User Manual

GNUSH (ELF) Toolchain

Version: 5.4

Table of Contents

1 Introduction

GNU Tools is a set of GNU development tools for Renesas microcontrollers.

GNU tools are evolutionary embodiment of Free Software Foundation (FSF) GNU Compiler Collection (GCC) as envisioned by Richard Stallman of FSF in the mid-to-late eighties. Through the collective efforts of hundreds of volunteers and organizations GNU Tools have grown to a mature and robust suite of tools that are readily available in source form. The source code is freely distributable under the license terms of General Public License (GPL) from the FSF.

The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. GNU is a recursive acronym for ―GNU's Not Unix‖. Variants of the GNU operating system, which use the kernel called Linux, are now widely used; though these systems are often referred to as ―Linux‖, they are more accurately called GNU/Linux systems.

The Free Software Foundation (FSF), established in 1985, is dedicated to promoting computer users' rights to use, study, copy, modify, and redistribute computer programs. ―Free software‖ is a matter of liberty, not price. FSF promotes the development and use of free software, particularly the GNU operating system. FSF is the principal organizational sponsor of the GNU Project.

1.1 About GNU Tools

The KPIT GNUSH tools are pre-built binary toolchains packaged in ready to use Windows and Linux Installers along with Release Notes, Tutorials and FAQs. Full source code for the tools is also provided. Windows users can use the tools with Renesas' High-performance Embedded Workshop (HEW) IDE and other Renesas tools.

GNUSH Tools are based on the FSF GNU sources (Binutils, GCC) and Newlib library sources. All of these are released under free software and can be used for developing commercial applications with no royalties attached.

Terms of Use

The KPIT Cummins GNU Tools are based on the FSF GNU sources (Binutils, GCC) & Newlib library sources. All of these are released under free software licenses and can be used for developing commercial applications with no royalties attached.

If you use GNU Tools to develop your application, you should NOT have to release the source code of your application

The GNUSH/H8/M16CM32C/RX installers are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

The GNUSH/H8/M16CM32C/RX installers, FAQs, Migration Guides, User manual are intellectual property of KPIT Cummins. You may download and use the tools and information available on this website.

The GNUSH/H8/M16CM32C/RX Linux RPMs and SH-Linux RPMs and packages are not re-distributable from other websites. Users can only direct their customers/end-users to this site. The Windows GNUSH/H8/M16CM32C/RX installers can be distributed to customers/end -users. The customers/end-users must be registered with GNU Tools and Support website in order to obtain an 'Activation code' required for proceeding with the toolchain installation. Registration is free, quick and easy and only required once

1.2 System Requirements

KPIT Tools provides a GNU Linux -like development environment on Windows and Linux hosts. It is assumed that the user has a working knowledge of UNIX / Linux command line utilities and software tools.

1.2.1 Systems Requirement (Windows Host environment)

a. Intel architecture (i586/i686) PC running one of the following Microsoft operating systems

- Windows 7
- Windows Vista
- Windows XP Professional
- Windows 2000 Professional
- Windows NT 4.0 Workstation, with Service Pack 4 or higher
- Windows 9x is no longer supported in v3.x due to limited system resources in the operating system.

b. CPU Clock Rate: 200 MHz or higher

c. Minimum System RAM:128 MB

d. Free Disk Space: 300 MB

1.2.2 Systems Requirement (Linux Host environment)

a. Intel architecture (i586/i686) PC running a Linux 2.x kernel.

b. CPU Clock Rate: 200 MHz or higher

c. Minimum System RAM: 128 MB

d. Free Disk Space: 300 MB

1.3 Targets Supported

GNU Tools have toolchains for RX, H8, SH and R8C, M16C, M32CM and M32C families of microcontrollers and supports various targets.

Supported targets from SH family are;

SH-1, SH-2, SH-2A, SH-2E, SH2-DSP, SH-3, SH-3E, SH3-DSP, SH-4, SH-4A, SH4AL

2 How to use GNUSH Tools

This section is intended to provide an abbreviated overview of using the GNU compiler and tools to generate executable programs for a target processor. It starts from installation and environment set and continues further with tools usage. Mastering the tools and using it to develop embedded systems on a practical scale requires greater in-depth knowledge than can not be covered here.

2.1 Install and Setup Procedure

There are two host versions of GNUSH Tools: Windows and Linux. The information in this Section gives the user details of installation, environment configuration, removal of toolsuite, setup testing.

2.1.1 Install GNUSH Tools on Windows Host

Download installer executable from GNU Tools website. Name of this installer executable has the form ―GNUSH<version>-ELF.exe‖, where <version> is toolchain version. For example if toolchain version is ―v0902‖ then its installer will have the name ―GNUSHv0902-ELF.exe‖. In the following sections, installer executable is referred to as ―windows installer‖.

2.1.1.1 Installing GNUSH when HEW and KPIT GNU Eclipse are not present.

Step 1

Start the installation by double clicking on the installer. (Use Windows Explorer to navigate to the place where installer was downloaded.)

Step 2

After starting the installation, follow the onscreen instructions. This will include providing email id and activation code. Enter the email id and the activation code. Accept the license agreement to proceed futher.

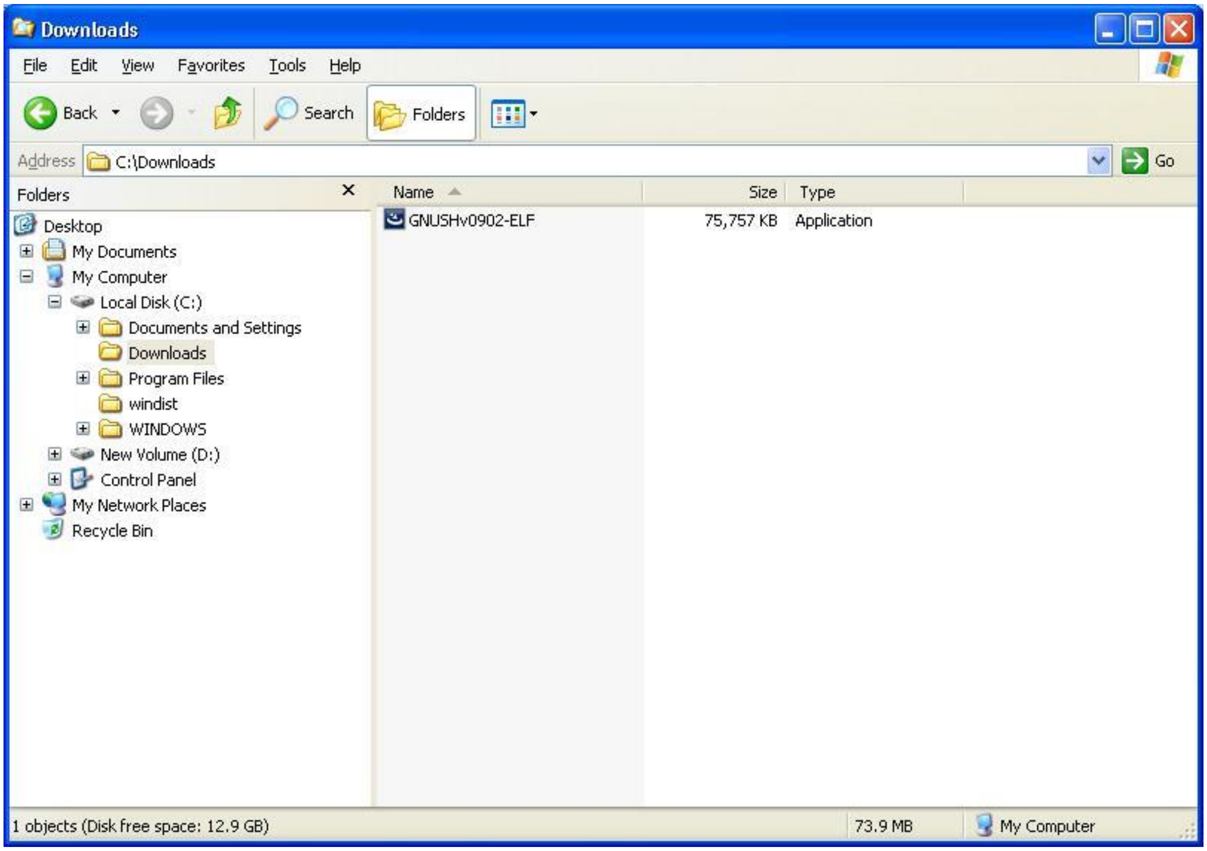
Step 3

Select the installation folder and components to install.

Following series of screen shots from ―Step 4‖ and onward show a typical sequence of installation process.

Step 4

Double click on the installer executable to start installation

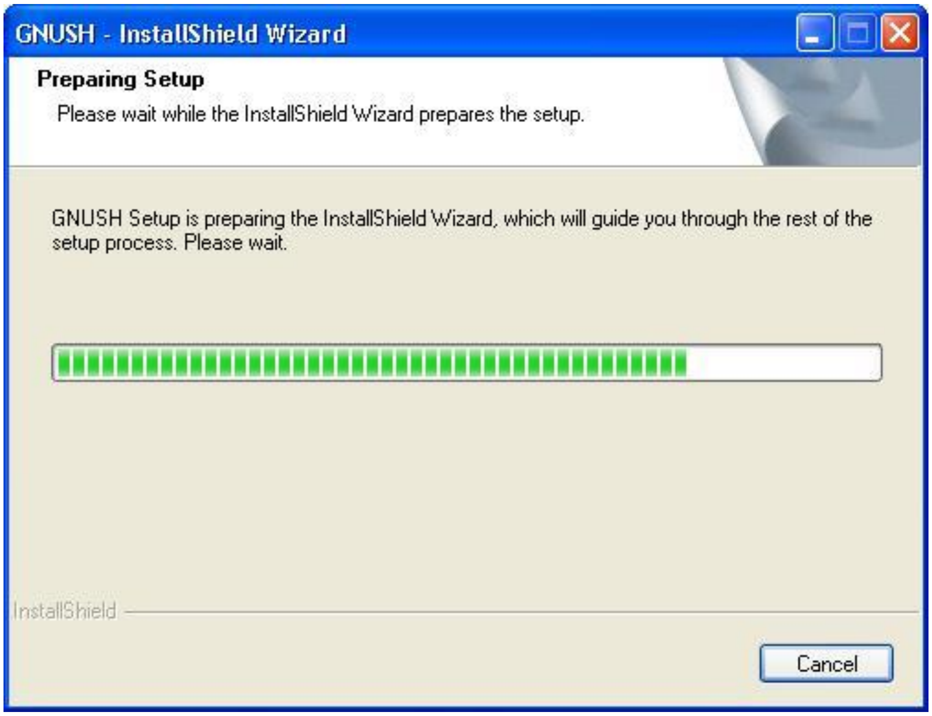


Name	Size	Type
GNUSHv0902-ELF	75,757 KB	Application

1 objects (Disk free space: 12.9 GB) 73.9 MB

Step 5

First step that installer takes is, it extracts files required for installation. This step doesn‘t require any user intervention.



GNUSH - InstallShield Wizard

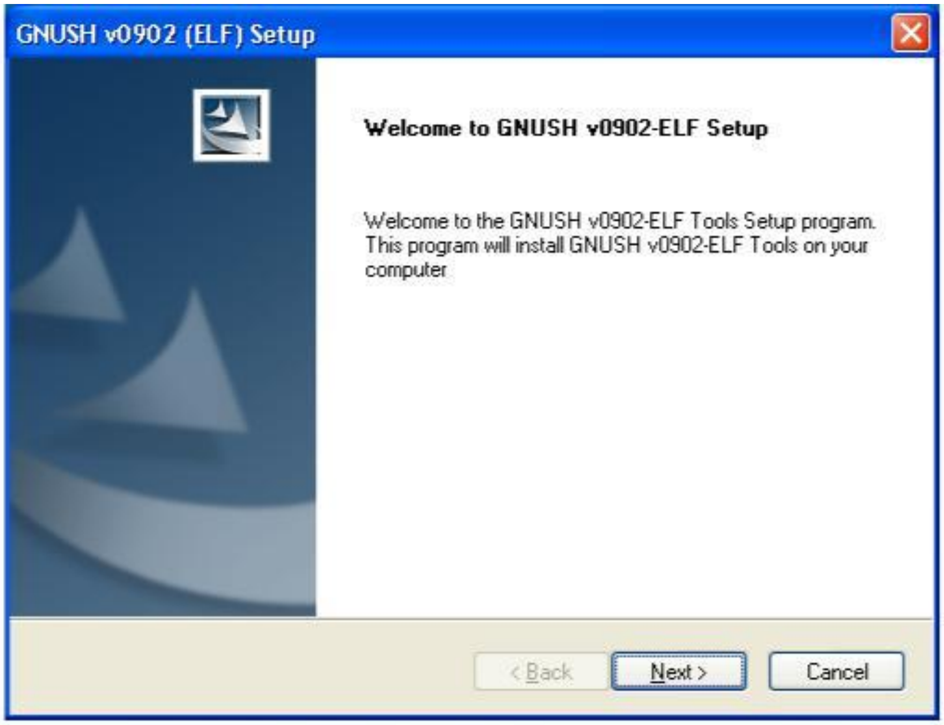
Preparing Setup

Please wait while the InstallShield Wizard prepares the setup.

GNUSH Setup is preparing the InstallShield Wizard, which will guide you through the rest of the setup process. Please wait.

Step 6

Welcome note

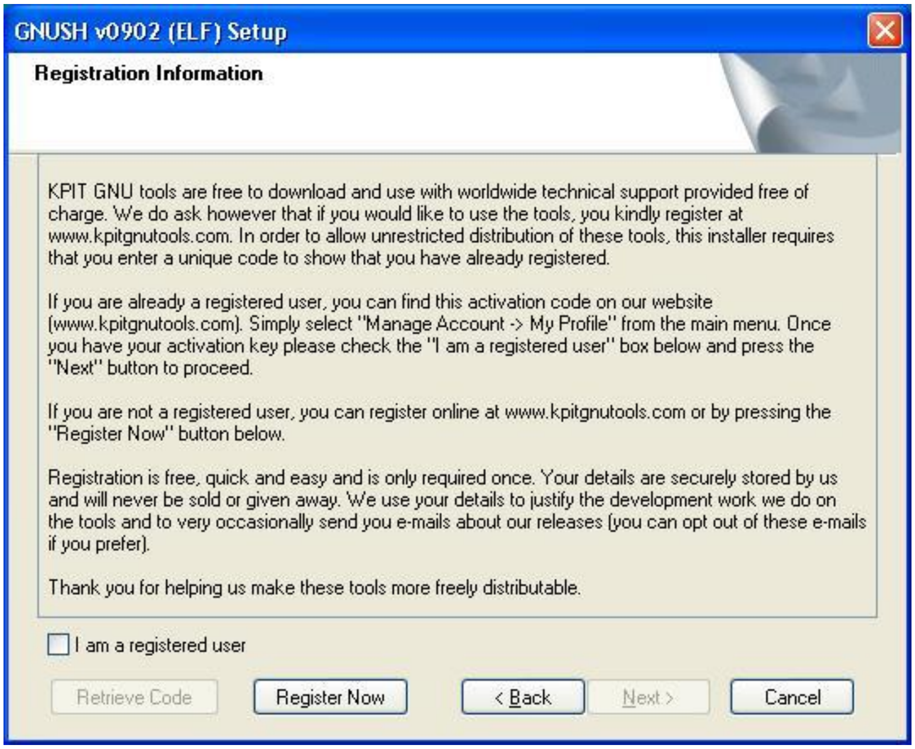


GNUSH v0902 (ELF) Setup

Welcome to GNUSH v0902-ELF Setup

Welcome to the GNUSH v0902-ELF Tools Setup program. This program will install GNUSH v0902-ELF Tools on your computer

Once you check the _I am a registered user‘ option _Next‘ button will get activated. If you are not a registered user please follow the procedure described in figure below to register online.



KPIT GNU tools are free to download and use with worldwide technical support provided free of charge. We do ask however that if you would like to use the tools, you kindly register at www.kpitgnutools.com. In order to allow unrestricted distribution of these tools, this installer requires that you enter a unique code to show that you have already registered.

If you are already a registered user, you can find this activation code on our website (www.kpitgnutools.com). Simply select "Manage Account -> My Profile" from the main menu. Once you have your activation key please check the "I am a registered user" box below and press the "Next" button to proceed.

If you are not a registered user, you can register online at www.kpitgnutools.com or by pressing the "Register Now" button below.

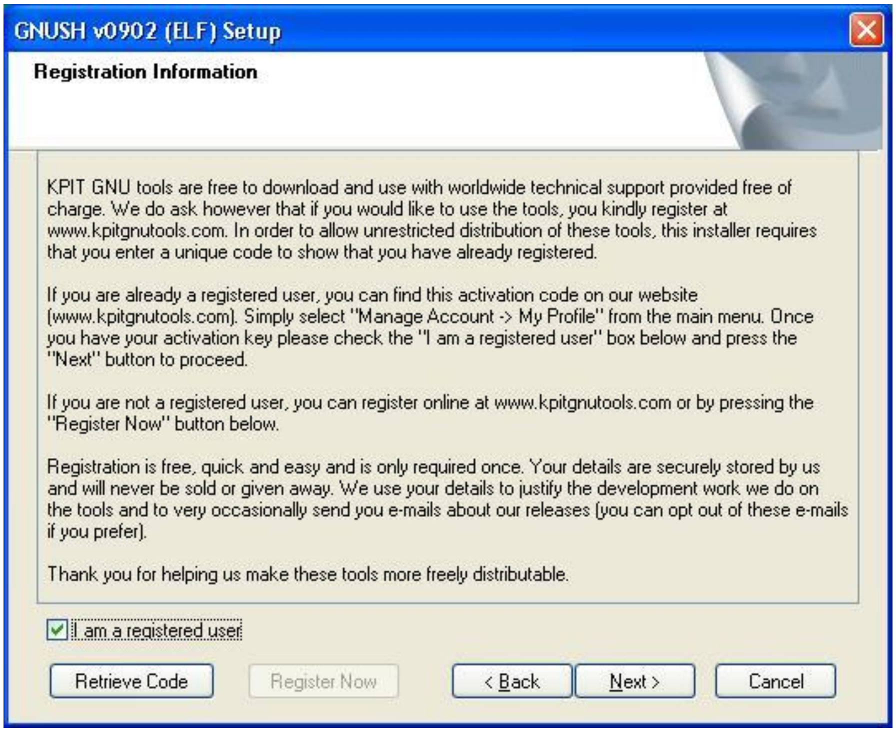
Registration is free, quick and easy and is only required once. Your details are securely stored by us and will never be sold or given away. We use your details to justify the development work we do on the tools and to very occasionally send you e-mails about our releases (you can opt out of these e-mails if you prefer).

Thank you for helping us make these tools more freely distributable.

□ I am a registered user

Step 7

After checking ‗I am a registered user' option click ‗Next' button to continue installation



GNUSH v0902 (ELF) Setup

Registration Information

KPIT GNU tools are free to download and use with worldwide technical support provided free of charge. We do ask however that if you would like to use the tools, you kindly register at www.kpitgnutools.com. In order to allow unrestricted distribution of these tools, this installer requires that you enter a unique code to show that you have already registered.

If you are already a registered user, you can find this activation code on our website (www.kpitgnutools.com). Simply select "Manage Account -> My Profile" from the main menu. Once you have your activation key please check the "I am a registered user" box below and press the "Next" button to proceed.

If you are not a registered user, you can register online at www.kpitgnutools.com or by pressing the "Register Now" button below.

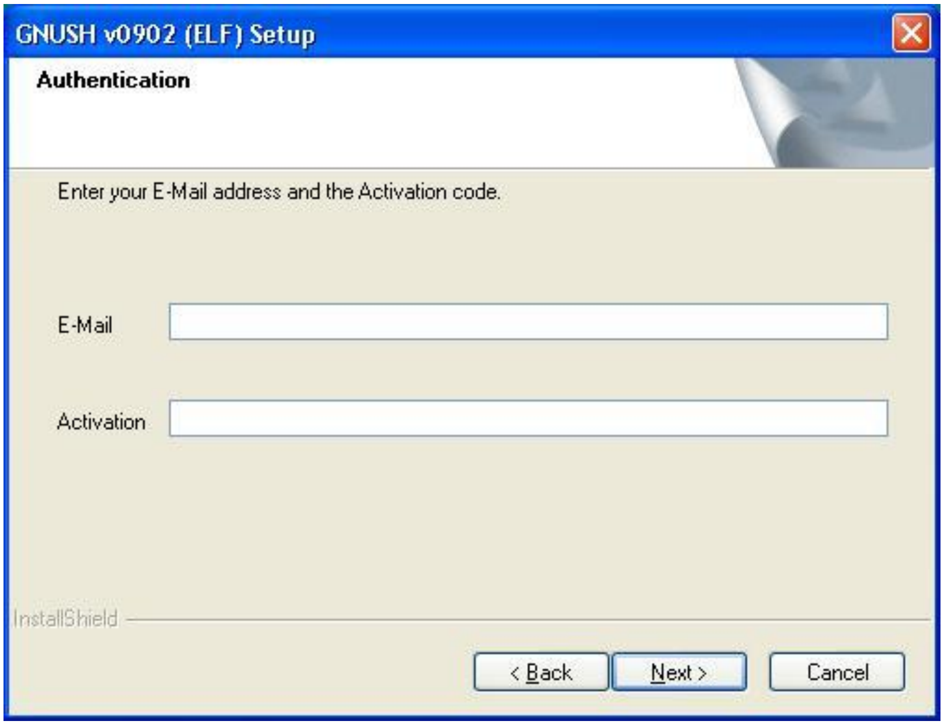
Registration is free, quick and easy and is only required once. Your details are securely stored by us and will never be sold or given away. We use your details to justify the development work we do on the tools and to very occasionally send you e-mails about our releases (you can opt out of these e-mails if you prefer).

Thank you for helping us make these tools more freely distributable.

☑ I am a registered user

Step 8

Enter your email address (used for registration) and activation code and click ‗Next' button. (If you are already a registered user, you can find this activation code on our website. Simply select "Manage Account -> My Profile" from the main menu.)



GNUSH v0902 (ELF) Setup

Authentication

Enter your E-Mail address and the Activation code.

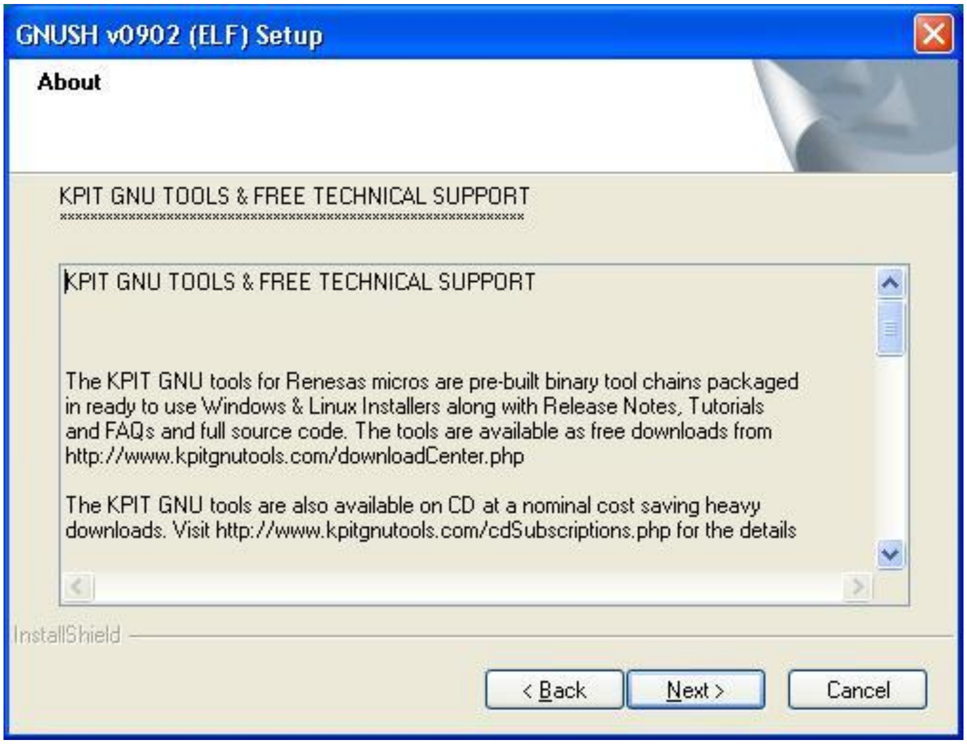
E-Mail

Activation

Note - The windows installers for GNUSH v0803 toolchain (and later) are modified to store the e-mail address and the activation code entered by the user during the first installation on the system. For any subsequent installations (of higher tooolchain versions) / re-installation, the installer will display the stored e-mail address and activation code in the "Authentication" dialog box. The user will not have to enter the details during every installation.

Step 9

This dialog provides brief description about GNU Tools. Click _Next' to continue.



KPIT GNU TOOLS & FREE TECHNICAL SUPPORT
****

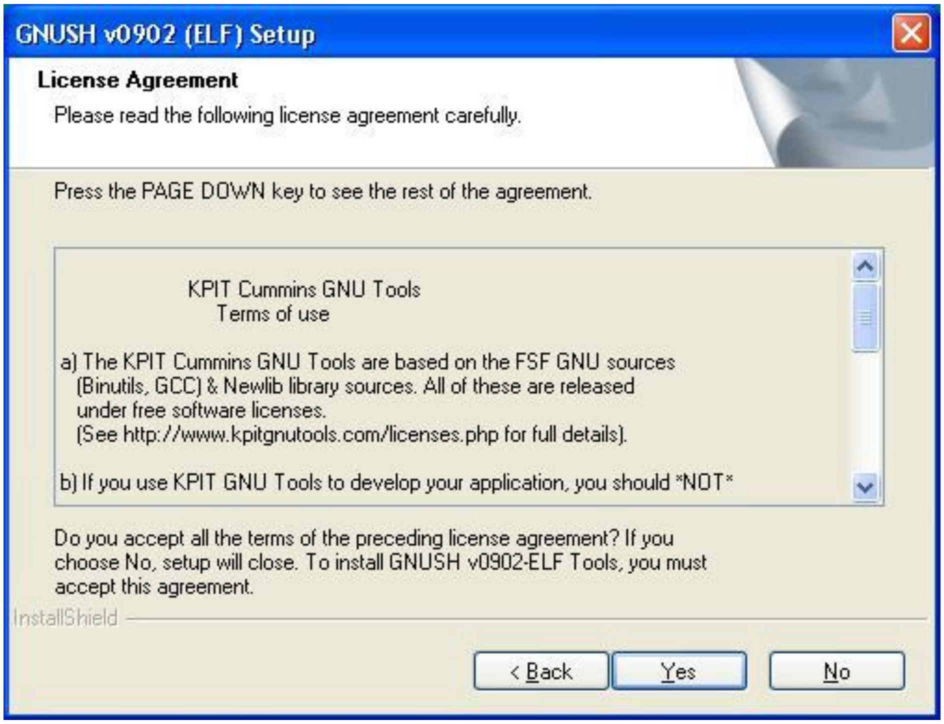
KPIT GNU TOOLS & FREE TECHNICAL SUPPORT

The KPIT GNU tools for Renesas micros are pre-built binary tool chains packaged in ready to use Windows & Linux Installers along with Release Notes, Tutorials and FAQs and full source code. The tools are available as free downloads from http://www.kpitgnutools.com/downloadCenter.php

The KPIT GNU tools are also available on CD at a nominal cost saving heavy downloads. Visit http://www.kpitgnutools.com/cdSubscriptions.php for the details

Step 10

To accept the License Agreement and continue with installation click _Yes‘



GNUSH v0902 (ELF) Setup

License Agreement

Please read the following license agreement carefully.

Press the PAGE DOWN key to see the rest of the agreement.

KPIT Cummins GNU Tools
Terms of use

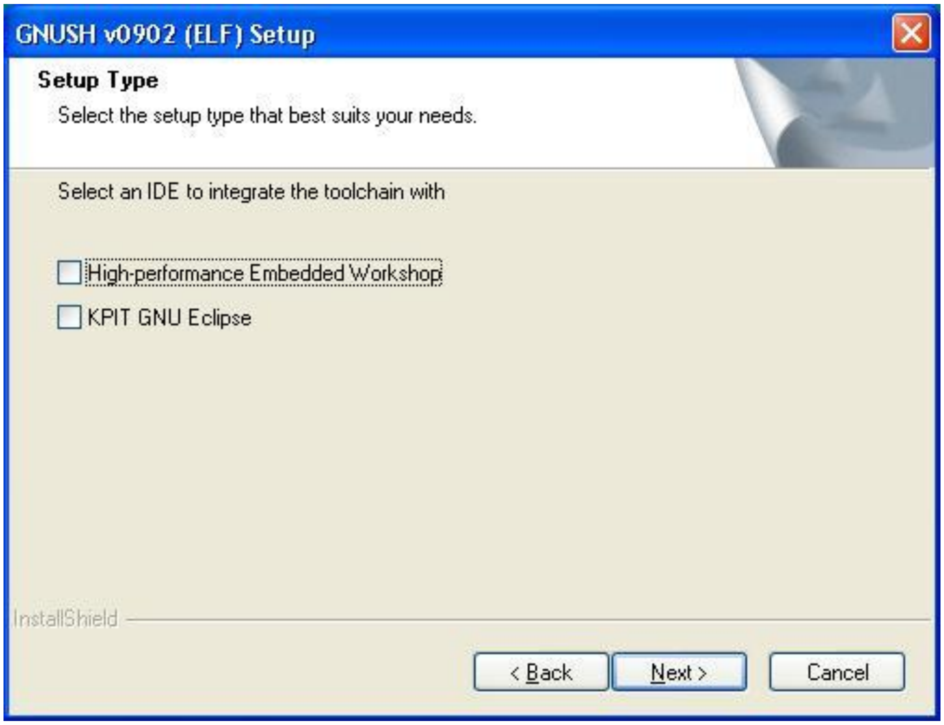
a) The KPIT Cummins GNU Tools are based on the FSF GNU sources (Binutils, GCC) & Newlib library sources. All of these are released under free software licenses. (See http://www.kpitgnutools.com/licenses.php for full details).

b) If you use KPIT GNU Tools to develop your application, you should *NOT*

Do you accept all the terms of the preceding license agreement? If you choose No, setup will close. To install GNUSH v0902-ELF Tools, you must accept this agreement.

Step 11

Once license agreement is accepted installer asks to select the IDE to integrate the toolchain. Click _Next‘ to continue.



GNUSH v0902 (ELF) Setup

Setup Type

Select the setup type that best suits your needs.

Select an IDE to integrate the toolchain with

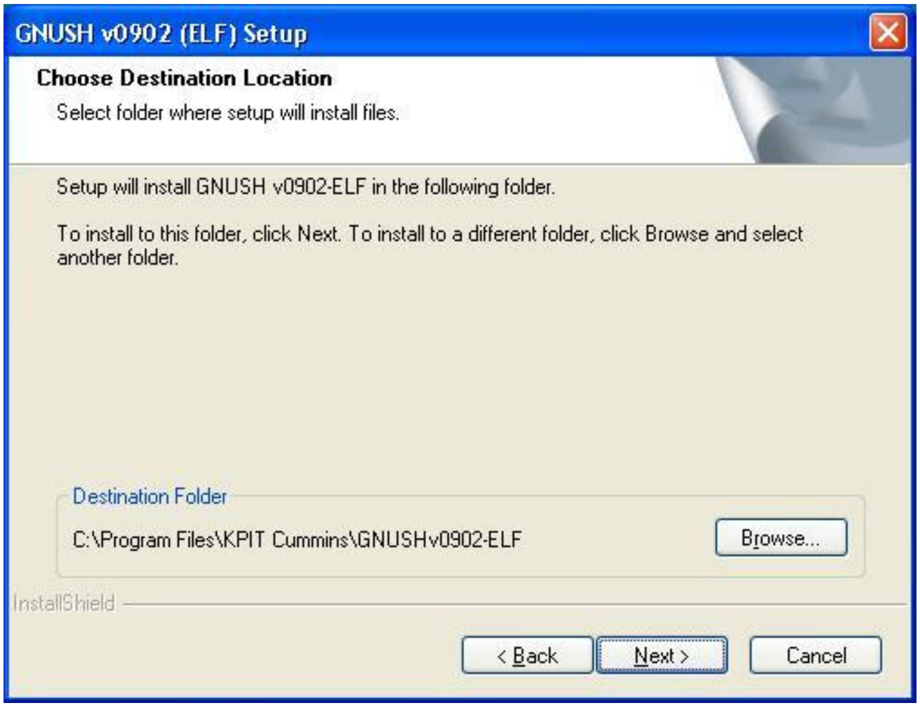
□ High-performance Embedded Workshop

□ KPIT GNU Eclipse

Note: The above dialog to integrate the GNUSH toolchain with HEW and KPIT GNU Eclipse IDE is provided in GNUSH v0902 (and later) toolchain installers.

Step 12

Click _Browse‘ to change installation folder if required and click _Next‘ to continue.



Choose Destination Location

Select folder where setup will install files.

Setup will install GNUSH v0902-ELF in the following folder.

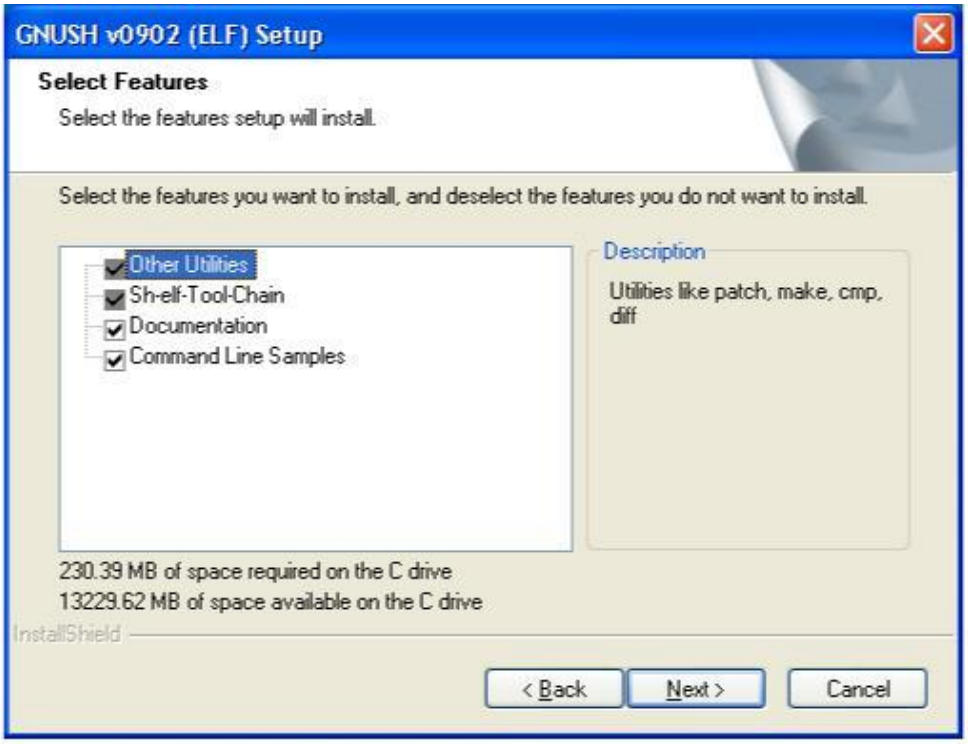
To install to this folder, click Next. To install to a different folder, click Browse and select another folder.

Destination Folder

C:\Program Files\KPIT Cummins\GNUSHv0902-ELF

Step 13

Select/deselect components to be installed by clicking on the appropriate checkboxes. Click _Next‘ to continue.



Select Features

Select the features setup will install.

Select the features you want to install, and deselect the features you do not want to install.

- ☑ Other Utilities
- ☑ Sh-elf-Tool-Chain
- ☑ Documentation
- ☑ Command Line Samples

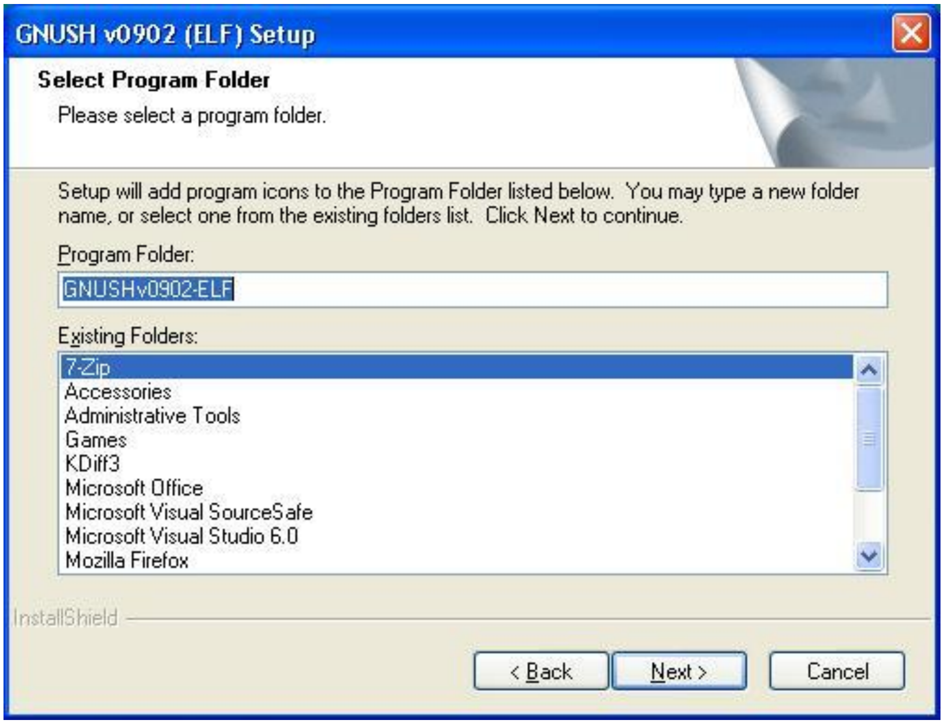
Description

Utilities like patch, make, cmp, diff

230.39 MB of space required on the C drive
13229.62 MB of space available on the C drive

Step 14

Change program folder if desired and click ‗Next' to continue.



Setup will add program icons to the Program Folder listed below. You may type a new folder name, or select one from the existing folders list. Click Next to continue.

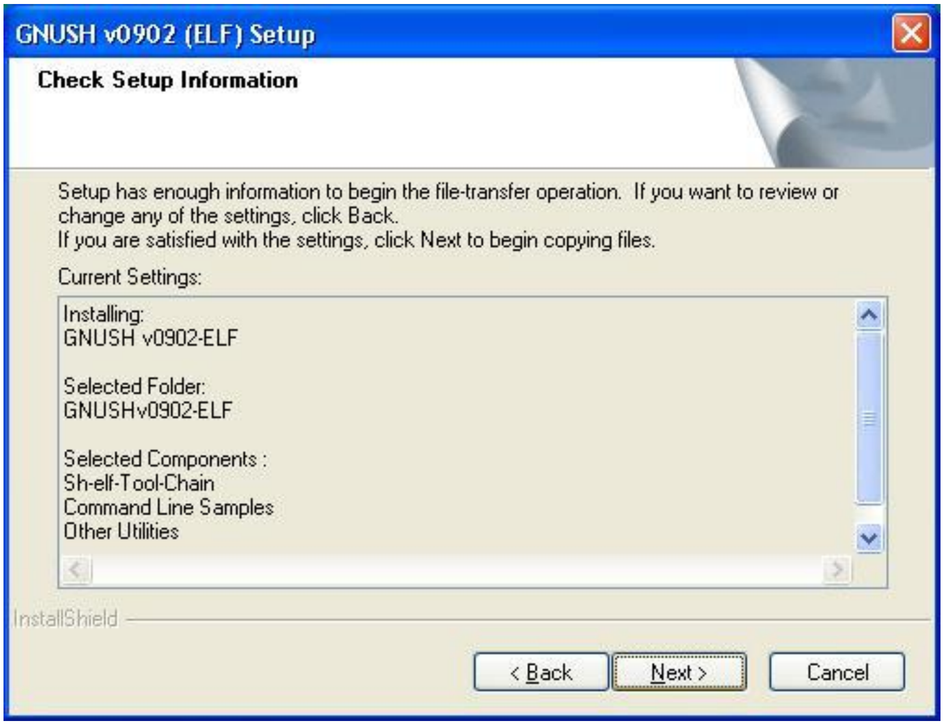
Program Folder: GNUSHv0902-ELF

Existing Folders:

- 7-Zip
- Accessories
- Administrative Tools
- Games
- KDiff3
- Microsoft Office
- Microsoft Visual SourceSafe
- Microsoft Visual Studio 6.0
- Mozilla Firefox

Step 15

This dialog confirms all the chosen options. Click ‗Next‘ to continue. To change any of the earlier selection you can click ‗Back‘ and go to the desired option and re-select the options.



Check Setup Information

Setup has enough information to begin the file-transfer operation. If you want to review or change any of the settings, click Back.
If you are satisfied with the settings, click Next to begin copying files.

Current Settings:

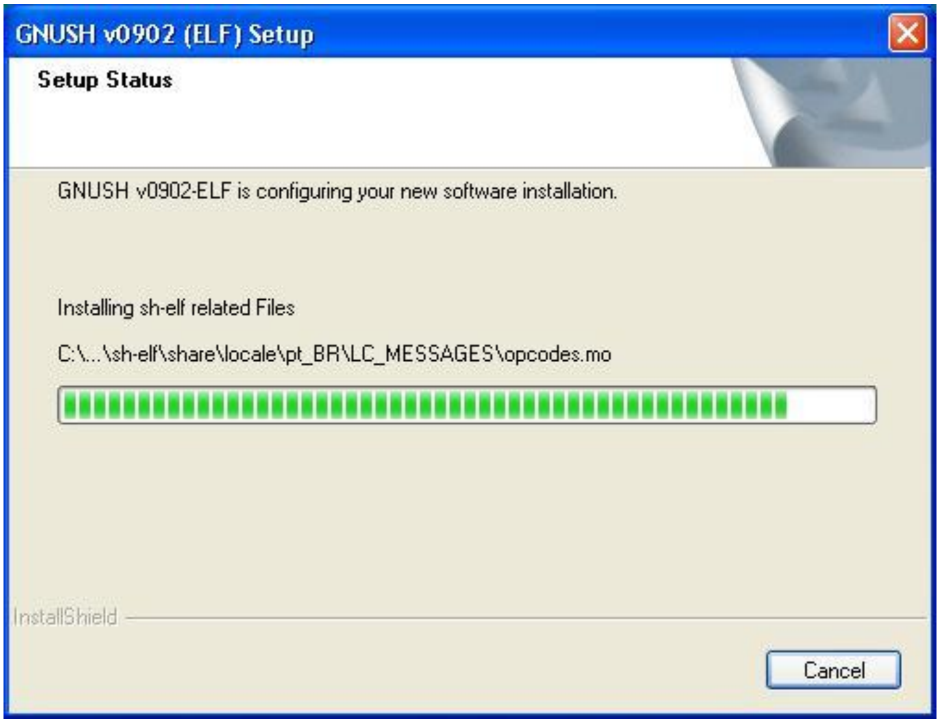
Installing:
GNUSH v0902-ELF

Selected Folder:
GNUSHv0902-ELF

Selected Components :
Sh-elf-Tool-Chain
Command Line Samples
Other Utilities

Step 16

Installer shows the status of installation process. Clicking ‗Cancel' here will exit installation.



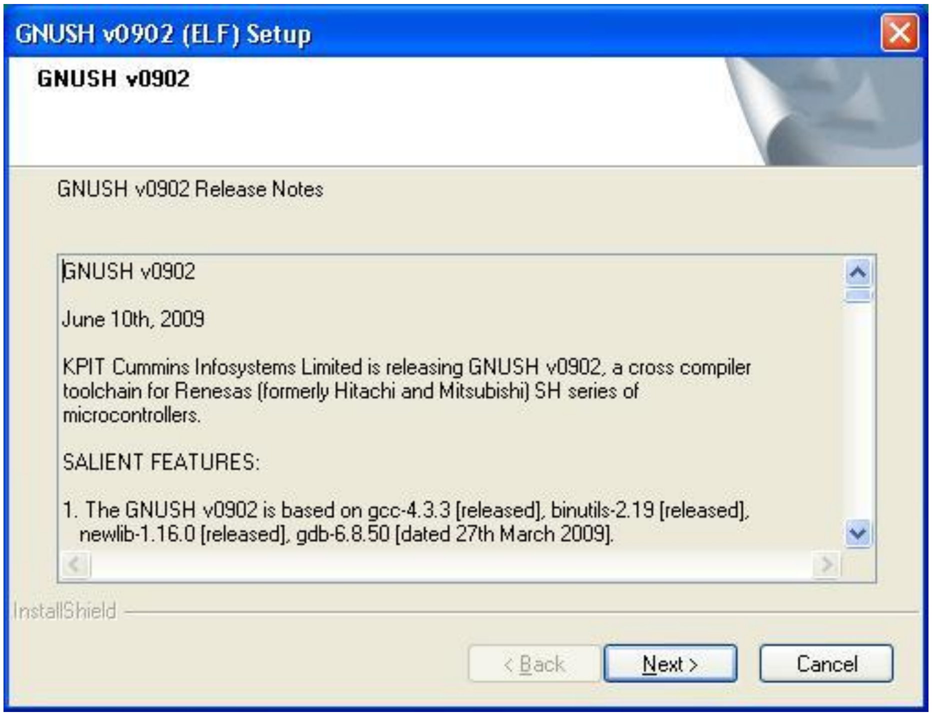
GNUSH v0902-ELF is configuring your new software installation.

Installing sh-elf related Files

C:\...\sh-elf\share\locale\pt_BR\LC_MESSAGES\opcodes.mo

Step 17

This dialog shows release notes for this toolchain version. Click ‗Next' to continue.



GNUSH v0902 (ELF) Setup

GNUSH v0902

GNUSH v0902 Release Notes

GNUSH v0902

June 10th, 2009

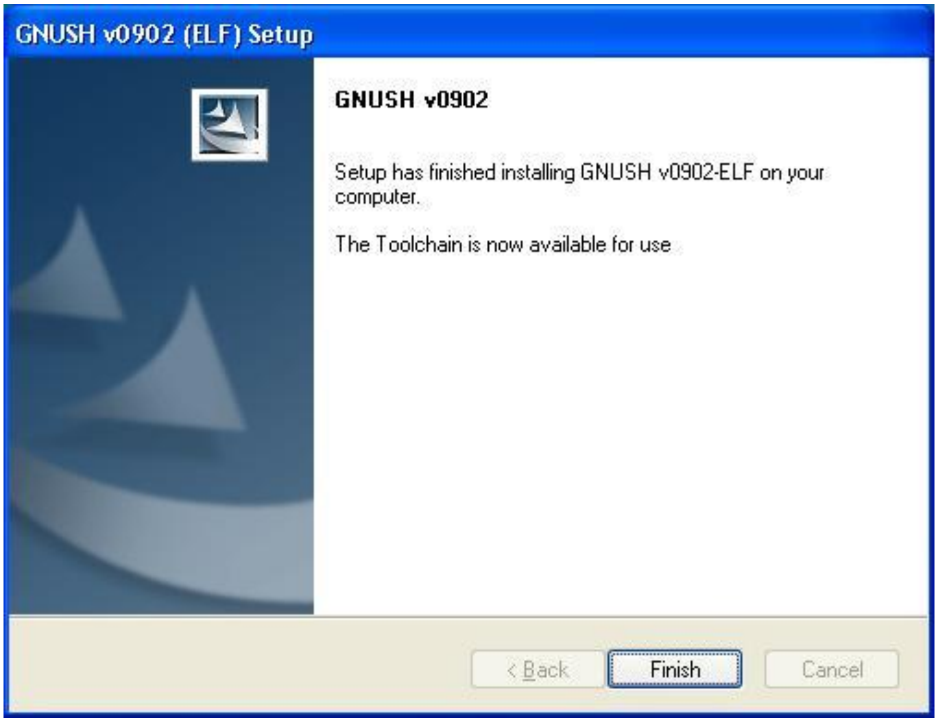
KPIT Cummins Infosystems Limited is releasing GNUSH v0902, a cross compiler toolchain for Renesas (formerly Hitachi and Mitsubishi) SH series of microcontrollers.

SALIENT FEATURES:

1. The GNUSH v0902 is based on gcc-4.3.3 [released], binutils-2.19 [released], newlib-1.16.0 [released], gdb-6.8.50 [dated 27th March 2009].

Step 18

Click ‗Finish' to complete the installation.



GNUSH v0902 (ELF) Setup

GNUSH v0902

Setup has finished installing GNUSH v0902-ELF on your computer.

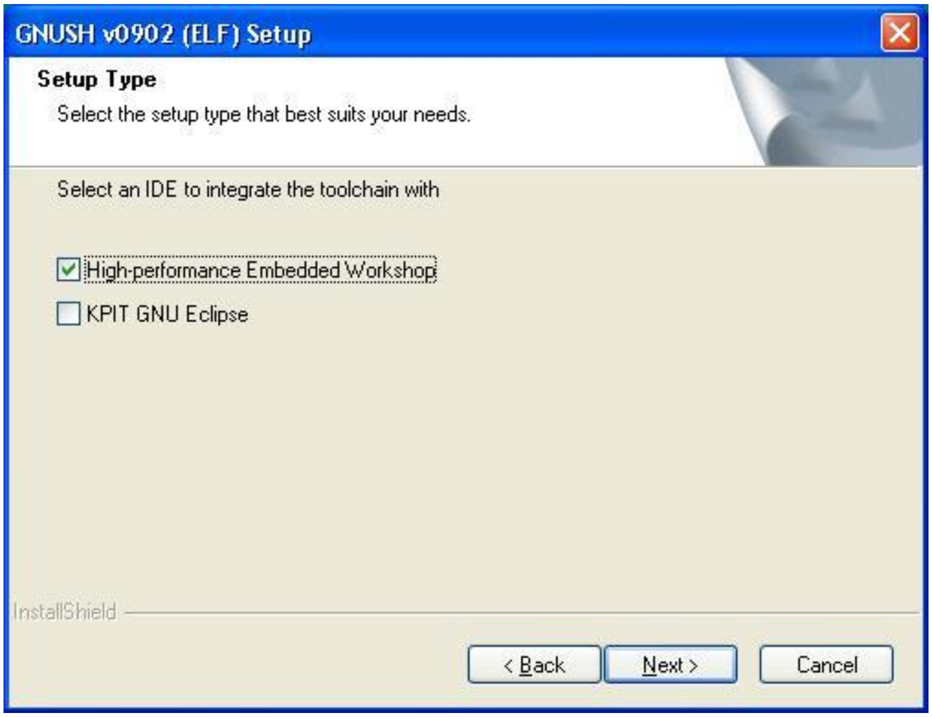
The Toolchain is now available for use

2.1.1.2 Installing GNUSH when HEW is present

Installation process till Step 10 is same as above. However, if HEW is present on system, the following Steps are different.

Step 11

Select the ‗High-performance Embedded Workshop' to integrate the toolchain with HEW.



GNUSH v0902 (ELF) Setup

Setup Type

Select the setup type that best suits your needs.

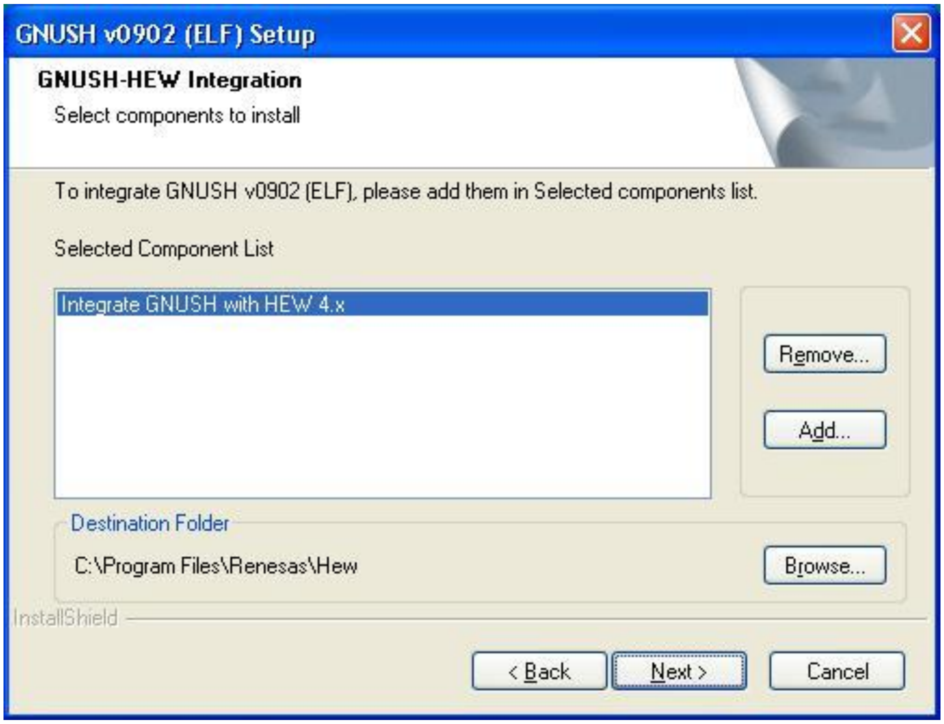
Select an IDE to integrate the toolchain with

☑ High-performance Embedded Workshop

☐ KPIT GNU Eclipse

Step 12

Installer detects HEW installation on system. You can integrate toolchain with different HEW installations by adding/removing desired HEW versions to selected components list using Add/Remove buttons.



GNUSH v0902 (ELF) Setup

GNUSH-HEW Integration

Select components to install

To integrate GNUSH v0902 (ELF), please add them in Selected components list.

Selected Component List

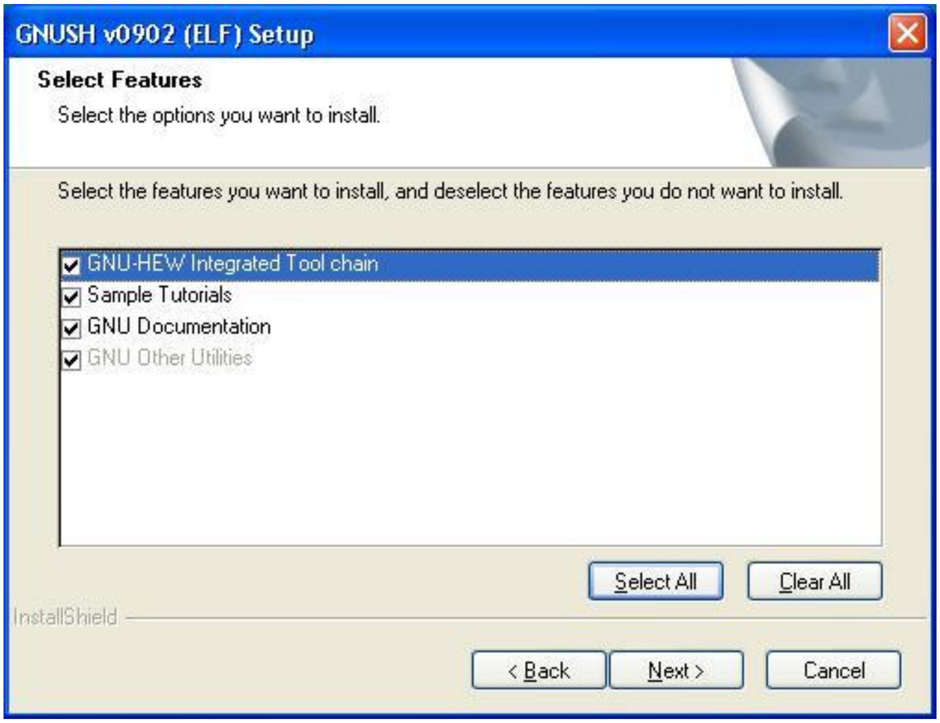
Integrate GNUSH with HEW 4.x

Destination Folder

C:\Program Files\Renesas\Hew

Step 13

Select/deselect components to be installed by clicking on the appropriate checkboxes. Click ‗Next' to continue.



GNUSH v0902 (ELF) Setup

Select Features

Select the options you want to install.

Select the features you want to install, and deselect the features you do not want to install.

- ☑ GNU-HEW Integrated Tool chain
- ☑ Sample Tutorials
- ☑ GNU Documentation
- ☑ GNU Other Utilities

The Installation steps after Step 12 are same as Steps 14 to 18 from previous section (section 2.1.1.1).

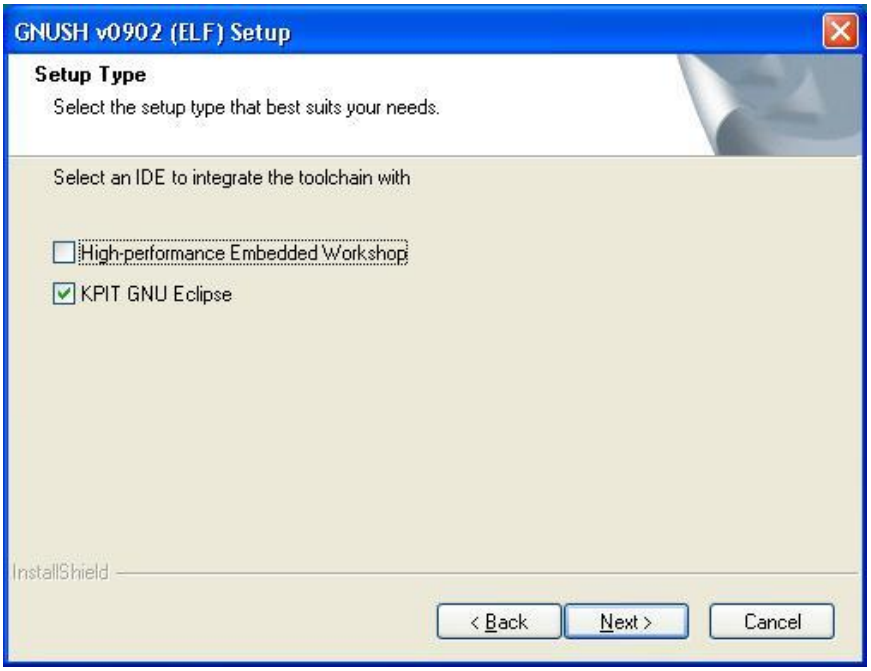
2.1.1.3 Installing GNUSH when ECLIPSE is present

Installation process till Step 10 is same as above. However, if Eclipse is present on system, the following steps are different.

Step 11

Select the ‗KPIT GNU Eclipse' to integrate the toolchain with Eclipse.

Click ‗Next' to continue.



GNUSH v0902 (ELF) Setup

Setup Type

Select the setup type that best suits your needs.

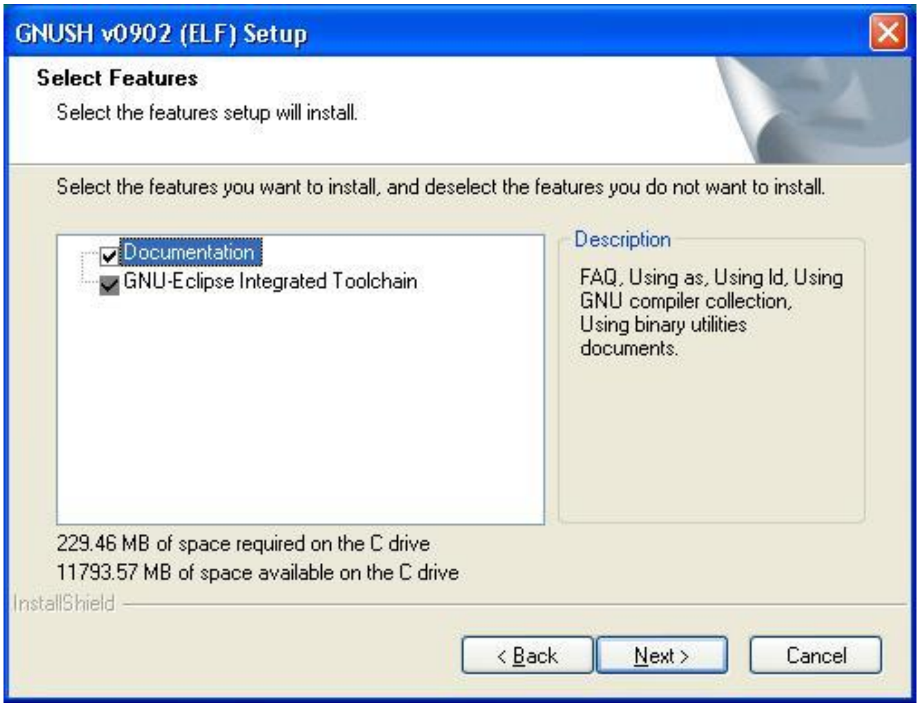
Select an IDE to integrate the toolchain with

☐ High-performance Embedded Workshop

☑ KPIT GNU Eclipse

Step 12

Select/deselect components to be installed by clicking on the ‗Documentation' checkbox. Click ‗Next' to continue.



GNUSH v0902 (ELF) Setup

Select Features

Select the features setup will install.

Select the features you want to install, and deselect the features you do not want to install.

- ☑ Documentation
- ☑ GNU-Eclipse Integrated Toolchain

Description

FAQ, Using as, Using ld, Using GNU compiler collection, Using binary utilities documents.

229.46 MB of space required on the C drive

11793.57 MB of space available on the C drive

The Installation steps after Step 12 are same as Steps 14 to 18 from previous section (section 2.1.1.1).

2.1.2 Uninstall GNUSH Tools on Windows Host

Following sections explain the un-installation procedure for GNUSH toolchain in following two scenarios;

a. GNUSH toolchain is not integrated with HEW

b. GNUSH toolchain is integrated with HEW

2.1.2.1 Uninstalling GNUSH toolchain not integrated with HEW

To uninstall GNUSH Toolchain if it is not integrated with HEW, follow the below procedure;

Step 1

Go to the toolchain shortcut in ‗Start' menu

Step 2

Select option ‗Uninstall GNUSH v<VERSION> (ELF)'

2.1.2.2 Uninstalling GNUSH toolchain integrated with HEW

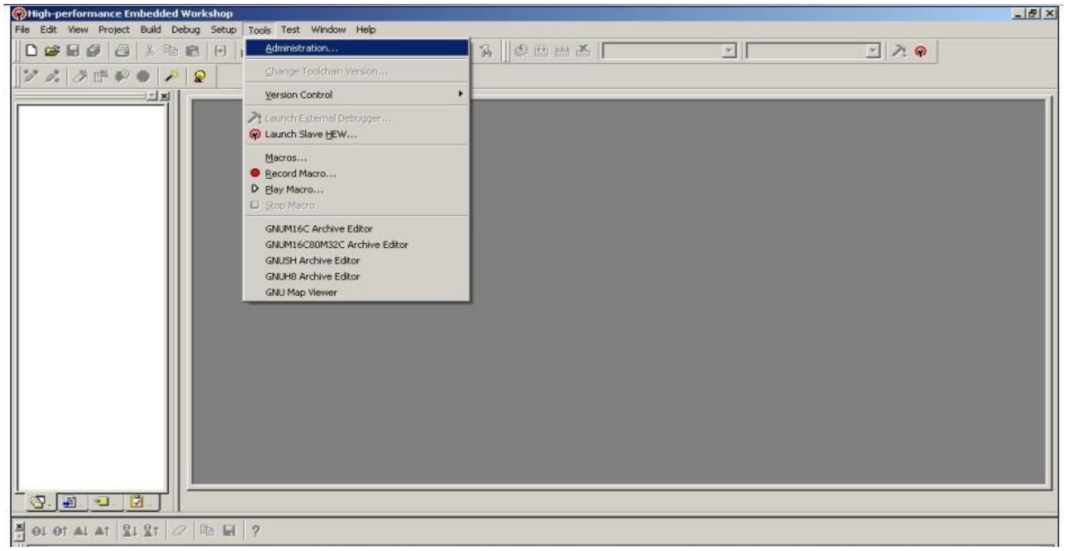
(This feature is not supported for HEW 4.5 and above)

Step 1

Invoke HEW.

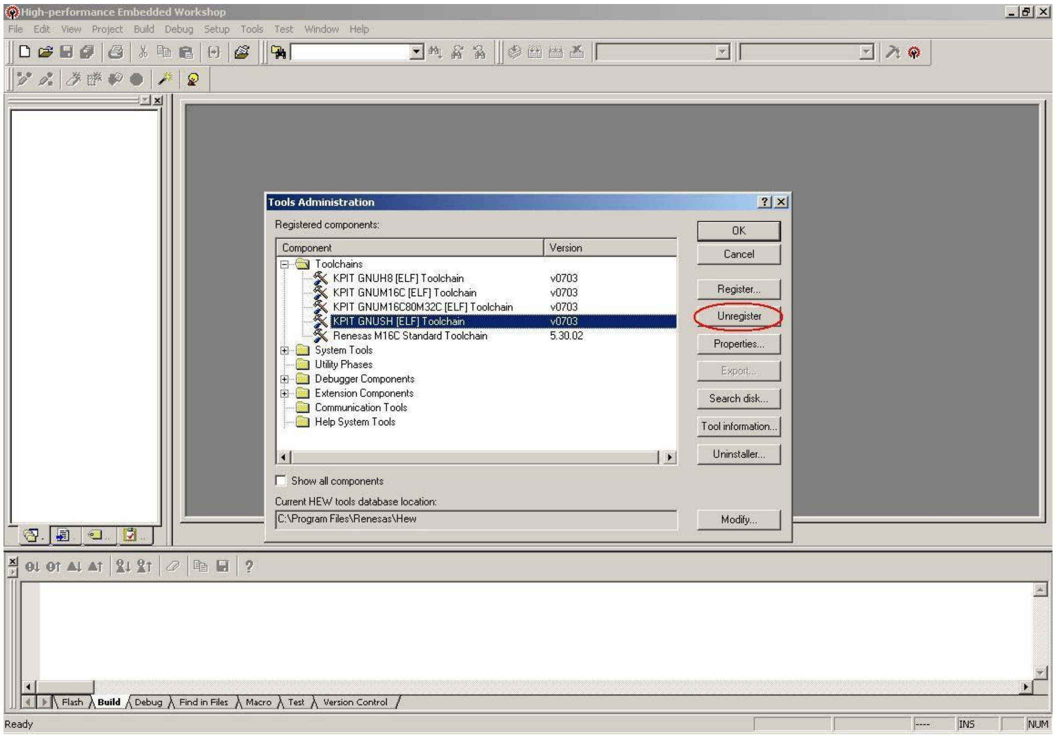
Step 2

Go to ―Tools->Administrator‖


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Help
Administration...
Change Toolchain Version...
Version Control
Launch External Debugger...
Launch Slave HEW...
Macros...
Record Macro...
Play Macro...
Stop Macro
GNUM16C Archive Editor
GNUM16C80M32C Archive Editor
GNUSH Archive Editor
GNUH8 Archive Editor
GNU Map Viewer

Step 3

Under ¯Tools Administration -> Toolchain‖, select the toolchain to be un-installed and click on Unregister‘.


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Help
Tools Administration
OK
Cancel
Register...
Unregister
Properties...
Export...
Search disk...
Tool information...
Uninstaller...
Modify...
Flash Build Debug Find in Files Macro Test Version Control
Ready
INS
NUM

Registered components:

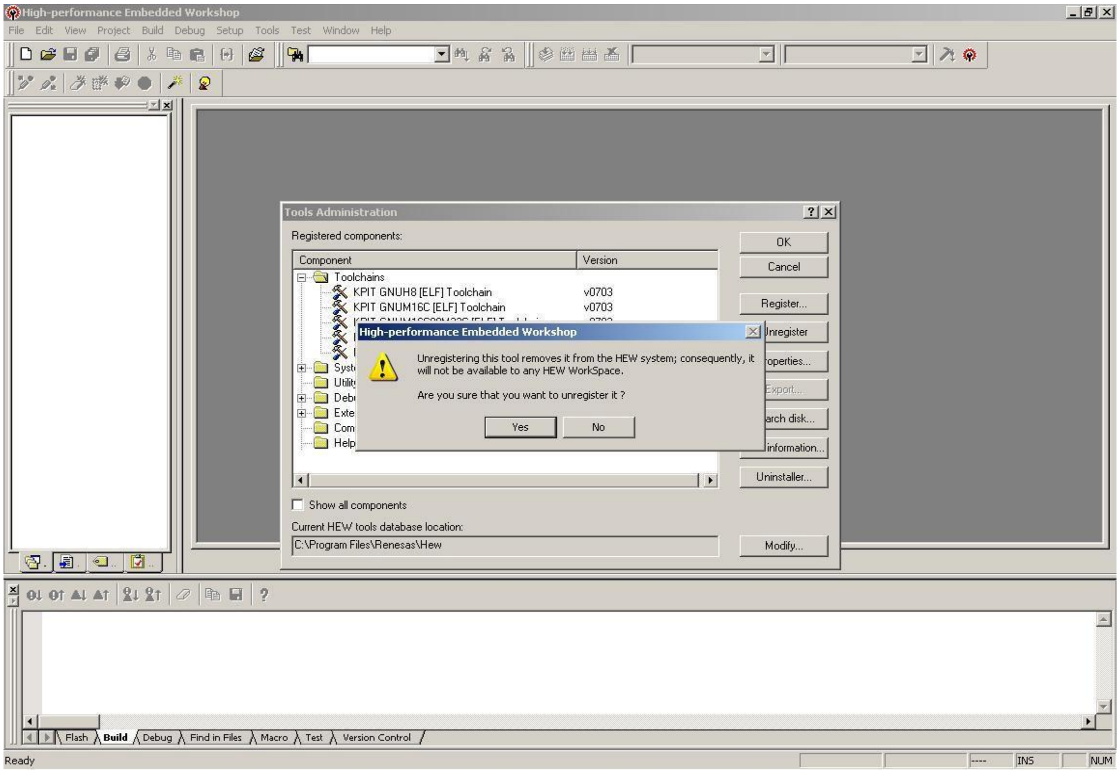
Component	Version
Toolchains	
KPIT GNUH8 [ELF] Toolchain	v0703
KPIT GNUM16C [ELF] Toolchain	v0703
KPIT GNUM16C80M32C [ELF] Toolchain	v0703
KPIT GNUSH [ELF] Toolchain	v0703
Renesas M16C Standard Toolchain	5.30.02
System Tools	
Utility Phases	
Debugger Components	
Extension Components	
Communication Tools	
Help System Tools	

☐ Show all components

Current HEW tools database location:
C:\Program Files\Renesas\Hew

Step 4

It displays the message asking for confirmation for un-registering the tools from HEW. Click _Yes‘.



Registered components:

Component	Version
Toolchains	
KPIT GNUH8 [ELF] Toolchain	v0703
KPIT GNUM16C [ELF] Toolchain	v0703

☐ Show all components

Current HEW tools database location: C:\Program Files\Renesas\Hew

High-performance Embedded Workshop

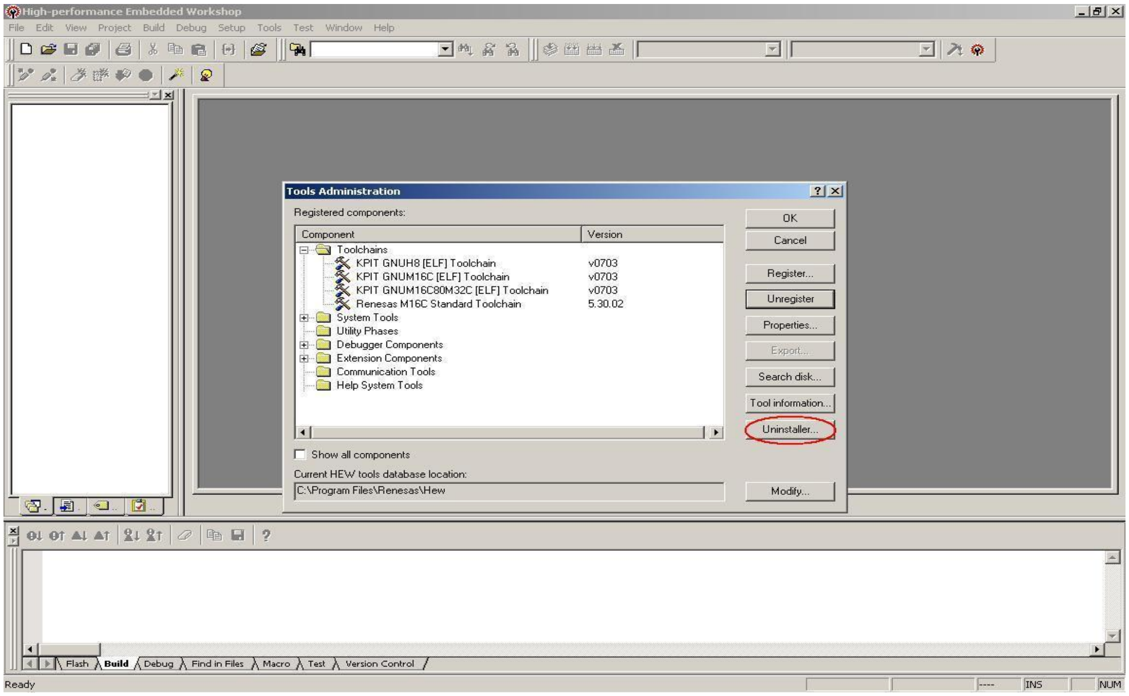
Unregistering this tool removes it from the HEW system; consequently, it will not be available to any HEW WorkSpace.

Are you sure that you want to unregister it ?

Yes No

Step 5

After successful un-registration of the tools, click _Uninstaller‘.



Registered components:

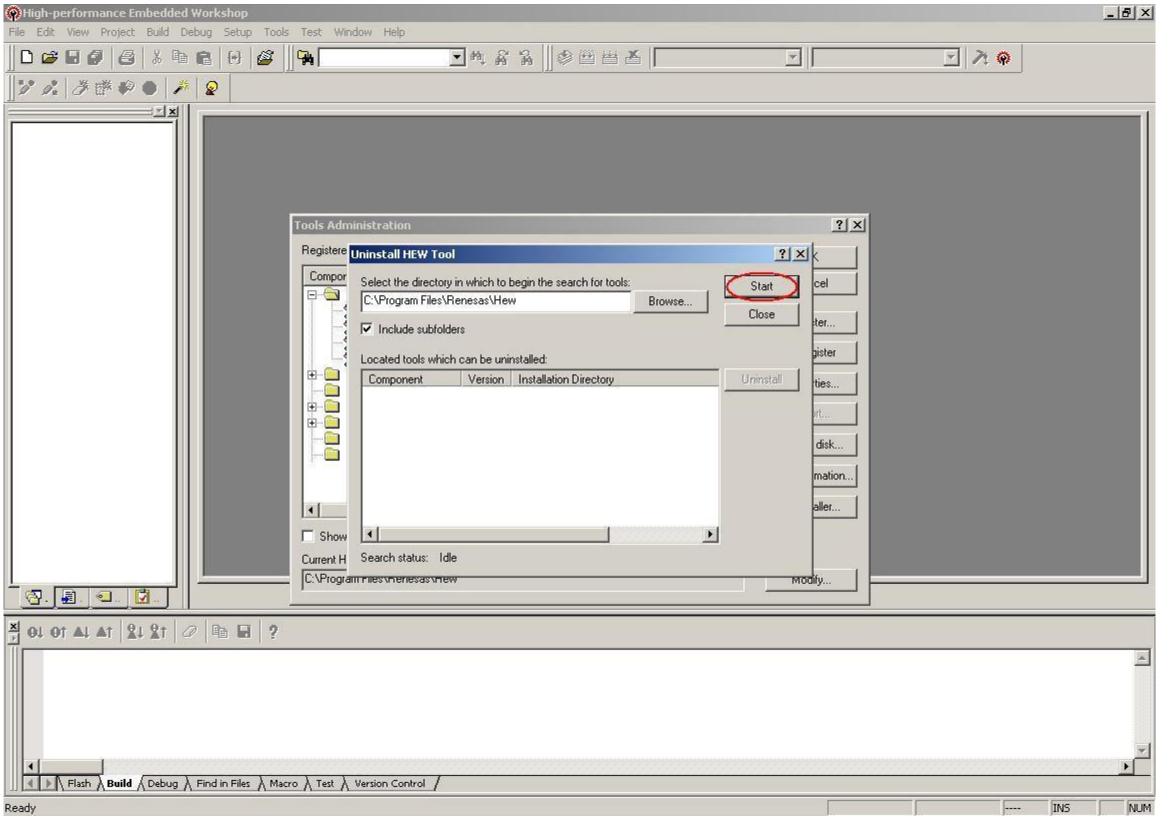
Component	Version
Toolchains	
KPIT GNUH8 [ELF] Toolchain	v0703
KPIT GNUM16C [ELF] Toolchain	v0703
KPIT GNUM16C80M32C [ELF] Toolchain	v0703
Renesas M16C Standard Toolchain	5.30.02
System Tools	
Utility Phases	
Debugger Components	
Extension Components	
Communication Tools	
Help System Tools	

□ Show all components

Current HEW tools database location:
C:\Program Files\Renesas\Hew

Step 6

Click on ‗Start' button. This will display the list of tools currently registered with HEW.



Select the directory in which to begin the search for tools:
C:\Program Files\Renesas\Hew

☑ Include subfolders

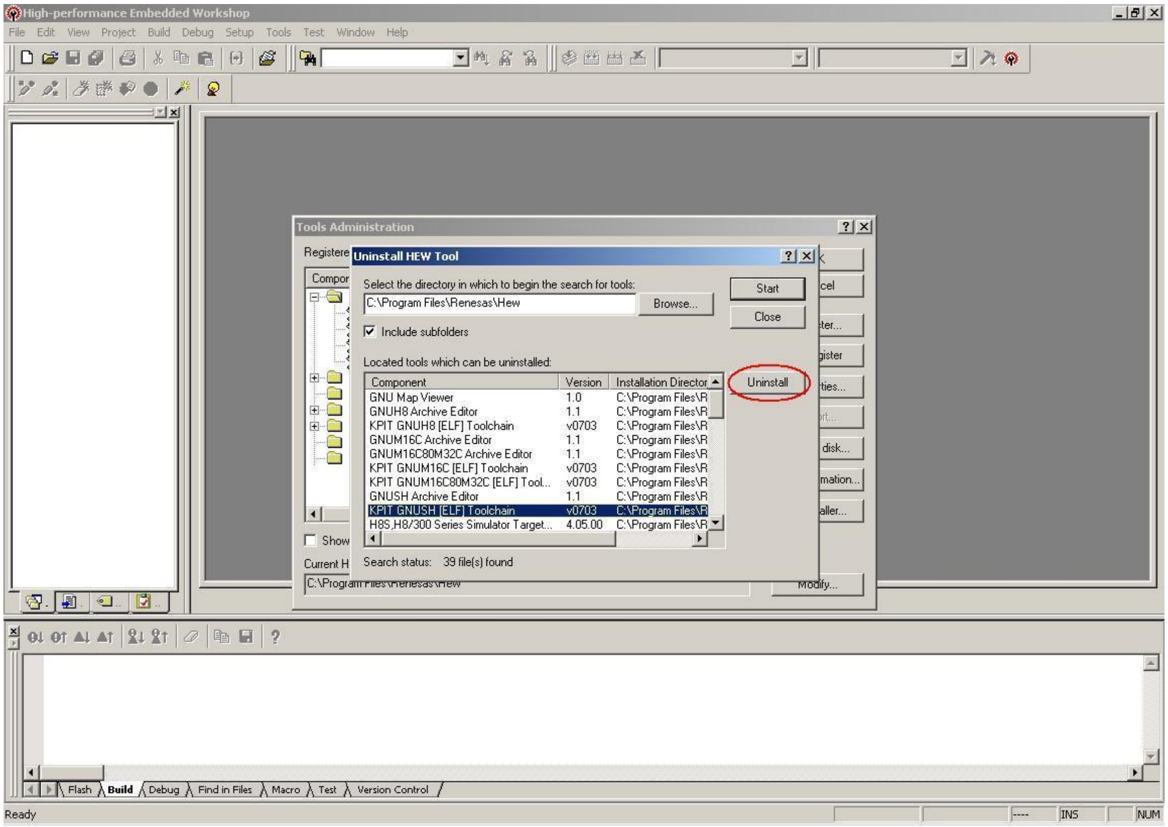
Located tools which can be uninstalled:

Component	Version	Installation Directory

Search status: Idle

Step 7

Choose the tool that needs to be un-installed from HEW and then click _Uninstall‘.



High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Help
Tools Administration
Uninstall HEW Tool
Start
Close
Browse...
Uninstall
Flash Build Debug Find in Files Macro Test Version Control
Ready
INS
NUM

Select the directory in which to begin the search for tools:
C:\Program Files\Renesas\Hew

☑ Include subfolders

Located tools which can be uninstalled:

Component	Version	Installation Director
GNU Map Viewer	1.0	C:\Program Files\R
GNUH8 Archive Editor	1.1	C:\Program Files\R
KPIT GNUH8 [ELF] Toolchain	v0703	C:\Program Files\R
GNUM16C Archive Editor	1.1	C:\Program Files\R
GNUM16C80M32C Archive Editor	1.1	C:\Program Files\R
KPIT GNUM16C [ELF] Toolchain	v0703	C:\Program Files\R
KPIT GNUM16C80M32C [ELF] Tool...	v0703	C:\Program Files\R
GNUSH Archive Editor	1.1	C:\Program Files\R
KPIT GNUSH [ELF] Toolchain	v0703	C:\Program Files\R
H8S,H8/300 Series Simulator Target...	4.05.00	C:\Program Files\R

Search status: 39 file(s) found

Step 8

Above procedure _un-registers‘ and _un-installs‘ the tools from HEW. In order to remove the tools completely from your system, after Step 6 above, follow the steps given in section _Uninstalling GNUSH toolchain not integrated with HEW.‘

2.1.3 Install GNUSH Tools on Linux Host

On Linux GNUSH Tools are installed using RPM package manager. RPM can be downloaded form GNU Tools website. Use the command specified below on bash to install tools.

```
#rpm –ivh gnush_<version>_elf-1-1.i386.rpm
```

In the above command <version> is tools version that you are installing. For example if installing v0902 tools then command will be;

```
#rpm –ivh gnush_v0902_elf-1-1.i386.rpm
```

Tools will be installed under ―/usr/share/‖ directory.

Note : The user should have root access to system to install RPM in /usr/share directory. For this; user needs to login as root user or use ‗su' command to pseudo login as root

2.1.4 Uninstall GNUSH Tools on Linux Host

To uninstall GNUSH Tools rpm package use the following command;

```
#rpm –e gnush_<version>_elf-1-1
```

In above command <version> is tools version. For example if installing v0902 tools then command will be;

```
#rpm –e gnush_v0902_elf-1-1
```

During un-installation, RPM removes only the files that were copied by the rpm installer. This means if user builds any sample application, for example ―/usr/share/gnush_v0902_elf-1/samples/elf/EDK7145/app1‖, then objects and other files generated by ‗make' will not be removed automatically. User needs to remove these files manually.

Note : The user should have root access to system to install RPM in /usr/share directory. For this; user needs to login as root user or use ‗su' command to pseudo login as root.

2.1.5 Setup Test

Tools installation can be verified by the procedure mentioned below. Below explanation assumes that user has installed v0902 tools with all default selections during installation.

2.1.5.1 Verifying Installation on Windows

a. Go to Start menu->All Programs->GNUSHv0902-ELF->sh-elf Toolchain

b. This will open up a command prompt. Here the environment is set properly to use installed tools. Command prompt should show path as ―C:\Program Files\Renesas\Hew\Tools\KPIT Cummins\GNUSH-ELF\v0902‖.

c. Use the following command to go to samples program‘s directory.

```
#cd Samples\EDK7145\app1
```

Here, app1 is sample application for EDK7145 board.

d. Makefile is provided to build this application. Type _make‘ at command prompt to build application.

```
#make
```

e. Application should build without any error.

2.1.5.2 Verifying Installation on Linux

a. Use the following command to set path for installed tools

```
#export PATH=/usr/share/gnush_v0902_elf-1/bin/:$PATH
```

b. Change directory to EDK7145‘s app1 program using the following command

```
#cd                    /usr/share/gnush_v0902_elf-
1/samples/elf/EDK7145 /app1/
```

c. Use _make‘ command to build sample

```
#make
```

d. Application should build without any error.

2.1.6 Toolchain Directory Structure

The default path at which the toolchain gets installed is; C:\Program Files\Renesas\Hew\Tools\KPIT Cummins\ (If HEW is installed on the system)

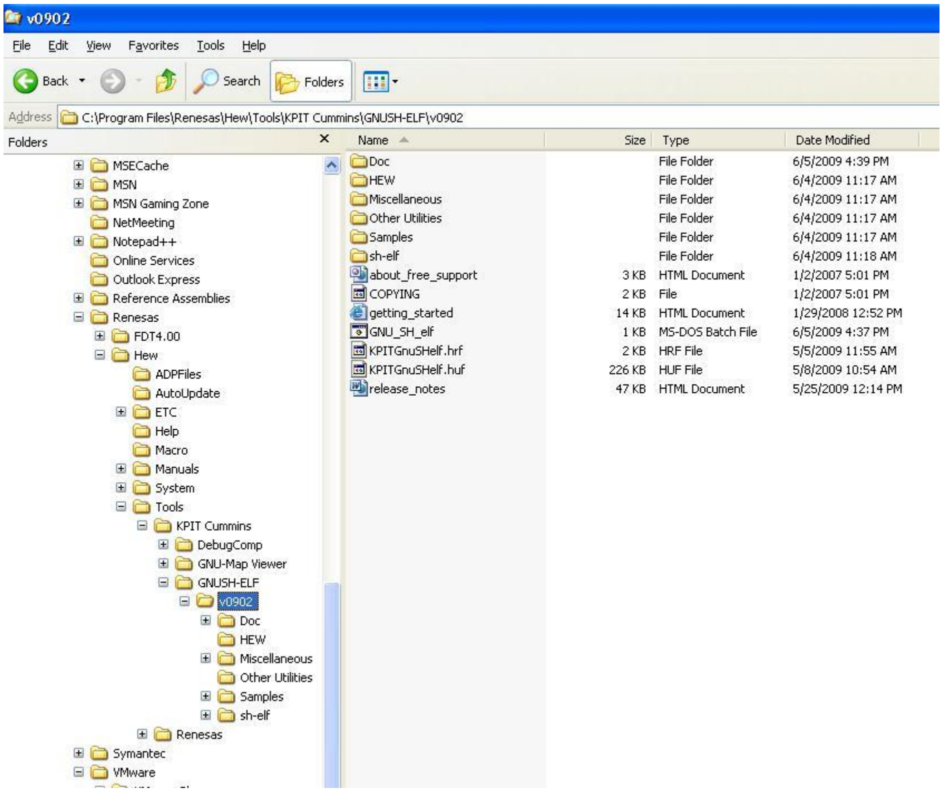
C:\Program Files\KPIT Cummins\

(If HEW is not installed on the system)

The toolchain gets installed at the above path (whichever applicable) in the folder GNUSH.

For example, if GNUSH v0902 toolchain has been installed, the default path at which the toolchain gets installed is;

C:\Program Files\Renesas\Hew\Tools\KPIT Cummins\GNUSH-ELF\v0902



C:\Program Files\Renesas\Hew\Tools\KPIT Cummins\GNUSH-ELF\v0902

Name	Size	Type	Date Modified
Doc		File Folder	6/5/2009 4:39 PM
HEW		File Folder	6/4/2009 11:17 AM
Miscellaneous		File Folder	6/4/2009 11:17 AM
Other Utilities		File Folder	6/4/2009 11:17 AM
Samples		File Folder	6/4/2009 11:17 AM
sh-elf		File Folder	6/4/2009 11:18 AM
about_free_support	3 KB	HTML Document	1/2/2007 5:01 PM
COPYING	2 KB	File	1/2/2007 5:01 PM
getting_started	14 KB	HTML Document	1/29/2008 12:52 PM
GNU_SH_elf	1 KB	MS-DOS Batch File	6/5/2009 4:37 PM
KPITGnuSHelf.hrf	2 KB	HRF File	5/5/2009 11:55 AM
KPITGnuSHelf.huf	226 KB	HUF File	5/8/2009 10:54 AM
release_notes	47 KB	HTML Document	5/25/2009 12:14 PM

The folders present at this path are as follows;

sh-elf

This is the gnush-elf toolchain folder. The contents of this folder are follows,

bin: This folder contains the binary utilities for gnush toolchain.

include: This folder contains header files.

lib: This folder contains the startup files, libgcc.a and libgcov.a files for SH targets (little endian as well big endian modes).

libexec: This folder contains the gcc and g++ executables.

share: This folder contains .mo files for language support.

sh-elf: This folder contains the standard C library archives for SH targets (little endian as well big endian modes).

Doc

This folder contains the GNU tools manuals for GCC, assembler, linker and also documents for FAQs, migration guide, etc.

HEW

This folder gets installed only if the toolchain is integrated with HEW during installation. It contains HEW specific files required for integration of toolchain with HEW.

Miscellaneous

This folder contains the lite versions of printf() & scanf() functions for the target SH7729SE.

Other Utilities

This folder contains other utilities like ‗make.exe', ‗diff.exe', ‗patch.exe', ‗rm.exe', etc.

Samples

This folder contains sample programs for some EVBs/ EDKs/ Solution engine boards of SH family.

Note : On Windows Vista and Windows 7 the GNUSH v0803 toolchain samples are installed at the following location,
"C:\Users\<Username>\Documents\KPIT Cummins\<Toolchain name>".

Besides the above-mentioned folders, the toolchain also installs the folders ‗DebugComp', ‗GNU-Map Viewer' and ‗GNUSH-Archive Editor' if the toolchain is integrated with HEW. The contents of these folders are as follows,

DebugComp

This folder contains HEW-specific files that are needed for debugging using HEW simulator.

GNU-Map Viewer

This folder contains the GNU Map viewer tool exe. GNU Map Viewer is a GUI based viewing tool (for windows OS only). GNU Map viewer is integrated with HEW and gets installed with the toolchain.

GNUSH-Archive Editor

This folder contains the GNU archive editor exe. The user can edit GNUSH archives through user friendly GUI support.

2.2 Using GNUSH on Command Line

Following two sections explain in details about how to use the GNUSH tools on command line for windows users as well as Linux users.

2.2.1 Using GNUSH on Command Line for Windows Users

The steps given below assume that GNUSH toolchain has already been installed.

a. Run the toolchain short cut from Windows ‗Start' menu.

b. Create a folder "test" and go to that folder.

Write a "Hello World!" program in file hello.c.

```
#include      <stdio.h>
int main(void)
    {
        int ret;
        ret = printf("Hello World!\r\n");
        return ret;
    }
```

c. Compile the program using the following command line:

```
#sh-elf-gcc hello.c -o hello.out -g
```

The above command will generate an executable file called hello.out. Sections ‗Controlling tools using GCC', ‗Using the GNUSH Assembler' and ‗Using the GNUSH Linker' give more information on the various options that can be passed to the compiler, assembler and linker while generating the executable.

d. Run the program using sh-elf simulator as follows;

```
#sh-elf-run hello.out
```

Execution of the program will print "Hello World!" string on the console.

2.2.2 Using GNUSH on Command Line for Linux Users

The steps given below assume that GNUSH toolchain has already been installed.

a. Export the path of the toolchain. For example;

```
$ export PATH=$PATH:/usr/share/gnush_v0902_elf-1/bin/
```

b. Create a folder "test" and go to that folder.

Write a "Hello World!" program in file hello.c.

```
#include <stdio.h>
int main(void)
  {
  int ret;
  ret = printf("Hello World!\r\n");
  return ret;
  }
```

c. Compile the program using the following command line;

```
$sh-elf-gcc hello.c –o hello.out -g
```

The above command will generate an executable file called hello.out.

d. Run the program using sh-elf simulator as follows;

```
$sh-elf-run hello.out
```

Execution of the program will print "Hello World!" string on the console.

2.3 Toolchain Components

The GNUSH toolchain supports number of components. The standard toolchain components for SH processor supported are listed in the Tables below.

The GNUSH toolchain is installed in its appropriate installation directory (For windows it is installed in <Installation Directory>\GNUSH<version>\sh-elf\bin whereas for LINUX it is installed in /usr/share/<Toolchain Directory>/bin). For example, the full path name for the GNUSH v0902 compiler on LINUX will be ‗/usr/share/gnush_v0902/bin/sh-elf-gcc'.

Component Name	Description
sh-elf-gcc	GNU Compiler Collection (GCC) with control to the C compiler
sh-elf-g++	C++ compiler (g++)
sh-elf-as	GNU assembler (as)
sh-elf-ld	GNU linker (ld)
sh-elf-run	Instruction set simulator
sh-elf- addr2line	Converts addresses into file names and line numbers
sh-elf-ar	Creates, modifies & extracts from object code archives
sh-elf-nm	Lists symbols from object files
sh-elf-objcpoy	Copies and translates object files
sh-elf-objdump	Displays information from object files
sh-elf- ranlib	Generates index to archive contents
sh-elf- -readelf	Displays information about ELF format object files
sh-elf- size	Lists file section sizes and total sizes
sh-elf-strings	Lists printable strings from files
sh-elf-strip	Strips debug symbols from binaries
sh-elf-gdb	Command Line GDB debugger
sh-elf-convrenesaslib	Convert Renesas library file to GNU Archive file
sh-elf-libgen	Generates library with the user specified options for Renesas targets
libc.a and libm.a	Unrestricted Newlib C and Math Libraries
libstdc++.a	GNU Standard C++ and Template Library

2.4 Using the GNUSH Tools

Refer to Section ―Using GNUSH on Command Line‖ to use GNUSH tools on command line. This section explains usage of GNUSH toolchain for Windows users as well as Linux users.

2.4.1 GNUSH Tools Description

The GCC program is actually a control program that executes the compiler components to produce the desired output, which is usually a compiled and linked executable program image. The GCC, however, is but a single component. It is actually a control program that calls other components that perform separate steps to create an executable binary.

The sequence of commands executed by a single invocation of GCC consists of the following stages:

- Preprocessing (to expand macros)
- Compilation (from source code to assembly language)
- Assembly (from assembly language to machine code)
- Linking (to create the final executable)

The components are described in following sections. The Figure: Compilation Flow Diagram below demonstrates the graphical representation for the same.

As an example, following sections examine these compilation stages individually using the same ‗Hello World' program ‗hello.c' described in the previous section.

2.4.1.1 Preprocessor

The first stage of the compilation process is to use GNU C preprocessor ‗sh-elf-cpp ', which is part of the GNUSH Tools package, to expand macros and header files. To perform this stage, GCC executes the following command:

```
$ sh-elf-cpp hello.c > hello.i
```

The result is a file ‗hello.i' which contains the source code with all macros expanded. By convention, preprocessed files are given the file extension ‗.i' for C programs and ‗.ii' for C++ programs. In practice, the preprocessed file is not saved to disk unless the option ‗-save-temps' is used.

2.4.1.2 Compiler

Next stage of the process is actual compilation of preprocessed source code to assembly language, for a specific processor. The command lin e option ‗-S' instructs sh-elf-gcc to convert the preprocessed C source code to assembly language without creating an object file:

```
$ sh-elf-gcc -Wall -S hello.i
```

The resulting assembly language is stored in the file ‗hello.s'. If assembly file is given as input the compiler passes it directly to the assembler.

2.4.1.3 Assembler

The purpose of GNUSH assembler (‗sh-elf-as') is to convert assembly language into machine code and generate an object file. When there are calls to external functions in the assembly source file, the assembler leaves the addresses of the external functions undefined, to be filled in later by the linker. The assembler can be invoked with the following command line:

```
$ sh-elf-as hello.s -o hello.o
```

As with GCC, the output file is specified with the option ‗-o'. The resulting file ‗hello.o' contains the machine instructions for the Hello World program, with an undefined reference to printf.

2.4.1.4 Linker

The final stage of compilation is linking of all the object files (newly compiled, and those specified as input) to create an executable. In practice, an executable requires many external functions from system and C run-time libraries.

GCC performs this entire linking process by internally invoking the GNUSH Linker ‗ld'.

The following command transparently performs the linking:

```
$ sh-elf-ld hello.o
```

This links the object file ‗hello.o' to the C standard library, and produces an executable file ‗a.out':

```
$ sh-elf-run a.out
```

Hello, world!

An object file for a C++ program can be linked to the C++ standard library in the same way with a single sh-elf-g++ command.

GCC normally invokes all of these compilation steps when converting a C source program into an executable. By using command line options for GCC, these steps may be invoked separately or in some combinations. This provides some flexibility when building large programs, or using assembly language sources, or debugging.

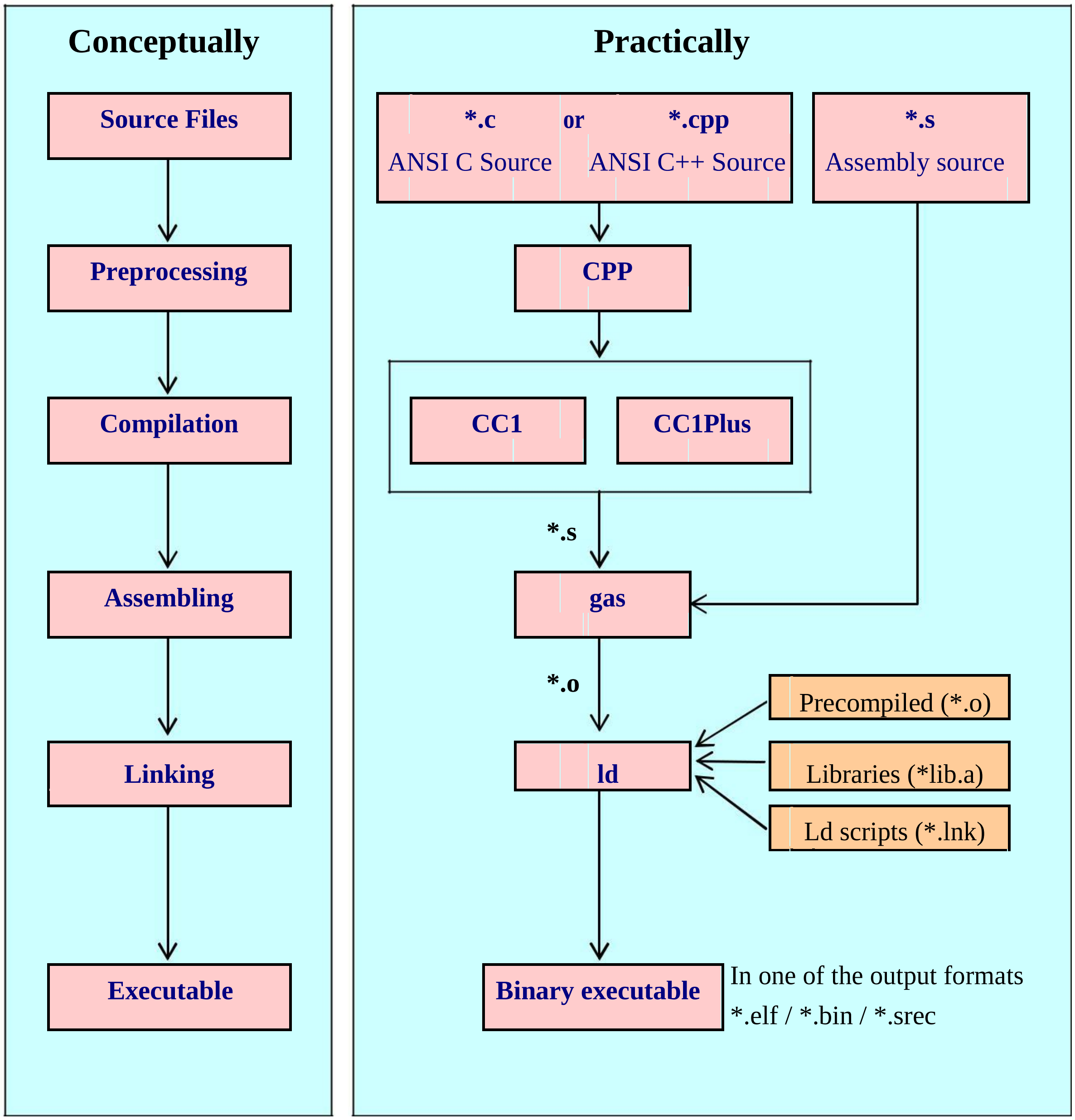


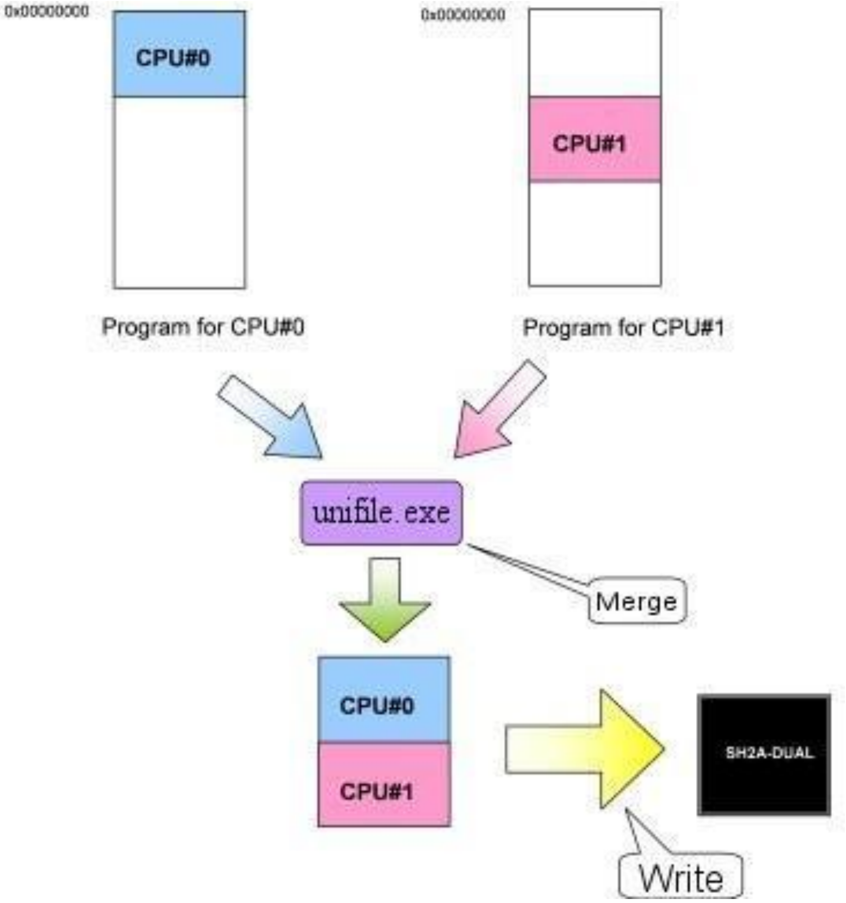
Figure: Compilation Flow Diagram

2.4.1.5 Unifile

Unifile is a utility that merges two or more Motorola format files (commonly known as Mot file or S-Record file) or Intel format files (commonly known as Hex file) to generate a single Motorola format file (S-Record / Mot file) or Intel format file (Hex file).

This utility can be used from the command line or it can be integrated into the Renesas HEW IDE by adding a custom build phase. This document describes how to use the tool.

A diagram showing how the Unifile utility is used is shown below:


0x00000000
CPU#0
0x00000000
CPU#1
Program for CPU#0
Program for CPU#1
unifile.exe
Merge
CPU#0
CPU#1
SH2A-DUAL
Write

2.4.2 Controlling tools using GCC

2.4.2.1 GCC Options Commonly Used

This section enlists the popularly used GNUSH compiler options.

`-c`

Compile or assemble the source files, but do not link. The linking stage is not done. The ultimate output is in the form of an object file for each source file.

`-o file`

Place output in the file file. This applies regardless to whatever sort of output is being produced, whether it is an executable file, an object file, an assembler file or preprocessed C code.

`-Wall`

It enables most of warning options starting with ‾-W‖.

`-g`

Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF 2). GDB can work with this debugging information. In case of GNUSH, default format is DWARF2.

`-O0, -O1, -O2, -O3 and -Os`

It enables various levels of optimization. Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.

`-fomit-frame-pointer`

Do not keep the frame pointer in a register for functions that do not need one. This avoids the instructions to save, set up and restore frame pointer; it also makes an extra register available in many functions. Moreover, it also makes debugging impossible on some machines.

`-m1`

Generate code for SH1.

`-m2`

Generate code for SH2.

`-m2e`

Generate code for SH2e.

`-m2a-nofpu`

Generate SH2a FPU-less code.

`-m2a-single-only`

Generate code for SH2a with a floating-point unit that only supports single-precision arithmetic.

`-m2a-single`

Generate code for the SH2a assuming the floating-point unit is in single-precision mode by default.

`-m2a`

Generate default double-precision SH2a-FPU code.

`-m3`

Generate code for SH3.

`-m3e`

Generate code for SH3e.

`-m4-nofpu`

Generate code for SH4 without a floating-point unit.

`-m4-single-only`

Generate code for SH4 with a floating-point unit that only supports single-precision arithmetic.

`-m4-single`

Generate code for SH4 assuming the floating-point unit is in single-precision mode by default.

`-m4`

Generate code for SH4.

`-m4a-nofpu`

Generate code for SH4al-dsp, or for a SH4a in such a way that the floating-point unit is not used.

`-m4a-single-only`

Generate code for SH4a with a floating-point unit that only supports single-precision arithmetic.

`-m4a-single`

Generate code for SH4a assuming the floating-point unit is in single-precision mode by default.

