Audio kompresori

Matijević, Karlo

Sveučilište Josipa Jurja Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike Osijek
Sveučilišni preddiplomski studij matematike i računarstva

Karlo Matijević

Audio kompresori

Završni rad

Osijek, 2023.

Sveučilište Josipa Jurja Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike Osijek
Sveučilišni preddiplomski studij matematike i računarstva

Karlo Matijević

Audio kompresori

Završni rad

Mentor: izv.prof.dr.sc. Domagoj Matijević
Komentor: dr. sc. Jurica Maltar

Osijek, 2023.

Sažetak

Ovaj se rad bavi audio kompresorima te njihovom implementacijom u programskom jeziku C++ koristeći JUCE razvojni okvir. Audio efekti koriste se u televiziji, filmovima, igrama i glazbenoj produkciji čime je kompresija jedna od najvažnijih efekata. U povijesti su se kompresori koristili isključivo za poboljšanje kvalitete snimaka i ispravljanje neželjenih artefakata nastalih snimanjem, no suvremeni audio efekti koriste se na mnogo različitih i kreativnih načina. Kompresori imaju različite svrhe; od smanjenja dinamičkog raspona ulaznog audio signala te sve do analogne distorzije. Oni se uvrštavaju u dinamičke audio efekte koji primjenjuju "gain" na ulazni signali na nekom prijašnje postavljenom vremenskom periodu. Način na koji se prijašnje spomenuti "gain" primjenjuje je obično opisan nelinearnom funkcijom. Spomenut ćemo različite vrste kompresora i objasniti njihove značajne karakteristike.

Ključne riječi

Audio signali, Dinamički raspon, Audio kompresija, C++, JUCE razvojni okvir

Abstract

This paper deals with audio compressors and their implementation in the programming language C++ using the JUCE framework. Audio effects are used in television, films, games and music production and compression is one of the most important effects. In the past, compressors have strictly been used to enchance audio recordings and correct unwanted recording artifacts, but modern audio effects are used in a variety of different and creative ways. Compressors have an abundance of different purposes; from decreasing the dynamic range of an input audio signal to analog distortion. They are a dynamic audio effect which applies "gain" to the input signal in a previously set period of time. The way a compressor applies "gain" is usually based on a non-linear function. We will mention the different kinds of compressors and explain their different characteristics.

Key words

Audio signals, Dynamic range, Audio compression, C++, JUCE framework

Sadržaj

1 Uvod

U svijetu glazbene produkcije, audio kompresija je izraz koji se često koristi. Neophodan je alat za kontroliranje dinamičkog raspona audio signala te se koristi za "poliranje" snimaka čineći ih profesionalnijim.

Važno je napomenuti da kompresija nije "lijek" za sve probleme koji se nalaze u prostoru obrade audio signala. Prekompresiran signal zvuči "zdrobljen" i bez razlika u razini zvuka, čineći ga beživotnim i nezanimljivim. Kombiniranje kompresije sa drugim audio efektima poput automatizacije razine signala te EQ *(eng. equalisation)* nužno je kako bi se postigao transparentan i dinamičan zvuk.

Kompresija dinamičkog raspona audio signala predstavlja mapiranje ulaznog dinamičkog raspona audio signala na neki manji raspon. Kompresija to radi tako što postoji vrijednost, takozvani "threshold" ili prag. Od praga, audio signal je smanjen za određeni "gain", dok audio signal ispod granice nije promijenjen. Na taj je način signal koji je glasniji od granice stišan dok je signal tiši od granice ostao isti; efektivno smanjujući razliku između najtišeg i najglasnijeg dijela ulaznog audio signala.

Od bubnjeva, sintisajzera, činela pa sve do akustičnih gitara, kompresija se može primijeniti na sve njih na način koji je pogodan i koji poboljšava karakteristike ulaznog signala. Također se primjenjuje na cijele skupine instrumenata ili cijeli aranžman. Prednost kompresije je što može oblikovati signal po našoj želji, čineći ga prodornijim ili pak više kontroliranim ovisno o postavljenim parametrima.

Sav programski kod ovog završnog rada se nalazi na sljedećim git repository-ima: VST sintisajzer plugin i VST kompresor plugin.

2 Audio kompresori

2.1 Što je to dinamički raspon audio signala?

Dinamički raspon audio inženjeri definiraju kao omjer amplituda najglasnijeg nedistorziranog audio signala i najtišeg čujnog signala. Zapisujemo ga u omjerima; npr. ako imamo zvučni sustav kojemu je najglasniji zvuk koji može proizvesti napona $5V$, a najtiši napona $0.1\mu V$, taj sustav ima dinamički raspon $500000 : 1$. Koristimo decibele (dB) pri opisivanju omjera dinamičkog raspona; relativna su mjerna jedinica koja opisuje eksponencijalni omjer između dvije vrijednosti. Ako se dva sustava razlikuju u vrijednosti od $1dB$, glasniji je sustav glasniji za 10^1 puta od tišeg sustava, ako je razlika $2dB$ onda je glasniji 10^2 puta itd.

$$L = 20 \cdot \log_{10}\left(\frac{5V}{0.1\mu V}\right) = 20 \cdot 5.7 = 114dB.$$

Kad pričamo o obradi audio signala, dinamički se raspon definira na isti način; razlika između dijela signala sa najnižom razinom glasnoće i dijela signala sa najvišom razinom glasnoće. Manjak dinamičkog raspona čini audio signal nezanimljivim za slušanje jer je bez ikakvih razlika u glasnoći tokom cijelog signala. Zbog tog razloga važno je održati prirodne vrhove i doline zvučnog vala pri kompresiji. Vrhove audio signala nazivamo **tranzijentima** (*eng. transient*); dijelove signala visoke amplitude i kratkog trajanja.

