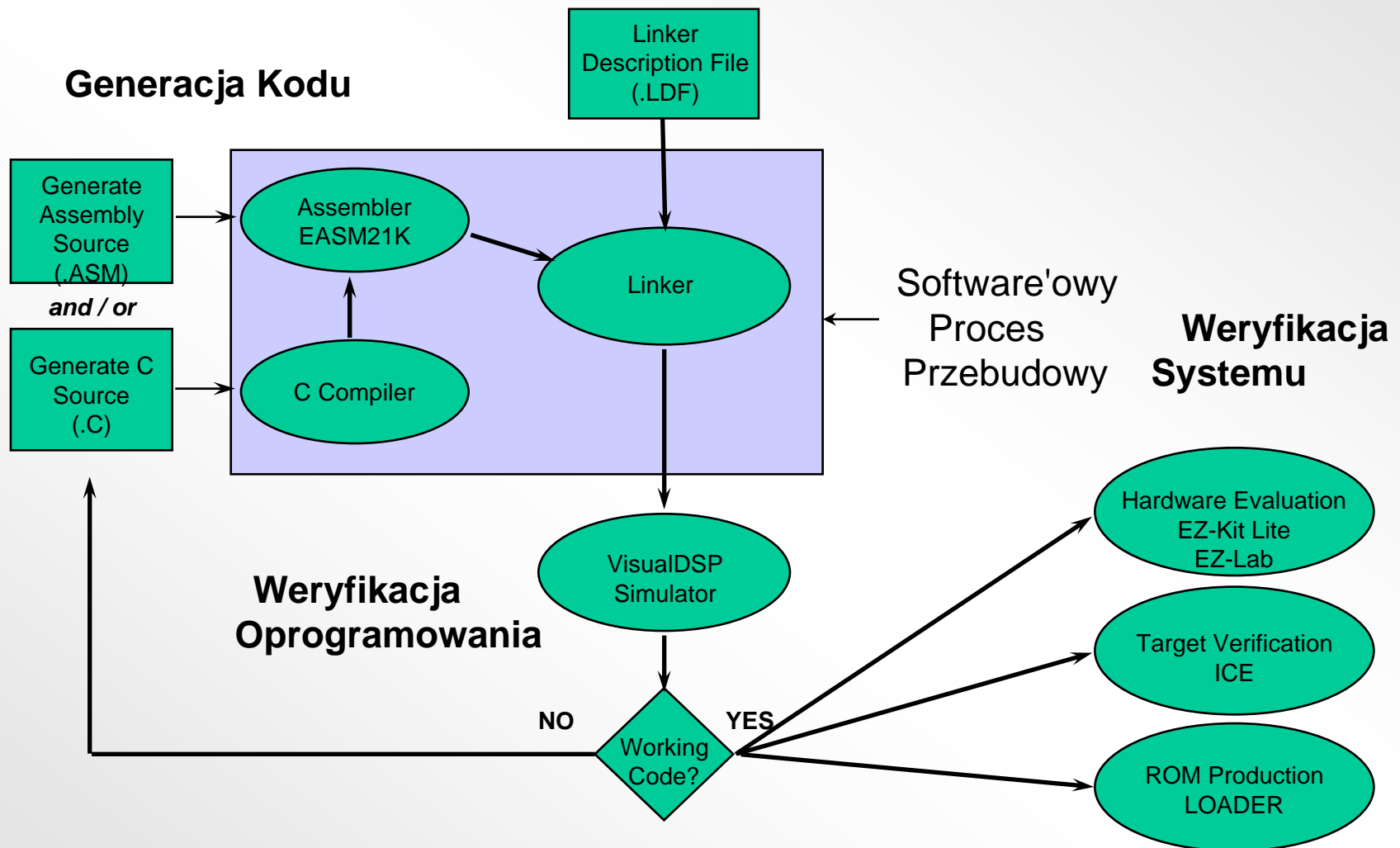


Asemlblacja Kodu w VisualDSP++

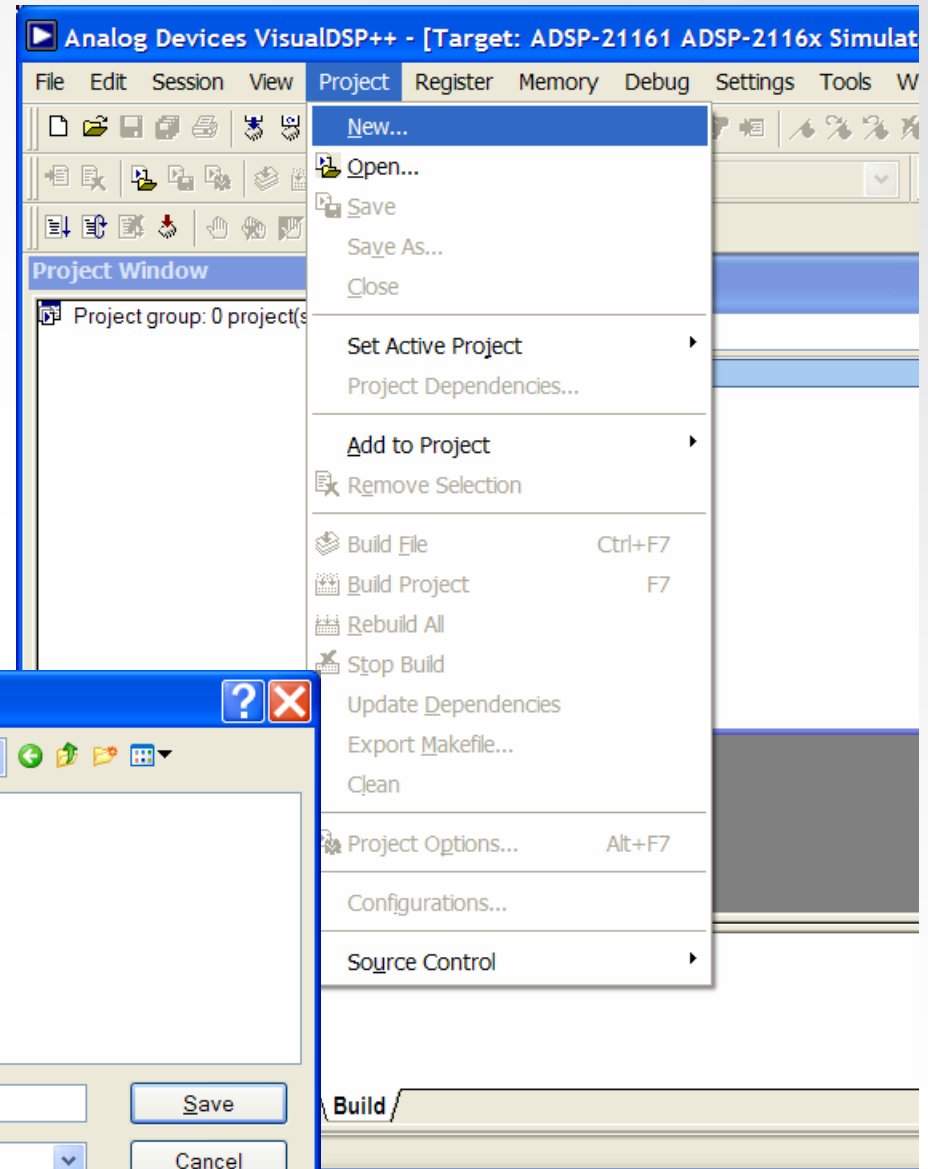
Sekcja 6

Programowe Tworzenie Potoku



Budowa Projektu

- **Tworzenie projektu**
 - przebudowa projektu w VisualDSP++ jest zawarta w projekcie.
 - Plik projektu (.DPJ) zawiera informacje o przebudowanym programie: liste plików źródłowych, opcje ustawień narzędzi przebudowy



Opcje Projektu

Project Options

Project General VIDL Compile Assemble Link Split Load

Target

Processor: ADSP-21161

Type: DSP executable file

Name: Effects Processor

Tool Chain

Compiler: C/C++ Compiler for SHARC (210xx/211xx/212xx)

Assembler: ADSP-21xxx Family Assembler

Linker: ADSP-21xxx Family Linker

Loader: ADSP-21xxx Family Loader

Splitter: ADSP-21xxx Family Splitter

Settings for configuration: Debug

OK Cancel

Wybierz procesor

Ustaw typ pliku .exe, wymagane przez debugger

Zatwierdź

Pojawi się nowe okno. Jeśli nie używasz systemu operacyjnego VDK, wcisnij **NO**.

Kroki Tworzenia Projektu

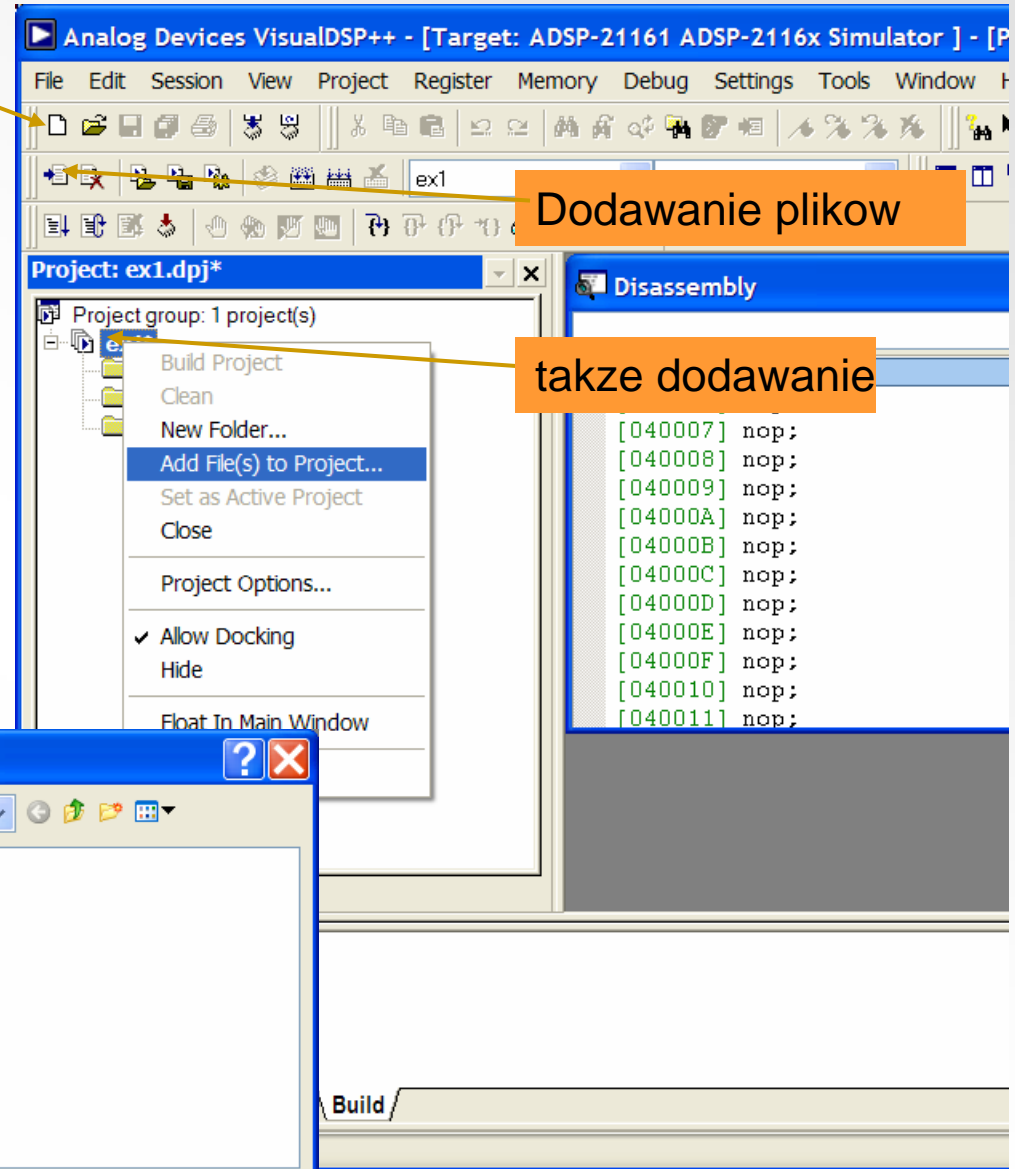
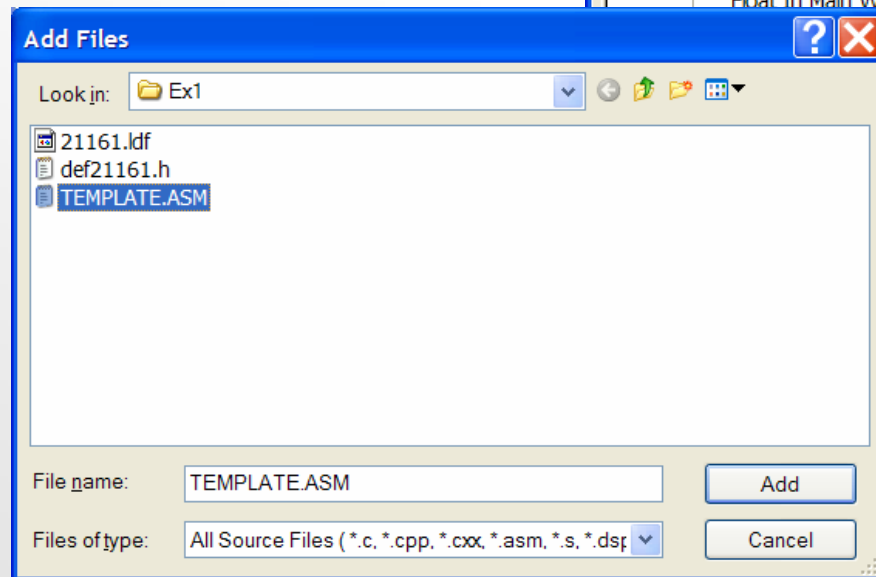
- **Tworzenie plików zrodlowych**

- Projekt zawiera jeden lub wiecej plikow zrodlowych C, C++, lub asm.
- Po stworzeniu projektu i zdefiniowaniu procesora, dodaj nowe lub istniejace pliki do projektu importujac lub zapisujac je.
- Edytor VisualDSP++ pozwala stworzyc nowe pliki lub edytowac istniejace pliki tekstowe

