Freescale Semiconductor Application Note

Linux kernel and modules debug using CodeWarrior for QorlQ LS series – ARM V7 ISA

1. Introduction

This document describes the steps required for Linux kernel and modules debugging using the CodeWarrior for QorIQ LS series – ARM V7 ISA.

This document includes the following sections:

- Build the Linux sources.
- Perform Linux kernel and modules debug in CodeWarrior for QorIQ LS series ARM V7 ISA.

2. Preliminary background

The following are the steps required to compile Linux for the LS1021AQDS board.

Contents

1.	Introduction1
2.	Preliminary background1

- 4. Linux kernel debug support8
- 5. Linux kernel and module debug......10



Preliminary background

2.1. Downloads

Before Linux kernel and modules debug, download LS1021A SDK ISO.

2.2. Install the SDK

To install SDK on the host machine, perform these steps:

1. Mount the ISO on your machine

\$ sudo mount -o loop LS1021A-SDK-<version>-<target>-<yyyymmdd>yocto.iso /mnt/cdrom

2. As a non-root user, install the SDK:

```
$ cd /mnt/cdrom
$ ./install
```

3. When prompted to input the install path, ensure that the current user has correct permissions for the install path.

NOTE There are no uninstall scripts. To uninstall Yocto, you can remove the <yocto_install_path>/LS1021A-SDK-<version>-<target>-

2.3. Host Environment

Yocto requires some packages to be installed on host folder. The following steps are used for preparing Yocto:

```
1. $ cd <yocto_install_path>
```

```
2.$ ./scripts/host-prepare.sh
```

```
3.$ source ./fsl-setup-poky -m <machine>
```

NOTE For example, for LS1021AQDS board, the above command will be: \$ source ./fsl-setup-pocky -m ls1021aqds -j 4 -t 4, where -j is the number of jobs to spawn during the compilation stage and -t is the number of BitBake tasks that can be issued in parallel.

2.4. Builds

To build various packages, the following steps need to be performed:

```
1. $ cd <yocto install path>/
```

```
2. $ source ./build <machine> release/SOURCE THIS
```

```
3. $ bitbake <package-recipe>
```

NOTE u-boot, rcw, kernel, dtb, and rootfs images are can be found in: build_<machine>_release/tmp/deploy/images/<machine>.

2.5. Linux Kernel

In some cases, it is necessary to configure and rebuild the Linux Kernel. In our case, this is necessary for adding the debug symbols.

1. Do menuconfig

\$ bitbake -c menuconfig virtual/kernel

- 2. The kernel configuration window will be opened. Go to *Kernel hacking > Compile-time checks and compiler option* and select **Compile the kernel with debug info** checkbox.
- 3. Save the new configuration and rebuild the Linux Kernel

```
$ bitbake -c virtual/kernel
```

NOTE	E a. vmlinux image with debug symbols can be found in folder:		
	<pre>build_<machine>_release/tmp/work/<machine>-fsl-linux-</machine></machine></pre>		
	gnueabi/linux-layerscape-sdk/3.12-r0/git/		
	b. vmlinux elf file will be imported in CodeWarrior for QorIQ LS series – ARM V7		
	ISA.		

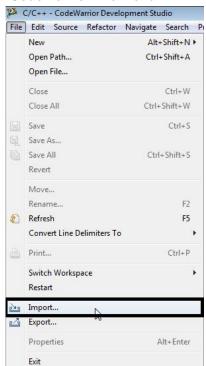
3. Create ARMv7 project

To create an ARMv7 project for Linux kernel debug, follow these steps:

- 1. Start CodeWarrior for QorIQ LS series ARM V7 ISA.
- 2. Select **File** > **Import** and import the vmlinux executable file generated during the Linux kernel compilation. For details, see <u>Linux Kernel</u> section.

Create ARMv7 project

Figure 1. CodeWarrior File menu



3. Choose the source to Import and click Next.

Figure 2. Import wizard

Import 1	
select Import a CodeWarrior Executable file and create a project	r La
Select an import source:	
type filter text	
C/C++ CodeWarrior CodeWarrior Executable Importer Example Project	
⊱ Install ⊱ Run/Debug ⊱ Team	
🗁 Run/Debug	

4. Specify **Project name** and **Location**, or use the default location and click **Next**.

Figure 3. Importing executable file page

🊨 Import a Cod	eWarrior Executable file	
Import a Code	Warrior Executable file	
Choose the loc	ation for the new project	
Project name:	LinuxKernelDebug	
☑ Use <u>d</u> efault	location	
Location: C:\	Users\b11883\workspace\armv7 140707\LinuxKernelDebug	Browse
?	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish	Cancel
0		

5. Browse to the vmlinux executable file and select Open. By default, CodeWarrior looks for an .elf extension, so change the file type in the lower right corner of Select File dialog.