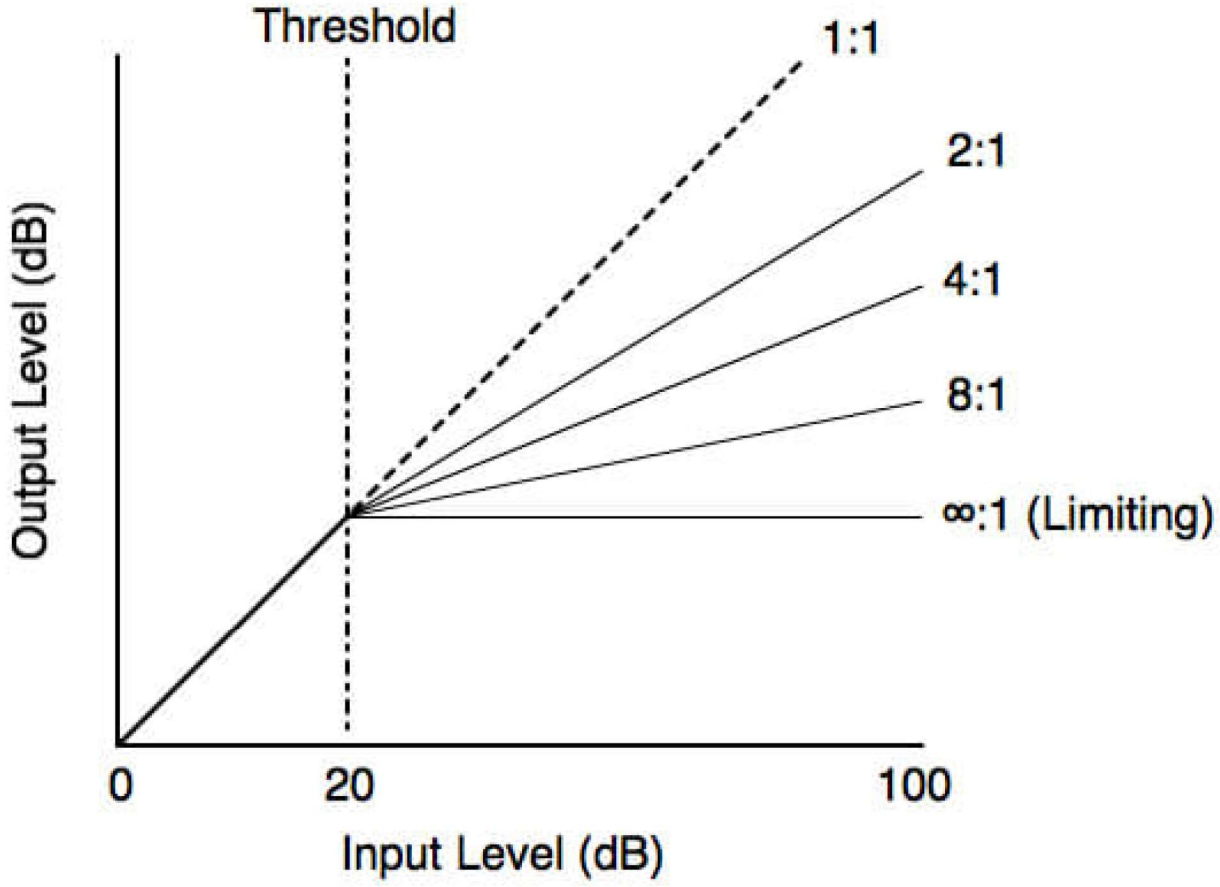
Kao što kompresori smanjuju razinu signala koji pređe određenu granicu, ekspanderi *(eng. expanders)* povećavaju razinu signala kada prijeđe granicu. Kompresori i ekspandori dijele iste parametre i kontrole.

2.2 Kontrole kompresora

Kompresori imaju određen skup kontrola koje su povezane s njegovim parametrima. U suštini, kompresor je varijabilna kontrola glasnoće, gdje količina dodane ili oduzete glasnoće ovisi o razini ulaznog signala. Prigušenje signala se primjenjuje (glasnoća se smanjuje) kada je razina signala iznad granice čime kompresor smanjuje dinamički raspon ulaznog signala. Glasnoću signala mjerimo u decibelima (dB).

Threshold *(hrv. prag)* kompresora je razina glasnoće iznad koje je kompresor aktivan. **Ratio** *(hrv. omjer)* kompresora određuje omjer ulaznog i izlaznog signala za signale koje je glasniji od granice. Omjer 5:1 znači da će kompresor za svakih 5 dB ulaznog signala koji je iznad granice povećati razinu izlaznog signala za 1 dB. **Limiting** *(hrv. ograničavanje)* je posebna vrsta kompresije gdje je omjer kompresora stavljen na $\infty : 1$. Kompresor u tom slučaju ne dopušta da signal premaši granicu; ograničava ulazni signal. Kompresori koji ograničavaju signal na opisan način nazivaju se **limiteri**.

Slika 1: Ponašanje kompresora ovisno o tome koji je omjer postavljen



Attack je parametar kojim definiramo koliko je vremena potrebno kompresoru za smanjivanje signala. **Release** je parametar kojim definiramo koliko je vremena potrebno kompresoru za povratak signala na normalnu glasnoću nakon sto je glasnoća ulaznog signala otišla ispod granice. Kompresija obično smanji generalnu glasnoću signala te onda koristimo **make-up gain** kako bi glasnoća izlaznog signala ostala vjerodostojna izvornome signalu bez kompresije. **Knee** *(hrv. koljeno)* kompresora određuje glatkoću krivulje kojom je opisana kompresija. Što je krivulja zavijenija, kompresija je manje primjetna dok krivulja sa vrhom predstavlja agresivan oblik kompresije ulaznog audio signala. Zavijena krivulja predstavlja postepen rast od originalnog 1:1 omjera (nekompresiran signal) sve do željenog omjera kompresije.

