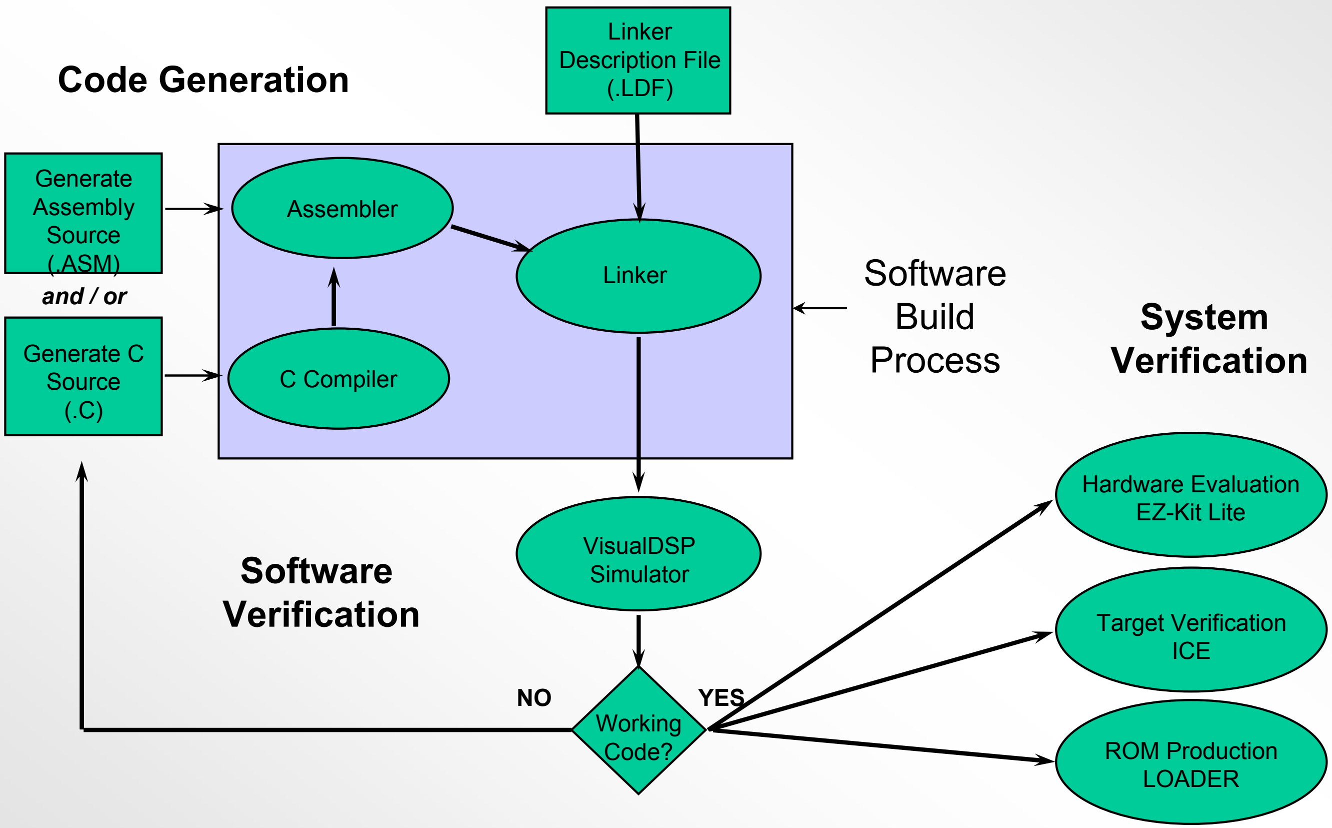
Section 8

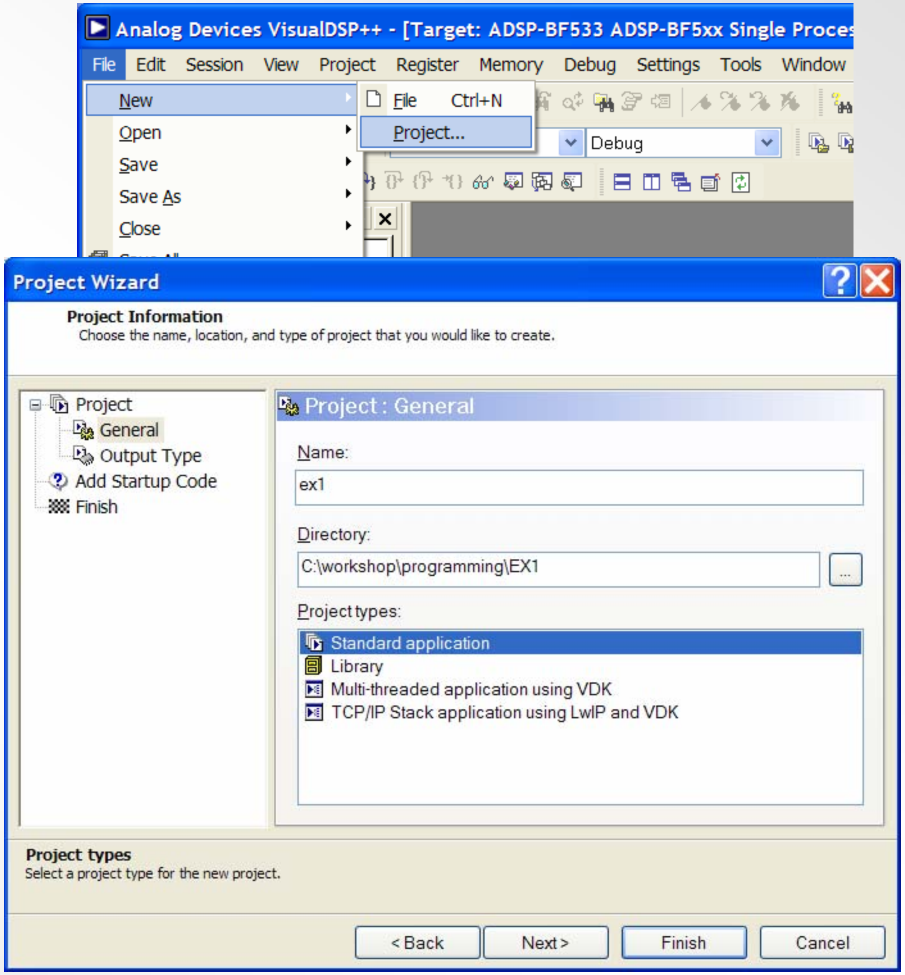
Programming

Software Development Flow



Project Development

- **Create a project**
  - All development in VisualDSP++ occurs within a project.
  - The project file (.DPJ) stores your program's build information: source files list and development tools option settings
  - A project group file (.DPG) contains a list of projects that make up an application (eg ADSP-BF561 dual core application)



Project Information
Choose the name, location, and type of project that you would like to create.

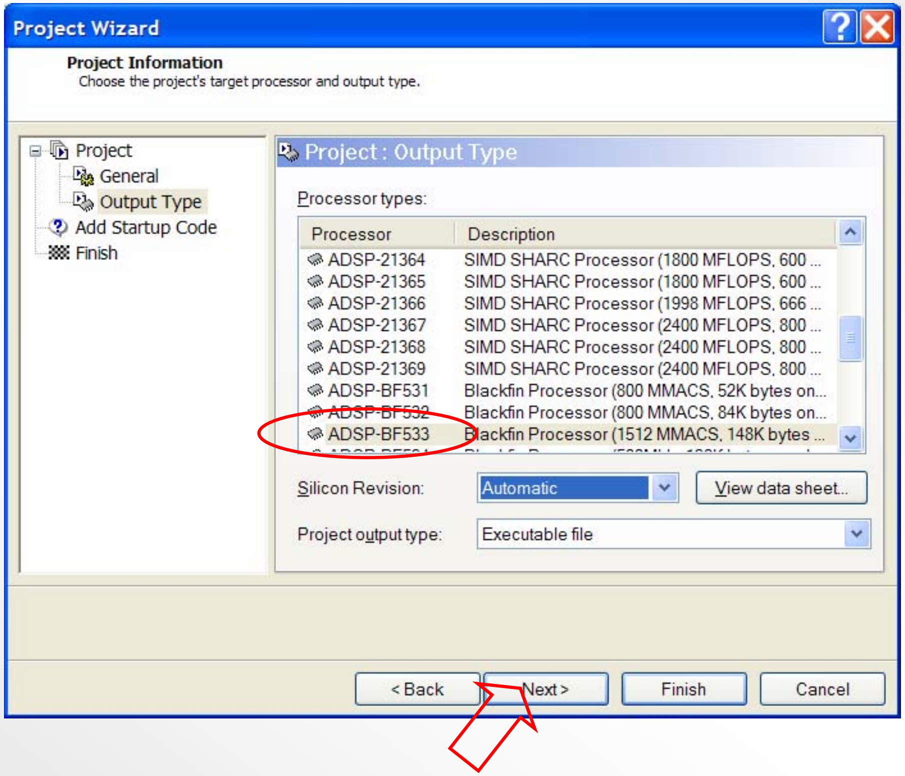
Name: ex1

Directory: C:\workshop\programming\EX1

Project types:
- Standard application
- Library
- Multi-threaded application using VDK
- TCP/IP Stack application using LwIP and VDK

Project types
Select a project type for the new project.

Select Target Processor



Project Information

Choose the project's target processor and output type.

Project : Output Type

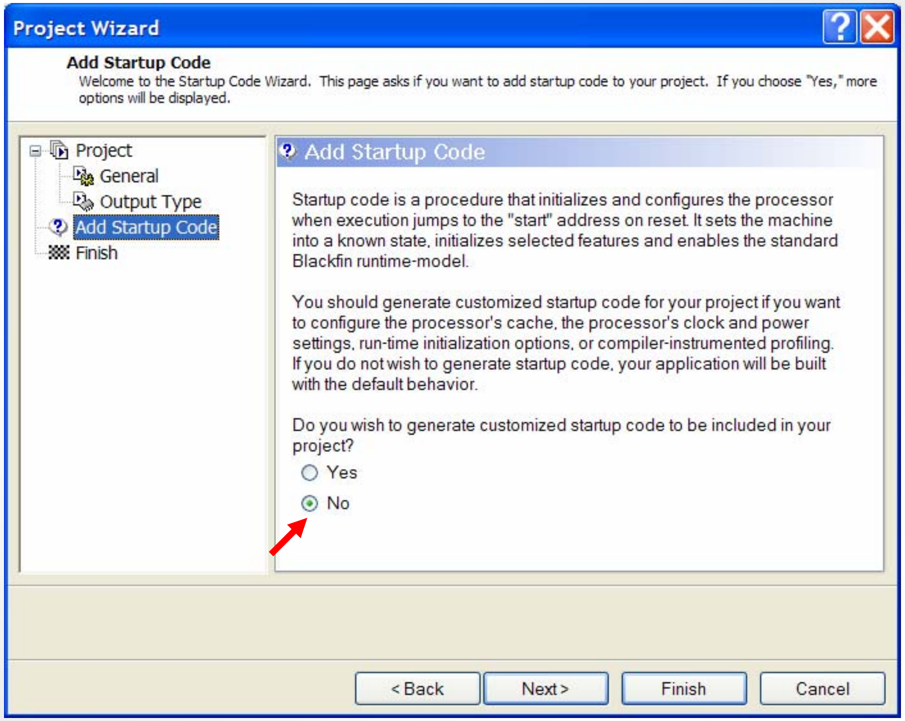
Processor types:

Processor	Description
ADSP-21364	SIMD SHARC Processor (1800 MFLOPS, 600 ...
ADSP-21365	SIMD SHARC Processor (1800 MFLOPS, 600 ...
ADSP-21366	SIMD SHARC Processor (1998 MFLOPS, 666 ...
ADSP-21367	SIMD SHARC Processor (2400 MFLOPS, 800 ...
ADSP-21368	SIMD SHARC Processor (2400 MFLOPS, 800 ...
ADSP-21369	SIMD SHARC Processor (2400 MFLOPS, 800 ...
ADSP-BF531	Blackfin Processor (800 MMACS, 52K bytes on...
ADSP-BF532	Blackfin Processor (800 MMACS, 84K bytes on...
ADSP-BF533	Blackfin Processor (1512 MMACS, 148K bytes ...

Silicon Revision: Automatic

Project output type: Executable file

Startup Code



Add Startup Code

Welcome to the Startup Code Wizard. This page asks if you want to add startup code to your project. If you choose "Yes," more options will be displayed.

Add Startup Code

Startup code is a procedure that initializes and configures the processor when execution jumps to the "start" address on reset. It sets the machine into a known state, initializes selected features and enables the standard Blackfin runtime-model.

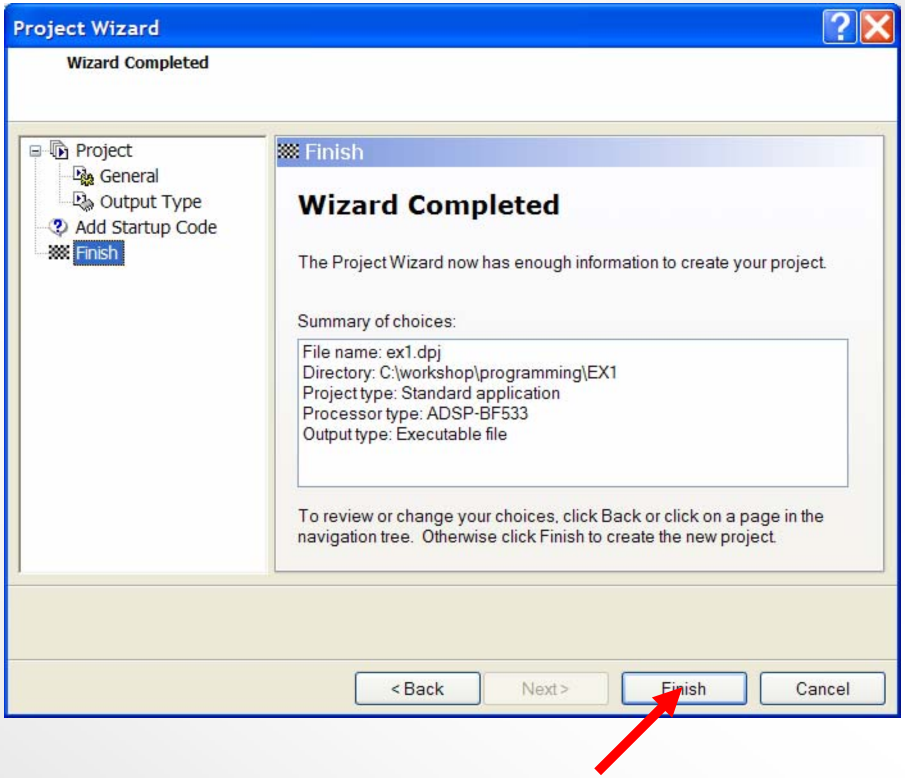
You should generate customized startup code for your project if you want to configure the processor's cache, the processor's clock and power settings, run-time initialization options, or compiler-instrumented profiling. If you do not wish to generate startup code, your application will be built with the default behavior.

Do you wish to generate customized startup code to be included in your project?

○ Yes

● No

Finish



Wizard Completed

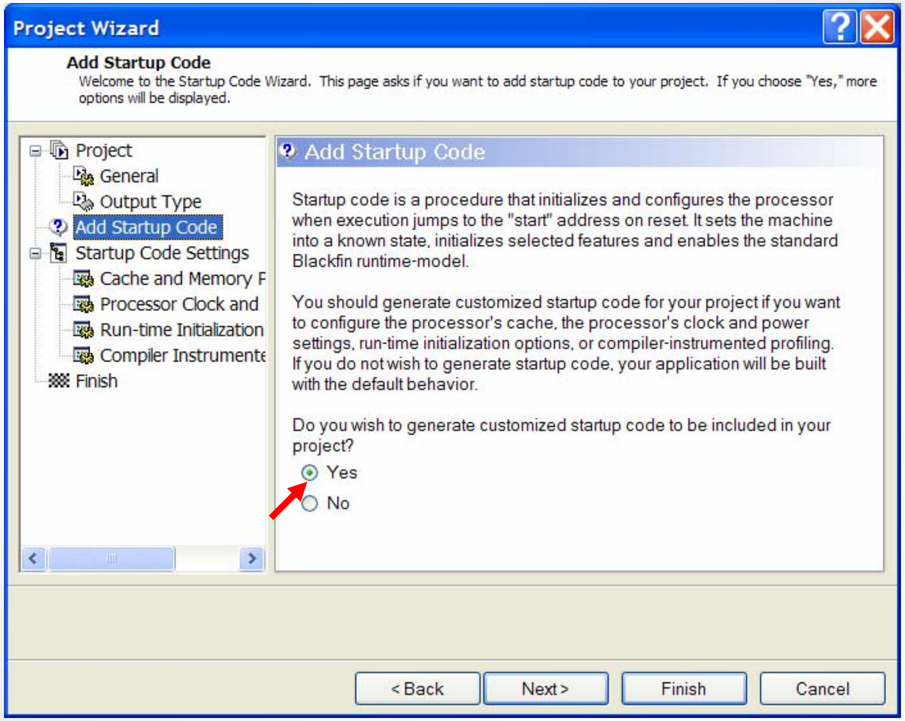
The Project Wizard now has enough information to create your project.

Summary of choices:

File name: ex1.dpj
Directory: C:\workshop\programming\EX1
Project type: Standard application
Processor type: ADSP-BF533
Output type: Executable file

To review or change your choices, click Back or click on a page in the navigation tree. Otherwise click Finish to create the new project.

C/C++ Project - Startup Code



Add Startup Code

Welcome to the Startup Code Wizard. This page asks if you want to add startup code to your project. If you choose "Yes," more options will be displayed.

Add Startup Code

Startup code is a procedure that initializes and configures the processor when execution jumps to the "start" address on reset. It sets the machine into a known state, initializes selected features and enables the standard Blackfin runtime-model.

You should generate customized startup code for your project if you want to configure the processor's cache, the processor's clock and power settings, run-time initialization options, or compiler-instrumented profiling. If you do not wish to generate startup code, your application will be built with the default behavior.

