Freescale Semiconductor
Application Note

Document Number: AN5044
Rev. 05/2015

AIOP Task Aware Debug

Contents

1 Introduction....1
2 Preliminary background.... 1
3 OSM application example....2
4 Debugging OSM example project using CodeWarrior IDE....3
5 AIOP OS awareness features.... 6
6 AIOP task specific breakpoints....12
7 AIOP per-task stepping.... 19
8 AIOP run-control operations.... 24
9 Running OSM example on LS2 simulator....25

1 Introduction

This document describes the how to debug the AIOP tasks. It also describes the AIOP task visibility, AIOP specific breakpoints and per-task stepping.

2 Preliminary background

The AIOP hardware operates in a different manner as compared to a traditional architecture (for example, Power Architecture), as a result the debugging is also slightly different for an AIOP. Following are some features, supported by the AIOP hardware:

- AIOP has the Core Task Scheduler (unit of execution is task)
    - Up to 16 tasks per core
    - Tasks are not preempted by the scheduler
    - Tasks yield when they make a call to an accelerator
- AIOP does not produce tasks by default. Tasks are created under these two conditions:
    - When a packet arrives
    - When a TMAN counter expires
- AIOP application has the following features:
    - Its supports multiple entry points
    - The entry points are executed by tasks

© 2014–2015 Freescale Semiconductor, Inc.

The CodeWarrior supports the AIOP for run-control operations at the following levels:

- AIOP system level
- Core level
- Task level

AIOP is a C programmable processor comprised of a variable number of e200 cores which are further grouped into *core clusters* as an implementation artifact. The table below lists and describes various AIOP specific components.

Component	Description
AIOP core	An AIOP core is an instance of an e200 core comprising the AIOP.
AIOP cluster	A core cluster consists of four e200 cores, each with their own I-cache, workspace RAM, and a shared Instruction RAM (IRAM). The AIOP clusters are not represented in any way in the debugger GUI but the debugger handles the AIOP clusters internally by downloading code or setting software breakpoints in all AIOP clusters comprising the AIOP system.
AIOP system	The totality of AIOP cores comprising an AIOP instance is also referred to as the AIOP system. The debugger offers run control commands at the AIOP system level.
AIOP SMP architecture	CodeWarrior debugger for AIOP supports Symmetric Multi_Processing (SMP) and is capable of debugging the same executable (or set of executables) on the whole number of cores comprising the AIOP system.
AIOP tasks	The fundamental unit of operation in an AIOP system is the task. Tasks are created and terminated by the hardware. A finite number of tasks (maximum of 256) can exist and execute simultaneously inside the AIOP system.

The system and core level debugging are easy to use and predictable, but the novelty appears to the AIOP task level debugging. This documents explains what are the available debug capabilities at task level and how can you use them in the CodeWarrior for Advanced Packet Processing.

There are two important run control states of the AIOP system, as listed below:

- **AIOP global halt**

  Global halt represents a state in which all AIOP cores comprising the AIOP system are suspended for debug purposes (not to be confused with AIOP tasks not being scheduled for execution by the task scheduler).

- **AIOP running**

  AIOP running represents a state in which all the AIOP cores comprising the AIOP system are in running mode (that is, not suspended for debug purposes). AIOP running state scribes only the state of the AIOP cores and not of the tasks. During an AIOP running state, various number of tasks can be suspended for debug purposes, but the e200 cores on which they are scheduled keeps running other tasks.

3 OSM application example

The below listed use-cases and debugging examples, are based on the AIOP OSM application available at the following location:

```
<CW_Layout>\LS\CodeWarrior_Examples\Bareboard_Examples\AIOP_OSM_example
```

This example project contains a simple application of creating tasks using TMan and controlled using order in scope manager (OSM). TMan and work scheduler are initialized by the default task (task 0) and after executing an accelerator call the task scheduling is triggered. The flow after scheduling are in short terms, in which each task starts execution in state XX (execution exclusive), transitions to XC (execution concurrent) within one of two flows encouraged to go out of order and transitions to XX mode again. The transitions and egress order are recorded and verified in the end by the last scheduled task. In order to verify that the test has passed, you need to run multicore and when finished, verify that the *final_result* variable is 1 (pass).

4 Debugging OSM example project using CodeWarrior IDE

To debug the OSM example project using the CodeWarrior IDE, follow these steps:

1. Open the CodeWarrior for APP IDE.
2. Import the OSM example to your workspace by selecting **File > Import > General > Existing Projects into Workspace** from the IDE menu bar, then click **Next**.
3. Browse the OSM example project, from this location: `<CW_Layout>\LS\CodeWarrior_Examples\Bareboard_Examples\AIOP_OSM_example`
4. Click **Finish**.

   The project appears in the **CodeWarrior Projects** view.

5. Ensure that the hardware target is selected.

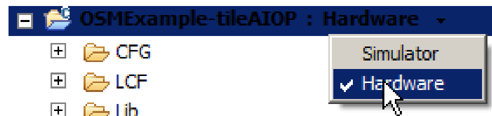

Figure 1. CodeWarrior Projects view - Hardware target

6. Right-click on the project and select **Build Project** from the context-menu.

   The IDE build your project.

7. Configure the connection settings as follows:
   a. Select **Run > Debug Configurations** from the IDE menu bar.

      The **Debug Configurations** dialog appears.

   b. Click **Edit** from the **Target Settings** group and select the connection type and the probe IP to which the hardware target is connected.

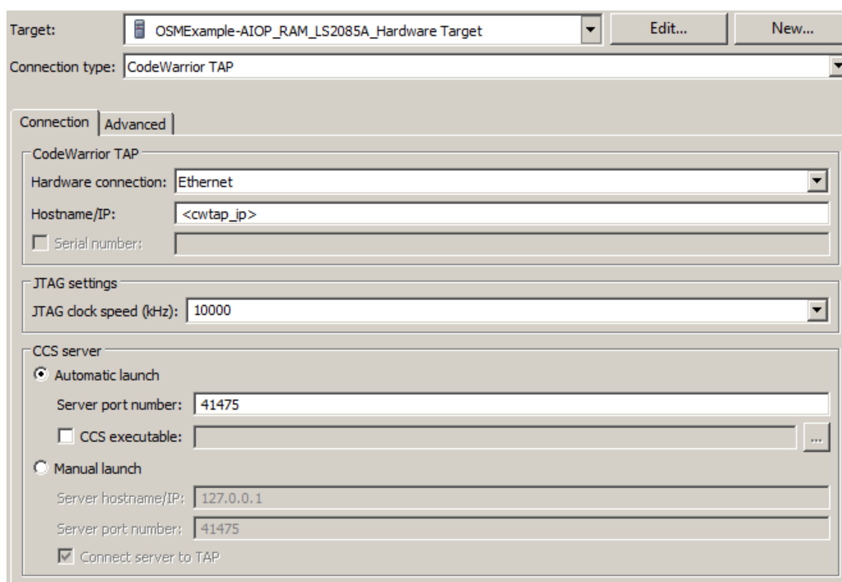

Figure 2. Debug Configuration - Configuring the connection

8. Click **Debug**.
9. The application stops at the **main**() function.

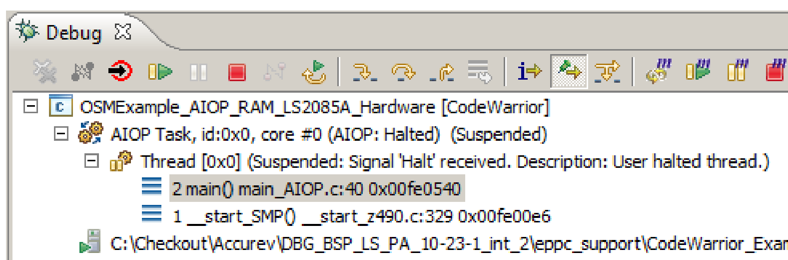

Figure 3. Debug view - Application stopped at main()

