

# Debugging Linux Kernel and Modules using CodeWarrior for QorIQ 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. Introduction.....	1
2. Preliminary background.....	1
3. Creating an ARMv7 project.....	3
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 [www.freescale.com](http://www.freescale.com).

### 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>-<yyyymmdd>-yocto` folder manually.

---

### 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-linux-gnueabi/ linux-layerscape-sdk/3.12-r0/git/*

---

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

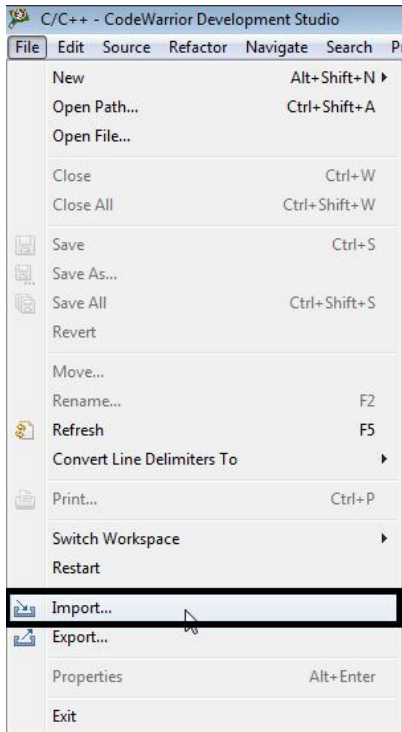
## 3. Creating an ARMv7 project

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 [Configure and rebuild Linux kernel](#).

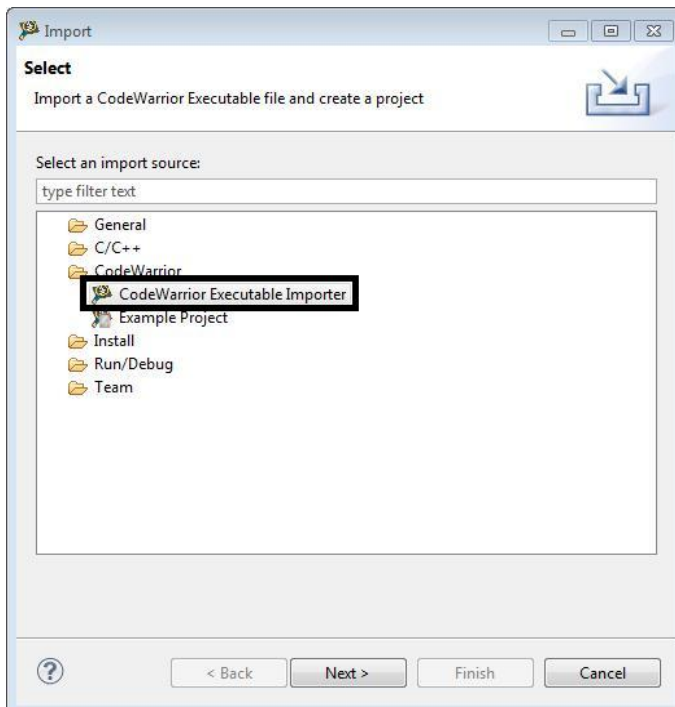
## Creating an ARMv7 project

**Figure 1. CodeWarrior File menu**



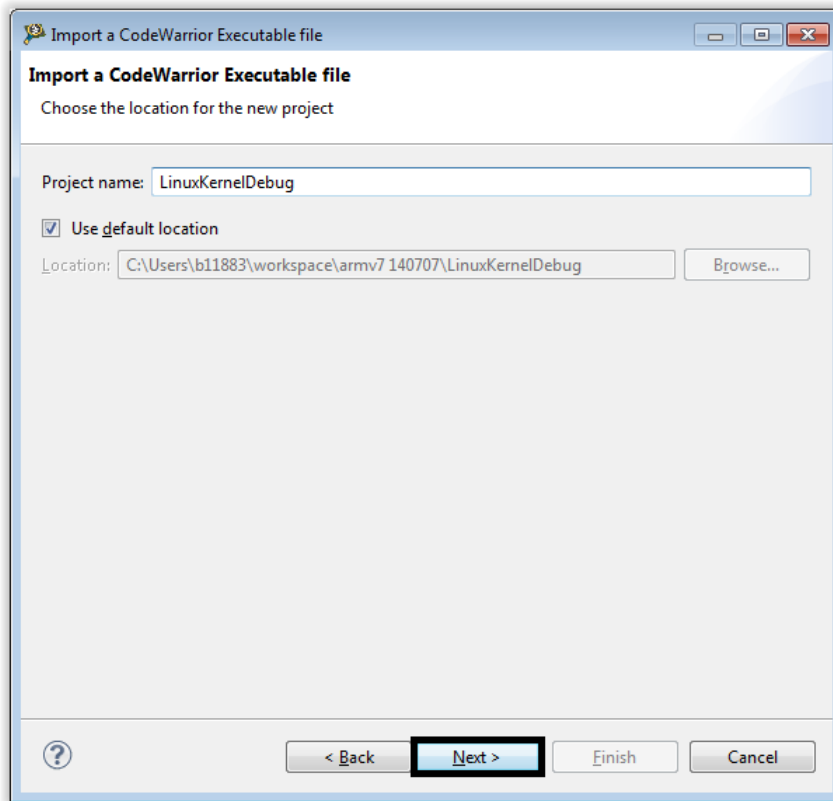
3. On the **Select** page of the **Import** wizard, choose the source to be imported and click **Next**.

