ODP Reflector Application Debug

1 2

3

4

1 Introduction

The most popular user space application for SDK/ODP (OpenDataPlane) is the packet reflector reference application.

This application note is focused on the odp_reflector application. It's the extension of odp_pktio application with the addition of the schedule PUSH mode, where scheduled packets are received in the PUSH mode. Like odp_pktio, the odp_reflector application swaps the Ethernet and IP header addresses of the received frames and transmits them on the same interface.

This application note explains a use case, where odp_reflector is running on a single board using 2 ports connected back-to-back. If you need another hardware setup or full details about the ODP applications, see *LS2085 SDK Quick Start Guide*.

This application note explains:

- How you can build a real hardware setup for running odp_reflector
- How you can import, download, run, and debug the odp_reflector application from CodeWarrior
- How you can attach to a running odp_reflector application and debug it using CodeWarrior

Contents

Introd	luction	1
Get th	e ODP reflector source files	2
Hardy	ware setup	2
3.1	Hardware setup using only one board	2
Code	Warrior Setup	6
4.1	Import and start the odp_reflector application from CodeWarrior	6
4.2	Attach to a running odp_reflector application and debug using CodeWarrior	11
	4.2.1 odp_reflector debug from entry_point	14
4.3	Debug capabilities	15



2 Get the ODP reflector source files

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

3 Hardware setup

The reflector application reflects back the packet received on the same interface where the packets are originally received, and the source and destination MAC and IP addresses of the received packet are swapped.

3.1 Hardware setup using only one board

In order to demonstrate the traffic "reflected", you can use a single board with 2 ports connected back-to-back. In the following figure, the copper red ports 5 and 6 are connected.

- Port 5 Linux container
- Port 6 ODP container

Hardware setup



Figure 1. Single board with 2 ports connected back-to-back

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

Run these commands when the U-Boot prompt is available.

At the U-Boot console, bring up the board via TFTP. Or, you can write the images to the flash memory using the CodeWarrior for ARMv8 flash programmer tool.

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

Hardware setup

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

<restart the board for loading new images>

```
Prepare target for running ODP 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 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. 00000000006 is the MAC of the dpmac.6 (this information can be verified using restool).

```
root@ls2085ardb:~# ifconfig ni0 6.6.6.1 up
root@ls2085ardb:~# arp -s 6.6.6.10 00000000006
```

Configure the eth0 interface.

root@ls2085ardb:~# ifconfig eth0 192.168.1.2

Connect a new DPNI for DPMAC.6 and create the rest of the objects for the ODP container using the dynamic_dpl.sh script. This script uses a Linux user space tool called **restool** that can be used to create/destroy containers and objects dynamically.

```
root@ls2085ardb:~# cd /usr/odp/scripts
root@ls2085ardb:/usr/odp/scripts# ./dynamic dpl.sh dpmac.6
Available DPRCs
dprc.1
dprc.2 Created
Validating the arguments.....
DPNI parameters :-->
        MAX SENDERS = 8
        MAX TCS = 1
        MAX DIST PER TC = 8
        MAX DIST KEY SIZE = 32
        DPNI OPTIONS =
DPNI OPT MULTICAST FILTER, DPNI OPT UNICAST FILTER, DPNI OPT DIST HASH, DPNI OPT DIST FS, DPNI OP
T FS MASK SUPPORT
DPCON parameters :-->
        DPCON PRIORITIES = 8
DPSECI parameters :-->
        DPSECI_QUEUES = 8
        DPSECI PRIORITIES = 2,2,2,2,2,2,2,2
DPIO parameters :-->
        DPIO PRIORITIES = 8
####### Parsing argument number 1 (dpmac.6) #######
        dpni.1 created with MAC addr = 00:00:00:00:0:6
```

Hardware setup

Verify if the DPNI interfaces are correctly allocated using restool.

root@ls2085ardb:/usr/odp/scripts# restool dpni info dpni.0 dpni version: 6.0 dpni id: 0 plugged state: plugged endpoint state: 1 endpoint: dpmac.5, link is up link status: 1 - up mac address: 4a:3e:27:d2:df:f6 dpni attr.options value is: 0x180 DPNI OPT UNICAST FILTER DPNI_OPT_MULTICAST_FILTER max senders: 1 max traffic classes: 1 max distribution's size per RX traffic class: class 0's size: 1 max unicast filters: 16 max multicast filters: 64 max vlan filters: 0 max QoS entries: 0 max QoS key size: 0 max distribution key size: 0 root@ls2085ardb:/usr/odp/scripts# restool dpni info dpni.1 dpni version: 6.0 dpni id: 1 plugged state: plugged endpoint state: 0 endpoint: dpmac.6, link is down link status: 0 - down mac address: 00:00:00:00:00:06 dpni_attr.options value is: 0x401b0 DPNI OPT DIST HASH DPNI_OPT_DIST_FS DPNI_OPT_UNICAST_FILTER DPNI_OPT_MULTICAST_FILTER DPNI_OPT_FS_MASK_SUPPORT max senders: 8 max traffic classes: 1 max distribution's size per RX traffic class: class 0's size: 8 max unicast filters: 16 max multicast filters: 64 max vlan filters: 0 max QoS entries: 0 max QoS key size: 0 max distribution key size: 32

Start the odp_reflector and check if this works using ping.