10. Click **Resume multicore**.
11. Wait for the target to stop (this occurs when there is no task left for scheduling).

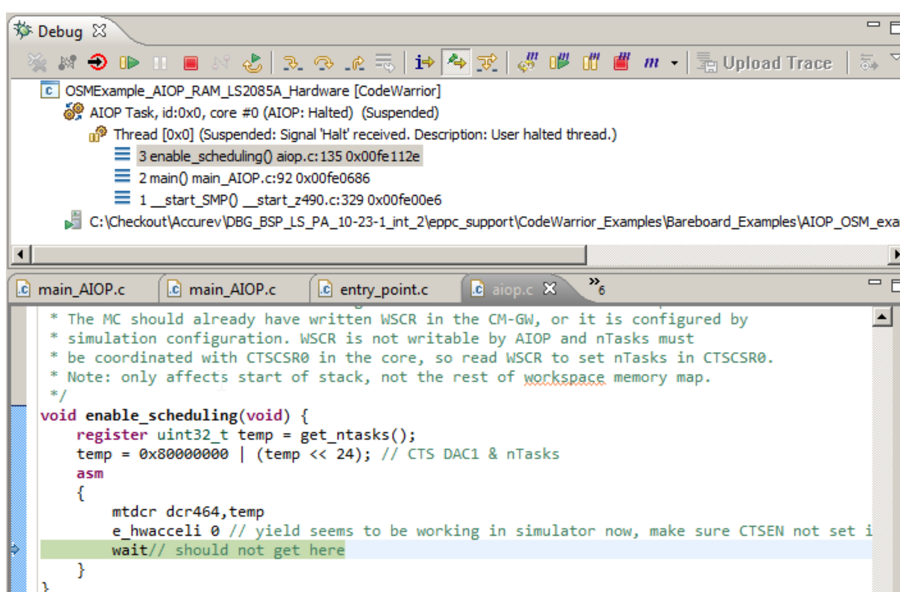

Figure 4. Debug perspective

12. Open the **System Browser** view and find the task that is in **Executing** mode. This is the task that has been scheduled last. Double-click the task to target it in the **Debug** view. You can see the stack crawl, memory, registers and variables in the context of the selected task in **Debug** view.

Task Id	Core	PC	Status	Accel Id
0x18	1	0xfe0532	Idle	NA
0x19	1	0xfe0532	Idle	NA
0x1a	1	0xfe0532	Idle	NA
0x1b	1	0xfe0532	Idle	NA
0x1c	1	0xfe0532	Idle	NA
0x1d	1	0xfe0532	Idle	NA
0x1e	1	0xfe0532	Idle	NA
0x1f	1	0xfe0532	Idle	NA
0x20	2	0xfe0e72	Executing	NA
0x21	2	0xfe0532	Idle	NA
0x22	2	0xfe0532	Idle	NA

Figure 5. System Browser view

For verifying if the application ran successfully you may select **Variables** view and look for the value of **final_result** variable in the context of the task just scheduled.

Figure 6. Debug perspective - Variables view

13. For debugging the task scheduling flow you may restart the debug session and set the breakpoints in `osm_timer_callback()` in `entry_point.c` source file (this function is the task entry point) and run multicore in order to hit them.

```
main_AIOP.c    entry_point.c
    ext_addr = mem_alloc(DMA_SIZE, DDR_MEM);
}


void osm_timer_callback()
{
    uint32_t mode, handle;
    int32_t ordered_arrival;
    uint8_t tmi_id;
    uint8_t ws_dst[DMA_SIZE];
    enum speed_t speed;
    enum flow_t flow;
    struct flow_tracker *tracker;
    uint32_t loops, i;
    uint16_t checksum;
    uint64_t mutex;
    uint32_t scopeid;
    ;; // end of variable declarations
    handle = TMAN_GET_TIMER_HANDLE(0x60);
    tmi_id = (uint8_t)handle;
```

Figure 7. entry_point.c source file displaying breakpoints

5 AIOP OS awareness features

All AIOP specific debug features can be enabled and customized from the **OS Awareness** tab. You can open this tab by selecting, **Run > Debug Configurations** from the IDE menu bar, then select **Debugger > OS Awareness** tab from the right panel of the **Debug Configurations** dialog.

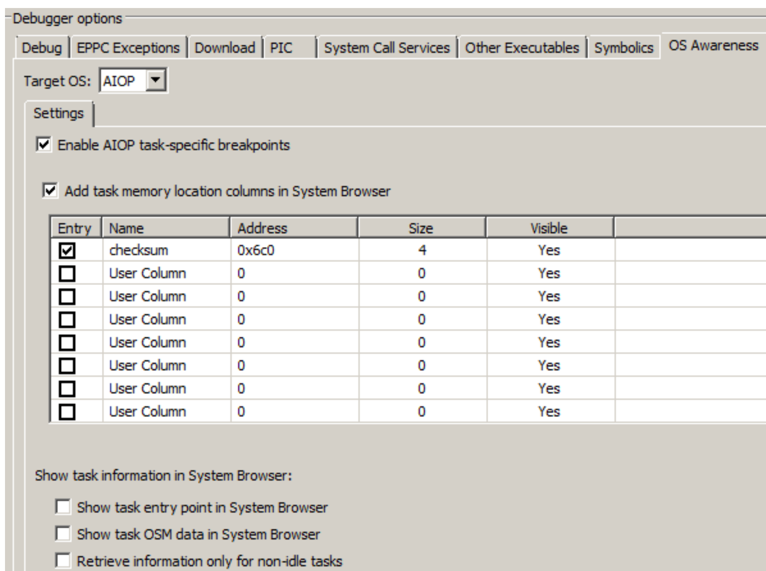

Figure 8. OS Awareness tab

The table below lists and describes the options available in OS awareness page:

Table 2. OS Awareness tab options

Option	Description
Target OS	Activates AIOP task awareness services.

Table continues on the next page...

Table 2. OS Awareness tab options (continued)

Option	Description
Enable AIOP task-specific breakpoints	Controls the support for task-specific AIOP breakpoint types.
Add task memory location in System Browser	Adds the possibility of defining and selecting up to eight memory locations that can be displayed as columns in the **System Browser** view.
Show task entry point in System Browser	Controls the display of the following AIOP task information in the **System Browser** view: • Task entry point • Task OSM data
Show task OSM data in System Browser	Displays the OSM data for the AIOP task in the **System Browser** view.
Retreive information only for non-idle tasks	Improves performance in the **System Browser** view by retrieving just the information about non-idle tasks.

You can view the list of the tasks, the scheduling state for every task and also the following custom settings in the **System Browser** view:

- System Browser view default columns
- Targeting specific AIOP task
- Task specific actions
- Enabling task memory locations
- Adding OSM data and entry point columns
- Viewing and interpreting OSM data and entry point information

5.1 System Browser view default columns

In the AIOP suspended state, you can view all tasks created in the the **System Browser**.

Figure 9. System Browser view

By default, the **System Browser** view displays the following columns:

Table 3. System Browser view columns

Column name	Description
Task ID	Displays the AIOP task id number. This column cannot be hidden from the view.

Table 3. System Browser view columns

Core	Displays the core number of the task.
PC	Displays the current program counter of the task.
Status	Displays the status of the task. Following is the list of supported values: • Idle • Allocated • Ready to execute • Executing • Accelerator job requested • Executing on accelerator • Ready to execute, inhibited • Accelerator job requested, inhibited • Executing on accelerator, inhibited
Accel ID	Displays the accelerator id number (if available) of a task called.

5.2 Targeting specific AIOP task

Each task is described in **Debug** view with its ID and the core number that it pertains to (as listed in the **System Browser**) the AIOP state is embedded to either, *Halted* or *Running*. You can target a specific AIOP task using any of the methods listed below:

- The default task (0x0) is automatically targeted in all cases.
- On hitting a breakpoint the task is automatically targeted and presented in **Debug** view. The task needs to be the first one which hit that breakpoint.
- To manually target a task (in order to inspect the stack, set a task specific breakpoint, etc.), you need to double-click on the task row in the **System Browser** view.

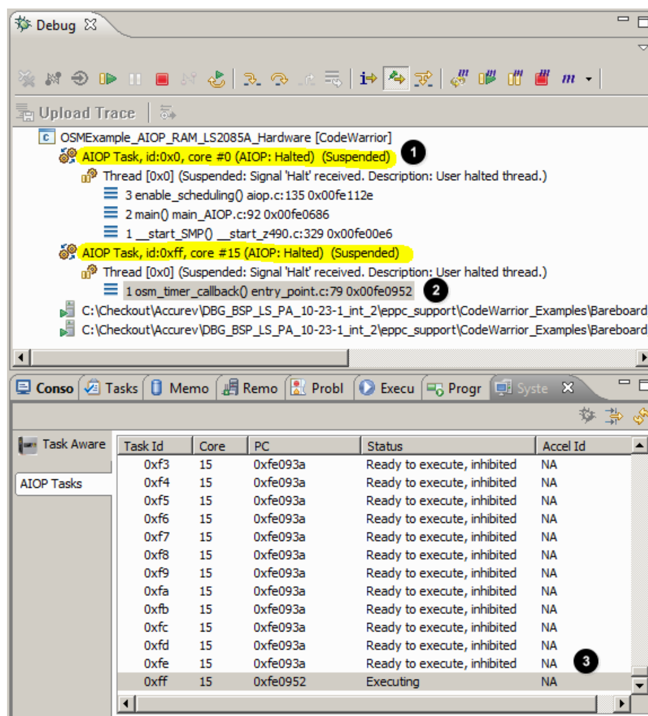

Figure 10. Targeting specific AIOP task

5.3 Task specific actions

While debugging an AIOP target with task awareness services activated, following targeting actions can be performed on the tasks:

Read/Write Registers

1. Target another process from the **System Browser** view.
2. Perform some read/write operations on the task specific data including the entire GPR set) from the **Register** view. You can observe different GPR set for each process.