3 Vrste kompresora

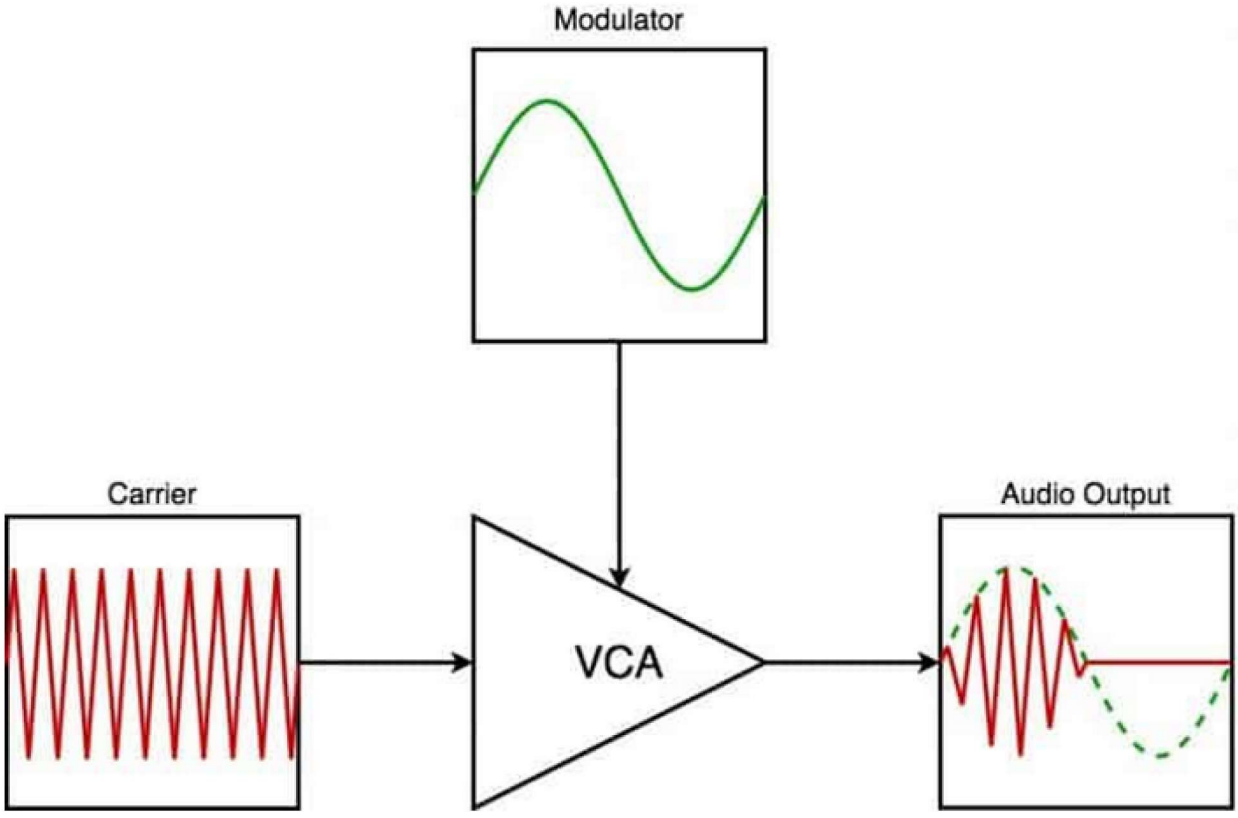
Mnogo je različitih vrsta kompresora no četiri najpoznatija su **VCA**, **optički**, **FET** i **tube** kompresori. Svaki od njih ima određene karakteristike i svrhe koje ćemo proučiti u ovom poglavlju.

3.1 VCA kompresori

VCA je kratica od *voltage-controlled amplifier* što znači da VCA kompresori koriste pojačalo signala kojim se upravlja pomoću količine napona koji se nalazi unutar sklopa. Iako im je *amplifier* u imenu, VCA kompresor ne pojačava signal nego ga čak i smanjuje. Ovakva vrsta kompresora količinu "gain"-a mijenja ovisno o tome koliko je propušteno kontrolnog napona (CV).

Različiti signali u slučaju VCA kompresora su: carrier (*hrv. nosač*), modulator i output (*hrv. izlazni*) signali. Carrier je ulazni audio signal koji je *bipolaran* (ima pozitivne i negativne amplitude). Signal modulator je CV i unipolaran je; ima samo pozitivne amplitude. Vanjski signal je signal koji je kompresiran.

Slika 2: Vizualizacija osnovnog VCA sklopa



VCA kompresori se koriste zbog vrlo niske razine distorzije signala, vrlo brzih **attack** i **release** peroida, transparentnog zvuka i ekonomične cijene. Obično se primjenjuju kod ritmičnih audio signala koji imaju puno tranzijenata. Nedostatak im je manjak jasnoće u visokim frekvencijama.

3.2 Optički kompresori

Optički kompresori koriste izvor svjetlosti te detektor svjetlosti u skopovlju za smanjivanje glasnoće. Jednostavnije rečeno, **optički** kompresori koriste otpornik koji mijenja otpor ovisno o količini svjetla koje prima (*LDR - light-dependent resistor*). LDR-ovi rade na principu poluvodičke svjetlosne provodljivosti. Energija iz fotona koja dolazi u kontakt sa poluvodičem smanjuje otpornost LDR-a; što je lampica u optičkom kompresoru intenzivnijeg svjetla, manji je otpor LDR-a. Izvor svjetlosti pretvara glasnoću audio signala u elektromagnetsku energiju što pridodaje "nelinearnom" ponašanju optičkog kompresora.

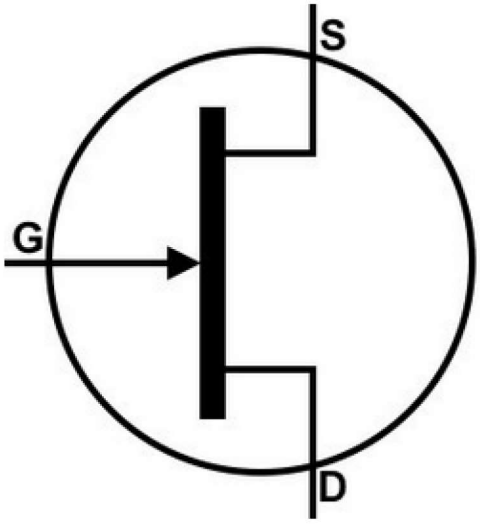
Imaju vrlo nisku distorziju signala te poveće vremenske periode za **attack** i **release**. **Attack** ove vrste kompresora ovisi o tome koje frekvencije signala su najizraženije. Transparentan zvuk te prirodno zakrivljeno koljeno (zakrivljena linija kojom se smanjuje glasnoća signala) također obilježavaju **optičke** kompresore.

3.3 FET kompresori

FET (*field-effect transistor*) kompresori su analogni kompresori koji koriste tranzistore sa efektom polja u svojem sklopovlju za smanjivanje glasnoće. Ova vrsta kompresora je iznimno dobra u očuvanju tranzijenata zvučnog signala.

Tranzistori sa efektom polja imaju 3 terminala: **source** (ulazni signal), **drain** (izlazni signal) i **gate**. Kompresori koji sadržavaju ovakve tranzistore kontroliraju koliko struje prolazi između ulaznog i izlaznog signala mijenjajući provodljivost između ulaznog signala i ulaznog signala s naponom koji je primjenjen na **gate**.

Slika 3: Tri terminala unutar tranzistora koji se koristi u **FET** kompresorima



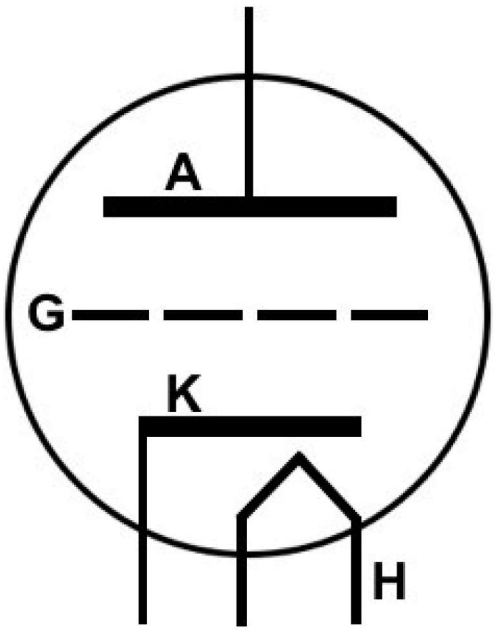
FET kompresori imaju iznimno brz **attack** i **release**. Također na signal primjenjuju nelinarnu kompresiju kojom se postiže čuvena harmonična distorzija signala. Dodatno, signal se zasićuje zbog transformera unutar sklopa. Uz ove prednosti, **FET** kompresori zahtjevaju

signal niske glasnoće; zbog ovog se uvjeta često pozadinska buka više čuje nakon intenzivne kompresije.