```
root@ls2085ardb:~# cd /usr/odp/bin/
root@ls2085ardb:/usr/odp/bin# export DPRC=dprc.2
root@ls2085ardb:/usr/odp/bin# ./odp reflector -i dpni.1 -m 0 -c 8 &
```

```
<enter>
root@ls2085ardb:~# ping 6.6.6.10 -c 2
PING 6.6.6.10 (6.6.6.10) 56(84) bytes of data.
64 bytes from 6.6.6.10: icmp_seq=1 ttl=64 time=0.093 ms
64 bytes from 6.6.6.10: icmp_seq=2 ttl=64 time=0.055 ms
--- 6.6.6.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.055/0.074/0.093/0.019 ms
```

4 CodeWarrior Setup

This topic explains:

- Import and start the odp_reflector application from CodeWarrior
- Attach to a running odp_reflector application and debug using CodeWarrior
- Debug capabilities

4.1 Import and start the odp_reflector application from CodeWarrior

After compiling the odp_reflector application, with debug information (-ggdb) you'll need to use in CodeWarrior the elf file containing the debug symbols. This will generate the correct elf/DWARF symbolic needed for the CodeWarrior parser to make data to symbols and symbols to data (this is very important).

- 1. Select File > Import > C/C++ > CodeWarrior Debug Projects.
- 2. Click Next.
- 3. Select the odp_reflector elf available at:

```
<Yocto>/build_ls2085ardb_release/tmp/work/aarch64-fsl-linux/odp/1.4-r0/git/example/
reflector/odp_reflector
```

CodeWarrior automatically detects the elf type and makes the settings for the Linux Application debug flow.

😣 💿 CodeWarrior Executable Importer	t.	
Import CodeWarrior executable files,	type is auto-detected	
CodeWarrior auto-detected a Linux Applica	ation executable	
Select executable: ux/odp/1.4-r0/git/exam	ple/reflector/odp_reflector Browse	·
Select type:	Details:	
Linux Application	AArch64, little endian	
○ Bare-board		
🔿 Linux Kernel		
O U-boot		
O UEFI		
Back	Next > Cancel Finish	
Figure 2. CodeWarri 4. Set the remote absolute path to odp_reflector and the function of the set	or Executable Importer he commands to execute before the application.	
Connection: ScpConnection	\$ New Edit Prop	erties
Remote Absolute File Path for C/C++ Application:		
/home/root/odp_reflector	Bro	wse
Commands to execute before application		
cnmoa +x /nome/root/oap_reflector; export DPRC=c	aprc.2	
Figure 3. Absolute path	and commands to execute	

5. If you want, you can set up directly the remote path of the odp_reflector without downloading it (use **Skip download to target path**) In this case, **check** to have the reflector from target obtained from the same yocto/rootfs build with the one imported in CodeWarrior.

CodeWarrior	Setup				
Connection:	ScpConnection	÷)[New	Edit	Properties
Remote Abs	olute File Path for C/C++ Applicatio	n:			
/usr/odp/od	dp_reflector				Browse
Commands t	o execute before application				
export DPRC	C=dprc.2				
Skip dow	nload to target path.				

Figure 4. Skip download to target path

6. Set the host/IP name of the ODP Linux target using the Edit button in the Connection area.

😣 🗉 Properties for Scp	Connection			
type filter text	Host	↓ ↓ ↓	•	
Connector Services Host	Resource type: Parent profile: System type:	Connection to remote system fsr-ub1264-121 SSH with SCP		
	Host name:	192.168.1.2	•	Search Project
	Connection name:	ScpConnection		
	Default User ID:	Iroot		-
	Description:			
	Verify host name			Settings
	Configure proxy setting	<u>2</u>		Edit
	Default encoding			
	Note: This setting can	only he changed when no subsystem is connected		

Figure 5. Scp Connection properties

7. Set sysroot in .gdbinit from the Debugger > Main tab appending this command: set sysroot <path_to_sdk>/ build_ls2085ardb_release/tmp/sysroots/ls2085ardb

🗎 Main (↔ Arguments (🏇 Debugger 🛛 🦫 Source) 🔲 <u>C</u> ommon	
Stop on startup at: main	
Debugger Options	
Main Shared Libraries Gdbserver Settings	
GDB debugger: "\${eclipse_home}/ARMv8/gdb/bin/aarch64-fsl-gdb"	Browse
GDB command file: \${eclipse_home}/ARMv8/gdb/bin/.gdbinit	Browse

Figure 6. Set sysroot

8. Set the gdbserver port used by ssh tunnel by selecting the **Debugger > Gdbserver Settings** tab.

type filter text	🗎 Main 🕬 Arguments 🕸 Debugger 🛛 🦻 Source
 C/C++ Application C/C++ Attach to Application C/C++ Postmortem Debugger C/C++ Remote Application 	Stop on startup at: main Debugger Options Main Shared Libraries Gdbserver Settings
 Debug_reflector GDB Hardware Debugging Launch Group 	Gdbserver path: gdbserver Port number: 12345

Figure 7. Gdbserver port

- 9. The odp_reflector also needs some specific arguments in order to start correctly. These can be set up using the **Arguments** tab in the **Debug Configurations** dialog. You can refer the *LS2085 SDK_Quick_Start_Guide* for details about the legal arguments for the reflector application.
- 10. Select **Run > Debug Configurations > C/C++ Remote Application**.
- 11. Click the **Debug** button to run the reflector application.

A login window appears.

12. Enter the user ID for linux target as root and the password is blank. Click OK.

Please note that now odp_reflector can only run as root and also the CodeWarrior will warn you, if necessary, about the changed RSA key mapping for root and the remote target Linux. For example, this can happen if you are restarting the Linux target and the remote Linux generates a different RSA key.

13. The connection between gdb and gdbserver is established, the gdbserver will start the odp_reflector and the bind script will be run. You can observe this in the remote shell console.

🖻 Console 🛿 🖉 Tasks 🔝 Problems 📀 Executables 🕕 Memory		х	*	B.	6 1	B	Ģ
Debug_odp_reflector[C/C++ Remote Application] Remote Shell root@ls2085ardb:~# echo \$PWD'>' /home/root> root@ls2085ardb:~# chmod +x /home/root/odp_reflector; export DPRC=dprc.2;gdbserv er :1234 /home/root/odp_reflector - Process /home/root/odp_reflector created; pid = 2015 Listening on port 1234 Remote debugging from host 192.168.1.1	-i dj	pni-	1 -m	10-	c 8;	;exi	t

Figure 8. Remote shell console

14. The reflector debug sessions starts and all the debug capabilities are available. By default, the reflector will be stopped at main() function as per the **Debugger** tab settings.

Name: Debug_reflector	
📄 Main 🏁 Arguments 🕸 Debugger 🤄 🦆 Source 🔲 <u>C</u> ommon	
Stop on startup at: main Debugger Options	
Main Shared Libraries Gdbserver Settings	

Figure 9. Debugger tab

🏘 Debug 🛙	🍇 🖉 🕘 🦸 🖬 🙀 👻 🖵 🛛	-	(x)= Variables •• Breakp	oints 🛛	1919 Registers	🛗 Peripherals	*
 Debug_odp_reflector [ector [C/C++ Remote Application] 1966] [cores: 5] 56 [core: 5] (Suspended : Breakpoint)		Ø ∞ odp_reflector.c	[line: 228	3]		
≡ main() at oc	p_reflector.c:217 0x4031fc						
局 Remote Shell 局 "/home/b3233"	I/CW/CW_NetApps_v2016.07_160229/CW_AR	Mv٤	No details to display fo	r the cur	rent selection.		
ndp_reflector.c ☎		-	🗄 Outline 🚟 Disassem	nbly 🛙	🖪 Target Conr	nections	Ente
<pre>> 217 218 odp_lin 219 odp_pool 220 int num_ 221 int i; 222 int cpu; 223 odp_cpum 224 char cpu 225 odp_pool 226 odp_plat</pre>	<pre>ux_pthread_t thread_tbl[MAX_WORKERS]; _t pool; workers; ask_t cpumask; maskstr[ODP_CPUMASK_STR_SIZE]; _param_t params; form_init_t plat_init;</pre>		<pre> 0000000004031fc: 000000000403200: 000000000403204: 000000000403208: 228 00000000040320c: 217 000000000403210: 00000000403214: </pre>	main: sub stp mov stp arg adrp { mov stp	<pre>sp, sp, # x29, x30, x29, sp x25, x26, s = calloc(1 x26, 0x44 w25, w0 x23, x24,</pre>	0x500 [sp,#-96]! [sp,#64] , sizeof(args b000 <code_ac [sp,#48]</code_ac 	:_t)); :quire

Figure 10. Debugging starts

- 15. Following are the key functions to debug:
 - a. pktio_thread
 - $b. \ \texttt{odp_schedule}$
 - $c. \ \texttt{odp_packet_from_event}$
 - $d. \ {\tt swap_pkt_addrs}$
 - e. odp_pktio_send

Set up breakpoints at all these functions from the gdb command line to see how the new threads are spawned and executed.

🗐 Console 🛿 🧔 Tasks 🖹 Problems 🜔 Executables 📋 Memory