Read/Write Memory

1. Target another process from the **System Browser** view.
2. Open **Memory** view.
3. Perform some read/write operations on the task specific data and on shared memory.

   You can view the `LCF/LS2085A_AIOP_RAM.lcf` from the `OSM` project to see which sections are shared and which sections are specific to each task.

Per-Task Global Variables

1. Target another process from the **System Browser** view.
2. For both the processes, open the **Variables** view and add a global variable that has been declared as per-task variable, using `__declspec(section ".tdata")`.

```
12
13 __declspec(section ".tdata") unsigned int T1 = 9;
14
```

Figure 11. Editor view displaying __declspec(section ".tdata")

You can observer that the variables have the same virtual address but they point to different physical addresses and in consequence have different values.

5.4 Enabling task memory locations

You can add task memory location columns in the **System Browser** view as follows:

1. Select **Add task memory location columns in System Browser** from the **OS Awareness** tab.
2. To add a new column entry, select a specific address for WS RAM (task specific). In the example listed below, the **checksum** variable/entry was added from the task entry point, `osm_timer_callback()` function and the address (0x6c0) was extracted from the **Variables** view when the execution reached the entry point.

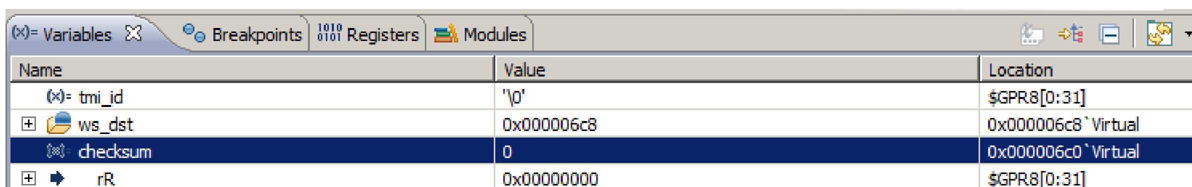

Figure 12. Variable view displaying selected checksum variable

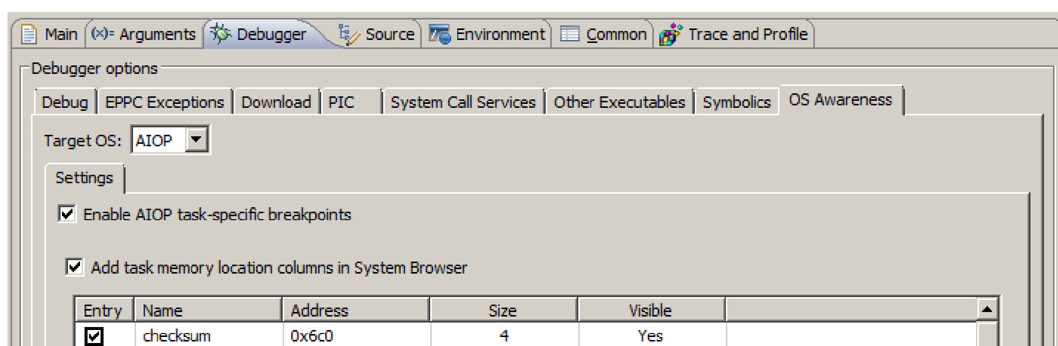

Figure 13. OS Awareness tab displaying checksum entry

3. In the debug session, you can view the value of the **checksum** value directly from the **System Browser** view.

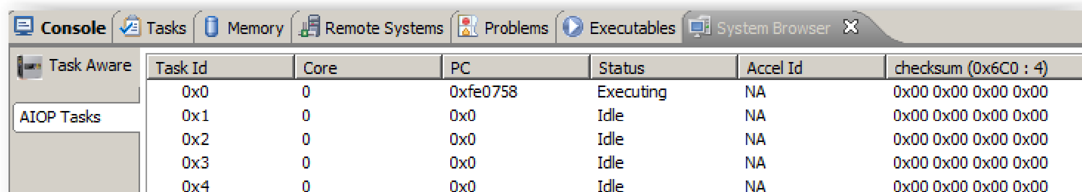

Figure 14. System Browser view with checksum column

5.5 Adding OSM data and entry point columns

From the **OS Awareness** tab, you can activate task entry point and OSM data columns.

Show task information in System Browser:

☑ Show task entry point in System Browser

☑ Show task OSM data in System Browser

☐ Retrieve information only for non-idle tasks

Figure 15. Show task information in System Browser options

You can enable the columns from the **System Browser** view during the debug session. To enable, right-click in **System Browser** table and select **Add/Remove columns > OSM [State, XPOS, TPOS]:SCOPE_ID**.

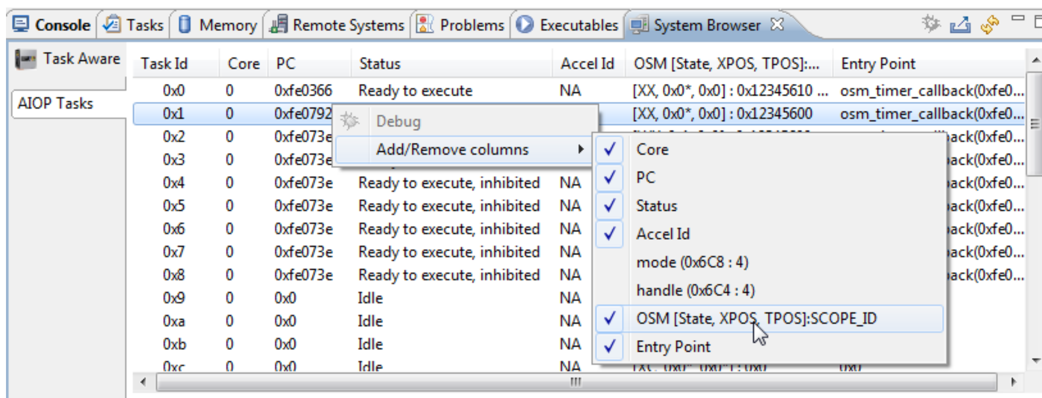

Figure 16. System Browser view - Adding OSM data and entry point columns

5.6 Viewing and interpreting OSM data and entry point information

The example below shows:

- task 0xef is currently executing (XX).
- task 0xff is not exclusive (XC) anymore. This task is also the first in line to exit scope.
- tasks 0x3f ..0xdf are in waiting for exclusivity (WX). Those tasks are set to **Ready to execute, inhibited** state by the OSM, no task-specific breakpoint is hit.
- task 0xfe is not scheduled yet.

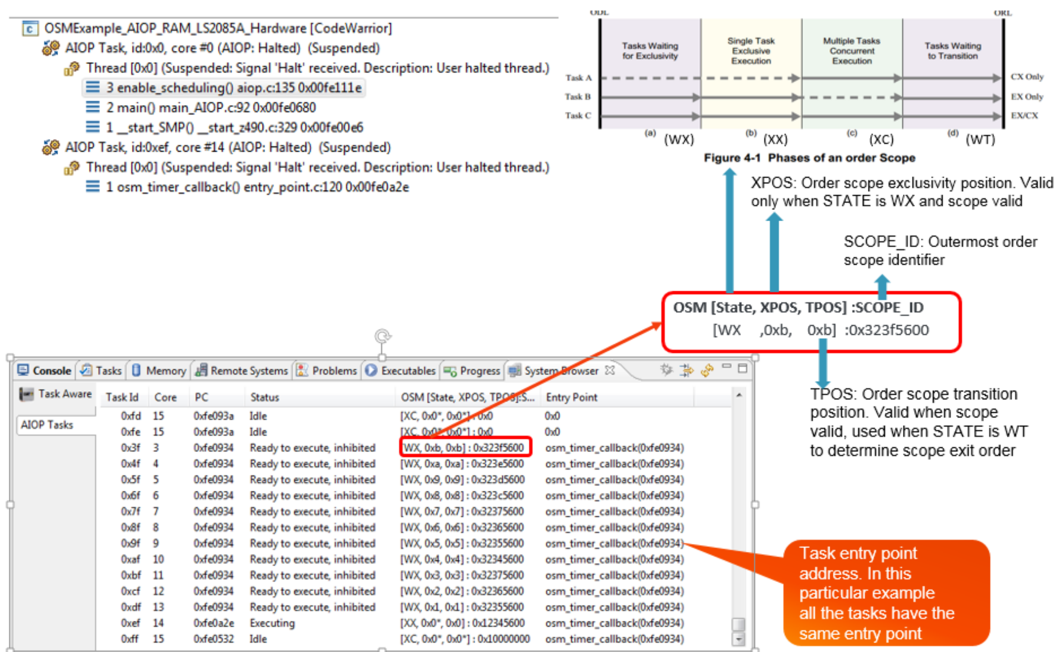

Figure 17. Viewing and interpreting OSM data and entry point information

6 AIOP task specific breakpoints

A finite number of tasks can exist and execute simultaneously inside an AIOP. The life cycle of an AIOP task is usually very short, and at any given point of time, on a core, so there may be tasks from different sources and different networking protocols. AIOP specific breakpoints provide a mechanism for debugging only the selected tasks, without affecting other tasks.

