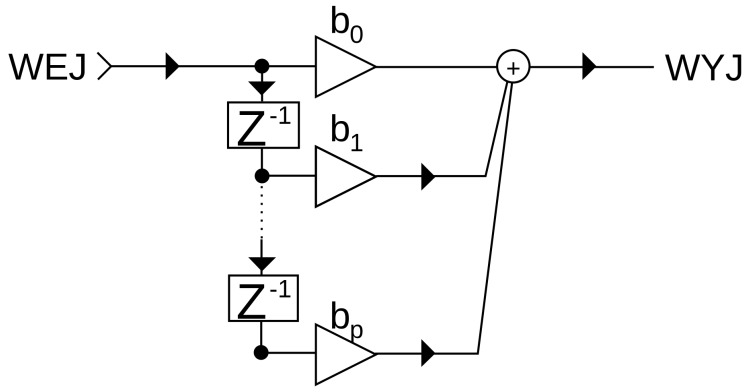
Implementacja filtru FIR

1 Wprowadzenie

1.1 Krótki opis filtrów FIR

Filtry FIR, czyli o skończonej odpowiedzi impulsowej, cechują się łatwością implementacji (ze względu na brak pętli zwrotnych), są stabilne, ale także zużywają więcej zasobów niż filtry IIR.

Odpowiedź impulsowa filtru, jest to odpowiedź filtru na impuls jednostkowy. W przypadku filtru FIR, odpowiedź impulsowa filtru, jest równa jego współczynnikom. Kiedy sekwencja współczynników filtru jest symetryczna, filtr ma liniową fazę, co oznacza, że opóźnia każdą składową częstotliwościową sygnału, o tę samą wartość.



Rysunek 1: Schemat blokowy filtru FIR [1]

Pokazane na schemacie blokowym wartości b_i to współczynniki filtru, a elementy Z^{-1} (transformata Z), to bloki opóźniające o 1 próbkę.

1.2 Format Q15

Jako, że procesor, który ma wykorzystywać tę implementację filtru, nie posiada jednostki zmiennoprzecinkowej, wszystkie liczby ułamkowe, muszą być zapisane w formacie Q15. W ten sposób kazdą liczbę z zakresu <-1, 1), możemy zapisać jako liczbę całkowitą z zakresu <-32768, 32767> z dokładnością $2^{-15} = 0,000030517578125$.

1.2.1 Operacje na liczbach Q15

Podczas dodawania liczb Q15 należy uważać na przepełnienie zakresu. Na procesorach z rodziny C55xx, typ int z języka C, ma zakres 16 bitów, a więc <-32768, 32767>. Gdy wynikiem obliczeń jest wartość większa niż dany zakres, trzeba użyć większego typu danych, jak long, lub stworzyć bufor przepełnienia, czyli drugą zmienną która będzie przechowywać starsze bity, dzięki czemu jedna wartość będzie mogła być zapisana w 2 zmiennych.
Mnożąc liczby w formacie Q15 dostajemy wynik w postaci Q30, aby dostać liczbę Q15, musimy podzielić wynik przez 2^{15} co jest równoznaczne z przesunięciem bitowym w prawo o 15 bitów:

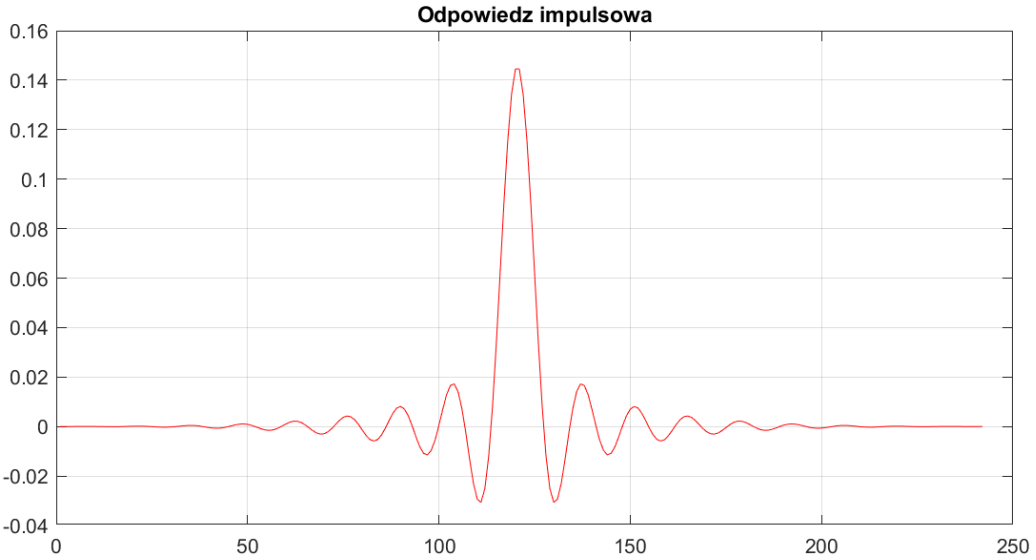
$x_{q30} >> 15 = x_{q15}$

2 Rozpoznanie parametrów filtru

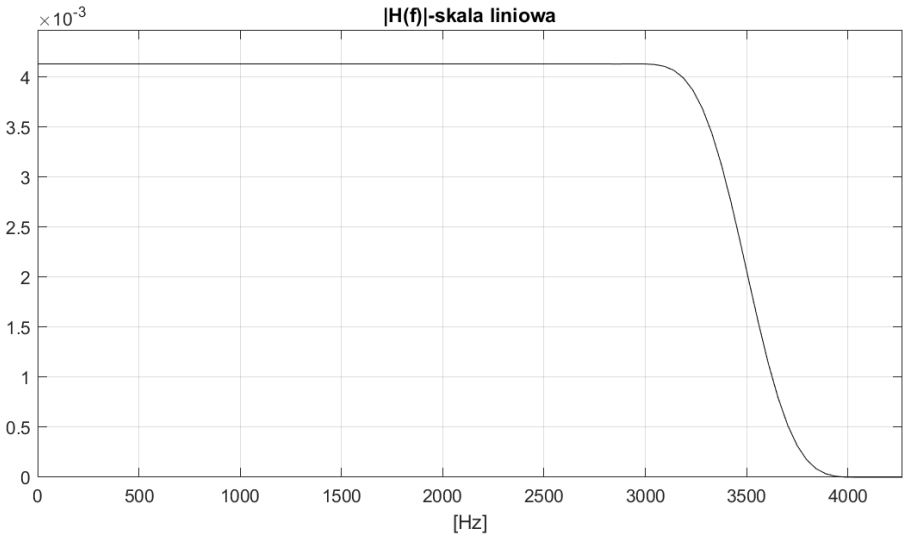
Skrypt w matlabie:

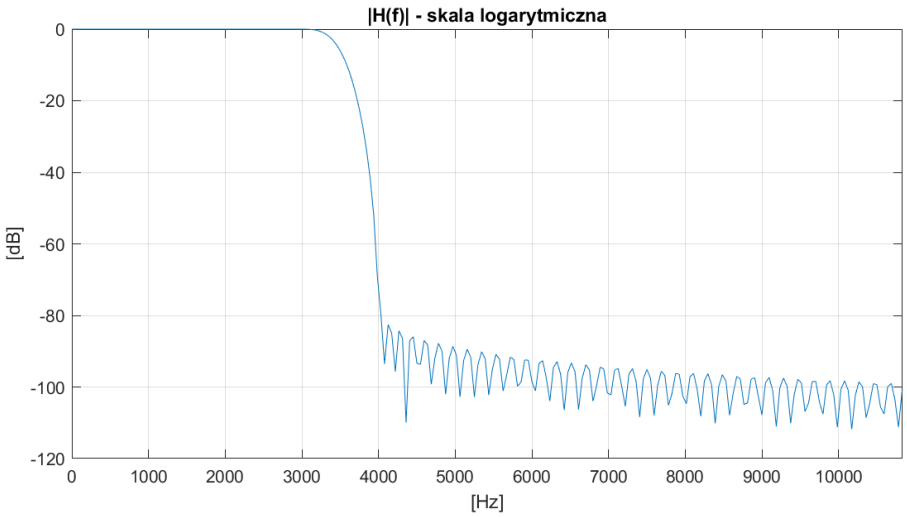
```
% czestotliwosc probkowania
fs = 48000;

%odpowied impulsowa filtru
k=1:n;
k(1)=0;
x = double(kroneckerDelta(sym(k)));
y = filter(coeff,1,x);

% funkcja przenoszenia filtru - H dziedzina funkcji przenoszenia - w
[H,w]=freqz(y,n);

% zmiana jednostek dziedziny na [Hz]
w=w*fs/(2*pi);

% funkcja przenoszenia w skali liniowej
Habs=abs(H);

% funkcja przenoszenia w skali logarytmicznej
Hmax=max(Habs);
HdB=20*log10(Habs/Hmax);
```

Aby uzyskać funkcję odpowiedzi impulsowej filtru, przefiltrowaliśmy deltę kroneckera za pomocą funkcji matlabowej filter2.



Następnie, żeby znaleźć częstotliwości graniczne oraz rodzaj filtru, użyliśmy funkcję freqz zwracającą funkcję przenoszenia filtru.





Z wykresów widzimy, że jest to filtr dolnoprzepustowy o częstotliwości granicznej około 4080Hz

3 Implementacja filtru FIR w środowisku Code Composer Studio

3.1 Konwersja i zapisanie do pliku współczynników filtru z matlaba

Konwersja współczynników na Q15:

```
function [y]=to_q15(coeffs)

y=coeffs.*32768;
y=round(y);
```

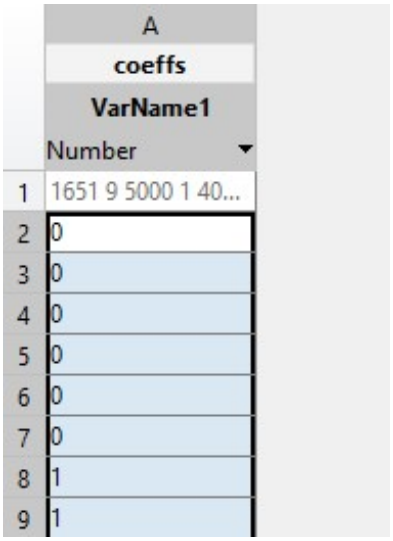
po konwersji, współczynniki są przybliżane do najbliższej liczby całkowitej, ponieważ w formacie Q15 liczby muszą być typu integer.