Nowy Plik

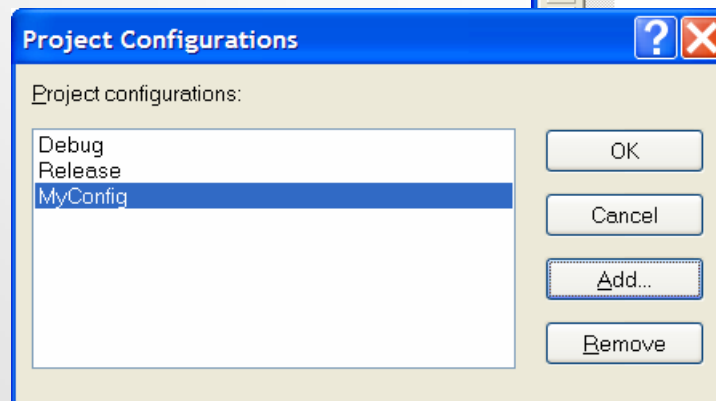
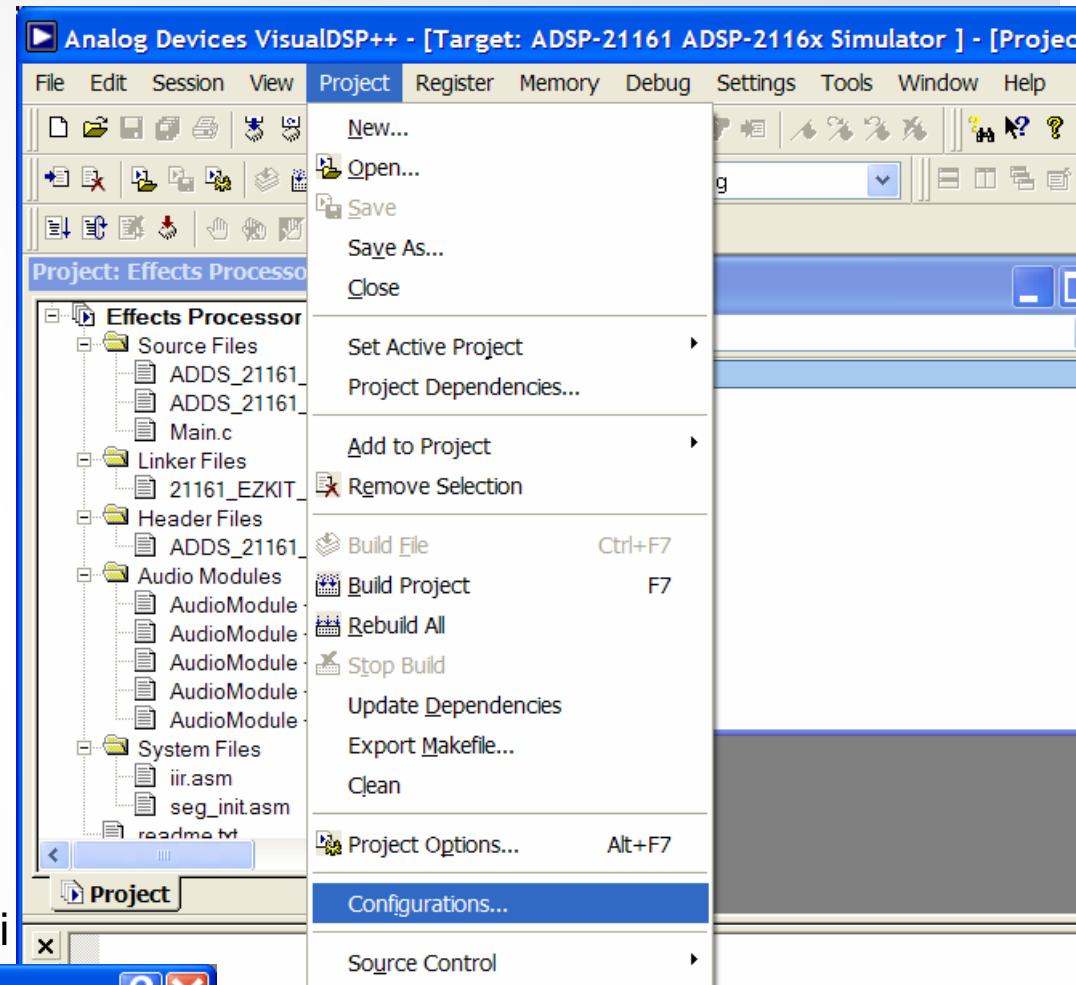
Dodawanie plikow

takze dodawanie



Kroki Rozbudowy Projektu

- Definiowanie opcji budowy projektu
 - Ustawienia konfiguracji projektu kontroluje przebudowę. Domyslnie wybierane sa Debug lub Release.
 - Debug'owanie
 - Przewaznie ma wiecej opcji debugowania ustawianych narzedziami.
 - Kompilator generuje informacje debugowania uwzgledniajac zrodlowy poziom debug'owania.
 - Release (wyzwalanie)
 - Przewaznie ma mniej (lub wcale) opcji debugowania ustawianych narzedziami
 - Przebudowy sa zwykle optymalizowane dla zwiekszenia wydajnosci



Console Build
r selects project configuration

Menu VisualDSP++

The screenshot shows the VisualDSP++ IDE interface. The title bar reads "Analog Devices VisualDSP++ - [Target: ADSP-21161 ADSP-2116x Simulator] - [Project: Effects Processor]". The menu bar includes File, Edit, Session, View, Project, Register, Memory, Debug, Settings, Tools, Window, and Help. The toolbar contains various icons for file operations and development. The main workspace is divided into several panes: a Project Explorer on the left showing a tree view of the project files (Source Files, Linker Files, Header Files, Audio Modules, System Files); a Disassembly window in the center showing assembly code; and a source code editor at the bottom showing the contents of "ADDS_21161_EzKit.c". The source code includes preprocessor directives and comments. At the bottom, there is an Output window and a Console window.

Przeladowanie

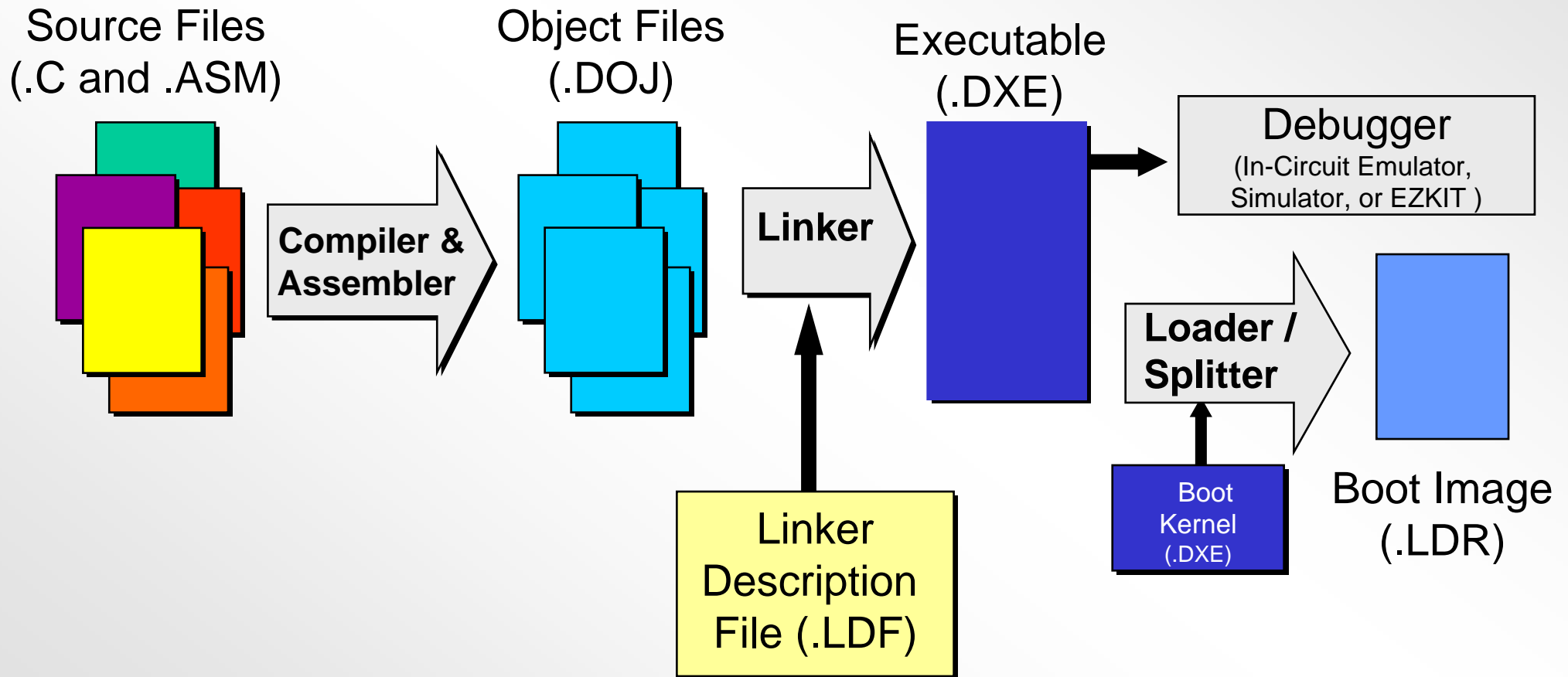
Budowa projektu

dodawanie plikow naglowkowych, zrodlowych, i .ldf do projektu.

**opcje specyficzne pliku
wybor pliku - prawy przycisk myszy, wybor: File Options**

Programowa Budowa Potoku

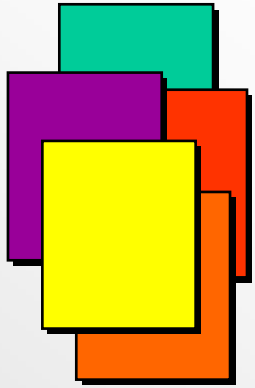
Jakie pliki sa uzywane?



Programowa Budowa Potoku

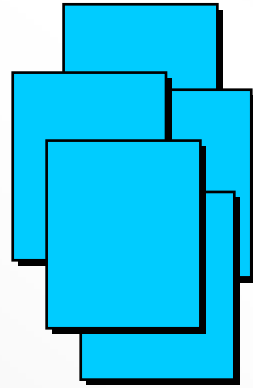
Jakie pliki sa uzywane?

Source Files
(.C and .ASM)



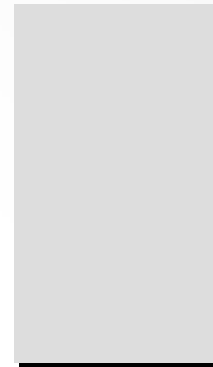
Compiler &
Assembler

Object Files
(.DOJ)



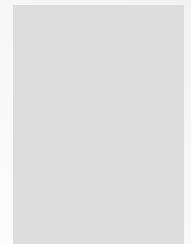
Linker

Executable
(.DXE)



Debugger
(In-Circuit Emulator,
Simulator, or EZKIT)

Loader /
Splitter



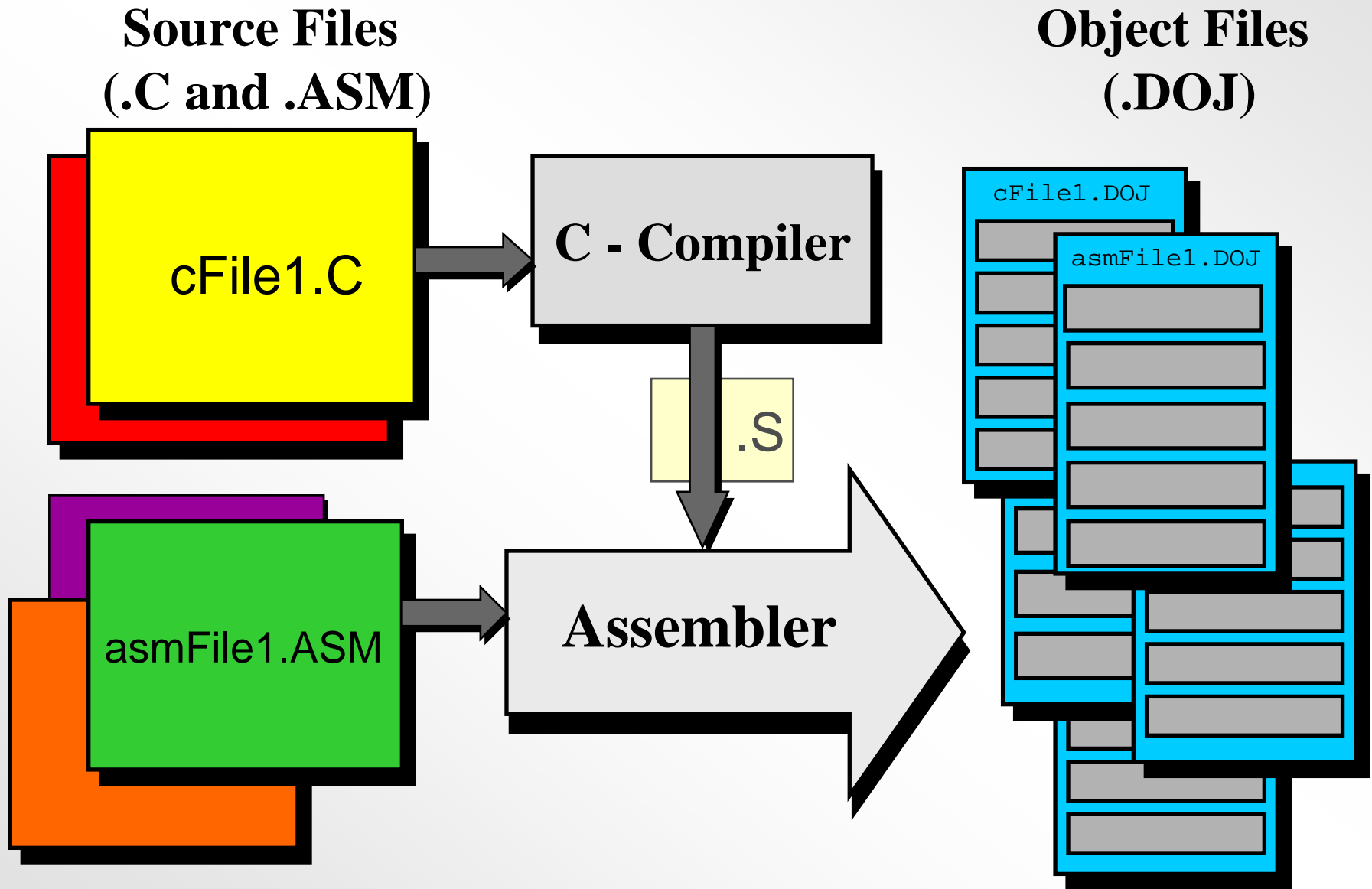
Boot Image
(.LDR)

Linker
Description
File (.LDF)

Boot Kernel
(.DXE)

Programowy Proces Budowy

Krok 1 - Kompilacja i Asemblacja



Programowy Proces Budowy

Krok 1 - Przykład: Asemblacja Zrodla

asmFile1.ASM

```
.section /dm data1;
    .var array[10]

.section /pm code1;
start:r0 = 0x1234;
    r1 = 0x5678;
    r2 = r1 + r2;
    jump start;
```

Assembler

asmFile1.DO

Object Section = data1

```
array[0]
array[1]
...
...
array[9]
```

Object Section = code1

```
start:
r0 = 0x1234;
r1 = 0x5678;
r2 = r1 + r2;
jump start;
```

Programowy Proces Budowy

Krok 1 - Przykład: plik C

cFile1.C

```
main()
{
    int j = 12;
    int k = 0;
    k += j * 2;
    func1();
}

void func1(void)
{
    int var1;
    foo = 1;
    foo ++;
}
```

C-Compiler

.S

Assembler

cFile1.DOJ

Object Section = seg_pmco

_main:

...

r2 = r3 * r4;

r0 = r0 + r2;

dm(_k) = r0;

ccall _func1;

_func1:

r1 = dm(m3, i6)

r1 = r1 + 1;

...

Object Section = seg_stak

_j : 12

_k : 0

_var1: 1

Programowy Proces Budowy

Krok 1 Przykład: plik C z alternatywnymi sekcjami

foo.C

```
section ("extern") int array[256];

section ("foo") void bar(void)
{
    int foovar;
    foovar = 1;
    foovar ++;
}
```

C-Compiler

Assembler

foo.DOJ

```
Object Section = extern
Type           = DM
Width          = 32
.....
```

```
_array [00]
_array [01]
...
_array [255]
```

```
Object Section = foo
Type           = PM
Width          = 48
.....
```

```
_bar :
r0 = dm(_foovar);
r0 = r0 + 1;
```

```
Object Section = seg_stak
Type           = PM
Width          = 32
.....
```

```
_foovar: 1
```

Dyrektywy Asemblera

.SECTION	Oznacza początek sekcji sąsiednich pamięci
.PREVIOUS	Odwoluje do poprzednio zaznaczonej .SECTION
* .SEGMENT	Komenda wymienna z .section
* .ENDSEG	
.ALIGN	Określa ustawienie danych
.VAR	Definicja i inicjalizacja buforów danych i zmiennych
.GLOBAL	Umożliwia dostęp z innych plików źródłowych
.EXTERN	Odwolanie do odnośnika w innych plikach źródłowych
.PRECISION 32	Ustawienie precyzji dla inicjalizowanych stałych i zmiennych
.PRECISION 40	(tylko zmiennoprzecinkowe)
.ROUND_NEAREST	Ustawienie trybu round dla inicjalizowanych stałych i zmiennych
.ROUND_MINUS	gdy wartość nie pasuje do docelowemu formatowi.
.ROUND_PLUS	
.ROUND_ZERO	

Operatory Asemblera

- Akceptowane wszystkie wartosci numeryczne wykazane w skladni instrukcji asemblerowych.

