SOFTVERSKA ZVUČNA SINTEZA

Jukić, Krešimir

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

SOFTVERSKA ZVUČNA SINTEZA

Krešimir Jukić

Zagreb, siječanj 2020.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, datum.

Krešimir Jukić

Predgovor

Ova stranica treba sadržavati izjavu ili zahvalu kandidata…..

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Ovaj rad će prikazati izradu softverskog sintesajzera upotrebom programskog jezika FAUST (engl. *Functional Audio Stream*). Kao podloga za izradu ovakvog tipa softvera opisat će se osnovni koncepti u zvučnoj sintezi, način korištenja određenih vrsta zvučne sinteze, međusobna interakcija među zvučnim modulima, a nakon toga programski jezik FAUST, njegove mogućnost i sintaksa. Pokazat će se i funkcionalnosti praktičnog dijela rada (izrađenog sintesajzera). Koristit će se neke od Faustovih ugrađenih biblioteka kao i Faustov *online* uređivač i prevoditelj. Napisani kôd biti će izvezen u obliku projekta za JUCE okvir, pomoću kojeg će se u konačnici izraditi korisničko sučelje i izvršiti prevođenje u audio dodatak (engl. *plugin*).

Cilj je, kroz analizu mogućnosti i primjer konkretnog izrađenog softvera pokazati kako Faust zaista pruža mogućnost relativno brzog i značajno olakšanog razvoja, nego bi to bilo u slučaju npr. programskog jezika C++ i to u okviru kompetentnosti programera prvostupnika. Izrađeni softver bit će upotrebljiv u kontekstu suvremene audio produkcije.

Također, ovakav rad mogao bi biti i pitak način za ulazak glazbenog producenta u svijet izrade alata koje svakodnevno koristi.

Ključne riječi: FAUST, audio *plugin*, sintesajzer, softverska zvučna sinteza.

Abstract

This paper will show how to create a software synthesizer using FAUST programming language. As a basis for creating this type of software, some of the fundamental concepts in sound synthesis, the use of certain types of sound synthesis and interaction between the certain audio modules will be described. The programming language FAUST, its ability and syntax, together with created programs logic and its functionalities, will also be explained.

The code will be exported as a JUCE project and at the end compiled as an audio *plugin* ready to be used in a DAW (Digital Audio Workstation).

The goal of the paper is to show how FAUST really provides the opportunity for rapid and very easy development, which wouldn't be the case with programming languages such as C++, all within the competence of the bachelor student. The created software will be usable in the context of contemporary audio production.

Also, this kind of work could be a way to get music producers interested into the world of making the tools they use every day.

Keywords: FAUST, audio plugin, synthesizer, software audio synthesis.

Sadržaj

1. Uvod

Upotrebom digitalne tehnologije u audio produkciji, dolaskom digitalnih audio radnih stanica tzv. DAW-ova (engl. *Digital Audio Workstation*), te porastom performansi računala, softverski sintesajzeri postaju gotovo neizostavni dio produkcije glazbe. Od raznih unikatnih rješenja do kompletnih emulacija hardverskih originala, softverska zvučna sinteza otvara vrata ka do tada nezamislivim manipulacijama zvukom, automatizaciji parametara, novim karakteristikama zvuka, pa i čitavim novim glazbenim žanrovima.

Softverski sintesajzeri u suvremenoj produkciji obično se koriste u obliku dodatka (engl. *plugin*) koji predstavlja softversku komponentu pokrenutu unutar programa domaćina (Pirkle, 2015). Ovime se proširuje funkcionalnost programa domaćina, ali često i funkcionalnosti samog *plugin*-a pomoću alata koje nudi domaćin.

Zbog potrebe visokog stupnja performansi, programski jezik koji prednjači kod pisanja ovakvog tipa softvera je C++. Neophodno je napredno znanje ovog zahtjevnog programskog jezika, da bi se pomoću njega mogao izraditi jedan softverski sintesajzer. Tu dolazimo do programskog jezika FAUST (engl. *Functional Audio Stream*) - funkcijskog programskog jezika za sintezu zvuka i obradu zvuka s velikim naglaskom na dizajn sintesajzera, glazbenih instrumenata, audio efekata itd.

Ovim radom pokazati će se primjer jednog takvog sintesajzera koji će u sebi sadržavati neke od oblika zvučne sinteze obično raspoloživih na većini softverskih sintesajzera. Nakon uvoda i definiranja projektnog zadatka i hipoteze rada, prvi dio rada baviti će se tematikom zvučne sinteze, tako će u trećem poglavlju prvo biti definirane osnovne komponente sintesajzera, njihova funkcija i način rada. Četvrto poglavlje opisuje nekoliko vrsta zvučne sinteze, odnosno načina na koji možemo graditi i oblikovati zvuk, kao i protokolom koji se koristi za komunikaciju između sintesajzera i ostalih uređaja/softvera u produkciji. Nakon toga dolazi se do programskog dijela rada, koji opisuje programski jezik Faust, izrađeni sintesajzer i način njegove izrade. Otvaranje petog poglavlja započinje upoznavanjem sa programskim jezikom Faust, a nastavlja objašnjavanjem njegove sintakse. Šesto poglavlje konkretno opisuje pojedine dijelove izrađenog sintesajzera i programsku implementaciju njegovih određenih dijelova. Poglavlje se zatvara opisom izvoza kôda i prevođenjem u *plugin* spreman za korištenje u programu domaćinu. Rad završava validacijom ostvarenosti hipoteze i zaključkom.

2. Projektni zadatak

2.1. Definicija projektnog zadatka i hipoteza

Projektni zadatak ovog rada je izrada softverskog sintesajzera u obliku *plugin*-a koji će se izvršavati u programu domaćinu. Na tržištu danas postoji velik broj softverskih sintesajzera, a njihove izvedbe su također šarolike. Postoje emulacije postojećih hardverskih analognih i digitalnih sintesajzera, npr. legendarni Moog sintesajzeri - Moog Modular[1] i Moog ARP[2] 2600, kao i potpuno nove programske izvedbe. Razini kompleksnosti i dubini opcija koje pružaju kao i brzini razvoja teško se može konkurirati nekakvim individualnim projektima. S druge strane, velikim brojem opcija na tržištu (u obzir treba uzeti i njihovu nezanemarivu cijenu) lako je doći u situaciju prezasićenosti i gubitka fokusa.

Česte su i situacije prerobusnih, za upotrebu prekompleksnih, a za računalo zahtjevnih izvedbi, dok jednostavnije alternative možda nemaju sve željene opcije. Za glazbenog profesionalca idealna opcija bi bila mogućnost dizajniranja vlastitog alata, sintesajzera izrađenog po mjeri i nadograđivanog po potrebi. Za ovakav pothvat već postoje razna rješenja u vidu grafičkog načina programiranja (programski jezik Pure Data[3] ili Native Instruments – Reaktor[4] – omogućava izgradnju sintesajzera kao modula za njihov Kontakt *plugin*) i predstavljaju vrlo učinkovit način izrade.

Međutim, pisanje samog kôda daje ipak šire mogućnosti; odabir platforme, odabir (ne)vezivanja za određenog proizvođača, pitanje licenci, šire opcije kod izrade grafičkog sučelja, duboka kontrola najosnovnijih gravidnih elemenata sintesajzera i sl. U ovom radu, odabir jezika Faust trebao bi pokazati dobru opciju za iskorak u ovakav (tekstualni) način programiranja. Fokus će, dakle biti na programskom jeziku Faust kojim će program biti i izrađen.

Glavna hipoteza ovoga rada je, da je s kompetencijama stečenim na preddiplomskom studiju programskog inženjerstva moguće napraviti netrivijalan i upotrebljiv zvučni modul koji se može koristiti u nizu profesionalnih DAW-ova na Windows i macOS okruženju. Pokazat će

[1] Moog Modular - https://www.arturia.com/products/analog-classics/modular-v/overview ,siječanj 2020.
[2] Moog ARP 2600 - https://www.arturia.com/products/analog-classics/arp2600-v/ ,siječanj 2020.
[3] Pure Dana - https://puredata.info/ , siječanj 2020.
[4] Native Instruments – Reactor - https://www.native-instruments.com/en/products/komplete/synths/reaktor-6/ ,siječanj 2020.

se i programska pozadina nekih od karakterističnih metoda zvučne sinteze korištenih u većini sintesajzera, a zatim i njihova implementacija samom izradom sintesajzera. Ograničenja ovog sintesajzera, a time i cjelokupnog teorijskog dijela rada koja treba uzeti u obzir su iskustvo programera, vrijeme izrade koje je dato na raspolaganje, ali i proces upoznavanja s nestandardnim programskim jezikom kao što je Faust. Bez obzira što izrađeni audio *plugin* neće moći konkurirati komercijalnim rješenjima, trebao bi odraditi svoju svrhu prikaza mogućnosti tzv. „uradi sam“ opcije koja kao i u svemu ne pruža tvorničke, ali nudi neke druge pogodnosti. U slučaju softvera oni su puno veći, jer takvo se vlastito rješenje može mijenjati i dorađivati bez ograničenja.

2.2. Arhitektura planiranog rješenja

Izrađeni sintesajzer, funkcionalno, trebao bi pokrivati neke od standardnih tipova zvučne sinteze, raznih tehnika manipulacije zvuka i sl. Sadržavati će module generatora zvuka, module kontrola za oblikovanje proizvedenog zvuka (karakteristike ponašanja kroz vrijeme, boje tona, sumiranje više izvora u jedan), grafičko sučelje jasno organizirano u sekcije i pod sekcije, kako bi se to i očekivalo u nekakvom standardnom sintesajzeru tog tipa. Svaki od tri glavna generatora imati će mogućnost odabira 6 tipova zvučnih valova. Ova funkcionalnost koristiti će Faustovu biblioteku generatora zvuka (*oscillators.lib*), a programska logika omogućit će odabir zvučnog vala na svakom generatoru posebno, a onda i njihovo kombiniranje. Globalno sva tri izlaza zvuka bit će moguće podvrgnuti dodatnoj vrsti sinteze modulacijom frekvencije. Isto tako, kao izvor signala bit će i dvije vrste šuma (iz *noises.lib*) - vrsta signala također vrlo upotrebljiva u dizajnu zvuka.

Posebno, sa vlastitom kontrolom pokretanja i moduliranja zvuka, neovisno o ostalim generatorima biti će integriran i modul koji će dodatno pokazati jednu od opisanih vrsta zvučne sinteze ranije u radu. Ostali elementi sintesajzera predviđeni su za oblikovanje postojećeg zvuka, a kao bitna značajka suvremenog sintesajzera biti će ugrađena opcija kontrole boje zvuka kroz vrijeme (pomoću tzv. oscilatora niske frekvencije i zvučnih omotnica). Neki od samih parametara kontrola generatora će moći biti kontrolirani drugim kontrolama, čime će se dobiti i određeni karakteristični efekti ritmičke promjene karakteristika zvuka, a sve to bit će moguće sinkronizirati sa programom domaćinom kako bi se dobiveni efekti mogli uklopiti u tempo koji je definiran u projektu programa domaćina. Sintesajzer će, naravno, imati mogućnost korištenja pomoću klavijature (u tzv. polifonom načinu rada), putem standardnog protokola za komunikaciju između audio komponenti.

Na kraju, instanca kreiranog *plugin*-a pokretati će se unutar digitalne radne audio stanice Logic X na platformi macOS.

3. Komponente sintesajzera

3.1. Oscilator

Elektronički oscilator je elektronički sklop koji proizvodi periodični, oscilirajući elektronski signal. Koristi se u raznim elektroničkim uređajima - digitalnim instrumentima, računalima, sintesajzerima, itd.

Oscilatori čine osnovu sintesajzera, kao glavni izvor zvuka. Zvuk nastaje osciliranjem između dva stanja, i kao što vibrirajuće tijelo stvara zvuk, tako oscilirajući elektronički sklop stvara valni oblik koji se može pojačati i koristiti kao izvor zvuka (Delton, 1984.). Oscilatori stvaraju cikličke oblike valova koji se mjere u hercima (skraćeno Hz - 1 Hz predstavlja ponavljanje događaja u jednoj sekundi). Što je viši stupanj oscilacije, više će ciklusa biti u određenom vremenu, a time će biti viša frekvencija (tj. visina tona).

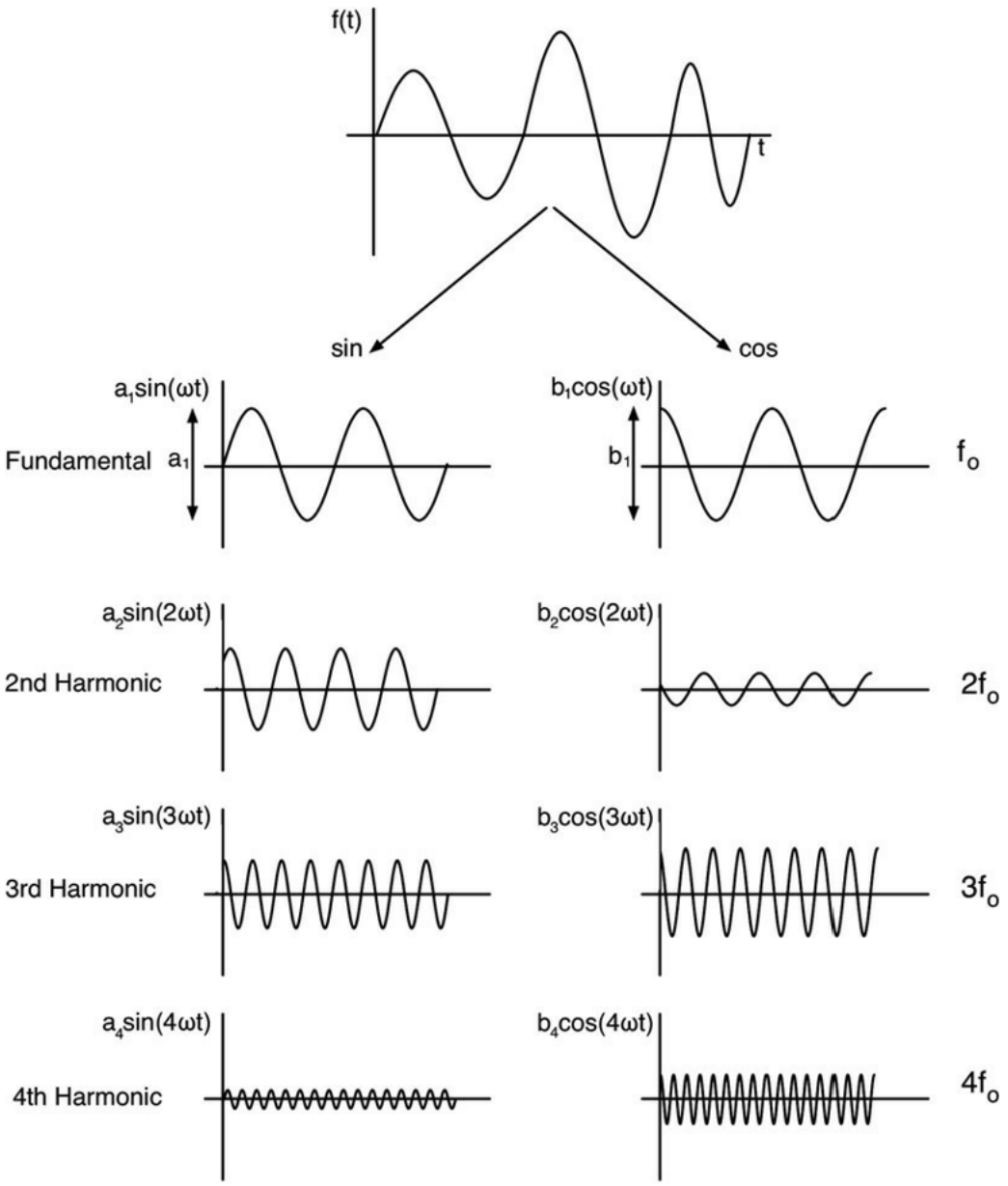
$$F = \frac{1}{t}$$

gdje je: t - vrijeme.

Jedinica mjere koja će biti posebno bitna kod raznih modulacija parametara izrađenog sintesajzera je BPM (engl. *beats per minute*) koji predstavlja puls u minuti, a koristi se za tempo kompozicije/projekta u DAW-u.

Frekvencijom su također definirani tonovi glazbene skale (npr. ton A iznad srednjeg C = 440 Hz). Na ovaj način moći ćemo definirati početnu frekvenciju oscilatora na sintesajzeru, kao referencu za ostale frekvencije koje će se generirati prema primljenim MIDI (engl. *Musical Instrument Digital Interface,* **poglavlje 4.1.**) porukama. Možemo reći da ćemo na ovaj način ugoditi oscilator.

Kod složenih zvučnih signala koji postoje u prirodi, a koje generiraju glazbeni instrumenti i sintesajzeri postoji više frekvencija osim fundamentalne. Frekvencija kojom je definiran ton (engl. *pitch*) naziva se fundamentalnom frekvencijom. Oscilator koji generira sinus, generira samo tu frekvenciju, ostali složeniji oblici osciliranja će generirati prateće tzv. harmonike. Harmonici su množitelji fundamentalne frekvencije (**Slika 1.**).

