WOJSKOWA AKADEMIA TECHNICZNA
im. Jarosława Dąbrowskiego
Wydział ELEKTRONIKI

PRACA DYPLOMOWA

Realizacja efektów akustycznych w procesorze sygnałowym TMS320C6713

(temat pracy dyplomowej)

Łukasz KOPCZYŃSKI, s. Michała

(stopień wojskowy, tytuł zawodowy, imiona i nazwisko, imię ojca dyplomanta)

ELEKTRONIKA I TELEKOMUNIKACJA
(kierunek studiów)

Systemy Cyfrowe
(specjalność)

STACJONARNE, STUDIA PIERWSZEGO STOPNIA - INŻYNIERSKIE
(forma i rodzaj studiów))*

dr inż. Paweł Dąbal
(stopień wojskowy, tytuł i stopień naukowy, imię i nazwisko promotora pracy dyplomowej)

WARSZAWA 2022

Spis treści

Wykaz rysunków

Wykaz akronimów i symboli

CCS – (*ang. Code Composer Studio*) – środowisko programistyczne do implementacji efektów akustycznych dla procesora sygnałowego TMS320C6713

d – opóźnienie sygnału

D – opóźnienie sygnału dla zależności modulujących sygnał

$d(n)$ – zmienna przedstawiająca zależność modulującą sygnał

f_{Cycle} – częstotliwość sygnału kosinusoidalnego dla którego realizowany jest efekt flanger

f_{Delay} – częstotliwość sygnału LFO dla którego realizowany jest efekt chorus

f_{mod} – wartość częstotliwości sygnału modulującego dla efektu tremolo

f_{shift} – częstotliwość sygnału piłokształtnego dla którego realizowany jest efekt pitch shiftera

$f_{shift_{up}}$ – częstotliwość narastania sygnału piłokształtnego dla których realizowany jest efekt zmiany tonacji w układzie detune

$f_{shift_{down}}$ – częstotliwość opadania sygnału piłokształtnego dla których realizowany jest efekt zmiany tonacji w układzie detune

LFO – (*ang. Low Frequency Oscillator*) – oscylator niskich częstotliwości

G_n – wzmocnienie sygnału

Mono – *(*ang. Monophonic*) – sygnał jednokanałowy*

n – numer kolejnej próbki sygnału

Stereo – (*ang. Stereophonic*) – sygnał wielokanałowy

$x(n)$ – wartość sygnału wejściowego dla n-tego elementu próbki

$y(n)$ – wartość sygnału wyjściowego dla n-tego elementu próbki

1. Wprowadzenie

Słuch jest jednym z najważniejszych zmysłów człowieka i to co człowiek słyszy może wpływać na jego emocje. Muzyka, która jest silnie kulturotwórczą dziedziną życia człowieka ewoluowała wraz z nim. Nowe możliwości technologiczne, w tym cyfrowe przetwarzanie sygnałów, pozwoliły powstanie nowych instrumentów muzycznych oraz zmianę brzmienia instrumentów znanych. Jedną z dziedzin nauki zajmujących się powstawaniem i rozchodzeniem się dźwięku jest akustyka, a jednym z jej zagadnień jest realizacja efektów akustycznych zmieniających dźwięk. Przykładowo współczesne rozwiązania umożliwiają zmianę nagrania zrealizowanego w studio nagraniowym tak, że będzie brzmieć jak nagrane w sali filharmonii.

Wspomniane cyfrowe przetwarzanie sygnałów realizowane było pierwotnie w sposób programowy, jednakże problemy z rosnącymi wymaganiami i złożonością algorytmów spowodowały powstanie dedykowanych układów cyfrowych zwanych procesorami sygnałowymi. Samo projektowanie, testowanie i implementacja algorytmów przyczyniła się do tego, że powstały narzędzia wspomagające inżynierów na tym etapie prac jak np. środowisko MatLAB.

Praca składa się z sześciu rozdziałów. W rozdziale drugim omówione zostały powszechnie używane efekty akustyczne wraz z podaniem przykładów istniejących układów sprzętowych realizujących dany efekt. Ponadto kluczowym dla dalszych prac było przytoczenie stosownych zależności matematycznych uzupełnionych o stosowną interpretację graficzną. Rozdział trzeci zawiera implementacje efektów w postaci skryptów dla środowiska MatLAB. Przedstawione zostały symulacja efektów zarówno w postaci stosownych wykresów jak i również możliwości porównania brzmienia przez odsłuch fragmentu utworu. Czwarty rozdział zawiera implementację efektów akustycznych w procesorze sygnałowym TMS320C6713 przygotowaną w języku C. Nałożenie efektów odbywa się w czasie rzeczywistym poprzez wybranie odpowiedniej kombinacji przełączników dostępnych na płycie uruchomieniowej. Piąty rozdział zawiera weryfikację eksperymentalną zaimplementowanych efektów dla procesora TMS320C6713. W rozdziale szóstym przedstawiono zwarte podsumowanie zrealizowanej pracy.

2. Przegląd efektów akustycznych

Fala akustyczna stanowi przemieszczające się w przestrzeni periodyczne zaburzenie ciśnienia ośrodka, w którym się rozchodzi. Falę taką można scharakteryzować następującymi parametrami: λ - długością, v - prędkością rozchodzenia się i f – częstotliwością, które to związane są zależnością:

$$\lambda = \frac{v}{f}. \tag{2.1}$$

Sygnał akustyczny może być przetworzony na sygnał elektryczny przy użyciu odpowiednich przetworników, np. mikrofonów. Reprezentacja częstotliwościowa sygnału akustycznego dla ludzkiego ucha może zawierać się w przedziale od 20 [Hz] do 20000 [Hz], natomiast próg bólu dla ucha ludzkiego określany jest na poziomie 130 [dB]. Efekty akustyczne możemy podzielić na efekty wpływające na widmo sygnału, wpływające na jego zależność czasową oraz modulujące sygnał.

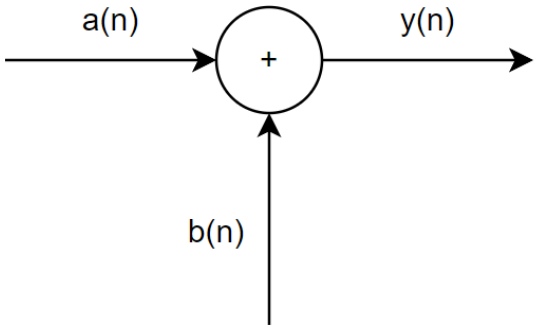
2.1. Operacje bazowe używane w efektach akustycznych

Efekty akustyczne są wynikiem wykonywania operacji na sygnale wejściowym: sumowania, mnożenia, opóźniania i ograniczania amplitudy bądź kształtowania charakterystyki widmowej. Operacja sumowania polega na dodaniu do siebie dwóch sygnałów. Realizacja efektu sumowania odbywa się np. w układzie sumatora, który powoduje dodanie do siebie sygnału wielokanałowego (ang. *stereophonic*) w celu uzyskania sygnału jednokanałowego (ang. *monophonic*). Efekt mnożenia polega na mnożeniu próbek w dziedzinie czasu. Zabieg ten wykorzystywany jest przykładowo w modulacji amplitudowej. Opóźnienie sygnału wymaga użycia linii opóźniającej składającej się z elementów opóźniających. Operacja ta realizowana jest poprzez bloki opóźniające. Ograniczenie amplitudy polega na określeniu wartości dla której amplituda sygnału nie będzie większa od ograniczenia. Ograniczenie amplitudy może być zrealizowane przy pomocy kompresorów oraz limiterów. Kształtowanie charakterystyki przez filtrację jest to zabieg polegający na mnożeniu danych zakresów częstotliwościowych w celu ich wzmocnienia bądź stłumienia, co realizują np. korektory graficzne.

Sumowanie sygnałów ma na celu takie ich dodanie aby sygnał wejściowy nie przekroczył zakresu napięciowego w jakim sygnał może się znajdować, ponieważ przekroczenie dopuszczalnych progów napięciowych może powodować pojawienie się

silnych zniekształceń nieliniowych sygnału. Operacja ta jest używana między innymi do przekształcenia sygnału stereofonicznego w monofoniczny. Na rys. 2.1 przedstawiono symbol blokowy sumatora z opisem wyprowadzeń. Składa się on z dwóch wejść, które są podłączone do bloku sumującego. Blok sumujący dodaje sygnał $a(n)$ (lewy kanał) oraz sygnał $b(n)$ (prawy kanał) i na jego wyjściu otrzymujemy sumę sygnałów podzieloną przez 2. Układ ten można przedstawić w postaci zależności matematycznej:

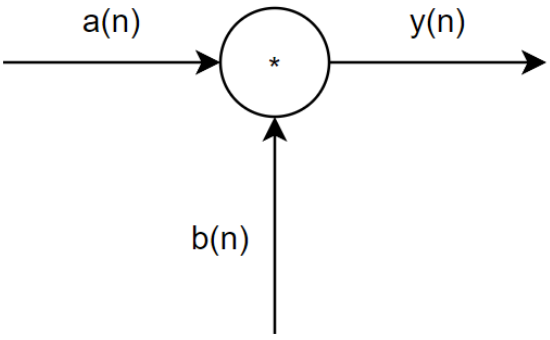
$$y(n) = \frac{a(n) + b(n)}{2} \ . \tag{2.2}$$



Rys. 2.1 Schemat blokowy sumatora

Mnożenie dwóch sygnałów polega na utworzeniu trzeciego nowego sygnału pochodzącego z próbek sygnałów wejściowych. Iloczyn sygnałów w dziedzinie czasu jest równy ich splotowy w dziedzinie częstotliwości. Rys. 2.2 przedstawia układ realizujący operację mnożenia. Sygnały $a(n)$ oraz $b(n)$ zostają poddane operacji splotu i na wyjściu otrzymujemy sygnał wyjściowy $y(n)$ będący wynikiem splotu widm tych dwóch sygnałów co przedstawia poniższe wyrażenie:

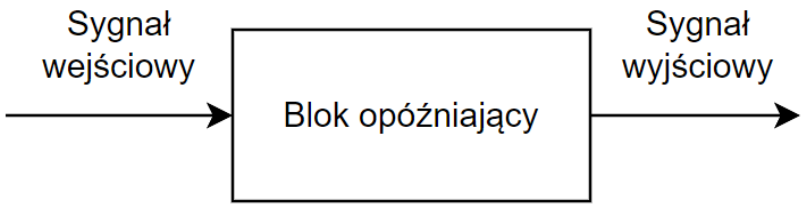
$$y(n) = a(n) * b(n) \tag{2.3}$$



Rys. 2.2 Schemat blokowy układu mnożenia

Operacja opóźnienia używana jest do opóźnienia w czasie każdej kolejnej próbki sygnału wejściowego. Operacja ta używana jest między innymi w akustyce w celu opóźnienia sygnału dźwiękowego. Rys. 2.3 przedstawia schemat blokowy układu opóźnienia. Sygnał wejściowy *x* zostaje przekazany do bloku opóźniającego, który realizuje opóźnienie próbek sygnału *n* wejściowego o zadaną wartość *d*. Sygnał wyjściowy *y* jest opóźnionym sygnałem wejściowym.

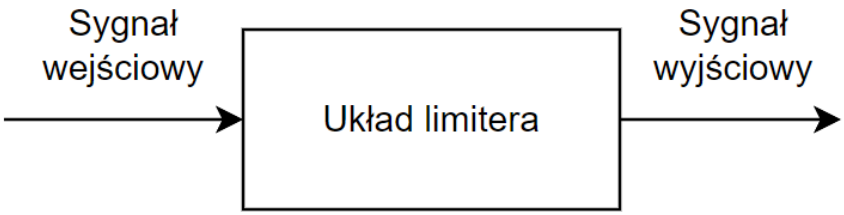
$$y(n) = x(n - d) \tag{2.4}$$



Rys. 2.3 Schemat blokowy układu opóźniającego

Organicznie amplitudy sygnału zapobiega przekroczeniu amplitudy sygnału czyli sytuacji, kiedy np. nadmierna wartość amplitudy mogłaby spowodować uszkodzenie głośnika na skutek wychylenia membrany. Nadanie maksymalnej wartości powoduje, że sygnał ograniczony np. do wartości -3dB nie będzie mógł przekroczyć tej wartości. Jeżeli sygnał znajduje się poniżej tej wartości zostaje przesłany w postaci niezmienionej. Układami realizującymi ograniczenie amplitudy są limitery. Schemat blokowy wraz z wyprowadzeniem został przedstawiony poniżej. Dla poniższej zależności w przypadku wartości mniejszej od wartości maksymalnej z sygnał przyjmuje wartości niezmienione, natomiast przy przekroczeniu wartości granicznej przyjmuje wartości maksymalne z.

$$y(n) = \begin{cases} x(n) & dla\ x(n) < z \\ z & dla\ x(n) \geq z \end{cases} \tag{2.5}$$

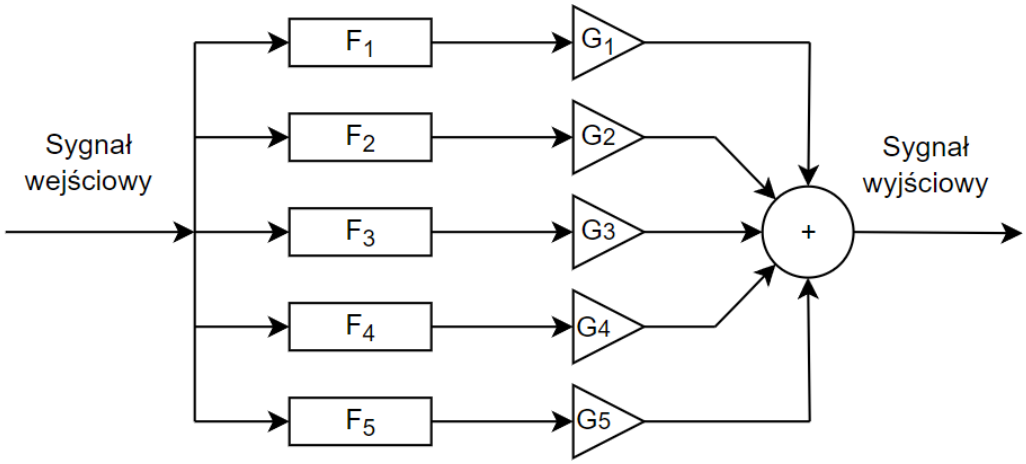


Rys. 2.4 Schemat blokowy układu ograniczającego amplitudę

Kształtowanie charakterystyki przez filtrację pozwala na manipulację widmem czyli reprezentacją częstotliwościową sygnału. Korektory graficzne pozwalają na manipulację widmem sygnału, wzmacniając bądź tłumiąc określone przedziały

częstotliwościowe. Schemat blokowy z zależnością matematyczną realizującą kształtowanie charakterystyki przedstawiono poniżej. Dla każdego pasma częstotliwości F_n dokonywane jest mnożenie poprzez wartość wzmocnienia zadaną dla tego pasma G_n.

$$y = F_1 \cdot G_1 + F_2 \cdot G_2 + F_3 \cdot G_3 + F_4 \cdot G_4 + F_5 \cdot G_5 \tag{2.6}$$



Rys. 2.5 Schemat blokowy układu korektora graficznego

2.2. Efekty akustyczne wpływające na zależność czasową

Efekty wpływające na zależność czasową towarzyszą nam na co dzień. Są one zjawiskiem naturalnym w otaczającym nas świecie, natomiast zagadnienie jakim jest cyfrowe przetwarzanie sygnałów pozwoliło w szerszym stopniu wpłynąć na dany sygnał dzięki czemu uzyskana została większa kontrola nad zmianą opóźnienia sygnału.

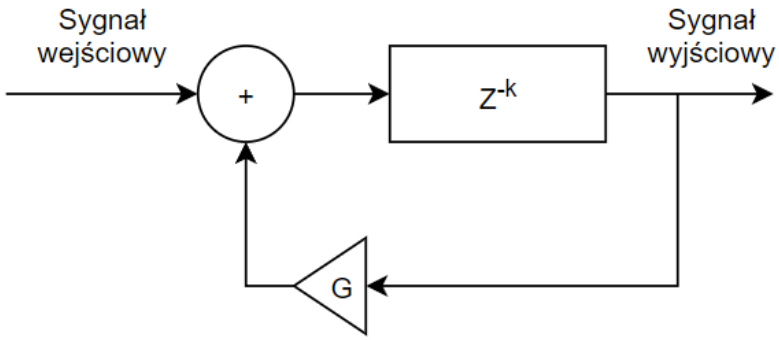
2.2.1. *Pogłos*

Pierwszym przykładem efektu, który wpływa na zależność czasową jest pogłos (ang. *reverb*). Pogłos jest zjawiskiem akustycznym polegającym na przedłużeniu czasu trwania dźwięku, które wywołane jest działaniem fal odbitych. Takie zjawisko można zaobserwować poprzez wydawanie bardzo głośnych dźwięków w małym pomieszczeniu (pogłos rodzaju *room*) , bądź też w dużym pomieszczeniu np. w kościele (określany jest jako *hall*) . Zasada działania efektu pogłosu opiera się o działanie filtru grzebieniowego bez sprzężenia zwrotnego. Filtr grzebieniowy jest to jeden z podstawowych filtrów, którego działanie polega na dodawaniu sygnału do jego opóźnionej wersji sygnału wejściowego. Schemat blokowy układu realizującego efekt pogłosu został przedstawiony na rys. 2.6. Sygnał wejściowy zostaje wprowadzony do bloku sumującego, a następnie wynik sumowania wprowadzony jest do bloku opóźniającego. Po przejściu przez blok opóźniający sygnał w sprzężeniu zwrotnym sygnał jest wprowadzany na układ

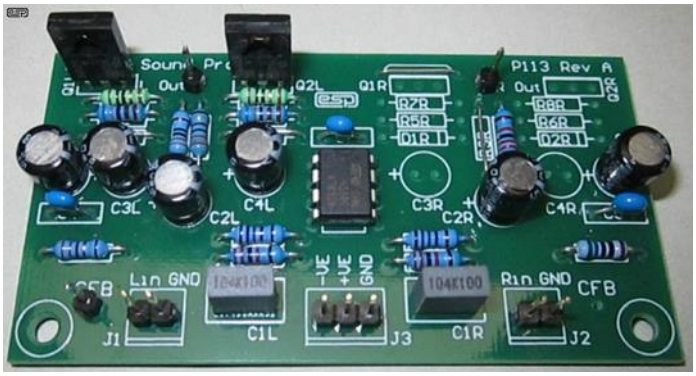
wzmacniacza, a z niego na drugie wejście sumatora. Taki sygnał zawierający sumę sygnału wejściowego oraz pogłos, który zostaje podany na wyjście układu. dyskretna realizacja efektu pogłosu opisana jest zależnością:

$$y(n) = x(n) + G \cdot y(n - d) \tag{2.7}$$

gdzie $y(n)$ to sygnał wyjściowy, G to wzmocnienie sygnału, zaś n oznacza kolejną próbkę, natomiast d jest to opóźnienie sygnału. Przykładem układu realizującego pogłos jest układ firmy Elliott Sounds Products o nazwie P113 , przedstawiony na rys. 2.7.



