

Elektrotehnički fakultet

Univerzitet u Banjoj Luci

**IZVJEŠTAJ PROJEKTNOG ZADATKA**

iz predmeta

**SISTEMI ZA DIGITALNU OBRADU SIGNALA**

Student: Mentori:

*Emanuela Buganik 1209/18* prof. dr Mladen Knežić

prof. dr Mitar Simić

ma Vedran Jovanović

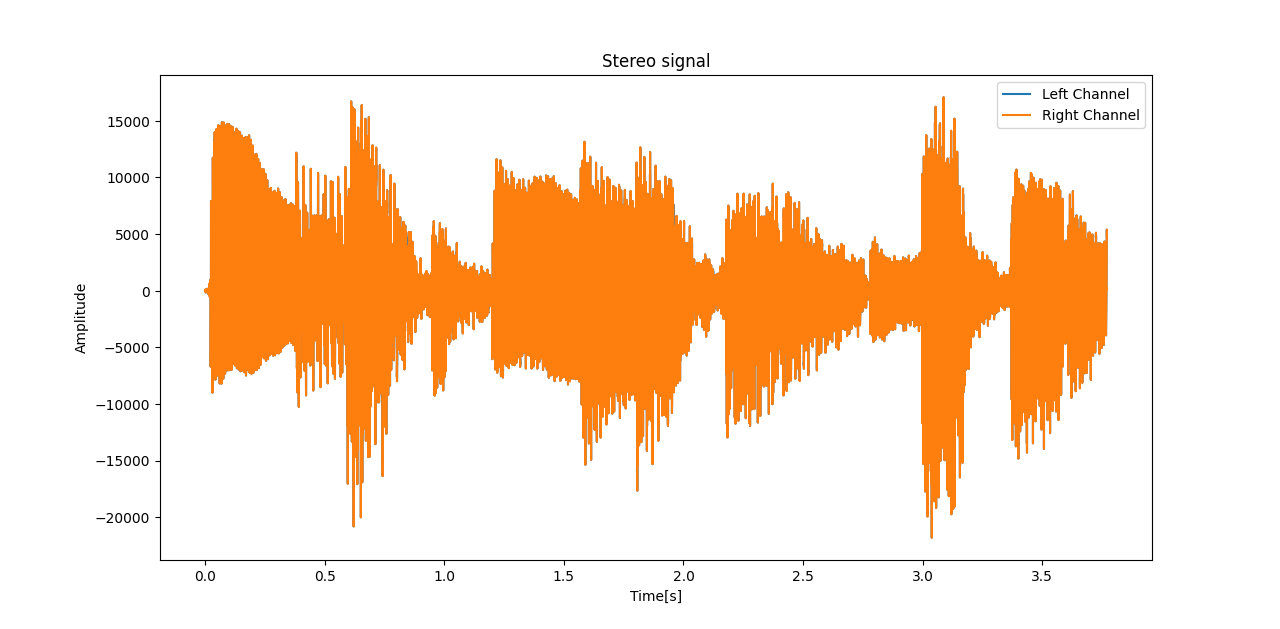
dipl. inž. Damjan Prerad

Februar 2024. godine

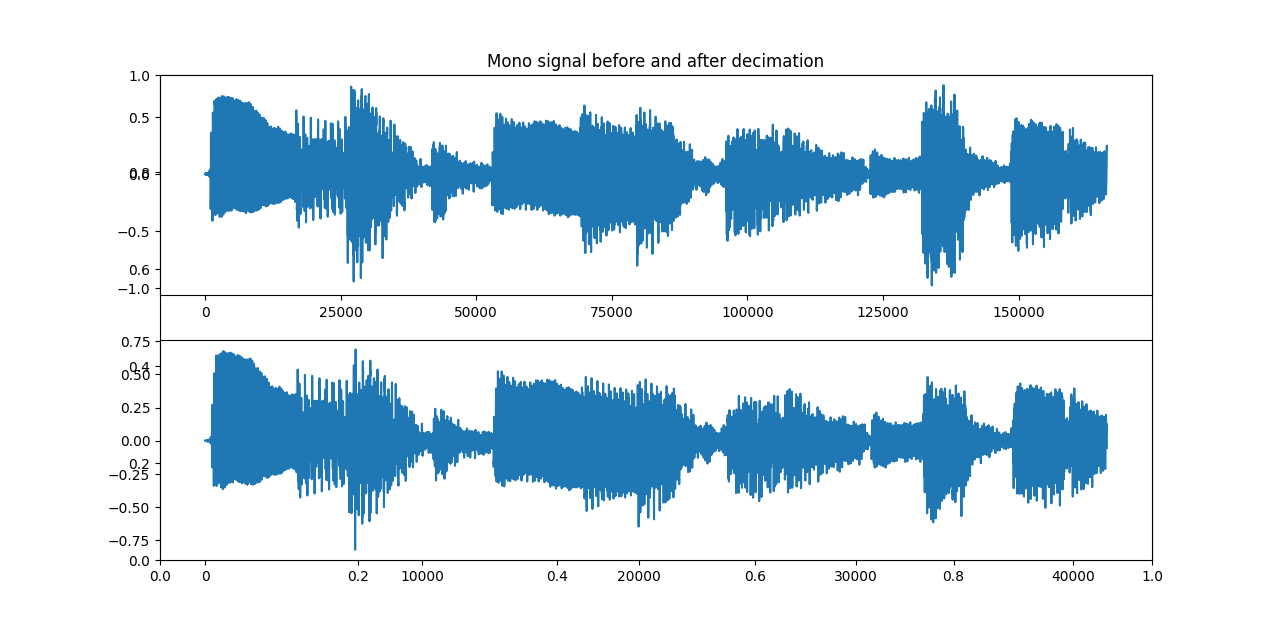
# Opis projektnog zadatka

Cilj izrade projektnog zadatka je implementacija muzičkih efekata koji se mogu primijeniti na audio signal, korištenjem razvojnog okruženja ADSP-21489. Prvi dio izrade, ujedno i referenca za dalji rad odnosi se na generisanje referentnog signala i implementaciju gitarskih efekata (iz grupa definisanih projektnim zadatkom ili proizvoljno), a u Python programskom jeziku. Drugi dio izrade odnosi se na implementaciju istih efekata, ali u C programskom jeziku i unutar *CrossCore Embedded Studio* programskog paketa za ciljanu platformu. Odmjerci signala koji je korišten za analizu u prvom dijelu izrade upisuju se u file koji se zatim učitava u CCES kao *header* file i omogućava obradu istih. Moguće je reprodukovati rezultate obrade signala i u jednom i u drugom slučaju, eksportovati kao *.wav* file, a vršeno je i profilisanje koda s ciljem da se uoče potencijalni *bottleneck*-ovi i ukoliko je moguće, izvrši optimizacija koda na DSP-u. Efekti o kojima će biti riječ u narednom poglavlju su efekti iz prve i druge grupe - manje zahtijevni filtri *Delay* i *Echo* te umjereno zahtijevni *Tremolo*, *Flanger* i *Bit Crusher*.

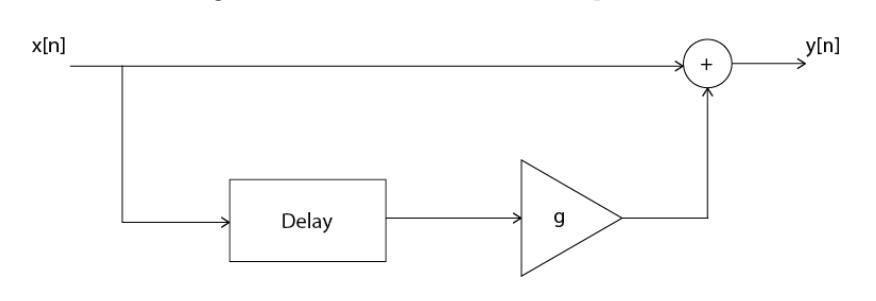
# Izrada projektnog zadatka

S obzirom na primjenu efekata koje je potrebno implementirati, kao audio signal koji će se koristiti za obradu u daljem radu koristi se stereo zvuk gitare u trajanju od 3,77 s, a čija je frekvencija odmjeravanja prvobitno 44 100 Hz. Jedna od opcija da se analizira stereo signal jeste izdvajanje kanala, primjena efekata, a zatim ponovno objedinjavanje u 2D niz. Međutim, kako je prikazano i na narednoj *Slici 1.1*