Operator	Opis
(wyrazenie)	Wyrazenie w nawiasie wykonane wczesniej
LENGTH(bufor)	Dlugosc bufora w slowach
@bufor	Zapis operatora. Dlugosc bufora w slowach
~	Ones complement
-	Unary minus
*	Mnozenie
/	Dzielenie
%	Modulo
+	Suma
-	Roznica
<<	Przesuniecie w lewo (logiczne)
>>	Przesuniecie w prawo (logiczne)

Operatory Asemblera (c.d.)

Operator	Opis
<	Mniejsze
<=	Mniejsze rowne
>	Większe
>=	Większe rowne
==	Rowne
!=	Rozne
&	Bitowe: AND
^	Bitowe: exclusive OR
	Bitowe: inclusive OR
&&	Logiczne: AND
	Logiczne: OR

Przyklad: `L0=length(data_buffer_1); /* ustawia dlugosc w slowach */`
`/* data_buffer_1 into register L0 */`
`L0=@data_buffer_1; /* dawniej uzywano */`

Asemblerowe Dyrektywy Preprocesora

Komenda	Opis
#define	Definiuje makro
#if	Poczatek pary #if ...#endif
#else	Identyfikuje instrukcje alternatywne wewnarz #if ...#endif
#elif	Dzieli pare #if ... #endif
#endif	Zakonczenie pary #if ...#endif
#error	Generuje wiadomosc o bledzie
#ifdef	Rozpoczecie pary #ifdef ... #endif i sprawdzanie def makra
#ifndef	Rozpoczecie pary #ifndef ... #endif i sprawdzanie def makra
#include	Zalaczenie zawartosci pliku
#line	Numer linii wyjsciowej
#undef	Usuwa definicje macra
#warning	Generuje wiadomosc ostrzezenia
#	Zamienia argument makra na stala typu string
##	Laczy dwa string'i

Przykład Użycia Dyrektyw Asemblera

```
Comments      /* komentarz */
              // komentarz

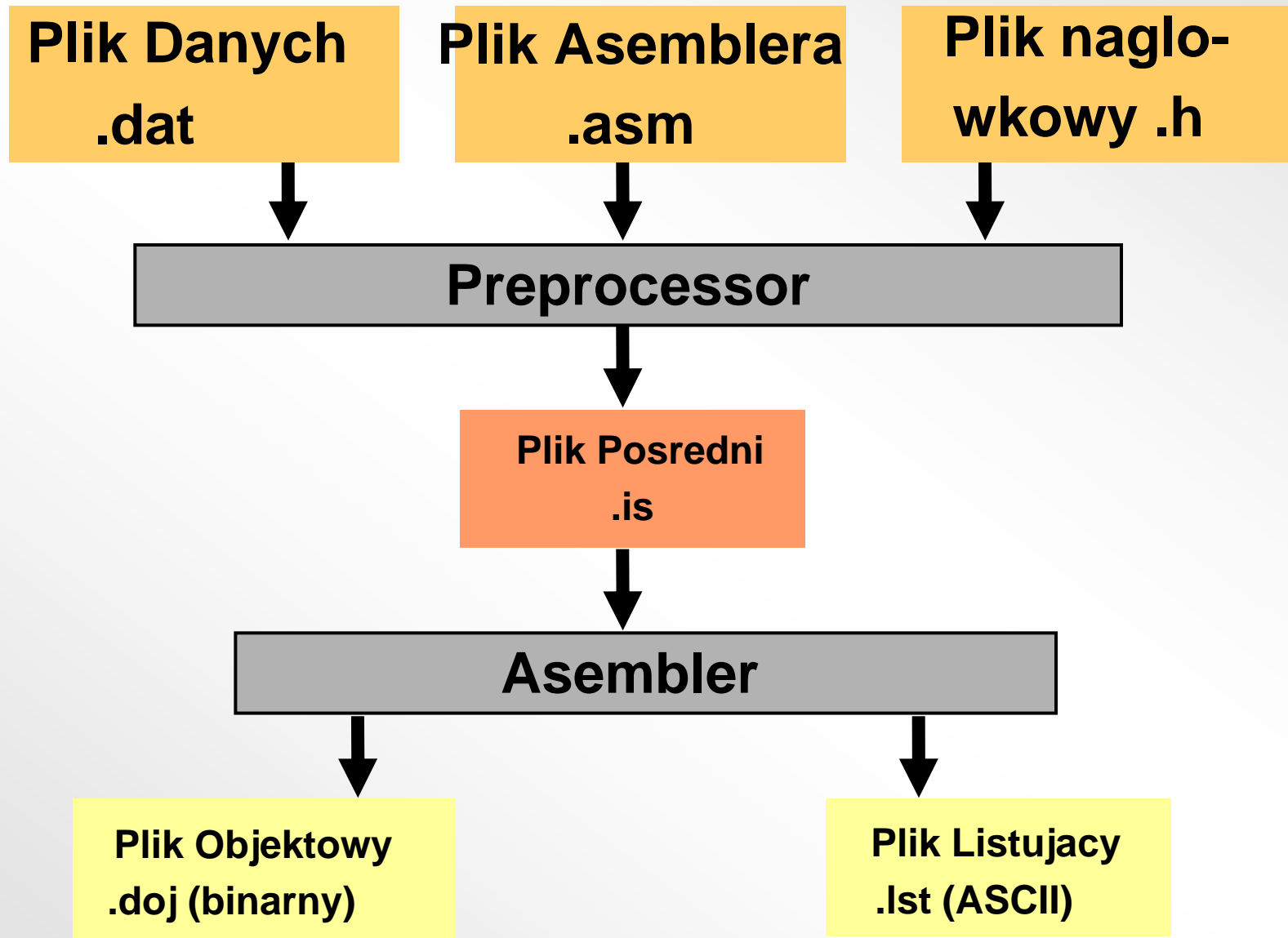
Preprocessor   #define N 100
              #if N >= 50
                #define NLARGE
              #endif

Data Storage   .Section/dm      DM_DATA;
              .VAR      DM_BUF_1[N];
              .VAR      DM_BUF_2[200];

              .Section/pm      PM_DATA;
              .VAR      PM_BUF_1[N];
              .VAR      PM_BUF_2[200];

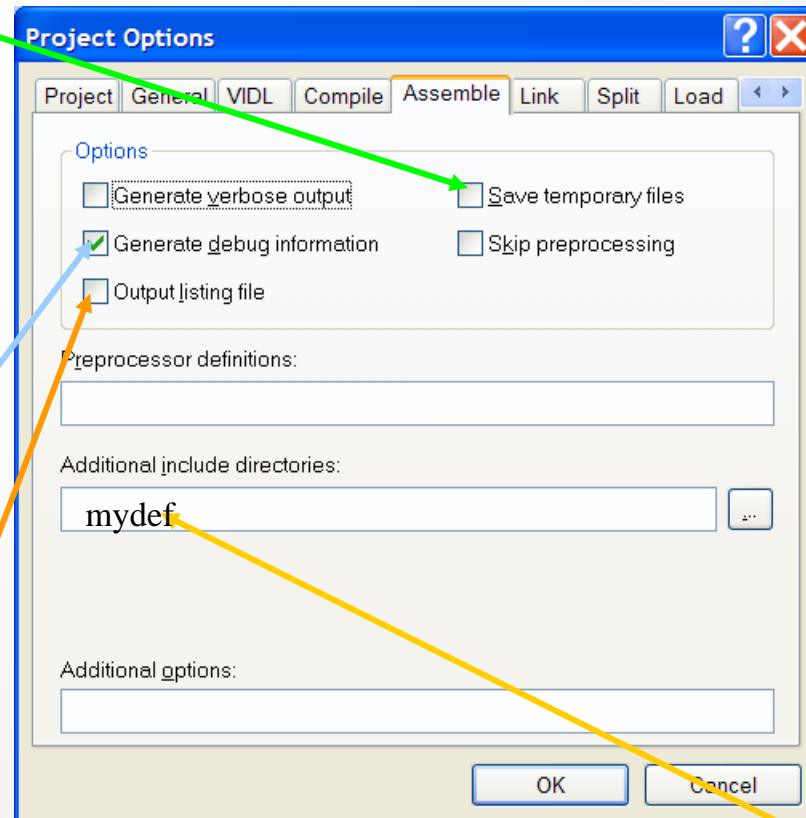
Code Storage   .Section/pm      PM_CODE;
              B0=DM_BUF_1;
              L0=LENGTH(DM_BUF_1);
              M0=1;
```

Asembler



Zakładka 'Assemble'

Zaznaczenie stworzy plik tymczasowy .is



Zaznaczone, można debug'ować kod źródłowy

Zaznaczone, stworzy plik z listingiem

```
#include <def21161.h>  
#include "myheader.h"
```

```
#ifdef mydef
```

```
R0 = R0 + 1;
```

```
#else
```

```
R0 = R0 - 1;
```

```
#endif
```

Zależnie od definicji, można wybrać różne kody

Uzycie def21161.h definicji Bitowego Rejestru Systemu

- Nazwy symboliczne dla programisty
 - rejestry IOP
 - Bity rejestru i pola bitowe

Przykłady:

```
#define SYSCON 0x0
#define HMSWF 0x00000080
```

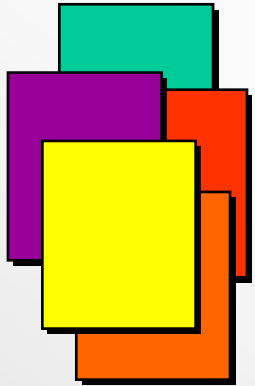
- Indywidualne definicje bitow moga byc **OR**'owane operatorem “|”
- Przykłady:

```
it set MODE1 BR0 | IRPTEN | RND32;
USTAT1 = BSO | HBW8 | HMSWF;
dm(SYSCON) = USTAT1;
```
- ```
#include <def21161.h>
```

# Programowa Budowa Potoku

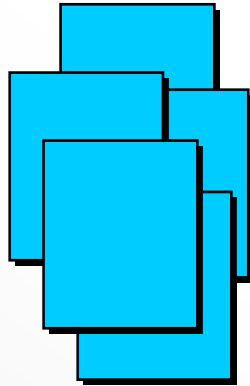
## Krok 1 - Kompilacja & Asemblacja

Source Files  
(.C and .ASM)



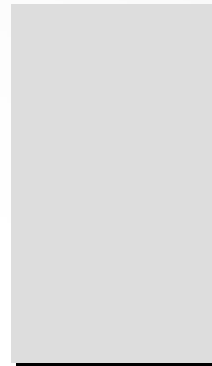
Compiler &  
Assembler

Object Files  
(.DOJ)



Linker

Executable  
(.DXE)



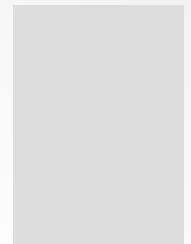
Debugger  
(In-Circuit Emulator,  
Simulator, or EZKIT)

Loader /  
Splitter

Boot Kernel  
(.DXE)

Boot Image  
(.LDR)

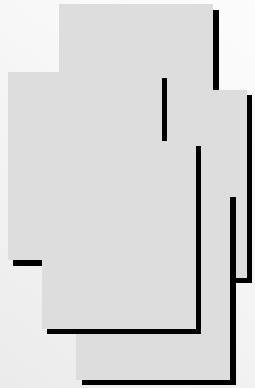
Linker  
Description  
File (.LDF)



# Programowa Budowa Potoku

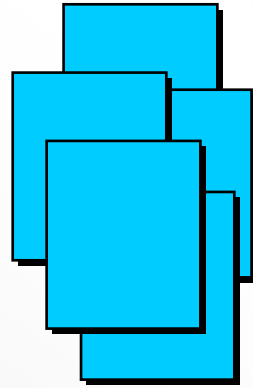
## Krok 2 - Linking

Source Files  
(.C and .ASM)



Compiler &  
Assembler

Object Files  
(.DOJ)



Linker

Executable  
(.DXE)



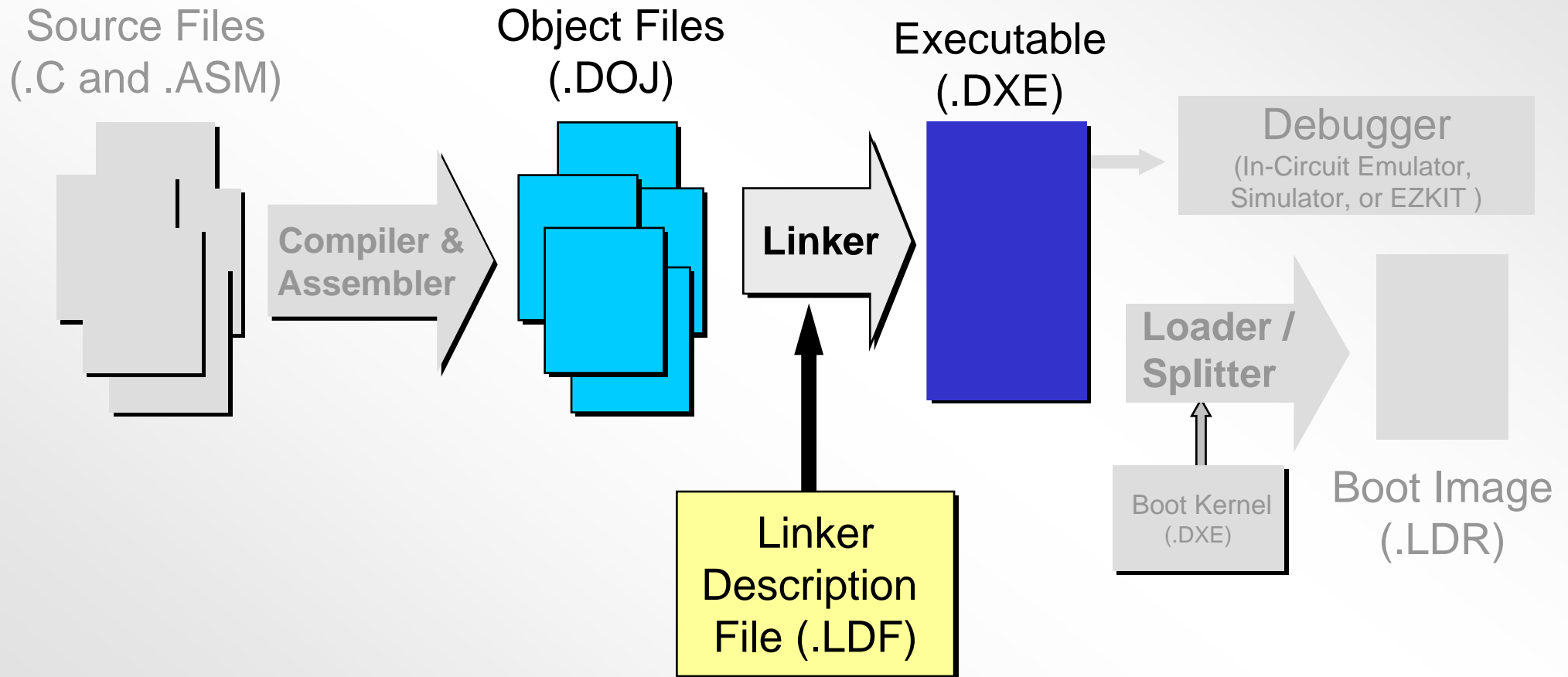
Debugger  
(In-Circuit Emulator,  
Simulator, or EZKIT )

Loader /  
Splitter

Boot Kernel  
(.DXE)

Boot Image  
(.LDR)