`-m4a`

Generate code for SH4a.

`-m4al`

This option is same as `-m4a-nofpu', except that it implicitly passes `-dsp' to the assembler. GCC does not generate any DSP instructions at the moment.

`-mb`

Compile code for the processor in big endian mode.

`-ml`

Compile code for the processor in little endian mode.

`-mrelax`

Shorten some address references at link time, when possible; uses the linker option `-relax'.

`-mbitops`

The bit instructions will be generated on enabling this command line option.

`-mrenesas`

Comply with the calling conventions defined by Renesas. Also enables assembler and linker option '-renesas'.

2.4.2.2 Compilation Options

This section details the other important options that are available with GCC. There are many, such options that control various details of compilation and optimization.

Displaying compiler behavior

`-c`

Compile or assemble the source files, but do not link. The linking stage is not done. The ultimate output is in the form of an object file for each source file.

`-o file`

Place output in the file *file*. This applies regardless to whatever sort of output is being produced, whether it is an executable file, an object file, an assembler file or preprocessed C code.

`-S`

Stop after the stage of compilation proper; do not assemble. The output is in the form of an assembler code file for each non-assembler input file specified.

`-E`

Stop after the preprocessing stage; do not run the compiler proper. The output is in the form of preprocessed source code that is sent to the standard output.

`-x` *language*

Specify explicitly the language for the following input files (rather than letting the compiler choose a default based on the file name suffix). This option applies to all following input files until the next `-x' option. Possible values for language are:

c, c-header, c-cpp-output, c++, c++-header, c++-cpp-output, assembler assembler-with-cpp.

`-x none`

Turn off any specification of a language, so that subsequent files are handled according to their file name suffixes (as they are if `-x' has not been used at all).

C Language Options

`-std=standard (c9x, gnu9x, c++98, gnu++98)`

Determine the language standard. This option is currently only supported when compiling C or C++.

`-ansi`

In C mode, support all ISO C90 programs. In C++ mode, remove GNU extensions that conflict with ISO C++.

`-funsigned-char`

Enables `unsigned char` as default char type.

`-fsigned-char`

Enables `signed char` as default char type.

`-fshort-enums`

Allocate to an enum type only as many bytes as it needs for the declared range of possible values. Specifically, the enum type will be equivalent to the smallest integer type which has enough room.

Warning: the `` `-fshort-enums' `` switch causes GCC to generate code that is not binary compatible with code generated without that switch. Use it to conform to a non-default application binary interface.

`-fno-builtin -fno-`

`builtin-`*`function`*

Don't recognize built-in functions that do not begin with `` `__builtin_' `` as prefix.

GCC normally generates special code to handle certain built-in functions more efficiently; for instance, calls to `alloca` may become single instructions that adjust the stack directly, and calls to memcpy may become inline copy loops. The resulting code is often both smaller and faster, but since the function calls no longer appear as such, you cannot set a breakpoint on those calls, nor can you change the behavior of the functions by linking with a different library. In addition, when a function is recognized as a built-in function, GCC may use information about that function to warn about problems with calls to that function, or to generate more efficient code, even if the resulting code still contains calls to that function. For example, warnings are given with `` `-Wformat' `` for bad calls to `printf`, when `printf` is built in, and `strlen` is known not to modify global memory.

With the `-fno-builtin-*function*' option only the built-in function *function* is disabled. *function* must not begin with `__builtin_'. If a function is named that is not built-in in this version of GCC, this option is ignored. There is no corresponding `-fbuiltin-*function*' option; if you wish to enable built-in functions selectively when using `-fno-builtin' or `-ffreestanding', you may define macros such as:

```
#define abs(n)          __builtin_abs ((n))
#define strcpy(d, s)    __builtin_strcpy ((d), (s))
```

-fno-asm

Do not recognize asm, inline or typeof as a keyword, so that code can use these words as identifiers. You can use the keywords __asm__, __inline__ and __typeof__ instead. `-ansi' implies `-fno-asm'.

In C++, this switch only affects the typeof keyword, since asm and inline are standard keywords. You may want to use the `-fno-gnu-keywords' flag instead, which has the same effect. In C99 mode (`-std=c99' or `-std=gnu99'), this switch only affects the asm and typeof keywords, since inline is a standard keyword in ISO C99.

@*file*

Read command-line options from *file*. The options read are inserted in place of the original @*file* option. Options in *file* are separated by white space. The *file* may itself contain additional @*file* options; any such options will be processed recursively.

-pedantic

Issue all the mandatory diagnostics listed in the C standard.

Preprocessor options

-Xpreprocessor *option*

Pass *option* as an option to the preprocessor. You can use this to supply system-specific preprocessor options that GCC does not know how to recognize. If you want to pass an option that takes an argument, you must use `-Xpreprocessor' twice, once for the option and once for the argument.

-D *name*

Predefine *name* as a macro, with definition 1.

-D *name=definition*

The contents of *definition* are tokenized and processed as if they appeared during translation phase three in a `#define' directive. In particular, the definition will be truncated by embedded newline characters.

`-Wall`

Turns on all optional warnings that are desirable for normal code

`-Werror`

Make all warnings into hard errors.

`-w`

Suppress all warnings.

Options to Specify Libraries, Paths and Startup Files

`-I`*`dir`*

Add the directory *dir* to the head of the list of directories to be searched for header files. This can be used to override a system header file, substituting your own version, since these directories are searched before the system header file directories. However, you should not use this option to add directories that contain vendor-supplied system header files (use `-isystem' for that). If you use more than one `-I' option, the directories are scanned in left-to-right order; the standard system directories come after.

`-include` *`file`*

Process *file* as if `#include "file"` appeared as the first line of the primary source file. However, the first directory searched for *file* is working directory of the preprocessor *instead of* the directory containing the main source file. If not found there, it is searched for in the remainder of the `#include "..."` search chain as normal. If multiple `` `-include' `` options are given, the files are included in the order they appear on the command line.

`-nostdinc`

Do not search the standard system directories for header files. Only the directories you have specified with `-I' options (and the directory of the current file, if appropriate) are searched.

`-nostdinc++`

Do not search for header files in the C++-specific standard directories, however, search the other standard directories. (This option is used when building the C++ library.)

`-L`*`dir`*

Add directory *dir* to the list of directories to be searched for `-l'.

`-specs=`*`file`*

Process *file* after the compiler reads in the standard `` `specs' `` file, in order to override the defaults that the `` `gcc' `` driver program uses when determining what switches to pass to `` `cc1' ``, `` `cc1plus' ``, `` `as' ``, `` `ld' ``, etc. More than one `` `-specs= ``*`file`*`'` can be specified on the command line, and they are processed from left to right.

`-nostartfiles`

Do not use the standard system startup files when linking. The standard system libraries are used normally, unless `` `-nostdlib' `` or `-nodefaultlibs'is used.

`-nostdlib`

Do not use the standard system startup files or libraries when linking. No startup files and only the libraries you specify will be passed to the linker. The compiler may generate calls to `memcmp`, `memset`, `memcpy` and `memmove`. These entries are usually resolved by entries in libc. These entry points should be supplied through some other mechanism when this option is specified.

`-nodefaultlibs`

Do not use the standard system libraries when linking. Only the libraries you specify will be passed to the linker. The standard startup files are used normally, unless `` `-nostartfiles' `` is used. The compiler may generate calls to `memcmp`, `memset`, `memcpy` and `memmove`. These entries are usually resolved by entries in libc. These entry points should be supplied through some other mechanism when this option is specified.

`-static`

On systems that support dynamic linking, this prevents linking with the shared libraries. On other systems, this option has no effect.

`-shared`

Produce a shared object that can then be linked with other objects to form an executable. Not all systems support this option.

`-l`*`library`*

Search the library named *library* when linking. The linker searches a standard list of directories for the library, which is actually a file named `lib*library*.a'. The linker then uses this file as if it had been specified precisely by name. The directories searched include several standard system directories plus any that you specify with `-L'. It makes a difference where in the command user write this option; the linker searches and processes libraries and object files in the order they are specified. Thus, `foo.o -lz bar.o' searches library `z' after file `foo.o' but before `bar.o'. If `bar.o' refers to functions in `z', those functions may not be loaded.

Debugging Options

`-g`*`level`*

Request debugging information and also use *level* to specify how much information. The default level is 2. Level 1 produces minimal information, enough for making backtraces in parts of the program that user do not plan to debug. This includes descriptions of functions and external variables, but no information about local variables and no line numbers. Level 3 includes extra information, such as all the macro definitions present in the program. Some debuggers support macro expansion when you use `-g3'.

`-gdwarf-2`

Produce debugging information in DWARF version 2 format (if that is supported). With this option, GCC uses features of DWARF version 3 when they are useful; version 3 is upward compatible with version 2, but may still cause problems for older debuggers.

`-save-temps`

Store the usual "temporary" intermediate files permanently; place them in the current directory and name them based on the source file. Thus, compiling `` `foo.c' `` with `` `-c -save-temps' `` would produce files `` `foo.i' `` and `` `foo.s' ``, as well as `` `foo.o' ``. This creates a preprocessed `` `foo.i' `` output file even though the compiler now normally uses an integrated preprocessor.

`-feliminate-unused-debug-types`

Normally, when producing DWARF2 output, GCC will emit debugging information for all types declared in a compilation unit, regardless of whether or not they are actually used in that compilation unit. Sometimes this is useful, such as if, in the debugger, you want to cast a value to a type that is not actually used in your program (but is declared). More often, however, this results in a significant amount of wasted space. With this option, GCC will avoid producing debug symbol output for types that are nowhere used in the source file being compiled.

`-d`*`letters`*

This is used for debugging the RTL-based passes of the compiler. The file names for most of the dumps are made by appending a pass number and a word to the *dumpname*. *dumpname* is generated from the name of the output file, if explicitly specified and it is not an executable, otherwise it is the basename of the source file. Most debug dumps can be enabled either passing a *`letter`* to the `` `-d' `` option, or with a long `` `-fdump-rtl' `` switch.

Passing Options to the Assembler or Linker

`-Wa,`*option*

Pass *option* as an option to the assembler. If *option* contains commas, it is split into multiple options at the commas.

`-Xassembler` *option*

Pass *option* as an option to the assembler. You can use this to supply system-specific assembler options that GCC does not know how to recognize. If you want to pass an option that takes an argument, you must use `-Xassembler' twice, once for the option and once for the argument.

`-Wl,`*`option`*

Pass *option* as an option to the linker. If *option* contains commas, it is split into multiple options at the commas.

`-Xlinker` *`option`*

Pass *option* as an option to the linker. You can use this to supply system -specific linker options that GCC does not know how to recognize. If you want to pass an option that takes an argument, you must use `` `-Xlinker' `` twice, once for the option and once for the argument.

2.4.2.3 Optimization Options

This section enlists some commonly used optimization options as well as SH specific optimization options.

`-O`

`-O1`

Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function. With option `-O', the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time. `-O' also turns on `-fomit-frame-pointer' on machines where doing so does not interfere with debugging.

`-O2`

Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. The compiler does not perform loop unrolling or function inlining when you specify `-O2'. As compared to `-O', this option increases both, compilation time and performance of the generated code.

`-O2' turns on all optimization flags specified by `-O'.

`-O3`

Optimize yet more. `-O3' turns on all optimizations specified by `-O2' and also turns on the `-finline-functions', `-funswitch-loops' and `-fgcse-after-reload' options.

`-Os`

Optimize for size. ` -Os' enables all `-O2' optimizations that do not typically increase code size. It also performs further optimizations designed to reduce code size. It enables option `-fsort-data'. `-Os' disables the following optimization flags:

```
{-falign-functions -falign-jumps -falign-loops -falign-
labels -freorder-blocks -freorder-blocks-and-partition -
fprefetch-loop-arrays -ftree-vect-loop-version}
```

If you use multiple `-O' options, with or without level numbers, the last such option is the one that is effective.

`-fomit-frame-pointer`

Do not keep the frame pointer in a register for functions that don't need one. This avoids the instructions to save, set up and restore frame pointer; it also makes an extra register available in many functions. Moreover, it also makes debugging impossible on some machines.

-fdata-sections -

ffunction-sections

Place each function or data item into its own section in the output file if the target supports arbitrary sections. The name of function or name of the data item determines name of the in the output file.

-funroll-loops

Unroll loops whose number of iterations can be determined at compile time or upon entry to the loop. `-funroll-loops' implies `-frerun-cse-after-loop', `-fweb' and `-frename-registers'. It also turns on complete loop peeling (i.e. complete removal of loops with small constant number of iterations). This option makes code larger, and may or may not make it run faster.

-funroll-all-loops

Unroll all loops, even if their number of iterations is uncertain when the loop is entered. This usually makes programs run more slowly. `-funroll-all-loops' implies the same options as `-funroll-loops'.

-ffast-math

This option causes the preprocessor macro __FAST_MATH__ to be defined. This option should never be turned on by any `-O' option since it can result in incorrect output for programs which depend on an exact implementation of IEEE or ISO rules/specifications for math functions.

Following are some of the SH specific optimization options,

-fsort-data

Sorts initialized data variables and constant data based on their alignment and place in separate sections. The N(1/2/4) byte aligned data will be placed either in section ".data.alignN" or in ".rodata.alignN" depending on read-only attribute. The COFF toolchain ignores this option. This option will be ignored if used along with option `-fdata-sections'. Enabled at level `-Os'.

-mrelax

Shorten some address references at link time, when possible; uses the linker option `-relax'. A new 'optimize' attribute has been added from GCC-4.4 onwards to allow programmers to change the optimization level and particular optimization options for an individual function. Sample use of this attribute is shown below,
int foo(int i) __attribute__((optimize("-O3")));

2.4.3 Using the GNUSH Assembler

The GNU assembler, ―as‖ is intended to assemble the output of GNU C compiler to be use by linker. The GNU assembler can be configured to produce several alternative object file formats.

2.4.3.1 Invoking ‗as‘

The GNU assembler ‗as‘, translates text-based assembly language source files into binary-based object files. Normally it operates without you being aware of it, as the compiler driver program (gcc) invokes it automatically. If, while using gcc, you want to pass special command-line options to the assembler to control its behavior, you need to use the `-Wa,<text>' command-line option. This option passes <text> directly to the assembler's command line.

For example:

```
#gcc –c –g –o –Wa,-alh, -L file.c
```

It passes the ‗-alh ‘ command line option on to the assembler. (This causes the assembler to emit a listing file that shows the assembler source generated by the compiler for each line of the C source file *file*.c).

However, if you are creating your own assembler source files, you must invoke as directly. Below is brief summary of how to invoke assembler:

```
as [-a [cdhlns][=file]] [-D] [--defsym
sym=val] [-f] [--gstabs] [--gdwarf2] [-help]
[-I dir] [-J] [-K] [-L]
[--listing-lhs-width=NUM] [--listing-lhs-width2=NUM]
[--listing-rhs-width=NUM] [--listing-rhs-width2=NUM]
[--keep-locals] [-o objfile] [-R] [--statics] [-v] [-
version] [--version] [-W] [--warn] [--fatal-warnings]
[-w] [-x] [-Z] [--target-help] [target-options] [-h-
tick-hex]
```

Description of the above options is mentioned below:

```
-a[cdhlns]
```

Turn on the listings in any of the variety ways:

```
-ac : Omit false conditional -ad :
Omit debugging directives -ah :
Include high level source -al :
Include assembly
```

-an : Omit forms processing -

as : Include symbols --defsym

sym=value

Define symbol sym to be value before assembling the input file. Value must be an integer constant.

-f

¯fast‖ -skip whitespace and comment preprocessing (assume source is compiler output).

--gstabs

Generate stabs debugging information for each assembler line. This option is only supported by some targets, not all of them.

--gdwarf2

Generate DWARF2 debugging information for each assembler line. This option is only supported by some targets, not all of them.

--help

Print a summary of the command line options and exit.

--target-help

Print a summary of all target specific options and exit.

-I dir

Add directory dir to the search list for .include directives.

-J

Don‘t warn about signed overflow.

-K

Issue warnings when difference tables altered for long displacements

-L

Labels beginning with ‗L‘are called local labels. This option tells ¯as‖ to retain the ‗L‘ symbols in object file.

--keep-locals

Keep local symbols in symbol table.

--listing-lhs-width=NUM

Set the maximum width, in words, of the output data column for an assembler listing to number.

--listing-lhs-width2=NUM

Set the maximum width, in words, of the output data column for continuation lines in assembler listings to number.

`--listing-rhs-width=number`

Set the maximum width of an input source line, as displayed in a listing, to number bytes.

`-o objfile`

Name the object file output from as objfile.

`-R`

Fold the data section into text section.

`Statistics`

Print the maximum space (in bytes) and total time (in seconds) used by assembly

`--strip-local-absolute`

Remove local absolute symbols from the outgoing symbol table.

`-version`

Print the ¯as‖ version.

`--version`

Print the ¯as‖ version and exit.

`--warn`

Don‘t suppress warning messages or treat them as errors.

`--fatal-warnings`

Treat warnings as errors.

2.4.3.2 Assembler Options

This describes the command line options available in all versions of the GNUSH assembler.

`Include search path: ‗I‛ path`

Use this option to add a path to the list of directories as searches for files specified in .include directives. Use of ‗I‘ as many times as necessary to include a variety of paths. The current working directory is always search first, after that, ‖as‖ searches any ‗I‘ directories in the same way as they were specified on the command line.

`Dependency Tracking: ‗--MD‛`

¯as‖ can generate dependency file for the file it creates. This file consists of a single rule suitable for make describing the dependencies of the main source file. The rule is written to the file named in its argument. This feature is used in the automatic updating of makefiles.

`Name the object file: ‗-o‛`

There is always one object file output when ¯as‖ runs. By default it has name 'a.out'. This option is used to give object file a different name. Whatever the object file is called, ‖as‖ overwrites any existing file of the same name.

`Compatible Output: ‗—traditional-format‘`

For some targets, the output of ¯as‖ is different in some ways from the output of some existing assembler. This option requests ¯as‖ to use traditional format instead. E.g. it disables exception frame optimization which ¯as‖ normally does by default on gcc output.

`Control warnings: ‗-W‘,‘—no-warn‘`

If use ¯-W‘ and ‗–no-warn‘ options, no warnings are issued. This only affects the warning messages. Errors, which stops the assembly are still reported.

2.4.3.3 SH Dependent Features

The machine instruction sets are different on each machine where ¯as‖ runs. Floating point representations vary as well, and ¯as‖ often support a few additional directives or command line options with compatibility with other assemblers on particular platforms.

`Options`

¯as‖ has following command line options for SH family:

`-little`

Generate little endian code.

`-big`

Generate big endian code.

`-relax`

After jump instructions for long displacements

`-h-tick-hex`

Support H'00 style hex constants in addition to 0x00 style

`-small`

Align sections to 4 bytes boundaries, not 16 bytes.

`-dsp`

Enable sh-dsp instructions, and disables sh3e/sh4 instructions.

`-renesas`

Disable optimization with section symbol for compatibility with Renesas assembler.

`-isa=sh4 | sh4a`

Specify the sh4 or sh4a instruction set

`-isa=fp`

Enable sh2e, sh3e and sh4a instruction sets.

-isa=all

Enable sh1, sh2, sh2e, sh3, sh3e, sh4, sh4a and sh-dsp instruction sets.

Syntax

Special Characters

‗!‘ is a line comment character.

You can use ‗;‘ instead of a newline to separate statements.

Since ‗$‘ has no special meaning, you may use it in symbol names.

Register Names:

You can use the predefined symbols ‗r0‘, ‗r1‘, ‗r2‘, ‗r3‘, ‗r4‘, ‗r5‘, ‗r6‘, ‗r7‘, ‗r8‘, ‗r9‘, ‗r10‘, ‗r11‘, ‗r12‘, ‗r13‘, ‗r14‘ and ‗r15‘ to refer to the SH registers.

The SH also has following control registers:

pr	:	Procedure registers (Holds return address)
pc	:	Program counter
match macl	:	High and low multiply accumulator register
sr	:	Status register
gbr	:	Global base register
vbr	:	Vector base register (for interrupt vectors)

Addressing Modes:

―as‖ understands following addressing modes for SH.

Rn in following refers any of the numbered register but not the control register.

Rn	:	Register direct
@Rn	:	Register indirect
@-Rn	:	Register indirect with pre-decrement
@Rn+	:	Register indirect with post-increment
@ (disp, Rn)	:	Register indirect with displacement
@ (R0, Rn)	:	Register indexed
@ (disp, GBR)	:	GBR offset
@ (R0, GBR)	:	GBR indexed

Addr,@ (disp, pc): PC relative address (for branch or for addressing memory). The ―as‖ implementation allows you to use the simpler form addr anywhere a PC relative address is called for; the alternate form is supported for compatibility with other assemblers.

#imm : Immediate data

Floating Point

SH2E, SH3E, SH4E groups have on-chip floating point unit. Other SH group has no hardware floating point, but the .float directive generates IEEE floating-point numbers for compatibility with other development tools.

SH2E and SH3E support single-precision floating point calculations as well as entirely PCAPI compatible emulation of double-precision floating point calculations. SH2E and SH3E instructions are a subset of the floating point calculations conforming to the IEEE-754 standard.

In addition to single and double-precision floating point operation capability, the on-chip FPU on SH4 has a 128-bit graphic engine that enables 32-bit floating point data to be processed 128 bits at a time. It also supports 4*4 array operations and inner product operations. Also, a superscalar architecture is employed that enables simultaneous execution of 2 instructions (including floating point instructions), providing performance of up to twice that of conventional architectures at the same frequency.

SH Machine Directives:

uaword
ualong

¯as‖ will issue a warning when a misaligned .word or .long directive is used. You may used .uaword or .ualong to indicate that the value is intentionally misaligned.

Opcode:

¯as‖ implements all the standard SH opcodes. No additional pseudo-instructions are needed on this family. Since ¯as‖ supports a simpler form of PC-relative addressing, you may simply write (for example)

mov.l bar, r0

Where other assemblers might require an explicit displacement to bar from the program counter:

mov.l @ (disp, PC)

Summary of SH opcodes is as

follows: Legend:

Rn : Numbered register

Rm : Another numbered register #imm :

Immediate data

disp : Displacement disp8 : 8-bit

displacement

disadd #imm,Rn	lbs. @Rn+,PR
add Rm,Rn	mac.w @Rm+,@Rn+

addc Rm,Rn

addv Rm,Rn

and #imm,R0

and Rm,Rn

and.b #imm,@(R0,GBR)

bf disp8

bra disp12

bsr disp12

bt disp8

clrmac

clrt

cmp/eq #imm,R0

cmp/eq Rm,Rn

cmp/ge Rm,Rn

cmp/gt Rm,Rn

cmp/hi Rm,Rn

cmp/hs Rm,Rn

cmp/pl Rn

cmp/pz Rn

cmp/str Rm,Rn

div0s Rm,Rn

div0u

div1 Rm,Rn

exts.b Rm,Rn

exts.w Rm,Rn

extu.b Rm,Rn

extu.w Rm,Rn

jmp @Rn

jsr @Rn

ldc Rn,GBR

ldc Rn,SR

ldc Rn,VBR

mov #imm,Rn

mov Rm,Rn

mov.b Rm,@(R0,Rn)

mov.b Rm,@-Rn

mov.b Rm,@Rn

mov.b @(disp,Rm),R0

mov.b @(disp,GBR),R0

mov.b @(R0,Rm),Rn

mov.b @Rm+,Rn

mov.b @Rm,Rn

mov.b R0,@(disp,Rm)

mov.b R0,@(disp,GBR)

mov.l Rm,@(disp,Rn)

mov.l Rm,@(R0,Rn)

mov.l Rm,@-Rn

mov.l Rm,@Rn

mov.l @(disp,Rn),Rm

mov.l @(disp,GBR),R0

mov.l @(disp,PC),Rn

mov.l @(R0,Rm),Rn

mov.l @Rm+,Rn

mov.l @Rm,Rn

mov.l R0,@(disp,GBR)

mov.w Rm,@(R0,Rn)

mov.w Rm,@-Rn

mov.w Rm,@Rn

mov.w @(disp,Rm),R0

mov.w @(disp,GBR),R0

mov.w @(disp,PC),Rn

mov.w @(R0,Rm),Rn

mov.w @Rm+,Rn

mov.w @Rm,Rn

```
ldc.l @Rn+,GBR
ldc.l @Rn+,SR
ldc.l @Rn+,VBR
lds Rn,MACH
lds Rn,MACL
lds Rn,PR
lds.l @Rn+,MACH
lds.l @Rn+,MACL
nop
not Rm,Rn
or #imm,R0
or Rm,Rn
or.b #imm,@(R0,GBR)
rotcl Rn
rotcr Rn
rotl Rn
rotr Rn
rte
rts
sett
shal Rn
shar Rn
shll Rn
shll16 Rn
shll2 Rn
shll8 Rn
shlr Rn
shlr16 Rn
shlr2 Rn
shlr8 Rn
sleep
stc GBR,Rn
mov.w R0,@(disp,Rm)
mov.w R0,@(disp,GBR)
mova @(disp,PC),R0
movt Rn
muls Rm,Rn
mulu Rm,Rn
neg Rm,Rn
negc Rm,Rn
stc VBR,Rn
stc.l GBR,@-Rn
stc.l SR,@-Rn
stc.l VBR,@-Rn
sts MACH,Rn
sts MACL,Rn
sts PR,Rn
sts.l MACH,@-Rn
sts.l MACL,@-Rn
sts.l PR,@-Rn
sub Rm,Rn
subc Rm,Rn
subv Rm,Rn
swap.b Rm,Rn
swap.w Rm,Rn
tas.b @Rn
trapa #imm
tst #imm,R0
tst Rm,Rn
tst.b #imm,@(R0,GBR)
xor #imm,R0
xor Rm,Rn
xor.b #imm,@(R0,GBR)
xtrct Rm,Rn
```

stc SR,Rnp12 : 12-bit displacement

2.4.4 Using the GNUSH Linker

Linker combines a number of objects and archive files, relocates their data and ties up symbol references. Usually the last step in compiling the program is to run ―ld‖. ―ld‖ accepts Linker Command Language files written in a superset of AT&T‘s Link Editor Command Language syntax, to provide explicit and total control over the linking process. This version of ―ld‖ uses the general purpose BFD libraries to operate on object files. This allows ―ld‖ to read, combine and write object files in many different formats, e.g. COFF or 'a.out'. Many different formats may be linked together to produce any available kind of object file.

2.4.4.1 Invoking ‗ld‘

The GNU linker ―ld‖ is meant to cover a broad range of situations and to be as compatible as possible with other linkers. For instance a frequent use of ―ld‖ is to link standard UNIX object files on a standard supported UNIX system. On such a system to link a file hello.o:

```
#ld –o output /lib/crt0.o hello.o -lc
```

This tells ―ld‖ to produce a file called output as the result of linking the file /lib/crt0.o with hello.o and the library libc.a, which will come from standard search directories.

Usually linker is invoked with at least one object file, but you can specify other forms of binary input files using ‗-l‘,‘-R‘ and the script command language. If no binary input files at all specified, the linker does not produce any output and issuing the message ―No input files‖.

If the linker is being invoked indirectly, via a gcc then all the linker command line options should be prefixed by ‗-Wl‘ (or whatever is appropriate for the particular gcc) like this :

```
#gcc –Wl,--startgroup foo.o bar.o –Wl,--endgroup
```

This is important because otherwise the compiler driver program may silently drop the linker options, resulting in a bad link.

2.4.4.2 Linker Options

Following are the command line options accepted by the GNU linker.

```
ld [ -o output ]  objfile...
  [-Aarchitecture] [-b input-format] [-
  Bstatic] [-Bdynamic] [-Bsymbolic] [-c
  MRI-commandfile] [-d | -dc | -dp] [-
  defsym symbol=expression]
  [-dynamic-linker file ] [-embedded-relocs ]
  [-e entry]  [-F]  [-F format]
  [-format input-format] [-g] [-G size] [-
  help] [-i ] [-larchive ] [-Lsearchdir] [-M]
  [-Map mapfile] [-m emulation]
  [-N | -n] [-noinhibit-exec ] [-no-keep-memory ]
  [-oformat output-format] [-R filename ]
  [-relax] [-retain-symbols-file filename ]
  [-r | -Ur] [-rpath dir] [-rpath-link dir ]
  [-S] [-s] [-soname name] [-shared] [-sort-
  common ] [-stats] [-T commandfile] [-Ttext
  org] [-Tdata org]
  [-Tbss org]  [-t ] [-traditional-format ]
  [-u symbol] [-V] [-v] [-verbose] [-version] [-
  warn-common] [-warn-constructors] [-warn-once ]
  [-y symbol] [-X] [-x]
  [-([archives] -)]
  [--start-group [archives] --end-group] [-
  split-by-reloc count ] [-split-by-file ]
  [--whole-archive]
```

-b input-format --
format=input-format

¯ld‖ may be configured to support more than one kind of object file. If ¯ld‖ is configured this way, you can use the ‗ -b‘ option to specify binary format for input object files that follow this option on the command line. Even wh en ¯ld‖ is configured to support alternative object formats, you don‘t usually need to specify this, as ¯ld‖ should be configured to expect as default input format the most usual format on each machine. input-format is a text string, the name of a particular format supported by the BFD libraries.

You can also use ‗-b‘ to switch formats explicitly (when linking object files of different formats), by including ‗-b input-format‘ before each group of object files in a particular format.

-Bstatic

Do not link against shared libraries. This is only meaningful on platforms for which shared libraries are supported.

-Bdynamic

Link against dynamic libraries. This is only meaningful on platforms for which shared libraries are supported. This option is normally the default on such platforms.

`-Bsymbolic`

When creating a shared library, bind references to global symbols to the definition within the shared library, if any. Normally, it is possible for a program linked against a shared library to override the definition within the shared library. This option is only meaningful on ELF platforms which support shared libraries.

`-c MRI-commandfile`

For compatibility with linkers produced by MRI, ld accepts script files written in an alternate, restricted command language. Introduce MRI script files with the option `-c'; use the `-T' option to run linker scripts written in the general-purpose ld scripting language. If MRI-cmdfile does not exist, ld looks for it in the directories specified by any `-L' options.

`-d`

`-dc`

`-dp`

These 3 options are equivalent; multiple forms are supported for compatibility with other linkers. They assign space to common symbols even if the relocatable output file is specified (with ‗-r‘).

`-defsym symbol=expression`

Create a global symbol in the output file, containing the absolute address given by expression. You may use this option as many times as necessary to define multiple symbols in the command line. A limited form of arithmetic is supported for the expression in this context: you may give a hexadecimal constant or the name of an existing symbol, or use + and - to add or subtract hexadecimal constants or symbols. If you need more elaborate expressions, consider using the linker command language from a script.

Note: there should be no white space between symbol, the equals sign ("="), and expression.

`-embedded-relocs`

This option is only meaningful when linking MIPS embedded PIC code, generated by the -membedded-pic option to the GNU compiler and assembler. It causes the linker to create a table which may be used at runtime to relocate any data which was statically initialized to pointer values. See the code in testsuite/ld-empic for details.

`-e entry --`

`entry=entry`

Use entry as the explicit symbol for beginning execution of your program, rather than the default entry point.

`-E --export-`

`dynamic`

When creating a dynamically linked executable, add all symbols to the dynamic symbol table. The dynamic symbol table is the set of symbols which are visible from dynamic objects at run time. If you don‘t use this option, the dynamic symbol table will normally contain only those symbols which are referenced by some dynamic object mentioned in the link.

`-EB`

Link big-endian objects. This affects the default output format.

`-EL`

Link Little-endian objects. This affects the default output format.

`-F -`

`Fformat`

Ignored. Some older linkers used this option throughout a compilation toolchain for specifying object-file format for both input and output object files. The mechanisms ld uses for this purpose (the `-b' or `-format' options for input files, `-oformat' option or the TARGET command in linker scripts for output files, the GNUTARGET environment variable) are more flexible, but ld accepts the `-F' option for compatibility with scripts written to call the old linker.

`-format input-format`

Synonym for `-b input-format'.

`-f`

`--auxiliary name`

When creating an ELF shared object, set the internal DT_AUXILIARY field to the specified name. This tells the dynamic linker that the symbol table of the shared object should be used as an auxiliary filter on the symbol table of the shared object name.

If you later link a program against this filter object, then, when you run the program, the dynamic linker will see the DT_AUXILIARY field. If the dynamic linker resolves any symbol from the filter object, it will first check whether there is a definition in the shared object name. If there is one, it will be used instead of the definition in the filter object. The shared object name need not exist. Thus the shared object may be used to provide an alternative implementation of certain function, perhaps for debugging or for machine specific performance. This option may be specified more than once. The DT_AUXILIARY entries will be created in the order in which they appear on the command line.

`-F name --`

`filter name`

When creating an ELF shared object, set the internal DT_FILTER field to the specified name. This tells the dynamic linker that the symbol table of the shared object which is being created should be used as a filter on the symbol table of the shared object name.

If you later link the program against this filter object, then, when you run the program the dynamic linker see the DT_FILTER filed. The dynamic linker will resolve symbols according to the symbol table of the filter object as usual, but it will actually link to the definitions found in the shared object name. Thus the filter object can be used to select the subset of the symbols provided by the object name. Some older linkers used the ‗-F' option throughout a compilation toolchain for specifying object file format for both input and output object files. The GNU linker uses other mechanisms for this purpose: the ‗-b',‘–format', ‗–oformat' options, the TARGET command in the linker script and the GNUTARGET environment variable. The GNU linker will ignore the ‗-F' option when not creating an ELF shared object.

`-fini name`

When creating an ELF executable or shared object, call NAME when the executable or shared object is unloaded, by setting DT_FINI to the address of the function. By default, the linker uses the _fini as the function to call.

`-Gvalue`

`-G value`

`--gpsize=value`

Set the maximum size of objects to be optimized using the GPR to size. This is only meaningful for object file formats such as MIPS ECOFF which supports putting large and small objects into different sections. This is ignored for other file formats.

`-help`

Print a summary of the command-line options on the standard output and exit.

`-hname -`

`soname=name`

When creating an ELF shared object, set the internal DT_SONAME field to the specified name. When an executable is link with a shared object which has a DT_SONAME field, then when the executable is run the dynamic linker will attempt to load the shared object specified by the DT_SONAME field rather than the using the file name given to the linker.

`-i`

Perform an incremental link (same as option ‗-r‘).

`-init name`

When creating an ELF executable or shared object, call NAME when executable or shared object is loaded, by setting DT_INIT to the address of the function. By default, the linker uses _init as the function to call.

`-larchive --`

`library=archive`

Add archive file archive to the list of files to link. This option may be used any number of times. ―ld‖ will search its path-list for occurrences of ‗libarchive.a‘ for every archive specified. On support which shared support libraries, ―ld‖ may also search for libraries with extensions other than ‗.a‘. Specifically, on ELF and SunOS systems, ―ld‖ will search a directory for a library with an extension of ‗.so‘ before searching for one with an extension of ‗.a‘. By convention, a ‗.so‘ extension indicates a shared library.

The linker will search an archive only once, at the location where it is specified on the command line. If the archive defines a symbol which was undefined in some object which appeared before the archive on the command line, the linker will include the appropriate files from the archive. However, an undefined symbol in an object appearing later on the command line will not cause the linker to search the archive again.

`-Lsearchdir --library-`

`path=searchdir`

Add path searchdir to the list of paths the ―ld‖ will search for archive libraries and ―ld‖ control scripts. You may use this option any number of times. The directories are searched in the order in which they are specified on the command line. Directories specified on the command line are searched before the default directories. All ‗-L‘ options apply to all ‗-l‘ options, regardless of the order in which the options appear.

The searchdir begin with =, then the = will be replaced by the sysroot prefix, a path specified when the linker is configured.

The default set of paths searched (without being specified with ‗ -L‘) depends on which emulation mode ―ld‖ is using and in some cases also on how it was configured. The paths can also be specified in a link script with the SEARCH_DIR command. Directories specified this way are searched at the point in which linker script appears in the command line.

`-memulation`

Emulate the emulation linker. You can list the available emulations with the ‗–verbose‘ or ‗-V‘ option.

If the ‗-m‘ option is not used, the emulation is taken from the DEMULATION environment variable, if that is defined. Otherwise, the default emulation depends upon how the linker was configured.

`-M`

Print (to the standard output) a link map--diagnostic information about where symbols are mapped by ld, and information on global common storage allocation.

`-Map mapfile`

Print to the file mapfile a link map--diagnostic information about where symbols are mapped by ld, and information on global common storage allocation.

`-n --`

`nmagic`

Turn off page alignment of sections, and mark the output as NMAGIC if possible.

`-N --`

`omagic`

Set the text and data sections to be readable and writable. Also, do not page align the data segment, and disable linking against shared libraries. If the output format supports UNIX style magic numbers, mark the output as OMAGIC.

`--no-omagic`

This option negates most of the effects of the ‗ -N‘ option. It sets the text section to be read-only, and forces the data segments to be page aligned.

Note: This option does not enable linking against shared libraries. Use ‗-Bdynamic‘ option for this.

`-noinhibit-exec`

Retain the executable output file whenever it is still usable. Normally, the linker will not produce an output file if it encounters errors during the link process; it exits without writing an output file when it issues any error whatsoever.

`-no-keep-memory`

ld normally optimizes for speed over memory usage by caching the symbol tables of input files in memory. This option tells ld to instead optimize for memory usage, by rereading the symbol tables as necessary. This may be required if ld runs out of memory space while linking a large executable.

-o output --

output=output

Use output as the name for the program produced by ―ld‖. If this option is not specified, the name ‗a.out‗ is used by default. The script command output can also specify the output file name.

-oformat output-format

ld may be configured to support more than one kind of object file. If your ld is configured this way, you can use the `-oformat' option to specify the binary format for the output object file. Even when ld is configured to support alternative object formats, you don't usually need to specify this, as ld should be configured to produce as a default output format the most usual format on each machine. output-format is a text string, the name of a particular format supported by the BFD libraries. (You can list the available binary formats with `objdump -i'.) The script command OUTPUT_FORMAT can also specify the output format, but this option overrides it.

--print-map

Print a link map to the standard output. A link map provides information about the link, including the following:

- Where object files and symbols are mapped into memory.
- How common symbols are allocated.
- All archive members included in the link, with a mention of the symbol which caused the archive member to be brought in.

-0 level

If level is a numeric values greater than zero ―ld‖ optimizes th e output. This might take significantly longer and therefore probably should only be enabled for the final binary.

-q --emit-

relocs

Leave relocation sections and contents in fully linked executables. Post link analysis and optimization tools may need this information in order to perform correct modifications of executables. This results in larger executables. This option is currently only supported on ELF platforms.

-r --

relocatable

Generate relocatable output: i.e. generate an output file that can in turn serve as input to ―ld‖. This is often called partial linking. As a side effect , in environment that support standard Unix magic numbers, this option also sets the output file‘s magic number to OMAGIC. If this option is not specified, an absolute file is produced. When linking C++ programs, this option will not resolve references to constructors; to do that use ‗-Ur‘.

When an input file does not have the same format as the output file, partial linking is only supported if that input file does not contain any relocations. Different output formats can have further restrictions, e.g. some a.out based formats do not support partial linking with input files in other formats at all. This option does the same thing as ‗-i‘.

-R filename --just-

symbols=filename

Read symbol names and their addresses from filename, but do not relocate it or include it in the output. This allows your output file to refer symbolically to absolute locations of memory defined in other programs. You may use this option more than once.

For compatibility with other ELF linkers, if the ‗-R‘ option is followed by a directory name, rather than a file name, it is treated as the ‗-rpath‘ option.

-relax

An option with machine dependent effects. On some platforms, the `-relax' option performs global optimizations that become possible when the linker resolves addressing in the program, such as relaxing address modes and synthesizing new instructions in the output object file.

-retain-symbols-file filename

Retain only the symbols listed in the file filename, discarding all others. filename is simply a flat file, with one symbol name per line. This option is especially useful in environments where a large global symbol table is accumulated gradually, to conserve run-time memory. ` -retain-symbols-file' does not discard undefined symbols, or symbols needed for relocations. You may only specify `-retain-symbols-file' once in the command line. It overrides `-s' and `-S'.

-s --

strip-all

Omit all symbol information from the output file.

-S --strip-

debug

Omit debugger symbol information (but not all symbols) from the output file.

-shared

Create a shared library. This is currently only supported on ELF and SunOS platforms. On SunOS, the linker will automatically create a shared library if the -e option is not used and there are undefined symbols in the link.

```
-sort-common
```

Normally, when ld places the global common symbols in the appropriate output sections, it sorts them by size. First come all the one byte symbols, then all the two bytes, then all the four bytes, and then everything else. This is to prevent gaps between symbols due to alignment constraints. This option disables that sorting.

```
-split-by-reloc count
```

Tries to create extra sections in the output file so that no single output section in the file contains more than count relocations. This is useful when generating huge relocatable for downloading into certain real time kernels with the COFF object file format; since COFF cannot represent more than 65535 relocations in a single section. Note that this will fail to work with object file formats which do not support arbitrary sections. The linker will not split up individual input sections for redistribution, so if a single input section contains more than count relocations one output section will contain those many relocations.

```
-split-by-file
```

Similar to -split-by-reloc but creates a new output section for each input file.

```
-stats
```

Compute and display statistics about the operation of the linker, such as execution time and memory usage.

```
-t --

trace
```

Prints the names of the input files as ¯ld‖ processes them.

```
-Tbss  org

-Tdata org

-Ttext org
```

Use org as the starting address for--respectively--the bss, data, or the text segment of the output file.org must be a single hexadecimal integer; for compatibility with other linkers, you may omit the leading `0x' usually associated with hexadecimal values.

```
-T commandfile

-Tcommandfile
```

Read link commands from the file commandfile. These commands replace ld's default link script (rather than adding to it), so commandfile must specify everything necessary to describe the target format. If commandfile does not exist, ld looks for it in the directories specified by any preceding `-L' options. Multiple `-T' options accumulate.

```
-T scriptfile --
script=scriptfile
```

Use scriptfile as the linker script. This script replaces ―ld‘s‖ default linker script (rather than adding to it), so command file must specify everything necessary to describe the output file. If scriptfile does not exist in the current directory, ―ld‖ looks for it in the directories specified by any preceding ‗-L‘ option. Multiple ‗-T‘ option accumulate.

```
-u symbol --
undefined=symbol
```

Forced symbol to be entered in the output file as an undefined symbol. Doing this may, for example, trigger linking of additional modules from standard libraries. ‗-u‘ may be repeated with different option arguments to enter additional undefined symbols. This option is equivalent to the EXTERN linker script command.

```
-Ur
```

For anything other than C++ programs, this option is equivalent to ‗-r‘: it generates relocatable output – i.e., an output file that can in turn serve as input to ―ld‖. When linking C++ programs, ‗-Ur‘ does not resolve references to constructors, unlike ‗-r‘. It does not work to use ‗ -Ur‘ on files that were themselves linked with ‗-Ur‘. Once the constructor table has been built, it cannot be added to. Use ‗-Ur‘ only for the last partial link, and ‗-r‘ for the others.

```
--Unique[=SECTION]
```

Creates a separate output section for every input section matching SECTION, or if the optional wildcard SECTION argument is missing, for every orphan input section. An orphan section is one not specifically mentioned in a linker script. You may use this option multiple times on the command line. It prevents the normal, merging of input sections with the same name, overriding output section assignments in a linker scripts.

```
-v --
version -
V
```

Display the version number for ―ld‖. The ‗-V‘ option also lists the supported emulations.

`-warn-common`

Warn when a common symbol is combined with another common symbol or with a symbol definition. Unix linkers allow this somewhat sloppy practice, but linkers on some other operating systems do not. This option allows you to find potential problems from combining global symbols. Unfortunately, some C libraries use this practice, so you may get some warnings about symbols in the libraries as well as in your programs. There are three kinds of global symbols, illustrated here by C examples:

`int i = 1;

A definition, which goes in the initialized data section of the output file.

`extern int i;

An undefined reference, which does not allocate space. There must be either a definition or a common symbol for the variable somewhere.

`int i;

A common symbol. If there are only (one or more) common symbols for a variable, it goes in the uninitialized data area of the output file. The linker merges multiple common symbols for the same variable into a single symbol. If they are of different sizes, it picks the largest size. The linker turns a common symbol into a declaration, if there is a definition of the same variable.

The `-warn-common' option can produce five kinds of warnings. Each warning consists of a pair of lines: the first describes the symbol just encountered, and the second describes the previous symbol encountered with the same name. One or both of the two symbols will be a common symbol.

- Turning a common symbol into a reference, because there is already a definition for the symbol. file(section): warning: common of `symbol' overridden by definition file(section): warning: defined here
- Turning a common symbol into a reference, because a later definition for the symbol is encountered. This is the same as the previous case, except that the symbols are encountered in a different order. file(section): warning: definition of `symbol'
  overriding common
  file(section): warning: common is here
- Merging a common symbol with a previous same-sized common symbol.
  file (section): warning: multiple common of `symbol'
  file (section): warning: previous common is here
- Merging a common symbol with a previous larger common symbol.
  file(section): warning: common of `symbol'
  overridden by larger common
  file(section): warning: larger common is here

- Merging a common symbol with a previous smaller common symbol. This is the same as the previous case, except that the symbols are encountered in a different order.
  file(section): warning: common of `symbol' overriding smaller common
  file(section): warning: smaller common is here

`-warn-constructors`

Warn if any global constructors are used. This is only useful for a few object file formats. For formats like COFF or ELF, the linker can not detect the use of global constructors.

`-warn-once`

Only warn once for each undefined symbol, rather than once per module which refers to it. For each archive mentioned on the command line, include every object file in the archive in the link, rather than searching the archive for the required object files. This is normally used to turn an archive file into a shared library, forcing every object to be included in the resulting shared library.

`-x --discard-`

`all`

Discard all local symbols.

`-X --discard-`

`locals`

Delete all temporary local symbols. For most targets, this is all local symbols whose names begin with ‗L‘.

`-y symbol --trace-`

`symbol=symbol`

Print the name of each linked file in which symbol appear. This option may be given any number of times. On many systems it is necessary to prepend an underscore.

This option is useful when you have an undefined symbol in your link but don‘t know where the reference is coming from.

`-Y path`

Add path to the default library search path. This option exists for Solaris compatibility.

`-z keyword`

The recognized keywords are initfirst, interpose, loadfltr, nodefaultlib, nodelete, nodlopen, nodump, now, origin, combreloc, nocombreloc and nocopyreloc. The other keywords are ignored for Solaris compatibility.

initfirst marks the object to be initialized first at runtime before any other objects.

interpose marks the object that its symbol table interposes before all symbols but the primary executable.

loadfltr marks the object that its filtees be processes immediately at runtime.

nodefaultlib marks the object that the search for dependencies of this object will ignore any default library search paths.

nodelete marks the object shouldn't be unloaded at run time.

nodlopen marks the object not available to dlopen.

nodump marks the object cannot be dumped by

dldump. now marks the object with the non-lazy runtime

binding. origin marks the object may contain $ORIGIN.

defs disallows undefined symbols.

muldefs allows multiple definitions.

combreloc combines multiple reloc section and sort them to make dynamic symbol lookup caching possible.

nocombreloc disables multiple reloc sections combining.

nocopyreloc disables production of copy relocs.

2.4.4.3 Linker Scripts

Every link is controlled by a linker script. This script is written in the linker command. The main purpose of the linker script is to describe how the sections in the input file should be mapped into the output file, and to control the memory layout of the output file. The linker always uses the linker scripts. If you don't supply one yourself, the linker will use a default script that is compiled into the linker executable.

Basic Linker Script Concepts

The linker combines input files into single output file. The output file and each input file are in the special data format known as an object file format. Each file is called an object file. The output file is often called an executable or object file. Each object file has, among other things, a list of sections.

Each section in an object file has a name and a size. Most sections also have an associated block of data, know as the section contents. A section may be marked as loadable, which mean that the contents should be loaded into memory when the output file is run. A section with no contents may be allocatable, which means that an area in memory should be set aside, but nothing in particular should be loaded there. A section which is neither loadable nor allocatable typically contains some sort of debugging information.

Every loadable or allocatable output section has two addresses. The first is the VMA (virtual memory address). This is the address the section will have when the output file is run. The second is the LMA (load memory address). This is the address at which the section will be loaded. In most cases the two addresses will be same. An example of when they might be different is when a data section is loaded onto ROM, and then copied into RAM when the program starts up (this technique is often used to initialize global variables in a ROM based system). In this case ROM address would be the LMA, and the RAM address would be the LMA.

You can see the sections in an object file by using the objdump program with the ‗-h‘ option. Every object file also has a list of symbols, known as the symbol table. A symbol may be defined or undefined. Each symbol has a name, and each defined symbol has an address, among other information. If you compile C or C++ program into an object file, you will get a defined symbol for every defined function and global or static variable. Every undefined function or global variable which is referenced in the input file will become an undefined symbol.

In short linker script does the following:

- Sets up the memory map for the application

  When your application loads into memory, it allocates some RAM, some disk space for I/O, and some registers. The linker script makes a memory map of this memory allocation. This is important to embedded systems because you gain the ability to manage the behavior of the chip, even though they have no OS.

- Sets up the constructor and destructor tables for GNU C++ compiling

  Actual section names vary depending on your object file format. For a.out and COFF formats, .text , .data , and .bss are the three main sections.

- Sets the default values for variables in use by sbrk() and the crt0 file, which are typically called by _bss_start and _end . You can use the ‗–verbose‘ command line option to display the default linker script. Certain command line options, such as ‗-r‘ or ‗-N‘ will affect the default linker script

- You may supply your own linker script by using the ‗-T‘ command line option. When you do this, your linker script will replace the default linker script.

- You may also use linker scripts implicitly by naming them as input files to the linker, as though they were files to be linked.

Linker Script Format

Linker scripts are text files. You write linker script as series of commands. Each command is either a keyword, possibly followed by arguments, or an assignment to a symbol. You may separate commands using semicolons. Whitespace is generally ignored.

String such as file or format names are normally be entered directly. If the file name contains a character such as a comma which would otherwise serve to separate file names, you may put the file name in double quotes. There is no way to use a double quote character in a file name.

You may include comments in linker scripts just as in C delimited by ‗/*' and ‗*/'.

Simple Linker Script Commands

- Setting the Entry Point

  The first instruction to execute in a program is called the entry point. You can use the ENTRY linker script command to set the entry point. The argument is a symbol name:

  ```
  ENTRY(symbol)
  ```

  There are several ways to set the entry point. The linker will set the entry point by trying each of the following methods in order, and stopping when one of them succeeds:

  - The ‗-e' entry command-line option
  - The ENTRY(symbol) command in a linker script
  - The value of the symbol start, if defined
  - The address of the first byte of the ‗.text' section, if present
  - The address 0

- Commands Dealing with Files

  Several linker script commands deal with files.

  ```
  INCLUDE filename
  ```

  Include the linker script filename at this point. The file will be searched for in the current directory, and in any directory specified with the ‗-L' option. You can nest calls to INCLUDE up to 10 levels deep.

  ```
  INPUT (file, file, …)
  INPUT (file file …)
  ```

  The input command directs the linker to include the named files in the link, as though they were named on the command line.

For example, if you always want to include ‗test.o' any time you do a link, but you can't be bothered to put it on every link command line, then you can put ‗INPUT(test.o)' in your linker script.

In fact, if you like, you can list all of your input files in the linker scripts, and then invoke the linker with nothing but a ‗-T' option.

In case a sysroot prefix is configured, and the filename starts with the ‗/' character, and the script being processed was located inside the sysroot prefix, the filename will be looked for in the sysroot prefix. Otherwise the linker will try to open the filename in the current directory.If it is not found, the linker will search through the archive library search path. If y6ou use ‗INPUT(-lfile)', ―ld‖ will transform the name to libfile.a, as with the command line argument ‗-l'.

When you use the input command in an implicit linker script, the files will be included in the link at the point at which the linker script file is included. This can affect archive searching.

```
GROUP(file,file, …)
```

```
GROUP(file file …)
```

The GROUP command is like input, except that the named files should all be archive, and they are searched repeatedly until no new undefined references are created.

```
OUTPUT(filename)
```

The OUTPUT command names the output file. Using OUTPUT(filename) in the linker script is exactly like using ‗-o filename' on the command line. If both are used, the command line option takes precedence.

You can use the output command to define a default name for the output file other than the usual default of ‗a.out'.

```
SEARCH_DIR(path)
```

The SEARCH_DIR command adds path to the list of paths where ―ld‖ looks for archive libraries. Using SEARCH_DIR(path) is exactly like using ‗-L path' on the command line. If both are used, then the linker will search both paths. Paths specified using the command line options are searched first.

```
STARTUP(filename)
```

The STARTUP command is just like the INPUT command, except that filename will become the first input file to be linked, as though it were specified first on the command line. This may be useful when using a system in which the entry point is always the start of the first file.

- Commands Dealing with Object File Formats

Following linker script commands deal with object file formats.

```
OUTPUT_FORMAT(bfdname)
```

```
OUTPUT_FORMAT(default, big, little)
```

The OUTPUT_FORMAT command names the BFD format to use for the output file. Using OUTPUT_FORMAT(bfdname) is exactly ike using ‗–oformat bfdname' on the command line. If both are used, the command line option takes precedence.

You can use OUTPUT_FORMAT with three arguments to use different formats based on the ‗-EB' and ‗-EL' command line options. This permits the linker script to set the output format based on the desired endianness.

If neither ‗-EB' nor ‗-EL' are used, then the output format will be the first argument, default. If ‗-EB' is used, the output format will be second argument, big. If ‗-EL' is used, the output format will be the third argument, little.

For example, the default linker script for the MIPS ELF target uses this command:

```
OUTPUT_FORMAT (elf2-bigmips, elf32-bigmips, elf32-
littlemips)
```

This says that the default format for the output file is ‗elf32-bigmips', but if the user uses the ‗-EL' command line option, the output file will be created in the ‗elf32-littlemips' format.

```
TARGET (bfdname)
```

The target command names the BFD format to use when reading input files. It affects subsequent INPUT and GROUP commands. If the TARGET command is used but OUTPUT_FORMAT is not, then the last TARGET command is also used to set the format for the output file.

- Other Linker Script Command

Following are the few other linker script commands:

```
ASSERT (exp, message)
```

Ensure that exp is non zero. If it is zero the exit the linker with an error code and print message.

```
EXTERN (symbol, symbol …)
```

Force symbol to be entered in the output file as an undefined symbol. Doing this may, for example trigger linking of additional modules from standard libraries. You may list several symbols for each EXTERN, and you may use EXTERN multiple times. This command has the same effect as the ‗-u' command line option.

```
FORCE_COMMON_ALLOCATION
```

This command has the same effect as the ‗-d‘ command-line option: to make ld assign space to common symbols even if a relocatable output file is specified (‗-r‘).

`INHIBIT_COMMON_ALLOCATION`

This command has the same effect as the ‗–no-define-common‘ command line option: to make ld omit the assignment of addresses to common symbols even for a non-relocatable output file.

`NOCROSSREFS(section section …)`

This command may be used to tell ld to issue an error about any references among certain output sections.

In certain types of programs, particularly on embedded systems when using overlays, when one section is loaded into memory, another section will not be. Any direct references between the two sections would be the errors. For example, it would be an error if code in one section called a function defined in the other section. The NOCROSSREFS command takes a list of output section names. If ld detects any cross references between the sections, it reports an error and returns non-zero exit status. Note that the NOCROSSREFS command uses output section names, not input section names.

`OUTPUT_ARCH (bfdarch)`

Specify particular output machine architecture. The argument is one of the names used by the BFD library. You can see the architecture of an object file by using the objdump program with the ‗-f‘ option.

- Assigning Values to Symbols

You may assign a value to a symbol in linker script. This will define the symbol as a global symbol.

`Simple Assignments`

You may assign to a symbol using any of the C assignment

operators: symbol = expression ;

symbol += expression ;

symbol -= expression ;

symbol *= expression ;

symbol /= expression ;

symbol <<= expression ;

symbol >>= expression ;

symbol &= expression ;

symbol |= expression ;

The first case will define symbol to the value of expression. In the other cases, symbol must already be defined, and the value will be adjusted accordingly.

The special symbol name ‗.‘ indicates the location counter. You may only use this within a SECTION command.

The semicolon after expression is required. Below is an example showing the three different places that symbol assignments may be used:

```
Floating_point=0;
SECTIONS
{
     .text :
      {
           *(.text)
            _etext = .;
     }
     _bdata = ( . + 3)  & ~3;
     .data : { * (.data)}
}
```

In above example, symbol ‗floating-point ‗ will be defined as zero. The symbol ‗_etext‘ will be defined as the address following the last ‗.text‘ input section. The symbol ‗_bdata‘ will be defined as the address following the ‗.text‘ output section aligned upward to a 4 byte boundary.

`PROVIDE`

In some cases, it is desirable for a linker script to define a symbol only if it is referenced and it is not defined by any object included in the link. For example, traditional linkers defined the symbol ‗etext‘. However ANSI C requires that the user be able to use ‗etext‘ as a function name without encountering an error. The PROVIDE keyword may be used to define a symbol,

- SECTIONS Command

The SECTION command tells the linker how to map input sections into the output sections, and how to put the output sections in memory. The format of the SECTIONS command is:

```
SECTIONS
{
Sections-command
Sections-command
…
}
```

Each sections-command may of be one of the following:

- An ENTRY command
- A symbol assignment
- An output section description
- An overlay description

The ENTRY command and symbol assignments are permitted inside the SECTIONS command for convenience in using the location counter in those commands. This can also make the linker script easier to understand because you can use those commands at meaningful points in the layout of the output file.

Output section descriptions and overlay descriptions are described below:

If you do not use the SECTIONS command in your linker script, the linker will place each input section into the identically name output section in the order that the sections are first encountered in the input files. If all input sections are present in the first file, for example, the order of sections in the output file will match the order in the first input file. The first section will be at address zero.

Input Section Description

The input section description is the most basic linker script operation. You use output sections to tell the linker how to lay out your program in memory. You use input section descriptions to tell the linker how to map the input files into your memory layout.

Input Section Basics

The most common input section description is to include all input sections with a particular name in the output section. For example, to include all input ‗.text' sections, you would write:

*(.text)

Here the ‗*' is a wildcard which matches any file name. To exclude a list of files from matching the file name wildcard, EXCLUDE FILE may be used to match all files except the ones specified in the EXCLUDE FILE list.

For example:

(*(EXCLUDE_FILE (*crtend.o *otherfile.o) .ctors))

This will cause all .ctors sections from all files except ‗crtend.o' and ‗otherfile.o' to be included. There are two ways to include more than one section:

*(.text .rdata)

*(.text) *(.rdata)

The difference between these is the order in which the _.text‘ and _.rdata‘ input sections will appear in the output section. In the first example, they will be intermingled, appearing in the same order as they are found in the linker input. In the second example, all _.text‘ input sections will appear first, followed by all _.rdata‘ input sections. You can specify a file name to include sections from a particular file. You would do this if one or more of your files contain special data that needs to be at a particular location in memory.

For example:

data.o(.data)

If you use a file name without a list of sections, then all sections in the input file will be included in the output section. This is not commonly done, but it may by useful on occasion.

For example: data.o

When you use a file name which does not contain any wild card characters, the linker will first see if you also specified the file name on the linker command line or in an INPUT command. If you did not, the linker will attempt to open the file as an input file, as though it appeared on the command line. Note that this differs from an INPUT command, because the linker will not search for the file in the archive search path.

Input Section Wildcard pattern

In an input section description, either the file name or the section name or both may be wildcard patterns.

The file name of _*‘ seen in many examples is a simple wildcard pattern for the file name. The wildcard patterns are like those used by the Unix shell.

_*‘ matches any number of characters

_?‘ matches any single character

_[chars]‘ matches a single instance of any of the chars; the _-‘ .character may be used to specify a range of characters, as in _[a-z]‘ to match any lower case letter

_\‘ quotes the following character

File name wildcard patterns only match files which are explicitly specified on the command line or in an INPUT command. The linker does not search directories to expand wildcards. If a file name matches more than one wildcard pattern, or if a file name appears explicitly and is also matched by a wildcard pattern, the linker will use the first match in the linker script. For example, this sequence of input section descriptions is probably in error, because the _data.o‘ rule will not be used:

.data : { *(.data) }

.data1 : { data.o(.data) }

Normally, the linker will place files and sections matched by wildcards in the order in which they are seen during the link. You can change this by using the SORT keyword, which appears before a wildcard pattern in parentheses (e.g., SORT(.text*)).

When the SORT keyword is used, the linker will sort the files or sections into ascending order by name before placing them in the output file. If you ever get confused about where input sections are going, use the ‗-M‘ linker option to generate a map file. The map file shows precisely how input sections are mapped to output sections. This example shows how wildcard patterns might be used to partition files. This linker script directs the linker to place all ‗.text‘ sections in ‗.text‘ and all ‗.bss‘ sections in ‗.bss‘. The linker will place the ‗.data‘ section from all files beginning with an upper case character in ‗.DATA‘; for all other files, the linker will place the ‗.data‘ section in ‗.data‘.

```
SECTIONS {
.text : { *(.text) }
.DATA : { [A-Z]*(.data) }
.data : { *(.data) }
.bss : { *(.bss) }
}
```

Input Section For Common Symbols

A special notation is needed for common symbols, because in many object file formats common symbols do not have a particular input section. The linker treats common symbols as though they are in an input section named ‗COMMON‘. You may use file names with the ‗COMMON‘ section just as with any other input sections. You can use this to place common symbols from a particular input file in one section while common symbols from other input files are placed in another section. In most cases, common symbols in input files will be placed in the ‗.bss‘ section in the output file.

For example:

```
.bss { *(.bss) *(COMMON) }
```

Some object file formats have more than one type of common symbol. For example, the MIPS ELF object file format distinguishes standard common symbols and small common symbols. In this case, the linker will use a different special section name for other types of common symbols. In the case of MIPS ELF, the linker uses ‗COMMON‘ for standard common symbols and ‗.scommon‘ for small common symbols. This permits you to map the different types of common symbols into memory at different locations. You will sometimes see

‗[COMMON]‘ in old linker scripts. This notation is now considered obsolete. It is equivalent to ‗*(COMMON)‘.

Input Section And Garbage Collection

When link -time garbage collection is in use (‗--gc-sections‘), it is often useful to mark sections that should not be eliminated. This is accomplished by surrounding an input section‘s wildcard entry with KEEP(), as in KEEP(*(.init)) or KEEP(SORT(*)(.ctors)).

Input Section Example

The following example is a complete linker script. It tells the linker to read all of the sections from file ‗all.o‘ and place them at the start of output section ‗outputa‘ which starts at location ‗0x10000‘. All of section ‗.input1‘ from file ‗foo.o‘ follows immediately, in the same output section. All of section ‗.input2‘ from ‗foo.o‘ goes into output section ‗outputb‘, followed by section ‗.input1‘ from ‗foo1.o‘. All of the remaining ‗.input1‘ and ‗.input2‘ sections from any files are written to output section ‗outputc‘.

```
SECTIONS {
outputa 0x10000 :
{
all.o
foo.o (.input1)
}
outputb :
{
foo.o (.input2)
foo1.o (.input1)
}
outputc :
{
*(.input1)
*(.input2)
}
}
```

Output Section Description

The address is an expression for the VMA (virtual memory address) of the output section. If you do not provide address, the linker will set it based on region if present, or otherwise based on the current value of the location counter. If you provide address, the address of the output section will be set to precisely that. If you provide neither address nor region, then the address of the output section will be set to the current value of the location counter aligned to the alignment requirements of the output section. The alignment requirement of the output section is the strictest alignment of any input section contained within the output section.

For example,

```
.text . : { *(.text)
} And
.text : { *(.text) }
```

are subtly different. The first will set the address of the _.text‘ output section to the current value of the location counter. The second will set it to the current value of the location counter aligned to the strictest alignment of a _.text‘ input.

The address may be an arbitrary expression;

For example, if you want to align the section on a 0x10 byte boundary, so that the lowest four bits of the section address are zero, you could do something like this:

```
.text ALIGN(0x10) : { *(.text) }
```

This works because ALIGN returns the current location counter aligned upward to the specified value. Specifying address for a section will change the value of the location counter.

The full description of an output section looks like this:

```
section [address] [(type)] : [AT(lma)]
{
output-section-command
output-section-command
…
}   [>region]   [AT>lma_region]   [:phdr   :phdr   ...]
[=fillexp]
```

Most output sections do not use most of the optional section attributes. The whitespace around section is required, so that the section name is unambiguous. The colon and the curly braces are also required. The line breaks and other white space are optional.

Each output-section-command may be one of the following:

- A symbol assignment
- An input section description
- Data values to include directly
- A special output section keyword

Output Section name

The name of the output section is section. Section must meet constraint of your output format. In formats which only support a limited number of sections, such as a.out, the name must be one of the name supported by the format (a.out, for example, allows only ‗.text‘, ‗.data‘, or ‗.bss‘)

If the output format supports any number of sections, but with numbers, not names, the name should be supplied as a quoted numeric string. The section name may consist of any sequence of characters, but the name which contains any unusual characters such as commas must be quoted.

Output Section data

You can include explicit bytes of data in an output section by using BYTE, SHORT, LONG, QUAD and SQUAD as an output section command. Each keyword is followed by an expression in parentheses providing the value to store. The value of the expression is stored at the current value of the location counter. The BYTE, SHORT, LONG, QUAD commands one, two, four and eight bytes respectively. After storing the bytes the location counter is incremented by the number of bytes stored. For example, this will store the byte 1 followed by the four byte value of the symbol ‗addr‘:

```
BYTE(1)
LONG(addr)
```

When using a 64 bit host or target, QUAD and SQUAD are the same; they both store an 8 byte, or 64 bit, value. When both host and target are 32 bits, an expression is computed as 32 bits. In this case QUAD stores a 32 bit value zero extended to 64 bits, and SQUAD stores a 32 bit value sign extended to 64 bits. If the object file format of the output file has an explicit endianness, which is the normal case, the value will be stored in that endianness. When the object file format does not have an explicit endianness, as is true of, for example, S-records, the value will be stored in the endianness of the first input object file.

Note:: These commands only work inside a section description and not between them, so the following will produce an error from the linker:

```
SECTIONS {.text : { *(.text) } LONG(1) .data : {
*(.data) } }
```

whereas this will work:

```
SECTIONS {.text : {*(.text) ; LONG(1) } .data : {
*(.data) } }
```

You may use the FILL command to set the fill pattern for the current section. It is followed by an expression in parentheses. Any otherwise unspecified regions of memory within the section (for example, gaps left due to the required alignment of input sections) are filled with the value of the expression, repeated as necessary. A FILL statement covers memory locations after the point at which it occurs in the section definition; by including more than one FILL statement, you can have different fill patterns in different parts of an output section. This example shows how to fill unspecified regions of memory with the value ‗0x90':

FILL(0x90909090)

The FILL command is similar to the ‗=fillexp' output section attribute, but it only affects the part of the section following the FILL command, rather than the entire section. If both are used, the FILL command takes precedence.

Output Section Keywords

There are a couple of keywords which can appear as output section commands.

- CREATE_OBJECT_SYMBOLS
  The command tells the linker to create a symbol for each input file. The name of each symbol will be the name of the corresponding input file. The section of each symbol will be the output section in which the CREATE_OBJECT_SYMBOLS command appears.
  This is conventional for the a.out object file format. It is not normally used for any other object file format.
- CONSTRUCTORS

  When linking using the a.out object file format, the linker uses an unusual set construct to support C++ global constructors and destructors. When linking object file formats which do not support arbitrary sections, such as ECOFF and XCOFF, the linker will automatically recognize C++ global constructors and destructors by name. For these objects file formats, the CONSTRUCTORS command tells the linker to place constructor information in the output section where the CONSTRUCTORS command appears. The CONSTRUCTORS command is ignored for other object file formats. The symbol __CTOR_LIST__ marks the start of the global constructors, and the symbol __DTOR_LIST marks the end. The first word in the list is the number of entries, followed by the address of each constructor or destructor, followed by a zero word. The compiler must arrange to actually run the code. For these object file formats gnu C++ normally calls constructors from a subroutine __main; a call to __main is automatically inserted into the startup code for main. gnu C++ normally runs destructors either by using atexit, or directly from the function exit. For object file formats such as COFF or ELF which support arbitrary section names, gnu C++ will normally arrange to put the addresses of global constructors and destructors into the .ctors and .dtors sections. Placing the following sequence into your linker script will build the sort of table which the gnu C++ runtime code expects to see.

```
__CTOR_LIST__ = .;
LONG((__CTOR_END__ - __CTOR_LIST__) / 4
- 2)
*(.ctors)
LONG(0)
__CTOR_END__ = .;
__DTOR_LIST__ = .;
LONG((__DTOR_END__ - __DTOR_LIST__) / 4
- 2)
*(.dtors)
LONG(0)
__DTOR_END__ = .;
```

If you are using the gnu C++ support for initialization priority, which provides some control over the order in which global constructors are run, you must sort the constructors at link time to ensure that they are executed in the correct order. When using the CONSTRUCTORS command, use _SORT(CONSTRUCTORS)‘ instead. When using the .ctors and .dtors sections, use _*(SORT(.ctors))‘ and _*(SORT(.dtors))‘ instead of just _*(.ctors)‘ and _*(.dtors)‘. Normally the compiler and linker will handle these issues automatically, and you will not need to concern yourself with them. However, you may need to consider this if you are using C++ and writing your own linker scripts.

Output Section Discarding

The linker will not create output section which does not have any contents. This is for convenience when referring to input sections that may or may not be present in any of the input files.

For example:

```
.foo { *(.foo) }
```

will only create a _.foo‘ section in the output file if there is a _.foo‘ section in at least one input file. If you use anything other than an input section description as an output section command, such as a symbol assignment, then the output section will always be created, even if there are no matching input sections. The special output section name _/DISCARD/‘ may be used to discard input sections. Any input sections which are assigned to an output section named _/DISCARD/‘ are not included in the output file.

Output Section Type

Each output section may have a type. The type is a keyword in parentheses. The following types are defined:

```
NOLOAD
```

The section should be marked as not loadable, so that it will not be loaded into memory when the program is run.

```
DSECT
COPY
INFO
OVERLAY
```

These type names are supported for backward compatibility, and are rarely used. They all have the same effect: the section should be marked as not allocatable, so that no memory is allocated for the section when the program is run.

The linker normally sets the attributes of an output section based on the input sections which map into it. You can override this by using the section type. For example, in the script sample below, the _ROM‘ section is addressed at memory location _0‘ and does not need to be loaded when the program is run. The contents of the _ROM‘ section will appear in the linker output file as usual.

```
SECTIONS {
        ROM 0 (NOLOAD) : { ... }
        ...
        }
```

Output Section LMA

The linker will normally set the LMA equal to the VMA. You can change that by using the AT keyword. The expression lma that follows the AT keyword specifies the load address of the section. Alternatively, with ‗AT>lma_region‘ expression, you may specify a memory region for the section‘s load address. This feature is designed to make it easy to build a ROM image. For example, the following linker script creates three output sections: one called ‗.text‘, which starts at 0x1000, one called ‗.mdata‘, which is loaded at the end of the ‗.text‘ section even though its VMA is 0x2000, and one called ‗.bss‘ to hold uninitialized data at address 0x3000. The symbol _data is defined with the value 0x2000, which shows that the location counter holds the VMA value, not the LMA value.

```
        SECTIONS
        {
        .text 0x1000 : { *(.text) _etext = . ; }
        .mdata 0x2000 :
        AT ( ADDR (.text) + SIZEOF (.text) )
        { _data = . ; *(.data); _edata = . ; }
        .bss 0x3000 :
        { _bstart = . ; *(.bss) *(COMMON) ; _bend =
        . ;}
        }
```

The run-time initialization code for use with a program generated with this linker script would include something like the following, to copy the initialized data from the ROM image to its runtime address. Notice how this code takes advantage of the symbols defined by the linker script.

```
        extern char _etext, _data, _edata, _bstart,
_bend;
        char *src = &_etext;
        char *dst = &_data;
        /* ROM has data at end of text; copy it. */
        while (dst < &_edata) {
        *dst++ = *src++;
        }
        /* Zero bss */
        for (dst = &_bstart; dst< &_bend;
        dst++) *dst = 0;
```

Output Section Region

You can assign a section to a previously defined region of memory by using ‗>region‘.

Here is a simple example:

```
MEMORY {rom : ORIGIN = 0x1000, LENGTH = 0x1000 }
  SECTIONS { ROM : { *(.text) } >rom }
```

Output Section Phdr

You can assign a section to a previously defined program segment by using ‗:phdr‘. If a section is assigned to one or more segments, then all subsequent allocated sections will be assigned to those segments as well, unless they use an explicitly :phdr modifier. You can use :NONE to tell the linker to not put the section in any segment at all.

Here is a simple example:

```
PHDRS { text PT_LOAD ; }
SECTIONS { .text : { *(.text) } :text }
```

- PHDRS Command

The ELF object file format uses program headers, also knows as segments. The program headers describe how the program should be loaded into memory. You can print them out by using the objdump program with the ‗-p‘ option.

When you run an ELF program on a native ELF system, the system loader reads the program headers in order to figure out how to load the program. This will only work if the program headers are set correctly. The linker will create reasonable program headers by default. However, in some cases, you may need to specify the program headers more precisely. You may use the PHDRS command for this purpose. When the linker sees the PHDRS command in the linker script, it will not create any program headers other than the ones specified. The linker only pays attention to the PHDRS command when generating an ELF output file. In other cases, the linker will simply ignore PHDRS. This is the syntax of the PHDRS command. The words PHDRS, ILEHDR, AT, and FLAGS are keywords.

```
PHDRS
    {
    name type [ FILEHDR ] [ PHDRS ] [ AT (
    address ) ]
    [ FLAGS ( flags ) ] ;
    }
```

The name is used only for reference in the SECTIONS command of the linker script. It is not put into the output file. Program header names are stored in a separate name space, and will not conflict with symbol names, file names, or section names. Each program header must have a distinct name. Certain program header types describe segments of memory which the system loader will load from the file. In the linker script, you specify the contents of these segments by placing allocatable output sections in the segments. You use the ‗:phdr‘ output section attribute to place a section in a particular segment. It is normal to put certain sections in more than one segment. This merely implies that one segment of memory contains another. You may repeat ‗:phdr‘, using it once for each segment which should contain the section. If you place a section in one or more segments using ‗:phdr‘, then the linker will place all subsequent allocatable sections which do not specify ‗:phdr‘ in the same segments. This is for convenience, since generally a whole set of contiguous sections will be placed in a single segment. You can use :NONE to override the default segment and tell the linker to not put the section in any segment at all. You may use the FILEHDR and PHDRS keywords appear after the program header type to further describe the contents of the segment. The FILEHDR keyword means that the segment should include the ELF file header. The PHDRS keyword means that the segment should include the ELF program headers themselves. The type may be one of the following. The numbers indicate the value of the keyword.

- PT_NULL (0)
  Indicates an unused program header.
- PT_LOAD (1)
  Indicates that this program header describes a segment to be loaded from the file.
- PT_DYNAMIC (2)
  Indicates a segment where dynamic linking information can be found.
- PT_INTERP (3)
  Indicates a segment where the name of the program interpreter may be found.
- PT_NOTE (4)
  Indicates a segment holding note information.
- PT_SHLIB (5)
  A reserved program header type, defined but not specified by the ELF ABI.
- PT_PHDR (6)
  Indicates a segment where the program headers may be found. Expression. An expression giving the numeric type of the program

header. This may be used for types not defined above. Here is an example of PHDRS. This shows a typical set of program headers used on a native ELF system.

```
PHDRS
{
headers PT_PHDR PHDRS ;
interp PT_INTERP ;
text PT_LOAD FILEHDR PHDRS ;
data PT_LOAD ;
dynamic PT_DYNAMIC ;
}
SECTIONS
{
. = SIZEOF_HEADERS;
.interp : { *(.interp) } :text :interp
.text : { *(.text) } :text
.rodata : { *(.rodata) } /* defaults to
:text */
...
. = . + 0x1000; /* move to a new page
in memory */
.data : { *(.data) } :data
.dynamic : { *(.dynamic) } :data :dynamic
...
}
```

- VERSION Command

The linker supports symbol versions when using ELF. Symbol versions are only useful when using shared libraries. The dynamic linker can use symbol versions to select a specific version of a function when it runs a program that may have been linked against an earlier version of the shared library. You can include a version script directly in the main linker script, or you can supply the version script as an implicit linker script. You can also use the ‗--version-script‘ linker option.

The syntax of the VERSION command is simply

VERSION {version-script-commands}

- Expressions in Linker Scripts

The syntax for expressions in the linker script language is identical to that of C expressions. All expressions are evaluated as integers. All expressions are evaluated in the same size, which is 32 bits if both the host and target are 32 bits, and is otherwise 64 bits. You can use and set symbol values in expressions. The linker defines several special purpose built-in functions for use in expressions.

Constants

As in C, the linker considers an integer beginning with ‗0‘ to be octal, and an integer beginning with ‗0x‘ or ‗0X‘ to be hexadecimal. The linker considers other integers to be decimal. In addition, you can use the suffixes K and M to scale a constant by 1024 or 10242 respectively.

For example, the following all refer to the same

quantity: _fourk_1 = 4K;

_fourk_2 = 4096;

_fourk_3 = 0x1000;

Symbol Names

Unless quoted, symbol names start with a letter, underscore, or period and may include letters, digits, underscores, periods, and hyphens. Unquoted symbol names must not conflict with any keywords. You can specify a symbol which contains odd characters or has the same name as a keyword by surrounding the symbol name in double quotes:

"SECTION" = 9;

"with a space" = "also with a space" + 10;

Since symbols can contain many non-alphabetic characters, it is safest to delimit symbols with spaces. For example, ‗A-B‘ is one symbol, whereas ‗A - B‘ is an expression involving subtraction.

The Location Counter

The special linker variable dot ‗.‘ always contains the current output location counter. Since the ‗.‘ always refers to a location in an output section, it may only appear in an expression within a SECTIONS command. The ‗. ‘ symbol may appear anywhere that an ordinary symbol is allowed in an expression. Assigning a value to ‗.‘ will cause the location counter to be moved. This may be used to create holes in the output section. The location counter may never be moved backwards.

```
SECTIONS
{
output :
{
file1(.text)
. = . + 1000;
file2(.text)
. += 1000;
file3(.text)
} = 0x12345678;
}
```

In the previous example, the ‗.text‘ section from ‗file1‘ is located at the beginning of the output section ‗output‘. It is followed by a 1000 byte gap. Then the ‗.text‘ section from ‗file2‘ appears, also with a 1000 byte gap following before the ‗.text‘ section from ‗file3‘. The notation ‗= 0x12345678‘ specifies what data to write in the gaps

Note: Actually refers to the byte offset from the start of the current containing object. Normally this is the SECTIONS statement, whose start address is 0, hence, can be used as an absolute address. If . is used inside a section description however, it refers to the byte offset from the start of that section, not an absolute address. Thus in a script like this:

```
SECTIONS
{
. = 0x100
.text: {
*(.text)
. = 0x200
}
. = 0x500
.data: {
*(.data)
. += 0x600
}
}
```

The ‗.text‘ section will be assigned a starting address of 0x100 and a size of exactly 0x200 bytes, even if there is not enough data in the ‗.text‘ input sections to fill this area. (If there is too much data, an error will be produced because this would be an attempt to move . backwards). The ‗.data‘ section will start at 0x500 and it will have an extra 0x600 bytes worth of space after the end of the values from the ‗.data‘ input sections and before the end of the ‗.data‘ output section itself.

Evaluation

Linker only computes the value of an expression when absolutely necessary. The linker needs some information, such as the value of the start address of the first section, and the origins and lengths of memory regions, in order to do any linking at all. These values are computed as soon as possible when the linker reads in the linker script. However, other values (such as symbol values) are not known or needed until after storage allocation. Such values are evaluated later, when other information (such as the sizes of output sections) is available for use in the symbol assignment expression. The sizes of sections cannot be known until after allocation, so assignments dependent upon these are not performed until after allocation. Some expressions, such as those depending upon the location counter ‗.‘, must be evaluated during section allocation. If the result of an expression is required, but the value is not available, then it results in an error. For example, a script like the following will cause the error message ‗non constant expression for initial address‘.

```
SECTIONS
{
.text 9+this_isnt_constant :
{ *(.text) }
}
```

The Section of an Expression

When the linker evaluates an expression, the result is either absolute or relative to some section. A relative expression is expressed as a fixed offset from the base of a section. The position of the expression within the linker script determines whether it is absolute or relative. An expression which appears within an output section definition is relative to the base of the output section. An expression which appears elsewhere will be absolute. A symbol set to a relative expression will be relocatable if you request relocatable output using the ‗-r‘ option. That means that a further link operation may change the value of the symbol. The symbol‘s section will be the section of the relative expression. A symbol set to an absolute expression will retain the same value through any further link operation. The symbol will be absolute, and will not have any particular associated section. You can use the built-in function ABSOLUTE to force an expression to be absolute when it would otherwise be relative. For example, to create an absolute symbol set to the address of the end of the output section ‗.data‘:

```
SECTIONS
{
.data : { *(.data) _edata = ABSOLUTE(.); }
}
```

If ‗ABSOLUTE‘ were not used, ‗_edata‘ would be relative to the ‗.data‘ section.

- Built-in Functions

The linker script language includes a number of built-in functions for use in linker script expressions.

ABSOLUTE(exp)

Return the absolute (non-relocatable, as opposed to non-negative) value of the expression *expression*. It is primarily useful to assign an absolute value to a symbol within a section definition, where symbol values are normally section relative.

ADDR(section)

Return the absolute address (the VMA) of the named section. Your script must previously have defined the location of that section. In the following example, symbol_1 and symbol_2 are assigned identical values:

```
SECTIONS {...
.output1 :
{
start_of_output_1 = ABSOLUTE(.);
...
}
.output :
{
symbol_1 = ADDR(.output1);
symbol_2 = start_of_output_1;
}
... }
```

ALIGN(exp

)

Return the location counter (.) aligned to the next exp boundary. ALIGN doesn‘t change the value of the location counter—it just does arithmetic on it. Here is an example which aligns the output .data section to the next 0x2000 byte boundary after the preceding section and sets a variable within the section to the next 0x8000 boundary after the input sections:

```
SECTIONS { ...
.data ALIGN(0x2000):
{ *(.data)
variable = ALIGN(0x8000);
}
... }
```

The first use of ALIGN in this example specifies the location of a section because it is used as the optional address attribute of a section definition .The second use of ALIGN is used to defines the value of a symbol. The built-in function NEXT is closely related to ALIGN.

BLOCK(exp)

This is a synonym for ALIGN, for compatibility with older linker scripts. It is most often seen when setting the address of an output section.

DATA_SEGMENT_ALIGN(maxpagesize, commonpagesize)

This is equivalent to either

```
(ALIGN(maxpagesize) + (. & (maxpagesize -
1))) Or
(ALIGN(maxpagesize) + (. & (maxpagesize - commonpagesize)))
```

depending on whether the latter uses fewer commonpagesize sized pages for the data segment (area between the result of this expression and DATA_SEGMENT_END) than the former or not. If the latter form is used, it means commonpagesize bytes of runtime memory will be saved at the expense of up to commonpagesize wasted bytes in the on-disk file.

- o This expression can only be used directly in SECTIONS commands, not in any output section descriptions and only once in the linker script. Commonpagesize should be less or equal to maxpagesize and should be the system page size the object wants to be optimized for Example:
  . = DATA_SEGMENT_ALIGN(0x10000, 0x2000);
- o DATA_SEGMENT_END(exp)
  This defines the end of data segment for DATA_SEGMENT_ALIGN evaluation purposes.
- o . = DATA_SEGMENT_END(.);
- o DEFINED(symbol)

  Return 1 if symbol is in the linker global symbol table and is defined, otherwise return 0. You can use this function to provide default values for symbols. For example, the following script fragment shows how to set a global symbol _begin' to the first location in the _.text' section—but if a symbol called _begin' already existed, its value is preserved:

```
SECTIONS { ...
.text : {
begin = DEFINED(begin) ? begin : . ;
...
}
...
}
```

- LOADADDR(section)

  Return the absolute LMA of the named section. This is normally the same as ADDR, but it may be different if the AT attribute is used in the output section definition

- MAX(exp1, exp2)
  Returns the maximum of exp1 and exp2.

- MIN(exp1, exp2)
  Returns the minimum of exp1 and exp2.

- NEXT(exp)

  Return the next unallocated address that is a multiple of exp. This function is closely related to ALIGN(exp); unless you use the MEMORY command to define discontinuous memory for the output file, the two functions are equivalent.

- SIZEOF(section)

  Return the size in bytes of the named section, if that section has been allocated. If the section has not been allocated when this is evaluated, the linker will report an error. In the following example, symbol_1 and symbol_2 are assigned identical values:

```
SECTIONS{ ...
.output {
.start = . ;
...
.end = . ;
}
symbol_1 = .end - .start ;
symbol_2 = SIZEOF(.output);
... }
```

- SIZEOF_HEADERS

- sizeof_headers

  Return the size in bytes of the output file's headers. This is information which appears at the start of the output file. You can use this number when setting the start address of the first section, if you choose, to facilitate paging. When producing an ELF output file, if the linker script uses the SIZEOF_HEADERS built-in function, the linker must compute the number of program headers before it has determined all the section addresses and sizes. If the linker later discovers that it needs additional program headers, it will report an error ‗not enough room for program headers'. To avoid this error, you must avoid using the SIZEOF_HEADERS function, or you must rework your linker script to avoid forcing the linker to use additional program headers, or you must define the program headers yourself using the PHDRS command

- Implicit Linker Script

  If you specify a linker input file which the linker can not recognize as an object file or an archive file, it will try to read the file as a linker script. If the file can not be parsed as a linker script, the linker will report an error. An implicit linker script will not replace the default linker script. Typically an implicit linker script would contain only symbol assignments, or the INPUT, GROUP, or VERSION commands. Any input files read because of an implicit linker script will be read at the position in the command line where the implicit linker script was read. This can affect archive searching.

2.4.4.4 Link-Order Optimization

If you have done a lot of development work, you have probably noticed that the order in which you link your files can have a significant effect on performance. By changing the link order, you are changing the way the executable file lies in the instruction cache. The cache is a fast area of memory that stores pages of instructions so that the processor does not have to go back to slower parts of memory (or even worse, the disk) for every new instruction. Certain link orders minimize instruction-cache miss. The effect usually is not large, but in pathological cases (a really bad link order on a machine that is very sensitive to cache miss), link optimization can speed up runtime by 50 percent. In general, it is a good idea to place modules that call each other near each other on the command line. The reasoning behind the heuristic is simple: if function A makes many calls to function B, and both A and B can fit into the cache simultaneously, you will not pay a penalty for cache miss. Your best chance of fitting both functions into the cache simultaneously occurs when they are located next to each other in the object file.

The –M option, which produces a load map, shows you how the object file is arranged; it will help you investigate cache performance.

Normally, rearranging the order of the object modules on the command line is sufficient for experimenting wit linkorder optimization. However, you can get very fine control over your executable file by writing a linker script. If you have a thorough knowledge of your target machine‘s architecture, you may be able to use this to your advantage –though you will probably reach the point of diminishing returns fairly quickly.

In simple words, during the linkage phase of program compilation, the linker relocates program units in an attempt to improve locality of reference.

For example, if a procedure references another procedure, the linker may make the procedures adjacent in the load module, so that both procedures fit into the same page of virtual memory. This can reduce paging overhead. When the first procedure is referenced for the first time and the page containing it is brought into real memory, the second procedure is ready for use without additional paging overhead.

In very large programs where paging occurs excessively for pages of your program's code, you may decide to impose a particular link order on the linker. You can do this by arranging control sections in the order you want them linked, and by using the -bnoobjreorder option to prevent the linker from reordering. A control section is the smallest replaceable unit of code or data in an object module.

```
-bnoobjreorder option
```

-bnoobjreorder option is a loader option that blocks the loader from reordering the objects within the executable file (which the loader normally does in order to optimize paging behavior). This option must be given or Totalview will not be able to find objects within the program. Making this the default is a local modification.

However, there are a number of risks involved in specifying a link order. Any link reordering should always be followed by thorough performance testing to demonstrate that your link order gives superior results for your program over the link order that the linker chooses. Take the following points into account before you decide to establish your own link order:

a. You must determine the link order for all control sections in your program. The control sections must be presented to the linker in the order in which you want to link them. In a large program, such an ordering effort is considerable and prone to errors.

b. A performance benefit observed during development of a program can become a performance loss later on, because the changing code size can cause control sections that were previously located together in a page to be split into separate pages.

c. Reordering can change the frequency of instruction cache-line collisions. On implementations with an instruction cache or combined data and instruction cache that is two-way set-associative, any line of program code can only be stored in one of two lines of the cache. If three or more short, interdependent procedures have the same cache-congruence class, instruction-cache thrashing can reduce performance. Reordering can cause cache-line collisions where none occurred before. It can also eliminate cache-line collisions that occur when -bnoobjreorder is not specified.

If you attempt to tune the link order of your programs, always test performance on a system where total real storage and memory utilization by other programs are similar to the anticipated working environment. A link order that works on a quiet system with few tasks running can cause page thrashing on a busier system.

2.4.4.5 The C Runtime (crt0)

To link and run C or C++ programs, you need to define a small module, usually written in assembly as crt0.s, but sometimes written as a C file as crt0.c, to initialize hardware using C conventions before calling main. There are some examples available in the sources of Newlib C library; perform a simple search through the source tree for crt0.s code, as well as examples of system calls with sub-routines. To write your own crt0.s or crt0.c module, you need specific information about your target.

a. C Runtime Environment(crt0)

- The memory map, showing the size of available memory and memory location
- The way the stack grows
- The output format in use

At a minimum, your crt0.s module must provide the following processes to link and run your programs:

- Define the symbol, start (_start in assembler code). Execution begins with it.
- Set up sp , the stack pointer

  Choose where to store your stack within the constraints of your target's memory map, the simplest choice being a fixed-size area somewhere in the uninitialized data section (often bss). Whether you choose a low address or a high address in this area depends on the direction your stack grows.
- Initialize all memory in the uninitialized-data bss section to zero

  Use a linker script to define symbols such as bss_start and bss_end to record the boundaries of this section; then you can use a ¯for‖ loop to initialize all memory between them in the crt0.s module.
- Define an _exit subroutine (the C name in your assembler code). Use the label _ _exit (with two leading underbars); its precise behavior depends on the details of your system and on your choice of your system's behavior. Possibilities include trapping back to the boot monitor (if there is one), to the loader (if there is no monitor), or to the start symbol.
- If your target has no monitor for output from the debugger, set up the hardware exception handler in the crt0.s module.
- Perform other hardware-dependent initialization (for example, initializing an mmu or an auxiliary floating-point chip).

- Define low-level input and output subroutines (for example, the crt0.s module is a convenient place to define the minimal assembly-level routines).

b. crt0, Main Startup file

The crt0 (C Runtime 0) file contains the initial startup code for embedded development; its object is linked in first and bootstraps the rest of your application. A crt0 file is available for most platforms, although you may want your own crt0 file for each target. The crt0 file is usually written in assembler and named crt0.s.

The crt0 file defines a special symbol such as _start , which is both the default base address for the application and the first symbol in the executable binary image.

If you plan to use any routines from the standard C library, you will also need to implement the functions on which newlib depends.

The crt0 file initializes everything that your program requires for the host and target. This initialization section varies for some systems:

- If you are developing an application that gets downloaded to a ROM monitor, there is usually no need for special initialization because the ROM monitor handles it for you.
- If you plan to burn your code in a ROM, the crt0 file typically does all of the hardware initialization required to run an application. This can include initializing serial ports and running a memory check; however, results vary depending on your hardware.

The following shows a basic initialization of a crt0.s file; your file may differ, although there will be a similar initialization process.

- If you are developing an application that gets downloaded to a ROM monitor, there is usually no need for special initialization because the ROM monitor handles it for you.
- If you plan to burn your code in a ROM, the crt0 file typically does all of the hardware initialization required to run an application. This can include initializing serial ports and running a memory check; however, results vary depending on your hardware.

  The crt0 file initializes everything that your program requires for the host and target. This initialization section varies for some systems:

- Set up concatenation macros:

```
#define  CONCAT1(a,  b)  CONCAT2(a,
b) #define CONCAT2(a, b) a ## b
```

- Set up label macros:

```
#ifndef  _  _USER_LABEL_PREFIX_
_           #define        _
_USER_LABEL_PREFIX_ _ #endif
#define SYM(x) CONCAT1 (_ _USER_LABEL_PREFIX_ _,
x)
```

These macros make the code portable between COFF and a.out ; COFF has _ _ (two underlines) before its global symbol names, while a.out has none.

- Set up register names (with the appropriate prefix):

  ```
  #ifndef _ _ REGISTER_PREFIX _
  _ #define _ _ REGISTER_PREFIX
  _ _ #endif
  /* Use the right prefix for registers. */
  #define REG(x) CONCAT1 _ _ REGISTER_PREFIX _ _ ,
  x)
  #define d0 REG (d0)
  #define d1 REG (d1)
  #define d2 REG (d2)
  #define d3 REG (d3)
  #define d4 REG (d4)
  #define d5 REG (d5)
  #define d6 REG (d6)
  #define d7 REG (d7)
  #define a0 REG (a0)
  #define a1 REG (a1)
  #define a2 REG (a2)
  #define a3 REG (a3)
  #define a4 REG (a4)
  #define a5 REG (a5)
  #define a6 REG (a6)
  #define fp REG (fp)
  #define sp REG (sp)
  ```

  Register names are for portability between assemblers. Some register names have % or $ prepended to them.

- Set up space for the stack by assigning part of memory. (Although you can also set this allocation in the linker script, it typically is done at this point.)

  ```
  .set stack_size, 0x2000.
  comm SYM (stack), stack_size
  ```

- Define empty space for an environment:

  ```
  .data
  .align 2
  ```

```
SYM (environ):
.long 0
```

This call is unlike most for a ROM monitor. Generally it is best to set up a valid address and then pass the address to main() . In this way if an application checks for an empty environment, it finds one.

- Set up a few global symbols:

  ```
  .align 2
  .text
  .global SYM (stack) .global SYM (main)
  .global SYM (exit)
  .global __ bss_start
  ```

  The last line really should not be SYM (_ _bss_start) , since declaring _ _bss_start this way ensures the appropriate setting of its value in the linker script.

- Set up the start global symbol for the linker to use as the default address for the .text section. This helps your program run.

  ```
  SYM (start):
  link a6, #-8
  moveal #SYM (stack) + stack_size, sp
  ```

c. crt0 zeroes the .bss section.

Clear the .bss section for uninitialized data using the following example. Initialize all of the addresses in the .bss section to zero so programs will get predictable results when there is no check for the default values of new variables.

```
moveal # _ _ bss_start,
a0 moveal #SYM (end), a1
1:
movel #0, (a0)
leal 4(a0), a0
cmpal a0, a1
bne 1b
```

Applications can get unpredictable effects with the .bss section left uncleared, causing particular problems with some implementations of malloc() calls.

d. crt0 calls main()

If your ROM monitor supports it, set up argc and argv for command line arguments and an environment pointer before the call to main() . F or GNU C++ compiling, the code generator inserts a branch to _ _main at the top of your main() routine, while the GNU C++ compiler uses _ _main to initialize its internal tables and then returns control to your main() routine. For crt0 to call your main() routine, use the following example. Set up the environment pointer and jump to main() and call the main routine from the application with main (argc, argv, environ) , using argv as a pointer to NULL .

```
pea 0
pea SYM (environ)
pea sp@(4)
pea 0
jsr SYM (main)
movel d0, sp@-4
```

e. crt0 calls (exit)

After main() runs, the crt0 file then returns control of the hardware from the application. On some hardware there is nothing to return to, especially if your program is in ROM. If that is the case, you need to do a hardware reset or branch back to the original start address. Using a ROM monitor you can usually call a user trap to provide the ROM control. Pick a safe vector with no side effects. Some ROMs have a built-in trap handler for just this case. Exit from the application with _exit ; normally, this causes a user trap to return to the ROM monitor for another run. Proceed by using the following example:

```
SYM (exit):
trap #0
```

Both rom68k and bug can handle an exception of 0 with no side effects; although the bug monitor has a user-caused trap that returns control to the ROM monitor, the bug monitor is more portable.

2.4.4.6 Map files

Basically Map File tells you in what portions of memory your .text .rodata .bss .data and other sections and their respective symbol contents are loaded and relocated to in case your image address is different from the load address.

A mapfile is a text file that contains the following information about the program being linked:

- The module name, which is the base name of the file

A list of groups in the program, with each group's start address (as section: offset), length, group name, and class

- A list of public symbols, with each address (as section:offset), symbol name, flat address, and .obj file where the symbol is defined
- The entry point (as section:offset)

Below are some command line option mentioned for generation of Map File:

With -M or -Map option to the linker, we can create a map file. You can use -MAP option in the following way:

```
-Map <filename> create map listing in file
```

The linker names the mapfile with the base name of the program and the extension .map. The optional filename allows you to override the default name for a mapfile.In simple term, if a file name is not supplied then the map file name will be the same as the executable file name with a ".map" file extension.

```
--print-map
```

This command prints a link map to the standard output. A link map provides information about the link, including the following:

- Where object files and symbols are mapped into memory.
- How common symbols are allocated.
- All archive members included in the link, with a mention of the symbol which caused the archive member to be brought in.

```
--cref
```

This tells the linker to add cross-reference information to the map file. This information is useful for determining why a particular module or object was linked into the executable.

2.4.5 Using the Unifile utility

2.4.5.1 Supported File Formats

The Unifile utility merges files of the following file formats:

- S-Record / Motorola format files (commonly known as Mot file)
- Intel format files (commonly known as Hex file)

2.4.5.2 Unifile Utility Description

The Unifile utility merges two or more files of Motorola format (commonly known as Mot file or S-Record file) or Intel format (commonly known as Hex file) to generate a single Mot file or a Hex file. Input files are merged after successful validation of their section addresses. Sections from the input files are merged in increasing address order (sections at lower addresses are merged first).

Users can specify the name of the output file using the -output[=OutputFile] option. If an output filename is not specified, a default name is used:

"<Name of the first input file>_combine.mot/hex || .

The Unifile utility determines the format (Mot or Hex) of the input file by checking the first character of each input file: ‗S' in the case of a Mot file, or ‗:' in the case of a Hex file. If the first character is not ‗S' or ‗:', an error message is displayed and the utility terminates.

2.4.5.3 Unifile Utility Options

The following options are supported by the Unifile utility:

Option	Description
-output[=OutputFile]	Specify output filename
-list	Create information file
-s9	Output s9 at end record

2.4.5.4 Command Line Usage

Command
`#[1] unifile <Input_Mot/Hex_file_1> <Input_Mot/Hex_file_2>`
`e.g. # unifile 1.mot 2.mot`
This creates an output file with the name 1_combine.mot.

[1] Note that # indicates a command prompt under either Windows or Linux.

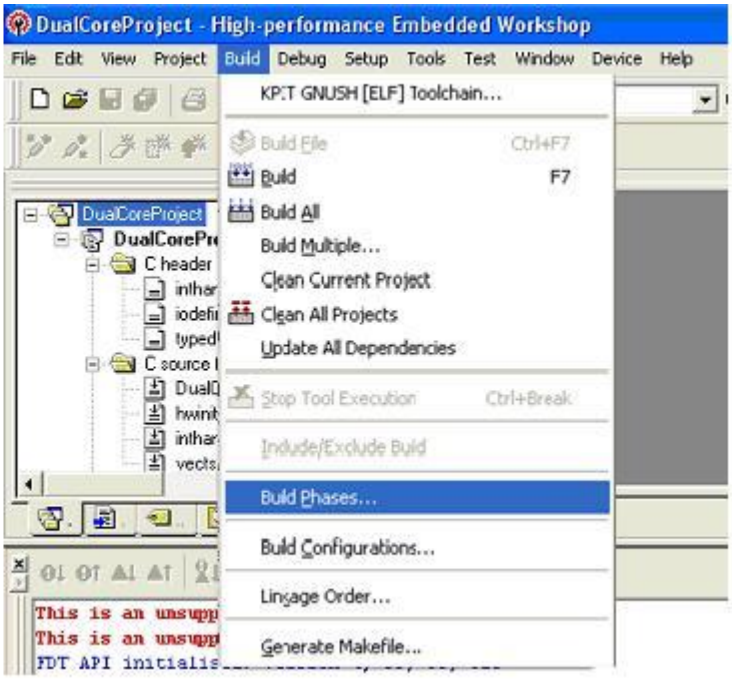
Command
`# unifile <Input_file_1> <Input_file_2> -output[=OutputFile]`
`e.g. # unifile 1.mot 2.mot -output=3.mot`
This creates an output file named 3.mot.

Command
`# unifile <Input_file_1> <Input_file_2> -output[=OutputFile] –list`
`e.g. # unifile 1.mot 2.mot -output=3.mot –list`
This creates an output file named 3.mot and a list (information) file with the same name as the output file i.e., 3.txt.

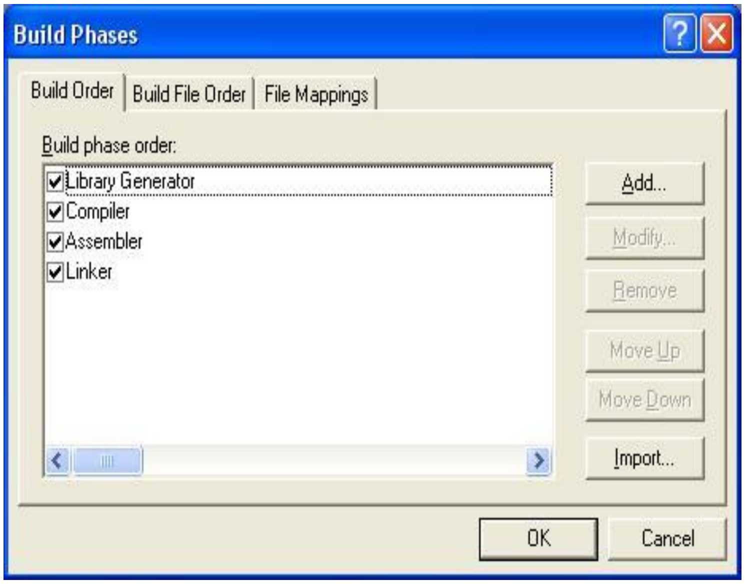
2.4.5.5 HEW Integration: Creation of custom build phase in HEW

The Unifile utility can be used as part of a project ¯Build‖ in the HEW IDE for Windows. To add a custom build phase for Unifile utility, please follow the steps below:

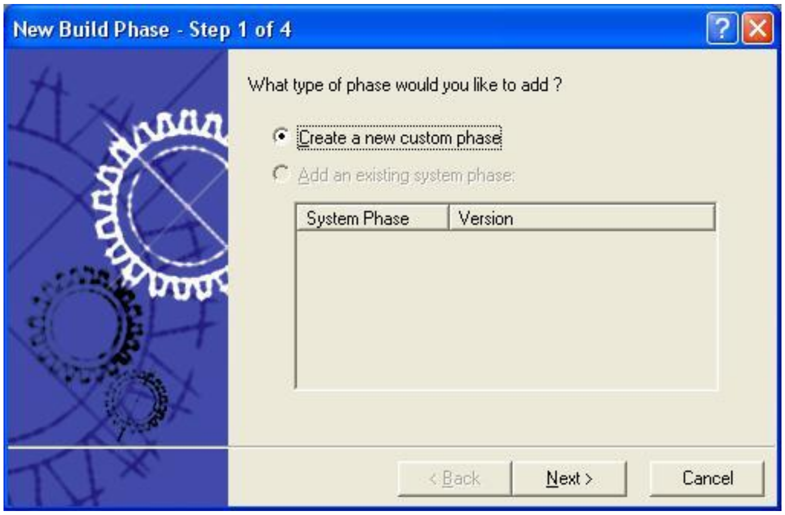
a. Click ¯Build‖ on the main menu, then click ¯Build Phases‖:



b. This will show the following dialog:



c. Click the ¯Add‖ button. The ¯New Build Phase‖ dialog will be shown:



New Build Phase - Step 1 of 4

What type of phase would you like to add ?

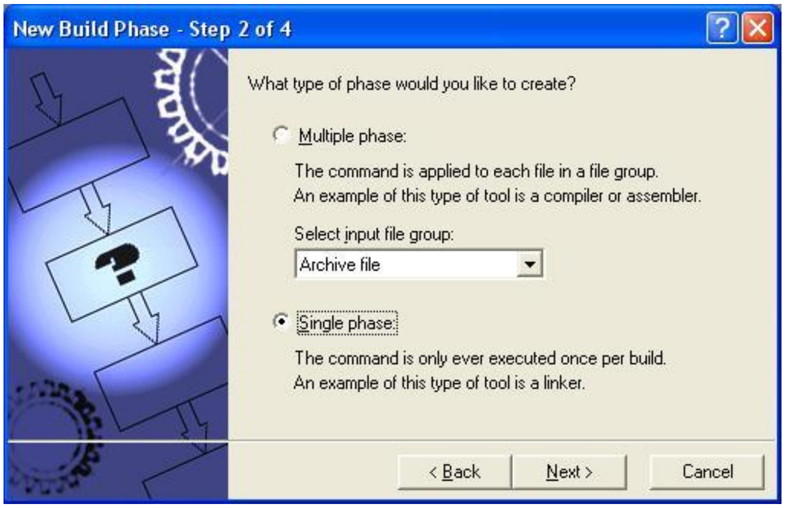
● Create a new custom phase

○ Add an existing system phase:

System Phase	Version

< Back Next > Cancel

d. Choose the default ¯Create a new custom phase‖ and click ¯Next‖.