*Slika* *2.1* Stereo signal

da se zapaziti kako nema razlike između lijevog i desnog kanala, te se postupak daljeg rada pojednostavljuje i tako radi sa mono signalom koji je nastao usrednjavanjem kanala stereo signala. Pored ovog razmatranja, u dalji rad se polazi idući koji korak unaprijed, pa se razmatra i kako uvesti odmjerke ovog sada mono signala u CCES, imajući na umu da se radi o signalu sa pomenutom frekvencijom odmjeravanja, gdje je broj odmjeraka 166 267. Postavlja se pitanje ima li smisla raditi sa velikim brojem odmjeraka kada radimo sa ograničenim signalom, uzimajući u obzir i da je potrebno omogućiti i učitavanje odmjeraka (pitanje skalabilnosti unutar *header*-a?), smještanje u memoriju, ispis obrađenih odmjeraka u *file* i slično. Sljedeći metod koji se može iskoristiti da pojednostavimo rad jeste *decimacija* (eng. downsampling), kojom za neki cjelobrojni faktor D možemo da smanjimo kako frekvenciju odmjeravanja nekog već odmjerenog signala, tako i njegov broj odmjeraka. U ovom slučaju izvršena je decimacija sa faktorom četiri, čime je nova frekvencija odmjeravanja 11 025 Hz, a broj odmjeraka 41 567. Korištena je python funkcija decimate, a prethodno je spesifikovan i niskopropusnik u vidu *Hann*-ovog prozora. Naredna slika prikazuje promjenu, tačnije mono signal prije i nakon decimacije.

*Slika 2.2* Mono signal po usrednjavanju i nakon decimacije

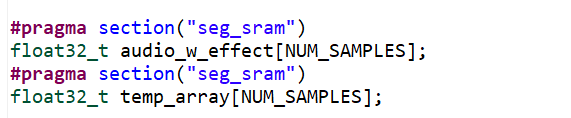
***Delay* efekat**, kao prvi efekat koji je obrađen realizuje se korištenjem FIR filtra, kako se efekat zakašnjelog signala implementira sabiranjem originalnog signala sa svojom pomjerenom, zakašnjelom verzijom, koja je pomnožena faktorom g, kako je prikazano i na blok dijagramu na *Slici 2.3*. Naivna implementacija u Pythonu prikazuje da se pomjerena verzija signala može ostvariti pomjeranjem originalnog signala za broj odmjeraka koji određuju kašnjenje.

