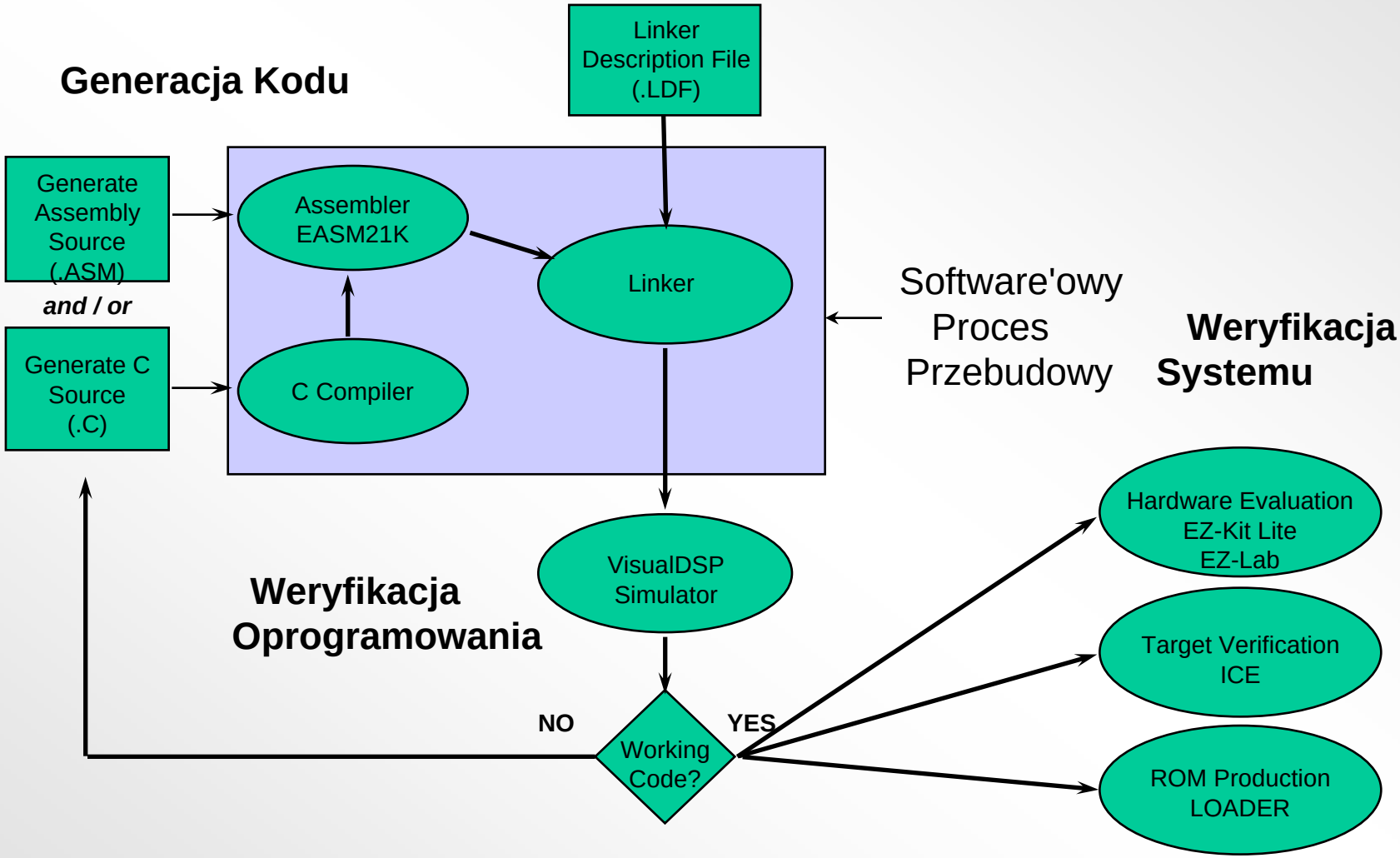
Asemblacja Kodu w VisualDSP++

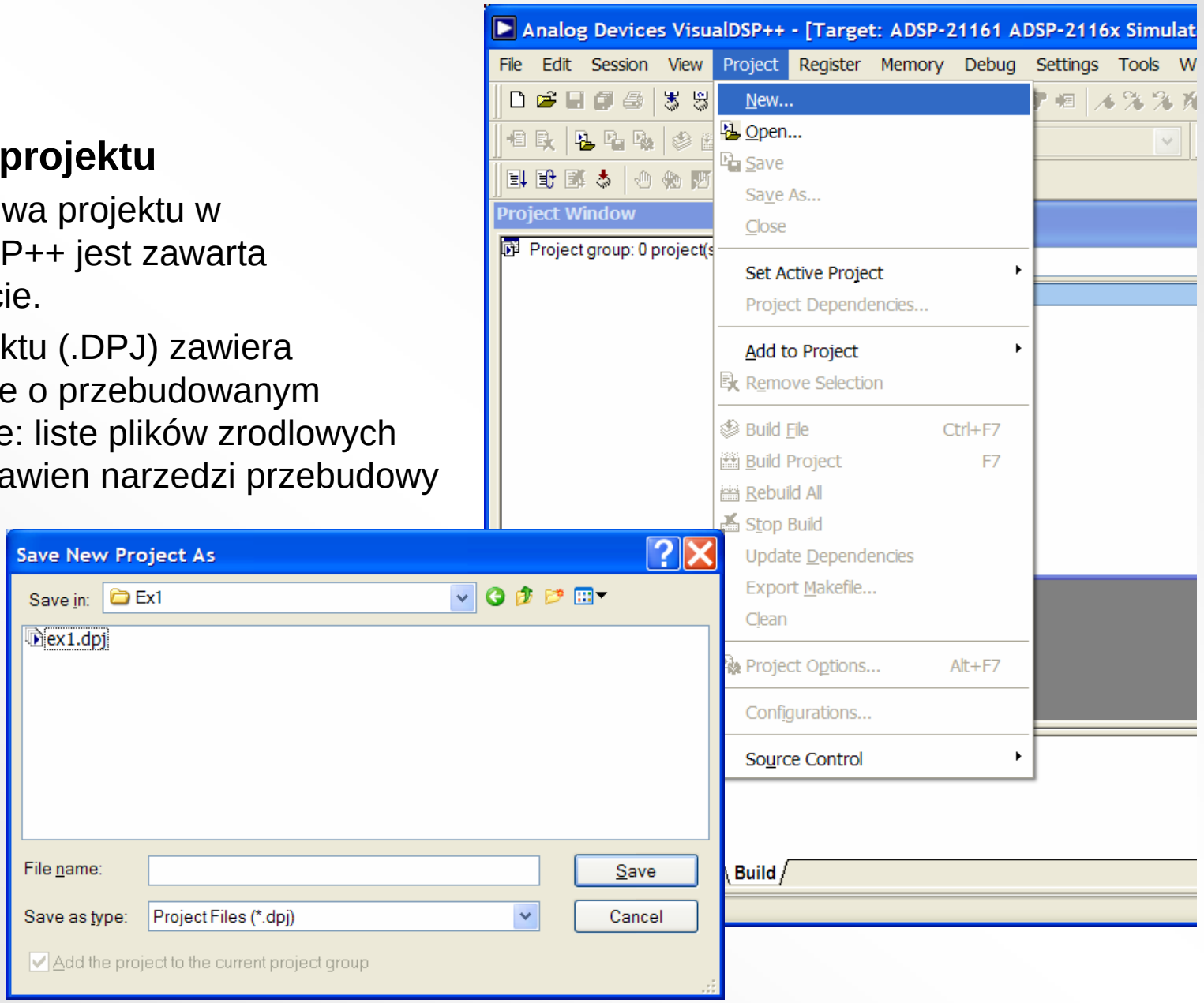
Sekcja 6

Programowe Tworzenie Potoku

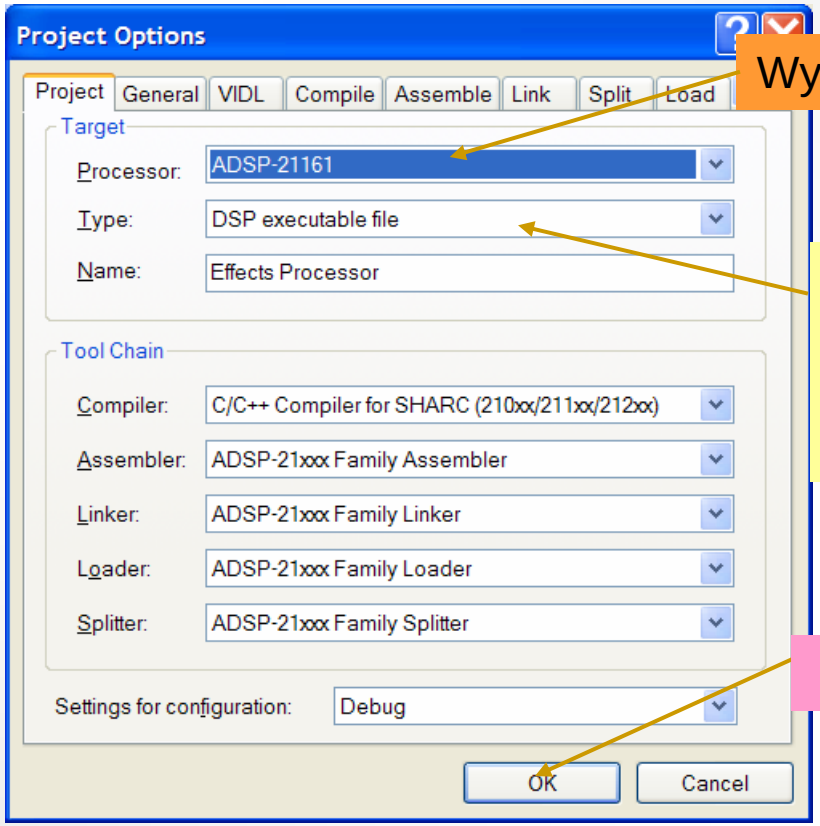


Budowa Projektu

- **Tworzenie projektu**
  - przebudowa projektu w VisualDSP++ jest zawarta w projekcie.
  - Plik projektu (.DPJ) zawiera informacje o przebudowanym programie: liste plików zrodlowych opcje ustawien narzedzi przebudowy



Opcje Projektu



Wybierz procesor

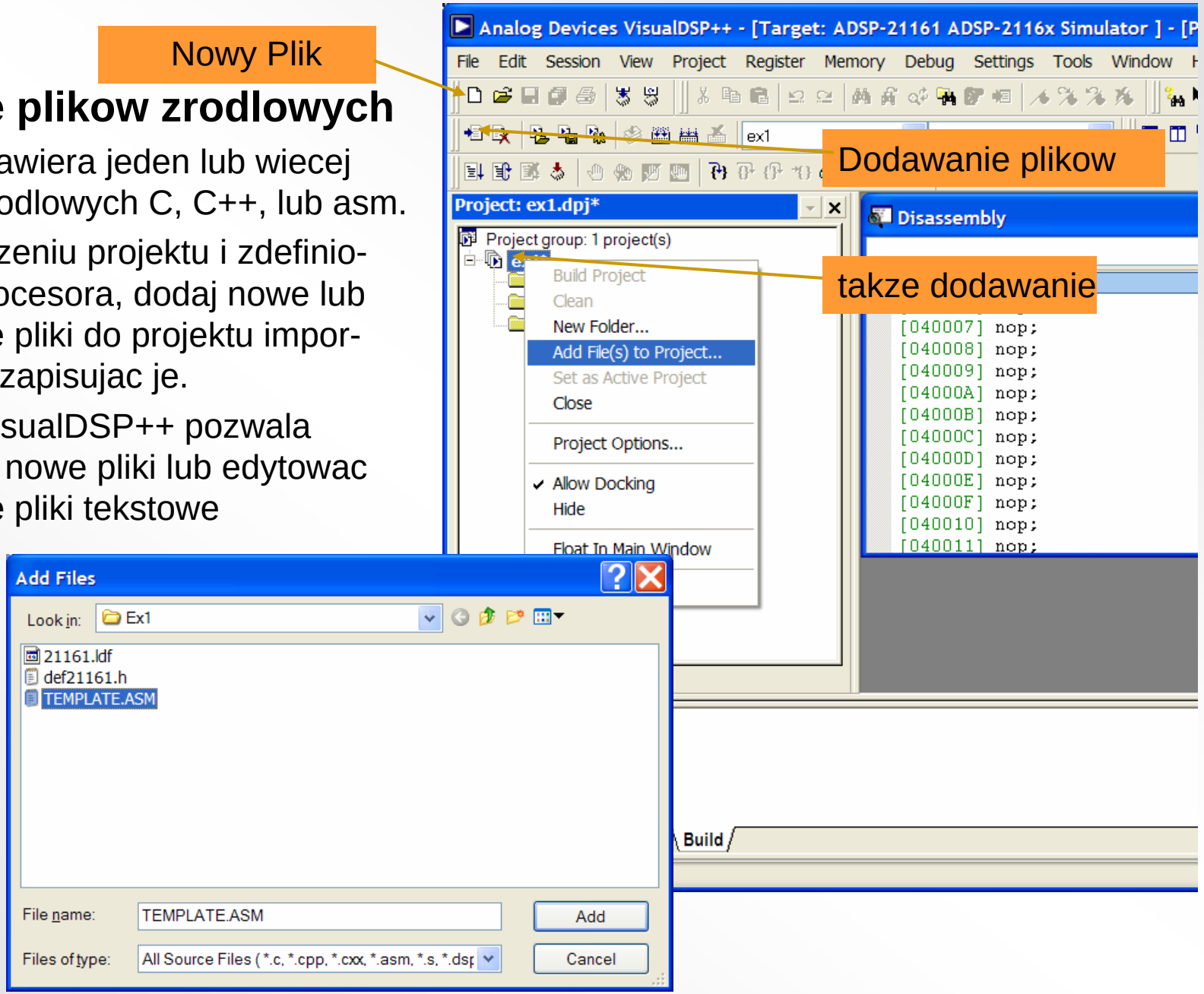
Ustaw typ pliku .exe, wymagane przez debuger

Zatwierdz

Pojawi sie nowe okno. Jesli nie uzywasz systemu operacyjnego VDK, wcisnij **NO**.