**Figure 2. Import wizard**



4. 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**.

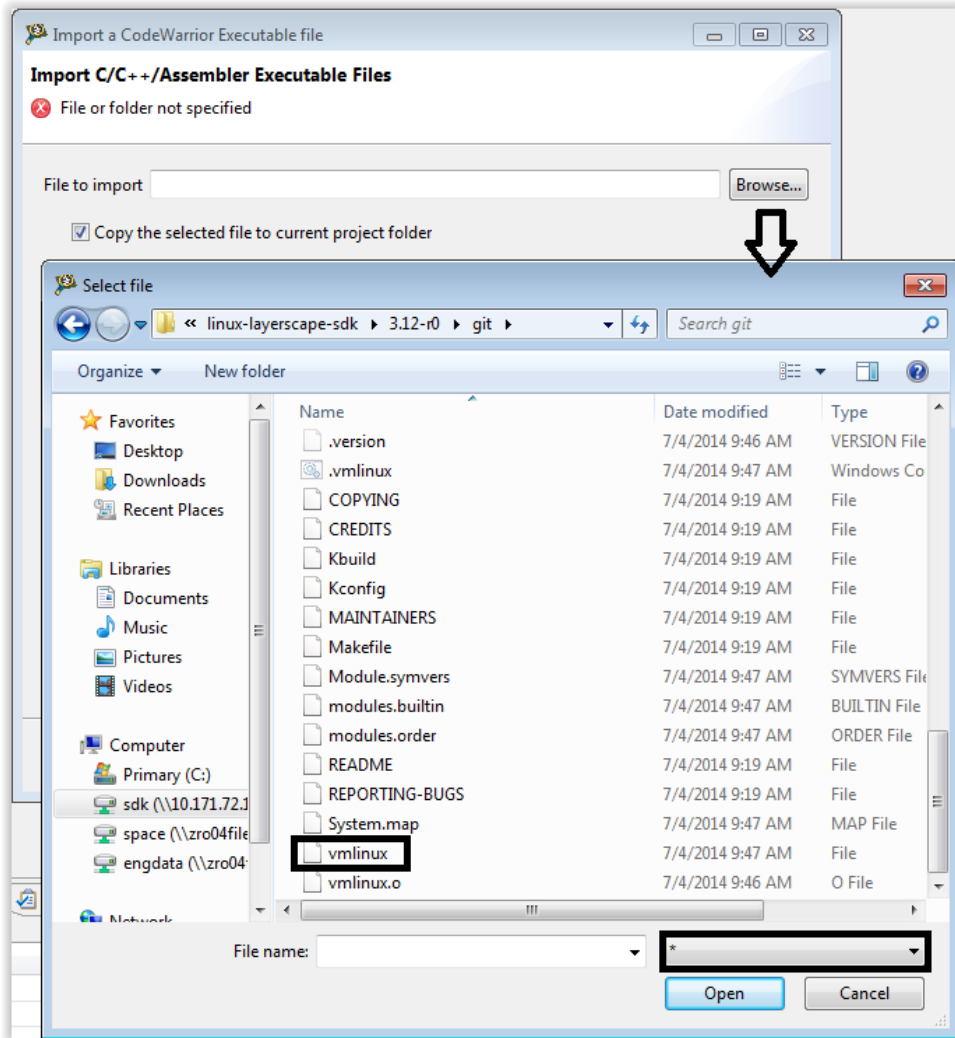
**Figure 3. Import a CodeWarrior Executable file page**



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.

