

Freescale Semiconductor Application Note Document Number: AN4980

# Debugging Linux Kernel and Modules using CodeWarrior for QorlQ LS series -ARM V7 ISA

# 1. Introduction

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

This document includes the following sections:

- Preliminary background
- Creating an ARMv7 project
- Linux kernel debug support
- Debugging Linux kernel and modules

# 2. Preliminary background

This section describes the steps required to compile Linux image for the LS1 boards.

### Contents

1.	ntroduction1	
2.	Preliminary background1	

- 4. Linux kernel debug support ......9
- 5. Debugging Linux kernel and modules.....12





Preliminary background

## 2.1. Download SDK

To debug Linux kernel and modules using CodeWarrior, download the latest QorIQ SDK from <u>www.freescale.com</u>.

# 2.2. Install SDK

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

1. Mount the ISO on your machine as illustrated below:

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

2. Install the SDK as a non-root user, using the following commands:

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

3. On seeing the prompt to input the install path, check and ensure that the current user has the appropriate permissions for the installation path.

**NOTE** There are no uninstall scripts. To uninstall Yocto, you need to remove the <yocto\_install\_path>/LS1021A-SDK-<version>-<target>-

### 2.3. Prepare host environment for Yocto

Yocto requires some packages to be installed on the host folder. Prepare Yocto for the host environment using the commands below:

\$ cd <yocto\_install\_path> \$ ./scripts/host-prepare.sh \$ source ./fsl-setup-poky -m <machine>

Below is an illustration of this command:

For example, for LS1021AQDS board, the above command will be:

\$ source ./fsl-setup-poky -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. Build Packages

To build various packages, run the following commands:



\$ cd <yocto\_install\_path>/
\$ source ./build\_<machine>\_release/SOURCE\_THIS
\$ bitbake <package-recipe>

**NOTE** U-Boot, RCW, kernel, DTB, and rootfs images can be found in: build\_<machine>\_release/tmp/deploy/images/<machine>

### 2.5. Configure and rebuild Linux kernel

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

1. Run the *bitbake* command with *menuconfig*:

\$ bitbake -c menuconfig virtual/kernel

- 2. On the kernel configuration window that opens, go to Kernel hacking > Compile-time checks and compiler option and select the Compile the kernel with debug info checkbox.
- 3. Save the new configuration and rebuild the Linux kernel using the command below:

```
$ bitbake virtual/kernel
```

**NOTE** You can find the *vmlinux* image, with debug symbols, in the following folder: build\_<machine>\_release/tmp/work/<machine>-fsl-linuxgnueabi/ linux-layerscape-sdk/3.12-r0/git/

The vmlinux ELF file will be imported into CodeWarrior for QorIQ LS series - ARM V7 ISA.

# 3. Creating an ARMv7 project

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

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



#### Figure 1. CodeWarrior File menu



3. On the Select page of the Import wizard, choose the source to be imported and click Next.

#### Figure 2. Import wizard

4

Import a CodeW	arrior Executable file and create a project	r L
Select an import	source:	
type filter text		
C/C++ CodeWi CodeWi Exan France Run/Del	arrior eWarrior Executable Importer nple Project bug	
👌 Ieam		



- 4. On the Import a CodeWarrior Executable file page, specify the project name.
- 5. Specify a location for the new project in the Location field or choose to use the default location.
- 6. Click Next.

Figure 3. Import a CodeWarrior Executable file page

🎾 Import a CodeWarrior Executable file	
Import a CodeWarrior Executable file	
Choose the location for the new project	
Project name: LinuxKernelDebug	
Location: C:\Users\b11883\workspace\armv7 14070	I7\LinuxKernelDebug B <u>r</u> owse
? < <u>B</u> ack	<u>N</u> ext > <u>Finish</u> Cancel

- 7. Browse to the *vmlinux* executable file and select **Open**.
- 8. By default, CodeWarrior looks for an .*elf* extension. Therefore, change the file type in the lower right corner of the **Select file** dialog, as shown in the figure below.







- 9. On the **Processor** page that appears, select the processor type.
- 10. Under Target OS, choose Linux Kernel and click Next.



#### Figure 5. Processor page

Import a CodeWarrior Executable file	
Processor	
Choose the processor for this project	
Processor	
type filter text	
▲ Layerscape Family	
⊿ QorIQ_LS1	
LS102MA	
LS1020A	
LS1021A	
LS1022A	
LS1024A	
loolchain	
Bareboard Application	
Clinux Application	
Target OS	
None	
Inux Kernel	

11. On the **Debug Target Settings** page that appears, select debugger connection type, board, launch configuration, and connection type, and click **Next**.