Rys. 2.6 Schemat blokowy efektu pogłosu



Rys. 2.7 Układ realizujący pogłos

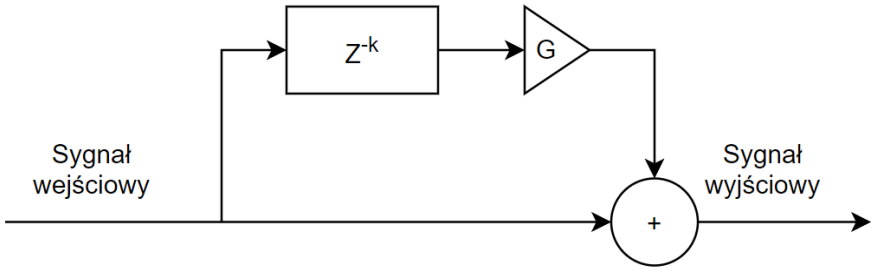
2.2.2. *Opóźnienie*

Opóźnienie jest efektem, który może najbardziej kojarzyć się z wpływem sygnału na zależność czasową. Efekt ten wpływa bezpośrednio na sygnał wejściowy i w formie niezmienionej przesyła go do wyjścia opóźniając go i łącząc z sygnałem podstawowym. Schemat blokowy realizujący opóźnienie został przedstawiony na rys 2.8. Sygnał wejściowy zostaje przekazany do bloku sumującego oraz do bloku realizującego opóźnienie. Z bloku opóźniającego sygnał jest przekazywany do wzmacniacza, który w zależności od preferencji użytkownika może wzmocnić lub stłumić sygnał opóźniony. Sygnał opóźniony w sumatorze zostaje dodany do oryginalnego sygnału wejściowego

i zostaje przekazany do wyjścia układu. Cyfrowa realizacja opóźnienia przedstawia się zależnością:

$$y(n) = x(n) + G \cdot x(n - d), \tag{2.8}$$

gdzie $y(n)$ oznacza sygnał wyjściowy, $x(n)$ sygnał wejściowy, natomiast G oznacza wzmocnienie zaś n oznacza numer próbki oraz d oznacza opóźnienie.



Rys. 2.8 Schemat blokowy układu opóźnienia

Przykładem praktycznej realizacji efektu opóźnienia jest urządzenie firmy *SubZero* o nazwie *Deep Freeze* został przedstawiony na rys. 2.9. Układ wyposażony jest w 3 pokrętła i jeden przycisk. Pokrętło Echo odpowiada za dodanie do opóźnienia charakterystycznego echa, potencjometr Time odpowiada za zmianę czasu opóźnienia efektu, natomiast Feedback odpowiada za natężenie dodanego efektu do sygnału wejściowego czyli wpływa na ilość końcowego efektu opóźnienia w sygnale wyjściowym.

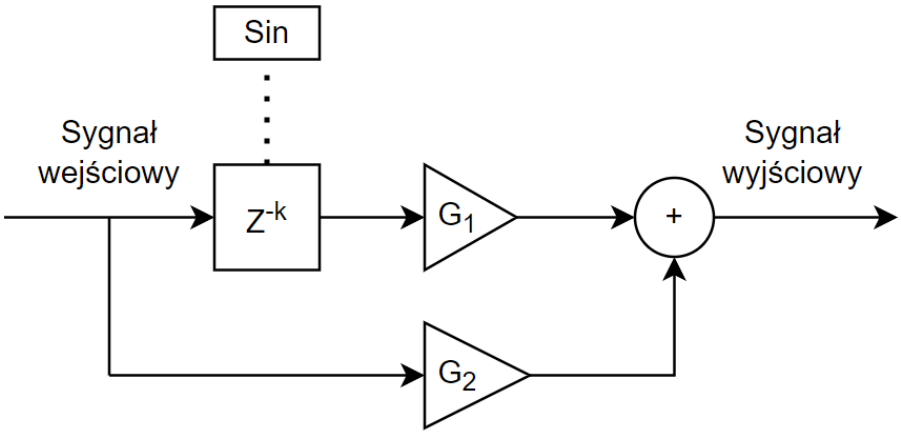
Rys. 2.9 Układ realizujący opóźnienie

2.3. Efekty akustyczne modulujące sygnał

Modulacja sygnału jest jednym z kluczowych elementów w trakcie poruszania zagadnień związanych z cyfrową obróbką sygnałów audio. Efekty modulacji pozwalają na całkowitą zmianę sygnału wejściowego, który po przetworzeniu może całkowicie nie przypominać oryginału. Modulowanie sygnału w dzisiejszych czasach daje możliwość na uzyskanie dowolnego dźwięku jaki jest możliwy do wytworzenia.

2.3.1. *Modulacja z użyciem sygnału sinusoidalnego*

Pierwszym przykładem efektu realizującego modulację sygnału jest modulacja sygnałem sinusoidalnym potocznie zwany jako *flanger*. Dźwięk uzyskany w wyniku modulacji efektem *flanger* można porównać do dźwięków, które wydają istoty pozaziemskie w hollywoodzkich produkcjach filmowych. Schemat blokowy realizujące efekt *flanger* został przedstawiony na rys. 2.10. Sygnał wejściowy zostaje przekazany na blok, który realizuje efekt opóźnienia przy pomocy funkcji sinusoidalnej. Tak opóźniony sygnał zostaje przekazany do bloku wzmocnienia gdzie sygnał w zależności od preferencji użytkownika może zostać stłumiony bądź wzmocniony, a następnie trafia do sumatora. W sumatorze sygnał wejściowy zostaje dodany z sygnałem zmodulowanym, a następnie przekazany do wyjścia układu.



Rys. 2.10 Schemat blokowy realizujący efekt flanger

Zależność realizująca efekt *flanger* opisana może być następująco:

$$y(n) = x(n) + G \cdot x\big(n - d(n)\big), \tag{2.9}$$

gdzie $y(n)$ oznacza sygnał wyjściowy, $x(n)$ sygnał wejściowy, natomiast G oznacza wzmocnienie zaś n oznacza numer próbki. Zmienna $d(n)$ przedstawia się zależnością:

$$d(n) = \frac{D}{2}\left[1 - cos\left(2\pi n f_{Cycle}\right)\right] \quad (2.10)$$

gdzie D to czas opóźnienia sygnału, natomiast n oznacza numer próbki sygnału zaś f_{Cycle} to częstotliwość sygnału kosinusoidalnego dla której realizowany jest efekt *flanger*. Wartości parametrów dla sygnału *flanger* to zakres od 0.25 do 25 [ms] dla czasu opóźnienia oraz częstotliwości rzędu kilkunastu herców. Dla efektu *flanger* stosowany jest sygnał sinusoidalny.

Przykładem układu realizującego efekt *flanger* jest *FROSTBITE* wykonany przez firmę *SubZero* przedstawiony został na rys. 2.11. Układ wyposażony jest w 3 pokrętła oraz jeden przycisk. Pokrętło *Color* odpowiada za regulację powrotu efektu, podczas gdy *Range* ogranicza zakres implementacji efektu, natomiast *Rate* reguluje szybkość efektu w trybie normalnym. Przycisk umieszczony na dole układu umożliwia włączenie oraz wyłączenie efektu. Przełącznik umieszczony na górze układu umożliwia również zastosowanie filtra, kiedy jest on przełączony w pozycję górną.

Rys. 2.11 Układ realizujący efekt *flanger*

2.3.2. *Modulacja sygnału oscylatorem niskich częstotliwości*

Drugim przykładem efektu modulacji jest modulacja sygnału oscylatorem niskich częstotliwości szerzej znany jako *chorus*. Efekt ten pozwala na uzyskanie charakterystycznego brzmienia chóralnego, stąd też nazwa efektu. Sygnał przetworzony przez efekt *chorus* sprawia wrażenie bardziej pełnego brzmienia ponieważ sygnał wejściowy jest podzielony na dwa sygnały, które są ze sobą zsumowane przed wyjściem.

Schemat blokowy na rys. 2.12 przedstawia sposób działania efektu *chorus*. Jest on bardzo podobny do efektu *flanger* dlatego też często zestawia się te efekty ze sobą w celu porównania. Sygnał wejściowy zostaje podany na blok opóźniający, który modulowany jest sygnałem wolno zmiennego oscylatora (ang. *Low Frequency Oscillator* - LFO) . Sygnał zmodulowany w ten sposób trafia do wzmacniacza. Następnie w sumatorze dodawany jest do siebie sygnał zmodulowany oraz sygnał wejściowy. Z sumatora dźwięk zostaje przekazany do wyjścia układu.

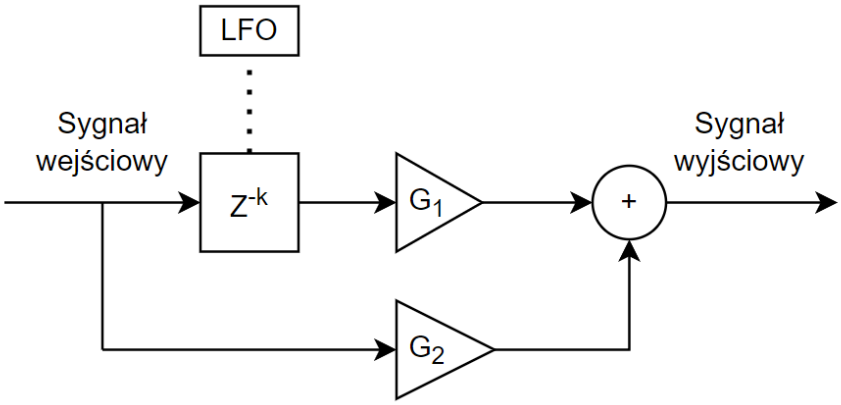
Zależność matematyczna opisująca efekt *chorus* sprowadza się wyrażenia:

$$y(n) = \frac{1}{2}x(n) + \frac{1}{2}x[n - d(n)], \tag{2.11}$$

gdzie $y(n)$ oznacza sygnał wyjściowy, $x(n)$ sygnał wejściowy zaś n oznacza numer próbki. Zmienna $d(n)$ przedstawia się zależnością:

$$d(n) = D\left[0{,}5 + LFO\left(2\Pi n f_{Delay}\right)\right], \tag{2.12}$$

gdzie D to czas opóźnienia sygnału, LFO oscylator niskich częstotliwości, natomiast n oznacza numer próbki sygnału zaś f_{Delay} to częstotliwość sygnału kosinusoidalnego dla której realizowany jest efekt *chorus*. Wartości sygnału LFO dla której jest realizowany efekt chóru przyjmuje wartości opóźnienia rzędu kilkunastu [ms] oraz częstotliwość na poziomie pojedynczych herców. Dla efektu *chorus* stosowany jest sygnał kosinusoidalny.



Rys. 2.12 Schemat blokowy efektu *chorus*

Jako fizyczny przykład realizacji tego efektu można wziąć urządzenie *Choir Boy Chorus* wykonany przez firmę *SubZero*. Układ wyposażony jest w 3 pokrętła oraz jeden przycisk. Pokrętło *Depth* kontroluje długość linii opóźnienia, natomiast pokrętło *Speed* odpowiada za prędkość zmiany czasu w przebiegu LFO. Potencjometr *Volume*

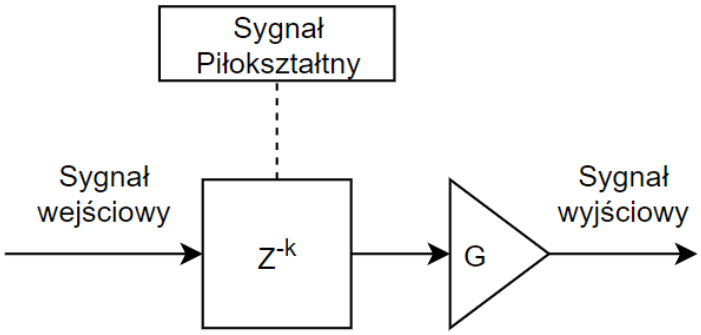
odpowiada za zmianę głośności efektu, zaś przycisk umieszczony pod nazwą układu odpowiada za aktywację, bądź też dezaktywację całego efektu.

Rys. 2.13 Układ realizujący efekt *chorus*

2.3.3. *Modulacja tonu sygnału*

Kolejnym efektem wartym uwagi podczas omawiania efektów modulacji jest modulacja tonu sygnału, która polega na zmianie tonacji dźwięku , czyli zmianie jego częstotliwości. Dzięki zastosowaniu tego rodzaju efektu dźwięk można obniżyć i otrzymać bardziej basowe brzmienie (charakteryzujące się niższymi częstotliwościami), jak również podwyższyć w celu uzyskania bardziej sopranowego brzmienia (charakteryzującego się wyższymi częstotliwościami). Popularnym zastosowaniem tego rodzaju efektu jest telewizja oraz Internet, gdzie w celu zamaskowania głosu osoby, która chce pozostać anonimowa moduluje się jej głos dzięki czemu jest on nie możliwy do rozpoznania.

Zgodnie ze schematem z rys. 2.14. sygnał wejściowy zostaje podany na blok opóźniający, który modulowany jest sygnałem piłokształtnym. Następnie sygnał trafia do bloku wzmocnienia, gdzie według przyjętych parametrów może został stłumiony bądź też wzmocniony. Następnie sygnał zostaje przekazany do wyjścia układu.



Rys. 2.14 Schemat blokowy *pitch shiftera*

Zależność matematyczną realizującą funkcję *pitch shiftera* przedstawia wyrażenie :

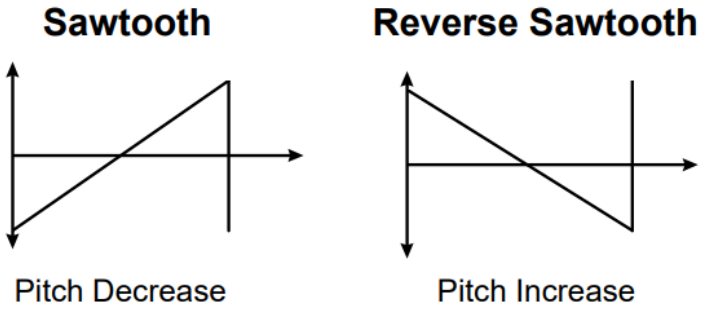
$$y(n) = G \cdot x[n - d(n)]. \tag{2.13}$$

gdzie $y(n)$ oznacza sygnał wyjściowy, $x(n)$ sygnał wejściowy, natomiast G oznacza wzmocnienie zaś n oznacza numer próbki. Zmienna $d(n)$ przedstawia się zależnością:

$$d(n) = D\left(sawtooth\left(2\pi n f_{shift}\right)\right) \tag{2.14}$$

gdzie D to czas opóźnienia sygnału, $sawtooth$ (*ang. sawtooth* – piłokształtny) funkcja realizująca sygnał piłokształtny, natomiast n oznacza numer próbki sygnału zaś f_{shift} to częstotliwość sygnału piłokształtnego dla którego realizowany jest efekt *pitch shiftera*.

W celu uzyskania wyższej tonacji należy użyć funkcji piłokształtnej o charakterze malejącym, natomiast w celu obniżenia tonacji należy użyć funkcji rosnącej jak zostało to zobrazowane na rys. 2.15.



Rys. 2.15 Sposoby implementacji sygnału piłokształtnego: *pitch decrease* oznacza obniżenie tonacji zaś *pitch increase* oznacza podwyższenie tonacji

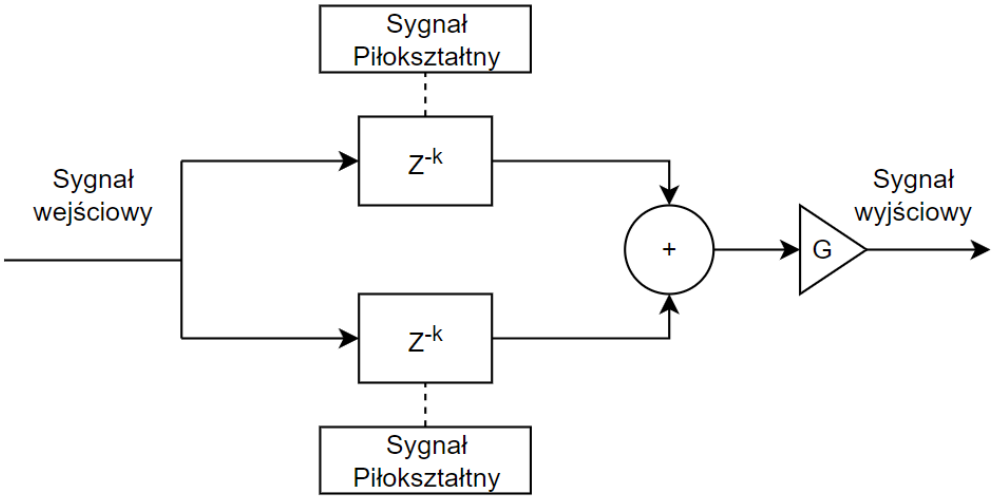
Przykładem układu realizującego efekt zmiany tonacji (*ang. pitch shift* – zmiana tonacji) jest Pitch Box wykonany przez firmę Mooer. Układ wyposażony jest w pokrętło *Pitch* odpowiadające za zmianę tonacji dźwięku oraz przełącznik pozwalający na

przełączenie pomiędzy funkcjami dodania harmonii, zmiany tonacji lub też implementacji efektu *detune*, natomiast przycisk umieszczony na dole *pitch shiftera* odpowiada za włącznie lub wyłącznie układu.

Rys. 2.16 Układ realizujący *pitch shifter*

2.3.4. *Podwójna modulacja tonu sygnału*

Efekt *detune* jest jednym z rodzajów *pitch shiftera*. Działanie tego efektu sprowadza się do rozdzielenia sygnału wejściowego na dwa niezależnie od siebie sygnały i zmianę ich tonacji o bardzo małą wartość rzędu pojedynczych herców. Implementacja tego efektu popularna jest szczególnie wśród wokalistów. Efekt ten różni się od efektu *chorus* tym, że w przypadku *detune* mamy do czynienia ze zmianą tonacji sygnału dzięki czemu uzyskujemy dwa sygnały o przeciwnej tonacji, które nie przypominają sygnału oryginalnego. Schemat blokowy efektu *detune* przedstawiono na rys. 2.17.



Rys. 2.17 Schemat blokowy efektu *detune*

Działanie sprowadza się do zastosowania dwóch gałęzi niezależnych realizujących efekt *pitch shiftera*. Po przejściu sygnału przez bloki zmiany tonacji sygnały zostają zsumowane w bloku sumatora i przekazane do wzmacniacza. Z bloku wzmocnienia sygnał trafia do wyjścia układu. Cyfrowa realizacja efektu *detune* sprowadza się do zastosowania wyrażenia :

$$y(n) = G\left[x\left(n - d(n_1)\right) + x\left(n - d(n_2)\right)\right] \tag{2.15}$$

gdzie $y(n)$ oznacza sygnał wyjściowy, $x(n)$ sygnał wejściowy, natomiast G oznacza wzmocnienie zaś n_1 oraz n_2 oznaczają numery próbek. Zmienne $d(n_1)$, $d(n_2)$ przedstawione są odpowiednio :

$$d(n_1) = D\left(-sawtooth\left(2\pi n_1 f_{shift_{up}}\right)\right), \tag{2.16}$$

$$d(n_2) = D\left(sawtooth\left(2\pi n_2 f_{shift_{down}}\right)\right), \tag{2.17}$$

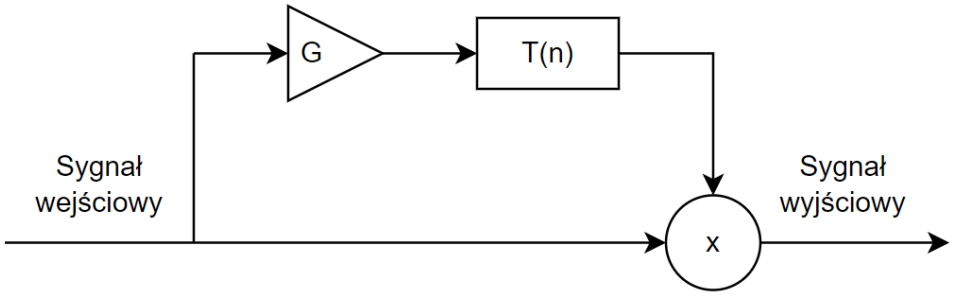
gdzie D to czas opóźnienia sygnału, $sawtooth$ funkcja realizująca sygnał piłokształtny, natomiast n_1 oraz n_2 oznaczają numery próbek sygnału zaś $f_{shift_{up}}$, $f_{shift_{down}}$ to częstotliwości sygnałów piłokształtnych dla których realizowany jest efekt zmiany tonacji w układzie *detune*.

