KATEDRA
SYSTEMÓW
MIKROELEKTRONICZNYCH

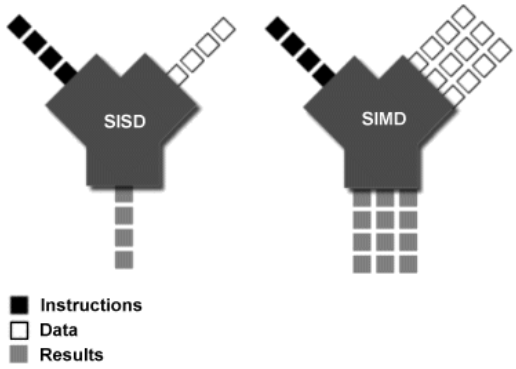
Laboratorium zastosowań procesorów sygnałowych

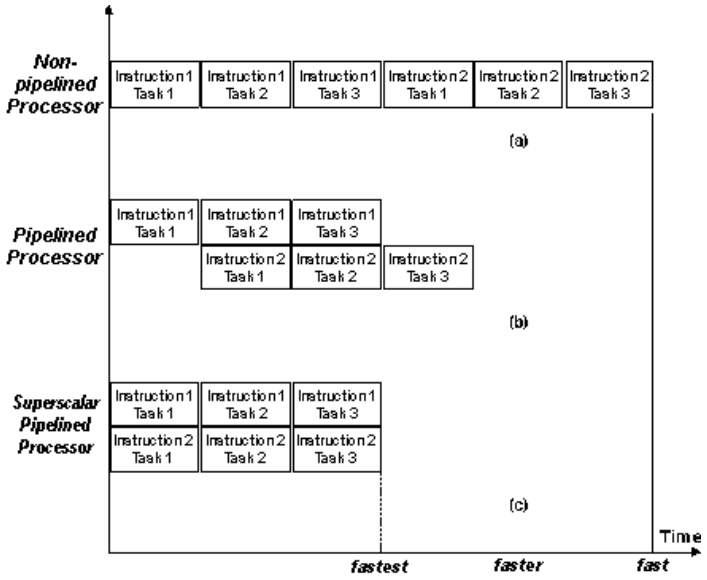
Wykorzystanie architektury SIMD.

Wydziału Elektroniki, Telekomunikacji i Informatyki
Politechniki Gdańskiej

Wiadomości wstępne

Jednym z trendów na rynku procesorów sygnałowych jest dążenie do zwiększenia proporcji między mocą obliczeniową procesorów a ich częstotliwością taktowania. Możliwość zmniejszenia częstotliwości pracy układu przy zachowaniu jego funkcjonalności niesie takie zalety jak zmniejszenie poboru mocy, możliwość obniżenia napięcia zasilającego czy uproszczenie layoutu samego układu scalonego poprzez eliminację części buforów. Dwie podstawowe metody zrównoleglenia operacji w procesorach to wykorzystanie instrukcji SIMD oraz zwielokrotniony pipelining. Technologia SIMD pozwala na zwiększenie ilości danych przetwarzanych przez pojedynczą daną, podczas gdy zwielokrotniony łańcuch pipeliningu pozwala wykonywać więcej niż jedną instrukcję w tym samym czasie (architektura nazywana superskalarną).





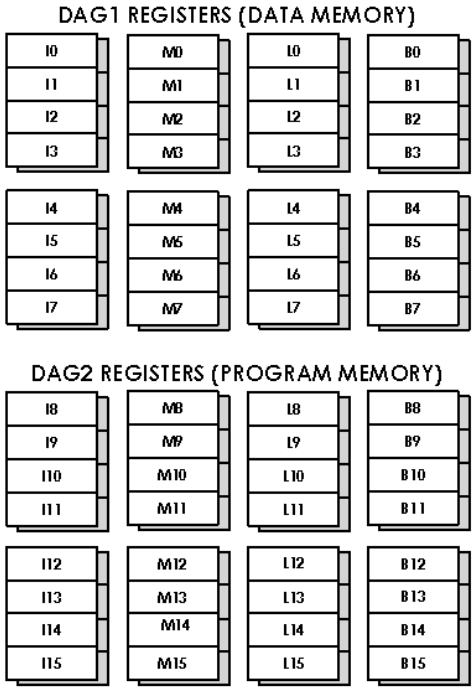
Zaletą superskalarności jest to, że wykonywane równocześnie instrukcje mogą się od siebie różnić. Problemem jest to, że muszą zostać odpowiednio sparowane aby nie następowała niepotrzebna blokada zasobów procesora i cykle oczekiwania, co utrudnia znacznie optymalne wykorzystanie możliwości. Procesory sygnałowe wykorzystywać mogą oba przedstawione mechanizmy. Wykorzystanie instrukcji SIMD wydaje się być ułatwione przy typowych dla nich operacjach jak np. filtracja sygnałów.