## Creating an ARMv7 project

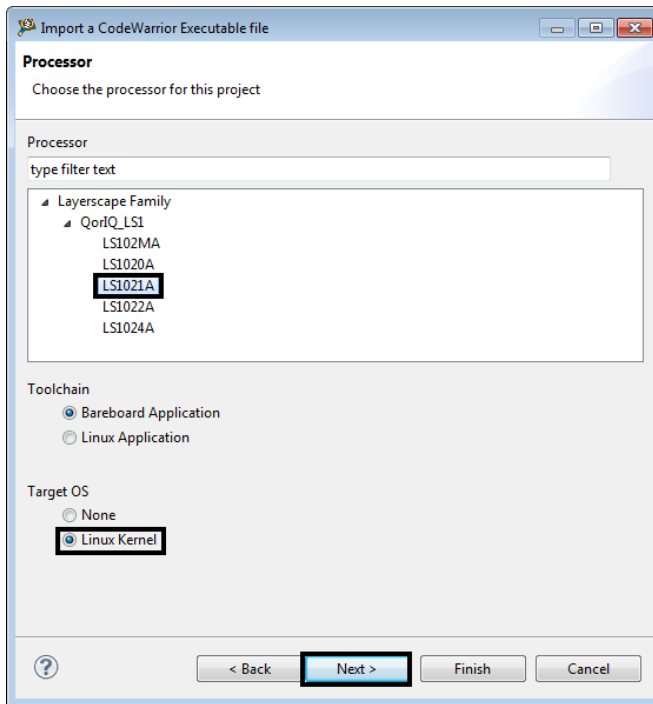
**Figure 4. Select a vmlinux executable file**



9. On the **Processor** page that appears, select the processor type.

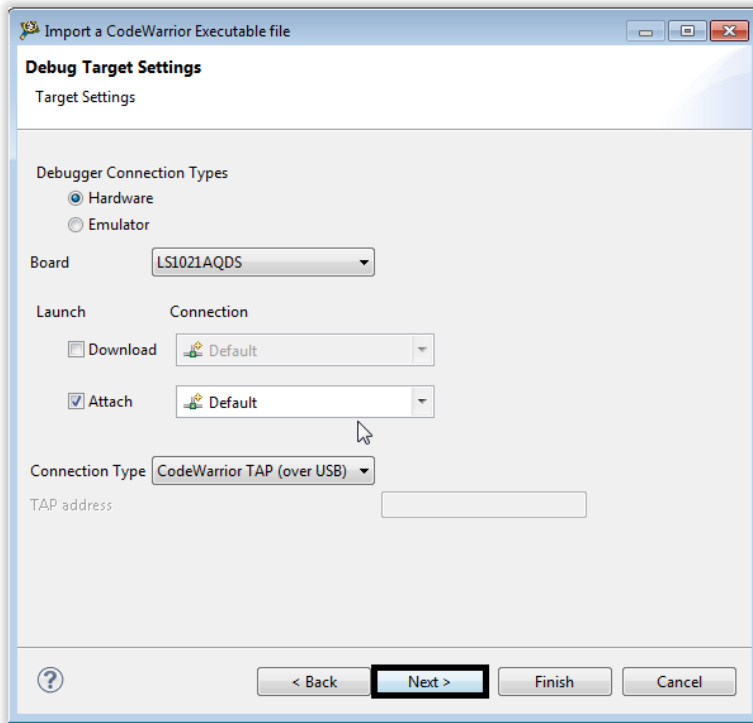
10. Under **Target OS**, choose **Linux Kernel** and click **Next**.

**Figure 5. Processor page**



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

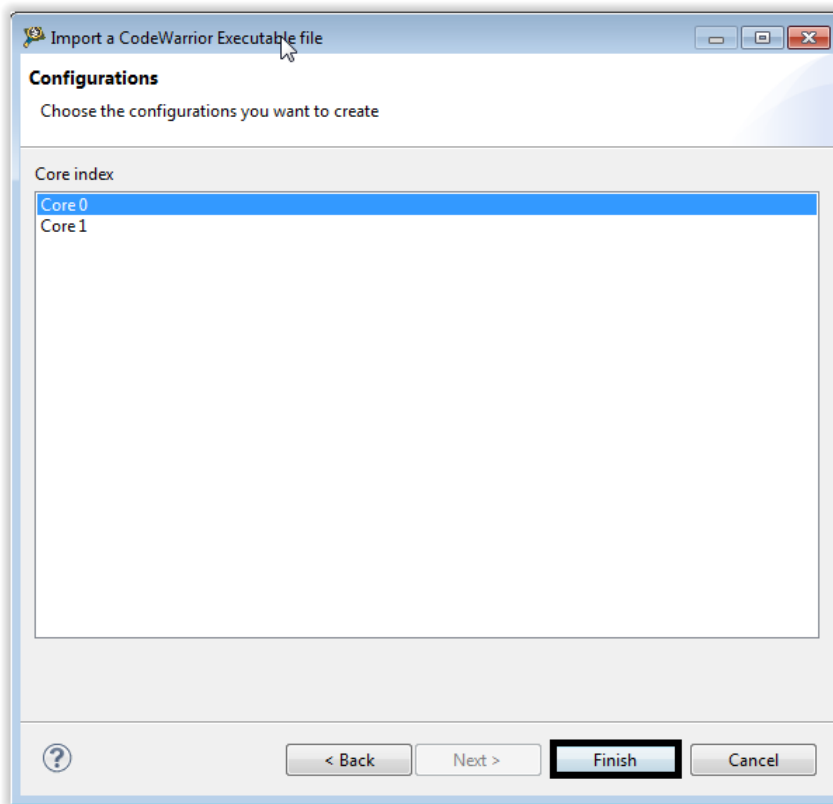
**Figure 6. Debug Target Settings page**



