CodeWarrior™ Development Studio

for StarCore™ and SDMA Architectures

Quick Start for Windows® Operating Systems and Nexus Trace

This Quick Start explains how to set up a sample project to collect Nexus trace data. The sample project demonstrates how to:

- use Enhanced On-Chip Emulation (EOnCE™) debugging to trace program flow in specific parts of code
- trace data changes in specific memory ranges
- use Data Acquisition trace to monitor write operations to a specific address
- use Nexus watchpoint messaging to capture watchpoint events

Section A: Configure the Project

1. Open sample project

a. Select **Start > Programs > Freescale CodeWarrior > CodeWarrior for StarCore and SDMA V1.0 > CodeWarrior IDE** from Windows task bar— IDE starts; CodeWarrior main window appears

b. From CodeWarrior menu bar, select **File > Open** — **Open** dialog box appears

c. Use dialog box to find and open Nexus directory at this location, where *CodeWarrior* is path to your CodeWarrior installation:

CodeWarrior`\(CodeWarrior_Examples)\StarCore_Examples\Nexus`

d. Select `nexus_example.mcp` file

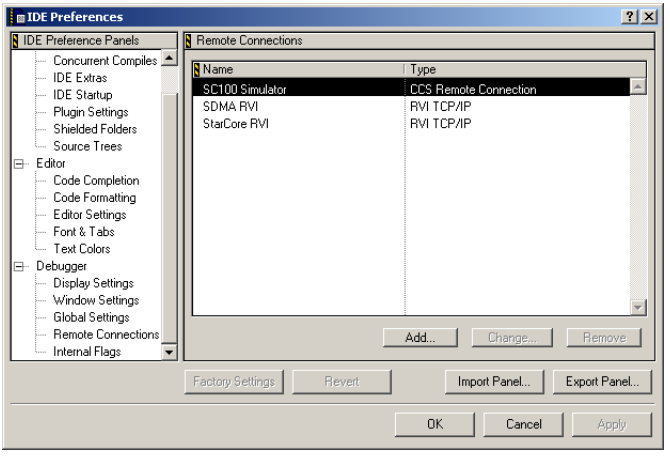
e. Click **Open** button — system opens project; project window appears, docked at left side of main window

NOTE As an example, this Quick Start shows how to create a remote connection for MXC-05 hardware. If you have different hardware, use these steps as a guide to create an appropriate remote connection.

a. Select **Edit > Preferences** from CodeWarrior menu bar — **IDE Preferences** window appears

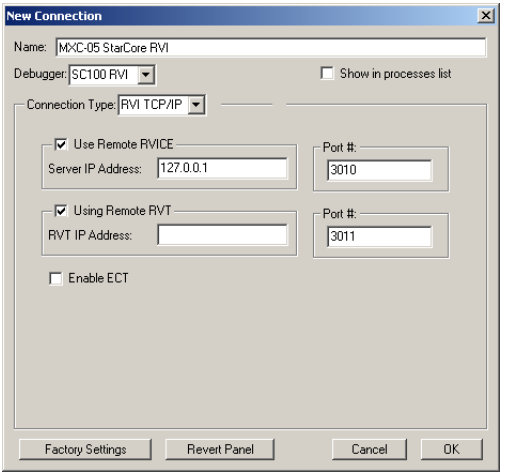
b. Select **Remote Connections** from **IDE Preference Panels** list on left side of window — panel appears in window

IDE Preferences Window — Remote Connections



Add button — **New Connection** dialog box appears

New Connection Dialog Box



d. In **Name** text box, enter `MXC-05 StarCore RVI`

e. Use **Debugger** list box to specify **SC100 RVI**

f. In **Server IP Address** text box, enter IP address of RVICE server

g. Click **OK** button — IDE saves new remote-connection configuration; name of new remote connection appears in panel list