Układ ADSP-21161 jest jednym z procesorów sygnałowych rodziny SHARC (Super Harvard Architecture) produkcji Analog Devices wyposażonych w rdzeń w architekturze SIMD. Zawiera dwie jednostki obliczeniowe (Processing Elements) zdolne do pracy jako pojedyncza jednostka SIMD. Każda z jednostek obliczeniowych, nazwanych PEX i PEY, zawiera ALU, układ mnożący, układ przesuwający i blok rejestrów (PEX: R0-R15, PEY: S0-S15). Jednostka PEX jest zawsze aktywna, w celu uaktywnienia jednostki PEY należy ustawić bit PEYEN w rejestrze MODE1.

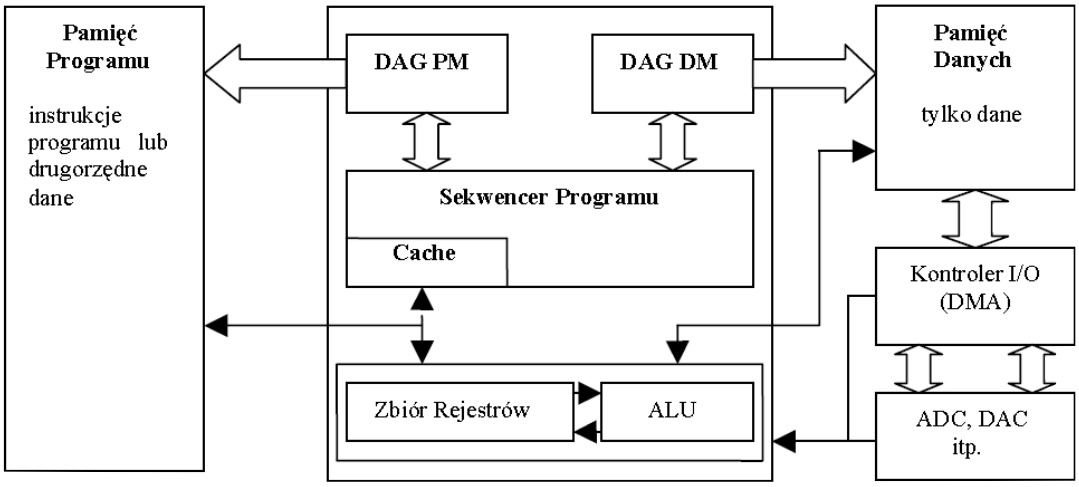
Po przejściu w tryb SIMD obie jednostki przetwarzające wykonują jednakowe instrukcje, jednak operują na różnych danych. Tryb SIMD wymaga zatem przyśpieszenia transferu danych między pamięcią a elementami przetwarzającymi. Z użyciem DAGs (Data Access Generators: DAG1, DAG2) dwie dane są przekazywane z każdą operacją dostępu do pamięci czy bloku rejestrów. Istotnym ograniczeniem jest to, że operacje SIMD są wspierane jedynie dla dostępu do pamięci wewnętrznej układu.

Każdy z generatorów adresowych zawiera cztery typy rejestrów:

- indeksowe (I), działające jako wskaźniki do pamięci
- modyfikujące (M), określające wartość pre-/postmodyfikacji operacji związaną z operacją
  np. `DM(I0, M1)` – interpretowane jako lokacja w pamięci pod adresem zawartym w rejestrze I0, adres modyfikowany o wartość M1 przy operacji
- długości (L),
- bazowe (B), wraz z rejestrami długości definiują bufory kołowe.



DAG1 wytwarza 32-bitowy adres dla magistrali DM (Data Memory), DAG2 adresuje na magistrali PM (Program Memory).
Przy realizacji bufora kołowego DAG musi pracować w trybie post-modify (zapis: `DM(Ix, Mx)`, `PM(Ix, Mx))`.



Pełne wykorzystanie możliwości architektury SIMD wymaga kodowania algorytmu z użyciem asemblera, jednak interesującą możliwością kompilatora zawartego w środowisku VisualDSP jest optymalizacja pętli for. Wskazanie pętli która ma zostać przetworzona na postać wykorzystującą instrukcje SIMD odbywa się z użyciem dyrektywy pragma:

```
#pragma SIMD_for
```

umieszczonej przed pętlą. Kompilator sprawdza czy jest możliwe przetworzenie pętli z wykorzystaniem instrukcji SIMD i dokonuje ewentualnej optymalizacji. Poniżej przedstawiony został przykład przetworzenia pętli przez kompilator.

