodne sinteze.

Popularniji virtualni instrumenti koji koriste PhM sintezu su Arturiin Brass, Modartov Pianoteq te AAS-ov String Studio.

[1] Hiller i Ruiz 1971.

4. Virtualna studijska tehnologija

Steinberg je značajna kompanija unutar glazbenih krugova, a njihova Cubase linija proizvoda predstavlja jedan od de-facto standarda za profesionalno sekvenciranje (MIDI ili audio podataka). Steinberg je jedan od prvih proizvođača, na Windows strani, koji je ponudio sučelje za proširenje ugrađenih mogućnosti sekvencera korištenjem dodataka.

VST dodaci se pokreću pretežito unutar *radne stanice za digitalni audio* (DAW) te pružaju *aplikaciji-domaćinu* (engl. host) dodatne mogućnosti. Većina VST dodataka može se kvalificirati kao instrument (VSTi) ili kao efekt. Postojeći proizvodi uglavnom pružaju svojstveno grafičko sučelje (GUI) koje vizualno nalikuje potenciometrima i prekidačima fizičkih uređaja. Ukoliko posebno grafičko sučelje nije izrađeno, aplikacija-domaćin pruža mogućnost generičkog korisničkog sučelja.

VST instrumenti na ulazu dobivaju MIDI podatke, a na izlazu generiraju digitalni audio signal; dok efekti dobivaju na ulazu audio podatke, a na izlazu daju obrađeni signal. Većina domaćina omogućava usmjeravanja izlaznih podataka jednog VST dodatka na ulaz drugog, što se naziva *ulančavanje* (engl. chaining). Primjerice: izlaz softverskog sintetizatora može se spojiti na flanger efekt.

Sa prikladnim hardverom i driverima, kao što su zvučne kartice koje podržavaju ASIO, VST dodaci se mogu upotrebljavati u realnom vremenu. ASIO zaobilazi Microsoftov sporiji audio engine, što pruža mogućnost niže latencije.

4.1. VST dodatak

Najadekvatniji opis VST dodatka, može se pronaći unutar Steinbergove VST SDK dokumentacije [5]:

„Esencijalno, VST dodatak je čista komponenta za obradu audio podataka, a ne audio aplikacija: to je komponenta koja se pokreće unutar

domaćinske aplikacije. Domaćinska aplikacija pruža tokove audio podataka koji bivaju obrađeni od strane dodatkovnog koda.

Općenito govoreći, VST dodatak može preuzeti podatkovni tok, primijeniti obradu na audio, te vratiti rezultat aplikaciji domaćinu. VST dodatak izvršava svoju normalnu obradu koristeći procesor računala; bez nužne potrebe za posvećenim procesorima za digitalnu obradu signala. Tok audio podataka biva izlomljen u seriju blokova. Domaćin sekvencirano pruža blokove. Domaćin i njegovo trenutno okruženje upravljaju veličinom bloka. VST dodatak održava status svojih parametara u odnosu na tekući proces obrade: domaćin ne sprema ikakve informacije o tome što je plugin napravio sa zadnjim blokom podataka koji je obradio.

Iz perspektive domaćina, VST dodatak je crna kutija sa odabranim brojem ulaza, izlaza (MIDI ili audio), i pridruženim parametrima. Domaćinu za uporabu dodatka nije potrebno implicitno znanje njegovog procesiranja. Dodatkovno procesiranje može koristiti, interno, bilo koji zaželjeni parametar, ali ovisno sa mogućnostima domaćina, može dozvoliti automatizaciju korisničkih parametara od strane domaćinske aplikacije.“

4.2. Prednosti virtualnog glazbenog instrumenta u odnosu na tehniku FM sinteze

U ovom poglavlju bit će dana usporedba između Yamahinog DX7 FM sintetizatora i njegovog softverskog emulatora: FM7 sintetizatora proizvedenog u kompaniji Native Instruments.

Prije svega, potrebno je iznijeti opće usporedne činjenice koje se odnose na korištenje hardverskih uređaja u odnosu na softverske sintetizatore. Korištenje programske podrške uvelike smanjuje potrebu za prostorom. Teoretski, virtualno je moguće pokrenuti beskonačno instrumenata u isto vrijeme. U realnosti je taj broj ograničen na procesorske mogućnosti računala. Današnjim računalom, primjerice, moguće je upogoniti više od deset profesionalnih instrumenata u isto vrijeme, bez prevelikog procesorskog opterećenja i sa zadržavanjem mogućnosti sinteze u realnom

vremenu. Ta mogućnost uvelike smanjuje troškove skupog studijskog prostora; kao i troškove prijevoza i postavljanja opreme pri izvedbama uživo ili turnejama.

Druga velika prednost, koja se odnosi na analogne komponente, jeste unošenje šuma. Analogne komponente, neizbježno unose šum u signal (kao što je, na primjer, termički) dok digitalne komponente pri obradi signala ne unose novi šum.

Suprotno ovim argumentima, mnogi i dalje preferiraju fizičke uređaje iz više razloga. Uz subjektivne razloge, kao što su: tvrdnja da softverski uređaji nemogu dati potrebnu „toplinu“ samom zvuku, te nedostatak „osjećaja“ prilikom podešavanja uređaja koristeći prave potenciometre i kliznike (u odnosu na podešavanje klikovima miša); glavni razlozi preferiranja hardvera ostaju: nestabilnost programske podrške (zamislimo što bi se dogodilo da se izvođaču, usred „žive svirke“ pred desecima tisuća ljudi, zablokira operativni sustav ili domaćinska aplikacija ili nešto treće) te dugotrajnost (softverske proizvode potrebno je stalno nadograđivati da bi bili kompatibilni sa novijim izdanjima operativnih sustava ili dodatkovnih standarda, dok jedan hardverski sintetizator može trajati više desetaka godina sa jednakom funkcionalnošću).

DX7 sintetizator (Slika 10.), temeljen na principu FM sinteze, proizveden je 1983. godine i bio je prvi komercijalno uspješni digitalni sintetizator zbog svoje preciznosti te fleksibilnosti digitalnog zvuka.

Slika 10. DX7 sintetizator

FM7 sintetizator (Slika 11.), proizveden 2001. godine, predstavlja oživljavanje DX7 sintetizatora sa dodanim mogućnostima. Razvoj FM7

sintetizatora prestao je 2006. godine kada je prešao u verziju FM8 koja uključuje neke ozbiljne preinake. FM7 sintetizator funkcionira kao samostalna aplikacija ili kao dodatak unutar VST, DirectX ili DirectConnect domaćinskih aplikacija.

Slika 11. FM7 sintetizator

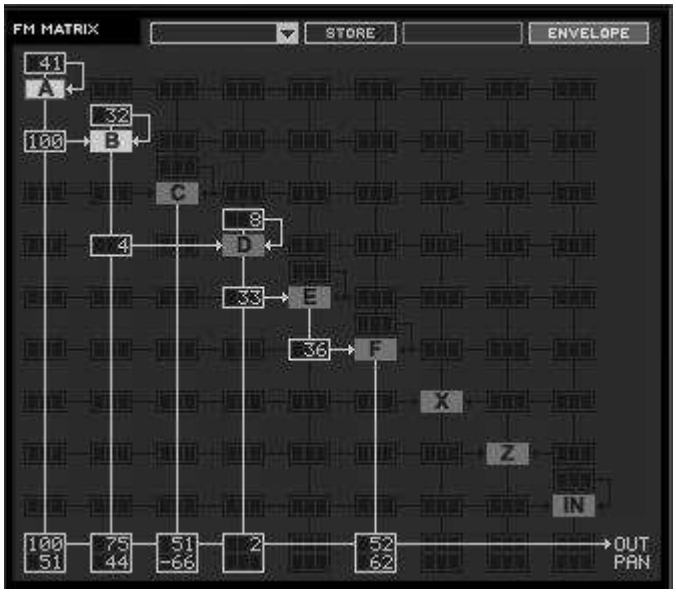
DX7 sintetizator se sastoji od 6 operatora. Operator predstavlja oscilator koji može biti korišten kao modulacijski ili kao generator signala nosioca. FM7 proširuje broj operatora na 8 (Slika 12.). Važno proširenje u odnosu na DX7 sintetizator nalazi se i u broju valnih oblika koje generiraju operatori. Dok DX7 podržava samo sinusni valni oblik, FM7 podržava 32 valna oblika uključujući formativne valne oblike te TX valne oblike.

FM7 sučelje je puno pristupačnije za programiranje zvukova, sadržava FM programibilnu matricu (Slika 12.), sučelje pomoću kojeg se povezuju oscilatori te odabire njihova funkcija (modulator/nosioc); dok DX7 u sebi sadrži 32 gotova algoritma za povezivanje operatora. Također, u FM7 moguće je učitati 128 različitih sačuvanih programiranih zvukova (engl.

preset) u isto vrijeme, dok je u DX7 moguće učitati samo 32 (FM7 podržava importiranje DX postavki u obliku sysex formata).

Uz navedeno valja napomenuti da FM7 ima ugrađene efekte koje je moguće dodati na isprogramirani zvuk.

Gledano iz perspektive kvalitete zvuka, mišljenja se razlikuju. Činjenica je da DX7 unosi šum neovisan o vještini programiranja. Dio kompozitora tvrdi da DX7 daje više „udarca“ (engl. punch), teksture i čistoće zvuka; te da zvuči „metalično“ dok softverska izvedba zvuči „plastično“. Suprotno tome, zastupana je i tvrdnja da su razlike u kvaliteti suptilne te da nisu vrijedne zamaranja s hardverom (teže programiranje, potreba za dodavanjem vanjskih efekata itd...).