Przykładem układu *detune* jest *Luxe Anti-Chorus* wykonany przez firmę *Digitech* przedstawiony na rys. 2.18. Układ wyposażony jest w pokrętło *Level*, które odpowiedzialne jest za regulację poziomu efektu *detune* oraz potencjometr *Detune* pozwalający na regulację efektu *detune* z zakresu od -0.5 do 0.5 tonu [5]. Przycisk *True Bypass* umożliwia bezpośrednie przesłanie sygnału z wejścia do wyjścia.

Rys. 2.18 Układ realizujący efekt *detune*

2.3.5. *Modulacja amplitudy*

Ostatnim z efektów modulacji poruszanym w ramach pracy jest tremolo. Mimo braku tak wielkiej sławy jak chorus czy flanger jest on również bardzo istotnym efektem akustycznym modulującym sygnał. Tremolo, słowo pochodzące z języka włoskiego oznaczające drżenie, to określenie rodzaju artykulacji, która polega na szybkim wykonaniu wielu dźwięków o tej samej wysokości. Schemat blokowy efektu tremolo przedstawiono na rys. 2.19.



Rys. 2.19 Schemat blokowy efektu tremolo

Sygnał wejściowy zostaje przekazany do bloku wzmocnienia, a następnie zostaje przekazany do bloku odpowiadającego za modulację wzmocnionego sygnału wejściowego. Sygnał zmodulowany zostaje w układzie pomnożony z sygnałem wejściowym, a następnie zostaje przekazany do wyjścia układu. Cyfrowa realizacja efektu tremolo przedstawia się wyrażeniem matematycznym:

$$y(n) = x(n) \cdot [1 + G \cdot T(n)], \tag{2.18}$$

gdzie $y(n)$ oznacza sygnał wyjściowy, $x(n)$ sygnał wejściowy, natomiast G oznacza wzmocnienie zaś n oznacza numer próbki. Zmienna $T(n)$ wyraża się zależnością:

$$T(n) = cos(2\pi n f_{mod}), \tag{2.19}$$

gdzie $T(n)$ oznacza sygnał wyjściowy, n numer kolejnej próbki sygnału, f_{mod} to wartość częstotliwości sygnału modulującego dla efektu tremolo.

Przykładem układu wykorzystującego efekt tremolo jest *Choka Tremolo* zaprojektowanu przez *TC ELECTRONIC* . Układ wyposażony jest w trzy pokrętła oraz przycisk. Potencjometry *Speed* oraz *Depth* odpowiadają kolejno za regulację częstości oraz intensywności efektu modulacji. Pokrętło *LFO* reguluje kształt fali oscylatora o niskiej częstotliwości w celu dostosowania charakteru dźwięku. Przycisk *True Bypass*

pozwala na przekazanie sygnału bezpośrednio z wejścia do wyjścia układu bez wprowadzania modulacji sygnału.

Rys. 2.20 Układ realizujący efekt tremolo

2.4. Podsumowanie

Powyższy przegląd przedstawia popularne efekty akustyczne używane powszechnie wraz z komercyjnymi przykładami stanowiącymi zaawansowany układ dla danego efektu. Dla każdego efektu omówienie schematu blokowego razem z równaniem realizującym dany efekt pozwala na zrozumienie sposobu implementacji z jednoczesną możliwością odtworzenia efektu na podstawie przedstawionego wyrażenia matematycznego.

Przedstawione powyżej przykłady efektów używane są najczęściej jako układy do instrumentów gitarowych. W przypadku cyfrowego przetwarzania sygnałów znacznie częściej używa się programów komputerowych. W przypadku komputerów osobistych mamy większy zakres możliwości związany z ich mocą obliczeniową co pozwala na efektywniejsze przetwarzanie większej ilości próbek w krótszym czasie.

3. Implementacja i symulacja efektów akustycznych w środowisku MatLAB

Środowisko MatLAB jest jednym z najpopularniejszych środowisk programistycznych dla inżynierów oraz naukowców. Ponad trzydzieści pięć lat doświadczenia pozwoliło twórcom na udoskonalenie środowiska oraz stworzenie wielu narzędzi ułatwiających pracę inżynierów. MatLAB posiada bogatą ilość dodatków dedykowanych zagadnieniom związanym z cyfrowym przetwarzaniem sygnałów. Dzięki temu implementacja poniższych efektów mogła zostać zrealizowana w prosty i zrozumiały sposób.

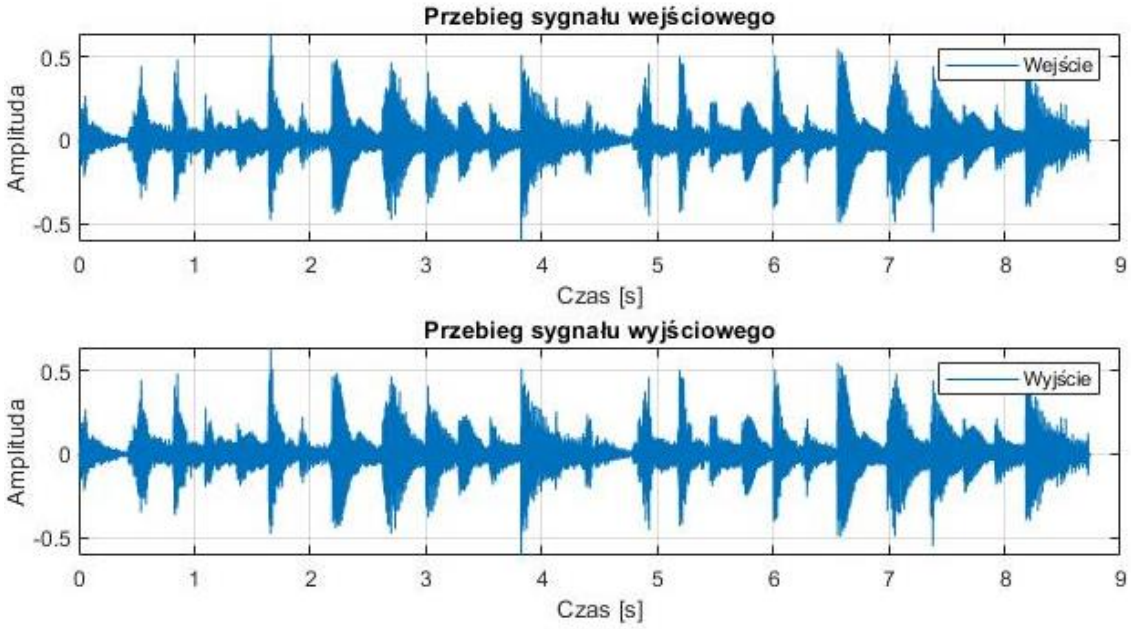
3.1. Wprowadzenie

Przygotowana została implementacja efektów akustycznych w postaci funkcji wykonanych w środowisku MatLAB. Każda funkcja wykonuje cyfrową realizację efektu zgodnie z jej nazwą. Funkcje realizujące efekty to kolejno: *Stereo.m, Reverb.m, Delay.m, Flanger.m, Chorus.m, Pitch_Shifter.m, Detune.m, Tremolo.m*. Każda z funkcji umożliwia użytkownikowi na wprowadzenie argumentów pozwalających na przetworzenie wybranego pliku poprzez podanie paramentów ścieżki, nazwy oraz rozszerzenia danego pliku jako argumentu. Są to parametry wspólne dla każdej funkcji. Pozostałe parametry, które są indywidualne dla danej funkcji zostały omówione dalej w odpowiednim podrozdziale. Wykresy dla każdego efektu przedstawiają zmianę sygnału w czasie oraz częstotliwości. Dodatkowo możliwe jest dokonanie subiektywnej oceny przetworzonego sygnału wejściowego przez odsłuch. Po uruchomieniu programu, dla każdej funkcji wykonywane zostaje najpierw otworzenie sygnału oryginalnego, a następnie sygnału poddanego obróbce cyfrowej. Sygnał wyjściowy zostaje zapisany do folderu *Zapisane_pliki_koncowe*, w którym znajdują się zapisane ścieżki poddane obróbce cyfrowej, w celu ułatwienia możliwości odtworzenia nagrania w przyszłości. Na potrzeby pracy inżynierskiej symulacja została zrealizowana dla przykładowych parametrów celem przedstawienia zobrazowań symulacyjnych. Wszystkie zobrazowania symulacyjne zostały wykonane na podstawie kodu zaprezentowanego w listingu 2.

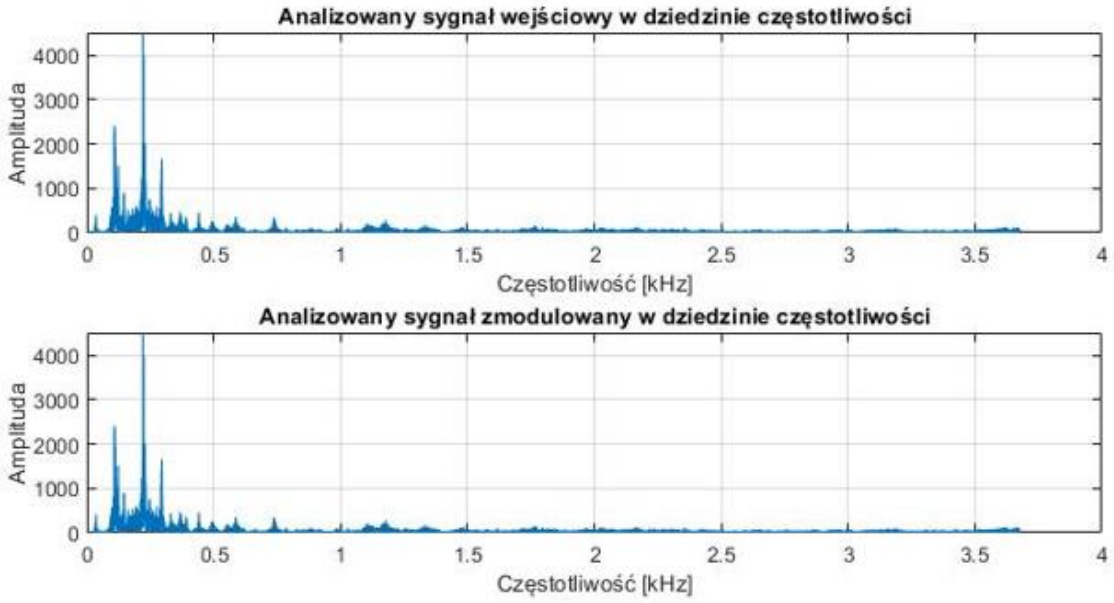
3.2. Sumowanie kanałów

Funkcja realizująca sumowanie kanałów dla środowiska MatLAB została przedstawiona w załączniku i oznaczona jako listing 1. Realizuje ona układ sumatora

zgodnie ze schematem blokowym przedstawionym na rys. 2.1 oraz równanie (2.2). Funkcja po przyjęciu parametrów pliku zadanego przez użytkownika dokonuje jego analizy . W przypadku, jeżeli utwór jest jednokanałowy wykonuje się instrukcja warunkowa *if z linii 7* i sygnał przekazywany jest w formie niezmienionej. Jeżeli sygnał jest w postaci dwukanałowej wykonana zostaje instrukcja *elseif z linii 9* i następuje sumowanie kanałów oraz normalizacja poziomu głośności w celu dostosowania jej do rzeczywistego sygnału mono. Ostatnia instrukcja *else (linia 22)* została umieszczona w przypadku ewentualnej konieczności o poinformowaniu użytkownika o błędzie.

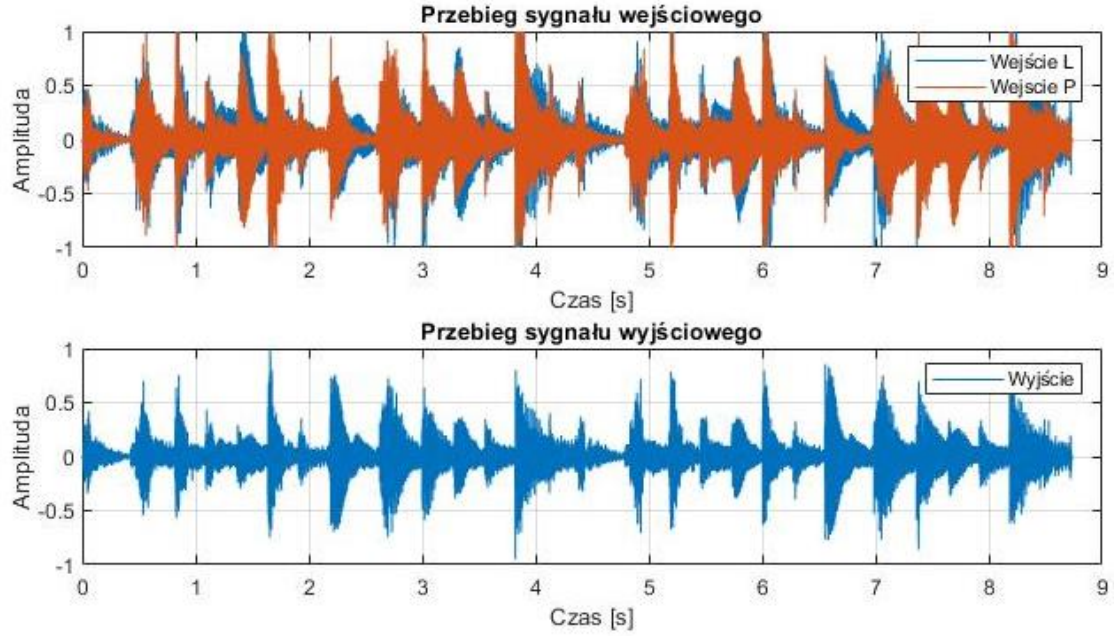


Rys. 3.1 Przykładowa realizacja zależności amplitudy od czasu dla wejścia jednokanałowego

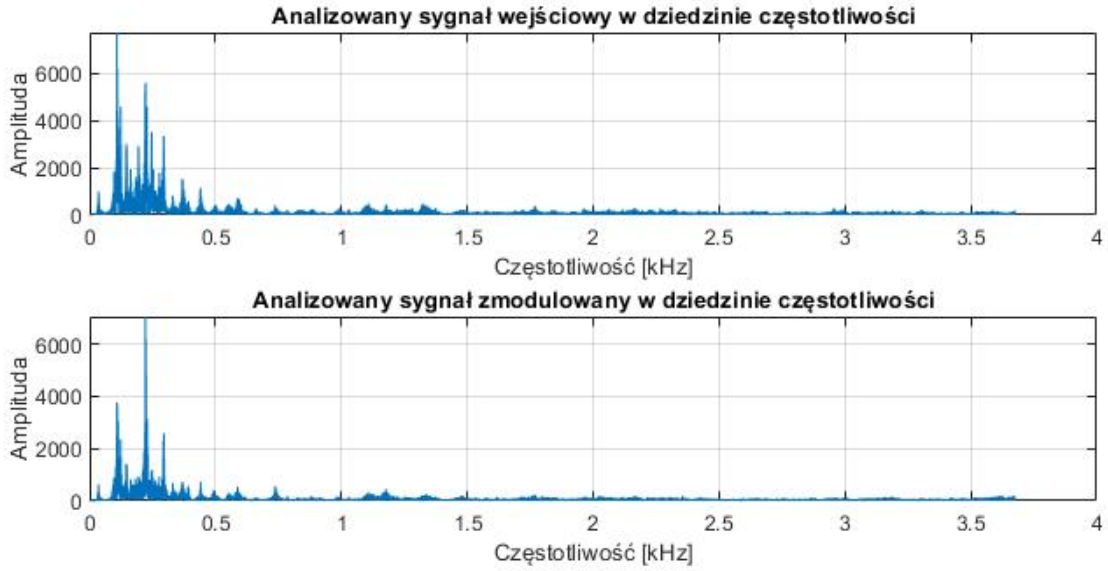


Rys. 3.2 Przykładowa realizacja widma dla wejścia jednokanałowego

Powyższe wykresy przedstawiają przebieg amplitudowy oraz częstotliwościowy dla sygnału jednokanałowego. Powyższe symulacje zostały wykonane w oparciu o kod znajdujący się w listingu 2, który służy również do symulowania pozostałych efektów wykonanych w środowisku MatLAB. Z powyższych symulacji sygnał wyjściowy prezentuje identyczne widmo oraz amplitudę tak jak sygnał wejściowy co jest zgodnie z oczekiwaniem.