Skrypt zapisujący współczynniki w q15 do pliku 'coeffs.dat':

```
coeffs_q15 = to_q15(coeff);
header = '1651 9 5000 1 400 2';

fid = fopen('coeffs.dat', 'w');
fprintf(fid, '%s\n', header);
fprintf(fid, '%d\n', coeffs_q15);
fclose(fid);
```

Aby code composer studio, mogło odczytać dane z pliku, dodaliśmy wymagany pierwszy wiersz, znajdujący się w zmiennej header.

Po wykonaniu się skryptu, otrzymujemy plik z 243 wierszami:



3.2 Implementacja filtru z użyciem biblioteki DSPLIB

```
const int no_of_coeffs = 242;
DATA coeffs_q15[242];
DATA _sample_history[244];


void main( void )
{
    int i;

    DATA sample;
    DATA output;

    for(i=0;i<no_of_coeffs+2;i++) _sample_history[i]=0;

    for(i=0;i<no_of_coeffs;i++)
    {
        output=0;

        if(i==0) sample = 32767;
        else sample =0;

        fir(&sample, coeffs_q15, &output, _sample_history, 1, no_of_coeffs);

        printf("Wynik: %d\n", coeffs_q15[i]-output);

    }

}
```

Funkcja fir z biblioteki dsplib, przyjmuje 6 argumentów:

1. DATA *x - wskaźnik do tablicy próbek wejściowych
2. DATA *h - wskaźnik do tablicy współczynników
3. DATA *r - wskaźnik do tablicy próbek wyjściowych
4. DATA *dbuffer - wskaźnik do tablicy bufora opóźnień, czyli historii próbek
5. ushort nx - liczba próbek wejściowych
6. ushort nh - liczba współczynników filtru

Do funkcji przekazujemy po jednej próbce wejściowej, dlatego przekazujemy adres zmiennej wejściowej oraz zmiennej wyjściowej.

Zgodnie z dokumentacją powodu bezpieczeństwa tablica bufora opóźnień powinna być dłuższa od tablicy współczynników o 2 elementy

Zmienne lokalne są przechowywane na stosie, więc aby nie doprowadzić do jego przepełnienia tablice współczynników i bufora opóźnień zostały zadeklarowane jako zmienne globalne.

Pętla for została stworzona w taki sposób, aby jako wejście filtra przekazać deltę kroneckera, tzn. pierwszy element to najwyższa wartość z zakresu w Q15 a każda następna jest równa 0.

3.3 Własna implementacja filtru w języku C

```
int fir(int sample, int *coeffs, int *sample_history, int coeffs_length)
{
    //pozycja aktualnej probki w tablicy sample_history
    static int current_sample_delay=0;

    //ustawienie wskaznika na pozycje aktualnej probki
    int *current_sample = sample_history+current_sample_delay;
    //dodanie wartosc sample do tablicy sample_history
    *current_sample=sample;

    int output =0;
    int i;

    for(i=0;i<coeffs_length;i++)
    {
        //implementacja bufora kolowego
        if(i<=current_sample_delay) current_sample =
            (sample_history+current_sample_delay-i);
        else current_sample =sample_history+(coeffs_length-1)-(i-current_sample_delay);

        //obliczanie wyjcia filtru
        output += (int)(((*current_sample*(long)*(coeffs+i))+16384)>>15);
    }

    if(current_sample_delay>=coeffs_length) current_sample_delay = 0;
    else current_sample_delay+=1;

    return output;
}
```

Powyższa implementacja filtru FIR, niewiele różni się od tej z DSPLIB.
Główna różnica to parametry funkcji.

1. int sample - wartość próbki wejściowej, w przeciwieństwie do DSPLIB nie można podać więcej niż jednej próbki jako wejście
2. int *coeffs - wskaźnik do tablicy współczynników
3. int *sample_history - wskaźnik do bufora opóźnień
4. int coeffs_length - ilość współczynników filtru

Funkcja zwraca wyjście, więc nie potrzebuje wskaźnika do tablicy wyjściowej tak jak fir z dsplib

Funkcja przechowuje pozycję próbki przekazanej do funkcji w tablicy opóźnień, w zmiennej current_sample_delay. Za każdym wywołaniem funkcji, inicjalizowany jest wskaźnik do miejsca gdzie ma znajdować się przekazana próbka i zostaje tam wpisana.

W pętli for wykonuje się algorytm obliczania wyjścia funkcji fir.
Na początku pętli for, znajduje się implementacja bufora kołowego, który przesuwa wskaźnik current_sample do tyłu w tablicy historii próbek, gdy adres na, który wskazuje wskaźnik jest równy początkowi tablicy, wskaźnik jest ustawiany na koniec tablicy.

Wyjście funkcji jest obliczane ze wzoru:

$$y(x) = \sum_{i=0}^{n} h(i)x(n-i)$$

Gdzie:

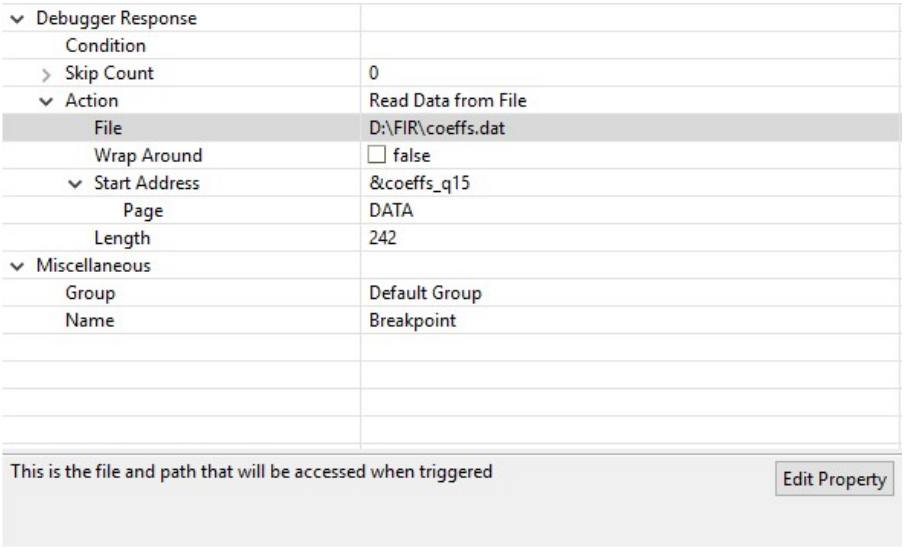
- n - liczba współczynników filtru / rząd filtru
- h(i) - współczynnik filtru o indeksie i
- x(n-i) - wartość z tablicy opóźnień, gdzie x(n) to najnowsza próbka

Przy obliczaniu wyjścia funkcji, dzielimy wynik mnożenia przez 32768 (»15), ponieważ wykonujemy to działanie na wartościach w formacie Q15. Przed tym, do wartości mnożenia, dodajemy wartość 0.5, która w Q15 wynosi 16384. Jest tak dlatego, że podczas dzielenia wartości int w języku C, wartości, które powinny być przybliżane w górę, są przybliżane w dół, więc aby temu zapobiec dodajemy 0.5.

```
const int no_of_coeffs = 242;
int coeffs_q15[242];
int _sample_history[242];

int fir(int sample, int *coeffs, int *sample_history, int coeffs_length);

int main()
{
    int sample =0;

    int i;
    int output;

    for(i=0;i<no_of_coeffs+2;i++) _sample_history[i]=0;

    for(i=0; i<no_of_coeffs; i++)
    {
        output=0;

        if(i<1) sample = 32767;
        else sample = 0;

        output = fir(sample, coeffs_q15, _sample_history, no_of_coeffs);
    }
}
```

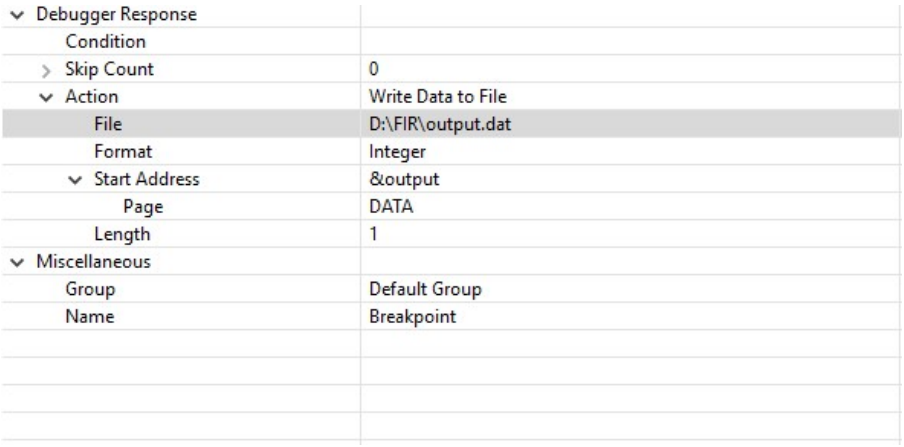
Funkcja main nie różni się dużo od tej z implementacji DSPLIB.

3.4 Zapisanie wyniku filtracji do pliku i porównanie w matlabie



Żeby wczytać współczynniki z pliku, zmieniamy ustawienia breakpointu, ustawionego na początku funkcji main, i ustawiamy adres początkowy na początek tablicy coeffs_q15 oraz długość na 242.

Na końcu na funkcji for tworzymy podobny breakpoint, tym razem do zapisywania danych.