Slika 12. FM matrica

4.3. VST razvojno sučelje

Prilikom razvijanja VST dodatka (instrumenta ili efekta), prvi korak je pravilno postavljanje VST razvojnog sučelja (VST SDK). Pošto prodaja Steinbergovih softverskih proizvoda, a time i profit, ovisi o dostupnosti i raznovrsnosti dodataka; kako bi potaknuli programere na razvoj aplikacija, SDK se može besplatno[2] preuzeti sa Steinbergovih web stranica.

Nedavno izašla verzija sučelja je 3, međutim ona predstavlja veliku preinaku unutar VST standarda (koji je podržan u najnovijoj verziji Cubase aplikacije); pa mnogi developeri i dalje ostaju pri standardu 2.x za koji postoji bezbroj domaćinskih aplikacija i dodataka (od kojih je veliki broj besplatan te sa otvorenim izvornim kodom). Ovo poglavlje osvrnut će se na temeljne funkcionalnosti i mogućnosti koje pruža posljednja 2.x verzija (2.4. sa drugom revizijom).

VST SDK 2.4 se sastoji od:

- alata za Windows i Machintosh platformu
- dokumentacije [5]
- zaglavnih datoteka (s ekstenzijom .h) te datoteka sa programskim kodom (.cpp) koji sadrže bazne klase potrebne za kreiranje dinamički povezanih biblioteka
- primjera izvornih kodova jednostavnih dodataka za različita compilerska okruženja koji mogu poslužiti kao polazišna točka za razumijevanje temeljnih principa razvoja novih projekata

Uz navedeno, u SDK su pridružene datoteke i dokumentacija *open-source* projekta VSTGUI (stabilne 3.0 grane) koje omogućuju lakšu implementaciju grafičkog sučelja uz zadržavanje platformske nezavisnosti.

[2] Naravno, uz pristajanje na licenčni ugovor.

4.3.1. Dinamički povezana biblioteka

VST dodatak dolazi u obliku dinamički povezane biblioteke (u Windows operativnom sustavu takve datoteke su najčešće predstavljene ekstenzijom .dll) koje se unutar domaćinske aplikacije učitavaju u, prema kategoriji (instrument ili efekt) za to predviđeni prostor.

Dinamičko povezivanje omogućuje učitavanje bibliotečnih potprograma za *vrijeme izvršavanja* glavnog programa (u ovom slučaju to je VST kompatibilna domaćinska aplikacija). Dodatci, općenito, su česta primjena dinamičkog povezivanja; što je osobito korisno kada je potrebno učitanu biblioteku zamijeniti sa drugom bibliotekom koja sadrži slično sučelje. U ovom slučaju, to sučelje i potrebne implementacije potprograma (funkcija) određeni su VST standardom.

4.3.2. Najvažnije funkcije i njihovi pozivi

Slika 13. prikazuje dijagram poziva funkcija prilikom inicijalizacije dodatka unutar domaćinske aplikacije. Oznaka *:AudioMaster* predstavlja objekt domaćinske aplikacije, a oznaka *:AudioEffectX* predstavlja instancu dodatkovnog objekta.

AudioEffectX je naziv za apstraktnu baznu klasu koja sadrži deklaracije funkcija potrebnih za kreiranje VST 2.4 dodatka. Dodatkovni objekt se realizira nasljeđivanjem *AudioEffectX* klase te implementacijom njenih virtualnih funkcija.

Prva funkcija koju zove domaćin je konstruktor (`.AudioEffectx()`). Konstruktor prima sljedeće argumente: *AudioMasterCallBack* objekt, broj programa, te broj parametara. Nakon toga poziva se funkcija `.open()`, a poslije njenog izvršavanja poziva se `.setSampleRate()`.

Funkciju `setSampleRate()` domaćinska aplikacija zove na početku, te prilikom promjene operativne frekvencije uzorkovanja domaćina. Njezin prototip glasi:

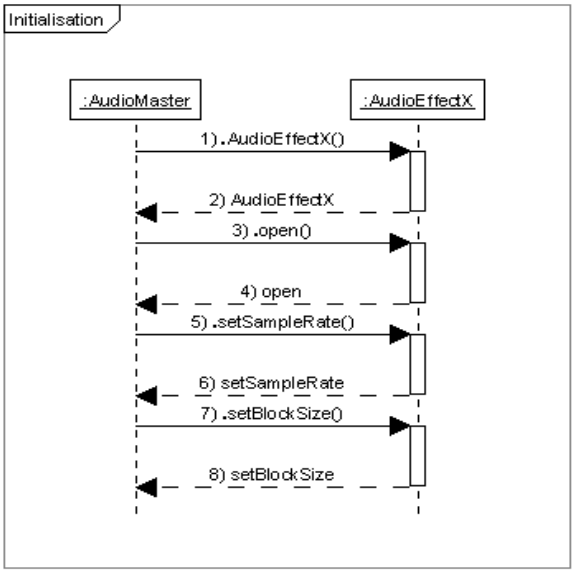
```
virtual void AudioEffect::setSampleRate(float sampleRate)
```

Ova funkcija poziva se samo u suspendiranom stanju dodatka, te sprema informaciju o trenutnoj frekvenciji uzorkovanja u varijablu *sampleRate.*

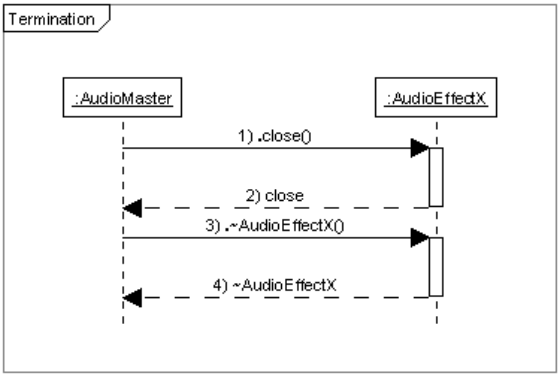
Nakon tog poziva, prilikom inicijalizacije, izvršava se poziv funkcije `.setBlockSize()`. Njezin prototip glasi:

```
void AudioEffect::setBlockSize(VstInt32 blockSize)
```

Ova funkcija se također poziva samo u suspendiranom stanju i vrlo je bitna jer određuje veličinu bloka audio podataka (uzoraka) koje funkcija domaćin šalje dodatku. Taj podatak je spremljen u varijabli *blockSize.*



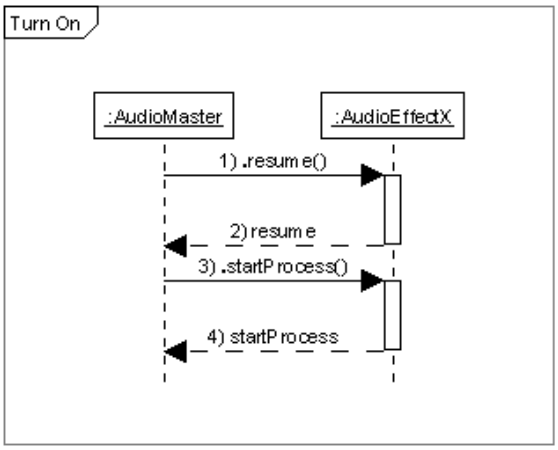
Slika 13. Inicijalizacija dodatka[5]



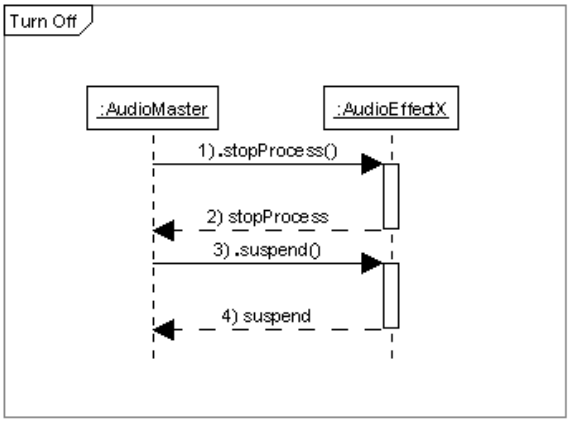
Slika 14. Terminacija dodatka [5]

Slika 14. prikazuje dijagram poziva funkcija prilikom terminacije dodatka: nakon `.close()` funkcije poziva se destruktor AudioEffectX objekta (`.~AudioEffectX()`). Unutar destruktora potrebno je svu alociranu memoriju dealocirati, te je uobičajena praksa postaviti pokazivače na nulu.

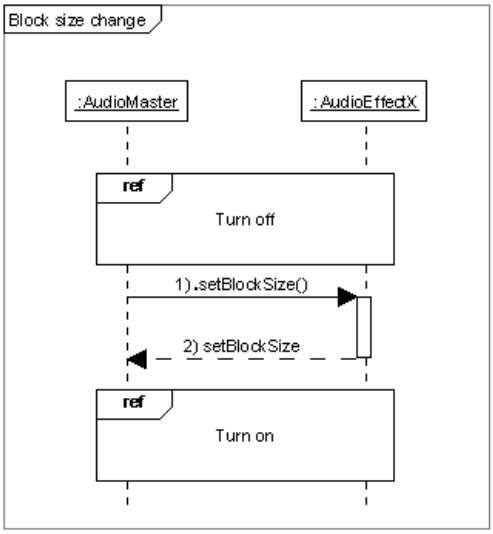
Na slikama 15. i 16. prikazani su dijagrami poziva funkcija prilikom paljenja i suspendiranja dodatka koji je već prethodno inicijaliziran. Prilikom tih radnji poziva se funkcija `.resume()` prethodno pozivu `.startProcess()`; odnosno `.stopProcess()` prethodno `.suspend()`. Procedure paljenja i suspendiranja pozivaju su prije, odnosno nakon promjene veličine bloka podataka ili promjene frekvencije uzorkovanja.