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



## 4. Linux kernel debug support

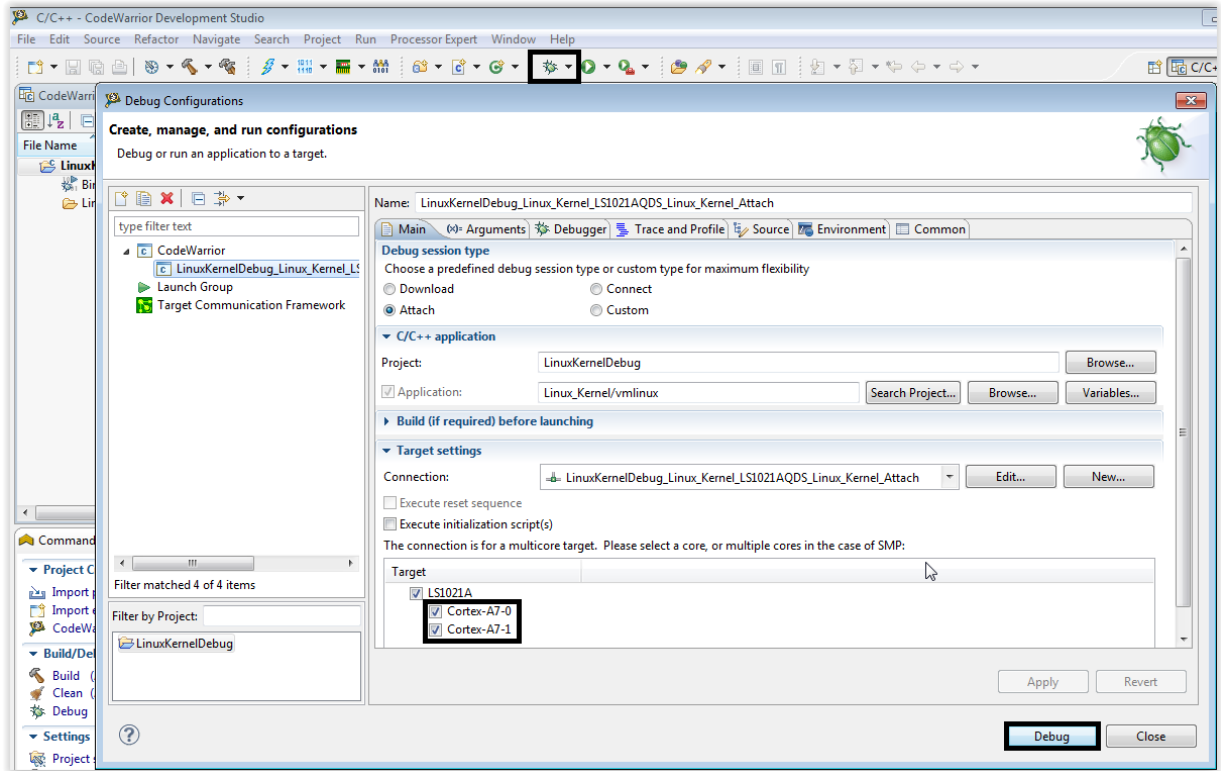
This section describes the debugger settings for Linux kernel debugging.

The `vmlinux` executable file generated during the Linux kernel compilation should be imported as CodeWarrior project (for more information, see [Creating an ARMv7 project](#)).

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**.

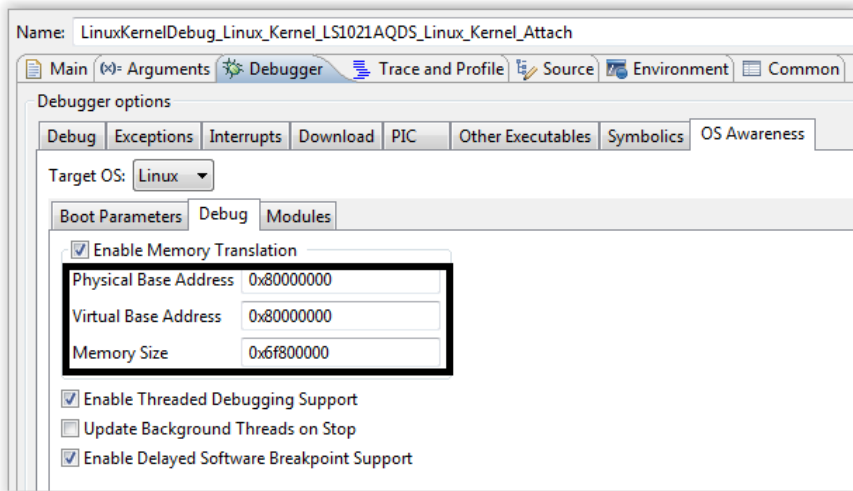
**Figure 8. Debug Configurations dialog**