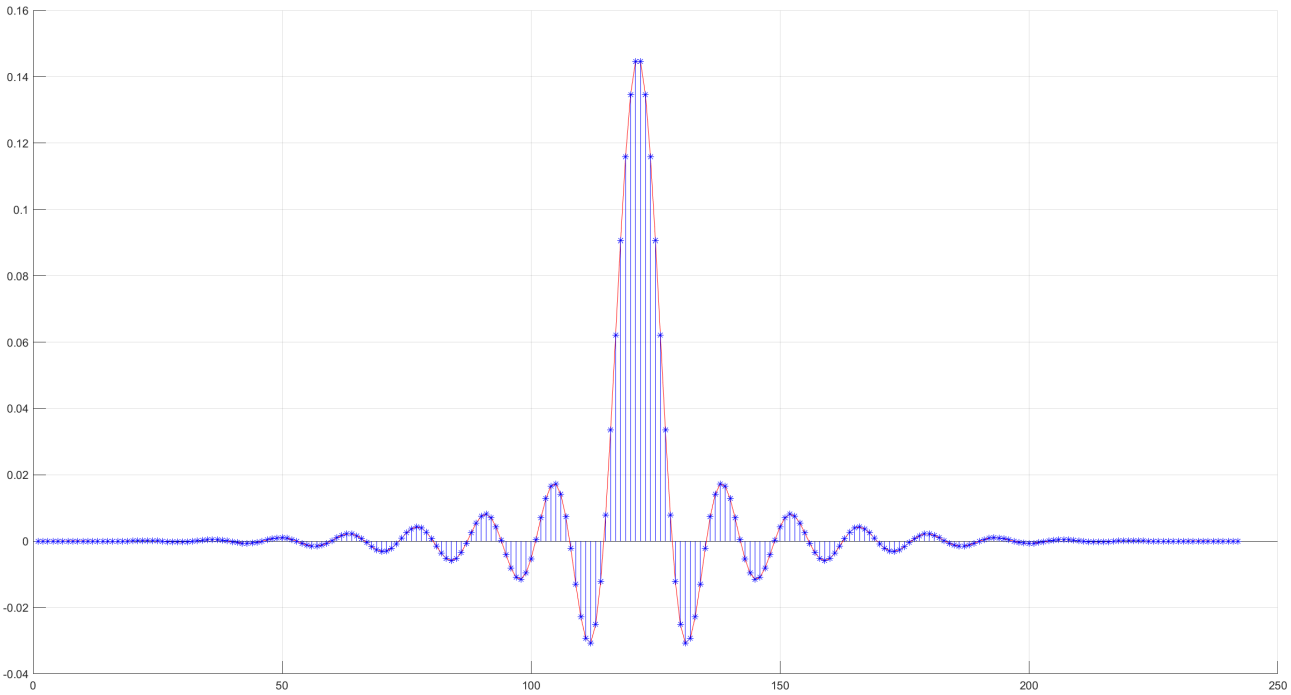


Po wykonaniu programu, wykonujemy skrypt w matlabie:

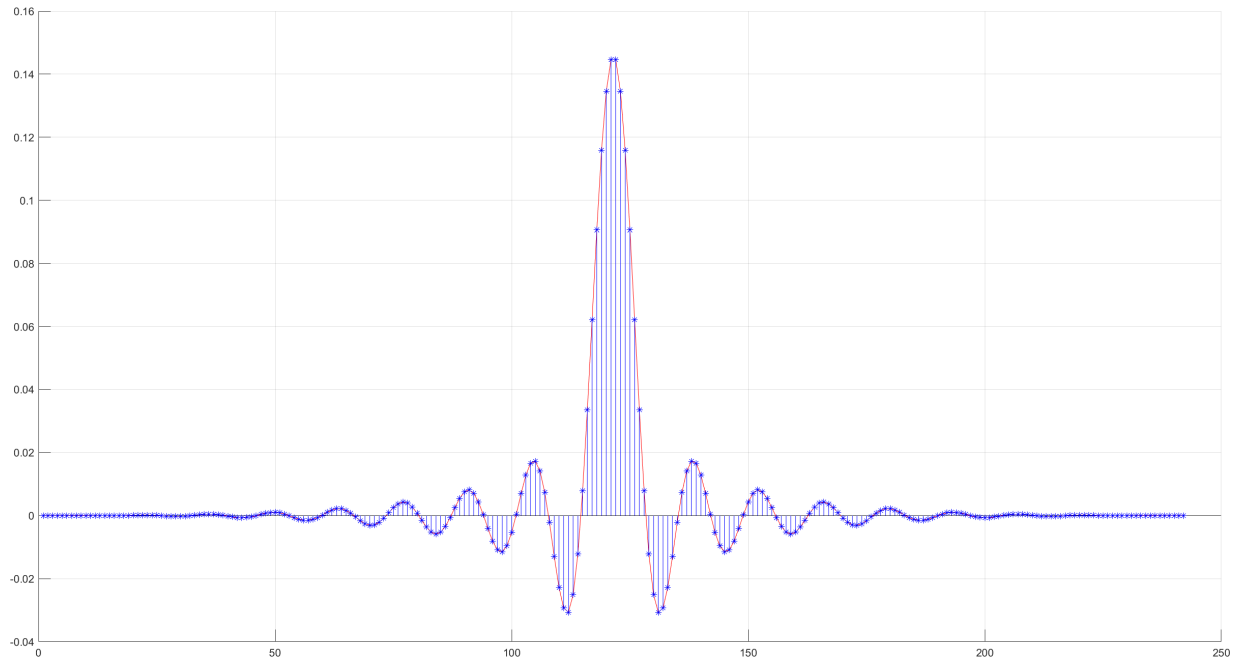
```
coeff=[-6.875475350454e-006,-8.817704749919e-006,-8.748117386016e
    -006,-5.724146095245e-006, % itd];


import_output; % automatycznie wygenerowany skrypt matlaba zapisujacy dane
     z pliku output.dat do zmiennej "output"
y = table2array(output)./32768;

grid on; zoom on; hold on;

plot(y,'r');
stem(coeff,'b*');
```

Skrypt wczytuje wartości z pliku 'output.dat' i wyświetla wykres porównujący te wartości ze współczynnikami filtru

