

# PROCESORY SYGNAŁOWE

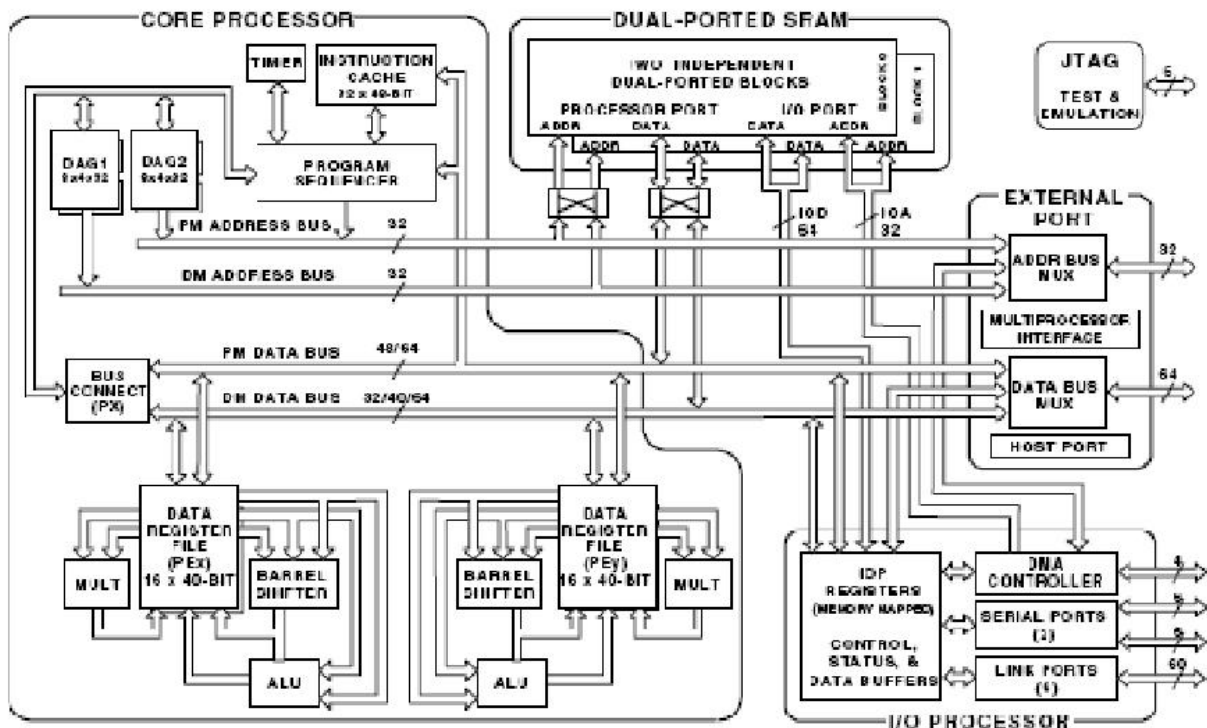
## Laboratorium

### 1. Wstęp

Poniższe ćwiczenie przedstawia możliwości wykorzystania zestawu laboratoryjnego SHARC EZ-KIT Lite wyposażonego w procesor sygnałowy ADSP-21161 ShARC firmy Analog Devices.

Procesor sygnałowy ADSP-21161 SHARC jest wydajnym 32-bitowym procesorem przeznaczonym do zastosowań w dziedzinie przetwarzania sygnałów akustycznych oraz wizyjnych.

Jego schemat blokowy przedstawia rys.:



W konstrukcji tego procesora wyróżnić można kilka głównych bloków:

- Core Procesor – rdzeń systemu, jednostka odpowiedzialna za wykonywanie programu i wszystkie operacje obliczeniowe.
- Dual-Ported SRAM – dwuportowa pamięć RAM podzielona na dwa bloki.
- I/O Procesor – procesor wyjścia-wejścia odpowiedzialny za niezależną od jednostki obliczeniowej wymianę danych z otoczeniem.
- External Port – moduł odpowiedzialny za komunikację procesora z innymi procesorami w systemach wieloprocessorowych lub za komunikację z procesorem nadrzędnym.
- JTAG – port umożliwiający podłączenie emulatora EZ-ICE.