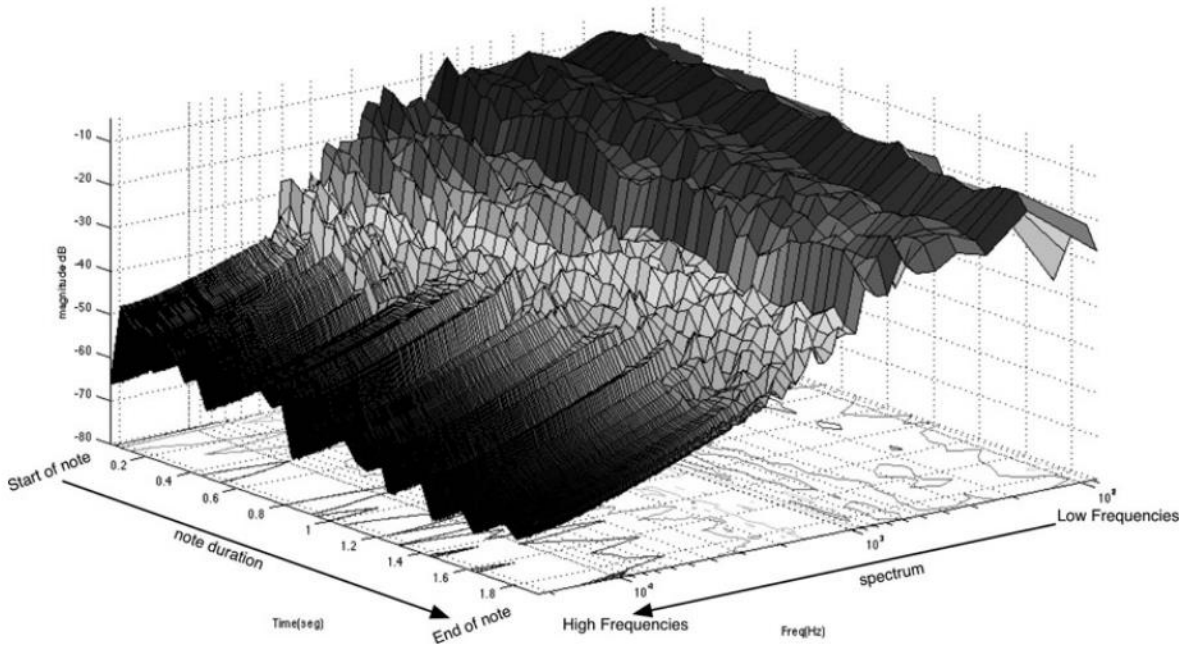


Slika 1 Složeni zvučni val[5]

Harmonici su zaslužni za karakteristiku ili tzv. boju (timbar) ili "kvalitetu" tona. Različite kombinacije broja, vrijednosti, brzine odumiranja u vremenu tih generiranih harmonika omogućiti će karakterističan ton svakog instrumenta (Salford University, 2020). Samim time

[5] Preuzeto iz Pirkle, Will. Designing Software Synthesizer Plug-Ins in C++: For RackAFX, VST3, and Audio Units (p. iii). Taylor and Francis.

ljudsko uho može razaznati razliku između npr. tona A odsviranog na klaviru i istog tona odsviranog na gitari. Primjer tona i razvoj harmonika u vremenu možemo vidjeti na **slici 2**.



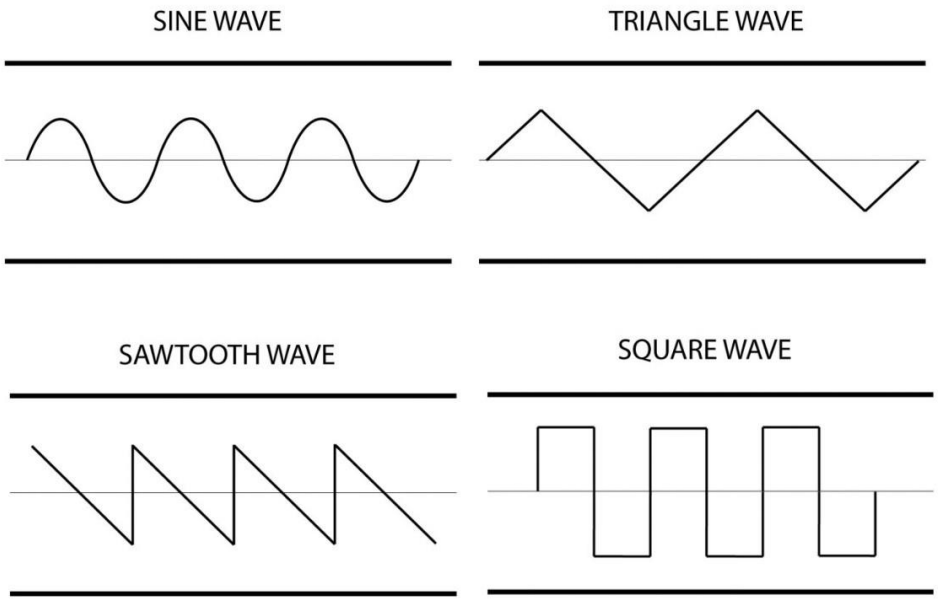
Slika 2 Razvoj harmonika signala u vremenu[6]

Neke od osnovnih oblika zvučnih valova generiranih oscilatorima kakve uobičajeno možemo naći i kakvi će biti korišteni u sintesajzeru su:

- sinusni (engl. *sine*)
- kvadratni (engl. *square*)
- trokutasti (engl. *triangle*)
- zupčasti (engl. *sawtooth*)

Njihove oblike možemo vidjeti na **slici 3.**

[6] Preuzeto iz Pirkle, Will. Designing Software Synthesizer Plug-Ins in C++: For RackAFX, VST3, and Audio Units (p. iii). Taylor and Francis.



Slika 3 Oblici zvučnih valova[7]

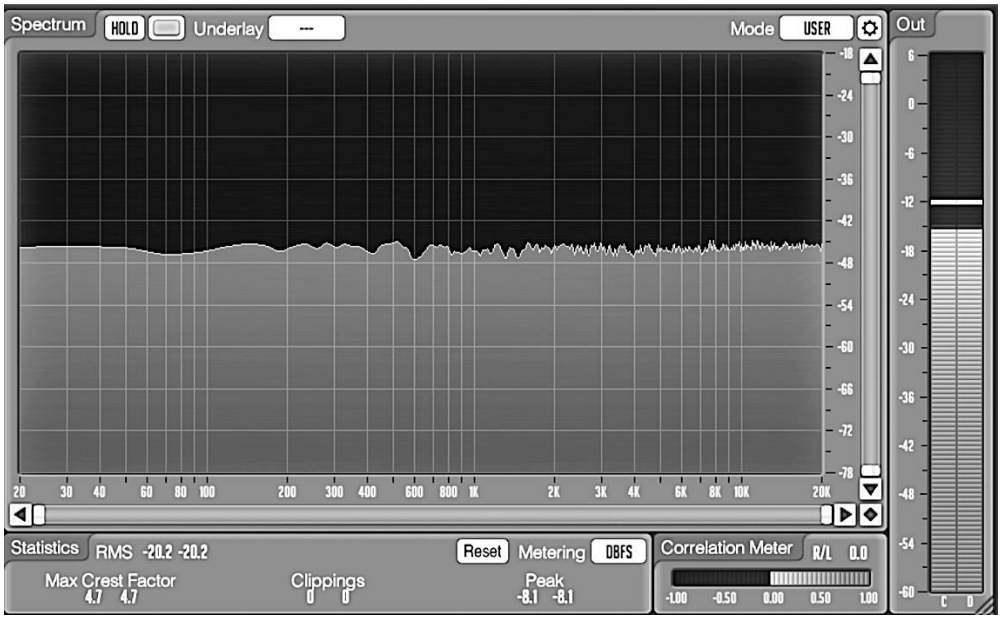
Svi složeniji oblici valova, odnosno svi oni koji nisu sinusi, mogu se dobiti kombinacijom sinusnih valova (Meinard, 2015). U ovom radu koristiti ću ugrađene oscilatore iz biblioteka koje nudi Faust.

3.2. Generator šuma

Osim oscilatora kojima dobivamo cikličke oblike signala, u sklopu jednog sintesajzera također su nam potrebni i izvori zvuka kojima bi simulirali razne aperiodične zvukove. To mogu biti razni tipovi neartikuliranih zvukova, šumova, zvučnih efekata, simulacija prirodnih pojava (grom, šum valova,…). Generatore šuma često koristimo za kreiranje perkusivnih zvukova - npr. elementa iz seta bubnjeva (doboš, činele...). Za sve navedene slučajeve koriste se najčešće dva tipa generatora šuma, tzv. generator ružičastog šuma (engl. *pink noise*) i generator bijelog šuma (engl. *white noise*).

Bijelim šumom nazivamo signal s ravnim frekvencijskim spektrom, ako ga prikažemo kao linearnu funkciju frekvencije. Ovaj signal ima jednaku snagu u bilo kojem pojasu određene širine pojasa. Npr. bijeli šum u pojasu raspona frekvencija između 40 Hz i 60 Hz sadrži istu količinu zvučne snage kao i raspon između 400 Hz i 420 Hz, jer su oba intervala širine 20 Hz (**Slika 4.**).

[7] Preuzeto sa https://www.thedawstudio.com/types-of-sound-waves/ ,siječanj 2020.



Slika 4 Spektar bijelog šuma - ravni spektar snage

Frekvencijski spektar ružičastog šuma ima jednaku snagu u pojasevima koji su proporcionalno široki (**Slika 5.)**. To znači da bi ružičasti šum imao jednaku snagu u frekvencijskom rasponu od 40 do 60 Hz kao u pojasu od 4000 do 6000 Hz. Ljudsko uho percipira udvostručavanje frekvencije (oktava) jednako i u slučaju npr. intervala 40 – 60 Hz kao udaljenost od 4000 – 6000 Hz. Budući da u svakoj oktavi imamo jednaku količinu energije, ružičasti šum često koristimo kao referentni signal u audio inženjerstvu (podešavanje razglasa, analiziranje frekvencijske izbalansiranosti režije, itd...).

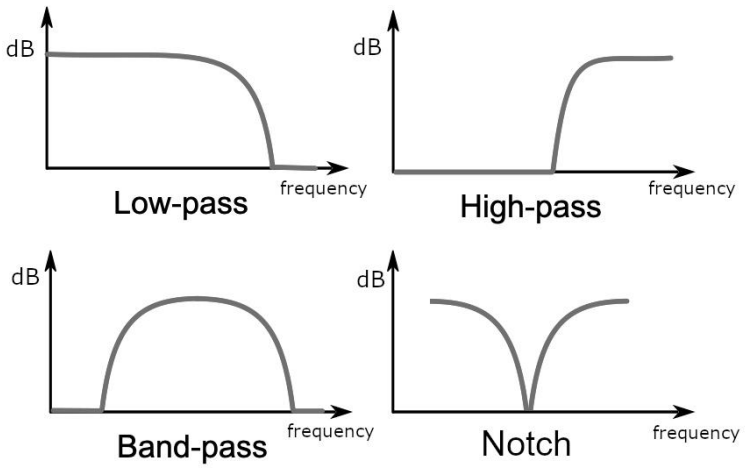
Slika 5 Spektar ružičastog šuma - ravni spektar snage

U obzir treba uzeti da je os x logaritamska skala, a ne linearna, što rezultira time da ista fizička širina pokriva veći raspon Hz na višim frekvencijama nego na nižim.

3.3. Filteri

Filteri se obično koriste za uklanjanje (u nekim slučajevima pojačavanje) određenog dijela frekvencijskog spektra iz složenog signala, a tehnika se često naziva subtraktivna sinteza (Indiana University, siječanj 2020.). Tako npr. ranije spomenuti oscilator sinusa ne bi imalo smisla puštati kroz filter, jer se sastoji od samo jedne frekvencije. Prema tome, nikakav efekt u promjeni zvuka ne bi dobili osim što bi eventualno stišali tu frekvenciju ako bi bila izvan granica onih frekvencija koje smo odlučili filtrirati. Neki od osnovnih tipova filtera su (**slika 6.**):

- nisko propusni (engl. *low-pass*) filter - uklanja visoke frekvencije, odnosno propušta niske frekvencije
- visoko propusni (engl. *highpass*) filter - uklanja niske frekvencije, odnosno propušta visoke frekvencije
- pojasni propusni (engl. *bandpass*) filter - propušta određen raspon frekvencija, a reže frekvencije van obje strane pojasa
- filter usjeka *(*engl. *notch filter)* - uklanja vrlo uzak raspon frekvencija oko centralne, odabrane, frekvencije



Slika 6 LPF, HPF, band-pass filter i notch filter[8]

Filteri ne odstranjuju/pojačavaju potpuno određeni spektar frekvencija, nego uvijek postoji stupanj "opadanja" ili krivulja prema kojoj se stišavaju frekvencije van određenih granica. Obično se te krivulje izražavaju vrijednostima dB (decibel) po oktavi. Što imamo veću vrijednost dB to će krivulja pada ili uspona biti oštrija. Krivulje također nemaju uvijek linearno kretanje. Npr. kod low-pass filtera (**slika 6)**, možemo primijetiti da stupanj smanjivanja vrijednosti dB postaje veći na nižim vrijednostima dB skale, a time i krivulja opadanja postaje oštrija, čime u ovom slučaju dobivamo inverzno eksponencijalan oblik funkcije opadanja.

Osim oblikovanja timbra zvuka puštanjem određenog/ih signala kroz jedan ili više filtera, velika uloga filtera u zvučnoj sintezi i glazbenoj kompoziciji u određenim glazbenim žanrovima dolazi do izražaja kada se parametri filtera mijenjaju u vremenu (dinamički). Na taj način dobivamo dinamiku u boji zvuka, određene ritmičke efekte i pulsacije, ako se te promjene odvijaju u tempu glazbe i slično. Neke od tih vrsta modulacija biti će objašnjene u primjeru izrađenog sintesajzera.

3.4. Modulatori

Prilikom opisivanja filtera, dotaknute su neke od tehnika u kojima se vrijednosti filtera mijenjaju tokom vremena. Takve promjene obično nazivom modulacijom. Moduliranja se obično postižu tzv. LFO-ima (engl. *Low Frequency Oscillators*) iliti oscilatorima niskih frekvencija i zvučnim omotnicama (engl. *envelope*).

[8] Prilagođeno sa https://en.wikipedia.org/wiki/Filter_(signal_processing), siječanj 2020.

3.4.1. Oscilator niske frekvencije

LFO je oscilator koji oscilira vrijednostima koje se kreću ispod ljudskog čujnog spektra (do 20 Hz). Takav signal ima ulogu modulatora neke druge komponente. Tako npr. možemo poslati signal LFO-a u pojačalo oscilatora (zaduženo za glasnoću signala) i time modulirati tu vrijednost u vremenu i to frekvencijom na koju je LFO postavljen. Ovime ćemo dobiti efekt "pumpanja" zvuka, često korišten npr. u produkciji elektronske plesne glazbe (engl. *EDM – Electronic dance music*).

Druga, vrlo česta upotreba je moduliranje parametara filtera kroz koji smo pustili zvuk. U pravilu je, kod softverskih sintesajzera, moguće modulirati bilo koji parametar - frekvenciju rezanja (engl. *cutoff frequency*), Q (širinu pojasa zahvaćanja kod određenog tipa filtera) itd. Budući da sintesajzeri također, u pravilu imaju ugrađen sat (u slučaju softverskih to je npr. MIDI sat[9]), LFO je moguće sinkronizirati sa tempom, koji će u slučaju *plugin*-a slati program domaćin, i time dobiti ritmičke izmjene u promjeni zvuka koje će se moći smisleno uklopiti u aranžman glazbenog dijela.

Ideje i opcije za modulaciju pomoću LFO-a su neograničene i praksa moduliranja obuhvaća veliku paletu i komponenata i njihovih parametara u suvremenoj glazbenoj produkciji - od moduliranja oscilatora, generatora šuma, moduliranja modulatora, moduliranja parametara zvučnih omotnica, varijacije su neograničene.

3.4.2. Zvučna omotnica

Zvučnom omotnicom definiramo kako će se zvuk razvijati tijekom vremena. Običnim pokretanjem i zaustavljanjem oscilatora pokrenuti ćemo i prekinuti zvuk, ali nemamo mogućnost određivanja karakteristika amplitude tona. Zvučna omotnica se prije svega odnosi na amplitudu, ali se također kao i LFO može primjenjivati i kod moduliranja raznih elemenata i parametara sintesajzera, ako takva implementacija postoji.