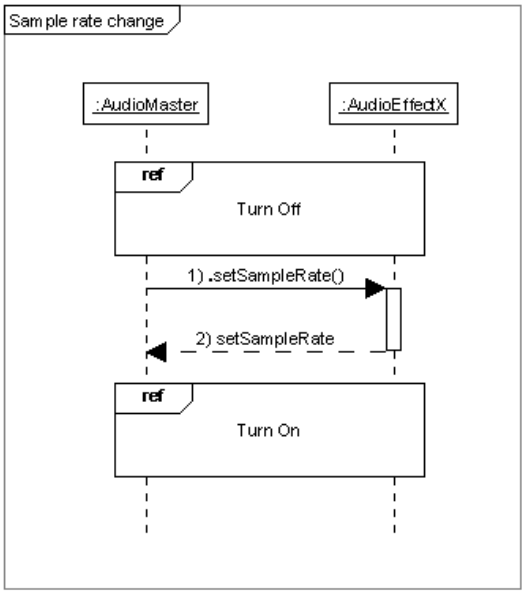
Slika 15. Paljenje dodatka [5]



Slika 16. Suspenzija dodatka [5]



Slika 17. Promjena veličine bloka [5]



Slika 18. Promjena frekvencije uzorkovanja [5]

4.3.3. Funkcija za obradu bloka podataka

Najvažnija funkcija unutar koje se vrše algoritmi za obradu ili sintezu signala je `processReplacing()` funkcija. Njezin prototip glasi:

```
void AudioEffect::processReplacing (float** inputs,float** outputs, VstInt32 sampleFrames)
```

Funkcija prima tri parametra: pokazivač na polje unutar kojega su sadržani ulazni podaci, pokazivač na polje koje je predviđeno za izlazne podatke te cjelobrojni broj koji određuje broj uzoraka poslan funkciji. Ova funkcija je čisto virtualna što znači da je potrebno napraviti njezinu implementaciju. Vrijednosti uzoraka signala u VST-u prikazani su pomoću realnih varijabli tipa *float* sa 32 bitnom preciznošću; unutar intervala [-1.0, 1.0].

Sljedeći primjer programskog koda prikazuje funkciju obrade podataka dodatka koji nema nikakvog efekta već samo preusmjerava ulazne podatke na izlaz:

```
void NeRadiNista::processReplacing(float **inputs, float **outputs,
long sampleFrames)
{
     float *ulaz1  =  inputs[0];
     float *ulaz2  =  inputs[1];
     float *izlaz1 = outputs[0];
     float *izlaz2 = outputs[1];
     while(--sampleFrames >= 0)
     {
       (*izlaz1++) = (*ulaz1++) ;
       (*izlaz2++) = (*ulaz2++) ;
     }
}
```

Varijable *ulaz1* i *ulaz2* izjednačavaju se sa ulaznim poljima podataka lijevog i desnog kanala (varijable *inputs[0]* i *inputs[1]*); dok se varijable *izlaz1* i *izlaz2* izjednačavaju sa izlaznim poljima podataka (*outputs[0]* i *outputs[1]*).

While petlja smanjuje *sampleFrames* varijablu, koristeći - - prefix operator, dok ona ne dostigne vrijednost 0. Pri svakom izvršavanju petlje, kopira se vrijednost ulaza na izlaz te se pomiče pokazivač na idući uzorak.

Nakon što *sampleFrames* dostigne 0, polje podataka je obrađeno, i funkcija završava. Nakon toga cijeli proces počinje ispočetka sa novim baferom audio podataka.

Da smo, na primjer, unutar *while* petlje izjednačavali izlazne varijable sa vrijednošću 0.5 na izlazu bi dobili DC signal vrijednosti -6dB.

U slučaju da je VST dodatak sintetizator, a ne efekt, podaci se ne šalju pomoću ***inputs* bafera (pošto ulazni podaci nisu uzorci audio signala već MIDI podaci - događaji) već se oni pridobivaju pomoću funkcije za obradu MIDI događaja.

4.3.4. Funkcija za obradu MIDI događaja

Ova funkcija koristi koristi se unutar dodataka koji su sintetizatori, a ostvaruje obradu MIDI podataka koje domaćinska aplikacija šalje dodatku. Njezin prototip glasi:

```
VstInt32 AudioEffectX::processEvents(VstEvents*  events)
```

Funkcija se poziva kad stignu novi događaji (npr. MIDI podaci koji označuju da je nota bila pritisnuta/otpuštena, ili kontinuirani kontroler (CC) pomaknut). Ova metoda je članica klase *AudioEffectX* koja predstavlja proširenje u odnosu na VST 1.0 baznu klasu *AudioEffect*, nastalo u verzijama 2.x kada je real-time sinteza implementirana u VST sučelje.

U VST-u, MIDI događaji su podskup veće vrste događaja poznate kao *VST event*. Ti događaji su predstavljeni u obliku strukture tipa *VstEvents*. Uz MIDI događaj, audio, video i okidački događaji su također podržani. Sintetizatori ili drugi instrumenti moraju signalizirati da potražuju VST događaje. To se realizira pozivanjem funkcije `isSynth()` unutar dodatkovnog konstruktora; te implementacijom funkcije `canDo()` unutar koje se treba naznačiti koje vrste događaja prima dodatak. Slijedi primjer `canDo()` funkcije za sintetizator koji prima VST događaje:

```
VstInt32 VstXSynth::canDo (char* text)
{
      if (!strcmp (text, "receiveVstEvents"))
            return 1;
      if (!strcmp (text, "receiveVstMidiEvent"))
            return 1;
      if (!strcmp (text, "midiProgramNames"))
            return 1;
      return -1;
}
```

MIDI događaji identificiraju se (unutar strukture **events*) pomoću vrijednosti *kVstMidiType.* Nakon što je događaj identificiran kao MIDI, na strukturu se može primjeniti *cast* operator koji ju konvertira u tip strukture *VstMidiEvent.* Članovi te strukture daju informacije o vrsti note koja je pritisnuta/otpuštena, vremenske podatke te podatke o kontinuiranim kontrolerima. Nakon što programer spomenutom metodom dobavi bitove, može, koristeći MIDI specifikacije, odlučiti kako će s njima postupiti u daljnjoj obradi. Dobar primjer MIDI specifikacija može se pronaći na Internetu, naveden je u popisu literature pod brojem [6].

Primjer realizacije funkcije `processEvents()` može se pronaći u dodatku A ovog rada.

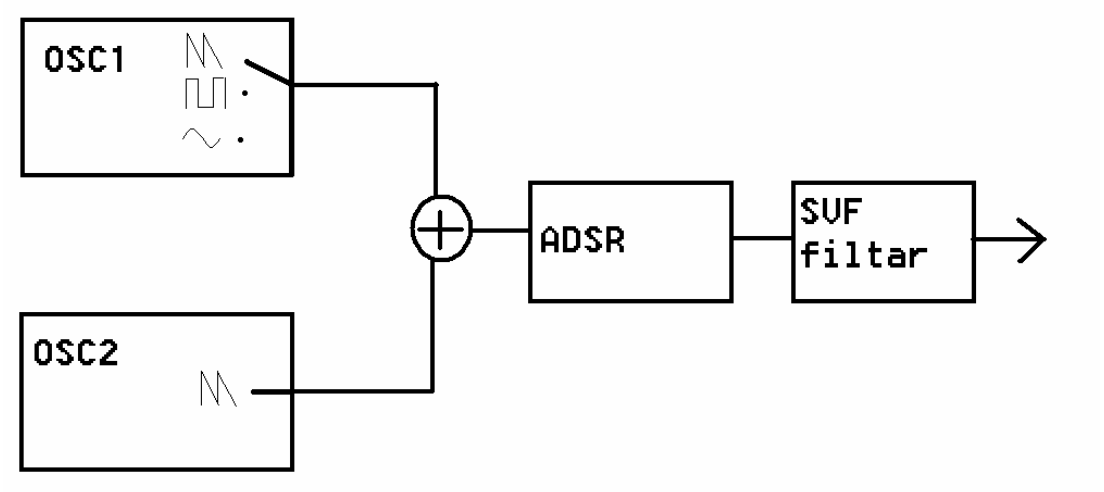
5. Sinthex – primjer realizacije virtualnog glazbenog instrumenta

Kao rezultat dosadašnjih proučavanja i razmatranja nastao je primjer realizacije VST sintetizatora, nazvan tvorbom složenice modificiranih engleskih riječi: *Synthesizer Example.* Razvijen je koristeći Microsoft Visual Studio 2005 IDE (integrirano razvojno okruženje) u kombinaciji sa VST SDK-om 2.4 uporabom C++ programskog jezika. Metoda korištena za sintezu mogla bi se nazvati kombinacijom aditivne i subtraktivne (signali iz oscilatora se zbrajaju, te se na kraju pri izlazu primjenjuje filtar).

Sintetizator je razvijen modularno, svaka generatorska jedinica (oscilator, anvelopa, polifonija itd.) predstavljena je zasebnim sučeljem (dijelom programa koji sadrži deklaraciju klase i pripadajućih funkcija, te se navodi u datoteci s ekstenzijom *.h*) te pripadajućom implementacijom (sam kod funkcija koje pripadaju određenoj klasi, koji se navodi u datoteci s ekstenzijom *.cpp*). Kao kostur za razvoj korišten je Steinbergov bazični primjer *vstxsynth* koji u sebi ne sadrži dodatne module i daje osnovni skup postavki za početak razvoja virtualnog instrumenta.