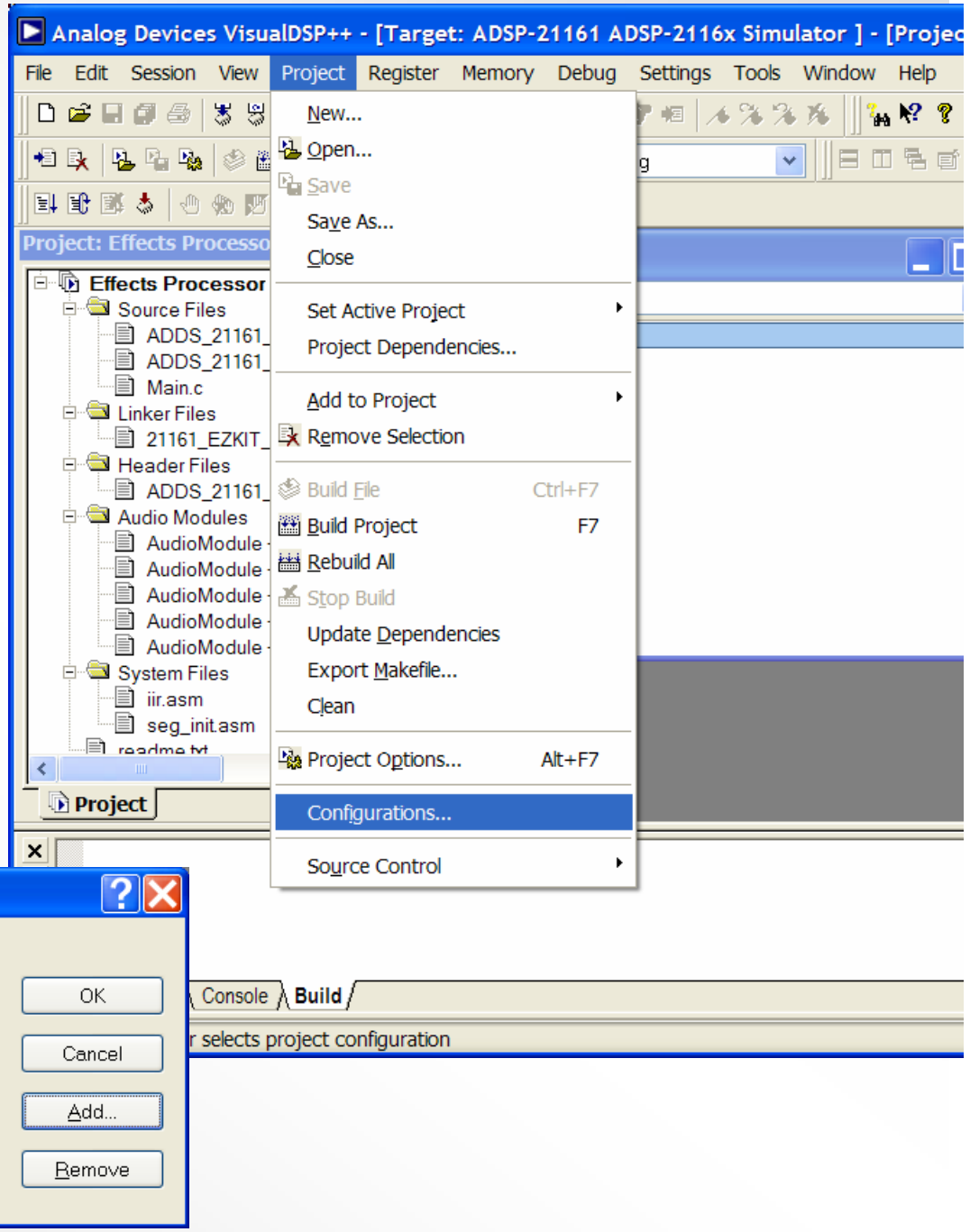
Kroki Tworzenia Projektu

- **Tworzenie plikow 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

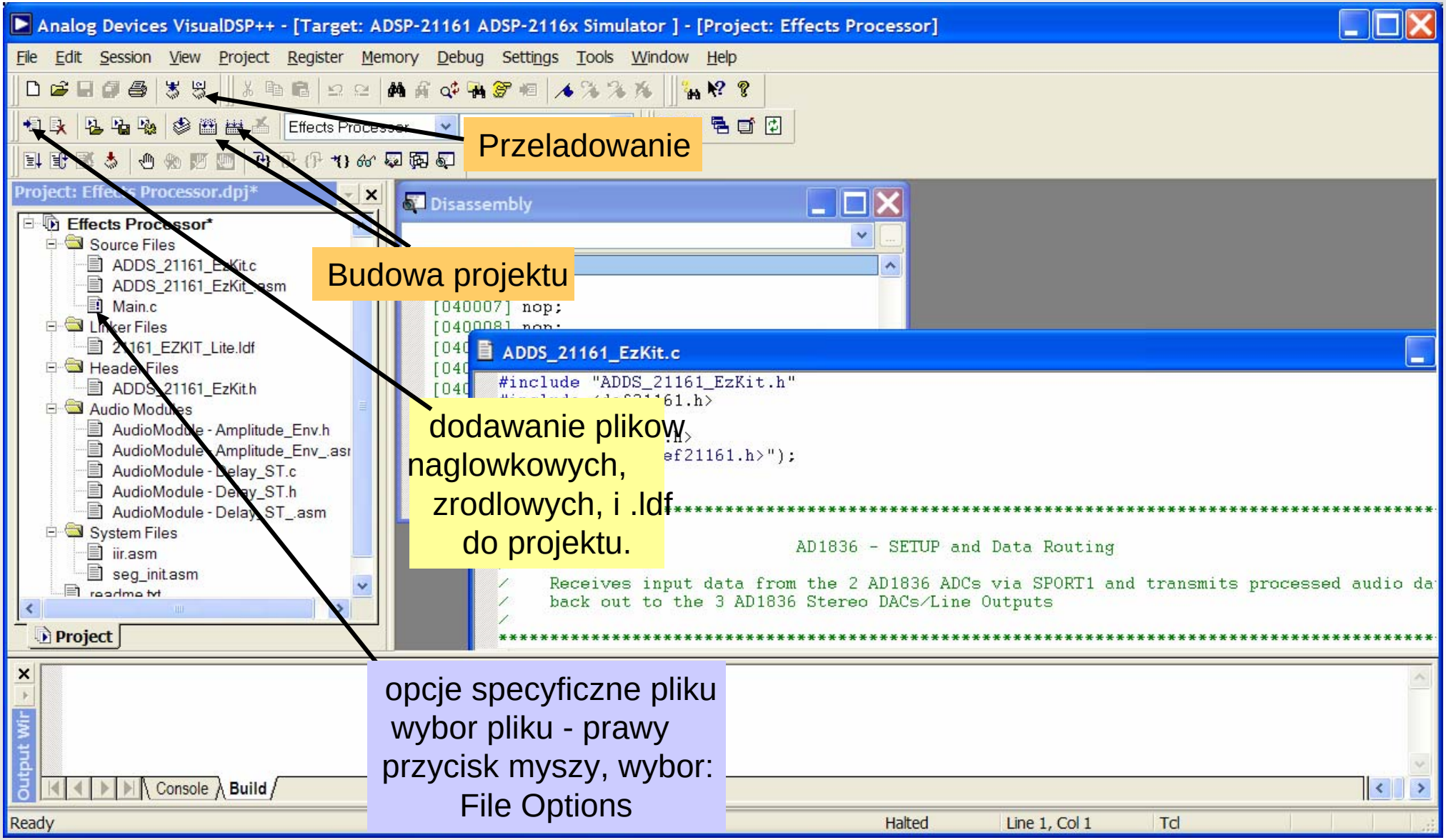


Kroki Rozbudowy Projektu

- Definiowanie opcji budowy projektu
  - Ustawienia konfiguracji projektu kontroluje przebudowe. 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 optymali-zowane dla zwiekszenia wydajnosci

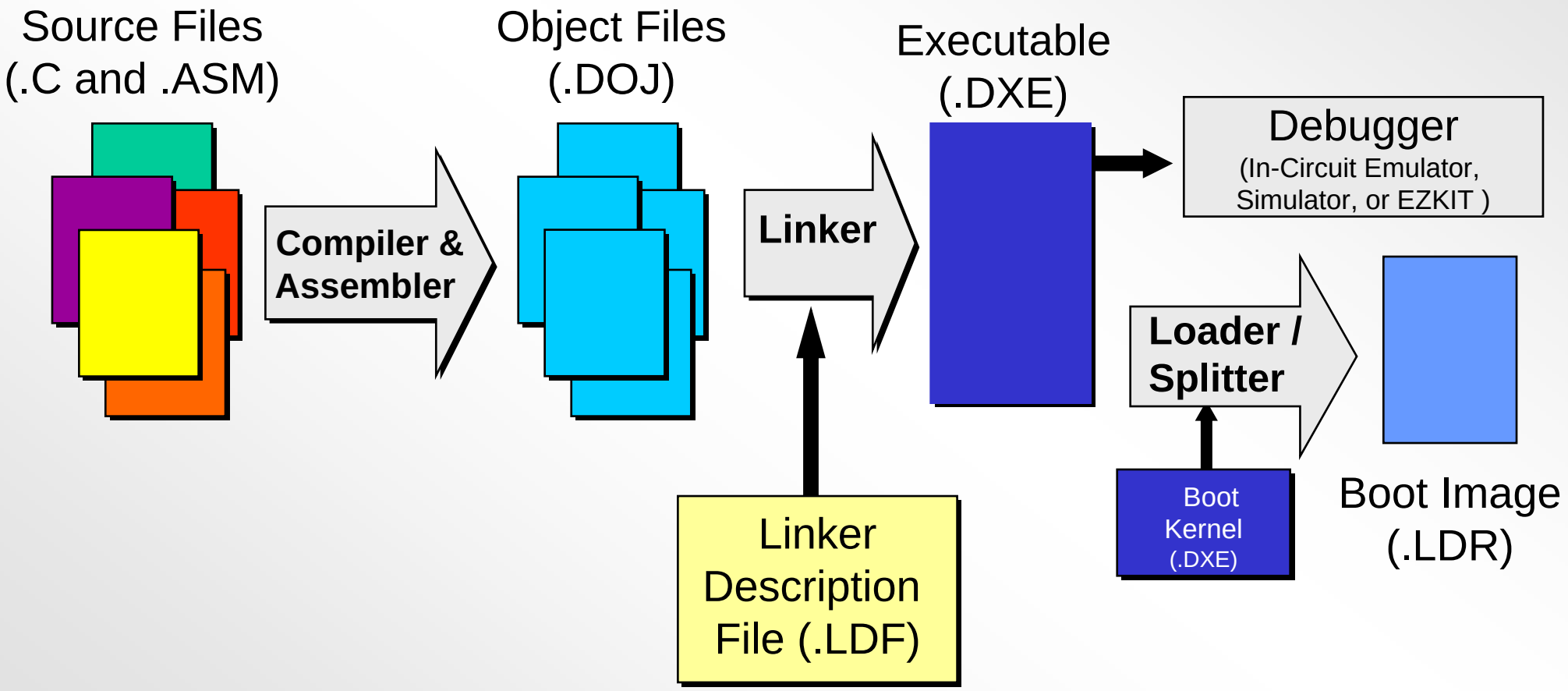


Menu VisualDSP++



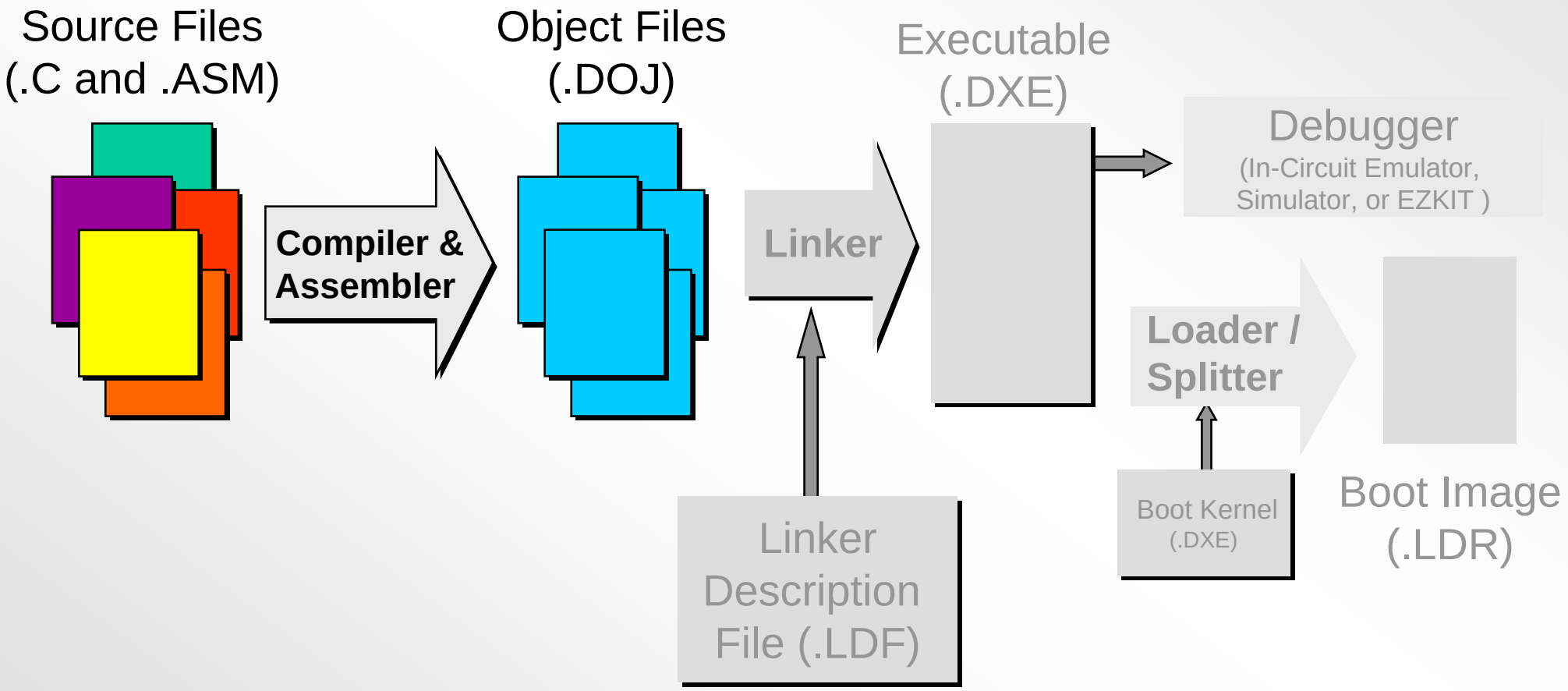
Programowa Budowa Potoku

Jakie pliki sa uzywane?



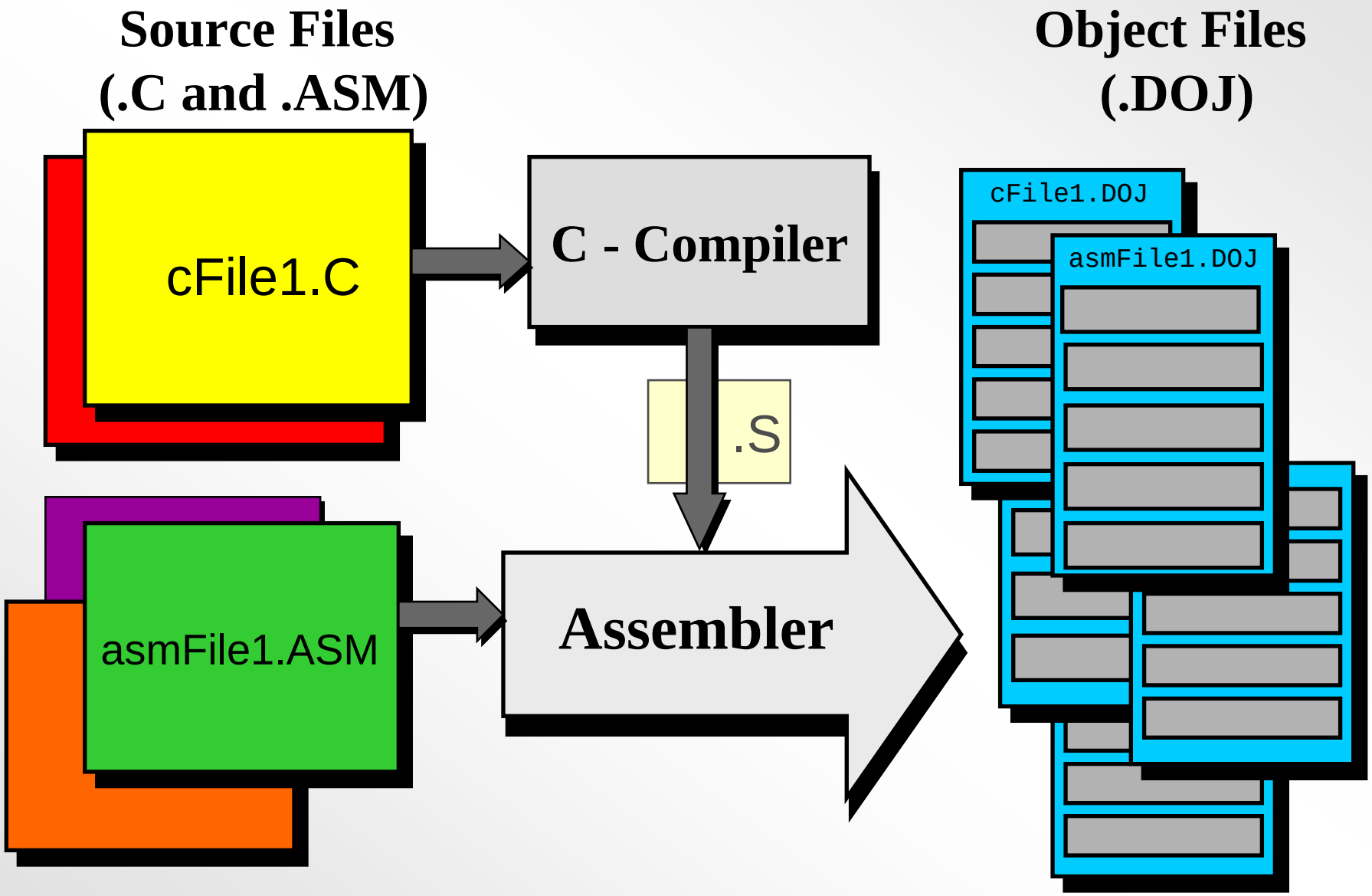
Programowa Budowa Potoku

Jakie pliki sa uzywane?



