Zastosowania Procesorów Sygnałowych
Raport z zadania projektowego nr 1
Generowanie Sygnałów na DSP
Mateusz Miler 171577
01.05.2020

Zadanie 1 – generowanie sygnału piłokształtnego

Obliczenia wartości kroku amplitudy

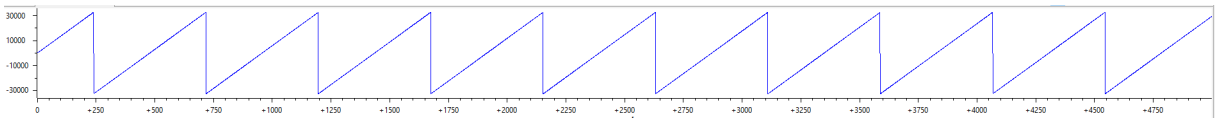
Krok = (2*32768)/(48000/f) ≈ (f*22368)/16384 = (f*22368)>>14

Krok = STEP_SAW = round((2*32768)/(48000/f)) = round(136,5333) = 137

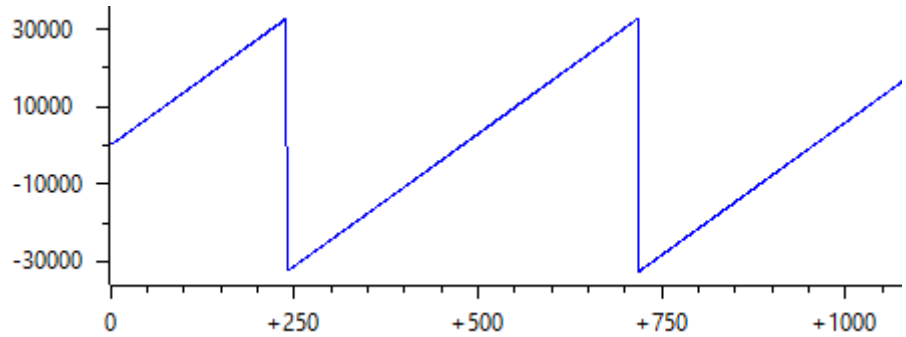
Kod funkcji saw

```
16 void saw(int* buffer, unsigned int buflen, int step)
17 {
18     static int accumulator = 0;
19     int i;
20     for(i = 0; i < buflen; i++)
21     {
22         buffer[i] = accumulator;
23         accumulator += step;
24     }
25 }
```

