

Embest IDE Pro for ARM 2004

User Guide



Shenzhen Embest Info&Tech Co.,LTD. All rights reserved.

Tel: +86-755-25635626/25635656/25638952/25638953 Fax: +86-755-25616057

Room 210, Luohu Science&Technology Building, #85 Taining Road,

Shenzhen, Guangdong, China

Preface

This preface introduces the Embest Integrated Development Environment (EmbestIDE) and its documentation. It contains the following sections:

- About this book
- Related Publications
- Typographical convention
- Feedback

About this book

This book is user manual for EmbestIDE for ARM. It describes the major features of EmbestIDE, installing EmbestIDE for ARM, graphical user interface components of EmbestIDE, and provides information on debugging applications with EmbestIDE.

This book is organized into the following chapters:

Chapter 1 [Overview](#)

Read this chapter for an introduction to EmbestIDE.

Chapter 2 [Installing](#) EmbestIDE for ARM

Details for installing EmbestIDE, registering, files and folder structures of EmbestIDE.

Chapter 3 [Project examples](#)

Give some examples to lead users to start with EmbestIDE quickly

Chapter 4 [Editor](#)

Read this chapter for details about how to use EmbestIDE built-in text editor. It describes the basic functionality of the editor.

Chapter 5 [Project management](#)

Read this chapter for details about how to use project files to organize your project source files.

Chapter 6 [Project build](#)

Details about specify the output from compiling and linking your source. This chapter gives details about how to configure compile options and link options of a project.

Chapter 7 [Software debug](#)

Introduce the debugger of EmbestIDE, details about how to debug arm-based application with EmbestIDE.

Chapter 8 [Customization and Options](#)

Read this chapter for details about customizes EmbestIDE.

Appendix A [JTAG Emulator Hardware Reference](#)

Appendix B [Debug Output Reference](#)

Appendix C [Debug Command List](#)

Appendix D [Memory Map File](#)

Appendix E [Command Script Reference](#)

Appendix F [Additional Software Tools of Embest IDE](#)

Appendix G [Common questions](#)

Further Reading

This book describes all the details about Embest IDE. Refer to the following books for information on other components of developing embedded application base on ARM processors:

- ARM Architectural Reference Manual (ARM DUI 0100)
- ARM Reference Peripheral Specification (ARM DDI 0062)
- ARM Target Development System User Guide (ARM DUI 0061)

---you can get these papers from: www.arm.com

- Program reference of Embest IDE

Typographical Convention

The following typographical convention are used in this book:

Menu quote Use > to separate the main menu and submenu

Commands Highlights important notes

Notes Italic denotative arguments with lines above and below

Notes: this is an important note, and please pay attention to it.

Feed Back on This Book

If you have any problems with this book, please send email to press@embedinfo.com giving:

- ◇ the document title
- ◇ the page number(s) to which your comments apply
- ◇ a concise explanation of the problem.

General suggestions for additions and improvements are also welcome.

1. Overview

This chapter introduces the Embest IDE. It contains the following sections:

- ◇ About the Embest IDE
- ◇ main characteristic of Embest IDE
- ◇ User Interface Basics

1.1 About Embest IDE

Embest IDE is an Integrated Development Environment for software cross-development. The EmbestIDE is an application that provides a simple and versatile graphical user interface and tools for developing embedded software. It is an Integrated Development Environment (IDE) that facilitates managing and building projects, establishing and managing host-target communication, running and debugging applications. It provides an efficient way for developing embedded applications. EmbestIDE comprises the following elements (Figure 1-1):

- An integrated source-code editor.
- A project management facility.
- Integrated C and ASM compilers and linker.
- a source-level debugger.
- an integrated development environment.
- an ARM simulator

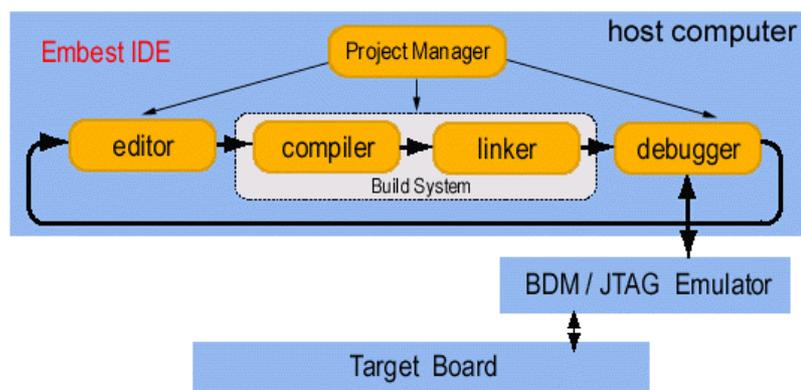


Figure 1-1: EmbestIDE Tools

EmbestIDE uses host-target cross model for developing embedded applications. EmbestIDE runs on a host computer. Your target board is connected to the host computer through debug device (BDM/JTAG Emulator). You can edit and build your projects on the host, and create a target executable file with EmbestIDE. Download the target file to the target, and then use EmbestIDE debugger to debug it through the communication of debug device

connected between the host computer and the target board. Figure 1-2 illustrates the host-target cross model with EmbestIDE.

When you use EmbestIDE for ARM, a debug device--Embest JTAG Emulator is provided. See appendix A "JTAG Emulator Connection" for detail describe about Embest JTAG Emulator.

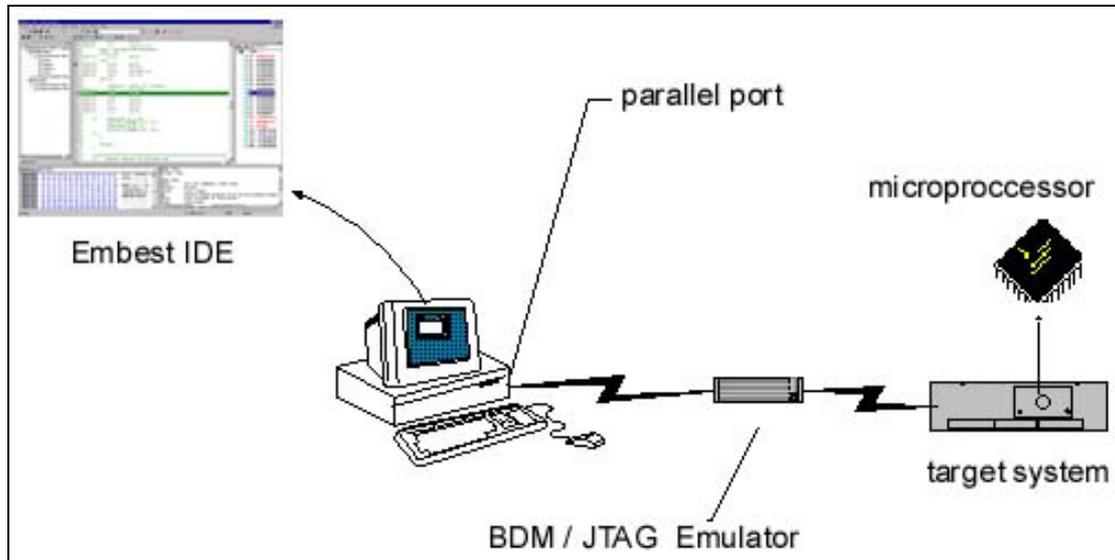


Figure 1-2 the Host-target Cross Model with Embest IDE

1.2 Major Features of Embest IDE

EmbestIDE runs under Windows 95 above and Windows NT 4.0 above. EmbestIDE supports target microprocessors debugging on the ARM-core microprocessor families, 68K, PowerPC, and Coldfire microprocessor families. EmbestIDE provides a simple, versatile and graphical user interface for managing your software development projects. You can use EmbestIDE for ARM to develop C and ARM assembly language code targeted at ARM and Thumb processors. It speeds up your development cycle by providing:

- Comprehensive project management capabilities
- Code navigation routines to help you locate routines quickly.

EmbestIDE enables you to organize source code files, library files, other files, and configuration settings into a project. Each project enables you to create and manage multiple configurations of build target settings.

Major features of EmbestIDE:

◆ **Support development language:** ANSI C, ARM assembly language.

◆ **User interface:** consists of an integrated set of windows, tools, menus, toolbars, directories, and other elements that allow you to create, test, and refine your application. It is just like Microsoft Visual Studio's user interface.

◆ **Source code Editor:** Standard text manipulation capabilities; C and ASM syntax-element color highlight; Debugger integration, the editor window tracks code execution; Compiler integration, the project management utility links compiler warnings and errors directly to the affected source in the editor window. Support file print; Capable of search and replace, and performs batch searches in multiple files.

◆ **Project Management:** The EmbestIDE project facility simplifies organizing, configuring, and building embedded applications. It includes graphical configuration of the build environment (including compiler flags).

◆ **Compiler:** includes the GNU compiler, as well as a collection of supporting tools that provide a complete development tool chain: cpp, C preprocessor; gcc, C compiler; make, program building automation tool; ld, programmable static linker; as, portable assembler; binary utilities. EmbestIDE supports commercial versions of the leading-edge GNU tools originally

developed by the Free Software Foundation (FSF). Users of the GNU tools benefit from the innovative FSF development environment as well as from testing and support by Embest Info&Tech Co., LTD. Among other features, the EmbestIDE project facility provides a GUI for the GNU tools that is powerful and easy to use.

◆ **EmbestIDE debugger:** a powerful graphical debugger that enables program loading, executing, running control, and monitoring; a source-level debugger, view your application code as C, as assembly-level code, or in a mixed mode that shows both; full-featured debugging, provides an exhaustive set of debugging features, designed to make it easy to find and fix bugs; set a breakpoint or clear a breakpoint by single click, supports conditional and command breakpoints; single stepping, "step into" traces execution of every individual instruction even when functions are called, "step over" does not trace into the called function, "step out " brings execution back to the calling function; supports register and variables value display and modify, supports function stack display and memory display, several specialized windows display these debugging information. Figure 7-1 shows these debugging information display windows, when you are debugging, you can access these windows using the "View" menu; EmbestIDE supports graphical debug and command-line debug. For complex or unpredictable debugging needs, the command-line interface gives you full access to a wealth of specialized debugging commands.

1.3 User Interface of EmbestIDE

The user interface is the portion of the environment where display information and specify action. These topics describe user interface's basic structure of EmbestIDE.

To step into EmbestIDE, just run EmbestIDE.exe. EmbestIDE user interface consists of an integrated set of windows, tools, menus, toolbars, directories, and other elements that allow you to create, test, and refine your application. Figure 1-3 shows the main GUI of EmbestIDE. The user interface uses standard Windows interface functionality along with a few additional features to make your development environment easy to use. The basic features that you use most often are windows and document views, toolbars, menus, directories, and keyboard shortcuts.

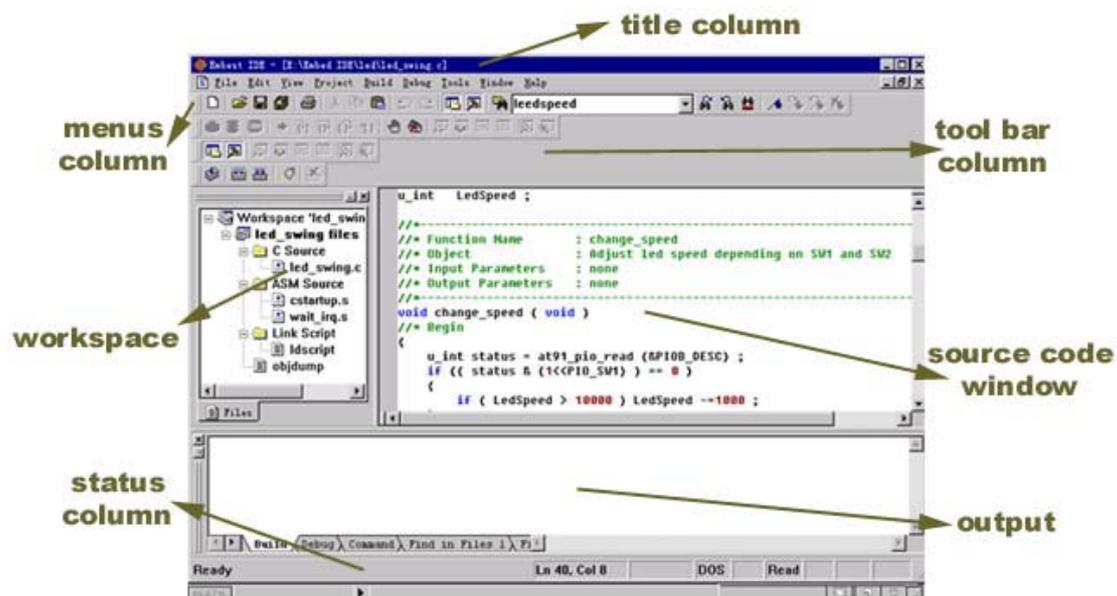


Figure 1-3 Embest IDE Main GUI

Title column displays current opening file name. Menus column and tool bar column are below title column. The menu bar is a special toolbar at the top of the screen that contains menus such as File, Edit, and Build. The standard toolbar appears just below the menu bar. You can move the toolbars to different locations to suit your needs. Workspace window shows file information about current opening projects. Source code window is the usual used window with which you edit and browse your code in. You can open several source code file windows at one time. Output window displays build information, debug

information, file search output information and command-line debug input and output. Status column displays detail information about menus and tool bars, it also displays the current line number and column number of the cursor in source code window.

2. Installation

2.1 System Requirement

| | |
|-------------------|--|
| Host | A PC with a Pentium or higher processor |
| Memory | More than 64M |
| Disk Space | 150M |
| Monitor | VGA or Super VGA color monitor |
| OS | Microsoft Windows 98, Windows NT with Service Pack 3 or later, Windows 2000, Windows Me, Windows XP |
| Others | Mouse, Parallel Port, CD-ROM |

2.2 Installation

Step 1: Invoke Microsoft Windows operating system on your PC;

Step 2: Place the EmbestIDE disk in the proper drive, and run Setup.exe;

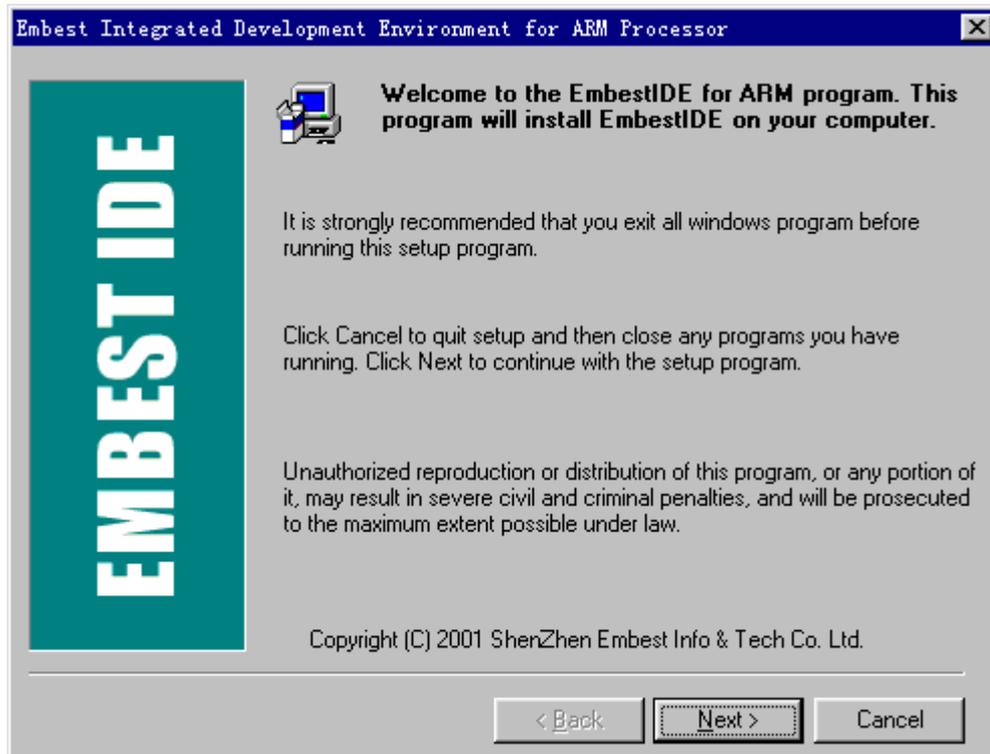


Figure 2-1 Welcome Dialog

Step 3: Click on "Next" button in welcom dialog to go on;

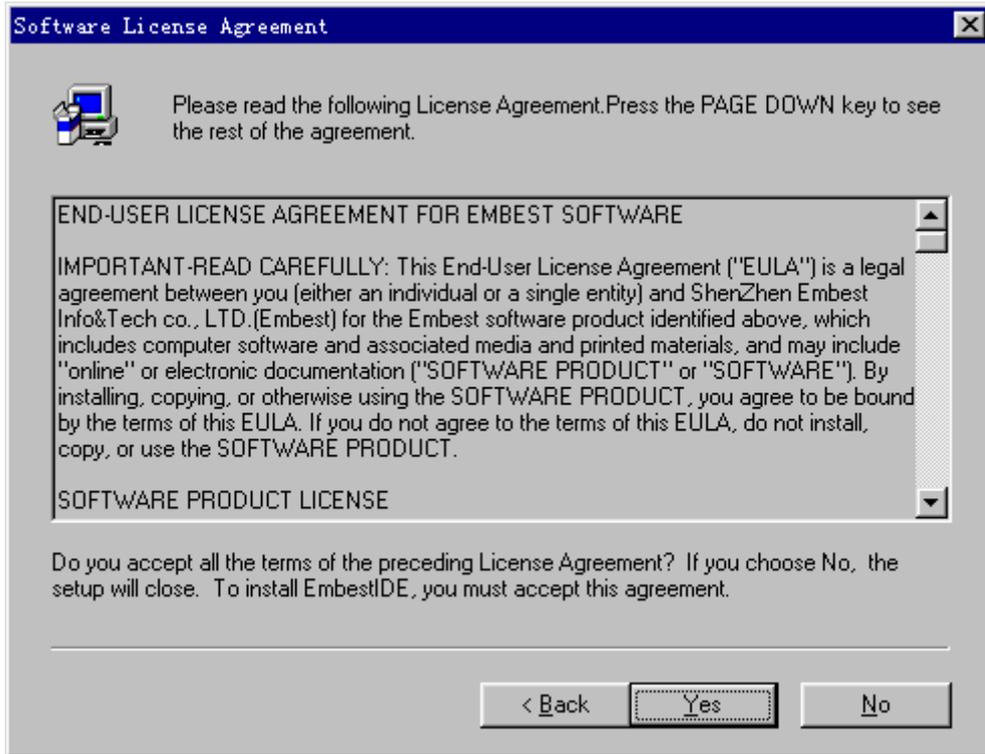


Figure 2-2 License Dialog

Step 4: Click on "Yes" button in License dialog to go on, if you accept the license agreement list in the dialog;

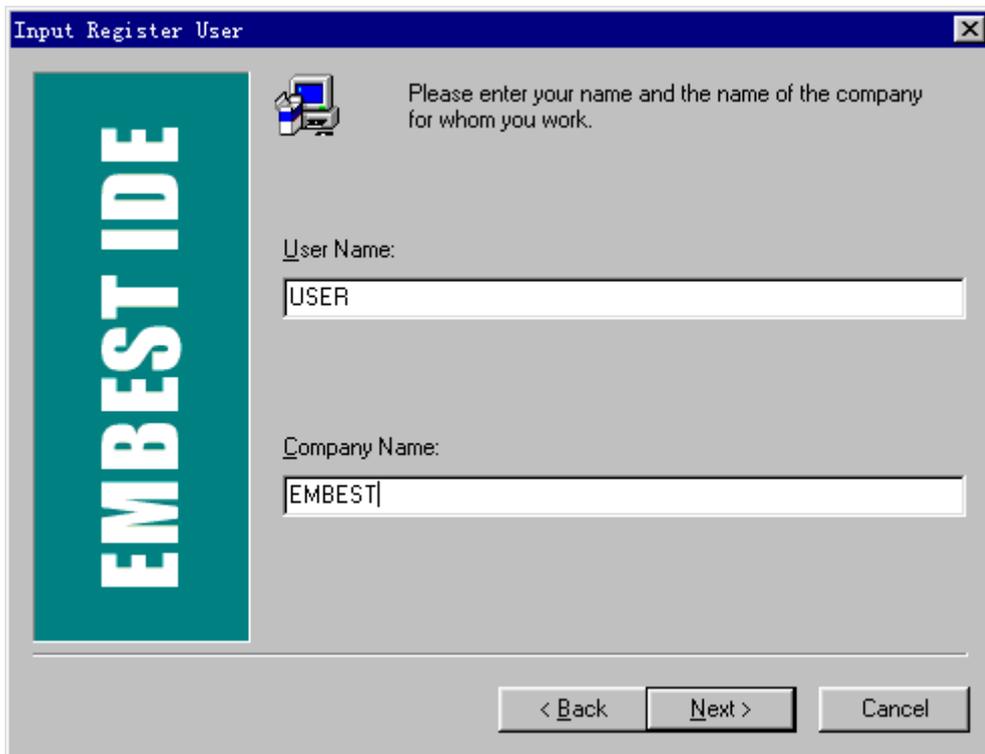


Figure 2-3 User Information Dialog

Step 5: Click on "Next" button in User Information Dialog to keep going; after the information is inputted;

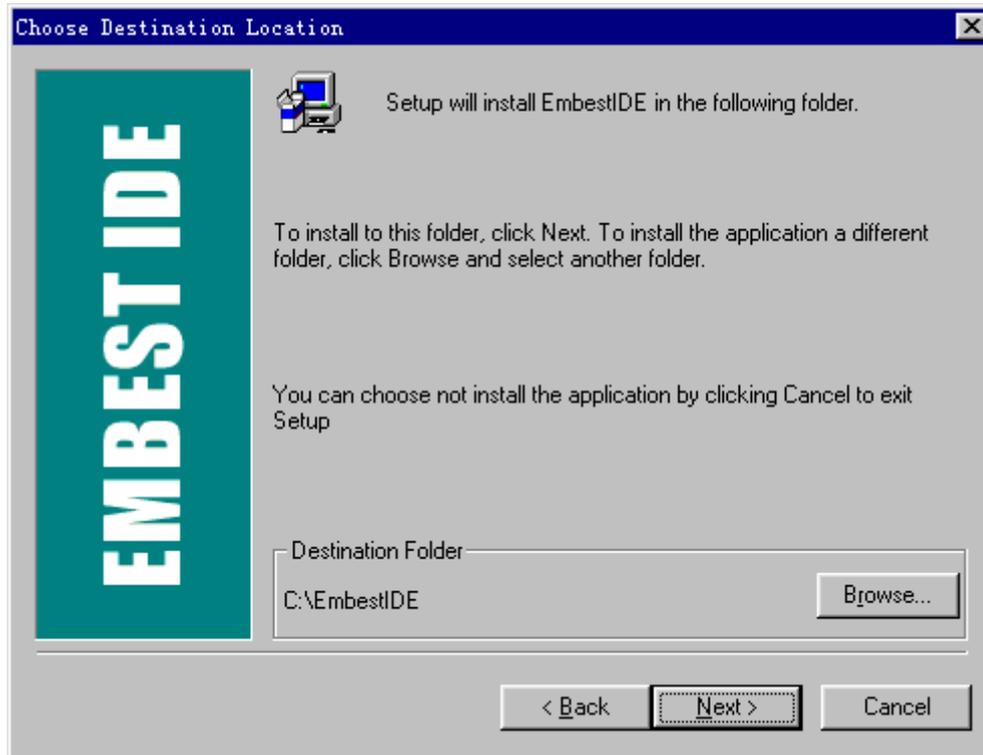


Figure 2-4 Destination Location Dialog

Step 6: After setup program is loaded, you will need to indicate the directory where you want to install EmbestIDE. By default, it will be installed in the following directory:

C:\EmbestIDE (where, C:\ is the system driver)

Click on the "OK" button if the above default directory is the location where you want to install EmbestIDE. We recommend that you use the default **C:\EmbestIDE** directory so that the changes you will need to make are minimal. Otherwise, you can edit the path as necessary. Then click on the "Next" button to go on.

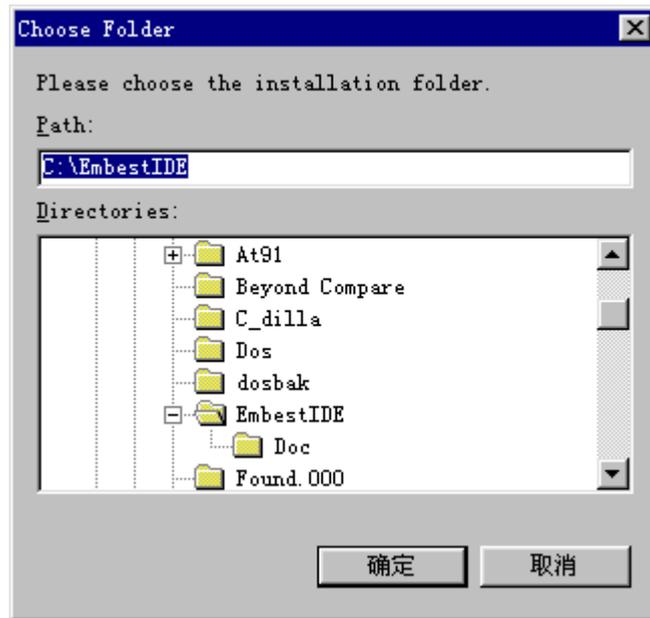


Figure 2-5 Choose Destination Folder Dialog

Note: The full path name of the destination location must not contain the blank character, because of the GNU Cross-Compiler.

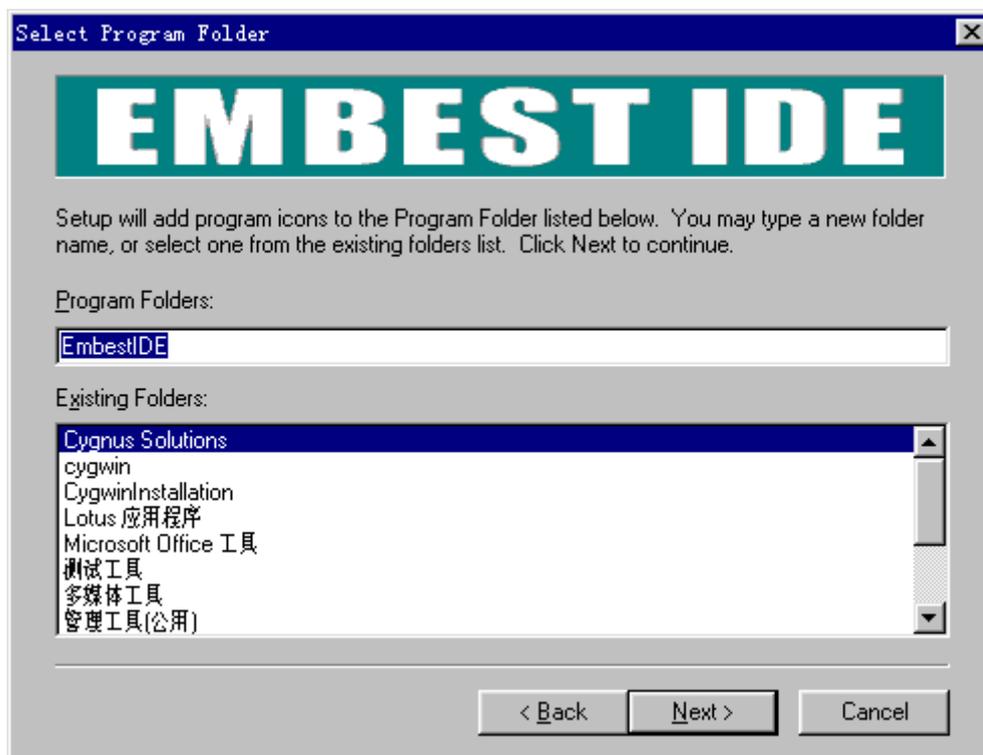


Figure 2-6 Program Folder Dialog

Step 7: click on the "Next" button to go on, in the Program Folder dialog;

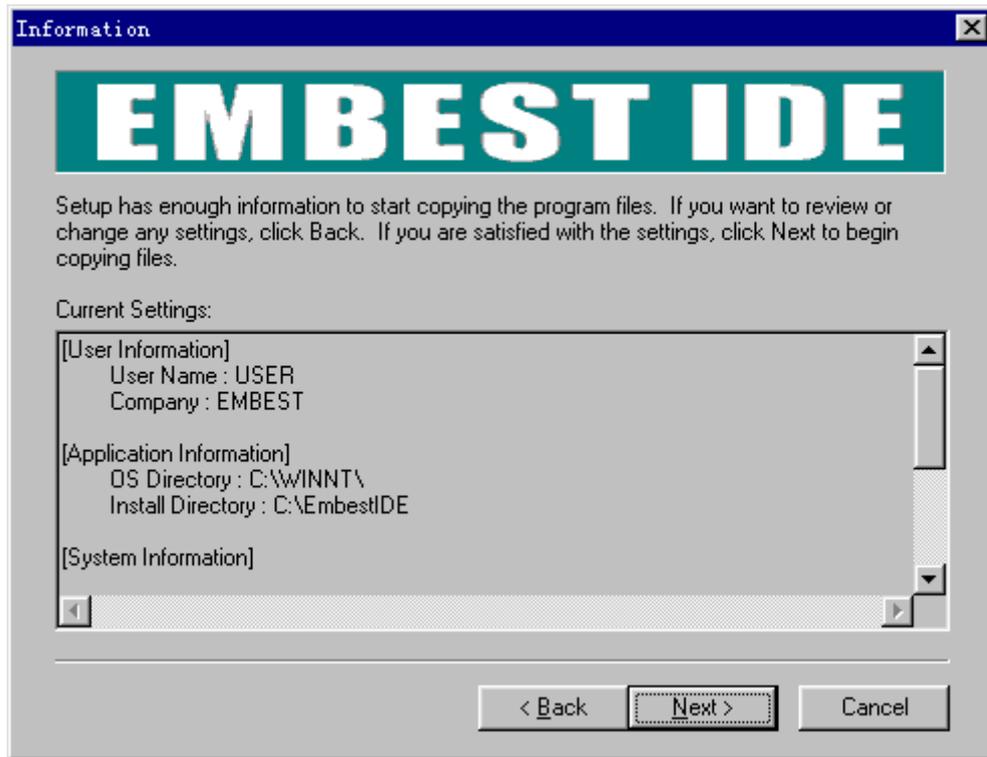


Figure 2-7 Information Dialog

Step 8: Click on the "Next" button in the Information dialog, the setup program will copy and decompress the files to the installation directory.

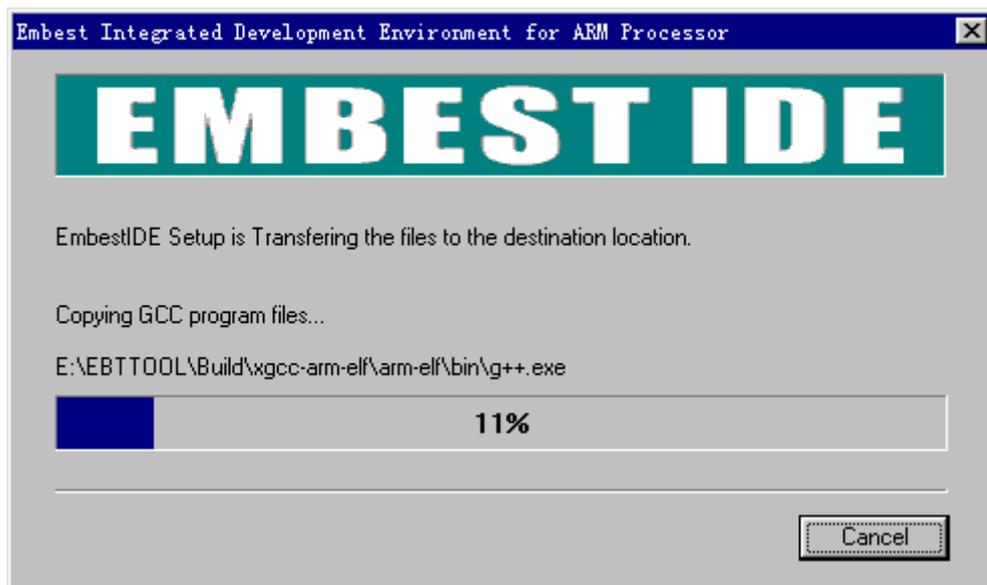


Figure 2-8 File Copying Status Dialog

Step 9: After the files copying and decompression is completed, setup program will ask if you want to restart you computer right now or not.

Before running EmbestIDE, You must reboot the host computer and get a runtime permit file.

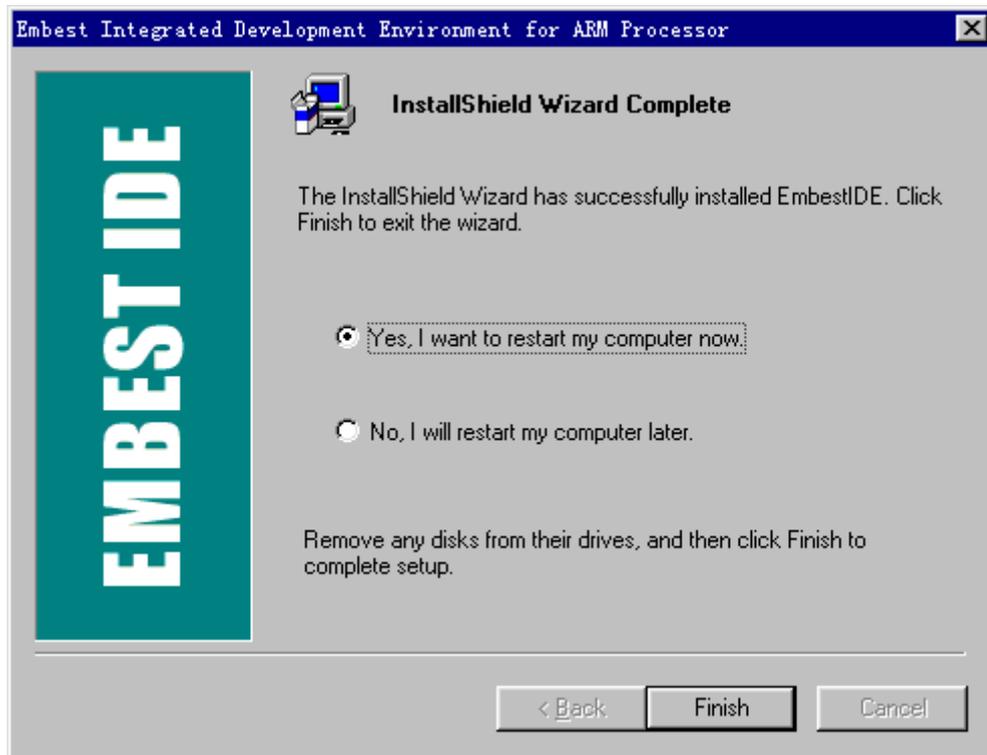


Figure 2-9 Installation Finish Dialog

2.3 Directories & Files

2.3.1 Directories & Files

The files of Embest IDE release are copied to a directory and its subfolders, when the [installation](#) steps finished. For example, if the destination location is **C:\EmbestIDE**, several sub-directories will be created on it, and the files will be copied according their functions (show as Table 2-3-1-1) .

Table 2-3-1-1

| Folder/File | Description |
|-------------|--|
| Bin | Folder with main application, DLL files and DLL folders |
| Build | Folder with GNU Cross-Compiler for ARM |
| Doc | Folder with documents |
| Examples | Folder with samples |
| ide.ini | Configuration files of Embest IDE, it shows the folder of the modules and other settings in running. |
| Licenses | Folder with license files |
| Peripheral | The information library of the peripheral registers of the processors set by the system |
| Target | Folder with BSP files |
| License.txt | License file |
| Readme.Txt | Readme file |
| Tools | Folder with the tools in system application include Flash Programmer, elf2bin etc. |

Description of directory C:\EmbestIDE\Bin show at Table 2-3-1-2.

Table 2-3-1-2

| Folder/File | Description |
|-------------|--|
| Build | Folder with cross-compiler interface files |
| CPU | Folder with CPU interface files |
| Device | Folder with device interface files |

Driver Folder with driver interface files

Description of directory C:\EmbestIDE\Examples Table 2-3-1-3

Table 2-3-1-3

| Folder/File | Description |
|--------------------|--|
| ARM | Command examples for ARM include particular assemble examples |
| AT91 | Examples for each evaluation board of the Atmel 91 processors . Include AT91EB40, AT91EB40A, AT91EB55, AT91EB63. |
| AT91/ucos | Example of ucos II operating system for AT91EB40 evaluation board. |
| Cirrus | Examples for Cirrus Logic ARM processor EP7312. |
| Oki | Examples for OKI ARM processor ML674000. |
| Samsung | Examples for Samsung evaluation board. Include NET-Start , 44bdvk,Nbc4510b,SNDS100 evaluation board. |
| Samsung/ucos | Example of ucos II operating system for 44bdvk evaluation board. |

2.3.2 File Types

Table 2-3-2-1

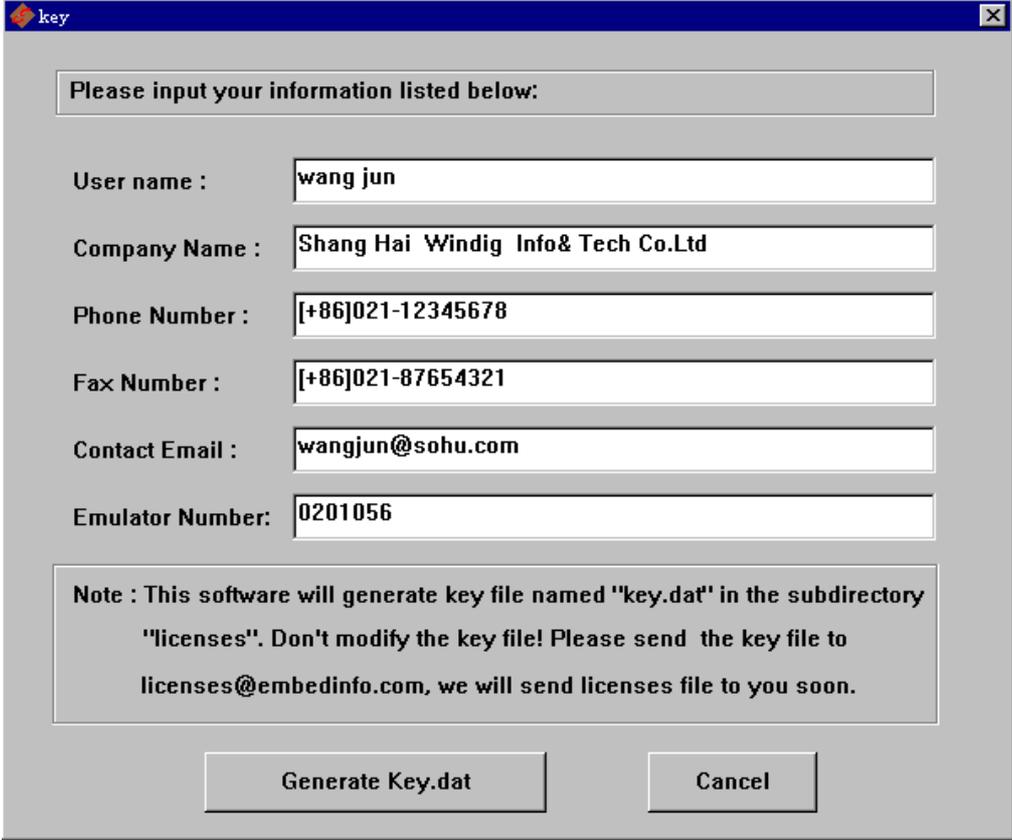
| Name | Description |
|-------------|-----------------------------|
| *.ews | Workspace file of EmbestIDE |
| *.pjf | Project file of EmbestIDE |
| *.opt | Workspace status file |
| *.ini | Config file of EmbestIDE |
| *.c | C source file |
| *.C | C++ source file |
| *.cc | C++ source file |
| *.cp | C++ source file |
| *.cxx | C++ source file |
| *.c++ | C++ source file |
| *.cpp | C++ source file |
| *.s | Assemble source file |
| *.asm | Assemble source file |
| *.h | C/C++ header file |
| *.inc | Assemble include file |
| *.mac | Macro file |
| *.map | Memory map file |
| *.cs | Command script file |
| *.ld | Link script file |
| *.o | Object file |
| *.a | Library file |

2.4 User Register

Registration needs two steps: 1.generate register information file; 2.submit register information file and acquire runtime permit file.

2.4.1 Generate Register Information File

Run "key.exe" program which locate at "\$Embest IDE/licenses" directory or "EmbestIDE.exe" program, fill correlative information and push "Generate Key.dat" button, "key.dat" file will be created in "\$Embest IDE/licenses" directory (Note: please do not modify the file). Program interface show as following figure 2-10:



The screenshot shows a dialog box titled "key" with a blue title bar. Inside, there is a prompt: "Please input your information listed below:". Below this are six input fields with labels: "User name : wang jun", "Company Name : Shang Hai Windig Info& Tech Co.Ltd", "Phone Number : [+86]021-12345678", "Fax Number : [+86]021-87654321", "Contact Email : wangjun@sohu.com", and "Emulator Number: 0201056". At the bottom, there is a note: "Note : This software will generate key file named 'key.dat' in the subdirectory 'licenses'. Don't modify the key file! Please send the key file to licenses@embedinfo.com, we will send licenses file to you soon." Below the note are two buttons: "Generate Key.dat" and "Cancel".

Figure 2-10 Key.dat Generation Dialog

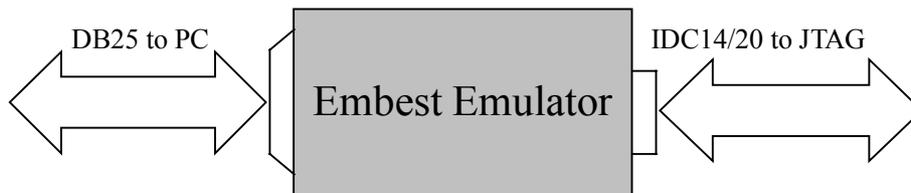
Action: Don't modify the key.dat file.

2.4.2 Acquire Runtime Permit File

Please send email of the "key.dat" file to licenses@embedinfo.com, or print and fax it to (+86)0755-25716057 ext 805, or copy to a floppy disk and send the disk to Embest Inc, user will get "licenses.dat" file soon by email or floppy disk. This file is the runtime permit file. Please copy it to "\$Embest IDE/licenses" directory, then the registration is finished.

2.5 Connecting the Emulator Hardware

Embest Emulator is a JTAG-based debugging channel for ARM microprocessors. It provides an interface between Embest IDE and an ARM microprocessor deeply embedded.



A standard male-to-female 25-way parallel cable connects the Embest Emulator to the PC's parallel port.

The connection to the target board is made by a 20-way (or 14 - way) female IDC header cable (BT224 type) with all pins connected straight through (1-1, 2-2, ... 20-20).

Note: Connecting cable dose not provide hot swap

For further pin-out details, refer to Embest IDE [JTAG interface connections](#) on page A-2.

3. Quick Start

3.1 Simple Example

This section shows users step by step how to create, compile and debug a simple project. The project described below is to create a random number.

The project demo locates at \Examples\arm\explasm under EmbestIDE installing directory. The files to be used are:

Random.s **Assemble file of random number function**

Randtest.c **Main program file**

Note: Only source files above are used. The project can only be run and debugged in target RAM under EmbestIDE. Parameters are established according ATMEL EB40 evaluation circuit board. The parameters for linking need to be change when the project is used with the other circuit board. The project doesn't link to any function library and doesn't use Linker scripts. The program created by the project doesn't have a function to initiate any circuit board. Therefore, it can't run itself when it is downloaded to FLASH.

3.1.1 Create a Project

1) Creating Project

Select **File > New Workspace** menu, Give a project name and specify project directory as shown in Figure 3-1:

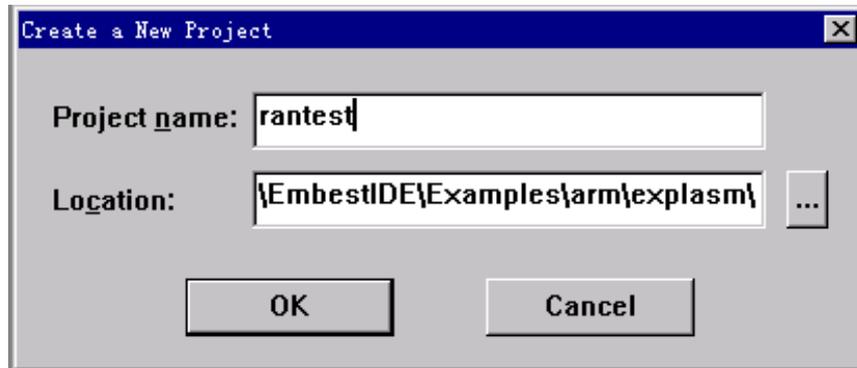


Figure 3-1 Create a New Project Dialog box

Rantest has been used as a project name, and project directory has been specified as D:\EmbestIDE\Example\arm250\explasm\, D:\EmbestIDE is default installing directory of EmbestIDE unless noted.

After clicking **OK**, two files will be created in the project directory:

| | |
|---------------------|-----------------------|
| Rantest.ews | Workspace file |
| Randtest.pjf | Project file |

Note: Workspace and project files are maintained by EmbestIDE system itself. Users cannot edit these files manually.

After the project has been created, files pane will appear in EmbestIDE workspace window as shown in Figure 3-2.

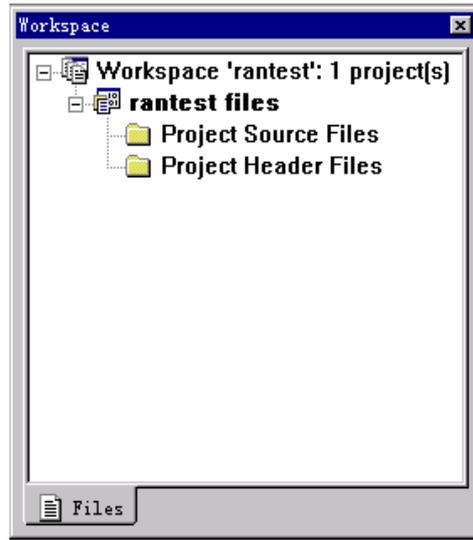


Figure 3-2 Workspace Window

The Figure 3-2 shows current workspace is rantest which contains a project named rantest. Boldface indicates that rantest is an active project.

2) Add Source Files

Choose **Project Source Files** folder in rantest workspace window. Click Project > **Add To Project > Files** to add source files. You can also right click **Project Source Files** folder to add source files, See Figure 3-3:

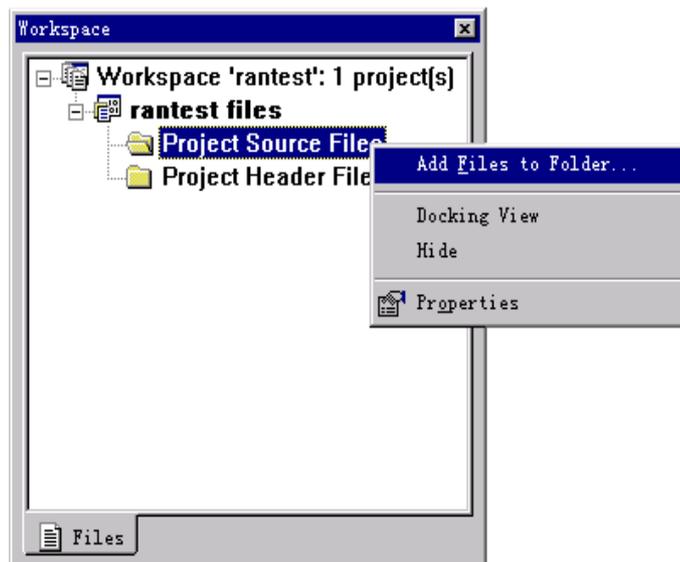


Figure 3-3 Add Source Files In Workspace Window

You can add the source files in the pop up dialog box. To select source files, press CTRL key and hold, click source files of random.s and randtest.c in Project directory, See Figure 3-4.

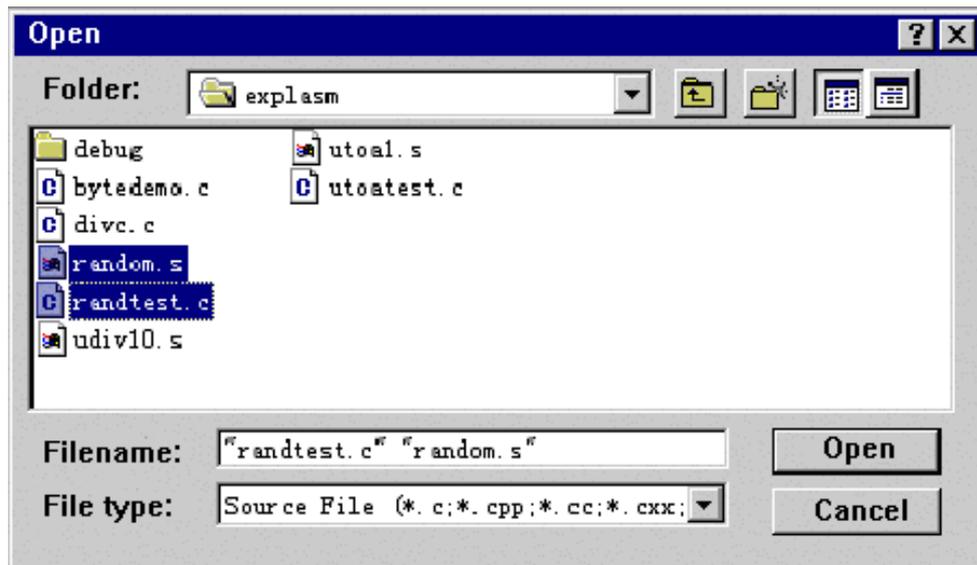


Figure 3-4 File Open Window

After the source files added, workspace window is shown as Figure 3-5.

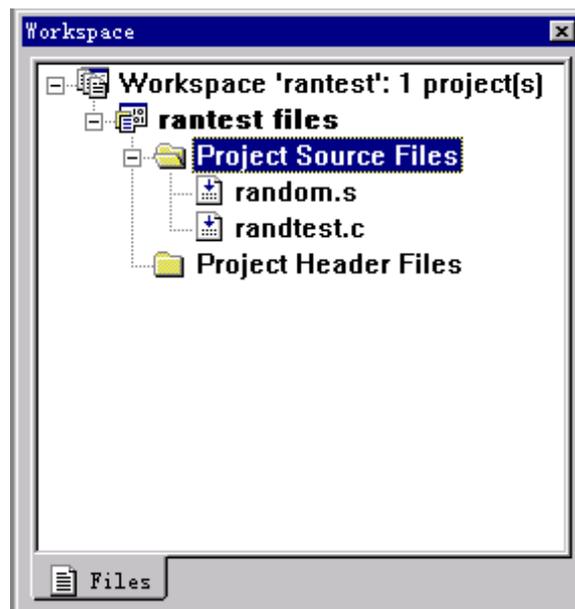


Figure 3-5 Workspace Window

3.1.2 Project Settings

After creating a project, you need to configure the project. The project settings include CPU settings, emulator settings, debug settings, directory settings, compiler settings, assembler settings and linker settings. The project settings is a critical step to entire software development.

Click **Project > Settings**, Project Settings dialog box pops up, then Choose randtest files in the dialog.

1) CPU Settings

Select Processor pane from Project Settings dialog box. Define **CPU module** as ARM7, **CPU family** and **CPU member** as ARM7 too. Switch Endian to little endian. Build tools are GNU tools for ARM. Figure 3-5 is a Project Settings dialog.

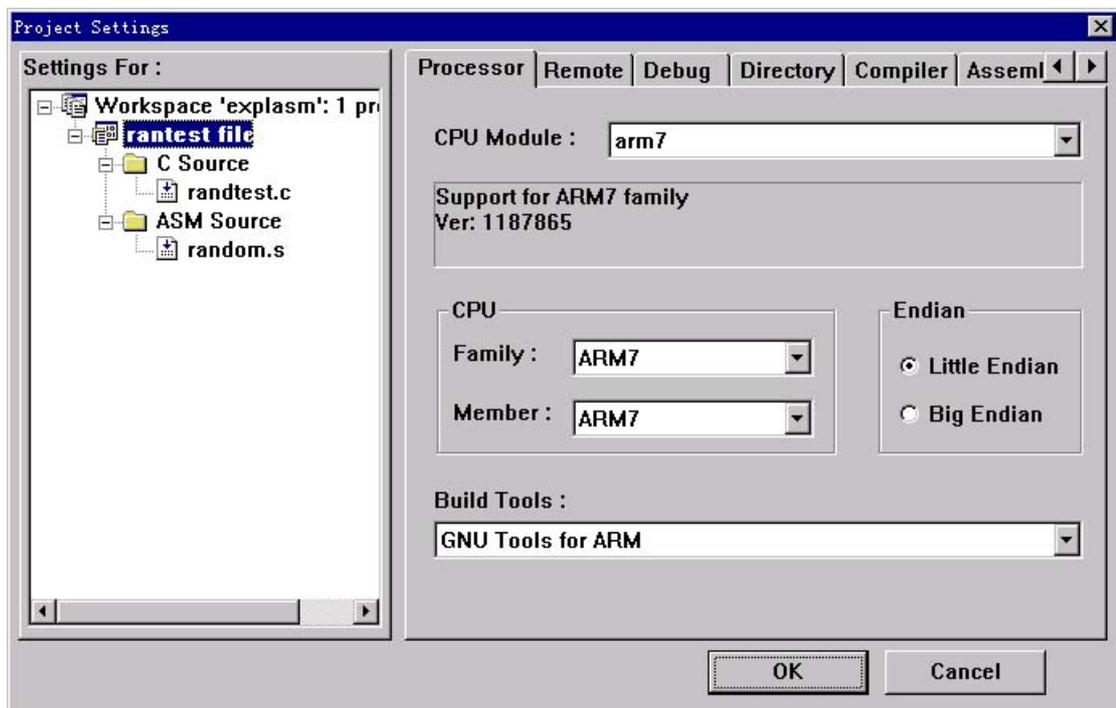


Figure 3-5 Processor pane of Project Settings Dialog

After build tool has been set, Project Settings dialog will show the pane of this build tool.

2) Emulator Settings

Select Remote pane from Project Settings dialog. Define Remote Device as jtagarm7. Because of Embest Emulator for ARM does not support to change the working speed, don't worry about the setting of Speed item. When using Embest

PowerICE for ARM, select the valid work speed: Full Speed(120Kbyte/s)、High Speed、Medium Speed、Low Speed. Communication port with remote device is PARALLEL. Communication channel is LPT1 as shown on Figure 3-6.

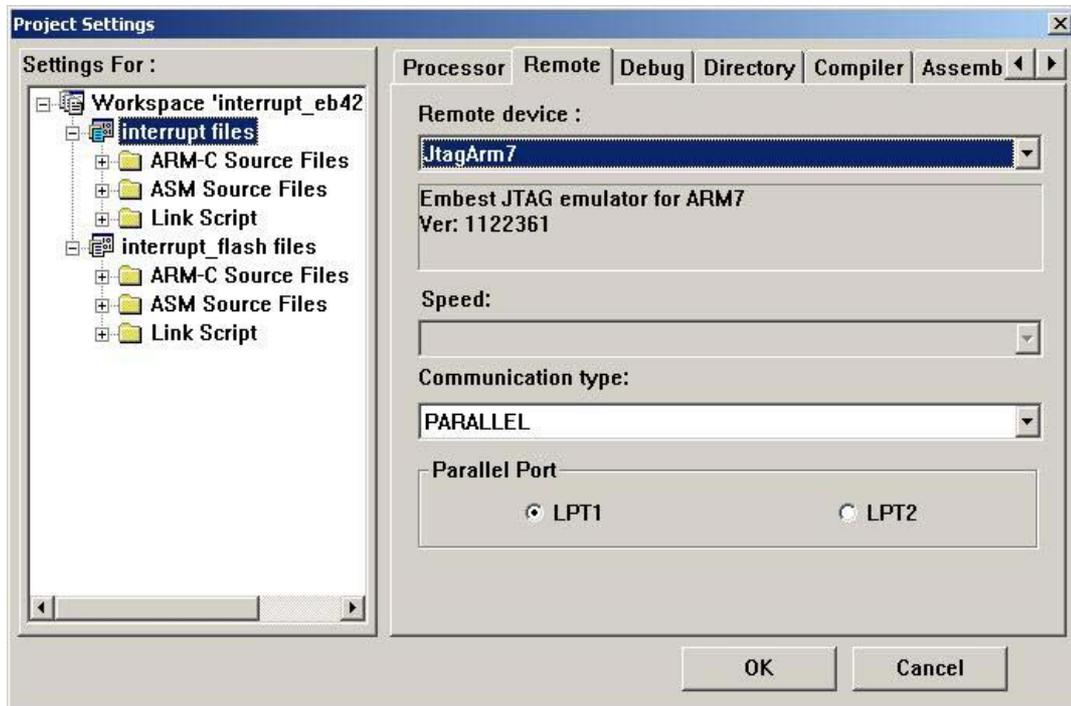


figure 3-6 Remote Pane of Project Settings Dialog

3) Debug Settings

Select Debug pane from Project Settings dialog. Select General from Category, Set **symbol file** as `./debug/rantest.elf` and **Action after connected** as None. See Figure 3-7.

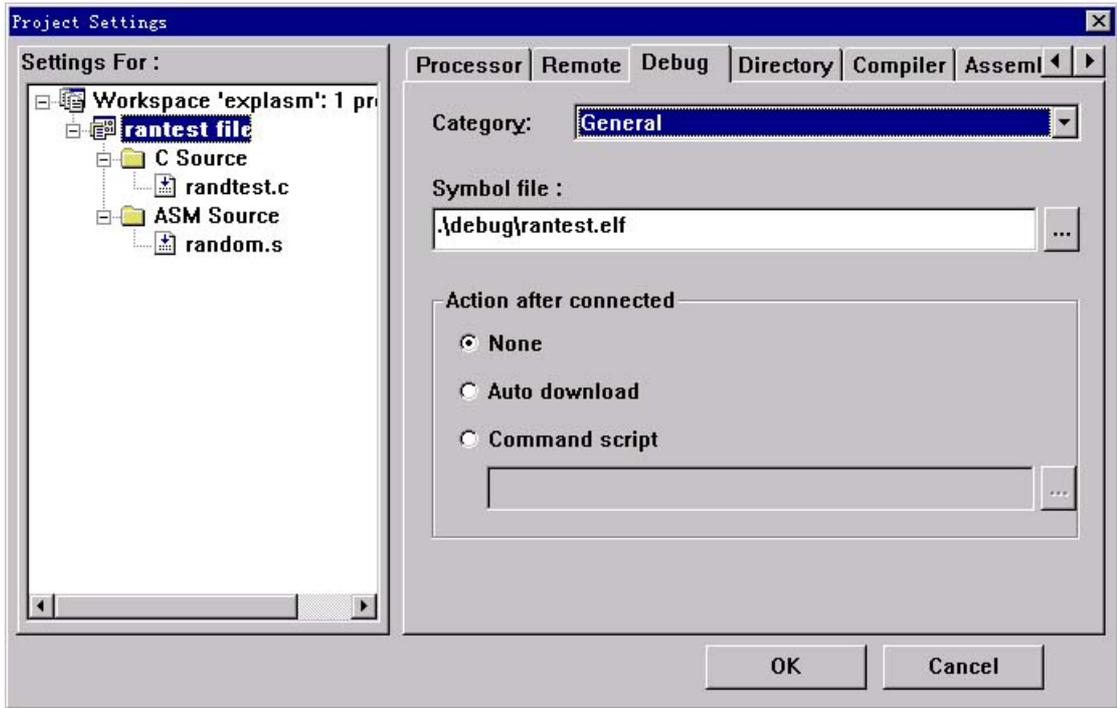


figure3-7 Debug General Options of Project Settings Dialog

Select Download from Category. Set **Download file** as ./debug/rantest.elf. set **Download Verify** option on. Set Download file to address 0x2000000. Execute program starting from **download address**. See Figure 3-8.

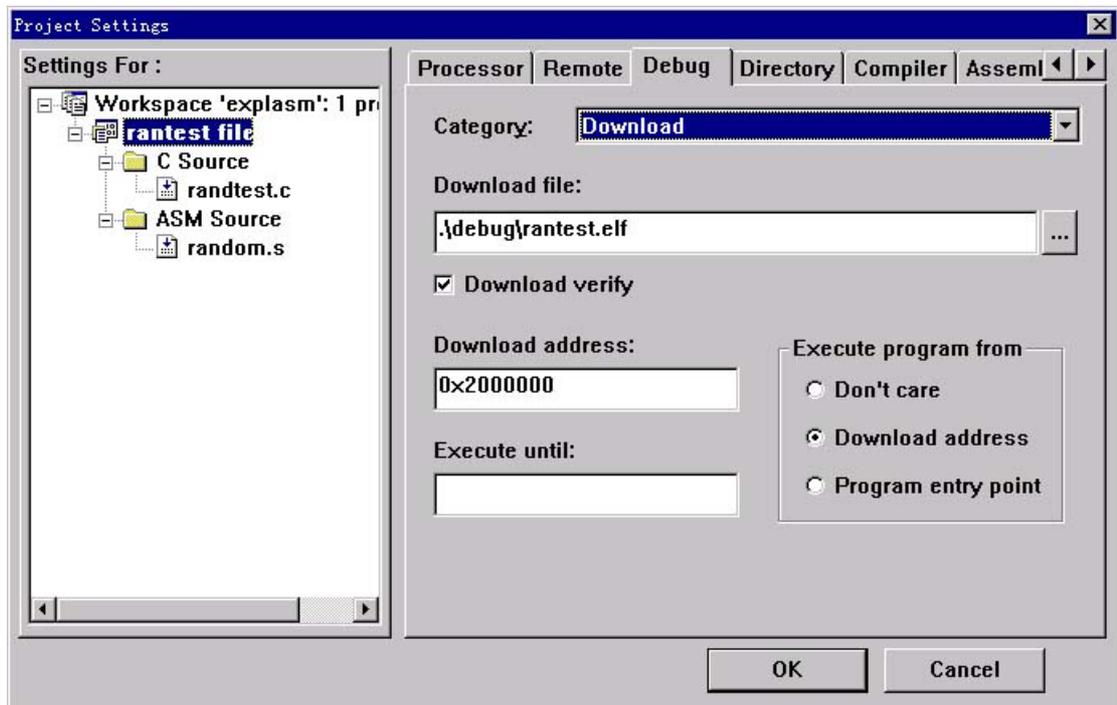


figure 3-8 Debug Download Options of Project Settings Dialog

Note: Flat Binary format can be used for download files. The tool ELF2BIN provided by IDE can transfer ELF files to the files with flat binary format. If you use different circuit board, please change download file address to RAM address of the circuit board.

Set **Memory Map** to **use map file**, and set target memory map file as `$(EMBEST_IDE)\targets\at91\targets\eb40\eb40.map`. See Figure 3-9.

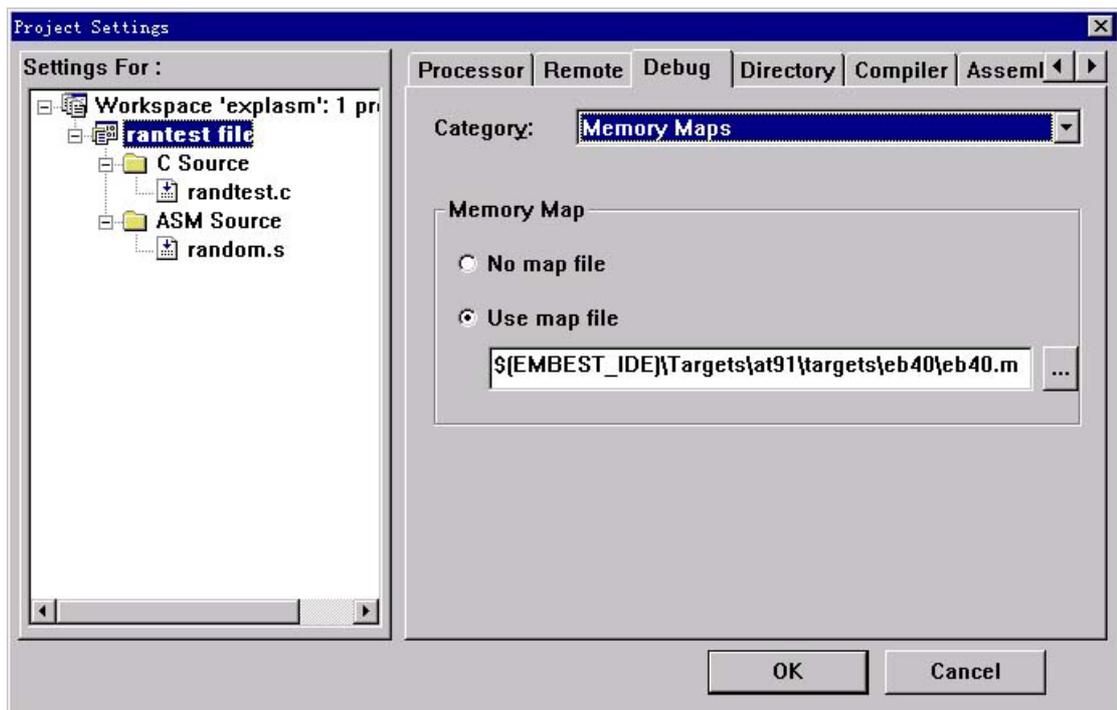


Figure 3-9 Debug Memory Maps Options of Project Settings

4) Compiler Settings

Select Compiler pane from Project Settings dialog. Set **Object Files Location** as `./debug` and others as default. See Figure 3-10.

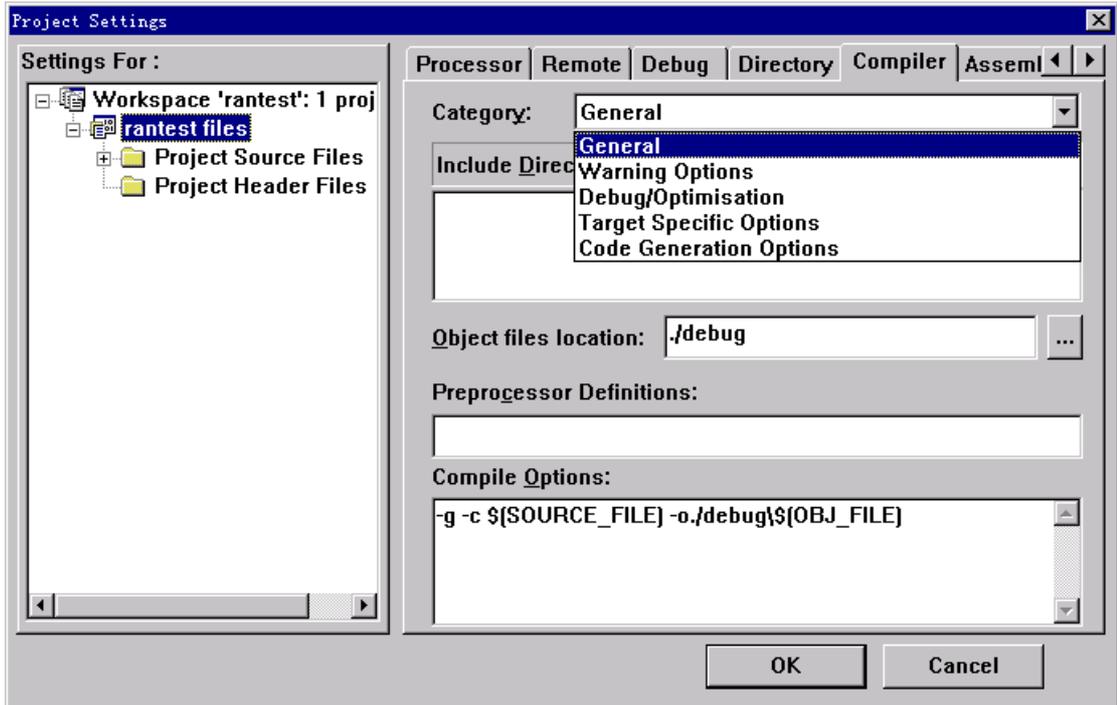


Figure 3-10 Compiler Pane of Project Settings Dialog

5) Assembler Settings

Select Assembler pane from Project Settings Dialog. Set **Object Files Location** as ./debug and other as default. See Figure 3-11.

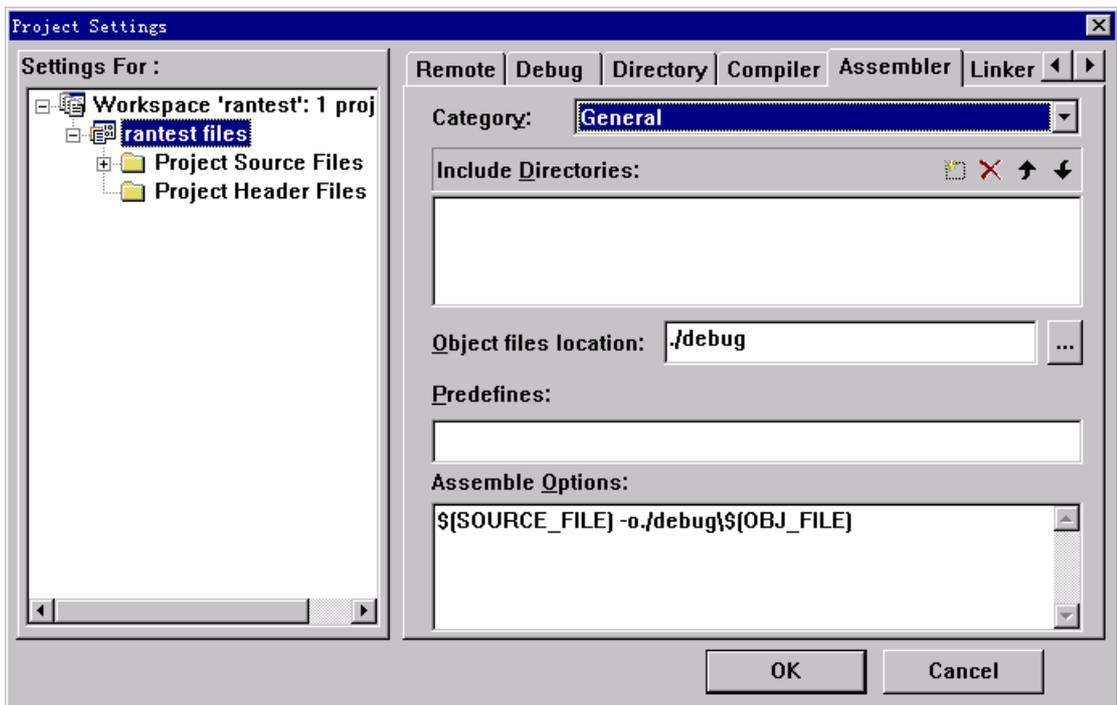


Figure 3-11 Assembler Pane of Project Settings Dialog

6) Linker Settings

Select Linker pane from Project Settings Dialog. Select General from Category. Set **Object Files Location** as ./debug and others as default. See Figure 3-12.

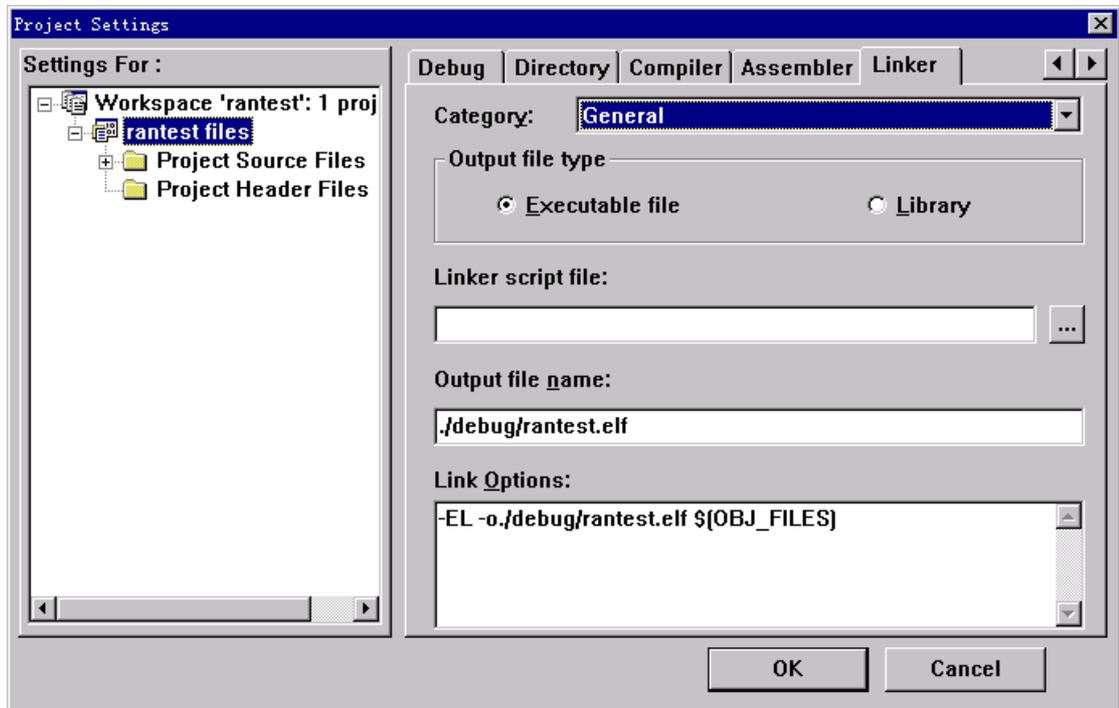


Figure 3-12 Linker General Options of Project Settings

Select Image Entry Options from Category. Set **Select Entry File** as rantest.o. Select entry point as Main. Add "-Ttext 0x2000000" in link options. See Figure 3-13.