Rys. 3.3 Przebieg amplitudowy dla sygnał wejściowego stereofonicznego

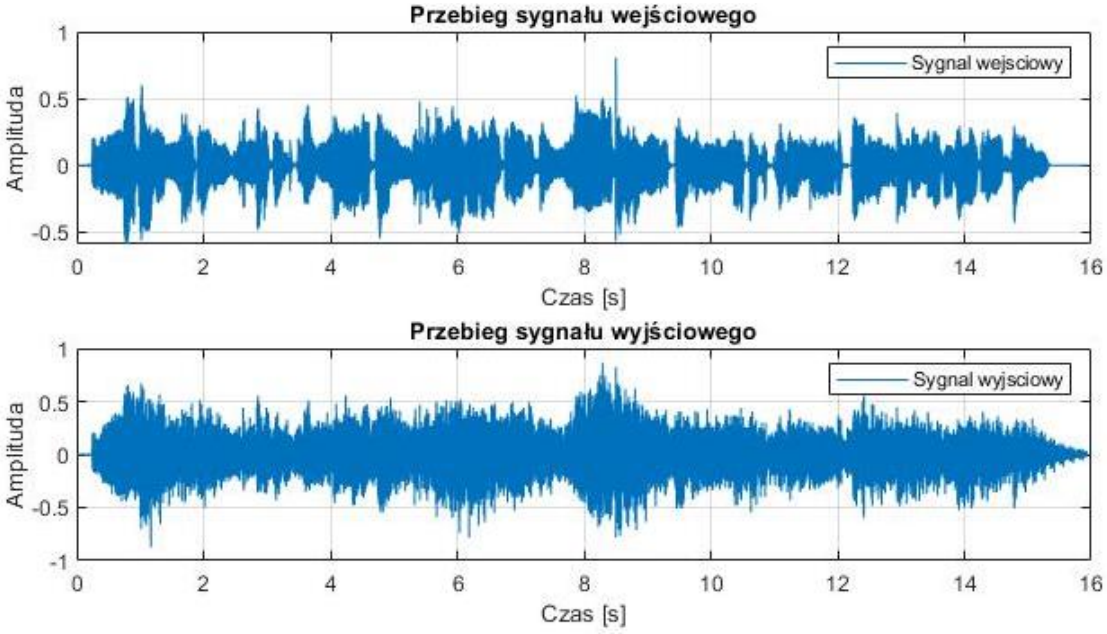


Rys. 3.4 Przykładowa realizacja widma dla wejścia stereofonicznego

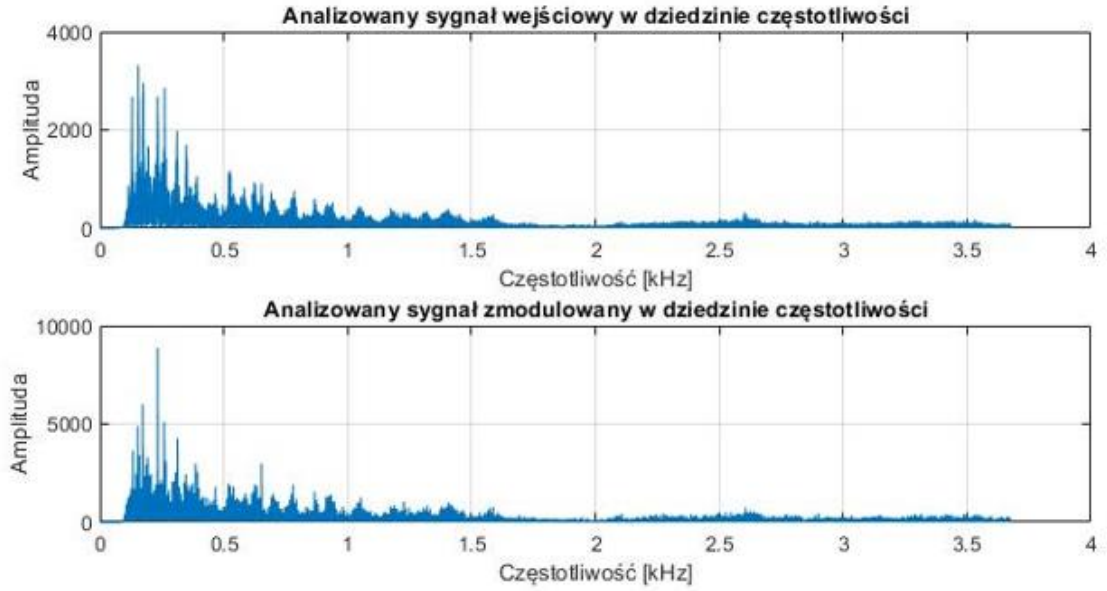
Powyższe wykresy przedstawiają przykładową obróbkę cyfrową sygnału stereofonicznego. Sygnał wyjściowy (monofoniczny) charakteryzuje się zmniejszoną amplitudą – co jest wynikiem jej normalizacji oraz mniejszą ilością prążków w widmie w porównaniu do sygnału wejściowego.

3.3. Pogłos

Funkcja realizująca pogłos została oznaczona jako listing 3. została oparta o schemat blokowy przedstawiony na rys 1.6 oraz zależność matematyczną (2.7). Funkcja pobiera od użytkownika parametry pliku dźwiękowego, wzmocnienia oraz opóźnienia. Realizacja wyrażenia została przedstawiona w pętli *for (linia 12)*, gdzie sygnał wejściowy jest sumą sygnału wejściowego oraz sygnału wyjściowego poddanego obróbce cyfrowej. W *linii 17* zostało zastosowanie pomnożenie sygnału przez wartość 0.7 w celu uniknięcia efektu przebicia podczas zapisywania pliku.



Rys. 3.5 Przykładowa symulacja zależności amplitudy od czasu dla efektu pogłosu

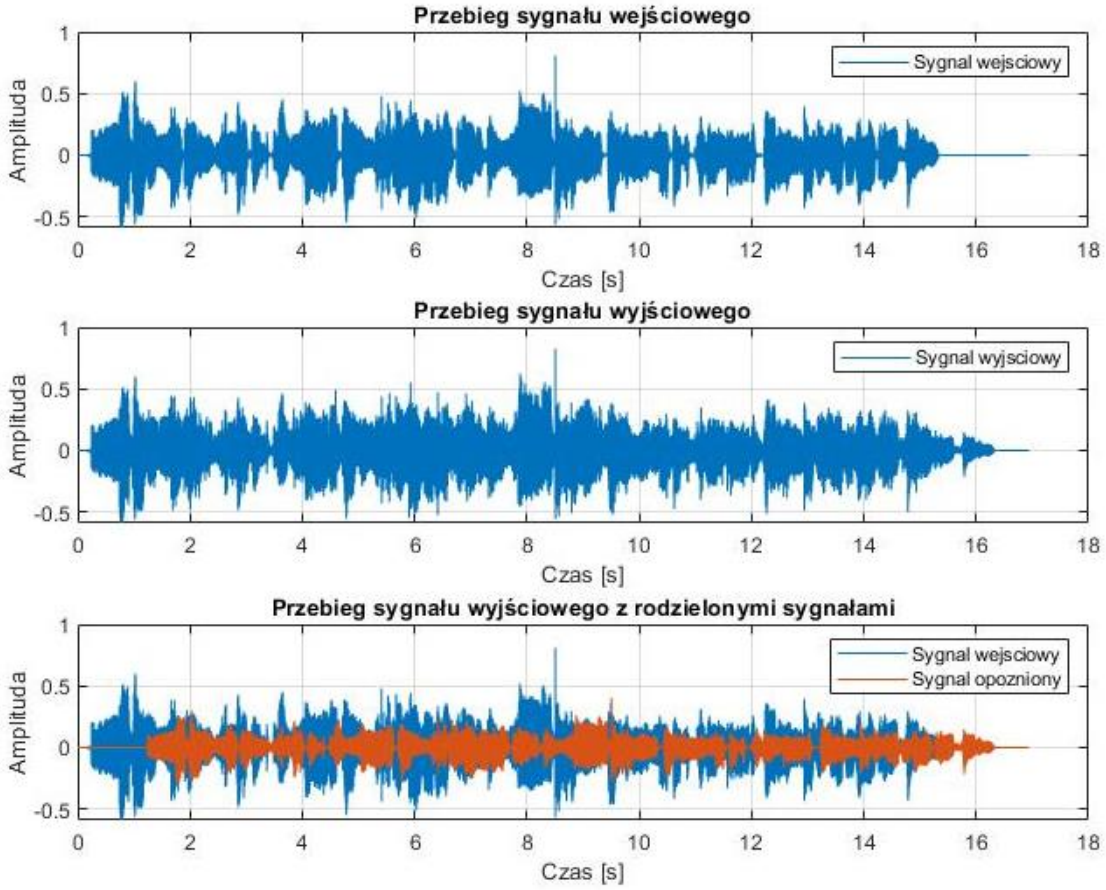


Rys. 3.6 Przykładowa symulacja widmowa pogłosu

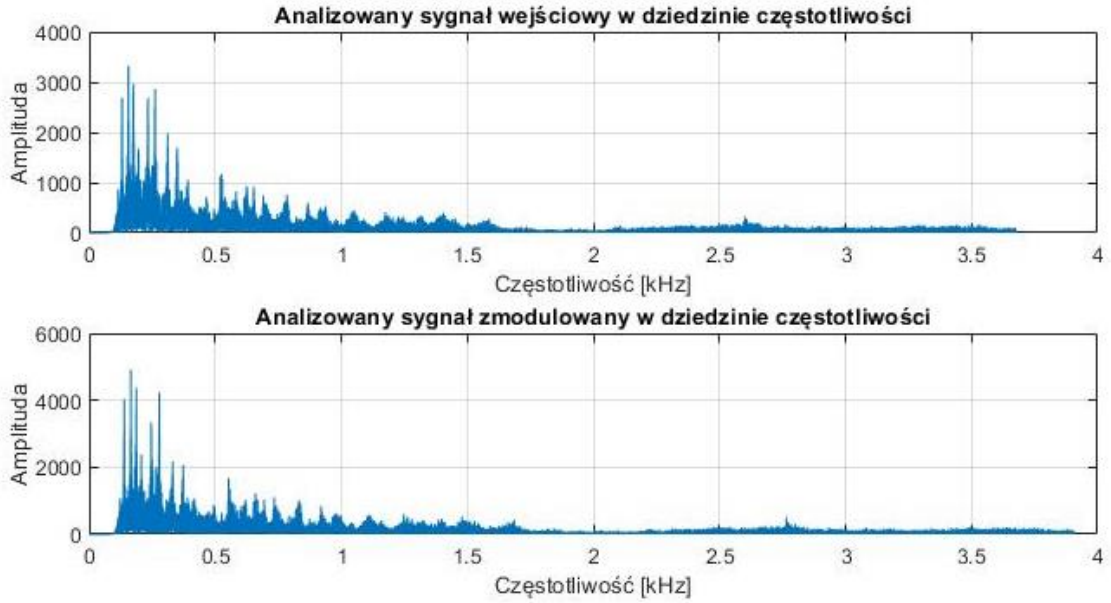
Symulacja dla efektu pogłosu została zrealizowana dla parametrów wzmocnienia na poziomie 0,7 oraz opóźnienia 150 [ms]. Sygnał amplitudy zobrazowuje działanie efektu pogłosu oraz nałożenie się na niego sygnału oryginalnego powodując sumaryczne zwiększenie amplitudy zaś widmo sygnału prezentuje nam wzmocnienie prążków oraz większe natężenie widma w okolicach 300 [Hz].

3.4. Opóźnienie

Funkcja realizująca opóźnienie dla środowiska MatLAB została przedstawiona w załączniku i oznaczona jako listing 4. Funkcja *Delay* realizuje efekt opóźnienia pobierając od użytkownika parametry pliku dźwiękowego, wartość wzmocnienia dla pliku oraz wartość opóźnienia. Za realizację tego efektu odpowiadają dwa bufory. Użycie buforów zostało oparte na zależności matematycznej oznaczonej jako (2.8) oraz schemat blokowy przedstawiony na rys. 2.8. Bufor wejściowy (*linia 8*) przechowuje sygnał wejściowy, natomiast bufor wyjściowy (*linia 9*) przechowuje opóźniony sygnał wejściowy z określonym wzmocnieniem. Jako wyjście *y* otrzymujemy sumę tych dwóch buforów czyli sygnał wejściowy, na który nałożona jest jego opóźniona kopia.



Rys. 3.7 Przykładowa symulacja amplitudowa funkcji *delay*

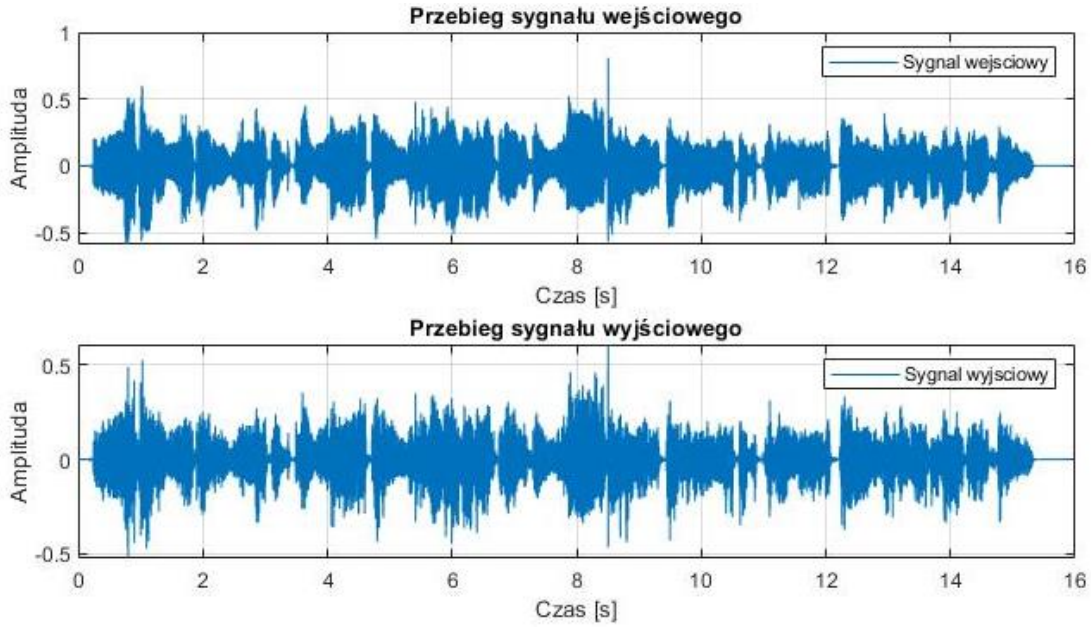


Rys. 3.8 Przykładowa realizacja częstotliwościowa funkcji delay

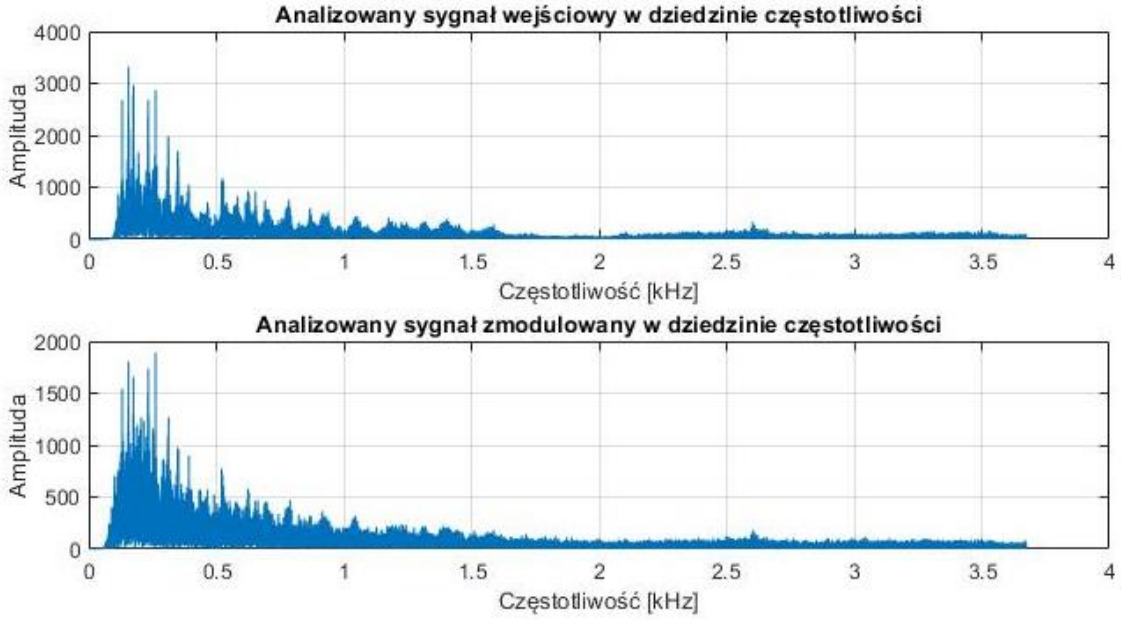
Parametrami zadeklarowanymi do realizacji przebiegu czasowego oraz częstotliwościowego jest wzmocnienie na poziomie 0.5 oraz opóźnienie 1000 [ms]. W celu lepszego zobrazowania przedstawiono sygnał wyjściowy jako sumę sygnału wejściowego oraz sygnału opóźnionego oraz został sporządzony trzeci wykres z rozdzielonymi sygnałami na którym sygnał opóźniony został przedstawiony kolorem pomarańczowym, dzięki czemu widać jego zmniejszoną amplitudę oraz opóźnienie w czasie w stosunku do sygnału wejściowego. Reprezentacja częstotliwościowa charakteryzuje się większą amplitudą prążków w związku z sumą sygnału oryginalnego i opóźnionego.

3.5. Flanger

Funkcja *flanger* realizuje cyfrowe przetworzenie sygnału wejściowego umożliwiające uzyskanie sygnału zmodulowanego o zadanych przez użytkownika parametrach wzmocnienia, opóźnienia oraz częstotliwości modulacji oraz wybranym pliku zgodnie ze schematem blokowym (rys. 2.10). Realizacja efektu została zamieszczona w listingu 5 oraz oparta o zależność (2.9), która w pętli *for (linia 11)* oblicza wartość modulacji dla efektu *flanger*. Zależność (2.10) którą realizuje zmienna *opóźnienie_flanger (linia 13),* realizuje schemat blokowy efektu wyznacza wartość sygnału wyjściowego dla efektu *flanger*. Do realizacji efektu *flanger* zalecane jest stosowanie wartości opóźnienia w zakresie 0.25 do 25 [ms]. Zaleca się również stosowanie niskich wartości częstotliwości w okolicach 10 herców [4].



Rys. 3.9 Przykładowy przebieg amplitudy w czasie dla funkcji *flanger*

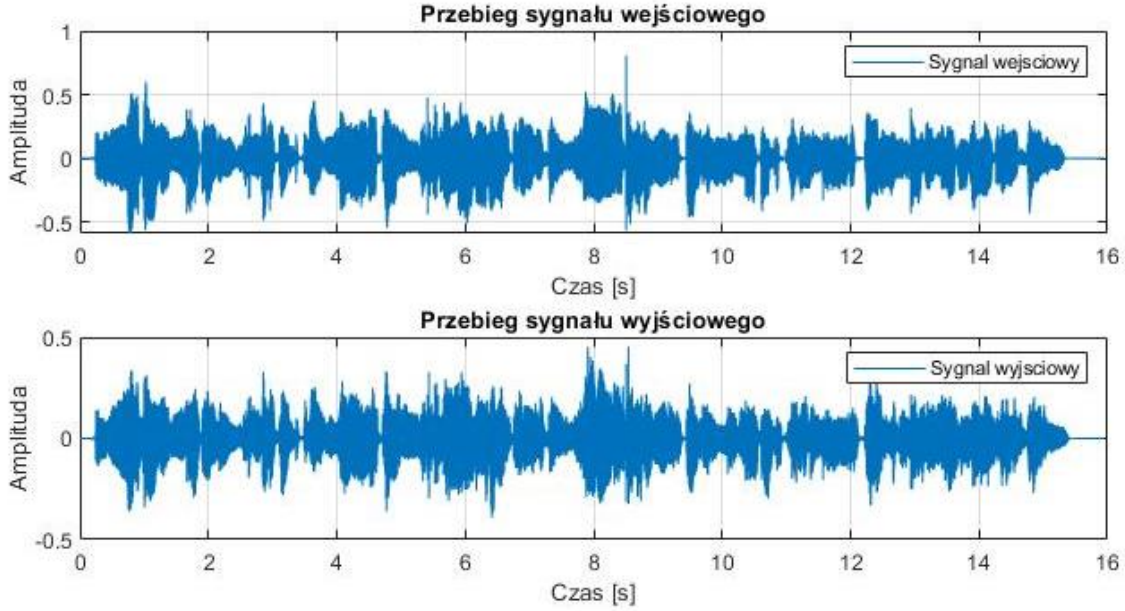


Rys. 3.10 Przykładowa realizacja częstotliwościowa dla funkcji *flanger*

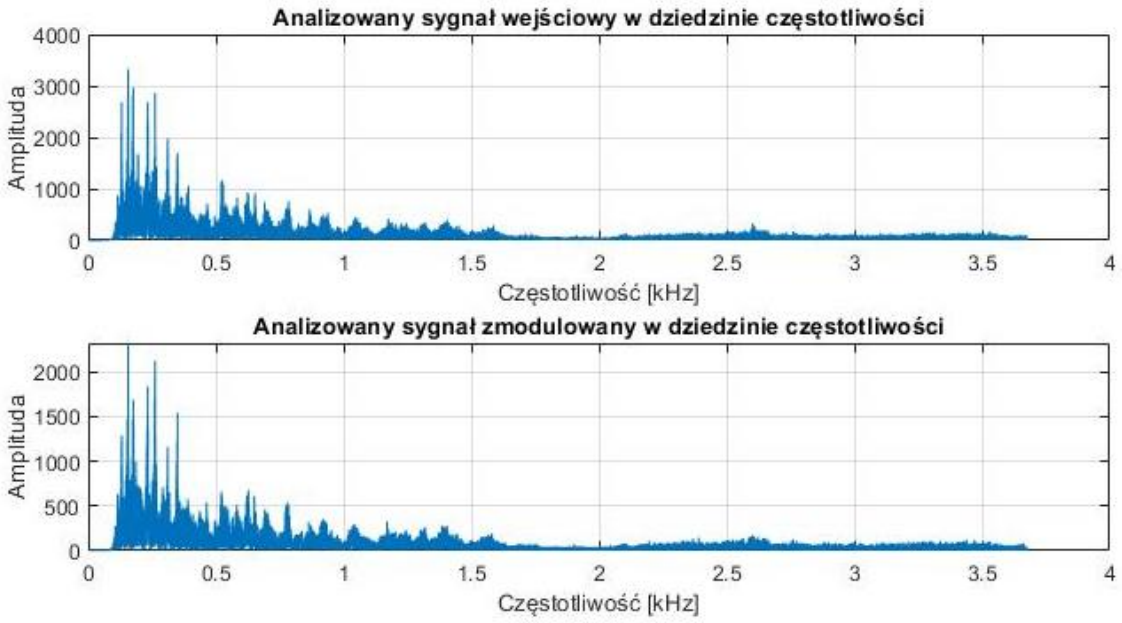
Przebieg amplitudy w czasie oraz częstotliwości dla efektu flanger o wzmocnieniu sygnału 0.5, opóźnieniu 10 [ms] oraz częstotliwości modulacji 10 [Hz] został przedstawiony powyżej. Zauważalne jest silne zmodulowanie amplitudy wyjściowej z jednoczesnym stłumieniem w związku z użyciem wzmocnienia na poziomie 0.5 dla obu sygnałów, natomiast widmo charakteryzuje się zmniejszoną wartością amplitud prążków z jednoczesnym zwiększeniem liczby prążków w okolicach częstotliwości 300 [Hz].

