Sadržaj

1. Uvod 1

2. Teorija rada kompresora audio signala 2

2.1. Osnovni parametri kompresije 2

2.2. Višepojasna kompresija 4

3. Implementacija višepojasnog audio kompresora 5

3.1. JUCE framework 5

3.2. Implementacija funkcionalnosti 6

3.2.1. ProcessBlock 6

3.2.2. PrepareToPlay 7

3.2.3. CompressorBand 8

3.2.4. FIFO spremnici 10

3.2.5. Linkwitz-Riley filtri 11

3.3. Implementacija korisničkog sučelja 11

3.4. Primjer korištenja i funkcionalnost 13

3.4.1. ControlBar 13

3.4.2. Višefunkcijski zaslon 14

3.4.3. GlobalControls 17

3.4.4. CompressorBandControls 19

3.5. Upute za pokretanje 24

4. Zaključak 27

5. Literatura 28

6. Sažetak 29

7. Summary 30

8. Privitak 31

1. Uvod

Digitalna obrada zvuka igra ključnu ulogu u suvremenoj glazbenoj produkciji, omogućujući inženjerima zvuka i glazbenicima da oblikuju i poboljšaju zvuk na različite načine. Jedna od ključnih tehnika u produkciji zvuka je kompresija audio signala, koja se koristi za kontrolu njegovog dinamičkog raspona audio signala. Višepojasni kompresori posebno su moćan alat koji omogućuje neovisnu obradu različitih frekvencijskih pojaseva u zvuku, što rezultira preciznijom i višeslojnom obradom zvuka.

Pregled relevantne literature o digitalnoj obradi zvuka i višepojasnim kompresorima otkriva mnoge ključne koncepte, tehnike i implementacijske detalje. Studije poput "Multiband Compression and Expansion" (D. J. Van de Par i J. S. Abel, 2002) pružaju detaljan pregled tehnika višepojasnog komprimiranja i ekspanzije zvuka, ističući važnost precizne analize frekvencijskih pojaseva u zvuku i njihovu neovisnu obradu.

Druga istraživanja, poput "Audio Signal Compression Using Multiband Dynamic Range Compression" (N. De-Figueiredo i M. H. Brandão, 2016), fokusiraju se na algoritme i implementacijske detalje višepojasnih kompresora, istražujući različite pristupe komprimiranju i ekspanziji zvuka te njihovu primjenu u praksi.

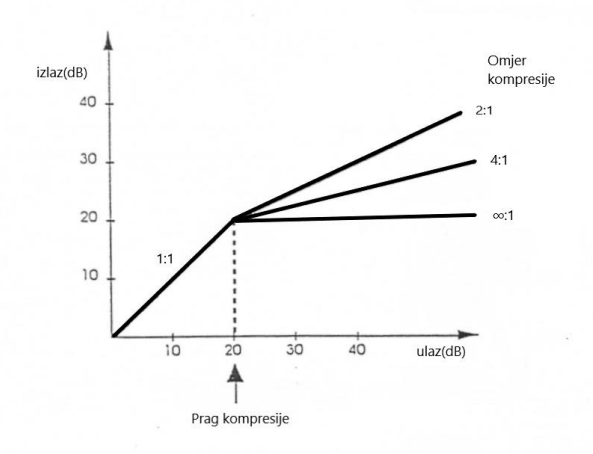
Cilj ovog rada je razviti višepojasni kompresor kao dodatak za digitalne audio radne stanice (DAW) koristeći JUCE framework za razvoj audio aplikacija. Višepojasni kompresor omogućuje korisnicima preciznu kontrolu nad dinamikom audio signala na različitim frekvencijskim pojasevima, što rezultira bogatijim i balansiranijim zvukom. Ovaj rad opisuje arhitekturu, funkcionalnosti i implementaciju razvijenog dodatka te pruža analizu algoritama i tehnika obrade zvuka koji su korišteni.

2. Teorija rada kompresora audio signala

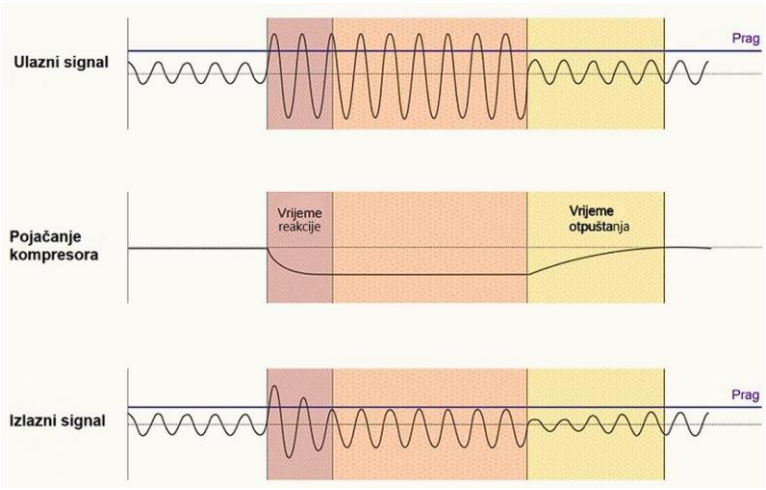
2.1. Osnovni parametri kompresije

Audio kompresori su uređaji ili softverski algoritmi koji smanjuju dinamički raspon cjelokupnog audio signala, tj. razliku razina najglasnijih i najtiših dijelova signala. To se postiže smanjenjem razine najglasnijih dijelova signala i/ili povećanjem razine najtiših dijelova. Ključni parametri koji definiraju rad kompresora uključuju:

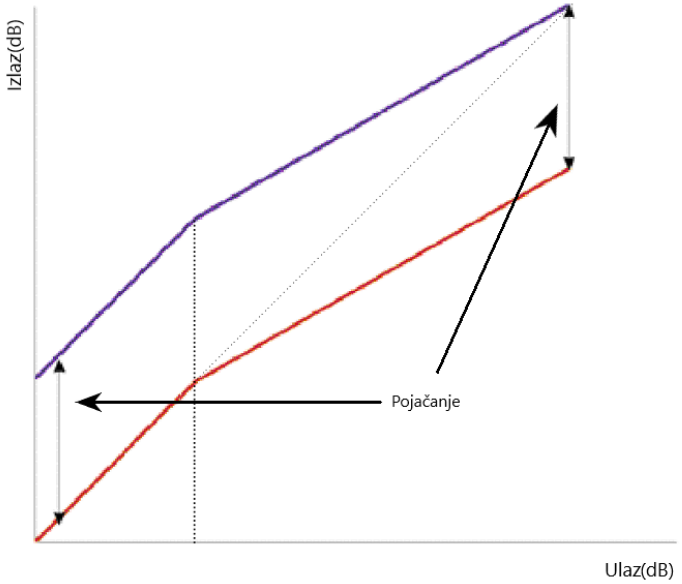
- Prag kompresije (*engl*. Threshold): Razina iznad koje kompresor počinje smanjivati razinu izlaznog signala.
- Omjer kompresije (*engl*. Compression Ratio): Omjer promjene razine ulaznog i promjene razine izlaznog signala u području djelovanja kompresora (iznad praga). Primjerice, ako promjena razine ulaznog signala od 10 dB rezultira promjenom razine izlaznog signala od samo 2dB, onda je omjer kompresije 10:2 odnosno 5:1.
- Vrijeme reakcije (*engl*. Attack Time): Vrijeme koje je potrebno kompresoru da reagira i počne smanjivati razinu izlaznog signala nakon što razina ulaznog signala prijeđe zadanu vrijednost praga.
- Vrijeme otpuštanja (*engl*. Release Time): Vrijeme koje je potrebno kompresoru da prestane smanjivati razinu izlaznog signala nakon što razina ulaznog signala padne ispod zadane vrijednosti praga.
- Pojačanje (*engl.* Make-up Gain): iznos pojačanja koje se nakon kompresije primjenjuje na audio signal kako bi cjelokupni signal imao veću razinu, a zvuk koji nastaje reprodukcijom toga signala bio glasniji



Slika 1: Prag i omjer kompresije



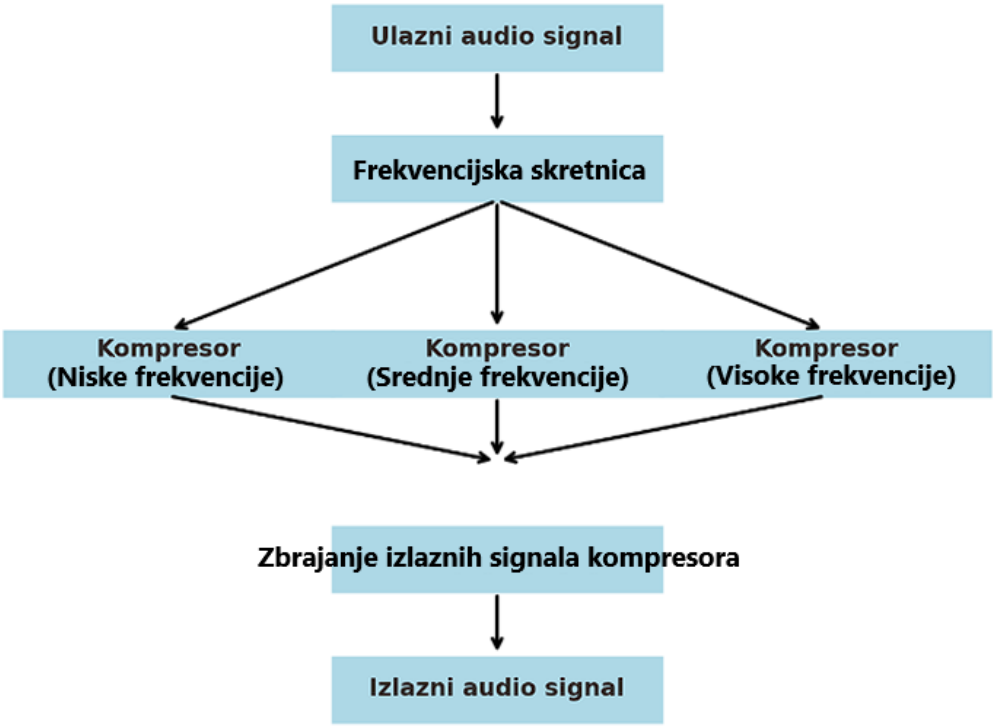
Slika 2: Prikaz vremena reakcije i vremena otpuštanja na signalu



Slika 3: Utjecaj pojačanja na signal

2.2. Višepojasna kompresija

Višepojasni audio kompresori, poput onih korištenih u ovom projektu, dijele audio signal na nekoliko frekvencijskih pojaseva, a svaki pojas se obrađuje neovisno, tj. zasebnim kompresorom kojemu je moguće namjestiti vrijednosti svih parametara opisanih u prethodnom odjeljku. To omogućuje preciznu kontrolu nad dinamičkim rasponom različitih dijelova spektra audio signala, što rezultira boljom prilagodbom zvuka specifičnim potrebama i željama korisnika.