### Figure 6. Debug Target Settings page

🏜 Import a CodeWa	rrior Executable file		
Debug Target Set	tings		
Target Settings			
Debugger Connect	ion Types		
Hardware			
Emulator			
Board L	S1021AODS -		
Launch	Connection		
Download	上 Default	Ŧ	
🔽 Attach	上 Default	-	
	R		
Connection Type	CodeWarrior TAP (over USB) 🔻		
TAP address			
2	< Back	Next > Fi	nish Cancel

12. On the **Configurations** page that appears, select the configurations that you want to create and click **Finish** to close the wizard.



### Figure 7. Configurations page

P Import a CodeWarrior Executally file	
Configurations	
Choose the configurations you want to create	
Core index	
Core 0	
Core 1	
(?) < Back Next > Finish	Cancel

# 4. Linux kernel debug support

This section describes the 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>Creating an ARMv7 project</u>).

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

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



### Linux kernel debug support

### Figure 8. Debug Configurations dialog

🎾 C/C++ - Coo	deWarrior Development Studio						c
File Edit Sou	irce Refactor Navigate Search Project Ru	n Processor Expert Window	/ Help				
1 - 8 6	🖹 🕲 <b>+ % + %</b> 🥖 <b>+ 111 + 111 +</b>	🛗 - C - G -	🎄 🕶 🖸 🕶 💁 🛷 🕶 💷 🖞 👻	<b>₽ • \$</b> \$ \$ • \$	•	E	<u>ץ פור כ∖כ+</u>
CodeWarri	🥦 Debug Configurations						×
Eile Name	Create, manage, and run configurations					1	K
	Debug or run an application to a target.					R	<b>N</b>
🐰 Bir 🅞 Lir		Name: LinuxKernelDebug_Lin	nux_Kernel_LS1021AQDS_Linux_Kernel_Attach				
	type filter text	Main (X)= Arguments	🕸 Debugger 📑 Trace and Profile 🤤 Source 👼 Enviro	onment 🔲 Common			
	▲ C CodeWarrior	Debug session type					^
	C LinuxKernelDebug_Linux_Kernel_L	Choose a predefined debug	session type or custom type for maximum flexibility				
	Target Communication Framework	Download     Attach	Connect				
		<ul> <li>C/C++ application</li> </ul>	Caston				
		Project:	LinuxKernelDebug			Browse	
		The state			_		1
		Application:	Linux_Kernel/vmlinux	Search Project	Browse	Variables	
		Build (if required) before	e launching				E
		<ul> <li>Target settings</li> </ul>					
		Connection:	= LinuxKernelDebug_Linux_Kernel_LS1021AQDS_Linux_	Kernel_Attach 👻	Edit	New	
		Execute reset sequence					
Commund		Execute initialization scrip	ot(s)				
Command	4 111	The connection is for a mult	icore target. Please select a core, or multiple cores in the ca	ise of SMP:			
✓ Project C	Filter matched 4 of 4 items	Target		S.			
Import	Eilter by Dreiert	Cortex-A7-0					
🥦 CodeWa		Cortex-A7-1					-
▼ Build/Del							
🔏 Build (					Apply	Rever	£
愛 Clean ( 参 Debug							
▼ Settings	?				Debu	g Clo	se
Roject :	Ŭ						

**NOTE** Ensure that no initialization file is used.

- 2. On the **Debugger** tab, open the **OS** Awareness tab.
- 3. Deselect all the checkboxes, because an Attach launch configuration is used to attach to a running Linux kernel.
- 4. On the **Debug** tab, select the **Enable Memory Translation** checkbox.
- 5. Configure the remaining settings as shown in the figure below.



Figure 9.OS Awareness – Debug tab

Na	Name: LinuxKernelDebug_Linux_Kernel_LS1021AQDS_Linux_Kernel_Attach								
	📄 Main 🕬 Arguments 🕸 Debugger 📃 Trace and Profile) 🦆 Source 🚾 Environment 🔲 Common 🗌								
lr'	Debugger options								
	Debug	Exceptions	Interrupts	Download	PIC	Other Executables	Symbolics	OS Awareness	
	Target (	OS: Linux 🔻	•						
	Boot F	arameters [	Debug Mo	dules					
	- 🔽 E	nable Memory	y Translatio	n		1			
	Phys	sical Base Add	ress 0x800	00000					
	Virtu	ial Base Addre	ess 0x800	00000					
	Memory Size 0x6f800000								
	Enable Threaded Debugging Support								
	Up	date Backgrou	und Thread	s on Stop					
	Opdate background Threads on Stop     Inable Delayed Software Breakpoint Support								

6. On the **Modules** tab, select the checkboxes labeled "Detect module loading" and "Prompt for symbolics path if not found".