Programowy Proces Budowy

Krok 1 - Kompilacja i Asemblacja



Programowy Proces Budowy

Krok 1 - Przyklad: Asemblacja Zrodla

asmFile1.ASM

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

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

Assembler

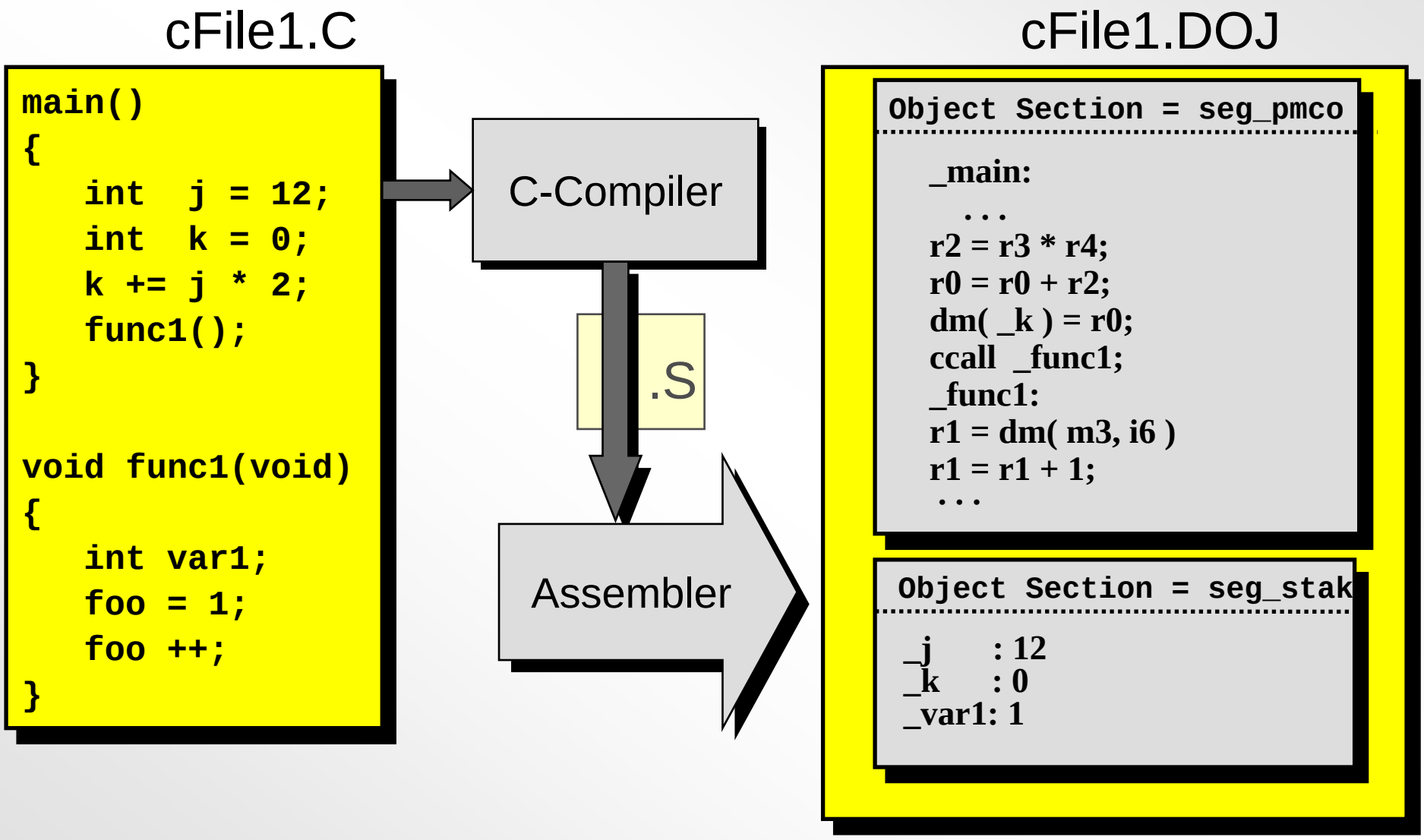
asmFile1.DOJ

```
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 - Przyklad: plik C

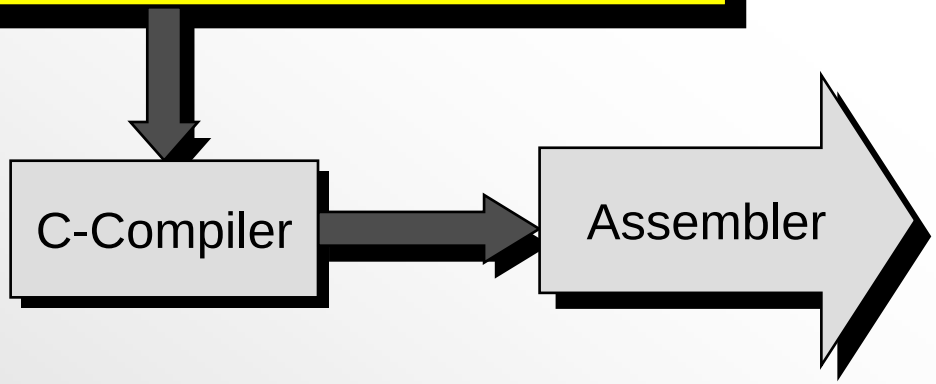


Programowy Proces Budowy

Krok 1 Przyklad: plik C z alternatywnymi sekcjami

foo.C

```
section (“extern”) int array[256];

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



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 poczatek sekcji sasiednich pamieci
.PREVIOUS	Odwoluje do poprzednio zaznaczonej .SECTION
*** .SEGMENT**	Komenda wymienna z .section
*** .ENDSEG**	
.ALIGN	Okresla ustawienie danych
.VAR	Definicja i inicjalizacja buforow danych i zmiennych
.GLOBAL	Umozliwia dostep z innych plikow zrodlowych
.EXTERN	Odwolanie do odnosnika w innych plikach zrodlowych
.PRECISION 32 **.PRECISION 40** **.ROUND_NEAREST** **.ROUND_MINUS** **.ROUND_PLUS** **.ROUND_ZERO**	Ustawienie precyzji dla inicjalizowanych stalych i zmiennych (tylko zmiennoprzecinkowe) Ustawienie trybu round dla inicjalizowanych stalych i zmiennych gdy wartosc nie pasuje docelowemu formatowi.

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
>	Wieksze
>=	Wieksze 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 */
```

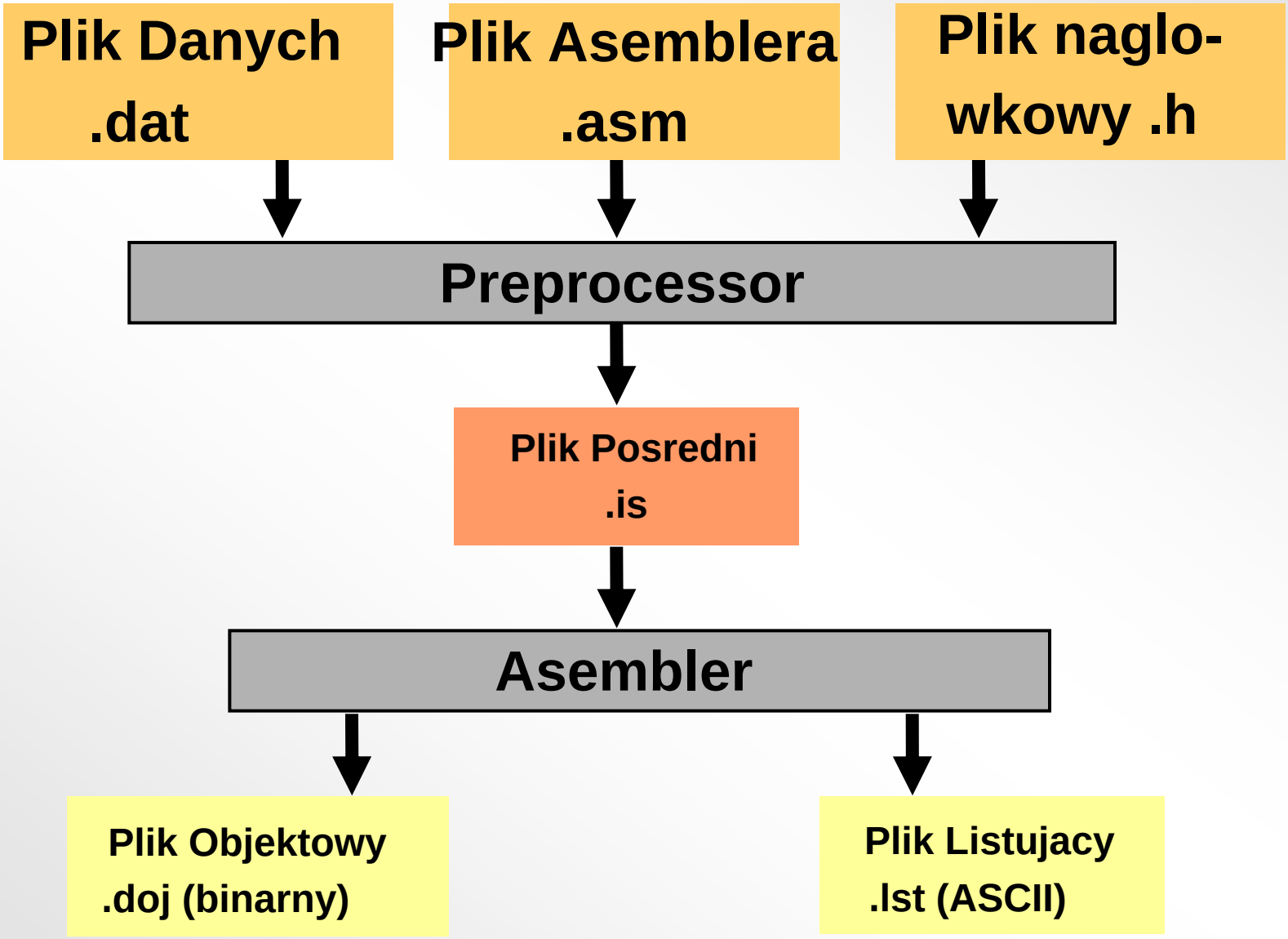
Asemblerowe Dyrektywy Preprocesora

Komenda	Opis
#define	Definiuje makro
#if **#else** **#elif** **#endif**	Poczatek pary #if …#endif Identifikuje instrukcje alternatywne wewnarz #if …#endif Dzieli pare #if … #endif Zakonczenie pary #if …#endif
#error	Generuje wiadomosc o bledzie
#ifdef	Rozpoczecie pary #ifdef … #endif i sprawdzanie def makra
#ifndef	Rozpoczecie pary #ifdef … #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

Przyklad Uzycia 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

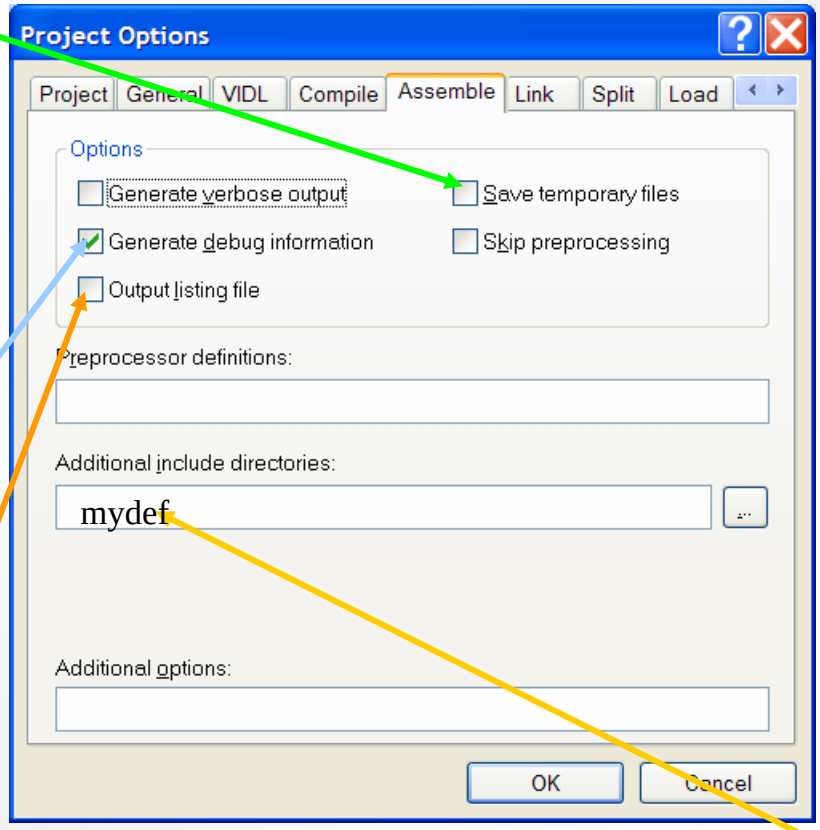

Plik Danych
.dat
Plik Asemblera
.asm
Plik naglo-
wkowy .h
Preprocessor
Plik Posredni
.is
Asembler
Plik Objektowy
.doj (binarny)
Plik Listujacy
.lst (ASCII)

Zakladka 'Assemble'

Zaznaczenie stworzy plik tymcza-sowy .is

Zaznaczone, mozna debug'owac kod zrodlowy

Zaznaczone, stworzy plik z listingiem




```
#include <def21161.h>
#include “myheader.h”

#ifdef mydef
R0 = R0 + 1;
#else
R0 = R0 - 1;
#endif
```

Zaleznie od definicji, mozna wybrac rozne kody

Uzycie def21161.h definicji Bitowego Rejestru Systemu

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

    Przyklady:

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

- Indywidualne definicje bitow moga byc **OR**'owane operatorem " | "

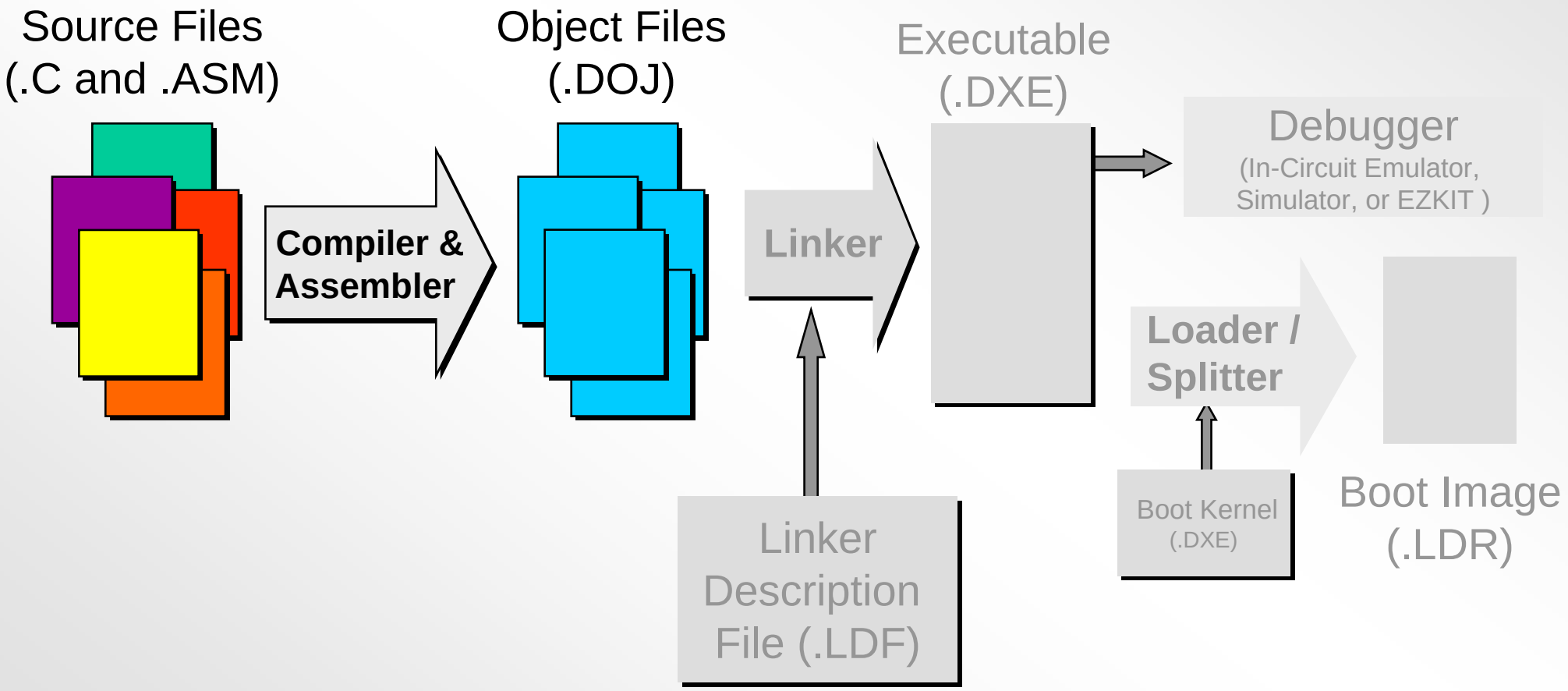
- Przyklady:

```
it set MODE1 BR0|IRPTEN|RND32;
USTAT1 = BSO|HBW8|HMSWF;
dm(SYSCON)=USTAT1;
```

- `#include <def21161.h>`

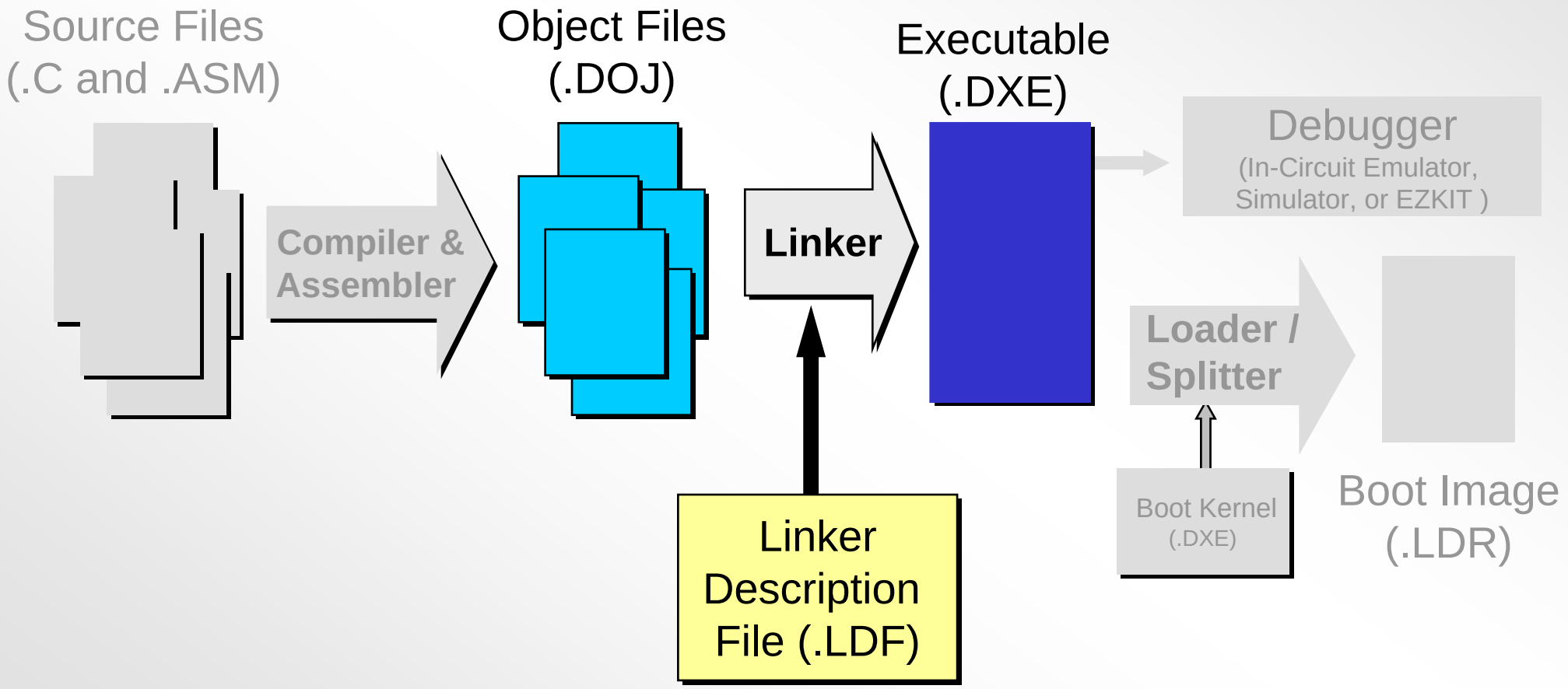
Programowa Budowa Potoku

Krok 1 - Kompilacja & Asemblacja



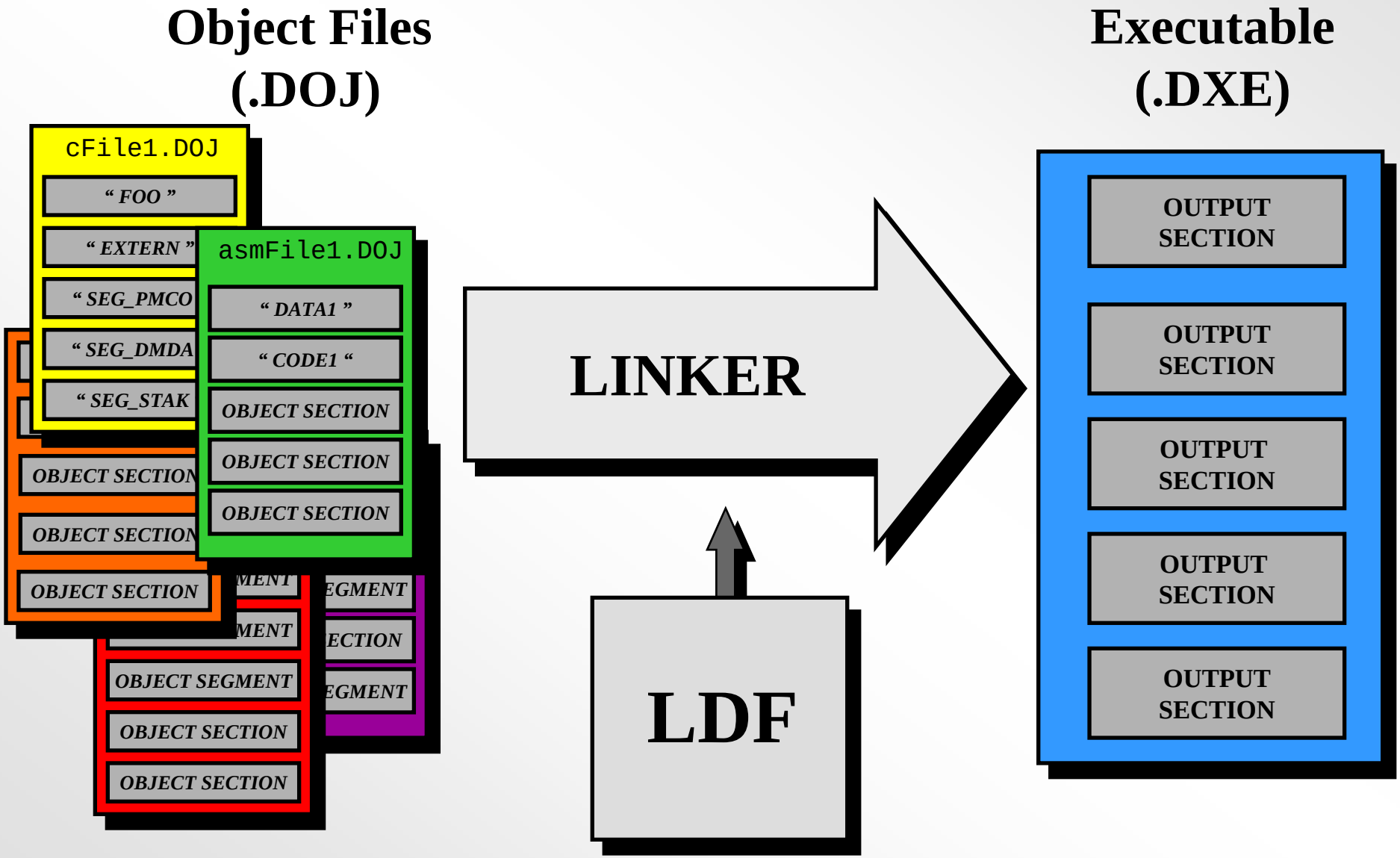
Programowa Budowa Potoku

Krok 2 - Linking



Opis Pliku Linkowania

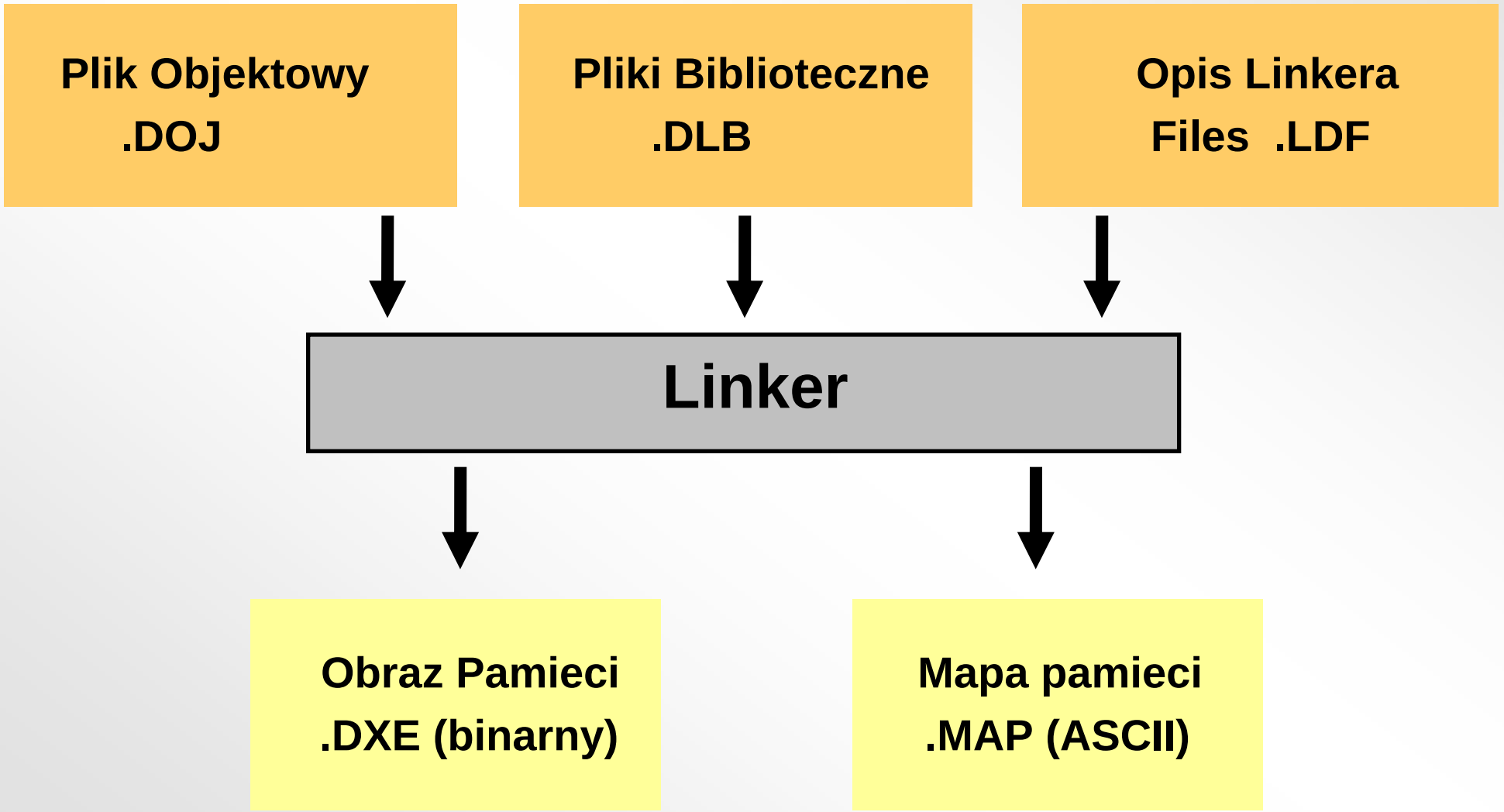
Krok 2 - Linkowanie



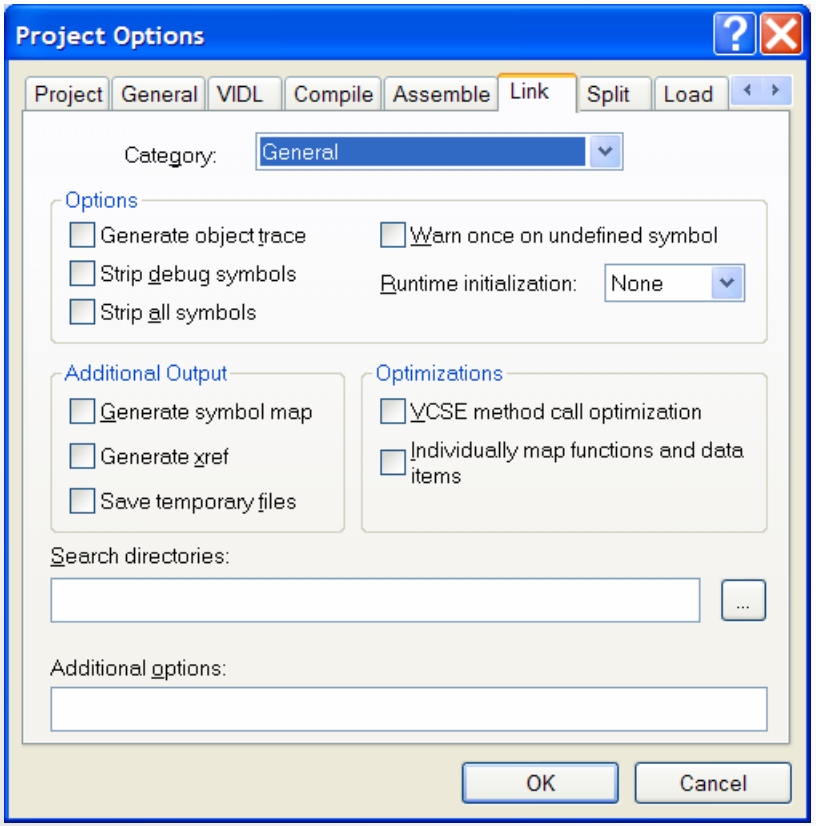
Opis Pliku Linkowania (LDF)

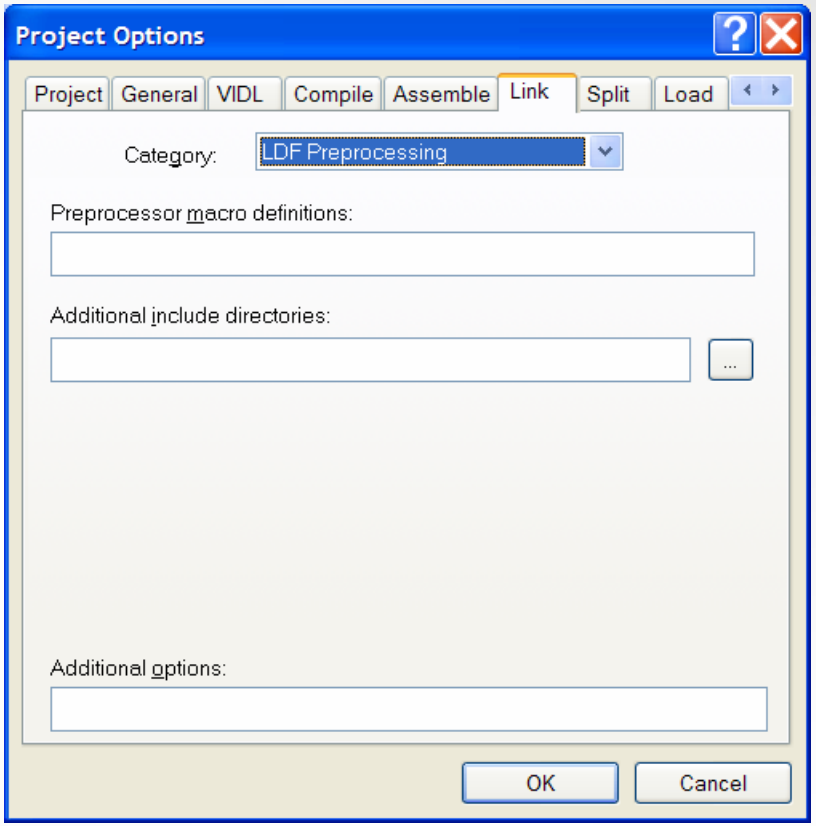
- **Jezyk komend linkowania kontroluje calkowity proces**
  - **Konfiguracja pamieci**
    - **Pamiec wewnetrzna i zewnetrzna**
    - **Pamiec dzielona**
    - **Overlays**
  - **Wejsciowe pliki linkera**
  - **Wyjsciowe pliki linkera**
  - **Linkowanie multiprocessora**

Linker


Plik Objektowy
.DOJ
Pliki Biblioteczne
.DLB
Opis Linkera
Files .LDF
Linker
Obraz Pamieci
.DXE (binarny)
Mapa pamieci
.MAP (ASCII)

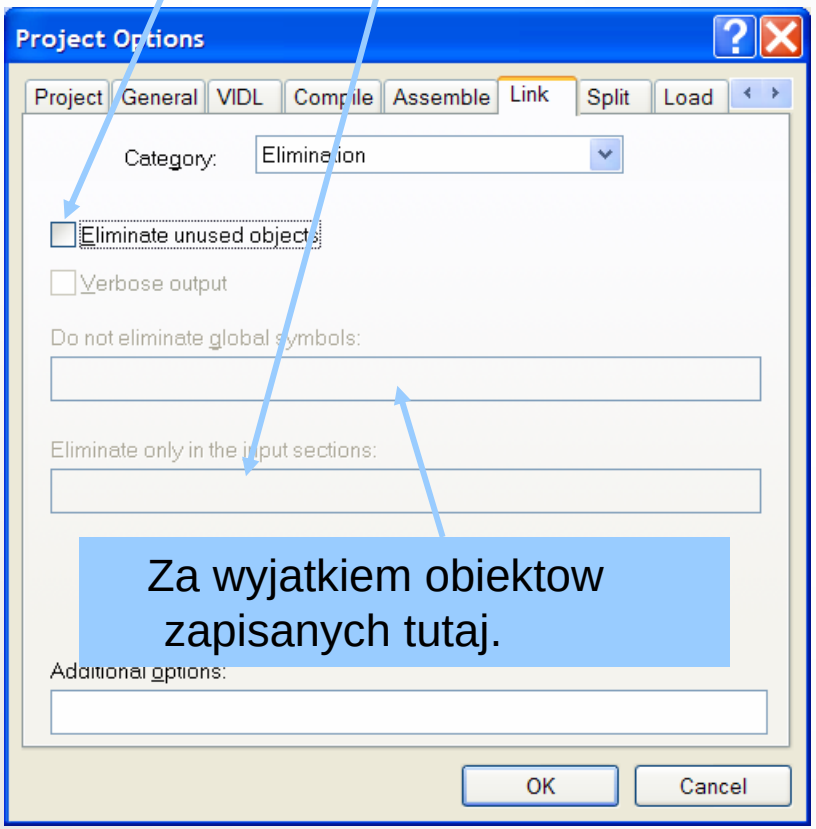
Zakladka 'Link'



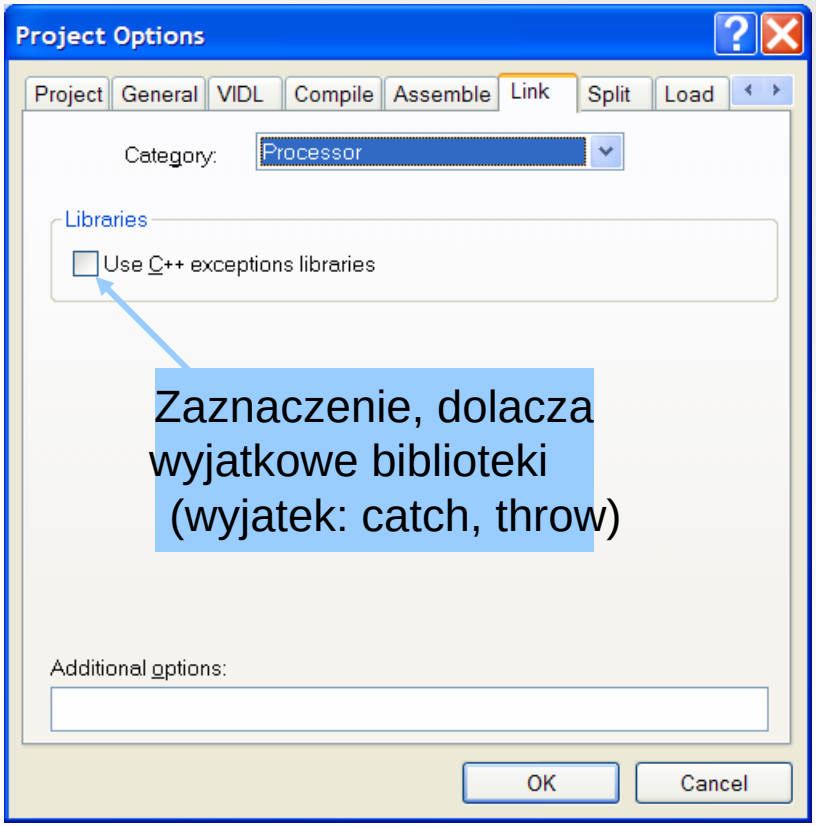


Zakladka 'Link'

Zaznaczenie - globalnie lub selektywnie eliminuje nieuzywane obiekty...



Za wyjatkiem obiektow zapisanych tutaj.



Zaznaczenie, dolacza wyjatkowe biblioteki (wyjatek: catch, throw)

Plik Opisujacy Linkowanie (LDF)

- **Komendy globalne**
  - Definiuja architekture lub procesor
  - Sciezki przeszukiwania katalogu
  - Zalaczane pliki biblioteczne i objektowe

- **Opis pamieci**
  - Definiuje segmenty pamieci
  - Definiuje miejsce pamieci multiprocesora

- **Komendy linkujace projekt**
  - Umozliwia linkowanie listy plikow objektu
  - Nazwa pliku wyjsciowego
  - Mapowanie sekcji wejsciowych do segmentow pamieci

Przyklad LDF

Komendy globalne i Opis pamieci