**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



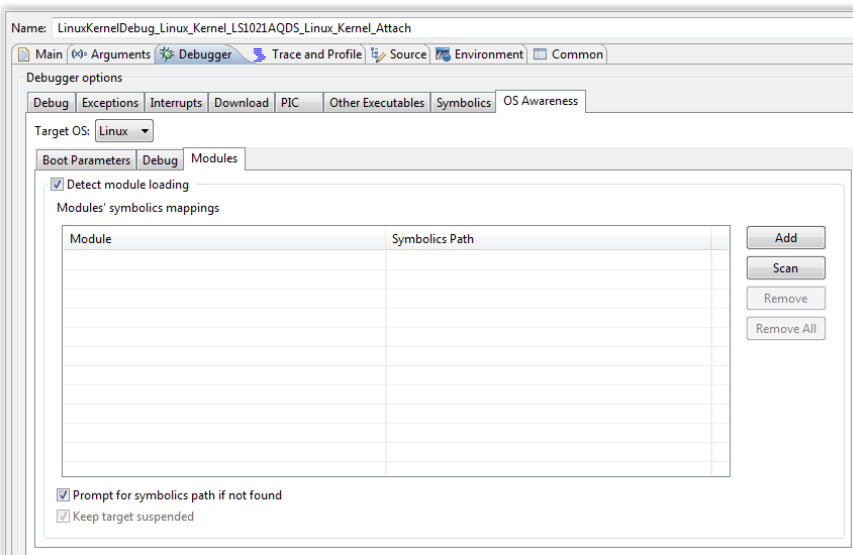
- On the **Modules** tab, select the checkboxes labeled “Detect module loading” and “Prompt for symbolics path if not found”.

---

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

---

Figure 10. OS Awareness – Modules tab



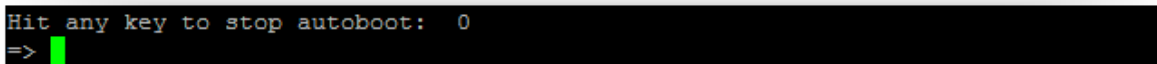
## 5. Debugging Linux kernel and modules

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

### 5.1. Debugging Linux kernel

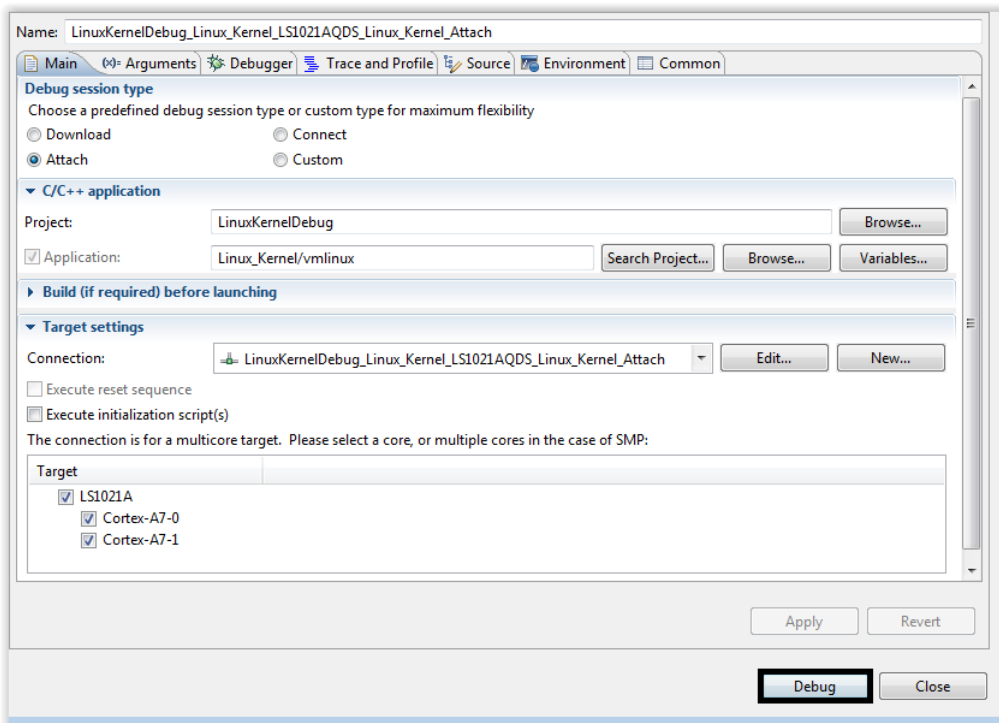
1. Power on the board and stop at the U-Boot console.

