AIOP SDK Applications Debug

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.

Contents

1	Over	view	1	
2	Prere	quisites	2	
3	Build	ing AIOP reflector APP	2	
4	Hardware setup			
5	Importing and building AIOP reflector project			
5	Debu Code	gging AIOP APP using Warrior	8	
	6.1	Debugging AIOP from system entry point	11	
	6.2	Debugging AIOP from application entry point	13	
7	Colle	cting hardware trace	15	
	7.1	GCov code coverage	17	



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):

Hardware setup



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

AIOP SDK Applications Debug, Rev. 10.3.2, 08/2018

Hardware setup

\$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 hugepagesz=26

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 0000000006

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 etho (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
                             pluqged-state
dpni.1
                             plugged
                             plugged
dpbp.3
dpbp.2
                             plugged
                             plugged
dpbp.1
----
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
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
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
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
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: 00-00-00-00-00-00
IP SRC: 6.6.6.1 IP DST: 6.6.6.10

Importing and building AIOP reflector project

```
👷 Problems 🧔 Tasks 📮 Console 🔲 Properties 🅻 Debug Print 🔀 🖏 Progress
1863. <DBG> 6087.586860 busybox.nosuid(1504):
                                               IP SRC: 198.20.1.1 IP DST: 198.19.1.150
1864. <DBG> 6087.586861 busybox.nosuid(1504):
1865. <DBG> 6087.586862 busybox.nosuid(1504): RX on DPNI:7 | CORE:15
1866. <DBG> 6087.586863 busybox.nosuid(1504): MAC SA: 100040c-07-07-07-07-07
1867. <DBG> 6087.586864 busybox.nosuid(1504): IP SRC: 198.20.1.1 IP DST: 198.19.1.151
1868. <DBG> 6087.586864 busybox.nosuid(1504):
1869. <DBG> 6087.586865 busybox.nosuid(1504): RX on DPNI:7 | CORE:14
1870. <DBG> 6087.586866 busybox.nosuid(1504): MAC SA: 100040c-07-07-07-07-07
1871. <DBG> 6087.586867 busybox.nosuid(1504): IP SRC: 198.20.1.1 IP DST: 198.19.1.152
1872. <DBG> 6087.586868 busybox.nosuid(1504):
1873. <DBG> 6087.586869 busybox.nosuid(1504): RX on DPNI:7 | CORE:15
1874. <DBG> 6087.586870 busybox.nosuid(1504): MAC SA: 100040c-07-07-07-07-07
1875. <DBG> 6087.586870 busybox.nosuid(1504): IP SRC: 198.20.1.1 IP DST: 198.19.1.153
1876. <DBG> 6087.586871 busybox.nosuid(1504):
1877. <DBG> 6087.586872 busybox.nosuid(1504): RX on DPNI:7 | CORE:14
1878. <DBG> 6087.586873 busybox.nosuid(1504): MAC SA: 100040c-07-07-07-07-07
1879. <DBG> 6087.586874 busybox.nosuid(1504): IP SRC: 198.20.1.1 IP DST: 198.19.1.154
1880. <DBG> 6087.586875 busybox.nosuid(1504):
1881. <DBG> 6087.586875 busybox.nosuid(1504): RX on DPNI:7 | CORE:15
1882. <DBG> 6087.586876 busybox.nosuid(1504): MAC SA: 100040c-07-07-07-07-07
1883. <DBG> 6087.586877 busybox.nosuid(1504): IP SRC: 198.20.1.1 IP DST: 198.19.1.155
1884. <DBG> 6087.586878 busybox.nosuid(1504):
1885. <DBG> 6087.586879 busybox.nosuid(1504): RX on DPNI:7 | CORE:14
1886. <DBG> 6087.586880 busybox.nosuid(1504): MAC SA: 100040c-07-07-07-07-07
1887. <DBG> 6087.586881 busybox.nosuid(1504):
                                               IP SRC: 198.20.1.1 IP DST: 198.19.1.156
1888. <DBG> 6087.586881 busybox.nosuid(1504):
1889. <DBG> 6087.586882 busybox.nosuid(1504): RX on DPNI:7 | CORE:15
1890. <DBG> 6087.586883 busybox.nosuid(1504): MAC SA: 100040c-07-07-07-07-07
1891. <DBG> 6087.586884 busybox.nosuid(1504): IP SRC: 198.20.1.1 IP DST: 198.19.1.157
1892. <DBG> 6087.586885 busybox.nosuid(1504):
1893. <DBG> 6087.586886 busybox.nosuid(1504): RX on DPNI:7 | CORE:15
1894. <DBG> 6087.586887 busybox.nosuid(1504): MAC SA: 100040c-07-07-07-07-07
1895. <DBG> 6087.586888 busybox.nosuid(1504): IP SRC: 198.20.1.1 IP DST: 198.19.1.158
info: Collection delayed.
1897. <DBG> 6088.586986 busybox.nosuid(1504):
1898. <DBG> 6088.586988 busybox.nosuid(1504): RX on DPNI:7 | CORE:14
1899. <DBG> 6088.586989 busybox.nosuid(1504): MAC SA: 100040c-07-07-07-07-07
1900. <DBG> 6088.586990 busybox.nosuid(1504): IP SRC: 198.20.1.1 IP DST: 198.19.1.159
🎩 Terminals 🐹 📲 Remote Systems
             🐙 reflector 1
reflector
                              X
p_console
root@ls2085ardb:/run/test#
root@ls2085ardb:/run/test# LD_PRELOAD=/run/test/libls.linux.debugprint.libd.so.0.0 tail -f /dev/fs1_
```

```
aiop_console
```