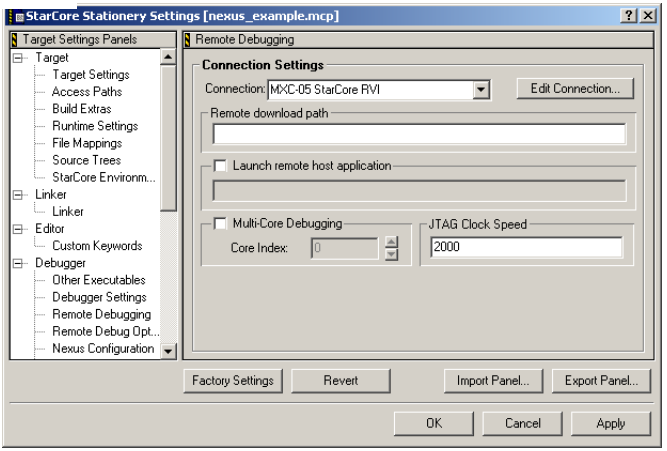
h. Click **OK** button — IDE Preferences window closes

3. Configure remote debugging for project

a. Select **Edit > StarCore Stationery Settings** from CodeWarrior menu bar — Target settings window appears

b. Select **Remote Debugging** from **Target Settings Panels** list on left side of window — panel appears in window

Target Settings Window — Remote Debugging

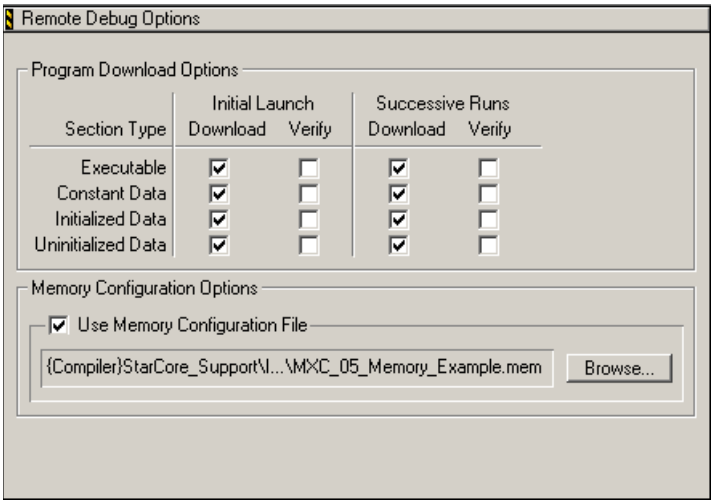


c. Use **Connection** list box to specify remote connection that you created in step 2.

4. Configure remote-debugging options for project

a. Select **Remote Debug Options** from **Target Settings Panels** list on left side of window — panel appears in window

Target Settings Window — Remote Debug Options



:k **Use Memory Configuration File** checkbox

c. Click **Browse** button — **Choose the Debugger Memory Configuration File** dialog box appears

d. Use dialog box to navigate to

   CodeWarrior`\StarCore_Support\Initialization_Files\MemoryConfigFiles`

   where *CodeWarrior* is path to your CodeWarrior installation

e. Use dialog box to select appropriate memory-configuration file for your hardware:

   – `i.300-30_Memory_Example.mem`
   – `MXC_05_Memory_Example.mem`
   – `MXC275-30_Memory_Example.mem`

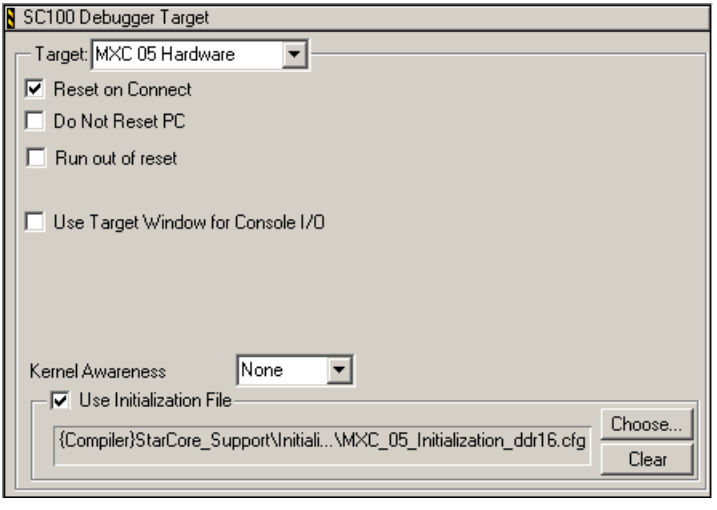
f. Use **Path Type** list box to specify **Compiler Relative**

g. Click **Open** button — path to selected memory-configuration file now appears to left of **Browse** button