AIOP task specific breakpoints can be set via debugger shell or directly from GUI (source files).

6.1 Enabling AIOP task-specific breakpoints

The AIOP task-specific breakpoints support is enabled by default in CodeWarrior, but in case you want to disable/enable it, you can do this by selecting **Debugger > OS Awareness** tab from the **Debug Configurations** dialog, then check/clear the **Enable AIOP task-specific breakpoints** option, as the following figure shows:

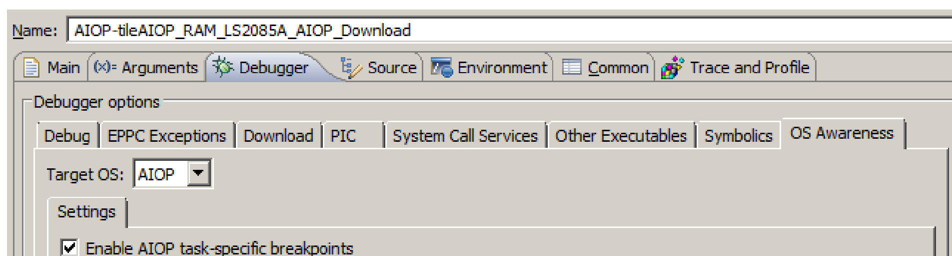

Figure 18. Enable AIOP task-specific breakpoints option

Currently, following five types of AIOP specific breakpoints are supported, as listed below:

Table 4. AIOP specific breakpoints

Type	Scope	Effect	Description
AIOP, Any Task, Global Halt, Software Breakpoints	Any Task	Global Halt	Any task may hit the breakpoint and when it does it triggers the AIOP global halt.
AIOP, Any Task, Global Halt, Hardware	Any Task	Global Halt	Any task may hit the breakpoint and when it does it triggers the AIOP global halt. This method uses a hardware breakpoint.
AIOP, Any Task, Task Halt, Software	Any Task	Task Halt	Any task may hit the breakpoint and when it does it is suspended from execution.
AIOP, One Task, Global Halt, Software	Current Task	Global Halt	Only current task may hit the breakpoint and when it does it triggers the AIOP global halt.
AIOP, One Task, Task Halt, Software	Current Task	Task Halt	Only current task may hit the breakpoint and when it does it is inhibited for scheduling for debug purposes.

The AIOP breakpoint types can be categorized as follows:

- TASK specific
  - A kind of software breakpoints that are used to stop tasks (not cores) from executing in an non-intrusively way, without halting the AIOP system.
  - To make this possible tasks are placed in debug inhibited for scheduling mode. Not to confuse with inhibited for scheduling mode set by Ordering Scope Manager (OSM).
  - Task specific breakpoints are used to stop a single task or all tasks that reach a certain point in code.
  - Task specific breakpoints are used to step tasks without interfering with other Core Task Schedulers in the AIOP system.
- GLOBAL HALT
  - These breakpoints behaves like normal code breakpoints and they cause global halt on the entire AIOP system, that is stop all CTS from running.
  - Global halt breakpoints are hardware breakpoints and task specific software breakpoints, set for a certain task or all tasks that reach a part of code.
  - They are used as default for stepping while tasks are not scheduled (example, during CTS initialization or inside `main()`)

6.2 Setting AIOP task specific breakpoint from source files

To.set an AIOP task specific breakpoint from source files. follow these steps:

1. Open a source file.
2. Select the breakpoint type, to do this, right-click on the bar from the left side of the source file and select **Breakpoint Types** from the context-menu.

NOTE

This setting is global and any further breakpoint set from the source files will be installed with this property.

3. Then select the type of the breakpoint you want to set.

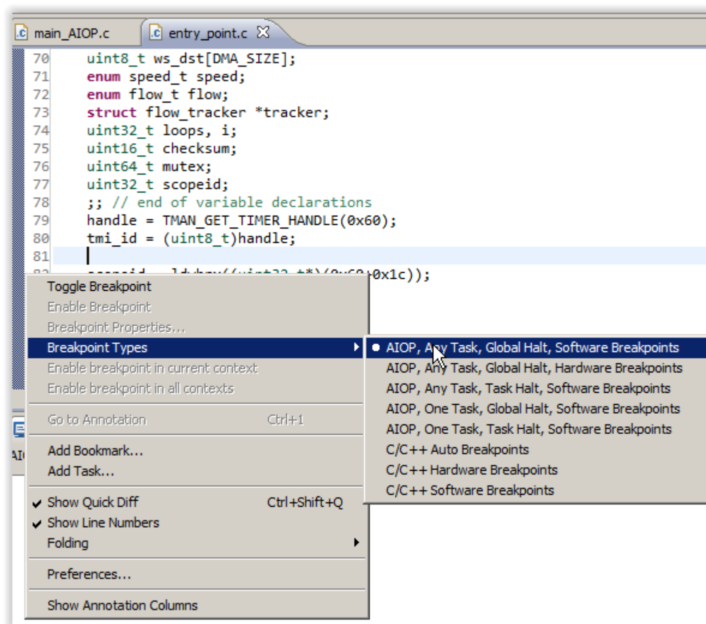

Figure 19. Editor view - Breakpoint Types option

4. Double-click on the line on which you want the breakpoint to be set. To demonstrate the AIOP per task breakpoints capabilities, the breakpoints need to be set in the entry point, that is, `osm_timer_callback()`.

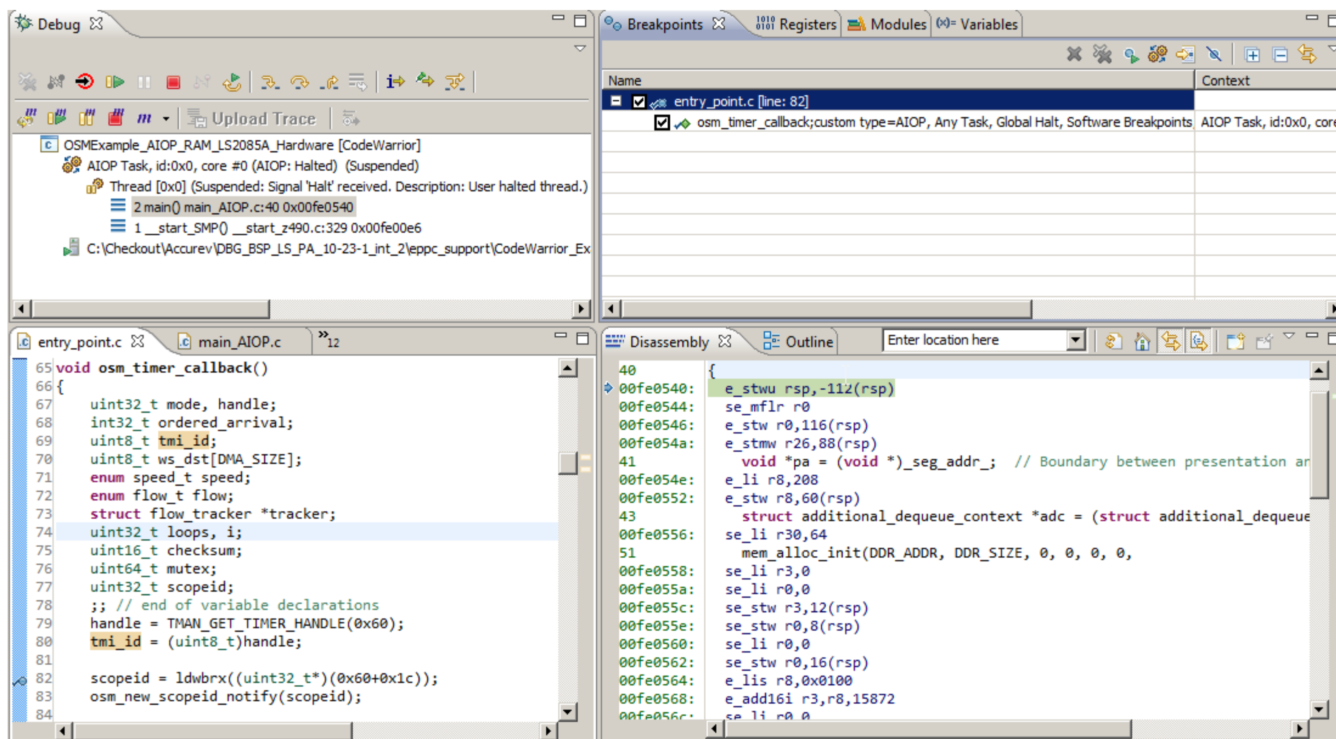

Figure 20. Editor view - Breakpoint Types option

5. Now you can run the project, the behavior for information about each specific AIOP per-task breakpoint see Enabling AIOP task-specific breakpoints.

