

# Wprowadzenie do VisualDSP++ 5.0

W tym rozdziale omówione zostaną poniższe tematy:

- “Przegląd”
- “Ćwiczenie Pierwsze: Tworzenie i uruchamianie programu w C”
- “Ćwiczenie Drugie: Modyfikowanie programu w C w celu wywołania funkcji Assemblera”
- “Ćwiczenie Trzecie: Wizualizacja Danych”
- “Ćwiczenie Czwarte: Profilowanie Liniowe”
- “Ćwiczenie Piąte: Optymalizacja na podstawie Profilingu”

## Przegląd

Poniższe Wprowadzenie demonstruje kluczowe funkcje i możliwości programu VisualDSP++ Integrated Development and Debugging Environment (IDDE). Ćwiczenia korzystają z przykładowych programów napisanych w języku C i Assemblerze dla procesorów Analog Devices.

Można używać różnorodnych procesorów rodziny Blackfin z niewielkimi tylko zmianami w plikach Linker Description Files (.LDFs), które są załączone do każdego projektu. VisualDSP++ zawiera podstawowe Linker Description Files dla każdego typu procesora w folderze `ldf`. Dla procesorów Shark, domyślnym folderem instalacyjnym jest:

```
Program Files\Analog Devices\VisualDSP 5.0\211xx\ldf
```

Pliki źródłowe dla poniższych ćwiczeń są zainstalowane wraz z VisualDSP++ oraz znajdują się na

```
D:\dsp_dokumenty\dsp5\211xx\Examples\ADSP-21161 EZ-KIT Lite.
```

Projekty własne należy zgrywać i tworzyć na tym samym komputerze we własnym katalogu:

```
D:\dsp_dokumenty\lab\DZIEN\nazwisko1_nazwisko2\cwNUMER, gdzie: DZIEN jest dniem tygodnia z godz. rozpoczęcia, tj. pn14,cz12,cz14, zaś NUMER jest numerem ćwiczenia.
```

Wprowadzenie zawiera cztery ćwiczenia:

- W **Ćwiczeniu Pierwszym**, należy uruchomić VisualDSP++, utworzyć projekt zawierający kod źródłowy w C oraz zbadać wydajność funkcji w C.
- W **Ćwiczeniu Drugim**, należy utworzyć nowy projekt, stworzyć Linker Description File wykorzystujący program w Assemblerze, przebudować projekt oraz zbadać szybkość działania programu Assemblera.
- W **Ćwiczeniu Trzecim**, należy wykreślić różnorodne kształty sygnałów wygenerowanych przez algorytm Finite Impulse Response (FIR) – skończona odpowiedź impulsowa.
- W **Ćwiczeniu Czwartym**, stosowane jest profilowanie liniowe w celu zbadania szybkości działania algorytmu FIR z Ćwiczenia Trzeciego. Korzystając z zebranych danych, należy wskazać najbardziej

czasochłonne obszary algorytmu, które prawdopodobnie będą wymagały poprawienia na poziomie Assemblera.

We wszystkich ćwiczeniach używany jest symulator rodziny ADSP-211xx oraz procesor ADSP-21161.



Rys. 2-1 Pasek narzędzi VisualDSP++

## Ćwiczenie Pierwsze: Tworzenie i uruchamianie programu w C

W tym ćwiczeniu należy:

- Uruchomić VisualDSP++ Environment
- Otworzyć i zbudować istniejący już projekt
- Zapoznać się z oknami i komunikatami
- Uruchomić program

Źródła do tego ćwiczenia znajdują się w folderze dot\_product.c. Domyślna ścieżka to:

D:\dsp\_dokumenty\dsp5\211xx\Examples\ADSP-21161 EZ-KIT Lite  
\211xx\Examples\ No Hardware Required \lab1.

### Krok 1: Uruchom VisualDSP++ i Otwórz Projekt

Aby uruchomić VisualDSP++ i otworzyć projekt należy:

1. Kliknąć przycisk **Start; Programs, Analog Devices, VisualDSP++ 5.0 i VisualDSP++ Environment.**

Jeśli uruchamiasz VisualDSP++ po raz pierwszy, otworzy się okno **New Session** (Rys. 2-6), które pozwoli założyć nową sesję.

- a. Ustaw wartości pokazane w Tabeli 2-1.

Tabela 2-1. Session Specification – ustawienia sesji

Box	Value
Processor	ADSP-21161
Connection type	Simulator
Platform (Target Name)	ADSP-2116x Simulator (ADSP- 2116x Family Simulator)
Session Name	ADSP-21161 ADSP-2116x Simulator

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 pracowałeś. By zamknąć ten projekt, wybierz **Close** z menu **Project**.

2. Z menu **File** wybierz **Open** i **Project**.

VisualDSP++ otworzy okienko dialogowe **Open Project**.

3. Najpierw skopiuj do swojego katalogu pliki z lab1

i następnie w oknie **Look in**, otwórz folder `dot_product_c`

klikając dwukrotnie w swoim podkatalogu

4. Kliknij dwukrotnie plik projektu `dotprodc (.dpj)`.

VisualDSP++ załaduje projekt w oknie **Project**, jak na [Figure 2-2](#). Program wyświetli komunikaty w oknie **Output**, kiedy ustali ustawienia i zależności pomiędzy plikami.



Rys. 2-2. Projekt w Oknie Projektu

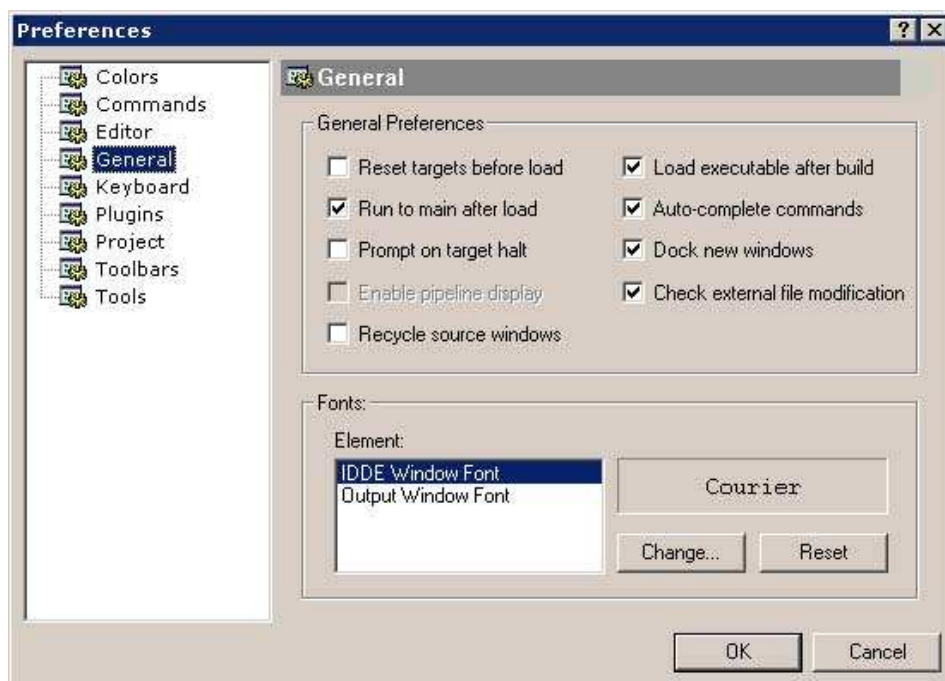
Projekt `dotprodc` zawiera dwa pliki źródłowe w C: `dotprod.c` oraz `dotprod_main.c`, które deklarują tablice oraz obliczają ich iloczyny skalarne.

5. Z menu **Settings** wybierz **Preferences** by otworzyć okno z [Rys. 2-3](#).

6. 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**.



Rys. 2-3 Okno Właściwości

Pojawi się główne okno VisualDSP++. Teraz można utworzyć nowy projekt.

## Krok 2: Utwórz Projekt dotprodc

By utworzyć projekt dotprodc:

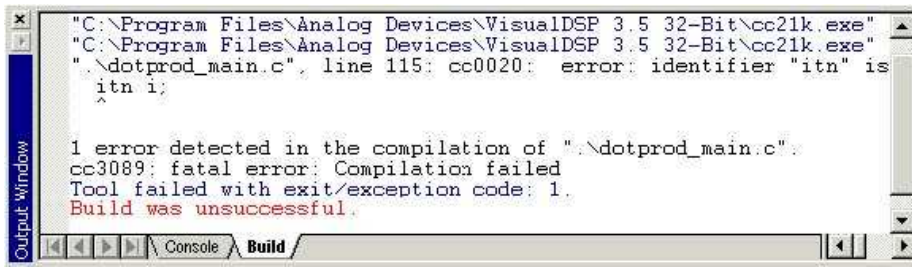
1. Z menu **Project** wybierz **Build Project**.

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

W miarę postępu, w oknie **Output** wyświetlane są komunikaty stanu (błędów oraz informacyjne). Na przykład, jeśli jedno z narzędzi programu wykryje nieprawidłową składnię lub brakujący odnośnik, wyświetlony zostanie raport błędu w okienku **Output**.

Po dwukrotnym kliknięciu nazwy pliku w komunikacie błędu, VisualDSP++ otworzy plik źródłowy w oknie edytora. Można edytować źródło by naprawić błąd, przebudować źródło i uruchomić debugger. Jeżeli konstrukcja projektu pozostała niezmienna (pliki, zależności oraz opcje nie zostały zmodyfikowane od ostatniego polecenia “build”), build nie jest wykonywane, chyba że zostanie wybrane polecenie **Rebuild All**. Pojawi się wtedy informacja “Project is up to date.” W przypadku braku błędów, ujrzysz informację: “Build completed successfully.”

W tym przykładzie (Rys. 2-4) kompilator wykrył nieznaną deklarację i wyświetlił następujący błąd w widoku **Build** okna **Output**.



Rys. 2-4 Przykład komunikatu błędu

2. Kliknij dwukrotnie tekst błędu w oknie **Output**.

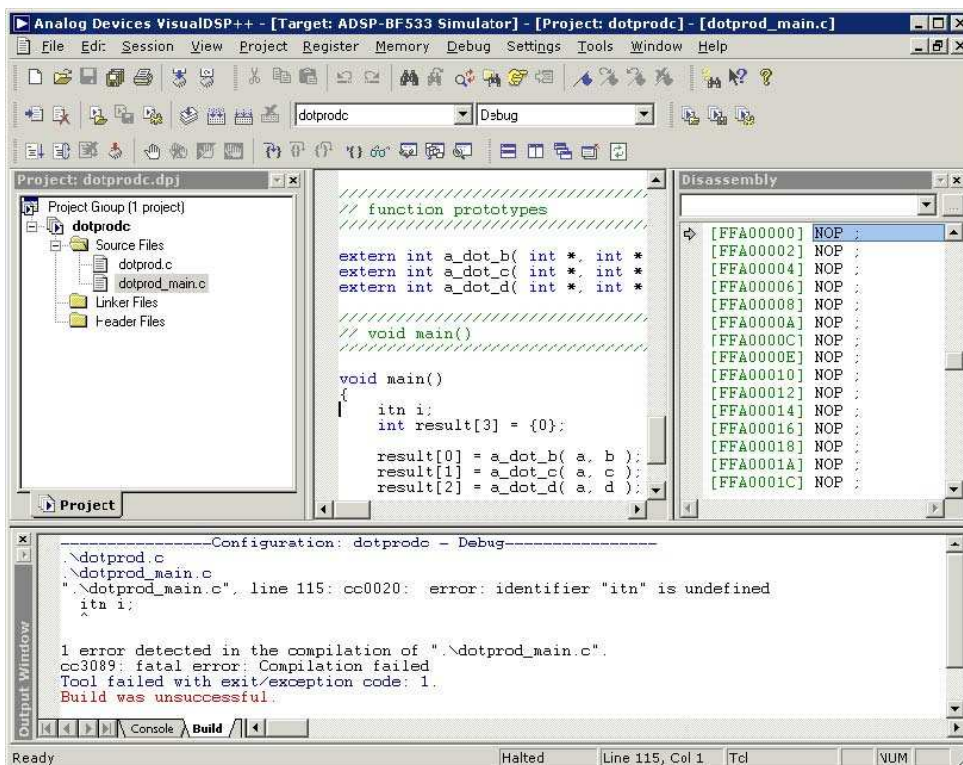
VisualDSP++ otworzy plik źródłowy C dotprod\_main.c w oknie edytora i ustawi kursor w linii zawierającej błąd. (zob. Rys. 2-5).

W oknie edytora ukazanym na Rys. 2-5 widać, że w deklaracji zmiennej całkowitej nastąpiła literówka I int zostało napisane jako itn.

3. W oknie edytora kliknij na itn by zmienić je na int. Zauważ, że int jest teraz kolorowe dla zaznaczenia, że zostało rozpoznane przez kompilator C.

4. Zachowaj plik dotprod\_main.c z menu **File**→**Save**.

5. Wybierz **Build Project** z menu **Project** menu. Projekt jest teraz utworzony bez żadnych błędów, co zostało zakomunikowane w widoku **Build** okna **Output**.



Rys. 2-5 Okno Output i Edytora

Skoro tworzenie projektu zakończyło się sukcesem, można go teraz uruchomić.

### Krok 3: Uruchamianie Programu

W tym kroku należy:

- Ustawić debugger przed uruchomieniem programu
- Przejrzeć okna debuggera i okna dialogowe

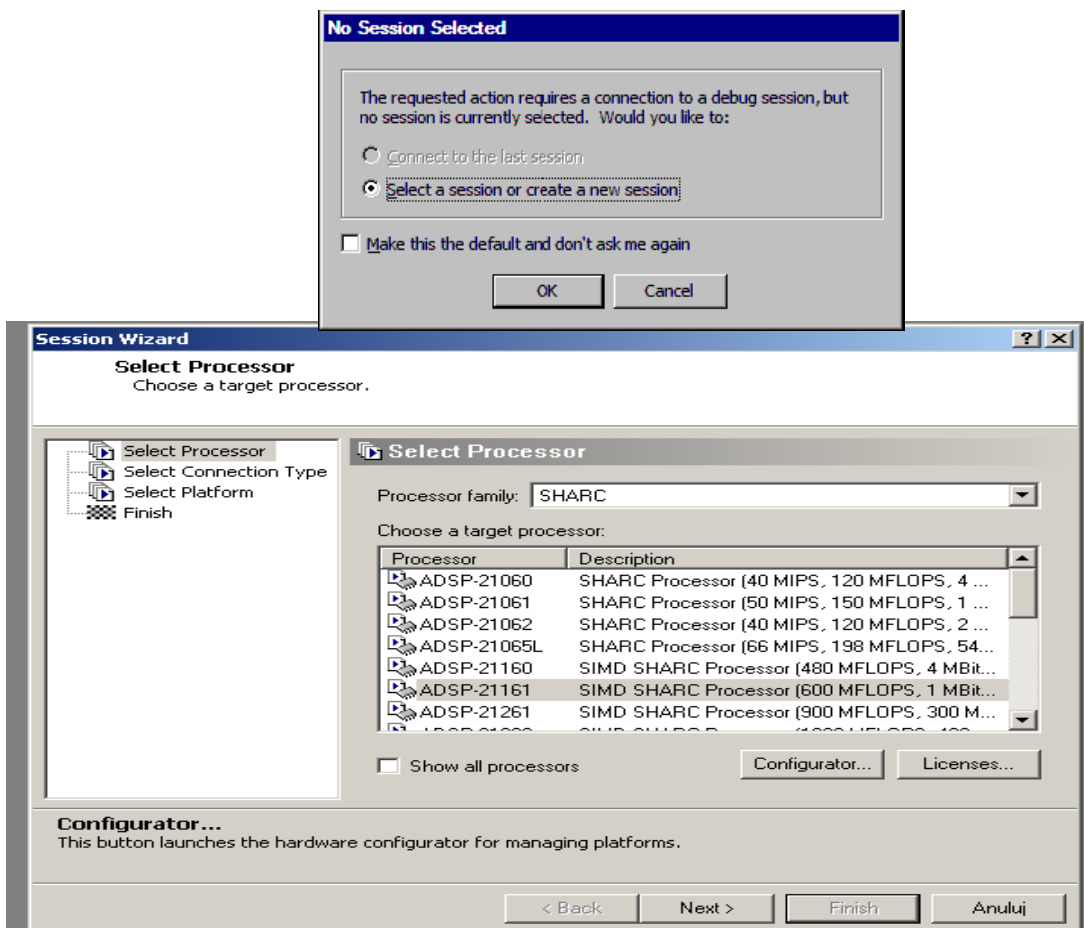
Jeśli zaznaczona jest opcja **Load executable after build** w zakładce **General** okna **Preferences**, plik wykonawczy `dotprodc.dxe` jest automatycznie ładowany do urządzenia docelowego. Jeżeli procesor użyty w sesji debuggera nie pokrywa się z urządzeniem docelowym, VisualDSP++ zakomunikuje niezgodność i zaproponuje czy nie chcesz wybrać innej sesji przed załadowaniem pliku wykonawczego do urządzenia.