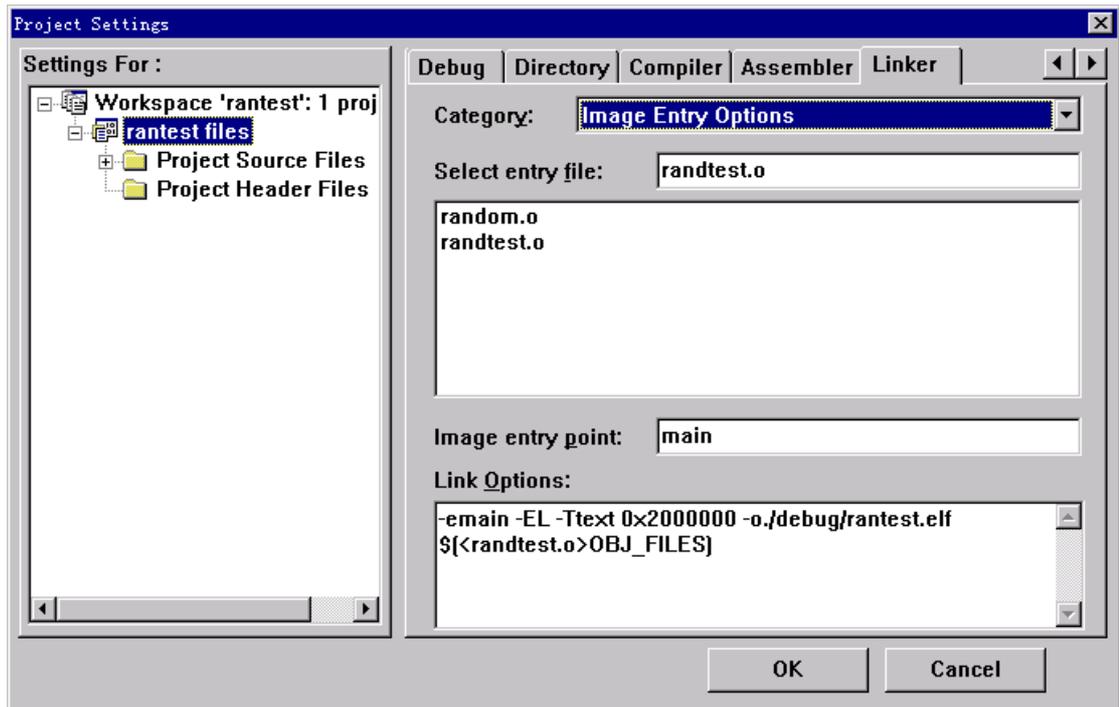


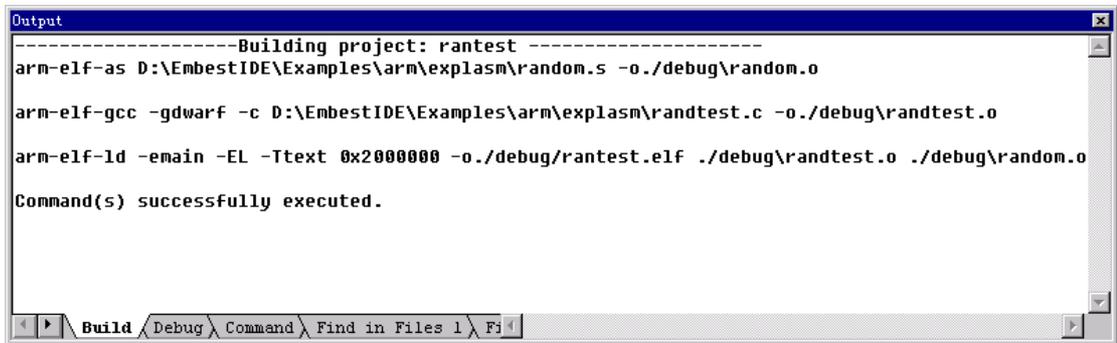
Figure 3-13 Linker Image Entry Options of Project Settings Dialog

Entry File 'Rantest.o' means that executive code compiled and linked 'rantest.c' will locate at starting point of entire executive program. Entry Point 'main' means that executive code will run starting from 'main'. "-Ttext 0x2000000" means that address of entire executive program code will start from 0x2000000.

Note: The step described above can be skipped if users open workspace file explasm.ews created under the directory.

3.1.3 Compile and Link

Click **Build > Build rantest** to complete entire project building. You can view the output in the **output** window, the following messages show as Figure 3-14.



```
Output
-----Building project: rantest -----
arm-elf-as D:\EmbestIDE\Examples\arm\explasm\random.s -o./debug/random.o
arm-elf-gcc -gdwarf -c D:\EmbestIDE\Examples\arm\explasm\randtest.c -o./debug/randtest.o
arm-elf-ld -emain -EL -Ttext 0x2000000 -o./debug/rantest.elf ./debug/randtest.o ./debug/random.o
Command(s) successfully executed.
```

Figure 3-14 Build Output Window

The message shown in the window indicates that this project building has been success.

3.1.4 Debug

1) Create Executive File

Click **Tools > Elf to Bin** to automatically transfer the project output file called "project name +.elf" under 'debug' sub-directory to executive file. The executive file has flat binary file format. The final file will be named as "project name +.bin" and saved in 'debug' sub-directory under project directory.

You can also complete file transfer by running elf2bin.exe located in Bin sub-directory under install directory.

Note: This step can be skipped if you have taken a choice of downloading ELF format file.

2) Active connection

Click **Debug > Remote Connect** to activate the connection with the target board through Embest JTAG emulator. The target board will be one of the two kinds of status:

- Running status: When the target board is in this status, debug output window shows messages as "Info: target running, all breakpoints disabled." as shown in Figure 3-15.

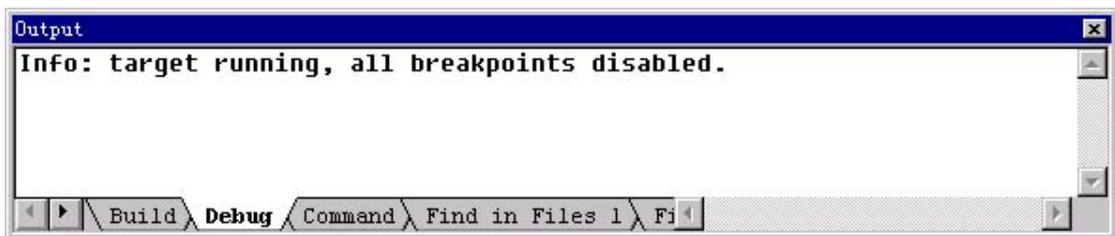


Figure 3-15 Debug Output Window

In this case, Embest IDE Disassemble window shows nothing.

- Stop status (This happens when you stopped the activating connection with the target board previously, you stopped running target board prior). When the target board is in this status, debug output window shows message as "Info: CPU was in debug state before connect! Current values may be incorrect!" as shown in Figure 3-16.



Figure 3-16 Debug Output Window

In this case, Embest IDE Disassemble window shows assemble instruction, with the instruction currently PC is pointing to beginning. See Figure 3-17.

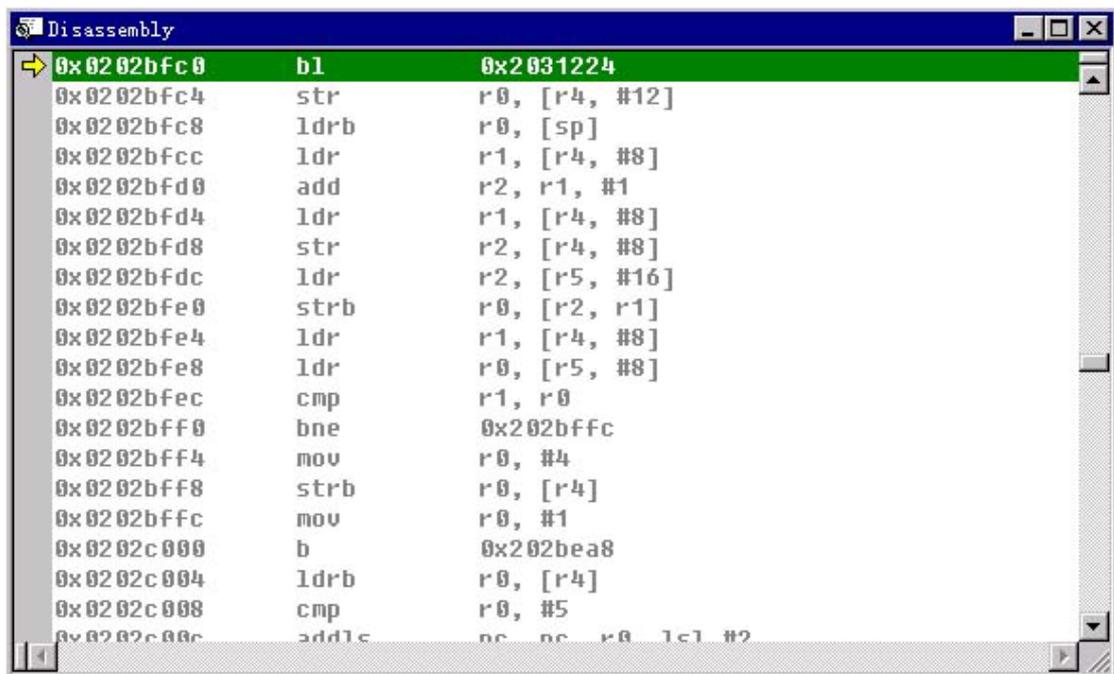


Figure 3-17 Disassemble Window

Note: In general, disassemble window which you see is not as same as above.

3) Download File

Click **Debug** > **Stop** to stop running target board if current target board is on running status. Click **Debug** > **Download** to download file if current target board is on stopping status. Downloading parameters were set through Debug pane in Project Settings dialog. When downloading files, status bar will show

“Download File” with progress bar to indicate download progress. Status bar will show “Download Completed” when the downloading finished.

When download is completed, program pointer will automatically return to program starting point. Disassemble window will appear as shown in Figure 3-18.

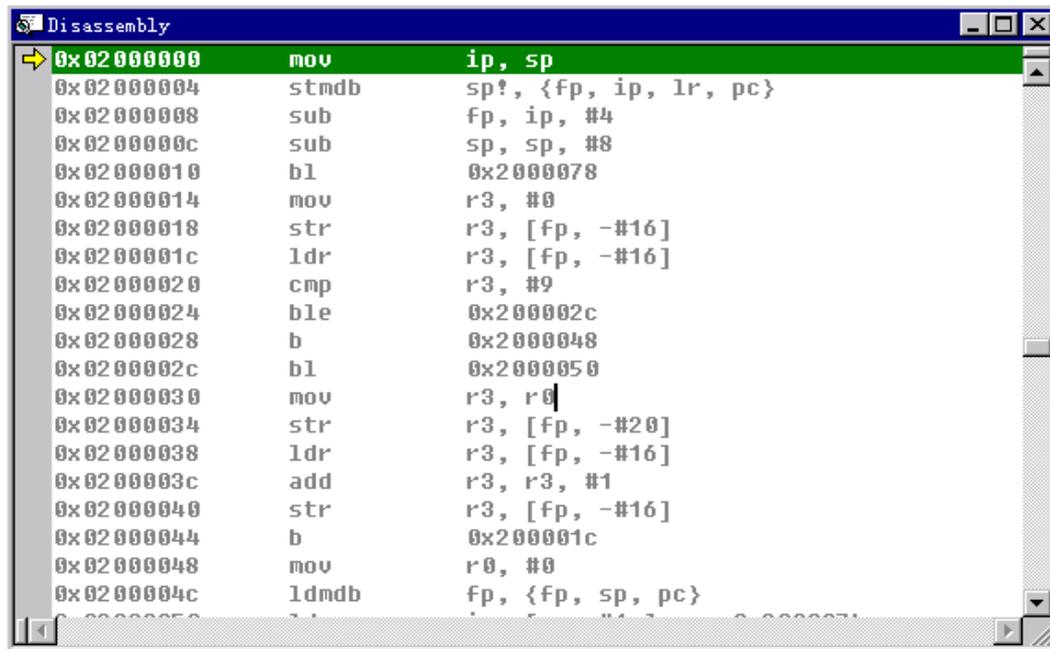


Figure3-18 Disassemble Window

At the same time, rantest.c source file window will appear as well as shown in Figure 3-19.

```
D:\...\EmbestIDE\Examples\Arm\explasm\randtest.c

*/
#include <stdio.h>

/* this function prototype is needed because 'randomnumber' is external
extern unsigned int randomnumber( void );

int main()
{
    int loop;
    unsigned int random;

    for( loop = 0; loop < 10; loop++ )
    {
        random = randomnumber();
    }

    return( 0 );
}
```

Figure 3-19 Source File Window

4) Debug

Click **Debug** > **StepOver** to run program to this line:

random = randomnumber();

Click **View** > **Debug Windows** > **Variables** menu to show the Variable window. Continuous to click **Debug** > **StepOver** to view the changes of function variables as shown in Figure 3-20.

| Name | Value |
|--------|----------|
| loop | 0x2 |
| random | 0xaaaaa0 |
| | |
| | |
| | |

Local Global

Figure 3-20 Variables Window

3.2 A Complete Example

This section shows you step by step to create, compile and debug a complete project. The project demonstrates communication between PC and AT91EB40 evaluation board through parallel port. This is a complete embedded application.

The example project locates at \Examples\At91\example_terminal40 under Embest IDE installed directory. The files to be used are:

| | |
|-----------------------|--|
| term.c | Terminal Test Main Program File |
| Terminal_irq.s | Assemble Interrupt Handle Program File |
| Cstartup.s | Startup Program File for evaluation board. The file locates at the following sub-directory under installed directory \targets\At91\targets\Eb40\. |

The differences between the project described in this section and the project described in the previous section are:

- The project of this section contains complete target board startup codes. After the program is debugged successfully, you can change AT91_DEBUG_ICE=1 defined by project settings to AT91_DEBUG_NONE=1, and then rebuild it, and this can be downloaded to FLASH to run itself.

- The project of this section uses individual linker scripts. The file is

| | |
|-----------------|---|
| ldscript | Linked Scripts, The file is located at the following directory \Targets\At91\Targets\. |
|-----------------|---|

- The project of this section links to standard C function library. The file of function library

| | |
|---------------|--|
| libc.a | Standard C function library, support arm interwork, The file is located at \build\xgcc-arm-elf\arm-elf\lib\arm-inter. |
|---------------|--|

- The project of this section connects to peripheral driver function library and device function library of AT91 series CPU provided by ATMEL. They include:

Lib_drv_32.lib **Peripheral Driver Function Library, The file is located at**
\targets\at91\drivers\lib_drv\arm-inter
under installed directory. The project file
creating the library is located at the directory
one level up. It can create THUMB instruction
library or ARM Interwork library by changing
project settings. The example shown here
uses ARM Interwork function library.

R40807_lib32.lib **Device Function Library. The file is located at**
\targets\at91\parts\r40807\arm-inter
under installed directory. The project file
creating the library is located at the directory
one level up. It can create THUMB instruction
library or ARM Interwork library by changing
project settings. The example shown here
uses ARM Interwork function library.

Note 1: Before using the project, you must create peripheral driver function library named lib_drv_32.lib by using workspace file lib_drv_32.ews, which located at \target\at91\drivers\lib_drv\ directory and create device function library named r40807_lib32.libby using workspace file r40807_lib32.ews, which located at \targets\at91\parts\.

Note 2: If you don't want to use ARM Interwork, you can create function library, which support ARM only. And don't choose to support ARM interwork in project compiler and assembler settings.

3.2.1 Create Project

The project called Term can be created and related source files can be added by following the same procedure described in the previous section at \examples\at91\example_terminal under installed directory. Final workspace window will be appeared as shown in the Figure 3-21.

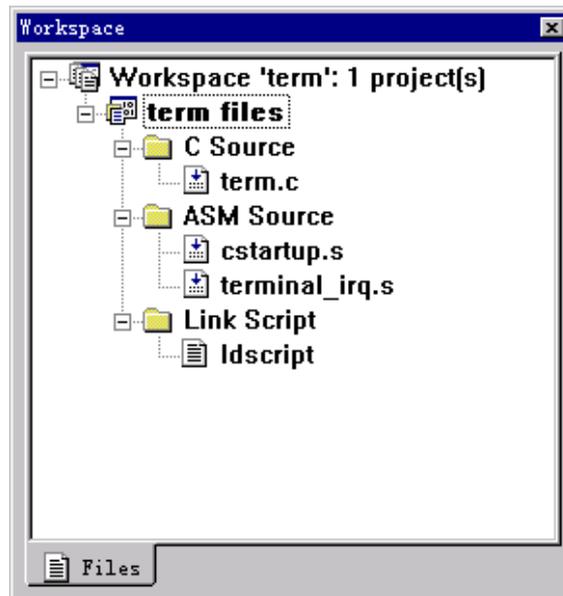


Figure 3-21 Workspace Window

3.2.2 Project Settings

The processor settings and remote settings can be done by following the same procedure described in Section 3.1.2 previously. Please refer to it.

1) Debug Settings

Select Debug pane from Project Settings dialog. Set **Symbol file** settings as **.\debug\term.elf**. Set **Action after connected** option to **None**. See Figure 3-22.

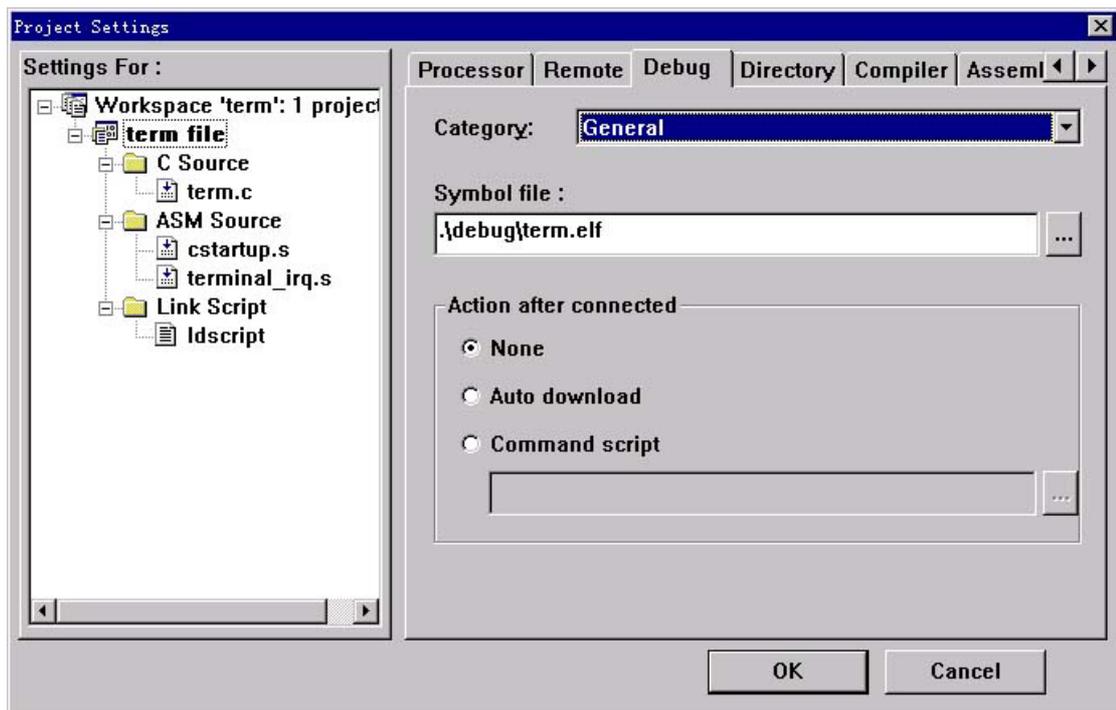


Figure 3-22 Debug General Options of Project Settings Dialog

Select **Download** from **Category**. Set **download file** to **.\debug\term.elf**. Set **Download Verify** option on. Set **download address** for download file as 0x2000000. Execute program starting from download address as shown in Figure 3-23.

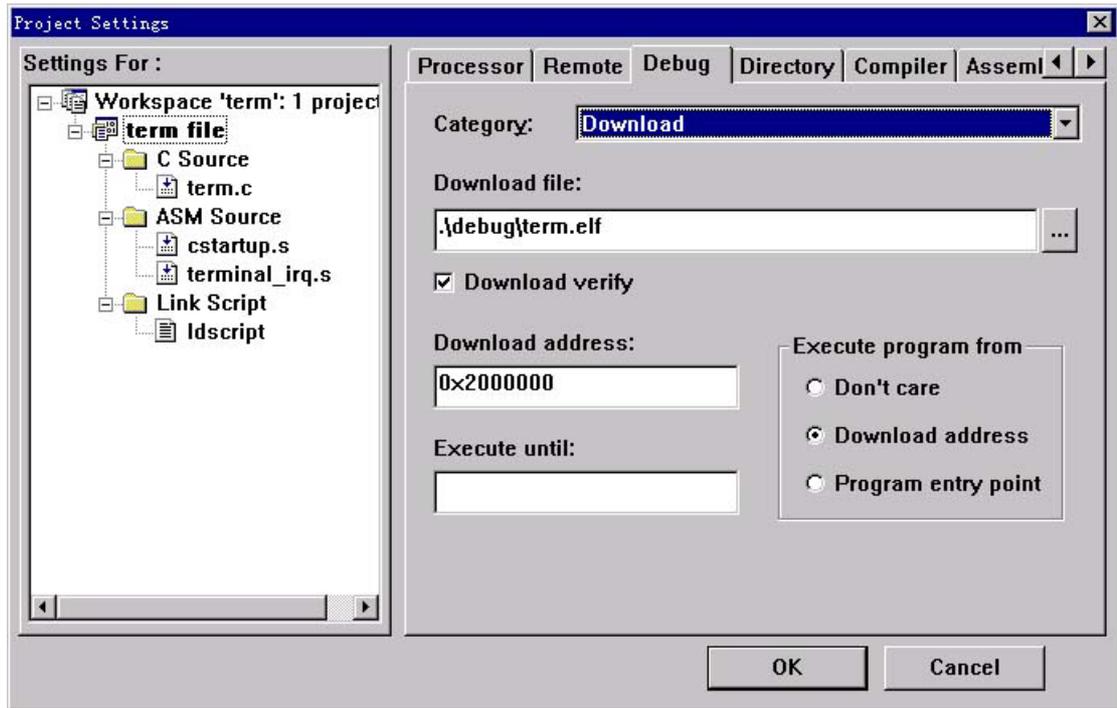


Figure 3-23 Debug Download Options of Project Settings Dialog

Memory Maps Settings is the same as the first section. Please refer to it.

2) Directory Settings

Select Directory pane from Project Settings Dialog. Set **additional source file directory** as `$(EMBEST_IDE)\Target\at91\drivers\terminal`.

`$(EMBEST_IDE)` means Embest IDE installed directory. If you want to track the program in driver function library or device function library, you can add more source file directories here as shown in Figure 3-24.

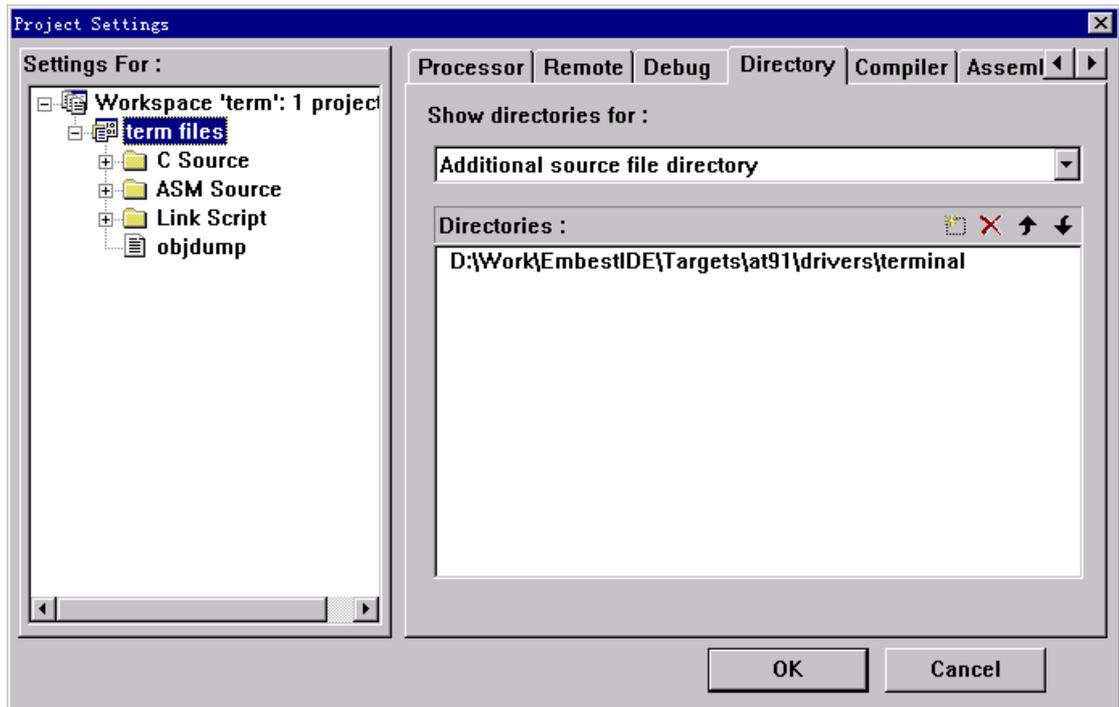


Figure 3-24 Directory Pane of Project Settings Dialog

3) Compiler Settings

Select Compiler pane from Project Settings Dialog. Select **General** from **Category**. Set **Object files location** as `.\debug` and add two include directories: `..\..\..\targets\at91` and `..\..\..\build\xgcc-arm-elf\arm-elf\include`. Select **Target Specific Options** from **category**. Set **Instruction Sets** as ARM interworking and others as default. See Figure 3-25.

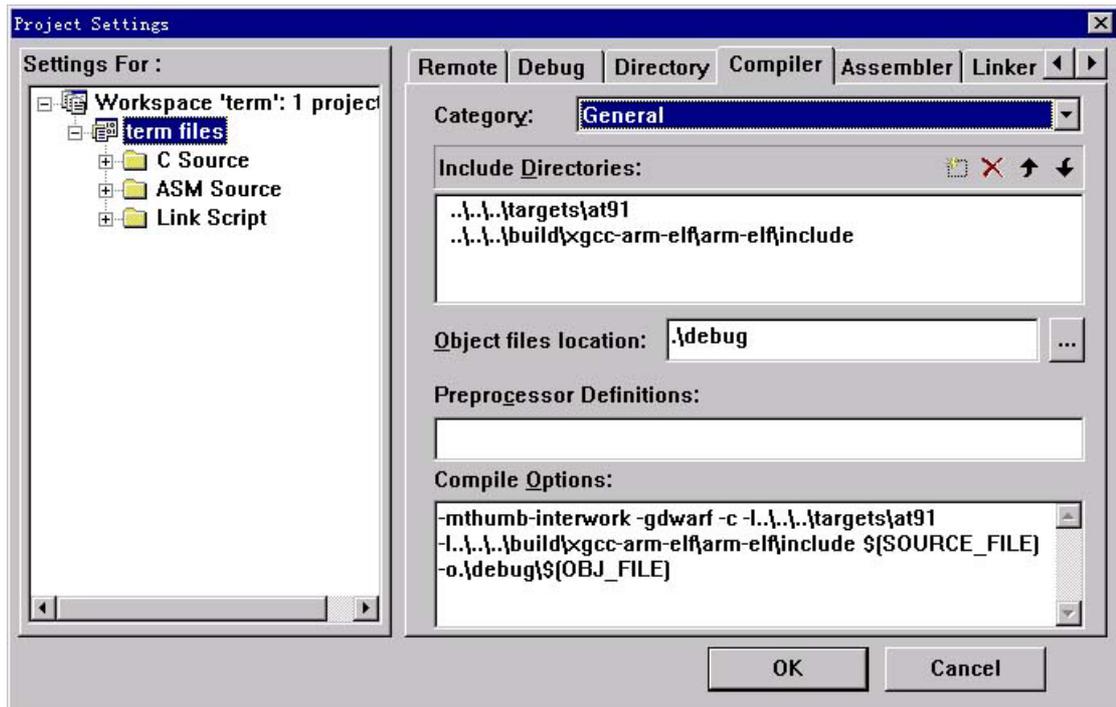


Figure 3-25 Compiler General Options of Project Settings Dialog

4) Assembler Settings

Select Assembler pane from Project Settings dialog. Select **general** from **category**. Set **Object file location** as `.\debug`. Add three include directories: `..\..\..\targets\at91`, `..\..\..\targets\at91\parts\r40807`, `..\..\..\targets\at91\targets\eb40`. Add three predefines: `AT91R40807=1`, `AT91EB40=1`, `AT91_DEBUG_ICE=1`. Select **Target Specific Options** from **category**. Make the assembled code as supporting interworking and others as default. See Figure below.

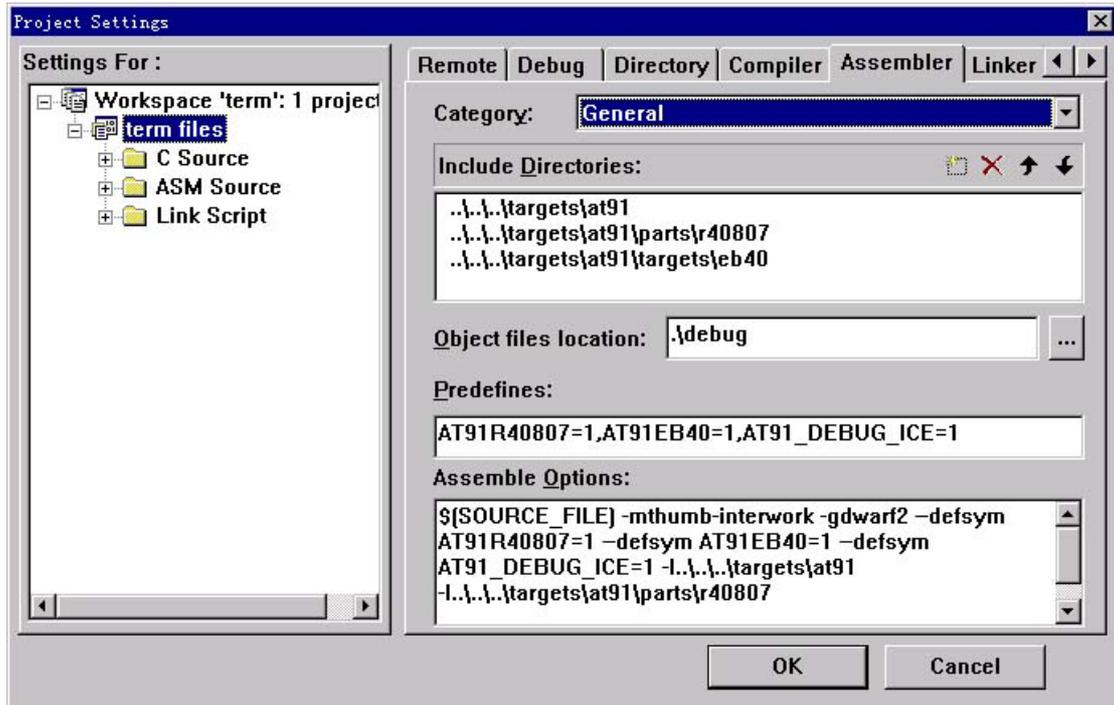


Figure 3-26 Assembler Pane of Project Settings Dialog

5) Linker Settings

Select Linker pane from Project Settings dialog. Select **general** from **category**. Set **Output file name** as `.\debug\term.elf` and **linker script file** as `..\..\..\targets\at91\targets\ldscript` as shown in Figure 3-27.

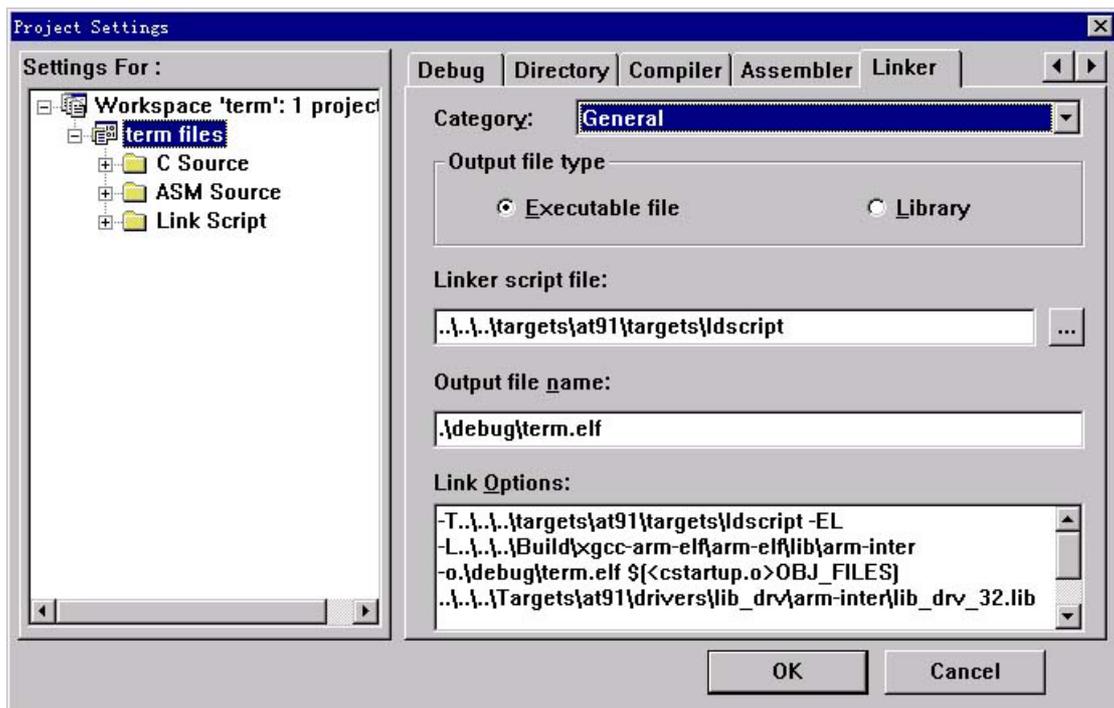


Figure 3-27 Linker General Options of Project Settings Dialog

Select **Image Entry Options** from **category**. Make entry file as cstartup.o as shown in Figure 3-28.

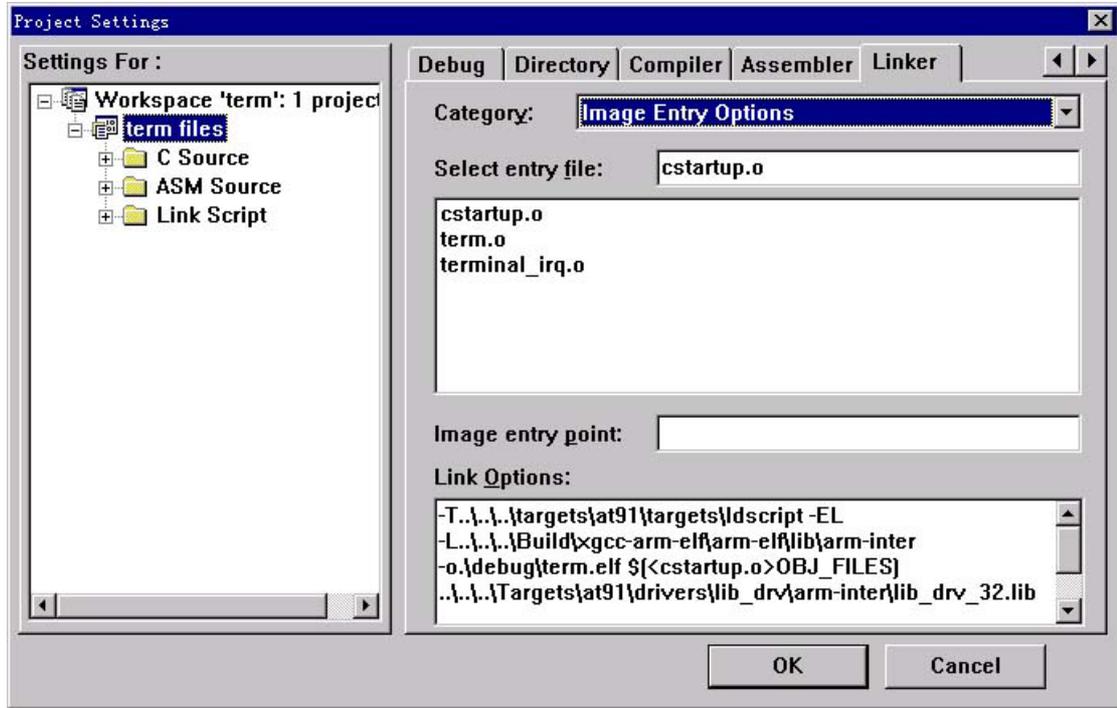


Figure 3-28 Linker Image Entry Options of Project Settings Dialog

Select **Include Object and Library Modules** from **category**. Add three library files:

```
..\\..\\..\\Targets\\at91\\drivers\\lib_drv\\arminter\\lib_drv_32.lib,  
..\\..\\..\\Targets\\at91\\parts\\r40807\\arm\\r40807_lib32.lib,  
..\\..\\..\\build\\xgcc-arm-elf\\arm-elf\\lib\\arm-inter\\libc.a.
```

See Figure below.

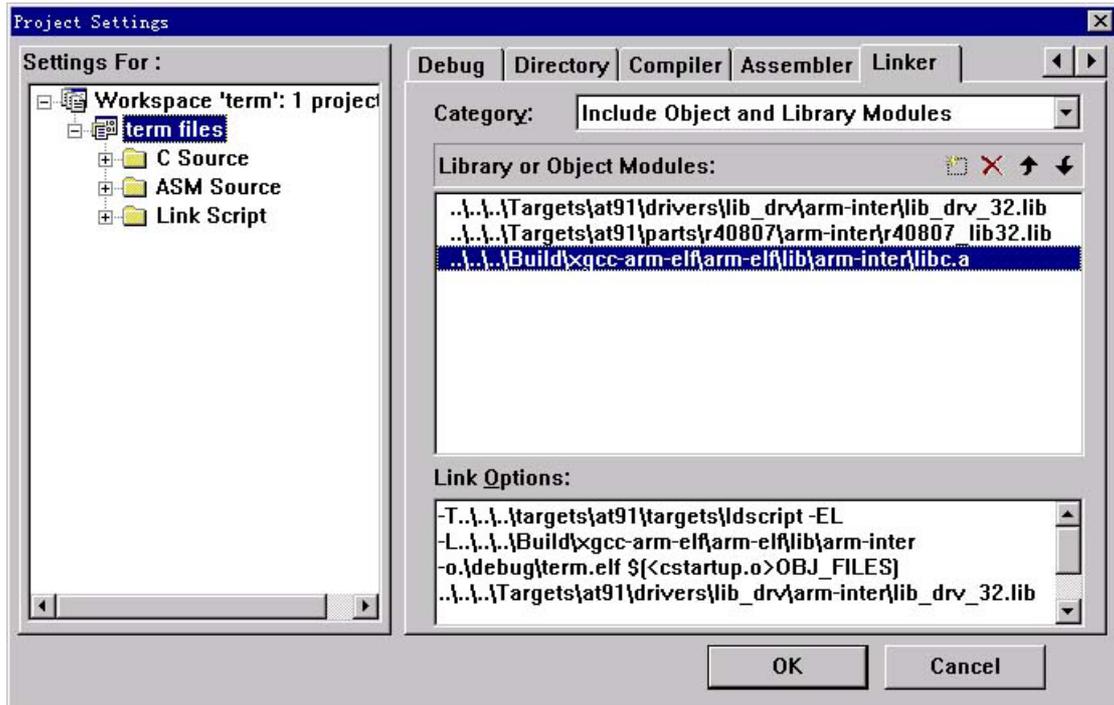


Figure 3-29 Linker Include Object Options of Project Settings Dialog

Note: The step above can be skipped if you directly open workspace file term.ews created in the directory.

3.2.3 Compile and Link

Click **Build > Build term** to complete compiling, assembling and linking the entire project. When “Command(s) successfully executed” appears at build window of output window, it means the project has been successfully built.

3.2.4 Debug

1) Terminal Program Settings

Connect Serial A on EB40 circuit board to COM port of PC by using a standard series port wire. Run Terminal program of windows. Click **Setting** > **Communication** and set communication parameters as shown in Figure 3-30.

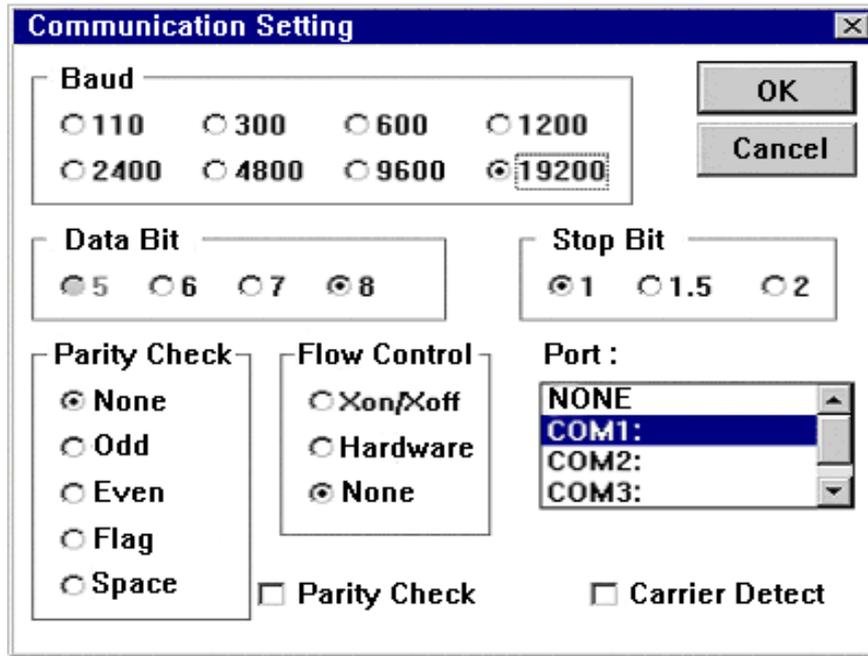
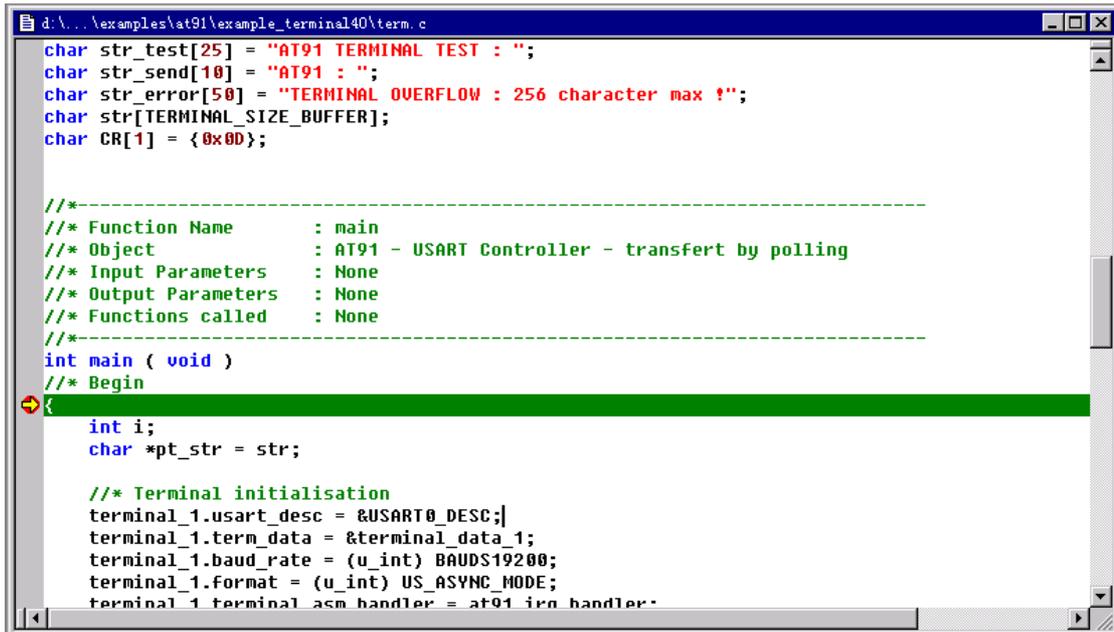


Figure 3-30 Terminal Program Communication Settings Dialog

Select port according to COM port of PC connected. Figure 3-30 above is set to COM1 Port.

2) Debug

Create executive file, active the connection with target board through Embest JTAG emulator, and then download the executive file. Disassemble window will pop out after downloading file. Disassemble window will show assemble code starting from address 0x2000000. Source file window relating with instruction address 0x2000000 will pop out then. You can open file of Term.c and set breakpoint at the beginning of main () function. Click **Debug** > **Go** and execute program up to this line as shown in Figure below.



```
d:\...\examples\at91\example_terminal40\term.c
char str_test[25] = "AT91 TERMINAL TEST : ";
char str_send[10] = "AT91 : ";
char str_error[50] = "TERMINAL OVERFLOW : 256 character max !";
char str[TERMINAL_SIZE_BUFFER];
char CR[1] = {0x0D};

/*-----
/* Function Name      : main
/* Object             : AT91 - USART Controller - transfert by polling
/* Input Parameters   : None
/* Output Parameters  : None
/* Functions called   : None
/*-----
int main ( void )
/* Begin
{
    int i;
    char *pt_str = str;

    /* Terminal initialisation
    terminal_1.usart_desc = &USART0_DESC;
    terminal_1.term_data = &terminal_data_1;
    terminal_1.baud_rate = (u_int) BAUDS19200;
    terminal_1.format = (u_int) US_ASYNC_MODE;
    terminal_1.terminal_asm_handler = at91_irq_handler;
```

Figure 3-27 Source File Window

Click **Debug > Go** again. The program starts running. Terminal program shows message of "AT91 TERMINAL TEST:" as shown in Figure below.

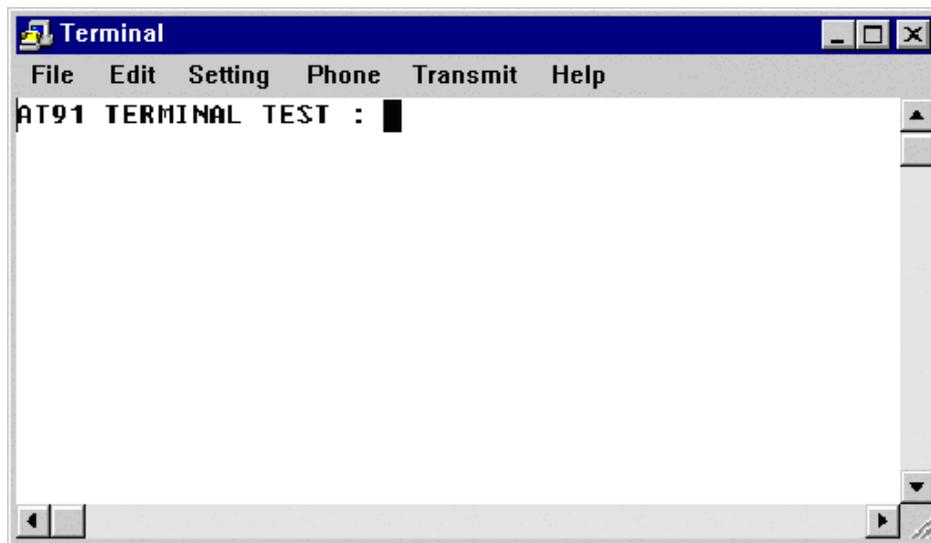


Figure 3-27 Terminal Program Window

Input "Hello, Embest". Terminal program will show "AT91: Hello, Embest".

3.3 Example Project of S3C4510B

The example program introduced in this part is a complete program based on S3C4510, which may be debugged in RAM, and solidification may be normally conducted in ROM. Led will be lit at interval during normal operation period, and press the button to light another led.

For the convenience of users to rapidly understand S3C4510 programming and application, the whole software includes only two program source files:

- starting assembling file: `init.s`
- C source program file: `ledint.c`

All macro definitions are included in the source files, and no other header files are used.

The example program has integrally demonstrated the starting process of S3C4510 processor, including configuration of memory area, stack setting and interrupt vector setting, and has demonstrated the control to IO port after completion of starting and function processing of interrupt. The example program is the minimum program framework of S3C4510, and users can extend their own application on this base.

The example program is based on EV4510 EVM board of Embest Corporation, and the basic configuration includes 2M of Flash area, 16M of DRAM area, IO port P16 connected to LED1, IO port P17 connected to LED2, and IO port P9 connected to the button. If developing personnel want to apply the program to the PCB designed by themselves, they only need to modify the value of the system configuration register and the setting value of the memory area register in the starting file, meanwhile, modify the value of the relevant register of IO port in C-file according to actual IO port connection.

3.3.1 Interpretation of Source Program

- **starting assembling file**

Starting assembling file "init.s"; starting assembling file will complete in turn: setting of interrupt vector, setting of system configuration register, configuration of memory area, copy of data segment used in program to RAM area, initialization of stack space, and entering into C language program entry.

What shall be paid attention to includes:

- 1) if software is debugged in RAM, the configuration of memory area will be completed by integration environment through command script file, therefore, it is unnecessary to use memory area configuration code in program, which can be completed through switching to ROM symbol definition;
- 2) when debugging is made in RAM, it is unnecessary to copy the content of data segment, which will be automatically selected in program through judgment whether the addresses of read only area and read-write area are overlapped;
- 3) no handing to primary exception vector such as failure of prefetch has been made in the code, perfect program shall handle any of primary vectors, including saving the implementing state before entering into primary exception vector for reference and returning to implement the program after removing possible mistake.

The source code and the detailed interpretation of "Init.s" are as follows:

```
/******  
* file name: init.s  
* description: S3C4510 starting program  
*****/  
  
#=====  
# programming register bit definition  
#=====  
.EQU LOCKOUT, 0xC0 @ forbid all interrupts  
.EQU MODE_MASK, 0x1F @ processor mode bit  
.EQU UDF_MODE, 0x1B @ undefined mode UDF  
.EQU ABT_MODE, 0x17 @ abnormal mode ABT  
.EQU SUP_MODE, 0x13 @ superuser mode SVC  
.EQU IRQ_MODE, 0x12 @ interrupt mode IRQ  
.EQU FIQ_MODE, 0x11 @ fast interrupt mode FIQ  
.EQU USR_MODE, 0x10 @ user mode USR
```

```

=====
# set interrupt and primary exception vector
=====
ENTRY:
    B        Reset_Handler          @ implement from here after S3C4510
reset
    B        SystemUndefinedHandler
    B        SystemSwiHandler
    B        SystemPrefetchHandler
    B        SystemAbortHandler
    B        SystemReserv
    B        SystemIrqHandler
    B        SystemFiqHandler

=====
# reset entry point
=====
    .global Reset_Handler
Reset_Handler:                    @ reset entry point

=====
# set system configuration register
=====
    LDR r0, =0x3FF0000             @ address of system configuration
register: 0x3FF00000
    LDR r1, =0x83FFFF90           @ value of register is set as 0x83FFFF90
    STR r1, [r0]                  @ significance of register: to use
synchronical DRAM, peripheral register base value 0x3FF0000, which can
cache

=====
# configuration of memory area, not define ROM when debugging the program
in RAM, and define when solidified
=====
#ifdef ROM
    LDR    r0, =SystemInitDataSDRAM @ load the setting value of memory
area register to save address
    LDMIA  r0, {r1-r12}             @ load 12 setting values
    LDR r0, =0x3FF0000 + 0x3010     @ load the address of memory area
register
    STMIA  r0, {r1-r12}            @ set memory area register
#endif

=====

```

```

# introduce external symbol, symbol definition is in link script file
#=====
.extern Image_RO_Limit      @ size of read only area
.extern Image_RW_Base      @ initial address of read-write memory
area
.extern Image_ZI_Base      @ initial address of clear area, the
area of the uninitialized variable .bss segment in code

.extern Image_ZI_Limit     @ size of clear area

#=====
# memory area needed to use to initialize C code
#=====
LDR r0, =Image_RO_Limit    @ obtain the size of read only area
LDR r1, =Image_RW_Base    @ obtain the initial address of
read-write memory area
LDR r3, =Image_ZI_Base    @ obtain the initial address of clear
area
CMP r0, r1                 @ compare whether the read only area and
the read-write area are overlapped
BEQ LOOP1
LOOP0:
CMP r1, r3                 @ copy the content of ".data" data
segment in program to the read-write area
LDRCC r2, [r0], #4
STRCC r2, [r1], #4
BCC LOOP0
LOOP1:
LDR r1, =Image_ZI_Limit    @ commence from the top of the clear
area
MOV r2, #0
LOOP2:
CMP r3, r1                 @ clear
STRCC r2, [r3], #4
BCC LOOP2

#=====
# initialize stack space
#=====
INITIALIZE_STACK:
MRS r0, cpsr
BIC r0, r0, #LOCKOUT | MODE_MASK

ORR r2, r0, #USR_MODE

```

```

ORR r1, r0, #LOCKOUT | FIQ_MODE
MSR cpsr_cf, r1
MSR spsr_cf, r2
LDR sp, =FIQ_STACK @ set fast interrupt stack space

ORR r1, r0, #LOCKOUT | IRQ_MODE
MSR cpsr_cf, r1
MSR spsr_cf, r2
LDR sp, =IRQ_STACK @ set interrupt stack space
ORR r1, r0, #LOCKOUT | ABT_MODE
MSR cpsr_cf, r1
MSR spsr_cf, r2
LDR sp, =ABT_STACK @ set abnormal stack space

ORR r1, r0, #LOCKOUT | UDF_MODE
MSR cpsr_cf, r1
MSR spsr_cf, r2
LDR sp, =UDF_STACK @ set undefined abnormal stack space

ORR r1, r0, #LOCKOUT | SUP_MODE
MSR cpsr_cf, r1
MSR spsr_cf, r2
LDR sp, =SUP_STACK @ set superuser stack space

#=====
# switch to user mode and set user stack space
#=====
MRS r0, cpsr
BIC r0, r0, #LOCKOUT | MODE_MASK
ORR r1, r0, #USR_MODE
MSR cpsr_cf, r0
LDR sp, =USR_STACK

#=====
# enter into C language program entry
#=====
.extern __main
BL __main

#=====
#definition of vector function, definition of internal function in C
program
#=====

```

```

SystemUndefinedHandler: B SystemUndefinedHandler
SystemSwiHandler:      B SystemSwiHandler
MakeSVC:              B MakeSVC
SystemPrefetchHandler: B SystemPrefetchHandler
SystemAbortHandler:   B SystemAbortHandler
SystemReserv:         B SystemReserv

SystemIrqHandler:     @ interrupt vector
    .extern ISR_IrqHandler
    STMTD  sp!, {r0-r12, lr} @ interrupt stack saving
    BL  ISR_IrqHandler      @ interrupt C manipulation function
    LDMFD sp!, {r0-r12, lr} @ recover interrupt stack
    SUBS  pc, lr, #4        @ return to the program index before
interrupt

SystemFiqHandler:     @ fast interrupt vector
    .extern ISR_FiqHandler
    STMTD  sp!, {r0-r7, lr} @ fast interrupt stack saving
    BL  ISR_FiqHandler      @ fast interrupt C manipulation function
    LDMFD  sp!, {r0-r7, lr} @ fast recover interrupt stack
    SUBS  pc, lr, #4        @ return to the program index before fast
interrupt

#=====
# memory area register setting value
#=====

SystemInitDataSDRAM:
    .long 0x00003E02 @ EXTDBWTH setting value
    .long 0x02000060 @ ROMCOONO setting value, 0~0x200000
    .long 0x00000060
    .long 0x00000060
    .long 0x00000060
    .long 0x00000060
    .long 0x00000060
    .long 0x14010301 @ DRAMCONO setting value,
0x400000~0x1400000
    .long 0x00000000
    .long 0x00000000
    .long 0x00000000
    .long 0x9C298360 @ REFEXTCOM setting value

#=====
# stack space definition interval
#=====

```

```

.data
    .SPACE    1024
USR_STACK:  .SPACE    512
UDF_STACK:  .SPACE    512
ABT_STACK:  .SPACE    512
IRQ_STACK:  .SPACE    512
FIQ_STACK:  .SPACE    512
SUP_STACK:

```

- **C master program file**

C master program file "led_int.c" completes the initialization of IO port and interrupt, realization of interrupt function, and handling to the uncompleted fast interrupt in the program, and remain an empty function of fast interrupt handling for the need of link.

The source code and detailed interpretation of "Led_int.c" are as follows:

```

/*****
* file name:    led_int.c
* description  S3C4510 control IO and interrupt demonstration program
*/
*
*      P16/TOUT0 connected to LED1
*      P17/TOUT1 connected to LED2
*
*      P9/XIRQ1  connected to button
*****/
#define VPint      *(volatile unsigned int *)

#define Base_Addr  0x3ff0000                // register base address

#define IOPMOD     (VPint(Base_Addr+0x5000)) // IO mode register
#define IOPCON     (VPint(Base_Addr+0x5004)) // IO control register
#define IOPDATA    (VPint(Base_Addr+0x5008)) // IO data register

#define INTMOD     (VPint(Base_Addr+0x4000)) // interrupt mode register
#define INTPEND    (VPint(Base_Addr+0x4004)) // interrupt pending
register
#define INTMASK    (VPint(Base_Addr+0x4008)) // interrupt mask register
#define INTOFFSET  (VPint(Base_Addr+0x4024)) // interrupt shift register

```

```

void ISR_IrqHandler(void);
void ISR_FiqHandler(void);

/*****
* name:    __main
* function: C language entry master function
*****/
void __main(void)
{
    int i, j;

    IOPMOD = 0x00030000; // set P16 and P17 as output, other IO port as
input
    IOPCON = 0x320;      // set P9 as external interrupt, high level
advancing edge triggering
    INTMOD = 0x0;        // interrupt is IRQ mode
    INTMASK = 0x3FFFD;  // mask interrupts except XIRQ

    for(;;)
    {
        IOPDATA = 0x00010000; // light LED1
        for(i=0; i < 65000; i++); // simple delay
        IOPDATA = 0;          // put out LED1
        for(i=0; i < 65000; i++);
    }
}

/*****
* name:    ISR_IrqHandler
* function: interrupt handling function
*****/
void ISR_IrqHandler(void)
{
    unsigned int    IntOffSet;

    IntOffSet = (unsigned int)INTOFFSET; // obtain interrupt source
    INTPEND = 1 << (IntOffSet >> 2);    // remove interrupt pending
symbol
    IOPDATA = 0x00020000;                // light LED2
}

```

```
/******  
* name:    ISR_FiqHandler  
* function: fast interrupt handling function  
*****/  
void ISR_FiqHandler(void)  
{  
  
}
```

3.3.2 Creation of Project

Select menu File > New Workspace, the system will pop up project creation dialog, as shown in Fig 3-35:

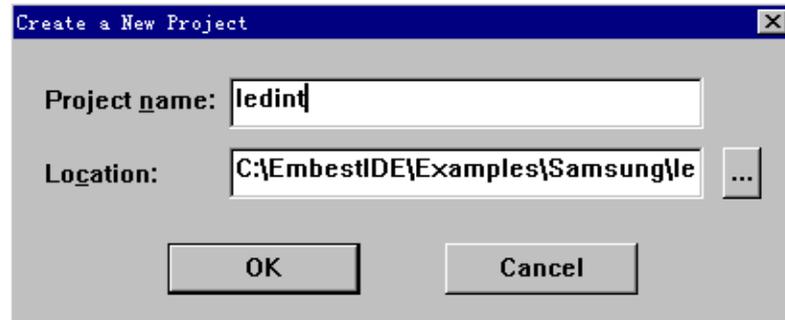


Fig 3-35 project creation dialog

Input the project name of the newly-created project "led_int" in the edit box of project name, and input the directory path "C:\EmbestIDE\Examples\Samsung\ledint" in the edit box of project location.

Select button OK to create new project ledint, and the integration environment will create the workspace and project with the same name as the project.

Select the right key menu to create source file folder in the window of workspace and add relevant source file, at last, the workspace is shown as Fig 3-36:

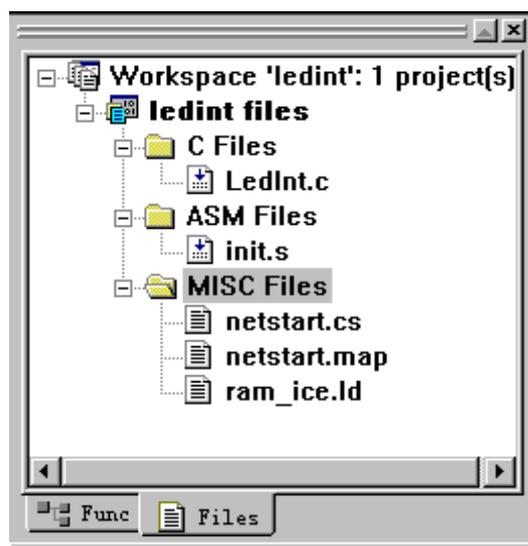


Fig 3-36 window of workspace

3.3.3 Configuration of Project

The processor configuration and emulator configuration in the project is the same as in the example described in Part 1 of this chapter, what needs to pay attention to is that, during the course of processor configuration, for Maker in Peripheral Register, select S3C4510B, for Chip, select S3C4510B, because the example is based on S3C4510B chip of Samsung Corporation, as shown in Fig 3-37, and the please set other configurations according to the following steps.

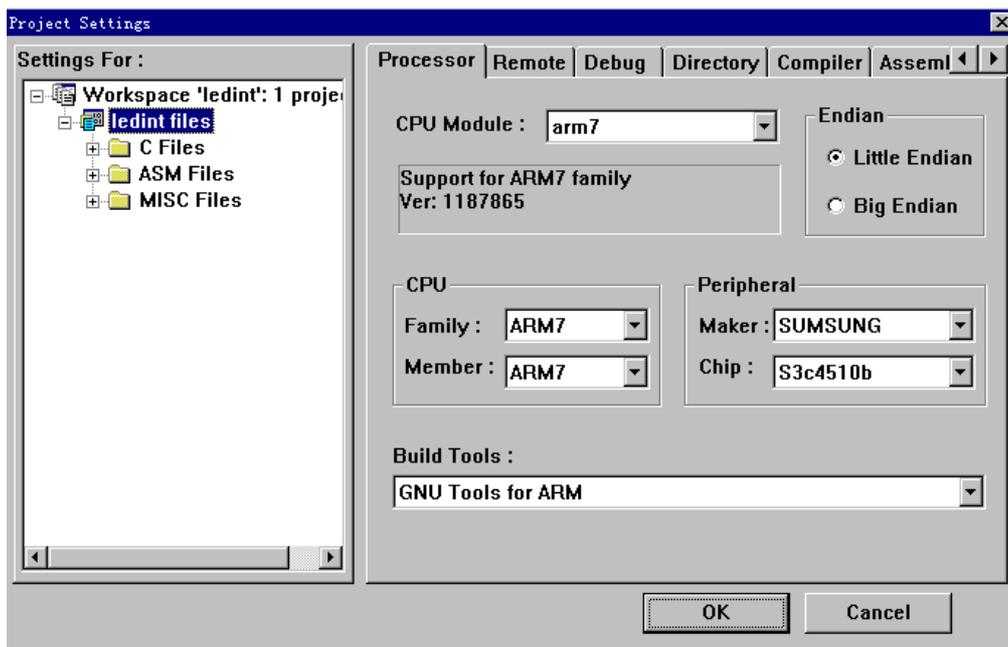


Fig 3-37 project configuration processor page layout

- Debugging equipment configuration

Select project configuration dialog box to debug equipment debug page layout, as shown in Fig 3-38:

- Select General option in Category
- ✧ set debugging symbol file as ./debug/ledint.elf;
- ✧ if debugged in RAM, set "Action after connected" of emulator as command script, then select the command script file to implement as "net-start.cs";
- ✧ if debugged in ROM, set "Action after connected" of emulator as "none";

command script file is a serial commands for integration environment to implement, which will usually complete the initialization work needed by processor, including reset, memory area configuration, and disable interrupt, etc, users shall use command script file to map RAM area to 0 address when debugging in RAM in this example, which is the same with the address of Flash

to be solidified to finally, so that the users download program to O address to debug whether the phenomenon is consistent to the actual operation; the command script "net-start.cs" used in this example and the interpretation is as follows:

```

reset                ; reset processor
stop                 ; stop processor operation
memwrite 0x3ff4008 0xffffffff ; remove all interrupts
memwrite 0x3ff4004 0xffffffff ; mask all interrupts
memwrite 0x3ff0000 0x83FFF90  ; config. System register
memwrite 0x3ff3010 0x00003e02 ; config. Access width register
memwrite 0x3ff3014 0x1a060040 ; config.ROM0 (0x1800000-0x1a00000)
memwrite 0x3ff302C 0x10000301 ; config.RAM0从0x0到0x1000000
memwrite 0x3ff303C 0x9c298360 ; config. DRAM brush parameter register

```

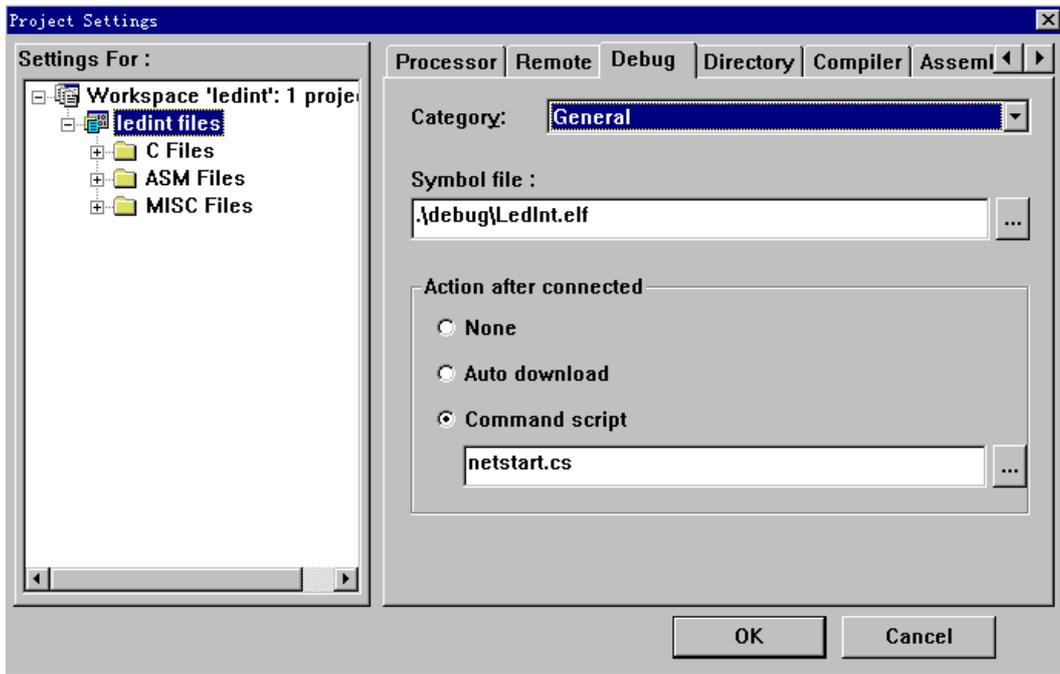


Fig 3-38 general option of project configuration debugging page layout

- select Download option in Category, as shown in Fig 3-39:
- ✧ set download file as ./debug/ledint.elf;
- ✧ set download address as 0x0;
- ✧ other default settings used;

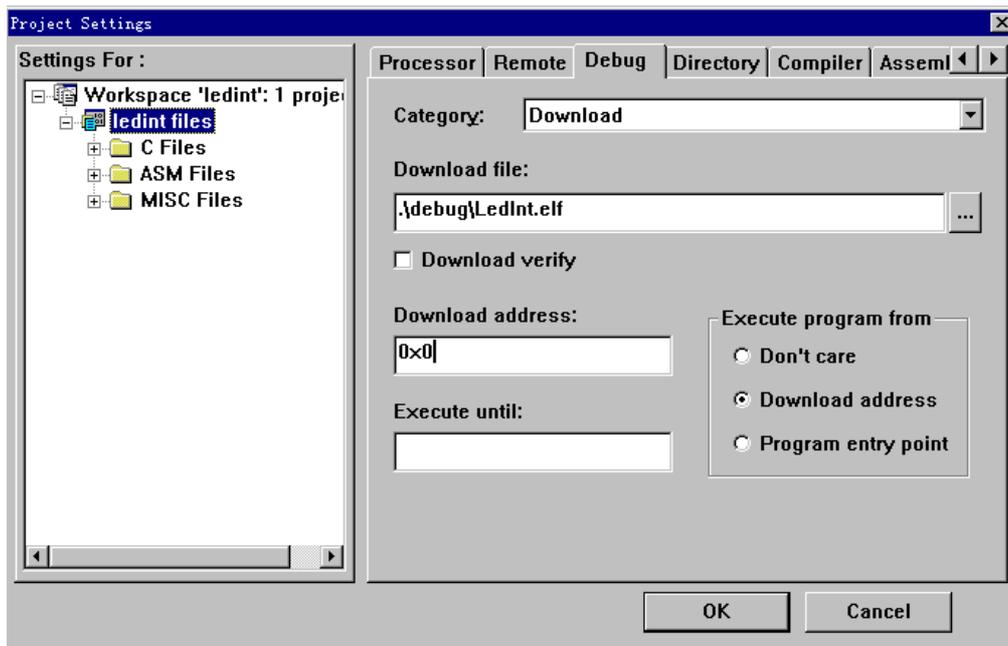


Fig 3-39 download option of project configuration debugging configuration page layout

➤ select debugging option Memory Maps in Category, as shown in Fig 3-40:

✧ Set Memory Map as No map file;

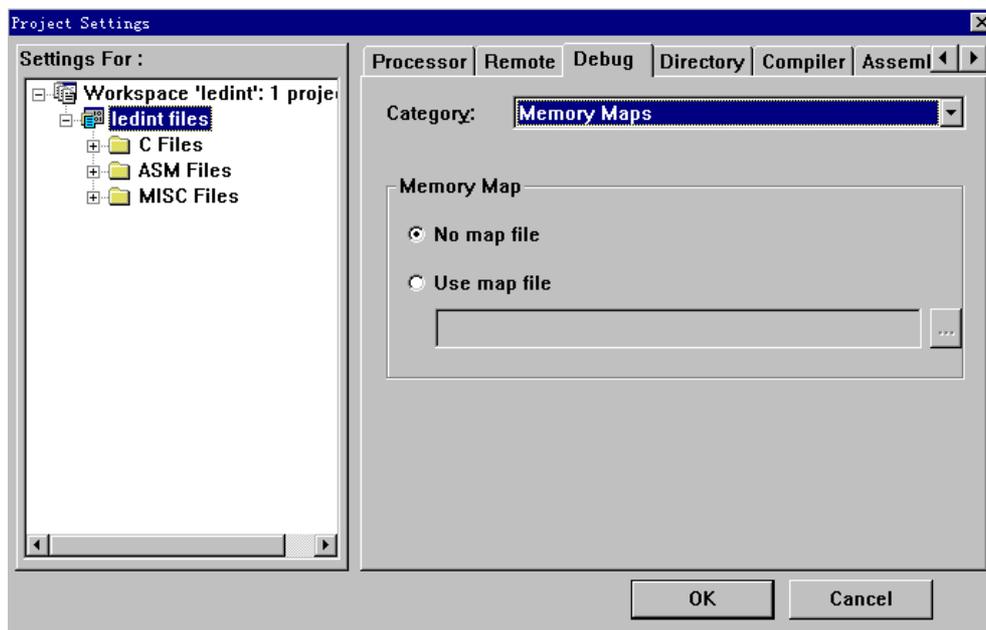


Fig 3-40 memory mapping option of project configuration debugging page layout

Memory mapping file “.map” file is used for debugging in integration environment. During the process of software debugging, it will produce abnormalities in some processors to access illegal memory area, if the abnormalities fail to be handled, they will cause the software debugging process

not able to continue; to use memory area mapping file (*.map) may prevent the above problems and adjust emulator access speed to reach the moderate level.