Wykres czasowy funkcji saw

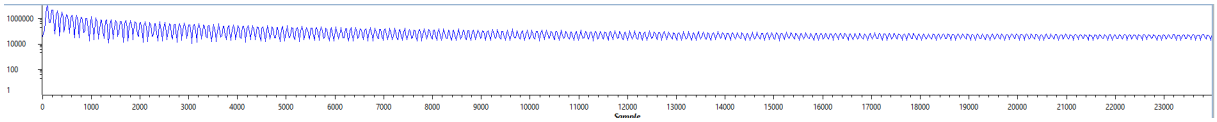


Fragment wykresu czasowego funkcji saw w powiększeniu

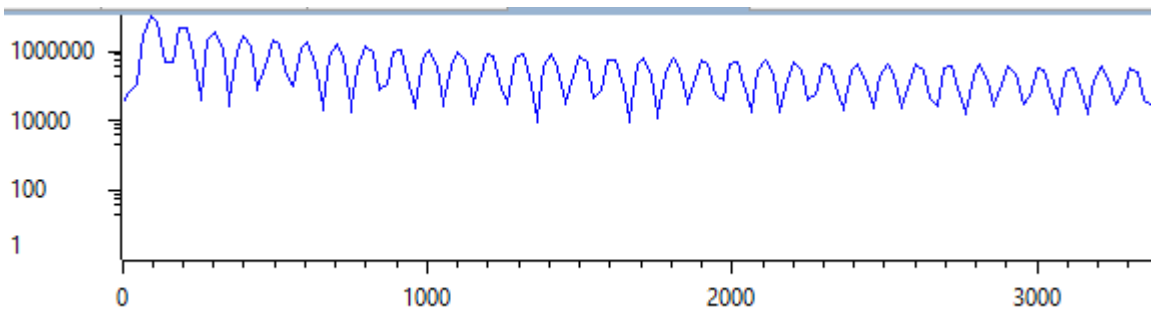


Wartości na wykresie zmieniają się pełnym zakresie od -32768 do 32767 w Q15. Przy częstotliwości sygnału 100 Hz jeden okres powinien trwać 1/100 sekundy oraz być reprezentowany przez około 480 próbek, co zgadza się z wykresem. Cały wykres składa się z 5000 próbek, zatem powinniśmy zauważyć 10,41 okresu podstawowego piły, co jest również zgodne z załączonym zdjęciem. Błąd związany z zaokrągleniem liczby 136,5(3) do 137 skutkuje z przyspieszeniem fazowym o około 17 próbek co każde 5000 próbek.

Wykres widma funkcji saw w skali logarytmicznej



Fragment widma funkcji saw w skali logarytmicznej w przybliżeniu



Wykres widmowy ma kształt grzebienia, podobnie jak to miało miejsce w przykładach prezentowanych na wykładzie.

Rzeczywista częstotliwość wygenerowanego sygnału

(2*32768*f)/(48000) = 137

f = (137*48000)/(2*32768) = 100,3417969 Hz

Rzeczywista częstotliwość wygenerowanego sygnału wynosi 100,3417969 Hz. Różnica wynika z precyzji zapisu liczbowego w komputerze. Większą precyzję można uzyskać poświęcając więcej pamięci na zapis liczby. Nie możemy uzyskać częstotliwości dokładnie 100 Hz, ponieważ liczba 136,5(3) nie ma skończonego zapisu dziesiętnego.

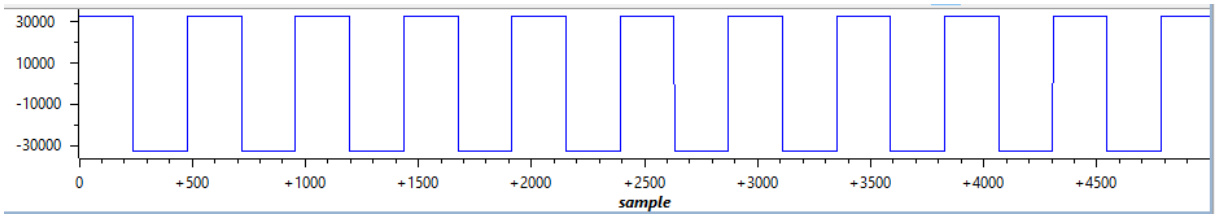
Odwróconą piłę można by wygenerować odejmując krok zamiast go dodawać do akumulatora w funkcji saw.

Zadanie 2 – generowanie sygnału prostokątnego

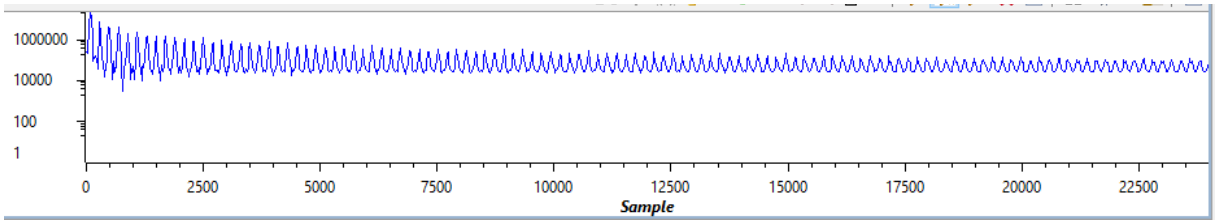
Kod funkcji rect

```
27 void rect(int* buffer, unsigned int buflen, int step, int threshold)
28 {
29     saw(buffer, buflen, step);
30     int i;
31     for(i = 0; i < buflen; i++)
32     {
33         if (buffer[i] < threshold)
34             buffer[i] = -32768;
35         else
36             buffer[i] = 32767;
37     }
38 }
```