*Slika 2.3* Blok dijagram *Delay* efekta

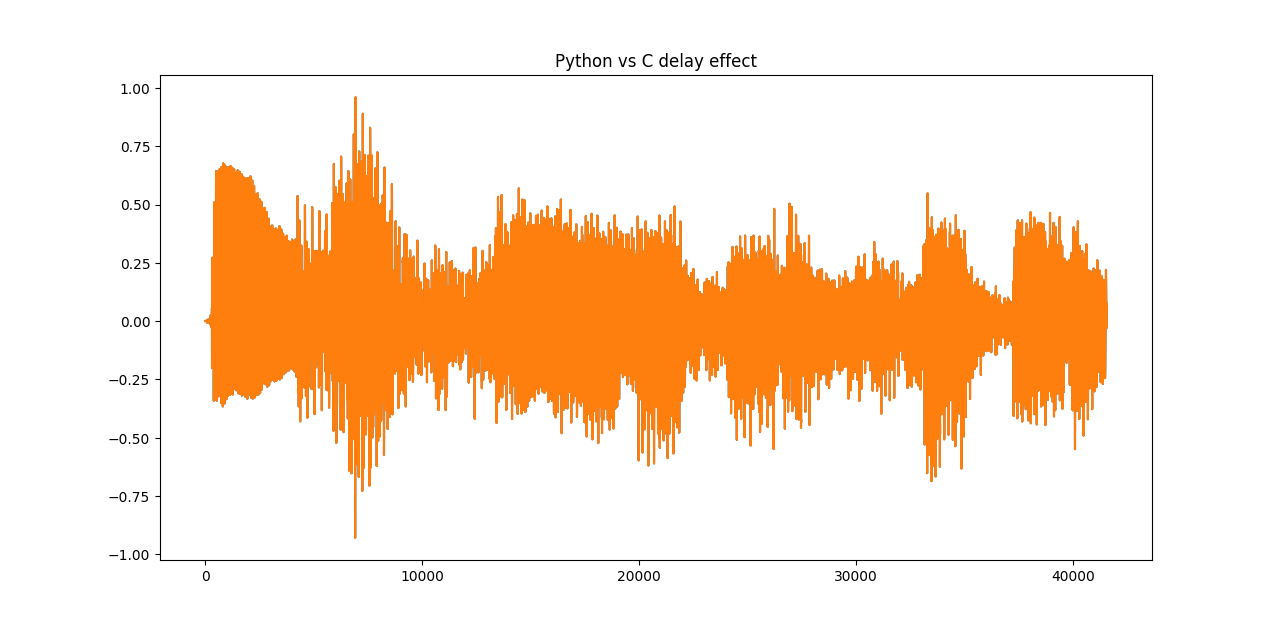
U ovom pa i narednim primjerima realizacije efekata, vrijednost faktora g je između -1 i 1, broj odmjeraka za pomjeraj (određuje se kao umnožak frekvencije odmjeravanja i vremena kašnjenja u sekundama.

(2.1)

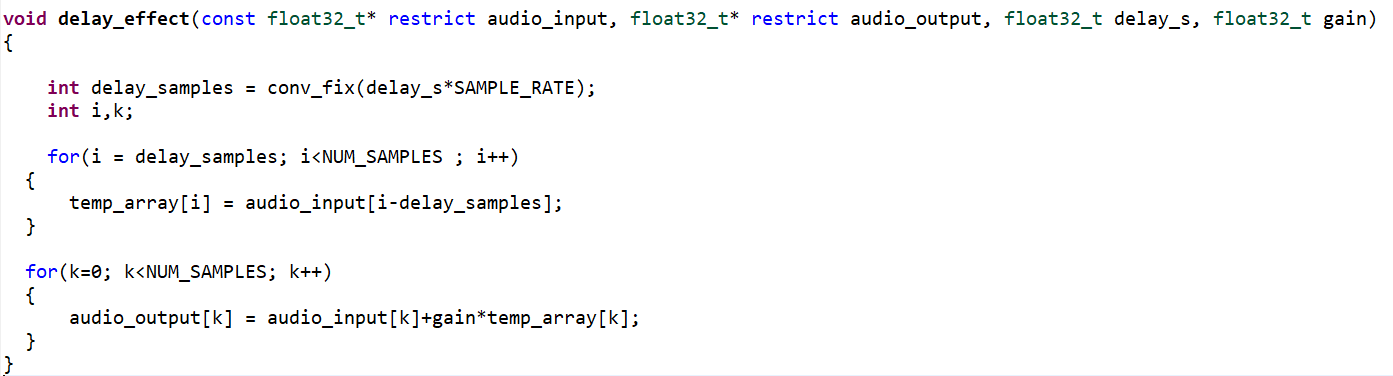
*Slika 2.4* Rezultat delay efekta nad originalnim audio signalom

S obzirom na to da će nam Python poslužiti kao referenca, za implementaciju *Delay* efekta u C-u neophodna su nam dva niza – pomoćni koji predstavlja zakašnjenu verziju originalnog signala te niz koji predstavlja rezultat obrade. Kako je realizacija ovog projektnog zadatka svedena na odmjerke konačnog broja, pritom i redukovane prethodno pomenutim postupkom decimacije, nije bilo potrebe za korištenjem dinamičke alokacije memorije, ali je nizove koje ćemo koristiti svakako potrebno „pripisati“ nekoj memoriji. Za početak analize, pomenuti nizovi su definisani unutar korisnički kreirane sekcije, koja pripada SRAM (a koja maksimalno može da pohrani 512 KB).

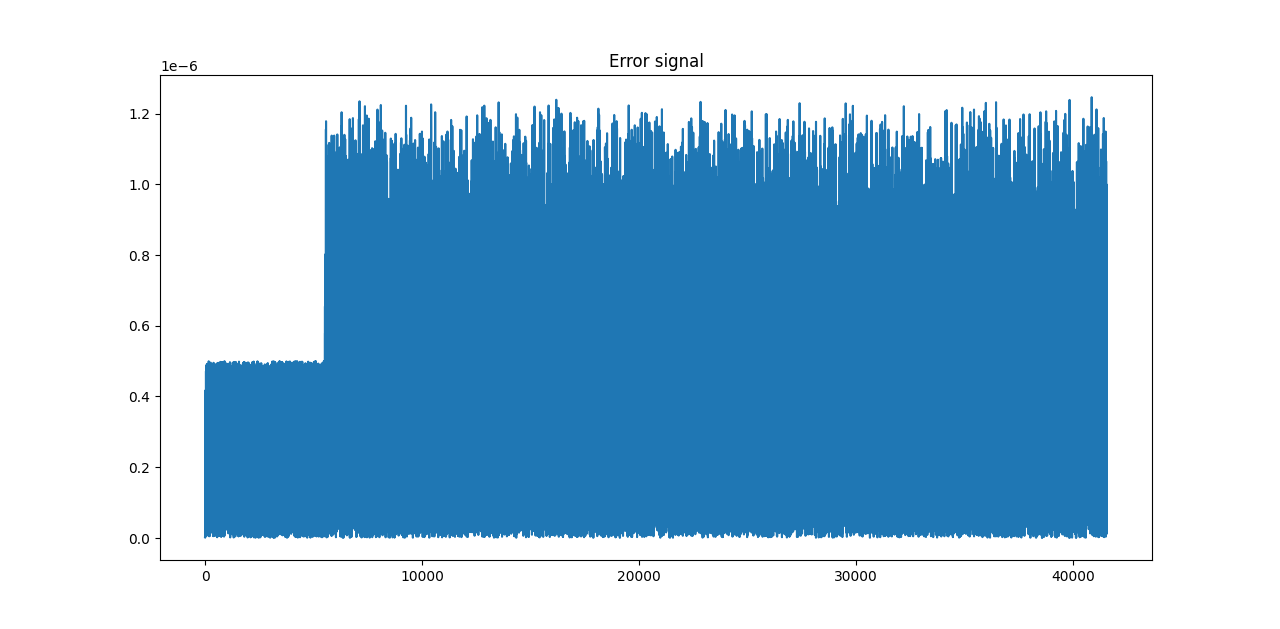
*Slika 2.5* Prikaz smještanja nizova u korisnički definisanu sekciju

