Freescale Semiconductor
Application Note

Document Number: AN4253
Rev. 1, 01/2011

Converting Earlier Versions of CodeWarrior for StarCore DSPs Projects to Version 10.1.8

by *DevTech Customer Engineering*
Freescale Semiconductor, Inc.
Austin, TX

This application note describes an important change in the behavior of the Eclipse-based CodeWarrior for StarCore DSPs development tools starting in release 10.1.8.

In earlier tool releases, the system information was stored in each launch configuration. This made development of multicore DSP applications tricky because separate launch configurations are used to manage each core. The redundant copies of system information in each configuration were difficult to manage and synchronize. With CodeWarrior v10.1.8, the system information is consolidated into a new Remote System Explorer (RSE) framework. This framework allows system information to be stored in a repository and applied across multiple configurations.

However, because of this change, projects made with the CodeWarrior tools prior v10.1.8 lack the RSE framework. These projects require special steps to convert them for use with the 10.1.8 version of the tools. This application note explains this conversion process.

Contents

1 Overview

One useful feature of the CodeWarrior for StarCore DSPs IDE is its ability to store and manage much of the settings information associated with building and debugging a DSP application. This allows the developer to enter information once through the CodeWarrior GUI, and then focus on writing code while the IDE handles the settings and communications details for test and debug sessions. The IDE groups related settings into logical containers known as *configurations*. The IDE has the following configurations:

- Launch configuration—These settings specify how the IDE launches the application being developed. Such settings describe the execution target that the application runs on, which might be a hardware board, an emulator, or an instruction set simulator (ISS). Launch configurations, as their name implies, set up the runtime environment of the target before starting the application in either run mode or debug mode.
- System configuration—These settings describe how the IDE communicates with the execution target, such as through a hardware JTAG probe, or via a USB or Ethernet connection. It can also specify information about an execution target's hardware, such as its processor type and memory map.

NOTE

The configuration most apparent to the user is the launch configuration, since it is used to start the application, either as stand-alone or under debug control. The system configuration information did not appear as a user-selectable object until CodeWarrior version 10.1.8.

In earlier releases of CodeWarrior for StarCore DSPs, what is now known as system configuration settings were stored with the launch configuration settings and in other locations in the project. For single core applications this arrangement served the purpose. However, for complex multicore applications this scheme began to complicate matters. Specifically, the CodeWarrior tools manage a multicore application by assigning a launch configuration to each processor core. That is, a CodeWarrior project for the six-core MSC8156 processor has six launch configurations. Attaching a launch configuration to each core permits fine-grained control over the entire processor, and allows a sophisticated DSP application to be partitioned into virtual subsystems that use one or more processor cores. However, this arrangement requires the developer to duplicate and save identical system information into each launch configuration. If a change is made to the system configuration, synchronizing the information among all of the launch configurations was both tedious and error-prone.

With CodeWarrior for StarCore DSPs v10.1.8, the system configuration information has been separated from the launch configurations and consolidated into its own group (Figure 1). The RSE framework stores the system configuration data into a repository and manages it. The advantage to this scheme that the system settings are automatically shared among those multiple launch configurations that request the data. Furthermore, a change to the system configuration is conveyed to the affected launch

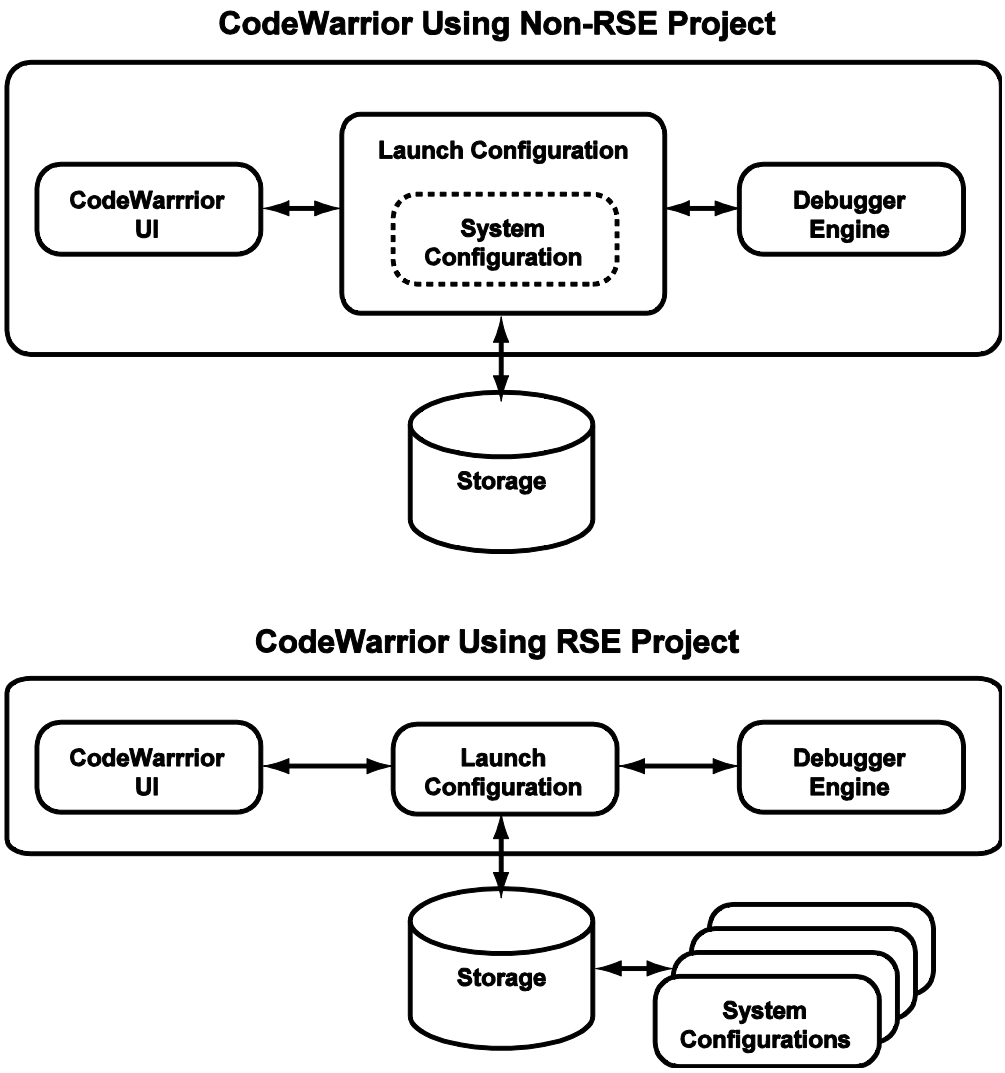

Figure 1. Differences in How System Configurations Are Stored in a Project.

configurations. This eliminates having to maintain duplicate copies of the system settings in those launch configurations that require it.

NOTE

To view the RSE system configuration for a particular project, choose **Window > Show View > Remote Systems**, right-click on the desired configuration, then choose **Properties**.

This new scheme improves the IDE's support of the development of multicore applications. However, this change also breaks compatibility with projects made with earlier versions of the CodeWarrior tools. Importing an older project that lacks the RSE framework into v10.1.8 causes problems because the system configuration data is not located in the repository. The rest of this application note covers the steps to follow to convert a non-RSE project to RSE-based project. This procedure is necessary when upgrading CodeWarrior for StarCore DSPs from v10.1.3 or v10.1.5 to v10.1.8. Those already using the CodeWarrior for StarCore DSPs v10.1.8 toolset do not need to perform these steps.

2 Importing a Non-RSE Project

To import and convert a non-RSE CodeWarrior project into a RSE-compliant v10.1.8 project, proceed as follows:

1. From the menu, select **File > Import**.

The **Import Window** appears, and displays the Select page.

2. Expand the **General** item, and select **Existing Projects into Workspace** (Figure 2).

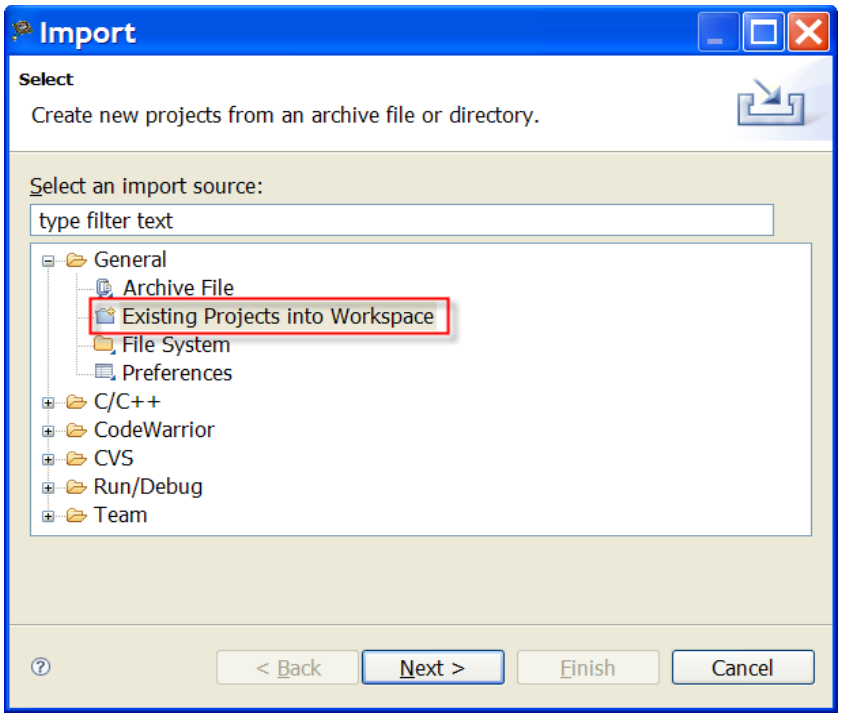

Figure 2. Choosing the Existing Projects Import Option.

3. Click **Next**.

The **Import Projects** page appears (Figure 3).

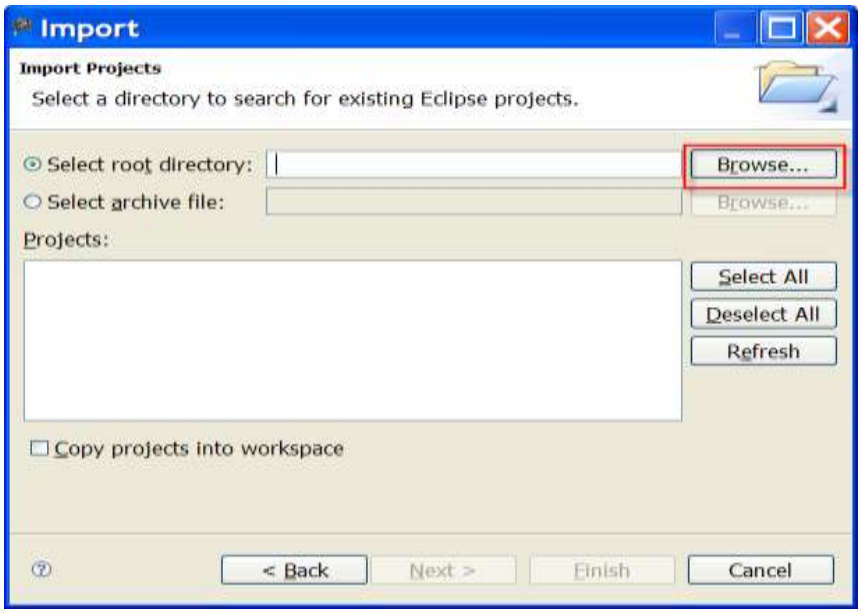

Figure 3. Selecting the Directory to Search.

4. For the **Select Root Directory** option, click **Browse**.

The **Browse for Folder** dialog appears.

5. In the Browse for Folder dialog, select the project existing project to import. Click **OK**.

The directory of the project being imported appears in the **Select Root Directory** option. The name of the project appears in the **Projects** option (Figure 4).

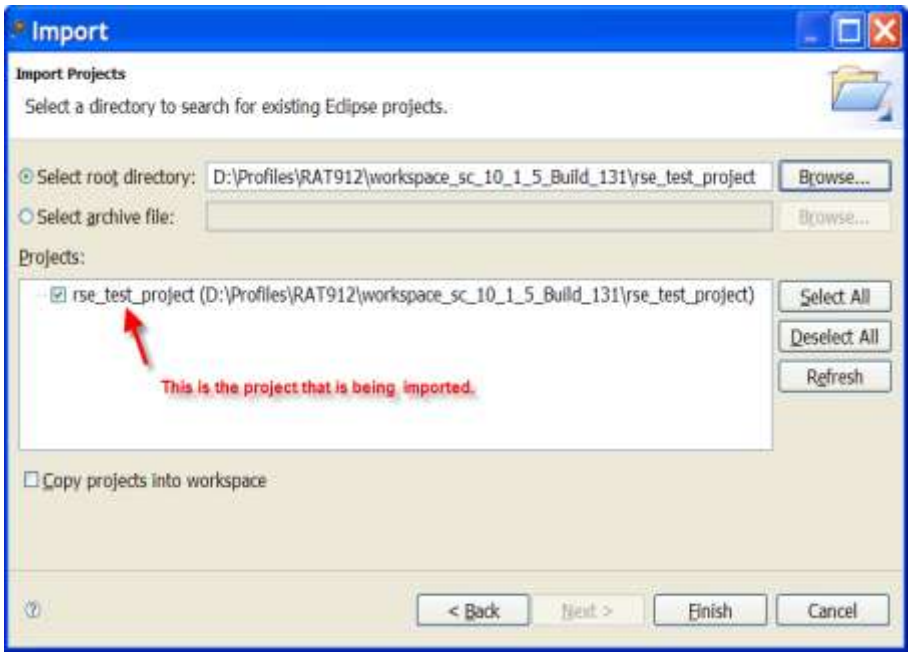

Figure 4. The Page Displaying the Project to Import.

6. Click **Finish**.

The **Launch Configuration Migration to RSE Required** alert appears (Figure 5). It notifies the user when a new set of launch configurations need to be migrated.