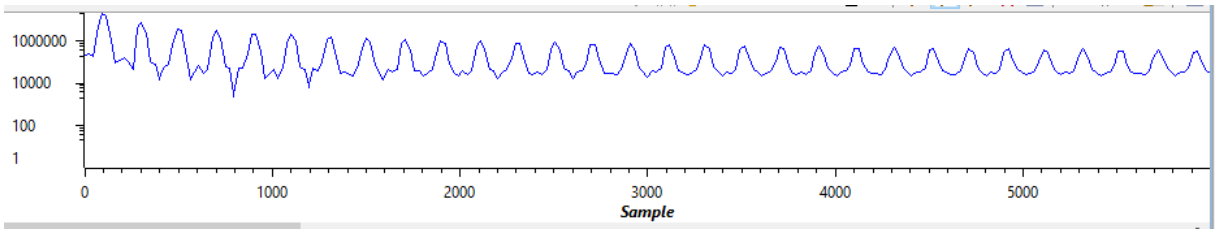
Wykres czasowy funkcji rect dla progu 0



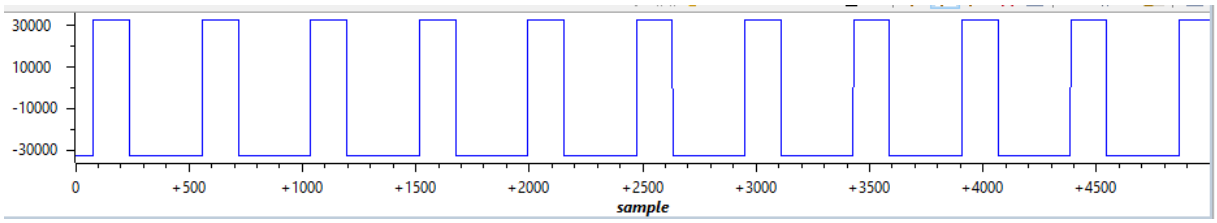
Widmo funkcji rect w skali logarytmicznej dla progu 0



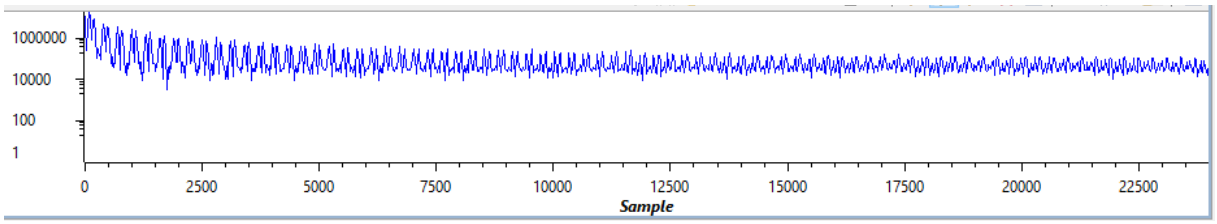
Fragment widma funkcji rect w skali logarytmicznej dla progu 0 w powiększeniu



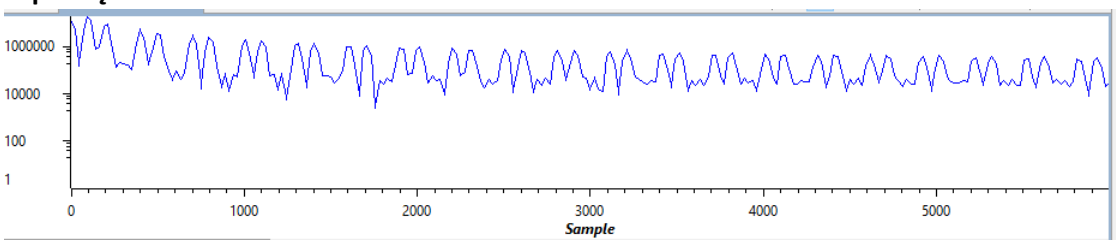
Wykres czasowy funkcji rect dla progu 10923 (wypełnienie 1/3)



Widmo funkcji rect w skali logarytmicznej dla progu 10923 (wypełnienie 1/3)



Fragment widma funkcji rect w skali logarytmicznej dla progu 10923 (wypełnienie 1/3) w powiększeniu