```
Debug_odp_reflector[C/C++ Remote Application]"/home/b32331/CW/CW_NetApps_v2016.07_160229/CW_ARMv8
b pktio_thread
Breakpoint 3 at 0x403b50: file odp_reflector.c, line 159.
b odp_schedule
Breakpoint 4 at 0x40b874: file odp_schedule.c, line 541.
b odp_packet_from_event
Breakpoint 5 at 0x407604: file odp_packet.c, line 85.
b swap_pkt_addrs
Breakpoint 6 at 0x403bac: file ../../platform/linux-dpaa2/include/odp/packet_flags.h, line 206.
b odp_pktio_send
Breakpoint 7 at 0x40829c: file ./include/odp packet io internal.h, line 74.
```

Figure 11. Set breakpoints at different functions

- 16. Disable all these breakpoints from GUI, resume the application.
- 17. At the Linux container, issue a ping (that is ping 6.6.6.10 -c 10) and during this run enable all breakpoints again. The breakpoints will be hit in different conditions.

* Debug 🕅	% H - #	i> 👳	⊽ ⊓		🕬 Variables 💁 Breakpoints 🛿 🔠 Registers 🚻 Peripherals 🛋 Mod	lules
🔻 🧬 Thread #9 2025 [core: 7] (Suspended : Breakpoint)				1	🗧 🐹 😹 🕹 🖓 🕷	. 🖻
odp_packet_from_event() at odp_packet.c:85 0x407604	e.				Solution: odp_packet.c [function: odp_packet_from_event]	
pktio_thread() at odp_reflector.c:187 0x403bf4					Solution of the send of the se	
odp_run_start_routine() at linux.c:36 0x42fe20					S a odp schedule.c [function: odp schedule]	
0xfffb7faee10				0	🖉 🔊 packet_flags.h [line: 206]	
0xfffb7fce000						
📓 Remote Shell				Ų	No details to display for the current selection.	
**************************************	DAALO /A DAALO (adb /b	ala/aarehe	A Eal add	······································		
B odp_reflector.c B odp_packet.c B C 0xffffb7fafea4	🔁 nadk_conc_ldpa	a "s			🗄 Outline 🔤 Disassembly 🕴 📓 Target Connections	
81				6	Enter location here 🚽 👔 🏠	
<pre>82@ odp_packet_t odp_packet_from_event(odp_event_t ev 83 0</pre>)			2	> 000000000000000000000000000000000000	
84 return (odp packet t)ev;				1	90 }	
85 }					odp_packet_to_event:	
86 870 odn event t odn nacket to event(odn nacket t nkt)					0000000000407608: ret	
88 {					odp packet head:	
<pre>89 return (odp_event_t)pkt;</pre>					00000000040760c: ldr x0, [x0,#8]	
90 }					000000000407610: ret	
24				0	oup_packet_but_ten:	

Figure 12. Re-enable breakpoints

4.2 Attach to a running odp_reflector application and debug using CodeWarrior

- 1. Import the odp_reflector elf file as explained in Import and start the odp_reflector application from CodeWarrior.
- 2. In the **Debug Configurations** dialog, select **C/C++ attach to Application** and click **New launch configuration**.
- 3. Click the **Debugger** tab and select **gdbserver** from the **Debugger** drop-down list.
- 4. Click the **Connection** subtab, set **Type** as **TCP**. Also, set the host IP of the Linux target and the port number for the gdbserver.

Image: A state of the state	Name: Debug_odp_reflector Configuration
type filter text	🗎 Main 🏇 Debugger 🛛 🦻 Source 🔲 <u>C</u> ommon
 C/C++ Application C/C++ Attach to Application 	Debugger: gdbserver ‡
 Debug_odp_reflector Configuration C/C++ Postmortem Debugger C/C++ Remote Application Debug_odp_reflector GDB Hardware Debugging Launch Group 	Debugger Options Main Shared Libraries Connection Type: TCP ‡ Host name or IP address: 192.168.1.2 Port number: 1234

Figure 13. Set connection type

5. Start the gdbserver and the odp application standalone on the Linux target.

