Freescale Semiconductor
Application Note

Document Number: AN5227

Configuring LS1 Processors for Secure Boot and Secure Debug using CodeWarrior for ARMv7

1. Introduction

This document describes the necessary steps required to configure an LS1 processor for secure boot and secure debug using CodeWarrior for QorIQ LS series for ARMv7 ISA.

This document explains:

- Steps to generate keys and code signing tool
- Steps to program fuses using CodeWarrior for ARMv7
- Secure debug
- Board bring-up and board recovery

Contents

2. Key generation

Freescale provides code signing tool (CST) to assist you with the secure process. Using the tools available with CST, various keys can be generated:

- RSA public and private keys
- CSF header
- OTPMK key
- Debug response value register

For more information, see SDK Knowledge Center.

3. Fuse programming

To enable fuse programming, POVDD must be high.

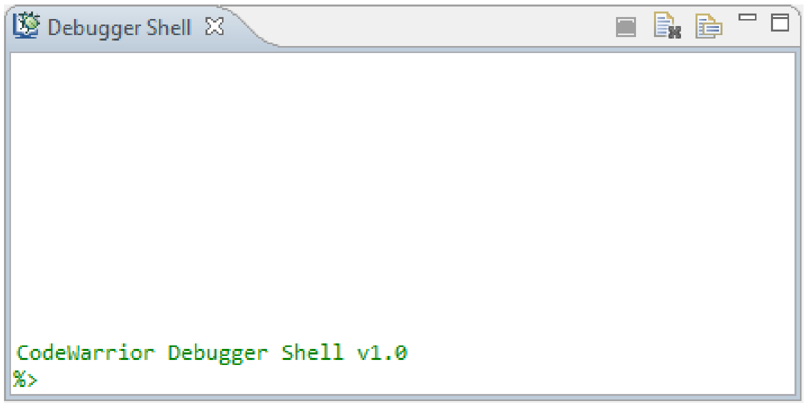
To burn fuses using CodeWarrior for ARMv7, perform these steps:

1. Create a CodeWarrior for ARMv7 bareboard project.

NOTE	LS1021A Rev. 2 is the default CPU supported by the latest CodeWarrior for ARMv7. If LS1021A Rev. 1 is used, then see the `README.txt` file from the project, for information about debugging a target with this CPU revision.

2. Choose **Run** > **Debug** from CodeWarrior IDE menu bar to start a debug session.
3. Choose **Window** > **Show View** > **Other**. The Show View dialog appears.
4. Expand the **Debug** node, select **Debugger Shell**, and click **OK**. The **Debugger Shell** view appears, as shown in the figure below.

Figure 1. Debugger Shell view



```
CodeWarrior Debugger Shell v1.0
%>
```

In the **Debugger Shell** console, you can run specific commands for programming fuses.

The following sections explain how to perform various fuse programming tasks:

- Programming SFP Configuration Register (SFPCR)
- Programming One Time Programmable Master Key (OTPMK)
- Programming Super Root Key Hash (SRKH)
- Programming Intend to Secure (ITS)
- Programming Debug Challenge/Response Value Register (DCVR/DRVR)
- Programming Write Protect (WP)

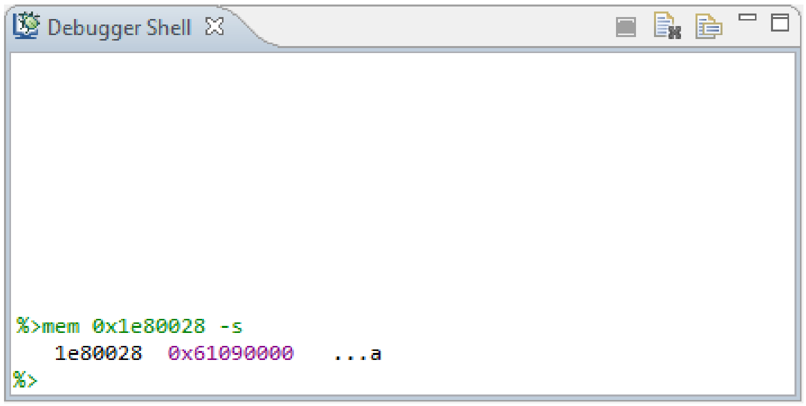
3.1. Programming SFP Configuration Register (SFPCR)

The SFPCR register needs to be configured with appropriate default value for writing fuses on top frequency chips. When performing fuse programming at other platform frequencies, the default value of program pulse width (PPW) must be overwritten prior to writing the Instruction Register.

Follow these steps to program the SFPCR register:

1. Read SFPCR default value, as shown in the figure below.

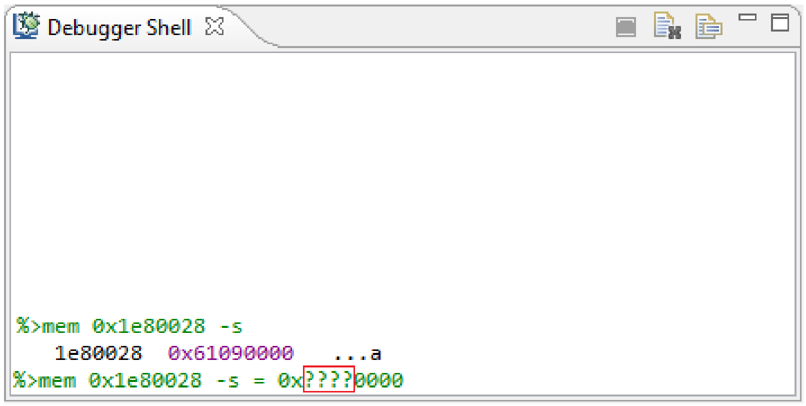
Figure 2. Read SFPCR default value



```
%>mem 0x1e80028 -s
   1e80028  0x61090000   ...a
%>
```

2. Program PPW with the optimal value, as shown in the figure below.

Figure 3. Program PPW



```
%>mem 0x1e80028 -s
   1e80028  0x61090000   ...a
%>mem 0x1e80028 -s = 0x????0000
```

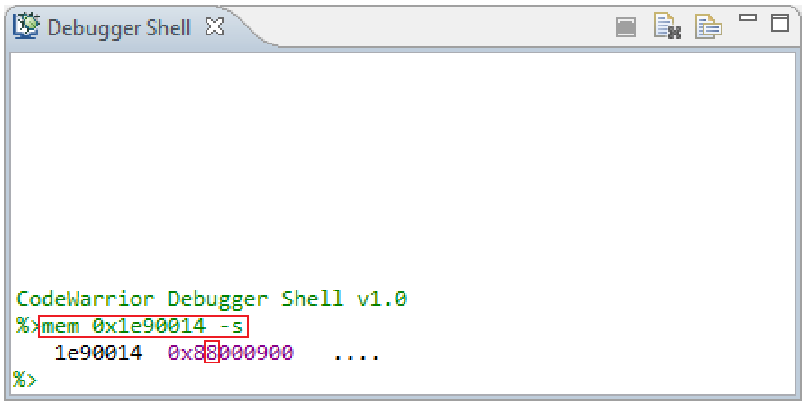
After board reset, you need to program PPW again.