New Build Phase - Step 2 of 4

What type of phase would you like to create?

○ Multiple phase:

The command is applied to each file in a file group.
An example of this type of tool is a compiler or assembler.

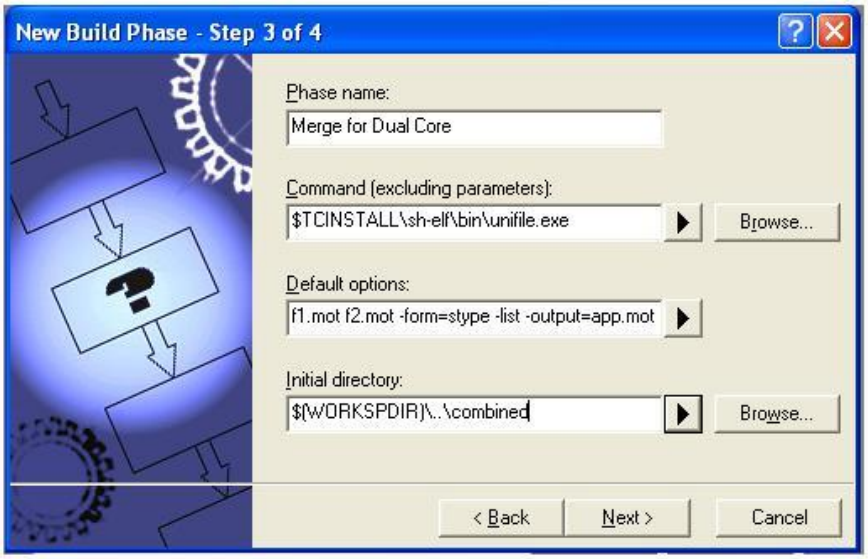
Select input file group:
Archive file

● Single phase:

The command is only ever executed once per build.
An example of this type of tool is a linker.

< Back Next > Cancel

e. Select ¯Single phase‖ (***Note***: this is not the default option) and click ¯Next‖.



New Build Phase - Step 3 of 4

Phase name: Merge for Dual Core

Command (excluding parameters): $TCINSTALL\sh-elf\bin\unifile.exe

Default options: f1.mot f2.mot -form=stype -list -output=app.mot

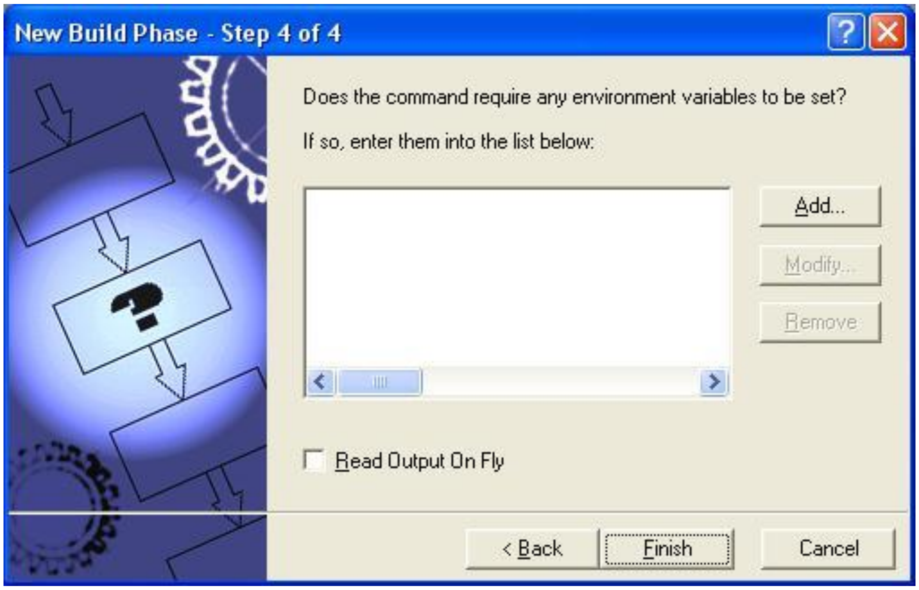
Initial directory: $(WORKSPDIR)\..\combined

f. Enter the following information according to your project and click ¯Next‖.

Field	Purpose	Example(s)
Phase name	Name for this build phase	¯Merge for Dual Core‖ or ¯Unifile‖
Command	Path to the Unifile utility	¯$TCINSTALL\sh-elf\bin\unifile.exe‖ or ¯C:\GNUSH\bin\unifile.exe‖
Default options	Command-line options for the Unifile utility	¯C:\path\to\motfile1.mot C:\path\to\motfile2.mot -form=stype -list -output C:\path\to\merged-motfile.mot‖ or ¯cpu1.mot cpu2.mot –form=stype -list -output merged.mot‖
Initial directory	Working directory for the Unifile utility during execution.	¯C:\path\to\workingdirectory‖ or ¯$(WORKSPDIR)\..\combined‖

Note: You can use placeholders such as $WORKSPDIR, $PROJDIR, $TCINSTALL, etc. in place of your HEW workspace directory path. This will allow your HEW workspace to be moved to a different file-system path and still build correctly.

g. No environment variables need to be set. Click on ¯Finish‖

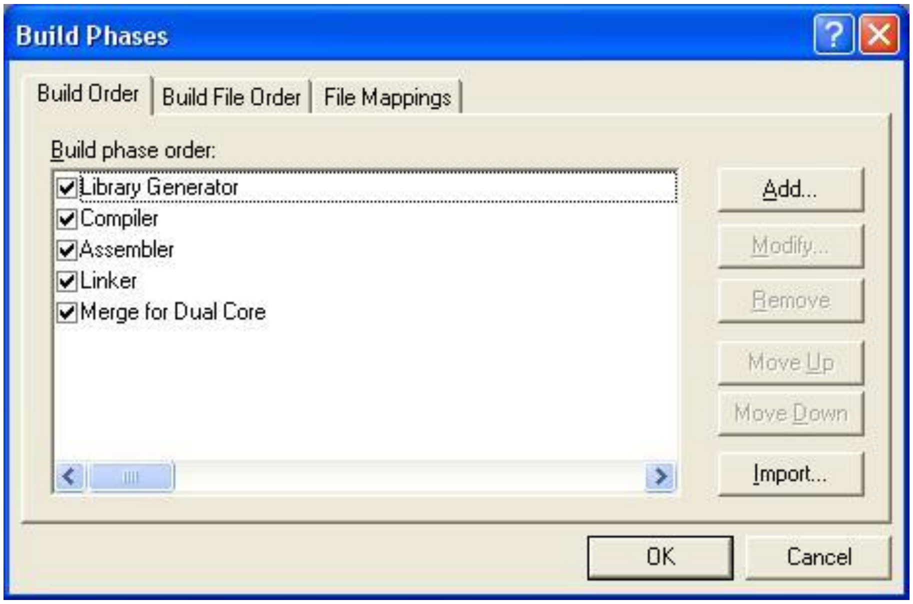


Does the command require any environment variables to be set?

If so, enter them into the list below:

☐ Read Output On Fly

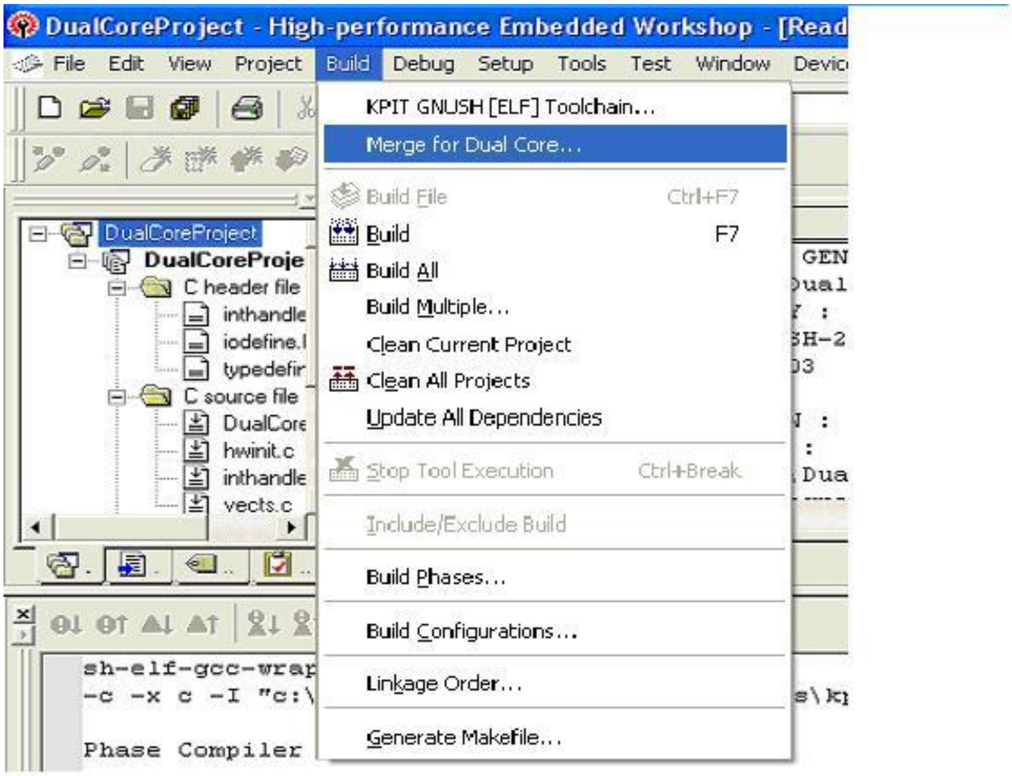
h. The new build phase is now visible in the list of build phases. Its position at the end of the list of build phases means that it will be executed after the main application has been built, which is expected. Click ¯OK‖ to close this dialog.



Build phase order:

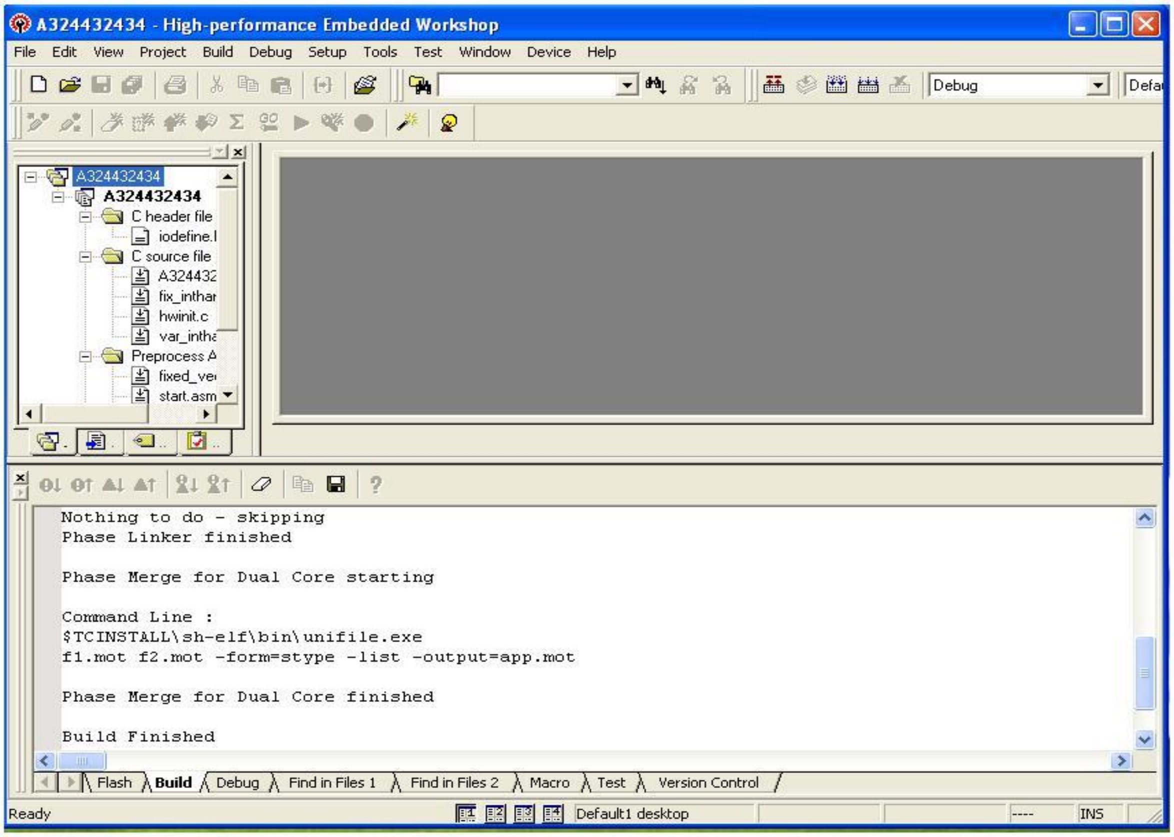
☑ Library Generator
☑ Compiler
☑ Assembler
☑ Linker
☑ Merge for Dual Core

i. You should now see the new build phase (Merge for Dual Core) in the ¯Build‖ menu:



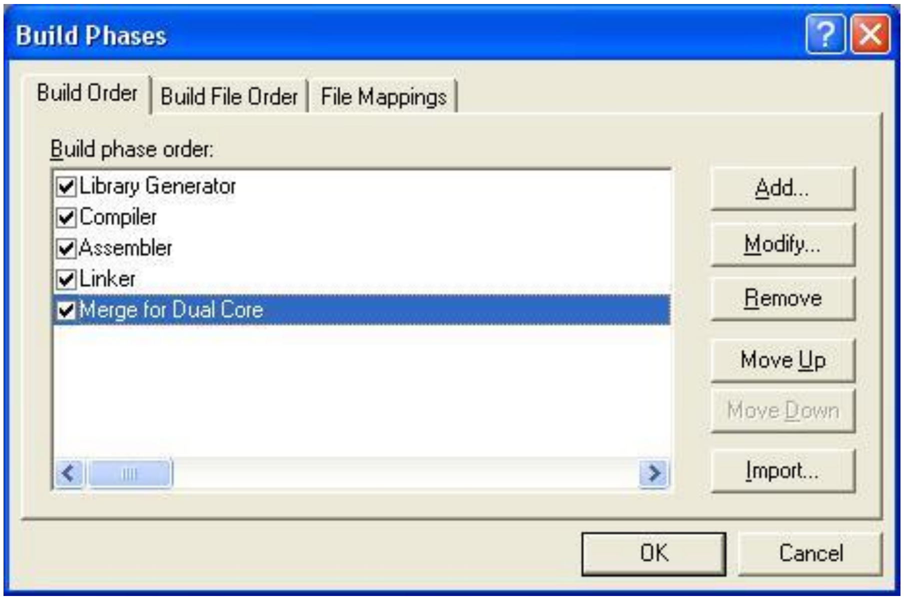
```
sh-elf-gcc-wrap
-c -x c -I "c:\

Phase Compiler
```

j. Additionally, by building your project (e.g. by choosing ―Build All‖ from the ―Build‖ menu), you should see the new build phase is executed:

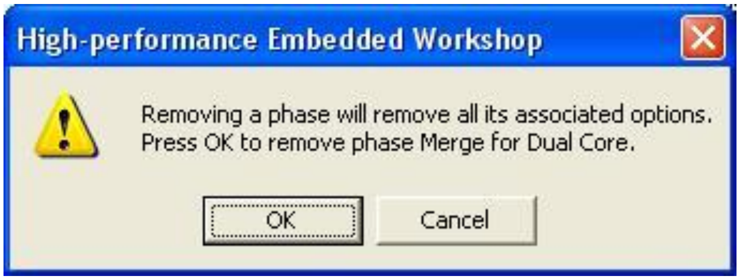

A324432434 - High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Device Help
Debug
A324432434
A324432434
C header file
iodefine.l
C source file
A324432
fix_inthar
hwinit.c
var_intha
Preprocess A
fixed_ve
start.asm
Flash Build Debug Find in Files 1 Find in Files 2 Macro Test Version Control
Ready
Default1 desktop
INS

```
Nothing to do - skipping
Phase Linker finished

Phase Merge for Dual Core starting

Command Line :
$TCINSTALL\sh-elf\bin\unifile.exe
f1.mot f2.mot -form=stype -list -output=app.mot

Phase Merge for Dual Core finished

Build Finished
```

k. If at a later stage you would like to remove this new build phase, choose ¯Build Phases‖ again from the ¯Build‖ menu to display the ¯Build Phases‖ dialog:



l. To remove the ¯Merge for Dual Core‖ build phase, select it and click the ¯Remove‖ button. A dialog will be displayed to confirm this operation:



High-performance Embedded Workshop

Removing a phase will remove all its associated options.
Press OK to remove phase Merge for Dual Core.

2.4.5.6 Error Information

Error Number and Level	Error Message	Error Description
[0000 (w)]	Entry address conflicts ¯file_name1‖ Address=0X********** Two or more files of a different entry address of Address=0X**********	Entry address in first input file conflicts with the entry address in the other input file/s.
[1000 (E)]	Overlap address ¯file_name1‖ , ¯file_name2‖ Address=0X********	The sections in the input files ¯file_name1‖ and ¯file_name2‖ overlap at the address "Address=0X********".
[1001 (E)]	Cannot open file ¯file_name1‖	The file ¯file_name1‖ cannot be opened for processing.
[1002 (E)]	Cannot read file ¯file_name1‖	The file ¯file_name1‖ cannot be opened in READ mode.
[1003 (E)]	Cannot write file ¯file_name1‖	The file ¯file_name1‖ cannot be opened in WRITE mode.
[1004 (E)]	Cannot get memory	Insufficient memory available for allocation.

[1005 (E)]	Illegal ¯file_type‖ file format ¯fail_name1‖	The format of the input file is not supported.
		Supported file formats are:
		Mot = Motorola file format / S-Record file format
		Hex = Intel file format

[1006 (E)]	Invalid command parameter ¯option‖	The option specified is invalid.
[1007 (E)]	Command parameter specified twice ¯option‖	The command parameter ¯option‖ is specified multiple times.
[9999 (－)]	Internal error	Internal processing error

2.4.6 Using the GNUSH Binary Utilities

This section will explain in details ―objdump‖ and ―readelf‖ Binary Utilities.

2.4.6.1 ―objdump‖ utility

objdump [options] objfiles

―objdump‖ displays information about one or more object files. The options control what particular information to display. This information is mostly useful to programmers who are working on the compilation tools, as opposed to programmers who just want their program to compile and work. ―*objfile*...‖ are the object files to be examined. When you specify archives, objdump shows information on each of the member object files.

The long and short forms of options, shown here as alternatives, are equivalent. At least one option from the list `-a,-d,-D,-e,-f,-g,-G,-h,-H,-p,-r,-R,-s,-S,-t,-T,-V,-x' must be given.

-a --archive-

header

If any of the objfile files are archives, display the archive header information (in a format similar to ‗ls -l'). Besides the information you could list with ‗ar tv',
`objdump -a' shows the object file format of each archive member.

--adjust-vma=*offset*

When dumping information, first add *offset* to all the section addresses. This is useful if the section addresses do not correspond to the symbol table, which can happen when putting sections at particular addresses when using a format which can not represent section addresses, such as a.out.

-b *bfdname*

--target=*bfdname*

Specify that the object-code format for the object files is *bfdname*. This option may not be necessary; *objdump* can automatically recognize many formats.

--demangle[=*style*]

Decode (*demangle*) low-level symbol names into user-level names. Besides removing any initial underscore prepended by the system, this makes C++ function names readable. Different compilers have different mangling styles. The optional demangling style argument can be used to choose an appropriate demangling style for your compiler.

-g --

debugging

Display debugging information. This attempts to parse debugging information
stored in the file and print it out using a C like syntax. Only certain types of

debugging information have been implemented. Some other types are supported by readelf -w. See section ‗readelf'.

`-e --debugging-`

`tags`

Like ‗-g', but the information is generated in a format compatible with ctags tool.

`-d --`

`disassemble`

Display the assembler mnemonics for the machine instructions from objfile. This option only disassembles those sections which are expected to contain instructions.

`-D --`

`disassemble-all`

Like ‗-d', but disassemble the contents of all sections, not just those expected to contain instructions.

`--prefix-addresses`

When disassembling, print the complete address on each line. This is the older disassembly format.

`-EB`

`-EL --`

`endian={big|little}`

Specify the endianness of the object files. This only affects disassembly. This can be useful when disassembling a file format which does not describe endianness information, such as S-records.

`-f --file-`

`headers`

Display summary information from the overall header of each of the *objfile* files.

`--file-start-context`

Specify that when displaying interlisted source code/disassembly (assumes ‗ -S') from a file that has not yet been displayed, extend the context to the start of the file.

`-h --section-`

`headers --`

`headers`

Display summary information from the section headers of the object file. File segments may be relocated to nonstandard addresses, for example by using the ‗-Ttext', ‗-Tdata', or ‗-Tbss' options to ld. However, some object file formats, such as a.out, do not store the starting address of the file segments. In those situations, although ld relocates the sections correctly, using ‗objdump -h‘ to list the file section headers cannot show the correct addresses. Instead, it shows the usual addresses, which are implicit for the target.

`-H --`

`help`

Print a summary of the options to objdump and exit.

`-i --`

`info`

Display a list showing all architectures and object formats available for specification with ‗-b‘ or ‗-m‘.

`-j` *name*

`--section=`*name*

Display information only for section *name*.

`-l --line-`

`numbers`

Label the display (using debugging information) with the filename and source line numbers corresponding to the object code or relocs shown. Only useful with ‗-d‘, ‗-D‘, or ‗-r‘.

`-m` *machine*

`--architecture=`*machine*

Specify the architecture to use when disassembling object files. This can be useful when disassembling object files which do not describe architecture information, such as S-records. You can list the available architectures with the ‗-i‘ option.

`-M` *options*

`--disassembler-options=`*options*

Pass target specific information to the disassembler. Only supported on some targets. If it is necessary to specify more than one disassembler option then multiple `-M' options can be used or can be placed together into a comma separated list.

`no-aliases`

Print the 'raw' instruction mnemonic instead of some pseudo instruction mnemonic. I.e., print ‗daddu‘ or 'or' instead of ‗move‘, ‗sll‘ instead of ‗nop‘, etc.

`gpr-names=`*ABI*

Print GPR (general-purpose register) names as appropriate for the specified ABI. By default, GPR names are selected according to the ABI of the binary being disassembled.

`fpr-names=ABI`

Print FPR (floating-point register) names as appropriate for the specified ABI. By default, FPR numbers are printed rather than names.

`cp0-names=ARCH`

Print CP0 (system control coprocessor; coprocessor 0) register names as appropriate for the CPU or architecture specified by *ARCH*. By default, CP0 register names are selected according to the architecture and CPU of the binary being disassembled.

`hwr-names=ARCH`

Print HWR (hardware register, used by the rdhwr instruction) names as appropriate for the CPU or architecture specified by *ARCH*. By default, HWR names are selected according to the architecture and CPU of the binary being disassembled.

`reg-names=ABI`

Print GPR and FPR names as appropriate for the selected ABI

`reg-names=ARCH`

Print CPU-specific register names (CP0 register and HWR names) as appropriate for the selected CPU or architecture.
For any of the options listed above, *ABI* or *ARCH* may be specified as ‗numeric' to have numbers printed rather than names, for the selected types of registers. You can list the available values of *ABI* and *ARCH* using the ‗--help' option.
For VAX, you can specify function entry addresses with ‗-M entry:0xf00ba'. You can use this multiple times to properly disassemble VAX binary files that don't contain symbol tables (like ROM dumps). In these cases, the function entry mask would otherwise be decoded as VAX instructions, which would probably lead the rest of the function being wrongly disassembled.

`-p --private-`

`headers`

Print information that is specific to the object file format. The exact information printed depends upon the object file format. For some object file formats, no additional information is printed.

`-r --`

`reloc`

Print the relocation entries of the file. If used with ‗-d' or ‗-D', the relocations are printed interspersed with the disassembly.

`-R --dynamic-`

`reloc`

Print the dynamic relocation entries of the file. This is only meaningful for dynamic objects, such as certain types of shared libraries.

`-s --full-`

`contents`

Display the full contents of any sections requested. By default all non-empty sections are displayed.

`-S --`

`source`

Display source code intermixed with disassembly, if possible. Implies ‗-d‘.

`--show-raw-insn`

When disassembling instructions, print the instruction in hex as well as in symbolic form. This is the default except when ‗--prefix-addresses‘ is used.

`--no-show-raw-insn`

When disassembling instructions, do not print the instruction bytes. This is the default when ‗--prefix-addresses‘ is used.

`-W --`

`dwarf`

Displays the contents of the DWARF debug sections in the file, if any are present.

`-G --`

`stabs`

Display the full contents of any sections requested. Display the contents of the .stab and .stab.index and .stab.excl sections from an ELF file. This is only useful on systems (such as Solaris 2.0) in which .stab debugging symbol -table entries are carried in an ELF section. In most other file formats, debugging symbol-table entries are interleaved with linkage symbols, and are visible in the `--syms' output. For more information on stabs symbols, see section `Stabs Overview' in *The "stabs" debug format*.

`--start-address=`*address*

Start displaying data at the specified address. This affects the output of the `-d', `-r' and `-s' options.

`--stop-address=`*address*

Stop displaying data at the specified address. This affects the output of the `-d', `-r' and `-s' options.

-t --syms

Print the symbol table entries of the file. This is similar to the information provided by the `nm' program.

-T --dynamic-syms

Print the dynamic symbol table entries of the file. This is only meaningful for dynamic objects, such as certain types of shared libraries. This is similar to the information provided by the `nm' program when given the `-D' (`--dynamic') option.

--special-syms

When displaying symbols include those which the target considers to be special in some way and which would not normally be of interest to the user.

-V --version

Print the version number of objdump and exit.

-x --all-headers

Display all available header information, including the symbol table and relocation entries. Using ‗-x‘ is equivalent to specifying all of ‗-a -f -h -p -r -t‘.

-w --wide

Format some lines for output devices that have more than 80 columns. Also do not truncate symbol names when they are displayed.

-z --disassemble-zeroes

Normally the disassembly output will skip blocks of zeroes. This option directs the disassembler to disassemble those blocks, just like any other data.

2.4.6.2 ¯readelf‖

readelf *option*[...] *elffiles*

‗readelf‘ displays information about one or more ELF format object files. The options control what particular information to display. ¯*elffile*...‖ are the object files to be examined. 32-bit and 64-bit ELF files are supported, as are archives containing ELF files.

This program performs a similar function to ¯objdump‖ but it goes into more detail and it exists independently of the BFD library, so if there is a bug in BFD then ¯readelf‖ will not be affected.

The long and short forms of options, shown here as alternatives, are equivalent. At least one option besides ‗-v‘ or ‗-H‘ must be given.

```
-a -
-all
```

Equivalent to specifying ‗--file-header‘, ‗--program-headers‘, ‗–sections‘, ‗–symbols‘, ‗–relocs‘, ‗–dynamic‘, ‗–notes‘ and ‗--version-info‘.

```
-h --file-
header
```

Displays the information contained in the ELF header at the start of the file.

```
-l --program-
headers --
segments
```

Displays the information contained in the file's segment headers, if it has any.

```
-S --
sections
--section-headers
```

Displays the information contained in the file's section headers, if it has any.

```
-g --section-
groups
```

Displays the information contained in the file's section groups, if it has any.

```
-t --section-
details
```

Displays the detailed section information. Implies ‗-S‘.

`-s --`

`symbols -`

`-syms`

Displays the entries in symbol table section of the file, if it has one.

`-e --`

`headers`

Display all the headers in the file. Equivalent to ‗-h -l -S‘.

`-n --`

`notes`

Displays the contents of the NOTE segments and/or sections, if any.

`-r --`

`relocs`

Displays the contents of the file's relocation section, if it has one.

`-u --`

`unwind`

Displays the contents of the file's unwind section, if it has one. Only the unwind sections for IA64 ELF files are currently supported.

`-d --`

`dynamic`

Displays the contents of the file's dynamic section, if it has one.

`-V --version-`

`info`

Displays the contents of the version sections in the file, it they exist.

`-A --arch-`

`specific`

Displays architecture-specific information in the file, if there is any.

`-D --use-`

`dynamic`

When displaying symbols, this option makes readelf use the symbol table in the file's dynamic section, rather than the one in the symbols section.

`-x <number or name> --hex-dump=<number or name>`

Displays the contents of the indicated section as a hexadecimal dump. A number identifies a particular section by index in the section table; any other string identifies all sections with that name in the object file.

`-w[liaprmfFsoR]`

`--debug-dump[=line,=info,=abbrev,=pubnames,=aranges,=macro,=frames, =frames-interp,=str,=loc,=Ranges]`

Displays the contents of the debug sections in the file, if any are present. If one of the optional letters or words follows the switch then only data found in those specific sections will be dumped.

`-I --histogram`

Display a histogram of bucket list lengths when displaying the contents of the symbol tables.

`-v --version`

Display the version number of readelf.

`-W --wide`

Don't break output lines to fit into 80 columns. By default readelf breaks section header and segment listing lines for 64-bit ELF files, so that they fit into 80 columns. This option causes readelf to print each section header resp. each segment one a single line, which is far more readable on terminals wider than 80 columns.

`-H --help`

Display the command line options understood by readelf.

2.4.6.3 Libgen

```
libgen [`-S'|`--select-lib='] [`-
       H'|`--header-files='] [`-
       C'|`--compiler-options='] [`-
       A'|`--assembler-options=']
       [`-o'|`--output'] [`-I'|`--
       interactive'] [`-v'|`--
       version'] [`-h'|`--help']
```

The GNU library generator tool libgen, builds the Newlib and Optimized libraries with the user specified options for the Renesas targets. It helps in generating the libraries that are fine tuned with the user's application.

The libgen operates in two modes viz. command line mode and interactive mode.

In the command line mode, the entire command line consisting of options and their suboptions has to be provided.

In the interactive mode, a menu is displayed on the command line. Following are the various menu options,

`Select Library`

Supported options are newlib and optlib. By default 'optlib' will be selected.

`Select Headers`

Supported options are all, stdio, stdlib, string, ctype and math.

`Enter Compiler Options`

Specify compiler options. By default '-fno-builtin' and '-c' options will be passed.

`Enter Assembler Options`

Specify assembler options.

`Enter Path & Name for Library`

Specify the path and name of the archive. By default library name is 'libout.a'.

The libgen utility builds the user selected library from the library sources installed with the toolchain. Depending upon the headers specified by the user, library sources will be compiled using gcc utility and archived using ar utility.

The options which can be used with libgen are listed below.

`-I --`
`interactive`

Specify the command line options in user friendly interactive mode.

`-S --select-lib=`

Specify the sources to be used for building the library. If nothing is specified by the user under this option, 'optlib' will be selected..

`-H --header-files=`

Specify the library header file to be compiled. If nothing is specified by the user under this option, an error message will be displayed. The sub-options should be seperated by ','.

`-C --compiler-options=`

Specify the compiler options i.e. cpu specific options like -ms, -mh, -m2, -m4a, -mcpu=m16c etc and/or optimization options like -O2, -Os, -fomit-frame-pointer etc. The sub -options should be seperated by ','. The '-fno-builtin' and '-c' options are passed by default.

`-A --assembler-options=`

Specify the assembler options (in case of assembly files). These options should not be prefixed with '-Wa,'. The sub-options should be seperated by ','.

`-o --output=`

Specify the name for the library archive. By default, the library is generated in the current working directory for command line applications.

`-h --help`

Print (on the standard output) the description of the command line options supported by the 'libgen' tool.

`-v --version`

Show the version number of the libgen and exit.

2.4.6.4 Other Binary Utilities

This section explains the other binary utilities in brief. **ar - Archive Editor**

```
ar [-]p[mod [relpos] [count]] archive [member...]
```

```
ar -M [ <mri-script ]
```

The GNU ar program creates, modifies, and extracts from archives. An *archive* is a single file holding a collection of other files in a structure that makes it possible to retrieve the original individual files (called *members* of the archive).

The original files' contents, mode (permissions), timestamp, owner, and group are preserved in the archive, and can be restored on extraction.

GNU ar can maintain archives whose members have names of any length; however, depending on how ar is configured on your system, a limit on member-

name length may be imposed for compatibility with archive formats maintained with other tools. If it exists, the limit is often 15 characters (typical of formats related to a.out) or 16 characters (typical of formats related to coff).
`ar` is considered a binary utility because archives of this sort are most often used as *libraries* holding commonly needed subroutines.
ar creates an index to the symbols defined in relocatable object modules in the archive when you specify the modifier ‗s‘. Once created, this index is updated in the archive whenever ar makes a change to its contents (save for the `q' update operation). An archive with such an index speeds up linking to the library, and allows routines in the library to call each other without regard to their placement in the archive.

nm

```
nm [`-a'|`--debug-syms'] [`-g'|`--extern-only']
   [`-B'] [`-C'|`--demangle'[=style]] [`-D'|`--dynamic']
   [`-S'|`--print-size'] [`-s'|`--print-armap']
   [`-A'|`-o'|`--print-file-name'][`--special-syms']
   [`-n'|`-v'|`--numeric-sort'] [`-p'|`--no-sort']
   [`-r'|`--reverse-sort']     [`--size-sort']     [`-u'|`--
undefined-only']
   [`-t' radix|`--radix='radix] [`-P'|`--portability']
   [`--target='bfdname] [`-f'format|`--format='format]
   [`--defined-only']     [`-l'|`--line-numbers']     [`--no-
demangle']
   [`-V'|`--version'] [`-X 32_64'] [`--help']  [objfile...]
```

GNU nm lists the symbols from object files *objfile*.... If no object files are listed as arguments, nm assumes the file `a.out'.
For each symbol, nm shows:

a. The symbol value, in the radix selected by options (see below), or hexadecimal by default.

b. The symbol type. At least the following types are used; others are, as well, depending on the object file format. If lowercase, the symbol is local; if uppercase, the symbol is global (external).
   - A: The symbol's value is absolute, and will not be changed by further linking.
   - B: The symbol is in the uninitialized data section (known as BSS).
   - C: The symbol is common. Common symbols are uninitialized data. When linking, multiple common symbols may appear with the same name. If the symbol is defined anywhere, the common symbols are

treated as undefined references. For more details on common symbols, see the discussion of --warn-common in section `Linker options' in *The GNU linker*.

- D: The symbol is in the initialized data section.
- G: The symbol is in an initialized data section for small objects. Some object file formats permit more efficient access to small data objects, such as a global ‗int‘ variable as opposed to a large global array.
- I: The symbol is an indirect reference to another symbol. This is a GNU extension to the a.out object file format which is rarely used.
- N: The symbol is a debugging symbol.
- R: The symbol is in a read only data section.
- S: The symbol is in an uninitialized data section for small objects.
- T: The symbol is in the text (code) section.
- U: The symbol is undefined.
- V: The symbol is a weak object. When a weak defined symbol is linked with a normal defined symbol, the normal defined symbol is used with no error. When a weak undefined symbol is linked and the symbol is not defined, the value of the weak symbol becomes zero with no error.
- W: The symbol is a weak symbol that has not been specifically tagged as a weak object symbol. When a weak defined symbol is linked with a normal defined symbol, the normal defined symbol is used with no error. When a weak undefined symbol is linked and the symbol is not defined, the value of the symbol is determined in a system-specific manner without error. On some systems, uppercase indicates that a default value has been specified.
- ‗-‘: The symbol is a stabs symbol in an a.out object file. In this case, the next values printed are the stabs other field, the stabs desc field, and the stab type. Stabs symbols are used to hold debugging information. For more information, see section `Stabs Overview' in *The "stabs" debug format*.
- ? : The symbol type is unknown, or object file format specific.

ranlib

ranlib [`-vV'] *archive*
ranlib generates an index to the contents of an archive and stores it in the archive. The index lists each symbol defined by a member of an archive that is a relocatable object file.
You may use ‗nm -s‘ or ‗nm --print-armap‘ to list this index.
An archive with such an index speeds up linking to the library and allows routines in the library to call each other without regard to their placement in the archive.
The GNU ranlib program is another form of GNU ar; running ranlib is completely equivalent to executing ‗ar -s‘.

`-v`

`-V`

`--version`

Show the version number of ranlib.

size

size [`-A'|`-B'|`--format='compatibility]
   [`--help']
      [`-d'|`-o'|`-x'|`--
      radix='number] [`-t'|`--totals']
      [`--target='bfdname] [`-V'|`--
      version'] [objfile...]

The GNU size utility lists the section sizes--and the total size--for each of the object or archive files *objfile* in its argument list. By default, one line of output is generated for each object file or each module in an archive.

objfile... are the object files to be examined. If nothing is specified, the file a.out will be used.

strings

strings [`-afov'] [`-'min-len]
        [`-n' min-len] [`--bytes='min-len]
        [`-t' radix] [`--radix='radix]
        [`-e' encoding] [`--encoding='encoding]
        [`-'] [`--all'] [`--print-file-name']
        [`-T' bfdname] [`--target='bfdname]
        [`--help'] [`--version'] file...

For each *file* given, GNU strings prints the printable character sequences that are at least 4 characters long (or the number given with the options below) and are followed by an unprintable character. By default, it only prints the strings from the initialized and loaded sections of object files; for other types of files, it prints the strings from the whole file.

strings is mainly useful for determining the contents of non-text files.

strip

```
strip [`-F' bfdname |`--target='bfdname] [`-
    I' bfdname |`--input-target='bfdname]
    [`-O' bfdname |`--output-target='bfdname]
    [`-s'|`--strip-all'] [`-S'|`-g'|`-d'|`--strip-
    debug']
    [`-K' symbolname |`--keep-
    symbol='symbolname] [`-N' symbolname |`--
    strip-symbol='symbolname] [`-w'|`--wildcard']
    [`-x'|`--discard-all'] [`-X' |`--discard-locals']
    [`-R' sectionname |`--remove-section='sectionname]
    [`-o' file] [`-p'|`--preserve-dates'] [`--keep-file-
    symbols']
    [`--only-keep-debug']
    [`-v' |`--verbose'] [`-V'|`--version']
    [`--help'] [`--info']
    objfile...
```

GNU strip discards all symbols from object files *objfile*. The list of object files may include archives. At least one object file must be given.

strip modifies the files named in its argument, rather than writing modified copies under different names.

c++filt

```
c++filt [`-_'|`--strip-underscores']
      [`-n'|`--no-strip-
      underscores'] [`-p'|`--no-
      params'] [`-t'|`--types'] [`-i'|`--
      no-verbose']
      [`-s' format|`--format='format] [`-
      -help'] [`--version'] [symbol...]
```

The C++ and Java languages provide function overloading, which means that you can write many functions with the same name, providing that each function accepts parameters of different types. In order to be able to distinguish these similarly named functions, C++ and Java encode them into a low-level assembler name that uniquely identifies each different version. This process is known as mangling. The c++filt program does the inverse mapping: it decodes (demangles) low-level names into user-level names so that they can be read.

Every alphanumeric word (consisting of letters, digits, underscores, dollars, or periods) seen in the input is a potential mangled name. If the name decodes into a C++ name, the C++ name replaces the low-level name in the output, otherwise the original word is output. In this way you can pass an entire assembler source file, containing mangled names, through c++filt and see the same source file containing de-mangled names.

You can also use c++filt to decipher individual symbols by passing them on the command line:

```
c++filt symbol
```

If no symbol arguments are given, c++filt reads symbol names from the standard input instead. All the results are printed on the standard output. The difference between reading names from the command line versus reading names from the standard input is the command line arguments are expected to be just mangled names and no checking is performed to separate them from surrounding text. For example:

```
c++filt -n _Z1fv
```

will work and demangle the name to "f()" whereas:

```
c++filt -n _Z1fv,
```

will not work. (Note the extra comma at the end of the mangled name which makes it invalid). This command however will work:

```
echo _Z1fv, | c++filt -n
```

and will display "f()," ie the demangled name followed by a trailing comma. This behavior is because when the names are read from the standard input it is expected that they might be part of an assembler source file where there might be extra, extraneous characters trailing after a mangled name. eg:

```
.type   _Z1fv, @function
```

addr2line

addr2line [`-b' bfdname|`--target='bfdname]
        [`-C'|`--demangle'[=style]]
        [`-e' filename|`--exe='filename] [`-
        f'|`--functions'] [`-s'|`--basename']
        [`-i'|`--inlines'] [`-j'|`--
        section='name]
        [`-H'|`--help'] [`-V'|`--
        version'] [addr addr ...]

The addr2line translates addresses into file names and line numbers. Given an address in an executable or an offset in a section of a relocatable object, it uses the debugging information to figure out which file name and line number are associated with it.

The executable or relocatable object to use is specified with the `-e' option. The default is the file `a.out'. The section in the relocatable object to use is specified with the `-j' option.

addr2line has two modes of operation.

In the first, hexadecimal addresses are specified on the command line, and addr2line displays the file name and line number for each address.

In the second, addr2line reads hexadecimal addresses from standard input, and prints the file name and line number for each address on standard output. In this mode, addr2line may be used in a pipe to convert dynamically chosen addresses.

The format of the output is `FILENAME:LINENO'. The file name and line number for each address is printed on a separate line. If the -f option is used, then each `FILENAME:LINENO' line is preceded by a `FUNCTIONNAME' line which is the name of the function containing the address.

If the file name or function name can not be determined, addr2line will print two question marks in their place. If the line number can not be determined, addr2line will print 0.

convrenesaslib

convrenesaslib [`--remove-underscore']
 [`--help']
 [`--verbose'] [`--
 version'] renesas-
 library-file(s)...

The GNU convrenesaslib program converts the input Renesas SH library file(s) into a GNU archive. The converted GNU archive has *'.rlib'* extension. This utility supports only Renesas SH library files. Multiple Renesas library files can be provided on command line at a time.

This utility extracts the object files from the input Renesas library and appends the filenames with the extension *'.robj '*. The 'ar' utility is then invoked to create a GNU archive of the extracted object files with the extension *'.robj'*. The GNU archive created will have extension *'.rlib'* appended to the name. These new extensions are necessary so as to distinguish from other files with same names. The extracted object files are deleted after the creation of the GNU archive.

renesas-library-file(s)...are the Renesas library files to be converted to GNU archive files.

The options that can be used with convrenesaslib are listed below. The long and short forms of options, shown here as alternatives, are equivalent.

`--remove-underscore`

This will remove all leading underscores appended to global symbols in the created GNU archive.

`--help`

Show a summary of the options to convrenesaslib and exit.

`--verbose`

Verbosely list the files processed.

`--version`

Show the version number of convrenesaslib and exit.

For more details on the above mentioned Binary Utilities please refer the Binary

2.4.7 Object Translation (ELF to Binary, SREC, etc.)

objcopy [*options*] *infile* [*outfile*]

Copy the contents of the input object file to another file, optionally changing the file format in the process (but not the endian-ness). If `outfile` is not specified, **objcopy** creates a temporary file and renames it to `infile` when the copy is complete, destroying the original input file. The GNU Binary File Descriptor (BFD) library is used to read and write the object files.

`--add-section section=file`

Add a new section to the output object file with the specified section name and the contents taken from the specified file. Available only for formats that allow arbitrarily named sections.

`--alt-machine-code=n`

If the output architecture has alternate machine codes, use the *n*th code instead of the default.

`-b n, --byte=n`

Copy only every *n*th byte. Header data is not affected. The value of *n* can be from 0 to *interleave*-1, where *interleave* is specified by -i (default is 4). This option is useful for creating files to program ROM and is typically used with srec as the output format.

`-B bfdarch, --binary-architecture=bfdarch`

Set the output architecture to *bfdarch* (e.g., i386) for transforming a raw binary file into an object file. Otherwise, this option is ignored. After the conversion, your program can access data inside the created object file by referencing the special symbols _binary_objfile_start, _binary_objfile_end, and _binary_objfile_size.

`--change-addresses=incr, --adjust-vma=incr`

Change the VMA and LMA addresses of all sections, plus the start address, by adding *incr*. Changing section addresses is not supported by all object formats. Sections are not relocated.

`--change-leading-char`

For object formats that use a special character (such as an underscore) to begin symbols, change the leading character when converting between formats. If the character is the same in both formats, the option has no effect. Otherwise, it adds, removes, or changes the leading character as appropriate for the output format.

`--change-section-address` *`section`*`{=|+|-}`*`val`*`,`

`--adjust-section-vma` *`section`*`{=|+|-}`*`val`*

Set or change the VMA and LMA addresses of the specified section. With =, set the section address to the specified value; otherwise, add or subtract the value to get the new address.

`--change-section-lma` *`section`*`{=|+|-}`*`val`*

Set or change the LMA address of the specified section. With =, set the section address to the specified value; otherwise, add or subtract the value to get the new address.

`--change-section-vma` *`section`*`{=|+|-}`*`val`*

Set or change the VMA address of the specified section. With =, set the section address to the specified value; otherwise, add or subtract the value to get the new address.

`--change-start` *`incr`*`, --adjust-start` *`incr`*

Add *incr* to the start address to get a new start address. Not supported by all object formats.

`--change-warnings, --adjust-warnings`

Issue a warning if the section that is specified in one of the options --change-section-address, --change-section-lma, or --change-section-vma does not exist.

`--debugging`

Convert debugging information if possible.

`-F bfdname, --target=bfdname`

Set the binary format for both input and output files to the binary file descriptor name *bfdname*. No format translation is done. Use the -h option for a list of supported formats for your system.

`-g, --strip-debug`

Do not copy debugging information.

`-G` *`symbol`*`, --keep-global-symbol=`*`symbol`*

Copy only the specified global symbol, making all other symbols local to the file. May be specified multiple times.

`--gap-fill=`*`val`*

Fill gaps between sections with the specified value; applies to the load address (LMA) of the sections.

`-h, --help`

Print help information, including a list of supported target object formats, then exit.

`-i interleave, --interleave=interleave`

Copy one out of every *interleave* bytes. Use -b to set the byte to copy (default is 4). This option is ignored if -b is not specified.

`-I bfdname, --input-target=bfdname`

Set the binary file format of the input file using its binary file descriptor name, *bfdname*.

`-j section, --only-section=section`

Copy only the specified section. May be specified multiple times.

`-K symbol, --keep-symbol=symbol`

Copy only the specified symbol from the source file. May be specified multiple times.

`--keep-global-symbols=filename`

Apply the option --keep-global-symbol to each symbol listed in the specified file. The file should have one symbol per line, with comments beginning with a hash mark (#). May be specified multiple times.

`--keep-symbols=file`

`Apply the option --keep-symbol to each symbol listed in the specified file. The file should have one symbol per line, with comments beginning with a hash mark (#). May be specified multiple times.`

`-L symbol, --localize-symbol=symbol`

Make the specified symbol local. May be specified multiple times.

`--localize-symbols=filename`

Apply the option --localize-symbol to each symbol listed in the specified file. The file should have one symbol per line, with comments beginning with a hash mark (#). May be specified multiple times.

`-N symbol, --strip-symbol=symbol`

Do not copy the specified symbol. May be specified multiple times.

`--no-change-warnings, --no-adjust-warnings`

Do not issue a warning even if the section specified in one of the options --change-section-address, --change-section-lma, or --change-section-vma does not exist.

`-O bfdname, --output-target=bfdname`

Set the binary file format of the output file using its binary file descriptor name, *bfdname*. The format srec generates S -records (printable ASCII versions of object files), and binary generates a raw binary file. Use -h for other available formats.

`-p, --preserve-dates`

Preserve the input file's access and modification dates in the output file.

`--pad-to=`*`addr`*

Pad the output file up to the load address. Use the fill value specified by --gap-fill (default is 0).

`-R `*`section`*`, --remove-section=`*`section`*

Do not copy any section with the specified name. May be specified multiple times.

`--redefine-sym `*`old=new`*

Change the name of the symbol *old* to *new*.

`--remove-leading-char`

If the first character of a global symbol is a special character (such as an underscore) used by the input object file format, remove it. Unlike --change-leading-char, this option always changes the symbol name when appropriate, regardless of the output object format.

`--rename-section `*`oldname=newname`*`[,`*`flags`*`]`

Rename a section from *oldname* to *newname*, optionally also changing the flags to *flags*.

`-S, --strip-all`

Do not copy relocation and symbol information.

`--set-section-flags `*`section=flags`*

`Set flags for the specified section as a comma-separated string of flag names. Not all flags are meaningful for all object formats. The possible flags are alloc, code, contents, data, debug, load, noload, readonly, rom, and share.`

`--set-start=`*`val`*

Set the start address of the new file to the specified value. Not supported by all object formats.

`--srec-forceS3`

Force all srec output records to be type S3 records.

`--srec-len=ival`

Set the maximum length of srec output records to the specified value. The length includes the address, data, and crc fields.

`--strip-symbols=`*`filename`*

Apply the option --strip-symbol to each symbol listed in the specified file. The file should have one symbol per line, with comments beginning with a hash mark (#). May be specified multiple times.

`--strip-unneeded`

Strip all symbols not needed for relocation processing.

`-v, --verbose`

Run in verbose mode, listing all object files modified; for archives, list all archive members.

`-V, --version`

Print version information and exit.

-W *symbol*, --weaken-symbol=*symbol*

Make the specified symbol weak. May be specified multiple times.

`--weaken`

Make all global symbols weak.

`--weaken-symbols=`*`filename`*

Apply the option --weaken-symbol to each symbol listed in the specified file. The file should have one symbol per line, with comments beginning with a hash mark (#). May be specified multiple times.

`-x, --discard-all`

Do not copy non-global symbols.

`-X, --discard-locals`

Do not copy compiler-generated local symbols (usually those starting with L)

2.4.8 GNUSH Libraries

Following sections explain about the Newlib and optimized libraries that are provided along with GNU Tools.

2.4.8.1 Newlib

Technically, the name newlib refers to a collection of source code assembled by Cygnus Solutions (now Red Hat) from several different origins. This point is interesting only for heritage and licensing purposes, because the code itself is released, managed, and used as a single functional unit, just as you would expect from any collection of C code distributed under a single name and version number.

Newlib is one of several options for small, GNU-friendly C runtime environments. It is unique, however, in that it is a mature and well-supported product with an active developer and user base. And its architecture addresses the needs of deeply embedded systems quite well.

Features

Some of the features of the library are useful enhancements to a "typical" embedded setting (whatever that is), while others allow newlib to be about as POSIX-like as a compact C runtime setting can be. A POSIX-like API can be a big help when porting applications to embedded hardware.

printf vs. iprintf

Newlib contains a complete implementation of the standard **printf()** and family. By the implementation's own admission, "this code is large and complicated" (**vfprintf.c:144**), but it is essential for systems that need full ANSI C input and output support, including capabilities for representing and parsing floating-point numbers.

Many embedded systems do not use floating-point math, however, and great pains are taken in most embedded runtime libraries to cull this code-bloating functionality whenever possible. Newlib actually approaches this problem in two ways:

- A **FLOATING_POINT** macro that allows you to selectively disable floating-point support in each of the stdio library functions that can offer it.
- A new **iprintf()** function that only knows how to display integer objects

If an embedded system needs floating-point support in only a few of the standard input and output functions, newlib can be rebuilt to exclude floating point from places where it is not needed. User can omit floating point for everything except **scanf()**, for example, by either un-defining the **FLOATING_POINT** macro everywhere except in the **scanf** source file, or by modifying newlib's build process to do the same thing, in a more automated fashion.

For situations where only integer output is required, newlib provides the iprintf() function: a version of printf built without FLOATING_POINT enabled. It behaves exactly like printf, except that it does not understand the %f, %F, %g, or %G type specifiers, and therefore has a much smaller code footprint than its full-featured big brother.

More on stdio

Newlib's standard input and output facilities are quite complete. The complete C file API is also provided, complete with read and write buffering, seeking, and stream flushing capabilities. Variations like **sprintf, fprintf**, and **vfprintf** (takes **va_list** arguments) are also included, which makes a newlib environment look strikingly similar to one user would expect to see in a more workstation-oriented programming environment.

An unfortunate limitation of newlib's stdio library is that it requires at least a minimal **malloc()** for proper operation. When the printf() machinery detects a floating-point format specifier, it allocates a few bytes of memory to use as a workspace for constructing the text representation of the number. It's actually a poor design decision, because in other places, newlib statically allocates memory to accomplish the same thing. But that's the way it is. On the plus side, newlib includes a good dynamic memory allocator that is straightforward to set up and use.

Unix API

Newlib also includes a lot of the familiar Unix API functions like **open()** and **write()**. Most of them map directly to the code stubs newlib uses to invoke external system resources, so their simplicity eliminates the need for **malloc()**. These stubs will be discussed later, in the section on porting.

Libm

Newlib contains a complete IEEE math library called libm. In addition to offering the standard math functions like **exp(), sin()**, and **pow()**, this code also provides **matherr()**, a modifiable math error handler that the library invokes whenever a serious math-related error, such as an underflow or loss of precision, is detected. By customizing this function, you can handle these situations in whatever way is appropriate for your system.

As a surprising bonus, libm also includes functions that take **float** parameters, instead of **double**. These extensions are named after their full -precision equivalents. For example, **sinf()** is the single-precision version of the **sin()** function. The reduced-precision functions have a considerable speed advantage over their IEEE-compliant double-precision counterparts, which can put some floating-point operations within reach of hardware that is too weak for full double-precision computations.

Reentrancy

Newlib's C and math libraries are reentrant when properly integrated into a multithreaded environment. The implementation is not obvious at first glance, so following few paragraphs describes how it works. Once you know the details, it will be clear how to make sure you set it up properly in your system.

The ANSI C standard specifies a global integer called **errno** that the runtime library sets when an error occurs. Once set, **errno**'s value persists until the application clears it, which simplifies error notification by the library but can create reentrancy problems when multiple execution threads are working in the library at the same time-neither thread knows who generated the error.

Newlib addresses this problem by redefining **errno** as a macro that references a global pointer to a structure of type **struct _reent** (the pointer is called **_impure_ptr**). (Actually, the **errno** macro calls the function **__errno()**, which in turn references **__impure_ptr[errno.c:11]**. But you get the idea.) This structure contains the traditional errno value specified by ANSI, but it also contains a lot of other stuff, including signal handler pointers and file handles for standard input, output, and error streams.

Newlib's **errno** macro hides all of this from the user, of course. To check for errors, an application only needs to refer to the value of **errno**, just like it would in any ANSI environment. To determine why an **fopen()** failed, for example, you still do this:

```
fp = fopen( "myfile.txt", "rw" );
if (fp == NULL) {
switch( errno ) {
case EACCESS:
/* we don't have permissions */
...
}
```

just like ANSI C says to. The only difference is that the reference to errno is a macro that returns **_impure_ptr->errno**, instead of the value of a global integer.

Newlib declares **one _reent** structure and **aims _impure_ptr** at it during initialization, so everything starts out correctly for situations where only one thread of execution is in the library at a time. To facilitate multiple contexts, you must take two additional steps: you must provide one _reent structure for each execution thread, and you must move **_impure_ptr** between these structures during context switches.

If you also want a reentrant malloc(), you must provide the functions **__malloc_lock()** and **__malloc_unlock()** to protect your memory pool from corruption during allocations. This step usually is not necessary if user memory pool is built from memory management components supplied by an RTOS, as the functions there are often reentrant already, and need no additional protection. Hooks called **__env_lock** and **__env_unlock** are also provided, so that user can implement a reentrant environment variable pool as well.

Designed for portability

All of newlib's functionality builds on a set of 17 stub functions that newlib uses to hook into the execution environment. By replacing these stubs, user can integrate newlib with just about any system imaginable, from one with no operating system at all to one based on an RTOS to one with a complete POSIX OS like Linux.

Newlib's documentation provides details on which stubs are needed for each library function, so user only need to provide stubs for the portions of newlib that user intend to use. For example, newlib has a stub called **_fork**, but user does not need to do anything with it unless users application will call **system()** or **fork()**.

Newlib does not include a filesystem, though it may seem like it requires one for proper operation, particularly when user considers that it provides file-oriented functions like fprintf() and fseek().

Building newlib

Building newlib for a supported target is a straightforward process that follows the conventions adhered to by most open source projects. After user downloads and decompresses the source code, he simply configures and build it using a previously constructed cross compiler like gcc. In other words:

```
tar xzvf newlib-1.14.0.tar.gz
mkdir build-newlib
cd build-newlib
../newlib-1.14.0/configure
target=#&60;target-name>
make all install info install-
info
```

The build process will have produced the files **libc.a**, **libg.a** (a debugging-enabled libc), and libm.a, in the directory **/usr/local/#target-name>**. If the target user specified has several variants, the build process will produce multiple files, each with optimizations specific to that variant. Link one or more of these files with your application, and there you have it-a free C runtime environment.

Newlib does not require use of the GNU compiler, but if user uses something else, he needs to make adjustments to newlib's makefiles after the configuration step.

Newlib's build process also produces documentation, in the files libc.info and libm.info. By default these files go into /usr/local/info, and they can be browsed using info, a documentation browser included with most Linux distributions. Just change to the directory containing these files, and type:

¯info -f ./libc.info‖

Newlib's configuration utility supports several options, not the least of which are the definitions of the target system (what CPU and OS the library will run under) and where to put the files generated during the build. As an example, the following command will set up a build for the Hitachi SH CPU using the elf output file format, and put the resulting libraries in **/home/bgat/newlib-install**. Since user does not specify an operating system, the build scripts will assume that the target system does not have one, with the idea that user will provide code stubs with the application.

```
../newlib-1.14.0/configure -
target=sh-elf -prefix=/home/
bgat/newlib-install
```

Use the **-help** option to see a complete list of command line options.

Since the library is implemented almost entirely in C, newlib can also be used on unsupported microprocessors simply by replacing its startup code and assembly language implementations of **memcpy()** and **setjmp()/longjmp()**, and by obtaining a cross compiler for the target system.

Newlib can be built in both Unix-like environments, and under Cygwin, Red Hat's freely available POSIX-compatibility environment that makes Win32 hosts more UNIX-like. (user does not need to modify the previous example-Cygwin's shell makes the Win32 directory structure look enough like UNIX that newlib will build without problems.

Tweaks

Newlib's source code has a few configuration points, and user will want to use them to eliminate unneeded stubs, to optimize for code size instead of speed, or to remove floating-point support. User will not want to spend much time with this in the early stages of a project, before user has enough of a system in place to evaluate the benefits of his changes.

The best way to go about tweaking newlib is to change the values in the Makefile generated by the configure command. User should check for the variable **CFLAGS_FOR_TARGET**, and add flags there like:

-DINTEGER_ONLY

to build an integer-only library, or:

-DPREFER_SIZE_OVER_SPEED

to enable a few small changes that reduce library code size.

User can also adjust the value of the CFLAGS variable, to affect the way the library is compiled. For example, if user is using the GNU C compiler then he can use:

-Os (instead of -O2)

to tell it to optimize for code size over performance, or:

-O3 (instead of -O2)

to tell the compiler to optimize for raw performance over everything else.

-fomit-frame-pointer

to tell the compiler to not build stack frames for functions in the library that don't need them, which saves some space and boosts performance slightly. If user intends to step through code inside of newlib itself, he should not eliminate stack frames as it likely will not work-like most debuggers, the GNU debugger needs a valid stack frame at all times.

make clean all install

Porting newlib

The newlib C standard library supports more than a dozen target CPU architectures, but it does not come with stubs to connect it to very many operating systems. As a result, the odds are almost certain that user will need to do some porting work to get newlib running in his system. Fortunately, the process is both straightforward and relatively painless.

As mentioned, all of newlib's functionality builds on a set of 17 stubs that supply capabilities newlib cannot provide itself-low-level filesystem access, requests to enlarge its memory heap, getting the time of day, and various types of context management like process forking and killing. These stubs are fully documented in newlib's libc.info file, in the section called "Syscalls." The key to a successfully ported newlib is finding the functionality in system to hook these stubs to.

2.4.8.2 Optimized Libraries

GNU Tools provide Optimized Libraries along with the GNUSH Toolchain for all the SH targets. When linked with an application, the optimized libraries produce approximately 30% less code as compared to the newlib libraries.

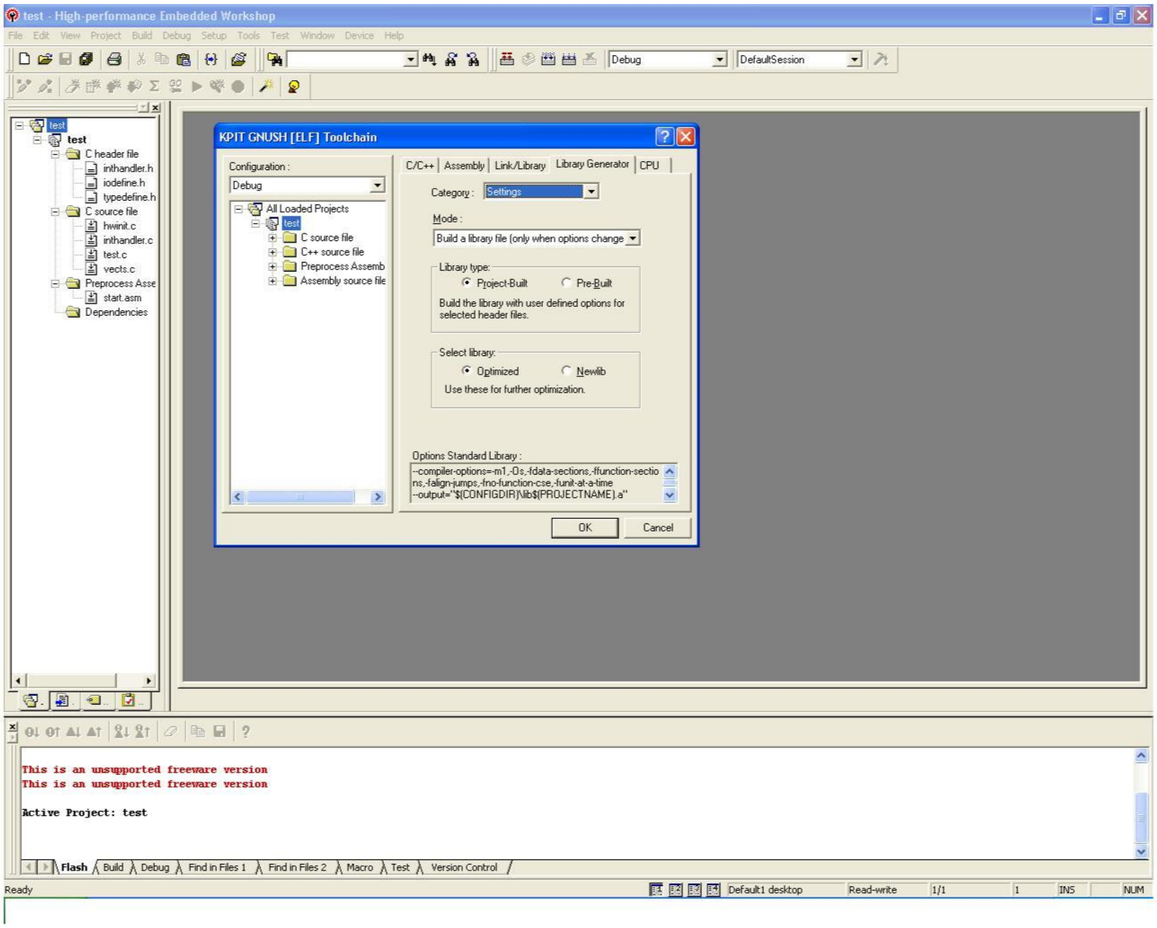
To use these Optimized Libraries, user needs to pass ‗-loptc‘ and ‗-loptm‘ while compiling any application. Also, user needs to pass the options ‗-nostbinc‘ (to restrain the Optimized Libraries from using the default headers). The Optimized Libraries have their own set of header files placed at the following path, `<Toolchain Install Directory\sh-elf\sh-elf\lib\optinc>`

User needs to pass this path to application while compiling it with Optimized Libraries. Use ‗-I‘ option for the same.

For Example:

```
#sh-elf-gcc test.c -loptc -loptm -lgcc -nostdinc
 –I ―<Toolchain Install Directory\sh-elf\sh-
elf\lib\optinc> ‖
```

In HEW, select the option "Settings" from ―Category‖ drop down box under "Library Generator"-tab to use optimized libraries.



Configuration : Debug

Category : Settings

Mode : Build a library file (only when options change

Library type:
● Project-Built ○ Pre-Built
Build the library with user defined options for selected header files.

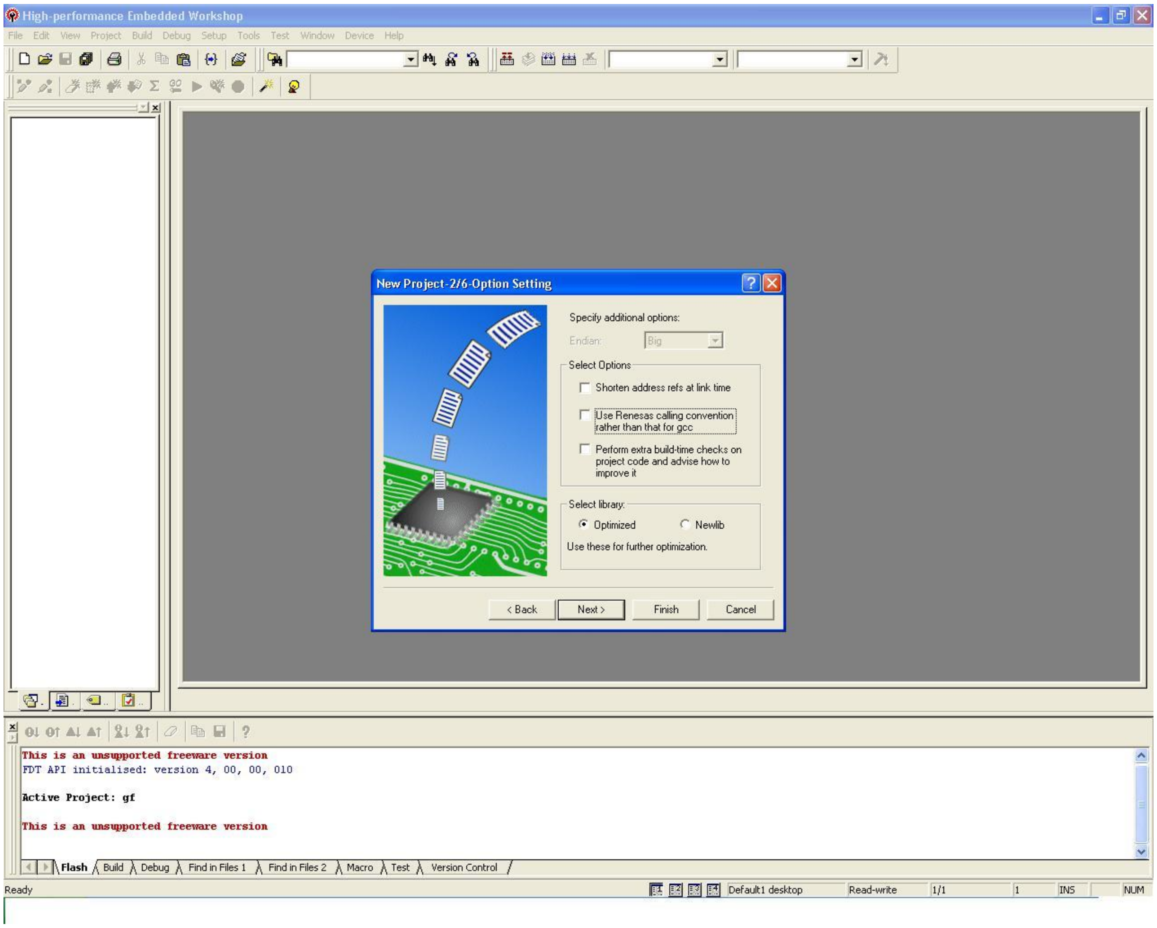
Select library:
● Optimized ○ Newlib
Use these for further optimization.

Options Standard Library :

```
--compiler-options=-m1,-Os,-fdata-sections,-ffunction-sectio
ns,-falign-jumps,-fno-function-cse,-funit-at-a-time
--output="$(CONFIGDIR)\lib$(PROJECTNAME).a"
```

```
This is an unsupported freeware version
This is an unsupported freeware version

Active Project: test
```

In HEW the Optimized Libraries can also be selected while creating the project as shown below,


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Device Help
New Project-2/6-Option Setting
< Back
Next >
Finish
Cancel
Flash
Build
Debug
Find in Files 1
Find in Files 2
Macro
Test
Version Control
Ready
Default1 desktop
Read-write
1/1
1
INS
NUM

Specify additional options:

Endian: Big

Select Options

□ Shorten address refs at link time

□ Use Renesas calling convention rather than that for gcc

□ Perform extra build-time checks on project code and advise how to improve it

Select library:

● Optimized ○ Newlib

Use these for further optimization.

```
This is an unsupported freeware version
FDT API initialised: version 4, 00, 00, 010

Active Project: gf

This is an unsupported freeware version
```

2.5 Using Inline Assembly Language in GNUSH

We can instruct the compiler to insert the code of a function into the code of its callers, to the point where actually the call is to be made. Such functions are inline functions.

This method of inlining reduces the function-call overhead. And if any of the actual argument values are constant, their known values may permit simplifications at compile time so that not all of the inline function‘s code needs to be included. The effect on code size is less predictable, it depends on the particular case. To declare an inline function, we‘ve to use the keyword inline in its declaration. Inline assembly is important primarily because of its ability to operate and make its output visible on C variables.

Because of this capability, "asm" works as an interface between the
assembly instructions and the "C" program that contains it.

2.5.1 Syntax

GCC, the GNU C Compiler for Linux, uses **AT&T**&sol;**UNIX** assembly syntax.

- Source-Destination Ordering

  The first operand is the source and the second operand is
  the destination. ie, "Op-code src dst".

- Register Naming

  Register names are prefixed by % ie, if r0 is to be used, write %r0.

- Immediate Operand

  Immediate operands are preceded by ‘#‘. For hexadecimal constants a ‘0x‘ is suffixed to the constant. So, for hexadecimals, we first see a ‘#‘, then ‘0x‘ and finally the constants.

- Operand Size

  Size of memory operands is determined from the last character of the op-code name. Op-code suffixes of ‘b‘, ‘w‘, and ‘l‘ specify byte(8-bit), word(16-bit), and long (32-bit) memory references.

- Memory Operands

  Base register is enclosed in ‗(‘ and ‘)‘ and indirect memory reference is like ‾section&colon;disp(base, index, scale)‖. When a constant is used for disp and sol; scale, ‘#‘ shouldn‘t be prefixed.

2.5.2 Basic Inline Assembly

The format of basic inline assembly is very much straight forward. Its basic form is;

```
asm("assembly code");
```

For Example:

```
asm("nop");
asm("mov.l r0,r1"); /* moves the contents of r0 to r1
*/ __asm__("mov.b r0,@r1");
/* contents of r0 to at memory address stored in r1 */
```

We can use __asm__ if the keyword asm conflicts with something in our program. If we have more than one instructions, we write one per line in double quotes, and also suffix a '\n' and '\t' to the instruction. This is because gcc sends each instruction as a string to **as** (GAS) and by using the newline&sol;tab we send correctly formatted lines to the assembler.

For Example.

```
      __asm__("mov #0x56,r0\n\t"
             "mov #0x8a,r1\n\t"
             "and r0,r1\n\t" "mov
             r1,@r14\n\t");
```

If we change the contents of some registers in our code and return from _asm' without fixing those changes, something wrong may happen. This is because GCC have no idea about the changes in the register contents and this leads us to trouble, especially when compiler makes some optimizations. It will suppose that some register contains the value of some variable that we might have changed without informing GCC, and it continues like nothing happened. What we can do is either use those instructions having no side effects or fix things when we quit. This is where we want extended inline assembly to get some extended functionality.

2.5.3 Extended Inline Assembly

In basic inline assembly, we had only instructions. In extended assembly, we can also specify the operands. It allows us to specify the input registers, output registers and a list of clobbered registers. It is not mandatory to specify the registers to use, we can leave that to GCC and that probably fit into GCC's optimization scheme better.

Here is the basic format:

```
asm ( assembler template
      : output operands                 /* optional */
      : input operands                  /* optional */
      : list of clobbered registers     /* optional */
    );
```

The assembler template consists of assembly instructions. Each operand is described by an operand-constraint string followed by the C expression in parentheses. A colon separates the assembler template from the first output operand and another separates the last output operand from the first input, if any. Commas separate the operands within each group. The total number of operands is limited to ten or to the maximum number of operands in any instruction pattern in the machine description, whichever is greater.

If there are no output operands but there are input operands, user must place two consecutive colons surrounding the place where the output operands would go.

```
__inline__ void set_gbr(void *gbr_reg)
{
      asm("mov %0, r2"
         : /*no output operands*/
         :"r"(gbr_reg)
         :"r2"); asm("ldc
      r2, gbr");
}
int a=10, b;
  asm ("mov.l\t%1, r1\n\t"
       "mov.l\tr1, %0"
       :"=r"(b)         /* output */
       :"r"(a)          /* input */
       :"r1"            /* clobbered register */
       );
```

The detailed description of each field in the basic assembly format is explained in next sections.

2.5.3.1 Assembler Template

The assembler template contains the set of assembly instructions that gets inserted inside the C program. The format is like either each instruction should be enclosed within double quotes, or the entire group of instructions should be within double quotes. Each instruction should also end with a delimiter. The valid delimiters are newline (\n) and semicolon (&semi;). '\n' may be followed by a tab (\t). Operands corresponding to the C expressions are represented by %0, %1 ... etc.

2.5.3.2 Operands

C expressions serve as operands for the assembly instructions inside "asm". Each operand is written as an operand constraint in double quotes. For output operands, there'll be a constraint modifier also within the quotes and then follows the C expression which stands for the operand. For example ‗=' can be used to specify the operand as write-only. Constraints are primarily used to decide the addressing modes for operands. They are also used in specifying the registers to be used. If we use more than one operand, they are separated by comma.

In the assembler template, each operand is referenced by numbers. If there are a total of n operands (both input and output inclusive), then the first output operand is numbered 0, continuing in increasing order, and the last input operand is numbered n-1. Output operand expressions must be lvalues (results, LHS values). The input operands are not restricted like this. They may be expressions. Ordinary output operands must be write -only; GCC will assume that the values in these operands before the instruction are dead and need not be generated. Extended asm also supports input-output or read-write operands.

2.5.3.3 Clobber List

Some instructions clobber some hardware registers. We have to list those registers in the clobber-list, ie the field after the third ':' in the asm function. This is to inform gcc that we will use and modify them ourselves. So gcc will not assume that the values it loads into these registers will be valid. We shouldn't list the input and output registers in this list. Because, gcc knows that "asm" uses them (because they are specified explicitly as constraints). If the instructions use any other registers, implicitly or explicitly (and the registers are not present either in input or in the output constraint list), then those registers have to be specified in the clobbered list.

If our instruction can alter the condition code register, we have to add "cc" to the list of clobbered registers.

If our instruction modifies memory in an unpredictable fashion, add "memory" to the list of clobbered registers. This will cause GCC to not keep memory values cached in registers across the assembler instruction. We also have to add the **volatile** keyword if the memory affected is not listed in the inputs or outputs of the asm.

We can read and write the clobbered registers as many times as we like.

2.5.3.4 Keyword `volatile`

If our assembly statement must execute where we put it, (i.e. must not be moved out of a loop as an optimization), put the keyword volatile after asm and before the ()'s. So to keep it from moving, deleting and all, we declare it as

```
asm volatile ( ... : ... : ... : ...);
```

Use `__volatile__` when we have to be very much careful.

2.5.3.5 Constraints

Constraints can say whether an operand may be in a register, and which kinds of register; whether the operand can be a memory reference, and which kinds of address; whether the operand may be an immediate constant, and which possible values (ie range of values) it may have.... etc.

Commonly used constraints;

- Register operand constraint(r)

- When operands are specified using this constraint, they get stored in General Purpose Registers(GPR).

  ```
  Ex:- asm ("movl   %%r1, %0\n" :"=r"(myval));
  ```

  ‗myval‘ is kept in a register, the value in register r1 is copied onto that register, and the value of myval is updated into the memory from this register. When the "r" constraint is specified, gcc may keep the variable in any of the available GPRs. To specify the register, user must directly specify the register names by using specific register constraints.

- Memory operand constraint (m)

  When the operands are in the memory, any operations performed on them will occur directly in the memory location, as opposed to register constraints, which first store the value in a register to be modified and then write it back to the memory location.

  ```
  Ex: - asm("sidt %0\n" : :"m"(loc));
  ```

- Matching(Digit) constraints
  In some cases, a single variable may serve as both the input and the output operand. Such cases may be specified in "asm" by using matching constraints.

  ```
  Ex:- asm ("incl %0" :"=a"(var):"0"(var));
  ```

  This constraint can be used:

  - In cases where input is read from a variable or the variable is modified and modification is written back to the same variable.
  - In cases where separate instances of input and output operands are not necessary.

2.5.4 Some Useful Examples

- Program to add two numbers

  ```
  int main(void)
  {
       int foo = 10, bar = 15;
        __asm__ __volatile__("add  %1,%0"
                                  :"=r"(foo) :"r"(bar),

                                  "0"(foo) );

        printf("foo+bar=%d\n", foo);
        return 0;
  }
  ```

  Here we insist GCC to store foo in %0, bar in %1 and we also want the result in %1. The ‘=‘ sign shows that it is an output register.

- Program for setting the TBR register

```
extern __inline__ void set_tbr(void *tbr)
{
        asm("mov %0,r6"::"r"(tbr):"r6");
        asm("ldc r6,tbr");
}
```

This function sets the value of the TBR register to the value specified by the user using the variable tbr.

2.6 Using GNUSH with HEW

This section is intended to guide new users of KPIT Cummins GNU tools through downloading and installing the KPIT Cummins GNU tools for Renesas micros. The tools are free, easy to use and are backed by unlimited free technical support worldwide.

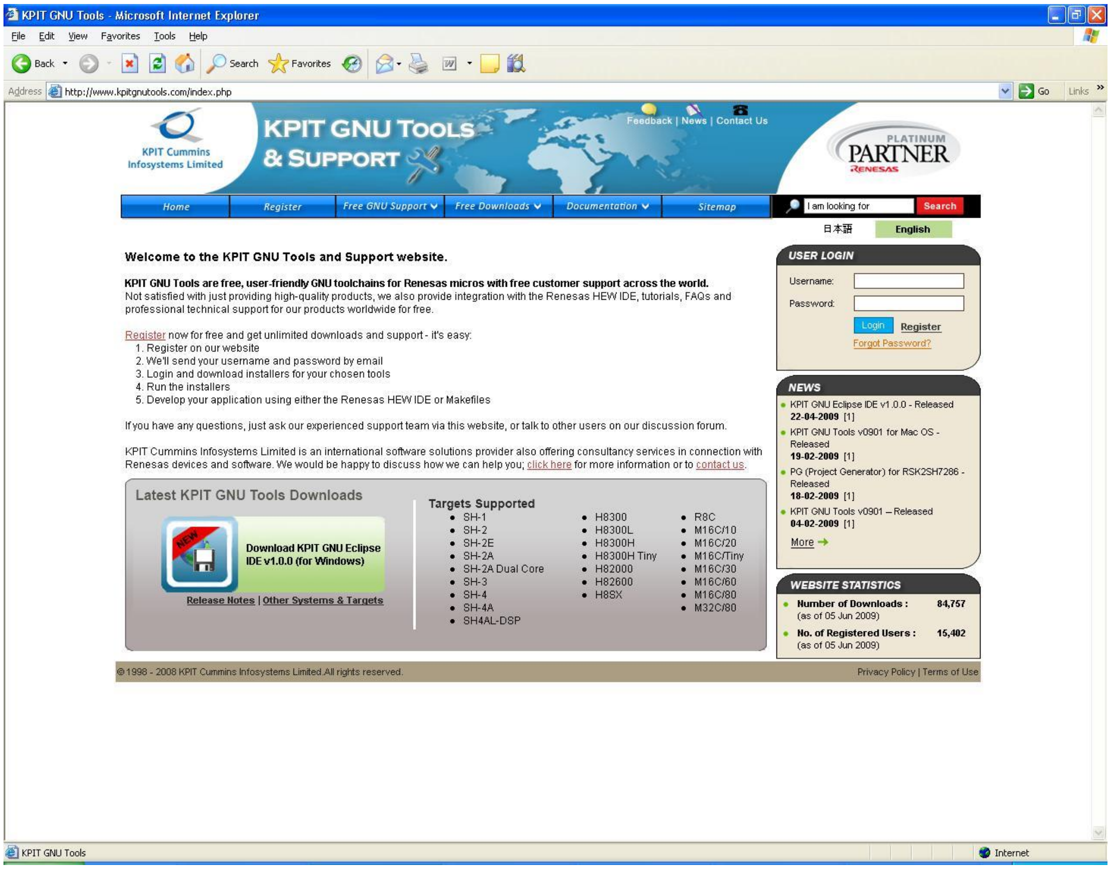
The examples are based on a user wishing to use the GNUSH [ELF] v0902 toolchain for use with the Renesas HEW IDE for Windows.

2.6.1 Downloading and Installing the Tools

Step 1

If you have not already registered for free tools and technical support, click on the "**Register**" link and fill out your details. Confirmation of your registration will be emailed to the email address you provide together with a password. KPIT has a strict privacy policy published on the website and will never sell the information you register with us nor will that information be distributed outside our working group. We need it purely to justify the work we do and to monitor demand for our services.

Having registered, enter your registered email address and password and click "**Login**".



Welcome to the KPIT GNU Tools and Support website.

KPIT GNU Tools are free, user-friendly GNU toolchains for Renesas micros with free customer support across the world.
Not satisfied with just providing high-quality products, we also provide integration with the Renesas HEW IDE, tutorials, FAQs and professional technical support for our products worldwide for free.

Register now for free and get unlimited downloads and support - it's easy:

1. Register on our website
2. We'll send your username and password by email
3. Login and download installers for your chosen tools
4. Run the installers
5. Develop your application using either the Renesas HEW IDE or Makefiles

If you have any questions, just ask our experienced support team via this website, or talk to other users on our discussion forum.

KPIT Cummins Infosystems Limited is an international software solutions provider also offering consultancy services in connection with Renesas devices and software. We would be happy to discuss how we can help you; click here for more information or to contact us.

Latest KPIT GNU Tools Downloads

NEW

Download KPIT GNU Eclipse IDE v1.0.0 (for Windows)

Release Notes | **Other Systems & Targets**

Targets Supported

- SH-1
- SH-2
- SH-2E
- SH-2A
- SH-2A Dual Core
- SH-3
- SH-4
- SH-4A
- SH4AL-DSP
- H8300
- H8300L
- H8300H
- H8300H Tiny
- H82000
- H82600
- H8SX
- R8C
- M16C/10
- M16C/20
- M16C/Tiny
- M16C/30
- M16C/60
- M16C/80
- M32C/80

USER LOGIN

Username:

Password:

Login **Register**

Forgot Password?

NEWS

- KPIT GNU Eclipse IDE v1.0.0 - Released
  22-04-2009 [1]
- KPIT GNU Tools v0901 for Mac OS - Released
  19-02-2009 [1]
- PG (Project Generator) for RSK2SH7286 - Released
  18-02-2009 [1]
- KPIT GNU Tools v0901 – Released
  04-02-2009 [1]

More →

WEBSITE STATISTICS

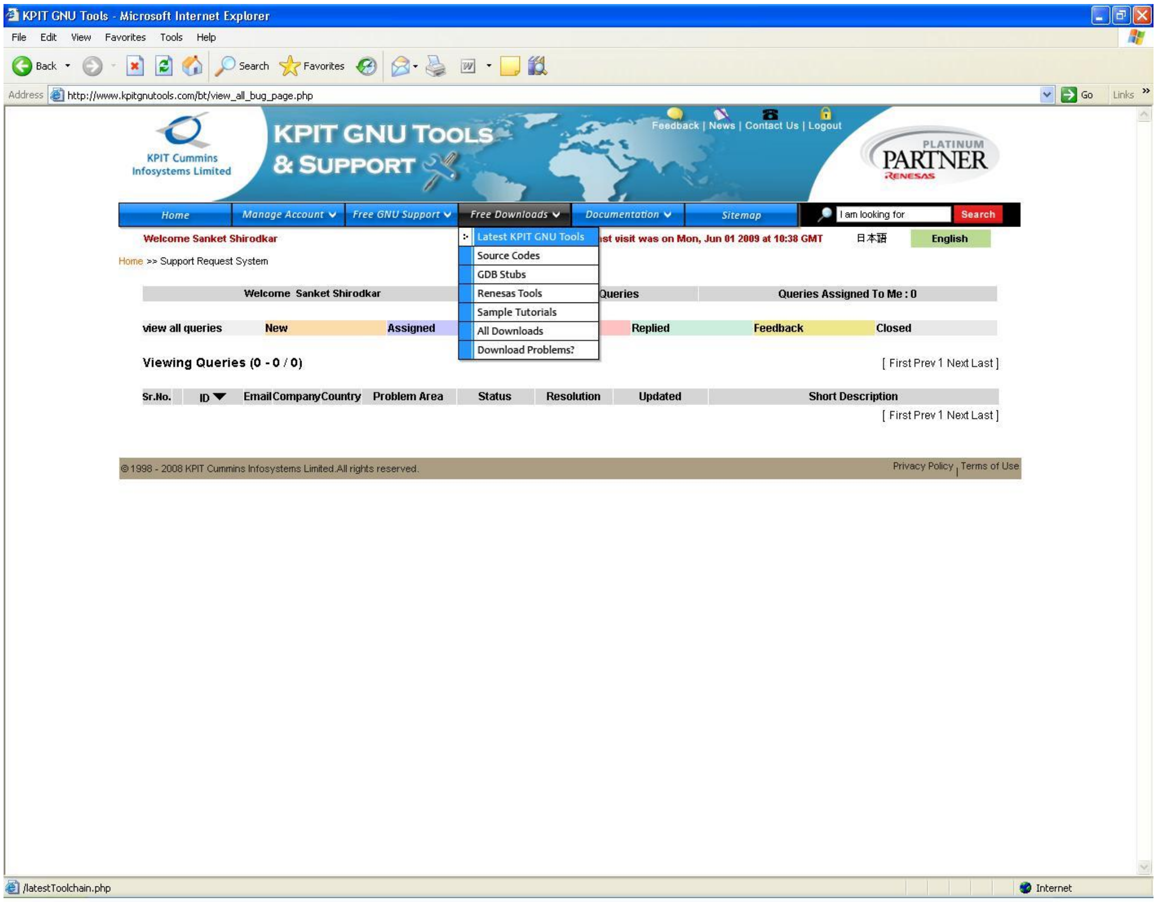
- **Number of Downloads : 84,757**
  (as of 05 Jun 2009)
- **No. of Registered Users : 15,402**
  (as of 05 Jun 2009)

© 1998 - 2008 KPIT Cummins Infosystems Limited.All rights reserved.

Privacy Policy | Terms of Use

Step 2

Now that you are logged into the support/download system, select ¯Latest GNU Tools‖ sub menu from the "Free Downloads" menu.



Welcome Sanket Shirodkar ...st visit was on Mon, Jun 01 2009 at 10:38 GMT

Home >> Support Request System

Welcome Sanket Shirodkar ... Queries Queries Assigned To Me : 0

view all queries New Assigned ... Replied Feedback Closed

Viewing Queries (0 - 0 / 0) [First Prev 1 Next Last]

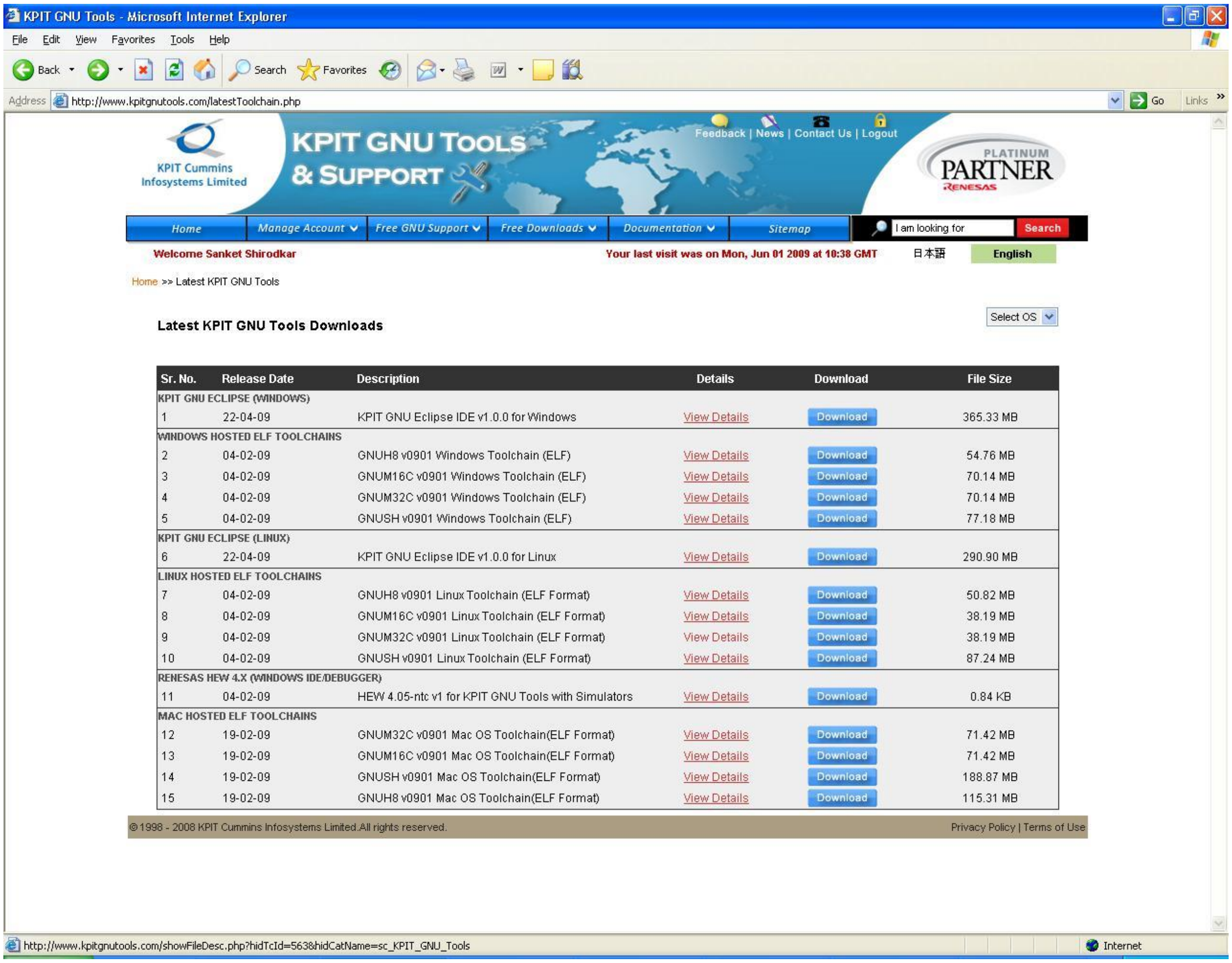
Sr.No.	ID	Email	Company	Country	Problem Area	Status	Resolution	Updated	Short Description

[First Prev 1 Next Last]

©1998 - 2008 KPIT Cummins Infosystems Limited.All rights reserved. Privacy Policy | Terms of Use

Step 3

Select the download you require from the list displayed. For this tutorial we will be using the GNUSH [ELF] v0901 toolchain, so click on the "Download‖ button to the right of "GNUSH v0901 Windows Toolchain (ELF)" in the list.


KPIT GNU Tools - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.kpitgnutools.com/latestToolchain.php
Go Links
KPIT Cummins Infosystems Limited
KPIT GNU TOOLS & SUPPORT
Feedback | News | Contact Us | Logout
PLATINUM PARTNER RENESAS
Home
Manage Account
Free GNU Support
Free Downloads
Documentation
Sitemap
I am looking for
Search
日本語
English
Select OS
http://www.kpitgnutools.com/showFileDesc.php?hidTcId=563&hidCatName=sc_KPIT_GNU_Tools
Internet

Welcome Sanket Shirodkar

Your last visit was on Mon, Jun 01 2009 at 10:38 GMT

Home >> Latest KPIT GNU Tools

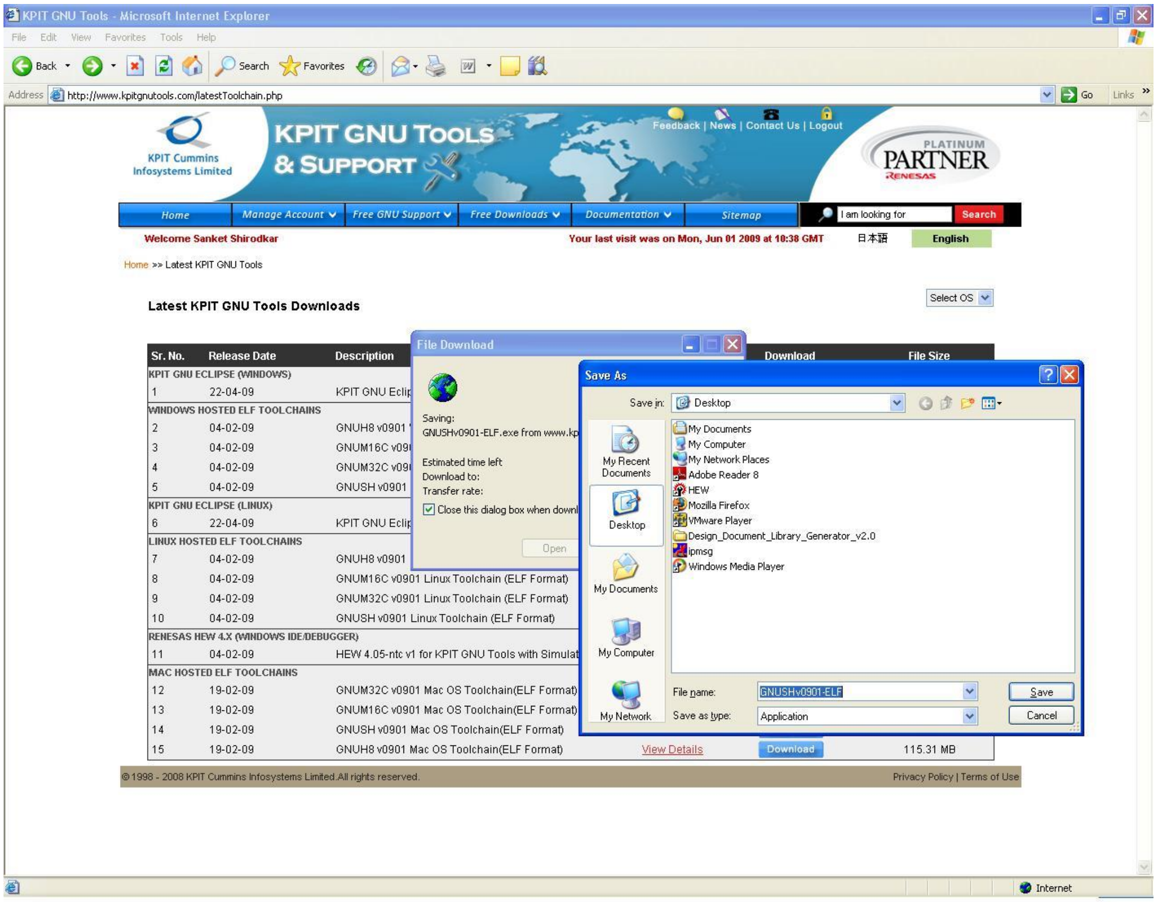
Latest KPIT GNU Tools Downloads

Sr. No.	Release Date	Description	Details	Download	File Size
KPIT GNU ECLIPSE (WINDOWS)					
1	22-04-09	KPIT GNU Eclipse IDE v1.0.0 for Windows	View Details	Download	365.33 MB
WINDOWS HOSTED ELF TOOLCHAINS					
2	04-02-09	GNUH8 v0901 Windows Toolchain (ELF)	View Details	Download	54.76 MB
3	04-02-09	GNUM16C v0901 Windows Toolchain (ELF)	View Details	Download	70.14 MB
4	04-02-09	GNUM32C v0901 Windows Toolchain (ELF)	View Details	Download	70.14 MB
5	04-02-09	GNUSH v0901 Windows Toolchain (ELF)	View Details	Download	77.18 MB
KPIT GNU ECLIPSE (LINUX)					
6	22-04-09	KPIT GNU Eclipse IDE v1.0.0 for Linux	View Details	Download	290.90 MB
LINUX HOSTED ELF TOOLCHAINS					
7	04-02-09	GNUH8 v0901 Linux Toolchain (ELF Format)	View Details	Download	50.82 MB
8	04-02-09	GNUM16C v0901 Linux Toolchain (ELF Format)	View Details	Download	38.19 MB
9	04-02-09	GNUM32C v0901 Linux Toolchain (ELF Format)	View Details	Download	38.19 MB
10	04-02-09	GNUSH v0901 Linux Toolchain (ELF Format)	View Details	Download	87.24 MB
RENESAS HEW 4.X (WINDOWS IDE/DEBUGGER)					
11	04-02-09	HEW 4.05-ntc v1 for KPIT GNU Tools with Simulators	View Details	Download	0.84 KB
MAC HOSTED ELF TOOLCHAINS					
12	19-02-09	GNUM32C v0901 Mac OS Toolchain(ELF Format)	View Details	Download	71.42 MB
13	19-02-09	GNUM16C v0901 Mac OS Toolchain(ELF Format)	View Details	Download	71.42 MB
14	19-02-09	GNUSH v0901 Mac OS Toolchain(ELF Format)	View Details	Download	188.87 MB
15	19-02-09	GNUH8 v0901 Mac OS Toolchain(ELF Format)	View Details	Download	115.31 MB

© 1998 - 2008 KPIT Cummins Infosystems Limited.All rights reserved. Privacy Policy | Terms of Use

Step 4

Depending on which web browser you are using, you may now be given various options. In this case we used the IE 6 browser and chose to save the file to disk. Whichever method you use, you should save the download somewhere and then open the executable to install the GNUSH [ELF] v0901 toolchain.


KPIT GNU Tools - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address
Go Links
KPIT Cummins Infosystems Limited
KPIT GNU TOOLS & SUPPORT
Feedback | News | Contact Us | Logout
PLATINUM PARTNER RENESAS
Home | Manage Account | Free GNU Support | Free Downloads | Documentation | Sitemap
I am looking for
Search
日本語
English
Select OS
Internet

http://www.kpitgnutools.com/latestToolchain.php

Welcome Sanket Shirodkar

Your last visit was on Mon, Jun 01 2009 at 10:38 GMT

Home >> Latest KPIT GNU Tools

Latest KPIT GNU Tools Downloads

Sr. No.	Release Date	Description		Download	File Size
KPIT GNU ECLIPSE (WINDOWS)					
1	22-04-09	KPIT GNU Eclip			
WINDOWS HOSTED ELF TOOLCHAINS					
2	04-02-09	GNUH8 v0901			
3	04-02-09	GNUM16C v09			
4	04-02-09	GNUM32C v09			
5	04-02-09	GNUSH v0901			
KPIT GNU ECLIPSE (LINUX)					
6	22-04-09	KPIT GNU Eclip			
LINUX HOSTED ELF TOOLCHAINS					
7	04-02-09	GNUH8 v0901			
8	04-02-09	GNUM16C v0901 Linux Toolchain (ELF Format)			
9	04-02-09	GNUM32C v0901 Linux Toolchain (ELF Format)			
10	04-02-09	GNUSH v0901 Linux Toolchain (ELF Format)			
RENESAS HEW 4.X (WINDOWS IDE/DEBUGGER)					
11	04-02-09	HEW 4.05-ntc v1 for KPIT GNU Tools with Simulat			
MAC HOSTED ELF TOOLCHAINS					
12	19-02-09	GNUM32C v0901 Mac OS Toolchain(ELF Format)			
13	19-02-09	GNUM16C v0901 Mac OS Toolchain(ELF Format)			
14	19-02-09	GNUSH v0901 Mac OS Toolchain(ELF Format)			
15	19-02-09	GNUH8 v0901 Mac OS Toolchain(ELF Format)	View Details	Download	115.31 MB

©1998 - 2008 KPIT Cummins Infosystems Limited.All rights reserved. Privacy Policy | Terms of Use

File Download

Saving:
GNUSHv0901-ELF.exe from www.kp

Estimated time left
Download to:
Transfer rate:

☑ Close this dialog box when downl

Open

Save As

Save in: Desktop

My Documents
My Computer
My Network Places
Adobe Reader 8
HEW
Mozilla Firefox
VMware Player
Design_Document_Library_Generator_v2.0
ipmsg
Windows Media Player

File name: GNUSHv0901-ELF

Save as type: Application

Save

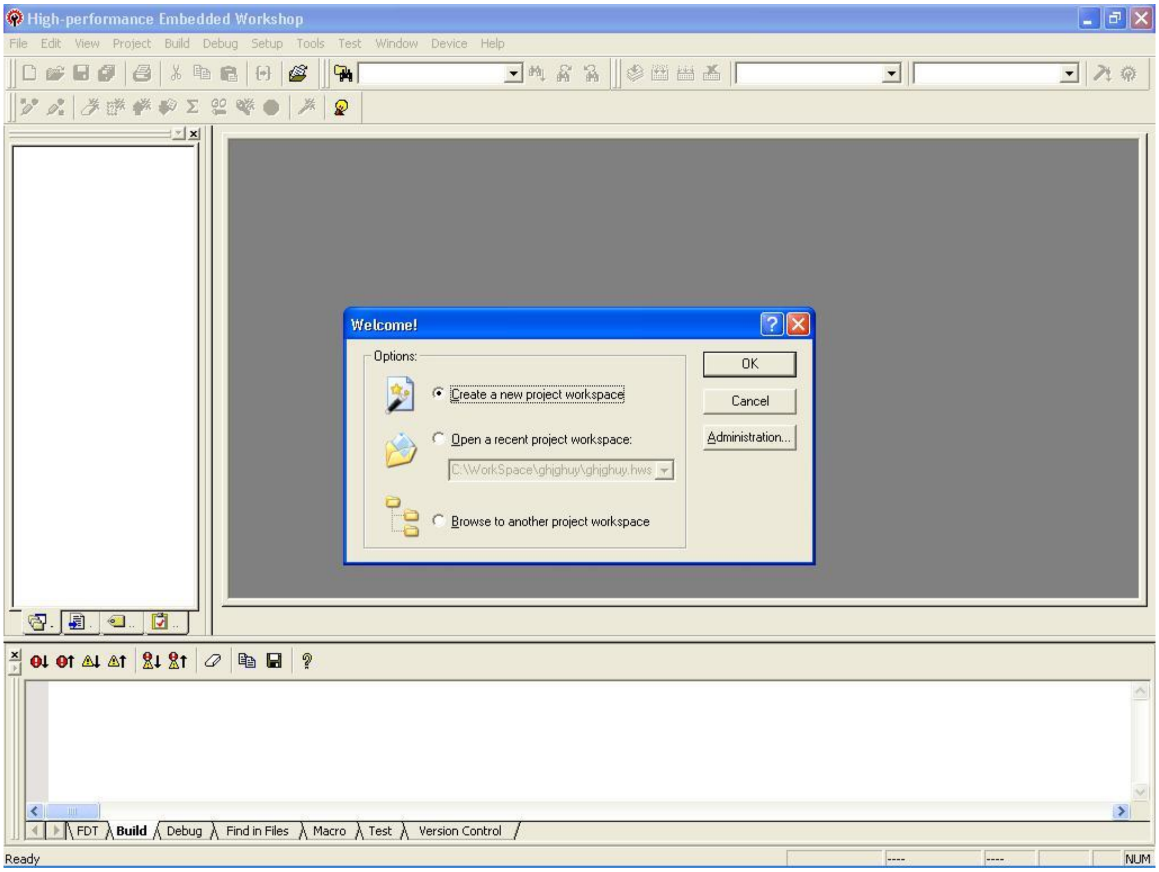
Cancel

[*Note*: the installer will detect any HEW v4.x or later version you have previously installed and will automatically register the GNU tools with HEW. If you have not already installed HEW, however, you should download the latest version from the support/download system and install it prior to installing GNUSH toolchain]

2.6.2 Creating and Building a Sample Project Using HEW

Step 1

Launch HEW from the Start menu and select ¯Create a new project workspace‖ from the welcome menu. Click OK to proceed.