5.1. Blok shema

Slika 19. prikazuje Sinthexov blok dijagram generatorskih jedinica. Element *OSC1* je digitalni oscilator sa mogućnošću odabira tri različita valna oblika: pravokutni, pilasti te sinusoidalni. Oscilator radi pomoću metode potraživanja u valnoj tablici, te rabi linearnu interpolaciju za izbjegavanje greške prilikom izračunavanja vrijednosti uzorka. *OSC2* na izlazu daje samo pilasti signal, ali za razliku od prvog oscilatora, radi na principu generiranja pojasno-ograničenih valnih tablica za izbjegavanje *aliasinga* (engl. *band-limited waveforms*).



Slika 19. Blok shema Sinthex instrumenta

Nakon zbrajanja spomenutih signala, signal ulazi u četverostupanjski *generator anvelope* (element na slici označen oznakom *ADSR*). Generator anvelope koristeći princip odziva jednopolnog niskopropusnog filtra na step funkciju eksponencijalno oblikuje uzdizanje, odnosno spuštanje signala. Nakon dinamičkog oblikovanja signala, vrši se frekvencijska obrada *SVF* modulom (engl. *State Variable Filter*). SVF je, sa svojom velikom popularnošću u domeni analogne opreme i sintetizatora, brzo nakon početka razvoja DSP uređaja dobio svoju digitalnu reinkarnaciju, te je njegov algoritam primijenjen i na ovaj instrument.

5.2. Realizacija oscilatora

5.2.1. Oscilator s potraživanjem u valnoj tablici

Iako bi, kao što je već prije spomenuto, bilo jednostavnije realizirati oscilator na način da računalo izračunava vrijednost signala za svaki uzorak, ta metoda je „preskupa“ za *real time* zahtjeve. Zato se najčešće pribjegava sljedećoj metodi: na disk ili u radnu memoriju se pohrani niz određenog broja uzoraka koji čine jednu periodu željenog valnog oblika; nakon toga se u ovisnosti o željenoj izlaznoj frekvenciji te frekvenciji uzorkovanja na koju je podešen DAW, valna tablica, moglo bi se reći, *reuzorkuje* (*resampleira*) u potrebnom omjeru.

Pretpostavimo da imamo periodu sinusnog signala spremljenu unutar 1000 uzoraka, te da je frekvencija uzorkovanja sustava f_s = 1kHz. Direktnim iščitavanjem uzoraka izlazna frekvencija signala iznosila bi f_s/1000 = 1 Hz. Kada bi za svaki izlazni uzorak iščitavali svaki drugi uzorak iz valne tablice, izlazna frekvencija iznosila bi f_s/ (1000/2) = 2Hz.

Dakle, oscilator preskače vrijednosti u tablici za *inkrement* dodan trenutnoj *faznoj lokaciji* u valnoj tablici kako bi postigao potrebnu frekvenciju. Time, osnovni algoritam za oscilator mogao bi se objasniti kao program u dva koraka:

1. *fazni_index* = mod_L(*prethodna faza* + *inkrement*)
2. *izlaz* = *amplituda* x *valna_tablica*[*fazni index*]

Prvi korak algoritma sastoji se od zbrajanja i modulo operacije (označene s mod_L). Modulo operacija dijeli sumu sa duljinom valne tablice L i zadržava samo ostatak, koji je uvijek manji ili jednak L. Drugi korak sadrži potraživanje u valnoj tablici i množenje. Ovo je relativno malo računarski zahtjevno, uz pretpostavku da su valne tablice već popunjene vrijednostima signala. Ako su duljina valne tablice i frekvencija uzorkovanja fiksirani, što je najčešći slučaj, onda frekvencija izlaznog signala oscilatora ovisi o vrijednosti inkrementa. Odnos između željene frekvencije i inkrementa dan je sljedećom jednadžbom:

$$inkrement = \frac{L \times f}{f_s} \tag{2}$$

Gdje su: *L* - duljina valne tablice u broju uzoraka (najčešće potencija broja dva), *f* - željena izlazna frekvencija oscilatora, f_s - frekvencija uzorkovanja. Na primjer ako su duljina tablice 1000, frekvencija uzorkovanja 40 kHz i željena izlazna frekvencija 2kHz, onda inkrement iznosi 50 što implicira sljedeću jednadžbu za frekvenciju:

$$f = \frac{inkrement \times f_s}{L} \tag{3}$$

U realiziranom primjeru koriste se tri različite valne tablice – po jedna za svaki valni oblik: sinusoida, pilasti i pravokutni. Svaka se sastoji od 4096 uzoraka pohranjenih u polju *float* brojeva tipa *static.* Static elementi su upotrebljeni jer su oni zajednički za sve objekte nekog tipa [7] te se time izbjegava potrošnja suvišne radne memorije prilikom kreiranja više instanci jednog te istog oscilatora (u ovom konkretnom slučaju koristi ih se 16, za svaki glas sintetizatora po jedan). Generiranje valnih tablica se vrši u konstruktoru objekta *C_osc1*. *Fazni indeks* oscilatora postavlja se na 0 u *noteOn* metodi unutar koje se također izračunava željena izlazna frekvencija. Ta metoda poziva se u trenutku kada je određena nota pritisnuta. Gore spomenuti algoritam za oscilator vrši se unutar *processAcc* metode uz dodatak linearne interpolacije za točniju reprodukciju valnog oblika.

Oscilator ima sljedeće kontrolne parametre u glavnom sučelju: razina, frekvencijski pomak u polutonovima te tip valnog oblika.

5.2.2. Linearna interpolacija

Za većinu vrijednosti duljina valnih tablica, frekvencija te frekvencija uzorkovanja u formuli (2) rezultirajući inkrement nije cjelobrojni, već realni broj sa frakcionalnim dijelom iza decimalne točke. Unatoč tome, način na koji potražujemo vrijednost unutar valne tablice je taj da je lociramo pomoću njenog indeksa koji je cijeli broj. Dakle, potrebno je nekako polučiti cjelobrojnu vrijednost iz realnog inkrementa.

Realna vrijednost može se *odsjeći* i time polučiti cijeli broj potreban za tablični indeks. To bi značilo jednostavno brisanje svih znamenki desno od decimalne točke, čime bi, primjerice, broj 6.99 postao 6.

Uzmimo, za pretpostavku, inkrement vrijednosti 1,125. Tablica 1 uspoređuje izračunate vrijednosti faznog indeksa usporedo sa odsječenim vrijednostima. Nepreciznosti uzrokovane odsijecanjem znači da dobivamo približne vrijednosti valnog oblika, ali ne točno one koje trebamo. Kao posljedica, male količine *distorzije* postaju prisutne u signalu. Ta vrst smetnje se naziva *šumom potraživanja u tablici.* Različita pomoćna sredstva mogu umanjiti taj šum. Veća valna tablica je jedna mogućnost. Drugi način bi bio

zaokruživanje vrijednosti faznog indeksa na najbližu vrijednost, te bi time 6.99 postao 7, što je mnogo točnije nego 6.

Ali daleko najefikasnije performanse postižu se *interpolirajućim oscilatorom*, što je možda računarski „skuplje“ ali generira čišće signale.

Izračunata vrijednost	Odsječena vrijednost
1.000	1
2.125	2
3.250	3
4.375	4
5.500	5
6.625	6
7.750	7
8.875	8
10.000	10
11.125	11
12.250	12
13.375	13
14.500	14
15.625	15
16.750	16
17.875	17
19.000	19

Tablica 1. Izračunate i odsječene vrijednosti faznog indeksa unutar valne tablice [4]

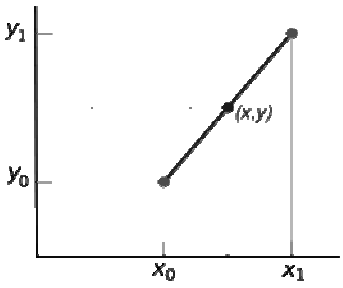
Metoda za smanjivanje šuma potraživanja korištena u ovom primjeru je linearna interpolacija. *Linearni interpolant* između dvije točke na određenom intervalu računa se prema:

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0} \tag{4}$$

Gdje oznake pripadaju koordinatama na slici 20. Primjenom izraza (4) na algoritam za digitalni oscilator izraz poprima sljedeći oblik:

$$y[n] = \text{tablica}[x_0] + (\mathit{fazni\_indeks} - x_0)(\text{tablica}[x_0 + 1] - \text{tablica}[x_0]) \tag{5}$$

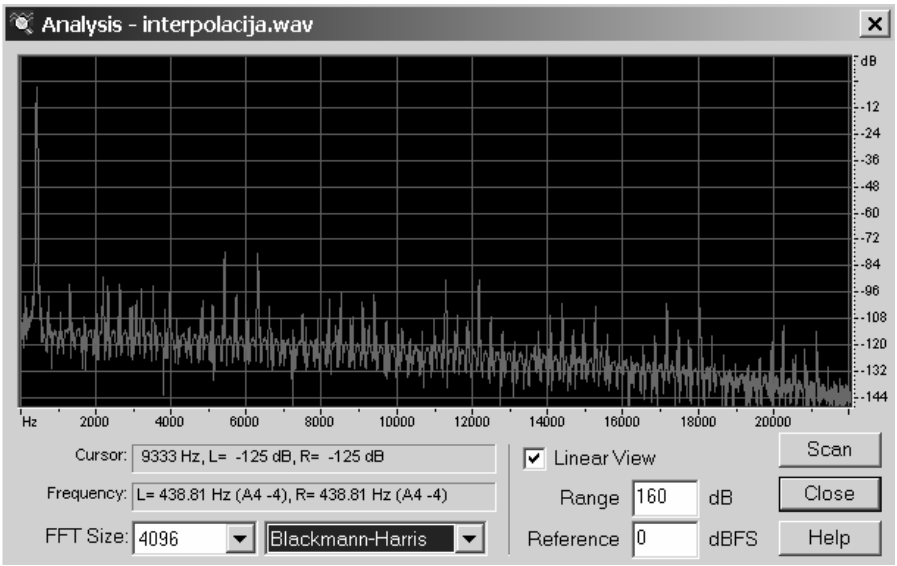
Gdje su: y[n] - vrijednost trenutnog izlaznog uzorka, x_0 - odsječena vrijednost faznog indeksa.