6.3 Setting AIOP task specific breakpoint using debugger shell

To set an AIOP task specific breakpoint using debugger shell, follow these steps:

1. Select **Window > Show view > Debugger Shell** from the IDE menu bar.

   The **Debugger Shell** view appears.

2. Type `bp -custom` to see all available AIOP task specific breakpoint types.

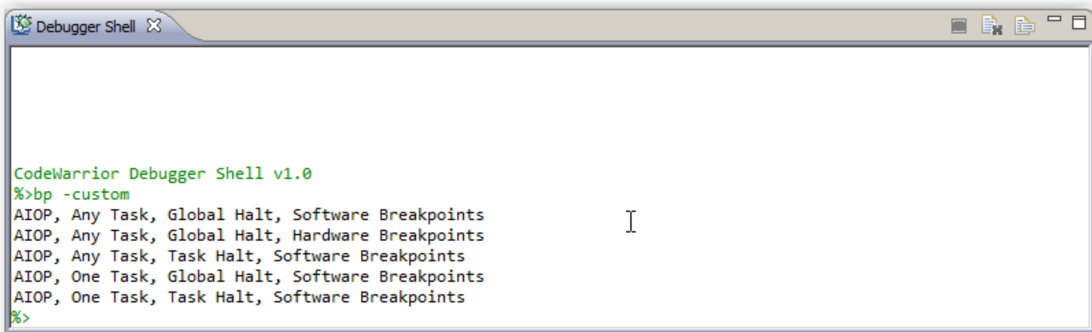

Figure 21. Debugger Shell view displaying all available AIOP task specific breakpoint types

3. In the debugger shell, type next command: `bp -custom "BreakpointType" func`
   Where,
   - `BreakpointType` is one of the five types presented above
   - `func` is the desired function/symbol name where you want to place the breakpoint

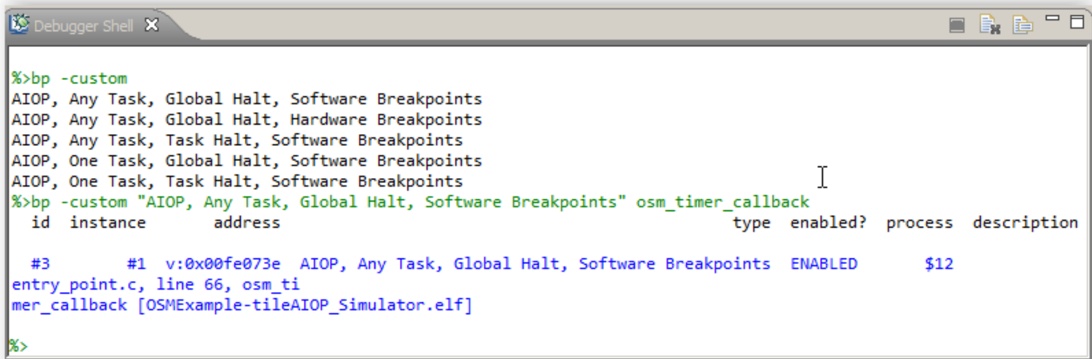

Figure 22. Debugger Shell - bp -custom command

(x)= Variables | Breakpoints | Registers | Modules

Name	Context	Address	Type
entry_point.c [function: osm_timer_callback]			
osm_timer_callback;custom type=AIOP, Any Task, Global Halt, Softwa	AIOP Task, id:0x0, core #0 (AIOP Tile: Halted)	Virtual:0x00fe073e	Software

Figure 23. Breakpoints view

.

6.4 Example for using Breakpoint AIOP, Any Task, Global Halt

To reach the state in the initial state as the following figure shows, is enough to enter in debug as described in the Debugging OSM example project using CodeWarrior IDE section.

Any task that would uses this type of breakpoint halts the whole system.

Initial State:

1. Task 0x0 is at main function
2. Set a breakpoint of the type, **AIOP, Any Task, Global Halt** in `entry_point.c` (`osm_timer_callback()` `function - line 79`)
3. Click Multicore resume.

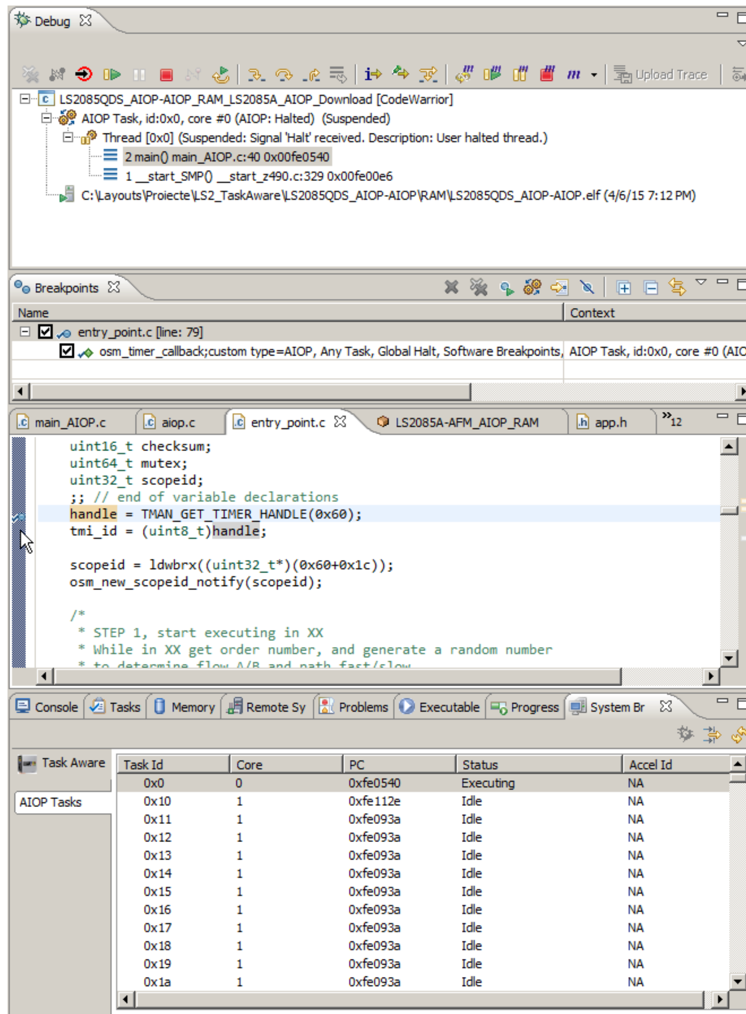

Figure 24. Initial state

Final State:

1. The AIOP is Halted (Suspended), a breakpoint is reached for task 0xff.
2. Task 0xff is targeted as result of reaching the breakpoint.
3. The whole system is halted.

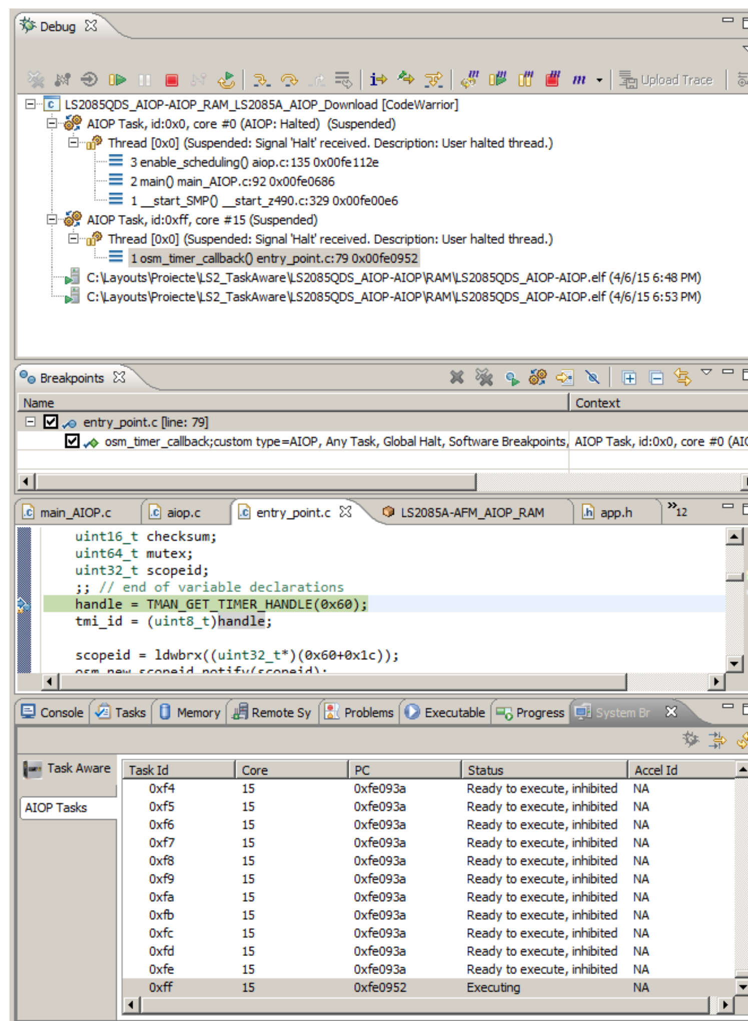

Figure 25. Final state

6.5 Example for using Breakpoint AIOP, Any Task, Task Halt

For this type of breakpoint, the task is inhibited without halting the system.