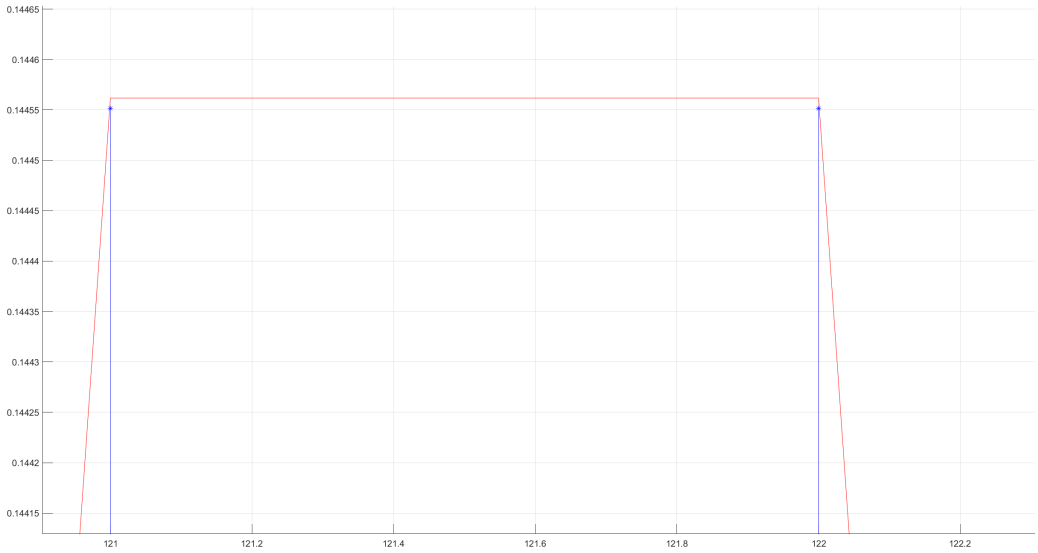
Wykres porównujący współczynniki z wyjściem z programu używającego implementację DSPLIB:



Wykres czerwony to wyjście z filtru FIR a niebieski to współczynniki.
Wykres porównujący współczynniki z wyjściem z programu uzywającego własną implementację w C:



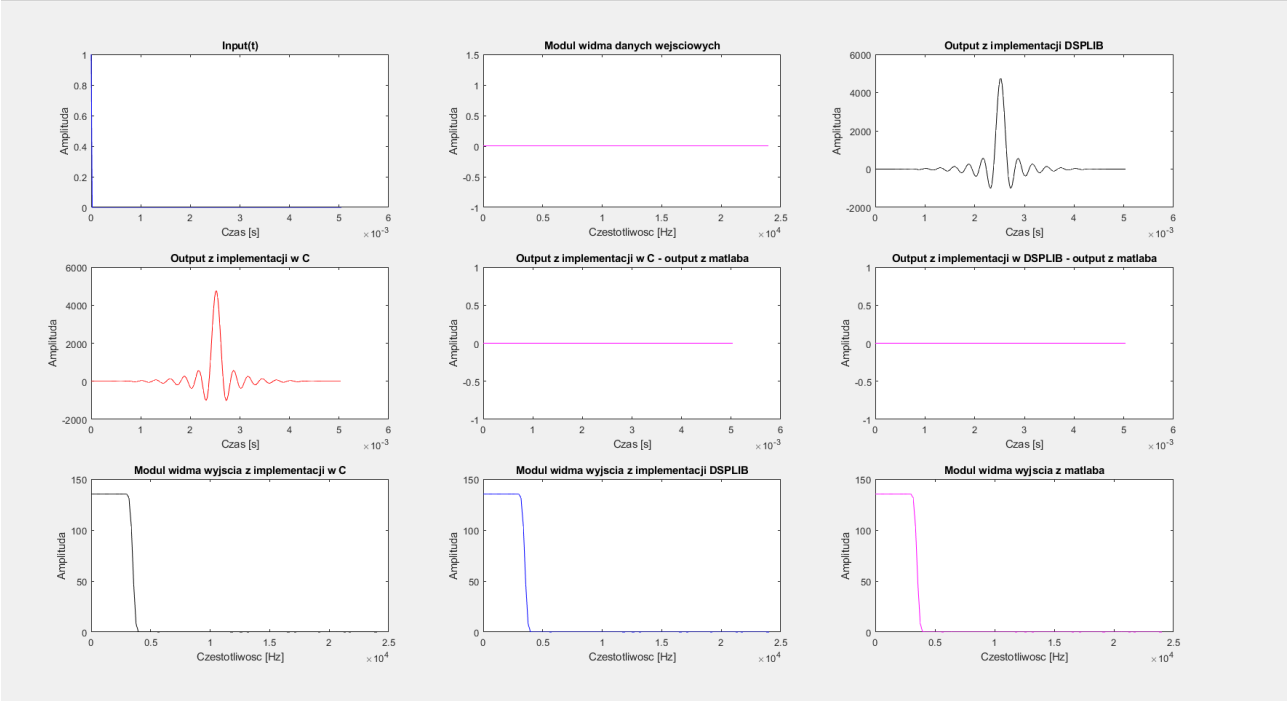
Jak widać wartości są do siebie bardzo zbliżone, ale jeśli przyjrzeć się bliżej to pojawiają się małe różnice:



Różnice te wynikają prawdopodobnie z przybliżeń wartości podczas konwersji, z i na Q15.