kod wejściowy:

```
float sum, c, A[N];
…
sum = 0;
#pragma SIMD_for
for (j=0; j<N; j++)
       {
       sum += c * A[j];
       }
```

kod po transformacji przez preprocesor/kompilator:

```
// declare SIMD temporaries
float t_sum[2], t_c[2];
// initialize both partial sums
t_sum[0] = t_sum[1] = 0;
// initialize both parts of scalar constant
t_c[0] = t_c[1] = c;
// ENTER_SIMD_MODE -- set machine mode
for (j=0; j<N; j+=2)      //zwiększanie indeksu o 2 w każdym obiegu pętli
       {
       t_sum[0] += t_c[0] * A[j];
       // -- implicit SIMD processing performs:
       // t_sum[1] += t_c[1] * A[j+1];
       }
// LEAVE_SIMD_MODE
// combine partial sums
sum = t_sum[0] + t_sum[1];
```

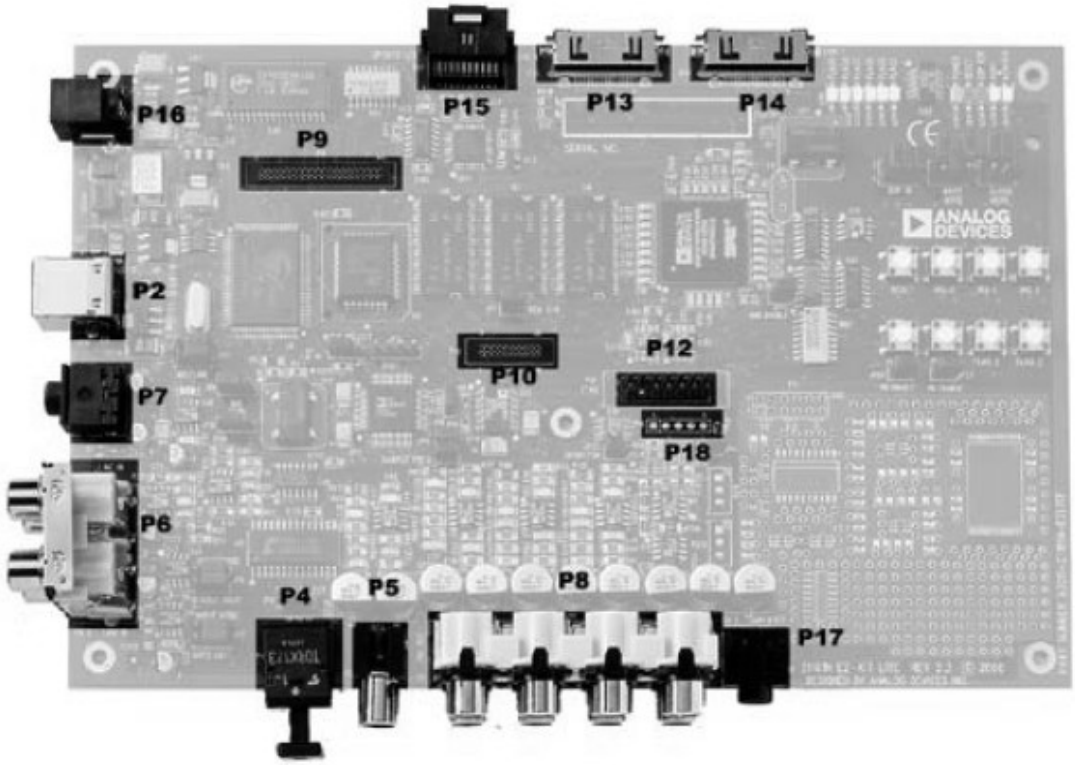
Praktyczne wykorzystanie instrukcji SIMD przedstawione zostało na przykładzie prostego dwukanałowego filtru typu FIR. Operacja filtracji jest wykonywana w przedstawionej aplikacji równocześnie na danych z obu kanałów. Program pozwala na włączenie jednego z czterech filtrów, których współczynniki deklarowane są w plikach *.dat. Przełączanie między poszczególnymi filtrami odbywa się poprzez wywołanie przerwania IRQ2. Możliwe jest również całkowite wyłączenie filtracji (FLAG3).

Aplikacja SISD_FIR_filter.dpj realizuje identyczną funkcjonalność jak przedstawiona wcześniej, lecz nie wykorzystuje instrukcji SIMD. Zostanie wykorzystana do porównania efektywności przetwarzania.

Zadania.

Przed wykonaniem zadań zapoznaj się z materiałami pomocniczymi oraz z kodem źródłowym prezentowanych aplikacji (szczególnie komentarzy zawartych w kodzie).

