

U-Boot/Barebox Debug using CodeWarrior for QorIQ LS series - ARM V7 ISA

1. Introduction

This document describes the steps required for U-Boot/Barebox debugging using CodeWarrior Development Studio for QorIQ LS series - ARM V7 ISA.

This document includes the following sections:

- Preliminary background
- Creating an ARMv7 project
- Debugging U-Boot
- Debugging U-Boot SPL
- Debugging Barebox
- Downloading U-Boot/Barebox binary on target board
- Calculating PIC load address for U-Boot DDR relocation

Contents

1. Introduction.....	1
2. Preliminary background.....	1
3. Creating an ARMv7 project.....	2
4. Debugging U-Boot.....	6
5. Debugging U-Boot SPL.....	13
6. Debugging Barebox.....	13
7. Downloading U-Boot/Barebox binary on target board.....	16
8. Calculating PIC load address for U-Boot DDR relocation	18

2. Preliminary background

This section describes the steps required to compile U-Boot/Barebox for the LS1 boards.

Creating an ARMv7 project

2.1. Download SDK

To debug U-Boot/Barebox using CodeWarrior, download the latest SDK for QorIQ from www.freescale.com.

2.2. Compile U-Boot/Barebox

U-Boot/Barebox binary must be built with debug information to be debugged using CodeWarrior.

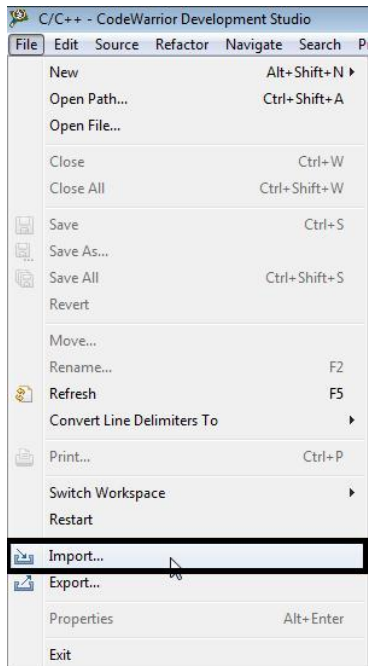
Also, U-Boot/Barebox binary must be downloaded on the target board and must be from the same build as U-Boot/Barebox image. See [Downloading U-Boot/Barebox binary on target board](#) for details on how to download U-Boot/Barebox binary on the target board.

3. Creating an ARMv7 project

To create an ARMv7 bare-metal project for U-Boot/Barebox debug, follow these steps:

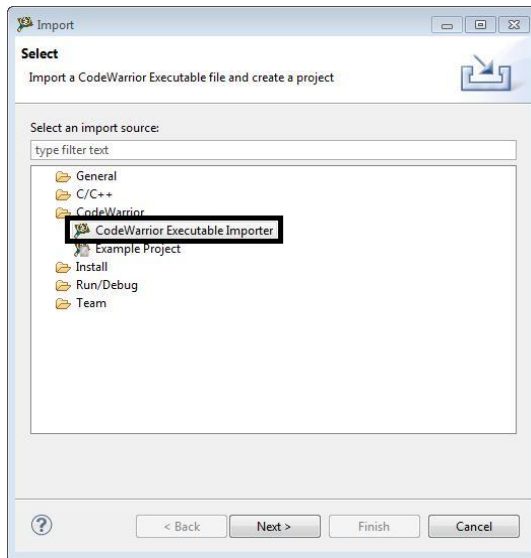
1. Start CodeWarrior for QorIQ LS series - ARM V7 ISA.
2. Choose **File** > **Import** to import the U-Boot/Barebox executable file generated during the U-Boot/Barebox compilation. It can be found in the *U-Boot/Barebox* folder.

Figure 1. CodeWarrior File menu



3. Choose the source to import and click **Next**.

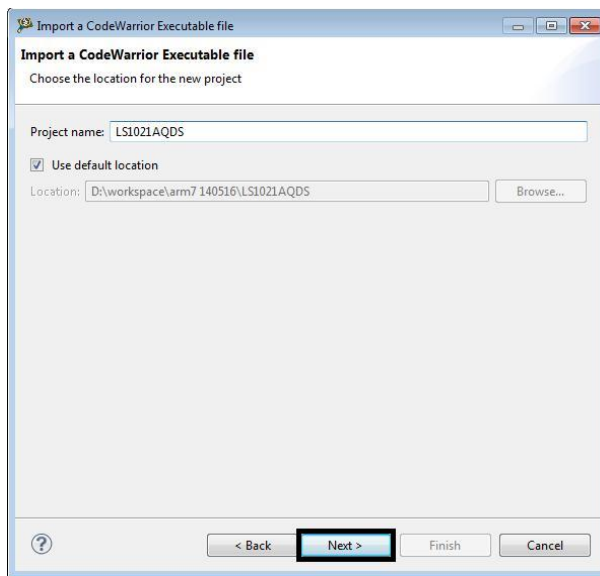
Figure 2. Import dialog



The **Import a CodeWarrior Executable file** wizard starts, as shown in the figure below.

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

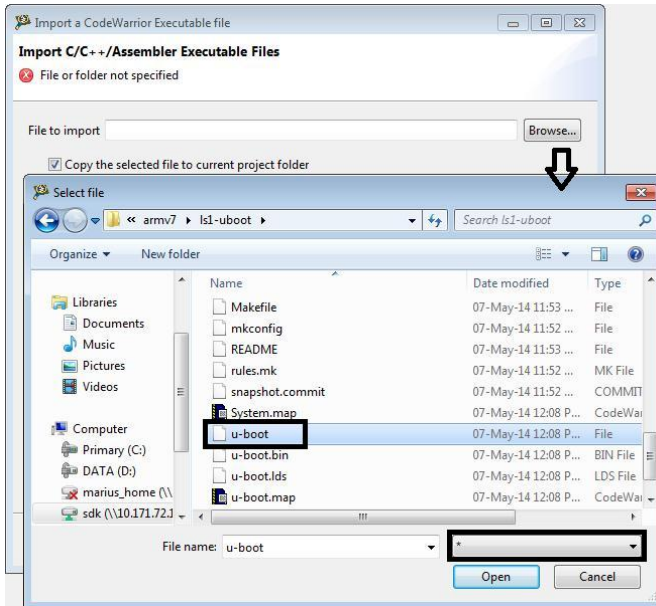
Figure 3. Import a CodeWarrior Executable file page



5. Browse to the U-Boot/Barebox executable file and click **Open**. By default, CodeWarrior looks for an `.elf` extension; therefore, change the file type in the lower-right corner of the **Select file** dialog.