**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**



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

**Figure 13. Set breakpoint at entry point in Debugger Shell**

```

Debugger Shell System Browser

%>bp -hw 0x80008000
  id instance      address  type  enabled?  process
description
#36      #1  x:0x80008000  -hw   ENABLED   $0
head.S, line 87, (AsmSection) [vmlinux]
%>
    
```

---

**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**

```

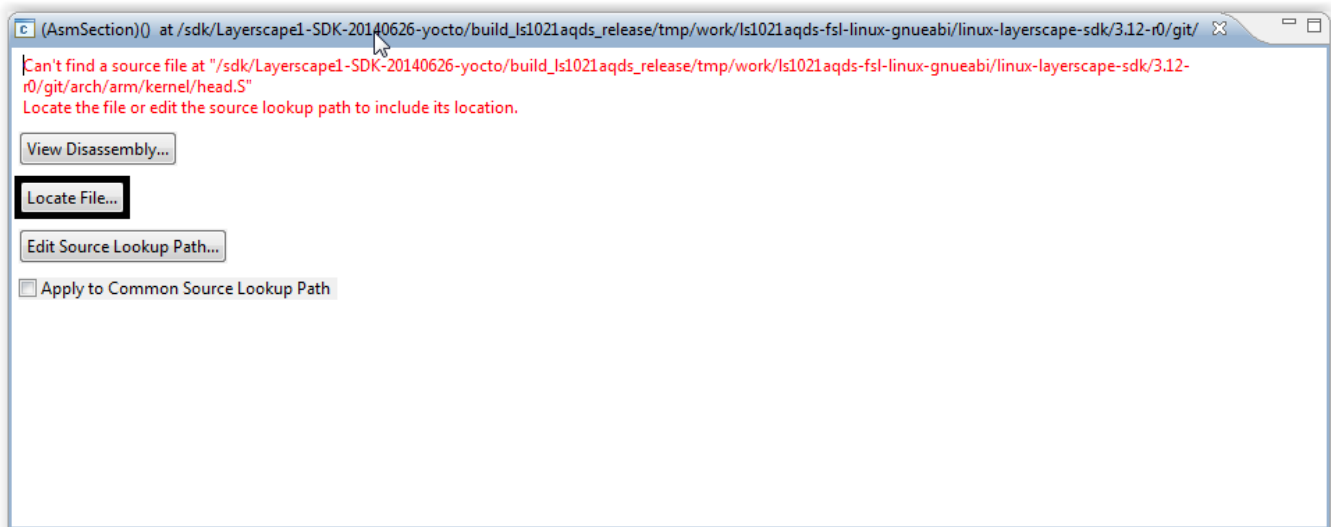
## Booting kernel from Legacy Image at 82000000 ...
Image Name:      Linux-3.12.0+
Image Type:      ARM Linux Kernel Image (uncompressed)
Data Size:       3053688 Bytes = 2.9 MiB
Load Address:    80008000
Entry Point:     80008000
Verifying Checksum ... OK
## Loading init Ramdisk from Legacy Image at 88000000 ...
Image Name:      fsl-image-core-ls1021aqds-201406
Image Type:      ARM Linux RAMDisk Image (gzip compressed)
Data Size:       19170910 Bytes = 18.3 MiB
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.

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



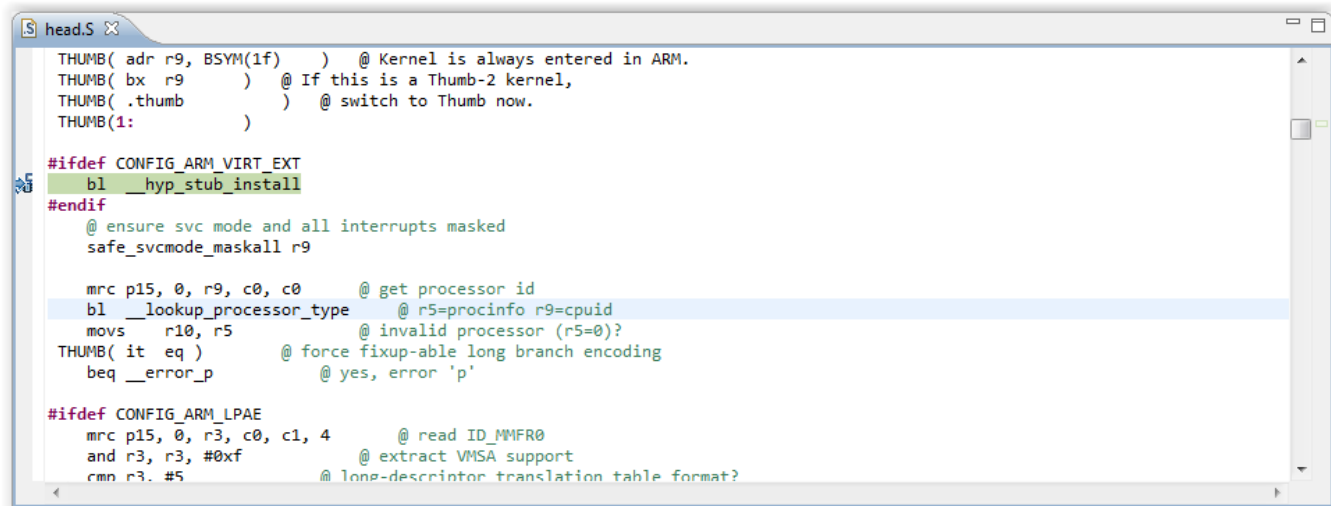
After the missing file is located, the actual source file will open in CodeWarrior.

---

**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**



- 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**

