NXP Semiconductors
Application Note

Document Number: AN12246
Rev. 0, 11/2018

Loading Boot Loader on LS1046ARDB through PCIe

Contents

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.

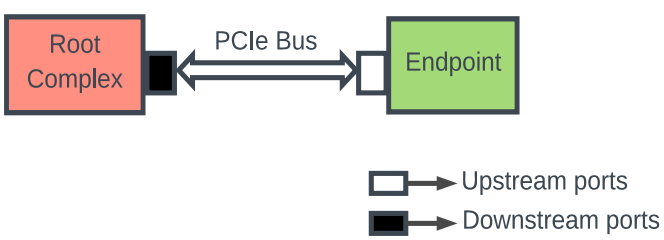

Figure 1. Use case setup diagram

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.

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.

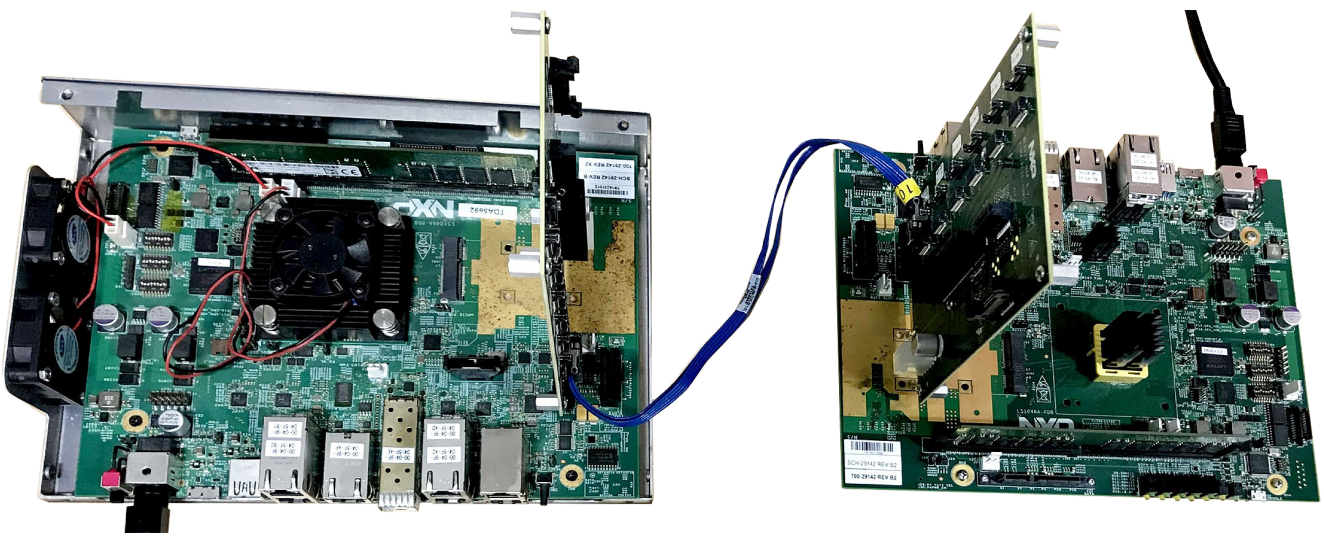

Figure 2. Use case setup

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

DIP switch	Settings	Notes
SW3[1:8]	01000110	• 0 indicates OFF • 1 indicates ON
SW4[1:8]	00111011	
SW5[1:8]	00100010	

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)

Field name	Description	Value
RCW[HOST_AGT_PEX]	Selects between Root Complex (RC) and Endpoint (EP) modes	3'b000: All Host mode
RCW[SRDS_PRTCL_Sn]	Determines the link width	0x1133: SRDS_PRTCL_S1 0x5506: SRDS_PRTCL_S2
RCW[SRDS_DIV_PEX_Sn]	Determines the link speed	2'b00: SRDS_DIV_PEX_S1 2'b00: SRDS_DIV_PEX_S2

3.3 Program outbound windows

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

Table 3. PCIe controller base address

PCIe controller	Base address
PCIe controller 1	340_0000h
PCIe controller 2	350_0000h
PCIe controller 3	360_0000h

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.

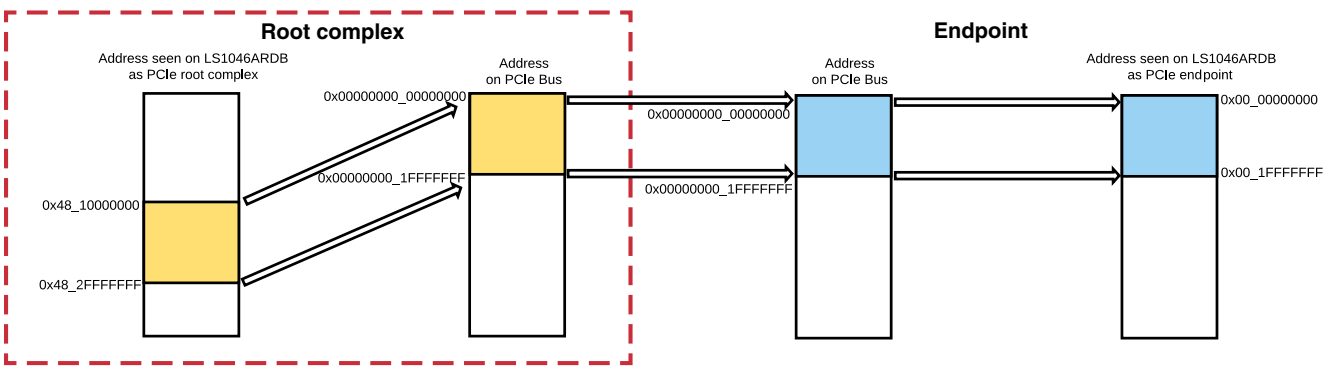

Figure 3. Outbound window 2

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.

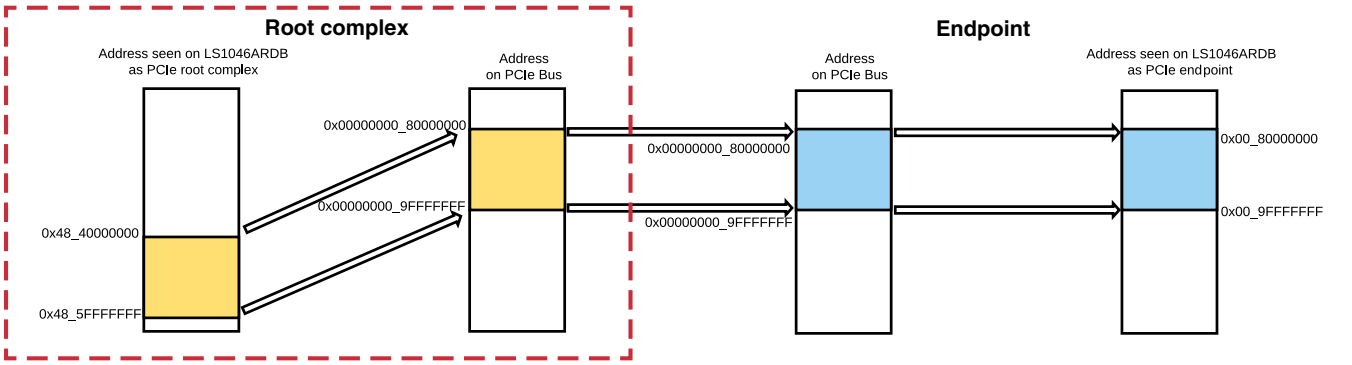

Figure 4. Outbound window 3

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.

```
=> md 3500900
03500900: 00000002 00000000 80000000 10000000    ................
03500910: 00000048 2fffffff 00000000 00000000    H...../.........
03500920: 00000000 00000000 00000000 00000000    ................
03500930: 00000000 00000000 00000000 00000000    ................
03500940: 00000000 00000000 00000000 00000000    ................
03500950: 00000000 00000000 00000000 00000000    ................
```

Outbound window base address Register
Limit Address Register
Outbound translation address Register

Figure 5. Outbound window 2 register dump

2. Configure DDR controller registers of PCIe endpoint:

NOTE

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

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

```
=> mw.l 4811080008 ff010000
=> mw.l 4811080084 22030080
=> mw.l 48110800c4 00000000
=> mw.l 4811080010 00000000
=> mw.l 4811080088 00000000
=> mw.l 48110800c8 00000000
=> mw.l 4811080018 00000000
=> mw.l 481108008c 00000000
=> mw.l 48110800cc 00000000
=> mw.l 4811080104 180077d1
=> mw.l 4811080100 00111002
=> mw.l 4811080108 4390fcf2
=> mw.l 481108010c 97015900
=> mw.l 4811080b28 00000480
=> mw.l 4811080b2c c1000000
=> mw.l 4811080110 00400465
=> mw.l 4811080128 efbeadde
=> mw.l 4811080114 11114000
=> mw.l 4811080118 30060103
=> mw.l 4811080200 30060100
=> mw.l 4811080208 30060100
=> mw.l 4811080210 30060100
=> mw.l 481108011c 00021000
=> mw.l 4811080204 00021000
=> mw.l 481108020c 00021000
=> mw.l 4811080214 00021000
=> mw.l 4811080220 00050000
=> mw.l 4811080228 00040000
=> mw.l 4811080230 00040000
=> mw.l 4811080238 00040000
=> mw.l 4811080224 00004008
=> mw.l 481108022c 00004008
=> mw.l 4811080234 00004008
=> mw.l 481108023c 00004008
=> mw.l 4811080124 0000fe1f
=> mw.l 4811080130 00000002
=> mw.l 4811080160 02000000
=> mw.l 4811080164 00144005
=> mw.l 4811080260 00000000
=> mw.l 4811080168 00000000
=> mw.l 481108016c 00006026
=> mw.l 4811080250 00682205
=> mw.l 4811080400 5475c532
=> mw.l 4811080404 d40bbbd4
=> mw.l 4811080408 54f5c22e
=> mw.l 481108040c 01405dd9
=> mw.l 4811080170 0507098a
=> mw.l 4811080174 09f67586
=> mw.l 4811080190 110d0c0a
=> mw.l 4811080194 0e151412
=> mw.l 4811080f94 00000080
=> mw.l 4811080110 004004e5
```

3. Program outbound window 3 to access DDR region of PCIe endpoint:

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

Programmed registers are shown below.