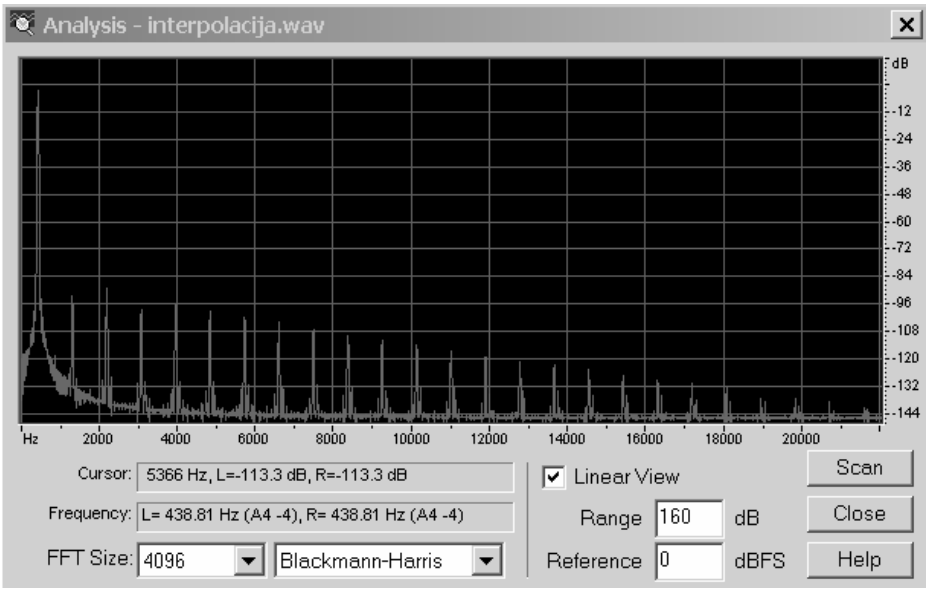
Slika 20. Linearna interpolacija

Izrazom (5) aproksimira se vrijednost signala između dvije točke unutar kojih pada trenutni fazni indeks.

Na slikama 21. i 22. može se primijetiti značajna razlika u količini šuma generiranog pomoću neinterpolirajućeg, odnosno interpolirajućeg oscilatora. Mjerenje je izvršeno FFT analizom unutar 4096 točaka nad sinusnim signalom frekvencije 440 Hz (što odgovara noti A4, odnosno noti a^1 u prvoj oktavi). Još bolji rezultati mogu se postići koristeći naprednije interpolacijske postupke (Hermitova krivulja, sinc interpolacija itd..) ali za potrebe audia u realnom vremenu najčešće se koristi linearna interpolacija zbog relativno „jeftine“ računarske zahtjevnosti i zadovoljavajuće kvalitete.



Slika 21. Frekvencijski prikaz sinusnog signala generiranog s neinterpolirajućim oscilatorom



Slika 22. Frekvencijski prikaz sinusnog signala generiranog s linearno interpolirajućim oscilatorom

5.2.3. Generator pilastog signala s ograničenom širinom pojasa

Element *OSC2* na slici 19. predstavlja oscilator pilastog signala sa ograničenom širinom pojasa u svrhu izbjegavanja *aliasinga*. Kod sintetiziranja analognih tipova valnih oblika (kao što su pilasti signal, pravokutni impuls ili trokutasti signal) u digitalnoj domeni posebna pozornost se mora posvetiti pojasnoj ograničenosti izlaznog signala. Jednostavna metoda generiranja pilastog signala može se opisati sljedećim formulama:

- Analogno

$$y(t) = A(tf_0 - Int(tf_0)) \tag{6}$$

- Digitalno

$$y[n] = A(\frac{nf_0}{f_s} - Int(\frac{nf_0}{f_s})) \tag{7}$$

Gdje su: y - izlaz, A - amplituda, t - vrijeme, n - broj uzorka u diskretnoj digitalnoj domeni, f_0 - frekvencija valnog oblika, f_s - frekvencija uzorkovanja, Int(x) – najveći cijeli broj manji ili jednak x.

Analogna jednadžba daje poželjne rezultate zbog rada u kontinuiranoj vremenskoj domeni dok digitalna vraća zvuk koji sadrži aliasing. U procesu analogno-digitalne konverzije signal prolazi kroz niskopropusni filtar prije nego što dolazi na sklop za uzorkovanje. Direktna sinteza u digitalnoj domeni ne provodi ovaj postupak, te time nastaje aliasing koji zagađuje cijeli spektar i nemoguće ga je isfiltrirati.

Pojasno ograničena valna tablica sadrži harmonike čije frekvencije su niže od Nyquistove. Frekvencije harmonika u valnoj tablici ovise o frekvenciji ponavljanja valne tablice. Broj harmonika *h* dozvoljenih za određenu valnu tablicu za frekvenciju f_0 može se odrediti iz izraza za Nyquistov uvjet maksimalne frekvencije, gdje je f_{max} najviši harmonik osnovne frekvencije f_0:

$$f_{\max} < f_s / 2 \tag{8}$$

$$h < \frac{f_s}{2 f_0} \tag{9}$$

Pomoću izraza (9) može se generirati niz valnih tablica sa različitim brojem harmonika. U primjeru realizacije Sinthex, implementirano je rješenje iz [8] koje koristi 128 valnih tablica, po jednu za svaku MIDI notu, sa odgovarajućim brojem harmonika. Tablice se sastoje od 4096 elemenata, uz korištenje linearne interpolacije.

Pilasti signal može se generirati koristeći razvoj u Fourierov red:

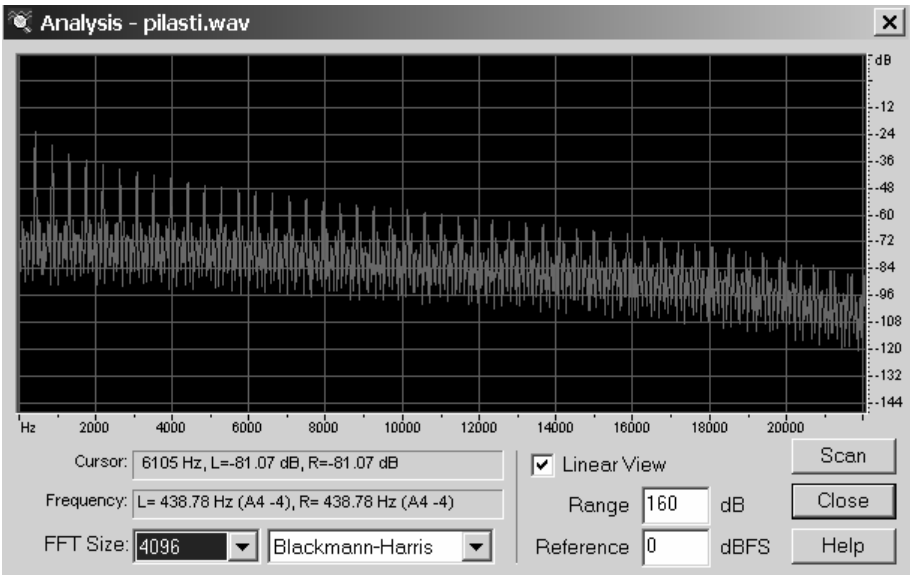
$$\sin(x) + \frac{1}{3}\sin(2x) + \frac{1}{3}\sin(3x) + \ldots \quad \text{za } 0 \le x \le \pi \tag{10}$$

Za svaku valnu tablicu, potrebno je sumirati red do člana $(1/h)\sin(hx)$. Prilikom generiranja signala pomoću Fourierovog reda, javlja se nadvišenje signala koje se naziva *Gibbsovim efektom.* Ono nastaje kao posljedica činjenice da korišteni red u slučaju pojasno ograničenih valnih tablica ne sadrži beskonačno članova. Za minimizaciju ovog efekta, potrebno je smanjiti amplitudu viših članova reda. U implementiranom rješenju, svaki član reda pomnožen je s koeficijentom *m* koji se može odrediti iz sljedećeg izraza:

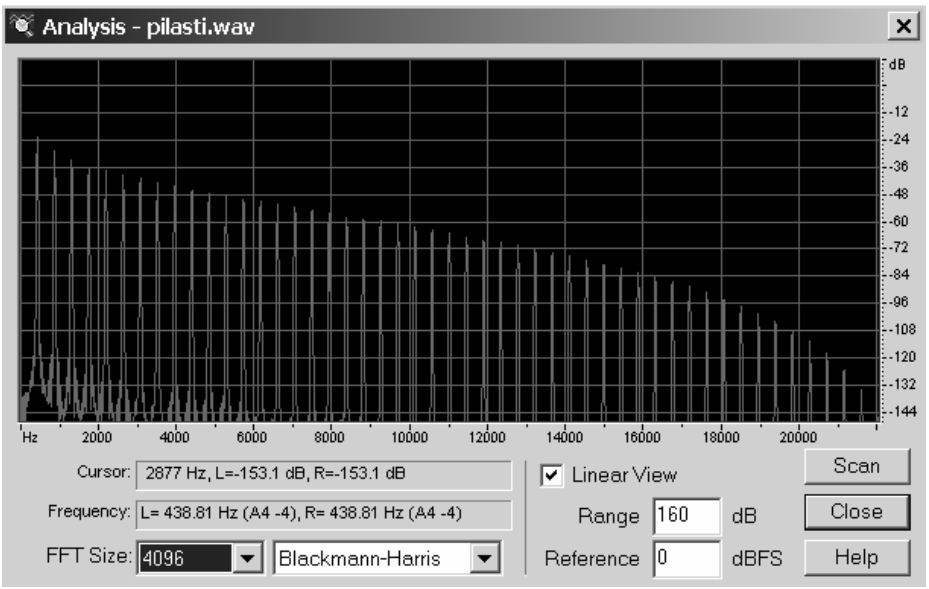
$$m = \cos^2((n-1)/k) \tag{11}$$

Gdje su: $k = (\pi/2)/h$, n=trenutni harmonik. Koristeći ovo rješenje, Fourierov red za određenu valnu tablicu može se zapisati u sljedećem obliku:

$$\sum_{n=1}^{h} \frac{1}{n} \sin(nx) \cos^2((n-1)/k) \qquad (12)$$



Slika 23. Frekvencijski prikaz pilastog signala na izlazu iz nelimitiranog oscilatora

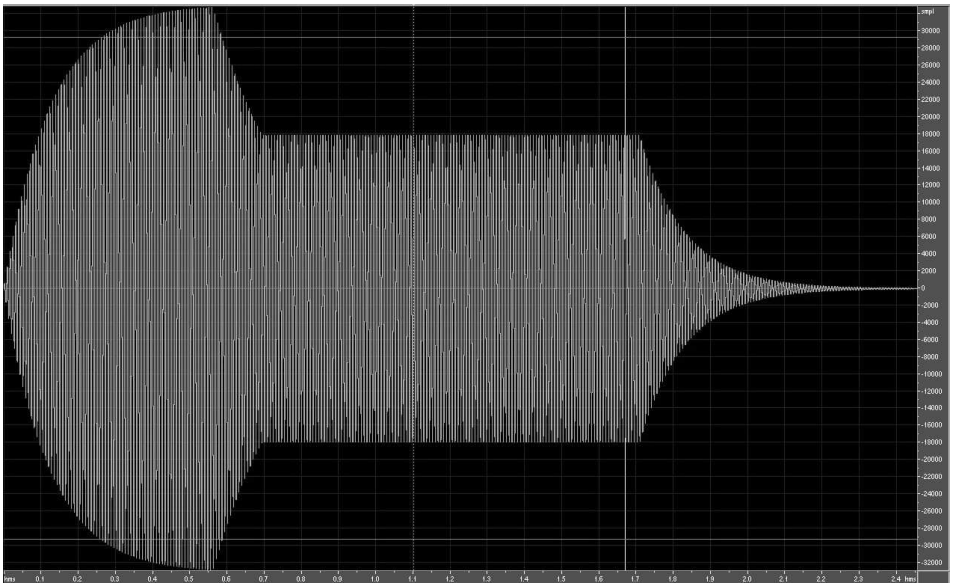


Slika 24. Frekvencijski prikaz pilastog signala na izlazu iz pojasno ograničenog oscilatora

Na slikama 23. i 24. nalazi se usporedba signala generiranih jednostavnim pilastim oscilatorom, odnosno oscilatorom sa pojasno ograničenim valnim tablicama u kojima se očituje povećana razina šuma u prvom slučaju.

5.3. Realizacija generatora anvelope

Element *ADSR* na slici 19. predstavlja četverostupanjski generator eksponencijalne anvelope (amplitudne ovojnice, vidi sliku 25.). Ovaj modul generira eksponencijalni oblik uzlaza amplitude (engl. *attack*) prilikom početka note, te eksponencijalnu silaznu putanju amplitude (engl. decay, *release*).

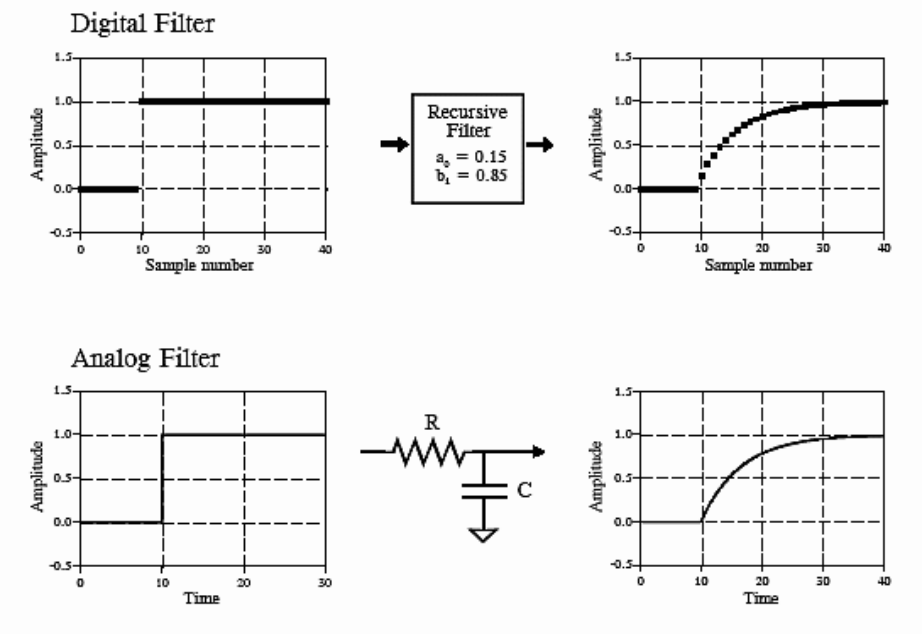


Slika 25. Izlazni signal oblikovan eksponencijalnom ovojnicom

Eksponencijalna ovojnica je najčešći izbor u dizajniranju sintetizatora jer zvuči najbliže prirodnim zvukovima, za razliku od npr. linearne. Ova vrst generatora radi na principu odziva jednopolnog niskopropusnog filtra na step funkciju. Rekurzivna jednadžba ovog filtra glasi:

$$y[n] = a_0x[n] + b_1y[n-1] \quad (13)$$

Gdje su y[] - izlazna vrijednost, x[] - ulazna vrijednost, n - broj uzorka u digitalnoj diskretnoj domeni, a_0 i b_0 – rekurzivni koeficijenti filtra.



Slika 26. Odziv jednopolnog niskopropusnog filtra na step funkciju [9]

Algoritam za oblikovanje ovojnice mora se podijeliti na četiri dijela, po jedan za svaku fazu oblikovanja signala, te se unutar svakog (osim *sustaining* perioda jer je onda vrijednost signala konstanta) moraju zasebno računati koeficijenti a_0 i b_0. Veza između koeficijenata može se izraziti preko izraza $a_0 = C$, $b_1 = 1-C$. Primjenom izraza (13) dobija se jednadžba za određenu fazu generiranja ovojnice:

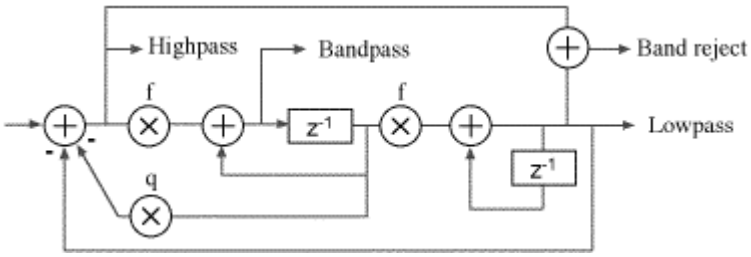
$$y[n] = y[n-1] + C(\textit{odredište} - y[n-1]) \quad (14)$$

Gdje je: *odredište* - vrijednost koja ovisi o tome da li signal pada ili raste, u slučaju da raste (*attack* faza) iznosi 1, a u slučaju da pada (*decay, release* faza) iznosi 0. Vrijednost C izračunava se u ovisnosti o postavljenom vremenu pada, odnosno porasta signala *t* prema izrazu:

$$C = 1 - 0.01^{\frac{1}{t \cdot f_s}} \quad (15)$$

5.4. State Variable Filter

Izlazna jedinica u primjeru Sinthex je digitalna rekonstrukcija (Slika 27.) analognog *State Variable Filtra,* koja je postala veoma popularna jednako kao i njezin analogni dvojnik. SVF, kao sintetizatorski filtar, ima nekoliko prednosti nad *bikvadratnim* filtrima. Niski propust, visoki propust, pojasni propust te pojasna brana su dostupni u isto vrijeme; također kontrola frekvencije i *q* parametra su neovisni jedno o drugom i ne zahtijevaju puno kalkulacije.



Slika 27. State Variable Filter

Frekvencijski kontrolni koeficijent *f* izračunava se pomoću izraza:

$$f = 2\sin(\frac{\pi \cdot f_c}{f_s}) \tag{16}$$

Gdje je f_c granična, odnosno centralna frekvencija filtra. Koeficijent *q* izračunava se pomoću sljedeće formule:

$$q = \frac{1}{Q} \tag{17}$$

Gdje *Q* preuzima vrijednosti iz intervala [0.5,+∞>. U slučaju kada *q* postaje 0, tj. kad Q postane beskonačan, filter se pretvara u stabilni oscilator harmonijskog signala. Pad filtra iznosi 12dB/okt., što je jednako i kod njegovog analognog dvojnika te kod bikvadratnih filtarskih izvedbi.

