Loading Boot Loader on LS1046ARDB through PCIe

1 Introduction

This document details the implementation of a use case where PCIe root complex (RC) provides boot images to the LS1046A configured as a PCIe endpoint (EP). The figure below shows the setup diagram for this use case.

![Use case setup diagram](image)

**Figure 1. Use case setup diagram**

To provide access to CCSR registers and memory space, the endpoint has to configure BARs (inbound windows) using PBI commands. Similarly, the root complex needs to configure its outbound windows. With this, the root complex can configure endpoint’s DDR controller registers and copy boot loader directly into endpoint’s DDR.

Using PBI commands, OCRAM of endpoint is programmed with minimal firmware to configure CSU registers. The firmware waits until boot loader is loaded on endpoint. The root complex will set a flag indicating the completion of copy. On detection of the flag, the endpoint comes out of polling loop and jumps to DDR for loading U-Boot.

### Contents

1. Introduction........................................1
2. Use case setup....................................2
3. Configuring LS1046ARDB as PCIe root complex..........................2
4. Configuring LS1046ARDB as PCIe endpoint........................................9
5. Appendix: Using C29XPCIE-RDB as PCIe root complex..................22
6. Appendix: Hardware and software resources.............................24
7. Appendix: Related documentation..........................25
8. Revision history.......................................26
For this use case, two LS1046ARDBs are connected back to back using a PCIe extended cable. One LS1046ARDB is configured as a root complex and the other LS1046ARDB is configured as an endpoint.

This document describes:
- Steps to configure the LS1046ARDB as a PCIe root complex
- Steps to configure the LS1046ARDB as a PCIe endpoint
- Procedure for transferring boot images from root complex to endpoint and booting endpoint

## 2 Use case setup

To implement the use case, two LS1046ARDBs are connected to each other using a PCIe extended cable. One of the two boards is configured as root complex and the other as endpoint. The figure below shows the use case setup.

![Use case setup](image)

**Figure 2. Use case setup**

RCW and PBI binary must be programmed on the endpoint as explained in Configuring LS1046ARDB as PCIe endpoint before connecting endpoint to root complex.

## 3 Configuring LS1046ARDB as PCIe root complex

This section describes how to configure an LS1046ARDB as a PCIe root complex and how to load boot loader on another LS1046ARDB configured as a PCIe endpoint.

The root complex needs to complete the following functions:
- Perform link training with endpoint
- Program two outbound windows for memory transaction
- Initialize endpoint’s DDR controller
- Copy RAMboot image to endpoint’s DDR
- Trigger boot process of endpoint
3.1 Board switch settings

The table below shows DIP switch settings of the LS1046ARDB where QSPI is the boot source.

**Table 1. DIP switch settings**

<table>
<thead>
<tr>
<th>DIP switch</th>
<th>Settings</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW3[1:8]</td>
<td>01000110</td>
<td>• 0 indicates OFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 indicates ON</td>
</tr>
<tr>
<td>SW4[1:8]</td>
<td>00111011</td>
<td></td>
</tr>
<tr>
<td>SW5[1:8]</td>
<td>00100010</td>
<td></td>
</tr>
</tbody>
</table>

3.2 RCW fields for PCIe configuration

The table below details the RCW fields to configure the LS1046ARDB as root complex.

**Table 2. POR parameters for PCIe controller (root complex)**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCW[HOST_AGT_PEX]</td>
<td>Selects between Root Complex (RC) and Endpoint (EP) modes</td>
<td>3'b000: All Host mode</td>
</tr>
<tr>
<td>RCW[SRS_PRTCL_Sn]</td>
<td>Determines the link width</td>
<td>0x1133: SRDS_PRTCL_S1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x5506: SRDS_PRTCL_S2</td>
</tr>
<tr>
<td>RCW[SRS_DIV_PEX_Sn]</td>
<td>Determines the link speed</td>
<td>2'b00: SRDS_DIV_PEX_S1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2'b00: SRDS_DIV_PEX_S2</td>
</tr>
</tbody>
</table>

3.3 Program outbound windows

The table below describes the base addresses for LS1046A’s PCIe controllers.

