

SBL User Guide

SBL (Secondary Boot Loader)

User Guide

Copyright © 2015 Texas Instruments Incorporated. All rights reserved.

Information in this document is subject to change without notice. Texas Instruments may have pending patent applications, trademarks, copyrights, or other intellectual property rights covering matter in this document. The furnishing of this documents is given for usage with Texas Instruments products only and does not give you any license to the intellectual property that might be contained within this document. Texas Instruments makes no implied or expressed warranties in this document and is not responsible for the products based from this document.

1 Table of Contents

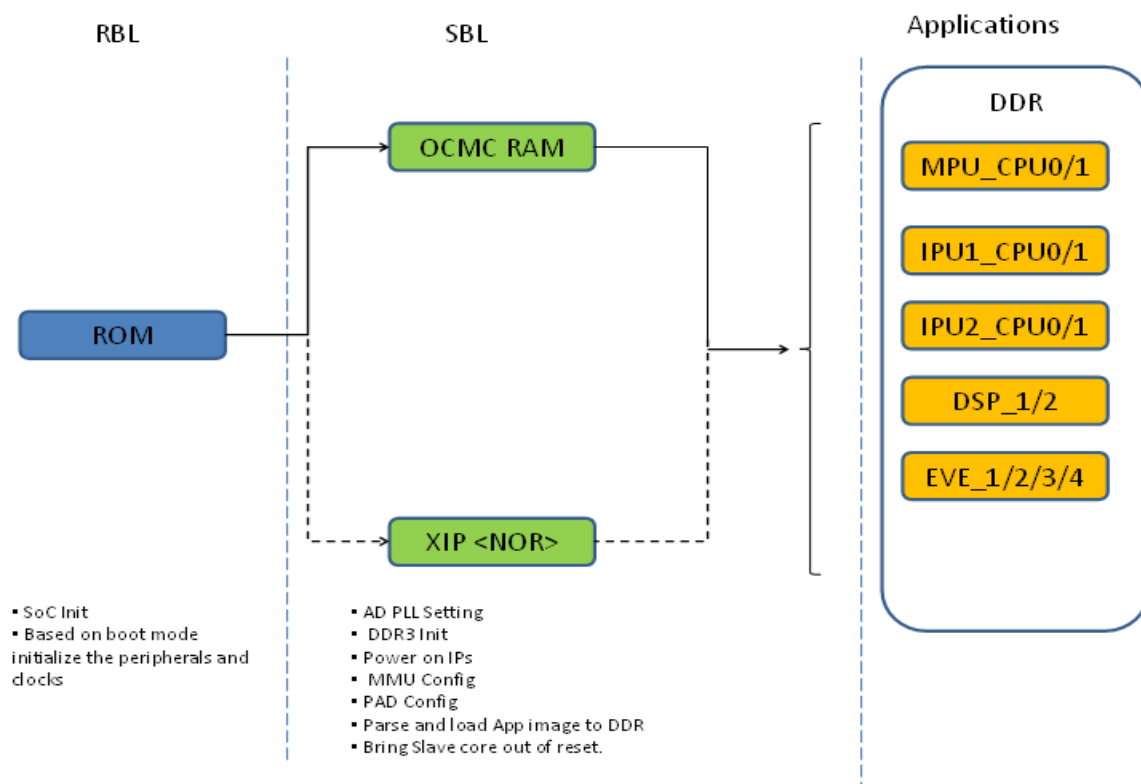
1	Table of Contents	2
1.	Introduction.....	4
2	SBL Supported Devices	4
3	Image Formats.....	4
3.1	SBL Bootloader Image	5
3.2	CH Image	6
3.3	Application Image.....	6
3.3.1	TDA3xx SBL	9
4	Building the SBL.....	10
5	Boot Modes of SBL	10
5.1	Flash Tools	12
5.1.1	QSPI Flash Writer	12
5.1.2	NOR Flash Writer.....	12
5.2	QSPI boot mode.....	12
5.3	NOR Boot Mode	13
5.4	MMCSD Boot Mode.....	14
5.4.1	Using PC tool.....	14
5.4.2	Option 2: Steps to prepare a bootable SD card using DISKPART	14
5.5	QSPI_SD Boot Mode	15
6	Build Mode	15
7	EMIFMODE	16
8	Multi OPP Support.....	16
9	SBL Optimization Level	17
10	WatchDog Timer2 (WD_TIMER2).....	17

11	DSP Boot address alignment	18
12	AD PLL Clock Frequency	18
13	Boot-Up time	18
13.1	TDA2xx Device	18
13.1.1	QSPI Boot mode:	18
13.1.2	NOR Boot mode:.....	18
13.1.3	SD Boot mode:.....	19
13.2	TDA2Ex Device.....	19
13.3	TDA3xx Device	20
14	Component Folder.....	22
15	UART Console	22
16	Memory Usage	24
17	Prebuilt Binaries	25
18	Multi core Image generation script.....	26
19	SBL Multicore Mailbox app.....	26
20	Regression Support	26
21	Known Issues	26

1. Introduction

The Secondary Bootloader (SBL) initializes the execution environment for multi-core application. It sets-up the AD PLL clock to values specified in TDA2xx, TDA2Ex or TDA3xx datasheet, powers on the I/O Peripherals, initializes the DDR, loads the application image into DDR & brings the slave cores out of reset.