```
export DPRC=dprc.2
/usr/odp/bin/odp_reflector -i dpni-1 -m 0 -c 8 &
gdbserver --multi :1234
```

6. Click **Debug**. The gdb client will connect to gdbserver. Now, you can attach to any application from the Linux target, including the reflector, using the green button shown below.



Figure 14. Connect to a process

😕 🗊 🛛 Select Processes

Select one or more processes to attach to:



Figure 15. Select process

7. Click **OK** to attach to the running reflector. This enables all the debug capabilities. The stack after attach is shown below.

‡ Debug ⊠		-	M 🕘	∦ i⇒	¢9 -	~ -	
▼	uration [C/C++ Attach to Applic	ation	n]				F
▼ 🔐 odp_reflector [1951] [cores	: 0,1,2,3,4,5,6,7]						
▼ 🗗 Thread #1 1951 [core: 0] (Suspended : User Request)						
≡ 0xffffb44f0ea4							
0xffffd32d62b8							
Thread #2 1952 [core: 0] (Suspended : Container)						
Thread #3 1953 [core: 1] (Suspended : Container)						
▶ 🔊 Thread #4 1954 [core: 2] (Suspended · Container)						
							- P.

Figure 16. Debug session

8. For setting breakpoints to the global symbols, such as main, load the odp_reflector debug symbols using the *file* <*path_to_elf>* command.

```
file /home/b32331/LS2/sdk/EAR6.0/prerelease_iso/LS2085A-SDK-20160304-yocto/
build_ls2085ardb_release/tmp/work/aarch64-fsl-linux/odp/1.4-r0/git/example/reflector/
odp_reflector
```

4.2.1 odp_reflector debug from entry_point

To debug odp_reflector from entry_point:

1. You can manually start the reflector via gdbserver on the Linux target as below and then just attach with CodeWarrior to the gdbserver/odp_reflector. Note that you must run the bind script manually. This is basically the same solution presented in Import and start the odp_reflector application from CodeWarrior where all steps are made by CodeWarrior.

```
export DPRC=dprc.2
gdbserver --multi :1234 /usr/odp/bin/odp_reflector -i dpni-1 -m 0 -c 8
2. Click Debug. CodeWarrior will attach to reflector and stop at the entry_point.
```

† Debug ⊠	×	M 🕀	∄ i⇒ 🗞		3
Debug_odp_reflector Configuration [C/C++ Attach to Application]			
🔻 況 odp_reflector [1978] [cores: 4]					
🔻 🗗 Thread #1 1978 [core: 4] (Suspend	ed : User Request)				
Cxffffb7fd3d00					
"/home/b32331/CW/CW_NetApps_	v2016.07_160229/CW_ARM	v8/ARMv8	/gdb/bin/aarch6	4-fsl-gdb" (7	.8.

3. For setting breakpoints to global symbols, such as main, you should load the odp_reflector debug symbols using *file* <*path_to_elf>* command.