Jeżeli VisualDSP++ nie otworzy okna **Session List**, pomiń kroki 1–4.

By ustawić sesję debuggera:

1. W oknie **Session List**, kliknij **New Session** (Rys. 2-6)

Dla następujących po sobie sesji debuggera użyj polecenia **New Session** z menu **Sessions**.



Rys. 2-6 Okno dialogowe nowej sesji i „session wizard” poniżej

2. Ustaw procesor urządzenia docelowego i jego parametry według [Tabeli 2-2](#).
3. Kliknij **OK** by zamknąć okno **New Session** i powrócić do okna **Session List**.
4. Podświetl nazwę sesji i kliknij **Activate**.

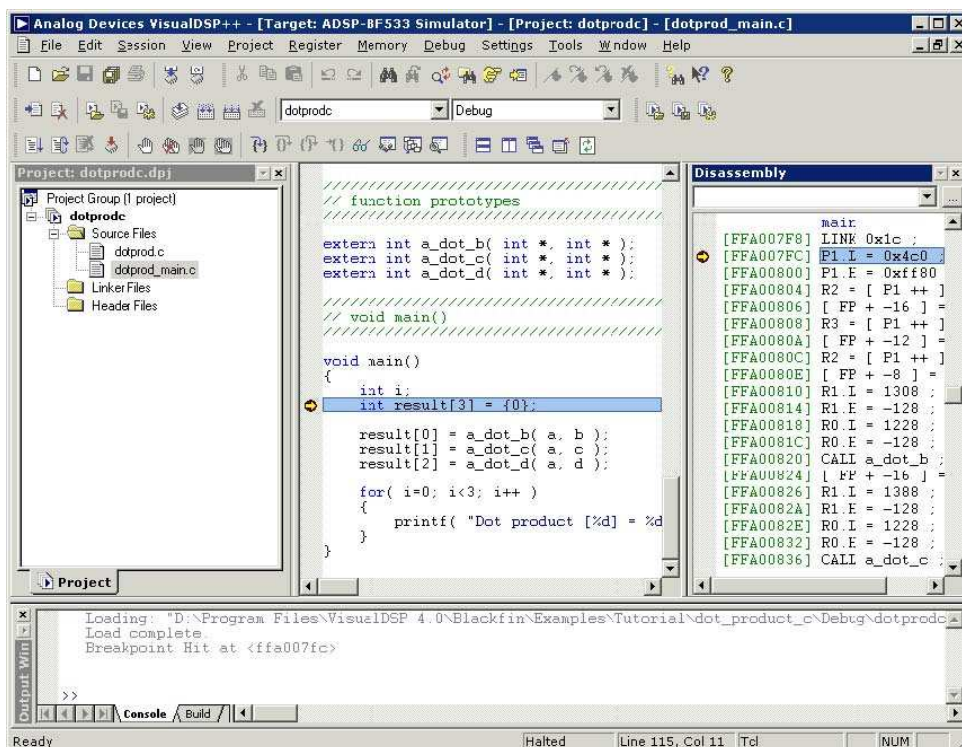


Tabela 2-6. Specyfikacja Sesji

Box	Value
Processor	ADSP-21161
Connection type	Simulator
Platform (Target Name)	ADSP-2116x Simulator (ADSP- 2116x Family Simulator)
Session Name	ADSP-21161 ADSP-2116x Simulator

Jeśli nie klikniesz **Activate**, wiadomość o niezgodności sesji pojawi się ponownie.

VisualDSP++ zamknie okno **Session List** i automatycznie załaduje plik wykonawczy projektu (`dotprodc.dxe`) oraz przejdzie do głównej funkcji kodu (zob. [Rys. 2-7](#)).



Rys. 2-7 Ładowanie dotprodc.dxe

## 5. Spójrz na informacje zawarte w oknach.

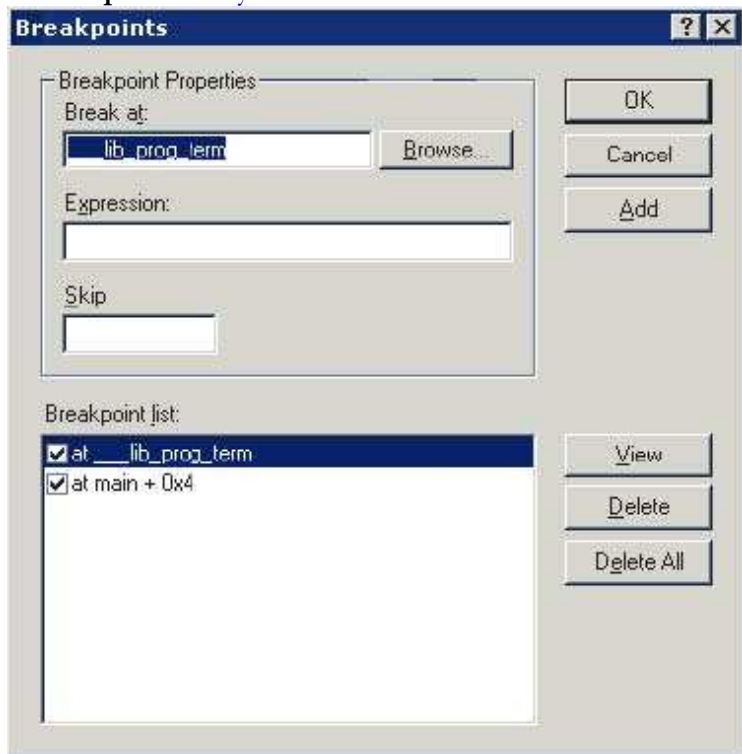
Strona **Console** okna **Output** zawiera informacje o stanie sesji debuggera. W tym przypadku, VisualDSP++ poinformował, że ładowanie `dotprodc.dxe` zostało ukończone.

Okno **Disassembly** wyświetla kod assemblera dla pliku wykonawczego.

Zauważ, że na początku programu opisanego jako "main" pojawiło się pełne, czerwone kółko oraz żółta

strzałka. Kółko ( ● ) oznacza, że na danej instrukcji ustawiony jest punkt stop, a strzałka ( ➔ ) wskazuje na instrukcję, na której aktualnie zatrzymał się procesor. Po załadowaniu programu w C, VisualDSP++ ustawia automatycznie dwa punkty stop (na początku i na końcu). Punkty stop mogą się nieco różnić od pokazanych w przykładzie.

6. Z menu **Settings** wybierz **Breakpoints** by przejrzeć punkty stop zawarte w programie. VisualDSP++ otworzy okno **Breakpoints** – Rys. 2-8.



Rys. 2-8 Okno punktów stop

Punkty stop są ustawione w programie w następujących miejscach:

- at main + 0x04
- at \_\_lib\_prog\_term

Okno **Breakpoints** pozwala przeglądać, dodawać i usuwać punkty stop oraz przeglądać symbole. W oknach **Disassembly** i edytora podwójne kliknięcie na linii kodu przelacza (dodaje lub usuwa) punkt stop. Jednakże w oknie edytora należy dodatkowo umieścić kursor myszki w rowku przed kliknięciem. Te przyciski służą do przelaczania punktów stop:



, Przelacza punkt stop w danej linii




Usuwa wszystkie punkty stop

7. Kliknij **OK** lub **Cancel** by opuścić okno **Breakpoints**.



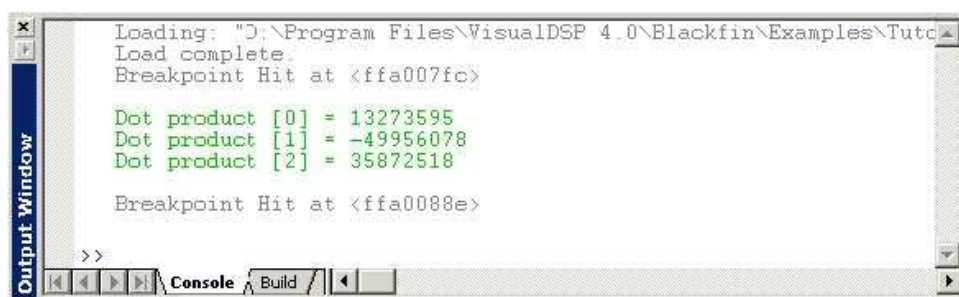
## Krok 4: Uruchomienie dotprodc

By uruchomić `dotprodc`, kliknij przycisk **Run** 

lub wybierz **Run** z menu **Debug**.

VisualDSP++ oblicza iloczyny skalarne i wyświetla następujące wyniki w widoku **Console** (Rys. 2-9) okna **Output**.

```
Dot product [0] = 13273595
Dot product [1] = -49956078
Dot product [2] = 35872518
```



Rys. 2-9 Wynik programu

## Ćwiczenie Drugie: Modyfikowanie programu w C w celu wywołania funkcji Assemblera

W ćwiczeniu pierwszym utworzyłeś i uruchomiłeś program w C. W tym ćwiczeniu należy:

- zmodyfikować program w C w celu wykorzystania funkcji w Assemblerze
- stworzyć Linker Description File w celu połączenia z funkcją
- przebudować projekt

Pliki projektu są zasadniczo identyczne z tymi, które zostały użyte w ćwiczeniu pierwszym. Tylko nieznaczne modyfikacje zostały wprowadzone w celu zrealizowania zadania ćwiczenia.

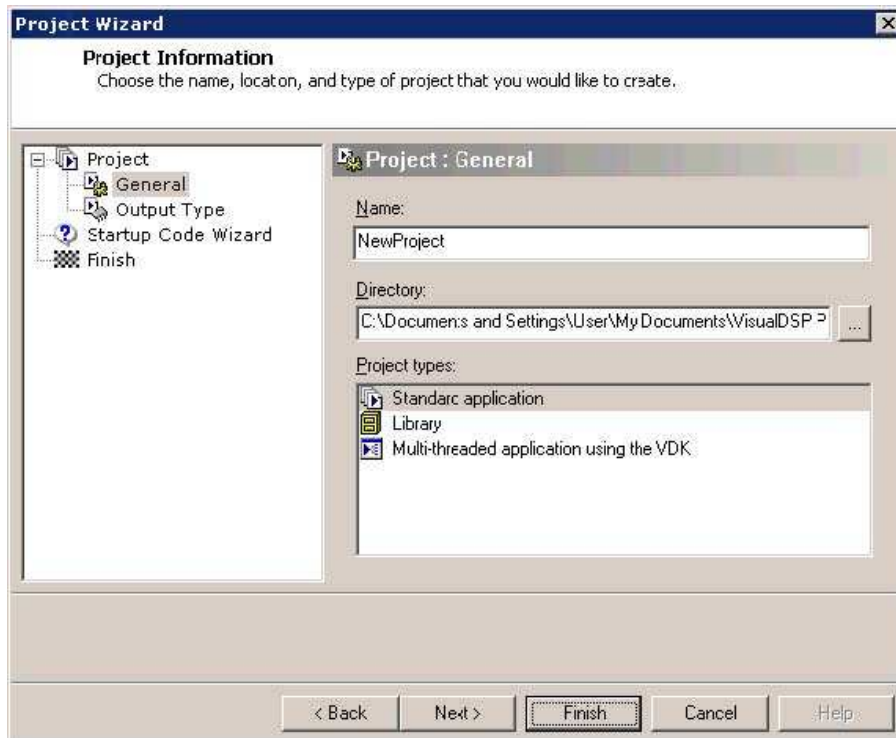
### Krok 1: Utwórz Nowy Projekt

1. Z menu **File** wybierz **Close** i **Project dotprodc** by zamknąć projekt `dotprodc`.

Kliknij **Yes** by zamknąć wszystkie okna źródeł.

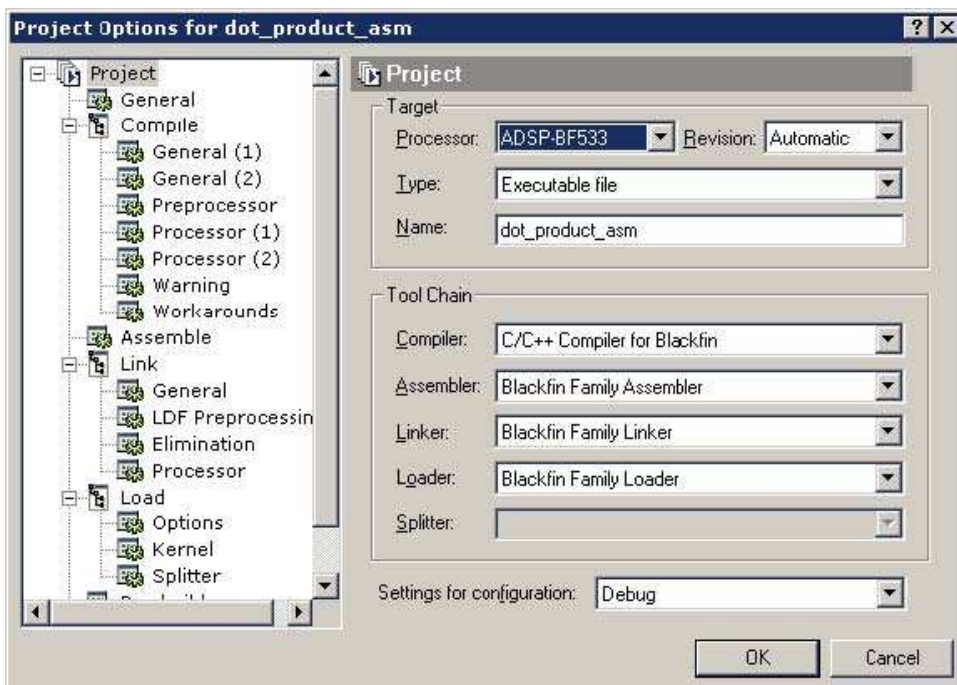
Jeśli modyfikowałeś projekt podczas sesji, zostaniesz zapytany czy chcesz go zapisać. Kliknij **No**.

2. Z menu **File** wybierz **New** i **Project** by otworzyć **Project Wizard**, Rys 2-10.



Rys. 2-10 Project Wizard

3. Kliknij przycisk **Browse** [...] na prawo od pola **Directory** by otworzyć okienko **Browse For Folder** . Znajdź folder `dot_product_asm` i kliknij **OK**. Domyślnie jest on w lab1



Rys. 2-11 Okno opcji projektu str.1 To okno pozwala określić parametry tworzenia projektu.