- compiling configuration

select project configuration dialog box compiling configuration (Compiler) page layout, as shown in Fig 3-41:

- select General option in Category
- ✧ set object files location as .\debug;
- other default settings used;

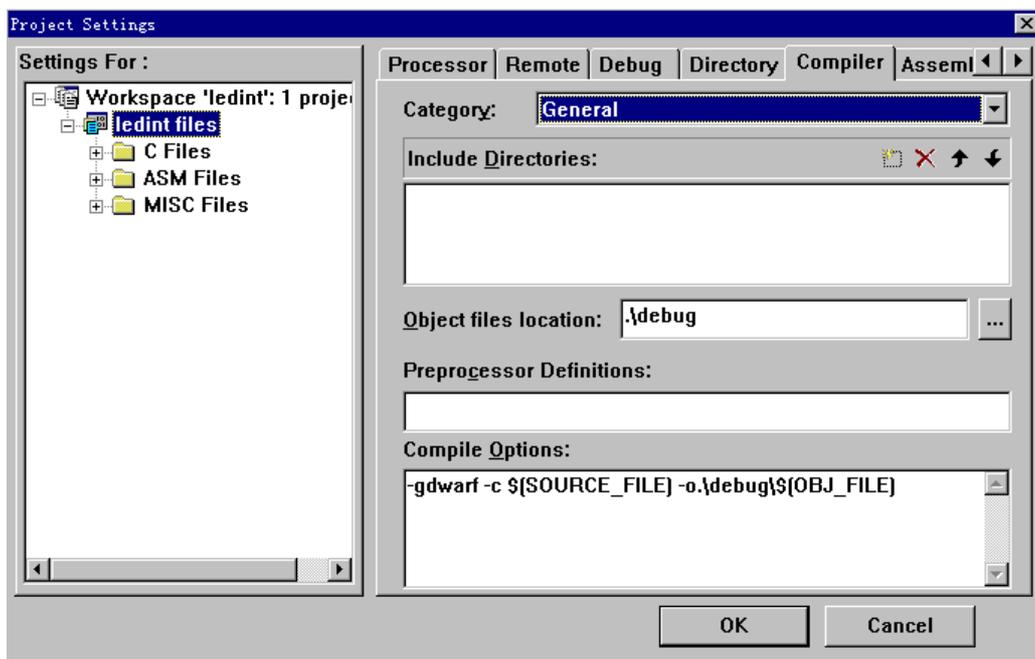


Fig 3-41 general option of project configuration compiling page layout

- assembly configuration

select project configuration dialog box assembly configuration (Assembler) page layout, as shown in Fig 3-42:

- select General option in Category
- ✧ Set object files location as .\debug:
- ✧ Set predefine as ROM=1 if it needed to build final solidification program and debug in ROM; do not set any predefine for debugging in RAM.
- other default settings used;

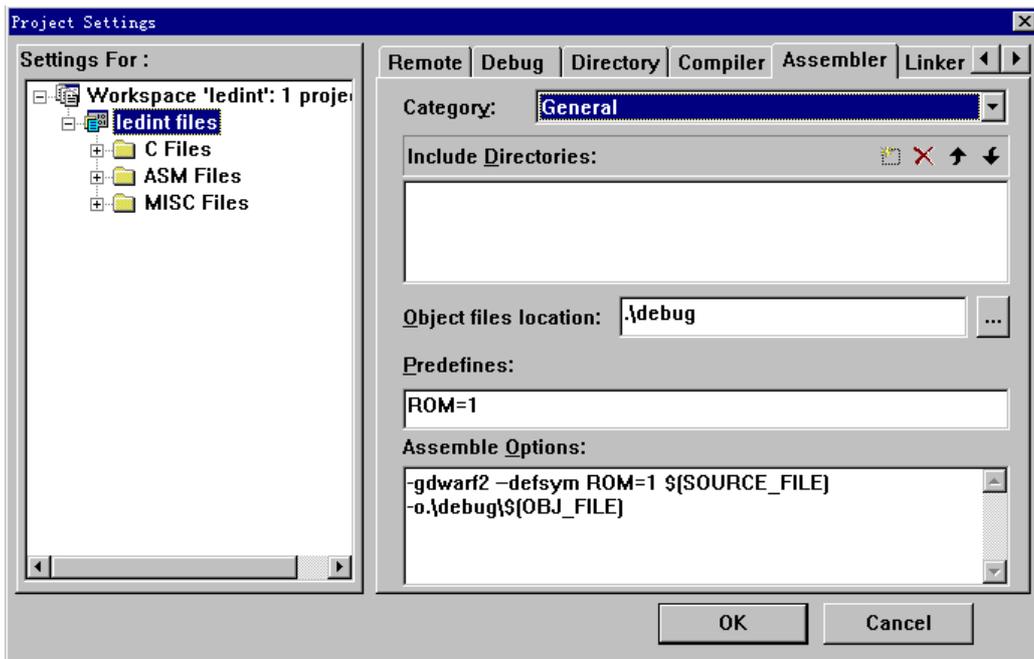


Fig 3-42 general option of project configuration assembly configuration page layout

- link configuration

select project configuration dialog box link configuration (Linker) page layout, as shown in Fig 3-43:

- select General option in Category
- ✧ set linker script file as flash.ld when solidifying program or debugging in ROM;
- ✧ set linker script file as ram_ice.ld when debugging program in RAM;
- ✧ set output file name as .\debug\ledint.elf;
- other default settings used;

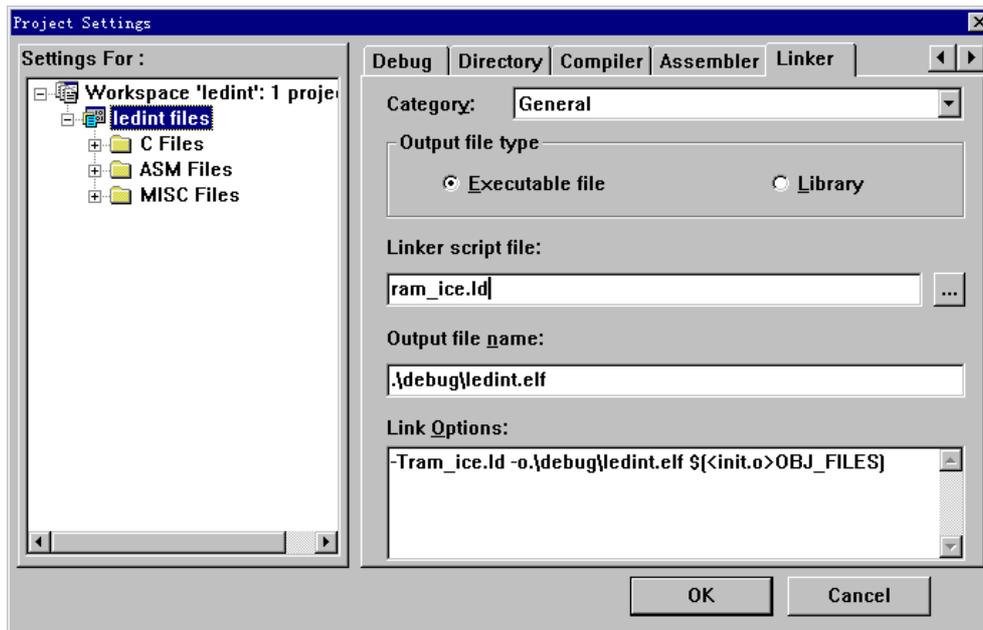


Fig 3-43 general option of project configuration link page layout

link location file shall be used in embed development of system level, and the file describes the relevant information of code link location, including code segment, data segment address, etc, the linker shall use the file to make correct location to the codes of the whole system. The link location files used for RAM debugging and solidification are different, and the following are the link location files used for solidification:

```

SECTIONS
{
. = 0x000000;          set the current address as 0
.text : { *(.text) };  code segment, symbol lay program code here from 0
.rodata : { *(.rodata) }; read only data segment, the fixed values such as
static global variable in program is laid in this segment
    Image_RO_Limit = .;  read only area length, the symbol used in starting
program
. = 0x0400000;        set the current address as 0x400000
    Image_RW_Base = .;  read and write area base address, the symbol used in
starting program
.data : { *(.data) };  data segment, the initialized global variable in
program is laid in this segment
    Image_ZI_Base = .;  clear area base address, the symbol used in starting
program
.bss : { *(.bss) };    contain uninitialized globally useable data, such as
uninitialized global variable
    Image_ZI_Limit = .;  clear area length, the symbol used in starting program
end = .;              end address
.debug_info           0 : { *(.debug_info) };  debugging information output
segment
.debug_line           0 : { *(.debug_line) }
.debug_abbrev         0 : { *(.debug_abbrev)}
.debug_frame          0 : { *(.debug_frame) }
}

```

the link location file used for debugging in RAM:

```
SECTIONS
{
  . = 0x000000;          set the current address as 0
  .text : { *(.text) };  code segment, symbol lay program code here from 0
  Image_RO_Limit = .;   read only area length, the symbol used in starting
program
  Image_RW_Base = .;    read and write area base address, the symbol used in
starting program
  .rodata : { *(.rodata) }; read only data segment, the fixed values such as
static global variable in program is laid in this segment
  .data : { *(.data) };  data segment, the initialized global variable in program
is laid in this segment
  Image_ZI_Base = .;   clear area base address, the symbol used in starting
program
  .bss : { *(.bss) };   contain uninitialized globally useable data, such as
uninitialized global variable
  Image_ZI_Limit = .;   clear area length, the symbol used in starting program
  end = .;              end address
  .debug_info          0 : { *(.debug_info) };  debugging information output
segment
  .debug_line          0 : { *(.debug_line) }
  .debug_abbrev        0 : { *(.debug_abbrev)}
  .debug_frame         0 : { *(.debug_frame) }
}
```

3.3.4 Debugging of Program in RAM

Debugging of software may be completed in ROM area or RAM area, because it is convenient to read and write in RAM area and the access speed is high, all the debugging during the process of software development shall be completed in RAM area if only the hardware condition allows.

The following steps shall be completed before debugging of software: compile link project, to connect emulator and target board, and download program.

- compile link project

Users select Build menu, compile corresponding file or project, and output relevant compiling and link information at the Build subwindow of the Output window. According to link configuration, led_int.elf file will be built under .\led_int\debug\ directory after passing of program compiling, and the file contains the execution file of debugging information.

- connect emulator and target board

Select Remote Connect submenu on Debug menu, and the debugger in integration environment will be connected to the target system through emulator.

- download program

After connection of target system, if automatic download option is set in the debugging configuration option, the debugger will automatically download software; or it will select Download submenu download program of the menu Debug. Now, the debugger will download binary system instruction file to the location designated by the target board memory area after removing the debugging information in led_int.elf, meanwhile, will display the download progress on the status bar. In this example, the download address set is 0x0, and the address is the initial address of RAM memory area through command script mapping. After download successfully, the status bar will display "Download Completed" in blue, or the information of "Download Failed" will be displayed in red status bar.

The debugging interface of Embest IDE after download of program is shown in Fig 3-44, and now the debugging of program may be started.

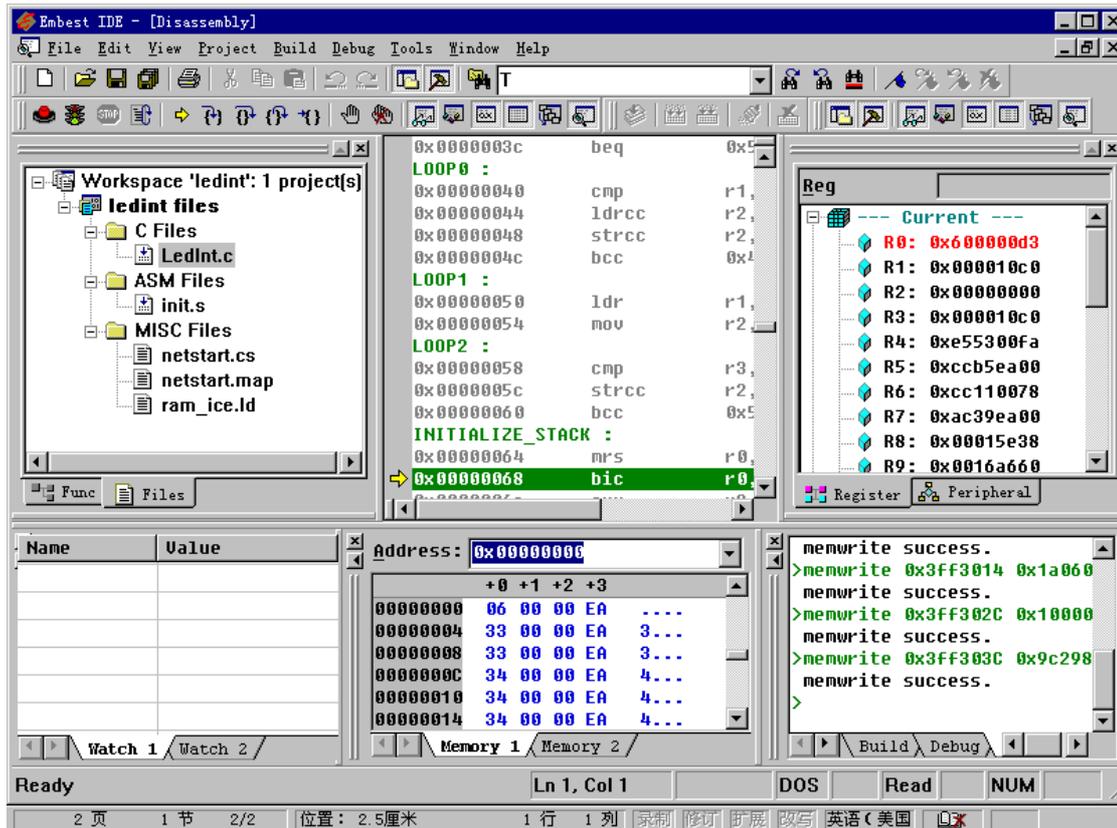


Fig 3-44 Embest IDE debugging interface

Embest IDE will display the corresponding assembly instruction at the current download address after download of program, and to select "Go to source" submenu on the right key menu may switch to the window of the source file.

Now it may, through setting breakpoint at the window of anti-assembly file or source file, implement the operation such as single step run to debug program and analyze the problems in program, as shown in Fig 3-45.

```
void ISR_FiqHandler(void);

/*****
* Func Name: __main
* Function: master function & Entry of program
*****/
void __main(void)
{
    int i, j;

    IOPMOD = 0x00030000;           // set P16,P17: output; other IO: input
    IOPCON = 0x320;               // set P9: extral Int, interrupt at high leve

    INTMOD = 0x0;                 // IRQ mode
    INTMASK = 0x3FFFFD;          // Mask Int except XIRQ1

    for(;;)
    {
        IOPDATA = 0x00010000;     // lit LED1

        for(i=0; i < 65000; i++) ; // delay
    }
}
```

Fig 3-45 set breakpoint in source program

3.3.5 Download onto the Flash ROM

The program passed through debugging in RAM is different from the program finally downloaded onto Flash ROM, and users shall:

- Set ROM=1 in the predefine option of assembly, or directly add ".equ ROM 1" in init.s file, and the starting file rather than command script will complete the re-mapping of memory area.
- Select the link file flash.ld in Linker, and the link file and starting file will combine to complete the handling job of data segment downloaded onto Flash originally.

After completion of the above modification, compile program again. Then use Elf to Bin tool to change ledint.elf file into binary format file ledint.bin.

At last, use Embest Flash Programmer to download ledint.bin to the Flash ROM, as show in Fig 3-46.

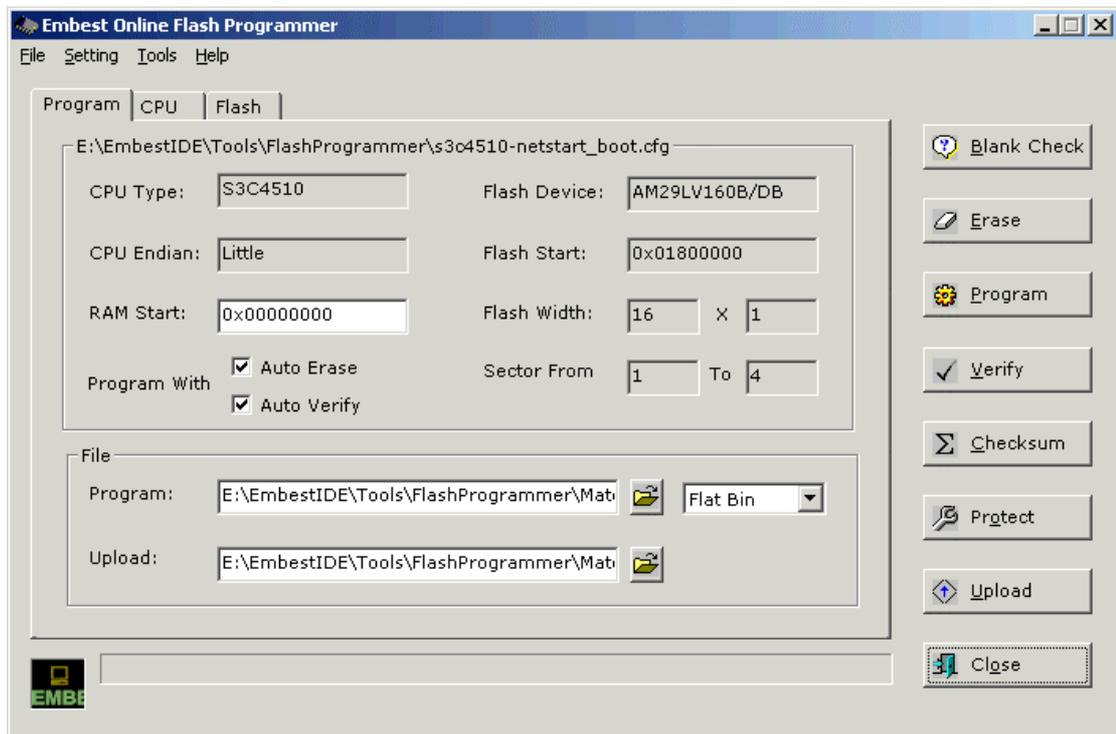


Fig 3-46 Flash programmer interface program page layout

3.3.6 Debugging of Program in Flash

When target board limited by software resources, such as the space of RAM area is less than the space of program code, which causes that debugging can not be made in RAM area, or it is needed to observe the actual operation situation of program in Flash, it may select to debug program after completing the solidification works described in the previous part.

The project configuration for debugging of program in Flash is different from that of the debugging in RAM:

- it is unnecessary to implement script file in debugging option, the work is completed during the course of starting file, and it is needed to change the option "Action after connected" into "None";

And the debugging processes are different too;

- after connecting to emulator, it unnecessary to implement Download program operation
- if it is to commence debugging program at the entry of starting program, reset command shall be implemented, and now the program will stop at the zero address;
- two hardware breakpoints may be set at most when program is debugged in Flash.

3.3.7 Startup Program Design of S3C44B0X

S3C44B0 starting program design is different from the starting program of S3C4510 as we describe in the above example, the major reason lies in that S3C44B0 has no memory mapping function, all the addresses of memory area are fixed, in addition, S3C44B0 provides vector interrupt function and reduces interrupt delay. Therefore, in the starting program of S3C44B0, the appearance of vector interrupt function leads to extension of vector table, meanwhile, for the convenience of design of program and the debugging in RAM, interrupt entry is shifted to the tip end of RAM through address definition mode.

The starting program of S3C4510 represents the starting flow of processor chips of most integrate ARM, and S3C44B0 represents the rest, to understand the starting program of these two processors is helpful to the starting design of other processors.

The following codes are the source codes and the interpretation of the starting program of S3C44B0, in order to effectively apply the space, some similar interrupt entry definitions and function macro definitions are omitted, and the omitted parts is replaced by "....." and notes are given, if users want to use the following source codes as starting programs, they shall add the omitted parts by themselves.

```
# *****
# file name: INIT.S
# description: S3c44b0x starting file
# *****

# =====
# register definition and bit definition
# =====

.equ  INTMSK,      0x01e0000c
.equ  WTCON,       0x01d30000

.equ  CLKCON,      0x01d80004
.equ  LOCKTIME,    0x01d8000c

.equ  FIQMODE,     0x11
.equ  IRQMODE,     0x12
.equ  SVCMODE,     0x13
.equ  ABORTMODE,   0x17
```

```

.equ    UNDEFMODE,  0x1b
.equ    MODEMASK,   0x1f

.equ    NOINT,      0xc0

# =====
# interrupt handling macro
# =====
    .macro HANDLER HandleLabel
    sub    sp, sp, #4                @ stack space degression save jump
address
    stmfD  sp!, {r0}                @ save work register r0 to stack
    ldr    r0, =\HandleLabel        @ load interrupt entry address location
to r0
    ldr    r0, [r0]                 @ load interrupt entry address to r0
    str    r0, [sp,#4]              @ save interrupt entry address to stack
    ldmfd  sp!, {r0,pc}             @ recover work register and jump to
interrupt function
    .endm

# =====
# set ARM7 interrupt and primary exception vector
# =====
ENTRY:
    b    ResetHandler              @ implement from here after S3C4510
reset
    b    HandlerUndef              @ undefined primary exception vector
    b    HandlerSWI                @ soft interrupt vector
    b    HandlerPabort             @ obtain fetch primary exception vector
    b    HandlerDabort            @ obtain data primary exception vector
    b    .                          @ keep
    b    HandlerIRQ                @ interrupt vector
    b    HandlerFIQ               @ fast interrupt vector

# =====
# set 44B0 interrupt vector table
# =====
VECTOR_BRANCH:
    ldr  pc,=HandlerEINT0          @ mGA H/W interrupt vector table
    ldr  pc,=HandlerEINT1          @
    .....                          @ omit
    ldr  pc,=HandlerADC            @ mGKB
    .....                          @ omit

```

b .

```
#=====
```

```
# interrupt vector handling macro
```

```
#=====
```

```
HandlerFIQ:      HANDLER HandleFIQ
HandlerIRQ:      HANDLER HandleIRQ
HandlerUndef:   HANDLER HandleUndef
HandlerSWI:      HANDLER HandleSWI
HandlerDabort:  HANDLER HandleDabort
HandlerPabort:  HANDLER HandlePabort
```

```
HandlerADC:      HANDLER HandleADC
..... @ omit
```

```
HandlerEINT1:   HANDLER HandleEINT1
```

```
HandlerEINT0:   HANDLER HandleEINT0
```

```
#=====
```

```
# interrupt vector handling macro
```

```
#=====
```

```
ResetHandler:
```

```
ldr    r0,=WTCON           @ watchdog forbidden
ldr    r1,=0x0
str    r1,[r0]
```

```
ldr    r0,=INTMSK
ldr    r1,=0x07ffffff      @ all interrupts forbidden
str    r1,[r0]
```

```
#=====
```

```
# set clock control controller
```

```
#=====
```

```
ldr    r0, =LOCKTIME
ldr    r1, =0xffff
str    r1, [r0]
```

```
ldr    r0, =CLKCON
ldr    r1, =0x7ff8         @ clock of all modules unlocked
str    r1, [r0]
```

```
#=====
```

```
# set memory area controller
```

```
#=====
```

```

ldr    r0, =SMRDATA
ldmia  r0, {r1-r13}
ldr    r0, =0x01c80000
stmia  r0, {r1-r13}
#=====
# initialize stack space
#=====
ldr    sp, =SVCStack      @ switch to superuser stack space
bl    InitStacks

#=====
# introduce external symbol, symbol definition is in link script file
#=====
.extern Image_RO_Limit    @ size of read only area
.extern Image_RW_Base     @ initial address of readable-writable
memory area
.extern Image_ZI_Base     @ initial address of clear area
.extern Image_ZI_Limit    @ size of clear area

#=====
# initialize the memory area needed to use by C code
#=====
LDR    r0, =Image_RO_Limit @ obtain the size of read only area
LDR    r1, =Image_RW_Base @ obtain the initial address of readable-
writable memory area
LDR    r3, =Image_ZI_Base @ obtain the initial address of clear area
CMP    r0, r1 @ compare whether the read only area and the
readable-writable area are overlapped
BEQ    LOOP1
LOOP0:
CMP    r1, r3 @ copy the content of ".data" data segment in program to
the readable-writable area
LDRCC  r2, [r0], #4
STRCC  r2, [r1], #4
BCC    LOOP0
LOOP1:
LDR    r1, =Image_ZI_Limit @ commence from the top of the clear
area
MOV    r2, #0
LOOP2:
CMP    r3, r1 @ clear
STRCC  r2, [r3], #4

```

```

BCC    LOOP2

#=====
# enter into C language program entry
#=====
# .extern __main
# BL    __main

#=====
# initialize the function of stack space
#=====
InitStacks:
    mrs    r0, cpsr
    bic    r0, r0, #MODEMASK
    orr    r1, r0, #UNDEFMODE | NOINT
    msr    cpsr_cxsf, r1
    ldr    sp, =UndefStack        @ set undefined abnormal stack space

    orr    r1, r0, #ABORTMODE|NOINT
    msr    cpsr_cxsf, r1
    ldr    sp, =AbortStack        @ set abnormal stack space

    orr    r1, r0, #IRQMODE|NOINT
msr    cpsr_cxsf, r1
    ldr    sp, =IRQStack          @ set interrupt stack space

    orr    r1, r0, #FIQMODE|NOINT
    msr    cpsr_cxsf, r1
    ldr    sp, =FIQStack          @ set fast interrupt stack space

    bic    r0, r0, #MODEMASK|NOINT
    orr    r1, r0, #SVCMODE
    msr    cpsr_cxsf, r1
    ldr    sp, =SVCStack          @ set superuser stack space

    mov    pc,lr                  @ function return

#=====
# setting value of relevant register of memory area
#=====
SMRDATA:
    .long 0x11110101              @ memory area access width control
register

```

```

.long 0x00000600      @ BANK0 control register
.long 0x00007FFC      @ BANK1 control register
.long 0x00007FFC      @ BANK2 control register
.long 0x00007FFC      @ BANK3 control register
.long 0x00007FFC      @ BANK4 control register
.long 0x00007FFC      @ BANK5 control register
.long 0x00018000      @ BANK6 control register
.long 0x00018000      @ BANK7 control register
.long 0x00860459      @ SDRAM brush control register
.long 0x10             @ SDRAM memory area size
.long 0x20             @ BANK6 SDRAM mode register
.long 0x20             @ BANK7 SDRAM mode register

```

```

.equ  STARTADDRESS, 0xc7fff0
# =====
# stack space definition
# =====
.equ  UserStack,    STARTADDRESS-0x500      @ c1(c7)ffa00
.equ  SVCStack,     STARTADDRESS-0x500+256  @ c1(c7)ffb00
.equ  UndefStack,   STARTADDRESS-0x500+256*2 @ c1(c7)ffc00
.equ  AbortStack,   STARTADDRESS-0x500+256*3 @ c1(c7)ffd00
.equ  IRQStack,     STARTADDRESS-0x500+256*4 @ c1(c7)ffe00
.equ  FIQStack,     STARTADDRESS-0x500+256*5 @ c1(c7)fff00
# =====
# ARM interrupt vector entry definition
# =====
.equ  HandleReset,  STARTADDRESS
.equ  HandleUndef,  STARTADDRESS+4
.equ  HandleSWI,    STARTADDRESS+4*2
.equ  HandlePabort, STARTADDRESS+4*3
.equ  HandleDabort, STARTADDRESS+4*4
.equ  HandleReserved, STARTADDRESS+4*5
.equ  HandleIRQ,    STARTADDRESS+4*6
.equ  HandleFIQ,    STARTADDRESS+4*7
# =====
# S3C44B0 interrupt vector entry definition
# =====
.equ  HandleADC,    STARTADDRESS+4*8
..... @ omit
.equ  HandleEINT4567, STARTADDRESS+4*29
.equ  HandleEINT3,  STARTADDRESS+4*30

```

```
.equ  HandleEINT2,  STARTADDRESS+4*31
.equ  HandleEINT1,  STARTADDRESS+4*32
.equ  HandleEINT0,  STARTADDRESS+4*33  @ 0xc1(c7)fff84
```

4. Editor

Embest development environment includes an integrated Text editor to manage, edit, and print source files.

4.1 Editor Overview

The EmbestIDE source code editor includes standard text manipulation capabilities, as well as the following specialized features:

- ◇ C and assembly syntax color highlighting.
- ◇ Debugger integration: the editor window tracks code execution.
- ◇ Compiler integration: compiler messages links to the editor window.

The EmbestIDE editor also provides features tailored to the program environment. Editor display program syntactic elements such as C keywords, preprocessor directives, and comments in color. Because the editor is integrated with the debugger, the editor also keeps pace automatically with program execution during debugging session.

You can work on as many files simultaneously as your computer's memory allows. This is convenient for you are working with more than one module, or want to edit both a source and header file at the same time.

Typically, developers use the EmbestIDE editor to work with source files and header files. However, because it is a text editor, it can also be used on any text file. For example, you can view a bug report, or save a note in a text file.

The EmbestIDE editor uses standard Windows editing commands and conventions. Most of the procedures for using the editor should seem familiar if you have used other Windows-based text editors. With the Text editor, you can:

- ◇ Perform advanced find and replace operations in a single file or multiple files, including using regular expressions and incremental searching.
- ◇ Use Bookmarks to mark frequently accessed lines in your source file.
- ◇ Customize the Text editor with the selection margin, indent, and drag and drop.
- ◇ Select lines, multiple lines, or columns.
- ◇ Use drag-and-drop editing within one editor window, between editor windows, and between the Text editor and the debugger.
- ◇ Manage the source window.

While using the Text editor, in many instances you can right-click mouse to display a shortcut menu of frequently used commands. The commands available depend on what the mouse pointer is pointing to.

4.2 The Standard Toolbar

The Standard toolbar has buttons for frequently used editing commands that are also available in the File and Edit menus. Figure 4-1 shows the Standard toolbar.

When you first start EmbestIDE in its default configuration, the Standard toolbar appears just below. You can click View>Toolbars>Standard to show and hide the standard tool bar.



Figure4-1 Standard Tool Bar

The following are summary descriptions of each buttons and the equivalent menu option.

Table 4-1 Standard Tool Bar Buttons Description

| button | menu | description |
|---|----------------|--|
|  | File>New | Create a new file |
|  | File>Open | Open an existing file |
|  | File>Save | Save current file |
|  | File>Save All | Save all opening files |
|  | File>Print | Print current file |
|  | Edit>Cut | Delete the selection and place it into the clipboard |
|  | Edit>Copy | Copy the selection to the clipboard |
|  | Edit>Paste | Insert the clipboard text at the insertion point |
|  | Edit>Undo | Undo the previous operation |
|  | Edit>Redo | Redo the previous operation |
|  | View>Workspace | View or hide the workspace window |

| | | |
|---|---|---|
|  | View>output | View or hide the output window |
|  | Edit>Find in Files | Search strings in multiple files |
|  | Edit>Find Next | Search for another instance of the same string in the same search direction |
|  | Edit>Find Previous | Repeat the search, but in the opposite direction |
|  | Edit>Replace | Specify both a string to find and a replacement for it |
|  | Edit>More Bookmarks>Toggle Bookmark | Toggles an unnamed bookmark for the current line |
|  | Edit>More Bookmarks>Next Bookmark | Moves to the line containing the next bookmark |
|  | Edit>More Bookmarks >Previous Bookmark | Moves to the line containing the previous bookmark |
|  | Edit>More Bookmarks>Clear All Bookmarks | Clears all bookmarks in the window |

4.3 File Management

The following sections describe file management commands.

4.3.1 Create a file

Table 4-2 Operation of Create a New File

| Button | Shortcut | File menu |
|---|----------|------------|
|  | Ctrl+N | File > New |

To create a new file, click File> New, an empty window appears, ready for input text.

4.3.2 Open a File

Table 4-3 Operation of Open a File

| Button | Shortcut | File menu |
|---|----------|-------------|
|  | Ctrl+O | File > Open |

To open an existing file, click File>Open. A standard Windows file browser allows you to select which file to be open.

Recent files of the File menu (File > Recent Files) lists the most recently opened files. You can choose one of these without navigating through the open dialog box.

4.3.3 Save and Close a File

Table 4-4 Save and Close File Button

| Button | Shortcut | File menu |
|---|----------|-----------------|
|  | Ctrl+S | File > Save |
| N/a | | File > Save As |
|  | | File > Save All |
| N/a | Ctrl+F4 | File > Close |

To save current file, click File>Save. If need specify a new name (or path) for current file, click Save As instead. (Save is disabled until you modify current file.)

If you open multiple files, you can also click File>save all to save the modification of all opening files.

Click Close to dismiss the editor window for current file. If the file has changed since you last saved, a confirmation dialog box offers you the opportunity to save the file before closing it.

4.3.4 Print

Table 4-5 Print Buttons

| Button | Shortcut | File menu |
|---|----------|----------------------|
|  | Ctrl+P | File > Print |
| N/a | | File > Print Preview |
| N/a | | File > Page Setup |
| N/a | | File > Print Setup |

To print current edit window, click File>Print. A standard print dialog box (Figure 4-2) appears.

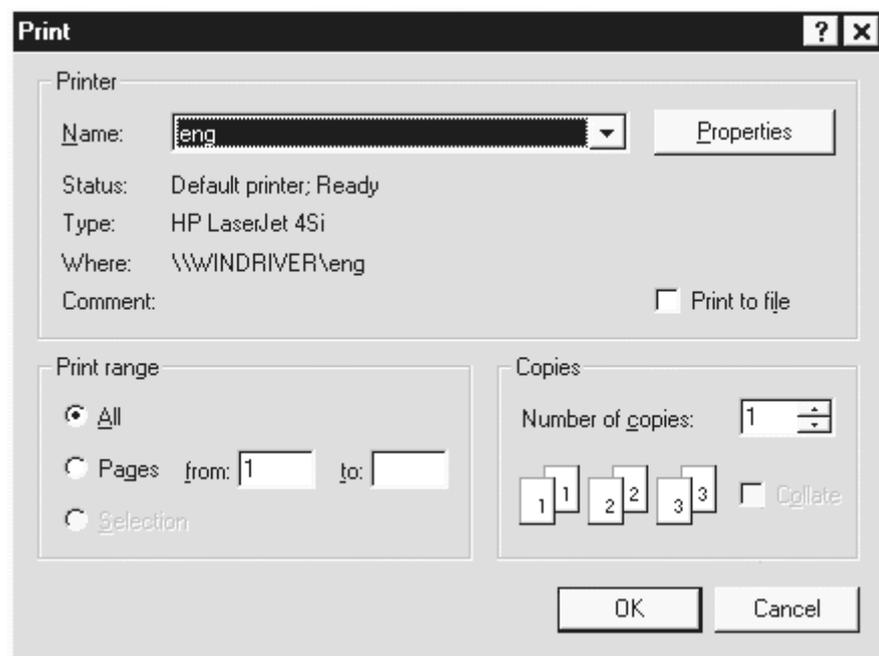


Figure 4-2 Print Dialog Box

Click File > Print Preview, to preview the page will be printed. Click File > Page Setup to change the margin size of each page in printout.

4.4 Typing and Editing

Only one edit window in EmbestIDE is active at any time. The active window contains a text cursor, a blinking vertical line also called an insertion point. Whatever you type appears at the text location indicated by the text cursor.

The editor is designed for edit source files. As such, it does not provide the "word wrap" feature found in many word edit software. You must press ENTER to start a new line. If a line is too long for the current width of the edit window, the text scrolls horizontally as necessary to display the text you are editing.

There are two edit modes in EmbestIDE: overwrite mode, which replaces the existing text under the cursor as you type, and insert mode (the default), which displaces text to the right while adding the characters you type. Use the INSERT key on your keyboard to toggle between these two modes. The edit mode does not change when you switch edit windows; the last mode you selected continues to apply, even if you switch to a window that you last edited with the other edit mode.

NOTE: If you cannot type inside an editor window, check for a READ indicator on the status bar at the bottom of EmbestIDE window. If that indicator appears, the editor is displaying as a read-only file. To enable type in that window, click Edit>Read Only menu to turn off the read-only file attribute.

4.4.1 Edit Text

Table 4-6 Edit Button

| Button | Shortcut | Edit menu |
|---|----------|--------------------|
|  | Ctrl+Z | Edit > Undo Typing |
|  | Ctrl+Z | Edit > Redo |
|  | Ctrl+X | Edit > Cut |
|  | Ctrl+C | Edit > Copy |
|  | Ctrl+V | Edit > Paste |
| | Del | Edit > Delete |
| | Ctrl+A | Edit > Select All |

The edit menu supports the Windows standard edit functions: Undo, Cut, Copy, Paste, Delete, and Select All, with standard shortcuts. With the text editor, you can cut, copy, and paste selected text using menu commands or drag-and-drop operation. You can also undo and redo selected edit actions.

The editor uses the standard Windows keys and mouse actions for moving throughout the file. For example, the following specialized keys have the standard effects:

Table 4-7 The Standard Windows Edit Keys

| | |
|-----------|---|
| Page Up | Display the previous portion of text. |
| Page Down | Display the next portion of text. |
| End | Move the cursor to the end of the line. |
| Ctrl+End | Display the end of the document. |
| Home | Move the cursor to the beginning of the line. |
| Ctrl+Home | Display the start of the document. |
| ← ↑ ↓ → | Move the cursor into the direction of the arrow, one character or line at a time. |

| | |
|----------|--|
| Ctrl + ← | Move the cursor one word at a time, in the |
| Ctrl + → | direction of the arrow. |
| Ctrl + ↑ | Scroll the window by one line, in the direction of |
| Ctrl + ↓ | the arrow, without changing cursor position. |

You can set bookmarks to mark source lines that need frequently access in your source file. Once a bookmark is set, you can use menu or keyboard commands to move to it. You can remove a bookmark when you no longer need it.

4.4.2 Find and Replace Text

Table 4-8 Find and Replace Button

| Button | Shortcut | Menu |
|---|----------|----------------------|
| N/a | Ctrl+F | Edit > Find |
|  | F3 | Edit > Find Next |
|  | Shift+F3 | Edit > Find Previous |
|  | | Edit > Find in Files |
|  | Ctrl+H | Edit > Replace |

The text editor supports string searching. You can search text in a single source file or in multiple files.

Click Edit>Find to search for a string in the current active file window. Figure 4-4 shows the Find dialog box. Enter the string you are looking for, set the options, and click find next. The option buttons under Direction determine whether the editor searches back (Up) from the cursor position, or forwards (Down). The option of match whole word only means Matches all occurrences of a text string not preceded or followed by an alphanumeric character or the underscore (_). The option of match case means Searches for text that matches the capitalization of the text string. If the text is not found, the editor displays an error message; the Find dialog box remains open, in case you need to correct the search string. To continue your search, use the Find Next or Find Previous shortcut keys, or the equivalent toolbar buttons on the standard toolbar. The default shortcut key for Find Next is F3; the default key combination for Find Previous is SHIFT+F3.

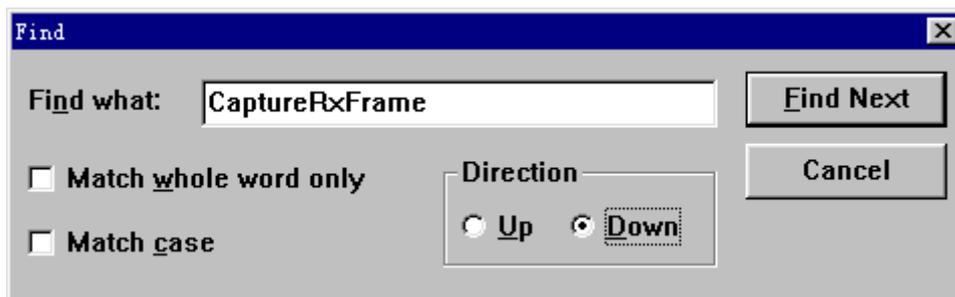


Figure 4-4 Find Dialog Box

Click Edit > Replace to specify both a string to find and a replacement for it. Figure 4-5 shows the Replace dialog box. The buttons in the Replace dialog box allow you to replace all occurrences of a string, or examine each individual occurrence before decide whether or not to replace it.

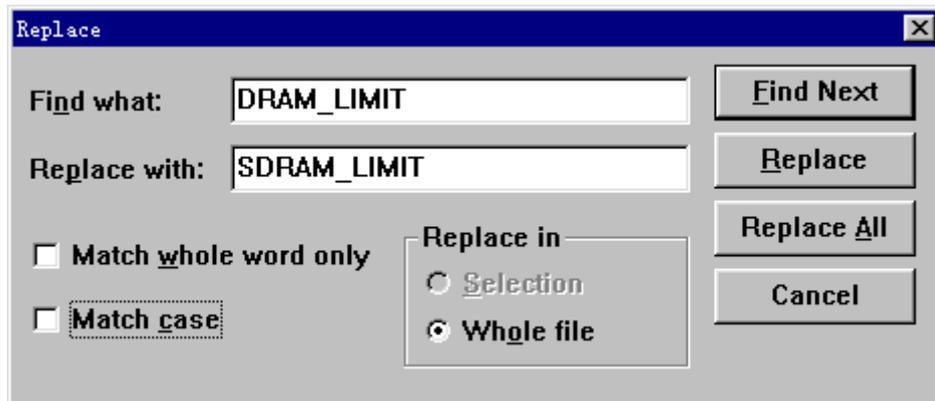


Figure 4-5 Replace Dialog Box

You can also find text in multiple files. Just click Edit > Find in Files to do this search. The Find in Files command supports two output panes. This allows you to conduct a second search through multiple files without losing the results from your first search. Figure 4-6 shows the Find in Files dialog box. In the Find what box, type the search text. In the In files/file types box, select the file types you want to search. You can use the drop-down list to select from common file types or to type text specifying other file types. In the In folder box, select the primary folder that you want to search. Click the Browse button to display the Choose Directory dialog box if you want to change drivers and directories. If necessary, select one or more of the Find options. If you want to direct the search output to a second Find in Files pane, select the Output to pane 2 check box. Click the Find button to begin the search.

The Output window displays the list of file locations where the text string appears. Each occurrence lists the fully qualified filename, followed by the line number of the occurrence and the line containing the match. To open a file that contains a match, double-click the entry in the Output window.

An editor window contains the file opens with the line contain the match selected. You can jump to other occurrences of the text string by double-clicking the specific entries in the Output window.

When you jump to a found string location specified in the Output window, the corresponding source file is loaded if it is not already open in the editor.

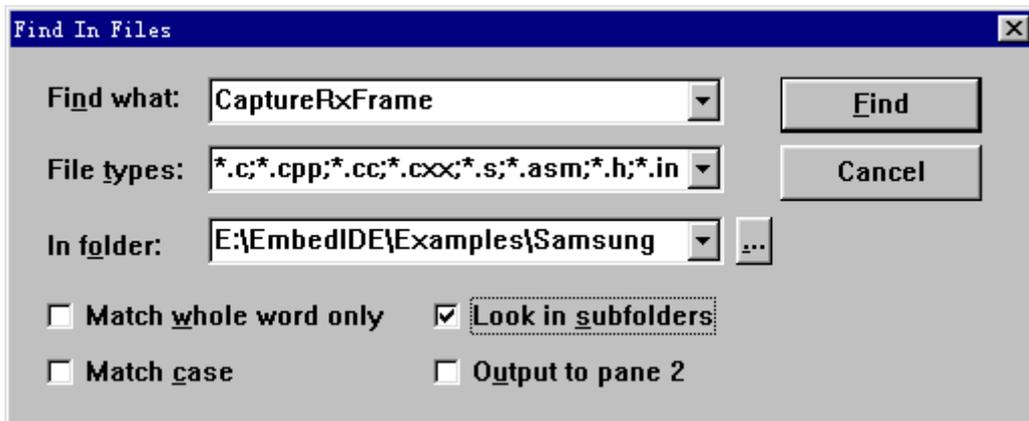


Figure 4-6 Find in Files Dialog Box

4.4.3 Hex file Editor

Embest IDE Hex file editor can open the HEX or Bin format files, and read, modify and save.

Open file: when you open file with the menu File Open, drop down file type select box and select Hex Files (*.hex;*.bin) to open the file.

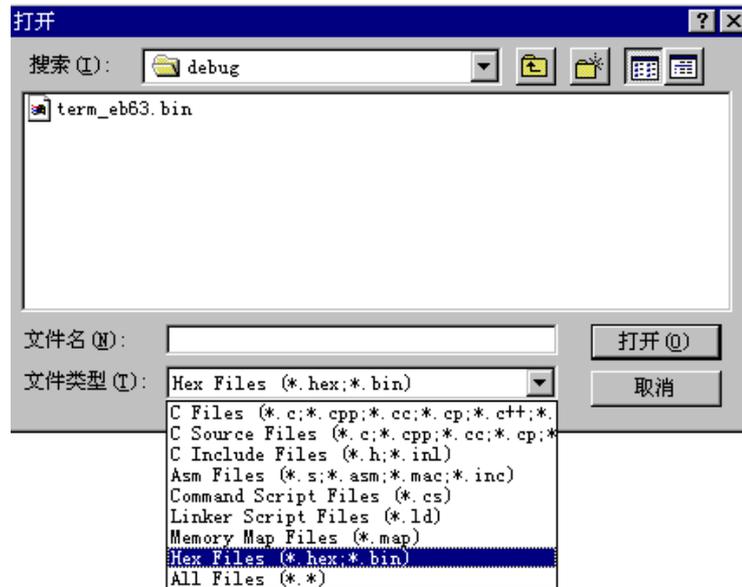


Fig. 4-7 Open Hex file dialog box

Edit file: after open Hex file, the display in Source Program Window is shown in fig. 4-8. The user can directly modify the content of file in left Hex area. The editor will automatically identify ASCII characters and display in the right. As the same, the user can also directly input ASCII code in right ASCII area.

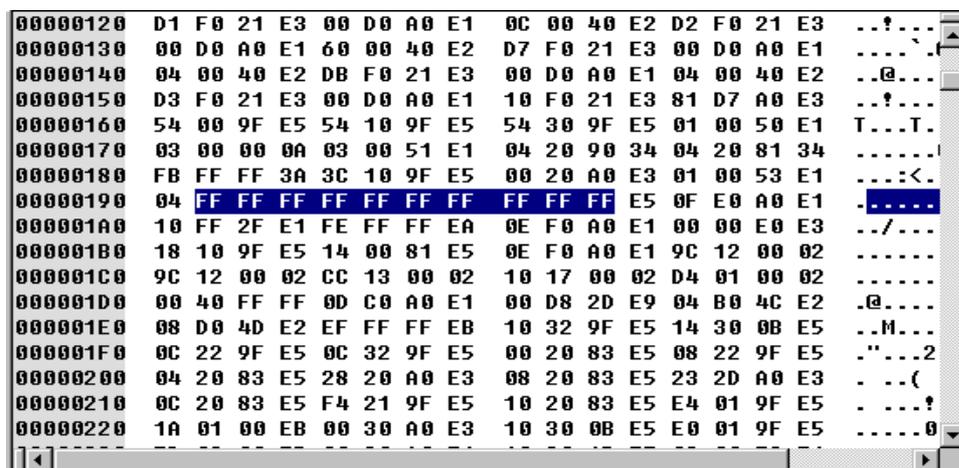


Fig. 4-8 Hex file Editor in Source Program Window

4.4.4 Find in Hex file

Hex file editor supports the forward and back finding of ASCII character string and Hex code. The user highlights Find in menu option to activate Find Dialog Box with right key of mouse in Source Program Window, or activates it through Find option in Edit menu:

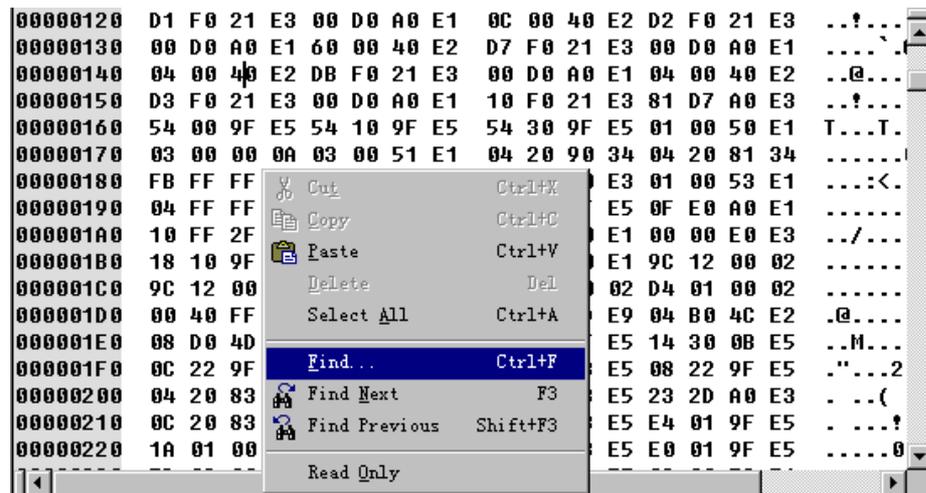


Fig. 4-9 Hex file finding interface

The user can input Hex code while finding:



Fig. 4-10 Hex file finding input dialog box

The user can also input ASCII code while finding, and highlight Find ASCII option:



Fig. 4-11 Hex binary file finding input dialog box

Hex file editor also supports copy, stick, cancel, print and print interview so that the user is easy to view Hex file.

4.5 Function List Window of Source Program

When users open C language or C ++ source file in editing window, source file function listing window will dynamically display the function contained in the current source file, and user can select the display format of function through menu.

Users can self-define the display mode of function in list:

- display detailed information of function
- only display function name
- do not display function return style
- do not display function call parameter

4.5.1 Introduction

Click the menu Tools > Options, select Extra Function Prototype in the ejected dialog box; the window Func can be found in workspace window; the function list currently opening C/C++ program is shown in the window; double click the position of function name fast positioning function in source program. As shown in fig. 4-12.



Fig 4-12 Dynamic list window of source program function

Click right key in list window to select the display style of function. The ejected right key menu is shown in fig. 4-13.

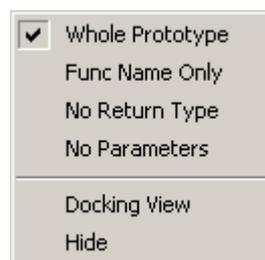


Fig 4-13 Right key menu in dynamic list window of functions

Introduction to main options of right key menu in dynamic list window of functions

Whole Prototype: display all information concerning definition of function, including return type, function name, and parameter

Func Name Only: Only display function name

No Return Type: display function name, parameter of function definition

No Parameters: display return type, function name of function definition

5. Project Management

5.1 Introduction

The project facility is a key element of EmbestIDE. It provides graphical and automated mechanisms for create applications that can be downloaded to target. A project consists of the source code files, build settings, and binary codes that are used to create a downloadable application. The project facility provides a simple means for define, modify, and maintain a variety of build options for each project. Each project requires its own directory.

In EmbestIDE, the Project Workspace is a container for your development projects. When you create a new project, a workspace is created at the same time. You can use the Project Workspace window to view and access the various elements of your projects. After you have created a project workspace, you can add new projects, including independent projects.

The workspace directory is the root directory for the project workspace. The projects you add to the project workspace can be located on other paths, even on a different drive.

The project facility provides mechanisms for:

- ◇ Organize the files that make up of a project. A project is a collection of source files, library files, and other input files. You can organize the files in a project in various ways to provide a logical structure to your source files.

- ◇ Group related projects into a workspace.

- ◇ Define varied sets of build options. The project facility provides a simpler mean for configuration and building, a project settings dialog box enable you define target processor, debug device, debug information, output file, compiling option, assemble option, linker option, and so an.

- ◇ Build applications. A build toolbar is provided, which provides access to all the major build commands.

- ◇ Download application objects to the target.

This chapter describes many of the basic tasks involving projects, such as:

- ◇ create projects

- ◇ open projects
- ◇ add files to projects
- ◇ save projects
- ◇ move files in the project window
- ◇ configure projects

5.2 GUI

Figure 5-1 and figure 5-2 shows the operation menu about project management in file menu and project menu of EmbestIDE.

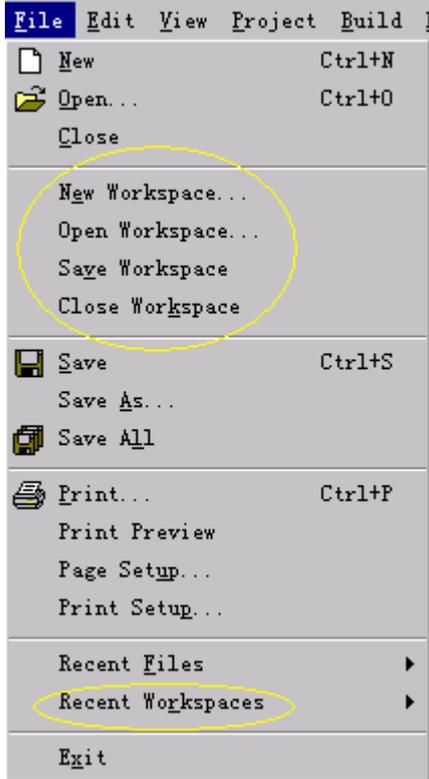


Figure 5-1 File Menu

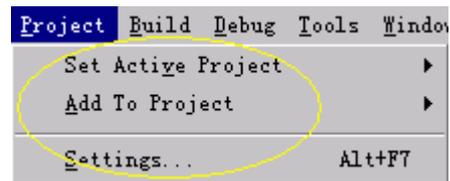


Figure 5-2 Project Menu

Operation about project management in file menu describes below:

Table 5-1 Project Management in File Menu

| File menu | Describe |
|-------------------|---|
| New Workspace... | Create a new project, a workspace |
| Open Workspace... | Open a workspace and the projects in the workspace |
| Save Workspace | Save workspace and the projects in the workspace |
| Close Workspace | Close a workspace and the projects in the workspace |
| Recent Workspace | Open a recently used project workspace |

Table 5-2 describes the project menu :

Table 5-2 Project Menus

| Project menu | Describe |
|---------------------|--------------------------------------|
| Set Active Project | Active a project |
| Add To Project | Insert file or folder into a project |
| Settings... | Configure the active project |

Workspace window is the project management window, click View>Workspace (shortcut: Alt+0) to hide or show workspace window. The workspace window shows information about the projects and the files in that project. The workspace window provides an outline view of project categories. The workspace Window is modeled after the Microsoft Windows Explorer, it shows the relationships among the files included in the project workspace. The relationships in the project workspace are logical ones, not physical ones. It does not reflect the organization of files on your hard disk.

You can highlight every level folder in the Workspace window then right-click to invoke its pop-up menu. We will describe these menus following.

To navigate the workspace window, use the vertical scroll bar on the right side of the window, or the Up and Down Arrow keys on your keyboard. If the workspace window contains many files, use the Home key to scroll to the top of the list, or use the End key to scroll to the end of the list. Use the Page Up and Page Down keys to scroll one page up or one page down the workspace window.

Workspace directory is the root directory of project workspace. Every project consists of several relative folders such as C Source Files, Assembly Source files, Link Files, Include Files. You can create new folder in the project. To do this, highlight the project file then choose Create New Folder from the shortcut menu, in the pop up new folder dialog box fill in the folder name you want to create. To delete the folder you created or the files you added just press Delete key.

5.3 Operations

5.3.1 Create a New Project

When you first create a new project, a workspace is created at the same time.

- 1) Click File>New Workspace, a dialog box pops up.

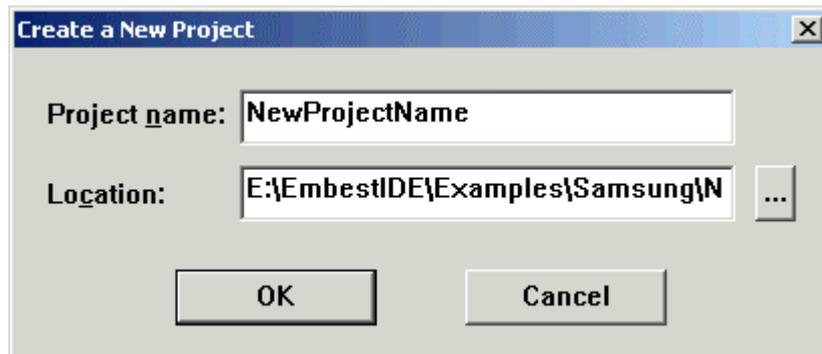


Figure 5-3 Create a New Project

- 2) Enter the new Project name and its path, and click  to browse the path.
- 3) Click Ok button, a new project will be created, and a same name workspace will be created too.

You can insert a new project into an existing project workspace too:

- 1) Open the project workspace that you want to add a new project to.
- 2) Highlight the root directory of the project workspace, right-click to invoke its pop-up menu, then Click "Add New Project to Workspace...".
- 3) Enter a new project name and its path on the dialog box.
- 4) Click ok button.

The new project that you just created becomes the default active project in the workspace.

Notes: when a new workspace and project is created, two files with the same main file name and .ews and .pjf extend file name is created in the project saved path. The file with .ews extend file name is the workspace file. This file saved the workspace information. The file with .pjf extend file name is the project file, which saved the information about this project.

You can't edit these two files with manual method.

5.3.2 Open a Project

To open an existing project workspace:

- 1) On the File menu, click Open Workspace.
- 2) Select the driver and directory that contain the project workspace that you want to open.
- 3) Select the .ews or .prj file for the project workspace from the File Name list and click OK.

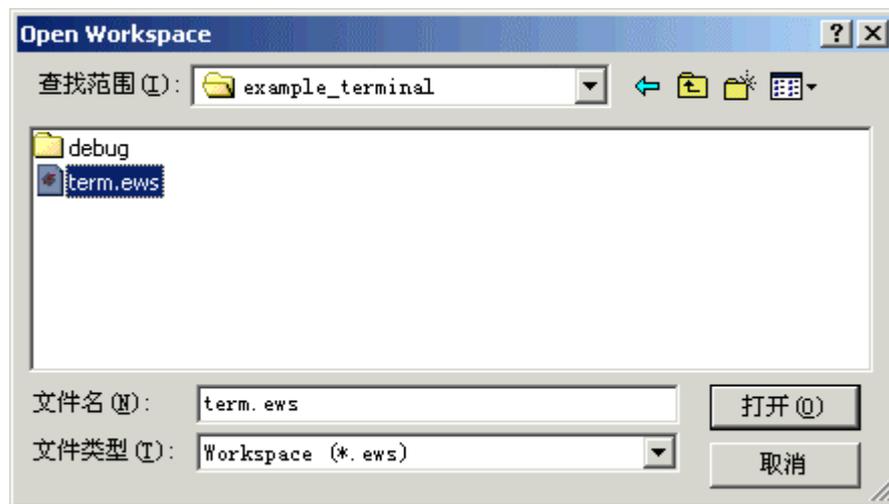


Figure 5-4 Opening a Project

To reopen a recently used project workspace:

On the File menu, click recent workspaces, and then click the name of the recently used workspace.

5.3.3 Workspace Operation

Workspace window displays and manages the files and projects in workspace window. All source files in projects and all projects in workspace can be browsed intuitively in workspace window. The top level catalogue of workspace displays current workspace name and project numbers in this workspace, the second level catalogue displays each project name, the third level catalogue displays source file name or source file group name, the fourth level catalogue displays source file name. Source files can be divided into many groups according to file type. Default groups have 'Project Source Files' and 'Project Header Files'. Group is a logic concept, not according to actual file directory, create group just for explicit manage source files.

Each catalogue in workspace corresponds to different right mouse menu.

The top level catalogue's right mouse menu is workspace right mouse menu, show as follows figure:

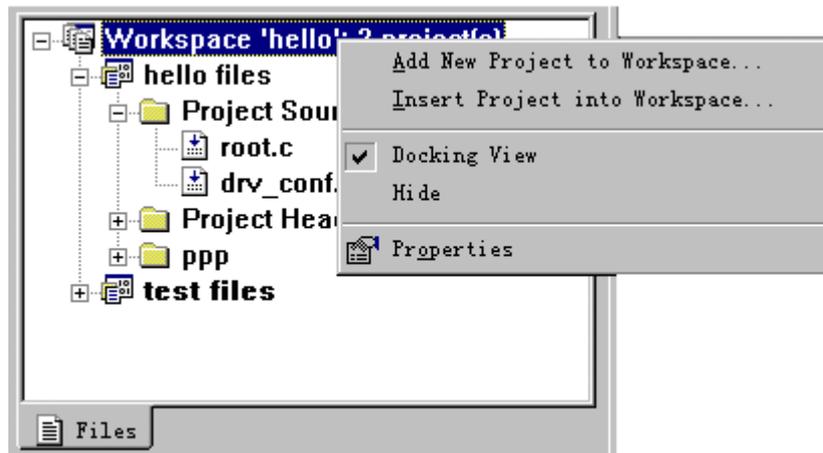


Figure5-5 Workspace Right Mouse Menu

The operation of workspace right mouse menu is:

Table5-3 Operation of The Top Level Catalogue's Right Mouse Menu

| Menu Item | Operation |
|----------------------------------|--|
| Add New Project to workspace... | Create a new project into current workspace |
| Insert Project into Workspace... | Insert a existent project into current workspace |
| Docking View | Switch display mode of workspace window |
| Hide | Hide workspace window |
| Properties | Popup workspace's property dialog box |

The second level catalogue's right mouse menu is project right mouse menu, show as follows figure:

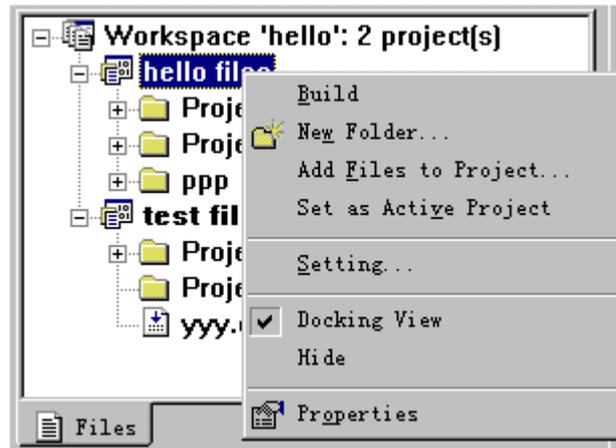


Figure5-6 Project Right Mouse Menu

The operation of project right mouse menu is:

Table5-4 Operation of The Second Level Catalogue's Right Mouse Menu

| Menu Item | Operation |
|-------------------------|---|
| Build | Compile current active project |
| New Folder.. | Create a new group into current project |
| Add Files to Project... | Add a file into current project |
| Set as Active Project | Set current project as active project |
| Setting... | Project setting management, detail see sect 3.3 |
| Docking View | Switch display mode of workspace window |
| Hide | Hide workspace window |
| Properties | Popup workspace's property dialog box |

The third level catalogue's right mouse menu is file group right mouse menu, show as follows figure:

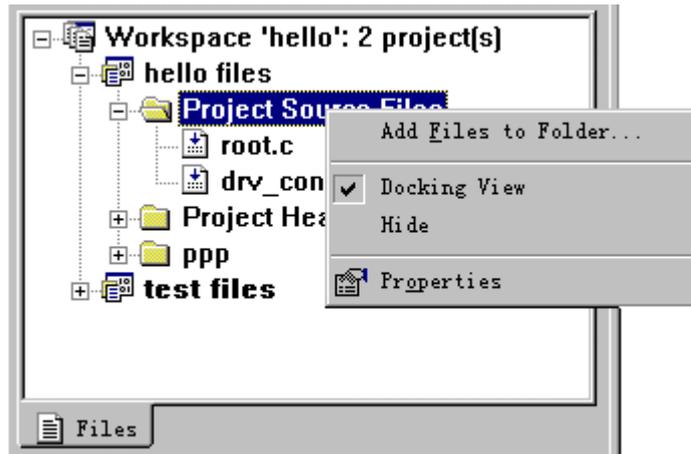


Figure5-7 File Group Right Mouse Menu

The operation of file group right mouse menu is:

Table5-5 Operation of The Third Level Catalogue's Right Mouse Menu

| Menu Item | Operation |
|-----------------------|---|
| Add Files to Folder.. | Add a file into current file group |
| Docking View | Switch display mode of workspace window |
| Hide | Hide workspace window |
| Properties | Popup workspace's property dialog box |

The fourth level catalogue's right mouse menu is file right mouse menu, show as follows figure:

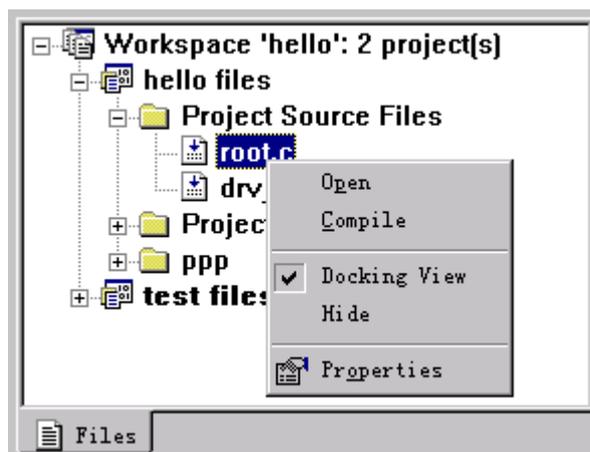


Figure5-8 File Right Mouse Menu

The operation of file right mouse menu is:

Table5-6 Operation Of The Fourth Level Catalogue's Right Mouse Menu

| Menu Item | Operation |
|------------------|---|
| Open | Open current file |
| Compile | Compile current file |
| Docking View | Switch display mode of workspace window |
| Hide | Hide workspace window |
| Properties | Popup workspace's property dialog box |

User can convenient manage projects in workspace and files in projects through all level catalogue's right mouse menu. If need delete something, just select that object (such as workspace, file group, file etc), push 'Del' key.

5.3.4 Save and Close Workspace

If user want to save the change of workspace, just click menu 'File > Save Workspace'. Save a workspace or project is not save concrete files in project or workspace, it saves correlative file management information.

Click menu 'File > Close Workspace' can close current workspace, EmbestIDE will auto check whether or not exist change after last workspace saves when closing workspace, if exist change, EmbestIDE will popup a hint dialog box to hint user whether save change or not.

Notes: When close workspace, just hide workspace window, not actual close workspace window.

5.3.5 Set Active Project

One workspace can contain several projects, but only one project can be active project, only active project can be build and debug. The active project's icon will be in colorful. Show as in figure 5-9.

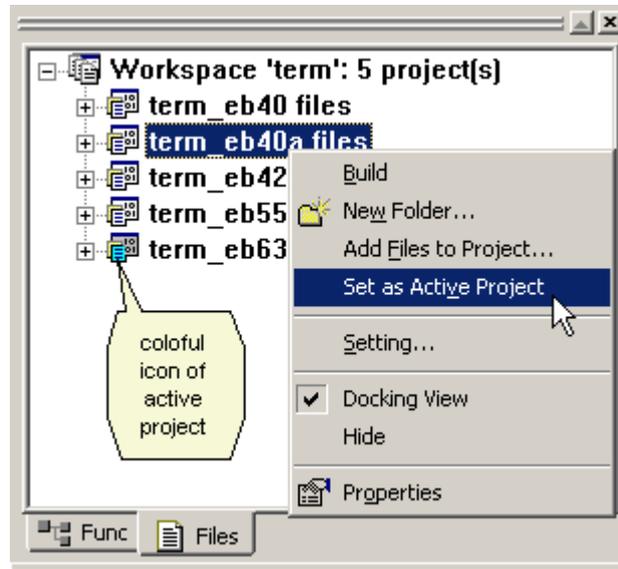


Figure 5-9 right key set as current active project and it's colorful icon

If want to set a project as active project, select that project, right click mouse and click 'Save as Active Project' in popup menu, show as in figure 5-9; or click menu 'Project > Set Active Project', then select the project want to set active, show as in figure 5-10.

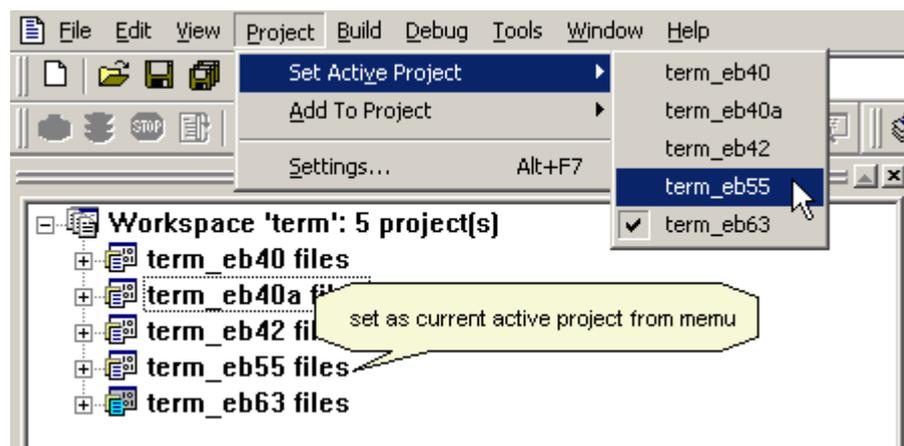


Figure 5-10 set as current active project from menu bar

5.4 Project Basic Configuration

5.4.1 Processor Configuration

Select menu 'Project > Settings...' will popup project setting dialog box. In project setting dialog box, select 'Processor' property page, and user can configure CPU on target board. The dialog box show as following figure:

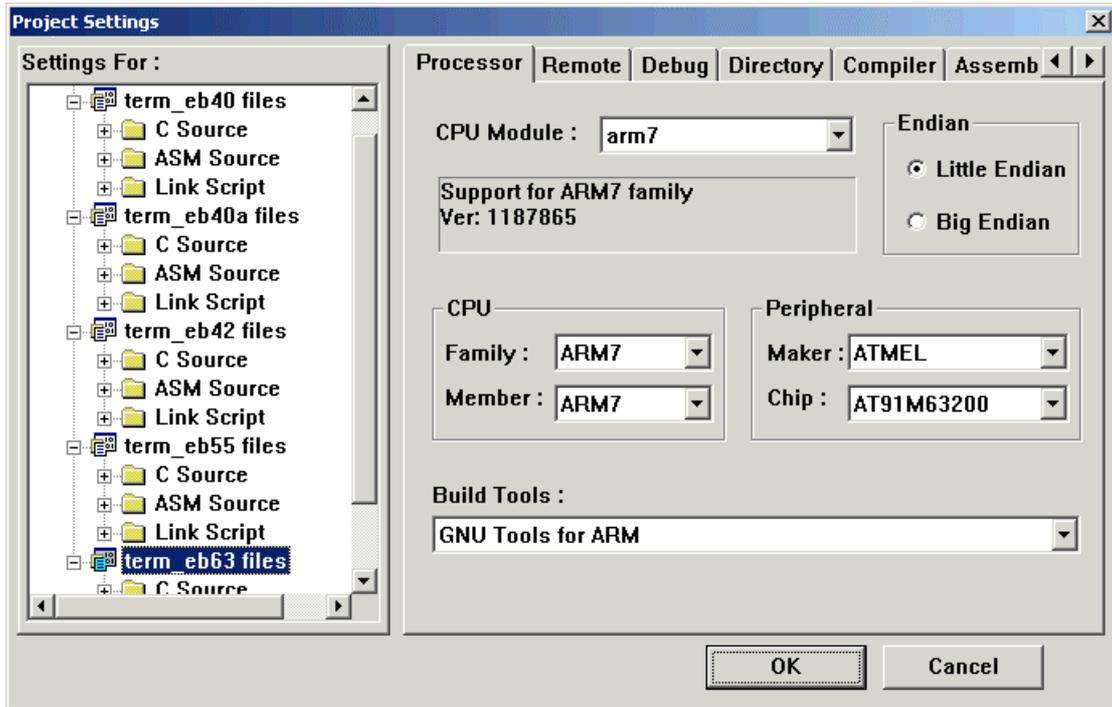


Figure5-11 Processor Configuration Dialog Box

CPU Module: select current CPU module, different CPU module will sustain different CPU series. Embest IDE has supported the core of ARM7 and ARM9 series processor.

CPU Family: select CPU series which user's CPU belong to.

CPU Member: select detail CPU type.

Endian: set memory area byte order of user's CPU is big endian or little endian.

Peripheral Maker: select the Maker of processor on the target board.

Peripheral Chip: select the processor name on the target board.

Build Tools: set compiler and linker which according to user's CPU.