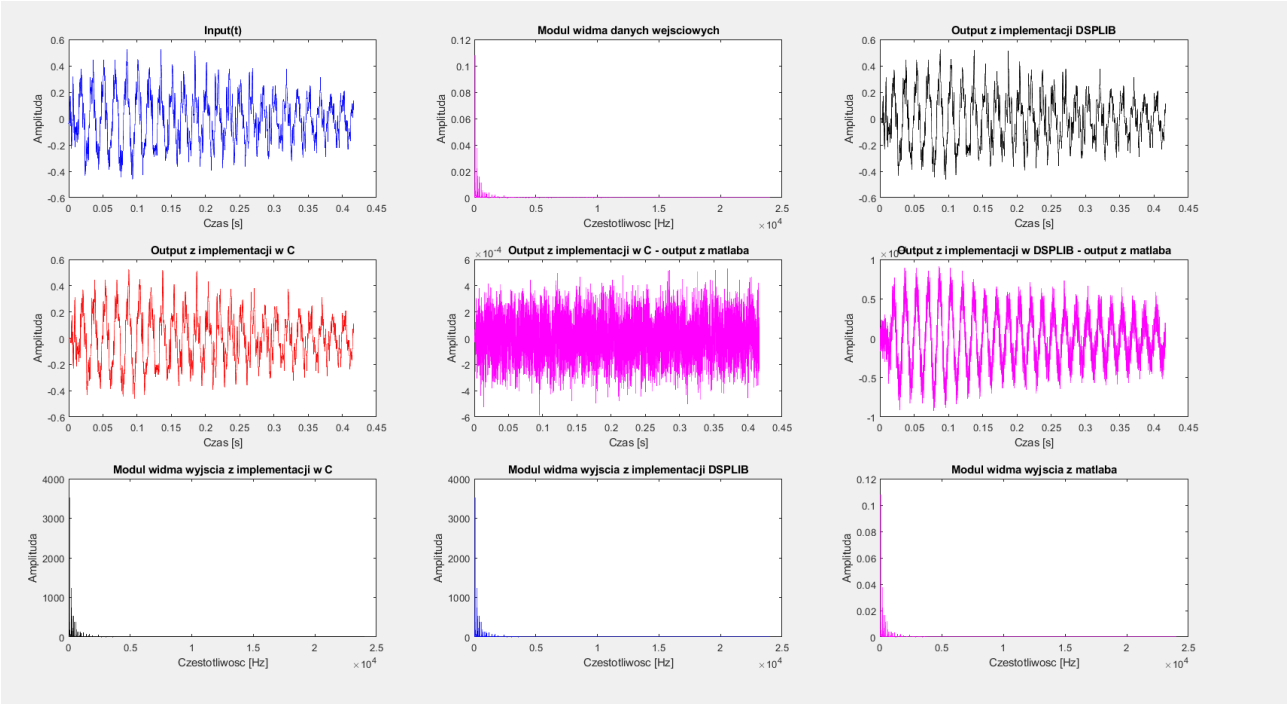
4 Testowanie implementacji za pomocą sygnałów testowych

4.1 Delta kroneckera



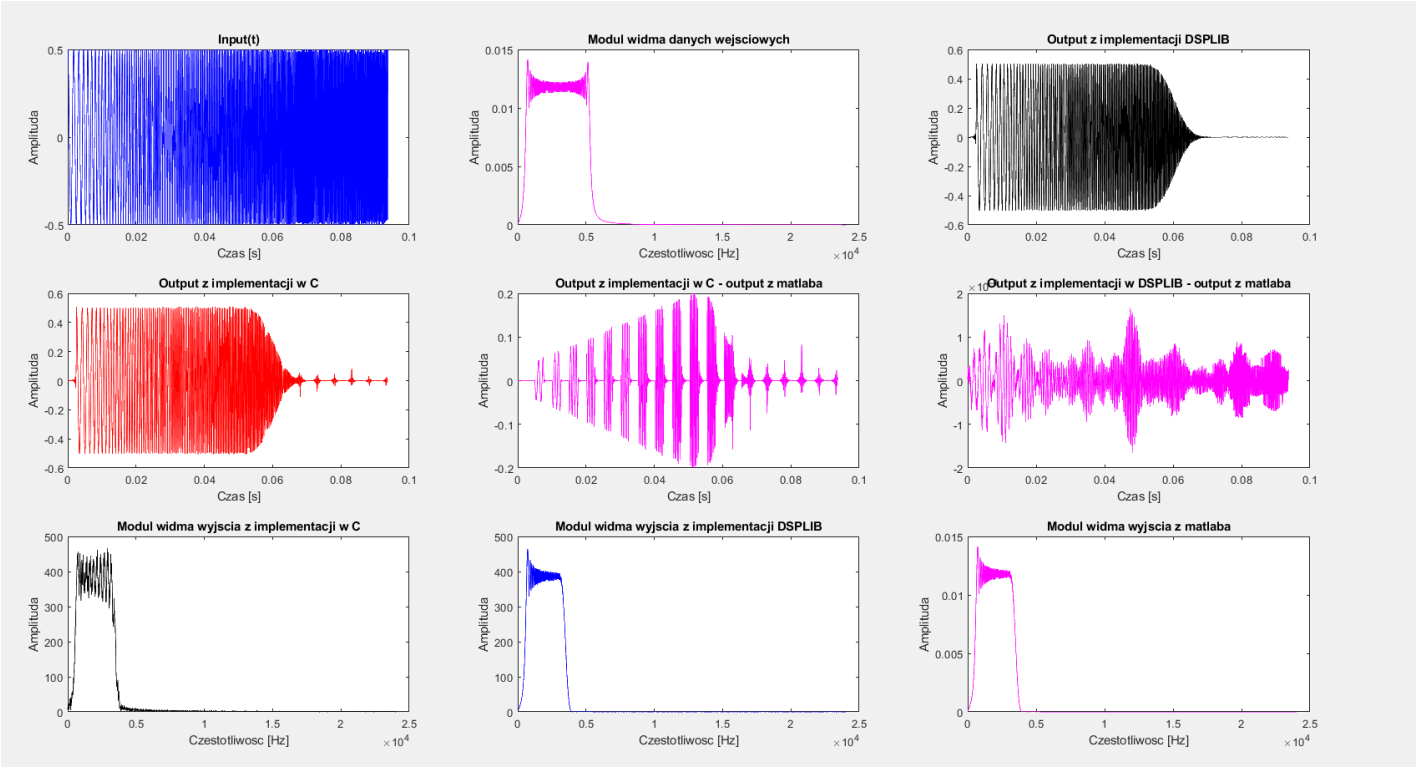
W przypadku delty kroneckera, otrzymujemy to samo wyjście z każdej implementacji. Po transformacie fouriera odpowiedzi impulsowej, otrzymujemy charakterystykę częstotliwościową filtru, na której widać, że jest to filtr dolnoprzepustowy o częstotliwości granicznej około 4000Hz.