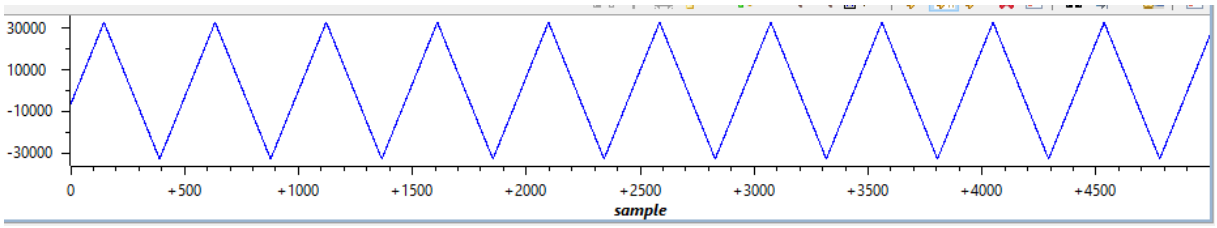
Wartości na wykresie zmieniają się pełnym zakresie od -32768 do 32767 w Q15. Podobnie jak w poprzednim zadaniu, na jeden okres przypada 480 próbek (ta sama częstotliwość) – zgadza się. Na wykresach czasowych widać, że wartości progowe zostały właściwie dobrane dla oczekiwanych poziomów wypełnienia. Wykresy widmowe wyglądają bardzo podobnie i mają kształt grzebieni, jednak o rzadszych ząbkach niż w przypadku sygnału piłokształtnego. Widmo sygnału o wypełnieniu 1/3 charakteryzuje się „podwójnymi ząbkami".

Zadanie 3 – generowanie sygnału trójkątnego

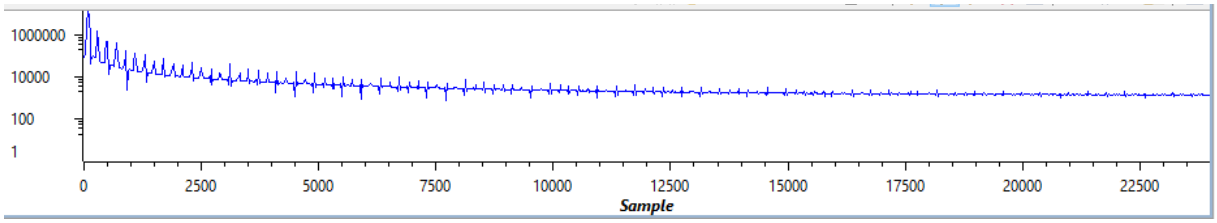
Kod funkcji tri

```
40 void tri(int* buffer, unsigned int buflen, int step)
41 {
42     saw(buffer, buflen, step);
43     int i;
44     int temp;
45     for(i = 0; i < buflen; i++)
46     {
47         temp = buffer[i] < 0 ? -buffer[i] : buffer[i];
48         buffer[i] = (temp - 16384)<<1;
49     }
50 }
```

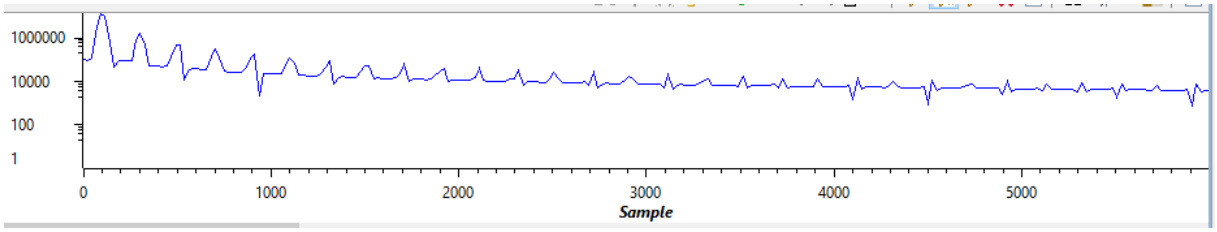
Wykres czasowy funkcji tri



Widmo funkcji tri w skali logarytmicznej



Fragment widma funkcji tri w skali logarytmicznej w powiększeniu



Wartości na wykresie zmieniają się pełnym zakresie od -32768 do 32767 w Q15. Wyciągniecie mnożenia przez 2 przed nawias w równaniu sygnału trójkątnego pozwoliło uniknąć przepełnienia. Przebieg czasowy jest zgodny z oczekiwaniami. Podobnie jak wcześniej okres składa się z 480 próbek. Widmo ma kształt przerzedzonego grzebienia. Jest mniej poszarpane od widma sygnału