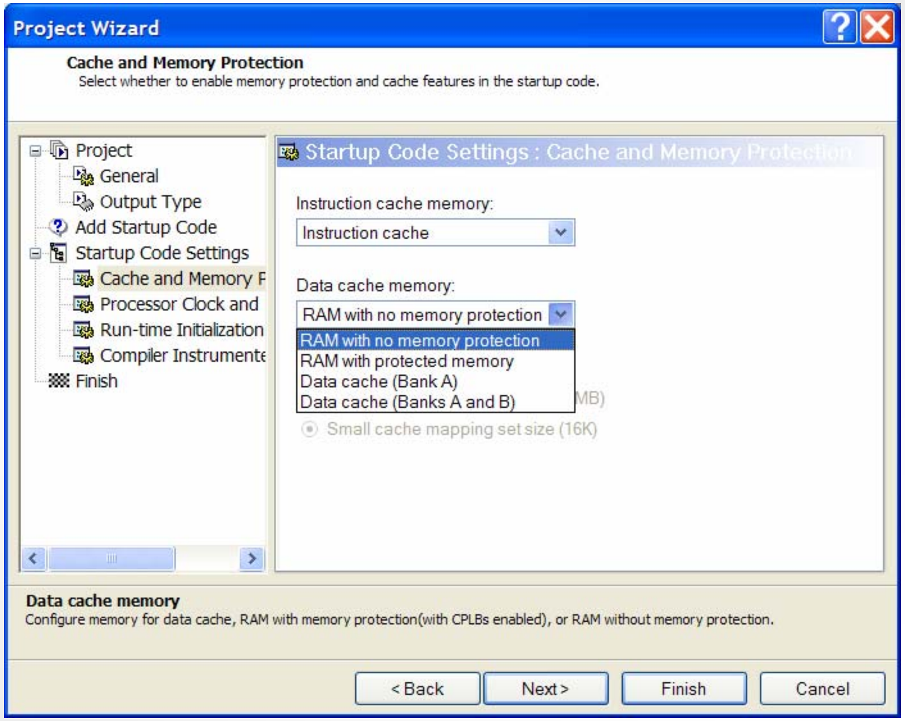
Do you wish to generate customized startup code to be included in your project?

● Yes

○ No

For pure assembly code applications, select 'NO' option. For C/C++ applications, select 'YES' to customize a run time header for you application.

Setup of Configurable Memory Blocks in L1



Project Wizard

Cache and Memory Protection

Select whether to enable memory protection and cache features in the startup code.

Startup Code Settings : Cache and Memory Protection

Instruction cache memory: Instruction cache

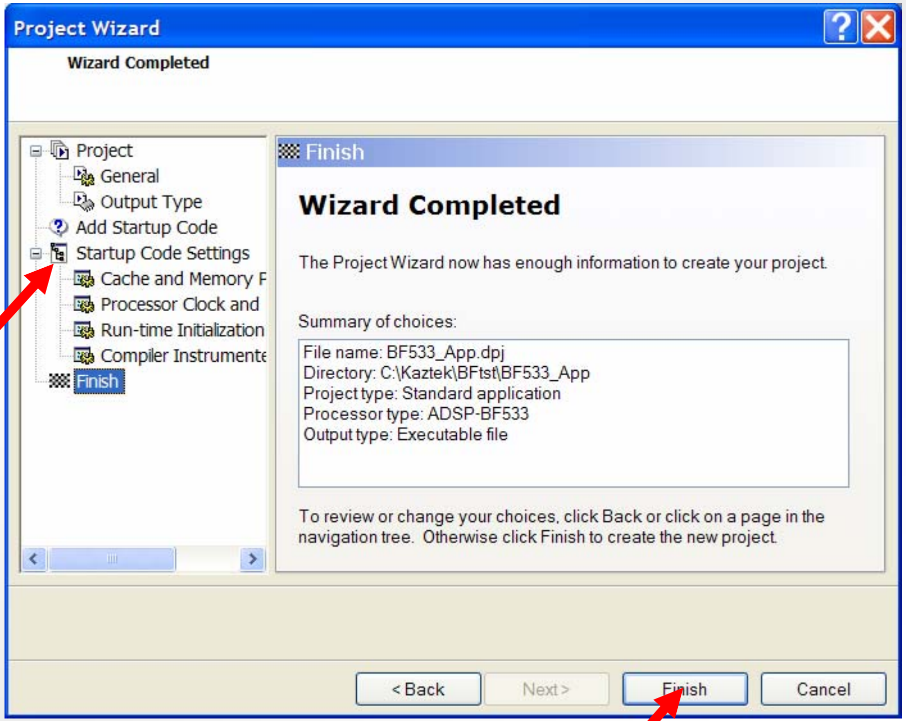
Data cache memory: RAM with no memory protection

RAM with no memory protection
RAM with protected memory
Data cache (Bank A)
Data cache (Banks A and B)
MB)
○ Small cache mapping set size (16K)

Data cache memory

Configure memory for data cache, RAM with memory protection(with CPLBs enabled), or RAM without memory protection.

Wizard is Done



Wizard Completed

The Project Wizard now has enough information to create your project.

Summary of choices:

File name: BF533_App.dpj
Directory: C:\Kaztek\BFtst\BF533_App
Project type: Standard application
Processor type: ADSP-BF533
Output type: Executable file

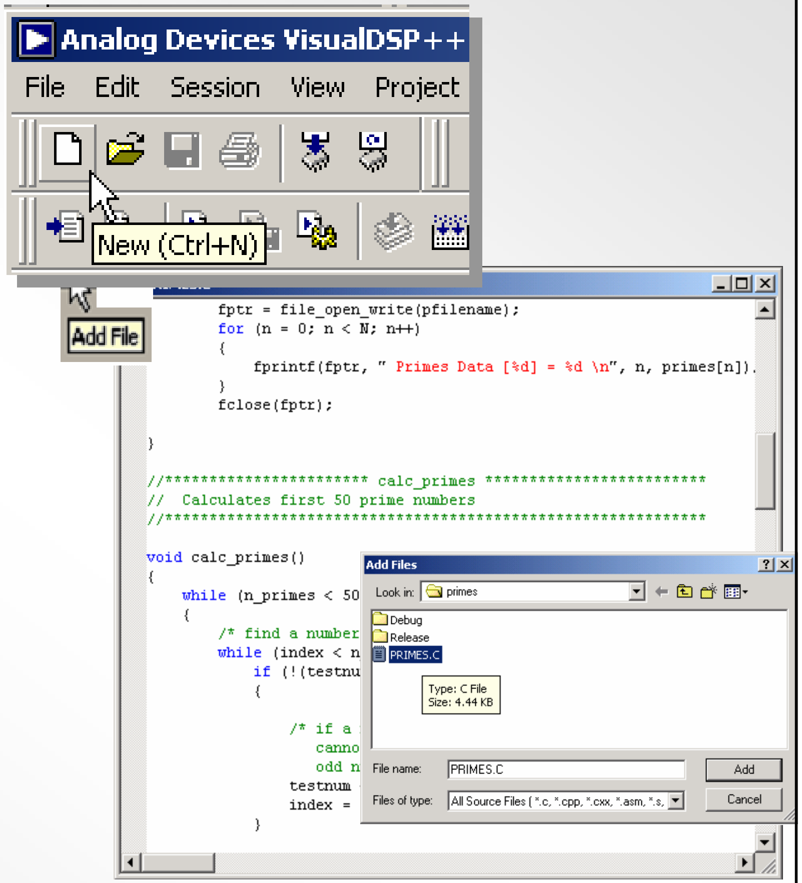
To review or change your choices, click Back or click on a page in the navigation tree. Otherwise click Finish to create the new project.

At a later time, the CRT Header can be modified by selecting Project Options/Startup Code Settings and making changes.

When finished, the wizard creates a customized C Run Time Header.

Project Development Steps

- **Create project source files**
  - A project normally contains one or more C, C++, or assembly language source files.
  - After you create a project and define its target processor, you add new or existing files to the project by importing or writing them.
  - The VisualDSP++ Editor lets you create new files or edit any existing text file



```
        fptr = file_open_write(pfilename);
        for (n = 0; n < N; n++)
        {
            fprintf(fptr, " Primes Data [%d] = %d \n", n, primes[n]).
        }
        fclose(fptr);

}

//************************ calc_primes ************************
//  Calculates first 50 prime numbers
//**************************************************************

void calc_primes()
{
    while (n_primes < 50
    {
        /* find a number
        while (index < n
            if (!(testnu
            {

                /* if a
                   canno
                   odd n
                testnum
                index =
            }
```

Add Files

Look in: primes

Debug
Release
PRIMES.C

Type: C File
Size: 4.44 KB

File name: PRIMES.C

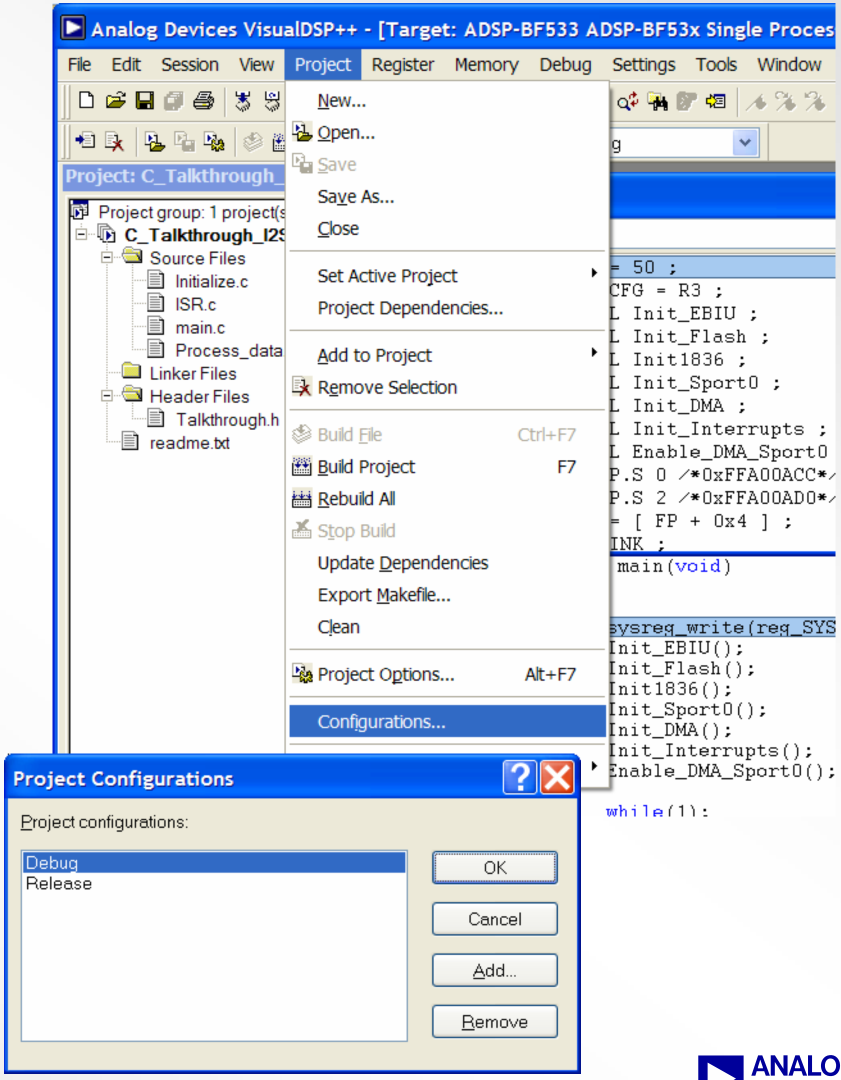
Files of type: All Source Files (*.c, *.cpp, *.cxx, *.asm, *.s,

Add

Cancel

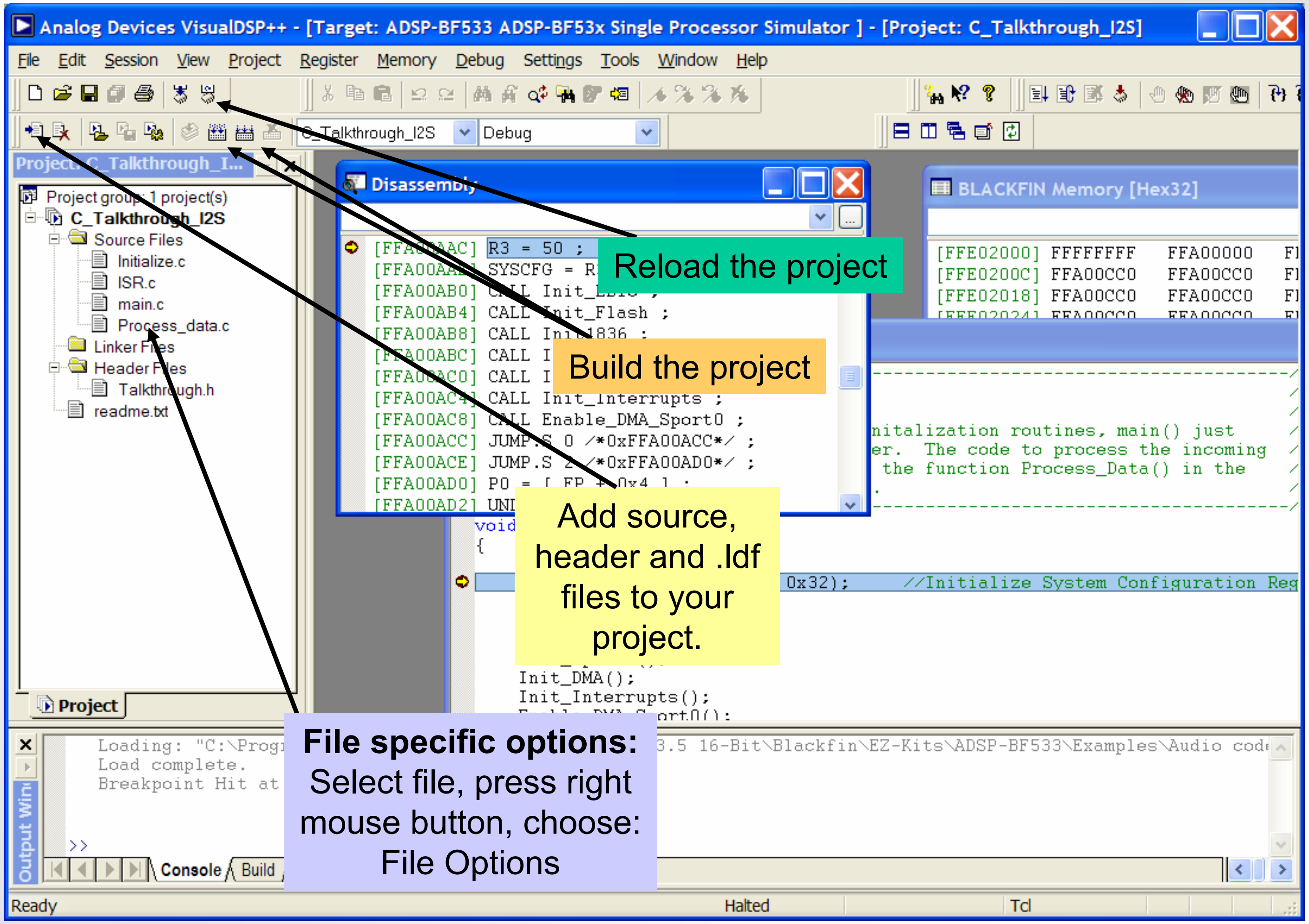
Project Development Steps

- **Define project build options**
  - A project's configuration setting controls its build. By default, the choices are Debug or Release.
  - Debug
    - Typically has more debug options set for the tools.
    - compiler generates debug information to allow source level debug.
  - Release
    - Typically has fewer or no debug options set for the tools
    - builds are usually optimised for performance



```
= 50 ;
CFG = R3 ;
L Init_EBIU ;
L Init_Flash ;
L Init1836 ;
L Init_Sport0 ;
L Init_DMA ;
L Init_Interrupts ;
L Enable_DMA_Sport0
P.S 0 /*0xFFA00ACC*/
P.S 2 /*0xFFA00AD0*/
= [ FP + 0x4 ] ;
INK ;
 main(void)

sysreg_write(reg_SYS
Init_EBIU();
Init_Flash();
Init1836();
Init_Sport0();
Init_DMA();
Init_Interrupts();
Enable_DMA_Sport0();

while(1);
```

VisualDSP++ Menu


Analog Devices VisualDSP++ - [Target: ADSP-BF533 ADSP-BF53x Single Processor Simulator] - [Project: C_Talkthrough_I2S]
File Edit Session View Project Register Memory Debug Settings Tools Window Help
C_Talkthrough_I2S
Debug
Project: C_Talkthrough_I...
Project group: 1 project(s)
C_Talkthrough_I2S
Source Files
Initialize.c
ISR.c
main.c
Process_data.c
Linker Files
Header Files
Talkthrough.h
readme.txt
Project
Disassembly
BLACKFIN Memory [Hex32]
Output Window
Console
Build
Ready
Halted
Tcl

```
[FFA00AAC] R3 = 50 ;
[FFA00AA   SYSCFG = R
[FFA00AB0] CALL Init_EBIU ;
[FFA00AB4] CALL Init_Flash ;
[FFA00AB8] CALL Init1836 ;
[FFA00ABC] CALL I
[FFA00AC0] CALL I
[FFA00AC4] CALL Init_Interrupts ;
[FFA00AC8] CALL Enable_DMA_Sport0 ;
[FFA00ACC] JUMP.S 0 /*0xFFA00ACC*/ ;
[FFA00ACE] JUMP.S 2 /*0xFFA00AD0*/ ;
[FFA00AD0] P0 = [ FP + 0x4 ] ;
[FFA00AD2] UNI
```

```
[FFE02000] FFFFFFFF   FFA00000   FI
[FFE0200C] FFA00CC0   FFA00CC0   FI
[FFE02018] FFA00CC0   FFA00CC0   FI
```

```
--------------------------------------------/
                                            /
                                            /
nitalization routines, main() just          /
er.  The code to process the incoming       /
 the function Process_Data() in the         /
.                                           /
--------------------------------------------/
void
{
                0x32);     //Initialize System Configuration Reg

    Init_DMA();
    Init_Interrupts();
```

```
Loading: "C:\Progr                3.5 16-Bit\Blackfin\EZ-Kits\ADSP-BF533\Examples\Audio code
Load complete.
Breakpoint Hit at
>>
```

Reload the project

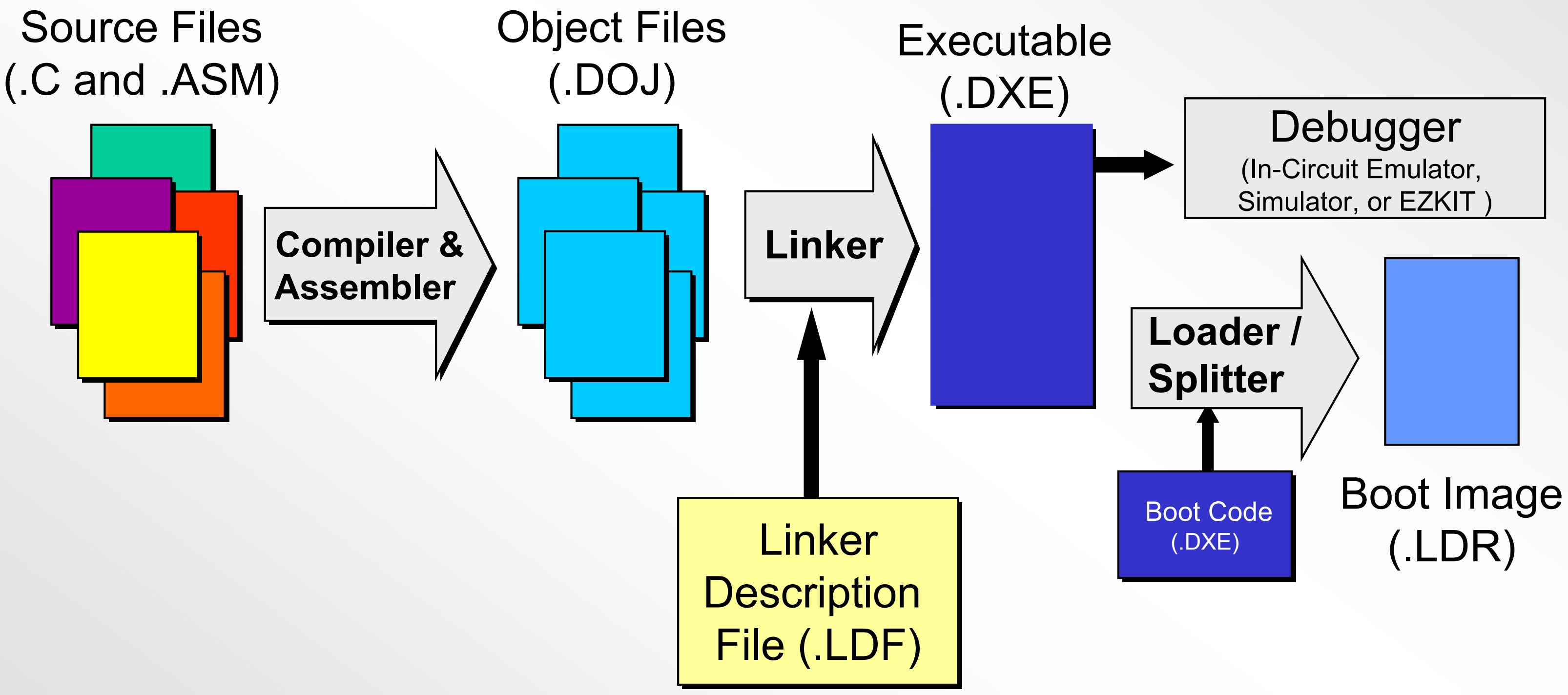
Build the project

Add source, header and .ldf files to your project.