Glavna mana ovog filtra je pojava nestabilnosti pri višim frekvencijama. Ona ovisi o postavki Q parametra, ali u principu gornja granica stabilnosti je kada *f* poprimi vrijednost 1, što se javlja u slučaju kada f_c poprimi vrijednost 1/6 frekvencije uzorkovanja. Pri frekvenciji 44100 kHz ta granica bi iznosila

7350 Hz. Ovaj problem može se riješiti *naduzorkovanjem.* U realiziranom primjeru ulazni signal je naduzorkovan faktorom 2, što pomiče granicu stabilnosti na 14700 Hz te time zadovoljava zahtjeve za frekvencijski pojas korišten za područje glazbe.

Naduzorkovanje je realizirano kreiranjem dodatnog uzorka čija se vrijednost linearno interpolira između dva susjedna. Nakon toga se filtarski algoritam primjenjuje po jednom na svaki uzorak, te se jedan izlazni uzorak odbacuje, čime se ponovno dobiva jednak broj uzoraka. Sam algoritam filtra u praktičnoj izvedbi je vrlo jednostavan, a njegova izvedba može se naći u dodatku A ovog rada.

5.4.1. Denormalni brojevi

Prilikom implementacija generatorskih jedinica koje sadrže unutar sebe povratnu vezu (engl. *feedback*), na novijim računalima javlja se značajno usporenje kalkulacija uzrokovano obradom vrlo malih brojeva. Taj problem posebice uzrokuje velike smetnje kod elemenata kao što su filtri sa beskonačnim impulsnim odzivom (IIR filtri), tj. oni filtri koji u svojoj rekurzivnoj formuli sadrže prošle izlazne elemente. SVF filtar je takav tip filtra, te su se u njegovoj implementaciji također javili problemi sa denormalnim brojevima.

Postoje tri vrste preciznosti za *floating point* brojeve: 32 bitna (standardna), 64 bitna (dvostruka) i 80 bitna (proširena dvostruka). Prve dvije se najčešće koriste za obradu signala, s tim da se ipak preferira standardna preciznost (engl. *single precision*). Standardna preciznost se preferira jer daje dovoljnu točnost uz manji trošak memorije.

Postoje dva moda zapisivanja *floating point* brojeva. *Normalni* mod pokriva veći dio raspona. Prikaz broja sastoji se od tri dijela: predznaka, karakteristike i mantise. Vrijednost realnog broja dana je izrazom:

$$x = \pm\left(1 + \frac{M}{2^P}\right) \times 2^{E-B} \tag{18}$$

Gdje je: M - vrijednost mantise, B - pozivni broj ovisan o preciznosti. Za standardnu preciznost on iznosi 127 i služi za izbjegavanje prikaza negativnog eksponenta. P - rezolucija mantise, prikazana u bitovima.

U denormalnom modu kodirani broj može se predstaviti sljedećim izrazom:

$$x = \pm M \times 2^{1-B-P} \tag{19}$$

Glavni problem sa denormalnim prikazom brojeva je procesorsko vrijeme potrebnu za njihovu obradu. Primjerice: na AMD Thunderbird procesoru množenje sa denormalnim brojem zauzima oko 170 ciklusa, što je više od 30 puta sporije nego množenje sa normalnim operandima [10].

Rješenje implementirano u Sinthex naziva se *dodavanje konstantne vrijednosti.* Ono se realizira dodavanjem male vrijednosti (1^{-18}) na ulaz povratne veze filtra. Dodavanje te vrijednosti absorbira denormalne brojeve koji se javljaju na ulazu, a sama vrijednost je preniska da bi se mogla čuti (-360dB).

5.5. Realizacija polifonije

Polifoničnost instrumenta (mogućnost sviranja više od jedne note u isto vrijeme) realizirana je poljem zasebnih klasa nazvanih `voice`. `Voice` klasa zapravo predstavlja cijelu instancu sintetizatora, osim filtra koji se globalno primjenjuje na zbrojeni izlaz. Dakle jedna instanca klase sadrži u sebi generatore signala (*OSC1* i *OSC2*) te generator anvelope (*ADSR*). Osim toga, svaka instanca sadrži u sebi *process*, *noteOn* i *noteOff* metode.

Maksimalni broj glasova koji mogu svirati paralelno iznosi 16. Ta brojka je odabrana zbog ograničenja procesorskih i memorijskih mogućnosti (pritiskom jedne tipke potrošnja procesora se uvećava za cijeli postotak jedne instance sintetizatora), a ujedno je realno dovoljna za izvođenje većine standardnih kompozicija (mala je vjerojatnost da će se negdje pojaviti više od 16 nota paralelno za jedan instrument).

Algoritam polifonije mogao bi se predstaviti pomoću sljedećih koraka:

1. Prilikom pritiska na tipku poziva se glavna *noteOn* metoda.

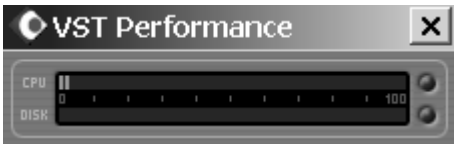
2. Glavna *noteOn* metoda pretražuje polje glasova i traži slobodan glas (koji je određen fazom generatora anvelope).
3. Ako je slobodan glas pronađen, poziva se njegova *noteOn* metoda koja pokreće generator anvelope i time postavlja status glasa na *zauzeto*.
4. Unutar glavne *process* metode pretražuje se polje glasova te se za aktivne glasove poziva njihova pripadajuća *process* metoda čiji se rezultat pribraja na izlaz sintetizatora.
5. Prilikom puštanja tipke poziva se glavna *noteOff* metoda.
6. Glavna *noteOff* metoda pretražuje niz glasova te kad nađe zadani glas, zove njegovu *noteOff* metodu koja postavlja generator anvelope u *release* fazu, nakon koje po njezinom završetku glas preuzima status *slobodno.*

Ovo je jedna moguća realizacija polifonije, ostala rješenja ponekad uključuju drugačije tipove podataka od polja kao što su *vezane liste* ili *stogovi* te još sadrže i *voice stealing* metode (metode krađe glasova, potrebne ako su *release* vremena podešena na velike vrijednosti).

6. Diskusija o dobivenim rezultatima

Realizirani instrument testiran je koristeći Fruity Loops 7 i Cubase SX 3 domaćinske aplikacije. *Debagiranje* (pronalaženje grešaka) aplikacije bilo je vršeno *Toybear Minihost* aplikacijom. Prilikom korištenja instrument nije pokazao značajnije opterećenje Intelovog Core 2 Duo procesora (model E4600 na frekvenciji takta 2.4 Ghz). Na slici 28. može se vidjeti da prikaz opterećenja procesora prilikom paralelnog sviranja 16 glasova i uključenom automatizacijom na svim parametrima (dakle najviše moguće kalkulacije) ne pokazuje preko 5% potrošnje. Dakako, u skladu s očekivanjima, Sinthex ipak opterećuje procesor nijansu više nego neka dostupna profesionalna rješenja. Uzrok tome leži u dvije činjenice. Jedna je ta da razvoj DSP modula zahtjeva ekstenzivno poznavanje optimizacije koda. Unutar ovog rješenja implementirane su različite optimizacijske metode, kao što su: korištenje *inline* funkcija za procesorski važne funkcije, *per-block processing* (obrada signala po bloku), brze funkcije za zaokruživanje brojeva (engl. *fast rounding functions*), aproksimacije nekih matematičkih funkcija (npr. *sinus*), eliminacija denormalnih brojeva, korištenje inicijalizacijskih listi, uspoređivanje s nulom pri provjeravanju uvjeta u petlji, korištenje *prefix* inkrement operatora umjesto *postfix* itd.

Druga činjenica čija posljedica je nijansu veće opterećenje Sinthexa u odnosu na određene profesionalne instrumente je ta da profesionalni proizvodi međusobno konkuriraju na tržištu te svaka kompanija zaštićuje i čuva svoja dostignuća u optimizaciji koda što dovodi nezavisnog programera u situaciju koja bi se mogla opisati idućom izrekom: „Otkrivanje tople vode“ (ili sličnom engleskom frazom: „*Reinventing the wheel*“).



Slika 28. Procesorsko opterećenje Sinthexa pri maksimalnoj polifoniji i automatizaciji

Nadalje, treba naglasiti da specijalizirano grafičko sučelje nije implementirano jer naglasak unutar ovoga istraživanja leži na principima sinteze i obrade zvuka te bi implementacija karakterističnog GUI-a zahtijevala detaljniju dodatnu elaboraciju koja bi izlazila iz zadanih granica ovoga rada.

Unutar ove analize valja se osvrnuti i na pruženu dokumentaciju i sam API Steinbergovog SDK-a. Sama službena dokumentacija dostupna uz Steinbergov VST SDK može se smatrati nepristupačnom te poprilično zbunjujućom, te je zapravo vrlo lako za pretpostaviti da se započinjanje novog VST projekta teško može pokrenuti bez referenciranja na dodatne alate i dokumentaciju nastalu od strane neovisnih stranaka.