3.2. Programming One Time Programmable Master Key (OTPMK)

Follow these steps to program OTPMK:

1. Check initial state of the SecMon_HP Status register, as shown in the figure below.

Figure 4. Check SecMon_HP Status register initial state



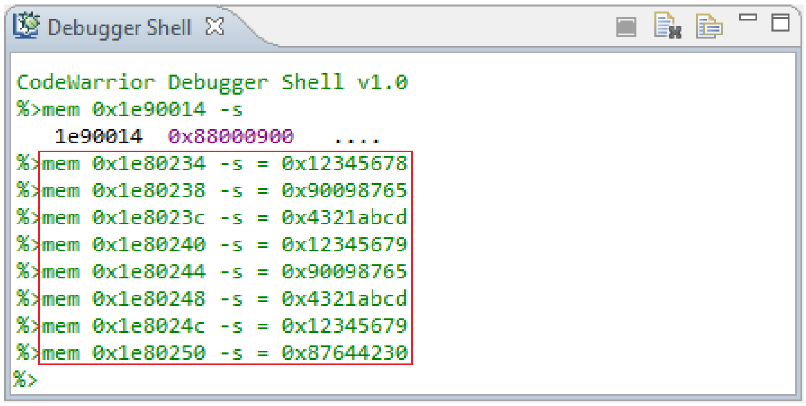
```
CodeWarrior Debugger Shell v1.0
%>mem 0x1e90014 -s
   1e90014  0x88000900   ....
%>
```

OTPMK_ZERO = 1 indicates that OTPMK is not programmed in the fuse bank.

2. Program OTPMK in the fuse bank. If the generated OTPMK key is: *1234567890098765 4321abcd1234567890098765 4321abcd1234567887654321*, then write the key into the eight OTPMK mirror registers, as follows:

```
OTPMKR_0 = 12345678
OTPMKR_1 = 90098765
OTPMKR_2 = 4321abcd
OTPMKR_3 = 12345679
OTPMKR_4 = 90098765
OTPMKR_5 = 4321abcd
OTPMKR_6 = 12345679
OTPMKR_7 = 87644230
```

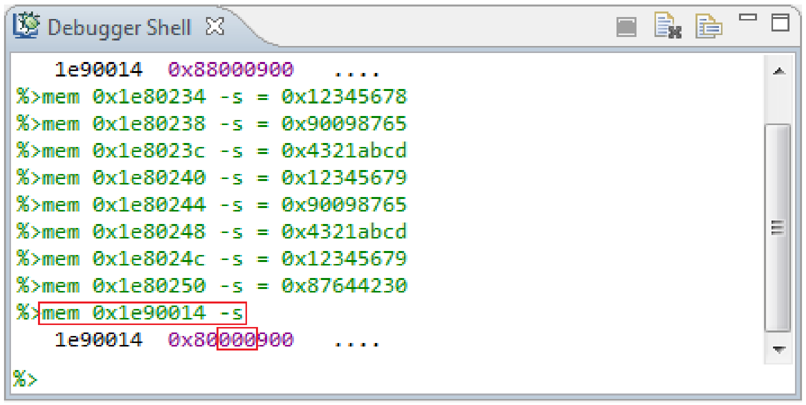
Figure 5. Write OTPMK key into mirror registers



```
CodeWarrior Debugger Shell v1.0
%>mem 0x1e90014 -s
   1e90014  0x88000900   ....
%>mem 0x1e80234 -s = 0x12345678
%>mem 0x1e80238 -s = 0x90098765
%>mem 0x1e8023c -s = 0x4321abcd
%>mem 0x1e80240 -s = 0x12345679
%>mem 0x1e80244 -s = 0x90098765
%>mem 0x1e80248 -s = 0x4321abcd
%>mem 0x1e8024c -s = 0x12345679
%>mem 0x1e80250 -s = 0x87644230
%>
```

3. Check SecMon_HP Status register state again, as shown in the figure below. Ensure that there is no parity error.

Figure 6. Check SecMon_HP Status register state

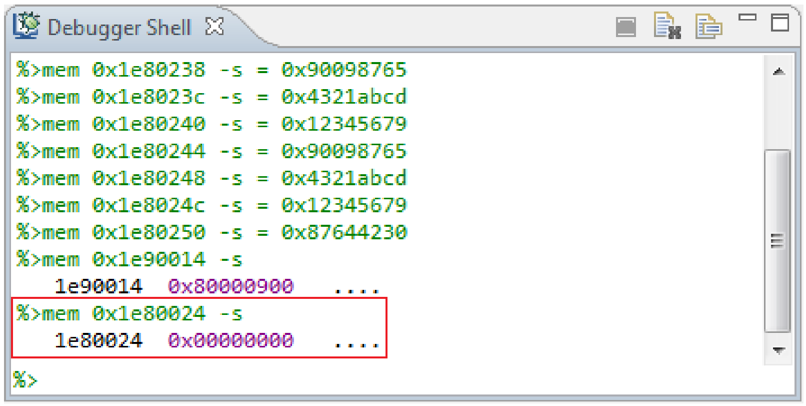


```
   1e90014  0x88000900   ....
%>mem 0x1e80234 -s = 0x12345678
%>mem 0x1e80238 -s = 0x90098765
%>mem 0x1e8023c -s = 0x4321abcd
%>mem 0x1e80240 -s = 0x12345679
%>mem 0x1e80244 -s = 0x90098765
%>mem 0x1e80248 -s = 0x4321abcd
%>mem 0x1e8024c -s = 0x12345679
%>mem 0x1e80250 -s = 0x87644230
%>mem 0x1e90014 -s
   1e90014  0x80000900   ....
%>
```

OTPMK_ZERO = 0 indicates that OTPMK is programmed in the fuse bank. In addition, OTPMK_SYNDROME = 0 (bites marked in red) indicates that there is no parity error.

4. Check Secret Value Hamming Error Status Register (SFP_SVHESR). Any non-zero value read from this register indicates that a hamming code error has been detected.

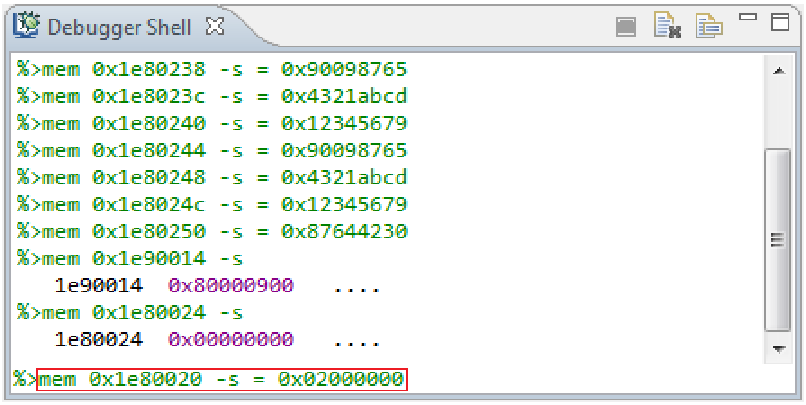
Figure 7. Check SFP_SVHESR register



```
%>mem 0x1e80238 -s = 0x90098765
%>mem 0x1e8023c -s = 0x4321abcd
%>mem 0x1e80240 -s = 0x12345679
%>mem 0x1e80244 -s = 0x90098765
%>mem 0x1e80248 -s = 0x4321abcd
%>mem 0x1e8024c -s = 0x12345679
%>mem 0x1e80250 -s = 0x87644230
%>mem 0x1e90014 -s
   1e90014  0x80000900   ....
%>mem 0x1e80024 -s
   1e80024  0x00000000   ....
%>
```