Slika 4: Blok dijagram višepojasnog audio kompresora

U implementaciji višepojasnog kompresora ključno je korištenje odgovarajućih filtera za izdvajanje pojedinih frekvencijskih pojaseva čiji je sadržaj potrebno obraditi kompresorom. Linkwitz-Riley filtri su posebno korisni u ovoj primjeni jer osiguravaju precizno izdvajanje bez faznih pomaka i drugih izobličenja, što je ključno za očuvanje kvalitete zvuka.

Kompresori su također opremljeni parametrima za bypass, mute i solo, omogućujući korisnicima dodatnu fleksibilnost u procesu obrade zvuka. Bypass omogućuje zaobilaženje kompresora, mute isključuje zvuk određenog pojasa, dok solo omogućuje slušanje samo jednog frekvencijskog pojasa, što je korisno za fino podešavanje parametara kompresije.

3. Implementacija višepojasnog audio kompresora

3.1. JUCE framework

JUCE (Jules' Utility Class Extensions) je popularni C++ okvir za razvoj multiplatformskih audio aplikacija, razvijen od strane Juliana Storera. Ovaj okvir je postao standard u industriji za razvoj profesionalnih audio aplikacija zbog svoje jednostavnosti upotrebe, efikasnosti, visokih performansi i podrške za različite platforme.

Jedna od glavnih prednosti JUCE-a je njegova sposobnost da omogući razvoj visokokvalitetnih audio aplikacija za različite operativne sustave, uključujući Windows, macOS, Linux, iOS i Android. To je omogućeno zahvaljujući arhitekturi koja je neovisna o platformi i pruža apstrakciju za hardverske i operativne sustave.

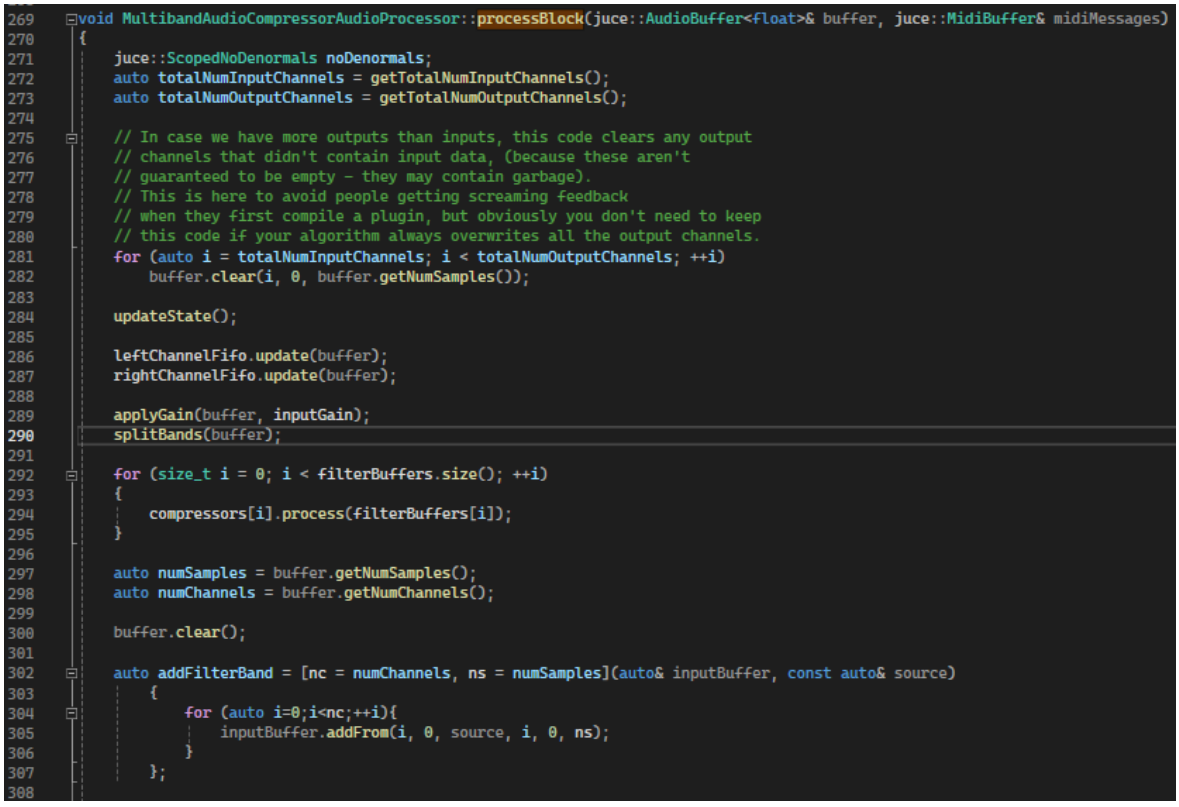
JUCE pruža bogatu kolekciju klasa, komponenata i alata koji olakšavaju razvoj audio aplikacija. Neke od ključnih značajki uključuju:

- Grafika i korisničko sučelje: JUCE pruža moćne komponente za izradu korisničkog sučelja, uključujući dugmad, prozore, trakice za kontrolu i još mnogo toga. Ove komponente su prilagodljive i podržavaju različite stilove i teme.
- Audio obrada: JUCE nudi efikasne alate za obradu audio signala, uključujući podršku za nisku latenciju, analizu i sintezu zvuka u realnom vremenu te podršku za popularne formate zvuka poput WAV, AIFF i MP3.
- Mrežno programiranje: Za aplikacije koje zahtijevaju mrežnu komunikaciju, JUCE pruža API za TCP/IP i UDP mrežno programiranje, što omogućuje jednostavnu integraciju mrežnih značajki u audio aplikacije.
- Multimedijski formati: JUCE podržava rad s različitim multimedijskim formatima, uključujući slike, videozapise i 3D grafiku. Ovo olakšava integraciju multimedije u audio aplikacije.
- Alati za razvoj: Okvir dolazi s raznim alatima za razvoj, uključujući alat za izradu korisničkog sučelja, analizu performansi i debugiranje, što olakšava razvoj i održavanje aplikacija.

3.2. Implementacija funkcionalnosti

3.2.1. ProcessBlock

Funkcija processBlock je jedna od najvažnijih funkcija unutar JUCE-ovog AudioProcessor-a. Ova funkcija se automatski poziva od strane audio engine-a svaki put kada se izvodi nova serija audio uzoraka. U našem pluginu, processBlock je odgovoran za obradu ulaznih i izlaznih audio podataka, primjenu pojačanja, dijeljenje audio signala na pojedinačne frekvencijske pojaseve, primjenu kompresije na svaki od tih pojaseva i kombiniranje obrađenih podataka u izlazni audio signal.



