NXP Semiconductors
Application Note

Document Number: AN5165
Rev. 10.3.2, 08/2018

AIOP SDK Applications Debug

Contents

1 Overview....1
2 Prerequisites....2
3 Building AIOP reflector APP....2
4 Hardware setup....2
5 Importing and building AIOP reflector project....6
6 Debugging AIOP APP using CodeWarrior....8
6.1 Debugging AIOP from system entry point....11
6.2 Debugging AIOP from application entry point....13
7 Collecting hardware trace....15
7.1 GCov code coverage....17

1 Overview

This application note describes how to debug an AIOP SDK application with CodeWarrior for APP. The application targeted by this document is AIOP Packet reflector.

AIOP packet reflector provides an entry-level demonstration about how to use and program an AIOP. It has no predefined NXP infrastructure that is required to be used by the end user. It uses the AIOP SL-Service Layer routines only.

The purpose of this sample application is to demonstrate a simple application data path on AIOP. The application is available in these two flavors:

- A basic reflector for every IPv4 frame (further referenced as *Reflector*). It works much like the NADK Packet Reflector application, except that it runs on AIOP.
- The second one applies an extra classification and only accepted frames are further reflected (further referenced as *Reflector-Classifier*).

For more details about this application, see the *AIOP 'packet reflector' sample application* chapter of the *LS2085 SDK Quick Start Guide*.

This application note focuses on the *Reflector* flavor.

An updated version of the Application Note is available at CodeWarrior Development Suites for Networked Applications Product Summary Page.

2 Prerequisites

Before you debug an AIOP SDK application on CodeWarrior for App, ensure the following prerequisites.

NOTE

The references used in this application note are from a Linux 64-bit host machine for simulator. For hardware, you can use either Linux or Windows.

The table below shows the requisite components.

Component	Version
CodeWarrior for APP	10.2.0 or later
SDK	EAR6.0 or later
LSDK	17.12 or later

3 Building AIOP reflector APP

To get the latest AIOP APP source files, follow the steps from SDK documentation or from Layerscape-SDK documentation.

4 Hardware setup

To demonstrate the *reflected* traffic, you can use only one board with two ports connected back-to-back, as the following figure shows (in the example below, the copper ports 5 and 6 are connected):

Figure 1. Hardware setup using one board with two ports connected back-to-back

The Linux container role is played by the port 5 and the AIOP container role is played by the port 6.

```
LINUX                                                          AIOP
dpni.0  <-> dpmac.5 <------------------------------> dpmac.6 <-> dpni.1
(ni0)
```

After you get a U-Boot prompt on the board, use these commands:

Bring up the board via tftp from U-Boot (or you can write the images to the flash using the flash programmer from CodeWarrior for ARMv8).

```
setenv filesize; setenv myaddr 0x580100000; tftp 0x80000000 u-boot-nor.bin; protect off
$myaddr +$filesize; erase $myaddr +$filesize; cp.b 0x80000000 $myaddr $filesize; protect on
$myaddr +$filesize

setenv filesize; setenv myaddr 0x580000000; tftp 0x80000000 PBL.bin; protect off $myaddr +
$filesize; erase $myaddr +$filesize; cp.b 0x80000000 $myaddr $filesize; protect on $myaddr +
$filesize

setenv filesize; setenv myaddr 0x580300000; tftp 0x80000000 mc.itb; protect off $myaddr +
$filesize; erase $myaddr +$filesize; cp.b 0x80000000 $myaddr $filesize; protect on $myaddr +
$filesize

setenv filesize; setenv myaddr 0x580700000; tftp 0x80000000 dpl-eth.0x2A_0x41.dtb; protect
off $myaddr +$filesize; erase $myaddr +$filesize; cp.b 0x80000000 $myaddr $filesize; protect
on $myaddr +$filesize

setenv filesize; setenv myaddr 0x580800000; tftp 0x80000000 dpc-0x2a41.dtb; protect off
```

```
$myaddr +$filesize; erase $myaddr +$filesize; cp.b 0x80000000 $myaddr $filesize; protect on
$myaddr +$filesize
```

Prepare target for AIOP application

```
fsl_mc start mc 580300000 580800000 && fsl_mc apply dpl 580700000
tftp a0000000 kernel-ls2085ardb.itb
bootm a0000000
```

NOTE

bootargs needs to contains minimal parameters in order to have a correct setup for AIOP application. Make sure `bootargs=console=ttyS1,115200 root=/dev/ram0 earlycon=uart8250,mmio,0x21c0600 ramdisk_size=0x2000000 default_hugepagesz=2m hugepagesz=2m hugepages=256`

Configure the ni0 interface and create a static ARP entry. Set the destination MAC as the ARP hardware address for all the IP flows on which the packet needs to be sent:

```
$ ifconfig ni0 6.6.6.1 up
$ arp –s 6.6.6.10 000000000006
```

Prepare the AIOP container using the following steps:

1. Run the following script on the linux target.

   `<yocto_path>/build_ls2085ardb_release/tmp/work/aarch64-fsl-linux/aiopapp-refapp/scripts/dynamic_aiop_root.sh`

2. Delete the lines between 205 and 225 and update `DPMAC1="dpmac.6"`.
3. Copy the script and the `aiop_reflector.elf` on the linux target using `scp` from the linux host and the `eth0` (connected to e1000#0 PCI card) interface.

On the linux target:

```
$ ifconfig eth0 192.168.1.2 up
```

On the linux host:

```
$ ifconfig eth0 192.168.1.1 up
$ scp <yocto_path>/build_ls2085ardb_release/tmp/work/aarch64-fsl-linux/aiopapp-refapp/
scripts/dynamic_aiop_root.sh root@192.168.1.2:.
$ scp <yocto_path>/build_ls2085ardb_release/tmp/work/aarch64-fsl-linux/aiopapp-refapp/demos/
reflector/out/aiop_reflector.elf root@192.168.1.2:.
```

On the linux target:

```
root@ls2085ardb:~# chmod +x dynamic_aiop_root_test.sh
root@ls2085ardb:~# ./dynamic_aiop_root_test.sh
Creating AIOP Container
Assigned dpbp.1 to dprc.2
Assigned dpbp.2 to dprc.2
Assigned dpbp.3 to dprc.2
Assigned dpni.1 to dprc.2
Connecting dpni.1<------->dpmac.6
AIOP Container dprc.2 created
----- Contents of AIOP Container: dprc.2 -----
dprc.2 contains 4 objects:
object          label           plugged-state
dpni.1                          plugged
dpbp.3                          plugged
dpbp.2                          plugged
dpbp.1                          plugged
-----
=====================================================
Creating AIOP Tool Container
Assigned dpaiop.0 to dprc.3
Assigned dpmcp.22 to dprc.3
AIOP Tool Container dprc.3 created
```

```
----- Contents of AIOP Tool Container: dprc.3 -----
dprc.3 contains 2 objects:
object          label           plugged-state
dpaiop.0                        plugged
dpmcp.22                        plugged
-----
=====================================================
Performing VFIO mapping for AIOP Tool Container (dprc.3)
Performing vfio [  234.804575] vfio-fsl-mc dprc.3: Binding with vfio-fsl_mc driver
mapping for dprc.3
[  234.814384] vfio-fsl-mc dpaiop.0: Binding with vfio-fsl_mc driver
[  234.821209] vfio-fsl-mc dpmcp.22: Binding with vfio-fsl_mc driver
========== Summary ================================
 AIOP Container: dprc.2
 AIOP Tool Container: dprc.3
=====================================================
```

Load the AIOP application using `aiop_tool`.

Initiate ping on the interface to forward packets to the *Reflector* application running on the AIOP container board. Basically, this is a ping from ni0 interface (dpni.0 – dpmac.5) to dpni.1 – dpmac.6.

```
$ aiop_tool load -f aiop_reflector.elf -g dprc.3
AIOP Image (aiop_reflector.elf) loaded successfully.
$ ping 6.6.6.10
```

To check if the AIOP reflector application loaded successfully, execute the following command in the Linux command shell:

```
$ root@ls2085ardb:~# cat /dev/fsl_aiop_console
```

The command output displays the number of DPNIs that are successfully configured, together with the DPNIs that are provided to the AIOP Reflector Application:

```
REFLECTOR : Successfully configured ni0 (dpni.1)
REFLECTOR : dpni.1 <---connected---> dpmac.6 (MAC addr: 00:00:00:00:00:06)
> TRACE [CPU 0, dpci_drv.c:524 dpci_event_handle_removed_objects]: Exit
> INFO [CPU 0, init.c:289 core_ready_for_tasks]: AIOP core 0 completed boot sequence
> INFO [CPU 0, init.c:295 core_ready_for_tasks]: AIOP boot finished; ready for tasks...
```

The AIOP Logger prints a brief information about every frame that is reflected, as listed below. You can also view these logs in the CodeWarrior IDE in a simple manner using the Debug Print feature. For more information about the Debug Print feature, see the *Debug Print Application Note*.