Wykonane ćwiczenia pozwalają przetestować część tych bloków.

## 1.1 Aproksymacja funkcji sinus

Istnieje wiele sposobów aproksymacji sinusa, wśród nich aproksymacja wielomianem zgodnie ze wzorem Taylora:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \dots$$

Stosując ten wzór otrzymaliśmy funkcję prawie idealnie odwzorowującą sinusa. Stosowany procesor sygnałowy, ze względu na swoje ograniczenia – stałooprzecinkowość, zniekształcał nieznacznie wyniki.

W programie VisualDSP możliwe jest również generowanie sinusa stosując funkcję *sin*.

## 1.2 Generator liczb pseudolosowych

**Generator liczb pseudolosowych** (*Pseudo-Random Number Generator*, lub **PRNG**) to program lub podprogram, który na podstawie niewielkiej ilości informacji (tzw. *seed*) generuje deterministycznie potencjalnie nieskończony ciąg bitów, który pod pewnymi względami jest nieodróżnialny od ciągu uzyskanego z prawdziwie losowego źródła.

Generatory liczb pseudolosowych nie generują całkiem losowych ciągów – jeśli generator jako seed bierze  $k$  bitów informacji, to może wygenerować  $n$ -bitowy ciąg jedynie na  $2^k$  sposobów spośród  $2^n$  możliwych.

Do bardzo wielu zastosowań taka pseudolosowość zupełnie wystarcza – w grach komputerowych, obliczeniach probabilistycznych (takich jak np. całkowanie Monte Carlo) potrzebujemy jedynie liczb zachowujących się *mniej więcej* jak liczby losowe. W każdym przypadku użycia nowego generatora do celów obliczeń numerycznych należy sprawdzić jego własności (bezpośrednio udowodnić, lub znaleźć odpowiednie opracowania), ponieważ jeżeli generator jest *niewystarczająco losowy* to obliczenia z jego użyciem mogą znacznie odbiegać od rzeczywistych wyników uzyskanych innymi metodami.

Przykładowy prosty generator:

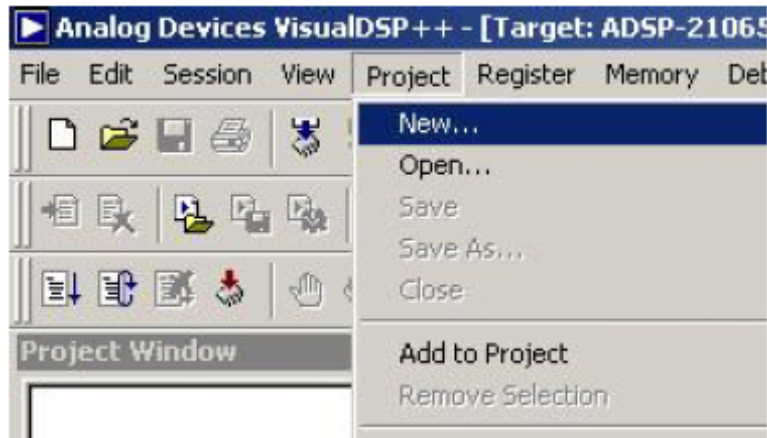
nowy stan =  $a \times \text{stary stan} + b \pmod{c}$