Create ARMv7 project

Figure 4. Select vmlinux executable file page

F 1 1 1			
File to import Copy the selected file t	o current project folder	Browse	
🥦 Select file			
🔾 🗢 📕 « linux-la	ıyerscape-sdk ► 3.12-r0 ► git ►	✓ Search git	
Organize 🔻 New fo	lder	8==	- 🗆 (
Favorites	Name	Date modified	Туре
Desktop	.version	7/4/2014 9:46 AM	VERSION Fi
Desktop	.vmlinux	7/4/2014 9:47 AM	Windows C
Recent Places		7/4/2014 9:19 AM	File
Recent Places	CREDITS	7/4/2014 9:19 AM	File
📜 Libraries	Kbuild	7/4/2014 9:19 AM	File
Documents	Kconfig	7/4/2014 9:19 AM	File
Music		7/4/2014 9:19 AM	File
Pictures	Makefile	7/4/2014 9:19 AM	File
Videos	Module.symvers	7/4/2014 9:47 AM	SYMVERS F
	modules.builtin	7/4/2014 9:47 AM	BUILTIN File
🖳 Computer	modules.order	7/4/2014 9:47 AM	ORDER File
Primary (C:)	README	7/4/2014 9:19 AM	File
♀ sdk (\\10.171.72.1	REPORTING-BUGS	7/4/2014 9:19 AM	File
🖵 space (\\zro04file	System.map	7/4/2014 9:47 AM	MAP File
engdata (\\zro04	vmlinux	7/4/2014 9:47 AM	File
	wnlinux.o	7/4/2014 9:46 AM	O File

6. Select **Processor** type, **Linux Kernel** as Target OS and click **Next**.

Figure 5. Processor page

P Import a CodeWarrior Executable file	
Processor	
Choose the processor for this project	
Processor	
type filter text	
Layerscape Family QorlQ_LS1	
LS1020A LS1021A	
L\$1022A	
Toolchain	
Bareboard Application	
Clinux Application	
Target OS	
None None	
Linux Kernel	
(?) < Back Next >	Einish Cancel

7. Select Debugger Connection Types, Board, Launch, Connection Type and click Next.

🥦 Import a CodeWarr	ior Executable file		- • •
Debug Target Setti	ngs		
Target Settings			
Debugger Connectio	n Types		
Hardware			
Emulator			
Board LS	1021AQDS -		
Launch (Connection		
Download	🗳 Default	*	
V Attach	🗳 Default	-	
	2		
Connection Type Co	odeWarrior TAP (over USB) 🔹		
TAP address			
2	< Back	Next > Fini	sh Cancel
\odot	- Dack	Finit	Cancel

Linux kernel and modules debug using CodeWarrior for QorIQ LS series – ARM V7 ISA Application Note Freescale Semiconductor Linux kernel debug support

8. Select the **Configurations** that you want to create and then click **Finish** to close the wizard.

Figure	7.	Configurations	page
	•••	e e i i i gai a i e i i e	page

Marrior Executable file	- • •
Configurations	
Choose the configurations you want to create	
Core index	
Core 0	
Core 1	
	Cancel

4. Linux kernel debug support

4.1. Debugger settings for Linux kernel debugging

The vmlinux executable file generated during the Linux kernel compilation should be imported as CodeWarrior project (for more information, see <u>Create ARMv7 project</u> section).

After the CodeWarrior project is created, perform these steps to start Linux kernel debug:

1. Select **Run** > **Debug configurations**, to open **Debug configurations** dialog and click **Debug**.

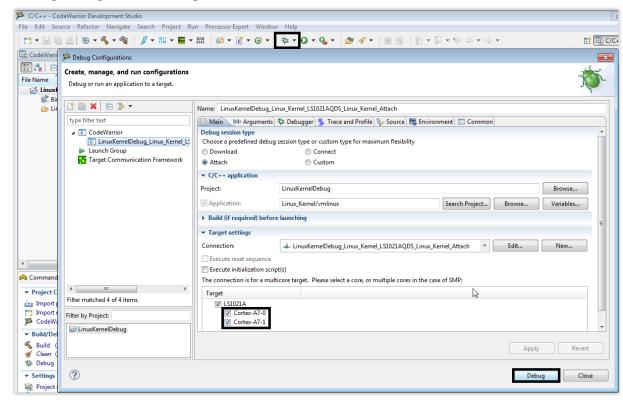


Figure 8. Debug Configurations dialog

NOTE Both cores should be selected. Also, make sure no initialization file is used.