**Table 3. PCIe controller base address**

<table>
<thead>
<tr>
<th>PCIe controller</th>
<th>Base address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIe controller 1</td>
<td>340_0000h</td>
</tr>
<tr>
<td>PCIe controller 2</td>
<td>350_0000h</td>
</tr>
<tr>
<td>PCIe controller 3</td>
<td>360_0000h</td>
</tr>
</tbody>
</table>

The root complex needs to access CCSRBAR register space and DDR SDRAM memory space of the endpoint. This can be done by programming root complex's outbound windows. In the current example, two outbound windows are used, which can be defined as follows:

- Outbound window 2: Represents a memory window of 512 MB from 0x48_1000_0000 to 0x48_2FFF_FFFF, to be translated to 0x00000000 for accessing endpoint's CCSRBAR and OCRAM memory space
- Outbound window 3: Represents a memory window of 512 MB from 0x48_4000_0000 to 0x48_5FFF_FFFF, to be translated to 0x80000000 for accessing endpoint's DDR SDRAM memory space
Program outbound window 2

Perform these steps to program outbound window 2:

1. Set up Index register:
   a. Write 0x00000002 to address {0x700 + 0x200} to set outbound window 2 as the current window.
2. Set up Region Base and Limit Address registers:
   a. Write 0x10000000 to address {0x700 + 0x20C} to set the lower base address.
   b. Write 0x00000048 to address {0x700 + 0x210} to set the upper base address.
   c. Write 0x2fffffff to address {0x700 + 0x214} to set the limit address.
3. Set up Target Address registers:
   a. Write 0x00000000 to address {0x700 + 0x218} to set the lower target address.
   b. Write 0x00000000 to address {0x700 + 0x21C} to set the upper target address.
4. Configure the window through Region Control 1 register:
   a. Write 0x00000000 to address {0x700 + 0x204} to define the type of the window to be memory space.
5. Enable the window:
   a. Write 0x80000000 to address {0x700 + 0x208} to enable the window.

Program outbound window 3

Perform these steps to program outbound window 3:

1. Set up Index register:
   a. Write 0x00000003 to address {0x700 + 0x200} to set outbound window 3 as the current window.
2. Set up Region Base and Limit Address registers:
   a. Write 0x40000000 to address {0x700 + 0x20C} to set the lower base address.
   b. Write 0x00000048 to address {0x700 + 0x210} to set the upper base address.
   c. Write 0x5fffffff to address {0x700 + 0x214} to set the limit address.
3. Set up Target Address registers:
   a. Write 0x80000000 to address {0x700 + 0x218} to set the lower target address.
   b. Write 0x00000000 to address {0x700 + 0x21C} to set the upper target address.
4. Configure the window through Region Control 1 register:
   a. Write 0x00000000 to address {0x700 + 0x204} to define the type of the window to be memory space.
5. Enable the window:
   a. Write 0x80000000 to address {0x700 + 0x208} to enable the window.
3.4 Initialize endpoint's DDR

The root complex initializes endpoint’s DDR by configuring DDR controller registers in CCSR space. DDR configuration for the LS1046ARDB is provided in a target initialization file in CodeWarrior for ARMv8. Below is the path to the LS1046ARDB target initialization file within CodeWarrior installation directory:

`<CWInstallDir>\Freescale\CW4NET_v2017.03\CW_ARMv8\Config\boards\LS1046A_RDB_init.py`

QCVS tool can also be used to extract data for initializing DDR controller registers.

Perform below steps to initialize DDR of PCIe endpoint from PCIe root complex. All commands are run on the U-Boot prompt of the root complex (LS1046ARDB).

1. Program outbound window 2 to access CCSR space and OCRAM of PCIe endpoint:

   ```
   => mw.l 3500900 2
   => mw.l 3500904 0
   => mw.l 350090c 10000000
   => mw.l 3500910 00000048
   => mw.l 3500914 2fffffff
   => mw.l 3500918 00000000
   => mw.l 350091c 0
   => mw.l 3500908 80000000
   ```

   Programmed registers are shown below.

   ![Outbound window 2 register dump](image)

2. Configure DDR controller registers of PCIe endpoint:

   ```
   => mw.l 4811080000 ff010000
   => mw.l 4811080080 22030480
   => mw.l 48110800c0 00000000
   ```

   ![Outbound window 2 register dump](image)

   NOTE

   Take endianess of DDR registers into consideration while configuring these registers.