wygenerowany bit = nowy stan  $\pmod{2}$

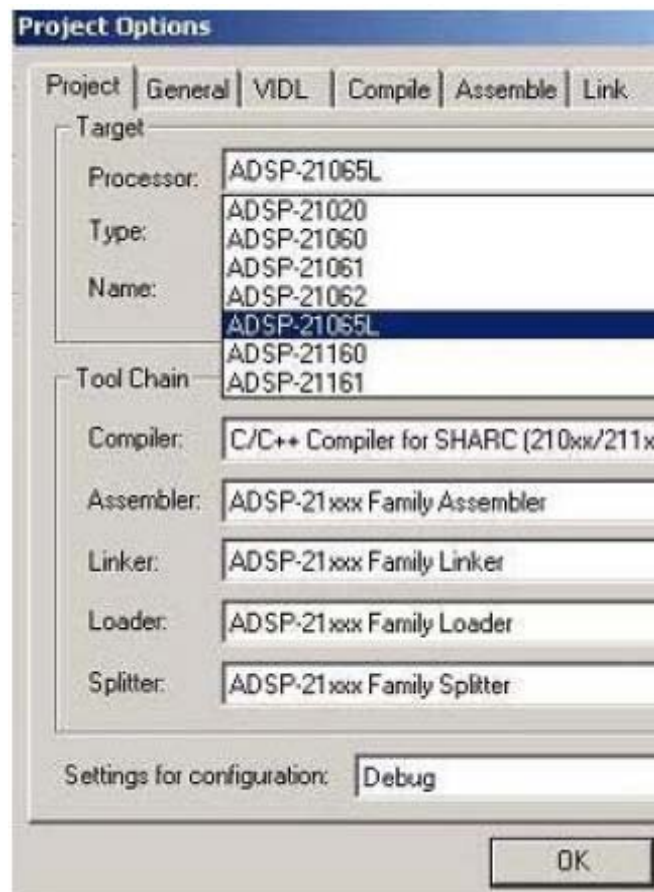
## 2. Opis programu VisualDSP

Poniżej znajduje się instrukcja obsługi programu VisualDSP

### 2.1 Otwarcie nowego projektu **Project -> New**



### 2.2 Następnie w zakładce **Project Options** wybieramy rodzaj procesora:



### 2.3 Następnie w zakładkach **Compile**, **Assemble** i **Link** zaznaczamy opcję: „Generate debug information”.

### 2.4 Aby projekt był gotowy do pracy należy jeszcze dołączyć pliki \*.ldf do **Linker Files**.

2.5 Z paska wybieramy **File->New** i tworzymy nowy plik z rozszerzeniem **\*.c**.

2.6 Następnie dodajemy utworzony plik do projektu „**Add files to folder**”.

2.7 Aby skompilować/zasymulować program należy z menu wybrać :  
**Project/Rebuild All**.

### 3. Przykładowe programy

Poniżej znajduje się listing kodu programu generującego histogram oraz wykres sinusa:

#### 3.1. Generacja histogramu

```
#define PI 3.141592654
float a = PI/180;

void main()
{
    int i, k, M, p, lw, a;
    int N=12;    /*liczba probek zmiennej losowej*/
    float roz[12]={0}; /*tablica w której są gromadzone wartości poszczególnych próbek
    histogramu*/
    int li[N];

    /*liczba podprzedzialow histogramu*/
    M=12;

    /*generacja tablicy liczb pseudolosowych
    wedlug wzoru li(n)=(li(n-1)+p)mod 1000*/
    p=500;    /*pierwszy element tablicy*/
    lw=587;    /*liczba dodawana w algorytmie*/
    a=7;    /*mnoznik liczby poprzedniej*/
    li[0]=p;
    for( i=1; i<N; i++ )
    {
        li[i]=(a*li[i-1]+lw)%1000;
    }
    /*wyznaczanie histogramu*/
    for (i=0; i<N; i++)
    {
        k=(li[i]*M/1000);    /*indeks podprzedzialu*/
        roz[k]=roz[k]+1;
    }
    for( i=0; i<M; i++ )
    {
        roz[i]=roz[i]*M/(N*1000);
    }

    /*wydruk histogramu*/
    for( i=0; i<M; i++ )
    {
        printf( "histogram [%d] = %f\n", i, roz[i] );
    }
    /* wydruk adresu tablicy "roz" */
    printf( "%x\n", roz );
}
```

**3.2. Generacja sinusa przy pomocy wzoru Taylora**

```

float dm s[360];
float r, r2;
float sinus(void);
#define Kat 180
#define PI 3.141592654

void main()
{
    sinus();
}

float sinus(void)
{
    int g; //wartość bieżąca kąta
    g=0;
    for(g = 0; g < Kat; g++)
    {
        r = PI * g / 180; r2=r*r;
        s[g] = (1.0-r2*(0.1666666666
                    -r2*(0.00833333333
                    -r2*(0.000198413
                    -r2*(0.000002756
                    -0.00000002505*r2)))))*r;

        printf( "sin [%d] = %f\n", g, s[g]);
    }
}