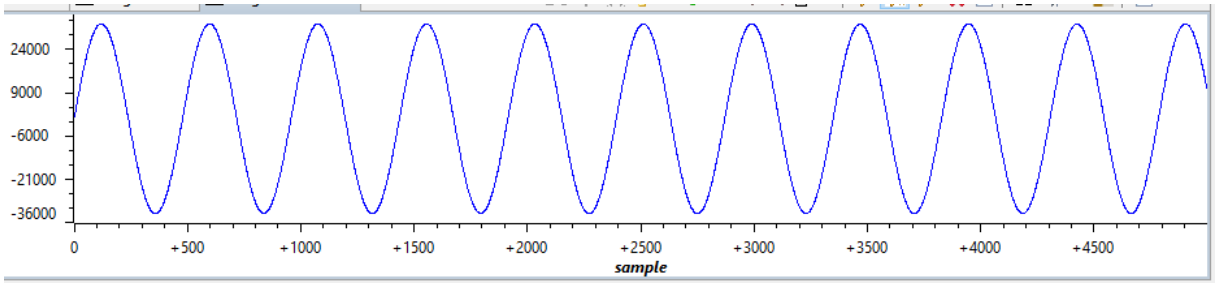
prostokątnego. Występują w nim charakterystyczne „wypłaszczenia” przypominające wskazania medycznego elektrokardiogramu.

Zadanie 4 – generowanie sygnału sinus za pomocą szeregu Taylora

Kod funkcji sint

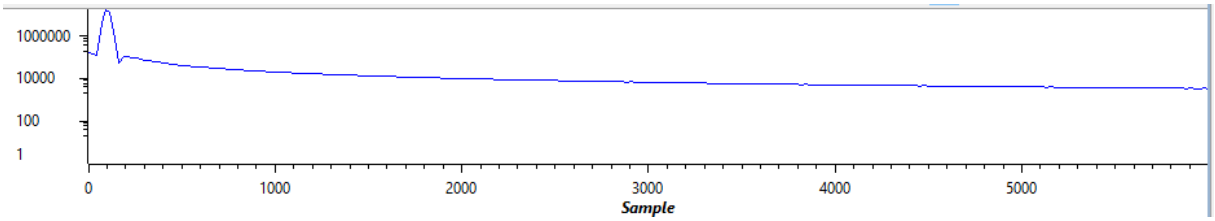
```
53 void sint(int* buffer, unsigned int buflen, int step)
54 {
55     //mnożnik pi, wartoci <-1;1) - przemnożone przez pi daje faze
56     saw(buffer, buflen, step);
57     //stałe współczynniki ciagu Taylora przmnożone przez odpowiednie potegi pi
58     long a1 = 12868;
59     long a3 = -21167;
60     long a5 = 10445;
61     long a7 = -2455;
62     //argumenty do potegi 1, 2, 3, 5, 7
63     int x1, x2, x3, x5, x7;
64     long taylor;
65 
66     int i;
67     int temp;
68     int isPositive;
69     for(i = 0; i < buflen; i++)
70     {
71         //faza trójkatna: 0 ... 0,5 ... 0 ... 0,5
72         temp = buffer[i];
73         if (temp >= 0) isPositive = 1;
74         else isPositive = 0;
75         temp = temp < 0 ? -temp : temp;
76         if(temp > 16384) temp = 32767 - temp;
77 
78         //argumenty do potegi 1, 2, 3, 5, 7
79         x1 = temp;
80         x2 = _smpy(x1, x1);
81         x3 = _smpy(x2, x1);
82         x5 = _smpy(x3, x2);
83         x7 = _smpy(x5, x2);
84 
85         //z szeregu Taylora
86         taylor = a1*x1 + a3*x3 + a5*x5+ a7*x7;
87         buffer[i] = (int)((taylor + (1<<11))>>12);
88         if (isPositive == 0) buffer[i] = -buffer[i];
89     }
90 }
```