**NUM\_SAMPLES je makro unutar *audio.h* file-a i definiše broj odmjeraka zvuka gitare, a nad kojim će se vršiti testiranje algoritama. Tip podataka sa kojim očekujemo raditi je float, kako bismo prilikom

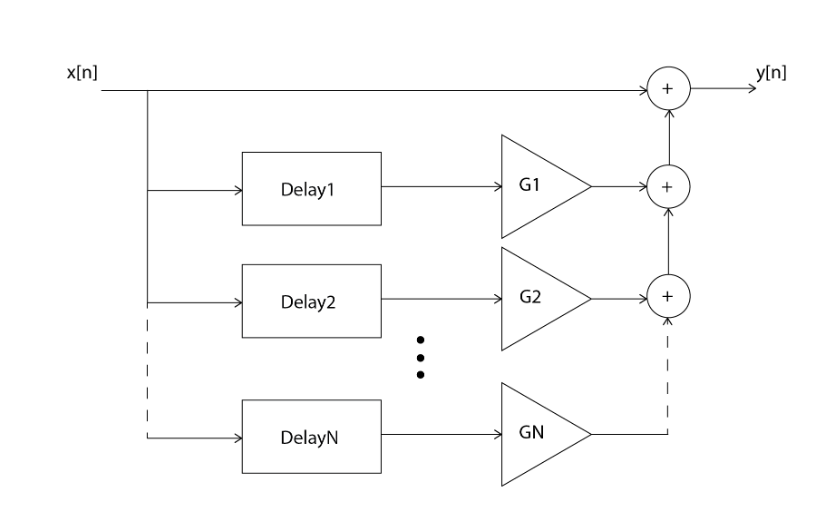
*Slika 2.5* Rezultat izvršavanja *Delay* efekta u Pythonu i na ploči (poklapanje)

**operacija nad odmjercima ostvarili što veću preciznost i rezultate koji će biti približni onom dobijenom u *Pythonu.* Konkatenaciju koja je ondje korištena u C kodu zamijenjena je upisivanjem odmjeraka u pomoćni niz od pozicije *delay\_samples* do NUM\_SAMPLES, kako bismo osigurali da se nad nizovima u nastavku može odvijati operacija sabiranja i skaliranja te kako bi konačan rezultat bio iste dužine kao i ulazni niz.

*Slika 2.6* Implementacija *Delay* efekta u C kodu

Kao što je učinjeno i u Pythonu, svi argumenti funkcije, osim frekvencije odmjeravanja (koja je navedena kao makro) prosljeđuju se kao float vrijednosti. U C-u stavljamo naglasak na to da je u pitanju doista 32-*bitna vrijednost. Na slici 2.6 da se primijetiti upotreba intrinsic funkcije ili ugrađene funkcije, koja* omogućava efikasnije iskorištavanje hardverskih resursa, u ovom slučaju za dobijanje broja odmjeraka za koje se vrši pomjeranje. Kvalifikator *restrict* ukazuje na \*\*\*. Kroz dvije jednostavne for petlje vrši se prvo dobijanje zakašnjelih odmjeraka, a zatim skaliranje i dodavanje polaznom signalu. Ovakva, naivna realizacija funkcije iziskuje čak 4 351 940 ciklusa izvršavanja, a kada bi se se petlje sažimale u jednu, s obzirom na to da je krajnja vrijednost NUM\_SAMPLES, svega 4 156 747. O potencijalnim poboljšanima i primijenjenim tehnikama za smanjenje broja ciklusa biće više pomenuto na kraju ovog poglavlja. Na

*Slika 2.7* Signal greške kod Delay efekta

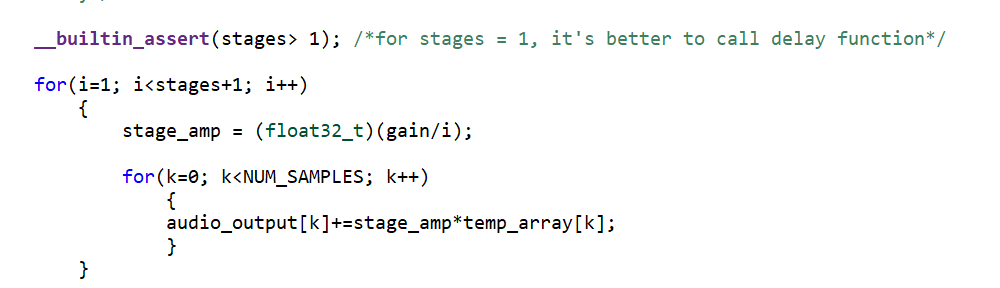
slici 2.7 možemo vidjeti razliku u obradi i Pythonu i C-u, a za koju se dalo naslutiti na osnovu slike 2.6 kako će doista da bude u vidu male greške signala. Sljedeći efekat koji možemo nadovezati na Delay jeste ***Echo efekat***, višestruki efekat kašnjenja koji se može realizovati kroz kašnjenje na više linija ili *stage*s, kako je prikazano na narednoj slici:

*Slika 2.4* Blok dijagram *Echo* efekta

Kako se da i primijetiti, *Echo* efekat se sastoji od dodavanja originalnog audio signala N broju zakašnjenih verzija istog, gdje faktor skaliranja (, gdje je i = 1, 2, 3, …) opada sa svakom linijom kašnjenja. Očito je da je za slučaj N=1, u pitanju gorepomenuti slučaj *Delay* efekta.