5. Permanently write data from the mirror registers into the fuse array (PROGFB).

Figure 8. Write data into fuse array

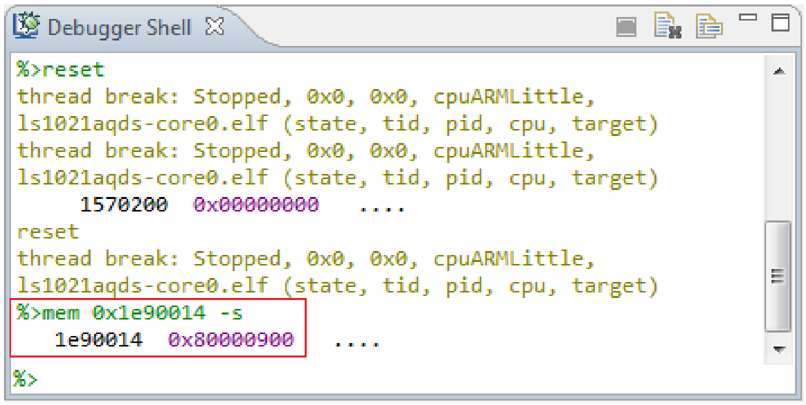


Debugger Shell

```
%>mem 0x1e80238 -s = 0x90098765
%>mem 0x1e8023c -s = 0x4321abcd
%>mem 0x1e80240 -s = 0x12345679
%>mem 0x1e80244 -s = 0x90098765
%>mem 0x1e80248 -s = 0x4321abcd
%>mem 0x1e8024c -s = 0x12345679
%>mem 0x1e80250 -s = 0x87644230
%>mem 0x1e90014 -s
   1e90014  0x80000900   ....
%>mem 0x1e80024 -s
   1e80024  0x00000000   ....
%>mem 0x1e80020 -s = 0x02000000
```

6. Reset and check if OTPMK_ZERO is set to 0.

Figure 9. Reset and check OTPMK_ZERO



Debugger Shell

```
%>reset
thread break: Stopped, 0x0, 0x0, cpuARMLittle,
ls1021aqds-core0.elf (state, tid, pid, cpu, target)
thread break: Stopped, 0x0, 0x0, cpuARMLittle,
ls1021aqds-core0.elf (state, tid, pid, cpu, target)
     1570200  0x00000000   ....
reset
thread break: Stopped, 0x0, 0x0, cpuARMLittle,
ls1021aqds-core0.elf (state, tid, pid, cpu, target)
%>mem 0x1e90014 -s
   1e90014  0x80000900   ....
%>
```

3.3. Programming Super Root Key Hash (SRKH)

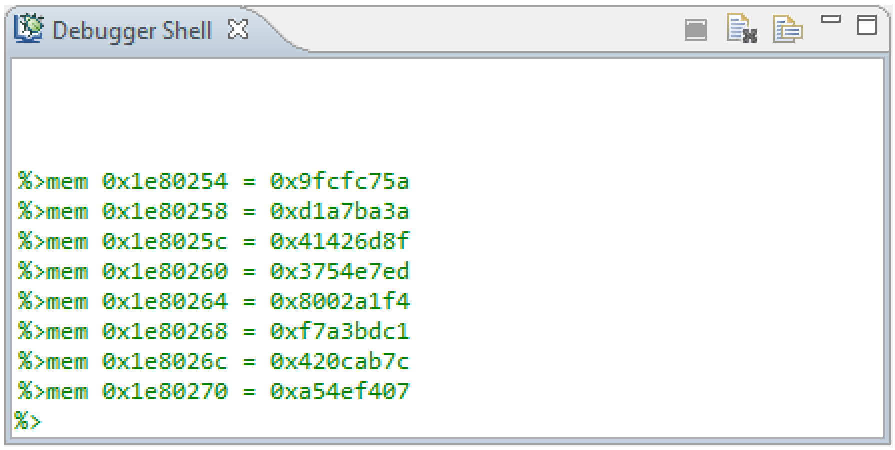
The public key hash obtained while signing images needs to be written in the SRKH registers of the SFP block. If the key hash obtained while signing images is: *9fcfc75ad1a7ba3a41426d8f3754e7ed8002a1f4f7a3bdc1420cab7ca54ef407*, then write the key into the eight SRKH mirror registers, as follows:

```
SRKHR_0 = 9fcfc75a
SRKHR_1 = d1a7ba3a
SRKHR_2 = 41426d8f
SRKHR_3 = 3754e7ed
SRKHR_4 = 8002a1f4
SRKHR_5 = f7a3bdc1
SRKHR_6 = 420cab7c
SRKHR_7 = a54ef407
```

Follow these steps to program SRKH:

1. Write SRKH fuse values into mirror registers. These values must be swapped before writing the SRKH mirror registers. Because the Debugger Shell write operation is done via core, and the core access is little-endian; therefore, using the *-s* option is no longer required.

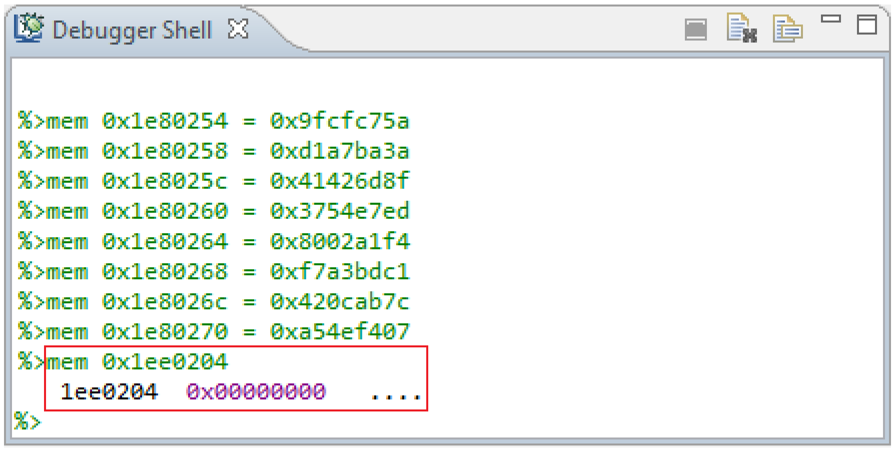
Figure 10. Write SRKH fuse values into mirror registers



```
%>mem 0x1e80254 = 0x9fcfc75a
%>mem 0x1e80258 = 0xd1a7ba3a
%>mem 0x1e8025c = 0x41426d8f
%>mem 0x1e80260 = 0x3754e7ed
%>mem 0x1e80264 = 0x8002a1f4
%>mem 0x1e80268 = 0xf7a3bdc1
%>mem 0x1e8026c = 0x420cab7c
%>mem 0x1e80270 = 0xa54ef407
%>
```

2. Check the status of the Scratch Read/Write Register (DCFG_CCSR_SCRATCHRW2), as shown in the figure below. Any non-zero value read from DCFG_CCSR_SCRATCHRW2 indicates a mismatch in the hash.