```

Wyświetla pierwszą połówkę okresu sinusa od 0-180°. Zastosowane współczynniki są wyliczone na podstawie wzoru Taylora. Element tablicy S jest wartością próbki dla odpowiedniego kąta, i tak np. S[3] jest wartością sinusa dla kąta 3°.

```

s[g+180] = -(1.0-r2*(0.1666666666
                    -r2*(0.00833333333
                    -r2*(0.000198413
                    -r2*(0.000002756
                    -0.00000002505*r2)))))*r;

```

Dla kąta powyżej 180° generowana tablica jest taka sama jak dla przedziału 0-180° z tym, że poprzedzona znakiem „-”, czyli np. dla g=3 w tablicy zapisywana jest wartość s[3] i s[183] = - s[3].

Powoduje to, że podczas jednej pętli instrukcji for w tablicy zapisujemy dwie wartości dla kąta x i 180+x.

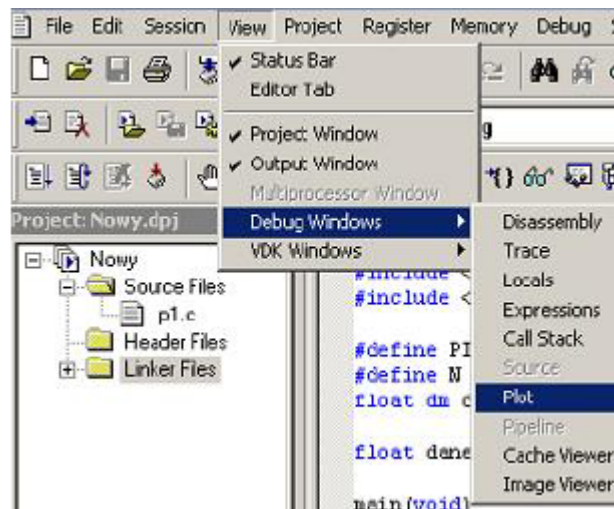
```

        printf( "sin [%d] = %f\n", g+180, s[g+180]); //wyświetla drugą połówkę okresu sinusa
    }
    return(0);
}

```

## 3.3. Wykreślanie wykresów

Aby wyświetlić wykres należy wybrać **View -> DebugWindows -> Plot -> New...**



Pojawi się okno **Plot Configuration**



W tym okienku interesują nas takie opcje:

**Title:** Nazwa jaka nadamy naszemu wykresowi – a będzie to wykres przedstawiający wartości kolejnych zmiennych tablicy s

**Memory:** Ustawiamy w pozycji DM

**Address:** I w tym właśnie miejscu spójrz jeszcze raz na kod naszego programu! Zwróć szczególną uwagę na literki dm które znajdują się przed deklaracją zmiennej s:  
float dm s[360];