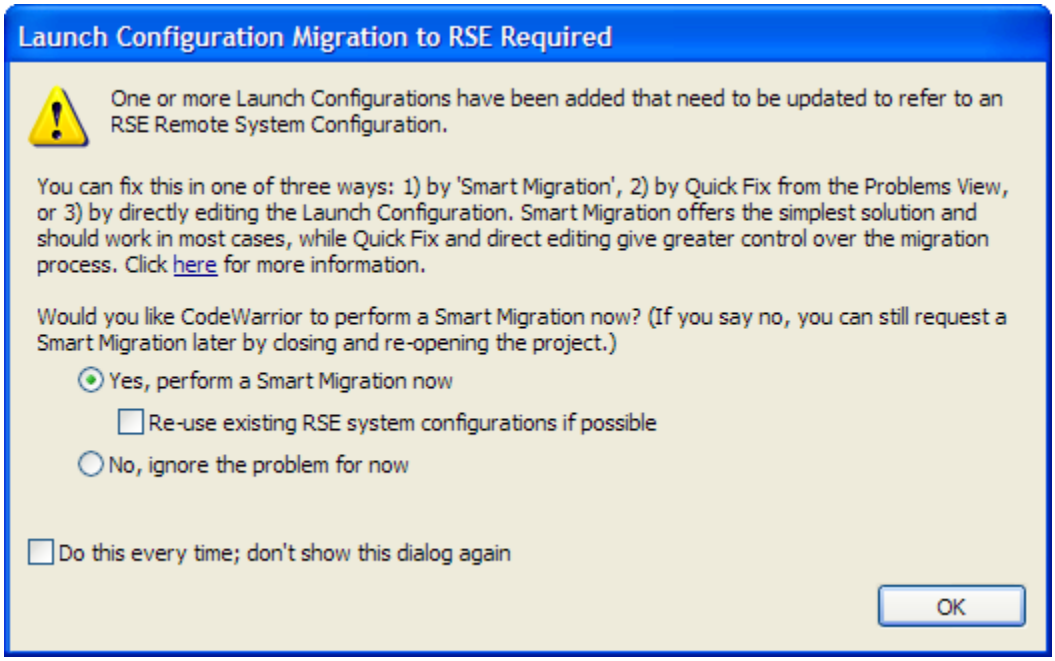

Figure 5. The Migration Alert.

Table 1 summarizes the purpose for each of the dialog options. For this example, a Smart Migration is not performed and the Quick Fix feature is used instead to handle the migration. For more information on the other options and how to use them, consult the *Freescale Eclipse Extensions Guide*.

Table 1. Options for the Launch Configuration Migration to RSE Required Dialog

Option	Description
Yes, perform a Smart Migration now	Uses the Smart Migration logic to import a project with or without RSE information.
Re-use existing RSE system configurations if possible	Use any existing RSE information that might be available to construct the RSE information. Otherwise, extract the information from the launch configuration.
No, ignore the problem for now	Do not perform the migration for now. The user can manually start the migration by using Quick Fix feature from the Problem view.
Do this every time	Saves the choices to be used on all subsequent non-RSE project imports.

7. Click **No**. This allows the user to go directly to the **Problems View** window.

The new imported project appears under the **CodeWarrior Projects** view (Figure 6). Markers appear next to project folder, and as corresponding items in the **Problems** view.

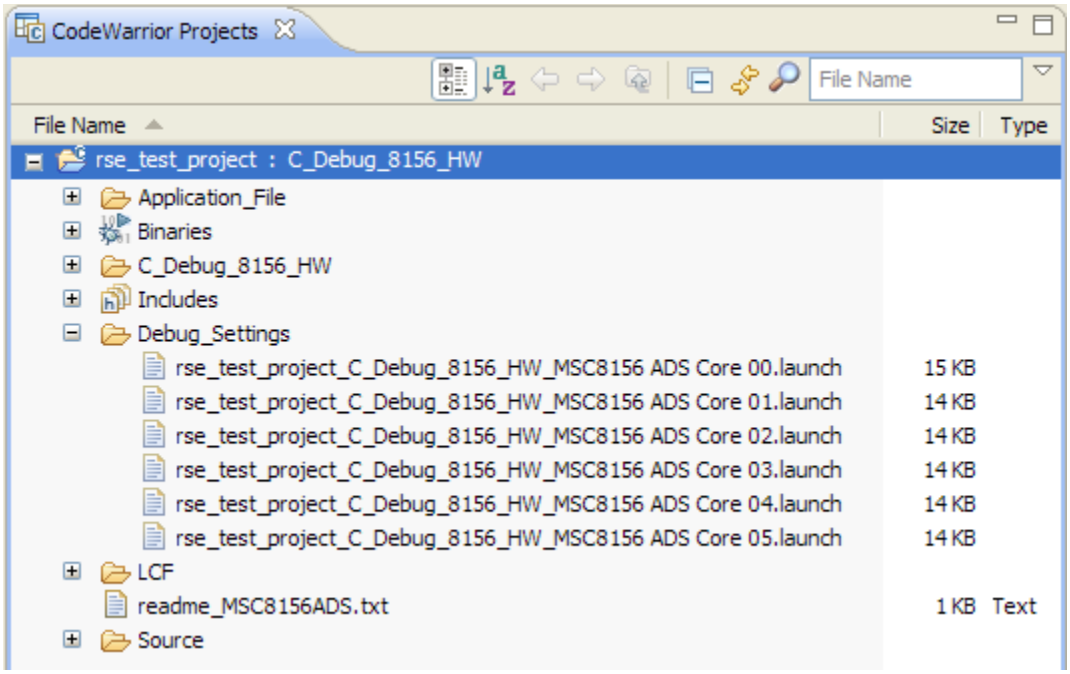

Figure 6. The Imported Project Display.

8. In the **Problems** view, expand the **Warnings** item (Figure 7). Warning messages are displayed for each of the configurations that are migration candidates for a specific target board.

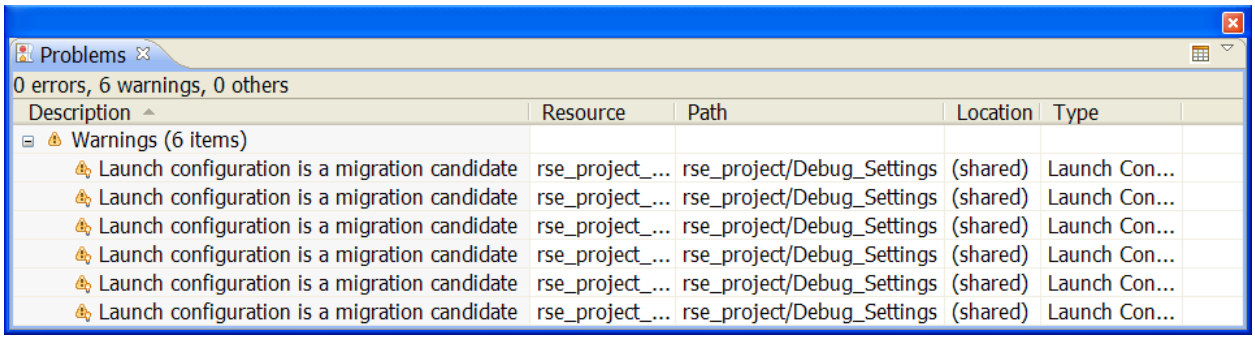

Figure 7. The Migration Candidates Displayed in the Problems View.

9. Right-click on a launch configuration in the **Problems** view.

A context menu appears (Figure 8).

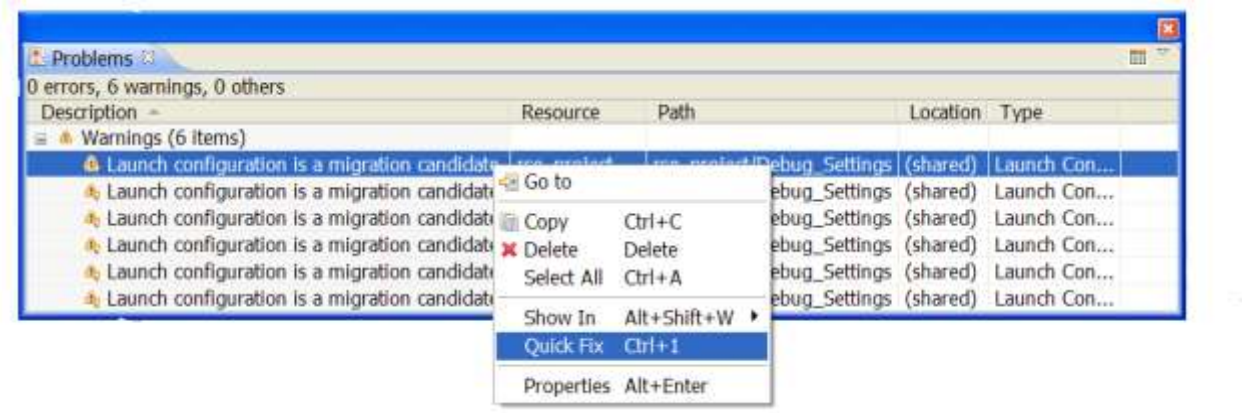

Figure 8. Applying a Quick Fix to the Chosen Launch Configuration.

10. From context menu, select **Quick Fix**. Selecting **Quick Fix** for any candidate starts the migration process.

A **Quick Fix** dialog appears (Figure 9) and displays all of the possible migration candidates.

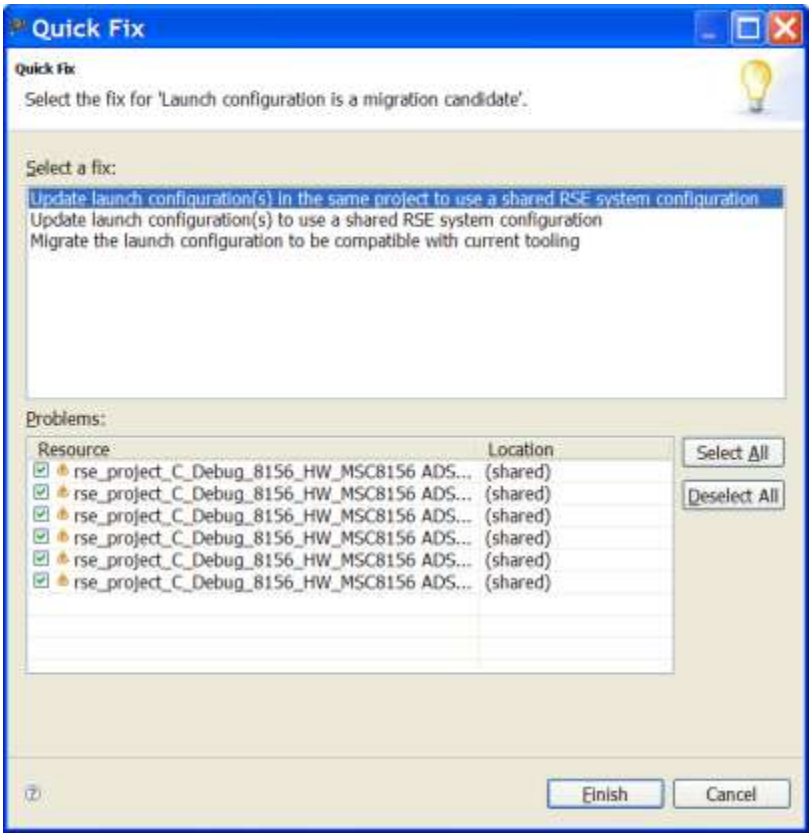

Figure 9. The Quick Fix dialog.

By selecting the fix method, the user can control the scope of the migration process. Put another way, the choice of fix method filters for relevant configurations that are then displayed for repair in the **Problems** list. Table 2 summarizes how these choices filter for connection type, processor and project.

Table 2. Quick Fix Migration Options.

Option	Description
Update launch configuration(s) in the same project to use a shared RSE system configuration	Performs the migration on a subset of launch configurations having the same connection type, processor and project.
Update launch configuration(s) to use a shared RSE system configuration	Performs the migration on a subset of launch configurations having the same connection type and processor.
Migrate the launch configuration to be compatible with current tooling	Performs the migration all selected launch configurations regardless the connection type, processor or project.

11. Choose the migration method in **Select a Fix** option. Click **Finish**.

The **Migrate Launch Configuration for RSE** dialog appears (Figure 10).

Figure 10. The Migrate Launch Configuration Dialog.

Since the project being imported does not have any system information in a RSE format, the **Create new RSE system named** option is selected.

12. Click **OK**.

The importer converts the system data into a RSE-compliant system configuration and the warnings disappear.

13. Choose Debug **As > Debug Configurations**. The **Debug Configurations** window appears (Figure 11). It displays the converted launch configurations.

