

Freescale Semiconductor Application Note

Document Number: AN4956

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

2. Preliminary background

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

© Freescale Semiconductor, Inc., 2015. All rights reserved.

Contents

1.	Introduction1
2.	Preliminary background1
3.	Creating an ARMv7 project2
4.	Debugging U-Boot6
5.	Debugging U-Boot SPL13
6.	Debugging Barebox13
7.	Downloading U-Boot/Barebox binary on target
	board16
8.	Calculating PIC load address for U-Boot DDR
	relocation18





Creating an ARMv7 project

2.1. Download SDK

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

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 <u>Downloading U-Boot/Barebox binary on target board</u> 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

Select	
Import a CodeWarrior Executable file and create a project	r ² 5
Select an import source:	
type filter text	
 C/C++ CodeWarrior CodeWarrior Executable Importer Example Project Install Run/Debug Team 	

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

mport a CodeWa	the second as a second second second second	
Change the la ti-	rrior Executable file	
Choose the locatio	n for the new project	
Project name: LS	1021AQDS	
Use default loo	ation	
Location: D:\wor	kspace\arm7 140516\LS1021AQDS	Browse
2	< Back Next >	Finish Cancel

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

Import a CodeWarrior Executable file	
Processor	
Choose the processor for this project	
Processor	
type filter text	
Layerscape Family	
QorIQ_LS1	
LS1020A	
LS1021A	
ESTOLER	
Taalshain	
Bareboard Application	
Linux Application	
O Linux Application	
Target OS	
None	
🔘 Linux Kernel	

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



Creating an ARMv7 project

Figure 6. Debug Target Settings page

Debug Target Settings	5		
Target Settings			
Debugger Connection T	ypes		
Hardware			
Board LS1021	LAQDS	•	
Launch Con	nection		
🔄 Download 🛛 💰	Default	-	
🔽 Attach 🛛 🕌	Default	*	
Connection Type CodeV	Varrior TAP (over USB)	•	
TAP address			

- 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

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

U-Boot/Barebox Debug using CodeWarrior for QorIQ LS series - ARM V7 ISA Application Note Freescale Semiconductor



Debugging U-Boot

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 <u>Creating an ARMv7 project</u>).

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

🎾 C/C++ - CodeWarrior	Development Studio			
File Edit Source Ref	actor Navigate Search Project Run W	lindow Help		
CodeWarrior Project	🥦 Debug Configurations		R	8
File Name	Create, manage, and run configuration Debug or run an application to a target.	15		Ť
🐞 Binaries		Name: LS1021AQDS_Debug	LS1021A_Attach	
💝 u-boot	type filter text	Main (x)= Arguments	The Debugger to Source The Environment Common	
	C CodeWarrior C LS1021AQDS_Debug_LS1021A_r Launch Group Target Communication Frameworl	Debug session type Choose a predefined debug Download Attach	session type or custom type for maximum flexibility Connect Custom 	
		C/C++ application		
		Build (if required) before Build (if required) before Build configuration: Enable auto build Use workspace setting	e launching launching Debug Select configuration using 'C/C++ Application' Disable auto build configure Workspace Settings	
۰ III		▼ Target settings		
Commander X Project Creation Import project Monort example r	< >	Connection: Execute reset sequence Execute initialization scri The connection is for a multiple	LSI021AQDS_Debug_LSI021A_Attach Edit New pt(s) ticore target. Please select a core. or multiple cores in the case of SMP:	
🗯 CodeWarrior Bare	Filter matched 4 of 4 items	Target	-	
▼ Build/Debug	Filter by Project:	▼ LS1021A		
≪ Build (All) ∳ Clean (All) ☆ Debug	ES1021AQDS	Cortex-A7-0		
 ✓ Settings Image: Project settings Image: Build settings Image: Debug settings 	3		Apply	Close

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



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





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

The Reset dialog opens (see figure below).