5. Configure SC100 debugger target for project

a. Select **SC100 Debugger Target** from **Target Settings Panels** list on left side of window — panel appears in window

Target Settings Window — SC100 Debugger Target



Target list box to specify appropriate target for your hardware

c. Check **Use Initialization File** checkbox

d. Click **Choose** button — **Choose the StarCore Register Init File** dialog box appears

e. Use dialog box to navigate to

   CodeWarrior`\StarCore_Support\Initialization_Files\RegisterConfigFiles`

   where *CodeWarrior* is path to your CodeWarrior installation

f. Use dialog box to select appropriate register-initialization file for your hardware:

   – `i.300-30\i.300-30_Initialization.cfg`
   – `MXC_05\MXC_05_Initialization_ddr16.cfg`
   – `MXC275-30\MXC275-30_Initialization.cfg`

g. Use **Path Type** list box to specify **Compiler Relative**

h. Click **Open** button — path to selected initialization file now appears to left of **Choose** button

6. Build and debug project

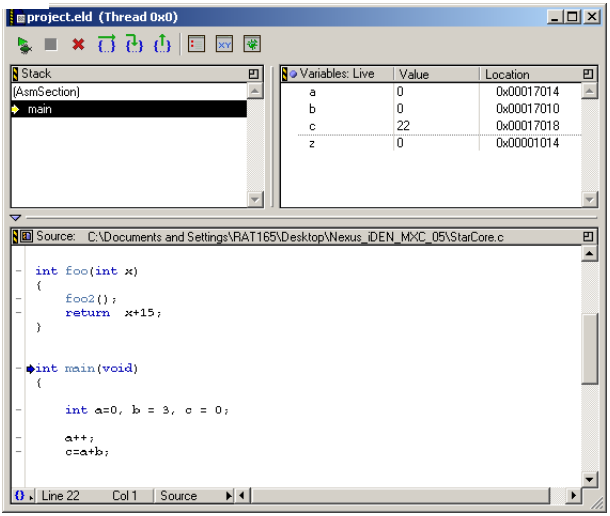
a. Select **Project > Make** from CodeWarrior menu bar — IDE builds (compiles, assembles, and links) source code; IDE generates executable program

b. Select **Project > Debug** — debugging session starts; thread window appears

---

NOTE The thread window shows the relationship among variables, their values, and their addresses. You will use this information to complete Section B.

Thread Window



Section B: Configure Nexus Settings

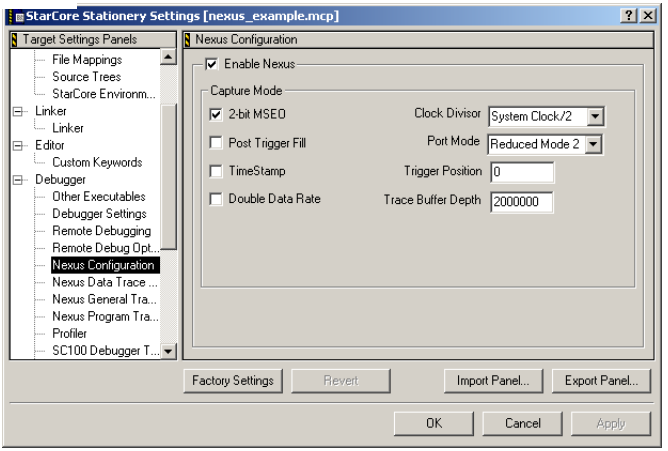
1. **Specify Nexus configuration**

   a. Select **Edit > StarCore Stationery Settings** from CodeWarrior menu bar — Target Settings window appears