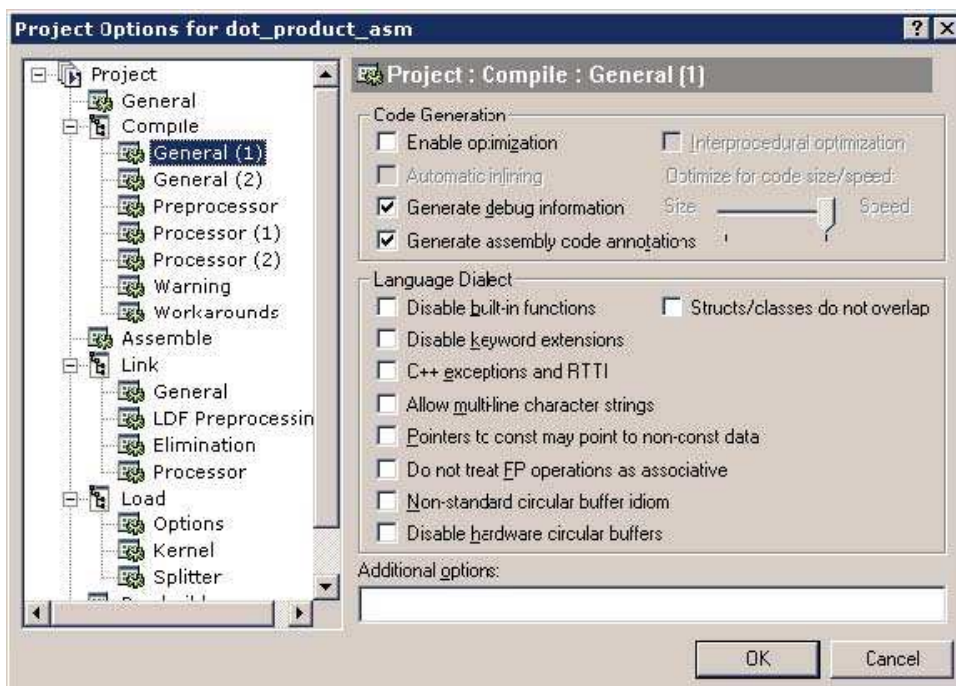
4. W polu **Name** wpisz `dot_product_asm` i kliknij **Finish**. Nowy projekt zostanie utworzony i będzie go można ujrzeć w oknie **Project** IDE.
5. Z menu **Project** wybierz **Project Options** (Rys. 2-11).
6. Przejrzyj różne strony okienka **Project Options** wybierając je z drzewka wyboru po lewej: **Project**, **General**, **Compile**, **Assemble**, **Link**, **Load**, **Pre-Build** i **Post-Build**. Na każdej ze stron można ustawiać narzędzia używane podczas tworzenia projektu.
7. Sprawdź, czy wartości na stronie **Project** (Rys. 2-11) pokrywają się z zawartymi w Tabeli 2-3.

Tabela 2-3.

Field	Value
Processor	ADSP-21161
Revision	Automatic
Type	Executable file
Name	dot_product_asm
Settings for configuration	Debug

Powyższe dane zawierają informacje niezbędne przy tworzeniu plików wykonawczych dla procesora ADSP-BF533. Pliki te zawierają informacje dla debuggera, więc możliwe jest sprawdzenie wykonania programu.

8. Kliknij przycisk **Compile** by przejrzeć stronę **Compile** z Rys. 2-12.



Rys. 2-12 Okno opcji projektu str.2


9. Ustaw pola **Code Generation**:

- a. Zaznacz **Enable optimization**.
- b. Zaznacz **Generate debug information** dla kodu w C..

Te ustawienia zapewnią optymalizację pracy kompilatora dla procesora ADSP-21161. Ponieważ optymalizacja korzysta z architektury DSC oraz Assemblera, niektóre informacje debuggera C mogą nie zostać zachowane. Dlatego operacja debuggingu jest wykonywana na poziomie języka Assembler.

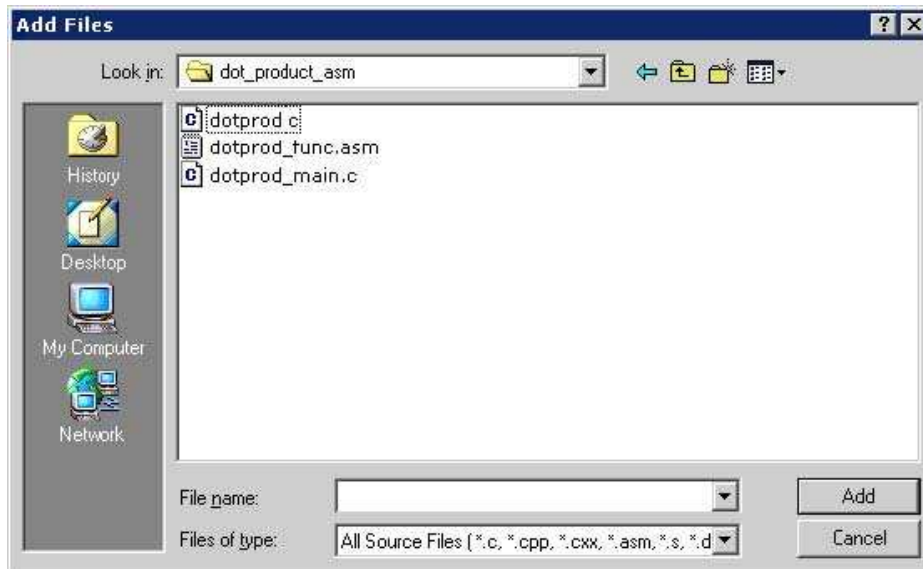
10. Kliknij **OK** by zachować zmiany i zamknąć okno **Project Options**. W razie pytania o “add support for the VisualDSP++ kernel?”, kliknij **No**.

## Krok 2: Dodawanie plików źródłowych do dot\_product\_asm

1. Kliknij przycisk **Add File** ,

lub z menu **Project** wybierz **Add to Project**, i **File(s)**.

Pojawi się okienko **Add Files** (Rys 2-13).



Rys. 2-13 Okno dodawania plików

2. W oknie **Look in** znajdź folder `dot_product_asm`.
3. W **Files of type** zaznacz **All Source Files** z listy.
4. Przytrzymaj **Ctrl** i kliknij `dotprod.c` i `dotprod_main.c`, a potem **Add**.

By przejrzeć dodane pliki otwórz folder `Source Files` w oknie **Project**.

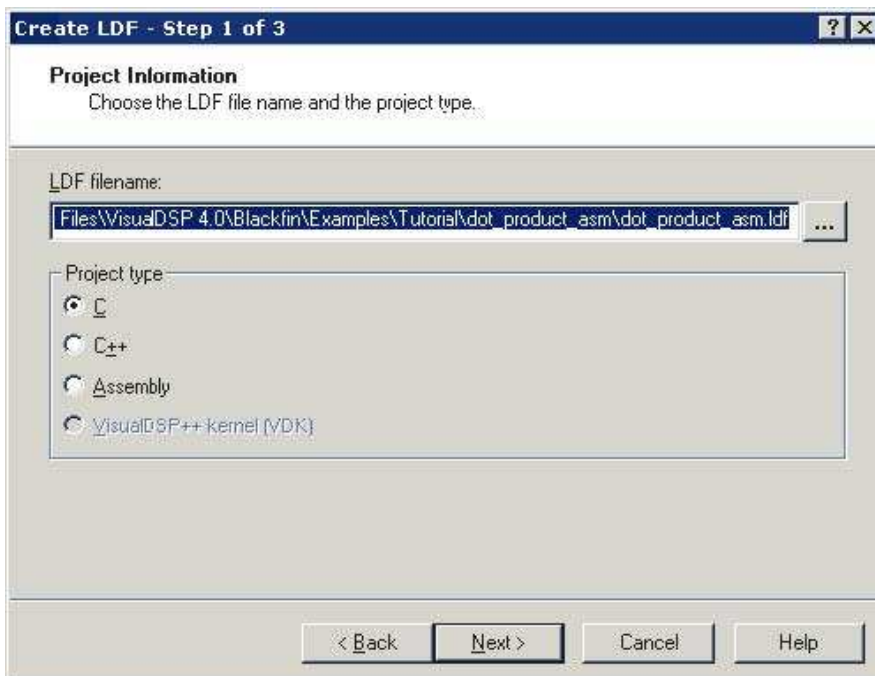
## Krok 3: Tworzenie Linker Description File

W celu utworzenia Linker Description File, należy użyć narzędzia Expert Linker.

1. Z menu **Tools** wybierz **Expert Linker** i **Create LDF**, by otworzyć **Create LDF Wizard**, (Rys. 2-14).
2. Kliknij **Next** by wyświetlić stronę **Create LDF – Step 1 of 3** (Rys. 2-15).



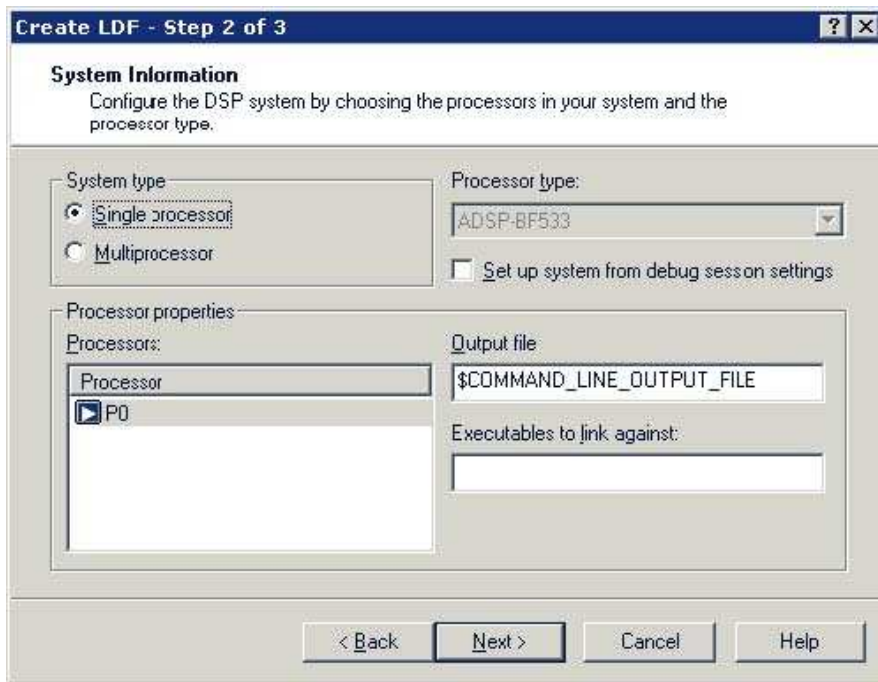
Rys. 2-14 Kreator LDF Wizard



Rys. 2-15 Tworzenie LDF str.1

Ta strona pozwala wybrać nazwę pliku **LDF** (zwykle nazwa\_projektu.ldf) oraz język projektu **Project type**.

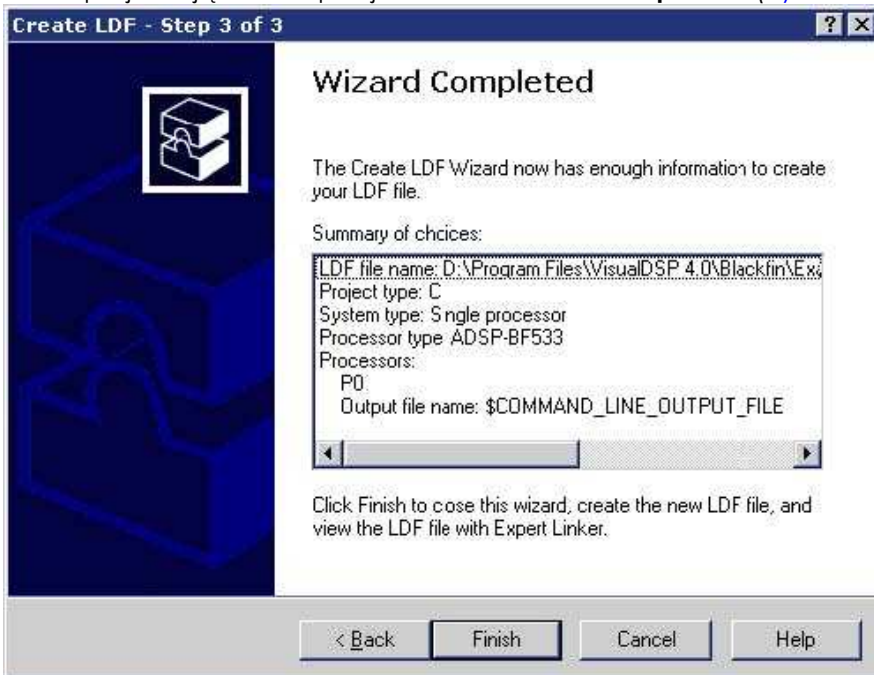
3. Zaakceptuj klikając **Next**; wyświetli się **Create LDF – Step 2 of 3** (Rys. 2-16).



Rys. 2-16 Tworzenie LDF str.2

Można tu ustawić typ systemu **System type** (domyślnie **Single processor**), typ procesora **Processor type** (domyślnie **ADSP-21161**) oraz nazwę pliku **Output file** linkera (domyślnie – nazwa wybrana przez projekt).

4. Zaakceptuj klikając **Next** i przejdź do **Create LDF – Step 3 of 3** (Rys. 2-17.)



Rys. 2-17 Tworzenie LDF str.3

5. Przeglądaj podsumowanie **Summary of choices** i kliknij **Finish** by utworzyć plik.LDF.

Stworzyłeś teraz plik.LDF w swoim projekcie. Znajduje się on w folderze **Linker Files**, w oknie **Project**.



Otworzy się okno Expert **Linker** z plikiem .LDF. Jest on gotowy by zadziałać w projekcie. Zamknij okno **Expert Linker**.

6. Kliknij przycisk **Rebuild All** 

Otworzy się plik źródłowy w C w oknie edytora, a wykonanie zostanie wstrzymane.

Projekt w wersji w C jest teraz gotowy. Teraz można zmodyfikować źródła by przywołać funkcję w Assemblerze.

## Krok 4: Modyfikowanie plików źródłowych projektu

- Zmodyfikuj `dotprod_main.c` by przywołał `a_dot_c_asm` zamiast `a_dot_c`
- Zachowaj zmodyfikowany plik

1. Dostosuj okno edytora dla lepszego widoku.
2. Z menu **Edit** wybierz **Find** (Figure 2-18).
3. W polu **Find What** wpisz `/*` i kliknij **Mark All**.



Rys. 2-18 Okno Find

Edytor zaznaczy wszystkie linie zawierające `/*` i ustawi kursor przy pierwszym znaku `/*` przy deklaracji `extern int a_dot_c_asm`.

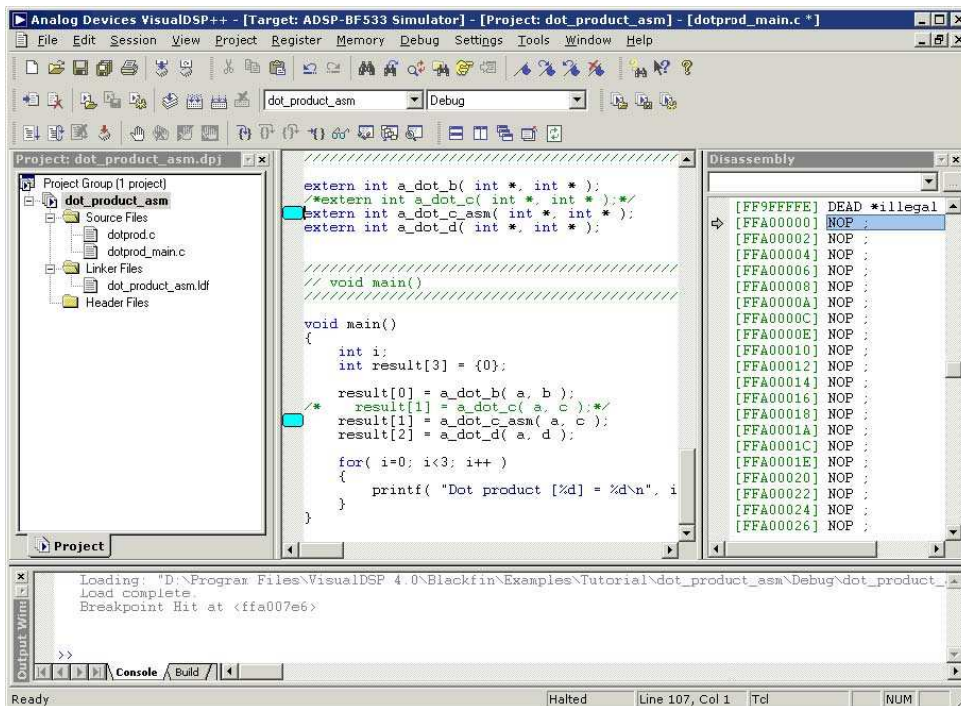
4. Zaznacz znaki komentarza `/*` i wciśnij **Ctrl+X**, by usunąć znaki komentarza z deklaracji `a_dot_c_asm`. Teraz przesun kursor o jedną linię w górę i wciśnij **Ctrl+V** by wkleić znaki komentarza przed deklarację `a_dot_c`. Deklaracje powinny zmienić kolor. Powtórz tę czynność dla znaków końca komentarza `*/`.