```
file /home/b32331/LS2/sdk/EAR6.0/prerelease_iso/LS2085A-SDK-20160304-yocto/
build_ls2085ardb_release/tmp/work/aarch64-fsl-linux/odp/1.4-r0/git/example/reflector/
odp_reflector
```

4. Now, you can set breakpoints from the gdb console, such as b main. After the hitting the breakpoint from the main function the stack will look as below.

* Debug ⊠	% H 🔿	3 i> 🍫	~
▼	lication]		
odp_reflector [1978] [cores: 4]			
Thread #1 1978 [core: 4] (Suspended : Breakpoint)			
main() at odp_reflector.c:217 0x4031fc			
"/home/b32331/CW/CW_NetApps_v2016.07_160229/CV	W_ARMv8/ARMv8,	/gdb/bin/aarch6	4-fsl-gdb" (7.8.

4.3 Debug capabilities

In the debug session, various debug capabilities are available:

1. GDB console (selected from right side). In this console, you can run gdb commands.



Figure 17. Select console

🗏 Console 🛿 🧟 Tasks	🕄 Problems 📀 Executables 🔋 Memory
Debug_odp_reflector [C	/C++ Remote Application] "/home/b32331/CW/CW_NetApps_v2016.07_160229/(
bt #0 main (argc=7, ar info thread	gv=0xfffffffffffc48) at odp_reflector.c:217
Id Target Id * 1 Thread 2002	Frame main (argc=7, argv=0xffffffffffc48) at odp_reflector.c:217

Figure 18. Console view

In the gdb traces console, you can see full trace details about the gdb client commands running on the host Linux.

In the **Remote Shell** console, you can see the remote commands (and outputs) which are executed by CodeWarrior on the remote Linux target.

In the **gdb** console, you can run gdb commands.

2. **Source path mapping resolved automatically** by the CodeWarrior software if you are running the CodeWarrior software on the same machine where the reflector is built.



Figure 19. Source path mapping resolved automatically

If not, the CodeWarrior software will ask you where the reflector source files are located as shown below.

🖻 main.c	📧 main() at /home/b32331/Desktop/LS2_setup/SDK_phase_2.0/Layerscape2-SDK-20140829-yocto/build_ls2085a-simu_rele 🛛	-	٥
Can't find a work/ls2085 Locate the f	source file at "/home/b32331/Desktop/LS2_setup/SDK_phase_2.0/Layerscape2-SDK-20140829-yocto/build_ls2085a-simu_release/ 5a_simu-fsl-linux/nadk-static/git-r0/git/apps/reflector_demo/reflector.c" file or edit the source lookup path to include its location.	:mp/	
View Disas	sembly		
Locate File			
Edit Source	e Lookup Path		

Figure 20. Not able to locate reflector source files

3. Access to registers and memory, both from the GDB and CodeWarrior views.

(x)= Variables	nts 1989 Registers 🖾	👬 Peripherals 🛋 Modules	•	
		🖾 🍕 🖻 📑 🛃	¢ą	▽
Name	Value	Description		í
▼ A General Registers	0x5 0x7ffffffc48	General Purpose and FPU	Regis	ter

Figure 21. Registers in CodeWarrior view

🔄 Console 🛱 🧟 Tasks 🕄 Problems 🔘 Executables

Debug_reflector [C/C++ Remote Application] gdb info registers pc pc 0x402538 0x402538 <main+16>

Figure 22. Registers in GDB console view

4. Run control per process, per thread

By default, the run control is enabled per process, but you can enable it per thread, using the scheduler-locking option in the gdb console. For example, you can set a breakpoint at the line 193 in odp_reflector.c, run until the breakpoint, and then you can try the scheduler-locking to be set on. Using <info threads> and <thread id>, you can switch between different threads and perform run control operations (stepi, step, next) per thread.