   b. Position Target Settings window so that thread window's variable values and addresses remain visible

   c. Select **Nexus Configuration** from **Target Settings Panels** list on left side of window — panel appears in window

arget Settings Window — Nexus Configuration

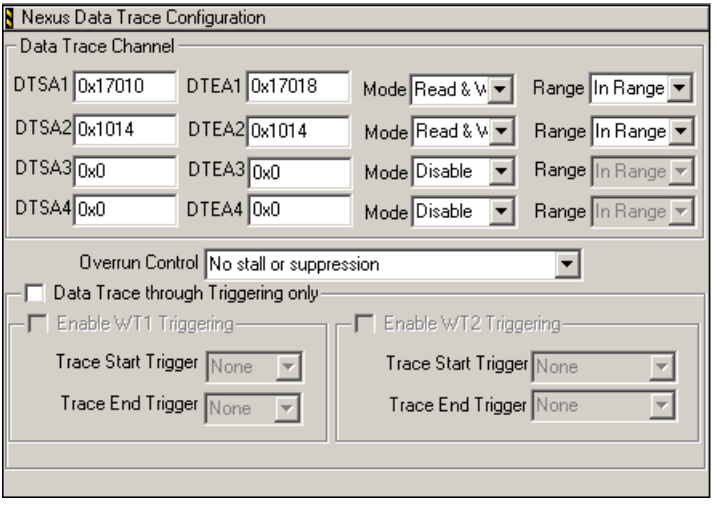


d. Check **Enable Nexus** checkbox

2. **Specify Nexus data trace configuration**

a. Select **Nexus Data Trace Configuration** from **Target Settings Panels** list on left side of window — panel appears in window

Target Settings Window — Nexus Data Trace Configuration



thread window to determine range of addresses for variables a, b, and c

c. Enter starting address in **DTSA1** text box

d. Enter ending address in **DTEA1** text box

e. Enter address of variable z in **DTSA2** and **DTEA2** text boxes

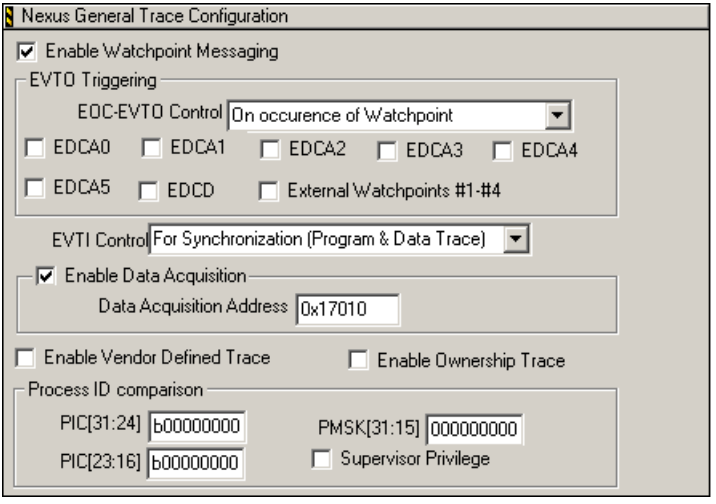
f. Use **Mode** list boxes to specify **Read & Write** for both DTSA1/DTEA1 pair and DTSA2/DTEA2 pair — IDE configures Nexus trace to generate trace messages for read events and write events

g. Use **Range** list boxes to specify **In Range** for both DTSA1/DTEA1 pair and DTSA2/DTEA2 pair — IDE configures Nexus trace to monitor only addresses within specified ranges

NOTE Restricting the range of addresses for data-trace channels and data trace through EOnCE triggering is a good way to collect only the most relevant data.

a. Select **Nexus General Trace Configuration** from **Target Settings Panels** list on left side of window — panel appears in window

Target Settings Window — Nexus General Trace Configuration



b. Check **Enable Watchpoint Messaging** checkbox — IDE configures Nexus debugging to generate watchpoint messages