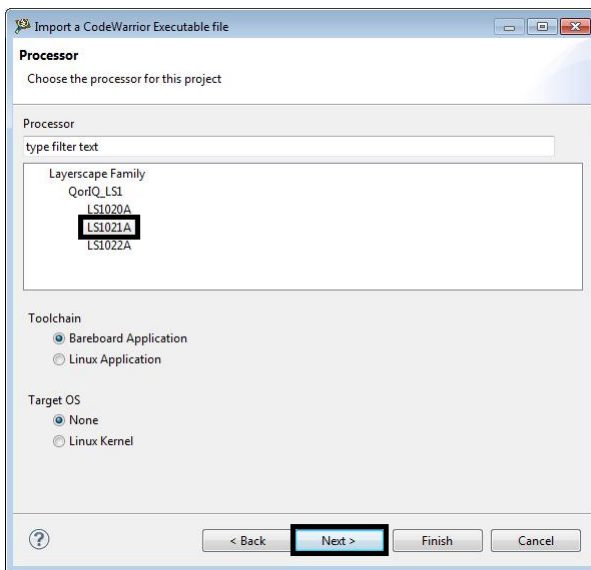
Creating an ARMv7 project

Figure 4. Select U-Boot executable file



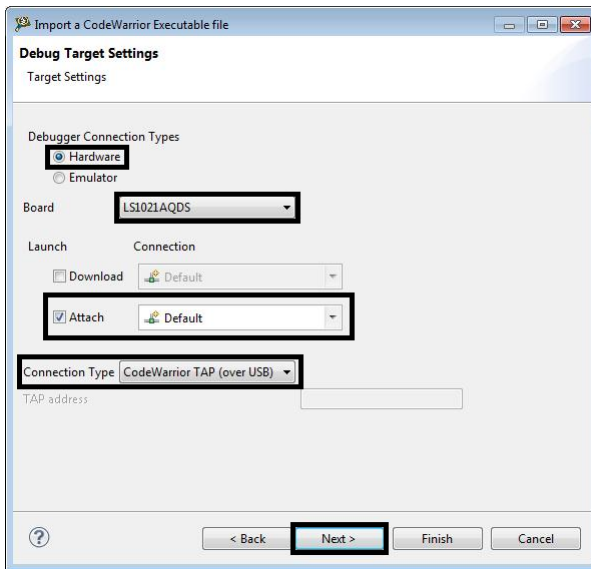
6. Select processor type for the project and click Next.

Figure 5. Processor page



7. Select the debugger connection type, board, launch configuration, and connection type, and click Next.

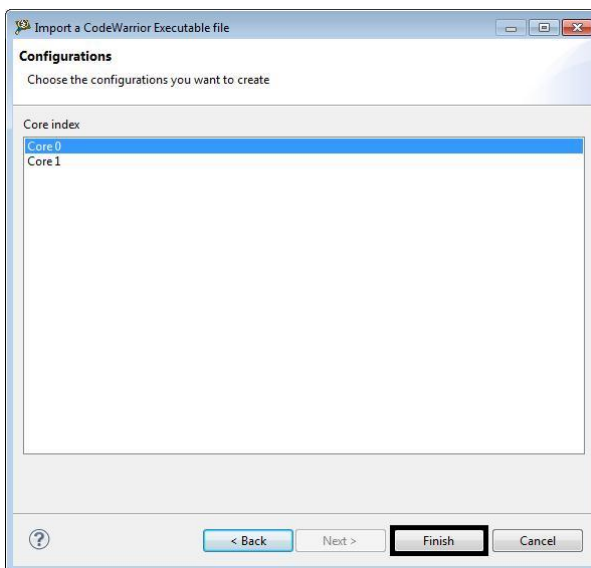
Figure 6. Debug Target Settings page



NOTE By default, U-Boot is generated as a *shared object file*, and not as an *executable file*. In this case, the Download launch configuration does not work; therefore, you need to use the Attach launch configuration. If U-Boot is not available on the target board, then Flash Programmer should be used to program U-Boot on the target board.

8. Choose the configurations you want to create, and then click **Finish** to close the wizard.

Figure 7. Configurations page



4. Debugging U-Boot

When U-Boot starts, it is running from ROM space. However, running from flash would make it almost impossible to read from flash while executing code from flash, not to mention updating the U-Boot image in flash itself. To be able to do that, U-Boot relocates itself into RAM. Because of this, we have two phases with different program addresses. The following sections show how to debug U-Boot in both phases.