```
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

Przyklad 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 objektow (.DOJ) i bibliotek (.DLB) podanych w linii komend.
- $COMMAND_LINE_LINK_AGAINST
  - Lista plikow wykonawczych (.DXE, .SM) podanych w linii komend.
- $COMMAND_LINE_OUTPUT_FILE
  - Wyjsciowe pliki wykonawcze specjalnie nazwanych w linii komend znacznik -o.
- $ADI_DSP
  - Sciezka do katalogu VisualDSP.
- $macro
  - Zdefiniowane macro uzytkownika (typowo dla listy plikow.)
  - przyklad `$OBJECTS = crt.obj, $COMMAND_LINE_OBJECTS;`

Expert Linker

Zastosowanie 'LDF Wizard'

Cechy 'Expert Linker'a'

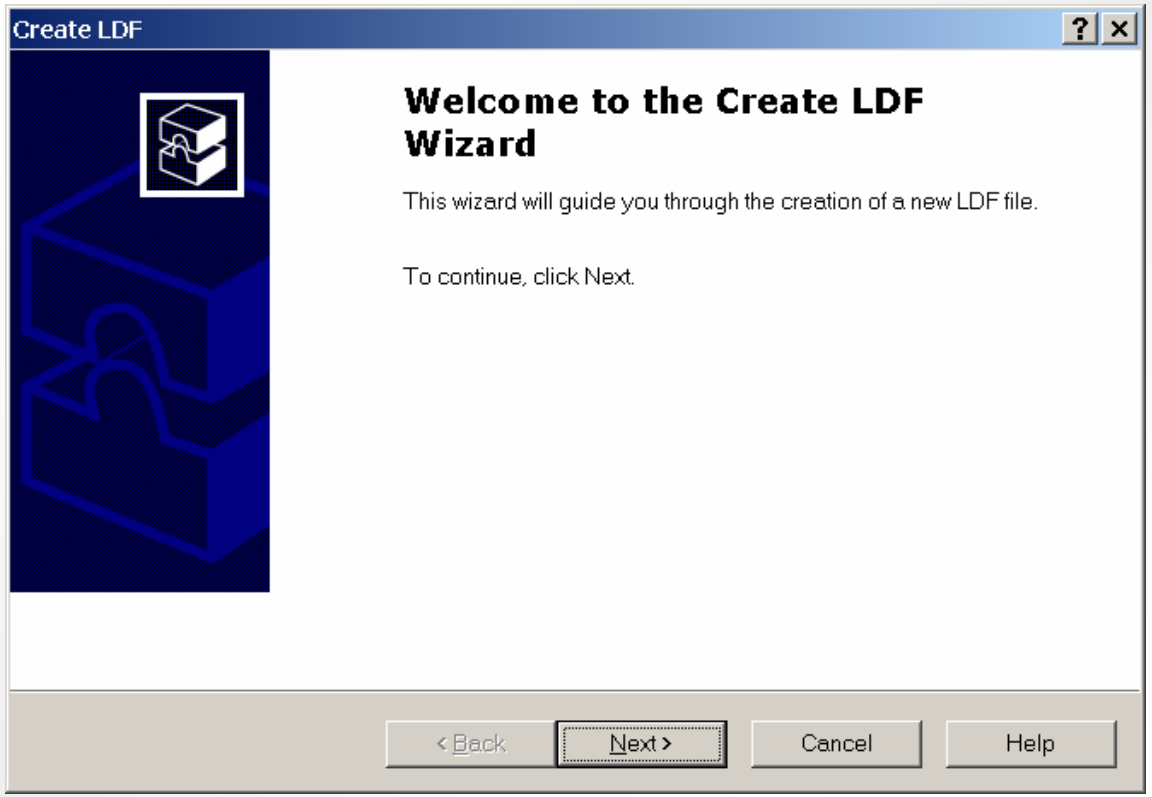
- 'Expert Linker' to graficzne narzedzia ktore pozwalaja na:
  - Uzycie wizard'a do stworzenia plikow LDF
  - Definiowanie mapy pamieci DSP’s
  - Przenoszenie i umieszczanie sekcji objektow w mapie pamieci
  - Tworzenie nakladek graficznych
  - Import plikow LDF
  - Eliminacje graficznego uwydatnionego kodu nieuzywanych objektow
  - Przekroj sekcji objektow w pamieci

Uruchomienie LDF Wizard

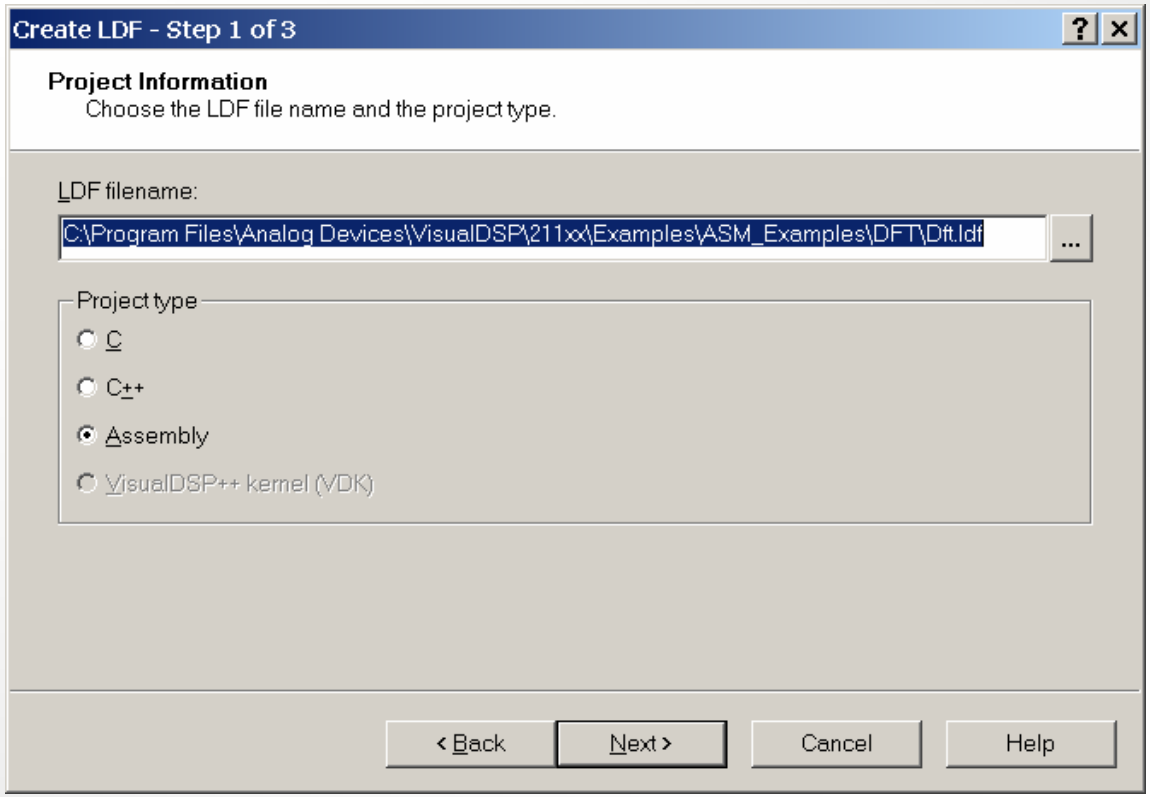
- **Uzycie Expert Linker'a do stworzenia pliku LDF**
- **Uruchomienie kreatora przez wybor:**