   ```
   => mw.l 4811080000 ff010000
   => mw.l 4811080080 22030480
   => mw.l 48110800c0 00000000
   ```
3. Program outbound window 3 to access DDR region of PCIe endpoint:

```assembly
=> mw.1 3500900 3
=> mw.1 3500904 0
=> mw.1 350090c 40000000
=> mw.1 3500910 00000048
=> mw.1 3500914 5ffffff
=> mw.1 3500918 80000000
=> mw.1 350091c 0
=> mw.1 3500908 80000000
```

Programmed registers are shown below.
4. Read endpoint’s DDR from root complex.

5. Use below commands to load RAMboot image on endpoint’s DDR memory at 0x82000000. See the "Binary Files of RAM boot" section of AN12081 for details on RAMboot image. On the root complex, the RAMboot image is read from a USB stick.

```bash
=> usb start
=> fatload usb 0 4842000000 <file_name>
```

The figure below shows the output of the above commands.

6. Trigger the boot-up process of PCIe endpoint by writing a non-zero value at 0x10010000 location of OCRAM:

```bash
=> mw.1 4820010000 1
```

7. Open endpoint (LS1046ARDB) serial port terminal to see U-Boot log, as shown in the figure below.
Figure 9. U-Boot log on endpoint terminal
4 Configuring LS1046ARDB as PCIe endpoint

This section describes how to configure the LS1046ARDB as PCIe endpoint using CodeWarrior.

4.1 Board switch settings

For board switch settings, see Board switch settings.

4.2 RCW fields for PCIe configuration

The table below details the RCW fields to configure the LS1046ARDB as endpoint.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCW[HOST_AGT_PEX]</td>
<td>Selects between Root Complex (RC) and Endpoint (EP) modes</td>
<td>3'b001: All Agent mode</td>
</tr>
<tr>
<td>RCW[SRDS_PRTCL_Sn]</td>
<td>Determines the link width</td>
<td>0x1133: SRDS_PRTCL_S1 0x5506: SRDS_PRTCL_S2</td>
</tr>
<tr>
<td>RCW[SRDS_DIV_PEX_Sn]</td>
<td>Determines the link speed</td>
<td>2'b00: SRDS_DIV_PEX_S1 2'b00: SRDS_DIV_PEX_S2</td>
</tr>
</tbody>
</table>

4.3 Minimal firmware on endpoint's OCRAM

The endpoint’s OCRAM needs to be programmed with minimal firmware, using PBI commands. Minimal firmware is required on endpoint’s OCRAM to perform the following functions:

- Program CSU registers to allow secure world to make various peripherals’ registers accessible to non-secure world software and bus masters
- Write 0x0 at 0x10010000 location of OCRAM
- Poll for non-zero value at 0x10010000 location of OCRAM
- Jump to 0x82000000 location of DDR if 0x10010000 location of OCRAM is programmed to a non-zero value

Below is the assembly code of OCRAM firmware:

```assembly
/* Update CSU registers */
ldr x2, =0x1510000
ldr x0, =0xb8
ldr w1, =0xFF00FF00
loopa:  str w1, [x2,x0]
sub x0,x0,#0x4
cbnz x0,loopa
/* Write 0x0 at 0x10010000 */
```

NXP Semiconductors
ldr x2, =0x10010000
ldr x0, =0x0
str x0, [x2]
/* Poll for non-zero value */
loopb: ldr x0, [x2]
cbz x0, loopb
/* Jump to 0x82000000 */
ldr x30, =0x82000000
br x30

4.3.1 Convert assembly file into binary file

Perform these steps in Ubuntu to convert the assembly file into binary file:

1. Make object (.o) file from assembly (.S) file:
   aarch64-linux-gnu-gcc -c <assembly_filename.S>
2. Make binary (.bin) file from object (.o) file:
   aarch64-linux-gnu-objcopy -O binary <object_filename.o> <binary_filename.bin>

4.4 Program scratch register

SCFG_SCRATCHRW2 register is programmed with the OCRAM location where GPP jumps to after executing boot ROM code.

Program SCFG_SCRATCHRW2 register (offset 0x01570604) with 0x10000000 (address of OCRAM). Minimal firmware is programmed at OCRAM address 0x10000000.