Wykres czasowy funkcji sint

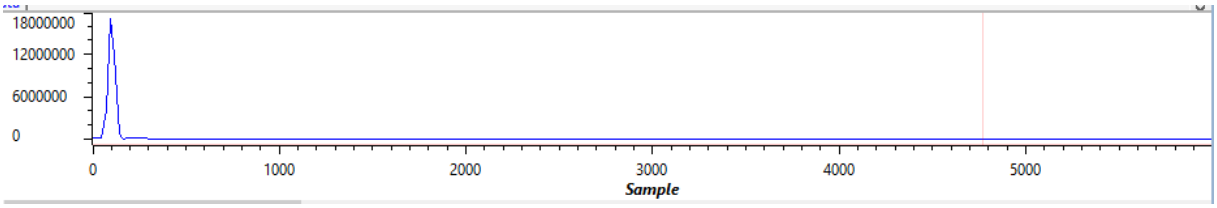


Wartości na wykresie czasowym wyglądają na odwzorowane prawidłowo. Zgadza się okres oraz zbiór wartości.

Widmo funkcji sint w skali logarytmicznej

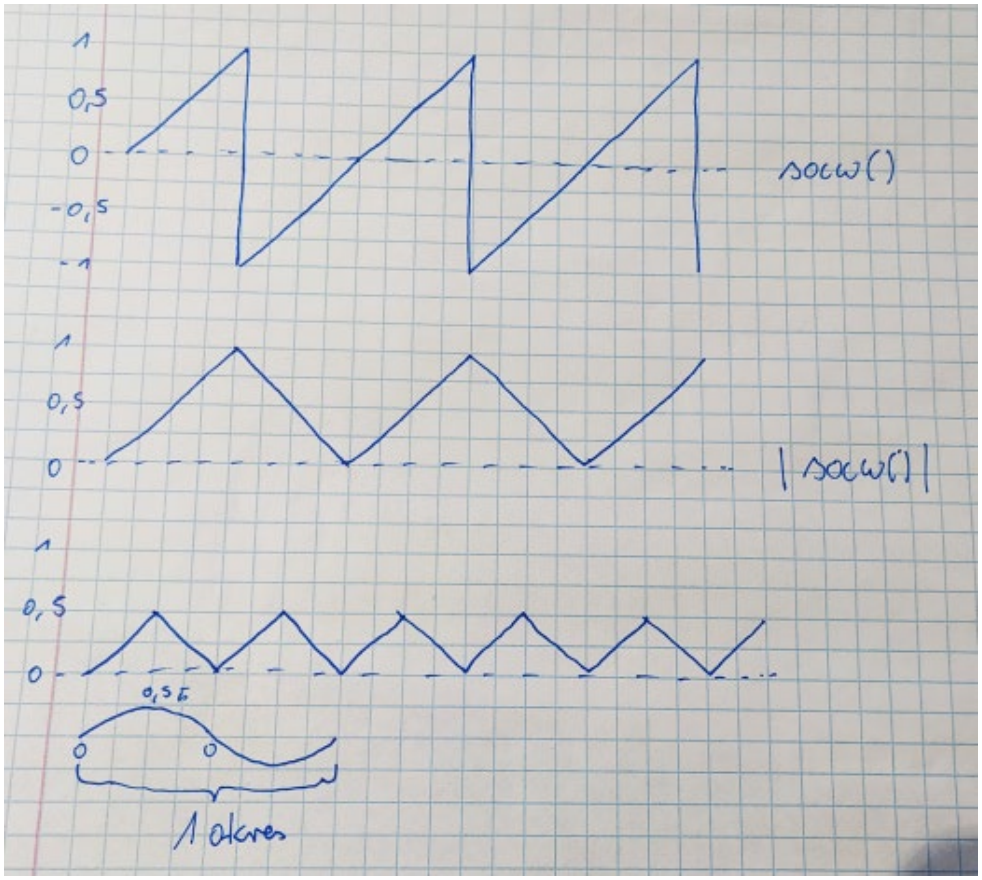


Widmo funkcji sint w skali liniowej



Wykres widmowy zawiera jeden prążek na częstotliwości około 100 Hz. Prążek nie jest idealną deltą Diraca, tylko „rozpływa" się na sąsiadujące częstotliwości. Mimo wszystko jest on dosyć bliski ideału, co świadczy o bardzo niewielkich zniekształceniach.