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

Figure 10. Reset dialog

P Debug 🕄			·····	Variables 💁 Breakpoints 🕄 🔪	Cache IIII Regis
CodeWarrior ARM V7 Debugger, CodeWarrior ARM V7 Debugger, Thread (ID: 0x0) (Running) D:\workspace\arm7140516\LS10	Name	Context			
	23 Reset				0.0
	Execute a target reset:	20000000000			
	Target	Run out of reset	Initialize ta	get Initialize target script	Move Up
	a LSI0ZIA	E3.	10		Move Down
	Cortex-A7-1	10	B		Restore Orde
	Note: Target initialization Reload settings from the to	files only apply to debu irget configuration: Re	gged cores. load	Reset	Cancel

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

4. Click Locate File and specify file path.



Debugging U-Boot

Figure 11. Specify file location

🖸 qixis_write() at /SDK 🔊 - uboot/board/freescale/common/qixis.c: 41 🛛	- 8
Can't find a source file at "/SDK/Is1-uboot/board/freescale/common/qixis.c"	
Locate the file or edit the source lookup path to include its location.	
View Disassembly	
Lacata Fila	
Locate rile	
Edit Source Lookup Path	
Apply to Common Source Lookup Path	

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

Figure 12. File editor





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

Figure 13. Set a hardware breakpoint at _start



NOTE Hardware breakpoint must be used before DDR initialization.

6. Resume debugging using F8 or Debugger Shell command go.

Figure 14. Perform debugging



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



Figure 15. File editor



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



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

🗱 Variables 💁 Breakpoints	🚺 Cache 🚻 Registers 🖾	A Modules	
		🌆 📲 📄 🐼 🗕 👬	📫 🍸
Name	Value	Location	-
👬 Core Registers			
1919 RO	0xbff46000	\$R0	
10101 R1	0x1001ff10	\$R1	
1919 R2	0x0000000	\$R2	
1111 R3	0x01ee0200	\$R3	
	III		•
•	III		•

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 1r*.



Debugging U-Boot

Figure 18. File editor



- 2. Step into bx 1r. 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



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

Debugging U-Boot SPL

Figure 20. File editor and Disassembly view

📄 sd_uboot_debug5416814461025796368.txt 🛛 🕼 crt0.S 🛛	- 0	E Disassembly	🛿 📲 Outline			- 0
add lr, lr, r0 ldr r0 [r9 #50 PELOCADDD] /* r0 = gd_>relocaddr */	*			Enter location here	- 🔊	🟠 🔽 🖾 🔁 🖾
h relocate code		bff5b3ac:	ldr r0,[r9,#4	8]		
here:		bff5b3b0:	b ØxBFF5B3F8	(0xbff5b3f8) ; 0xbff	F5b3f8	
		<pre>bff5b3b4:</pre>	bl 0xBFF5A338	(0xbff5a338) ; 0xbf	ff5a338	
/* Set up final (full) environment */		bff5b3b8:	ldr r0,[pc,#44	4]		
		bff5b3bc:	ldr r1,[pc,#44	4]		
bl c runtime cou setup /* we still call old routine here */		bff5b3c0:	mov r2,#0x0			
		bff5b3c4:	cmp r0,r1			
<pre>ldr r0. = bss start /* this is auto-relocated! */</pre>		bff5b3c8:	strcc r2,[r0]			
ldr r1, = bss end /* this is auto-relocated! */		bff5b3cc:	addcc r0,r0,#	0x4		
		bff5b3d0:	bcc 0xBFF5B3C4	4 (0xbff5b3c4) ; 0xb	0ff5b3c4	
mov r2, #0x00000000 /* prepare zero to clear BSS */		bff5b3d4:	b1 0xBFF5E868	(0xbff5e868); 0xbf	rf5e868	
		bff5b3d8:	DI ØXBFF5E86C	(0xbff5e86c); 0xbf	115e86c	
<pre>clbss_l:cmp r0, r1 /* while not at end of BSS */</pre>		bff5b3dc:	cpy r0,r9	-1		
<pre>strlo r2, [r0] /* clear 32-bit BSS word */</pre>		bff5b3e0:	1dr r1,[r9,#4	o]		
addlo r0, r0, #4 /* move to next */		bffsb3e4:	Iar pc,[pc,#8]		
blo clbss_1		DTT5D3e8:	andne pc,ri,ri	0,15r #30		
	E	bffsb3f0	svcit Øxtabbb	4		
bl coloured_LED_init		bffEb2f4	svelt Oxferau	*		-
bl red_led_on		bffchofe,	Ido of foc #7	0 61		
		bffEb2fc	subs p4 p0 p1	0]		
/* call board_init_r(gd_t *id, ulong dest_addr) */	-	hffsh400	hea AvREESBAA	8 (Avhff5h448) · Avh	off5b448	-
	F		< [•

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 <u>Calculating PIC load address for U-Boot</u> <u>DDR relocation</u>.

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 <u>Creating an ARMv7 project</u>).

After creating the CodeWarrior project, debug U-Boot SPL using the steps provided in <u>U-Boot debug</u> <u>before relocation</u>.

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 <u>Creating an ARMv7 project</u>).

