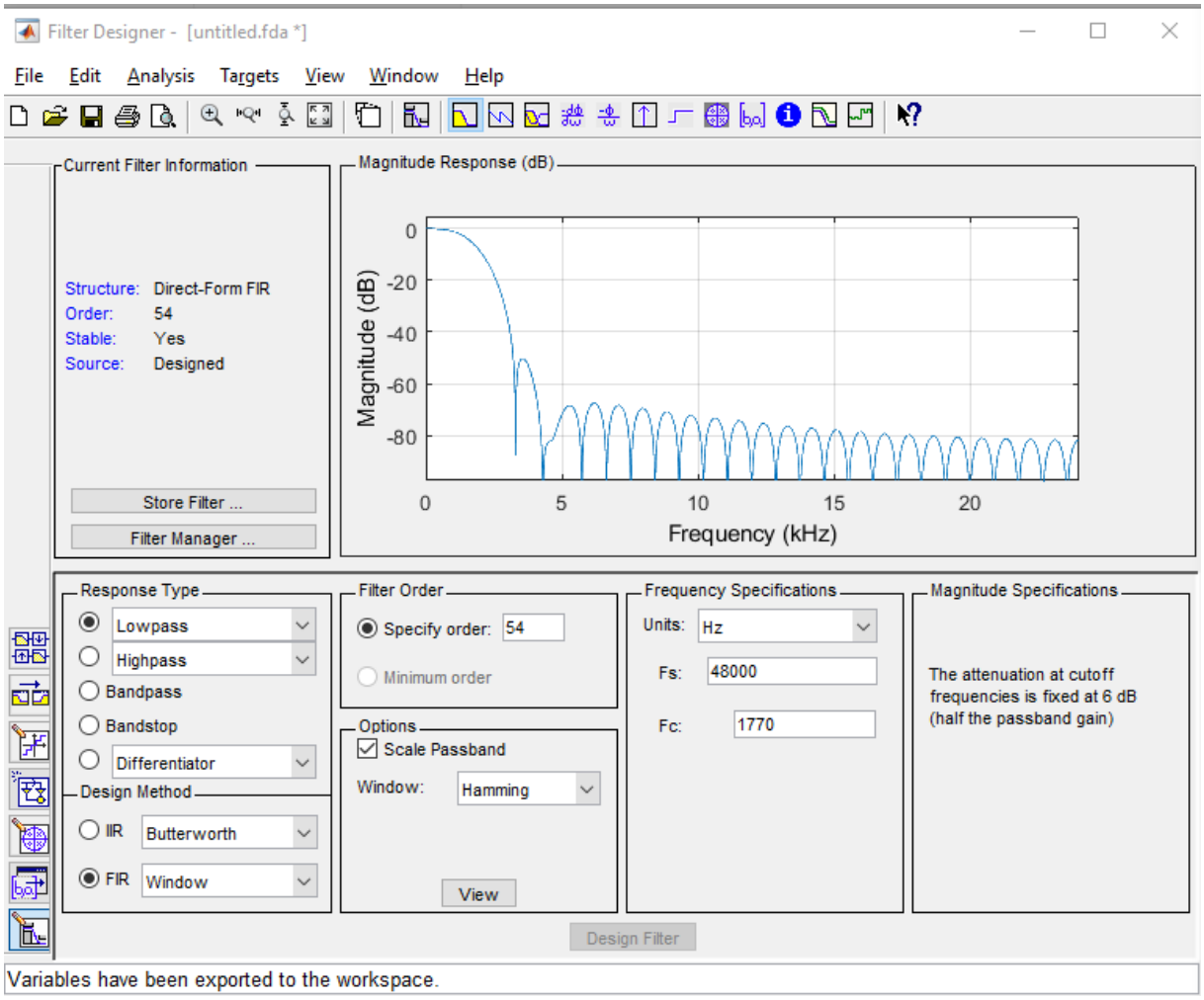
Zastosowania Procesorów Sygnałowych
Raport z zadania projektowego nr 2
Filtry FIR
Mateusz Miler 171577
17.05.2020