Initial State:

1. Task 0x0 is at main function.
2. Set a breakpoint of the type, **AIOP, Any Task, Task Halt** in `entry_point.c` (`osm_timer_callback() function - line 79`).
3. Click **Multicore resume**.

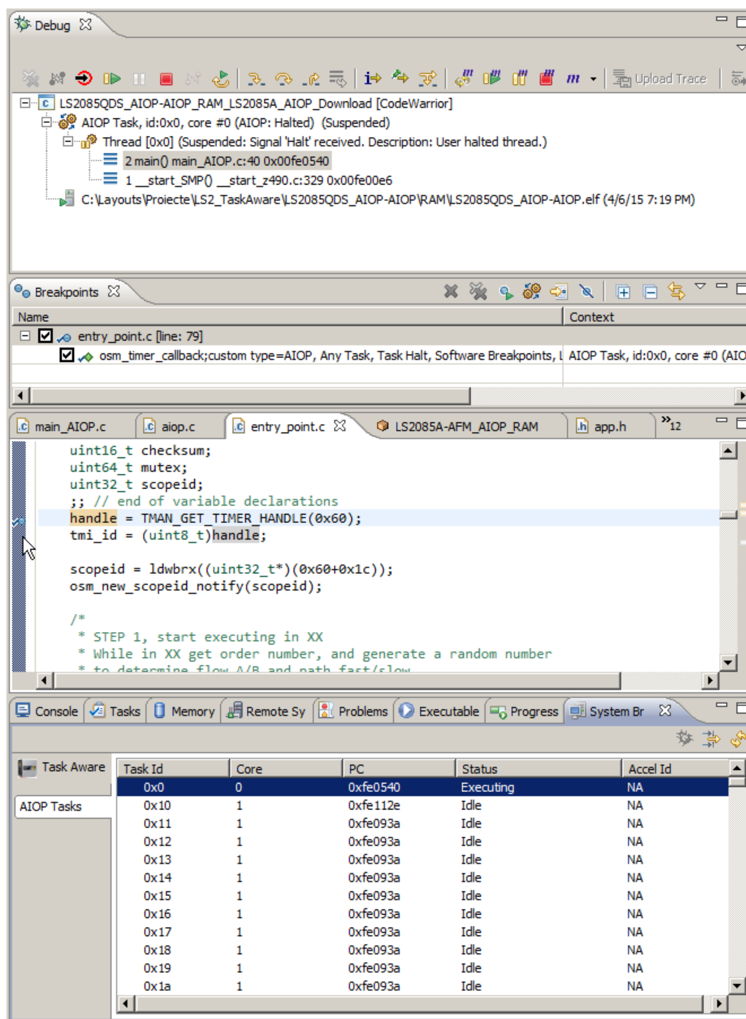

Figure 26. Initial state

Final State:

1. The AIOP is in Running state, a breakpoint is reached for task 0xff, and the system is NOT halted.
2. Task 0xff is targeted as result of reaching the breakpoint.
3. Task 0x0 is running because it did not reach the breakpoint. There are three supported scheduling order modes: unordered, concurrent, and exclusive. In this example, AIOP is running in exclusive mode (only one task has the privilege to run at a specific moment and the rest are inhibited by OSM until the lock is released). If the AIOP is running in concurrent mode (several tasks may run concomitantly), in such case, several tasks reach the breakpoint but only the task that reach the breakpoint *first* is targeted.

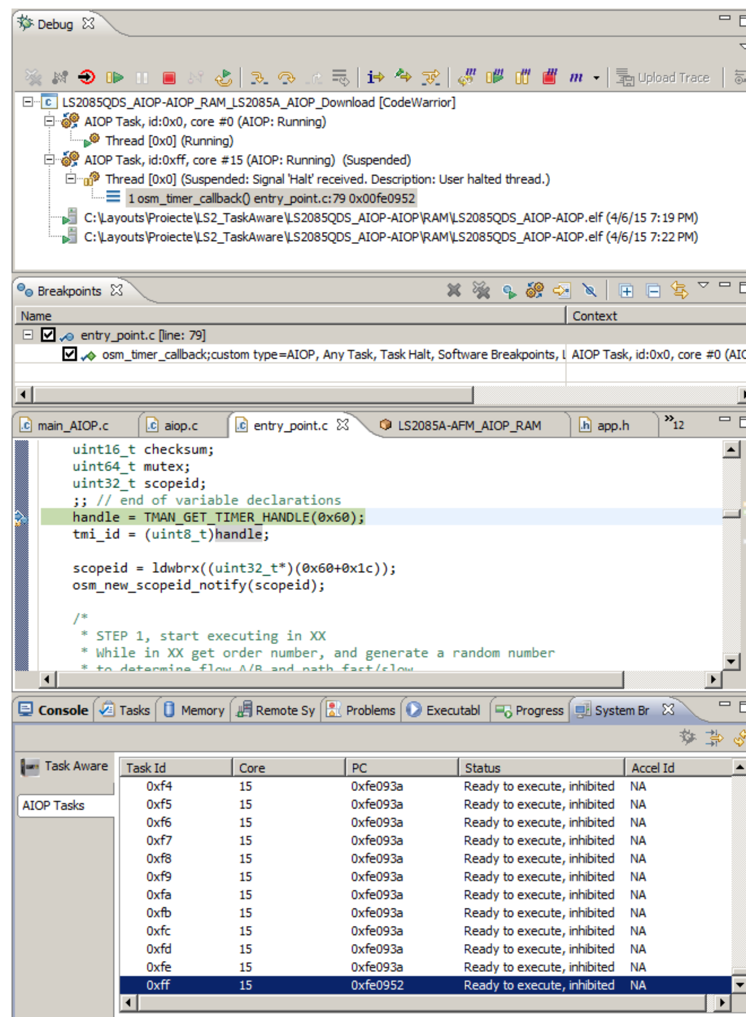

Figure 27. Final state

7 AIOP per-task stepping

The default step-in and step-over operations are using run-control operations per system. More exactly, when a step-over is used a temporary breakpoint is set and the entire system will be run. But, there is also support that emulates the per-task stepping – when the per-task stepping mode is active only the targeted task will make that step operation. For AIOP per-task stepping is used a temporary custom breakpoint – “AIOP, One Task, Task Halt, Software Breakpoints”.

The following section describes how you can enable or disable from debugger shell or GUI the AIOP per-task mode:

- Enabling AIOP per-task stepping from GUI
- Enabling AIOP per-task stepping from debugger shell
- Common task stepping issue
- Step per-task only on active tasks

7.1 Enabling AIOP per-task stepping from GUI

You can enable the AIOP per-task stepping mode from GUI using the **Task Stepping mode** button.

Figure 28. Task Stepping mode button

After a task step, that task will be shown in the **System Browser** as Ready to execute, inhibited.