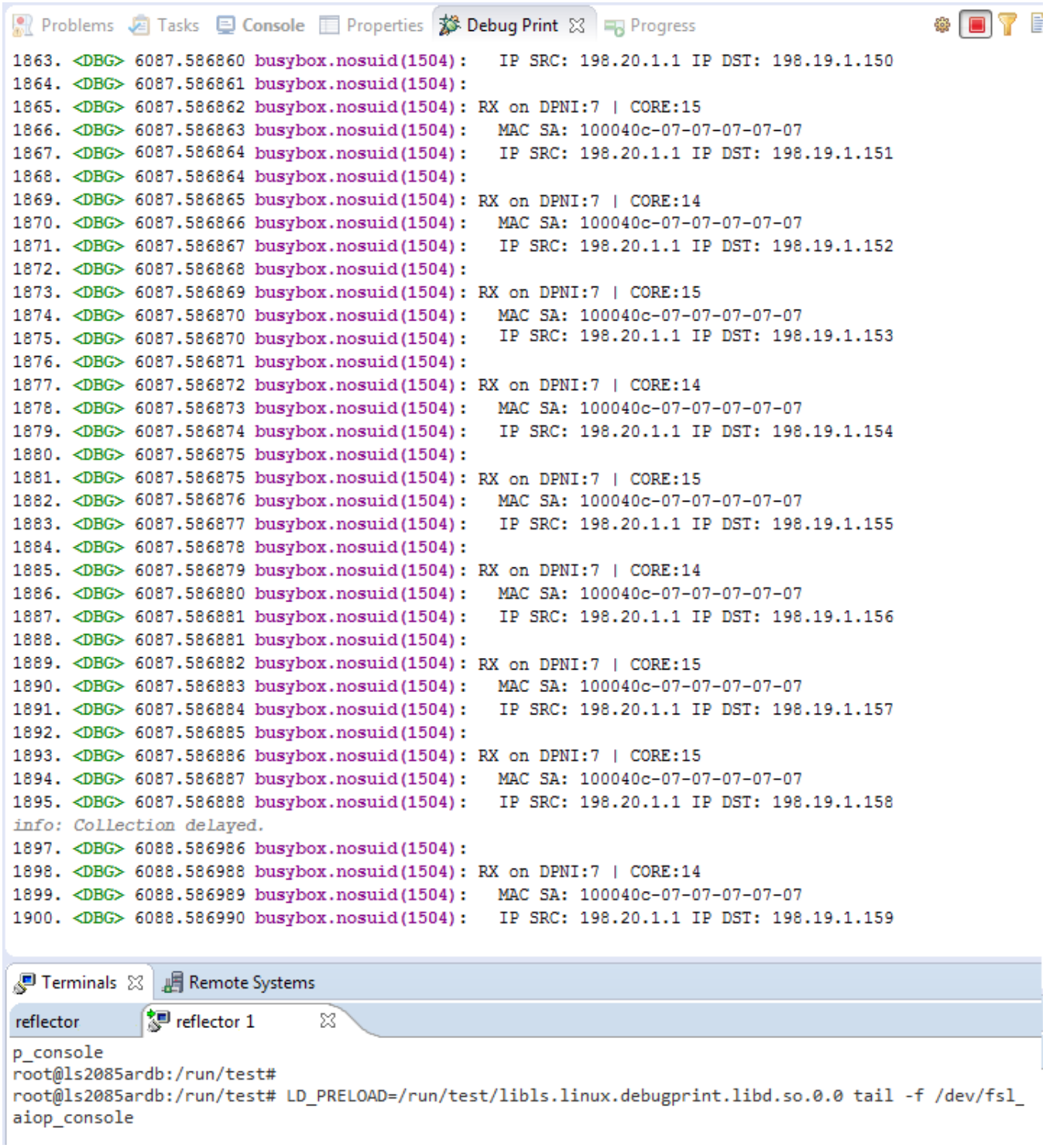
```
$ root@ls2085ardb:~# tail -f /dev/fsl_aiop_console


RX on DPNI 1 | CORE:15
  MAC_SA: 02-00-c0-a8-48-01 MAC DA: 00-00-00-00-00-06
  IP SRC: 6.6.6.1 IP DST: 6.6.6.10

RX on DPNI 1 | CORE:15
  MAC_SA: 02-00-c0-a8-48-01 MAC DA: 00-00-00-00-00-06
  IP SRC: 6.6.6.1 IP DST: 6.6.6.10

RX on DPNI 1 | CORE:15
  MAC_SA: 02-00-c0-a8-48-01 MAC DA: 00-00-00-00-00-06
  IP SRC: 6.6.6.1 IP DST: 6.6.6.10

RX on DPNI 1 | CORE:15
  MAC_SA: 02-00-c0-a8-48-01 MAC DA: 00-00-00-00-00-06
  IP SRC: 6.6.6.1 IP DST: 6.6.6.10
```



5 Importing and building AIOP reflector project

To import and build the AIOP reflector project, follow these steps:

1. Start the CodeWarrior and create a new workspace.
2. Import (**File > Import > General > Existing Projects Into Workspace**) the **reflector** and **aiop_sl** projects from this location: `<yocto_path>/build_<target>_release\tmp\work\aarch64-fsl-linux\aiopsl`