5.4.2 Configure Emulator

Select menu 'Project>Settings...' will popup project setting dialog box. In project setting dialog box, by select 'Remote' property page, user can figure emulator connect setting. The dialog box show as following figure:

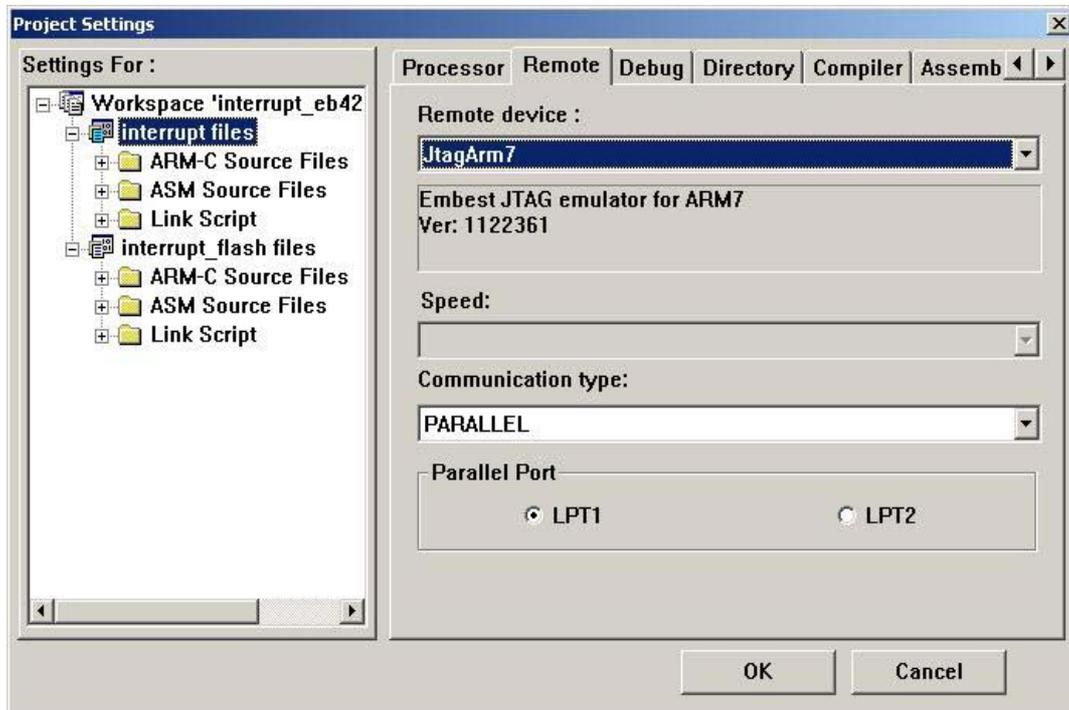


Figure5-12 Emulator Configuration Dialog Box

Remote device: select debug device, Embest Emulator for ARM is according to 'jtagarm7', and Embest PowerICE for ARM is according to 'PowerICEARM7'. Series of ARM9 CPU, Embest Emulator for ARM is according to 'jtagarm9', and Embest PowerICE for ARM is according to 'PowerICEARM9'. Embest Simulator for ARM7 is according to 'simarm7'. The information of current device will display on the pane.

Speed: setting the Emulator work speed. Embest Emulator for ARM was worked at constant speed 25Kbyte per second, and Embest PowerICE for ARM was support to change the working speed valid: Full Speed(120Kbyte/s)、High Speed、Medium Speed、Low Speed. The valid work speed of Embest PowerICE for ARM show as following figure 5-13:

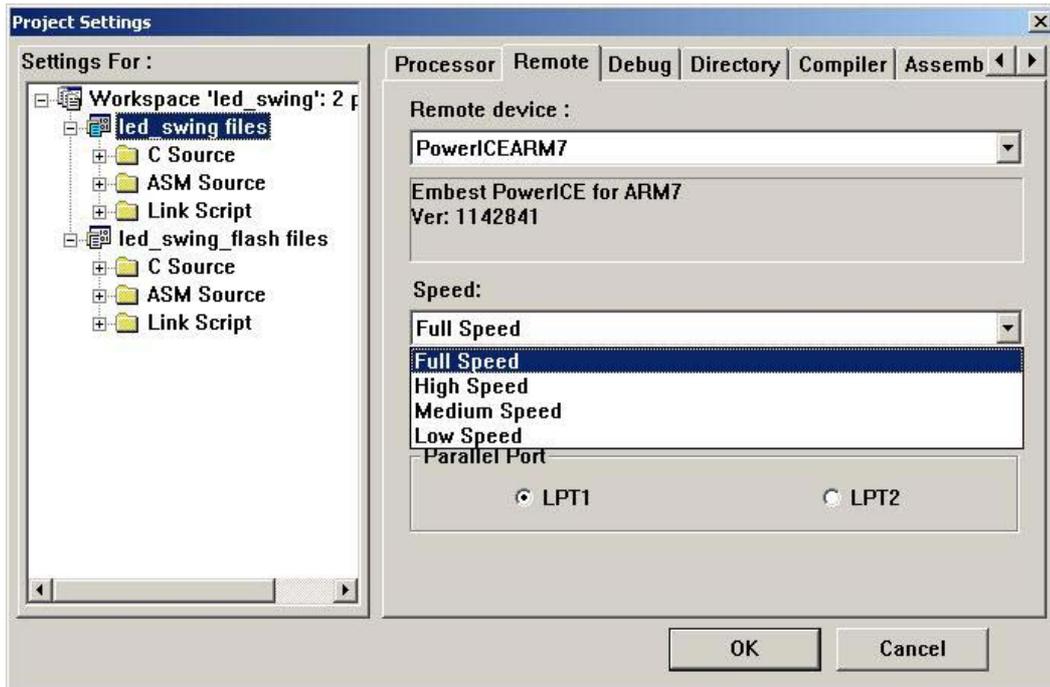


Figure 5-13 Embest PowerICE for ARM work speed

Communication type: select connect communication type which emulator connect to host machine, such as Embest Emulator for ARM use parallel port connect, so select 'PARALLEL'.

Parallel Port: set according to actual connect.

Note: Usually, Embest Emulator for ARM use parallel port mode, would be set EPP(0x278/0x378).

5.5 ARM SDT and ADS project operation

5.5.1 SDT software project opening

Click menu item File > Open Workspace, and a dialog box will pop up for opening workspace. Select file type ARM SDT Project, with the suffix for the software project file of SDT as apj, as shown in the following figure.

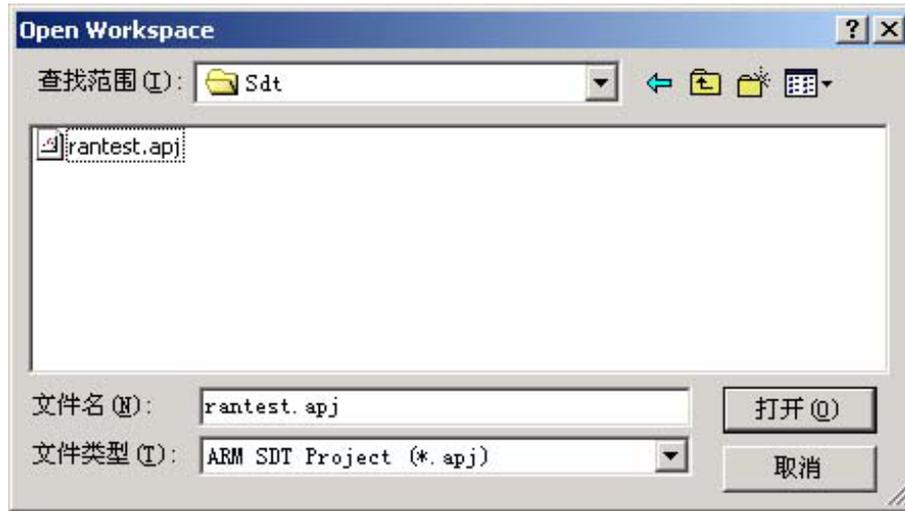


Fig. 5-13 Open SDT project

Select the SDT project file to be opened, press the open button to open the project. At the same time opening the SDT project, the integrated environment will automatically generate the Embest IDE workspace and project file corresponding to the project, in which the name and number of the project files correspond to the variant in the SDT project, as shown in the following figure.

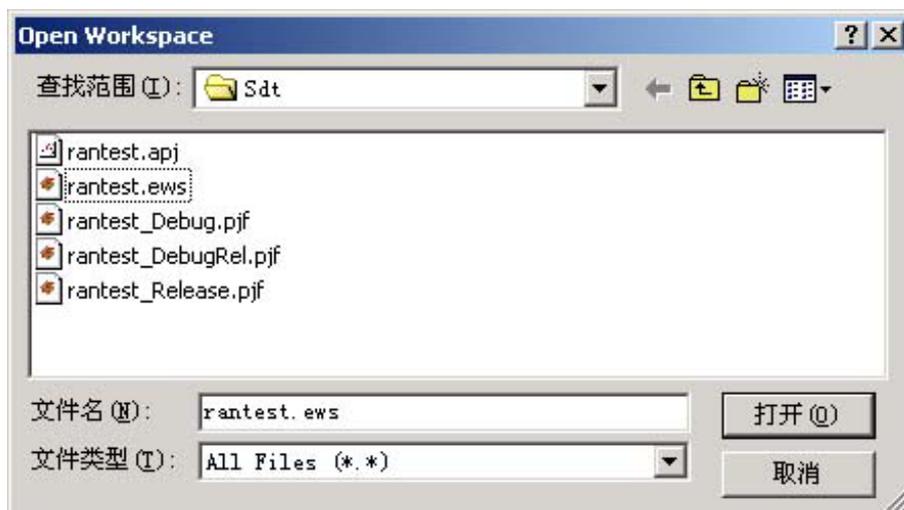


Fig. 5-14 Automatic generation of Embest IDE workspace and project file

When the user opens the SDT project for the second time, the integrated environment will present a dialog box as shown in the following figure:



Fig. 5-15 Prompt dialog box for the second time of opening the SDT project

The user can select “Yes” to directly open the Embest IDE workspace that was automatically generated during the first time of opening the SDT project. Or, he can select “No” to regenerate and open the Embest IDE workspace and project.

◦ Note: In opening the SDT project of Embest IDE, the user is in fact finally opening the automatically generated Embest IDE workspace after conversion. The follow-up configurations of the user will be directly saved in the Embest IDE workspace and project. Therefore, if the user has performed new configuration and modification after opening the SDT project, Embest IDE workspace will be directly opened during the follow-up development process.

The project interface opened by the ARM SDT software is illustrated in the following figure:

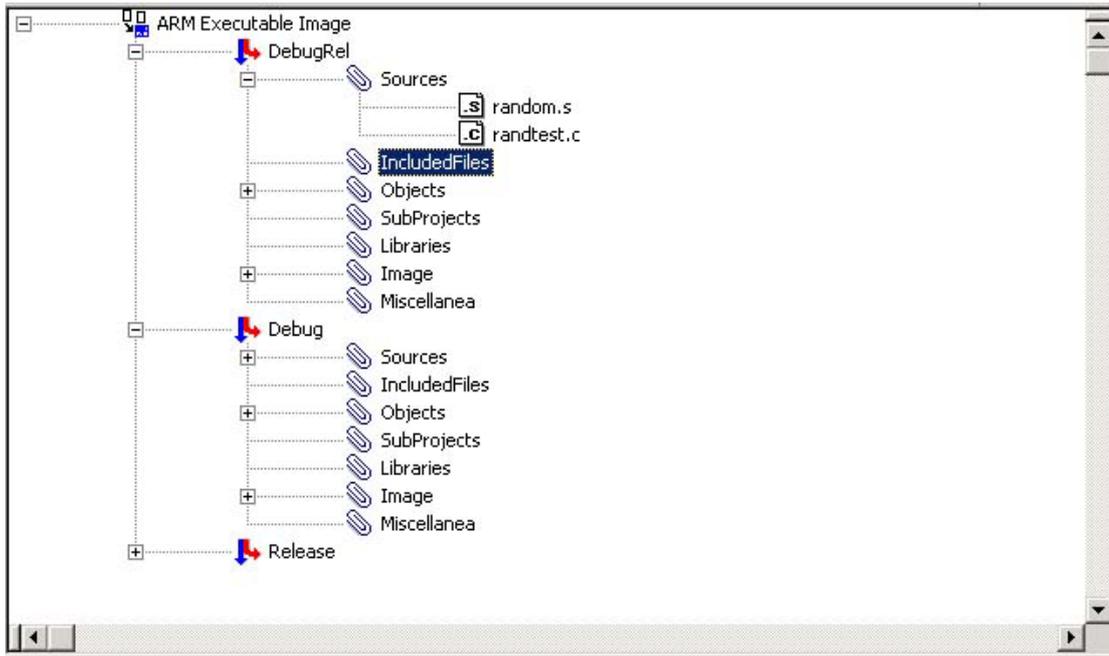


Fig. 5-16 Project interface opened by the ARM SDT

After conversion, the workspace interface of the project opened by the Embest IDE is illustrated in the following figure:

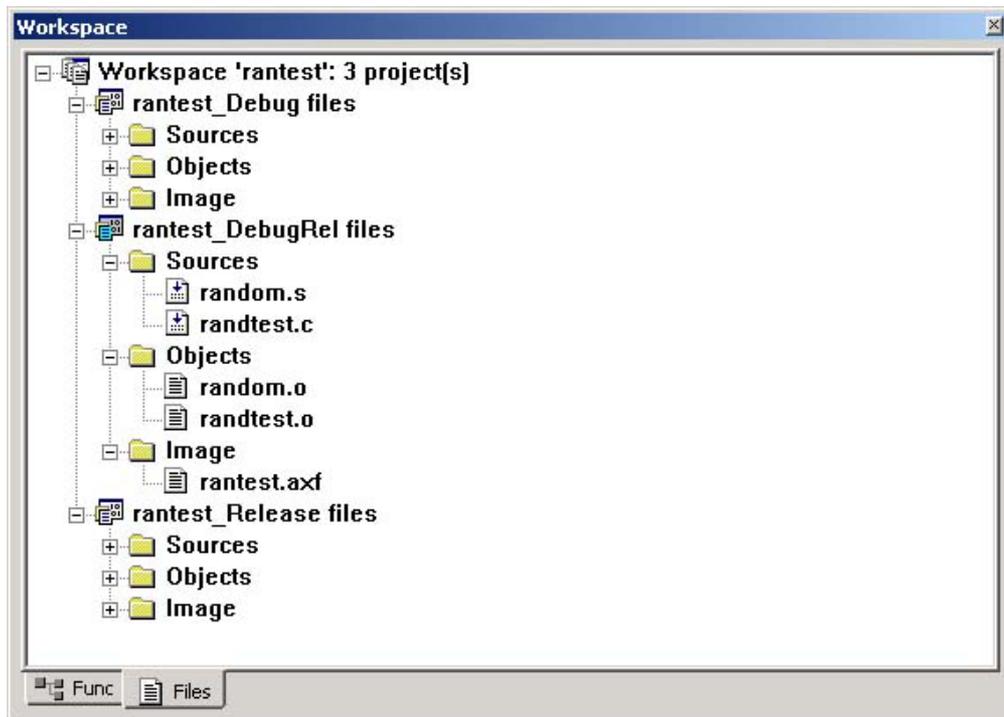


Fig. 5-17 Project interface opened by the ARM SDT

During opening, the Embest IDE automatically deletes file folders that do not contain any files in the SDT project.

5.5.2 ADS software project opening

Click menu item File > Open Workspace, and a dialog box will pop up for opening workspace. Select file opening type ARM ADS Project, with the suffix for the software project file of ADS as mcp, as shown in the following figure.

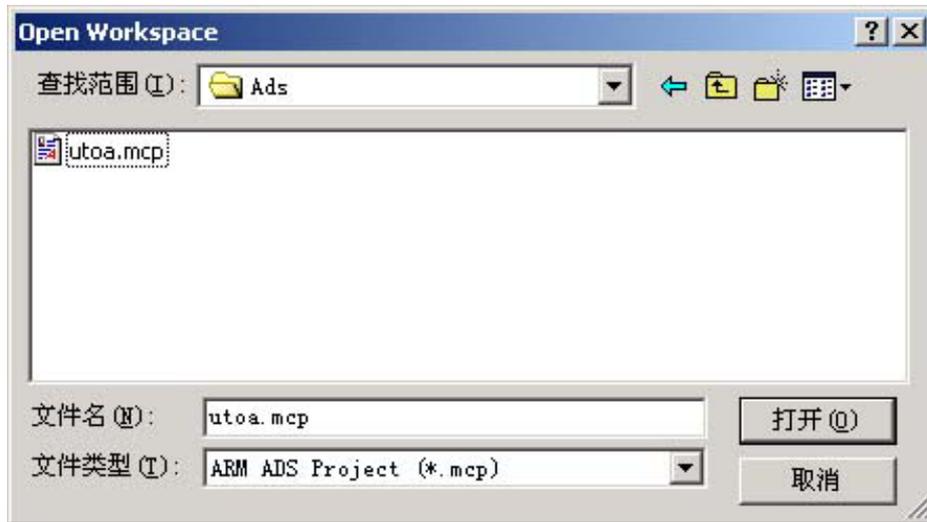


Fig. 5-18 Open ADS project

Select the ADS project file to be opened, press the open button to open the project. At the same time opening the ADS project, the integrated environment will automatically generate the Embest IDE workspace and project file corresponding to the project, in which the name and number of the project files correspond to the Target in the SDT project, as shown in the following figure.

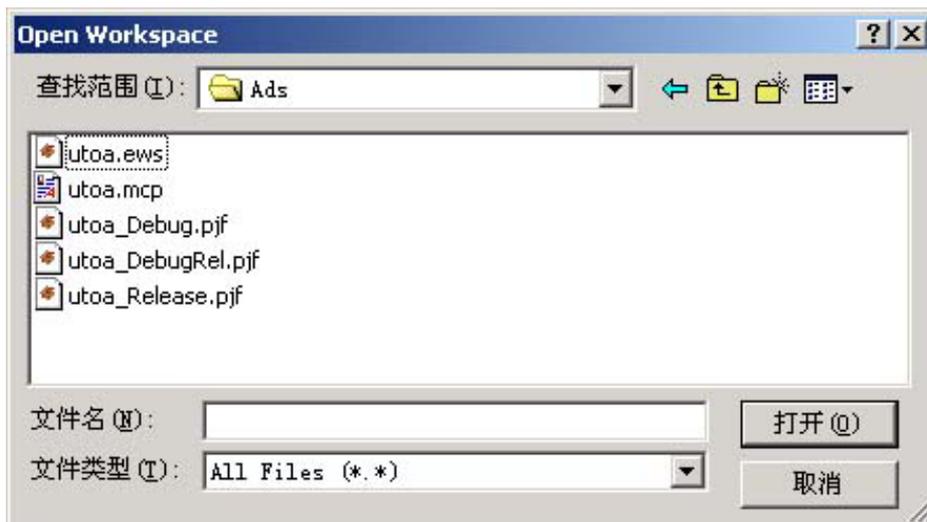


Fig. 5-19 Automatic generation of Embest IDE workspace and project file

When the user opens the ADS project for the second time, the integrated environment will present a dialog box as shown in the following figure:

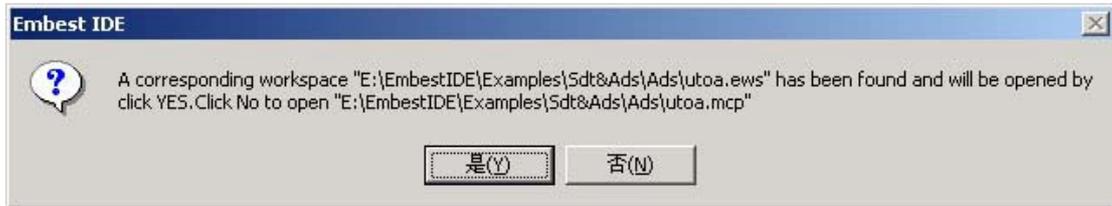


Fig. 5-20 Prompt dialog box for the second time of opening the ADS project

The user can select "Yes" to directly open the Embest IDE workspace that was automatically generated during the first time of opening the ADS project. Or, he can select "No" to regenerate and open the Embest IDE workspace and project.

Note: In opening the ADS project of Embest IDE, the user is in fact finally opening the automatically generated Embest IDE workspace after conversion. The follow-up configurations of the user will be directly saved in the Embest IDE workspace and project. Therefore, if the user has performed new configuration and modification after opening the ADS project, Embest IDE workspace will be directly opened during the follow-up development process.

The project interface opened under the ARM ADS software is illustrated in the following figure:

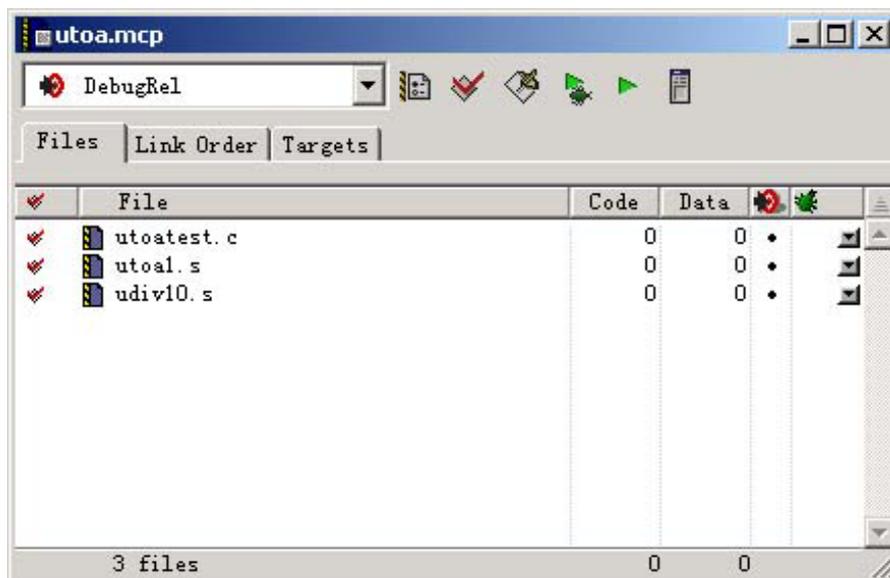


Fig. 5-21 Project interface opened by ARM ADS

The workspace interface opened by the Embest IDE after conversion is illustrated in the following figure:

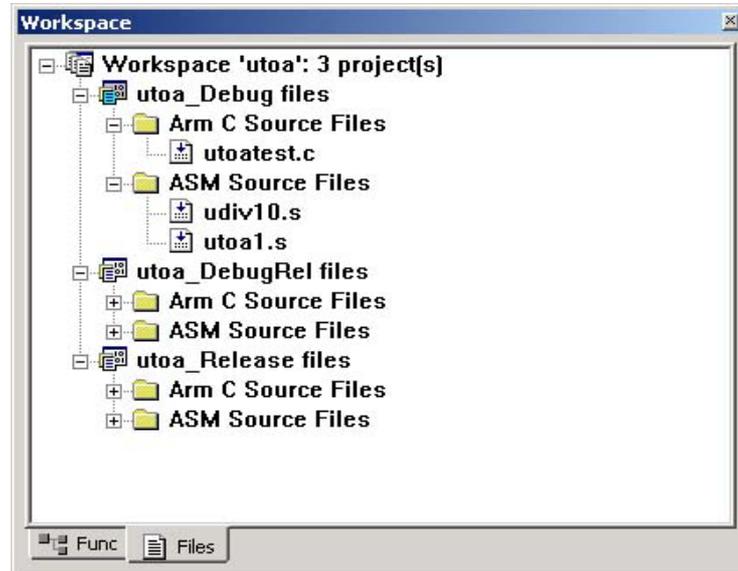


Fig. 5-22 ADS project interface opened by Embest IDE

5.5.3 Configuration after opening SDT or ADS project

Default setting is applied to the processor and debugging equipment after the ARM SDT or ADS project is opened by the Embest IDE, and users have to perform adjustment according to the specific conditions.

Adjusting processor configuration

Select menu item Project > Settings... or use shortcut key Alt+F7, and a dialog box for project setting will pop up. Select processor setting dialog box in the project setting dialog box as shown in the following figure, and perform adjustment on the options in the blue elliptical box according to specific conditions.

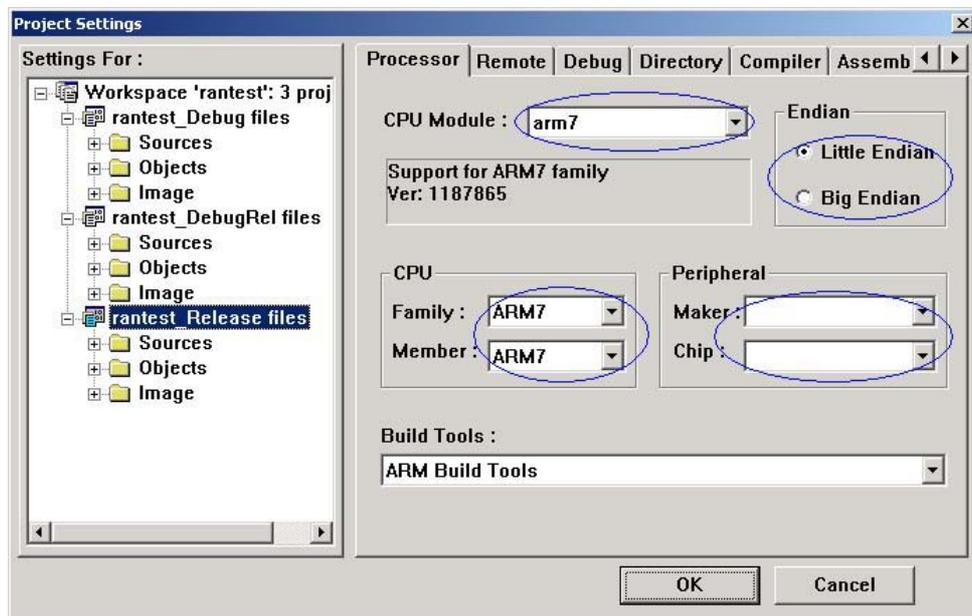


Fig. 5-23 Adjusting processor configuration in project setting dialog box

Adjusting emulator configuration

Select menu item Project > Settings..., and a dialog box for project setting will pop up. Select Remote setting dialog box in the project setting dialog box as shown in the following figure, and perform adjustment on the options in the blue elliptical box according to specific conditions.

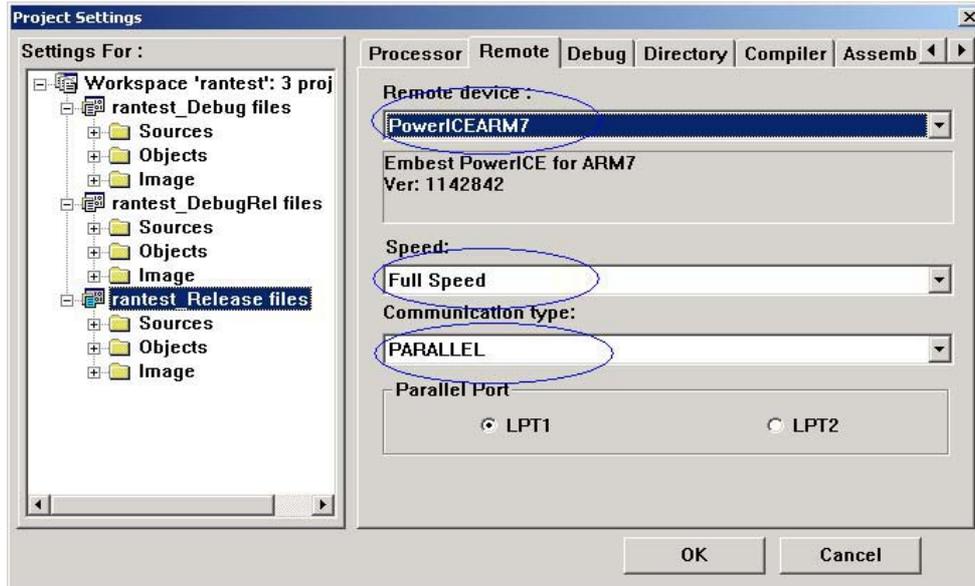


Fig. 5-24 Adjusting emulator configuration in project setting dialog box

Library file path setting

After the Embest IDE opens the SDT or ADS project, the default compiler will be ARM compiler. If the user links library files in the software project, he has to manually set the library file path.

Select menu item Project>Settings..., and a dialog box for project setting will pop up. In the dialog box for project setting, select Linker setting dialog box, select the Information Options, and search the path for setting library files according to specific conditions, as shown in the following figure.

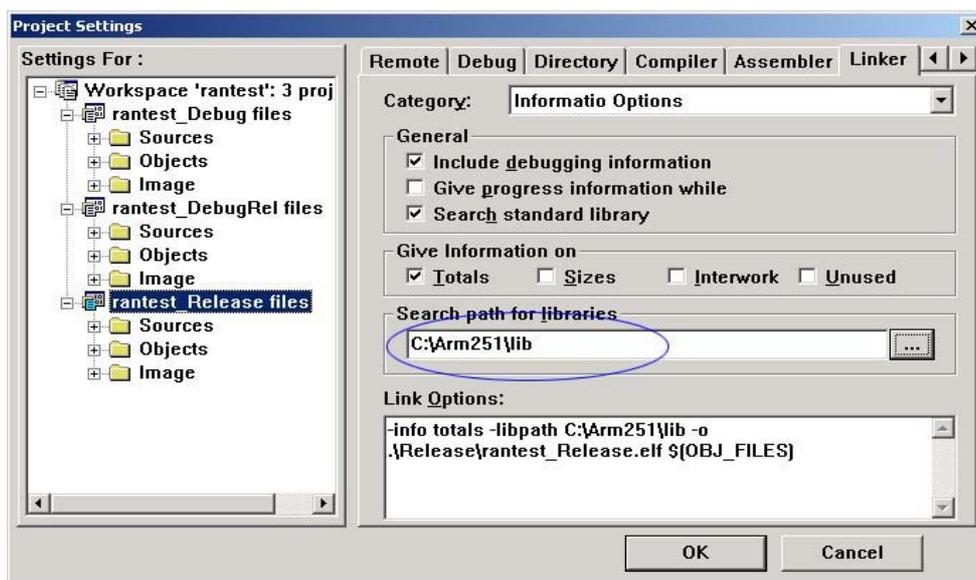


Fig. 5-25 Setting searching path for object file

6. Project Configuring & Building

6.1 Build Tools Introduction(by Embest IDE)

The build module is a visual interface to configure a project and build the target program. The following Project Settings dialog box handles settings that affect how the EmbestIDE builds a specific target file within a project.

Because several cross-compilers can integrate with the plug-in technique, you must select a compiler for each project before build the target. Show as Figure 6-1, in the Processor page of Project Settings dialog box, the Build Tools should be selected before the further configure .

Build Tools that Embest IDE project sets up selects and includes **GNU Tools for ARM** or **ARM Build Tools**, namely can compatible **GNU** compiling Tools and **ARM** compiling Tools.

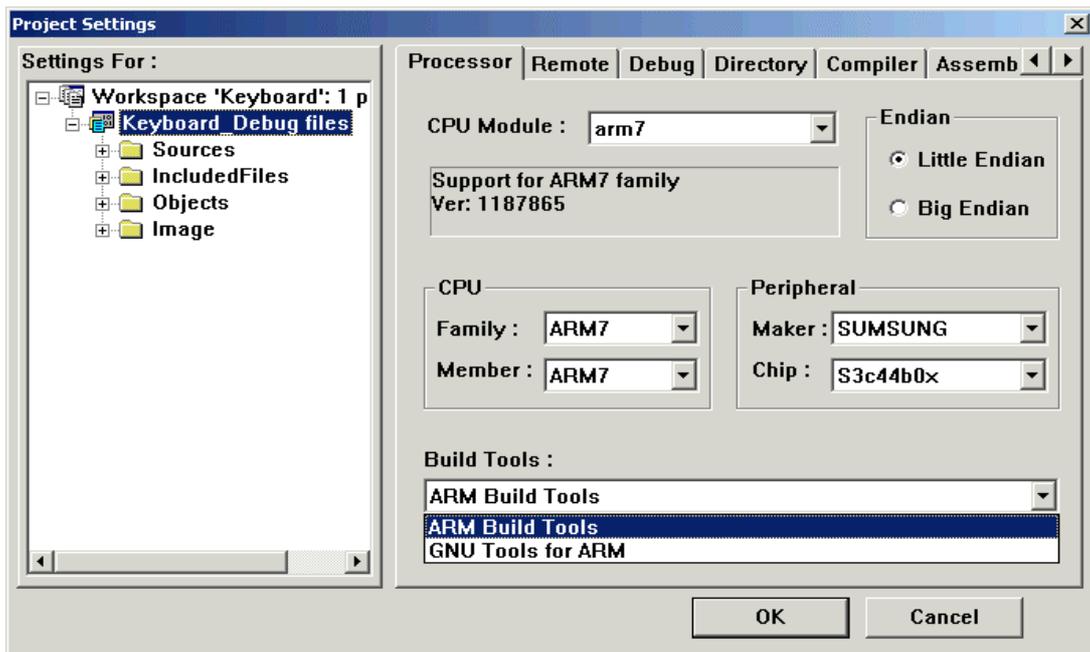


Figure 6-1-1 Project Settings Dialog

Figure 6-1-1 shows an example of project settings. The selected Build Tools for project led_swing is GNU Tools for ARM.

Note: If the Build Tools is not configured for a project, its property pages of compiler, assembler and linker will not be displayed. And the prompt information—"No build configuration for project: xxxx"—will be displayed in the build pane, as you build it at this moment.

6.2 GNU Tools for ARM

It is one that **GNU Tools for ARM** compiles and selects the compiling device is set up GCC cross-compiler of ARM processor specially; GNU Tools for ARM is the **free software**; GNU Tools for ARM is a dedicative cross-compiler for ARM processor, created from GNU source code. It mainly **includes C/C++ compiler, assembler, linker, standard libraries** for embedded system and other tools.

6.2.1 Files Type and Compiler

Which application of the cross-compiler will be invoked by EmbestIDE to handle a source file lies on its postfix. Figure 6-2-1 shows each postfix and its relevant application.

Figure 6-2-1

| Postfix | Application of Cross-Compiler |
|---------|-------------------------------|
| *.c | C Compiler |
| *.C | C++ Compiler |
| *.cpp | C++ Compiler |
| *.cc | C++ Compiler |
| *.cp | C++ Compiler |
| *.c++ | C++ Compiler |
| *.cxx | C++ Compiler |
| *.s | Assembler |
| *.asm | Assembler |

Files type of GNU tools chain shows as Figure 6-2-2.

Figure 6-2-2

| Files | Files Type |
|----------------|------------------------------|
| file.stuf | source files of GNU |
| file.h/.inc/.a | head file |
| file.o | ELF formation target file |
| file.elf | ELF formation debugging file |

stuf stand for *.c/C, *.cpp/C++, *.s/asm, are the source files which can compiling or assembling by GNU tools chain(Note: it is permitted that there is the **blank** within the filename or directory.)

6.2.2 Options for Compiler

The Compiler property page, as in the Figure 6-3, is used to configure the compile options for C/C++ compiler of GNU Tools for ARM cross-compiler, All the options user selected are displayed in the Compile Options edit box with the following format:

```
[Opt-1] [Opt-2] ... $(SOURCE_FILE) ... [Opt-n] ... -o[Path]$(OBJ_FILE)
```

Note: You can input or modify the options manually in the edit box, but the blank character between each option must be reserved, and the macros \$(SOURCE_FILE), \$(OBJ_FILE) should not be deleted or modified. \$(SOURCE_FILE) means the source file to be compiled, \$(OBJ_FILE) means the output of the compiling. There will be replaced with the actual file name by the EmbestIDE at the time of building.

Note: When you configure a project setting, you should consider that the location of the project file (.pjf) is the current directory.*

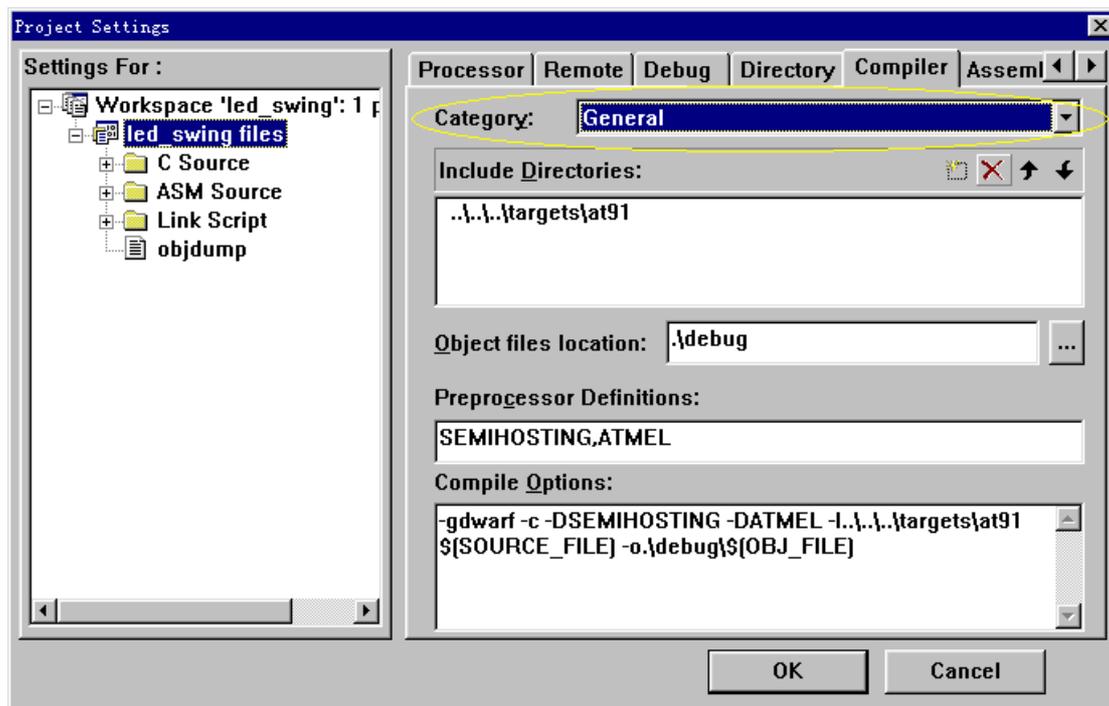


Figure 6-2-1 Compiler General Settings

Show in Figure 6-3, compiler options are divided into five Categories:

- **General**

- **Warning Options**
- **Debug/Optimization**
- **Target Specific Options**
- **Code Generation Options**

Note: All this compiler options are also compiled in the document <<Program Reference>>, and explained more detailed.

| General Cluster | Description |
|--------------------------|---|
| Include Directories | Add <dir> to the end of the main include paths |
| Object file location | The location of the compile outcome will be placed, and no blank character can be included. If not exist, EmbestIDE create it automatically. |
| Preprocessor Definitions | Define macros, each macro separated by comma and with blank character. |

Warning Options

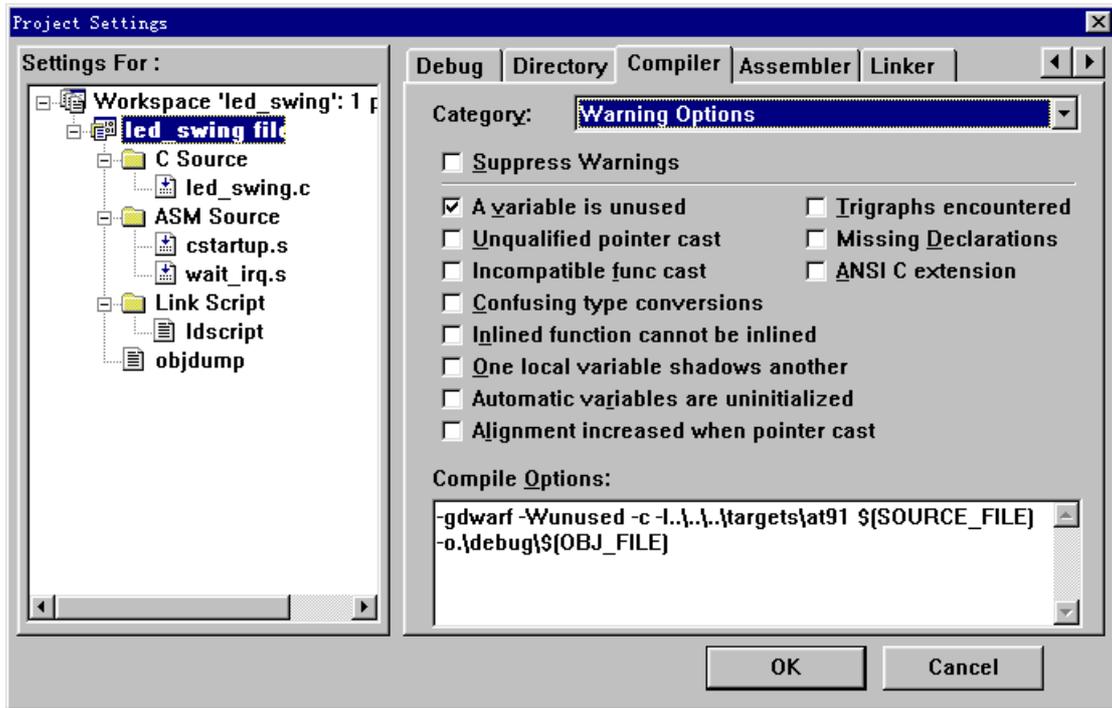


Figure 6-2-2 Compiler Warning Options

| Warning Options | Cluster | Description |
|---------------------------------------|---------|---|
| Suppress Warnings | | -w, Inhibit warning messages |
| A variable is unused | | -Wunused, Warn when a variable is unused |
| Unqualified pointer cast | | -Wcast-qual, Warn about cast which discard qualifiers |
| Incompatible func cast | | -Wbad-function-cast, Warn about casting functions to incompatible types |
| Confusing type conversions | | -Wconversion, Warn about possibly confusing type conversions |
| Inlined function cannot be inlined | | -Winline, Warn when an inlined function cannot be inlined |
| One local variable shadows another | | -Wshadow, Warn when one local variable shadows another |
| Automatic variables are uninitialized | | -Wuninitialized, Warn about uninitialized |

| | |
|---------------------------------------|---|
| | automatic variables |
| Alignment increased when pointer cast | -Wcast-align, Warn about pointer casts which increase alignment |
| Trigraphs encountered | -Wtrigraphs, Warn when trigraphs are encountered |
| Missing Declarations | -Wmissing-declarations, Warn about global functions without previous declarations |
| ANSI C extension | -pedantic, Issue warnings needed by strict compliance to ANSI C |

Debug/Optimization

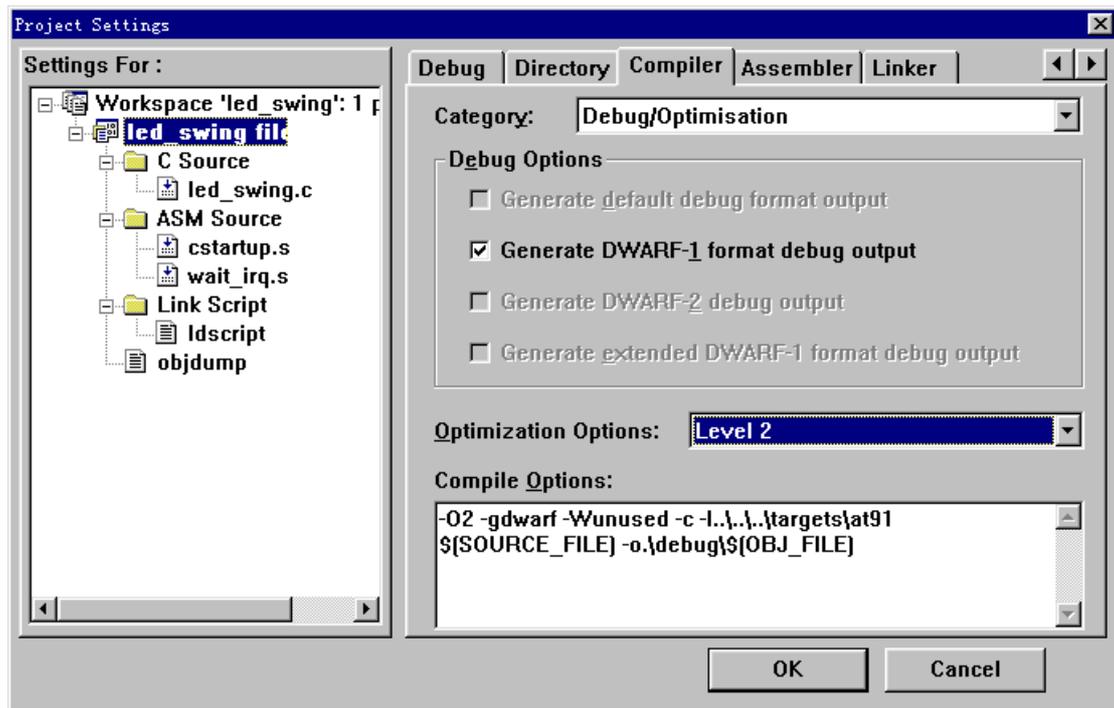


Figure 6-2-3 Compiler Debug/Optimization

| Debug/Optimization Cluster | Description |
|---|---|
| Generate default debug format output | -g |
| Generate DWARF-1 format debug output | -gdwarf |
| Generate DWARF-2 debug output | -gdwarf-2 |
| Generate extended DWARF-1 format debug output | -gdwarf+ |
| Optimization Options | -O[number]/-Os, Set Optimization level to [number]/Optimize for space rather than speed |

Target Specific Options

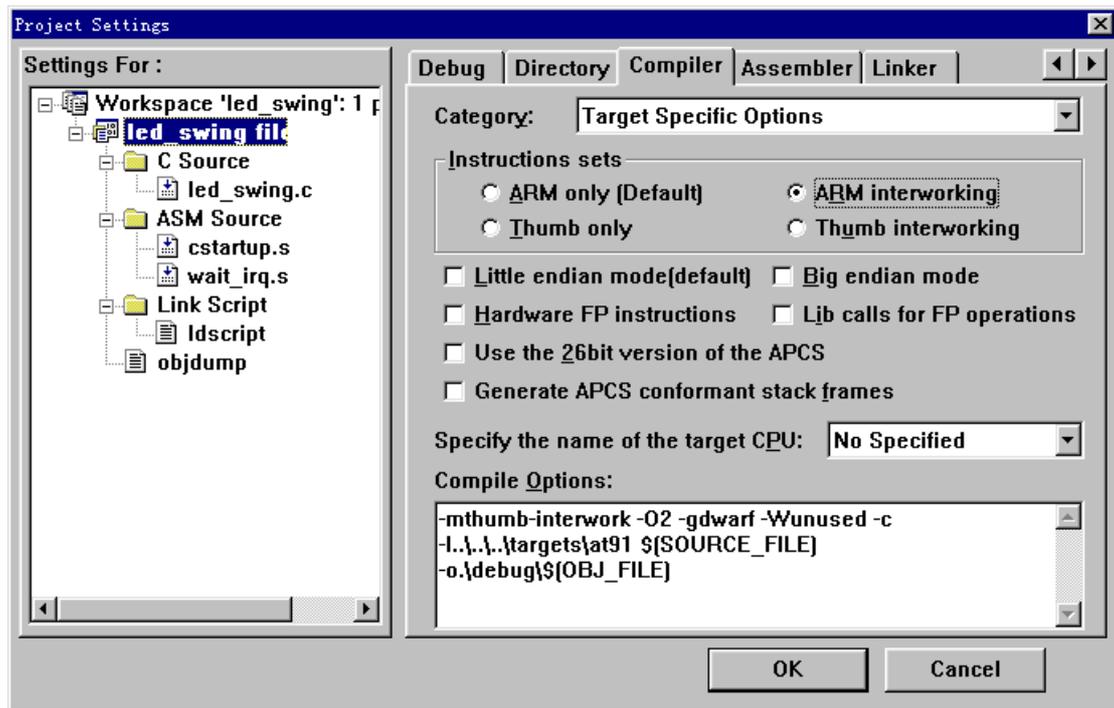


Figure 6-2-4 Compiler Target Specific Options

| Target Specific Options | Cluster | Description |
|------------------------------|---------|--|
| ARM only (Default) | | Generate ARM instructions |
| ARM interworking | | -mthumb-interwork, Generate ARM instructions supporting calls between THUMB and ARM instructions sets |
| Thumb only | | -mthumb, Generate THUMB Instructions |
| Thumb interwork | | -mthumb -mthumb-interwork, Generate THUMB instructions supporting calls between THUMB and ARM instructions |
| Little endian mode (Default) | | -mlittle-endian, Assume target CPU is configured as little endian |
| Big endian mode | | -mbig-endian, Assume target CPU |

| | |
|---------------------------------------|---|
| Hardware FP instructions | is configured as big endian -mhard-float, Use hardware floating point instructions |
| Lib call for FP operations | -msoft-float, Use library calls to perform FP operations |
| Use the 26bit version of APCS | -mapcs-26, Use the 26bit version of APCS |
| Generate APCS conformant stack frames | -mapcs-frame, Generate APCS conformant stack frames |
| Specify the name of the target CPU | Specify the name of the target CPU |

Code Generation Options

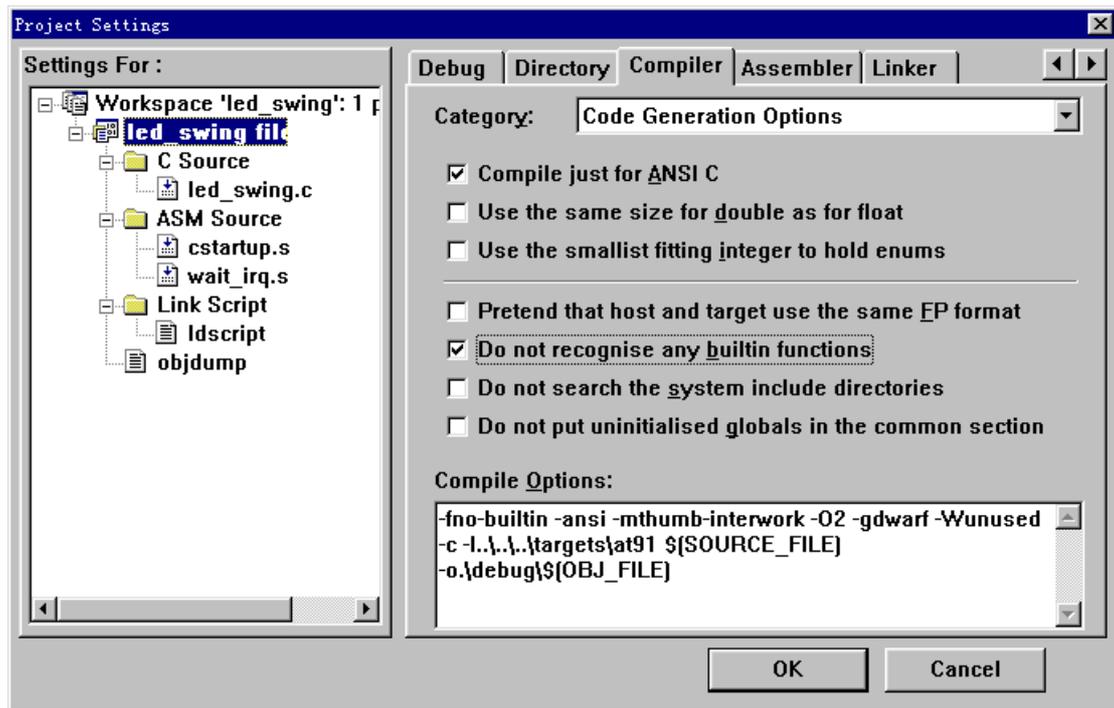


Figure 6-2-5 Compiler Code Generation Options

| Code Generation Options | Cluster | Description |
|--|---------|--|
| Compile just for ANSI C | | -ansi, Compile just for ANSI C |
| Use the same size for double as for float | | -fshort-double, Use the same size for double as for float |
| Use the smallest fitting integer to hold enums | | -fshort-enums, Use the smallest fitting integer to hold enums |
| Pretend that host and target use the same FP format | | -fpretend-float, Pretend that host and target use the same FP format |
| Do not recognize any built-in functions | | -fno-builtin, Do not recognize any built in functions |
| Do not search the system include directories | | -nostdinc, Do not search the system include directories |
| Do not put uninitialized globals in the common section | | -fcommon, Do not put uninitialised globals in the |

common section

6.2.3 Options for Assembler

The Assembler property page, as in the Figure 6-8, is used to configure the assembling options for assembler of GNU Tools for ARM cross-compiler, All the options user select are displayed in the Assemble Options edit box with the following format:

[Opt-1] [Opt-2] ... \$(SOURCE_FILE) ... [Opt-n] ... -o[Path]\$(OBJ_FILE)

Note: You can input or modify the options manually in the edit box, but the blank character between each option must be reserved, and the macros \$(SOURCE_FILE), \$(OBJ_FILE) should not be deleted or modified. \$(SOURCE_FILE) means the source file to be compiled, \$(OBJ_FILE) means the output of the assembling. There will be replaced with the actual file name by the Embest IDE at the time of building.

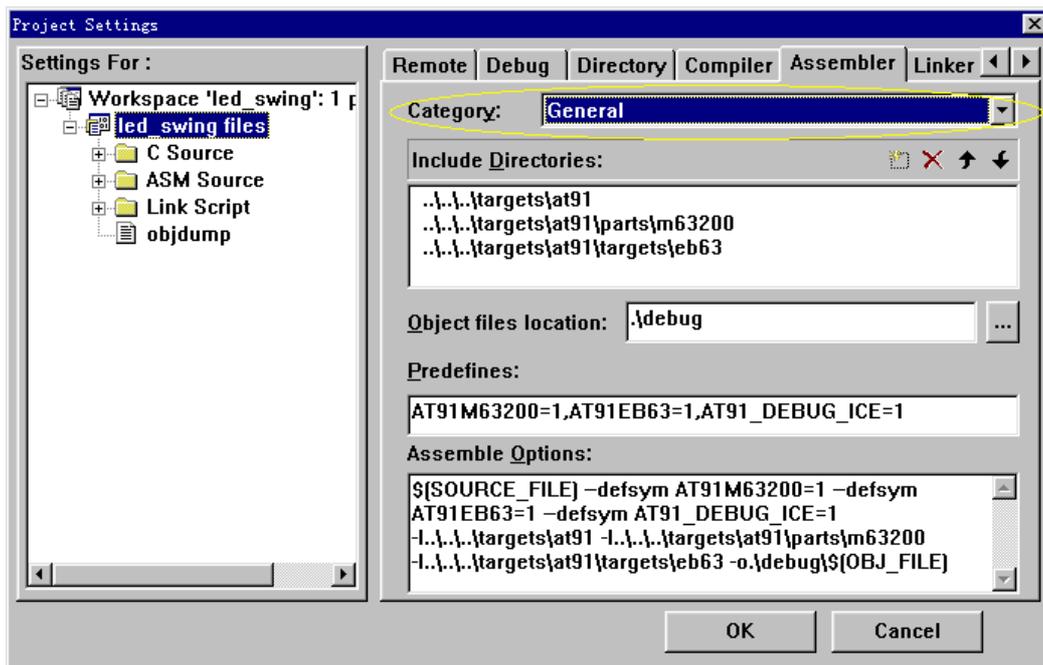


Figure 6-2-6 Assembler General Settings

Show in Figure 6-2-6, assembler options are divided into four Categories:

- **General**
- **Code Generation**
- **Target Specific Options**

● Warning Options

Note: All this assembler options are also compiled in the document <<Program Reference>>, and explained more detailed.

| General Cluster | Description |
|--------------------------|---|
| Include Directories | -I<dir>, Add dir to search list for include directories |
| Object file location | The location of the assembling outcome will be placed, and no blank character can be included. If not exist, Embest IDE create it automatically. |
| Preprocessor Definitions | Define macros, each macro separated by comma and with blank character. |

Code Generation Options

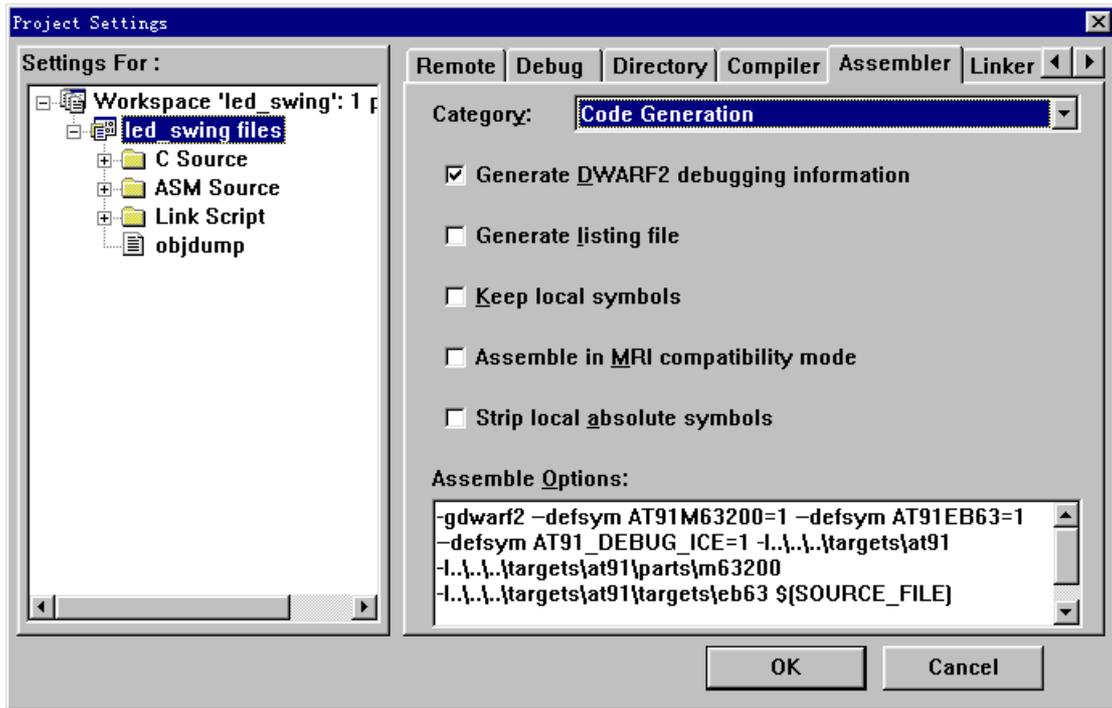


Figure 6-2-7 Assembler Code Generation Options

| Code Generation | Cluster | Description |
|--|------------------------|-------------|
| Generate DWARF-2 debugging information | -gdwarf2 | |
| Generate listing file | -a | |
| Keep local symbols | -L | |
| Assemble in MRI compatibility | -M | |
| Strip local absolute symbols | --strip-local-absolute | |

Target Specific Options

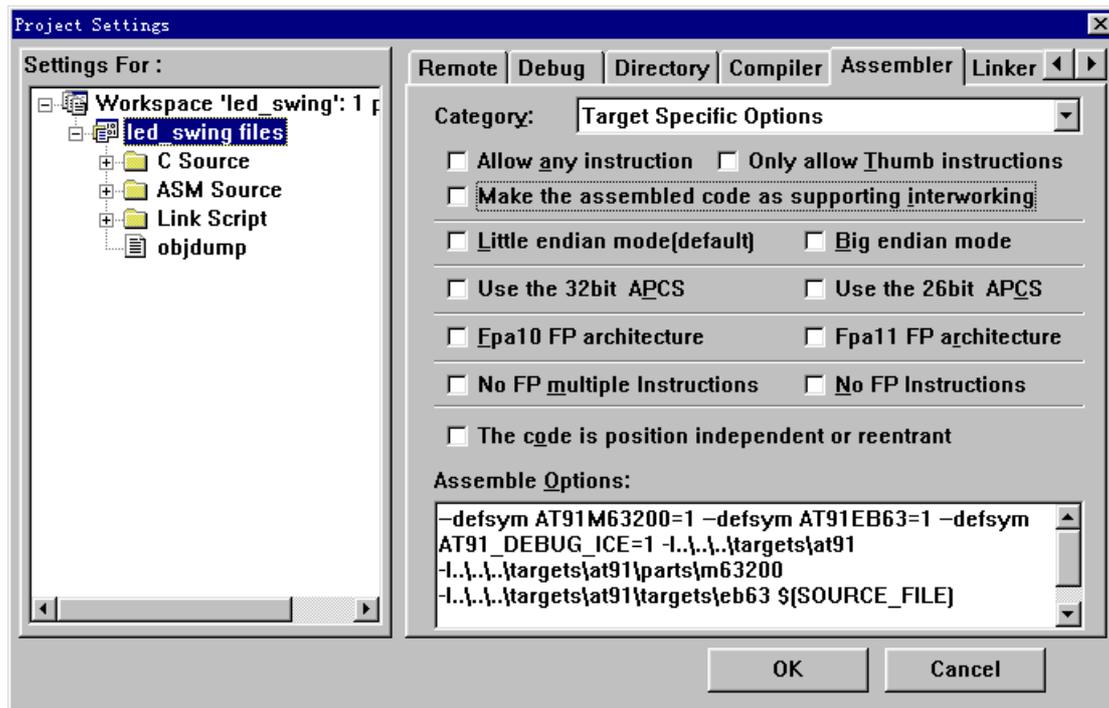


Figure 6-2-8 Assembler Target Specific Options

| Target Specific Options | Cluster | Description |
|---|---------|--|
| Allow any instruction (Default) | | -mall |
| Only allow thumb instructions | | -mthumb |
| Make the assembled code as support interworking | | -mthumb-interwork |
| Little endian mode (Default) | | -EL, Assemble code for a little endian cpu |
| Big endian mode | | -EB, Assemble code for a big endian CPU |
| Use the 32bit APCS | | -mapcs-32 |
| Use the 26bit APCS | | -mapcs-26 |
| Fpa10 FP architecture | | -mfpa10 |
| Fpa11 FP architecture | | -mfpa11 |
| No FP multiple instructions | | -mfpe-old |

No FP instructions

-mno-fpu

The code is position independent or reentrant

-mapcs-reentrant

Warning Options

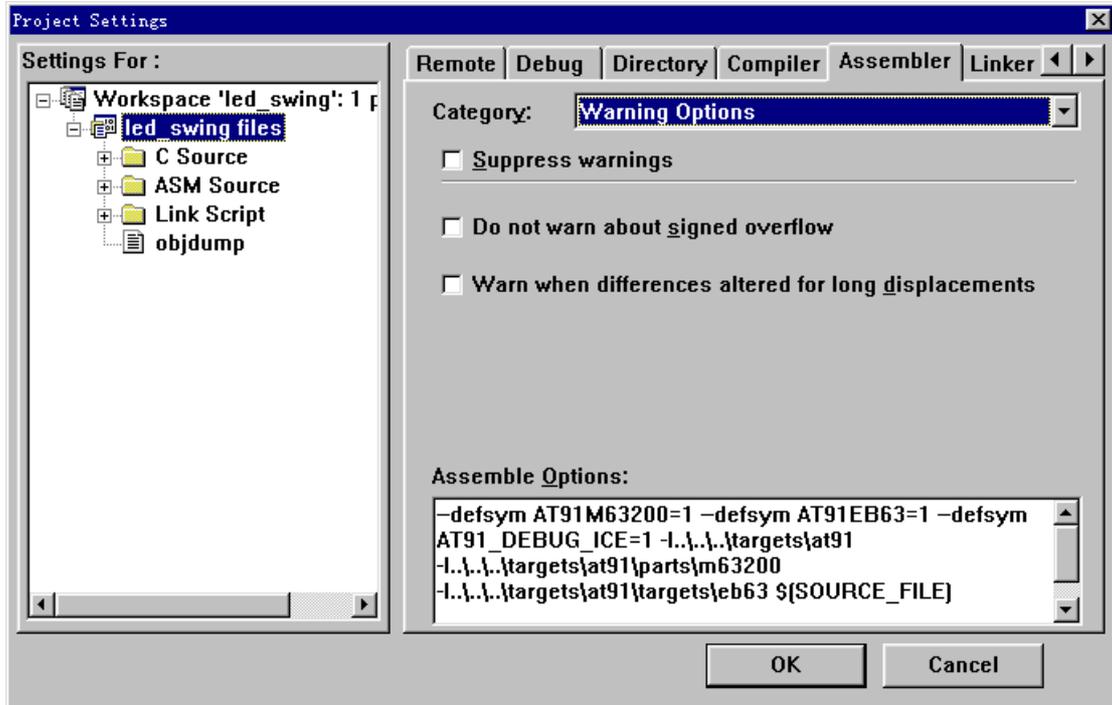


Figure 6-2-9 Assembler Warning Options

| Warning Options | Cluster | Description |
|--|---------|-------------|
| Suppress Warnings | -W | |
| Do not warn about signed overflow | -J | |
| Warn when differences altered for long displacements | -K | |

6.2.4 Options for Linker

The linker property page, as in the Figure 6-12, is used to configure the link options for linker of GNU Tools for ARM cross-compiler, All the options user select are displayed in the Link Options edit box with the following format as the output file is executable:

```
[Opt-1] ... -o[Path]$(TARGET_NAME) $(OBJ_FILES) [Lib-1] ...
```

The \$(TARGET_NAME) is a macro for executable file name, \$(OBJ_FILES) is also a macro for the collection of all object files to be linked.

As the target file is a library, the format of the options is:

```
[Opt-1] ... $(TARGET_NAME) $(OBJ_FILES) [Lib-1] ...
```

The \$(TARGET_NAME) is a macro for library name.

Note: You can input or modify the options manually in the edit box, but the blank character between each option must be reserved, and the macros \$(TARGET_NAME), \$(OBJ_FILES) should not be deleted or modified. There will be replaced with the actual file name by the Embest IDE at the time of building.

Note: \$(TARGET_NAME) will be replaced with the default that consists of project name and postfix, elf or lib.

Note:Macro \$(<entry.o>OBJ_FILES), for example, means that the file entry.o is the first object file in the collection of all object files to be linked.

Show in Figure 6-2-10, linker options are divided into five Categories:

- **General**
- **Image Entry Options**
- **Code Generation Options**
- **Include Object and Library Modules**
- **Add Library Searching Path**

Linker General Settings

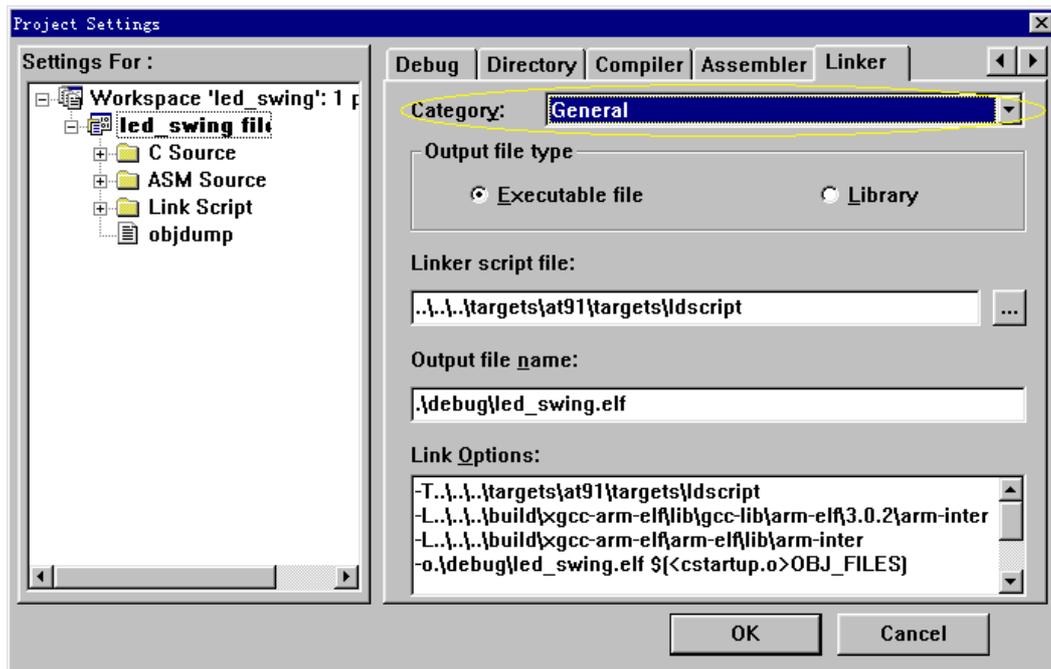


Figure 6-2-10 Linker General Settings

| General Cluster | Description |
|--------------------|---------------------------------------|
| Executable file | The output file is an executable file |
| Library | The output file is a library |
| Linker script file | The link script file |
| Output file name | The file name of output file |

Image Entry Options

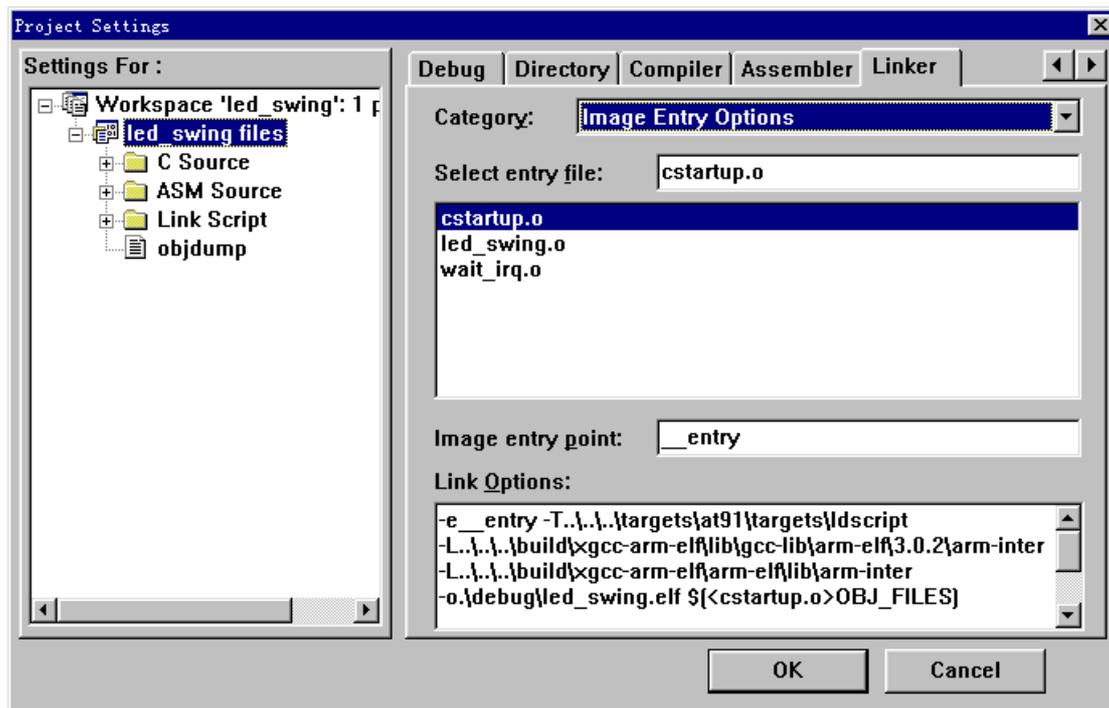


Figure 6-2-11 Linker Image Entry Options

| Image Entry Options | Cluster | Description |
|---------------------|---------|---|
| Select entry file | | The collection of all object files to be linked is displayed in a list box, and you can select one as the first object file to be linked. |
| Image entry point | | <code>-e<address/symbol></code> , The entry point of the executable target. |

Note: Image Entry Options Cluster can be set while the output file is executable.

Code Generation Options

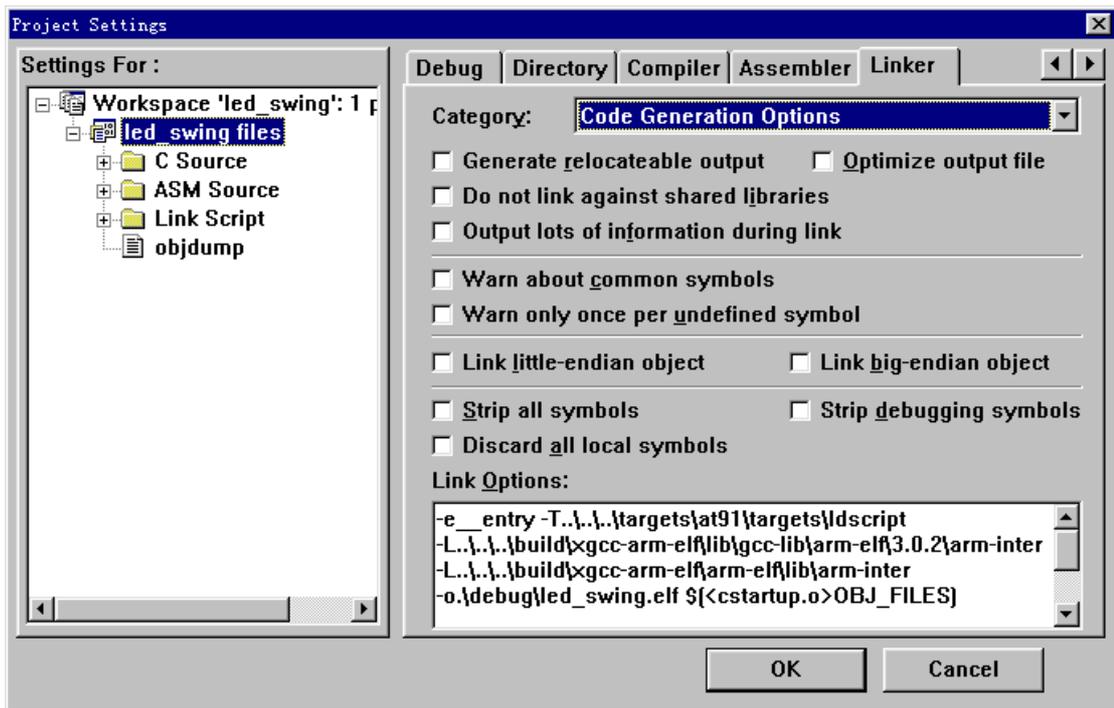


Figure 6-2-12 Linker Code Generation Options

| Code Generation Options | Description |
|--|--|
| Cluster | |
| Generate relocateable output | -r, Generate relocateable output |
| Optimize output file | -O1, Optimize output file |
| Do not link against shared libraries | -static, Do not link against shared libraries |
| Output lots of information during link | -verbose, Output lots of information during link |
| Warn about common symbols | --warn-common, Warn about duplicate common symbols |
| Warn only once per undefined symbol | --warn-once, Warn only once per undefined symbol |
| Link little-endian object | -EL, Link little-endian objects |
| Link big-endian object | -EB, Link big-endian objects |
| Strip all symbols | -s |

| | |
|---------------------------|----|
| Strip debugging symbols | -S |
| Discard all local symbols | -x |

Note: Code Generation Options Cluster can be set while the output file is executable.