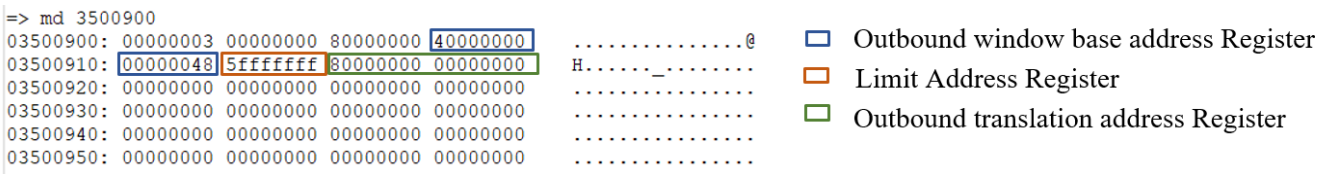

Figure 6. Outbound window 3 register dump

4. Read endpoint's DDR from root complex.

Figure 7. Endpoint's DDR data dump on 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.

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

The figure below shows the output of the above commands.

```
=> usb start
starting USB...
USB0:   Register 200017f NbrPorts 2
Starting the controller
USB XHCI 1.00
USB1:   Register 200017f NbrPorts 2
Starting the controller
USB XHCI 1.00
USB2:   Register 200017f NbrPorts 2
Starting the controller
USB XHCI 1.00
scanning bus 0 for devices... 2 USB Device(s) found
scanning bus 1 for devices... 1 USB Device(s) found
scanning bus 2 for devices... 1 USB Device(s) found
       scanning usb for storage devices... 1 Storage Device(s) found
=> fatload usb 0 4842000000 u-boot.bin reading u-boot.bin
709818 bytes read in 51 ms (13.3 MiB/s)
```

Figure 8. Loading RAMboot image on endpoint's DDR

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

```
=> mw.l 4820010000 1
```

7. Open endpoint (LS1046ARDB) serial port terminal to see U-Boot log, as shown in the figure below.

```
U-Boot 2016.092.0+ga06b209 (Aug 21 2017 - 11:32:33 +0530)

SoC:  LS1046AE Rev1.0 (0x87070010)
Clock Configuration:
       CPU0(A72):1600 MHz  CPU1(A72):1600 MHz  CPU2(A72):1600 MHz
       CPU3(A72):1600 MHz
       Bus:      600  MHz  DDR:      2100 MT/s  FMAN:     700  MHz
Reset Configuration Word (RCW):
       00000000: 0c150010 0e000000 00000000 00000000
       00000010: 11335506 40000012 40025000 c1000000
       00000020: 00200000 00000000 00000000 00238800
       00000030: 20124000 00003101 00000096 00000001
Model: LS1046A RDB Board
Board: LS1046ARDB, boot from QSPI vBank 4
CPLD:  V2.2
PCBA:  V2.0
SERDES Reference Clocks:
SD1_CLK1 = 156.25MHZ, SD1_CLK2 = 100.00MHZ
I2C:   ready
DRAM:  Detected UDIMM 18ASF1G72AZ-2G3B1
8 GiB (DDR4, 64-bit, CL=15, ECC on)
       DDR Chip-Select Interleaving Mode: CS0+CS1
SEC0: RNG instantiated
FSL_SDHC: 0
MMC: no card present
MMC: block number 0x2801 exceeds max(0x0)
MMC/SD read of PPA FIT header at offset 0x500000 failed
PSCI: PSCI does not exist.
Waking secondary cores to start from fff0c000
All (4) cores are up.
Using SERDES1 Protocol: 4403 (0x1133)
Using SERDES2 Protocol: 21766 (0x5506)
NAND:  512 MiB
MMC:   MMC: no card present
*** Warning - MMC init failed, using default environment

EEPROM: NXID v1
In:    serial
Out:   serial
Err:   serial
SATA link 0 timeout.
AHCI 0001.0301 32 slots 1 ports 6 Gbps 0x1 impl SATA mode
flags: 64bit ncq pm clo only pmp fbss pio slum part ccc apst
Found 0 device(s).
SCSI:  Net:
MMC read: dev # 0, block # 2080, count 128 ...
MMC: no card present
MMC: block number 0x8a0 exceeds max(0x0)
Fman1: Data at 00000000ffe13670 is not a firmware
PCIe0: pcie@3400000 Endpoint: no link
PCIe1: pcie@3500000 Endpoint: x1 gen1
PCIe2: pcie@3600000 Endpoint: no link
No ethernet found.
Hit any key to stop autoboot:  0
=>
```

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 4. POR parameters for PCIe controller (endpoint)