4.1. U-Boot debug before relocation

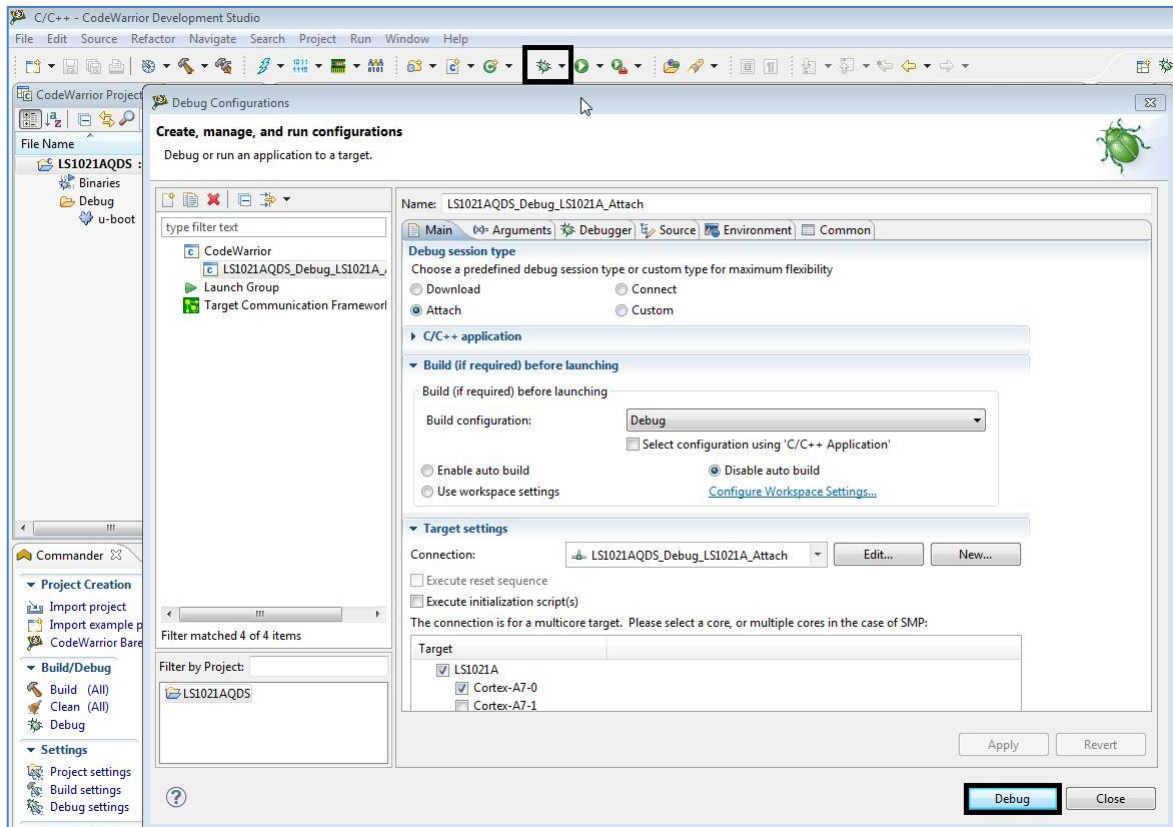
Before U-Boot relocation, the addresses from the ELF file can be used as it is.

The U-Boot executable file generated during the U-Boot compilation should be imported as a CodeWarrior project (for more information, see [Creating an ARMv7 project](#)).

After the CodeWarrior project is created, perform these steps to start U-Boot debug:

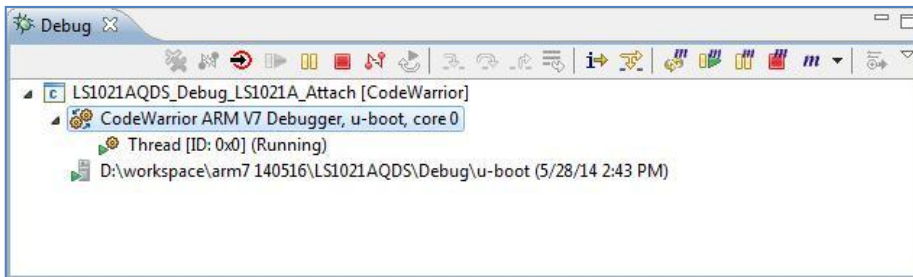
1. Choose **Run > Debug configurations** to open the **Debug Configurations** dialog, and click **Debug** (see figure below).

Figure 8. Debug Configurations dialog



The connection initializes and configures the TAP, and then it will attach to the board (see figure below).

Figure 9. Debug view

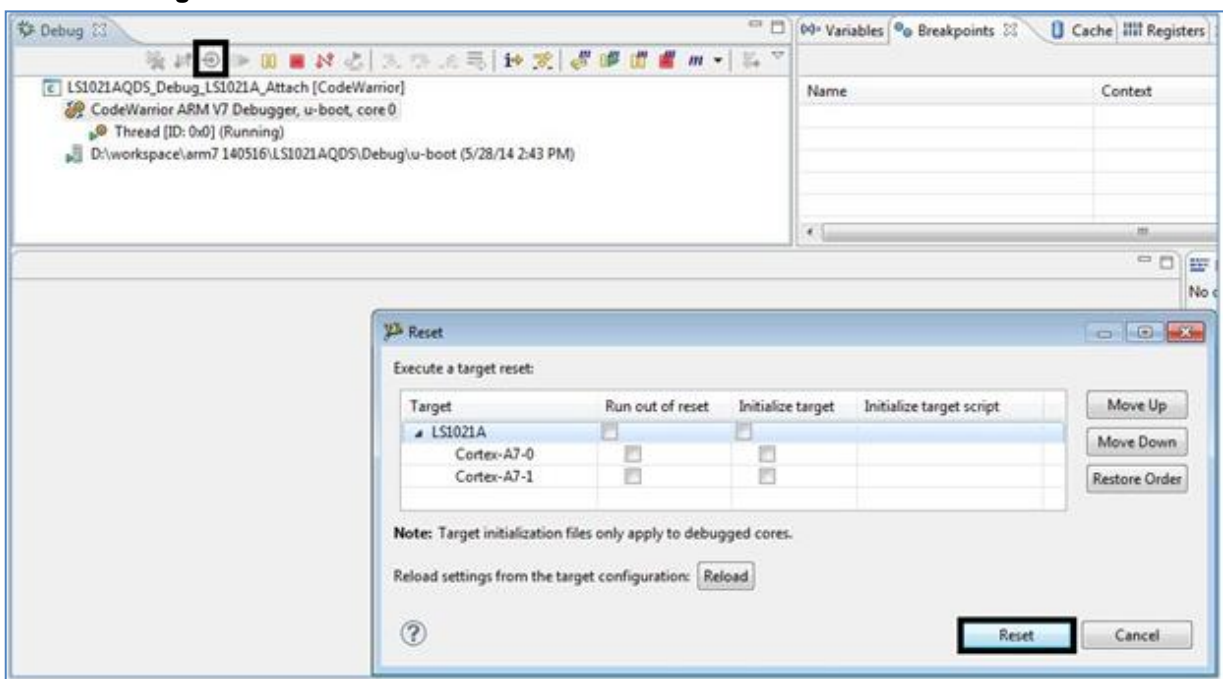


- To reinitialize the target from CodeWarrior, click the **Reset** icon in the **Debug** view toolbar.

The **Reset** dialog opens (see figure below).

- Ensure that no initialization file is selected in the **Reset** dialog and click **Reset**.