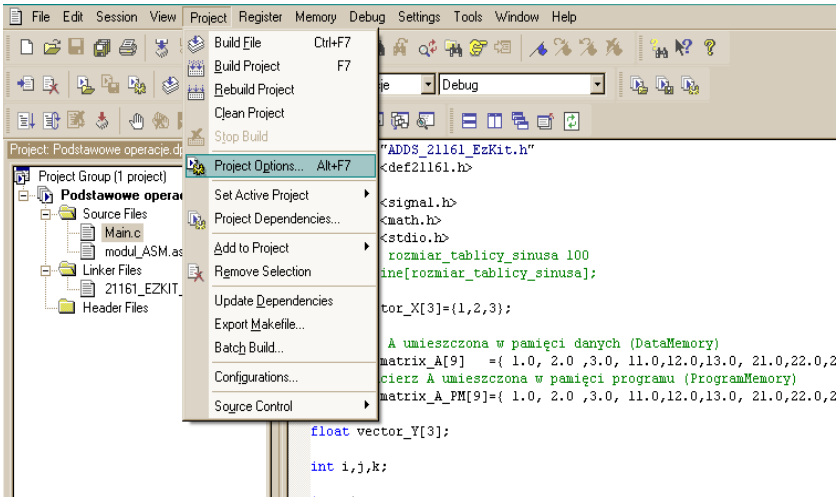
1. Podłącz zestaw uruchomieniowy do komputera.

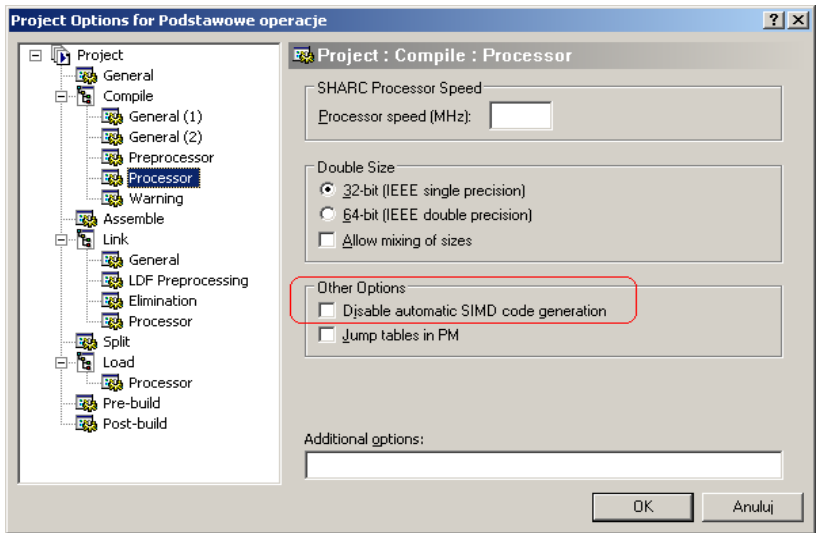


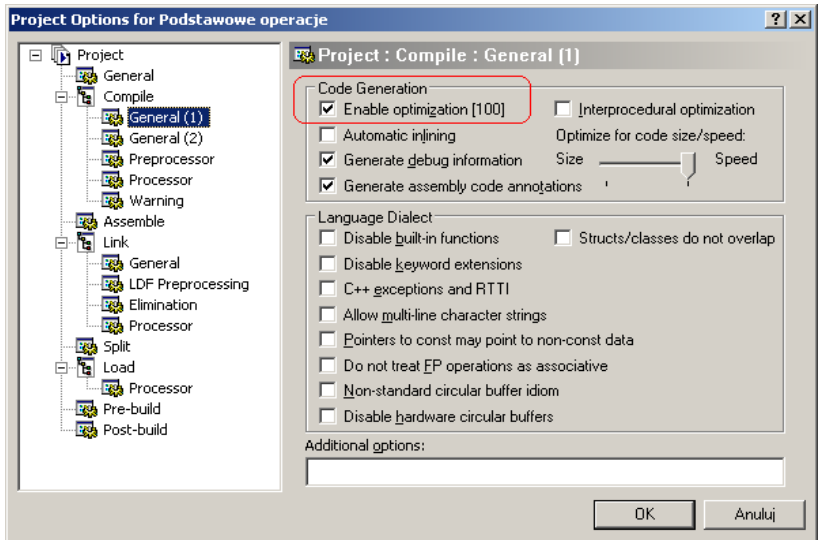
P16 – gniazdo zasilania
P2 – gniazdo USB
P6, gniazda umieszczone z prawej strony – wykorzystywane wejście sygnału audio
P17 – wyjście słuchawkowe

2. Uruchom środowisko Visual DSP++. (Jeśli korzystasz z VisualDSP++ po raz pierwszy - patrz **DODATEK A**)

3. Otwórz projekt *Podstawowe operacje.dpj* z katalogu SIMD C. Uruchom aplikację. Sprawdź ustawienie opcji (zakreślonych w rysunkach poniżej).