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

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:

```
/* 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 */
```

```
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.

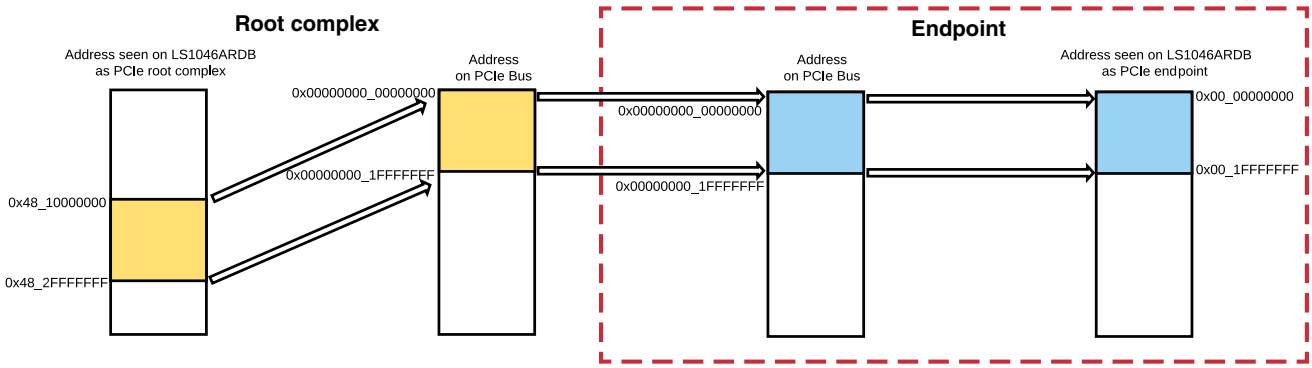

Figure 10. Endpoint inbound window 0

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.

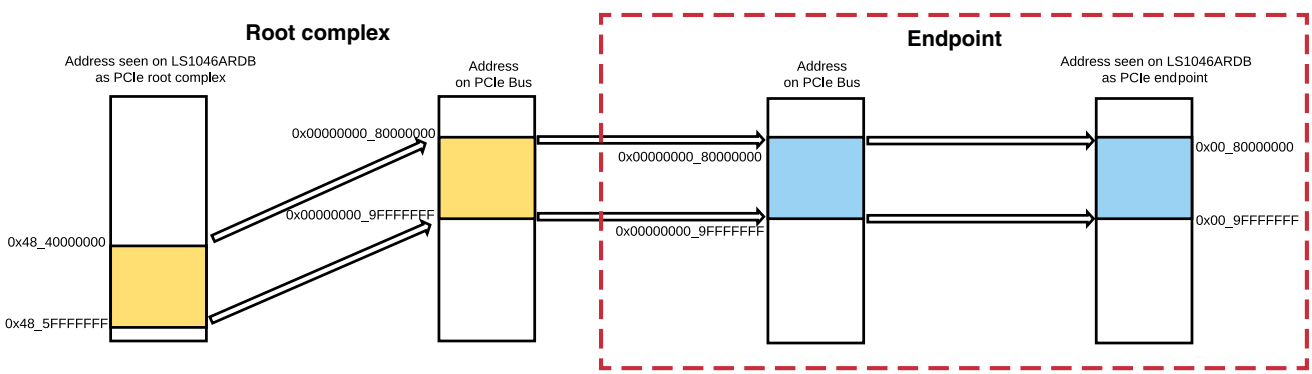

Figure 11. Endpoint inbound window 1

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

- 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**.

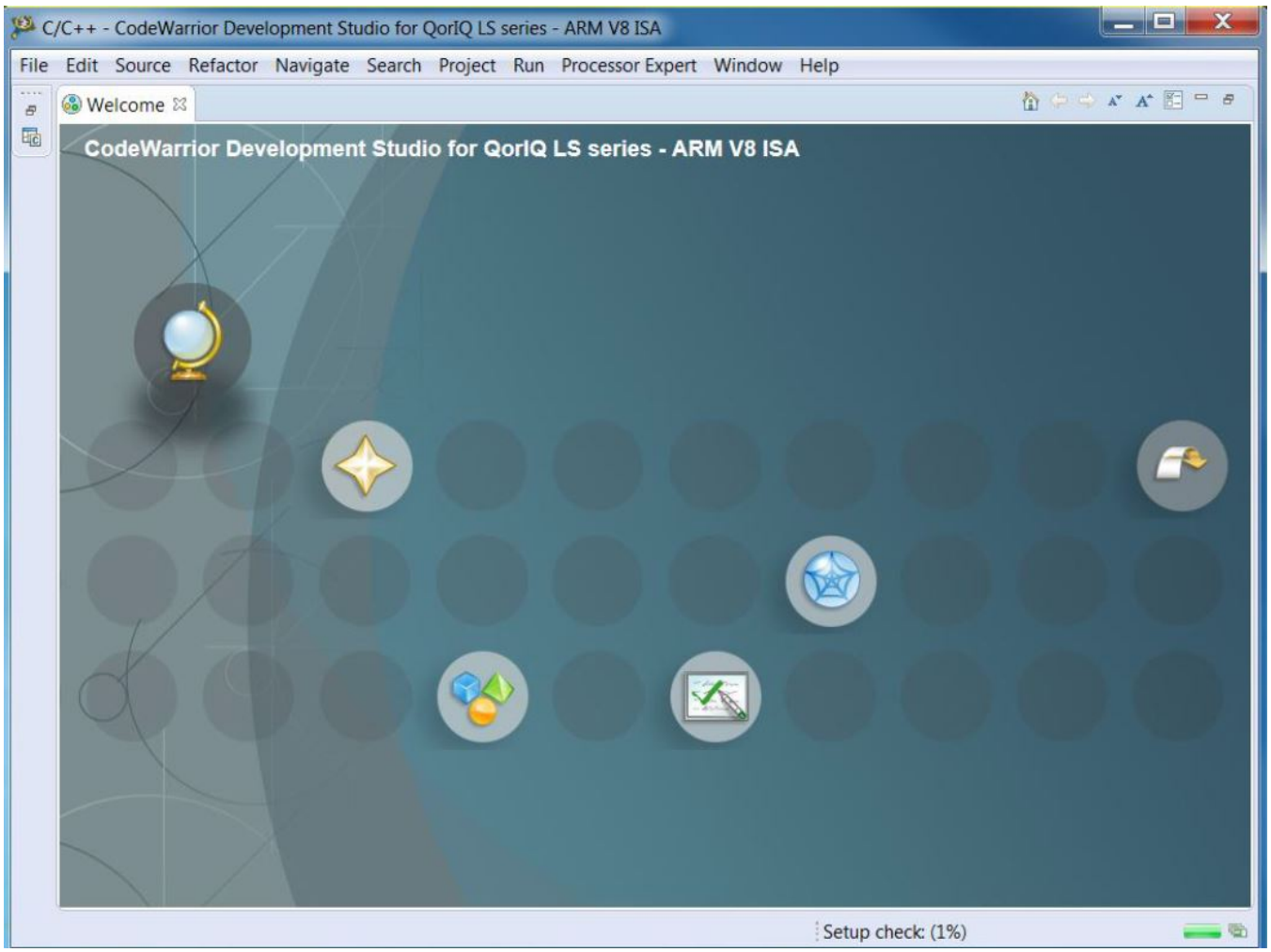

Figure 12. CodeWarrior welcome window

2. Choose **File > New > QorIQ Configuration Project** to create a new QCVS project.

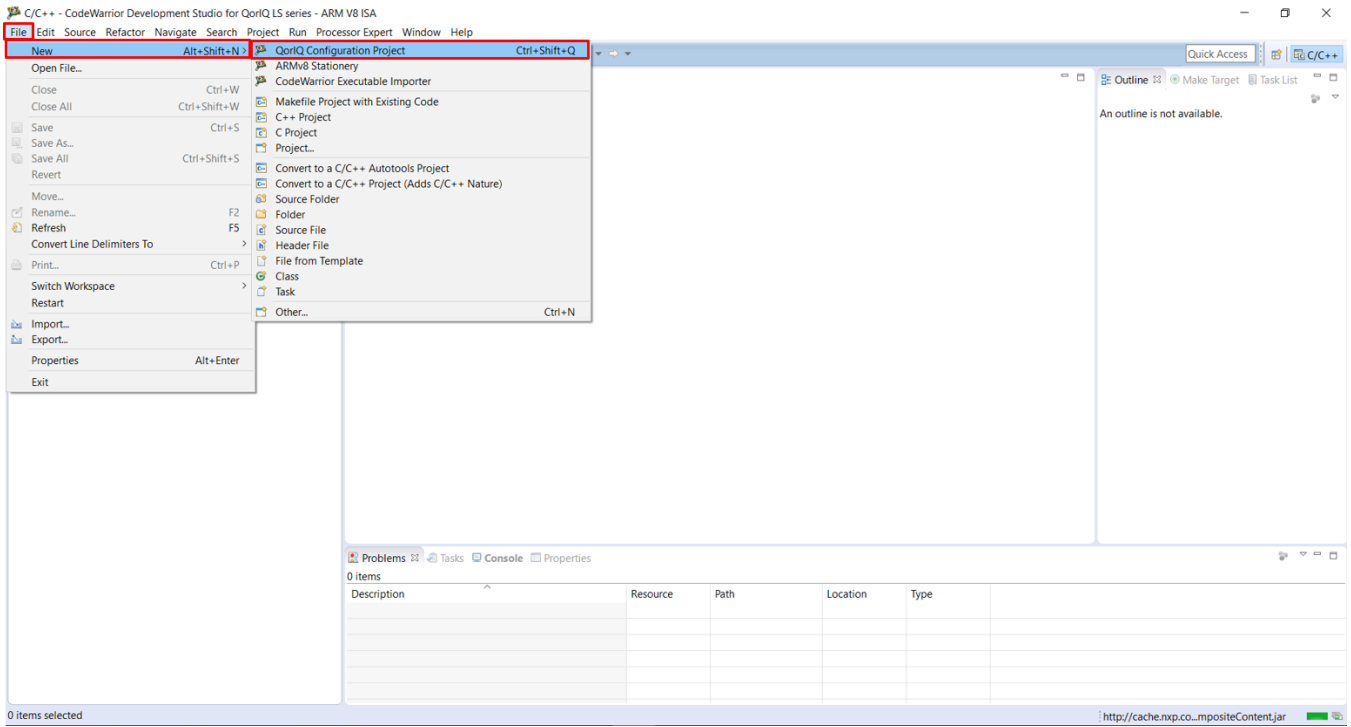

Figure 13. Creating a QCVS project

3. Specify project name. Click **Next**.

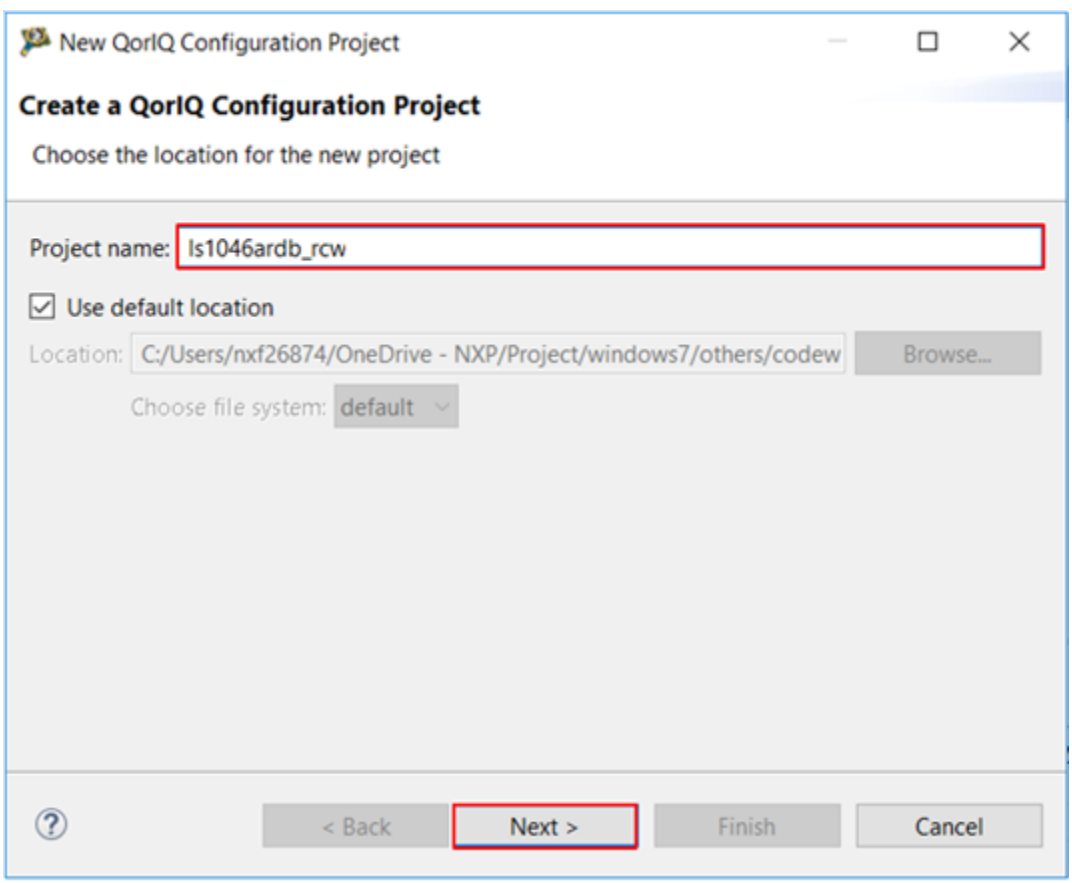

Figure 14. Specifying project name

4. Select **LS1046A** as the processor to be used. Click **Next**.

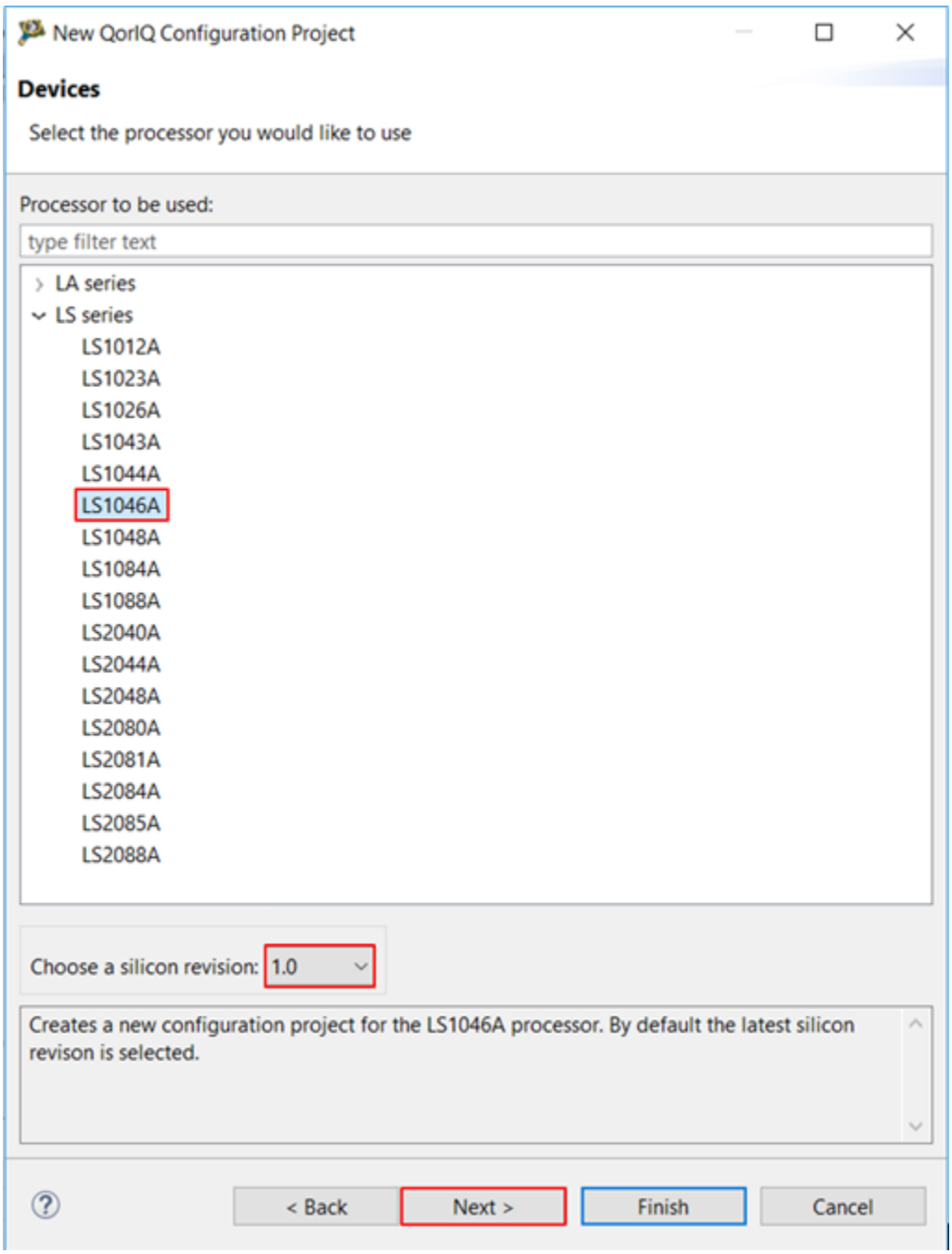

Figure 15. Selecting processor

5. Select **PBL – Preboot Loader RCW Configuration** as the QCVS component. Click **Next**.