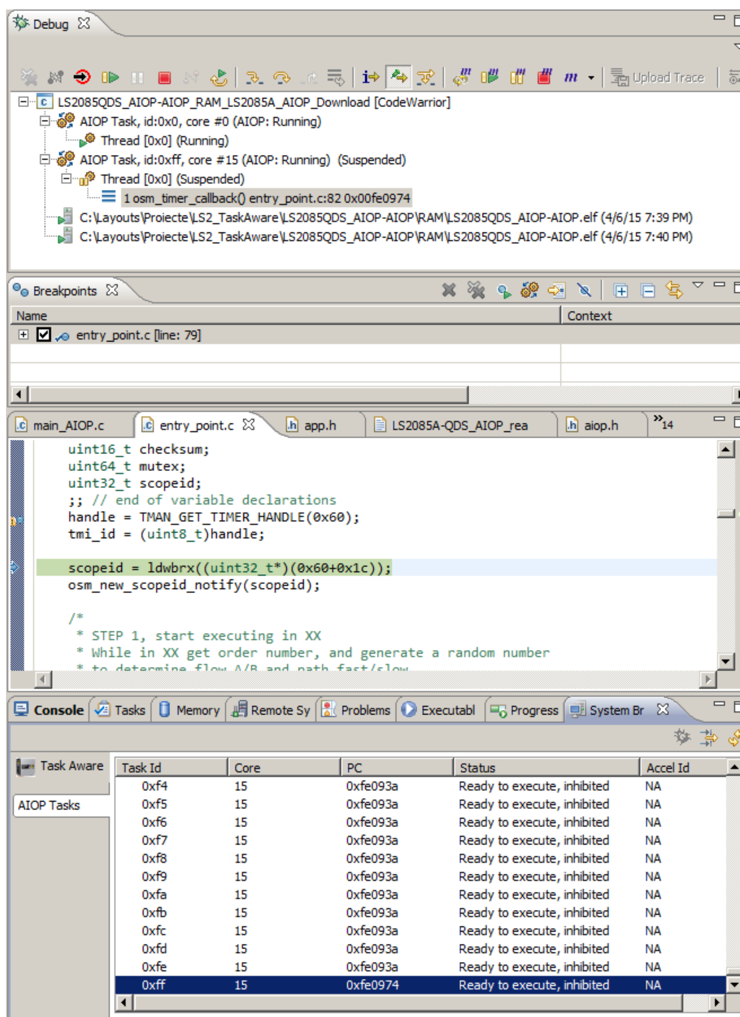

Figure 29. System Browser as Ready to execute, inhibited

7.2 Enabling AIOP per-task stepping from debugger shell

To enable the AIOP per-task stepping from the debugger shell, follow these steps:

1. Open **Debugger Shell** view by selecting, **Window > Show View > Debugger Shell** from the IDE menu bar.
2. Use the `cmdwin::eppc::taskstepmode` command to activate/deactivate task level stepping. It can be used as follows:
   - `cmdwin::eppc::taskstepmode` – gets the task stepping mode used
   - `cmdwin::eppc::taskstepmode on` – activates stepping at task level
   - `cmdwin::eppc::taskstepmode off` – deactivates stepping at task level; in this case, the debugger uses stepping at system level

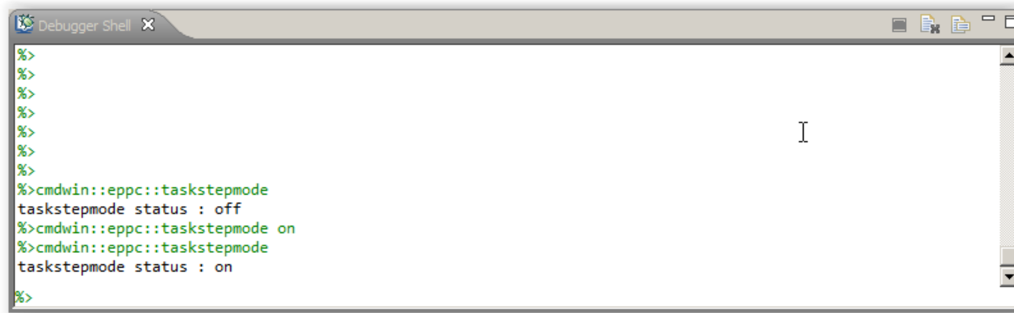

Figure 30. Debugger Shell view - cmdwin::eppc::taskstepmode command

7.3 Common task stepping issue

A common situation were a step can apparently fail is when a Global Halt breakpoint is left behind while stepping through a task:

1. Set a **Global Halt** breakpoint at `osm_scope_enter_to_exclusive_with_increment_scope_id`.
2. Click **Multicore resume**, a task reaches the breakpoint.

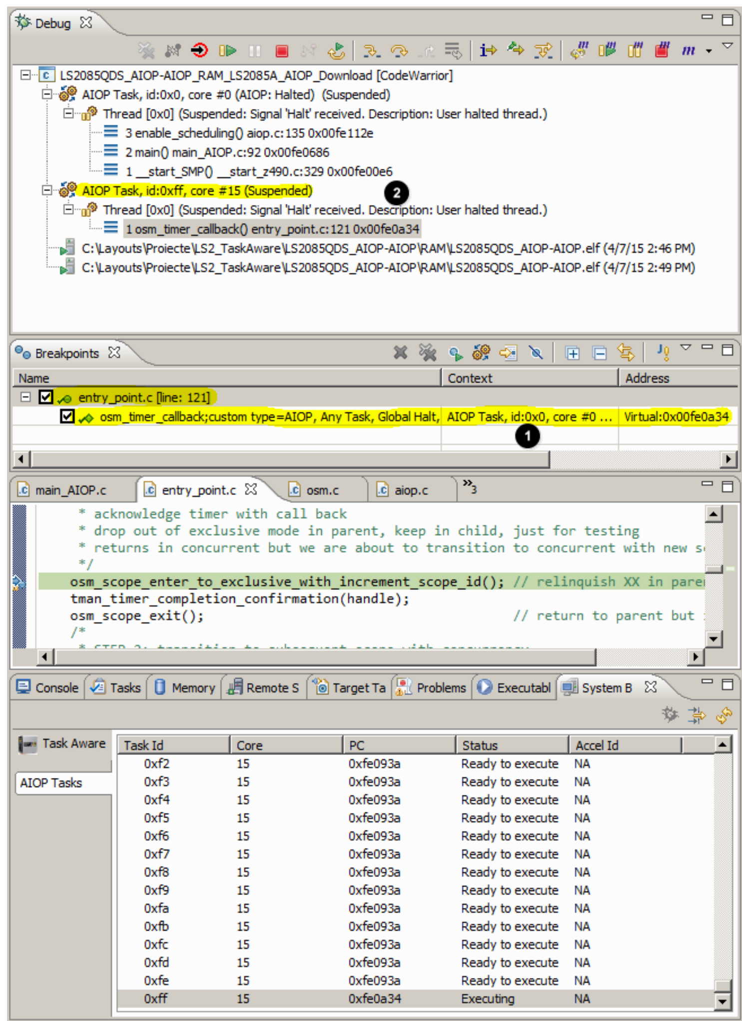

Figure 31. Multicore resume - task reaches the breakpoint

3. Delete the **Global Halt** breakpoint at **osm_scope_enter_to_exclusive_with_increment_scope_id** and set a **Global Halt** breakpoint in the entry point (at the beginning of `osm_timer_callback` function).
4. Single step task 0xff - the task apparently fails to reach the next line because another task just kicks in and triggers an AIOP global halt.
5. Task 0xff reaches the breakpoint from entry point.

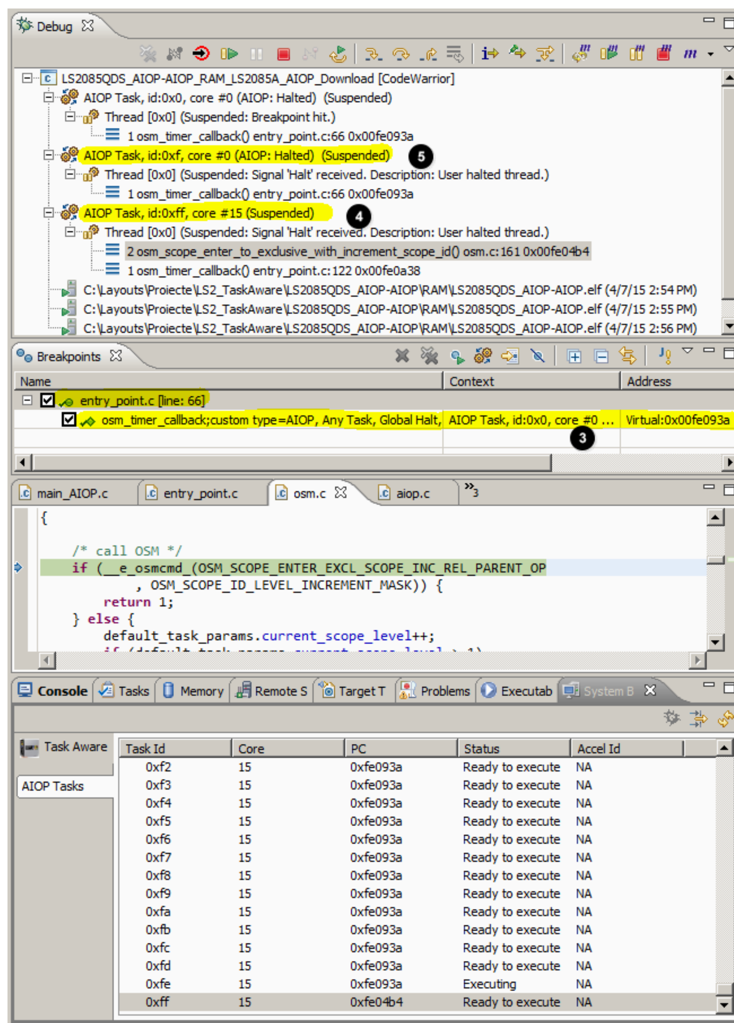

Figure 32. Task 0xff reaches the breakpoint from entry point

To avoid this situation use **Task Halt** breakpoints or just uninstall (do not delete them) any Global Halt breakpoints so they don't affect the stepping.