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

AIOP SDK Applications Debug, Rev. 10.3.2, 08/2018

Importing and building AIOP reflector project

🥬 Import							
Import Projects Select a directory to search	h for existing Edipse projects.						
A							
 Select roo<u>t</u> directory: 	<-20150822-yocto\build_ls2085ardb_release\tmp\work\aar	ch64-fsl-linux\aiop-refapp	Browse				
○ Select <u>a</u> rchive file:			Browse				
Projects:							
aiop_core_lib_veri	f (Z:\LS2\sdk\EAR5.0\prerelease\Layerscape2-SDK-2015082 k\EAR5_0\prerelease\Layerscape2-SDK-20150822-vocto\bui	2-yocto\build_ls2085ardb d_ls2085ardb_release\tmr	Select All				
aiopsl_stack_estim	ation (Z: \LS2\sdk\EAR5.0\prerelease\Layerscape2-SDK-201	50822-yocto\build_ls2085a	Deselect All				
app_process_pack	et (Z:\LS2\sdk\EAR5.0\prerelease\Layerscape2-SDK-20150 2\sdk\EAR5.0\prerelease\Layerscape2-SDK-20150822-vocto	322-yocto\build_ls2085ardl \build_ls2085ardb_release	R <u>e</u> fresh				
classifier (Z:\LS2\s	dk\EAR5.0\prerelease\Layerscape2-SDK-20150822-yocto\b	uild_ls2085ardb_release\tn					
cmdif_test (Z:\LS2	\sdk\EAR5.0\prerelease\Layerscape2-SDK-20150822-yocto S2\sdk\EAR5.0\prerelease\Layerscape2-SDK-20150822-yoct	build_ls2085ardb_release\ o\build_ls2085ardb_releas					
ipf_demo (Z:\LS2\	sdk\EAR5.0\prerelease\Layerscape2-SDK-20150822-yocto\b	uild_ls2085ardb_release\t					
ipr_demo (Z:\LS2\	sdk\EAR5.0\prerelease\Layerscape2-SDK-20150822-yocto\b :\LS2\sdk\EAR5.0\prerelease\Layerscape2-SDK-20150822-y	uild_ls2085ardb_release\ti octo\build_ls2085ardb_rele					
reflector (Z:\LS2\s	dk\EAR5.0\prerelease\Layerscape2-SDK-20150822-yocto\b	uild_ls2085ardb_release\tn					
roc_process_pack	et (Z:\LS2\sdk\EAR5.0\prerelease\Layerscape2-SDK-201508 \\ S2\sdk\EAR5.0\prerelease\Layerscape2-SDK-20150822-vi	22-yocto\build_ls2085ardb octo\build_ls2085ardb_rele					
	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,						
<u></u>							
Copy projects into wo	rkspace						
Working sets							
Add project to worki	ng sets						
Wgrking sets: Select							
?	< <u>B</u> ack	ext > Einish	Cancel				

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.

🥦 Properties for reflector						
type filter text	Settings - reflector					
 Resource Builders C/C++ Build Build Variables Discovery Options Environment Logging Settings Tool Chain Editor C/C++ General Linux Tools Path Run/Debug Settings 	Configuration: aiop_dbg [Active] Tool Settings Build Steps Build Artifact Binary Parsers Error Parsers Build Tool Versions CPU Debugging Optimization Level O Bessages Speed vs. Size Speed Command line: -opt level=0 Inlining Smart Bottom-up Inlining Bottom-up Inlining Preprocessor Preprocessor Warnings Warnings Warnings Processor					

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**.