Include Object and Library Modules

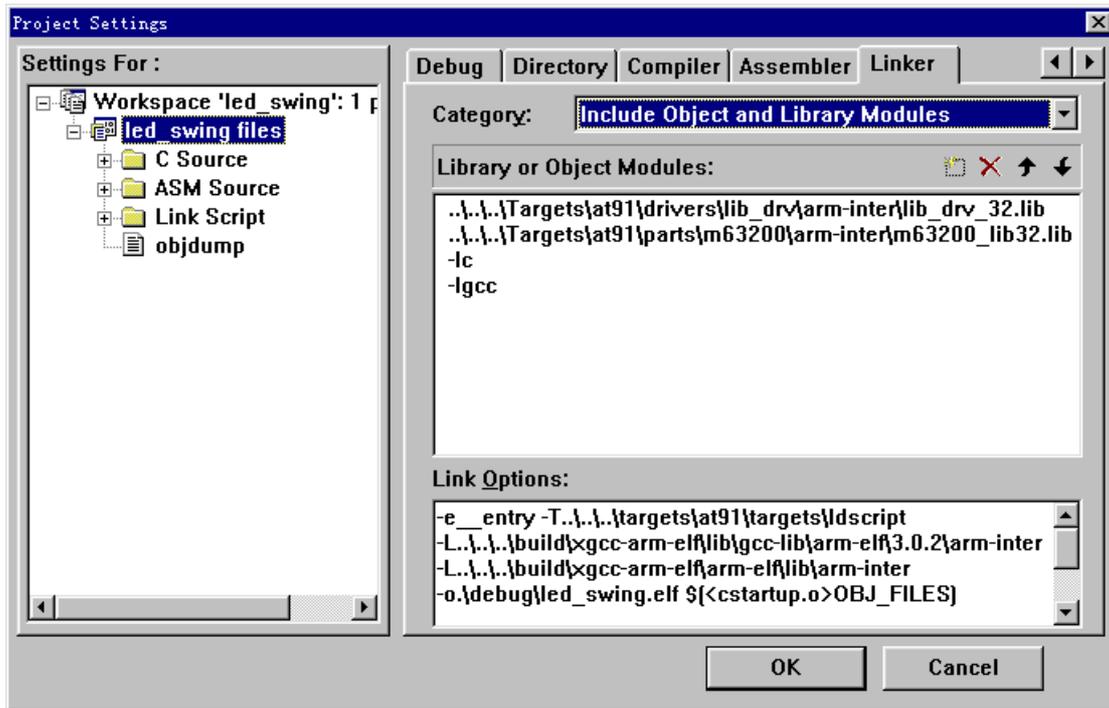


Figure 6-2-13 Linker Include Object and Library Modules

| Include Object and Library Modules | Description |
|------------------------------------|---|
| Library or Object Modules | The libraries that should be linked against |

Add Library Searching Path

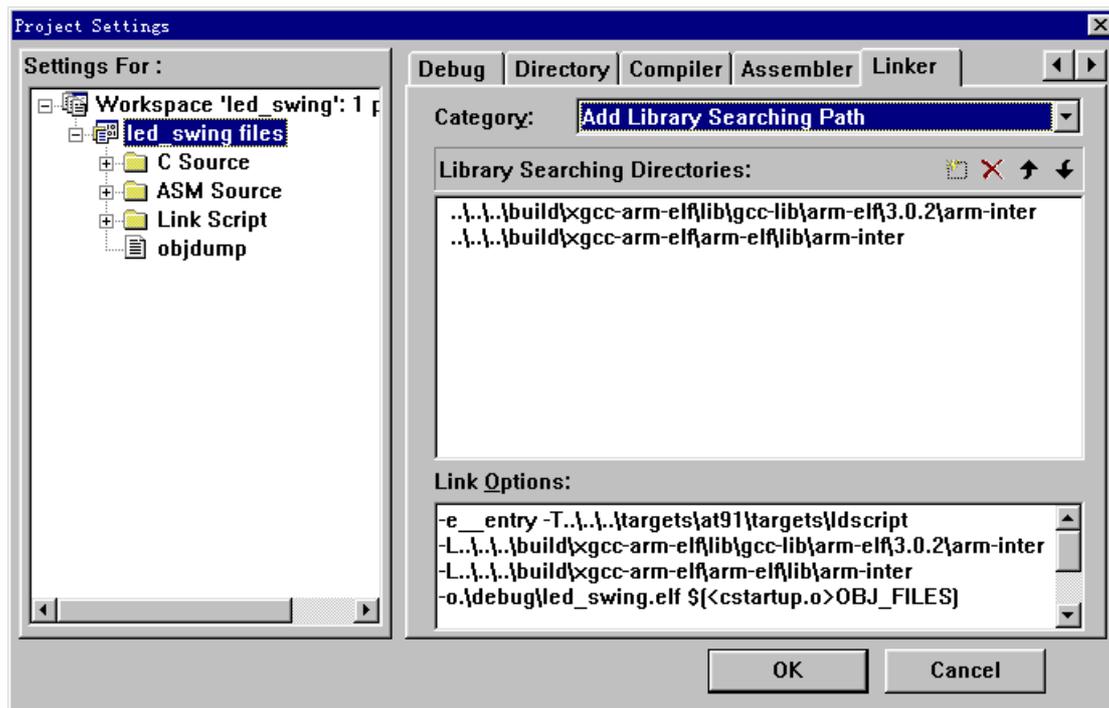


Figure 6-2-14 Linker Add Library Searching Path

| Add Library Searching Path | Description |
|-------------------------------|---|
| Library Searching Directories | -L<Directory>, Add directory to library search path |

Note: Add Library Searching Path can be set while the output file is executable.

Note: The detailed descriptions of all the options of GNU Tools for ARM cross-compiler can be found in <<Embest IDE Program Reference>>

6.3 ARM Build Tools

ARM Build Tools is a dedicative cross-compiler for ARM processor, created from ARM source code. It mainly includes C/C++ compiler, assembler, linker, standard libraries for embedded system and other tools.

Compiling device supported in Embest IDE compatible SDT 2.51 edition at present, this edition compiles and chains tools including ARM C/C++ compiling device Abbreviated as ARMCC , ARM collecting device (abbreviated as ARMASM) and ARM chaining device(abbreviate as ARMLINK).

6.3.1 ARM Build Tools and Files

Which application of the cross-compiler will be invoked by EmbestIDE to handle a source file lies on its postfix. Figure 6-3-1 shows each postfix and its relevant application.

Figure 6-3-1

| Postfix | Application of Cross-Compiler |
|---------|-------------------------------|
| *.c | C Compiler |
| *.C | C++ Compiler |
| *.cpp | C++ Compiler |
| *.cc | C++ Compiler |
| *.cp | C++ Compiler |
| *.c++ | C++ Compiler |
| *.cXX | C++ Compiler |
| *.s | Assembler |
| *.asm | Assembler |

Files type of ARM tools chain shows as Figure 6-3-2.

Figure 6-3-2

| Files | Files Type |
|------------------|------------------------------|
| file.stuf | source files of ARM |
| file.h/.inc/.alf | head file |
| file.o | ELF formation target file |
| file.elf | ELF formation debugging file |

stuf stand for *.c/C, *.cpp/C++, *.s/asm, are the source files which can compiling or assembling by ARM tools chain(Note: it is permitted that there is the **blank** within the filename or directory.)

6.3.2 ARM Compiler options setting

The compiler attribute tab of the ARM Build Tools compiler is illustrated in Fig. 6-3. There are altogether 7 categories of list (namely, setting option), namely, General, Target Specific, Warning Options, Error Handling, Debug & Optimization, Code Generation and Include Paths. The various configuration options are used for ARMCC compiler. All the settings of the user will be displayed in the Compile Option edit box in the form of command line switch options. When it is the first time for a new project to select ARM Build Tools, the system will provide the compiler's default setting.

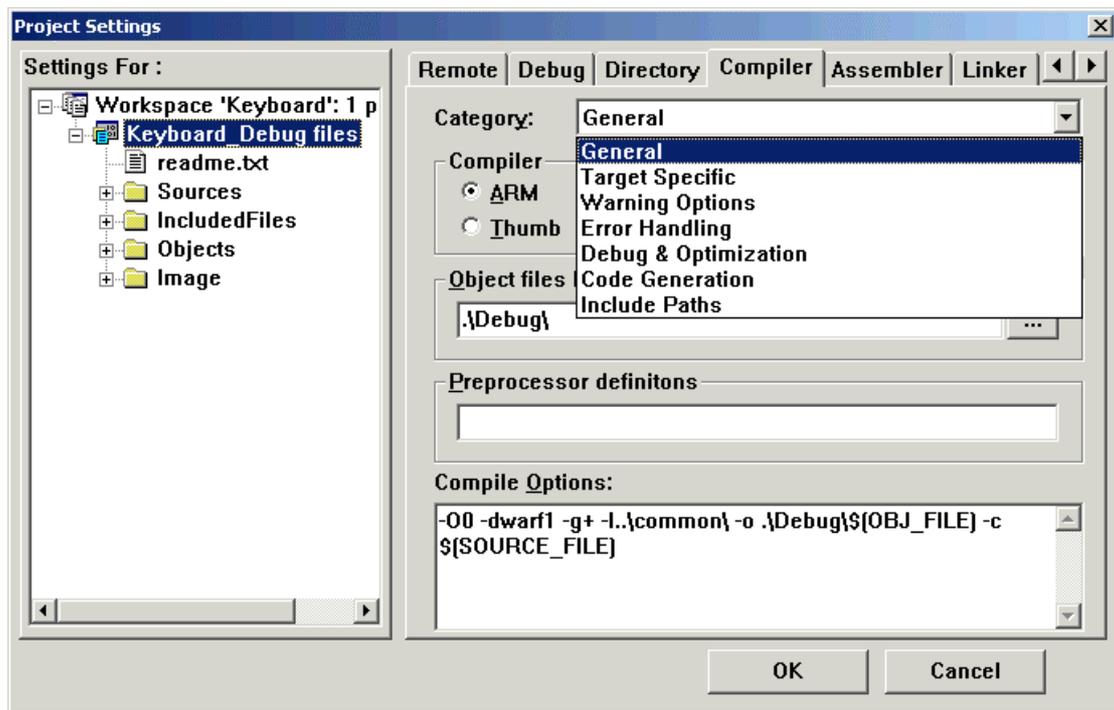


Fig. 6-3-1 Project compiler setting and category options list

General

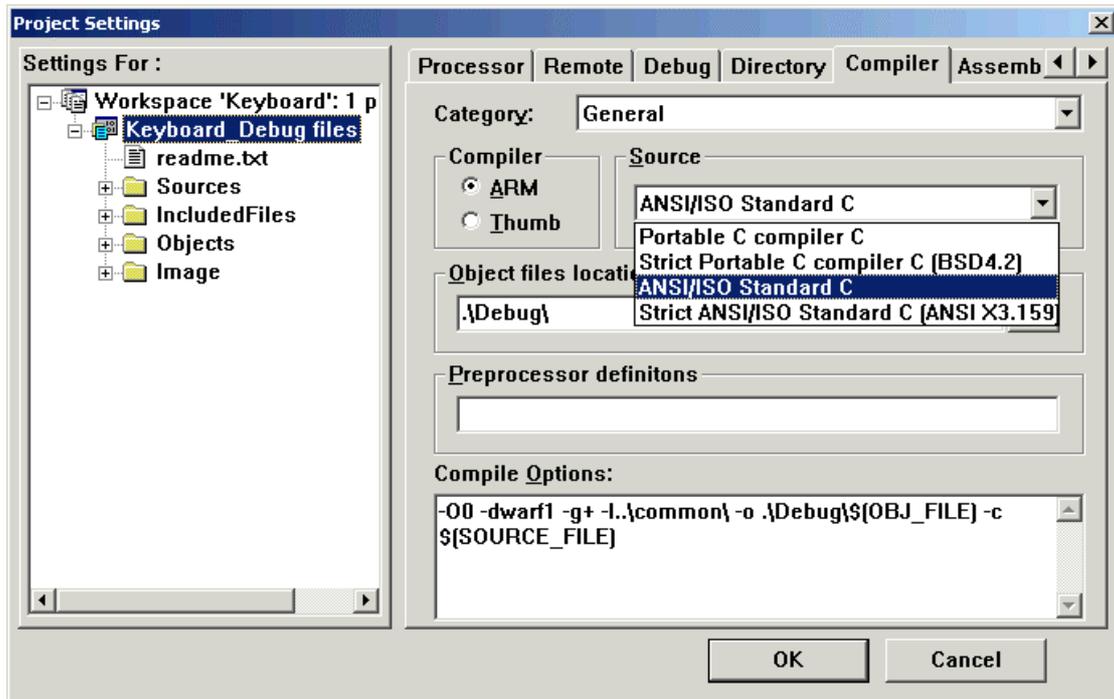


Figure 6-3-2 settings of General

shows as Figure 6-3-1, in the Compiler dialog window, click the Category drawing menu, select the option in the Source dialog window, the definition as following:

| General Cluster | Description |
|-----------------------|---|
| Compiler | instruction format of the Target file ARM supports ARM code Thumb supports Thumb code |
| Source | source files type |
| Portable C compiler C | -pcc, Compiles (BSD 4.2) Portable C compiler C. This dialect is based on the original Kernighan and Ritchie definition of C, and is the one used to build UNIX systems. The -pcc option alters the language accepted by the compiler, however the built-in ANSI C headers are still used. See also, the -zc option in Controlling code generation. The -pcc |

option alters the language accepted by the compilers in the following ways:

- char is signed
- sizeof is signed
- an approximation of early UNIX-style C preprocessing is used.

Strict Portable C compiler -pcc -strict, is extra strict about enforcing conformance to the ANSI C standard, Draft C++ standard, or PCC conventions. For example, in C++ mode the following code gives an error when compiled with -strict and a warning without:

```
static struct T {int i; };
```

Because no object is declared, the static is spurious. In a strict reading of the C++ Draft Standard, it is illegal.

ANSI/ISO standard C -ansi, Compiles ANSI standard C. This is the default for armcc

Strict ANSI/ISO standard C[ANSI X3.159] -ansi -strict

Object file locations settings the directory to store object file(s)

Preprocessor Definitions Defines symbol as a preprocessor macro, as if the following line were at the head of the source file: #define symbol [value] This option can be repeated.

Target Specific

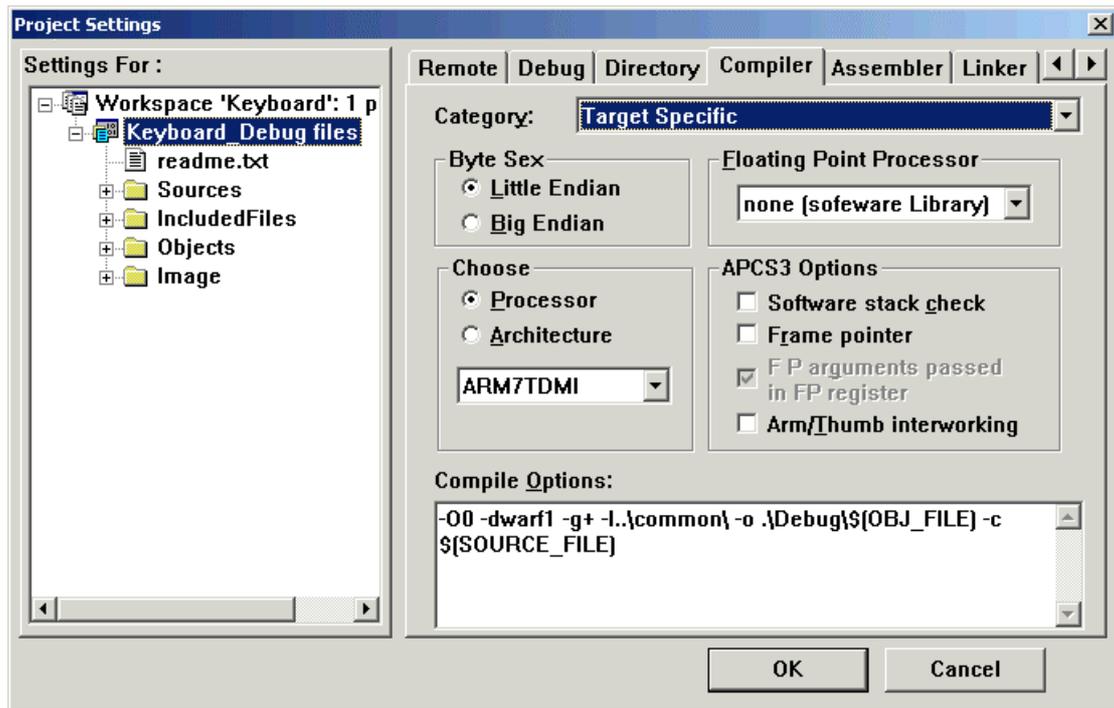


Figure 6-3-3 settings of Target Specific

shows as Figure 6-3-1, in the Compiler dialog window, click the Category drawing menu, select Target Specific, the definition of compile as following:

| Target Specific Cluster | Description |
|-------------------------|--|
| Little Endian | -li, instructs suitable for Little Endian ARM |
| Big Endian | -bi, instructs suitable for Big Endian ARM |
| Processor | instruction code support by processor |
| Architecture | -arch, sets the target architecture. Some processor-specific instructions produce either errors or warnings if assembled for the wrong target architecture. See also the -unsafe assembler option. Valid values for architecture are 3, 3m, 4, 4T, 4TxM. |
| Float point processor | -fpu, select the target FPU, where name is one of: |

| | |
|---------------------------------|--|
| | <p>none No FPU. Use software floating point library. This option implies /softfp.</p> <p>fpa Floating Point Accelerator. This option implies /hardfp.</p> |
| Software stack check | -apcs /swst, specifies that the code in inputfile carries out software stack checking. |
| Frame pointer | <p>-apcs /fp, specifies that the code in inputfile uses a frame pointer.</p> <p>This option is obsolete and is provided for backwards compatibility only.</p> |
| FP arguments passed in register | <p>FP (selected) specifies that the code in inputfile does not use a frame pointer. This is the default.</p> |
| ARM/Thumb interworking | -apcs /inter, specifies that the code is suitable for ARM/Thumb interworking. This option has the same effect as specifying the INTERWORK attribute for all code areas in the source files to be assembled. Refer to the ARM Software Development Toolkit User Guide for more information on ARM/Thumb interworking. |

Warning Options

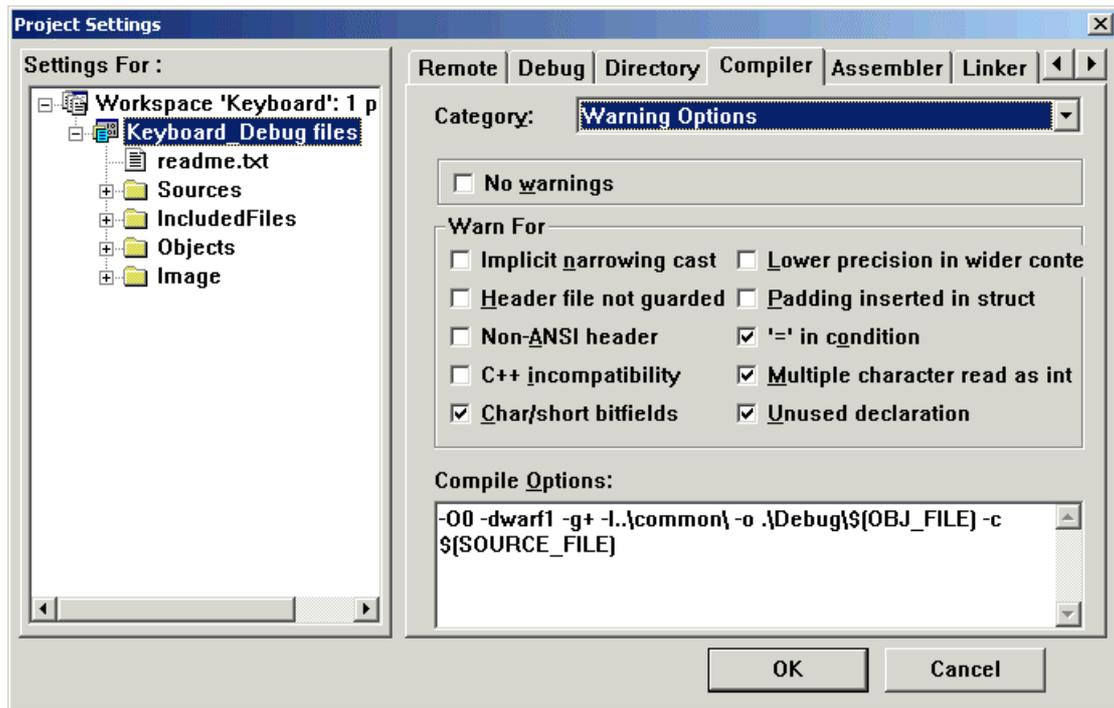


Figure 6-3-4 settings of Warning Options

shows as Figure 6-3-1, in the Compiler dialog windows, click the Category drawing menu to set the warning information of the compile.

The compiler issues warnings to indicate potential portability problems or other hazards. The compiler options described below allow you to turn specific warnings off.

For example, you may wish to turn warnings off if you are in the early stages of porting a program written in old-style C. The options are on by default, unless specified otherwise. See also Specifying additional checks on page 2-32 for descriptions of additional warning messages.

The general form of the `-W` compiler option is:

`-W[options][+][options]`

where options are one or more characters.

If the `+` character is included in the characters following the `-W`, the warnings corresponding to any following letters are enabled rather than suppressed.

You can specify multiple options. For example:

-Wad+fg

turns off the warning messages specified by a, d, and turns on the warning message specified by f and g.

Following description for the **Warning Options** shows in Figure 6-3-4:

| Warning Options | Cluster | Description |
|-------------------------|----------------|--|
| No Warnings | | -W, suppresses all warnings. If one or more letters follow the option, only the warnings controlled by those letters are suppressed. |
| Implicit narrowing cast | | <p>-W+n Suppresses the warning message:</p> <p>implicit narrow cast</p> <p>This warning is issued when the compiler detects the implicit narrowing of a long expression in an int or char context, or the implicit narrowing of a floating-point expression in an integer or narrower floating-point context.</p> <p>Such implicit narrowing casts are almost always a source of problems when moving code that has been developed on a fully 32-bit system (such as ARM C++) to a system in which integers occupy 16 bits and longs occupy 32 bits. This is suppressed by default.</p> |
| Header file not guarded | | <p>-W+g, Suppresses the warning given if an unguarded header file is #included.</p> <p>This warning is off by default. It can be enabled with -W+g. An unguarded header file is a header file not wrapped in a declaration such as:</p> <pre>#ifdef foo_h</pre> |

```
#define foo_h
/* body of include file */
#endif
```

Non-ANSI header

-W+p, Suppresses the warning message:

non-ANSI #include <...>

The ANSI C standard requires that you use #include <...> for ANSI C

headers only. However, it is useful to disable this warning when compiling code not conforming to this aspect of the standard. This option is suppressed by default, unless the -fussy option is specified.

C++ incompatibility

-W+u, For C code, suppresses warnings about future compatibility with C++ for both armcpp and tcpp. This option is off by default. It can be enabled with -W+u.

Char/short bitfields

-Wb, Suppresses the warning message:

ANSI C forbids bit field type 'type' where 'type' is char or short.

Lower precision in wider content

-W+l, **Lower precision in wider context.**

This warning arises in cases like:

```
long x; int y, z; x = y*z
```

where the multiplication yields an int result that is then widened to long. This warns in cases where the destination is long long, or where the target system defines 16-bit integers or 64-bit longs. This option is off by default. It can be enabled with -W+l.

Padding inserted in struct

-Ws, Warns when **the compiler inserts**

padding in a struct. This warning is off by default. It can be enabled with -W+s.

'=' in condition

-Wa, Suppresses the warning message:

Use of the assignment operator in a condition context

This warning is given when the compiler encounters a statement such as:

if (a = b) {...

where it is possible that:

if ((a = b) != 0) {...

was intended, or that:

if (a == b) {...

was intended. This warning is suppressed by default in PCC mode.

Multiple character read as int -Wm

Unused declaration -Wx, Disables not used warnings such as:

Warning: function 'foo' declared but not used

Note: User can input the warning switch(s) directly in Compiler options

Error Handling

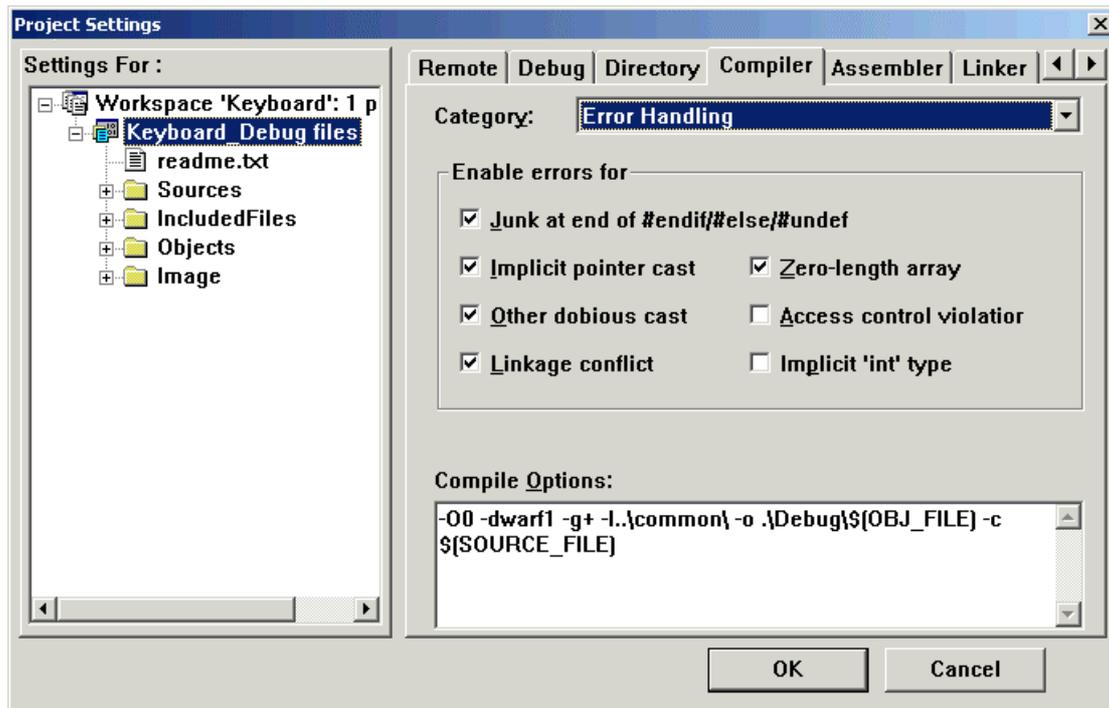


Figure 6-3-5 settings of Error Handling

shows as Figure 6-3-1, in the Compiler dialog window, click the Category drawing menu, select Error Handling, the definition of close and down lever in the compile error information compiler as following:

The compiler issues errors to indicate that serious problems exist in the code it is attempting to compile.

The compiler options described below allow you to:

- turn specific recoverable errors off
- downgrade specific errors to warnings.

The general form of the `-e` compiler option is:

`-e[options][+][options]`

where options are one or more of the letters described below.

If the `+` character is included in the characters following the `-e`, the errors corresponding to any following letters are enabled rather than suppressed.

You can specify multiple options. For example:

`-eac`

turns off the error messages specified by a and c

Following description for the **Warning Options** shows in Figure 6-3-5:

| Error Handling Cluster | Description |
|----------------------------------|---|
| Junk at end of #end/#else/#undef | -Ep, Suppresses the error that occurs if there are extraneous characters at the end of a preprocessor line. This error is suppressed by default in PCC mode. |
| Implicit pointer cast | -Ec, Suppresses all implicit cast errors, such as implicit casts of a non-zero int to pointer. |
| Other double cast | -Ef, Suppresses errors for unclear casts, such as short to pointer. |
| Linkage conflict | -El, Suppresses errors about linkage disagreements where functions are implicitly declared extern and later defined as static. This option applies to C++ only. |
| Zero-length array | -Ez, Suppresses the error that occurs if a zero-length array is used. |
| Access control violation | -E+a, This option applies to C++ only. Downgrades access control errors to warnings. For example: class A { void f() {} }; // private member A a; void g() { a.f(); } // erroneous access |
| Implicit "int" type | -E+I, Downgrades constructs of the following kind from errors to warnings. For example: const i; |

Error: declaration lacks
type/storage-class (assuming 'int'): 'i'

This option applies to C++ only.

Debug & Optimization

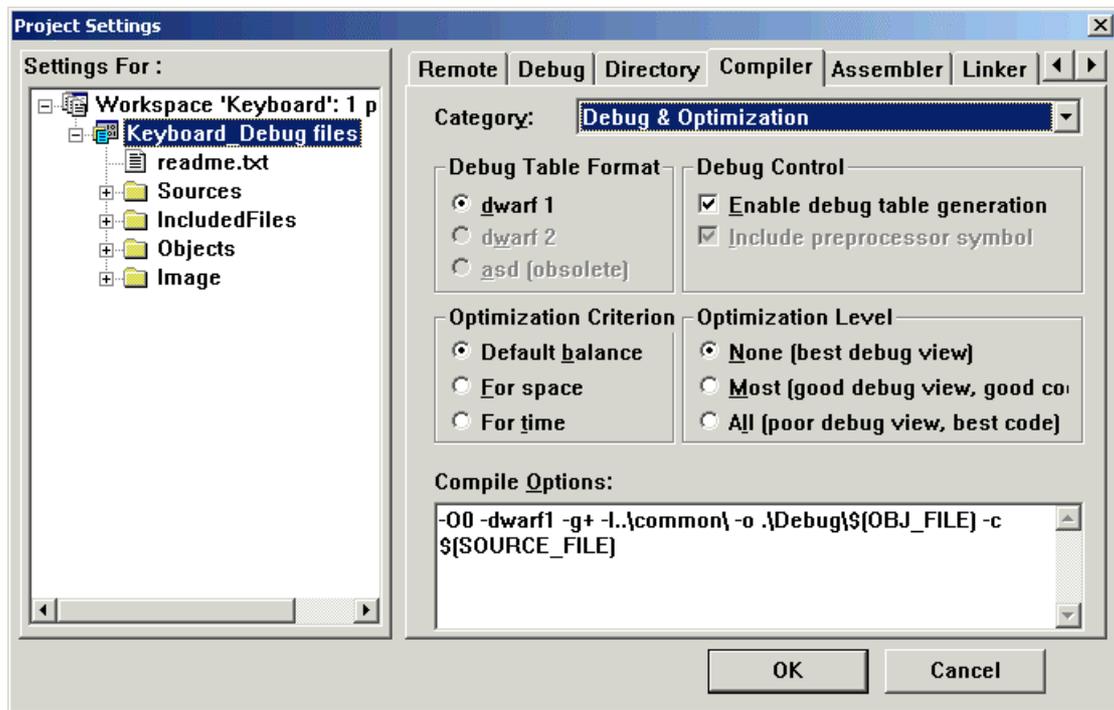


Figure 6-3-6 settings of Debug & Optimization

shows as Figure 6-3-1, in the Compiler dialog window, click the Category drawing menu, select Debug * Optimization, the definition as following:

| Debug & Optimization Cluster | Description |
|------------------------------|--|
| dwarf 1 | -dwarf1, Use DWARF1 debug table format. This option is not recommended for C++. If DWARF1 debug tables are generated and a procedure call standard that does not use a frame-pointer register is used (always |

the case with Thumb, and the default with ARM), local variables that have been allocated to the stack cannot be displayed by the debugger. In addition, stack backtrace is not possible.

dwarf 2

`-dwarf2`, to select DWARF2 debug tables. This is the default and is selected if `-g` with no dwarf option is specified. This is the default.

asd (obsolete)

`-asd`, Use ASD debug table format. This option is obsolete and is provided for backwards compatibility only.

Enable debug table generation

`-g`, instructs the assembler to generate debug tables. Use the following command-line options to control the behavior of `-g`:

`-dwarf` to select DWARF1 debug tables. This option is obsolete. Use `-dwarf2` or `-dwarf1`.

`-dwarf1` to select DWARF1 debug tables. This option is not recommended for C++.

`-dwarf2` to select DWARF2 debug tables. This is the default and is selected if `-g` with no dwarf option is specified.

Include preprocessor symbol

Not use

| | |
|------------------------|--|
| Default balance | <p>If neither <code>-Otime</code> or <code>-Ospace</code> is specified, the compiler uses a balance between the two. You can compile time-critical parts of your code with <code>-Otime</code>, and the rest with <code>-Ospace</code>. You should not specify both <code>-Otime</code> and <code>-Ospace</code> at the same time.</p> |
| For space | <p><code>-Ospace</code>, Optimize to reduce image size at the expense of increased execution time.</p> <p>For example, large structure copies are done by out-of-line function calls instead of inline code.</p> |
| For time | <p><code>-Otime</code>, Optimize to reduce execution time at the expense of a larger image.</p> <p>For example, compile:</p> <pre>while (expression) body...; as: if (expression) { do body...; while (expression); }</pre> |
| None (best debug view) | <p><code>-O0</code>, Turn off all optimization, except some simple source transformations. This is the default optimization level if debug tables are generated with <code>-g+</code>. It gives the best debug view and the lowest level of optimization.</p> |

Most (good debug view good code) -O1, Turn off the following optimizations:

- structure splitting
- range splitting
- cross-jumping
- conditional execution.

If this option is specified and debug tables are generated with -g+ it gives a satisfactory debug view with good code density.

All (poor debug view best code) -O2, Generate fully optimized code. If used with -g+, this option produces fully optimized code that is acceptable to the debugger, though the mapping of object code to source code is not always clear. This is the default optimization level if debug tables are not generated.

Code Generation

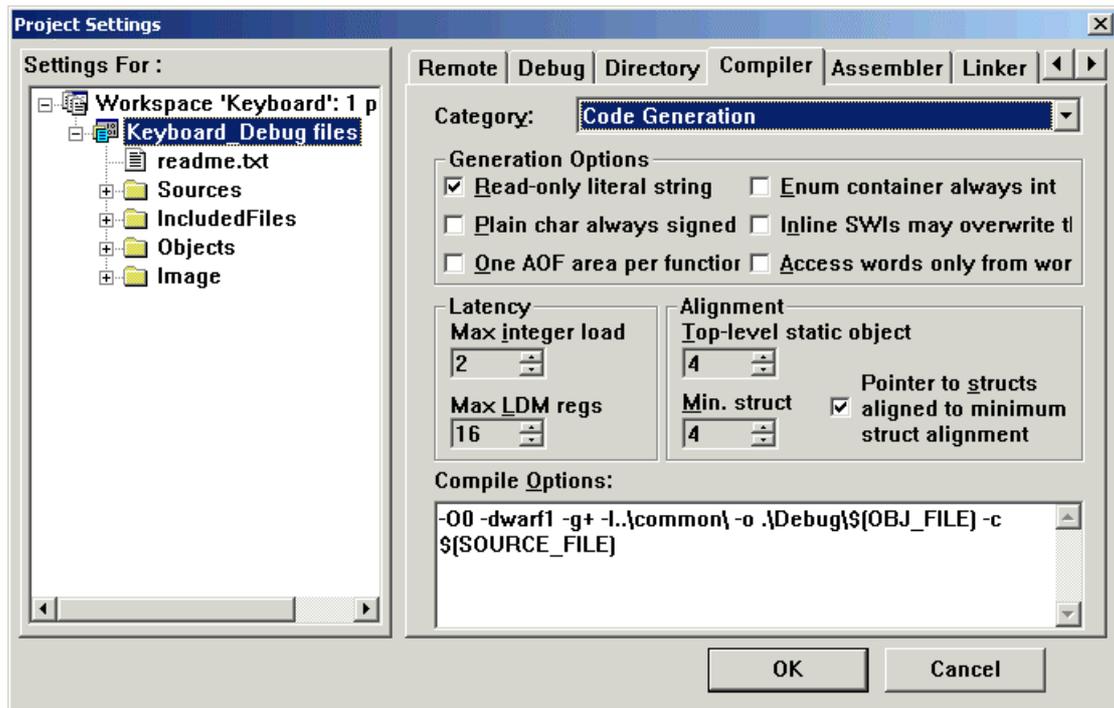


Figure 6-3-7 settings of Code Generation

shows as Figure 6-3-1, in the Compiler dialog window, click the Category drawing menu, select Code Generation, the definition of the compiler output target files as following:

| Code Generation Cluster | Description |
|--------------------------|--|
| Read-only literal string | -fw, Allows string literals to be writable, as expected by some UNIX code, by allocating them in the program data area rather than the notionally read-only code area. This also stops the compiler reusing a multiply occurring string literal. |
| Plain char always signed | -zc, Make char signed. It is normally unsigned in C++ and ANSI C modes, and signed in PCC mode. The sign of char is set by the last option |

specified that would normally affect it. For example, if you specify both the -ansic and -zc options and you want to make char signed in ANSI C mode, you must specify the -zc option after the -ansic option.

One AOF area per function

-zo, Generates one AOF area for each function. This can result in increased code size. Normally the compiler generates one AOF function for each C compilation unit, and at least one AOF function for each C++ compilation unit. This option enables the linker to remove unused functions when the **-remove** linker option is specified.

Enum container always int

-fy, Treats enumerations as signed integers. This option is off by default (no forced integers).

Inline SWIs may overwrite the Link

-fz, Instructs the compiler that an inline SWI may overwrite the contents of the link register. This option is usually used for modules that run in Supervisor mode, and that contain inline SWIs. You must use this option when compiling code that contains inline SWIs.

Access word only from word aligned

-za0/1, Specifies whether LDR may only access word-aligned addresses. Valid values are:

Max integer load

-za0 LDR is not restricted to accessing word-aligned addresses. This is the default.

-za1 LDR may only access word-aligned addresses.

-zi2, The compiler selects a value for the maximum number of instructions allowed to generate an integer literal inline before using LDR rx,= value on the basis of the -Otime, -Ospace, and -processor options.

You can alter this behavior by setting Number to an integer between 1 and 4. Lower numbers generate less code at the possible expense of speed, depending on your memory system. The effect of altering this value is small, and is usually not significant.

Max LDM regs

-zr16, Limits the number of register values transferred by load multiple and store multiple instructions generated by the compiler to Number. Valid values for Number are 3 to 16 inclusively. The default value is 16.

You can use this option to reduce interrupt latency. Note that the inline assembler is not subject to the limit imposed by the -zr option.

The Thumb compiler does not support

| | |
|--|---|
| Top-level static object | <p>this option.</p> <p><code>-zat4</code>, Specifies the minimum byte alignment for top-level static objects, such as global variables. Valid values for Number are: 1, 2, 4, 8</p> <p>The default is 4 for the ARM compilers and 1 for the Thumb compilers.</p> |
| Min. struct | <p><code>-zas4</code>, Specifies the minimum byte alignment for structures. Valid values for Number are: 1, 2, 4, 8</p> <p>The default is 4 for both ARM and Thumb compilers. This allows structure copying to be implemented more efficiently by copying in units of words, rather than bytes. Setting a lower value reduces the amount of padding required, at the expense of the speed of structure copying.</p> |
| Pointer to structs aligned to minimum struct alignment | <p>Specifies whether pointers to structures are assumed to be aligned on at least the minimum byte alignment boundaries, as set by the <code>-zas</code> option. Valid values are:</p> <p>-zap1 Pointers to structures are assumed to be aligned on at least the minimum byte alignment boundaries set by <code>-zas</code>. This is the default.</p> <p>-zap0 Pointers to structures are not assumed to be aligned on at least the</p> <hr/> |

minimum byte alignment boundaries set by `-zas`. Casting `short[]` to `struct {short, short,...}` does not cause a problem.

Include Paths

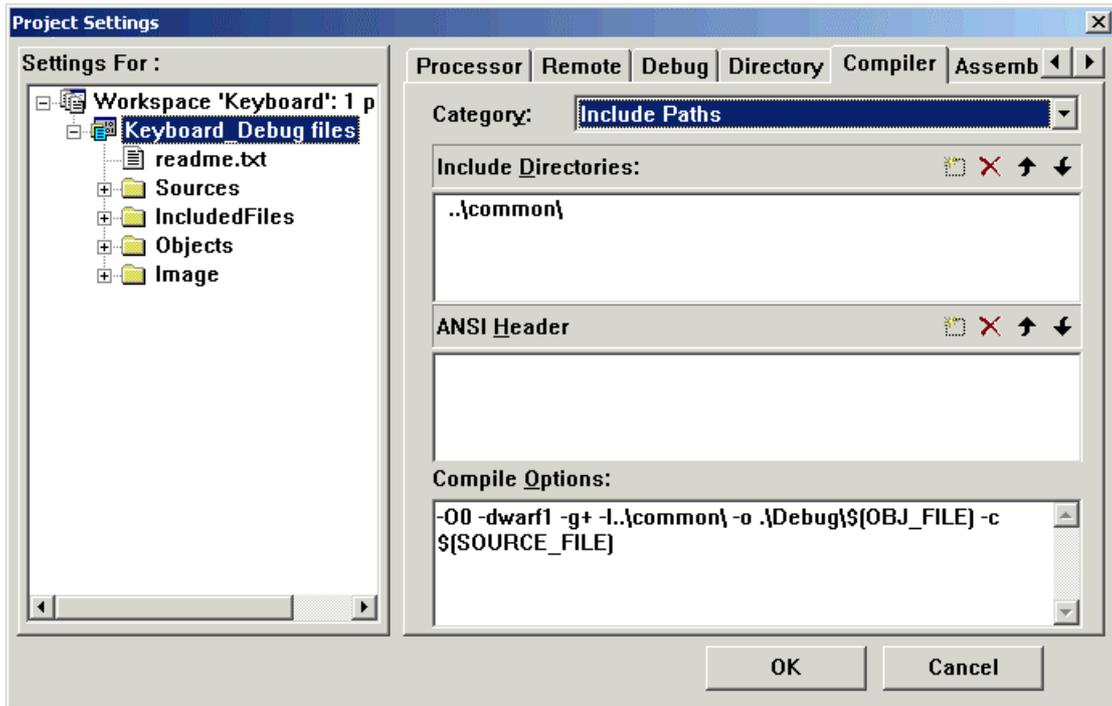


Figure 6-3-8 settings of Include Paths

shows as Figure 6-3-1, in the Compiler dialog window, click the Category drawing menu, set the search folder of the user define head files and the ANSI lib head files compiler, definition as following:

| Include Paths Cluster | Description |
|-----------------------|--|
| Include Directories | -i<dir> , adds directories to the source file search path so that arguments to GET/INCLUDE directives do not need to be fully qualified. |
| ANSI Header | -j, Adds the specified comma-separated list of directories to the end of the search path, after |

all directories specified by -I options. Use -j- to search the in-memory file system.

Compile Options Window

The Compiler property page, as in the figure 6-3-1, is used to configure the compile options for C/C++ compiler of ARM Build Tools cross-compiler, All the options user selected are displayed in the Compile Options edit box with the following format:

[Opt-1] [Opt-2] ...-o[Path]\$(OBJ_FILE) ... [Opt-n] ...\$(SOURCE_FILE)

Note: You can input or modify the options manually in the edit box, but the blank character between each option must be reserved, and the macros \$(SOURCE_FILE), \$(OBJ_FILE) should not be deleted or modified. \$(SOURCE_FILE) means the source file to be compiled, \$(OBJ_FILE) means the output of the compiling. There will be replaced with the actual file name by the EmbestIDE at the time of building.

Note: When you configure a project setting, you should consider that the location of the project file (.pjf) is the current directory.*

shows as Figure 6-3-1, the Switch and meaning of the command in the Compile Options Window:

| Switch | Description |
|-------------------------|---|
| -O0 | Turn off all optimization, except some simple source transformations. |
| -dwarf1 | Compile output file by dwarf1 format |
| -g+ | The target file include debug information figure (Function as -g) |
| -I.\common\ | Use the common file folder which in the project base forward folder as the search catalog of head files |
| -o .\Debug\\$(OBJ_FILE) | Compile output target file to the Debug folder |
| -c | Only compile C language program without link |
| \$(SOURCE_FILE) | Compile all source files of the active project |

6.3.3 ARM Assembler options setting

The Assembler attribute tab of the ARM Build Tools compiler is illustrated in Fig. 6-3-9. There are altogether 6 categories of list (namely, setting option), namely, General, Target Specific, Call Standard Options, Debug Options, Predefines and Listing Options. The various configuration options are used for ARMASM compiler. All the settings of the user will be displayed in the Assemble Options edit box in the form of command line switch options. When it is the first time for a new project to select ARM Build Tools, the system will provide the compiler's default setting.

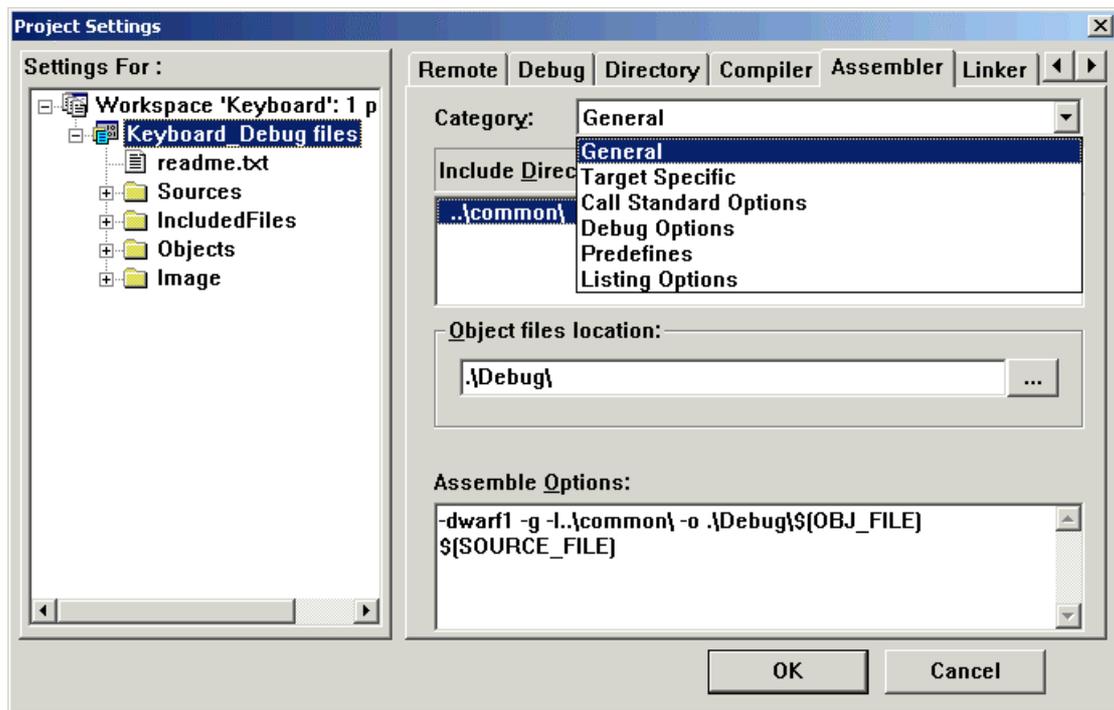


Fig. 6-3-9 Project compiler setting and category options list

General

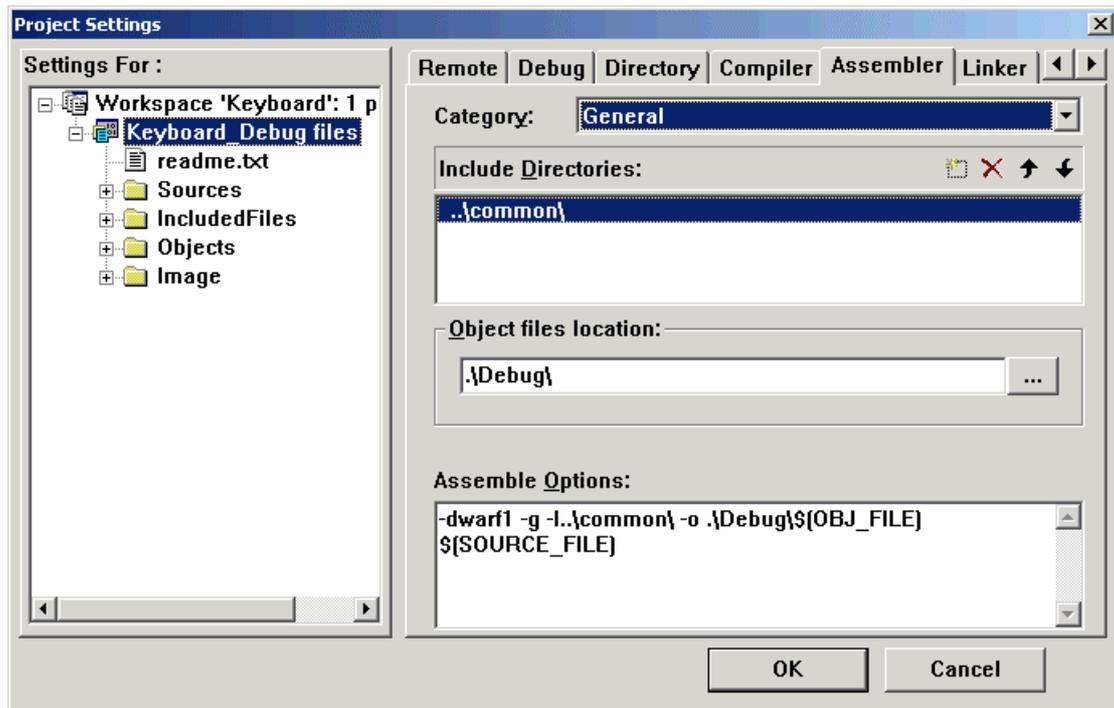


Figure 6-3-10 settings of General

shows as Figure 6-3-9, In the Assembler dialog window, click the Category drawing menu, choose General, the search and assembler output file folder of the head source files, definition as following:

| General Cluster | Description |
|-----------------------|---|
| Include Directories | -i<dir> , adds directories to the source file search path so that arguments to GET/INCLUDE directives do not need to be fully qualified. |
| Object files location | settings the directory to store object file(s) |

Target Specific

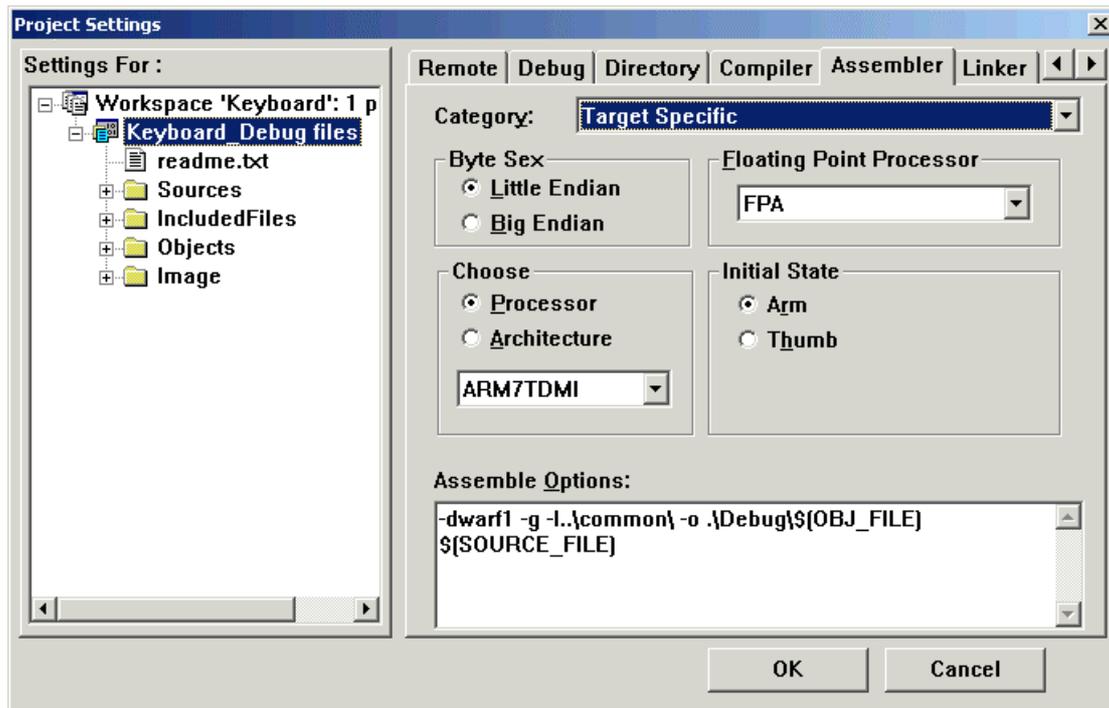


Figure 6-3-11 settings of Target Specific

shows as Figure 6-3-9, In the Assembler dialog window, click the Category drawing menu, choose Target Specific, the definition of the assembler target file, definition as following:

| Target Specific Cluster | Description |
|-------------------------|---|
| Little Endian | -li , instructs suitable for Little Endian ARM |
| Big Endian | -bi , instructs suitable for Big Endian ARM |
| Processor | instruction code support by processor |
| Architecture | -arch , sets the target architecture. Some processor-specific instructions produce either errors or warnings if assembled for the wrong target architecture. See also the <code>-unsafe</code> assembler option. Valid values for architecture are 3, 3m, 4, 4T, 4TxM. |
| Float point processor | -fpu , select the target FPU, where name is one of: |

none No FPU. Use software floating point library. This option implies /softfp.

fpa Floating Point Accelerator. This option implies /hardfp.

Initial State

instruction format of the Target file

ARM supports ARM code

Thumb supports Thumb code

Call standard Options

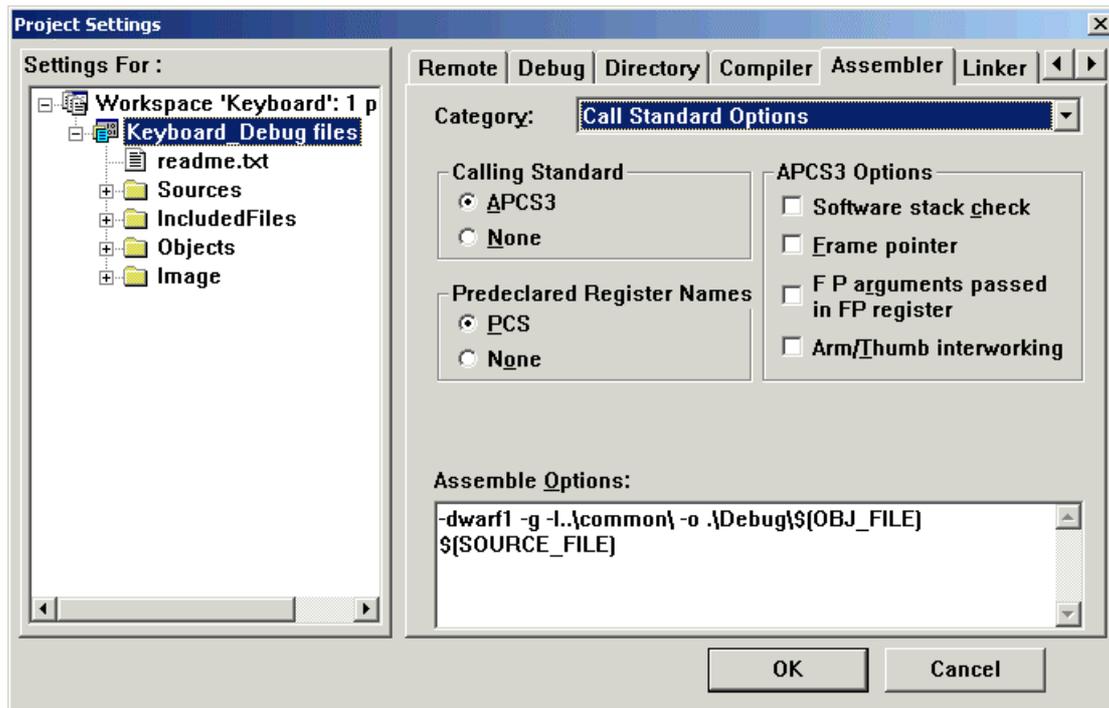


Figure 6-3-12 settings of Call standard Options

shows as Figure 6-3-9, In the Assembler dialog window, click the Category drawing menu, choose Call Standard Options, the attributes of the procedure calling in cross compile ,definition as following:

| Call Standard Options | Cluster | Description |
|-----------------------|---------|---|
| Call Standard | | <p>-apcs none/3, specifies whether you are using the ARM Procedure Call Standard or not, and may specify some attributes of code areas. User Guide for more information.</p> <p>none specifies that inputfile does not use APCS. APCS registers are not set up. Qualifiers are not allowed.</p> <p>3 specifies that inputfile uses APCS version 3. APCS registers are set up. This is the default.</p> |

| | |
|------------------------------------|--|
| Predeclared Register Names | APCS register name in the rule: PCS -- PCS rule None -- no rule |
| Software stack check | -apcs /swst , specifies that the code in inputfile carries out software stack checking. |
| Frame pointer | -apcs /fp , specifies that the code in inputfile uses a frame pointer. This option is obsolete and is provided for backwards compatibility only. |
| FP arguments passed in FP register | (selected) specifies that the code in inputfile does not use a frame pointer. This is the default. |
| ARM/Thumb interworking | -apcs /inter , specifies that the code is suitable for ARM/Thumb interworking. This option has the same effect as specifying the INTERWORK attribute for all code areas in the source files to be assembled. Refer to the ARM Software Development Toolkit User Guide for more information on ARM/Thumb interworking. |

Debug Options

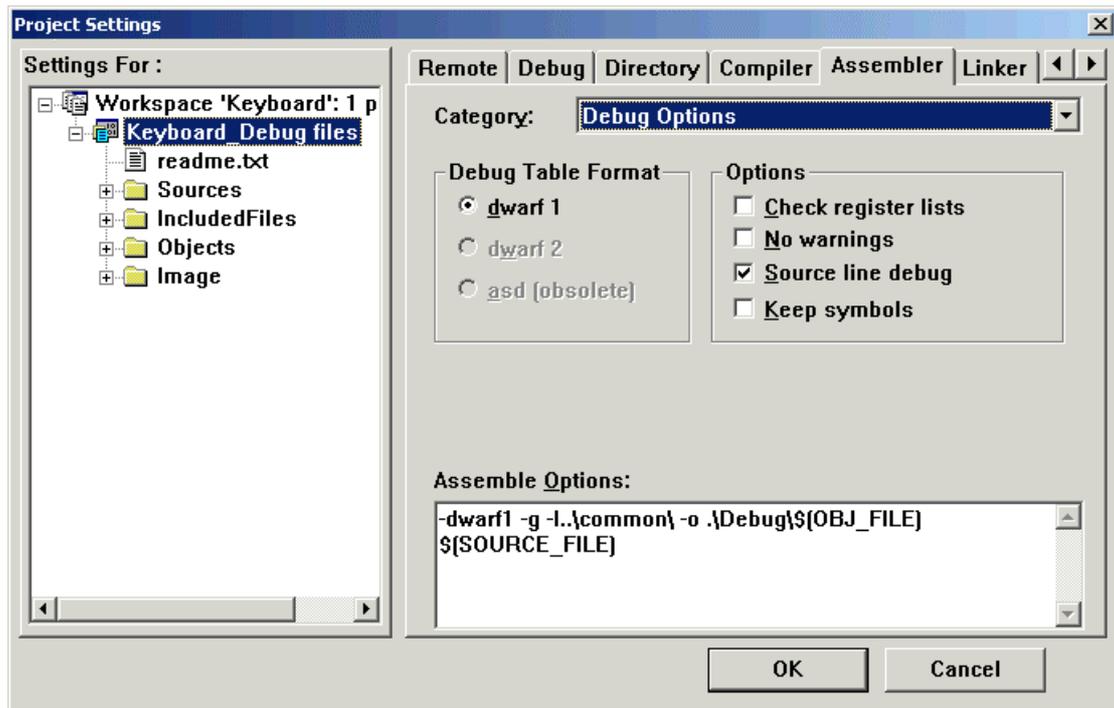


Figure 6-3-13 settings of Debug Options

shows as Figure 6-3-9, In the Assembler dialog window, click the Category drawing menu, choose Debug Options, the attributes of assemble output target files ,definition as following:

| Debug Options Cluster | Description |
|-----------------------|--|
| dwarf 1 | -dwarf1 , Use DWARF1 debug table format. This option is not recommended for C++. If DWARF1 debug tables are generated and a procedure call standard that does not use a frame-pointer register is used (always the case with Thumb, and the default with ARM), local variables that have been allocated to the stack cannot be displayed by the debugger. In addition, stack backtrace is not possible. |
| dwarf 2 | -dwarf2 , to select DWARF2 debug tables. This is the default and is selected if -g with no dwarf |

| | |
|----------------------|---|
| | option is specified. This is the default. |
| asd (obsolete) | -asd , Use ASD debug table format. This option is obsolete and is provided for backwards compatibility only. |
| Check register lists | -checkreglist , instructs the assembler to check RLIST, LDM, and STM register lists to ensure that all registers are provided in increasing register number order. If this is not the case, a warning is given. |
| No warnings | -nowarn , turns off warning messages. |
| Source line debug | -g , instructs the assembler to generate debug tables. Use the following command-line options to control the behavior of -g : -dwarf to select DWARF1 debug tables. This option is obsolete. Use -dwarf2 or -dwarf1 . -dwarf1 to select DWARF1 debug tables. This option is not recommended for C++. -dwarf2 to select DWARF2 debug tables. This is the default and is selected if -g with no dwarf option is specified. |
| Keep symbols | -keep , instructs the assembler to keep local labels in the symbol table of the object file, for use by the debugger. |

Predefines

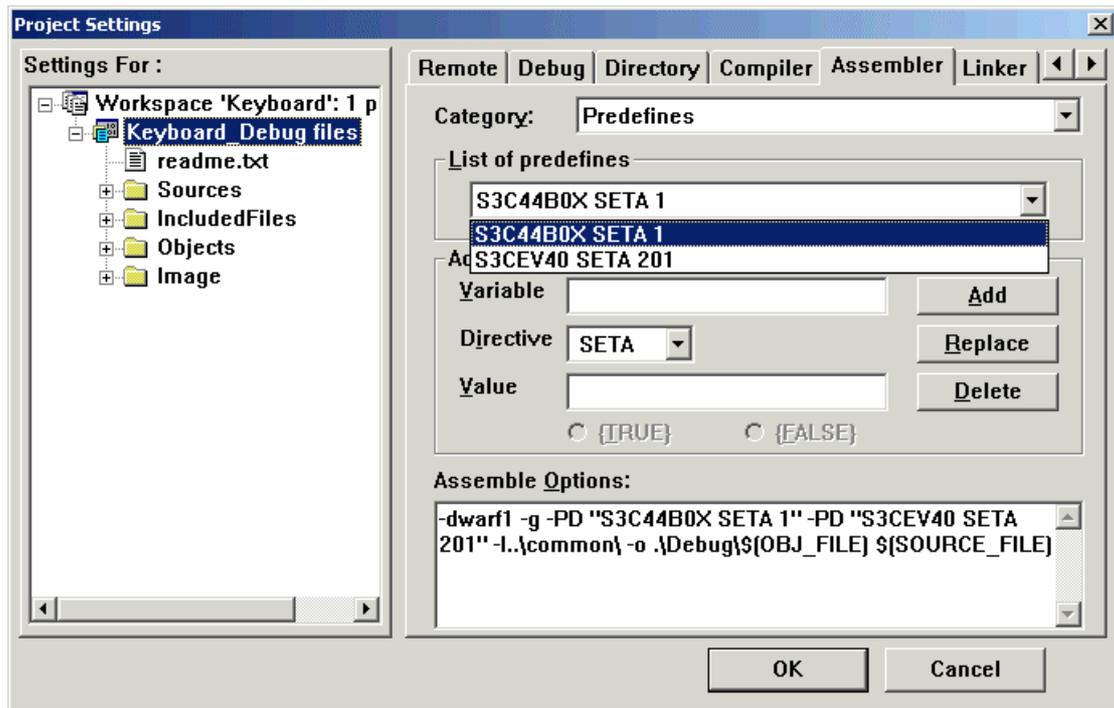


Figure 6-3-14 settings of Predefines

shows as Figure 6-3-9, In the Assembler dialog window, click the Category drawing menu, choose Predefines, the attributes of predefined macro, definition as following:

| Predefines Cluster | Description |
|----------------------------|--|
| Listing of predefines | The list of the active predefined |
| Variable, Directive, Value | <p>-PD, instructs the assembler to pre-execute one of the SET directives. You must enclose directive in double quotes. See:</p> <ul style="list-style-type: none"> • SETA directive on page 5-82. • SETL directive on page 5-83. • SETS directive on page 5-84. <p>The assembler executes a corresponding GBLL, GBLS, or GBLA directive to define the variable before setting its value. Arguments to SETS must be</p> |

enclosed in escaped double quotation marks,

for example:

```
-pd "Version SETS \"beta-4\""
```

```
-pd "VersionNum SETA 4"
```

Listing Options

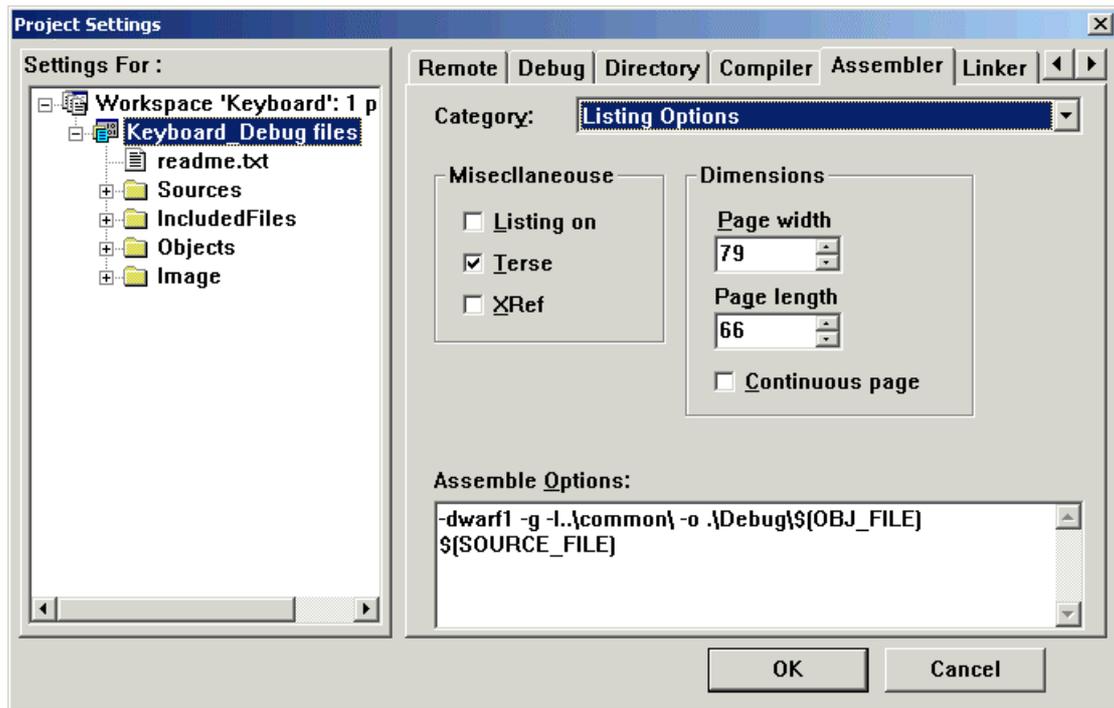


Figure 6-3-15 settings of 信息列表文件

shows as Figure 6-3-9, In the Assembler dialog window, click the Category drawing menu, choose Listing Options, the attributes of the assemble output files, definition as following:

| Listing Options | Cluster | Description |
|-----------------|---------|--|
| Listing on | | -liston , list generation is on. |
| Terse | | -noterse , turns the terse flag off. When this option is on, lines skipped due to conditional assembly do not appear in the listing. If the terse option is off, these lines do appear in the listing. The default is on. |
| XRef | | -xref , instructs the assembler to list cross-referencing information on symbols, including where they were defined and where they were used, both inside and outside macros. The default is off. |

| | |
|-----------------|---|
| Page width | -width , sets the listing page width. The default is 79 characters. |
| Page length | -length n(>0) , sets the listing page length. Length zero means an unpagged listing. The default is 66 lines. |
| Continuous page | This a synonym for -length 0 . |

Assemble Options Window

The Assembler property page, as in the figure 6-8, is used to configure the assembling options for assembler of ARM Build Tools cross-compiler, All the options user select are displayed in the Assemble Options edit box with the following format:

`[Opt-1] [Opt-2] ...-o[Path]$(OBJ_FILE) ... [Opt-n] ...$(SOURCE_FILE)`

Note: You can input or modify the options manually in the edit box, but the blank character between each option must be reserved, and the macros `$(SOURCE_FILE)`, `$(OBJ_FILE)` should not be deleted or modified. `$(SOURCE_FILE)` means the source file to be compiled, `$(OBJ_FILE)` means the output of the assembling. There will be replaced with the actual file name by the Embest IDE at the time of building.

shows as Figure 6-3-9, Assemble Options Window indicates:

| Switch | Description |
|-------------------------|--|
| -dwarf 1 | -dwarf1, Use DWARF1 debug table format. |
| -g | Include debug information. |
| -I..\common\ | Use the common folder which in the project last layer catalog as the head files search folder. |
| -o .\Debug\\$(OBJ_FILE) | Set the assemble target files output to Debug folder. |
| \$(SOURCE_FILE) | Assemble all the source files in the project. |

6.3.4 ARM Linker options setting

The linker attribute tab of the ARM Build Tools linker is illustrated in Fig. 6-3-16-16. If the various option configurations in the figure are used for the linker, all the settings of the user will be displayed in the Link Options edit box in the form of command line. When a new project selects the corresponding Build Tools, the target file output by the linker will be an execution file.

In the Linker attribute tab, change the pull-down window of the Category, and respectively set the various category options for the ARM Build Tools linker.

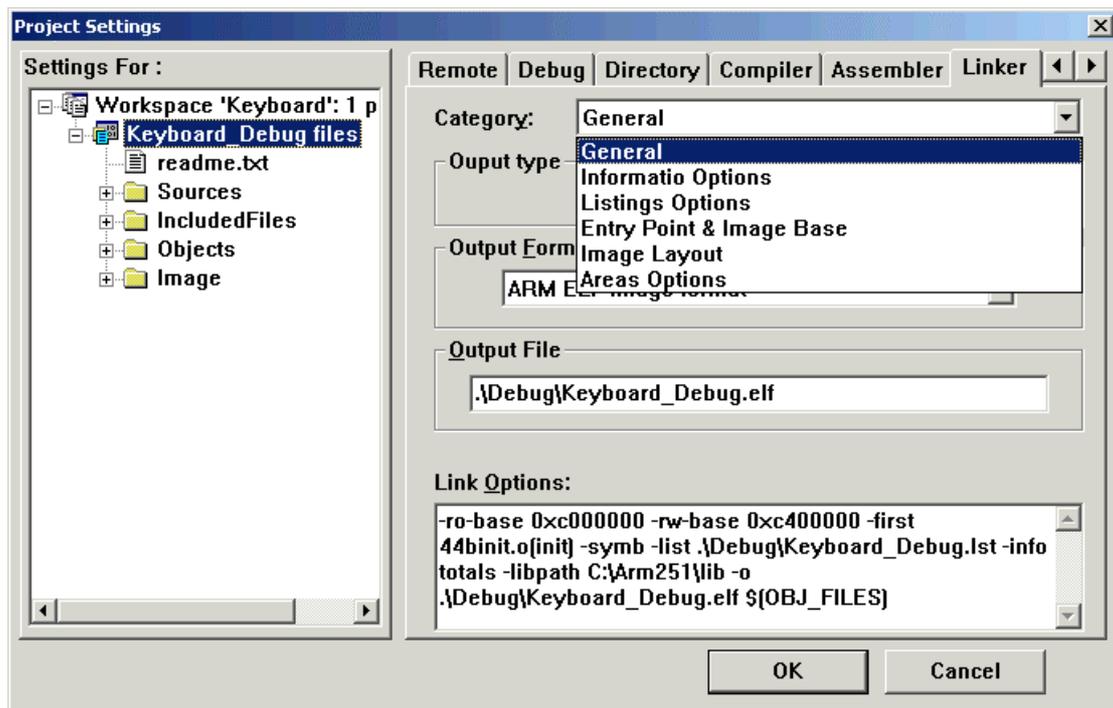


Figure 6-3-16 settings of General

General

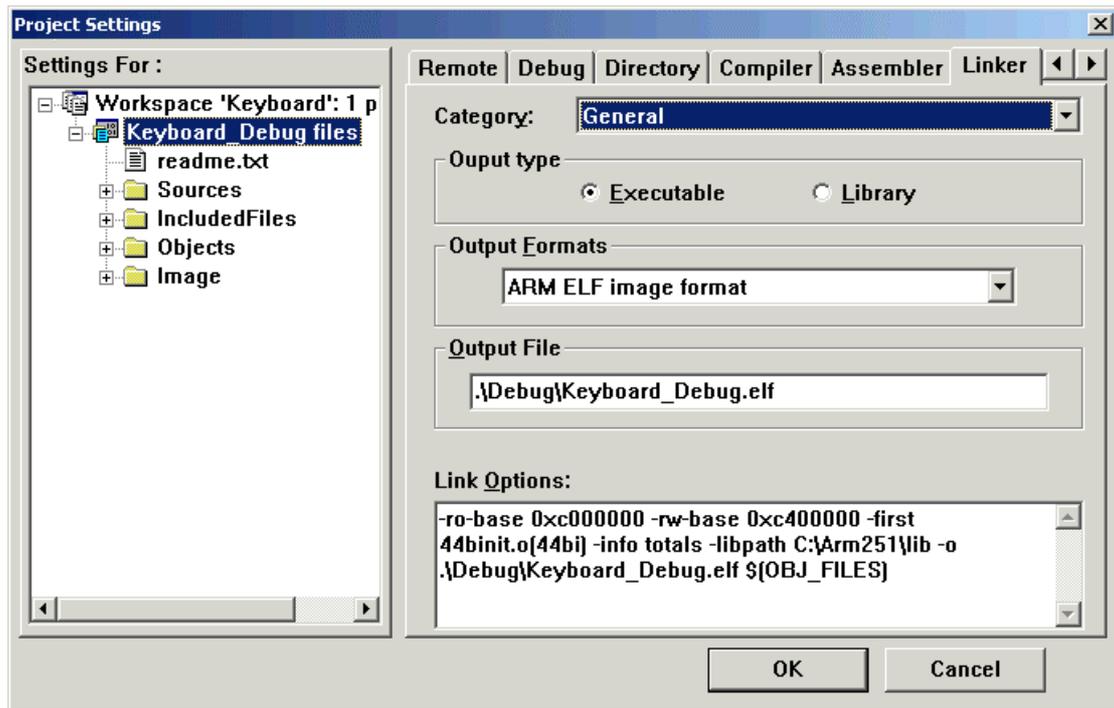


Figure 6-3-17 settings of General

shows as Figure 6-3-16, In the Linker dialog window, click the Category drawing menu, choose General, the attributes of ARM linker definition as following:

| General Cluster | Description |
|-----------------|---|
| Executable file | The output file is an executable file |
| Library | The output file is a library |
| Output formats | <p>select linker output file formation</p> <p>-elf generates the image in ELF format. This is the default. Future versions of the ARM linker will output images in ELF file format only. You can use the fromELF utility to convert an ELF file to another format.</p> <p>-aof generates the consolidated object in AOF. Because AOF can only be used to represent an object, this option is interpreted by the</p> |

linker as a request for partial linking of the input objects into a consolidated object.