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Device Help
Welcome!
OK
Cancel
Administration...
FDT Build Debug Find in Files Macro Test Version Control
Ready
NUM

Options:

● Create a new project workspace

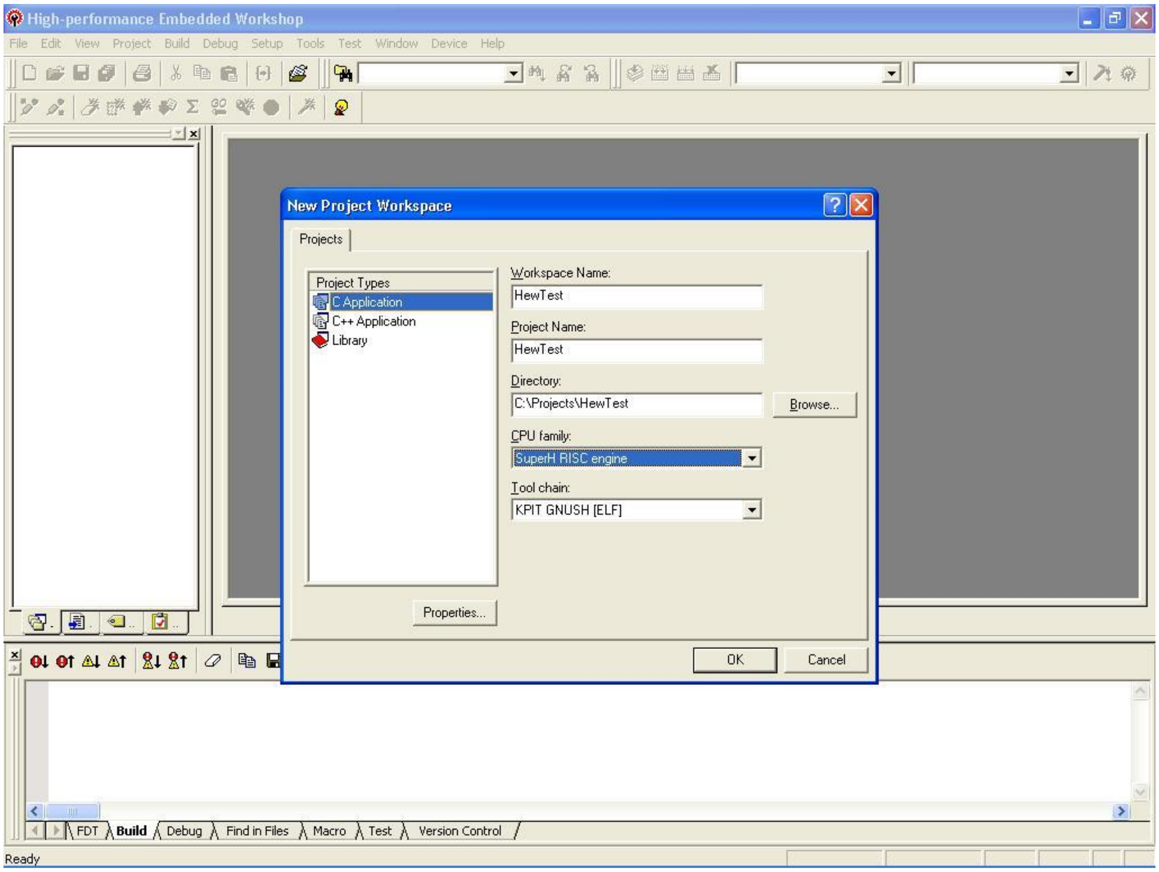
○ Open a recent project workspace:

C:\WorkSpace\ghjghuy\ghjghuy.hws

○ Browse to another project workspace

Step 2

Type the name of your project (‖HewTest‖) into the ‖Workspace Name‖ box. Next, use the ‖Browse‖ button to choose a folder in which to create the project folder. For this tutorial we will use ‖C:\Projects‖. As we will be creating an SH project, choose ‖SuperH RISC engine‖ as the CPU family. ‖KPIT GNUSH [ELF]‖ should be selected as the toolchain. Select the ‖C Application‖ project type. Finally, press the OK button to proceed to the next stage of project generation.



New Project Workspace

Project Types: C Application (selected), C++ Application, Library

Workspace Name: HewTest

Project Name: HewTest

Directory: C:\Projects\HewTest

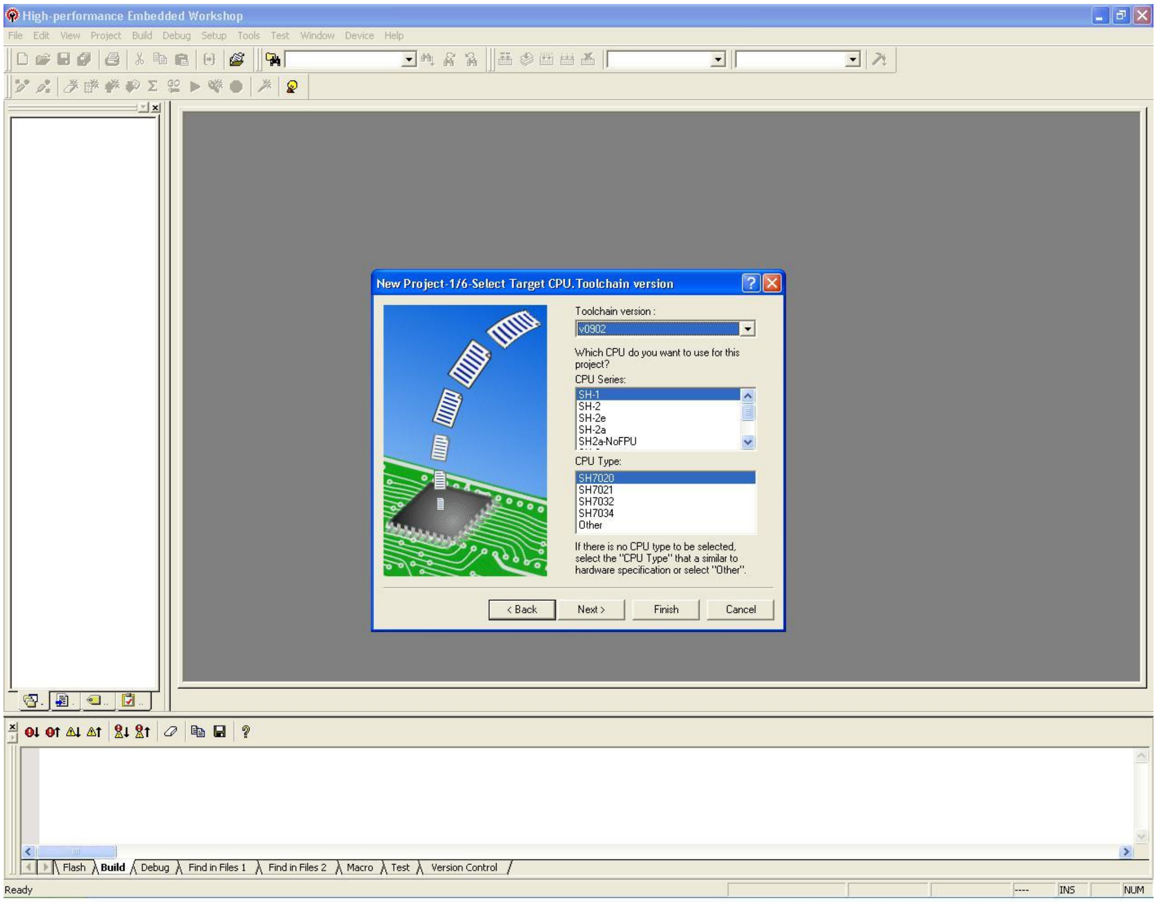
CPU family: SuperH RISC engine

Tool chain: KPIT GNUSH [ELF]

High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Device Help
Projects
Browse...
Properties...
OK
Cancel
FDT
Build
Debug
Find in Files
Macro
Test
Version Control
Ready

Step 3

The project generator now offers you a choice of toolchain version and target CPUs to create your project for. For this tutorial we will create a project with GNUSH v0902 toolchain and for a SH-1 device. ―v0902‖ should therefore be selected as ―Toolchain Version‖ and ―SH-1‖ should be selected from the ―CPU Series‖ list. Here we will select ―CPU Type‖ as ―SH7020‖. Having done this, click the ―Next‖ button.



New Project-1/6-Select Target CPU.Toolchain version

Toolchain version : v0902

Which CPU do you want to use for this project?

CPU Series:
SH-1
SH-2
SH-2e
SH-2a
SH2a-NoFPU

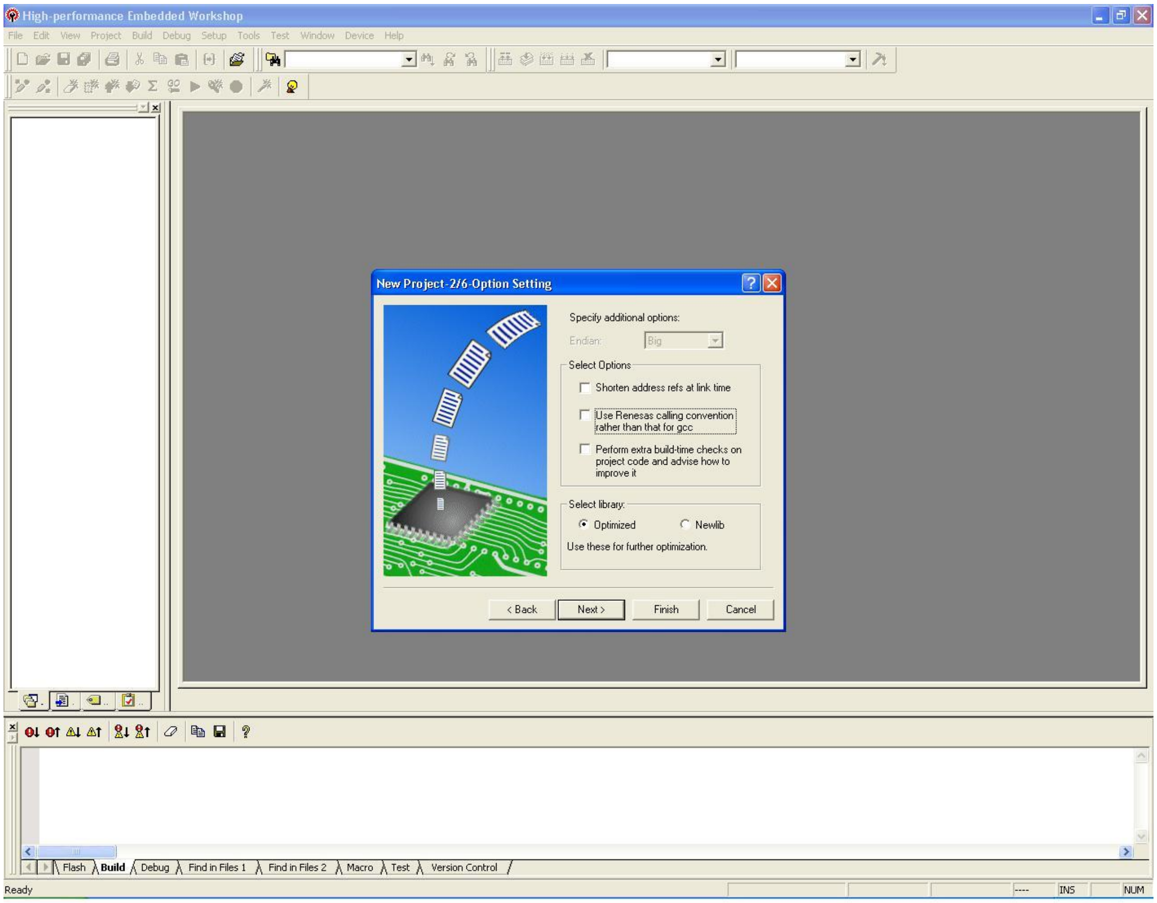
CPU Type:
SH7020
SH7021
SH7032
SH7034
Other

If there is no CPU type to be selected, select the "CPU Type" that a similar to hardware specification or select "Other".

Step 4

The ―Option Setting‖ sheet allows selection of some commonly used compiler features, such as Endianess to be selected. Because we have chosen to generate a project for SH-1 target and as SH-1 targets are Big Endian only the ―Endian‖ switch is set to ―Big‖ by default. All options on this page can also be activated/deactivated later by modifying the compiler options in HEW. For now just leave the options at their default settings and click the ―Next‖ button to proceed.

In case of ‗C Application' projects, ‗Project-Built' ‗Optimized' library will be selected by default and in case of ‗C++ Application' projects, ‗Project-Built' ‗Newlib' library will be selected by default.


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Device Help
Flash Build Debug Find in Files 1 Find in Files 2 Macro Test Version Control
Ready
INS
NUM

New Project-2/6-Option Setting

Specify additional options:

Endian: Big

Select Options

□ Shorten address refs at link time

□ Use Renesas calling convention rather than that for gcc

□ Perform extra build-time checks on project code and advise how to improve it

Select library:

● Optimized ○ Newlib

Use these for further optimization.

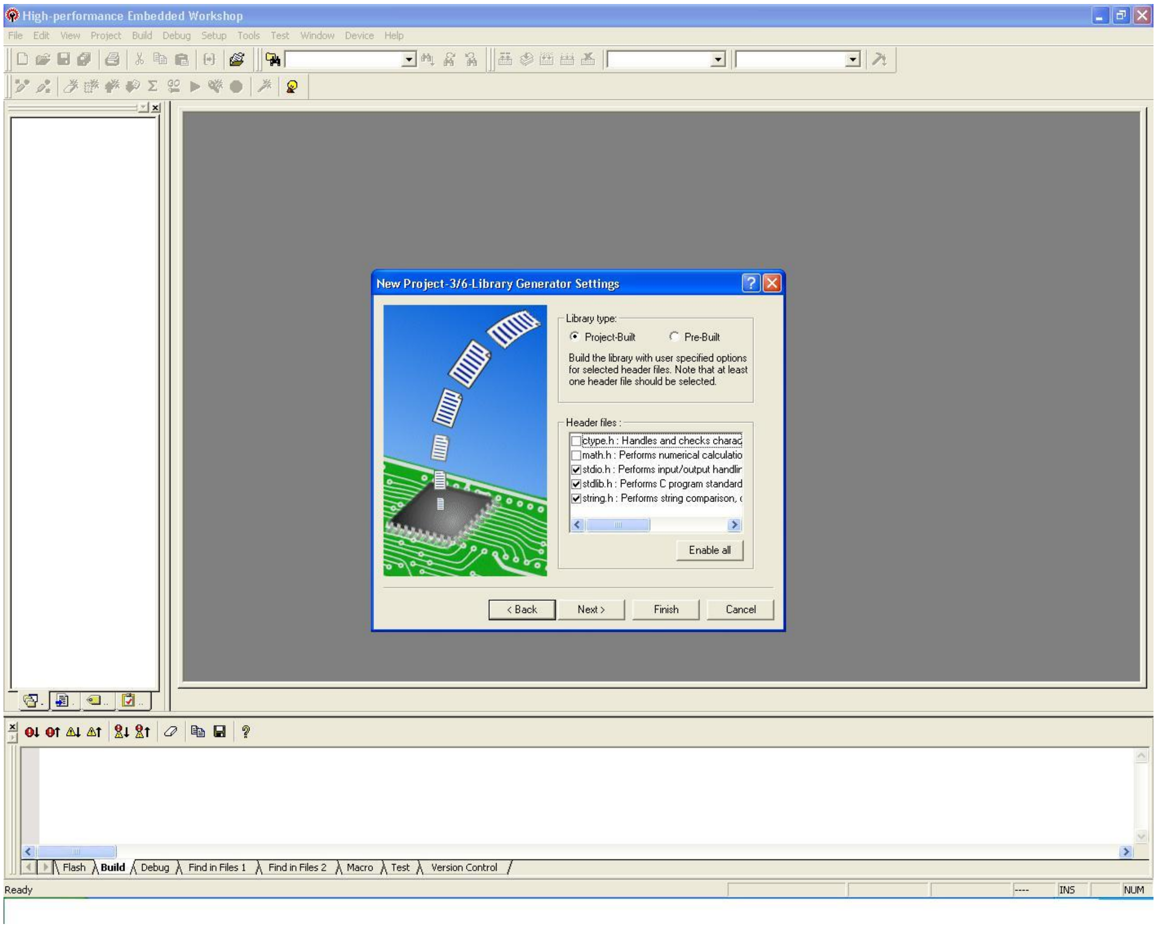
< Back
Next >
Finish
Cancel

Step 5

The ―Library Generator Settings‖ Page allows user to select the ‗Library Type' i.e ‗Project-Built' or ‗Pre-Built'. User can select the header files which will be included in the project.

If the ‗Library type' selected is ‗Project-Built', the library will be built and linked along with the project. However, if the ‗Library type' selected is ‗Pre-Built' the Library Generator phase will not be invoked while building the project. The Pre-Built libraries installed along with the toolchain will be linked with the project. The library will be built depending on the selection of header files.

Click the ‗Next' button to proceed.


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Device Help
Flash Build Debug Find in Files 1 Find in Files 2 Macro Test Version Control
Ready
INS
NUM

New Project-3/6-Library Generator Settings

Library type:

● Project-Built ○ Pre-Built

Build the library with user specified options for selected header files. Note that at least one header file should be selected.

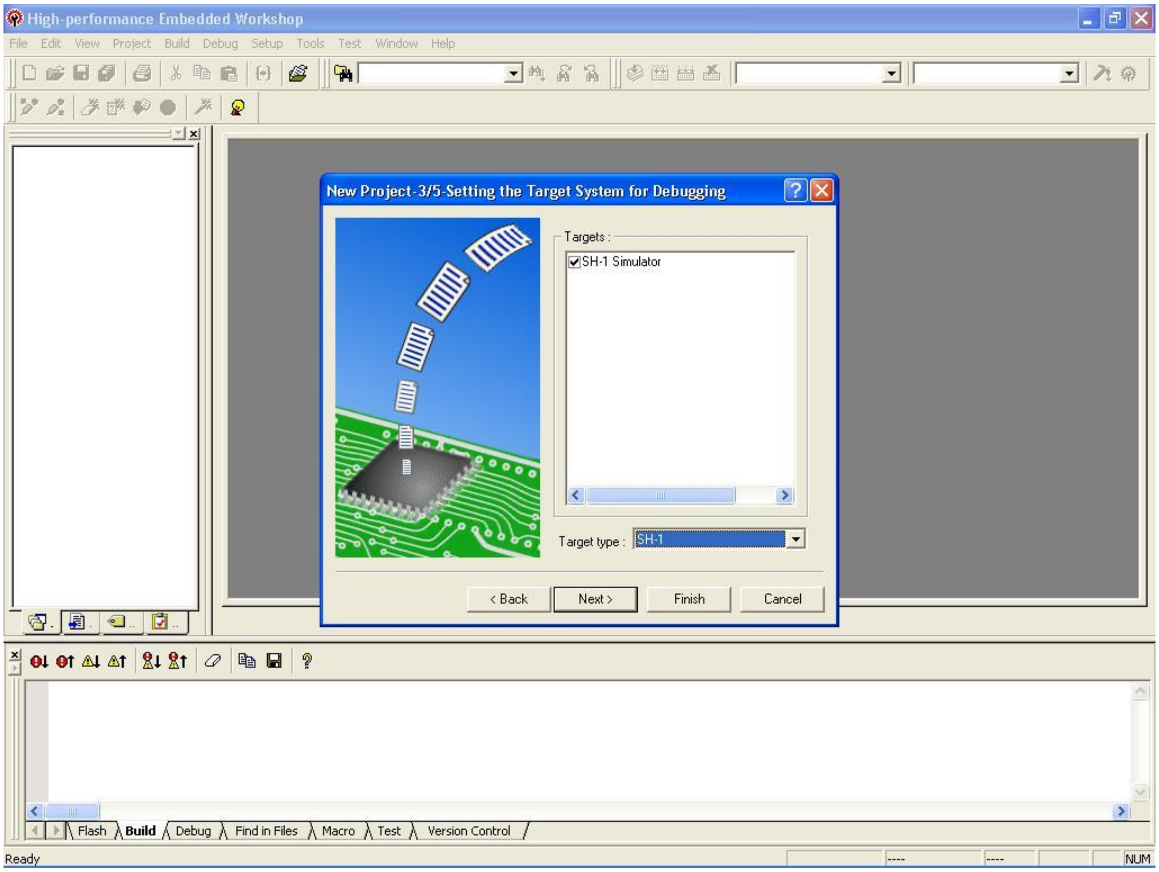
Header files :

- ☐ ctype.h : Handles and checks charac
- ☐ math.h : Performs numerical calculatio
- ☑ stdio.h : Performs input/output handlir
- ☑ stdlib.h : Performs C program standard
- ☑ string.h : Performs string comparison, c

Step 6

The ―Setting the Target System for Debugging‖ page provides the list of target specific si mulators. ―Targets‖ tab sets the debugger targets. User can select the debugger targets (by checking). ―Target type‖ tab specifies the type of the targets displayed in ―Targets‖.

For SH-1 target select ―SH-1 Simulator‖ and click the ―Next‖ button to proceed.



New Project-3/5-Setting the Target System for Debugging

Targets :

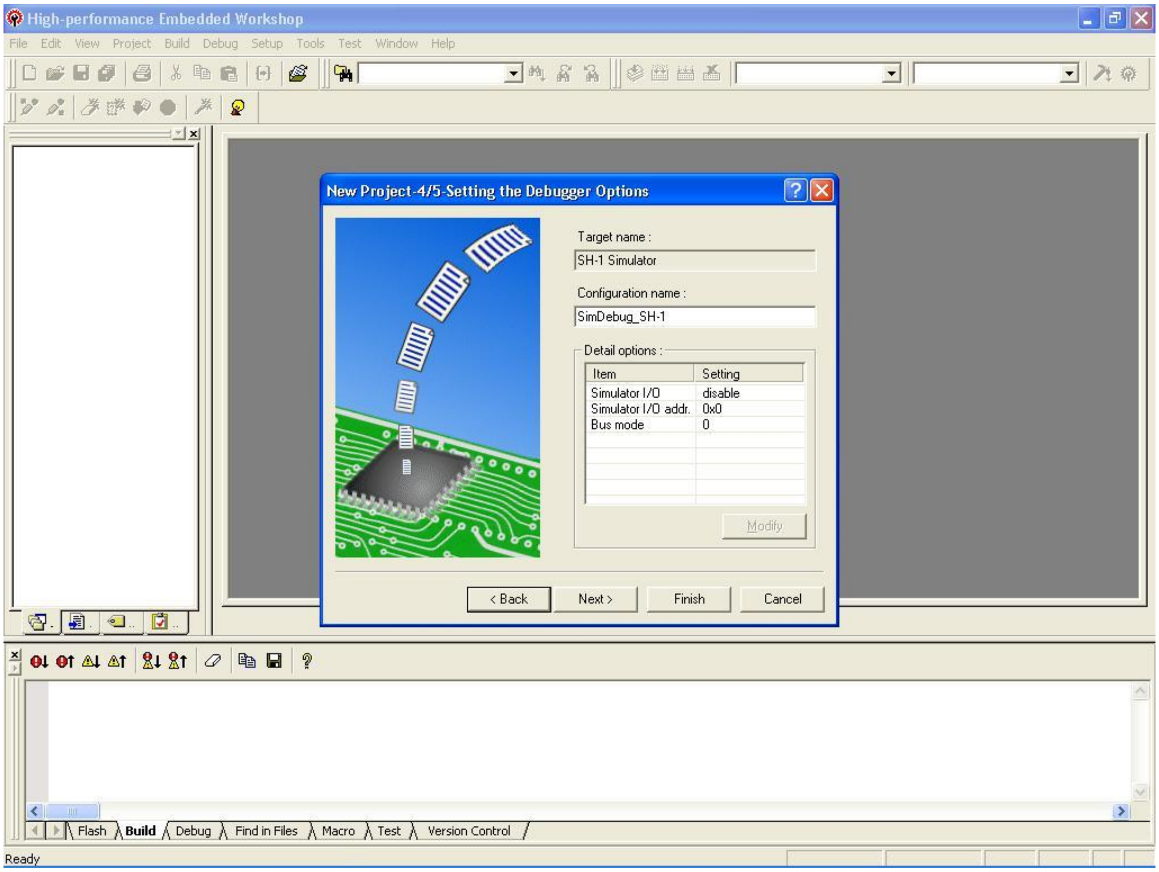
☑ SH-1 Simulator

Target type : SH-1

High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Help
< Back
Next >
Finish
Cancel
Flash
Build
Debug
Find in Files
Macro
Test
Version Control
Ready
NUM

Step 7

The ―Setting the Debugger options‖ page will be displayed only if simulator is selected in step 5. This page gives the ―Target Name‖, ―Configuration Name‖ and ―Detail Options‖. By default, the HEW generates two configurations: [Release] and [Debug]. If a debugger target has been selected, a configuration for the selected target is also generated (an abbreviation including the target name). User can change the [Configuration name].


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Help
New Project-4/5-Setting the Debugger Options
< Back
Next >
Finish
Cancel
Modify
Flash
Build
Debug
Find in Files
Macro
Test
Version Control
Ready

Target name : SH-1 Simulator

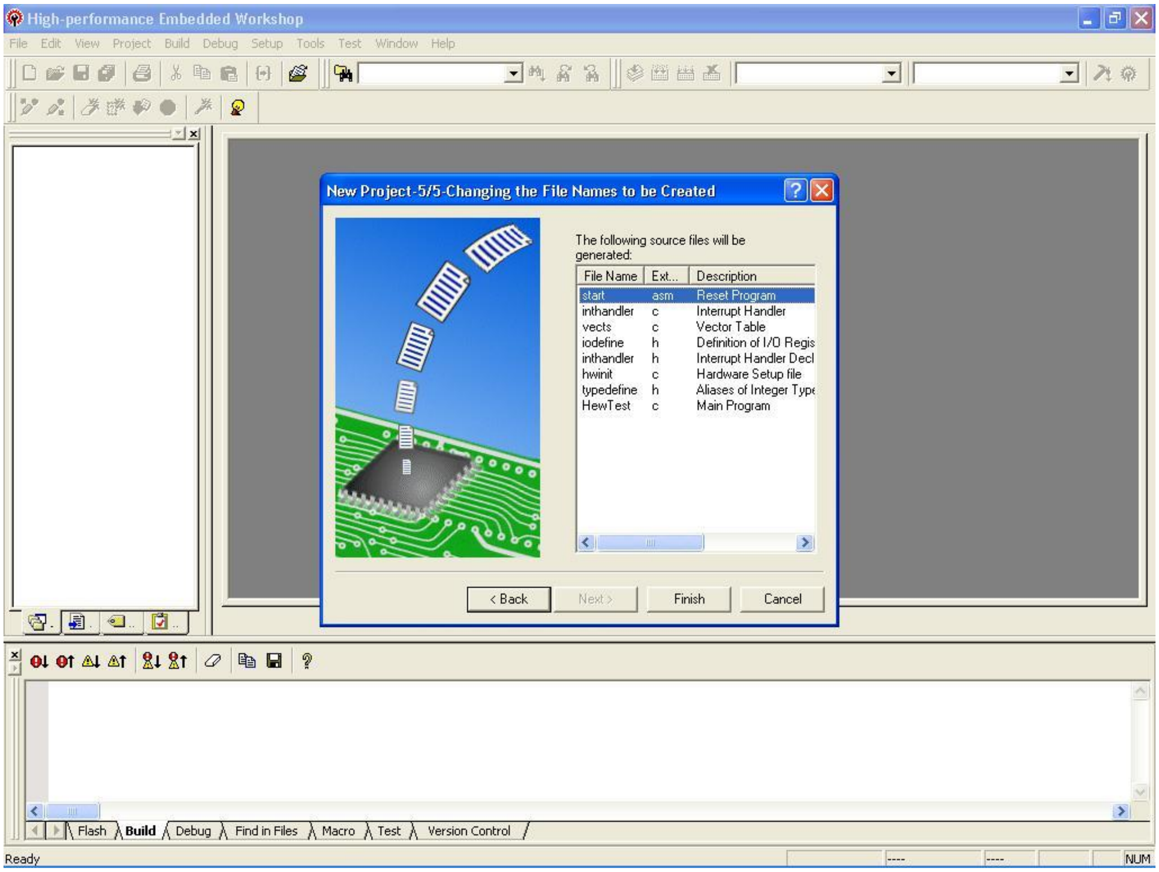
Configuration name : SimDebug_SH-1

Detail options :

Item	Setting
Simulator I/O	disable
Simulator I/O addr.	0x0
Bus mode	0

Step 8

The next page is ―Changing the File Names to be Created‖. This page shows a list of the project files that will be generated along with their extensions and short description. User can change the name of the file. In most cases, these files will need to be tailored after project generation to match the specific hardware in use. In the case of this tutorial, however, hardware will be simulated and the generated project will not require alteration. For now just leave the names at their default settings and click the ―Finish‖ button to generate the project.

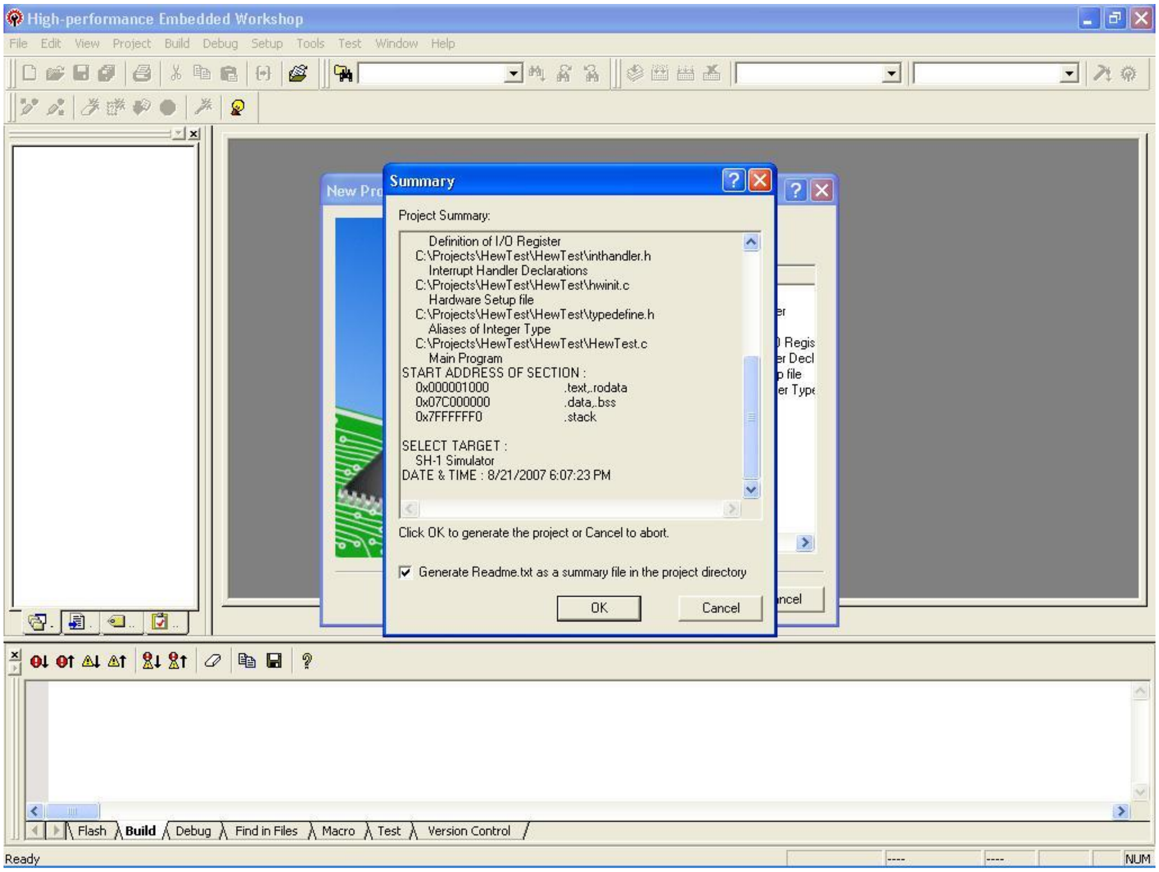

High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Help
New Project-5/5-Changing the File Names to be Created
< Back
Next >
Finish
Cancel
Flash
Build
Debug
Find in Files
Macro
Test
Version Control
Ready
NUM

The following source files will be generated:

File Name	Ext...	Description
start	asm	Reset Program
inthandler	c	Interrupt Handler
vects	c	Vector Table
iodefine	h	Definition of I/O Regis
inthandler	h	Interrupt Handler Decl
hwinit	c	Hardware Setup file
typedefine	h	Aliases of Integer Type
HewTest	c	Main Program

Step 9

HEW now generates the project according to the settings. A summary of the generated files and the start addresses of sections are displayed for information. ¯Generate Readme.txt as a summary file in the project directory‖ is checked by default. It will generate ¯Readme‖ file in the project. Click the ¯OK‖ box to complete project generation.


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Help
New Pro
Summary
Flash Build Debug Find in Files Macro Test Version Control
Ready
NUM

Project Summary:

```
      Definition of I/O Register
   C:\Projects\HewTest\HewTest\inthandler.h
      Interrupt Handler Declarations
   C:\Projects\HewTest\HewTest\hwinit.c
      Hardware Setup file
   C:\Projects\HewTest\HewTest\typedefine.h
      Aliases of Integer Type
   C:\Projects\HewTest\HewTest\HewTest.c
      Main Program
START ADDRESS OF SECTION :
   0x000001000              .text,.rodata
   0x07C000000              .data,.bss
   0x7FFFFFF0               .stack

SELECT TARGET :
   SH-1 Simulator
DATE & TIME : 8/21/2007 6:07:23 PM
```

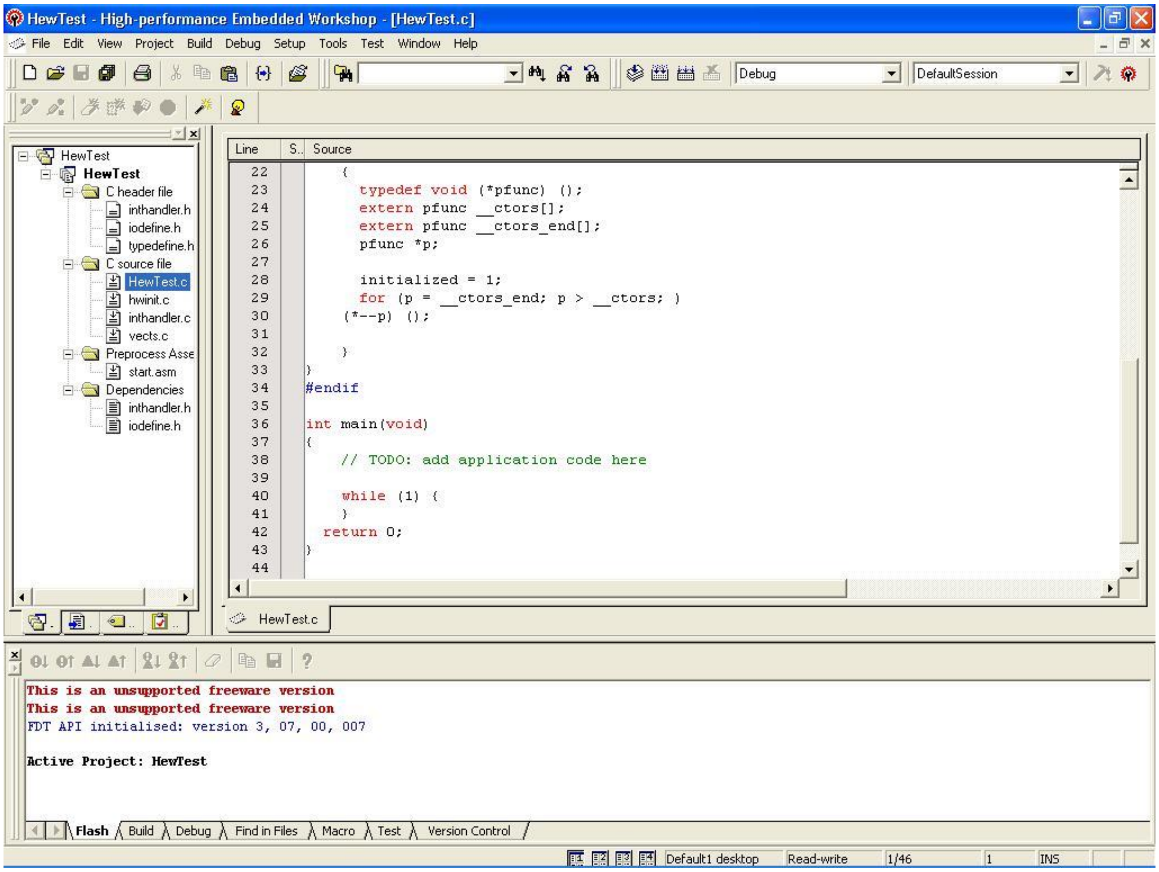
Click OK to generate the project or Cancel to abort.

☑ Generate Readme.txt as a summary file in the project directory

OK Cancel

Step 10

Once the project is generated, HEW now presents a list of project files on the left. These can be double_clicked to display the content of the file in the main editor window. Double click ¯HewTest.c‖ to show the application‘s main() function.

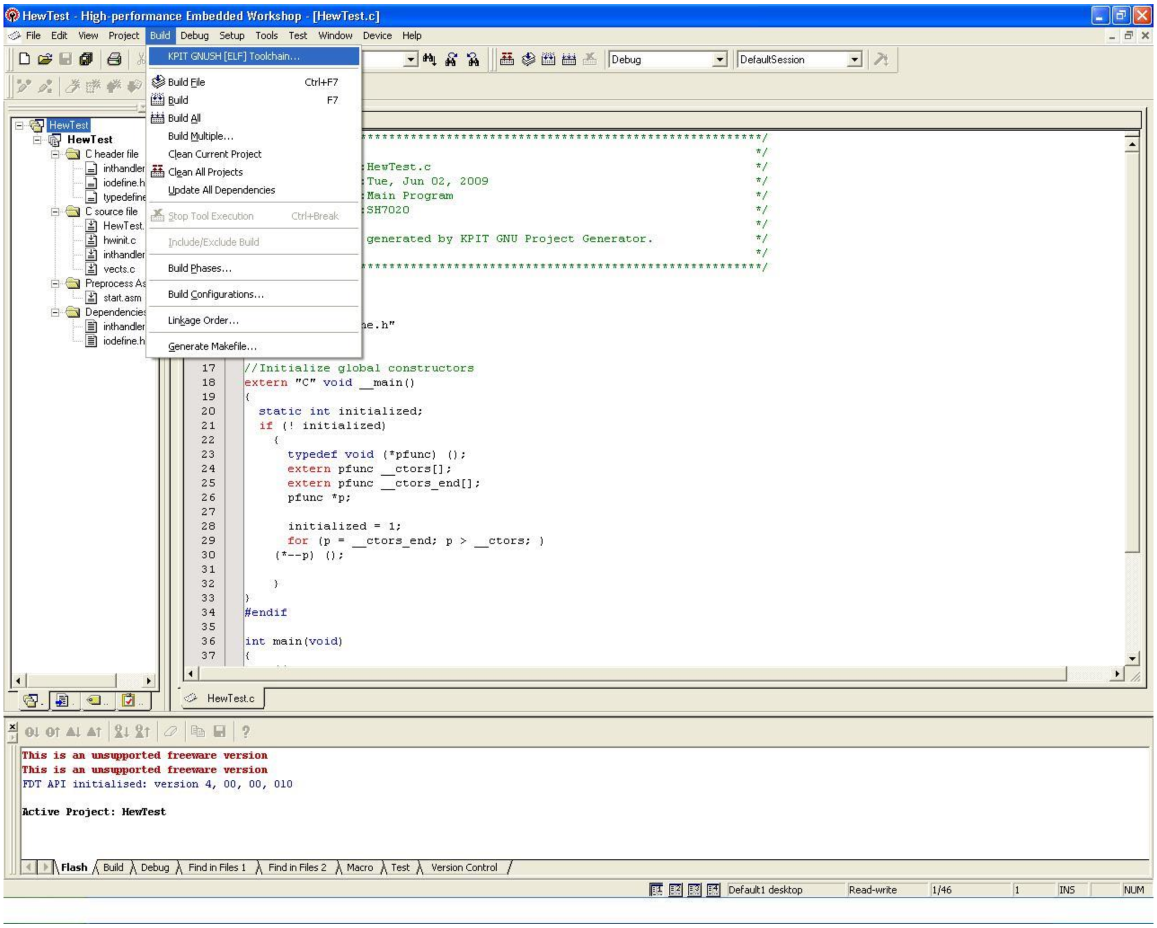


```
    {
      typedef void (*pfunc) ();
      extern pfunc __ctors[];
      extern pfunc __ctors_end[];
      pfunc *p;

      initialized = 1;
      for (p = __ctors_end; p > __ctors; )
    (*--p) ();

    }
}
#endif

int main(void)
{
    // TODO: add application code here

    while (1) {
    }
  return 0;
}
```

```
This is an unsupported freeware version
This is an unsupported freeware version
FDT API initialised: version 3, 07, 00, 007

Active Project: HewTest
```

Step 11

To switch on optimization for this project, select the ―KPIT GNUSH[ELF] Toolchain…‖ item from the ―Build‖ menu on the menu bar.

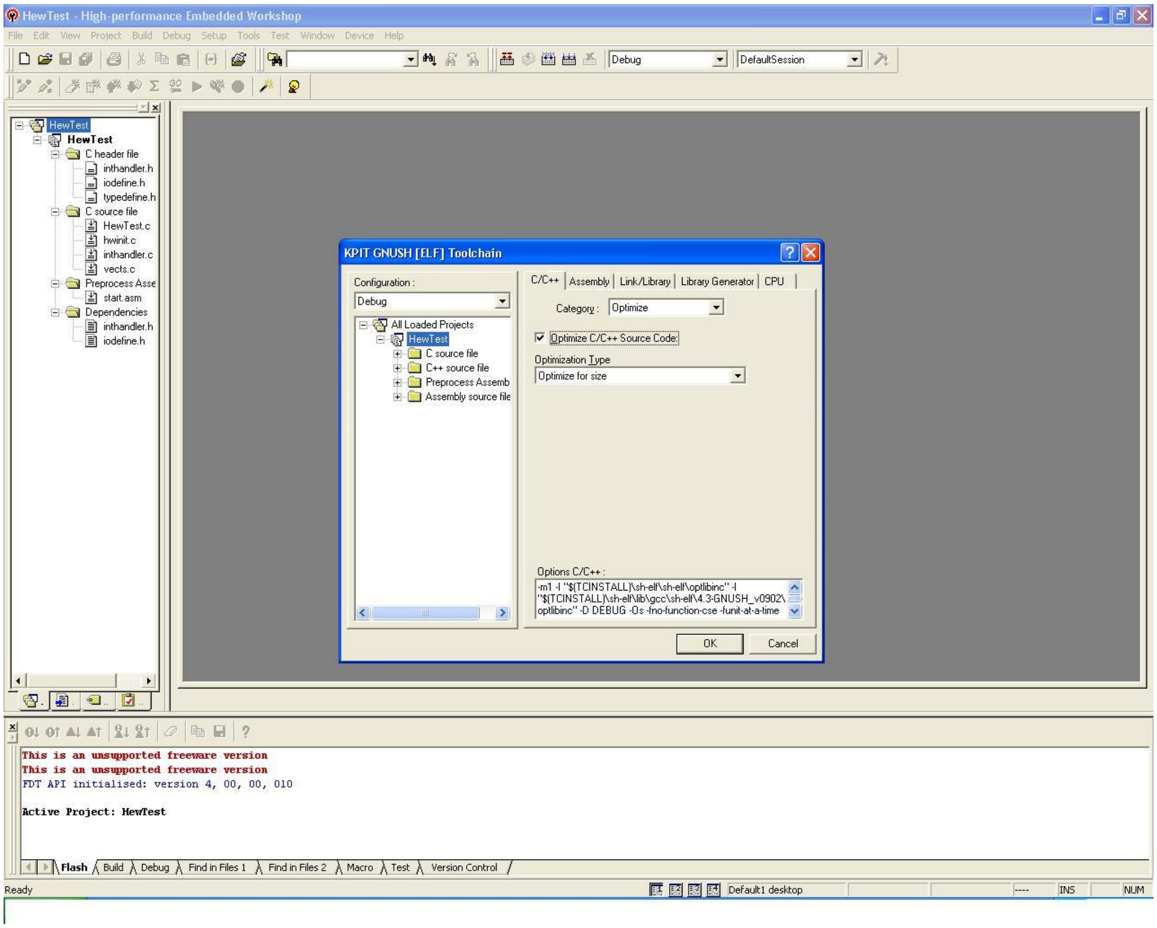


```
************************************************************/
                                                           */
:HewTest.c                                                 */
:Tue, Jun 02, 2009                                         */
:Main Program                                              */
:SH7020                                                    */
                                                           */
 generated by KPIT GNU Project Generator.                  */
                                                           */
************************************************************/

ne.h"

//Initialize global constructors
extern "C" void __main()
{
  static int initialized;
  if (! initialized)
    {
      typedef void (*pfunc) ();
      extern pfunc __ctors[];
      extern pfunc __ctors_end[];
      pfunc *p;

      initialized = 1;
      for (p = __ctors_end; p > __ctors; )
    (*--p) ();

    }
}
#endif

int main(void)
{
```

```
This is an unsupported freeware version
This is an unsupported freeware version
FDT API initialised: version 4, 00, 00, 010

Active Project: HewTest
```

Step 12

In the C/C++ dialog box that appears, select the Category as ―Optimize‖ from the drop down menu. Click on the ―Optimize C/C++ Source Code‖ tick box.



Configuration : Debug

Category : Optimize

☑ Optimize C/C++ Source Code:

Optimization Type: Optimize for size

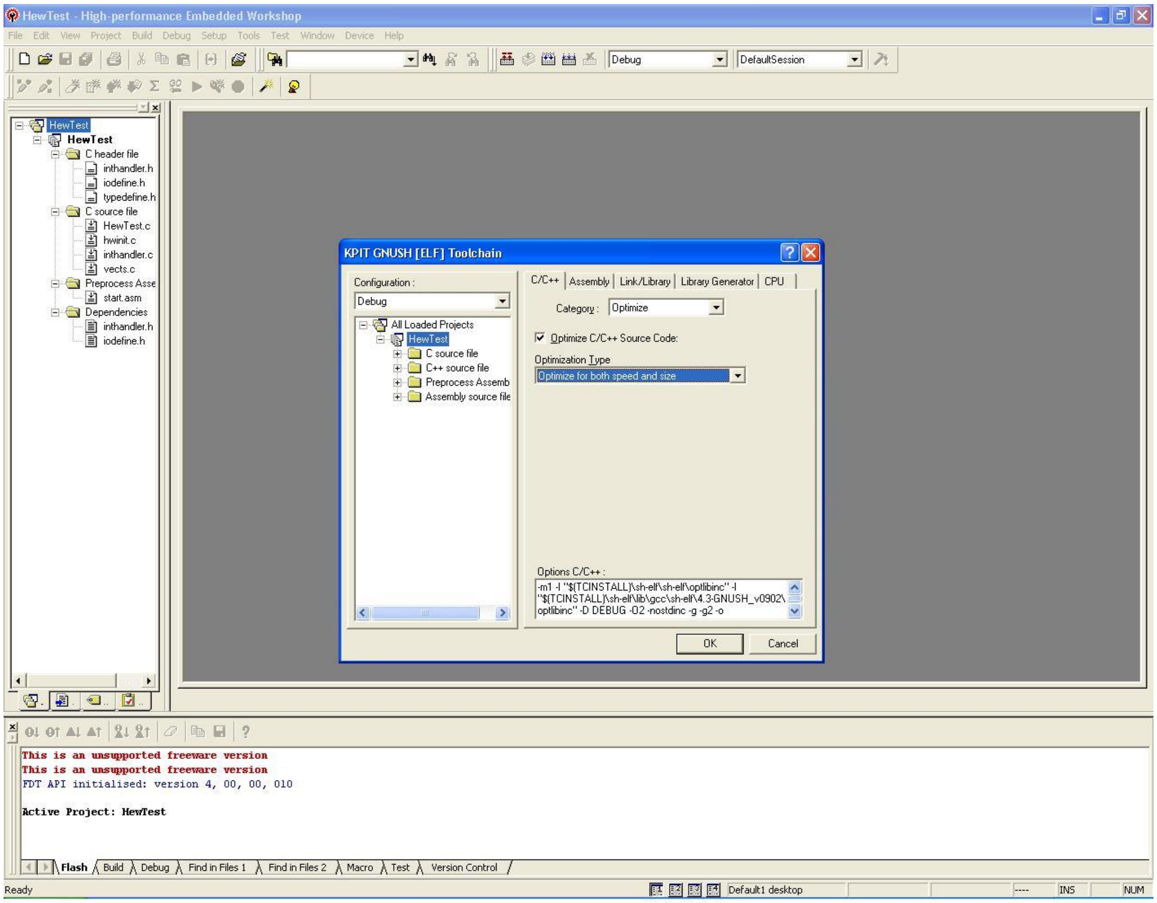
Options C/C++ :

```
-m1 -I "$(TCINSTALL)\sh-elf\sh-elf\optlibinc" -I
"$(TCINSTALL)\sh-elf\lib\gcc\sh-elf\4.3-GNUSH_v0902\
optlibinc" -D DEBUG -Os -fno-function-cse -funit-at-a-time
```

```
This is an unsupported freeware version
This is an unsupported freeware version
FDT API initialised: version 4, 00, 00, 010

Active Project: HewTest
```

Step 13

Click on the drop down menu labeled ―Optimisation Type‖ and a menu of optimization options will appear. From this, choose ―Optimize for both speed and size‖. By expanding the tree view on the left, settings can be changed for individual files.



Configuration : Debug

Category : Optimize

☑ Optimize C/C++ Source Code:

Optimization Type: Optimize for both speed and size

Options C/C++ :

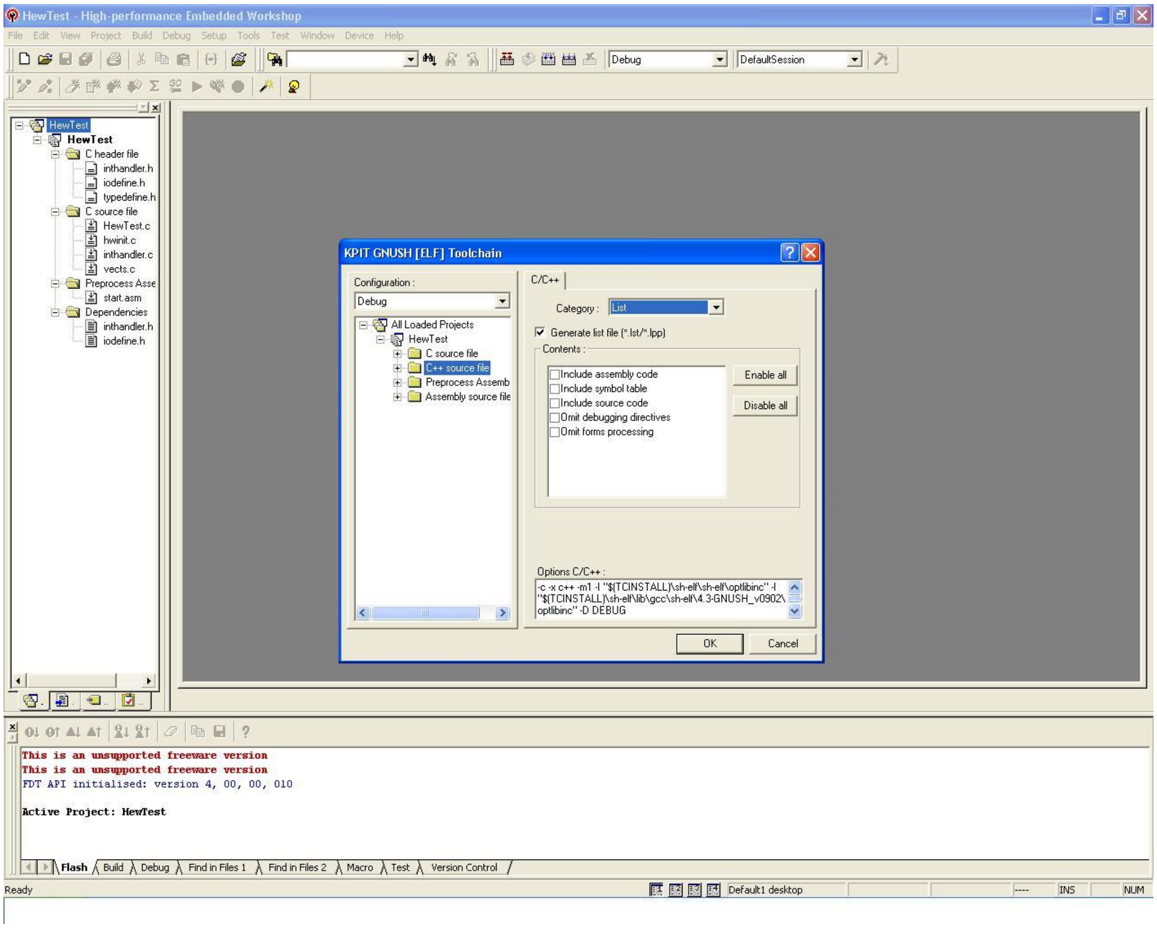
```
-m1 -I "$(TCINSTALL)\sh-elf\sh-elf\optlibinc" -I
"$(TCINSTALL)\sh-elf\lib\gcc\sh-elf\4.3-GNUSH_v0902\
optlibinc" -D DEBUG -O2 -nostdinc -g -g2 -o
```

```
This is an unsupported freeware version
This is an unsupported freeware version
FDT API initialised: version 4, 00, 00, 010

Active Project: HewTest
```

Step 14

Next, select the category as ―List‖ from the drop down menu. Click on the ―Generate list file‖ tick. Each list file will show the source code together with the assembly code the compiler generated for that source. Each list file is generated into the configuration folder.

Finally, click ―OK‖ to commit the changes to the project and to close compiler options dialog box.



HewTest - High-performance Embedded Workshop

KPIT GNUSH [ELF] Toolchain

Configuration : Debug

C/C++

Category : List

☑ Generate list file (*.lst/*.lpp)

Contents :

□ Include assembly code
□ Include symbol table
□ Include source code
□ Omit debugging directives
□ Omit forms processing

Options C/C++ :

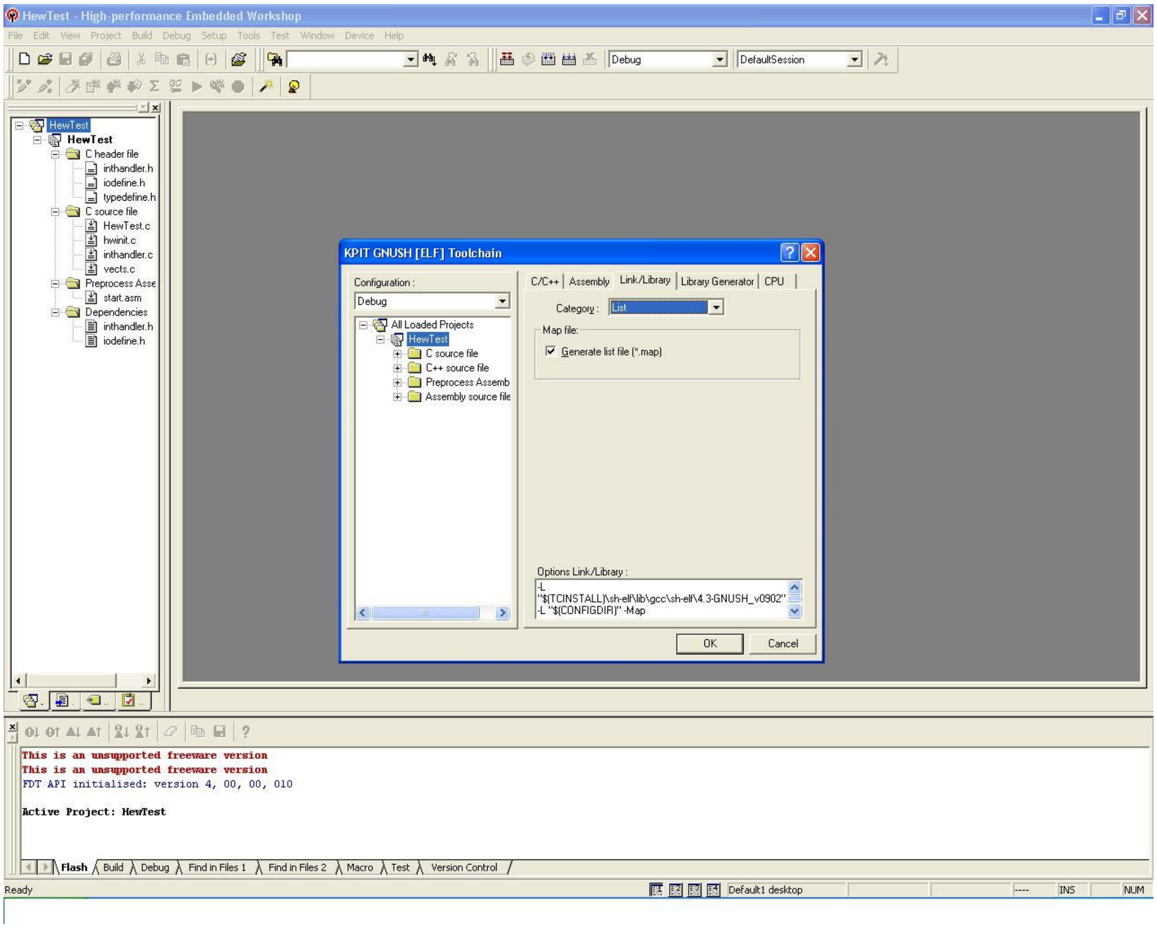
```
-c -x c++ -m1 -I "$(TCINSTALL)\sh-elf\sh-elf\optlibinc" -I
"$(TCINSTALL)\sh-elf\lib\gcc\sh-elf\4.3-GNUSH_v0902\
optlibinc" -D DEBUG
```

```
This is an unsupported freeware version
This is an unsupported freeware version
FDT API initialised: version 4, 00, 00, 010

Active Project: HewTest
```

Step 15

Next, select the ―Link/Library‖ tab. Select the ―List‖ from ―Category‖ dropdown box.

Click on the option ―Generate list file(*.map)‖. This will cause the linker to generate a mapfile showing the symbols linked into the final object file, the sections to which they were allocated and information about the size and linked addresses of those sections. The mapfile is generated into the configuration folder. For now, do not modify the settings.



Configuration : Debug

Category : List

Map file:

☑ Generate list file (*.map)

Options Link/Library :

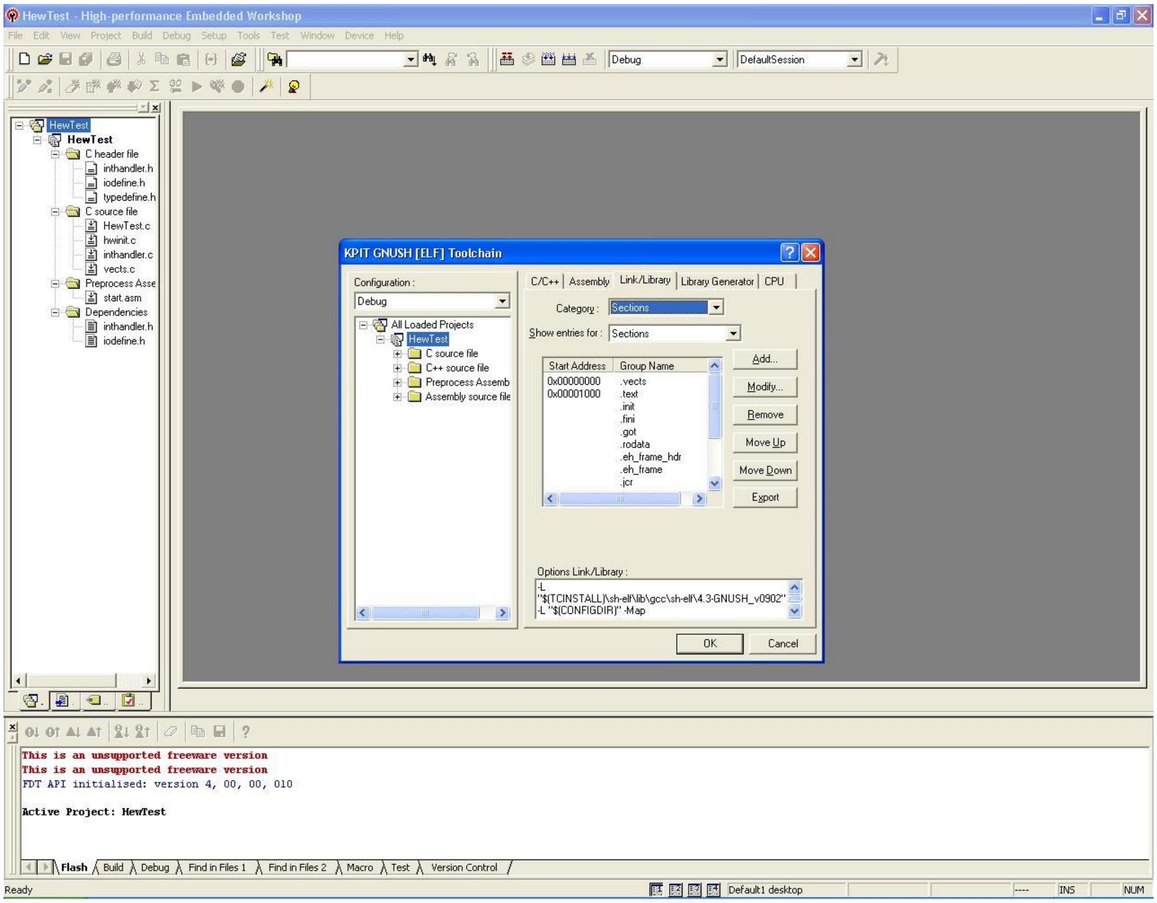
```
-L
"$(TCINSTALL)\sh-elf\lib\gcc\sh-elf\4.3-GNUSH_v0902"
-L "$(CONFIGDIR)" -Map
```

```
This is an unsupported freeware version
This is an unsupported freeware version
FDT API initialised: version 4, 00, 00, 010

Active Project: HewTest
```

Ready

Step 16

Next, select the ―Sections‖ from ―Category‖ dropdown box. This allows you to change the addresses to which the various code and data sections will be linked in your target devices address space. The default settings are correctly configured for a sample device in the CPU family we selected when generating the project (SH-1). When building code for a different device you may need to modify these settings. For now, do not modify the settings, but instead click ―OK‖ to close the linker options dialog box.



Configuration : Debug

Category : Sections

Show entries for : Sections

Start Address	Group Name
0x00000000	.vects
0x00001000	.text
	.init
	.fini
	.got
	.rodata
	.eh_frame_hdr
	.eh_frame
	.jcr

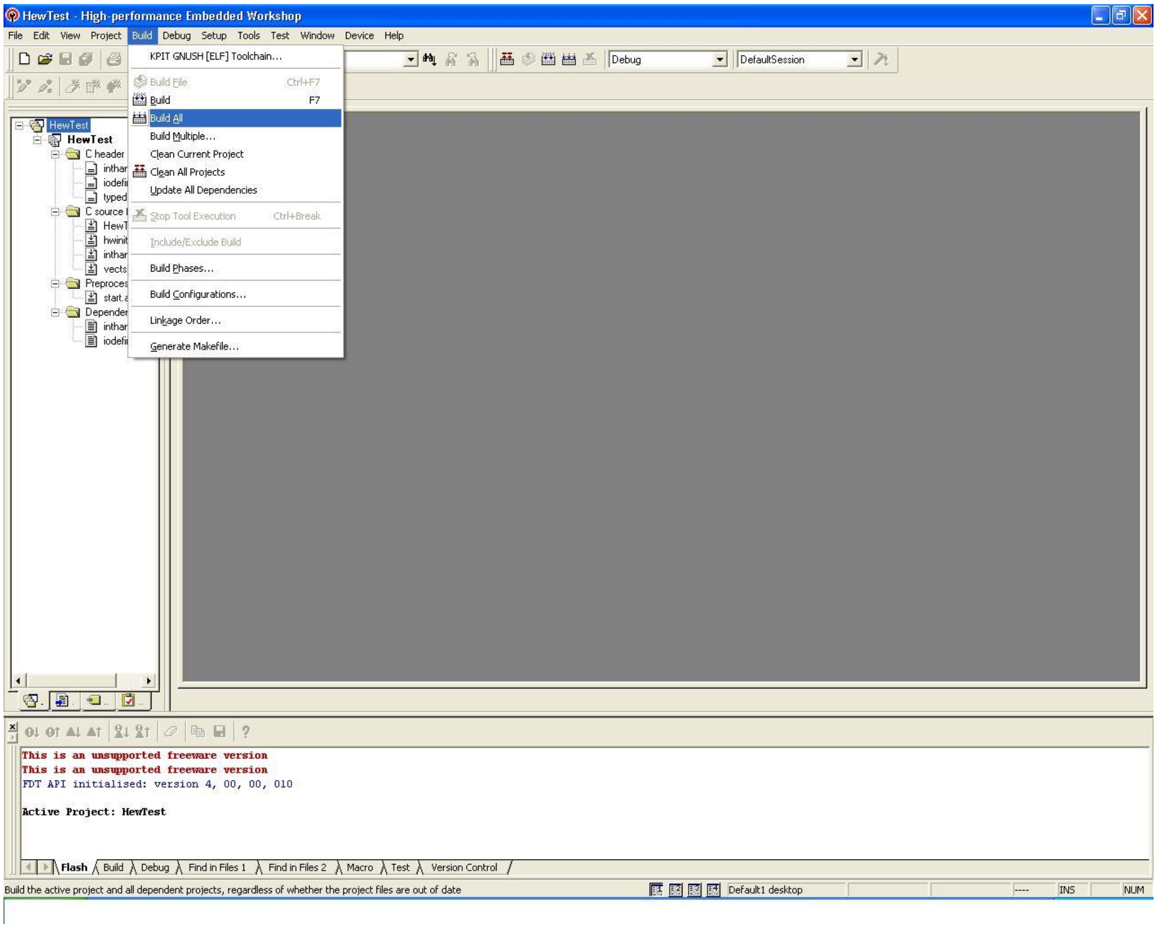
Options Link/Library :

```
-L
"$(TCINSTALL)\sh-elf\lib\gcc\sh-elf\4.3-GNUSH_v0902"
-L "$(CONFIGDIR)" -Map
```

```
This is an unsupported freeware version
This is an unsupported freeware version
FDT API initialised: version 4, 00, 00, 010

Active Project: HewTest
```

Step 17

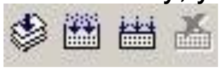
Now that we have made some minor adjustments to the toolchain settings, we will build the project. To do this, select ¯Build All‖ item from the ¯Build‖ menu on the menu bar.



```
This is an unsupported freeware version
This is an unsupported freeware version
FDT API initialised: version 4, 00, 00, 010

Active Project: HewTest
```

Build the active project and all dependent projects, regardless of whether the project files are out of date

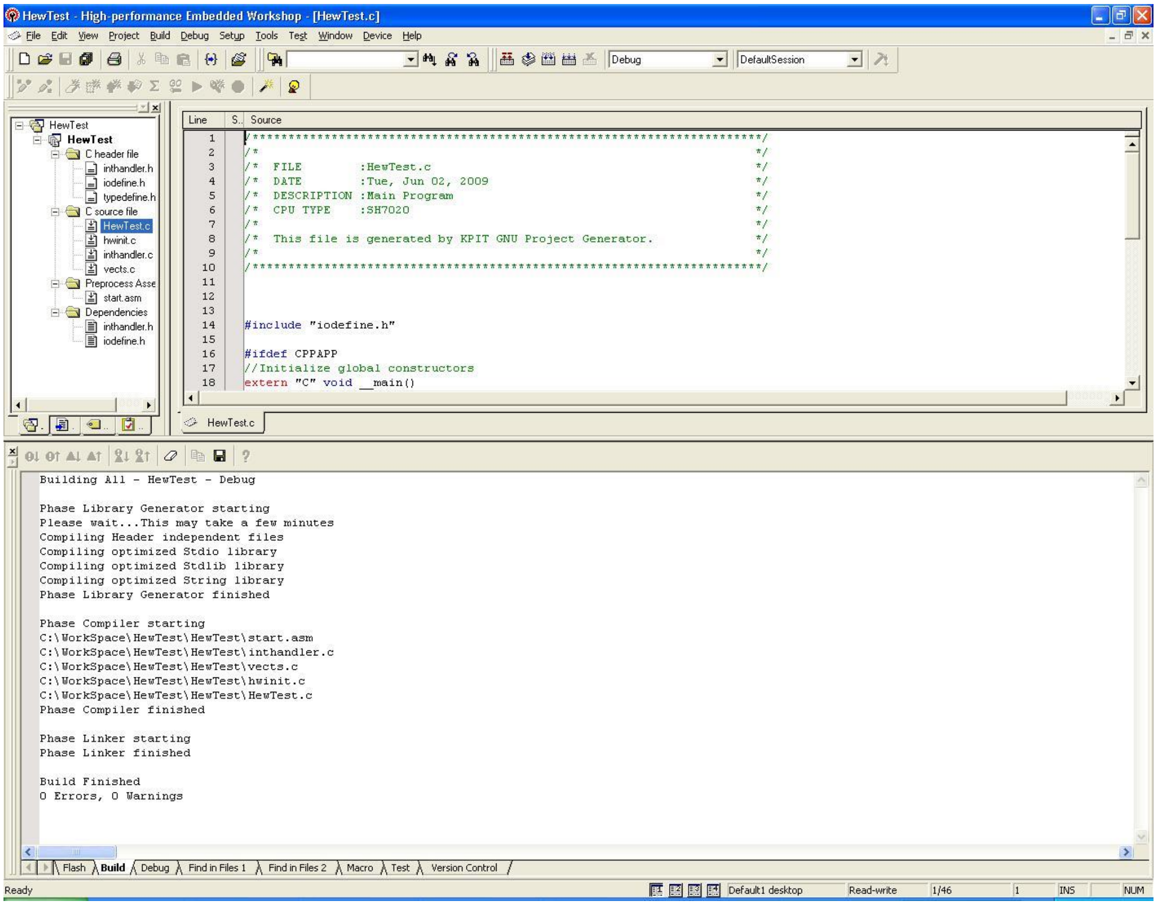
Alternatively, you can click the ¯Build All‖ icon from the ¯Build‖ toolbar:



The ¯Build All‖ icon is located third from left on the toolbar.

Step 18

HEW now creates the library, compiles, assembles and links the various project files appropriately and shows any warnings, errors or other toolchain output in the ¯Build‖ console window in the lower section of the HEW window. If any messages appear that relate to specific project files, double clicking the message will typically display the appropriate line of that project file in the editor window for convenience.

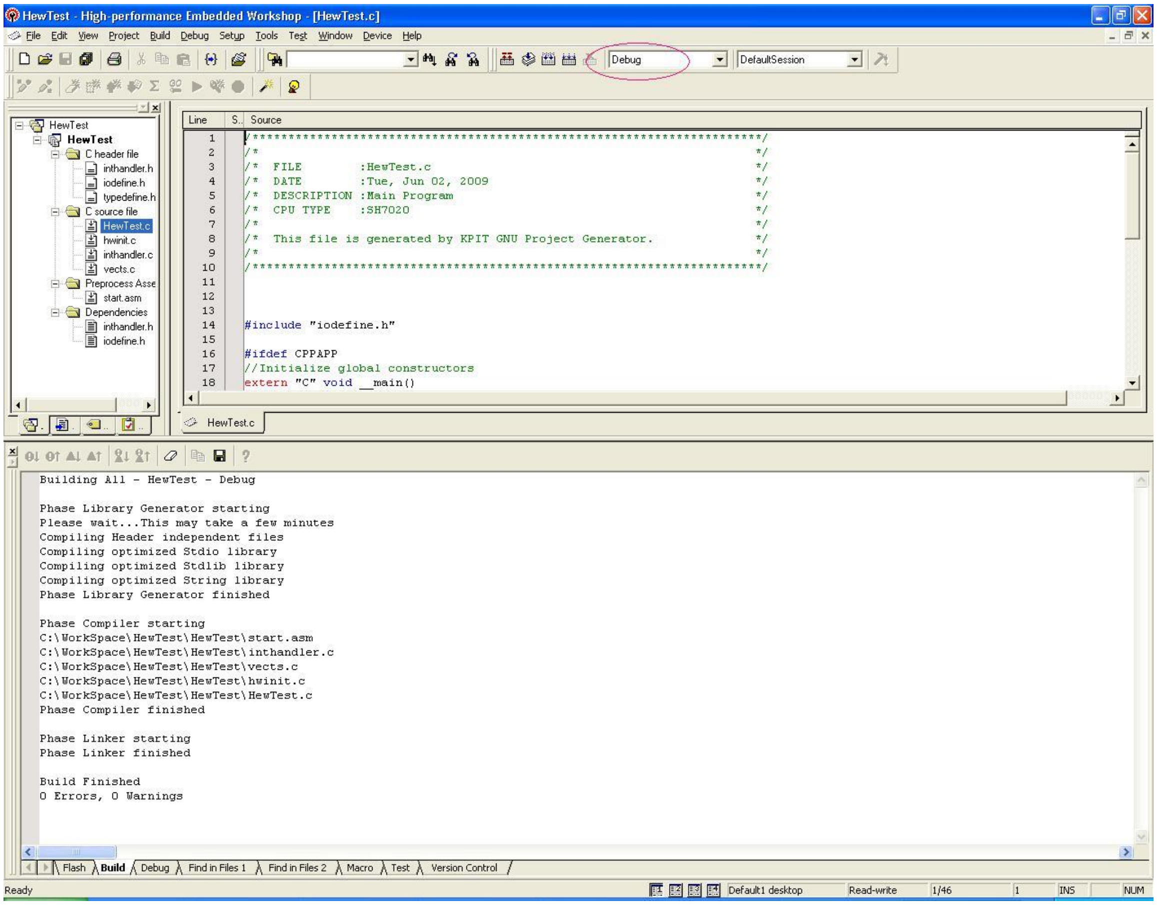


```
/**********************************************************************/
/*                                                                    */
/*  FILE        :HewTest.c                                            */
/*  DATE        :Tue, Jun 02, 2009                                    */
/*  DESCRIPTION :Main Program                                         */
/*  CPU TYPE    :SH7020                                               */
/*                                                                    */
/*  This file is generated by KPIT GNU Project Generator.             */
/*                                                                    */
/**********************************************************************/



#include "iodefine.h"

#ifdef CPPAPP
//Initialize global constructors
extern "C" void __main()
```

```
Building All - HewTest - Debug

Phase Library Generator starting
Please wait...This may take a few minutes
Compiling Header independent files
Compiling optimized Stdio library
Compiling optimized Stdlib library
Compiling optimized String library
Phase Library Generator finished

Phase Compiler starting
C:\WorkSpace\HewTest\HewTest\start.asm
C:\WorkSpace\HewTest\HewTest\inthandler.c
C:\WorkSpace\HewTest\HewTest\vects.c
C:\WorkSpace\HewTest\HewTest\hwinit.c
C:\WorkSpace\HewTest\HewTest\HewTest.c
Phase Compiler finished

Phase Linker starting
Phase Linker finished

Build Finished
0 Errors, 0 Warnings
```

Step 19

Earlier in this tutorial, we very briefly mentioned configuration subfolders. As well as changing compiler, assembler and linker options on a file-by -file or project-wide basis, groups of such options can be grouped together into configurations. By default the project generator creates two configurations: ―Debug‖ and ―Release‖ and you will now observe that the ―Debug‖ configuration was selected when we built the project in the last step (see highlighted area). The default settings for the ―Debug‖ configuration result in debug information being generated by the assembler, compiler and linker, while the default settings for the ―Release‖ configuration omit this information producing code suitable for production. Each configuration can be tailored to your requirements in the compiler; assembler and linker options dialogs and additional configurations can be created and deleted as required using the ―Build Configurations‖ menu item in the ―Options‖ menu on the HEW menu bar.

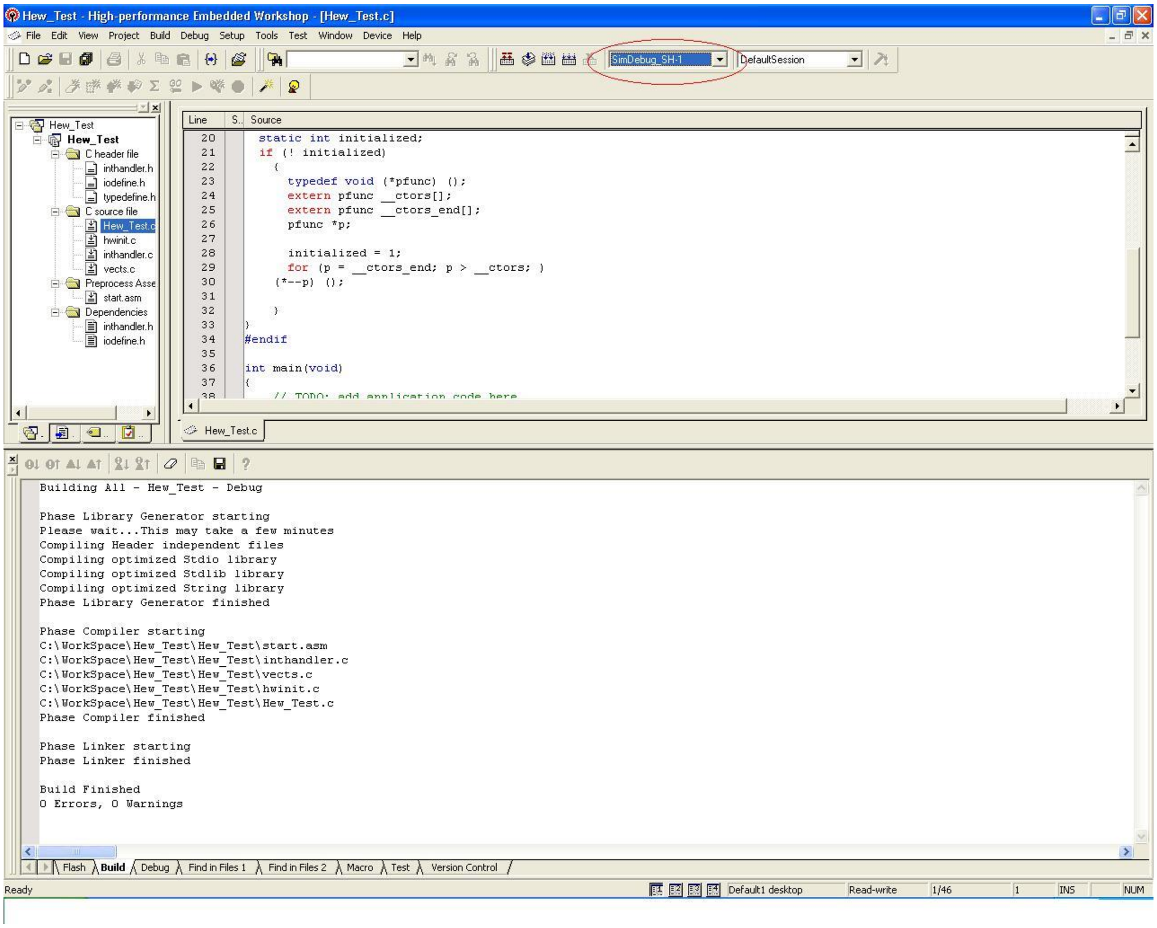


```
/***********************************************************************/
/*                                                                     */
/*  FILE        :HewTest.c                                             */
/*  DATE        :Tue, Jun 02, 2009                                     */
/*  DESCRIPTION :Main Program                                          */
/*  CPU TYPE    :SH7020                                                */
/*                                                                     */
/*  This file is generated by KPIT GNU Project Generator.              */
/*                                                                     */
/***********************************************************************/



#include "iodefine.h"

#ifdef CPPAPP
//Initialize global constructors
extern "C" void __main()
```

```
Building All - HewTest - Debug

Phase Library Generator starting
Please wait...This may take a few minutes
Compiling Header independent files
Compiling optimized Stdio library
Compiling optimized Stdlib library
Compiling optimized String library
Phase Library Generator finished

Phase Compiler starting
C:\WorkSpace\HewTest\HewTest\start.asm
C:\WorkSpace\HewTest\HewTest\inthandler.c
C:\WorkSpace\HewTest\HewTest\vects.c
C:\WorkSpace\HewTest\HewTest\hwinit.c
C:\WorkSpace\HewTest\HewTest\HewTest.c
Phase Compiler finished

Phase Linker starting
Phase Linker finished

Build Finished
0 Errors, 0 Warnings
```

Step 20

If a debugger target has been selected in step 6, a configuration for the selected target is also generated (an abbreviation including the target name). For this tutorial the debugger target is ―SH-1 Simulator‖. Therefore a configuration with name ―SimDebug_SH-1‖ will be generated.



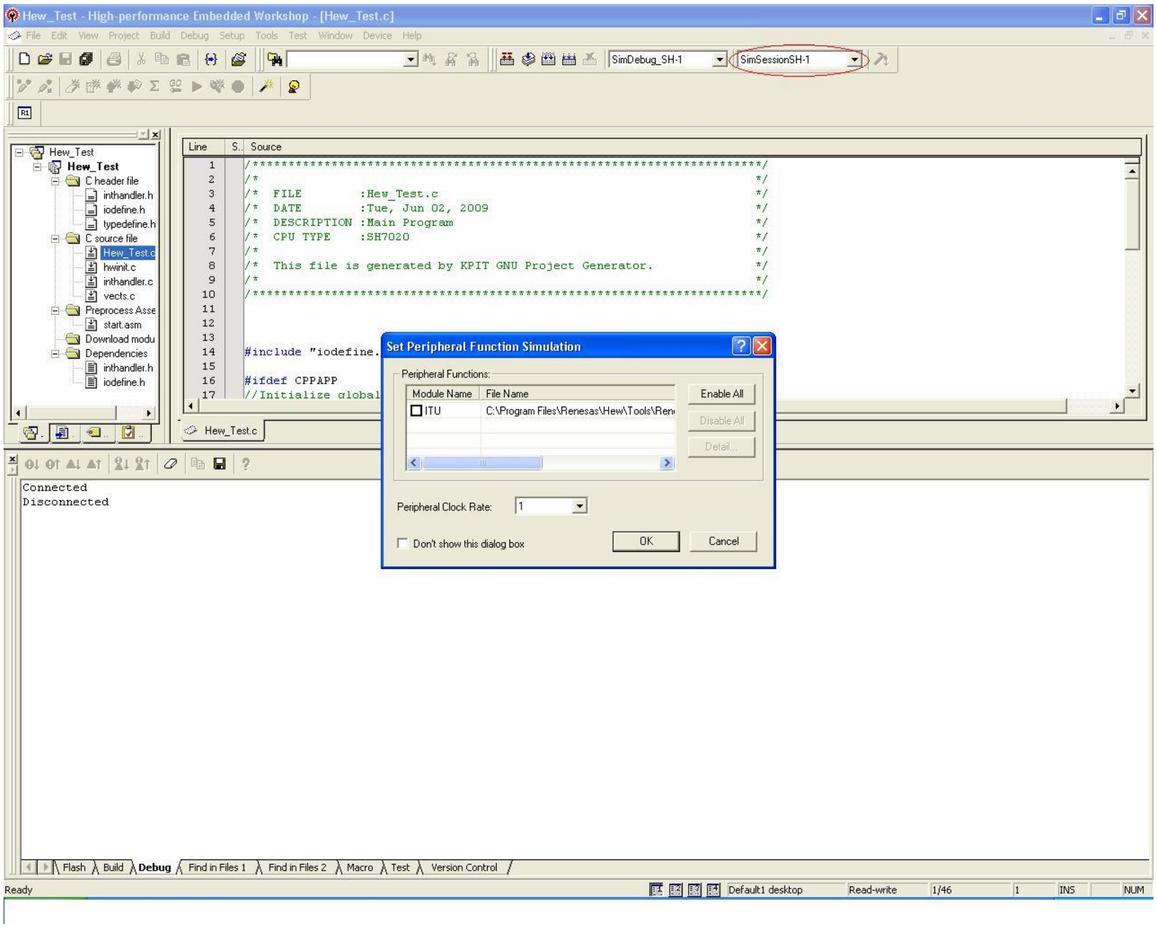
```
  static int initialized;
  if (! initialized)
    {
      typedef void (*pfunc) ();
      extern pfunc __ctors[];
      extern pfunc __ctors_end[];
      pfunc *p;

      initialized = 1;
      for (p = __ctors_end; p > __ctors; )
    (*--p) ();

    }
}
#endif

int main(void)
{
    // TODO: add application code here
```

```
Building All - Hew_Test - Debug

Phase Library Generator starting
Please wait...This may take a few minutes
Compiling Header independent files
Compiling optimized Stdio library
Compiling optimized Stdlib library
Compiling optimized String library
Phase Library Generator finished

Phase Compiler starting
C:\WorkSpace\Hew_Test\Hew_Test\start.asm
C:\WorkSpace\Hew_Test\Hew_Test\inthandler.c
C:\WorkSpace\Hew_Test\Hew_Test\vects.c
C:\WorkSpace\Hew_Test\Hew_Test\hwinit.c
C:\WorkSpace\Hew_Test\Hew_Test\Hew_Test.c
Phase Compiler finished

Phase Linker starting
Phase Linker finished

Build Finished
0 Errors, 0 Warnings
```

Step 21

When a project is created after selection of the toolchain, the HEW automatically creates session "DefaultSession". If you have selected a target debugging platform at creation of the project (Step 6), a session is automatically created for connecting the HEW with the selected debugging platform. For example, here we have selected "SH-1 Simulator" in "Target" at creation of a project, session "SimSessionSH-1" is automatically created.

Select the ¯SimSessionSH-1‖ session as shown in the figure below. ¯Set I/O DLL‖ window will appear. Click ¯OK‖.



```
/**********************************************************************/
/*                                                                    */
/*  FILE        :Hew_Test.c                                           */
/*  DATE        :Tue, Jun 02, 2009                                    */
/*  DESCRIPTION :Main Program                                         */
/*  CPU TYPE    :SH7020                                               */
/*                                                                    */
/*  This file is generated by KPIT GNU Project Generator.             */
/*                                                                    */
/**********************************************************************/



#include "iodefine.

#ifdef CPPAPP
//Initialize global
```

Set Peripheral Function Simulation

Peripheral Functions:

Module Name	File Name
□ ITU	C:\Program Files\Renesas\Hew\Tools\Ren

Enable All | Disable All | Detail...

Peripheral Clock Rate: 1

□ Don't show this dialog box

OK | Cancel

```
Connected
Disconnected
```

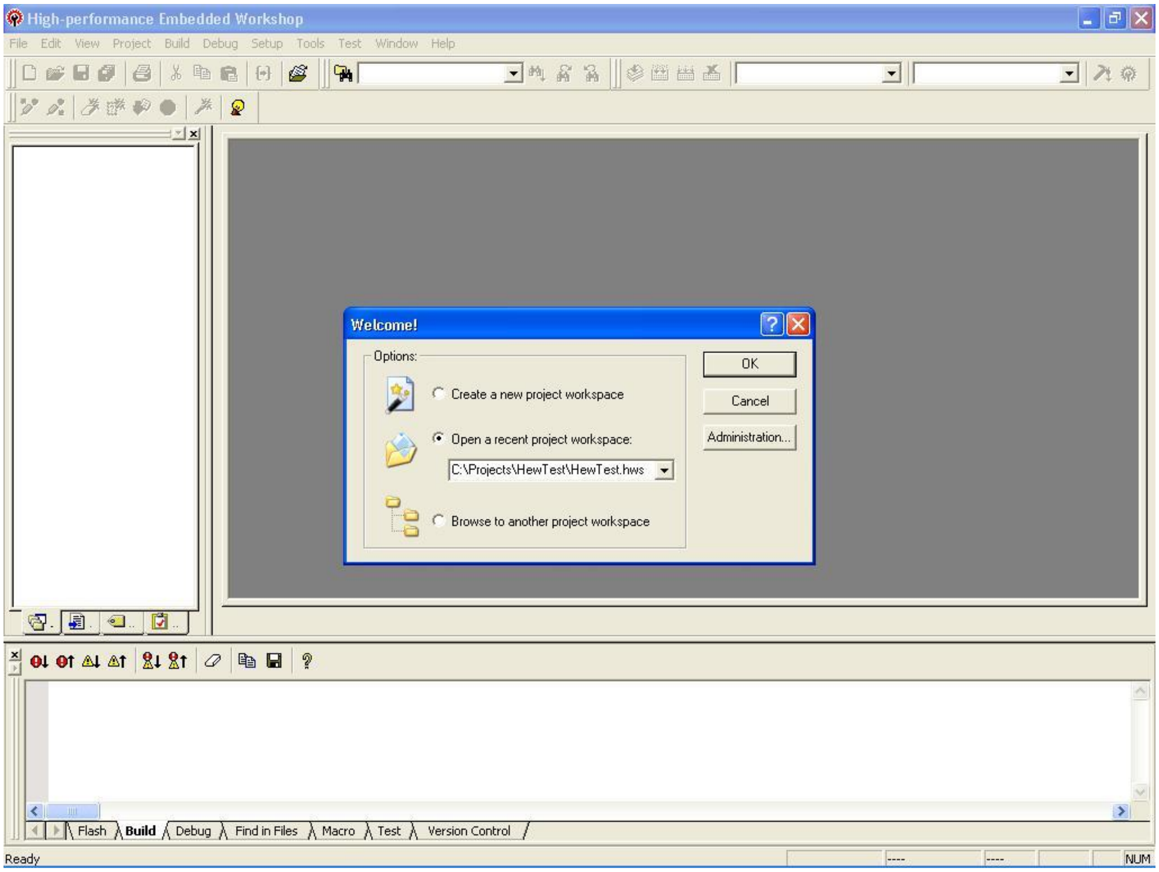
The ¯SimSessionSH-1‖ session has all the debug settings. User just has to Build the application and download the ¯.x‖ file. Details on ¯Debugging the Project with the HEW Debugger‖ are covered in next section.

2.6.3 Running and Debugging the Project with the HEW Simulator/Debugger

This section of the tutorial explains how to execute/debug projects using HEW‘s integrated debugger.

Step 1

If you haven‘t done so already, launch HEW and load the tutorial project workspace from ¯C:\Projects\HewTest‖. Ensure the ¯Debug‖ configuration is selected and the project has been built (see 2.7.2: Creating and Building a Sample Project Using HEW for full details).


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Help
Flash Build Debug Find in Files Macro Test Version Control
Ready
NUM

Welcome!

Options:

○ Create a new project workspace

● Open a recent project workspace:

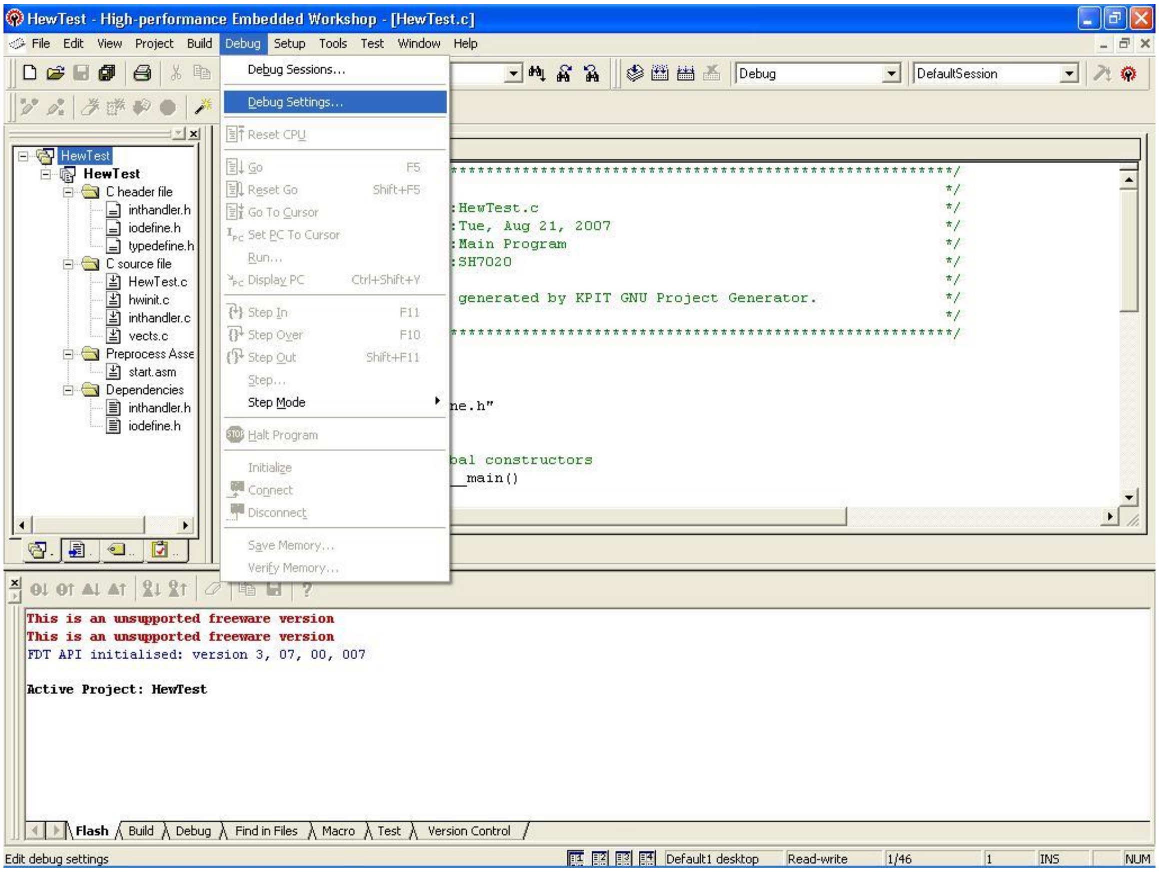
C:\Projects\HewTest\HewTest.hws

○ Browse to another project workspace

OK
Cancel
Administration...

Step 2

Select the ¯Debug Settings‖ item from the ¯Debug‖ menu on the menu bar.

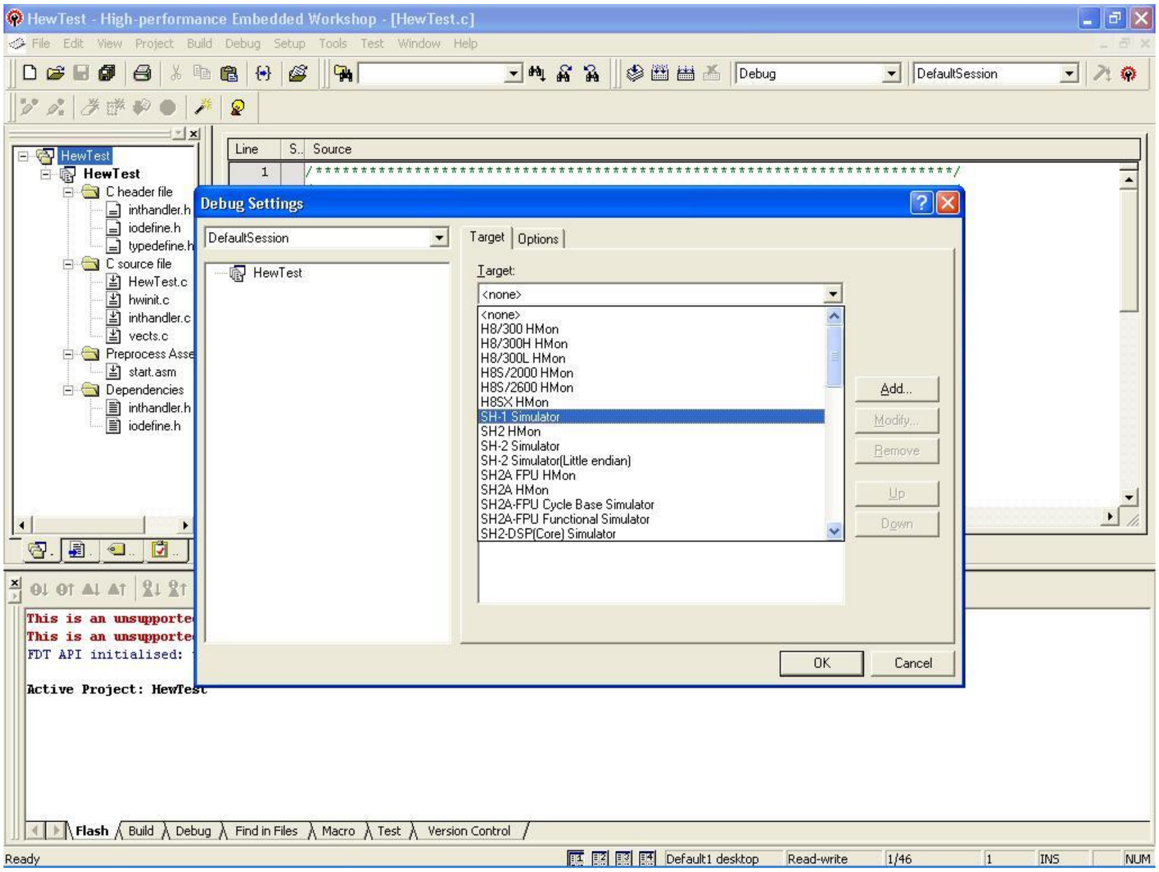


```
***************************************************************/
                                                              */
:HewTest.c                                                    */
:Tue, Aug 21, 2007                                            */
:Main Program                                                 */
:SH7020                                                       */
                                                              */
 generated by KPIT GNU Project Generator.                     */
                                                              */
***************************************************************/

ne.h"

bal constructors
__main()
```

```
This is an unsupported freeware version
This is an unsupported freeware version
FDT API initialised: version 3, 07, 00, 007

Active Project: HewTest
```

Step 3

In the ¯Debug Settings‖ dialog box that appears, select the ¯Target‖. For simplicity, we will use a simulator rather than actual hardware. As our project was built for a sample SH-1 device, the SH-1 simulator should be selected (¯SH-1 Simulator‖).



Debug Settings

DefaultSession

Target:

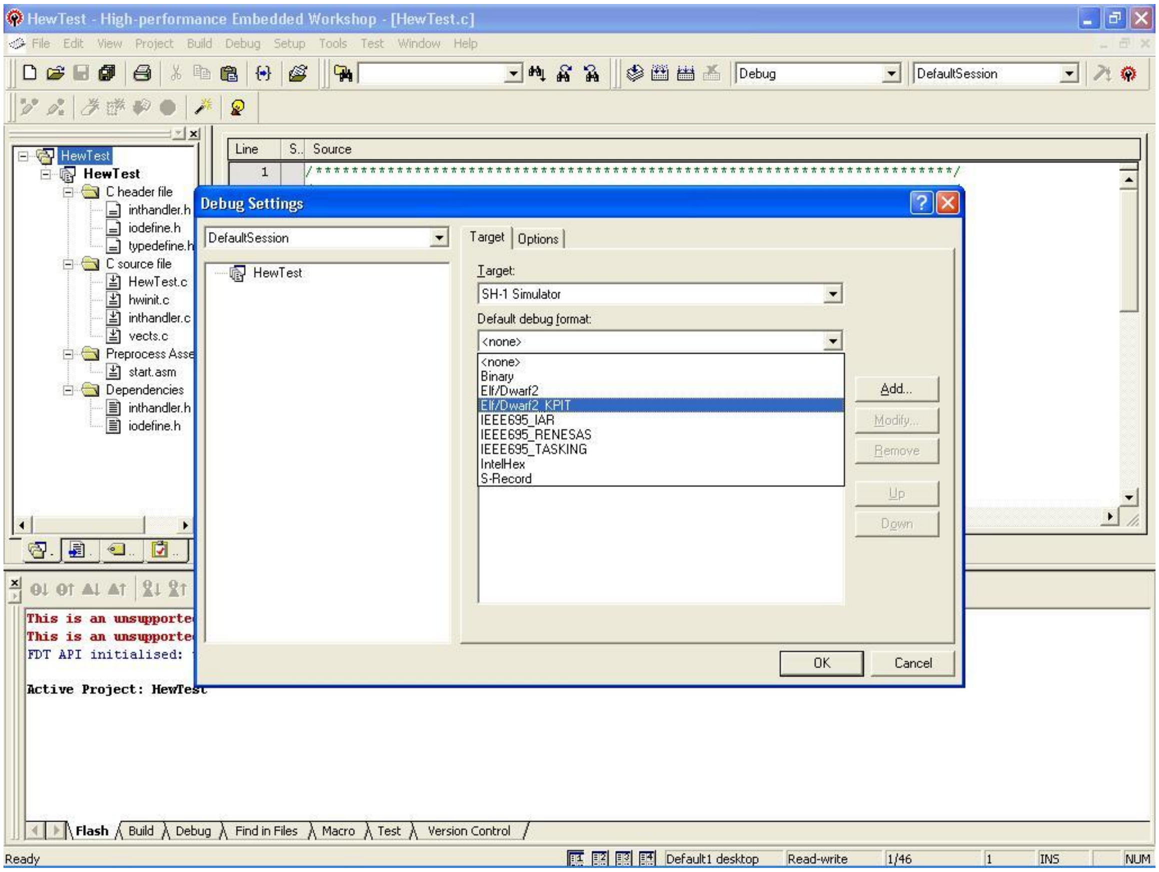
<none>

- <none>
- H8/300 HMon
- H8/300H HMon
- H8/300L HMon
- H8S/2000 HMon
- H8S/2600 HMon
- H8SX HMon
- SH-1 Simulator
- SH2 HMon
- SH-2 Simulator
- SH-2 Simulator(Little endian)
- SH2A FPU HMon
- SH2A HMon
- SH2A-FPU Cycle Base Simulator
- SH2A-FPU Functional Simulator
- SH2-DSP(Core) Simulator

```
This is an unsupporte
This is an unsupporte
FDT API initialised: 

Active Project: HewTest
```

Step 4

Now select the ¯Default Debug Format‖. Be sure to select the ¯Elf/Dwarf2 for KPIT‖ option, as any other selection including ¯Elf/Dwarf2‖ is incorrect and may cause HEW to become unstable or read the debug information in your project file incorrectly.



HewTest - High-performance Embedded Workshop - [HewTest.c]

Debug Settings

DefaultSession

Target:
SH-1 Simulator

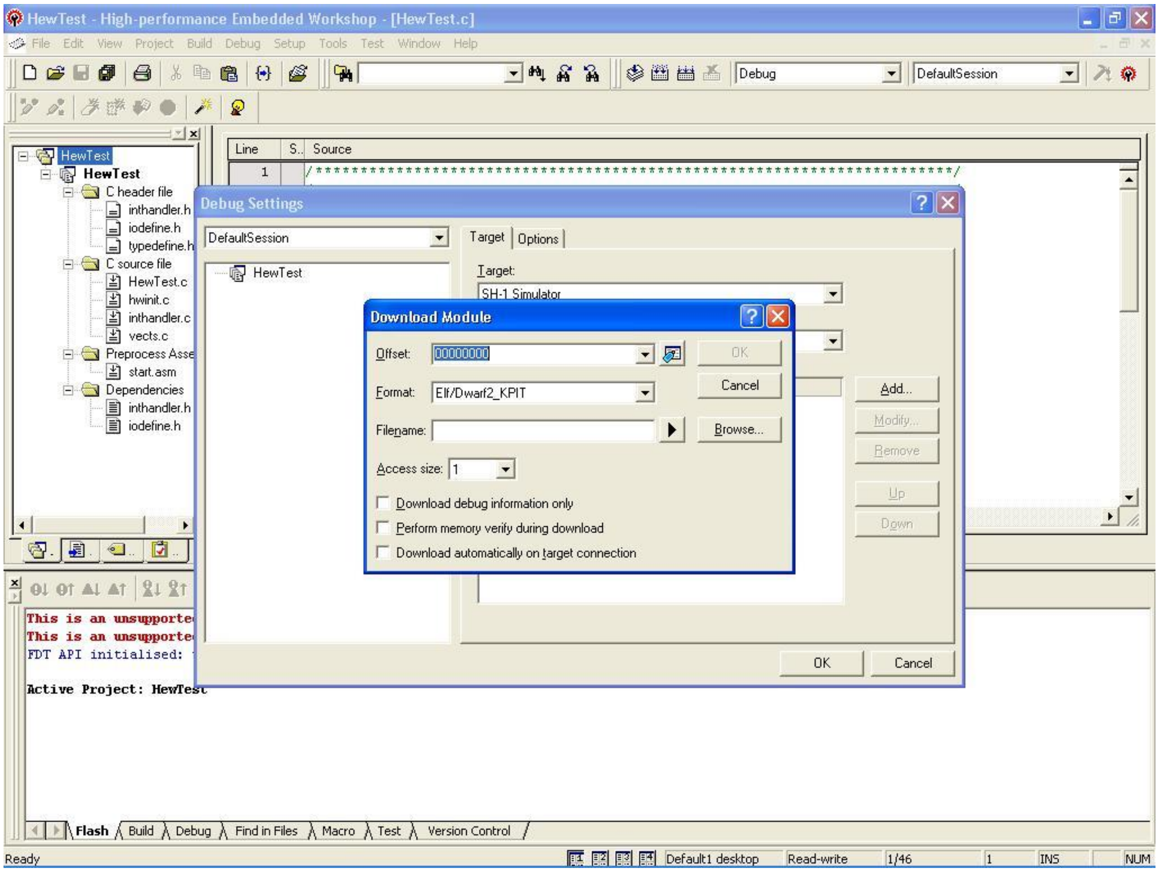
Default debug format:
<none>

<none>
Binary
Elf/Dwarf2
Elf/Dwarf2_KPIT
IEEE695_IAR
IEEE695_RENESAS
IEEE695_TASKING
IntelHex
S-Record

```
This is an unsupporte
This is an unsupporte
FDT API initialised:

Active Project: HewTest
```

Step 5

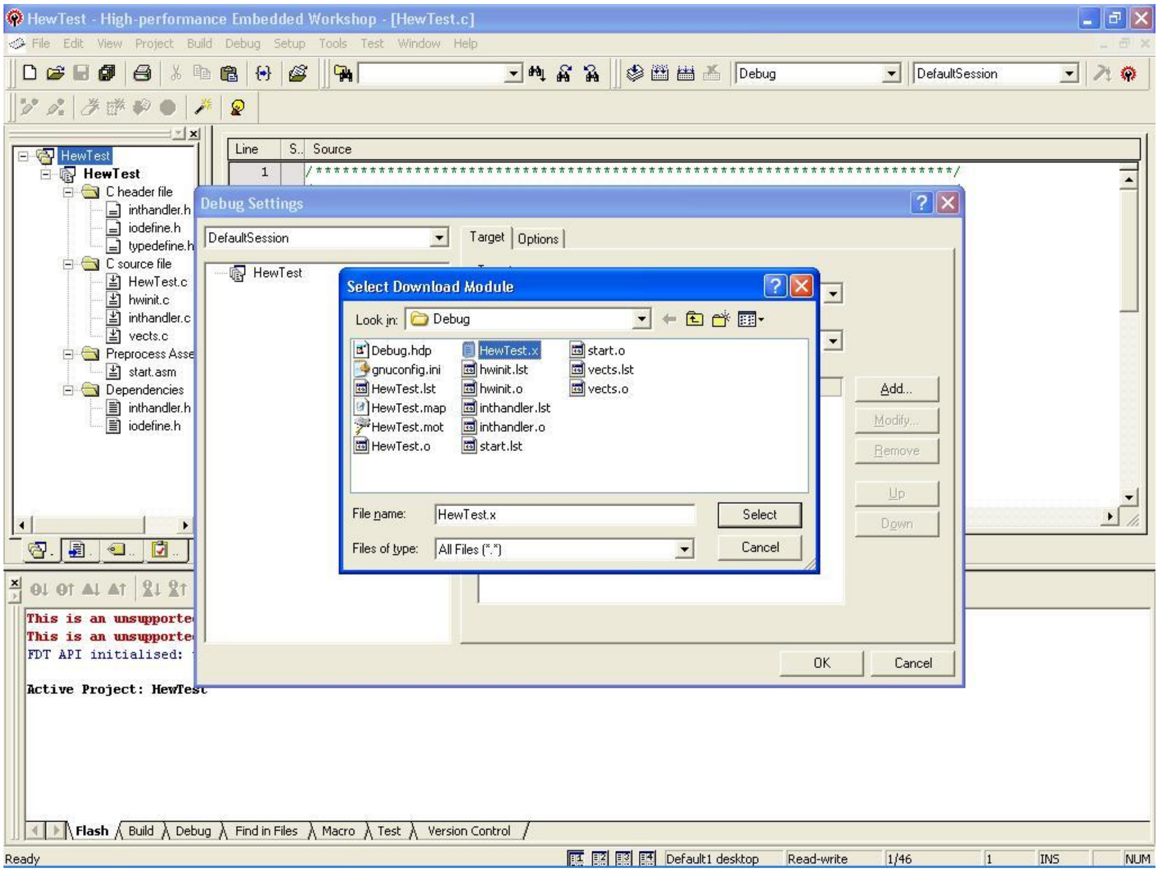
Next we will inform HEW of the application we wish to debug. Click the ¯Add‖ button on the ¯Debug Settings‖ dialog box. This will cause a ¯Download Module‖ dialog box to be displayed.



```
This is an unsupporte
This is an unsupporte
FDT API initialised:

Active Project: HewTest
```

Step 6

Click the ¯Browse‖ button on the ¯Download Module‖ dialog box and a file selection dialog box will appear. Use this to navigate to the ¯C:\Projects\HewTest\HewTest\Debug\‖ folder. In this folder, select the file ¯HewTest.x‖ and then click the ¯Select‖ button to confirm the selection.