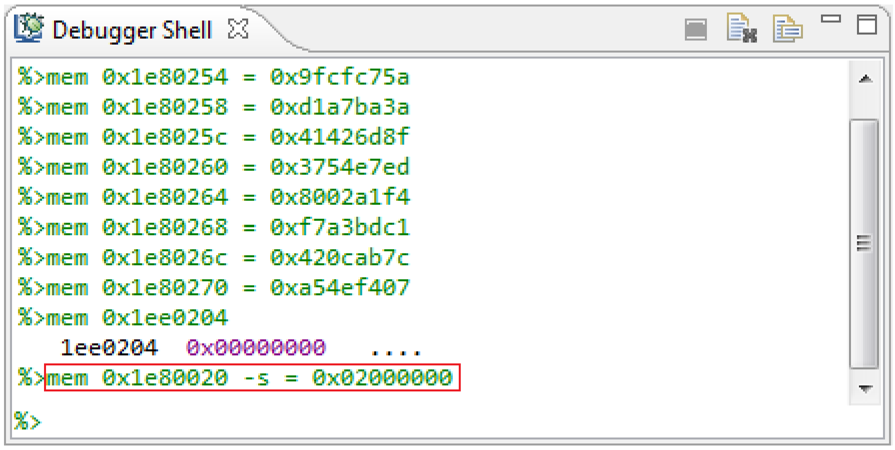
Figure 11. Check Scratch Read/Write Register status



```
%>mem 0x1e80254 = 0x9fcfc75a
%>mem 0x1e80258 = 0xd1a7ba3a
%>mem 0x1e8025c = 0x41426d8f
%>mem 0x1e80260 = 0x3754e7ed
%>mem 0x1e80264 = 0x8002a1f4
%>mem 0x1e80268 = 0xf7a3bdc1
%>mem 0x1e8026c = 0x420cab7c
%>mem 0x1e80270 = 0xa54ef407
%>mem 0x1ee0204
  1ee0204  0x00000000   ....
%>
```

3. Permanently write data from the mirror registers into the fuse array (PROGFB), as shown in the figure below.

Figure 12. Write data into fuse array



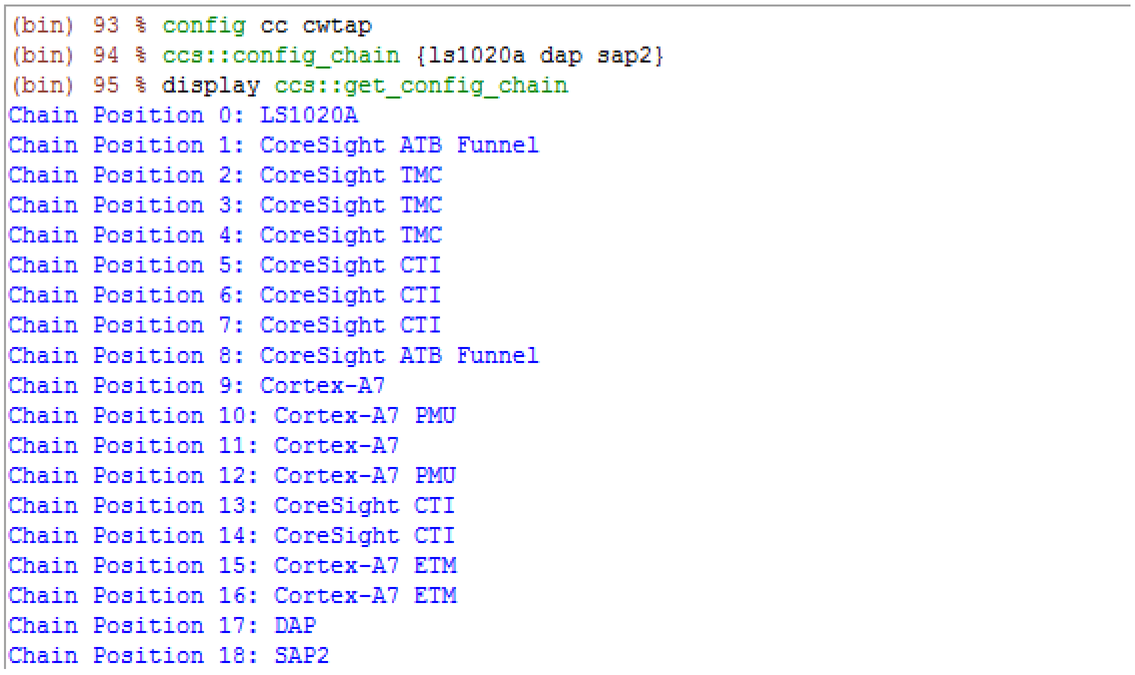
```
%>mem 0x1e80254 = 0x9fcfc75a
%>mem 0x1e80258 = 0xd1a7ba3a
%>mem 0x1e8025c = 0x41426d8f
%>mem 0x1e80260 = 0x3754e7ed
%>mem 0x1e80264 = 0x8002a1f4
%>mem 0x1e80268 = 0xf7a3bdc1
%>mem 0x1e8026c = 0x420cab7c
%>mem 0x1e80270 = 0xa54ef407
%>mem 0x1ee0204
   1ee0204  0x00000000   ....
%>mem 0x1e80020 -s = 0x02000000

%>
```

To avoid writing data permanently into the fuse array for the SRKH mirror registers, follow these steps:

1. Put the core in the boot hold off state at startup. For this, load the secure boot reset configuration word (RCW) with BOOT_HO = 1. You can generate a new RCW using the Pre-Boot Loader (PBL) component of QorIQ Configuration and Validation Suite (QCVS).
2. Open the CodeWarrior connection server (CCS) console and configure a connection to the board, as shown in the figure below.

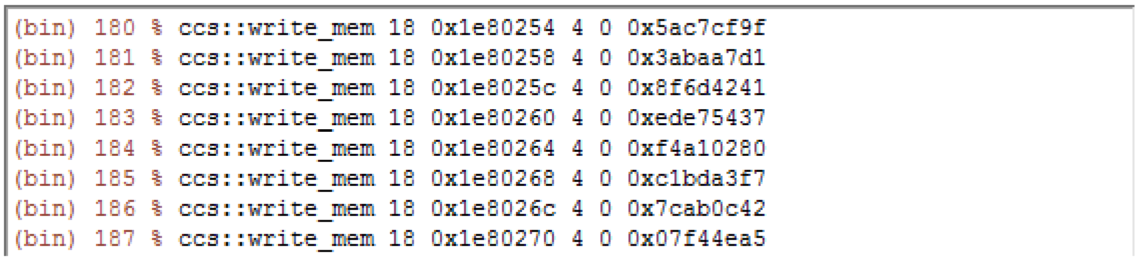
Figure 13. Configure a connection in CCS console



```
(bin) 93 % config cc cwtap
(bin) 94 % ccs::config_chain {ls1020a dap sap2}
(bin) 95 % display ccs::get_config_chain
Chain Position 0: LS1020A
Chain Position 1: CoreSight ATB Funnel
Chain Position 2: CoreSight TMC
Chain Position 3: CoreSight TMC
Chain Position 4: CoreSight TMC
Chain Position 5: CoreSight CTI
Chain Position 6: CoreSight CTI
Chain Position 7: CoreSight CTI
Chain Position 8: CoreSight ATB Funnel
Chain Position 9: Cortex-A7
Chain Position 10: Cortex-A7 PMU
Chain Position 11: Cortex-A7
Chain Position 12: Cortex-A7 PMU
Chain Position 13: CoreSight CTI
Chain Position 14: CoreSight CTI
Chain Position 15: Cortex-A7 ETM
Chain Position 16: Cortex-A7 ETM
Chain Position 17: DAP
Chain Position 18: SAP2
```

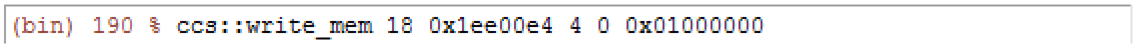
3. Write SRKH fuse values into mirror registers. These values must be swapped before writing the SRKH mirror registers.