4.2 Fragment utworu muzycznego



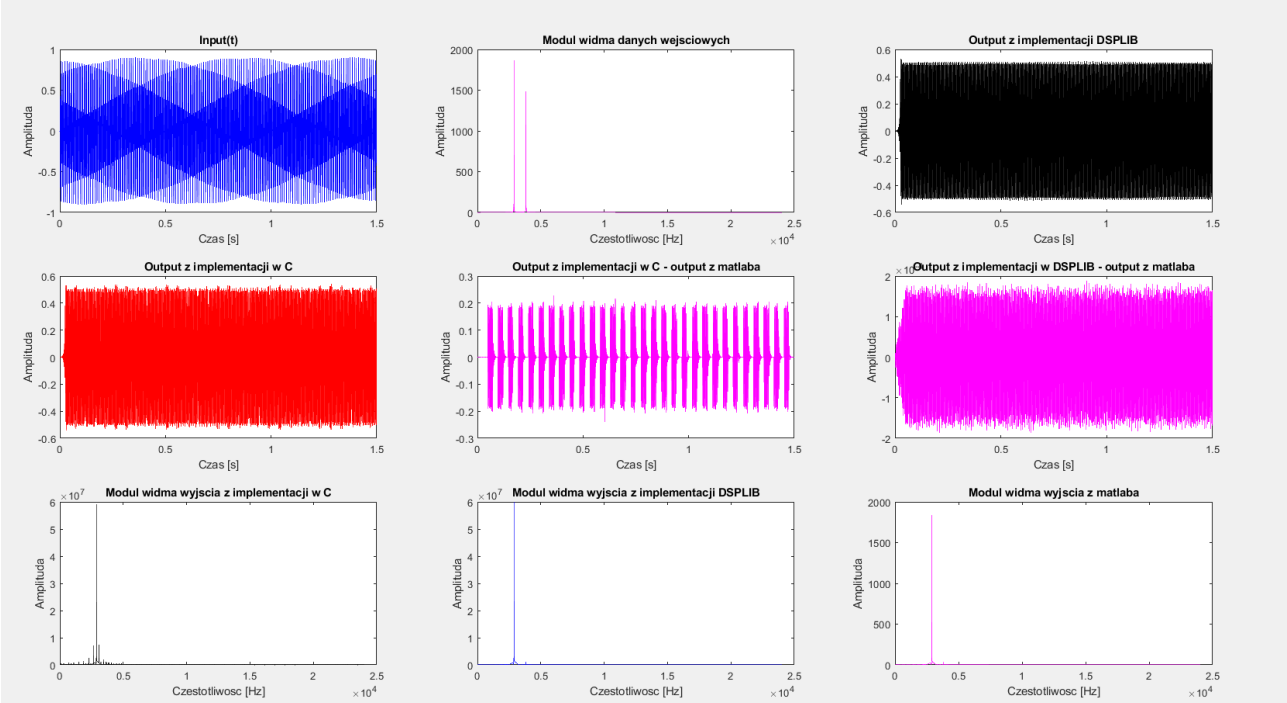
Dla fragmentu utworu muzycznego jako wejście, otrzymaliśmy lekko różniące się wyjście z implementacji w C niż tej z użyciem biblioteki DSPLIB. Natomiast filtr nadal spełnił swoją funkcję, filtrując częstotliwości powyżej 4000Hz. Amplitudy tych sygnałów były niskie, stąd nie widać tego, aż tak dobrze na powyższych wykresach.

4.3 Sygnał chirp



Filtrując sygnał chirp filtrem z implementacji w C, widać, że powyżej pewnej częstotliwości pozostawiła ona skoki w amplitudzie, których nie ma w wyjściu z implementacji DSPLIB ani matlabowej funkcji filter. Widać to również w widmie wyjścia tych implementacji. Nadal jednak odfiltrowane zostały częstotliwości powyżej 4000Hz.

4.4 Suma sinusów



Przy wynikach filtrowania sumy sinusów, można zauważyć przeciek widma, na wykresie widma z implementacji w C. Może to wynikać ze sposobu przybliżania wartości podczas dzielenia, przez co jak widać również na wykresie, wyjście z implementacji w C nie ma tak równej amplitudy w całym przedziale jak chociażby wyjście z filtru z DSPLIB. Ten sam problem może dotyczyć sygnału chirp.