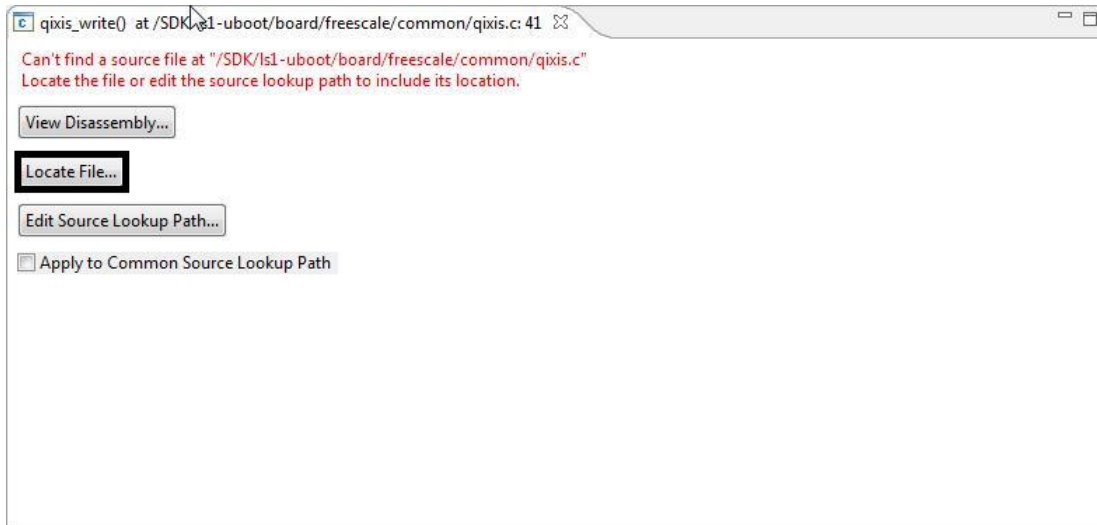
Figure 10. Reset dialog



After reset, debugger will prompt for source location, as shown in the figure below.

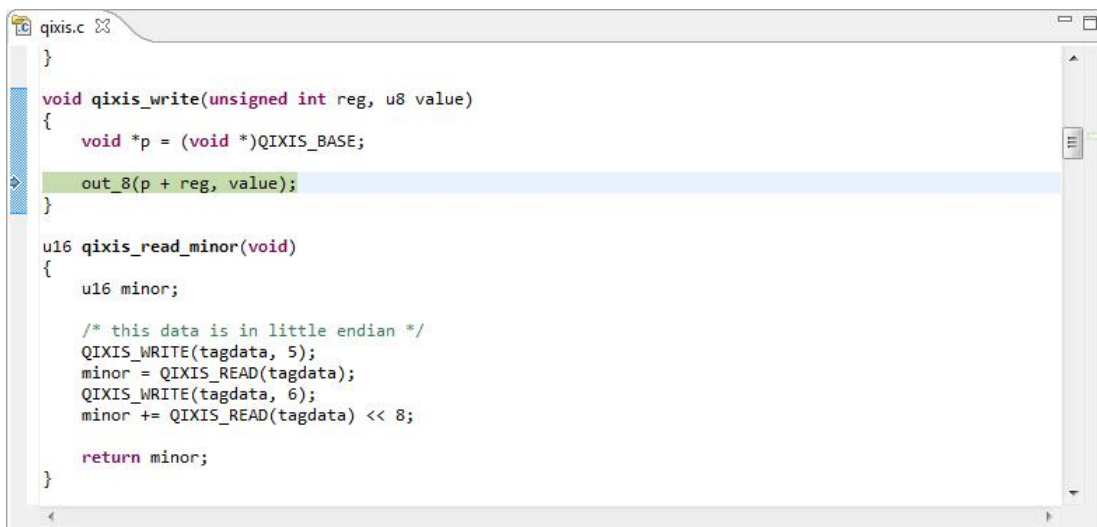
- Click **Locate File** and specify file path.

Figure 11. Specify file location



After the path is provided, source will become available in CodeWarrior.

Figure 12. File editor



- Set a hardware breakpoint at `_start`, using Debugger Shell command `bp -hw _start`.

Figure 13. Set a hardware breakpoint at `_start`

```

CodeWarrior Debugger Shell v1.0
%>bp -hw _start
  id instance      address  type  enabled?  process
description
#3      #1  x:0x67f80000 -hw   ENABLED   $0 start.S,
line 23 [u-boot]
%>
    
```

NOTE Hardware breakpoint must be used before DDR initialization.

- Resume debugging using **F8** or Debugger Shell command `go`.

Figure 14. Perform debugging

```

CodeWarrior Debugger Shell v1.0
%>bp -hw _start
  id instance      address  type  enabled?  process
description
#3      #1  x:0x67f80000 -hw   ENABLED   $0 start.S,
line 23 [u-boot]
%>go
%>
    
```

- Breakpoint will be hit and U-Boot debugging can be performed from `_start`.

Figure 15. File editor

```

start.S
/*
 * armboot - Startup Code for OMAP3530/ARM Cortex CPU-core
 * Copyright (c) 2004 Texas Instruments <r-woodruff2@ti.com>
 * Copyright (c) 2001 Marius Gräßler <mag@sysgo.de>
 * Copyright (c) 2002 Alex ZÄkpkke <azu@sysgo.de>
 * Copyright (c) 2002 Gary Jennejohn <garyj@denx.de>
 * Copyright (c) 2003 Richard Woodruff <r-woodruff2@ti.com>
 * Copyright (c) 2003 Kshitij <kshitij@ti.com>
 * Copyright (c) 2006-2008 Syed Mohammed Khasim <x0khasim@ti.com>
 *
 * SPDX-License-Identifier: GPL-2.0+
 */

#include <asm-offsets.h>
#include <config.h>
#include <version.h>
#include <asm/system.h>
#include <linux/linkage.h>

.globl _start
_start: b reset
    ldr pc, _undefined_instruction
    ldr pc, _software_interrupt
    ldr pc, _prefetch_abort
    ldr pc, _data_abort
    ldr pc, _not_used
    ldr pc, _irq
    ldr pc, _fiq
  
```

The next section describes how to perform U-Boot debug after relocation in RAM.