```
📮 Console 🛱 🧔 Tasks 🖹 Problems 🜔 Executables 📋 Memory
Debug_odp_reflector [C/C++ Remote Application] "/home/b32331/CW/CW_NetApps_v2016.07_160229/CW_ARMv8/ARMv8/gdb/bir
show scheduler-locking
Mode for locking scheduler during execution is "off".
set scheduler-locking on
info threads
 Id
      Target Id
                         Frame
                         qbman result has new result (s=s@entry=0x49cae0, dq=dq@entry=0xffffb73316c0) at qbmar
 9
       Thread 1968
 8
       Thread 1967
                         0x000000000429900 in r32 uint32 t (val=16777216, width=8, lsoffset=24) at qbman/driv
                         qb attr code decode (cacheline=0xffffb7331f44, code=0x44afb0 <code_dqrr_tok_detect>)
 7
      Thread 1966
 6
      Thread 1964
                         0x00000000004298a8 in qb attr code decode (cacheline=0xffffb73327c4, code=0x44afb0 <<
 5
      Thread 1963
                         gbman swp pull (s=s@entry=0x4a22a0, d=0xffffb47fe8a8, d@entry=0xffffb47fe8f8) at qbma
  4
      Thread 1962
                         odp packet from event (ev=0xffffb6f5f400) at odp packet.c:85
 3
       Thread 1961
                         qb attr code decode (cacheline=0xffffb7333484, code=0x44afb0 <code dqrr tok detect>)
                         pktio thread (arg=<optimized out>) at odp_reflector.c:193
 2
       Thread 1965
 1
       Thread 1958
                         0x0000ffffb7fafea4 in ?? ()
thread 2
[Switching to thread 2 (Thread 1965)]
#0 pktio thread (arg=<optimized out>) at odp reflector.c:193
                        pktio_tmp = odp_packet_input(pkt);
193
thread 1
[Switching to thread 1 (Thread 1958)]
#0 0x0000ffffb7fafea4 in ?? ()
stepi
```

Figure 23. Run control per process, per thread

5. **OS Resources**: You can view information about processes (PID), Threads (TID), Sockets, shared-memory regions. To open this view, select **Window > Show view > Other > OS Resources**.

📮 Console 🧔 Ta	asks 🔝 Prol	blems 🜔 Executa	bles 🚺 Memo	ry 🤔 OS Reso	ources 🖾			Sockets	•
Local address 🔻	Local port	Remote address	Remote port	State	User	Family	Protocol		
0.0.0.0	111	0.0.0.0	0	LISTEN	root	INET	STREAM		
0.0.0.0	1234	0.0.0.0	0	LISTEN	root	INET	STREAM		
0.0.0	22	0.0.0.0	0	LISTEN	root	INET	STREAM		
0.0.0.0	23	0.0.0.0	0	LISTEN	root	INET	STREAM		
0.0.0.0	52280	0.0.0.0	0	LISTEN	rpcuser	INET	STREAM		
0.0.0.0	12865	0.0.0.0	0	LISTEN	root	INET	STREAM		
0.0.0.0	111	0.0.0.0	0	CLOSE	root	INET	DGRAM		
3.0.0.0	986	0.0.0.0	0	CLOSE	root	INET	DGRAM		
0.0.0.0	59210	0.0.0.0	0	CLOSE	rpcuser	INET	DGRAM		
127.0.0.1	998	0.0.0.0	0	CLOSE	root	INET	DGRAM		
192.168.1.2	1234	192.168.1.1	33503	ESTABLISHED	root	INET	STREAM		
192.168.1.2		192.168.1.1	43433	ESTABLISHED	root	INET	STREAM		

Figure 24. OS Resources view

🕒 Con	sole	🖉 Tas	ks 🖹 Problems 🔘 Executables 🔋 Memory 🖉 OS Resources 😂 👘
Pid 🔻	Tid	Соге	Command
1687	1687	6	syslogd
1689	1689	2	klogd
1697	1697	7	xinetd
1702	1702	0	sh
1771	1771	1	udevd
1946	1946	0	sshd
1954	1954	2	sh
1957	1957	0	gdbserver
1958	1958	0	odp_reflector
958	1961	0	odp_reflector
1958	1962	1	odp_reflector
1958	1963	2	odp_reflector
1958	1964	3	odp_reflector
1958	1965	4	odp_reflector
1958	1966	5	odp_reflector
1958	1967	6	odp_reflector
1958	1968	7	odp_reflector

Cor	isole 🧟	Tasks 🔝 Pr	oblems 🜔 Executables 🔋 Memory 🖉 OS Resources 🛙	Processes	*	1
Pid +	User	Cores	Command			
1040	root	7	[kworker/7:1]			
1056	root	3	/lib/udev/udevd -d			
1132	root	3	/lib/udev/udevd -d			
1580	root	0	[kworker/0:2]			
1653	root	0	/usr/sbin/sshd			
1663	rpc	2	/usr/sbin/rpcbind			
1670	rpcuser	3	/usr/sbin/rpc.statd			
1678	root	6	/usr/sbin/netserver			
1687	root	6	/sbin/syslogd			
1689	root	2	/sbin/klogd			
1697	root	7	/usr/sbin/xinetd -pidfile /var/run/xinetd.pid -stayalive			
1702	root	0	-sh			
1771	root	1	/lib/udev/udevd -d			
1946	root	0	sshd: root@pts/0			
1954	root	2	-sh			
1957			gdbserver :1234 /home/root/odp_reflector -i dpni-1 -m 0 -c 8			
1958	root	0,1,2,3,4,5,6	6,7 /home/root/odp_reflector -i dpni-1 -m 0 -c 8			

6. **Breakpoints** can be read/written from the IDE or gdb console.





7. Full Remote Shell console. You can enable it from Window > Show view > Other Remote Shell.

🖫 Remote Shell 🛛			a.	~ -	
	Local	Launch	•		
	ScpConnection				

Figure 26. Remote Shell view

