Freescale Semiconductor
Application Note

Document Number: AN4145
Rev. 0, 06/2010

Debugging Multicore StarCore DSP Applications with Eclipse

by DevTech Support
Freescale Semiconductor, Inc.
Austin, TX

With the release of CodeWarrior for StarCore DSPs v10, the Freescale debugging tools are managed by the Eclipse Integrated Development Environment (IDE). This user interface (UI) differs from the UI of the original “Classic” CodeWarrior IDE. In particular, the new multicore debugging interface differs significantly from its predecessor. This document describes the Eclipse interface features that are specific to multicore debugging, and how to use them.

Contents

© 2010 Freescale Semiconductor, Inc.

1 Starting the Debugger with Multiple Cores

After the multicore application is built, the next step is to download/execute it on multiple cores. The CodeWarrior debugger provides two options to accomplish this. The first option is to launch the code on all cores simultaneously with one mouse click. The second option is to launch the code successively on one core after the other. The requirements of the application being debugged determine which option should be used.

1.1 Option One: Starting All Cores Simultaneously

To be able to start and debug a number of cores at the same time, a launch group must be defined. A *launch group* specifies which cores to start and the debugger settings that are used during their execution.

1.1.1 Create a Launch Group

To create a launch group:

1. Open the Debug Configuration dialog by clicking on the arrow next to the green bug icon and selecting **Debug Configuration**, as shown in Figure 1.

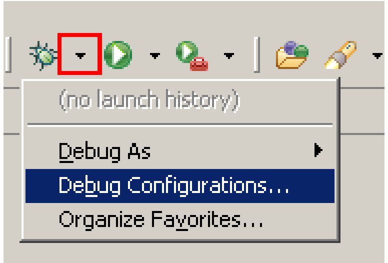
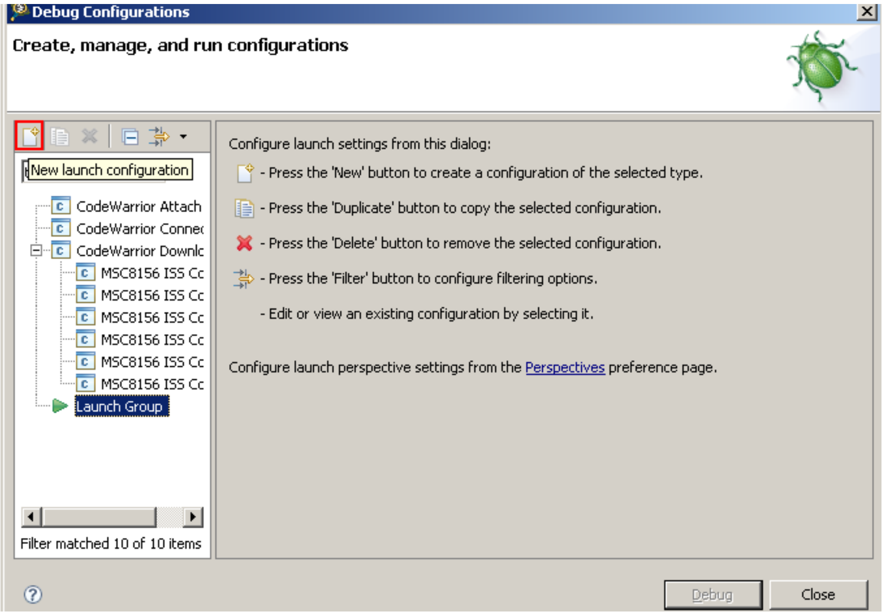


Figure 1. Opening the Debug Configuration dialog

2. In the debug configuration dialog, select **Launch Group** and then click on **New Launch Configuration** icon, as shown in Figure 2.



Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.

- Press the 'Duplicate' button to copy the selected configuration.

- Press the 'Delete' button to remove the selected configuration.

- Press the 'Filter' button to configure filtering options.

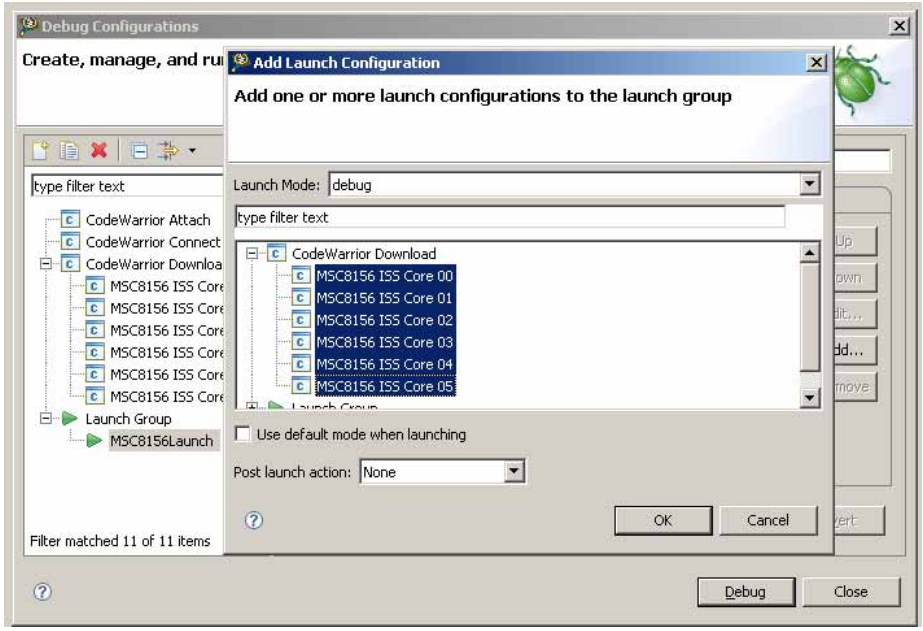
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

Figure 2. Creating a Launch Group

3. Specify a name for the launch group in the **Name** option (for this example, the `MSC8156Launch` was used) and click **Apply**.
4. Click **Add**.

5. In the **Add Launch Configuration** dialog that appears, expand the **CodeWarrior Download** group. For a core to execute code, it must have a launch configuration assigned to it. Each launch configuration specifies the debugger settings used while controlling the designated core.
6. Select all of the launch configurations to be associated with this launch group. For example, to have the launch group manage all six processor cores, choose the launch configurations `MSC8156 ISS Core 00` through `MSC8156 ISS Core 05`. See Figure 3.



Add one or more launch configurations to the launch group

Launch Mode: debug

□ Use default mode when launching

Post launch action: None

Filter matched 11 of 11 items

Figure 3. Selecting the Launch Configurations that Belong to a Launch Group

7. Choose a post launch action in the **Post launch action** option, if necessary.

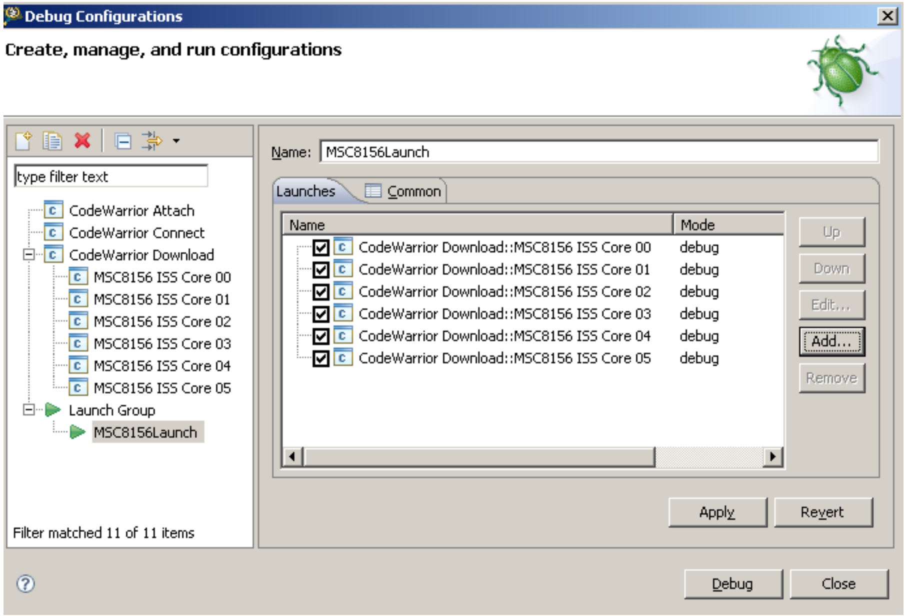
   Table 1 summarizes the choices of actions that the debugger can take after it starts the launch configuration for each core.

Table 1. Summary of Post Launch Actions

Option	Description
None	The debugger immediately moves on to launch the next launch configuration. This is the recommended settings in most of the cases.
Wait until terminated	The debugger waits indefinitely until the debug session spawned by the last launch terminates, and then it moves on to the next launch configuration.
Delay	The debugger waits for specified number of seconds before moving on to the next launch configuration.

8. Click **OK**.

   A list of the launch configurations associated with the launch group appears, as show in Figure 4.



Name: MSC8156Launch

Name	Mode
☑ CodeWarrior Download::MSC8156 ISS Core 00	debug
☑ CodeWarrior Download::MSC8156 ISS Core 01	debug
☑ CodeWarrior Download::MSC8156 ISS Core 02	debug
☑ CodeWarrior Download::MSC8156 ISS Core 03	debug
☑ CodeWarrior Download::MSC8156 ISS Core 04	debug
☑ CodeWarrior Download::MSC8156 ISS Core 05	debug

Figure 4. Launch Configurations That Are a Part of the MSC8156Launch Group

NOTE

Make sure that the core that manages any shared code (typically core 0) loads first. This is necessary because the shared code is linked to the core 0 image. Since the startup code and run time library code are shared among all the cores, if core 0 does not load first, none of the other cores can execute any startup code and reach their respective `main()` functions.

9. Click **Apply** to use the settings, then **Close** to save them and dismiss the Debug Configuration dialog.

1.1.2 Save the Launch Group's Configuration File

By default, the steps in section 1.1.1 make a `.launch` file in the directory

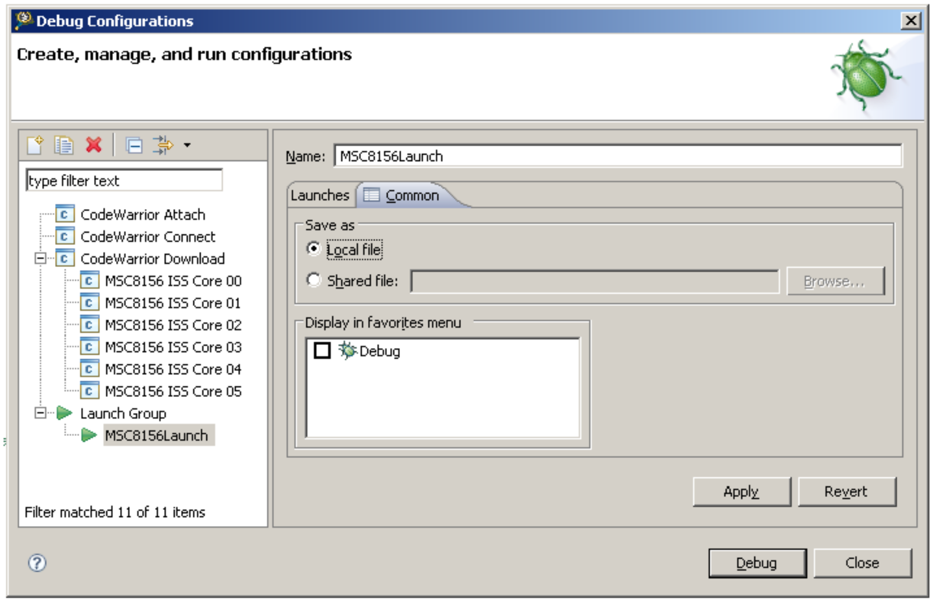
`{workspace}\.metadata\.plugins\org.eclipse.debug.core\.launches`

This enables the launch group's configuration to be available each time that the workspace loads into the CodeWarrior IDE.