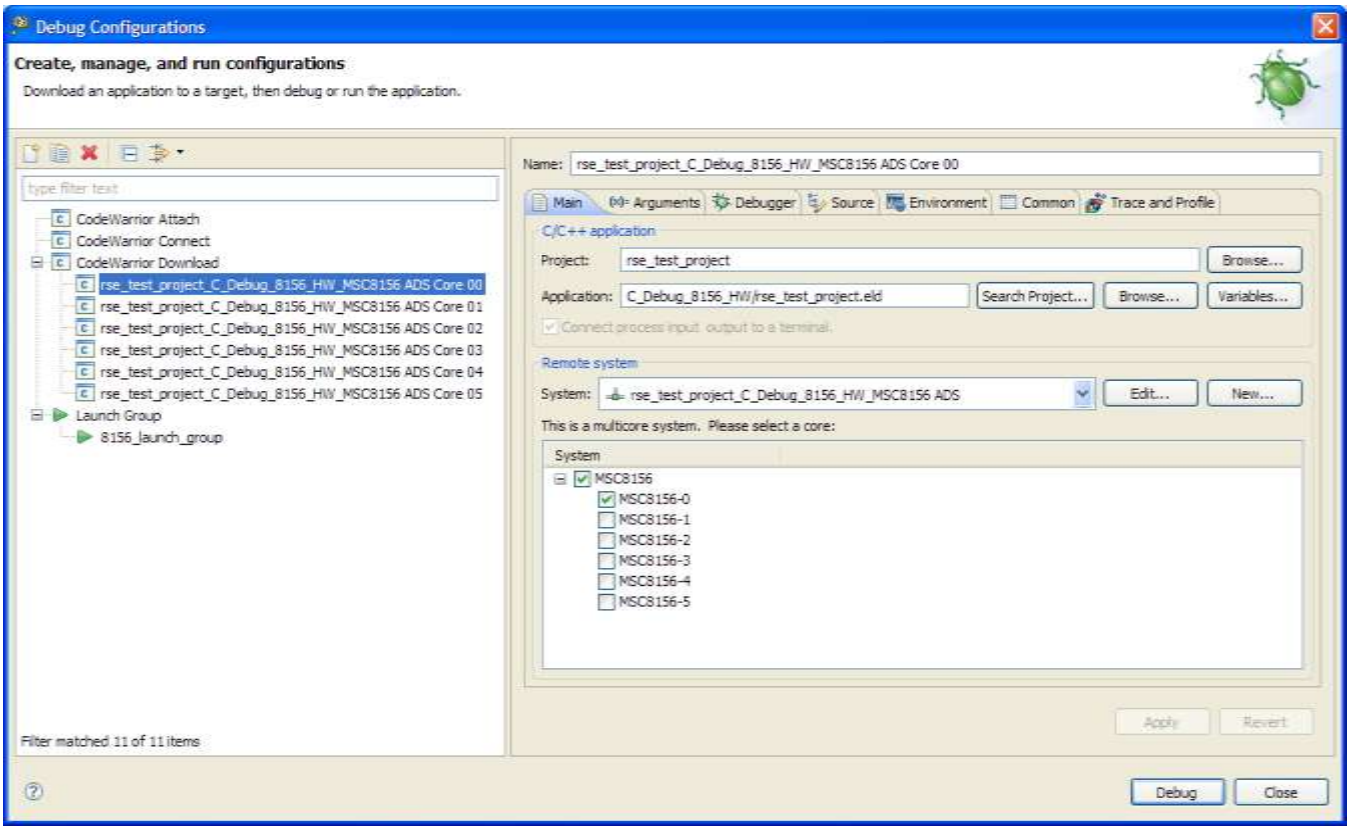

Figure 11. The Debug Configuration Window, Displaying the Converted Launch Configurations.

14. If the code for one core is being debugged, select its debug configuration from under the **CodeWarrior Download** list.

15. Click **Debug**.

The debugger perspective appears.

16. Click **Resume** to execute the application under debugger control.

2.1 Importing Launch Groups

One thing to be aware of when importing and exporting projects is that the launch groups are not imported unless they have been shared in the original project. To work around this potential issue, the launch groups should be save into the workspace so that the IDE stores the launch configuration file and it can be committed into the repository. If these settings are not shared, then they must be added manually to these launch groups after the project is imported. In order to create a launch group and share these settings, follow the steps below:

1. Right-click on a project name, and context menu appears.
2. From context menu, select **Debug As > Debug Configurations**.
3. Select **Launch Group**.
4. From the toolbar, select **New Launch Configuration**.

Under **Launch Group**, `New_configuration` appears.

5. Under the **Launches** tab, select **Add**.

The **Add Launch Configuration** window appears.

6. Expand the **CodeWarrior Download** item.
7. Click on the first configuration.
8. Hold down the Shift key.
9. Click on the last configuration. All of the configuration should be selected (highlighted).
10. Click **OK**.

All the core launch configurations appear in the **Launches** tab.

NOTE

For more information on how to create launch groups and share their configurations, consult the *Freescale Eclipse Extensions Guide*.

3 System Settings Differences Between Non-RSE Projects and RSE-based Projects

Because the RSE framework has factored out the system settings from the launch configurations, it is to be expected that this restructuring also affects the layout of the IDE's settings. This section documents some of these changes.

3.1 Target Processor

In a non-RSE project, the selection of the target hardware is from the **Target Processor** option (Figure 12). Under the RSE scheme, the target hardware is specified under **System Type** (Figure 13). The System Type settings are located under the **Main** tab of **Debug Configurations** window.

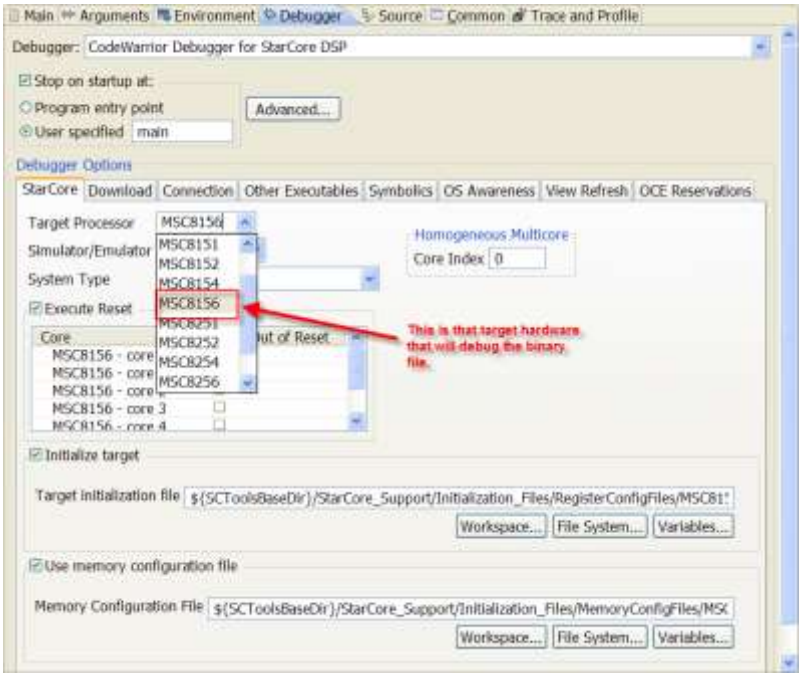

Figure 12. Non-RSE CodeWarrior Project, Hardware Target Setting.

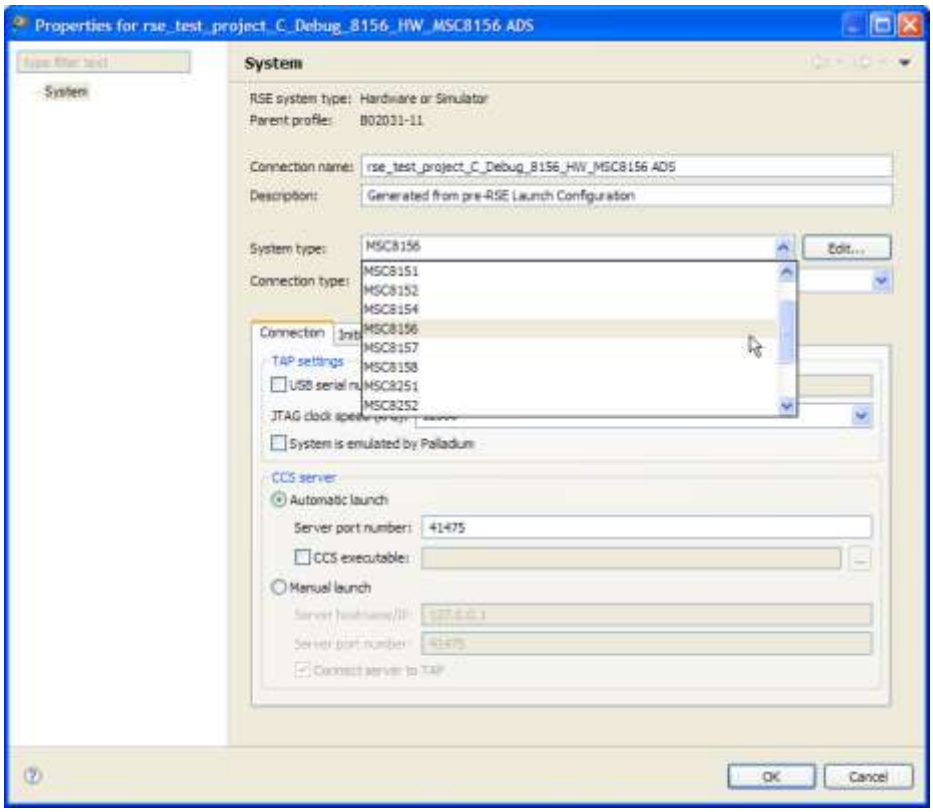

Figure 13. RSE-based CodeWarrior Project, Hardware Target Setting.

3.2 Instruction Set Simulator (ISS)

In a non-RSE CodeWarrior project, to connect to a simulator, choose `CCSSIM2 ISS` under the **Simulator/Emulator** option (Figure 14). For the RSE-based project, the **Connection Type** option displays the simulator choices (Figure 15).

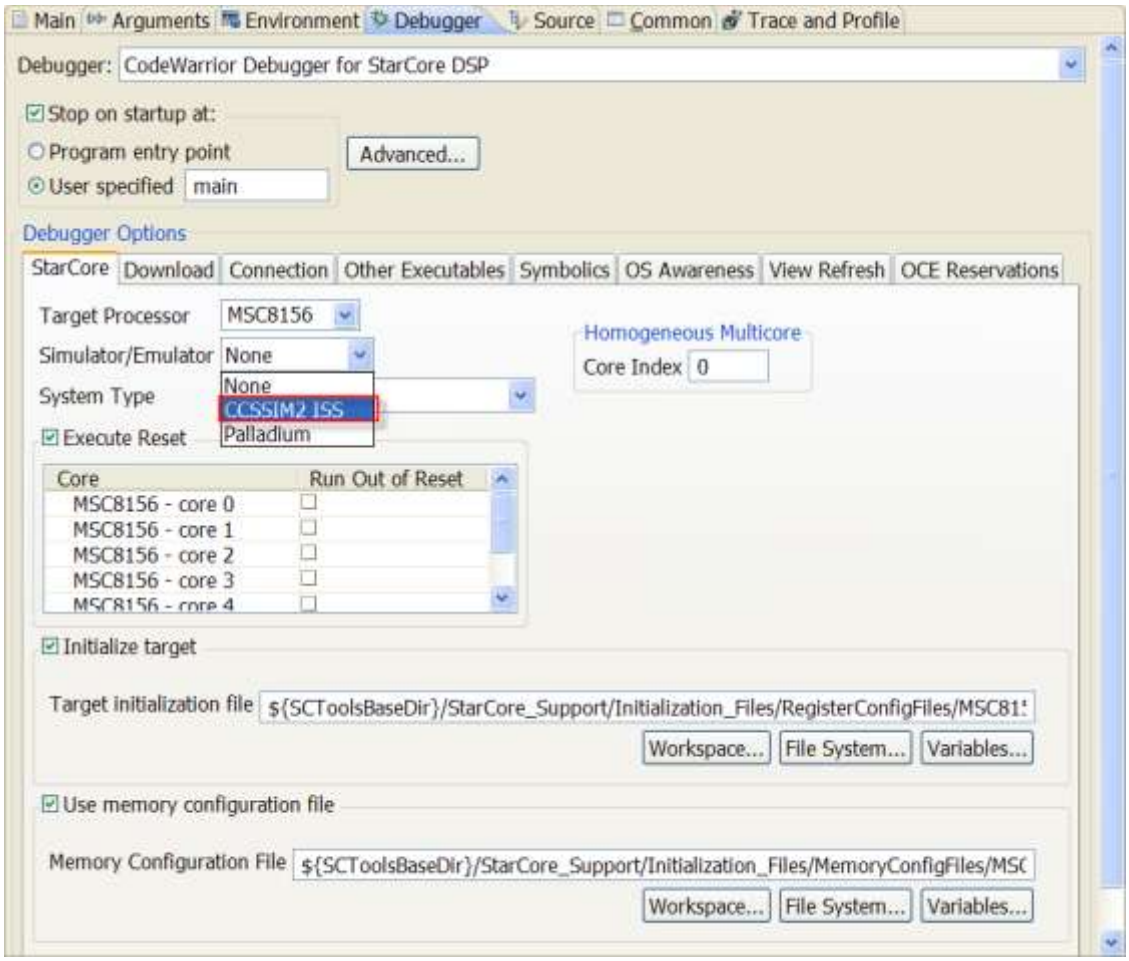

Figure 14. Non-RSE CodeWarrior Project, ISS Setting.

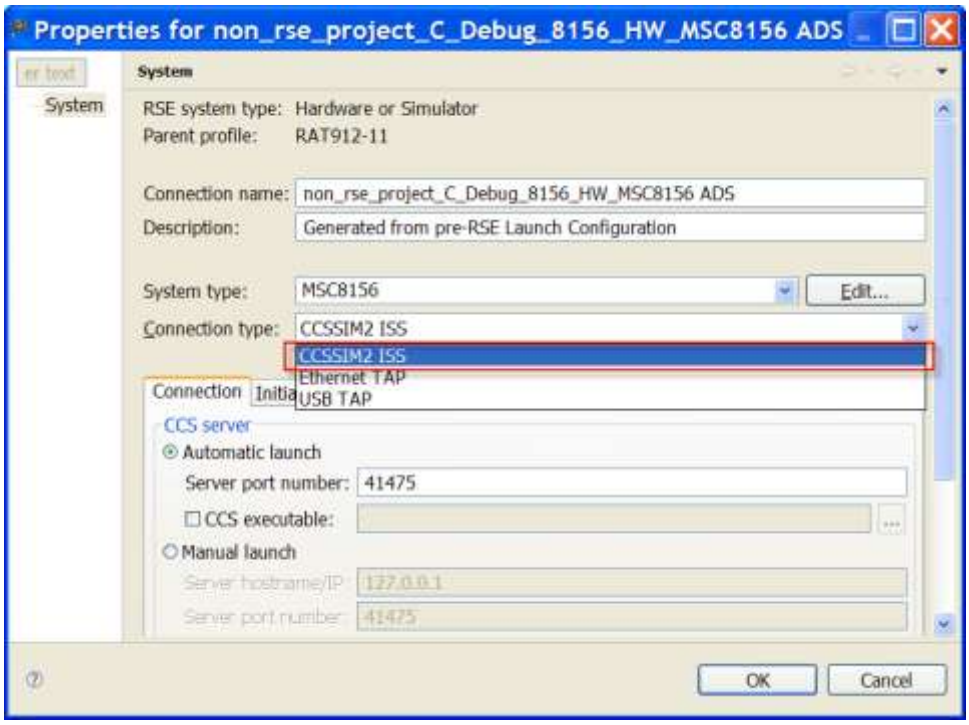

Figure 15. RSE-based CodeWarrior Project, ISS Settings.