1. Obliczanie współczynników filtru

Długość filtru: 55
Częstotliwość graniczna: 1770
Wybrano okno Hamminga



Rys.1. Widok narzędzia fdatool do projektowania filtru

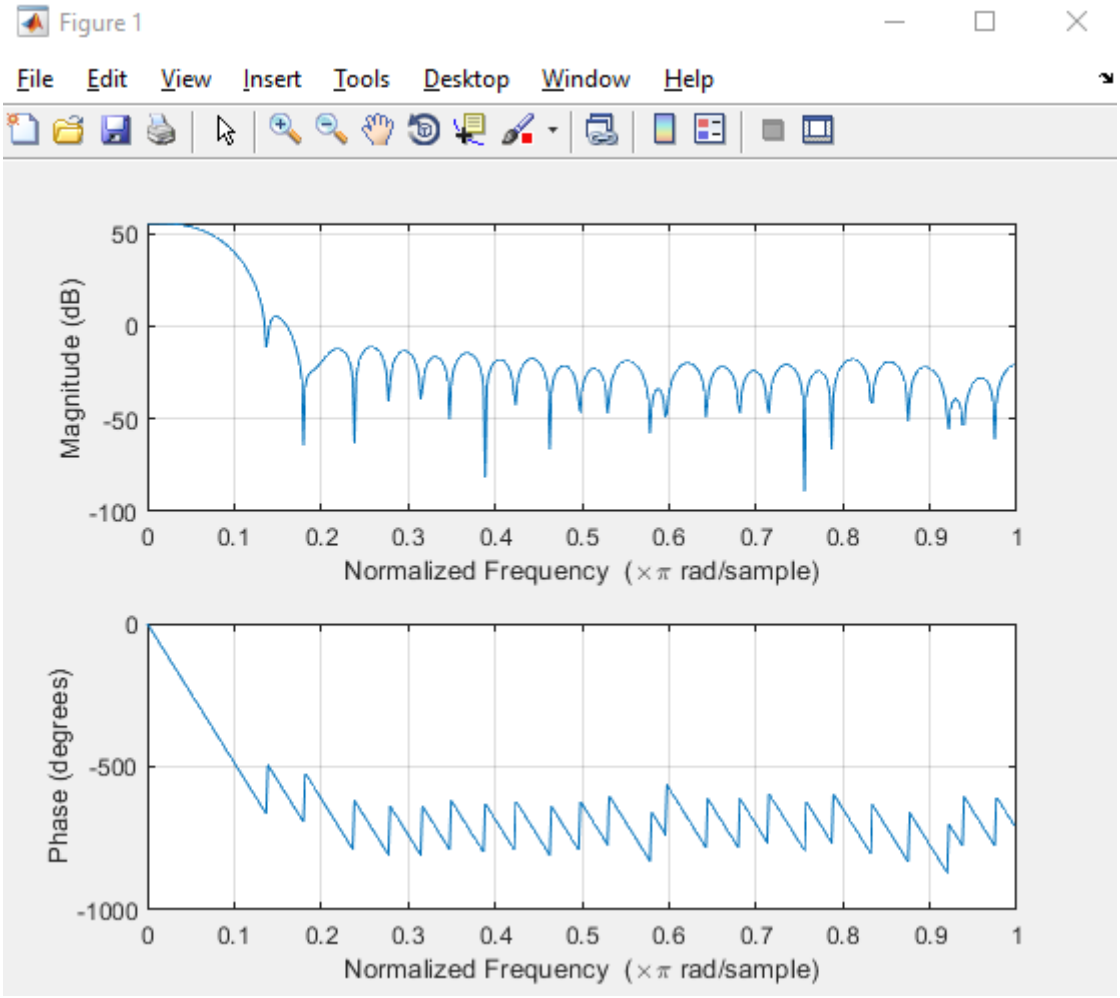
```
>> Wspolczynniki=round(32768*Num2)
```