Przekształcenia wykorzystywanej w obliczeniach fazy



Przedstawiona implementacja wymagała **1 019 613 cykli zegarowych** do wygenerowania 5000 próbek.

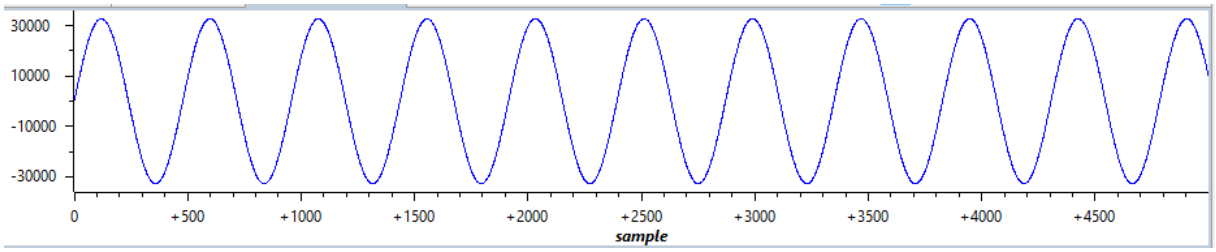
Uwzględnianie dalszych składników szeregu Taylora nie jest zasadne, ponieważ składniki te są pomijalnie małe przez wysoką silnię w mianowniku oraz podnoszenie do wysokiej potęgi licznika, będącego liczbą mniejszą lub równą 0,5. Zatem użycie kolejnych składników nie poprawiłoby sensownie dokładności wyników, za to zdecydowanie wydłużyłoby ilość potrzebnych do wykonania operacji (skomplikowane dla procesora mnożenia), a więc i czas wykonywania programu, na którym bardzo często szczególnie nam zależy w przypadku procesorów sygnałowych.

Zadanie 5 – generowanie sygnału sinus za pomocą funkcji z DSPLIB

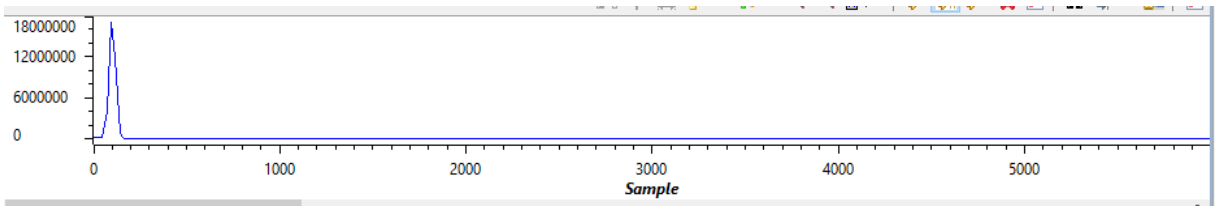
Wywołanie funkcji sine

```
111     //zad5
112     saw(samples, NUM_SAMPLES, STEP_SAW);
113     sine((DATA*)samples, (DATA*)samples, NUM_SAMPLES);
```

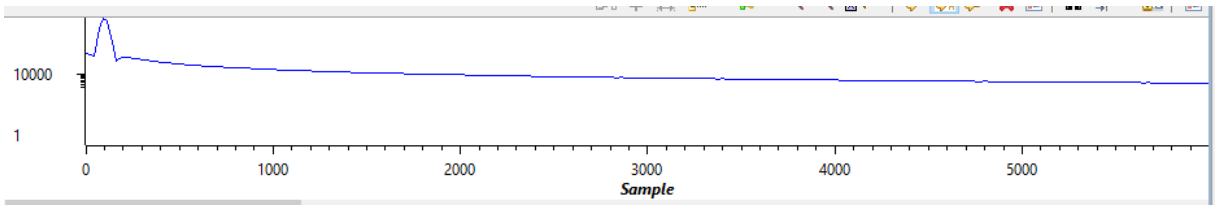
Wykres czasowy funkcji sine



Widmo funkcji sine w skali liniowej



Widmo funkcji sine w skali logarytmicznej



Wykres czasowy funkcji sine wygląda tak samo, jak wykres funkcji zaimplementowanej samodzielnie. Wykres widmowy jest bardzo podobny do tych z samodzielnej implementacji.