SBL supports three boot modes: QSPI, MMCSD & NOR for TDA2xx & TDA2Ex devices and QSPI, QSPI_SD & NOR for TDA3xx device. It copies the Multicore App Image from storage memory device, parses it & loads the executables into DDR.



2 SBL Supported Devices

This release of SBL supports the TDA2xx, TDA2Ex and TDA3xx devices.

3 Image Formats

SBL Image Format:

In non-XIP boot mode, RBL expects the GP header to execute SBL and in XIP boot mode, it expects the SBL image in bin format. SBL image along with GP header is referred as tiimage. For more information on tiimage, refer to the section 3.1 of this userguide.

Application Image Format:

AppImage generation is two-step process: RPRC format conversion & Multicore Image file generation.

- i) Firstly, application executable has to be converted from ELF/COFF format to custom TI RPRC image format. For more information on RPRC format, refer to the section 3.3.1 of this user guide.
- ii) AppImage is a Multicore image file that includes the RPRC image file of individual cores. It contains the CPU ID & boot-up sequence. For more information on Multicore Image format, refer to the section 3.3.2 of this user guide.

3.1 SBL Bootloader Image

The Boot loader Image should be converted as given in this section:

1. Convert the ELF image to binary.

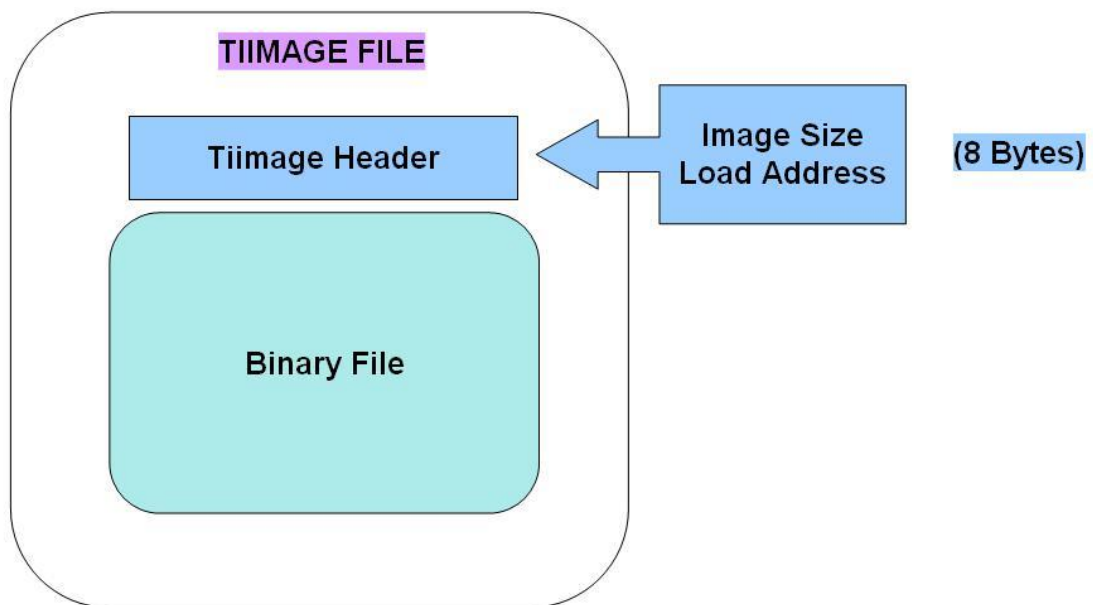
SBL should be changed into binary format using the following command:

```
$ arm-none-eabi-objcopy.exe --gap-fill=0xff -O binary SBL.bin SBL.bin
```

arm-none-eabi-objcopy.exe is part of Linaro or Code-sourcery tool chain and is not provided as tool in Starterware package.

2. Convert the binary file to tiimage file.

RBL expects the secondary bootloader with GP header only in non-XIP boot mode. The tiimage format is as given below:



Using following command convert the generated bin file to tiimage:

```
$tiimage.exe <Load Address> <ENDIAN> SBL.bin SBL.tiimage
```

SBL sections should be mapped to OCMC region using the linker command file while building the SBL.