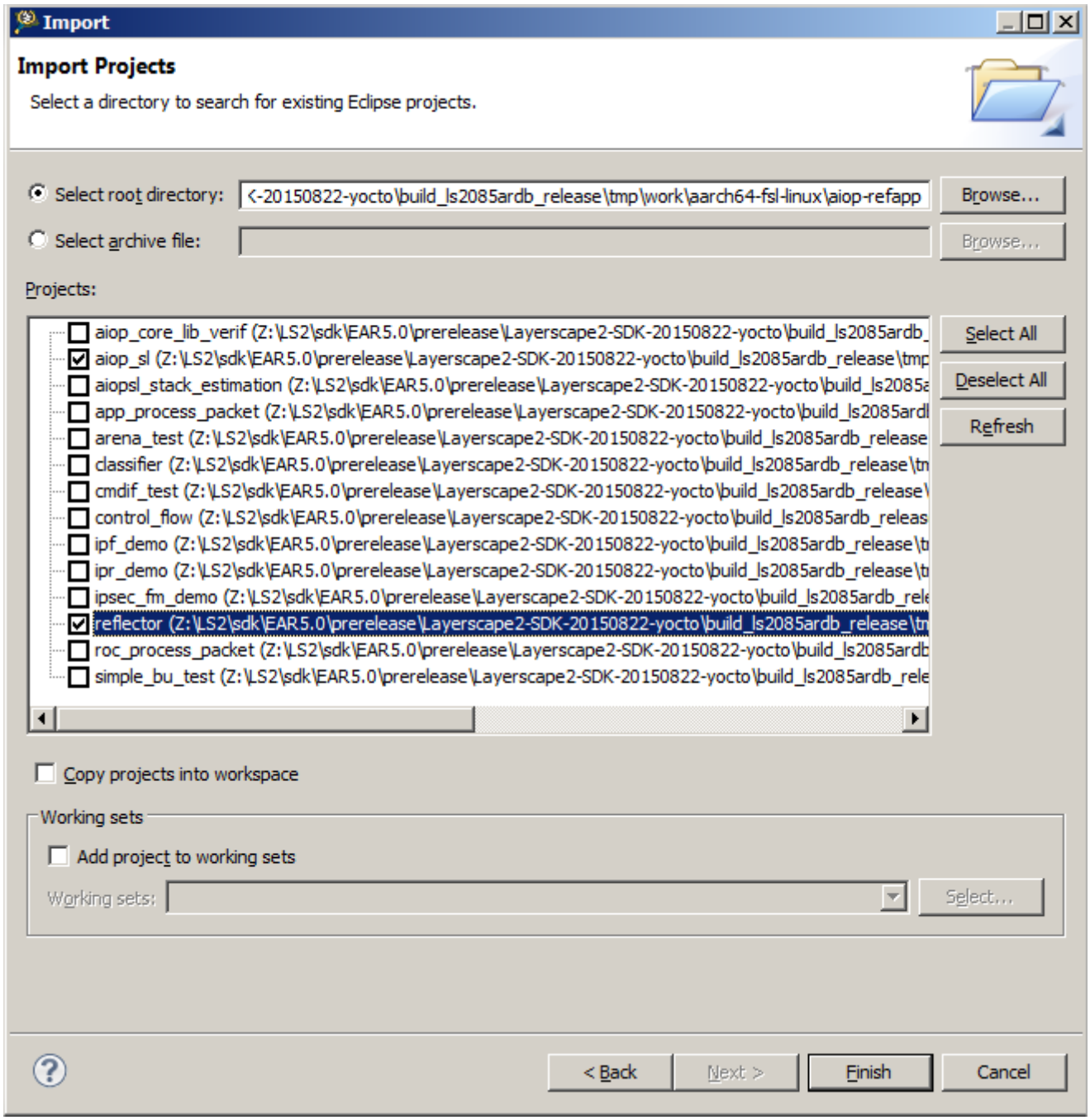


Figure 2. Import dialog - Import Projects page

3. The aiop reflector project (`aiop_reflector.elf`) is already built by Yocto, but if you want you can edit the sources and build the project directly from the CodeWarrior. To do this, right-click on the project in the **CodeWarrior Projects** view and select **Build Project**. The IDE also rebuilds the `aiop_sl` library project that is linked to the reflector project. It is recommended to use –O0 level optimization for improved debugging. To access **Optimization Level**, select **Project Properties > C/C++ Build > Settings > Compiler > Optimization > Optimization Level**.

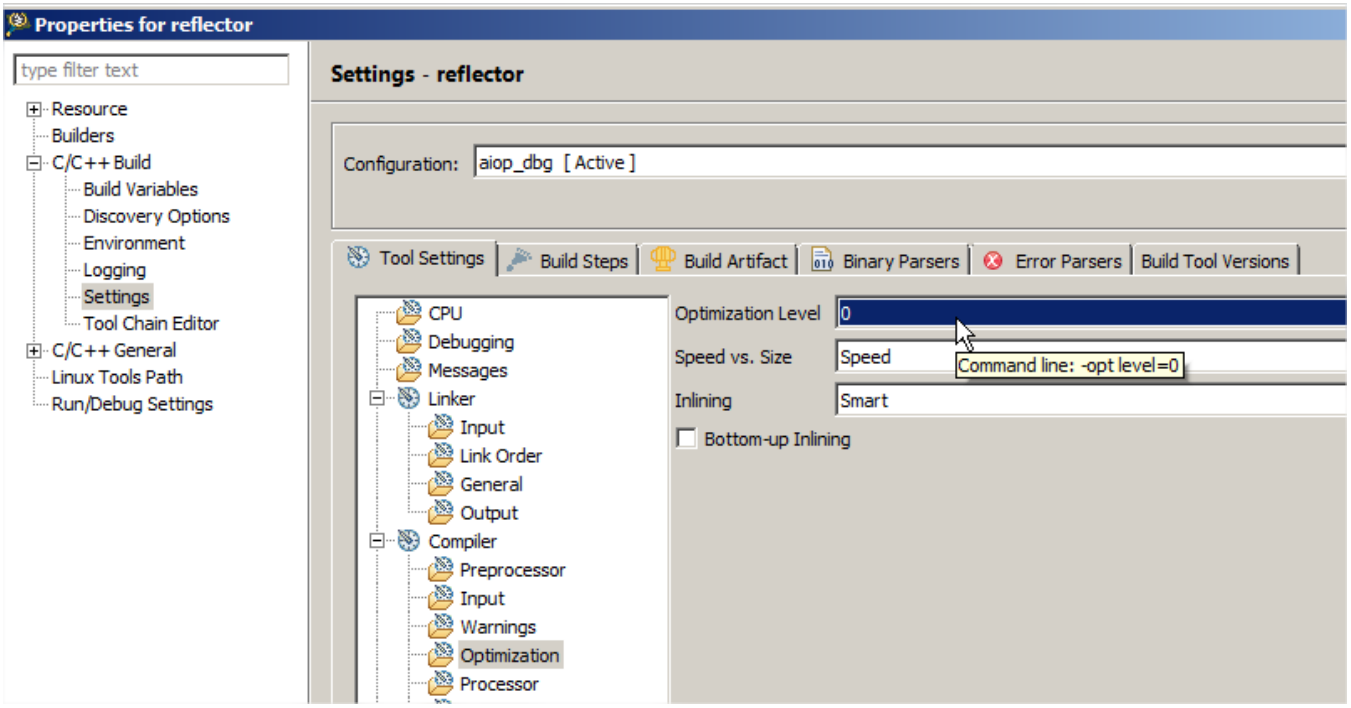


Figure 3. Properties for reflector project - Settings window

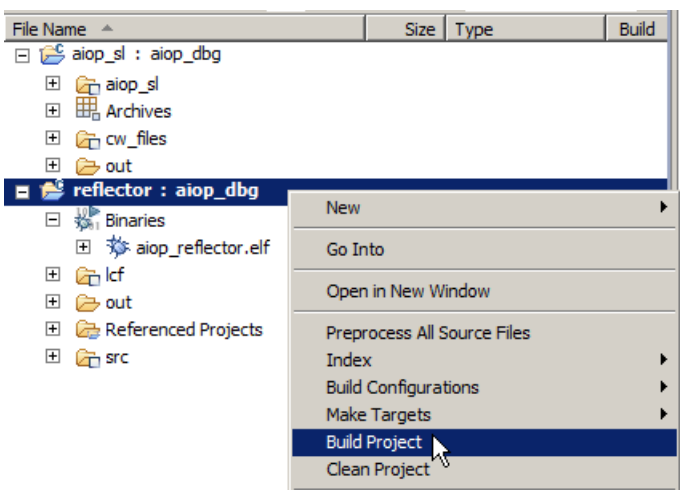


Figure 4. CodeWarrior Projects view - Build Project option

6 Debugging AIOP APP using CodeWarrior

To debug the AIOP using the CodeWarrior for APP IDE, follow these steps:

1. Copy the new `aiop_reflector.elf` just compiled with CodeWarrior or yocto to the linux board. To locate the elf, expand the **Binaries** group from reflector project, right click on the `aiop_app.elf` and select **Show in Windows Explorer** for Windows, or **Show in File Manager** for Linux.

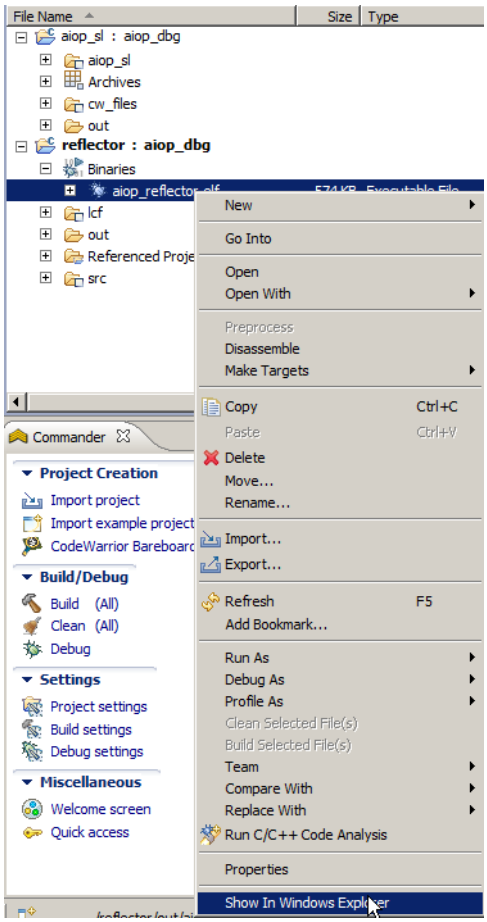


Figure 5. Show in Windows Explorer option

2. Select **Run > Debug Configurations** from the IDE menu bar.

   The **Debug Configuration** dialog appears.