5. Naciśnij **F2** by przejść do następnego `/*`.  
Powtórz czynność opisaną powyżej

Funkcja `main()` przywołuje teraz `a_dot_c_asm` zamiast `a_dot_c` (wykorzystywaną w ćwiczeniu pierwszym).  
[Rys. 2-19](#)

7. Z menu **File** wybierz **Save** i **File dotprod\_main.c** by zachować zmiany.

8. Ustaw kursor w oknie edytora i zamknij `dotprod_main.c`.





Rys. 2-19 Modyfikowanie plików

## Krok 5: Użyj narzędzia Expert Linker by zmodyfikować dot\_prod\_asm.ldf

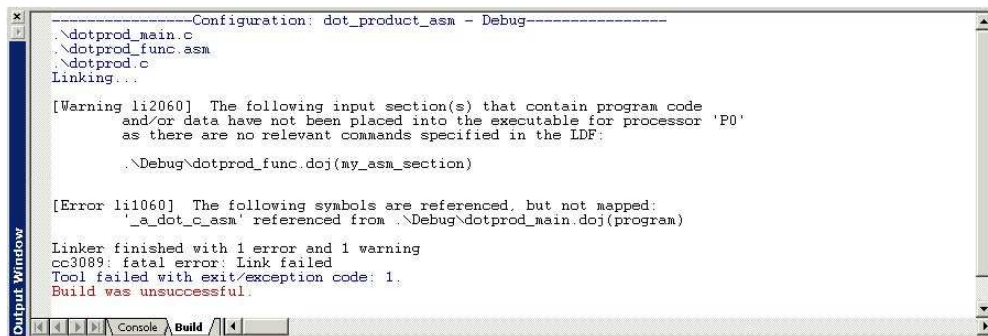
Przejrzyj utworzony plik.LDF w Expert Linker

- Zmodyfikuj plik.LDF w celu uwzględnienia funkcji a\_dot\_c\_asm assemblera

1. Kliknij przycisk **Add File** .
2. Wybierz dotprod\_func.asm i kliknij **Add**.
3. Zbuduj projekt poprzez:
  - Kliknięcie przycisku **Build Project** 
  - z menu **Project** wybierz **Build Project**.
4. W oknie **Output** wyświetli się błąd:

HIDC\_LDR\_BF\_MODE\_SPI liHIDC\_LDR\_BF\_MODE\_SPIInker  
(Rys. 2-20).

5. W oknie **Project** kliknij dwukrotnie plik dot\_prod\_asm.ldf . Otworzy się okno **Expert Linker** (Rys. 2-21) z graficzną reprezentacją pliku.

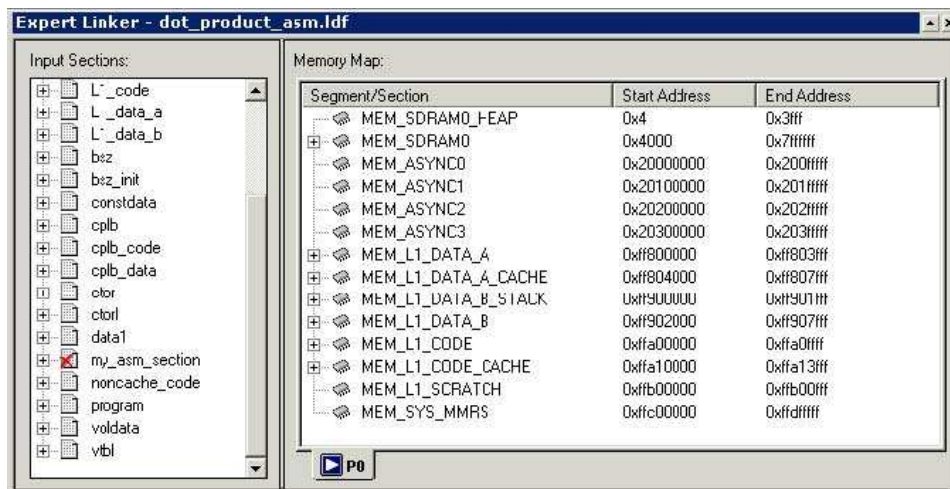


Rys. 2-20 Błąd linkera

Zmień rozmiar okna dla lepszego widoku. By wyświetlić drzewo z Rys. 2-21, kliknij prawym klawiszem myszy na prawym polu i wybierz **View Mode** oraz **Memory Map Tree**.

Lewe pole (**Input Sections**) zawiera listę input sections w projekcie lub w pliku.LDF. Zauważ, że przed “my\_asm\_section” znajduje się czerwony x, ponieważ Expert Linker stwierdził, że ta sekcja nie jest uwzględniona w pliku.LDF.

Prawe pole (**Memory Map**) zawiera reprezentację segmentów pamięci, które zarezerwował Expert Linker przy tworzeniu pliku.LDF.

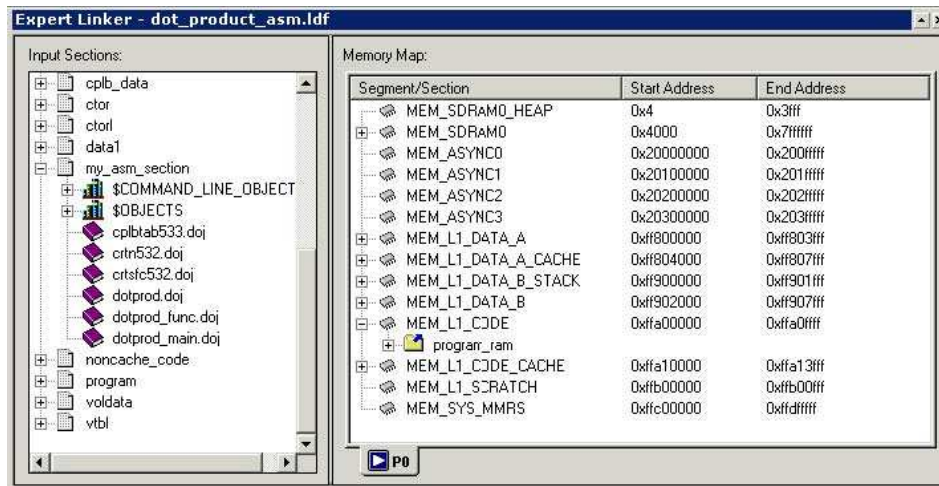


Rys. 2-21 Okno Expert Linker

#### 6. Uwzględnij my\_asm\_section w segmencie pamięci nazwanym MEM\_PROGRAM:

W polu **Input Sections** otwórz my\_asm\_section klikając na znak plus. Sekcja ta rozwinie się ukazując, że makra linkera \$COMMAND\_LINE\_OBJECTS i \$OBJECTS oraz plik dotprod\_func.doj zawierają sekcje, które nie zostały uwzględnione. W polu **Memory Map** rozwiń MEM\_L1\_CODE i przeciągnij ikonkę przed \$OBJECTS nad sekcję wyjściową program\_ram pod MEM\_L1\_CODE.

Na Rys. 2-22, widać, że czerwony x zniknął, bo sekcja my\_asm\_section została teraz uwzględniona.




Rys. 2-22 Przeciąganie obiektów

7. Z menu **Tools** wybierz **Expert Linker** oraz **Save**. Zamknij okno **Expert Linker**.

Jeśli zapomnisz zachować projekt i przebudujesz go poleceniem **Build Project**, VisualDSP++ zachowa go automatycznie.

## Krok 6: Przebuduj i uruchom dot\_product\_asm

1. Przebuduj project poprzez:

- Kliknij przycisk **Build Project** 
- Z menu **Project** wybierz **Build Project**.

Na koniec tworzenia w widoku **Build** okna **Output** wyświetli się komunikat:

“Build completed successfully.”

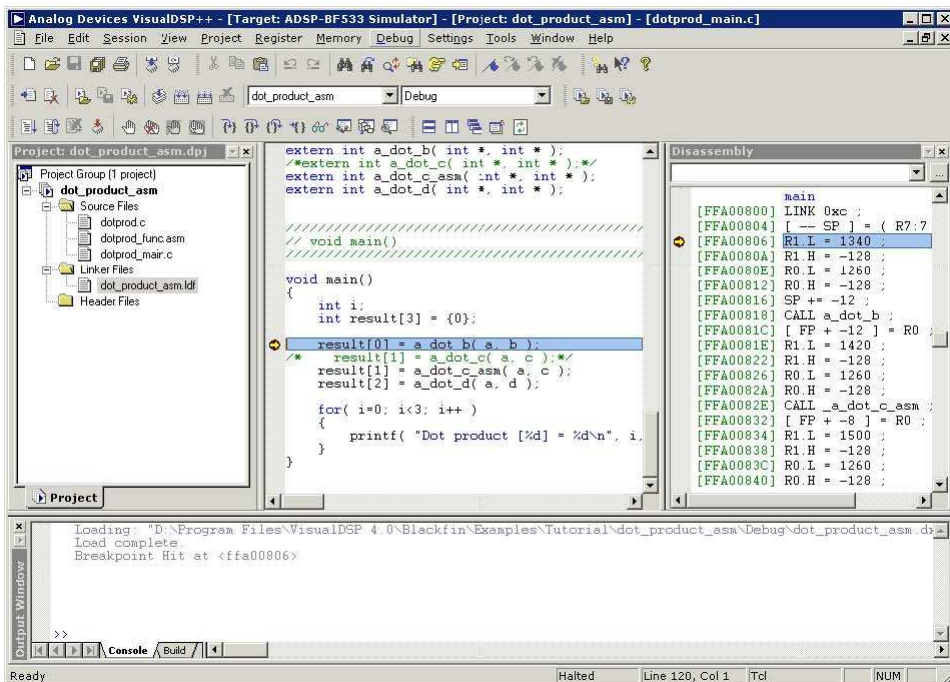
VisualDSP++ załaduje program, przejdzie do widoku głównego i wyświetli okna

**Output**, **Disassembly** oraz edytora (Rys. 2-23).

2. Kliknij przycisk **Run**  by uruchomić dot\_product\_asm.

Program oblicza trzy iloczyny skalarne i wyświetla wyniki w widoku **Console** okna **Output**. Po zakończeniu pracy programu wyświetlona zostanie informacja “Halted” w pasku stanu na dole głównego okna VisualDSP++. Poniższe wyniki są identyczne z otrzymanymi w ćwiczeniu pierwszym.

```
Dot product [0] = 13273595
Dot product [1] = -49956078
Dot product [2] = 35872518
```



Rys. 2-23 Potrzebne okna w tym ćwiczeniu

## Ćwiczenie Trzecie: Wykreślanie Danych

- Załaduj i zdebuguj przygotowany program, który stosuje prosty filtr Finite Impulse Response (FIR) na zestawie danych
- Użyj narzędzia do wykresów zawartego w VisualDSP++ by zobrazować różne zestawy danych przed i po przetworzeniu przez program.

### Krok 1: Załaduj Program FIR

1. Zamknij wszystkie okna prócz **Disassembly**, strona **Console**
2. Z menu **File** wybierz **Load Program** lub kliknij

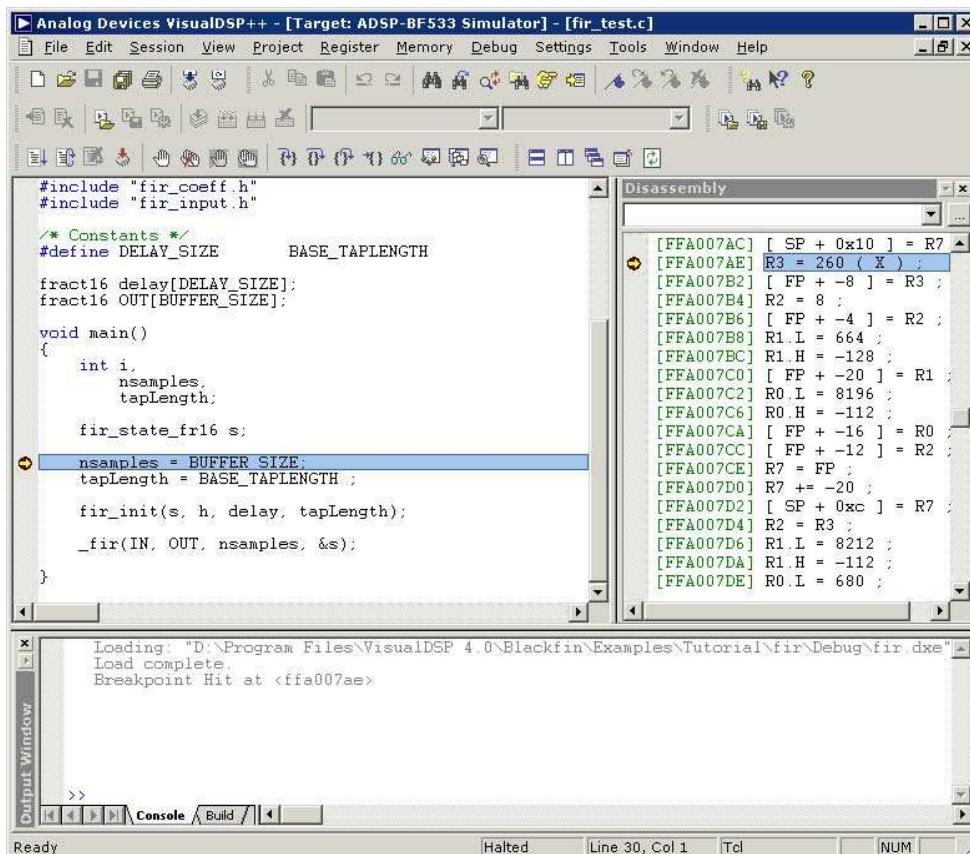


3. Wybierz poniższy program FIR:

- a. Otwórz folder `Analog Devices` i kliknij dwukrotnie w swoim folderze `lab1\fir`:
- b. Kliknij dwukrotnie podkatalog `Debug`.
- c. Kliknij dwukrotnie `FIR.DXE`.

Jeżeli VisualDSP++ nie otworzy okna edytora (Rys. 2-24), kliknij prawym klawiszem myszy na oknie **Disassembly** i wybierz **View Source**.





Rys. 2-24 Ładowanie FIR

#### 4. Spójrz na kod program FIR.

Zauważysz dwie globalne tablice danych:

- IN
- OUT

Zobaczysz też jedną funkcję, `fir`, która przetwarza te tablice.

### Krok 2: Otwórz okno wykresu

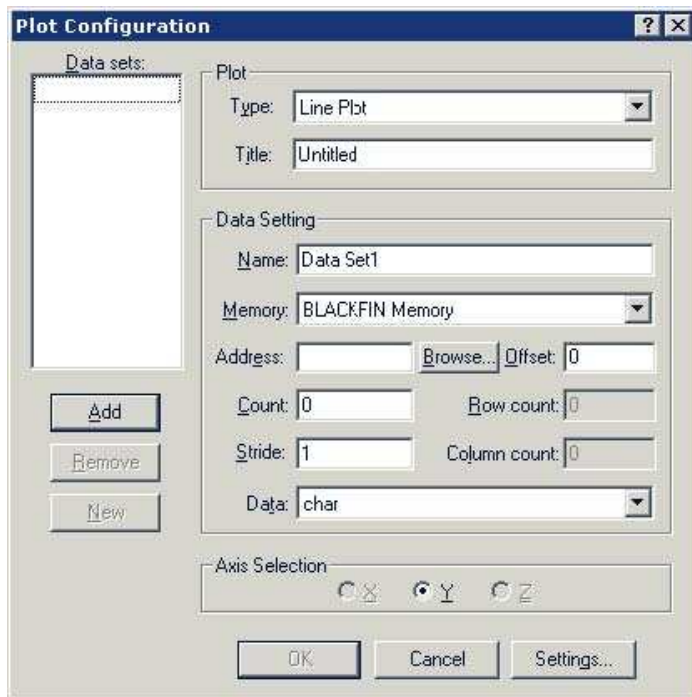
1. Z menu **View** wybierz **Debug Windows** i **Plot**. Kliknij **New** by otworzyć okno **Plot Configuration** – Rys. 2-25.