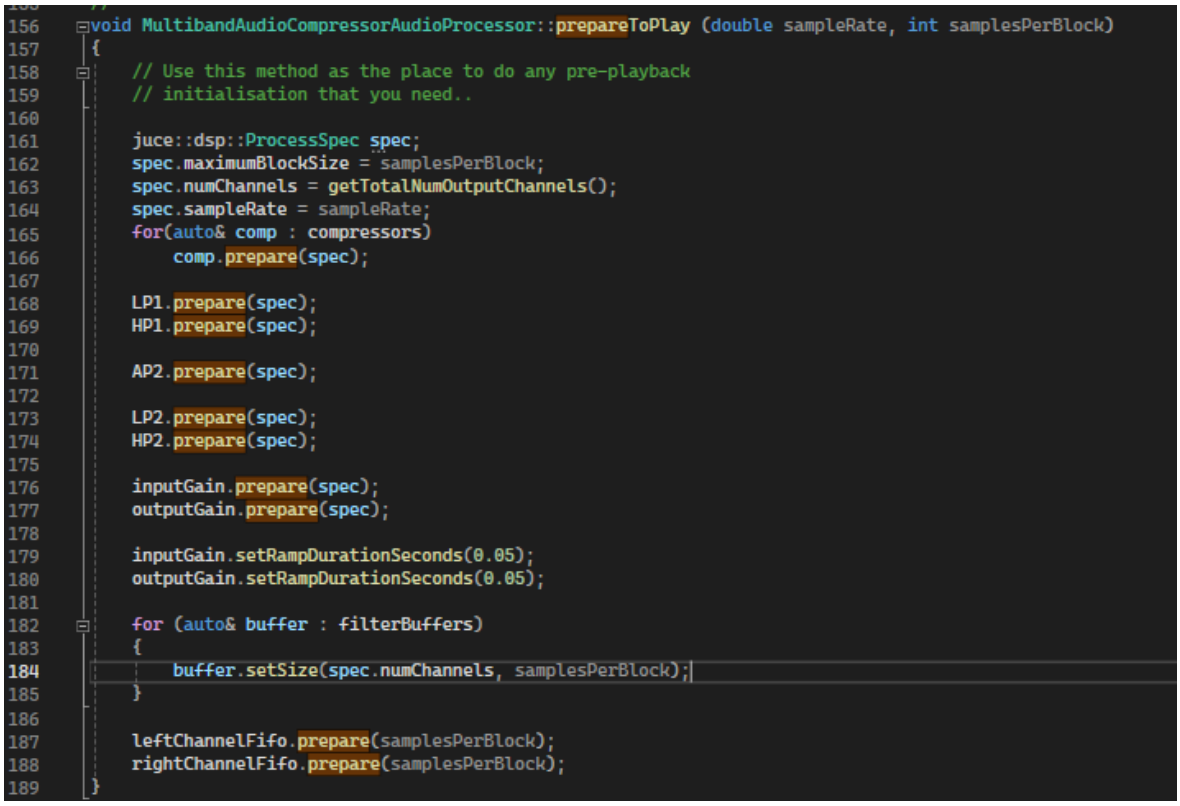
```
269  void MultibandAudioCompressorAudioProcessor::processBlock(juce::AudioBuffer<float>& buffer, juce::MidiBuffer& midiMessages)
270  {
271      juce::ScopedNoDenormals noDenormals;
272      auto totalNumInputChannels = getTotalNumInputChannels();
273      auto totalNumOutputChannels = getTotalNumOutputChannels();
274
275      // In case we have more outputs than inputs, this code clears any output
276      // channels that didn't contain input data, (because these aren't
277      // guaranteed to be empty - they may contain garbage).
278      // This is here to avoid people getting screaming feedback
279      // when they first compile a plugin, but obviously you don't need to keep
280      // this code if your algorithm always overwrites all the output channels.
281      for (auto i = totalNumInputChannels; i < totalNumOutputChannels; ++i)
282          buffer.clear(i, 0, buffer.getNumSamples());
283
284      updateState();
285
286      leftChannelFifo.update(buffer);
287      rightChannelFifo.update(buffer);
288
289      applyGain(buffer, inputGain);
290      splitBands(buffer);
291
292      for (size_t i = 0; i < filterBuffers.size(); ++i)
293      {
294          compressors[i].process(filterBuffers[i]);
295      }
296
297      auto numSamples = buffer.getNumSamples();
298      auto numChannels = buffer.getNumChannels();
299
300      buffer.clear();
301
302      auto addFilterBand = [nc = numChannels, ns = numSamples](auto& inputBuffer, const auto& source)
303          {
304              for (auto i=0;i<nc;++i){
305                  inputBuffer.addFrom(i, 0, source, i, 0, ns);
306              }
307          };
308
```

```
309     auto bandsAreSoloed = false;
310     for (auto& comp : compressors)
311     {
312         if (comp.solo->get())
313         {
314             bandsAreSoloed = true;
315             break;
316         }
317     }
318
319     if (bandsAreSoloed)
320     {
321         for (size_t i = 0; i < compressors.size(); ++i)
322         {
323             auto& comp = compressors[i];
324             if (comp.solo->get())
325             {
326                 addFilterBand(buffer, filterBuffers[i]);
327             }
328         }
329     }
330     else
331     {
332         for (size_t i = 0; i < compressors.size(); ++i)
333         {
334             auto& comp = compressors[i];
335             if (!comp.mute->get())
336             {
337                 addFilterBand(buffer, filterBuffers[i]);
338             }
339         }
340     }
341
342     applyGain(buffer, outputGain);
343 }
```

Slika 5 i Slika 6: implementacija funkcije processBlock

3.2.2. PrepareToPlay

Funkcija prepareToPlay koristi se za inicijalizaciju različitih dijelova procesora prije reprodukcije zvuka. U ovom dijelu koda, postavljaju se parametri i pripremaju se resursi potrebni za obradu zvuka tijekom reprodukcije.



```
156 void MultibandAudioCompressorAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)
157 {
158     // Use this method as the place to do any pre-playback
159     // initialisation that you need..
160
161     juce::dsp::ProcessSpec spec;
162     spec.maximumBlockSize = samplesPerBlock;
163     spec.numChannels = getTotalNumOutputChannels();
164     spec.sampleRate = sampleRate;
165     for(auto& comp : compressors)
166         comp.prepare(spec);
167
168     LP1.prepare(spec);
169     HP1.prepare(spec);
170
171     AP2.prepare(spec);
172
173     LP2.prepare(spec);
174     HP2.prepare(spec);
175
176     inputGain.prepare(spec);
177     outputGain.prepare(spec);
178
179     inputGain.setRampDurationSeconds(0.05);
180     outputGain.setRampDurationSeconds(0.05);
181
182     for (auto& buffer : filterBuffers)
183     {
184         buffer.setSize(spec.numChannels, samplesPerBlock);
185     }
186
187     leftChannelFifo.prepare(samplesPerBlock);
188     rightChannelFifo.prepare(samplesPerBlock);
189 }
```

Slika 7: implementacija funkcije prepareToPlay

Funkcija prepareToPlay inicijalizira parametre kompresora, filtra prvog i drugog reda, pojačanja ulaznog i izlaznog signala te FIFO spremnike za lijevi i desni kanal. Također, postavljaju se trajanja rampi za glatko povećanje i smanjenje razine ulaznog i izlaznog signala. Nakon ovih inicijalizacija, procesor je spreman za reprodukciju zvuka.

3.2.3. CompressorBand

Struktura CompressorBand implementira funkcionalnost pojasa kompresora unutar višepojasnog kompresora. Ova struktura omogućuje kontrolu nad postavkama kompresora, kao i procesiranje audio podataka kroz kompresor.

Ova struktura omogućuje kontrolu nad svim ključnim parametrima kompresora, kao i praćenje efektivnih razina ulaznih i izlaznih signala. Funkcija process provodi kompresiju audio podataka kroz kompresor, dok se funkcija prepare koristi za pripremu kompresora prije reprodukcije audio podataka.

```cpp
struct CompressorBand
{
    juce::AudioParameterFloat* attack{ nullptr };
    juce::AudioParameterFloat* release{ nullptr };
    juce::AudioParameterFloat* threashold{ nullptr };
    juce::AudioParameterChoice* ratio{ nullptr };
    juce::AudioParameterBool* bypassed{ nullptr };
    juce::AudioParameterBool* mute{ nullptr };
    juce::AudioParameterBool* solo{ nullptr };

    void prepare(const juce::dsp::ProcessSpec& spec)
    {
        compressor.prepare(spec);
    }
    void updateCompressorSettings()
    {
        compressor.setAttack(attack->get());
        compressor.setRelease(release->get());
        compressor.setThreshold(threashold->get());
        compressor.setRatio(ratio->getCurrentChoiceName().getFloatValue());
    }
    void process(juce::AudioBuffer<float>& buffer)
    {
        auto preRMS = computeRMSLevel(buffer);
        auto block = juce::dsp::AudioBlock<float>(buffer);
        auto context = juce::dsp::ProcessContextReplacing<float>(block);

        context.isBypassed = bypassed->get();

        compressor.process(context);

        auto postRMS = computeRMSLevel(buffer);

        auto convertToDb = [](auto input)
            {
                return juce::Decibels::gainToDecibels(input);
            };

        rmsInputLevelDb.store(convertToDb(preRMS));
        rmsOutputLevelDb.store(convertToDb(postRMS));
    }

    float getRMSInputLevelDb() const { return rmsInputLevelDb; }
    float getRMSOutputLevelDb() const { return rmsOutputLevelDb; }

private:
    juce::dsp::Compressor<float> compressor;

    std::atomic<float> rmsInputLevelDb{ NEGATIVE_INFINITY };
    std::atomic<float> rmsOutputLevelDb{ NEGATIVE_INFINITY };

    template<typename T>
    float computeRMSLevel(const T& buffer)
    {
        int numChannels = static_cast<int>(buffer.getNumChannels());
        int numSamples = static_cast<int>(buffer.getNumSamples());
        auto rms = 0.f;
        for (int chan = 0; chan < numChannels; ++chan)
        {
            rms += buffer.getRMSLevel(chan, 0, numSamples);
        }

        rms /= static_cast<float>(numChannels);
        return rms;
    }
};
```

Slika 8: Implementacija strukture pojasa kompresora

3.2.4. FIFO spremnici

FIFO (First In, First Out) spremnici koriste se za sinkronizaciju i prijenos audio podataka između različitih dijelova procesora. U ovom projektu, FIFO spremnici se koriste za učinkovito upravljanje audio podacima.