In order to make it easier to package the launch group together with the application, the group configuration file can be saved along with the launch configuration files for each core in the `{Project}Debug_Settings\` directory.

This is done as follows:

1. Open the Debug Configuration dialog by clicking on the arrow next to the green bug icon and selecting **Debug Configuration**.
2. In the **Debug Configuration** dialog, expand the **Launch Group** folder and select your launch group.
3. Switch to the **Common** tab, as shown in Figure 5.



Debug Configurations

Create, manage, and run configurations

type filter text

- CodeWarrior Attach
- CodeWarrior Connect
- CodeWarrior Download
  - MSC8156 ISS Core 00
  - MSC8156 ISS Core 01
  - MSC8156 ISS Core 02
  - MSC8156 ISS Core 03
  - MSC8156 ISS Core 04
  - MSC8156 ISS Core 05
- Launch Group
  - MSC8156Launch

Filter matched 11 of 11 items

Name: MSC8156Launch

Launches | Common

Save as

● Local file

○ Shared file:

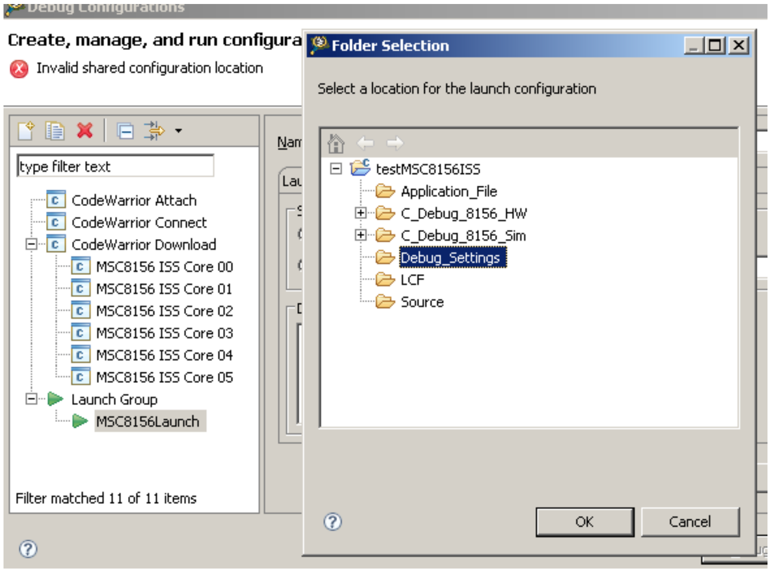
Display in favorites menu

☐ Debug

Figure 5. Using the Common Tab to Save the Launch Group Settings

4. Choose the **Shared file** option.

5. Click **Browse** and in the **Folder Selection** dialog and navigate to the project's `Debug Settings` folder. See Figure 6.



Create, manage, and run configura

Invalid shared configuration location

Select a location for the launch configuration

Filter matched 11 of 11 items

Figure 6. Selecting the Folder to Save the Launch Settings

6. Click **OK** to dismiss the **Folder Selection** dialog; then click **Apply** to save the settings.
7. Click **Close** to close the **Debug Configuration** dialog.

   From now on, the `.launch` file for the launch group is stored in the `{Project}\Debug Settings` folder.

NOTE

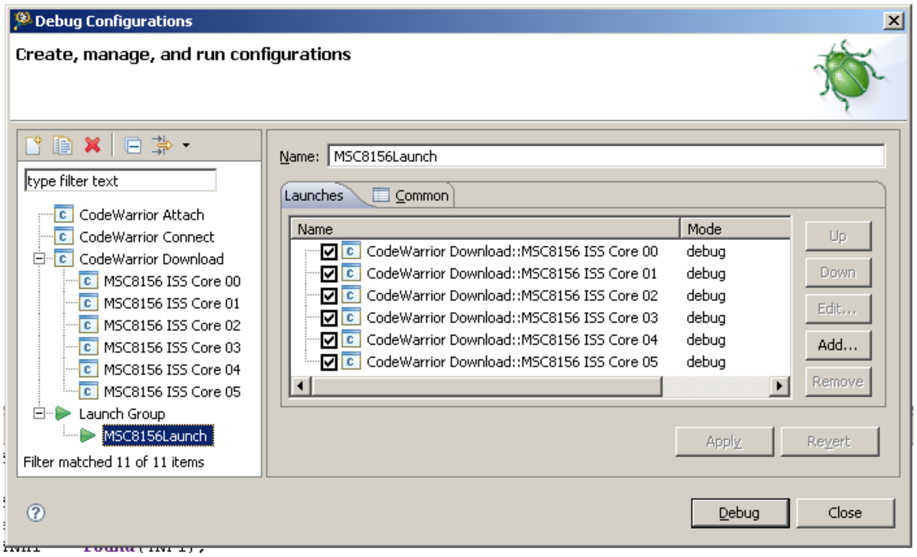
If you have already created the launch group in the `{workspace}\.metadata\.plugins\org.eclipse.debug.core\.launches` directory, it is good practice to physically delete the file from this location and restart the CodeWarrior IDE. Otherwise you will end up with two Launch groups with the same name in the **Debug Configuration** dialog.

1.1.3 Debugging With the Launch Group

To start debugging using a previously saved launch group:

1. Open the Debug Configuration dialog by clicking on the arrow next to the green bug icon and selecting **Debug Configuration**.

2. In the **Debug Configuration** dialog, expand the **Launch Group** folder and select the launch group you want to debug as shown in Figure 7.



Debug Configurations

Create, manage, and run configurations

type filter text

- CodeWarrior Attach
- CodeWarrior Connect
- CodeWarrior Download
  - MSC8156 ISS Core 00
  - MSC8156 ISS Core 01
  - MSC8156 ISS Core 02
  - MSC8156 ISS Core 03
  - MSC8156 ISS Core 04
  - MSC8156 ISS Core 05
- Launch Group
  - MSC8156Launch

Filter matched 11 of 11 items

Name: MSC8156Launch

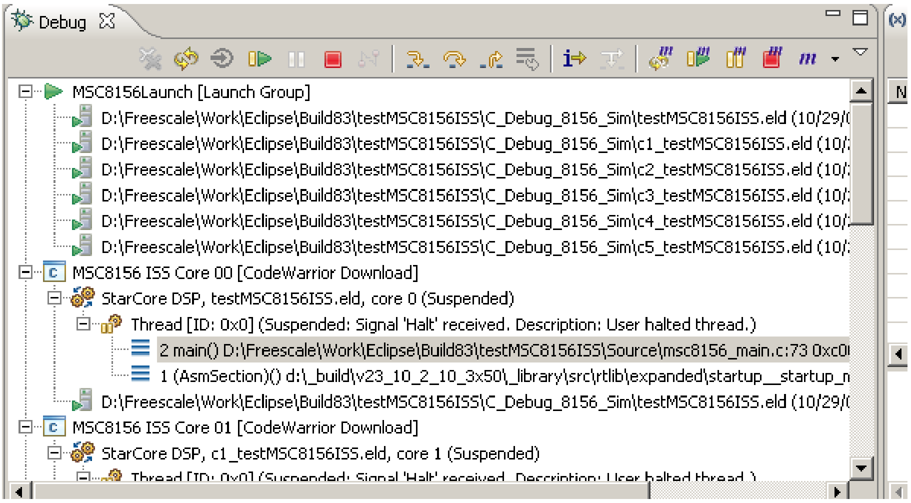
Launches | Common

	Name	Mode
☑	CodeWarrior Download::MSC8156 ISS Core 00	debug
☑	CodeWarrior Download::MSC8156 ISS Core 01	debug
☑	CodeWarrior Download::MSC8156 ISS Core 02	debug
☑	CodeWarrior Download::MSC8156 ISS Core 03	debug
☑	CodeWarrior Download::MSC8156 ISS Core 04	debug
☑	CodeWarrior Download::MSC8156 ISS Core 05	debug

Figure 7. Selecting a Launch Group for Debugging

3. Click **Debug** to start the multicore debug session.

   The debugger starts a multicore debug session using the specified launch group(s). Each core executes any startup code and then halts at its `main()` function, unless configured otherwise. For this example, the **Debug** view in Figure 8 shows the six cores halted and awaiting new debugger commands.



```
MSC8156Launch [Launch Group]
    D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\C_Debug_8156_Sim\testMSC8156ISS.eld (10/29/(
    D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\C_Debug_8156_Sim\c1_testMSC8156ISS.eld (10/:
    D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\C_Debug_8156_Sim\c2_testMSC8156ISS.eld (10/:
    D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\C_Debug_8156_Sim\c3_testMSC8156ISS.eld (10/:
    D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\C_Debug_8156_Sim\c4_testMSC8156ISS.eld (10/:
    D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\C_Debug_8156_Sim\c5_testMSC8156ISS.eld (10/:
MSC8156 ISS Core 00 [CodeWarrior Download]
    StarCore DSP, testMSC8156ISS.eld, core 0 (Suspended)
        Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
            2 main() D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\Source\msc8156_main.c:73 0xc0(
            1 (AsmSection)() d:\_build\v23_10_2_10_3x50\_library\src\rtlib\expanded\startup__startup_n
    D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\C_Debug_8156_Sim\testMSC8156ISS.eld (10/29/(
MSC8156 ISS Core 01 [CodeWarrior Download]
    StarCore DSP, c1_testMSC8156ISS.eld, core 1 (Suspended)
        Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
```

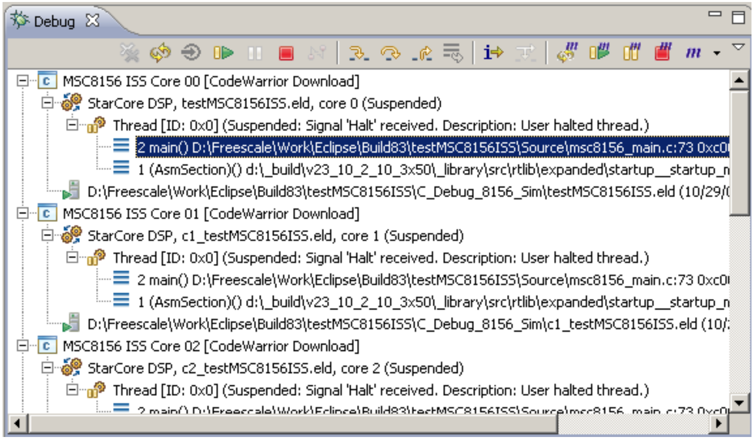
Figure 8. A Multicore Debug Session as Started by the Launch Group

1.2 Option Two: Launch Cores One by One

Certain situations might require the cores to be loaded separately, particularly if something needs to happen between the loadings of the different cores. To do so, follow these steps:

1. Open the Debug Configuration dialog by clicking on the arrow next to the green bug icon and choosing **Debug Configuration**.
2. Expand the **CodeWarrior Download** group.
3. Select the `Core 00` launch configuration and click **Debug**.
4. Once the download for core 0 completes, open the **Debug Configuration** dialog, select the `Core 01` launch configuration, then click **Debug**.
5. Once the download for core 1 completes, open the **Debug Configuration** dialog, select the `Core 02` launch configuration, then click **Debug**.
6. Once the download for core 2 completes, open the **Debug Configuration** dialog, select the `Core 03` launch configuration, then click **Debug**.
7. Once the download for core 3 completes, open the **Debug Configuration** dialog, select the `Core 04` launch configuration, then click **Debug**.
8. Once the download for core 4 completes, open the **Debug Configuration** dialog, select the `Core 05` Launch Configuration, then click **Debug**.