Tu dodaje się zestawu danych, które mają być wyświetlone w oknie wykresu.

2. W grupie **Plot** wybierz:

- W polu **Type** wybierz z listy **Line Plot**.
- W polu **Title** wpisz `fir`.





Rys. 2-25 Konfiguracja wykresu

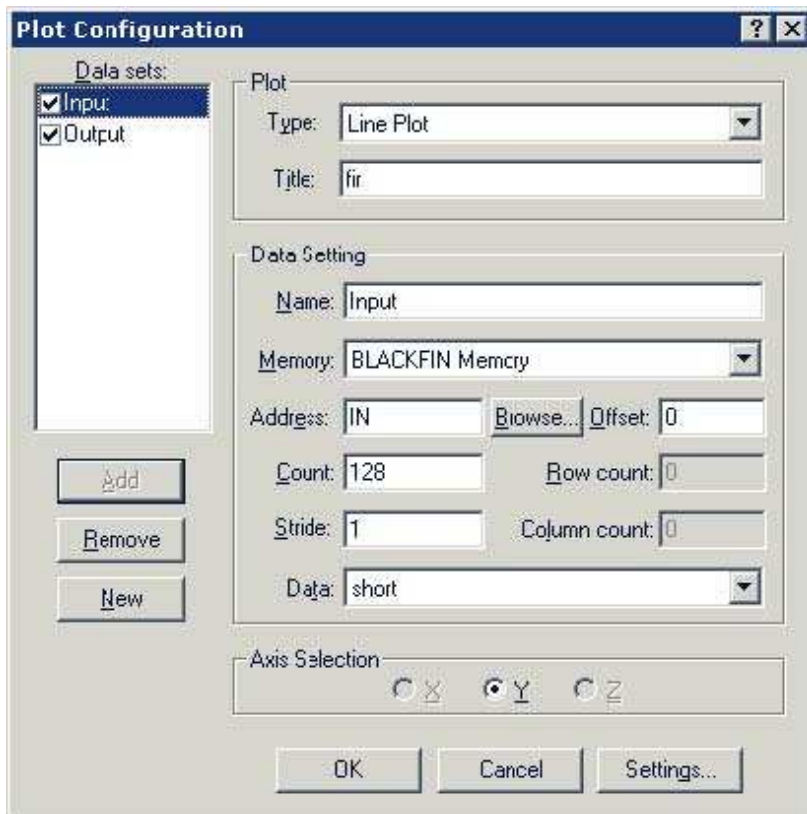
Dodaj dwa zestawy danych korzystając z danych w Tabeli 2-4.

Tabela 2-4. Wejście i wyjście

Box	Input Data Set	Output Data Set	Opis
Name	Input	Output	Zestaw danych
Memory	BLACKFIN Memory	BLACKFIN Memory	Pamięć
Address	IN	OUT	Kliknij <b>Browse</b> by wybrać z listy.
Count	128	128	Tablice są 260-o elementowe, ale należy użyć tylko pierwszych 128-u.
Stride	1	1	Dane są ciągle w pamięci.
Data	short	short	Dane są liczbami całkowitymi.

3. Po wpisaniu obu zestawów, kliknij **Add** by dodać je do **Data sets**.

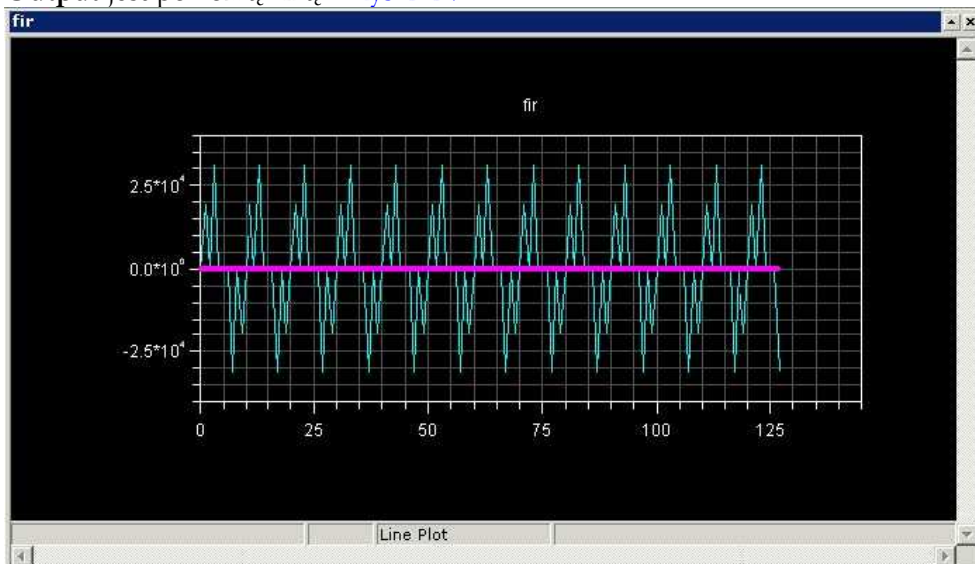
Okno **Plot Configuration** powinno teraz wyglądać tak jak na Rys. 2-26.



Rys. 2-26 Konfiguracja wykresu i zestawu I/O

4. Kliknij **OK** by zachować zmiany.

Okno wykresu wyświetla teraz dwie tablice. Domyślnie, symulator wypełnia pamięć zerami, dlatego **Output** jest poziomą linią – Rys. 2-27.



Rys. 2-27 Przed uruchomieniem FIR

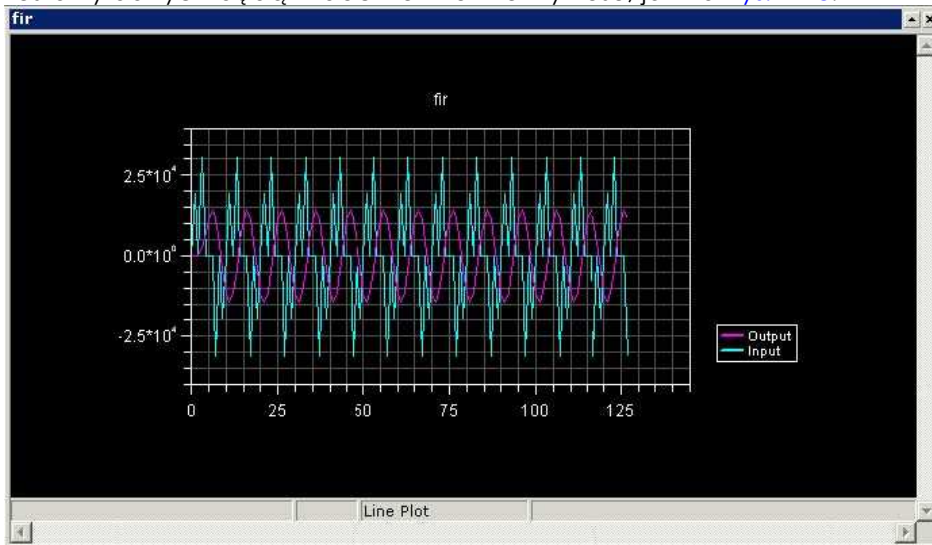
Zmiana rozmiarów okna wykresu wpływa na skalę osi x i y.

5. Kliknij prawym przyciskiem myszy na oknie i wybierz **Modify Settings**. Na stronie **General**, grupie **Options**, zaznacz **Legend** i kliknij **OK** by wyświetlić legendę.

### Krok 3: Uruchamianie programu FIR i zobrazowanie danych

1. Naciśnij F5 lub kliknij przycisk **Run**  by uruchomić program.

Po zakończeniu pracy programu, ujrzysz rezultaty pracy filtra FIR w tablicy Output. Obydwa zestawy danych będą widoczne w oknie wykresu, jak na [Rys. 2-28](#).



Rys. 2-28 Wykres po uruchomieniu FIR

Następnie przybliż wybrany fragment wykresu.

2. Kliknij na wykresie i przeciągnij kursor by narysować prostokątny obszar, który ma zostać przybliżony. Zwolnij przycisk.

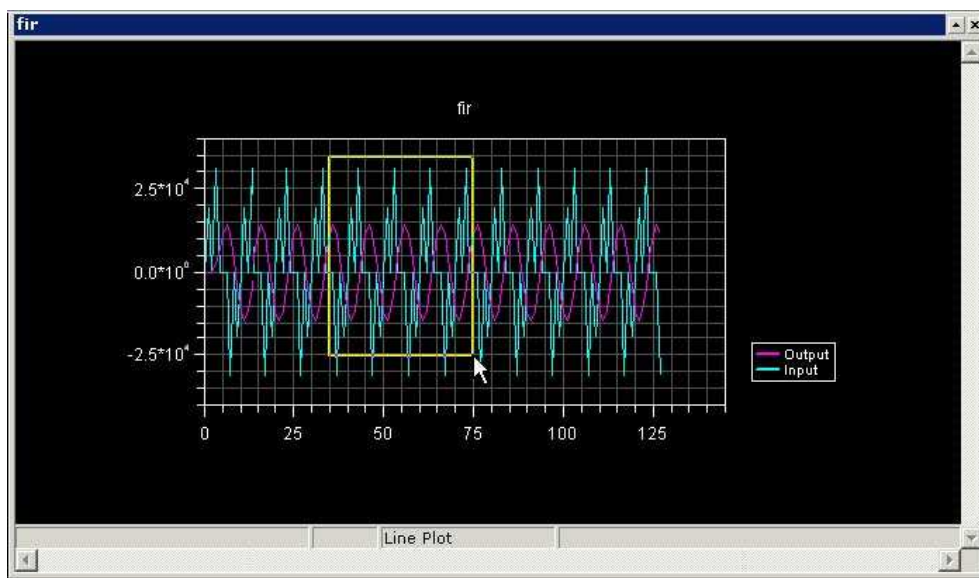
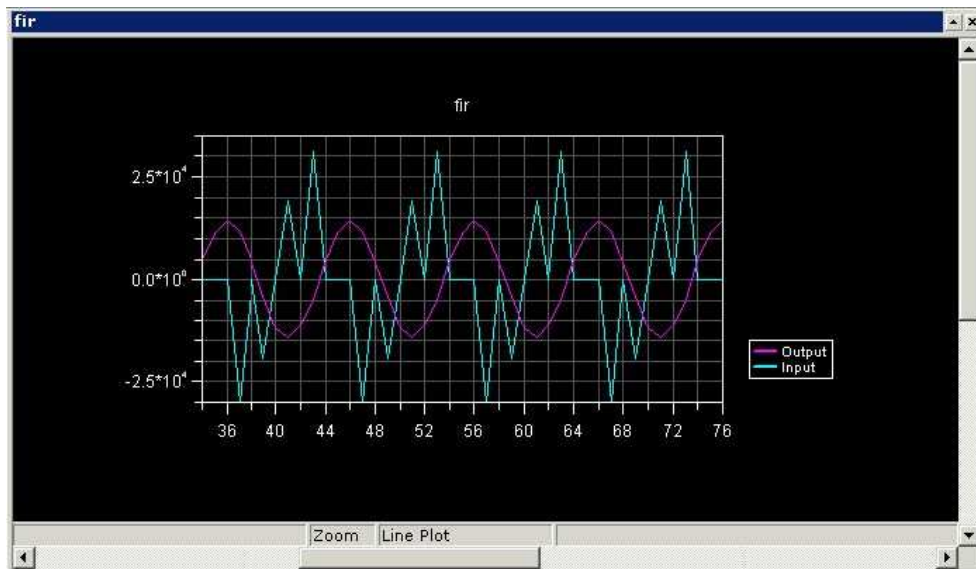


Figure 2-29. Wybór obszaru powiększenia



Rys. 2-30 Powiększenie

By powrócić do poprzedniego widoku, kliknij prawym przyciskiem myszy na wykres i wybierz **Reset Zoom**.

3. Teraz kliknij prawym przyciskiem myszy i wybierz **Data Cursor**. Możesz przemieszczać się między kolejnymi punktami wykresu używając kursorów Lewo-Prawo na klawiaturze. Do przełączania się między zestawami danych służą kursory Góra-Dół. Wartości charakterystyczne dla aktualnego punktu są wyświetlane w lewym dolnym rogu okna – Rys. 2-31.

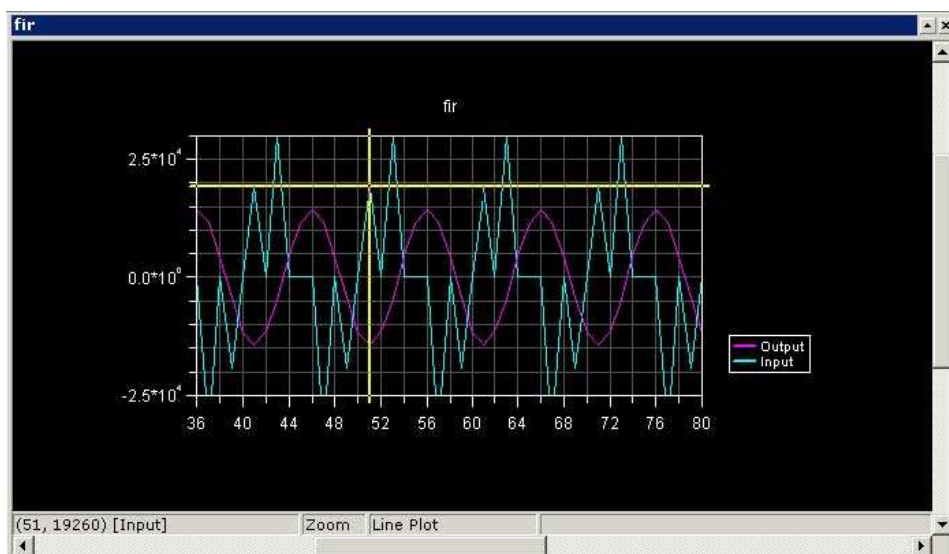
4. Kliknij prawym przyciskiem myszy i wybierz **Data Cursor**.

Teraz będziesz mógł obejrzeć wykresy w dziedzinie częstotliwości.

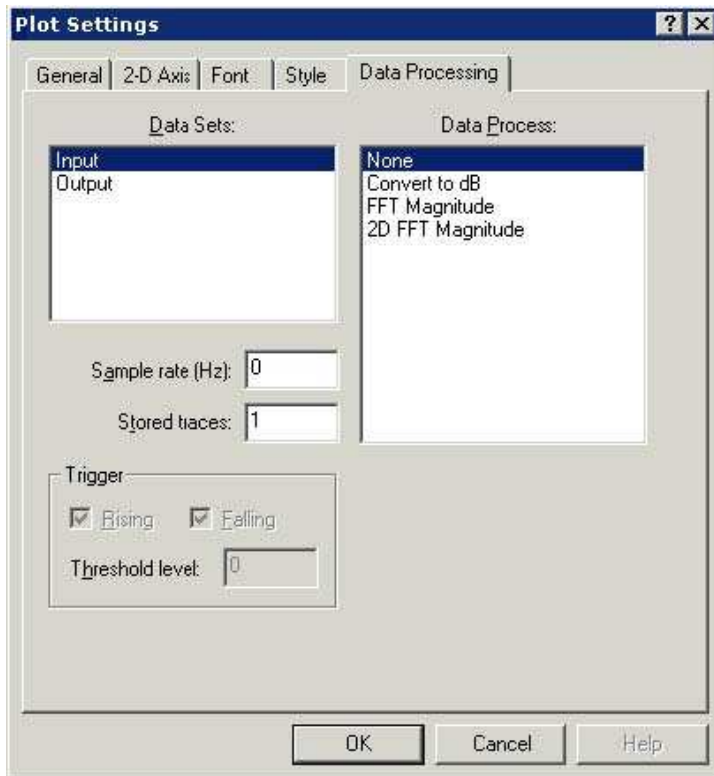
5. Kliknij prawym przyciskiem myszy i wybierz **Modify Settings** by otworzyć okno **Plot Settings**.