(2.2)

Ovo znači da za npr. N=3, vrijednost izlaza zavisi od vrijednosti kašnjenja na izlazima na linijama kašnjenja, tj. prethodnim vrijednostima izlaza koji se superponiraju.

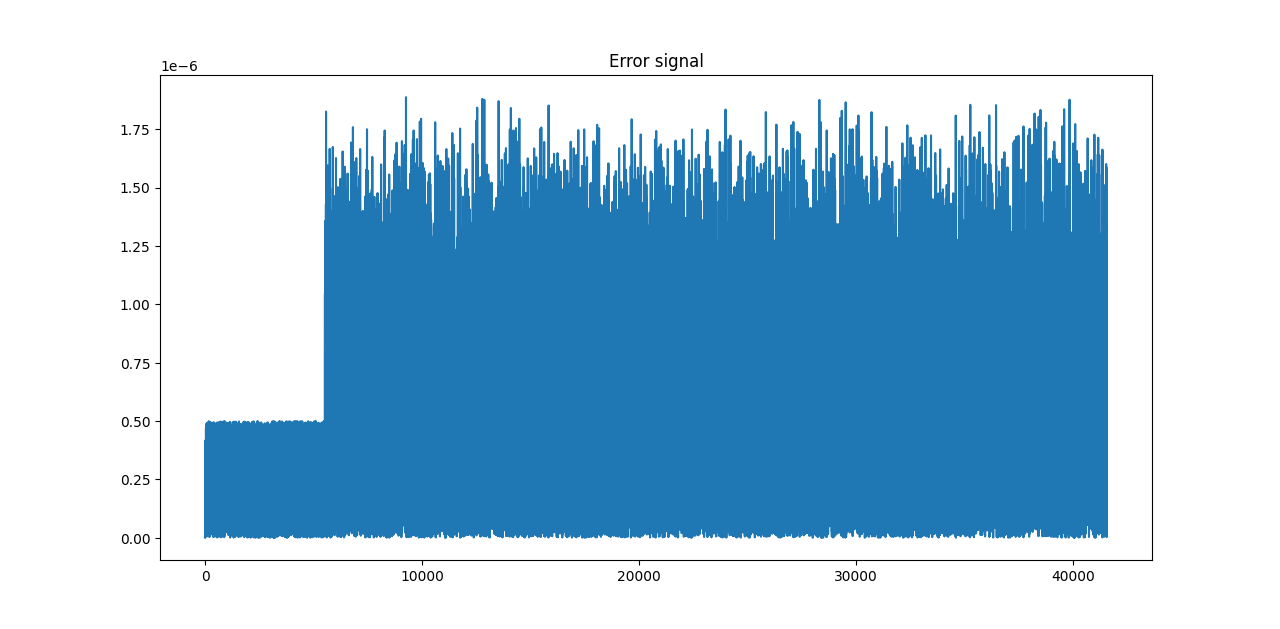


*Slika 2.5* Efekat kašnjenja na linijama kašnjenja u C-u

Dio koda na slici 2.5 je ujedno i dio funkcije Echo koji iziskuje najviše izvršavanja, što je na neki način i očekivano i sa povećanjem broja linija kašnjenja i broj ciklusa potencijalno raste, a neophodno je proći kroz sve odmjerke signala i sabrati ih sa zakašnjenom verzijom. Iako se pokušaj optimizovanja ovog dijela koda pokazao kao zahtjevniji i kao što će rezultati kasnije da prikažu jedino kompajlerska optimizacija ima mogućnost da dobrano smanji broj ciklusa izvršavanja, ovaj kod prikazuje dobru praksu koju navodi i Manual za Sharc kompajler, a to je da u slučaju ugniježđenih petlji unutrašnja uvijek ima više ciklusa izvršavanja u odnosu na spoljašnju, jer je očekivano da će ondje da provede najviše vremena. Varijanta u prethodnim verzijama projekta da se obrne redoslijed dovela je do velikih povećanja u broju ciklusa.

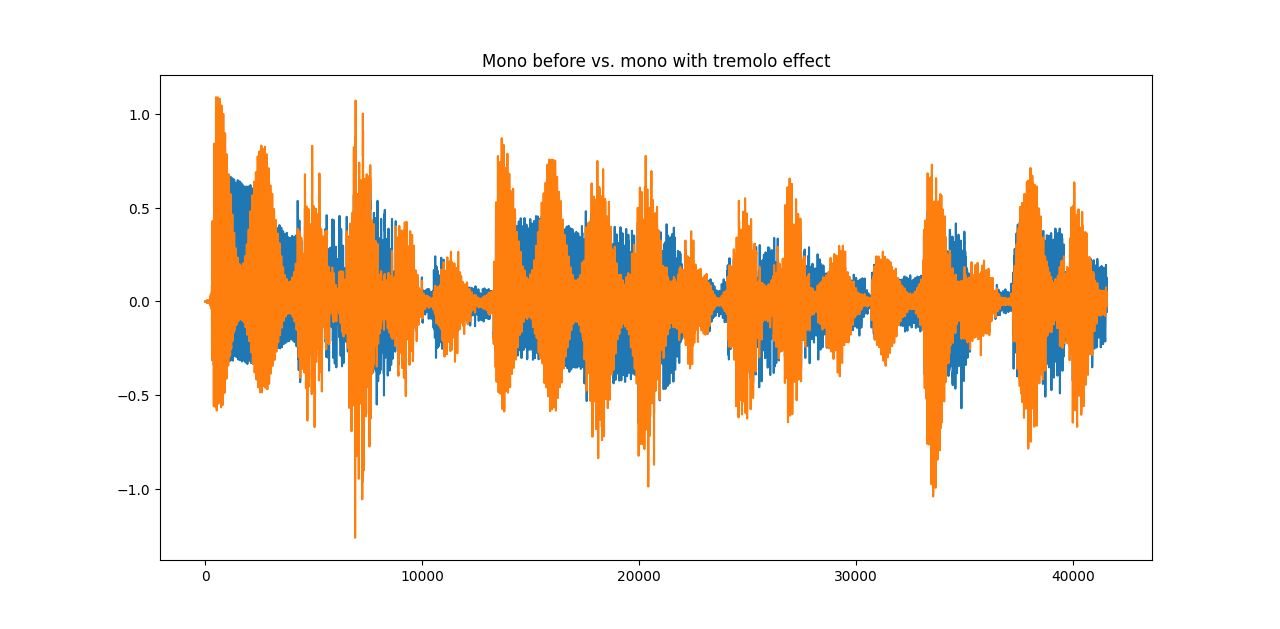
*\_\_builtin\_assert* je funkcija koja može znatno pomoći kompajleru u radu, te mu tako pruža informaciju o vrijednosti parametara koji se prosljeđuju u funkciju što na primjer on sam ne može da zaključi. Tako je i u ovom slučaju očekivano da vrijednost parametra *stages* bude veća od jedan, jer bi u suprotnom bio slučaj Delay efekta i nema smisla pozivati ovu funkciju za izvršenje.