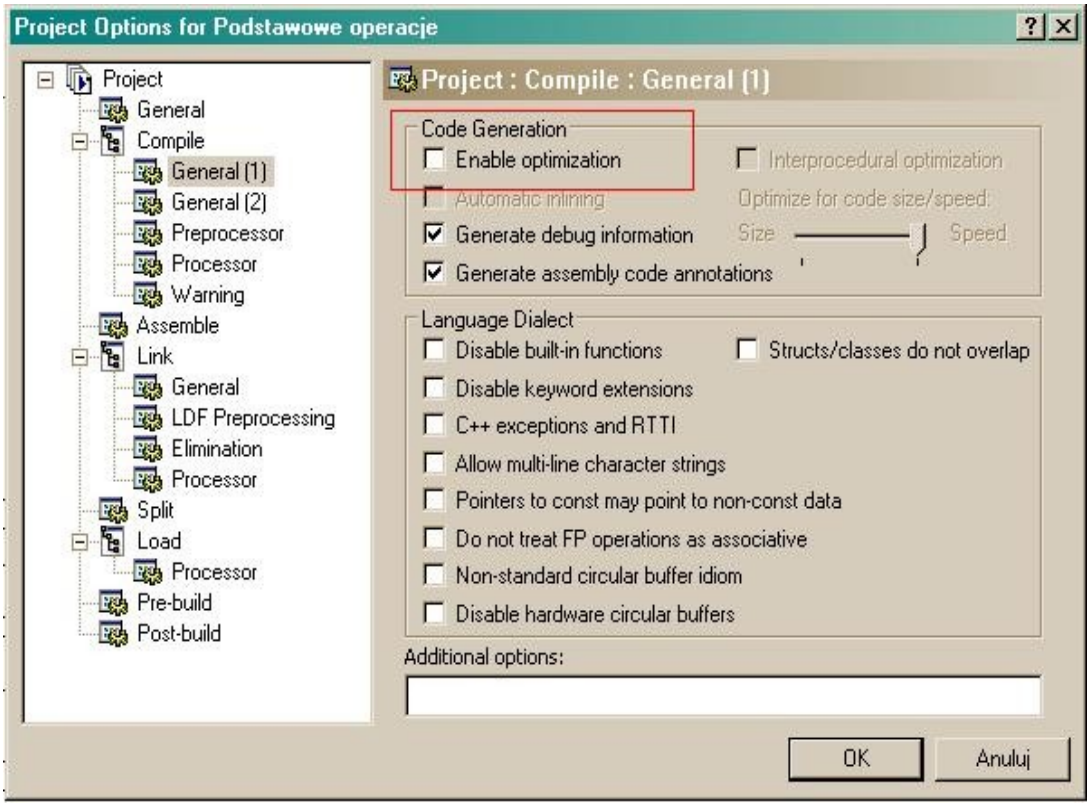


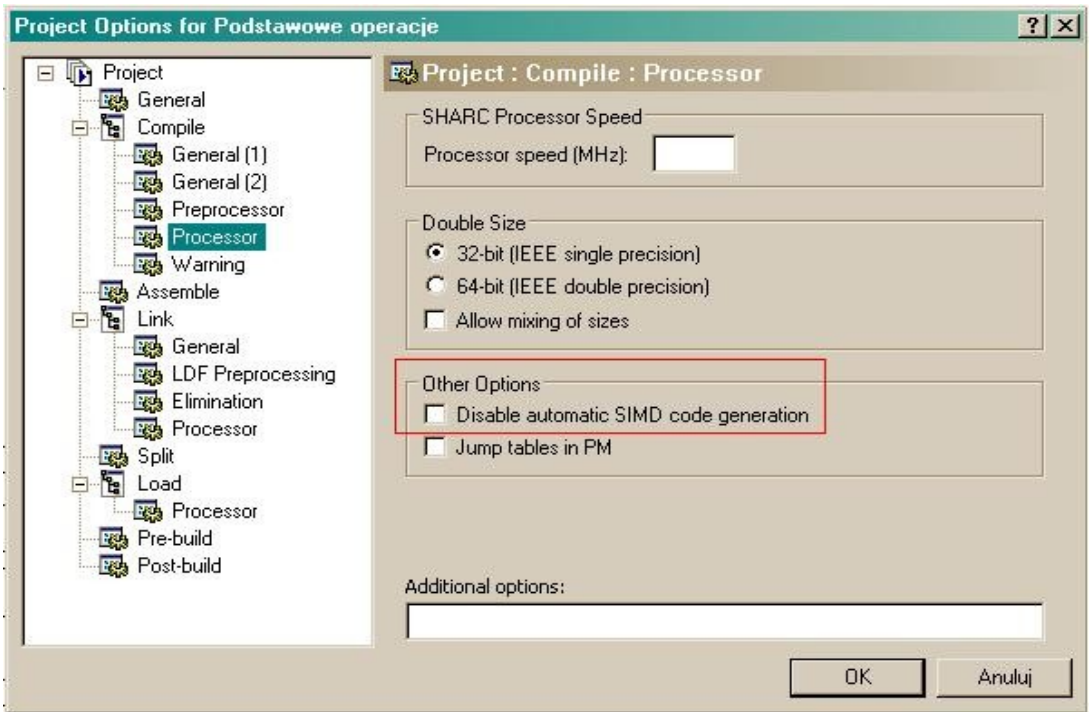




Przeładuj program poleceniem „Rebuild” i uruchom go przyciskiem „Run”. Zanotuj wyświetlane w oknie konsoli czasy wykonywania pętli bez optymalizacji i z włączoną optymalizacją SIMD (włączona opcja „Enable optymalization[100]”).