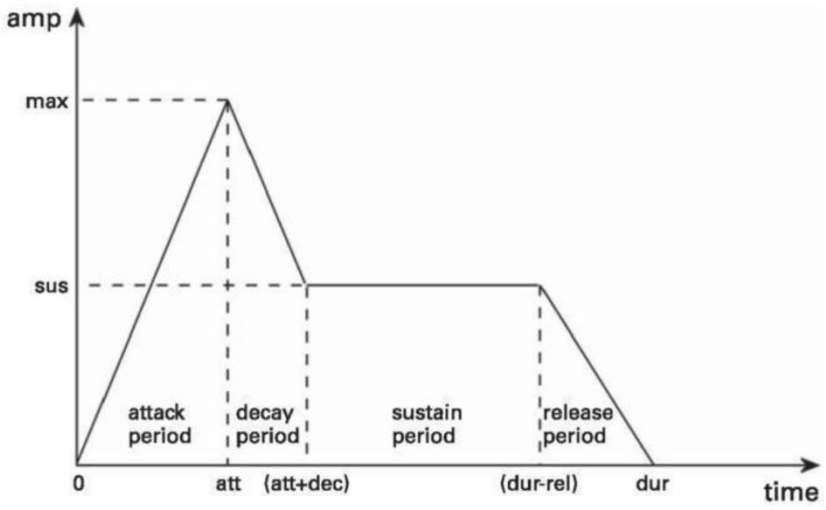
[9] MIDI sat (engl. *MIDI Clock*) je signal koji se odašilje preko MIDI protokola kako bi se osigurala sinkronizacija među uređajima. Ovisan je o definiranom tempu.

3.4.2.1. Parametri zvučne omotnice

Generatori omotnica najčešće imaju četiri stupnja:

- A - Napad (engl. *Attack)* - vrijeme potrebno za početno pokretanje razine od nule do vrha, počevši od pritiska na tipku.
- D - Opadanje (engl. *Decay*) - vrijeme potrebno za naknadno spuštanje s razine napada na zadanu razinu održavanja.
- S - Držanje (engl. *Sustain*) - razina tijekom glavnog slijeda trajanja zvuka, sve dok se tipka ne pusti.
- R - Otpuštanje (engl. *Release*) - vrijeme potrebno da razina padne s razine držanja na nulu nakon puštanja tipke na klavijaturi

  Grafički prikaz omotnice možemo vidjeti na **slici 7**.



Slika 7 Zvučna omotnica[10]

ADSR ima značajnu ulogu kod dizajniranja zvuka. Ako npr. želimo kreirati zvuk kakav bi odgovarao nekakvom nenametljivom i mekanom glazbenom tepihu, osnovne karakteristike takvog zvuka bi bile (osim samog timbra) dugačak *attack* i dugačak *release*. Ovakav *envelope* bi omogućio pretapajuću izmjenu harmonija sličnu načinu sviranja violinskog orkestra. Isto tako, ako bi željeli dobiti izrazito perkusivan zvuk postavke bi bile gotovo obrnute.

Ovakve modulacije, kao što je već spomenuto, mogu se primijeniti i na druge parametre sintesajzera, a jedna od uobičajenih je modulacija vrijednosti rezanja filtera, tzv. modulacija

[10] Preuzeto iz Richard Boulanger; Victor Lazzarini; Max Mathews. The Audio Programming Book (Kindle Location 1), 2011. Massachusetts Institute of Technology

otvaranja/zatvaranja. Ovo je još jedan efektan način kako dobiti dinamičnost, kretanje i osjećaj bogatosti generiranog zvuka.

4. Vrste zvučne sinteze i MIDI

4.1. MIDI

MIDI[11] (engl. *Musical Instrument Digital Interface*) je tehnički standard koji istovremeno opisuje komunikacijski protokol, digitalno sučelje i elektronički priključak. Razvojem MIDI-ja omogućena je komunikacija među različitim elektroničkim instrumentima tako da se bilo koja MIDI tipkovnica (ili šire rečeno bilo koji tzv. MIDI kontroler) može povezati s bilo kojim drugim MIDI kompatibilnim sekvencerom, zvučnim modulom, kolokvijalno "ritam-mašinom", sintesajzerom, itd., proizvedenim od različitih proizvođača. Uobičajeno je da DAW bude posebno dizajniran za rad s MIDI-jem kao integralnom komponentom. MIDI editori razvijeni su u mnogim DAW-ovima tako da se snimljene MIDI poruke mogu lako mijenjati, reorganizirati i općenito manipulirati njihovim popratnim vrijednostima.

4.1.1. MIDI poruke

MIDI poruke šalju se pomoću MIDI kontrolera (pritiskom tipke, okretanjem potenciometra i sl.) u različite zvučne module, druge kontrolere i/ili DAW. Prilikom izvedbe, te poruke sadrže skup informacija koje, moglo bi se reći, potpuno opisuju izvedbu. Jednom zapisana izvedba može se naknadno potpuno izmijeniti – promjenom tonaliteta, tempa, raznih kontrolnih poruka koje se odnose na glasnoću, ekspresiju zvuka, automatizaciju parametara modula, itd. Uobičajena praksa u audio produkciji također je programiranje same izvedbe pri čemu se ovdje ne misli na pisanje kôda nego upisivanje notnog zapisa sa pratećim artikulacijama i svim ostalim podacima, u ovisnosti što zvučni modul u koji se šalju nudi, a dio kojih bi se inače snimio u realnom vremenu prilikom izvedbe glazbenika (tj. sviranja) na MIDI klavijaturi.

MIDI zapis može se spremiti u MIDI datoteku, distribuirati i reproducirati na bilo kojem računalu ili elektroničkom instrumentu koji se također pridržava istih MIDI standarda, ili djelomično prilagoditi specifičnim postavkama drugačijeg modula od onog za koji je

[11] MIDI specifikacija - https://www.midi.org/ siječanj, 2020.

prvotno bio namijenjen. Budući da MIDI datoteke u sebi ne sadržavaju zvuk, veličina im je značajno manja od audio zapisa.

MIDI poruka sastoji se od statusnog bajta i jednog ili dva bajta podataka (MIDI messages, siječanj 2020.). Na najvišoj razini MIDI poruke dijelimo na kanalne i sistemske poruke. Neke od poruka koje su neophodne za korištenje izrađenog sintesajzera su:

Note On / Note Off / Velocity

šalju signal aktivacije (*Note On*) i otpuštanja (*Note Off*) pojedine tipke kontrolera (tj. trajanje note) te "jačinu" pritiska note (*velocity*) koja utječe na glasnoću i „ekspresiju“ tona note (ako takva funkcionalnost u sintesajzeru postoji).

Control Change (CC)

koriste se da bi se omogućilo mapiranje određenih parametara sintesajzera sa fizičkim tipkama na kontroleru ili kako bi se omogućila kontrola i/ili automatizacija istih pomoću DAW-a.

4.2. Zvučna sinteza

Zvučna sinteza, jednostavno rečeno, je tehnika kreiranja zvuka. Kako općenito riječ “sinteza" ima za značenje spajanje ili sastavljanje više dijelova u nekakvu cjelinu, i u ovom slučaju govorimo o spajanju, i to audio signala (složenijih ili jednostavnijih) u nekakav drugi rezultirajući signal.

Postoje razne vrste zvučne sinteze, a neke od njih su:

- aditivna sinteza - kombiniranje tonova, obično harmonika različitih amplituda
- subtraktivna sinteza - filtriranje već postojećih složenijih zvukova
- sinteza moduliranjem frekvencije (engl. *Frequency Modulation - FM*) - modulacija vala nositelja s jednim ili više valova modulatora
- sinteza bazirana na uzorkovanju (engl. *Sample-based synthesis*) - korištenje uzorkovanih zvukova koji su naknadno modificirani
- sinteza pomoću faznih izobličenja (engl. *Phase distortion synthesis*) - promjena brzine valnih oblika spremljenih u valnim tablicama tijekom reprodukcije

- granularna sinteza (engl. *Granular synthesis*) - kombiniranje nekoliko malih zvučnih segmenata u novi zvuk
- sinteza fizičkim modeliranjem (engl. *Physical modelling synthesis*) - generiranje zvuka pomoću matematičkog modela (simulacija fizičkih izvora zvuka)

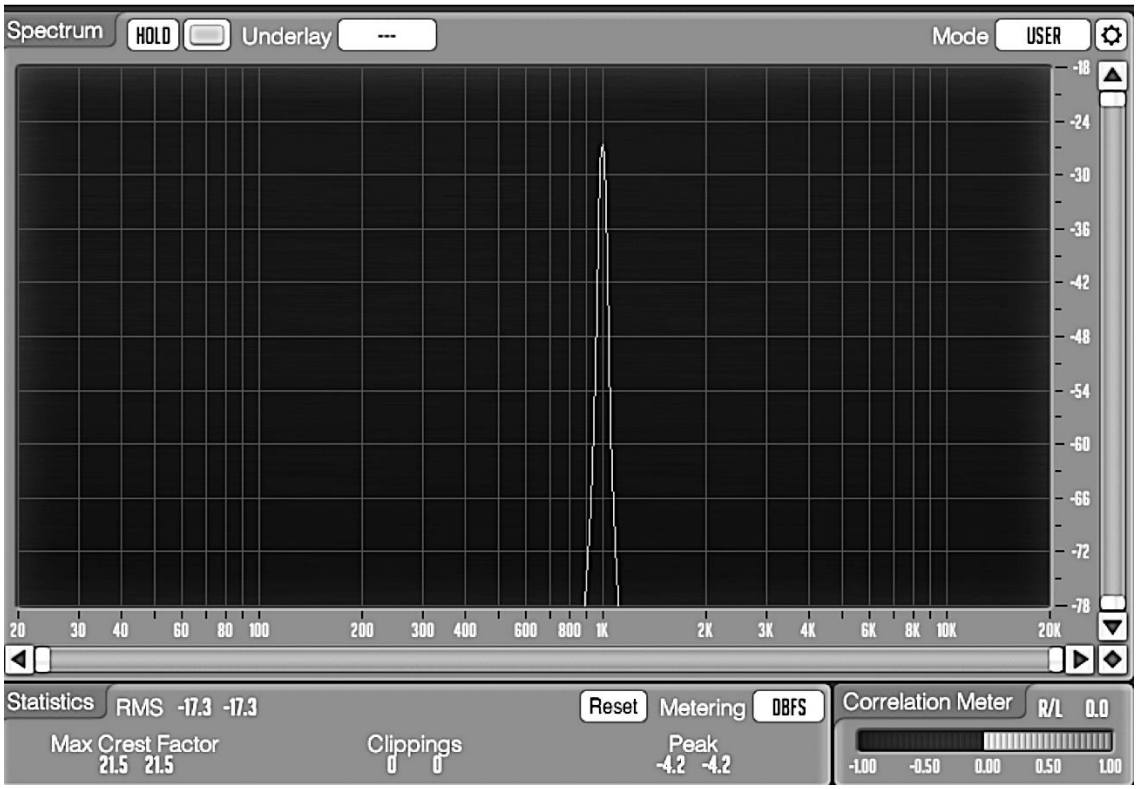
U nastavku će biti objašnjene sve vrste sinteza koje su korištene u izradi praktičnog dijela ovoga rada[12].

4.2.1. Aditivna sinteza

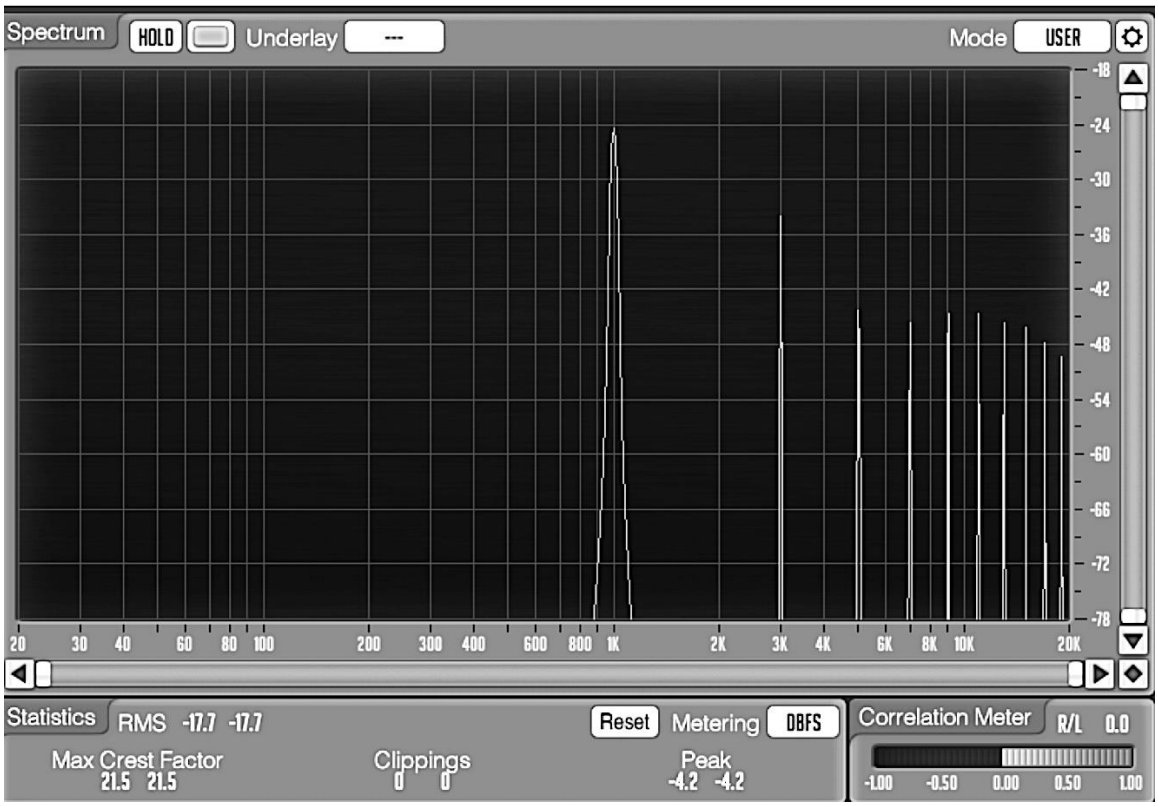
Aditivnom sintezom možemo nazvati spajanje sinusnih valova počevši od fundamentalnog, sa dodatnim tzv. harmonicima također sinusnim valovima. Fourier-ova analiza[13] pokazuje kako je sve složene zvukove teoretski moguće sastaviti od sinusnih valova. Osnovni ton, tj. fundamentalni sinus, posjeduje određen broj harmonika (nekad nazvanih i alikvotni tonovi) u određenim jačinama (vrijednostima amplitude sinusa) kao i njihovim rasporedom.

Na **slikama 8., 9. i 10.** vidljive su razlike u frekvencijskim karakteristikama zvučnih signala.

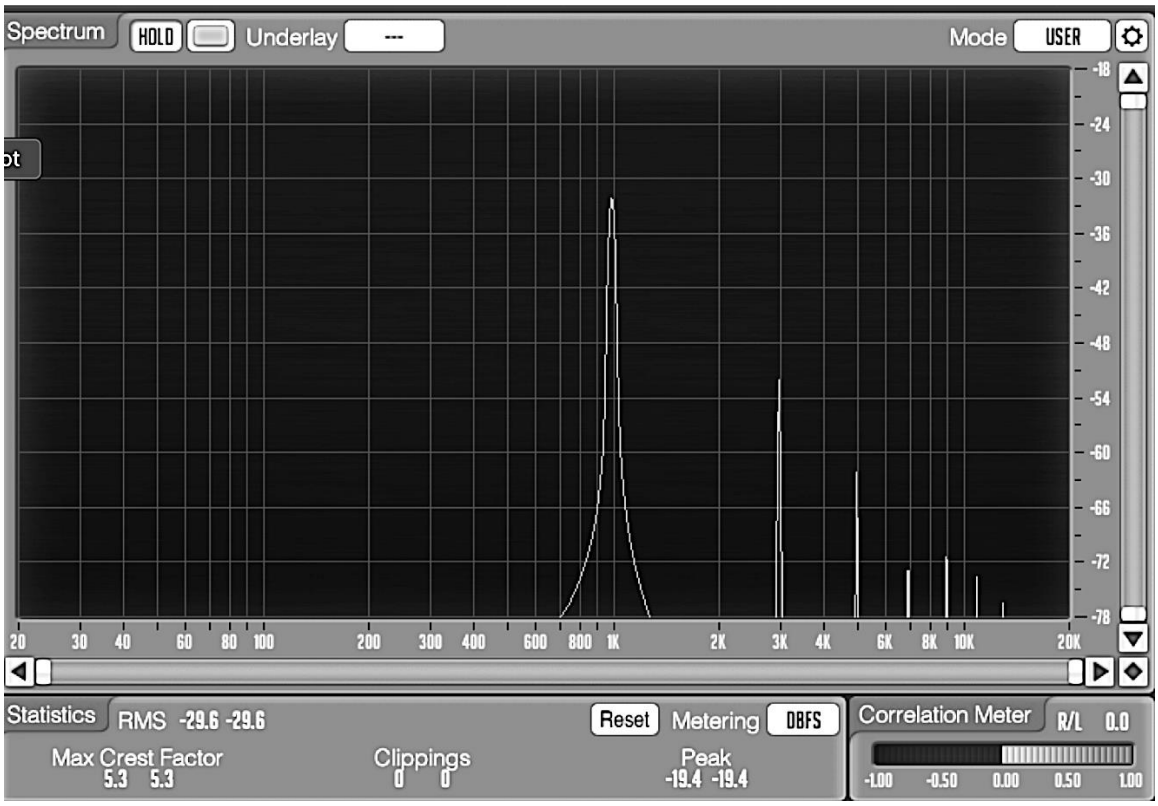
[12] Više o ostalim navedenim vrste sinteze na:
Sinteza bazirana na uzorkovanju - https://en.wikipedia.org/wiki/Sample-based_synthesis
Sinteza pomoću faznih izobličenja - https://en.wikipedia.org/wiki/Phase_distortion_synthesis
Granularna sinteza - https://en.wikipedia.org/wiki/Granular_synthesis
Sinteza fizičkim modeliranjem - https://en.wikipedia.org/wiki/Physical_modelling_synthesis
[13] http://mathworld.wolfram.com/FourierSeries.html