6. Następnie:

- a. Kliknij **Data Processing** Rys. 2-32.
- b. W polu **Data Sets** upewnij się, że zaznaczone jest słowo **Input** (domyślnie), a w polu **Data Process** wybierz **FFT Magnitude**.



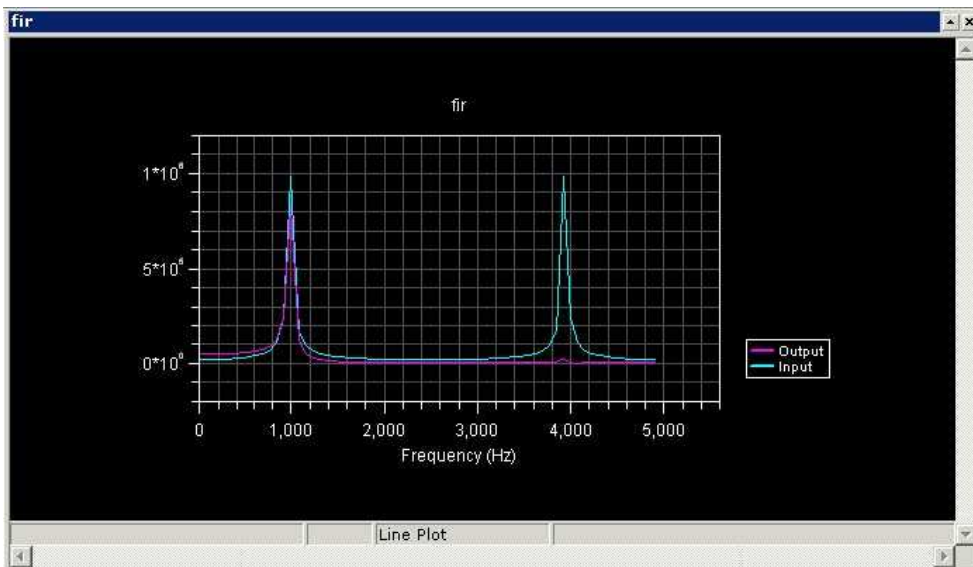
Rys. 2-31 Używanie kursora



Rys. 2-32 Ustawienia wykresu

- c. W **Sample rate (Hz)** wpisz **10000**.
- d. W polu **Data Sets** zaznacz teraz **Output**, a w polu **Data Process - FFT Magnitude**
- e. Kliknij **OK** by opuścić okno **Plot Settings**.

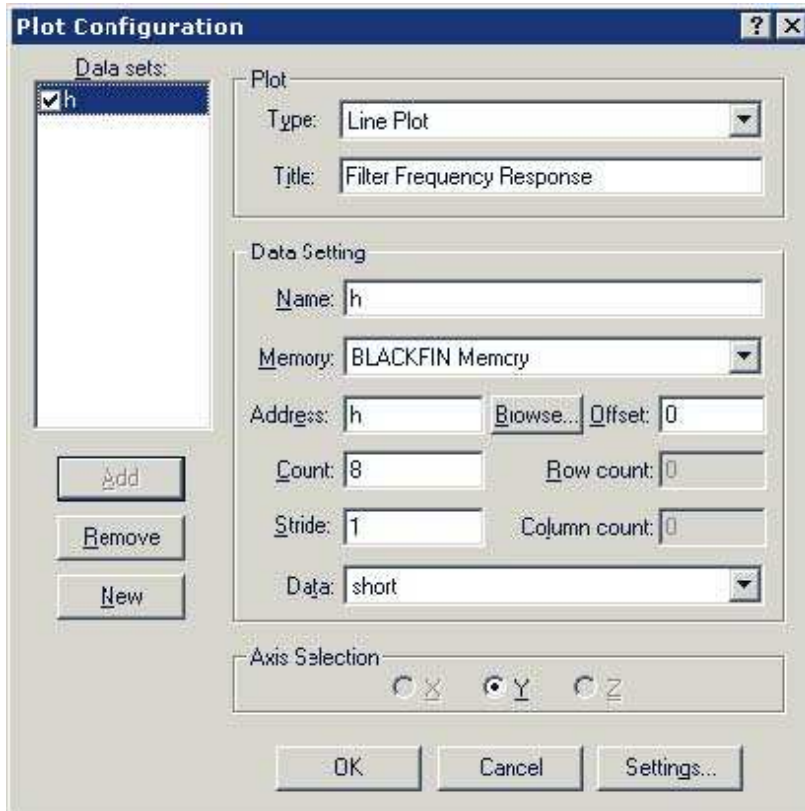
VisualDSP++ wykona teraz Szybką Transformatę Fouriera - Fast Fourier Transform (FFT) na wybranym zestawie danych. FFT pozwala na rozpatrywanie sygnału w dziedzinie częstotliwości [Rys. 2-33](#).



Rys. 2-33 FFT wykonane na zestawie danych

Teraz wykonaj następujące kroki, by obejrzeć działanie filtra FIR w dziedzinie częstotliwości.

1. Z menu **View** wybierz **Debug Windows** i **Plot**. Następnie **New** by otworzyć okno **Plot Configuration**.
2. Ustaw wykreślanie Filter Frequency Response wypełniając pola **Plot** i **Data Setting** jak na [Rys. 2-34](#).

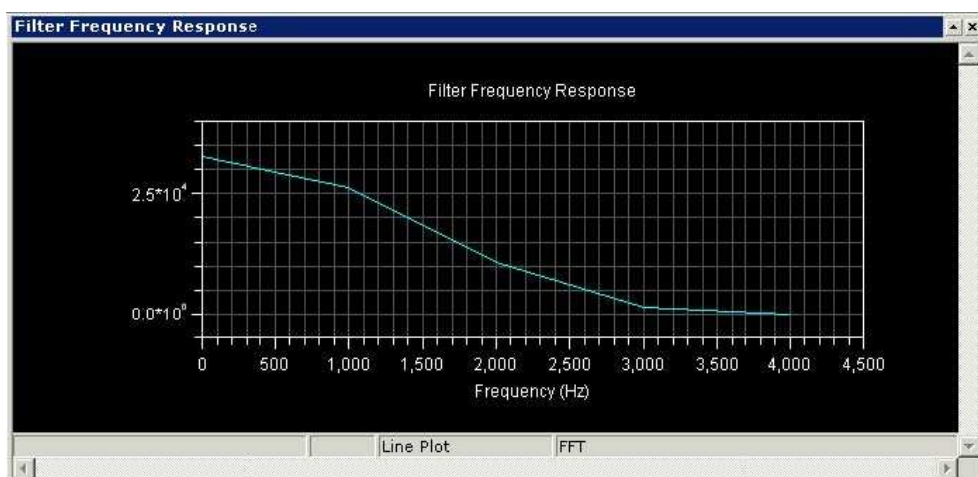


Rys. 2-34 Zestaw danych filtra FIR

3. Kliknij **Add** by dodać dane do **Data sets**.
4. Kliknij **OK** by wprowadzić zmiany.
5. Kliknij prawym przyciskiem myszy w oknie wykresu i wybierz **Modify Settings**.
6. Kliknij zakładkę **Data Processing** – [Rys. 2-32](#) Wypełnij jak poniżej:
  - a. W **Data Sets** wybierz **h**.
  - b. W **Data Process** wybierz **FFT Magnitude**.
  - c. W **Sample rate (Hz)** wybierz **10000**.
  - d. Kliknij **OK** by opuścić okno **Data Processing**.

VisualDSP++ wykona Fast Fourier Transform (FFT) na wybranym zestawie danych i pozwoli przeanalizować odpowiedź filtra w dziedzinie częstotliwości jak na [Rys. 2-35](#).





Rys. 2-35 Odpowiedź filtra

Ten wykres przedstawia filtr dolnoprzepustowy FIR, który usuwa wszystkie składowe powyżej 4000 Hz. Sygnał wyjściowy posiada wtedy tylko harmoniczne poniżej 4 000 Hz. ]

## Ćwiczenie Czwarte: Profilowanie Liniowe

- Załaduj i zdebuguj program FIR z poprzedniego ćwiczenia
- Użyj profilowania liniowego by ocenić wydajność programu i ustalić, które części kodu są najbardziej czasochłonne.

VisualDSP++ obsługuje dwa typy profilowania: liniowe i statystyczne.

- Na symulatorze używa się profilowania liniowego. Licznik okna **Linear Profiling Results** zwiększ swą wartość za każdym razem, kiedy wykonywana jest instrukcja assemblera.
- Profilowania statystycznego używa się z emulatorem JTAG podłączonym do procesora docelowego. Licznik okna **Statistical Profiling Results** oparty jest na losowym badaniu licznika programu.

### Krok 1: Załaduj program FIR

1. Zamknij wszystkie okna oprócz **Disassembly** i **Output**.

2. Z menu **File** wybierz **Load Program** lub kliknij .

Pojawi się okno **Open a Processor Program**.

3. Wybierz poniższy program

- Otwórz folder `Analogue Devices` i kliknij dwukrotnie:

`Lab1\fir`

- Kliknij dwukrotnie podkatalog `Debug`.
- Kliknij dwukrotnie `FIR.DXE` by załadować i uruchomić program FIR.

Jeżeli VisualDSP++ nie otworzy okna edytora (Rys. 2-37), kliknij prawym klawiszem myszy w oknie **Disassembly** i wybierz **View Source**.

## Krok 2: Otwórz okno Profiling

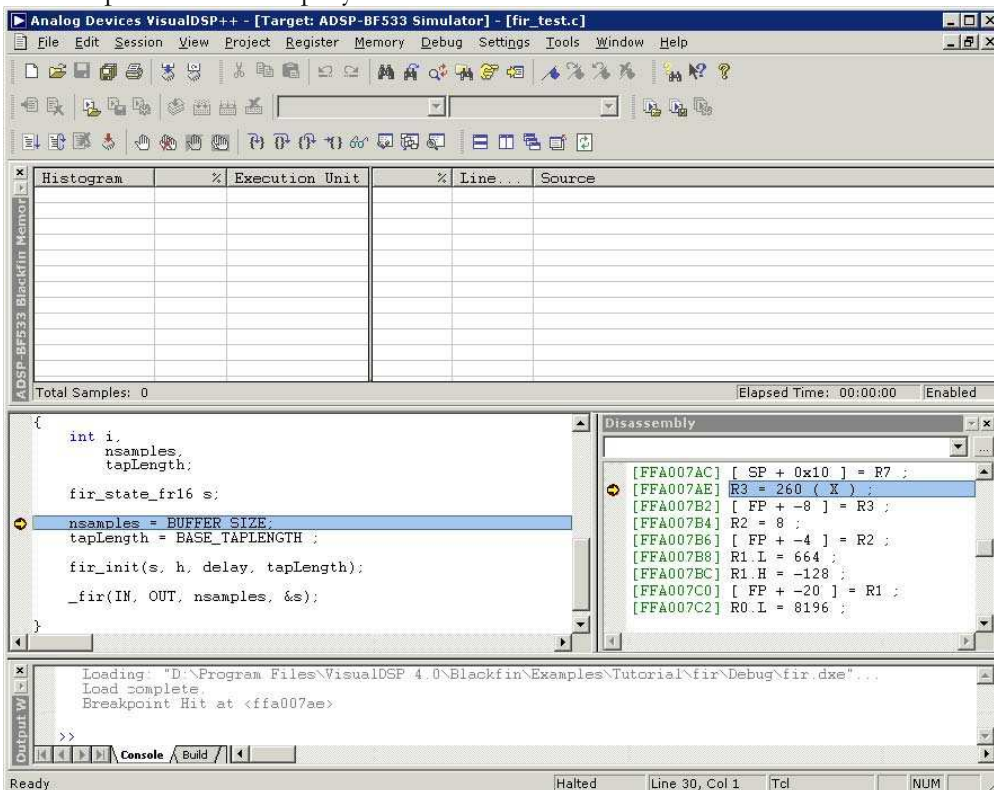
By otworzyć okno **Linear Profiling Results**:

1. Z menu **Tools** wybierz **Linear Profiling** i **New Profile**.



Rys. 2-36 Ustawianie Profilowania Liniowego

2. Kliknij na pasku tytułowym okna profilowania i przeciągnij je w górę głównego okna VisualDSP++ jak na Rys. 2-37. Zapewnisz sobie lepszy widok.




Rys. 2-37 Okno profilowania liniowego

Okno **Linear Profiling Results** jest początkowo puste. Profilowanie liniowe jest wykonywane podczas

pracy programu FIR. Po uruchomieniu programu i zebraniu danych, w oknie pojawią się rezultaty sesji profilowania.

### Krok 3: Zbieranie i badanie danych Profilowania Liniowego

1. Wciśnij **F5** lub kliknij  by uruchomić w całości program.

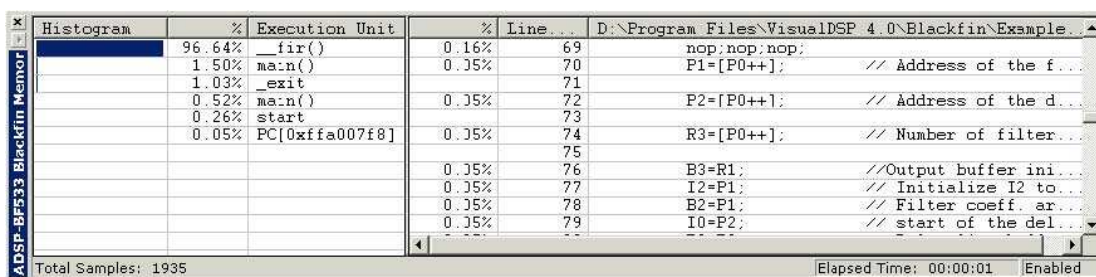
Po zatrzymaniu programu, jego profil liniowy pojawi się w oknie **Linear Profiling Results**.

2. Zbadaj rezultaty.

Okno **Linear Profiling Results** jest podzielone na dwa trójkolumnowe pola.

Lewe pole prezentuje wyniki profilowania – czas wykonania każdej funkcji/adresu w procentcie czasu całkowitego.

Dwukrotne kliknięcie linii z daną funkcją pozwala przejrzeć plik źródłowy, który tę funkcję zawiera. Na przykład, podwójny klik na funkcji `fir` spowoduje wyświetlenie się w prawym polu pliku źródłowego assemblera (`fir.asm`) – Rys. 2-38.



Histogram	%	Execution Unit	%	Line	Code
96.64%	0.16%	fir()	0.16%	69	nop:nop:nop;
1.50%	0.35%	main()	0.35%	70	P1=[P0++]; // Address of the f...
1.03%		_exit		71	
0.52%	0.35%	main()	0.35%	72	P2=[P0++]; // Address of the d...
0.26%		start		73	
0.05%	0.35%	PC[0xffa007f8]	0.35%	74	R3=[P0++]; // Number of filter...
				75	
	0.35%		0.35%	76	B3=R1; //Output buffer ini...
	0.35%		0.35%	77	I2=P1; // Initialize I2 to...
	0.35%		0.35%	78	B2=P1; // Filter coeff. ar...
	0.35%		0.35%	79	I0=P2; // start of the del...

Rys. 2-38 Rezultaty profilowania liniowego

Wartości lewego pola mają następujące znaczenie:

**Histogram** Graficzna reprezentacja procentu czasu potrzebnego na wykonanie poszczególnych fragmentów kodu w stosunku do całkowitego czasu wykonania programu. Im dłuższy słupek, tym więcej czasu potrzeba na wykonanie konkretnego fragmentu. Okno **Linear Profiling Results** zawsze sortuje fragmenty poczynając od najbardziej czasochłonnego.