Figure 14. Write SRKH fuse values into mirror registers



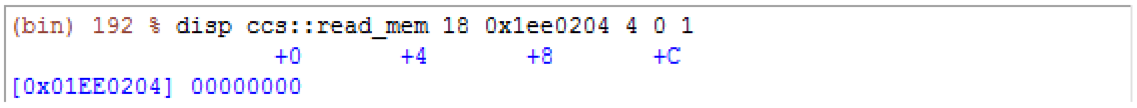
```
(bin) 180 % ccs::write_mem 18 0x1e80254 4 0 0x5ac7cf9f
(bin) 181 % ccs::write_mem 18 0x1e80258 4 0 0x3abaa7d1
(bin) 182 % ccs::write_mem 18 0x1e8025c 4 0 0x8f6d4241
(bin) 183 % ccs::write_mem 18 0x1e80260 4 0 0xede75437
(bin) 184 % ccs::write_mem 18 0x1e80264 4 0 0xf4a10280
(bin) 185 % ccs::write_mem 18 0x1e80268 4 0 0xc1bda3f7
(bin) 186 % ccs::write_mem 18 0x1e8026c 4 0 0x7cab0c42
(bin) 187 % ccs::write_mem 18 0x1e80270 4 0 0x07f44ea5
```

4. Get the core out of the boot hold off state.



```
(bin) 190 % ccs::write_mem 18 0x1ee00e4 4 0 0x01000000
```

5. Check the status of the Scratch Read/Write Register (DCFG_CCSR_SCRATCHRW2). Any non-zero value read from DCFG_CCSR_SCRATCHRW2 indicates a mismatch in the hash.



```
(bin) 192 % disp ccs::read_mem 18 0x1ee0204 4 0 1
                   +0       +4       +8       +C
[0x01EE0204] 00000000
```

NOTE You are recommended to use SAP2 access to write SRKH fuse values into mirror registers. DAP access is also available but it is little-endian, whereas SAP2 access is big-endian.

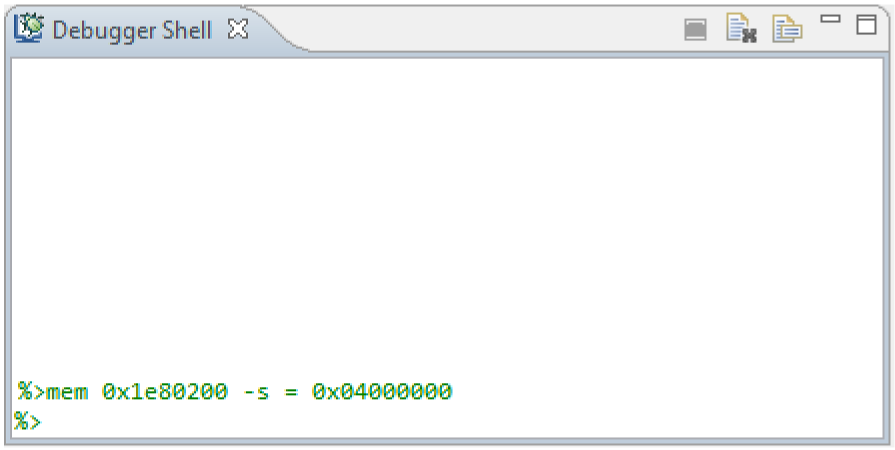
3.4. Programming Intend to Secure (ITS)

The ITS bit signifies the intensions of the original equipment manufacturer (OEM) to make the system secure. This is part of OEM Security Policy Register (SFP_OSPR).

Follow these steps to program ITS:

1. Set ITS, as shown in the figure below.

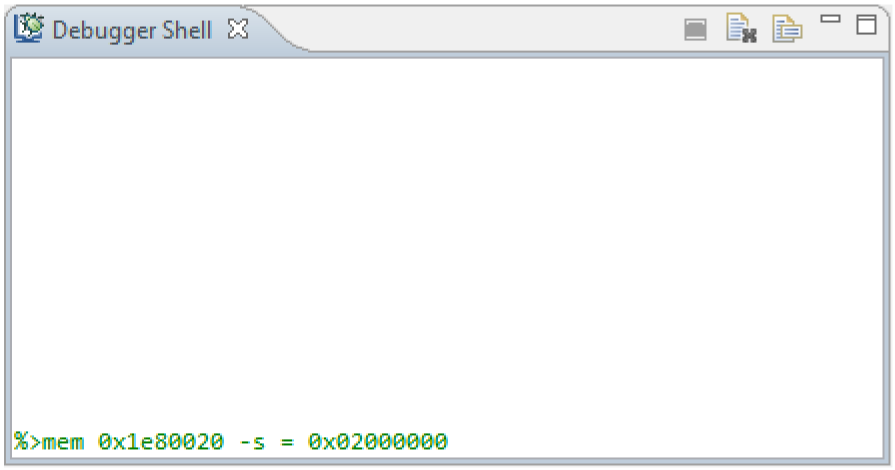
Figure 15. Set ITS



```
%>mem 0x1e80200 -s = 0x04000000
%>
```

2. Permanently write data from the mirror registers into the fuse array (PROGFB), as shown in the figure below.

Figure 16. Write data into fuse array



```
%>mem 0x1e80020 -s = 0x02000000
```

3.5. Programming Debug Challenge/Response Value Register (DCVR/DRVR)

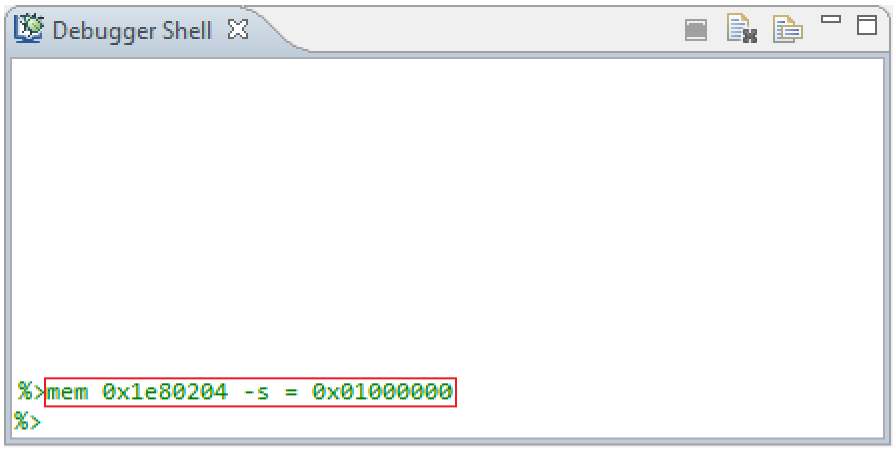
It is mandatory to program DCVR and DRVR fuses to activate secure debug. Original 64-bit value for DCVR is *c0c0c0c0d0d0d0d0* and if DRVR key is *f1f1f1f1a5a5a5a5*, then this needs to be written into the four DRVR mirror registers, as follows:

```
DCVR_0 = d0d0d0d0
DCVR_1 = c0c0c0c0
DRVR_0 = a5a5a5a5
DRVR_1 = f1f1f1f1
```

Follow these steps to program DCVR/DRVR:

1. Set Debug Level (DBLEV) to "001, conditionally open via challenge response, without notification".

Figure 17. Set Debug Level

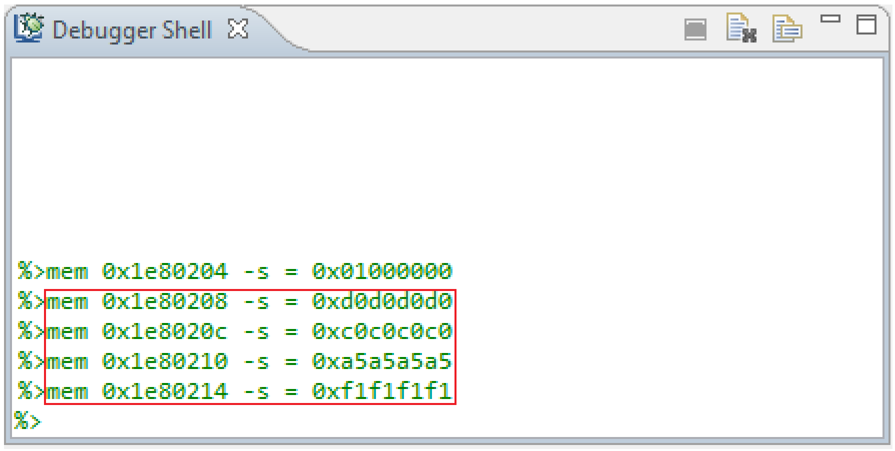


```
%>mem 0x1e80204 -s = 0x01000000
%>
```

NOTE The Debug Challenge Value Register and Debug Response Value Register are intended to be used only when debug permissions are set to one of the conditional access modes.
CodeWarrior only handles the "Debug Level set to 001, conditionally open via challenge response, without notification" conditional access mode.

2. Write DCVR and DRVR values into mirror registers, as shown in the figure below.

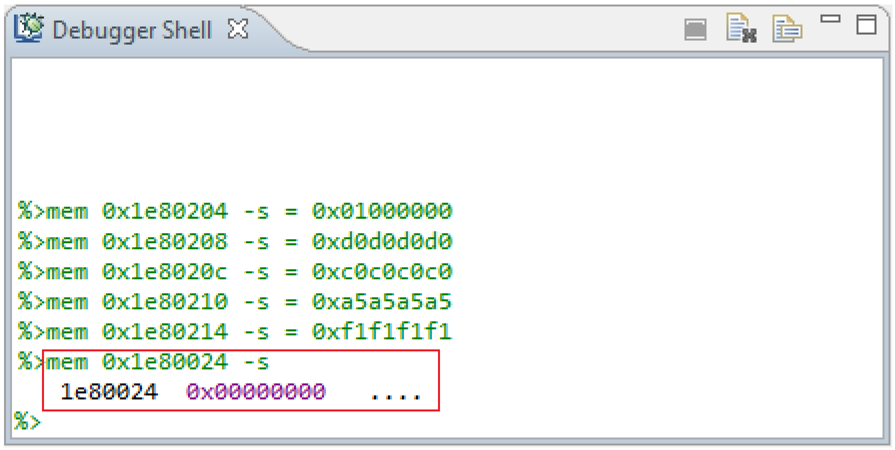
Figure 18. Write DCVR and DRVR values into mirror registers



```
%>mem 0x1e80204 -s = 0x01000000
%>mem 0x1e80208 -s = 0xd0d0d0d0
%>mem 0x1e8020c -s = 0xc0c0c0c0
%>mem 0x1e80210 -s = 0xa5a5a5a5
%>mem 0x1e80214 -s = 0xf1f1f1f1
%>
```

3. Check Secret Value Hamming Error Status Register (SFP_SVHESR). Any non-zero value read from this register indicates that a hamming code error has been detected.

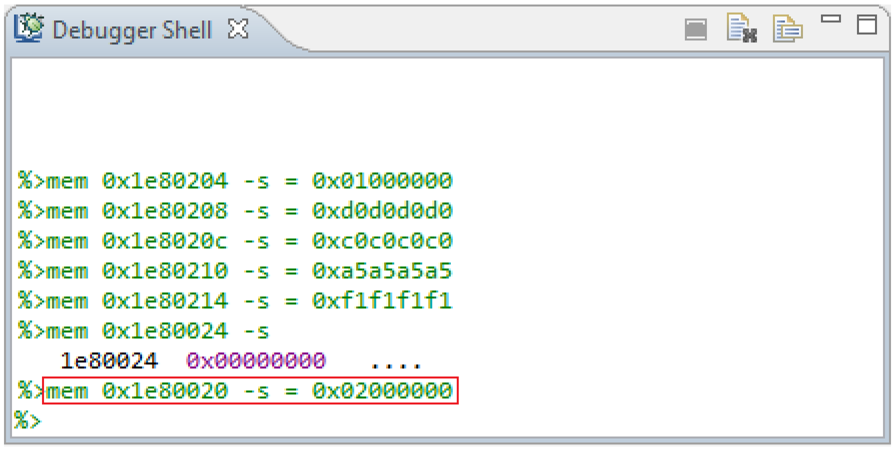
Figure 19. Check SFP_SVHESR register



```
%>mem 0x1e80204 -s = 0x01000000
%>mem 0x1e80208 -s = 0xd0d0d0d0
%>mem 0x1e8020c -s = 0xc0c0c0c0
%>mem 0x1e80210 -s = 0xa5a5a5a5
%>mem 0x1e80214 -s = 0xf1f1f1f1
%>mem 0x1e80024 -s
  1e80024  0x00000000   ....
%>
```

3. Permanently write data from the mirror registers into the fuse array (PROGFB), as shown in the figure below.

Figure 20. Write data into fuse array



```
%>mem 0x1e80204 -s = 0x01000000
%>mem 0x1e80208 -s = 0xd0d0d0d0
%>mem 0x1e8020c -s = 0xc0c0c0c0
%>mem 0x1e80210 -s = 0xa5a5a5a5
%>mem 0x1e80214 -s = 0xf1f1f1f1
%>mem 0x1e80024 -s
   1e80024  0x00000000   ....
%>mem 0x1e80020 -s = 0x02000000
%>
```

NOTE Secure debug requires you to set the Secure Access register, CSU_SA0, to 0x00000100. Configuring a register is possible using the `pbi` command; however, Central Security Unit (CSU) registers cannot be accessed from PBL, during Secure Boot. Therefore, you need to configure the CSU_SA0 register in U-Boot.

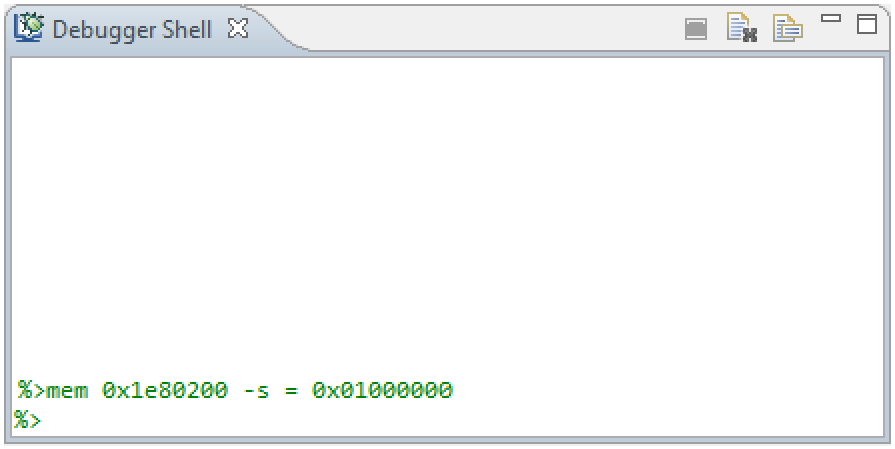
3.6. Programming Write Protect (WP)

Setting WP bit prevents further writes to the mirror registers of the OEM section, until the next system-on-chip (SoC) reset. After the WP fuse is programmed, writes to mirror registers and further programming of the fuse block is permanently disabled.