Slika 8 Frekvencijski prikaz sinusnog vala



Slika 9 Frekvencijski prikaz kvadratnog vala



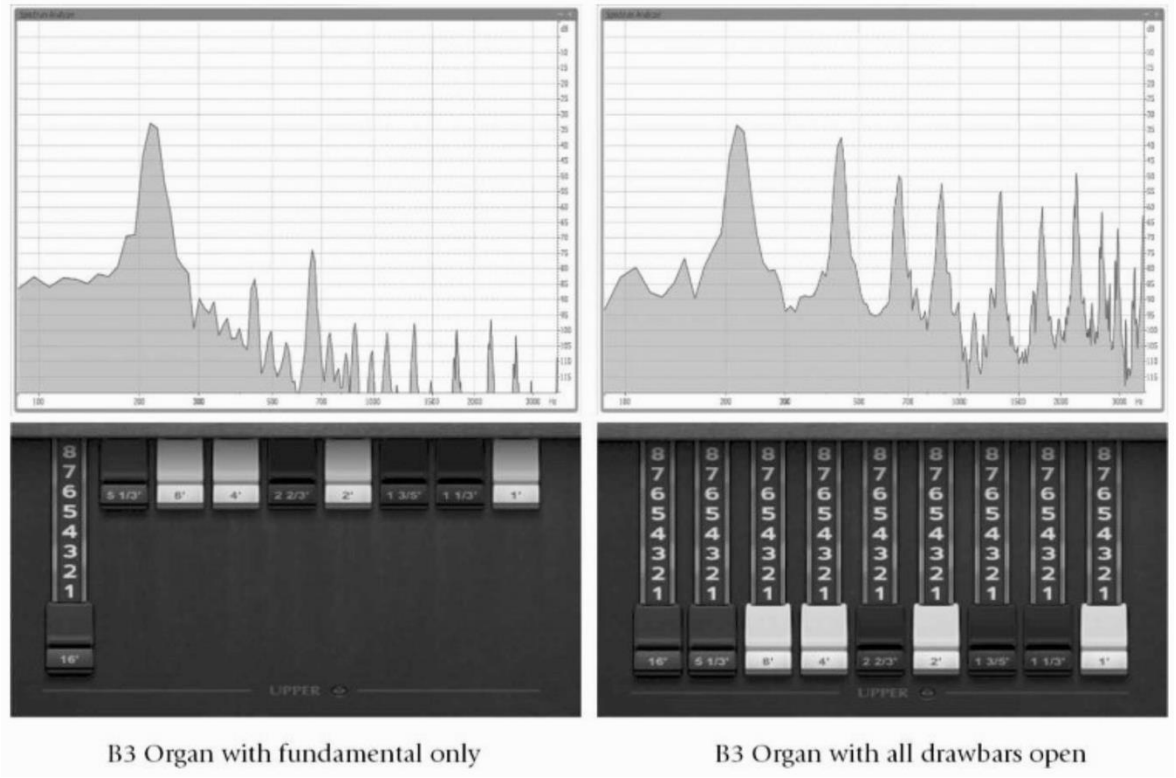
Slika 10 Frekvencijski prikaz zupčastog vala

U slučaju sinusa vidimo samo jednu frekvenciju od 1 kHz. Sinus je najjednostavniji mogući oblik i nema viših harmonika. Kod kvadratnog i zupčastog vala vidimo dodane više harmonike, oba vala ih imaju, ali se razlikuju po svom rasporedu i intenzitetu, iako se u glazbenom smislu radi o istom tonu od 1 kHz.

Ove razlike čine timbar ili boju zvuka, i tim harmonicima pokušavamo manipulirati kako bi dobili željnu boju (Pejrolo i Metcalfe, 2017).

Primjer implementacije ove vrste sinteze u fizičkom obliku možemo vidjeti na primjeru Hammod [14]orgulja. Povlačenjem kontrola na konzoli orgulja omogućeno je pojačavanje ili stišavanje određenih harmonika. U ovom slučaju orgulje nisu proizvodile čisti sinusni val, ali koncept izgradnje zvuka je isti. **Slika 11** pokazuje softversku emulaciju izgleda tih kontrola.

[14] Hammond orgulje - http://theatreorgans.com/grounds/docs/history.html ,siječanj 2020.



Slika 11 Hammond kontrole harmonika, postavka sa samo fundamentalnim tonom (lijevo), postavka sa svim klizačima otvorenim (desno)[15]

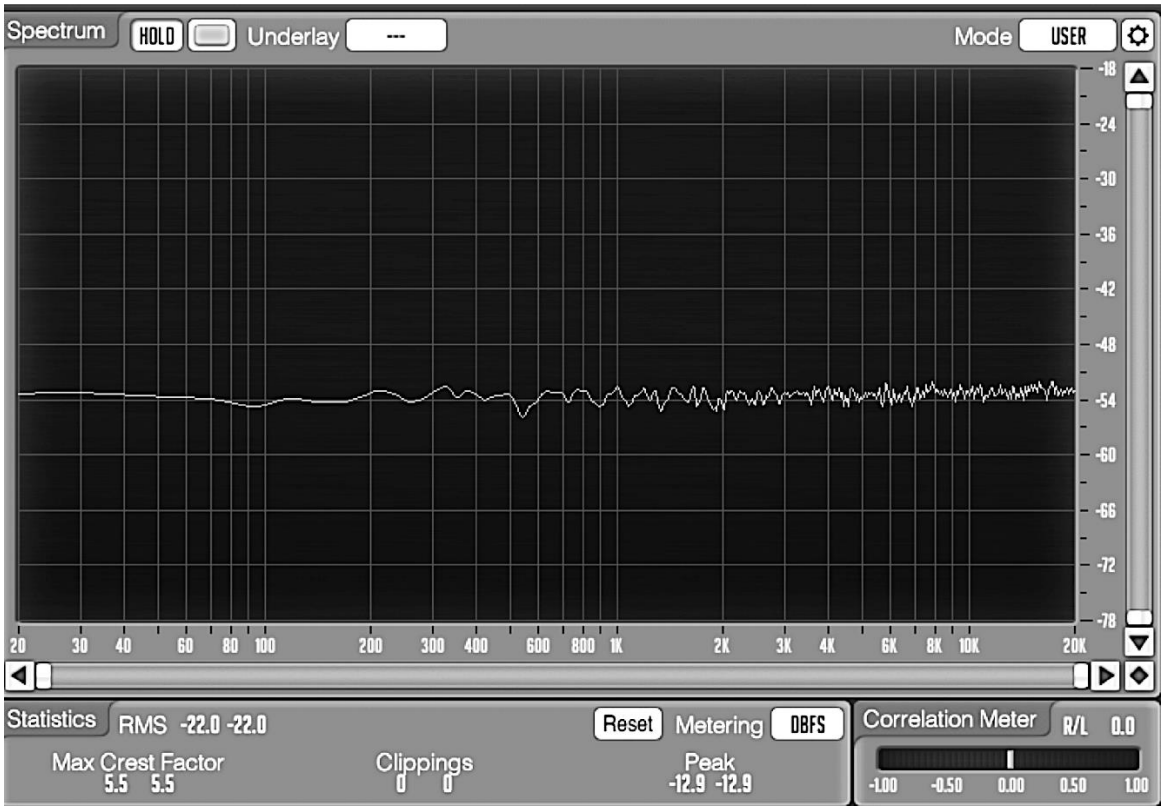
4.2.2. Subtraktivna sinteza

Jedan od najšire korištenih oblika zvučne sinteze je subtraktivna zvučna sinteza. Za razliku od aditivne, gdje krećemo od najjednostavnijih zvučnih valova i na tome dalje gradimo zvuk, ovdje krećemo od već složenih zvukova. Takve, već kompleksne zvukove, podvrgavamo raznim obradama kako bi dobili drugačiji zvuk od originalnog. Za ovaj tip sinteze, neće nam biti pogodan sinusni val, ali možemo krenuti već od zupčastog, kvadratnog, vala ili njihovih raznih kombinacija, ili naravno, bilo kakvog drugog kompleksnog signala.

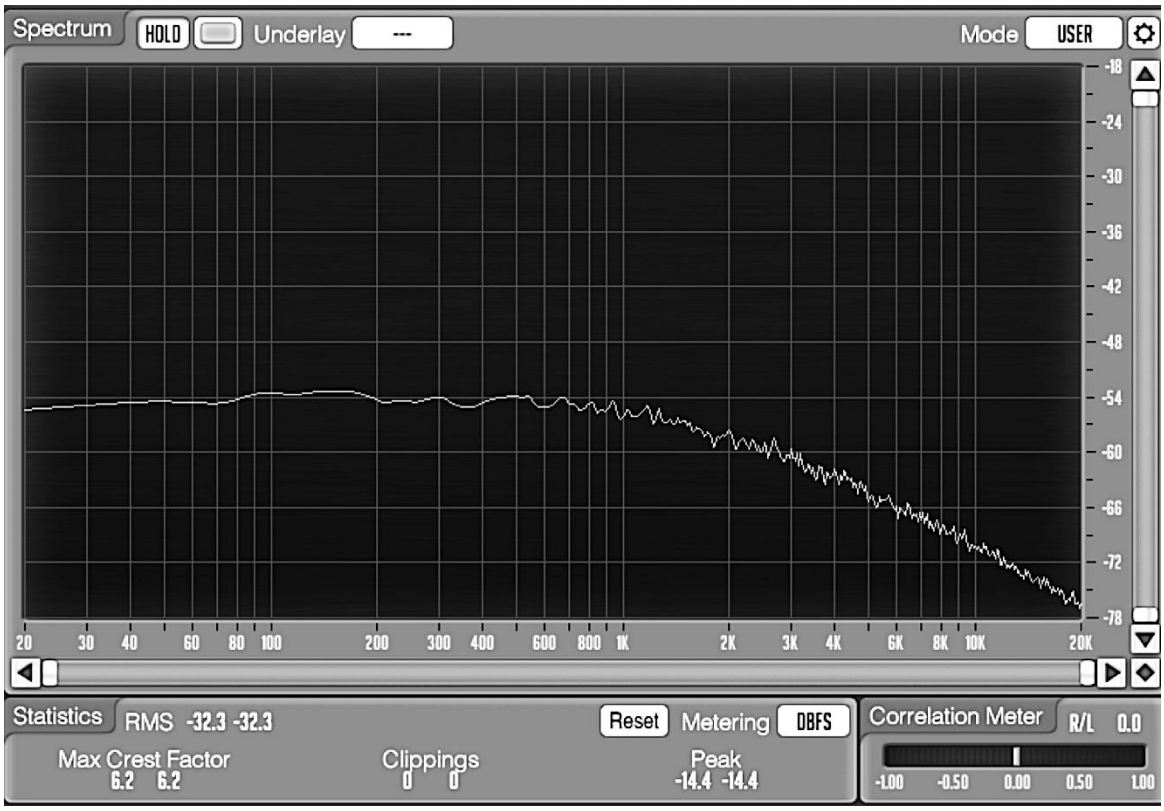
Neke od uobičajenih tehnika su filtriranje signala. Generiramo signal i puštamo ga kroz određene filtere koji mu mijenjaju karakter stišavajući, pojačavajući, izobličavajući određeni dio spektra.

[15] Preuzeto iz Andrea Pejrolo, Scott B. Metcalfe, Creating Sounds from scratch, 2017.

Na primjeru bijelog šuma možemo vidjeti razliku u frekvencijskom spektru prije i poslije primjene filtera na signal. **Slika 12.** prikazuje bijeli šum, dok **slika 13.** prikazuje isti šum pušten kroz LPF. Rezultat filtriranja bijelog šuma u ovom slučaju dati će tamniji i mekši ton.



Slika 12 Bijeli šum

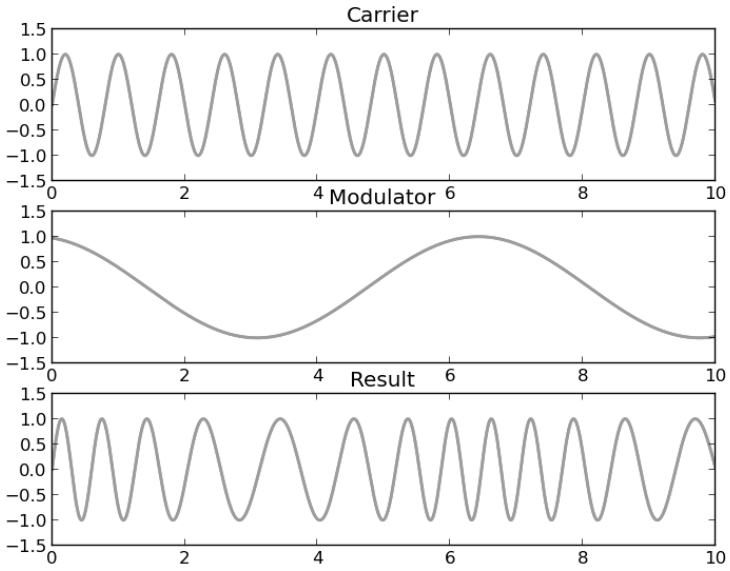


Slika 13. Bijeli šum sa redukcijom signala na 1540 Hz, 6 dB/oktavi

Vrlo uobičajena praksa, pogotovo u žanrovima elektronske glazbe, je manipulirati količinom redukcije signala u realnom vremenu, što znači da zvuk neće imati iste karakteristike u svakom trenutku, nego će se one mijenjati kroz trajanje tona. Na ovaj načni obogaćujemo i stvaramo određeni razvoj tona, te stvaramo interes kod slušatelja.

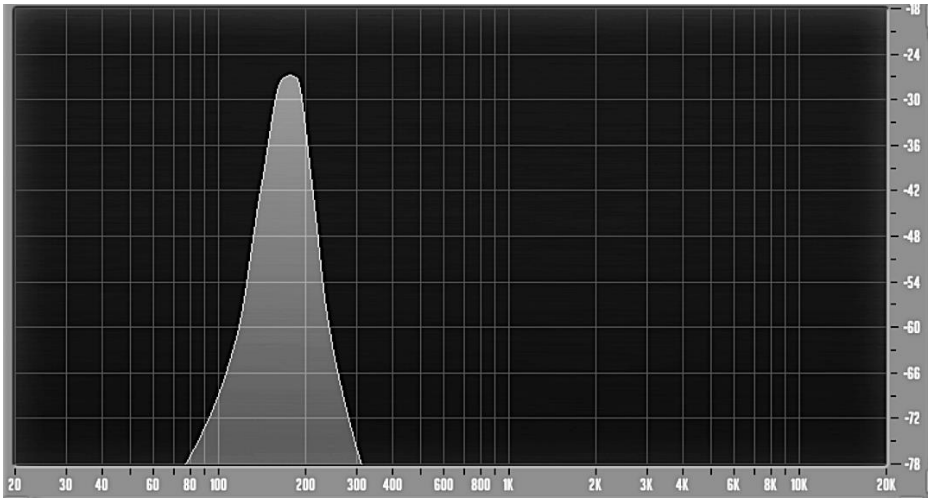
4.2.3. FM sinteza

Kod FM sinteze postoje dva osnovna elementa - nositelj (engl. *carrier*) i modulator. U osnovi, FM sintezu je moguće ostvariti sa samo dva sinusna vala. Umjesto uobičajenog zbrajanja signala, signal nositelj modulira se signalom modulatorom i to na sličan način kao i kod modulacije modulatorom niske frekvencije (LFO) čije se vrijednosti kreću van slušnog spektra, međutim ovdje se signal modulira signalom frekvencija koje su unutar tog spektra, čime se javljaju vrlo interesantni zvučni valovi, vrlo karakterističnog zvuka. Kako povećavamo količinu frekvencije modulacije, dobivamo sve složeniji zvuk. Primjer FM modulacije možemo vidjeti na **slici 14.**

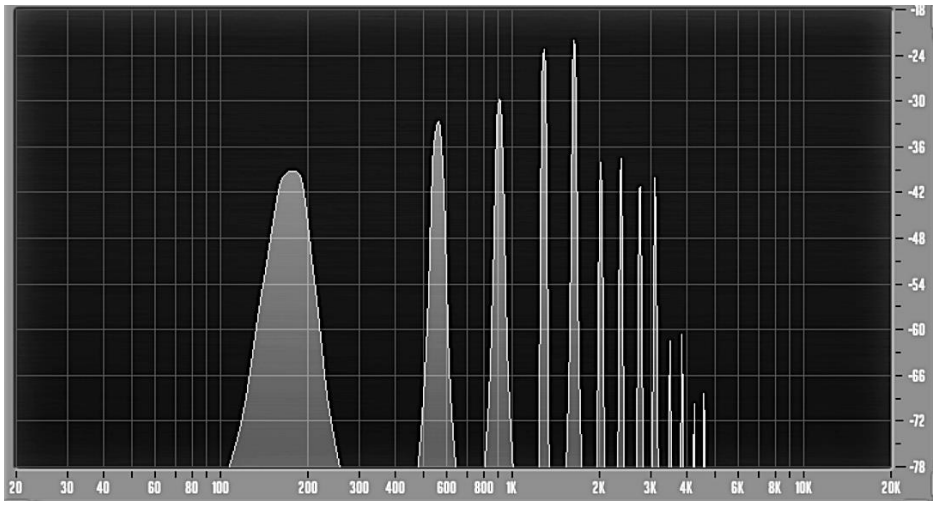


Slika 14 Sinusni nositelj frekvencije moduliran sinusnim modulatorom i rezultirajući val

Karakteristika dobivenog zvuka su dodane frekvencije koje nazivamo popratnim pojasevima (engl. *side bands*) koji se miješaju sa signalom nositeljem. Kad je modulator postavljen na nisku razinu, rezultat će sadržavati frekvenciju nosača plus frekvenciju modulatora za jedan popratni pojas i frekvenciju nosača minus frekvenciju modulatora za drugi popratni pojas - to se naziva zbroj i razlika. Izgled sinusnog vala i sinusa sa popratnim harmonicima možemo vidjeti na **slikama 15. i 16.**



Slika 15 Nositelj, sinusni val, F3 – 174 Hz



Slika 16 Nositelj, sinus F3 - 174 Hz sa popratnim harmonicima, dobivenim primjenom FM sinteze

5. Programski jezik FAUST

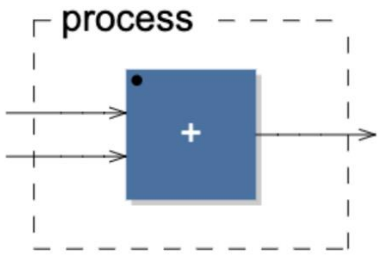
Faust je funkcijski programski jezik za sintezu i obradu zvuka. Pogodan je za dizajn sintesajzera, glazbenih instrumenata, audio efekata i sl. (Faust manual, siječanj 2020).