Nakon pripreme u funkciji prepareToPlay FIFO spremnici se koriste za ažuriranje audio podataka u svakom bloku procesiranja u funkciji processBlock što se može vidjeti u prethodnim potpoglavljima.

```
template<typename T>
struct Fifo
{
    void prepare(int numChannels, int numSamples)
    {
        static_assert(std::is_same_v<T, juce::AudioBuffer<float>>,
            "prepare(numChannels, numSamples) should only be used when the Fifo is holding juce::AudioBuffer<float>");
        for (auto& buffer : buffers)
        {
            buffer.setSize(numChannels,
                numSamples,
                false,
                true,
                true);
            buffer.clear();
        }
    }

    void prepare(size_t numElements)
    {
        static_assert(std::is_same_v<T, std::vector<float>>,
            "prepare(numElements) should only be used when the Fifo is holding std::vector<float>");
        for (auto& buffer : buffers)
        {
            buffer.clear();
            buffer.resize(numElements, 0);
        }
    }

    bool push(const T& t)
    {
        auto write = fifo.write(1);
        if (write.blockSize1 > 0)
        {
            buffers[write.startIndex1] = t;
            return true;
        }

        return false;
    }

    bool pull(T& t)
    {
        auto read = fifo.read(1);
        if (read.blockSize1 > 0)
        {
            t = buffers[read.startIndex1];
            return true;
        }

        return false;
    }

    int getNumAvailableForReading() const
    {
        return fifo.getNumReady();
    }
private:
    static constexpr int Capacity = 30;
    std::array<T, Capacity> buffers;
    juce::AbstractFifo fifo{ Capacity };
};
```

Slika 9: implementacija FIFO spremnika

3.2.5. Linkwitz-Riley filtri

U implementaciji višepojasnog kompresora koriste se Linkwitz-Riley filtri za razdvajanje ulaznog signala na odgovarajuće frekvencijske pojaseve. Ovi filtri omogućuju precizno razdvajanje frekvencijskog spektra audio signala i distribuciju tako razdvojenih dijelova audio signala prema zasebnim kompresorima.

```
LP1.prepare(spec);
HP1.prepare(spec);

AP2.prepare(spec);

LP2.prepare(spec);
HP2.prepare(spec);
```

Slika 10: priprema Linkwitz-Riley filtera

```
223  void MultibandAudioCompressorAudioProcessor::updateState()
224  {
225      for (auto& compressor : compressors)
226      {
227          compressor.updateCompressorSettings();
228      }
229
230      auto lowMidCuttofFreq = lowMidCrossover->get();
231      LP1.setCutoffFrequency(lowMidCuttofFreq);
232      HP1.setCutoffFrequency(lowMidCuttofFreq);
233
234      auto midHighCuttofFreq = midHighCrossover->get();
235      AP2.setCutoffFrequency(midHighCuttofFreq);
236      LP2.setCutoffFrequency(midHighCuttofFreq);
237      HP2.setCutoffFrequency(midHighCuttofFreq);
238
239
240      inputGain.setGainDecibels(inputGainParam->get());
241      outputGain.setGainDecibels(outputGainParam->get());
242  }
```

Slika 11: ažuriranje Linkwitz-Riley filtera

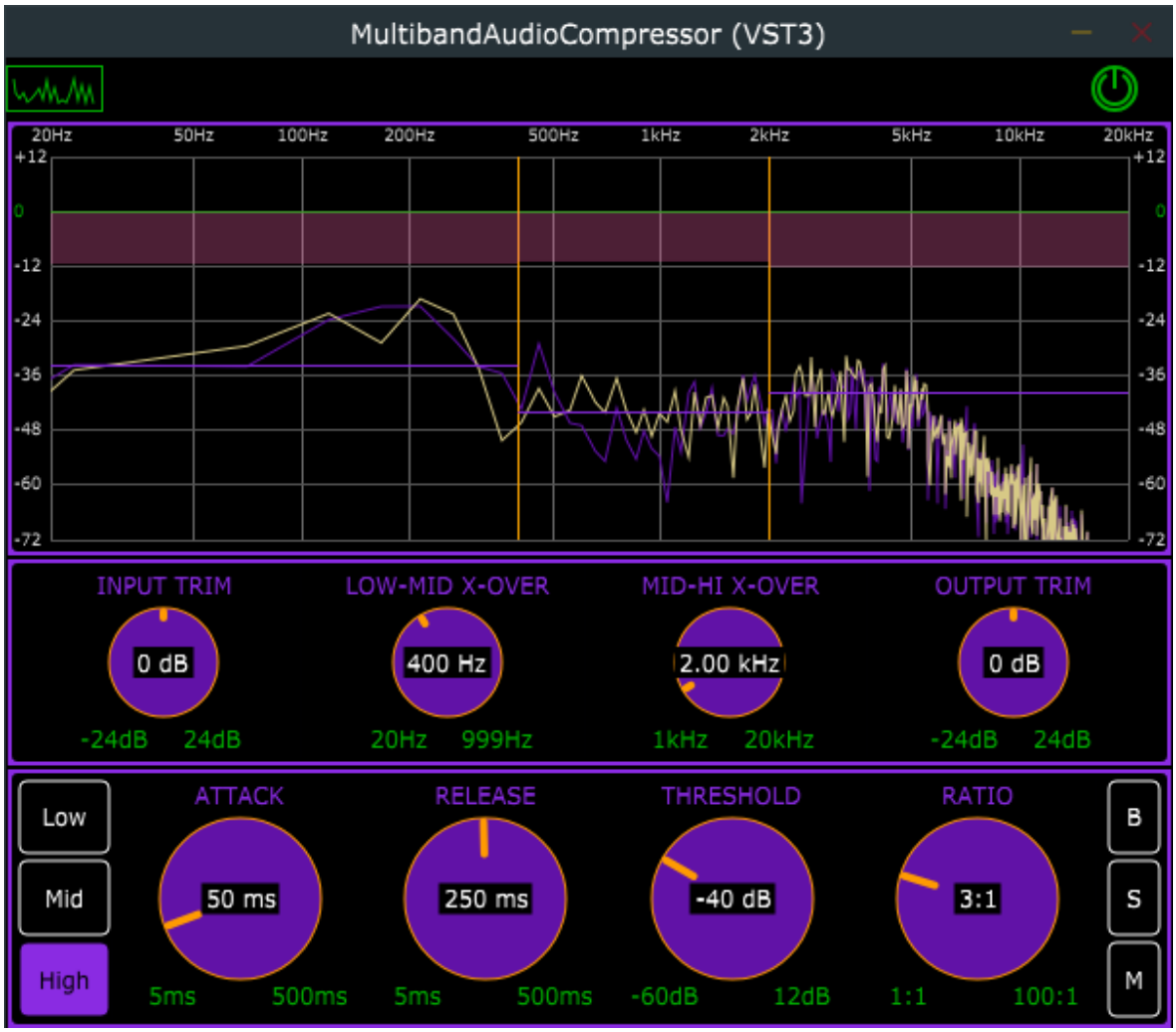
Pripremaju se u funkciji prepareToPlay, a ažuriraju se u funkciji updateState u processBlocku. Nakon pripreme, ulazni audio signal prolazi kroz ove filtre kako bi bio razdvojen na odgovarajuće pojaseve frekvencija prije kompresije.

3.3. Implementacija korisničkog sučelja

Korisničko sučelje (UI) višepojasnog audio kompresora ključni je dio softverskog rješenja, osiguravajući intuitivno i učinkovito upravljanje funkcijama kompresora. Sučelje je

dizajnirano tako da korisnicima pruža jasnu i preglednu kontrolu nad svim aspektima obrade zvuka. Podijeljeno je u četiri glavne komponente, svaka sa specifičnom funkcijom: ControlBar, SpectrumAnalyzer, GlobalControls, CompressorBandControls. Implementacija i funkcionalnost ovih komponenti bit će opisani u sljedećim potpoglavljima.

Kombinacija ovih komponenti čini korisničko sučelje moćnim alatom za preciznu i detaljnu kontrolu nad višepojasnim audio kompresorom, omogućujući korisnicima optimizaciju zvuka prema njihovim specifičnim potrebama.



Slika 12: korisničko sučelje višepojasnog audio kompresora

3.4. Primjer korištenja i funkcionalnost

3.4.1. ControlBar

Slika 13: ControlBar

ControlBar je ključni dio korisničkog sučelja višepojasnog audio kompresora, omogućujući korisnicima jednostavan pristup glavnim kontrolama i postavkama plugina. U sklopu ControlBar-a implementirana su dva glavna gumba koja omogućuju osnovne funkcionalnosti: Gumb za isključivanje plugina i gumb za gašenje prikaza spektralne analize.

3.4.1.1 Gumb za isključivanje plugina

Ovaj gumb omogućuje korisniku da potpuno isključi plugin, zaobilazeći svu obradu zvuka i propuštajući zvučni signal direktno kroz plugin bez ikakvih promjena. Ovo je korisno za brzu provjeru originalnog zvuka bez kompresije.

Slika 14: Gumb za isključivanje plugina

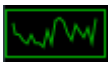
Gumb je implementiran kao toggle gumb koji mijenja svoj status između uključenog i isključenog stanja. Kada je gumb pritisnut, provjerava se status svih frekvencijskih pojaseva i svi se zaobilaze ako su uključeni.

```
1137  void MultibandAudioCompressorAudioProcessorEditor::updateGlobalBypassButton()
1138  {
1139      auto params = getBypassParams();
1140      bool allBandsAreBypassed = std::all_of(params.begin(), params.end(), [](const auto& param) {return param->get(); });
1141      controlBar.globalBypassButton.setToggleState(allBandsAreBypassed, juce::NotificationType::dontSendNotification);
1142  }
```

Slika 15: implementacija funkcionalnosti gumba za isključivanje plugina

3.4.1.2 Gumb za gašenje prikaza spektralne analize

Ovaj gumb omogućuje korisniku da uključi ili isključi prikaz spektralne analize. Ova funkcionalnost je korisna kada korisnik želi smanjiti opterećenje procesora ili kada analiza spektra nije potrebna.