3.4 Tube kompresori

U ranim počecima audio produkcije, vakuumske cijevi (*eng. tube*) bile su jedini način da se pomoću neke opreme poveća glasnoća audio signala. Rade na sljedećem principu; ulazni signal prolazi kroz pojačalo koje pojačava signal te ide u detektor koji prepoznaje amplitude signala i šalje ih u kontrolno pojačalo gdje se odvija kompresija. Osnovno pojačalo u tube kompresorima je trioda koja se sastoji od anode (A), katode (K) i upravljačke rešetke (G).

Slika 4: Dizajn triode



U tube kompresorima se koristi posebna vrsta detektora zvuka koja se zove peak-responding (odziv na vrhove), koja je poznata po svojoj toplijoj i mekšoj kompresiji. Nakon kompresije, signal se ponovno pojačava i šalje na izlaz. Drugi naziv za ovakvu vrstu kompresora je *varijabilni-mu* kompresor. Izraz "varijabilni" se koristi jer **ratio** kompresora ovisi o glasnoći ulaznog signala; što je signal glasniji, omjer će se povećavati i samim time će se primjenjivati više kompresije. Tube kompresori se također koriste za dodavanje karakterističnog zvučnog potpisa (*engl. coloration*) zvuku. Poznati su po tome što dodaju tzv. harmonike (harmonijska iskrivljenja, višekratnici određene frekvencije) u zvuk, što pridonosi toplim i ugodnim zvučni karakterom.

Tube kompresori se koriste zbog svoje sposobnosti da dodaju karakter i topao osjećaj, kao i zbog njihove sposobnosti da obrade velike dinamičke raspone sa malim artefaktima.

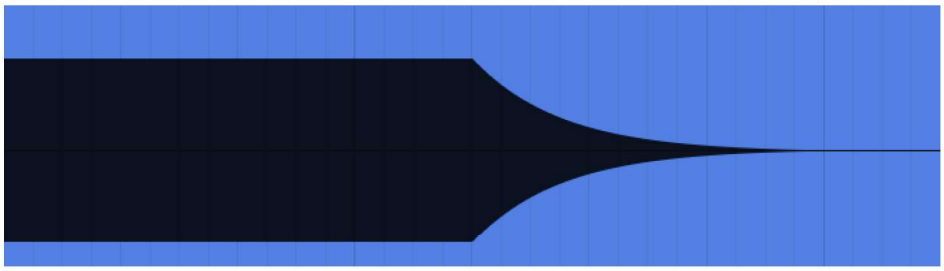
4 Primjeri kompresije

U ovom ćemo poglavlju prokomentirati primjere kompresije sa različitim parametrima kako bismo vidjeli kako kompresija utječe na neki signal.

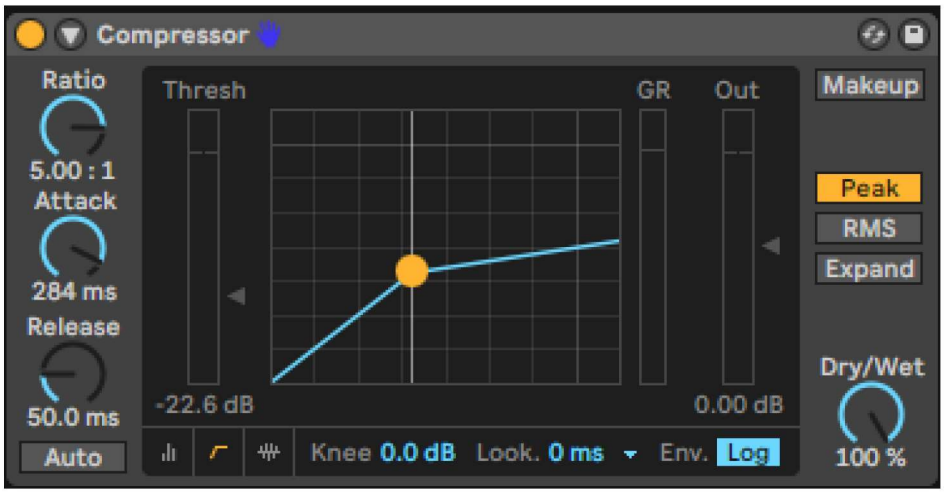
4.1 Omjer kompresora

Uočimo: svjetlo plava linija kod kompresora predstavlja nelinarnu funkciju kojom se primjenjuje "gain" na ulazni signal. Krivulja postaje sve paralelnija sa x-osi što je veći omjer kompresora.