-aif generates the image in executable AIF format. Because `-aif` will not be supported in future releases, you are recommended to use `-elf` to produce the output file, then run the `fromELF` utility to convert to AIF format.

-aif -bin generates the image in non-executable AIF format. Because `-aif -bin` will not be supported in future releases, you are recommended to use `-elf` to produce the output file, then run the `fromELF` utility to convert to AIF BIN.

-bin generates the image in plain binary format. Because `-bin` will not be supported in future releases, you are recommended to use `-elf` to produce the output file, then run the `fromELF` utility to convert to BIN.

Output file

-o, The file name of output file (with the project file name and the stuff `.elf` or `.lib/.alf`)

Information Options

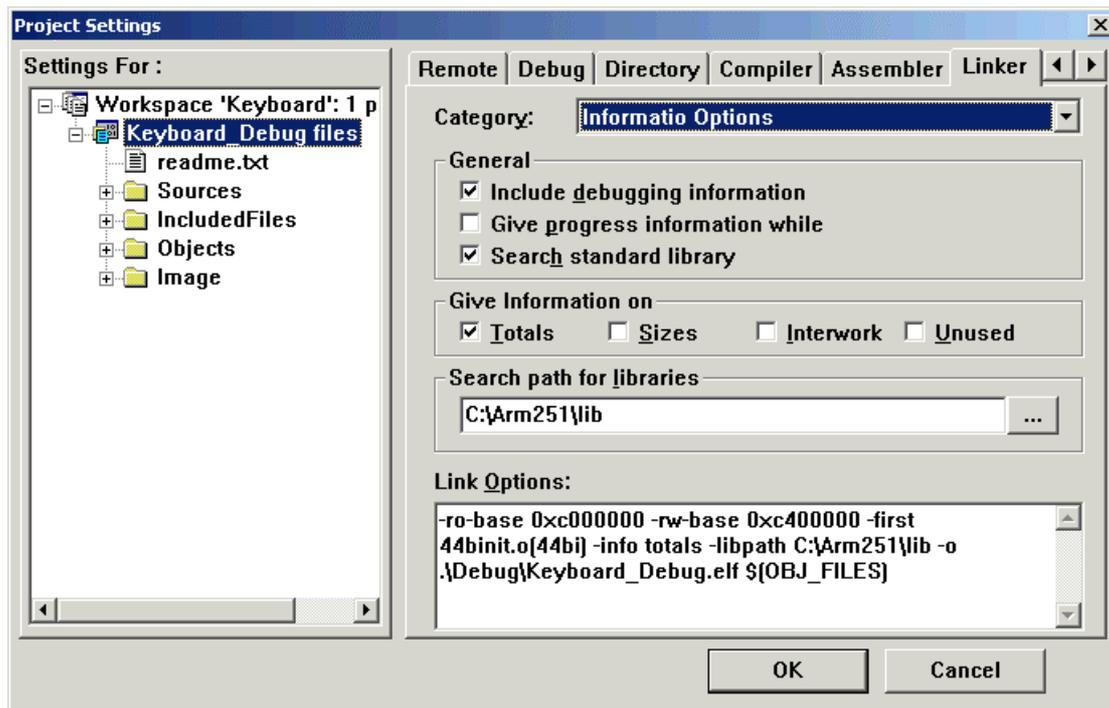


Figure 6-3-18 settings of Information Options

shows as Figure 6-3-16, In the Linker dialog window, click the Category drawing menu, choose Information Options, the definition of the ARM linker debug target files as following:

| Information Options Cluster | Description |
|---------------------------------|--|
| Include debugging information | -nodebug , turns off the inclusion of debug information in the output file. The image is then smaller, but you cannot debug it at source level. |
| Give progress information while | -verbose , prints messages indicating progress of the link operation. |
| Search standard library | -noscanlib , prevents the scanning of default libraries in a link step. This is the opposite of -scanlib . (See also -libpath above). |
| Give information on | -info < topic > , prints information about |

specified topics, where topic-list is a comma-separated list of topic keywords. A topic keyword may be one of the following:

Totals reports the total code and data sizes in the

image. The totals are broken down into separate totals for object and library files.

Sizes gives a detailed breakdown of the code and data sizes for each input object and interworking veneers. **Interwork** is ignored in this release of the linker.

Unused lists all unused areas, when used with the -remove option.

Note that spaces are not allowed between keywords in a list. For example, you can enter:

-info sizes,totals but not:

-info sizes, totals

Search path for libraries

-libpath, specifies a path that is used to search for libraries. This path overrides the path specified by the ARMLIB environment variable.

If you do not specify a path using -libpath, the linker searches in the path specified by ARMLIB, else searches the libraries defined in the file Lib\$\$Request\$\$library\$\$ variant.

Listings Options

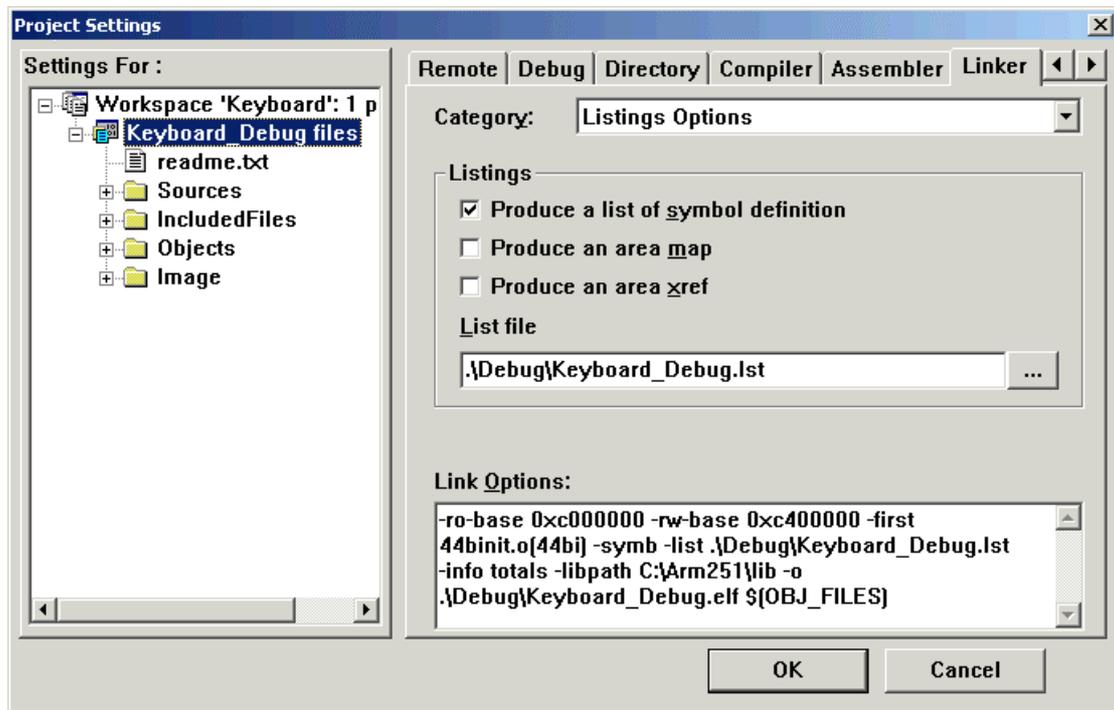


Figure 6-3-19 settings of Listings Options

shows as Figure 6-3-16, In the Linker dialog window, click the Category drawing menu, choose Listings Options, the setting of the listing file attributes , definition as following:

| Listings Options Cluster | Description |
|-------------------------------------|---|
| Produce a list of symbol definition | -symb , lists each symbol used in the link step (including linker-generated symbols) and its value, in the named file. A filename of minus (-) names the standard output stream instead of a file. |
| Produce an area map | -map , creates an image map listing the base and size of each constituent area. |
| Produce an area xref | -xref , lists cross-references between |

input areas.

List file

-list, redirects the standard output stream to file. This is useful in conjunction with **-map**, **-xref**, and **-symbols**.

Entry Point & Image Base

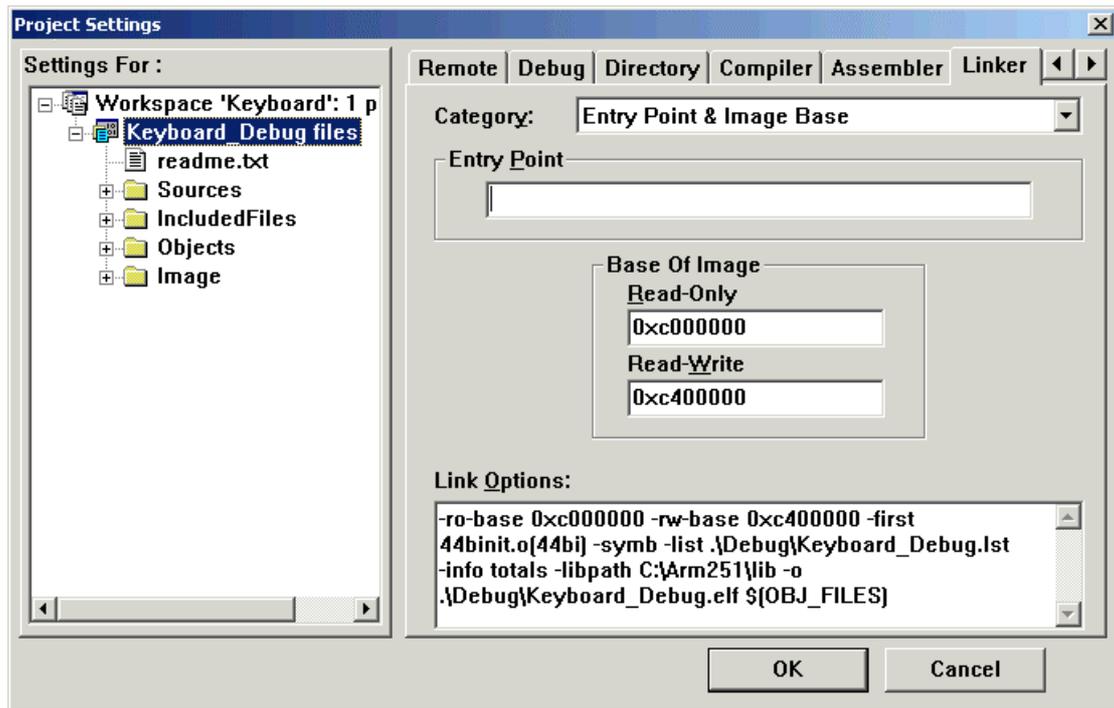


Figure 6-3-20 settings of Entry Point & Image Base

shows as Figure 6-3-16, In the Linker dialog window, click the Category drawing menu, choose Entry Point & Image Base, definition as following:

| Entry Point & Image Base Cluster | Description |
|----------------------------------|---|
| Entry Point | -entry , specifies the entry point of the image. The entrypoint may be given as either. |
| Base of image | <p>-ro-base, instructs the linker to place the Read-Only section at exec_address (for example, the address of the first location in ROM), set in Read-Only</p> <p>-rw-base, instructs the linker to place the Read-Write section at exec_address, set in Read-Write</p> |

Image Layout

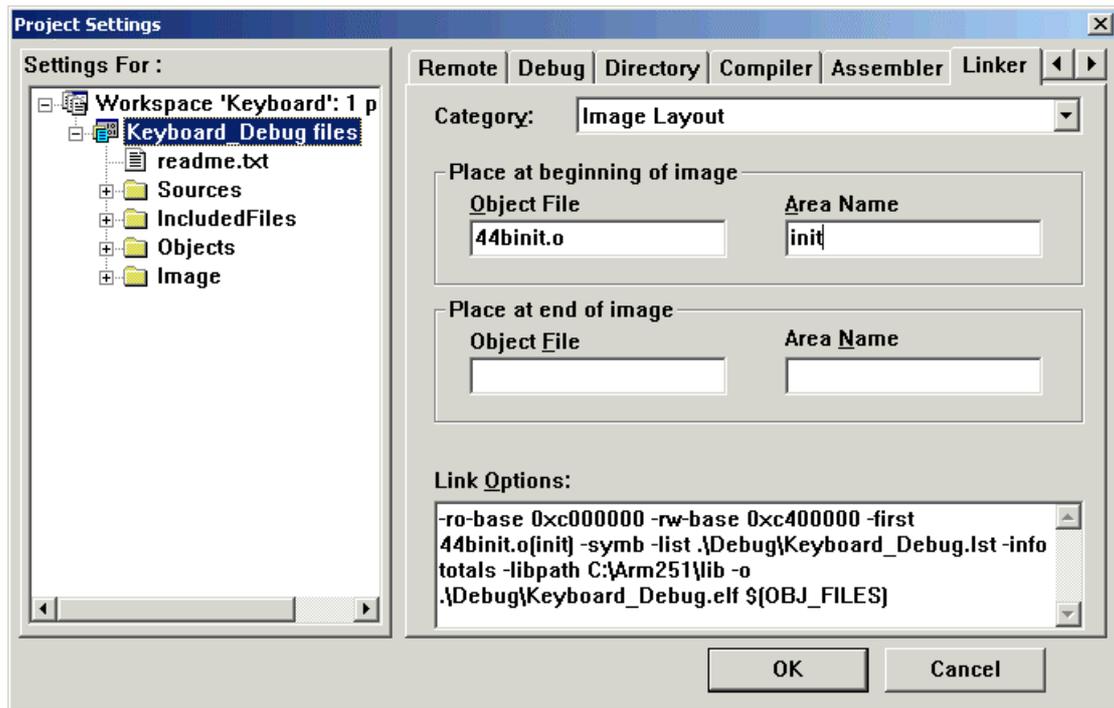


Figure 6-3-21 settings of Image Layout

shows as Figure 6-3-16, In the Linker dialog window, click the Category drawing menu, choose Image Layout, definition of the start image and end image in the ARM linker target files as following:

| Image Layout Cluster | Description |
|-----------------------------|---|
| Place at beginning of image | <p>-first, places area from object first in the RO section of the image if it is a non ZI area. If it is a ZI area, it is placed first in the ZI section. This can be used to force an area that maps low addresses to be placed first (typically the reset and interrupt vector addresses). There must be no space between object and the following open parenthesis.</p> <p>When using scatter loading, use +FIRST instead.</p> |

Place at end of image

Object file assemble object files

Area Name a certain Area name

-last, places area from object last in the RW or RO section of the image if it is a non-ZI area. If it is a ZI area, it is placed last in the ZI section. For example, this can be used to force an area that contains a checksum to be placed last in the RW section. There must be no space between object and the following open parenthesis.

When using scatter loading, use +LAST instead.

Object file assemble object files

Area Name a certain Area name

Note: Object file must include Area Name in it.

Areas Options

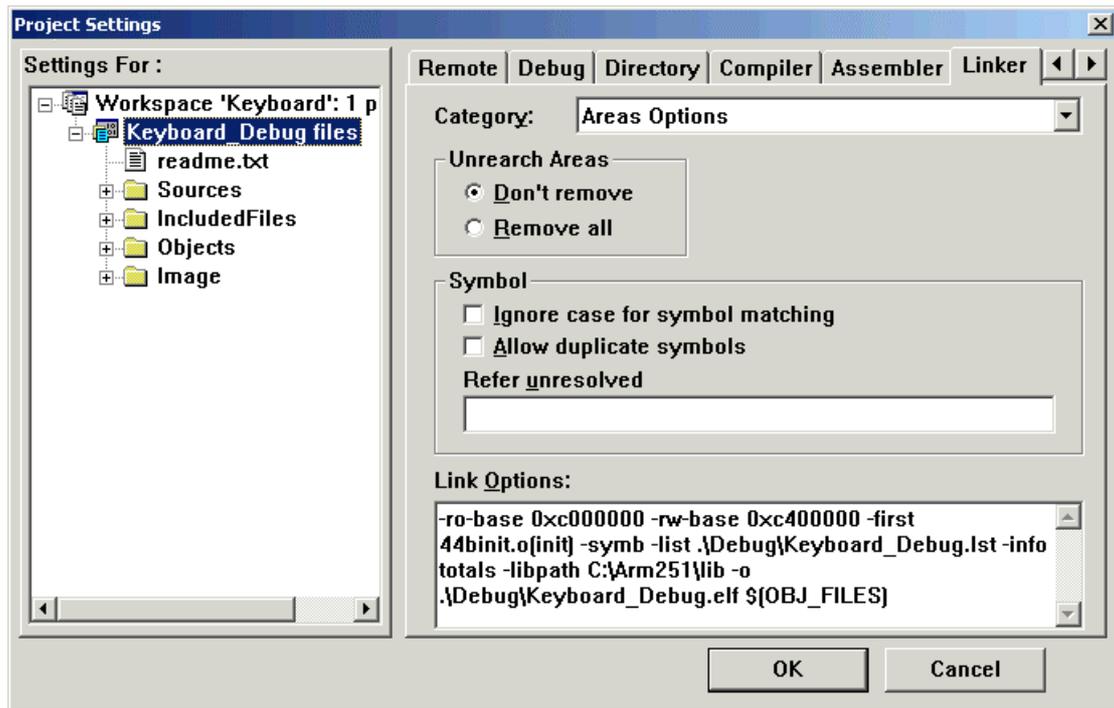


Figure 6-3-22 settings of Areas Options

shows as Figure 6-3-16, In the Linker dialog window, click the Category drawing menu, choose Areas Options, definition of the label segment in the ARM linker target files as following:

| Areas Options Cluster | Description |
|---------------------------------|--|
| Unreach Areas | <p>-remove, removes unused areas from the image. An area is considered to be used if it contains the image entry point, or if it is referred to from a used area. You must take care not to remove interrupt handlers when using -remove.</p> <p>Don't remove do not remove</p> <p>Remove all remove unused areas</p> |
| Ignore case for symbol matching | <p>-case, uses case-sensitive symbol name matching. This is the default.</p> |

Allow duplicate symbols

-dupok, allows duplicate symbols so that an area can be included more than once in the image. However, if **-noremove** is also specified, the image must not contain multiple copies of the area.

Refer unresolved

-unresolved, matches each reference to an undefined symbol to the global definition of symbol. Note that symbol must be both defined and global, otherwise it will appear in the list of undefined symbols, and the link step will fail. This option is particularly useful during top-down development, when it may be possible to test a partially-implemented system (where the lower levels of code are missing) by connecting each reference to a missing function to a dummy function that does nothing. This option does not display warnings.

Linker Options Window

The linker property page, as in the Figure 6-3-16, is used to configure the link options for linker of ARM Build Tools cross-compiler, All the options user select are displayed in the Link Options edit box with the following format as the output file is executable:

```
[Opt-1] ... -o[Path]$(TARGET_NAME) $(OBJ_FILES) [Lib-1] ...
```

The \$(TARGET_NAME) is a macro for executable file name, \$(OBJ_FILES) is also a macro for the collection of all object files to be linked.

As the target file is a library, the format of the options is:

```
[Opt-1] ... $(TARGET_NAME) $(OBJ_FILES) [Lib-1] ...
```

The \$(TARGET_NAME) is a macro for library name.

Note: You can input or modify the options manually in the edit box, but the blank character between each option must be reserved, and the macros \$(TARGET_NAME), \$(OBJ_FILES) should not be deleted or modified. There will be replaced with the actual file name by the Embest IDE at the time of building.

Note: \$(TARGET_NAME) will be replaced with the default that consists of project name and postfix, elf or lib.

Note: Macro \$(<entry.o>OBJ_FILES), for example, means that the file entry.o is the first object file in the collection of all object files to be linked.

shows as Figure 6-3-16, in the Assemble Options Window, the definition of the command as following:

| Switch | Description |
|------------------------|--|
| -ro-base 0xc000000 | set the address of the first location in ROM: 0xc000000 |
| -rw-base 0xc400000 | place the Read-Write section at exec_address: 0xc40000 |
| -first 44binit.o(init) | set init.o image file as the entrance of target files |

| | |
|--|---|
| -symb | output list symbol files to the Debug folder |
| -list .\Debug\Keyboard_Debug.lst | of the project, include label and across reference information. |
| -info totals | debug information in the debug symbol files |
| -libpath C:\Arm251\lib | use Lib which in the folder C:\Arm251\lib |
| -o .\Debug\Keyboard_Debug.elf \$(OBJ_FILES) | Linker output the debug file which name as the project name to the debug folder |

6.4 Project Settings & Folder Settings

More than one source file is included in a single project at the most time, and each file maybe has its own compile options different from the others. Thus, we used to add the files that have the same compile options into a folder of the project, and set options for the folder instead of each source file.

Show as Figure 6-2-1, GNU Tools for ARM, project led_swing has three folders --- C Source, ASM Source and Link Script. The files in folder C Source have their own compile options different from the project settings.

There is the same settings while selecting the ARM Build Tools which SDT or ADS project folds.

Note: For example, a project for ARM based application includes a list source files, part of them should be compiled to generate thumb object files, and the others should result with arm object files. Here, the source files can be divided into two parts, and added to two folders separately with relevant option settings.

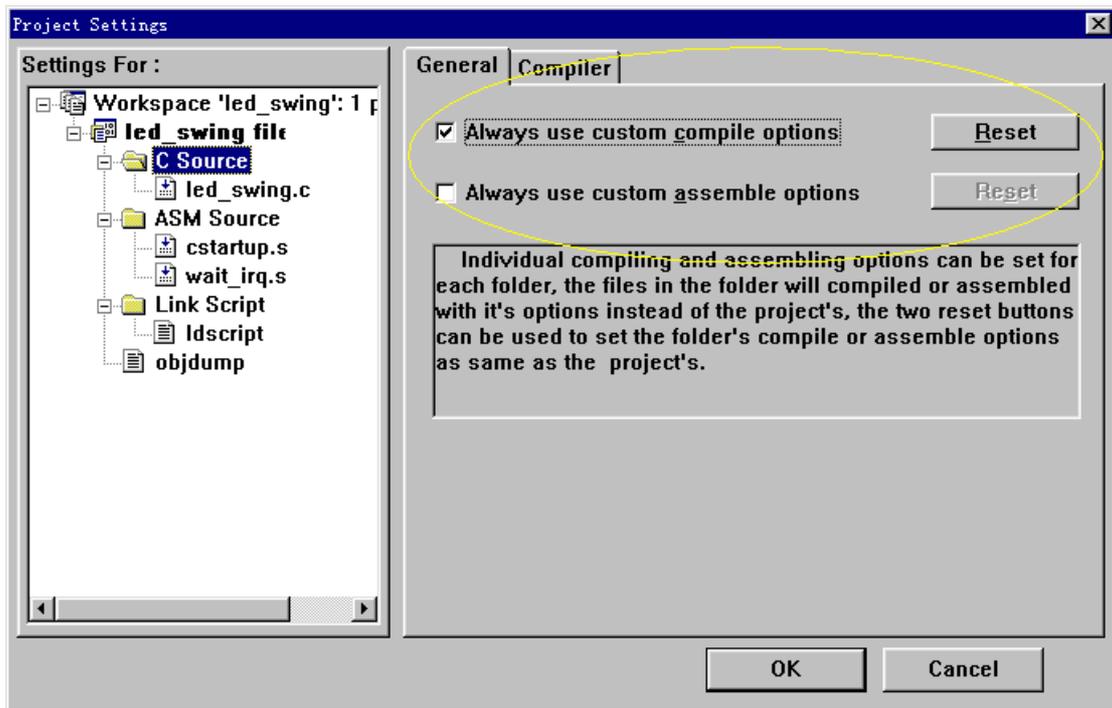


Figure 6-4-1 Folder Settings Dialog

Show as Figure 6-4-1, select Always Use Custom Compile Options check box, the relevant Compiler property page will displayed, in this property page,

you can set the compile options for the folder. Otherwise, the folder has the same compile settings with the project settings.

If select Always Use Custom assemble Options check box, the relevant assembler property page will displayed, in this property page, you can set the assemble options for the folder. Otherwise, the folder has the same assemble settings with the project.

Click Reset buttons, valid when the matched check box is selected, to reshuffle the relevant settings to be as same as the project.

6.5 Project Building

Do one of the following to build project:

1. Click on Build button on Build toolbar.
2. Click on Build item on Project menu.
3. Click on Rebuild All item on Project menu to rebuild the project.

If build succeeds, a target file will be generated in the output directory specified in the Project Settings dialog.

The output information is displayed in the following [build pane](#). If any error occurs, the building operation will be terminated, and the error(s) will be displayed in the [build pane](#).

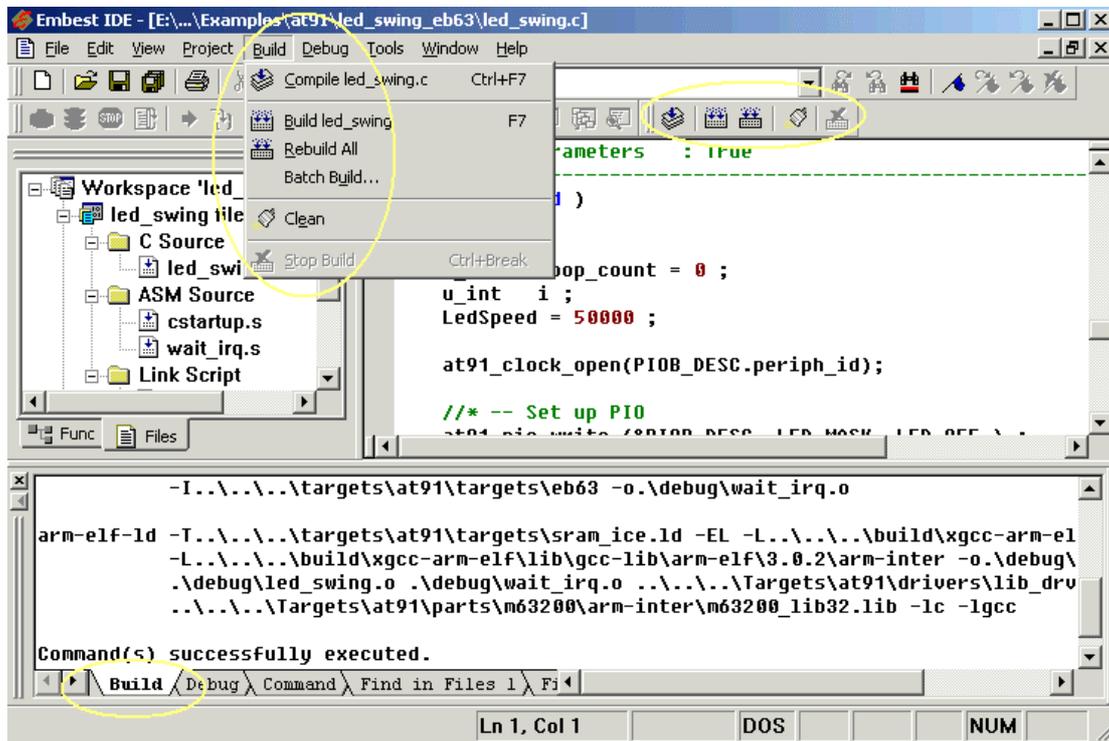


Figure 6-5-1 Build Menu and Toolbar

6.5.1 Project File Compiling and Assembling

Do one of the following to compile or assemble file:

1. Click on Compile button on Build toolbar.
2. Click on Compile item on Project menu.
3. Click on Compile item on Workspace popup menu (show as Figure 6-18).

The compiler can handle what kind of file or assembler that is described in File type chapter.

| Menu Item | Description |
|--|---|
|  Compile | <p>Before compile or assemble the active document, EmbestIDE will check several dependence relationship, if the following assumptions are true, the active document will not be handled:</p> <ol style="list-style-type: none">1、 the object file are more up-to-the-minute than the source file;2、 the object file are more up-to-the-minute than the all dependence files of the source file;3、 No compiling or assembling options are modified, since the last building or rebuilding all operation. |

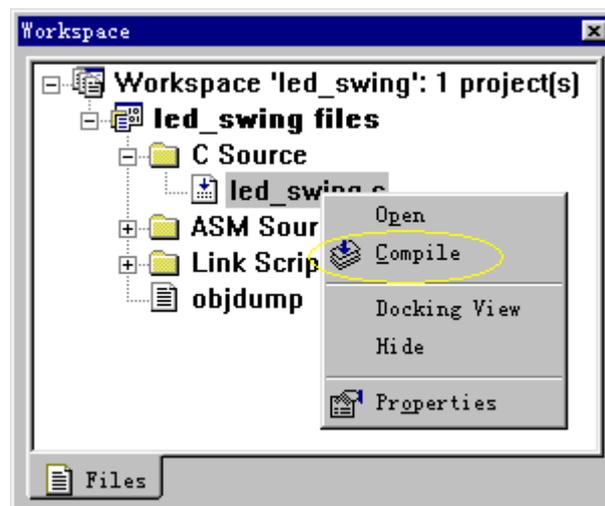


Figure 6-5-2 Compiling Menu Item

6.5.2 Project Build

The menu items----Build, Rebuild All, Batch build, Clean and Stop Build----are all used to handle the active project:

| Menu Item | Description |
|---|--|
|  Build | <p>Compile all the source files that need to be compiled (as the description in Files type chapter), and link the object files to generate the target file.</p> <p>Before build the active project, EmbestIDE will check several dependence relationship, if the following assumptions are true, the active project will not be handled:</p> <ol style="list-style-type: none">1、 No source file need to be compiled;2、 The target file are more up-to-the-minute than the dependence files of linking operation;3、 No linking options are modified, since the last building or rebuilding all operation. |
|  Rebuild All | <p>EmbestIDE deletes the existing object files and target file first and then generates them again, suggest use this operation if there more than one project in current workspace.</p> |
| Batch Build | <p>Batch Build the Projects and order the building sequence in current workspace.</p> |
|  Clean | <p>Delete all the intermediate files include the object files and target files</p> |
|  Stop Build | <p>Stop the building or rebuilding all operation</p> |

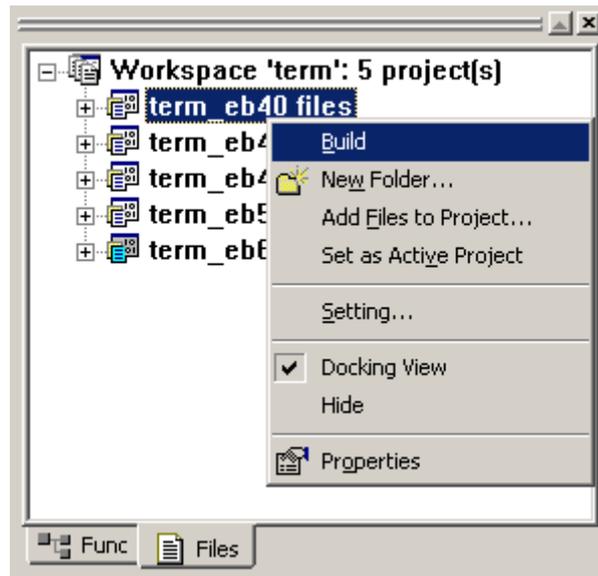


Figure 6-5-3 Build Pop Menu in Workspace Pane

6.5.3 Projects batch build

The user can simultaneously carry out batch build upon several projects in current workspace. Click the menu Build > Batch Build to pop out the dialogue box as shown in fig. 6-5-4.

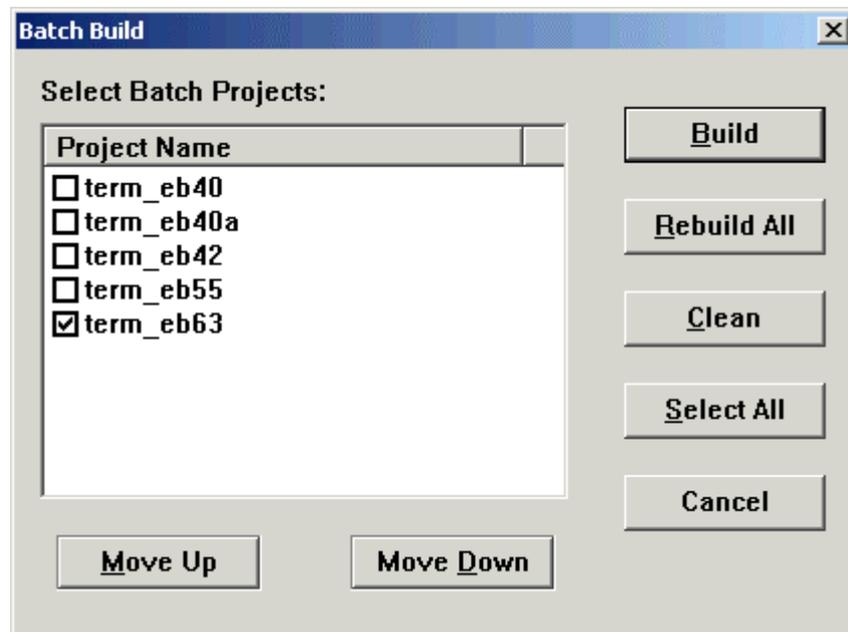


Fig 6-5-4 Batch Build operating window

Introduction to options of Batch Build operating window:

Project Name: select the project to be built by the user; the project not be selected will not be built.

Move Up: move the highlighted project up a step.

Move Down: move the highlighted project down a step.

Select All: select all projects in workspace.

Build: Build the selected projects in the workspace. The regulated rule of building is same as the definition of Build menu of the system.

Rebuild All: Re-build the selected projects in the workspace. The regulated rule of building is same as the definition of Build menu of the system.

Clean: completely delete the files produced in building.

Note: for several projects, if it is set that the file folder with same name

store building output result, it is suggested using the operation of Rebuild All so that it may not occur any error while building and connecting files with same name.

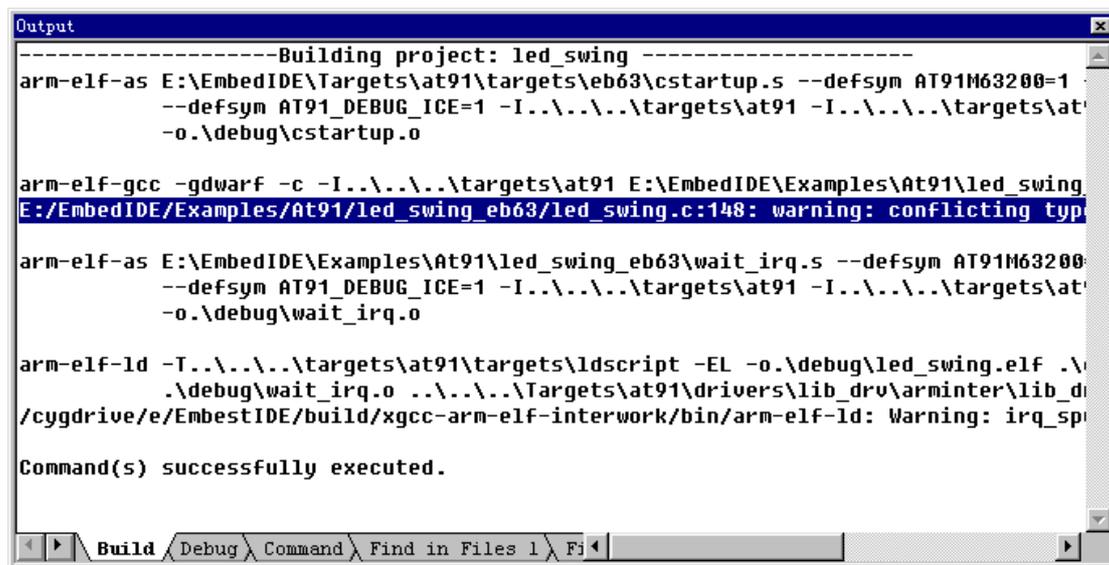
6.6 Building Information in Output window

At the beginning of compiling or building operation, the build pane will be set active to display the output information. In the Figure 6-21, the first line in this window prompt you which project or file is handled now.

If all the command execute successfully, EmbestIDE will print line ---Command(s) successfully executed, otherwise, line---Error executing above command.

If fail to complete the compiling operation, error will displayed in build pane. To locate to the corresponding source quickly for those syntax errors, simply double click on the line or press key F4 (shift + F4).

Building output information shows as Figure 6-5-5 to Figure 6-5-8:



```
Output
-----Building project: led_swing -----
arm-elf-as E:\EmbedIDE\Targets\at91\targets\eb63\cstartup.s --defsym AT91M63200=1
--defsym AT91_DEBUG_ICE=1 -I... \targets\at91 -I... \targets\at91
-o.\debug\cstartup.o

arm-elf-gcc -gdwarf -c -I... \targets\at91 E:\EmbedIDE\Examples\At91\led_swing
E:/EmbedIDE/Examples/At91/led_swing_eb63/led_swing.c:148: warning: conflicting typ

arm-elf-as E:\EmbedIDE\Examples\At91\led_swing_eb63\wait_irq.s --defsym AT91M63200
--defsym AT91_DEBUG_ICE=1 -I... \targets\at91 -I... \targets\at91
-o.\debug\wait_irq.o

arm-elf-ld -T... \targets\at91\targets\ldscript -EL -o.\debug\led_swing.elf .\
.\debug\wait_irq.o ... \Targets\at91\drivers\lib_drv\arminter\lib_d
/cygdrive/e/EmbestIDE/build/xgcc-arm-elf-interwork/bin/arm-elf-ld: Warning: irq_sp

Command(s) successfully executed.
```

Figure 6-5-5 Build Pane

```

Output
-----Building project: Keyboard_Debug -----
armasm -dwarf1 -g -o .\Debug\44binit.o D:\EmbestIDE000\myExamples\myExp\SDTprj\S3CEU40\
armcc -O0 -dwarf1 -g+ -o .\Debug\44BLIB.o -c D:\EmbestIDE000\myExamples\myExp\SDTprj\S3
armcc -O0 -dwarf1 -g+ -o .\Debug\Eint.o -c D:\EmbestIDE000\myExamples\myExp\SDTprj\S3CE
armcc -O0 -dwarf1 -g+ -o .\Debug\Etc.o -c D:\EmbestIDE000\myExamples\myExp\SDTprj\S3CEU
armcc -O0 -dwarf1 -g+ -o .\Debug\main.o -c D:\EmbestIDE000\myExamples\myExp\SDTprj\S3CE

armlink -ro-base 0xc000000 -rw-base 0xc400000 -first 44binit.o(init) -info totals -libp
-o .\Debug\Keyboard_Debug.elf .\Debug\44binit.o .\Debug\44BLIB.o .\Debug\Eint.o
      code  inline  inline  'const'  RW  0-Init  debug
      size  data  strings  data    data    data    data
Object totals  5432   412    708     0     36     0   14776
Library totals 20220   312    440    148    644    80   1216
Grand totals  25652   724   1148    148    680    80  15992

Debug Area Optimization Statistics

Input debug total(excluding low level debug areas) 16304 (15.92Kb)
Output debug total                               15992 (15.62Kb)
% reduction                                       1.91%

Command(s) successfully executed.
Build Debug Command Find in Files 1 Fi

```

(a) not use list file output

```

Output
-----Building project: Keyboard_Debug -----
armasm -dwarf1 -g -I..\common\ -o .\Debug\44binit.o D:\EmbestIDE000\myExamples\
armcc -O0 -dwarf1 -g+ -I..\common\ -o .\Debug\44BLIB.o -c D:\EmbestIDE000\myExa
armcc -O0 -dwarf1 -g+ -I..\common\ -o .\Debug\Eint.o -c D:\EmbestIDE000\myExamp
armcc -O0 -dwarf1 -g+ -I..\common\ -o .\Debug\Etc.o -c D:\EmbestIDE000\myExamp
armcc -O0 -dwarf1 -g+ -I..\common\ -o .\Debug\main.o -c D:\EmbestIDE000\myExamp

armlink -ro-base 0xc000000 -rw-base 0xc400000 -first 44binit.o(init) -symb -list
totals -libpath C:\Arm251\lib -o .\Debug\Keyboard_Debug.elf .\Debug\44b
.\Debug\Etc.o .\Debug\main.o

Command(s) successfully executed.
Build Debug Command Find in Files 1 Fi

```

(b) use list file output

Figure 6-5-6 GNU Tools for ARM (success)

```
-----Compiling file: D:\EmbestIDE000\myExamples\myExp\test\maintest.c
arm-elf-gcc -gdwarf -c D:\EmbestIDE000\myExamples\myExp\test\maintest.c -c
D:/EmbestIDE000/myExamples/myExp/test/maintest.c: In function `Main':
D:/EmbestIDE000/myExamples/myExp/test/maintest.c:59: parse error before ">"
Error executing above command.
```

Figure 6-5-7 GNU Tools for ARM (failure)

```
-----Compiling file: D:\EmbestIDE000\myExamples\myExp\SDTprj\S3CEU40\common\44BLIB.C -----
armcc -O0 -dwarf1 -g+ -I..\common\ -o .\Debug\44BLIB.o -c D:\EmbestIDE000\myExamples\myExp\SDTprj\S3C
"D:\EmbestIDE000\myExamples\myExp\SDTprj\S3CEU40\common\44BLIB.C", line 33: Error: expected ';' or ','
D:\EmbestIDE000\myExamples\myExp\SDTprj\S3CEU40\common\44BLIB.C: 0 warnings, 1 error, 0 serious error
Error executing above command.
```

Figure 6-5-8 ARM Build Tools (failure)

7. Program Debugging

7.1 Overview

The Embest IDE debugger combines the best features of graphical debug and command-line debug, provides multifarious debug ways.

The most common debugging activities, such as setting breakpoints and controlling program execution, are available through convenient point-and-click interfaces. Similarly, program listings and data-inspection windows provide an immediate visual context for the crucial portions of your application. For complex or unpredictable debugging needs, the command-line interface gives you full access to a wealth of specialized debugging commands.

Embest IDE provides much advanced features as follows:

- Supports assemble language debugging and source code debugging; supports many program windows (include source program window, disassemble program window and mix-mode window).
- Many emulational debug methods: Go, Reset, Stop, Step, Step into, Step over, Step out, Goto Cursor, Goto Source and Goto Address etc.
- Supports Unconditional Breakpoint, Conditional Breakpoint and Watchpoint.
- Register value display and modification.
- Memory content according can be displayed with byte, half-word or word length and Hexdecimal or Ascii mode.
- Supports global and local variables display and modification, and also supports expression value compute.
- When value-change occurs in memory, variables, registers , corresponding interface content will be displayed with red color.
- Supports function stack display.
- Saves debugging environment information with each project.

7.2 Debugger GUI

Figure 7-1 illustrates the GUI elements you can use to interact with Embest IDE debugger.

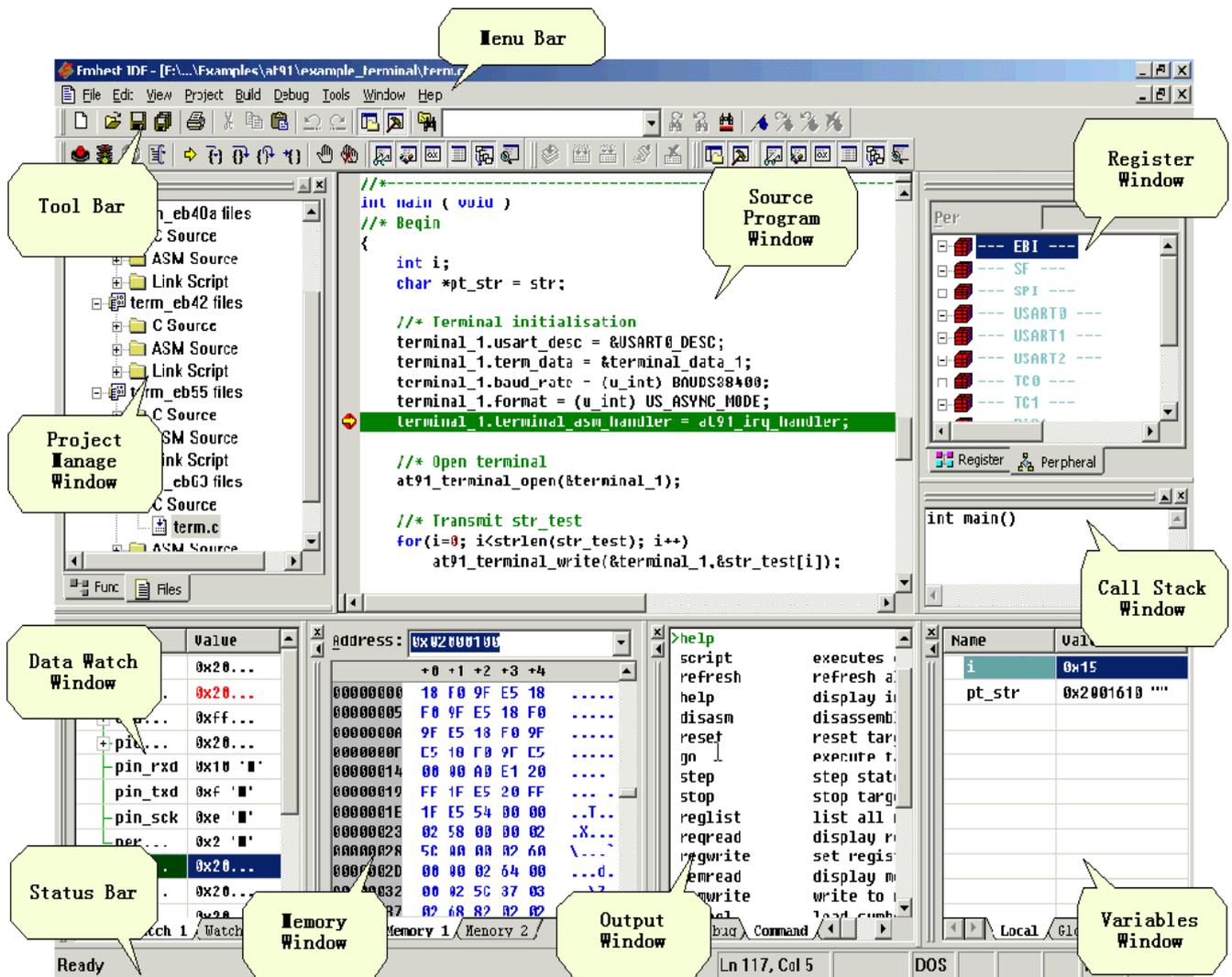


Figure 7-1 Embest IDE debugger interface

The Debug menu provides a complete list of Embest IDE GUI debugger commands, as well as their keyboard shortcuts.

The Debug toolbar provides buttons for the most common debugger commands, as well as for opening and closing all kind of windows and program compile and link etc.

The Debug status bar displays system status and detail explanations of menu's and toolbar's role.

7.2.1 Debugger Toolbar, Buttons, Menu and Shortcut

The debug toolbar has many buttons for the most common debugging commands, as well as display auxiliary debugger windows. The toolbar shown as a floating palette interface as figure 7-2.



Figure7-2 debugger toolbar

The commands in the debug menu include alternatives to the buttons in the debug toolbar, as well as additional debugger functions. Keyboard shortcuts are also available for all graphical debugger commands.

The debugger buttons and menu commands are described in following table 7-1.

Table 7-1 debug button and menu command

| Button | Menu Command | Description |
|---|------------------------|--|
| | Connect/ Disconnect | Connect or Disconnect target system |
| | Download | Download debug file to target system |
|  | Restart | Restart the program from entry point |
|  | Go | Run debug file on the target under debugger control |
|  | Stop | Stop target system |
|  | Reset | Reset target system |
|  | Step into | Step to the next line of code, in order of execution (not necessarily the next line displayed in the editor) |
|  | Step over | Step to the next line displayed on the screen. If there is a subroutine call on the current line, the |

| | | | |
|---|---------------------|-----|---|
| | | | button executes that subroutine in its entirety, then stops at the line after the subroutine call |
|  | Step out | | Finish the current subroutine. Execution continues until the current subroutine returns to its caller. |
|  | Run to Cursor | | Run to the line where cursor staying |
|  | Show Next Statement | | Show next code line which will be execute |
|  | Toggle Breakpoint | | Set or remove a task-level breakpoint on the current line of the editor window |
| | Enable Breakpoints | All | Enable all breakpoints |
| | Disable Breakpoints | All | Disable all breakpoints |
|  | Delete Breakpoints | All | Delete all breakpoints |
| | Breakpoints... | | Show the breakpoints management dialog |
|  | Watch | | Open or close the Watch window, which displays the values of specified variables throughout the execution of the program |
|  | Variables | | Open or close the Variables window, which displays the values of local and global variables |
|  | Registers | | Open or close the Registers window, which displays values of the target registers |
|  | Memory | | Open or close the Memory window, which displays target memory information |
|  | Call Stack | | Open or close the Call Trace window, which displays stack information |



Disassembly

Open or close the **Disassembly** window, which displays disassemble code

7.3 Debug Setting

Debug setting window, which is used to configure debug software, locates at the setting dialog window of project. The configurations is divided to three categories:

- **General**
- **Download**
- **Memory Maps**

General debug setting page interface show as following figure7-3:

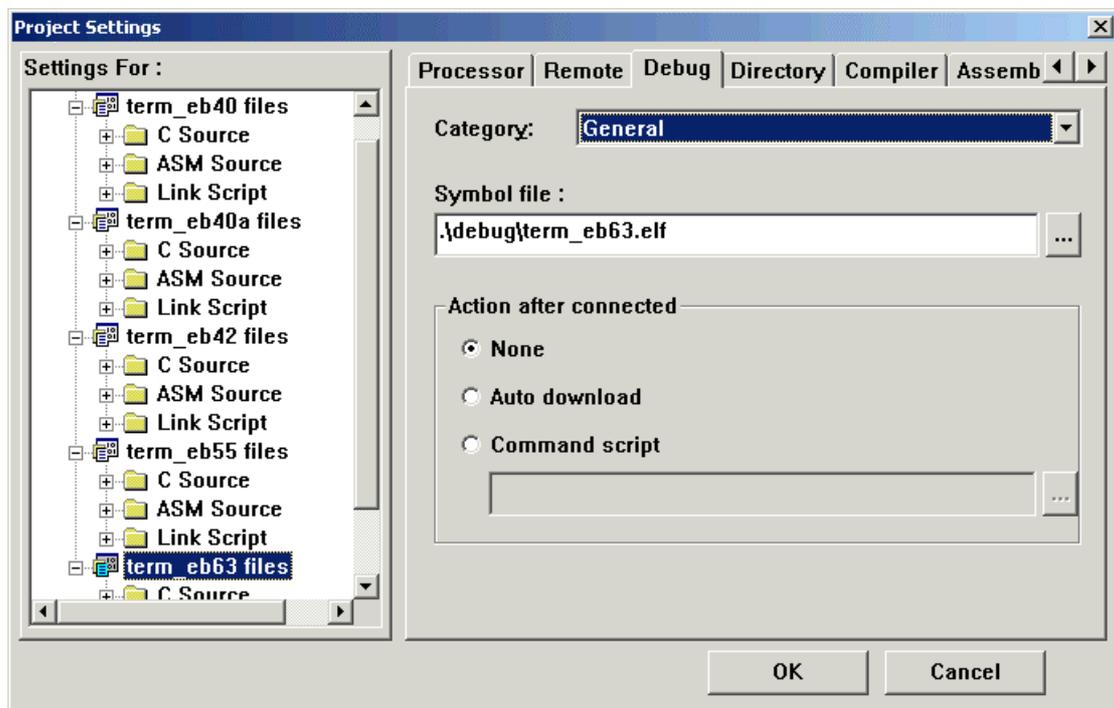


Figure7-3 General Debug Setting Page

Symbol file column specifies the debug symbol file name and directory, debug symbol file contains debug information for debugger, usually symbol file has Elf-format or Coff-format.

Auto Download option item is used for whether or not auto download file after the target system is resetted or debugger connects target. If be selected, debugger will finish the download operation automaticly.

Command Script option item specifies the command script file, if selected, the debugger will auto execute the commands listed in this file after system connects target board.

Download debug setting page interface show as following figure7-4:

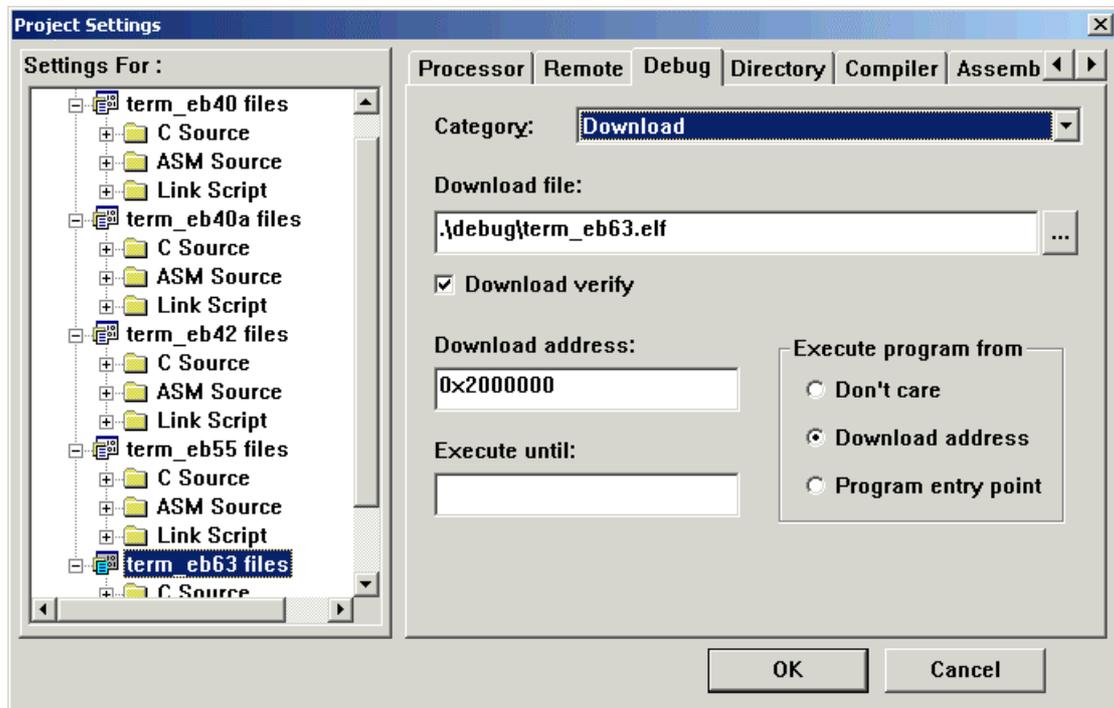


Figure7-4 Download Setting Page

Download file column specifies the executable file to be downloaded, this file is the program which will run on a target system.

Download Verift option item is used for whether or not auto checksum download file. If selected, debugger will auto compare the target memory file with the download file.

Download address column specifies the start memory address of download file, the download file will ordinarily be stored from this address.

Execute until column specifies a symbol to which program will run, after it is downloaded.

Download address option item means debugger will auto set PC's value with the download start address after file downloaded.

Program entry point option item means debugger will auto set PC's value with the entry-point address of the executable file after it is downloaded

Memory maps setting page interface show as following figure7-5:

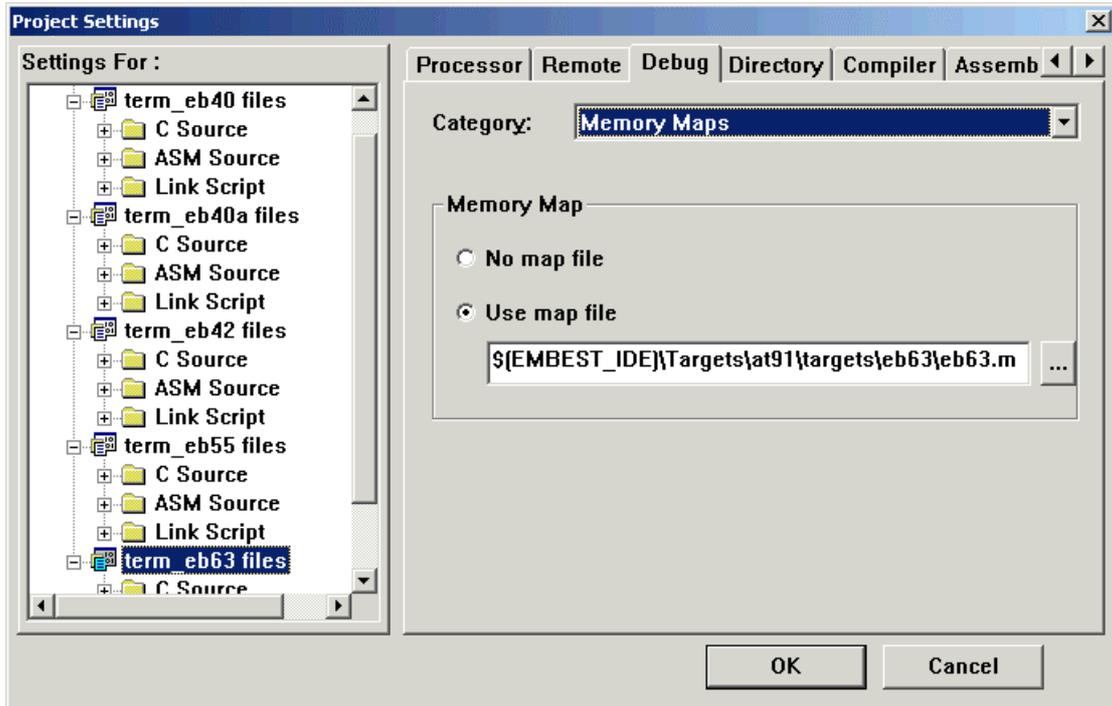


Figure7-5 Memory Maps Setting Page

Use map file option item means the range of memory access should be specified when users debug program, and the range is described in the memory map file assigned in the following editbox.

7.4 Start to Debug the program

When a project is compiled and linked successfully, and you have correctly filled debug setting dialog, you can debug the project now with the following steps --- 1, connect debug emulator device; 2, download program.

7.4.1 Connect Emulator

Before connect emulator device, please read [Appendix A JTAG Emulator Connect](#).

Connect computer's parallel port and the Embest Emulator's DB25 interface through standard DB25 male-to-female parallel cable. Embest Emulator connect target board through a header which mates which IDC sockets mounted on a straight through ribbon cable. And then, with the target board powered, hardware connection is established.

Click Debug menu, select 'Remote Connect' menu item(show as figure7-6), or push F8 key, debugger will connects target system through emulator device.



Figure7-6 Debug Menu before Connection

If connection is failed to set up, debugger will show possible reason in debug pane of the Output Window. Please refer to [Appendix A](#), and check whether power and cable connection are correct or not. If the connection established successfully, Debug menu will show as following figure7-7:



Figure7-7 Debug Menu after Remote Connect

7.4.2 Program download

After connection between host and target system set up, we can download executable file to target system now. If Auto Download option is set in Debug Setting Dialog, debugger will auto progress this step. If not, please click Debug menu, and select Download item. Target file will be downloaded to the predeterminate address on target system. The rate of download process will displayed on the status bar, show as following figure 7-8:



Figure 7-8 Status Bar as Program Downloading

If download succeed, status bar displays "Download Completed" in blue, otherwise, "Download Failed" in red. Show as following Figure 7-9 and Figure 7-10.



Figure 7-9 Status Bar as Program Download completed



Figure 7-10 Status Bar as Program Download failed

7.5 Control Program Executing

Debugger can control target program as it execute, and disassemble binary code in the target system, and also can control target program by set breakpoint to help user faster debugging program.

7.5.1 Program Running

Executing program state includes run-state、stop-state、reset-state mainly. Run-state expresses program is executing according to code order; Stop-state expresses it is stopped at certain code and waiting for debugger to read needed information; Reset-state expresses target system is staying at system entry point, all system information keeps at the initially state.

Click 'Debug' menu and select 'Go' menu item, or push 'F5' key, or click 'Go' button on debug toolbar, program will run from the stopped position, and mouse shape will change to funnelform shape, program running interface is shown as following figure7-11:

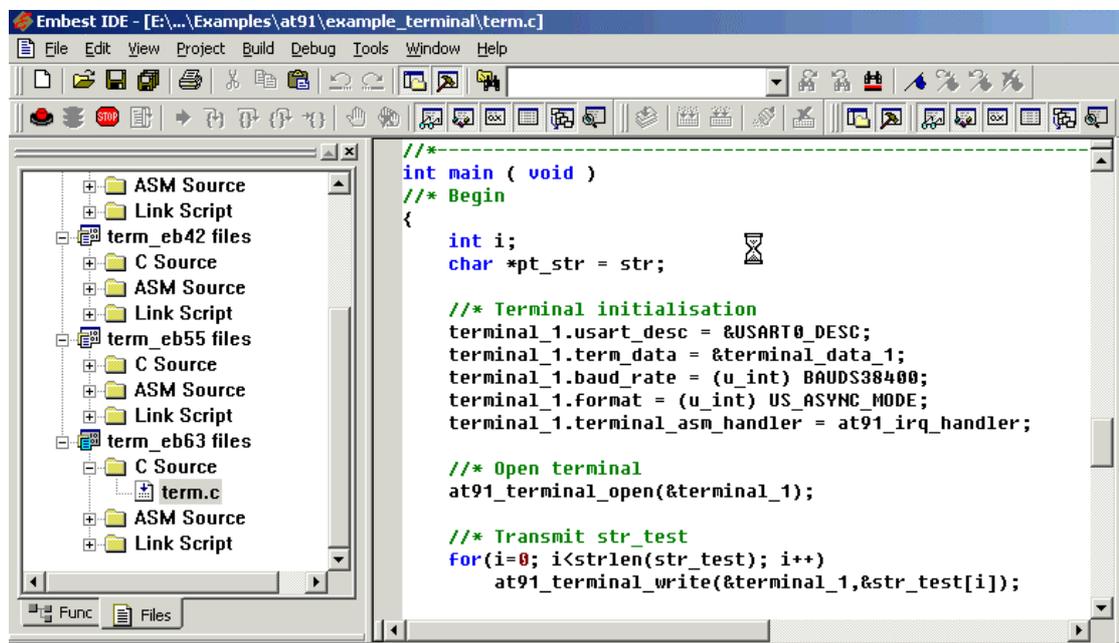


Figure 7-11 Program Running Interface

Click 'Debug' menu and select 'Stop' menu item, or push Shift+F5 key, or click 'Stop' button on debug toolbar, can make program stop, and mouse shape will change to primary shape, program stop interface is shown as following Figure7-12:

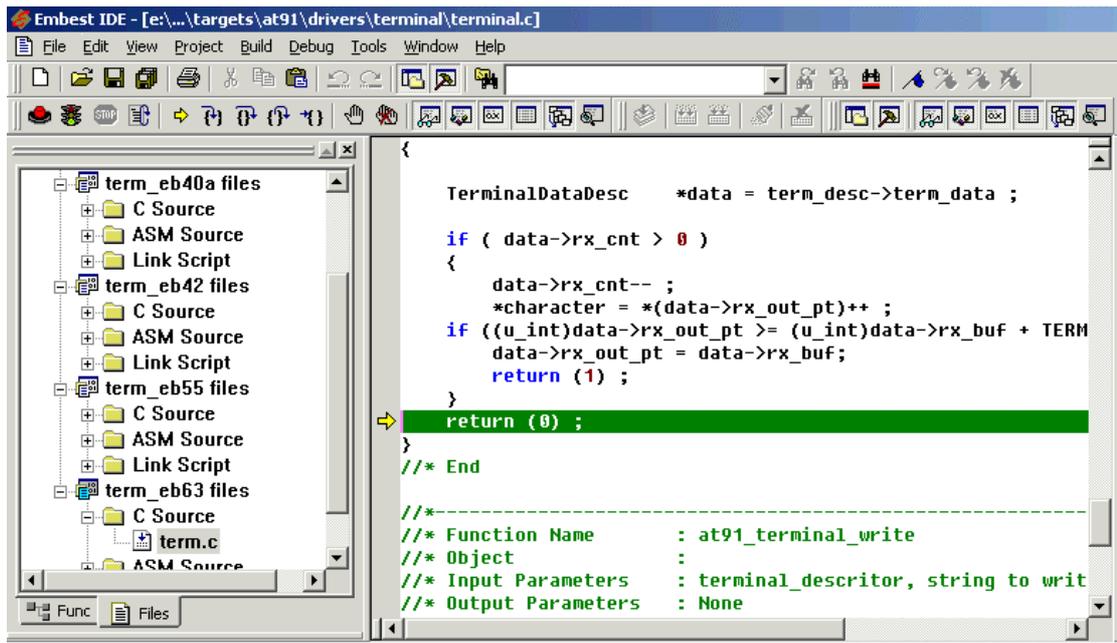


Figure 7-12 Program Stop Interface

When program stopped, if stop at certain source code, corresponding source code line will be highlight, and set a current-line flag in front of the line (current-line flag is yellow rightward arrow), source code interface is shown as following figure7-13:

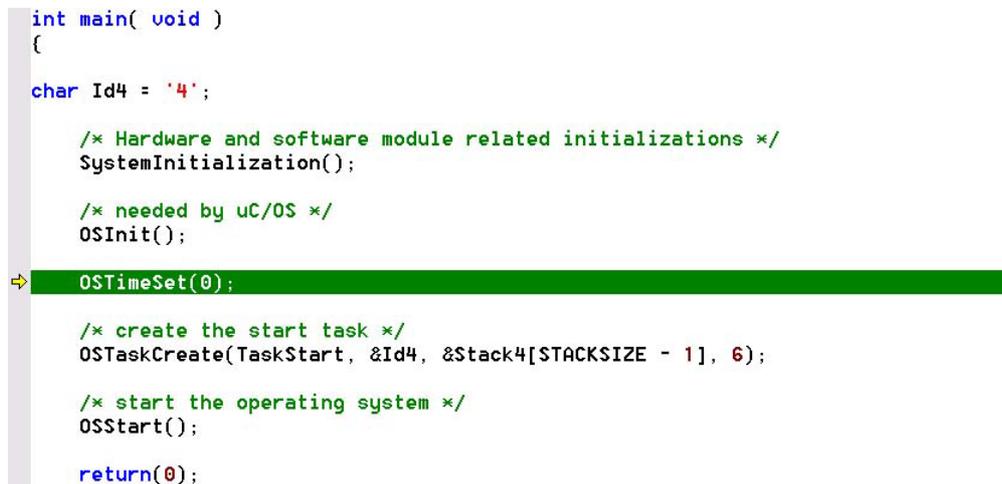


Figure 7-13 Program Stop Interface with Source Code

Click 'Debug' menu and select 'Reset' menu item, or push Ctrl+R key, or click 'Reset' button on debug toolbar, program will stop and system will transfer to initial state, and mouse shape will change to primary shape, program reset interface is shown as following figure 7-14:

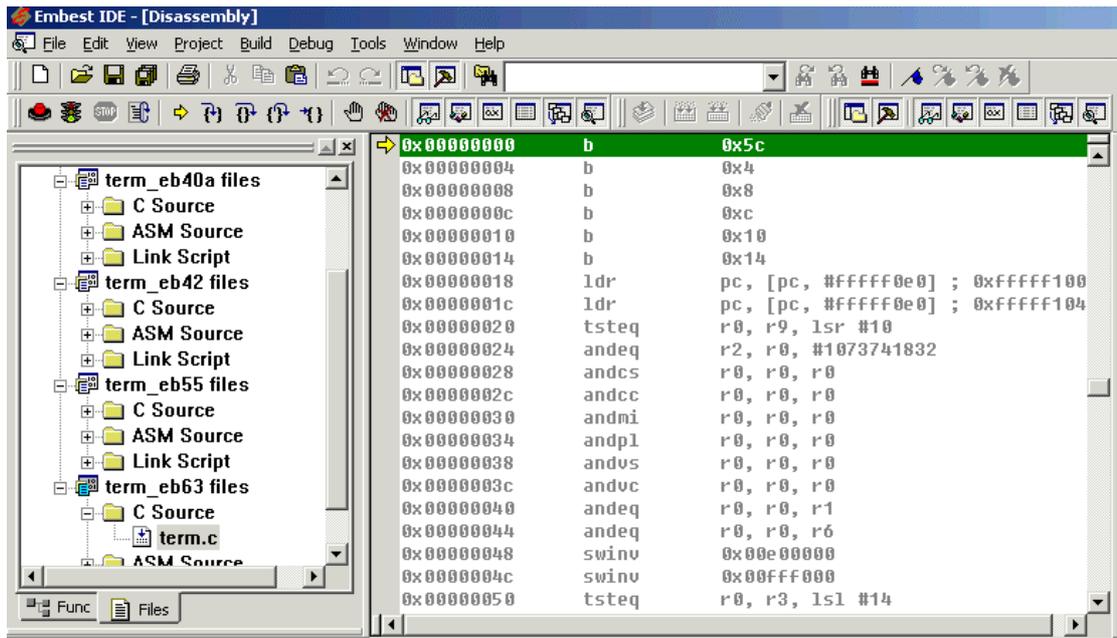


Figure 7-14 Program Reset Interface

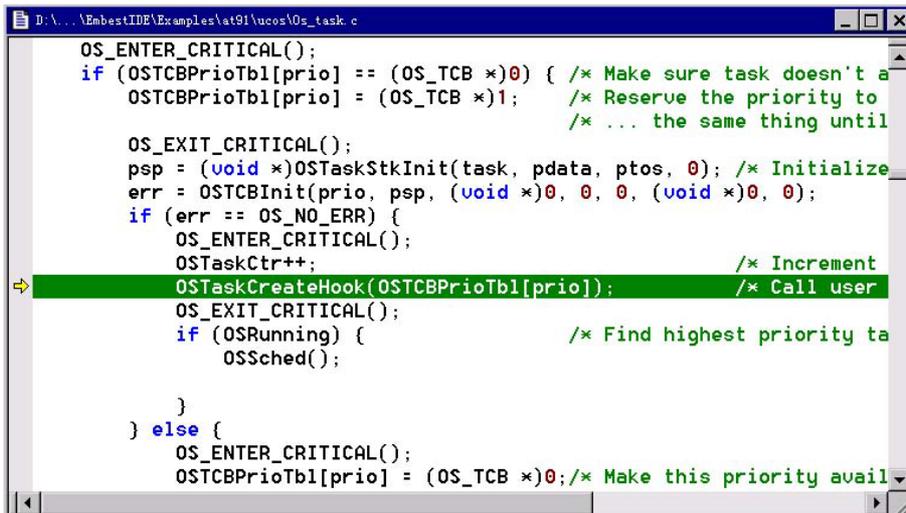
Source program also can be executed step by step. Step execute include Step Into、Step Over and Step Out mode.

1、 Step Into mode: If there is a subroutine call in the current line, Step Into takes program to the first line of that subroutine, not to the next line currently displayed on your screen.

2、 Step Over mode: Step Over steps program to the next line display on the screen.

3、 Step Out mode: program execution continues until the current subroutine completes, then the debugger regains control in the calling statement.