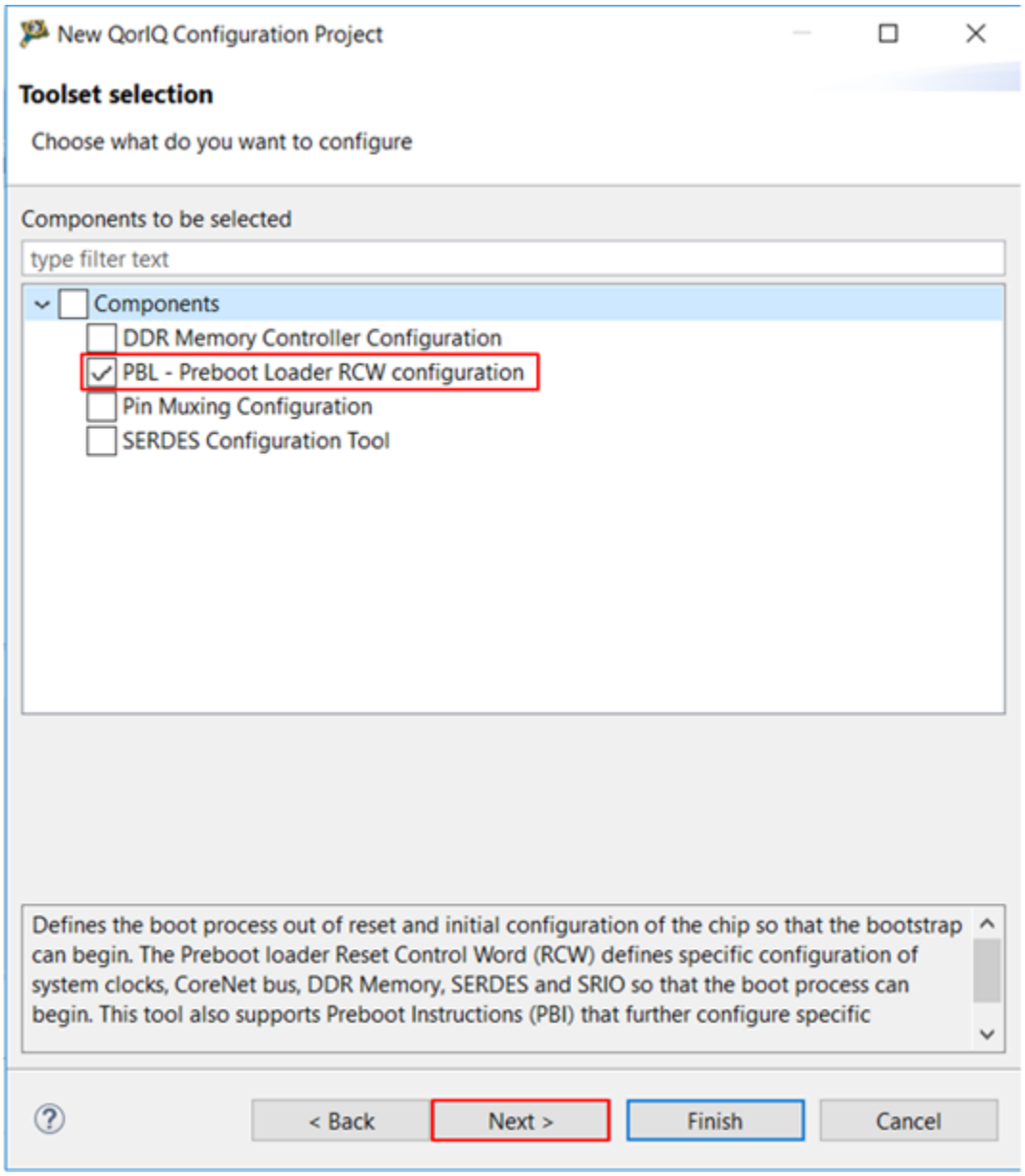

Figure 16. Selecting PBL as toolset

6. Select **Create default configuration** to start with the default PBL configration 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**.

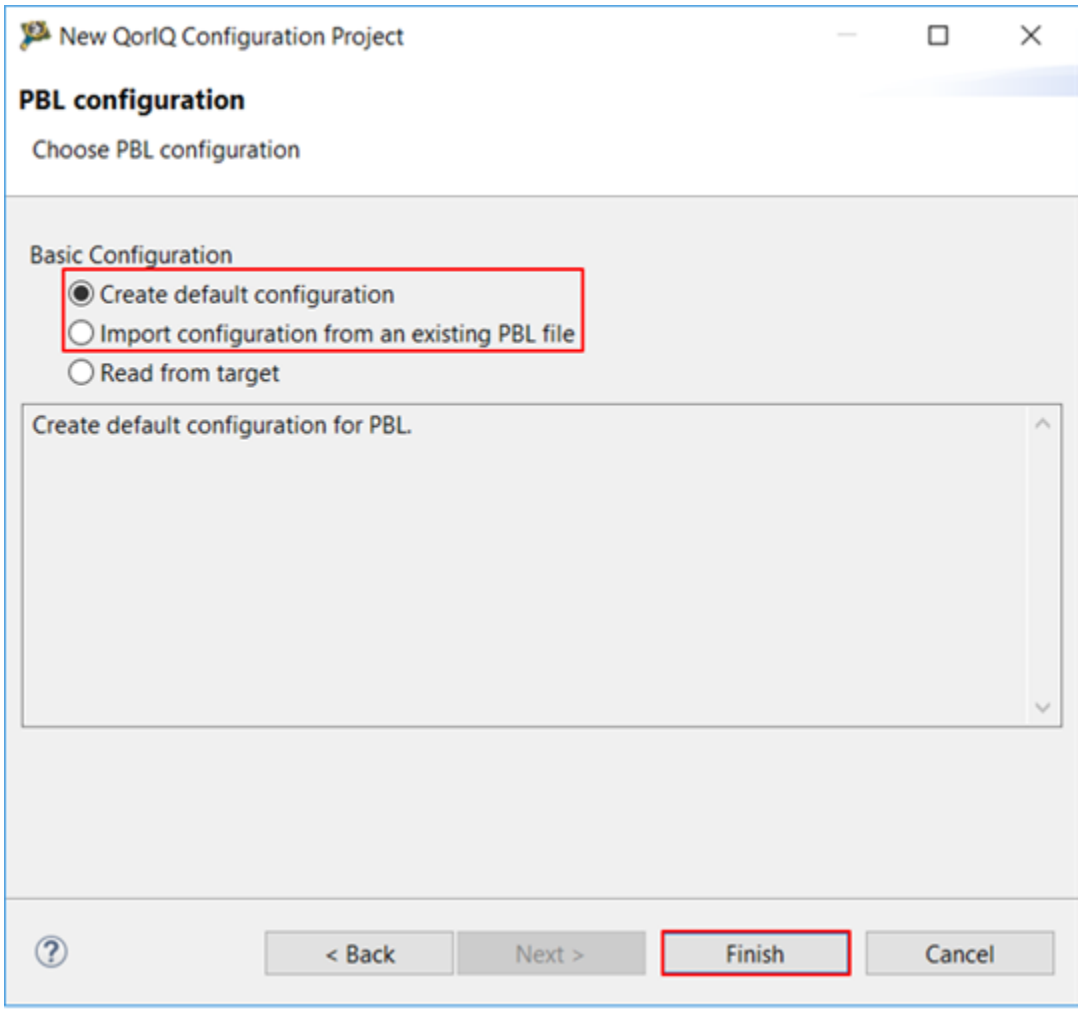

Figure 17. Selecting PBL configuration

7. On **Properties** page of **Component Inspector** view, check clock settings of the LS1046ARDB.

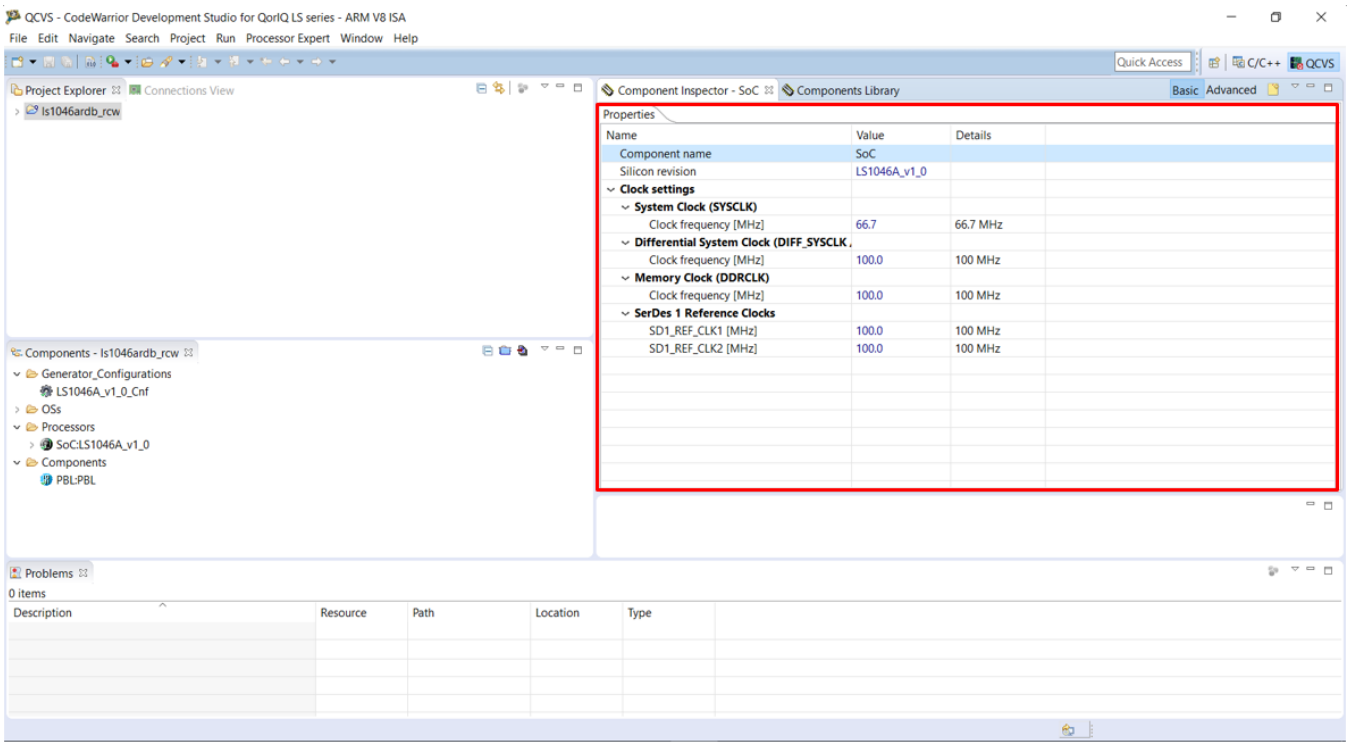

Figure 18. Clock settings of LS1046ARDB

8. Select the PBL component under **Components** folder in **Components** view to configure RCW fields.

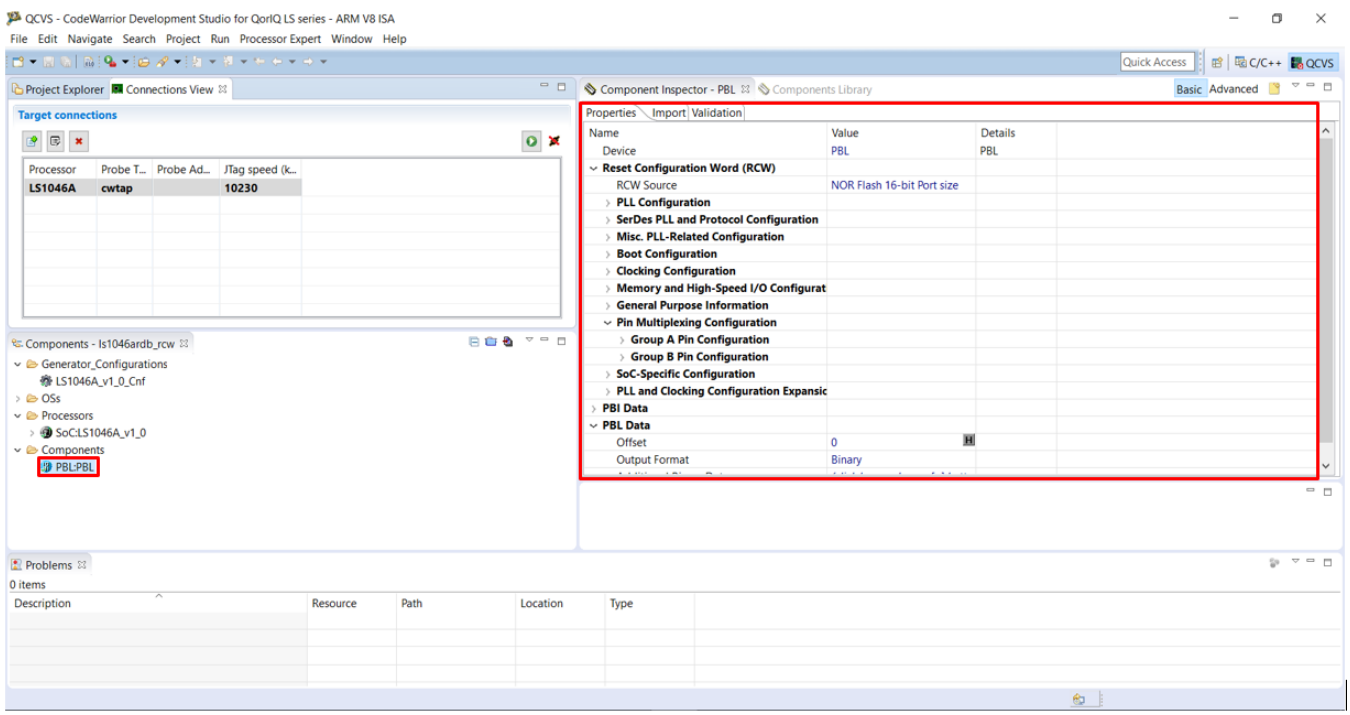

Figure 19. Configuring RCW fields

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.

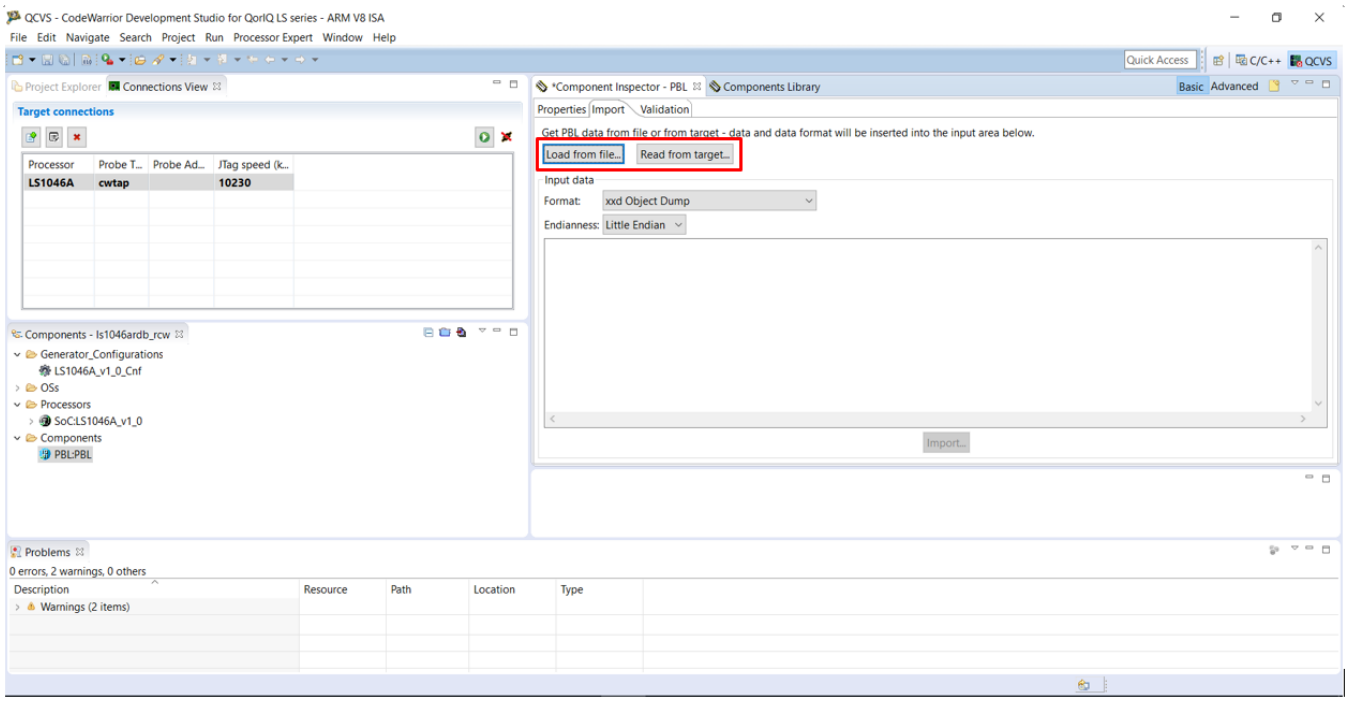

Figure 20. Accessing PBL data from file or from target

10. Click **Import** to update PBL configuration. In the current project, PBL data is loaded from the LSDK 18.03 image, `rcw_1600.bin`.

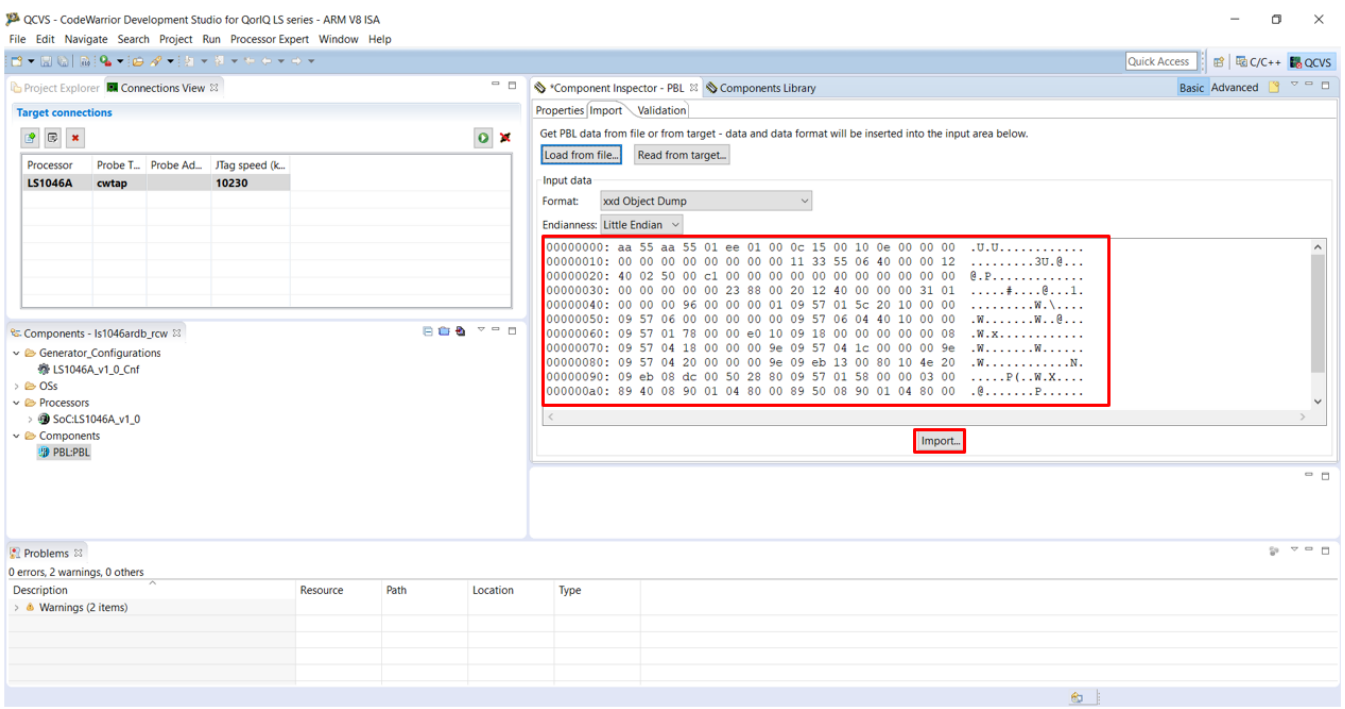

Figure 21. Imported RCW file

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.

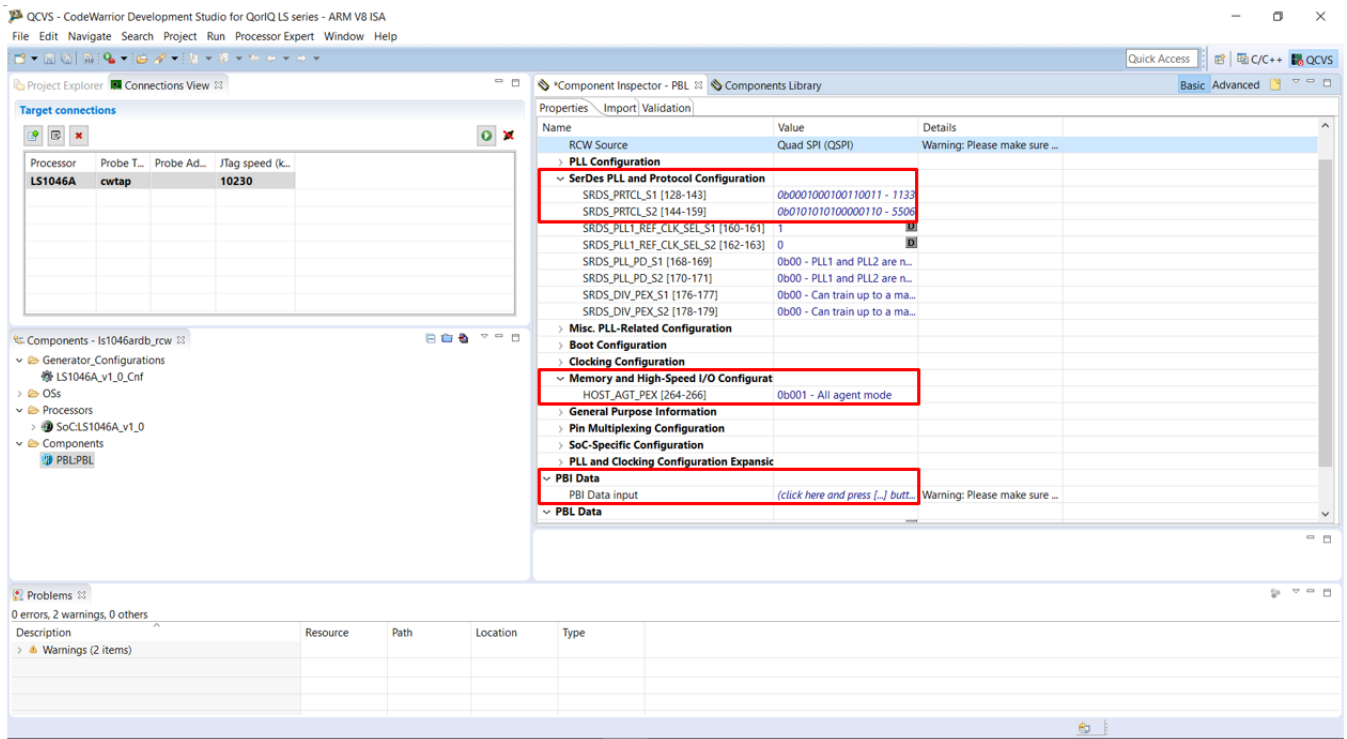