   After all of the launch configurations are started, the multicore debug session the **Debug** view resembles Figure 9.



```
MSC8156 ISS Core 00 [CodeWarrior Download]
  StarCore DSP, testMSC8156ISS.eld, core 0 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\Source\msc8156_main.c:73 0xc0
      1 (AsmSection)() d:\_build\v23_10_2_10_3x50\_library\src\rtlib\expanded\startup__startup_n
  D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\C_Debug_8156_Sim\testMSC8156ISS.eld (10/29/
MSC8156 ISS Core 01 [CodeWarrior Download]
  StarCore DSP, c1_testMSC8156ISS.eld, core 1 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\Source\msc8156_main.c:73 0xc0
      1 (AsmSection)() d:\_build\v23_10_2_10_3x50\_library\src\rtlib\expanded\startup__startup_n
  D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\C_Debug_8156_Sim\c1_testMSC8156ISS.eld (10/
MSC8156 ISS Core 02 [CodeWarrior Download]
  StarCore DSP, c2_testMSC8156ISS.eld, core 2 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() D:\Freescale\Work\Eclipse\Build83\testMSC8156ISS\Source\msc8156_main.c:73 0xc0
```

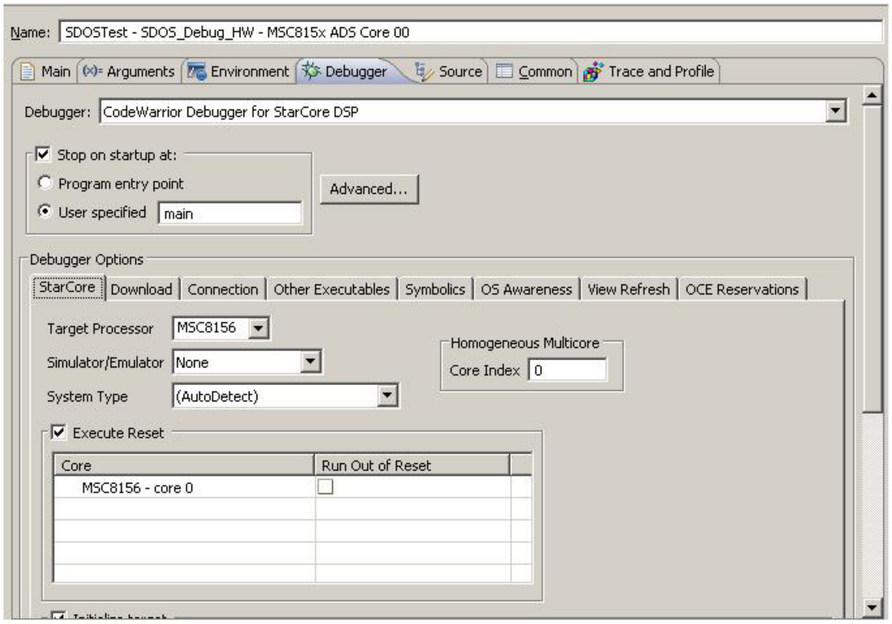
Figure 9. A MultiCore Debug Session in Progress

1.3 Troubleshooting

If problems occur with the multicore sessions, check the following points:

- Make sure that the debug settings on all of the cores are identical. To check these, open the **Debugger Settings** dialog, click on the **Debugger** tab, and study the options in the **Debugger Options** group for differences.

- — For the **StarCore** tab shown in Figure 10, the **Target Processor**, **Simulator/Emulator** and **System Type** options should be identical for all cores. The value specified in the **Core Index** option should be different for each core, however.



Name: SDOSTest - SDOS_Debug_HW - MSC815x ADS Core 00

Debugger: CodeWarrior Debugger for StarCore DSP

☑ Stop on startup at:

○ Program entry point

● User specified main

Debugger Options

Target Processor MSC8156

Simulator/Emulator None

System Type (AutoDetect)

Homogeneous Multicore

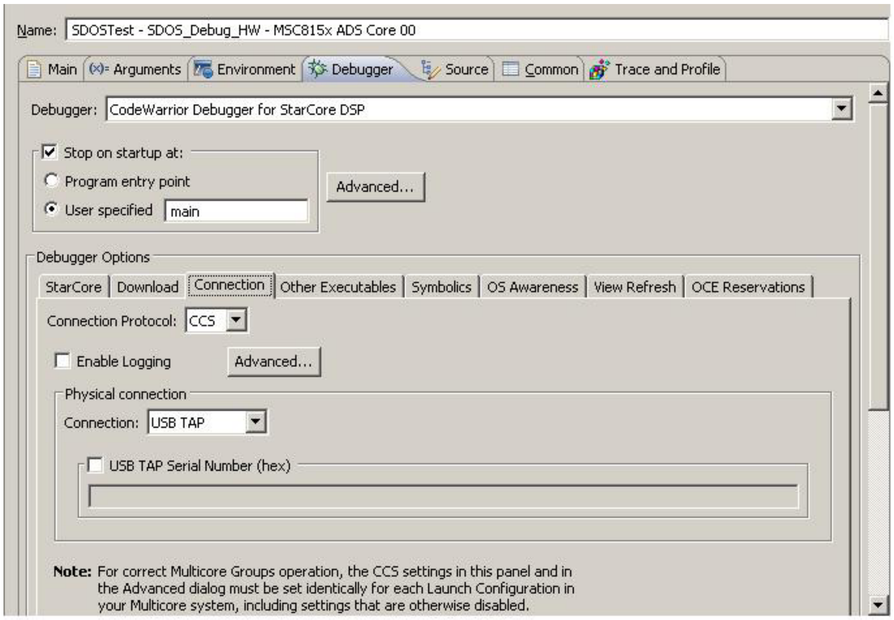
Core Index 0

☑ Execute Reset

Core	Run Out of Reset
MSC8156 - core 0	☐

Figure 10. StarCore Tab Options

- — For the **Connection** tab shown in Figure 11, the **Connection Protocol**, **Physical Connection**, and **CCS Advanced Settings** should be identical for all cores. You can examine the **CCS Advance Settings** by clicking on the **Advanced** button within the **Connection** tab.



Name: SDOSTest - SDOS_Debug_HW - MSC815x ADS Core 00

Debugger: CodeWarrior Debugger for StarCore DSP

☑ Stop on startup at:
○ Program entry point
● User specified main

Connection Protocol: CCS

□ Enable Logging

Physical connection
Connection: USB TAP
□ USB TAP Serial Number (hex)

Note: For correct Multicore Groups operation, the CCS settings in this panel and in the Advanced dialog must be set identically for each Launch Configuration in your Multicore system, including settings that are otherwise disabled.

Figure 11. Connection Tab Options

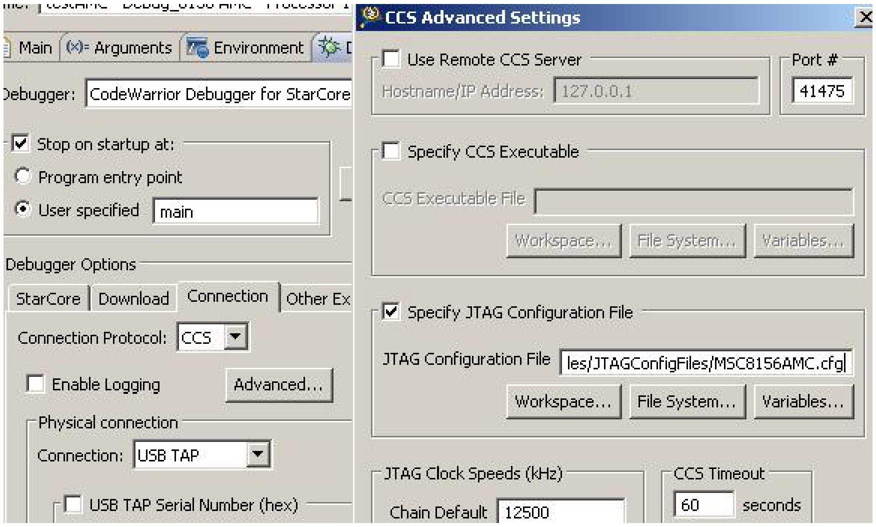
- Always load core 0 (where all shared sections reside) first. The loading sequence for the other cores does not matter, unless the application requires a specific sequence to start the different cores.
- In the **StarCore** tab, the **Execute Reset** option must be checked for core 0 only. It must be unchecked for all other cores. In addition, the **Initialize target** option and the **Use Memory configuration file** option must be checked for core 0 only. It should be unchecked for all other cores.

NOTE

When debugging a multi-device system:

- The **Execute Reset** option must be set for core 0 on first device (processor) only.
- The **Initialize Target** option must be checked, and the **Use Memory configuration file** option must be set for core 0 on each device. (That is, these options should be set for core 0, core 6, and core 12 on a MSC8156AMC board).

- For all cores, the JTAG settings such as the configuration file, CCS executable, JTAG clock speed, and CCS timeout *must* be identical in the **CCS Advanced Settings** dialog, as shown in Figure 12.



Debugger: CodeWarrior Debugger for StarCore

☑ Stop on startup at:

○ Program entry point

● User specified main

Debugger Options

Connection Protocol: CCS

☐ Enable Logging

Physical connection

Connection: USB TAP

☐ USB TAP Serial Number (hex)

CCS Advanced Settings

☐ Use Remote CCS Server

Hostname/IP Address: 127.0.0.1

Port # 41475

☐ Specify CCS Executable

CCS Executable File

☑ Specify JTAG Configuration File

JTAG Configuration File les/JTAGConfigFiles/MSC8156AMC.cfg

JTAG Clock Speeds (kHz)

Chain Default 12500

CCS Timeout

60 seconds

Figure 12. CCS Advanced Settings Dialog

NOTE

For more information on how to configure the hardware and the CodeWarrior IDE's settings for debugging a board with multiple devices, consult the application note, AN3908, "A Guide to Configuring Multiple MSC8156 Devices on a Single JTAG Chain Using CodeWarrior Development Studio for StarCore DSP Architectures v10.0".