4. Przebuduj i uruchom ponownie projekt zmieniając ustawienia projektu: wyłączając globalną optymalizację oraz wyłączając generację kodu SIMD przy włączonej optymalizacji globalnej.





Porównaj i skomentuj czasy wykonywania procedury mnożenia.

5. Przeanalizuj kod źródłowy zawarty w pliku *SIMD_FIR_demo.asm*. Sprawdź jak uporządkowane są współczynniki filtrów w plikach *.dat. Wskaż instrukcje odpowiedzialne za inicjalizację generatorów adresowych oraz instrukcję wejścia i wyjścia do/z trybu SIMD.
6. Uruchom aplikację z projektu *SIMD_FIR_filter.dpj*. Sprawdź działanie po wywołaniu przerwania IRQ2 oraz funkcji bypass po wciśnięciu i przytrzymaniu przycisku FLAG3.
7. Otwórz projekt *SISD_FIR_filter.dpj*. Zwróć uwagę na różnice w kodzie odpowiedzialnym za filtrację w stosunku do aplikacji wykorzystującej instrukcje SIMD.
8. Sprawdź działanie aplikacji z projektu *SISD_FIR_filter.dpj*.

Fragment kodu z pliku SIMD_FIR_demo.asm z projektu SIMD_FIR_filter.dpj, odpowiedzialny za przetwarzanie próbek z wykorzystaniem techniki SIMD:

```
/* ADSP-21161 System Register bit definitions */
#include       "def21161.h"

/* *** C preprocessor declarations (defines, includes, etc.) *** */
#define DOUBLE_FILTER_TAPS 132
#define FILTER_TAPS 66
#define HALF_FILTER_TAPS 33


.section /dm    dm_data;

.VAR    filter_counter = 1;    /* default is low-pass filter */
.ALIGN 2;                      /* wymuszenie odpowiedniego ułożenia zmiennych w pamięci */
                               /* (address alignment)*/
.VAR    dline [DOUBLE_FILTER_TAPS];                /* delay line of input samples */

.endseg;

/*------------------------------------------------------------------------*/

.section /pm     pm_data;

.ALIGN 2;
.VAR    coef_lo_300Hz[FILTER_TAPS] = "coeffs65_300Hz.dat";
/* declare and initialize the higpass filter coefficients*/
.ALIGN 2;
.VAR    coef_lo_600Hz[FILTER_TAPS] = "coeffs65_600Hz.dat";
/* declare and initialize the lowpass filter coefficients*/
.ALIGN 2;
.VAR    coef_hi_4kHz[FILTER_TAPS] = "coeffs65_4kHz.dat";
/* declare and initialize the higpass filter coefficients*/
.ALIGN 2;
.VAR    coef_hi_8kHz[FILTER_TAPS] = "coeffs65_8kHz.dat";
/* declare and initialize the lowpass filter coefficients*/

/* "coeffs129_300Hz.dat" & "coeffs129_600Hz.dat" contains the coefficients for a low pass
filter and "coeffs129_4kHz.dat" & "coeffs129_4kHz.dat" contains the coefficients for a high
pass filter */

.endseg;


.section /pm pm_code;

/* deklaracje etykiet */
.GLOBAL        init_fir_filter;
.GLOBAL        fir;
.GLOBAL        change_filter_coeffs;

/* deklaracje zmiennych dostępnych w innych plikach */
.EXTERN        RX_left_flag;
.EXTERN        Left_Channel_In1;
.EXTERN        Right_Channel_In1;
.EXTERN        Left_Channel_Out0;
.EXTERN        Right_Channel_Out0;
.EXTERN        Left_Channel_Out1;
.EXTERN        Right_Channel_Out1;

init_fir_filter:
       bit set mode1 CBUFEN ;
       nop ;

       /* initialize the DAGS (dual Data Address Generators) */
       b0 = dline;           l0 = @dline;          /* pointer to the samples (delay line) */
       m1 = 2;

       b9 = coef_lo_300Hz;   l9 = @coef_lo_300Hz;
       /* pointer to the highpass filter coefficients  */
       m9 = 1 ;

       i3=dline;      l3=@dline;     r12=r12 xor r12;      /* clear the delay line */

       bit set mode1 PEYEN ;                       /* przejscie do trybu pracy SIMD (PEY
ENable) */
       lcntr=FILTER_TAPS, do clear until lce;
/* lcntr: dlugosc petli, clear: etykieta ostatniej linii kodu, lce: warunek zakonczenia */
clear:    dm(i3,2)=f12;                            /* wyzerowanie lokacji pamięci */
```

```
       bit clr mode1 PEYEN ;                        /* opuszczenie trybu SIMD */

   rts;

init_fir_filter.end:

/* ////////////////////////////////////////////////////////////////////////////// *
 *                            FIR filter subroutine                               *
 * ////////////////////////////////////////////////////////////////////////////// */

fir:
       /* process Left Channel Input */
       r2 = -31;                              /* scale the sample to the range of +/-1.0 */
       r0 = DM(Left_Channel_In1);             /* pobranie probki */
       r1 = DM(Right_Channel_In1);            /* pobranie probki */
       f0 = float r0 by r2 ;                  /* and convert to floating point */
       f1 = float r1 by r2 ;                  /* and convert to floating point */

       f8 = f0;                      /* in case we are in bypass mode, copy to output */
       f9 = f1;                      /* in case we are in bypass mode, copy to output */

       if FLAG2_IN jump exit_filter;

do_filtering:
       dm(i0,m1)=f0 (LW);

       // Enable broadcast mode and also set the PEYEN.
       bit set mode1 PEYEN | BDCST9;
       bit clr mode1 IRPTEN;

       r12=r12 xor r12;              /* clear f12,put sample in delay line */
       f8=pass f12, f0=dm(i0,m1), f4=pm(i9,m9);/* clear f8,get data & coef */

       lcntr=FILTER_TAPS-1, do macloop until lce;
macloop:   f12=f0*f4, f8=f8+f12, f0=dm(i0,m1), f4=pm(i9,m9);

       f12=f0*f4, f8=f8+f12;         /* perform the last multiply */
       f8=f8+f12;                    /* perform the last accumulate */

       bit clr mode1 PEYEN | BDCST9;
       bit set mode1 IRPTEN;
       r9 = s8;

exit_filter:
       r1 = 31;                      /* scale the result back up to MSBs */
       r8 = fix f8 by r1;            /* convert back to fixed point */
       r9 = fix f9 by r1;            /* Convert back to fixed point */
       DM(Left_Channel_Out0) = r8;   /* send result to AD1836 DACs */
       DM(Right_Channel_Out0) = r9;
       DM(Left_Channel_Out1) = r8;                 /* send result to AD1836 DACs */
       DM(Right_Channel_Out1) = r9;

       rts;
fir.end:
```