  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.**

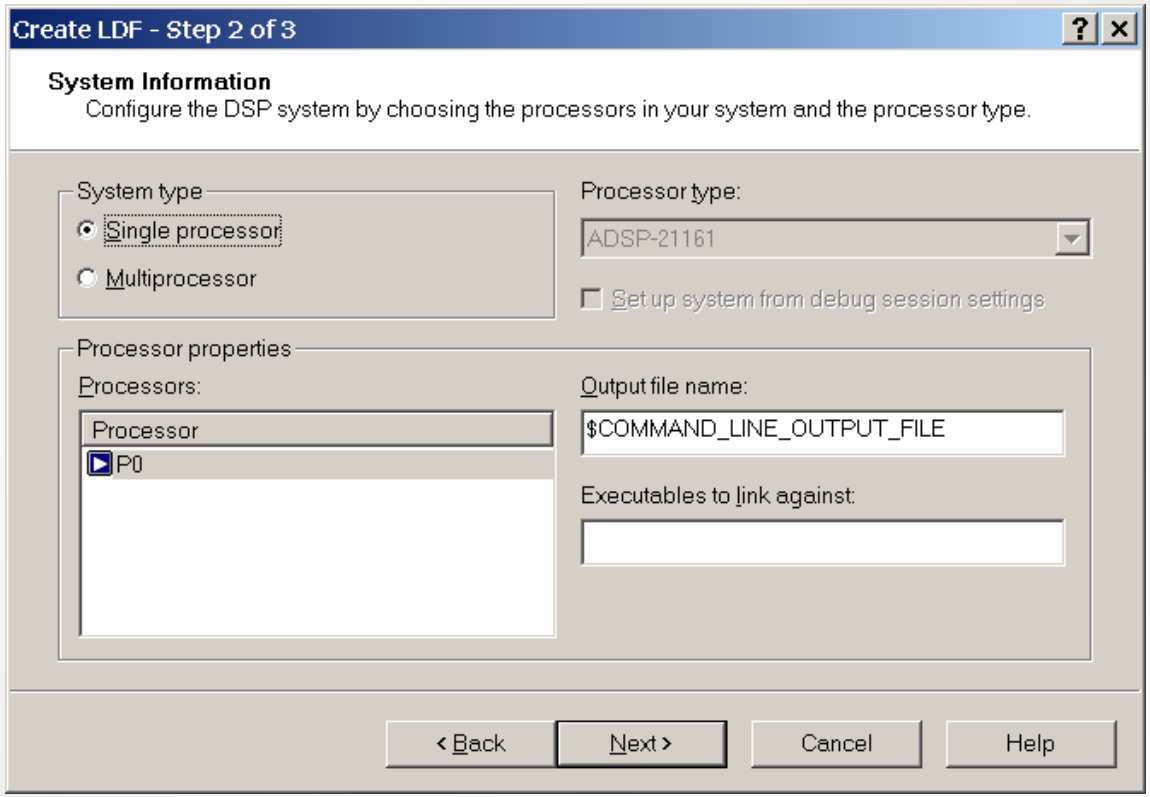
Create LDF Wizard



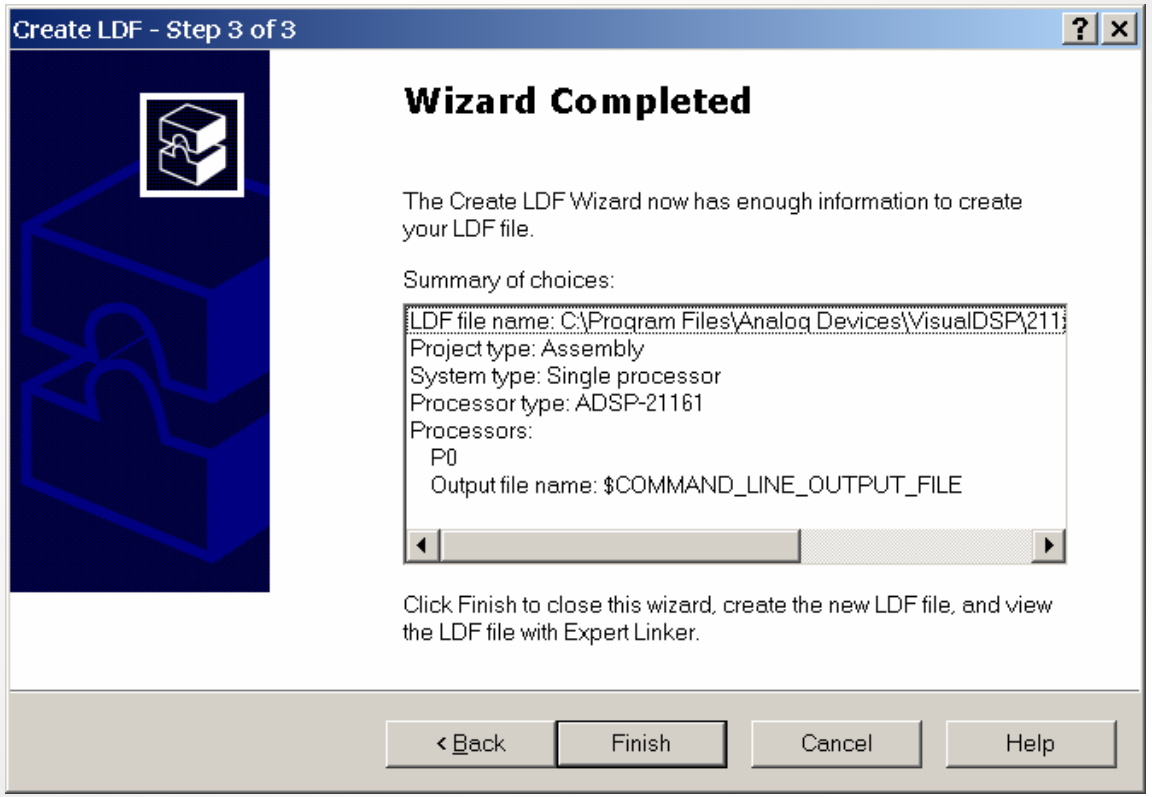
LDF Wizard: Krok 1



LDF Wizard: Krok 2

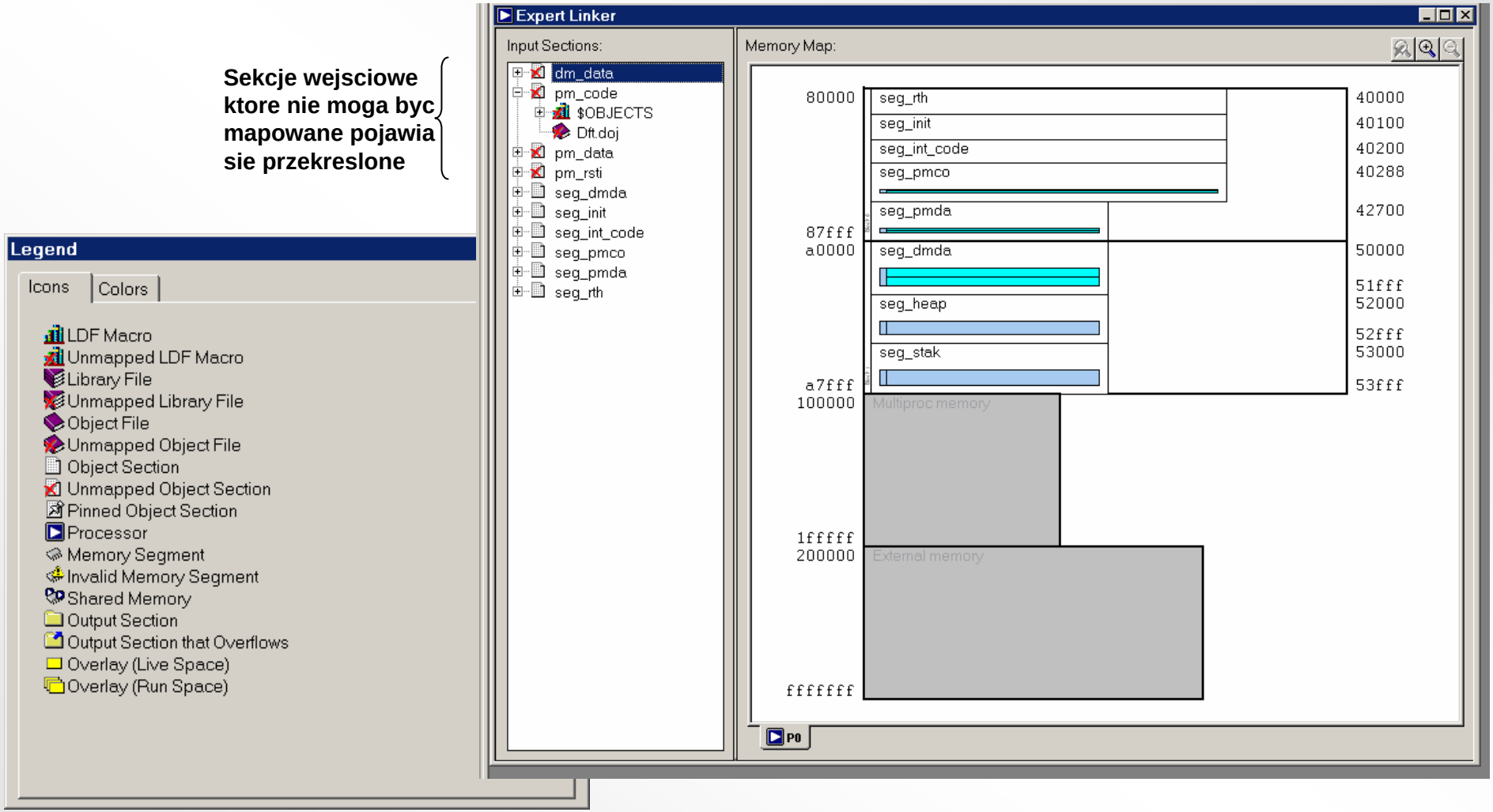


LDF Wizard: Krok 3



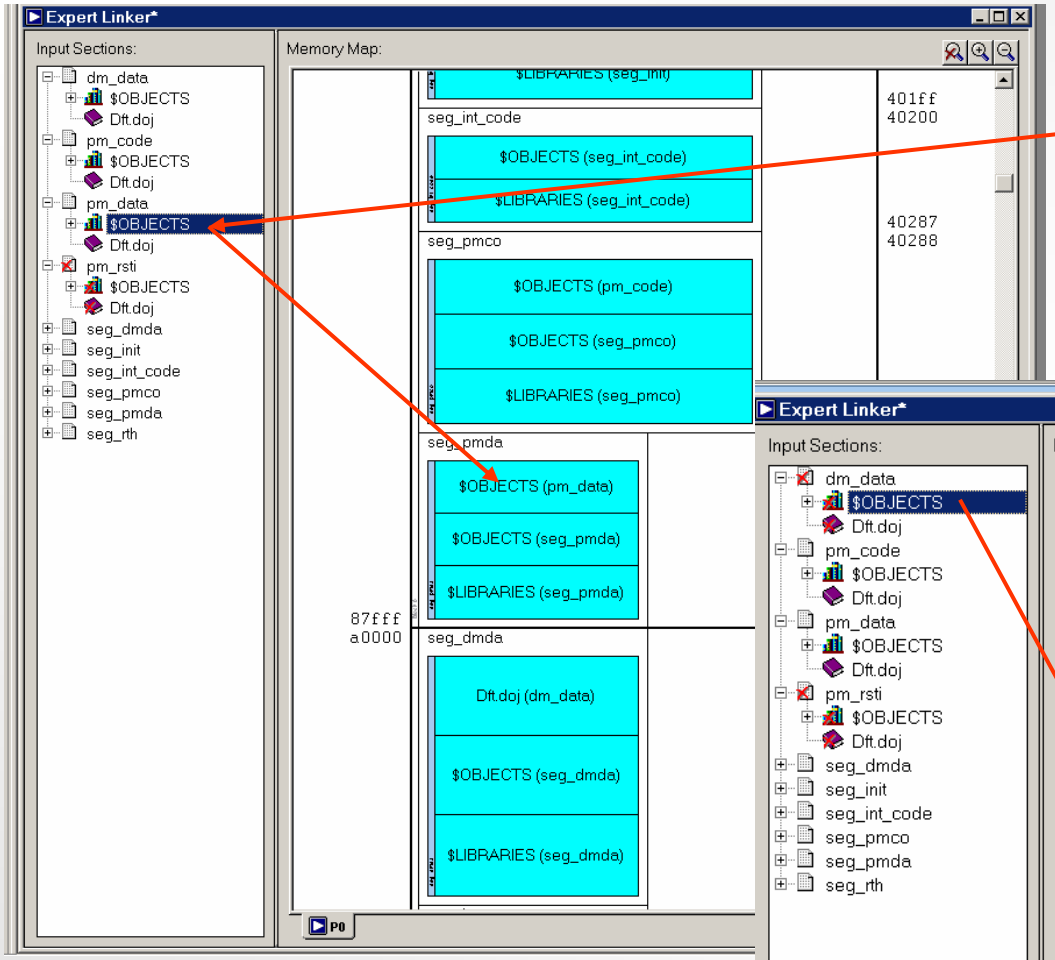
Resultaty Expert Linker'a

Sekcje wejsciowe ktore nie moga byc mapowane pojawia sie przekreslone



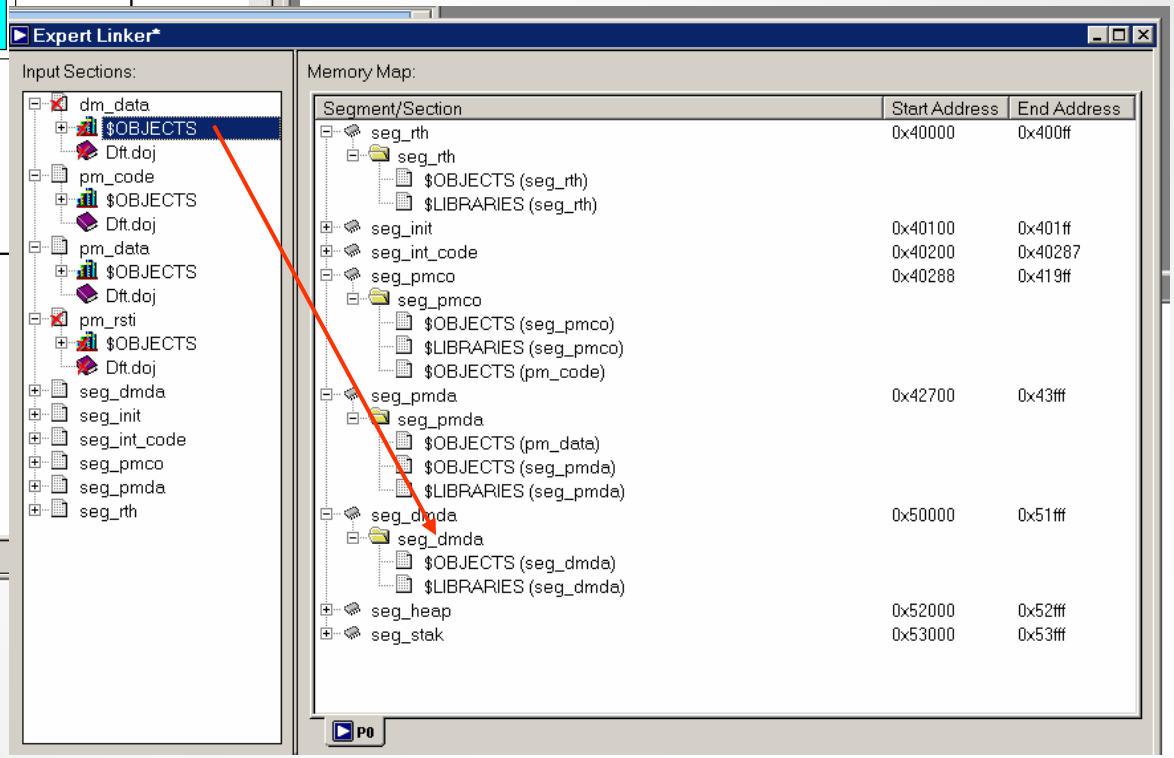
Widok graficznego obrazu pamieci generowanego pliku .ldf.
Podwojne klikniecie na segmencie powieksza go.

Kontrola Mapowania Sekcji



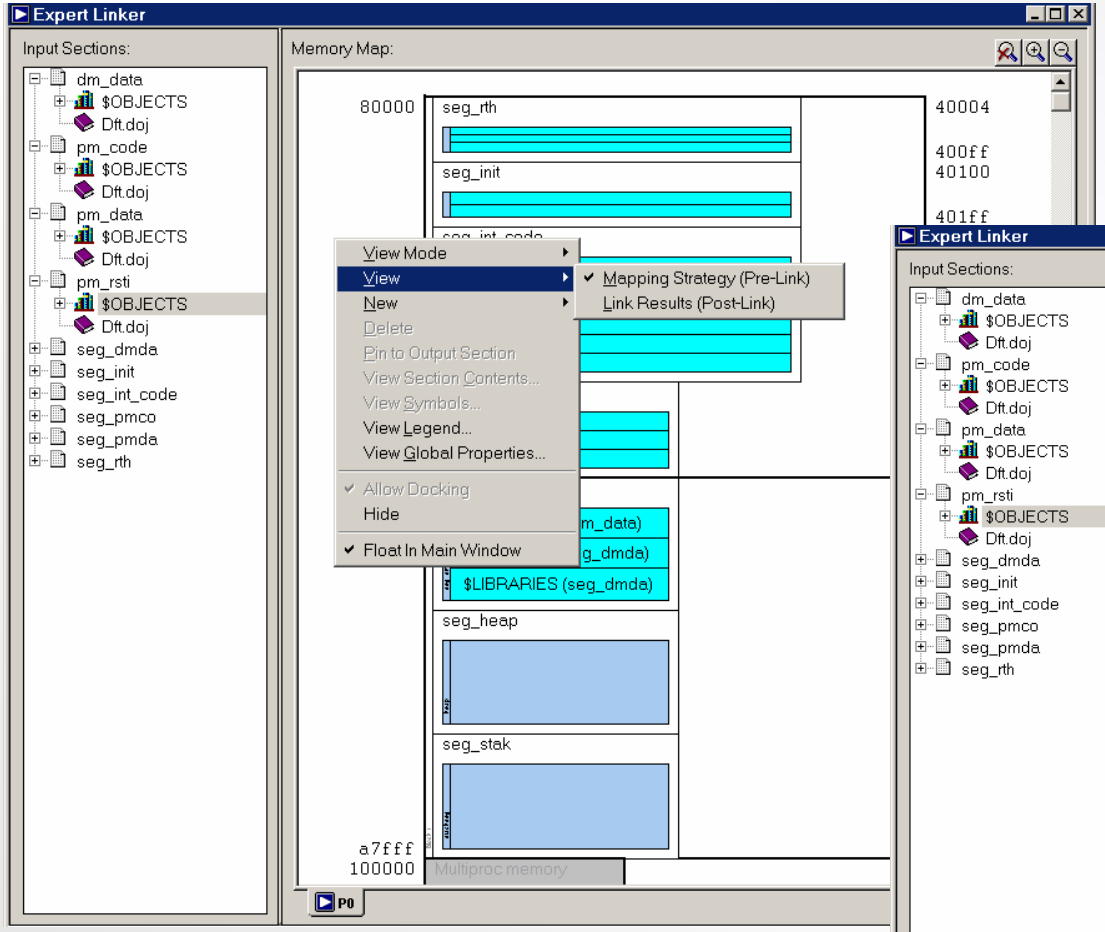
Widok graficzny mapy pamieci

Niemapowane sekcje moga byc ‘mapowane’poprzez przeciaganie do odpowiednich segmentow pamieci.