```
🖳 Console 🧔 Tasks 🖹 Problems 🕡 Executables 🏮 Memory 🖉 OS Resources 🖳 Remote Shell 🗱
📓 ScpConnection 😂
   root
            1958 14.1 0.1 101080 18344 pts/0
                                                tLl+ 16:49
                                                             1:05 /home/root/odp
            1970 0.0 0.0
                             3252 2532 pts/1
                                                Ss
                                                     16:56
                                                             0:00 -sh
   root
   root
            1973 0.0 0.0
                             2928 1624 pts/1
                                                R+
                                                     16:56
                                                             0:00 ps aux
   root@ls2085ardb:~#
   root@ls2085ardb:~#
  ls -la
  total 1740
                          1024 Feb 29 16:49 .
   drwxr-xr-x 2 root root
  drwxr-xr-x 3 root root
                          1024 Feb 17 14:10 ...
   -rw----- 1 root root
                              66 Feb 29 16:49 .bash history
   -rwxr-xr-x 1 root root 1770424 Feb 29 16:49 odp_reflector
   root@ls2085ardb:~#
/home/root>
   root@ls2085ardb:~#
   root@ls2085ardb:~#
Command
```

8. **Dynamic printf** – Right-click the left side panel of the source code to see the menu below. A dynamic printf works like a breakpoint and when this is hit, a custom message is printed. This feature is very useful for debug.

CodeWarrior Setup									
훱 odp_reflector.c 🛛	🔁 odp_packet.c	C 0xffffb7fafe	4 📆	qbman_	private.h	**8		-	
192 /* Er	nqueue the packet	for output */							8
Toggle Breakpoint	t Shif	t+Ctrl+B tmp, &	p <mark>kt, 1</mark>) e failed) { d.\n", 1	thr);) +	
	Fig	jure 27. Dynan	nic prin	tf					
(x)= Variables	Breakpoints 🛛 🖁	🕅 Registers 🎁	l Peripl	nerals	🛋 Modu	lles		-	٥
			×	× 🔐		Ŧ	B	4 20	▽
🐱 🕫 опр_раск	et.t [une. oz]								
🛛 🗢 odp_refle	ctor.c [line: 190]								
🗸 🌽 odp_refle	ctor.c [line: 193] ["Hit line %d of	/home/	/b3233	1/LS2/sd	k/EAR	6.0/p	огеге	ele
No details to disc	olav for the curre	nt selection.							

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 registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2016-18 NXP B.V.

Document Number AN5269 Revision 06/2018