5.1. Razvoj i primjena

Faustom je moguće programirati aplikacije za obradu signala visokih performansi, audio *plugin*-ove i to za široku paletu platformi i standarda. Osim toga, pogodan je i za korištenje u edukaciji i istraživanju, na pozornici i umjetničkim produkcijama. Faust teži ka pružanju adekvatne notacije koja će opisati procesor signala iz matematičke perspektive i što je moguće više zadržati odmaknutost od detalja implementacije. Prevoditelj transformira Faustov kôd u njegov npr. C++ ekvivalent. Tvorci i razvijatelji Fausta tvrde da tako generiran kôd, nakon prevođenja, katkada ima bolje performanse i od ručno napisanog C++ kôda iskusnih programera. Također, Faustov kôd radi na razini uzorka (engl. *sample level*) što omogućava izradu DSP funkcija na niskoj razini, samoodrživ je i ne ovisi o nikakvoj DSP biblioteci ili sustavu izvođenja. Većina audio opreme može se modelirati kao procesor signala s audio ulazima, audio izlazima, kontrolnim signalima povezanim s klizačima, mjernim uređajima – Faust omogućava opisivanje takvog procesora signala.

Neka od ograničenja Fausta su nemogućnost efikasne implementacije algoritama koji zahtijevaju istovremeno različite vrijednosti uzorkovanja signala, također Faustova konciznost može predstavljati problem kod pokušaja definiranja kompleksnih algoritama koji sadržavaju puno rekurzivnih signala, jer obično je potrebno imati jasnu mentalnu sliku algoritma, što je u takvim slučajevima teško.

Faustova sintaksa omogućava izražavanje bilo kakvog DSP algoritma u obliku blok dijagrama. Primjer, operator + predstavlja funkciju koja prima dva argumenta (signala) i vraća 1 (**Slika 17.**):




```
process = +;
```

Slika 17 Blok dijagram, operator +

5.1.1. Meta podaci

Meta podaci omogućuju nam dodavanje elemenata u kôd koji nisu dio Faust programa. To mogu biti npr. naziv programa koji pišemo, ime autora, razne opcije vezane za prevođenje programa. Postoje tri različita tipa meta podataka:

- Globalni meta podaci: meta podaci generalno za Faust kôd
    - Počinju sa oznakom *declare*, a nakon toga slijede par ključ – vrijednost, npr.

```
declare name "Naziv programa";
```

- Meta podaci funkcije: meta podaci specifični za funkciju
    - Počinju sa oznakom *declare* iza čega slijedi naziv funkcije i par ključ – vrijednost, npr.

```
declare add author „Pero Perić“
oscilator = os.osc(440);
```

- Meta podatci UI (engl. *User Interface*): meta podaci specifični za elemente korisničkog sučelja
    - Sastoje se od para ključ – vrijednost u uglatim zagradama, a tipično se koriste za poboljšavanje izgleda korisničkog sučelje, npr.

```
[style:knob]
```

Na ovaj način ćemo umjesto određenog klizača, dobiti oblik potenciometra na elementu na kojem smo primijenili ovaj stil

5.2. Osnovna sintaksna i semantička pravila

5.2.1. Operacija kompozicije

Iako je Faust tekstualni jezik, orijentiran je načinu pisanja kôda gdje kompozicija funkcija opisuje rezultirajući blok dijagram. U tu svrhu Faust se oslanja na pet operacija kompozicije algebre blok dijagrama. Ovo su te operacije i njihova sintaksa:

sekvencijalna (engl. *sequential*)

:

paralelna (engl. *parallel*)

,

rekurzivna (engl. *recursion*)

~

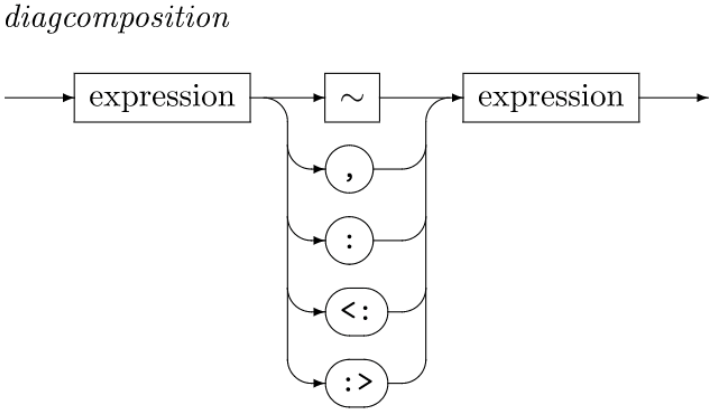
razdjelnička (engl. *split*)

<:

sumarna (engl. *merge*)

:>

Može se reći da je svaka od ovih kompozicijskih operacija jedan način spajanja dvaju blok dijagrama (**Slika 17. vrste kompozicije**).



Slika 18 Vrste kompozicije

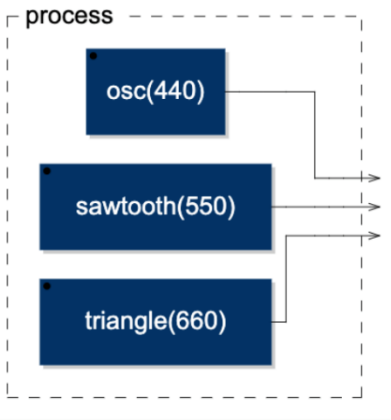
Prioritet i asocijativnost ovih pet operacija je sljedeća (**Tablica 1.**):

Tablica 1 Prioritet i asocijativnost

Sintaksa	Prioritet	Asocijativnost	Opis
izraz ~ izraz	4	lijeva	Rekurzivna kompozicija
izraz , izraz	3	desna	Paralelna kompozicija
izraz : izraz	2	desna	Sekvencijalna kompozicija
izraz <: izraz	1	desna	Kompozicija razdjele
izraz :> izraz	1	desna	Kompozicija sumiranja

5.2.2. Paralelna kompozicija

Paralelna kompozicija je asocijativna operacija: (A, (B, C)) i ((A, B), C) su ekvivalentne (**Slika 19.**).




```
import("stdfaust.lib");

process = os.osc(440), os.sawtooth(550),os.triangle(660);
```

Slika 19 Blok dijagram i primjer kôda paralelne kompozicije

5.2.3. Sekvencijalna kompozicija

Povezuje svaki izlaz A s odgovarajućim ulazom B: A[i]→[i]B

(A: (B: C)) i ((A: B): C) su ekvivalenti. Ako se ne koriste zagrade, kao u A: B: C: D, Faust koristi desnu asocijativnost i stoga interno gradi izraz (A:(B:(C:D))).

Budući da se u Faustu sve smatra signalom, sekvencijalna kompozicija može se koristiti kod slanja argumenta u funkciju (**Slika 20.**):


```
import("stdfaust.lib");

process = 440 : os.osc;
```

Slika 20 Dijagram sekvencijalne kompozicije i primjer kôda

5.2.4. Kompozicija razdjele

U razdjelničkoj kompoziciji, npr. A <: B, izlazi od A distribuiraju se na ulaze od B. Da bi operacija bila validna broj ulaza od B mora biti množitelj broja izlaza od A.

Primjer, dupliciranja izlaza oscilatora prikazan je na **slici 21.**


```
import("stdfaust.lib");

process = os.sawtooth(440) <: _,_,_;
```

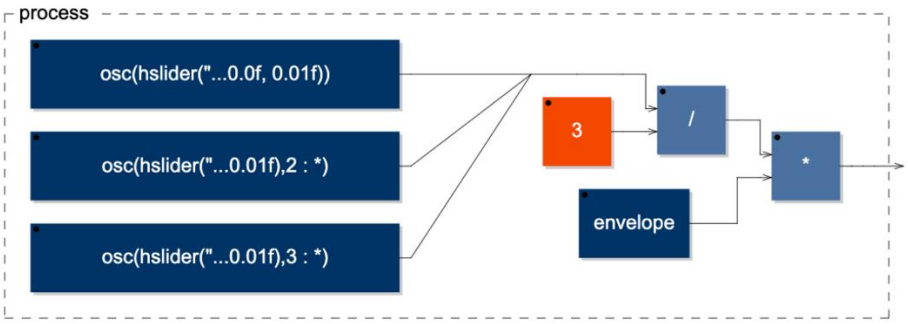
Slika 21 Dijagram razdjele i primjer kôda

Generalno, na ovaj način je moguće spajati blokove s manje izlaza sa blokovima s više ulaza (**slika 22.**):

Slika 22 Povezivanje mono (jedan izlaz) efekta sa stereo efektom (dva ulaza)

5.2.5. Kompozicija sumiranja

Broj izlaza od A mora biti množitelj broja ulaza od B. Primjer sumiranja izlaza 3 oscilatora prikazan je na **slici 23.**



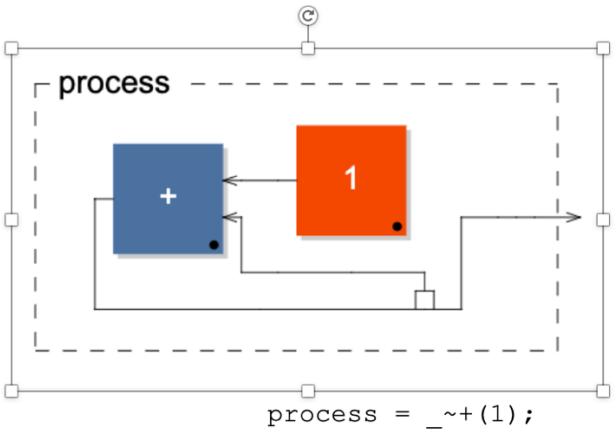

```
import("stdfaust.lib");

freq = hslider("freq",440,50,3000,0.01);

gain = hslider("gain",1,0,1,0.01);

gate = button("gate");

envelope = gain*gate : si.smoo;

process = os.osc(freq),os.osc(freq*2),os.osc(freq*3) :> /(3)*envelope;
```

Slika 23 Dijagram sumiranja i primjer kôda

5.2.6. Rekurzivna kompozicija

Rekurzivna kompozicija, npr. A~B koristi se za kreiranje povrata signala u blok dijagramu kojima izražavamo rekurzivna računanja i predstavlja najkompleksniju operaciju kada se radi o povezivanju.

Rekurzivna kompozicija se može koristiti za implementaciju vremenskog brojača (engl. *timer*) koji broji svaki sample počevši od vremena n = 0 (**slika 24.**).

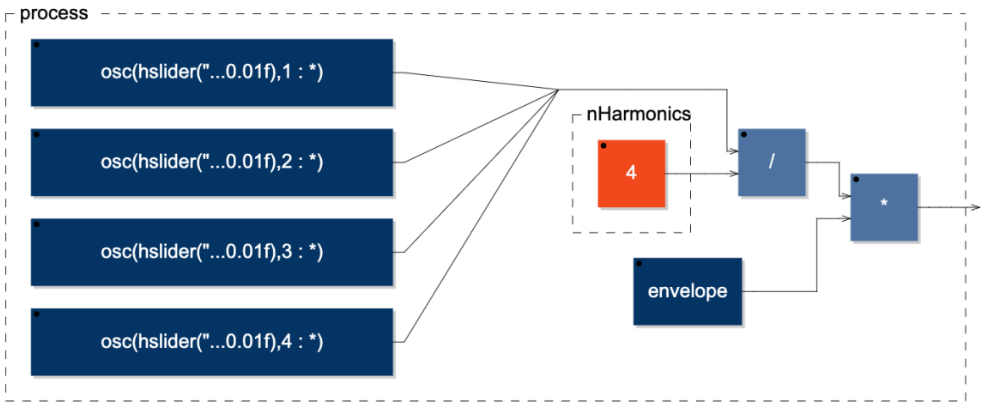


Slika 24 Dijagram rekurzivne kompozicije i primjer kôda

izlazni signal ove kompozicije biti će: 1, 2, 3, 4, …

5.2.7. Ponavljanje (iteracija)

Par ponavljanje koristimo kod paralelnog ponavljanja izraza. Primjer iteracije možemo vidjeti na **slici 25**.




```
import("stdfaust.lib");
freq = hslider("freq",440,50,3000,0.01);
gain = hslider("gain",1,0,1,0.01);
gate = button("gate");
envelope = gain*gate : si.smoo;
nHarmonics = 4;
process = par(i,nHarmonics,os.osc(freq*(i+1))) :>
/(nHarmonics)*envelope;
```

Slika 25 Dijagram funkcije iteracije i primjer kôda

5.3. Razvojna okolina

Faust ima vrlo praktičnu mogućnost razvoja u web pregledniku. Osim podržanog prepoznavanja sintakse, što je moguće postići i u editorima kao što su Atom i Visual Studio Code, glavna prednost razvijanja u browseru je mogućnost prevođenja u Faustovom online prevoditelju. Dakle, nije potrebno ništa instalirati na računalo, nije potrebno pratiti nove verzije Fausta ili podešavati sustav, dovoljno je kôd napisan u editoru po izboru, pokrenuti u Web Editoru. Ako se radi o polifonom sintesajzeru[16], dovoljno je uključiti opciju "Activate MIDI Polyphonic Mode" na "On" i jednostavno priključeni MIDI kontroler biti će prepoznat (za sada samo u pregledniku Chrome). Pokrenuti sintesajzer možemo testirati, na taj način, direktno u pregledniku. Sljedeća, vrlo efektna mogućnost, je prikaz blok dijagrama u razinama. Moguće je iscrtati u pregledniku najvišu razinu, a zatim pritiskom miša ići sve dublje u spojeve blokova.

[16] Polifoni sintesajzeri daju mogućnost sviranja više glasova (engl. *voices*) istovremeno. U praksi to znači više pritisnutih tipki na kontroleru odjednom. Nasuprot tome su monofoni instrumenti koji su ograničeni na produciranje samo jedne note u određenom periodu.

6. Opis rješenja projektnog zadatka

Praktični dio ovog rada je izrada softverskog sintesajzera koji se pokreće kao dodatak u programu domaćinu (*plugin*). Iako se mnogi softverski sintesajzeri mogu pokrenuti kao samostalne (engl. *standalone*) instance, velika većina dolazi i u obliku, a nerijetko i samo u obliku *plugin*-a. Razlog tomu je sam proces glazbene produkcije i mogućnosti koje nudi DAW. DAW zapravo predstavlja virtualni glazbeni studio, a glavne dvije komponente su zasigurno mikser i sekvencer. Mikser omogućava, u najgrubljem opisu, balansiranje raznih signala i njihovo sumiranje u završni, najčešće, dvokanalni zapis, dok je sekvencer zadužen za zapis, editiranje, sinkronizaciju, programiranje, itd. audio i MIDI zapisa.