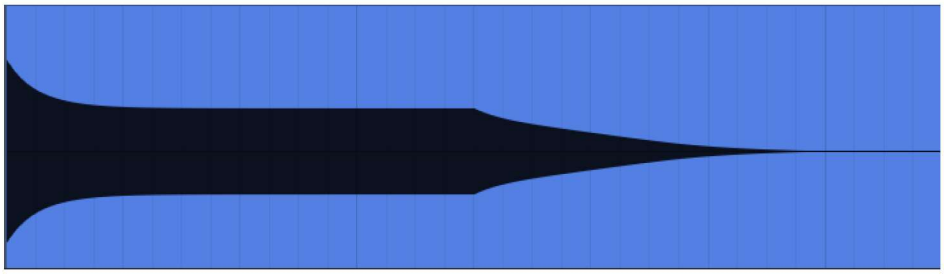
Slika 5: Ulazni signal bez kompresije



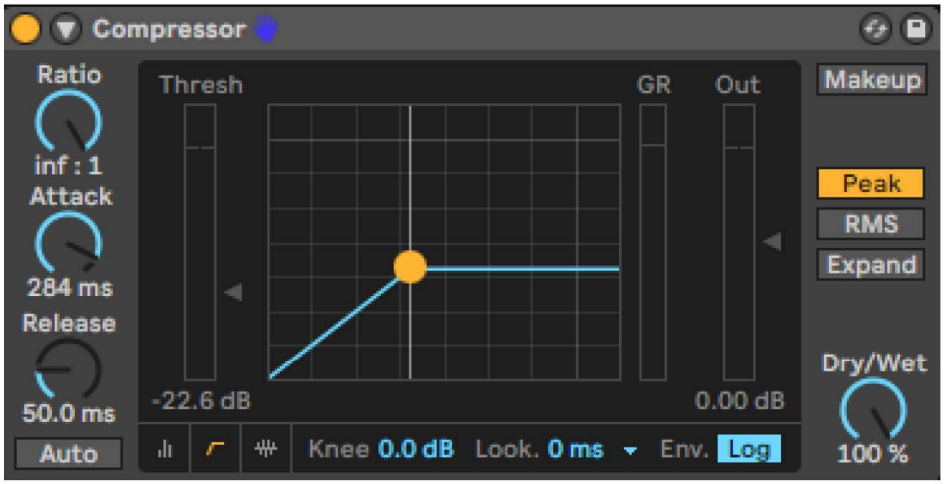
Slika 6: Kompresor sa $5:1$ omjerom



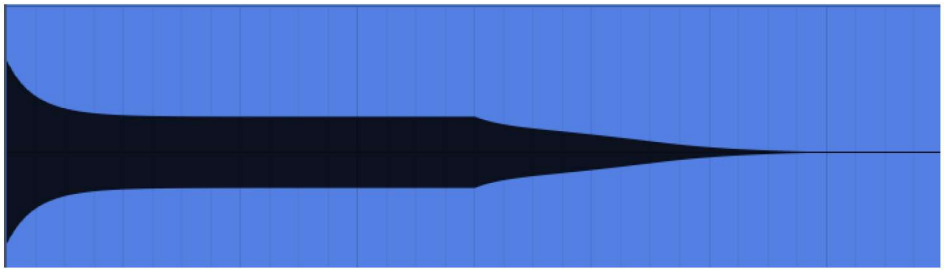
Slika 7: Kompresirana sinosoida sa $5:1$ omjerom



Slika 8: Kompresor sa $\infty:1$ omjerom (limiter)



Slika 9: Kompresirana sinosoida sa $\infty:1$ omjerom



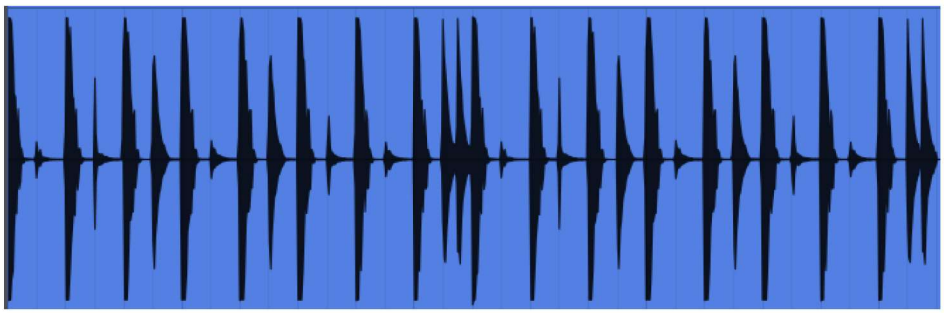
Kad usporedimo omjere $5:1$ i $\infty:1$, vidimo da je sinosuida značajnije kompresirana sa omjerom $\infty:1$ te da je razina signala sinusoide $\infty:1$ manja naspram $5:1$. Budući da su **attack** i **release** parametri jednaki te se samo mijenja **ratio**, kompresija je u isto vrijeme primijenjena na audio signal. Razlika je u amplitudama signala nakon što je primjenjena kompresija; veći omjer kompresije osigurava značajnije manju amplitudu. Kada primjenjujemo kompresiju na audio signal, dobro je imati na umu sljedeće; visoki **ratio** čini kompresiju više čujnom i agresivnijom dok su niske vrijednosti **ratio** parametra prikladinije za suptilnu kompresiju.

4.2 Attack i release kompresora

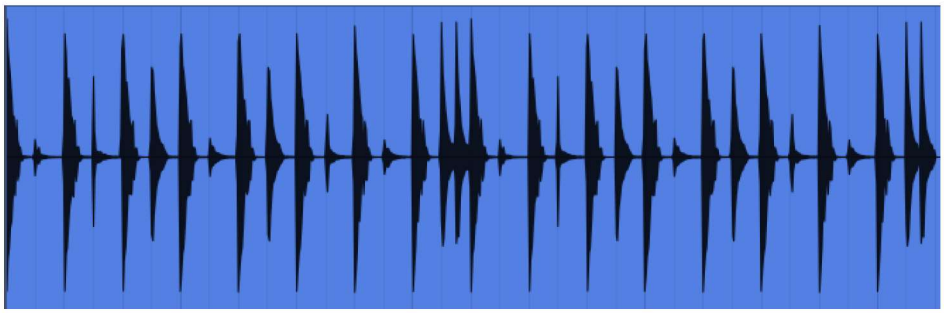
Kada pričamo o kompresorima, raspravljati o specifičnim vrijednostima **attack** i **release** parametara nema praktičnog smisla; svaki audio signal je drugačiji te zahtijeva posebne vrijednosti parametara za postizanje efektivne kompresije. U većini slučajeva, brz **attack** i **release** parametri audio signalu oplemenjuju s više "agresije" i većom percipiranom glasnoćom signala, dok sporije vrijednosti daju "glađi" zvuk.

Sada ćemo pokazati primjere sa ekstremnim vrijednostima **attack** i **release** parametara. Ulazni će signal biti snimka bubnjeva od nekoliko taktova.