3.3 Emulator

In a non-RSE CodeWarrior project, to connect to a hardware emulator choose `Palladium` under the **Simulator/Emulator** option (Figure 16). For a RSE-compliant project, the emulator choice appears in the **Connection Type** option (Figure 17). In order to access the **Connection Type** option in the RSE-based project, the emulator connection must be specified first. For example, when connecting to the target via USB TAP, first choose select **USB TAP** under the **Connection Type** option. Once the connection type has been selected, the emulation option can be chosen by checking the **System is emulated by Palladium** option.

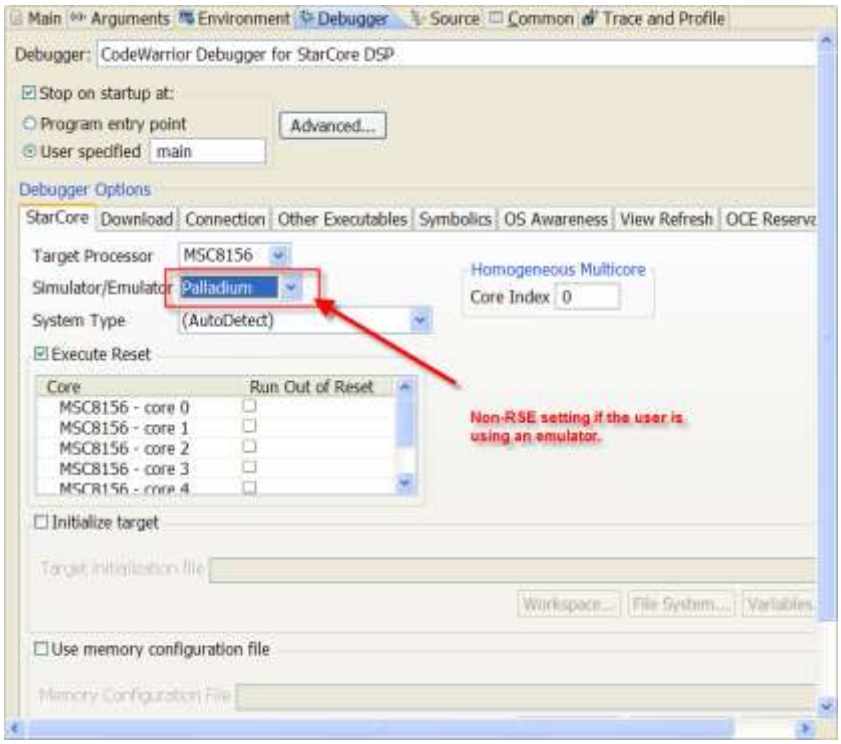

Figure 16. Non-RSE CodeWarrior Project, Emulator Setting.

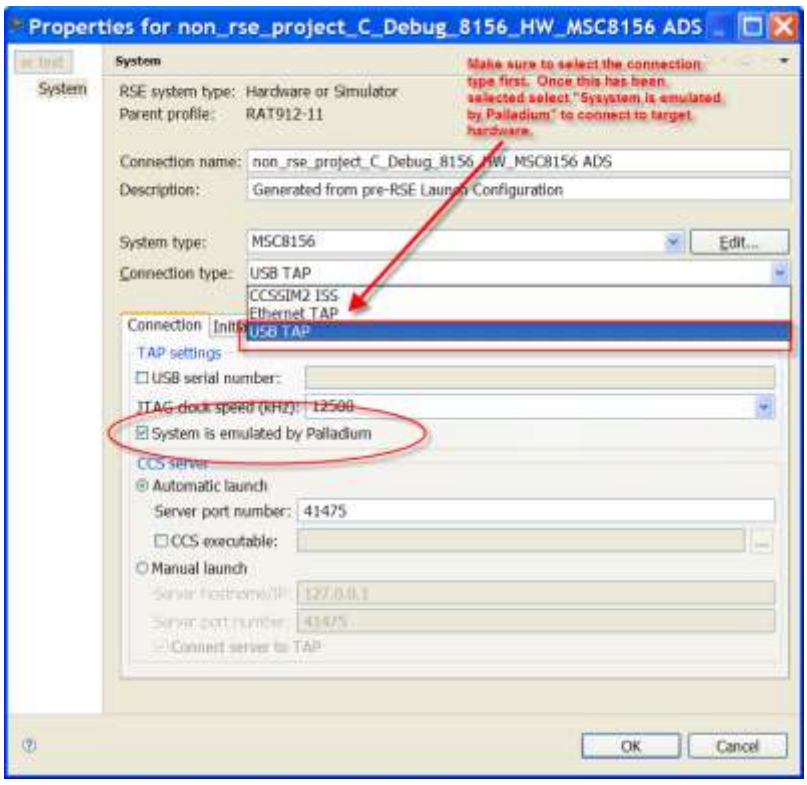

Figure 17. RSE-based CodeWarrior Project, Emulator Setting.

3.4 System Type

In a non-RSE CodeWarrior project, when connecting to a multicore system choose **System Type** under the **System Type** option (Figure 18). By default this option is set to `(Auto Detect)`, which detects the presence of a multicore system using the following criteria:

1. Checks the connection preference to see if a particular JTAG configuration file was specified.
2. If a JTAG file exists, checks to see if the system type was previously created using the JTAG file.
3. If an imported system type was found, use its system type. Otherwise, create the system type and set it using the system information imported from the JTAG config file.
4. If the JTAG setting or system cannot be determined for the specified JTAG, as a last resort the IDE consults an internal list of predefined values to specify the system type.

If the JTAG file must specify a custom system, this is accomplished through the **Edit Multicore System Types** option.

For a RSE-based CodeWarrior project, the system type can be chosen from **System** option, which stores a list of supported configurations on the debugged target (Figure 19).

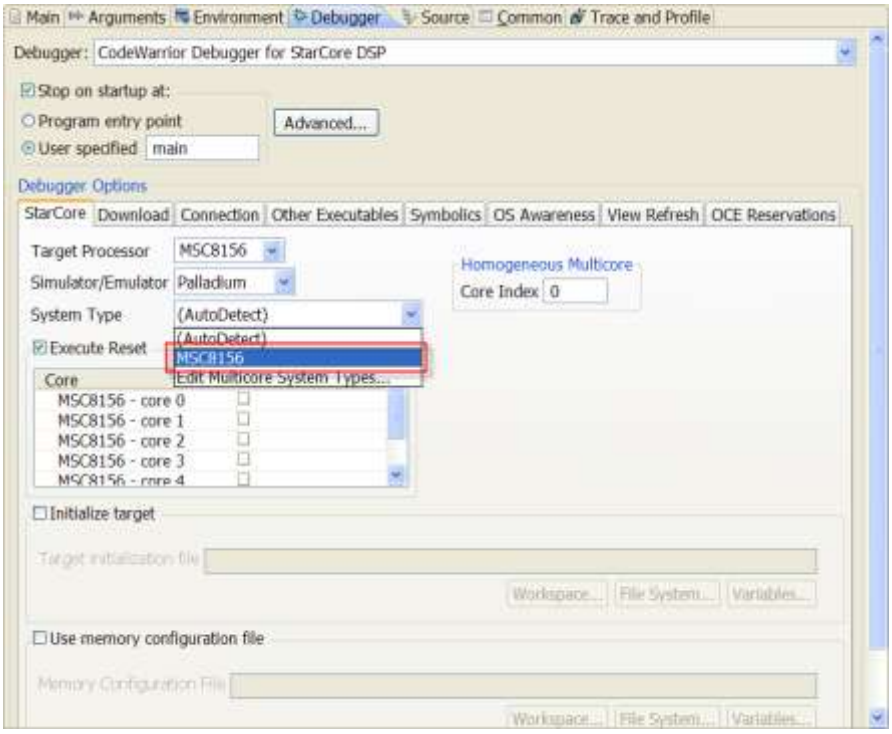

Figure 18. Non-RSE CodeWarrior Project, System Setting.

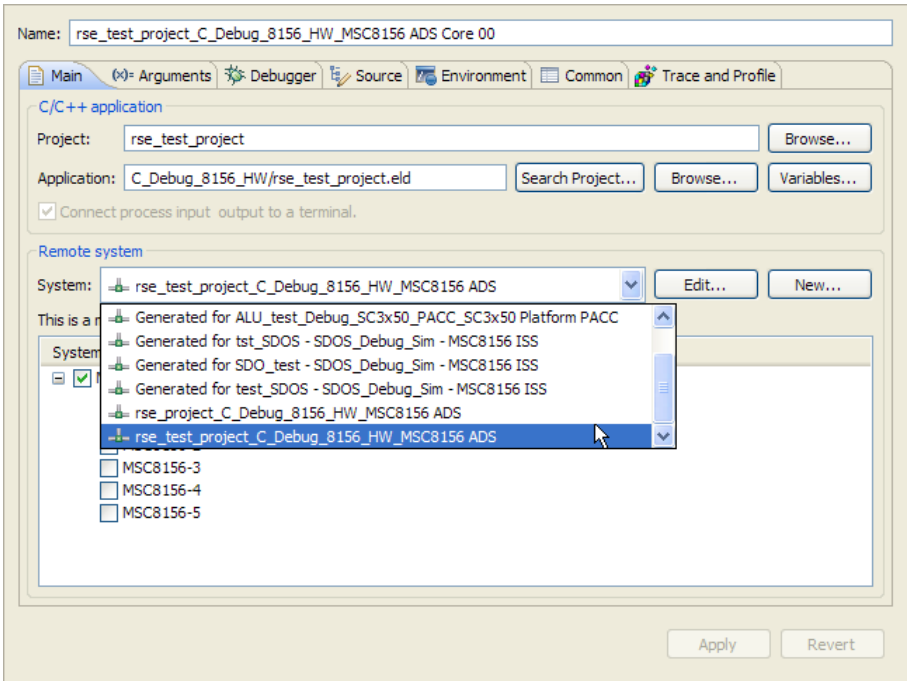

Figure 19. RSE-based CodeWarrior Project, System Setting.

3.5 Core Index

In a non-RSE CodeWarrior project, the core index value designates which core the debugger uses to execute the program. It is specified in the **Core Index** option (Figure 20). For a RSE-based project, the core index is located on the **Main** tab as an option under the **Remote system** group (Figure 21). For a non-RSE project if the core index was `0`, then in the RSE-based system this value is `MSC8156-0`.

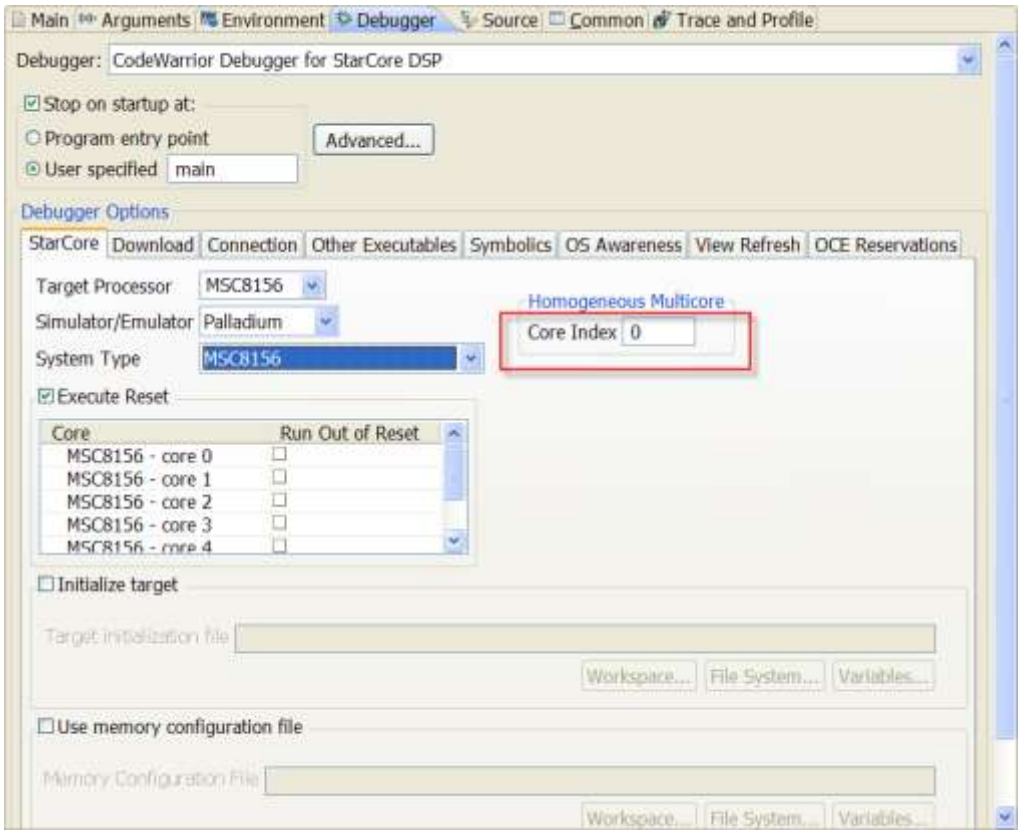

Figure 20. Non-RSE CodeWarrior Project, Core Index Setting.