Figure 22. Selecting SerDes protocols and switching all PCIe controllers to Agent/ Endpoint mode

12. Program SCFG_SCRATCHRW2 register as 0x10000000 at 0x1570604.

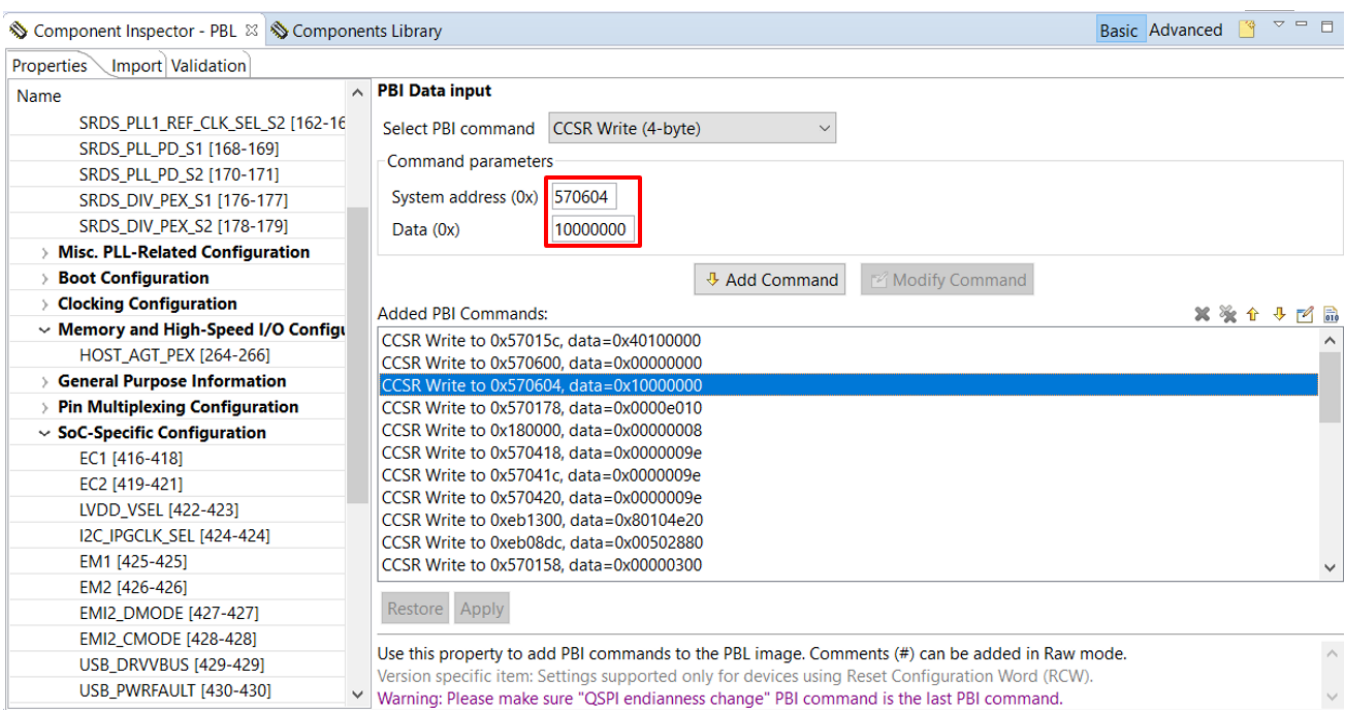

Figure 23. Programming SCFG_SCRATCHRW2 register

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**.

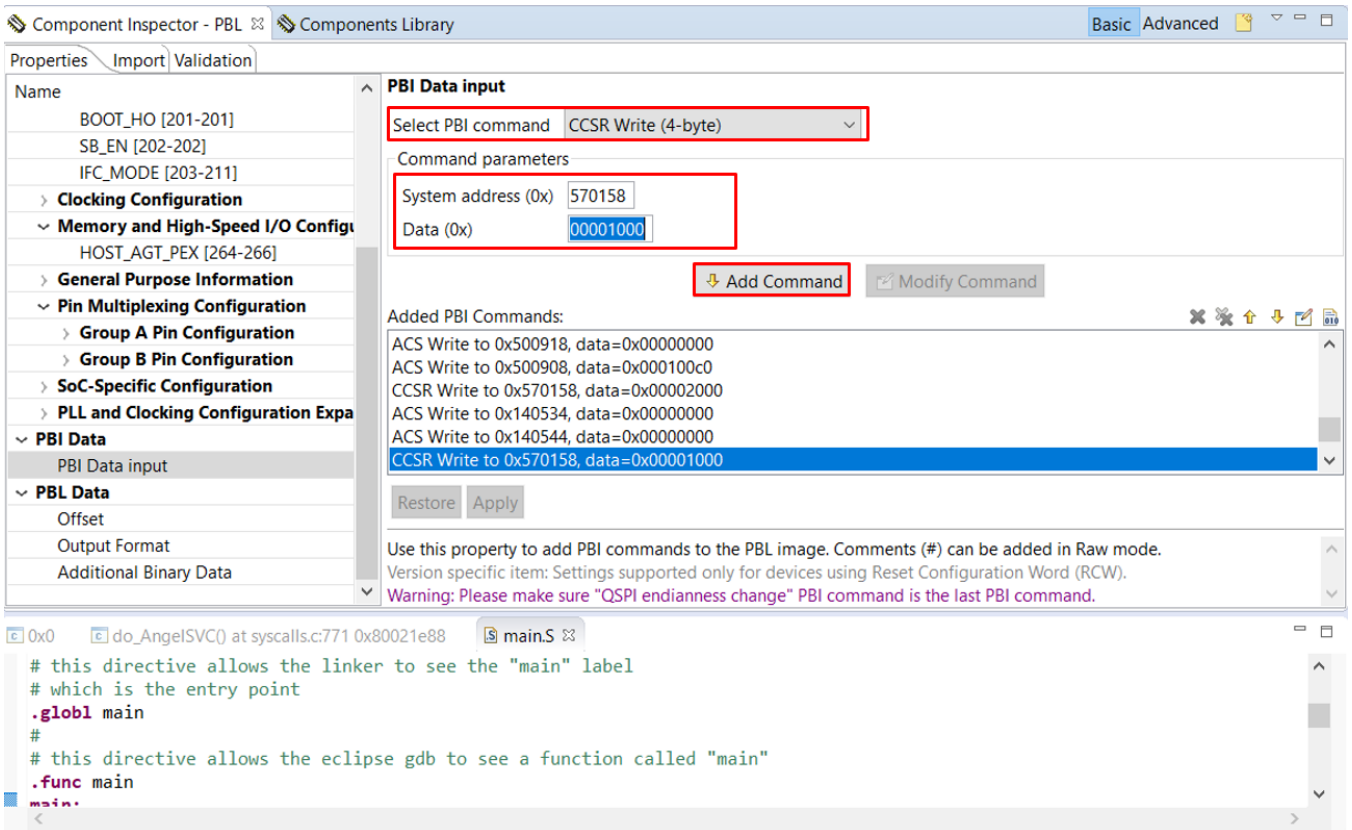

Figure 24. Programming ALTCBAR register

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.

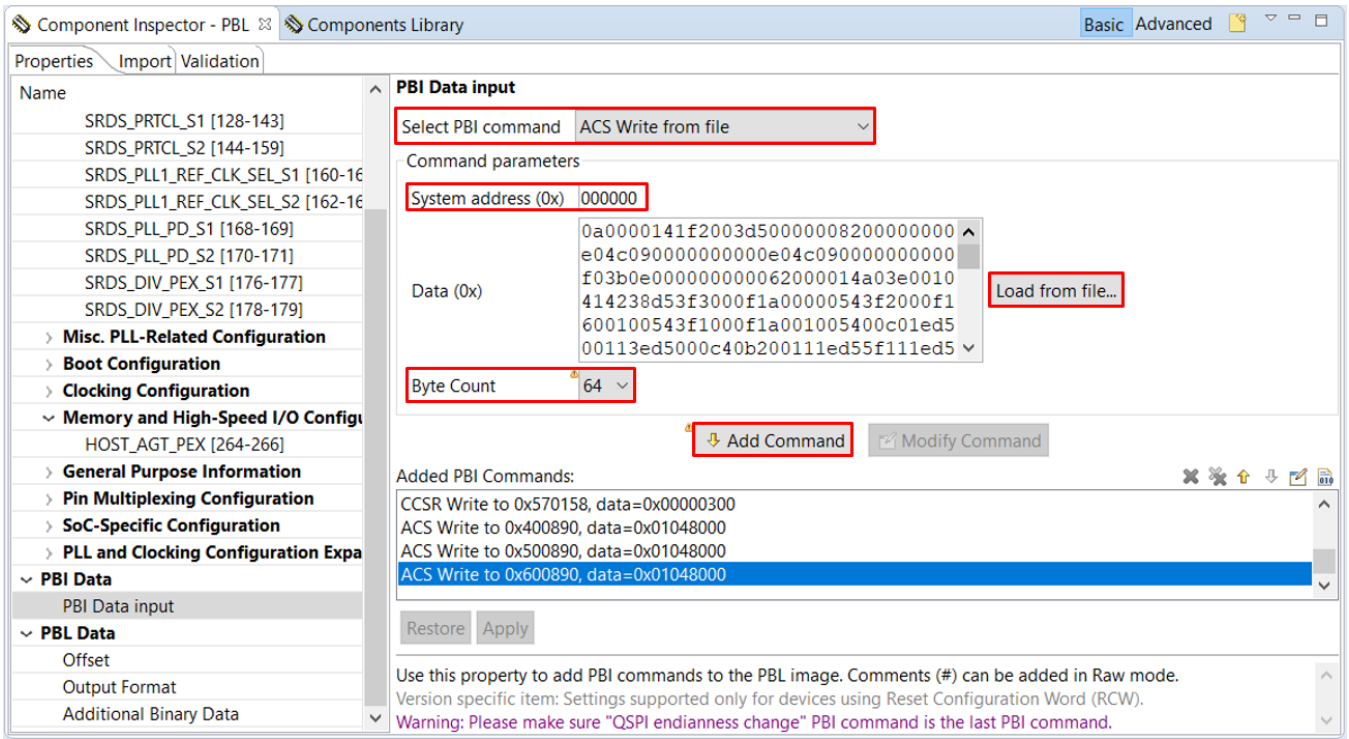

Figure 25. Adding PBI commands on PBI data from a binary file

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

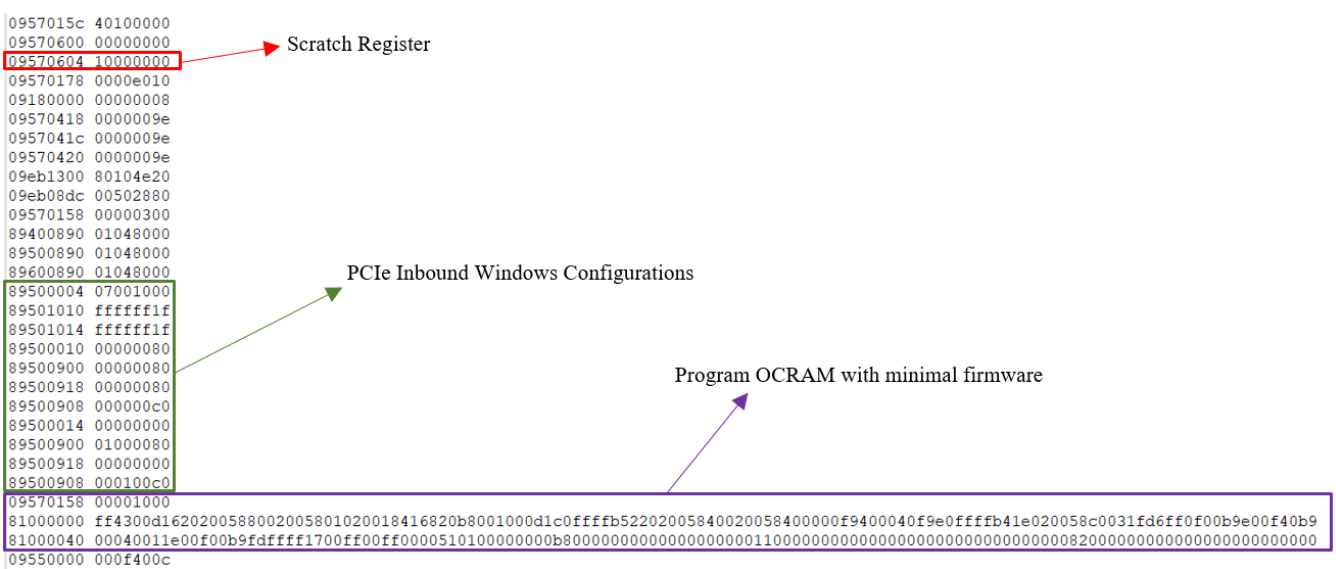

Figure 26. Additions and modifications to PBI data

15. Click **Generate Processor Expert Code** to generate the PBL file.

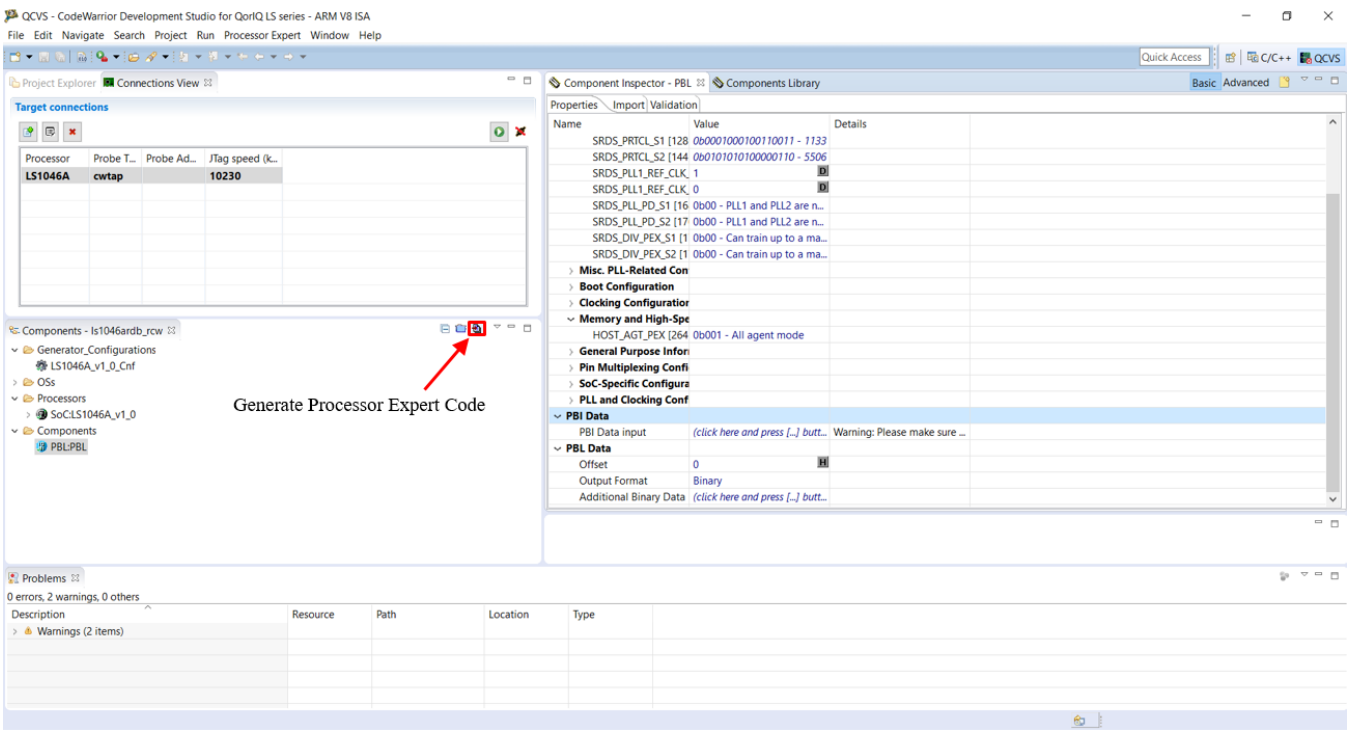

Figure 27. Generating PBL file

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 5. DIP switch settings

DIP switch	Settings	Notes
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 *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_PEXOWBAR1 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.

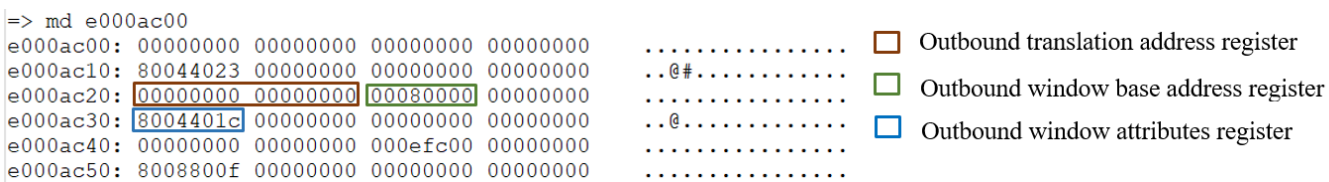