Rys.2. Przekształcenie wyeksportowanych parametrów do Q15

```
//wspolczynniki filtru FIR
//Dlugosc filtru 55, liczba wspolczynnikow 55, rzad filtru 54
const short coefficients[] = {-1, -9, -18, -31, -48, -68,   -92, -116, -138, -153,
                -156, -142, -104, -39, 59, 190, 355, 551, 774, 1016,
                1268, 1520, 1760, 1975, 2156, 2293, 2378, 2407, 2378, 2293,
                2156, 1975, 1760, 1520, 1268, 1016, 774, 551, 355, 190,
                59, -39, -104, -142, -156, -153, -138, -116, -92, -68,
                -48, -31, -18, -9, -1};

```

Rys.3. Współczynniki filtru przekształcone do Q15



Wyk.1. Wykres charakterystyki częstotliwościowej filtru dla stałoprzecinkowych współczynników

Charakterystyki wydają się spełniać założenia projektowe.

Stałe wzmocnienie filtru

Gain = 32767

Wzmocnienie mieści się w zakresie liczb Q15 <-32768; 32767>. Współczynniki nie wymagają dodatkowego przeskalowania.

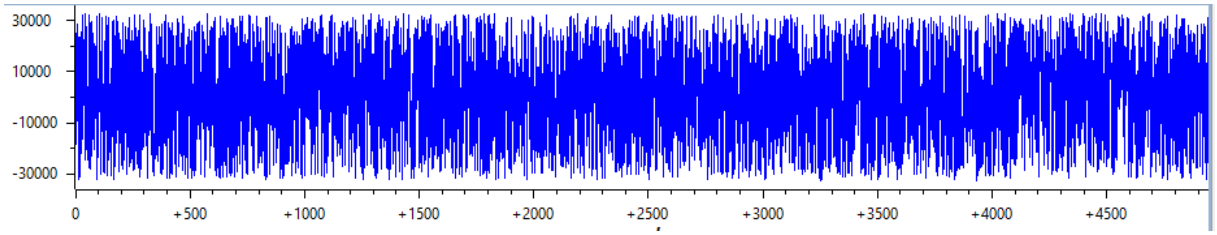
2. Własna implementacja filtru FIR

```
//Filtr FIR
//input: wskaźnik do tablicy zawierającej próbki sygnału do przefiltrowania
//filter: wskaźnik do tablicy zawierającej współczynniki filtru
//output: wskaźnik do tablicy, w której zostaną zapisane wyniki filtracji
//numSamples: liczba próbek w tablicy
//numFilter: liczba współczynników filtru
void blockfir(short* input, const short* filter, short* output, int numSamples, int numFilter)
{
    int i;
    int j;
    long y;
    //Zaczynamy od ostatniej probki (najnowszej), zeby w przypadku podania
    //tej samej tablicy wejsciowej i wyjsciowej, móc ją nadpisać
    //(probek od konca nie bedziemy juz uzywac do obliczen)
    for(i = numSamples-1; i >= 0; i--)
    {
        for(j = 0; j < numFilter; j++)
        {
            if(i-j < 0) break;
            y = _smaci(y, filter[j], input[i-j]);
        }
        //Zaokraglenie i zamiana z Q30 do Q15
        y = (y + (1<<14))>>15;
        output[i] = y;
    }
}
```

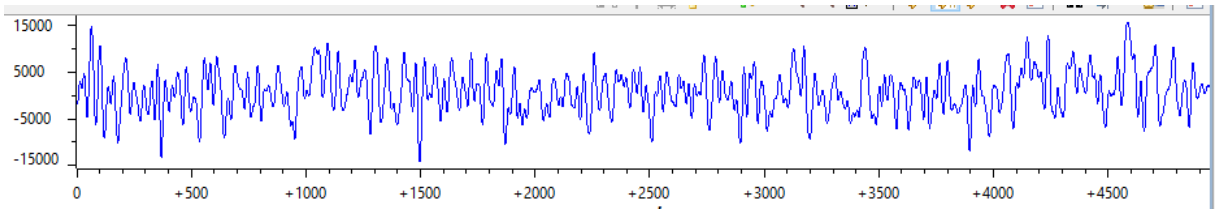
Rys.4. Kod napisanej funkcji

Liczba operacji procesora potrzebnych do przefiltrowania tablicy 5000 próbek: **9 176 626**