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

3. 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

D	barebox752404	2954909774475	i.bd 🕱	
	83000074: 83000078: 8300007c: 83000080: 83000084: 83000088:	e1a0f003 00000034 00009fc0 83009fc0 8300b7a4 83001b14	mov pc, r3 .word 0x00000034 .word 0x83009fc0 .word 0x83009fc0 .word 0x83009fc0 .word 0x8300b7a4 .word 0x83001b14	•
	8300008c <r 8300008c: 83000090:</r 	reset>: e10†3000 e3c3301f	mrs r3, CPSR bic r3, r3, #31	
	83000094: 83000098: 8300009c:	e38330d3 e129f003 eb0012ae	orr r3, r3, #211 ; 0xd3 msr CPSR fc, r3 bl 83004b5c (arch_init_lowlevel> bl 83004b5c (arch_init_lowlevel>	
	830000a4: 830000a8: 830000a8:	ee113f10 e3c33d8e e3c33005	DI 03000064 <mmul_cnl_cnl_lusn> mrc 15, 0, r3, cnl_cr0, {0} bic r3, r3, #9088 ; 0x2380 bic r3, r3, #5</mmul_cnl_cnl_lusn>	
	830000b0: 830000b4: 830000b8:	e3833a01 e3833002 ee013f10	orr r3, r3, #4096 ; 0x1000 orr r3, r3, #2 mcr 15, 0, r3, c1, cr0, {0}	
	830000bc: 830000c0:	f57ff06f eaffffdb	isb sy b 83000034 <board_init_lowlevel_return></board_init_lowlevel_return>	

4. 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 🛛		8	è		
%>bn -bw 8300008c					
id instance address type enabled? process description					
#34 #1 x:0x8300008c -hw ENABLED 0x0 [barebox.el	F]				
%>go					
thread break: Stopped, 0x0, 0x0, cpuARMLittle, barebox.elf (state, ti	1, p:	10,	срі	, ا	
s s s s s s s s s s s s s s s s s s s					

5. 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*.
- 5. Set the PIC load address to 0x82FFFFE8 using the Debugger Shell command setpicloadaddr 0x82FFFFE8, as shown in the figure below.

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



Figure 23. Set PIC load address



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

6. 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



7. 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

🔞 startup.c 🛛	- 0	🗉 🔤 Disassembly 🕴 🗄 Outline 🛛 Enter location here 💿 🔹 👔 🕵 🔯 🖆 🖄 🖆
<pre>mount("none", "devfs", "/dev"); return 0; } fs_initcall(mount_root); #endif</pre>	*	.di18 { > 000049bc: stmfd spl,{r4-r5,ln} 000049c4: ldr r4,[pc,#200] 135 for (inital] = harehox initalls start:
<pre>void start_barebox (void) %0 { initcall_t *initcall; int result; if result; #ifdef CONFIG CONFIG CONFIG</pre>		000049c8: ldr r5,[pc,#208] 000049c8: bstart_barebox+0x28 (0x49e4); 0x000049e4 139 result = (*initcall)(); 000049d0: ldr r3,[r4],#4 000049d4: blx r3 140 if (result)
struct stat s; #endif #ifdef CONFIG_HAS_EARLY_INIT		000049d8: cmp r0,#0x0 000049d6: beq start_barebox+0x28 (0x49e4); 0x000049e4 141 hang(); 000049e0: bl hang (0x49a4) ; 0x000049a4
<pre>/* We are running from RAM now, copy early initdata from * early RAM to RAM */ memcpy(&_early_init_data_begin, init_data_ptr, (ulong)&_early_init_data_end - (ulong)&_early_init_data_begin); init_data_ptr = & early_init_data_begin;</pre>	4	<pre>135 for (initcall =barebox_initcalls_start; 0000494e4: cmp r4,r5 0000494e8: bcc start_barebox+0x14 (0x49d0); 0x000049d0 0 ulong mstart = mem_mallo_start(); 0000494ec: bl mem_malloc_start((0x26e5); 0x000028e8 000049f0: cpy r5,r0 51 ulone mend = mem_malloc_end(): 7</pre>
<pre></pre>	•	

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.