3. Select the reflector project.
4. Select **aiop_dbg** launch configuration from the left panel.
5. Click **Edit** from **Connection**.
6. Specify the **Hostname/IP**.

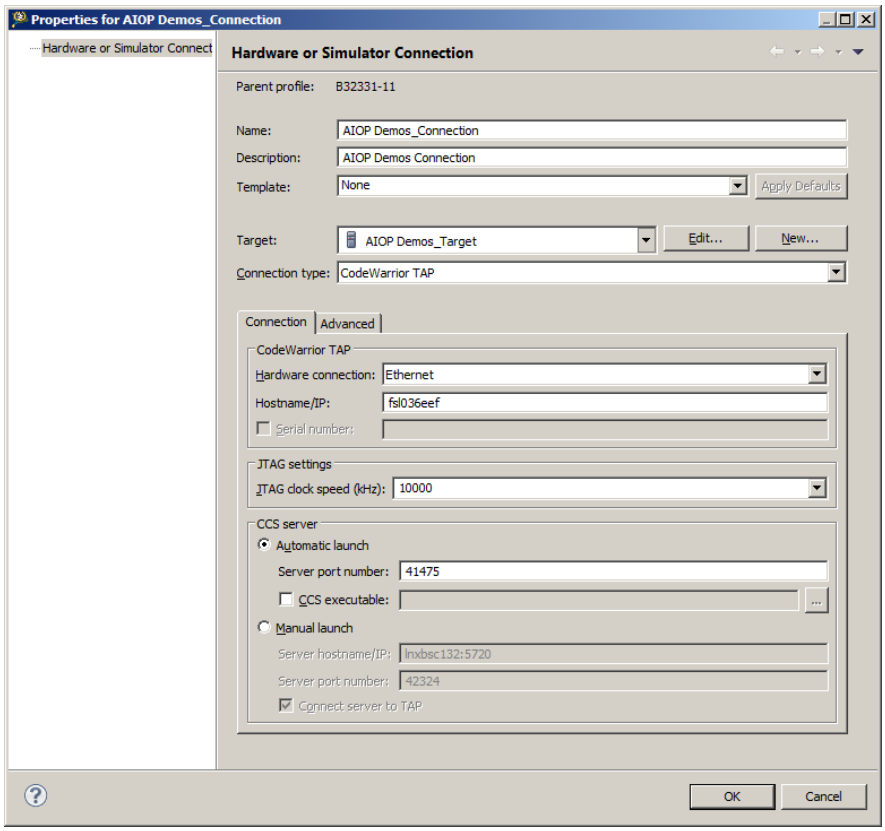


Figure 6. Properties for <connection> dialog - Hostname/IP option

7. Click **OK**.
8. Ensure that the AIOP OS awareness is enabled. To do this, open the **Debugger > OS Awareness** tabs and ensure that the **AIOP** is selected in the **Target OS** group.

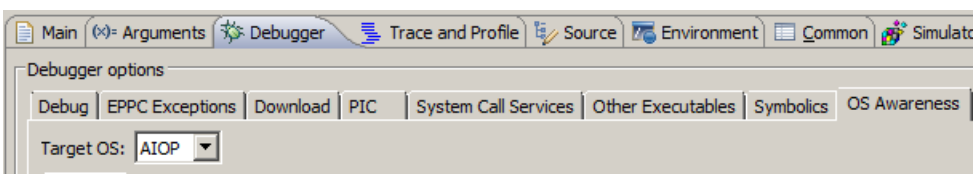


Figure 7. Selecting AIOP Target OS

9. Click **Debug** for attaching to the AIOP.

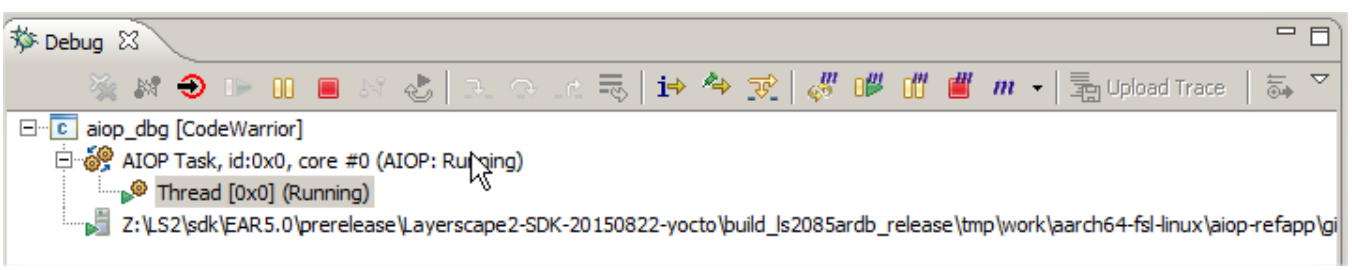


Figure 8. Debug view - Attaching AIOP

You can debug the AIOP APP using the following two methods:

- Debugging AIOP from system entry point
- Debugging AIOP from application entry point

6.1 Debugging AIOP from system entry point

1. To access the very first AIOP instruction (the entry point), you need to control the entire system booting process (U-Boot/GPP > MC > AIOP) and have run-control on the GPP core side.
2. Click Reset.

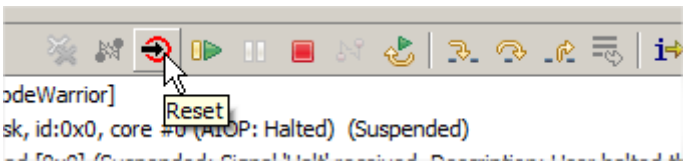


Figure 9. Debug view showing Reset button

The AIOP debugging halts.

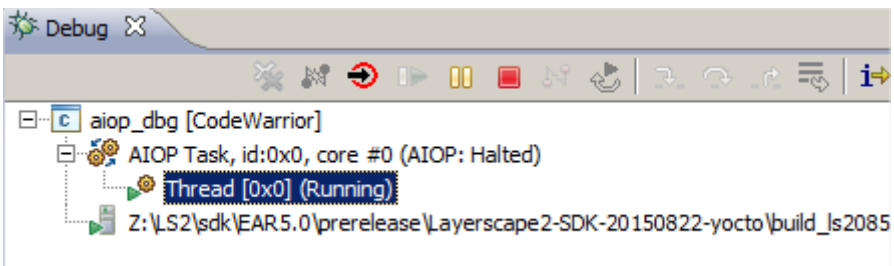


Figure 10. Debug view

3. Open the *CodeWarrior for APP* IDE.
4. Set a breakpoint at `__sys_start`.

NOTE

This is possible from both the source file and the **Debugger Shell** view. The breakpoint from the `__sys_start init` hits just after the AIOP tool loads the AIOP application.

Figure 11. CodeWarrior for APP - Editor view

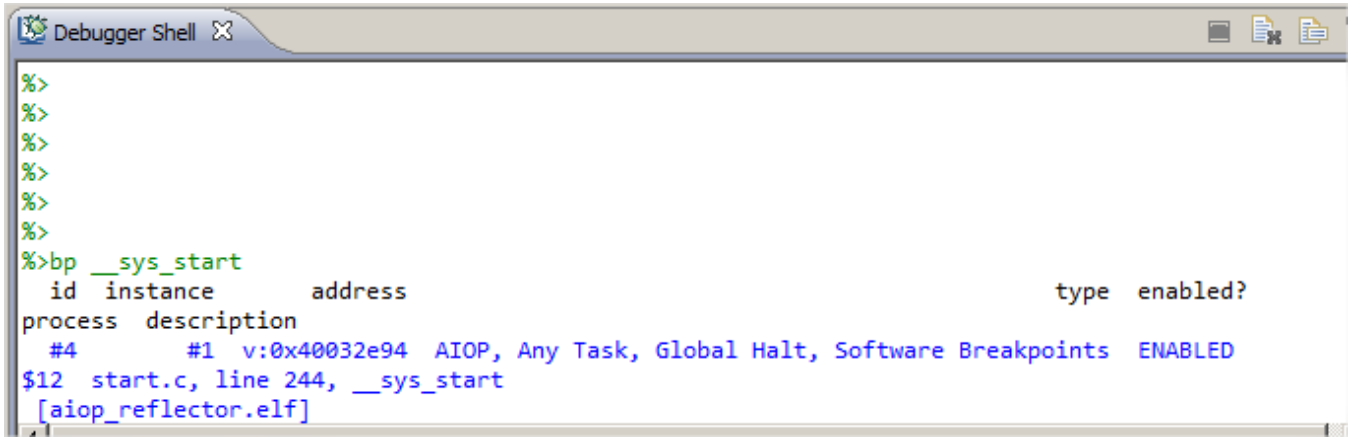


Figure 12. CodeWarrior for APP - Debugger Shell view

5. Click **Resume** to boot the entire eco-system (*u-boot/GPP* > *MC* > *Linux* > *AIOP*) using the Debugger Shell view. Write the following command in the Debugger Shell view `<protocol ccs::run_core 288>`