I u slučaju greške kod Echo efekta, u pitanju su jako male vrijednosti razlike, kao što se može prikazati i na Slici 2.6.

*Slika 2.6* Signal greške kod Echo efekta

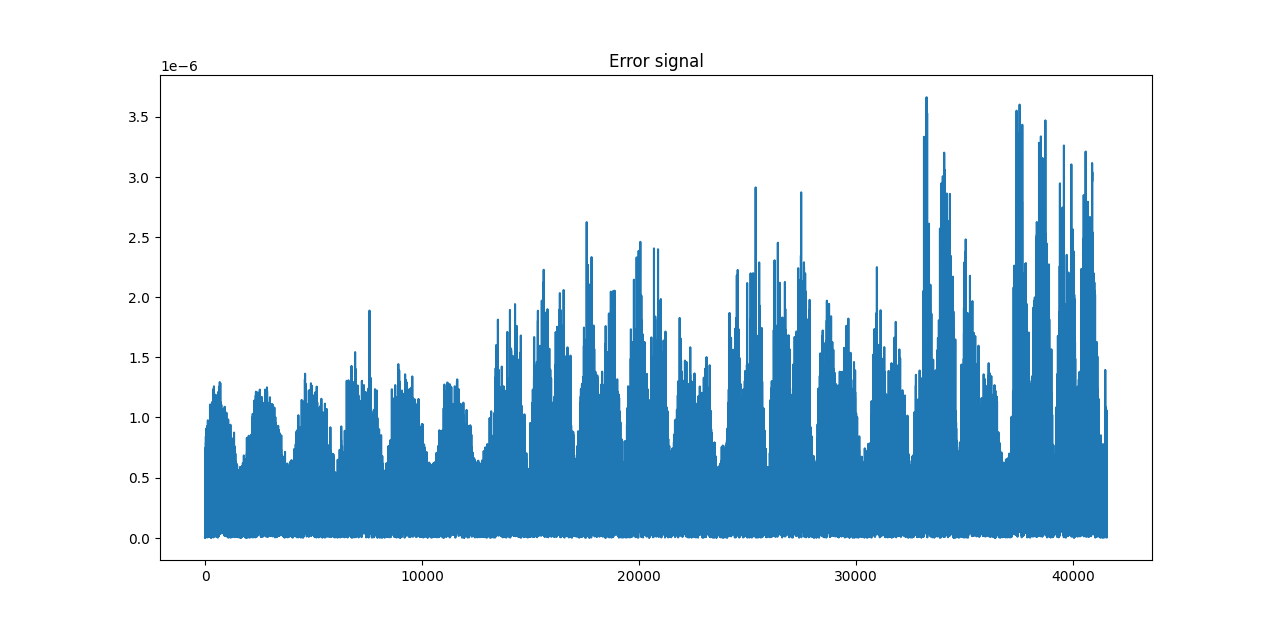
***Tremolo efekat*** iz druge grupe efekata realizuje se kao vid amplitudske modulacije, što znači da se amplituda ulaznog signala (eng*. carrier*) mijenja sa promjenama LFO (oscilatora na niskim frekvencijama). Narednom jednačinom to se može prikazati na sljedeći način:

(2.3)

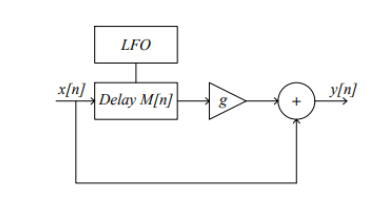
Gdje je n)

*Slika 2.7* Signal prije i nakon primjene *Tremolo* efekta u Pythonu

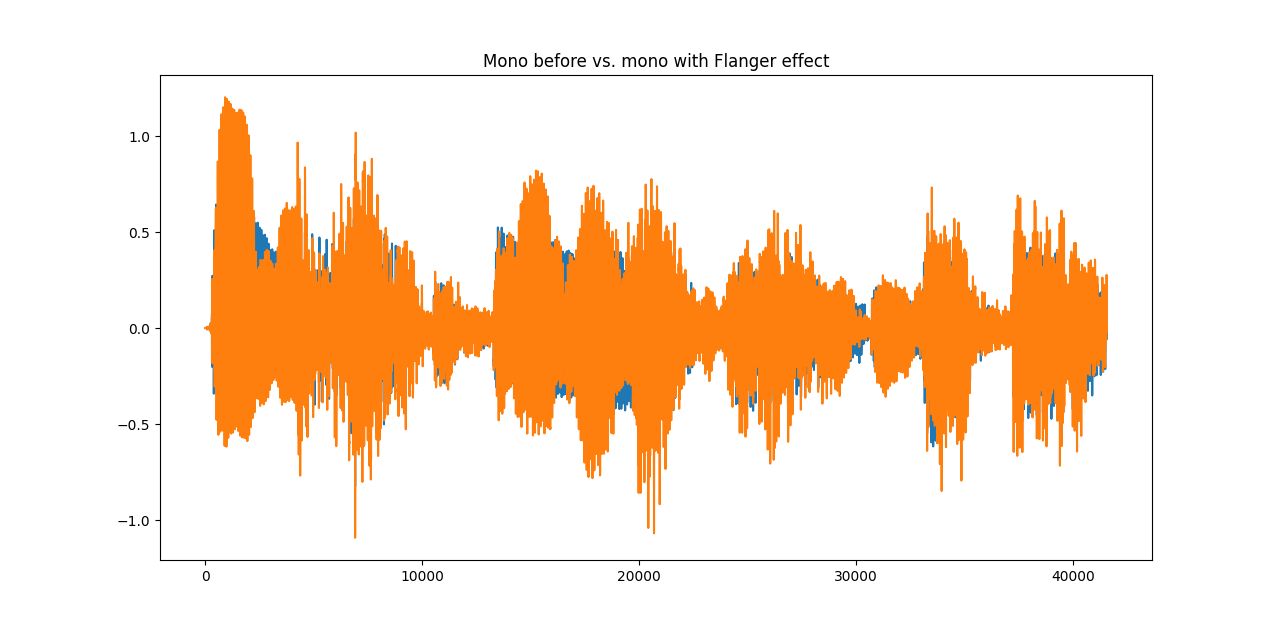
*Depth* određuje amplitudu signala modulatora, uopšte – dubinu amplitudske modulacije, a frekvenciju modulatora (obično u opsegu od 1 Hz do 25 Hz).