File specific options: Select file, press right mouse button, choose: File Options

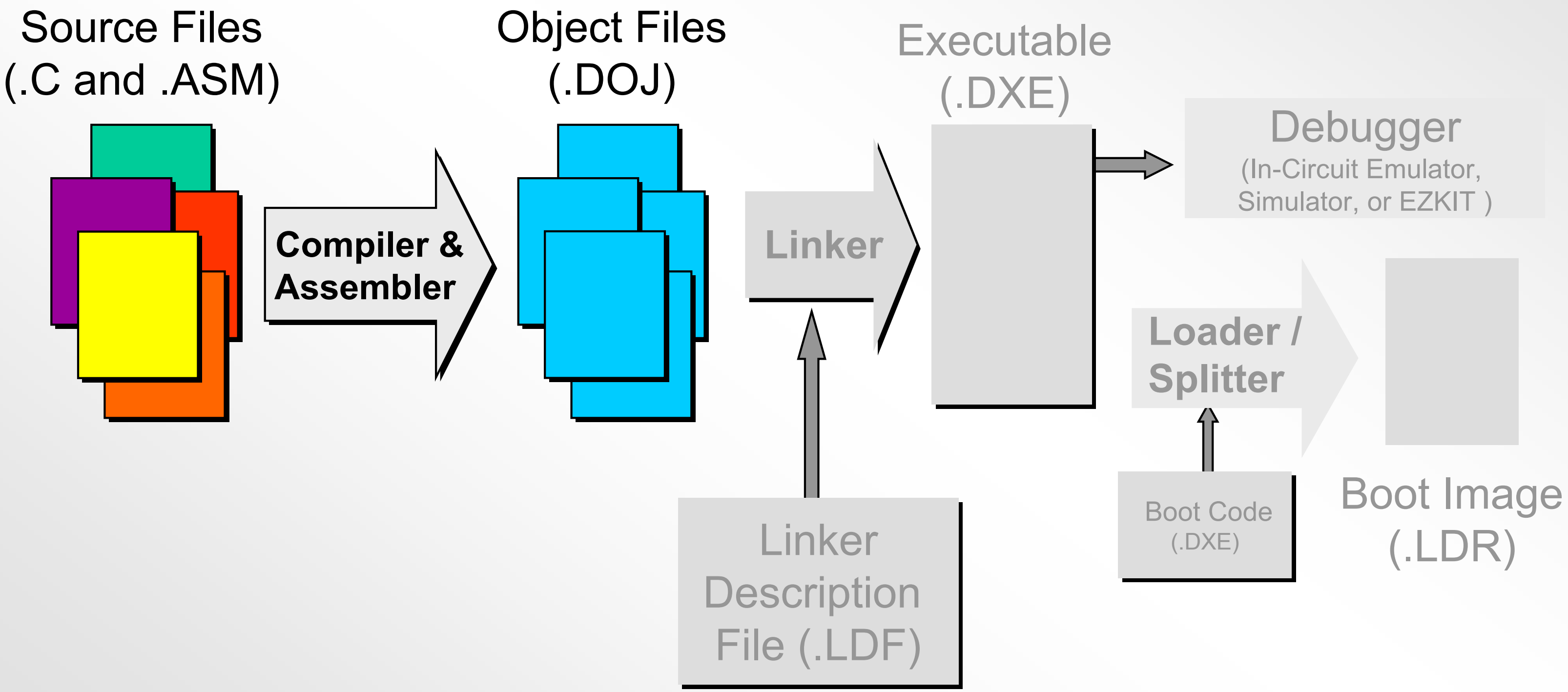
Software Development Flow

What Files Are Involved?



Software Development Flow

What Files Are Involved?

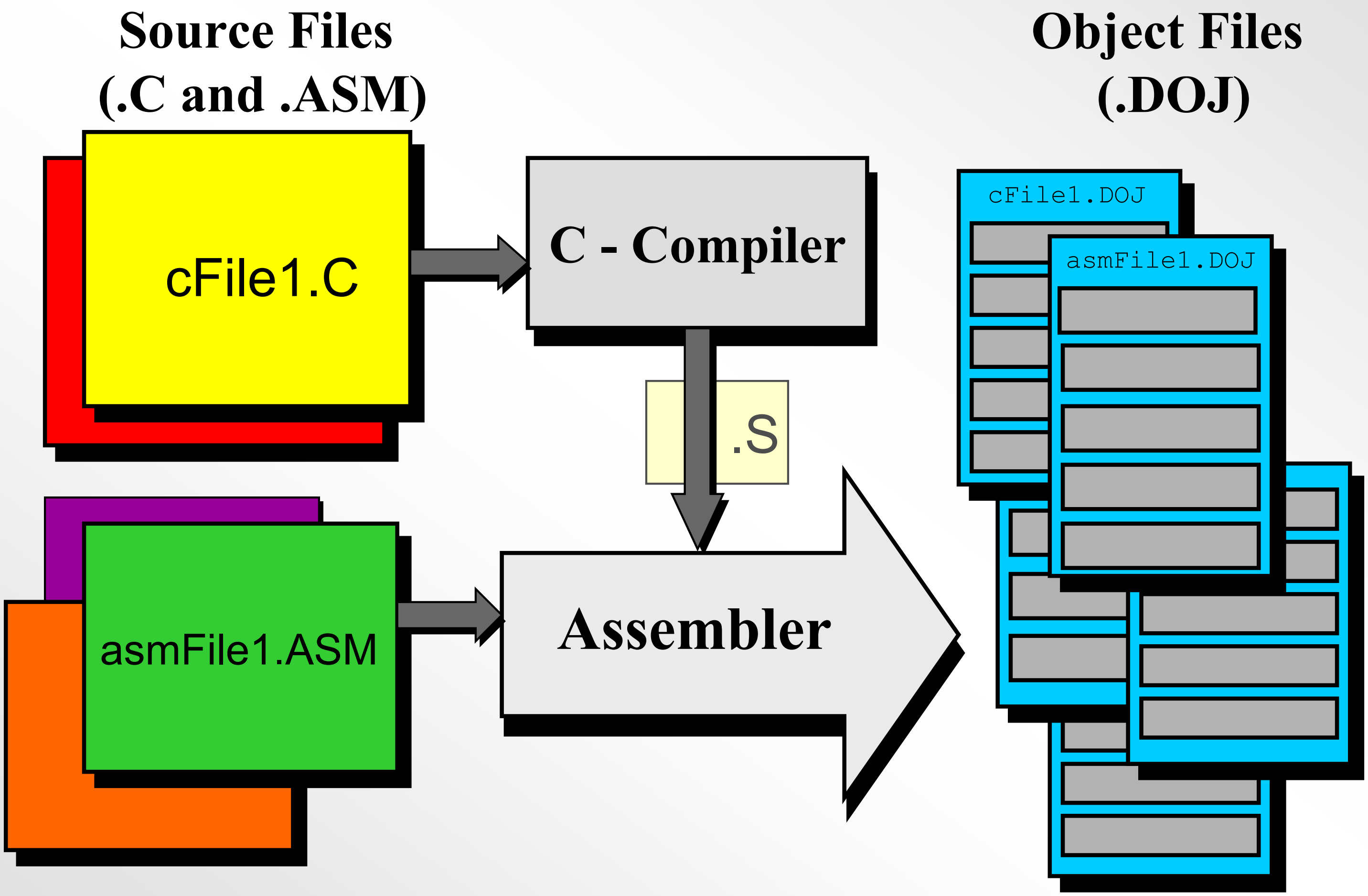


KAZTEK ENGINEERING

ANALOG DEVICES

Software Build Process

Step 1 - Compiling & Assembling

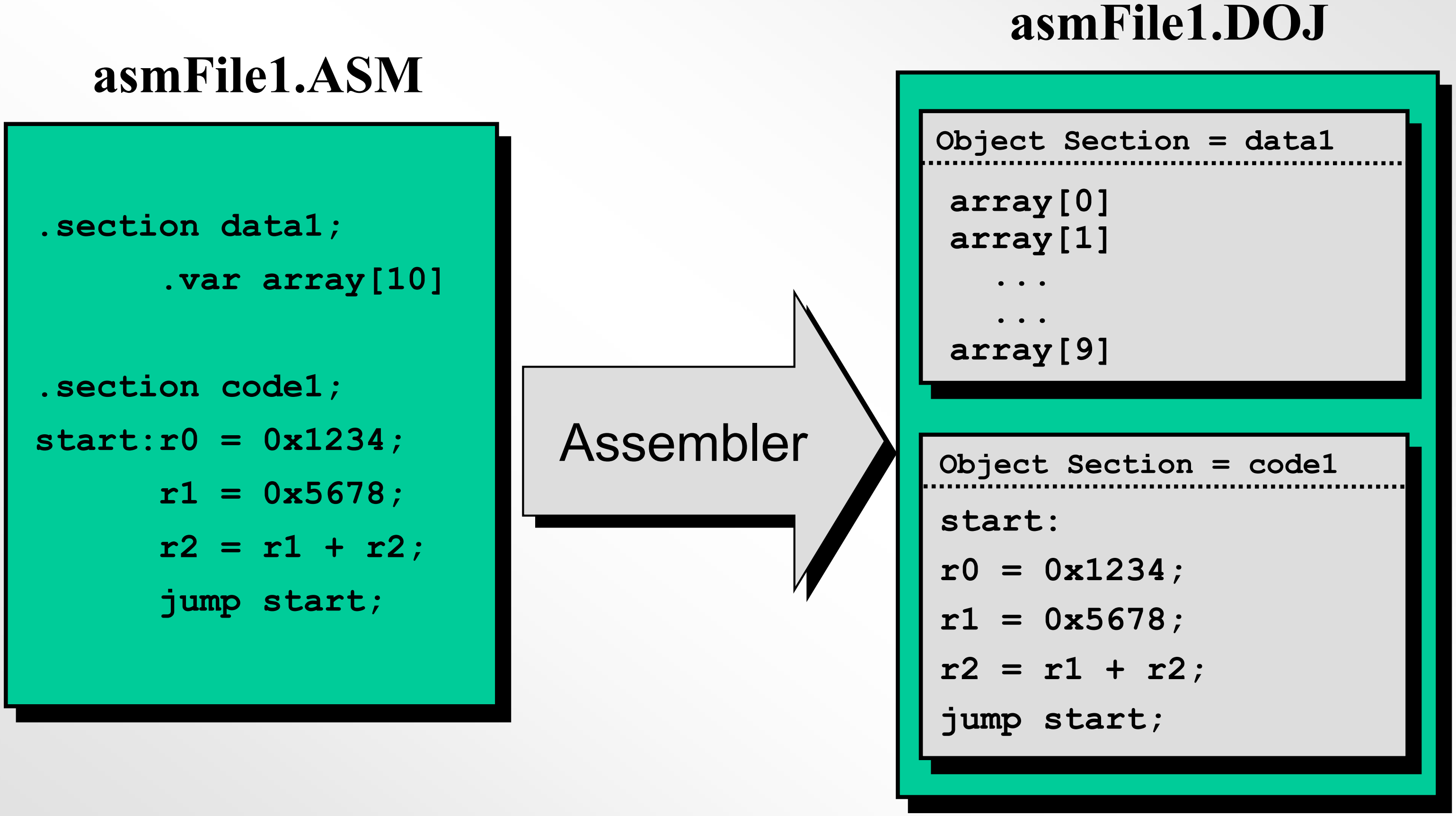


KAZTEK ENGINEERING

ANALOG DEVICES

Software Build Process

Step-1 Example: Assembly Source



asmFile1.ASM

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

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

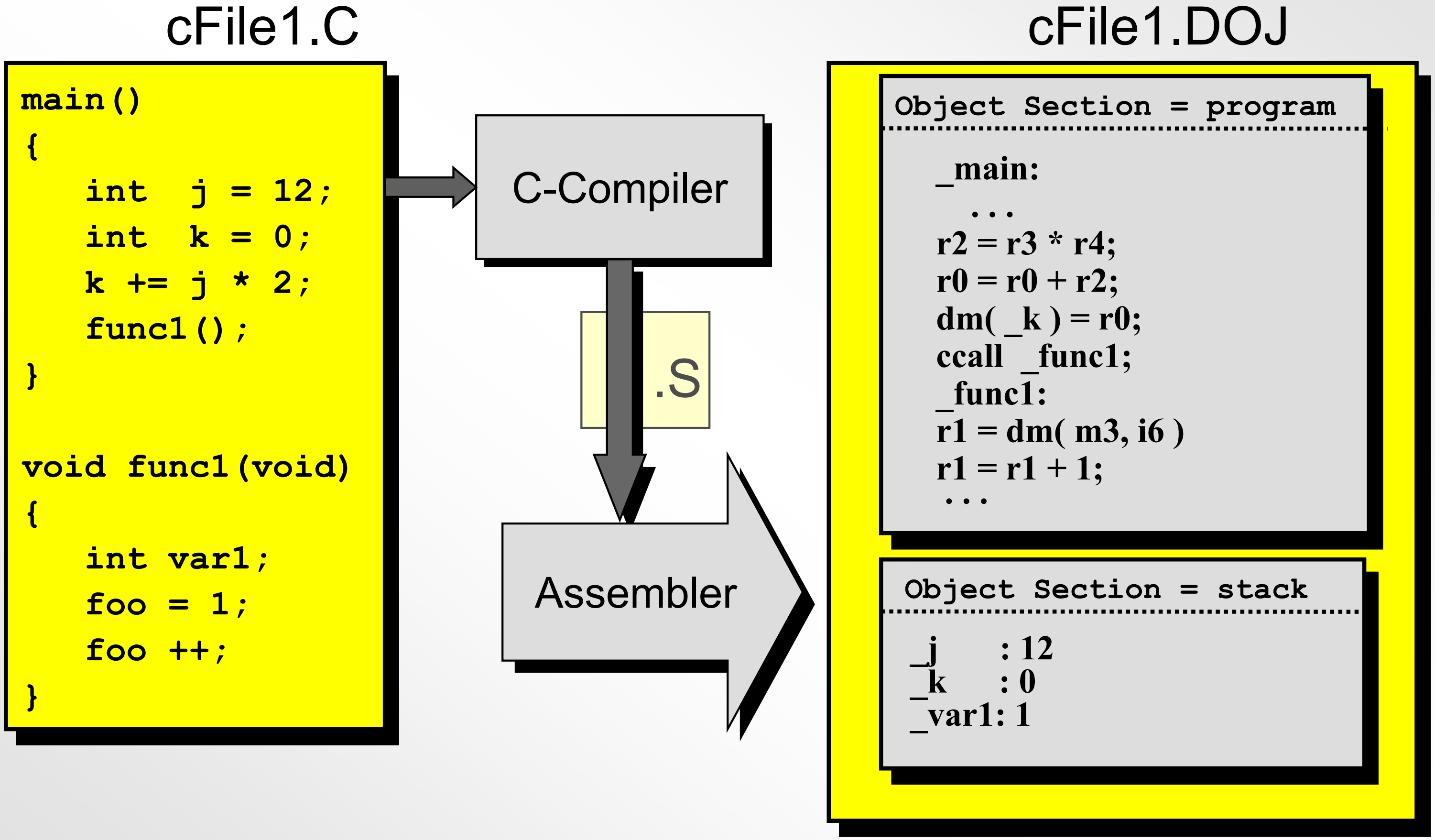
asmFile1.DOJ

```
Object Section = data1
........................................
 array[0]
 array[1]
    ...
    ...
 array[9]
```

```
Object Section = code1
........................................
start:
r0 = 0x1234;
r1 = 0x5678;
r2 = r1 + r2;
jump start;
```

Software Build Process

Step-1 Example: C Source



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

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

```
Object Section = program

  _main:
    . . .
  r2 = r3 * r4;
  r0 = r0 + r2;
  dm( _k ) = r0;
  ccall  _func1;
  _func1:
  r1 = dm( m3, i6 )
  r1 = r1 + 1;
   . . .
```