U kontekstu teme ovog rada to su periferne stvari pa se može samo reći, da se sama glazbena kompozicija, tj. zapis nota i njihova manipulacija (mijenjanje visine tona, dužine tona, ekspresija, itd..) odvija pomoću sekvencera, a zapisuje u jedan od kreiranih kanala. Kreiranjem nove instance određenog tipa kanala u DAW-u, dobivamo i objekt kanal (engl. *track*) na kojemu ćemo otvoriti i instancu *plugin*-a. Zvučni izlaz iz sintesajzera direktno će biti spojen na izlaz kanala koji povezan na glavni (engl. *master*) izlaz iz DAW-a, a koji je opet povezan na zvučnu karticu i njen pretvarač digitalnog signala u analogni.

6.1. Programsko rješenje zvučnog modula

Izrađeni sintesajzer, programski, ima dva glavna dijela:

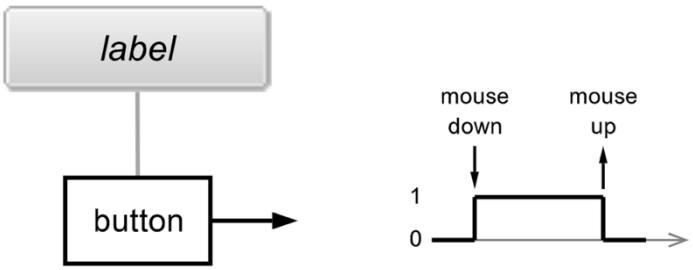
- Grafičko (korisničko) sučelje (engl. *user interface - UI*)
- Programsku logiku sintesajzera

Radi jednostavnijeg opisa kompletnog rješenja i povezivanja kôda sa elementima sučelja, u postupku opisivanja koristit će se odozgo prema dolje pristup. Bit će objašnjeno koje vrste elementa/kontrola nudi Faust, kako neke od njih rade, kako ih se postavlja u određenu hijerarhiju i grupira. Nakon opisa grafičkog sučelja objašnjavaju se neki ključni dijelovi kôda koji povezuju ranije opisane audio koncepte i funkcionalnosti na sučelju.

6.1.1. Korisničko sučelje

Faust omogućava apstraktni opis korisničkog sučelja tijekom razvoja. Takav opis je neovisan o alatima/razvojnim okvirima koji će se kasnije koristiti. Kontrole na sučelju postoje u tri oblika: diskretne, kontinuirane i organizacijske.

Pomoću diskretnih definiramo određena diskretna stanja - npr. kontrola *button*. Budući da su diskretni elementi ujedno i generatori signala *button*, kada je pritisnut generira signal vrijednosti 1, a otpušten 0. To može biti kontrolirano pritiskom tipke miša ili tipke na MIDI kontroleru. Na taj način možemo kontrolirati prolaz/stopiranje signala iz generatora zvuka (**slika 26.**).



Slika 26 Način rada elementa *button*

Kao i diskretne, kontinuirane kontrole također generiraju vrijednosti. Slučajevi gdje ih koristimo su kontroliranje vrijednosti u određenom rasponu, npr. želimo kontrolirati glasnoću signala (amplitudu zvučnog vala) između vrijednosti 0 i 1. Koristit ćemo kontinuiranu kontrolu *slider* kojoj možemo odrediti i "korak" vrijednosti u rasponu 0 - 1. Na taj način simuliramo potenciometre na hardverskom mikseru, kojima bi inače mijenjali vrijednosti na određenom modulu.

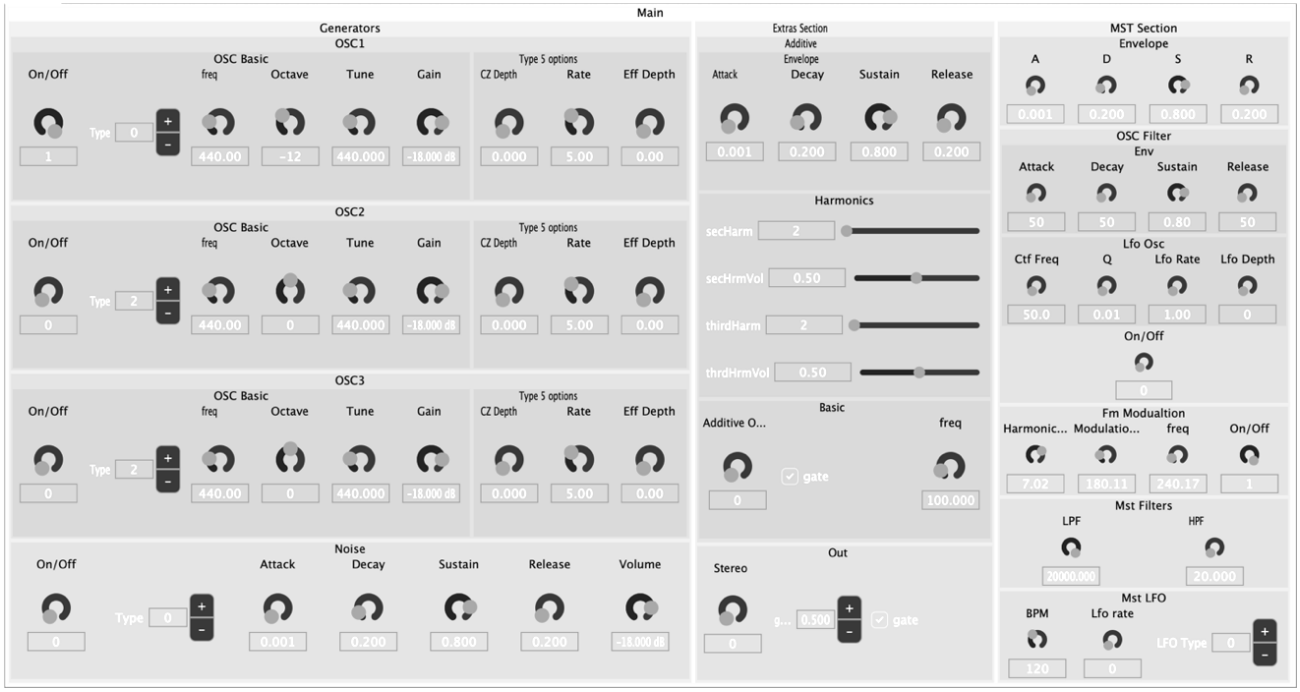
Grafičko sučelje potrebno je na neki način organizirati. Organizacijske kontrole nemaju ulogu u zvuku, nego samo postoje kako bi se definirale pozicije i grupiranja kontrola na sučelju sintesajzera. To su npr. *hgroup* (grupa sa horizontalnim slaganjem elemenata unutar grupe) i *vgroup* (grupa sa vertikalnim slaganjem elemenata unutar grupe).

Još jedna dodatna mogućnost koju omogućava Faust kod definiranja grafičkog sučelja su meta podaci, koji se u kôdu dodaju na način vrlo sličan CSS[17]-u web dizajnu. Na **slikama 27 i 28**, je prikazana usporedba grafičkog sučelja u razvojnom obliku i slika kreiranog sučelja

[17] CSS – jezik koji opisuje stil elemenata HTML (Hypertext Markup Language) dokumenta

uz pomoć apstraktnog opisa u Faust kôdu, a generirana pomoću JUCE razvojnog okvira (više o JUCE-u u poglavlju **6.2.1.2.**).

Slika 27 Korisničko sučelje tijekom razvoja u web editoru



Slika 28 Korisničko sučelje nakon kreiranja *plugin*-a

Postavljanje elemenata na točne pozicije unutar sučelja riješeno je definiranjem putanje, na način sličan onom u datotečnom sustavu. Pa npr., ako želimo postaviti LPF potenciometar (u Faustu naziv elementa je *hslider*, tipa *knob*), koji se nalazi unutar *Mst Filters* sekcije, kako je prikazano na **slici 28**, to možemo učiniti ovako:

```
cutoffControll_lowPass = hslider("h:Main/v:[2]MST
Section/h:[2]MstFilters/[0]LPF[style:knob][scale:log]",20000,
20,20000,0.001) : si.smoo;
```

Dio kôda koji se ovdje odnosi na poziciju je:

```
h:Main/v:[2]MST Section/h:[2]Mst Filters/[0]LPF
```

dakle, definirali smo da se *hslider* nalazi u sekciji *Main/MST Section/Mst Filters*.

Brojevi u uglatim zagradama definiraju poziciju elementa u odnosu na druge elemente unutar pojedine sekcije, a *h:* i *v:* označavaju da se radi o horizontalnoj odnosno vertikalnoj grupi.

Na ovom primjeru možemo također uočiti i druge spomenute meta podatke. Nakon definiranje naziva kontrole LPF, vidimo i

```
[style:knob] [scale:log]
```

ovime smo definirali da će, prvo - horizontalni klizač poprimiti kružni oblik (oblik potenciometra), i drugo - da će odziv (hod) kontrole prilikom okretanja recimo hardverskog potenciometra na klavijaturi, a koji je mapiran pomoću MIDI-ja na tu kontrolu, biti eksponencijalan. U praksi to bi značilo različiti raspon vrijednosti tokom "hoda" samog potenciometra na manjim i većim vrijednostima.

Na ovaj način moguće je definirati pozicije i oblik svih elemenata na sučelju.

6.1.2. Organizacija sučelja izrađenog sintesajzera

Korisničko sučelje izrađenog sintesajzera, kao prilog ovom radu, podijeljeno je na sekcije i pod sekcije radi lakšeg snalaženja korisnika, a njihova hijerarhija je sljedeća (**tablica 2.**):

Tablica 2 Organizacija korisničkog sučelja

Main				Glavna grupa unutar koje se nalaze sve ostale
	Generators			Sadrži izvore zvuka sintesajzera i neke od modulatora
		OSC 1/2/3		Sadrži oscilator grupu i efekte (modulacije) za određeni tip oscilatora
			OSC Basic	Osnovne kontrole oscilatora
			Type 5 Options	Dodatne kontrole za oscilator u slučaju da je odabran Type 5 - tip zvučnog vala (objašnjeno u detaljnom opisu kontrola)
		Extra section		Sadrži dodatne funkcionalnosti i neke neophodne elemente sintesajzera
		Additive		Sadrži modul za aditivnu vrstu sinteze i zvučnu omotnicu
			Envelope	Zvučna omotnica za aditivni modul
			Harmonics	Kontrole glasnoće harmonika aditivne sinteze
			Basic	Osnovni elementi za generiranje zvuka iz aditivnog modula
		Out		Glavni izlaz zvuka sintesajzera, efekt, osnovne kontrole i slično

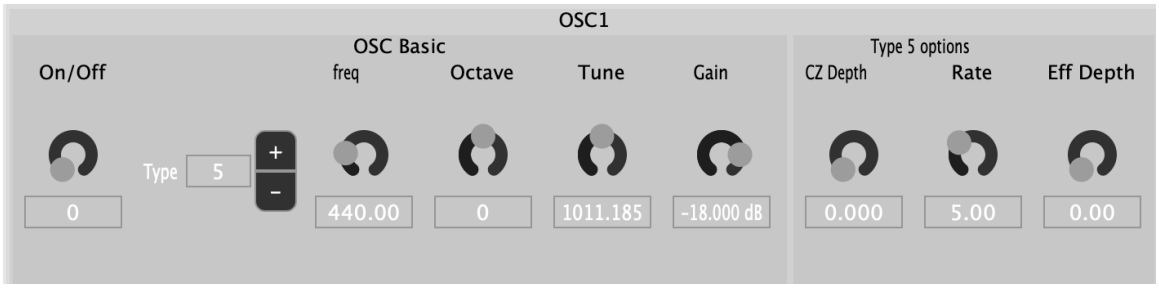
MST Section			Globalne kontrole
	Envelope		Zvučna omotnica za sva tri oscilatora
	OSC Filter		Filter i modulacija filtera za sva tri oscilatora
		Env	Zvučna omotnica za kontroliranje „otvaranja“ filtera
		LFO Osc	LFO za moduliranje filtera - u kombinaciji sa **Mst LFO**
	Fm modulation		Modul modulacije frekvencije
	Mst filters		Globalni filteri za sva tri oscilatora
	Mst LFO		Globalni LFO koji modulira sva tri oscilatora i globalni filter

6.1.3. Opis sekcija sučelja i njihovih elemenata

6.1.3.1. Sekcija OSC 1/2/3

Oscilatori predstavljaju generatore zvuka, pomoću kojih će sintesajzer stvarati zvuk. Postoje razni oblici zvučnih valova koje mogu generirati, a ti oblici u konačnici definiraju boju, kvalitetu, tzv. timbar zvuka. **Tablica 3.** opisuje elemente sekcija OSC1, OSC2 i OSC3.

Tablica 3 Elementi sekcije *OSC 1,2,3*



On/Off	uključivanje ili isključivanje oscilatora
Type[0,1,2,3,4,5]	moguće je odabrati između 6 vrsta oscilatora: Type 0 - *sine* (oscilator sinusnog vala) Type 1 - *triangle* (oscilator trokutastog vala) Type 2 - *sawtooth* (oscilator zupčastog vala)

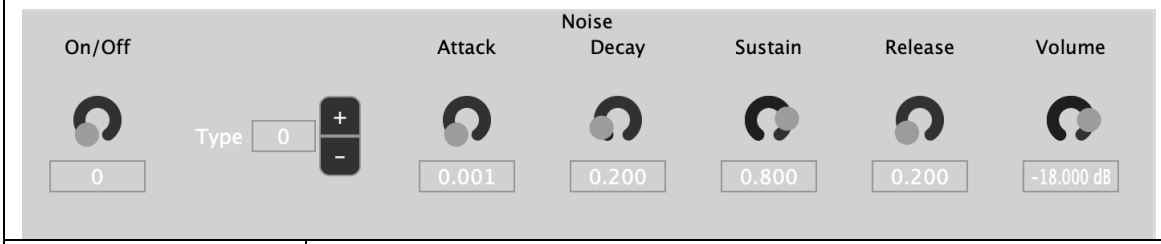
	Type 3 - *square* (oscilator kvadratnog vala) Type 4 - *noise* (generator bijelog šuma) Type 5 - *CZsquare* – oscilator koji "glumi" Casio CZ pulse oscilator. Iz Faustove biblioteke (os.)CZsquare (Faust libraries manual, siječanj 2020.)
freq, gain, gate	Ova tri parametra su grupirana, jer predstavljaju tri elementa, neophodna za implementaciju polifonog instrumenta. Faustova arhitektura daje ovu mogućnost samim korištenjem standardnih naziva na sučelju: *freq* (frekvencija), *gain* (glasnoća/amplituda signala), *gate* (onemogućavanje/omogućavanje protoka signala). Ova tri parametra su direktno povezana sa MIDI *key-on* i *key-off* događajima, a ponašaju se ovako: Kada je primljen događaj *key-on, gate* je postavljen na 1, a na *key-off* gate je 0, na taj način gate se tipično koristi kao okidač zvučne omotnice. Frekvencija u Hz, *freq*, automatski se izračunava u odnosu na notu koja je primljena (vrijednost *pitch* sadržana u MIDI *key-on, key-off* poruci) *Gain* vrijednost između 0 i 1 kao funkcija vrijednosti *velocity*, također sadržana u *key-on, key-off* poruci Ovi elementi moraju biti definirani kako bi koristili polifoniju u Faustu.
Octave	Podizanje ili spuštanje oktave oscilatora (oktava predstavlja interval između dva tona dvostruke vrijednosti frekvencije, npr. 440 Hz -> oktava iznad = 880 Hz)
Tune	mogućnost ugađanja oscilatora promjenom vrijednosti koje mogu biti i između raspona od jednog tona.