Slika 16: Gumb za gašenje prikaza spektralne analize

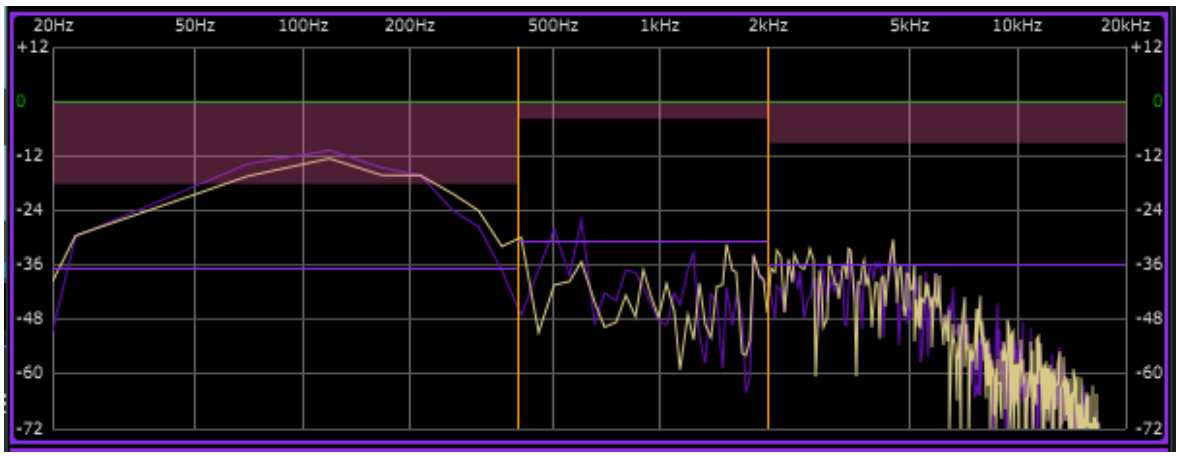
Ovaj gumb koristi toggle funkcionalnost za prebacivanje između prikaza i sakrivanja spektralne analize. Kada je gumb pritisnut, funkcija spektralne analize se zaustavlja, čime se smanjuje opterećenje procesora, a prikaz krivulje spektra se uklanja iz korisničkog sučelja.

```
1079    controlBar.analyzerButton.onClick = [this]()
1080        {
1081            auto shouldBeOn = controlBar.analyzerButton.getToggleState();
1082            analyzer.toggleAnalysisEnablement(shouldBeOn);
1083        };
```

Slika 17: Implementacija funkcionalnosti gumba za gašenje prikaza spektralne analize

3.4.2. Višefunkcijski zaslon

Višefunkcijski zaslon je ključni dio korisničkog sučelja višepojasnog audio kompresora. Omogućuje vizualni prikaz frekvencijskog sadržaja ulaznog i obrađenog audio signala, pružajući korisnicima detaljan uvid u spektralni sadržaj audio signala i način na koji obrada višepojasnim kompresorom utječe na njega.



Slika 18: SpectrumAnalyzer

Osim navedenog, na višefunkcijskom zaslonu prikazane su glavne značajke kompresije, kao i prikaz podjele cjelokupnog područja čujnih frekvencija tri frekvencijska pojasa definirana dvjema graničnim frekvencijama između njih.

3.4.2.1 Prikaz spektra ulaznog i izlaznog signala

Implementacija prikaza spektralnog analizatora uključuje:

- Priprema za prikaz: Inicijalizacija potrebnih resursa i parametara za iscrtavanje spektra.
- Prikupljanje podataka: Korištenje FIFO (First-In-First-Out) spremnika za prikupljanje audio podataka koji će se koristiti za analizu.
- Računanje spektra: Korištenje Fast Fourier Transform (FFT) algoritma za pretvorbu vremenskih podataka u frekvencijsku domenu.
- Iscrtavanje spektra: Crtanje frekvencijskog spektra koristeći grafičke funkcije JUCE framework-a

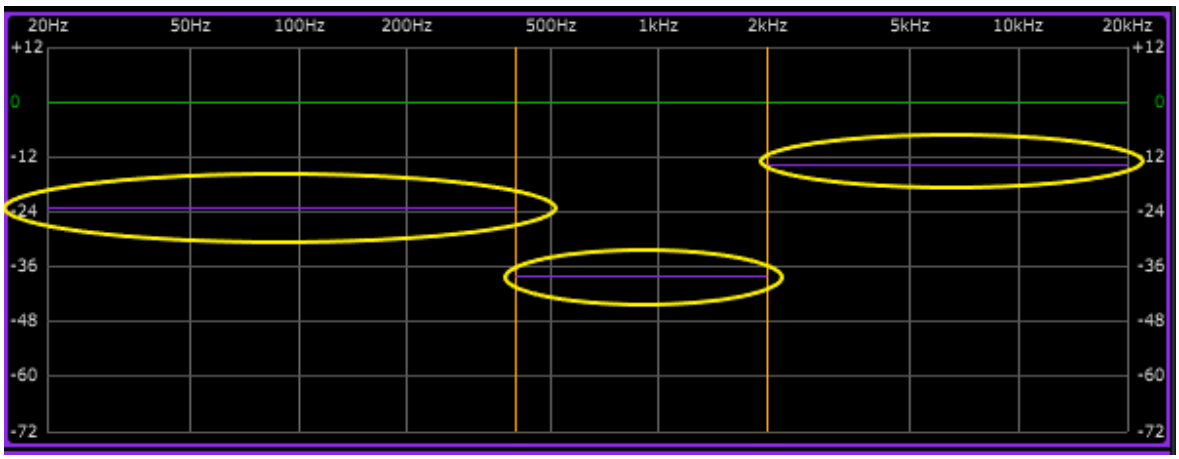
```
template<typename BlockType>
struct FFTDataGenerator
{
    void produceFFTDataForRendering(const juce::AudioBuffer<float>& audioData, const float negativeInfinity)
    {
        const auto fftSize = getFFTSize();

        fftData.assign(fftData.size(), 0);
        auto* readIndex = audioData.getReadPointer(0);
        std::copy(readIndex, readIndex + fftSize, fftData.begin());

        window->multiplyWithWindowingTable(fftData.data(), fftSize);

        forwardFFT->performFrequencyOnlyForwardTransform(fftData.data());

        int numBins = (int)fftSize / 2;

        for (int i = 0; i < numBins; ++i)
        {
            auto v = fftData[i];
            if (!std::isinf(v) && !std::isnan(v))
            {
                v /= float(numBins);
            }
            else
            {
                v = 0.f;
            }
            fftData[i] = v;
        }

        for (int i = 0; i < numBins; ++i)
        {
            auto data = juce::Decibels::gainToDecibels(fftData[i], negativeInfinity);
            fftData[i] = data;
        }

        fftDataFifo.push(fftData);
    }

    void changeOrder(FFTOrder newOrder)
    {
        order = newOrder;
        auto fftSize = getFFTSize();

        forwardFFT = std::make_unique<juce::dsp::FFT>(order);
        window = std::make_unique<juce::dsp::WindowingFunction<float>>(fftSize, juce::dsp::WindowingFunction<float>::blackmanHarris);

        fftData.clear();
        fftData.resize(fftSize * 2, 0);

        fftDataFifo.prepare(fftData.size());
    }
    //================================================================
    int getFFTSize() const { return 1 << order; }
    int getNumAvailableFFTDataBlocks() const { return fftDataFifo.getNumAvailableForReading(); }
    //================================================================
    bool getFFTData(BlockType& fftData) { return fftDataFifo.pull(fftData); }
```

Slika 19: implementacija FFT algoritma

3.4.2.2 Prikaz ključnih značajki kompresije

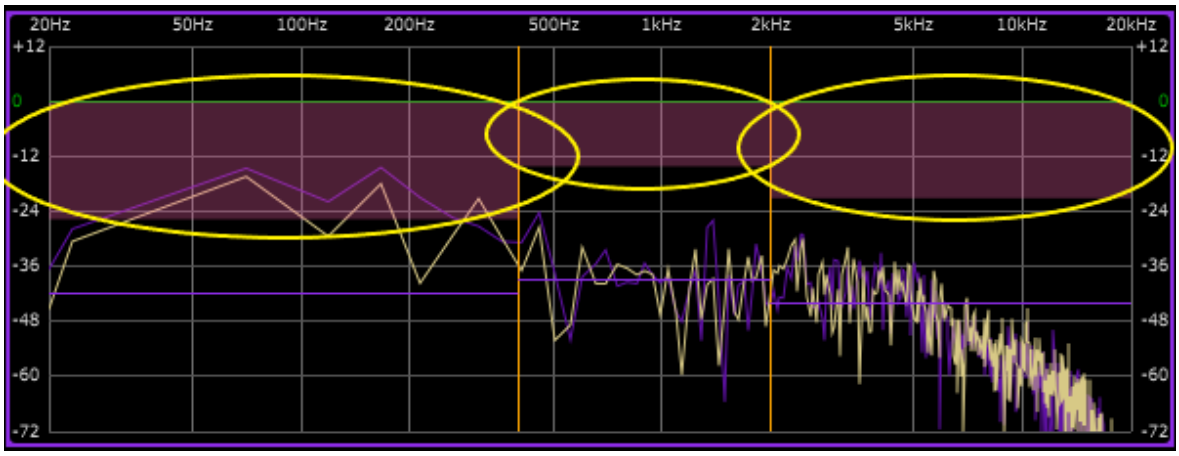
Prikaz spektralnog sadržaja audio signala ujedno omogućuje vizualizaciju ključnih parametara kompresije, što uključuje postavku praga (threshold) i smanjenje pojačanja kompresora za svaki od triju elementarnih kompresora.

Linije praga prikazuju trenutne postavke pragova triju kompresora, omogućujući korisnicima da vide pri kojoj se razini ulaznog signala aktivira određeni kompresor. Ove su linije prikazane trima horizontalnim crtama u trima frekvencijskim pojasevima koje se pomiče gore-dolje u skladu s promjenama postavki praga za svaki kompresor.