Ovdje može dio koda da se ubaci …

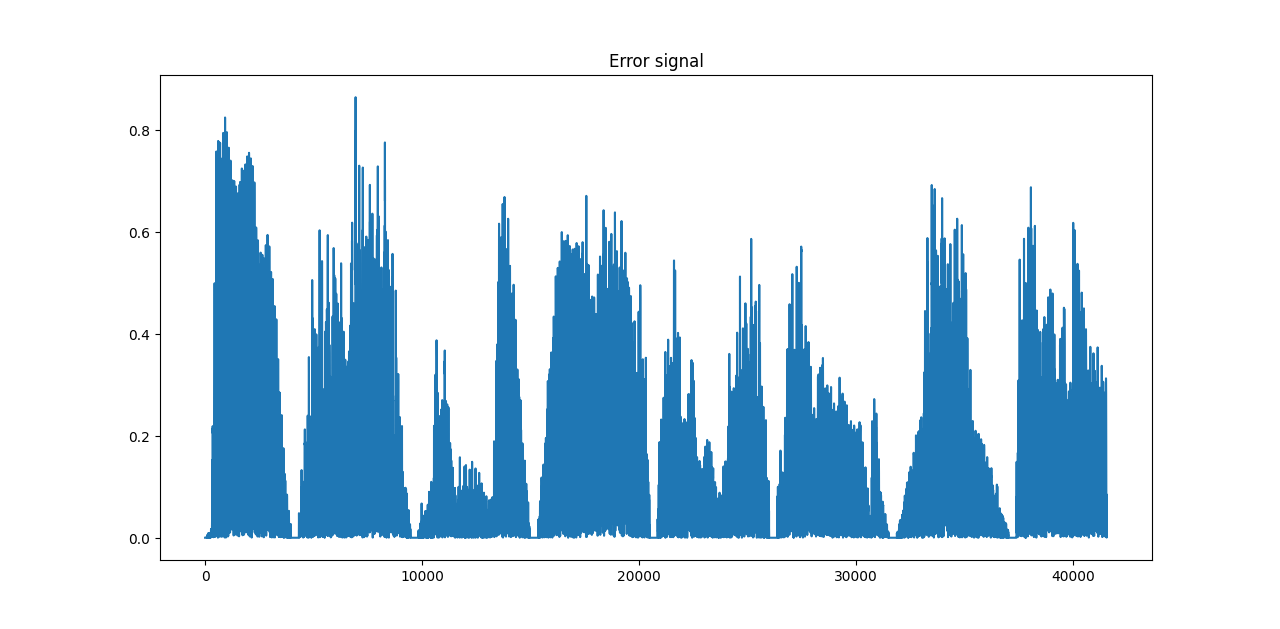
*Slika 2.8* Signal greške *Tremolo* efekta

Osnovna ideja iza *Flanger* efekta je stvaranje konstruktivne i destruktivne interferencije, dodavanjem originalnog zvuka zakašnjeloj verziji zvuka, čije je kašnjenje određeno funkcijom vremena, a što prikazuje i sljedeći blok dijagram:

*Slika 2.9* Blok dijagram *Flanger* efekta

Pomenuta funkcija vremena je u ovom slučaju LFO, kao u primjeru realizacije efekta *Tremolo*, a čija frekvencija modulatora iznosi između 0.1 Hz do 10 Hz.

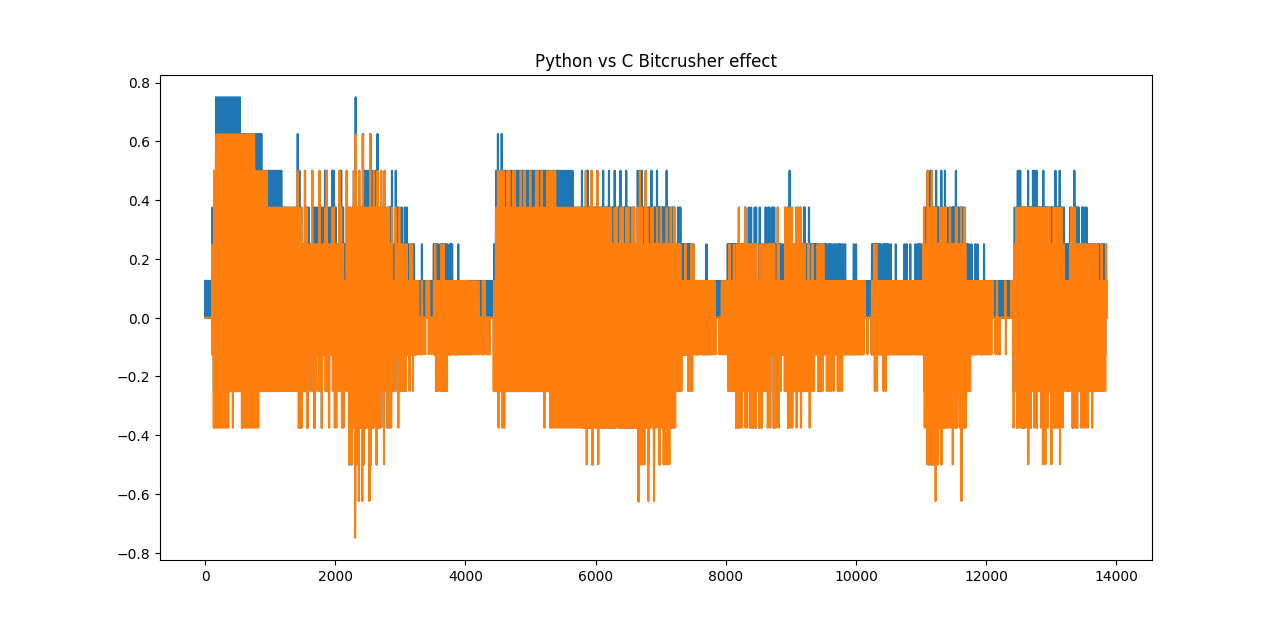
*Slika 2.10* Signal prije i nakon primjene *Flanger* efekta u Pythonu