3.6. Chorus

Realizacja modulacji sygnałem LFO została przedstawiona w załączniku i oznaczona jako listing 6. Realizacja efektu *chorus* (schemat z rys 2.12) opierająca się o funkcję *chorus* sprowadza się do zastosowania wzorów (2.11) jako zmienna wyjściowa *y (linia 13)* oraz (2.12) jako zmienna *Chorus_delay (linia 12)*. Powyższa funkcja przyjmuje od użytkownika dwa parametry czasu opóźnienia oraz częstotliwości modulującej LFO, które umożliwiają użytkownikowi manipulację efektem oraz parametry pliku audio. Zalecane wartości dla realizacji efektu *chorus* to czas opóźnienia w okolicach 10 [ms] oraz niewielkie częstotliwości oscylujące w granicach 3 [Hz] [4]. Dla większych wartości zastosowanie efektu może być zbyt intensywne i efekt końcowy może znacznie odbiegać od oczekiwanego.



Rys. 3.11 Przykładowy przebieg amplitudowy dla funkcji *chorus*

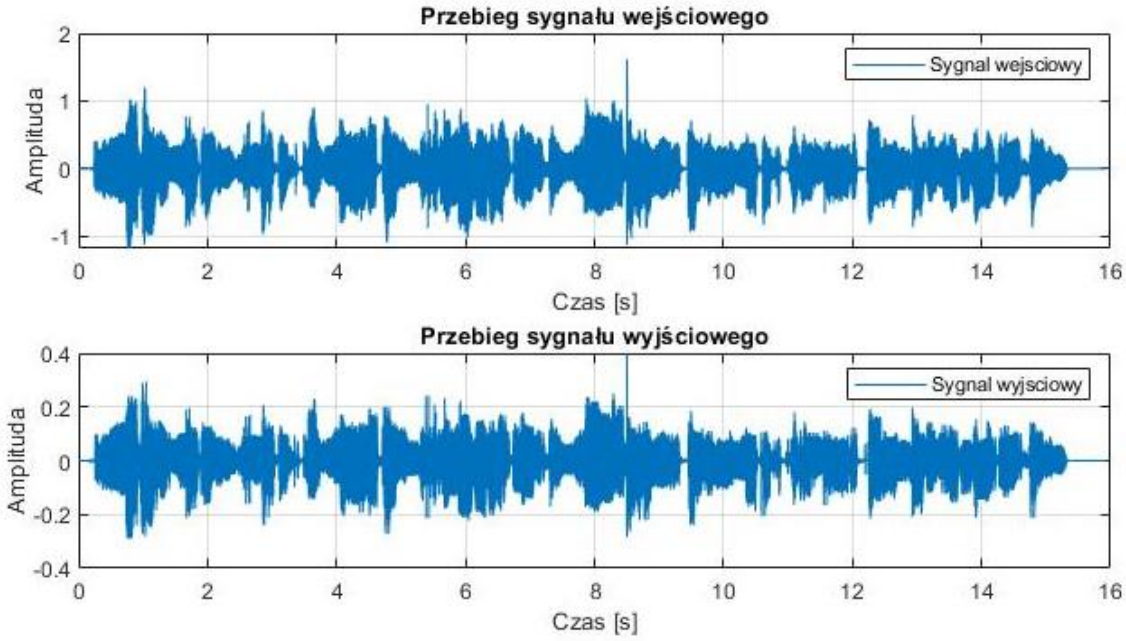


Rys. 3.12 Przykładowa realizacja częstotliwościowa dla funkcji *chorus*

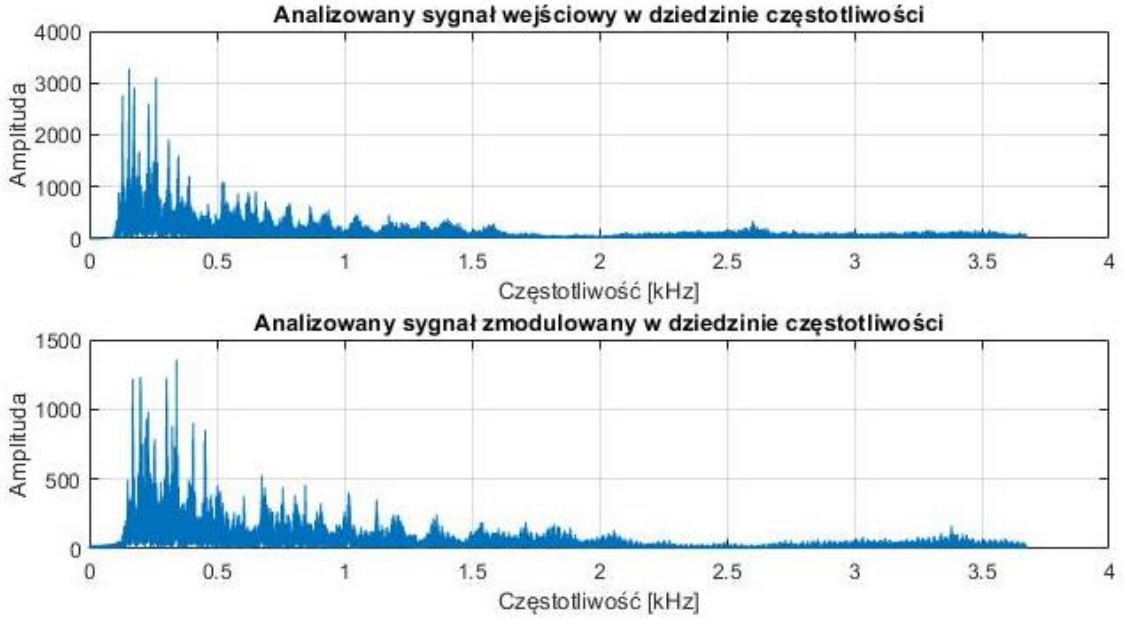
Symulacja funkcji chorus została wykonana dla parametrów wejściowych wynoszących 10 [ms] dla opóźnienia oraz 3 [Hz]. Analizując przebieg amplitudowy zauważalne są zmniejszenia wartości amplitudy dla sygnału wyjściowego oraz zmniejszenie się wartości prążków dla widma sygnału wyjściowego.

3.7. Pitch shifter

Realizacja zmiany tonacji dla środowiska MatLAB została oznaczona jako listing 7. Funkcja *Pitch_Shifter* realizuje efekt przesunięcia tonu dla wybranego przez użytkownika pliku zgodnie ze schematem z rys. 2.14. Parametr *up_or_down* dla argumentu *up* wykonuje pierwszą instrukcję warunkową *if (linia 12)*, która powoduje podwyższenie tonu sygnału zgodnie z zależnością (2.14) biorąc wartość ze znakiem minus. Zgodnie z rys. 2.15. podwyższenie tonu realizowane jest dla malejącej funkcji piłokształtnej. W przypadku wyboru argumentu *down (linia 18)* dla parametru *up_or_down* realizowana jest zależność (2.14), jednak posługując się rys. 2.15 funkcja piłokształtna ma charakter narastający w celu otrzymania sygnału wyjściowego o podwyższonym tonie.



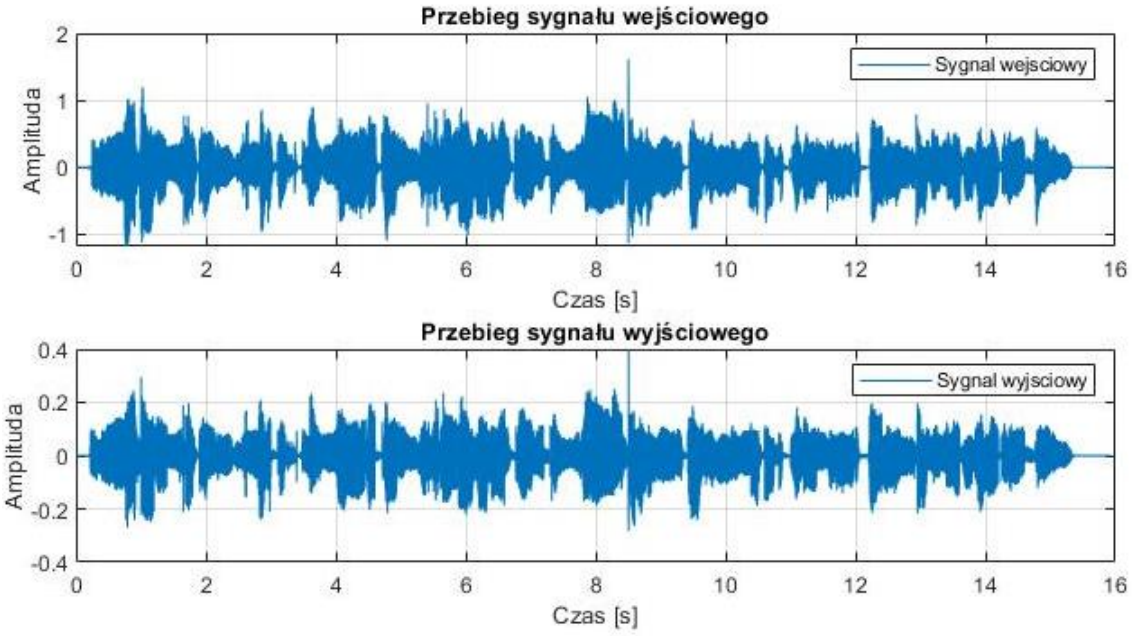
Rys. 3.13 Przykładowy przebieg amplitudowy funkcji *Pitch_Shifter* podwyższającej ton sygnału



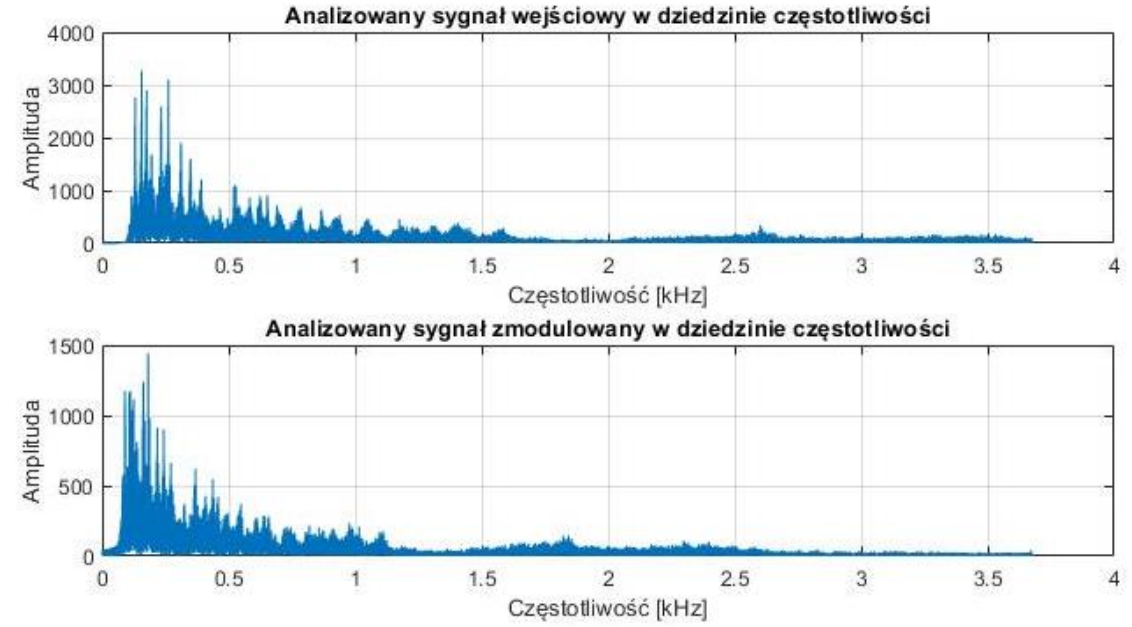
Rys. 3.14 Przykładowa realizacja częstotliwościowa funkcji *Pitch_Shifter* podwyższającej ton

Symulacja amplitudowa oraz częstotliwościowa została przeprowadzona dla podwyższonego tonu o wzmocnieniu 0.5 oraz opóźnieniu 30 [ms] i częstotliwości 5 [Hz]. Zauważalne zniekształcenia amplitudowe są spowodowane sposobem implementacji efektu poprzez użycie funkcji piłokształtnej, czemu towarzyszą zniekształcenia [10].

Zauważalne jest wzbogacenie widma w całym paśmie zwłaszcza przy częstotliwościach środkowych (okolice 1500 [Hz]).



Rys. 3.15 Przykładowa realizacja amplitudowa funkcji *Pitch_Shihter* obniżającej ton sygnału



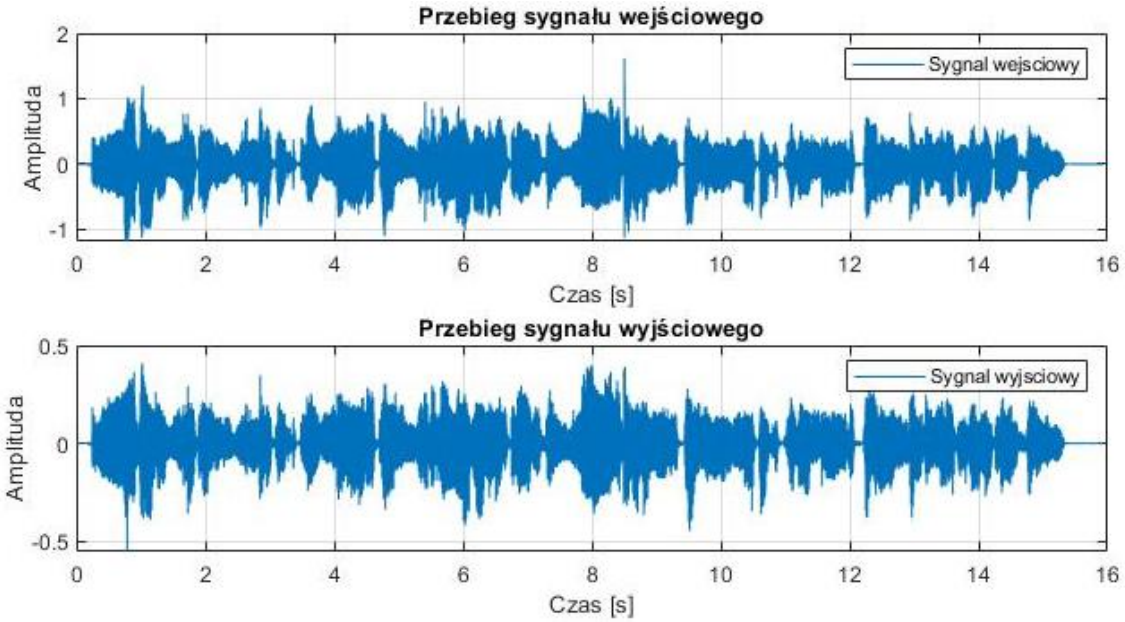
Rys. 3.16 Przykładowe widmo funkcji *Pitch_Shifter* obniżającej ton

Symulacja amplitudowa oraz częstotliwościowa przeprowadzona dla obniżonego tonu o wzmocnieniu 0.5 oraz opóźnieniu 30 [ms] i częstotliwości 5 [Hz] została przedstawiona powyżej. W przebiegu amplitudowym zauważalne są zniekształcenia,

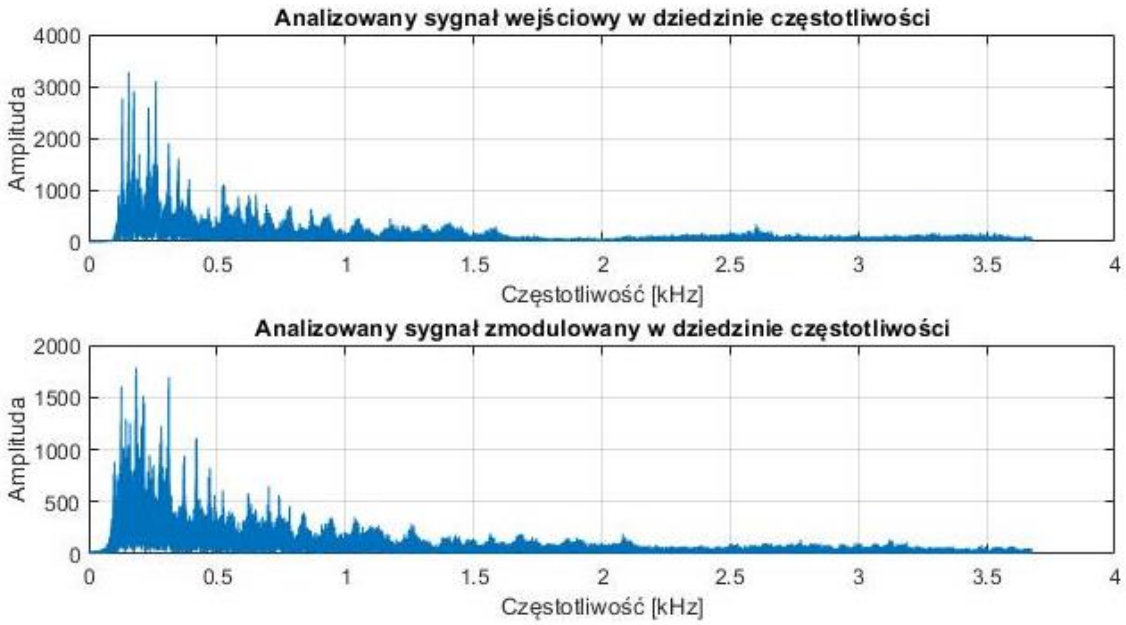
natomiast widmo ma zmniejszoną amplitudę oraz wysokie natężenie prążków w zakresach niskich częstotliwości.