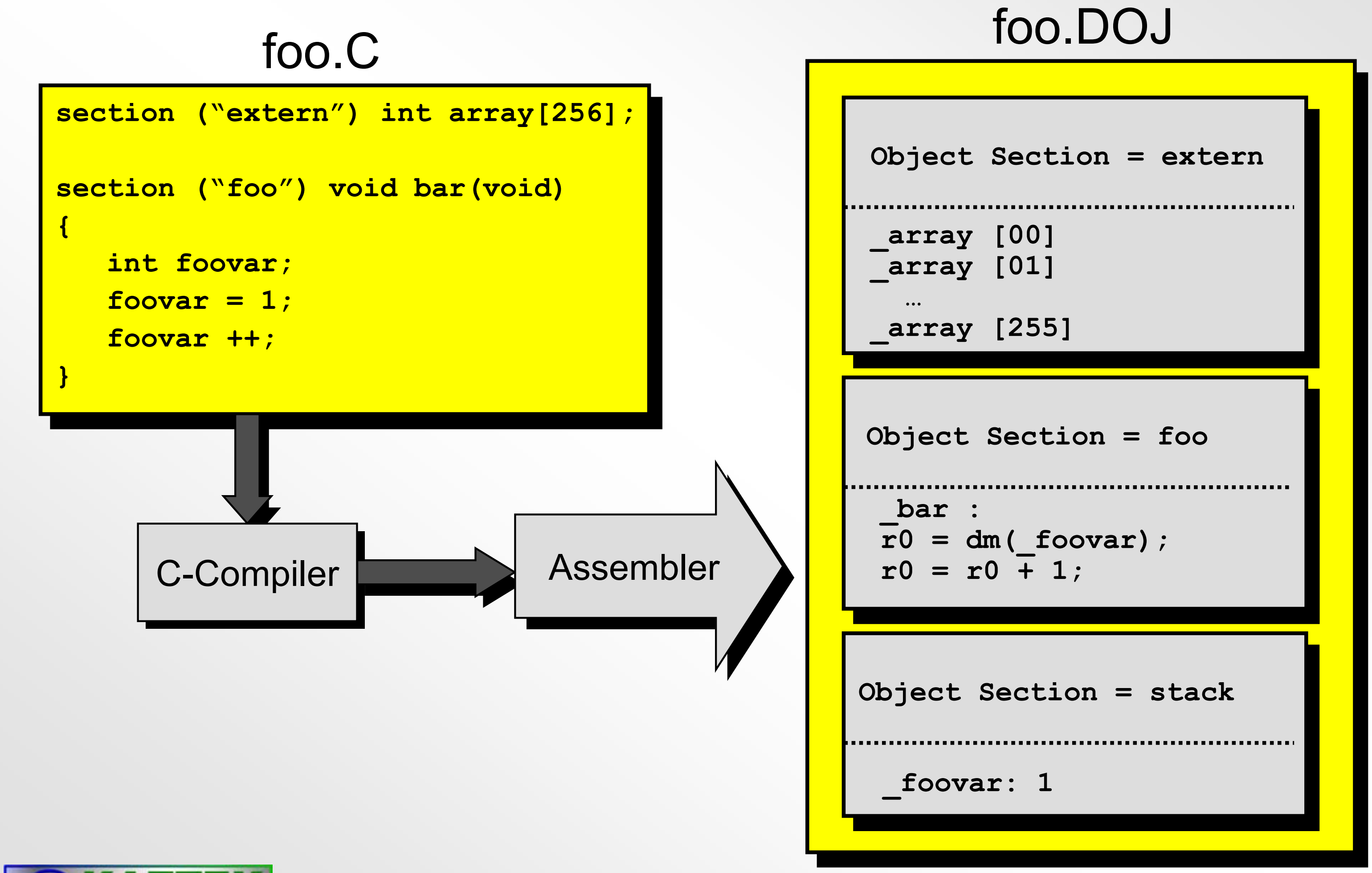
```
Object Section = stack

_j     : 12
_k     : 0
_var1: 1
```

KAZTEK ENGINEERING

ANALOG DEVICES

Software Build Process

Step 1 Example: C Source with Alternate Sections



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

```
Object Section = foo
.............................................
 _bar :
 r0 = dm(_foovar);
 r0 = r0 + 1;
```

```
Object Section = stack
.............................................
 _foovar: 1
```

Directives

- Preprocessor Directives
  - #define - define a macro or constant
  - #undef - undo macro definition
  - #if, #endif - conditional assembly
  - #else, #elif - multiple conditional blocks
  - #ifdef, #ifndef - condition based on macro definition
  - #include - include source code from another file
  - #error - report an error message
- Assembler directives
  - .ALIGN - specify alignment for code/data
  - .BYTE | .BYTE2 | .BYTE4 - define and initialize one-, two-, and four-byte data
  - .VAR - define and initialise 32-bit data object
  - .EXTERN - allow reference to global variable
  - .GLOBAL - change symbols scope to global
  - .SECTION - mark beginning of a section

Assembler

- Assembler operators

~	- ones complement
-	- unary minus
*	- multiply
/	- divide
%	- modulus
+	- addition
-	- subtraction
<<	- shift left
>>	- shift right
&	- bitwise AND (preprocessor only)
|	- bitwise inclusive OR
^	- bitwise exclusive OR (preprocessor only)

Assembler

- Assembler operators (cont’d)

– ADDRESS(symbol)	- address of symbol
– BITPOS(constant)	- bit position
– symbol	- address pointer to symbol
– LENGTH(symbol)	- length of symbol

Assembler

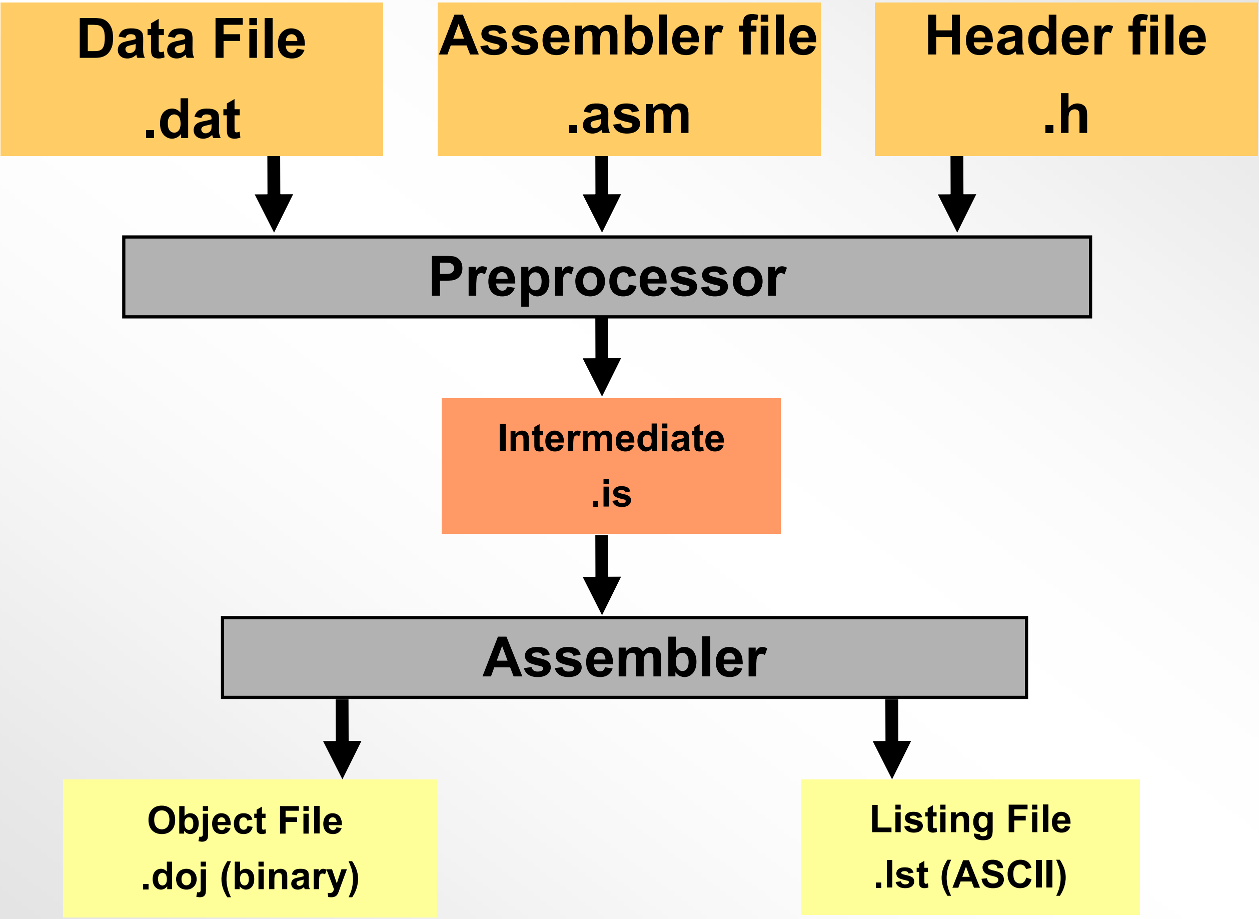
- Assembler command line switches
  - -Dmacro [definition] - define macro
  - -g - generate debug information
  - -h - output list of assembler switches
  - -i directory - search directory for included files
  - -l filename - output named listing file
  - -li filename - output named listing file with #include files
  - -M - generate dependencies for #include and data files
  - -MM - generate make dependencies for #include and data files
  - -Mo filename - write make dependencies to file
  - -Mt filename - specify the make dependencies target name

Assembler

- Assembler command line switches (cont’d)

– -micaswarn	- treat multi-issue conflicts as warning
– -o filename	- output the named object file
– -pp	- run preprocessor only (do not assemble)
– -proc processor	- specify processor
– -sp	- assemble without preprocessing
– -v	- display information on each assembly phase
– -version	- display version information for assembler
– -w	- remove all assembler-generated warnings
– -Wnumber	- suppress any report of the specified warning

Assembler



Data File
.dat

Assembler file
.asm

Header file
.h

Preprocessor

Intermediate
.is

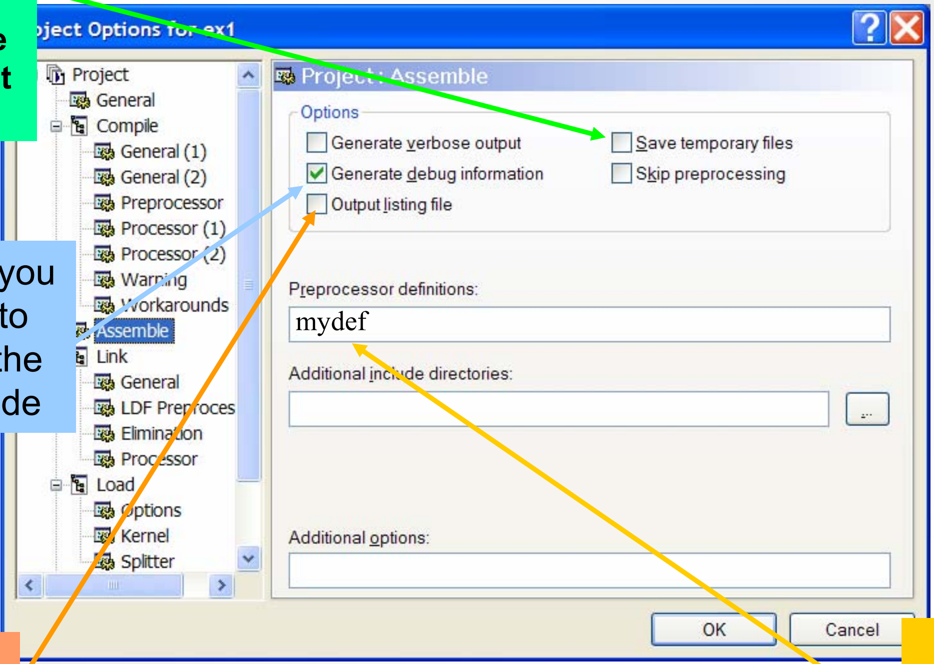
Assembler

Object File
.doj (binary)

Listing File
.lst (ASCII)

Assembler Property Page

If you want to get the intermediate .is file, select here



Project: Assemble

Options

- ☐ Generate verbose output
- ☑ Generate debug information
- ☐ Output listing file
- ☐ Save temporary files
- ☐ Skip preprocessing

Preprocessor definitions: mydef

Additional include directories:

Additional options:

If chosen, you are able to debug in the source code

If chosen, a listing file will be created

```
#include <defBF533.h>
#include "myheader.h"

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

Depending on definitions, you can select different codes

Sections in Assembler Files

- The .SECTION directive marks the beginning of a logical section
  - data and code form the content of a section
  - Multiple sections may be used within a single source file
  - Any section name may be chosen

```
.SECTION   data_a;
        .BYTE data_array[N];

.SECTION   data_b;
        .VAR coeff_array[N];
        .VAR x = 0x12345689;

.SECTION   program;
_main:  P0.H=data_array;
        P0.L=data_array;
        L0=length(data_array);
            . . .
```

The defBF533.h Header Files

- **Allows Programmer to Use Symbols for Memory Mapped Registers**
- **Located in: `\\VisualDSP\Blackfin\include\`**

  To include it use:

  ```
  #include <defBF533.h> or
  #include <defLPBlackfin.h>
  ```

Example:

```
P0.L = LO(TIMER0_CONFIG);
P0.H = HI(TIMER0_CONFIG );
R0 = 0x2345(Z);
W[P0] = R0.L; // Write 0x2345 to TIMER0_CONFIG
```

- **Operators LO(*expression*) and HI*(expression)* must be used to load the 32-bit macros that are #define'd in defBF533.h into 16-bit registers.**

NOTE: *expression* can be symbolic or constant

Assembler Source File Example

```
#include <defBF533.h>

#define N  20                        // replace N by 20

.GLOBAL  start;

.SECTION data_a;                     // data in L1 memory bank A
.VAR     buffer[N]="fill.dat";       // initialize data from file

.SECTION data_b;                     // data in L1 memory bank B
.VAR     xy = 0x12345678;            // initialize var with 32bit value

.SECTION L2_program;                 // instructions in L2 memory

start:     I0 = buffer (z);          // get low address word of array and load index register
           I0.H = buffer;            // get high address word of array and load index register
           B0=I0;                    // load base register with address

           L0=N*4;                   // size of array (circular buffer!) in bytes

           R0=0;
           P0=N;

           lsetup(loopstart,loopend) LC1 = P0;        // setup loop
loopstart: R0 += 1;                  // 1st instruction in loop
loopend:   [I0++]=R0;                // last instruction in loop
```

Macros

```
#define mymacro(x,y)          R0 = x; R1 = y; R2 = R0 + R1
```

Semicolon either here or here

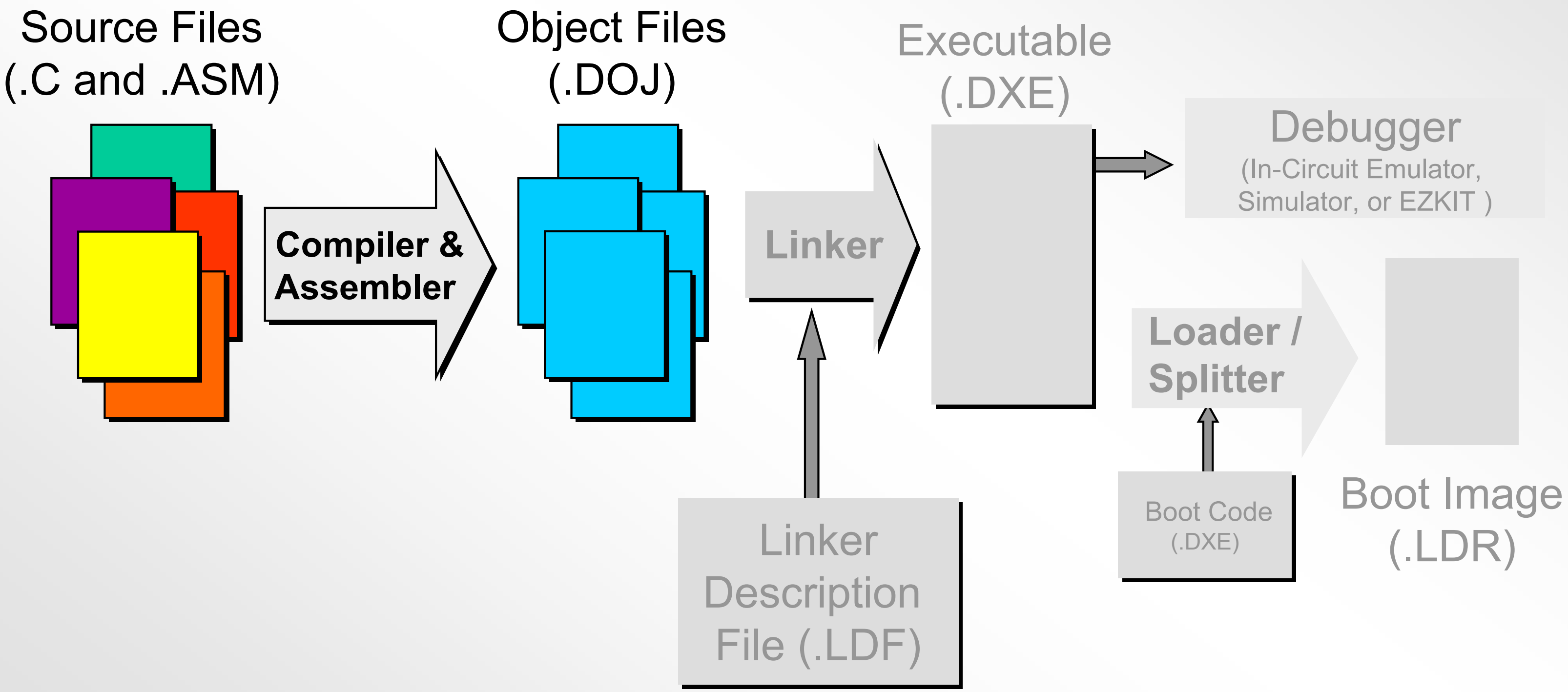
```
.SECTION  program;

start:      mymacro(0x4,P0);

             [I0++] = R2;
```

The Preprocessor will create the following:

```
start:       R0 = 0x4 (Z);
             R1 = P0;
             R2 = R0 + R1;
             [I0++] = R2;
```

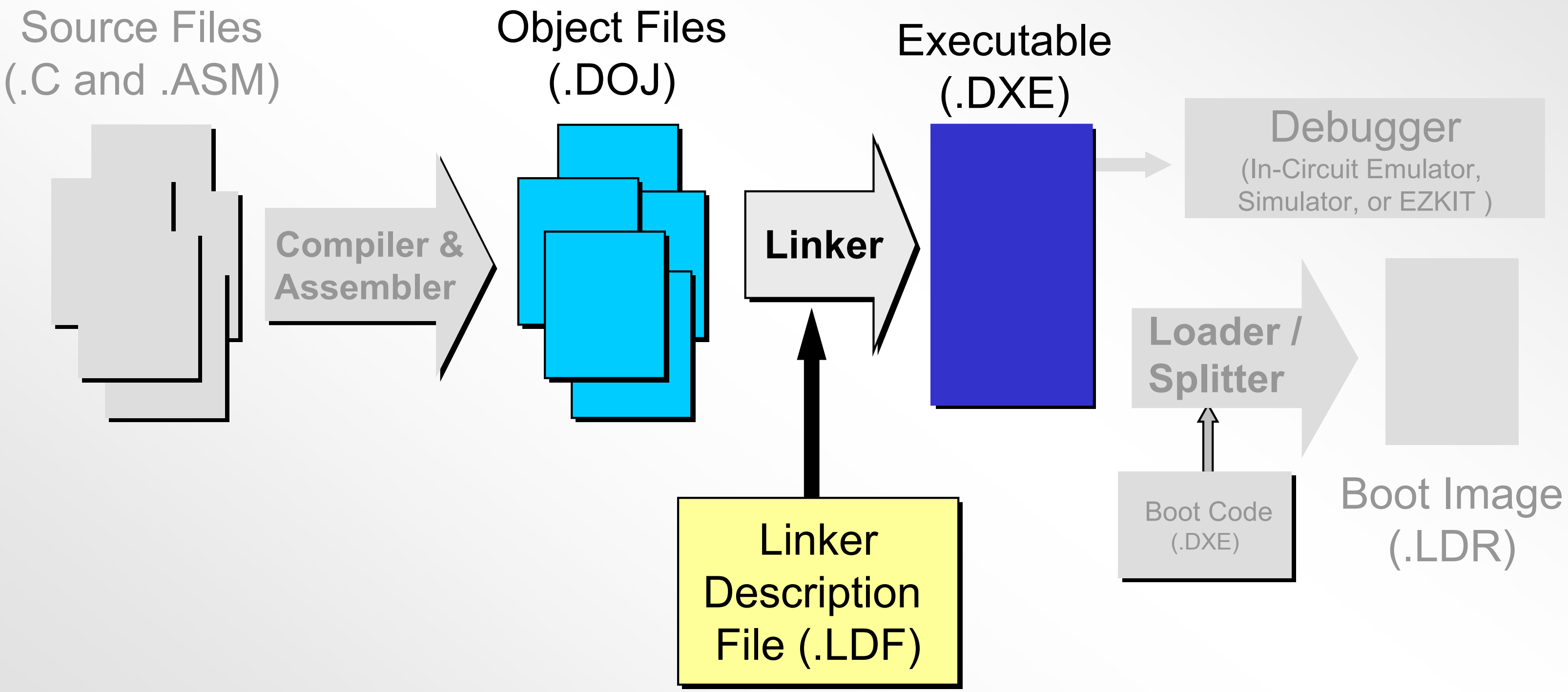
Software Development Flow

Step 1- Compiling & Assembling



Software Development Flow

Step 2 - Linking

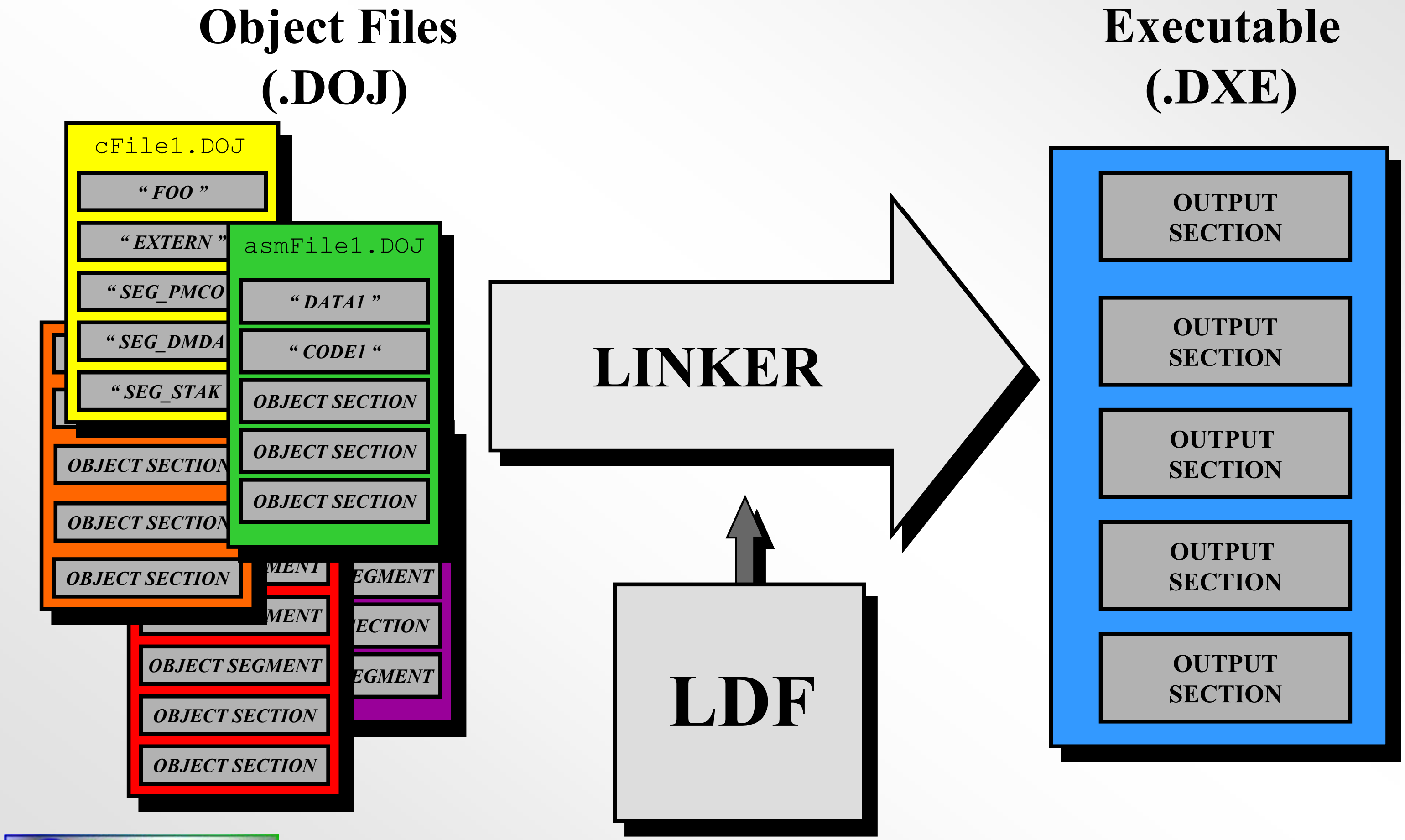


KAZTEK ENGINEERING

ANALOG DEVICES

Linker Description File

Step 2 - Linking



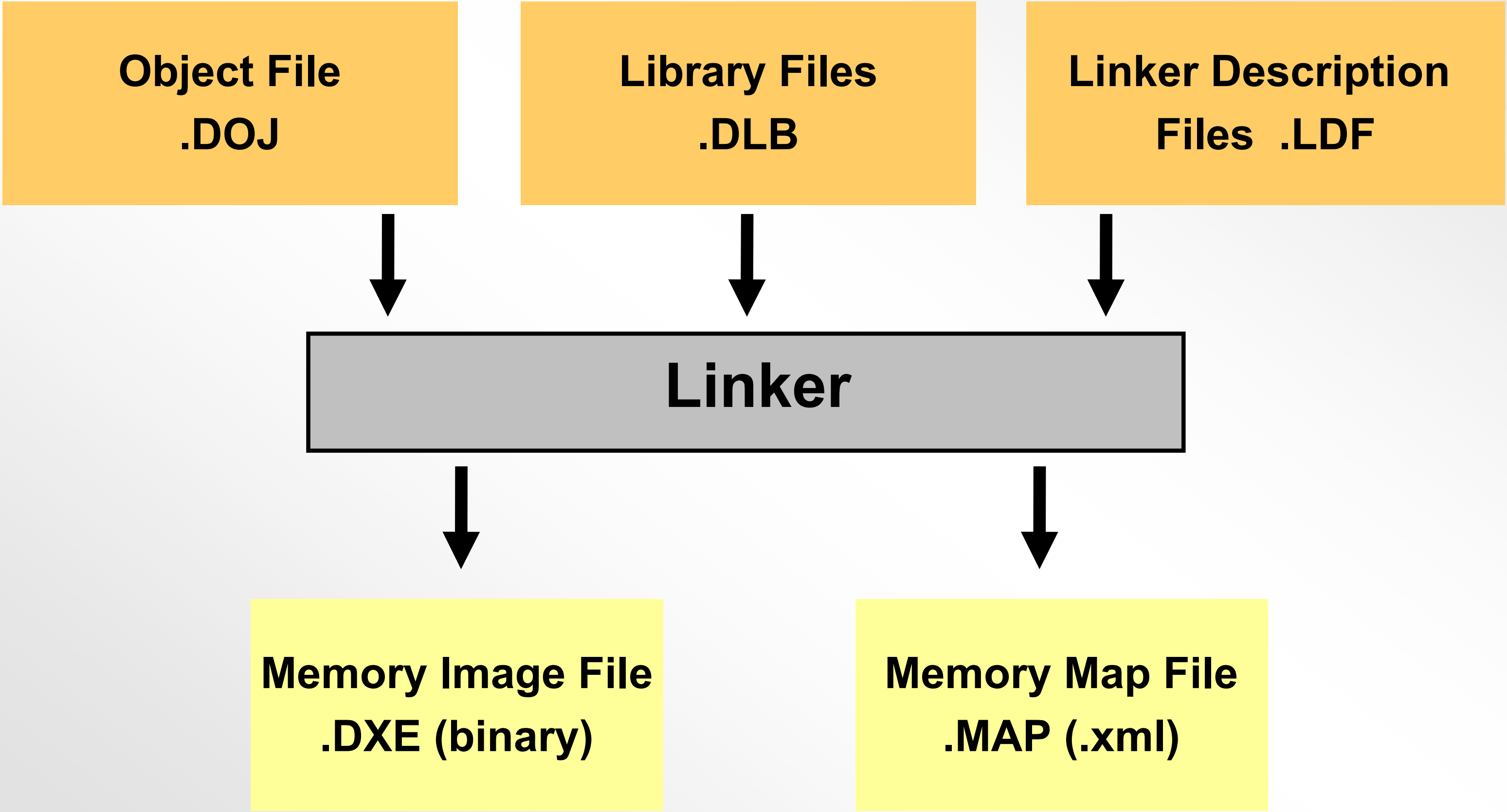
KAZTEK ENGINEERING

ANALOG DEVICES

Linker

- Generates a Complete Executable DSP Program (.dxe)
- Resolves All External References
- Assigns Addresses to re-locatable Code and Data Spaces
- Generates Optional Memory Map
- Output in ELF format
  - Used by downstream tools such as Loader, Simulator, and Emulator
- Controlled by linker commands contained in a linker description file (LDF)
  - An LDF is required for each project
  - Typically modify a default one to suit target application

Linker



Object File
.DOJ

Library Files
.DLB

Linker Description
Files .LDF

Linker

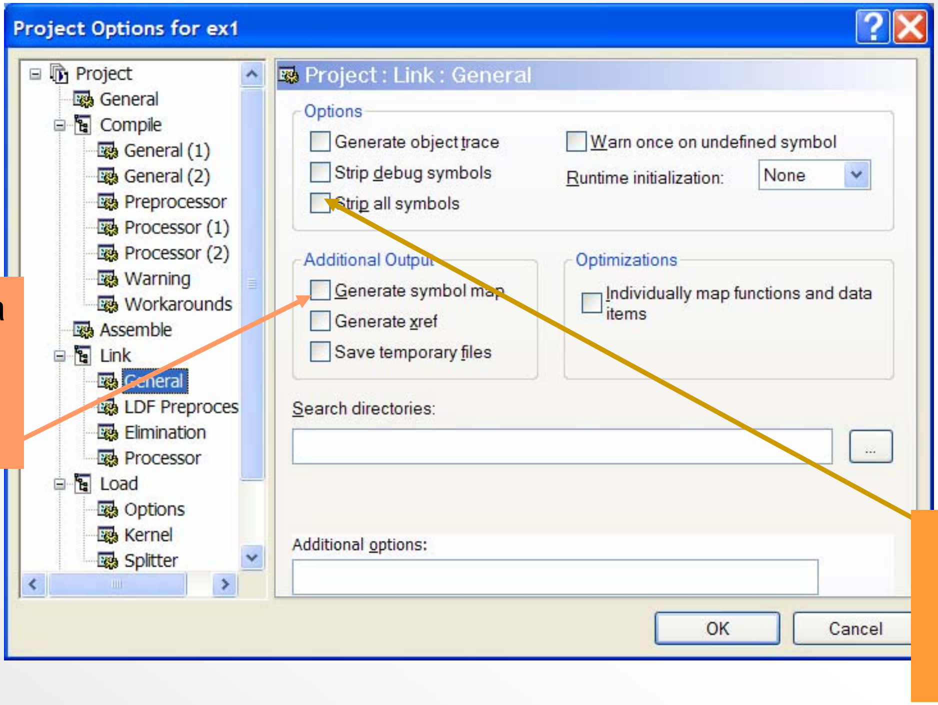
Memory Image File
.DXE (binary)

Memory Map File
.MAP (.xml)

KAZTEK ENGINEERING

ANALOG DEVICES

Linker Property Page



Project Options for ex1

- Project
  - General
  - Compile
    - General (1)
    - General (2)
    - Preprocessor
    - Processor (1)
    - Processor (2)
    - Warning
    - Workarounds
  - Assemble
  - Link
    - General
    - LDF Preproces
    - Elimination
    - Processor
  - Load
    - Options
    - Kernel
    - Splitter

Project : Link : General

Options

□ Generate object trace

□ Strip debug symbols

□ Strip all symbols

□ Warn once on undefined symbol

Runtime initialization: None

Additional Output

□ Generate symbol map

□ Generate xref

□ Save temporary files

Optimizations

□ Individually map functions and data items

Search directories:

Additional options:

If chosen, a .map file will be created

All symbol names will be removed, if chosen

The Linker Description File (LDF)

- **The link process is controlled by a linker command language**
- **The LDF provides a complete specification of mapping between the linker's input files and its output.**
- **It controls**
    - input files
    - output file
    - target memory configuration
- **Preprocessor Support**

LDF consists of three primary parts

- **Global Commands**
  - Defines architecture or processor
  - Directory search paths
  - Libraries and object files to include
- **Memory Description**
  - Defines memory segments
- **Link Project Commands**
  - Mapping of input sections to memory segments
  - Output file name
  - Link against object file list

Example LDF

Global Commands & Memory Description

```
ARCHITECTURE (ADSP-BF533)
SEARCH_DIR ($ADI_DSP\Blackfin\lib)
$OBJECTS = $COMMAND_LINE_OBJECTS;

MEMORY
{
     seg_data_a      { TYPE(RAM) START(0xFF800000)  END(0xFF803FFF) WIDTH(8) }
     seg_data_b      { TYPE(RAM) START(0xFF900000)  END(0xFF903FFF) WIDTH(8) }
     seg_data_scr    { TYPE(RAM) START(0xFFB00000) END(0xFFB00FFF) WIDTH(8) }
     seg_prog        { TYPE(RAM) START(0xFFA00000) END(0xFFA03FFF) WIDTH(8) }
}
```

Global Commands

Segment name

Start address

End address

Memory width

Example LDF (con‘t)

Link Commands

```
PROCESSOR p0
{
     OUTPUT( $COMMAND_LINE_OUTPUT_FILE )
     SECTIONS

{
          sec_data_a
          { INPUT_SECTIONS( $OBJECTS(data_a) ) } > seg_data_a
          sec_data_b SHT_NOBITS
          { INPUT_SECTIONS( $OBJECTS(data_b) ) } > seg_data_b
          sec_data_scr
          { INPUT_SECTIONS( $OBJECTS(data_scr) ) } > seg_data_scr
          sec_prog
          { INPUT_SECTIONS( $OBJECTS(prog) ) } >seg_prog
     }
}
```

OBJECT SECTIONS from assembly files

MEMORY SEGMENTS Declared in the LDF

DXE SECTION NAMES Used in .map file

Keyword: Data in that SECTION will not be initialized

Expert Linker

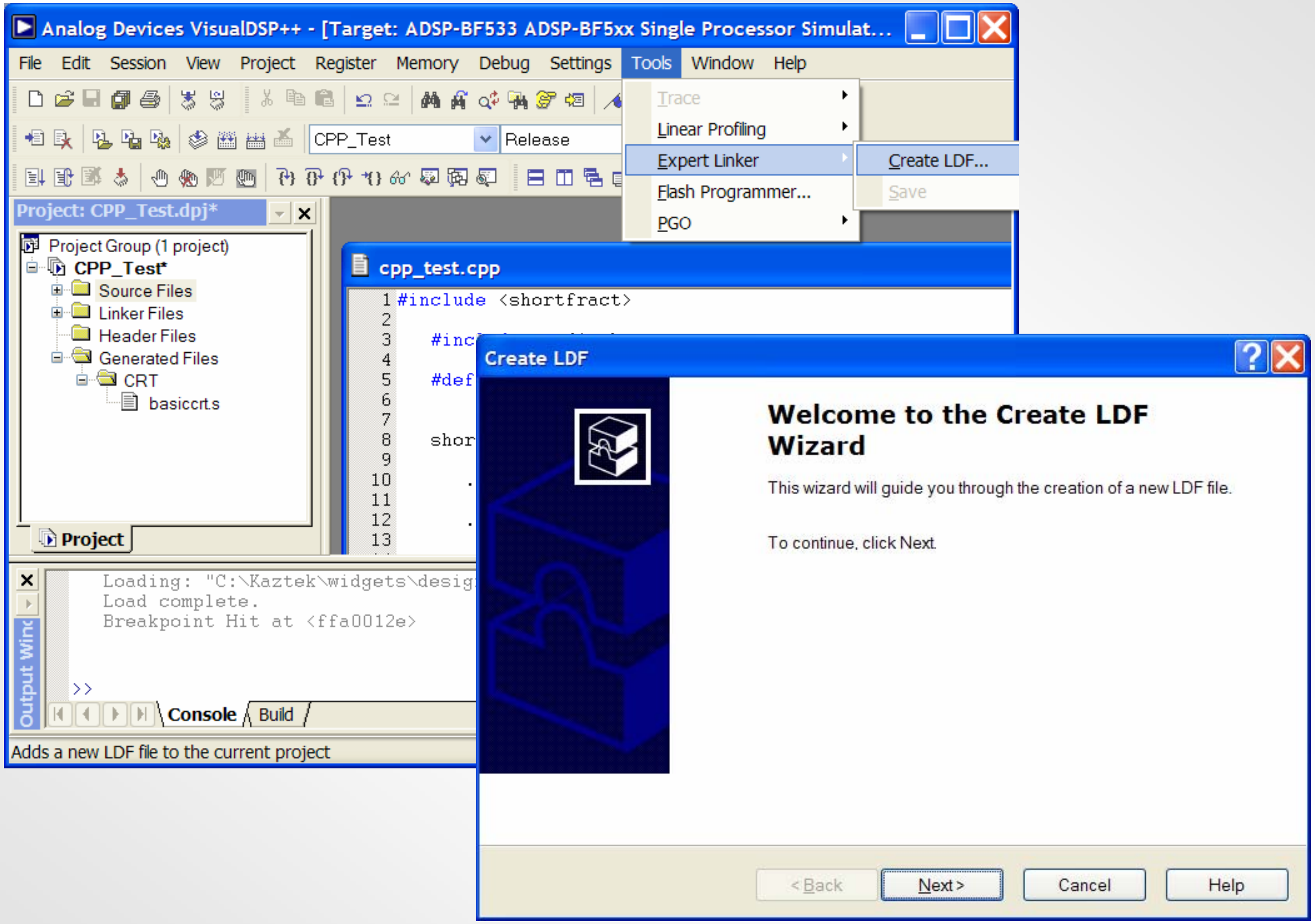
Using the LDF Wizard

Expert Linker Features

Expert Linker is a Graphical tools that can:

- Use wizards to create LDF files
- Define a DSP's target memory map
- Drag and Drop object sections into the memory map
- Present watermarks for max Heap and Stack usage
- Graphically Manage Overlay support
- Import Legacy LDF files
- Graphically highlights code elimination of unused objects
- Profile object sections in memory

Create LDF Wizard



```
#include <shortfract>

   #inc

   #def


   shor

        .