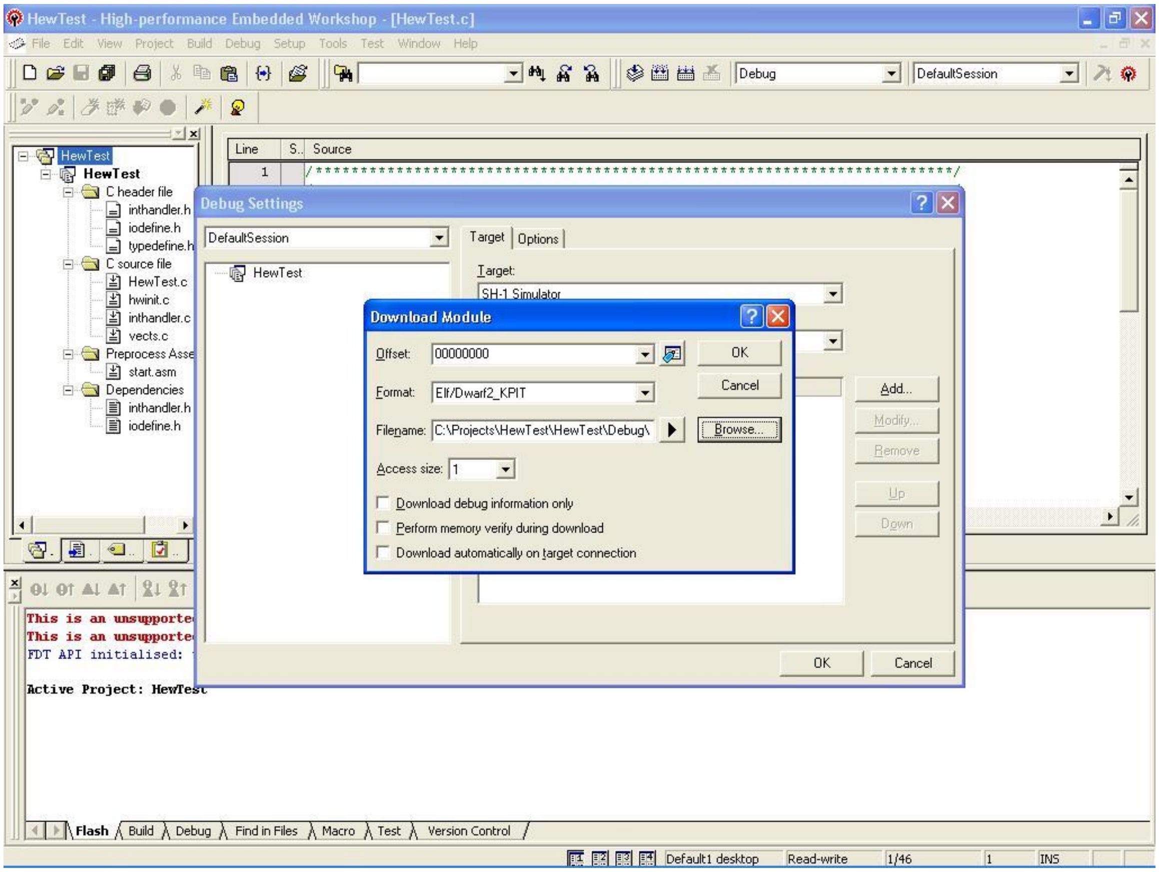
File name: HewTest.x

Files of type: All Files (*.*)

```
This is an unsupporte
This is an unsupporte
FDT API initialised:

Active Project: HewTest
```

Step 7

HEW returns you to the ¯Download Module‖ dialog box; ensure that the ¯Format‖ option is set to ¯Elf/Dwarf2 for KPIT‖ option. As before, be sure to select the ¯Elf/Dwarf2 for KPIT‖ option, as any other selection including ¯Elf/Dwarf2‖ is incorrect and may cause HEW to become unstable or read the debug information in your project file incorrectly. Ensure that the ¯Offset‖ field is at its default value of H'00000000 and the ¯Download debug information only‖ option is off (again, this is its default value). Now click the ¯OK‖ button on the ¯Download Module‖ dialog box.



Debug Settings: DefaultSession; Target: SH-1 Simulator

Download Module

Offset: 00000000

Format: Elf/Dwarf2_KPIT

Filename: C:\Projects\HewTest\HewTest\Debug\

Access size: 1

□ Download debug information only

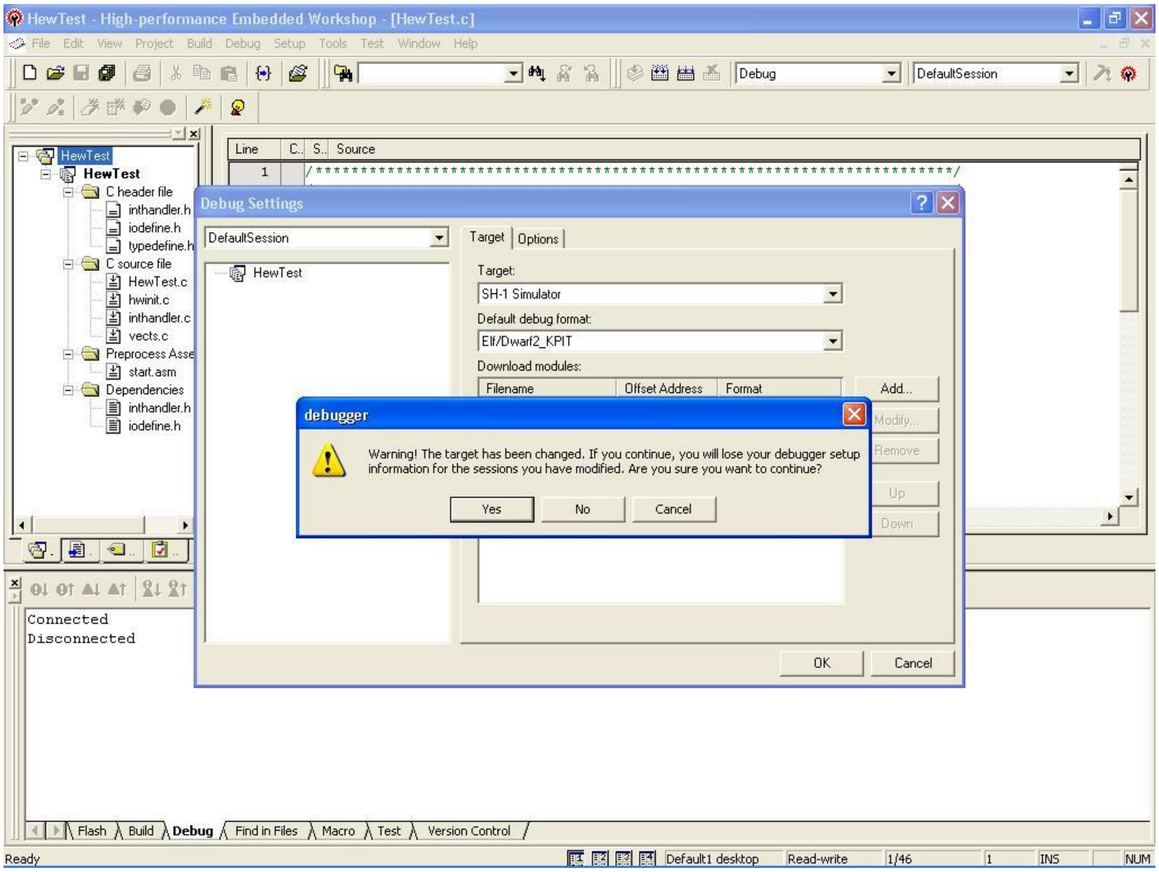
□ Perform memory verify during download

□ Download automatically on target connection

```
This is an unsupporte
This is an unsupporte
FDT API initialised:

Active Project: HewTest
```

Step 8

Click ¯OK‖ on the ¯Debug Settings‖ dialog box. The warning will be displayed. Click ¯Yes‖. ¯Set I/O DLL‖ window will appear (same as window shown in **step 21** of 2.9.2 section). Click ¯OK‖. HEW now loads various debug components. Note that numbers of new toolbars are added to the user interface.



HewTest - High-performance Embedded Workshop - [HewTest.c]

Debug Settings

DefaultSession

Target: SH-1 Simulator

Default debug format: Elf/Dwarf2_KPIT

Download modules:

Filename	Offset Address	Format

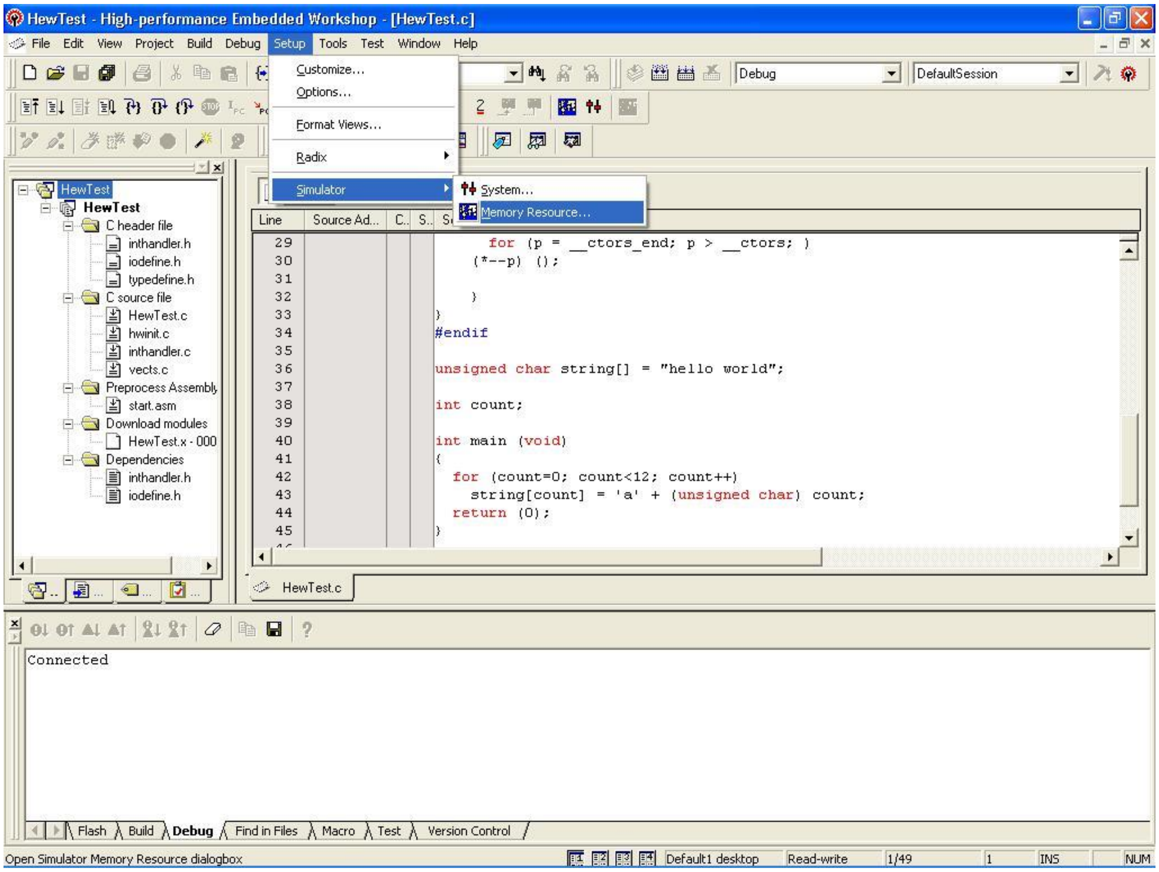
debugger

Warning! The target has been changed. If you continue, you will lose your debugger setup information for the sessions you have modified. Are you sure you want to continue?

```
Connected
Disconnected
```

Step 9

Although the SH-1 simulator is configured with an address space typical of this device family, it is necessary to specify the exact memory existing in simulated device. To specify this memory, first select the ¯Simulator‖ item from the ¯Setup‖ menu on the menu bar. In the sub-menu that appears, click on ¯Memory Resource‖.



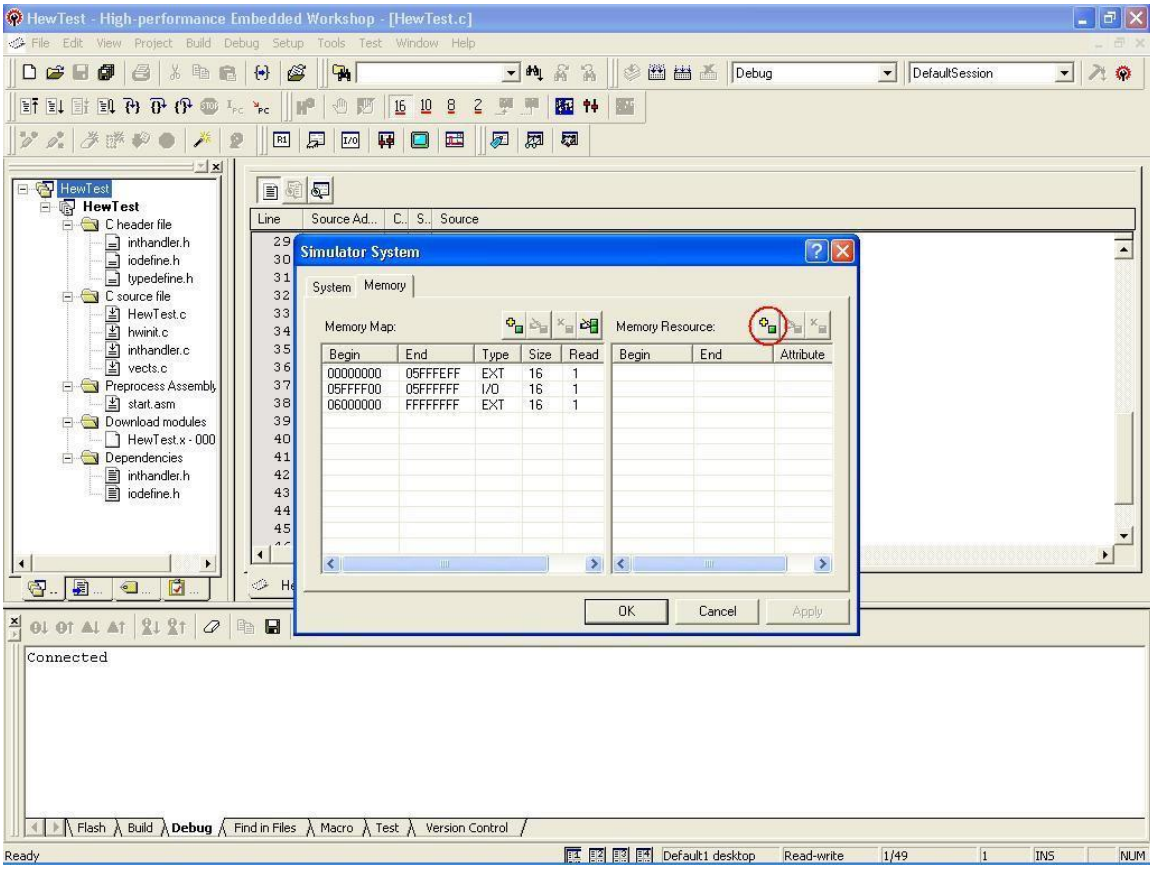
```
      for (p = __ctors_end; p > __ctors; )
    (*--p) ();

    }
}
#endif

unsigned char string[] = "hello world";

int count;

int main (void)
{
  for (count=0; count<12; count++)
    string[count] = 'a' + (unsigned char) count;
  return (0);
}
```

```
Connected
```

Open Simulator Memory Resource dialogbox

Step 10

HEW now presents a ¯Simulator System‖ dialog box. Click on the ¯Add‖ button as we will be adding a memory area to the simulated target.



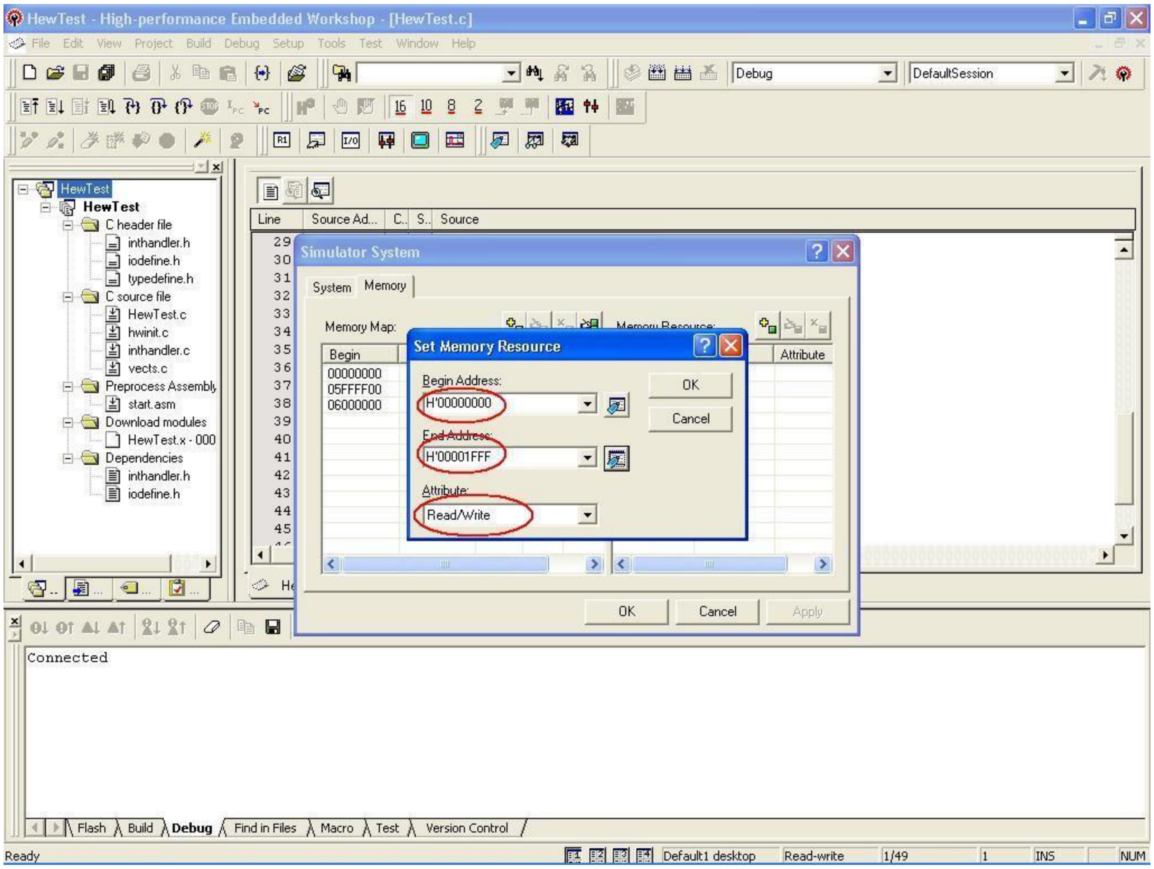
Memory Map:

Begin	End	Type	Size	Read
00000000	05FFFEFF	EXT	16	1
05FFFF00	05FFFFFF	I/O	16	1
06000000	FFFFFFFF	EXT	16	1

```
Connected
```

Step 11

In the ¯System Memory Resource Modify‖ dialog box, set the ¯Start address‖ to H'00000000 and the ¯End address‖ to H'00001FFF. The access type should be set to ¯Read/Write‖. When done, click the ¯OK‖ button.



HewTest - High-performance Embedded Workshop - [HewTest.c]

Simulator System — Memory

Memory Map: Begin 00000000, 05FFFF00, 06000000

Set Memory Resource

Begin Address: H'00000000

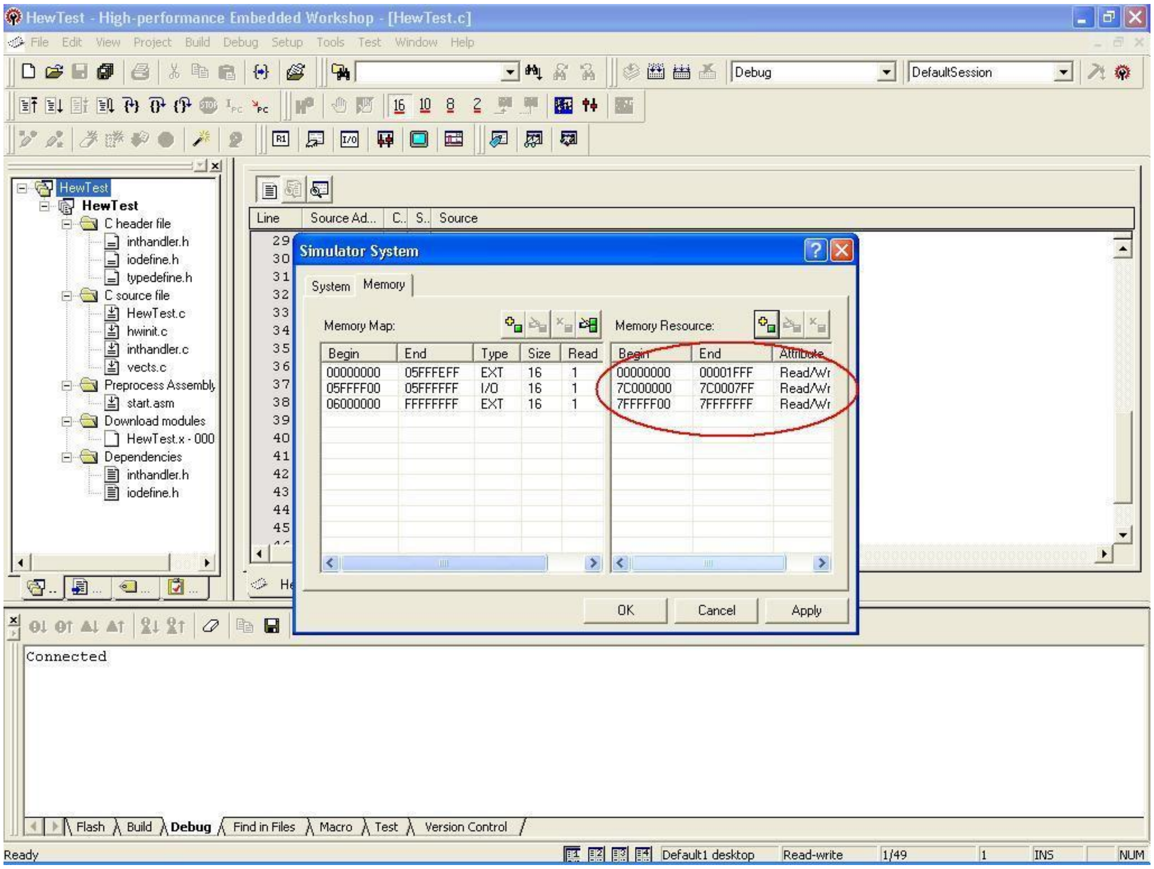
End Address: H'00001FFF

Attribute: Read/Write

```
Connected
```

Step 12

Similarly add the resources starting from H'7C000000 to H'7C0007FF and from H'7FFFFF00 to H'7FFFFFFF. The access type should be set to ¯Read/Write‖. When done, click the ¯OK‖ on ¯Set Memory Resource‖ and then click ¯OK‖ on ¯Simulator System‖ dialog box.



Memory Map:

Begin	End	Type	Size	Read
00000000	05FFFEFF	EXT	16	1
05FFFF00	05FFFFFF	I/O	16	1
06000000	FFFFFFFF	EXT	16	1

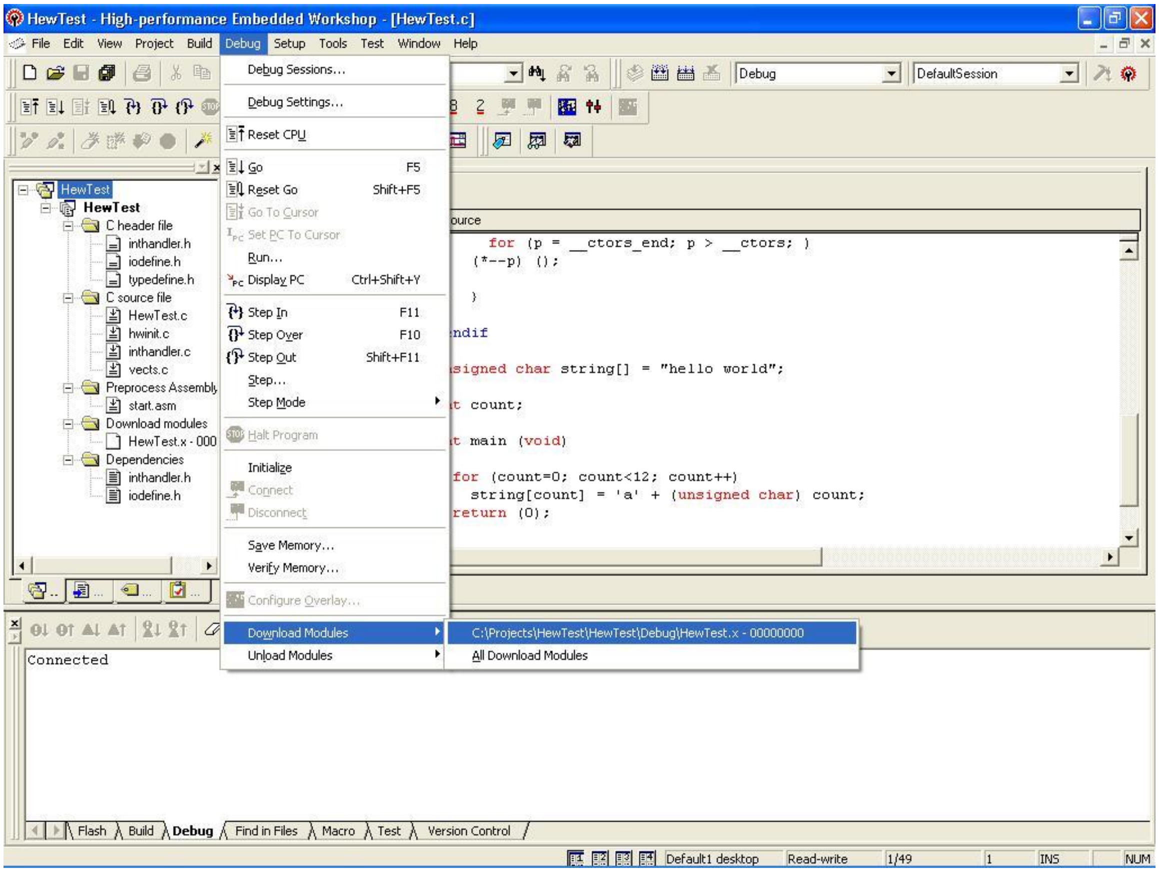
Memory Resource:

Begin	End	Attribute
00000000	00001FFF	Read/Wr
7C000000	7C0007FF	Read/Wr
7FFFFF00	7FFFFFFF	Read/Wr

```
Connected
```

Step 13

Now we will download our tutorial application into the simulated target‘s memory. Select the ¯Download Modules‖ item from the ¯Debug‖ menu on the menu bar. In the sub-menu that is displayed, select the option ¯C:\Projects\HewTest\HewTest\Debug\HewTest.x - H'00000000‖. HEW will now download the tutorial program into the simulator‘s memory. This includes both ROM and RAM sections; both code and data. If any error relating to invalid addresses appears at this stage, be sure you configured the memory map correctly. If no messages are reported, the download was successful.



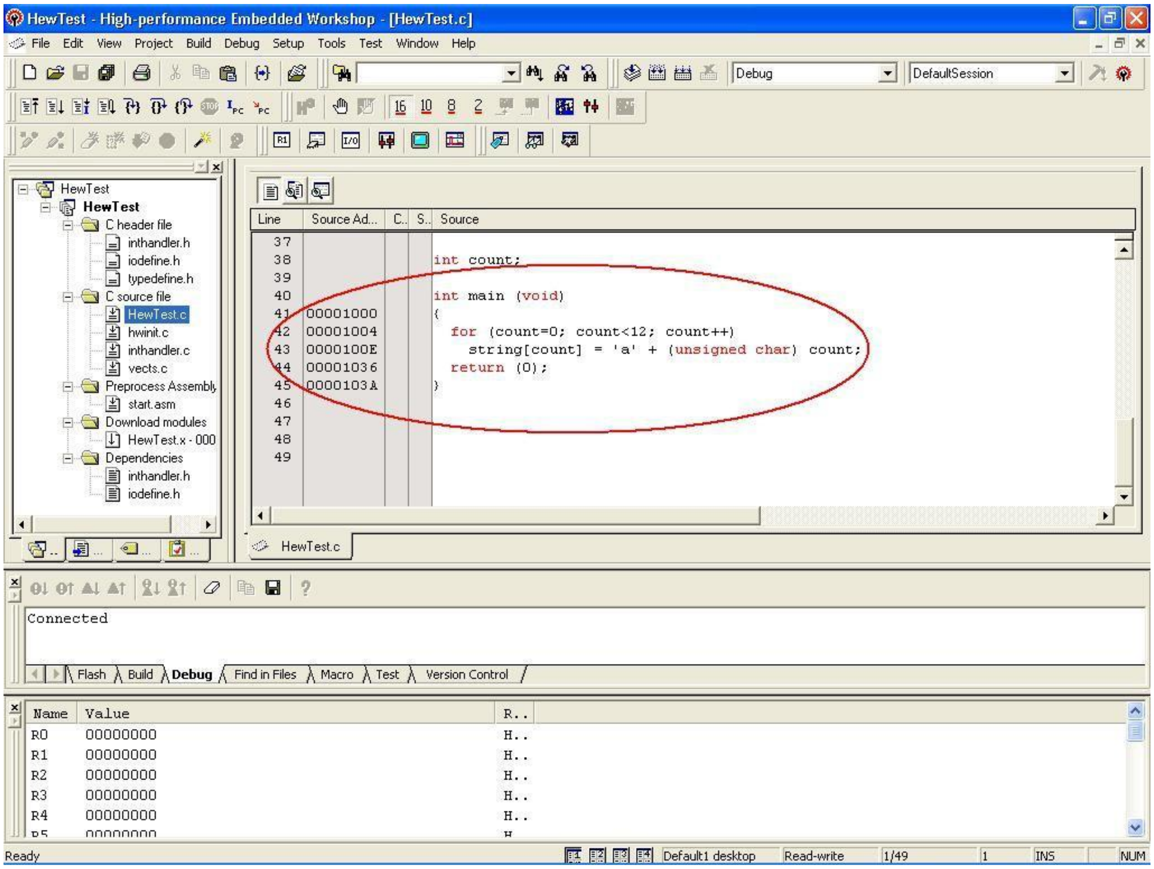
```
  for (p = __ctors_end; p > __ctors; )
(*--p) ();

}

ndif

signed char string[] = "hello world";

t count;

t main (void)

for (count=0; count<12; count++)
  string[count] = 'a' + (unsigned char) count;
return (0);
```

```
Connected
```

[*Note*: In HEW there are a number of ways to download the tutorial application. Selecting ¯Build All‖ item from the ¯Build‖ menu on the menu bar will give you the option of downloading the created module. Alternatively, right clicking the ¯HewTest.x‖ entry in the project files view will display a context menu from which ¯Download module‖ can be selected.]

Step 14

Now our tutorial program is loaded into the target‘s memory and the associated debugging information has been loaded into the HEW debugger, we can debug the application at the source code level. To do this, we must first display one of the application source files. Double click the file ―HewTest.c‖. By default this is located on the left hand side of the HEW window and it contains blank main() function. Here we have added small code in function main() for debugging.



```
                int count;

                int main (void)
00001000        {
00001004          for (count=0; count<12; count++)
0000100E            string[count] = 'a' + (unsigned char) count;
00001036          return (0);
0000103A        }
```

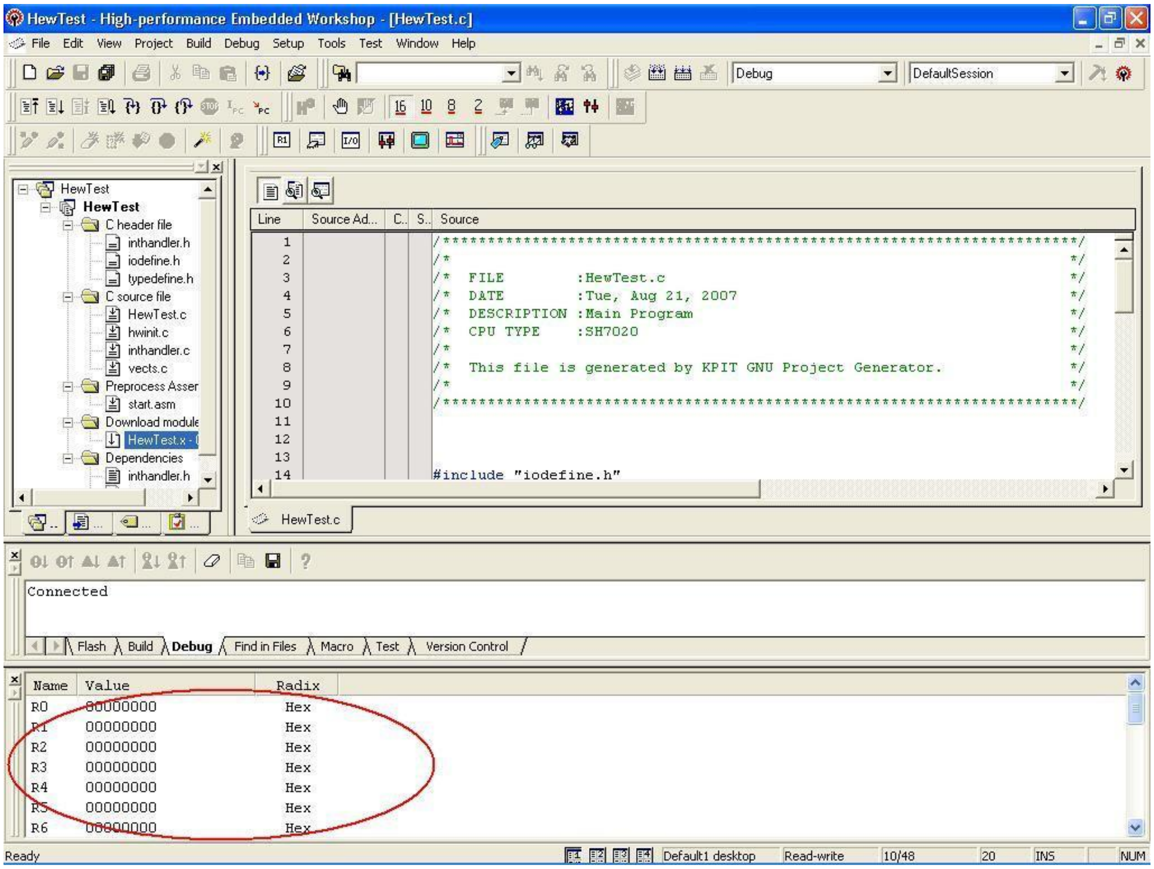
```
Connected
```

```
Name  Value       R..
R0    00000000    H..
R1    00000000    H..
R2    00000000    H..
R3    00000000    H..
R4    00000000    H..
R5    00000000    H
```

Step 15

HEW now displays the file main.c in the editor window. Notice that assembly labels are displayed against appropriate lines and linked address information is shown against each line of source code. Now add some extra windows to give more information about the state of the target device during program execution.

Select the ¯CPU‖ item from the ¯View‖ menu on the menu bar. Now select ¯Registers‖ from the sub-menu that is displayed. A frame showing the target‘s register values is added to the bottom of the HEW window.



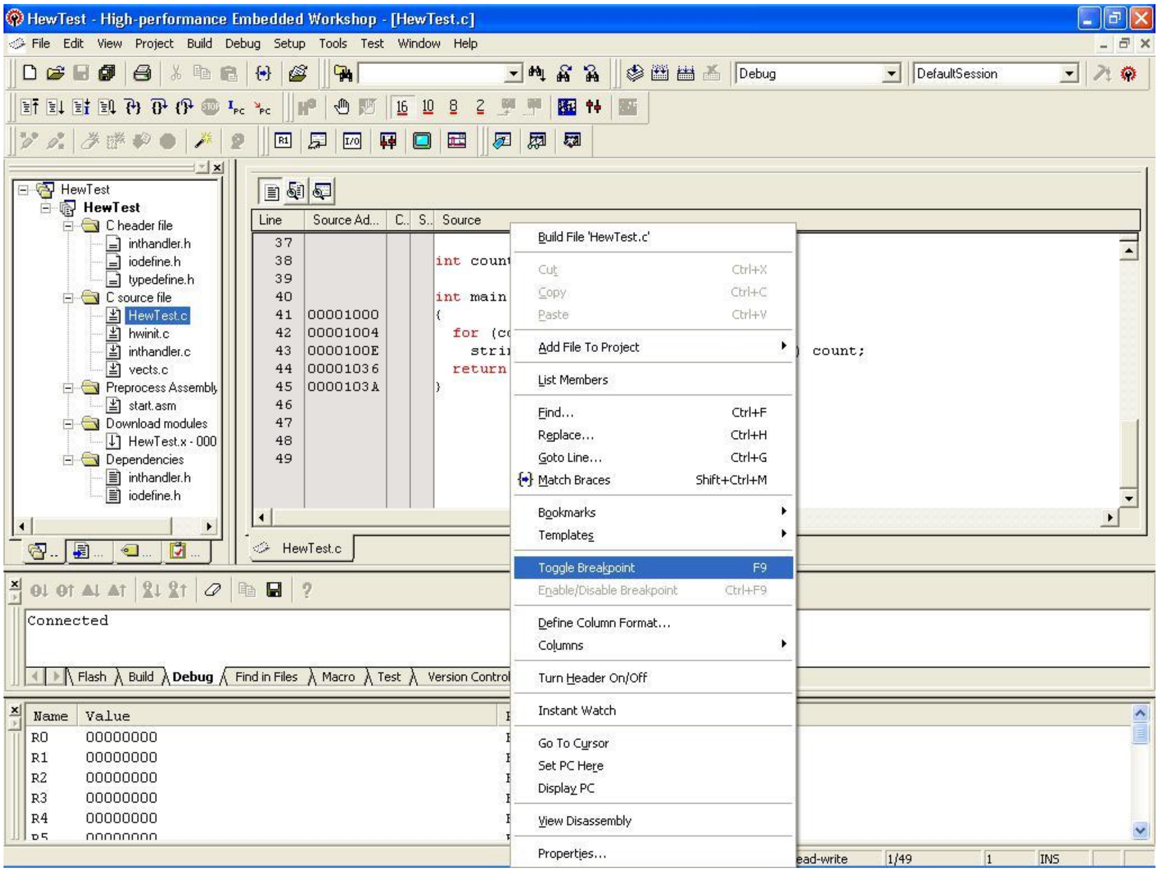
```
/*************************************************************************/
/*                                                                       */
/*  FILE        :HewTest.c                                               */
/*  DATE        :Tue, Aug 21, 2007                                       */
/*  DESCRIPTION :Main Program                                            */
/*  CPU TYPE    :SH7020                                                  */
/*                                                                       */
/*  This file is generated by KPIT GNU Project Generator.                */
/*                                                                       */
/*************************************************************************/



#include "iodefine.h"
```

```
Connected
```

Name	Value	Radix
R0	00000000	Hex
R1	00000000	Hex
R2	00000000	Hex
R3	00000000	Hex
R4	00000000	Hex
R5	00000000	Hex
R6	00000000	Hex

Step 16

Add a breakpoint to the body of the for(...) loop in the main() function. Position the mouse pointer on source line 43, which starts ¯string[count]...‖ (see illustration) and click the right mouse button. In the context menu that appears, select the ¯Toggle Breakpoint‖ item.



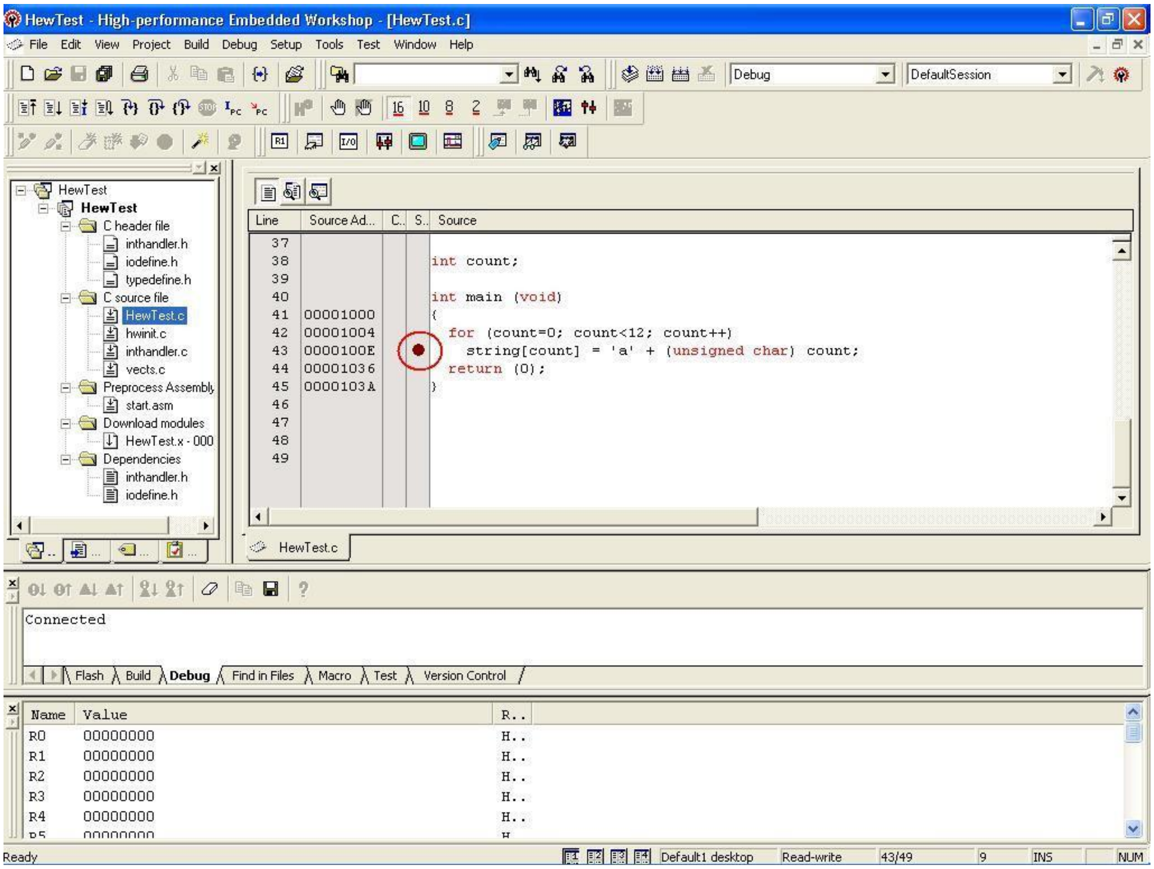
```
Line  Source Ad...  C.. S.. Source
 37
 38                         int count
 39
 40                         int main
 41   00001000              {
 42   00001004                for (c
 43   0000100E                  stri                 ) count;
 44   00001036                return
 45   0000103A              }
 46
 47
 48
 49
```

```
Connected
```

```
Name  Value
R0    00000000
R1    00000000
R2    00000000
R3    00000000
R4    00000000
R5    00000000
```

Step 17

Observe that a red dot appears in the column closest to the program code on line 43 indicating that the breakpoint has been set.



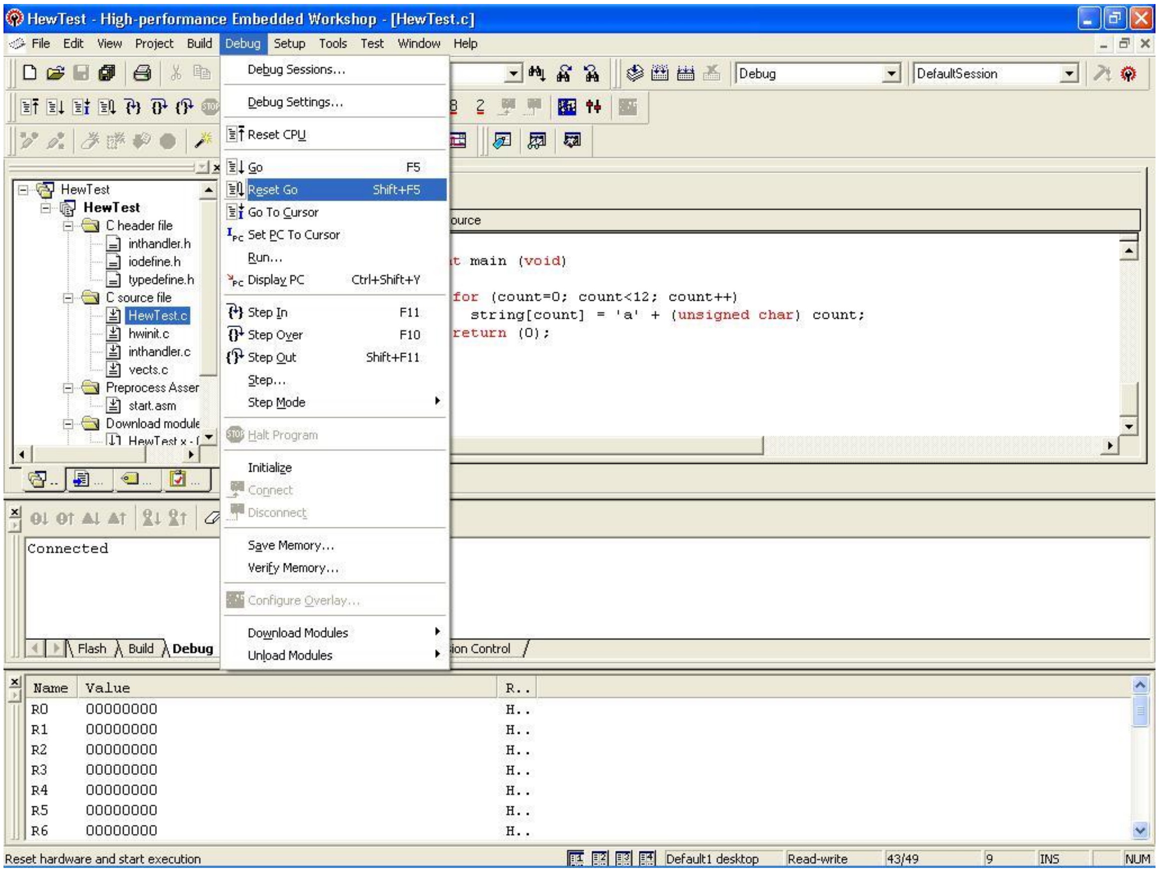
```
37
38                          int count;
39
40                          int main (void)
41  00001000                {
42  00001004                  for (count=0; count<12; count++)
43  0000100E      ●             string[count] = 'a' + (unsigned char) count;
44  00001036                  return (0);
45  0000103A                }
46
47
48
49
```

```
Connected
```

```
Name   Value                                                  R..
R0     00000000                                               H..
R1     00000000                                               H..
R2     00000000                                               H..
R3     00000000                                               H..
R4     00000000                                               H..
R5     00000000                                               H
```

Step 18

With the breakpoint set, we can start execution. Select the ―Reset Go‖ item from the ―Debug‖ menu on the menu bar. This will reset the CPU and then continue executing the code until the breakpoint.



```
t main (void)

 for (count=0; count<12; count++)
     string[count] = 'a' + (unsigned char) count;
 return (0);
```

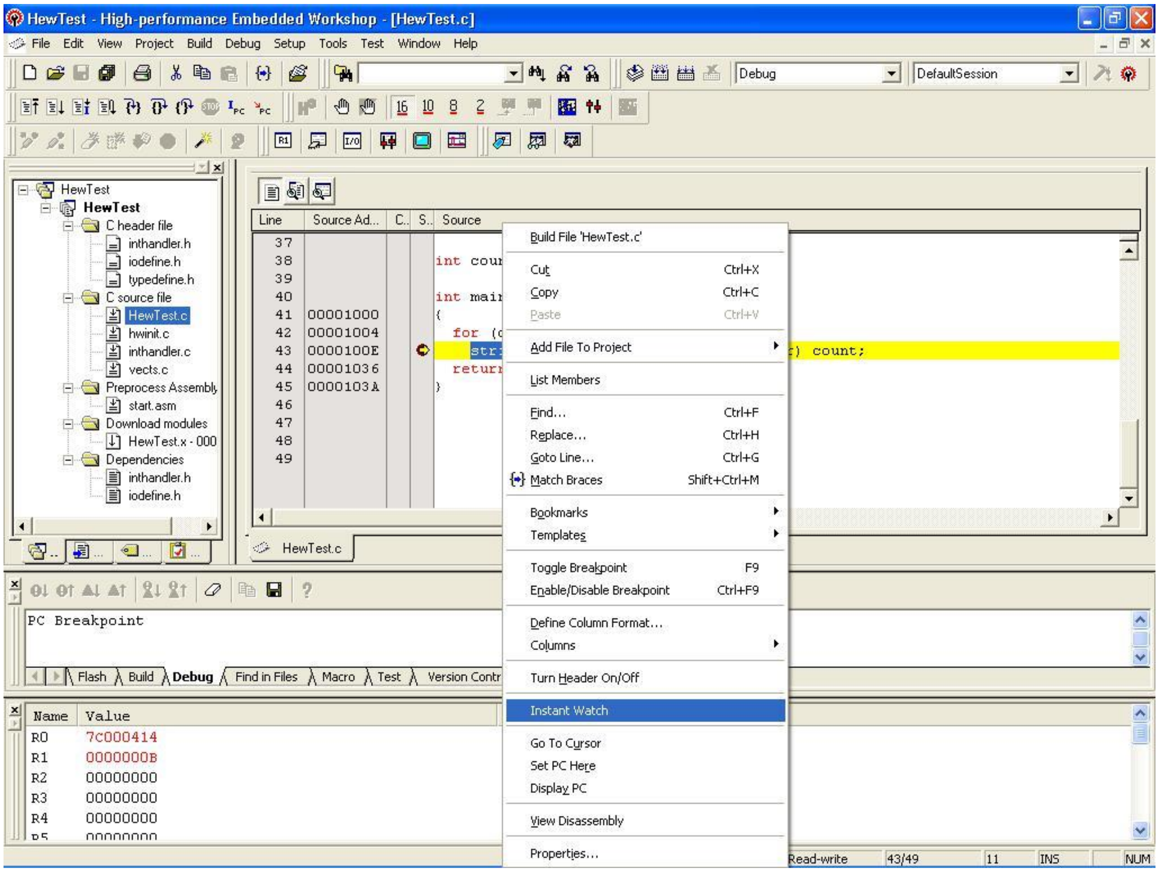
```
Connected
```

```
Name  Value     R..
R0    00000000  H..
R1    00000000  H..
R2    00000000  H..
R3    00000000  H..
R4    00000000  H..
R5    00000000  H..
R6    00000000  H..
```

Reset hardware and start execution

Step 19

The yellow arrow in the editor window over the red breakpoint dot indicates that the target‘s program counter is positioned at an area of program code corresponding to the indicated line of source code and that the breakpoint has interrupted program execution. The sample program in main() simply initializes a string in RAM to ―hello world\n‖ and then changes each letter in turn until the string is changed to ―abcdefghijkl‖. Watch the string variable in RAM to see this happen character by character. Position the mouse over the ―string‖ variable on line 43 in the editor window and click the right mouse button to bring up a context menu. From this select ―Instant Watch‖.



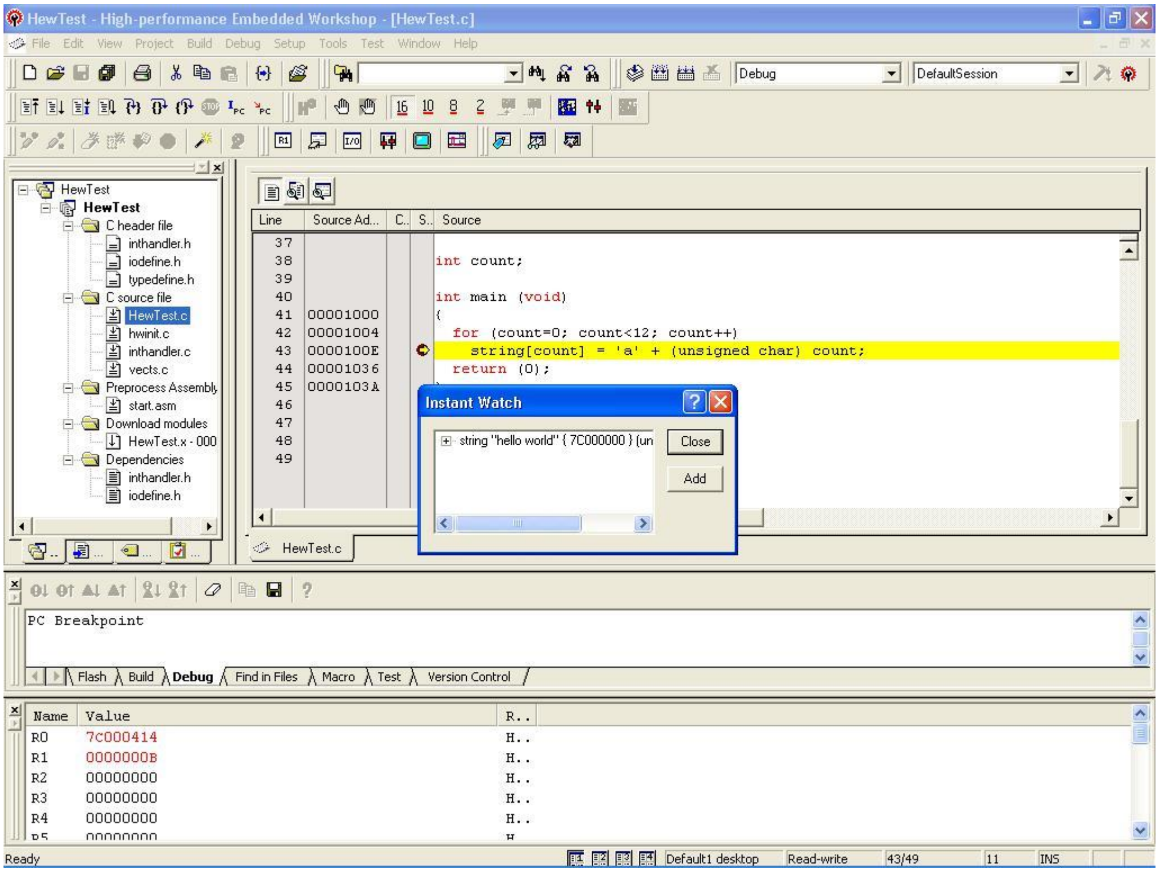
```
37
38                       int cour
39
40                       int mair
41    00001000           {
42    00001004             for (c
43    0000100E               stri                       ) count;
44    00001036             returr
45    0000103A           }
46
47
48
49
```

```
PC Breakpoint
```

Name	Value
R0	7C000414
R1	0000000B
R2	00000000
R3	00000000
R4	00000000
R5	00000000

Step 20

HEW displays an ¯Instant Watch‖ window containing ¯string‖. Click on the ¯Add‖ button.



HewTest - High-performance Embedded Workshop - [HewTest.c]

```
int count;

int main (void)
{
  for (count=0; count<12; count++)
    string[count] = 'a' + (unsigned char) count;
  return (0);
```

Instant Watch

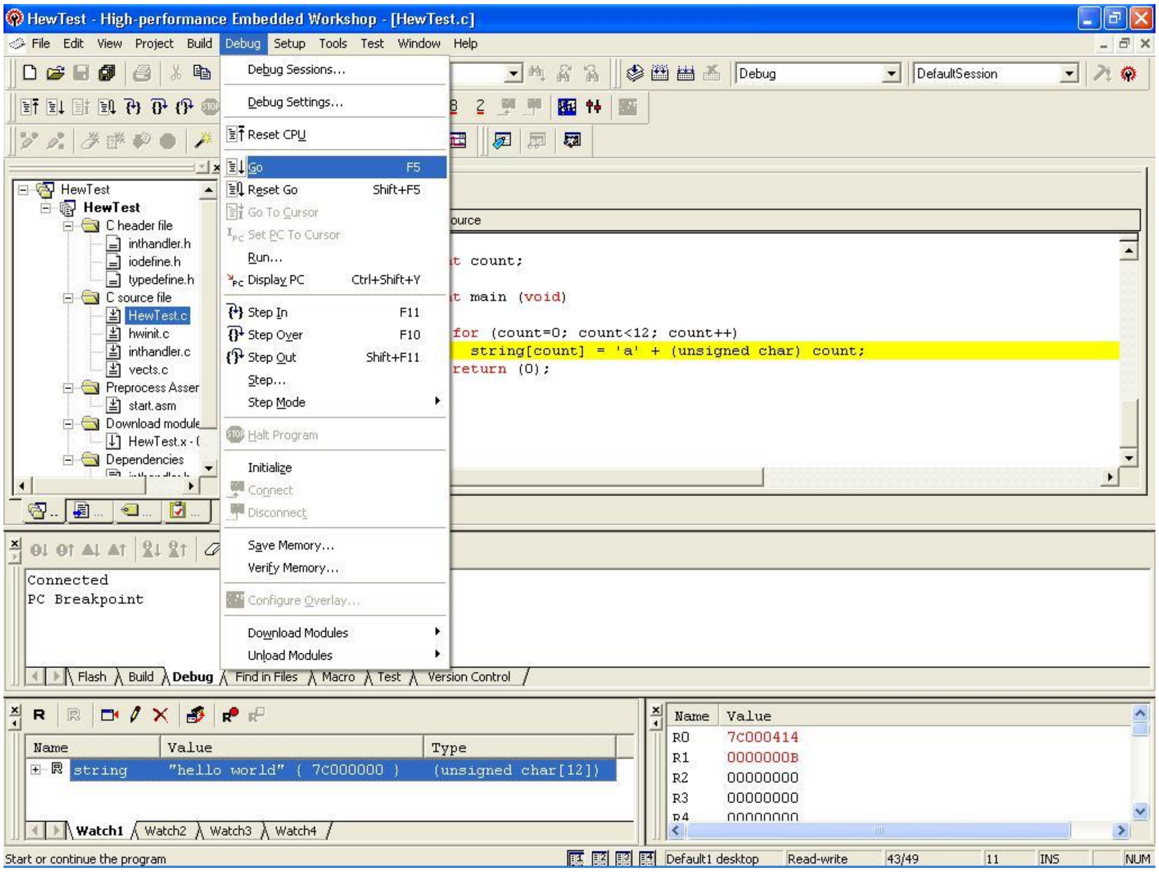
string "hello world" { 7C000000 } (un

```
PC Breakpoint
```

Name	Value	R..
R0	7c000414	H..
R1	0000000B	H..
R2	00000000	H..
R3	00000000	H..
R4	00000000	H..

Step 21

A new ¯Watch Window‖ is displayed showing the string variable‘s address in RAM, data type and value. At this time you may need to adjust the height and/or position of the various frames in the HEW window so you can see the necessary information. Next, select the ¯Go‖ item from the ¯Debug‖ menu on the menu bar.



HewTest - High-performance Embedded Workshop - [HewTest.c]

```
t count;

t main (void)

for (count=0; count<12; count++)
    string[count] = 'a' + (unsigned char) count;
return (0);
```

```
Connected
PC Breakpoint
```

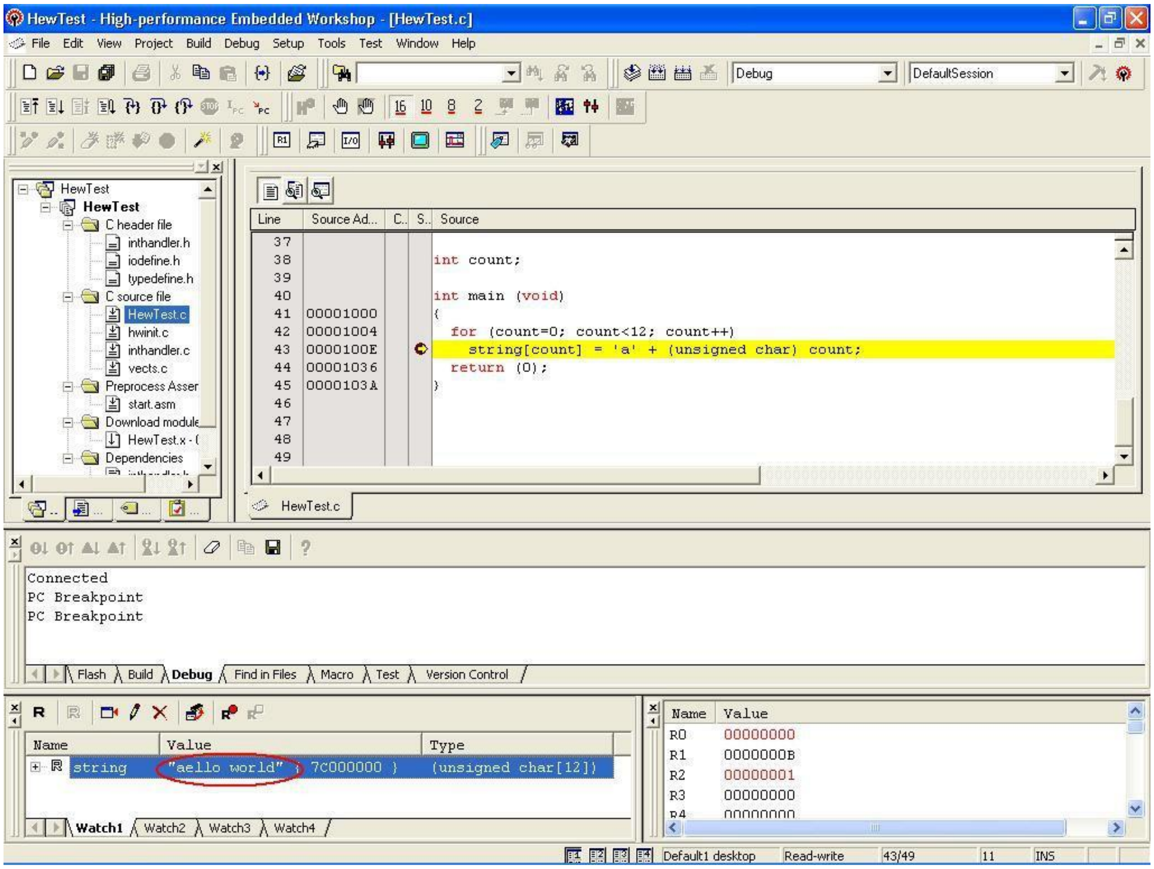
Name	Value	Type
string	"hello world" { 7C000000 }	(unsigned char[12])

Name	Value
R0	7C000414
R1	0000000B
R2	00000000
R3	00000000
R4	00000000

Start or continue the program | Default1 desktop | Read-write | 43/49 | 11 | INS | NUM

Step 22

You will see the first character of the ¯string‖ variable has changed in the watch window so that string now reads, ¯Hello world\n‖. Register values that were modified during code execution are highlighted in red in the ¯Registers‖ window.



```
            int count;

            int main (void)
00001000    {
00001004      for (count=0; count<12; count++)
0000100E        string[count] = 'a' + (unsigned char) count;
00001036      return (0);
0000103A    }
```

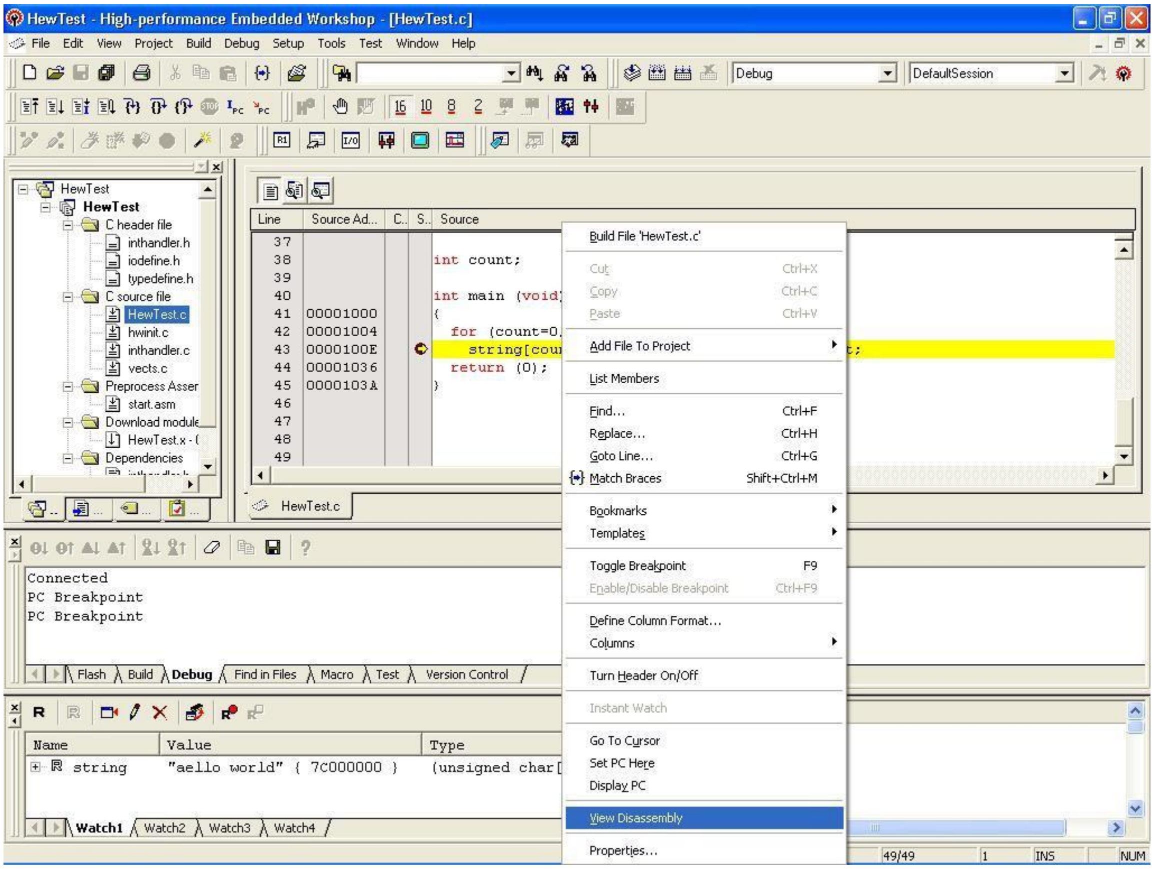
```
Connected
PC Breakpoint
PC Breakpoint
```

Name	Value	Type
string	"aello world" 7C000000 }	(unsigned char[12])

Name	Value
R0	00000000
R1	0000000B
R2	00000001
R3	00000000
R4	00000000

Step 23

User can add a further window to this session. This window will display a mixed view of both source code and the generated machine code for that source. This will allow us to see and step through the individual instructions executed on the target. To add this window, click the left mouse button anywhere in the source code window and then click the right mouse button to show the context menu.
From this, select ―View Disassembly‖. The new ―Disassembly‖ window appears.
Note: that the yellow arrow overlaid on the red breakpoint dot now shows the individual instruction that will be executed next.



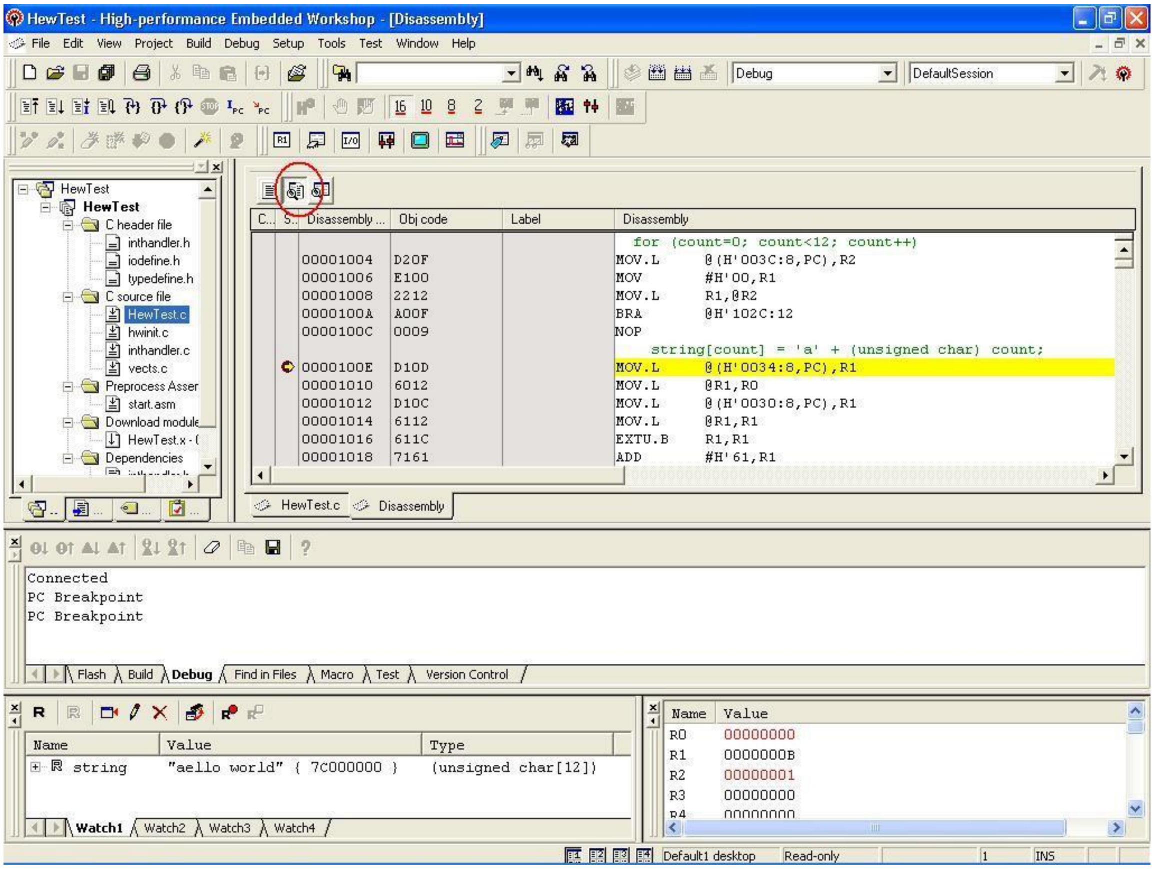
```
37
38                int count;
39
40                int main (void)
41  00001000      {
42  00001004        for (count=0,
43  0000100E          string[cou                       t;
44  00001036        return (0);
45  0000103A      }
46
47
48
49
```

```
Connected
PC Breakpoint
PC Breakpoint
```

Name	Value	Type
string	"aello world" { 7C000000 }	(unsigned char[

Step 24

In order to view both source and disassembled object code in the disassembly view, click the button shown in the figure below. Source code lines are added to the disassembled object code.



```
C... S.. Disassembly ... Obj code  Label  Disassembly
                                            for (count=0; count<12; count++)
         00001004        D20F               MOV.L     @(H'003C:8,PC),R2
         00001006        E100               MOV       #H'00,R1
         00001008        2212               MOV.L     R1,@R2
         0000100A        A00F               BRA       @H'102C:12
         0000100C        0009               NOP
                                              string[count] = 'a' + (unsigned char) count;
         0000100E        D10D               MOV.L     @(H'0034:8,PC),R1
         00001010        6012               MOV.L     @R1,R0
         00001012        D10C               MOV.L     @(H'0030:8,PC),R1
         00001014        6112               MOV.L     @R1,R1
         00001016        611C               EXTU.B    R1,R1
         00001018        7161               ADD       #H'61,R1
```

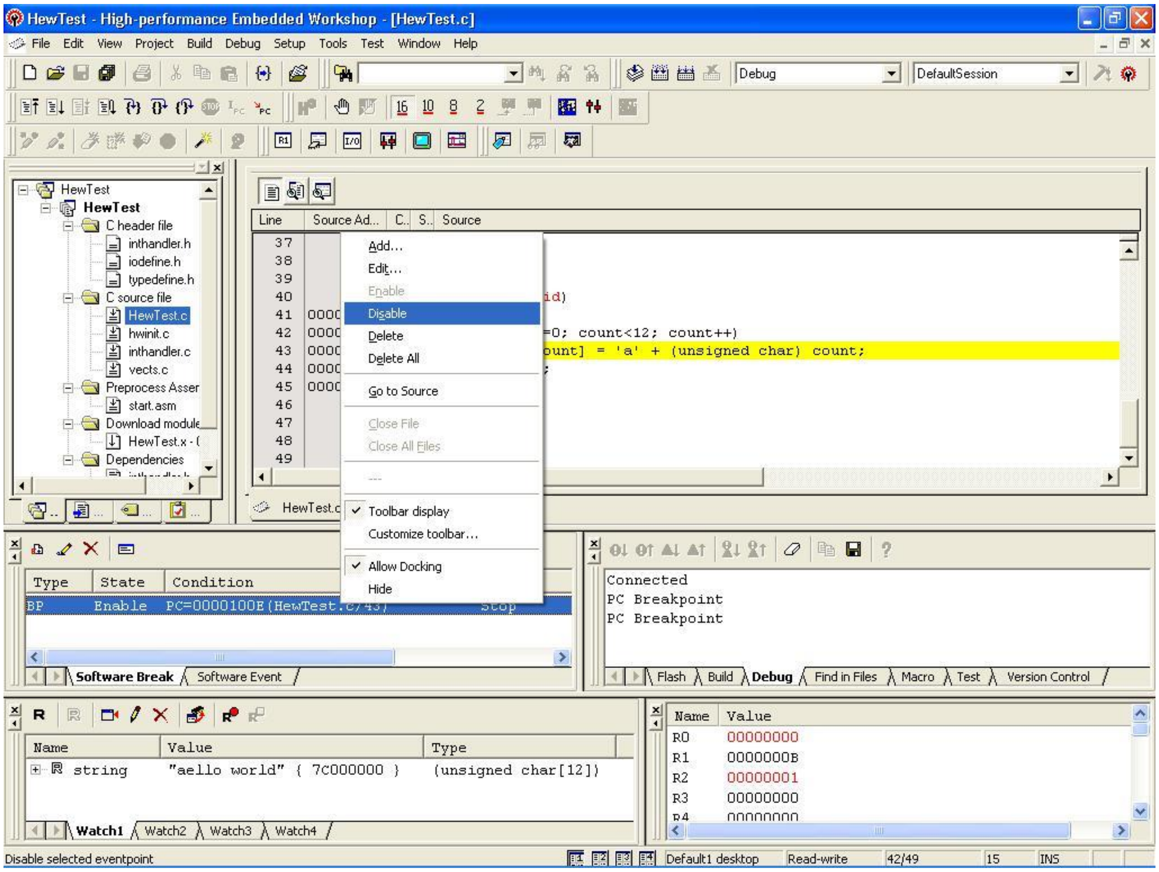
```
Connected
PC Breakpoint
PC Breakpoint
```

Name	Value	Type
string	"aello world" { 7C000000 }	(unsigned char[12])

Name	Value
R0	00000000
R1	0000000B
R2	00000001
R3	00000000
R4	00000000

Step 25

Now we will use another debugging feature of the HEW debugger. Select the ―Code‖ item from the ―View‖ menu on the menu bar. In the sub-menu that is displayed, select the ―Eventpoints‖ item. This displays a new ―Event‖ view showing the breakpoint we added earlier (***Note:*** you may need to adjust the sizes of the various frames in the HEW window again at this point). This time we wish to temporarily disable the breakpoint, so position the mouse pointer over the single line in the ―Event‖ view and click the right mouse button. From the context menu that appears, select the ―Disable‖ item. The breakpoint is now disabled and will not interrupt execution.



Line	Source Ad...	C..	S..	Source
37				
38				
39				
40				id)
41	0000			
42	0000			=0; count<12; count++)
43	0000			ount] = 'a' + (unsigned char) count;
44	0000			;
45	0000			
46				
47				
48				
49				

Type	State	Condition	
BP	Enable	PC=0000100E(HewTest.c/43)	Stop

```
Connected
PC Breakpoint
PC Breakpoint
```

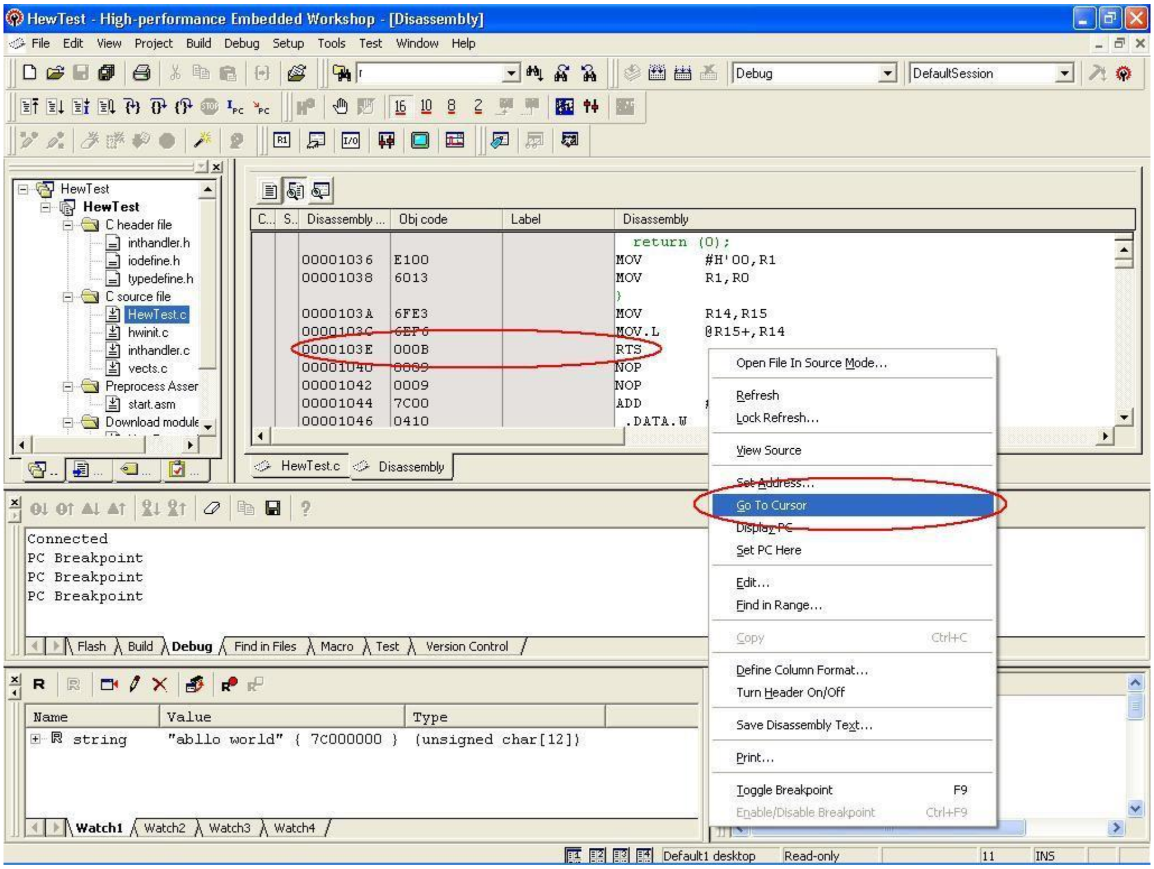
Name	Value	Type
string	"aello world" { 7C000000 }	(unsigned char[12])

Name	Value
R0	00000000
R1	0000000B
R2	00000001
R3	00000000
R4	00000000

Disable selected eventpoint | Default1 desktop | Read-write | 42/49 | 15 | INS

Step 26

As a final demonstration of debugging, we will execute a specific range of instructions starting from the highlighted instruction. Close the ¯Event‖ view using the ¯X‖ button at the top left of the view. Now position the mouse pointer over the ¯Disassembly‖ window and click the left mouse button. Locate the ¯RTS instruction‖ at address 0x103E. Position the mouse pointer over the instruction and click the left mouse button. Now click the right mouse button and select ¯Go To Cursor‖ from the context menu.



HewTest - High-performance Embedded Workshop - [Disassembly]

```
                             return (0);
00001036  E100               MOV        #H'00,R1
00001038  6013               MOV        R1,R0
                             }
0000103A  6FE3               MOV        R14,R15
0000103C  6EF6               MOV.L      @R15+,R14
0000103E  000B               RTS
00001040  0009               NOP
00001042  0009               NOP
00001044  7C00               ADD        #
00001046  0410                .DATA.W
```

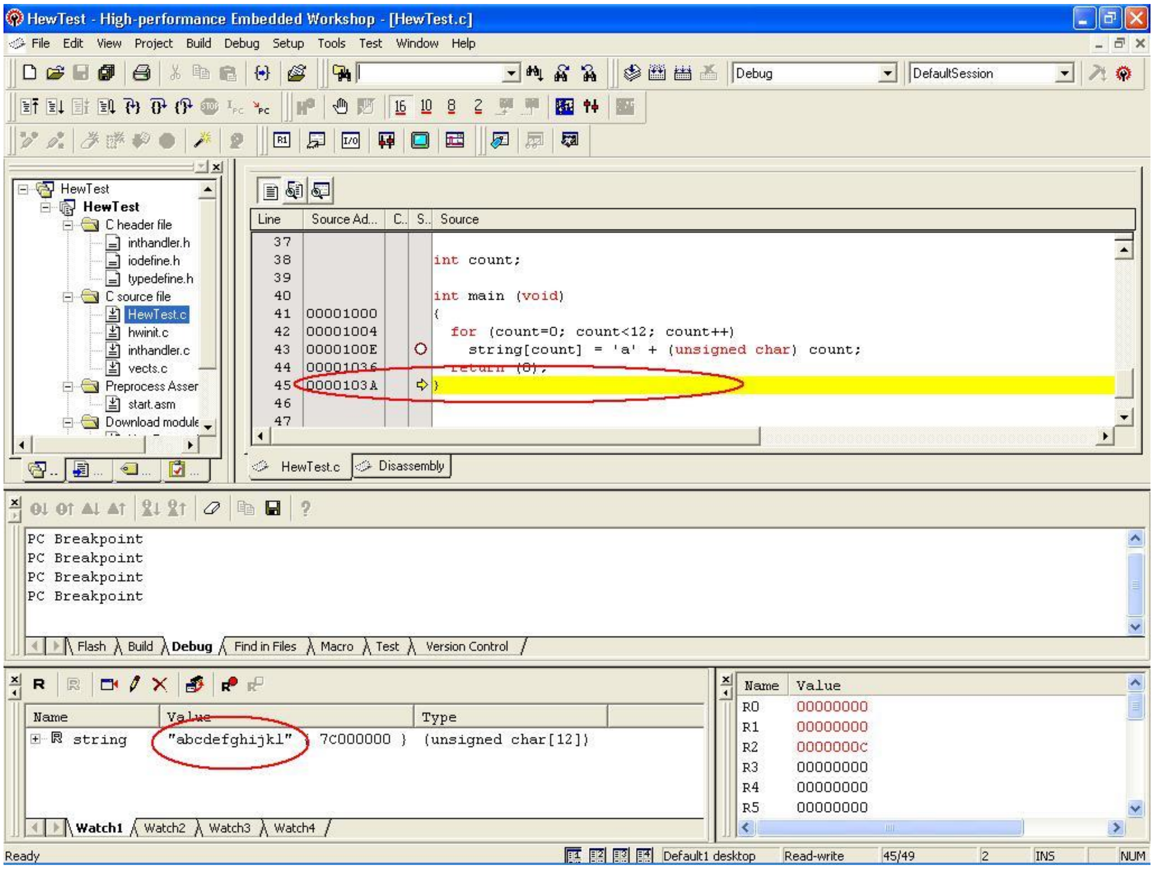
```
Connected
PC Breakpoint
PC Breakpoint
PC Breakpoint
```

Name	Value	Type
string	"abllo world" { 7C000000 }	(unsigned char[12])

Step 27

Return to the C source code and the HEW editor now changes the yellow arrow to indicate that execution stopped just before exiting the main instruction. Notice also that the ¯string‖ variable in the watch window now shows the value ¯abcdefghijkl‖ and the register values have changed once more.

This completes the HEW debugging interface tutorial. Note that you can use the ¯Save Session‖ item in the ¯File‖ menu on the menu bar to save your entire session including simulator configuration and window layout. This will save you having to reconfigure these settings each time you debug your code.



```
Line  Source Ad...  C..  S..  Source
 37
 38                          int count;
 39
 40                          int main (void)
 41   00001000               {
 42   00001004                 for (count=0; count<12; count++)
 43   0000100E           O       string[count] = 'a' + (unsigned char) count;
 44   00001036                 return (0);
 45   0000103A           ⇨   }
 46
 47
```

```
PC Breakpoint
PC Breakpoint
PC Breakpoint
PC Breakpoint
```

Name	Value	Type
⊞ string	"abcdefghijkl" { 7C000000 }	(unsigned char[12])

Name	Value
R0	00000000
R1	00000000
R2	0000000C
R3	00000000
R4	00000000
R5	00000000

2.7 Porting HMON

While porting HMON code to the Renesas micro-controllers with custom board configurations, the following changes needs to be done in the HMON tutorial project;

- hmonserialconfiguser.c

  The function _HMONInitHWSerial needs to be modified for the following:

  a. Enabling external SRAM and its setup

  b. Enabling address, data lines and chip selects

  c. Other h/w initialization necessary for board boot up

- edkXXXX.h

  User LEDs may have been connected on different ports than EDK. This needs to be changed as per the custom board.

- HMONSerialConfigUser.h

  The serial channel used by the custom board may be different than the EDK. This needs to be checked and if required the base of the channel needs to be changed as per the custom board.

- vecttbl.c , vect.h and IntPRG.c

  Depending upon the macro HMON_Debug the service routines for the serial channel interrupt are selected. The HMON_Debug macro needs to be placed such that the proper interrupt service routine for serial channel is used. Above mentioned files need to be changed for this.

- hmonconfiguser.h

  The definition HMON_POWER_ON_RESET_PTR_PTR needs to be changed as per user_vector location

- Linker Sections

  If a particular micro-controller supports hardware and software breakpoints then the application code can be in SRAM or in flash. Hence the location of the application code section needs to be changed in the linker sections. In HEW, it can be done by selecting 'Options' -> 'Linker' -> 'Sections'.

2.8 Using E10A Emulator

Following sections explains how to use E10A Emulator with RSK

2.8.1 RSK Quick start Guide Generating a Test Project

2.8.1.1 Software Required

Please ensure that you have following installed on your machine in the sequence given below

a. USB Driver for E8 emulator

b. HEW-4.x

c. KPIT GNUSH toolchain.

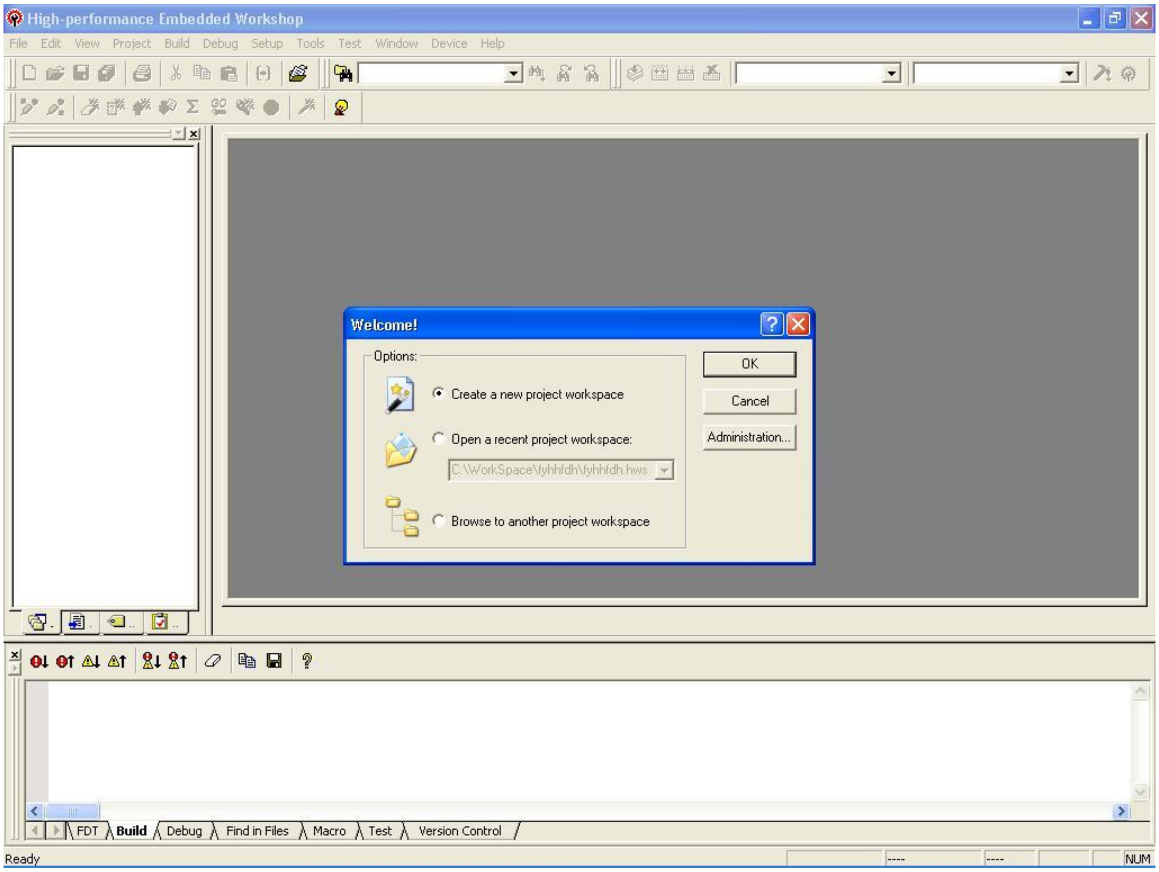
d. E8 Debugger Support Software for SH2 HMON.

e. KPIT RSKSH7124 installer.

Once you have installed the required software you can quit the menu program and start High-performance Embedded Workshop (HEW).

Step 1

Start HEW4 by using the Start Menu to navigate to and select "High - performance Embedded Workshop". In the "Welcome" dialog box, select "Create a new project workspace" and click "OK".


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Device Help
Welcome!
OK
Cancel
Administration...
FDT
Build
Debug
Find in Files
Macro
Test
Version Control
Ready
NUM

Options:

● Create a new project workspace

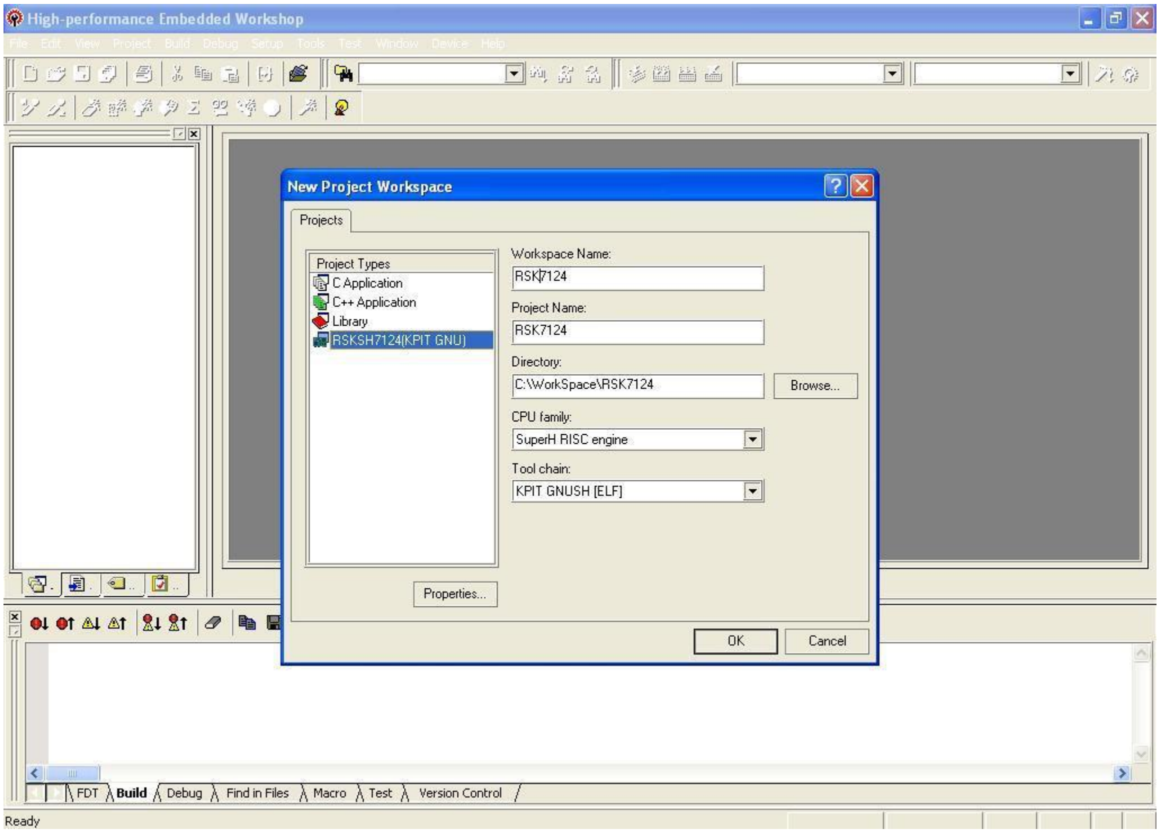
○ Open a recent project workspace:

C:\WorkSpace\fyhhfdh\fyhhfdh.hws

○ Browse to another project workspace

Step 2

In the "New Project Workspace" dialog box, change the directory to a suitable location. Enter the Workspace Name, "RSK7124 ". Enter the Project Name, "RSK7124". Select the CPU Family for the RSK: "SuperH RISC Engine". Select the Toolchain: "KPIT GNUSH ELF". Select the Project Type "RSKSH7124 (KPIT GNU)" for your RSK. Click "OK". This will start the RSK Project Generator wizard, which will set up the correct environment for your RSK.



New Project Workspace

Project Types
- C Application
- C++ Application
- Library
- RSKSH7124(KPIT GNU)

Workspace Name: RSK7124

Project Name: RSK7124

Directory: C:\WorkSpace\RSK7124

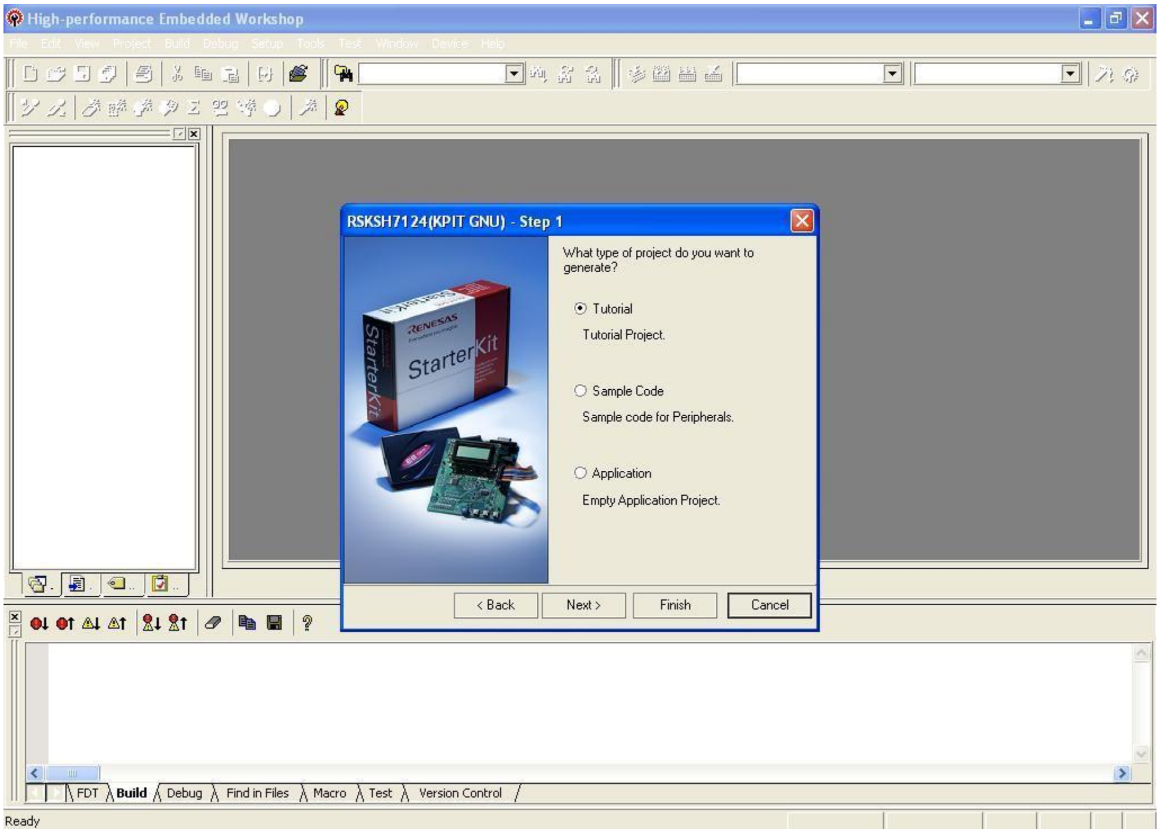
CPU family: SuperH RISC engine

Tool chain: KPIT GNUSH [ELF]

High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Device Help
Projects
Browse...
Properties...
OK
Cancel
FDT
Build
Debug
Find in Files
Macro
Test
Version Control
Ready

Step 3

When asked "What type of project do you want to generate?", select from Tutorial/Sample Code/Application and click "Next >". Here we have selected ―Tutorial‖ project.



RSKSH7124(KPIT GNU) - Step 1

What type of project do you want to generate?

● Tutorial

Tutorial Project.

○ Sample Code

Sample code for Peripherals.

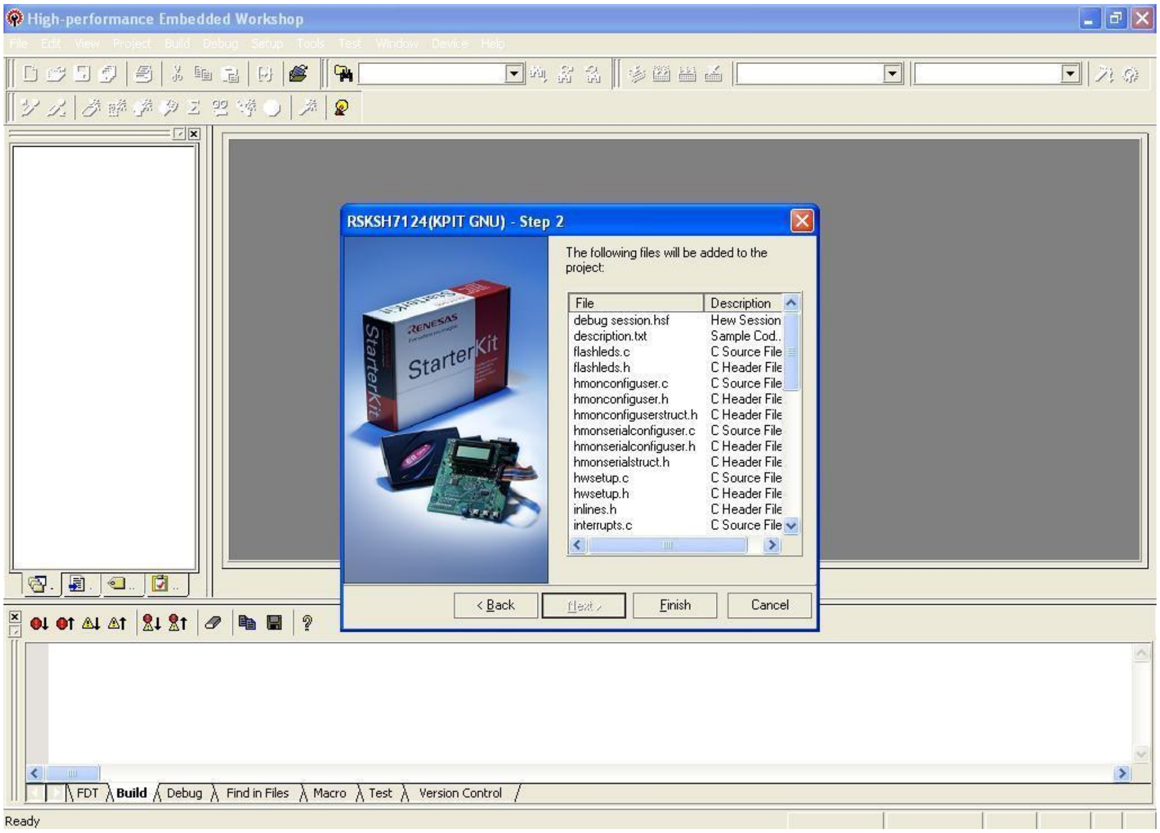
○ Application

Empty Application Project.

High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Device Help
< Back
Next >
Finish
Cancel
FDT
Build
Debug
Find in Files
Macro
Test
Version Control
Ready

Step 4

Select the list of project files to be included is displayed. Click "Finish".