Follow these steps to program WP:

1. Set WP, as shown in the figure below.

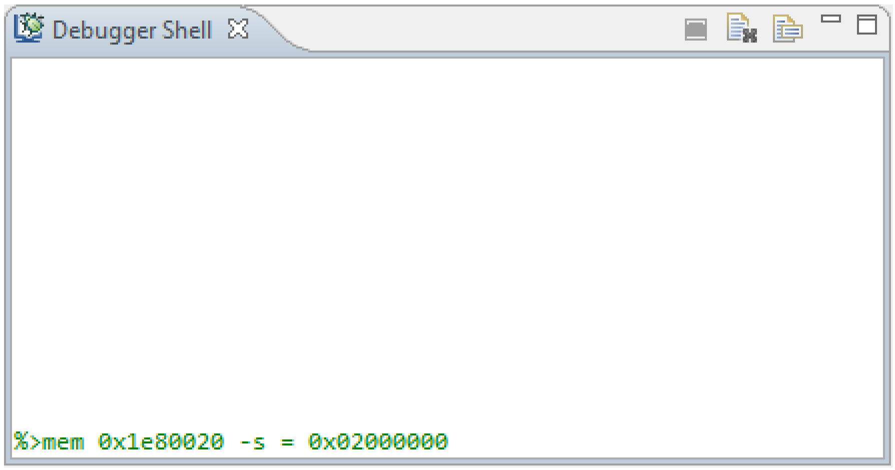
Figure 21. Set WP



```
%>mem 0x1e80200 -s = 0x01000000
%>
```

2. Permanently write data from the mirror registers into the fuse array (PROGFB).

Figure 22. Write data into fuse array



```
%>mem 0x1e80020 -s = 0x02000000
```

NOTE Ensure that WP is programmed in the end.

After fuses have been burned, a pop-up window appears in CodeWarrior asking for Secure Debug key.

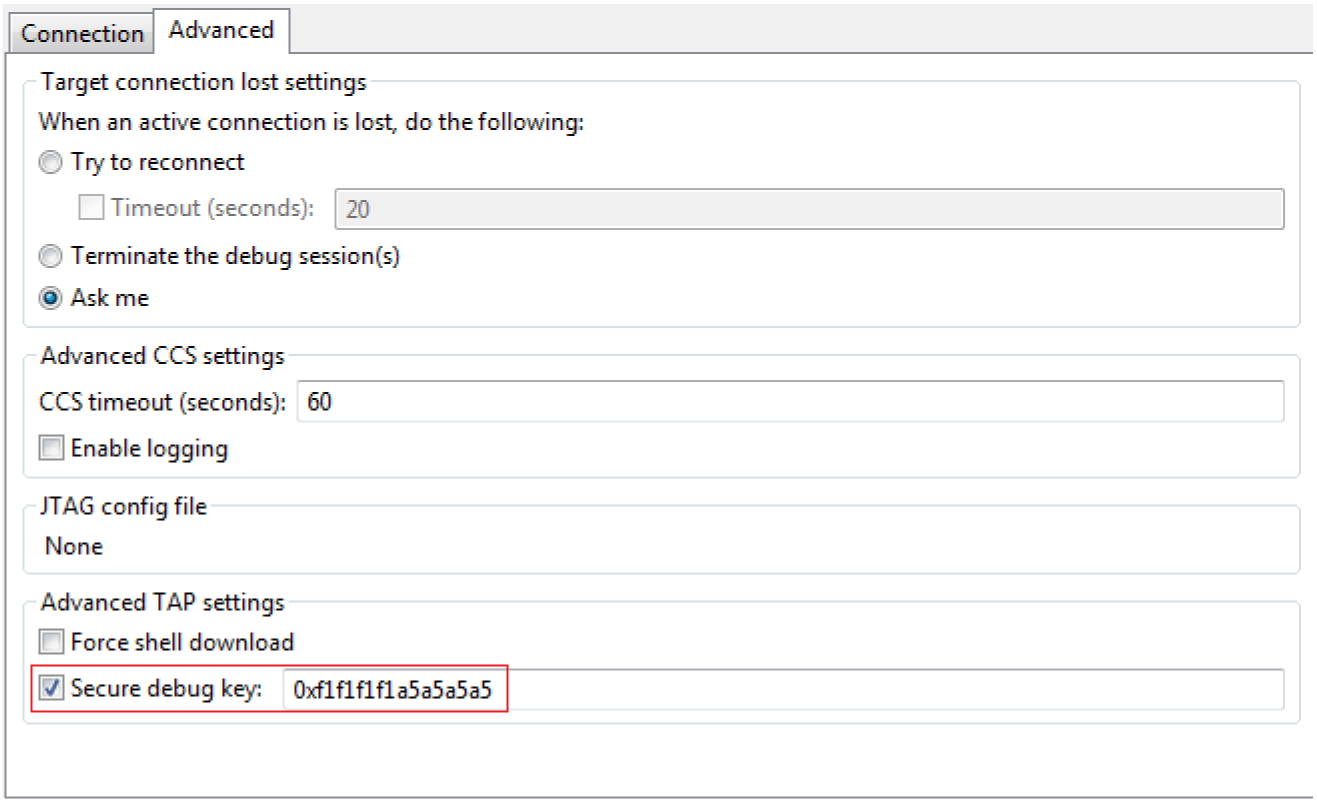
NOTE You need to use the *–s* option with the *mem* command because memory access is little-endian, by default, and it needs to be changed to big-endian.
All commands issued in Debugger Shell are scriptable.

4. Secure debugging using CodeWarrior

To use CodeWarrior for debugging a target with a Debug Secure LS1 CPU, perform these steps:

1. Choose **Run** > **Debug Configurations** from the CodeWarrior IDE menu bar. The **Debug Configurations** dialog appears.
2. Click **Edit** next to the **Connection** menu in the **Target Settings** panel. The **Properties for <*connection launch configuration*>** window appears.
3. Click **Advanced**, select **Secure debug key** option, and enter the key.

Figure 23. Enter secure debug key



Target connection lost settings

When an active connection is lost, do the following:

- ○ Try to reconnect
  - ☐ Timeout (seconds): 20
- ○ Terminate the debug session(s)
- ● Ask me

Advanced CCS settings

CCS timeout (seconds): 60

☐ Enable logging

JTAG config file

None

Advanced TAP settings

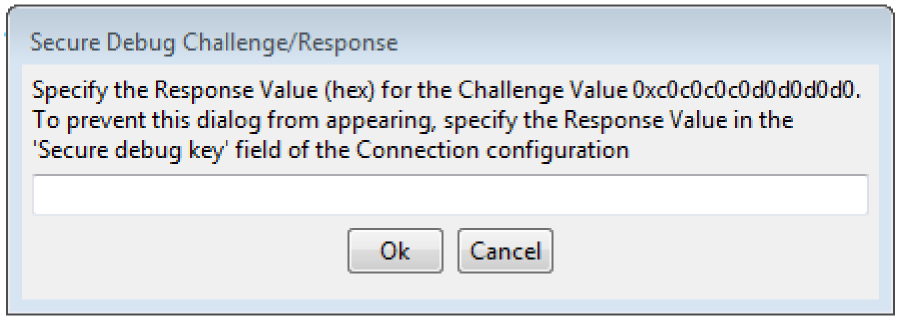
☐ Force shell download

☑ Secure debug key: 0xf1f1f1f1a5a5a5a5

4. Click **OK** to close the **Properties for <*connection launch configuration*>** window.
5. Click **Debug** in the **Debug Configurations** dialog to debug the project.