        .
```

```
Loading: "C:\Kaztek\widgets\desig
Load complete.
Breakpoint Hit at <ffa0012e>

>>
```

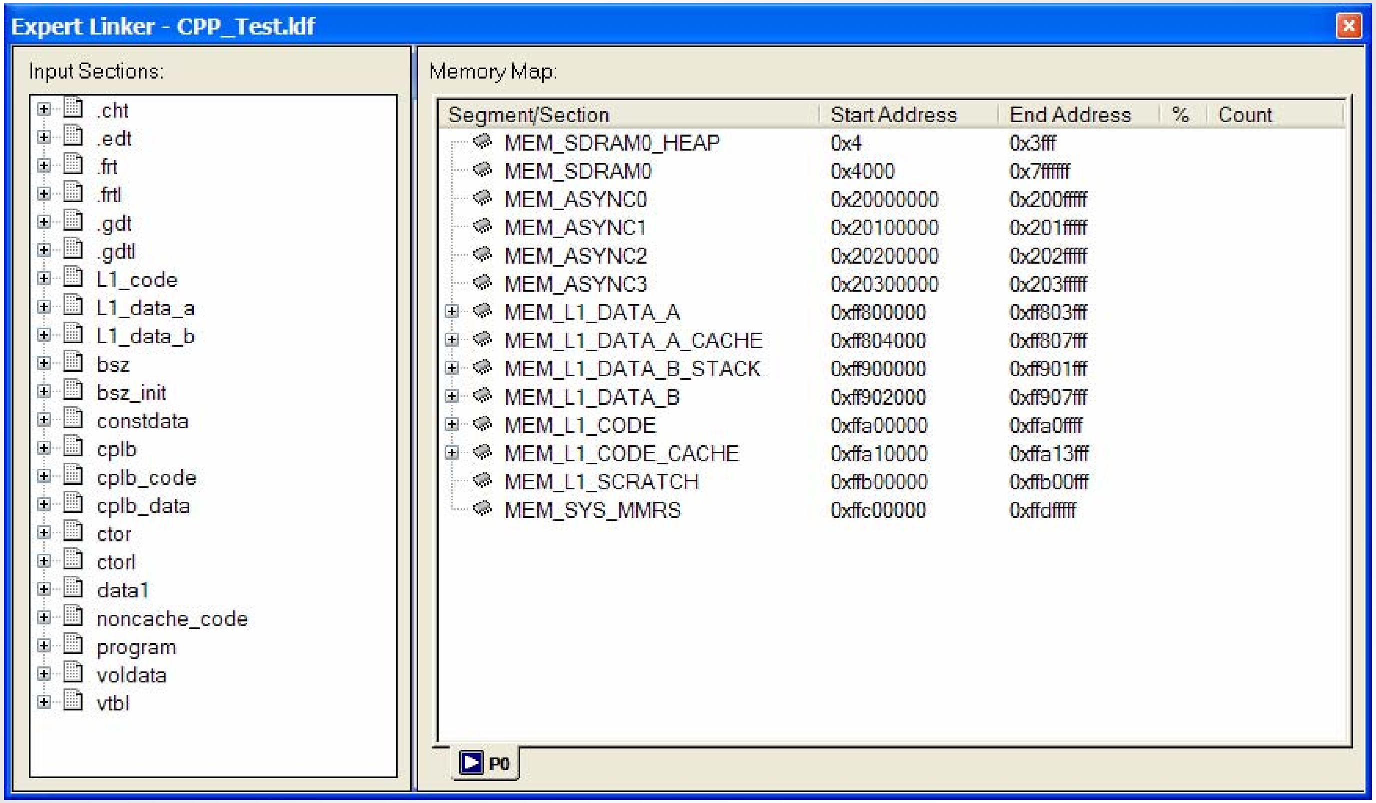
Adds a new LDF file to the current project

Welcome to the Create LDF Wizard

This wizard will guide you through the creation of a new LDF file.

To continue, click Next.

LDF Result



Expert Linker - CPP_Test.ldf

Input Sections:

- .cht
- .edt
- .frt
- .frtl
- .gdt
- .gdtl
- L1_code
- L1_data_a
- L1_data_b
- bsz
- bsz_init
- constdata
- cplb
- cplb_code
- cplb_data
- ctor
- ctorl
- data1
- noncache_code
- program
- voldata
- vtbl

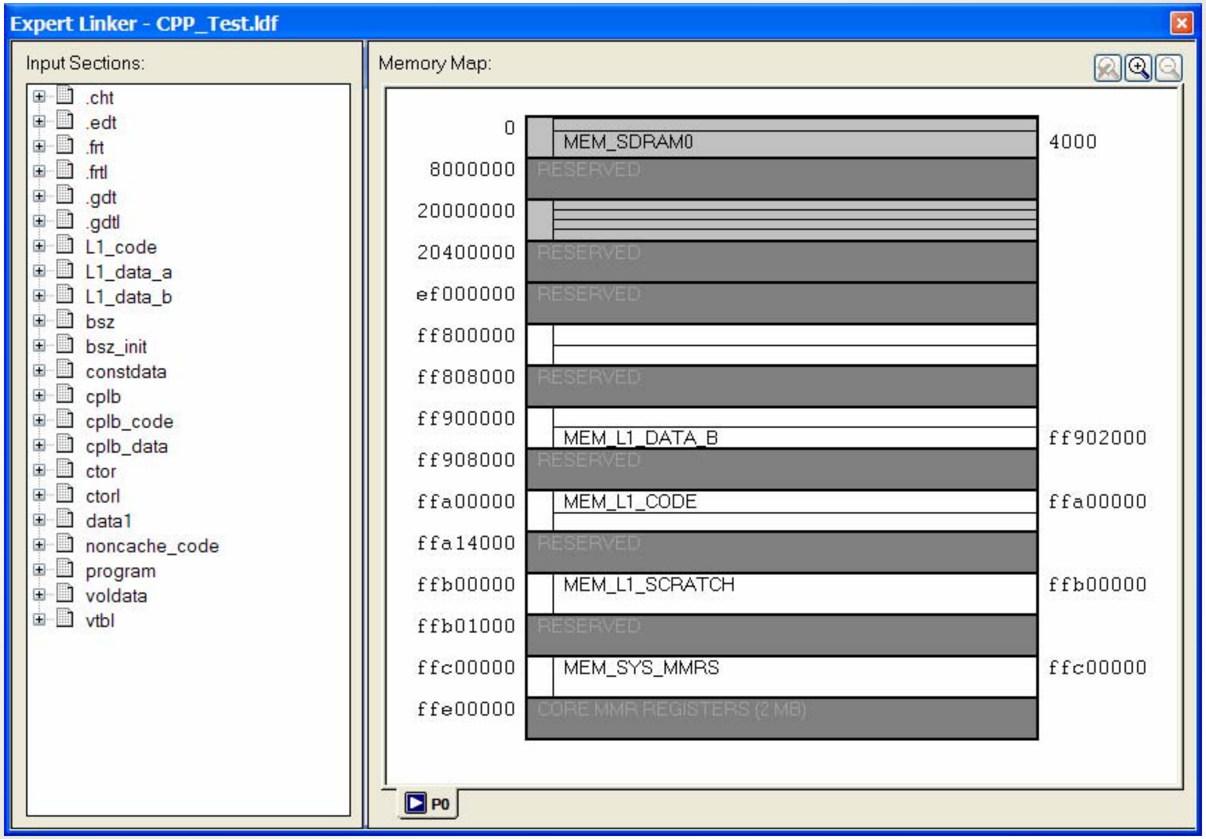
Memory Map:

Segment/Section	Start Address	End Address	%	Count
MEM_SDRAM0_HEAP	0x4	0x3fff		
MEM_SDRAM0	0x4000	0x7ffffff		
MEM_ASYNC0	0x20000000	0x200fffff		
MEM_ASYNC1	0x20100000	0x201fffff		
MEM_ASYNC2	0x20200000	0x202fffff		
MEM_ASYNC3	0x20300000	0x203fffff		
MEM_L1_DATA_A	0xff800000	0xff803fff		
MEM_L1_DATA_A_CACHE	0xff804000	0xff807fff		
MEM_L1_DATA_B_STACK	0xff900000	0xff901fff		
MEM_L1_DATA_B	0xff902000	0xff907fff		
MEM_L1_CODE	0xffa00000	0xffa0ffff		
MEM_L1_CODE_CACHE	0xffa10000	0xffa13fff		
MEM_L1_SCRATCH	0xffb00000	0xffb00fff		
MEM_SYS_MMRS	0xffc00000	0xffdfffff		

P0

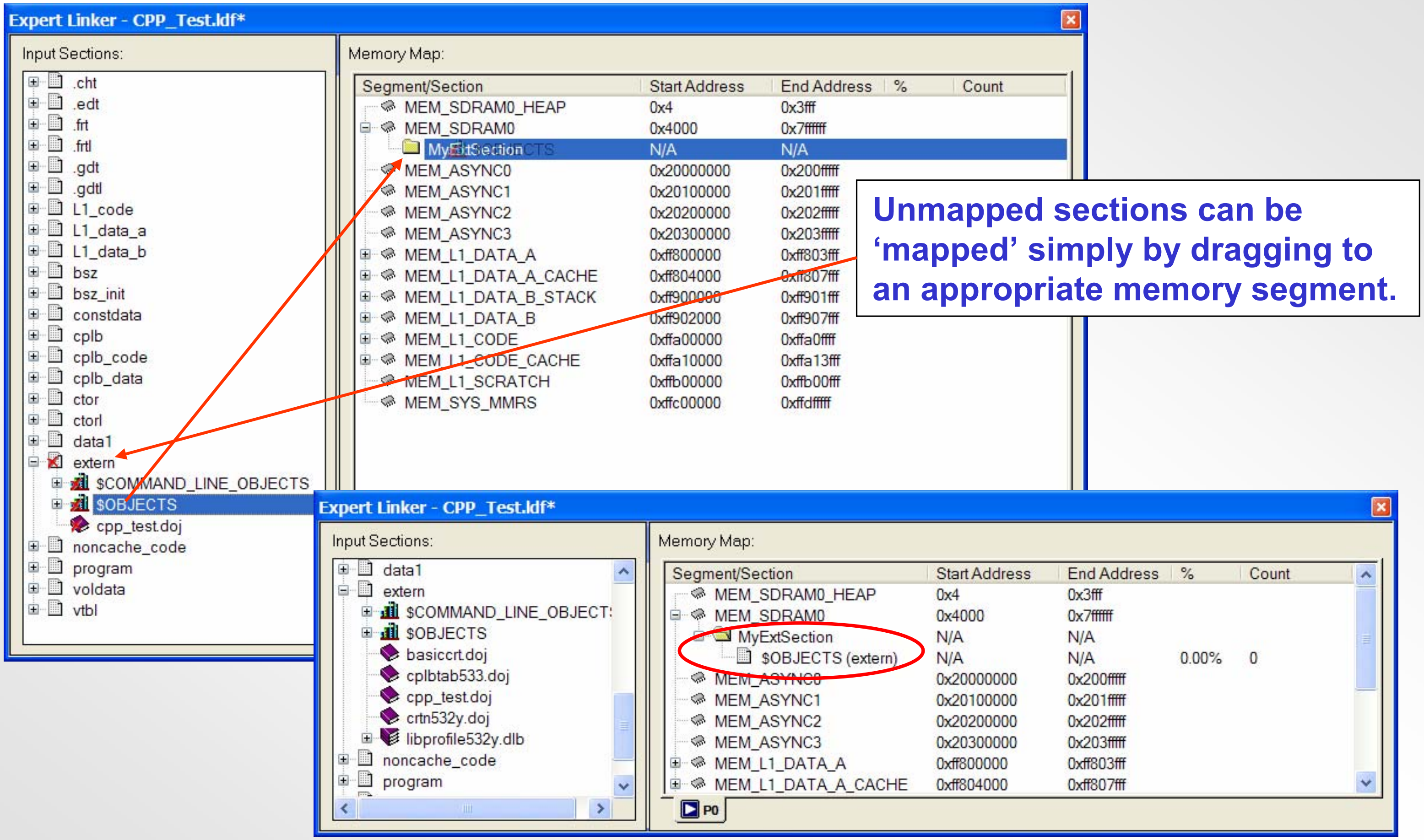
This is a memory map view of the generated .ldf file. In this mode, each section's start and end address are shown in a list format.

LDF Result (cont'd)



This is a graphical view of the memory map. Double click on the section to zoom in.

Control Mapping of Sections



Expert Linker - CPP_Test.ldf*

Input Sections:

- .cht
- .edt
- .frt
- .frtl
- .gdt
- .gdtl
- L1_code
- L1_data_a
- L1_data_b
- bsz
- bsz_init
- constdata
- cplb
- cplb_code
- cplb_data
- ctor
- ctorl
- data1
- extern
  - $COMMAND_LINE_OBJECTS
  - $OBJECTS
  - cpp_test.doj
- noncache_code
- program
- voldata
- vtbl

Memory Map:

Segment/Section	Start Address	End Address	%	Count
MEM_SDRAM0_HEAP	0x4	0x3fff		
MEM_SDRAM0	0x4000	0x7ffffff		
MyExtSection	N/A	N/A		
MEM_ASYNC0	0x20000000	0x200fffff		
MEM_ASYNC1	0x20100000	0x201fffff		
MEM_ASYNC2	0x20200000	0x202fffff		
MEM_ASYNC3	0x20300000	0x203fffff		
MEM_L1_DATA_A	0xff800000	0xff803fff		
MEM_L1_DATA_A_CACHE	0xff804000	0xff807fff		
MEM_L1_DATA_B_STACK	0xff900000	0xff901fff		
MEM_L1_DATA_B	0xff902000	0xff907fff		
MEM_L1_CODE	0xffa00000	0xffa0ffff		
MEM_L1_CODE_CACHE	0xffa10000	0xffa13fff		
MEM_L1_SCRATCH	0xffb00000	0xffb00fff		
MEM_SYS_MMRS	0xffc00000	0xffdfffff		

Unmapped sections can be 'mapped' simply by dragging to an appropriate memory segment.

Expert Linker - CPP_Test.ldf*

Input Sections:

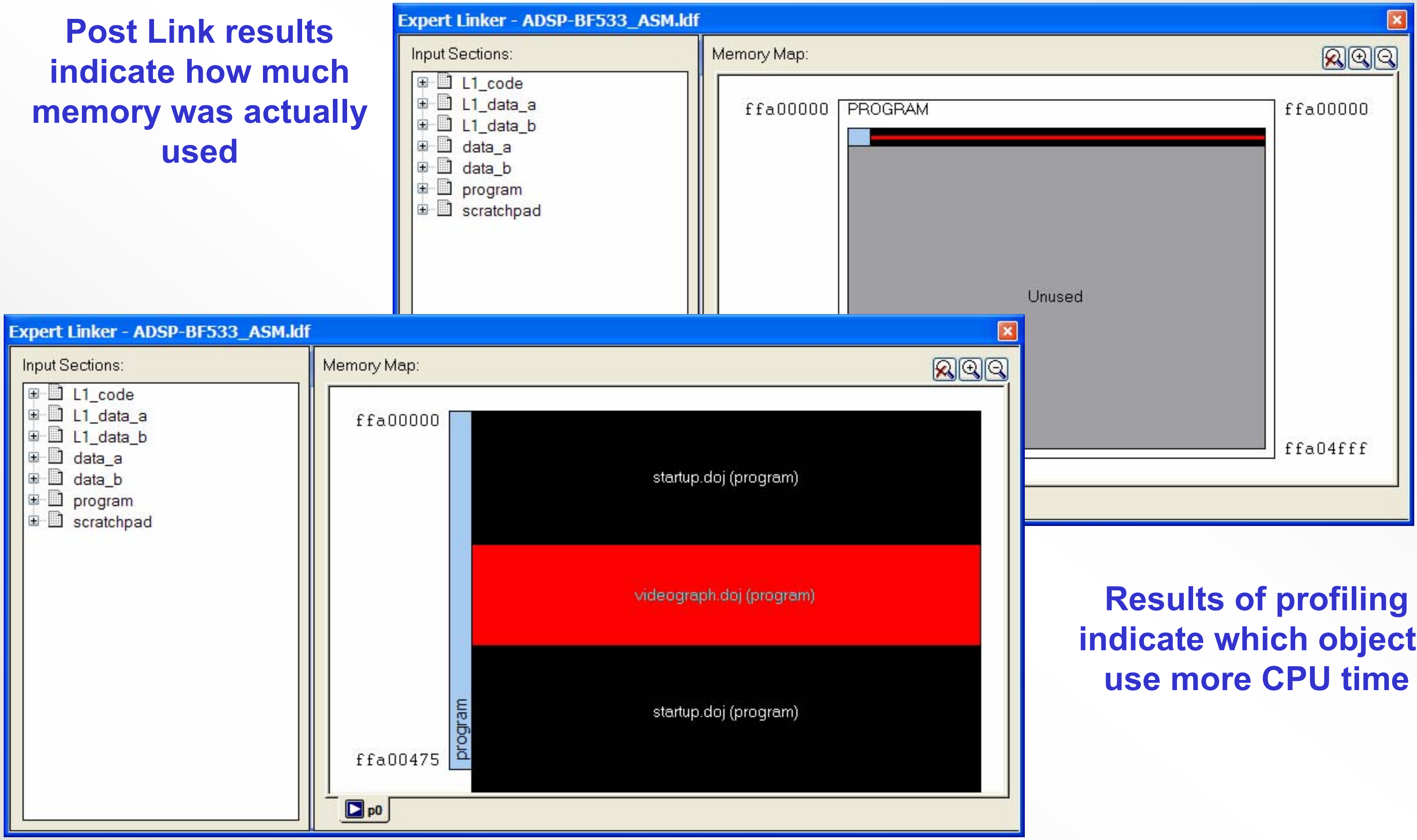
- data1
- extern
  - $COMMAND_LINE_OBJECT:
  - $OBJECTS
  - basiccrt.doj
  - cplbtab533.doj
  - cpp_test.doj
  - crtn532y.doj
  - libprofile532y.dlb
- noncache_code
- program

Memory Map:

Segment/Section	Start Address	End Address	%	Count
MEM_SDRAM0_HEAP	0x4	0x3fff		
MEM_SDRAM0	0x4000	0x7ffffff		
MyExtSection	N/A	N/A		
$OBJECTS (extern)	N/A	N/A	0.00%	0
MEM_ASYNC0	0x20000000	0x200fffff		
MEM_ASYNC1	0x20100000	0x201fffff		
MEM_ASYNC2	0x20200000	0x202fffff		
MEM_ASYNC3	0x20300000	0x203fffff		
MEM_L1_DATA_A	0xff800000	0xff803fff		
MEM_L1_DATA_A_CACHE	0xff804000	0xff807fff		

Post Link and Profiling Results

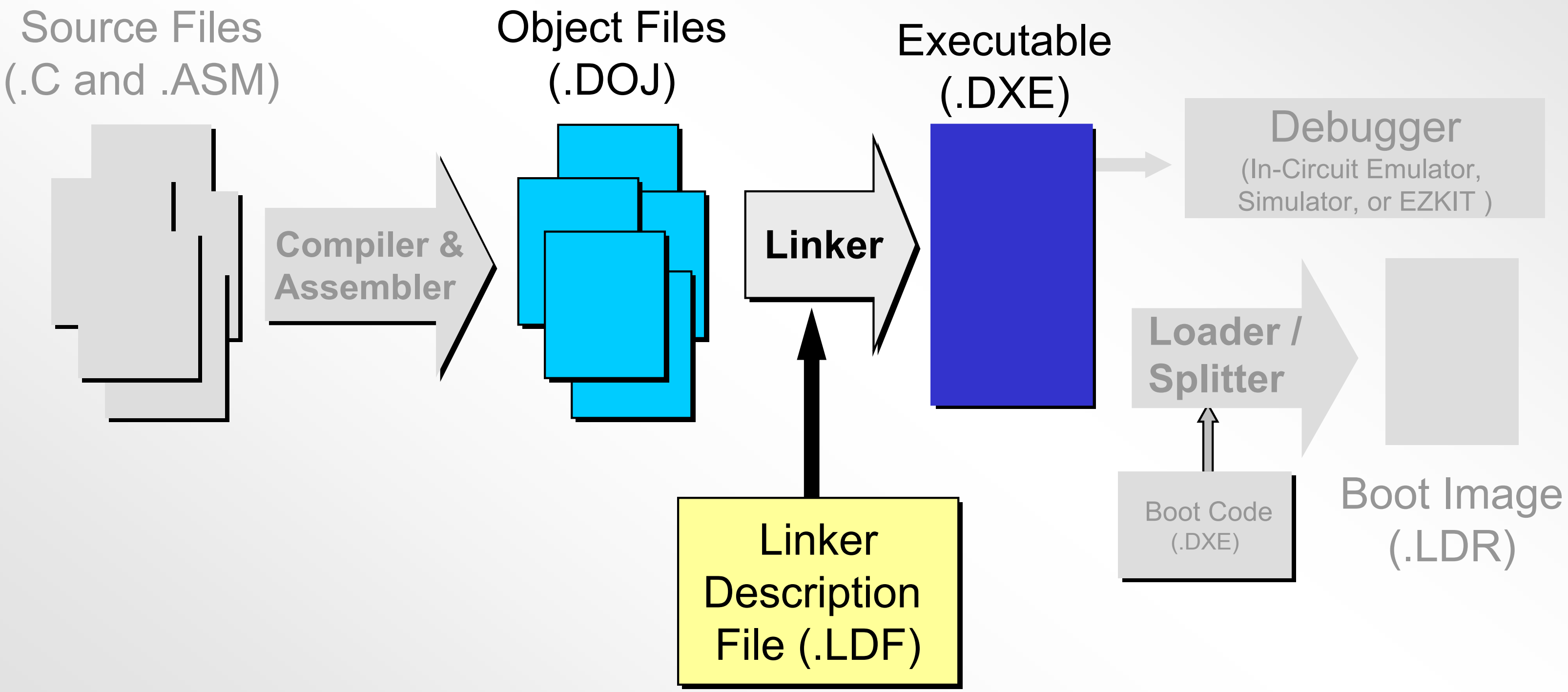
Post Link results indicate how much memory was actually used



Results of profiling indicate which objects use more CPU time

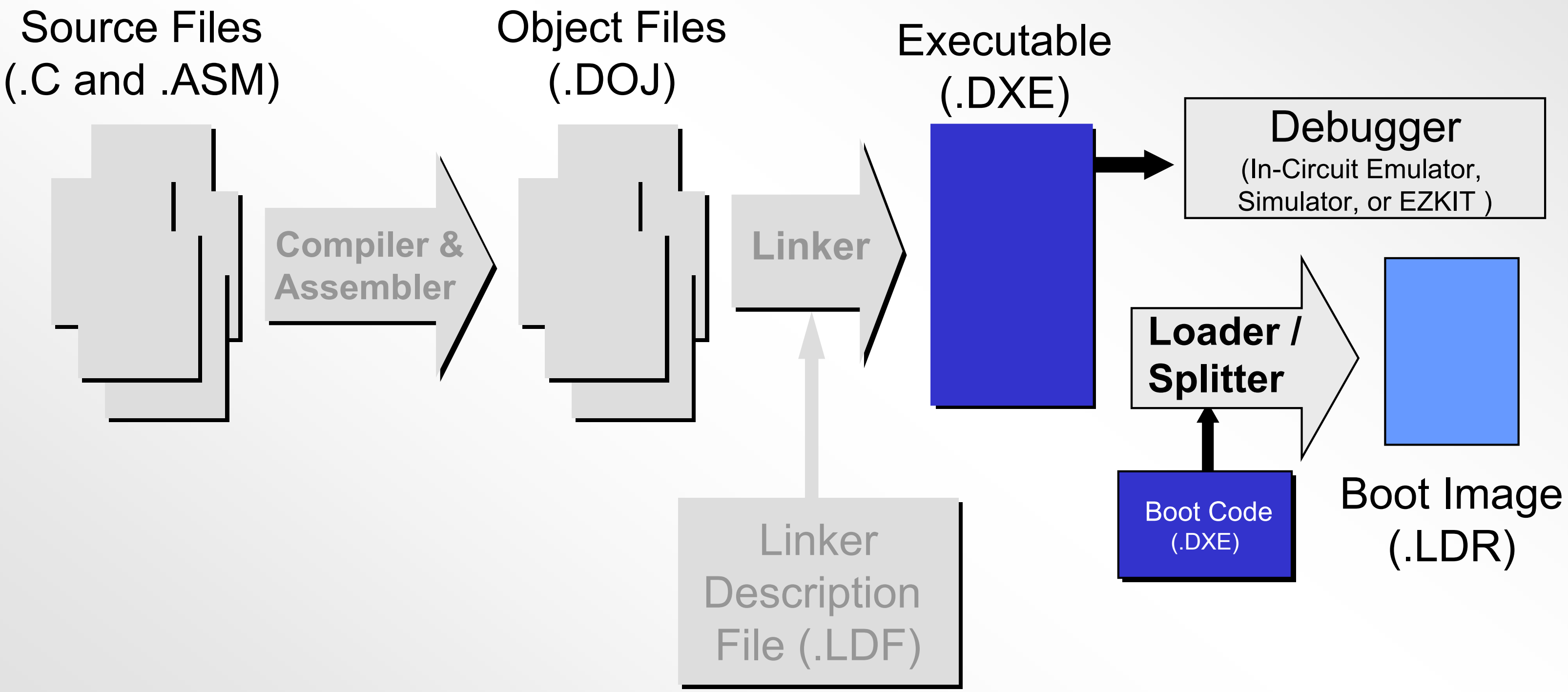
Software Development Flow

Step 2 - Linking



Software Development Flow

Step Three - Debugging



KAZTEK ENGINEERING

ANALOG DEVICES

Debugger

Debugger Features

- Single step
- Run
- Halt
- Run to breakpoint
- Profiling
- Pipeline Viewer
- Cache Viewer
- Plotting
- Simulate Standard I/O, Interrupts and Streams
- Compiled simulation for faster simulation times
- Run To Main
- STDIO

Compiled Simulation

- Traditional simulator decodes/interprets one instruction at a time
  - large processing overhead during simulation
- With Compiled Simulation a Blackfin DXE file is “preprocessed” and converted into an executable for the system hosting VisualDSP++
  - processing overhead during simulation is drastically reduced
- Can be executed
  - in VisualDSP++ using debug features (breakpoints, single step, displaying registers and memory, etc)
  - “stand-alone” without VisualDSP++ using streams and file input/output

VisualDSP++ Debug Control

- Breakpoints
  - Symbol
  - Address
- Conditional Breakpoints (“watchpoints”) [Simulation Only]
  - Register
    - Any Read or Write
    - Read or Write of an undefined value
    - Read or Write of a specific value.
  - Memory Ranges
    - Any Read or Write
    - Read or Write of an undefined value
    - Read or Write of a specific value

VisualDSP++ Debug Control

- Single Step (Step into)
  - Step through the program one instruction at a time
- Step Out Of, Step Over
  - Used when debugging C Code
- External Interrupts
  - Set number of instruction cycles between interrupts
  - Random Interval possible
- Stream I/O
  - Used to simulate IO, serial ports and parallel ports
  - Assign data-files as source/destination

VisualDSP++ Debugger Windows

- Disassembly Window
  - View disassembled assembly code
- Source Window
  - C, Mixed C/Assembly
- Local Window
  - Displays all local variables within current function
- Expressions Window
  - Any “C” expression
  - Register names preceded by a $ (for example $R12)
- Profile Window
  - Cycle-Count & Percentage of time spent executing in specified address ranges
- Plot
  - Enhanced plot capability

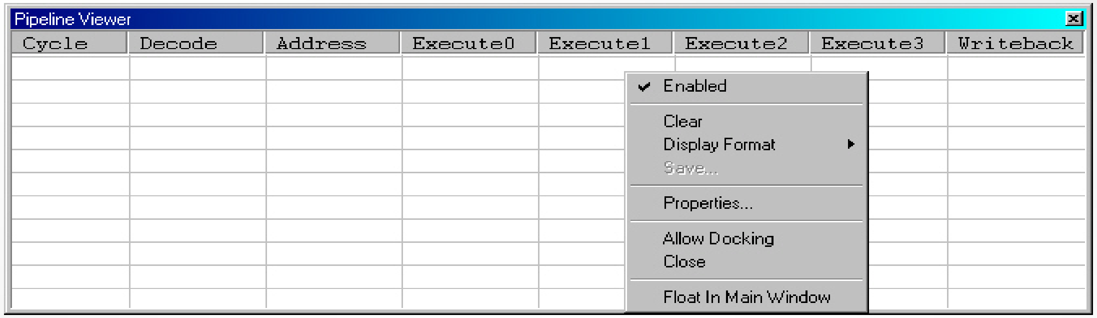
Run to Main & STDIO

- Run To Main