NOTEThese options are required for detecting automatic insertion/removal of kernel<br/>modules.<br/>If multiple modules are inserted at Linux boot, then you are recommended to<br/>activate these options only when connecting to Linux for module debugging.

#### Figure 10. OS Awareness - Modules tab

ebug Exceptions Interrupts Download I	PIC Other Executables Symbolics OS Awareness	
arget OS: Linux 💌		
Boot Parameters Debug Modules		
V Detect module loading		
Modules' symbolics mappings		
Module	Symbolics Path	Add
		Scan
		Remove
		Remove All



# 5. Debugging Linux kernel and modules

This section explains how to debug Linux kernel and Linux modules.

## 5.1. Debugging Linux kernel

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

### Figure 11. Target stopped at U-Boot prompt



2. Attach to U-Boot using Attach launch configuration, as shown in the figure below.

### Figure 12. Attach launch configuration

Name: LinuxKernelDebug	_Linux_Kernel_LS1021AQDS_Linux_Kernel_Attach		
📄 Main 🛛 🕬= Argumer	ts) 🕸 Debugger) 💺 Trace and Profile) 🧤 Source	📧 Environment 🔲 Common	
Debug session type Choose a predefined del	ug session type or custom type for maximum flexibi	ility	
<ul> <li>Download</li> <li>Attach</li> </ul>	<ul> <li>Connect</li> <li>Custom</li> </ul>		
▼ C/C++ application			
Project:	LinuxKernelDebug		Browse
Application:	Linux_Kernel/vmlinux	Search Project Bro	wse Variables
• Build (if required) be	fore launching		
<ul> <li>Target settings</li> </ul>			E
Connection:	📥 LinuxKernelDebug_Linux_Kernel_LS1021AQ	DS_Linux_Kernel_Attach 👻 Ec	lit New
Execute reset sequence	e		
Execute initialization :	cript(s)		
Target	functione target. Prease select a core, or multiple con	es in the case of SMP.	
V LS1021A V Cortex-A7-0 V Cortex-A7-1			
			Apply Revert
		Ι	Debug

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







**NOTE** This example was created on an LS1021ATWR board. For other LS1 boards, the kernel entry point address may differ.

4. Start kernel from the U-Boot console.

Figure 14. U-Boot log - Prepare images for starting Linux kernel

```
## Booting kernel from Legacy Image at 82000000 ...
  Image Name:
               Linux-3.12.0+
                ARM Linux Kernel Image (uncompressed)
  Image Type:
                3053688 Bytes = 2.9 MiB
  Data Size:
  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 ...
```

The breakpoint set above will be hit and CodeWarrior will prompt for the location of the Linux kernel sources to make a path mapping between the original location of the sources and the new location.

For example, in the illustration below, the Linux kernel sources were copied from a Linux machine to a Windows machine.



📧 (AsmSection)() at /sdk/Layerscape1-SDK-20140626-yocto/build_ls1021aqds_release/tmp/work/ls1021aqds-fsl-linux-gnueabi/linux-layerscape-sdk/3.12-r0/git/ 🕱 📃 🗖
Can't find a source file at "/sdk/Layerscape1-SDK-20140626-yocto/build_Is1021aqds_release/tmp/work/Is1021aqds-fsI-linux-gnueabi/linux-layerscape-sdk/3.12- r0/git/arch/arm/kernel/head.S" Locate the file or edit the source lookup path to include its location.
View Disassembly
Locate File
Edit Source Lookup Path
Apply to Common Source Lookup Path

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

After the missing file is located, the actual source file will open in CodeWarrior.

**NOTE** This example was created on an LS1021ATWR board. For other LS1 boards, you may need to instruct the debugger to reload the symbols with the position independent code (PIC) load address.