**%** Liczbowa reprezentacja informacji zawartej w histogramie. Można podejrzeć tę wartość jako bezwzględną liczbę próbek klikając prawym przyciskiem myszy w oknie **Linear Profiling Results** i wybierając **View Sample Count** z menu.

**Execution Unit** Określa fragment programu, do którego należą dane próbki. Jeżeli instrukcje są zawarte w funkcji w C lub C++, jednostką wykonawczą (**Execution Unit**) jest nazwą tej właśnie funkcji. Dla instrukcji nie odnoszących się do żadnej konkretnej nazwy, jak na przykład ręcznie wpisane fragmenty w assemblerze lub pliki źródłowe bez informacji z debugera, tą wartością będzie adres w formie `PC[xxx]`, gdzie `xxx` jest adresem instrukcji.

Jeżeli instrukcje są częścią pliku asemblera, jednostką wykonawczą będzie albo funkcja asemblera, albo plik asemblera z numerem linii w nawiasach.

Na Rys. 2-38 lewy panel ukazuje, że funkcja `fir` zużywa ponad 93% całkowitego czasu wykonania programu. Prawy panel (źródło) na Rys. 2-39, przedstawia procent czasochłonności każdej linii funkcji `fir`.

%	Line	D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tu...
	66	__fir :
	67	
0.05%	68	P0=[SP+12]; // Address of the filt...
0.16%	69	nop;nop;nop;
0.05%	70	P1=[P0++]; // Address of the filte...
	71	
0.05%	72	P2=[P0++]; // Address of the delay...
	73	
0.05%	74	R3=[P0++]; // Number of filter coe...
	75	
0.05%	76	B3=R1; //Output buffer initial...
0.05%	77	I2=P1; // Initialize I2 to the...
0.05%	78	B2=P1; // Filter coeff. array ...
0.05%	79	I0=P2; // start of the delay li...
0.05%	80	B0=P2; // Delay line buffer is...
0.05%	81	I1=P2; // start of the delay li...
0.05%	82	B1=P2; // Delay line buffer is...
	83	
0.05%	84	I3=R1;
0.05%	85	P1=R2;
0.05%	86	P2=R3;
	87	
0.05%	88	R2=R2+R2;
0.05%	89	CC=BITTST(R3,0); //Check if the number o...
0.05%	90	R3=R3+R3; //As the filter coeff. ...
0.05%	91	I2=R3; //Initialize the filter...
0.05%	92	P0=R0; // Address of the input...
	93	
0.26%	94	IF !CC JUMP FIR_CONTINUE (BP);
	95	R3+=2; //Make the filter taps ...
	96	I2=R3;
	97	NOP;NOP;NOP;NOP;
	98	I2-=2; // Location where zero...
	99	R0=0;
	100	W[I2++]=R0.L; //Set the last filter ...
	101	//force the number of filt...

Rys. 2-39 Profil liniowy dla `fir.asm`

## Ćwiczenie Piąte: Optymalizacja na podstawie Profilingu

### Wstęp do PGO

Optymalizacja na podstawie Profilingu (PGO) jest techniką optymalizacji używaną do zbierania informacji o profilu do kierowania optymalnymi decyzjami kompilatora

Tradycyjnie kompilator kompiluje każdą funkcję tylko raz i próbuje wytworzyć kod, który wystąpi dobrze w większości przypadków. Kompilator musi podjąć decyzje jak najlepiej wygenerować kod. Na przykład otrzymując konstrukcję `if...then...else` kompilator musi zdecydować się czy najbardziej wspólny wybór będzie poprzez `or` czy `else`. Możesz zaoferować surowe wytyczne – kompilacja pod kątem szybkości lub wielkości kodu - ale zwykle kompilator wykonuje kompilacje przy domyślnych ustawieniach.

Wraz z PGO kompilator podejmuje te decyzje opierając się na danych zgromadzonych podczas poprzedniego wykonania generowanego kodu. Ten proces pociąga za sobą następujące kroki.

1. Kompilacja aplikacji w celu zebrania informacji o profilu
2. Uruchomienie aplikacji w sesji symulatora używając reprezentatywnego zbioru danych.  
Symulator gromadzi dane profilu wskazujące gdzie aplikacja spędza najwięcej czasu.
3. Ponowna kompilacja aplikacji z użyciem zebranych danych profilu

Kompilator używa zgromadzone informacje chętniej niż domyślne ustawienia aplikacji, aby podjąć decyzje o względnym znaczeniu części aplikacji

Dane profilu zebrane podczas symulacji są zapisywane do pliku o rozszerzeniu .PGO. Można przetworzyć wielorakie dane dla pokrycia widma potencjalnych danych i utworzyć oddzielny plik .PGO dla każdego zbioru danych.

Faza ponownej kompilacji może przyjąć wiele plików .PGO jako pliki wejściowe.

Procedura postępowania przy korzystaniu z PGO:

- 1 Utwórz aplikację wykorzystującą PGO
- 2 Ustaw jedną lub więcej strumieni w symulatorze by dostarczyć zestaw wprowadzanych danych, które przedstawiałyby co aplikacja by zobaczyła w rzeczywistym środowisku docelowym
- 3 Wskaż dla symulatora, aby wygenerował plik .PGO z wyszczególnioną nazwą pliku.
- 4 Załaduj i uruchom aplikację by wytworzyć plik .PGO
- 5 Wykonaj Rebuild aplikacji i dołącz wszystkie pliki .PGO do kompilatora, które pozwolą wygenerować PGO jako rezultat optymalnej aplikacji.

W tym ćwiczeniu wykonaj:

- Załaduj projekt wykorzystujący PGO w środowisku VisualDSP++
- Utwórz dane dla profilu - kierowanej optymalizacji
- Dołącz strumienie danych wejściowych
- Utwórz plik .PGO przez wykonanie projektu z danymi wejściowymi.
- Skompiluj ponownie projekt przez użycie pliku .PGO by optymalizować kod
- Uruchom zoptymalizowaną wersję projektu z tymi samymi danymi wejściowymi
- Porównaj czas wykonania wszystkich trzech wykonań
- Zaprezentuj uzyskane wyniki prowadzącemu

Pliki używane w tym ćwiczeniu znajdują się w plikach .

Pliki używane w tym ćwiczeniu znajdują się w podkatalogach: pgo i pgo-surowe


## Krok 1: Ładuj Projekt

Aby otworzyć projekt VisualDSP++:

- 1 Uruchom VisualDSP++ i podłącz się do symulatora sesji ADSP-21161

## 2 Otwórz projekt PgoExample.dpj.

Ten projekt zawiera plik C, `PgoExample.c`. Gdy uruchomisz program, zacznie czytać dane z adresu i zacznie zliczać numery parzystych i nieparzystych wartości. Zliczanie zakończy się dyrektywą `if...then...else`. Jeśli większość wartości jest parzystych, program spędzi więcej czasu w `then...` branch. Normalnie kompilator nie ma możliwości określenia który skok będzie używany częściej. Przez użycie PGO kompilator będzie mógł określić, który skok jest częściej używany i zoptymalizuje następny kod.

Ten projekt ra również skrypty Visual Basic które demonstrują jak użyć Automatykacji API VisualDSF wykonać PGO.

Trzy pliki są używane jako wejściowe do programu C. Te proste pliki tekstowe zawierają listy wartości.

- `Dataset_1.dat` zawiera 128 parzystych (50%) i 128 nieparzystych wartości (50%).
- `Dataset_2.dat` zawiera 192 parzystych (75%) i 64 nieparzystych wartości (25%).
- `Dataset_3.dat` zawiera 256 parzystych (100%) i 0 nieparzystych wartości (0%).

By obejrzeć te pliki wybierz Open z menu File w VisualDSP++. Dwie możliwe wartości we wszystkich trzech plikach są albo `0x01`, albo `0x02`. Każdy plik zawiera 256 wartości

W tym ćwiczeniu przyjmij że ten program będzie używany w rzeczywistym świecie i że możesz oczekiwać podobnych wartości wejściowych z rzeczywistego świata.

Patrząc na kod napisany w C i potencjalne wejście możesz zobaczyć, że wykonywany program będzie spędzał więcej czasu w konstrukcji `else...` branch niż w `then...` branch. Bez użycia PGO kompilator nie jest w stanie podjąć takiej decyzji. Domyślnie będzie zakładał konstrukcje `then...` branch by wykonać ją najczęściej i skompiluje kod bez optymalizacji czasu wykonania.

Ponieważ przykład programu i wejście są bardzo proste, możesz rozwiązać problem poprzez wykonanie kilku drobnych zmian w kodzie. Ręczne udoskonalanie dużego programu by przyspieszyć czas wykonania zajęło by zbyt dużo czasu i musiałbyś sam przeanalizować próbki wejściowe. PGO dostarcza to szybko i w łatwy sposób by umożliwić kompilatorowi wykonanie tych udoskonaleń za ciebie.

## Krok 2: Konfiguracja zbioru danych

Pierwszym krokiem w procesie PGO jest utworzenie zbioru danych (data set) – zbiór próbek wejściowych dla programu poddawanego optymalizacji. Zbiór danych podawany jest na wejście wykonywanego programu i to wejście powoduje że program musi być wykonywany wzdłuż określonej ścieżki. Niektóre ścieżki będą używane częściej niż inne. Ta informacja jest rejestrowana przez symulator i zapisywana w pliku `.PGO` dla późniejszej optymalizacji przez kompilator. Najbardziej wspólne ścieżki będą zoptymalizowane by były wykonywane najszybciej a mniej uczęszczane ścieżki będą wykonywane wolniej.

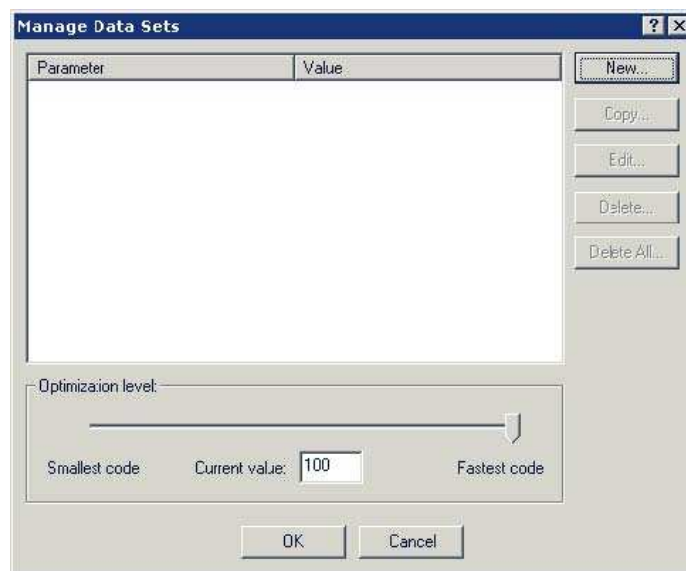


By utworzyć pierwszy z trzech zbiorów danych dla tego ćwiczenia:

1. Z menu Tools wybierz PGO i wtedy Manage Data Sets tak jak na rysunku 3-1.



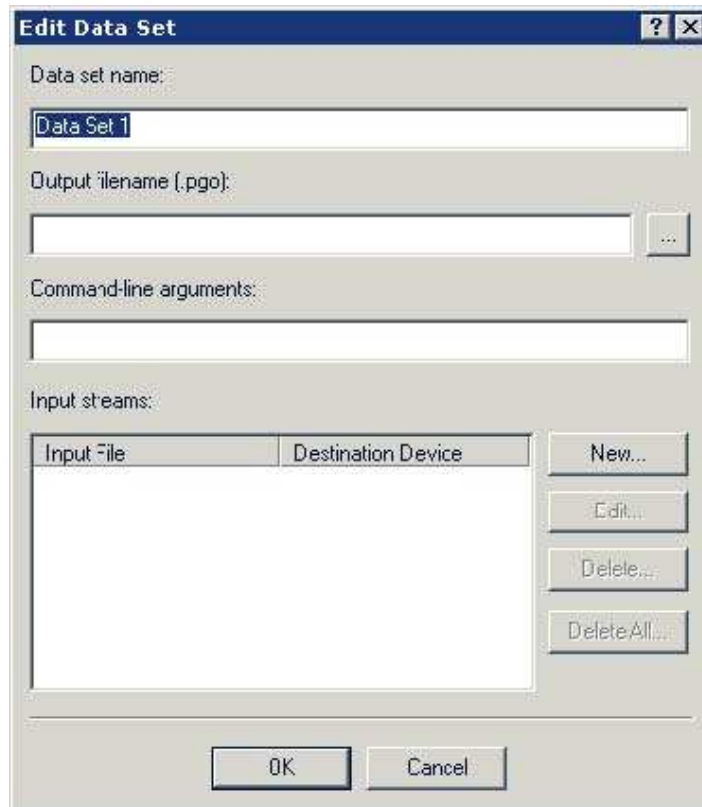
Rys. 3-1. Opcje menu Manage Data Sets



Rys. 3-2 Okno dialogowe Manage Data Sets.

Jest to okno dialogowe do zarządzania zbiorem danych. Zwróć uwagę na stopień optymalizacji ustawiany suwakiem. Ta kontrola pozwala na ręczne sterowanie optymalizacją. Przesunięcie suwaka w lewo do końca umożliwi budowę najmniejszego kodu kosztem szybkości wykonywania. Przesunięcie suwaka w prawo do końca umożliwi budowę szybko wykonywanego kodu kosztem wielkości kodu. Ustawienie suwaka w środkowej pozycji jest kompromisem między minimalnym kodem a optymalizacją szybkości kodu. Ustaw suwak w skrajnym prawym położeniu.

2. Naciśnij przycisk New aby otworzyć okno dialogowe Edit Data Set pokazane na rys. 3-3



Rys. 3-3 okno dialogowe Edit Data Set

3. Zamień domyślną nazwę zbioru danych na bardziej opisującą. Ponieważ pierwszy plik danych zawiera równą liczbę parzystych i nieparzystych wartości, nazwij go 50% Even - 50% Odd.

4. Określ nazwę pliku wyjściowego (gdzie informacja o optymalizacji zostanie zapisana na podstawie zbioru danych). Informacja optymalizacji jest zapisywana w pliku o rozszerzeniu .PGO.

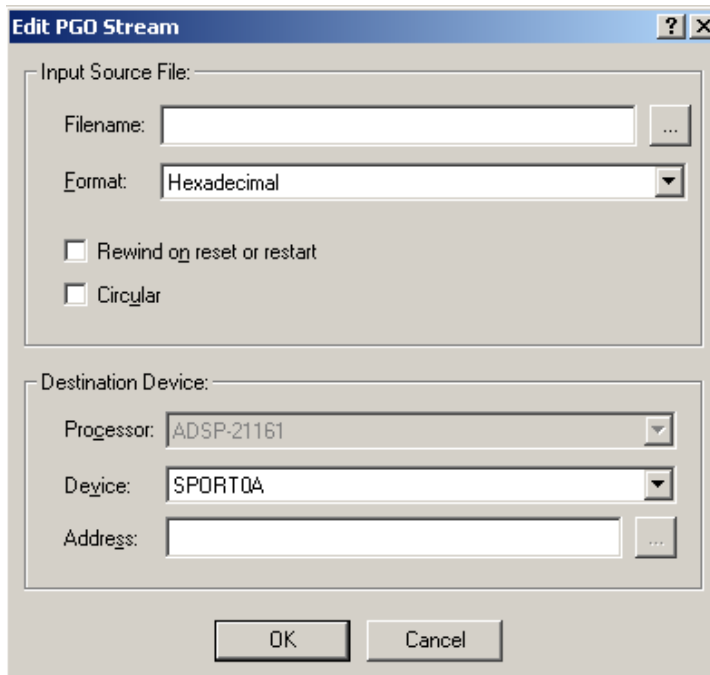
Zapisz plik pod nazwą `dataset_1.pgo`. Plik zostanie zachowany w katalogu projektu. Aby zapisać plik w innym miejscu podaj pełną ścieżkę.

Teraz połącz strumień wejściowy ze zbiorem danych.

### Step 3: Dołączanie strumienia wejściowego

W tym kroku dołącz strumień wejściowy ze zbiorem danych.