2 Controlling Execution

The CodeWarrior debugger provides a number of run-control commands that can start, step, stop, and restart a program. Many of these commands can be applied to

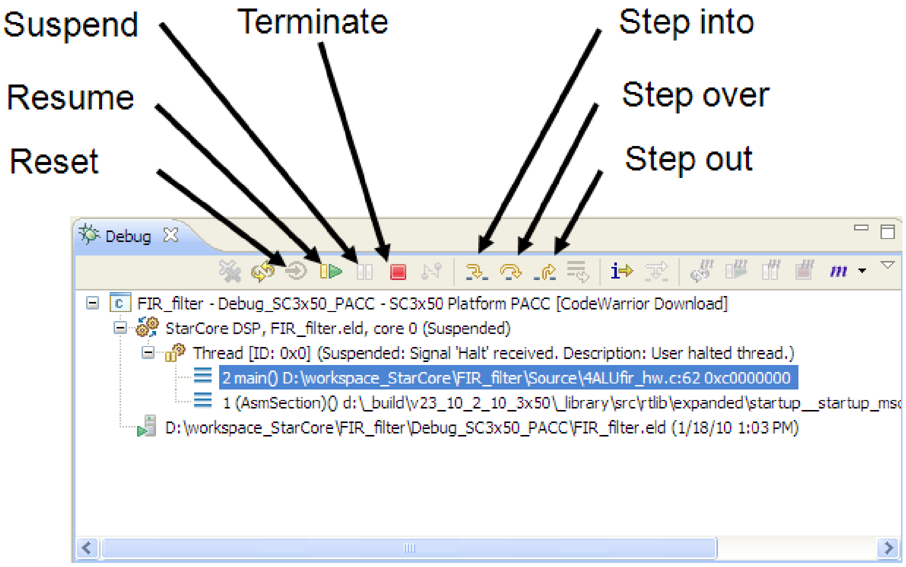
- One core
- All cores
- A specific set of cores

NOTE

The stepping command can only be applied to a single core, while the run (resume), stop (suspend), terminate (kill) commands can apply to multiple cores.

2.1 Controlling Execution in One Core

To control the code execution for a specific core, select the core you want to control in the **Debug** view by clicking on one of the lines within its thread. Next, select the control action. Control actions can be specified by clicking on one of the standard icons for **Restart**, **Resume** (Run), **Suspend** (Stop), **Terminate** (Kill), **Step into**, **Step over** and **Step out** (Step Return) in the view's toolbar, as shown in Figure 13.



FIR_filter - Debug_SC3x50_PACC - SC3x50 Platform PACC [CodeWarrior Download]
StarCore DSP, FIR_filter.eld, core 0 (Suspended)
Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
2 main() D:\workspace_StarCore\FIR_filter\Source\4ALUfir_hw.c:62 0xc0000000
1 (AsmSection)() d:_build\v23_10_2_10_3x50_library\src\rtlib\expanded\startup__startup_msc
D:\workspace_StarCore\FIR_filter\Debug_SC3x50_PACC\FIR_filter.eld (1/18/10 1:03 PM)

Figure 13. The Code Execution Controls

Alternatively, use the **Run** menu choices of **Restart**, **Resume**, **Suspend**, **Terminate**, **Step Into**, **Step Over** or **Step Return** to issue control commands.

2.2 Controlling Execution on Multiple Cores

In order to apply a run-control command to a specific set of cores, a multicore group must be defined first. There is no need to define a multicore group if you intend to debug on all the cores. Per default, multicore run-control commands apply to all the loaded cores.

2.2.1 Define a Multicore Group

To create a multicore group:

1. Enter the Debug perspective and start a multicore debugging session.

2. At the top of the **Debug** view, click on the blue **M** to expand the **Multicore Groups** menu. Select **Edit Multicore Groups**, as shown in Figure 14.

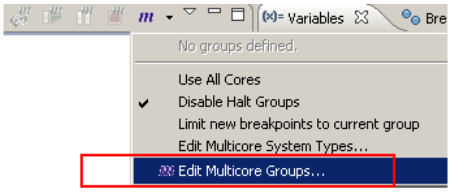


Figure 14. Using the MultiCore Groups Menu to Create a Multicore Group

3. In the **Multicore Groups** dialog, click **New**.
4. In the **New Multicore Group** dialog, select the target device (`MSC8156` in this example). Then click **OK**, as Figure 15 shows.

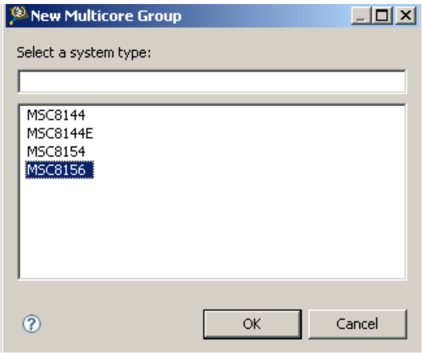
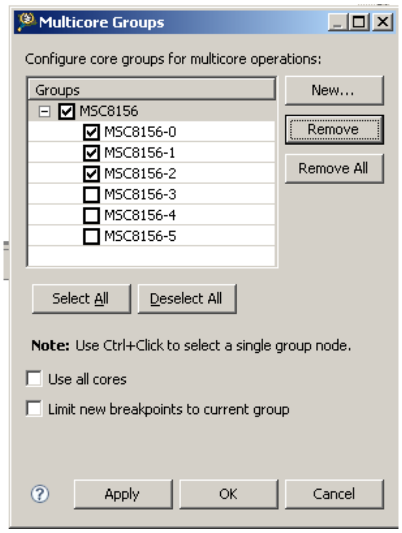


Figure 15. Choosing the Target Processor for a Multicore Group

5. In the **Multicore Groups** dialog, check those cores that should be part of the group, as shown in Figure 16.



Multicore Groups

Configure core groups for multicore operations:

Groups
- ☑ MSC8156
  - ☑ MSC8156-0
  - ☑ MSC8156-1
  - ☑ MSC8156-2
  - ☐ MSC8156-3
  - ☐ MSC8156-4
  - ☐ MSC8156-5

Note: Use Ctrl+Click to select a single group node.

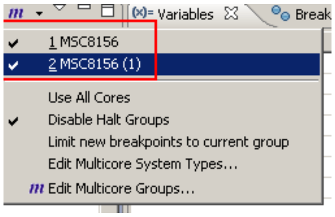
☐ Use all cores

☐ Limit new breakpoints to current group

Figure 16. Placing Cores in the Multicore Group

6. Click **Apply**.
7. To define additional multicore groups, repeat steps 4, 5, and 6 for each new group.
8. Click **OK** when done.

Clicking on the arrow adjacent to the **M** icon displays the **Multicore Group** menu choices and the names of available multicore groups, as shown in Figure 17.



✓ 1 MSC8156

✓ 2 MSC8156 (1)

Use All Cores

✓ Disable Halt Groups

Limit new breakpoints to current group

Edit Multicore System Types...

Edit Multicore Groups...

Figure 17. The Multicore Groups Menu Displaying a List of Available Multicore Groups

2.3 Multicore Control Commands

To control the code execution on multiple cores:

1. Select which multicore group to apply the command to. To do this, click on the arrow next to the blue **M** icon.
   - To apply the command to all cores currently in debug mode, make sure the menu choice **Use All Cores** is checked.
   - To apply the command to one or several multicore groups, make sure the menu choices for these groups are checked.

2. Once you have selected the set(s) of cores to apply the command to, click on one of multicore control icons in the **Debug** view's toolbar to issue a **Multicore Restart**, **Multicore Resume** (Run), **Multicore Suspend** (Stop) and **Multicore Terminate** (Kill) command. These icons are shown in Figure 18.

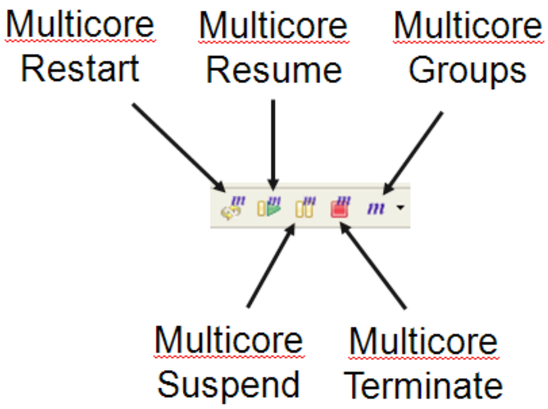
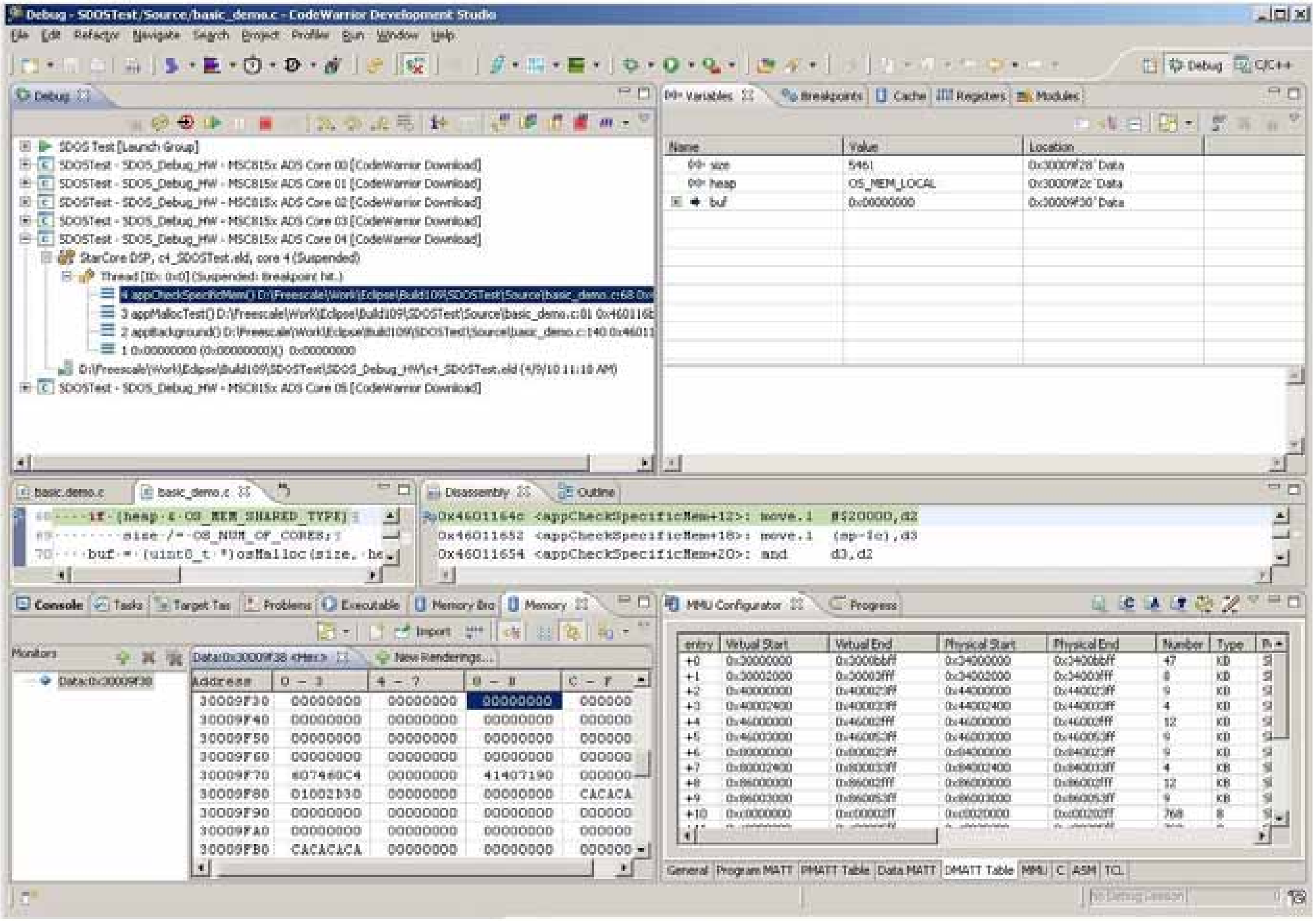


Figure 18. Multicore Code Execution Controls

Alternatively, use the **Run** menu choices of **Multicore Restart**, **Multicore Resume**, **Multicore Suspend** and **Multicore Terminate** to issue run-control commands.