*Slika 2.11* Signal greške kod *Flanger* efekta

Posljednji implementiran efekat je ***Bit Crusher***, poznatiji i kao *lo-fi* (eng. low-fidelity) efekat. Sa opcijama kao što su smanjenje frekvencije odmjeravanja (poznato i kao eng. „downsampling“ i „rate crush“) te smanjenje rezolucije (poznato i kao eng. „bit depth“ i „bit crush“), ovim efektom se postiže ciljana distorzija audio zapisa. Oba režima rada su implementirana u python i C kodu.

Iako je u funkciji ponovo korištena decimacija, verzija na kojoj smo se zadržali prilikom testiranja algoritma je ona bez niskopropusnog filtra. Signal koji mu je propušten već je filtriran u Pythonu, kao što je prethodno i decimovan, a ponovnim filtiranjem audio na izlazu izgubi svojstvenost efekta. Ipak, kao mogućnost u nekim drugim realizacijama i za potrebe drugih audio zapisa ovog projektnog zadatka unutar funkcije u Pythonu ostavlja se mogućnost filtiranja filtrom konačnog impulsnog odziva, a onda i generisanje koeficijenata filtra koji se mogu eksportovati u header file i zatim učitati u C-u. U C-u je

preporučeno koristiti funkciju iz *filter.h* biblioteke koja može da bude efektivnija kada je u pitanju određivanje izlaza i filtra, nego u slučaju funkcije *convolve*.

*Slika 2.12* Rezultat izvršavanja *Bit crusher* efekta u Pythonu i na ploči

# Zaključak

Za kraj izrade ovog projektnog zadatka napravićemo kratak osvrt na rad i dobijena rješenja. Istraživanje nekih najosnovnijih algoritama za implementaciju danas najpoznatijih efekata dalo je značajan uvid u to šta zapravo znači podešavanje parametara jedne gitarske pedale. S druge strane, potreba za implementacijom na DSP-u navodi nas da u cilju što boljih performansi radimo na optimizaciji pomenutih efekata i njihovom pojednostavljenju. Poznavanje samog rada DSP sistema, korištenog kompajlera, kao i preporučenih principa optimizacije vidimo da isti mogu značajno dovesti do poboljšanja kada je u pitanju izvršavanje na DSP-u. Svakako se nameće kao potreba i korištenje optimizacija koje nudi i kompajler, ali u svakom slučaju dobro je poraditi i na nekim drugim načinima implementacije algoritama, kao i razmatranju koja je memorija pogodnija (SRAM ili SDRAM), jer kao što se da primijetiti, imala je značajan uticaj u konačnom brojem ciklusa. Upoznali smo se djelimično i sa zahtijevima MISRA:C-2004 standarda koji nalaže da se ne koristi dinamička alokacija memorije. Tu je i način zapisa podataka, u ovom slučaju float u Pythonu koji se tretira kao 64-bitni, a u C-u 32-bitni, kao i konverzije koje su bile potrebne za rad u drugim slučajevima. Prisjetimo se pristupa i sa početka izrade, a to je korištenje decimacije u signalu i prethodno potreba za filtriranjem, odnosno potreba da smanjimo broj odmjeraka koje ćemo analizirati. Recimo da smo na taj način izgubili i neke detalje signala i tako utičemo na rezultate obrade (barem ono što čujemo), iako omogućavamo manje memorijsko zauzeće i omogućavamo bolje performanse. U pitanju su donekle oprečni zahtjevi, ali nije nemoguće raditi na poboljšanju, kako bi rezultati i izvršavanja na jednom DSP sistemu, ali i rezultat obrade bili uspješni.

# Literatura

[1] Materijali sa predavanja i laboratorijskih vježbi iz predmeta Sistemi za digitalnu obradu signala, Elektrotehnički fakultet, Univerzitet u Banjaluci

[2] CrossCore Embedded Studio, *CCES 2.9.0 C/C++ Complier Manual for SHARC Processors*, Revision 2.2, Analog Devices, May 2019.

[3] CrossCore Embedded Studio, *CCES 2.9.0 C/C++ Library Manual for SHARC Processors*, Revision 2.2, Analog Devices, May 2019.

1. Dodaci

Pored toga što je moguće pozivati funkciju za funkcijom i upisivati obrađene odmjerke u odgovarajuće *.txt* file-ove, paljenjem dioda se može ustanoviti u kojem dijelu obrade se trenutno nalazimo – podešavanje parametara, ulazak u funkciju, izlazak iz funkcije, početak upisa u file, te provjera ispisa trećine i polovine odmjeraka, a onda i završetak obrade. S obzirom na formirane nizove unutar korisničkih sekcija, isti se mogu koristiti za kaskadno uvezivanje efekata (odnosno, da izlaz jednog bude ulaz u drugi, pri čemu se korištenje efekta BitCrusher preporučuje kao posljednja stavka u nizu korištenih efekata). Zakomentarisan primjer nalazi se u C kodu.