Slika 20: linije pragova kompresije za svaki pojas kompresije

Smanjenje pojačanja svakog od triju kompresora ilustrirano je odgovarajućim pravokutnikom koji mijenja veličinu/visinu ovisno o veličini smanjenja pojačanja. Ovakav vizualni prikaz pomaže korisnicima da lakše razumiju učinak kompresora na signal, tj. koliko je kompresija izražena.

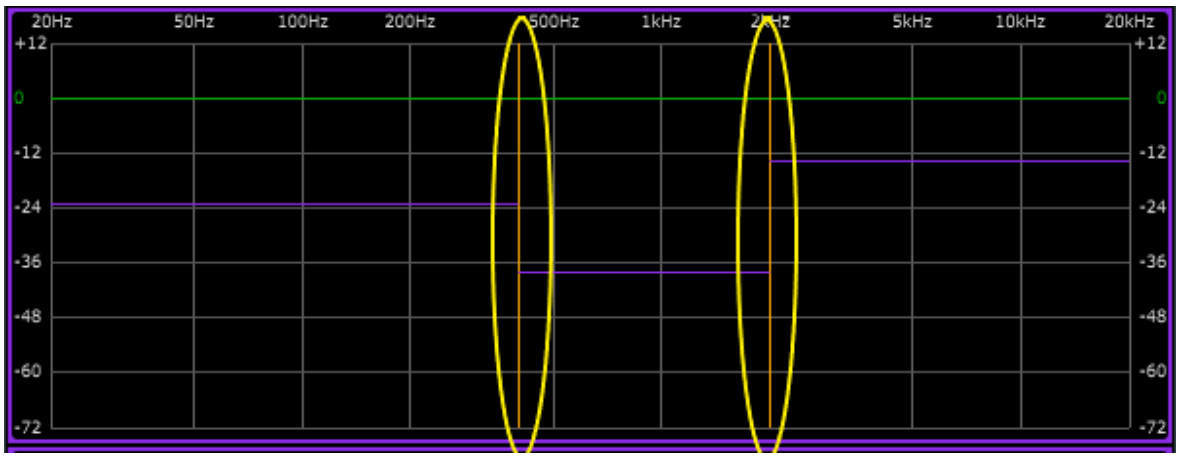


Slika 21: prikaz kompresije za svaki pojas kompresije

3.4.2.3 Prikaz podjele na pojaseve frekvencija

Prikaz podjele cjelokupnog čujnog frekvencijskog područja na pojaseve frekvencija daje korisnicima vizualnu, jasno razumljivu informaciju o tome kako je audio signal podijeljen na tri frekvencijska pojasa koji se obrađuju zasebnim kompresorima. Ova funkcionalnost je realizirana kroz vertikalne linije koje predstavljaju granične frekvencije između pojaseva, a korisnici mogu mijenjati te granične frekvencije pomoću slidera u globalnim kontrolama.

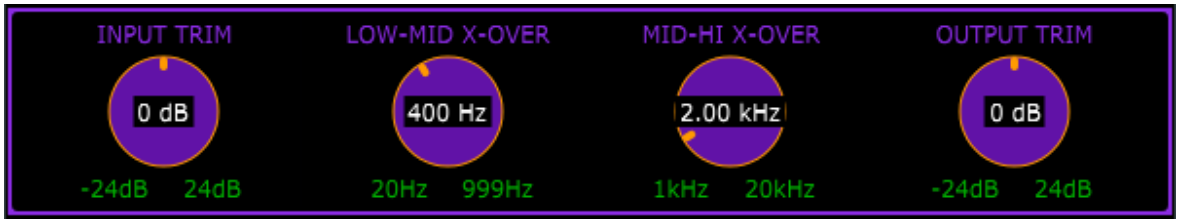
Svaka vertikalna linija označava graničnu frekvenciju između dva frekvencijska pojasa. Linije se mogu pomicati lijevo ili desno kako bi se prilagodile frekvencijske granice, što omogućuje preciznu kontrolu nad trima dijelovima audio signala koje će svaki od triju kompresora zasebno obraditi.



Slika 22: prikaz podjele kompresora na frekvencijske pojaseve

3.4.3. GlobalControls

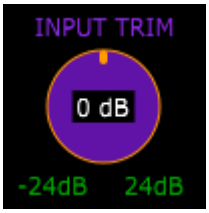
GlobalControls sučelje omogućava korisnicima prilagodbu ključnih parametara višepojasnog kompresora kako bi postigli željeni zvučni efekt. Sastoji se od četiri ključna slidera koji omogućavaju fino podešavanje ulaznog i izlaznog pojačanja te dviju graničnih frekvencija kojima su definirana tri frekvencijska pojasa.



Slika 23: GlobalControls

3.4.3.1 Ulazno pojačanje (*engl*. Input trim)

Slider za namještanje ulaznog pojačanja (Input trim) omogućava korisnicima da kontroliraju razinu signala koji ulazi u kompresor. Podešavanjem ovog parametra mogu prilagoditi jačinu ulaznog signala prije procesa kompresije. Predviđeni raspon vrijednosti ovog parametra je od -24 dB do 24 dB.



Slika 24: input trim slider

3.4.3.2 Granična frekvencija između pojasa niskih i pojasa srednjih frekvencija (*engl*. Low-Mid frequency crossover)

Slider za namještanje granične frekvencije između pojasa niskih i pojasa srednjih frekvencija omogućuje korisnicima precizno podešavanje točke prijelaza između niskog i srednjeg frekvencijskog pojasa kompresora. Predviđeni raspon ovog parametra je od 20 Hz do 999 Hz.

Slika 25: low-mid frequency crossover slider

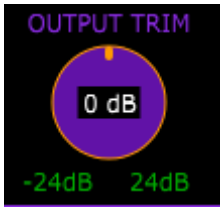
3.4.3.3 Granična frekvencija između pojasa srednjih i pojasa visokih frekvencija (*engl*. Mid-High frequency crossover)

Slider za frekvenciju prijelaza između niskih i srednjih frekvencija omogućuje korisnicima precizno podešavanje točke prijelaza između srednjeg i visokog frekvencijskog pojasa kompresora. Predviđeni raspon vrijednosti ovog parametra je od 1 kHz do 20 kHz.

Slika 26: mid-high frequency crossover slider

3.4.3.4 Izlazno pojačanje (*engl*. Output trim)

Slider za namještanje izlaznog pojačanja (Output trim) omogućava korisnicima da kontroliraju razinu signala koji izlazi iz kompresora. Podešavanjem ovog parametra mogu prilagoditi jačinu izlaznog signala koji dolazi iz procesa kompresije. Predviđeni raspon vrijednosti ovog parametra je od -24 dB do 24 dB.

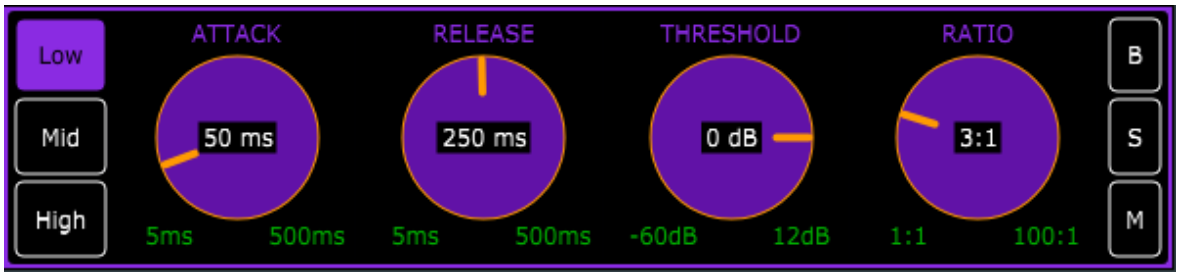


Slika 27: output trim slider

3.4.4. CompressorBandControls

CompressorBandControls sučelje omogućava korisnicima preciznu kontrolu nad parametrima svakog od triju kompresora unutar višepojasnog kompresora. Sastoji se od nekoliko ključnih elemenata koji omogućavaju podešavanje vrijednosti ključnih parametara kompresije za odabrani frekvencijski pojas, konkretno, vremena reakcije, vremena otpuštanja te praga i omjera kompresije.

Ovi elementi omogućavaju korisnicima preciznu kontrolu kompresije primijenjene u svakom od triju frekvencijskih pojaseva, omogućujući im da postignu željeni zvučni efekt u skladu sa svojim preferencijama i potrebama.



Slika 28: CompressorBandControls

3.4.4.1 Odabir frekvencijskog pojasa

Element za odabir frekvencijskog pojasa kompresora omogućava korisnicima da odaberu pojas frekvencija za koji žele namjestiti parametre kompresije pomoću ostalih opcija unutar ovog sučelja. Kada korisnik odabere određeni pojas, ostali parametri se automatski prilagođavaju za taj odabrani pojas.

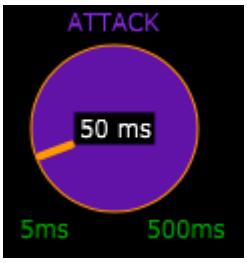
Slika 29: Odabir pojasa kompresora

3.4.4.2 Namještanje vremena reakcije (*engl*. Attack Time slider)

Attack Time slider omogućava korisnicima da prilagode vrijeme reakcije kompresora kada ulazni signal prijeđe zadani prag. Vrijeme reakcije određuje koliko brzo će kompresor početi smanjivati razinu izlaznog signala kada razina ulaznog signala prijeđe zadani prag kompresije. Niže vrijednosti vremena reakcije rezultiraju bržom reakcijom kompresora, dok više vrijednosti omogućuju sporiji, glatkiji početak kompresije.