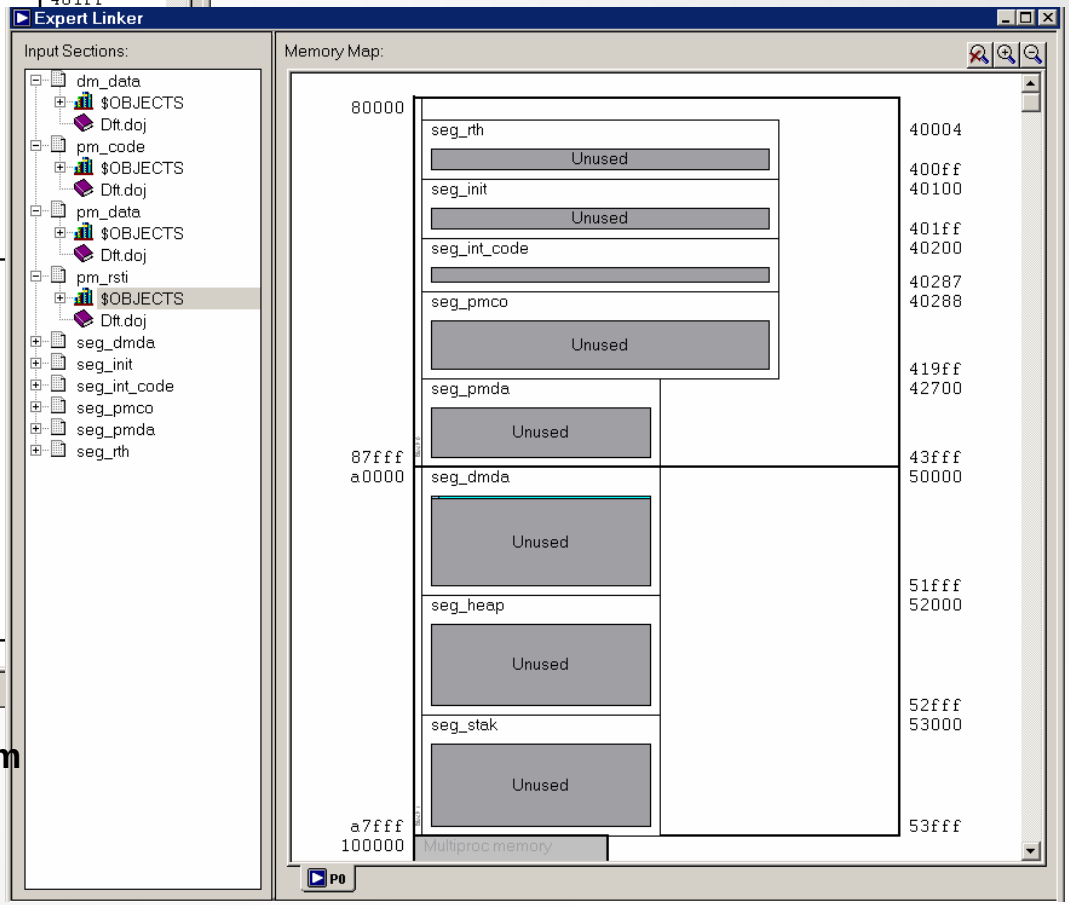


Widok drzewa mapy pamieci

Rezultaty przed i po linkowaniu

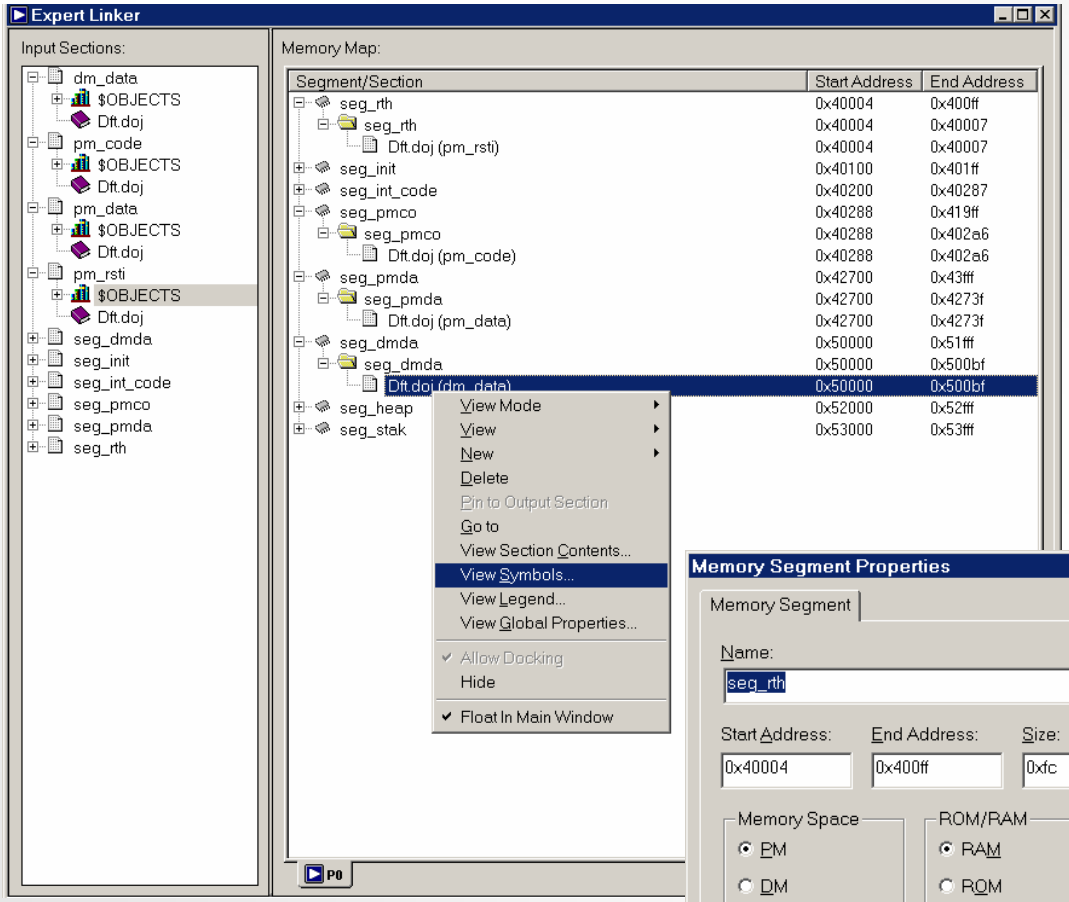


Strategia mapowania przed linkowaniem

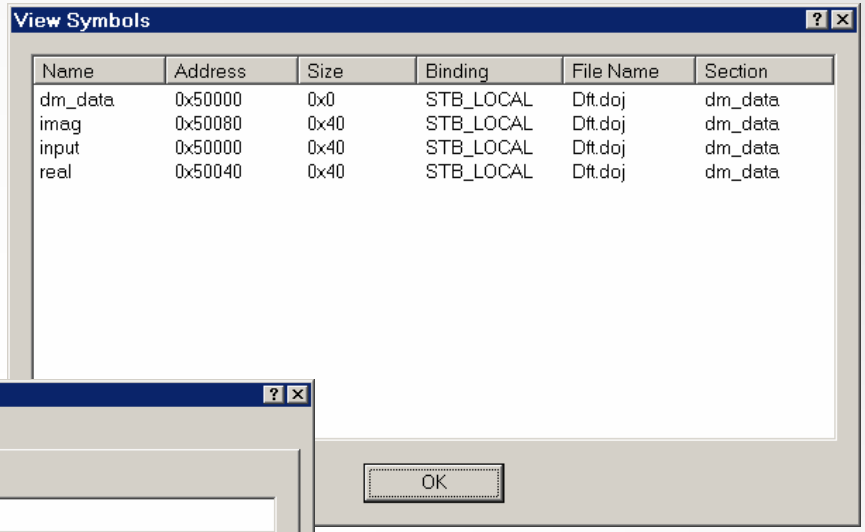


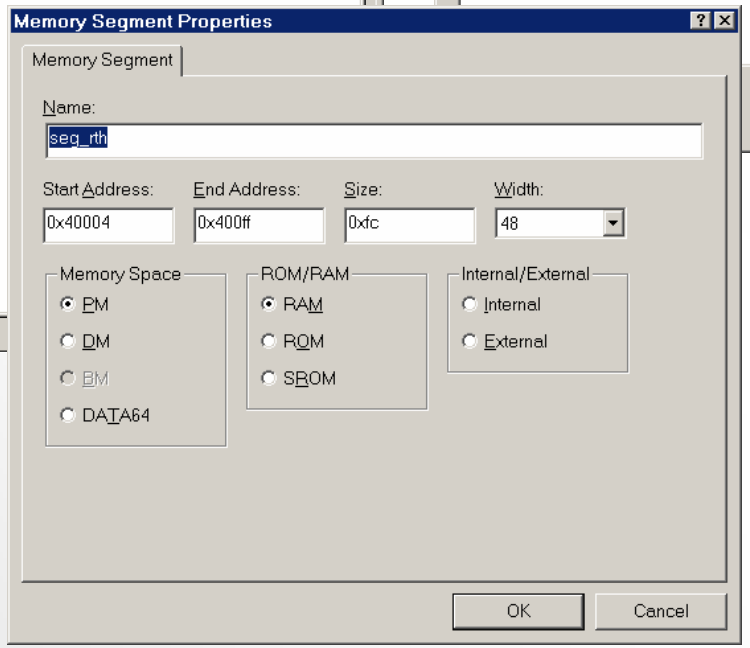
Resultaty po linkowaniu

Wlasciwosci Segmentu LDF



Widok mapy symboli dla specyfi-cznego segmentu pamieci

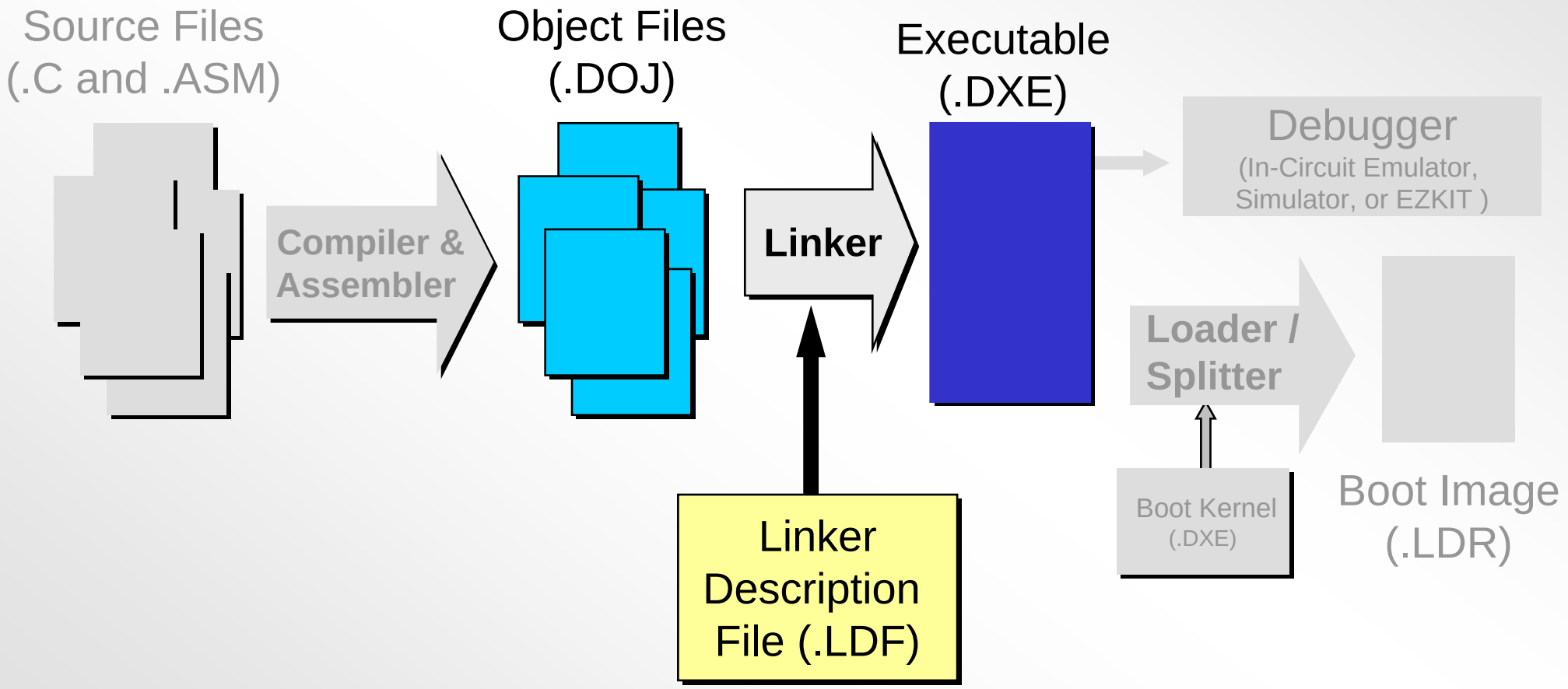




Definicja wlasciwosci dla specificznego segmentu pamieci

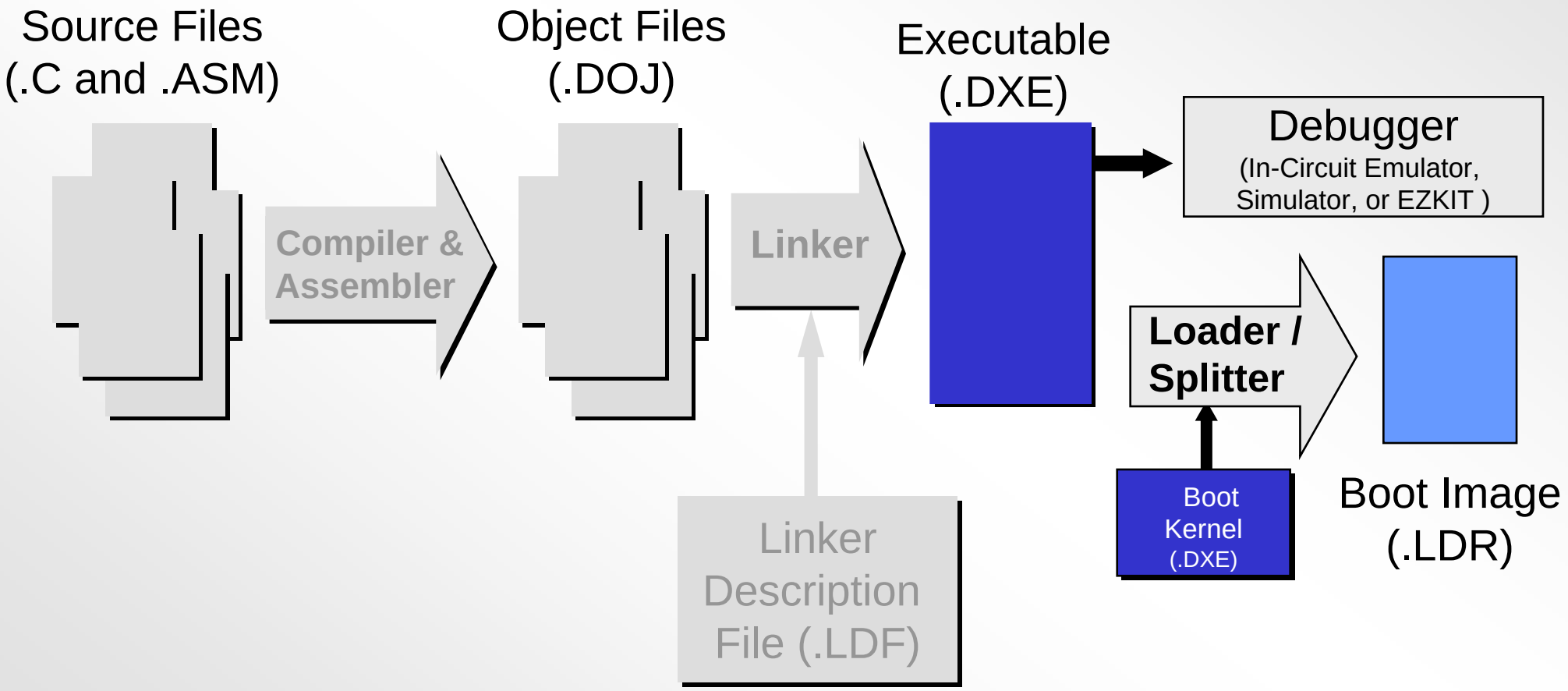
Programowa Budowa Potoku

Krok 2 - Linkowanie



Programowa Budowa Potoku

Krok 3 - Debugowanie



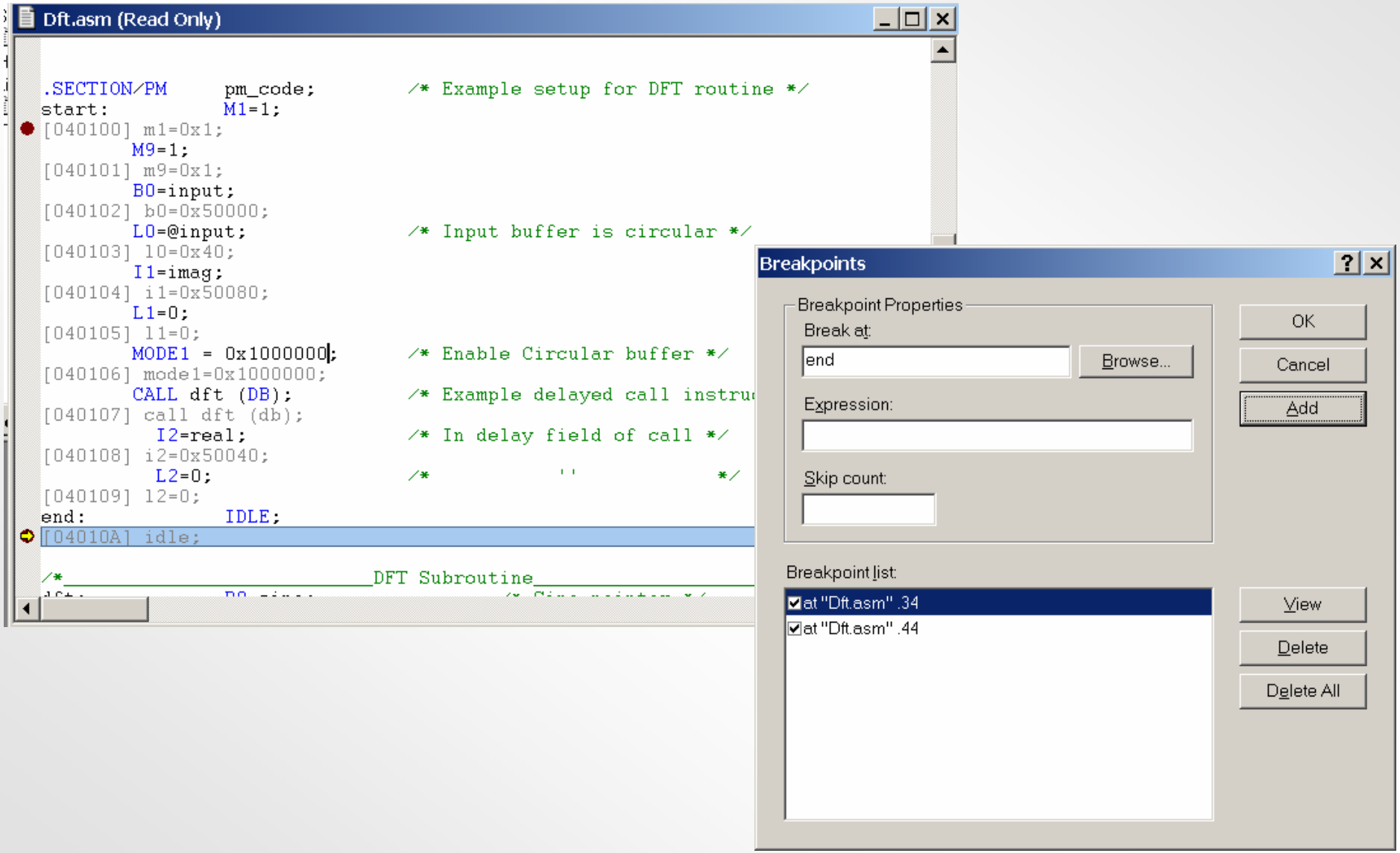
Symulator VisualDSP++

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

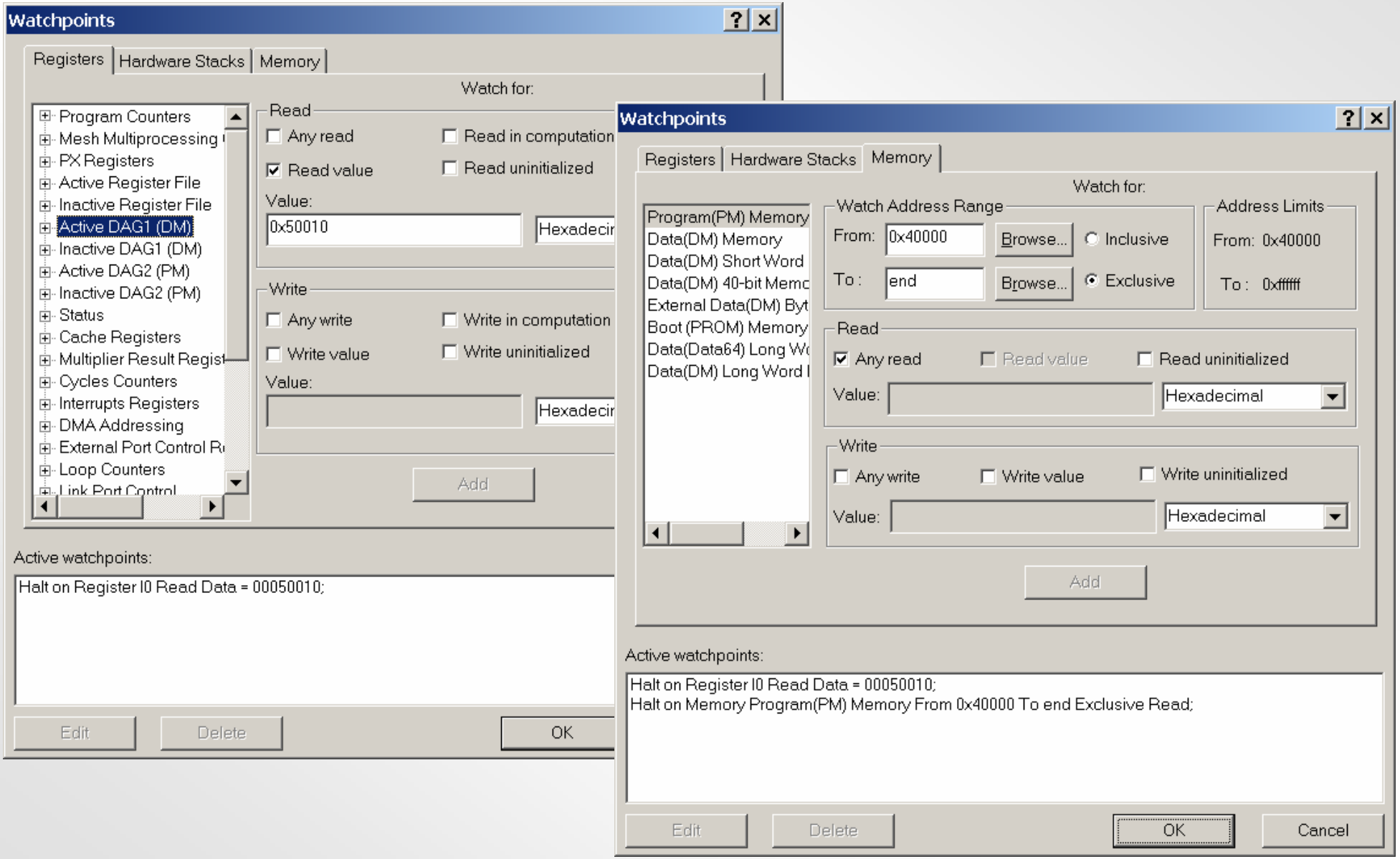
Cechy Symulatora VisualDSP++

- **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



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 rownoleglych
  - Przydziela pliki danych jako zrodlo- cel

VisualDSP++ Okna Debugera

- **Disassembly Window**
  - Widok kodu asm przeasemblowanego
- **Source Window**
  - C, mieszane C/asm
- **Local Window**
  - Pokazuje wszystkie zmienne lokalne wlaczajac bierzace funkcje
- **Expressions Window**
  - Dowolne wyrazenia w jezyku “C"
  - Nazwy rejestrow zaczynaja sie od $ (przyklad $R12)
- **Trace Window (tylko symulacja)**
  - Wyswietla historie wykonania instrukcji
  - Specyficzna glebokosc sciezek
- **Profile Window**
  - Licznik cykli i procentowe zuzycie czasu wykonanujac specyficzne zakresy zadresow
- **Plot**
  - Mozliwosc zwiekszania wykresow

Visual DSP++ Okna Debugera

- **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**

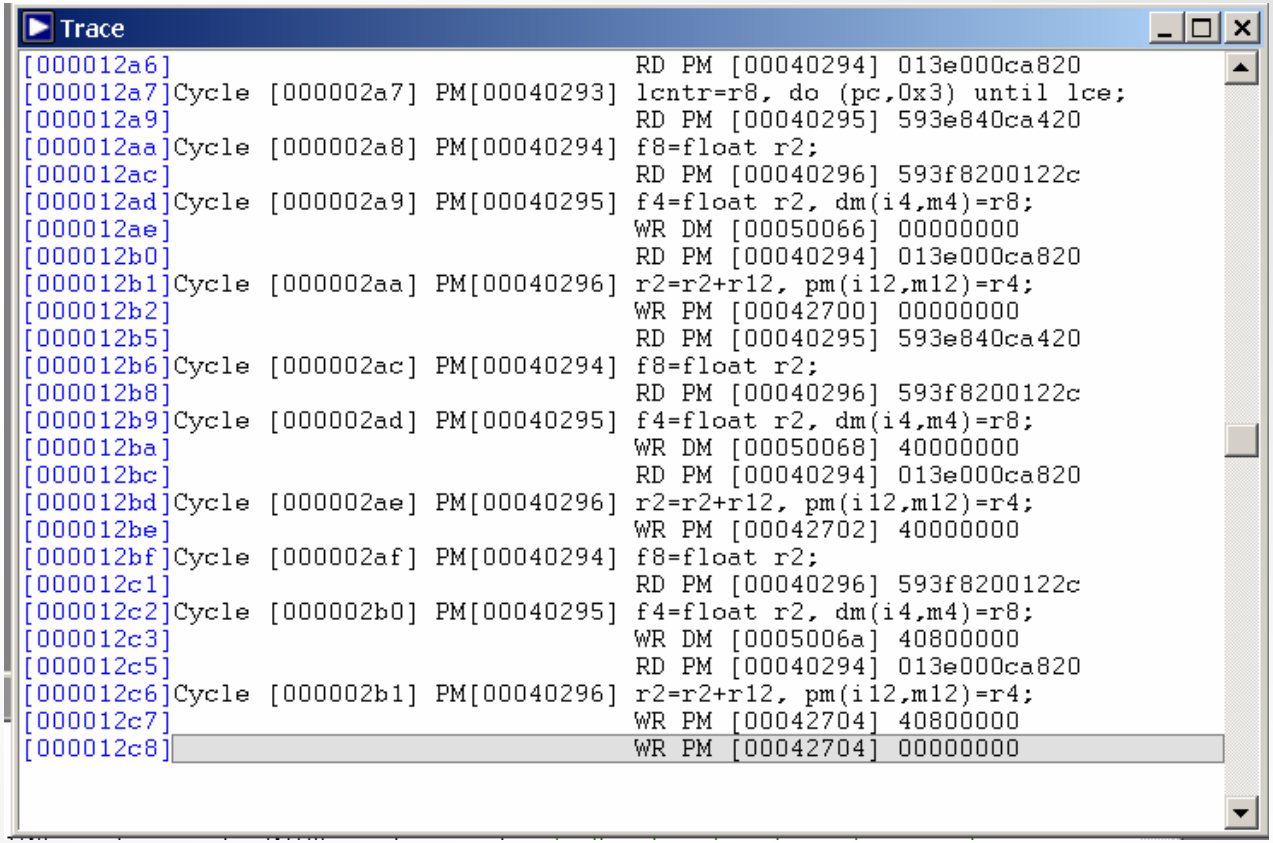
Zwolnienie/Zapelnienie pamieci

- **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 domyslny 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 wartoci cykli DSP.
  - Adres bierzacej instrukcji
  - Dostep do pamieci w formie wyrazenia (Mem=Value lub Value=Mem)