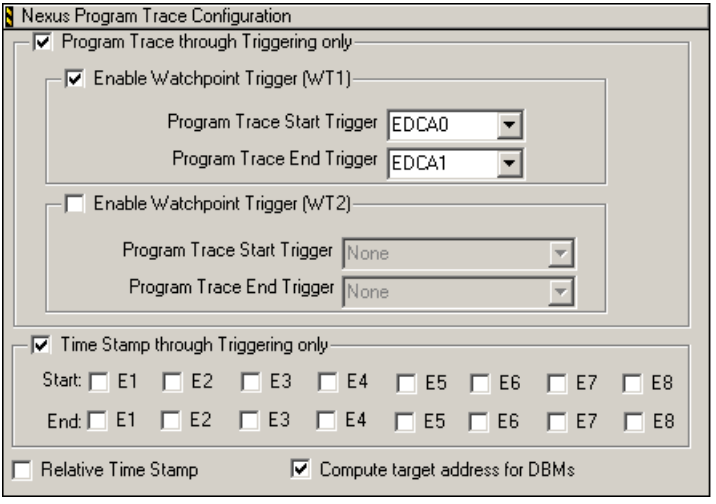
c. Check **Enable Data Acquisition** checkbox — IDE configures Nexus debugging to generate a data-acquisition message for each write to specified **Data Acquisition Address**

d. Use thread window to determine address of variable `b`

e. Enter address of variable `b` in **Data Acquisition Address** text box

a. Select **Nexus Program Trace Configuration** from **Target Settings Panels** list on left side of window — panel appears in window

Target Settings Window — Nexus Program Trace Configuration



b. Check **Program Trace through Triggering only** checkbox — IDE configures Nexus debugging to generate program trace based on EOnCE trigger events

c. Check **Enable Watchpoint Trigger (WT1)** checkbox — IDE configures Nexus debugging to collect program trace based on EOnCE trigger events on WT1

d. Use **Program Trace Start Trigger** list box to specify **EDCA0** — configures program trace to begin on watchpoint EDCA0

e. Use **Program Trace End Trigger** list box to specify **EDCA1** — configures program trace to end on watchpoint EDCA1

f. Check **Time Stamp through Triggering only** checkbox — IDE configures Nexus debugging to generate timestamps only for Embedded Cross Trigger (ECT) events

NOTE Restricting the range of addresses for program trace through EOnCE triggering is a good way to collect only the most relevant data.

g. Click **OK** button — Target Settings window closes; thread window remains open

Section C: Configure EOnCE Settings

1. Open EOnCE settings file

a. Select **Debug > EOnCE > Open EOnCE Configuration** — **Open** dialog box appears

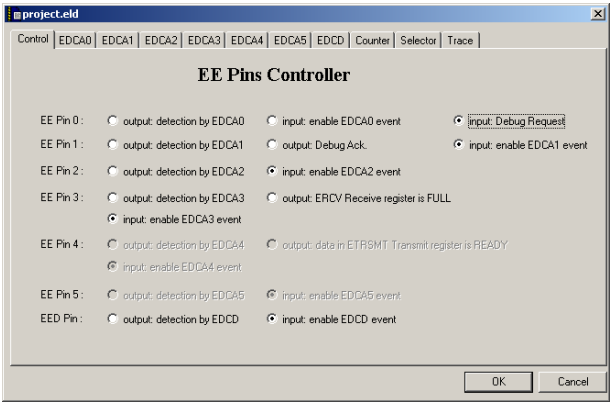
b. Use dialog box to find and open Nexus directory at this location, where *CodeWarrior* is path to your CodeWarrior installation:

CodeWarrior\`(CodeWarrior_Examples)\StarCore_Examples\Nexus`

c. Select `nexus_eonce_config.cfg` file

d. Click **Open** button — EOnCE configuration window appears

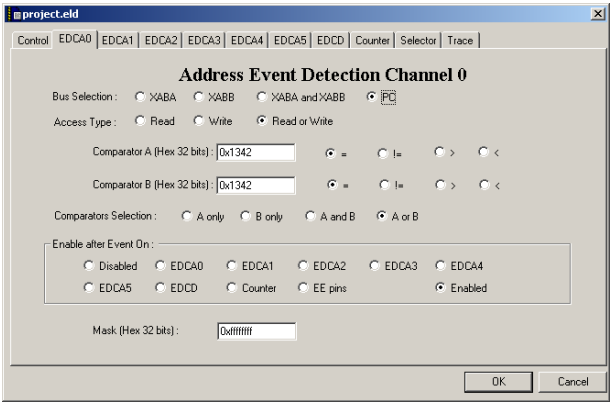
EOnCE Configuration Window



re EDCA0 page

a. Click **EDCA0** tab — **Address Event Detection Channel 0** page appears in window

EOnCE Configuration Window — EDCA0 Page

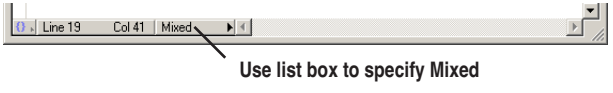


b. In **Bus Selection** group, click **PC** option button — IDE configures EOnCE settings to detect events based on source-code instruction addresses

c. In **Access Type** group, click **Read or Write** option button — IDE configures EOnCE settings to detect both read and write accesses

d. At bottom of thread window, use source-view list box to view Mixed source code and assembly-language instructions

Thread Window — Viewing Mixed



Use list box to specify Mixed

e. Use thread window to scroll to this comment line in source code:

```
// trigger program trace ON (EDCA0)
```

f. Use thread window to determine address of assembly-language instruction that appears *just before* comment line

g. In EOnCE Configuration window, enter address (in hexadecimal notation) into **Comparator A** and **Comparator B** text boxes

= option button next to both **Comparator A** and **Comparator B** text boxes

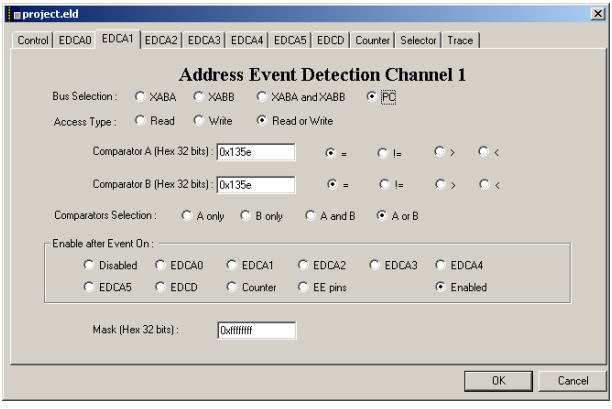
i. In **Comparators Selection** group, click **A or B** option button

j. In **Enable after Event On** group, click **Enabled** option button — IDE configures EOnCE debugging to always compare Comparator A to Comparator B (and not rely on specific event to trigger comparison)

3. **Configure EDCA1 page**

a. Click **EDCA1** tab — **Address Event Detection Channel 1** page appears in window

EOnCE Configuration Window — EDCA1 Page



b. In **Bus Selection** group, click **PC** option button — IDE configures EOnCE settings to detect events based on source-code instruction addresses

c. In **Access Type** group, click **Read or Write** option button — IDE configures EOnCE settings to detect both read and write accesses

d. Use thread window to scroll to this comment line in source code:

```
// trigger program trace OFF (EDCA1)
```

e. Use thread window to determine address of assembly-language instruction that appears *just after* comment line

OnCE Configuration window, enter address (in hexadecimal notation) into **Comparator A** and **Comparator B** text boxes

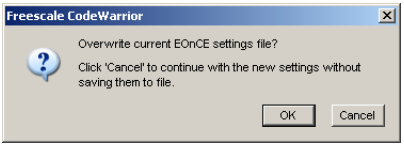
g. Click **=** option button next to both **Comparator A** and **Comparator B** text boxes

h. In **Comparators Selection** group, click **A or B** option button

i. In **Enable after Event On** group, click **Enabled** option button — IDE configures EOnCE debugging to always compare Comparator A to Comparator B (and not rely on specific event to trigger comparison)

j. Click **OK** button — EOnCE configuration window disappears; dialog box appears asking whether to overwrite current EOnCE settings file