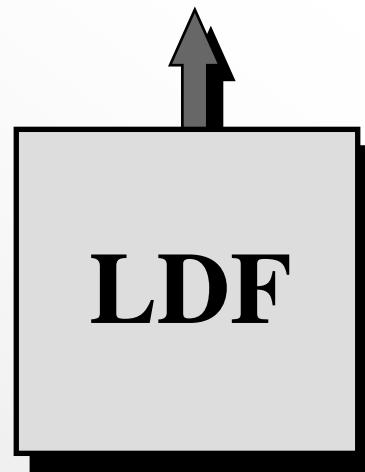
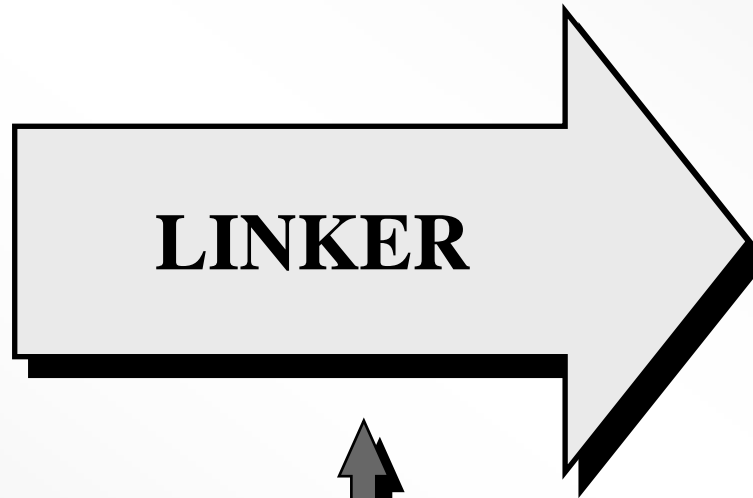
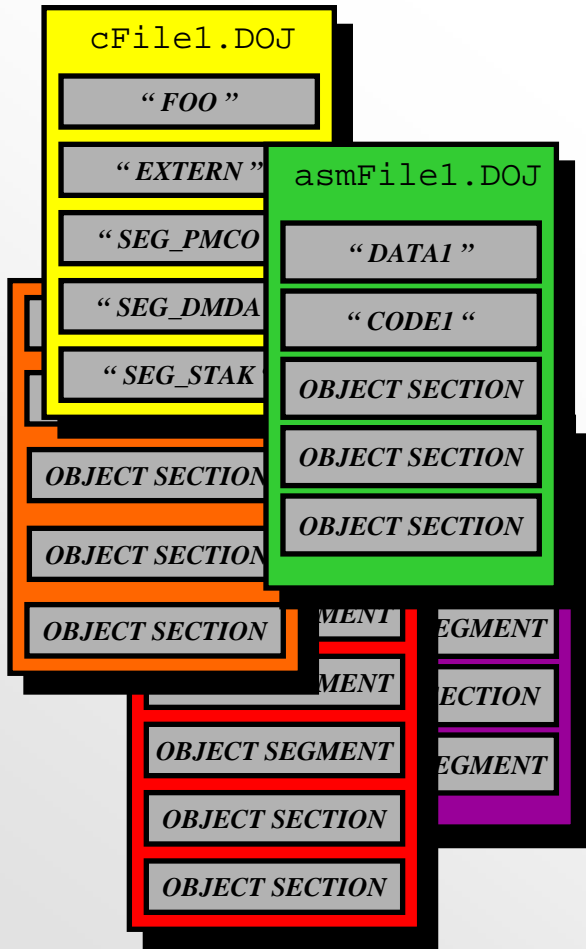
Linker  
Description  
File (.LDF)



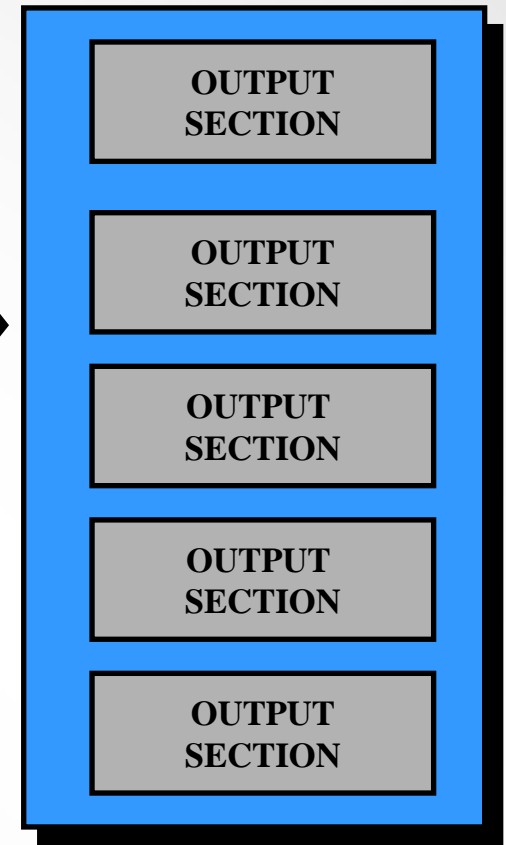
# Opis Pliku Linkowania

## Krok 2 - Linkowanie

### Object Files (.DOJ)



### Executable (.DXE)





# Opis Pliku Linkowania (LDF)

- **Jezyk komend linkowania kontroluje calkowity proces**
  - **Konfiguracja pamieci**
    - **Pamiec wewnetrzna i zewnetrzna**
    - **Pamiec dzielona**
    - **Overlays**
  - **Wejscowe pliki linkera**
  - **Wyjscowe pliki linkera**
  - **Linkowanie multiprocessora**

# Linker

**Plik Objektowy  
.DOJ**

**Pliki Biblioteczne  
.DLB**

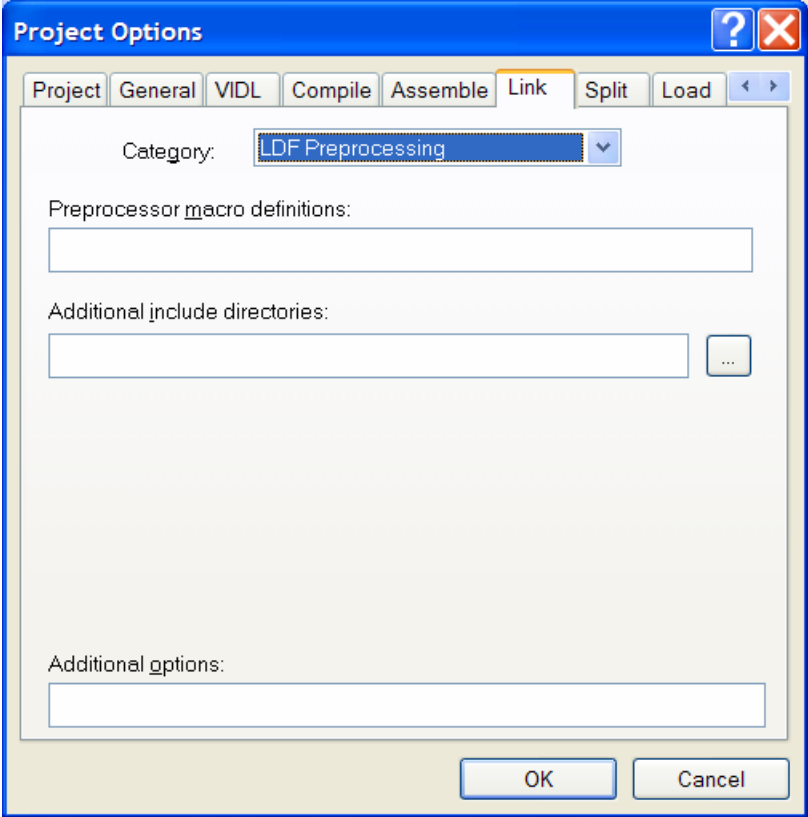
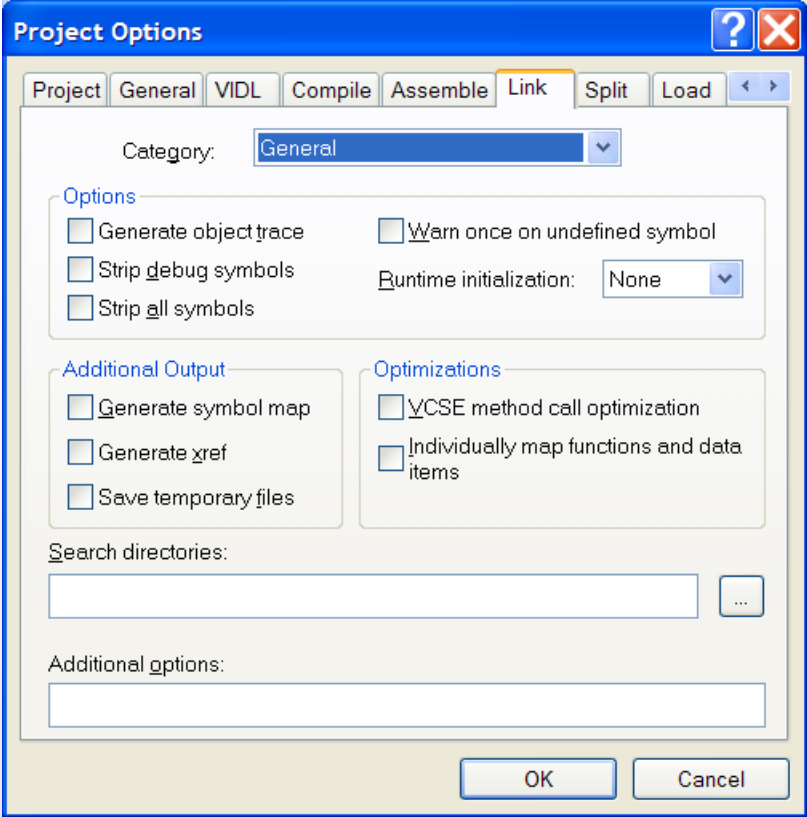
**Opis Linkera  
Files .LDF**

**Linker**

**Obraz Pamięci  
.DXE (binarny)**

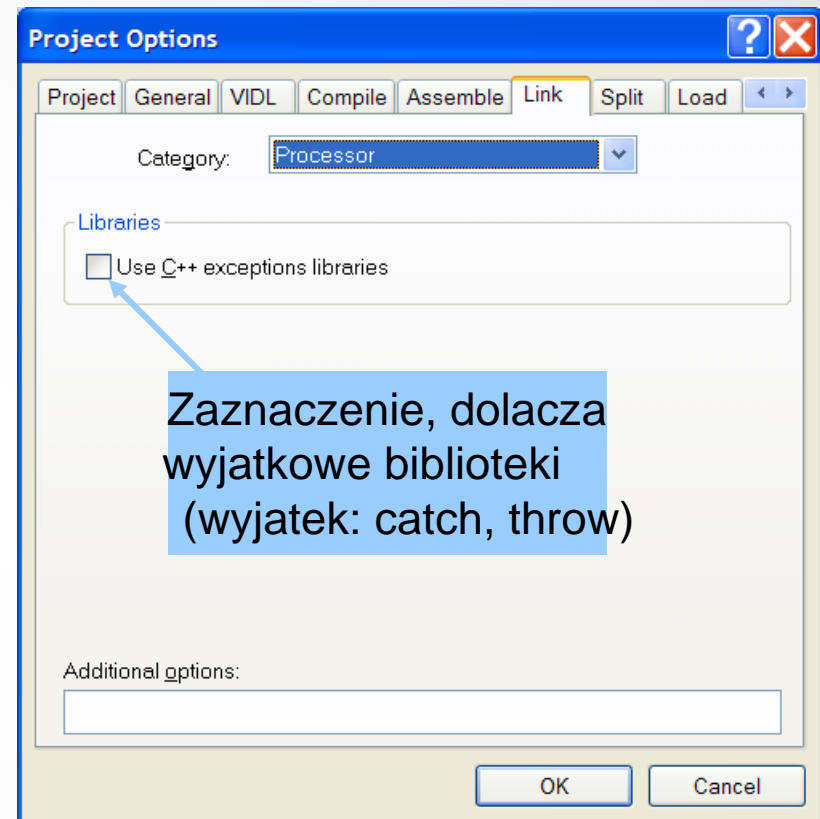
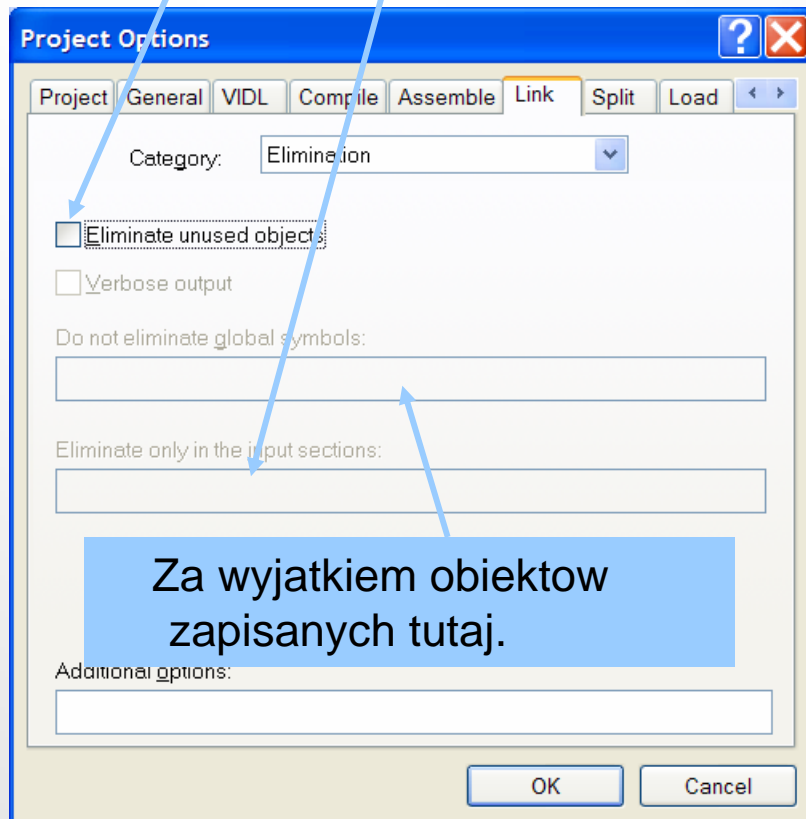
**Mapa pamięci  
.MAP (ASCII)**

# Zakladka 'Link'



# Zakładka 'Link'

Zaznaczenie - globalnie lub selektywnie eliminuje nieużywane obiekty...