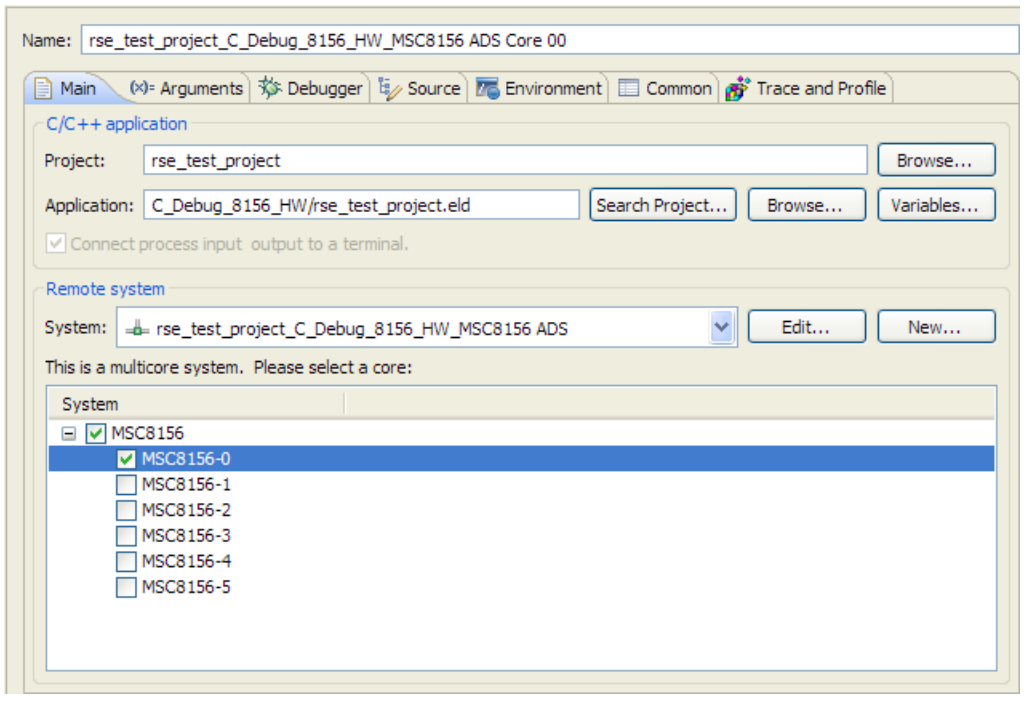

Figure 21. RSE-based CodeWarrior Project, Core Index Setting.

3.6 Execute Reset

In a non-RSE project, the execute reset option has the debugger reset the processor before downloading code to it (Figure 22). If a multicore processor is being debugged, this reset operation resets the specified core. In the RSE-based CodeWarrior project, execute reset is now part of the system configuration (Figure 23). Select **Execute reset (applies to initial target only)** to reset a specific core. If this option is not selected, then the debugger downloads the application to the target without first resetting the target. When doing multicore debugging, this option should typically be set to reset the first core launched, core 0.

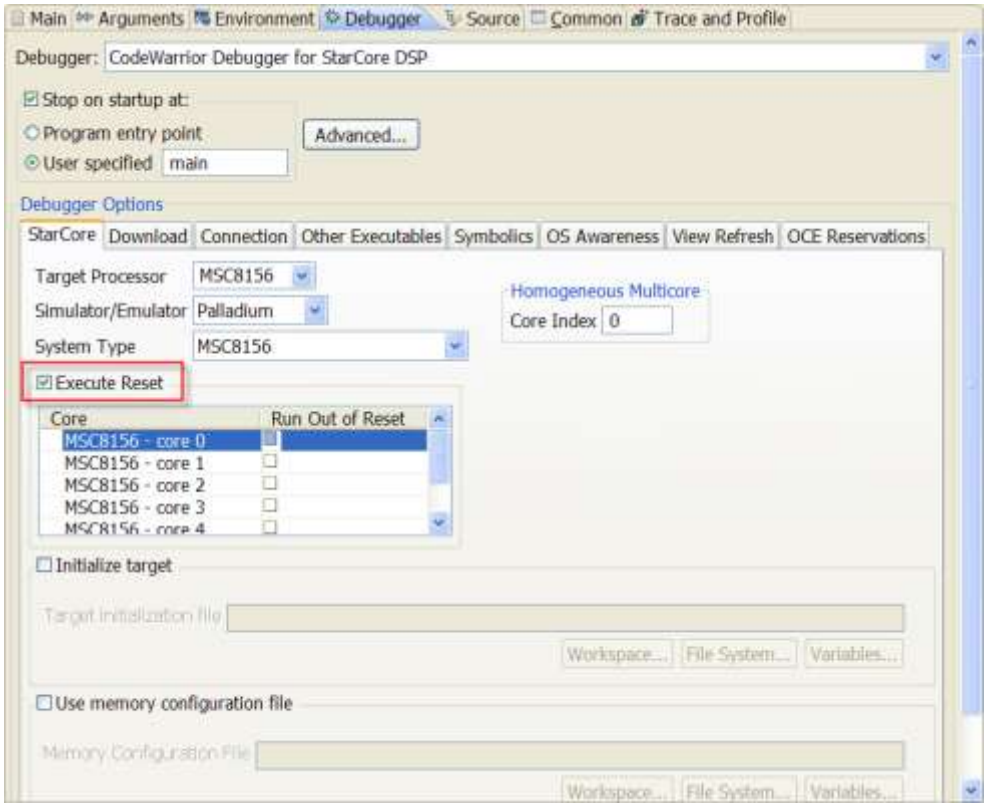

Figure 22. Non-RSE CodeWarrior Project, Execute Reset Setting.

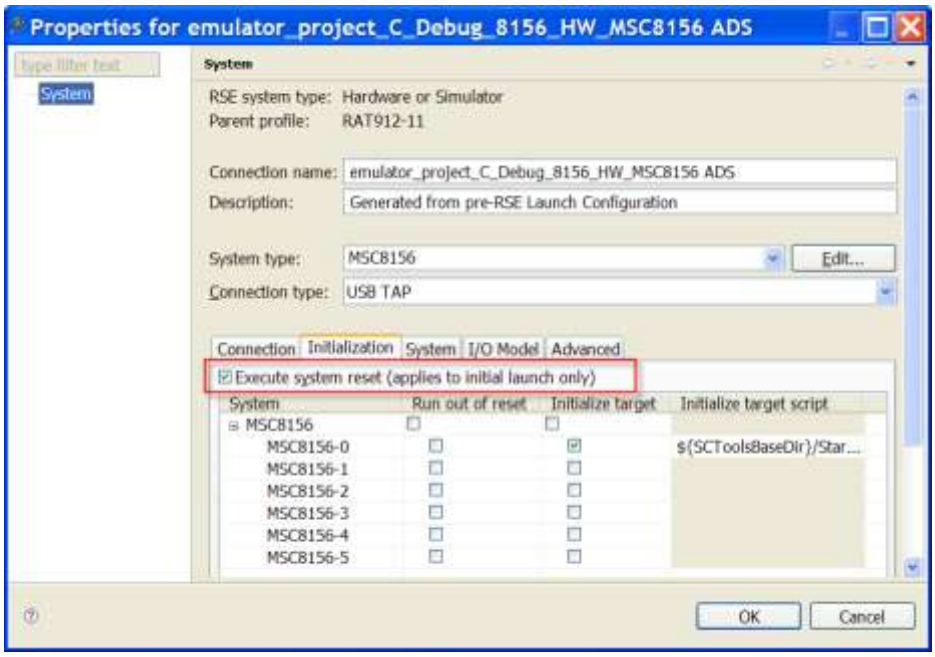

Figure 23. RSE-based CodeWarrior Project, Reset Memory Setting.

3.7 Run Out of Reset

In a non-RSE CodeWarrior project, check the **Run Out of Reset** option if the corresponding core in the core list should begin running after the IDE resets the core. If the **Run Out of Reset** option is unchecked, the corresponding core in the core list goes into debug mode after the IDE resets it (Figure 24). This option is located under the **StarCore** settings tab. In an RSE-based project, this setting is located under the **Initialization** tab for the system configuration (Figure 25). Here, the **Run Out of Reset** checkbox adjacent to the specified core determines whether it is simply reset or goes into debug mode after a processor reset.

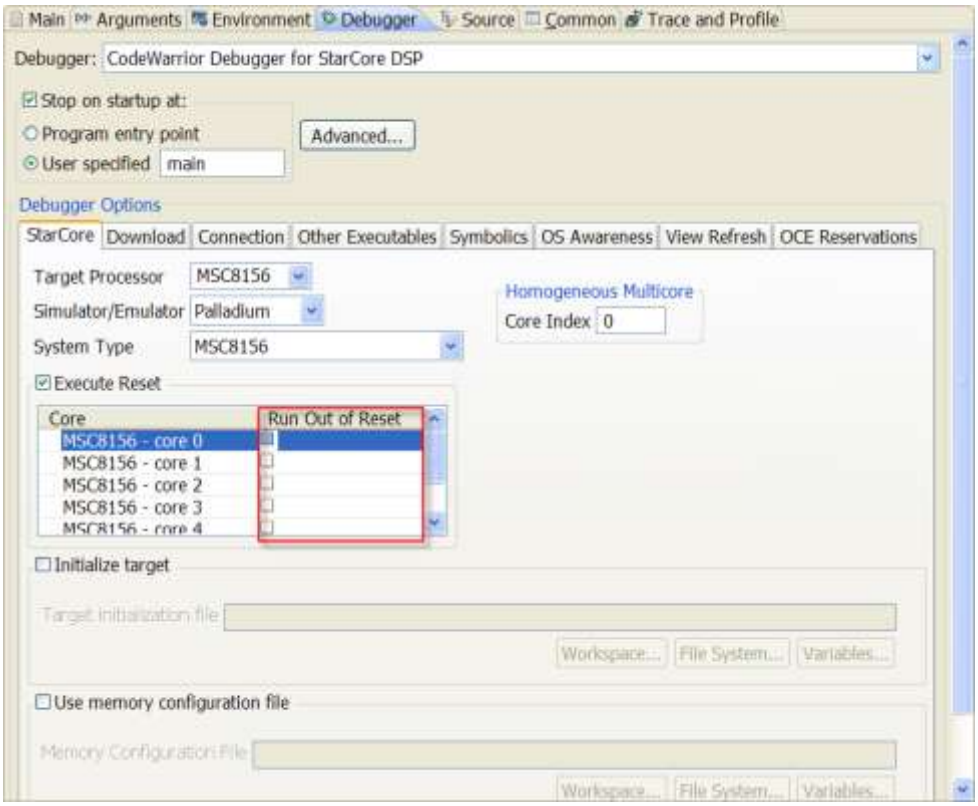

Figure 24. Non-RSE CodeWarrior Project, Run Out of Reset Setting.

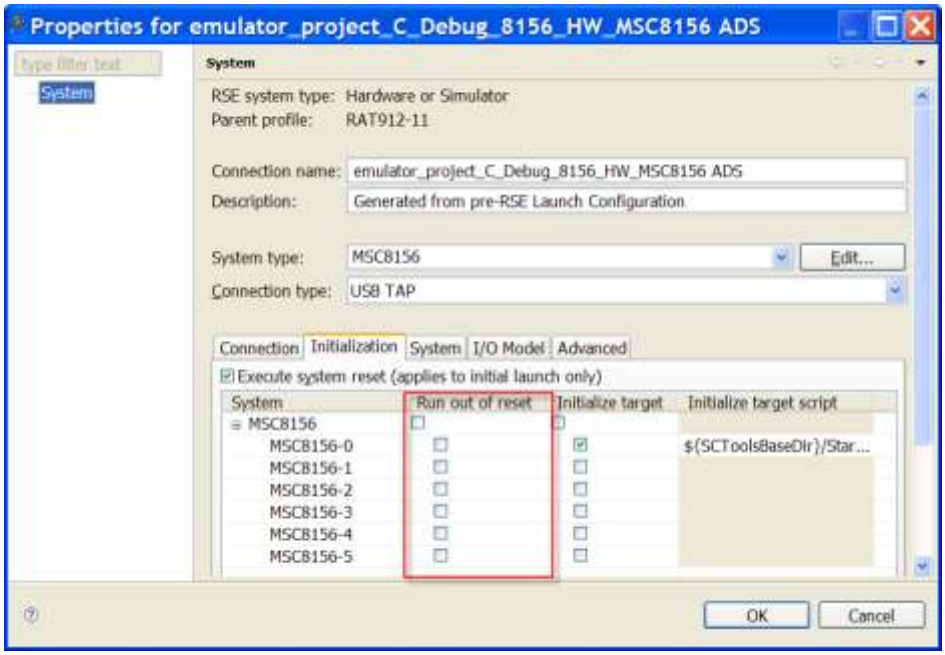

Figure 25. RSE-Based CodeWarrior Project, Run Out of Reset Setting.

3.8 Target Initialization File

For a non-RSE CodeWarrior project, the **Initialize Target** setting specifies the target initialization file the debugger reads when it initializes the processor. This file is read at the start of each debugging session. This option is located under the **StarCore** tab (Figure 26). In a RSE system, the setting is located under the **Initialization** tab. To use the setting, check the **Initialize Target** checkbox adjacent to the core to be initialized (Figure 27). The default initialization file name is `Initialize <target script>`. Only the first core has an initialization file specified. To specify a different initialization file name, click on the ellipsis button (…) and navigate to the new file. By default, the IDE uses the script file located in the tools initialization directory.

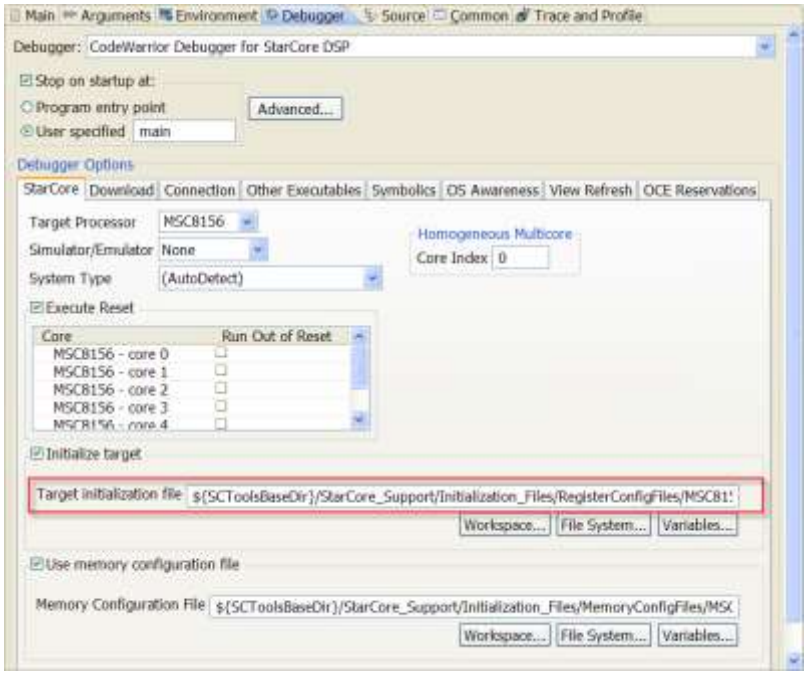

Figure 26. Non-RSE CodeWarrior Project, Target Initialization File Setting.