If you start a debug session without entering the secure debug key as mentioned in the steps above, then a pop-up window appears asking for secure debug key, as shown in the figure below.

Figure 24. Pop-up window asking for secure debug key



Specify the Response Value (hex) for the Challenge Value 0xc0c0c0c0d0d0d0d0.
To prevent this dialog from appearing, specify the Response Value in the 'Secure debug key' field of the Connection configuration

5. Board bring-up and board recovery

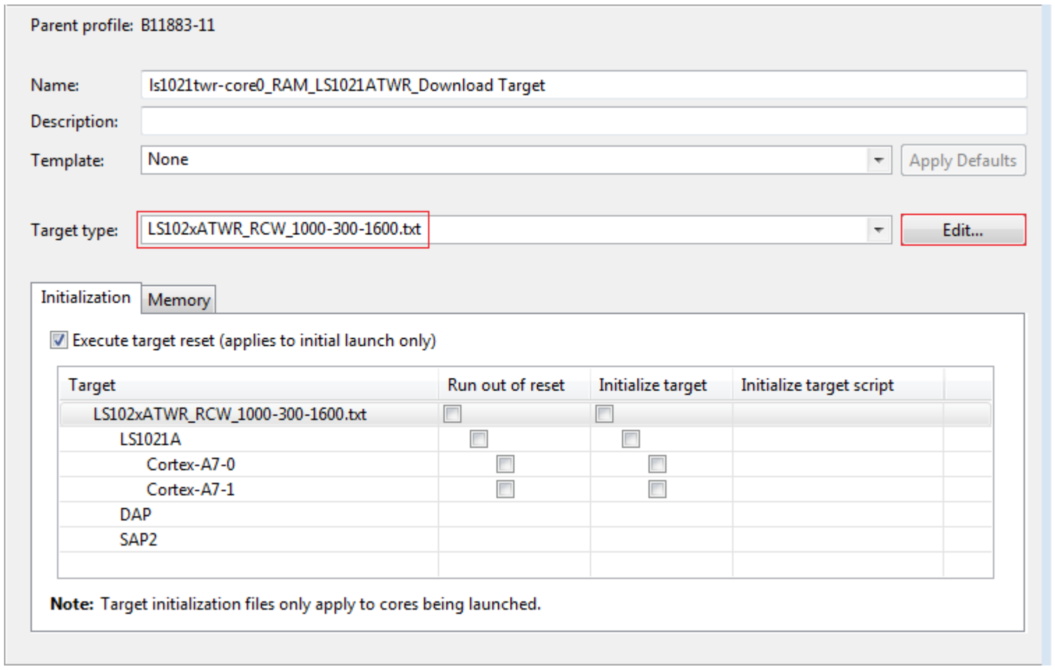
RCW is required to configure board connection and run commands for burning fuses. If the flash is empty and hardcoded RCW cannot be set for the board, then you should use the CodeWarrior RCW override feature to recover the board.

Due to the missing RCW, the double data rate (DDR) memory may not be functional, and an invalid memory error may occur while starting a debug session.

Perform these steps if starting with an empty flash or board recovery is needed:

1. Create a CodeWarrior for ARMv7 bareboard project. If the DDR memory is not functional, then create the project by selecting **Download OCRAM** as the launch configuration.
2. Choose **Run > Debug Configurations** from the CodeWarrior IDE menu bar. The **Debug Configurations** dialog appears.
3. Click **Edit** next to the **Connection** menu in the **Target Settings** panel. The **Properties for** ***<connection launch configuration>*** window appears.
4. Click **Edit** next to the **Target** menu. The **Properties for** ***<connection launch configuration>*** **Target** window appears.
5. Specify a JTAG configuration file for RCW override in the **Target type** menu by clicking the **Edit** button next to this menu, as shown in the figure below.

Figure 25. Select JTAG configuration file for RCW override


Parent profile: B11883-11

Name: ls1021twr-core0_RAM_LS1021ATWR_Download Target

Description:

Template: None

Target type: LS102xATWR_RCW_1000-300-1600.txt

☑ Execute target reset (applies to initial launch only)

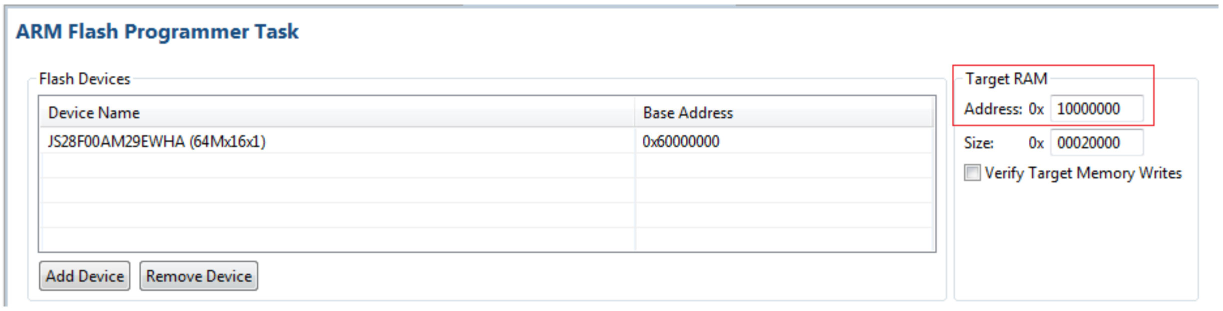
Target	Run out of reset	Initialize target	Initialize target script
LS102xATWR_RCW_1000-300-1600.txt	☐	☐	
LS1021A	☐	☐	
Cortex-A7-0	☐	☐	
Cortex-A7-1	☐	☐	
DAP			
SAP2			

Note: Target initialization files only apply to cores being launched.

NOTE The JTAG configuration file for RCW override contains valid values for RCW registers and allows CodeWarrior to connect to the board.

6. Select OCRAM file as the initialization file for the new target type on the **Initialization** page.
7. Select memory initialization file for the target type on the **Memory** page.
8. Click **OK** to close the **Properties for *<connection launch configuration>* Target** window.
9. Click **OK** to close the **Properties for *<connection launch configuration>*** window.
10. Click **Debug** in the **Debug Configurations** dialog to debug the project.
11. Import target task specific to the flash device.
12. Open the flash programmer target task in the **ARM Flash Programmer Task** editor window and change the running address of the algorithm with the OCRAM address of the processor, as shown in the figure below.

Figure 26. ARM Flash Programmer Task editor window


ARM Flash Programmer Task

Flash Devices

Device Name	Base Address
JS28F00AM29EWHA (64Mx16x1)	0x60000000

Target RAM

Address: 0x 10000000

Size: 0x 00020000

☐ Verify Target Memory Writes

13. Use the **Program/Verify Action** option of the **Add Action** menu available in the **ARM Flash Programmer Task** editor window to write RCW and U-Boot images into the flash device. For

more information, see Programming eMMC/SD Card using CodeWarrior for ARMv7 Application Note (AN5184)

NOTE Ensure to select the **Erase sectors before program** checkbox and apply correct address offset.

14. Execute the target task.

After the flash device has been programmed successfully with the U-Boot image, the board is recovered and is ready for use.

NOTE Flash programmer procedure is scriptable and therefore, you can run it from Debugger Shell.

The procedure mentioned above allows you to program all board images, including the header files, using CodeWarrior.

Document Number: AN5227

18 January 2016