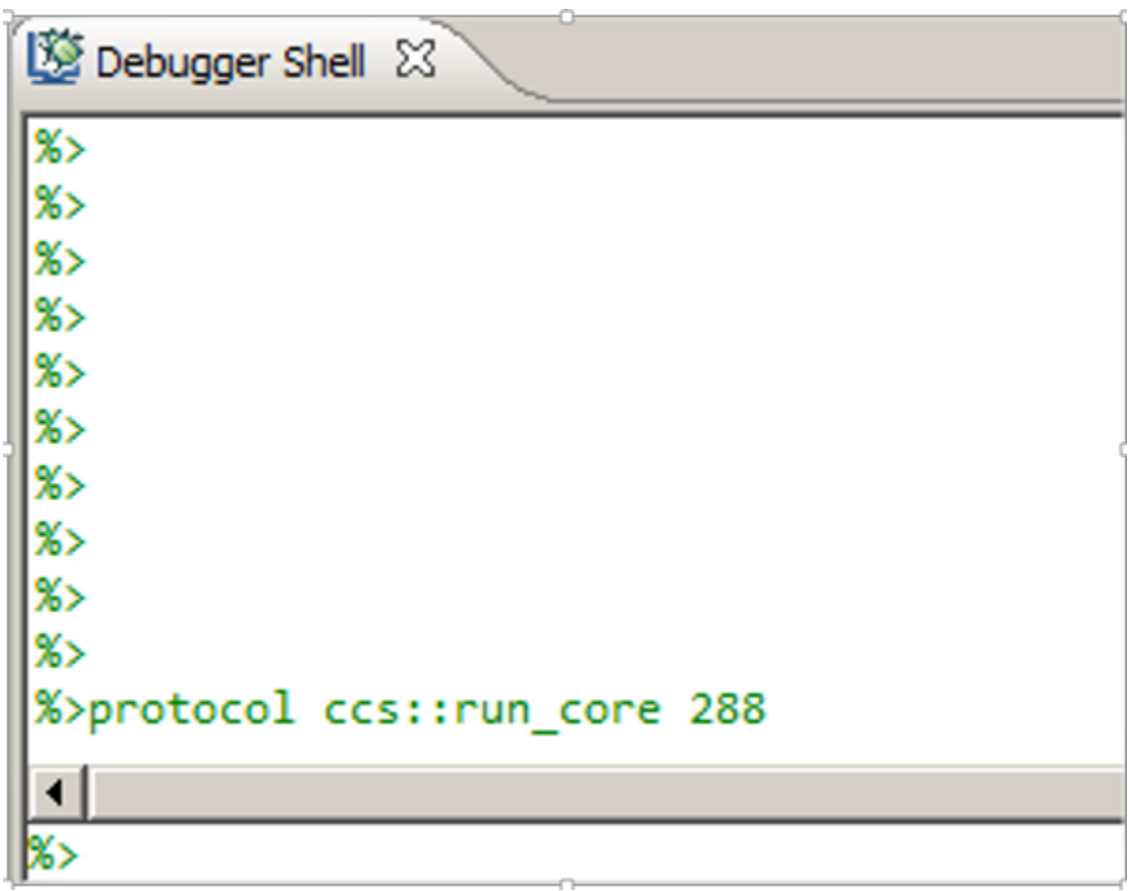


Figure 13. CodeWarrior for APP - Debug Shell view

6. The debugger hits the break point `__sys_start` after the `aiop_tool` loads the AIOP application from the linux target. For more details, see Hardware setup.

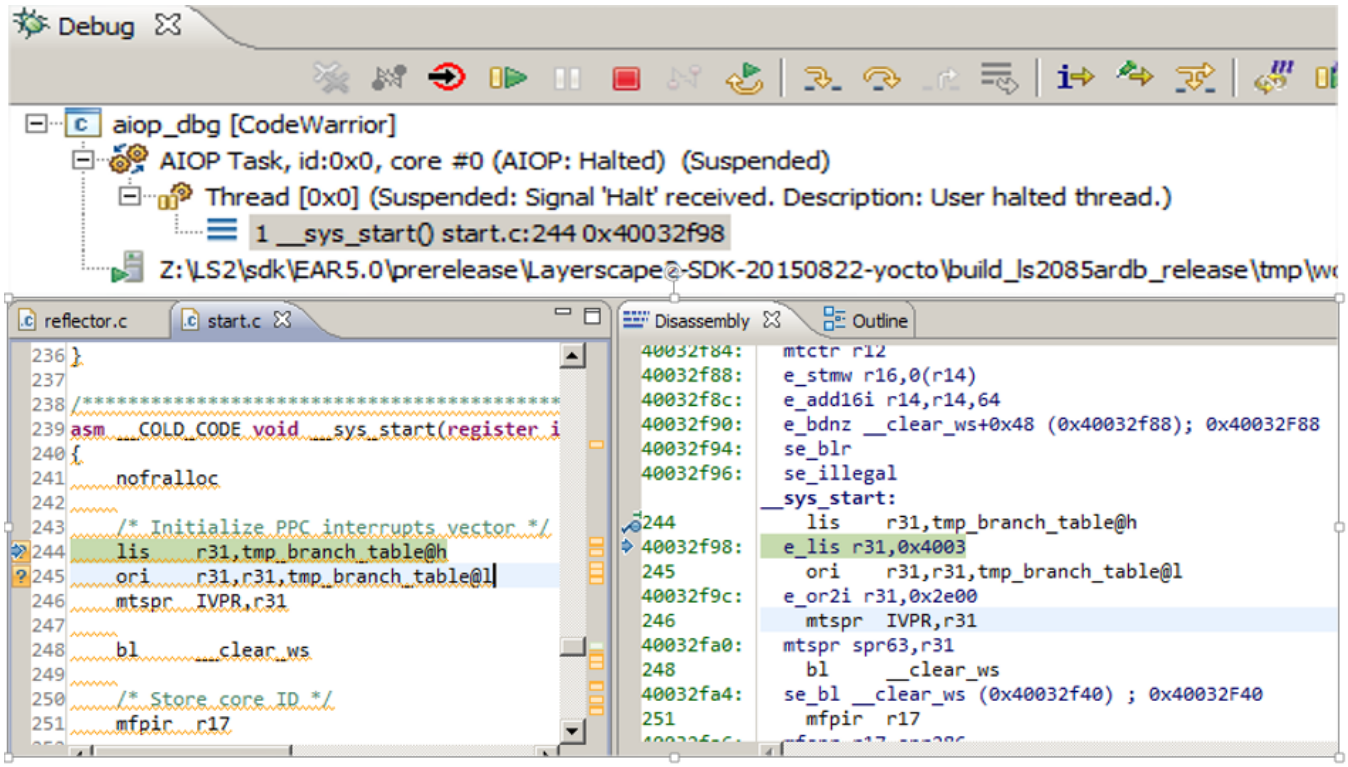


Figure 14. CodeWarrior for APP - Debug perspective

6.2 Debugging AIOP from application entry point

The entry point function executed by a triggered AIOP task is `app_reflector`. A breakpoint in this function hits when you generate a traffic using the ping command (see Hardware setup). To debug AIOP from the application entry point, follow the steps below:

1. Set up a breakpoint at `app_reflector` symbol using either the source file or the **Debugger Shell** view.

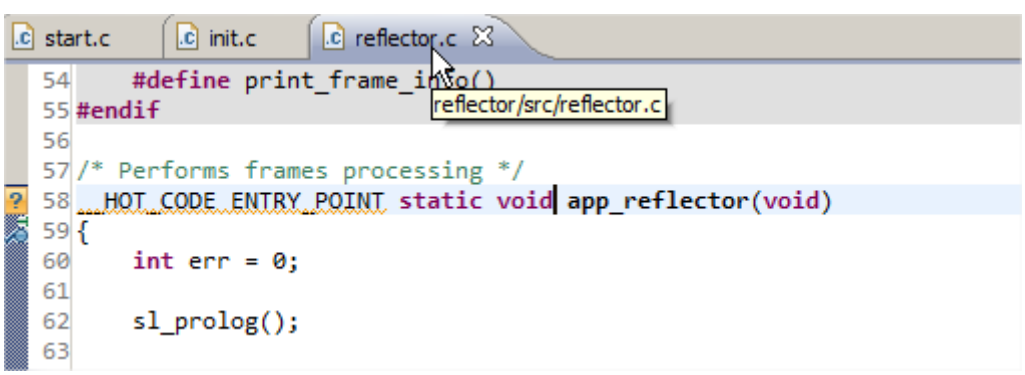


Figure 15. Setting breakpoint using source file

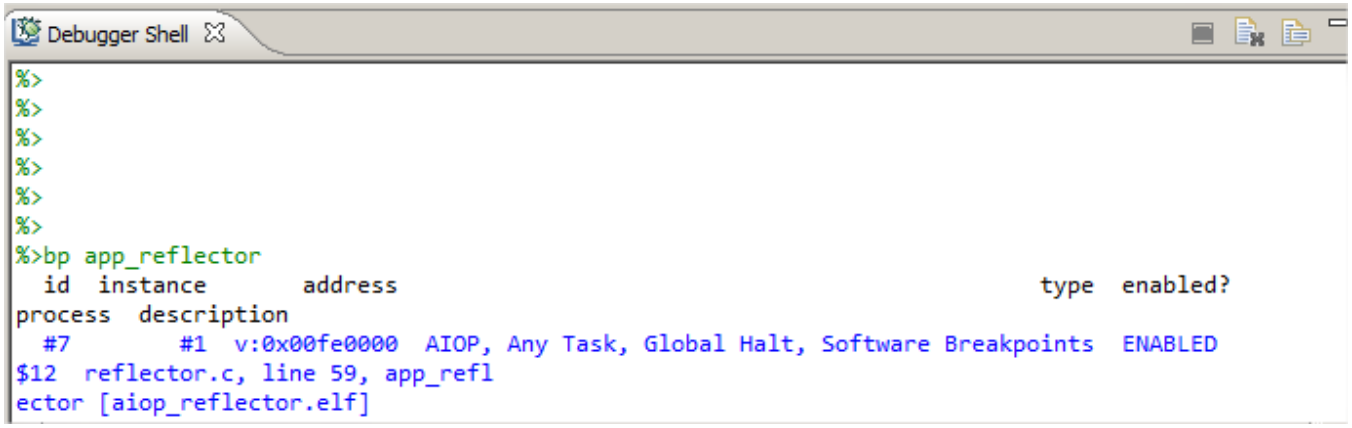


Figure 16. Setting breakpoint using Debugger Shell view

2. Click **Resume** from the **Debug** view.
   The figure below shows the AIOP task suspended in `core_ready_for_tasks()` function.

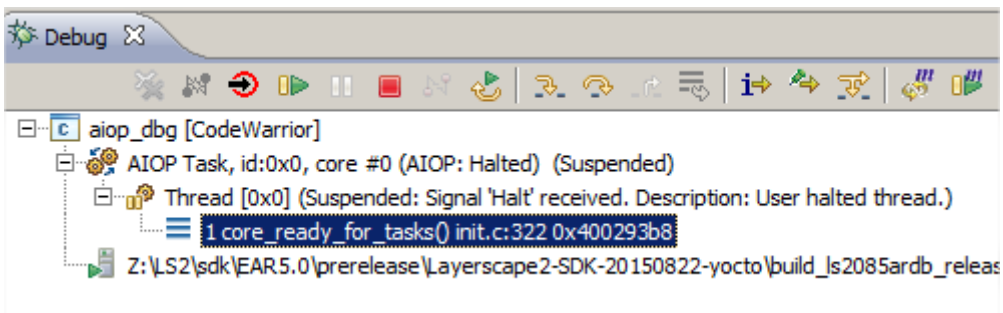


Figure 17. Debug view displaying core_ready_for_tasks() function

3. The core finishes to boot and waits for the tasks to be triggered.
4. Now, follow the AIOP reflector demonstration steps listed in the Hardware setup chapter.

NOTE

You need to load the kernel via the `tftp` and `bootm` commands. Sending the packets (with ping) to the AIOP interfaces generate tasks that can be observed/ debugged in the **System Browser** view and also hits the breakpoint from the `app_reflector` symbol. For full debugging capabilities of the System Browser and the AIOP Task Aware features, see the *AIOP Task Aware Debug* (document AN5044) application note.

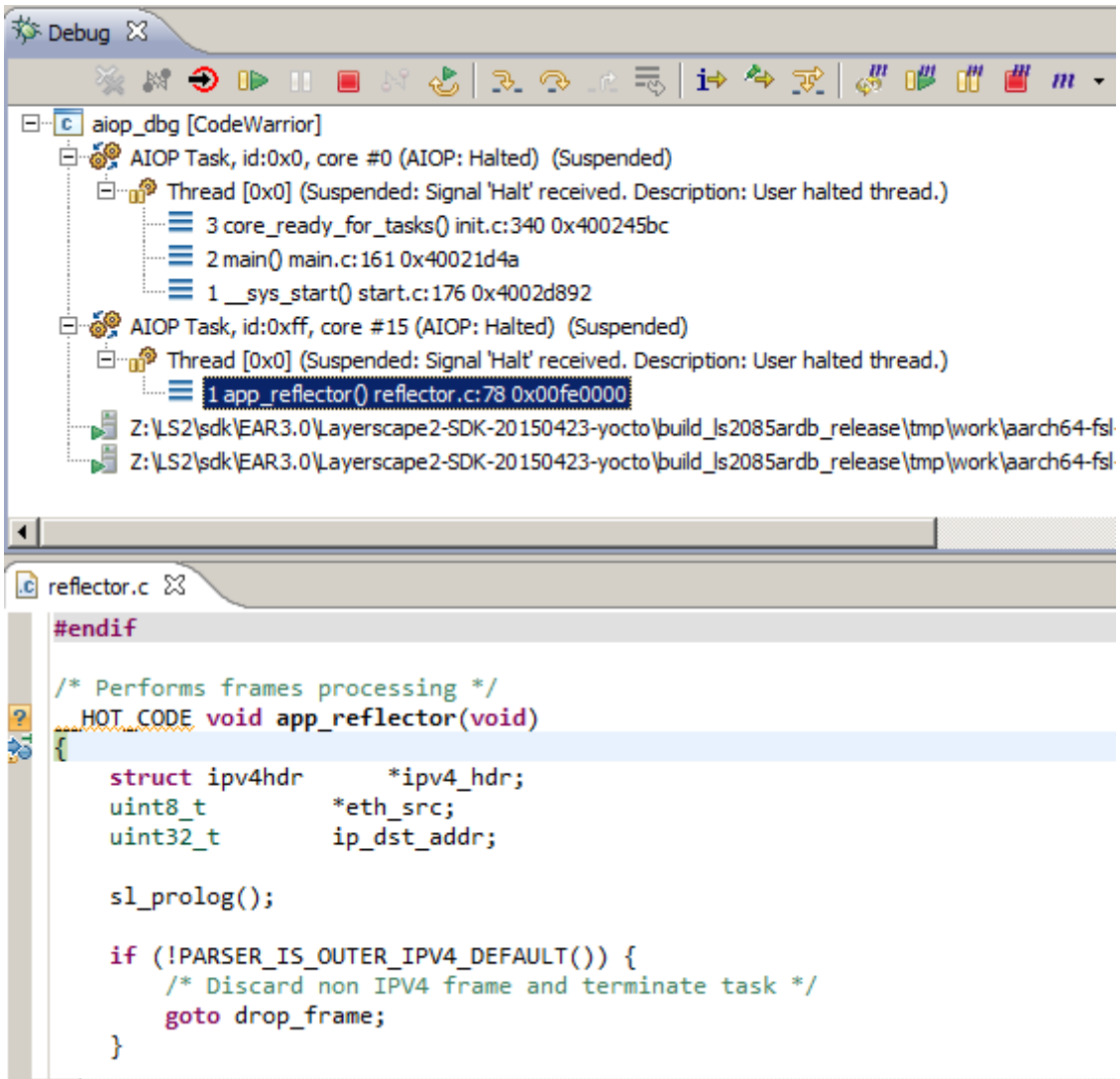
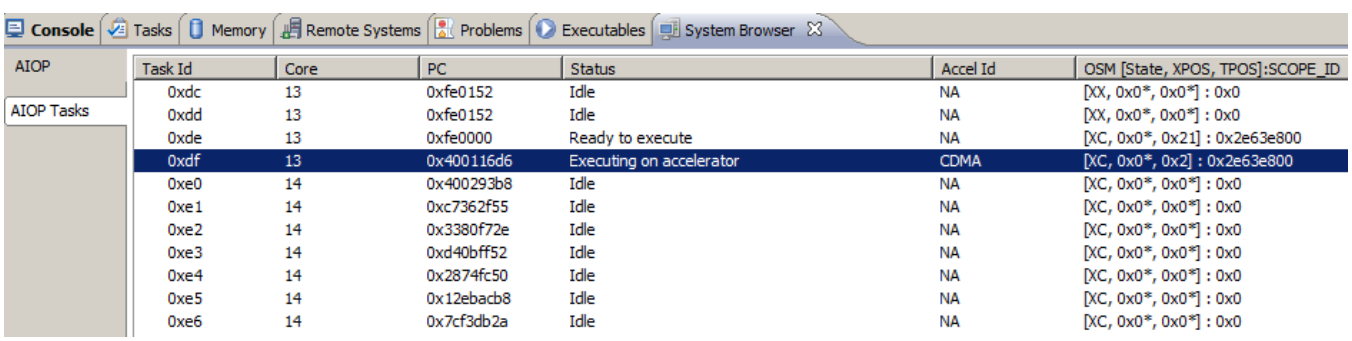