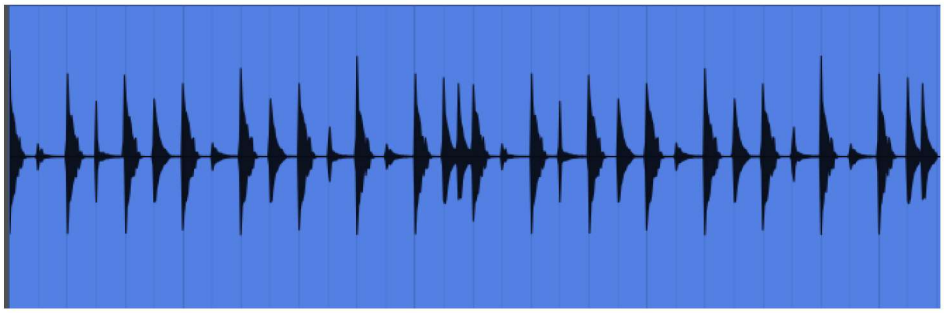
Slika 10: Ulazni signal bez kompresije



Slika 11: Kompresiran signal sa sporim **attack** parametrom

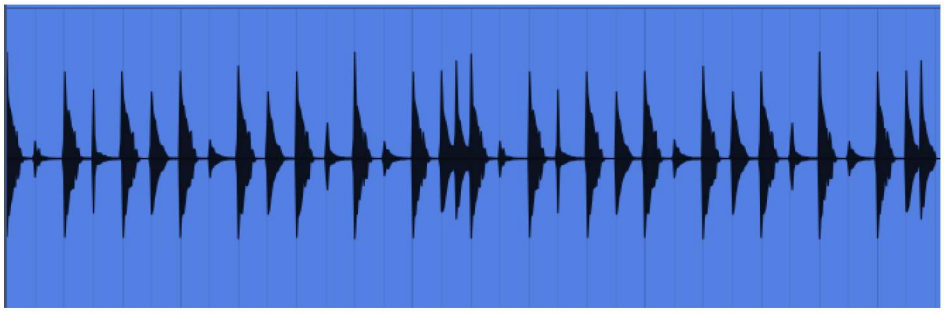


Slika 12: Kompresiran signal sa sporim **release** parametrom



Spor **release** značajno smanji dinamilčki raspon signala. Vidimo da **tranzijenti** signala nisu očuvani te da je njihova amplituda smanjena. Vidimo kako spor **attack** očuva najviše razine signala, **tranzijente**, i smanjuje razinu signala samo ako je on dovoljno dugo iznad granice.

Slika 13: Kompresiran signal sa brzim **attack** i **release** parametrima



Kompresor s brzim **attack** i **release** parametrima najznačajnije smanji dinamički raspon signala. Ovaj način kompresije se ne preporuča budući da nastaju audio artefakti pri vrlo malim vrijednostima za **attack** i **release**.Vrlo je lako primijeniti premalo ili previše kompresije. Stoga je važno unaprijed znati što s audio signalom želimo postići. Ako želimo audio signal učiniti uzbudljivijim i učiniti ga "glasnijim" ljudskom uhu, odgovor je kompresija s brzim parametrima. U drugom slučaju, ako želimo grupu audio signala učiniti sjedinjenom, dobra ideja bi bila kompresija sa sporim do srednje brzim postavljenim parametrima.

5 JUCE

JUCE je više platformski razvojni okvir otvorenog koda u programskom jeziku C++. Najviše se koristi za razvoj audio aplikacija i plugin-a. JUCE olakšava dizajniranje i razvoj samostalnih audio aplikacija ili plugin-a pomoću njegovih **DSP** (*digital signal processing*) "building block"-ova. Koristeći jedan JUCE projekt možemo izraditi VST, VST3, AU, AUv3, AAX i LV2 plugin-e koristeći isti izvorni kod. JUCE pruža svestranu apstrakciju UI-a koji se može pokrenuti na bilo kojoj korisničkoj platformi zajedno sa opcijom **hardware acceleration**-a pomoću OpenGL-a.

5.1 Dizajn audio plugina

JUCE ima mnoštvo već određenih instanca gotovo svih audio efekata. Samim time, pri izradi jednostavnijih plugina, obično nasljeđujemo te nativne klase te na njima gradimo željeni plugin. Sve parametre koje možemo mijenjati se spremaju u objekt tipa AudioProcessingValueTreeState (apvts). Svaka komponenta ima **prepare**, **process**, **update** i **reset** funkcije.

Prepare se funkcije koriste za proslijeđivanje podataka za inicijalizaciju komponente. **Process** se funkcije koriste pri render-u novog dijela (block) audio podataka. **Update** se funkcije koriste za ažuriranje parametara komponente. **Reset** se funkcije koriste za resetiranje parametara komponente.

JUCE ima sve komponente kojima je moguće napraviti i GUI za plugin. Također, ako želimo na brz način napraviti plugin, JUCE ima opciju da koristimo njegov built-in plugin editor koji nam sam napravi osnovni GUI.

5.2 Implementacija kompresora u JUCE-u

U implementaciji ćemo napraviti funkcionalni kompresor koristeći nativnu **juce::dsp::Compressor** klasu s kontrolama za **threshold** *(hrv. prag)*, **ratio** *(hrv. omjer)*, **attack** i **release**. Također ćemo imati dvije instance **juce::dsp::Gain** module-a. Jedan će nam služiti kao kontrola glasnoće ulaznog signala dok će nam drugi služiti za kontroliranje glasnoće izlaznog signala (možemo shvatiti kao neku vrstu make-up gaina). Budući da nam je cilj konstruirati funkcionalni kompresor, nećemo se previše zadržavati oko GUI-a te ćemo koristiti JUCE-ov generični **AudioProcessorEditor**.

Slika 14: createEditor metoda

```
1  juce::AudioProcessorEditor* BasicCompressorAudioProcessor::createEditor()
2  {
3      return new juce::GenericAudioProcessorEditor(*this);
4  }
```

U headeru moramo definirati value tree. Unutar ValueTree-a spremaju se sve vrijednosti parametara plugina. Na slici 14 definiramo sve module koje ćemo koristiti. U našem slučaju, imamo jedan **Compressor** i dva **Gain** module-a. Svi module-i primaju vrijednosti parametara tipa *float*.

Slika 15: Header

```
1  juce::AudioProcessorValueTreeState::ParameterLayout createParameterLayout();
2  void parameterChanged
3      (const juce::String& parameterID, float newValue) override;
4
5  juce::dsp::Gain<float> inputModule;
6  juce::dsp::Gain<float> outputModule;
7  juce::dsp::Compressor<float> compressorModule;
8  void updateParameters();
```

Želimo imati *listenera* za svaki parametar u value tree-u pa u konstruktoru **PluginProcessor**-a dodajemo sljedeće listenere.

Slika 16: Konstruktor listenera

```
1  BasicCompressorAudioProcessor::BasicCompressorAudioProcessor()
2  #ifndef JucePlugin_PreferredChannelConfigurations
3      : AudioProcessor(BusesProperties()
4  #if ! JucePlugin_IsMidiEffect
5  #if ! JucePlugin_IsSynth
6          .withInput("Input", juce::AudioChannelSet::stereo(), true)
7  #endif
8          .withOutput("Output", juce::AudioChannelSet::stereo(), true)
9  #endif
10     )
11     , treeState(*this, nullptr, "PARAMETERS", createParameterLayout())
12 #endif
13 {
14     treeState.addParameterListener("input", this);
15     treeState.addParameterListener("thresh", this);
16     treeState.addParameterListener("ratio", this);
17     treeState.addParameterListener("attack", this);
18     treeState.addParameterListener("release", this);
19     treeState.addParameterListener("output", this);
20 }
```

U konstruktoru također inicijaliziramo value tree te mu proslijeđujemo funkciju preko koje će se stvoriti svi parametri i dodati u njega.