```

CodeWarrior Debugger Shell v1.0
%>bp start_kernel
id instance address type enabled? process description
#4 #1 x:0x8058e82c -auto ENABLED $0 main.c, line 475, start_kernel
[vmlinux]
%>
    
```

---

**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**

```

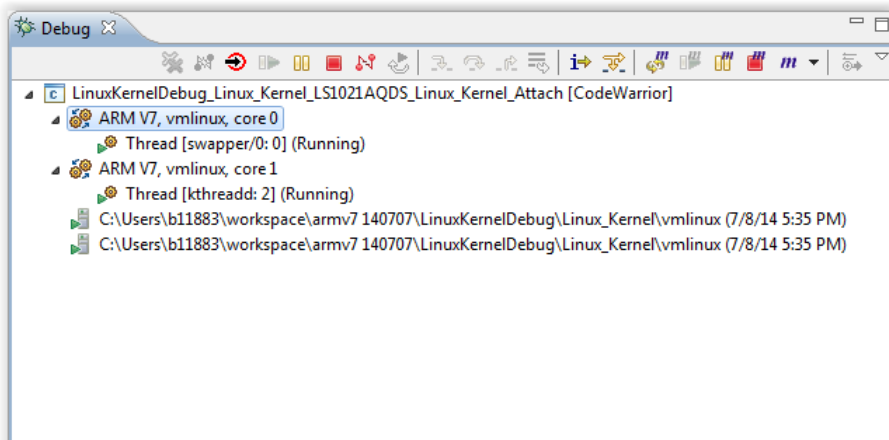
Poky (Yocto Project Reference Distro) 1.5 ls1021aqds /dev/ttyS0

ls1021aqds login: root
root@ls1021aqds:~#
    
```

2. Check to see if debugger is already attached to the target. If not, attach it 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
```

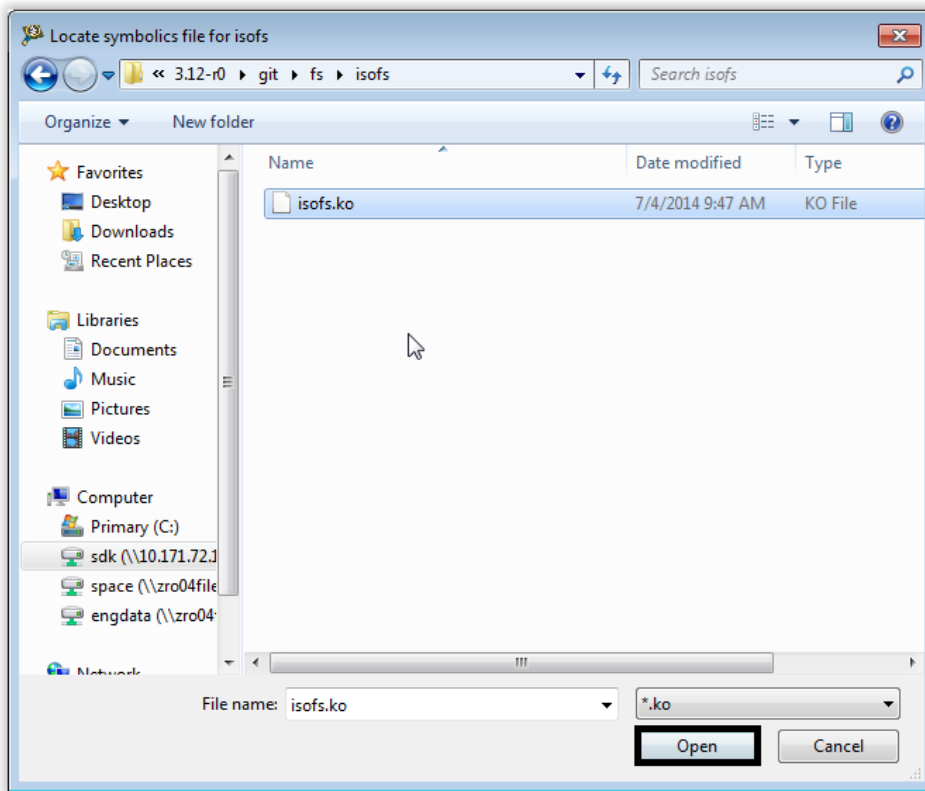
CodeWarrior will automatically detect any *insmod/modprobe/rmmod* operation. A pop-up window appears for locating the module debug symbols.