Kada korisnik pomakne attack slider, vrijednost slidera se ažurira i ta nova vrijednost se šalje audio procesoru. Audio procesor zatim prilagođava vrijeme reakcije kompresora na temelju nove vrijednosti, omogućujući korisnicima precizno upravljanje kako brzo kompresor reagira na promjene u razini signala.

Ovaj slider je ključan za kontrolu dinamike zvuka, jer omogućava finu prilagodbu brzine reakcije kompresora, što je posebno važno za različite vrste zvuka. Na primjer, za brze perkusivne zvukove može biti poželjno imati vrlo brzo vrijeme reakcije, dok za vokale ili druge glatke zvukove može biti poželjno imati sporije vrijeme reakcije kako bi se izbjegli nagli prijelazi.



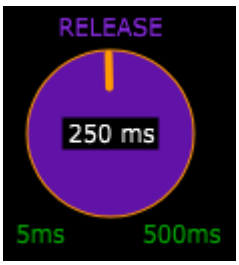
Slika 30: attack slider

3.4.4.3 Namještanje vremena otpuštanja (*engl*. Release Time slider)

Release Time slider omogućava korisnicima da prilagode vrijeme otpuštanja (*engl*. release time) kompresora, tj. vrijeme koje je kompresoru potrebno da prestane smanjivati razinu izlaznog signala nakon što razina ulaznog signala padne ispod zadanog praga. Vrijeme otpuštanja određuje koliko brzo će se kompresor vratiti u stanje bez kompresije nakon što signal padne ispod praga. Kraće vrijeme otpuštanja rezultira bržim oporavkom kompresora, dok dulje vrijeme omogućuje glatkiji prijelaz.

Kada korisnik pomakne release slider, vrijednost slidera se ažurira i ta nova vrijednost se šalje audio procesoru. Audio procesor zatim prilagođava vrijeme otpuštanja kompresora na temelju nove vrijednosti, omogućujući korisnicima precizno upravljanje koliko brzo ili sporo kompresor prestaje smanjivati glasnoću nakon što signal padne ispod praga.

Ovaj slider je ključan za kontrolu dinamike zvuka, jer omogućava finu prilagodbu brzine otpuštanja kompresora, što je posebno važno za različite vrste zvuka. Na primjer, za brze, impulzivne zvukove može biti poželjno imati kraće vrijeme otpuštanja kako bi se kompresor brzo oporavio, dok za sporije, glatke zvukove može biti poželjno imati dulje vrijeme otpuštanja kako bi se izbjegli nagli prijelazi i održao prirodniji zvuk.



Slika 31: release slider

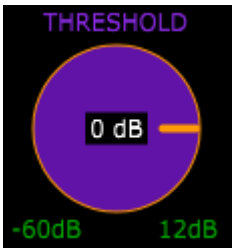
3.4.4.4 Namještanje praga kompresije (*engl*. Threshold slider)

Threshold slider omogućava korisnicima da odrede prag (*engl*. threshold) iznad kojeg će kompresor početi smanjivati razinu signala. Prag je ključni parametar u kompresoru jer

određuje na kojoj razini ulaznog signala kompresor počinje djelovati. Kada ulazni signal prijeđe zadani prag, kompresor aktivira smanjenje razine prema postavljenim parametrima omjera (*engl*. ratio), vremena reakcije (*engl*. attack), i vremena otpuštanja (*engl*. release).

Kada korisnik pomakne threshold slider, vrijednost slidera se ažurira i ta nova vrijednost se šalje audio procesoru. Audio procesor zatim prilagođava prag kompresora na temelju nove vrijednosti, omogućujući korisnicima precizno upravljanje na kojoj razini ulaznog signala kompresor počinje djelovati.

Threshold slider je ključan za upravljanje dinamikom audio signala. Pravilno postavljanje praga može značajno utjecati na karakteristike zvuka, omogućujući korisnicima da precizno kontroliraju kako kompresor reagira na različite razine signala. Kroz interaktivno podešavanje praga, korisnici mogu eksperimentirati s različitim postavkama kako bi postigli željeni efekt kompresije, bilo da žele suptilno kontrolirati dinamiku ili agresivno komprimirati signal za specifične kreativne efekte.



Slika 32: threashold slider

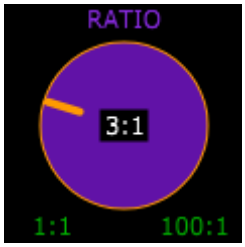
3.4.4.5 Namještanje omjera kompresije (*engl*. Ratio slider)

Ratio slider omogućava korisnicima da odrede omjer kompresije (*engl*. compression ratio), koji definira koliko će ulazni signal iznad praga (*engl*. threshold) biti smanjen. Omjer kompresije je ključni parametar koji određuje koliko će kompresor smanjiti razinu signala koji prelazi prag. Na primjer, omjer 4:1 znači da će promjena razine ulaznog signala od 4 dB signala biti smanjena na promjenu razine izlaznog signala od svega 1 dB.

Kada korisnik promijeni ratio slider, vrijednost slidera se ažurira i ta nova vrijednost se šalje audio procesoru. Audio procesor zatim prilagođava omjer kompresije na temelju nove vrijednosti, omogućujući korisnicima precizno upravljanje koliko će signal iznad praga biti smanjen

Ratio slider igra ključnu ulogu u oblikovanju ponašanja kompresora. Podešavanjem omjera kompresije, korisnici mogu precizno kontrolirati koliko će signal iznad praga biti smanjen.

Interaktivno prilagođavanje omjera omogućava eksperimentiranje s različitim postavkama, čime se postiže željeni efekt kompresije, bilo da se radi o suptilnoj kontroli dinamike ili agresivnom smanjivanju signala za specifične kreativne efekte.



Slika 33: ratio slider

3.4.4.6 Bypass, Solo, Mute kontrole

Bypass, solo, i mute kontrole omogućuju korisnicima dodatnu fleksibilnost i kontrolu nad kompresijskim pojasevima. Svaka od ovih kontrola ima specifičnu funkciju koja pomaže u preciznoj prilagodbi zvučnog signala.

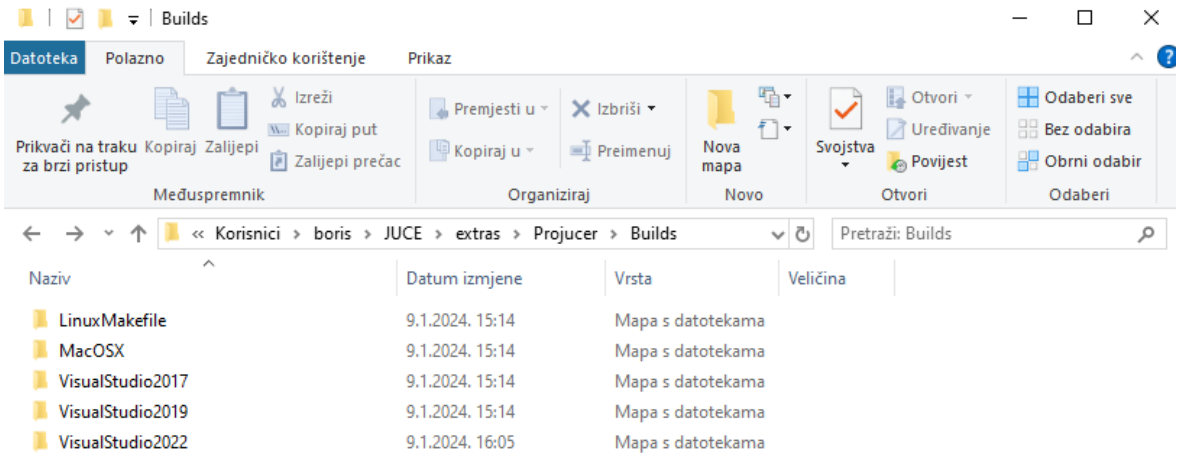
- Bypass gumb: Bypass gumb omogućava korisnicima da zaobiđu kompresor za odabrani frekvencijski pojas. Kada je bypass aktiviran, signal prolazi kroz kompresor bez ikakve obrade, što omogućuje korisnicima da čuju izvorni, neobrađeni zvuk u tom pojasu. Ova funkcija je korisna za usporedbu obrađenog i izvornog zvuka, te za odlučivanje o tome da li je kompresija za određeni pojas uopće potrebna.
- Solo gumb: Solo gumb izolira zvuk određenog frekvencijskog pojasa, omogućujući korisnicima da čuju samo taj pojas dok su ostali pojasevi utišani. Ova funkcija je izuzetno korisna pri preciznom podešavanju parametara kompresije za pojedini pojas, jer omogućuje fokusiranje na specifični dio zvučnog spektra bez utjecaja ostalih frekvencijskih pojaseva.
- Mute gumb: Mute gumb utišava zvuk za odabrani frekvencijski pojas. Kada je mute aktiviran, signal iz tog pojasa se potpuno uklanja iz izlaznog signala. Ova funkcija je korisna za identificiranje problematičnih frekvencijskih područja koja možda treba ukloniti ili dodatno obraditi, te za stvaranje specifičnih efekata gdje je uklanjanje određenih frekvencija poželjno.