4.5 Program inbound windows

After link training, the endpoint needs to make its CCSRBAR register space and DDR SDRAM memory space accessible from root complex. This can be done by programming endpoint’s inbound windows. In the current example, two inbound windows are used, which can be defined as follows:

- Inbound window 0: Represents a memory window that matches to BAR0 (BAR Match mode), which maps to 0x0000000000000000 in CCSRBAR and OCRAM memory space
- Inbound window 1: Represents a memory window that matches to BAR1 (BAR Match mode), which maps to 0x0000000080000000 in DDR SDRAM memory space

Program inbound window 0

Perform these steps to program inbound window 0:

1. Set the BAR Mask (size) in BAR0_MASK register:
   a. Write 0x1FFF_FFFF to BAR0_MASK at offset 0x1010.
2. Set up Base Address Register (BAR0):
   a. Write 0x80000000 to BAR0 at offset 0x10.
3. Set up Index register:
   a. Write 0x80000000 to Index Register at offset 0x900
4. Set up Region Control 2 register:
   a. Write 0xC0000000 to Region Control 2 register at offset 0x908.
5. Set up Translation Address registers:
   a. Write 0x80000000 to window 0’s Lower Target Address register at offset 0x918.
   b. Write 0x00000000 to window 0’s Upper Target Address register at offset 0x91C.

![Figure 10. Endpoint inbound window 0](image)

**Program inbound window 1**

Perform these steps to program inbound window 1:

1. Set the BAR Mask (size) in BAR1_MASK register:
   a. Write 0x1FFFFFFF to BAR1_MASK at offset 0x1014.
2. Set up Base Address Register (BAR1):
   a. Write 0x00000000 to BAR1 at offset 0x14.
3. Set up Index register:
   a. Write 0x80000001 to Index register at offset 0x900.
4. Set up Region Control 2 register:
   a. Write 0xC0000100 to Region Control 2 register at offset 0x908.
5. Set up Translation Address registers:
   a. Write 0x00000000 to window 1’s Lower Target Address register at offset 0x918.
   b. Write 0x00000000 to window 1’s Upper Target Address register at offset 0x91C.

![Figure 11. Endpoint inbound window 1](image)

### 4.6 PBL configuration using CodeWarrior

This section explains the steps to configure and generate RCW and PBI binary file using QCVS tool in CodeWarrior for ARMv8. Using CodeWarrior, you can perform the following functions:

- Modify RCW values
Configuring LS1046ARDB as PCIe endpoint

- Write 0x10000000 (the location of OCRAM) on SCFG_SCRATCHRW2 register (at 0x1570604) and 0x00000000 on SCFG_SCRATCHRW1 register (at 0x1570600)
- Program inbound window
- Program OCRAM with minimal firmware using PBI commands

Follow these steps in CodeWarrior to perform PBL configuration:

1. Start CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA.

   ![Figure 12. CodeWarrior welcome window](image_url)

   2. Choose File > New > QorIQ Configuration Project to create a new QCVS project.
3. Specify project name. Click Next.

Figure 13. Creating a QCVS project

Figure 14. Specifying project name
4. Select **LS1046A** as the processor to be used. Click **Next**.

![Figure 15. Selecting processor](image)

5. Select **PBL – Preboot Loader RCW Configuration** as the QCVS component. Click **Next**.
6. Select Create default configuration to start with the default PBL configuration or select Import configuration from an existing PBL file if you want to import PBL configuration from an SDK image (for example, rcw_xxxx.bin). Click Finish.
7. On Properties page of Component Inspector view, check clock settings of the LS1046ARDB.
8. Select the PBL component under Components folder in Components view to configure RCW fields.

9. On Import page of Component Inspector view, click Load from file or Read from target to load RCW+PBL data from computer or from target, respectively. See Layerscape Software Development Kit Documentation for building the LS1046ARDB SDK images.
10. Click **Import** to update PBL configuration. In the current project, PBL data is loaded from the LSDK 18.03 image, `rcw_1600.bin`.

11. Set **SRDS_PRTCL_S2** as 5506 to have PCIe controller 1/2/3 on SerDes2 lanes. Set **HOST_AGT_PEX** as 3'b001 to program all three PCIe controllers as Agent/Endpoint mode. You can configure other RCW fields as per your requirements. Click the link next to **PBI Data input** field to add/modify any PBI commands.
12. Program SCFG_SCRATCHRW2 register as 0x10000000 at 0x1570604.