Figure 28. Setting translation of C29XPCIE-RDB outbound window as 0x00000000

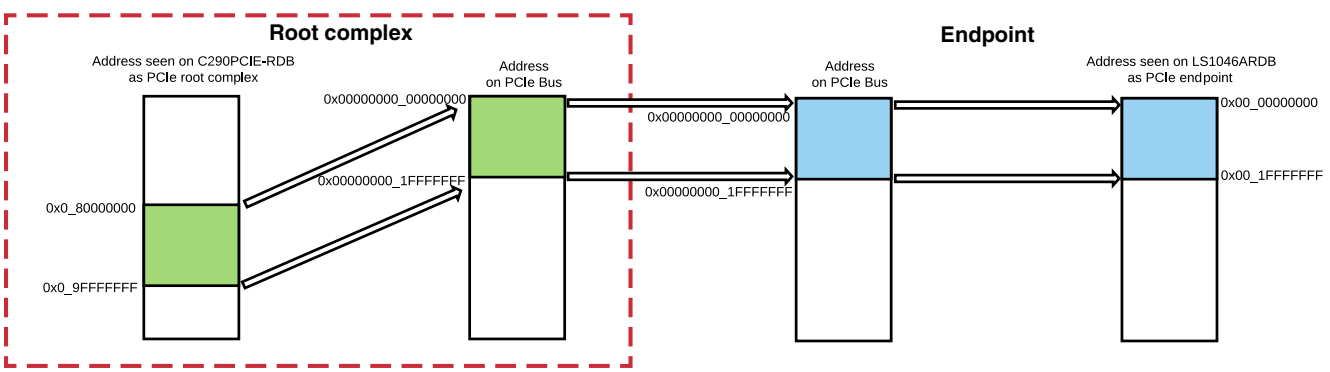

Figure 29. C29XPCIE-RDB outbound window with translation address as 0x00000000

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

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

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.

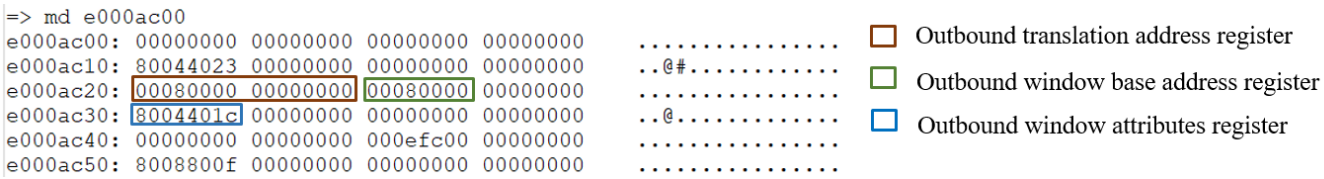

Figure 30. Setting translation of C29XPCIE-RDB outbound window as 0x80000000

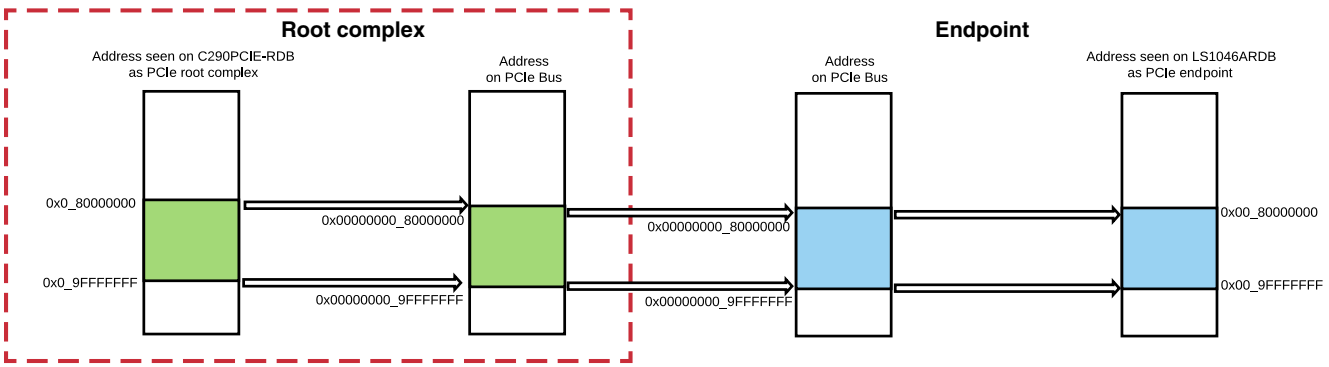

Figure 31. C29XPCIE-RDB outbound window with translation address as 0x80000000

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

Tool	How to access?
Hardware tools	
QorIQ LS1046A reference design board	www.nxp.com
C29x Crypto Coprocessor PCI Express Adapter Platform	www.nxp.com
CodeWarrior TAP	www.nxp.com
QorIQ LS Processor Probe Tips for CodeWarrior TAP	www.nxp.com
Power Architecture processor (COP) Probe Tips for CodeWarrior TAP	www.nxp.com
Software tools	
CodeWarrior Development Software for ARM v8 64-bit based QorIQ LS series Processors	www.nxp.com
CodeWarrior Development Studio for Power Architecture Processors	www.nxp.com