Fragment kodu z projektu SISD_FIR_filter.dpj odpowiedzialny za przetwarzanie próbek w trybie SISD – różnice w stosunku do aplikacji wykorzystującej instrukcje SIMD.

```
fir:
        /* process Left Channel Input */
        r2 = -31;                                       /* scale the sample to the range of +/-1.0
*/
        r0 = DM(Left_Channel_In1);                      /* pobranie probki */
        r1 = DM(Right_Channel_In1);                     /* pobranie probki */
        f0 = float r0 by r2 ;                           /* and convert to floating point */
        f1 = float r1 by r2 ;                           /* and convert to floating point */

        f8 = f0;                        /* in case we are in bypass mode, copy to output */
        f9 = f1;                        /* in case we are in bypass mode, copy to output */

        if FLAG2_IN jump exit_filter;

do_filtering:
        dm(i0,m1)=f0 (LW);
        dm(i0,m1)=f1 (LW);


        bit clr mode1 IRPTEN;

        r12=r12 xor r12;                                        /* clear f12,put sample in delay
line; niejawne zerowanie f12 */
        /* odniesienie sie do rejestrow przez f0-f15: operacje zmiennoprzecinkowe */
        /* odniesienie sie do rejestrow przez r0-r15: operacje staloprzecinkowe */

        /* f0 - dline, f4 - coeff */
        f8=pass f12, f0=dm(i0,m1), f4=pm(i9,m9);/* clear f8 (f12->f8),get data & coef */
        f9=pass f12, f0=dm(i0,m1), f4=pm(i9,m9);/* drugi kanal */

        lcntr=FILTER_TAPS-1, do macloop until lce;
        f12=f0*f4, f8=f8+f12, f0=dm(i0,m1), f4=pm(i9,m9);
macloop:   f12=f0*f4, f9=f9+f12, f0=dm(i0,m1), f4=pm(i9,m9);

        f12=f0*f4, f8=f8+f12;                           /* perform the last multiply */
        f8=f8+f12;                                      /* perform the last accumulate */

        f12=f0*f4, f9=f9+f12;                           /* perform the last multiply */
        f9=f9+f12;                                      /* perform the last accumulate */


        bit set mode1 IRPTEN;                           /* globalne wlaczenie przerwana */

exit_filter:
        r1 = 31;                                        /* scale the result back up to MSBs */
        r8 = fix f8 by r1;                              /* convert back to fixed point */
        r9 = fix f9 by r1;                              /* Convert back to fixed point */
        DM(Left_Channel_Out0) = r8;                     /* send result to AD1836 DACs */
        DM(Right_Channel_Out0) = r9;
        DM(Left_Channel_Out1) = r8;                     /* send result to AD1836 DACs */
        DM(Right_Channel_Out1) = r9;

        rts;
fir.end:
```

DODATEK **A**:

Krok 1: Uruchom VisualDSP++ i Otwórz Projekt

Aby uruchomić VisualDSP++ i otworzyć projekt należy:

1. Kliknąć przycisk Start; Programs, Analog Devices, VisualDSP++ 4.0, i VisualDSP++ Environment.

Jeśli uruchamiasz VisualDSP++ po raz pierwszy, otworzy się okno New Session, które pozwoli założyć nową sesję.

a. Ustaw wartości pokazane w Tabeli 2-1.