# Plik Opisujący Linkowanie (LDF)

- **Komendy globalne**
  - Definiują architekturę lub procesor
  - Ścieżki przeszukiwania katalogu
  - Załączane pliki biblioteczne i obiektowe
- **Opis pamięci**
  - Definiuje segmenty pamięci
  - Definiuje miejsce pamięci multiprocesora
- **Komendy linkujące projekt**
  - Umożliwia linkowanie listy plików obiektu
  - Nazwa pliku wyjściowego
  - Mapowanie sekcji wejściowych do segmentów pamięci

# Przykład LDF

## Komendy globalne i Opis pamięci

```
ARCHITECTURE(ADSP-21161)
SEARCH_DIR($ADI_DSP\211xx\lib)

$LIBRARIES = lib161.dlb, libc161.dlb;
$OBJECTS = $COMMAND_LINE_OBJECTS, 161_hdr.doj;
/* asmFile1.doj, cFile1.doj */
```

### MEMORY

```
{
 mem_pmco { TYPE(PM RAM) START(0x00040000) END(0x000419ff) WIDTH(48) }
 mem_pmda { TYPE(PM RAM) START(0x00042a00) END(0x00043fff) WIDTH(32) }
 mem_dmda { TYPE(DM RAM) START(0x00050200) END(0x00051fff) WIDTH(32) }
 mem_ext1 { TYPE(DM RAM) START(0x00200000) LENGTH(0x100) WIDTH(32) }
 mem_ext2 { TYPE(DM RAM) START(0x00200100) LENGTH(0xFFFFF) WIDTH(32) }
}
```

Memory  
Segment  
Name

Memory  
Type

Starting  
Address

Ending  
Address  
or Length

Data  
Word  
Width

# Przykład LDF (c.d.)

## Komendy linkowania

```
PROCESSOR p0
{

 OUTPUT($COMMAND_LINE_OUTPUT_FILE)
 // MyProject.dxe

 SECTIONS
 {

dog
 {INPUT_SECTIONS($OBJECTS(seg_pmco)
 $LIBRARIES(seg_pmco)) } > mem_pmco

cat
 {INPUT_SECTIONS(asmFile1.doj(data1)
 cFile1.doj(seg_dmda)
 161_hdr.doj(seg_dmda)
 $LIBRARIES(seg_dmda)) } > mem_dmda

bird
 {INPUT_SECTIONS(asmFile1.doj(extern)) } > mem_ext1
 ...
 }/*end sections*/
}/*end p0*/
```

***DXE SECTION NAMES***  
Only used by down-stream tools  
(debugger, loader, and splitter)

***OBJECT SECTIONS***  
from C and Assembly object files

***MEMORY SEGMENTS***  
declared in the LDF

# Komendy Pliku LDF

- **ARCHITECTURE():** Okresla przeznaczenie procesora w systemie
- **INCLUDE:** Okresla dodatkowy plik LDF ktory bedzie wykonany przed wykonaniem wlasciwego pliku
- **MAP():** Wyjscie pliku 'map' ze specjalna nazwa
- **MEMORY{}**: Definiuje pamiec fizyczna systemu
- **PLIT{}**: Dodaje procedure komend 'linkage table' do programu (Overlay support)
- **PROCESSOR{}**: Deklaruje procesor i informacje o jego linkowaniu
- **SEARCH\_DIR():** Okresla jeden lub wiecej katalogow do szukania
- **SECTIONS{}**: Okresla polozenie sekcji programu .SECTION's w pamieci
- **SHARED\_MEMORY{}**: Wyjscia wykonawczej dzielonej pamieci



# Macra Pliku LDF

- **\$COMMAND\_LINE\_OBJECTS**
  - Lista obiektów (.DOJ) i bibliotek (.DLB) podanych w linii komend.
- **\$COMMAND\_LINE\_LINK\_AGAINST**
  - Lista plików wykonawczych (.DXE, .SM) podanych w linii komend.
- **\$COMMAND\_LINE\_OUTPUT\_FILE**
  - Wyjściowe pliki wykonawcze specjalnie nazwanych w linii komend znacznik -o.
- **\$ADI\_DSP**
  - Ścieżka do katalogu VisualDSP.
- **\$macro**
  - Zdefiniowane macro użytkownika (typowo dla listy plików.)
  - przykład `$OBJECTS = crt.obj, $COMMAND_LINE_OBJECTS;`

# Expert Linker

**Zastosowanie 'LDF Wizard'**

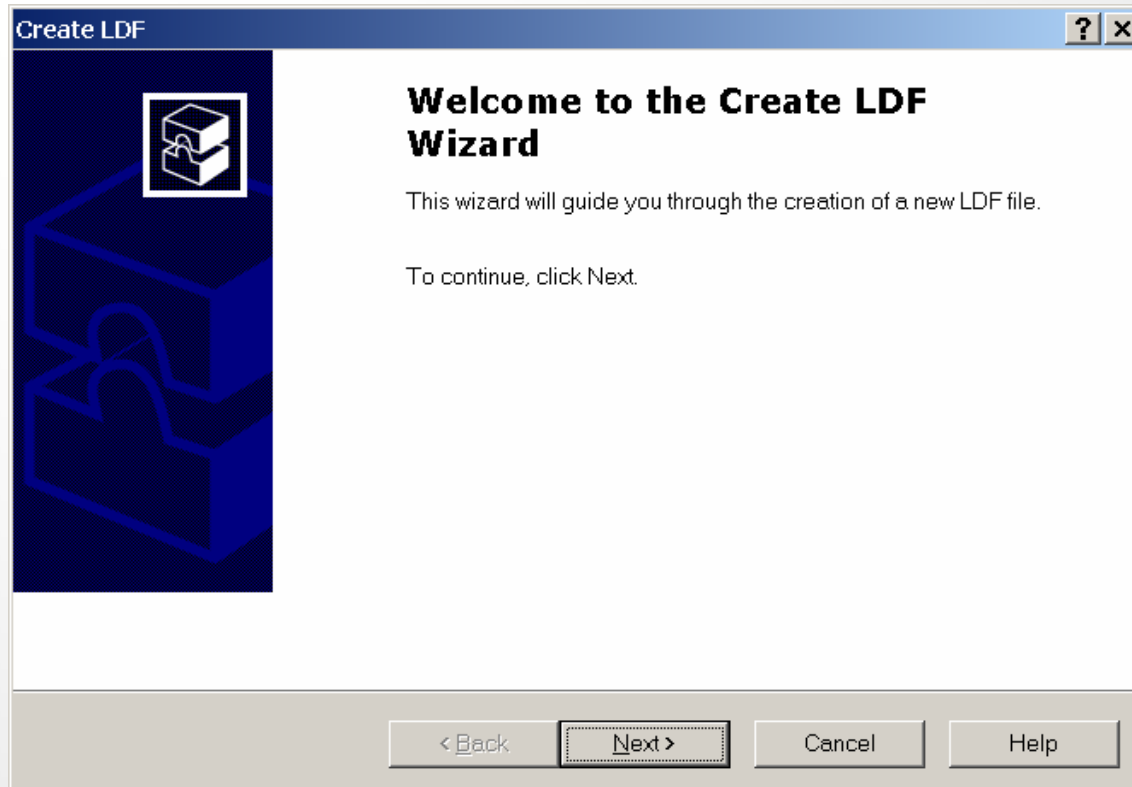
# Cechy 'Expert Linker'a'

- 'Expert Linker' to graficzne narzędzia które pozwalają na:
  - Użycie wizard'a do stworzenia plików LDF
  - Definiowanie mapy pamięci DSP's
  - Przenoszenie i umieszczanie sekcji obiektów w mapie pamięci
  - Tworzenie nakładek graficznych
  - Import plików LDF
  - Eliminacje graficznego wydawnionego kodu nieużywanych obiektów
  - Przekroj sekcji obiektów w pamięci

# Uruchomienie LDF Wizard

- **Uzycie Expert Linker'a do stworzenia pliku LDF**
- **Uruchomienie kreatora przez wybor:**  
**Tools -> Expert Linker -> Create LDF**
- **Kreator przeprowadzi 3-krokowy proces do generacji pliku LDF:**
  - **Krok 1: Nazwa pliku LDF i wybor jezyka programowania (C, C++, Assembly)**
  - **Krok 2: Wybor typu procesora i innych wlasciwosci.**
  - **Krok 3: Pokazane jest potwierdzenie zbiorcze wyborow. Zakonczenie przyciskiem 'Finish' generuje plik LDF.**

# Create LDF Wizard



# LDF Wizard: Krok 1

**Create LDF - Step 1 of 3**

**Project Information**  
Choose the LDF file name and the project type.

LDF filename:  
C:\Program Files\Analog Devices\VisualDSP\211xx\Examples\ASM\_Examples\DFT\Dft.ldf

Project type

- C
- C++
- Assembly
- VisualDSP++ kernel (VDK)

< Back   Next >   Cancel   Help

# LDF Wizard: Krok 2

**Create LDF - Step 2 of 3** [?] [X]

**System Information**  
Configure the DSP system by choosing the processors in your system and the processor type.

System type:

Single processor  
 Multiprocessor

Processor type:  
ADSP-21161

Set up system from debug session settings

Processor properties:

Processors:

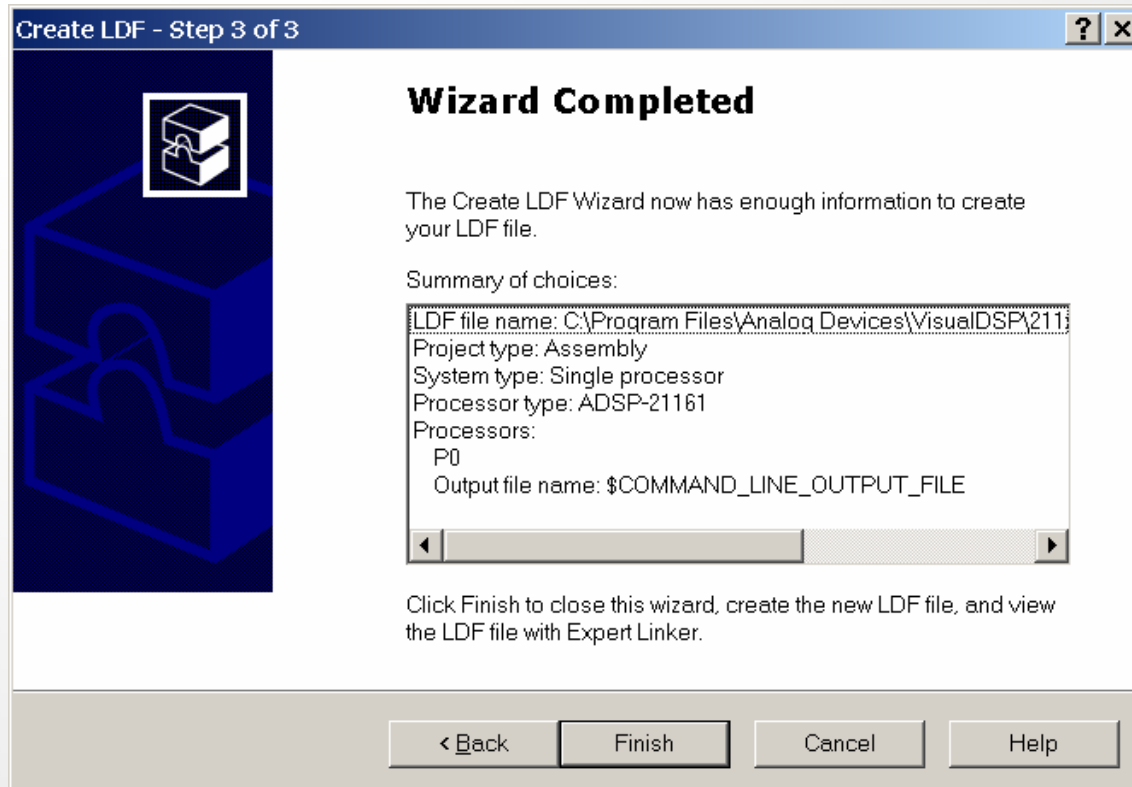
| Processor                              |
|----------------------------------------|
| <input checked="" type="checkbox"/> P0 |

Output file name:  
\$COMMAND\_LINE\_OUTPUT\_FILE

Executables to link against:  
[ ]

< Back    Next >    Cancel    Help

# LDF Wizard: Krok 3





# Resultaty Expert Linker'a

**Sekcje wejsciowe  
ktore nie moga byc  
mapowane pojawia  
sie przekreslone**

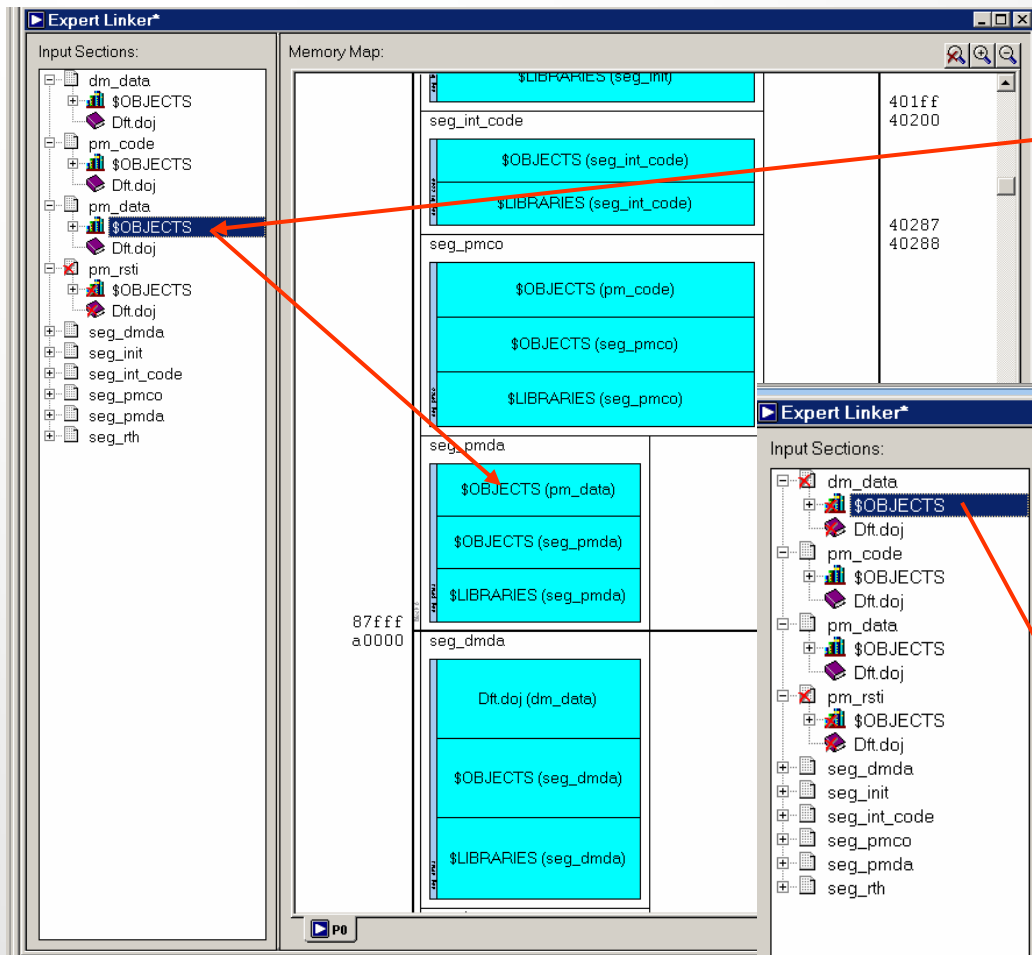
**Legend**

- Icons
- Colors
- LDF Macro
- Unmapped LDF Macro
- Library File
- Unmapped Library File
- Object File
- Unmapped Object File
- Object Section
- Unmapped Object Section
- Pinned Object Section
- Processor
- Memory Segment
- Invalid Memory Segment
- Shared Memory
- Output Section
- Output Section that Overflows
- Overlay (Live Space)
- Overlay (Run Space)

| Address | Segment Name     | End Address |
|---------|------------------|-------------|
| 80000   | seg_rth          | 40000       |
|         | seg_init         | 40100       |
|         | seg_int_code     | 40200       |
|         | seg_pmco         | 40288       |
|         | seg_pmda         | 42700       |
| 87fff   |                  |             |
| a0000   | seg_dmda         | 50000       |
|         | seg_heap         | 51fff       |
|         | seg_stak         | 52fff       |
| a7fff   |                  |             |
| 100000  | Multiproc memory |             |
| 1fffff  |                  |             |
| 200000  | External memory  |             |
| ffffff  |                  |             |