NOTE If you encounter reset skid issue, the program will not stop at `_start` symbol. As a workaround, you can set a hardware breakpoint at `_start`, and move PC to `_start` symbol address. This issue has been resolved in FPGA v11 image for the LS1 QDS board, but it is present for the LS1 TWR board using CMSIS-DAP probe.

4.2. U-Boot debug after relocation

For U-Boot debugging after relocation, you need to know the address U-Boot relocates itself to. Perform these steps to find out the relocation address:

1. Set a hardware breakpoint at `_main`. Resume debugging using **F8** or Debugger Shell command `go`.
2. Debug until `b relocate_code` (see figure below).

Figure 16. Perform debugging

```

 crt0.S
/*
 * Set up intermediate environment (new sp and gd) and call
 * relocate_code(addr_moni). Trick here is that we'll return
 * 'here' but relocated.
 */

ldr sp, [r9, #GD_START_ADDR_SP] /* sp = gd->start_addr_sp */
bic sp, sp, #7 /* 8-byte alignment for ABI compliance */
ldr r9, [r9, #GD_BD] /* r9 = gd->bd */
sub r9, r9, #GD_SIZE /* new GD is below bd */

adr lr, here
ldr r0, [r9, #GD_RELOC_OFF] /* r0 = gd->reloc_off */
add lr, lr, r0
ldr r0, [r9, #GD_RELOCADDR] /* r0 = gd->relocaddr */
b relocate_code
here:
/*
 * now relocate vectors
 */

```

3. The relocation address is stored in R0 register. Open **Registers** view and read the value for R0 register (see figure below).

Figure 17. Registers view

Name	Value	Location
Core Registers		
R0	0xbff46000	SR0
R1	0x1001ff10	SR1
R2	0x00000000	SR2
R3	0x01ee0200	SR3

For U-Boot debug after relocation, perform these steps:

1. Open the `relocate.S` file. In this file, the last instruction before completion of U-Boot relocation is `bx lr`.

Figure 18. File editor

```

relocate.S
    cmp r2, r3
    blo fixloop

relocate_done:
#ifdef __XSCALE__
    /*
     * On xscale, icache must be invalidated and write buffers drained,
     * even with cache disabled - 4.2.7 of xscale core developer's manual
     */
    mcr p15, 0, r0, c7, c7, 0 /* invalidate icache */
    mcr p15, 0, r0, c7, c10, 4 /* drain write buffer */
#endif

    /* ARMv4- don't know bx lr but the assembler fails to see that */

#ifdef __ARM_ARCH_4__
    mov pc, lr
#else
    bx lr
#endif

ENDPROC(relocate_code)
    
```

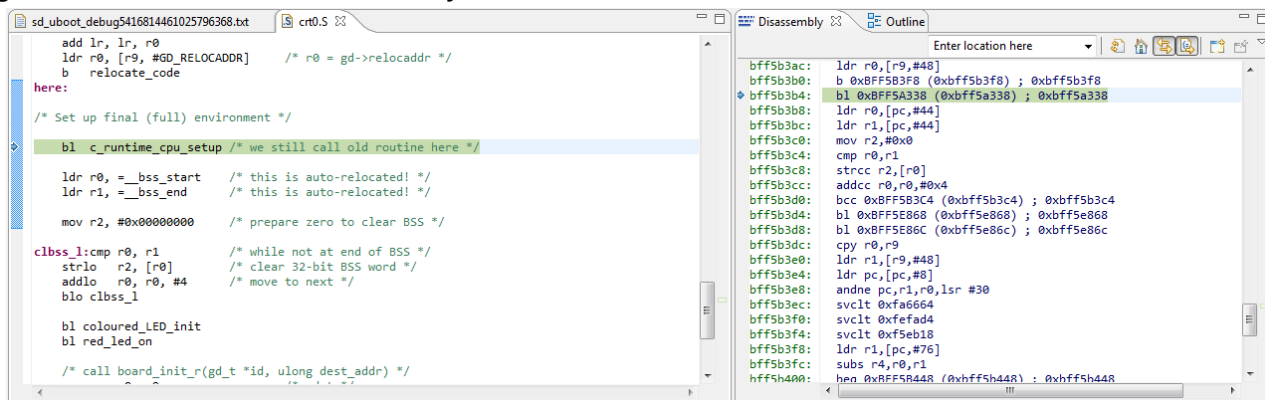
2. Step into `bx lr`. Only disassembly will be available.
3. Instruct the debugger to reload the symbols with position independent code (PIC) load address.
4. Set the PIC load address to `0xBFF46000` using the Debugger Shell command `setpicloadaddr 0xBFF46000`.

Figure 19. Debugger Shell view

```

%>setpicloadaddr 0xbff46000
Debugger now assumes 'u-boot.elf' is loaded at the specified address.
%>
    
```

The symbols are loaded and debugging (step, run, or breakpoint) can be done until the U-Boot boots up (see figure below).

Figure 20. File editor and Disassembly view


NOTE The relocation address can be read from U-Boot prompt using *bdinfo* command. In case the relocation address is not correct when reading R0, then to calculate the PIC load address after U-Boot relocation, see [Calculating PIC load address for U-Boot DDR relocation](#).

5. Debugging U-Boot SPL

For the situations when U-Boot is located in the NAND/SPI/SD card (flash devices that are not memory mapped), first load U-Boot SPL to initialize the hardware, and then load the U-Boot image. The U-Boot SPL executable file generated during U-Boot compilation should be imported as a CodeWarrior project (for more information, see [Creating an ARMv7 project](#)).

After creating the CodeWarrior project, debug U-Boot SPL using the steps provided in [U-Boot debug before relocation](#).

6. Debugging Barebox

Barebox is an alternative bootloader supported by LS1024A target. The Barebox executable file generated during Barebox compilation should be imported as a CodeWarrior project (for more information, see [Creating an ARMv7 project](#)).

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

1. Choose **Run > Debug configurations** to open the **Debug Configurations** dialog, and click **Debug**.
2. To debug Barebox from reset address, reinitialize the target from CodeWarrior by clicking the **Reset** icon in the **Debug** view toolbar.