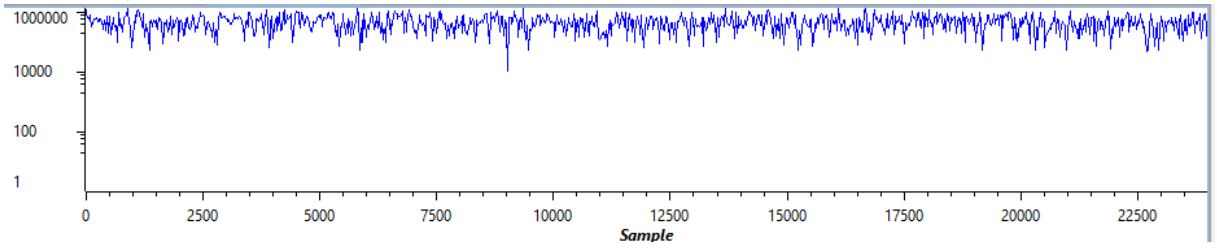
Wykresy białego szumu



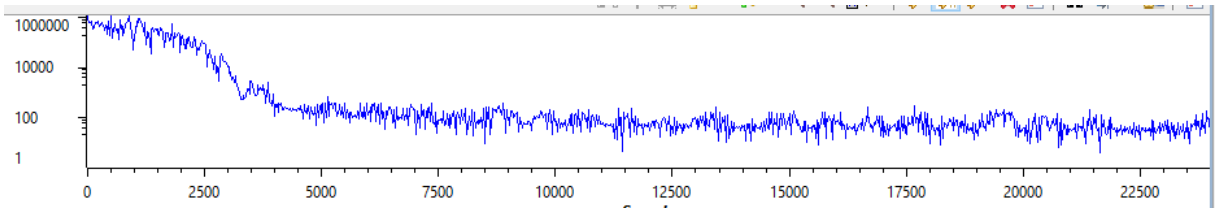
Wyk.2. Próbki białego szumu przed filtracją bez pierwszych 55 próbek (długość filtru)



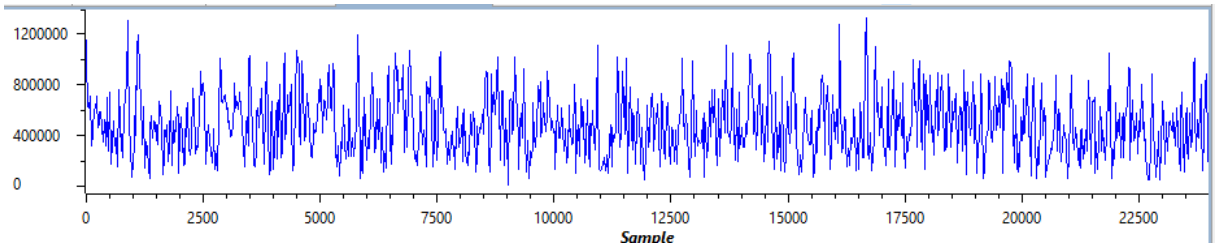
Wyk.3. Próbki białego szumu po filtracji bez pierwszych 55 próbek (długość filtru)



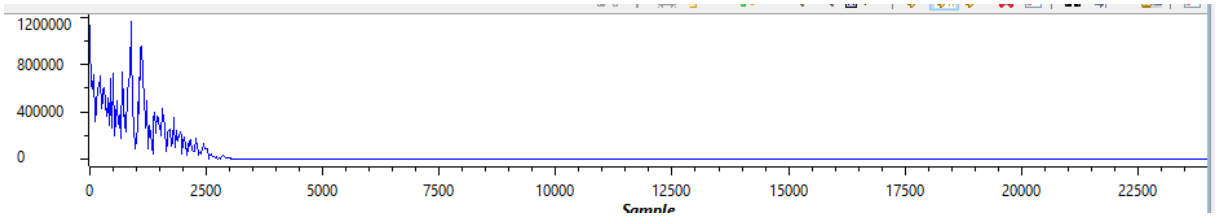
Wyk.4. Widmo białego szumu w skali logarytmicznej przed filtracją bez pierwszych 55 próbek (długość filtru)