3.8. Detune

Realizacja efektu *detune* dla środowiska MatLAB została przedstawiona w załączniku i oznaczona jako listing 8. Efekt *detune* jest bardzo podobny do efektu *pitch shiftera*, ponieważ realizuje on jednocześnie podwyższenie i obniżenie tonu. Funkcja realizująca efekt detune zgodnie ze schematem przedstawionym na rys. 2.17 pobiera od użytkownika parametry sygnału audio, wartość wzmocnienia, czas opóźnienia oraz częstotliwość sygnału dla którego ma być realizowany efekt detune. Pętla *for (linia 13)* realizuje zależności matematyczne kolejno (2.16) (*linia 14 – podwyższenie tonu*), (2.17) (*linia 15 – obniżenie tonu*). Wyrażenie (2.15) czyli wyjście *y (linia 16)* jest to suma podwyższenia oraz obniżenia tonu dla zadanego wzmocnienia.



Rys. 3.17 Przykładowy przebieg amplitudy dla funkcji *detune*

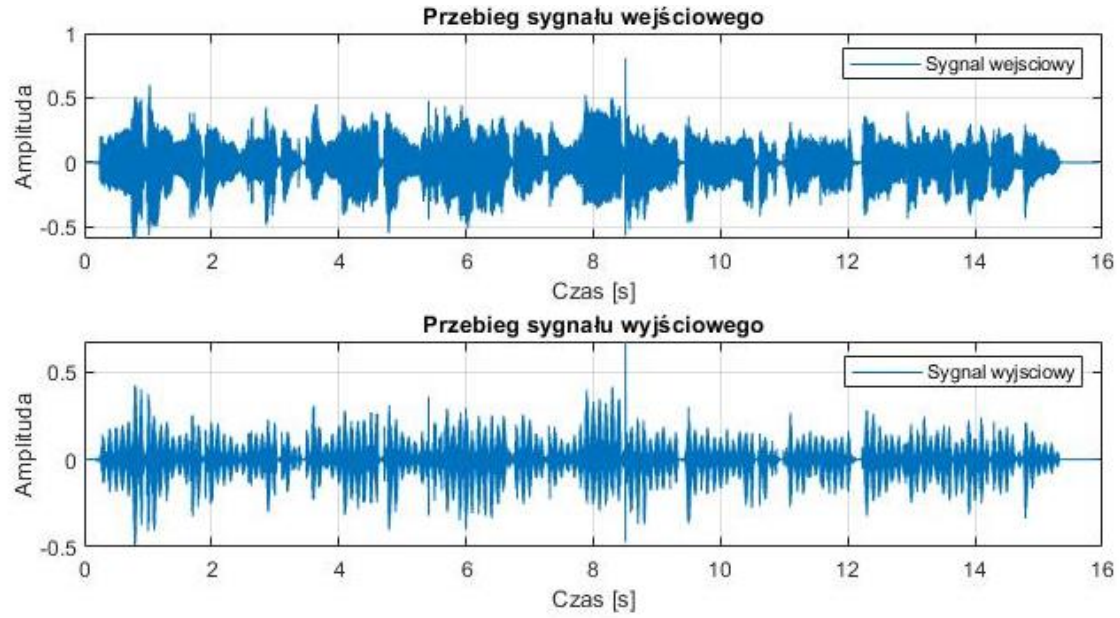


Rys. 3.18 Przykładowa realizacja częstotliwościowa dla funkcji *detune*

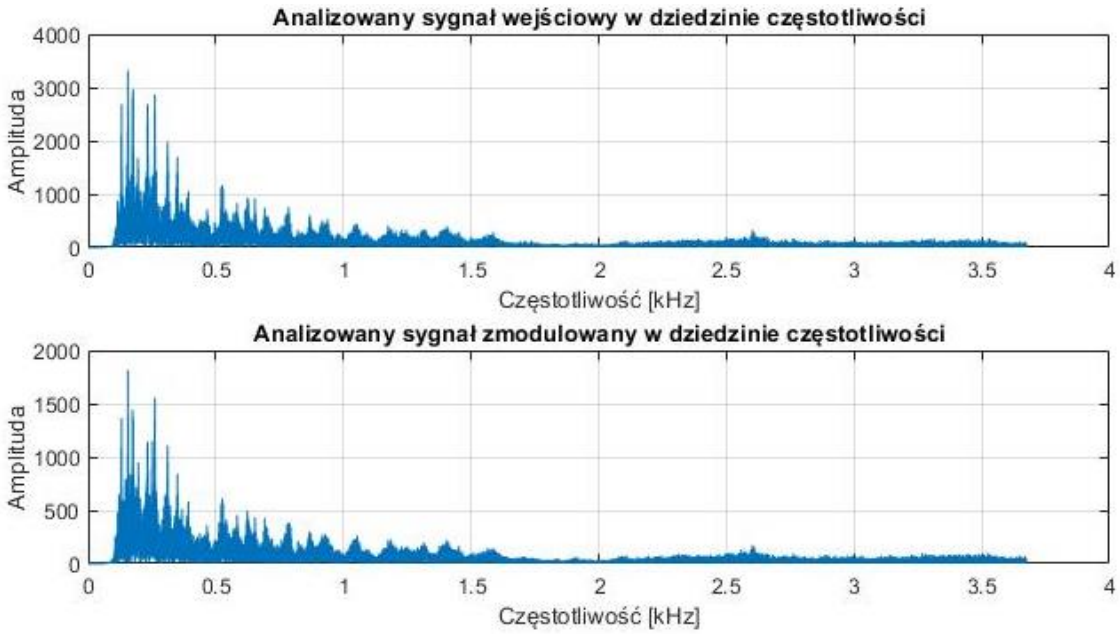
Przykład symulacji funkcji *detune* dla parametru wzmocnienia 0.5, częstotliwości 10 [Hz] oraz czasu opóźnienia 10 [ms] zobrazowującego zmianę amplitudy w czasie oraz widmo przedstawiono powyżej. Zmniejszenie amplitudy oraz zniekształcenia spowodowane są podaniem parametru o wartości 0.5 oraz zastosowaniem funkcji piłokształtnej. Widmo ma zmniejszoną amplitudę prążków oraz jest wzbogacone w paśmie niskich i środkowych częstotliwości.

3.9. Tremolo

Realizacja efektu tremolo dla środowiska MatLAB została przedstawiona w załączniku i oznaczona jako listing 9. Funkcja realizuje efekt tremolo zgodnie ze schematem blokowym przedstawionym na rys. 2.19. W pętli *for (linia 11)* obliczana jest wartość zmiennej *Tremolo_delay (linia 12)* zgodnie z zależnością (1.19), a następnie zgodnie z zależnością (2.18) obliczana jest wartość wyjściowa *y* przedstawiona w *linii 13*. W związku z modulowaniem amplitudy sugerowana jest wartość parametru *Amplituda* poniżej jedności. W celu uniknięcia efektu przebicia w trakcie zapisu do pliku wyjściowego zastosowano pomnożenie sygnału wyjściowego przez wartość 0.3 (*linia 16*).



Rys. 3.19 Przykładowy przebieg amplitudy w czasie dla efektu tremolo



Rys. 3.20 Przykładowy przebieg częstotliwościowy dla efektu temolo

Symulacja amplitudowa powyższej funkcji dla Amplitudy 0.7 oraz częstotliwości 10 [Hz] została przedstawiona powyżej. Zmodulowana amplituda wyjściowa ma charakterystyczny przebieg dla sygnałów przetworzonych przez efekt tremolo, natomiast widmo sygnału jest stłumione i niezauważalne są większe zmiany jeżeli chodzi o reprezentację częstotliwościową sygnału zmodulowanego.

3.10. Podsumowanie

Przedstawione powyżej rozwiązania są tylko jednym z wielu możliwych sposobów realizacji danego zagadnienia. Subiektywna metoda oceny jakości implementacji danego efektu będzie dla każdego badanego inna, a przedstawione wykresy zobrazowujące przebiegi amplitudowe oraz widmo pozwalają na obiektywną ocenę stopnia poprawności implementacji.

Niewątpliwie środowisko MatLAB bardzo ułatwia pracę i operację na sygnałach. Dedykowane dodatki oraz funkcje uprzednio zaimplantowane w środowisko pomagają w prosty i przejrzysty sposób dokonywać cyfrowej obróbki sygnałów. Dzięki szeroko rozwijanemu systemowi symulacji możliwe zostało przedstawienie w sposób graficzny zobrazowań przebiegów amplitudowych i częstotliwościowych. Wadami przetwarzania sygnałów w tym środowisku są częste aktualizacje składni co może być problematyczne podczas uruchamiania starszych aplikacji z tej dziedziny jednak podczas wykonywania pracy nie natrafiono odkryto wad tego środowiska.

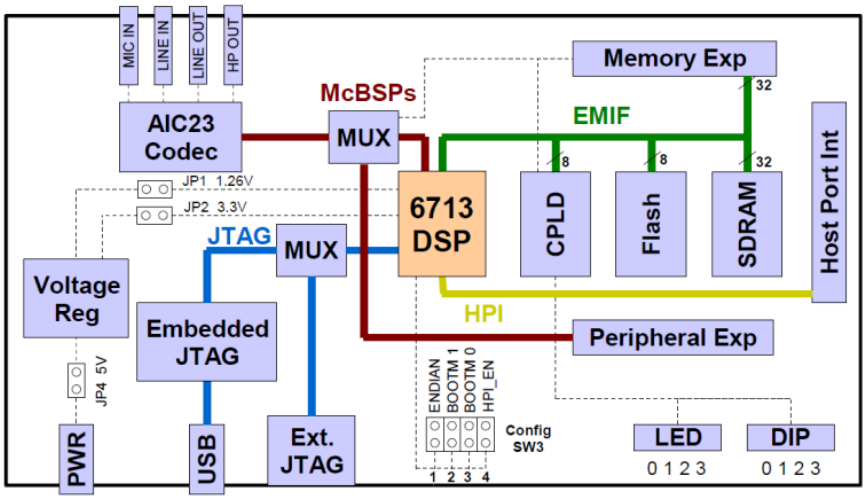
Praca w programie MatLAB nie powinna sprawiać trudności nawet dla początkujących programistów, gdyż składnia nie odbiega znacząco od języków C/C++. MatLAB oferuje również szereg dokumentacji oraz gotowych rozwiązań w celu ułatwienia pracy.

4. Implementacja efektów akustycznych w języku C dla procesora sygnałowego

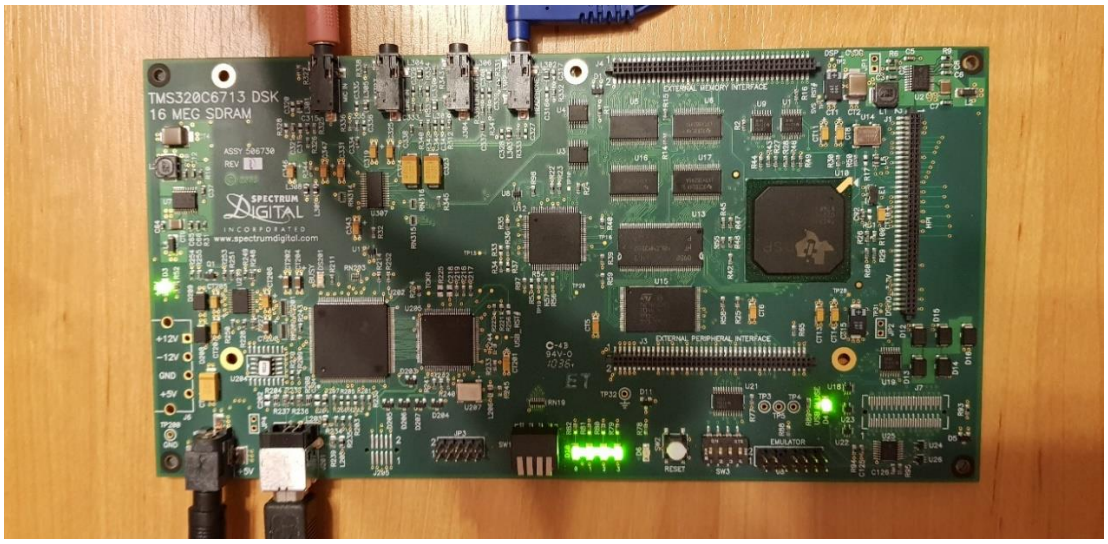
Procesory sygnałowe są to wyspecjalizowane układy elektroniczne, które wykonują złożone operacje na próbkach przy niskim zużyciu energii. Ciężko wyobrazić sobie przemysł muzyczny bez rozwinięcia układów procesorów sygnałowych, jednak są one powszechnie używane również w wielu urządzeniach związanych z telekomunikacją np. w telefonach, ale także w urządzeniach medycznych czy wojskowych.

4.1. Procesor sygnałowy TMS320C6713 i płyta uruchomieniowa DSK6713

Układ procesora sygnałowego TMS320C6713 to zmiennoprzecinkowy procesor sygnałowy wykorzystujący technologię VLIW (*ang. Very Long Instruction Word*) , która została rozwinięta przez amerykańską firmę Texas Instruments. Procesor jest przystosowany do pracy w systemach wielokanałowych i wielofunkcyjnych. Jego wydajność sięga rzędu 1350 milionów operacji zmiennoprzecinkowych na sekundę, natomiast częstotliwość zegara osiąga 225 MHz. Procesor posiada 32 rejestry od długości 32 bitów wraz z ośmioma niezależnymi jednostkami funkcjonalnymi. Układ zawiera również pamięć wbudowaną . W ramach układu dostępne jest 16 MB synchronicznej pamięci DRAM, 512 KB pamięci nieulotnej Flash, 4 mikroprzełączniki DIP oraz 4 diody LED, programowalny układ CPLD (*ang. Complex Logic Device*) do konfiguracji karty za pomocą zapisów w rejestrach.



Rys. 4.1 Schemat procesora sygnałowego TMS320C6713



Rys. 4.2 Sposób podłączenia procesora TMS320C6713

Przykład podłączenia procesora został przedstawiony powyżej. Do układu podłączone jest zasilanie (+5V), przewód USB-B łączący procesor z komputerem, wejście mikrofonowe (czerwony przewód) oraz wyjście słuchawkowe (niebieski przewód).

4.2. Środowisko programistyczne oraz konfiguracja

Implementacja efektów akustycznych została wykonana w środowisku programistycznym *Code Composer Studio* w wersji 8.3.1 oraz wersji kompilatora 7.4.24. Płyta jest obsługiwana przez wbudowany programator obsługiwany przez oprogramowanie *Spectrum Digital DSK-EVM-eZdsp*. Jest to istotne z punktu odtworzenia poniżej zaimplementowanych efektów, gdyż wyższa wersja programu oraz kompilatora nie obsługuje układu C6713 i uruchomienie wykonanych programów w ramach pracy dyplomowej będzie niemożliwe. W celu uzyskania efektów należy również odpowiednio skonfigurować środowisko poprzez załączenie odpowiednich plików oraz bibliotek, a także wybór określonych właściwości projektu, które zostały omówione poniżej.

1) Należy przejść do *Build* > *C6000 Compiler* > *Advanced Option* > *Predefined Symbols* w celu dodania definicji symbolu dla preprocesora. Wartość ta może być już ustawiona automatycznie jednak nie należy pomijać tego kroku w celu uniknięcia problemów przy kompilacji programu.
2) W oknie *Pre-define NAME* należy dodać wartość *CHIP_6713*, który dedykowany jest dla procesora sygnałowego TMS320C6713.

3) Kolejnym krokiem jest ustawienie parametru *Data access model* na wartość *far*. W tym celu należy przejść do *Build > C6000 Compiler > Advanced Option > Runtime Model Options.*

4) Kolejnym krokiem jest dodanie bibliotek do katalogu *C6000 Linker* w pod zakładce *File Search Path*. Wymagane biblioteki to : *csl6713e.lib* , *csl6713.lib* oraz *dsk6713bsl.lib*. W tym celu należy przejść do *Build > C6000 Linker > File Search Path.* Poprzez symbol dodania należy wskazać ścieżki do powyższych bibliotek w tabeli *Include library file*.

5) Ostatnią istotną rzeczą w konfiguracji projektu jest załączenie pliku inicjalizującego. W folderze *targetConfigs* wybieramy plik *TMS320C6713.ccml* i przechodzimy do jego właściwości zaawansowanych (*lewy dolny róg okna TMS320C6713.ccml)*. W sekcji *CPU Properties* należy dodać skrypt inicjalizujący o nazwie *DSK6713.gel* .

W celu odtworzenia poniżej przedstawionych efektów konieczne jest również dodanie dwóch plików dedykowanych dla procesora sygnałowego TMS320C6713. Są to *c6713dskinit.c* oraz *c6713dskinit.h*. Możliwie jest bezpośrednie dodanie biblioteki *c6713dskinit.h* do pliku poprzez dyrektywę *include* jednak mogą pojawić się problemy z identyfikacją lokacji pliku *c6713dskinit.c* dlatego zalecane jest dodanie go do folderu z projektem.

4.3. Realizacja efektów akustycznych w procesorze sygnałowym TMS320C6713

Realizacja efektów dla procesora sygnałowego została oparta o implementacje postaci funkcyjnej o stałej konfiguracji. Została przygotowana realizacja sumowania kanałów, opóźnienia, pogłosu, modulacji sygnałem sinusoidalnym (*flanger*), modulacji oscylatorem niskich częstotliwości (*chorus*) oraz modulacji amplitudy (*tremolo*). W celu odtworzenia danego efektu został przygotowany plik źródłowy *Multi_efekt.c*, który dla odpowiedniej konfiguracji przełączników realizuje przygotowany efekt.

4.3.1. *Bypass*

Realizacja przekazania sygnału z wejścia do wyjścia dla procesora sygnałowego została przedstawiona w załączniku i oznaczona jako listing 10. Implementacja *ByPass* realizuje bezpośrednie przesłanie sygnału z wejścia do wyjścia. Do zmiennej *wejście* zostaje zapisany sygnał wejściowy, który może być podany np. poprzez mikrofon lub

może być to sygnał wygenerowany z oscyloskopu. Instrukcja *output_sample* powoduje przesłanie próbek na wyjście procesora czego efektem w przypadku podłączenia słuchawek może być sygnał z mikrofonu.

4.3.2. *Sumowanie kanałów*

Realizacja sumowania kanałów dla procesora sygnałowego TMS320C6713 została oznaczona jako listing 11. Implementacja sumowania kanałów zrealizowana jest w oparciu o schemat blokowy sumatora przedstawiony na rys. 2.1. Pobrana zostaje wartość próbki z lewego oraz prawego kanału i dla zmiennej wyjście realizuje równanie (2.2). Zmienna *wyjście* przekazana zostaje na wyjście układu instrukcją *output_sample*.