13. Choose CCSR Write (4-byte) as the PBI command. Enter 570158 as the system address and 00001000 as the data to program ALTCBAR register with OCRAM base address. Click Add Command.
14. Choose **ACS Write from file** as the PBI command. Click **Load from file** and load binary file that needs to be programmed on OCRAM. Choose 64 as the byte count. Click **Add command**.

**NOTE**

The file size must be in multiples of 128 bytes.
Figure 25. Adding PBI commands on PBI data from a binary file

Similarly, other PBI commands can be added or modified in QCVS tool. The figure below shows the additions and modifications made to PBI data.

15. Click **Generate Processor Expert Code** to generate the PBL file.
16. Program QSPI flash with generated RCW+PBI binary file at 0x0 offset. Endpoint is ready to be plugged into the setup. Continue with instructions provided in Initialize endpoint’s DDR.

5 Appendix: Using C29XPCIE-RDB as PCIe root complex

Alternatively, this use case setup can be created by using C29XPCIE-RDB as root complex and LS1046ARDB as endpoint. Following subsections describe configuration of C29XPCIE-RDB as PCIe root complex.

5.1 Board switch settings

The C29XPCIE-RDB can be used as a root complex by configuring it in Standalone Host mode. The DIP switch settings of the C29XPCIE-RDB for Standalone Host mode are shown in the table below.

<table>
<thead>
<tr>
<th>DIP switch</th>
<th>Settings</th>
<th>Notes</th>
</tr>
</thead>
</table>
| SW4[1:8]   | 01011000 | • 0 indicates ON  
|            |          | • 1 indicates OFF |
| SW5[1:8]   | 11110000 |       |
| SW6[1:8]   | 00001111 |       |
| SW7[1:8]   | 10011111 |       |
| SW8[1:8]   | 00001011 |       |
5.2 Program outbound window

Programming outbound window involves mapping local address space to PCIe address space. This mapping can be performed by programming Local Access Window (LAW) registers of the C29x processor. For more details on LAW registers, see the C29x Crypto Coprocessor Family Reference Manual.

U-Boot programs LAW2 registers for mapping to PCIe address space. Below are some details of LAW2 registers:

- Base address: 0x80000000
- Size: 512 MB
- Target ID: 0x2

Perform these steps to program an outbound window on the C29XPCIE-RDB:

1. Program PEX_PEXOWAR1 register as 0x8004401c:
   - `PEX_PEXOWAR1[EN] = 1'b1`
   - `PEX_PEXOWAR1[RTT] = 4'b0100, memory read`
   - `PEX_PEXOWAR1[WTT] = 4'b0100, memory write`
   - `PEX_PEXOWAR1[OWS] = 6'b011100, size 512 MB`
2. Program PEX_PEXOBAR1 register as 0x00080000
   - Window base address = 0x80000000
3. Program PEX_PEXOTAR1:
   a. Set translation of outbound window as 0x00000000 (see figures below) using the following command:

```
=> mw.l e000ac20 0
```

This window can be used to access DDR controller registers and OCRAM space of endpoint. Follow instructions in Initialize endpoint's DDR to initialize the DDR controller of the endpoint.

```
=> md e000ac00
e000ac00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
e000ac10: 80044023 00000000 00000000 00000000 00000000 00000000 00000000 00000000
e000ac20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
e000ac30: 8004401c 00000000 00000000 00000000 00000000 00000000 00000000 00000000
e000ac40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

**Figure 28. Setting translation of C29XPCIE-RDB outbound window as 0x00000000**

b. Change translation of outbound window to 0x80000000 (see figures below) using the following command:

```
=> mw.l e000ac20 00080000
```

**Figure 29. C29XPCIE-RDB outbound window with translation address as 0x00000000**
Endpoint’s DDR region can now be accessed through this window. Follow instructions in Initialize endpoint’s DDR to load boot loader on endpoint’s DDR.