A example for step execute is shown as follows figure 7-15:(suppose system run into the position as the following figure, current subroutine is OSTaskCreate)

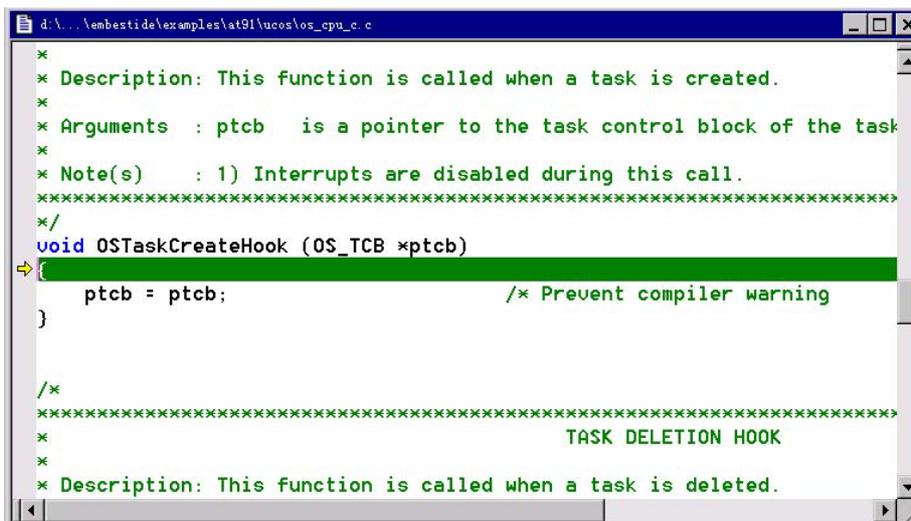


```
D:\...\EmbestIDE\Examples\at91\ucos\Os_task.c
OS_ENTER_CRITICAL();
if (OSTCBPrioTbl[prio] == (OS_TCB *)0) { /* Make sure task doesn't a
OSTCBPrioTbl[prio] = (OS_TCB *)1; /* Reserve the priority to
/* ... the same thing until

OS_EXIT_CRITICAL();
psp = (void *)OSTaskStkInit(task, pdata, ptos, 0); /* Initialize
err = OSTCBInit(prio, psp, (void *)0, 0, 0, (void *)0, 0);
if (err == OS_NO_ERR) {
OS_ENTER_CRITICAL();
OSTaskCtr++; /* Increment
OSTaskCreateHook(OSTCBPrioTbl[prio]); /* Call user
OS_EXIT_CRITICAL();
if (OSRunning) { /* Find highest priority ta
OSSched();
}
} else {
OS_ENTER_CRITICAL();
OSTCBPrioTbl[prio] = (OS_TCB *)0; /* Make this priority avail
```

Figure 7-15 Step Execute Start Position

If execute Step into, because the source line have a subroutine call, so system will run into the first line of OSTaskCreateHook subroutine, source window interface will changed as the following figure 7-16:

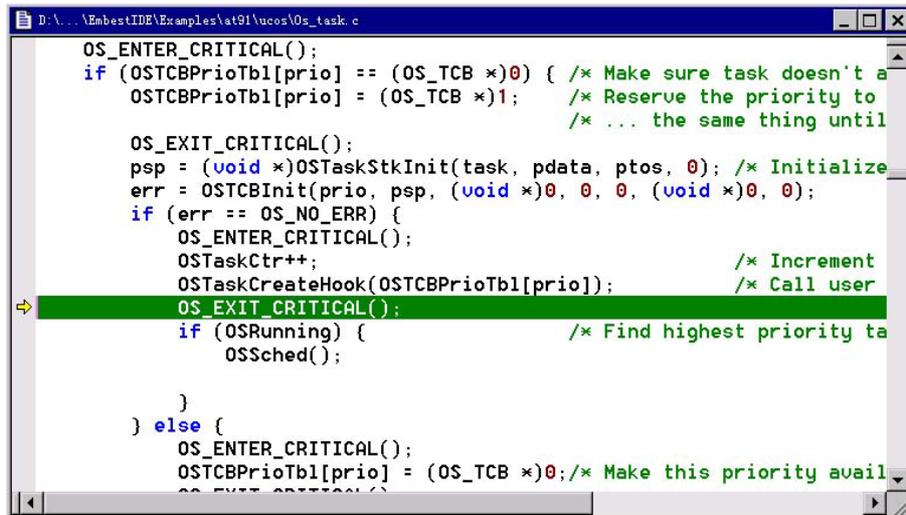


```
d:\...\Embestide\examples\at91\ucos\os_cpu_c.c
*
* Description: This function is called when a task is created.
*
* Arguments : ptcb is a pointer to the task control block of the task
*
* Note(s) : 1) Interrupts are disabled during this call.
*****
*/
void OSTaskCreateHook (OS_TCB *ptcb)
{
ptcb = ptcb; /* Prevent compiler warning
}

/*
*****
TASK DELETION HOOK
*
* Description: This function is called when a task is deleted.
```

Figure 7-16 Source Code Window Interface after Step Into

If execute Step over, system will stop at next line displayed on the screen, source code window will changed as the figure 7-17:



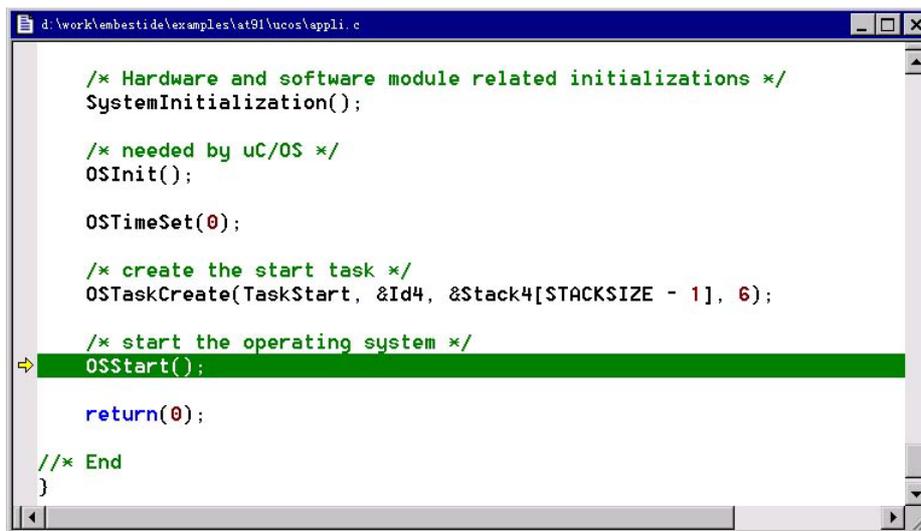
```
D:\... \EmbestIDE\Examples\at91\ucos\Os_task.c
OS_ENTER_CRITICAL();
if (OSTCBPrioTbl[prio] == (OS_TCB *)0) { /* Make sure task doesn't a
OSTCBPrioTbl[prio] = (OS_TCB *)1; /* Reserve the priority to
/* ... the same thing until

OS_EXIT_CRITICAL();
psp = (void *)OSTaskStkInit(task, pdata, ptop, 0); /* Initialize
err = OSTCBInit(prio, psp, (void *)0, 0, 0, (void *)0, 0);
if (err == OS_NO_ERR) {
OS_ENTER_CRITICAL();
OSTaskCtr++; /* Increment
OSTaskCreateHook(OSTCBPrioTbl[prio]); /* Call user
OS_EXIT_CRITICAL();
if (OSRunning) { /* Find highest priority ta
OSSched();

}
} else {
OS_ENTER_CRITICAL();
OSTCBPrioTbl[prio] = (OS_TCB *)0; /* Make this priority avail
OS_EXIT_CRITICAL();
```

Figure7-17 Source Code Window Interface after Step Over

If execute Step out, system will continues until the current subroutine completes, and stop at calling statement, source code window will changed as the following figure 7-18:



```
d:\work\embestide\examples\at91\ucos\appl1.c
/* Hardware and software module related initializations */
SystemInitialization();

/* needed by uC/OS */
OSInit();

OSTimeSet(0);

/* create the start task */
OSTaskCreate(TaskStart, &Id4, &Stack4[STACKSIZE - 1], 6);

/* start the operating system */
OSStart();

return(0);

/** End
}
```

Figure 7-18 Source Code Window Interface after Step Out

7.5.2 Disassemble Window

Disassemble window provides display of assemble code disassembled from binary machine code, and provides blend display between assemble code, source code, and binary code. Disassemble window can set and clear assemble breakpoint, and also can disassemble binary code in accoding with ARM or THUMB binary machine mode.

The buttons、shortcuts、and menu commands, correlative with disassemble window, are described in following table.

Table 7-2 Buttons、Shortcuts、and Menu Commands

| Button | Shortcut Key | Menu command |
|---|--------------|-----------------------------------|
|  | ALT+8 | View > Debug Windows > Call Stack |

Disassemble window is shown as following figure 7-19:

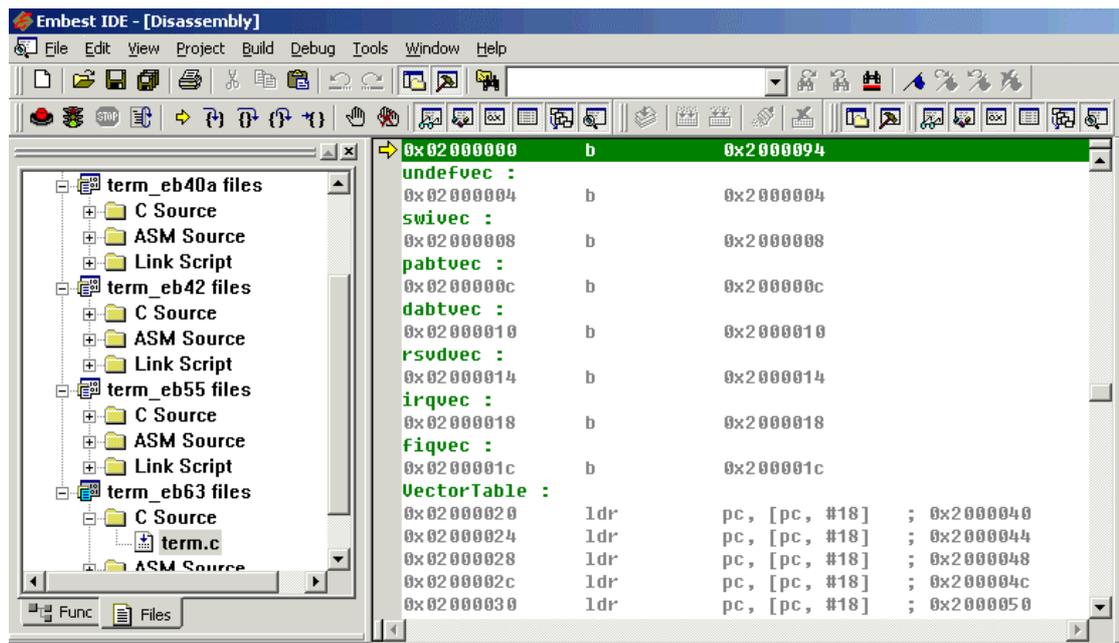


Figure 7-19 Disassemble Window Interface

Disassemble window blend source code and assemble code as the following figure 7-20:

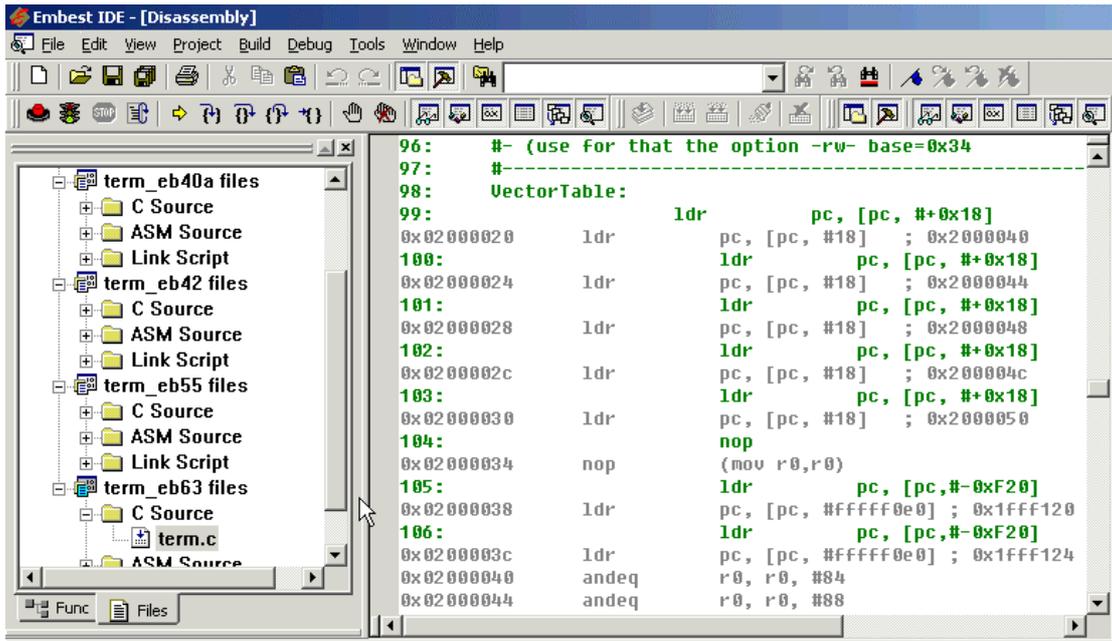


Figure 7-20 Disassemble Window Interface

Disassemble window blend binary code and assemble code as the following figure 7-21:

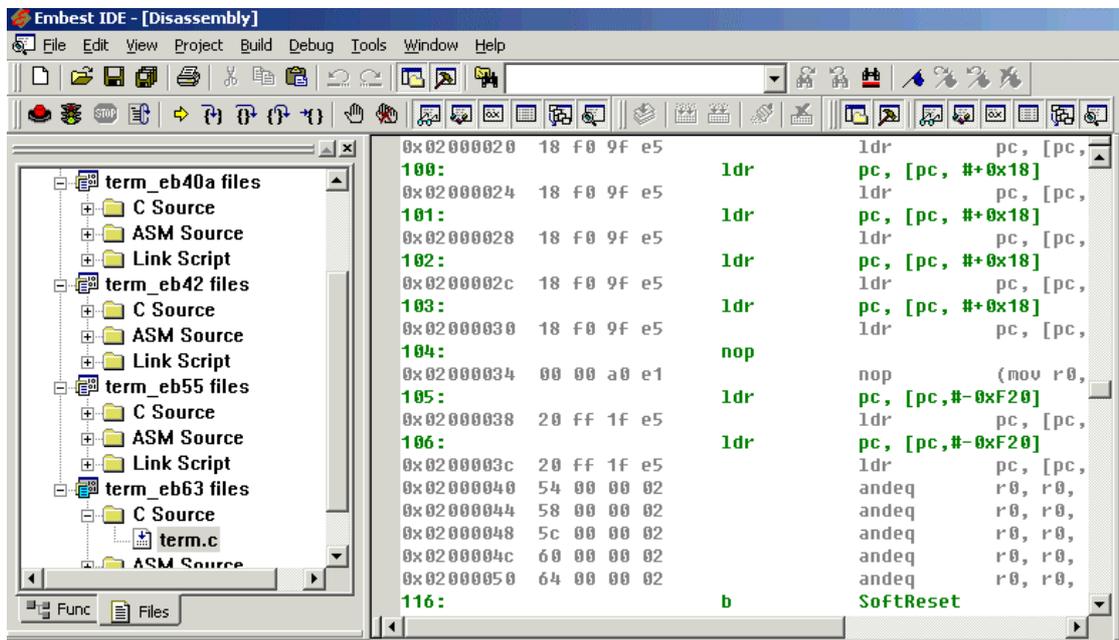


Figure 7-21 Disassemble Window Interface

The right mouse menu of disassemble window is shown as following figure 7-22.

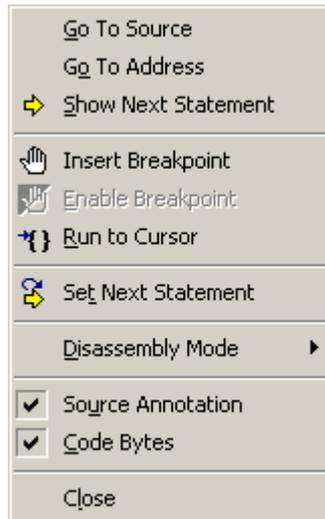


Figure 7-22 Disassemble Window Right Menu

Go To Source: show the source code line which in according with current assemble line, if the source line exists, debugger will show the source code window.

Go To Address: set start address of assemble code, and start disassembling binary code from that address.

Show Next Statement: show assemble code which will execute next step.

Insert Breakpoint: set a breakpoint at current assemble code line.

Enable Breakpoint: enable the breakpoint at current assemble code line.

Disable Breakpoint: disable the breakpoint at current assemble code line.

Delete Breakpoint: delete the breakpoint at current assemble code line.

Run to Cursor: run program to the line where the cursor is staying.

Set Next Statement: set the assemble code line, where the cursor is staying, as the line which will execute by system next.

Disassembly Mode: set disassemble mode ---- ARM or THUMB mode.

Source Annotation: display assemble code blend with source code.

Code Byte: display assemble code blend with binary code.

Close: close the disassemble window.

7.5.3 Breakpoint

The buttons, shortcut keys and menu items which relate with breakpoint is shown as following table 7-3:

Table7-3 Buttons, Shortcut keys and Menu items

| Button | Shortcut Key | Debug menu command | Right mouse menu |
|---|--------------|-------------------------|---|
|  | F9 | Toggle Breakpoints | Insert Breakpoint/ Delete Breakpoint |
| n/a | n/a | Enable All Breakpoints | |
| n/a | n/a | | Enable Breakpoint |
| n/a | n/a | Disable All Breakpoints | |
| n/a | n/a | | Disable Breakpoint |
|  | n/a | Delete All Breakpoints | |
| n/a | n/a | Breakpoints... | |

Before setting a breakpoint, symbol file must has been filled in debug setting dialog. Embest IDE debugger can set and clear breakpoints in source code window, assemble window, disassemble window and code blend display window.

Several way, hereinafter, to set breakpoint:

- 1、 Moving mouse to left grey margin of the source code window, mouse will change to hand shape, then click the left mouse button, a red-circle breakpoint flag will displays in the left grey margin and a yellow backgroundf bar will be shown at corresponding line.

- 2、 Set cursor to the line which need a breakpoint, and then press 'F9' .

- 3、 Set cursor to the line which need a breakpoint, then click 'Debug' menu and select 'Toggle Breakpoint' menu item.

The breakpoint, which is set first time, is a enable breakpoint, shown as following figure 7-23. The flag which is around by cyan circle is a enable breakpoint flag:

```

    /* 设置定时时间
    at91_tc_write(&TC0_DESC, timer_value);

    /* -- Software Trigger on Timer
    /* -- generates a software trigger simultaneously for each of the chan
    at91_tc_trig_cmd(&TC0_DESC, TC_TRIG_CHANNEL);

    /* 启动定时器
    at91_irq_open(TC0_DESC.periph_id, 7, AIC_SRCTYPE_INT_EDGE_TRIGGERED, &0

```

Figure 7-23 a Disable Breakpoint Flag

In a source code window, if a breakpoint is set at a invalid source line, Embest IDE will have none response.

Breakpoint state includes enable state and disable state, program will not auto stop at the disable breakpoints. Shown as the following figure 7-24, the flag around by cyan circle is a disable breakpoint flag:

```

    /* define led at PIO output
    at91_pio_open ( &PIO_DESC, LED_MASK, PIO_OUTPUT );

    /* define switch at PIO input
    at91_pio_open ( &PIO_DESC, SW_MASK, PIO_INPUT );

    /* Timer initialization
    at91_tc_open(&TC0_DESC, TC_WAVE|TC_CPCTR|TC_CLKS_MCK8,0,0);

```

Figure 7-24 Disable Breakpoint Flag

When running into a enable breakpoint line, program will stop at the breakpoint, shown as the following figure 7-25:

```

    /* 设置定时时间
    at91_tc_write(&TC0_DESC, timer_value);

    /* -- Software Trigger on Timer
    /* -- generates a software trigger simultaneously for each of the channels.
    at91_tc_trig_cmd(&TC0_DESC, TC_TRIG_CHANNEL);

    /* 启动定时器
    at91_irq_open(TC0_DESC.periph_id, 7, AIC_SRCTYPE_INT_EDGE_TRIGGERED, &0STickI

```

Figure 7-25 Program Stop at Breakpoint

User can query all breakpoint information and state through breakpoint list. Click 'Debug' menu, and select 'Breakpoint...' menu item, the breakpoints list dialog will pop-up, shown as the following figure 7-26:

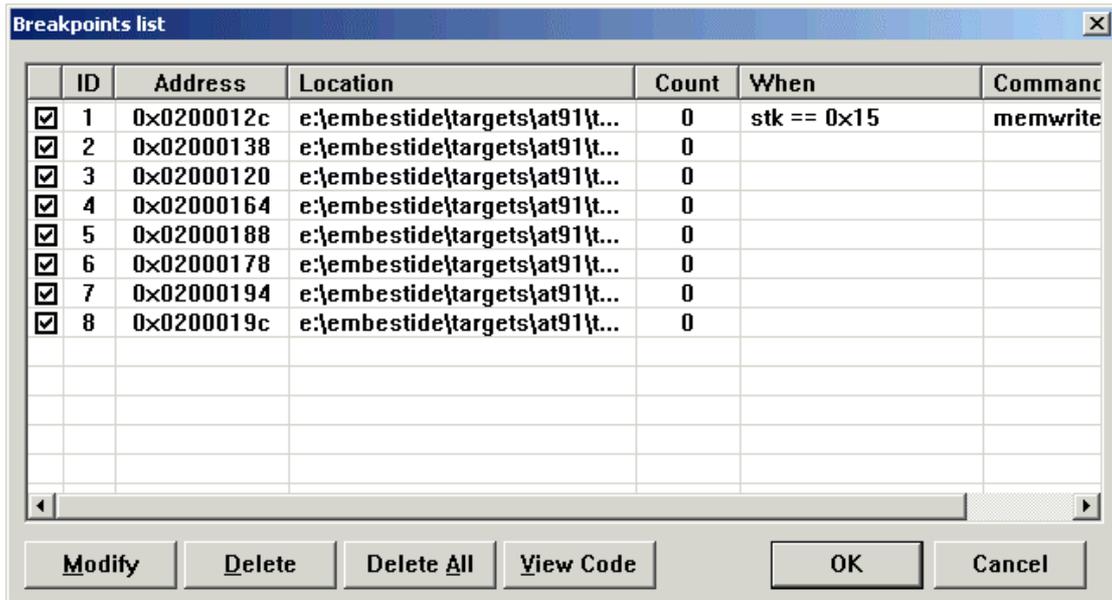


Figure 7-26 Breakpoints List Dialog

Double click left mouse button at a line of the breakpoint list or click Modify button, user can modify the breakpoint information in a dialog, shown as the following figure 7-27:

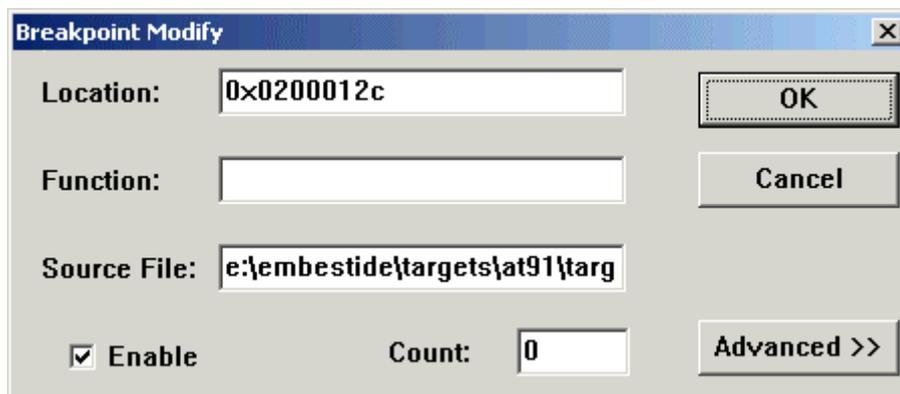


Figure 7-27 Breakpoint Modify Dialog

If want to set a conditional breakpoint, click 'Advanced' button which on breakpoint modify dialog, dialog will append a subdialog below the dialog, shown as the following figure 7-28, the 'When' editbox displays conditional express, the 'Command' editbox displays the command which will be auto executed as system reach the breakpoint.

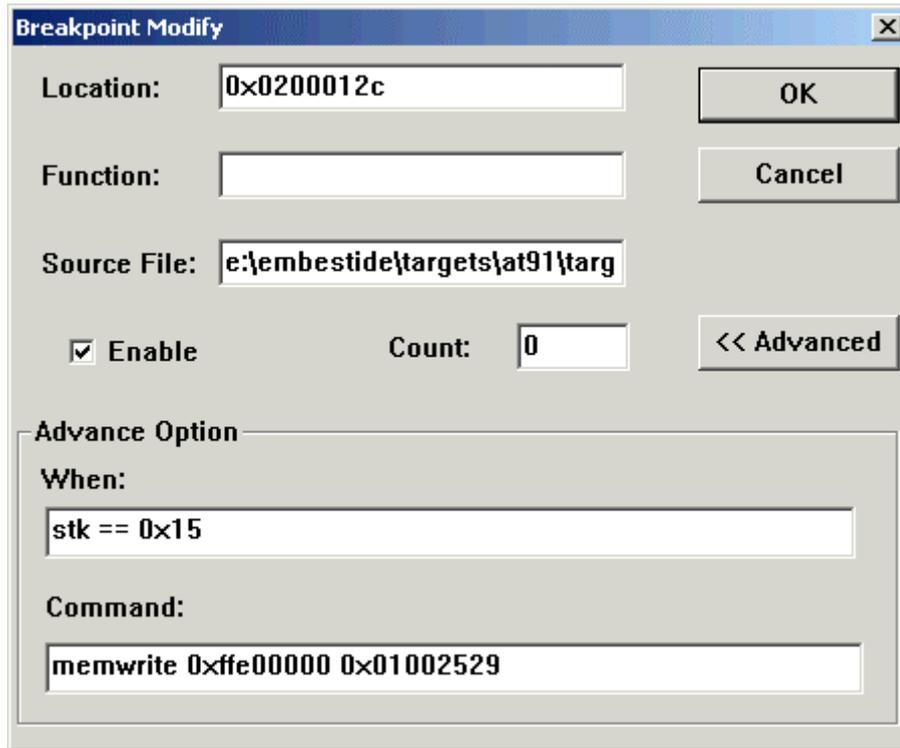


Figure 7-28 Conditional Breakpoint Modify Dialog

7.6 Debug Information

When users want to debug a program, they need much debug information to make sure the correctness of program and data, so can faster find the origin of errors. Embest IDE has visual debug information windows to display and modify debug information when user debug a target program, these windows include register window、memory window、watch window、variables window and call stack window.

7.6.1 Register Window

Register window can display and modify values of processor core registers and peripheral chip registers on the target system. Registers name and num depend on the type of target system processor, when debug different target system, the content of the register window also differ. Registers values can display on hexadecimal or decimal or binary format, and can auto refresh values or refresh by hand. Registers divide into register group, each group can set different display mode.

The buttons、shortcut、and menu commands which is correlative with register window are described in following table7-4.

table7-4 register window's buttons、shortcut、and menu commands

| Button | Shortcut Key | Menu command |
|---|--------------|---------------------------------|
|  | ALT+5 | View > Debug Windows > Register |

Register window show as following figure 7-29:

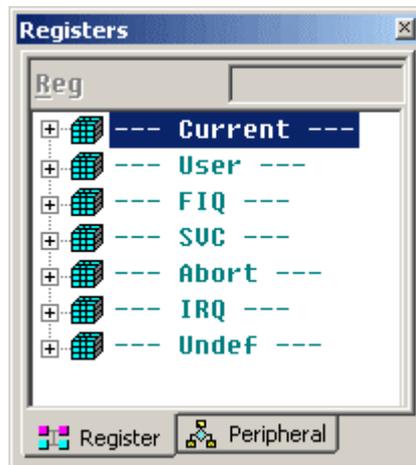


Figure7-29 register window interface

Click one register, the name and value of the register will display in input column which is on top of register window. User can modify the value in input column, the input column show as following figure7-30:

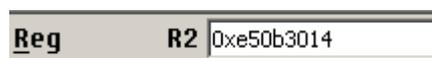


Figure 7-30 register value modify column

When register value modified, register window will show the value in red color, the interface of register window show as following figure7-31:

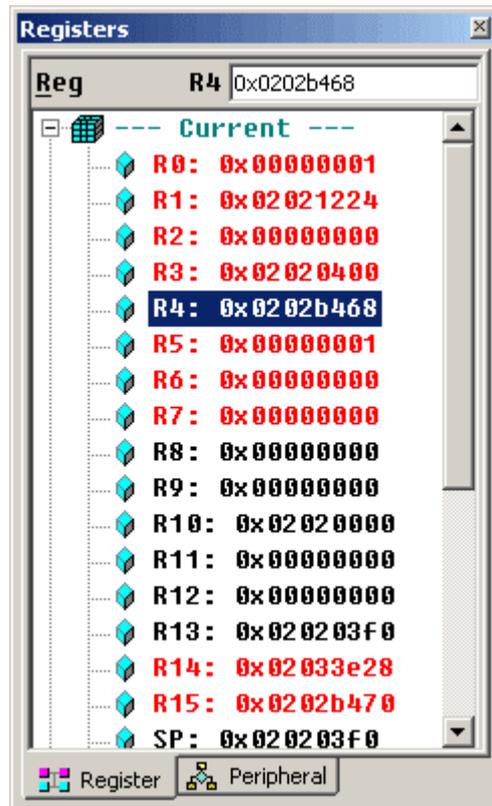


Figure 7-31 Register Window interface

Right click mouse on register window will show the register window menu, the register window menu show as following figure7-32, the meanings of menu item is:



Figure 7-32 Register Window Popup Menu

Refresh: handy refresh registers value to keep consistent with target system.

Auto Refresh: auto refresh option, if be set, register group name will in deep green, and register window will auto refresh registers value base on every operation of user to keep consistent with target system.

Hex Format: display registers value base on hexadecimal format.

Binary Format: display registers value base on binary mode.

Decimal Format: display registers value base on decimal format.

Expand ...: expand all register groups(...=All) or appointed register group(...=register group name).

Collapse ...: collapse all register groups(...=All) or appointed register group(...=register group name).

Docking View: window auto arrange option, if set, window will auto keep to the side and ordinal arrange.

Hide: hide register window.

Note: Setting the data display format, please attention by following:

1) right key on the register name, settings will effect on the appointed register.

2) right key on the register group name, settings will effect on the appointed register group.

3) right key on the blank of the register window, settings will effect on the all register group and registers.

7.6.2 Peripheral register window

Peripheral register window provides status display and operation of peripheral register of target processor. It can view and amend the content of peripheral register in this window. The relevant buttons, shortcut keys and menu commands of peripheral register window are shown in form 7-5.

Form 7-5 Buttons, shortcut keys and menu commands in storage area window

| Button | Shortcut key | Menu command |
|---|--------------|----------------------------------|
|  | ALT+5 | View > Debug Windows > Registers |

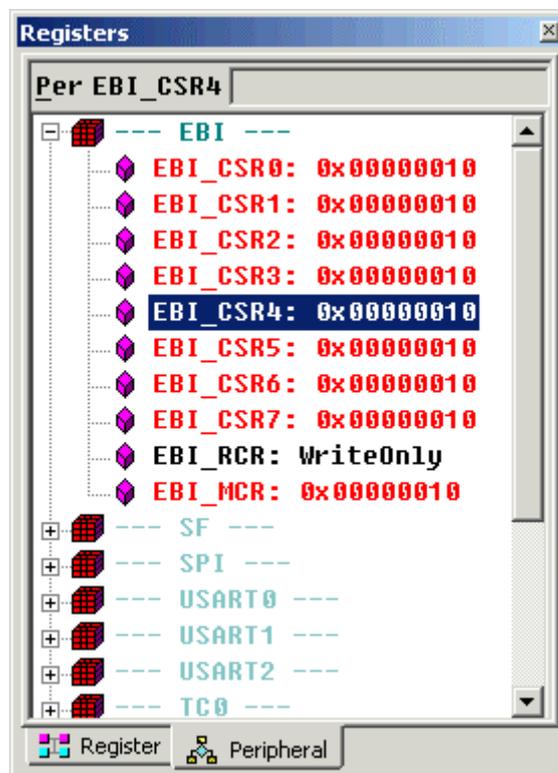


Fig. 7-33 Peripheral register window

As shown in the above figure, peripheral register window shows the peripheral register groups of current target processor with a list. The register group can be separately carried out refresh setting, data display format setting. Click  to view the list of peripheral registers of peripheral register group, and the user can amend the content of appointed register.

7.6.2.1 View peripheral register

Click  to view the list of peripheral registers of register group in peripheral register window. The display format of peripheral register is:

Name of register: current content or property of register

The readable register will directly display the current content; for write-only register, it will display the character string WriteOnly in the back of register.

When the mouse stops above register, the screen will prompt the description of register that the mouse currently points out, and the meanings of each part are:

Name of register (register mapping address): register description (accessing property)

As shown in Fig. 7-34:

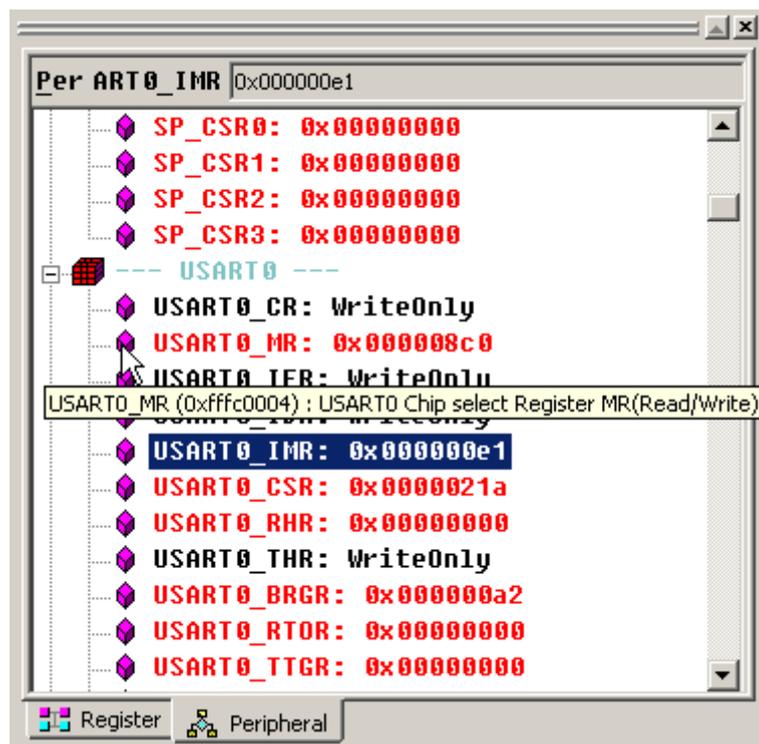


Fig. 7-34 Value display of peripheral register

7.6.2.2 Operation of peripheral register

In name of register group, click \oplus or click right key of mouse to select Expand ... to expand register group. Click a register, the name and value of this register will display in the input column in the upper part of register's window. In input column, it can change the value of this register. The input column is shown in fig. 7-35:



Fig. 7-35 Amendment and input column of register's value

The value of register will be shown in red after being amended, as shown in fig. 7-36:

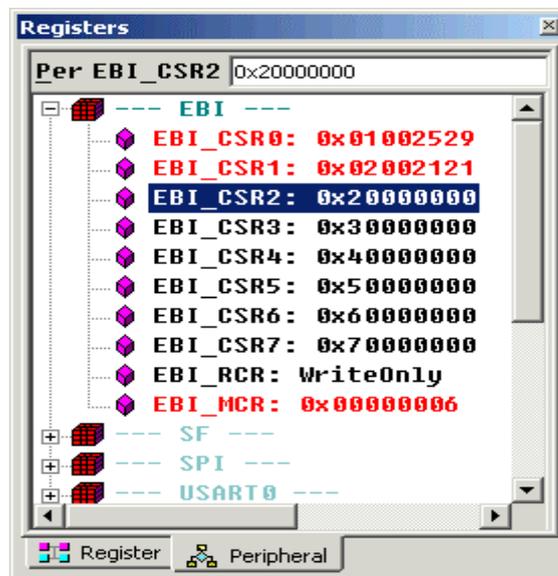


Fig. 7-36 Show corresponding register in red after the value of register is amended

While clicking right key in the window of peripheral register, it will pop out the menu as shown in fig. 7-37.



Fig. 7-37 Right key menu in window of peripheral register

| Menu | Function |
|----------------|--|
| Refresh ... | Manually refresh all value of registered groups (...=All) or value of appointed register groups (...=group name) so as to keep consistent with target system |
| Auto Refresh | Set automatic refresh. While setting, the window of register will automatically keep consistent with target system according to the operation of user in each step. While using this setting, the corresponding all names of register groups (All) or appointed register groups (name of group) will be shown in deep green. |
| Properties | Display detailed window of peripheral register |
| Hex Format | Display register's value according to hex format |
| Binary Format | Display register's value according to binary format |
| Decimal Format | Display register's value according to decimal format |
| Expand ... | Expand all register groups (...=All) or appointed register groups (...=group name) |
| Collapse ... | Collapse all register groups (...=All) or appointed register groups (...=group name) |

| | |
|--------------|--|
| Docking View | Whether the window is automatically arranged to the side. When selection, the window will be automatically arranged to the side. |
| Hide | Hide register window |

When the user uses the setting Auto Refresh to all register groups, the system will automatically pop out dialog box as shown in fig. 7-38. When the user executes a debugging operation, the integrated environment must read all values of peripheral registers, therefore the debugging speed may be affected in some content. It is suggested that the user should selectively use the function of Auto Refresh, and only set automatic refresh of some groups, use manual refresh to other register groups if necessary.



Fig. 7-38 Prompt when peripheral registers use the setting Auto Refresh

Notes:

- *While using refresh and automatic refresh of right key menu, it shall pay attention to the position where the mouse points:*
 - 1) *The setting that is carried out with right key above name of special register group is effective against this register group;*
 - 2) *The setting that is carried out with right key in other blank of register is effective against all register groups.*
- *While using data display format of right key menu, it shall pay attention to the position where the mouse points:*
 - 1) *The setting that is carried out with right key above name of special register group is effective against this register group;*
 - 2) *The setting that is carried out with right key in special register group*

is effective against this register group;

3) The setting that is carried out with right key in other blank of register is effective against all register groups.

The user shall be careful when he uses right key to operate.

7.6.2.3 Detailed dialog box of peripheral register

The dialog box of peripheral register provides for user the visual viewing measures of peripheral register of detailed register and flexible and convenient amendment methods to the value of peripheral register.

The detailed dialog box provides for user the detailed information about peripheral register, including the actual meaning of content, address and bit; the user can amend the value of peripheral register through directly inputting hex value or binary value, and can also amend the selected register area.

After selecting register, select sub-menu Properties in right key menu of mouse, or double click left key of mouse in register to pop out detailed dialog box of register as shown in fig. 7-39:

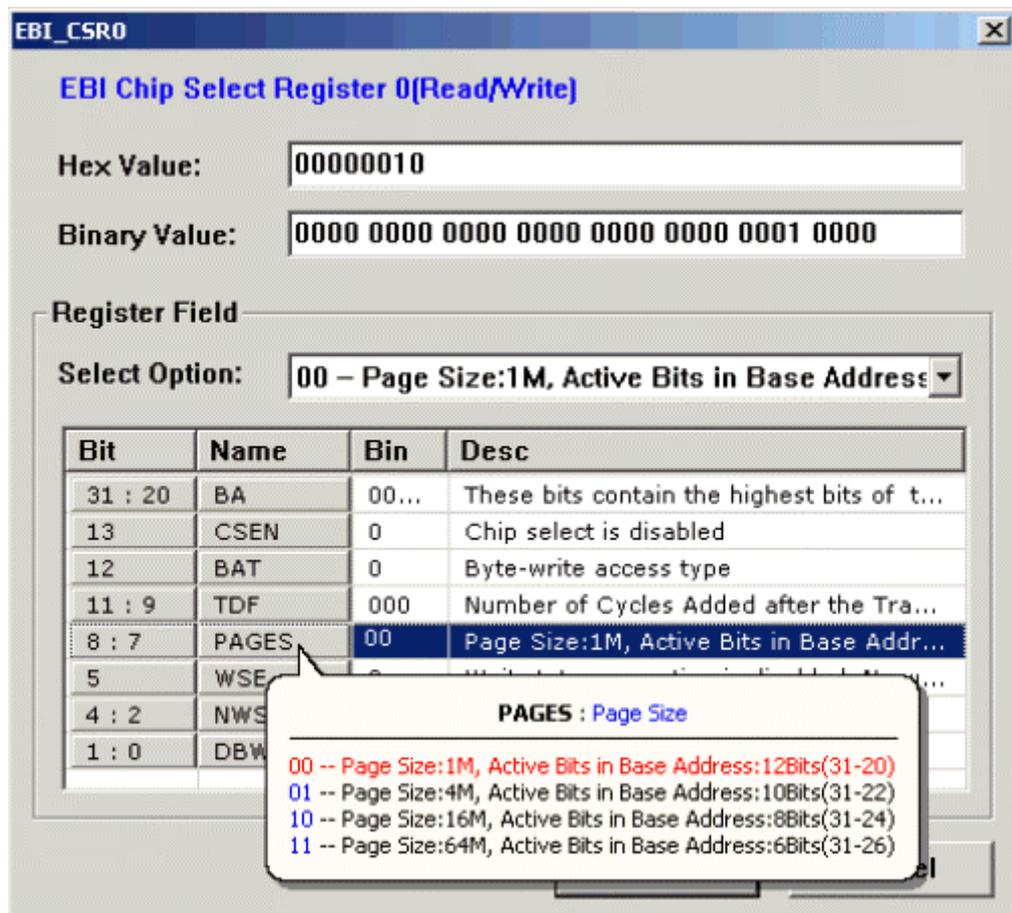


Fig. 7-39 Detailed dialog box of peripheral register

The detailed dialog box of register includes the following:

Heading of dialog box: display the name of register.

Blue letterform above dialog box: display the full description of current register.

Hex edit box (HEX Value): Display hex value of register, can also be used for amending the value of register.

Binary edit box (Binary Value): Display binary value of register, can also be used for amending the value of register.

Register Field:

Select drop-down box (Select Option) : Selectable value and the meaning that the register field currently selected is corresponding.

Bit: The bit group of register.

Name of field (Name) : Short name of bit group of register.

Binary value of field (Bin) : Binary value of bit group of register.

Description of field (Desc) : Description character string of current set of bit group of register.

Screen prompt: display short name of bit group of register, full name of bit group of register, all selectable values that the bit group of register is corresponding to, and the meaning when the value is set. The value of the bit group of register currently set and its meaning are shown in red.

If the user wants to amend the content of peripheral register, he can directly amend in hex and binary edit box, or amend the value in this field in Select Option or Edit Box after selection of register field. The drop-down select box of detailed window of peripheral register is shown in fig. 7-40.

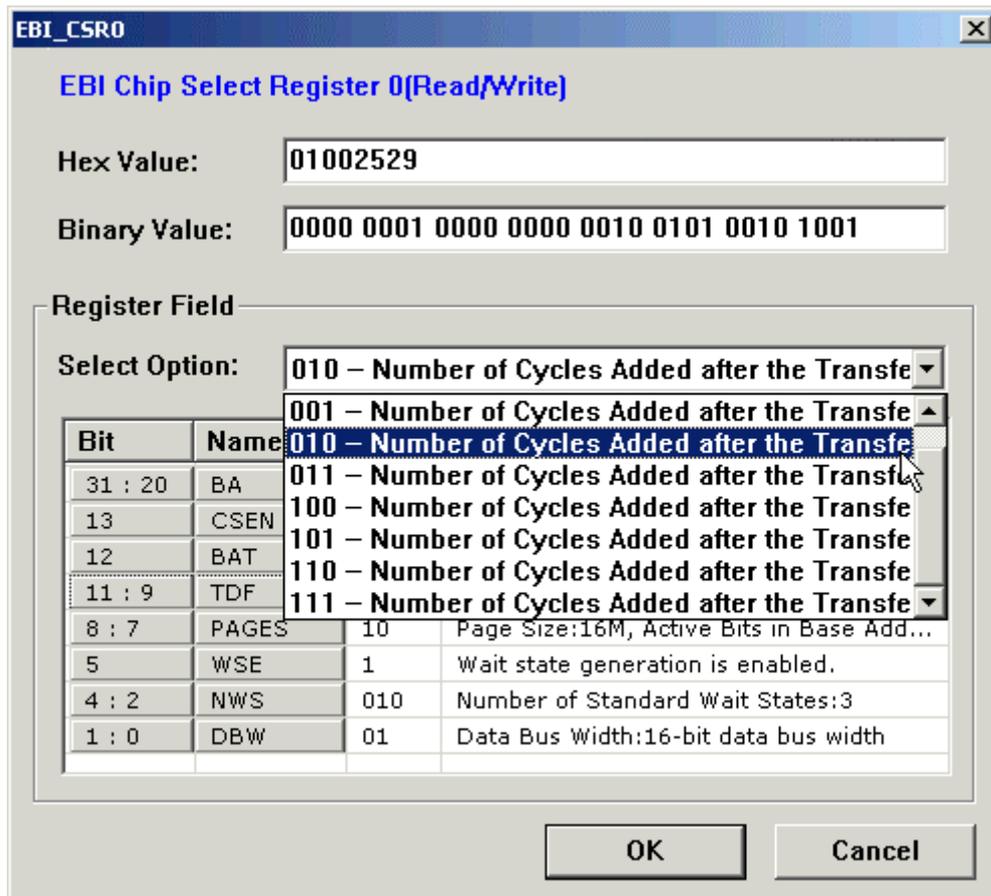


Fig. 7-40 Drop-down select box of detailed window of peripheral register

7.6.3 Memory Window

Memory window can display and modify memory content of target system. Memory window show memory content from the address which can be input by user, content length auto match the size of memory window. Memory content can display on byte、half-word or word length mode, and have hexadecimal digit format part and ASCII char format part which can respective display in accordance with memory content. When some memory content change, memory window will show these content on red color. Embest IDE provides two of the memory window named Memory Window1 and Memory Window2.

The buttons、shortcut、and menu commands which is correlative with memory window are described in following table7-5.

Table 7-5 memory window's buttons、shortcut、and menu commands

| Button | Shortcut key | Menu command |
|---|--------------|-------------------------------|
|  | ALT+6 | View > Debug Windows > Memory |

Memory window display base on byte length mode show as following figure 7-41:

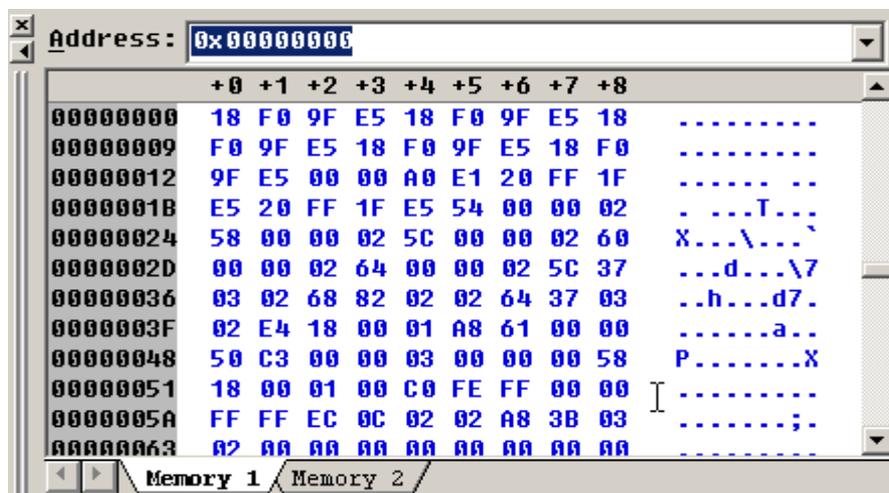


Figure 7-41 Memory Window Base on Byte Length Mode

User can modify original address of memory content in the input column which is on top of memory window. Ten of the user's input will remain in the list of the input column. If modified original address, memory window will immediately auto show new memory content. The input column show as following figure 7-42:

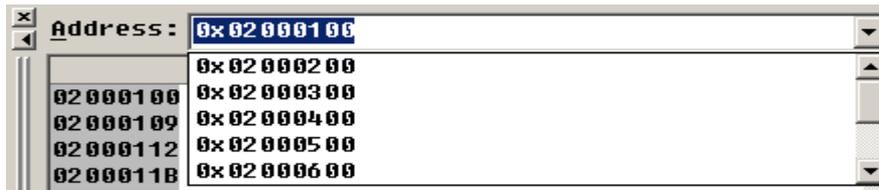


Figure 7-42 Original Memory Address Input Column

If need modify memory content, user can directly modify the content on hexadecimal digit part or ASCII char part, and new content will immediately write into corresponding memory, and show new content on red color. The interface which memory content modified show as following figure7-43:

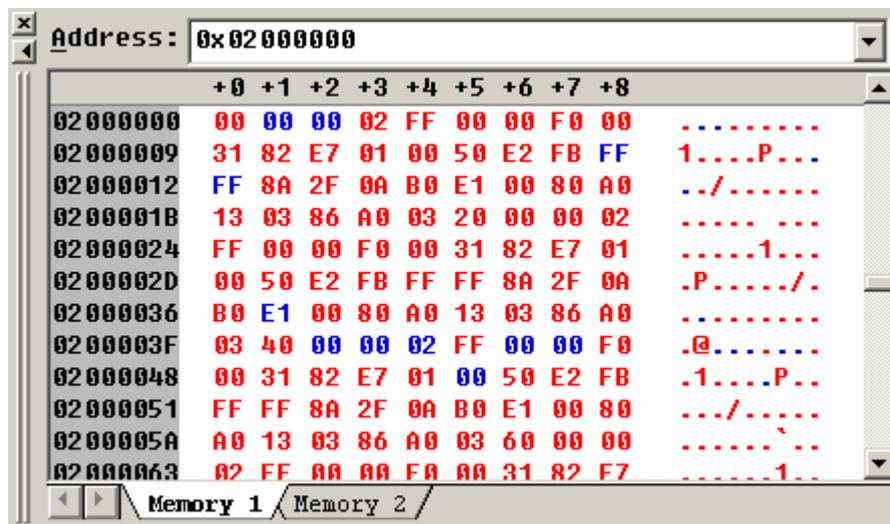


Figure 7-43 Memory Window with content change

Right click mouse on memory window will show the memory window menu, the memory window menu show as following figure7-44, the meanings of menu item is:

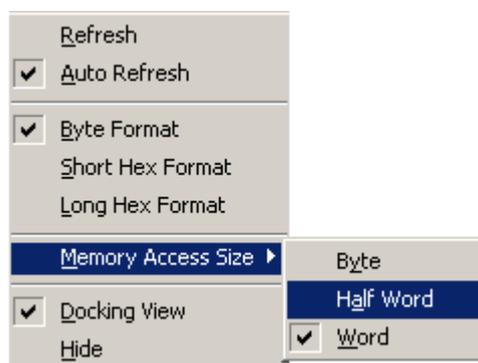


Figure 7-44 Memory Window Popup Menu

| 菜单 | 功能 |
|---------------------------|---|
| Refresh | handly refresh memory content to keep consistent with target system. |
| Auto Refresh | auto refresh option, if be set, memory window will auto memory content base on every operation of user to keep consistent with target system. |
| Byte Format | display memory content base on byte length mode. |
| Short Hex Format | display memory content base on half-word length mode. |
| Long Hex Format | display memory content base on word length mode. |
| Memory Access Size | Set the access size of memory Byte ---- by byte Half Word ---- by half word Word ---- by word |
| Docking View | window auto arrange option, if set, window will auto keep to the side and ordinal arrange. |
| Hide | hide memory window. |

Note: with Memory Window1for illustration above, so as Memory Window2 can be able to operate. But their options are independency.

7.6.4 Watch Window

Watch window can display variables value or compute expression result which user input for watch, user can add a new watch data or delete a watch data. Watch data value can display base on hexadecimal or decimal format. With every operation of user, watch window will auto compute and update watch data value. Watch window have two page: "Watch 1" and "Watch 2", each page can separate input different data. Watch data name will auto save follow project save or close, when open same project next time, watch window will auto load last watch data.

The buttons、shortcut、and menu commands which is correlative with watch window are described in following table7-6.

Table7-6 watch windows's buttons、shortcut、and menu commands

| Button | Shortcut key | Menu command |
|--|--------------|------------------------------|
|  | ALT+3 | View > Debug Windows > Watch |

Watch window default display base on hexadecimal format, window interface show as following figure7-45:

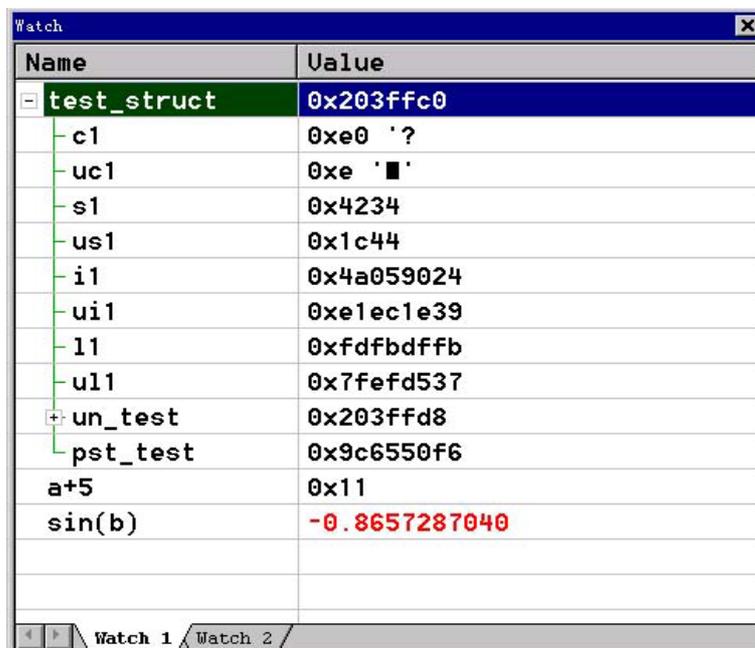


Figure 7-45 Watch Window Base on Hex Format

Watch window interface base on decimal format show as following figure 7-46:

| Name | Value |
|-------------|---------------|
| test_struct | 0x203ffc0 |
| - c1 | 0xe0 '?' |
| - uc1 | 0xe '■' |
| - s1 | 16948 |
| - us1 | 7236 |
| - i1 | 1241878564 |
| - ui1 | 3790347833 |
| - l1 | -33824773 |
| - u11 | 2146424119 |
| + un_test | 0x203ffd8 |
| - pst_test | 0x9c6550f6 |
| a+5 | 17 |
| sin(b) | -0.8657287040 |
| | |
| | |

Figure 7-46 Watch Window Base on Decimal Format

User can use two kinds of ways to add new watch data:

- 1、 Double click the name column of the blank line in watch window, then will put a input box into the line, user can input new data in that box, carriage return or click other line, watch window will auto compute data value and show it on corresponding value column. The interface show as following figure 7-47:

| Name | Value |
|-------------|---------------|
| test_struct | 0x203ffc0 |
| - c1 | 0xe0 '?' |
| - uc1 | 0xe '■' |
| - s1 | 16948 |
| - us1 | 7236 |
| - i1 | 1241878564 |
| - ui1 | 3790347833 |
| - l1 | -33824773 |
| - u11 | 2146424119 |
| + un_test | 0x203ffd8 |
| - pst_test | 0x9c6550f6 |
| a+5 | 17 |
| sin(b) | -0.8657287040 |
| &a+10 | |
| | |

Figure 7-47 Watch Data Input Column

2、Right click on watch window, select 'Add' menu item, then will show a data input dialog, enter data, push 'OK' button, watch window will compute data value and add it at the end of watch window. The input dialog show as following figure 7-48:



Figure 7-48 Watch Data Input Dialog

User can look over detail data property, select corresponding column, right click mouse, select 'Properties' menu item, will show data property dialog, interface show as following figure 7-49:

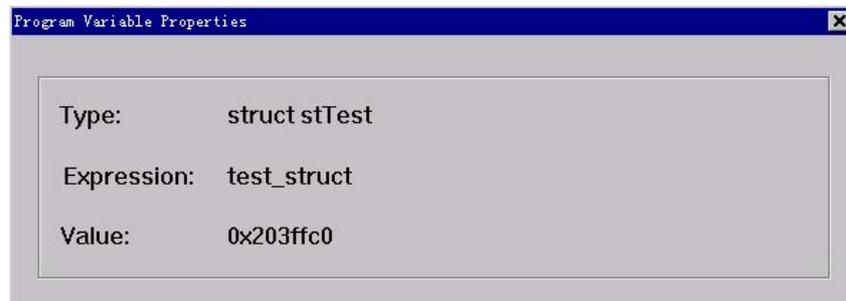


Figure 7-49 Watch Data Properties Dialog

Right click mouse on watch window will show the watch window menu, the watch window menu show as following figure7-50, the meanings of menu item is:



Figure 7-50 Watch Window Popup Menu

Add: add a new watch data.

Delete: delete current selected watch data.

Hexadecimal Display: data value format option, if set will show data value on hexadecimal format, if not set will show data value on decimal format.

Docking View: window auto arrange option, if set, window will auto keep to the side and ordinal arrange.

Hide: hide watch window.

Properties: current selected watch data property.

7.6.5 Variables Window

Variables window can display global and local variables information, and can modify variables's value. Variables window have two page: 'Global' and 'Local', respective display global variables and local variables. Variables value can display base on hexadecimal or decimal format. With every operation of user, watch window will auto create variables list and compute variables value. If one variable's value changed, variables window will display the value on red color.

The buttons、shortcut、and menu commands which is correlative with variables window are described in following table 7-7.

Table 7-7 variables windows's buttons、shortcut、and menu commands

| Button | Shortcut key | Menu command |
|---|--------------|----------------------------------|
|  | ALT+4 | View > Debug Windows > Variables |

Variables window default display base on hexadecimal format, window interface show as following figure 7-51:

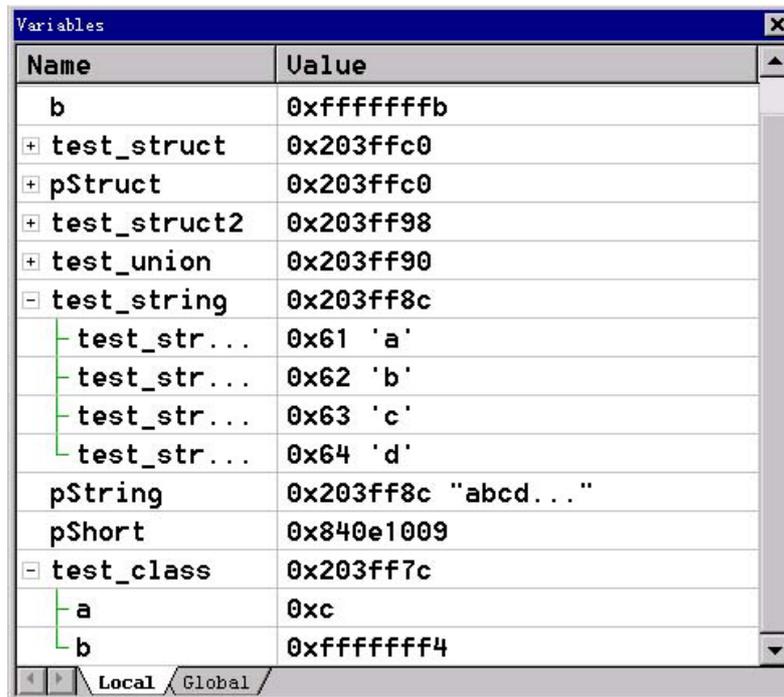


Figure 7-51 Variables Window Base on Hex Format

Variables window interface base on decimal format show as following figure 7-52:

| Name | Value |
|----------------|---------------------|
| b | 4294967291 |
| + test_struct | 0x203ffc0 |
| + pStruct | 0x203ffc0 |
| + test_struct2 | 0x203ff98 |
| + test_union | 0x203ff90 |
| - test_string | 0x203ff8c |
| - test_str... | 0x61 'a' |
| - test_str... | 0x62 'b' |
| - test_str... | 0x63 'c' |
| - test_str... | 0x64 'd' |
| pString | 0x203ff8c "abcd..." |
| pShort | 0x840e1009 |
| - test_class | 0x203ff7c |
| - a | 12 |
| - b | -12 |

Figure 7-52 Variables Window Base on Adecimal Format

User can look over detail variables property include variable name、 variable value and variable type, right click on variables window, select ' Properties' menu item, will show variables property dialog, interface show as following figure 7-53:

| | |
|-------------|------|
| Type: | int |
| Expression: | a |
| Value: | 3973 |

Figure 7-53 Variables Property Dialog

User can modify variable value. The way is double click the value column of the variable which need to be modified, and input new value in input box, interface show as following figure 7-54:

| Name | Value |
|----------------|---------------------|
| b | 0xffffffffb |
| + test_struct | 0x203ffc0 |
| + pStruct | 0x203ffc0 |
| + test_struct2 | 0x203ff98 |
| + test_union | 0x203ff90 |
| - test_string | 0x203ff8c |
| - test_str... | 0x61 'a' |
| - test_str... | 0x62 'b' |
| - test_str... | 0x63 'c' |
| - test_str... | 0x64 'd' |
| pString | 0x203ff8c "abcd..." |
| pShort | 0x840e1009 |
| - test_class | 0x203ff7c |
| a | 0x5 |
| b | 0xffffffff4 |

Figure 7-54 Variables modify

When new variable value is legal, watch window will immediately change variable's value on target system, and display new value on red color, interface show as following figure 7-55:

| Name | Value |
|----------------|---------------------|
| b | 0xffffffffb |
| + test_struct | 0x203ffc0 |
| + pStruct | 0x203ffc0 |
| + test_struct2 | 0x203ff98 |
| + test_union | 0x203ff90 |
| - test_string | 0x203ff8c |
| - test_str... | 0x61 'a' |
| - test_str... | 0x62 'b' |
| - test_str... | 0x63 'c' |
| - test_str... | 0x64 'd' |
| pString | 0x203ff8c "abcd..." |
| pShort | 0x840e1009 |
| - test_class | 0x203ff7c |
| a | 0x5 |
| b | 0xffffffff4 |

Figure 7-55 Variables Window after a variable value change

Right click mouse on variables window will show the variables window menu, the variables window menu show as following figure 7-56, the meanings of menu item is:



Figure 7-56 Variables Window Popup Menu

Hexadecimal Display: variable value format option, if set will show variable value on hexadecimal format, if not set will show variable value on decimal format.

Docking View: window auto arrange option, if set, window will auto keep to the side and ordinal arrange.

Hide: hide variables window.

Properties: current selected variable property.

7.6.6 Call Stack Window

Call stack window can display runtime relation of functions which be calling and called. Function parameter value can display base on hexadecimal or decimal format. The last called function(current running function) display on top line of the window, and arrange down base on function call relation, start function display on end line of the window.

The buttons、shortcut、and menu commands which is correlative with call stack window are described in following table7-8.

Table7-8 call stack window's buttons、shortcut、and menu commands

| Button | Shortcut key | Menu command |
|---|--------------|-----------------------------------|
|  | ALT+7 | View > Debug Windows > Call Stack |

Call stack window interface show as following figure 7-57 (have been set all function parameter display property):

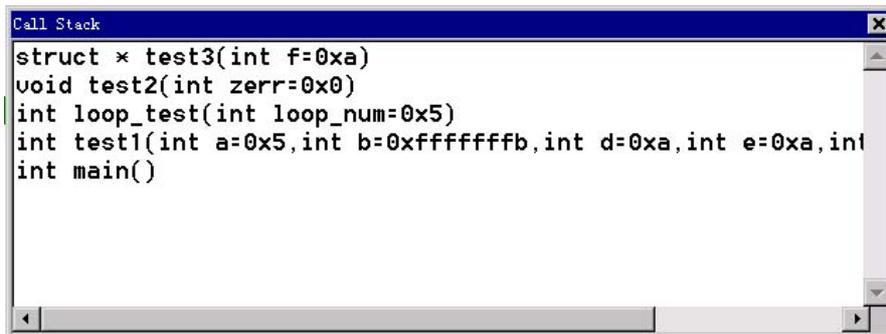


Figure7-57 Call Stack Window

Function Parameter's name or type or value can be set to display whether or not individual. Call stack window interface show as following figure7-58 (Close function parameter value display property)

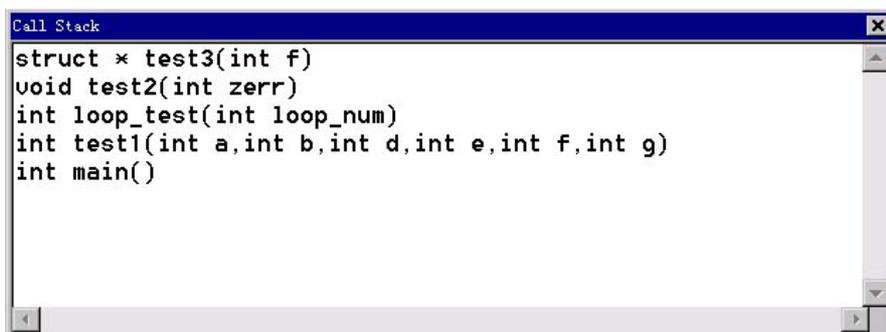


Figure 7-58 Call Stack Window with Function Parameter value

If close all function parameter display property will only show function name and its return type. Call stack window interface show as following figure 7-59 (Close all function parameter display property)



Figure7-59 Call Stack Window without Function Parameter

Double click one function line of call stack window, source window will show next code to be execute after the code which the function call above function, and put a blue rightward arrow on front of the source line, call stack window will also show the function line highlight. The interface show as following figure 7-60:

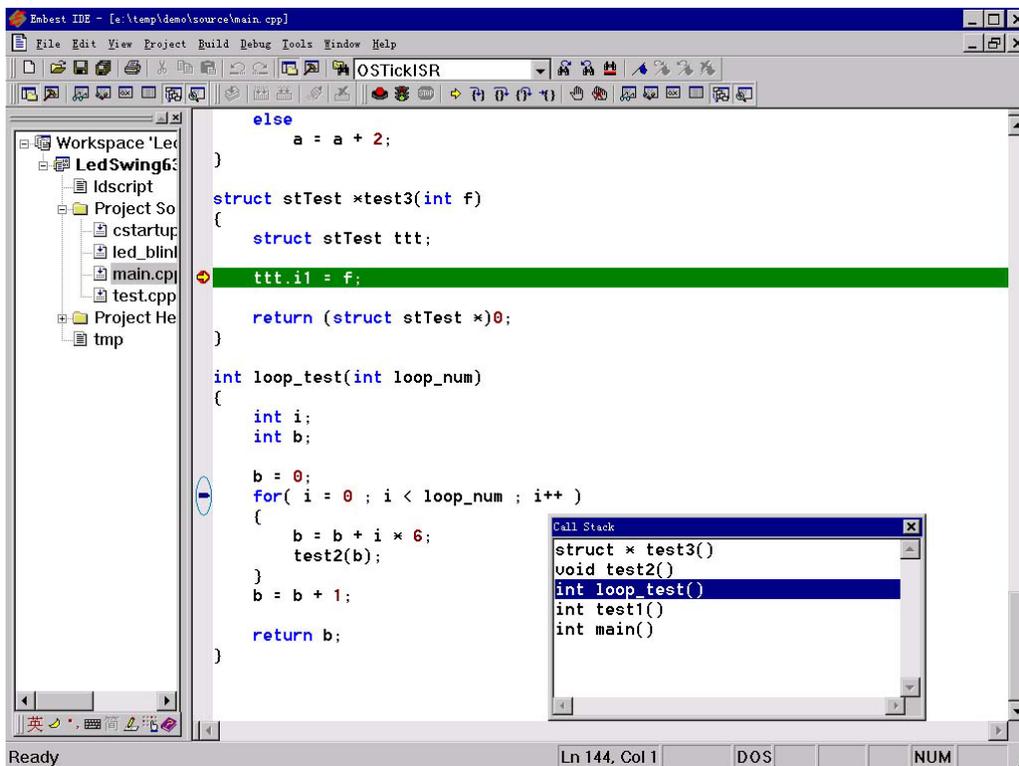


Figure 7-60 Debugger Interface as Double-Click Function Line

Right click mouse on call stack window will show the call stack window menu, the call stack window menu show as following figure, the meanings of menu item is:

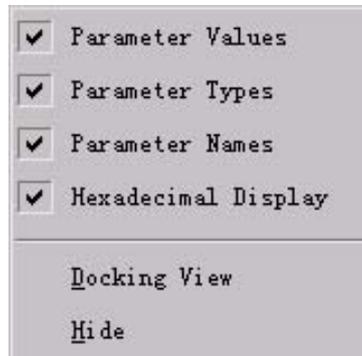


Figure 7-61 Call Stack Window Popup Menu

Parameter Values: parameter values display option, if set will show parameter values.

Parameter Types: parameter types display option, if set will show parameter types.

Parameter Names: parameter names display option, if set will show parameter names.

Hexadecimal Display: parameter values format option, if set will show parameter values on hexadecimal format, if not set will show parameter values on decimal format.

Docking View: window auto arrange option, if set, window will auto keep to the side and ordinal arrange.

Hide: hide call stack window.

8. Customization and Options

8.1 Introduction

Embest IDE not only allows you to customize the appearance of the display to match your preferences, but it also allows you to add menu entries for other tools you may wish to use. The Options entry in the Tools menu displays commands that change the editor settings, and plugin directory for Embest IDE. The Customize entry in the Tools menu opens a dialog box for adding menu items.

Table 8-1 Customization and Options Menu

| Button | Menu | Description |
|--------|--------------|-------------------------------|
| | Customize... | Open the Customize dialog box |
| | Options... | Open the Option dialog box |

8.2 Tools Menu Customization

Select Tools > Customize... menu, open the Customize dialog box:

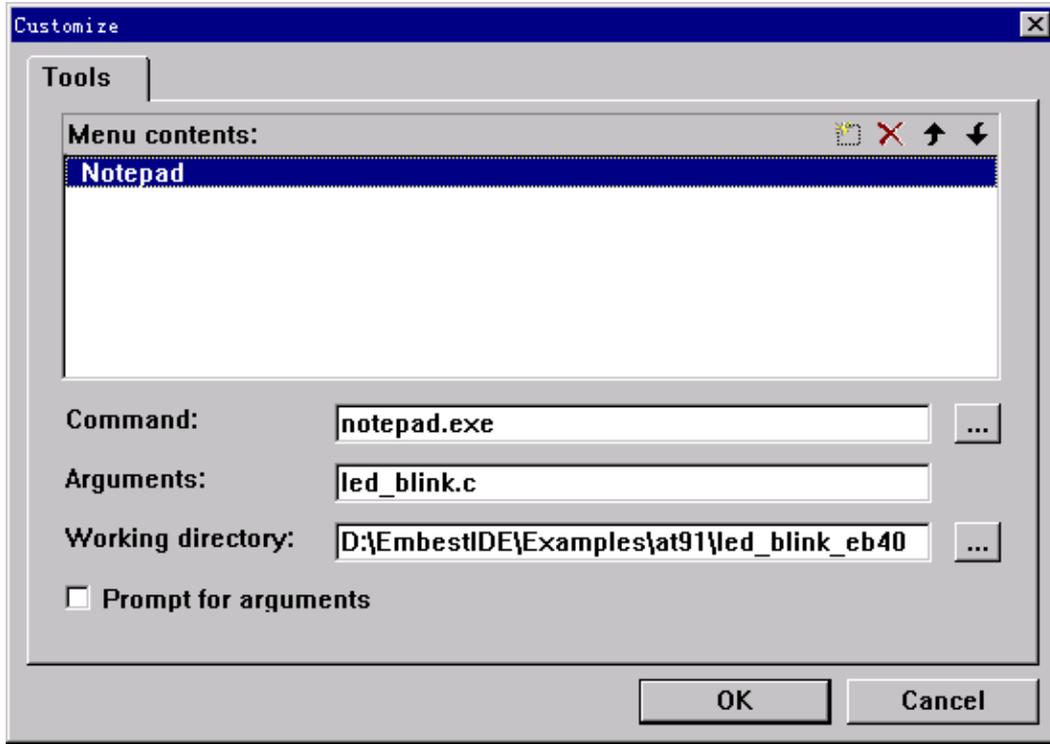


Figure 8-1 Tools Menu Customize Dialog Box

Menu Contents List Box lists commands you have added to the Tools menu. To add a command, click the New button above the Menu Contents list, type the text for the menu item in the box at the end of the Menu Contents list, and provide the necessary information in the boxes below. To modify a command, select it in the Menu Contents list and change the information specified in the boxes below. To delete a command, select it and click the delete button above the Menu Contents list. To move a command up or down on the Tools menu, select the command in the Menu Contents list and click one of the arrows above list.

Command Edit Box displays the path and filename of the tool currently selected in the Menu Contents list. Click the button at the right of the edit box opens a dialog box where you can browse and select a file.