Naravno, moramo paziti da one listenere koje smo dodali u konstruktoru uklonimo u destruktoru **PluginProcessor**-a.

Slika 17: Destruktor listenera

```
1 BasicCompressorAudioProcessor::~BasicCompressorAudioProcessor()
2 {
3     treeState.removeParameterListener("input", this);
4     treeState.removeParameterListener("thresh", this);
5     treeState.removeParameterListener("ratio", this);
6     treeState.removeParameterListener("attack", this);
7     treeState.removeParameterListener("release", this);
8     treeState.removeParameterListener("output", this);
9 }
```

createParameterLayout funkcija je kojom dodajemo parametre u value tree. Sve parametre stavljamo u vektor. Kad ih inicijaliziramo, moramo im dodijeliti ime, najmanju moguću vrijednost, najveću moguću vrijednost te početnu vrijednost parametra. Nakon inicijalizacije parametara ih stavljamo u vektor.

Budući da su **attack** i **release** jako osjetljivi parametri pogotovo pri malim vrijednostima, ima smisla normalizirati njihov raspon vrijednosti. To radimo pomoću **NormalisableRange** gdje onda pomoću **setSkewForCentre** postavljamo vrijednost koja će se nalaziti na sredini slidera. Na taj način imamo veću kontrolu nad manjim vrijednostima parametara.

Slika 18: createParameterLayout metoda

```
1  juce::AudioProcessorValueTreeState::ParameterLayout BasicCompressorAudioProcessor
2      ::createParameterLayout()
3  {
4      std::vector<std::unique_ptr<juce::RangedAudioParameter>> params;
5
6      juce::NormalisableRange<float> attackRange =
7          juce::NormalisableRange<float>(0.0f, 200.0f, 1.0f);
8      attackRange.setSkewForCentre(50.0f);
9
10     juce::NormalisableRange<float> releaseRange =
11         juce::NormalisableRange<float>(5.0f, 5000.0f, 1.0f);
12     releaseRange.setSkewForCentre(160.0f);
13
14     auto pInput = std::make_unique
15         <juce::AudioParameterFloat>("input", "Input", -60.0f, 24.0f, 0.0f);
16     auto pThresh = std::make_unique
17         <juce::AudioParameterFloat>("thresh", "Thresh", -60.0f, 10.0f, 0.0f);
18     auto pRatio = std::make_unique
19         <juce::AudioParameterFloat>("ratio", "Ratio", 1.0f, 20.0f, 1.0f);
20     auto pAttack = std::make_unique
21         <juce::AudioParameterFloat>("attack", "Attack", attackRange, 50.0f);
22     auto pRelease = std::make_unique
23          <juce::AudioParameterFloat>("release", "Release", releaseRange, 160.0f);
24     auto pOutput = std::make_unique
25         <juce::AudioParameterFloat>("output", "Output", -60.0f, 24.0f, 0.0f);
26
27     params.push_back(std::move(pInput));
28     params.push_back(std::move(pThresh));
29     params.push_back(std::move(pRatio));
30     params.push_back(std::move(pAttack));
31     params.push_back(std::move(pRelease));
32     params.push_back(std::move(pOutput));
33
34
35     return{ params.begin(), params.end() };
36 }
```

updateParameters funkcija koristi listenere koje smo definirali u konstruktoru te iz value tree-a dobivamo vrijednost promijenjenih parametara. Onda te parametre postavljamo kao vrijednosti trenutnih parametara koristeći nativne **Compressor** i **Gain** funkcije za postavljanje vrijednosti.

Slika 19: updateParameters metoda

```
1  void BasicCompressorAudioProcessor::updateParameters()
2  {
3      inputModule.setGainDecibels(treeState
4          .getRawParameterValue("input")->load());
5      compressorModule.setThreshold(treeState
6          .getRawParameterValue("thresh")->load());
7      compressorModule.setRatio(treeState
8          .getRawParameterValue("ratio")->load());
9      compressorModule.setAttack(treeState
10         .getRawParameterValue("attack")->load());
11     compressorModule.setRelease(treeState
12         .getRawParameterValue("release")->load());
13     outputModule.setGainDecibels(treeState
14         .getRawParameterValue("output")->load());
15 }
```

prepareToPlay funkcija se poziva kako bismo pluginu proslijedili sample rate i veličinu audio blockova. Također, pri naglim promjenama glasnoće, ponekad se čuju neželjeni artefakti. Taj se problem rješava takozvanim *ramp*-anjem. Signal u tom slučaju neće odmah skočiti sa x na y vrijednost, nego će imati "ramp" između te dvije vrijednosti što eliminira neželjene artefakte. *Ramping* smo primjenili na **Gain** module.

Slika 20: prepareToPlay metoda

```
1  void BasicCompressorAudioProcessor::prepareToPlay
2      (double sampleRate, int samplesPerBlock)
3  {
4      juce::dsp::ProcessSpec spec;
5      spec.maximumBlockSize = samplesPerBlock;
6      spec.sampleRate = sampleRate;
7      spec.numChannels = getTotalNumOutputChannels();
8
9      inputModule.prepare(spec);
10     inputModule.setRampDurationSeconds(0.02);
11
12     outputModule.prepare(spec);
13     outputModule.setRampDurationSeconds(0.02);
14
15     compressorModule.prepare(spec);
16
17     updateParameters();
18 }
```

processBlock funkcija proslijeđuje audio blockove modulima koji ih trebaju (**Compressor** i oba **Gain** modula).

Slika 21: processBlock metoda

```
1  void BasicCompressorAudioProcessor::processBlock
2      (juce::AudioBuffer<float>& buffer, juce::MidiBuffer& midiMessages)
3  {
4      juce::ScopedNoDenormals noDenormals;
5      auto totalNumInputChannels  = getTotalNumInputChannels();
6      auto totalNumOutputChannels = getTotalNumOutputChannels();
7
8      juce::dsp::AudioBlock<float> block{ buffer };
9      inputModule.process( juce::dsp::ProcessContextReplacing<float>(block));
10     compressorModule.process(juce::dsp::ProcessContextReplacing<float>(block));
11     outputModule.process(juce::dsp::ProcessContextReplacing<float>(block));
12
13 }
```

parameterChanged jednostavno poziva *updateParameters* koja učitava nove vrijednosti parametara.

Slika 22: parameterChanged metoda

```
1  void BasicCompressorAudioProcessor::parameterChanged
2      (const juce::String& parameterId, float newValue)
3  {
4      updateParameters();
5  }
```

setStateInformation i *getStateInformation* funkcije nam služe kako bi vrijednosti parametra bile spremljene nakon što je on ugašen. Na primjer, ako radimo na nekom projektu jedan dan i otvorimo ga sljedeći, ako funkcije get i set nisu implementirane plugin bi se vratio na default postavljene vrijednosti što naravno ne želimo.