---

**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).

---

**Figure 21. Locate “isofs” symbolics file**

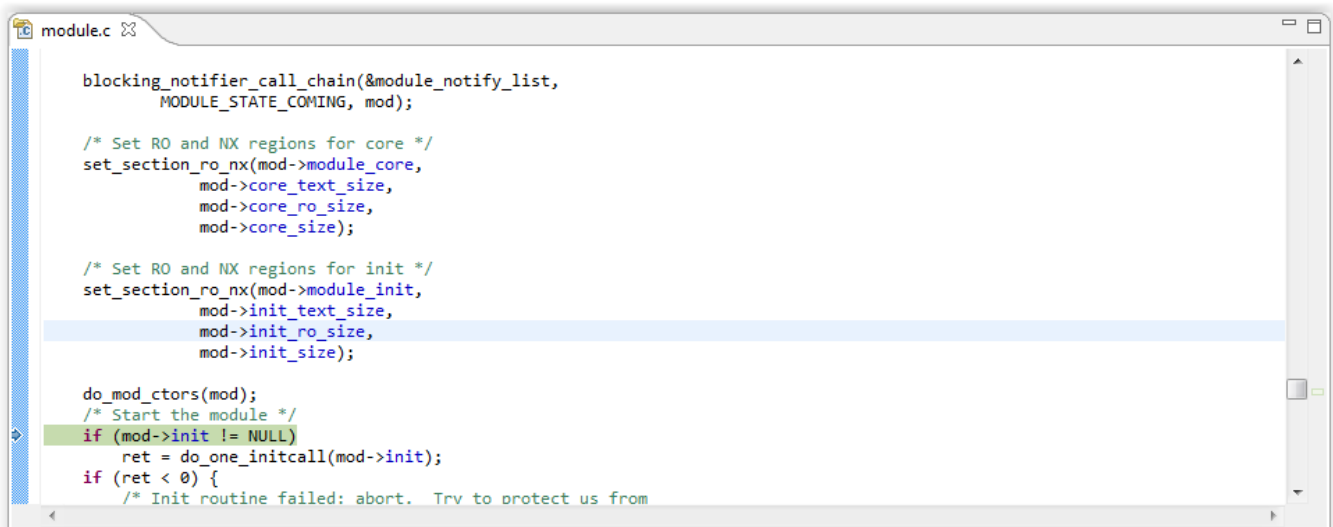



---

**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.

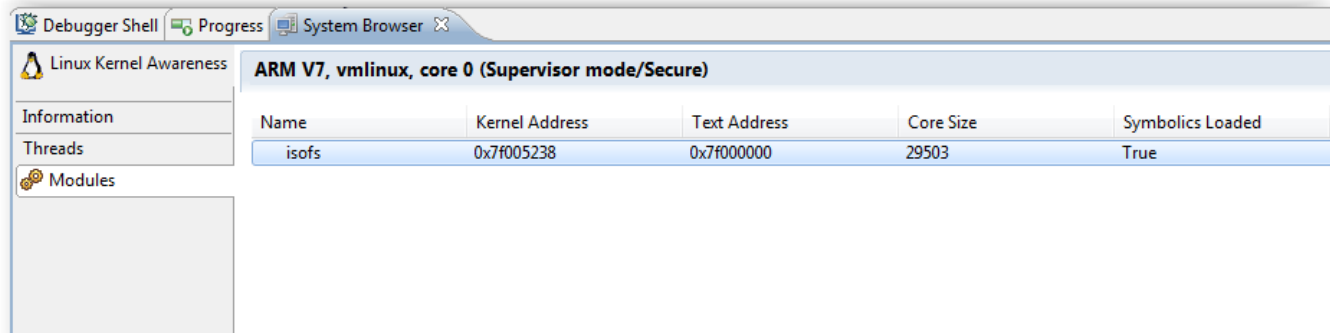
---

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



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**



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](http://www.freescale.com)

**E-mail:**

[support@freescale.com](mailto: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: [freescale.com/SalesTermsandConditions](http://freescale.com/SalesTermsandConditions).

Freescale, the Freescale logo, CodeWarrior, and QorIQ 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.