Debugging Barebox

- Find the reset address of the microloader. To do this, first import the microloader image in CodeWarrior and disassemble it, and then search for `<reset>` in the disassemble file (see figure below).

Figure 21. Find reset address of microloader

```

barebox7524042954909774475.txt
83000074: e1a0f003  mov pc, r3
83000078: 00000034  .word 0x00000034
8300007c: 00009fc0  .word 0x00009fc0
83000080: 83009fc0  .word 0x83009fc0
83000084: 8300b7a4  .word 0x8300b7a4
83000088: 83001b14  .word 0x83001b14

8300008c <reset>:
8300008c: e10f3000  mrs r3, CPSR
83000090: e3c3301f  bic r3, r3, #31
83000094: e38330d3  orr r3, r3, #211 ; 0xd3
83000098: e129f003  msr CPSR_fc, r3
8300009c: eb0012ae  bl 83004b5c <arch_init_lowlevel>
830000a0: eb000007  bl 830000c4 <_mmu_cache_flush>
830000a4: ee113f10  mrc 15, 0, r3, cr1, cr0, {0}
830000a8: e3c33d8e  bic r3, r3, #9088 ; 0x2380
830000ac: e3c33005  bic r3, r3, #5
830000b0: e3833a01  orr r3, r3, #4096 ; 0x1000
830000b4: e3833002  orr r3, r3, #2
830000b8: ee013f10  mcr 15, 0, r3, cr1, cr0, {0}
830000bc: f57ff06f  isb sy
830000c0: eaafffdb  b 83000034 <board_init_lowlevel_return>
  
```

- Set a hardware breakpoint at the reset address of the microloader, as shown in the figure below.

Figure 22. Set a hardware breakpoint at reset address

```

Debugger Shell
%>bp -hw 8300008c
  id instance  address  type  enabled?  process  description
  #34      #1  x:0x8300008c  -hw  ENABLED   0x0     [barebox.elf]
%>go
thread break: Stopped, 0x0, 0x0, cpuARMLittle, barebox.elf (state, tid, pid, cpu, target)
%>
  
```

- Resume debugging using **F8** or Debugger Shell command `go`.

Breakpoint will be hit and only disassembly will be available.

NOTE To have the mapping between the sources and the code, debugger must be instructed to reload the symbols with position independent code (PIC) load address. To calculate the PIC load address, disassemble Barebox executable to obtain the reset address. Calculate the difference between *reset address from microloader executable* and *reset address from Barebox executable*.

- Set the PIC load address to `0x82FFFFFFE8` using the Debugger Shell command `setpicloadaddr 0x82FFFFFFE8`, as shown in the figure below.

Figure 23. Set PIC load address

```

Debugger Shell
%>setpicloadaddr 0x82FFFFE8
Debugger now assumes 'barebox.elf' is loaded at the specified address.
%>
    
```

The symbols are loaded and debugging (step, run, or breakpoint) can be done until the Barebox DDR relocation.

- To continue debugging after DDR relocation, reset the PIC load address using the Debugger Shell command `setpicloadaddress reset`, and set a hardware breakpoint at `start_barebox`, as shown in the figure below.

Figure 24. Set a hardware breakpoint at start_barebox

```

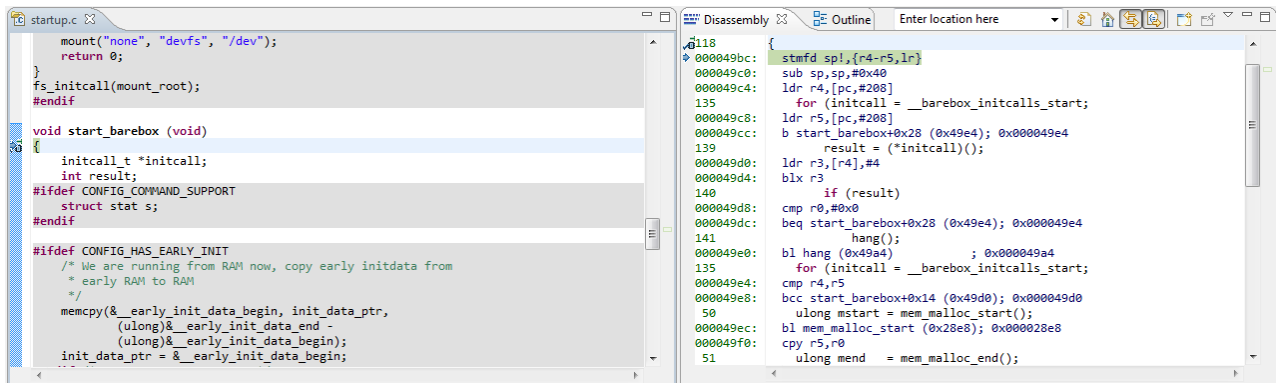
Debugger Shell
%>setpicloadaddr reset
Debugger now assumes 'barebox.elf' is loaded at its link-time address.
%>bp -hw start_barebox
id instance address type enabled? process description
#35 #1 x:0x000049bc -hw ENABLED 0x0 startup.c, line 118,
start_barebox [barebox.elf]
%>
    
```

- Resume debugging using **F8** or Debugger Shell command `go`.

Breakpoint will be hit and debugging (step, run, or breakpoint) can be done until the Barebox boots up.

Downloading U-Boot/Barebox binary on target board

Figure 25. Perform debugging



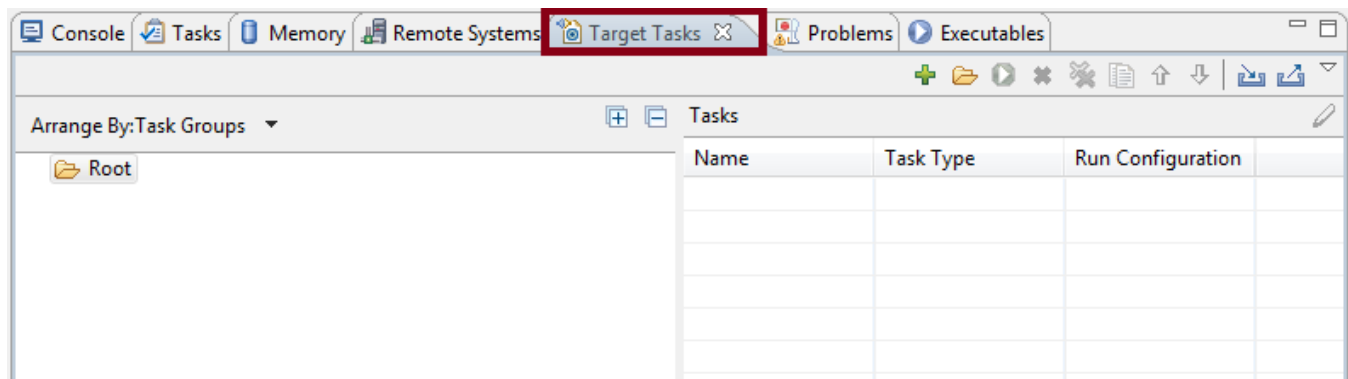
7. Downloading U-Boot/Barebox binary on target board