Figure 18. Debug view - app_reflector breakpoint



Task Id	Core	PC	Status	Accel Id	OSM [State, XPOS, TPOS]:SCOPE_ID
0xdc	13	0xfe0152	Idle	NA	[XX, 0x0*, 0x0*] : 0x0
0xdd	13	0xfe0152	Idle	NA	[XX, 0x0*, 0x0*] : 0x0
0xde	13	0xfe0000	Ready to execute	NA	[XC, 0x0*, 0x21] : 0x2e63e800
0xdf	13	0x400116d6	Executing on accelerator	CDMA	[XC, 0x0*, 0x2] : 0x2e63e800
0xe0	14	0x400293b8	Idle	NA	[XC, 0x0*, 0x0*] : 0x0
0xe1	14	0xc7362f55	Idle	NA	[XC, 0x0*, 0x0*] : 0x0
0xe2	14	0x3380f72e	Idle	NA	[XC, 0x0*, 0x0*] : 0x0
0xe3	14	0xd40bff52	Idle	NA	[XC, 0x0*, 0x0*] : 0x0
0xe4	14	0x2874fc50	Idle	NA	[XC, 0x0*, 0x0*] : 0x0
0xe5	14	0x12ebacb8	Idle	NA	[XC, 0x0*, 0x0*] : 0x0
0xe6	14	0x7cf3db2a	Idle	NA	[XC, 0x0*, 0x0*] : 0x0

Figure 19. System Browser view

7 Collecting hardware trace

To collect the hardware trace, follow the steps listed below:

1. Open **Run > Debug Configurations > Trace and Profile** tab.
2. Check the **Enable Trace and Profile** checkbox. For customizing the trace options, click **Edit**.

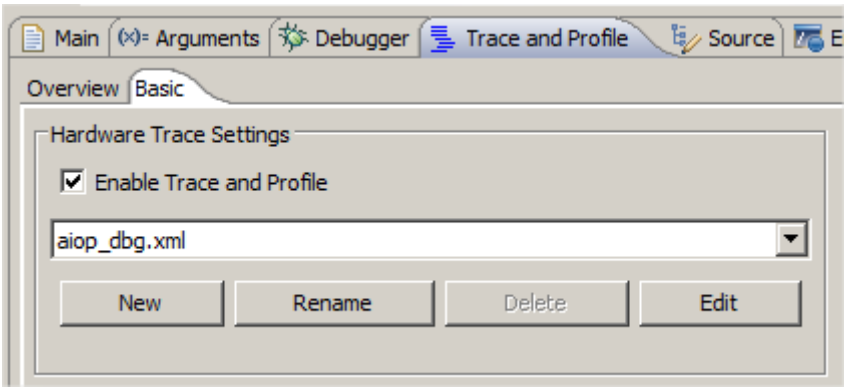


Figure 20. Trace and Profile tab

3. Click **Debug**.
   The trace gets collected between the two suspended events.

NOTE

After the *attach* is completed, it is mandatory for the task to process the suspend operation first.

4. Ensure that you set up the breakpoints in the `app_reflector` entry point.
5. Click **Resume**.
6. Send the ping traffic as suggested in the Hardware setup chapter.
7. The debugger hits the breakpoint.
8. Click **Resume** again for executing the entry point function and for generating the trace for your entry point function.
9. The debugger hits the breakpoint again.
10. Click **Upload Trace** to collect the trace.

Figure 21. Debug view - Collect Trace option

11. The collected trace appears in the **Analysis Results** view.

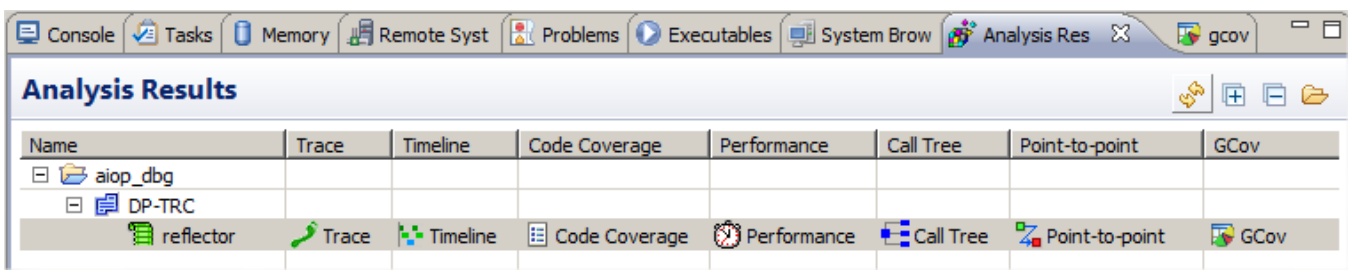


Figure 22. Analysis Results view

12. It is mandatory to open the **Trace** item first for letting the CodeWarrior IDE to decoding the gathered hardware trace.

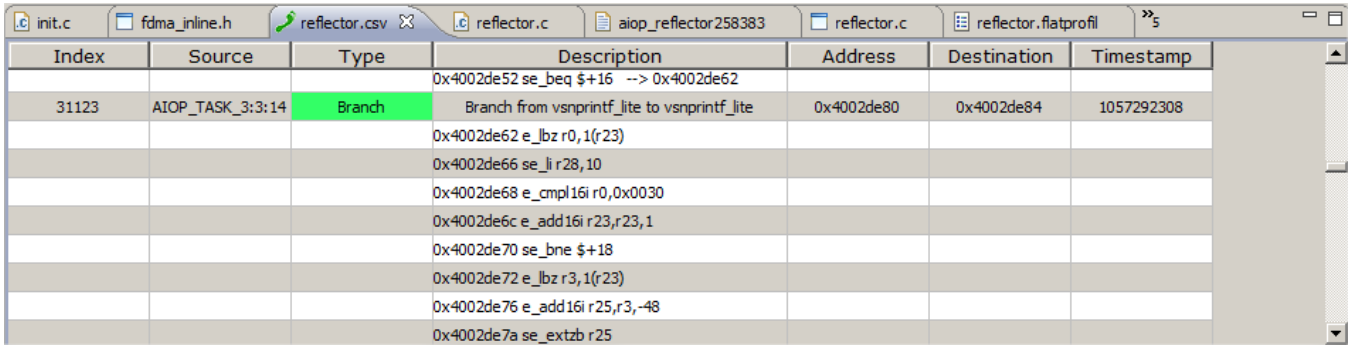


Figure 23. Hardware trace

For the rest of the items, ensure that you select the last task because the app reflector is enabling the tasks in a round-robin manner starting from the last task.