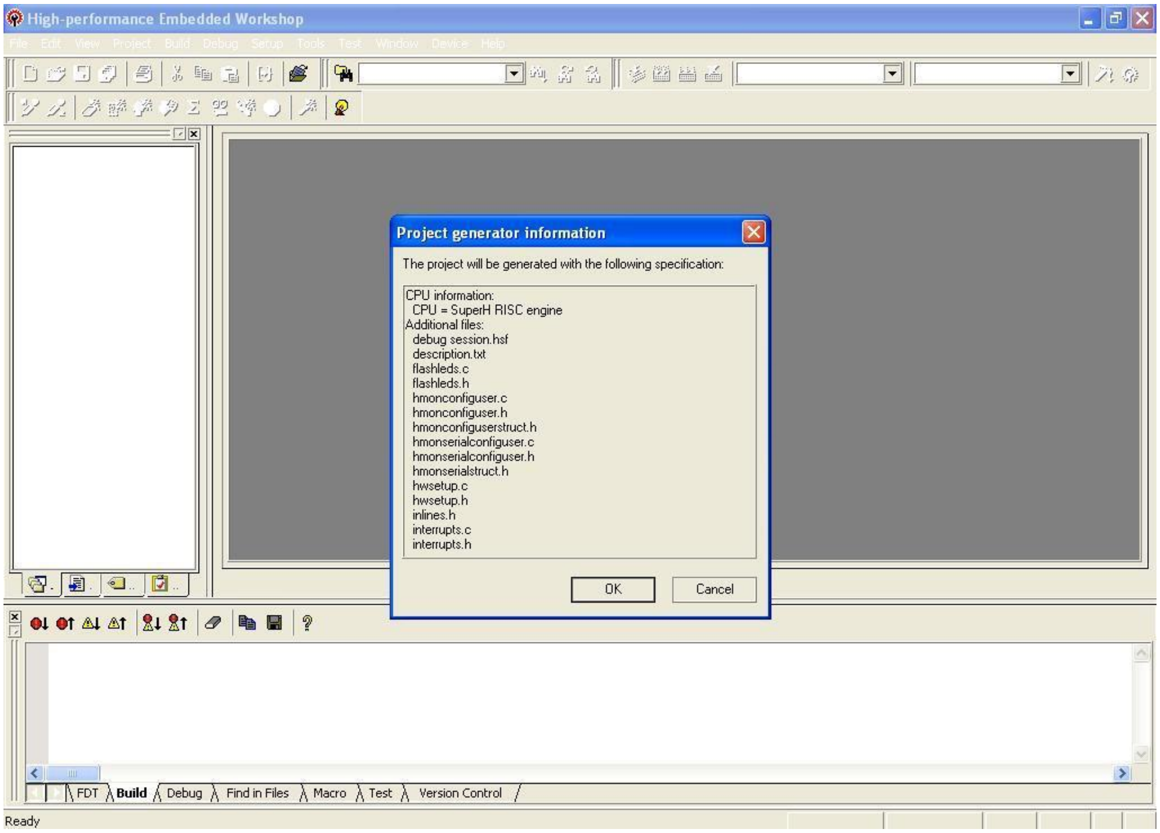
High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Device Help
RSKSH7124(KPIT GNU) - Step 2
< Back
Next >
Finish
Cancel
FDT
Build
Debug
Find in Files
Macro
Test
Version Control
Ready

The following files will be added to the project:

File	Description
debug session.hsf	Hew Session
description.txt	Sample Cod..
flashleds.c	C Source File
flashleds.h	C Header File
hmonconfiguser.c	C Source File
hmonconfiguser.h	C Header File
hmonconfiguserstruct.h	C Header File
hmonserialconfiguser.c	C Source File
hmonserialconfiguser.h	C Header File
hmonserialstruct.h	C Header File
hwsetup.c	C Source File
hwsetup.h	C Header File
inlines.h	C Header File
interrupts.c	C Source File

Step 5

In the "Project generator information" dialog box, you will see a list of project files appear in the Workspace window, showing that the Project Generator has created the project for you. Click "OK".


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Device Help
FDT
Build
Debug
Find in Files
Macro
Test
Version Control
Ready

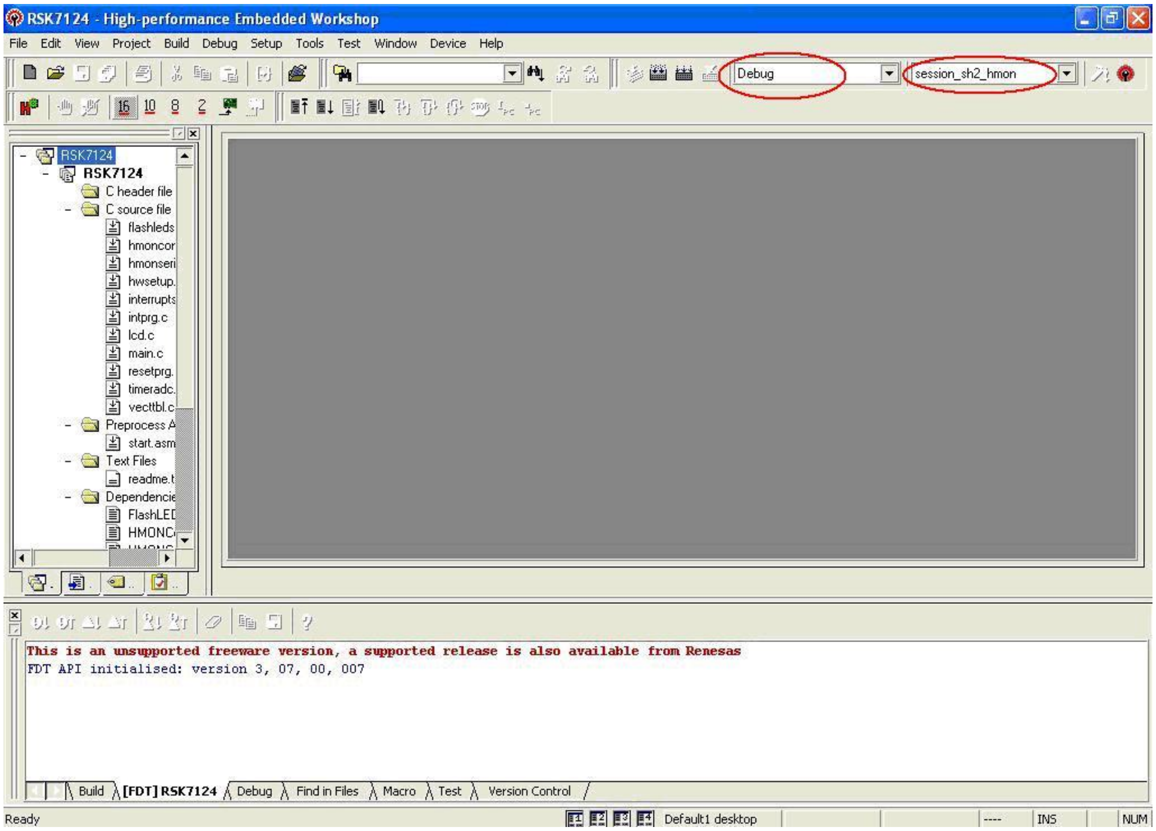
Project generator information

The project will be generated with the following specification:

```
CPU information:
  CPU = SuperH RISC engine
Additional files:
  debug session.hsf
  description.txt
  flashleds.c
  flashleds.h
  hmonconfiguser.c
  hmonconfiguser.h
  hmonconfiguserstruct.h
  hmonserialconfiguser.c
  hmonserialconfiguser.h
  hmonserialstruct.h
  hwsetup.c
  hwsetup.h
  inlines.h
  interrupts.c
  interrupts.h
```

Step 6

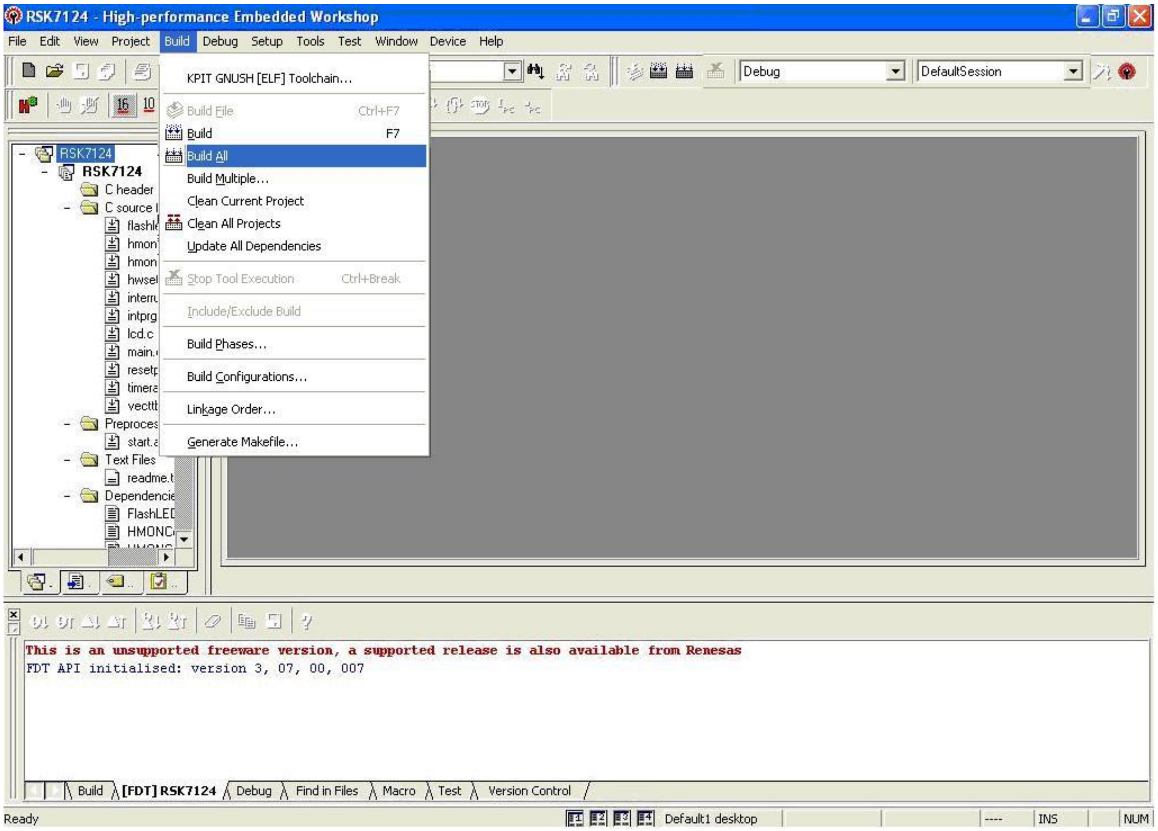
Ensure that the project workspace is set to the following: "Debug" configuration and session ¯Session_SH2_hmon".



```
This is an unsupported freeware version, a supported release is also available from Renesas
FDT API initialised: version 3, 07, 00, 007
```

Step 7

Select the "Build"-> "Build All" menu item to build the program. When the build is complete, proceed to the next section to program your RSK with the project.



```
This is an unsupported freeware version, a supported release is also available from Renesas
FDT API initialised: version 3, 07, 00, 007
```

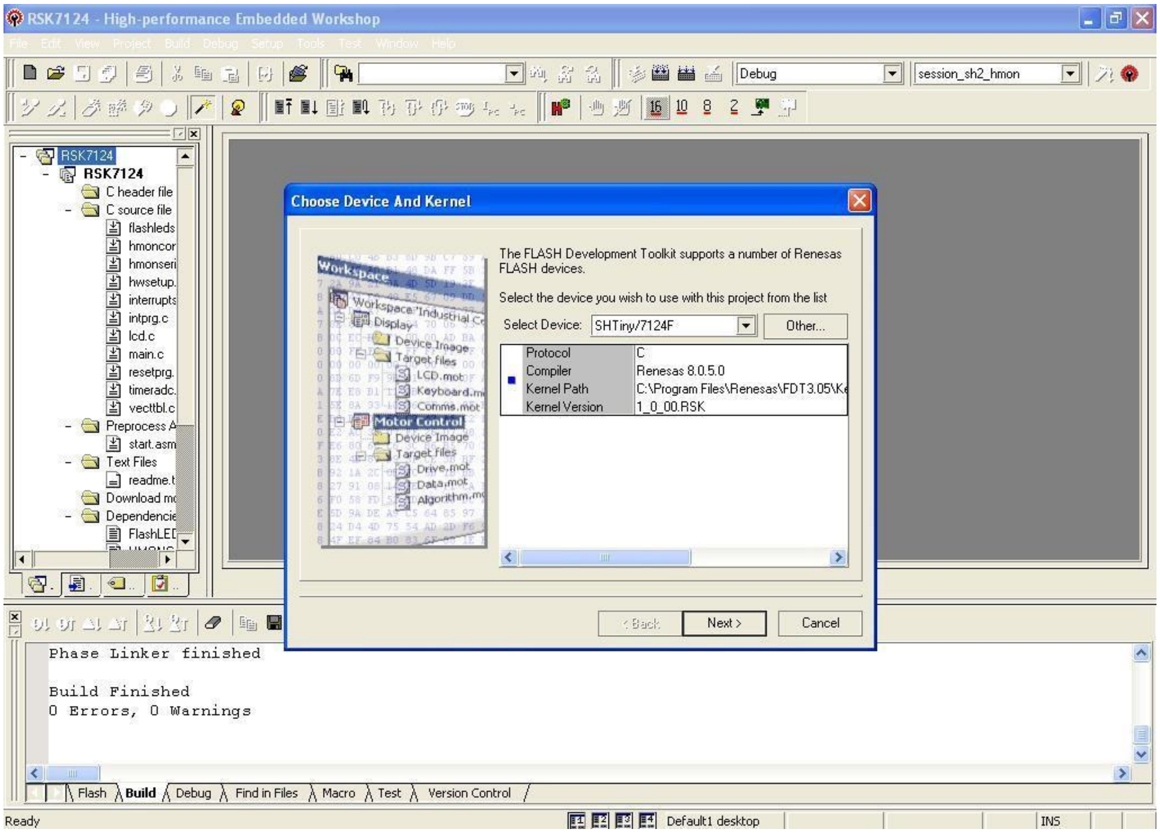
2.8.1.2 Programming the Board

Step 1

Ensure that the hardware along with E8 emulator is connected to the PC through USB port. If the FLASH control bar is not present on the menu bar then, right click anywhere on the menu bar and then select FDT. FDT menu bar will appear. If the FLASH control bar is grayed out in this mode, with only the magic wand visible, then the FDT settings for the board need to be set up. The FDT configuration wizard will appear after clicking magic wand. Select your device from the drop down menu. Click the _Kernels' for your RSK. These are denoted by an ―.RSK‖ after the version Kernels.

$(FDT Installation directory)/Kernels/$(Protocol)/$(Target)/Renesas/
1_X_XX.RSK

Click the "Next" button.



Choose Device And Kernel

The FLASH Development Toolkit supports a number of Renesas FLASH devices.

Select the device you wish to use with this project from the list

Select Device: SHTiny/7124F Other...

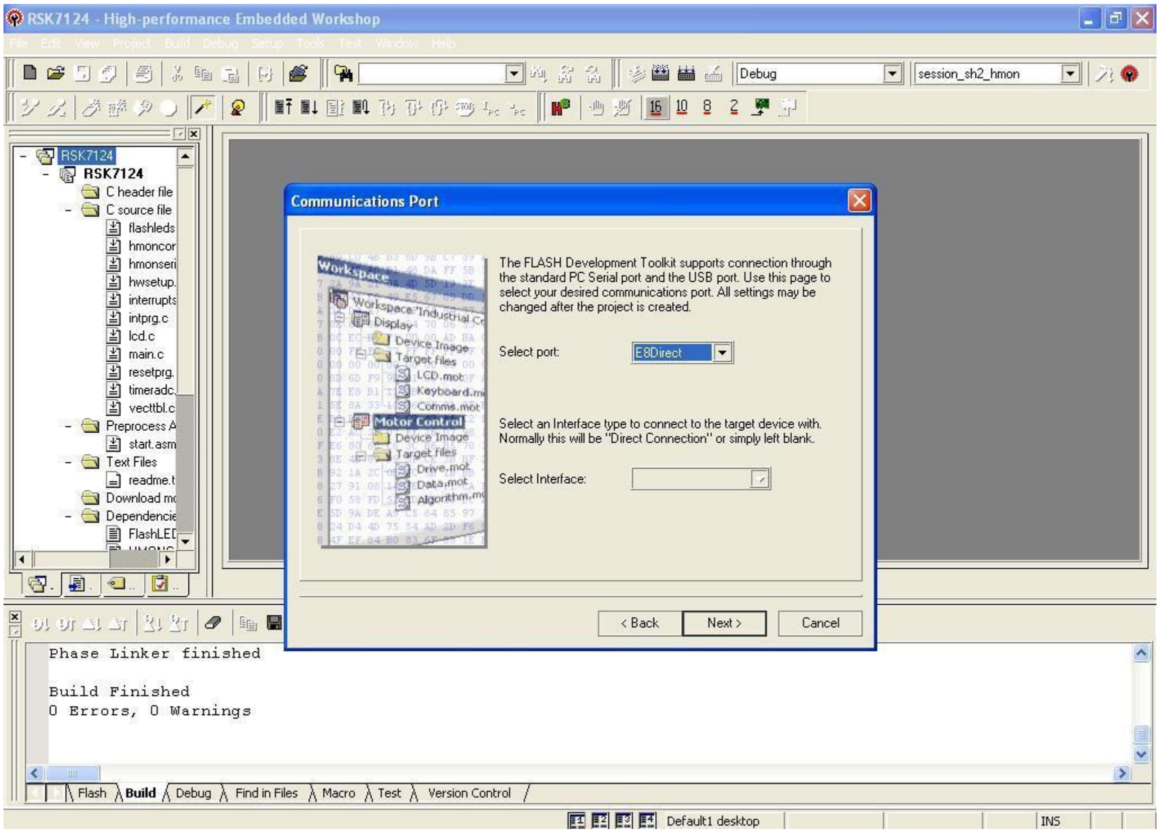
Protocol	C
Compiler	Renesas 8.0.5.0
Kernel Path	C:\Program Files\Renesas\FDT3.05\K
Kernel Version	1_0_00.RSK

< Back Next > Cancel

```
Phase Linker finished

Build Finished
0 Errors, 0 Warnings
```

Step 2

Select E8Direct to connect to your RSK, and click "Next".



The FLASH Development Toolkit supports connection through the standard PC Serial port and the USB port. Use this page to select your desired communications port. All settings may be changed after the project is created.

Select port: E8Direct

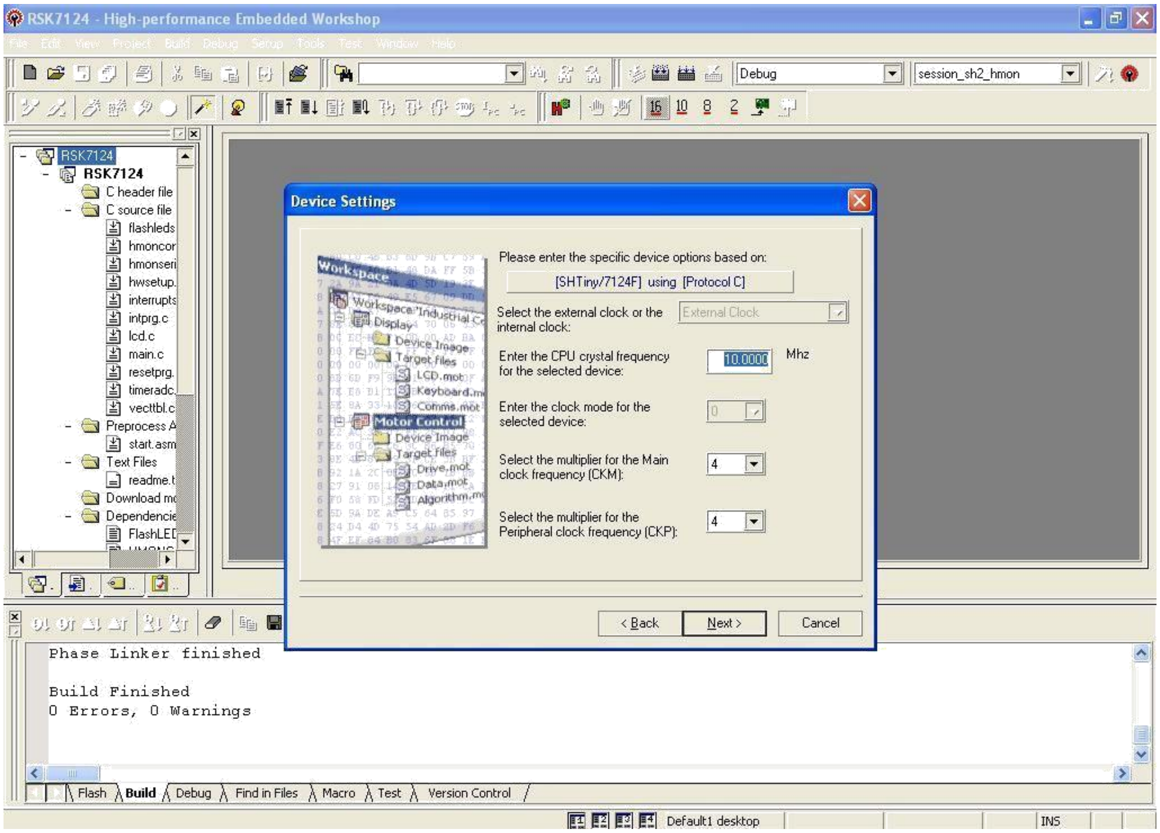
Select an Interface type to connect to the target device with. Normally this will be "Direct Connection" or simply left blank.

Select Interface:

```
Phase Linker finished

Build Finished
0 Errors, 0 Warnings
```

Step 3

The CPU Crystal Frequency of the RSK should match the preset value. If not matching please update it to match with that of the RSK (10.000 Mhz in case of SH/7124F). Click "Next" to accept the settings.



Device Settings

Please enter the specific device options based on:

[SHTiny/7124F] using [Protocol C]

Select the external clock or the internal clock: External Clock

Enter the CPU crystal frequency for the selected device: 10.0000 Mhz

Enter the clock mode for the selected device: 0

Select the multiplier for the Main clock frequency (CKM): 4

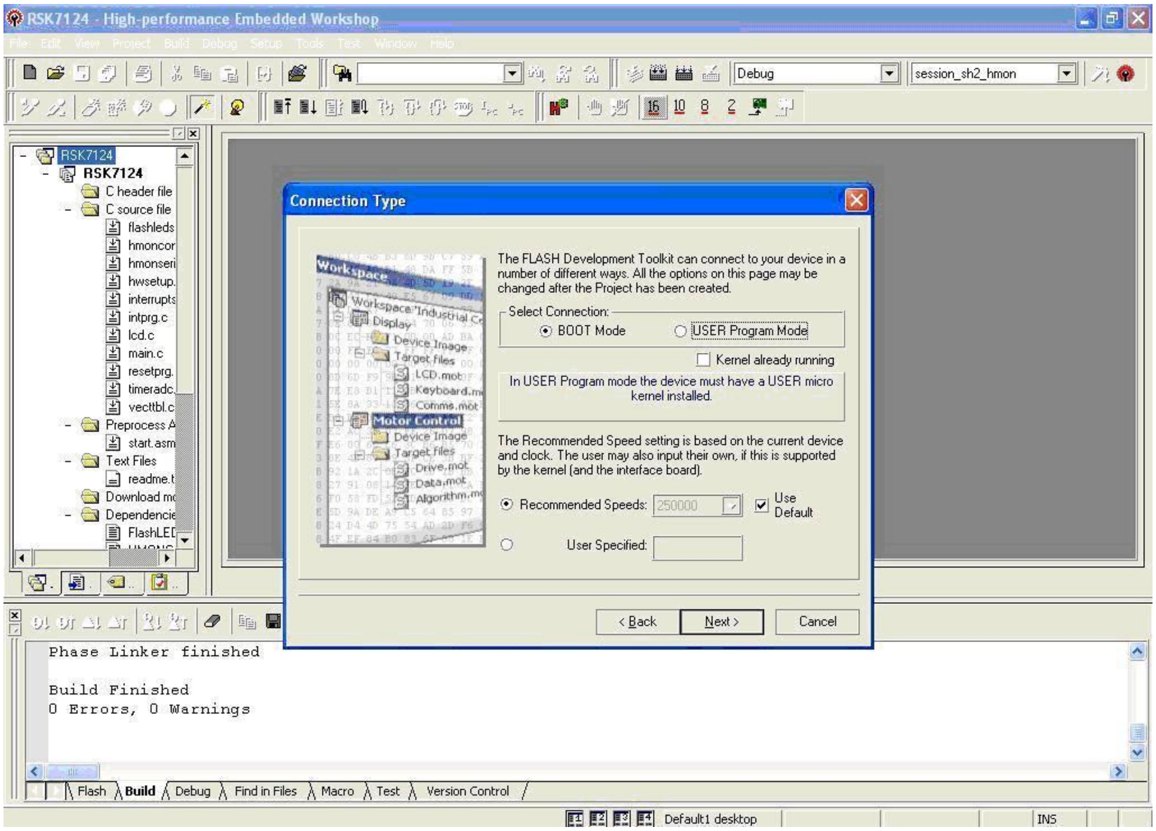
Select the multiplier for the Peripheral clock frequency (CKP): 4

< Back | Next > | Cancel

```
Phase Linker finished

Build Finished
0 Errors, 0 Warnings
```

Step 4

The default setting of ¯Select Connection‖ is on USER Program Mode. Select "BOOT Mode" and Click ¯Next‖.



The FLASH Development Toolkit can connect to your device in a number of different ways. All the options on this page may be changed after the Project has been created.

Select Connection:

● BOOT Mode ○ USER Program Mode

□ Kernel already running

In USER Program mode the device must have a USER micro kernel installed.

The Recommended Speed setting is based on the current device and clock. The user may also input their own, if this is supported by the kernel (and the interface board).

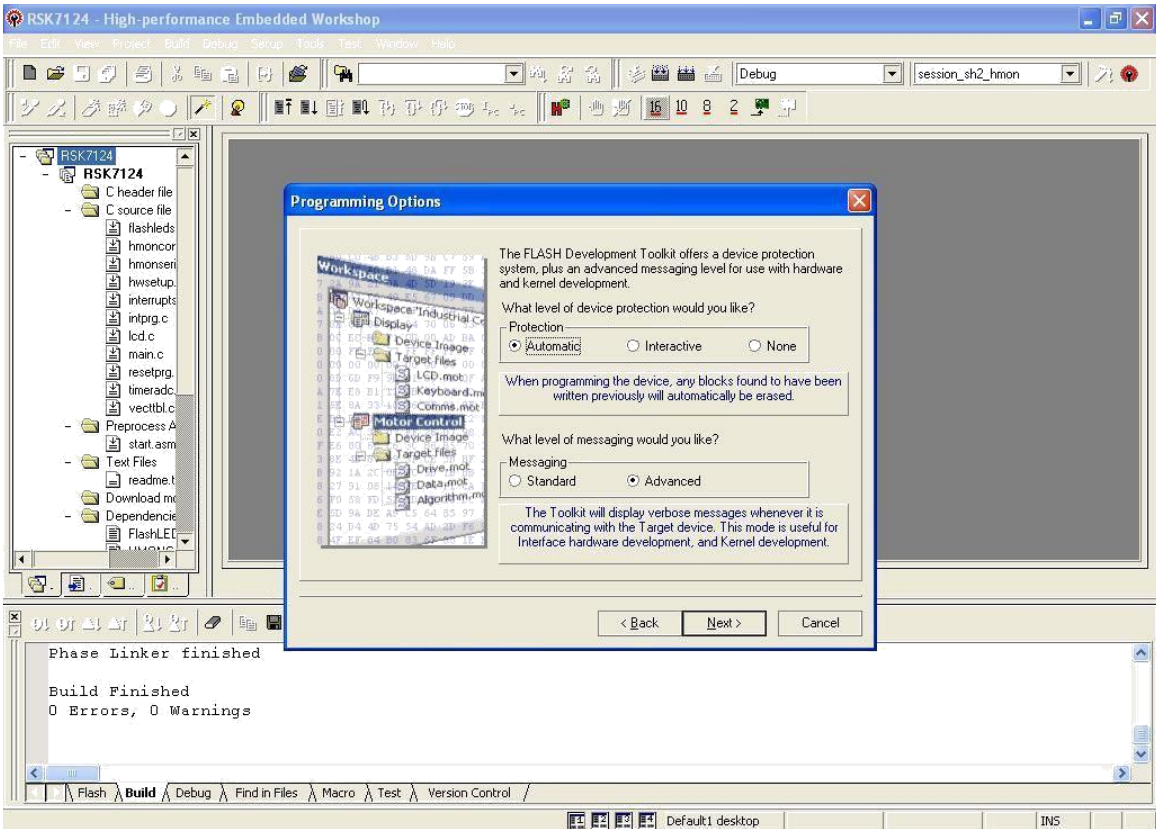
● Recommended Speeds: 250000 ☑ Use Default

○ User Specified:

```
Phase Linker finished

Build Finished
0 Errors, 0 Warnings
```

Step 5

Click "Next" to accept the default settings of ¯Protection‖ as ¯Automatic‖ and ¯‖Messaging Level‖ as ¯Advanced‖.



Programming Options

The FLASH Development Toolkit offers a device protection system, plus an advanced messaging level for use with hardware and kernel development.

What level of device protection would you like?

Protection

● Automatic ○ Interactive ○ None

When programming the device, any blocks found to have been written previously will automatically be erased.

What level of messaging would you like?

Messaging

○ Standard ● Advanced

The Toolkit will display verbose messages whenever it is communicating with the Target device. This mode is useful for Interface hardware development, and Kernel development.

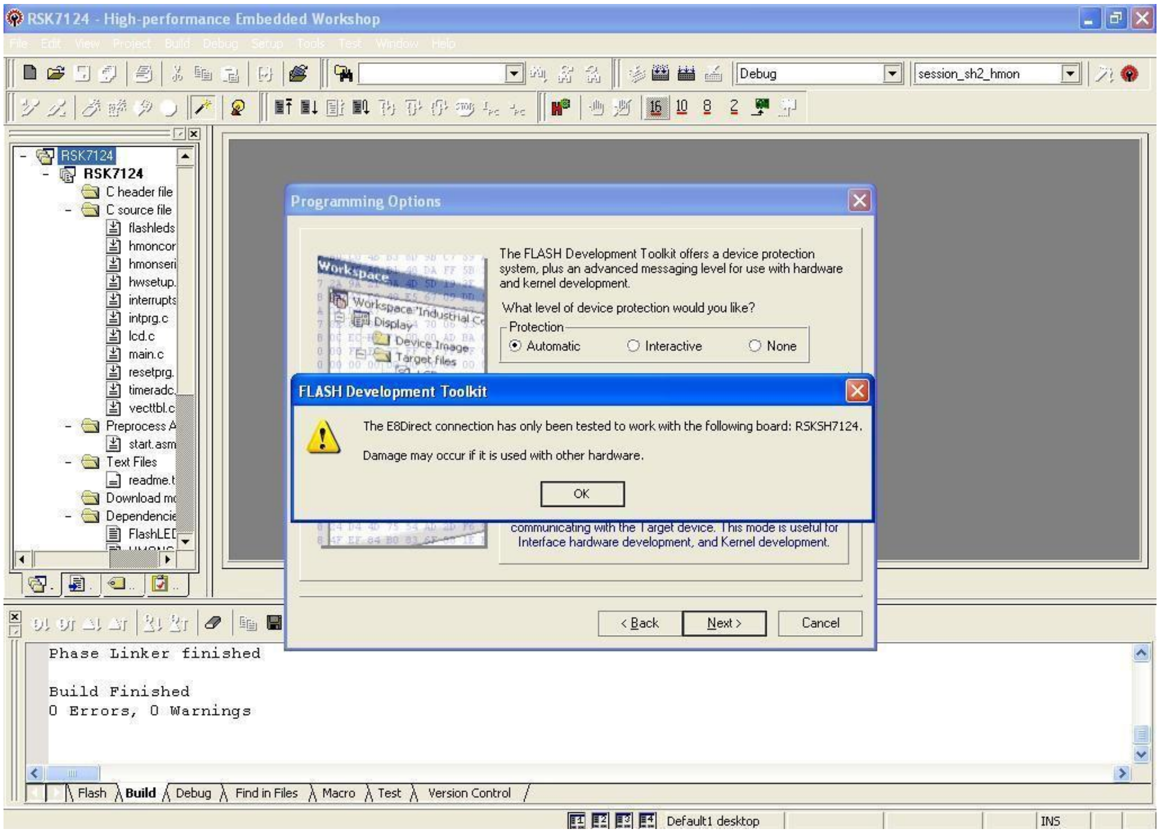
< Back | Next > | Cancel

```
Phase Linker finished

Build Finished
0 Errors, 0 Warnings
```

Step 6

Click "Next" a warning message will be displayed stating,

"The E8Direct connection has only been tested to work with the following board: RSKSH7124. Damage may occur if it is used with other hardware".



Programming Options

The FLASH Development Toolkit offers a device protection system, plus an advanced messaging level for use with hardware and kernel development.

What level of device protection would you like?

Protection: ● Automatic ○ Interactive ○ None

communicating with the Target device. This mode is useful for Interface hardware development, and Kernel development.

< Back | Next > | Cancel

FLASH Development Toolkit

The E8Direct connection has only been tested to work with the following board: RSKSH7124.

Damage may occur if it is used with other hardware.

OK

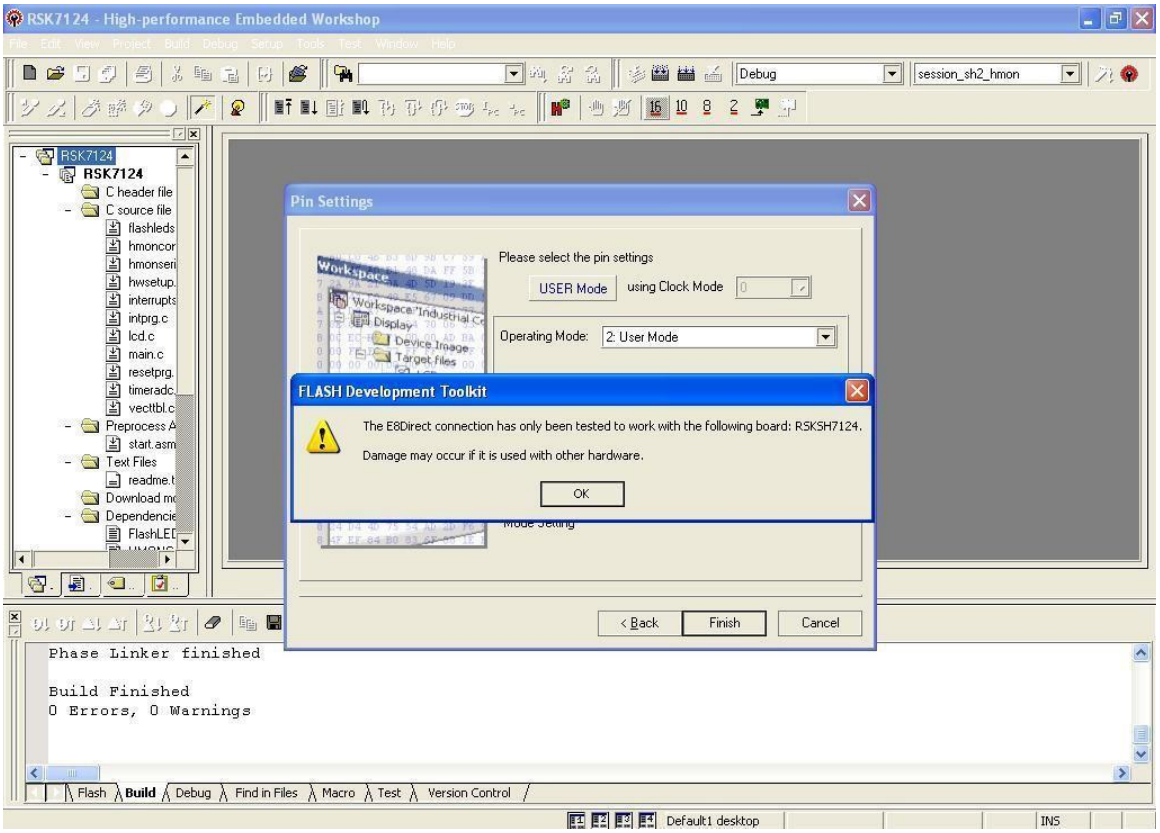
```
Phase Linker finished

Build Finished
0 Errors, 0 Warnings
```

Step 7

Click "OK" and click on "Next". Again the warning will be displayed stating:

"The E8Direct connection has only been tested to work with the following board: RSKSH7124. Damage may occur if it is used with other hardware".

Click "OK" and then "Finish".



The E8Direct connection has only been tested to work with the following board: RSKSH7124.

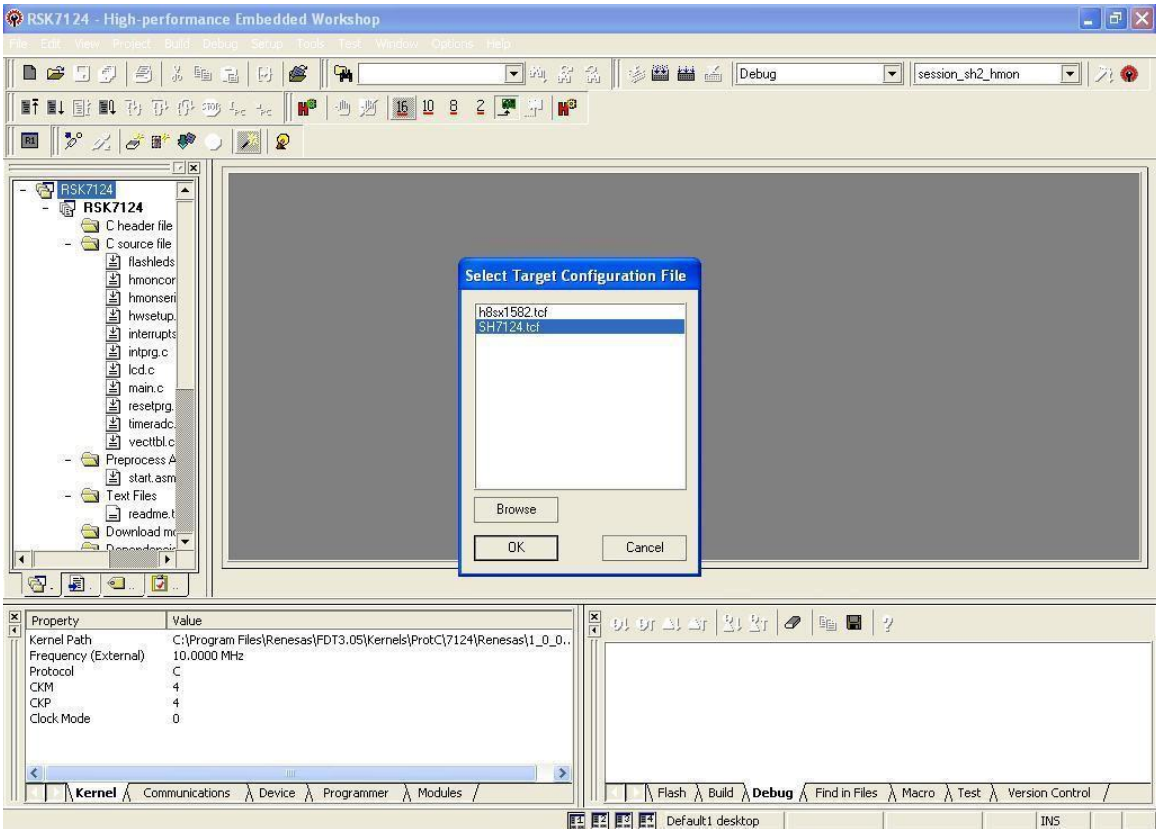
Damage may occur if it is used with other hardware.

```
Phase Linker finished

Build Finished
0 Errors, 0 Warnings
```

Step 8

FDT is now configured for use with the RSK. Choose "Save Session" and then ¯Save workspace‖ from the "File" menu.

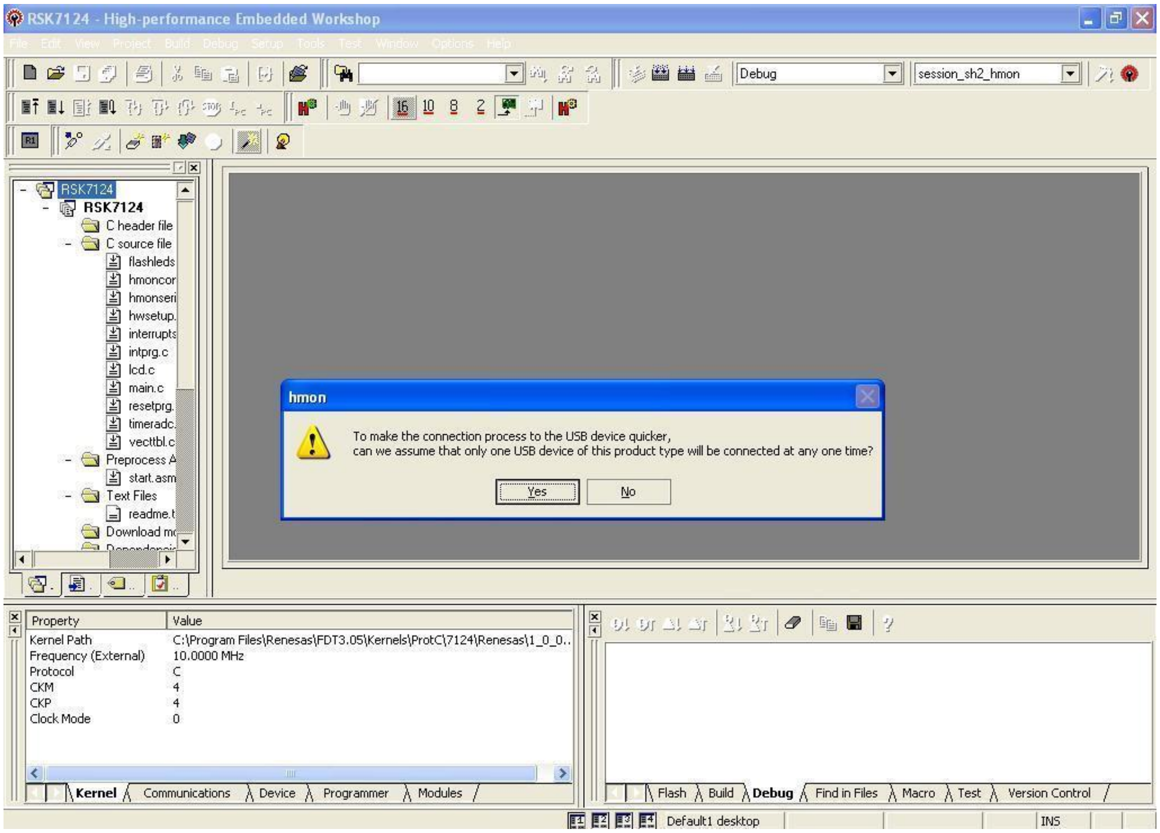
From the "Debug" drop down menu, select "Connect". If prompted for the "Select Target Configuration File", select the file from the list matching your RSK, and click "OK".



Property	Value
Kernel Path	C:\Program Files\Renesas\FDT3.05\Kernels\ProtC\7124\Renesas\1_0_0..
Frequency (External)	10.0000 MHz
Protocol	C
CKM	4
CKP	4
Clock Mode	0

Step 9

Then warning will be displayed to ask if USB device is connected. Click "YES".



hmon

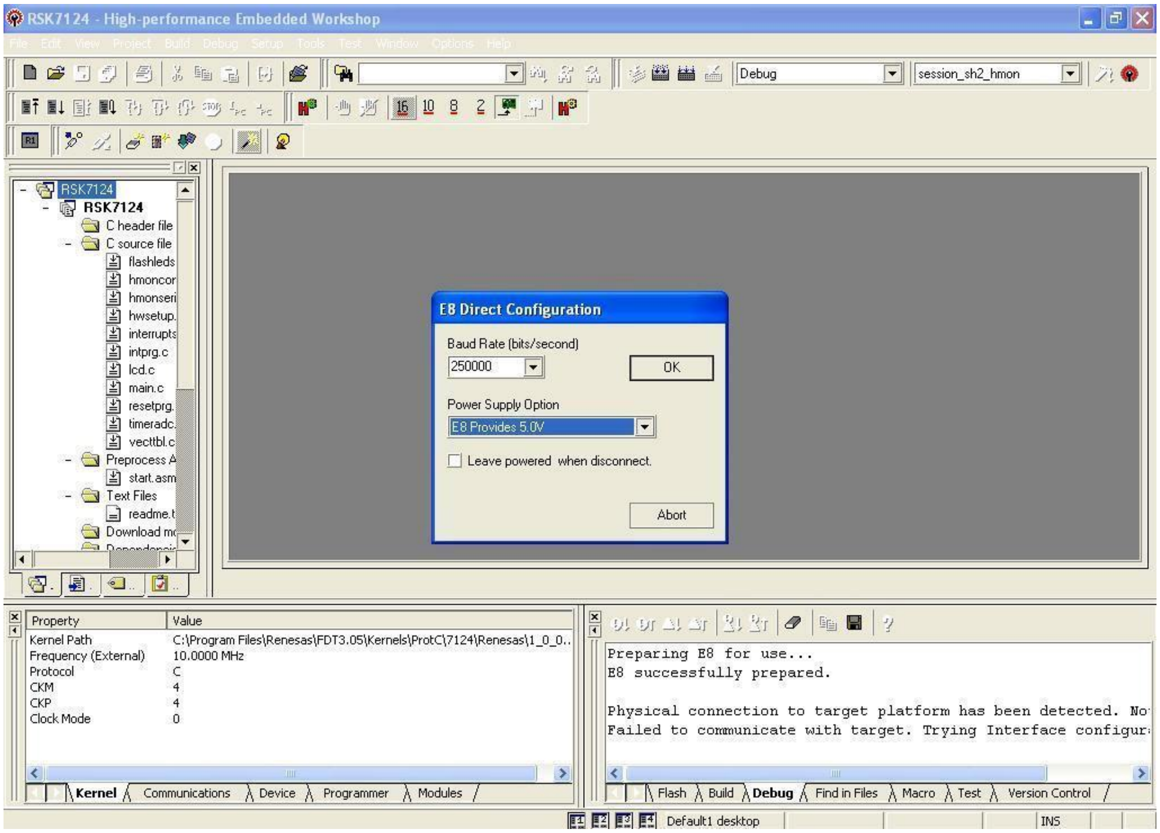
To make the connection process to the USB device quicker,
can we assume that only one USB device of this product type will be connected at any one time?

Yes No

Property	Value
Kernel Path	C:\Program Files\Renesas\FDT3.05\Kernels\ProtC\7124\Renesas\1_0_0..
Frequency (External)	10.0000 MHz
Protocol	C
CKM	4
CKP	4
Clock Mode	0

Step 10

Pop up window "Select USB Device" will be displayed. In the window "E8Direct - 0 (Closed)" should be present. Select "E8Direct - 0 (Closed)" and then press "OK". Ensure that the baud rate is set to "250000" and the "Power Supply Option" is "E8 Provides 5.0V".



E8 Direct Configuration

Baud Rate (bits/second): 250000

Power Supply Option: E8 Provides 5.0V

□ Leave powered when disconnect.

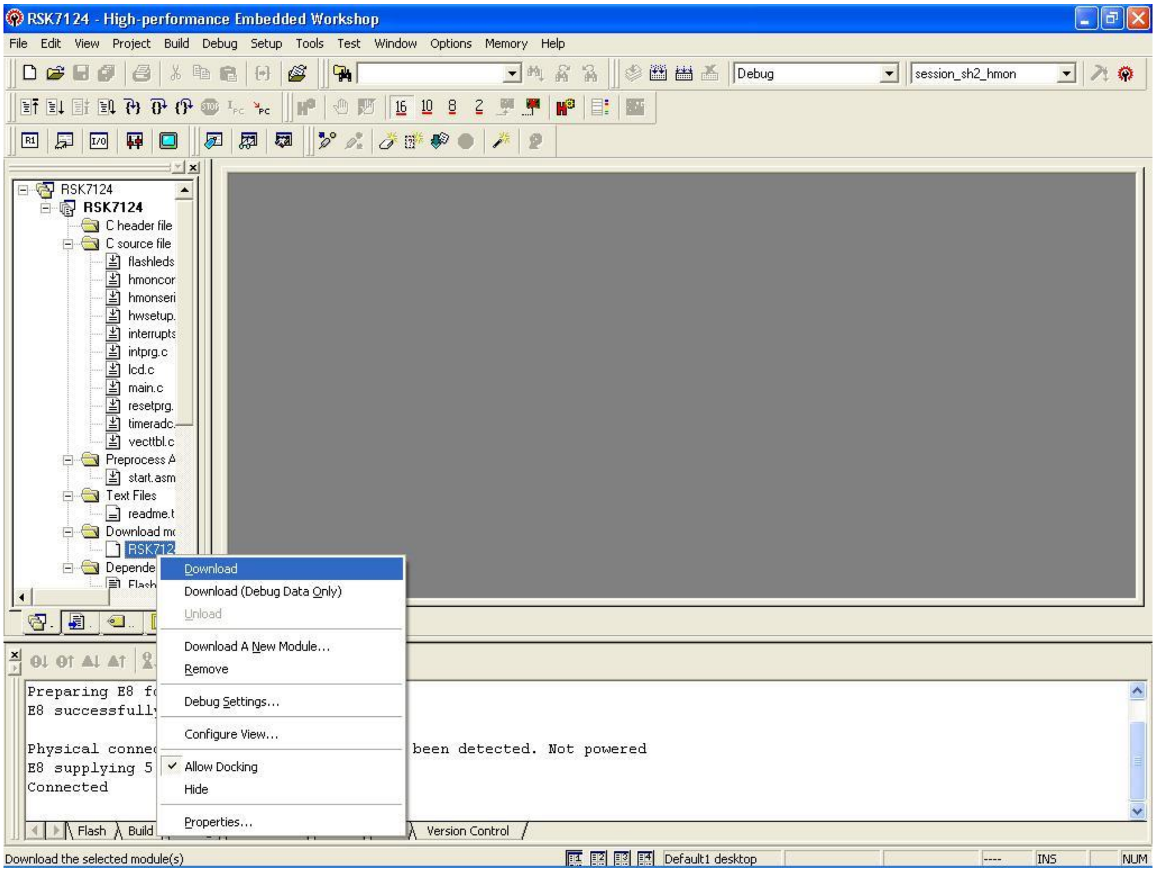
Property	Value
Kernel Path	C:\Program Files\Renesas\FDT3.05\Kernels\ProtC\7124\Renesas\1_0_0..
Frequency (External)	10.0000 MHz
Protocol	C
CKM	4
CKP	4
Clock Mode	0

```
Preparing E8 for use...
E8 successfully prepared.

Physical connection to target platform has been detected. No
Failed to communicate with target. Trying Interface configur
```

Step 11

If downloading for the first time: Click "OK" on the warning message displayed as "The E8Direct connection has only been tested to work with the following board: RSKSH7124. Damage may occur if it is used with other hardware". Click "Finish" on the next window that appears.

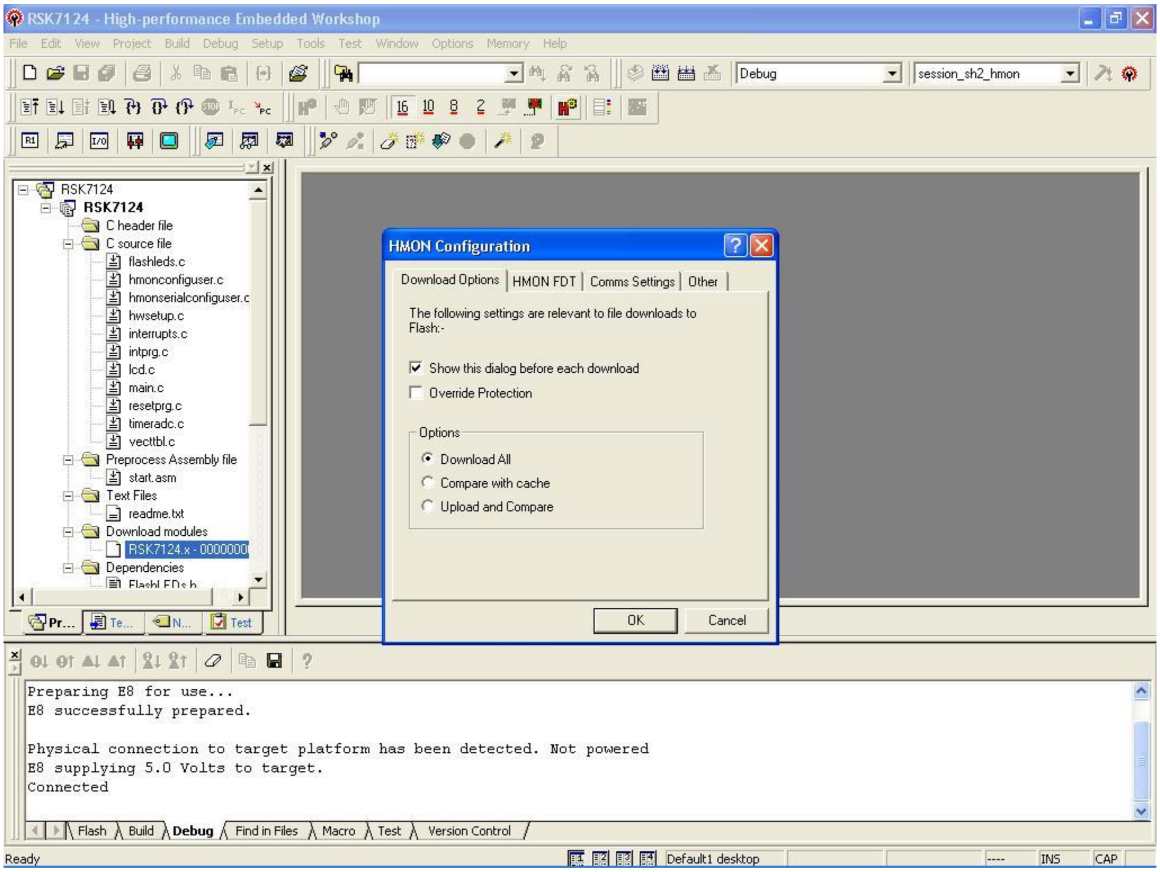
Then right click on "download modules" from right plane of HEW IDE and click "download".



```
Preparing E8 f(
E8 successfully

Physical connec                    been detected. Not powered
E8 supplying 5
Connected
```

Step 12

"HMON Configuration" window will appear. Under "Download Options" tab, select "Download All" option.



HMON Configuration

Download Options | HMON FDT | Comms Settings | Other

The following settings are relevant to file downloads to Flash:-

☑ Show this dialog before each download

☐ Override Protection

Options

● Download All

○ Compare with cache

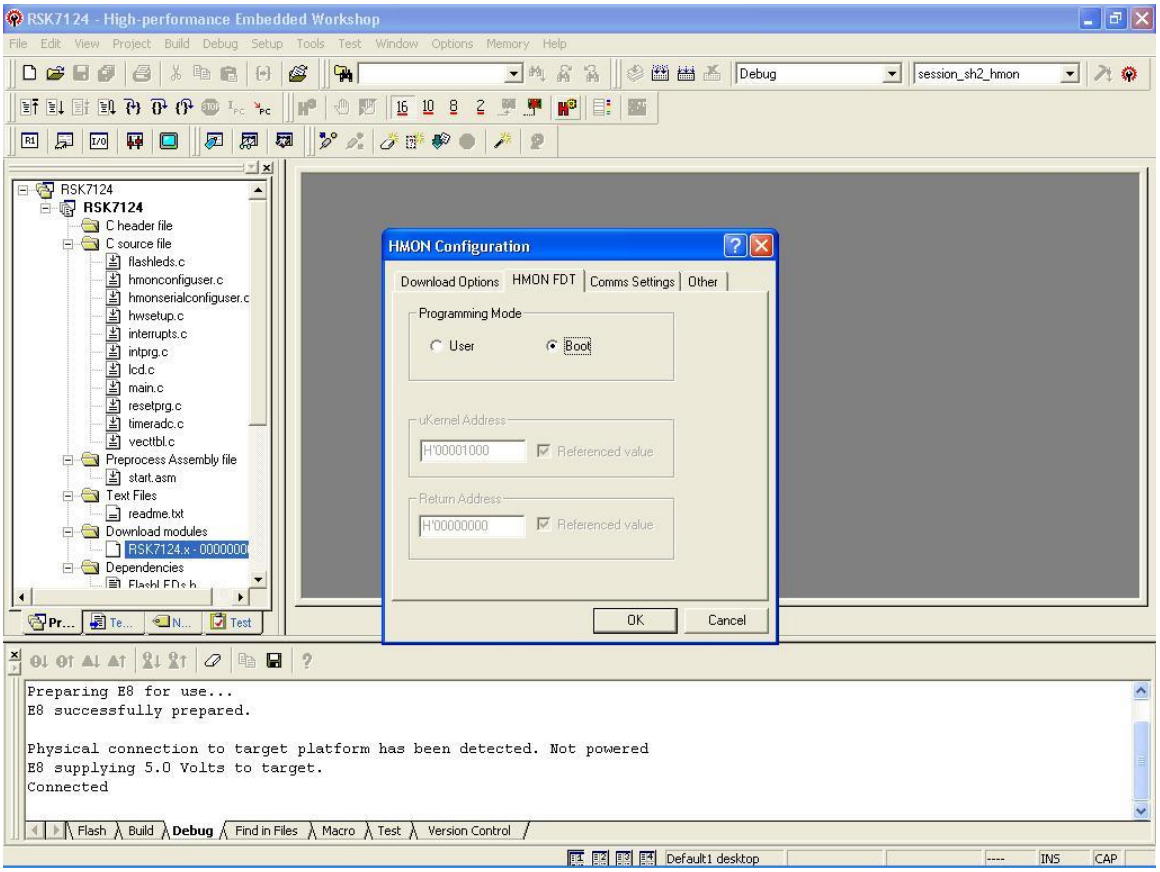
○ Upload and Compare

OK Cancel

```
Preparing E8 for use...
E8 successfully prepared.

Physical connection to target platform has been detected. Not powered
E8 supplying 5.0 Volts to target.
Connected
```

Step 13

Under "HMON FDT" tab, select "Boot" mode. And click "OK". This will allow you to download the file in boot mode to the device. Click "Save Session", from the "File" menu.



Programming Mode: ○ User ● Boot

uKernel Address: H'00001000 ☑ Referenced value

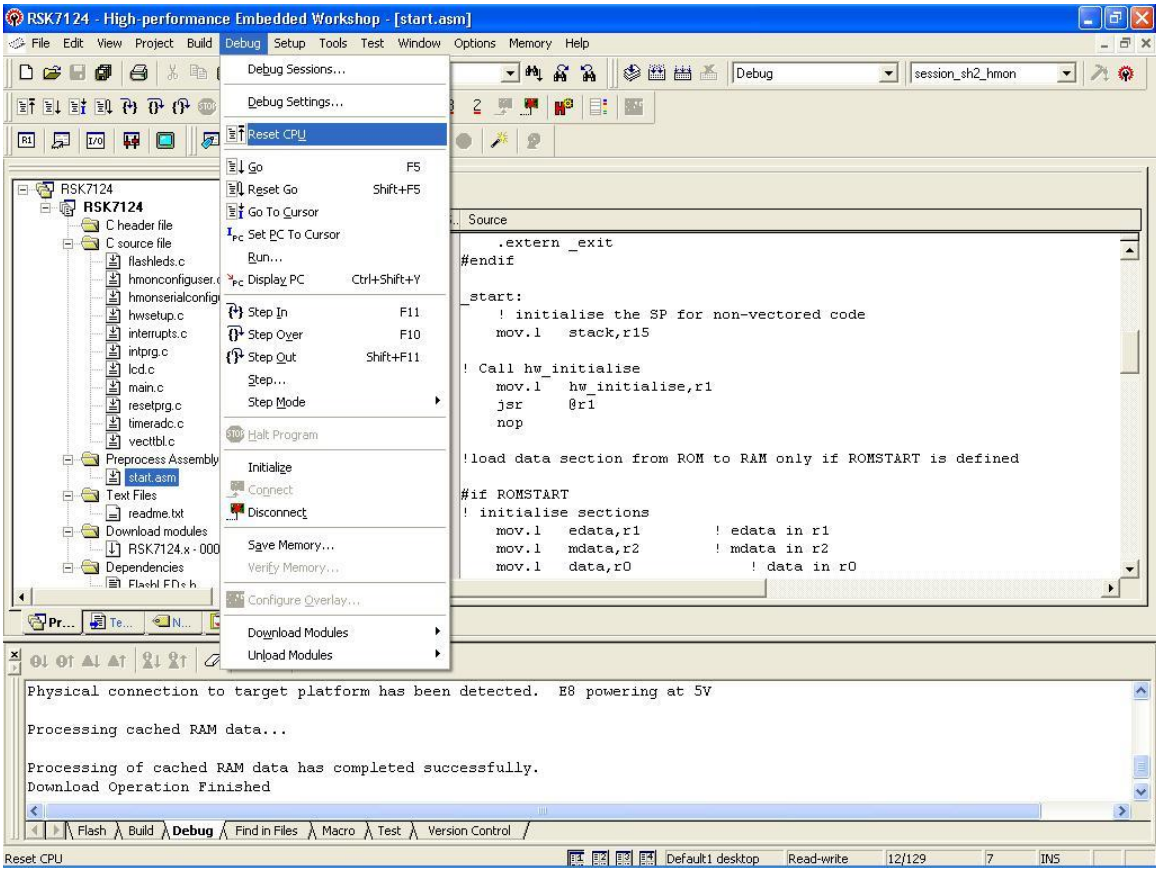
Return Address: H'00000000 ☑ Referenced value

```
Preparing E8 for use...
E8 successfully prepared.

Physical connection to target platform has been detected. Not powered
E8 supplying 5.0 Volts to target.
Connected
```

2.8.1.3 Debugging

Step 1

The code listing will appear with a margin at the left showing the areas in memory where the code has been loaded. Click on the "ResetCPU" button.

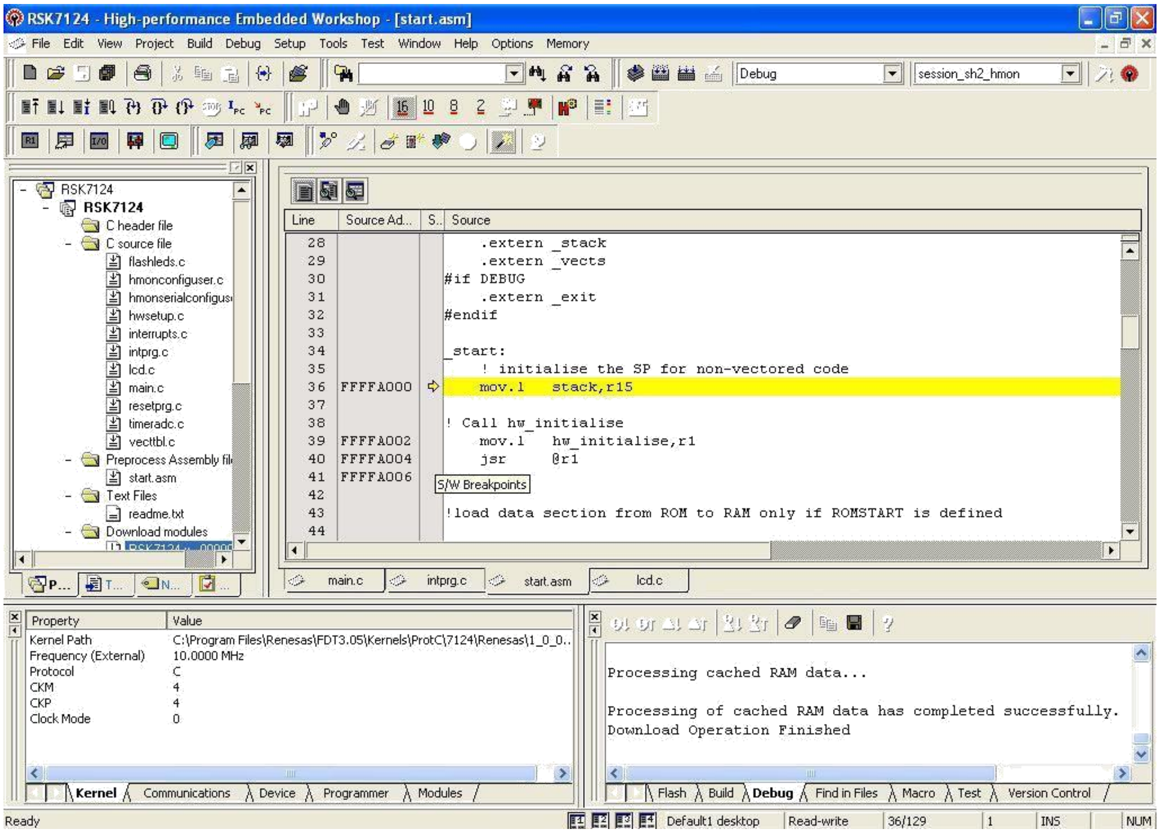


```
    .extern _exit
#endif

_start:
    ! initialise the SP for non-vectored code
    mov.l   stack,r15

! Call hw_initialise
    mov.l   hw_initialise,r1
    jsr     @r1
    nop

!load data section from ROM to RAM only if ROMSTART is defined

#if ROMSTART
! initialise sections
    mov.l   edata,r1        ! edata in r1
    mov.l   mdata,r2        ! mdata in r2
    mov.l   data,r0             ! data in r0
```

```
Physical connection to target platform has been detected.  E8 powering at 5V

Processing cached RAM data...

Processing of cached RAM data has completed successfully.
Download Operation Finished
```

Step 2

The Program Counter will go to the program start address, denoted by a yellow arrow in ―start.asm‖ file. Select the "Debug" -> "Reset Go" menu item. Now you can insert Breakpoints and debug the source code stepwise.



```
              .extern _stack
              .extern _vects
          #if DEBUG
              .extern _exit
          #endif

          _start:
              ! initialise the SP for non-vectored code
FFFFA000      mov.l   stack,r15

          ! Call hw_initialise
FFFFA002      mov.l   hw_initialise,r1
FFFFA004      jsr     @r1
FFFFA006

          !load data section from ROM to RAM only if ROMSTART is defined
```

Property	Value
Kernel Path	C:\Program Files\Renesas\FDT3.05\Kernels\ProtC\7124\Renesas\1_0_0..
Frequency (External)	10.0000 MHz
Protocol	C
CKM	4
CKP	4
Clock Mode	0

```
Processing cached RAM data...

Processing of cached RAM data has completed successfully.
Download Operation Finished
```

2.9 Trouble Shooting with Simulator Debugging

GDB, the GNU debugger, is a tool designed to help locate the cause of bugs in a computer program. To use it, the program‘s source files must be compiled with the -g option enabled. This instructs the compiler to generate extra information which is used by the debugger. GDB allows you to set breakpoints in the program being debugged. These will cause the program‘s execution to stop as soon as one of them is reached. It is then possible to examine the state of the program before resuming execution. It is also possible to step through the program a single line at a time, watching what happens as it does so.

The following sections describe the method to debug a program using GDBSH.

2.10 Debugging using GDBSH

Following sections explain how to use Command Line GDBSH.

2.10.1 Summary of GDB, the GNU Debugger

The purpose of GDB is to allow you to see what is going on inside another program while it executes or, what another program was doing at the moment it stopped. The GDB can do four things to help you catch ―bugs‖.

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another problem affecting your program

2.10.2 GDB as Free Software

The GNU debugger, GDB, is *free software*, protected by the GNU General Public License (GPL). The GPL gives you the freedom to copy or adapt a licensed program. Every person getting a copy also gets with it the freedom to modify that copy (which means that they must get access to the source code) and the freedom to distribute further copies. Typical software companies use copyrights to limit your freedoms; the Free Software Foundation uses the GPL to preserve these freedoms. Fundamentally, the General Public License is a license that says you have these freedoms and that you cannot take these freedoms away from anyone else.

2.10.3 Requirements of GDB

Before using GDB, you should understand the formal requirements and other expectations for GDB. Although some of these may seem obvious, there have been proposals for GDB that have run counter to these requirements. First of all, GDB is a debugger. It‘s not designed to be a front panel for embedded systems.

It‘s not a text editor. It‘s not a shell. It‘s not a programming environment. GDB is an interactive tool. Although a batch mode is available, GDB‘s primary role is to interact with a human programmer. The GDB should be responsive to the user. A programmer hot on the trail of a nasty bug, and operating under a looming deadline, is going to be very impatient of everything, including the response time to debugger commands. GDB should be relatively permissive, such as for expressions. While the compiler should be picky (or have the option to be made picky), since source code lives for a long time usually, the programmer doing debugging shouldn‘t be spending time figuring out to mollify the debugger. GDB will be called upon to deal with really large programs. Executable sizes of 50 to 100 megabytes occur regularly, and there are reports of programs approaching 1 gigabyte in size. GDB should be able to run everywhere. No other debugger is available for even half as many configurations as GDB supports.

2.10.4 Startup GDBSH

2.10.4.1 Using GDBSH on simulator

The example session given below describes the debugging of a simple program on simulator using GDBSH. This example is run on the sh-elf toolchain command prompt (from Windows Start menu).

Step 1

Write a ¯Hello World!‖ program in a file hello.c

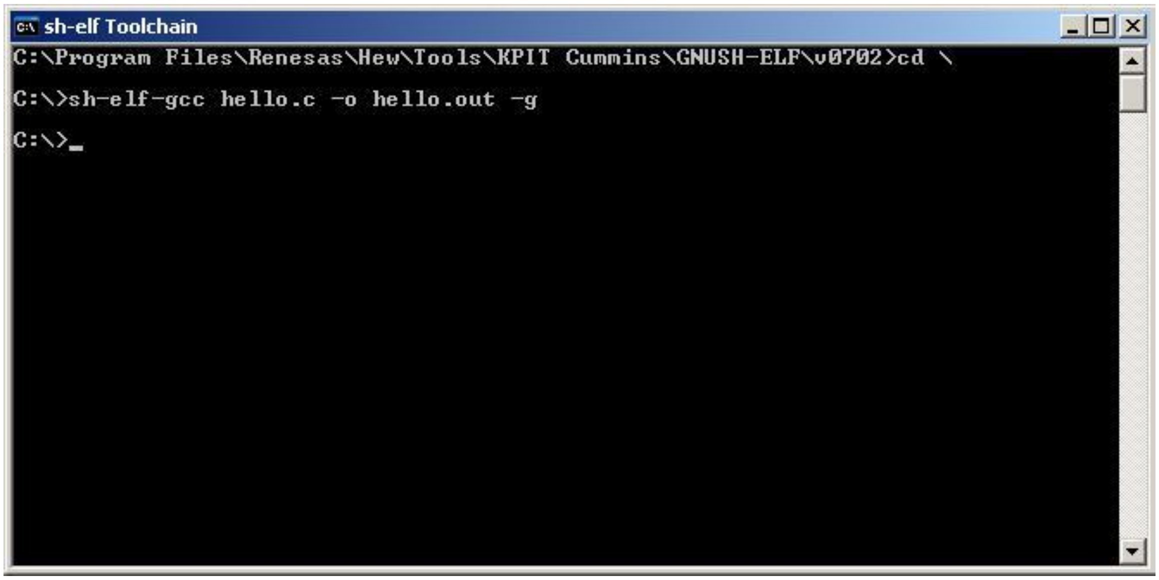
```
#include <stdio.h>
int main(void)
{
  int ret;
  ret = printf("Hello World!\r\n");
  return ret;
}
```

Step 2

Compile the program from the sh-elf command line as follows:

```
#sh-elf-gcc hello.c -o hello.out -g
```

Above command will generate an executable file called hello.out.



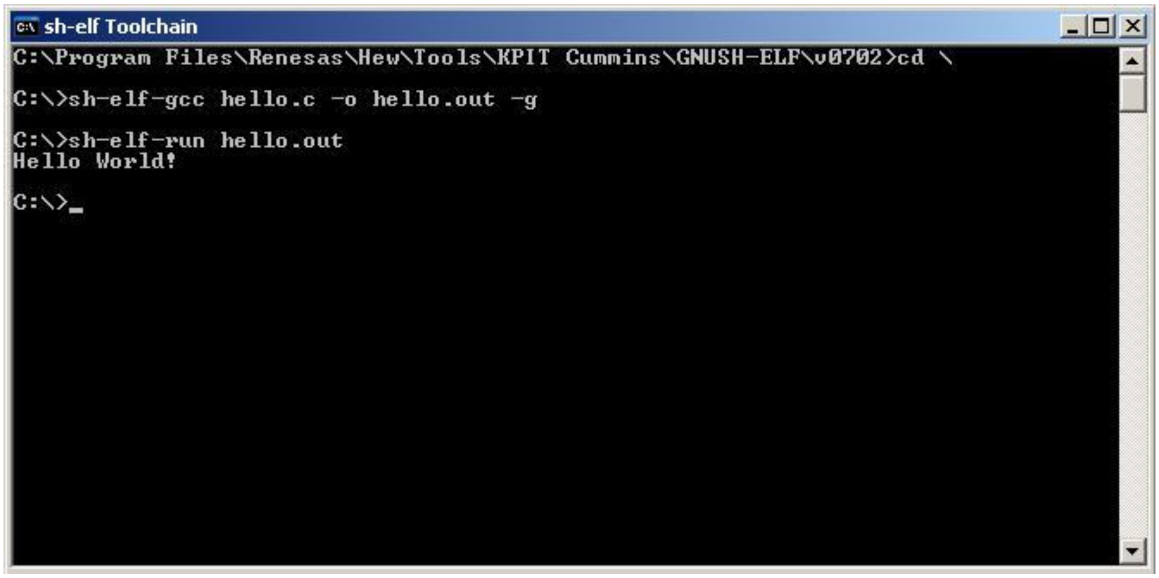
```
C:\Program Files\Renesas\Hew\Tools\KPIT Cummins\GNUSH-ELF\v0702>cd \

C:\>sh-elf-gcc hello.c -o hello.out -g

C:\>_
```

Step 3

Run the program using sh-elf simulator as follows:

```
#sh-elf-run hello.out
```

Execution of the program will print "Hello World!" string on the console.



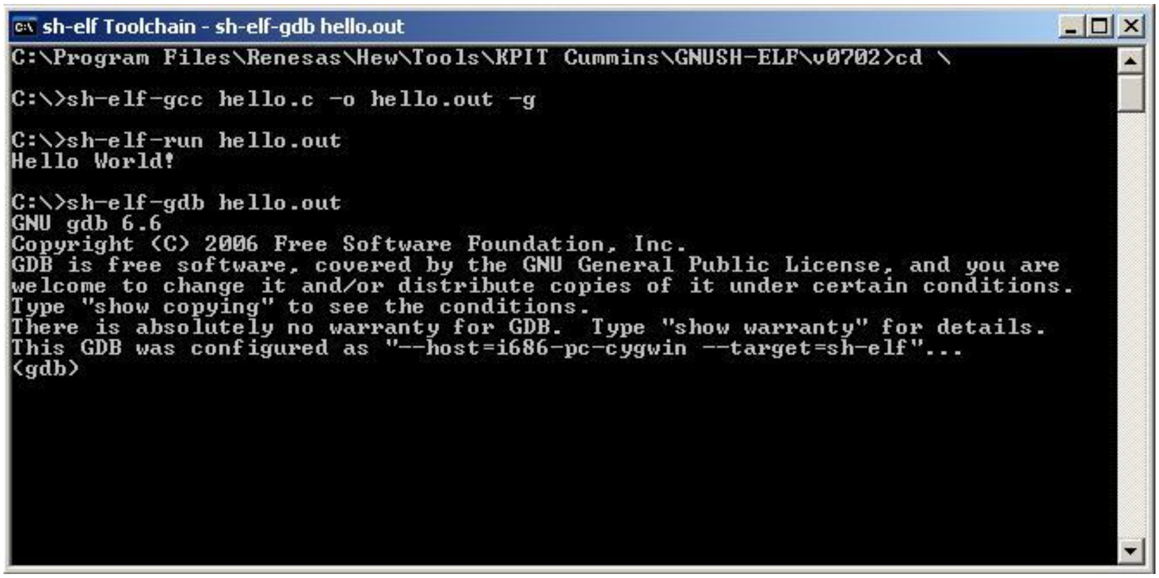
```
C:\Program Files\Renesas\Hew\Tools\KPIT Cummins\GNUSH-ELF\v0702>cd \

C:\>sh-elf-gcc hello.c -o hello.out -g

C:\>sh-elf-run hello.out
Hello World!

C:\>_
```

Step 4

To use GDBSH, type the following command line:

```
#sh-elf-gdb hello.out
```

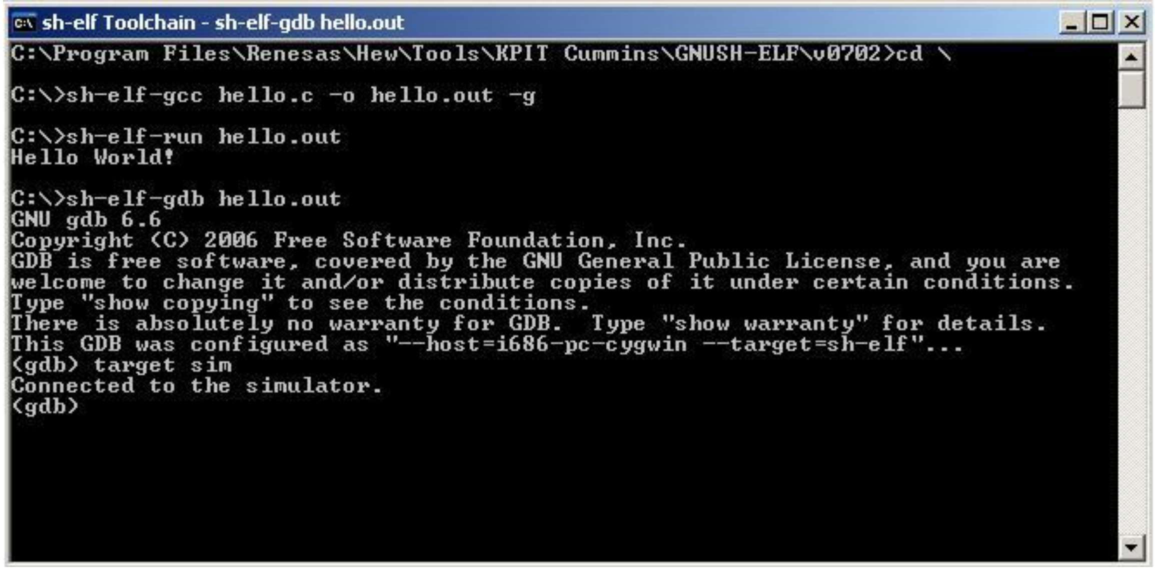
The GDB prompt <gdb> will appear on the console.



```
C:\Program Files\Renesas\Hew\Tools\KPIT Cummins\GNUSH-ELF\v0702>cd \

C:\>sh-elf-gcc hello.c -o hello.out -g

C:\>sh-elf-run hello.out
Hello World!

C:\>sh-elf-gdb hello.out
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=sh-elf"...
(gdb)
```

Step 5

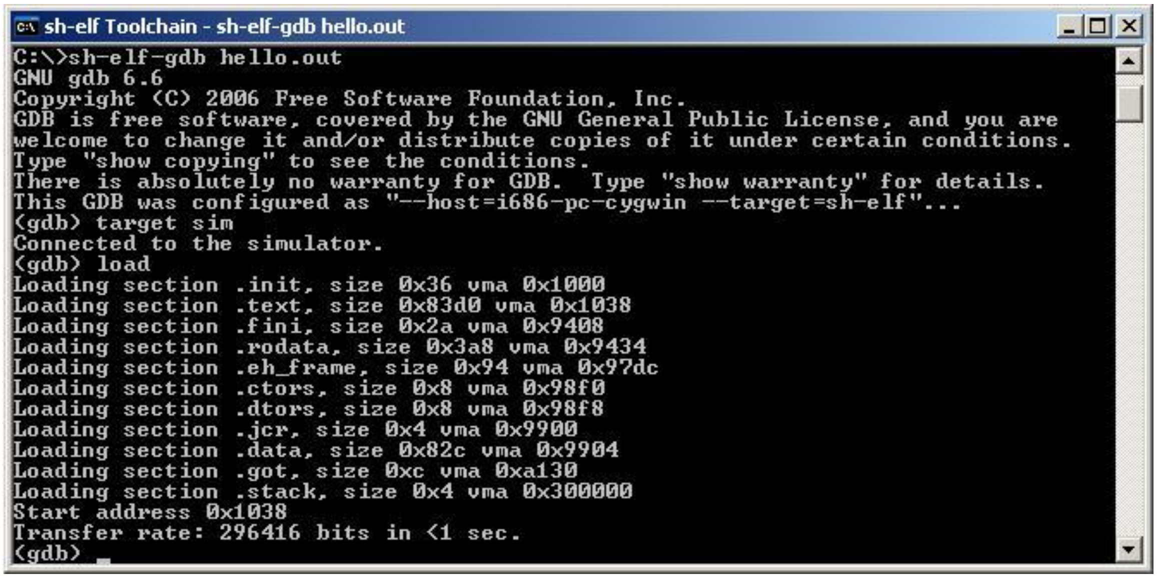
To specify the target on which to debug the executable (in this case the simulator) type ―target sim‖. This will connect the gdb to the target simulator and ―Connected to the simulator‖ message will appear on the console.



```
C:\Program Files\Renesas\Hew\Tools\KPIT Cummins\GNUSH-ELF\v0702>cd \

C:\>sh-elf-gcc hello.c -o hello.out -g

C:\>sh-elf-run hello.out
Hello World!

C:\>sh-elf-gdb hello.out
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=sh-elf"...
(gdb) target sim
Connected to the simulator.
(gdb)
```

Step 6

Type "load" on the command prompt to load the program in the memory.

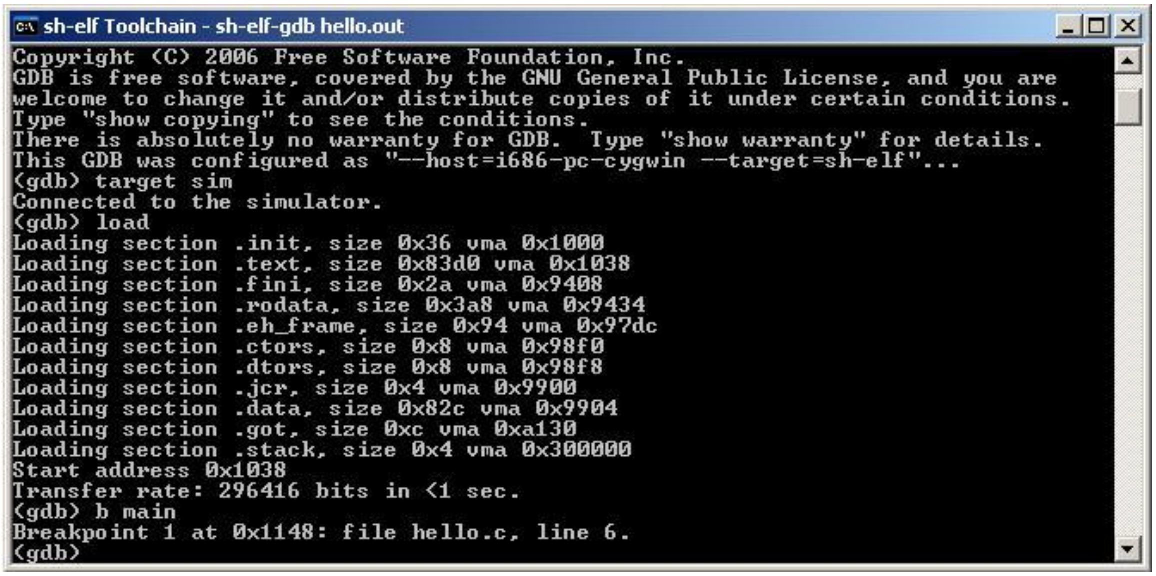


```
C:\>sh-elf-gdb hello.out
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=sh-elf"...
(gdb) target sim
Connected to the simulator.
(gdb) load
Loading section .init, size 0x36 vma 0x1000
Loading section .text, size 0x83d0 vma 0x1038
Loading section .fini, size 0x2a vma 0x9408
Loading section .rodata, size 0x3a8 vma 0x9434
Loading section .eh_frame, size 0x94 vma 0x97dc
Loading section .ctors, size 0x8 vma 0x98f0
Loading section .dtors, size 0x8 vma 0x98f8
Loading section .jcr, size 0x4 vma 0x9900
Loading section .data, size 0x82c vma 0x9904
Loading section .got, size 0xc vma 0xa130
Loading section .stack, size 0x4 vma 0x300000
Start address 0x1038
Transfer rate: 296416 bits in <1 sec.
(gdb) _
```

Step 7

Type command "b main" on the command prompt to insert a breakpoint at function main. A message "Breakpoint 1 at 0x1148: file hello.c, line 6." will appear on the console window.

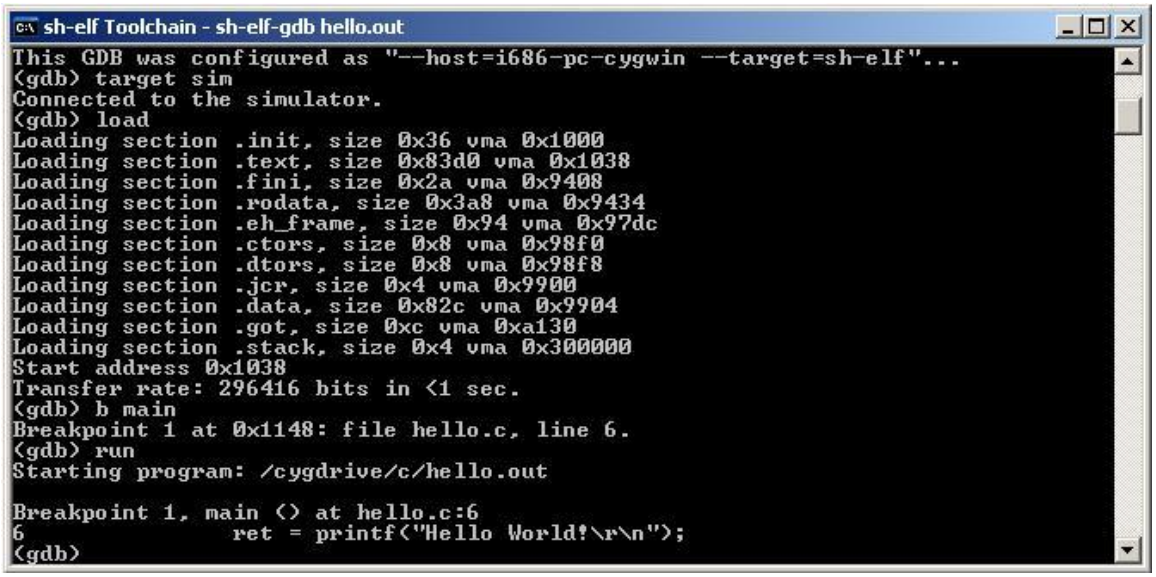
Note: that the exact address and line number may vary, depending upon the target architecture being simulated and the exact layout of the C code in the hello.c file.



```
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=sh-elf"...
(gdb) target sim
Connected to the simulator.
(gdb) load
Loading section .init, size 0x36 vma 0x1000
Loading section .text, size 0x83d0 vma 0x1038
Loading section .fini, size 0x2a vma 0x9408
Loading section .rodata, size 0x3a8 vma 0x9434
Loading section .eh_frame, size 0x94 vma 0x97dc
Loading section .ctors, size 0x8 vma 0x98f0
Loading section .dtors, size 0x8 vma 0x98f8
Loading section .jcr, size 0x4 vma 0x9900
Loading section .data, size 0x82c vma 0x9904
Loading section .got, size 0xc vma 0xa130
Loading section .stack, size 0x4 vma 0x300000
Start address 0x1038
Transfer rate: 296416 bits in <1 sec.
(gdb) b main
Breakpoint 1 at 0x1148: file hello.c, line 6.
(gdb)
```

Step 8

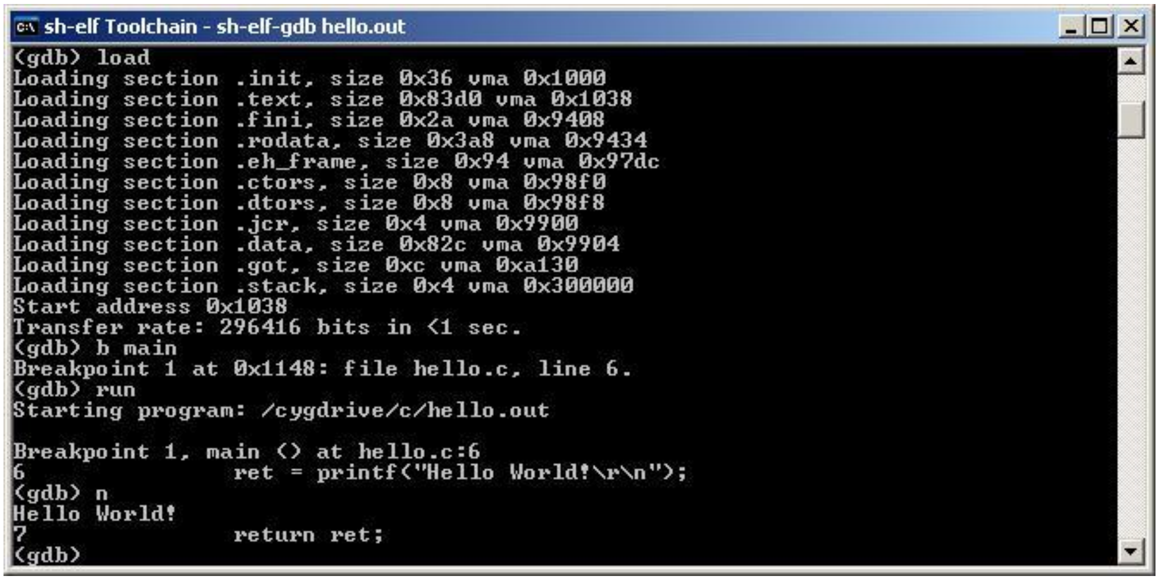
Type "run" to run the program. The program will stop at line 6 of the hello.c.



```
This GDB was configured as "--host=i686-pc-cygwin --target=sh-elf"...
(gdb) target sim
Connected to the simulator.
(gdb) load
Loading section .init, size 0x36 vma 0x1000
Loading section .text, size 0x83d0 vma 0x1038
Loading section .fini, size 0x2a vma 0x9408
Loading section .rodata, size 0x3a8 vma 0x9434
Loading section .eh_frame, size 0x94 vma 0x97dc
Loading section .ctors, size 0x8 vma 0x98f0
Loading section .dtors, size 0x8 vma 0x98f8
Loading section .jcr, size 0x4 vma 0x9900
Loading section .data, size 0x82c vma 0x9904
Loading section .got, size 0xc vma 0xa130
Loading section .stack, size 0x4 vma 0x300000
Start address 0x1038
Transfer rate: 296416 bits in <1 sec.
(gdb) b main
Breakpoint 1 at 0x1148: file hello.c, line 6.
(gdb) run
Starting program: /cygdrive/c/hello.out

Breakpoint 1, main () at hello.c:6
6               ret = printf("Hello World!\r\n");
(gdb)
```

Step 9

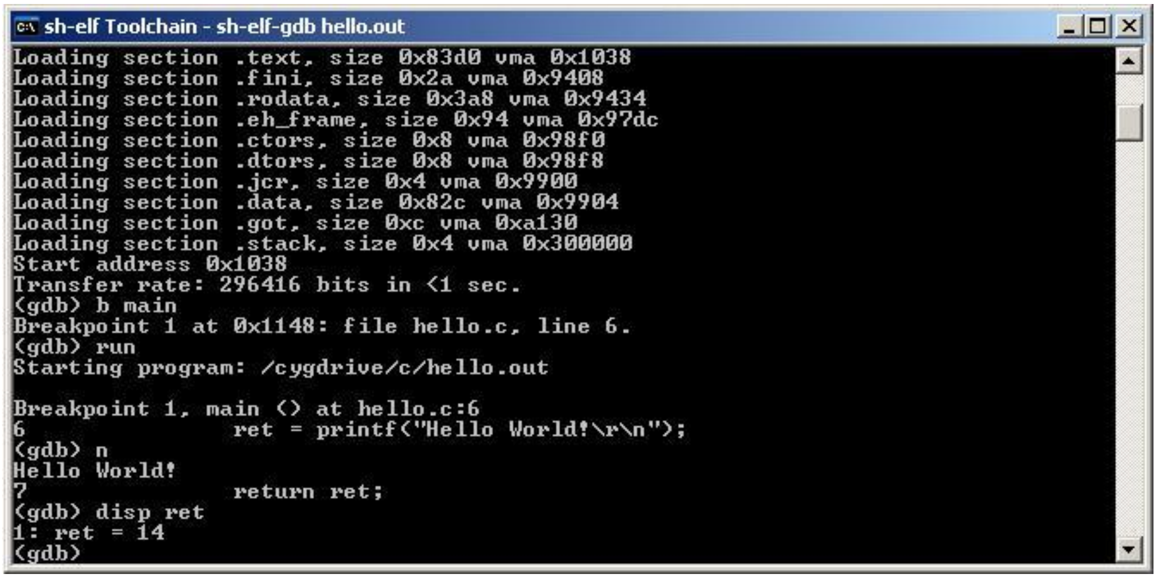
Press 'n' to go to next line. Pressing 'n' will print "Hello World!" on console and program will stop at line 7.



```
(gdb) load
Loading section .init, size 0x36 vma 0x1000
Loading section .text, size 0x83d0 vma 0x1038
Loading section .fini, size 0x2a vma 0x9408
Loading section .rodata, size 0x3a8 vma 0x9434
Loading section .eh_frame, size 0x94 vma 0x97dc
Loading section .ctors, size 0x8 vma 0x98f0
Loading section .dtors, size 0x8 vma 0x98f8
Loading section .jcr, size 0x4 vma 0x9900
Loading section .data, size 0x82c vma 0x9904
Loading section .got, size 0xc vma 0xa130
Loading section .stack, size 0x4 vma 0x300000
Start address 0x1038
Transfer rate: 296416 bits in <1 sec.
(gdb) b main
Breakpoint 1 at 0x1148: file hello.c, line 6.
(gdb) run
Starting program: /cygdrive/c/hello.out

Breakpoint 1, main () at hello.c:6
6               ret = printf("Hello World!\r\n");
(gdb) n
Hello World!
7               return ret;
(gdb)
```

Step 10

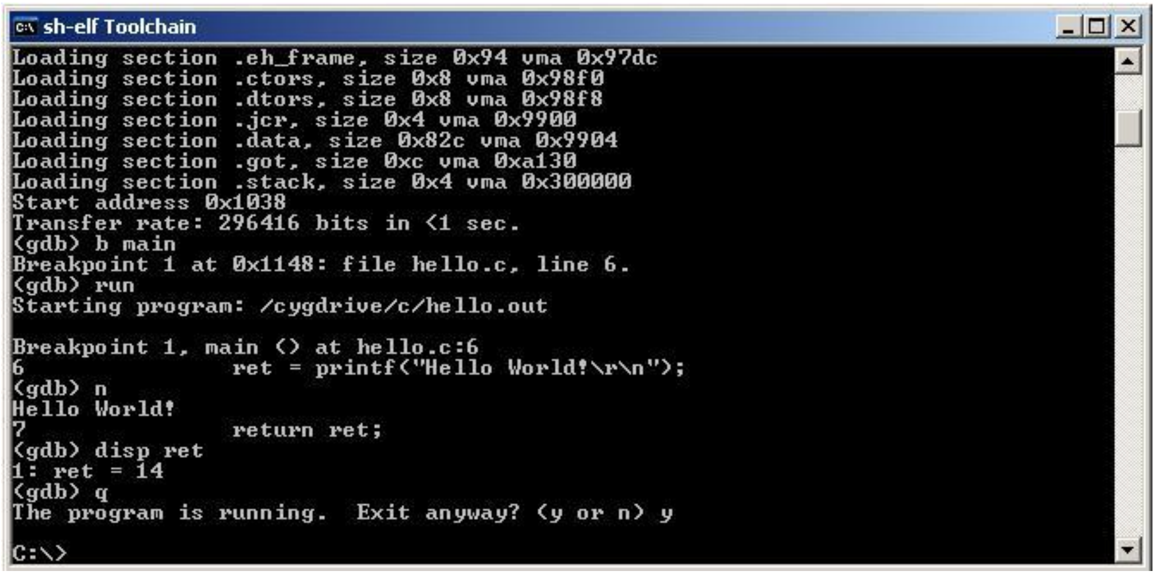
Type "disp ret" to view the value of "ret".



```
Loading section .text, size 0x83d0 vma 0x1038
Loading section .fini, size 0x2a vma 0x9408
Loading section .rodata, size 0x3a8 vma 0x9434
Loading section .eh_frame, size 0x94 vma 0x97dc
Loading section .ctors, size 0x8 vma 0x98f0
Loading section .dtors, size 0x8 vma 0x98f8
Loading section .jcr, size 0x4 vma 0x9900
Loading section .data, size 0x82c vma 0x9904
Loading section .got, size 0xc vma 0xa130
Loading section .stack, size 0x4 vma 0x300000
Start address 0x1038
Transfer rate: 296416 bits in <1 sec.
(gdb) b main
Breakpoint 1 at 0x1148: file hello.c, line 6.
(gdb) run
Starting program: /cygdrive/c/hello.out

Breakpoint 1, main () at hello.c:6
6               ret = printf("Hello World!\r\n");
(gdb) n
Hello World!
7               return ret;
(gdb) disp ret
1: ret = 14
(gdb)
```

Step 11

Type ―q‖ to exit from the debugger.



```
Loading section .eh_frame, size 0x94 vma 0x97dc
Loading section .ctors, size 0x8 vma 0x98f0
Loading section .dtors, size 0x8 vma 0x98f8
Loading section .jcr, size 0x4 vma 0x9900
Loading section .data, size 0x82c vma 0x9904
Loading section .got, size 0xc vma 0xa130
Loading section .stack, size 0x4 vma 0x300000
Start address 0x1038
Transfer rate: 296416 bits in <1 sec.
(gdb) b main
Breakpoint 1 at 0x1148: file hello.c, line 6.
(gdb) run
Starting program: /cygdrive/c/hello.out

Breakpoint 1, main () at hello.c:6
6               ret = printf("Hello World!\r\n");
(gdb) n
Hello World!
7               return ret;
(gdb) disp ret
1: ret = 14
(gdb) q
The program is running.  Exit anyway? (y or n) y

C:\>
```

2.10.4.2 Using GDBSH on Hardware

Among GDBSH‘s many noteworthy features, is its ability to debug programs ¯remotely‖, in a setup where the platform running GDBSH itself (the host) is connected to the platform running the application being debugged (the target) via a serial port, network connection, or some other means. The remote debugging capability of GDBSH, allows user to step through code, set breakpoints, examine memory, and interact with the target.

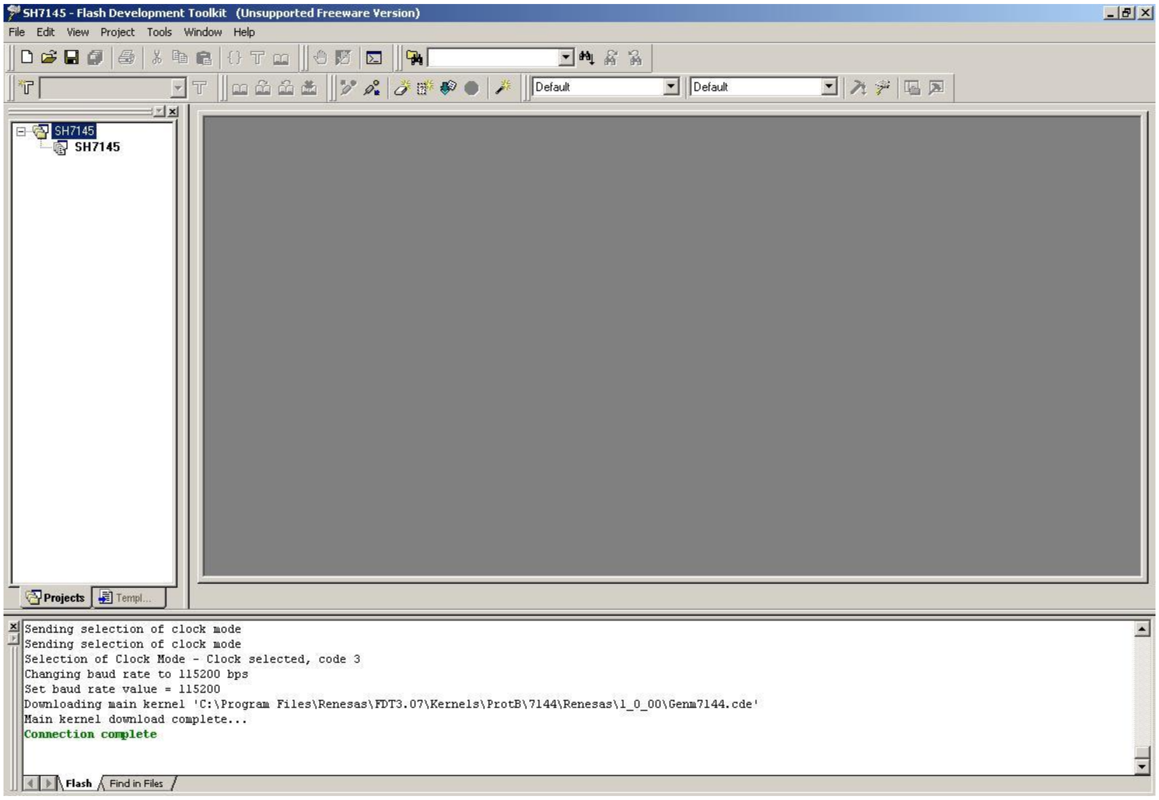
When debugging a remote target, GDBSH depends on the functionality provided by a debugging stub, a small piece of code in the embedded system that serves as the intermediary between the host running GDBSH and the application being debugged.

The debugging stub and GDBSH communicate via the GDB Remote Serial Protocol, an ASCII, message-based protocol containing commands to read and write memory, query registers, run the program, and so forth.

The example session given below describes the debugging of a simple program on hardware using GDBSH. For this example, the target hardware used is SH7145 and the processor specific GDBSH stub is used.

Step 1

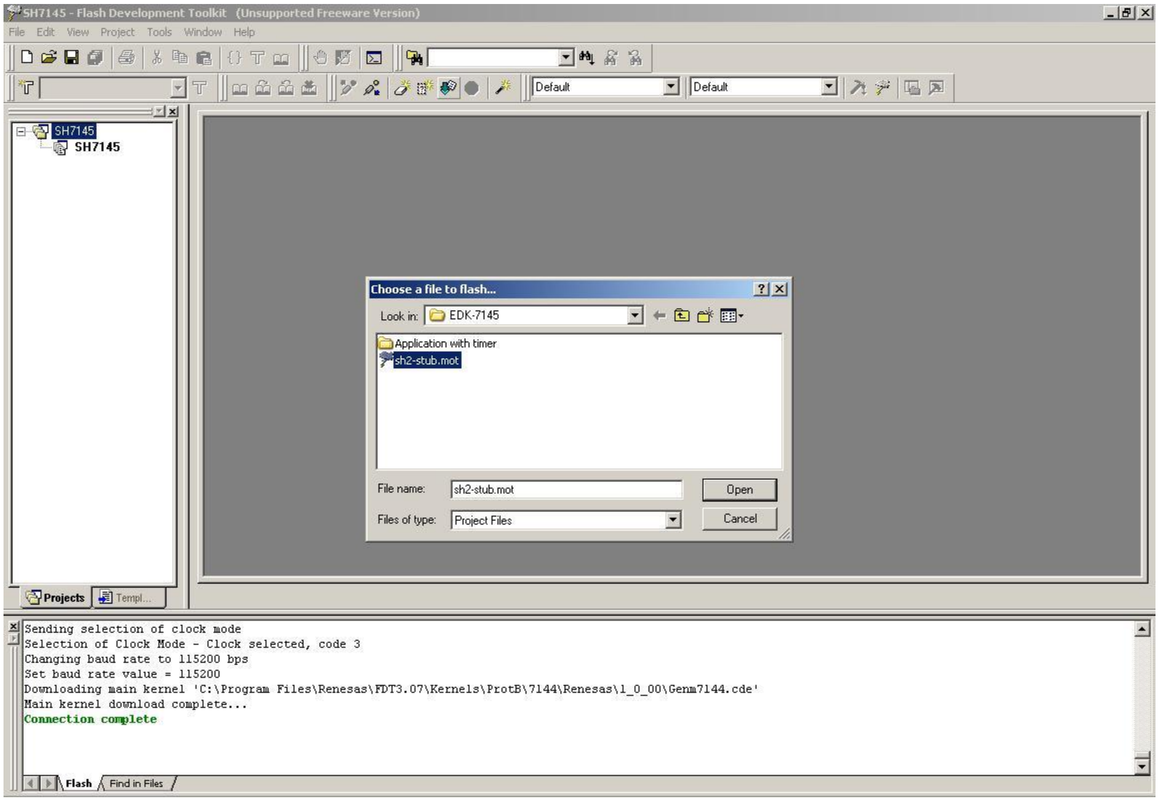
Connect the host machine to the target device using serial cable. Establish the connection using Renesas Flash Development Toolkit (FDT) as shown below.


SH7145 - Flash Development Toolkit (Unsupported Freeware Version)
File Edit View Project Tools Window Help
Default
Default
SH7145
SH7145
Projects
Templ...
Flash
Find in Files

```
Sending selection of clock mode
Sending selection of clock mode
Selection of Clock Mode - Clock selected, code 3
Changing baud rate to 115200 bps
Set baud rate value = 115200
Downloading main kernel 'C:\Program Files\Renesas\FDT3.07\Kernels\ProtB\7144\Renesas\1_0_00\Genm7144.cde'
Main kernel download complete...
Connection complete
```

Step 2

Download the .mot file of the GDBSH stub by clicking on the ―Download a File‖ button from the toolbar.