Wyk.5. Widmo białego szumu w skali logarytmicznej po filtracji bez pierwszych 55 próbek (długość filtru)

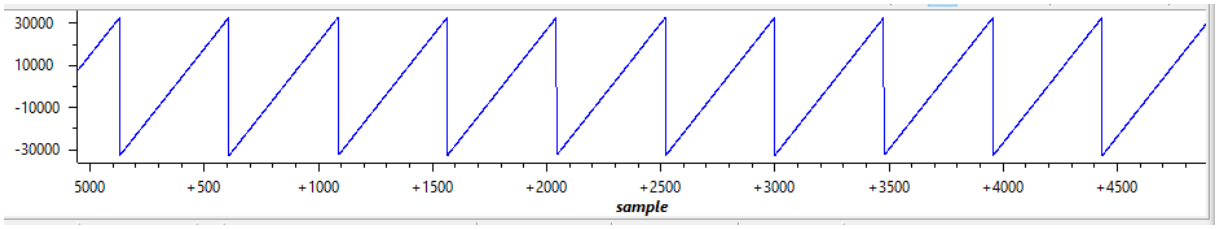


Wyk.6. Widmo białego szumu w skali liniowej przed filtracją bez pierwszych 55 próbek (długość filtru)

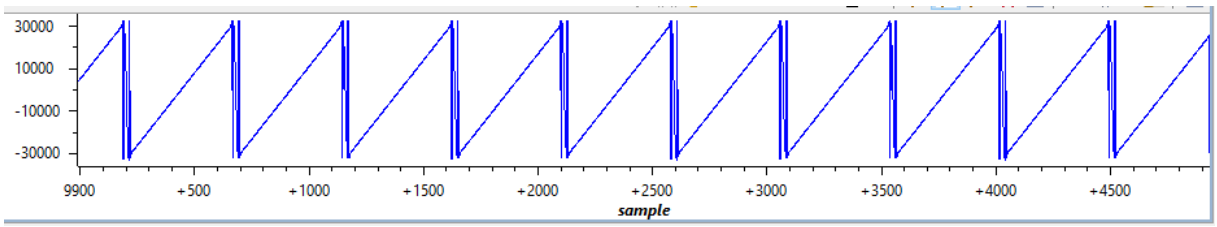


Wyk.7. Widmo białego szumu w skali liniowej po filtracji bez pierwszych 55 próbek (długość filtru)

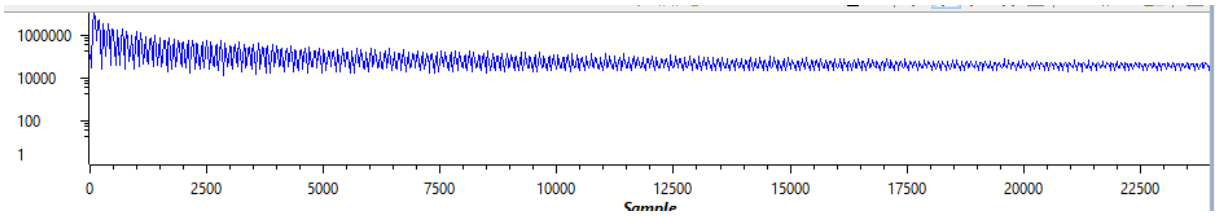
Wykresy sygnału piłokształtnego



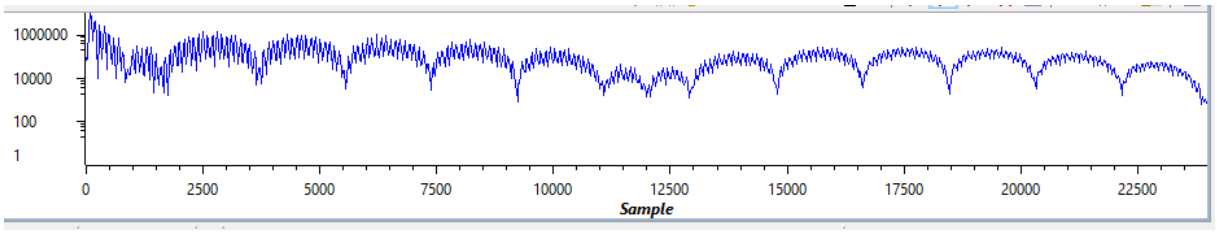
Wyk.8. Próbki sygnału piłokształtnego przed filtracją bez pierwszych 55 próbek (długość filtru)