- 2. Go to **Debugger** -> **OS Awareness**. Since an Attach Launch configuration is used to attach to a running Linux kernel, make sure that all the checkboxes are disabled.
- 3. In the **Debug** tab, you need to select the **Enable Memory Translation** checkbox and use the settings as described in the figure below for early kernel debug capabilities. Also, select the **Enable Threaded Debugging Support** and **Enable Delayed Software Breakpoint Support** checkboxes.

Figure 9.OS Awareness – Debug tab

Name: LinuxKernelDebug_Lin	nux_Kernel_LS1021	AQDS_Lir	ux_Kernel_Attach			
📄 Main 🛛 Arguments 🕉	E Debugger 🛛 📕	Trace an	d Profile) 🧤 Source	📧 Environ	ment 🔲 Common	
Debugger options						
Debug Exceptions Inter	rrupts Download	I PIC	Other Executables	Symbolics	OS Awareness	
Target OS: Linux 🔻						
Boot Parameters Debug						
Boot Parameters Debug	g Modules					
Enable Memory Tra	nslation		L. L.			
Physical Base Address	0x80000000		1			
Virtual Base Address	0x80000000					
Memory Size	0x6f800000					
Enable Threaded Deb	Enable Threaded Debugging Support					
Update Background 1	Threads on Stop					
Enable Delayed Softw	are breakpoint su	Enable Delayed Software Breakpoint Support				

4. In the **Modules** tab, you need to select the **Detect module loading** and **Prompt for symbolics path if not found** checkboxes. These options are necessary for automatic insertion/removal detection of kernel modules.

Figure 10. OS Awareness – Module tab

ame: LinuxXernelDebug_Linux_Kernel_LS1021AQDS_Linux_Ke Main (40- Arguments (54: Debugger) Trace and Prof Debugger options Debug Exceptions Interrupts Download PIC Oth Target OS: Linux		
Boot Parameters Debug Modules		
Module	Symbolics Path	Add Scan Remove Remove All
Prompt for symbolics path if not found Keep target suspended		

5. Linux kernel and module debug

5.1. Linux kernel debugging

1. Power on the board and stop to U-Boot console.

Figure 11. Target stopped at u-boot prompt

Hit	any	key	to	stop	autoboot:	0	
=>							

2. Attach to U-Boot using Attach launch.

Figure 12. The Attach Launch configuration

	nents) 🕸 Debugger) 💺 Trace and Profile) 🧤 Source	🔚 🚾 Environment 🕅 🔲 Common		
Debug session type				-
	debug session type or custom type for maximum flexib	ility		
Download	Connect			
Attach	Custom			
 C/C++ application 				
Project:	LinuxKernelDebug		Browse	
✓ Application:	Linux_Kernel/vmlinux	Search Project	Browse Variables	
Build (if required)	before launching			
 Target settings 				E
Connection:	LinuxKernelDebug_Linux_Kernel_LS1021AC	DS_Linux_Kernel_Attach 🔹 📄	Edit New	
Execute reset sequ				
	ence			
Execute initialization				
Execute initialization		es in the case of SMP:		
Execute initialization	on script(s)	es in the case of SMP:		
Execute initialization The connection is for	on script(s)	es in the case of SMP:		
Execute initialization The connection is for Target I LS1021A Cortex-A7	n script(s) a multicore target. Please select a core, or multiple cor -0	es in the case of SMP:		
Execute initialization The connection is for Target	n script(s) a multicore target. Please select a core, or multiple cor -0	es in the case of SMP:		
Execute initialization The connection is for Target I LS1021A Cortex-A7	n script(s) a multicore target. Please select a core, or multiple cor -0	es in the case of SMP:		
Execute initialization The connection is for Target I LS1021A Cortex-A7	n script(s) a multicore target. Please select a core, or multiple cor -0	es in the case of SMP:		
Execute initialization The connection is for Target I LS1021A Cortex-A7	n script(s) a multicore target. Please select a core, or multiple cor -0	es in the case of SMP:	Apply Revert	