```bash
=> md e000ac00
| e000ac00: 00000000 00000000 00000000 00000000 | @#...@#.....
| e000ac10: 80044023 00000000 00000000 00000000 |     @        
| e000ac20: 00080000 00000000 00080000 00000000 |     @        
| e000ac30: 0004401c 00000000 00000000 00000000 |     @        
| e000ac40: 00000000 00000000 00efc000 00000000 |     @        
| e000ac50: 8008800f 00000000 00000000 00000000 |     @        
```

| Outbound translation address register |
| Outbound window base address register |
| Outbound window attributes register |

**Figure 30. Setting translation of C29XPCIE-RDB outbound window as 0x80000000**

**Figure 31. C29XPCIE-RDB outbound window with translation address as 0x80000000**

NOTE
For programming an outbound window, bus master and memory space must be enabled in PCIe configuration space of C29XPCIE-RDB.

6 Appendix: Hardware and software resources

The table below shows the hardware and software tools you may require for use case setup.

**Table 6. Hardware and software tools**

<table>
<thead>
<tr>
<th>Tool</th>
<th>How to access?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware tools</strong></td>
<td></td>
</tr>
<tr>
<td>QorIQ LS1046A reference design board</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>C29x Crypto Coprocessor PCI Express Adapter Platform</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>CodeWarrior TAP</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>QorIQ LS Processor Probe Tips for CodeWarrior TAP</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>Power Architecture processor (COP) Probe Tips for CodeWarrior TAP</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td><strong>Software tools</strong></td>
<td></td>
</tr>
<tr>
<td>CodeWarrior Development Software for ARM v8 64-bit based QorIQ LS series Processors</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>CodeWarrior Development Studio for Power Architecture Processors</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>Layerscape Software Development Kit</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>Linux SDK for QorIQ Processors</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
</tbody>
</table>
7 Appendix: Related documentation

The table below lists documents that provide additional information related to the concept explained in this document.

<table>
<thead>
<tr>
<th>Document</th>
<th>Description</th>
<th>How to access?</th>
</tr>
</thead>
<tbody>
<tr>
<td>QorIQ LS1046A Reference Design Board Getting Started Guide (LS1046ARDBGSG)</td>
<td>Describes different components of the LS1046ARDB and explains how to set up and boot the board</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>QorIQ LS1046A Reference Design Board Reference Manual (LS1046ARDBRM)</td>
<td>Explains the LS1046ARDB interfaces and configuration</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>QorIQ LS1046A Reference Manual (LS1046ARM)</td>
<td>Provides a detailed description of the LS1046A multicore processor and its features, such as the memory map, serial interfaces, power supply, chip features, and clock information</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>QorIQ LS1046A Data Sheet (LS1046A)</td>
<td>Contains information on LS1046A pin assignments, electrical characteristics, hardware design considerations, package information, and ordering information</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>LS1046A Chip Errata (LS1046ACE)</td>
<td>Describes the latest fixes and workarounds for the chip. It is strongly recommended that this document is thoroughly researched prior to starting a design with the chip.</td>
<td>Contact your NXP sales representative</td>
</tr>
<tr>
<td>RAM Boot using CodeWarrior on LS1046ARDB Application Note (AN12081)</td>
<td>Explains how to deploy U-Boot directly to the DDR of the LS1046ARDB using CodeWarrior</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>C29x PCIe Card Quick Start Guide (C29XPCIEQS)</td>
<td>Explains C29x PCIe board settings and physical connections needed to boot the board</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>C29x PCIe Card User Guide (C29XPCIEUG)</td>
<td>Explains the C29x PCIe board interfaces and configuration</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>C29x Crypto Coprocessor Family Reference Manual (C29XRM)</td>
<td>Defines the functionality of the NXP C29x family</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>Layerscape Software Development Kit Documentation (LSDK-REV-yy-mm)</td>
<td>Describes LSDK, which is a complete Linux kit for NXP QorIQ Arm-based SoCs and the reference and evaluation boards available for them</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA, Targeting Manual (CWARMv8TM)</td>
<td>Explains how to use the CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA product</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
<tr>
<td>CodeWarrior TAP Probe User Guide (CWTAPUG)</td>
<td>Provides details of CodeWarrior® TAP, which enables target system debugging via a standard debug port (usually JTAG) while connected to a developer’s workstation via Ethernet or USB</td>
<td><a href="http://www.nxp.com">www.nxp.com</a></td>
</tr>
</tbody>
</table>
8 Revision history

The table below summarizes revisions to this document.

Table 8. Revision history

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Topic cross-reference</th>
<th>Change description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rev. 0</td>
<td>11/2018</td>
<td></td>
<td>Initial public release</td>
</tr>
</tbody>
</table>
Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP 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 NXP 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. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, Freescale, the Freescale logo, CodeWarrior, Layerscape, and QorIQ are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm and Cortex are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

© 2018 NXP B.V.