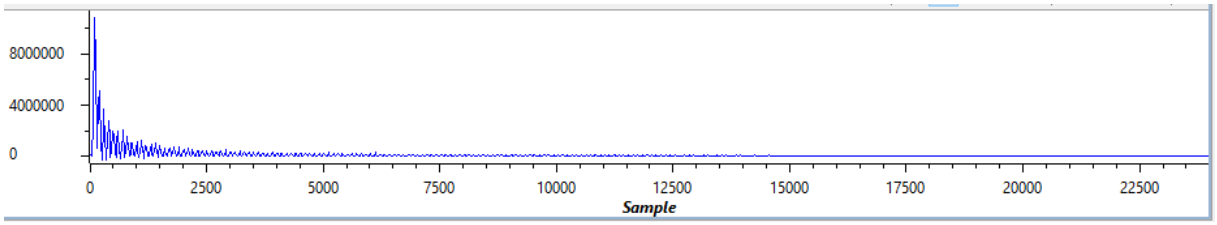
Wyk.9. Próbki sygnału piłokształtnego po filtracji bez pierwszych 55 próbek (długość filtru)



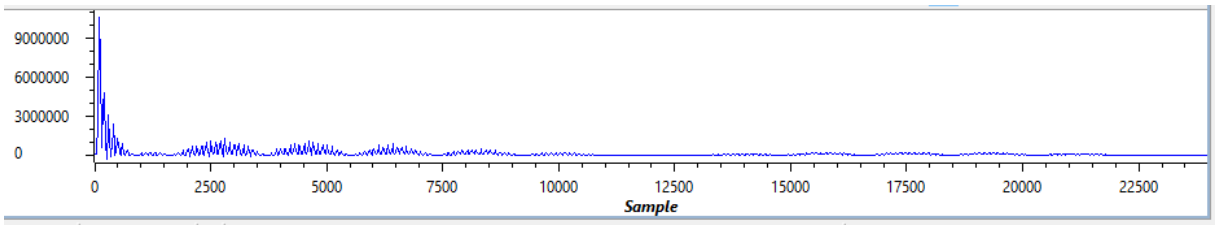
Wyk.10. Widmo sygnału piłokształtnego w skali logarytmicznej przed filtracją bez pierwszych 55 próbek (długość filtru)



Wyk.11. Widmo sygnału piłokształtnego w skali logarytmicznej po filtracji bez pierwszych 55 próbek (długość filtru)



Wyk.12. Widmo sygnału piłokształtnego w skali liniowej przed filtracją bez pierwszych 55 próbek (długość filtru)



Wyk.13. Widmo sygnału piłokształtnego w skali liniowej po filtracji bez pierwszych 55 próbek (długość filtru)

Sądząc po wykresach, filtracja wygląda na wykonaną prawidłowo. Wartości powyżej częstotliwości granicznej (1770 Hz) są silnie tłumione. Na wykresach czasowych widać, że z sygnałów pozbyto się ostrych krawędzi, za które odpowiedzialne są wysokie częstotliwości.

Na wykresach nie widać zniekształceń, ponieważ pierwsze 55 próbek sygnału zostały pominięte. Pominięto je, ponieważ obliczenie ostatnich 54 próbek wymagałoby użycia próbek, których nie znamy i przyjmujemy je za zera, co niekoniecznie musi być prawdą. Tak naprawdę, wystarczyłoby przesunąć wykresy jedynie o 54 próbki, ponieważ 55 próbka ma już właściwą wartość. Potrzebna jest taka ilość przyszłych próbek, jaki jest rząd filtra, czyli w tym wypadku 54. Wymaga tego równanie różnicowe filtru. Stąd w programie znalazła się instrukcja warunkowa „if(i-j<0) break;”, która przy okazji zapobiega odwoływania się do niedozwolonych miejsc w pamięci. Równie dobrze można by ją zamienić na

„if(i<55) break;”. Wtedy ostatnie próbki nie byłyby filtrowane wcale, zamiast filtrowania z zniekształceniami.