3. Set a breakpoint at kernel entry point, using Debugger Shell command: bp -hw 0x80008000

Figure 13. Setting breakpoint at entry point in Debugger Shell

🕸 Debugger Shell 🛛 🦪 System Browser		Þ	
%>bp -hw 0x80008000			
id instance address type enabled? process			
description			
#36 #1 x:0x80008000 -hw ENABLED \$0			
head.S, line 87, (AsmSection) [vmlinux]			
%>			

Linux kernel and modules debug using CodeWarrior for QorIQ LS series – ARM V7 ISA Application Note Freescale Semiconductor

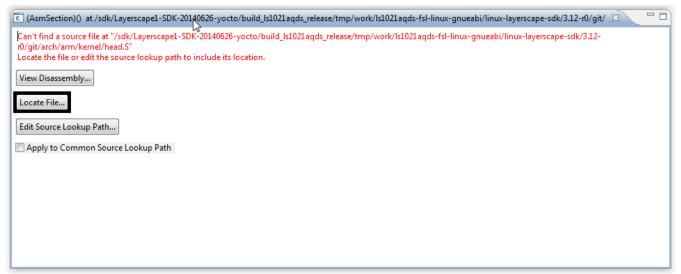
4. Start kernel from U-Boot console.

Figure 14. U-boot log - preparing the images for starting the Linux kernel

```
## Booting kernel from Legacy Image at 82000000 ...
  Image Name: Linux-3.12.0+
  Image Type:
                ARM Linux Kernel Image (uncompressed)
  Data Size: 3053688 Bytes = 2.9 MiB
  Load Address: 80008000
  Entry Point: 80008000
  Verifying Checksum ... OK
# Loading init Ramdisk from Legacy Image at 88000000 ...
  Image Name: fsl-image-core-ls1021aqds-201406
  Image Type: ARM Linux RAMDisk Image (gzip compressed)
               19170910 Bytes = 18.3 MiB
  Data Size:
  Load Address: 00000000
  Entry Point: 00000000
  Verifying Checksum ... OK
# Flattened Device Tree blob at 8f000000
  Booting using the fdt blob at 0x8f000000
  Loading Kernel Image ... OK
  Loading Ramdisk to cedb7000, end cffff65e ... OK
  Loading Device Tree to cedae000, end cedb6a91 ... OK
Starting kernel ...
```

5. The breakpoint set at step 5 will be hit and CodeWarrior will ask for the location of the Linux kernel sources in order to make a path mapping between the original location of the sources and the new location (for example, the Linux kernel sources were copied from a Linux machine on a Windows machine).

Figure 15. Source file not found when target is stopped at kernel entry point



6. After locating the missing file, CodeWarrior will display in Source view the actual source file.

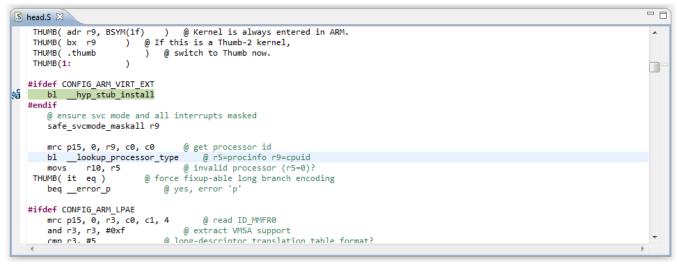
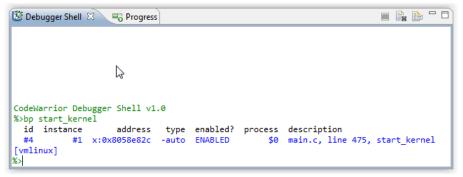


Figure 16. Target stopped at entry point, after path mapping was performed

7. To start kernel debug from start_kernel symbol, set a breakpoint at start_kernel, using **Debugger Shell** command: bp start_kernel.

Figure 17. Setting a breakpoint from Debugger Shell at 'start_kernel' method



8. **Resume** using F8 or use **Debugger Shell** command: go. The breakpoint will be hit and kernel debugging can be performed from start kernel.

Figure 18. Target stopped at 'start_kernel' method

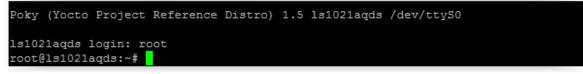
```
main.c S
    percpu_init_late();
    pgtable_cache_init();
    vmalloc_init();
    vmalloc_init();
    char * command_line;
    extern const struct kernel_param __start__param[], __stop__param[];
    /*
    * Need to run as early as possible, to initialize the
    * lockdep hash:
    */
    lockdep_init();
    smp_setup_processor_id();
    debug_objects_early_init();
    /*
    * Set up the the initial canary ASAP:
    */
```

9. At this point, you are able to perform full Linux kernel debug using run control (step/run/suspend), set/remove breakpoints, read/write memory/registers/variables, etc.