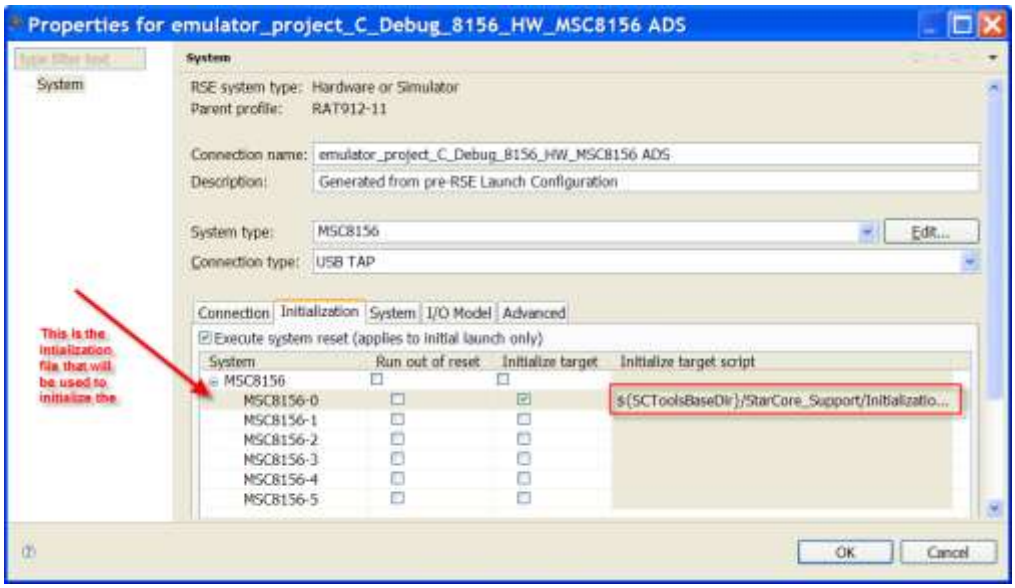

Figure 27. RSE-based CodeWarrior Project, Target Initialization File Setting.

3.9 Memory Configuration File

For a non-RSE CodeWarrior project, the **Use Memory File** setting specifies the memory configuration file the debugger reads when it initializes the processor. This file is read at the start of each debugging session and configures the target's memory space as to the amount of memory and the memory type. This option is located under the **StarCore** tab (Figure 28). In a RSE system, the setting is located under the **System** tab. To use the setting, check the **Memory Configuration** checkbox adjacent to the desired core (Figure 29). Only the first core has a memory configuration file specified. To specify a different configuration file name, click on the ellipsis button (…) and navigate to the new file. By default, the IDE uses the memory configuration file located in the tools initialization directory.

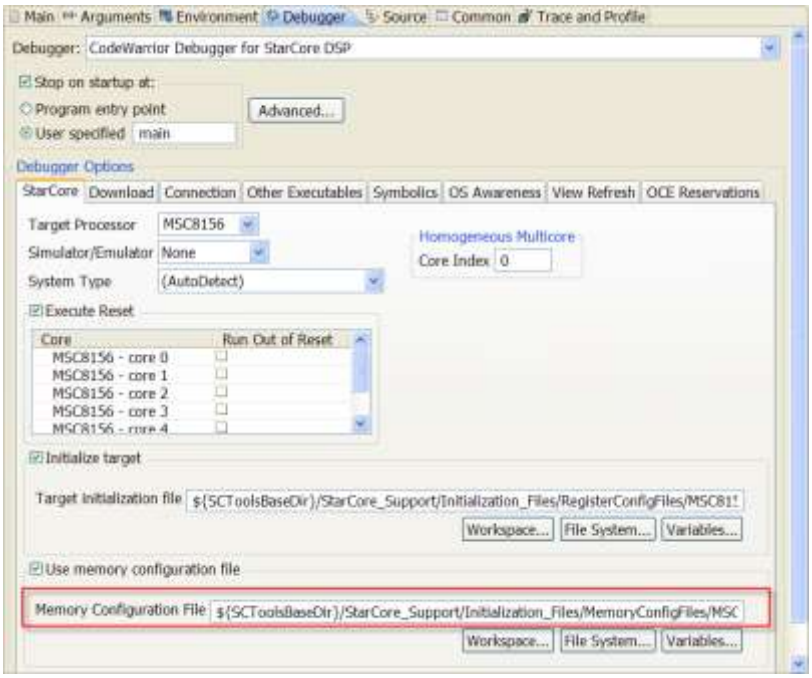

Figure 28. Non-RSE CodeWarrior Project, Memory Configuration File Setting.

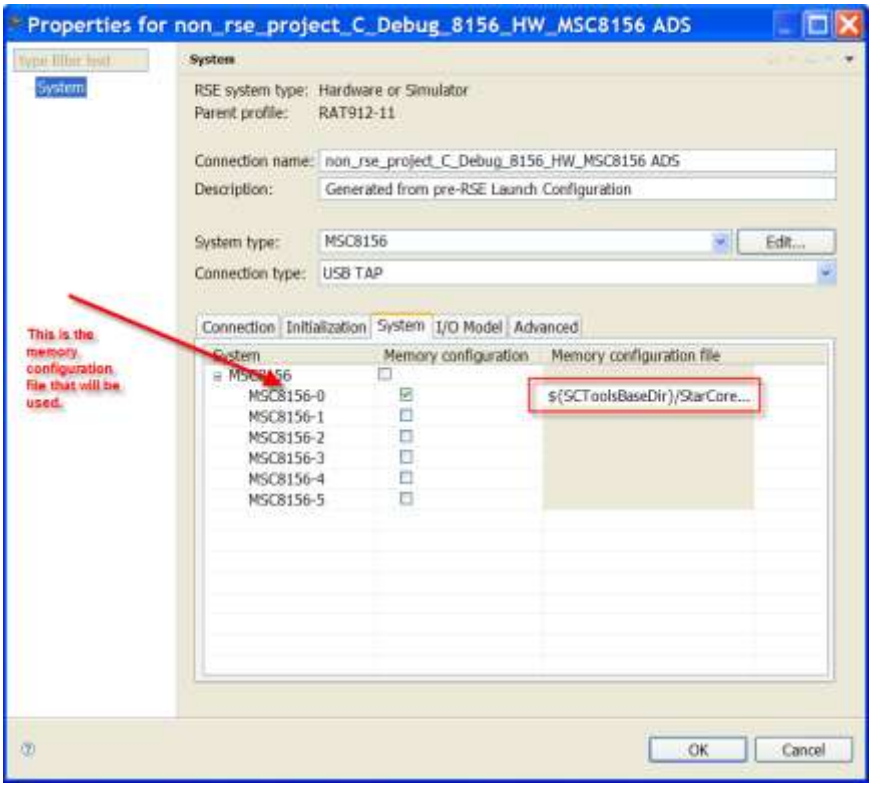

Figure 29. RSE-compliant CodeWarrior Project, Memory Configuration File Setting.

4 Connection Settings Differences Between Non-RSE Projects and RSE-based Projects

Because the RSE framework has factored out the connection settings from the launch configurations, this restructuring affects the layout of the IDE's settings regarding these settings. This section documents some of these changes.

4.1 Connection Protocol

In a non-RSE CodeWarrior project, the **Connection Protocol** option located under the **Connection** tab specifies the communications protocol used to interact with the target (Figure 30). The communications protocol is often managed by a CodeWarrior Connection Server (CCS), which provides flexibility in accessing a target board. For example, through a TCP/IP connection, the CCS allows each instance of the CodeWarrior debugger to access the target boards connected to any local or remote computer that is running a CCS instance. Each time a debugging session begins, it uses a local CCS connection; the CodeWarrior IDE automatically starts CCS if it is not running already. In a RSE-based project, the user no longer has to specify the connection protocol, just the connection type (Figure 31). This is because for CodeWarrior for StarCore the only available connection protocol is CCS, so the redundant setting was removed.

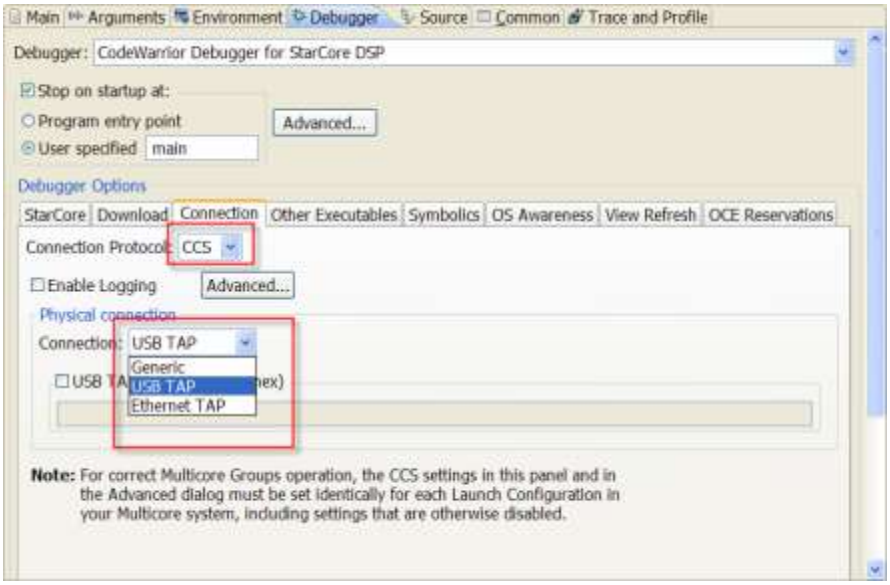

Figure 30. Non-RSE CodeWarrior Project, Connection Type Setting.

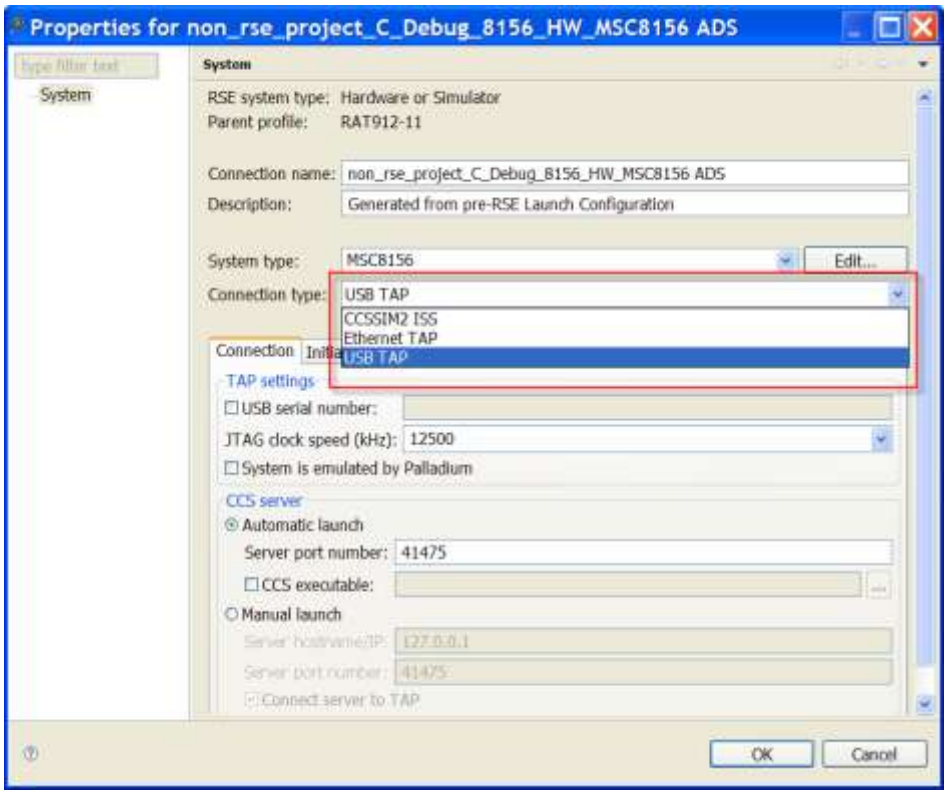

Figure 31. RSE-based CodeWarrior Project, Connection Type Setting.

4.2 Enable Logging

In a non-RSE CodeWarrior project, when the **Enable Logging** option is enabled, the debugger outputs connection protocol activity to a console in a **Console** view. This option is located under the **Connection** tab (Figure 32). For a RSE-based project, the **Enable Logging** option was moved to the **Advanced** tab (Figure 33).

Figure 32. Non-RSE CodeWarrior Project, Enable Logging Setting.

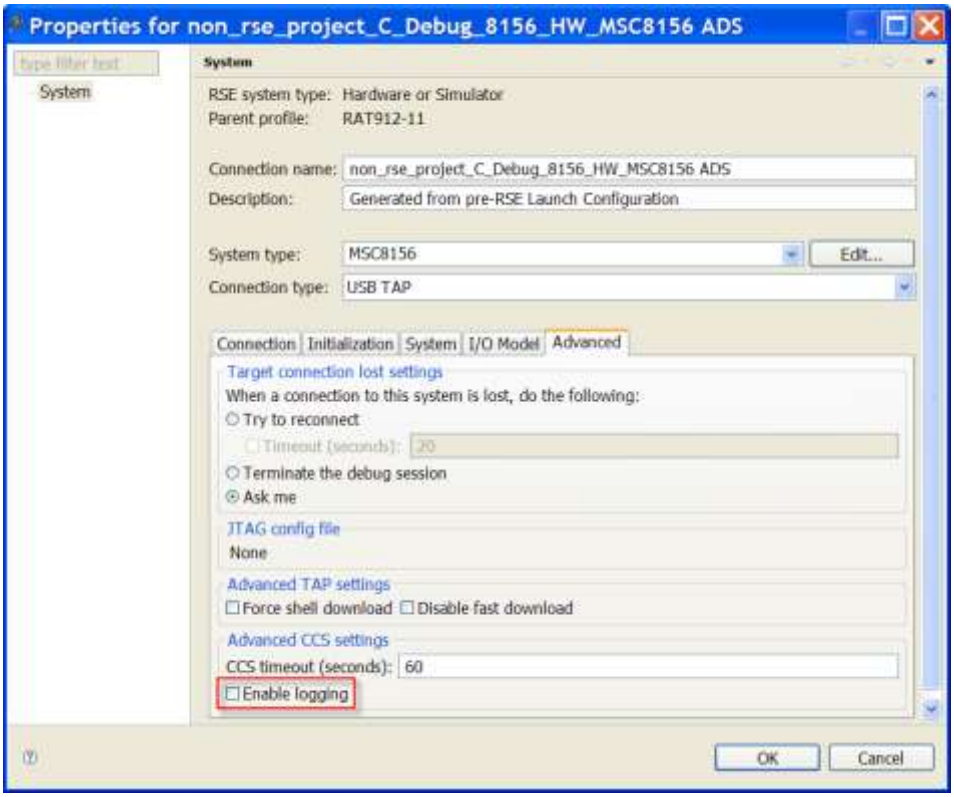

Figure 33. RSE-based CodeWarrior Project, Enable Logging Setting.