Użyto instrukcji akumulaty MAC, ponieważ jest ona zoptymalizowana pod procesor DSP. Instrukcje mnożenia są bardzo wymagające dla procesora.

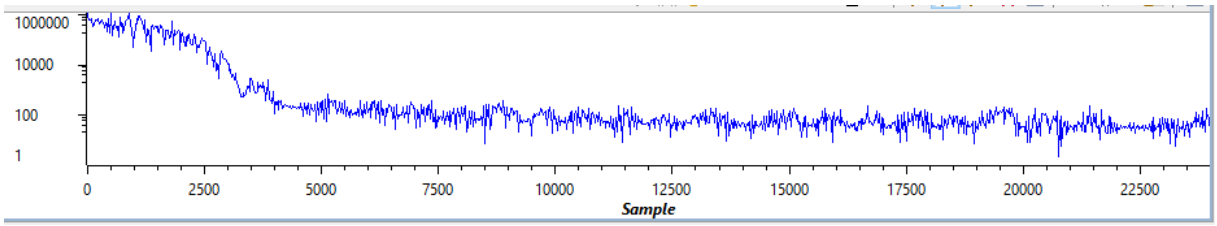
Jeżeli jako pierwszy i trzeci argument funkcji podamy wskaźnik do tej samej tablicy funkcja zostanie wykonana prawidłowo, ponieważ nadpisywane próbki nie są ponownie używane do obliczenia kolejnych próbek. Działanie sprawdzono i otrzymano takie same wyniki.

3. Filtracja blokowa sygnału za pomocą funkcji z DSPLIB

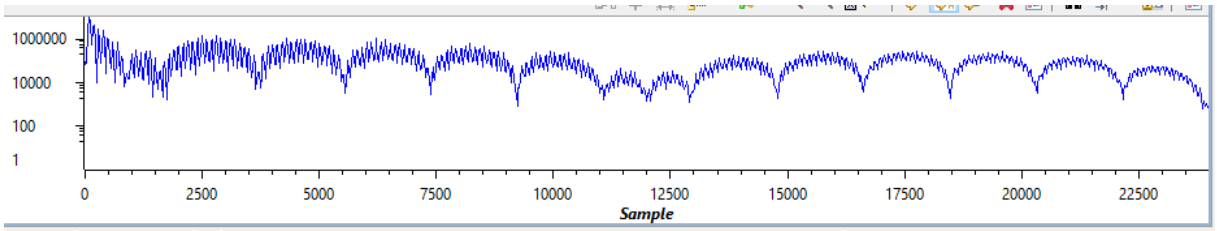
```
fir((DATA*)samples, (DATA*)coefficients, (DATA*)samples2, (DATA*)dbuffer, NUM_SAMPLES, NUM_COEF);
```

Rys.5. Kod wywołujący funkcję filtracji z biblioteki DSPLIB

Liczba operacji procesora potrzebnych do przefiltrowania tablicy 5000 próbek: **285 059**



Wyk.14. Widmo białego szumu w skali logarytmicznej po filtracji bez pierwszych 55 próbek (długość filtru) – funkcja z biblioteki DSPLIB



Wyk.15. Widmo sygnału piłokształtnego w skali logarytmicznej po filtracji bez pierwszych 55 próbek (długość filtru) – funkcja z biblioteki DSPLIB