Arguments Edit Box Specifies additional arguments for the tool each time you start it. You can use several macros in custom menu commands, see 8.4 *Use Macros* for explanations of these macros.

Working directory Edit Box specifies where (in what directory) to run the custom command. You can edit the directory name in place, or click the button at the right of this field to bring up a directory browser where you can search for the right directory.

Prompt for arguments Checked Box, When selected, displays a dialog box which prompts for command-line arguments each time you run the tool. You can specify additional arguments for each particular instance of the tool.

OK Button, Applies your changes to the Tools menu.

CANCEL Button, Discards your changes without modifying the Tools menu.

8.3 IDE Options

8.3.1 Editor Preferences

Select Tools > Options... menu, then click the Editor tab to adapt the editor to your preferences. The Editor page is shown in Figure 8-2:

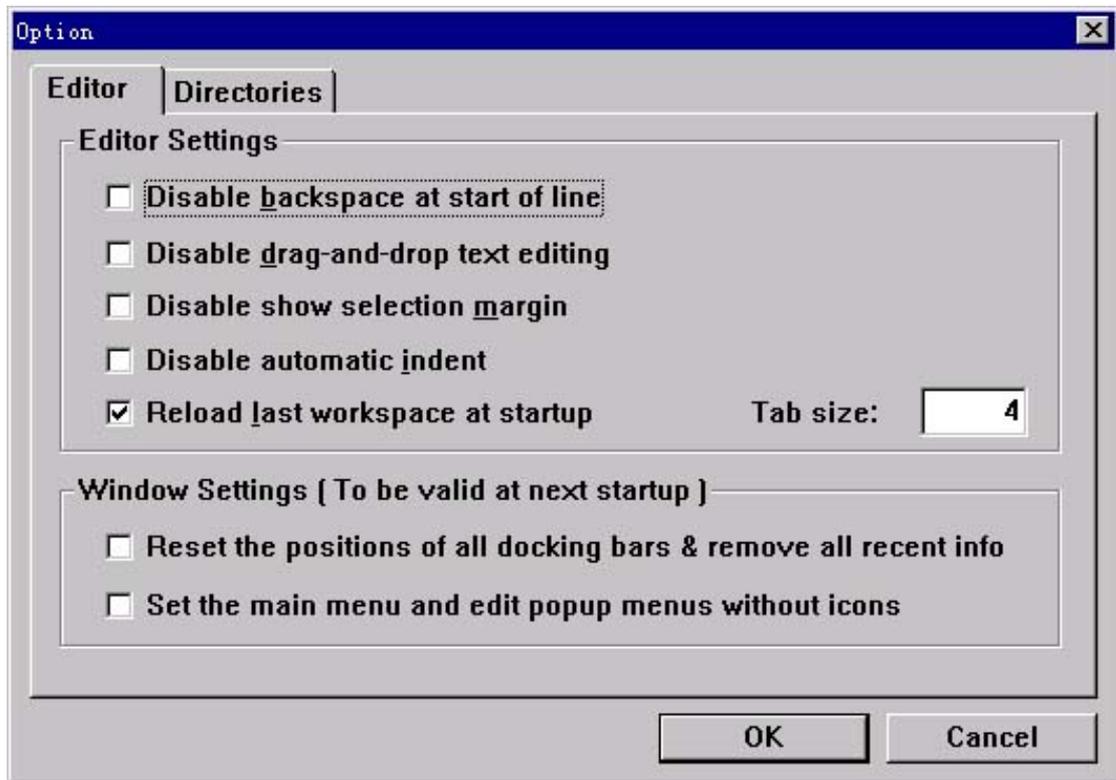


Figure 8-2 Editor Page

The following choices are available on the **Editor Settings**:

Disable backspace at start of line, prevents joining of lines by using the BACKSPACE key.

Disable drag-and-drop text editing, select this checkbox to disable drag-and-drop text editing so you can not move or copy selected text with the mouse.

Disable show selection margin, select this checkbox to disable display a margin to the left of each line of text. This margin display information about source lines, including breakpoints, instruction points, and tag pointers.

Disable automatic indent, select this checkbox to disable indent source code automatically.

Reload last workspace at startup, when selected, automatically loads the workspace you last worked on.

Tab Size, provides a place for you to specify the number of space characters that equal one tab character. The default is four space characters.

The following choices are available on the **Window Settings**:

Reset the positions of all docking bars & remove all recent info, all docking windows display at same position and size as you last worked on.

When selected, all docking windows display with the default position provided by the IDE and all history information reserved by IDE will be cleared.

Set the main menu and edit popup menus without icons, when selected, disable display icons corresponding with menu items. This function suits some operation system which limited GUI resources, such as WINDOWS 98.

8.3.2 Directory Options

Select Tools > Options... menu, then click the Directories tab. The Directories page is shown in Figure 8-3:

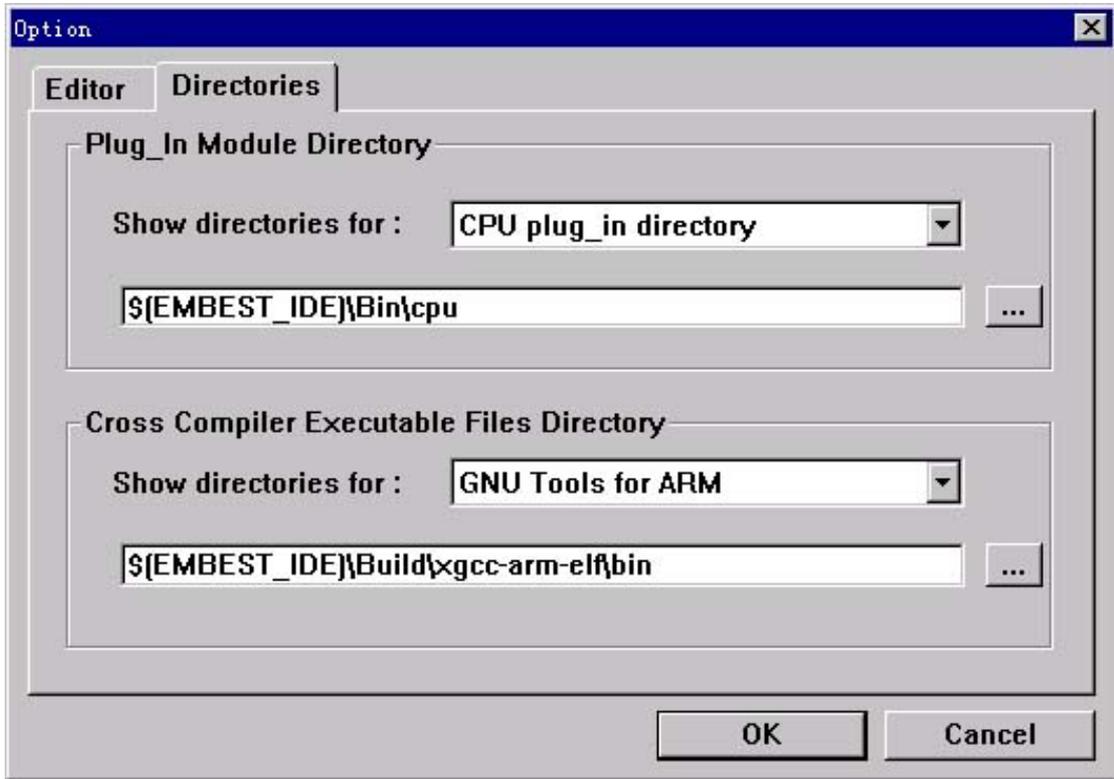


Figure 8-3 Directories Page

Plug_in Module Directory:

Table 8-2 is a description of the Embest IDE Plug_in Module Directories.

Table 8-2 Directory Description

| Function | Default Directory |
|----------------------|---------------------------|
| CPU Support | \$(EMBEST_IDE)\bin\cpu |
| Debug Device Support | \$(EMBEST_IDE)\bin\device |
| File Support | \$(EMBEST_IDE)\bin\file |
| Build Tools Support | \$(EMBEST_IDE)\bin\build |
| Driver | \$(EMBEST_IDE)\bin\driver |

In the Show Directories For list box, user select the type of module for the directory, then the edit box below the list box display directory for modules. You can edit the directory or click the button at the right of the edit box opens a dialog box where you can browse and select a directory.

Note: Plug_in Module Directories Use Default, general user do not change it.

Build Executable Files Directory:

IDE support some different compiler at the same time. You can setting **Build Executable Files Directory** corresponding with compiler. In the Show Directories For list box, user select compiler, then directory corresponding with compiler is displayed in the edit box. User can edit the directory or click the button at the right of the edit box opens a dialog box where you can browse and select a directory.

Note: The default directory specilized by IDE, general user do not change it.

8.4 Use Macro

You can use argument macros to specify arguments for a Tools menu command. Embest IDE provides the argument macros shown in the following table:

Table 8-3 Macro Description

| No | Macro Name | Expands to a string containing | Examples |
|----|-----------------------|---|------------------------|
| 1 | \$(DOWNLOAD_PATHFILE) | The directory and Name of symbol file(.elf) | D:\test\debug\test.elf |
| 2 | \$(EMBEST_IDE) | Installation directory of The Embest IDE. | D:\EmbestIDE |
| 3 | \$(LINK_FILE) | Name of the output file(.elf) | test.elf |
| 4 | \$(LINK_DIR) | The directory of the output file(.elf) | D:\test\debug\ |
| 5 | \$(LINK_PATHFILE) | The directory and Name of output file(.elf) | D:\test\debug\test.elf |
| 6 | \$(PROJECT_PATH) | The directory of the current project. | D:\test\ |
| 7 | \$(PROJECT_NAME) | The name of the current project. | test |
| 8 | \$(SYMBOL_PATHFILE) | The directory and Name of Symbol file(.elf) | D:\test\debug\test.elf |

To click menu Tools>Options, there is the template to use Macro.

Customer Service

Get support on demand. Connect Customer Service for more information on how to use the Embest's products.

- **Web Site**

Get the latest information and docs about Embest's products from the web site: <http://www.embedinfo.com>

You may have noticed some trouble issues at the support forums. In the meantime, you can get help by subscribing to the following forum:

<http://www.embedinfo.com/cforum/login.asp>

- **E-Mail**

If you have any question, comments, feedback or suggestions as to how our products could be improved, let us know at support@embedinfo.com

- **Telephone Number**

You can also call **86-755-25635626** with the extension to the Customer Service Center.

- **Fax Number**

Our fax number is **86-755-25616057**.

Appendix A Hardware Reference of Embest JTAG Emulator

Embest JTAG Emulator contains two types of product: Standard JTAG Emulator (Embest Emulator for ARM), and Enhanced JTAG Emulator (Embest PowerICE for ARM).

Standard Emulator (Embest Emulator for ARM) was the standard JTAG emulator for development series of ARM core CPU, early product of Embest Info&Tech Co., LTD. It works at 25Kbyte per second by transmission, and capability stabilization.

Enhanced Emulator (Embest PowerICE for ARM) was the New-generation of JTAG emulator. It's feature power supply can be provided by internal or external input, and works at highest speed 120Kbyte per second by transmission.

Embest JTAG Emulator has a Parallel port connecting to the Computer's parallel port, and a JTAG interface connecting to the target system.

There are 3 LEDs on the panel, indicating the Emulator's working state.

Note: Cable connection must not hot swap!

Embest PowerICE for ARM

JTAG Interface Connections

A standard male-to-female 25-way parallel cable connects the Embest PowerICE for ARM to the PC's parallel port. The connection to the target board is made by a 20-way (or 14 - way) female IDC header cable with all pins connected straight through (1-1, 2-2, ... 20-20). There are two types of IDC interface cable: 14pin and 20 pins. JTAG pin connections is described as figure A - 1 and A - 2.

| | | | |
|---------|----|----|-----|
| Vsupply | 1 | 2 | RES |
| RES | 3 | 4 | GND |
| TDI | 5 | 6 | GND |
| TMS | 7 | 8 | GND |
| TCK | 9 | 10 | GND |
| RES | 11 | 12 | GND |
| TDO | 13 | 14 | GND |
| nSRST | 15 | 16 | GND |
| RES | 17 | 18 | GND |
| RES | 19 | 20 | GND |

Figure A-1 20 Pin JTAG Connections

| | | | |
|---------|----|----|-----|
| Vsupply | 1 | 2 | RES |
| nSRST | 3 | 4 | GND |
| TDI | 5 | 6 | GND |
| TMS | 7 | 8 | GND |
| TCK | 9 | 10 | GND |
| TDO | 11 | 12 | GND |
| RES | 13 | 14 | GND |

Figure A-2 14 Pin JTAG Connections

Note: All GND pins should be connected to 0V on the target board.

The following table shows the JTAG pinouts.

| Signal | I/O | Description |
|---------|-------|---|
| Vsupply | Input | This is the supply voltage to Embest PowerICE for |

| | | |
|-------|--------|--|
| | | ARM. It draws its supply current from this pin via a step-up voltage convertor. This is normally fed by the target Vdd. Valid power supply voltage is form 2.7V to 5.5V. |
| GND | - | Ground. |
| TDI | Output | Test Data In signal from Embest PowerICE for ARM to the target JTAG port. It is recommended that this pin be pulled to a defined state. |
| TMS | Output | Test Mode signal from Embest PowerICE for ARM to the target JTAG port. This pin should be pulled up on the target so that the effect of any spurious TCKs when there is no connection is benign. |
| TCK | Output | Test Clock signal from Embest PowerICE for ARM to the target JTAG port. It is recommended that this pin be pulled to a defined state. |
| TDO | Input | Test Data Out from the target JTAG port to Embest PowerICE for ARM. |
| nSRST | Output | Open collector output from Embest PowerICE for ARM to the target system reset. This pin should be pulled up on the target to avoid unintentional resets when there is no connection. |
| RES | - | Reserved. |

Power Supply

Power is supplied to the Embest PowerICE for ARM via pin 1 of the 20-way (or 14-way) IDC connector. This is normally fed by the target Vdd. Valid power supply voltage is form 2.7V to 5.5V. Power of Embest PowerICE for ARM also can be supply by external input voltage valid 3V/5V. Connection jack of the external voltage input show as figure A-3 following:

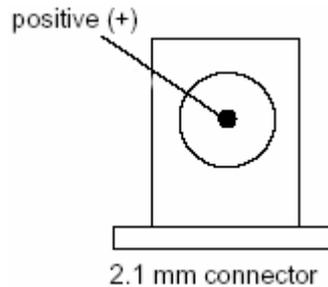


Figure A-3 connection jack of the external voltage input

Note:

- According to the way of voltage input, power supply switch of Embest PowerICE for ARM must place in the right position.
 - Embest PowerICE for ARM cannot work if power voltage out of range, even be badly damaged.
-

Target Interface Voltage Levels

The target interface voltage levels of Embest PowerICE for ARM depends on the input voltage levels. It is 3V/5V compatible. Normally, power supply by the external input voltage will give the output single voltage provided 3.3V.

LED Indicator

There are three LED in the panel of Embest PowerICE for ARM, labeled Power, Run, and Con.

LED Power: power indicator

LED Run: data indicator, indicate the data transmission between host pc and target CPU.

LED Con, connection indicator

Embest Emulator for ARM

JTAG Interface Connections

A standard male-to-female 25-way parallel cable connects the Embest Emulator for ARM to the PC's parallel port. The connection to the target board is made by a 20-way (or 14 - way) female IDC header cable with all pins connected straight through (1-1, 2-2, ... 20-20). There are two types of IDC interface cable: 14pin and 20 pins. JTAG pin connections is described as figure A - 1 and A - 2.

| | | | |
|---------|----|----|-----|
| Vsupply | 1 | 2 | RES |
| RES | 3 | 4 | GND |
| TDI | 5 | 6 | GND |
| TMS | 7 | 8 | GND |
| TCK | 9 | 10 | GND |
| RES | 11 | 12 | GND |
| TDO | 13 | 14 | GND |
| nSRST | 15 | 16 | GND |
| RES | 17 | 18 | GND |
| RES | 19 | 20 | GND |

Figure A-1 20 Pin JTAG Connections

| | | | |
|---------|----|----|-----|
| Vsupply | 1 | 2 | RES |
| nSRST | 3 | 4 | GND |
| TDI | 5 | 6 | GND |
| TMS | 7 | 8 | GND |
| TCK | 9 | 10 | GND |
| TDO | 11 | 12 | GND |
| RES | 13 | 14 | GND |

Figure A-2 14 Pin JTAG Connections

Note: All GND pins should be connected to 0V on the target board.

The following table shows the JTAG pinouts.

| Signal | I/O | Description |
|---------------|------------|--|
| Vsupply | Input | This is the supply voltage to Embest Emulator for ARM. It draws its supply current from this pin via a step-up voltage convertor. This is normally fed by the target Vdd. Valid power supply voltage is form 2.7V to 5.5V. |
| GND | - | Ground. |
| TDI | Output | Test Data In signal from Embest Emulator for ARM to the target JTAG port. It is recommended that this pin be pulled to a defined state. |
| TMS | Output | Test Mode signal from Embest Emulator for ARM to the target JTAG port. This pin should be pulled up on the target so that the effect of any spurious TCKs when there is no connection is benign. |
| TCK | Output | Test Clock signal from Embest Emulator for ARM to the target JTAG port. It is recommended that this pin be pulled to a defined state. |
| TDO | Input | Test Data Out from the target JTAG port to Embest Emulator for ARM. |
| nSRST | Output | Open collector output from Embest Emulator for ARM to the target system reset. This pin should be pulled up on the target to avoid unintentional resets when there is no connection. |
| RES | - | Reserved. |

Power Supply

Power is supplied to the Embest Emulator for ARM via pin 1 of the 20-way (or 14-way) IDC connector. This is normally fed by the target Vdd. Valid power supply voltage is form 2.7V to 5.5V.

Note: Emulator cannot work if power voltage out of range, even be badly damaged.

Target Interface Voltage Levels

The target interface voltage levels of Embest Emulator for ARM depends on the input voltage levels. It is 3V/5V compatible.

LED Indicator

There are three LED in the panel of Embest Emulator for ARM, labeled Power, Run, and Con.

LED Power: power indicator

LED Run: data indicator, indicate the data transmission between host pc and target CPU.

LED Con, connection indicator

Appendix B Debug Output Reference

Info Reference

| | | |
|-------------|---|---|
| 3001 | CPU was in debug state before connected, register values may be incorrect. | |
| | description | CPU was in debug state before connecting to the host, you may get incorrect register values. |
| | cause and resolution | Causation: <ul style="list-style-type: none">■ IDE Disconnected with target CPU when it is in the debug state. Resolution: If you want to run the program in the target system, you should rectify the value of PC and related registers. If you are going to download new program to the target system, notice the values in the stack-related registers. |

| | | |
|-------------|--|---|
| 3002 | target running, cannot auto download. | |
| | description | Target is in running state, auto download command cannot be executed. |
| | cause and resolution | Causation: <ul style="list-style-type: none">■ If auto_download check box been checked in the project setting dialog, when connect to the target, IDE will check the target status. If the target is in debug state, IDE will execute the download command, else prompt this info message. Resolution: No Resolution. |

Warning Reference

2001 Breakpoint xx are not on valid lines, disabled.

description The breakpoint is invalid, and then it is forbidden.

cause and resolution Reason of warning:
 ■ Breakpoint is set at invalid line. User must reset the breakpoint at a valid line.

2002 invalid line, set breakpoint failed.

description Failed to set breakpoint because current line is not execute statement.

cause and resolution Reason of warning:
 ■ Current line is not execute statement. user must set a breakpoint at execute line.

2003 load symbol file failed.

description Failed loading the symbol file.

cause and resolution Reason of warning:
 ■ Symbol file dos not exist.
 ■ Symbol file format dose not supported by IDE.

Select correct symbol file format and recompile and then set the symbol file config at Project setting > Debug >general dialog.

2004 open memory map file failed.

description Failed to open memory map file.

cause and resolution Reason of warning:
 ■ Memory map file does not exist.
 ■ Memory map file damaged, cannot be

opened.

2005 read program counter failed.

description Failed to read program counter.

cause and Reason of warning:

resolution

- Communication failed between Emulator and target system.
- Target program exception.

Reset the target CPU and/or reconnect to it.

2006 register doesn't exist.

description Register doesn't exist.

cause and Reason of warning:

resolution

- CPU module is incompatible with debug device module, please contact the provider.
-

2007 target running, all breakpoints disabled.

description Target is in running state, hence all of the breakpoint is forbidden.

cause and Reason of warning:

resolution

- Target is in running state, user must stop the target before enable all the breakpoint.
-

2008 target running, cannot toggle breakpoint.

description Target CPU is in running state, cannot toggle breakpoint.

cause and Reason of warning:

resolution

- Target CPU is in running state, you need stop the CPU before toggle breakpoint.
-

2009 too many breakpoints.

| | | |
|-------------|---|--|
| | description | Breakpoints' amount out of range. |
| | cause and resolution | Reason of warning: (Embest JTAG emulator for arm7 V2001): <ul style="list-style-type: none"> ■ More than two hardware breakpoints. ■ More then one hardware breakpoint and 255 software breakpoint. Delete at least one breakpoint before setting a new one. |
| 2010 | unable to compute express or variable value. | |
| | description | Failed to compute express or variable value. |
| | cause and resolution | Reason of warning: <ul style="list-style-type: none"> ■ Variable does not exist. ■ Express invalid. |
| 2011 | unable to locate address. | |
| | description | Unable to locate the instruction address corresponding with current line. |
| | cause and resolution | Reason of warning: <ul style="list-style-type: none"> ■ Symbol file is not matching with execute file. ■ Symbol file format dose not supported by IDE. User need rebuild the project and download again. |
| 2012 | unable to locate source file. | |
| | description | Unable to locate the source file corresponding with current instruction. |
| | cause and resolution | Reason of warning: <ul style="list-style-type: none"> ■ Symbol file is not matching with execute file. ■ Symbol file format is not supported by IDE. |

User need rebuild the project and download again.

2013 workspace does not exist.

description Workspace does not exist.

cause and resolution Reason of warning:

- Current command need a workspace be opened. User must create a workspace and project.

2014 write program counter failed.

description Failed to write program counter.

cause and resolution Reason of warning:

- Communication failed between Emulator and target system.
- Target program exception.

Reset the target CPU and/or reconnect to it.

Error Reference

1001 Can not find environment variable 'embest_ide'.

description Failed to find environment variable.

cause and resolution Reason of error:

- environment variable 'embest_ide' does not set in operation system. User must set environment variable
EMBESE_IDE=[Embtest IDE's setup dir].

1002 Can not find register group, maybe device and cpu module your selected are not compatible.

description Failed to find register group.

cause and resolution Reason of error:

- CPU module is not compatible with device module. Please connect to the provider.

1003 can not find download file ...

description Failed to find the specified download file.

cause and resolution Reason of error:

- download file do not exist. User need rebuild the project.
- download file path is not correct.

1004 can not read register.

description Read register error.

cause and resolution Reason of error:

- Communication failed between Emulator and target system.
- Target program exception.

Reset the target CPU and/or reconnect to it.

1005 can not initialize CPU module.

| | |
|----------------------|---|
| description | CPU module initialize failed. |
| cause and resolution | Another error statement tell you the reason of error if there are two error statement at same time, otherwise: <ul style="list-style-type: none"> ■ CPU module is not compatible with current IDE version. |

1006 can not initialize emulator module.

| | |
|----------------------|--|
| description | Failed to initialize emulator module. |
| cause and resolution | Another error statement tell you the reason of error if there are two error statement at same time, otherwise: <ul style="list-style-type: none"> ■ Emulator module is not compatible with current IDE version. ■ IDE do not support current emulator. |

1007 can not initialize file module.

| | |
|----------------------|--|
| description | Failed to initialize file module. |
| cause and resolution | Another error statement tell you the reason of error if there are two error statement at same time, otherwise: <ul style="list-style-type: none"> ■ file module is not compatible with current IDE version. |

1008 can not open ide.ini file or specified item does not exist.

| | |
|----------------------|--|
| description | De.ini cannot be opened or some error occur when open the file. |
| cause and resolution | Reason of error: <ul style="list-style-type: none"> ■ ide.ini do not exist. This file should in the embest_ide dir. ■ ide.ini file contain errors. User needs resetup EmbestIDE. |

1009 cursor are not positioned on valid lines.

| | |
|-------------|---|
| description | The line that contains cursor is invalid to |
|-------------|---|

| | | |
|-------------|--|--|
| | | the current command. |
| | cause and resolution | Reason of error: <ul style="list-style-type: none"> ■ move the cursor to a valid line. |
| 1010 | disassemble non-existent memory. | |
| | description | Disassemble non-existent memory space. |
| | cause and resolution | Reason of error: <ul style="list-style-type: none"> ■ memory config file does not set up all of the memory space. ■ CPU was in debug state before connecting to the host, so, IDE may get incorrect register values. PC point to a non-existent memory. This error can be ignored. ■ User program error. |
| 1011 | download address illegal, please modify download address. | |
| | Description | Download address illegal. |
| | cause and resolution | Reason of error: <ul style="list-style-type: none"> ■ project setting dialog specifies a download file name, but not specifies the download address. User need specifies a download address. ■ download address is not a valid value. User must modify the address to a valid value. |
| 1012 | emulator not found, please check power and hardware connection. | |
| | description | Failed to find the emulator. |
| | cause and resolution | Reason of error: <ul style="list-style-type: none"> ■ Emulator does not connect to the host computer. ■ Emulator do not powered on. ■ Communication port config error or |

damaged.

1013 failed to convert file from ELF format to BIN format.

description Failed convert file format from ELF to BIN.

cause and resolution Reason of error:

- file format is incorrect.
- Current file format is not supported by IDE.

User must rebuild the project.

1014 find symbol failed.

description Failed to find specified symbol.

cause and resolution Reason of error:

- No symbol file is specified.
- Specified symbol is not found in source file.

1015 get target status failed.

description Failed to get target status.

cause and resolution Reason of error:

- Communication failed between Emulator and target system.
- Target program exception.

Reset the target CPU and/or reconnect to it.

1016 internal error, please contact supplier.

description IDE internal error.

cause and resolution Reason of error:

- IDE internal error. Please contact the provider.

1017 invalid command.

description Invalid command.

| | | |
|-------------|------------------------------------|--|
| | cause and resolution | Reason of error: <ul style="list-style-type: none"> ■ execute a user customized command, but this command is invalid. User need modify the custom command. |
| 1018 | load script file error. | |
| | description | Failed to load script file. |
| | cause and resolution | Reason of error: <ul style="list-style-type: none"> ■ Specified script file do not exist. User need check the filename and path of the specified script file. ■ script file error, check and modify it. |
| 1019 | out of memory. | |
| | description | Failed to allocate memory space. |
| | cause and resolution | Reason of error: <ul style="list-style-type: none"> ■ Failed to allocate system memory space. User need close other application program or restart host computer. ■ IDE internal error. Please contact the provider. |
| 1020 | project doesn't exist. | |
| | description | Project does not exist. |
| | cause and resolution | Reason of error: <ul style="list-style-type: none"> ■ No workspace has been opened. User need open a workspace. ■ No active project. User need create a project and activate it. |
| 1021 | read register group failed. | |
| | description | Failed to read register group. |
| | cause and resolution | Reason of error: <ul style="list-style-type: none"> ■ Communication failed between Emulator and target system. |

-
- Target program exception.

Reset the target CPU and/or reconnect to it.

1022 reset target failed.

description Failed to reset target CPU.

cause and resolution Reason of error:

- hardware error or target CPU is not supported by current version of IDE. Check hardware interface.

1023 run target failed.

description Failed to run target program.

cause and resolution Reason of error:

- IDE get wrong Target state.
- Communication failed between Emulator and target system.
- Target program exception.

Reset the target CPU and/or reconnect to it.

1024 step failed, maybe symbol file incorrect.

description Source file step failed, possibly because symbol file incorrect.

cause and resolution Reason of error:

- Symbol file is not matching with execute file. User need rebuild the project and download again.

1025 step failed.

description Step failed.

cause and resolution Reason of error:

- Communication failed between Emulator and target system.
- Target program exception.

Reset the target CPU and/or reconnect to it.

1026 stop target failed.

description IDE cannot stop target CPU.

cause and resolution Reason of error:

- Communication failed between Emulator and target system.
- Target program exception.

Reset the target CPU and/or reconnect to it.

1027 target CPU not found.

description IDE cannot find target CPU.

cause and resolution Reason of error:

- Target board JTAG interface do not match with emulator.
- Target CPU do not work.
- Target CPU do not support JTAG debug mode.
- Emulator power level does not match with CPU.

1028 unable to load CPU module.

description Failed to load CPU module.

cause and resolution Reason of error:

- Specified emulator module does not exist. Reset the project and select corresponding emulator module.
- CPU module path error in the IDE tools-option dialog. Select Tools > Option menu, set CPU module path to `$(EMBEST_IDE)\bin\CPU`.

1029 unable to load emulator module.

description Failed to load emulator module.

cause and resolution Reason of error:

- Specified emulator module does not

exist. Reset the project and select corresponding emulator module.

- emulator module path error in the IDE tools-option dialog. Select Tools > Option menu, set emulator module path to \$(EMBEST_IDE)\bin\device.
-

1030 unable to load file module.

description Failed to load file module.

cause and Reason of error:
resolution

- Specified file module does not exist. Check the sub dir \bin\file in the IDE setup dir.
 - File module path error in the IDE tools-option dialog. Select Tools > Option menu, set file module path to \$(EMBEST_IDE)\bin\file.
-

1031 unable to load portcall.dll.

description Failed to load portcall.dll.

cause and ■ portcall.dll does not exist. Check the sub
resolution dir \bin\driver in the IDE setup dir.

- Driver module path error in the IDE tools-option dialog. Select Tools > Option menu, set Driver module path to \$(EMBEST_IDE)\bin\driver.
-

Appendix C Debug Command List

General Option of Debug Command

[...] Optional items. Items out of [] must be present.

Option can be used by all command.

-? Display the help information of the command.

Debug Command List

BKPTCLEAR – clear breakpoint

syntax: bkptclear [breakpoint ID]

description: Clear one or all the point
n:

Parameter: breakpoint ID An integer number which identify a certain breakpoint

option: none

example: bkptclear 1 Clear the breakpoint which ID is 1

bkptclear Clear all the breakpoint

BKPTDATA – set an data breakpoint

syntax: bkptdata option address

description: Set an data breakpoint at the specified memory locations
n:

Parameter: address Memory address

option: W breakpoint is valid when write memory

R breakpoint is valid when read memory

example: bkptdata -wr 0x7f4dfc Set a data breakpoint at address
c 0x7f4dfc, it is valid when read or
write this address.

BKPTINST – Sets an instruction breakpoint

syntax: bkptinst address

description: Sets an instruction breakpoint at the specified memory locations
n:

parameter: address The instruction address.

option: none

example: bkptinst 0x1024 Set a instruction breakpoint at address 0x1024

BKPTLIST –List all breakpoints

syntax: bkptlist

description: List all installed breakpoints.

n:

parameter: none

option: none

example: bkptlist List all installed breakpoints.

DISASM –Disassemble the target code

syntax: disassemble addr [line_num]

description: The Disasm command disassembles the target code beginning at a specified address. The Disasm command disassembles ten lines as a default, or you can include an optional [line_num] parameter

n:

parameter: addr address in target memory to begin disassembling.

line_num the number of instructions you wish to disassemble. The default is 10 lines.

option: None

example: Disassemble 100 8 disassemble 100 lines of code located at address 8.

DOWNLOAD –Download file

syntax: download [Option] filename address

description: Download file to the specified address.

n:

parameter: filename the specified download file.

address the specified address

option: v download verify

example: download d:\demo\ram.bin 0x2000000 Download d:\demo\ram.bin to the memory address at 0x2000000

| | |
|--------------------------------------|---|
| download -v d:\demo\ram.bin 0x200000 | Download d:\demo\ram.bin to the memory address at 0x2000000, verify when downloading. |
|--------------------------------------|---|

GO – Execute target program

syntax: go

description: Execute target program from current program counter

Parameter: none

option: none

example: Go

HELP – display help information

syntax: help [command name]

description: Display the help information of the specified command or brief information of all the command.

parameter: Command name Debug command name

option: none

example: help Display the brief information of all the command.

 help stop Display the help information of the command stop.

MEMREAD –Display the content of memory

syntax: memread address length

description: Displays the contents of the memory location requested. It accesses the memory in word format default.

parameter: address memory location

 length the length of memory to be read

option: h specifies access the memory in half word format.

b specifies access the memory in byte format.

example: memread 0x1000 0x200 Read 0x0200 words from 0x1000
200

MEMWRITE –Write to memory

syntax: memwrite [option] address value

description: Write value to the memory location requested. It accesses the memory in word format default.

parameter: address memory location
s

value Specifies value to write.

option: -h Specifies access the memory in half word format.

-b Specifies access the memory in byte format.

-e Write memory by Big endian mode

example: Memwrite 0x1000 0x5A Write 0x5a to 0x1000
A

memwrite -e 0x20000 0x55443322 Equal to memwrite 0x2000000 0
00 0x22334455 x55443322

REFRESH – refresh all windows

syntax: refresh

description: refresh all windows include register, memory, stack, watch, global/local

parameter: none

option: none

example: refresh

REGLIST –Display all registers

syntax: reglist

description: displays the properties of all of the processor's registers

parameter: none

option: none

example: reglist

REGREAD –display registers

Syntax: regread [option] register group name or register name

description: displays the contents of a particular register or registers.

Parameter: register group name or register name
Specifies register group name or register name

option: -g read register group

example: regread pc display PC register

regread -g user display all registers belong to 'User' group

REGWRITE – set register

syntax: Regwrite register name value

description: Set register

parameter: register name
Specifies register name

value The value to write

option: none

example: regwrite pc 0x3840 Set PC with the value 0x3840

RESET –Reset the target

syntax: reset
description: Reset the target device
parameter: none
option: none
example: reset

SCRIPT –Executes command script file

syntax: script filename
description: Executes command script file
parameter: filename Specifies script filename
option: none
example: script d:\demo\cmd.cs Executes command script file d:\demo\cmd.cs

STEP –Executes one statement or instruction

syntax: step
description: single-stepping begins at the address contained in the program counter.
parameter: none
option: none
example: step step one instruction

STOP –Stop the target

syntax: stop
description: Stop the target
parameter: none

parameter: none

option: none

example: stop

SYMBOL –Load symbol file

syntax: symbol [symbol filename]

description: Load symbol file

parameter: symbol filename Specifies symbol file

option: none

example: symbol d:\demo\ram.elf Load ram.elf symbol file in d:\demo

Appendix D Memory Map File

Description

By default, Embest IDE assumes that the entire address space is mapped to standard RAM, so IDE can read or write any memory address. In some cases it will cause exceptions, in which case you can manually create a memory map file. You should create your own memory map in the following situations:

- Your target has read-only memory areas or non-existent memory areas that should cause a bus fault when accessed.
- Your applications access non-standard memory locations.

A memory map is created in a *.MAP file which Embest IDE automatically executes when connect emulator.

Format

In memory map file, each line describe one memory block except for the line begin with '#' which is comment line. One line is consisting of nine parts and each part separate with blank space. The format of each line is:

name start size read-write bus-width access-size read-times write-times
burst-times

| ITEM | TYPE | DESCRIPTION |
|-------------|-------------|--|
| Name | string | A single word that you can use to identify the memory region. You can use any name. To ease readability of the memory access statistics, give a descriptive name such as SRAM, DRAM, or EPROM. |
| Start | hexadecimal | The start address of the memory region. |
| Size | hexadecimal | The size of the memory region. |
| Read-Write | string | The property of read-write, R for read, W for write. |
| Bus-Width | decimal | The width of the data bus in bytes (that is, 1 for an 8-bit bus, 2 for a 16-bit bus, or 4 for a 32-bit bus). |
| Access-Size | decimal | The width when access memory (that is, 1 for 8-bit, 2 for 16-bit, or 4 for 32-bit). |
| Read-Times | nanoseconds | The nonsequential and sequential read times. |
| Write-Times | nanoseconds | The nonsequential and sequential write times. |
| Burst-Times | nanoseconds | The nonsequential and sequential burst times. |

Note: The beginning four items is necessary, and the others which not concerned can be substituted by symbol '-'.

Example

A typical memory map looks like this:

| #Name | Start | Size | Attribute | notWorry |
|-----------------------|-----------------|----------------|------------------|-----------------|
| INTERNRAM | 10000000 | 2000 | RW | ----- |
| COREINTERNALIO | 78000000 | 8000000 | RW | ----- |
| STANDARDAPBIO | B0000000 | 8000000 | RW | ----- |
| COREAPBIO | B8000000 | 8000000 | RW | ----- |
| EXTERNDRAM | C0000000 | 800000 | RW | ----- |
| EXTERNSRAM | C8000000 | 100000 | RW | ----- |
| FLASH | C8100000 | 200000 | R | ----- |
| EXTERNSRAM | D0000000 | 80000 | RW | ----- |
| EXTERNIO | F0000000 | 8000000 | RW | ----- |

NotWorry contents of: **BusWidth AccessSize ReadWait WriteWait BurstWait.**

Appendix E Command Script Reference

Description

Command Script File is a text formatting file, is used to auto execute several commands continuously, the contents of the file consist of command one by one.

When debug software or between connect target system, often need execute several fixed command lists, for instance: after connect target system, need execute thereafter steps: stop target CPU—mask interrupt register—set external memory—remap memory or download executable file etc. If need input these command after connect target system every time, do you feel boring? Just write a command script file, EmbestIDE will auto execute these commands.

The commands that used in debug command window also can be used in script file. Debug command and its detail reference please refer to the sect [`Debug Command Lists`](#).

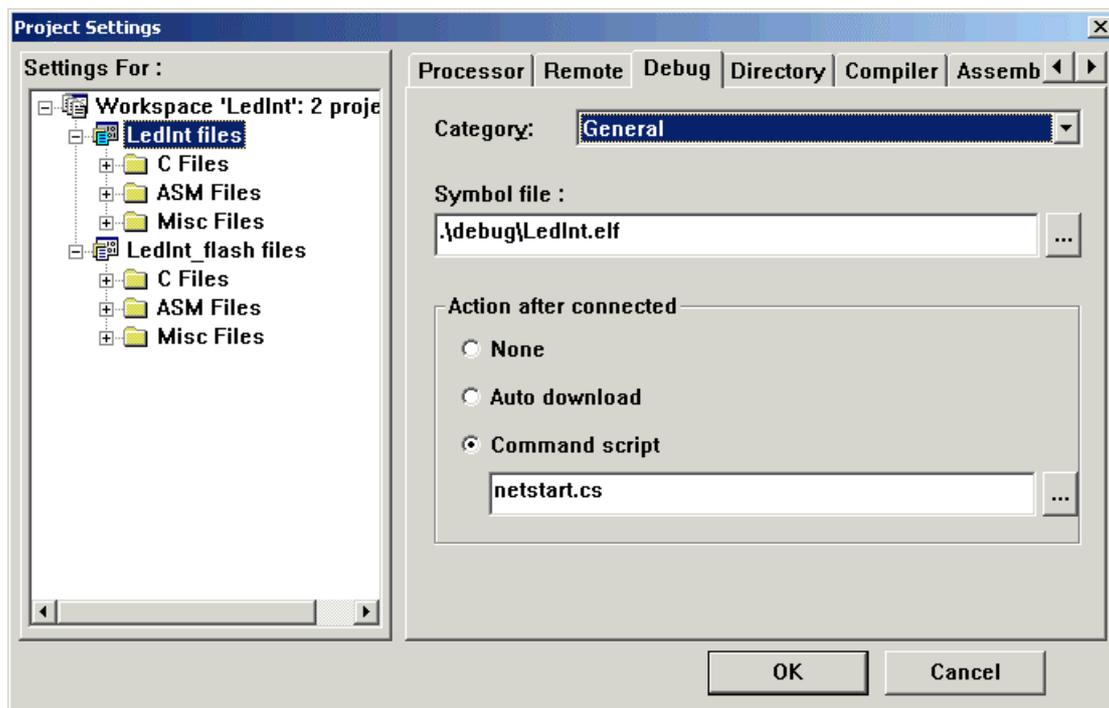
Executive

Command Script File has two executive ways:

1. In Command Input Window, input:

script <command script filename>

2. Designate in project setting dialog. When input appointed script filename, the script file will be auto executed after EmbestIDE successful connect target system.



Example

Example one: Atmel Eb40, command script file that will be auto executed after Embest IDE successful connect target system.

```
; stop target CPU  
stop  
; config memory  
memwrite 0xffe00000 0x01002535  
memwrite 0xffe00004 0x02002121  
memwrite 0xffe00024 0x06  
; remap memory  
memwrite 0xffe00020 0x01  
refresh  
download -v D:\Demo\armdemo\debug\led.elf 0x2000000  
; end
```

Example two: Samsung SNDS100, command script file that will be auto executed after Embest IDE successful connect target system.

```
; stop target CPU  
stop  
; mask interrupt  
memwrite 0x3ff4008 0xffffffff  
; config system  
memwrite 0x3ff0000 0x90FFFF83  
; set data-bus width  
memwrite 0x3ff3010 0xfaffff0f  
; config external FLASH  
memwrite 0x3ff3014 0x60000412  
memwrite 0x3ff3018 0x40800414  
memwrite 0x3ff301C 0x40000516  
memwrite 0x3ff3020 0x20800518  
memwrite 0x3ff3024 0x4000061a  
memwrite 0x3ff3028 0x4080061c  
; config external DRAM  
memwrite 0x3ff302C 0x80030004  
memwrite 0x3ff3030 0x80010108  
memwrite 0x3ff3034 0x8001020c  
memwrite 0x3ff3038 0x80010310  
memwrite 0x3ff303C 0x608327ce  
; end
```

Example Three: Samsung S3C44B0X, command script file that will be auto executed after Embest IDE successful connect target system.

```

Stop      ;stop target CPU
Reset     ;reset CPU
Stop
;set the system Registers
memwrite 0x01D30000 0x00000000 ;WTCON
memwrite 0x01E0000C 0x07ffffff ;INTMSK
memwrite 0x01D8000C 0x00000fff ;LOCKTIME
memwrite 0x01C80000 0x11110101 ;BWSCON
memwrite 0x01C80004 0x00000600 ; BANKCON0
memwrite 0x01C80008 0x00007FFC ; BANKCON1
memwrite 0x01C8000C 0x00007FFC ; BANKCON2
memwrite 0x01C80010 0x00007FFC ; BANKCON3
memwrite 0x01C80014 0x00007FFC ; BANKCON4
memwrite 0x01C80018 0x00007FFC ; BANKCON5
memwrite 0x01C8001C 0x00018000 ; BANKCON6
memwrite 0x01C80020 0x00018000 ; BANKCON7
memwrite 0x01C80024 0x00860459 ;REFRESH
memwrite 0x01c80028 0x00000010 ;BANKSIZE
memwrite 0x01C8002C 0x00000020 ;MRSRB6
memwrite 0x01C80030 0x00000020 ;MRSRB7
;end

```

Appendix F Additional Software Tools of Embest IDE

As shown in fig. F-1, click Tools in main menu to open the tool menu of Embest IDE. The attached tools include: Elf to Bin, Disassemble all, Symbol table, Flash Programmer and SplitBin.



Fig. 1 Interface of tool menu

Description

Elf to Bin

The tool Elf to Bin is the generation tool of binary file format. This tool can be used to change the debug information Elf file, built and generated by IDE, into the binary file necessary for solidification of program.

The user selects the sub-menu Elf to Bin under Tools menu to generate the sub-directory Debug under program directory into a Bin file with same name as Elf file.

The user can also directly use command line style to complete the above procedure. The execution program elf2bin.exe corresponding by the command line is located at sub-directory Tools under installation directory Embest IDE. The detailed usage of command line can be obtained through execution of elf2bin in control table.

Disassemble all

Disassemble all is a disassembly tool. The user can use this tool to change the debug information file elf into the disassembly file including source file and debug symbol.

The user selects the sub-menu Disassemble all under Tools menu to generate the sub-directory Debug under project directory into a disassembly file with the file name as objdump.

The user can also directly use command line style to complete the above procedure. The execution program arm-elf-objdump.exe corresponding by the command line is located at sub-directory Builds/xgcc-arm-elf/bin under installation directory Embest IDE. The detailed usage of command line can be obtained through execution of arm-elf-objdump in control table.

Symbol table

Symbol table is the generation tool of debug symbol file. The user can use this tool to generate the symbol table of corresponding project debug information. This symbol table mainly records the entrance addresses of various functional symbols.

The user selects the sub-menu Symbol table under Tools menu to generate the sub-directory Debug under project directory into a debug symbol file with file name as objdump.

The user can also directly use command line style to complete the above procedure. The execution program arm-elf-objdump.exe corresponding by the command line is located at sub-directory Builds/xgcc-arm-elf/bin under installation directory Embest IDE. The detailed usage of command line can be obtained through execution of arm-elf-objdump in control table.

Flash Programmer

The tool Flash Programmer solidifies the binary file that the user finally generates onto the FLASH chip of circuit board of the user, support to use the programming of FLASH chip in the system developed with ARM series processors, especially suitable to the user selecting component displacement FLASH. After open, the software interface is shown in fig. F-2. For the detailed introduction, refer to Embest Online Flash Programmer User's Manual.

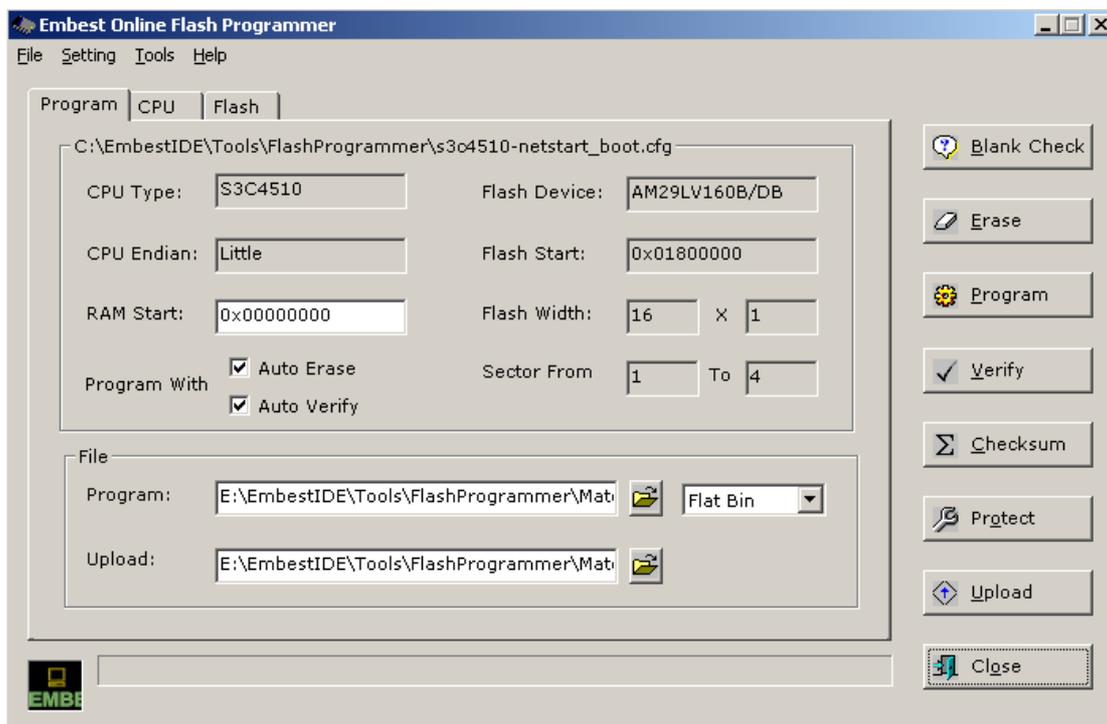
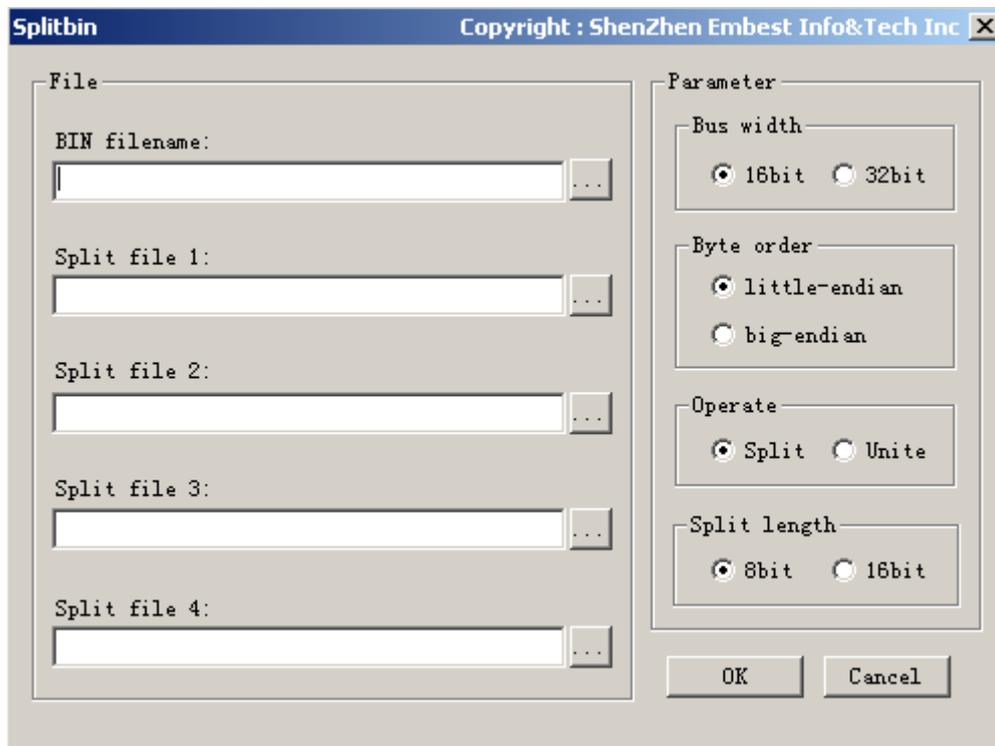


Fig. F-2 Software interface of Flash Programmer

SplitBin

Click SplitBin under Tools menu to open the SplitBin tool as shown in following fig. F-3. It can split or unite Bin file according to data width and coding manner.



F-3 Split tool interface of Bin file

Parameter>>Bus width: Select the data width of file to be split, 16bit or 32bit;

Parameter>>Byte order: Select coding manner of data, little Endian or big Endian;

Parameter>>Operate: Select whether the function of tool is split or unite;

Parameter>>Split length: Select data width of file after split;

File>>BIN filename: Select the routine and name of Bin file to be split or united;

Split file 1-4: If in split manner, it will automatically generate 2 or 4 split files according to data width selected in the right before and after split, for example, Bus width selects 16bit, Split length selects 8bit, and it will generate two files Split file 1 and Split file 2 after split; if in unite manner, select the route and name of Bin file to be united.

Appendix G Common questions

Question 1 About gccmain

Question:

It appears the following error prompt while connection, why?

```
.\debug\main.o: In function `main':
```

```
.\debug\main.o(.text+0xc): undefined reference to `__gccmain'
```

Answer:

While gcc builder is building and connecting source file, it will automatically connect main function to gccmain function internally provided in gcc, that is, if the programmer defines main function, the execution order of program is:

```
main()
{
    gccmain() ;
    Code compiled by customer
}
```

This function is provided in the base libgcc.a, and it must connect libg.a while connecting this base.

There are three measures for solving connection errors:

1) Settings in Embest IDE

Select the menu Project> Settings, open project configuration box, select the linker page and select the option add library searching path in drop-down box, increase the path:

```
..\..\..\build\xgcc-arm-elf\arm-elf\lib\arm-inter
```

```
..\..\..\build\xgcc-arm-elf\lib\gcc-lib\arm-elf\3.0.2\arm-inter
```

The above path is the path relative to Embest IDE routine, and the user can set absolute routine:

Select the linker page, select the drop-down option Include Object And Library Modules and add the base:

```
-lc
```

-lgcc

When this measure is linked, the program of user will be larger.

2) Write a gccmain assembly function or C language function by yourself

Prepare assembly function

```
                .global    __gccmain
__gccmain:
                mov     pc, lr
```

Prepare C language function

```
void __gccmain()
{
}
```

Re-link the program building:

3) Use `__main` or directly use `__gccmain` () as own function entrance;

Question 2 Connection errors of emulator

Question:

In general, what reasons will cause the connection emulator to appear the following error messages?

“target cpu not found”, “stop target failed” or “run target failed”

Answer:

The connection errors of emulator are generally caused due to following reasons:

- 1) The connection of JTAG interface circuit is not in accordance with the regulations of IEEE1149.1-1990, for example TDI, TCK, TMS and other signals are not connected pull-up resistance; nSRST shall be connected system reset instead of reset of JTAG. In addition, the JTAG interfaces have been correspondingly treated in the chips of some processors. The user should read the manual carefully.
- 2) Some chips of processors have appointed pins to control the operation of processor, they should be connected according to appointed style or otherwise it may cause CPU not to stop properly, for example nwait signal of ARM processors. The user should pay attention to checking whether nTRI or nWAIT is pulled up, whether the reset signal connection is correct, whether clock signal is proper, and referring to datasheet;
- 3) The emulator cannot obtain power from target board, for example the power of target board can only meet the requirements of target board, or JTAG interface has not leading power;
- 4) JTAG interface switch in the side of emulator has incorrect setting. The user shall correctly set the switch according to that the JTAG port in target board has 14 pins or 20 pins;
- 5) Project setting in IDE environment is not correct, the drop-down box of Project>Settings>Remote>Remote device shall select jtagarm7 or jtagarm9 according to CPU of target board;
- 6) Some computers need to amend parallel interface setting mode.

For user, generally check according to the following steps:

- 1) Power. Include voltage and current of power source. Embest Emulator is compatible to 3.3V and 5V. Emulator itself needs the current about several decades of mA.
- 2) Reset signal. Observe whether reset signal is proper or not, whether it is pulled down.

- 3) Clock check. Check whether crystal oscillator work normally, whether the clock input and output of processor is normal.
- 4) NWAIT Signal. Pull up.
- 5) nTRI Signal. Pull up.
- 6) For some processors, there are some modes or control pin (for example JTAGSEL signal) that are required to connect appointed level (pull-up or pull-down) in order to support JTAG debugging interface. Please read the datasheet carefully.

Question 3 To link the useful libraries

Question:

How to link the useful libraries provided by GNU, and what modes do they support?

Answer:

-lm means that linker will connect standard mathematic function base libm.a

-lc means that linker will connect standard C function base libc.a

-lg means that linker will connect the support base of standard function base libg.a

-lgcc means that linker will connect the support base of GCC libgcc.a

While connection, the arrangement order of these bases is generally as the following: -lm -lc -lgcc -lg

The C base files supplied by Embest IDE for ARM mainly support the following modes:

ARM Little-Endian

ARM Little-Endian Interwork

ARM Big-Endian

ARM Big-Endian Interwork

Thumb Little-Endian

Thumb Little-Endian Interwork

Thumb Big-Endian

Thumb Big-Endian Interwork

The lists in which Newlib C exists is corresponding to the above modes, and they are as follows respectively:

`$(EMBEST_IDE)\build\xgcc-arm-elf\arm-elf\lib`

`\lib\arm-inter`

`\lib\arm-big`

`\lib\arm-inter-big`

- \lib\thumb
- \lib\thumb-inter
- \lib\thumb-big
- \lib\thumb-inter-big

The lists in which libgcc exists:

\$(EMBEST_IDE)\Build\xgcc-arm-elf\arm-elf\lib

- \lib\arm-big
- \lib\arm-inter
- \lib\arm-inter-big
- \lib\thumb
- \lib\thumb-big
- \lib\thumb-inter
- \lib\thumb-inter-big

The lists in which libg.a base exists:

\$(EMBEST_IDE)\build\xgcc-arm-elf\lib\gcc-lib\arm-elf\3.0.2\

- \3.0.2\arm-inter
- \3.0.2\arm-big
- \3.0.2\arm-inter-big
- \3.0.2\thumb
- \3.0.2\thumb-inter
- \3.0.2\thumb-big
- \3.0.2\thumb-inter-big

Question 4 How to download the program onto Flash ROM

Question:

How to download the program completed debugging in RAM onto Flash ROM?

Answer:

The user shall pay attention to or change the following two positions when downloading the program completed debugging in RAM on target board:

- 1) Startup program: in RAM, through startup program of debugging, it is not necessary to copy the data section from read-only area to readable and writable area. The initialization of hardware can be completed through IDE command script or the program download onto Flash ROM. Therefore, the program completed debugging in RAM is generally download onto Flash ROM after amending startup program, for example in AT91 routine, the startup file for debugging in RAM is `cstartup_ice.s`, the startup file used in Flash ROM is `cstartup_flash.s`, according to the routine organization style of AT91, the displacement of startup file can be completed through a assembler pre-definition.
- 2) Linker script: the program passing through debugging in RAM use such linker script that takes RAM area as program code section address, and data section generally directly follow the code section, while the linker script used in Flash ROM uses ROM area as code section address, data section is in RAM area, therefore it shall distribute and amend linker script according to the address of final target board.

There are two examples on how to amend the program completed debugging in RAM then it can be downloaded onto target board.

Example 1: the program `Led_blink_EB40`, which lights LED in AT91EB40 evaluation board

- 1) Click Project menu>Settings, open project configuration box, change "AT91_DEBUG_ICE=1" as "AT91_DEBUG_NONE=1" in Assemble page>Predefines options.

2) Amend linker script file, and the linker script files before and after amendment are shown in following figure:

| | |
|---|---|
| <pre> 1 SECTIONS 2 { 3 * . = 0x02000000; 4 .text : { *(.text) } 5 !> Image_RO_Limit = .; 6 !> 7 Image_RW_Base = .; 8 .data : { *(.data) } 9 <! .rodata : { *(.rodata) } 10 Image_ZI_Base = .; 11 .bss : { *(.bss) } 12 Image_ZI_Limit = .; 13 __bss_start__ = .; 14 __bss_end__ = .; 15 __EH_FRAME_BEGIN__ = .; 16 __EH_FRAME_END__ = .; 17 PROVIDE (__stack = .); 18 end = .; 19 __end = .; 20 .debug_info 0 : { *(.debug_info) } 21 .debug_line 0 : { *(.debug_line) } 22 .debug_abbrev 0 : { *(.debug_abbrev) } 23 .debug_frame 0 : { *(.debug_frame) } 24 } </pre> | <pre> 1 SECTIONS 2 { 3 . = 0x1000000; 4 .text : { *(.text) } 5 .rodata : { *(.rodata) } 6 Image_RO_Limit = .; 7 . = 0x02000000; 8 Image_RW_Base = .; 9 .data : { *(.data) } 10 11 Image_ZI_Base = .; 12 .bss : { *(.bss) } 13 Image_ZI_Limit = .; 14 __bss_start__ = .; 15 __bss_end__ = .; 16 __EH_FRAME_BEGIN__ = .; 17 __EH_FRAME_END__ = .; 18 PROVIDE (__stack = .); 19 end = .; 20 __end = .; 21 .debug_info 0 : { *(.debug_info) } 22 .debug_line 0 : { *(.debug_line) } 23 .debug_abbrev 0 : { *(.debug_abbrev) } 24 .debug_frame 0 : { *(.debug_frame) } 25 } </pre> |
| In RAM | In Flash |

Fig. G-1 The linker scripts that the program uses in RAM and Flash and their difference

Comparing with original script file, the code section and read-only data section are put in the position starting from 0x1000000 address, that is, ROM storage area of the system; and present the starting address (RAM address) of readable and writable data section.

When the above operation is completed, re-build the project, then click IDE menu Tools>Elf to Bin, and change elf file into binary format file (*.bin). Finally use Embest Flash Programmer to download bin file onto Flash ROM of target board.

Example 2: LedInt that NET-START evaluation board lights LED

- 1) Set ROM = 1 in pre-definition options of assembler, or directly add ".equ ROM 1" in init.s file.
- 2) In link file of linker, select flash.ld. This link file and startup file mutually complete the transport of data section initially downloaded onto Flash. It has the following difference with the script file debugged in RAM: the address of current 0X0 is flash storage area, while it is RAM storage area formerly; the current RAM area is in the position 0X0400000. The linker script files before and after amendment are shown in following figure:

| | |
|---|---|
| <pre> 1 SECTIONS 2 { 3 . = 0x00000000; 4 .text : { *(.text) } 5 !> Image_RO_Limit = .; 6 !> Image_RW_Base = .; 7 <! .rodata : { *(.rodata) } 8 .data : { *(.data) } 9 Image_ZI_Base = .; 10 .bss : { *(.bss) } 11 Image_ZI_Limit = .; 12 <! __bss_start__ = .; 13 <! __bss_end__ = .; 14 <! PROVIDE (__stack = .); 15 end = .; 16 <! __end = .; 17 .debug_info 0 : { *(.debug_info) } 18 .debug_line 0 : { *(.debug_line) } 19 .debug_abbrev 0 : { *(.debug_abbrev) } 20 .debug_frame 0 : { *(.debug_frame) } 21 } </pre> | <pre> 1 SECTIONS 2 { 3 . = 0x00000000; 4 .text : { *(.text) } 5 .rodata : { *(.rodata) } 6 Image_RO_Limit = .; 7 . = 0x04000000; 8 Image_RW_Base = .; 9 .data : { *(.data) } 10 Image_ZI_Base = .; 11 .bss : { *(.bss) } 12 Image_ZI_Limit = .; 13 end = .; 14 .debug_info 0 : { *(.debug_info) } 15 .debug_line 0 : { *(.debug_line) } 16 .debug_abbrev 0 : { *(.debug_abbrev) } 17 .debug_frame 0 : { *(.debug_frame) } 18 } </pre> |
|---|---|

In RAM

In Flash

Fig. G-2 The linker scripts that the program uses in RAM and Flash and their difference

When the above operation is completed, re-build the project, then click IDE menu Tools>Elf to Bin, change elf file into binary format file (*.bin). Finally use Embest Flash Programmer to download bin file onto Flash ROM of target board.

Question 5 Command script, linker script and memory map file

Question:

What is command script, linker script and memory map script?

Answer:

Command script:

While integrating environment and target connection, in the course of debugging software and after resetting target board, sometimes the user needs to integrate environment to automatically finish some special functions such as resetting target board, clearing off watchdog, screening and interrupting register, memory map, etc., these special functions can be completed through executing a group of commands. The text file saving a group of command sequence is called as command scripts file.

The following are the examples of command script:

```
; stop target CPU
stop
; configurations of the special register
memwrite 0xffe00000 0x01002535
memwrite 0xffe00004 0x02002121
memwrite 0xffe00024 0x06
; configuration for Mapping Address
memwrite 0xffe00020 0x01
refresh
download -v D:\Demo\armdemo\debug\led.elf 0x2000000
;end
```

Linker script:

In the Embed system development, it needs to use linker position file. This file describes the relevant information of code linker position, including code section, data section, address section, and the linker shall use the code of this file to the whole system as correct position. This file is called as linker scripts file.

The following is an example of Linker script:

SECTIONS

```
{
  . = 0x02000000;
  .text : { *(.text) }
  Image_RO_Limit = .;
  Image_RW_Base = .;
  .data : { *(.data) }
  .rodata : { *(.rodata) }
  Image_ZI_Base = .;
  .bss : { *(.bss) }
  Image_ZI_Limit = .;
  __bss_start__ = .;
  __bss_end__ = .;
  end = .;
  .debug_info      0 : { *(.debug_info) }
  .debug_line      0 : { *(.debug_line) }
  .debug_abbrev    0 : { *(.debug_abbrev)}
  .debug_frame     0 : { *(.debug_frame) }
}
```

Memory linker map file:

In the course of debugging software, accessing illegal memory will occur abnormal in some processors and target boards. If the abnormality is not handled, it may cause the debugging course of software incapable to continue. In order to prevent the above issues and adjust the accessing speed of emulator to reach to proper level, a file for describing property of each memory area is provided. Such file is called as memory map file.

The following is an example of map file:

| | | | |
|------------|----------|---------|--------------|
| ONCHIPRAM | 0 | 8000 | RW - - - - - |
| EXTERNDRAM | 100000 | 128000 | RW - - - - - |
| FLASH | 1000000 | 128000 | R - - - - - |
| SRAM | 2000000 | 512000 | RW - - - - - |
| PERIREG | FFC00000 | 4000000 | R - - - - - |

Question 6 Definition of linker script

Question:

Please explain the meaning of linker scripts file?

Answer:

The linker script file is used in flash solidification:

```
SECTIONS
{
  . = 0x000000;          Assign current address as 0
  .text : { *(.text) };  Code section, put program code at this place
                        from 0 identification code
  .rodata : { *(.rodata) }; Read-only data section, static global variable
                        and other immovable values in program are put in this section
  Image_RO_Limit = .;   Length of read-only area, the symbol used in
                        the startup program
  . = 0x0400000;        Assign current address as 0x400000
  Image_RW_Base = .;    Read-write the frame of the area, start the
                        symbols used in program
  .data : { *(.data) }; Data section, the global variables initialized in
                        the program are put in this section
  Image_ZI_Base = .;    Frame of resetting area, the symbol used in
                        startup program
  .bss : { *(.bss) };   Include global accessible data not initialized,
                        for example not initialized global variables
  Image_ZI_Limit = .;   Length of resetting area, the symbol used in
                        startup program
  end = .;              Ending address
  .debug_info           0 : { *(.debug_info) };   Output section of
                        debugging information
  .debug_line           0 : { *(.debug_line) }
  .debug_abbrev         0 : { *(.debug_abbrev)}
  .debug_frame         0 : { *(.debug_frame) }
}
```

Linker scripts file used in debugging RAM:

```
SECTIONS
{
  . = 0x000000;          Assign current address as 0
  .text : { *(.text) };  Code section, put program code at this place
                        from 0 identification code
```

```

Image_RO_Limit = .; Length of read-only area, the symbol used in
startup program
Image_RW_Base = .; Read-write the frame of the area, the symbol
used in startup program
.rodata : { *(.rodata) }; Read-only data section, the static global
variables and other immovable values in the program are put in this section
.data : { *(.data) }; Data section, the global variables initialized in
the program are put in this section
Image_ZI_Base = .; Frame of resetting area, the symbol used in
startup program
.bss : { *(.bss) }; Include global accessible data not initialized,
for example not initialized global variables
Image_ZI_Limit = .; Length of resetting area, start the symbols
used in program
end = .; ending address
.debug_info 0 : { *(.debug_info) }; Output section of
debugging information
.debug_line 0 : { *(.debug_line) }
.debug_abbrev 0 : { *(.debug_abbrev)}
.debug_frame 0 : { *(.debug_frame) }
}

```

For the section code as follows:

```

int A1;
int A2 =5;
const int A3 = 10;

void main()
{
    int A4;
    register int A5;
    A4 = A3;
}

```

The variable A1, as not initialized variable, will be stored in the section .bss;

The variable A2, as initialized variable, will be stored in the section .data;

The constant A3 is stored in read-only data section .rodata;

The code that main function is corresponding is stored in the section .text;

The local variable A4 is stored in the corresponding function stack of main function when the program executes main function;

The register variable A5 is directly stored in a register of ARM;

Question 7 How to migrate SDT assemble program

Question:

How to migrate the assembly code in the environment of ARM SDT as assembly code supported by free software gnu assembler integrated by Embest IDE?

Answer:

Migration description on SDT assembly program:

- 1) The note line replaces ";" with "#"
- 2) Substitution of pseudo instruction characters

| pseudo instructions characters in SDT | pseudo instructions characters in Embest IDE |
|--|---|
| INCLUDE | .include |
| TCLK2 EQU PB25 | .equ ECLK2 , PB25 |
| EXPORT | .global |
| IMPORT | .extern |
| DCD | .long |
| IF:DEF: | .ifdef |
| ELSE | .else |
| ENDIF | .endif |
| :OR: | |
| :SHL | << |
| RN | .req |
| GBLA | .global |
| BUSWIDTH SETA 16 | .equ BUSWIDTH, 16 |
| MACRO | .macro |
| MEND | .endm |
| END | .end |
| AREA Word, CODE, READONLY | .text |
| AREA Block, DATA, READWRITE | .data |

| | |
|--------|--------|
| CODE32 | .arm |
| CODE16 | .thumb |
| LTORG | .ltorg |
| % | .fill |
| Entry | Entry: |

3) Substitution of operand and operator

| | |
|---------------------|----------------------|
| ldr pc, [pc, #&18] | ldr pc, [pc, #+0x18] |
| ldr pc, [pc, #-&18] | ldr pc, [pc, #-0x18] |

“&” refers to hex

Question 8 Register and stack of ARM processors

Question:

Please explain the register and stack of ARM processors?

Answer:

The corresponding form of registers in ARM7:

R0 ---> R0

... ..

R9 ---> R9

R10 ---> SL

R11 ---> FP Frame pointer points stack bottom

R12 ---> IP

R13 ---> SP The register of stack pointer points to stack top

R14 ---> LR

R15 ---> PC

Legend of stack:

ARM7 under gcc 2.95/2.97 building uses r11 register as function stack bottom pointer, r13 as current function stack top pointer.

A typical function stack information structure is as follows:

| | | |
|----------------|--|-------------------------------|
| | Memory high address | |
| | Parameters of current function | Last function stack top (r12) |
| Function Stack | PC value (of no use) after entering function | Stack bottom (r11) |
| | Return address of current function | |
| | Last function r13 register value | |
| | Last function r11 register value | |

| | | |
|--|---|-----------------|
| | Local variable of current function | |
| | Parameters transferred to next function | Stack top (r13) |
| | Memory low address | |

Note: directly take r11 value to take PC value, other than function return address

Question 9 Building error caused by GCC bug

Question:

Free software gcc builder integrated by Embest IDE includes a software bug. This bug causes the target code produced from building to bring collapse of stack in some situations while using `__attribute__((interrupt ("IRQ")))` style statement C language common interruption function

How to program to ensure the normal operation of common interruption?

Answer:

User can ensure normal operation of common interruption through the method of assembly programming.

```
.EXTERN irq_func
.GLOBAL irq_entry
irq_entry:
stmdb sp!, {r0-r11, ip, lr} /* Save r0-r11, ip, lr */
ldr r0, = irq_func
mov lr, pc
bx r0 /* Use C interruption program*/
ldmia sp!, {r0-r11, ip, lr} /* Resume r0, ip, lr */
subs pc, r14, #4 /* Interruption and return */
```

`irq_entry` is the entry of common interruption function. It completes the pushing in stack with relevant register in the common interruption mode, calls `irq_func` function, pops out register, interrupts return function.

`irq_func` is to use C language or assembly language to prepare interruption treatment function. This function is not necessary to use `__attribute__((interrupt ("IRQ")))` style statement.

In addition, this bug of gcc software will not affect other interruption vector mode, and the treatment function of other vector modes can still use `__attribute__((interrupt ("FIQ")))`, `__attribute__((interrupt ("SWI")))` style statement.

Question 10 To debug the project of ARM SDT

Question:

How to use Embest IDE to debug the project in ARM SDT environment?

Answer:

Embest IDE can debug the file to be produced with ARM SDT project building linker.

The user needs to make the following settings in the building environment of SDT:

In ARM SDT main menu click Tools>Configure,

- 1) Take <cc>=armcc menu, select the page Language and Debug, set Debug Table Format as dwarf 1 option, and set Optimization Level as None option;
- 2) Take <asm>=armasm menu, select the page Options, and set Debug Table Format as dwarf 1 option;
- 3) Select armlink menu, select the page Output and set Output Formats as Arm Elf Image format option;

The above setting can aim at single project and be realized through selecting Project > Tool Configuration menu. The settings are same as the above.

The user can use the above settings, under ARM SDT environment, to build and produce the corresponding output file of this project prj.elf.

In Embest IDE environment, the user selects Project > Settings menu, sets this output file as symbol file and download file for debugging in the Debug page of dialog box, and sets the local directory of source file in directory page simultaneously, then it can perform debugging upon this file at source file level or assembly level.

Question 11 FAQ for ARM compiler

Question A

Which versions of ARM compiler does Embest IDE support?

Answer

Embest IDE supports the compilers of ARM SDT V2.50 and ARM SDT V2.51, but does not support the compilers over ARM ADS V1.2 version.

ARM ADS V1.2 uses the same compiler version as ARM SDT V2.51, but it screens some output formats of debugging information.

Question B

Although sometimes there shows “Command(s) successfully executed.” when compiling, it actually generates no target files, why?

Answer

Because the user’s ARM compiler hasn’t acquired authorization.

If armcc command is executed in DOS command line, it would show the prompt similar to the following:

This licence has not yet been installed.

Question C

Compiling whilst there shows “...Fail to executing above command. "C:\Bin\armasm" maybe not found or executed!” ?

Answer

1, It hasn’t yet been installed or the installation is wrong in ARM compiler directories. User should choose item “Tools>Options” and then install the ARM compiler path on the directories page of Options dialog box.

2, ARM compiler or relevant files are damaged, so execution fails.

Question

When linking, it shows the prompt like: “Warning: File C:\LIB\armlib_cn.32l not found.” ?

Answer

The prompt indicates failure to link to library file C.

User should refer to 5.4.3, open the configurations behind SDT or ADS projects and then install the search path of library file; provided user has installed ARMLIB in the environment variable, they will have a same result as installing the search path of library file in projects.