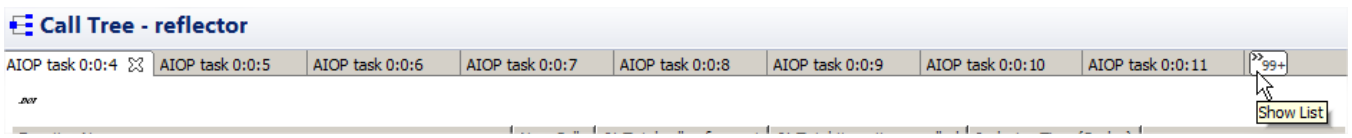


Figure 24. Call Tree view

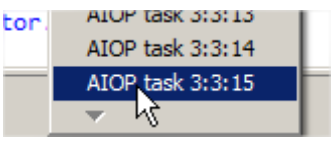


Figure 25. Selecting task

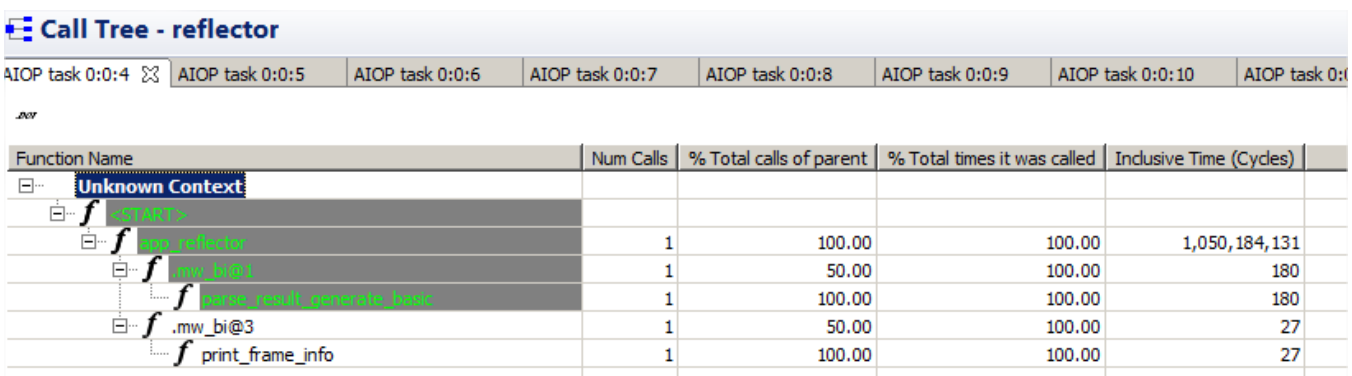


Figure 26. Collected trace

7.1 GCov code coverage

To enable GCov code coverage for reflector, follow the steps below:

1. Enable the **Generate Code Coverage File** option from the **Project > Properties > Settings > Tool Settings > Compiler > Processor** and re-build the project.

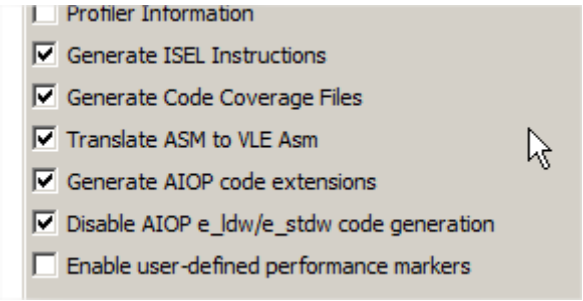


Figure 27. Generate Code Coverage File option

2. Follow the steps from Collecting hardware trace section to have the gcov results.

   For more details, see the section *6.3 GCov* of the *CodeWarrior Development Studio for Advanced Packet Processing Targeting Manual* (document CWAPPTM).

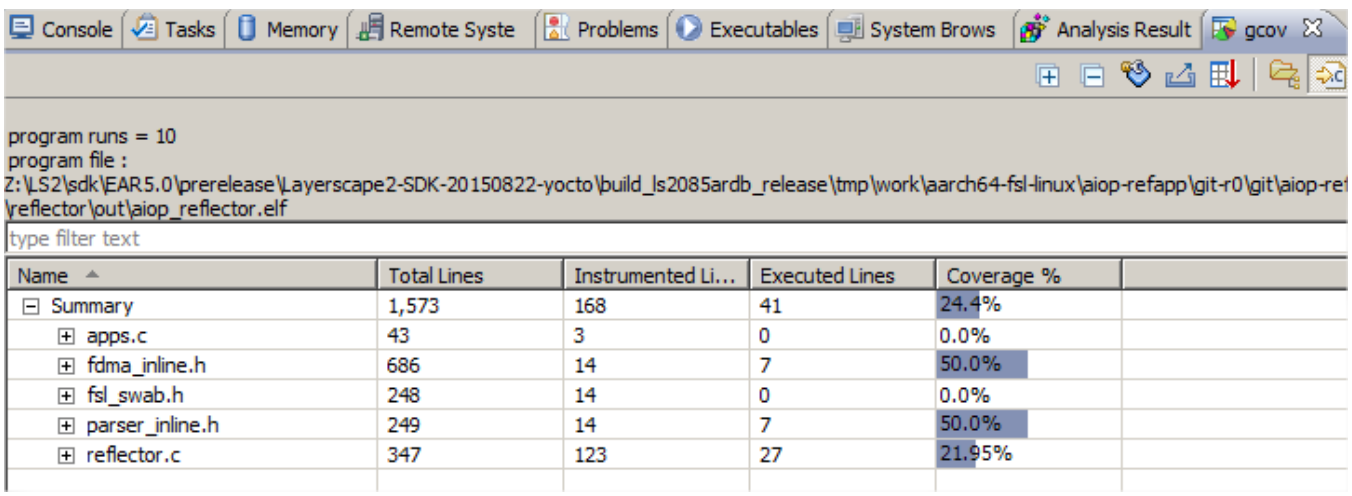


Figure 28. gcov view

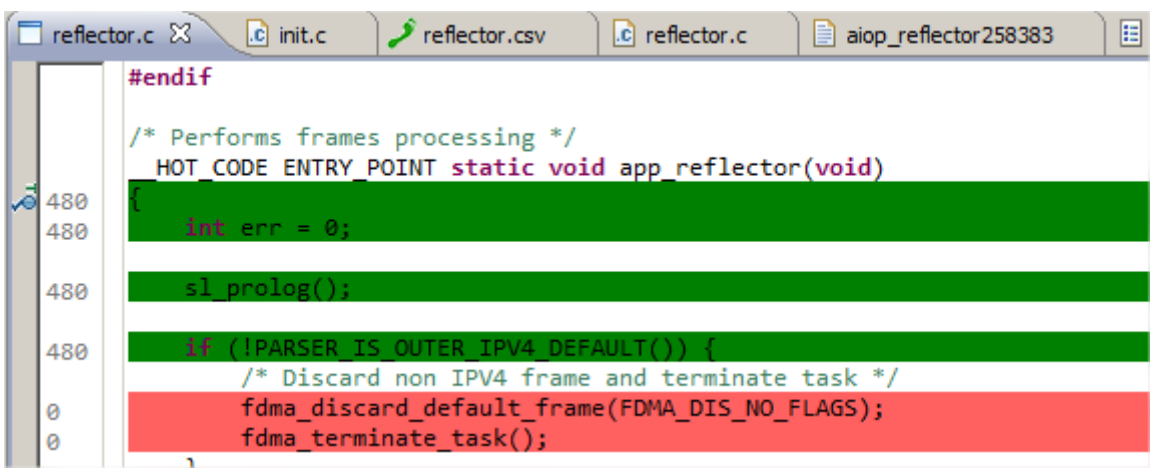


Figure 29. Editor view - reflector.c file

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number AN5165
Revision 10.3.2, 08/2018