4.3.3. *Reverb*

Realizacja efektu pogłosu dla procesora sygnałowego TMS320C6713 została oznaczona jako listing 12. Implementacja realizuje efekt pogłosu zgodnie z zależnością matematyczną (2.7) zgodnie ze schematem z rys. 2.7. Sygnał wejściowy zostaje przekazany do bufora, który przechowuje sygnał opóźniony. Zmienna *wyjście* zawiera w sobie sumę sygnału wejściowego oraz sygnału opóźnionego. Po przekazaniu zmiennej wyjście do instrukcji odpowiedzialnej za przekazanie sygnału do wyjścia do bufora przypisywany jest sygnał wyjściowy zgodnie z równaniem (2.7).

4.3.4. *Delay*

Realizacja efektu opóźnienia dla procesora sygnałowego TMS320C6713 została przedstawiona w załączniku i oznaczona jako listing 13. Rozwiązanie opóźniające sygnał wejściowy dla procesora sygnałowego TMS320C6713 została zrealizowana w oparciu o równanie (2.8). Sygnał wejściowy zostaje pobrany z wejścia i zostaje przekazany do bufora opóźniającego. Instrukcja *suma_delay* sumuje sygnał wejściowy oraz sygnał opóźniony, który jest pomnożony o wartość wzmocnienia. Sygnał przechowywany w zmiennej *suma_delay* przekazany jest na wyjście układu.

4.3.5. *Flanger*

Realizacja modulacji sygnału przy użyciu sygnału sinusoidalnego dla procesora sygnałowego TMS320C6713 została przedstawiona w załączniku i oznaczona jako listing 14. Efekt został zrealizowany na podstawie schematu blokowego z rys. 2.10. Zmienna *opóźnienie* realizuje zależność matematyczną (2.10), która odpowiada za modulację sygnału. Zmienna *efekt_flanger* odpowiada wykonuje zależność

matematyczną (2.9). Sygnał powstały w wyniku operacji na zmiennej *efekt_flanger* zawierający modulację sygnału zostaje przekazany do wyjścia układu.

4.3.6. *Chorus*

Realizacja modulacji sygnału przy użyciu LFO została oznaczona jako listing 15. Implementacja *Chorus* realizuje modulację z użyciem LFO korzystając z zależności matematycznych (2.11) oraz (2.12). Sygnał wejściowy podobnie jak w przypadku modulacji sygnałem sinusoidalnym zostaje poddany modulacji realizowanej przez zmienną *opóźnienie*, która dla efektu chorus wykonuje zależność (2.12). Zmienna *efekt_chorus* dokonuje sumowania sygnału wejściowego oraz sygnału zmodulowanego zgodnie z zależnością (2.11). Sygnał przechowywany w zmiennej *efekt_chorus* zostaje przekazany do wyjścia układu.

4.3.7. *Tremolo*

Realizacja modulacji amplitudy sygnału dla procesora sygnałowego TMS320C6713 została przedstawiona w załączniku i oznaczona jako listing 16. Implementacja *Tremolo* realizuje modulację amplitudy sygnału zgodnie ze schematem blokowym przedstawionym na rys. 2.19. Sygnał wejściowy zostaje poddany modulacji zgodnie z zależnością (2.19). Zmienna *efekt_tremolo* zawierająca sygnał ze zmodulowaną amplitudą zgodnie z wyrażeniem (2.18) zostaje przekazana do wyjścia układu.

4.4. Podsumowanie

Zdecydowanie praca z procesorem sygnałowym wymaga o wiele więcej pracy i zaangażowania w przeciwieństwie do środowiska MatLAB. Mimo, iż procesory sygnałowe są kluczowymi układami branży elektronicznej trzeba przyznać, iż procesor C6713 jest dość starym procesorem i jego ograniczenia są zauważalne. Układ zadebiutował na rynku w 2003 roku. Samo wsparcie w postaci pomocy naukowych nie jest również tak znaczące jak w przypadku programu MatLAB. Nadal jednak procesor TMS320C6713 jest godnym polecenia układem w przypadku decyzji o rozpoczęciu nauki w dziedziny cyfrowego przetwarzania sygnałów.

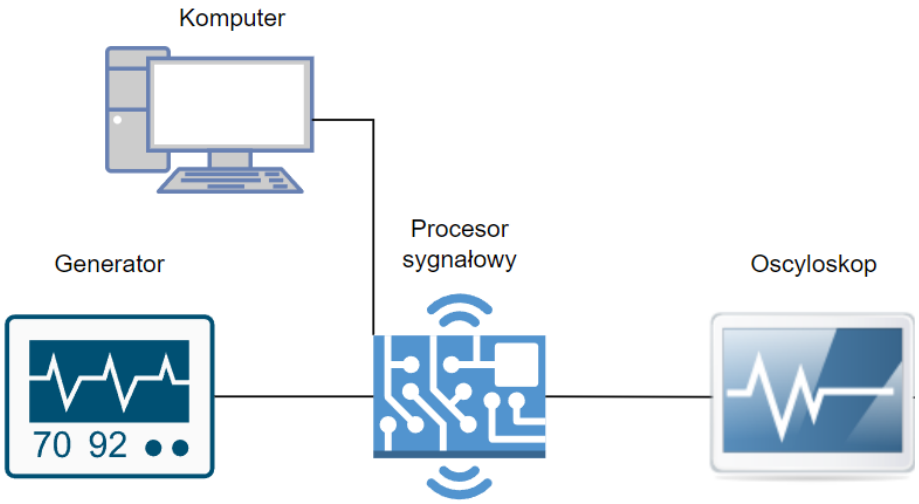
5. Weryfikacja eksperymentalna zaimplementowanych efektów akustycznych

Na potrzeby weryfikacji poprawności zaimplementowanych efektów została wykonana weryfikacja eksperymentalna. Badania oscyloskopowe pozwolą ocenić stopień poprawności implementacji efektów dla procesora sygnałowego TMS320C6713. Poniżej zostało przedstawiony opis stanowiska laboratoryjnego, a w następnym podpunkcie zobrazowania z oscyloskopu dla każdego z zaimplementowanych efektów.

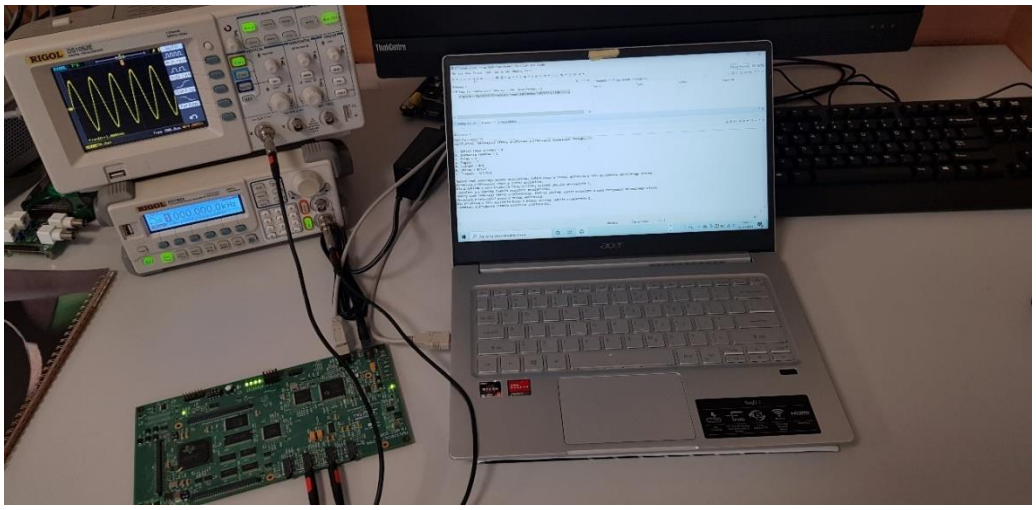
5.1. Stanowisko pomiarowe

W celu weryfikacji poprawności zaimplementowanych efektów została przeprowadzona weryfikacja oscyloskopowa sygnału wejściowego podanego z generatora. Obserwacje zostały wykonane na stanowisku pomiarowym w skład którego wchodziły:

- oscyloskop cyfrowy RIGOL DS1052E
- generator funkcyjny RIGOL DG1022



Rys. 5.1 Schemat układu pomiarowego

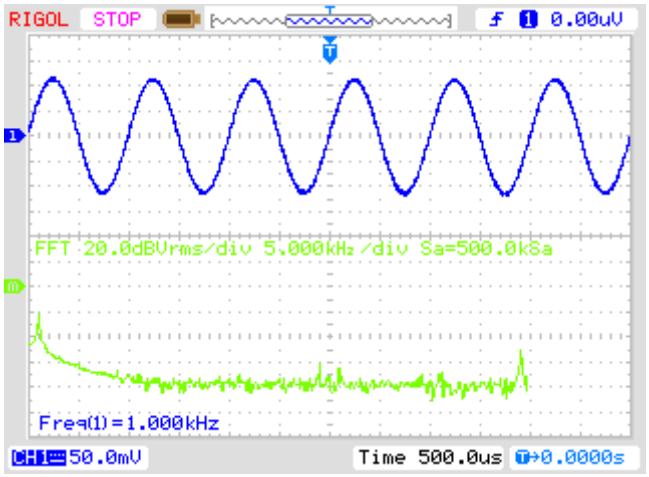


Rys. 5.2 Stanowisko pomiarowe

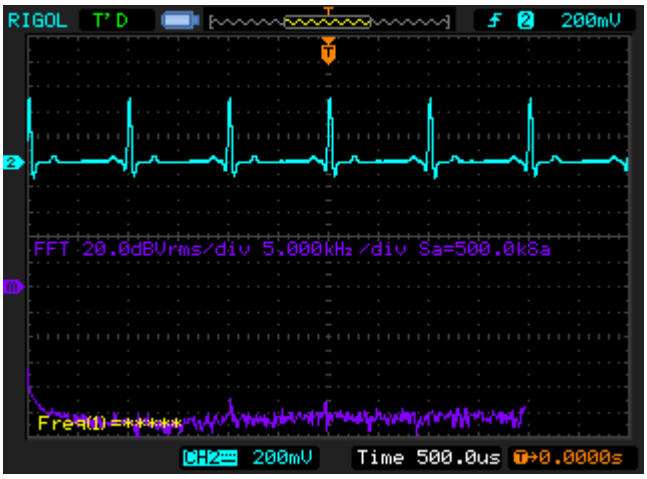
5.2. Symulacja efektów zrealizowanych dla procesora sygnałowego

Weryfikacja eksperymentalna z użyciem stanowiska pomiarowego przedstawionego powyżej polegała na przekazaniu sygnału z generatora arbitralnego na wejście układu procesora z zaimplementowanym efektem, a następnie obserwację sygnału wyjściowego na oscyloskopie. Weryfikacji zostały poddane wszystkie zaimplementowane efekty wykorzystując do tego sygnał sinusoidalny oraz sygnał arbitralny fragmentu sygnału elektrokardiograficznego (precyzyjnie zespół QRS).

Zrzut ekranu oscyloskopu przedstawiony na rys. 5.3 przedstawia sygnał sinusoidalny o częstotliwości 1kHz, który został wygenerowany w generatorze. Prezentuje on poprawność implementacji bypass oraz poprawność połączeń na stanowisku. W celu zaprezentowania efektów opóźnienia oraz modulacji został wykorzystany sygnał arbitralny, którego symulacja dla implementacji bypass została przedstawiona na rys. 5.4.

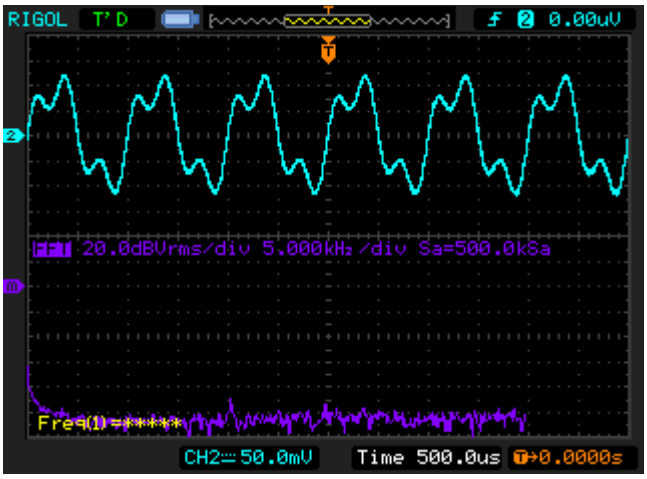


Rys. 5.3 Symulacja Bypass dla sygnału sinusoidalnego



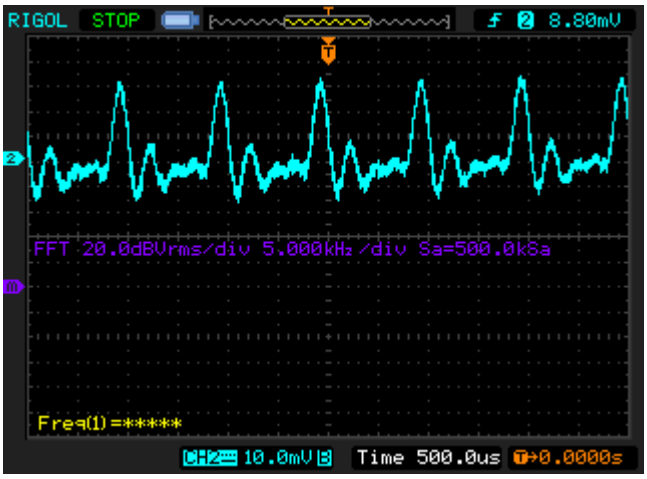
Rys. 5.4 Symulacja Bypass dla sygnału arbitralnego

Symulacja sumowania kanałów dla procesora sygnałowego została przedstawiona na rys. 5.5. Do przeprowadzenia symulacji sumowania kanałów został użyty sygnał sinusoidalny. Korzystając z zależności (2.2), próbki z kanału lewego i prawego zostają do siebie dodane, natomiast podzielenie przez dwa powoduje, że tylko połowa amplitudy zostaje przekazana na wyjście.



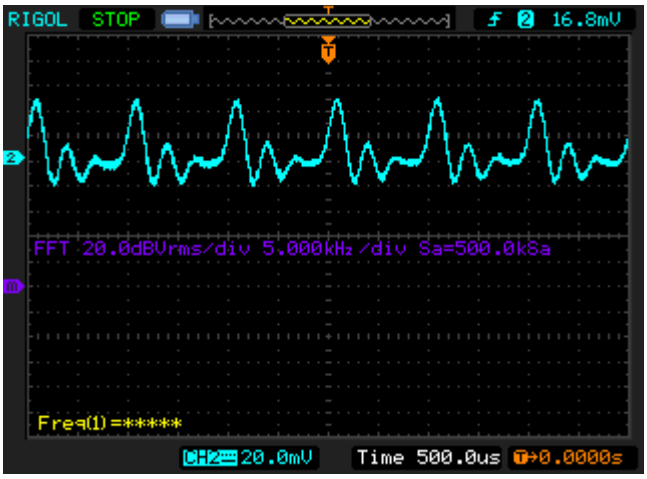
Rys. 5.5 Symulacja sumowania kanałów dla procesora sygnałowego

Symulacja efektu pogłosu dla procesora sygnałowego przedstawia rys. 5.6. W celu symulacji efektu pogłosu na wejście został podany sygnał arbitralny. Korzystając z zależności (2.7) otrzymujemy po charakterystycznym impulsie podanym z sygnału podstawowego sygnał opóźniony w czasie, o zmniejszonej amplitudzie w stosunku do impulsu bazowego będący składową sygnału wejściowego oraz sygnału wyjściowego jak pokazuje powyższe zobrazowanie



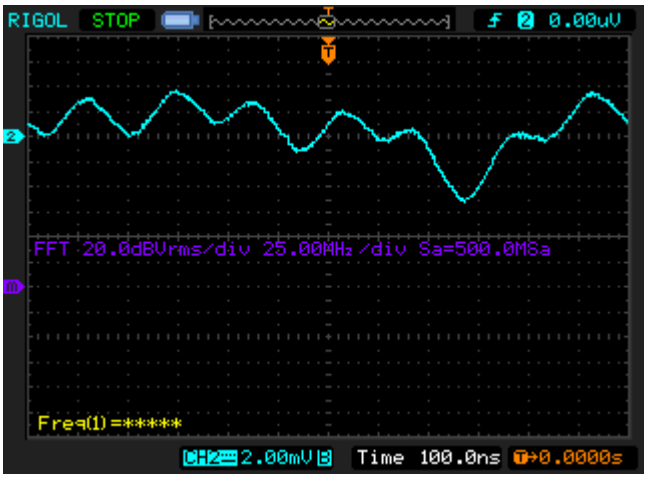
Rys. 5.6 Symulacja efektu pogłosu dla procesora sygnałowego

Symulacja efektu opóźnienia dla procesora sygnałowego TMS320C6713 została przedstawiona na rys. 5.7. W celu symulacji efektu opóźnienia na wejście został podany sygnał arbitralny, który zgodnie z zależnością matematyczną realizującą efekt opóźnienia dla sygnału wejściowego powoduje opóźnienie. Po impulsie wejściowym zauważalny jest kolejny impuls o zmniejszonej amplitudzie.



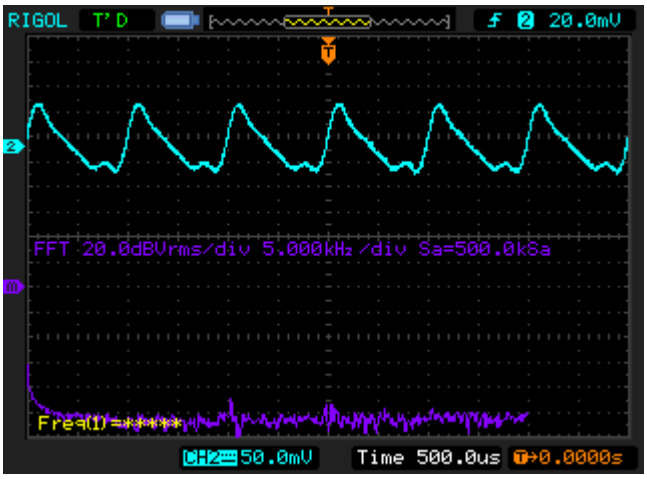
Rys. 5.7 Symulacja efektu opóźnienia dla procesora sygnałowego

Weryfikacja implementacji efektu *flanger* została przedstawiona na rys. 5.8. Podając na wejściu sygnał arbitralny oraz korzystając z zależności matematycznej (2.9) oraz (1.10) otrzymujemy sygnał poddany modulacji sygnałem sinusoidalnym. Sygnał został całkowicie zmodulowany i nie przypomina on sygnału wejściowego.



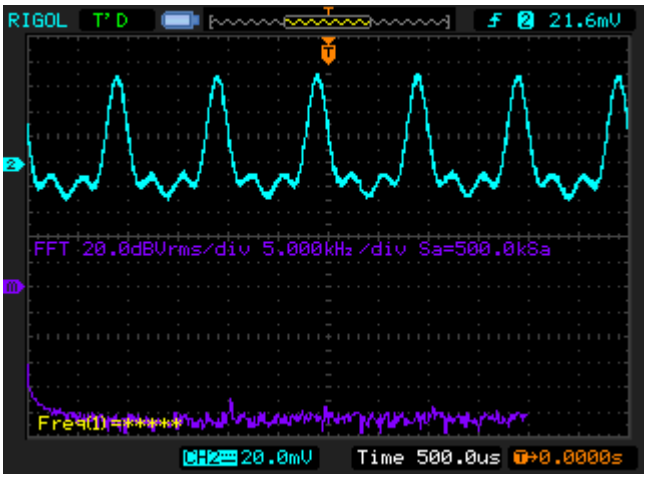
Rys. 5.8 Symulacja efektu *flanger dla procesora sygnałowego*

Zobrazowanie przedstawiające implementację efektu *chorus* dla procesora sygnałowego przedstawiono poniżej. W celu symulacji implementacji efektu *chorus* na wejście został podany sygnał arbitralny, na którym zostały zastosowane zależności (2.11) oraz (2.12). Efektem jest sygnał przypominający sygnał piłokształtny.