SH7145 - Flash Development Toolkit (Unsupported Freeware Version)
File Edit View Project Tools Window Help
Default
Default
SH7145
SH7145
Projects
Templ...
Flash
Find in Files

Choose a file to flash...

Look in: EDK-7145

Application with timer

sh2-stub.mot

File name: sh2-stub.mot

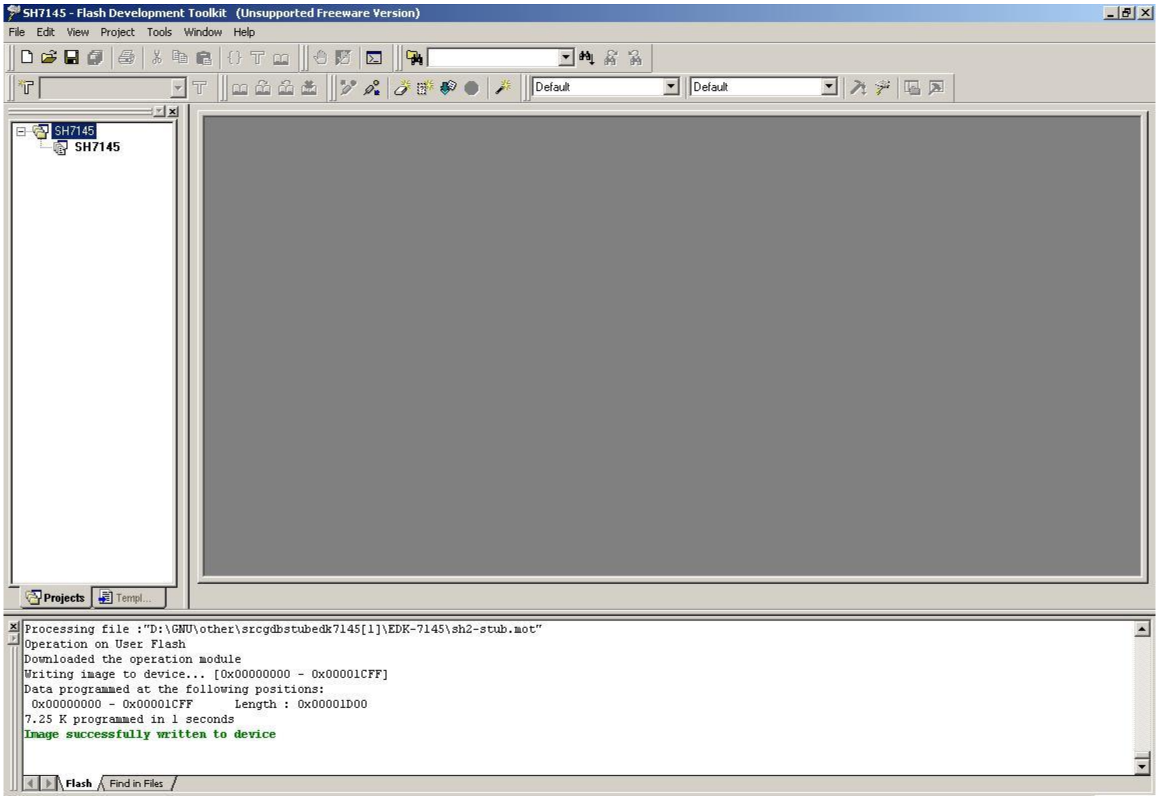
Files of type: Project Files

Open Cancel

```
Sending selection of clock mode
Selection of Clock Mode - Clock selected, code 3
Changing baud rate to 115200 bps
Set baud rate value = 115200
Downloading main kernel 'C:\Program Files\Renesas\FDT3.07\Kernels\ProtB\7144\Renesas\1_0_00\Genm7144.cde'
Main kernel download complete...
Connection complete
```

Step 3

After downloading the .mot file the message ―Image successfully written to device‖ will be displayed. Disconnect the link from FDT.



SH7145 - Flash Development Toolkit (Unsupported Freeware Version)
File Edit View Project Tools Window Help
Default
Default
SH7145
SH7145
Projects
Templ...
Flash
Find in Files

```
Processing file :"D:\GNU\other\srcgdbstubedk7145[1]\EDK-7145\sh2-stub.mot"
Operation on User Flash
Downloaded the operation module
Writing image to device... [0x00000000 - 0x00001CFF]
Data programmed at the following positions:
 0x00000000 - 0x00001CFF      Length : 0x00001D00
7.25 K programmed in 1 seconds
Image successfully written to device
```

Step 4

Write a simple program in a file test.c as follows:

```
int main(void)
{
      int ret=9;
      ret++;
      return ret;
}
```

Step 5

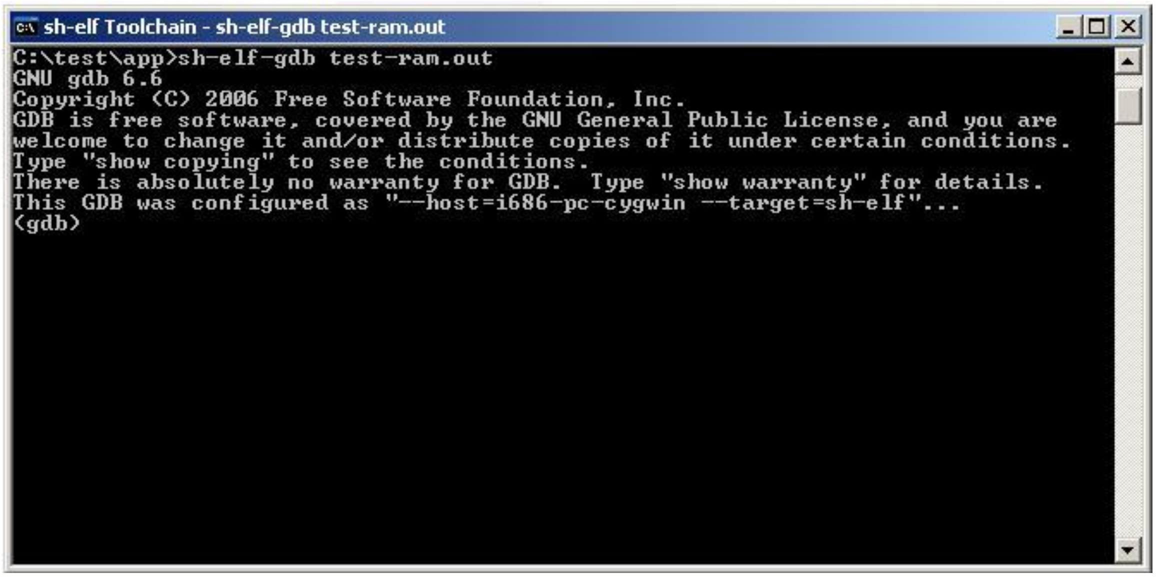
Generate the output file (.out) of the application, using the sh-elf toolchain command prompt. Pass -g option so that the .out file contains the debug information.

Step 6

To use GDBSH, type the following command line:

```
#sh-elf-gdb test-ram.out
```

The GDB prompt <gdb> will appear on the console.

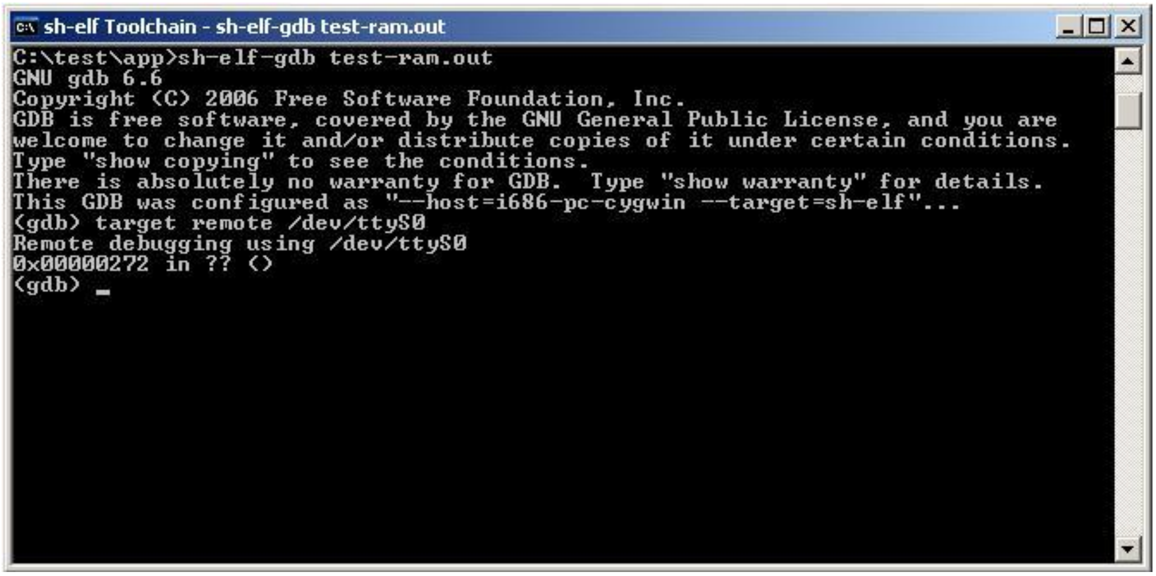


sh-elf Toolchain - sh-elf-gdb test-ram.out

```
C:\test\app>sh-elf-gdb test-ram.out
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=sh-elf"...
(gdb)
```

Step 7

Type the command ―target remote /dev/ttyS0‖ on the prompt. This will connect the gdb to the target hardware and ―Remote debugging using /dev/ttyS0‖ message will appear on the console.

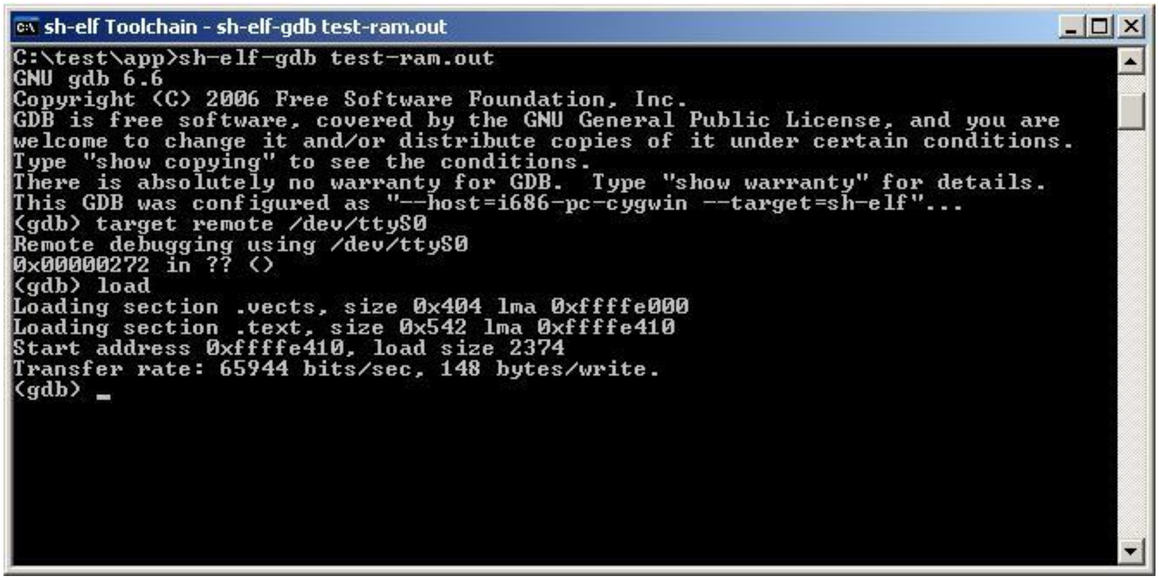


sh-elf Toolchain - sh-elf-gdb test-ram.out

```
C:\test\app>sh-elf-gdb test-ram.out
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=sh-elf"...
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
0x00000272 in ?? ()
(gdb) _
```

Step 8

Type "load" on the command prompt to load the program in the memory.

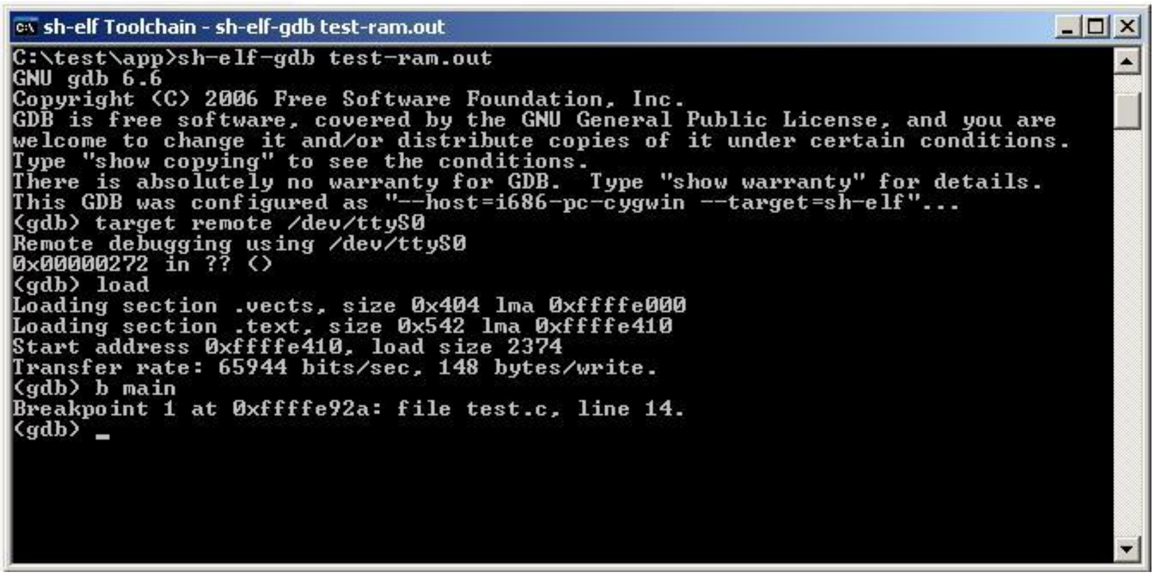


```
C:\test\app>sh-elf-gdb test-ram.out
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=sh-elf"...
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
0x00000272 in ?? ()
(gdb) load
Loading section .vects, size 0x404 lma 0xffffe000
Loading section .text, size 0x542 lma 0xffffe410
Start address 0xffffe410, load size 2374
Transfer rate: 65944 bits/sec, 148 bytes/write.
(gdb) _
```

Step 9

Type command "b main" on the command prompt to insert a breakpoint at function main. A message "Breakpoint 1 at 0xffffe92a: file test.c, line 14." will appear on the console window.

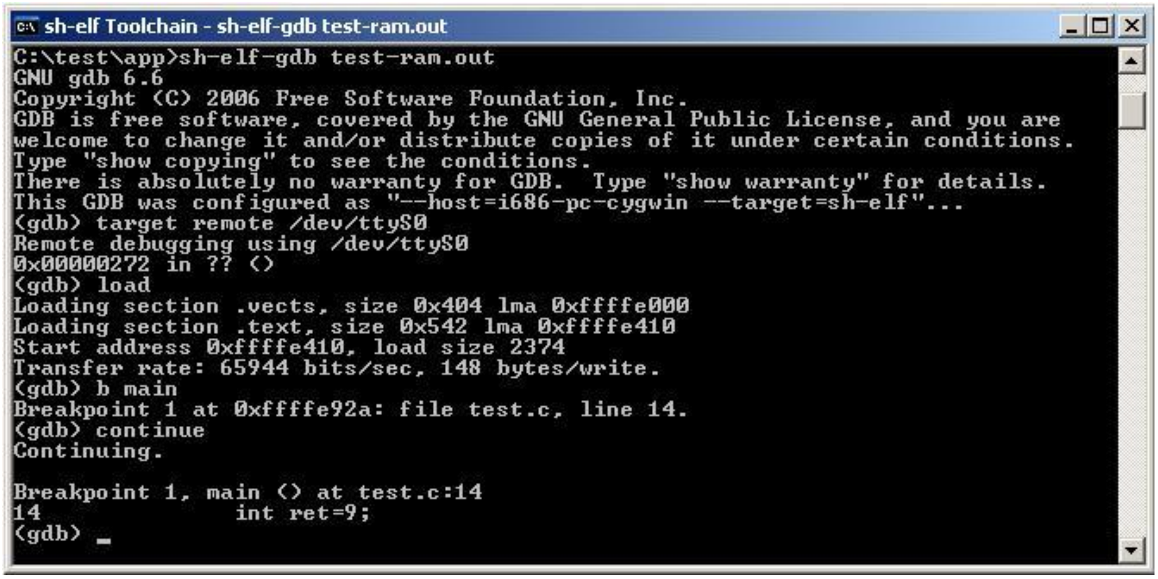
Note that the exact address and line number may vary, depending upon the target architecture being simulated and the exact layout of the C code in the test.c file.



```
C:\test\app>sh-elf-gdb test-ram.out
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=sh-elf"...
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
0x00000272 in ?? ()
(gdb) load
Loading section .vects, size 0x404 lma 0xffffe000
Loading section .text, size 0x542 lma 0xffffe410
Start address 0xffffe410, load size 2374
Transfer rate: 65944 bits/sec, 148 bytes/write.
(gdb) b main
Breakpoint 1 at 0xffffe92a: file test.c, line 14.
(gdb) _
```

Step 10

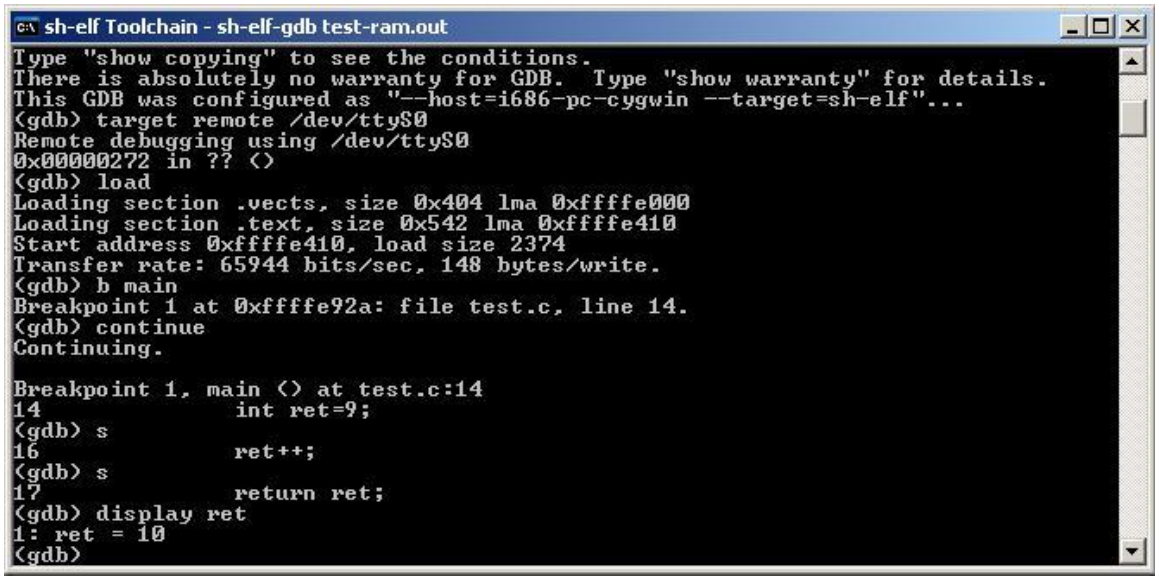
Type "Continue" to run the program. The program will stop at line 14 of the test.c.



```
C:\test\app>sh-elf-gdb test-ram.out
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=sh-elf"...
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
0x00000272 in ?? ()
(gdb) load
Loading section .vects, size 0x404 lma 0xffffe000
Loading section .text, size 0x542 lma 0xffffe410
Start address 0xffffe410, load size 2374
Transfer rate: 65944 bits/sec, 148 bytes/write.
(gdb) b main
Breakpoint 1 at 0xffffe92a: file test.c, line 14.
(gdb) continue
Continuing.

Breakpoint 1, main () at test.c:14
14              int ret=9;
(gdb) _
```

Step 11

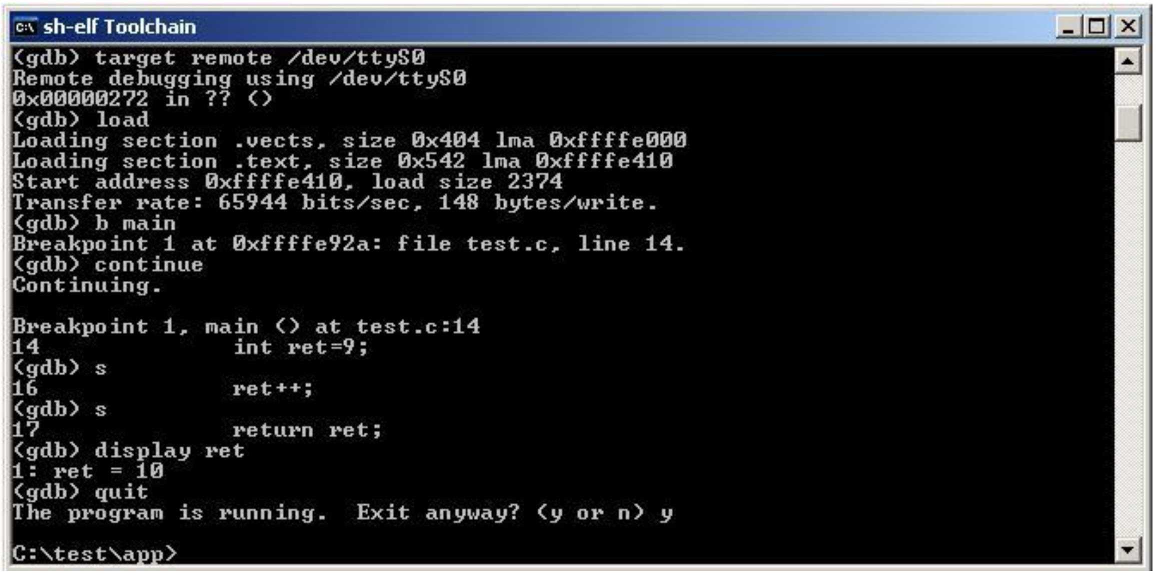
Press 's' to go to next line. Type "display ret" to view the value of "ret".



```
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=sh-elf"...
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
0x00000272 in ?? ()
(gdb) load
Loading section .vects, size 0x404 lma 0xffffe000
Loading section .text, size 0x542 lma 0xffffe410
Start address 0xffffe410, load size 2374
Transfer rate: 65944 bits/sec, 148 bytes/write.
(gdb) b main
Breakpoint 1 at 0xffffe92a: file test.c, line 14.
(gdb) continue
Continuing.

Breakpoint 1, main () at test.c:14
14              int ret=9;
(gdb) s
16              ret++;
(gdb) s
17              return ret;
(gdb) display ret
1: ret = 10
(gdb)
```

Step 12

Type ―quit‖ to exit from the debugger.



```
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
0x00000272 in ?? ()
(gdb) load
Loading section .vects, size 0x404 lma 0xffffe000
Loading section .text, size 0x542 lma 0xffffe410
Start address 0xffffe410, load size 2374
Transfer rate: 65944 bits/sec, 148 bytes/write.
(gdb) b main
Breakpoint 1 at 0xffffe92a: file test.c, line 14.
(gdb) continue
Continuing.

Breakpoint 1, main () at test.c:14
14              int ret=9;
(gdb) s
16              ret++;
(gdb) s
17              return ret;
(gdb) display ret
1: ret = 10
(gdb) quit
The program is running.  Exit anyway? (y or n) y

C:\test\app>
```

2.10.5 GDB Comprehensive Quick Reference

Essential Commands

COMMAND SYNTAX	DESCRIPTION
GDB *program* [*core*]	debug *program* [using coredump *core*]
b [*file*:]*function*	set breakpoint at *function* [in *file*]
run [*arglist*]	start your program [with *arglist*]
bt	backtrace: display program stack
p expr	display the value of an expression
c	continue running your program
n	next line, stepping over function calls
S	next line, stepping into function calls
info stack	view the call stack
print *(int*)0x10000	display memory location
set *(int*)0x1000=0x123	set memory
info registers	display all CPU registers
print $<reg-name>	displays register value
set $<reg-name>=<value>	sets register value
step	runs next line of code (step into)
stepi	runs next instruction
next	runs next instruction, but doesn‘t enter (step over)
info br	list breakpoints
br Init	insert breakpoint on the Init function

COMMAND SYNTAX	DESCRIPTION
br 33	insert breakpoint on line # 33
list Init	show source code listing
disassemble	shows assembly code of higher level code
file <name-of-object>	loads file for debugging
load	loads object sections
target <name>	selects <name> = remote, serial, sim, gdbserver, etc.

run	run a program in the simulator -- before using this command, a sequence must be executed as follows:
	$ <target-alias>-gdb <filename> (object file with debug symbols)
	$ target sim
	$ load
	$ run

x / FMT	display modifier – where FMT is x – print in hex d – print in decimal s - print as string w – print in 32-bit words h – print in 16-bit words b – print in 8-bit words # - print number of items

Starting GDB

COMMAND SYNTAX	DESCRIPTION
GDB	start GDB, with no debugging files
GDB *program*	begin debugging *program*
GDB *program core*	debug coredump *core* produced by *program*
GDB --help	describe command line options

Stopping GDB

COMMAND SYNTAX	DESCRIPTION
quit	exit GDB; also q or EOF (e.g., C-d)
INTERRUPT	(eg C-c) terminate current command, or send to running process

Getting Help

COMMAND SYNTAX	DESCRIPTION
help	list classes of commands
help *class*	one-line descriptions for commands in *class*
help *command*	describe *command*

Executing your Program

COMMAND SYNTAX	DESCRIPTION
run *arglist*	start your program with *arglist*
run	start your program with current argument list
run…<*inf*>*outf*	start your program with input, output redirected
kill	kill running program
tty dev	use dev as stdin and stdout for next run
set args *arglist*	specify *arglist* for next run
set args	specify empty argument list
show args	display argument list
show env	show all environment variables
show env *var*	show value of environment variable *var*

COMMAND SYNTAX	DESCRIPTION
set env *var string*	set environment variable *var*
unset env *var*	remove *var* from environment

Shell Commands

COMMAND SYNTAX	DESCRIPTION
cd dir	change working directory to dir
pwd	print working directory
make …	call \make"
shell *cmd*	execute arbitrary shell command string

Breakpoints and Watchpoints

COMMAND SYNTAX	DESCRIPTION
break [*file*:]line	set breakpoint at line number [in *file*]
B [*file*:]line	e.g.: break main.c:37
break [*file*:]*func*	set breakpoint at *func* [in *file*]
break +*offset*	set break at *offset* lines from current stop
break -*offset*	set break at *offset* lines from current stop
break **addr*	set breakpoint at address *addr*
break	set breakpoint at next instruction
break …if *expr*	break conditionally on nonzero *expr*
cond n [expr]	new conditional expression on breakpoint n; make unconditional if no *expr*

COMMAND SYNTAX	DESCRIPTION
tbreak…	temporary break; disable when reached
rbreak *regex*	break on all functions matching *regex*
watch *expr*	set a watchpoint for expression *expr*
catch *event*	break at *event*, which may be catch, throw, exec, fork, vfork, load, or unload
info break	show defined breakpoints
info watch	show defined watchpoints
Clear	delete breakpoints at next instruction
clear [*file*:]*fun*	delete breakpoints at entry to *fun()*
clear [*file*:]*line*	delete breakpoints on source line
delete [*n*]	delete breakpoints [or breakpoint *n*]
disable [*n*]	disable breakpoints [or breakpoint *n*]
enable [*n*]	enable breakpoints [or breakpoint *n*]
enable once [*n*]	enable breakpoints [or breakpoint *n*]; disable again when reached
enable del [*n*]	enable breakpoints [or breakpoint *n*]; delete when reached
ignore *n count*	ignore breakpoint *n*, *count* times
commands *n* [silent] *command-list*	execute GDB *command-list* every time breakpoint n is reached. [silent suppresses default display]
End	end of command-list

Program Stack

COMMAND SYNTAX	DESCRIPTION
backtrace [*n*]	print trace of all frames in stack; or of *n*
bt [*n*]	frames | innermost if *n*>0, outermost if *n*<0
frame [*n*]	select frame number *n* or frame at address *n*; if no *n*, display current frame
up *n*	select frame *n* frames up
down *n*	select frame *n* frames down
info frame [*addr*]	describe selected frame, or frame at *addr*
info args	arguments of selected frame
info locals	local variables of selected frame
info reg [*rn*]…	register values [for regs *rn*] in selected
info all-reg [*rn*]	frame; all-reg includes floating point

Execution Control

COMMAND SYNTAX	DESCRIPTION
continue [*count*] c [*count*]	continue running; if count specified, ignore this breakpoint next count times
step [*count*] s [*count*]	execute until another line reached; repeat count times if specified
stepi [*count*] si [*count*]	step by machine instructions rather than source lines
next [*count*] n [*count*]	execute next line, including any function calls
nexti [*count*] ni [*count*]	next machine instruction rather than source line
until [*location*]	run until next instruction (or location)

COMMAND SYNTAX	DESCRIPTION
finish	run until selected stack frame returns
return [*expr*]	pop selected stack frame without executing [setting return value]
signal *num*	resume execution with signal s (none if 0)
jump *line* jump **address*	resume execution at specified line number or address
set var=*expr*	evaluate expr without displaying it; use for altering program variables

Display

COMMAND SYNTAX	DESCRIPTION
print [/*f*] [*expr*] p [/*f*] [*expr*]	show value of *expr* [or last value $] according to format *f*.
x	Hexadecimal
d	signed decimal
u	unsigned decimal
o	Octal
t	Binary
a	address, absolute and relative
c	Character
f	floating point
call [/*f*] *expr*	like print but does not display void
x [/*Nuf*] *expr*	examine memory at address *expr*; optional format spec follows slash
N	count of how many units to display
u	unit size; one of b individual bytes

COMMAND SYNTAX	DESCRIPTION
	h halfwords (two bytes) w words (four bytes) g giant words (eight bytes)
f	printing format. Any print format, or s null-terminated string i machine instructions
disassem [*addr*]	display memory as machine instructions

Automatic Display

COMMAND SYNTAX	DESCRIPTION
display [/f] expr	show value of expr each time program stops [according to format *f*]
display	display all enabled expressions on list
undisplay n	remove number(s) n from list of automatically displayed expressions
disable disp n	disable display for expression(s) number n
enable disp n	enable display for expression(s) number n
info display	numbered list of display expressions

Expressions

COMMAND SYNTAX	DESCRIPTION
expr	an expression in C, C++, or Modula-2 (including function calls)
addr@len	an array of *len* elements beginning at *addr*
file::nm	a variable or function *nm* defined in *file*
{type}addr	read memory at *addr* as specified *type*
$	most recent displayed value
$*n*	*n*th displayed value

COMMAND SYNTAX	DESCRIPTION
$$	displayed value previous to $
$$*n*	*n*th displayed value back from $
$_	last address examined with x
$__	value at address $_
$var	convenience variable; assign any value
show values [n]	show last 10 values [or surrounding $n]
show conv	display all convenience variables

Symbol Table

COMMAND SYNTAX	DESCRIPTION
info address *s*	show where symbol *s* is stored
info func [*regex*]	show names, types of defined functions (all, or matching *regex*)
info var [*regex*]	show names, types of global variables (all, or matching *regex*)
whatis [*expr*] ptype [*expr*]	show data type of *expr* [or $] without evaluating; ptype gives more detail
ptype *type*	describe *type*, struct, union, or enum

GDB Scripts

COMMAND SYNTAX	DESCRIPTION
source *script*	read, execute GDB commands from file *script*
define *cmd* *command-list*	create new GDB command *cmd*; execute script defined by *command-list*
end	end of command-list

COMMAND SYNTAX	DESCRIPTION
document *cmd* *help-text*	create online documentation for new GDB command cmd
end	end of *help-text*

Signals

COMMAND SYNTAX	DESCRIPTION
handle signal act	specify GDB actions for signal:
print	announce signal
noprint	be silent for signal
stop	halt execution on signal
nostop	do not halt execution
pass	allow your program to handle signal
nopass	do not allow your program to see signal
info signals	show table of signals, GDB action for each

Debugging Targets

COMMAND SYNTAX	DESCRIPTION
target *type* param	connect to target machine, process, or file
help target	display available targets
attach param	connect to another process
detach	release target from GDB control

Controlling GDB

COMMAND SYNTAX	DESCRIPTION
set param value	set one of GDB's internal parameters
show param	display current setting of parameter

Parameters understood by set and show

COMMAND SYNTAX	DESCRIPTION
complaint *limit*	number of messages on unusual symbols
confirm *on/off*	enable or disable cautionary queries
editing *on/off*	control readline command-line editing
height *lpp*	number of lines before pause in display
language *lang*	Language for GDB expressions (auto, c or modula-2)
listsize *n*	number of lines shown by list
prompt *str*	use *str* as GDB prompt
radix *base*	octal, decimal, or hex number representation
verbose *on/off*	control messages when loading symbols
width *cpl*	number of characters before line folded
write *on/off*	Allow or forbid patching binary, core files (when reopened with exec or core)
history… h… h exp *off/on* h file *filename* h size *size* h save *off/on*	groups with the following options: disable/enable readline history expansion file for recording GDB command history number of commands kept in history list control use of external file for command history

COMMAND SYNTAX	DESCRIPTION
print… p… p address *on/off* p array *off/on* p demangl *on/off* p asm-dem *on/off* p elements *limit* p object *on/off* p pretty *off/on* p union *on/off* p vtbl *off/on*	groups with the following options: print memory addresses in stacks, values compact or attractive format for arrays source (demangled) or internal form for C++ symbols demangle C++ symbols in machine-instruction output number of array elements to display print C++ derived types for objects struct display: compact or indented display of union members display of C++ virtual function tables
show commands	show last 10 commands
show commands *n*	show 10 commands around number *n*
show commands +	show next 10 commands

Working Files

COMMAND SYNTAX	DESCRIPTION
file [*file*]	use file for both symbols and executable; with no arg, discard both
core [*file*]	read file as coredump; or discard
exec [*file*]	use file as executable only; or discard
symbol [*file*]	use symbol table from file; or discard
load *file*	dynamically link file and add its symbols
add-sym *file addr*	read additional symbols from *file*, dynamically loaded at *addr*
info files	display working files and targets in use
path *dirs*	add *dirs* to front of path searched for executable and symbol files
show path	display executable and symbol file path

COMMAND SYNTAX	DESCRIPTION
info share	list names of shared libraries currently loaded

Source Files

COMMAND SYNTAX	DESCRIPTION
dir *names*	add directory names to front of source path
dir	clear source path
show dir	show current source path
list	show next ten lines of source
list -	show previous ten lines
list lines [*file*:]*num* [*file*:]*function* +*off* -*off* **address*	display source surrounding lines, specified as: line number [in named file] beginning of function [in named file] *off* lines after last printed *off* lines previous to last printed line containing *address*
list *f,l*	from line *f* to line *l*
info line *num*	show starting, ending addresses of compiled code for source line *num*
info source	show name of current source file
info sources	list all source files in use
forw *regex*	search following source lines for *regex*
rev *regex*	search preceding source lines for *regex*

2.10.6 DWARF Specifications

This section describes the debugging information generated by KPIT GNU toolchains which is required for symbolic, source level debugging. This debugging information is output by GNU SH toolchain. The debugging information which is encoded in execututable is output using ¯readelf‖ utility.

The debugging information contains location description, line number information, macro information and call frame information. This is described in subsequent sections.

2.10.6.1 .debug_info Section

The .debug_info for each compilation unit consists of a compilation unit header and debugging information entries.

a. Compilation Unit Header
A compilation unit header consists of the four datas.

4-byte	2-byte	4-byte	1-byte
Length of .debug_info for compilation unit	Version of dwarf format	Offset into .debug_abbrev	Size of address

- Length of .debug_info for compilation unit

4-byte unsigned integer. The length of .debug_info section for a compilation unit, not including length field.

- Version of dwarf format

2-byte unsigned integer. The value is 2.

- Offset into .debug_abbrev

4 -byte unsigned integer. The offset into .debug_abbrev associated with the compile unit

- Size of address

1-byte unsigned integer. The byte size of address for target architecture. In case of SH : 4

b. Debugging Information Entry
Each debugging information entry consists of the entry code within abbreviation table and a series of attribute values.

ULEB128	data size*		data size*
Entry code within abbreviation table	Attribute value	…	Attribute value

- Entry code within abbreviation table

  Unsigned LEB128 (Little Endian Base) number.

  The entry code within abbreviation table which has the attribute name and attribute form for this entry. Debugging information entries consisting of only the 0 abbreviation code are considered null entries

- Attribute value

  The data size determined by the attribute form of abbreviation table. The appropriate value for the attribute name and attribute form in the abbreviation table.

2.10.6.2 .debug_abbrev Section

The .debug _abbrev contains abbreviation table for all compilation units. Multiple compilation units can share the same abbreviation table. The abbreviations for a given compilation unit end with an entry consisting of a 0 byte for the abbreviation code.

a. Abbreviation Table Entry
   Each Entry begins with the abbreviation entry code. The entry has the tag encoding, child entry, and a series of attribute specifications

ULEB128	ULEB128	1-byte	ULEB128		ULEB128
Entry code within abbreviation table	Tag entry	Child determination	Attribute specifications	…	Attribute specifications

- Entry code within abbreviation table

  Unsigned LEB128 number.
  The entry code within abbreviation table which has the attribute name and attribute form for this entry.

- Tag entry

  Unsigned LEB128 number.
  The value determines the classification of the debugging information entry.

- Child determination

  1-byte unsigned integer.
  The value that determines whether a debugging information entry using this abbreviation has children entry or not.

- Child determination encodings

Child determination name	Value

DW_CHILDREN_no	0x0
DW_CHILDREN_yes	0x1

- Attribute specifications

  Unsigned LEB128 number.
  Each attribute specification consists of the attribute name and attribute form. The series of attribute specifications ends with an entry containing 0 for the name and 0 for the form.

ULEB128	ULEB128
Attribute name	Attribute form

- Attribute name

  Unsigned LEB128 number.
  The attribute name defines the specific characteristics of the entry.

- Attribute form

  Unsigned LEB128 number.
  See the following table.
  Attribute Form Value Table

Attribute Form	Value	Data Size	Data Meaning
DW_FORM_addr	0x1	*1	An address on target machine which is relocatable in relocatable file.
DW_FORM_block1	0xa	1+*	1-byte length which is the number of information bytes that follow.
DW_FORM_block2	0x3	2+*	2-byte length which is the number of information bytes that follow.
DW_FORM_block4	0x4	4+*	4-byte length which is the number of information bytes that follow.
DW_FORM_block	0x9	ULEB+*	Variable length using unsigned LEB128 number which is the number of information bytes that follow
DW_FORM_data1	0xb	1	1-byte constant data
DW_FORM_data2	0x5	2	2-byte constant data
DW_FORM_data4	0x6	4	4-byte constant data
DW_FORM_data8	0x7	8	8-byte constant data
DW_FORM_sdata	0xd	LEB	Variable length constant data using signed LEB128 numbers
DW_FORM_udata	0xf	ULEB	Variable length constant data using

			unsigned LEB128 numbers
DW_FORM_flag	0xc	1	1-byte constant data
DW_FORM_ref1	0x11	1	1-byte Offset from beginning of the compilation unit header
DW_FORM_ref2	0x12	2	2-byte Offset from beginning of the compilation unit header
DW_FORM_ref4	0x13	4	4-byte Offset from beginning of the compilation unit header
DW_FORM_ref8	0x14	8	8-byte Offset from beginning of the compilation unit header
DW_FORM_ref_udata	0x15	ULEB	Variable length offset using unsigned LEB128 number from beginning of the compilation unit header
DW_FORM_ref_addr	0x10	4	Offset from beginning of file
DW_FORM_string	0x8	variable	A block of contiguous non-null bytes followed by null byte
DW_FORM_strp	0xe	4	Offset from beginning of the .debug_str section
DW_FORM_indirect	0x16	ULEB	Unsigned LEB128 number represents the form

- Relation between .debug_info and Abbreviation Table

  The appropriate entry of the abbreviation table (.debug_abbrev) provides the attribute name and attribute form for the debugging information entry.

  ex) Compilation Unit 1: .debug_info

Abbreviation Table: .debug_abbrev

```
Compilation Unit @ 0:
Length:           102
Version:         2
Abbrev Offset:  0
Pointer Size:   2

1 (abbreviation entry code)
<0><b>: Abbrev Number: 1
(DW_TAG_compile_unit)
     DW_AT_stmt_list    : 0
 o
     DW_AT_high_pc   : 0x117c
 o
     DW_AT_low_pc    : 0x1160
 o
     DW_AT_producer  : GNU C++
4.0-GNUSH v0601
     DW_AT_language    : 4
      (C++)
     DW_AT_name      : test1.c


     DW_AT_comp_dir  :
 C:\GNU\DWARF\Examples\A1p1
```

```
1 (abbreviation entry code)
1      DW_TAG_compile_unit
[has children]
    DW_AT_stmt_list
DW FORM data4
    DW AT high pc
DW_FORM_addr
    DW AT low pc
DW_FORM_addr
    DW_AT_producer
DW FORM string
    DW_AT_language
DW_FORM_data1
    DW_AT_name
DW FORM string
    DW_AT_comp_dir
DW_FORM_string
```

2.10.6.3 Program Scope Entries

Compilation Unit Entries

a. DW_TAG_compile_unit

Tag name	Attribute name	Classes	MEANING
DW_TAG_compile_unit	DW_AT_comp_dir	DW_FORM_string	Current working directory
	DW_AT_high_pc *1	DW_FORM_addr	Address of last machine instruction generated for that compilation unit + 1
	DW_AT_language	DW_FORM_data2	A code indicating the source language of the compilation unit. The set of language names and their meanings are given in the below table "Language encodings".
	DW_AT_low_pc *1	DW_FORM_addr	Address of first machine instruction generated for that compilation unit
	DW_AT_name	DW_FORM_string	source file name containing relative path name from which the compilation unit was derived or full path name (specified on command line)
	DW_AT_producer *2	DW_FORM_string	a null-terminated string containing tool produced the compilation unit.
	DW_AT_stmt_list	DW_FORM_data 4	Offset in .debug_line of the line information for this compilation unit

*1: Compiler does not output DW_AT_high_pc and DW_AT_low_pc when text section is divided using __attribute__ section.

*2: The value of DW_AT_producer is the

following Language encodings

Name	Value	case
DW_LANG_C89	0x1	ISO/ANSI C
DW_LANG_C	0x2	C
DW_LANG_C_plus_plus	0x4	C++ language

Subroutine and Entry Point Entries

a. DW_TAG_subprogram

Tag name	Attribute name	Classes	MEANING
DW_TAG_ subprogram	DW_AT_decl_file	DW_FORM_udata	File index corresponding to a file number in line information
	DW_AT_decl_line	DW_FORM_udata	Source line number at which the first character of the identifier of the declared object appears.
	DW_AT_external	DW_FORM_flag	Visibility outside of its containing compilation unit
	DW_AT_frame_base	DW_FORM_block1	Location description of frame base
	DW_AT_high_pc	DW_FORM_addr	Address of last machine instruction generated for that subprogram + 1
	DW_AT_low_pc	DW_FORM_addr	Address of first machine instruction generated for that subprogram
	DW_AT_name	DW_FORM_string	subprogram name
	DW_AT_specification	DW_FORM_ref_addr	Reference to debugging information entry representing the declaration of function member
	DW_AT_virtuality	DW_FORM_data1	Virtuality of declaration. The sets of virtuality are given in the below table "Virtuality encodings".
	DW_AT_declaration	DW_FORM_flag	The non-defining declaration
	DW_AT_ vtable_elem_location	DW_FORM_block1	Location description of virtual table
	DW_AT_prototyped	DW_FORM_flag	Declaration using function prototype
	DW_AT_return_addr	DW_FORM_block1	Location description of return address
	DW_AT_ containing_type	DW_FORM_ref_addr	Reference to debugging information entry

			representing class including member of this entry may point
	DW_AT_inline	DW_FORM_data1	Availability of inline expansion. The sets of availability are given in the below table "Inline encordings".
	DW_AT_sibling	DW_FORM_ref4	Debugging information entry relationship
	DW_AT_MIPS_linkage_name	DW_FORM_string	
	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing type returned by subprogram

Inline encordings

Name	Value
DW_INL_not_inlined	0x0
DW_INL_inlined	0x1
DW_INL_declared_not_inlined	0x2
DW_INL_declared_inlined	0x3

Virtuality encordings

Name	Value
DW_VIRTUALITY_none	0x0
DW_VIRTUALITY_virtual	0x1
DW_VIRTUALITY_pure_virtual	0x2

b. DW_TAG_inlined_subroutine

Tag name	Attribute name	Classes	MEANING
DW_TAG_ inlined_subroutine	DW_AT_ abstract_origin	DW_FORM_ref_addr	Reference to associated abstract instance entry(DW_TAG_inlined_subroutine)
	DW_AT_decl_file	DW_FORM_udata	File index corresponding to a file number in line information
	DW_AT_decl_line	DW_FORM_udata	Source line number at which the first character of the identifier of the declared object appears.
	DW_AT_high_pc	DW_FORM_addr	Address of last machine instruction generated for that expanded function + 1
	DW_AT_low_pc	DW_FORM_addr	Address of first machine instruction generated for that expanded function
	DW_AT_call_file	DW_FORM_data1	File containing inlined subroutine call. Generated for M16CM32C toolchain only.
	DW_AT_call_line	DW_FORM_data1	Line number of inlined subroutine call. Generated for M16CM32C toolchain only.

Lexical Block Entries

a. DW_TAG_lexical_block

Tag name	Attribute name	Classes	MEANING
DW_TAG_lexical_block	DW_AT_high_pc	DW_FORM_addr	Address of last machine instruction generated for the lexical block + 1
	DW_AT_sibling	DW_FORM_ref4	Debugging information entry relationship
	DW_AT_low_pc	DW_FORM_addr	Address of first machine instruction generated for the lexical block

Label Entries

a. DW_TAG_label

Tag name	Attribute name	Classes	MEANING
DW_TAG_label	DW_AT_external	DW_FORM_ flag	Visibility outside of its containing compilation unit
	DW_AT_decl_file	DW_FORM_udata	File index corresponding to a file number in line information
	DW_AT_decl_line	DW_FORM_udata	Source line number at which the first character of the identifier of the declared object appears.
	DW_AT_low_pc	DW_FORM_addr	Address of first machine instruction generated for the statement identified by label
	DW_AT_location	DW_FORM_block1	Location expression (Only in case of asm)
	DW_AT_name	DW_FORM_string	label name

Try and Catch Block Entries

a. DW_TAG_thrown_type (Not generated in KPIT GNU tools)

Tag name	Attribute name	Classes	MEANING
DW_TAG_thrown_type	DW_AT_ abstract_origin	DW_FORM_ref_addr	Reference to associated abstract instance entry (DW_TAG_thrown_type)
	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing type

b. DW_TAG_try_block (Not generated in KPIT GNU tools)

Tag name	Attribute name	Classes	MEANING
DW_TAG_try_block	DW_AT_ abstract_origin	DW_FORM_ref_addr	Reference to associated abstract instance entry(DW_TAG_try_block)
	DW_AT_high_pc	DW_FORM_addr	Address of last machine instruction generated for the try block + 1
	DW_AT_low_pc	DW_FORM_addr	Address of first machine instruction generated for the try block

c. DW_TAG_catch_block (Not generated in KPIT GNU tools)

Tag name	Attribute name	Classes	MEANING
DW_TAG_catch_block	DW_AT_ abstract_origin	DW_FORM_ref_addr	Reference to associated abstract instance entry(DW_TAG_catch_block)
	DW_AT_high_pc	DW_FORM_addr	Address of last machine instruction generated for the catch block + 1
	DW_AT_low_pc	DW_FORM_addr	Address of first machine instruction generated for the catch block

Data Object Entries

a. DW_TAG_constant (Not generated in KPIT GNU tools)

Tag name	Attribute name	Classes	MEANING
DW_TAG_constant	DW_AT_	DW_FORM_string	constant string
	const_value	DW_FORM_data	Const value(1,2,4,8)
		DW_FORM_block	Other than above

b. DW_TAG_variable

Tag name	Attribute name	Classes	MEANING
DW_TAG_variable	DW_AT_declaration	DW_FORM_ flag	the non-defining declaration
	DW_AT_decl_file	DW_FORM_udata	File index corresponding to a file number in line information
	DW_AT_decl_line	DW_FORM_udata	Source line number at which the first character of the identifier of the declared object appears.
	DW_AT_external	DW_FORM_ flag	Visibility outside of its containing compilation unit
	DW_AT_location	DW_FORM_block1	Location expression of the variable
		DW_FORM_data4	Offset in .debug_loc of location lists associated to the variable
	DW_AT_name	DW_FORM_string	variable name
	DW_AT_MIPS_linkage_name	DW_FORM_string	
	DW_AT_specification	DW_FORM_ref_addr	Reference to debugging information entry representing the declaration of function member
	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing type
	DW_AT_abstract_origin	DW_FORM_ref_addr	Reference to associated abstract instance entry(DW_TAG_variable)

c. DW_TAG_formal_parameter

Tag name	Attribute name	Classes	MEANING
DW_TAG_ formal_parameter	DW_AT_decl_file	DW_FORM_udata	File index corresponding to a file number in line information
	DW_AT_decl_line	DW_FORM_udata	Source line number at which the first character of the identifier of the declared object appears.
	DW_AT_location	DW_FORM_block1	Location expression of parameter
		DW_FORM_data4	Offset in .debug_loc of location lists associated to parameter
	DW_AT_artificial	DW_FORM_flag	Objects or types that are not actually declared in the source
	DW_AT_name	DW_FORM_string	Parameter name
	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing type

d. DW_TAG_unspecified_parameter

Tag name	Attribute name	Classes	MEANING
DW_TAG_ unspecified_ parameters			

e. DW_TAG_template_type_parameter (Not generated in KPIT GNU tools)

Tag name	Attribute name	Classes	MEANING
DW_TAG_template_ type_parameter	DW_AT_ abstract_origin	DW_FORM_ref_addr	Reference to associated abstract instance entry(DW_TAG_type_parameter)
	DW_AT_name	DW_FORM_string	parameter name
	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing actual type

f. DW_TAG_template_value_parameter (Not generated in KPIT GNU tools)

Tag name	Attribute name	Classes	MEANING
DW_TAG_template_Value_parameter	DW_AT_abstract_origin	DW_FORM_ref_addr	Reference to associated abstract instance entry(DW_TAG_template_value_param)
	DW_AT_const_value	DW_FORM_data	actual constant value
	DW_AT_name	DW_FORM_string	parameter name
	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing actual type

Base Type Entries

a. DW_TAG_base_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_base_type	DW_AT_byte_size	DW_FORM_data1	Size in bytes of the storage unit used to represent an object of the given type
	DW_AT_encoding	DW_FORM_data1	How to be encoded and be interpreted. The sets of encoding are given the below table "Base type encoding values".
	DW_AT_name	DW_FORM_string	type name

Base type encoding values

Name	Value	case
DW_ATE_void	0x0	Void
DW_ATE_address	0x1	Address
DW_ATE_boolean	0x2	bool
DW_ATE_complex_float	0x3	Complex float
DW_ATE_float	0x4	float, double, long double
DW_ATE_signed	0x5	[signed] short, [signed] int, [signed] long, [signed]long long
DW_ATE_signed_char	0x6	char, signed char
DW_ATE_unsigned	0x7	unsigned short, unsigned int, unsigned long, unsigned long long
DW_ATE_unsigned_char	0x8	unsigned char

Type Modifier Entries

a. DW_TAG_const_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_ const_type	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing type.

b. DW_TAG_pointer_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_ pointer_type	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing type
	DW_AT_byte_size	DW_FORM_data1	Size in bytes of the storage unit used to represent an object of the given type

c. DW_TAG_reference_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_ reference_type	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing type
	DW_AT_byte_size	DW_FORM_data1	Size in bytes of the storage unit used to represent an object of the given type

d. DW_TAG_volatile_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_ volatile_type	DW_AT_type	DW_FORM_ref4	Reference to debugging information entry representing type

Typedef Entries

a. DW_TAG_typedef

Tag name	Attribute name	Classes	MEANING
DW_TAG_ typedef	DW_AT_decl_file	DW_FORM_udata	File index corresponding to a file number in line information
	DW_AT_decl_line	DW_FORM_udata	Source line number at which the first character of the identifier of the declared object appears.
	DW_AT_name	DW_FORM_string	typedef name
	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing type

Array Type Entries

a. DW_TAG_array_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_ array_type	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing type
	DW_AT_sibling	DW_FORM_ref4	Debugging information entry relationship

Structure Union and Class Type

Entries a. DW_TAG_structure_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_ structure_type	DW_AT_byte_size	DW_FORM_udata	Size in bytes required to hold an instance and any padding
	DW_AT_decl_file	DW_FORM_udata	File index corresponding to a file number in line information
	DW_AT_decl_line	DW_FORM_udata	Source line number at which the first character of the identifier of the declared object appears.
	DW_AT_sibling	DW_FORM_ref4	Debugging information entry relationship
	DW_AT_ containing_type	DW_FORM_ref_addr	Reference to debugging information entry representing class including member of this entry may point
	DW_AT_declaration	DW_FORM_ flag	the non-defining declaration
	DW_AT_specification	DW_FORM_ref_addr	Reference to debugging information entry representing the declaration of function member
	DW_AT_name	DW_FORM_string	tag name

b. DW_TAG_union_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_ union_type	DW_AT_byte_size	DW_FORM_udata	Size in bytes required to hold an instance and any padding
	DW_AT_decl_file	DW_FORM_udata	File index corresponding to a file number in line information
	DW_AT_decl_line	DW_FORM_udata	Source line number at which the first character of the identifier of the declared object appears.
	DW_AT_name	DW_FORM_string	tag name
	DW_AT_sibling	DW_FORM_ref4	Debugging information entry relationship

c. DW_TAG_class_type (Not generated in KPIT GNU tools)

Tag name	Attribute name	Classes	MEANING
DW_TAG_ class_type	DW_AT_ abstract_origin	DW_FORM_ref_addr	Reference to associated abstract instance entry(DW_TAG_class_type)
	DW_AT_accessibility	DW_FORM_data1	Accessibility of declaration. The sets are given in 1.3.1(2) "Accessibility encordings".
	DW_AT_byte_size	DW_FORM_udata	Size in bytes required to hold an instance and any padding
	DW_AT_declaration	DW_FORM_ flag	the non-defining declaration
	DW_AT_decl_file	DW_FORM_udata	File index corresponding to a file number in line information
	DW_AT_decl_line	DW_FORM_udata	Source line number at which the first character of the identifier of the declared object appears.
	DW_AT_name	DW_FORM_string	tag name

d. DW_TAG_inheritance

Tag name	Attribute name	Classes	MEANING
DW_TAG_ inheritance	DW_AT_accessibility	DW_FORM_data1	Accessibility of declaration. The sets are given in 1.3.1(2) "Accessibility encordings".
	DW_AT_ data_member_location	DW_FORM_block1	Data member location: Offset from beginning of the entire class
	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing type
	DW_AT_virtuality	DW_FORM_data1	Virtuality of declaration. The sets are given in 1.3.1(2) "Virtuality encordings".

Accessibility encordings

Name	Value
DW_ACCESS_public	0x1
DW_ACCESS_protected	0x2
DW_ACCESS_private	0x3

e. DW_TAG_access_declaration (Not generated in KPIT GNU tools)

Tag name	Attribute name	Classes	MEANING
DW_TAG_ access_declaration	DW_AT_accessibility	DW_FORM_data1	Accessibility of declaration. The sets are given in 1.3.1(2) "Accessibility encordings".
	DW_AT_name	DW_FORM_string	object name

f. DW_TAG_friend (Not generated in KPIT GNU tools)

Tag name	Attribute name	Classes	MEANING
DW_TAG_ friend	DW_AT_friend	DW_FORM_ref_addr	Reference to debugging information entry describing friend entry

g. DW_TAG_member

Tag name	Attribute name	Classes	MEANING
DW_TAG_ member	DW_AT_accessibility	DW_FORM_data1	Accessibility of declaration. The sets are given in 1.3.1(2) "Accessibility encordings".
	DW_AT_artificial	DW_FORM_flag	Objects or types that are not actually declared in the source
	DW_AT_byte_size	DW_FORM_udata	Size in bytes required to hold an instance and any padding
	DW_AT_bit_offset	DW_FORM_data1	Offset in bits of the high order bit of a value of the given type
	DW_AT_bit_size	DW_FORM_data1	Actual size in bits used to represent a values of the given type
	DW_AT_ data_member_location	DW_FORM_block1	Data member location: Offset from beginning of the class
	DW_AT_decl_file	DW_FORM_udata	File index corresponding to a file number in line information
	DW_AT_decl_line	DW_FORM_udata	Source line number at which the first character of the identifier of the declared object appears.
	DW_AT_name	DW_FORM_string	member name
	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing

			type

Enumeration Type Entries

a. DW_TAG_enumeration_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_ enumeration_type	DW_AT_byte_size	DW_FORM_data1	SuperH : 4 H8 :1 or 2 M16C, M32C : 2
	DW_AT_decl_file	DW_FORM_udata	File index corresponding to a file number in line information
	DW_AT_sibling	DW_FORM_ref4	Debugging information entry relationship
	DW_AT_decl_line	DW_FORM_udata	Source line number at which the first character of the identifier of the declared object appears.
	DW_AT_name	DW_FORM_string	enum name

b. DW_TAG_enumerator

Tag name	Attribute name	Classes	MEANING
DW_TAG_ enumerator	DW_AT_const_value	DW_FORM_sdata	enum member value
	DW_AT_name	DW_FORM_string	enum member name

Subroutine Type Entries

a. DW_TAG_subroutine_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_	DW_AT_prototyped	DW_FORM_flag	Declaration using function prototype
subroutine_type	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing return type
	DW_AT_sibling	DW_FORM_ref4	Debugging information entry relationship

Subrange Type Entries

a. DW_TAG_subrange_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_ subrange_type	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing type
	DW_AT_upper_bound	DW_FORM_data1	Value indicating upper bound of the subrange

Pointer to Member Type Entries

a. DW_TAG_ptr_to_member_type

Tag name	Attribute name	Classes	MEANING
DW_TAG_ ptr_to_member_ Type	DW_AT_ containing_type	DW_FORM_ref_addr	Reference to debugging information entry representing class including member of this entry may point
	DW_AT_type	DW_FORM_ref_addr	Reference to debugging information entry representing class member type

2.10.6.4 Other Debug Information

a. Line Number Information

The line number information generated for a compilation unit is represented in the .debug_line section of an object file and is referenced by a corresponding unit debugging information entry in the .debug_info section.

The initial value of state machine registers;

address	0
file	1
line	1
column	0
is_stmt	determined by default_is_stmt in the statement

program prologue basic_block

"false" end_sequence

"false"

- .debug_line section

The line number information for a compilation unit consists of statement program prologue and statement program.

Statement Program Prologue

Uword	Uhalf	uword	ubyte
total_length	DWARF version	prologue_length	minimum_instruction_lengths

ubyte	Sbyte	ubyte	Ubyte	array of ubyte	string	string
initial_is_stmt	line_Base	line_range	opcode_base	standard_opcode_lengths	include_directories	file_names

total_length: The size in bytes of the statement information for this compilation unit (not including this field).

DWARF version: Version of statement information format. The value is 2.

prologue_length: The number of bytes following this field to beginning of the first byte of the statement program itself.

minimum_instruction_lengths: The number of bytes of the smallest target machine instruction. This value is 2.

initial_is_stmt: The initial value of the is_stmt register.

line_base: This parameter affects the meaning of the special opcodes. KPIT tools sets this value to -5.

line_range: This parameter affects the meaning of the special opcodes. KPIT tools sets this value to 14.

opcode_base: The number assigned to the first special opcode. This value is 0xa.

standard_opcode_lengths: This array of specifies the number of operands for each of standard opcodes.

standard_opcode_lengths

Index	Value	Data Meaning
[0]	0	The number of operands for DW_LNS_copy
[1]	1	The number of operands for DW_LNS_advance_pc
[2]	1	The number of operands for DW_LNS_advance_line
[3]	1	The number of operands for DW_LNS_set_file
[4]	1	The number of operands for DW_LNS_set_column
[5]	0	The number of operands for DW_LNS_negate_stme
[6]	0	The number of operands for

		DW_LNS_set_basic_block
[7]	0	The number of operands for DW_LNS_const_add_pc
[8]	1	The number of operands for DW_LNS_fixed_advance_pc

include_directories: the full path names for included source file in this compilation. (the path specified explicitly by the user and the compiler searches without explicit direction). The current directory of the compilation is understood to be the first entry and is not explicitly represented. The last entry is followed by a single null byte.

file_names: The each source file name that contributed to the statement information for this compilation. The last entry is followed by a single null byte.

String	ULEB128	ULEB128	ULEB128
file name	directory index	time	length

- file name

  Null terminated string containing the file name.

- directory index

  The number representing the directory index. The value 0 means the file was found in the current directory of the compilation.

- time

  The time of last modification for the file.

- length

  The length in bytes of the file.

Statement Program

Statement program is to build a matrix representing one compiling unit. Within a sequence, address may only increase.

Every statement program sequence must end with a DW_LNE_end sequence.

1-bye	
Opcode	…

Opcode	Meaning
0x0	extended opcodes

0x1 - 0x9	Standard opcodes
0xa - 0xff	special opecodes

extended opecodes

1-bye	ULEB128	1-byte	
Opcode(0)	Length	sub_opcode	…

- Length

  The number of bytes in the instruction itself (not including this field).

- sub_opcode

sub_opcode	Value	Data Meaning
DW_LNE_end_sequence	1	Sets the end_sequence register of the state machine to "true". Appends a row to the matrix using the current state machine register. Resets the registers to the initial value.
DW_LNE_set_address	2	Takes a single relocatable address as an operand which size is address size. Set the address register to the value given by the relocatable address.
DW_LNE_define_file	3	Takes 4 arguments. Add to file_names lists.

Standard opcodes

standard opcode	Value	No of op	operand	Data Meaning
DW_LNS_copy	1	0	-	Appends a row to the matrix using the current statement register. Sets register to "FALSE".
DW_LNS_advance_pc	2	1	ULEB128 Δaddress	address_register += operand * minimum_instruction_length
DW_LNS_advance_line	3	1	SLEB128 Δline	line_register += operand
DW_LNS_set_file	4	1	ULEB128 File index	Stores operand in file register.
DW_LNS_set_column	5	1	ULEB128 column No	Stores operands in column register
DW_LNS_negate_stmt	6	0	-	is_stmt register = !is_stmt
DW_LNS_set_basic_block	7	0	-	Sets basic block register to "TRUE"
DW_LNS_const_add_pc	8	0	-	Adds to the address register of the address increment value corresponding to special opcode 255.
DW_LNS_fixed_advance_pc	9	1	uhalf data Δaddress	Adds to the address register of the state machine the value of the (unencoded) operand.

Special opecodes

1-byte code. The value is encoded by the following.

opcode = (line_increment - line_base) + (line_range * address_advance)
+ opcode_base

*address_advance: address_increment / minimum_instruction_length

Macro Information

DWARF macro information gets generated only when -g3 option is specified at the time of compilation.

Macro Table Entry

Field name		Size	Meaning
Macro type		1	Macro type number. See below.
Line Number		ULEB128	Line number pre-processor directive occurred.
	File index	ULEB128	File index corresponding to a file number in line information.
	Defined string	string	A null terminated character string representing macro symbol and definition string for macro

Macro type Number

Macroinfo type	value	No of operands	1st operand	2nd operand
Unit end	0	0	-	-
DW_MACINFO _define	1	2	Line number	<Macro symbol>*△ <definition string>
DW_MACINFO _undef	2	2	Line number	<Macro symbol>
DW_MACINFO _start_file	3	2	Line number	File index
DW_MACINFO _end_file	4	0	-	-
DW_MACINFO _vendor_ext	255	Not supported	Not supported	Not supported

*<Macro symbol>:
In case of function-like macro is macro symbol followed immediately the macro formal parameter list including the surrounding parentheses (not include space)

Call Frame Information

a. Common Information Entry (CIE)

uword	uword	ubyte	string	ULEB128
length	CIE_id	version	augmentation	code_alignment_factor
SLEB128	ubyte			
data_alignment_factor	return_address_register	initial_instructions	...	padding

Length: The size in bytes of the CIE structure. (not including this field).

length % <address unit size> == 0

CIE_id: This is used to distinguish CIEs from FDEs. This value is 0xffffffff.

Version: Version of call frame information format. The value is 1.

Augmentation: A null terminated string that identifiers the augmentation to this CIE or to the FDEs.

code_alignment_factor: An unsigned LEB128 data.

data_alignment_factor: An signed LEB128 data.

return_address_register: A register number (not correspond to an actual machine register) represents the return address of the function.

initial_instructions: A sequence of rules that are interpreted to create the initial setting of each column in the table.

Padding: DW_CFA_nop instructions.

b. Frame Description Entry (FDE)

uword	uword	addr_size	addr_size	
length	CIE_pointer	initial_location	address_range	instructions

Length: The size in bytes of the FDE entry for this function. (not including this field).

length % <address unit size> == 0

CIE_pointer: Offset into the .debug_frame section that denotes the CIE that is associated with this FDE.

initial_location: The address of the first location associated with this FDE entry.

address_range: The number of bytes of program instructions described by this entry.

Instructions: A sequence of table defining instructions that are described below.

Instructions

Instruction	high 2bit	low 6bit	op1	op2	Meaning
DW_CFA_advance_loc	0x1	delta			PC += (delta * code_alignment_factor)
DW_CFA_offset	0x2	register	SLEB128 offset		Mov register to CFA + N (N: offset * data_alignment_factor)
DW_CFA_restore	0x3	register			Change register rule into initial_instruction in CIE
DW_CFA_set_loc	0x0	0x1	address		PC = address
DW_CFA_advance_loc1	0x0	0x2	ubyte delta		PC += (delta * code_alignment_factor)
DW_CFA_advance_loc2	0x0	0x3	uhalf delta		PC += (delta * code_alignment_factor)
DW_CFA_advance_loc4	0x0	0x4	uword delta		PC += (delta * code_alignment_factor)
DW_CFA_offset_extended	0x0	0x5	ULEB128 register	SLEB128 offset	Mov register to CFA + N (N: offset * data_alignment_factor)
DW_CFA_restore_extended	0x0	0x6	ULEB128 register		Change register rule into initial_instruction in CIE
DW_CFA_undefined	0x0	0x7	ULEB128 register		Specified register is undefined
DW_CFA_same_value	0x0	0x8	ULEB128 register		Specified register is same value
DW_CFA_register	0x0	0x9	ULEB128 register	ULEB128 register	Mov op1 to op2
DW_CFA_remember_state	0x0	0xa			Save the rules for every register on the stack
DW_CFA_restore_state	0x0	0xb			Pop the rules off the stack
DW_CFA_def_cfa	0x0	0xc	ULEB128 register	SLEB128 offset	Set cfa to register + N (N: offset * data_alignment_factor)

DW_CFA_def_cfa_register	0x0	0xd	ULEB128 register		Set cfa to register + N (N: offset * data_alignment_factor) offset is previous data
DW_CFA_def_cfa_offset	0x0	0xe	SLEB128 offset		Set cfa to register + N (N: offset * data_alignment_factor) register is previous data
DW_OP_nop	0x0	0x0			Padding

c. Register definition of call frame information cpu=SH

Data Meaning	Value
R0 - R64	0 - 64
PR	65
SPC	66
FR0 - FR63	67 - 130

Lookup By Name

For lookup by name, a table is maintained in .debug_pubnames section.

Uword	uhalf	uword	uword	
Length	version	offset	size	entries

Length: The length of entries for the set (not including this field).

Version: Version of DWARF format. The value is 2.

Offset: The offset from beginning of the .debug_info of the compilation unit entry referenced by the set.

Size: The size in bytes of .debug_info generated to represent that compilation unit.

Entries: A variable number of offset / name. Each set is terminated by a 4-byte word containing 0.

Uword	string		uword
offset	name	...	0

- offset
  The offset from beginning of the compilation unit entry corresponding to this set
- name
  A null terminated character string representing the name.

In case of the name of a static data member or function member of a C++ structure, class or union, the name is the fully class qualified name of the data or function member.

For Example:

```
    class C{
    public:
            static int a;
            int func(int);
        };
      int C::a;
```

Name in .debug_pubnames: ―C::a‖ ―C::func(int)‖

Lookup By Address

For lookup by address, a table is maintained in .debug_aranges section.

uword	uhalf	uword	ubyte	ubyte	
length	version	offset	addr-size	seg-size	entries

Length: The length of entries for the set (not including this field).

Version: Version of DWARF format. The value is 2.

Offset: The offset from beginning of the .debug_info of the compilation unit entry referenced by the set.

addr-size: The size in bytes of address size.

seg-size: The segment size. This value is 0.

Entries: A variable number of beginning address/length. Each set of tuples is terminated by a 0 for the address and 0 for the length.

addr-size	Addr-size		addr-size	addr-size
address	Length	...	0	0

- address
The beginning address of text or data covered by this compilation unit.
- length
The length of this text or data range.

String Table of Debugging information

The string table is contained in .debug_str section.

String Table

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
\0	S	y	m	b	o	l	n	a	m	e	\0	V	a	r	i
a	b	l	e	\0											

Location Description

There are two forms in Location Description, which are Location Expression and Location Lists. The two forms are distinguished in DW_FORM. As the value of an attribute, a location expression is encoded as a block and a location list is encoded as a constant offset into a location list table.

a. Location Expression

1-byte	1-byte	
block_size	Operation	…

block_size: 1-byte unsigned integer. The length of Location expression except block_size field

Operation: 1-byte unsigned integer.

b. Location definitions

Expression	Value	No of op	Data Meaning
DW_OP_addr	0x3	1	constant address. operand size is the same address size
DW_OP_deref	0x6	0	
DW_OP_const1u	0x8	1	1-byte constant
DW_OP_const1s	0x9	1	1-byte constant
DW_OP_const2u	0xa	1	2-byte constant
DW_OP_const2s	0xb	1	2-byte constant
DW_OP_const4u	0xc	1	4-byte constant
DW_OP_const4s	0xd	1	4-byte constant
DW_OP_const8u	0xe	1	8-byte constant
DW_OP_const8s	0xf	1	8-byte constant
DW_OP_constu	0x10	1	ULEB128 constant
DW_OP_consts	0x11	1	SLEB128 constant
DW_OP_dup	0x12	0	
DW_OP_drop	0x13	0	
DW_OP_over	0x14	0	
DW_OP_pick	0x15	1	1-byte stack index
DW_OP_swap	0x16	0	
DW_OP_rot	0x17	0	
DW_OP_xderef	0x18	0	

DW_OP_abs	0x19	0	
DW_OP_and	0x1a	0	
DW_OP_div	0x1b	0	
DW_OP_minus	0x1c	0	
DW_OP_mod	0x1d	0	
DW_OP_mul	0x1e	0	
DW_OP_neg	0x1f	0	
DW_OP_not	0x20	0	
DW_OP_or	0x21	0	
DW_OP_plus	0x22	0	
DW_OP_plus_uconst	0x23	1	ULEB128 addend
DW_OP_shl	0x24	0	
DW_OP_shr	0x25	0	
DW_OP_shra	0x26	0	
DW_OP_xor	0x27	0	
DW_OP_skip	0x2f	1	signed 2-byte constant
DW_OP_bra	0x28	1	signed 2-byte constant
DW_OP_eq	0x29	0	
DW_OP_ge	0x2a	0	
DW_OP_gt	0x2b	0	
DW_OP_le	0x2c	0	
DW_OP_lt	0x2d	0	
DW_OP_ne	0x2e	0	
DW_OP_litn	0x30 - 0x4f	0	Literals 0..31
DW_OP_regn	0x50 - 0x6f	0	reg 0..31 See below
DW_OP_bregn	0x70 - 0x8f	1	breg 0..31 See below
DW_OP_regx	0x90	1	ULEB128 register. See below
DW_OP_fbreg	0x91	1	SLEB128 offset
DW_OP_bregx	0x92	2	ULEB128 register followed by SLEB128 offset. See below
DW_OP_piece	0x93	1	ULEB128 size of piece addressed
DW_OP_deref_size	0x94	1	1-byte size of data retrieved
DW_OP_xderef_size	0x95	1	1-byte size of data retrieved
DW_OP_nop	0x96	0	
DW_OP_lo_user	0xe0		
DW_OP_no_alloc	0xe0	0	A member of struct does not exist
DW_OP_hi_user	0xff		

KPIT tools

definition: cpu=SH

Expression	Value	No of op	op1	op2	Data Meaning
DW_OP_regn	0x50 - 0x6f	0	-	-	Rn : n = 0 - 31
DW_OP_bregn	0x70 - 0x8f	1	offset	-	@(op,Rn) : n = 0 - 31

c. Location Expression Examples

Symbol allocated to stack :

@(0,sp)

1-byte	1-byte	SLEB128
2	DW_OP_breg14	0
block_size	Operation	offset

Symbol allocated to memory

Global symbol

1-byte	1-byte	4-byte
5	DW_OP_addr	0x7c000034
block_size	Operation	address

Structure member

struct S{int s1, int s2};

s1

1-byte	1-byte	ULEB128
2	DW_OP_plus_uconst	0
block_size	Operation	offset

s2

1-byte	1-byte	ULEB128
2	DW_OP_plus_uconst	4
block_size	Operation	offset

Constant data

const int a=1;

1-byte	1-byte	2-byte
3	DW_OP_addr	0x2454
block_size	Operation	Address (in rodata section)

d. Location Lists

addr size	addr size	
beginning address	ending address	Location expression
beginning address	ending address	Location expression
:	:	:
0	0	

beginning address

It marks the beginning of the address range over which the location is valid. This address is relative*1 to the base address of the compilation unit referencing this location.

ending address

It marks the ending + 1 of the address range over which the location is valid. This address is relative*1 to the base address of the compilation unit referencing this location.

Remark:

*1: When DW_TAG_compile_unit does not have DW_AT_high_pc and DW_AT_low_pc (i.e. text section is divided using __attribute__ ((section)), and do not specify any optimization option to the compiler option), beginning address and ending address is absolute address.

3 Differences between Renesas SHC and KPIT GNUSH

These sections explain the ABI specification differences, ELF/DWARF2 specifications differences between Renesas SHC toolchain and KPIT GNUSH toolchain. It also details the migration guide for migrating Renesas SHC applications to GNUSH.

3.1 Migrating from Renesas SHC to GNUSH

Following section details the migration guide for migrating Renesas SHC applications to GNUSH

3.1.1 Preface

This section contains useful information for embedded software developers who wish to migrate their code from Renesas tools to KPIT Cummins GNU tools (and vice versa) for SH targets. At present, this section contains the following:

1. Compiler command Line options
2. Compiler directives
3. Assembler directives
4. Intrinsic functions
5. C and Math library Functions
6. Using Inline Assembly Language

3.1.2 Compiler command line options

Renesas Compiler Options	Equivalent GCC compiler options
-include = <path name>[,...] Include paths.	**-I<path>**
-preinclude = <file name>[,...] Includes the specified files at the head of compiled files.	**-include <file>**
-define = <sub> [,...] **<sub>: <macro name> [= <string literal>]** Defines <string literal> as <macro name>	**-D<macro>**

Renesas Compiler Options	Equivalent GCC compiler options
-nomessage [= <error code> [- <error code>] [,...]] Specifies whether to output information-level messages.	**-w**
-preprocessor [= <file name>] Preprocessor to named file.	**-E -o filename**
-object [= <object file name>] Set object filename.	**-o filename**
-Code=Asmcode -object=<file name>] Assembly output to named file.	**-S filename**
-debug Generate debug information.	**-g[level]**
-string = { Const | Data } Outputs string literal to constant or data sections.	**-fwritable-strings**
-template = { None | Static | Used | All | Auto } Specifies the condition to generate template instances.	**-fno-implicit-templates**
-List [= <file name>] Assembly output to prefixed List filename.	**-Wa, -al=file**
-show=<sub>[,...] **<sub>: {source | nosource | object | noobject | statistics | nostatistics | allocation | noallocation | expansion | noexpansion | width = <numeric value> | length = <numeric value> | tab = { 4 | 8 } }** Specifies the contents and format of the list output by the compiler, and the cancellation of list output.	**-a[sub-option...]** **Sub-options [default hls]:** **c omit false conditionals** **d omit debugging directives** **h include high-level source** **l include assembly** **m include macro expansions** **n omit forms processing** **s include symbols** **=FILE list to FILE (must be last sub-option)**

Renesas Compiler Options	Equivalent GCC compiler options
-speed = <sub> [,…] **<sub>:{register shiift |** **loop [= { 1 | 2 }] | switch |** **inline[=<numeric value>] |** **struct | expression}** Speed optimization	-O[1-3]
-optimize = { 0 | 1 } Optimize for size.	**-Os**
-comment Enables nested comments to be written.	By default enabled. To generate warning for nested comments use option. **-Wcomment**
-macsave The contents of the **MAC** register always remain unchanged after an interrupt function is called.	**-mnomacsave**
-pack Specifies the boundary alignment of structures, unions, and class members.	**-fpack-struct**
-volatile When **volatile** is specified, the compiler does not optimize external variables.	**-fvolatile**
-cmncode Common subexpression elimination	**-fgcse**
-align16 Align labels after unconditional branches are aligned to a 16 byte boundary.	**-falign**
-loop Performs loop unrolling.	**-funroll-all-loops**
-cpu = {sh1|sh2|sh2e|sh3|sh4}	**-m1 -m2 -m2e -m3 -m4**
-endian = { big | little } [SH-3 - SH-4] specifies Endian type.	**-mb or -ml**

Renesas Compiler Options	Equivalent GCC compiler options
-fpu= {single | double} [SH-4] Processes floating-point operation in single or double precision.	**-fsingle-precision-constant**
-pic= { 0 | 1 } Position independent code	**-fpic**
-double=float Converting double-type values to Float-type values.	**-fshort-double**
-rtti = { ON |OFf } Controlling runtime type information.	**-fno-rtti**
-exception Enables the C++ exception processing (try, catch, throw).	**-fexceptions**
-lang = { C | CPp } Specifies the language of the source program.	**-x <language>**

Options only in GCC

-fomit-frame-pointer	Register r14 in case of GCC is used as frame pointer. To avoid it from being pushed and popped during a function call, -fomit-frame-pointer option is specified.

3.1.3 Compiler directives

Renesas Directives	Equivalent GNU directive or method

#pragma interrupt (ext_0(vect=8))
void ext_0(void)
{
p0=6;
}
 	
Code for functions in which #pragma interrupt is specified are automatically generated as interrupt functions by the compiler. Return from interrupts by RTE instruction, required register save recovery, and other processing are performed in an interrupt function.
	#define VECT_SECT
	__attribute__((section (".vects")))
	#define ISR __attribute__
	((interrupt_handler))
	typedef void (*function_ptr) (void);
	extern volatile int P0;
	extern void start (void);
	void ext_0() ISR;
	function_ptr HardwareVectors[]
	VECT_SECT =
	{
	
	start,0,0,0,0,0,0,0,ext_0,0,0,0,0,0,0,0
	};
	
	void ext_0()
	{
	P0 = 6;
	}

extern char STK[100];
#pragma interrupt (f(sp=STK+100))
or
__interrupt(sp=STK+100) void f(void);
	extern char STK[100];
	void f() __attribute__
	((interrupt_handler,
	sp_switch
	("STK+100")));

extern char STK[100];
#pragma interrupt (g(tn=2))
or
__interrupt(tn=2) void g(void);
	trap_exit
	void g() __attribute__
	((interrupt_handler,
	trap_exit(2)));

Renesas Directives	Equivalent GNU directive or method
#pragma abs8(c1) #pragma abs16(i1) char c1; /* c1 is assigned to $ABS8B */ int i1; /* i1 is assigned to $ABS16B */ *The #pragma absn (n: 16, 20, 28, or 32) directive is a declaration that tells the compiler that the variable or function is in the n-bit address area. The default is the 32-bit address area.*	const int i1 __attribute__ ((section(¯_cseg‖))); const char c1[] __attribute__((section(¯_cseg‖)))={"Hello World"}; Please note this requires an entry in the linker section for the memory location of ¯_cseg‖. For ex. in the linker script _cseg :{*(_cseg);} > memory region [ram/rom]
#pragma entry <function name> [<entry specification>] e.g. #pragma entry INIT(sp=0x8000) void INIT(){:} *When specification is performed in the entry function, save and restore codes for registers can be suppressed.*	Achieved through linker script.
#pragma inline (<function name>[,...]) _ _inline <type specifier> <function name> *Inline expansion is a type of optimization that inserts the body of a function at the point at which the function is called. You can use inline expansion when you expect that it will reduce function call overhead, making the program smaller and allowing it to run faster.*	inline void foo (const char) __attribute__((always_inline));

Renesas Directives	Equivalent GNU directive or method
#pragma global_register(x=R4) or _ _ global_register(R4L) char x; /* External variable x is allocated to R4 */ *#pragma global_register directive can be used to allocate global variables to registers. By allocating global variables to registers, you can reduce the number of load and store instructions.*	You can define a global register variable in GNU C like this: register int *x asm ("r4");

#pragma stacksize <constant>
e.g. #pragma stacksize 100
<Code expansion example>
.SECTION S, STACK
.RES.W 50
 	
Creates a stack section
#pragma stacksize can be specified in stacksct.h (List 2-2) to reserve a 0x400-byte stack area (S section) in the compiler. When a stack is used from higher addresses down to lower addresses, the start address of the S section needs to be (stack-pointer-address - stack-size).
	char stack[100] __attribute__((section ("STACK"))) = { 0 };
	
	main()
	{
	    /* Initialize stack pointer */
	    init_sp (stack + sizeof (stack));
	}

#pragma section [{<name> | <numeric value>}] *If you specify abs16, abs20, abs28, or abs32 for a variable or function, use the #pragma section directive to switch the section so that the section is placed in an address area that can be represented by the specified bit addressing during linkage.*	Pragma is not available but the functionality can be achieved using __attribute__ (section(¯<section name>‖)) For e.g: extern void foobar (void) __attribute__ ((section ("bar"))); Alternatively, this can be achieved using the linker script. In section definition, the contents of an output section can be specified by listing particular input files, listing particular input-file sections, or combination of the two.

Renesas Directives	Equivalent GNU directive or method
#pragma inline_asm(<function name> [(size=numeric value)][,...]) *In a C source file, you can code functions in assembly language by using the #pragma inline_asm directive to declare that the functions are written in assembly language. Such assembly functions are called inline assembly functions. In the #pragma inline_asm directive, you can also specify a size=numeric-value option to specify the size of an assembly function. Specifying this option might improve the efficiency of optimization.*	asm(<assembly-language instruction string>);

3.1.4 Assembler Directives

Renesas Assembler Directives	GNU Assembler Directives
.AIF	#if or .if
.AELSE	#else or .else
.AENDI	#endif or .endif
Sym_name: .ASSIGN value	set sym_name , value
.ALIGN <boundary alignment value>	.align <boundary alignment value> (for details on more options of aligning like filling specific values,etc, please see link below)
sym_name: .EQU value	.equ sym_name , value
.EXPORT	.global
.GLOBAL	.global
.IMPORT sym_name	.extern sym_name

Renesas Assembler Directives	GNU Assembler Directives
Sym_name .RES.B 1	.size sym_name , 1 .sym_name: .byte initial_value
Sym_name .RES.L 1	.size sym_name , 4 .sym_name: .long initial_value
Sym_name .RES.W 1	.size sym_name, 2 .sym_name: .word initial_value
sym_name: .EQU B' value (binary value)	.equ sym_name , 0x value (equivalent hex value)
.SECTION name, attribute	.section ¯ name‖ , ¯attributes‖ (for details on the possible attributes, please refer the link given below)
Sym_name: .SDATA ¯str‖	Sym_name: .string ¯str‖
Sym_name: .DATA.W H'18D Sym_name: .DATA.L H'18D Sym_name: .DATA.B H'8D	Sym_name: .word 0x18d Sym_name: .long 0x18d Sym_name: .byte 0x8d

3.1.5 Intrinsic Functions:

Download intrinsic functions Header file from http://www.kpitgnutools.com/manuals/functions_sh.zip

Renesas Intrinsic functions	Equivalent GCC inline code

void set_cr(int cr)	extern __inline__ void set_cr(int cr)
	{
	    asm("mov %0, r0"::"r"(cr):"r0");
	    asm("ldc r0, sr");
	}

int get_cr(void)	extern __inline__ int get_cr(void)
	{
	    long val;
	    asm("stc sr, r0":::"r0");
	    asm("mov.l r0, %0":"=g"(val));

Renesas Intrinsic functions	Equivalent GCC inline code

	return val;
	}

void set_imask(int mask)	extern __inline__ void set_imask(int mask)
	{
	asm ("mov.l r0,@-r15");
	mask <<= 4;
	mask &= 0xf0;
	
	asm("stc sr,r0");
	asm("mov #0xff,r3");
	asm("shll8 r3");
	asm("add #0x0f,r3");
	asm("and r3,r0");
	asm("mov %0,r3"::"r"(mask));
	asm("or r3,r0");
	asm("ldc r0,sr");
	asm("mov.l @r15+,r0");
	}

int get_imask(void)	extern __inline__ int get_imask()
	{
	int val;
	asm("mov.l r0,@-r15");
	asm("stc sr,r0");
	asm("shlr2 r0");
	asm("shlr2 r0");
	asm("and #15,r0");
	asm("mov.l r0,%0":"=g"(val));
	asm("mov.l @r15+,r0");
	return val;
	}

void set_vbr(void *base)	extern __inline__ void set_vbr(void *vbr)
	{
	asm("mov %0, r2"::"r"(vbr):"r2");
	asm("ldc r2,vbr");
	}

void *get_vbr(void)	extern __inline__ void* get_vbr(void)

Renesas Intrinsic functions	Equivalent GCC inline code

	{
	    void *ptr;
	    asm("stc vbr,r2":::"r2");
	    asm("mov.l r2, %0":"=m"(ptr));
	    return ptr;
	}

void set_gbr(void *base)	extern __inline__ void set_gbr(void *gbr)
	{
	    asm("mov %0, r2"::"r"(gbr):"r2");
	    asm("ldc r2, gbr");
	}

void *get_gbr(void)	extern __inline__ void* get_gbr(void)
	{
	    void *ptr;
	    asm("stc gbr,r2":::"r2");
	    asm("mov.l r2, %0":"=m"(ptr));
	    return ptr;
	}

unsigned char gbr_read_byte(int offset)
unsigned short gbr_read_word(int offset)
unsigned long gbr_read_long(int offset)
	extern __inline__ unsigned char gbr_read_byte(int offset)
	{
	    unsigned char val;
	    asm("mov %0, r1"::"r"(offset):"r1");
	    asm("stc gbr, r2":::"r2");
	    asm("add r1, r2");
	    asm("mov.b @r2, r0");
	    asm("extu.b r0, r2");
	    asm("mov.l r2, %0":"=m"(val));
	    return val;
	}
	Similarly gbr_read_word (int offset) and gbr_read_long(int offset)

void gbr_write_byte(int offset, unsigned char data)
void gbr_write_word(int offset, unsigned short data)
void gbr_write_long(int offset,
	extern __inline__ void gbr_write_word(int offset, unsigned short data)
	{
	    asm("mov %0, r1"::"r"(offset):"r1");

Renesas Intrinsic functions	Equivalent GCC inline code

unsigned long data)	asm("stc gbr, r2":::"r2");
	asm("add r1, r2");
	asm("mov %0,r1"::"r"(data));
	asm("mov r1,r0");
	asm("mov.w r0,@r2");
	}
	Similarly gbr_write_byte and gbr_write_long

void gbr_and_byte(int offset, unsigned char mask)
void gbr_or_byte(int offset, unsigned char mask)
void gbr_xor_byte(int offset, unsigned char mask)
void gbr_tst_byte(int offset, unsigned char mask)
	extern __inline__ void gbr_and_byte(int offset, unsigned char mask)
	{
	asm("mov %0, r0"::"r"(offset):"r0");
	asm("and.b %0, @(r0,gbr)"::"g"(mask));
	}
	Please replace occurrence of gbr_and_byte(offset,mask) with above.
	Similarly gbr_or_byte, gbr_xor_byte and gbr_tst_byte.

void sleep(void)	extern __inline__ void sleep(void)
	{
	asm("sleep");
	}

void tas(char *addr)	extern __inline__ void tas(char *addr)
	{
	asm("mov %0, r1"::"r"(addr):"r1");
	asm("tas.b @r1");
	}

void trapa(unsigned int trap_no)	asm("trapa %0"::"g"(trap_no)) Please replace occurance of trapa(trap_no) with above.
int trapa_svc(int trap_no, type1 para1, type2 para2, type3 para3, type4 para4)	Under Process
void prefetch(void *p)	extern __inline__ void prefetch(void

Renesas Intrinsic functions	Equivalent GCC inline code

	*p)
	{
	asm("mov %0, r2"::"r"(p):"r2");
	asm("pref @r2");
	}
	Available only for **sh3** and **sh4**

void trace(long v)	Under process. Available only for **sh3** and **sh4**

void ldtlb(void)	extern __inline__ void ldtlb(void)
	{
	asm("ldtlb");
	}
	Available only for **sh3** and **sh4**

void nop(void)	extern __inline__ void nop(void)
	{
	asm("nop");
	}

unsigned long dmuls_l(long data1, long data2)
unsigned long dmulu_l(unsigned long data1,
unsigned long data2)
	extern __inline__ unsigned long dmuls_l(long data1, long data2)
	{
	unsigned long val;
	asm("mov %0, r4"::"r"(data1):"r4");
	asm("mov %0, r5"::"r"(data2):"r5");
	asm("dmuls.l r4,r5");
	asm("sts macl,r6":::"r6");
	asm("sts mach,r2":::"r2");
	asm("lds r6,macl");
	asm("sts macl,r2");
	asm("mov.l r2, %0":"=m"(val));
	return val;
	}
	Similarly dmulu_l().
	dmuls _l() is not available for **sh1**.
	dmulu _l() available only for **sh1**.

long dmuls_h(long data1, long data2)	extern __inline__ long dmuls_h(long

Renesas Intrinsic functions	Equivalent GCC inline code

dmulu_h(unsigned long data1,
unsigned long data2)
	data1, long data2)
	{
	unsigned long val;
	asm("mov %0, r4"::"r"(data1):"r4");
	asm("mov %0, r5"::"r"(data2):"r5");
	asm("dmuls.l r4,r5");
	asm("sts macl,r6":::"r6");
	asm("sts mach,r2":::"r2");
	asm("lds r2,mach");
	asm("sts mach,r6");
	asm("mov.l r6, %0":"=m"(val));
	return val;
	}
	Similarly dmulu_h().
	Not available for **sh1**.

unsigned short swapb(unsigned short data)
unsigned long swapw(unsigned long data)
	extern __inline__ unsigned long swapw(unsigned long data)
	{
	unsigned long val;
	asm("mov %0, r2"::"r"(data):"r2");
	asm("swap.w r2,r6":::"r6");
	asm("mov.l r6, %0":"=m"(val));
	return val;
	}
	Similarly swapb()

Unsigned long end_cnvl(unsigned long data)	extern __inline__ unsigned long end_cnvl(unsigned long data)
	{
	unsigned long val;
	asm("mov %0, r6"::"r"(data):"r6");
	asm("mov r6,r2":::"r2");
	asm("swap.b r2,r6");
	asm("swap.w r6,r2");
	asm("swap.b r2,r6");
	asm("mov.l r6, %0":"=m"(val));
	return val;
	}

int macw(short *ptr1, short *ptr2, unsigned int count)	extern __inline__ int macl(int *ptr1, int *ptr2, unsigned int count)

Renesas Intrinsic functions	Equivalent GCC inline code

int macl(int *ptr1, int *ptr2,
unsigned int count)
	{
	int val;
	asm("mov %0,r4"::"r"(count):"r4");
	asm("clrmac");
	asm("sts macl,r2":::"r2");
	asm("sts mach,r6":::"r6");
	asm("tst r4,r4");
	asm("bt l11");
	asm("mov %0, r6"::"r"(ptr1):"r6");
	asm("mov %0, r5"::"r"(ptr2):"r5");
	asm("shll2 r4");
	asm("add r6,r4");
	asm("l13:");
	asm("lds r2,macl");
	asm("mac.l @r6+,@r5+");
	asm("sts macl,r2");
	asm("cmp/hi r6,r4");
	asm("bt l13");
	asm("l11:");
	asm("lds r2,macl");
	asm("sts macl,r6");
	asm("mov.l r6, %0":"=m"(val));
	return val;
	}
	Similarly int macw(short *ptr1, short *ptr2, unsigned int count)
	macl() is not available for **sh1**.

int macwl(short *ptr1, short *ptr2, unsigned int count, unsigned int mask)
 	
int macll(int *ptr1, int *ptr2, unsigned int count, unsigned int mask)
	extern __inline__ int macll(int *ptr1, int *ptr2,
	unsigned int count, unsigned int mask)
	{
	int val;
	asm("mov %0,r4"::"r"(count):"r4");
	asm("clrmac");
	asm("sts macl,r2":::"r2");
	asm("sts mach,r6":::"r6");
	asm("tst r4,r4");
	asm("bt l14");
	asm("mov %0, r6"::"r"(ptr1):"r6");
	asm("mov %0, r5"::"r"(ptr2):"r5");
	asm("mov %0, r7"::"r"(mask):"r7");
	asm("shll2 r4");

Renesas Intrinsic functions	Equivalent GCC inline code

	asm("add r6,r4");
	asm("l16:");
	asm("lds r2,macl");
	asm("mac.l @r6+,@r5+");
	asm("sts macl,r2");
	asm("and r7,r5");
	asm("cmp/hi r6,r4");
	asm("bt l16");
	asm("l14:");
	asm("lds r2,macl");
	asm("sts macl,r6");
	asm("mov.l r6, %0":"=m"(val));
	return val;
	}
	Similarly int macwl(short *ptr1, short *ptr2,
	unsigned int count, unsigned int mask)
	macll() is not available for **sh1**.

void set_fpscr(int cr)	extern __inline__ void set_fpscr(int cr)
	{
	asm("mov %0, r2"::"r"(cr):"r2");
	asm("lds r2,fpscr");
	}
	Available only for **sh2e**, **sh3e** and **sh4**.

int get_fpscr()	extern __inline__ int get_fpscr()
	{
	int val;
	asm("sts fpscr, r2":::"r2");
	asm("mov.l r2, %0":"=m"(val));
	return val;
	}
	Available only for **sh2e**, **sh3e** and **sh4**.
	C:\Program Files\Renesas\Hew\Tools\KPIT Cummins\GNUSH-ELF\v0702\Doc\gcc.html - SEC83

float fipr(float vect1[4], float vect2[4])	extern __inline__ float fipr(float vect1[4], float vect2[4]) {

Renesas Intrinsic functions	Equivalent GCC inline code

	float val;
	asm("mov %0,r2"::"r"(vect1):"r2");
	asm("fmov.s @r2+,fr0":::"fr0");
	asm("fmov.s @r2+,fr1":::"fr1");
	asm("fmov.s @r2+,fr2":::"fr2");
	asm("fmov.s @r2+,fr3":::"fr3");
	asm("add #-16,r2");
	asm("fmov fr3,fr8":::"fr8");
	asm("fmov fr2,fr9":::"fr9");
	asm("mov %0, r2"::"r"(vect2):"r2");
	asm("fmov.s @r2+,fr4":::"fr4");
	asm("fmov.s @r2+,fr5":::"fr5");
	asm("fmov.s @r2+,fr6":::"fr6");
	asm("fmov.s @r2+,fr7":::"fr7");
	asm("add #-16,r2");
	asm("fmov fr9,fr2");
	asm("fmov fr8,fr3");
	asm("fipr fv4,fv0");
	asm("fmov fr3,fr8");
	// asm("fmov fr8, %0":"=f"(val));
	// return val;
	}
	Available only for **sh4**.

void ftrv(float vec1[4],float vec2[4])	extern __inline__ void ftrv(float vec1[4],float vec2[4])
	{
	asm("mov %0,r6"::"r"(vec1):"r6");
	asm("fmov.s @r6+,fr0");
	asm("fmov.s @r6+,fr1");
	asm("fmov.s @r6+,fr2");
	asm("fmov.s @r6+,fr3");
	asm("add #-16,r6");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("ftrv xmtrx,fv0");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("mov %0,r6"::"r"(vec2));
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("add #16,r6");

Renesas Intrinsic functions	Equivalent GCC inline code

	asm("fmov.s fr3,@-r6");
	asm("fmov.s fr2,@-r6");
	asm("fmov.s fr1,@-r6");
	asm("fmov.s fr0,@-r6");
	}
	Available only for **sh4**.

void ftrvadd(float vec1[4], float vec2[4], float vec3[4])
 	
void ftrvsub(float vec1[4], float vec2[4], float vec3[4])
	extern __inline__ void ftrvsub(float vec1[4],
	float vec2[4], float vec3[4])
	{
	asm("mov %0,r2"::"r"(vec1):"r2");
	asm("fmov.s @r2+,fr0");
	asm("fmov.s @r2+,fr1");
	asm("fmov.s @r2+,fr2");
	asm("fmov.s @r2+,fr3");
	asm("add #-16,r2");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("ftrv xmtrx,fv0");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("mov %0,r2"::"r"(vec2));
	asm("fmov.s @r2+,fr4");
	asm("fmov.s @r2+,fr5");
	asm("fmov.s @r2+,fr6");
	asm("fmov.s @r2+,fr7");
	asm("add #-16,r2");
	asm("fsub fr4,fr0");
	asm("fsub fr5,fr1");
	asm("fmov fr8,fr2");
	asm("fsub fr6,fr2");
	asm("fmov fr2,fr8");
	asm("fmov fr9,fr3");
	asm("fsub fr7,fr3");
	asm("fmov fr3,fr9");
	asm("mov %0,r2"::"r"(vec3));
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("add #16,r2");
	asm("fmov.s fr3,@-r2");
	asm("fmov.s fr2,@-r2");

Renesas Intrinsic functions	Equivalent GCC inline code

	asm("fmov.s fr1,@-r2");
	asm("fmov.s fr0,@-r2");
	}
	Similarly ftrvadd().
	Available only for **sh4**.

void add4(float vec1[4], float vec2[4], float vec3[4])
 	
void sub4(float vec1[4], float vec2[4], float vec3[4])
	extern __inline__ void add4(float vec1[4],
	float vec2[4], float vec3[4])
	{
	asm("mov %0,r2"::"r"(vec1):"r2");
	asm("fmov.s @r2+,fr0");
	asm("fmov.s @r2+,fr1");
	asm("fmov.s @r2+,fr2");
	asm("fmov.s @r2+,fr3");
	asm("add #-16,r2");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("mov %0,r2"::"r"(vec2));
	asm("fmov.s @r2+,fr4");
	asm("fmov.s @r2+,fr5");
	asm("fmov.s @r2+,fr6");
	asm("fmov.s @r2+,fr7");
	asm("add #-16,r2");
	asm("fadd fr4,fr0");
	asm("fadd fr5,fr1");
	asm("fmov fr8,fr2");
	asm("fadd fr6,fr2");
	asm("fmov fr2,fr8");
	asm("fmov fr9,fr3");
	asm("fadd fr7,fr3");
	asm("fmov fr3,fr9");
	asm("mov %0,r2"::"r"(vec3));
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("add #16,r2");
	asm("fmov.s fr3,@-r2");
	asm("fmov.s fr2,@-r2");
	asm("fmov.s fr1,@-r2");
	asm("fmov.s fr0,@-r2");
	}
	Similarly sub4()
	Available only for **sh4**.