- **Ustawienia**
  - *Set Depth*: Liczba sciezek. Bufory to dlugich sciezkach spowalniaja debuger.
  - *Clear*: Czysci bierzaca sciezke
  - *Enable*: Zezwala lub zabrania na dostep do sciezki.



Profiler

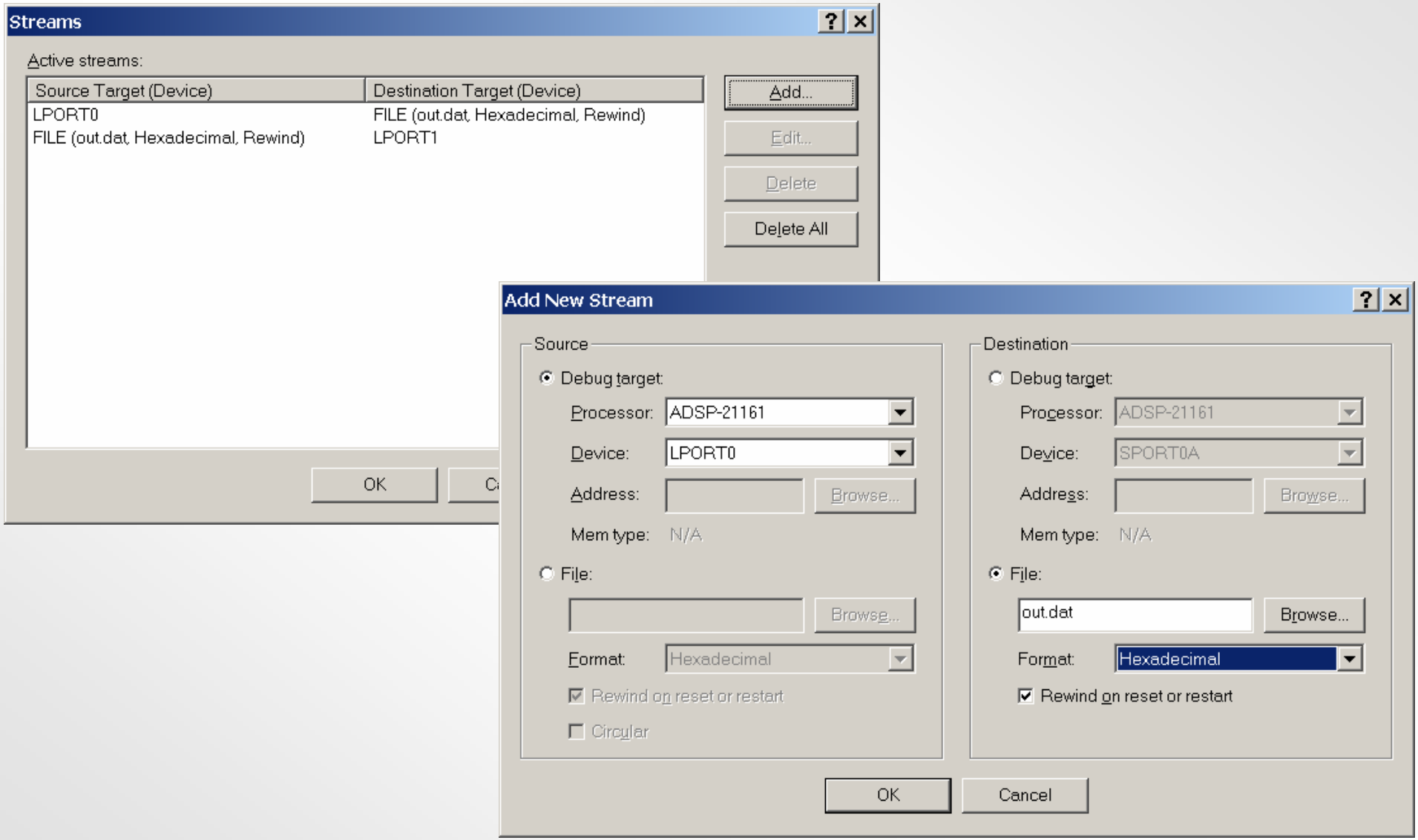
Linear Profiling Results

Histogram	%	Execu...
	29.36%	main()
	1.70%	PC[0...
	1.13%	PC[0...
	1.13%	PC[0...
	1.13%	PC[0...
	1.11%	PC[0...
	1.11%	PC[0...
	1.11%	PC[0...
	0.73%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...
	0.57%	PC[0...

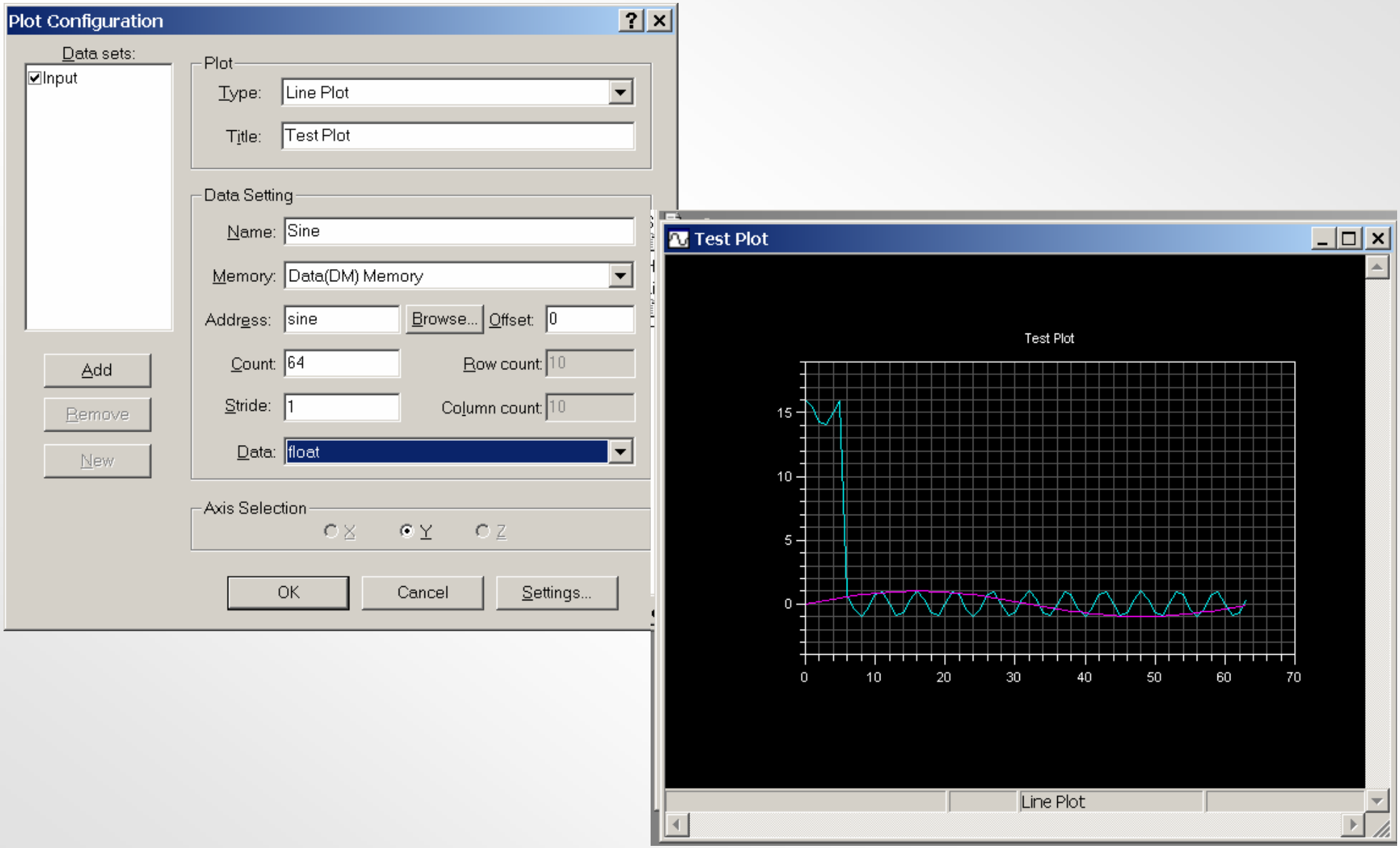
%	Line...	C:\Program Files\Analog Devices\VisualDSP\211xx\Examples\C_Exampl...
	19	`float dm output_SIMD=0;`
	20	
	21	`int count_start();`
	22	`int count_end(int);`
	23	`volatile int time_temp, SISD_cycle_count, SIMD_cycle_count; /* f...`
	24	
0.01%	25	`void main(void)`
	26	`{`
	27	`int i;`
	28	
	29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

Total Samples: 8823 Elapsed Time: 00:00:01 Enabled

Strumienie


Streams
Active streams:
Source Target (Device)
Destination Target (Device)
LPORT0
FILE (out.dat, Hexadecimal, Rewind)
FILE (out.dat, Hexadecimal, Rewind)
LPORT1
Add...
Edit...
Delete
Delete All
OK
Add New Stream
Source
Debug target:
Processor: ADSP-21161
Device: LPORT0
Address:
Browse...
Mem type: N/A
File:
Browse...
Format: Hexadecimal
Rewind on reset or restart
Circular
Destination
Debug target:
Processor: ADSP-21161
Device: SPORT0A
Address:
Browse...
Mem type: N/A
File:
out.dat
Browse...
Format: Hexadecimal
Rewind on reset or restart
OK
Cancel

Wykresy


Plot Configuration
Data sets:
Input
Add
Remove
New
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...
Test Plot
Test Plot
15
10
5
0
0
10
20
30
40
50
60
70
Line Plot

Run to Main & STDIO

- **Run To Main**
  - Pozwala uzytkownikowi kontrolowac (lub nie) debuger, rozpoczecie wykonynania w naglowku 'run time'.

- **STDIO**
  - Pelne wsparcie STDIO. Uzycie **`printf()`** i **`scanf()`** na dostep do plikow systemowych hosta.