Layerscape Software Development Kit	www.nxp.com
Linux SDK for QorIQ Processors	www.nxp.com

7 Appendix: Related documentation

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

Table 7. Related documentation

Document	Description	How to access?
QorIQ LS1046A Reference Design Board Getting Started Guide (LS1046ARDBGSG)	Describes different components of the LS1046ARDB and explains how to set up and boot the board	www.nxp.com
QorIQ LS1046A Reference Design Board Reference Manual (LS1046ARDBRM)	Explains the LS1046ARDB interfaces and configuration	www.nxp.com
QorIQ LS1046A Reference Manual (LS1046ARM)	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	www.nxp.com
QorIQ LS1046A Data Sheet (LS1046A)	Contains information on LS1046A pin assignments, electrical characteristics, hardware design considerations, package information, and ordering information	www.nxp.com
LS1046A Chip Errata (LS1046ACE)	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.	Contact your NXP sales representative
RAM Boot using CodeWarrior on LS1046ARDB Application Note (AN12081)	Explains how to deploy U-Boot directly to the DDR of the LS1046ARDB using CodeWarrior	www.nxp.com
C29x PCIe Card Quick Start Guide (C29XPCIEQS)	Explains C29x PCIe board settings and physical connections needed to boot the board	www.nxp.com
C29x PCIe Card User Guide (C29XPCIEUG)	Explains the C29x PCIe board interfaces and configuration	www.nxp.com
C29x Crypto Coprocessor Family Reference Manual (C29XRM)	Defines the functionality of the NXP C29x family	www.nxp.com
Layerscape Software Development Kit Documentation (LSDK-REV-yy-mm)	Describes LSDK, which is a complete Linux kit for NXP QorIQ Arm-based SoCs and the reference and evaluation boards available for them	www.nxp.com
CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA, Targeting Manual (CWARMv8TM)	Explains how to use the CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA product	www.nxp.com
CodeWarrior TAP Probe User Guide (CWTAPUG)	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	www.nxp.com

8 Revision history

The table below summarizes revisions to this document.

Table 8. Revision history

Revision	Date	Topic cross-reference	Change description
Rev. 0	11/2018		Initial public release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number AN12246
Revision 0, 11/2018