U-Boot/Barebox binary must be downloaded into the flash device on the target board, and must be from the same build as the U-Boot/Barebox image that is imported as a CodeWarrior project.

Perform these steps to download the U-Boot binary on the target board (the steps to download Barebox binary are similar, only the addresses may differ):

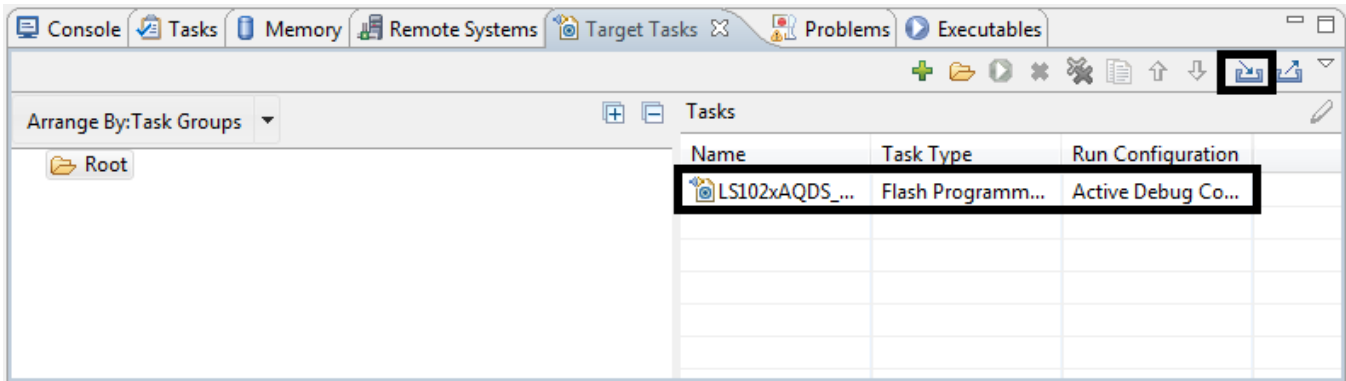
1. With the board in debug, open the **Target Tasks** view.

Figure 26. Target Tasks view



2. Click the **Import** icon, and import the target task (see figure below).

Figure 27. Import target task

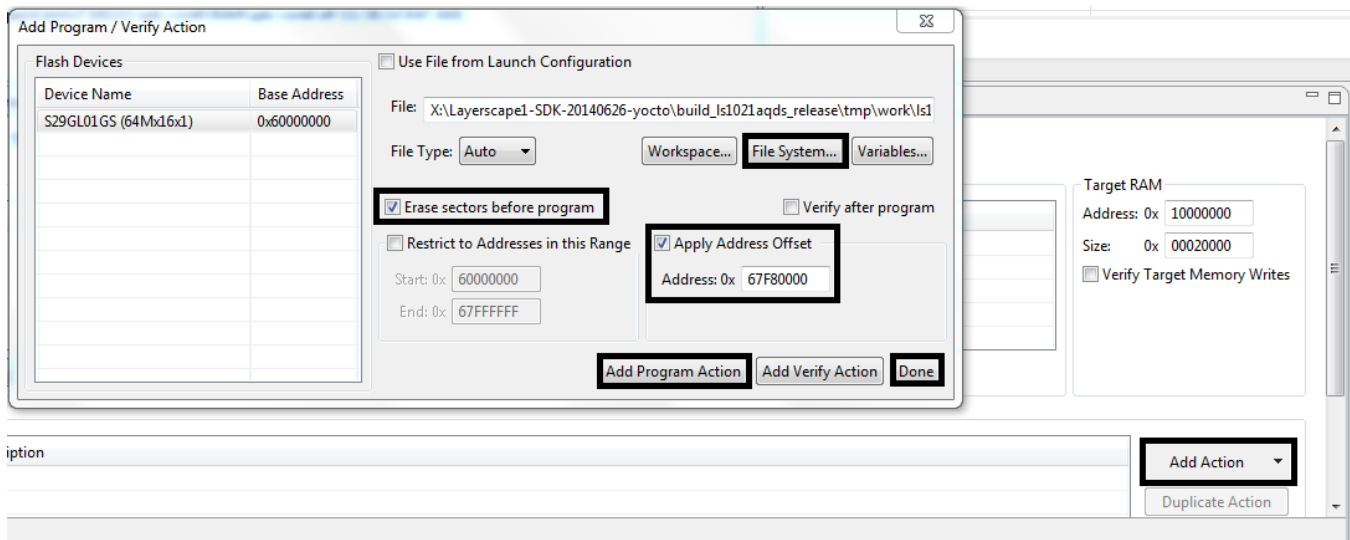


3. Open the imported target task in the **ARM Flash Programmer Task** window.
4. Click the **Add Action** down arrow and choose **Program / Verify**.

The **Add Program / Verify Action** dialog opens.