5.2. Linux kernel module debugging

1. Login to Linux.

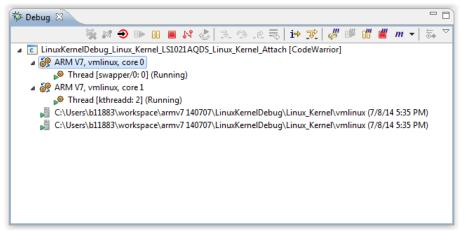
Figure 19. Linux prompt after logging in



2. Debugger is already attached to the target. If not, Attach to Linux using the Attach Launch configuration.

.

Figure 20. CodeWarrior attached to two running cores



3. Insert a module into Linux.

root@ls1021aqds:~# modprobe isofs

- 4. CodeWarrior will automatically detect any insmod/modprobe/rmmod operation. A pop-up window will be triggered for locating the module debug symbols.
 - NOTE In order to detect insertion/removal of kernel modules, CodeWarrior needs to be configured accordingly in the Debug configuration, Debugger tab -> OS Awareness tab -> Modules tab.

Figure 21. Prompt for 'isofs' symbolics file

Accate symbolics file for is	ofs			×
	git ▶ fs ▶ isofs 🔹 🕹	Search isofs		٩
Organize 🔻 New fold		:==	•	0
☆ Favorites	Name	Date modified	Туре	
ktop	📄 isofs.ko	7/4/2014 9:47 AM	KO File	
🝌 Downloads				
E Recent Places				
🛱 Libraries				
Documents	\searrow			
🎝 Music 🗉				
Pictures				
Videos				
🖳 Computer				
Primary (C:)				
🚽 sdk (\\10.171.72.1				
🚽 space (\\zro04file				
🚽 engdata (\\zro04 [.]				
🖬 Network 🔻	< III			F.
File n	ame: isofs.ko 👻	*.ko		•
		Open	Cancel	
		Open	Cancer	

NOTE It is mandatory that the kernel image running on the target is the same with the vmlinux image in debugger, in order to have the kernel modules insertion/removal detection enabled.

Figure 22. Target stopped in do_init_module after detection that an insmod/modprobe was performed

🔞 module.c 🛿		
<pre>blocking_notifier_call_chain(&module_notify_list, MODULE_STATE_COMING, mod);</pre>	~	
<pre>/* Set RO and NX regions for core */ set_section_ro_nx(mod->module_core,</pre>		
<pre>/* Set RO and NX regions for init */ set_section_ro_nx(mod->module_init,</pre>		
<pre>mod->init_ro_size,</pre>		
<pre>mod->init_size);</pre>		
<pre>do_mod_ctors(mod); /* Start the module */ if (mod->init != NULL) ret = do one_initcall(mod->init); </pre>]-
if (ret < 0) {		
/* Init routine failed: abort. Trv to protect us from	•	

Linux kernel and modules debug using CodeWarrior for QorIQ LS series – ARM V7 ISA Application Note

5. **System Browser** can be used to see information about Kernel version, modules and threads running on each core.

Figure 23. Kernel modules list displayed in System Browser

🦉 Debugger Shell 🕞 Progr	ess 🗐 System Brov	vser 🛛						
🛆 Linux Kernel Awareness	ARM V7, vmlinux, core 0 (Supervisor mode/Secure)							
Information	Name	Kernel Address	Text Address	Core Size	Symbolics Loaded			
Threads	isofs	0x7f005238	0x7f000000	29503	True			
Threads Modules	isofs	0x7f005238	0x7f000000	29503	True			
Modules								

6. For module debug, module's sources should be opened in CodeWarrior. Debugging (step, run, or breakpoint) can be done for the inserted module's.

How to Reach Us:

Home Page: www.freescale.com

E-mail: support@freescale.com Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or 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: <u>freescale.com/SalesTermsandConditions</u>.

Freescale, the Freescale logo, CodeWarrior, and QorlQ 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 and TrustZone are trademarks or registered trademarks of ARM Ltd or its subsidiaries in the EU and/or elsewhere. All rights reserved.

© 2014 Freescale Semiconductor, Inc.