3 Displaying the Context of a Specific Core

When debugging a multicore application, the content of the debugger views always reflect the context of the core that has focus in the **Debug** view, as shown in Figure 19.



SDOS Test [Launch Group]
SDOSTest - SDOS_Debug_HW - MSC815x ADS Core 00 [CodeWarrior Download]
SDOSTest - SDOS_Debug_HW - MSC815x ADS Core 01 [CodeWarrior Download]
SDOSTest - SDOS_Debug_HW - MSC815x ADS Core 02 [CodeWarrior Download]
SDOSTest - SDOS_Debug_HW - MSC815x ADS Core 03 [CodeWarrior Download]
SDOSTest - SDOS_Debug_HW - MSC815x ADS Core 04 [CodeWarrior Download]
StarCore DSP, c4_SDOSTest.eld, core 4 (Suspended)
Thread [ID: 0x0] (Suspended: Breakpoint hit.)
4 appCheckSpecificMem() D:\Freescale\Work\Eclipse\Build109\SDOSTest\Source\basic_demo.c:68 0x
3 appMallocTest() D:\Freescale\Work\Eclipse\Build109\SDOSTest\Source\basic_demo.c:81 0x460116
2 appBackground() D:\Freescale\Work\Eclipse\Build109\SDOSTest\Source\basic_demo.c:140 0x46011
1 0x00000000 (0x00000000)() 0x00000000
D:\Freescale\Work\Eclipse\Build109\SDOSTest\SDOS_Debug_HW\c4_SDOSTest.eld (4/9/10 11:18 AM)
SDOSTest - SDOS_Debug_HW - MSC815x ADS Core 05 [CodeWarrior Download]

Name	Value	Location
size	5461	0x30009f28`Data
heap	OS_MEM_LOCAL	0x30009f2c`Data
buf	0x00000000	0x30009f30`Data

```
    if (heap & OS_MEM_SHARED_TYPE)
        size /= OS_NUM_OF_CORES;
    buf = (uint8_t *)osMalloc(size, he
```

```
0x4601164c <appCheckSpecificMem+12>: move.l   #$20000,d2
0x46011652 <appCheckSpecificMem+18>: move.l   (sp-Ic),d3
0x46011654 <appCheckSpecificMem+20>: and      d3,d2
```

```
Address   0 - 3     4 - 7     8 - B     C - F
30009F30  00000000  00000000  00000000  000000
30009F40  00000000  00000000  00000000  000000
30009F50  00000000  00000000  00000000  000000
30009F60  00000000  00000000  00000000  000000
30009F70  607460C4  00000000  41407190  000000
30009F80  01002D30  00000000  00000000  CACACA
30009F90  00000000  00000000  00000000  000000
30009FA0  00000000  00000000  00000000  000000
30009FB0  CACACACA  00000000  00000000  000000
```

entry	Virtual Start	Virtual End	Physical Start	Physical End	Number	Type	P
+0	0x30000000	0x3000bbff	0x34000000	0x3400bbff	47	KB	S
+1	0x30002000	0x30003fff	0x34002000	0x34003fff	8	KB	S
+2	0x40000000	0x400023ff	0x44000000	0x440023ff	9	KB	S
+3	0x40002400	0x400033ff	0x44002400	0x440033ff	4	KB	S
+4	0x46000000	0x46002fff	0x46000000	0x46002fff	12	KB	S
+5	0x46003000	0x460053ff	0x46003000	0x460053ff	9	KB	S
+6	0x80000000	0x800023ff	0x84000000	0x840023ff	9	KB	S
+7	0x80002400	0x800033ff	0x84002400	0x840033ff	4	KB	S
+8	0x86000000	0x86002fff	0x86000000	0x86002fff	12	KB	S
+9	0x86003000	0x860053ff	0x86003000	0x860053ff	9	KB	S
+10	0xc0000000	0xc00002ff	0xc0020000	0xc00202ff	768	B	S

Figure 19. A Multicore Debug Session in Progress

In the figure, the debugger displays the context for core 4, because it is chosen in the **Debug** View. Therefore, the variables displayed in the **Variables** view are those for core 4. Likewise, the **Memory Dump** view shows content of virtual memory for core 4, as does the **MMU Configurator** view for the MMU configuration.

To display the context of a different core:

1. In the **Debug** view, expand the launch configuration of the desired core.
2. Click on one of the lines within its thread in the expanded configuration display.

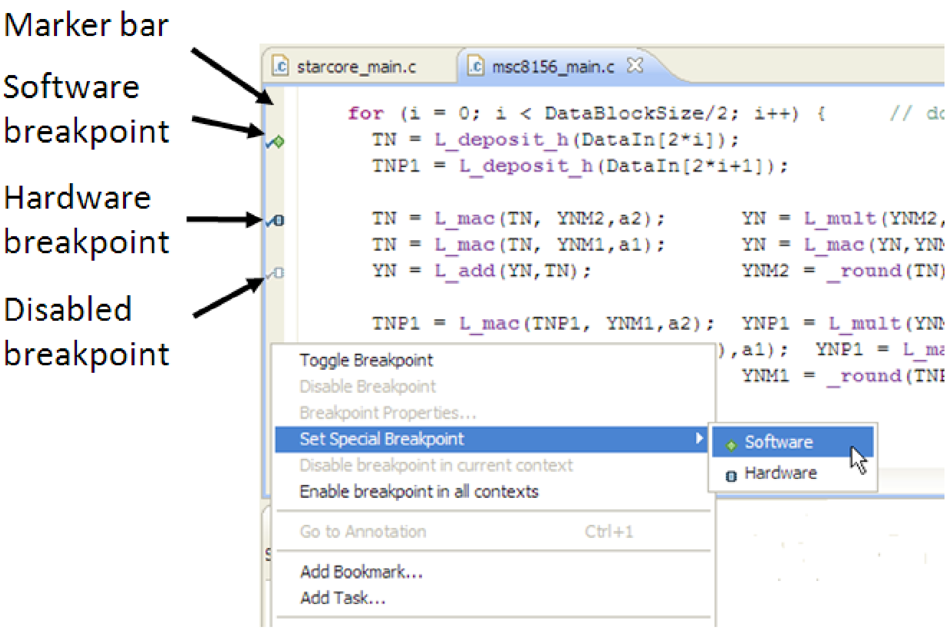
4 Breakpoints

The CodeWarrior debugger allows you to set two kinds of breakpoints:

- Software breakpoints: The debugger inserts a “debug” instruction at an instruction-aligned address in memory that represents the source code line where a breakpoint is desired.
- Hardware breakpoints: The debugger uses the on-chip emulator (OCE) module, which is a dedicated hardware block that manages breakpoints and their trigger conditions.

In order to set a software or hardware breakpoint:

1. Right-click in the marker bar area on the left side of an editor, beside the source line where the breakpoint should be set.
2. In the drop down menu that appears, select **Set Special Breakpoint** and then select **Software Breakpoint** or **Hardware Breakpoint**, depending on the kind of breakpoint required. See Figure 20.



```
    for (i = 0; i < DataBlockSize/2; i++) {      // dc
      TN = L_deposit_h(DataIn[2*i]);
      TNP1 = L_deposit_h(DataIn[2*i+1]);

      TN = L_mac(TN, YNM2,a2);        YN = L_mult(YNM2,
      TN = L_mac(TN, YNM1,a1);        YN = L_mac(YN,YNM
      YN = L_add(YN,TN);              YNM2 = _round(TN)

      TNP1 = L_mac(TNP1, YNM1,a2);  YNP1 = L_mult(YNM
                                    ),a1);  YNP1 = L_ma
                                      YNM1 = _round(TNE
```

Figure 20: Special Breakpoint Menu

When a breakpoint (software or hardware) is set in shared code, it is activated on all cores by default. That means an application stops in any core as soon as its code execution triggers the breakpoint condition.

4.1 Applying Breakpoints to Selected Cores

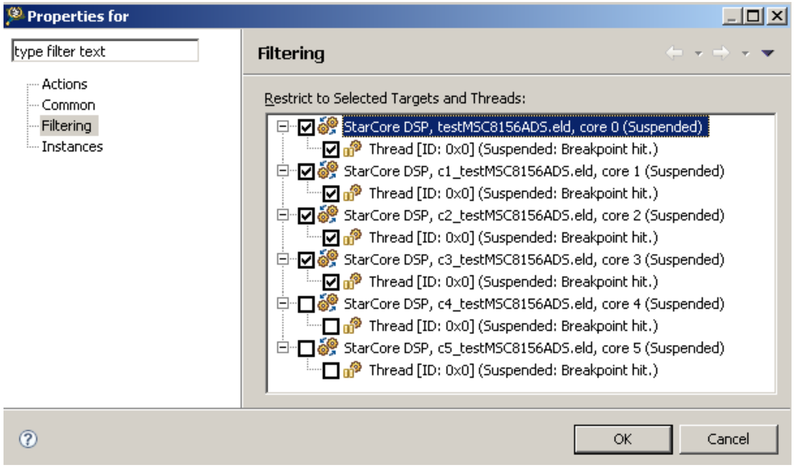
The CodeWarrior debugger provides the ability to define the breakpoint on only a set of specific cores. This is done as follows:

1. Set your breakpoint the usual way.

2. Right-click on the breakpoint icon in the marker bar and select **Breakpoint Properties**.

   The **Breakpoint Properties** dialog opens.

3. Switch to the **Filtering Panel** by clicking on the **Filtering** option, as shown in Figure 21.



Restrict to Selected Targets and Threads:

- ☑ StarCore DSP, testMSC8156ADS.eld, core 0 (Suspended)
  - ☑ Thread [ID: 0x0] (Suspended: Breakpoint hit.)
- ☑ StarCore DSP, c1_testMSC8156ADS.eld, core 1 (Suspended)
  - ☑ Thread [ID: 0x0] (Suspended: Breakpoint hit.)
- ☑ StarCore DSP, c2_testMSC8156ADS.eld, core 2 (Suspended)
  - ☑ Thread [ID: 0x0] (Suspended: Breakpoint hit.)
- ☑ StarCore DSP, c3_testMSC8156ADS.eld, core 3 (Suspended)
  - ☑ Thread [ID: 0x0] (Suspended: Breakpoint hit.)
- □ StarCore DSP, c4_testMSC8156ADS.eld, core 4 (Suspended)
  - □ Thread [ID: 0x0] (Suspended: Breakpoint hit.)
- □ StarCore DSP, c5_testMSC8156ADS.eld, core 5 (Suspended)
  - □ Thread [ID: 0x0] (Suspended: Breakpoint hit.)