Tabela 2-1. Session Specification – ustawienia sesji

Box	Value
Debug Target	EZ-KIT Lite(ADSP-21xxx)
Platform	ADSP-21xx EZ-KIT Lite
Session Name	ADSP-21161 ADSP-21xxx EZ-KIT Lite
Processor	ADSP-21161

b. Kliknij OK. Otworzy się główne okno VisualDSP++.

Jeśli już uruchamiałeś VisualDSP++, a opcja Reload last project at startup w Settings and Preferences jest zaznaczona, VisualDSP++ otworzy ostatni projekt, nad którym pracowano. By zamknąć ten projekt, wybierz Close z menu Project.

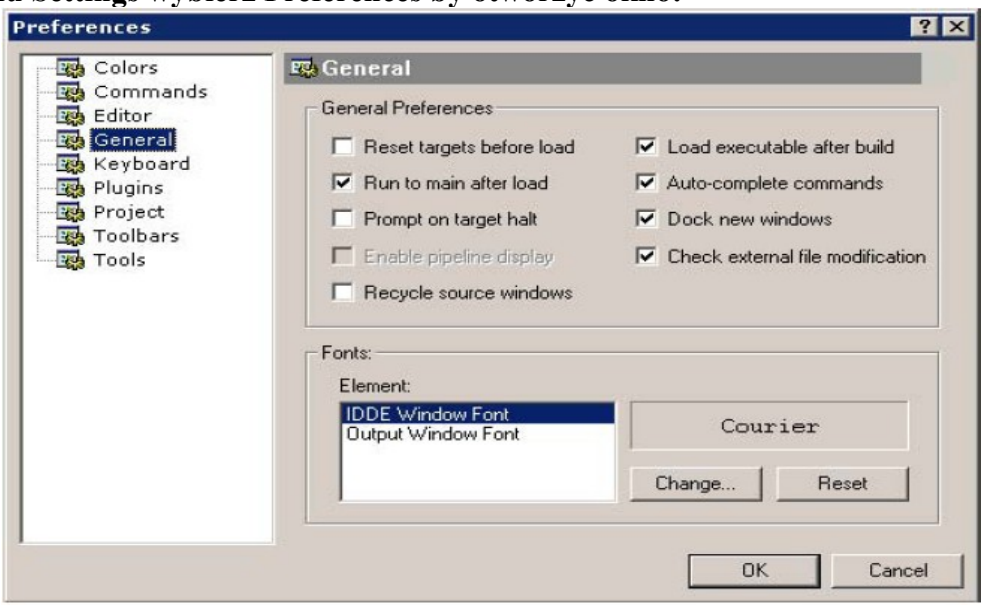
2. Z menu File wybierz Open i Project.

VisualDSP++ otworzy okienko dialogowe Open Project.

3. W oknie Look in, otwórz folder projektu I następnie kliknij dwukrotnie plik projektu Podstawowe operacje.dpj.

VisualDSP++ załaduje projekt w oknie Project. Program wyświetli komunikaty w oknie Output, kiedy ustali ustawienia i zależności pomiędzy plikami.

4. Z menu Settings wybierz Preferences by otworzyć okno:



5. Upewnij się, że na zakładce General, pod General Preferences, są zaznaczone następujące opcje:

- Run to main after load
- Load executable after build

7. Kliknij OK by zamknąć okienko Preferences.

Krok 2: Utwórz Projekt podstawowe operacje.dpj

By utworzyć projekt *Podstawowe operacje.dpj*:

1. Z menu Project wybierz Build Project.

VisualDSP++ najpierw sprawdza zmiany I zależności pomiędzy plikami a następnie tworzy projekt na podstawie plików źródłowych projektu.

Krok 3: Uruchamianie Programu

Jeśli zaznaczona jest opcja Load executable after build w zakładce General okna Preferences, plik wykonawczy Podstawowe operacje.dxe jest automatycznie ładowany do urządzenia docelowego. VisualDSP++ przejdzie do głównej funkcji kodu.

By uruchomić *Podstawowe operacje*, kliknij przycisk Run lub wybierz Run z menu Debug.

Literatura:

1. ADSP-21161 SHARC DSP Hardware Reference
2. ADSP-21161N EZ-KIT Lite® Evaluation System Manual
3. C/C++ Compiler and Library Manual for ADSP-21xxx Family DSPs
4. SIMD Architecture and Programming Model
5. Instruction Set Reference for ADSP-21160 SHARC DSPs