  - **Allows the user to control whether or not the debugger, on a load, starts execution in the run time header or at the first line in main().**

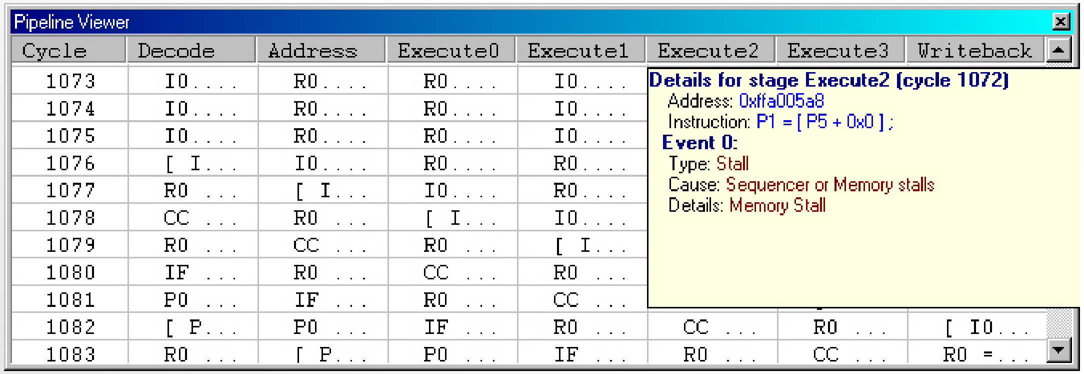
- STDIO
  - **Full STDIO support. Use printf() and scanf() to access files on the host system.**

Using the Pipeline Viewer

- **Accessed through** `View->Debug Windows->Pipeline Viewer` **in a simulator session (not available in emulator)**
- **Enabled through the context menu**



- **Place the cursor on a stall and press CTRL key to see more info about it**

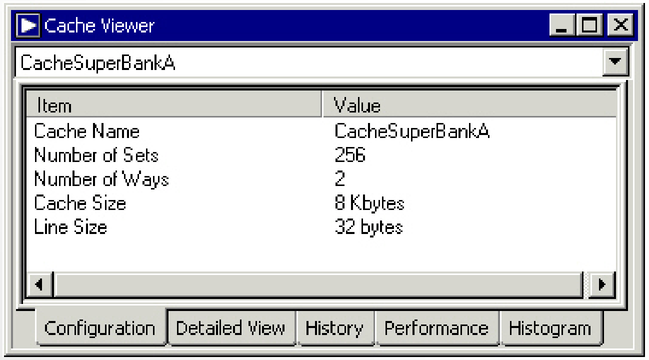


Cycle	Decode	Address	Execute0	Execute1	Execute2	Execute3	Writeback
1073	I0....	R0....	R0....	I0....			
1074	I0....	R0....	R0....	I0....			
1075	I0....	R0....	R0....	I0....			
1076	[I...	I0....	R0....	R0....			
1077	R0 ...	[I...	I0....	R0....			
1078	CC ...	R0 ...	[I...	I0....			
1079	R0 ...	CC ...	R0 ...	[I...			
1080	IF ...	R0 ...	CC ...	R0 ...			
1081	P0 ...	IF ...	R0 ...	CC ...			
1082	[P...	P0 ...	IF ...	R0 ...	CC ...	R0 ...	[I0...
1083	R0 ...	[P...	P0 ...	IF ...	R0 ...	CC ...	R0 =...

Details for stage Execute2 (cycle 1072)
Address: 0xffa005a8
Instruction: P1 = [P5 + 0x0] ;
Event 0:
Type: Stall
Cause: Sequencer or Memory stalls
Details: Memory Stall

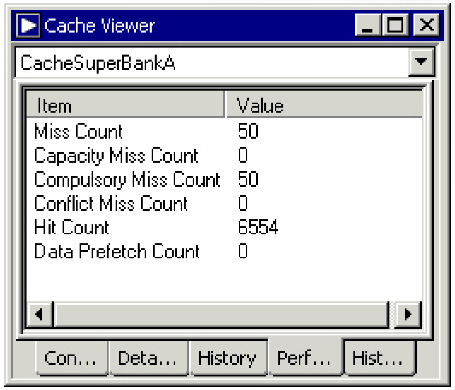
Using the Cache Viewer

- **Accessed through** `View->Debug Windows->Cache Viewer` **in a simulator session (not available in emulator)**
- **Enabled through the context menu**



Item	Value
Cache Name	CacheSuperBankA
Number of Sets	256
Number of Ways	2
Cache Size	8 Kbytes
Line Size	32 bytes

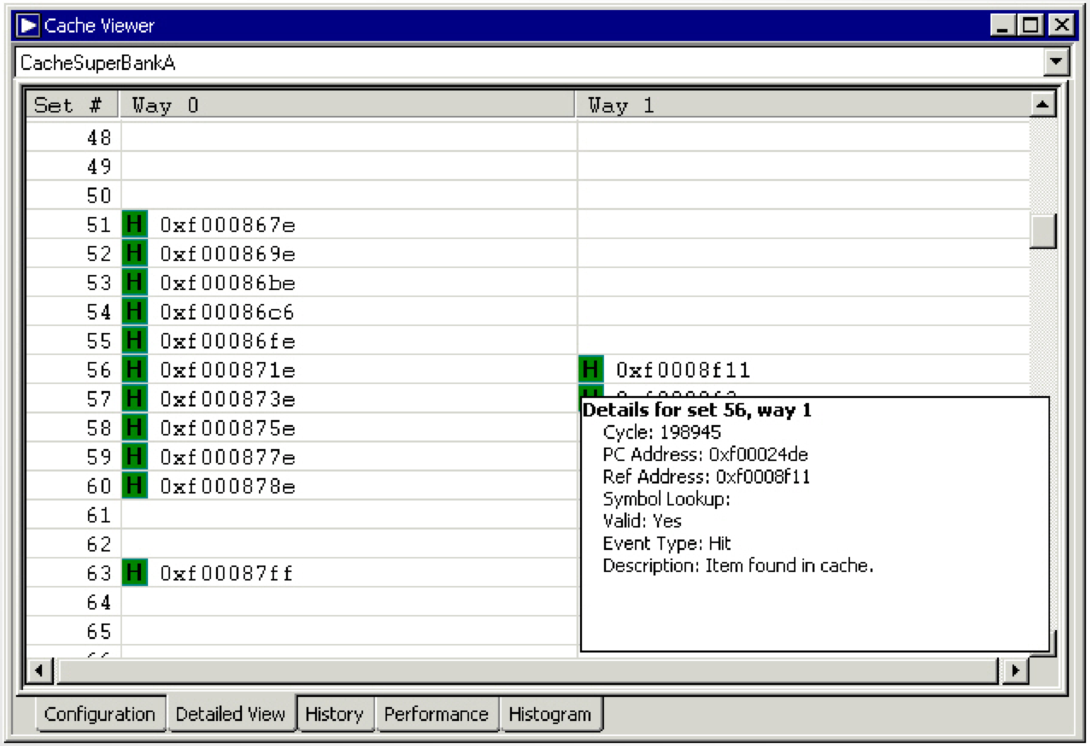
Provides information about the efficiency of the cache



Item	Value
Miss Count	50
Capacity Miss Count	0
Compulsory Miss Count	50
Conflict Miss Count	0
Hit Count	6554
Data Prefetch Count	0

Using the Cache Viewer

- **Place the cursor on a stall and press CTRL key to see more info about it**



Cache Viewer

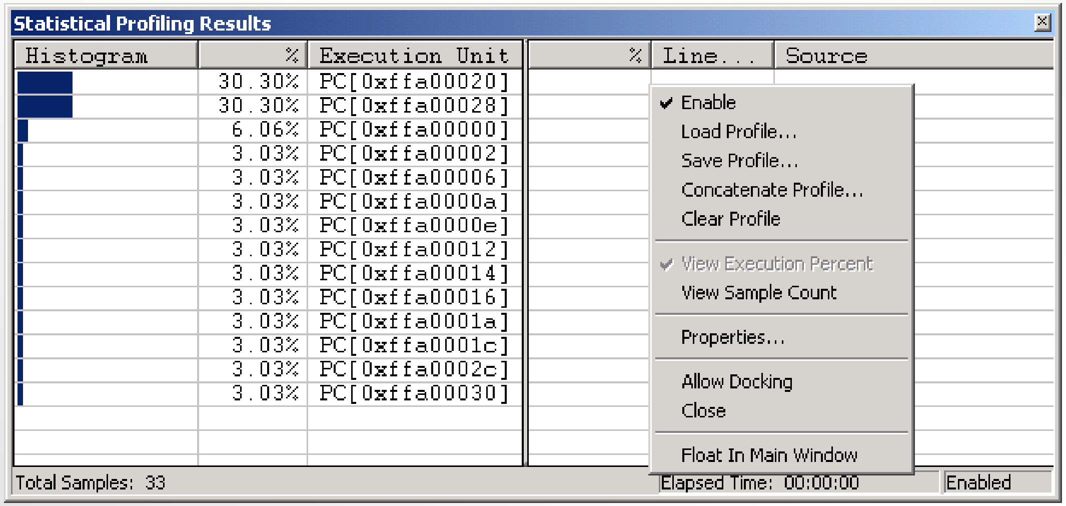
CacheSuperBankA

Set #	Way 0	Way 1
48		
49		
50		
51	H 0xf000867e	
52	H 0xf000869e	
53	H 0xf00086be	
54	H 0xf00086c6	
55	H 0xf00086fe	
56	H 0xf000871e	H 0xf0008f11
57	H 0xf000873e	H [illegible]
58	H 0xf000875e	
59	H 0xf000877e	
60	H 0xf000878e	
61		
62		
63	H 0xf00087ff	
64		
65		

Details for set 56, way 1
Cycle: 198945
PC Address: 0xf00024de
Ref Address: 0xf0008f11
Symbol Lookup:
Valid: Yes
Event Type: Hit
Description: Item found in cache.

Using Linear Profiling

- **Linear Profiling accessed through** `Tools->Linear Profiling->New Profile` **in a simulator session**
- **Enable the Linear Profiler through the context menu**
- **Single-step, or run and halt to update the results**



Statistical Profiling Results

Histogram	%	Execution Unit
	30.30%	PC[0xffa00020]
	30.30%	PC[0xffa00028]
	6.06%	PC[0xffa00000]
	3.03%	PC[0xffa00002]
	3.03%	PC[0xffa00006]
	3.03%	PC[0xffa0000a]
	3.03%	PC[0xffa0000e]
	3.03%	PC[0xffa00012]
	3.03%	PC[0xffa00014]
	3.03%	PC[0xffa00016]
	3.03%	PC[0xffa0001a]
	3.03%	PC[0xffa0001c]
	3.03%	PC[0xffa0002c]
	3.03%	PC[0xffa00030]

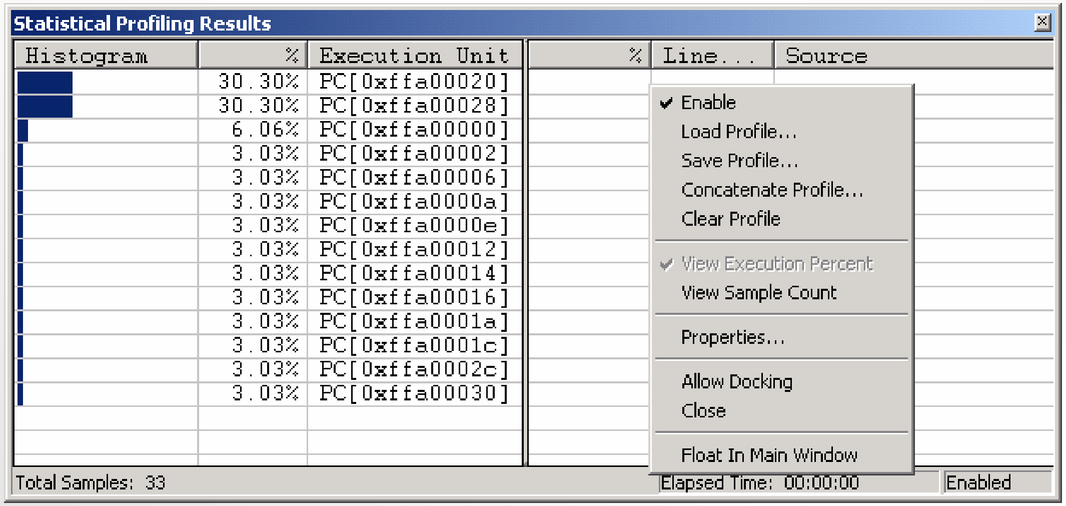
Total Samples: 33

Elapsed Time: 00:00:00

Enabled

Using Statistical Profiling

- **Statistical Profiling accessed through** `Tools->Statistical Profiling->New Profile` **in an emulator session**
- **Enable the Statistical Profiler through the context menu**
- **Run and watch as the results are updated in real-time; Halting keeps the last snapshot on the screen**



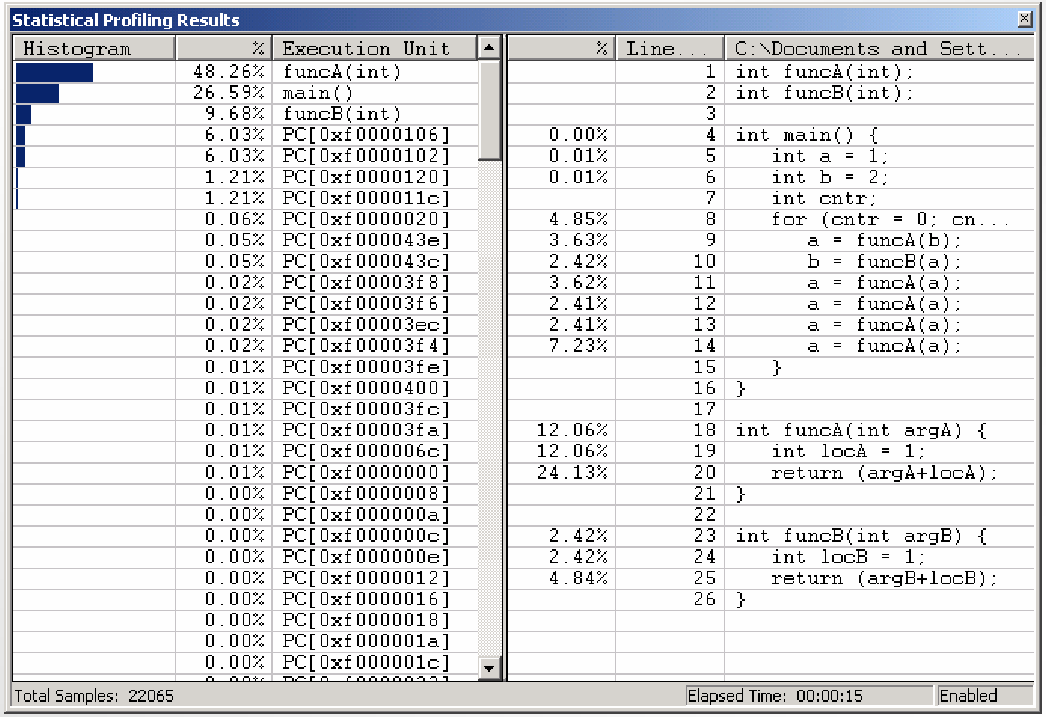
Histogram	%	Execution Unit
	30.30%	PC[0xffa00020]
	30.30%	PC[0xffa00028]
	6.06%	PC[0xffa00000]
	3.03%	PC[0xffa00002]
	3.03%	PC[0xffa00006]
	3.03%	PC[0xffa0000a]
	3.03%	PC[0xffa0000e]
	3.03%	PC[0xffa00012]
	3.03%	PC[0xffa00014]
	3.03%	PC[0xffa00016]
	3.03%	PC[0xffa0001a]
	3.03%	PC[0xffa0001c]
	3.03%	PC[0xffa0002c]
	3.03%	PC[0xffa00030]

%	Line...	Source

Total Samples: 33 Elapsed Time: 00:00:00 Enabled

C/C++ Profiler

- The profiler is very useful in C/C++ mode because it makes it easy to benchmark a system on a function-by-function (i.e. C/C++ function) basis
  - Assembly modules can be wrapped in C/C++ functions to take advantage of this



Statistical Profiling Results

Histogram	%	Execution Unit
▇▇▇▇▇	48.26%	funcA(int)
▇▇▇	26.59%	main()
▇	9.68%	funcB(int)
▍	6.03%	PC[0xf0000106]
▍	6.03%	PC[0xf0000102]
▏	1.21%	PC[0xf0000120]
▏	1.21%	PC[0xf000011c]
	0.06%	PC[0xf0000020]
	0.05%	PC[0xf000043e]
	0.05%	PC[0xf000043c]
	0.02%	PC[0xf00003f8]
	0.02%	PC[0xf00003f6]
	0.02%	PC[0xf00003ec]
	0.02%	PC[0xf00003f4]
	0.01%	PC[0xf00003fe]
	0.01%	PC[0xf0000400]
	0.01%	PC[0xf00003fc]
	0.01%	PC[0xf00003fa]
	0.01%	PC[0xf000006c]
	0.01%	PC[0xf0000000]
	0.00%	PC[0xf0000008]
	0.00%	PC[0xf000000a]
	0.00%	PC[0xf000000c]
	0.00%	PC[0xf000000e]
	0.00%	PC[0xf0000012]
	0.00%	PC[0xf0000016]
	0.00%	PC[0xf0000018]
	0.00%	PC[0xf000001a]
	0.00%	PC[0xf000001c]

%	Line...	C:\Documents and Sett...
	1	int funcA(int);
	2	int funcB(int);
	3	
0.00%	4	int main() {
0.01%	5	int a = 1;
0.01%	6	int b = 2;
	7	int cntr;
4.85%	8	for (cntr = 0; cn...
3.63%	9	a = funcA(b);
2.42%	10	b = funcB(a);
3.62%	11	a = funcA(a);
2.41%	12	a = funcA(a);
2.41%	13	a = funcA(a);
7.23%	14	a = funcA(a);
	15	}
	16	}
	17	
12.06%	18	int funcA(int argA) {
12.06%	19	int locA = 1;
24.13%	20	return (argA+locA);
	21	}
	22	
2.42%	23	int funcB(int argB) {
2.42%	24	int locB = 1;
4.84%	25	return (argB+locB);
	26	}

Total Samples: 22065 Elapsed Time: 00:00:15 Enabled

Programming Exercise #1

Lab 7

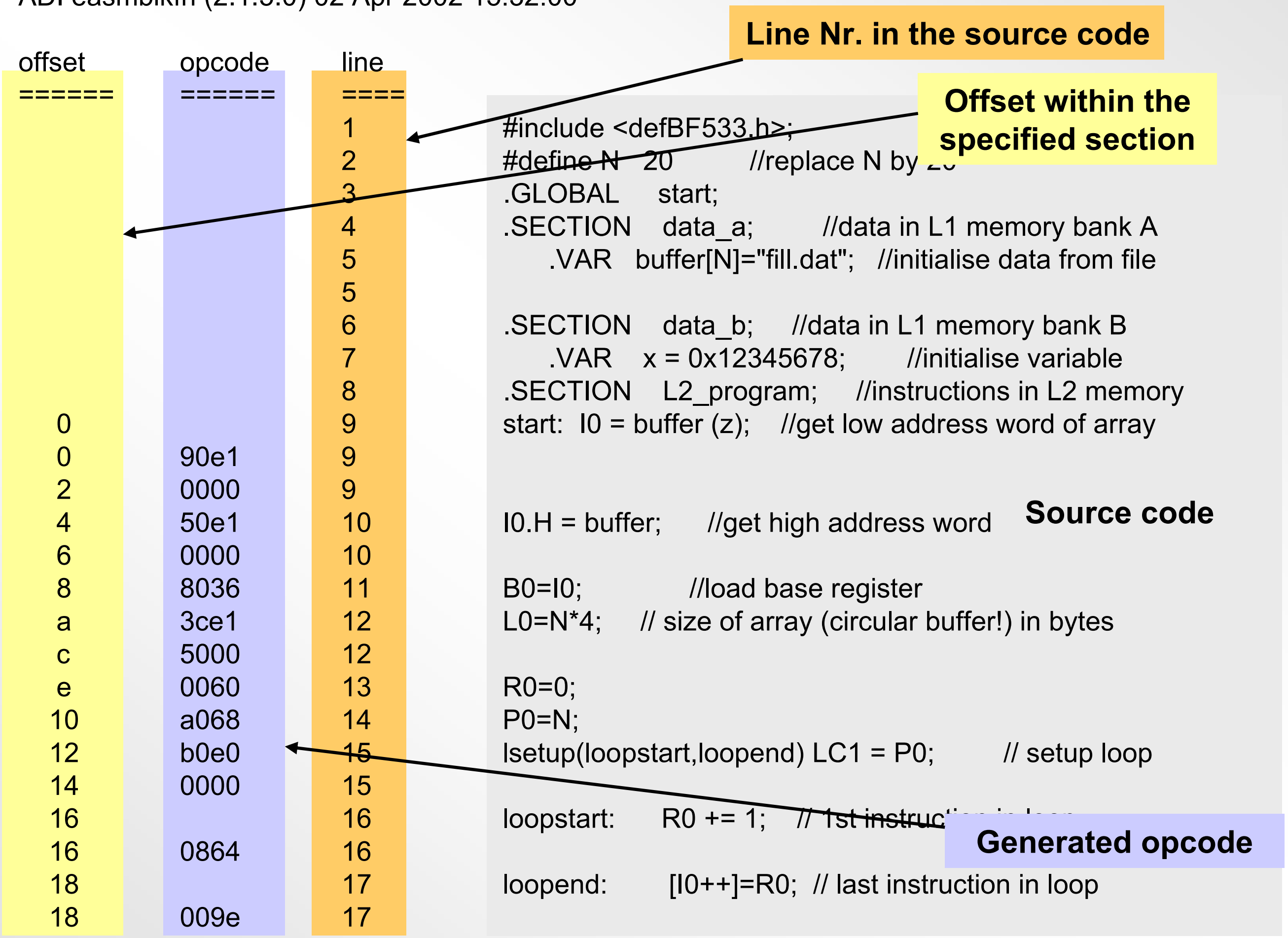
Reference Material

Code Development

Read The ReadMe Files!

Upgrades/Documentation/Tool Anomalies available at:
http://www.analog.com

Listing file (.lst)