Figure 21. Specifying the Cores That Respond to a Breakpoint

4. Uncheck the cores on which the breakpoint will not apply.
5. Click **OK**.

   From now on, the breakpoint affects only to the selected cores. That is, it will be ignored by the unchecked cores.

NOTE

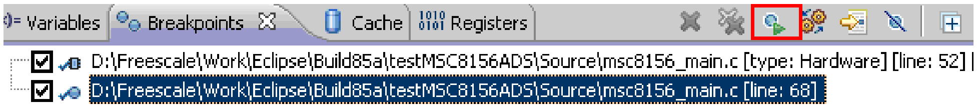
A software breakpoint applied to specific cores that happens to be set in shared code can break the application’s real-time execution. This is because a software breakpoint in shared code always temporarily halts every core that executes it. The debugger must check the core’s ID and if it is set as disabled, the debugger resumes the core’s execution. The overhead of this check has an impact on runtime behavior and the caches if the processor is not running in cache coherency mode.

4.2 Applying a Breakpoint to the Current Core Only

During the debugging of an application, all future breakpoints can be limited to the current debugging context. This is done as follows:

1. Debug the project.
2. Set the focus for the core to be debugged by clicking on one of the lines within its thread in the **Debug** view.
3. Switch to the **Breakpoints** view.

4. Click the **Limit New Breakpoints to Active Debug Context** icon from the Breakpoint view, as show in Figure 22.



☑ D:\Freescale\Work\Eclipse\Build85a\testMSC8156ADS\Source\msc8156_main.c [type: Hardware] [line: 52] |

☑ D:\Freescale\Work\Eclipse\Build85a\testMSC8156ADS\Source\msc8156_main.c [line: 68]

Figure 22. Limiting the Debugging Context of Breakpoints

From now on, all breakpoints are set only in the core being debugged.

Click the same icon in the **Breakpoints** view to return breakpoint activity to its normal behavior. (That is, a breakpoint is set in all of the cores).

NOTE

A software breakpoint applied to specific cores that happens to be set in shared code can break the application’s real-time execution. This is because a software breakpoint in shared code always halts every core that executes it. The debugger must check the core’s ID and if it is set as disabled, the debugger resumes the core’s execution. The overhead of this check has an impact on runtime behavior and the caches if the processor is not running in cache coherency mode.

5 Watchpoints

The CodeWarrior debugger’s handling of watchpoints is similar to breakpoints handling. There is one restriction though: the debugger only allows hardware watchpoints. The tools do not support software watchpoints. Therefore, when watchpoint is set, it applies to all of the cores.

Filtering can be specified in the same way as for breakpoints to specify those cores on which the watchpoint should apply, or to limit a watchpoint to the current core. Refer to sections 4.1 and 4.2 for more information.

6 Command Line Interface

The CodeWarrior debugger provides a command line interface through its Debugger Shell view. This view is a console where you can enter debugger and Tcl commands that control the execution of the cores. There are also commands that display or modify the contents of memory. Tcl scripts can be used to set up complex debugging scenarios or to automate testing.

As the Debugger Shell is not a default view, it must be started manually. This can be done in two ways:

- From the C/C++ Perspective: Choose **Window** > **Other** > **Debug** > **Debugger Shell**.
- From the Debug Perspective: Choose **Window** > **Show View** > **Debugger Shell**.

6.1 Starting a Multicore Debugging Session

The Debugger Shell's command line can be used to begin a multicore debug session where all of the cores start at once, or where each core starts separately. For example, given a launch group called `MyAppMSC8156.launch`, a multicore debug session can be started using the following command:

```
debug MyAppMSC8156
```

Alternatively, each launch configuration can be started separately, as follows:

```
debug “testMSC8156 - C_Debug_8156_HW - MSC8156 ADS Core 00”
debug “testMSC8156 - C_Debug_8156_HW - MSC8156 ADS Core 01”
debug “testMSC8156 - C_Debug_8156_HW - MSC8156 ADS Core 02”
debug “testMSC8156 - C_Debug_8156_HW - MSC8156 ADS Core 03”
debug “testMSC8156 - C_Debug_8156_HW - MSC8156 ADS Core 04”
debug “testMSC8156 - C_Debug_8156_HW - MSC8156 ADS Core 05”
```

Starting each core separately is useful when other commands must be issued before starting the next core with a `debug` command.

6.2 Running/Stopping Multiple Cores

Like the GUI version of the multicore debugging interface, to use multicore commands with the Debugger Shell, a multicore group must be defined first. This is necessary only if you want to apply multicore run control commands to a subset of the cores only. Per default the multicore commands apply to all the cores being loaded.

NOTE

All Tcl commands involved with multicore debugging have the prefix `mc::`.

6.2.1 Defining a Multicore Group

To determine which architectures/types are supported by the Debugger Shell, enter following command:

```
mc::type
```

The command `mc::group` allows the definition of multicore groups for a specific architecture. For example, to create a new multicore group for a StarCore MSC8156 processor architecture, use the following command:

```
mc::group new MSC8156
```

This command can be used to create multiple groups. After a group is created, entering the command `mc::group` without arguments displays a list of currently defined groups:

```
mc::group
Index    Group
-----    --------------
  0        MSC8156
  0.0         MSC8156
  0.1         MSC8156
  0.2         MSC8156
  0.3         MSC8156
  0.4         MSC8156
  0.5         MSC8156
```

NOTE

> The list shows that a multicore group for a MSC8156 processor type has been assigned an index of 0 (the default). This index number is used in certain multicore commands to reference the group. Also, note that each core in the group has its own sub-index number. That is, core 0 has an index of 0.0, core 1 has an index of 0.1, and so on. The sub-index number provides a reference to a specific core in the group.

The command `mc:group rename` is used to rename the different groups and give them meaningful names. For example, to change the previously created group from `MSC8156` to `cores_0_3`, enter:

```
mc::group rename 0 “cores_0_3”
```

In the above command, the 0 refers to the index of the newly-created multicore group. You can obtain group’s index using the command `mc::group`.

At the end of a debugger script, it is good practice to delete all multicore groups previously created using the command `mc::group remove` or `mc::group removeall`. For instance, to remove the group generated above, enter the following command:

```
mc::group remove “cores_0_3”
```

6.2.2 Controlling Code Execution on Multiple Cores

To control code execution on several cores:

1. Select the multicore group and cores that the commands are applied to. Use the command `mc::group enable` to select the group and its cores.
   - To enable cores 0, 1, 2 and 3 in multicore group that has an index of 0, enter following commands:

     ```
     mc::group enable 0.0
     mc::group enable 0.1
     mc::group enable 0.2
     mc::group enable 0.3
     ```

     To enable all cores in group with index 0, enter following command:

```
mc::group enable 0
```

2. Once the set of cores is selected, every future multicore control commands issued affects only the chosen cores. For example:

   ```
   #Resume all enabled cores
   mc::go
   #Suspend all enabled cores
   mc::stop
   #Terminate all enabled cores
   mc::kill
   #Restart all enabled cores
   mc::restart
   ```

6.2.3 Controlling Code Execution on a Single Core

The standard commands `restart`, `go` (resume), `halt` (suspend) and `kill` (terminate) are used to control a single core. However, before using the command, the core must be selected. This is done using command `switchtarget`. Like the `mc::group` command, when `switchtarget` is issued without an argument, it displays a list of cores.

For example, suppose a test is to be performed on a core with an index of 1 only. This is done through following commands:

```
# List all targets currently in debug session
switchtarget
--on-the-fly-connection-1------------------------------------------
 Index Status   Thread   Process   CPU           Target
    *0 Stopped  0x0      0x8000    cpuSC100Big   testmsc8156.eld
     1 Stopped  0x0      0x8001    cpuSC100Big   c1_testmsc8156.eld
     2 Stopped  0x0      0x8002    cpuSC100Big   c2_testmsc8156.eld
     3 Stopped  0x0      0x8003    cpuSC100Big   c3_testmsc8156.eld
     4 Stopped  0x0      0x8004    cpuSC100Big   c4_testmsc8156.eld
     5 Stopped  0x0      0x8005    cpuSC100Big   c5_testmsc8156.eld

# Set core with index 1 the current core
switchtarget 1
# Resume execution on current core
run
```

NOTE

> The `switchtarget` command has no impact on any of the Debug Perspective views outside of the Debugger Shell. That is, after executing a `switchtarget` command, the context of the **Variables**, **Memory**, and other views remain unchanged.

NOTE

> When debugging a multicore application with a Tcl command script, it is recommended that the following command be issued before the debug session is started:
>
> ```
> switchtarget –ResetIndex
> ```

Issuing the `switchtarget –ResetIndex` command before starting the debug sessions ensures that all of the cores are always associated with the same index. That is, core 0 is associated with index 0, core 1 with index 1, and so on.

6.3 Core-Specific Commands

Those debugger commands that do not start with a `mc::` prefix apply only to the current core. Therefore, it is important to switch to the appropriate core using the command `switchtarget` before issuing a command.

Debugger commands that apply only to the default core are shown in Table 2.

Table 2. Debug Commands That Act on One Core

ca::*	finish	nexti	setpc
caln*	funcs	redirect	stack
change	getpid	reg	step
copy	go	restart	stepi
disassemble	kill	restore	stop
display	mem	run	var
evaluate	next	save	

6.4 Breakpoints

When a breakpoint is set using the `bp` command in the Debugger Shell, it is valid for all cores.

6.5 Watchpoints

When a watchpoint is set using the `watchpoint` command in the Debugger Shell, it is valid for all cores.

6.6 Example Multicore Debugging Script

Here is an example of a simple debugging script for a multicore application:

```
# Reset core index
switchtarget -ResetIndex
```

```
# start debug session
debug TestMsc8156

# clear all breakpoints

bp all off


# Set Breakpoints at entry point of function func1
bp func1

# run till breakpoint is executed
mc::go

#wait till all cores reach the breakpoint
wait 10000

# display all available target
switchtarget

# activate core 1
switchtarget 1

# step twice on core 1
step
step

# print current value of PC
display PC

#switch to core 0
switchtarget 0

# print value of PC
display PC
```

NOTE

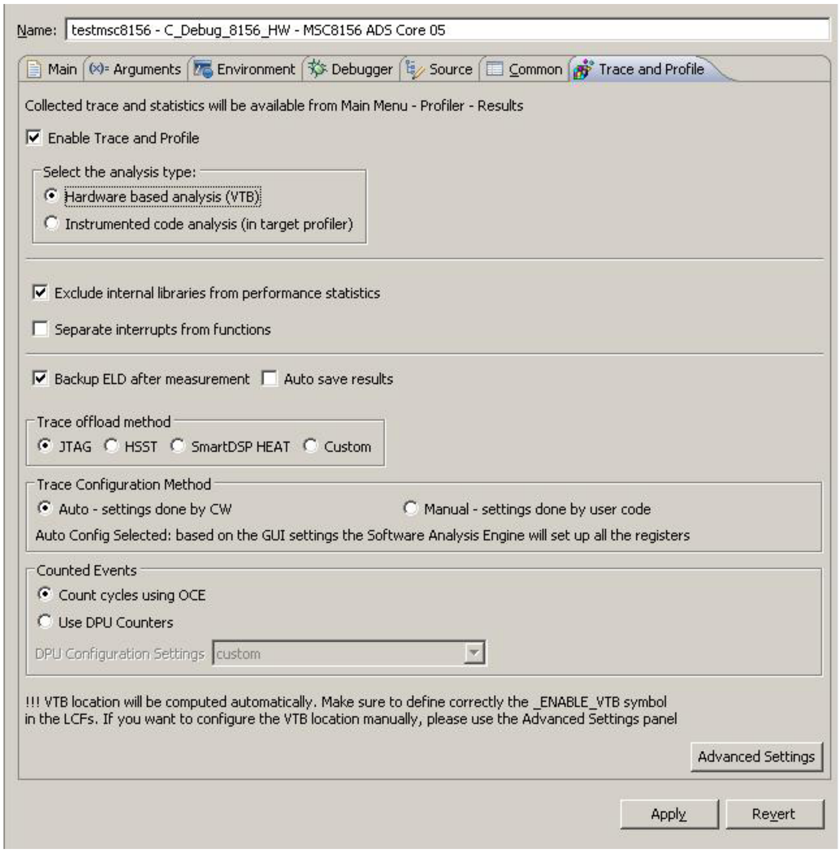
More information on how to use Tcl to manage breakpoints and automate testing can be found in the Application Note, AN4114 “CodeWarrior Debugger TCL Scripting by Example”.