Properties for AIOP Demos_C	onnection	_ 🗆 🗙
Hardware or Simulator Connect	Hardware or Simulator Connection	$\leftarrow \star \Rightarrow \star \bullet$
	Parent profile: B32331-11	
	Name: AIOP Demos_Connection	
	Description: AIOP Demos Connection	
	Template: None	Apply Defaults
	Target:	<u>N</u> ew
	Connection type: CodeWarrior TAP	_
	Connection Advanced	
	CodeWarrior TAP	
	Hardware connection: Ethernet	
	Hostname/IP: fsl036eef	
	Serial number:	
	_ JTAG settings	
	JTAG clock speed (kHz): 10000	•
	CCS server	
	Automatic launch	
	Server port number: 41475	
	CS executable:	
	C Manual launch	
	Server hostname/IP: hxbsc132:5720	
	Server port number: 42324	
	Connect server to TAP	
(?)	OK	Cancel

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.

📄 Main 😥 Arguments 🏇 Debugger 📃 Trace and Profile 🤯 Source 🚾 Environment 🔲 Common 💣 Simulato									
	Debugger options								
	Debug	EPPC Exceptions	Download I	PIC S	System Call Servi	ces Oth	er Executables	Symbolics	OS Awareness
	Target (DS: AIOP 💌							

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.

.0	sta	rt.c 23
	239	a COLD CODE void sys start(register int argc, register char **argv, register char **envp)
	240	∫aiop_sl/cw_files/start.c
	241	nofralloc
	242	
	243	/* Initialize_PPC_interrupts_vector_*/
?	244	lisR31.tmp_branch_table@h
?	245	ori <u>r31.r31.tmp_branch_table@l</u>
	246	mtsprIVPR.r31
	247	
	248	blclear_ws

Figure 11. CodeWarrior for APP - Editor view

🔯 Debugger Shell 🛛	🔳 🖹 🗎
%>	
%>	
%>	
%>	
%>	
%>bpsys_start	
1d instance address type	enabled?
#4 #1 view40022004 ATOP Any Tack Global Halt Software Breakneints	
\$12 start c line 244 sys start	LNADEED
[aion reflector.elf]	

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 oprotocol 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.

🏂 Debug 🖾						
🍇 💐 🔿 🕩 🗉	🔳 24 🌏	, 크, - : : : : : : : : : : : : : : : : : :				
□··· C aiop_dbg [CodeWarrior]						
AIOP Task, id:0x0, core #0 (AIOP: Ha	alted) (Suspe	ended)				
Thread [0x0] (Suspended: Signal	'Halt' receiver	d Description: User balted thread)				
	40022602	a. Description: Oser harted an edd.y				
I	x40032198					
Z:\LS2\sdk\EAR5.0\prerelease\Layers	cape@-SDK-2	20150822-yocto\build_ls2085ardb_release\tmp\wo				
c reflector.c 💽 start.c 🕱 🗖 🗖	Disassembly	🛛 🗧 Outline				
236 }	40032184:	mtctr r12				
237	40032f88:	e_stmw r16,0(r14)				
238 /************************************	40032f8c:	e_add16i r14,r14,64				
239 asm COLD CODE void sys start(register i	40032f90:	e_bdnzclear_ws+0x48 (0x40032f88); 0x40032F88				
240 🧜	40032f94:	se_blr				
241nofralloc	40032796:	se_illegal				
242	2244					
243 /* Initialize PPC interrupts vector */	▲ 40032f98:	e lis r31.0x4003				
244 IIS roi, cmp branch table@l	245	ori r31.r31.tmp branch table@l				
246 mtspr TVPR r31	40032f9c:	e or2i r31,0x2e00				
247	246	mtspr IVPR,r31				
248 bl clear ws	40032fa0:	mtspr spr63,r31				
249	248	blclear_ws				
250 <u>/* Store core ID */</u>	40032fa4:	se_blclear_ws (0x40032f40) ; 0x40032F40				
251mfpirr17	251	mtpir r17				

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

🔯 Debugger Shell 🕴	
%>	
%>	
%>	
%>	
%>	
%>	
%>bp app_reflector	
id instance address type	enabled?
process description	
<pre>#7 #1 v:0x00fe0000 AIOP, Any Task, Global Halt, Software Breakpoints</pre>	ENABLED
<pre>\$12 reflector.c, line 59, app_refl</pre>	
ector [aiop_reflector.elf]	

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

星 Console 🖉	Tasks 🚺 Memory	📲 Remote Systems	s 🔝 Problems 🚺	Executables 🗐 System Browser 🛛		
AIOP	Task Id	Core	PC	Status	Accel Id	OSM [State, XPOS, TPOS]:SCOPE_ID
	0xdc	13	0xfe0152	Idle	NA	[XX, 0x0*, 0x0*] : 0x0
AIOP Tasks	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:

AIOP SDK Applications Debug, Rev. 10.3.2, 08/2018

Collecting hardware trace

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

📄 Main 📯 Argume	nts (🕸 Debugge	er 📑 Tr	ace and Pro	file 🤤	Source) 🚾 E
Overview Basic						
Hardware Trace Se	ttings					
Enable Trace	and Profile					
aiop_dbg.xml						.
New	Rename		Delete		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.