CZ Depth, Rate, Eff Depth	Kada se selektira *Type 5* omogućeno je korištenje dodatnih parametara u sekciji *Type 5 options*, a odnose se na argumente CZsquare oscilatora: CZ Depth – amplituda efekta Rate – frekvencija efekta Eff Depth – Omjer između procesiranog i ne procesiranog signala efektom Sama funkcija *CZpulse* prima dvije vrijednosti: *CZpulse(fund, index)* *fund* - trokutasti val vrijednosti između 0 i 1, koji kontrolira oscilator *index* - svjetlina oscilatora, vrijednosti su između 0 i 1 i kreću se od sinusnog oblika ka pulsnom obliku vala

6.1.3.2. Sekcija Noise

Gotovo svaki sintesajzer ima i generator šuma, koji se često koristi i za "podebljavanje" određenog kreiranog zvuka, za dizajniranje elementa bubnja (npr. doboš), za razne zvučne efekte, itd. Sekcija generatora šuma opisana je u **tablici 4.**

Tablica 4 Elementi sekcije *Noise*



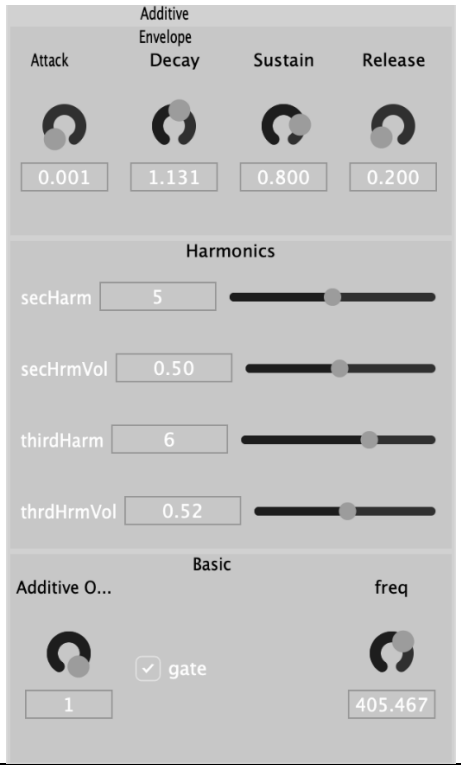
On/Off	isključivanje/uključivanje rada generatora šuma
Type[0,1]	mogućnost odabira vrste šuma Type 1 - *white noise* Type 2 - *pink noise*

Attack, Decay, Sustain, Release	Kontrole zvučne omotnice kojima oblikujemo generator šuma
Volume	određivanje glasnoće generatora šuma

6.1.3.3. Sekcija Additive

Sekcija modula aditivne sinteze opisana je u **tablici 5.**

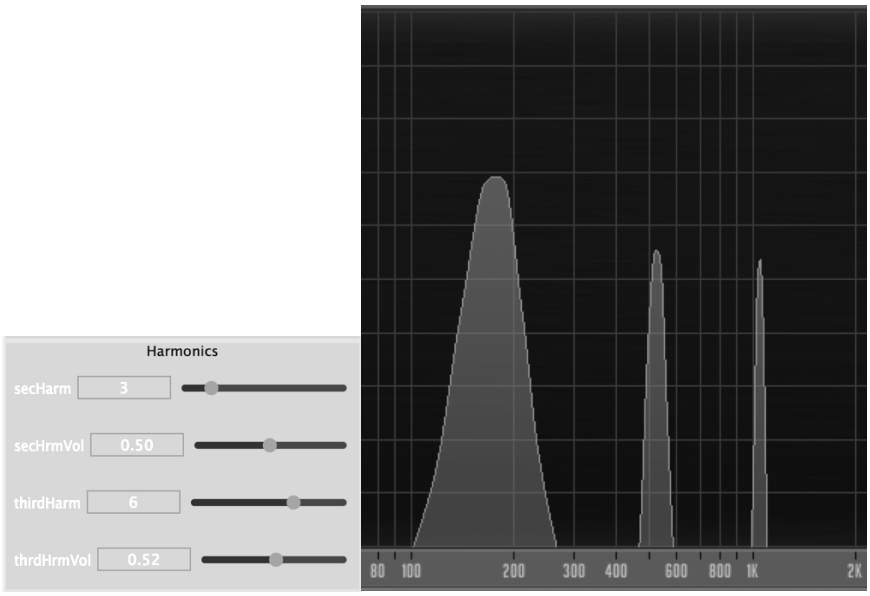
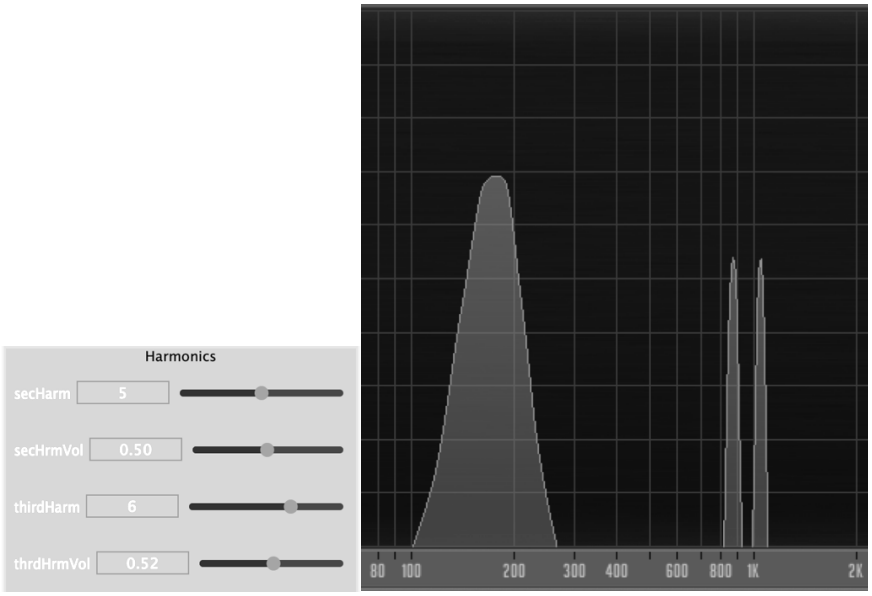
Tablica 5 Elementi sekcije *Additive*

<table>
<tr><td colspan="2"> 
</td></tr>
<tr><td>Envelope</td><td>Zvučna omotnica koja se odnosi na modul aditivne sinteze</td></tr>
<tr><td>Harmonics</td><td>Ovaj modul predstavlja jedan od primjera aditivne sinteze gdje se osnovnom zvučnom valu, u ovom slučaju sinusu dodaju viši harmonici kako bi se zvuk obogatio. Kontrole su sljedeće:

secHarm - drugi harmonik

secHamoVol - glasnoća drugog harmonika

thirdHarm - treći harmonik</td></tr>
</table>

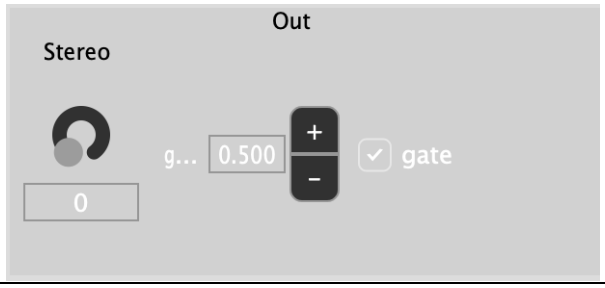
	thirdHarmVol - glasnoća trećeg harmonika Na ovaj način moguće je mijenjati odnose glasnoća harmonika iznad fundamentalnog tona i time utjecati na timbar zvuka. Jasna usporedba može se vidjeti na prikazu frekvencijskog spektra  Postavka 1 i rezultat 1  Postavka 2 i rezultat 2
Additive On/Off	Uključivanje/isključivanje modula aditivne sinteze

gate	Omogućavanje/onemogućavanje signala - okidač note
freq	Frekvencija, tj. visina osnovnog tona kojeg generira oscilator

6.1.3.4. Out

U **tablici 6.** opisani su dodatni i neophodni elementi sintesajzera, sekcije *Out*.

Tablica 6 Elementi sekcije *Out*

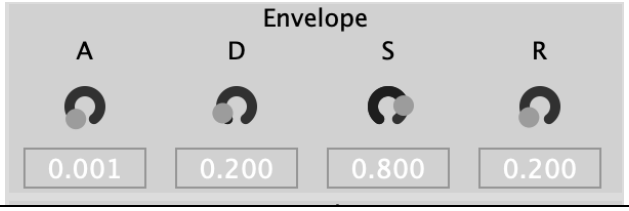


Stereo	Pretvara izlazni mono signal u pseudo stereo (vrijednost 0 = mono, vrijednost 1 = pseudo stereo) Ovdje je korišten izraz *pseudo*, jer se ne radi o dva različita signala koji će stvoriti stereo sliku, već o dva identična signala od kojih jedan kasni 500 uzoraka. Ovime dobivamo efekt širenja zvučne slike (tzv. Hass efekt).
Gain	Faustov obavezan element, da bi sintesajzer bio polifon (glasnoća globalno)
Gate	Faustov obavezan element, da bi sintesajzer bio polifon (okidač note)

6.1.3.5. Sekcija Envelope

Zvučna omotnica koja se primjenjuje na sva tri oscilatora iz sekcije *Generators* opisana je u **tablici 7.**

Tablica 7 Elementi sekcije *Envelope*



A,D,S,R	Zvučna omotnica koja se odnosi na sva tri oscilatora. Elementi rada zvučne omotnice objašnjeni su u poglavlju 3.4.2.

6.1.3.5. OSC Filter

Sekcija *OSC Filter* sadrži elemente zadužene za oblikovanje zvuka primjenom filtera i moduliranjem njihovih parametara (**tablica 8.**).

Tablica 8 Sekcija OSC Filter

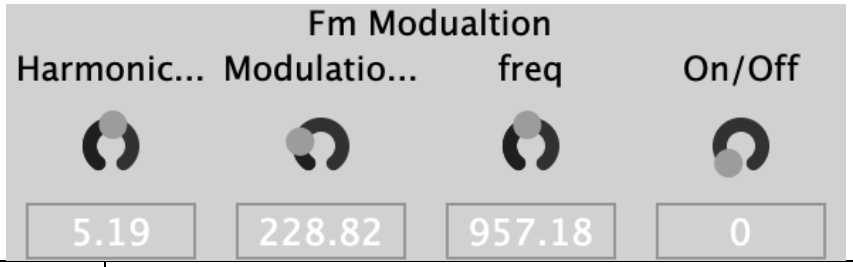
	
Attack, Decay, Sustain, Release	Zvučna omotnica koja kontrolira filter, filter se primjenjuje na sva tri oscilatora
Ctf Freq, Q	Ctf Freq (skraćeno od engl. *Cut Off Frequency*) - frekvencija rezanja, označava od koje frekvencije počinjemo ne propuštati (rezati) signal Q - širina pojasa oko odabrane rezonantne frekvencije filtera Filter se primjenjuje na sva tri oscilatora, a radi se o *lowpass* filteru iz Faustove standardne biblioteke filtera (*(fi.)resonlp*).
Lfo Rate	Frekvencija LFO-a kojim kontroliramo *Ctf Freq* kako bi dobili određeni efekt otvaranja i zatvaranja filtera kroz vrijeme.
Lfo Depth	Količina “efekta” LFO-a na filter

6.1.3.6. FM Modulation

FM modulacija odnosi se na moduliranje frekvencije oscilatora drugim oscilatorom, bitna značajka ove tehnike je da se frekvencija signala kojim moduliramo (modulator) val nositelj

(engl. *carrier*) kreće u čujnom spektru frekvencija (20 Hz – 20 kHz) Opis sekcije Fm Modulation nalazi se u **tablici 9**.

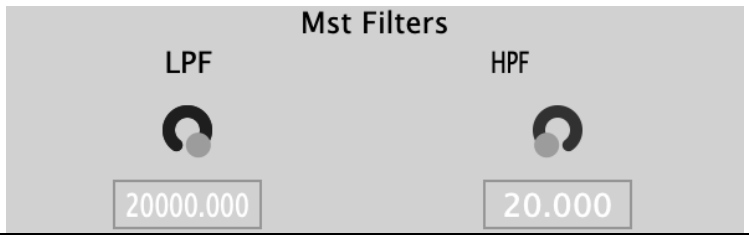
Tablica 9 Elementi sekcije *Fm Modulation*

	
Harmonicity ratio	Odnos *harmonicity ratio*-a i frekvencije nositelja definiraju konačnu vrijednost frekvencije modulatora
Modulation index	Količina modulacije primijenjene na frekvenciju signala nositelja
freq	Frekvencija nositelja
On/Off	Uključivanje/isključivanje FM modula

6.1.3.7. Mst Filters

Sekcija globalnih filtera odnosi se na sve oscilatore, a opisana je u **tablici 10**.

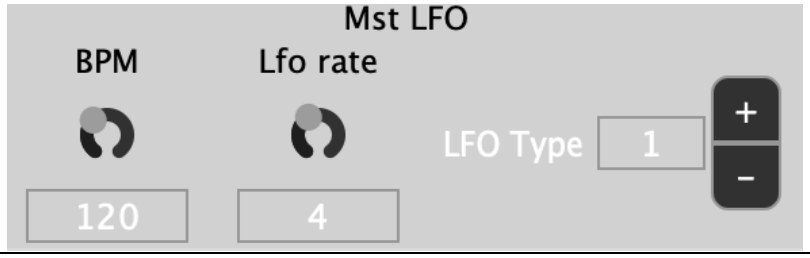
Tablica 10 Sekcija *Mst Filters*

	
LPF	*Lowpass* filter, djeluje globalno na sva tri oscilatora
HPF	*Highpass* filter, djeluje globalno na sva tri oscilatora

6.1.3.8. Mst LFO

Sekcija *Mst LFO* zadužena je za modulaciju i sinkronizaciju efekta modulacije sa tempom projekta DAW-a. Opisana je u **tablici 11**.

Tablica 11 Sekcija *Mst LFO*

	
BPM	*Beats Per Minute* - ovdje je definiran interni sat sintesajzera, prema kojemu će se voditi i vrijednosti LFO-a. Ovaj dio je nužan kako bi sintesajzer mogao biti sinkroniziran sa tempom projekta u domaćinu. Potrebno je postaviti vrijednost jednaku onoj koja je i u projektu, osim ako se ne želi postići nekakav drugi efekt.
Lfo rate	Množitelj odabranog BPM-a, određuje frekvenciju efekta LFO-a
LFO Type	Mogućnost odabira 2 tipa vala LFO-a: 0 - kvadratni val (iz Faust biblioteke - os.lf_squarewave()) 1 - trokutasti val (iz Faust biblioteke - os.lf_triangle())

6.1.4. Programska logika sintesajzera

Faust omogućava prikaz ukupnog programskog rješenja u obliku blok dijagrama. Objašnjeni su neki ključni dijelovi sintesajzera prikazom blok dijagrama i njihovim programskim ekvivalentom, a funkcionalnost koja je time dobivena, opisana je u dijelu o korisničkom sučelju.

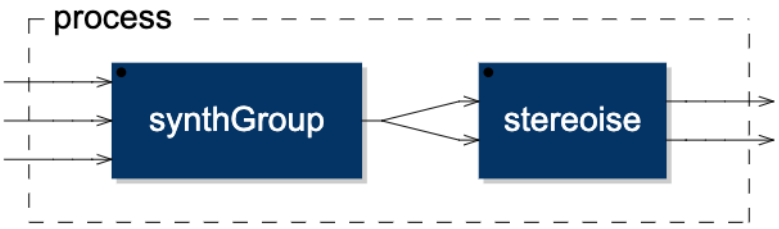
6.1.4.1. Funkcija process

Svaki program ima svoju *main* funkciju u kojoj se izvršava. U slučaju Fausta to je tzv. *process*. Sve funkcije koje se spajaju na kraju završavaju u *process*-u.

Kôd:

```
process = synthGroup <: stereoise;
```

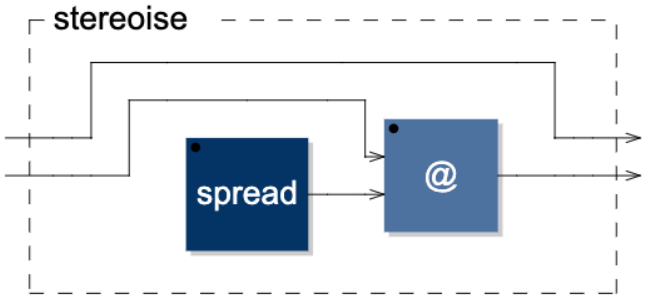
Rezultirajući blok dijagram (**slika 29.**):



Slika 29 Blok dijagram funkcije *process*

Vidljivo je kako se jednokanalni (mono) izlaz iz sintesajzera spaja pomoću operacije razdjeljivanja u funkciju *stereoise* (**slika 30.**), a daljnjim otvaranjem te funkcije može se vidjeti i njena implementacija:

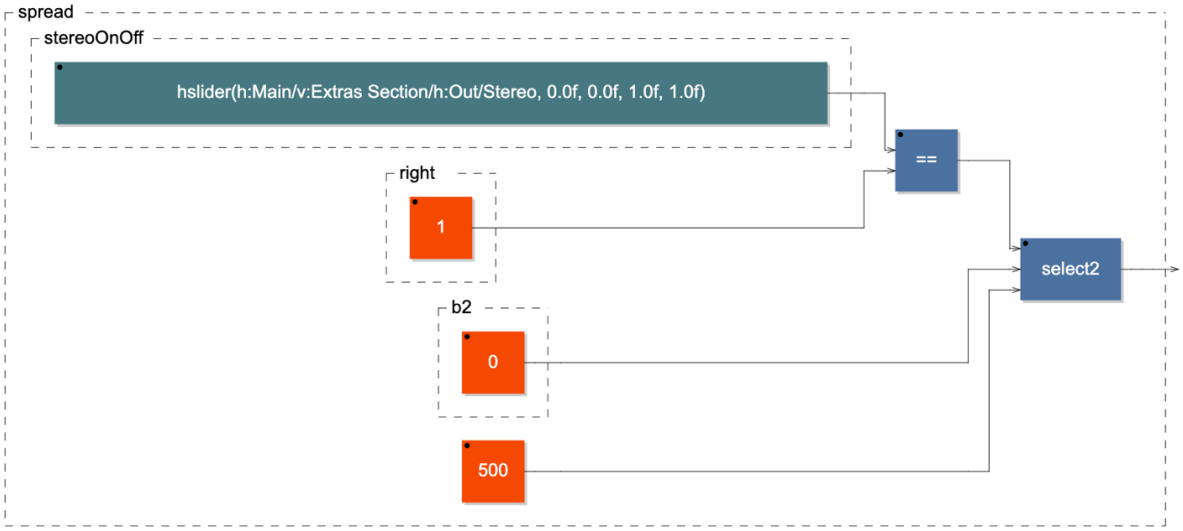
```
stereoise = _,_@(spread);
```



Slika 30 Blok dijagram funkcije *stereoise*

i dalje…

```
spread = ba.if(stereoOnOff == 1, 500, 0);
```



Slika 31 Blok dijagram funkcije *spread*

6.1.4.2. Deklaracije i uvoz

```
declare options "[midi:on][nvoices:16]";
```

Ovime obavještavamo prevoditelj da će se raditi o instrumentu koji čita MIDI poruke i koji može proizvesti 16 “glasova” istovremeno (polifoni instrument).

```
import("stdfaust.lib");
```

Ovime se uvozi Faustova standardna biblioteka, što omogućava pristup svim Faustovim bibliotekama - basics.lib, maths.lib, oscillators.lib, noises.lib, itd...