Rys. 5.9 Symulacja efektu *chorus dla procesora sygnałowego*

Symulację implementacji efektu tremolo dokonano poprzez podanie na wejście sygnału arbitralnego. Wykorzystane zależności w celu implementacji efektu Tremolo pozwoliły na uzyskanie zobrazowania z oscyloskopu, które zostało przedstawione na rys. 5.10. Zauważalne są charakterystyczne dla tego efektu skoki amplitudowe pomiędzy kolejnymi impulsami wejściowymi.



Rys. 5.10 Symulacja efektu Tremolo dla procesora sygnałowego

5.3. Podsumowanie

Przedstawione powyżej zobrazowania ukazują pewien stopień poprawności implementacji. Należy pamiętać, iż przedstawiony sposób implementacji jest jednym z wielu dostępnych rozwiązań, tak samo jak możliwe było wykonanie innego rodzaju procesu weryfikacji. Powyższa weryfikacja ukazuje ocenę implementacji w sposób obiektywny, natomiast subiektywnie należy dokonać oceny odsłuchowej, dla którego każdy będzie miał indywidulaną skalę oceny dla danego efektu.

6. Wnioski

W ramach pracy inżynierskiej został przygotowany przegląd popularnych efektów akustycznych prezentujących przykładowy układ realizujący dany efekt akustyczny, zależność matematyczną oraz schemat blokowy realizujący tę zależność. Opracowana została implementacja efektów akustycznych dla środowiska MatLAB z użyciem wyrażeń matematycznych oraz została przeprowadzona symulacja amplitudowa i częstotliwościowa dla każdego zaimplementowanego efektu. Wykonano również implementację efektów dla procesora sygnałowego TMS320C6713 w języku C, który realizuje implementację efektów w czasie rzeczywistym oraz wykonano testy wizualizujące stopień poprawności implementacji efektów.

Ocena stopnia poprawności implementacji danego efektu może odbyć się poprzez odsłuch sygnału wejściowego bez implementacji efektu oraz sygnału wyjściowego po obróbce cyfrowej jednak będzie to ocena subiektywna inna dla każdego badanego. Dla efektów wykonanych w środowisku MatLAB zostało wykonane porównanie amplitudowe oraz widmowe sygnałów wejściowych oraz wyjściowych dla każdego efektu oraz dla implementacji w środowisku CCS porównanie sygnałów przed oraz po implementacji. Symulacje pozwalają na porównanie zmiany sygnału oraz na ocenę poprawności stopnia implementacji danego efektu.

Niewątpliwie sposób implementacji rozważanych efektów nie jest idealny. Istnieje wiele opracowań oraz wiele zależności, niekiedy bardzo rozbudowanych, które pomagają osiągnąć podobny bądź lepszy efekt implementacji efektu, jednak na potrzeby pracy inżynierskiej ograniczono się do podstawowych zależności celem przybliżenia tematu efektów akustycznych oraz lepszego zrozumienia przedstawianych zagadnień.

Implementacji efektów akustycznych towarzyszą również pewne problemy, ponieważ ogół pracy wymagał wiedzy z zakresu programowania, elektroniki, akustyki oraz cyfrowego przetwarzania sygnałów. Zdobyte kompetencje dzięki powyższej pracy pozwoliły utrwalić wiedzę oraz rozwinąć kompetencje z zakresu programowania w środowisku MatLAB, bądź przybliżyć dziedzinę jaką jest cyfrowe przetwarzanie sygnałów.

7. Bibliografia

[1] M. McDonough, „5 Types of Reverb Explained: Hall, Chamber, Room, Plate, and Spring," 29 01 2020. [Online]. Available: https://www.sweetwater.com/insync/5-types-of-reverb-explained-hall-chamber-room-plate-and-spring/. [Data uzyskania dostępu: 2021 12 12].

[2] R. Eliott, „Reverb Drive And Recovery Amplifier," 01 12 2020. [Online]. Available: https://sound-au.com/project211.htm. [Data uzyskania dostępu: 2021 08 20].

[3] Gear4music, 2003. [Online]. Available: https://www.gear4msuic.pl. [Data uzyskania dostępu: 10 12 2021].

[4] J. Stolet, „Low-frequency Oscillators," [Online]. Available: https://pages.uoregon.edu/emi/31.php. [Data uzyskania dostępu: 2022 09 09].

[5] „https://en.wikipedia.org/wiki/Pitch_(music)," [Online]. [Data uzyskania dostępu: 2022 01 09].

[6] J. T. Dan Ledger, „Using The Low Cost, High Performance ADSP-21065L Digital Signal Processor For Digital Audio Applications," [Online]. Available: https://www.analog.com/media/en/technical-documentation/application-notes/21065l_audio_tutorial.pdf. [Data uzyskania dostępu: 2021 10 10].

[7] Muziker, „Pitch Box," [Online]. Available: https://www.muziker.pl/mooer-pitch-box. [Data uzyskania dostępu: 2021 11 07].

[8] Musicstore, „TC Electronic Choka Tremolo," [Online]. Available: https://www.musicstore.com/pl_PL/PLN/TC-Electronic-Choka-Tremolo. [Data uzyskania dostępu: 2021 12 01].

[9] S. Osowski, Cyfrowe przetwarzanie sygnałów z zastosowaniem MATLABA, Warszawa: Oficyna Wydawnictwa Politechniki Warszawskiej, 2016.

[10] S. W.Smith, Cyfrowe przetwarzanie sygnałów, BTC, 2003.

[11] P. Gdańska, „Wydział elektroniki, telekomunikacji i informatyki," [Online]. Available: https://eti.pg.edu.pl/documents/176593/26766624/Wyklad_KC_PG_2.pdf. [Data uzyskania dostępu: 2022 01 15].

[12] S. D. Inc., „TMS320C6713 DSK. Technical Reference," 2004.

[13] D. R. Rulph Chassaing, Digital Signal Processing and Applications with the TMS320C6713 and TMS320C6416 DSK, John Wiley & Sons, Inc., 2008.

8. Załącznik

```
%funkcja dla środowiska MatLAB realizująca sumowanie kanałów
function Stereo (sciezka,nazwa,rozszerzenie)
    ProbkaSygnalu = strcat(sciezka,'\',nazwa, rozszerzenie);
    [x,Fs] = audioread(ProbkaSygnalu);
    [m,kanal]=size(x);

    if kanal == 1
        y=x;
    elseif kanal == 2
        y=sum(x,2);

        Maxi_Amp=max(abs(y));

        y=y/Maxi_Amp;

        Maxi_Amp_Left=max(abs(x(:,1)));
        Maxi_Amp_Right=max(abs(x(:,2)));

        Maxi_Amp=max([Maxi_Amp_Left Maxi_Amp_Right]);

        y=y*Maxi_Amp;
    else
        disp('Blad wyboru');
        return;
end
audiowrite("Zapisane_pliki_koncowe\Stereo_efekt_koncowy.wav",y,Fs);

    syg_wej=audioplayer(x,Fs);
    playblocking(syg_wej);
    sound(y,Fs);
end
```

Listing 1. Realizacja sumowania kanałów dla środowiska MatLAB

```
%Realizacja symulacji dla srodowiska MatLAB
t = (0:length(y)-1)/Fs;

subplot(3,1,1)

      plot(t,x)
      legend('Sygnal wejsciowy')
      title("Przebieg sygnału wejściowego")
      xlabel("Czas [s]")
      ylabel("Amplituda")
      grid on

subplot(3,1,2)

      plot(t,y)
         legend('Sygnal wyjsciowy')
      title("Przebieg sygnału wyjściowego")
      xlabel("Czas [s]")
      ylabel("Amplituda")
      grid on


figure(2)
```

```
subplot(4, 1, 1)

      FFT_sig = fft(x);
         Y = abs(FFT_sig(1:round(length(x) / 2 + 1)));
      freq = (0:length(FFT_sig) - 1) * Fs / length(FFT_sig);
      l = floor(length(Y)/6);
      plot(freq(1:l) / 1000, Y(1:l))

            title('Analizowany sygnał wejściowy w dziedzinie
            częstotliwości')
      xlabel('Częstotliwość [kHz]')
      ylabel('Amplituda')
      grid on

subplot(4,1,2)

      FFT_sig2=fft(y);
      Z = abs(FFT_sig2(1:round(length(y)/2+1)));
      u = floor(length(Z)/6);
      plot(freq(1:u) / 1000, Z(1:u))

            title('Analizowany sygnał zmodulowany w dziedzinie
            częstotliwości')
      xlabel('Częstotliwość [kHz]')
      ylabel('Amplituda')
      grid on
```

Listing 2. Realizacja symulacji dla środowiska MatLAB

```
%Realizacja efektu poglosu dla srodowiska MatLAB
function Reverb (sciezka , nazwa , rozszerzenie , G , Opoznienie)

    ProbkaSygnalu = strcat(sciezka,'\',nazwa, rozszerzenie);
    [x, Fs] =audioread(num2str(ProbkaSygnalu));

    LEN = length(x);
    y = zeros(LEN,1);

    D =round( 0.001 * Opoznienie * Fs );

for n=D+1:LEN

            y(n)= x(n) + G * y(n-D);
end

    Reverb_zapis_pliku = y * 0.7;

audiowrite("Zapisane_pliki_koncowe\Poglos_efekt_koncowy.wav",Reverb_za
pis_pliku,Fs);

    syg_wej=audioplayer(x,Fs);
    playblocking(syg_wej);
    sound(y,Fs);

end
```

Listing 3. Realizacja efektu pogłosu dla środowiska MatLAB

```
%funkcja realizująca efekt opóźnienia dla środowiska MatLAB
function Delay (sciezka , nazwa , rozszerzenie , G , Opoznienie)
    ProbkaSygnalu = strcat(sciezka,'\',nazwa, rozszerzenie);
    [x, Fs] =audioread(num2str(ProbkaSygnalu));

    D =round( 0.001 * Opoznienie * Fs );

    bufor_wejsciowy = [x;zeros(D,1)];
    bufor_wyjsciowy = [zeros(D,1);x]*G;

    y = bufor_wejsciowy+bufor_wyjsciowy;
    audiowrite("Zapisane_pliki_koncowe\Delay_efekt_koncowy.wav",y,Fs);

    syg_wej=audioplayer(x,Fs);
    playblocking(syg_wej);
    sound(y,Fs); %
end
```

Listing 4. Realizacja efektu opóźnienia dla środowiska MatLAB

```
 %funkcja realizująca efekt modulacji flanger dla środowiska MatLAB
function flanger ( sciezka , nazwa , rozszerzenie , G , Opoznienie     ,
Czestotliwosc_modulacji )
      ProbkaSygnalu = strcat(sciezka,'\',nazwa, rozszerzenie);
      [x, fs] =audioread(num2str(ProbkaSygnalu));

      LEN = length(x);
      D =round( 0.001 * Opoznienie * fs );
      y = zeros(LEN,1);
      fcycle = Czestotliwosc_modulacji/fs;

      for n=D+1:LEN

   opoznienie_flanger=round((D/2)*(1-cos(2*pi*n*fcycle)));
      y(n) = 0.5 * x(n) + G * x(n-opoznienie_flanger);

      end

audiowrite("Zapisane_pliki_koncowe\Flanger_efekt_koncowy.wav",y,fs);

      syg_wej=audioplayer(x,fs);
       playblocking(syg_wej);
       sound(y,fs);
   end
```

Listing 5. Realizacja efektu flanger dla środowiska MatLAB

```
%funkcja realizująca efekt chorus dla środowiska MatLAB
function chorus (sciezka , nazwa,rozszerzenie , Czas_opoznienia
,Czestotliwosc_modulacji)
      ProbkaSygnalu = strcat(sciezka,'\',nazwa, rozszerzenie);
      [x, Fs] =audioread(num2str(ProbkaSygnalu));

      LEN = length(x);
      y = zeros(LEN,1);
      fdelay = Czestotliwosc_modulacji /Fs;

      for n=Czas_opoznienia+1:LEN

Chorus_delay=round(Czas_opoznienia*(0.5+(2*pi*n*fdelay)));
      y(n) = 0.5 *( x(n) + x(n-Chorus_delay));

      end
   audiowrite("Zapisane_pliki_koncowe\Chorus_efekt_koncowy.wav",y,Fs);

      syg_wej=audioplayer(x,Fs);
    playblocking(syg_wej);
    sound(y,Fs);
end
```

Listing 6. Realizacja efektu chorus dla środowiska MatLAB

```
%funkcja realizująca efekt Pitch Shifter dla środowiska MatLAB
function Pitch_Shifter (sciezka , nazwa , rozszerzenie , G ,
Czas_opoznienia , Czestotliwosc_sygnalu , up_or_down )

      ProbkaSygnalu = strcat(sciezka,'\',nazwa, rozszerzenie);
      [x, Fs] =audioread(num2str(ProbkaSygnalu));

      LEN = length(x);
      D = round( 0.001 * Czas_opoznienia * Fs ) ;
      y = zeros(LEN,1);
      fshift = Czestotliwosc_sygnalu / Fs;

      if up_or_down == ("up")
            for n=D+1:LEN
            Pitch_delay= round(D*(-sawtooth(2*pi*n*fshift)));
            y(n) = G *(x(n-Pitch_delay));
            end

      elseif up_or_down == ("down")
            for n=D+1:LEN
            Pitch_delay = round(D*(sawtooth(2*pi*n*fshift)));
            y(n) = G *(x(n-Pitch_delay));
            end
      else
            disp('Blad wyboru');
            return;
      end
audiowrite("Zapisane_pliki_koncowe\Pitch_Shifter_efekt_koncowy.wav",y,
Fs);
      syg_wej=audioplayer(x,Fs);
    playblocking(syg_wej);
    sound(y,Fs);
end
```

Listing 7. Realizacja efektu pitch shifter dla środowiska MatLAB

```
%funkcja realizujące efekt detune dla środowiska MatLAB
function Detune (sciezka , nazwa , rozszerzenie , G ,
Czestotliwosc_sygnalu, Czas_opoznienia)
      ProbkaSygnalu=strcat(sciezka,'\',nazwa, rozszerzenie);
      [x, Fs] =audioread(num2str(ProbkaSygnalu));

      LEN = length(x);
      y = zeros(LEN,1);
      D = round( 0.001 * Czas_opoznienia * Fs ) ;

      fshift_down = Czestotliwosc_sygnalu /Fs;
      fshift_up = Czestotliwosc_sygnalu /Fs;

      for n=D+1:LEN
      Delay_shift_down=round(D*(-sawtooth(2*pi*n*fshift_up)));
            Delay_shift_up=round(D*(sawtooth(2*pi*n*fshift_down)));
      y(n)=G*((x(n-Delay_shift_down))+(x(n-Delay_shift_up)));
      end

audiowrite("Zapisane_pliki_koncowe\Detune_efekt_koncowy.wav",y,Fs);
      syg_wej=audioplayer(x,Fs);
    playblocking(syg_wej);
    sound(y,Fs);
end
```

Listing 8. Realizacja efektu detune dla środowiska MatLAB

```
%funkcja realizująca efekt tremolo dla środowiska MatLAB
function Tremolo (sciezka , nazwa , rozszerzenie , Amplituda ,
Czestotliwosc_modulacji)

      ProbkaSygnalu = strcat(sciezka,'\',nazwa, rozszerzenie);
      [x, Fs] =audioread(num2str(ProbkaSygnalu));

      LEN = length(x);
      y = zeros(LEN,1);
      fmod = Czestotliwosc_modulacji/Fs;

      for n=1:LEN
            Tremolo_delay = (cos(2*pi*fmod*n));
            y(n) = 0.5 * (x(n)) * (1 + Amplituda * Tremolo_delay);
      end

      tremolo_zapis_pliku = y * 0.3;

audiowrite("Zapisane_pliki_koncowe\Tremolo_efekt_koncowy.wav",
tremolo_zapis_pliku,Fs);

      syg_wej=audioplayer(x,Fs);
      playblocking(syg_wej);
      sound(y,Fs);
end
```

Listing 9. Realizacja efektu tremolo dla środowiska MatLAB

```
//funkcja efektu bypass dla procesora sygnałowego
void ByPass()
{
    wejscie = input_sample();
    output_sample(wejscie);
}
```

Listing 10. Implementacja bypass dla procesora sygnałowego

```
//funkcja sumowania kanałów dla procesora sygnałowego
void Stereo()
{
AIC23_data.channel[RIGHT] = input_right_sample();
AIC23_data.channel[LEFT]=input_left_sample();
wyjscie=((AIC23_data.channel[LEFT])+(AIC23_data.channel[RIGHT]))/2;
output_sample(wyjscie);
}
```

Listing 11. Realizacja sumowania kanałów dla procesora sygnałowego

```
//funkcja pogłosu dla procesora sygnałowego
void Reverb()
{
 {
     wejscie = input_left_sample();
     opoznienie = buffer[n];
     wyjscie = wejscie + opoznienie;
     output_left_sample(wyjscie);
     buffer[n] = wejscie + opoznienie * Gain;
 }
}
```

Listing 12. Realizacja pogłosu dla procesora sygnałowego

```
//funkcja realizująca opóźnienie dla procesora sygnałowego
void Delay()
{
    {
        wejscie = input_left_sample();
        opoznienie = buffer[n];
        buffer[n] = wejscie;
        wyjscie = opoznienie;
        suma_delay = wejscie + Gain * wyjscie;
        output_left_sample(suma_delay);
    }
}
```

Listing 13. Realizacja opóźnienia dla procesora sygnałowego

```
//funkcja realizująca efekt flanger dla procesora sygnałowego
void Flanger()
{
 {
     wejscie = input_left_sample();
     opoznienie = round((D / 2) * (1 - cos(2 * pi * n * fcycle)));
     buffer[n] = wejscie * opoznienie;
     efekt_flanger = (0.5 * wejscie) + (Gain * buffer[n]);
     wyjscie = efekt_flanger;
     output_left_sample(wyjscie);
 }
}
```

Listing 14. Realizacja efektu *flanger* dla procesora sygnałowego

```
//funkcja realizująca efekt chorus dla procesora sygnałowego
void Chorus()
{

 {
     wejscie = input_left_sample();
     opoznienie = round((0.5 + (cos(2 * pi * n * fdelay))));
     buffer[n] = (wejscie * opoznienie);
     efekt_chorus = (0.5 * wejscie) + (0.5 * buffer[n]);
     wyjscie = efekt_chorus;
     output_left_sample(wyjscie);
 }
}
```

Listing 15. Realizacja efektu *chorus* dla procesora sygnałowego

```
//funkcja realizująca efekt tremolo dla procesora sygnałowego
void Tremolo()
{
    {
        wejscie = input_left_sample();
        opoznienie = (1 + Amplituda * (cos(2 * pi * fmod * n)));
        buffer[n] = wejscie;
        efekt_tremolo = buffer[n] * opoznienie;
        wyjscie = efekt_tremolo;
        output_left_sample(wyjscie);
    }
}
```

Listing 16. Realizacja efektu *tremolo* dla procesora sygnałowego