Tigule 20. Talget Tasks view								
🔄 Console 🤕 Tasks 🚺 Memory 🚚 Remote Systems	🔞 Target Tas	iks 🛛	🕄 Problems	Exec	utables			
				+ 🗁	0 8	🔆 🗈	分 步	2
Arrange By:Task Groups 🔻	E E	Tasks						
🔁 Root		Name	Ta	sk Type		Run Cor	nfiguratio	on
								_

Figure 26. Target Tasks view

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

പ



Downloading U-Boot/Barebox binary on target board

📮 Console ⁄ Tasks 🚺	Memory 🚚 Rei	mote Systems 🖺	🙆 Target Tas	ks 🖾 🛛 🔝 Probler	ms 🚺 Executables		
					🕂 🗁 🗘 🗶	🔆 🗈 ት 🕂	<u>b</u> 2 ~
Arrange By:Task Groups	-		E E	Tasks			0
Root				Name	Task Type	Run Configuration	
				` LS102xAQDS	Flash Programm	Active Debug Co	

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

ash Devices	Use File from Launch Configuration	
Device Name Base Address	File: VAL summer of CDK 20140626 us to build 1/1021 and release build 1/1	
i29GL01GS (64Mx16x1) 0x6000000	File Type: Auto Workspace File System Variables Image: Type: Auto Workspace File System Variables Image: Type: Auto Image: Type:	Target RAM Address: 0x Size: 0x O0020000 Verify Target Memory Writes

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



Calculating PIC load address for U-Boot DDR relocation

Figure 29. Execute target task

🔄 Console 🖉 Tasks 🚺 Memory 📕 Remote Systems 🔞 Target	Tasks 🛛 🖳 Proble	ms 🜔 Executables		- 8
		+ 🕞 🚺 🗱	🐐 🗈 🕆 🖓 🖻	•⊿ [~]
Arrange By:Task Groups 🔻 🖽 🗓	Tasks			0
Proof	Name	Task Type	Run Configuration	
	` 🚵 LS102xAQDS	Flash Programm	Active Debug Co	

- 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* 1r, 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.



Calculating PIC load address for U-Boot DDR relocation

Figure 30. Set a hardware breakpoint



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

C 0xBFF474D4 (0xBFF474D4)()	- 8	📟 Disassembly 🛛 📴 Outline 🛛 Enter location here 💿 🔹 👔 🗐 🕼 😭 🖄 😭 🖄 👘
67f814c4: e5930044 ldr r0, [r9, #68] ; 0x44 67f814c4: e888e000 add lr, 1r, r0 67f814cc: e5930030 ldr r0, [r9, #48] ; 0x30 67f814d0: ea000014 b 67f81528 <relocate_code> 67f814d4 (barax)</relocate_code>	•	Φ bff474d4: bl 0x8FF4751C (0x6Ff4751C) : 0xbff4751C bff47d8: bl 0x8FF4751C (0x6Ff4751C) : 0xbff4751C bff47d8: bl 0x6Ff4751C (0x6Ff4751C) : 0xbff4751C bff47d4C: ldr r0,[pc,#44] bff474e4: ddr r1,[pc,#44] bff474e4: mov r2,#0x0
67f81444: eb000010 bl 67f8151c <relocate_vectors> 67f81448: ebfft996 bl 67f8158 <c_runtime_cpu_setup> 67f81446: e59f002c ldr r0, [pc, #44] ; 67f81510 <closs_l+0x28> 67f814e0: e59f102c ldr r1, [pc, #44] ; 67f81514 <closs_l+0x2c> 67f814e4: e3a02000 mov r2, #0</closs_l+0x2c></closs_l+0x28></c_runtime_cpu_setup></relocate_vectors>		D1474e0: LBP 19,11 Df474f0: strcc r2,[r0] bf474f6: bcc %bf474f8 (%bff474e8); %bbff474e8 bff474f6: bl %xBF48BcG (%bbff4bac0); %xbff47ae8 bf474f6: bl %xBF48BcG (%bbff4bac0); %xbff4bac4 bff47500: cpy r0,r9
67f314e8 <clbss_1>: 67f314e8: e1500001 cmp r0, r1 67f314ec: 35802000 strcc r2, [r0] 67f314f3: 32800004 addcc r0, r0, #4 67f314f4: 3affffb bcc 67f314e8 <clbss_1> 67f314f4: eb001170 b1 67f35ac4 <red_led_on> 67f314fc: eb001170 b1 67f35ac4 <red_led_on></red_led_on></red_led_on></clbss_1></clbss_1>	Ŧ	bff47584: ldr r.1, [r9,#48] bff47580: idr pc, pc, r0, #8] bff47580: solt pc, r1, r9, lel pc bff47514: solt bff47514: bff47514: solt bff478 bff47512: 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



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

Freescale Semiconductor



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.

© 2015 Freescale Semiconductor, Inc.

Document Number: AN4956 7 July 2015