6.1.4.3. Zvučna omotnica

Definiranje zvučne omotnice kojom će se kontrolirati sva tri glavna oscilatora. Definiraju se varijable, elementi kojima ih se kontrolira (hslider), njihova pozicija i prikaz. Definirani su i rasponi vrijednosti kao i korak vrijednosti pri korištenju kontrole (kôd 1.)

```
//Envelope

attack = hslider("h:Main/v:[2]MST
Section/h:[0]Envelope/[2]A[style:knob][scale:log]",0.001,0.001,2,0.001);

decay = hslider("h:Main/v:[2]MST
Section/h:[0]Envelope/[3]D[style:knob]",0.2,0.001,2,0.001);

sustain = hslider("h:Main/v:[2]MST
Section/h:[0]Envelope/[4]S[style:knob]",0.8,0,1,0.001);

release = hslider("h:Main/v:[2]MST
Section/h:[0]Envelope/[5]R[style:knob]",0.2,0.001,6,0.001);

env(x) = en.adsr(attack,decay,sustain,release,x);
```

Kôd 1 Envelope za oscilatore *OSC 1, OSC 2, OSC 3*

Primjer:

attack ima vrijednosti -

0.001 [početna vrijednost],

0.001 [minimalna vrijednost],

2 [maksimalna vrijednost],

0.001 [“korak” vrijednosti]

Na kraju se sve šalje u funkciju *env(x)*, koja će primati parametar gate, jer ugrađena funkcija *en.adrs()* prima parametre attack, decay, sustain, release i gate.

6.1.4.4. Oscilatori

Ovim djelom kôda iterativno su izgrađene tri identične sekcije (kôd 2.) koje u sebi sadrže oscilator sa mogućnošću odabira više tipova zvučnog vala, kao i dodatne opcije u slučaju odabira 5. oscilatora (Type 5 na sučelju sintesajzera). Kod ovakvog načina "ugnježđivanja" funkcija, koje vrlo prirodno prati analogiju kreiranja sklopova i stavljanja tih sklopova jedan u drugi, postaje jasno da je funkcijski jezik vrlo prirodan i logičan izbor kod osmišljavanja jezika za DSP.

```
osc(i) = oscGroup(hgroup("[0]OSC%i",ba.selectn(6, selectWaveType,
os.osci(tune),
os.triangle(tune),
os.sawtooth(tune),
os.square(tune),
no.noise,
os.CZsquare(os.sawtooth(
tune),index * tremolo)
) * vol * oscOnOff));
Generators = par(i, 3,ba.if(fltrOnOff == 1, osc(i+1) : filterOsc,
osc(i+1))) :> / (3) * env(mainGate);
```

Kôd 2 Iterativno kreiranje oscilatora

6.1.4.5. Generator šuma

Kod generatora šuma (kôd 3.) korištena su dva ugrađena tipa - bijeli šum i ružičasti šum. Nakon definicije varijabli i pozicija na korisničkom sučelju korištena je funkcija *ba.selectn(...)* koja omogućava odabir između dva generatora. Može se primijetiti da u ovom slučaju kod elementa *nentry* koji predstavlja kontrolu odabira na sučelju imamo isti princip kao i kod definiranja vrijednosti za *hslider* u slučaju zvučne omotnice, ali u ovdje odabiremo korak 1, raspon od 0 - 1. Čime smo dobili odabir samo dvije vrijednosti.

```
//NOISE section

whiteNoise = no.noise;

pinkNoise = no.pink_noise;

//Noise on / off

noiseOnOff = vslider("h:Main/v:Generators/h:[1]Noise/[0]On/Off
[style:knob]",0,0,1,1);

selectNoiseype =
nentry("h:Main/v:Generators/h:[1]Noise/[1]Type",0,0,1,1);

noiseSelector = ba.selectn(2, selectNoiseype, whiteNoise, pinkNoise) *
noiseEnv(mainGate) ...
```

Kôd 3 Generator šuma

6.1.4.5. Modul aditivne sinteze

Iako je zbrajanje tri oscilatora iz sekcije *Generators* već samo po sebi vrsta aditivne sinteze, u sintesajzer je ugrađen dodatak koji će glumiti nešto što bi mogli naći npr. u orguljama. Ovdje direktno na početnom signalu (fundamentalna frekvencija) dodaje se 2. i 3. signal, čiji odnose je moguće mijenjati - i frekvencijski (*secHarm, thirdHarm*) i u glasnoći (*secHVol, thirdHVol*). Mijenjanjem vrijednosti ovih varijabli dobiva se drugačiji timbar zvuka jednog tona.

Ukupan signal dijeli se sa 3, budući da je ovo paralelan zbroj signala, i amplituda signala se povećava dodavanjem svakog novog vala, pa dijeljenjem sva tri signala sa 3, izbjegavamo eventualna izobličenja zvuka na pretvaraču (engl. *DAC - Digital to Analog Converter*) (kôd 4.).

```
additive = ((osc(fundamental) + osc(fundamental*secHarm) * secHVol +
osc(fundamental*thirdHarm) * thirdHVol )/3)

* en.adsr(attack,decay,sustain,release,g) : si.smoo * 10 *additiveOnOff
...
```

Kôd 4 Modul aditivne sinteze

6.2. Izvoz i uvoz u DAW

6.2.1. Izvoz

Budući da je naglasak u ovom radu bio na Faustu, kao i velika većina odrađenog posla, ostatak odrade projekta bit će opisan pod ovom kategorijom.

Projekt se sastoji od dva dijela:

- Faust kôd
- JUCE programski okvir + XCode (Apple-ov IDE, engl. *Integrated Development Environment*)

6.2.1.1. Faust kôd

Faust kôd sadrži logiku sintesajzera, kao i definiciju grafičkog sučelja. Nakon odrađenog tog glavnog dijela, ostalo je za odraditi završnu fazu prevođenja, odnosno izradu samog *plugin*-a koji će se izvršavati u DAW-u. Postoji dosta opcija u kojem obliku eksportirati kôd, budući da bi direktni eksport u jednu od verzija *plugin*-a rezultirao nepreglednim grafičkim sučeljem, koje bi zapravo imalo samo veliki broj klizača (kako bi ih pročitao DAW), bilo je potrebno eksportirati Faustov kôd u obliku JUCE projekta (**poglavlje 6.2.1.2.**).

6.2.1.2. JUCE programski okvir

JUCE je programski okvir, djelomično otvorenog kôda i koristi se za razvoj aplikacija na raznim platformama. Posebno je korišten zbog svojih biblioteka korisničkog sučelja. Program *Projucer* omogućava rad sa XCode-om (i drugim IDE-ima), tako da se prevođenje izvršava pomoću XCode-a, a definiranje korisničkog sučelja, definiranje tipa projekta i sl. se odrađuje u Projucer-u.

6.2.2. Prevođenje

Iako je dio završnog rada u kojem se koristio JUCE i XCode zahtijevao i dio programiranja, uvršten je ovdje u prevođenje, jer se sve zapravo odnosilo na prilagođavanje i pripremanje završnog prevođenja. Proces je objašnjen u nastavku.

Nakon napisanog Faust kôda, izvezen je JUCE projekt koji u sebi sadrži i Faust kôd preveden u C++ datoteku. Dio programiranja bilo je editiranje C++ datoteke. Razlog za to bilo je ponašanje sučelja nakon prvog prevođenja u audio *plugin*. Budući da su dimenzije sintesajzera izlazile izvan okvira ekrana, bilo je potrebno izmijeniti veličinu kontrola na sintesajzeru kako bi se smanjila ukupna veličina sučelja.

U Projucer-u su podešene postavke kojima se označava da se radi o instrumentu *plugin* - u, verzija C++ jezika, ciljani oblik eksporta *plugin*-a (*.component*). Nakon odrađenog prevođenja dobiven je *plugin* oblika *.component*, što je Apple-ov format audio *plugin*-a.

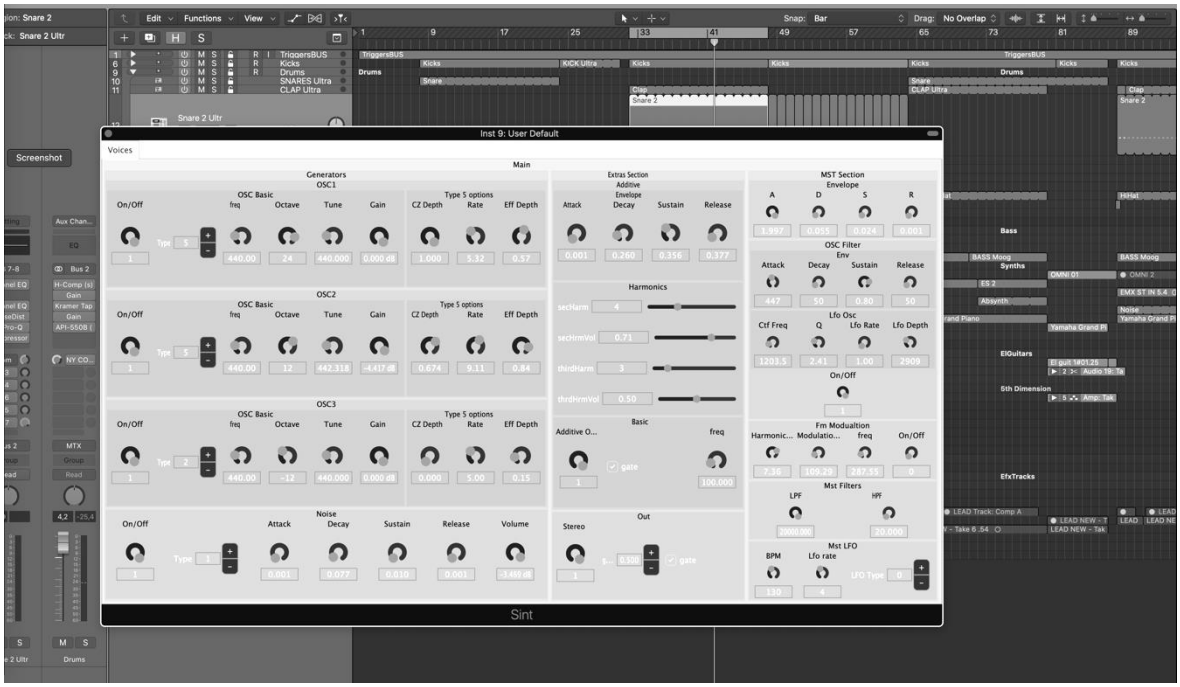
6.2.3. Uvoz u DAW

Uvoz ili instalaciju *plugin* - a predstavlja jednostavno kopiranje dobivene datoteke u direktorij *Library/Components* kojeg će DAW Logic X skenirati prilikom podizanja (potrebno je prije toga obrisati *com.apple.audiounits.cache*). Sintesajzer će se pokazati u meniju s ostalim audio *plugin*-ovima u Logic X-u.

6.3. Validacija ostvarenosti hipoteze

Izrađeni sintesajzer predstavlja potvrdu hipoteze upotrebljivosti Fausta kao programskog jezika pogodnog za izradu upotrebljivog sintesajzera u standardnom kontekstu glazbene produkcije. Uz određenu opću podlogu teorije zvučne sinteze i s druge strane solidno razumijevanje programskih principa, te relativno kratko iskustvo u programiranju, Faust je prikladan za pothvat prvostupnika programera ili producenta programera entuzijasta, da pokuša izraditi vlastiti alat po mjeri i točnim karakteristikama za koje je zainteresiran.

Kreirani *plugin* testiran je u DAW-u LogicX (**slika 32.**), gdje je uklopljen u glazbeni aranžman. Kao što je već spomenuto, njegove zvučne karakteristike nisu na razini komercijalnih sintesajzera, ovdje se ipak radi o generatorima zvuka vrlo bazičnih oblika. Međutim, postojeći set osnovnih tipova modulacije i dovoljan broj oscilatora, omogućit će zvučnu sintezu raznolikog tipa. Pa tako koristeći par instanci ovog sintesajzera, možemo pokriti više uloga unutar aranžmana – ritam sekciju (npr. bas bubanj, doboš, itd.), glazbeni tepih raznih boja i energetskih razina, razne zvučne efekte, zvuk pogodan za tzv. vodeće linije (glavne melodije), ritmičke pulsacije, i još dosta toga. Može se zaključiti da ovakav sintesajzer može naći svoju ulogu u suvremenoj produkciji, a isto tako može biti i jednostavno nadograđen.



Slika 32 Izrađeni sintesajzer

Faust je adekvatan za puno više od prezentiranog ovim radom, dubina koju nudi u izradi DSP algoritama, kreiranju vlastitih oscilatora, ne ostavlja ga iza veterana kao što je C++, štoviše nadopunjuje i obogaćuje čitavi proces izrade audio sintesajzera.

Zaključak

Područje softverske zvučne sinteze je multidisciplinarno - obuhvaća matematiku, glazbenu teoriju, akustiku, obradu digitalnog signala, audio tehnologiju, programiranje… Bez obzira na kompleksnost projekta, pojednostavljenje procesa izrade je svakako dobrodošlo, pogotovo u slučaju kada se želi tek zakoračiti u ovo područje.

Prvi susret sa ovim programskim jezikom, pokazao je kako je Faust izrazito praktičan i upotrebljiv za nekoga sa samo tri godine iskustva u programiranju. Izrada ovakvog sintesajzera korištenjem standarda kao što je C++ bila bi uvelike otežana, značajno produljena, a možda čak i prevelik zalogaj za programera prvostupnika.

Iako se radi o funkcijskom programskom jeziku, kakvog nije uobičajena praksa odslušati na kolegijima preddiplomskog studija, solidno poznavanje i iskustvo u radu sa objektno orijentiranim i skriptnim jezicima dat će dovoljno podloge za samostalno upoznavanje i rad s Faustom, uz korištenje Faustovog priručnika za upotrebu. Sama izrada, naravno, pretpostavlja i podlogu u znanju iz drugih povezanih područja, ali takav je slučaj i s bilo kojim drugim jezikom. Faust, koji je relativno jednostavan za savladati programeru početniku, možda samo korak do grafičkog načina programiranja, kao što su PureData ili Max, a s druge strane ostavlja mogućnost moćne obrade digitalnog signala i izrade kompleksnih algoritama na nižim razinama manipulacije kôdom.

Popis kratica

FAUST	Functional Audio Stream
DAW	Digital audio workstation
BPM	Beats per minute
MIDI	Musical Instrument Digital Interface
LFO	Low Frequency Oscillator
Hz	Herc
EDM	Electronic dance music
ADSR	Attack Decay Sustain Release
CC	Control Change
FM	Frequency Modulation
UI	User Interface
DAC	Digital to Analog Converter
IDE	Integrated Development Environment

Popis slika

Popis tablica

Popis kôdova

Literatura

[1] Pirkle, Will. Designing Software Synthesizer Plug-Ins in C++: For RackAFX, VST3, and Audio Units, 2015.

[2] Meinard Müller, Fundamentals of Music Processing Audio, Analysis, Algorithms, Applications, 2015.

[3] Delton T. Horn, Music Synthesizers a manual of design & construction, 1984.

[4] Andrea Pejrolo, Scott B. Metcalfe, Creating Sounds from scratch, 2017.

[5] Faust manual - https://faust.grame.fr/doc/manual/index.html, siječanj, 2020.

[6] Faust libraries manual - https://faust.grame.fr/doc/libraries/index.html, siječanj 2020.

[7] MIDI messages - https://www.midi.org/articles-old/about-midi-part-3-midi-messages ,siječanj, 2020.

[8] Acoustic fields - https://www.acousticfields.com/white-noise-definition-vs-pink-noise/ ,siječanj, 2020.

[9] Indiana University - https://cecm.indiana.edu/etext/synthesis/chapter4_filters.shtml ,siječanj, 2020.

[10] Salford University - http://www.acoustics.salford.ac.uk/acoustics_info/sound_synthesis/ ,siječanj, 2020.

Prilog

JUCE Projekt

Izrađeni sintesajzer – KJ_Synthesizer.component

Faust source kôd -KJ_Synth

ALGEBRA
VISOKO UČILIŠTE

SOFTVERSKA ZVUČNA SINTEZA

Pristupnik: Krešimir Jukić, 1192017087

Mentor: prof. dr. sc. Goran Đambić