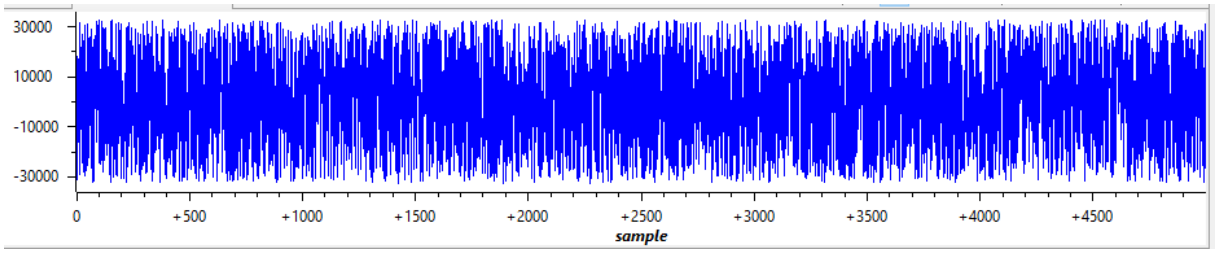
Funkcja sine z biblioteki DSPLIB wymagała **55 038 cykli zegarowych** do wykonania. Razem z przygotowaniem sygnału piłokształtnego było to **165 072 cykli zegarowych** do wygenerowania 5000 próbek. Dzieje się tak, ponieważ funkcja z DSPLIB jest zoptymalizowana pod używany procesor i część działań może zostać zrównoleglonych.

Zadanie 6 – generowanie białego szumu

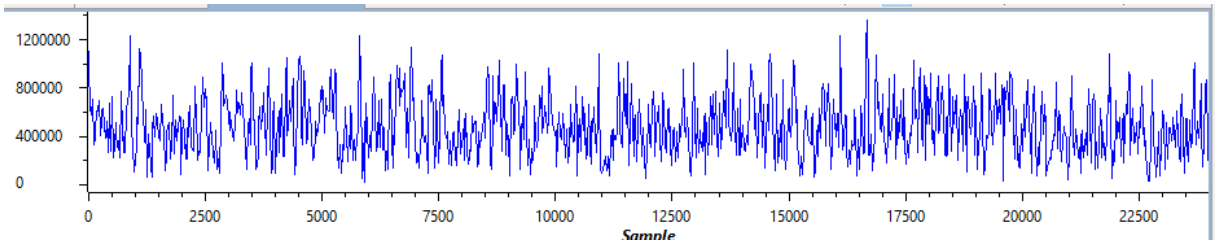
Kod programu

```
114     //zad 6
115     rand16init();
116     rand16((DATA*)samples, NUM_SAMPLES);
```

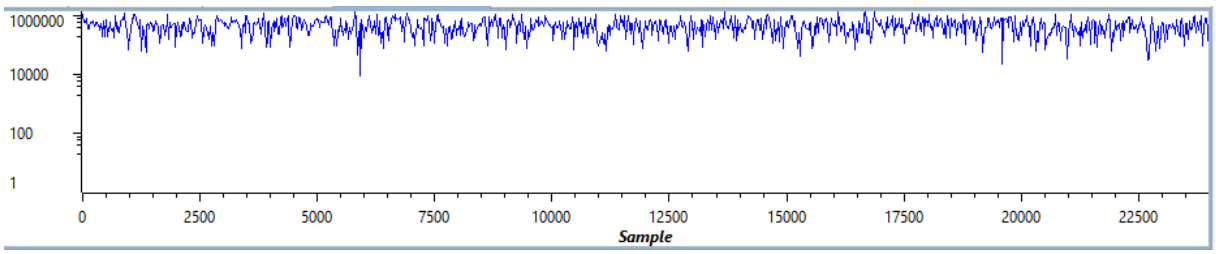
Wykres czasowy funkcji rand16



Widmo funkcji rand16 w skali liniowej



Widmo funkcji rand16 w skali logarytmicznej



Wykres czasowy przyjmuje losowe wartości. Widmo zgodnie z zamierzeniami jest dosyć płaskie. Szczególnie dobrze widać to na wykresie w skali logarytmicznej. Świadczy to równomiernym rozłożeniu mocy na osi częstotliwości.