Overwrite Settings Dialog Box

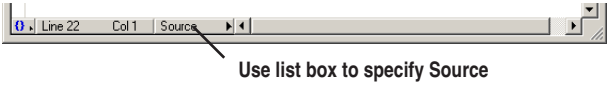


k. Click **Cancel** button—dialog box closes

NOTE By clicking the **Cancel** button, the settings in the EOnCE configuration window apply to the current debugging session only. If you click the **OK** button instead, the IDE overwrites the settings in the `project.eld.eonce` file with those you just specified in the EOnCE Configuration window.

l. At bottom of thread window, use source-view list box to view Source code

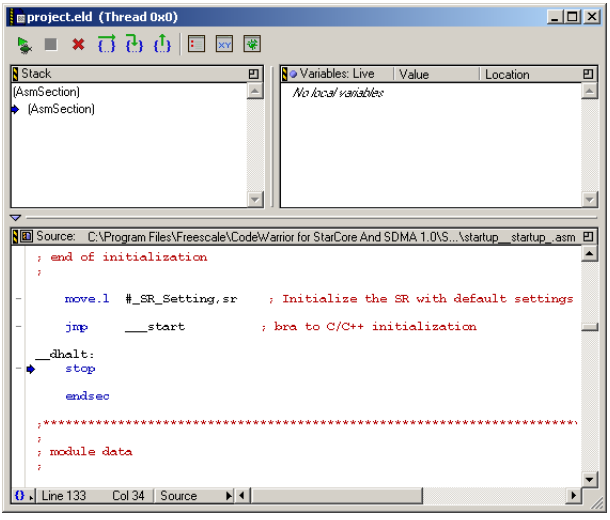
Thread Window — Viewing Source



1. **From CodeWarrior menu bar, select Project > Run**

   IDE runs project; program runs to completion; new thread window appears

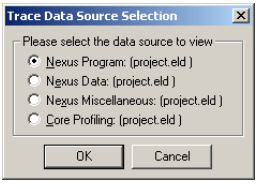
Thread Window



2. **Examine Nexus program trace**

   a. Select **Data > View Trace** — **Trace Data Source Selection** dialog box appears

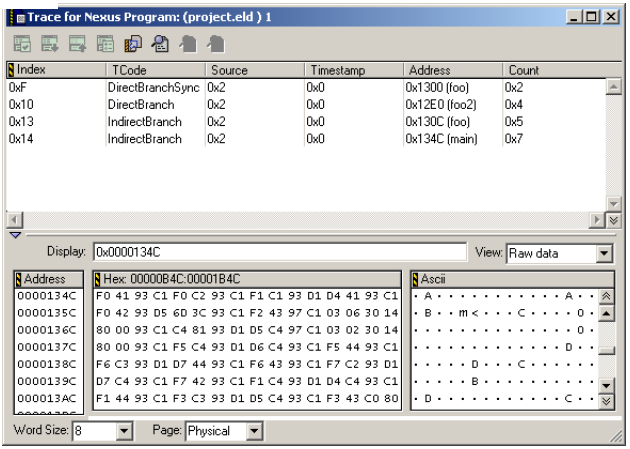
Trace Data Source Selection Dialog Box



   b. Select **Nexus Program** option button

   c. Click **OK** button — **Trace for Nexus Program** window appears

Trace for Nexus Program Window



NOTE Recall that in step 4 of Section B, you configured the **Program Trace through Triggering only**, **Program Trace Start Trigger**, and **Program Trace End Trigger** options so that the collected trace reflected program execution from the first source-code comment to the second source-code comment.

d. Use program-trace window to examine program flow of control for triggered block of code

The window shows how flow went first to function `foo()`, then to `foo2()`, then back to `foo()`, and finally back to `main()`

NOTE You can click the address associated with each function to view the code of that function.