Renesas Intrinsic functions	Equivalent GCC inline code

void mtrx4mul(float mat1[4][4],
            float mat2[4][4])
	extern __inline__ void mtrx4mul(float mat1[4][4],
	            float mat2[4][4])
	{
	    asm("mov %0,r2"::"r"(mat1):"r2");
	    asm("mov %0,r6"::"r"(mat2):"r6");
	    asm("fmov.s @r2+,fr0");
	    asm("fmov.s @r2+,fr1");
	    asm("fmov.s @r2+,fr2");
	    asm("fmov.s @r2+,fr3");
	    asm("fmov fr3,fr8");
	    asm("fmov fr2,fr9");
	    asm("fmov fr9,fr2");
	    asm("fmov fr8,fr3");
	    asm("ftrv xmtrx,fv0");
	    asm("fmov fr3,fr8");
	    asm("fmov fr2,fr9");
	    asm("fmov fr9,fr2");
	    asm("fmov fr8,fr3");
	    asm("add #16,r6");
	    asm("fmov.s fr3,@-r6");
	    asm("fmov.s fr2,@-r6");
	    asm("fmov.s fr1,@-r6");
	    asm("fmov.s fr0,@-r6");
	    asm("fmov.s @r2+,fr0");
	    asm("fmov.s @r2+,fr1");
	    asm("fmov.s @r2+,fr2");
	    asm("fmov.s @r2+,fr3");
	    asm("fmov fr3,fr8");
	    asm("fmov fr2,fr9");
	    asm("fmov fr9,fr2");
	    asm("fmov fr8,fr3");
	    asm("ftrv xmtrx,fv0");
	    asm("fmov fr3,fr8");
	    asm("fmov fr2,fr9");
	    asm("add #32,r6");
	    asm("fmov fr9,fr2");
	    asm("fmov fr8,fr3");
	    asm("fmov.s fr3,@-r6");
	    asm("fmov.s fr2,@-r6");
	    asm("fmov.s fr1,@-r6");
	    asm("fmov.s fr0,@-r6");
	    asm("fmov.s @r2+,fr0");
	    asm("fmov.s @r2+,fr1");

Renesas Intrinsic functions	Equivalent GCC inline code

	asm("fmov.s @r2+,fr2");
	asm("fmov.s @r2+,fr3");
	asm("fmov fr3,fr8");
	asm("fmov fr2,fr9");
	asm("fmov fr9,fr2");
	asm("fmov fr8,fr3");
	asm("ftrv xmtrx,fv0");
	asm("fmov fr3,fr8");
	asm("fmov fr2,fr9");
	asm("add #32,r6");
	asm("fmov fr9,fr2");
	asm("fmov fr8,fr3");
	asm("fmov.s fr3,@-r6");
	asm("fmov.s fr2,@-r6");
	asm("fmov.s fr1,@-r6");
	asm("fmov.s fr0,@-r6");
	asm("fmov.s @r2+,fr0");
	asm("fmov.s @r2+,fr1");
	asm("fmov.s @r2+,fr2");
	asm("fmov.s @r2+,fr3");
	asm("fmov fr3,fr8");
	asm("fmov fr2,fr9");
	asm("fmov fr9,fr2");
	asm("fmov fr8,fr3");
	asm("ftrv xmtrx,fv0");
	asm("fmov fr3,fr8");
	asm("fmov fr2,fr9");
	asm("add #32,r6");
	asm("fmov fr9,fr2");
	asm("fmov fr8,fr3");
	asm("fmov.s fr3,@-r6");
	asm("fmov.s fr2,@-r6");
	asm("fmov.s fr1,@-r6");
	asm("fmov.s fr0,@-r6");
	}
	Available only for **sh4**.

void mtrx4muladd(float mat1[4][4], float mat2[4][4], float mat3[4][4])
 	
void mtrx4mulsub(float mat1[4][4], float mat2[4][4], float mat3[4][4])
	extern __inline__ void mtrx4mulsub(float mat1[4][4], float mat2[4][4], float mat3[4][4])
	{
	asm("mov %0,r6"::"r"(mat1):"r6");
	asm("mov %0,r2"::"r"(mat2):"r2");
	asm("mov %0,r5"::"r"(mat3):"r5");
	asm("fmov.s @r6+,fr0");

Renesas Intrinsic functions	Equivalent GCC inline code

	asm("fmov.s @r6+,fr1");
	asm("fmov.s @r6+,fr2");
	asm("fmov.s @r6+,fr3");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("ftrv xmtrx,fv0");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("fmov.s @r2+,fr4");
	asm("fmov.s @r2+,fr5");
	asm("fmov.s @r2+,fr6");
	asm("fmov.s @r2+,fr7");
	asm("fsub fr4,fr0");
	asm("fsub fr5,fr1");
	asm("fmov fr8,fr2");
	asm("fsub fr6,fr2");
	asm("fmov fr2,fr8");
	asm("fmov fr9,fr3");
	asm("fsub fr7,fr3");
	asm("fmov fr3,fr9");
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("add #16,r5");
	asm("fmov.s fr3,@-r5");
	asm("fmov.s fr2,@-r5");
	asm("fmov.s fr1,@-r5");
	asm("fmov.s fr0,@-r5");
	asm("fmov.s @r6+,fr0");
	asm("fmov.s @r6+,fr1");
	asm("fmov.s @r6+,fr2");
	asm("fmov.s @r6+,fr3");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("ftrv xmtrx,fv0");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("fmov.s @r2+,fr4");
	asm("fmov.s @r2+,fr5");
	asm("fmov.s @r2+,fr6");
	asm("fmov.s @r2+,fr7");

Renesas Intrinsic functions	Equivalent GCC inline code

	asm("fsub fr4,fr0");
	asm("fsub fr5,fr1");
	asm("fmov fr8,fr2");
	asm("fsub fr6,fr2");
	asm("fmov fr2,fr8");
	asm("fmov fr9,fr3");
	asm("fsub fr7,fr3");
	asm("fmov fr3,fr9");
	asm("add #32,r5");
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("fmov.s fr3,@-r5");
	asm("fmov.s fr2,@-r5");
	asm("fmov.s fr1,@-r5");
	asm("fmov.s fr0,@-r5");
	asm("fmov.s @r6+,fr0");
	asm("fmov.s @r6+,fr1");
	asm("fmov.s @r6+,fr2");
	asm("fmov.s @r6+,fr3");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("ftrv xmtrx,fv0");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("fmov.s @r2+,fr4");
	asm("fmov.s @r2+,fr5");
	asm("fmov.s @r2+,fr6");
	asm("fmov.s @r2+,fr7");
	asm("fsub fr4,fr0");
	asm("fsub fr5,fr1");
	asm("fmov fr8,fr2");
	asm("fsub fr6,fr2");
	asm("fmov fr2,fr8");
	asm("fmov fr9,fr3");
	asm("fsub fr7,fr3");
	asm("fmov fr3,fr9");
	asm("add #32,r5");
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("fmov.s fr3,@-r5");
	asm("fmov.s fr2,@-r5");
	asm("fmov.s fr1,@-r5");

Renesas Intrinsic functions	Equivalent GCC inline code

	asm("fmov.s fr0,@-r5");
	asm("fmov.s @r6+,fr0");
	asm("fmov.s @r6+,fr1");
	asm("fmov.s @r6+,fr2");
	asm("fmov.s @r6+,fr3");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("ftrv xmtrx,fv0");
	asm("fmov fr3,fr9");
	asm("fmov fr2,fr8");
	asm("fmov.s @r2+,fr4");
	asm("fmov.s @r2+,fr5");
	asm("fmov.s @r2+,fr6");
	asm("fmov.s @r2+,fr7");
	asm("fsub fr4,fr0");
	asm("fsub fr5,fr1");
	asm("fmov fr8,fr2");
	asm("fsub fr6,fr2");
	asm("fmov fr2,fr8");
	asm("fmov fr9,fr3");
	asm("fsub fr7,fr3");
	asm("fmov fr3,fr9");
	asm("add #32,r5");
	asm("fmov fr8,fr2");
	asm("fmov fr9,fr3");
	asm("fmov.s fr3,@-r5");
	asm("fmov.s fr2,@-r5");
	asm("fmov.s fr1,@-r5");
	asm("fmov.s fr0,@-r5");
	}
	Similarly mtrx4muladd().
	Available only for **sh4**.

void ld_ext(float mat[4][4])	extern __inline__ void ld_ext(float mat[4][4])
	{
	asm("mov %0, r2"::"r"(mat):"r2");
	asm("frchg");
	asm("fmov.s @r2+,fr0":::"fr0");
	asm("fmov.s @r2+,fr1":::"fr1");
	asm("fmov.s @r2+,fr2":::"fr2");
	asm("fmov.s @r2+,fr3":::"fr3");
	asm("fmov.s @r2+,fr4":::"fr4");

Renesas Intrinsic functions	Equivalent GCC inline code

	asm("fmov.s @r2+,fr5":::"fr5");
	asm("fmov.s @r2+,fr6":::"fr6");
	asm("fmov.s @r2+,fr7":::"fr7");
	asm("fmov.s @r2+,fr8":::"fr8");
	asm("fmov.s @r2+,fr9":::"fr9");
	asm("fmov.s @r2+,fr10":::"fr10");
	asm("fmov.s @r2+,fr11":::"fr11");
	asm("fmov.s @r2+,fr12":::"fr12");
	asm("fmov.s @r2+,fr13":::"fr13");
	asm("fmov.s @r2+,fr14":::"fr14");
	asm("fmov.s @r2+,fr15":::"fr15");
	asm("frchg");
	asm("add #-64,r2");
	}
	Available only for **sh4**.

void st_ext(float mat[4][4])	extern __inline__ void st_ext(float mat[4][4])
	{
	asm("mov %0,r2"::"r"(mat):"r2");
	asm("add #64,r2");
	asm("frchg");
	asm("fmov.s fr15,@-r2");
	asm("fmov.s fr14,@-r2");
	asm("fmov.s fr13,@-r2");
	asm("fmov.s fr12,@-r2");
	asm("fmov.s fr11,@-r2");
	asm("fmov.s fr10,@-r2");
	asm("fmov.s fr9,@-r2");
	asm("fmov.s fr8,@-r2");
	asm("fmov.s fr7,@-r2");
	asm("fmov.s fr6,@-r2");
	asm("fmov.s fr5,@-r2");
	asm("fmov.s fr4,@-r2");
	asm("fmov.s fr3,@-r2");
	asm("fmov.s fr2,@-r2");
	asm("fmov.s fr1,@-r2");
	asm("fmov.s fr0,@-r2");
	asm("frchg");
	}
	Available only for **sh4**.

long clipsb(long data)	extern __inline__ long clips_byte(long var) {

Renesas Intrinsic functions	Equivalent GCC inline code

	volatile long result;
	asm ("mov %0,r2"::"r"(var):"r2");
	asm ("clips.b r2");
	asm ("mov r2,%0":"=r"(result):);
	return result;
	}
	Available only for sh2a.

long clipsw(long data)	extern __inline__ long clips_word(long var)
	{
	volatile long result;
	asm ("mov %0,r3"::"r"(var):"r3");
	asm ("clips.w r3");
	asm ("mov r3,%0":"=r"(result):);
	return result;
	}
	Available only for sh2a.

unsigned long clipub(unsigned long data)	extern __inline__ unsigned long clipu_byte(unsigned long var)
	{
	volatile unsigned long result;
	asm ("mov %0,r4"::"r"(var):"r4");
	asm ("clipu.b r4");
	asm ("mov r4,%0":"=r"(result):);
	return result;
	}
	Available only for sh2a.

unsigned long clipuw(unsigned long data)	extern __inline__ unsigned long clipu_word(unsigned long var)
	{
	volatile unsigned long result;
	asm ("mov %0,r5"::"r"(var):"r5");
	asm ("clipu.w r5");
	asm ("mov r5,%0":"=r"(result) :);
	return result;
	}
	Available only for sh2a.

void set_tbr(void *data)	extern __inline__ void set_tbr(void *tbr)
	{
	asm("mov %0,r6"::"r"(tbr):"r6");
	asm("ldc r6,tbr");

Renesas Intrinsic functions	Equivalent GCC inline code
	} **Available only for sh2a.**

void *get_tbr(void)	extern __inline__ void * get_tbr(void)
	{
	void *ptr;
	asm("stc tbr,r7":::"r7");
	asm("mov r7,%0":"=r"(ptr));
	return ptr;
	}
	Available only for sh2a.

Not provided by Renesas	extern __inline__ void stbank(long data, int rn, int bn)
	{
	volatile long address=0;
	asm("mov.l r0,@-r15");
	asm("mov %0,r0"::"r"(data):"r0");
	rn<<=2;
	address |= rn;
	bn<<=7;
	address |= bn;
	asm("mov %0,r8"::"r"(address):"r8");
	asm("stbank r0,@r8");
	asm("mov.l @r15+,r0");
	}
	Available only for sh2a.

Not provided by Renesas	extern __inline__ long ldbank(int rn, int bn)
	{
	volatile long address=0,result;
	rn<<=2;
	address |= rn;
	bn<<=7;
	address |= bn;
	asm("mov %0,r9"::"r"(address):"r9");
	asm("ldbank @r9,r0");
	asm("mov r0,%0":"=r"(result):);
	return result;
	}
	Available only for sh2a.

3.1.6 C and Math Library Functions

GNU C library namely Newlib is under free software license. The Newlib supports various processors and architectures.

a. Complete set of Standard input/output functions.

b. Float parameters support for Math library.

c. Option is available to build the libraries with floating point support.

d. A customizable math error handler matherr() is available.

e. All the functions are reentrant.

Drawbacks Involved

a. malloc() is used in every stdio routine.

b. There is no support for low-level routines to connect to the target.

C library Math Functions only in Renesas

No.	Function
General Utilities (stddef.h)	
1	Offsetof

C library Math Functions only in GNU

No.	Function	Description
Character type handling routines (ctype.h) Please refer the http://sources.redhat.com/newlib/libc.html#SEC35 for more information on these routines.		
1	Isascii	ASCII character predicate
2	Toascii	Control character predicate
3	Iswalnum	Alphanumeric wide character test
4	Iswalpha	Alphabetic wide character test
5	Iswcntrla	Wide character cntrl test
6	Iswdigit	Decimal digit wide character test
7	Iswgraph	Graphic wide character test
8	Iswlower	Lowercase wide character test
9	Iswprint	Printable wide character test
10	Iswpunct	Punctuation wide character test
11	Iswspace	Wide character space test
12	Iswupper	Uppercase wide character test
13	Iswxdigit	Hexadecimal digit wide character test
14	Iswctype	Extensible wide character test
15	Wctype	Get wide character classification type
16	*Towlower	Translate wide characters to lower case
17	*Towupper	Translate wide characters to upper case
18	Towctrans	Extensible wide character case mapping
19	Wctrans	Get wide character translation type

No.	Function	Description
Mathematics Functions (math.h) Please refer the http://sources.redhat.com/newlib/libm.html#SEC1 for more information on these routines.		
1	acosh, acoshf	Inverse hyperbolic cosine
2	asinh, asinhf	Inverse hyperbolic sine
3	atanh, atanhf	Inverse hyperbolic tangent
4	jN,jNf,yN,yNf	Bessel functions
5	erf, erff, erfc, erfcf	Error function
6	gamma, gammaf, lgamma, lgammaf, gamma_r, hypot, hypotf	Distance from origin
7	isnan, isnanf, isinf, isinff, finite, finitef	Test for exceptional numbers
8	remainder, remainderf	Round and remainder
9	cbrt, cbrtf	Cube root
10	copysign, copysignf	Sign of y, magnitude of x
11	expm, expmf	Exponential minus 1
12	ilogb, ilogbf	Get exponent of floating point number
13	infinity, infinityf	Representation of infinity
14	logp, logpf	Log of 1 + x
15	Matherr	Modifiable math error handler
16	nan, nanf	Representation of infinity
17	nextafter, nextafterf	Get next number
18	scalbn, scalbnf	Scale by integer

No.	Function	Description
General Utilities (stdlib.h) Please refer the http://sources.redhat.com/newlib/libc.html#SEC1 for more information on these routines.		
1	Atexit	Request execution of functions at program exit
2	Atoff	String to double or float
3	Ecvt, ecvtf, fcvt, fcvtf	Double or float to string
4	Gvcvt, gcvtf	Format double or float as string
5	Ecvtbuf, fcvtbuf	Double or float to string
6	__env_lock, __env_unlock	Lock environ variable
7	Getenv	Look up environment variable
8	Mallinfo, malloc_stats, mallopt	Malloc support
9	__malloc_lock, __malloc_unlock	Lock malloc pool
10	Mblen	Minimal multibyte length function
11	Mbstowcs	Minimal multibyte string to wide char converter
12	Mbtowc	Minimal multibyte to wide char converter
13	Rand48, drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48	Pseudo random number generators and initialization routines
14	Strtof	String to double or float
15	System	Execute command string
16	Wcstombs	Minimal wide char string to multibyte string converter
17	Wctomb	Minimal wide char to multibyte converter

No.	Function	Description
String and Memory routines (string.h) Please refer the http://sources.redhat.com/newlib/libc.html#SEC109 for more information on these routines.		
1	Bcmp	Compare two memory areas
2	Bcopy	Copy memory regions
3	Bzero	Initialize memory to zero
4	Index	Search for character in string
5	Memccpy	Copy memory regions with end token check
6	Mempcpy	Copy memory regions and return end pointer
7	Rindex	Reverse search for character in string
8	Strcasecmp	Case insensitive character string compare
9	Strlwr	Force string to lower case
10	Strncasecmp	Case insensitive character string compare
11	strtok_r,strsep	Get next token from a string
12	Strupr	Force string to uppercase
13	Swab	Swap adjacent bytes

No.	Function	Description
Input/output routines (stdio.h) Please refer the http://sources.redhat.com/newlib/libc.html#SEC67 for more information on these routines.		
1	Fgetpos	Record position in a stream or file
2	Fiprintf	Format output to file (integer only)
3	Fdopen	Turn open file into a stream
4	Freopen	Open a file using an existing file descriptor
5	fseeko	Set file position
6	Fsetpos	Restore position of a stream or file
7	ftello	Return position in a stream or file

8	Getw	Read a word (int)
9	Iprintf	Write formatted output (integer only)
10	mktemp, mkstemp	Generate unused file name
11	Putw	Write a word (int)
12	Remove	Delete a file's name
13	Rename	Rename a file
14	Setbuf	Specify full buffering for a file or stream
15	Siprintf	Write formatted output (integer only)
16	asprintf, snprintf	Format output
17	Tmpfile	Create a temporary file
18	Tmpnam,	Name for a temporary file
Large file input / output:		
19	Fopen64	Open a large file
20	Freopen64	Open a large file using an existing file descriptor
21	Ftello64	Return position in a stream or file
22	Fseeko64	Set file position for large file
23	Fgetpos64	Record position in a large stream or file
24	Fsetpos64	Restore position of a large stream or file
25	Tmpfile64	Create a large temporary file

Along with all the above, following set of routines are provided in the newlib library:

- Wide character strings (wchar.h):
  Please refer http://sources.redhat.com/newlib/libc.html#SEC144
- Signal handling (signal.h):
  Please refer http://sources.redhat.com/newlib/libc.html#SEC169
- Time functions (time.h):
  Please refer http://sources.redhat.com/newlib/libc.html#SEC172
- Locale specific routines (locale.h):
  Please refer http://sources.redhat.com/newlib/libc.html#SEC183

Useful Links

http://sources.redhat.com/newlib/libc.html
http://sources.redhat.com/newlib/libm.html

3.1.7 Using Inline Assembly Language

We can instruct the compiler to insert the code of a function into the code of its callers, to the point where actually the call is to be made. Such functions are inline functions.

This method of inlining reduces the function-call overhead. And if any of the actual argument values are constant, their known values may permit simplifications at compile time so that not all of the inline function‘s code needs to be included. The effect on code size is less predictable, it depends on the particular case. To declare an inline function, we‘ve to use the keyword inline in its declaration. Inline assembly is important primarily because of its ability to operate and make its output visible on C variables.

Because of this capability, "asm" works as an interface between the assembly instructions and the "C" program that contains it.

Syntax

GCC, the GNU C Compiler for Linux, uses AT&T&sol;UNIX assembly syntax.

a) Source-Destination Ordering
The first operand is the source and the second operand is the destination.
ie, "Op-code src dst".
This is similar to the syntax followed in Renesas.

b) Register Naming
Register names are prefixed by % ie, if r0 is to be used, write %r0.

c) Immediate Operand
Immediate operands are preceded by ‘#‘. For hexadecimal constants a ‘0x‘ is suffixed to the constant.
However, in order to support hexadecimal constants suffixed with a 'H', please pass ‗-h-tick-hex‘ option to the assembler.
e.g., Use the following command to build the ¯asm("mov #H'03,R0");‖ code statement,
$ sh-elf-gcc test.c -Wa,-h-tick-hex

d) Operand Size
Size of memory operands is determined from the last character of the op-code name.
Op-code suffixes of ‘b‘, ‘w‘, and ‘l‘ specify byte(8-bit), word(16-bit), and long (32-bit) memory references.

e) Memory Operands
Base register is enclosed in ‗(‘ and ‘)‘ and indirect memory reference is like ¯section&colon;disp(base, index, scale)‖. When a constant is used for disp and sol; scale, ‘#‘ shouldn‘t be prefixed.

3.2 ABI specification differences between Renesas SHC and KPIT GNUSH

This section details ABI specification differences between Renesas SHC and KPIT GNUSH.

3.2.1 Glossary

Aggregate: structure and union in C.

Scalar: antonym of aggregate, i.e. char, short and int.

Frame: stack space pushed for a function invocation.

3.2.2 Byte Ordering

The SH architecture supports both big-endian byte ordering and little-endian byte ordering. Big-endian code and little-endian code may not be mixed in the same program.

3.2.3 Use of floating-point unit (fpu)

Code may be created either to use the floating-point unit or to perform all floating-point operations in software. Code that uses the floating-point unit is said to use the fpu model and the code that performs all floating-point operations in software is said to use the no-fpu model. The choice of floating-point model affects the way that function arguments and results are passed.

In general, fpu model code and no-fpu model code may not be mixed in the same program.

3.2.4 Stack Layout

- Each called function creates and deletes its own frame.
- The stack grows from high address to low address.
- The top of stack (i.e. the lowest address) is always referenced by a register known as the stack pointer, SP.
- The stack pointer is always 4 byte aligned.
- The topmost frame is the frame of the currently executing function. When a function is called, it allocates its own frame by decreasing SP; on exit, it deletes the frame by restoring SP to the value upon entry. Each function is responsible for creating and deleting its own frame. Not all functions will require a stack frame and a stack frame is allocated only if required.
- As well as the stack pointer, a frame may also have a frame pointer, FP, a register used to address parts of the frame. Only a subset of frames need frame pointers.
- If a stack frame uses a frame pointer, the frame pointer is held in R14.

3.2.5 Data type sizes and alignments

The following table shows the size and alignment for all data types:

Type	Size (bytes)	Alignment (bytes)
char	1 byte	1 byte
short	2 byte	2 byte
int	4 byte	4 byte
long	4 byte	4 byte
long long	8 byte	4 byte
float	4 byte	4 byte
Double[#1]	8 byte	4 byte
long Double	8 byte	4 byte
Pointer	4 byte	4 byte

#1 Size of Double is 4 bytes in SH3e.

3.2.6 Register Usage

A register is **Caller Save** if its value is not guaranteed to be preserved across function calls.

A register is **Callee Save** if its value is guaranteed to be preserved across calls. The implication is that the callee will either not modify the register or else save it to memory.

A register is **Reserved** if it has some special use required either by software convention or by the hardware. They are not used by the compiler.

The registers and there usage is given below:
R0-R1: Return value, caller saves

R2-R3: Scratch, Caller saves

R2: Large struct return address, caller save (when -mhitachi is not specified)
R4-R7: Parameter passing, caller saves
R8-R13: Callee Saves
R14: Frame Pointer, FP, callee saves
R15: Stack Pointer, SP, callee saves
FR0-FR3: Return value, caller saves
FR4-FR11: Parameter passing, caller saves

FR12-FR15: Callee saves
MACH: Caller saves
MACL: Caller saves
(If **-mhitachi** option is used MAC registers are Callee save.)

PR: Linkage register (saves the subroutine return
address), caller saves
SR: Status register
GBR: Reserved
VBR: Reserved

3.2.7 Frame Pointer

R14 is used as the frame pointer. **-fomit-frame-pointer** can be used to eliminate the use of the frame pointer in favor of the stack pointer.

Function linkage and parameter passing
Parameter Passing
Registers R4-R7, FR4-FR11, and stack are used for parameter passing.

a. The first four arguments are passed in registers R4 through R7. The floating-point arguments are passed in fr4 through fr11. All the remaining arguments are pushed onto the stack, last to first, so that the lowest numbered argument not passed in a register is at the lowest address in the stack.

b. These registers are always filled in SH3. For e.g. if you have function foo (int a, int b, int c, long long d) a, b, c are passed in r4, r5, r6 respectively but d is passed partly in r7 and partly on stack.
But for SH3e and SH4 the entire argument is passed on stack. Hence, the parameter d will be passed entirely on stack.

c. If -mhitachi option is used the aggregate types are passed using stack. E.g. in following code, the members of structure s in the foo(s) function call are passed using stack. This would have been passed using registers without the option.

```
typedef  struct  _S  {  int  a;
}S; S s ;
S foo (S s)
{
     return s ;
}
void bar()
{
     s = foo(s) ;
}
```

Every stack push is rounded to a multiple of 4 bytes. If the size of a value pushed onto the stack is different from the size of an actual push, downward padding is done. For example, if 2 bytes of data, 0x1234, are to be pushed onto the stack, 4 bytes are pushed like so:

sp + 3 0x34
sp + 2 0x12
sp + 1 padding (unknown value)
sp + 0 padding (unknown value)

d. Extension of arguments: when you pass a char or short to a func or get a return value, the toolchain does not sign extend any values. It remains as it is.

3.2.8 Function Value

The scalar function values that are not more than 4 bytes in size is returned in R0 and FR0 and values not more than 8 bytes are returned in R0 and R1.

If R0 and R1 are used to return the function value, R0 contains the most significant part and R1 contains the least significant part for big-endian mode and vice-versa for little-endian mode.

For all other types, the function value is always returned in memory. Specifically, the caller allocates an instance of the aggregate type and passes a pointer to the instance as an invisible argument. The caller stores the function value into the memory location pointed to by the invisible pointer.

3.2.9 Bit-Field

A bit-field is allocated within a storage unit whose alignment and size are same as the alignment and size of the underlying type of the bit-field. The bit-field must reside entirely within this storage unit: a bit-field never straddles the natural boundary of the underlying type.

In the little-endian ABI, bit-fields are allocated from right to left (least to most significant) within a storage unit. In the big-endian ABI, bit-fields are allocated from left to right (most to least significant) within a storage unit.

A bit-field shares a storage unit with the previous structure member if there is sufficient space within the storage unit.

3.2.10 Structure Alignment

Every structure member is aligned to 4 bytes unless **__attribute__ ((packed))** is used.

3.3 ELF specifications for Renesas SHC and KPIT

Following section details the ELF specification differences between Renesas SHC and KPIT GNUSH toolchain.

3.3.1 ELF Specification Differences

3.3.1.1 Elf Header

An *ELF header* resides at the beginning and holds a "road map" describing the file's organization.

Field name		**Size**	**Renesas SHC**			**KPIT GNUSH-ELF**		
			Value	**Meaning**		**Value**	**Meaning**	
e_ident[16]	e_ident[0]	1	0x7f	Magic number		0x7f	Magic number	
	[1]	1	'E'			'E'		
	[2]	1	'L'			'L'		
	[3]	1	'F'			'F'		
	[4]	1	0x0	File class	: Invalid class	0x0	File class	: Invalid class
			0x1		: ELFCLASS32	0x1		: ELFCLASS32
			0x2		: ELFCLASS64	0x2		: ELFCLASS64
	[5]	1	0x0	Data encoding	: Invalid encoding	0x0	Data encoding	: Invalid encoding
			0x1		: Little endian	0x1		: 2s Complement, Little endian
			0x2		: Big endian	0x2		: 2s Complement, Big endian

	[6]	1	0x1	File version		0x0	File version	: Invalid version
						0x1		: Current version
	[7]	1	0x0	Padding		0x0	Operating System/ABI Specification	: UNIX System V ABI
						0x1		: HP-UNIX operating system
						0xff		: Standalone (embedded) app
	[8]	1	0x0			0x0	ABI Version	
	[9]~[15]	7	0x0			0x0	Padding	
e_type		2	0x0	Object file type	: No file type	0x0	Object file type	: No file type
			0x1		: Relocatable file	0x1		: Relocatable file
			0x2		: Executable file	0x2		: Executable file
			0x3		: Shared file	0x3		: Shared file
			0x4		: Core file	0x4		: Core file
						0xfe00		: Operating system-specific
						0xfeff		: Operating system-specific
						0xff00		: Processor - specific
						0xffff		: Processor - specific

e_machine	2	0x2a	CPU type	: SuperH	0x2a	CPU type	: SuperH
e_version	4	0x1	Object file version		0x0	Object file version	: Invalid version
					0x1		: Current version
e_entry	4	0x0	Entry point	: Relocatable file	0x0	Entry point	: Relocatable file
		Address		: Absolute file	Address		: Absolute file
e_phoff	4	0x0	Program header table's file offset	: Relocatable file	0x0	Program header table's file offset	: Relocatable file
		Offset		: Absolute file	Offset		: Absolute file
e_shoff	4	Offset	Section header table's file offset		Offset	Section header table's file offset	
e_flags	4	0x0		: neither fpu nor dsp	0x0		: SH Unknown
		0x1		: fpu	0x1		: SH1
		0x2		: dsp	0x2		: SH2
					0x3		: SH3
		0x4		: SH-2A	0x4		: SH DSP
		0x5		: SH-2A (FPU)	0x5		: SH3 DSP
					0x6		: SH4AL DSP
					0x8		: SH3E

					0x9		: SH4
					0xa		: SH5
					0xb		: SH2E
					0xc		: SH4A
					0xd		: SH2A
					0x10		: SH4 No FPU
					0x11		: SH4A No FPU
					0x12		: SH4 No MMU, No FPU
					0x13		: SH2A No FPU
					0x14		: SH3 No MMU
					0x15		: SH2A or SH4 No FPU
					0x16		: SH2A or SH3 No FPU
					0x17		: SH2A or SH4
					0x18		: SH2A or SH3E
e_ehsize	2	0x34	ELF header's size		0x34	ELF header's size	
e_phentsize	2	0x0	Size of one entry in program header table	: Relocatable file	0x0	Size of one entry in program header table	: Relocatable file
		0x20		: Absolute file	0x20		: Absolute file

e_phnum	2	0x0	Number of entry in program header table	: Relocatable file	0x0	Number of entry in program header table	: Relocatable file
		Number		: Absolute file	Number		: Absolute file
e_shentsize	2	0x28	Size of one entry in section header table		0x28	Size of one entry in section header table	
e_shnum	2	Number	Number of entry in section header table		Number	Number of entry in section header table	
e_shstrndx	2	Index	Section header table index of the entry associated with section name string table		Index	Section header table index of the entry associated with section name string table	

3.3.1.2 Program Header

Field name	Size	Renesas SHC		KPIT GNUSH-ELF	
		Value	Meaning	Value	Meaning
p_type	4		Segment type		Segment type
		0x0	Undefined	0x0	Undefined
		0x1	Loadable segment	0x1	Loadable segment
		0x2	Dynamic linking information segment	0x2	Dynamic linking information segment
		0x3	Interpreter segment	0x3	Program Interpreter segment
		0x4	Note segment	0x4	Note segment
		0x5	Undefined	0x5	Undefined
		0x6	Segment which specifies the location and size of the program header table both in the file and in the memory	0x6	Segment which specifies the location and size of the program header table both in the file and in the memory
				0x7	Thread local storage segment
				0x60000000	OS-specific
				0x6FFFFFFF	OS-specific
				0x70000000	Processor-specific
				0x7FFFFFFF	Processor-specific
p_offset	4	Offset	Segment file offset	Offset	Segment file offset
p_vaddr	4	Address	Virtual address	Address	Virtual address
p_paddr	4	0x0	Physical address	Address	Physical address
p_filesz	4	Size	Segment size in file	Size	Segment size in file
p_memsz	4	Size	Segment size in memory	Size	Segment size in memory
p_flags	4		Segment permissions		Segment permissions
		0x0	All access denied	0x0	All access denied
		0x1	Execute	0x1	Execute Only
		0x2	Write	0x2	Write Only
		0x3	Write, execute	0x3	Write, Execute
		0x4	Read	0x4	Read Only

		0x5	Read, execute	0x5	Read, Execute
		0x6	Read, write	0x6	Read, Write
		0x7	Read, write, execute	0x7	Read, Write, Execute
p_align	4	Alignment	Address alignment constraints	Alignment	Address alignment constraints

3.3.1.3 Section Header

An object file's section header table lets one locate all the file's sections.

Field name	Size	Renesas SHC			KPIT GNUSH-ELF		
		Value	Meaning		Value	Meaning	
sh_name	4	Index	Index into section header string table		Index	Index into section header string table	
sh_type	4		Section type			Section type	
		0x0	Undefined section	:SHT_NULL	0x0	Undefined section	: SHT_NULL
		0x1	Program section	:SHT_PROGBITS	0x1	Program section	: SHT_PROGBITS
		0x2	Symbol table section	:SHT_SYMTAB	0x2	Symbol table section	: SHT_SYMTAB
		0x3	String table section	:SHT_STRTAB	0x3	String table section	: SHT_STRTAB
		0x4	Relocation table with addends	:SHT_RELA	0x4	Relocation table with addends	: SHT_RELA
		0x5	Symbol hash table section	:SHT_HASH	0x5	Symbol hash table section	: SHT_HASH
		0x6	Dynamic linking information section	:SHT_DYNAMIC	0x6	Dynamic linking information section	: SHT_DYNAMIC
		0x7	Note section	:SHT_NOTE	0x7	Note section	: SHT_NOTE
		0x8	Uninitialized data section	:SHT_NOBITS	0x8	Uninitialized data section	: SHT_NOBITS

		0x9	Relocation table with no addends	:SHT_REL	0x9	Relocation table with no addends	: SHT_REL
		0xA	Unspecified section	:SHT_SHLIB	0xA	Unspecified section	: SHT_SHLIB
		0xB	Dynamic linking symbol table section	:SHT_DYNSYM	0xB	Dynamic linking symbol table section	: SHT_DYNSYM
		0x80000000	**Inter-module optimization info section**	**:SHT_HITACHI_IOP**	0xE	**Array of pointers to init functions**	**: SHT_INIT_ARRAY**
					0xF	Array of pointers to finish functions	: SHT_FINI_ARRAY
					0x10	Array of pointers to pre-init functions	: SHT_PREINIT_ARRAY
					0x11	Section contains a section group	: SHT_GROUP
					0x12	Indices for SHN_XINDEX entries	: SHT_SYMTAB_SHNDX
					0x60000000	First of OS specific semantics	: SHT_LOOS
					0x6FFFFFFF	Last of OS specific semantics	: SHT_HIOS
					0x6FFFFFF7	List of prelink dependencies	: SHT_GNU_LIBLIST

						0x70000000	Processor-specific semantics, low	: SHT_LOPROC
						0x7FFFFFFF	Processor-specific semantics, high	: SHT_HIPROC
						0x80000000	Application-specific semantics, low	: SHT_LOUSER
						0xFFFFFFFF	Application-specific semantics, high	: SHT_HIUSER
sh_flags	4	flags << 28 +Section attribute				0x0	All access denied	
		flags	**0x8**	**absolute address section**	**:SHF_ABS**	0x1	Write Only	: SHF_WRITE
			0x4	**section contains only SHmedia code**	**:SHF_SH5_ISA32**	0x2	Allocate (Read) Only	: SHF_ALLOC
			0x2	**section contains a mixture of SHmedia and SHcompact code**	**:SHF_SH5_ISA_MIXED**	0x3	Write, Allocate	: SHF_WRITE + SHF_ALLOC
						0x4	Execute Only	: SHF_EXECINSTR
		attrib	0x0	Undefined		0x5	Write, Execute	: SHF_WRITE + SHF_EXECINSTR
			0x1	SHF_WRITE		0x6	Execute, Allocate	: SHF_ALLOC + SHF_WRITE + SHF_EXECINSTR

			0x2	SHF_ALLOC	0x7	Allocate, Write, Execute	: SHF_ALLOC + SHF_EXECINSTR
			0x3	SHF_WRITE+SHF_ALLOC	0x10	Data in this section can be merged	: SHF_MERGE
			0x4	SHF_EXECINSTR	0x20	Contains null terminated character strings	: SHF_STRINGS
			0x5	SHF_WRITE + SHF_EXECINSTR	0x40	Holds section header table index	: SHF_INFO_LINK
			0x6	SHF_ALLOC + SHF_EXECINSTR	0x80	Preserve section ordering when linking	: SHF_LINK_ORDER
			0x7	SHF_WRITE + SHF_ALLOC + SHF_EXECINSTR	0x100	OS specific processing required	: SHF_OS_NONCONFORMING
					0x200	Member of a section group	: SHF_GROUP
					0x400	Thread local storage section	: SHF_TLS
					0x0FF00000	OS-specific semantics	: SHF_MASKOS
					0xF0000000	Processor-specific semantics	: SHF_MASKPROC
sh_addr	4			Start address		Start address	
		0x0		:Control section	0x0		: Control section

		Address		:Program section	Address		: Program section
sh_offset	4	Offset	Offset from the beginning of file to section		Offset	Offset from the beginning of file to section	
sh_size	4	Size	Section size		Size	Section size	
sh_link	4	Index	Section header index of associated symbol table	:SHT_REL,	Index	Section header index of associated symbol table	: SHT_REL,
				:SHT_RELA			: SHT_RELA
			Section header index of associated string table	:SHT_SYMTAB		Section header index of associated string table	: SHT_SYMTAB
		0x0	SHN_UNDEF	:Others	0x0	SHN_UNDEF	: Others
sh_info	4	Index	Section header index of the section to which the relocation applies	:SHT_REL,	Index	Section header index of the section to which the relocation applies	: SHT_REL,
				:SHT_RELA			: SHT_RELA
			Symbol table index grater than the last local symbol, that is, the symbol table index for the first non-local symbol	:SHT_SYMTAB		Symbol table index grater than the last local symbol, that is, the symbol table index for the first non-local symbol	: SHT_SYMTAB
		0x0		:Others	0x0		: Others
sh_addralign	4	Alignment	Address alignment constraints		Alignment	Address alignment constraints	

sh_entsize	4		Each entry size of section has fixed-size entry			Each entry size of section has fixed-size entry	
		0x0		:SHT_PROGBITS	0x0		: SHT_PROGBITS
				:SHT_NOBIT			: SHT_NOBIT
				:SHT_STRTAB			: SHT_STRTAB
		0xC		:SHT_RELA	0xC		: SHT_RELA
		0x10		:SHT_SYMTAB	0x10		: SHT_SYMTAB

3.3.1.4 Section Header

An object file's symbol table holds information needed to locate and relocate a program's symbolic definitions and references.

Field name	Size	Renesas SHC		KPIT GNUSH-ELF	
		Value	**Meaning**	**Value**	**Meaning**
st_name	4	0	No name	0	No name
		Index	String table index	Index	String table index
st_value	4	0	Case st_shndx: SHN_UNDEF	0	Case st_shndx: SHN_UNDEF
		Value	Case st_shndx: SHN_ABS	Value	Case st_shndx: SHN_ABS
		Alignment	Case st_shndx: SHN_COMMON	Alignment	Case st_shndx: SHN_COMMON
		Offset	Others (relocatable file):Offset from the beginning of the section that st_shndx	Offset	Others (relocatable file): Offset from the beginning of the section that st_shndx
		Address	Others(absolute file)	Address	Others (absolute file)
st_size	4	0	No size or unknown size	0	No size or unknown size
		Size	Symbol size	Size	Symbol size
st_info	1	Symbol Binding<<4 + Symbol Type		Symbol Binding<<4 + Symbol Type	

	Binding		0x0	STB_LOCAL		0x0	Symbol not visible outside obj	: STB_LOCAL
			0x1	STB_GLOBAL		0x1	Symbol visible outside obj	: STB_GLOBAL
			0x2	STB_WEEK	:Library global symbol	0x2	Like global, lower precedence	: STB_WEEK
						0xA	OS-specific semantics	: STB_LOOS
						0xC	OS-specific semantics	: STB_HIOS
						0xD	Application-specific semantics	: STB_LOPROC
						0xF	Application-specific semantics	: STB_HIPROC
	Type		0x0	STT_NOTYPE		0x0	Symbol type is unspecified	: STT_NOTYPE
			0x1	STT_OBJECT		0x1	Symbol is a data object	: STT_OBJECT
			0x2	STT_FUNC		0x2	Symbol is a code object	: STT_FUNC
			0x3	STT_SECTION		0x3	Symbol associated with a section	: STT_SECTION
			0x4	STT_FILE		0x4	Symbol gives a file name	: STT_FILE
			0xE	**STT_HITACHI_ENTRY**		0x5	An uninitialised common block	: STT_COMMON
						0x6	Thread local data object	: STT_TLS
						0xA	OS-specific semantics	: STT_LOOS
						0xC	OS-specific semantics	: STT_HIOS
						0xD	Application-specific semantics	: STT_LOPROC
			0xF	**STT_HITACHI_BIT_SYMBOL**		0xF	Application-specific semantics	: STT_HIPROC
st_other		1	0	No defined	:STO_UNDEF	0	No defined	: STO_UNDEF
			1	**Symbol is point to ISA32 codes**	**:STO_SH5_ISA32**	1	**Symbol is point to ISA32 codes**	**:STO_SH5_ISA32**
st_shndx		2	0	Symbol is undefined	:SHN_UNDEF	0	Undefined section reference	: SHN_UNDEF
			0xfff1	Symbol is not change because relocation	:SHN_ABS	0xFF00	Begin range of reserved indices	: SHN_LORESERVE

		0xfff2	Symbol labels common block	SHN_COMMON:	0xFF10	Begin range of application-specific	SHN_LOPROC
		Index	Section header table index of the section which contains symbol definition		0xFF1F	End range of application-specific	SHN_HIPROC
					0xFF20	OS specific semantics	SHN_LOOS
					0xFF3F	OS specific semantics	SHN_HIOS
					0xFFF1	Associated symbol is absolute	SHN_ABS
					0xFFF2	Associated symbol is in common	SHN_COMMON
					0xFFFF	Section index is held elsewhere	SHN_XINDEX
					0xFFFF	End range of reserved indices	SHN_HIRESERVE
					((unsigned) -1)		SHN_BAD

For further details on KPIT GNUSH ELF Specifications, please refer to http://www.kpitgnutools.com/ELF_Specifications.pdf

3.4 Interoperability between Renesas SHC and GNUSH

What is meant by ¯interoperability between Renesas SHC and GNUSH compiler‖?

Interoperability between Renesas SHC and GNUSH means it is possible that Renesas SHC generated relocatable object file can be linked with GNUSH linker and GNUSH generated relocatable (with ‗-mrenesas' option) can be linked with Renesas SHC linker.

Compiling with ‗-mrenesas' option in GNUSH generates code which follows Renesas SuperH calling conventions or Renesas ABI.

The SH Application Binary Interface (ABI) for GCC is available on our website at the following link:

http://www.kpitgnutools.com/manuals/SH-ABI-Specification.html

The SH Application Binary Interface (ABI) for Renesas SHC is mentioned in document available at the following link:

http://documentation.renesas.com/eng/products/tool/rej10b0152_sh.pdf

You may find some more information about difference in ABI at following link,

http://sources.redhat.com/ml/gdb-patches/2003-09/msg00508.html

3.4.1 Renesas ABI Implementations in GNUSH-ELF toolchain

The SH ABI for Renesas was not implemented fully (‗long long' and ‗double' data type were not following Renesas ABI requirements) in earlier toolchains of GNUSH. KPIT has applied a patch from v0601 GNUSH-ELF toolchain onwards which fixes the ABI differences between Renesas SHC compiler and GNUSH compiler with "-mrenesas" option for parameters of "double" and "long long" data types.

Applied patch implements the following Renesas ABI requirements in the GCC for SH targets.

a. Ensures that arguments of data type ‗long long' are passed on stack for all SH targets thus following Renesas SHC ABI.
(Previously function arguments of data type 'long long' were always passed in registers.) It's highlighted in yellow color in the table below.

b. Ensures that arguments of data type ‗long long' are returned in stack for all SH targets thus following Renesas SHC ABI.
(Previously function return value of data type 'long long' was always returned in registers for all targets.) It's highlighted in gray color in the table below.

For **example** consider the following code for SH2 target:

```
/* a.c */
long long func(long long
a){
return a;}

int main(void){
long long b, c;
c = func(b);
return 0;}
```

When the above code is compiled using the following command:
Command:

```
#sh-elf-gcc a.c -m2 -S -fomit-frame-pointer
-mrenesas
```

The assembly file _**a.s**' is as follows;

v0503 GNUSH-ELF toolchain(does not have the patch)	v0601 GNUSH-ELF toolchain(has the patch)
.file "a.c"	.file "a.c"
.text	.text
.text	.text
.align 1	.align 1
.global _func	.global _func
.type _func, @function	.type _func, @function
_func:	_func:
add #-8,r15	mov.l @r15,r0
mov.l r4,@r15	mov.l @(4,r15),r1
mov.l r5,@(4,r15)	mov.l @(8,r15),r2
mov.l @r15,r1	mov.l r1,@r0
mov.l @(4,r15),r2	mov.l r2,@(4,r0)
mov r1,r0	rts
mov r2,r1	nop
add #8,r15	.size _func, .-_func
rts	.align 1
nop	.global _main
.size _func, .-_func	.type _main, @function
.align 1	_main:
.global _main	sts.l pr,@-r15
.type _main, @function	add #-24,r15
_main:	mov r15,r1
sts.l pr,@-r15	add #16,r1

add #-16,r15	add #-12,r15
mov.l @(8,r15),r1	mov r15,r3
mov.l @(12,r15),r2	mov.l r1,@r3
mov r1,r4	mov.l @(20,r15),r1
mov r2,r5	mov.l @(24,r15),r2
mov.l .L5,r1	mov.l r1,@(4,r3)
jsr @r1	mov.l r2,@(8,r3)
nop	mov.l .L5,r1
mov r1,r2	jsr @r1
mov r0,r1	nop
mov.l r1,@r15	add #12,r15
mov.l r2,@(4,r15)	mov.l @(16,r15),r1
mov #0,r1	mov.l @(20,r15),r2
mov r1,r0	mov.l r1,@r15
add #16,r15	mov.l r2,@(4,r15)
lds.l @r15+,pr	mov #0,r1
rts	mov r1,r0
nop	add #24,r15
.L6:	lds.l @r15+,pr
.align 2	rts
.L5:	nop
.long _func	.L6:
.size _main, .-_main	.align 2
.ident "GCC: (GNU) 4.0-	.L5:
GNUSH_v0503"	.long _func
	.size _main, .-_main
	.ident "GCC: (GNU) 4.0-
	GNUSH_v0601"

c. Ensures that if the value returned by the function of data type 'long long' or 'double' is assigned to a local variable, then the local variable is updated in the function epilogue, thus following Renesas ABI.
(Previously under this condition the value was returned by the function in registers.)

d. Ensures that for SH targets without FPU, GCC library functions passes arguments of data type ‗double' on stack thus following Renesas SHC ABI.
(Previously the GCC library functions were passing argument of data type 'double' in registers.)

e. Ensures that GCC library functions return values of data type ‗double' on stack for all SH non-FPU targets thus following Renesas SHC ABI.
(Previously the GCC library functions were returning value of data type 'double' in registers.)

3.4.2 HEW Settings for using Interoperability

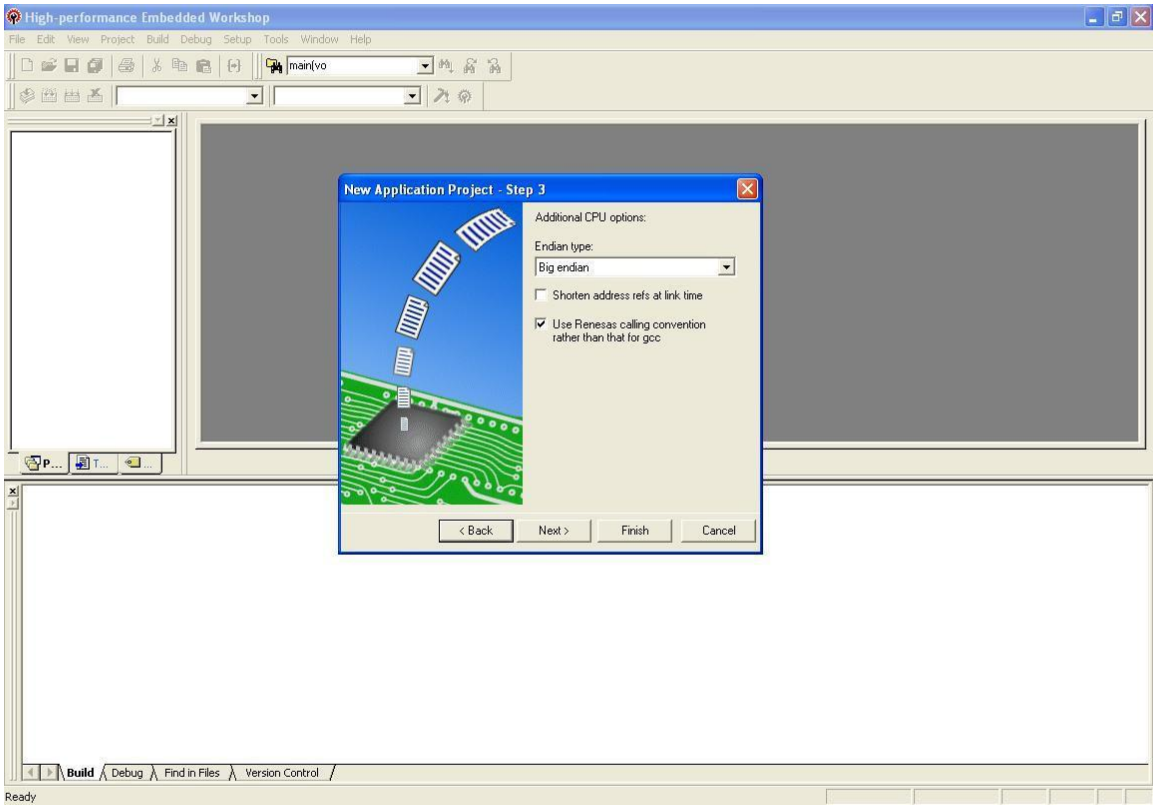
3.4.2.1 HEW Settings for linking Renesas SHC relocatable with GNUSH linker

(These settings are provided in GNUSH v0901 (and below) toolchain installers)

The following settings need to be done in GNUSH HEW project for linking a Renesas SHC relocatable with GNUSH linker.

Step 1

The GNUSH HEW project has to be configured to generate code that follows Renesas ABI. This can be done either while creating the project as shown in Figure1 or after creating the project in _Compiler Options‘ as shown in **Step 2** below.


High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Window Help
main(vo
New Application Project - Step 3
< Back
Next >
Finish
Cancel
Build
Debug
Find in Files
Version Control
Ready

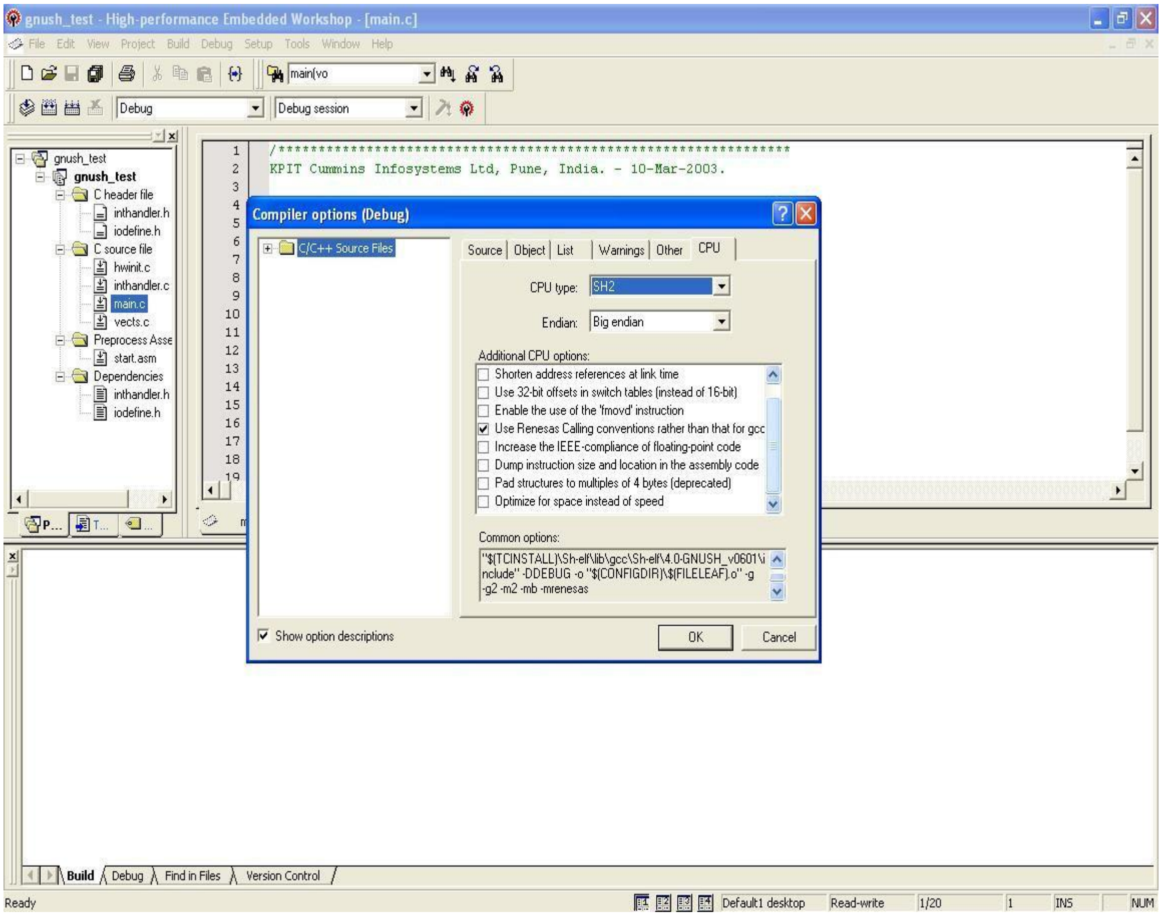
Additional CPU options:

Endian type: Big endian

☐ Shorten address refs at link time

☑ Use Renesas calling convention rather than that for gcc

Step 2



```
/*****************************************************************
KPIT Cummins Infosystems Ltd, Pune, India. - 10-Mar-2003.
```

Compiler options (Debug)

C/C++ Source Files

Source | Object | List | Warnings | Other | CPU

CPU type: SH2

Endian: Big endian

Additional CPU options:

- □ Shorten address references at link time
- □ Use 32-bit offsets in switch tables (instead of 16-bit)
- □ Enable the use of the 'fmovd' instruction
- ☑ Use Renesas Calling conventions rather than that for gcc
- □ Increase the IEEE-compliance of floating-point code
- □ Dump instruction size and location in the assembly code
- □ Pad structures to multiples of 4 bytes (deprecated)
- □ Optimize for space instead of speed

Common options:

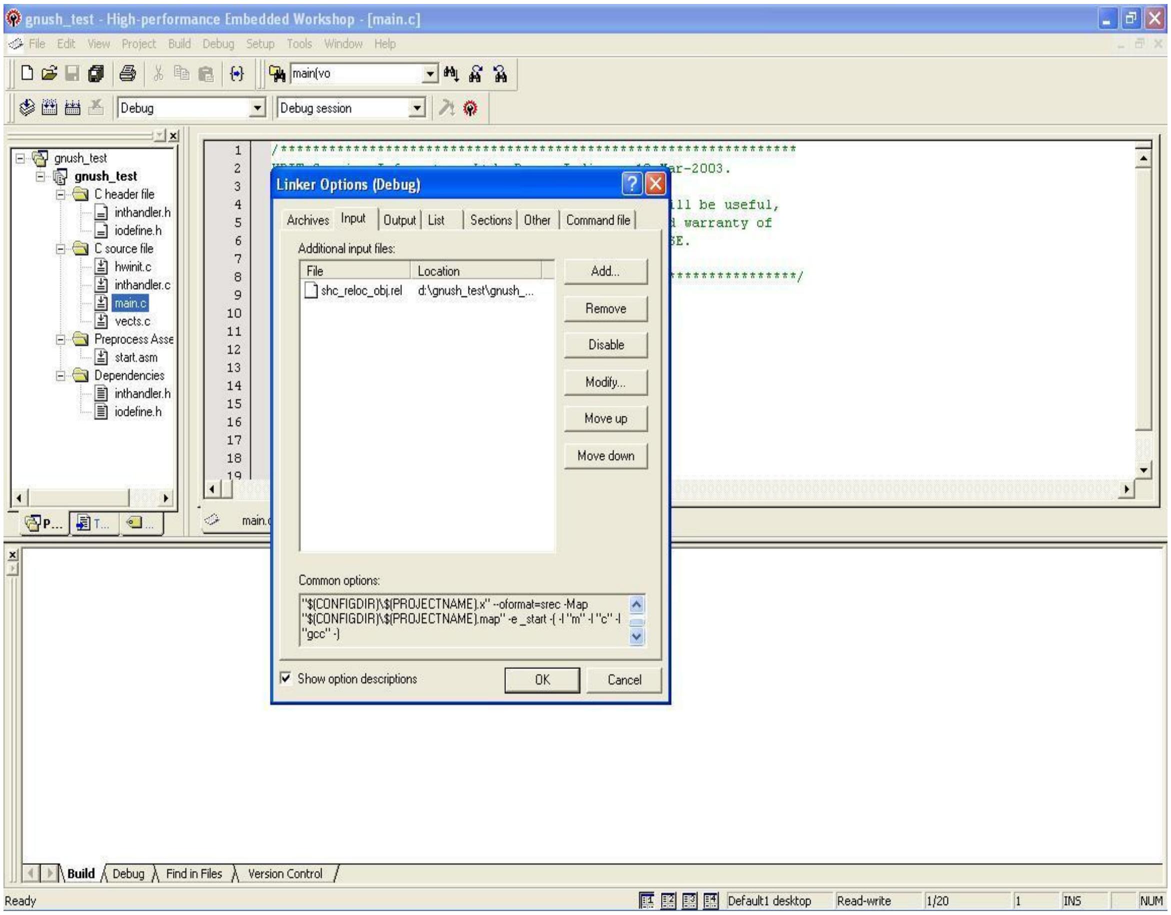
```
"$(TCINSTALL)\Sh-elf\lib\gcc\Sh-elf\4.0-GNUSH_v0601\include" -DDEBUG -o "$(CONFIGDIR)\$(FILELEAF).o" -g -g2 -m2 -mb -mrenesas
```

☑ Show option descriptions

OK Cancel

Step 3

The Renesas relocatable object file name can be specified in _Linker Options‘ as shown in Figure3.



Linker Options (Debug)

Archives | Input | Output | List | Sections | Other | Command file

Additional input files:

File	Location
shc_reloc_obj.rel	d:\gnush_test\gnush_...

Common options:

```
"$(CONFIGDIR)\$(PROJECTNAME).x" --oformat=srec -Map
"$(CONFIGDIR)\$(PROJECTNAME).map" -e _start -( -l "m" -l "c" -l
"gcc" -)
```

☑ Show option descriptions

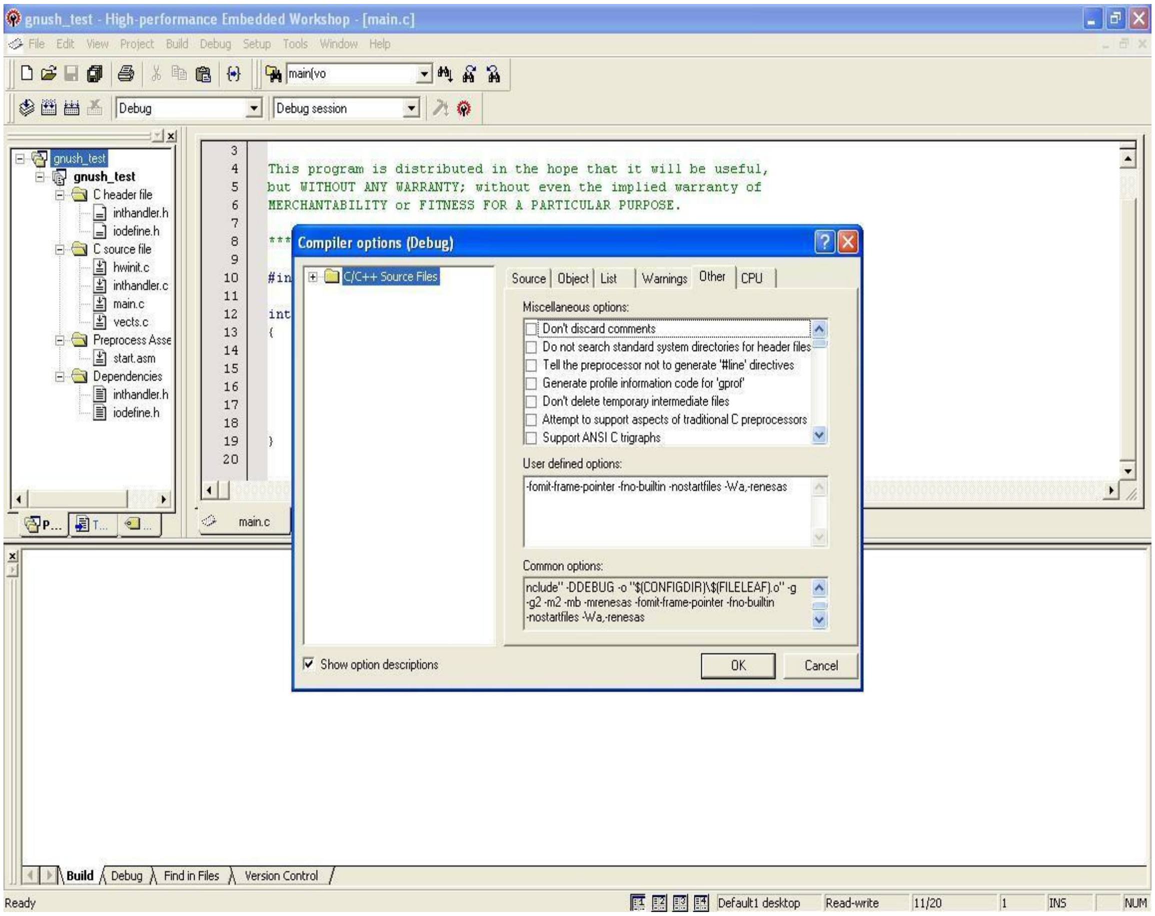
Step 4

The following options have to be passed to the GNUSH-ELF project for proper linking of Renesas relocatable with GNUSH relocatable files.

```
#-fomit-frame-pointer –fno-builtin –nostartfiles –Wa,-renesas
```

Note : The option **‗-Wa,-renesas'** passes the option ‗-renesas' to the assembler. This option disables optimization with section symbol for compatibility with Renesas assembler. In v0602, this option has to be passed manually as it is not enabled automatically with ‗-mrenesas'. This can be done as shown in figure below.

In v0702 toolchain, this option is automatically enabled on passing ‗-mrenesas' option and hence is not required to be passed separately.



gnush_test - High-performance Embedded Workshop - [main.c]

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Compiler options (Debug)

C/C++ Source Files

Source | Object | List | Warnings | Other | CPU

Miscellaneous options:

- □ Don't discard comments
- □ Do not search standard system directories for header files
- □ Tell the preprocessor not to generate '#line' directives
- □ Generate profile information code for 'gprof'
- □ Don't delete temporary intermediate files
- □ Attempt to support aspects of traditional C preprocessors
- □ Support ANSI C trigraphs

User defined options:

```
-fomit-frame-pointer -fno-builtin -nostartfiles -Wa,-renesas
```

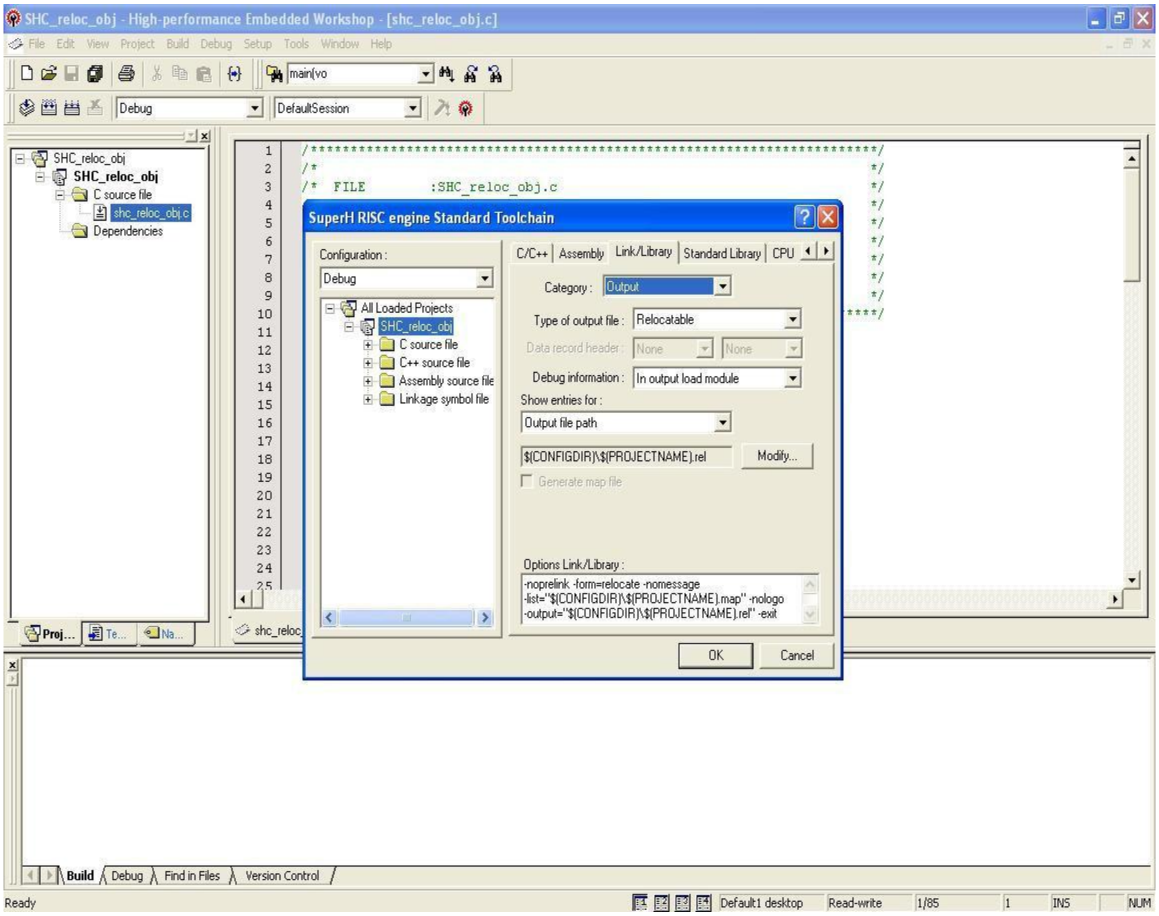
Common options:

```
nclude" -DDEBUG -o "$(CONFIGDIR)\$(FILELEAF).o" -g
-g2 -m2 -mb -mrenesas -fomit-frame-pointer -fno-builtin
-nostartfiles -Wa,-renesas
```

☑ Show option descriptions

Step 5

To generate Renesas relocatable using Renesas SHC HEW project, the _Type of output file‘ option should be _Relocatable‘ in _Link/Library‘ option as shown below;



```
/***************************************************************************/
/*                                                                         */
/*  FILE        :SHC_reloc_obj.c                                           */
```

Category : Output

Type of output file : Relocatable

Data record header : None None

Debug information : In output load module

Show entries for : Output file path

$(CONFIGDIR)\$(PROJECTNAME).rel

□ Generate map file

Options Link/Library :

```
-noprelink -form=relocate -nomessage
-list="$(CONFIGDIR)\$(PROJECTNAME).map" -nologo
-output="$(CONFIGDIR)\$(PROJECTNAME).rel" -exit
```

3.4.2.2 HEW Settings for linking GNUSH relocatable With Renesas Linker

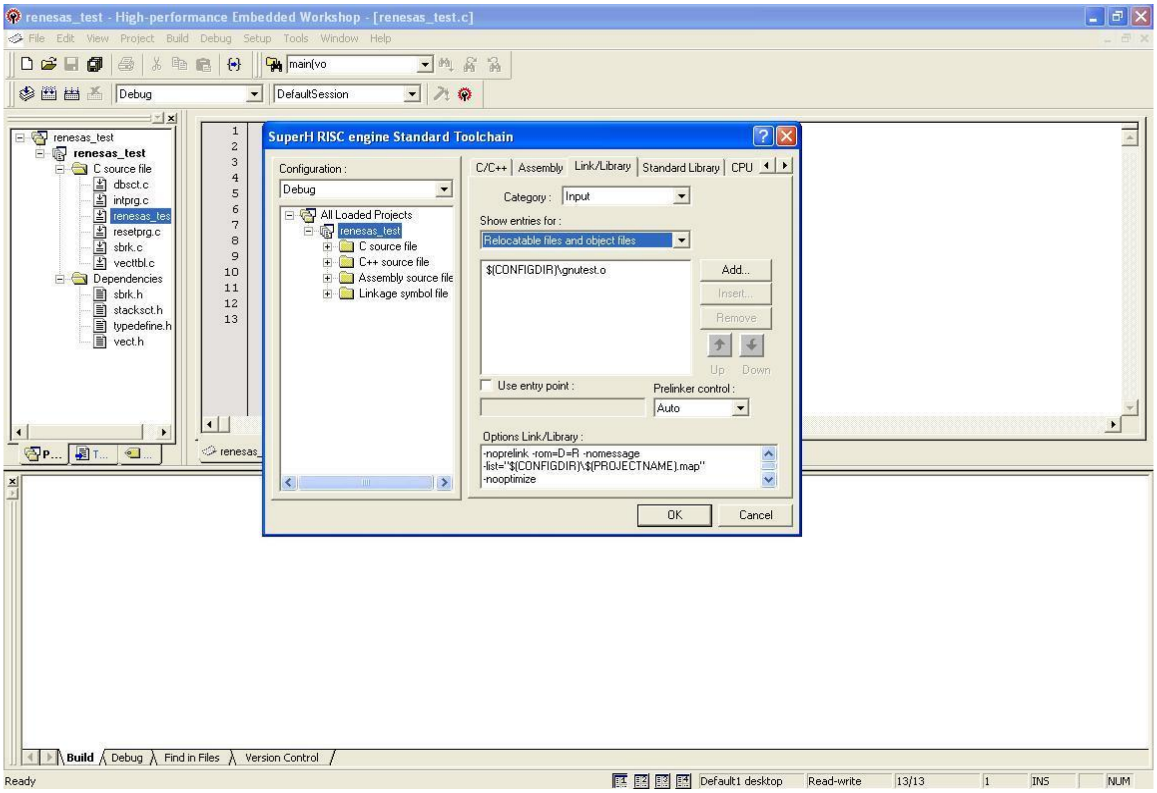
The GNUSH-ELF relocatable is generated using the following command line options:

```
#sh-elf-gcc gnutest.c -m2 -mrenesas -fomit-frame-pointer -
fno-builtin -nostartfiles -Wl,-r -o gnutest.o -lm
```

The following settings need to be done in Renesas HEW project for linking a GNUSH-ELF relocatable with Renesas linker,

Step 6

The GNUSH relocatable name can be specified in _Link/Library‘ option as shown in below;



Configuration : Debug

Category : Input

Show entries for : Relocatable files and object files

$(CONFIGDIR)\gnutest.o

□ Use entry point :

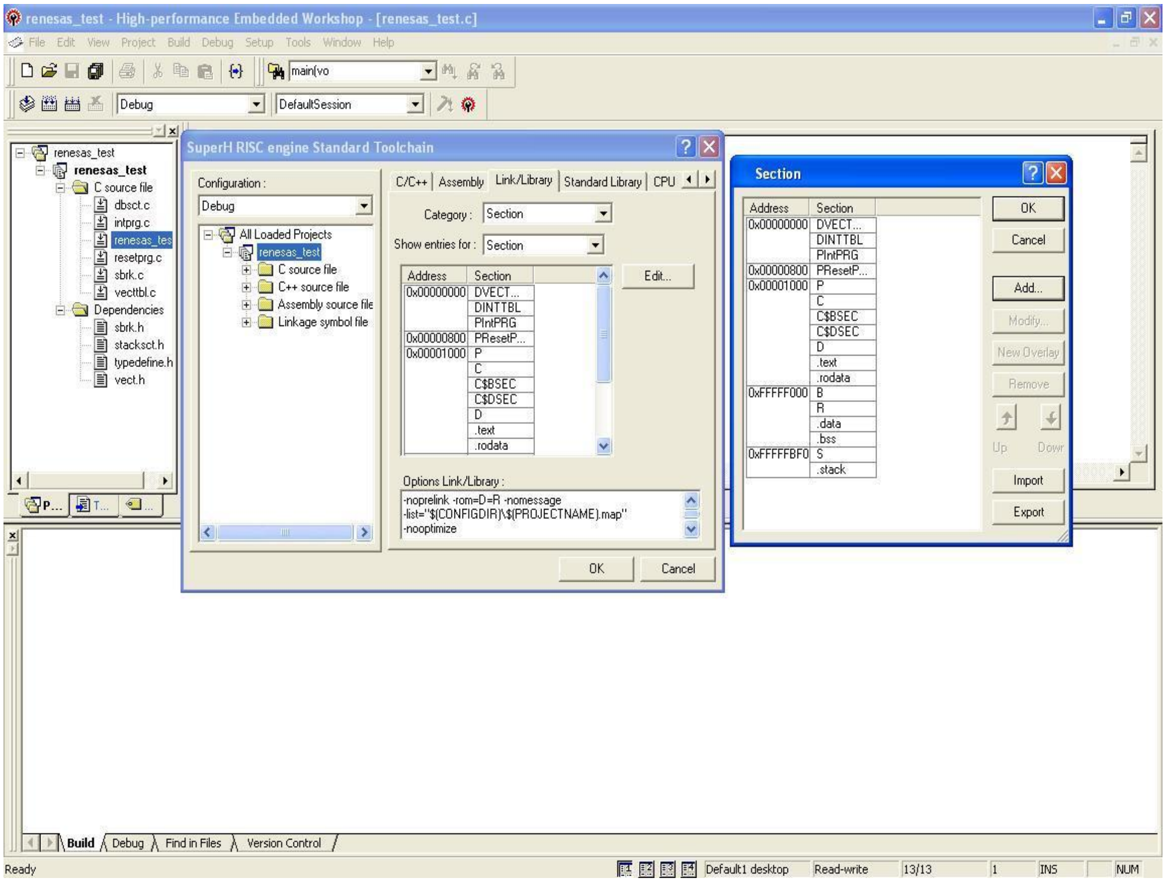
Prelinker control : Auto

Options Link/Library :

```
-noprelink -rom=D=R -nomessage
-list="$(CONFIGDIR)\$(PROJECTNAME).map"
-nooptimize
```

Step 7

The linker sections required by the GNUSH relocatable like .text, .rodata, .data, .bss and .stack have to be added in the Renesas linker as shown below;



Configuration : Debug

Category : Section

Show entries for : Section

Address	Section
0x00000000	DVECT...
	DINTTBL
	PIntPRG
0x00000800	PResetP...
0x00001000	P
	C
	C$BSEC
	C$DSEC
	D
	.text
	.rodata
0xFFFFF000	B
	R
	.data
	.bss
0xFFFFFBF0	S
	.stack

Options Link/Library :

```
-noprelink -rom=D=R -nomessage
-list="$(CONFIGDIR)\$(PROJECTNAME).map"
-nooptimize
```

3.4.3 Binary utility ‗convrenesaslib‘

KPIT has developed a binary utility ‗convrenesaslib‘ which converts the input Renesas SH library file(s) into a GNU archive. The converted GNU archive has *'.rlib'* extension. This utility supports only Renesas SH library files. Multiple Renesas library files can be provided on command line at a time.

This utility extracts the object files from the input Renesas library and appends the filenames with '.robj' extension. The 'ar' utility is then invoked to create a GNU archive of the extracted object files with '.robj' extension. The GNU archive created will have '.rlib' extension appended to the name. These new extensions are necessary so as to distinguish from other files with same names. The extracted object files are deleted after the creation of the GNU archive.

3.4.3.1 Using ‗convrenesaslib‘ on Command line

```
convrenesaslib [`--remove-
             underscore'] [`--help']
             [`--verbose'] [`--
             version'] renesas-
             library-file(s)...
```

renesas-library-file(s) are the Renesas library files to be converted to GNU archive files.

The options which can be used with convrenesaslib are listed below. The long and short forms of options, shown here as alternatives, are equivalent.

`--remove-underscore`

This will remove all leading underscores appended to global symbols in the created GNU archive.

`--help`

Show a summary of the options to convrenesaslib and exit.

`--verbose`

Verbosely list the files processed.

`--version`

Show the version number of convrenesaslib and exit.

3.4.4 Interoperability enhancements in v0702 GNUSH-ELF toolchain

Following were the interoperability related bugs present in earlier toolchain:

a. For SH2A the following error is observed while linking Renesas relocatable with GNUSH linker:

```
shc_reloc_obj.rel: uses dsp instructions while
previous modules use floating point instructions.
shc_reloc_obj.rel: uses instructions which are
incompatible with instructions used in previous
modules.
```

b. Relocatable files generated with GNUSH compiler but not having references to members in ".rodata" section are successfully linked with Renesas SH compiler.
This is because GAS does not generate correct "relocation addend field" relative to sections for SH target. So, the Renesas SH linker does not resolve the references to those sections correctly. Please refer to following link for mailing list discussion on this issue,
http://www.cygwin.com/ml/binutils/2003-08/msg00416.html

c. Also, the GNUSH compiler uses floating point registers for SH2E and SH2A targets for ‘double‘ data type whereas the Renesas compiler uses normal registers. So, interoperability for any computations involving ‗float‘ and ‗double‘ data types was not possible for these two targets at present.

In v0702 GNUSH toolchain, KPIT has applied a patch for the following:

a. Generate correct "relocation addend field" relative to sections for SH target (refer to section 3). This leads to the Renesas SH linker resolving the references to those sections correctly.

b. Resolve addend information generated by the Renesas toolchain.

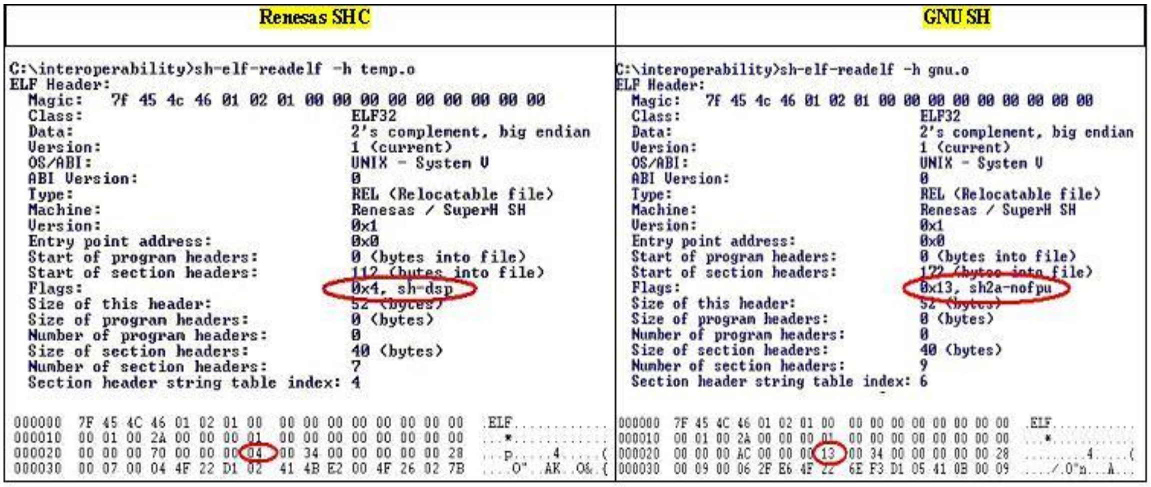
c. The binary utility ‗convrenesaslib‘ utility has been modified for resolving the linking error for SH2A object files.
The following error while linking Renesas relocatable with GNUSH linker for SH2A targets:

```
shc_reloc_obj.rel: uses dsp instructions while
previous modules use floating point instructions
shc_reloc_obj.rel: uses instructions which are
incompatible with instructions used in previous
modules.
```

This error occurs due to the mismatch in target flags.

In object files generated by SHC, the 'e_flag' field in the elf header is set to 0x4 (this value is defined as 'sh-dsp' in case of GNU) for SH2A-noFPU target and 0x5 (this value is defined as sh3-dsp in case of GNU) for SH2A-fpu targets in Renesas SHC compiler.

SHC	Target	GNUSH
0x4	SH2A (No FPU)	0x13
0x5	SH2A-FPU	0xd



Renesas SHC

```
C:\interoperability>sh-elf-readelf -h temp.o
ELF Header:
  Magic:   7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, big endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Renesas / SuperH SH
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          112 (bytes into file)
  Flags:                             0x4, sh-dsp
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         7
  Section header string table index: 4

000000  7F 45 4C 46 01 02 01 00  00 00 00 00 00 00 00 00   .ELF............
000010  00 01 00 2A 00 00 00 01  00 00 00 00 00 00 00 00   ...*............
000020  00 00 00 70 00 00 00 04  00 34 00 00 00 00 00 28   ...p.....4.....(
000030  00 07 00 04 4F 22 D1 02  41 4B E2 00 4F 26 02 7B   ....O"..AK..O&.{
```

GNU SH

```
C:\interoperability>sh-elf-readelf -h gnu.o
ELF Header:
  Magic:   7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, big endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Renesas / SuperH SH
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          172 (bytes into file)
  Flags:                             0x13, sh2a-nofpu
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         9
  Section header string table index: 6

000000  7F 45 4C 46 01 02 01 00  00 00 00 00 00 00 00 00   .ELF............
000010  00 01 00 2A 00 00 00 01  00 00 00 00 00 00 00 00   ...*............
000020  00 00 00 AC 00 00 00 13  00 34 00 00 00 00 00 28   .........4.....(
000030  00 09 00 06 2F E6 4F 22  6E F3 D1 05 41 0B 00 09   ..../.O"n...A...
```

The ‗convrenesaslib' utility has been modified to match the 'e_flags' (processor specific flag) field value in ELF header of SHC library file to that expected by GNUSH. With the modified ‗convrenesaslib', SHC and GNUSH objects can be successfully linked without any errors with GNUSH linker. The modified ―convrenesaslib‖ utility is provided along with GNUSH-v0702 toolchain release.

d. The gcc option ‗-mrenesas' internally enables the assembler option ‗-renesas'. This assembler option disables optimization with section symbol for compatibility with Renesas assembler. In v0602, this option has to be passed manually as it is not enabled automatically with ‗-mrenesas'.
   In v0702 toolchain, this option is automatically enabled on passing ‗-mrenesas' option and hence is not required to be passed.

e. The gcc option ‗-mrenesas' also internally enables a new linker option ‗-renesas' which resolve RELA type relocations, for compatibility with Renesas SHC.

3.4.5 Interoperability Test Results

The following targets with standard libraries (Math, Stdlib, Ctype and String) have been tested for interoperability,

Big Endian	Little Endian
SH1 (-m1)	
SH2 (-m2)	SH2 (-m2)
SH2E (-m2e)	
SH2A (-m2a)	
SH2A-NoFPU (-m2a-nofpu)	
SH2A-Single (-m2a-single)	
SH2A-Single Only (-m2a-single-only)	
SH3 (-m3)	SH3 (-m3)
SH4 (-m4)	SH4 (-m4)
SH4 Single (-m4-single)	SH4 Single (-m4-single)
SH4 Single Only (-m4-single-only)	SH4 Single Only (-m4-single-only)
SH4a (-m4a)	SH4a (-m4a)
SH4a Single (-m4a-single)	SH4a Single (-m4a-single)
SH4a Single Only (-m4a-single-only)	SH4a Single Only (-m4a-single-only)

3.4.5.1 Known problems in ANSI C library functions

`Renesas relocatable with GNUSH linker`

- Standard libraries were tested for interoperability. Most of the library functions passed with the exception of those mentioned below,

  strtod, atof, div, ldiv, ldexp, modf

`GNUSH relocatable with SHC linker`

- Standard libraries were tested for interoperability. Most of the library functions passed with the exception of those mentioned below,

  atof, strtod, div, ldiv, ldexp, frexp

- All the DOUBLE Precision math functions fail for target SH2E for a SHC relocatable with library.

SH2A target errors in ANSI C functions: Interoperability with GNUSH linker

- Some unexpected failures were observed for SH2A targets for the following ANSI C functions: cosh, tanh, frexp
- sh2a-single: Double precision functions are not working with _mrenesas‘ option in the toolchain itself. So interoperability is expected to fail. This problem needs to be fixed in the toolchain.
- sh2a: Single precision functions are not working with mrenesas option in the toolchain itself. So interoperability is expected to fail. This problem needs to be fixed in the toolchain.

SH2A target errors in ANSI C functions: Interoperability with SHC linker

- Some unexpected failures were observed for SH2A targets for the following ANSI C functions: exp, log, log10, modf. The errors _Memory access error‘ and _address error‘ are observed for the same.
- sh2a-single: Double precision functions are not working with _mrenesas‘ option in the toolchain itself. So interoperability is expected to fail. This problem needs to be fixed in the toolchain
- Double precision functions give Address error: For SH2A target, for relocatable containing library, the functions give 'Address error' in HEW.

3.4.5.2 Extended instructions in Renesas SHC

The Renesas SHC specific extended instruction is not supported in GNUSH. This can be better explained with the help of an example.

/*** 1.src ***/
    .GLOBAL _cal
A: .EQU   4
      .SECTION   CODE
_cal:
        mov.l   # cal+A,r0
        RTS
        .END
	/*** 2.src ***/
	    .GLOBAL _cal
	      .SECTION CODE
	
	_cal:
	        mov.l # cal+4,r0
	        RTS
	        .END

Command: create object file using SHC $asmsh 1.src -OB=1.obj	Command :create object file using SHC $asmsh 2.src -OB=2.obj

Command: view opcode of the object file
 	
$sh-elf-objdump -D 1.obj
 	
1.obj: file format elf32-sh
 	
Disassembly of section CODE:
 	
00000000 < cal>:
    0:d001   mov.l   8
<_cal+0x8>,r0 ! 0 <_cal>
    2:000b   rts
    4:a002   bra   c <A+0x9>
    6:0009   nop
    8:0000   .word 0x0000
        ...
	Command: view opcode of the object file
	
	$sh-elf-objdump -D 2.obj
	
	2.obj: file format elf32-sh
	
	Disassembly of section CODE:
	
	00000000 < cal>:
	    0:d0 01 mov.l 8
	<_cal+0x8>,r0 ! 0 <_cal>
	    2: 00 0b rts
	    4: a0 02 bra c <_cal+0xc>
	    6:00 09 nop
	    8:00 00 .word 0x0000
	        ...

Command: view relocations in the object file *$sh-elf-readelf -r 1.obj* Relocation section '.relaCODE' at offset 0xd0 contains 4 entries: Offset Info Type Sym.Value Sym. Name + Addend 00000008 00000480 unrecognized: 80 00000000 cal + 0 00000008 00000380 unrecognized: 80 00000004 A + 0 00000008 00000082 unrecognized: 82 00000000 00000008 00000041 unrecognized: 41 00000000	Command: view relocations in the object file *$sh-elf-readelf -r 2.obj* Relocation section '.relaCODE' at offset 0xbc contains 1 entries: Offset Info Type Sym.Value Sym. Name + Addend 00000008 00000301 R SH DIR32 00000000 cal + 4

As shown above, the opcodes generated for both the test cases (‗1.src' and ‗2.src') are same, but the relocations generated are different.

In ‗1.obj', the **relocations** generated are **unrecognized** by GNUSH. These relocations are not implemented in GNU assembler. In ‗2.obj', the **relocations** generated are of the type **‗R_SH_DIR32'**.

So, when the above object files are linked with ‗gnu.c‘ using GNUSH linker, the linker error **‗final link failed: Bad value‘** is observed in case of ‗1.obj‘.

```
/*** gnu.c ***/

extern cal();
int main()
{
       cal();
       return 0; }
```

Command:	Command:
$ sh-elf-gcc 1.obj gnu.c -mrenesas	*sh-elf-gcc 2.obj gnu.c -mrenesas*
/usr/share/gnush_v0603_elf-1/lib/gcc/sh-elf/4.2-GNUSH_v0603/../../../../sh-elf/bin/ld: final link failed: Bad value collect2: ld returned 1 exit status	‛a.out’ is created

3.4.6 Interoperability and C++

Interoperability testing was carried for C++ functionalities. C++ code debugging capability was tested for interoperability. However this gave linking errors due to mangling done for C++ code in GNUSH as well as SHC compiler. This error was not observed for custom functions in C++ file using "extern 'C'" and so they were successfully tested and found to be working for interoperability. However for C++ specific functionalities (like methods within 'class'), cross-linking gives linking errors due to mangled names. Due to this behaviour, C++ libraries cannot be cross-linked.

Name mangling is a technique used to solve various problems caused by the need to resolve unique names for programming entities in many modern programming languages.

The need arises where the language allows different entities to be named with the same identifier as long as they occupy a different namespace (where a namespace is typically defined by a module, class, or explicit namespace directive).

Name mangling in C++:

The C++ language does not define a standard decoration scheme, so each compiler uses its own. Combined with the fact that C++ decoration can become fairly complex (storing information about classes, templates, namespaces, operator overloading, etc), this means that object code produced by different compilers is not usually linkable.

Standardized name mangling in C++:

While it is a relatively common belief that standardized name mangling in the C++ language would lead to greater interoperability between implementations, this is not really the case. Name mangling is only one of several ABI issues in a C++ implementation, and other language details like exception handling, virtual table layout, structure padding, etc. would render differing implementations yet incompatible. Further, requiring a particular form of mangling would cause issues for systems where implementation limits (e.g. length of symbols) dictate a particular mangling scheme. A standardized requirement for name mangling would also prevent an implementation where mangling was not required at all – for example, a linker which understood the C++ language.

Example: For the code below

```
class Classes
{
   void class_member(void);
}
```

For the above class, **Renesas** SHC generates the symbol:

```
_class_member__7ClassesFv
```

Whereas, **GNUSH** generates the following symbol:

```
      __ZN7Classes12class_memberEv
```

The following link also gives some details on ‗C++ and interoperability‖, www.kpitgnutools.com/manuals/gcc.html#SEC143

Renesas compiler and GNU compiler use different types of **mangling** techniques. Hence interoperability between Renesas SHC and GNUSH for C++ is not possible.

3.4.7 Workarounds/Solutions for known errors in ANSI C functions

3.4.7.1 Interoperability Testing: GNUSH relocatable with SHC linker

Linker errors and their workaround are mentioned below;

a. Undefined external symbol ‗_end‘

   The following line has to be added in the source file
   : *unsigned volatile char *end = __secend("B");*
   Also, the SHC linker script in HEW needs to be modified so that the sections in RAM region are placed in the following order: .data, .bss, R, B

b. Undefined external symbol ‗___fpscr_values‘,

   The workaround mentioned in FPSCR-Workaround need to be done.

c. Undefined external symbol ‗__truncdfsf2‘

   The file "truncdfsf2.c" needs to be added in the project. This file contains the definition for function ‗__truncdfsf2‘.

d. Undefined external symbol ‗___ctype_ptr‘

   The file "testctype.c" needs to be added in the project. This file contains the definition of ‗__ctype_ptr‘.

4 Building GNUSH Tools

The build process for GNU tools presumes the existence of a native C compiler and linker, so the best way to get started with GNU tools is to buy and install a GNU/Linux distribution CD. With this approach you get a working Linux environment with a preinstalled native GNU compiler and linker, plus a debugger and other tools. The Cygwin environment is also an option, if you intend to use a Win32 development host.

This section details the procedure for building the GNUSH toolchain using toolchain sources available at KPIT GNU tools website. It explains the building of GNUSH toolchain for Linux (Cross toolchain) as well as Windows (Canadian Cross) Platform.

4.1 How to Download the GNUSH Sources

The toolchain sources for latest two toolchains are always available on KPIT GNU tools website. User needs to download GCC (Compiler Sources), Binutils (Binary Utility Sources) and Newlib (Newlib Sources) for the required version.

Note: In order to download the sources from KPIT GNU site, you need to be a registered user of our website. The registration is free, quick and easy and is only required once. Your details are securely stored by us and will never be sold or given away.

Alternately, these sources can be downloaded from GNU website http://www.gnu.org and a popular C runtime library used in the example script is Newlib, which is available from http://sources.redhat.com/newlib.

This section explains building of GNUSH toolchain for following versions of sources;

- Binutils 2.17.90
- GCC 4.2.1
- Newlib 1.15.0

4.2 Building the GNUSH Tools

To build the GNUSH tools on Linux system, login as user and set up directory structure in $HOME folder (e.g. /home/user), as explained below. There are three top level directories that need to be created before starting the toolchain building.

- <Source Directory> e.g. src
- <Build Directory> e.g. build
- <Prefix Directory> e.g. prefix

- All the sources required for toolchain building should be present in <Source Directory>

- The building of the toolchain is done in <Build Directory>. During the build process, all the intermediate build files are kept in this directory. All the script files required for building the toolchain, the build script, ‗timed_read‘, ‗native-version‘, ‗remove_cygwin_includes‘ and ‗assign_version‘ should be placed in this directory only.
- On successful building of the toolchain, the toolchain get installed in <Prefix Directory>

4.2.1 Building the Linux Toolchain

- Change the directory to $HOME. For Example,

```
#cd /home/user
```

- Create src, build and prefix directories

```
#mkdir src build prefix
```

- Change directory to ‗src‘

```
#cd src
```

- Copy all the down loaded sources in ‗src‘ directory
- Un-tar all the sources in ―src‖ directory

```
#tar –jxvf binutils-2.17.90.tar.bz2
#tar –jxvf gcc-4.2.1.tar.bz2
#tar -zxvf newlib-1.15.0.tar.gz
```

- This should create following folders in ‗src‘ directory

```
#ls –l binutils-
2.17.90 gcc-
4.2.1 newlib-
1.14.0
```

- Change directory to build folder

```
#cd ../build
```

- Create build directories for binutils, gcc and newlib

```
#mkdir binutils gcc newlib
```

- Build Binary utilities. These are then needed for GCC building
- Change directory to binutils.

```
#cd ../binutils
```

- Run the configure script included with the binutils package,

```
 ../../binutils-2.17.90/configure --target=$TARGET
--prefix=$PREFIX #../../binutils-
2.17.90/configure --target=sh-elf
  --prefix=/home/user/prefix
```

- This will create Makefile. Make to build binary utilities. and dump the output of this command to temporary file so as it can be used for debugging the build if any error occurs.

```
#make all > make.out 2>&1
```

- Install the binary utilities in prefix folder and dump the output of this command to temporary file so as it can be used for debugging the build if any error occurs.

  ```
  #make install > makeinstall.out 2>&1
  ```

- Completion of this step without any error means binary utilities building for GNUSH is done successfully.
- Now that we have an assembler and linker, we can build and install the GNUSH C/C++ compiler. The initial step in the procedure yields a **bootstrap compiler** that can only be used to build runtime libraries and header files. We will use this compiler to build the sh-elf version of the newlib C runtime library. Once that‘s done, we will rebuild the compiler completely, including internal libraries that need target-specific header files from newlib in order to be compiled properly.

The commands to build bootstrap compiler are given below,

- Change directory to ‗gcc‘ folder under ‗build‘ folder.

  ```
  #cd ../gcc
  ```

- Ensure that you are in correct build directory,

  ```
  #pwd
  In this case, present working directory shows
  following path,
  /home/user/build/gcc
  ```

- Export the path of newly built binaries

  ```
  #export PATH=$PATH:/home/user/prefix/bin
  ```

- Configure

  ```
  #../../src/gcc-4.2.1/configure --enable-
  languages=c,c++       --target=sh-elf -with-newlib -
  -prefix=/home/user/prefix       --disable-libstdcxx-
  pch
  ```

- This will create Makefile. Make to build bootstrap gcc and dump the output of this command to temporary file so as it can be used for debugging the build if any error occurs.

  ```
  #make LANGUAGES="c c++" all-gcc > make1.out 2>&1
  ```

- Install the built gcc in prefix folder and dump the output of this command to temporary file so as it can be used for debugging the build if any error occurs.

  ```
  #make LANGUAGES="c c++" install-gcc >
  makeinstall1.out 2>&1
  ```

- Completion of this step without any error means bootstrap compiler building for GNUSH is done successfully.

- Done with binary utilities and bootstrap compiler, hence time to build the newlib C runtime library.
- Change directory to _newlib' folder under _build' folder.

```
#cd ../newlib
```

- Ensure that you are in correct build directory,

```
#pwd
```

```
In this case, present working directory shows
following path,
/home/user/build/newlib
```

- Configure newlib

```
#../../src/newlib-1.15.0/configure --enable-
newlib-hw-fp --target=sh-elf --
prefix=/home/user/prefix
```

- This will create Makefile. Make to build newlib libraries and dump the output of this command to temporary file so as it can be used for debugging the build if any error occurs.

```
#make > make.out 2>&1
```

- Install the built libraries in prefix folder and dump the output of this command to temporary file so as it can be used for debugging the build if any error occurs.

```
#make install > makeinstall.out 2>&1
```

- Completion of this step without any error means newlib building is done successfully.

Building a complete cross compiler

As we have headers and libraries from newlib, we can build complete cross compiler Setup for C/C++ development.

- Change directory to _gcc' folder under _build' folder.

```
#cd ../gcc
```

- Ensure that you are in correct build directory,

```
#pwd
```

```
In this case, present working directory shows
following path,
/home/user/build/gcc
```

- Make to build the complete GNUSH cross compiler

```
# make > makelast.out 2>&1

# make proto > makeproto.out 2>&1
```

- Install the built final gcc in prefix folder and dump the output of this command to temporary file so as it can be used for debugging the build if any error occurs.

```
# make install > makeinstalllast.out 2>&1
```

- Completion of this step without any error means, our GNUSH cross compiler is ready to use!!!

4.2.2 Building the Canadian Cross Toolchain (mingw)

Canadian cross method is used for building the compiler running on mingw for target SH. This section explains building of Canadian cross toolchain for

- TARGET = sh-elf
- BUILD = i686-pc-linux-gnu
- HOST = i386-pc-mingw32msvc

The cross compilers are:

1. build-x-host compiler (linux-x-mingw)
2. build-x-target compiler (linux-x-sh)
3. host-x-target compiler (mingw-x-sh) using above 2 compilers

In order to build the ‗mingw' based Canadian Cross Compiler, user need to have Mingw native compiler available at your set-up. The following procedure explains the building of GNUSH ‗mingw' toolchain assuming,

- The native Mingw compiler is already available on users system. (If not available, please visit www.mingw.org for details on how to build Native Mingw Compiler)
- User already has GNUSH Linux toolchain installed on his system. The toolchain is required to build mingw toolchain.

Following procedure explains the building of ‗mingw' toolchain,

- Create a separate folder for mingw building and prefix

```
#cd /home/user

#mkdir    mingw-build    mingw-
prefix #cd mingw-build
```

- Create build directories for mingw build of binutils, gcc and newlib

```
# mkdir  m-binutils  m-gcc  m-newlib
```

- Export PATH for native mingw and GNUSH toolchain for Linux.

```
#export PATH=<PATH For Native Mingw>;<PATH for
GNUSH tools for Linux >;$PATH

#export PATH=/usr/share/mingw-
3.3/bin:/home/user/prefix/bin/:$PATH
```

- Build Mingw Binary utilities.
- Change directory to binutils.

```
#cd ../m-binutils
```

- Run the configure script included with the binutils package

```
../../binutils-2.17.90/configure --build=$BUILD -
- host=$HOST --target=$TARGET --prefix=$PREFIX

#../../binutils-2.17.90/configure --build=i686-pc-
linux-gnu   --host=i386-pc-mingw32msvc --target=sh-
elf                    --prefix=/home/user/mingw-
prefix

# make > make.out 2>&1

# make install > makeinstall.out 2>&1
```

- Completion of these steps without any error means mingw binary utilities building for GNUSH is done successfully.
- Steps to build bootstrap for Mingw GNUSH compiler
- Change directory to ‗m-gcc‘ folder under ‗mingw-build‘ folder.

```
# cd ../m-gcc
```

- Export PATH for newly built binary utilities

```
#export PATH=$PATH:/home/user/mingw-prefix/bin
```

- Configure

```
#../../src/gcc-4.2.1/configure --build=i686-pc-
linux-gnu       --host=i386-pc-mingw32msvc --
target=sh-elf --enable-languages=c --with-newlib
--prefix=/home/user/mingw-prefix > configure.out
2>&1

#make configure-build-libiberty > makec.out 2>&1

#make all-build-libiberty > makeac.out 2>&1

#make all-build-libiberty > makeac.out 2>&1

#cd ./gcc

#sed 's/USE_COLLECT2\ =\
collect2$(exeext)/USE_COLLECT=/' Makefile >
Makefile.out

#mv -f Makefile Makefile.org

#mv -f Makefile.out Makefile

#cd ../

#make LANGUAGES=c all-gcc > make1.out 2>&1

#make LANGUAGES=c install-gcc > makeinstall1.out
2>&1
```

- Completion of this step without any error means bootstrap mingw compiler building for GNUSH is done successfully.

- Done with binary utilities and bootstrap compiler, hence time to build the newlib C runtime library.
- Change directory to _newlib' folder under _build' folder.

```
# cd ../m-newlib
```

- Configure newlib

```
#../../src/newlib-1.15.0/configure --build=i686-
pc-linux-gnu --host=i386-pc-mingw32msvc --
target=sh-elf --prefix=/home/user/mingw-prefix


#cd ../m-newlib #make

> make.out 2>&1

#make install > makeinstall.out 2>&1
```

- Completion of this step without any error means Newlib building is done successfully.
- Building a complete Canadian cross compiler.
- Change directory to _gcc' folder under _build' folder.

```
#cd ../m-gcc

#cd ./gcc

#sed 's/USE_COLLECT2\ =\
collect2$(exeext)/USE_COLLECT=/' Makefile >
Makefile.out

#mv -f Makefile Makefile.org

#mv -f Makefile.out Makefile

#cd ../

#make > makelast.out 2>&1

#make install > makeinstalllast.out 2>&1
```

- Completion of this step without any error means, our GNUSH Canadian cross compiler is ready to use!!!

5 Glossary

ABI	:	Application Binary Interface
BSS	:	Block Started by Symbol (basically section for non initialized global variables)
BFD	:	Binary File Descriptor
COFF	:	Common Object File Format
CPU	:	Central Processing Unit
DWARF2	:	Debugging Information Format
EVB	:	Evaluation Board
EDK	:	Evaluation Development Kit
ELF	:	Executable and Linking Format
FDT	:	Flash Development Tool
FAQ	:	Frequently Asked Questions
FPU	:	Floating Point Unit
GCC	:	GNU C Compiler
GNUSH	:	GNU SH Compiler
GPR	:	General Purpose Register
GDBSH	:	GNU SH Debugger
GDB	:	GNU Debugger
GUI	:	Graphical User Interface
GNU	:	GNU Not Unix
GPL	:	General Purpose License
HEW	:	High Performance Embedded Workshop
HMON	:	Monitor Program for Debugging
ISO	:	International Organization for Standardization
ISS	:	Instruction Set Simulator
IDE	:	Integrated Development Environment
LMA	:	Load Memory Address
OS	:	Operating System
PIC	:	Position Independent Code
RTOS	:	Real Time Operating System
ROM	:	Read Only Memory
RAM	:	Read Access Memory

RSK	:	Renesas Starter Kit
SREC	:	S-REC File Format
SHC	:	Renesas SH Compiler
SH	:	SuperH target from Renesas
VMA	:	Virtual Memory Address
as	:	GNU Assembler
convrenesaslib	:	Binary Utility to convert Renesas library to GNU SH format
g++	:	GNU C++ Compiler executable
ld	:	GNU Linker
objdump	:	Binary Utility for viewing object dump of binary
readelf	:	Binary Utility for viewing the ELF/DWARF information in binary

6 Important Links

http://www.eu.renesas.com/

http://gcc.gnu.org/onlinedocs/gcc-3.2.2/gcc/Function-Attributes.html#Function%20Attributes

http://sources.redhat.com/binutils/docs-2.15/

http://sources.redhat.com/binutils/docs-2.15/ld/index.html

http://htmlhelp.berlios.de/books/chm.php

http://www.objsw.com/CrossGCC/

http://docs.freebsd.org/info/gcc/gcc.info.Environment_Variables.html

http://gcc.gnu.org/onlinedocs/

http://gnu.essentkabel.com/software/gcc/readings.html

http://sources.redhat.com/newlib/libc.html

http://sources.redhat.com/newlib/libm.html

7 Appendix – A (Interoperability between Renesas SHC and GNUSH)

FPSCR-Workaround

```
For the following Linker error: Undefined external
symbol "___fpscr_values",

The following code has to be added as .src file
which defines the same.


For double precision targets like m4, m4a, etc:

     .section    P, code

     .EXPORT ___fpscr_values


     .ALIGN    4


___fpscr_values:


     .DATA.L   H'00040000          ; Single Precision

     .DATA.L   H'000C0000          ; Double Precision


     .END
```

TRUNCDFSF2.c

```
/* the following deal with IEEE single-precision
numbers */
#define EXCESS      126L
#define SIGNBIT          0x80000000L
#define HIDDEN      (1L << 23L)
#define SIGN(fp)    ((fp) & SIGNBIT)
#define EXP(fp)          (((fp) >> 23L) & 0xFF)
#define MANT(fp)    (((fp) & 0x7FFFFFL) | HIDDEN)
#define PACK(s,e,m) ((s) | ((e) << 23L) | (m))


/* the following deal with IEEE double-precision
numbers */
#define EXCESSD          1022
#define HIDDEND          (1L << 20L)
#define EXPDBITS    11
#define EXPDMASK    0x7FF
#define EXPD(fp)    (((fp.l.upper) >> 20L) & 0x7FFL)
#define SIGND(fp)   ((fp.l.upper) & SIGNBIT)
#define MANTD(fp)   (((((fp.l.upper) & 0xFFFFF) |
HIDDEND) << 10) | \
                    (fp.l.lower >> 22))
#define MANTDMASK   0xFFFFF /* mask of upper part */


/* the following deal with IEEE extended-precision
numbers */
#define EXCESSX          16382
#define HIDDENX          (1L << 31L)
#define EXPXBITS    15
#define EXPXMASK    0x7FFF
#define EXPX(fp)    (((fp.l.upper) >> 16) & EXPXMASK)
#define SIGNX(fp)   ((fp.l.upper) & SIGNBIT)
#define MANTXMASK   0x7FFFFFFF /* mask of upper part */
```

TESTCTYPE.c

```
/* the following deal with IEEE single-precision numbers */
#define EXCESS          126L
#define SIGNBIT   0x80000000L
#define HIDDEN          (1L << 23L)
#define SIGN(fp)  ((fp) & SIGNBIT)
#define EXP(fp)   (((fp) >> 23L) & 0xFF)
#define MANT(fp)  (((fp) & 0x7FFFFFL) | HIDDEN)
#define PACK(s,e,m)     ((s) | ((e) << 23L) | (m))
/* the following deal with IEEE double-precision numbers */
#define EXCESSD         1022
#define HIDDEND         (1L << 20L)
#define EXPDBITS  11
#define EXPDMASK  0x7FF
#define EXPD(fp)  (((fp.l.upper) >> 20L) & 0x7FFL)
#define SIGND(fp) ((fp.l.upper) & SIGNBIT)
#define MANTD(fp) (((((fp.l.upper) & 0xFFFFF) | HIDDEND) << 10) |
(fp.l.lower >> 22))
#define MANTDMASK 0xFFFFF /* mask of upper part */
/* the following deal with IEEE extended-precision numbers */
#define EXCESSX         16382
#define HIDDENX         (1L << 31L)
#define EXPXBITS  15
#define EXPXMASK  0x7FFF
#define EXPX(fp)  (((fp.l.upper) >> 16) & EXPXMASK)
#define SIGNX(fp) ((fp.l.upper) & SIGNBIT)
#define MANTXMASK 0x7FFFFFFF /* mask of upper part */
union double_long
{
  double d;
  struct {
      long upper;
      unsigned long
    lower; } l;
};
```

```
#define _CTYPE_DATA_128_256 \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0, \
     0,   0,   0,   0,   0,   0,   0,  0
```

```
static const char _ctype_b[128 + 256] =
     { _CTYPE_DATA_128_256,
     _CTYPE_DATA_0_127,
     _CTYPE_DATA_128_256
};

const char *__ctype_ptr = _ctype_b + 128;

const char _ctype_[1 + 256] =
     { 0,
     _CTYPE_DATA_0_127,
     _CTYPE_DATA_128_256
};

//const char *__ctype_ptr = _ctype_ + 1;
```

Revision History

Version Number	Date	Overview of Changes
5.4	7th June 2010	Updated as per v10.02 Release
5.3	18th February 2010	Updated as per v10.01 Release
5.2	30th September 2009	Updated sections 2.5.2.1, 2.5.2.2 and 3.1.1.
5.1	2nd June 2009	1. Updated section 2.1.1 Install and setup procedure 2. Updated section 2.7.2 Creating and Building a Sample Project Using HEW
5.0	5th February 2009	Updated for website release
4.0	2nd February 2009	Released with GNUSH v0901 installer and RPM
3.1	30th January 2009	1. Added details for Unifile utility (Section 2.5.1.5 and 2.5.5) 2. Modified the migration guide (Section 3.1) for GCC 4.3.2 changes (extern keyword for inline)
3.0	1st October, 2008	Released
2.1	30th September, 2008	1. Modified section 3.1.7 for the ‗-h-tick-hex‘ option to support ―H‘‖ hexadecimal data representation 2. Added Vista support
2.0	30th May, 2008	1. Corrected the sample program in section 2.6.4. 2. Added the libgen part at section 2.5.5.3
1.1	7th March, 2008	1. Modified Section 3.1.1 to add ‗Using Inline Assembly Language in GNUSH‘. 2. Corrected Section 2.6.1 for GNUSH syntax
1.0	4th October, 2007	Draft Created