Slika 34: bypass, solo i mute kontrole

3.5. Upute za pokretanje

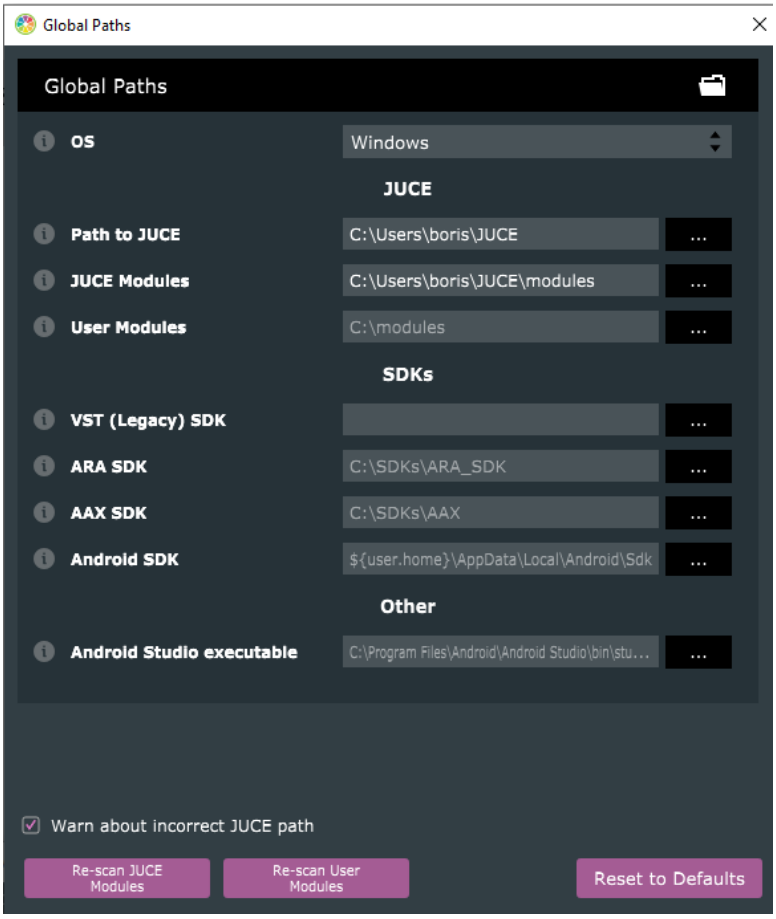
Instalacija JUCE frameworka:

Posjetite službenu web stranicu JUCE frameworka na https://juce.com i preuzmite JUCE installer za vaš operativni sustav (preporučeno je JUCE instalirati preko git naredbe *clone* jer osigurava stalno ažuriranje frrameworka i poboljšanje sustava. Nakon što ste preuzeli sve potrebne podatke za JUCE otvorite JUCE Projucer koji se nalazi unutar extras/Projucer/Builds datoteka. Ovisno o vašem operativnom sustavu i verziji IDE kojeg imate otvorite Projucer pomoću tog IDE-a.



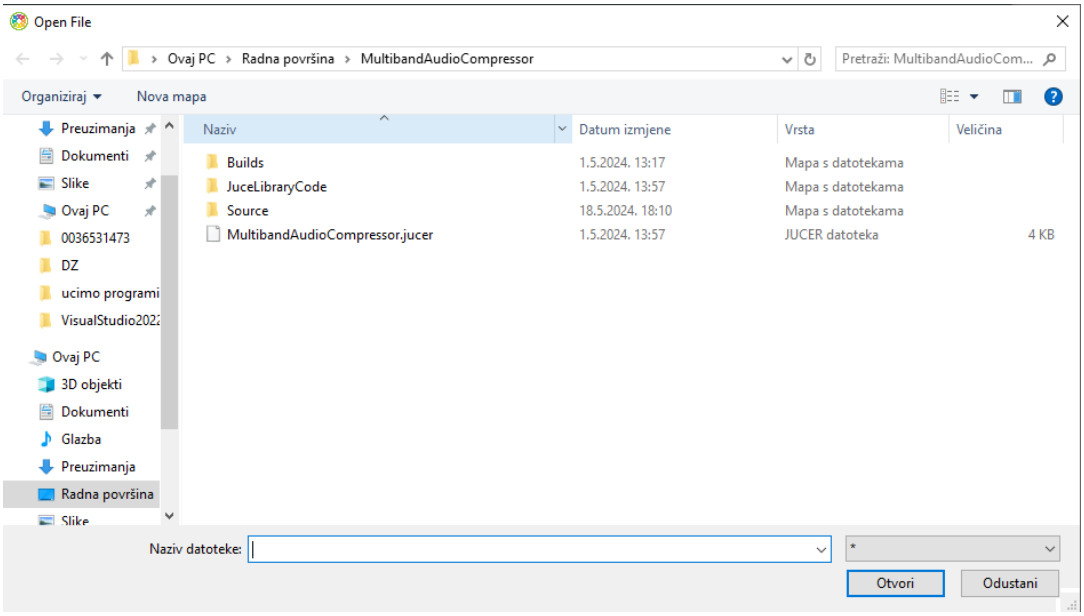
Slika 35: projucer path

Nakon što otvorite Projucer trebate provjeriti Global path kako biste namjestili prave module za uključivanje u projekt pomoću padajućeg izbornika file > Global Paths… .



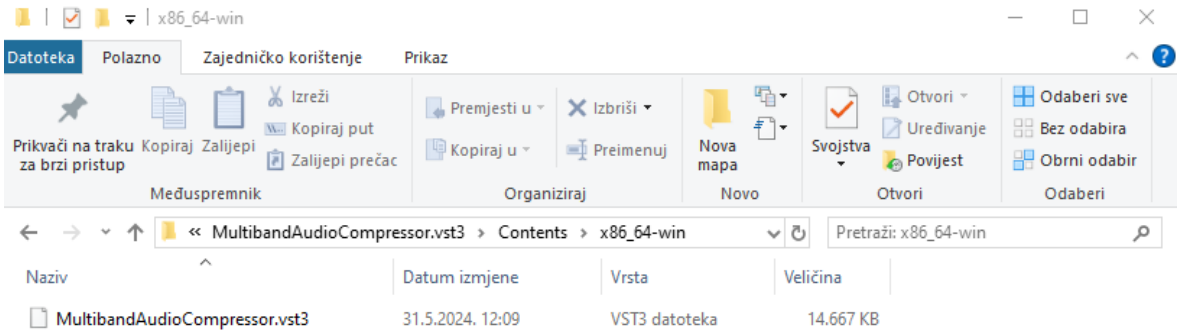
Slika 36: Global paths

Kada su moduli namješteni možemo otvoriti projekt. To možemo napraviti tako da pomoću padajućeg izbornika File > Open… odaberemo MultibandAudioCompressor.jucer datoteku, te u projuceru pritisnemo gumb za otvaranje IDE-a.



Slika 37: MultibandAudioCompressor.jucer

Nakon što se otvori IDE možemo buildati plugin klikom na Build > Build Solution ili Build > Rebuild solution. Naš završeni build će se nalaziti u folderu: …\MultibandAudioCompressor\Builds\VisualStudio2022\x64\Debug\VST3\MultibandAudioCompressor.vst3\Contents\x86_64-win



Slika 38: build

4. Zaključak

Višepojasna kompresija predstavlja moćan alat za dinamičku obradu audiosignala, omogućujući precizniju i fleksibilniju kontrolu nad zvukom. Implementacija VST dodatka s višepojasnim kompresorom pružit će korisnicima alat za unapređenje kvalitete njihovih audio produkcija, omogućujući im prilagodbu kompresije različitim dijelovima frekvencijskog spektra. Ovaj rad teorijski i praktično obrađuje sve aspekte višepojasne kompresije, pružajući temelj za daljnji razvoj i optimizaciju audio alata.

5. Literatura

Zölzer, U. (Ed.). (2011). DAFX: Digital Audio Effects. John Wiley & Sons.

Izhaki, R. (2017). Mixing Audio: Concepts, Practices and Tools. Focal Press.

Pohlmann, K. C. (2011). Principles of Digital Audio. McGraw-Hill.

Van de Par, D. J., & Abel, J. S. (2002). Multiband Compression and Expansion. AES Convention Papers

De-Figueiredo, N., & Brandão, M. H. (2016). Audio Signal Compression Using Multiband Dynamic Range Compression. Proceedings of the 139th AES Convention.

Linkwitz, S. (1976). Active Crossover Networks for Noncoincident Drivers. Journal of the Audio Engineering Society.

Riley, R. (1976). Design of Active Crossovers for loudspeakers. Journal of the Audio Engineering Society.

6. Sažetak

Ovaj rad bavi se razvojem višepojasnog kompresora kao dodatka za digitalne audio radne stanice (DAW) korištenjem JUCE frameworka. Višepojasni kompresor omogućuje neovisnu obradu različitih frekvencijskih pojaseva u zvuku, pružajući korisnicima preciznu kontrolu nad dinamikom audio signala. Opisane su teorijske osnove audio kompresije, algoritmi i tehnike korištene u implementaciji, te funkcionalnosti razvijenog dodatka. Poseban naglasak stavljen je na korisničko sučelje koje uključuje kontrolne elemente za podešavanje parametara kompresije i vizualizaciju frekvencijskog spektra. Rad također razmatra uporabu Linkwitz-Riley filtera za razdvajanje frekvencijskih pojaseva, čime se osigurava kvaliteta zvuka bez faznih pomaka i drugih izobličenja.

7. Summary

This thesis focuses on the development of a multiband compressor as a plugin for digital audio workstations (DAW) using the JUCE framework. The multiband compressor allows independent processing of different frequency bands in the audio signal, giving users precise control over the signal dynamics. Theoretical foundations of audio compression, algorithms, and techniques used in the implementation are described, along with the functionality of the developed plugin. Special emphasis is placed on the user interface, which includes control elements for adjusting compression parameters and visualizing the frequency spectrum. The work also discusses the use of Linkwitz-Riley filters for frequency band separation, ensuring sound quality without phase shifts and distortion.

8. Privitak

Github: https://github.com/borisboronjek/Zavrsni-rad