4.3 Physical Connection

In a non-RSE CodeWarrior project, the **Connection** tab is where the user specifies the interface that communicates with the hardware. Support is available for USB TAP, Ethernet TAP, and generic interfaces. The choice of connection is set through the **Physical Connections** option (Figure 34). For a RSE-based project, the settings are now part of the **System** properties window and under the **Connection Type** option (Figure 35). After the physical connection type is chosen, further configuration of the connection can be done through the options that appear in the **Connection** tab.

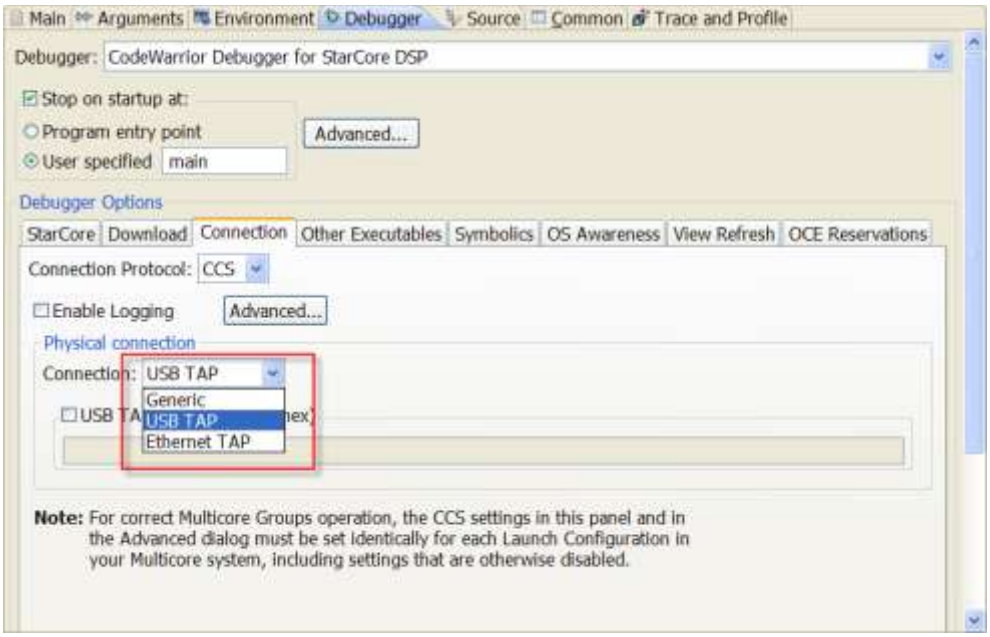

Figure 34. Non-RSE CodeWarrior Project, Physical Connection Setting.

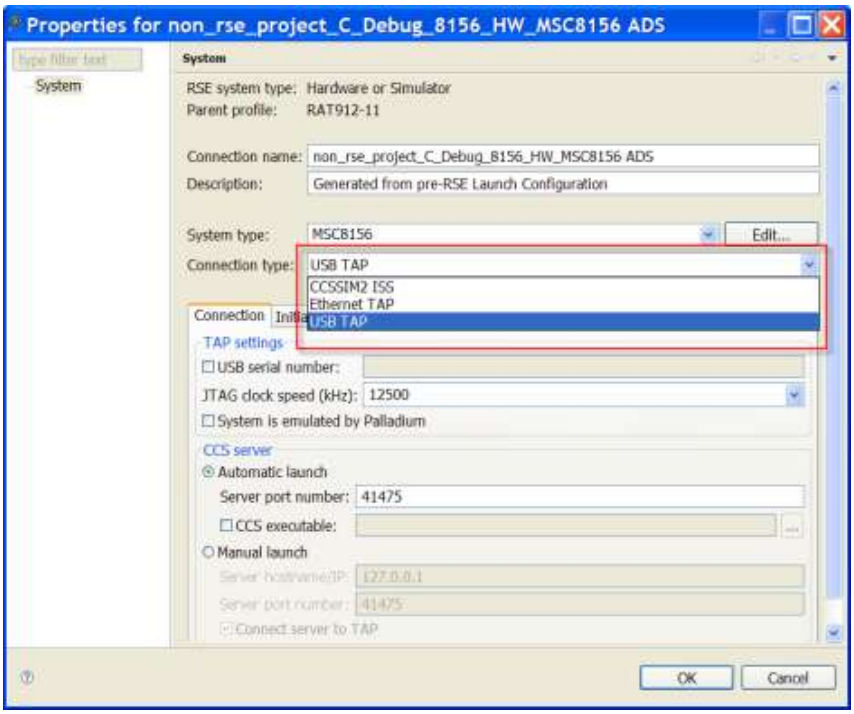

Figure 35. RSE-Based CodeWarrior Project, Physical Connection Setting.

4.4 Use CCS Remote Server

In a non-RSE CodeWarrior project, this option specifies the remote CCS instance with which the debugger communicates. This option describes CCS remote server's IP address, hostname, and IP port (Figure 36). For the RSE-based project, these settings are now part of the **Connection** tab. The **Manual Launch** option is where the IP address, hostname and port are entered (Figure 37).

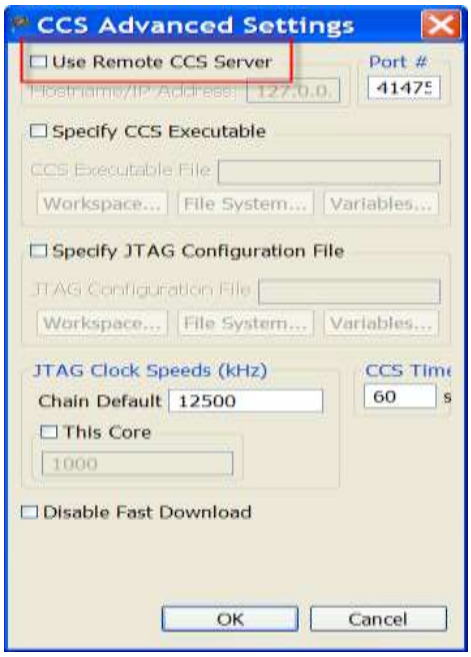

Figure 36. Non-RSE CodeWarrior Project, Use CCS Remote Server Setting.

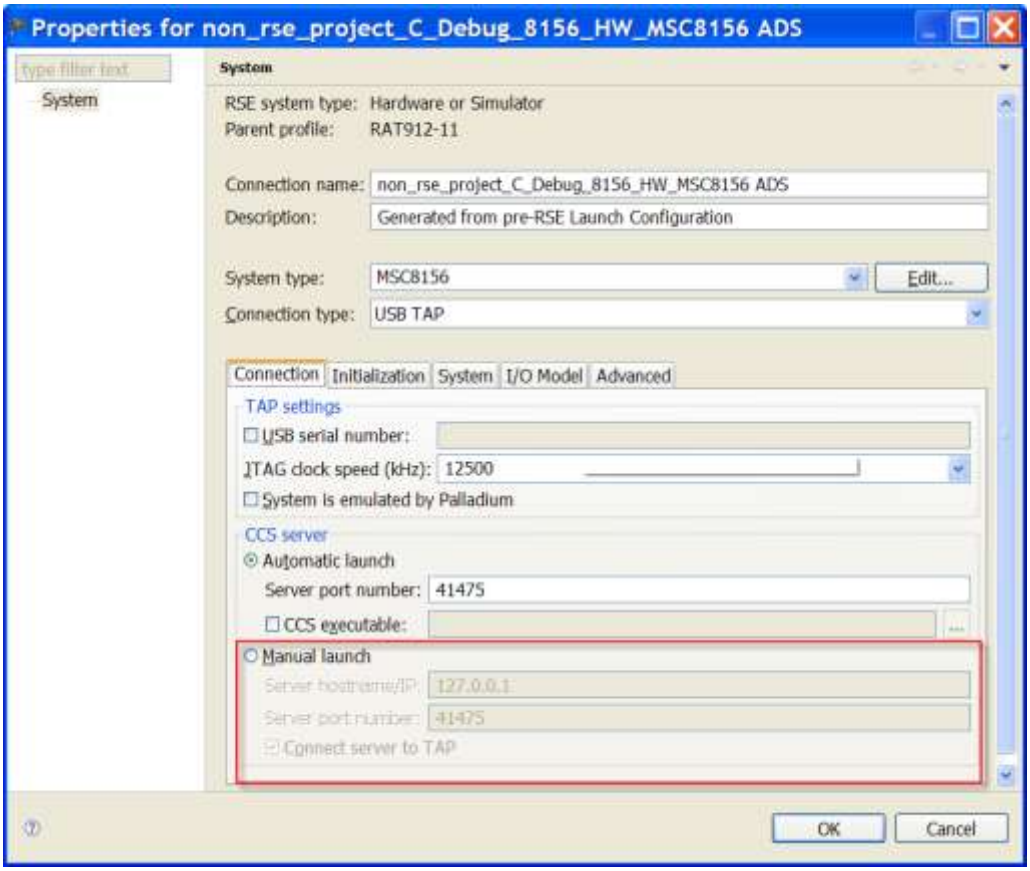

Figure 37. RSE-based CodeWarrior Project, Use CCS Remote Settings.

4.5 Specify CCS Executable

In a non-RSE project, when checked, this option specifies the path to the CCS executable file (other than the default) that the debugger launches if no CCS service is running when a debugging session starts. The full path the debugger uses is entered into the **CCS Executable File** option (Figure 38). Or, click **Browse** to open a dialog box and navigate to the desired executable file. In a RSE-based project, this setting has been moved into the **Automatic Launch** option (Figure 39). The full path is entered into the **CCS Executable** option, or click on the ellipsis button to display a dialog that is used to navigate to the file.

Figure 38. Non-RSE CodeWarrior Project, Specify CCS Executable Setting.

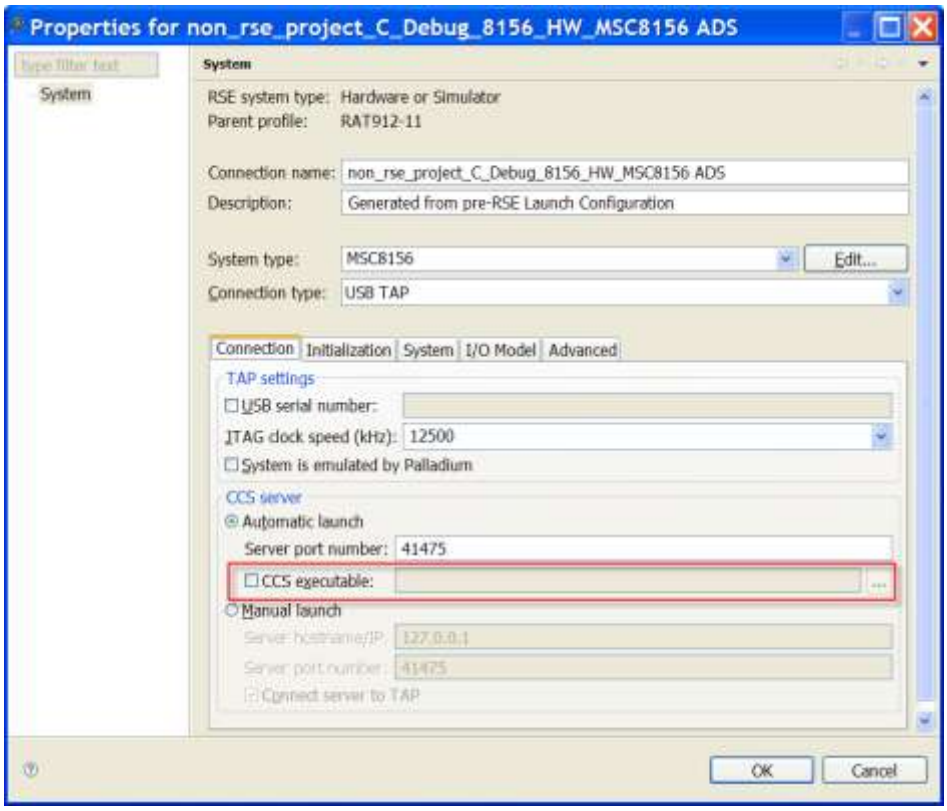

Figure 39. RSE-Based CodeWarrior Project, Specify CCS Executable Setting.

4.6 Specify JTAG Connection File

In a non-RSE project, this setting specifies the path that the debugger uses to locate the file that is read to configure the JTAG device chain. To have the debugger use a different file, check the **Specify JTAG Configuration File** option (Figure 40). The path to the file is entered into the **JTAG Configuration File** option. In a RSE-based project, this information is part of the system configuration. The setting is changed by choosing **Edit > Import** and selecting the appropriate file. If the JTAG configuration file was imported correctly, then this information appears in the **Advanced** tab, under the **JTAG config file** option (Figure 41).

Figure 40. Non-RSE CodeWarrior Project, Specify JTAG Connection File Setting.