```
Page 1  .\test.asm
ADI easmblkfn (2.1.5.0) 02 Apr 2002 15:32:00

offset    opcode    line
======    ======    ====
                    1     #include <defBF533.h>;
                    2     #define N   20        //replace N by 20
                    3     .GLOBAL    start;
                    4     .SECTION   data_a;       //data in L1 memory bank A
                    5          .VAR   buffer[N]="fill.dat";   //initialise data from file
                    5
                    6     .SECTION   data_b;     //data in L1 memory bank B
                    7          .VAR    x = 0x12345678;        //initialise variable
                    8     .SECTION   L2_program;     //instructions in L2 memory
  0                 9     start:  I0 = buffer (z);    //get low address word of array
  0       90e1      9
  2       0000      9
  4       50e1      10    I0.H = buffer;      //get high address word
  6       0000      10
  8       8036      11    B0=I0;              //load base register
  a       3ce1      12    L0=N*4;     // size of array (circular buffer!) in bytes
  c       5000      12
  e       0060      13    R0=0;
  10      a068      14    P0=N;
  12      b0e0      15    lsetup(loopstart,loopend) LC1 = P0;        // setup loop
  14      0000      15
  16                16    loopstart:      R0 += 1;    // 1st instruction in loop
  16      0864      16
  18                17    loopend:        [I0++]=R0;  // last instruction in loop
  18      009e      17
```

Line Nr. in the source code

Offset within the specified section

Source code

Generated opcode

Example Global Commands

```
ARCHITECTURE (ADSP-BF533)
   // Processor Used

SEARCH_DIR( $ADI_DSP\Blackfin\lib )
   // Directories to search for files

$OBJECTS = bootup.doj, $COMMAND_LINE_OBJECTS;
   // Macro listing all command line objects and bootup
```

Linker Description File Macros

- **$COMMAND_LINE_OBJECTS:**
  List of objects (.DOJ) and libraries (.DLB) passed on command line.

- **$COMMAND_LINE_OUTPUT_FILE:**
  Output executable file name specified on the command line with the -o switch.

- **$ADI_DSP:** Path to VisualDSP installation directory.

- **$macro:** User defined macro for a list of files.
  e.g.: $OBJECTS