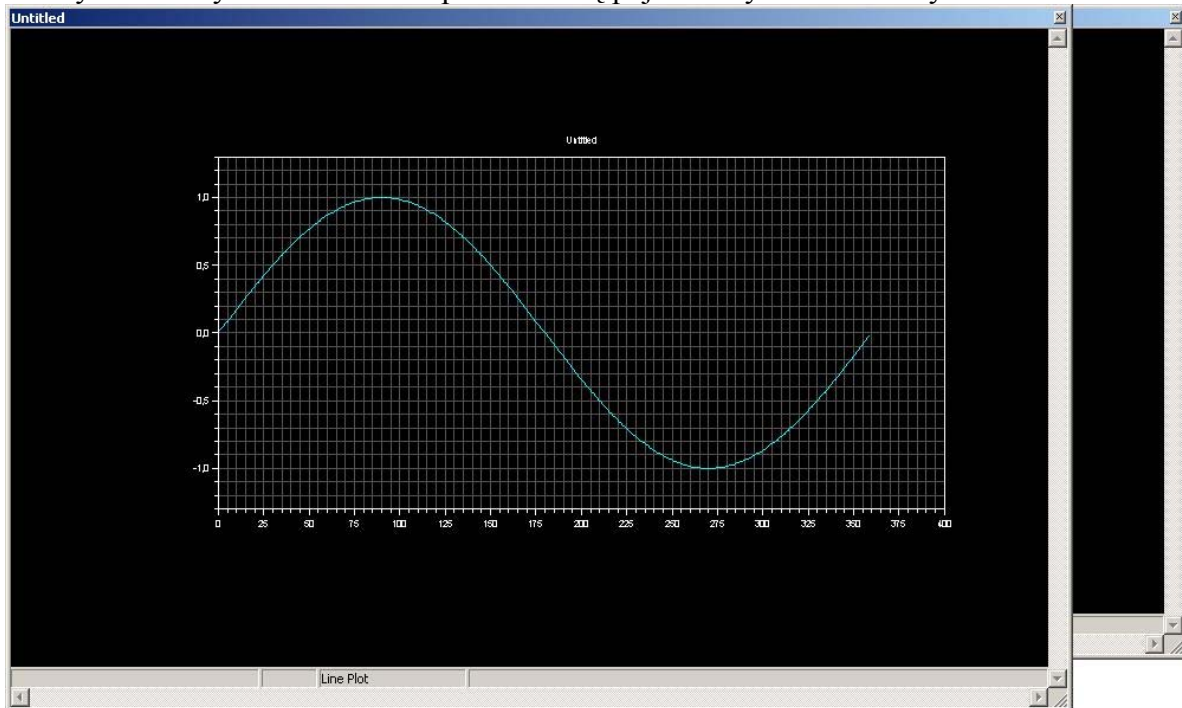
Dzięki temu nasza zmienną s możemy znaleźć poprzez **Browse...**

dm – Data Memory

**Count:** to liczba punktów wyświetlanych na osi

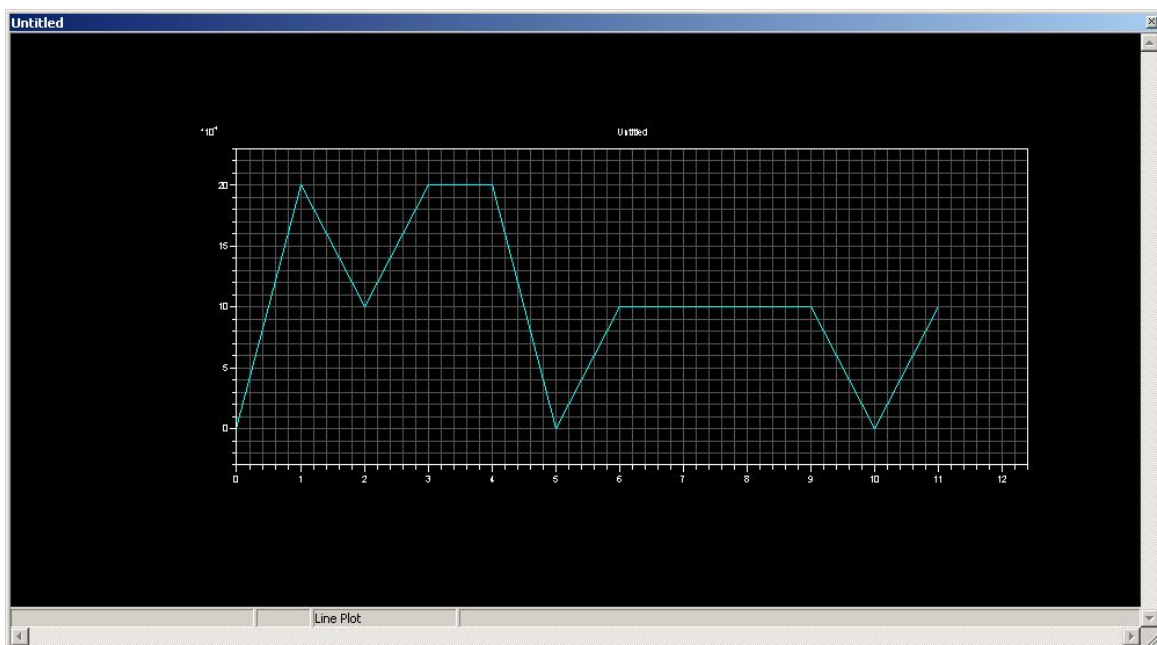
**Data:** to typ zmiennej, w naszym przypadku jest to zmienna float

Po czym klikamy **Add** oraz **OK** i powinien się pojawić wykres sinusoidy



Wykres 1. Sinus

Wykres histogramu generujemy analogicznie. Pamiętając o wpisaniu odpowiedniego adresu (wyświetla się on w linii komend programu zaraz pod wartościami histogramu).