7.4 Step per-task only on active tasks

It is common for AIOP tasks to be preempted after executing a hardware accelerator call, so they cannot step on non-active tasks (Ready to execute, inhibited).

While IDE tries to step on this task, following error message appears:

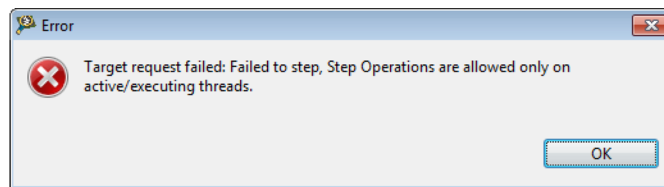

Figure 33. Error dialog

In this situation you need to wait until the task is scheduled again in order to continue stepping.

8 AIOP run-control operations

This section explains the AIOP run-control operations, and contains the following section:

- Resuming task execution

8.1 Resuming task execution

The following operations can be executed on the tasks, as shown by the figure listed below:

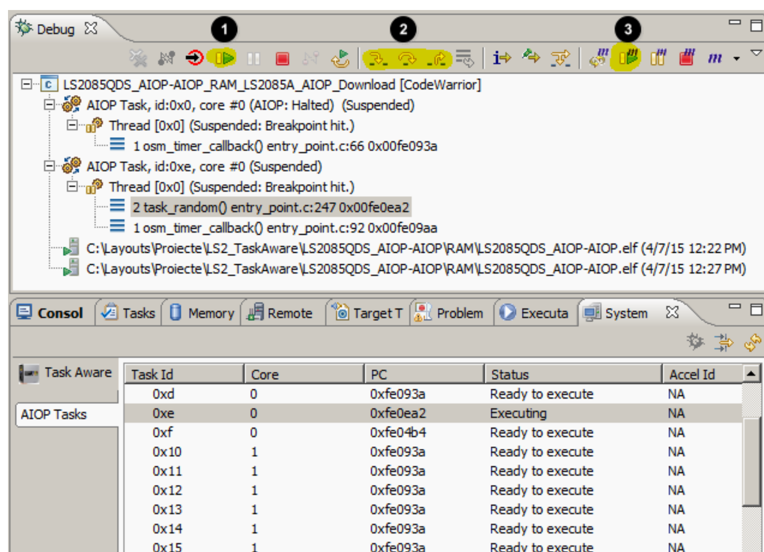

Figure 34. Resuming task execution

1. When the AIOP is Halted, resuming a single task results in an global resume similar to multicore resume. In Running mode, only the selected task continue execution (or brought out of debug inhibited for scheduling).
2. The stepping operations also resume execution of tasks, as in case of a single task continue.
3. Multi-core resume executes a global AIOP resume.

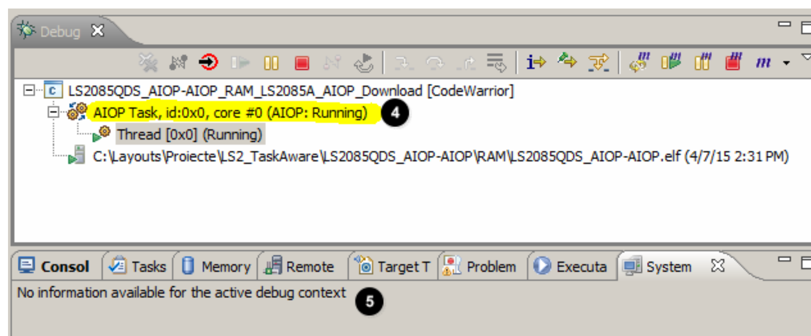

Figure 35. Multi-core resume

4. After a global resume all tasks are removed from the **Debug** view - except the default task. One exception is when single core resume is executed and the task was manually targeted from the **System Browser** view. On multicore resume this exception is not applicable. The targeted tasks from **Debug** view presents the AIOP state as Running.

5. The **System Browser** view shows only the debug inhibited for scheduling tasks. In AIOP Halted mode the tasks are assumed to be inhibited for scheduling.

9 Running OSM example on LS2 simulator

In case of using simulator connection, you need to install the Simulator package.

The simulator needs a Linux 64-bits machine, so if you are running the CodeWarrior for Linux you can run both CodeWarrior and simulator on the same machine.

If you're running the CodeWarrior for Windows, during installation a dialog appears on your screen to set the IP of the Linux 64-bits machine and the location where you want to install the simulator.

Simulator setup:

1. If you're running the CodeWarrior on a Linux machine, note that the simulator is already unpacked under `Common/CCSSim` folder, so you can skip to the *step 4*.
2. For running the CodeWarrior software on a Windows machine, if you did not make a remote installation of the simulator during your CodeWarrior installation process, you can get the same from the following location of the CodeWarrior layout:

   ```
   <CW_Layout>/Common/CCSSim/LS_SIM_RELEASE_0_x_0_xxxx_xxxxxx.tgz
   ```
3. Move the file to the Linux x86_64 machine and untar it.
4. Copy the following simulator initialization files from the `${CW_for_netAPP}/CW_APP/LS/CodeWarrior_Examples/Bareboard_Examples/AIOP_OSM_example` folder to `<CW_Layout>/Common/CCSSim`:
   - `LS2085A_system_test_cw_OSM.cfg`
   - `ls2085a_sim_init_params_OSM.cfg`
5. Start the `ccssim2` from `CCSSim` folder. In the example listed below, the debugging port is 40970 (this can also be set up in CodeWarrior launch configurations)

   ```
   ./ccssim2 -port <port> -smodel "ls_sim_config_file=LS2085A_system_test_cw_OSM.cfg" -
   imodel "ls_sim_init_file=ls2085a_sim_init_params_OSM.cfg"
   ```

CodeWarrior setup:

1. Select the Simulator target for the OSM example from the **CodeWarrior Projects** view.

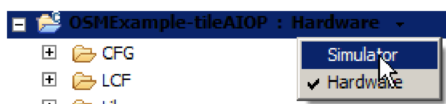

Figure 36. CodeWarrior Projects view

2. Right-click and select **Build Project**.

   The IDE builds the project.

3. Configure the connection settings. For connecting to simulator, you need to perform the following steps:
   a. Select **Run > Debug Configurations** from the IDE menu bar.

      The **Debug Configurations** dialog appears.

   b. From the left panel, select **OSMExample-AIOP_RAM_LS2085A_Simulator** launch configuration.
   c. From the right panel, click **Edit** from the **Target Settings** group and specify the IP of the Linux machine where the simulator is running and the debugging port used when starting the simulator.

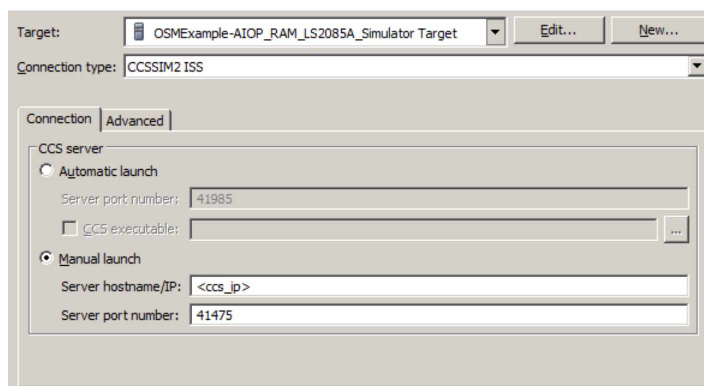

Figure 37. Debug Configurations dialog

4. Click **Debug**.

   All the other settings and features are similar to the hardware target.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. Freescale reserves the right to make changes without further notice to any products herein.

Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 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 does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, CodeWarrior, and QorIQ are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Layerscape is trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. ARM®, Cortex®, Cortex®-A53, Cortex®-A57, TrustZone® are trademarks of ARM Limited.

© 2014-2015, Freescale Semiconductor, Inc.

Document Number AN5044
Revision 05/2015