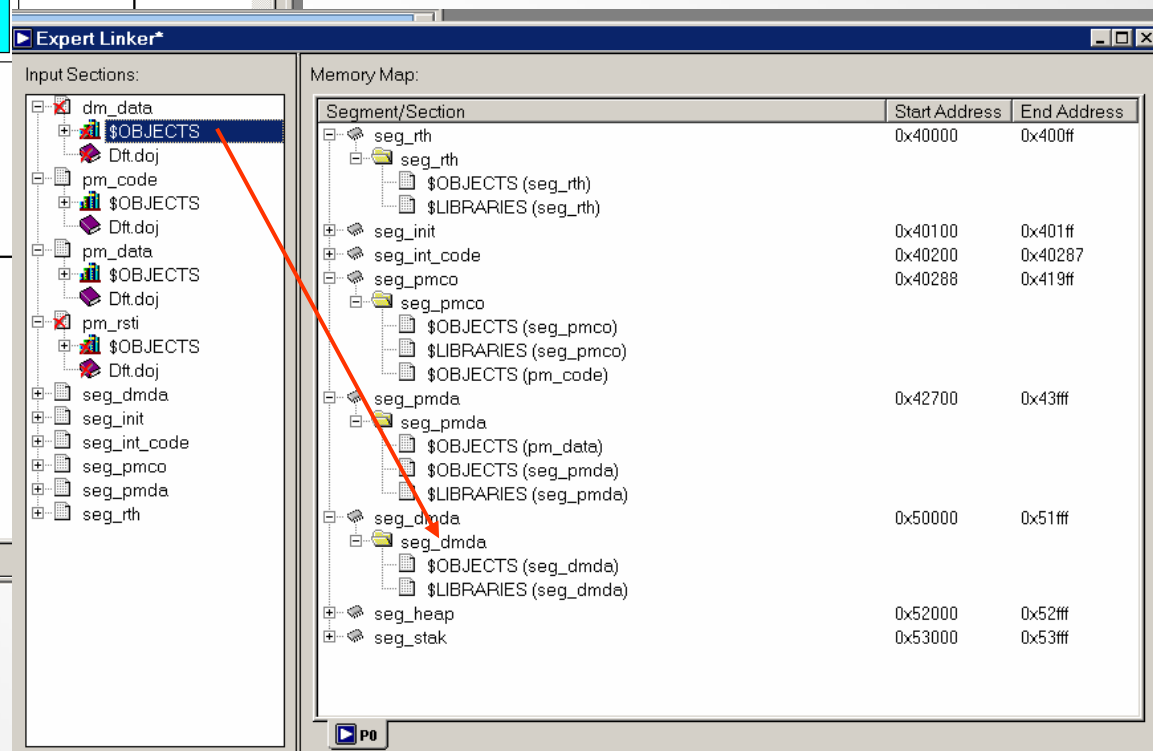
**Widok graficznego obrazu pamieci generowanego pliku .ldf.  
Podwojne klikniecie na segmencie powieksza go.**

# Kontrola Mapowania Sekcji



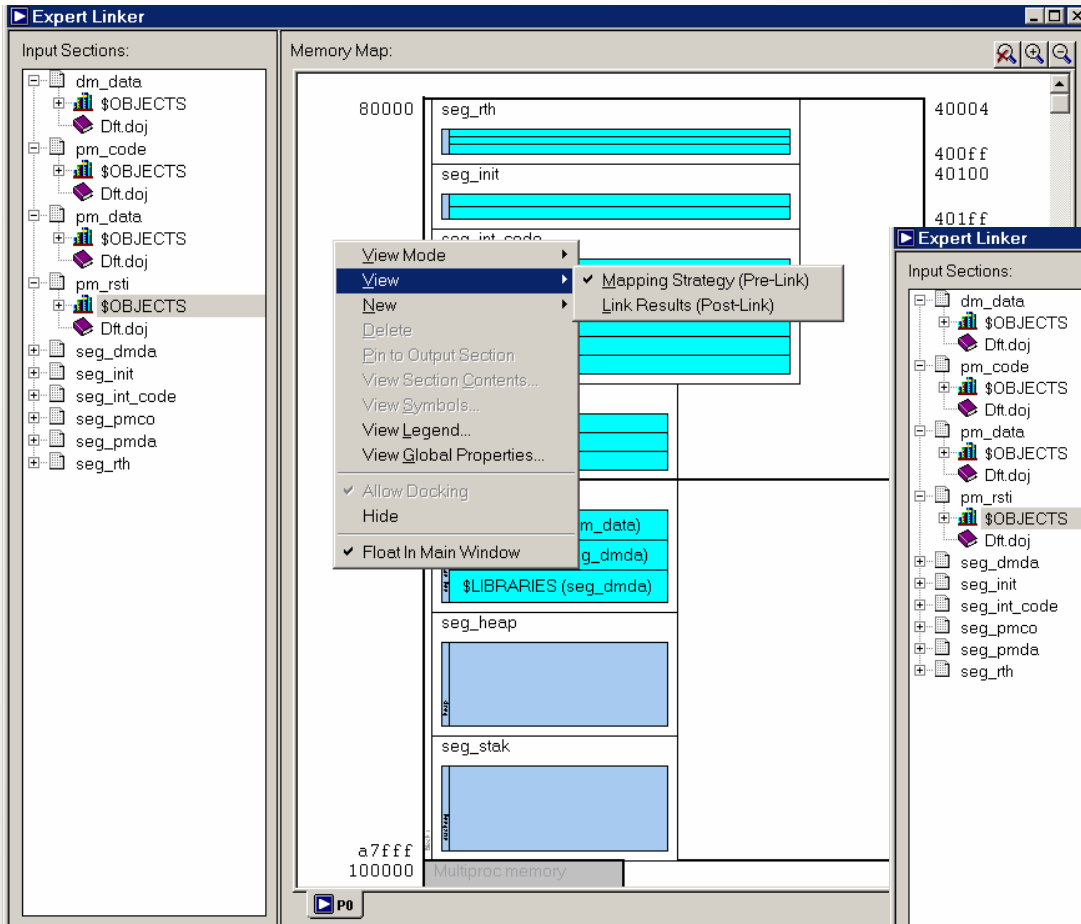
Niemapowane sekcje mogą być 'mapowane' poprzez przeciągnięcie do odpowiednich segmentów pamięci.

Widok graficzny mapy pamięci

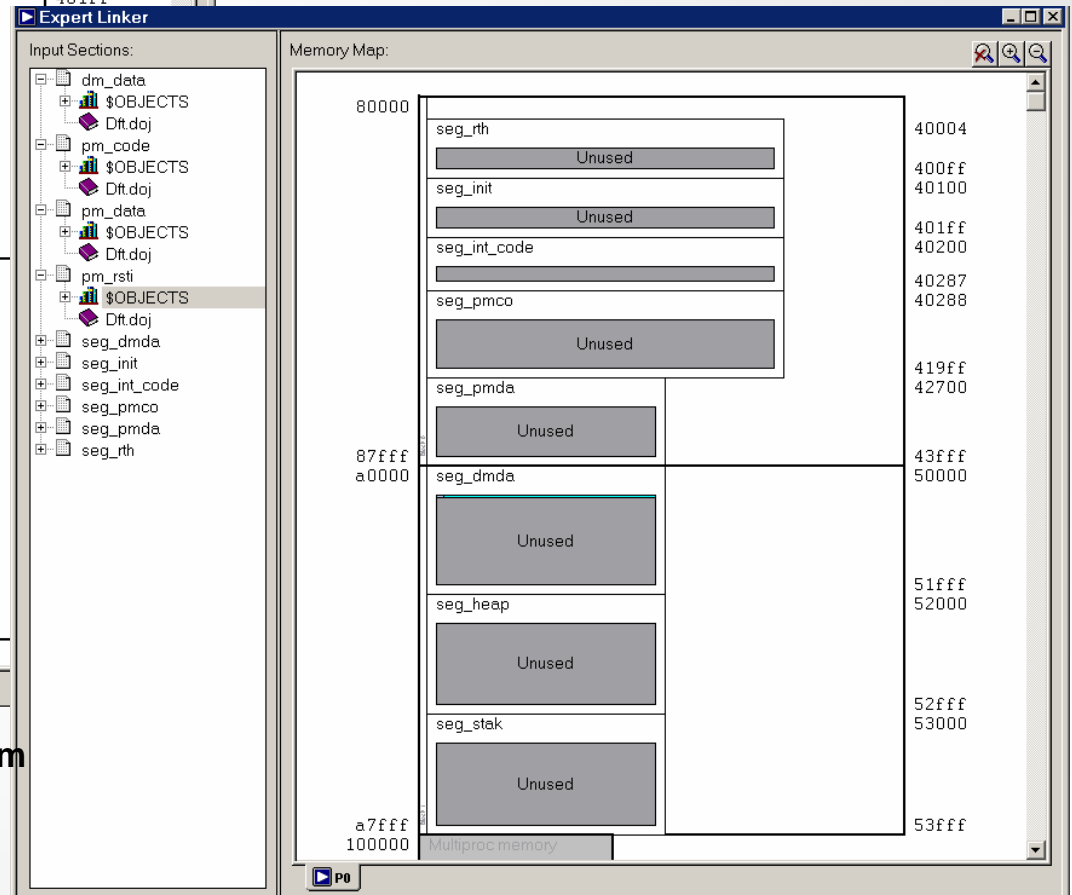


Widok drzewa mapy pamięci

# Rezultaty przed i po linkowaniu



Strategia mapowania przed linkowaniem



Rezultaty po linkowaniu

# Wlasciwosci Segmentu LDF

Input Sections:

- dm\_data
  - \$OBJECTS
  - Dft.doj
- pm\_code
  - \$OBJECTS
  - Dft.doj
- pm\_data
  - \$OBJECTS
  - Dft.doj
- pm\_rsti
  - \$OBJECTS
  - Dft.doj
- seg\_dmda
- seg\_init
- seg\_int\_code
- seg\_pmco
- seg\_pmda
- seg\_rth

Memory Map:

| Segment/Section   | Start Address | End Address |
|-------------------|---------------|-------------|
| seg_rth           | 0x40004       | 0x400ff     |
| seg_rth           | 0x40004       | 0x40007     |
| Dft.doj (pm_rsti) | 0x40004       | 0x40007     |
| seg_init          | 0x40100       | 0x401ff     |
| seg_int_code      | 0x40200       | 0x40287     |
| seg_pmco          | 0x40288       | 0x419ff     |
| seg_pmco          | 0x40288       | 0x402a6     |
| Dft.doj (pm_code) | 0x40288       | 0x402a6     |
| seg_pmda          | 0x42700       | 0x43fff     |
| seg_pmda          | 0x42700       | 0x4273f     |
| Dft.doj (pm_data) | 0x42700       | 0x4273f     |
| seg_dmda          | 0x50000       | 0x51fff     |
| seg_dmda          | 0x50000       | 0x500bf     |
| Dft.doj (dm_data) | 0x50000       | 0x500bf     |
| seg_heap          | 0x52000       | 0x52fff     |
| seg_stak          | 0x53000       | 0x53fff     |

Context Menu:

- View Mode
- View
- New
- Delete
- Pin to Output Section
- Go to
- View Section Contents...
- View Symbols...
- View Legend...
- View Global Properties...
- Allow Docking
- Hide
- Float In Main Window

Widok mapy symboli dla specyficznego segmentu pamieci

View Symbols

| Name    | Address | Size | Binding   | File Name | Section |
|---------|---------|------|-----------|-----------|---------|
| dm_data | 0x50000 | 0x0  | STB_LOCAL | Dft.doj   | dm_data |
| imag    | 0x50080 | 0x40 | STB_LOCAL | Dft.doj   | dm_data |
| input   | 0x50000 | 0x40 | STB_LOCAL | Dft.doj   | dm_data |
| real    | 0x50040 | 0x40 | STB_LOCAL | Dft.doj   | dm_data |

OK

Memory Segment Properties

Memory Segment

Name: seq\_rth

Start Address: 0x40004    End Address: 0x400ff    Size: 0xfc    Width: 48

Memory Space:

- PM
- DM
- BM
- DATA64

ROM/RAM:

- RAM
- ROM
- SBOM

Internal/External:

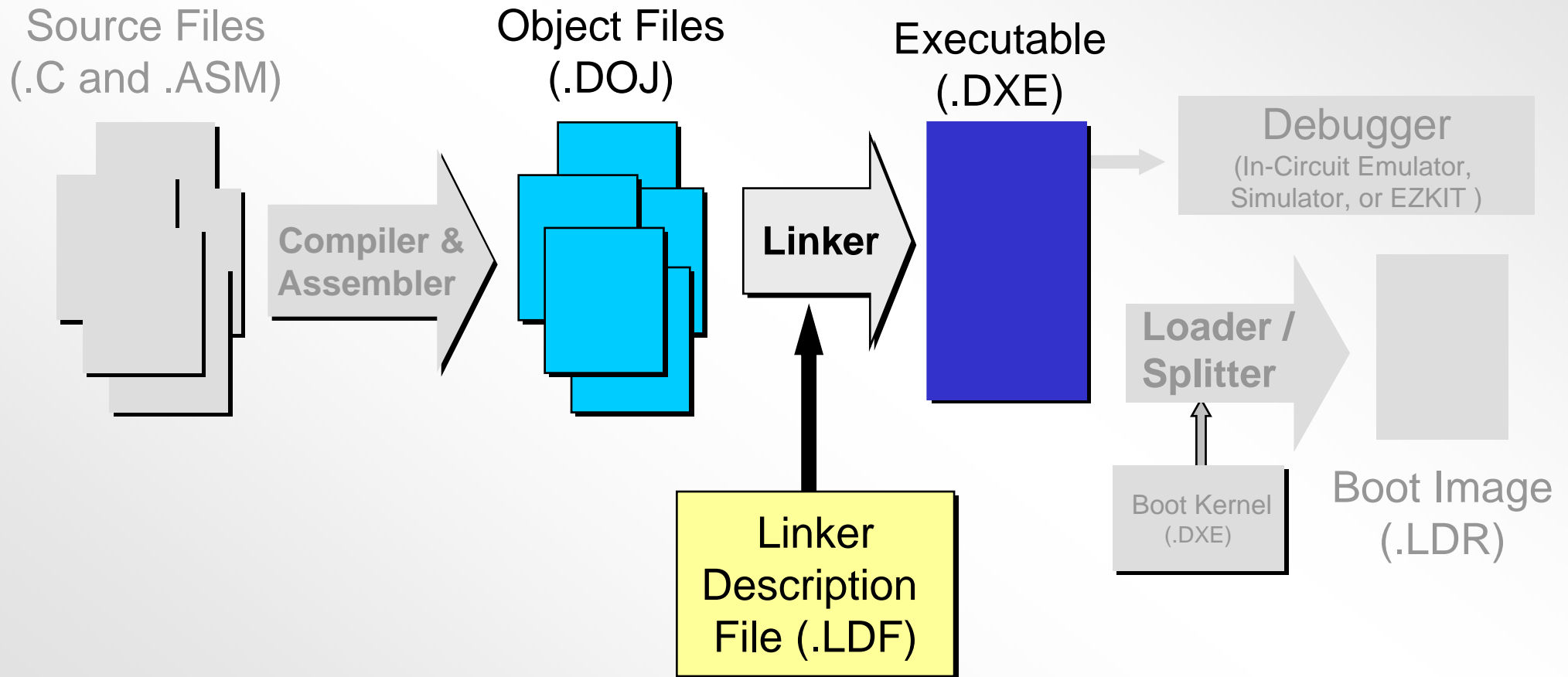
- Internal
- External

OK    Cancel

Definicja wlasciwosci dla specyficznego segmentu pamieci

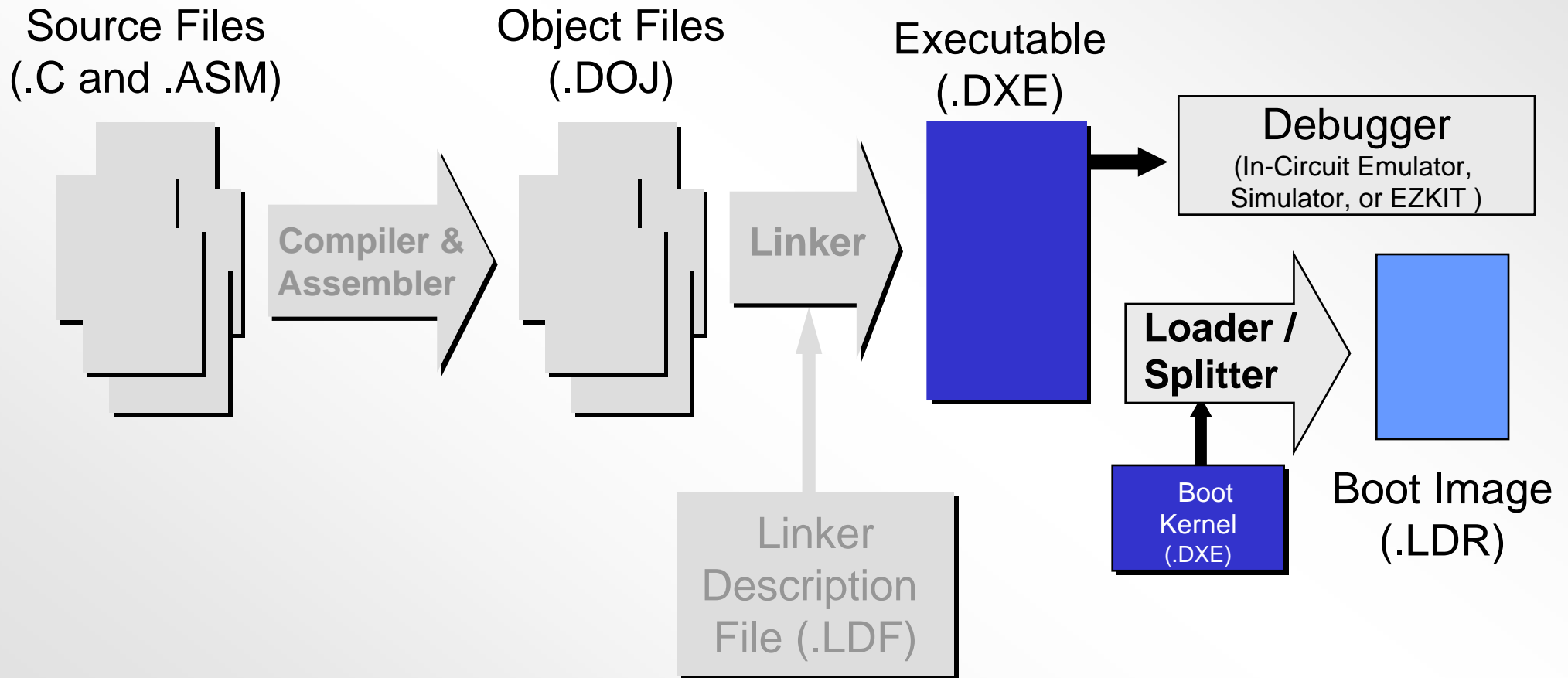
# Programowa Budowa Potoku

## Krok 2 - Linkowanie



# Programowa Budowa Potoku

## Krok 3 - Debugowanie



# **Symulator VisualDSP++**

**Odnosnik:  
VisualDSP++ User's Guide for ADSP-21xxx DSPs**

# Cechy Symulatora VisualDSP++

- **Symulator na poziomie instrukcji**
- **Debugowanie kodow zrodlowych ( C, Asm, lub mieszane)**
- **Profile kodu C i Asm**
- **Wykonanie instrukcji programow C i Asemblerowych**
- **Ustawnienie Breakpoint'ow (“watchpoints”)**
  - Wlaczanie (lub nie) zakres pamieci
  - Odczyt lub zapis roznych wartosci (takze specyficznych)
  - Stack Overflows and Underflows
- **Tworzy okna rejestrow**
- **Symuluje standardowe wej/wyj,przerwania i strumienie,**



# VisualDSP++ Kontrola Debugowania Breakpoints

Dft.asm (Read Only)

```
.SECTION/PM pm_code; /* Example setup for DFT routine */
start: M1=1;
[040100] m1=0x1;
 M9=1;
[040101] m9=0x1;
 B0=input;
[040102] b0=0x50000;
 L0=@input; /* Input buffer is circular */
[040103] l0=0x40;
 I1=imag;
[040104] i1=0x50080;
 L1=0;
[040105] l1=0;
 MODE1 = 0x1000000; /* Enable Circular buffer */
[040106] mode1=0x1000000;
 CALL dft (DB); /* Example delayed call instruction */
[040107] call dft (db); /* In delay field of call */
 I2=real; /*
[040108] i2=0x50040; '
 L2=0; */
[040109] l2=0;
end: IDLE;
[04010A] idle;
```

Breakpoints

Breakpoint Properties

Break at: end

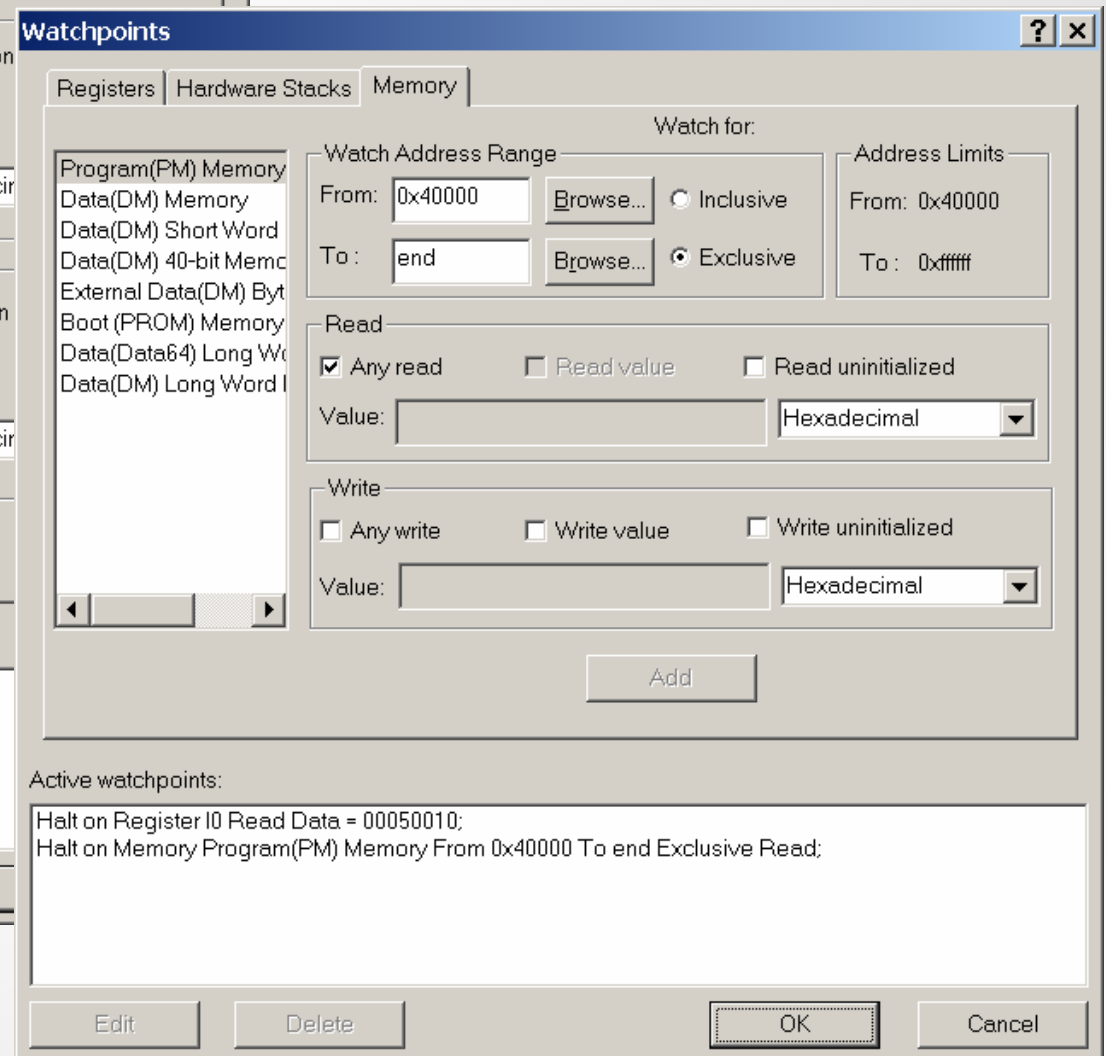
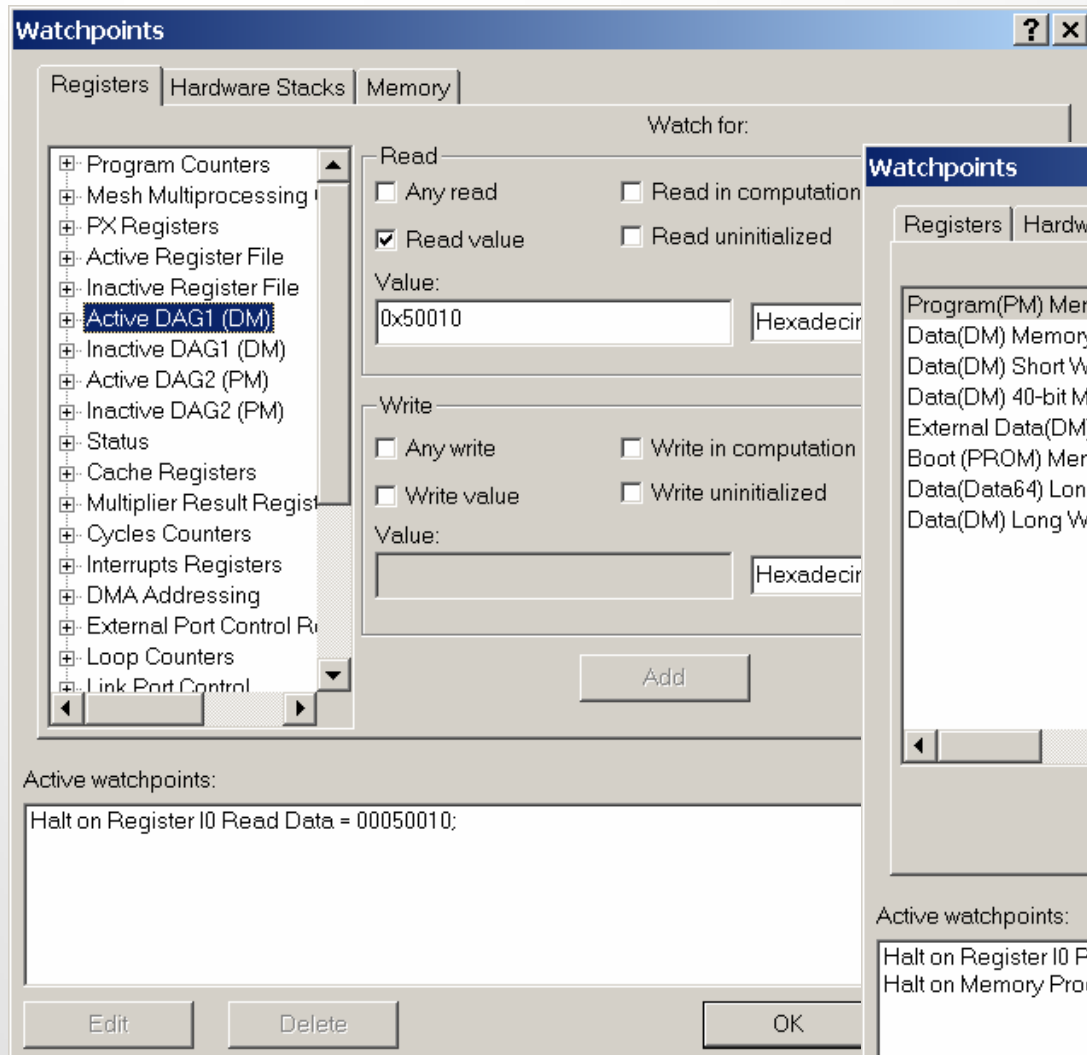
Expression:

Skip count:

Breakpoint list:

- at "Dft.asm" .34
- at "Dft.asm" .44

# VisualDSP++ Kontrola Debugowania Watchpoints



# VisualDSP++ Kontrola Debugowania

- **Single Step ( Step into )**
  - Jedna instrukcja na raz
- **Step Out Of, Step Over**
  - Uzywana debugujac kod C
- **External Interrupts**
  - Ustawia liczbe cykli instrukcji pomiedzy przerwaniami
  - Mozliwy jest przypadkowy odstep pomiedzy przerwaniami
- **Stream I/O**
  - Uzywane do symulowania wej/wyj, portow szeregowych i rownoległych
  - Przydziela pliki danych jako zrodlo- cel

# VisualDSP++ Okna Debuggera

- **Disassembly Window**
  - Widok kodu asm przeasemblowanego
- **Source Window**
  - C, mieszane C/asm
- **Local Window**
  - Pokazuje wszystkie zmienne lokalne włączając bierzącą funkcję
- **Expressions Window**
  - Dowolne wyrażenia w języku "C"
  - Nazwy rejestrów zaczynają się od \$ (przykład \$R12)
- **Trace Window (tylko symulacja)**
  - Wyświetla historię wykonania instrukcji
  - Specyficzna głębokość ścieżek
- **Profile Window**
  - Licznik cykli i procentowe zużycie czasu wykonując specyficzne zakresy adresów
- **Plot**
  - Możliwość zwiększania wykresów

# Visual DSP++ Okna Debuggera

- **Okna rejestru ogolnego**
  - Daja dostep do wszystkich rejestrow SHARC'a
  - Tworzy pozostale okna
- **Okna pamieci**
  - 2-kolumny, 3-kolumny lub 4-kolumny
  - Daja dostep do pamieci SHARC'ow
  - supports, memory fill, dump, mapa pamieci
- **Prawe klikniecie na ktorekolwiek okno dla opcji wyswietlania lub funkcjonalnosci**

# Zwolnienie/Zapelnienie pamieci

- **Zwolnienie/Zapelnienie pamieci**

- Zwolnienie poprzez zrzut zawartosci do pliku.
- Zapelnienie pamieci specjalna wartoscia lub z pliku.  
(pliki wymagaja jednego znaku ASCII na linie)
- Dozwolone rozne formaty

Format zmiennej jest zapisany w pierwszej linii jako domyslony dlatego debugger potrafi odczytac plik poprawnie. Moze to byc sprawdzone w oknie dialogowym Memory-Dump.

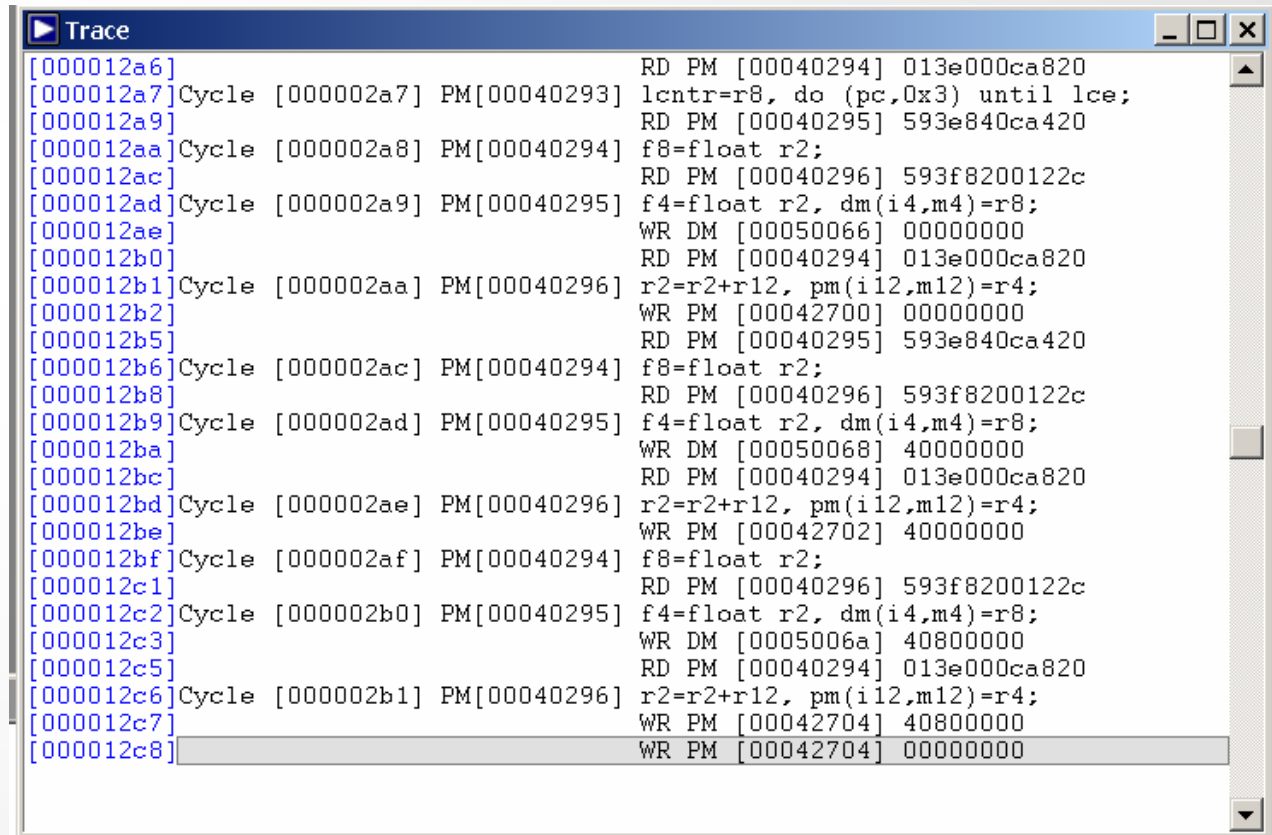
# Okno Trace

- **Wyswietlanie**

- *Trace Address:* Adres w sciezce bufora.
- *Cycle:* Kolejne wartosci cykli DSP.
- Adres bierzacej instrukcji
- Dostep do pamieci w formie wyrazenia (Mem=Value lub Value=Mem)