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

S head.S ⊠	- 8
THUMB( adr r9, BSYM(1f) ) @ Kernel is always entered in ARM.         THUMB( bx r9 ) @ If this is a Thumb-2 kernel,         THUMB(, thumb ) @ switch to Thumb now.	*
<pre>#ifdef CONFIG_ARM_VIRT_EXT blhyp_stub_install #endif</pre>	
@ ensure svc mode and all interrupts masked	
safe_svcmode_maskall r9	
mrc p15, 0, r9, c0, c0 @ get processor id	
bllookup_processor_type @ r5=procinfo r9=cpuid	
movs r10, r5 @ invalid processor (r5=0)?	
HUMB(liteq) (if torce tixup-able long branch encoding	
#ifdef CONFIG_ARM_LPAE	
mrc p15, 0, r3, c0, c1, 4 @ read ID_MMFR0	
and r3, r3, #0xt @ extract VMSA support	-
( muthe + ) in the state of the state of the formation	- F

5. To start kernel debug from *start\_kernel* symbol, set a breakpoint at *start\_kernel*, using Debugger Shell command *bp start\_kernel*.



Figure 17. Set a breakpoint from Debugger Shell at "start\_kernel" method



**NOTE** This example was created on an LS1021ATWR board. For other LS1 boards, you may need to use a different command for starting the kernel.

6. Resume debugging using F8 or the Debugger Shell command go. The breakpoint will be hit and you can perform kernel debugging from *start\_kernel*.



Figure 18. Target stopped at "start\_kernel" method

```
main.c &
    percpu_init_late();
    pgtable_cache_init();
    vmalloc_init();
}
asmlinkage void __init start_kernel(void)

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:
    */
```

At this point, you can perform a full Linux kernel debug using run control (step/run/suspend), set/remove breakpoints, read/write memory/registers/variables, and so on.

### 5.2. Debugging Linux modules

To debug the Linux modules, perform the following steps:

1. Log in to Linux.

### Figure 19. Linux prompt after login



2. Check to see if debugger is already attached to the target. If not, attach it to Linux using the Attach launch configuration.

.







3. Insert a module into Linux.

root@ls1021aqds:~# modprobe isofs

CodeWarrior will automatically detect any *insmod/modprobe/rmmod* operation. A pop-up window appears for locating the module debug symbols.

**NOTE** To detect insertion/removal of kernel modules, CodeWarrior needs to be configured accordingly in the **Debug Configurations** dialog (on **Modules** tab under **Debugger** tab > **OS Awareness** tab).





🥦 Locate symbolics file for isofs									
	git ▶ fs ▶ isofs  v ♦	Search isofs	م						
Organize 🔻 New folde	r	8==	• 🔟 🔞						
🔆 Favorites 🕺	Name	Date modified	Туре						
🧮 Desktop	isofs.ko	7/4/2014 9:47 AM	KO File						
🐌 Downloads									
Recent Places									
🚍 Libraries									
Documents									
🌙 Music 🗉	-								
Pictures									
Videos									
Commuter									
A Primary (C:)									
ge sdk (\\10.171.72.1									
🖵 space (\\zro04file									
🖵 engdata (\\zro04 <sup>.</sup>									
🗣 Network 🔻	< III		۴.						
File na	me: isofs.ko 👻	*.ko	•						
		Open	Cancel						
		Open	Cancer						

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

Figure 22. Target stopped at do\_init\_module after detection that an insmod/modprobe was performed

1	module.c 🛿	- 0
	<pre>blocking_notifier_call_chain(&amp;module_notify_list, MODULE_STATE_COMING, mod);</pre>	*
	<pre>/* Set R0 and NX regions for core */ set_section_ro_nx(mod-&gt;module_core,</pre>	
	/* Set RO and NX regions for init */ set_section_ro_nx(mod->module_init, mod->init_text_size,	
	<pre>mod-&gt;init_ro_size, mod-&gt;init_size);</pre>	
¢	<pre>do_mod_ctors(mod); /* Start the module */ if (mod-&gt;init != NULL)     ret = do_one_initcall(mod-&gt;init); if (ret &lt; 0) {</pre>	
	/* Init routine failed: abort. Trv to protect us from	т F

Debugging Linux Kernel and Modules using CodeWarrior for QorlQ LS series - ARM V7 ISA Application Note



4. Use the **System Browser** view to see the information about kernel version, modules, and threads running on each core.

Figure 23. Kernel modules list displayed in System Browser

👺 Debugger Shell 🕞 Progress 🗐 System Browser 🕱									
\Lambda 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				
Produces Modules									

5. For module debug, open the module's sources in CodeWarrior. Debugging (step, run, or breakpoint) can be done for the inserted modules.

**NOTE** Sometimes when the *remove module* command is executed, CodeWarrior may lose connection to the target. In such a case, ensure that all breakpoints are removed.



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/SalesTernsandConditions</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.

© 2015 Freescale Semiconductor, Inc.

Document Number: AN4980 8 July 2015