Slika 23: getStateInformation metoda

```
1  void BasicCompressorAudioProcessor::getStateInformation
2      (juce::MemoryBlock& destData)
3  {
4      juce::MemoryOutputStream stream(destData, false);
5      treeState.state.writeToStream(stream);
6  }
```

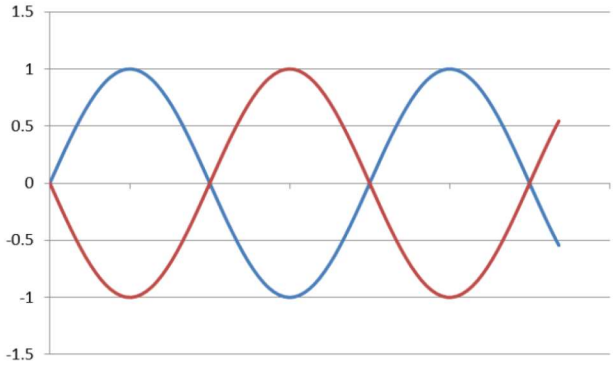
Slika 24: setStateInformation metoda

```
1  void BasicCompressorAudioProcessor::setStateInformation
2      (const void* data, int sizeInBytes)
3  {
4      auto tree = juce::ValueTree::readFromData(data, size_t(sizeInBytes));
5
6      if (tree.isValid())
7      {
8          treeState.state = tree;
9      }
10
11 }
```

5.3 Usporedba kompresora

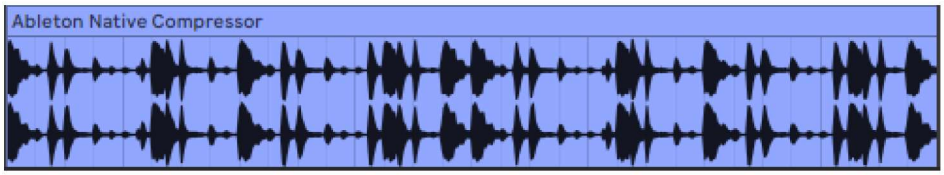
Nakon što smo konstruirali kompresor, usporediti ćemo ga sa stock kompresorom iz DAW (*Digital Audio Workstation*) Ableton. DAW su programi koji se koriste za produkciju glazbe i obradu audio signala te su opremljeni tzv. "stock" efektima i alatima koji se nalaze unutar DAW programa. Kompresori će imati iste vrijednosti parametara. Način na koji ćemo ih usporediti je *phase cancellation*. *Phase cancellation* se dogodi kada imamo dva signala koji imaju inverzne faze te se onda njihov zbrojen signal značajno smanji u glasnoći ili ako su signali identični postane tišina.

Slika 25: Primjer *phase cancellation*-a sa dvije sinusoide



Koristili smo isti ulazni signal te smo ga kompresirali sa našim JUCE kompresorom te stock Ableton kompresorom.

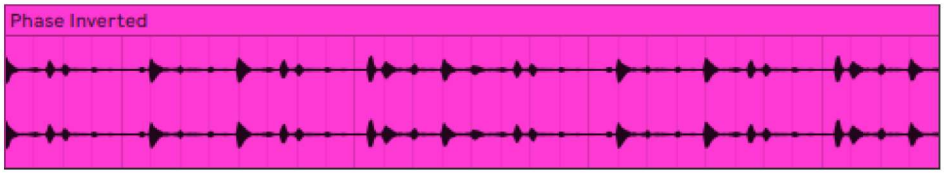
Slika 26: Signal kompresiran stock Ableton kompresorom



Slika 27: Signal kompresiran JUCE kompresorom

Kad usporedimo valove signala, vidi se da su vrlo slični. Naime, kad jednom signalu obrnemo fazu i zajedno ih snimimo, dobijemo ovo.

Slika 28: Zbroj dva kompresirana signala



Vidimo da se razina signala vrlo značajno smanjila, no nismo dobili signal tišine. Možemo zaključiti da Ableton kompresor i naš kompresor rade na sličan, ali ne i identičan način, te neće vratiti iste izlazne signale nakon kompresije s istim vrijednostima parametara.

6 Zaključak

U ovom smo se radu bavili istraživanjem audio kompresora, proučavajući njihovu primjenu u obradi audio signala i implementaciju u JUCE razvojnom okviru. Proučili smo osnovne koncepte kompresije zvuka, uključujući različite parametre kao što su **threshold**, **ratio**, **attack** i **release time**, te smo istražili kako se ti parametri mogu prilagoditi za postizanje različitih efekata kompresije.

Audio kompresori predstavljaju neizostavan dio pri obradi audio signala i te su izuzetno korisni. Međutim, važno je biti oprezan u primjeni kako bi se izbjegli neželjeni artefakti i osigurala visoka kvaliteta zvuka. Programerska implementacija kompresora zahtijeva određenu razinu znanja i stručnosti, ali pruža brojne mogućnosti za prilagodbu i optimizaciju procesa obrade zvuka.

Također smo istražili različite vrste kompresora, uključujući **VCA**, **optičke**, **FET** i **tube** kompresore, proučavajući njihove karakteristike, prednosti i nedostatke u primjeni.

Kao rezultat usporedbe kompresora utvrdili smo da je JUCE odličan alat za razvijanje VST plugina te da su njegove nativne klase vrlo korisne za osnovne funkcionalne plugine. Naime, ako želimo raditi na većem projektu, preporučamo da se moduli rade samostalno zbog ograničenja nativnih klasa.

Literatura

[1] Andrew McPherson and Joshua Reiss, Audio Effects: Theory, Implementation and Application, 2014.

[2] Will Pirkle, Designing Audio Effect Plug-Ins in C++: With Digital Audio Signal Processing Theory, 2013.

[3] Andreas Spanias, Atti Venkatraman, and Ted Painter, Audio Signal Processing and Coding, 2007.

[4] Max McAllister, What is Phase Cancellation?, 2018.

[5] Drew Swisher, Dynamic Range Compression: The 4 Goals of Compression, 2019.

[6] Arthur Fox, What Is A FET Compressor and How Does It Work?, 2023.

[7] Arthur Fox, What Is An Optical Compressor and How Does It Work?, 2023.

[8] SoundBridge, Tube Compressor, 2017.

[9] Arthur Fox, What Is A Variable-Mu (Tube) Compressor and How Does It Work?, 2023.

[10] Arthur Fox, What Is A VCA Compressor and How Does It Work?, 2023.