- **Ustawienia**

- *Set Depth:* Liczba scizek. Bufory to dlugich scizekach spowalniaja debugger.
- *Clear:* Czysci bierzaca sciezke
- *Enable:* Zezwala lub zabrania na dostep do sciezki.



```
Trace
[000012a6] RD PM [00040294] 013e000ca820
[000012a7] Cycle [000002a7] PM[00040293] lcntr=r8, do (pc,0x3) until lce;
[000012a9] RD PM [00040295] 593e840ca420
[000012aa] Cycle [000002a8] PM[00040294] f8=float r2;
[000012ac] RD PM [00040296] 593f8200122c
[000012ad] Cycle [000002a9] PM[00040295] f4=float r2, dm(i4,m4)=r8;
[000012ae] WR DM [00050066] 00000000
[000012b0] RD PM [00040294] 013e000ca820
[000012b1] Cycle [000002aa] PM[00040296] r2=r2+r12, pm(i12,m12)=r4;
[000012b2] WR PM [00042700] 00000000
[000012b5] RD PM [00040295] 593e840ca420
[000012b6] Cycle [000002ac] PM[00040294] f8=float r2;
[000012b8] RD PM [00040296] 593f8200122c
[000012b9] Cycle [000002ad] PM[00040295] f4=float r2, dm(i4,m4)=r8;
[000012ba] WR DM [00050068] 40000000
[000012bc] RD PM [00040294] 013e000ca820
[000012bd] Cycle [000002ae] PM[00040296] r2=r2+r12, pm(i12,m12)=r4;
[000012be] WR PM [00042702] 40000000
[000012bf] Cycle [000002af] PM[00040294] f8=float r2;
[000012c1] RD PM [00040296] 593f8200122c
[000012c2] Cycle [000002b0] PM[00040295] f4=float r2, dm(i4,m4)=r8;
[000012c3] WR DM [0005006a] 40800000
[000012c5] RD PM [00040294] 013e000ca820
[000012c6] Cycle [000002b1] PM[00040296] r2=r2+r12, pm(i12,m12)=r4;
[000012c7] WR PM [00042704] 40800000
[000012c8] WR PM [00042704] 00000000
```

# Profiler

| Linear Profiling Results |        |          |        |         |                                                                      |
|--------------------------|--------|----------|--------|---------|----------------------------------------------------------------------|
| Histogram                | %      | Execu... | %      | Line... | C:\Program Files\Analog Devices\VisualDSP\211xx\Examples\C_Exampl... |
|                          | 29.36% | main()   |        | 19      | float dm output_SIMD=0;                                              |
|                          | 1.70%  | PC[0...  |        | 20      |                                                                      |
|                          | 1.13%  | PC[0...  |        | 21      | int count_start();                                                   |
|                          | 1.13%  | PC[0...  |        | 22      | int count_end(int);                                                  |
|                          | 1.13%  | PC[0...  |        | 23      | volatile int time_temp, SISD_cycle_count, SIMD_cycle_count; /* f...  |
|                          | 1.11%  | PC[0...  |        | 24      |                                                                      |
|                          | 1.11%  | PC[0...  | 0.01%  | 25      | void main(void)                                                      |
|                          | 1.11%  | PC[0...  |        | 26      | {                                                                    |
|                          | 0.73%  | PC[0...  |        | 27      | int i;                                                               |
|                          | 0.57%  | PC[0...  |        | 28      |                                                                      |
|                          | 0.57%  | PC[0...  |        | 29      | for (i = 0; i < N; i++)                                              |
|                          | 0.57%  | PC[0...  |        | 30      | {                                                                    |
|                          | 0.57%  | PC[0...  | 5.67%  | 31      | x_input[i] = i;                                                      |
|                          | 0.57%  | PC[0...  | 11.50% | 32      | y_input[i] = i;                                                      |
|                          | 0.57%  | PC[0...  |        | 33      | }                                                                    |
|                          | 0.57%  | PC[0...  |        | 34      |                                                                      |
|                          | 0.57%  | PC[0...  | 0.06%  | 35      | time_temp = count_start(); /* Benchmark routine start */             |
|                          | 0.57%  | PC[0...  |        | 36      | for (i = 0; i < N; i++)                                              |
|                          | 0.57%  | PC[0...  | 5.93%  | 37      | output_SISD += x_input[i] * y_input[i];                              |
|                          | 0.57%  | PC[0...  | 0.06%  | 38      | SISD_cycle_count = count_end(time_temp); /* Benchmark routine...     |
|                          | 0.57%  | PC[0...  |        | 39      |                                                                      |
|                          | 0.57%  | PC[0...  | 0.09%  | 40      | printf("The cycle count for SISD execution is %d cycles.\n",S...     |
|                          | 0.57%  | PC[0...  |        | 41      |                                                                      |
|                          | 0.57%  | PC[0...  | 0.06%  | 42      | time_temp = count_start(); /* Benchmark routine start */             |
|                          | 0.57%  | PC[0...  |        | 43      | #pragma SIMD_for                                                     |
|                          | 0.57%  | PC[0...  |        | 44      | for (i = 0; i < N; i++)                                              |
|                          | 0.57%  | PC[0...  | 5.93%  | 45      | output_SIMD += x_input[i] * y_input[i];                              |
|                          | 0.57%  | PC[0...  | 0.06%  | 46      | SIMD_cycle_count = count_end(time_temp); /* Benchmark routin...      |
|                          | 0.57%  | PC[0...  |        | 47      |                                                                      |
|                          | 0.57%  | PC[0...  |        | 48      | printf("The cycle count for SIMD execution is %d cycles.\n",S...     |
|                          | 0.57%  | PC[0...  |        | 49      |                                                                      |
|                          | 0.57%  | PC[0...  |        | 50      | exit(0);                                                             |
|                          | 0.57%  | PC[0...  |        | 51      | }                                                                    |

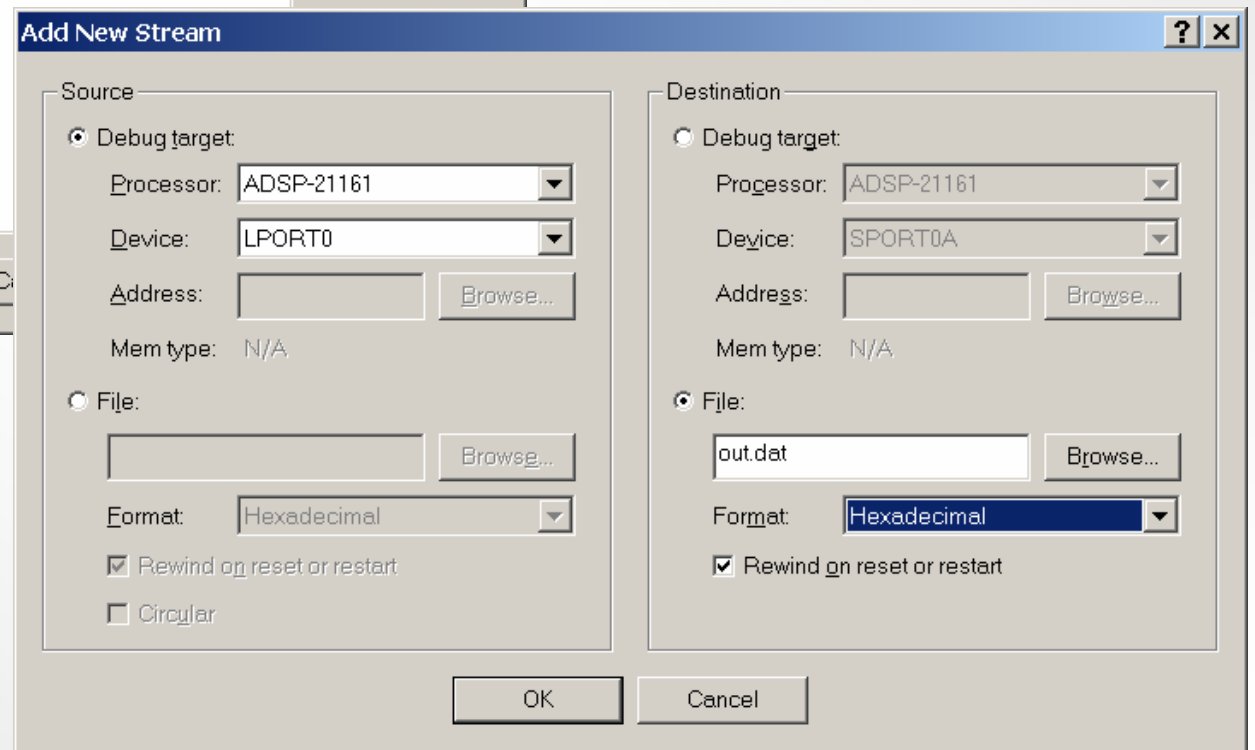
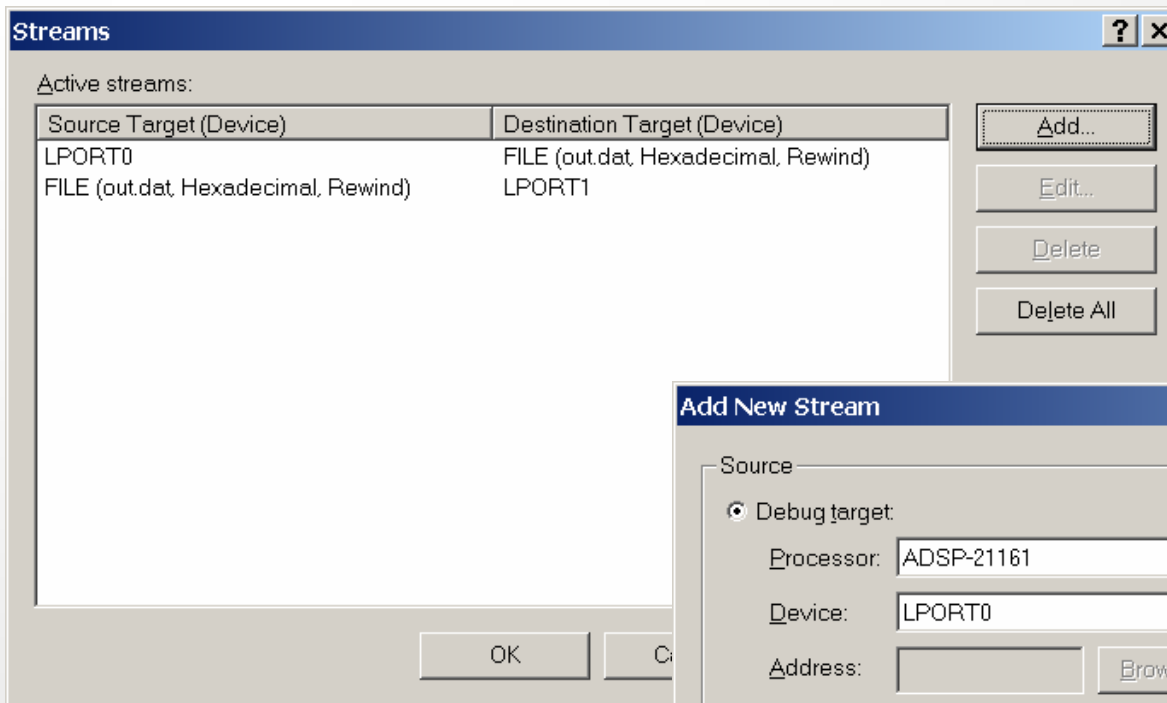
Total Samples: 8823

Elapsed Time: 00:00:01

Enabled



# Strumienie



# Wykresy

**Plot Configuration** [?] [X]

**Data sets:**

- Input

**Plot**

Type: Line Plot

Title: Test Plot

**Data Setting**

Name: Sine

Memory: Data(DM) Memory

Address: sine [Browse...] Offset: 0

Count: 64 Row count: 10

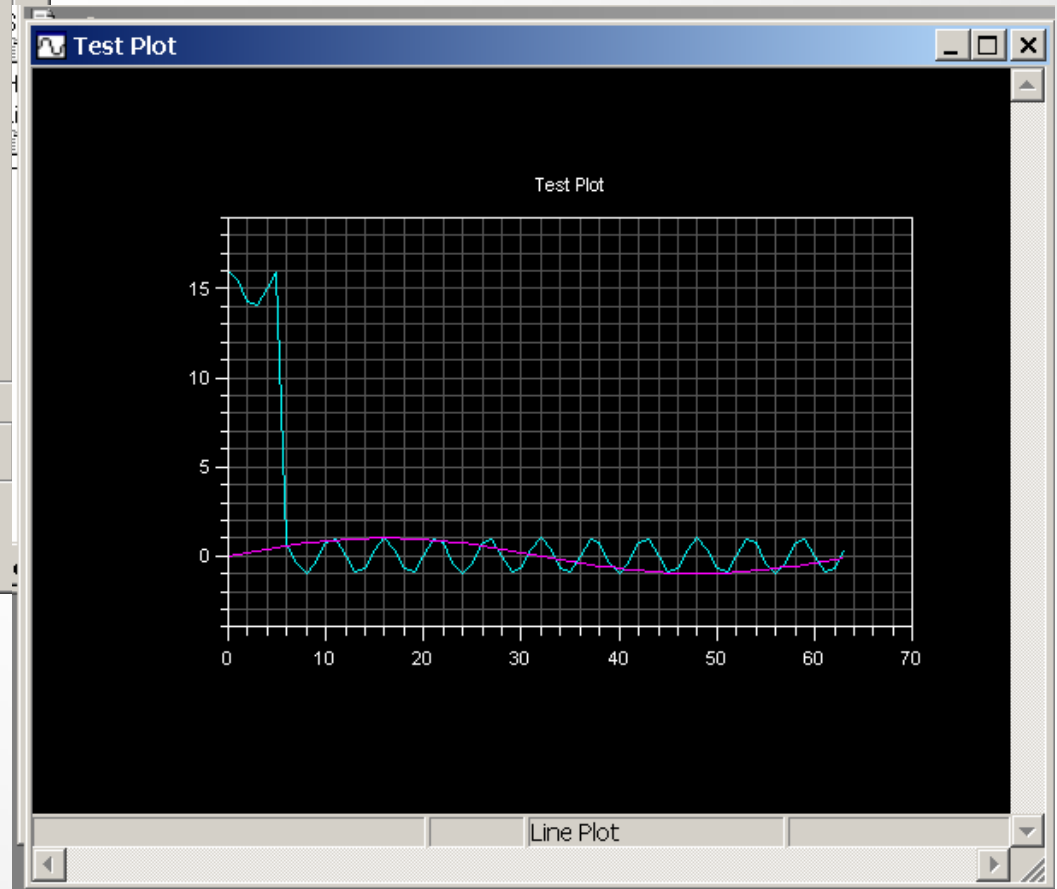
Stride: 1 Column count: 10

Data: float

**Axis Selection**

X  Y  Z

OK Cancel Settings...



# Run to Main & STDIO

- **Run To Main**
  - Pozwala użytkownikowi kontrolować (lub nie) debugger, rozpoczęcie wykonywania w nagłówku 'run time'.
- **STDIO**
  - Pełne wsparcie STDIO. Użycie `printf()` i `scanf()` na dostęp do plików systemowych hosta.