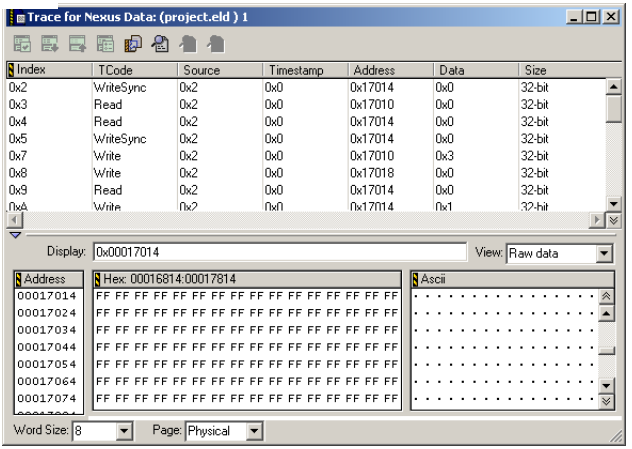
3. Examine Nexus data trace

a. Select **Data > View Trace** — **Trace Data Source Selection** dialog box appears

b. Select **Nexus Data** option button

c. Click **OK** button — **Trace for Nexus Data** window appears

Trace for Nexus Data Window



NOTE Recall that in step 2 of Section B, you configured the starting and ending addresses of Data Trace Channel 1 (**DTSA1/DTEA1**) and Data Trace Channel 2 (**DTSA2/DTEA2**) so that the collected trace covered the addresses of specific source-code variables.

d. Use data-trace window to examine read and write events for specified address range of source-code variables.

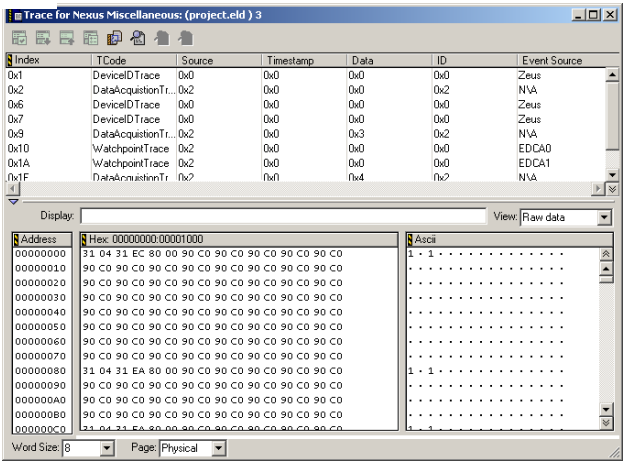
4. Examine Nexus miscellaneous trace

a. Select **Data > View Trace** — **Trace Data Source Selection** dialog box appears

b. Select **Nexus Miscellaneous** option button

c. Click **OK** button — **Trace for Nexus Miscellaneous** window appears

Trace for Nexus Miscellaneous Window



NOTE Recall that in step 3 of Section B, you configured the **Enable Watchpoint Messaging**, **Enable Data Acquisition**, and **Data Acquisition** options so that the collected trace corresponds to the variable `b` in the source code.

d. Use miscellaneous-trace window to examine watchpoint messages resulting from EOnCE watchpoint triggers (**EDCA0** and **EDCA1**) and data-acquisition messages resulting from write events to variable `b` in source code

NOTE You can use the **Index** column of each Nexus trace window to follow the order of operation from data trace to program trace to miscellaneous trace.

›ugging session

a. Select **Debug > Kill** — debug session ends; thread window closes

b. Close **StarCore Debugger Console** window

c. Close **RVI Information Console for StarCore** window

d. Select **File > Close** — project window for sample project disappears

Congratulations!

You just used CodeWarrior software to run a simple StarCore and SDMA program and collect Nexus trace data.

How to Contact Us

Corporate Headquarters	Freescale Semiconductor, Inc. 7700 West Parmer Lane Austin, TX 78729 U.S.A.
World Wide Web	http://www.freescale.com/codewarrior
Technical Support	http://www.freescale.com/support