1. Naciśnij przycisk New by otworzyć okno dialogowe Edit PGO Stream pokazane na rys. 3-4.



Rys. 3-4. okno dialogowe Edit PGO Stream

Wejściowy strumień odwzorowuje dane z pliku do urządzenia docelowego. W tym ćwiczeniu strumień wejściowy odwzorowuje trzy pliki z danymi do symulatora. Wejściowy strumień dostarcza do programu dane do wejścia jeśli są potrzebne podczas wykonywania.

2. Kompletny opis okna strumienia wejściowego (Input Source File) przedstawia tabela 3-1.

Table 3-1. Input Source File Group Box Settings

Pole/Kontrola	Opis działania/Wartość
Filename	Określa nazwę pliku poprzez wciśnięcie przycisku przeglądania i wybranie pliku źródłowego dataset_1.dat z katalogu pgo.
Format	Dane w tym pliku są zapisane w formacie szesnastkowym, więc pozostaw ustawienia takie jakie są.
Rewind on reset or restart	Wybierz tę opcję. Kiedy uruchomisz program ze strumieniem wejściowym, program może albo też nie będzie działać z wszystkimi danymi ze strumienia. Jeśli program napotka reset albo restart przed przetworzeniem całego strumienia danych i ta opcja będzie uaktywniona, następnego wykonanie programu zacznie się od początku strumienia wejściowego, W przeciwnym przypadku przetwarzanie danych zacznie się w miejscu w którym się zakończyło.

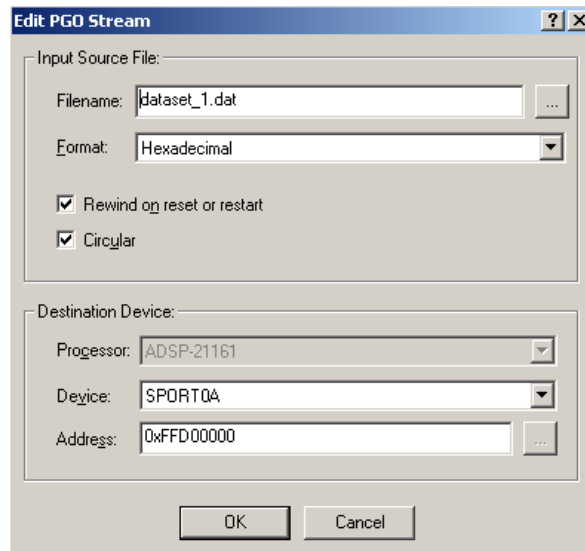
Circular	Wybierz tę opcję. Umożliwia ona programowi czytanie z wejściowego strumienia danych wiele razy podczas pojedynczego wykonania.
----------	--

3. Okno dialogowe Destination Device określa gdzie dane ze strumienia wejściowego są przesyłane. Szczegóły w tabeli 3-2.

Tabela 3-2. Ustawienia okna dialogowego Destination Device.

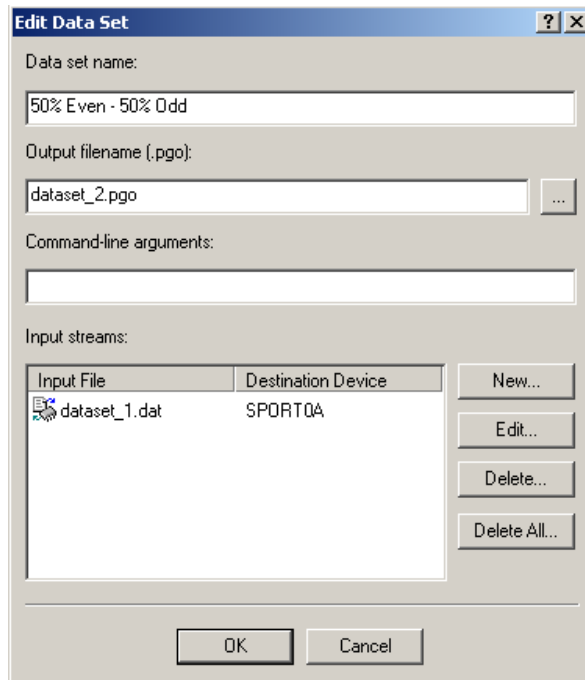
Pole/Kontrola	Opis działania/Wartość
Processor	To pole pozwala określić peryferia w innym procesorze albo jako urządzenie docelowe. Ten podręcznik zakłada, że jesteś podłączony do pojedynczej sesji procesora, więc to pole jest zablokowane
Device	To pole pozwala wybrać jakikolwiek strumień urządzenia obsługiwany przez symulator docelowy jako miejsce docelowe. Urządzenie może przydzielić przestrzeń adresową albo różne peryferia. Dostępne urządzenia zależą od użytego procesora. Więcej informacji na temat urządzeń znajdziesz w instrukcji dla danego procesora. Ten program czyta strumień wejściowy z pamięci, więc pozostaw to ustawienie takie jakie jest.
Address	Określa gdzie w pamięci wejście będzie wysyłane. Odtąd program w tym ćwiczeniu czyta dane z adresu 0xFFD00000 (odnieś się do PgoExample.c), wprowadź tą wartość.

Wypełnione okno dialogowe powinno wyglądać jak na rys. 3-5.



Rys. 3-5 Konfiguracja strumienia PGO.

4. Wciśnij OK by powrócić do okna Edit Data Set. Skonfigurowane okno powinno wyglądać jak na rys. 3-6.



Rys. 3-6 konfiguracja strumienia danych.

5. Wciśnij OK by zachować ustawienia i zamknąć okno. Teraz musisz utworzyć pozostałe dwa strumienie danych.

## Krok 4: Dodatkowa konfiguracja strumienia danych

By utworzyć pozostałe dwa strumienie danych powtórz kroki użyte podczas tworzenia pierwszego strumienia wejściowego i zamień odpowiednie pliki albo użyj przycisku Copy.

Następujące kroki wyjaśnią jak użyć przycisku Copy by utworzyć strumień wejściowy

1. Podświetl 50% Even - 50% odd, wciśnij przycisk Copy.

Otworzy się okno Edit Data Set z informacją dla strumienia 50% Even - 50% odd. Wciśnięcie OK. wykona kopie strumienia 50% Even - 50% odd. Jednak w tym ćwiczeniu zredagujesz tylko strumień danych.

2. W polu Data set name określ nazwę dla nowego zbioru danych.

Drugi zbiór wejściowy zawiera trzy razy więcej wartości parzystych niż nieparzystych, więc nadaj mu nazwę 75% Even - 25% Odd.

3. W polu Output filename wpisz dataset\_2.pgo by zachować .PGO w katalogu projektu

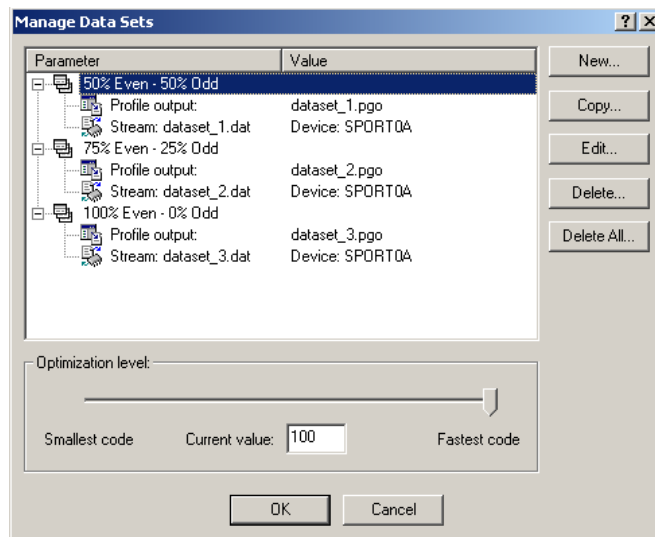
By zachować plik w innym miejscu wciśnij przycisk (⋮) by określić pełną ścieżkę.

4. W oknie Input streams podświetl `dataset_1.dat` w kolumnie Input File i wciśnij przycisk Edit.
5. Wciśnij przycisk (⋮) by zmienić Input Source File z `dataset_1.dat` na `dataset_2.dat`
6. Wciśnij OK. by powrócić do okna Edit Data Set

Druga dana jest teraz wprowadzona.

7. Wciśnij OK. by powrócić do Manage Data Sets
8. Utwórz trzeci zbiór danych od początku lub zmodyfikuj kopie jednego z istniejących zbiorów danych.

Upewnij się, że używasz plików `dataset_3.pgo` i `dataset_3.dat`. Trzeci zbiór danych zawiera tylko parzyste wartości więc nazwij go jako `100% Even - 0% Odd`. Kiedy skończysz rozwiń listę zbiorów danych w oknie Manage Data Sets i sprawdź z danymi na rys. 3-7.



Rys 3-7 Zbiory danych

Jeśli twoje zbiory danych zgadzają się z rys. 3-7 to twoje zbiory danych wymagają optymalizacji przez program.

9. Wciśnij OK by zachować zbiory danych i zamknąć okno dialogowe.

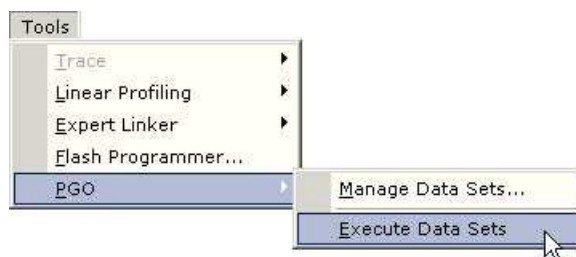
Teraz jesteś gotowy by utworzyć plik `.PGO`.

## Krok 5: Tworzenie pliku PGO i optymalizacja programu

Teraz kiedy masz skonfigurowane zbiory danych jesteś gotowy do optymalizacji programu.

Z menu Tools wybierz PGO a następnie Execute Data Sets tak jak to pokazano na rys. 3-8.





Rys.3-8

Kilka rzeczy dzieje się podczas wykonywania procesu. Pierwsze, projekt jest budowany z parametrem `-pguide` który umożliwia zbieranie danych dla PGO które są później ponownie wykorzystywane przez kompilator. Kompilator tworzy domyślnie założenie o tym która sekcja kodu będzie najczęściej wykonywana. Następnie rezultatem wykonania jest kolejne uruchomienie z każdym ze zbiorów danych. Podczas gdy program jest wykonywany symulator monitoruje ścieżki pokonywane przez program i liczbę cykli użytych na wykonanie. Jak ustalono wcześniej informacje są zapisywane w pliku `.PGO` który wyszczególniłeś kiedy tworzyłeś każdy ze zbiorów danych.

Gdy program został uruchomiony z każdym ze zbiorów danych, projekt był kompilowany ponownie. Jednak tym razem kompilator używa informacje zawarte w pliku `.PGO` do optymalizacji rezultatów wykonania. Ta optymalizacja wykonania jest uruchamiana z wejściem dostarczanym przez każdy zbiór danych i ponownie symulator monitoruje każde wykonanie

Teraz jesteś gotowy by zbadać wyniki optymalizacji.

## Krok 6: Porównanie czasu wykonania

Kiedy wykonywanie zostanie zakończone raport o rezultatach optymalizacji PGO jest generowany w postaci dokumentu XML i wyświetlany w oknie przeglądarki. Ten plik znajduje się w katalogu `pgo\debug` i został nazwany `PgoReport.date and time.xml` (na przykład `PgoReport.20031027145428.xml`)

Początek raportu zawiera nagłówek pokazany na rys. 3-9

### Profile Guided Optimization Results

```
Generated on: Fri Jan 20 01:08:25 2006
Application: C:\PGO\Lab\pgo\Debug\PgoExample.dxe
Project: C:\PGO\Lab\pgo\pgoexample.dpj
Optimization level: 100
Average cycle reduction: 5.24%
```

Rys. 3-9 Rezultat PGO – nagłówek raportu

Nagłówek dostarcza podstawowe informacje takie jak nazwa projektu, lokalizacja, i kiedy raport został wygenerowany. Umieszczony jest również stopień optymalizacji (który podałeś za pomocą suwaka w oknie dialogowym `Manage Data Sets` pokazany na rys. 3-2) i średni wyniki. Uzyskany wynik jest różnicą w całkowitym cyklu zliczania we wszystkich wykonaniach przed i po optymalizacji.

Przeciętny wynik otrzymany na twojej maszynie może się zmienić nieznacznie od wyników pokazanych na rys 3-9



Po nagłówku następują informacje o zbiorach danych (zobacz rys. 3-10)

```
Data Set: 50% Even - 50% Odd
Command line:
Input stream: File: C:\PGO\Lab\pgo\dataset_1.dat
              Device: SPORT0A
              PGO output: C:\PGO\Lab\pgo\dataset_1.pgo
Before optimization: 4881 cycles
After optimization: 4625 cycles
Cycle reduction: 5.24%
```

```
Data Set: 75% Even - 25% Odd
Command line:
Input stream: File: C:\PGO\Lab\pgo\dataset_2.dat
              Device: SPORT0A
              PGO output: C:\PGO\Lab\pgo\dataset_2.pgo
Before optimization: 4881 cycles
After optimization: 4625 cycles
Cycle reduction: 5.24%
```

```
Data Set: 100% Even - 0% Odd
Command line:
Input stream: File: C:\PGO\Lab\pgo\dataset_1.dat
              Device: SPORT0A
              PGO output: C:\PGO\Lab\pgo\dataset_3.pgo
Before optimization: 4881 cycles
After optimization: 4625 cycles
Cycle reduction: 5.24%
```

Informacje o pliku, włączając nazwy zbiorów danych, nazwy plików strumieni wejściowych i nazwę pliku wynikowego .PGO są umieszczone na początku. Następnie są pokazane rezultaty optymalizacji. Ilość cykli potrzebnych na wykonanie oryginalnego programu z tymi zbiorami danych (przed optymalizacją) a następnie liczba cykli potrzebnych do wykonania zoptymalizowanego programu z tymi zbiorami (po optymalizacji). Zauważ, że liczba cykli może być różna na różnych maszynach.

Ostatecznie procent różnicy pomiędzy dwoma programami (rezultatami) jest wyszczególniony. Dodatni procent wskazuje że optymalizowany projekt działa szybciej od oryginalnego.

Sekcja Execution Output w dzienniku pojawia się pierwsza. Rysunek 3-11 pokazuje kawałek Execution Output.

```
Execution Output
-----
Executing PGO Data Sets
-----

Building application with PGO support...
    Build complete.

Profiling Data Set: "50% Even - 50% Odd"...
    Loading application: PgoExample.dxe
    Setting command line:
    Creating input stream: File: dataset_1.dat -> Device: SPORT0A
    Setting PGO output: dataset_1.pgo
    Running application: PgoExample.dxe
    Profile results: 4881 cycles
```

Rys. 3-11 Rezultaty PGO - przykład Execution Output.

Ta informacja jest wynikiem który pojawia się w konsoli okna wyjściowego podczas trwania PGO. Wyjście wyświetla podstawowe zdarzenia które pojawiają się podczas wykonywania.

Sekcja Build Output pojawia się następnie na dole raportu. Ta sekcja zawiera przebieg budowy każdej budowy. Rys. 3-12 przedstawia przykładowy przebieg budowy.

```
Pre-Optimization Build Output
-----
-----Configuration: PgoExample - Debug-----
.\PgoExample.c
Linking...
Build completed successfully.

Post-Optimization Build Output
-----
-----Configuration: PgoExample - Debug-----
.\PgoExample.c
Linking...
Build completed successfully.
```

Rys . 3-12 Rezultaty – przykładowy przebieg budowy.

Ta informacja jest wynikiem, który został wyświetlony w widoku Build w oknie Output podczas trwania PGO.

Ta informacja wyjściowa pokazuje jak efektywny może być PGO. Jak pokazano na rys. 3-9 optymalizowanie wykonania uzyskało w przybliżeniu 18% mniej cykli niż oryginalne wykonanie. Zysk w wynikach jest znaczący, a zwłaszcza daje łatwość jego osiągnięcia.

**To jest koniec wprowadzenia i ćwiczenia 1. Gratulujemy.**