Wykres 2. Histogram

**4. Zadania do wykonania.****4.1. Wyświetlić funkcję sinus za pomocą funkcji sin();**

Kod próbkuje sygnał sinusoidalny a wartości próbek wpisuje do tablicy d[N]

```
// Program p1 do próbkowania sygnału sinusa
#include <stdio.h>
#include <math.h>

#define PI 3.141592654
#define Q 36;
float dm f_sin[Q];
float dane_wejscowe (void);
main(void)
{
    dane_wejscowe();
    return(0);
}
float dane_wejscowe(void)
{
    int i;
    for (i=0; i<Q; i++)
    {
        f_sin[i]=sin(2*PI*i/Q);
        printf( "sin [%d] = %f\n", i, f_sin[i]); //Wyświetlenie 36 próbek sinusa
    }
    return(0);
}
```

**4.2. Napisać program wyświetlający określoną ilość okresów sinusa.**

```
#define PI 3.141592654
#define Kat 180
const float a = 3.141592654/180;
float s[360];
float r, r2;
float sinus(void);

void main()
{
    sinus();
    return (0);
}

float sinus(void)
{
    int g;
    g=0;
    int k;
    k=1;

    for (k = 1; k < 3; k++)
    for(g = 0; g < Kat; g++)
    {
        r = PI * g / 180; r2=r*r;
        s[k*g] = (1.0-r2*(0.166666666666
                        -r2*(0.008333333333
                        -r2*(0.000198413
                        -r2*(0.000002756
```



## Laboratorium DSP

Maciej Miskurka, Maciej Rogalski, Patryk Tomczewski

```
-0.00000002505*r2)))))*r;

printf( "sin [%d] = %f\n", k*g, s[k*g]);

s[k*g+180] = -(1.0-r2*(0.1666666666
-r2*(0.00833333333
-r2*(0.000198413
-r2*(0.000002756
-0.00000002505*r2)))))*r;

printf( "sin [%d] = %f\n", k*g+180, s[k*g+180]);

}
return(0);
}
```

### 4.3. Porównać wartości próbek sinusa uzyskanego funkcją sin(), a szeregiem Taylora.

```
// Program do porównania sygnału sinusa
float r, r2;
float sinus(void);
#define Kat 180
#define PI 3.141592654
#define Q 360;
float dm f_sin[Q];
float dm s[360];

float porównanie(void);
main(void)
{
    porownanie();
    return(0);
}

float porownanie(void)
{
    int comp;
    for (comp=0; comp<Q; comp++)
    {
        r = PI * comp / 180; r2=r*r;
        s[comp] = (1.0-r2*(0.1666666666-r2*(0.00833333333-r2*(0.000198413-
        r2*(0.000002756-0.00000002505*r2)))))*r;
        s[comp+180] = -(1.0-r2*(0.1666666666-r2*(0.00833333333-r2*(0.000198413-
        r2*(0.000002756-0.00000002505*r2)))))*r;

        f_sin[comp]=sin(2*PI*comp/Q);

        roznica[comp]=f_sin[comp]-s[comp];
        printf( "roznica [%d] = %f\n", comp, roznica[comp]); //Wyświetlenie 360 próbek sinusa
        return(0);
    }
}
```

### 4.4. Pliki headerowe

W celu poprawnego uruchomienia przykładowych programów na początku każdego z plików należy dodać deklarację plików headerowych.

```
#include "ADDS_21161_EzKit.h" //wszystkie rejestry definiowane
#include <def21161.h>
#include <signal.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
```