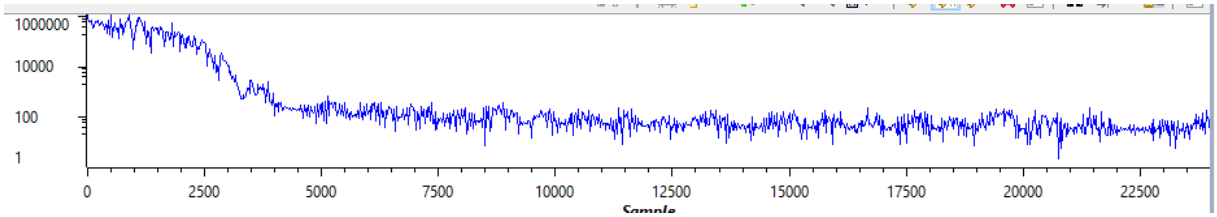
Wykresy wyglądają niemal identycznie jak te utworzone poprzez własną implementację filtru. Funkcja wykonała się za to znacznie szybciej, ponieważ jest lepiej zoptymalizowana i może zrównoleglić niektóre operacje.

4. Filtracja sygnału próbka po próbce za pomocą funkcji z DSPLIB

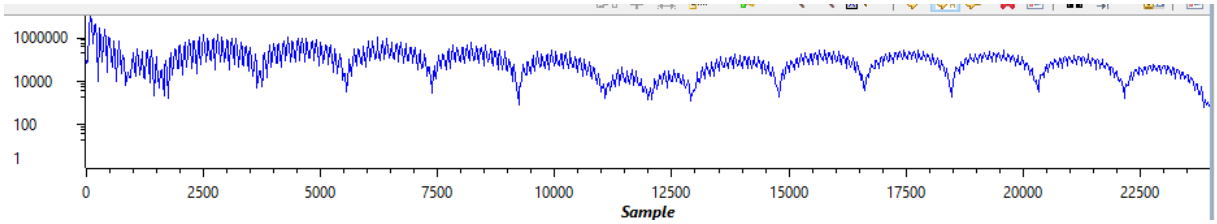
```
217     //zad 4
218     zeros(dbuffer, NUM_COEF+2);
219     testfun((int*)samples, NUM_SAMPLES);
220     int i;
221     for(i = 0; i < NUM_SAMPLES; i++)
222     {
223         fir(&samples[i], (DATA*)coefficients, &samples2[i], (DATA*)dbuffer, 1, NUM_COEF);
224     }
```

Rys.6. Kod wykonujący filtrację próbka po próbce

Liczba operacji procesora potrzebnych do przefiltrowania jednej próbki: **129**
Liczba operacji procesora potrzebnych do przefiltrowania tablicy 5000 próbek: **635 012**



Wyk.16. Widmo białego szumu w skali logarytmicznej po filtracji bez pierwszych 55 próbek (długość filtru) – funkcja z biblioteki DSPLIB wykonywana metodą „próbka po próbce”



Wyk.17. Widmo sygnału piłokształtnego w skali logarytmicznej po filtracji bez pierwszych 55 próbek (długość filtru) – funkcja z biblioteki DSPLIB wykonywana metodą „próbka po próbce”

Wyniki filtracji są identyczne z wynikami wykonanymi innymi metodami.

Zestawienie liczby potrzebnych cykli procesora dla poszczególnych metod filtracji

Własna implementacja funkcji – przetwarzanie blokowe: **9 176 626**

Funkcja z biblioteki DSPLIB – przetwarzanie blokowe: **285 059**

Funkcja z biblioteki DSPLIB – przetwarzanie próbka po próbce: **635 012**

Własna implementacja okazała się najbardziej kosztowna dla procesora pod względem liczby operacji. Funkcje z DSPLIB okazały się znacznie szybsze. Powodem tego jest lepsza ich optymalizacja przez twórców procesora i wykorzystanie buforów kołowych. Przetwarzanie próbka po próbce okazało się bardziej złożone od przetwarzania blokowego, ponieważ procesory DSP przystosowane są do przetwarzania większej ilości próbek na raz.

Zaleca się używanie przetwarzania blokowego wszędzie tam, gdzie opóźnienia nie są krytycznie ważnym elementem. Przetwarzanie próbka po próbce zaleca się używać w implementacjach przetwarzających próbki w czasie rzeczywistym, np. podczas filtrowania dźwięku mikrofonu, którego odsłuch jest podawany od razu na wyjście (nagrywanie z odsłuchem), czy transmisji na żywo.