7 Tracing and Profiling

While debugging a multicore application, the code’s execution can be profiled for any number of cores: one, a subset of cores, or all of them.

For each of the cores to be profiled, the following configuration must be performed:

1. Open the **Debug Configuration** dialog.
2. Switch to the **Trace and Profile** tab.
3. Check the **Enable Trace and Profile** option, as shown in Figure 23.



Name: testmsc8156 - C_Debug_8156_HW - MSC8156 ADS Core 05

Collected trace and statistics will be available from Main Menu - Profiler - Results

☑ Enable Trace and Profile

Select the analysis type:
● Hardware based analysis (VTB)
○ Instrumented code analysis (in target profiler)

☑ Exclude internal libraries from performance statistics

☐ Separate interrupts from functions

☑ Backup ELD after measurement ☐ Auto save results

Trace offload method
● JTAG ○ HSST ○ SmartDSP HEAT ○ Custom

Trace Configuration Method
● Auto - settings done by CW ○ Manual - settings done by user code
Auto Config Selected: based on the GUI settings the Software Analysis Engine will set up all the registers

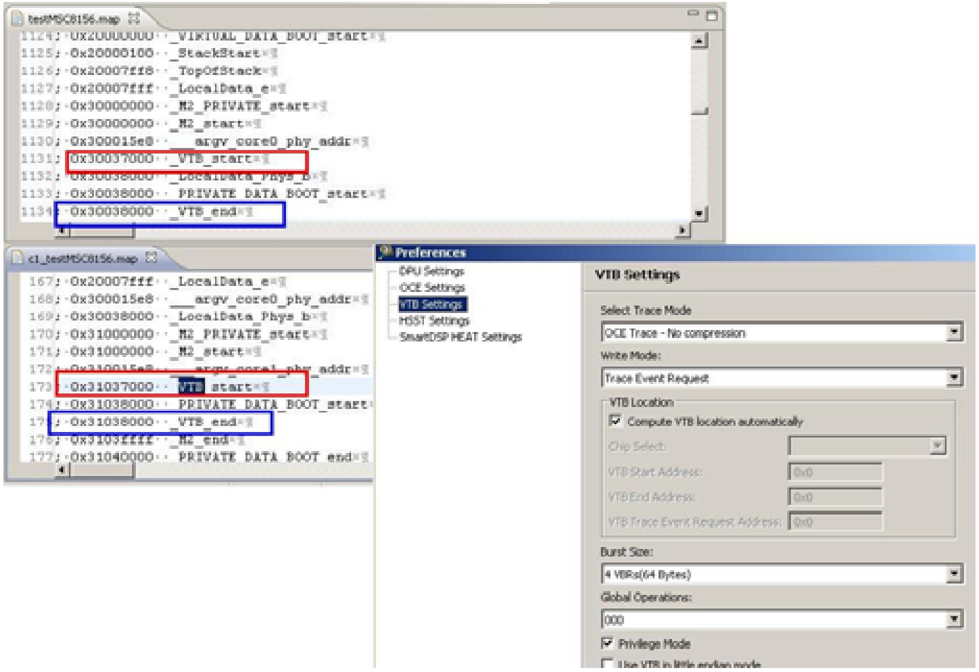
Counted Events
● Count cycles using OCE
○ Use DPU Counters
DPU Configuration Settings custom

!!! VTB location will be computed automatically. Make sure to define correctly the _ENABLE_VTB symbol
in the LCFs. If you want to configure the VTB location manually, please use the Advanced Settings panel

Figure 23. Activating the Code Trace and Profiling Feature

4. Adjust the other settings as required.
5. Click on **Advanced Settings** and check the VTB settings.

6. If **Compute VTB location automatically** is checked, make sure the symbols `_VTB_start` and `_VTB_end` have different values for each core you intend to profile in the application's `application.map` file, as shown in Figure 24.
   - — If the project was created by the wizard without SmartDSP OS support, make sure `_ENABLE_VTB` is set to 1, 2 or 3 in `mmu_attr.l3k`. Refer to comments in `mmu_attr.l3k` for information on meaning of the different settings for `_ENABLE_VTB`.
   - — If the project was created by the wizard with SmartDSP OS support, make sure `USING_VTB` is set to 1 in `os_msc815x_link.l3k`. This will reserve some memory in DDR1 for VTB with each core.



testMSC8156.map

```
1124; 0x20000000  _VIRTUAL_DATA_BOOT_start
1125; 0x20000100  _StackStart
1126; 0x20007ff8  _TopOfStack
1127; 0x20007fff  _LocalData_e
1128; 0x30000000  _M2_PRIVATE_start
1129; 0x30000000  _M2_start
1130; 0x300015e8  ___argv_core0_phy_addr
1131; 0x30037000  _VTB_start
1132; 0x30038000  _LocalData_Phys_b
1133; 0x30038000  _PRIVATE_DATA_BOOT_start
1134; 0x30038000  _VTB_end
```

c1_testMSC8156.map

```
167; 0x20007fff  _LocalData_e
168; 0x300015e8  ___argv_core0_phy_addr
169; 0x30038000  _LocalData_Phys_b
170; 0x31000000  _M2_PRIVATE_start
171; 0x31000000  _M2_start
172; 0x310015e8  ___argv_core1_phy_addr
173; 0x31037000  _VTB_start
174; 0x31038000  _PRIVATE_DATA_BOOT_start
175; 0x31038000  _VTB_end
176; 0x3103ffff  _M2_end
177; 0x31040000  _PRIVATE_DATA_BOOT_end
```

Preferences

VTB Settings

Select Trace Mode: OCE Trace - No compression

Write Mode: Trace Event Request

VTB Location

☑ Compute VTB location automatically

Chip Select:

VTB Start Address: 0x0

VTB End Address: 0x0

VTB Trace Event Request Address: 0x0

Burst Size: 4 VBRs(64 Bytes)

Global Operations: 000

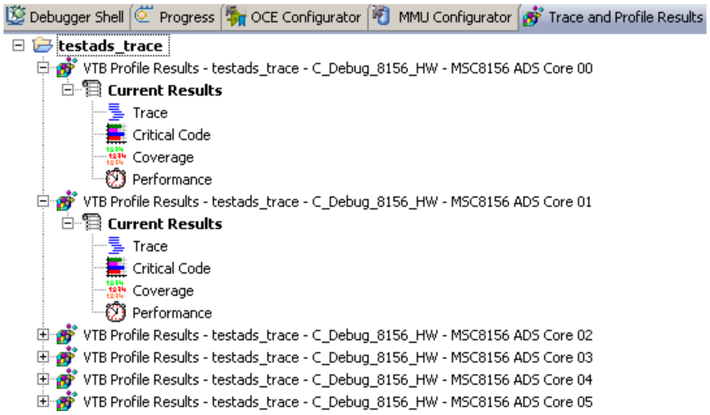
☑ Privilege Mode

☐ Use VTB in little endian mode

Figure 24. Configuring the VTB settings

7. If **Compute location automatically** is unchecked, a memory region for the VTB must be specified. This memory region needs to be different for each core, and should not overlap with the memory regions for application code and data.
   - — For this configuration, it is critical that the VTB buffers occupy separate and unused memory regions.
8. Apply the changes to all of the cores to be profiled.
9. Close the **Debug Configuration** dialog.

10. After the application executes and terminates, for all of the cores that were in use a **Trace and Profile Results** view appears, as shown in Figure 25.



```
testads_trace
  VTB Profile Results - testads_trace - C_Debug_8156_HW - MSC8156 ADS Core 00
    Current Results
      Trace
      Critical Code
      Coverage
      Performance
  VTB Profile Results - testads_trace - C_Debug_8156_HW - MSC8156 ADS Core 01
    Current Results
      Trace
      Critical Code
      Coverage
      Performance
  VTB Profile Results - testads_trace - C_Debug_8156_HW - MSC8156 ADS Core 02
  VTB Profile Results - testads_trace - C_Debug_8156_HW - MSC8156 ADS Core 03
  VTB Profile Results - testads_trace - C_Debug_8156_HW - MSC8156 ADS Core 04
  VTB Profile Results - testads_trace - C_Debug_8156_HW - MSC8156 ADS Core 05
```

Figure 25. Display to Access the Various Trace Results

The data for each core profiled can be displayed by clicking on the **Trace**, **Critical Code**, **Coverage**, or **Performance** option associated with each core in this view.

8 Multi-Device Considerations

The following section discussed how to configure the CodeWarrior tools to debug a system with multiple devices. That is, the system that has two or more StarCore processors on it.

8.1 Configuring the CodeWarrior Debugger to Use Multiple Devices

The CodeWarrior tools must be configured properly in order to debug multi-device systems without spurious issues appearing. For information on how to do this, refer to the application note, AN3908, “A Guide to Configuring Multiple MSC8156 Devices on a Single JTAG Chain Using CodeWarrior Development Studio for StarCore DSP Architectures v10.0”. This document appears as the file `AN3908.Multi-DSP JTAG Chain.pdf`, and can be found in the `{CodeWarrior installation}SC\Help\PDF` directory.