📮 Console 🧟 Tasks 🚺 M	emory 🗐 R	temote Syst 🛛	🚼 Problems 🚺 Exec	cutables 🗐 System	m Brow 💣 An	alysis Res 🛛 🔀	gcov 🗆 🗖
Analysis Results						Ŷ	
Name	Trace	Timeline	Code Coverage	Performance	Call Tree	Point-to-point	GCov
🗆 🗁 aiop_dbg							
🗆 🛃 DP-TRC							
🗐 reflector	🥒 Trace	Timeline	E Code Coverage	🛞 Performance	Call Tree	🌄 Point-to-point	🐼 GCov

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.

🝺 init.c 🛛 🗖 fi	dma_inline.h 💋	reflector.csv 🛛	. reflector.c	aiop_reflector258383	reflector.c	🗄 reflector.flatp	orofil ^{>>} 5	- 8
Index	Source	Туре	De	Description		Destination	Timestamp	
			0x4002de52 se_beq	\$+16> 0x4002de62				
31123	AIOP_TASK_3:3:14	Branch	Branch from vsn	printf_lite to vsnprintf_lite	0x4002de80	0x4002de84	1057292308	
			0x4002de62 e_lbz r0	, 1(r23)				
			0x4002de66 se_li r28	x4002de66 se_li r28,10				-
			0x4002de68 e_cmpl1	x4002de68 e_cmpl16i r0,0x0030				
			0x4002de6c e_add16	ii r23,r23,1				
			0x4002de70 se_bne	\$+18				
			0x4002de72 e_lbz r3	, 1(r23)				
			0x4002de76 e_add16	5i r25,r3,-48				
			0x4002de7a se_extz	b r 25				•

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.

E Call Tree -	reflector							
AIOP task 0:0:4 🔀	AIOP task 0:0:5	AIOP task 0:0:6	AIOP task 0:0:7	AIOP task 0:0:8	AIOP task 0:0:9	AIOP task 0:0:10	AIOP task 0:0:11	»» 99+
.007								Show List
			_					

Figure 24. Call Tree view

tor	ATOP TASK 3:3:13	
	AIOP task 3:3:14	
	AIOP_task 3:3:15	F
	- V5	

Figure 25. Selecting task

Call free -	reflector									
AIOP task 0:0:4 🔀	AIOP task 0:0:5	AIOP task 0:0:6	AIOP t	task 0:0:7	AIOP task 0:0:8	AIOP task 0:0:9	AIOP	task 0:0:10	AIOP ta	sk 0:(
<i>D07</i>										
Function Name				Num Calls	% Total calls of parent	% Total times it was o	alled	Inclusive Time (Cycles)	
⊡ Unknown	Context									
Ė… ∫ ≺star										
<i>f</i> au	reflector			1	100.00	1	00.00	1,050,	184, 131	
⊢ <i>f</i>	.mw_bi@1			1	50.00	1	00.00		180	
	f parse_result_gen	erate_basic		1	100.00	1	00.00		180	
⊡ _ <i>f</i>	.mw_bi@3			1	50.00	1	00.00		27	
	f print_frame_info			1	100.00	1	00.00		27	

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.

Profiler Information	
Generate ISEL Instructions	
Generate Code Coverage Files	
✓ Translate ASM to VLE Asm	
Generate AIOP code extensions	.0
Disable AIOP e_ldw/e_stdw code generation	
Enable user-defined performance markers	

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).



program runs = 10

ż:\LS2\sdk\EAR5.0\prerelease\Layerscape2-SDK-20150822-yocto\build_ls2085ardb_release\tmp\work\aarch64-fsl-linux\aiop-refapp\git+r0\git\aiop-ref \reflector\out\aiop_reflector.elf |type_filter_text

a libere su percar								
Name 🔺	Total Lines	Instrumented Li	Executed Lines	Coverage %				
 Summary 	1,573	168	41	24.4%				
apps.c	43	3	0	0.0%				
fdma_inline.h	686	14	7	50.0%				
	248	14	0	0.0%				
parser_inline.h	249	14	7	50.0%				
reflector.c	347	123	27	21.95%				

Figure 28. gcov view



Figure 29. Editor view - reflector.c file

program file :

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support Information in this document is provided solely to enable system and software implementers to use NXP 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. NXP reserves the right to make changes without further notice to any products herein.

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

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, Freescale, the Freescale logo, and QorlQ are trademarks of are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

© 2017-18 NXP B.V.

Document Number AN5165 Revision 10.3.2, 08/2018