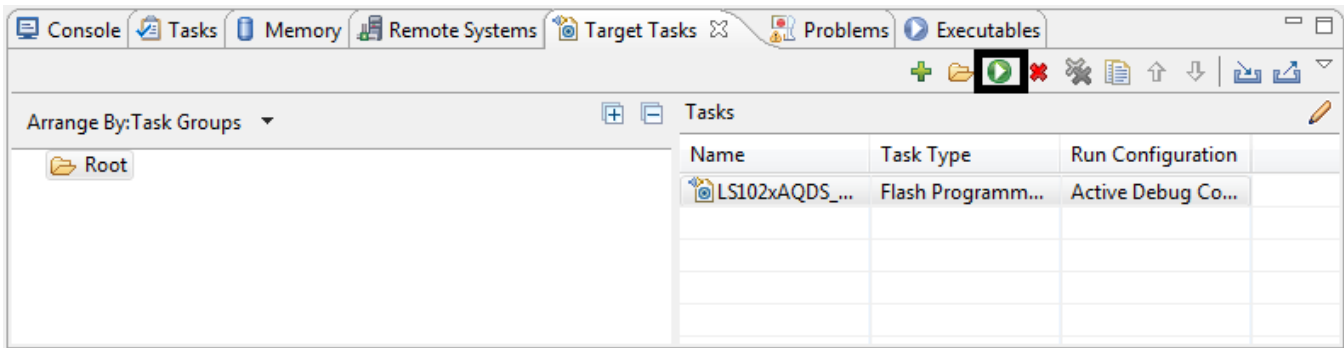
5. Browse for the U-Boot binary. In addition, ensure that the **Erase sectors before program** and **Apply Address Offset** checkboxes are selected, and correct address is specified for the **Apply Address Offset** option.
6. Click **Add Program Action**, and then click **Done**.

Figure 28. Add Program / Verify Action dialog



7. Execute the target task, as shown in the figure below.

Figure 29. Execute target task



8. After the downloading is complete on the target board, the U-Boot binary will be available that is in sync with the U-Boot image.

NOTE The example was done for downloading U-Boot binary to the NOR flash. If the NAND flash is used, ensure that correct target task address is specified for the flash device, specific U-Boot binary is used, and correct offset address is set. For more details about Flash Programmer, see Chapter 11 of *CodeWarrior Development Studio for QorIQ LS series - ARM V7 ISA Targeting Manual*.

8. Calculating PIC load address for U-Boot DDR relocation

To calculate the new PIC load address, after U-Boot relocation, apply this formula:

$$PIC\ address = Runtime\ symbol\ address\ (RAM\ symbol\ address\ in\ our\ case) - Compile\ time\ symbol\ address$$

After step into `bx lr`, in Debugger Shell, perform these operations:

1. Set PIC load address to `0x0`, using Debugger Shell command `setpicloadaddr 0x0`. It tells the debugger that the main executables are loaded at `0x0`.

NOTE This is not the same as `setpicloadaddr reset` command, which tells the debugger that the main executables are loaded at the address set in the ELF file.

2. Set a hardware breakpoint at the function code will jump to. In our case, this is `relocate_vectors`. It shows the compile-time symbol address.

Figure 30. Set a hardware breakpoint

```

Debugger Shell
%>setpicloadaddr 0x0
Debugger now assumes 'u-boot.elf' is loaded at the specified address.
%>bp -hw relocate_vectors
id instance #1 address type enabled? process description
#29 #1 x:0x0000151c -hw ENABLED 0x0 relocate.S, line 31
[u-boot.elf]
%>

```

3. Calculate the difference between the runtime symbol address (single step after `bx lr` instruction, using the address `relocate_vectors` will jump to) and compile-time symbol address:

$$PIC\ address = 0xBFF4751c\ (relocate_vectors\ jump\ address) - 0x0000151c\ (relocate_vectors\ breakpoint\ address) = 0xBFF46000$$

Figure 31. Calculate PIC load address

```

0x8FF474D4 (0x8FF474D4)
67F814c4: e5990044 ldr r0, [r9, #0] ; 0x44
67F814c8: e08ee000 add lr, lr, r0
67F814cc: e5990030 ldr r0, [r9, #40] ; 0x30
67F814d0: ea000014 b 67F81528 <relocate_code>

67F814d4 <here>:
67F814d4: eb000010 bl 67F8151c <relocate_vectors>
67F814d8: ebfff096 bl 67F80338 <<runtime_cpu_setup>
67F814dc: e59f002c ldr r0, [pc, #44] ; 67F81510 <<lbss_l+0x28>
67F814e0: e59f102c ldr r1, [pc, #44] ; 67F81514 <<lbss_l+0x2c>
67F814e4: e3a02000 mov r2, #0

67F814e8 <<lbss_l>:
67F814e8: e1500001 cmp r0, r1
67F814ec: 35802000 strcc r2, [r0]
67F814f0: 32800004 addcc r0, r0, #4
67F814f4: 3afffff0 bcc 67F814e8 <<lbss_l>
67F814f8: eb001170 bl 67F85ac0 <<coloured_LED_init>
67F814fc: eb001170 bl 67F85ac0 <<red_led_on>
67F81500: a1a00000 mov r0, r0

bfff474d4: bl 0xBFF4751C (0xBFF4751C) ; 0xBFF4751C
bfff474d8: bl 0xBFF46338 (0xBFF46338) ; 0xBFF46338
bfff474dc: ldr r0, [pc, #44]
bfff474e0: ldr r1, [pc, #44]
bfff474e4: mov r2, #0x0
bfff474e8: cmp r0, r1
bfff474ec: strcc r2, [r0]
bfff474f0: addcc r0, r0, #0x4
bfff474f4: bcc 0xBFF474E8 (0xBFF474E8) ; 0xBFF474E8
bfff474f8: bl 0xBFF4BAC0 (0xBFF4BAC0) ; 0xBFF4BAC0
bfff474fc: bl 0xBFF4BAC4 (0xBFF4BAC4) ; 0xBFF4BAC4
bfff47500: cpy r0, r9
bfff47504: ldr r1, [r9, #48]
bfff47508: ldr pc, [pc, #0]
bfff4750c: andne pc, r1, r0, lsl pc
bfff47510: svc1t 0xFa020
bfff47514: svc1t 0xFef1ac
bfff47518: svc1t 0xF4bde8
bfff4751c: ldr r0, [r9, #48]

```

5. Set the PIC load address to `0xBFF46000` using the Debugger Shell command `setpicloadaddr 0xBFF46000`.

Figure 32. Set PIC load address

```

Debugger Shell
%>setpicloadaddr 0xBFF46000
Debugger now assumes 'u-boot.elf' is loaded at the specified address.
%>

```

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: 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.