Također, može se reći da nedostaje propisivanje detaljnijeg standarda kojim bi se specificiralo ponašanje domaćinske aplikacije prema dodatku. Primjerice: kada idu koji pozivi funkcija, u kojem *threadu*, što se smije izvršiti istovremeno s čim itd. Nedostatak ovog standarda uzrokuje nekompatibilnost određenih dodataka sa određenim domaćinima, kao i potrebu programera za testiranjem proizvoda na više različitih VST aplikacija-domaćina. Postojanje ovakvog standarda oslobodilo bi višak vremena za razvoj samih DSP mogućnosti proizvoda, dok se sad to vrijeme troši na testiranja na različitim *hostovima.*

7. ZAKLJUČAK

Koristeći ovaj rad i realizirani primjer može se dobiti uvid u vrh „sante leda“ koju čini konstantno rastući svijet DSP programiranja, kako idejno – tako i praktički. Nedvojbeno se može tvrditi da budućnost glazbene, a i ostale audio produkcije leži u digitalnoj, a pogotovo računalno-orijentiranoj domeni baratanja sa signalom. Deseci novih VST dodataka, što besplatnih, što komercijalnih, izniču skoro svakodnevno, a da ne spominjemo činjenicu da je VST samo jedan u nizu standarda za kreiranje „DSP magije“.

Ipak, valja naglasiti da komercijalni pristup uvelike usporava razvoj i širenje novih ideja u smislu da se rješenja stvorena od strane različitih entuzijasta i „DSP gurua“ zaštićuju i pretvaraju u „poslovnu tajnu“ koja služi isključivo za ostvarivanje profita. Srećom, postojanje Interneta i na njemu prisutne široke zajednice unutar koje slobodno teku informacije daju predosjećanje da će se uskoro i u potpunosti prevazići i ta barijera.

8. Literatura

1. Wikipedia contributors: 'MUSIC-N'. *Wikipedia, The Free Encyclopedia,* s Interneta, http://en.wikipedia.org/w/index.php?title=MUSIC-N&oldid=229884337, 5. kolovoza 2008.
2. Reimer, Jeremy: Total share: 30 years of personal computer market share figures, s Interneta, http://arstechnica.com/articles/culture/total-share.ars/10, 14. prosinca 2005.
3. Newmann, Toby: How to write a VST plugin, s Interneta, http://www.asktoby.com/Toby%20Newman%20-%20Dissertation%20-%20%20how%20to%20write%20a%20VST%20plugin.pdf, travanj 2002.
4. Roads, Curtis... [et al.]: „The computer music tutorial“, The MIT Press, Cambridge, Massachusetts, 27. veljače 1996.
5. Steinberg Media Technologies GmbH: VST SDK 2.4 Documentation, s Interneta, http://www.steinberg.net, 15. studeni 2006.
6. Somascape: Guide to the MIDI Software Specification, s Interneta, http://www.somascape.org/midi/tech/spec.html, 30. studeni 2005.
7. Bujanović, Zvonimir; Petričević, Vinko: Računarski praktikum 1, Vježbe, s Interneta, http://web.math.hr/nastava/rp1/slideovi.php
8. Wright, Joe: Synthesising band limited waveforms using wavetables, s Interneta, http://www.musicdsp.org/files/bandlimited.pdf, 17. kolovoz 2000.
9. Smith, Steven W.: ”The Scientist & Engineer's Guide to Digital Signal Processing“, California Technical Pub., 1997.
10. Soras, Laurent de: Denormal numbers in floating point signal processing applications, s Interneta, http://www.musicdsp.org/files/denormal.pdf, 11. siječanj 2002.
11. Yehar, Olli Niemitalo: Yehar's digital sound processing tutorial for the braindead, s Interneta, http://www.student.oulu.fi/~oniemita/dsp/dspstuff.txt, 21. siječanj 2003.

Dodatak A: Dijelovi izvornog koda korišteni u realizaciji virtualnog glazbenog instrumenta Sinthex

Glavna funkcija za obradu bloka podataka

```
//**inputs bafer se ne koristi
void Sinthex::processReplacing (float** inputs, float** outputs,
VstInt32 sampleFrames)
{
   //izlazni bafer se postavlja na nulu
   memset( outputs[0],0,sampleFrames*sizeof(float));
   memset( outputs[1],0,sampleFrames*sizeof(float));

   float* o1=outputs[0];
   float* o2=outputs[1];

   VstInt32 i=NUM_VOICES;
   while (--i >= 0)
   {
      if (voices[i].env.stage)
      {
      //za svaki aktivni glas vrse se kalkulacije parametara unutar
      //update() metoda
      voices[i].osc1.update(fDetune,fWaveform1,fVolume1);
      voices[i].osc2.update(fVolume2);
      voices[i].env.update(fAttack,fDecay,fSustain,fRelease);

      //glasovna process metoda akumulira izlazne podatke na izlazni
      //bafer
      voices[i].processAcc(o1, sampleFrames);

      }
   }
   //kalkulacije za filtar
   filtar.update(fCut,fRes,fType);

   filtar.process(o1,sampleFrames);

   //desni kanal se izjednacuje s lijevim
   while (--sampleFrames >= 0) (*o2++)= ((*o1++)*= (this->fVolume));
}
```

Funkcija za obradu MIDI događaja

```
// dogadaji se šalju pomoću *ev parametra
VstInt32 Sinthex::processEvents (VstEvents* ev)
{
	for (VstInt32 i = 0; i < ev->numEvents; i++)
	{
		// prepoznavanje da li je događaj tipa MIDI
		if ((ev->events[i])->type != kVstMidiType)
			continue;

		VstMidiEvent* event = (VstMidiEvent*)ev->events[i];
		char* midiData = event->midiData;

		// ignorira se kanal
		VstInt32 status = midiData[0] & 0xf0;

		// gledaju se samo note
		if (status == 0x90 || status == 0x80)
		{
			// zapis note
			VstInt32 note = midiData[1] & 0x7f;
			//zapis glasnoce
			VstInt32 velocity = midiData[2] & 0x7f;
			// nota ugašena pomoću glasnoće 0
			if (status == 0x80)
				velocity = 0;
			if (!velocity)
				// poziv glavne noteOff metode
				noteOff (note);
			else
				//poziv glavne noteOn metode
				noteOn (note, velocity, event->deltaFrames);
		}
		else if (status == 0xb0)
		{
			// sve note isključene
			if (midiData[1] == 0x7e || midiData[1] == 0x7b)
			{
				//ako su sve note isključene
				//svi aktivni glasovi prelaze u release fazu
				int i=NUM_VOICES;
				while (--i >= 0)
					if (voices[i].env.stage)
					voices[i].noteOff();
			}
		}
		event++;
	}
	return 1;
}
```

Algoritam oscilatora s potraživanjem u valnoj tablici

```
//generiranje signala i akumulacija na ulazni bafer
void C_osc1::processAcc(float *output, VstInt32 sampleFrames)
{
      while (--sampleFrames >= 0)
      {
            //sljedeca linija koda se koristi umjesto modulo
            //operatora jer se % operator generalno smatra skupim
            if (phase > (float)kWaveSize )
                  phase -= (float)kWaveSize;
            float phase_tmp=phase;
            phase += freq;

            //(*output++) += volume*wave1[(VstInt32)phase_tmp];
            //ovo je tocka u kojoj bi neinterpolirajuci oscilator
            //zavrsio

            //linearna interpolacija
            VstInt32 x1=(VstInt32)phase_tmp;
            float y1=wave1[x1];
            float y2=wave1[x1+1];
            (*output++)+=volume*( y1+(phase_tmp -(float)x1)
            *(y2-y1));
      }
}
```

Algoritam generatora anvelope

```
inline void C_adsr::processMul(float *output, VstInt32 sampleFrames)
{
      while (--sampleFrames >= 0){
      //u ovisnosti o fazi, izracunava se odziv filtra prema
      //koeficijentima ili se prelazi na sljedecu fazu
            switch ( stage ) {
                case ATTACK:
                      envelope += attackCoeff * (1.0f - envelope);
                if ( envelope > 0.99f ) {
                            stage=DECAY;
                }
                break;
            case DECAY:
                envelope -= decayCoeff * envelope;
                if ( envelope <= sustainLevel + 0.001f ) {
                    stage = SUSTAIN;
                }
                break;
            case SUSTAIN:
                  if ( sustainLevel == 0.0f )
                        {
                              envelope = 0.f;
                              stage = OFF;
                        }
                break;
                        //release faza se poziva iz noteOff metode,
                        //nakon koje generator prelazi
                        //u OFF fazu sto javlja glavnom programu
                        //da je glass postao neaktivan
```

```
            case RELEASE:
                  envelope -= releaseCoeff * envelope;
                  //prelazak u stanje iskljuceno vrsi se kod
                  //pada signala na -60 dB
                  if ( envelope < 0.001f ) {
                        envelope = 0.f;
                        stage = OFF;
                  }
                break;
            case OFF:
                break;
      }
    (*output++) *= envelope; //ulaz se mnozi sa vrijednoscu anvelope
    }
}
```

Algoritam SVF filtra

```
void C_svf::process(float *output, VstInt32 sampleFrames)
{
      for(int i=0;i<sampleFrames;i++)
      {
            //naduzorkovanje sa faktorom 2 koristeci
            //linearnu interpolaciju
            float i1=(prev + *output) /2;
            prev = *output;

            //osnovni algoritam filtra
            //konstanta ANTIDENORMAL se dodaje
            //radi izbjegavanja operacija u denormalnom
            //modu
            notch = i1 - damp * band + ANTIDENORMAL;
            low   = low + freq * band;
            high  = notch - low;
            band  = freq * high + band;

            //algoritam se ponovno primjenjuje na originalnom
            //uzorku radi pomicanja granice stabilnosti
            notch = *output - damp * band;
            low   = low + freq * band;
            high  = notch - low;
            band  = freq * high + band;

            //u ovisnosti o odabranom tipu filtra postavlja se
            //izlaz
            switch(type)
            {
            case 0:           (*output++) = low/2;          break;
            case 1:           (*output++) = band/2;         break;
            case 2:           (*output++) = high/2;         break;
            case 3:           (*output++) = notch/2;        break;
            }
      }
}
```