8.2 Group Hierarchy

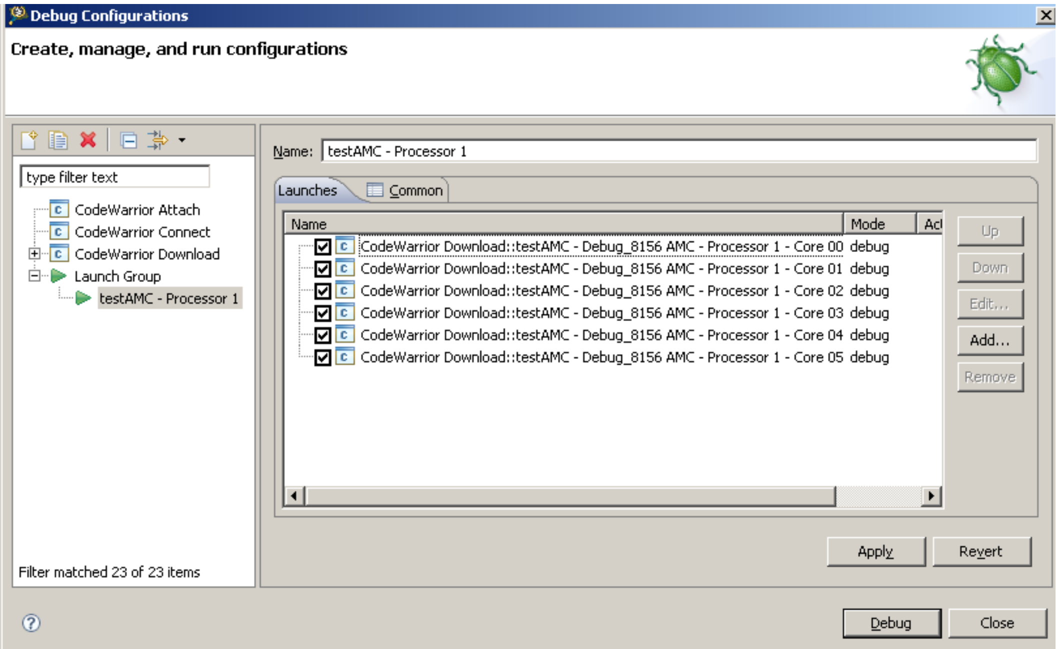
When debugging on multiple devices, a group hierarchy can be defined in order to debug all of the device’s cores with a single mouse click.

Section 1.1.1 describes how to define a launch group that manages all six cores on a MSC8156 device. If the system has multiple MSC8156 devices in the JTAG chain, a launch group can be defined for each one. A master launch group is then defined that includes the launch group for every device in the system.

8.2.1 Example for a MSC8156AMC Board

The MSC8156AMC board is a system that contains three MSC8156 processors. To have the CodeWarrior debugger manage and control all eighteen cores, define a launch group that starts all eighteen cores with a single mouse click. Proceed as follows:

1. Create a launch group named `testAMC - Processor 1` that manages all of the cores on the first device (cores 00 – 05) by following the steps outlined in section 1.1.1 and 1.1.2. The results appear in Figure 26.



Debug Configurations

Create, manage, and run configurations

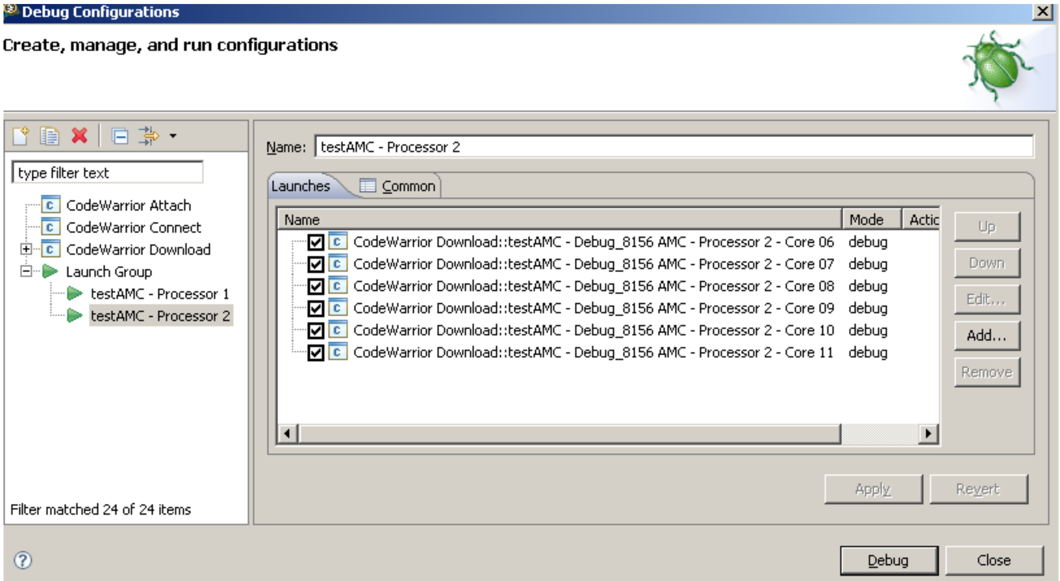
Name: testAMC - Processor 1

Name	Mode
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 1 - Core 00	debug
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 1 - Core 01	debug
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 1 - Core 02	debug
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 1 - Core 03	debug
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 1 - Core 04	debug
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 1 - Core 05	debug

Filter matched 23 of 23 items

Figure 26. Making the Launch Group Processor 1 That Controls Cores 0 Through 5

2. Create a launch group named `testAMC - Processor 2` that manages all of the cores on the second device (cores 06 – 11) by following the steps in section 1.1.1 and 1.1.2. See Figure 27.



Debug Configurations

Create, manage, and run configurations

Name: testAMC - Processor 2

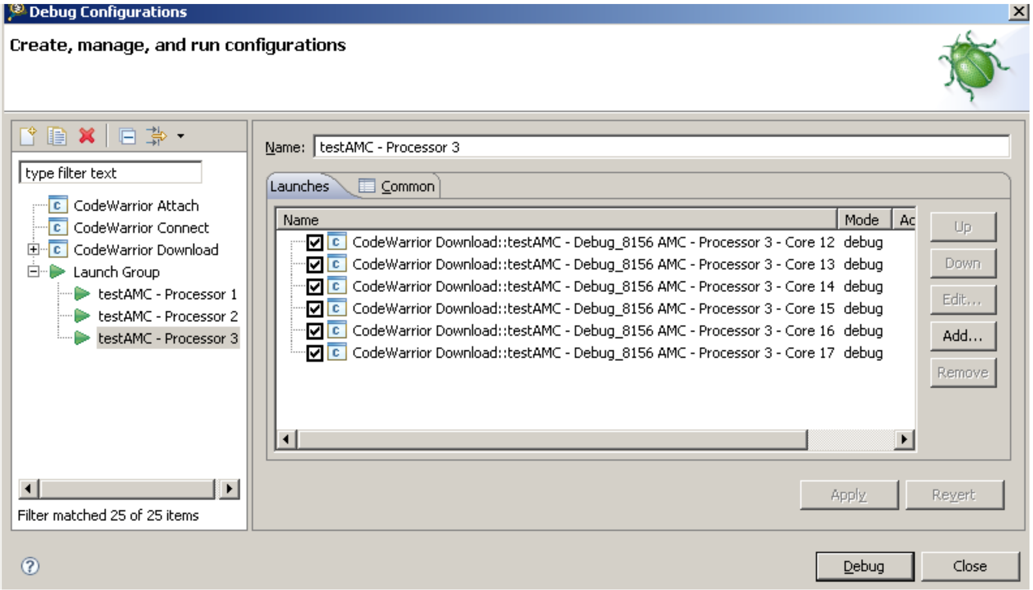
- CodeWarrior Attach
- CodeWarrior Connect
- CodeWarrior Download
- Launch Group
  - testAMC - Processor 1
  - testAMC - Processor 2

Filter matched 24 of 24 items

Name	Mode	Actic
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 2 - Core 06	debug	
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 2 - Core 07	debug	
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 2 - Core 08	debug	
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 2 - Core 09	debug	
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 2 - Core 10	debug	
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 2 - Core 11	debug	

Figure 27. The Launch Group Processor 2 That Manages Cores 6 Through 11

3. Create a launch group named `testAMC - Processor 3` that manages all of the cores on the third device (cores 12 – 17) by following the steps described in section 1.1.1 and 1.1.2. See Figure 28.



Debug Configurations

Create, manage, and run configurations

Name: testAMC - Processor 3

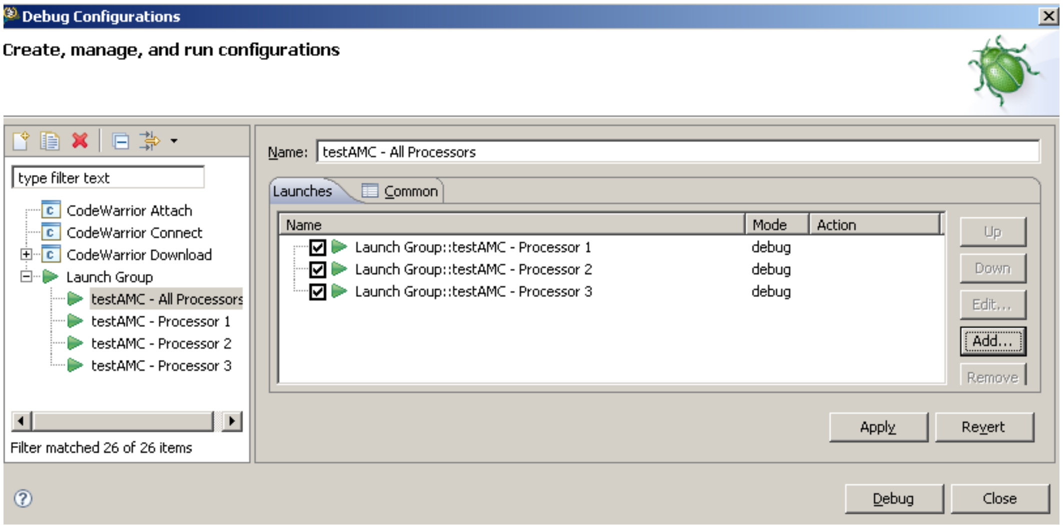
- CodeWarrior Attach
- CodeWarrior Connect
- CodeWarrior Download
- Launch Group
  - testAMC - Processor 1
  - testAMC - Processor 2
  - testAMC - Processor 3

Filter matched 25 of 25 items

Name	Mode	Ac
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 3 - Core 12	debug	
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 3 - Core 13	debug	
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 3 - Core 14	debug	
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 3 - Core 15	debug	
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 3 - Core 16	debug	
☑ CodeWarrior Download::testAMC - Debug_8156 AMC - Processor 3 - Core 17	debug	

Figure 28. Launch Group Processor 3 That Manages Cores 12 Through 17

4. Finally, create a launch group named `testAMC - All Processors` that incorporates the three launch groups just made in steps 1, 2, and 3. The results are shown in Figure 29.



- CodeWarrior Attach
- CodeWarrior Connect
- CodeWarrior Download
- Launch Group
  - testAMC - All Processors
  - testAMC - Processor 1
  - testAMC - Processor 2
  - testAMC - Processor 3

Filter matched 26 of 26 items

Name: testAMC - All Processors

Name	Mode	Action
☑ Launch Group::testAMC - Processor 1	debug	
☑ Launch Group::testAMC - Processor 2	debug	
☑ Launch Group::testAMC - Processor 3	debug	

Figure 29. Making the Master Launch Group That Controls the Three Other Launch Groups

To start debugging on all eighteen cores, select the `testAMC – All Processors` launch group in the **Debug Configuration** window and click on **Debug**. To start debugging on processor 1 only, just select the `testAMC – Processor 1` launch group in the **Debug Configuration** window and click on **Debug**.

NOTE

When defining a launch group for multiple devices, the **Execute Reset** option must be checked for only *one core* in the entire system (usually core 0 on Processor 1). Make sure that the launch group for that device is specified first in the All Processors launch group.

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
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale, the Freescale logo, CodeWarrior and StarCore are trademarks of Freescale Semiconductor, Inc. Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2010 Freescale Semiconductor, Inc.

Document Number:
Rev:0
Date: 06/2010