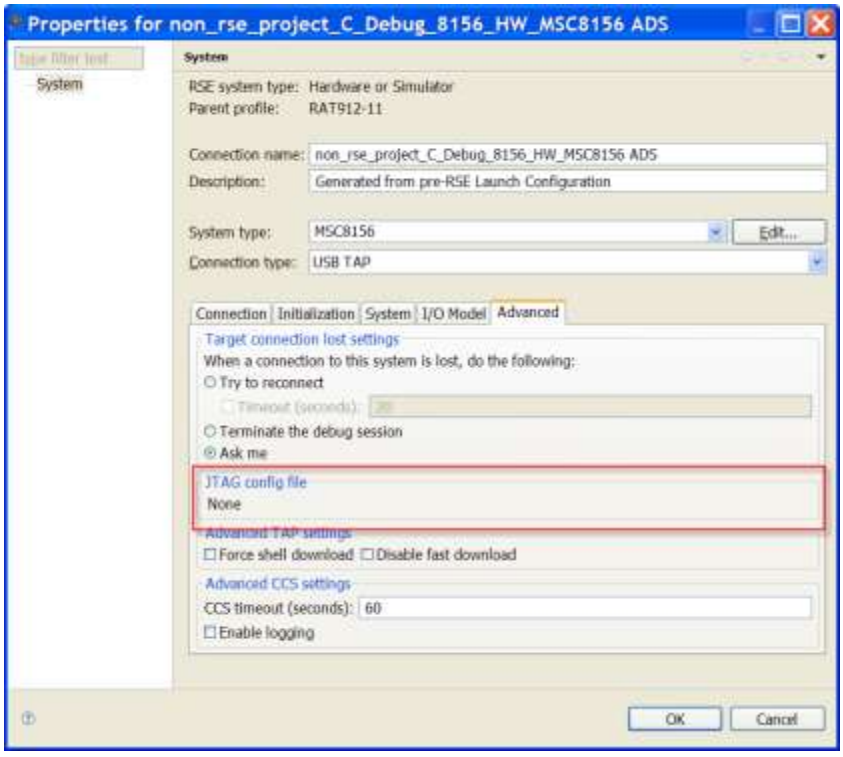

Figure 41. RSE-Based CodeWarrior Project, Specify JTAG Connection File Setting.

4.7 JTAG Clock Speed

In a non-RSE project, this setting managed the maximum clock speed for the JTAG interface in kHz. The speed is specified in the **Chain Default** option (Figure 42). For a RSE-based project, this setting is now only visible if it is used for configuring the selected connection. The clock speed is chosen from the **JTAG clock speed (kHz)** option (Figure 43).

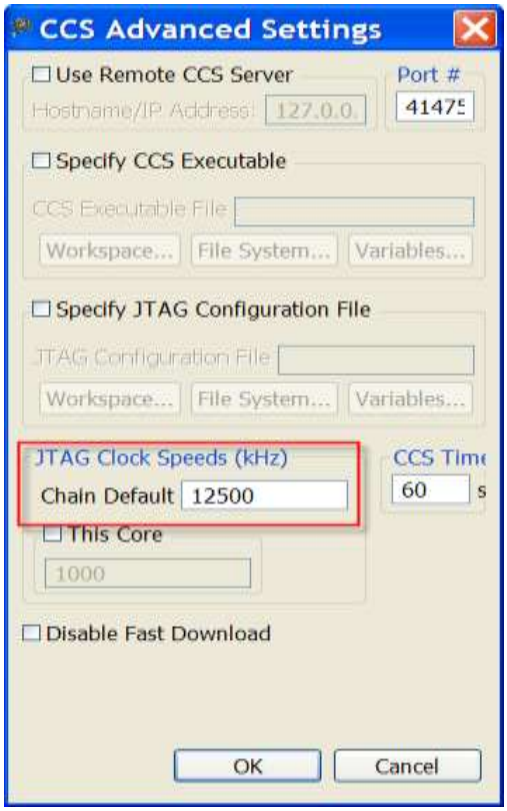

Figure 42. Non-RSE CodeWarrior Project, JTAG Clock Speed Setting.

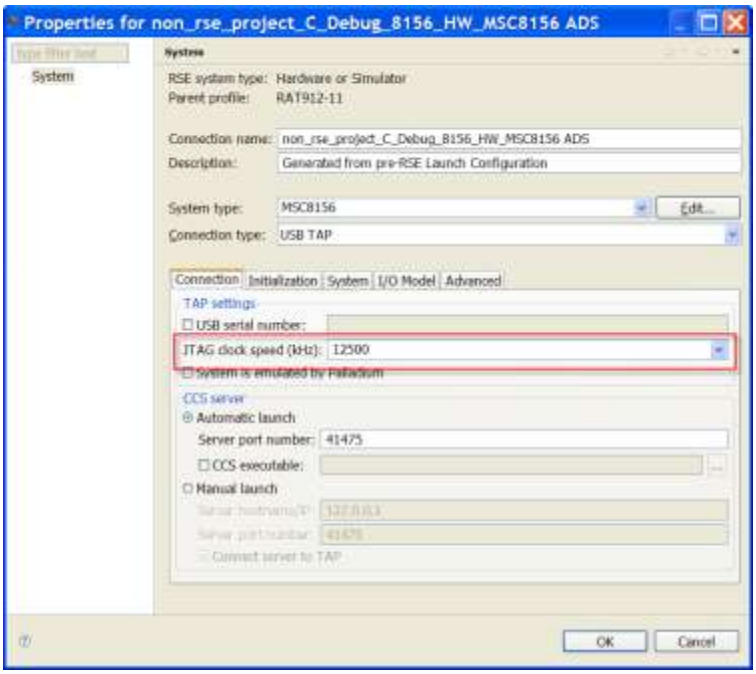

Figure 43. RSE_based CodeWarrior Project, JTAG Clock Speed Setting.

4.8 CCS Timeout

In a non-RSE project, this setting specifies the timeout for CCS communications in seconds. The value is entered into the **CCS Timeout** option (Figure 44). For a RSE-based project this setting was moved to the **Advanced** tab (Figure 45).

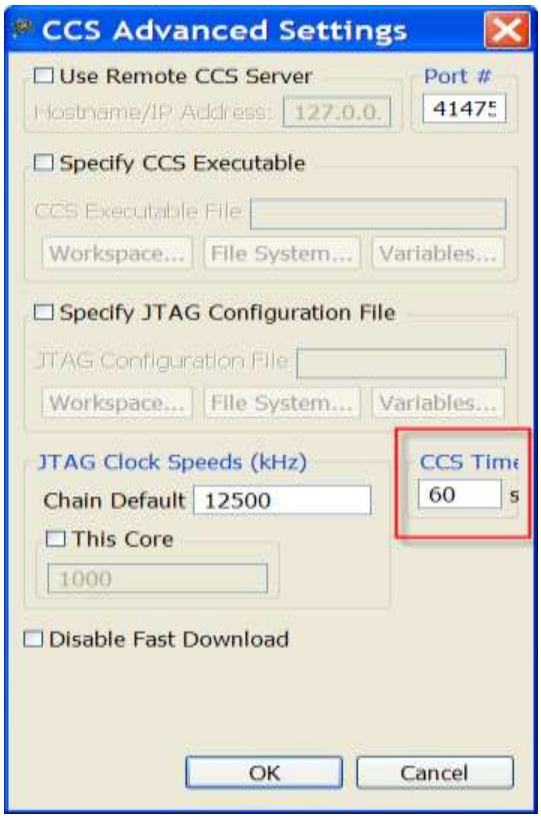

Figure 44. Non-RSE CodeWarrior Project, CCS Timeout Setting.

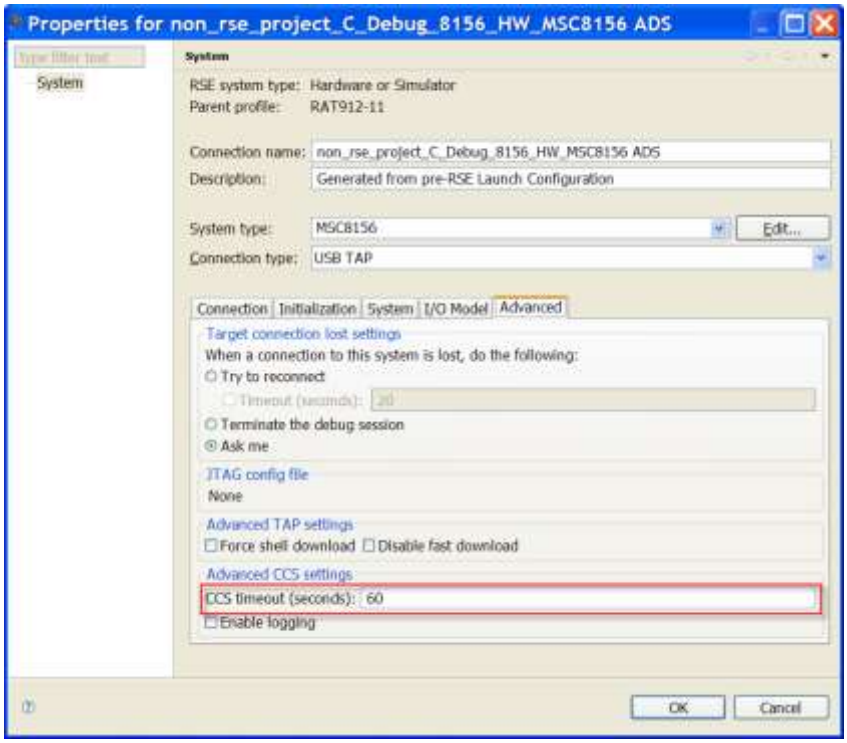

Figure 45. RSE-Based CodeWarrior Project, CCS Timeout Setting.

4.9 This Core

In a non-RSE project, this setting was used if a certain processor core required a different clock speed from the main core (Figure 46). In RSE-based projects, this setting has been removed.

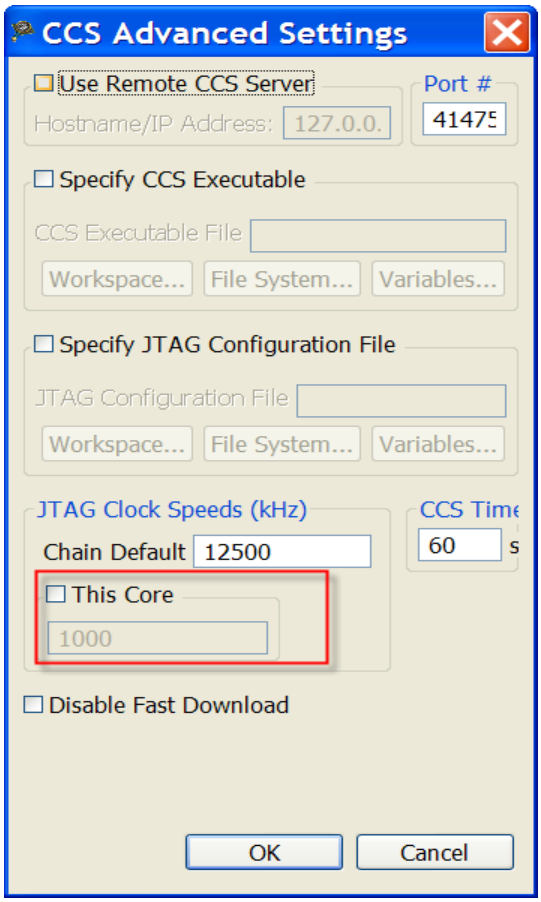

Figure 46. Non-RSE CodeWarrior Project, This Core Setting.

4.10 Disable Fast Download

In a non-RSE project, this setting specifies if the generated `.elf` binary file should be downloaded to the target using a fast download protocol. A slower yet more reliable download protocol is used if the **Disable Fast Download** option is checked (Figure 47). In a RSE-based project, this setting has been moved to the **Advanced** tab (Figure 48).

Figure 47. Non-RSE CodeWarrior Project, Disable Fast Download Setting.

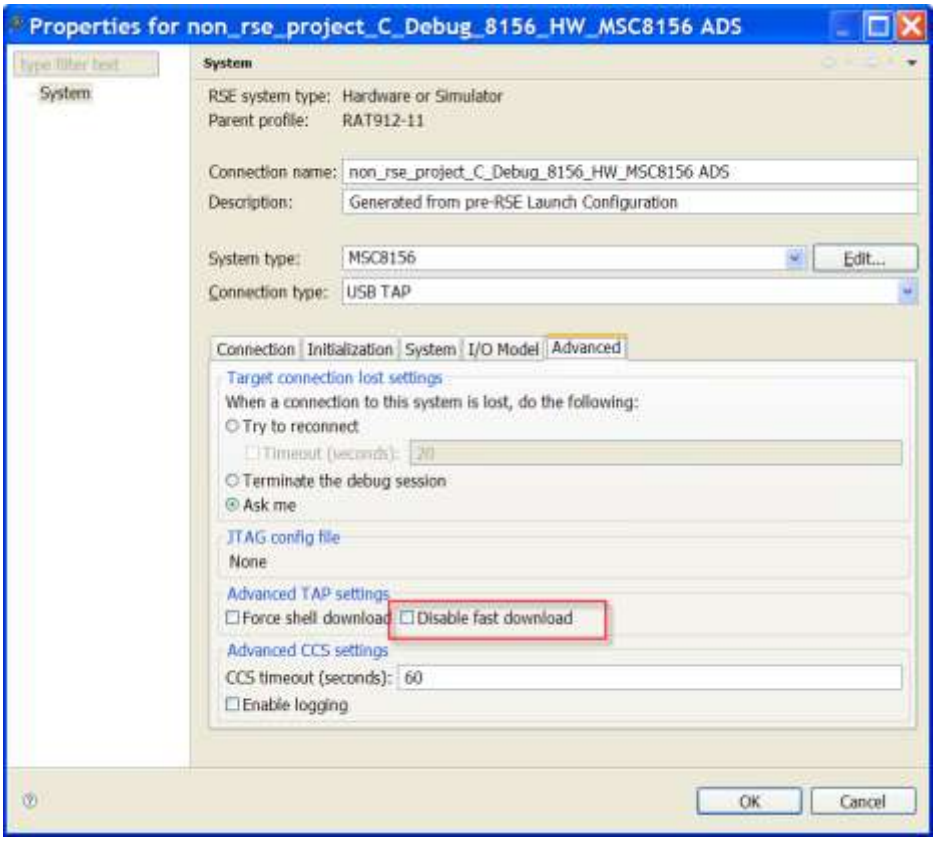

Figure 48. RSE-Based CodeWarrior Project, Disable Fast Download Setting.

5 Revision History

Table 3 provides a revision history for this application note.

Table 3. Revision History

Rev. Number	Date	Substantive Change
0	12/16/10	Initial creation.
1	01/19/11	Corrected information on specific options, modified description of launch configuration.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number:
Rev. 1
01/2011