Load Address for SBL is 0x4030 0000 for TDA2xx/TDA2Ex whereas on TDA3xx this address is 0x0030 0000 as RBL already configures AMMU mapping of medium pages.

ENDIAN (BE/LE) specifies whether the TI header and the binary is in Big Endian/Little Endian.
For SPI/QSPI boot mode, the RBL expects the SBL in Big Endian.
For other boot modes (SD), the RBL expects the SBL in Little Endian.

tiimage.exe is provided as part of the starterware package in tools folder (bootloader/Tools/tiimage).

3.2 CH Image

For fast boot on tda3xx platform in QSPI and QSPI_SD boot mode, RBL can also boot another type of image called chimage (Configuration Header Image). RBL sets up QSPI speed as 48 MHz by default. On adding the CH header to tiimage we can change the QSPI speed. The configuration header is prepended to tiimage to create chimage. The size of header is 0x200 bytes.

Boot loader Image should be converted to chimage using below command:

```
$chimage.exe CH.bin SBL.tiimage SBL.chimage
```

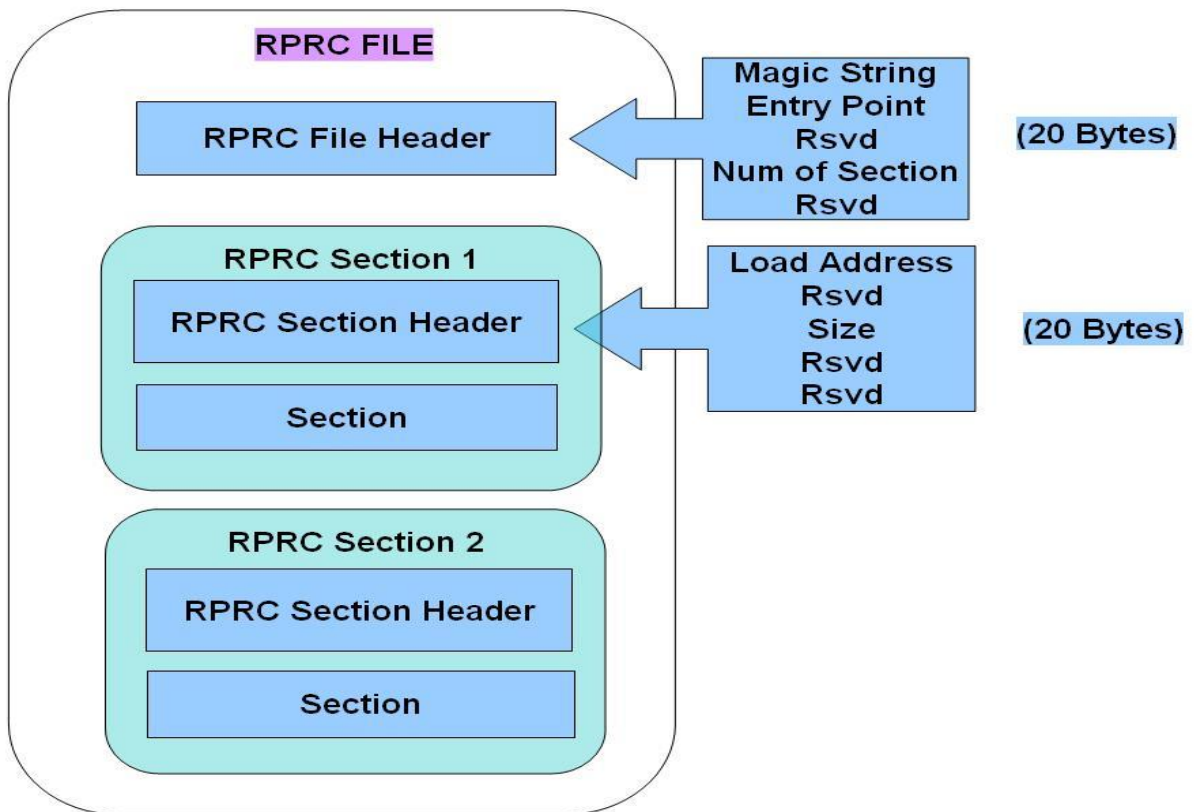
Chimage.exe tool and the CH bin for tda3xx (tda3x_chqspi_clock64mhz.bin) are present in folder bootloader\Tools\chimage.

3.3 Application Image

The Application Image should be converted as given in this section. The Application Images of all the Cores should be built in ELF/Coff format and converted to a single file called Multi core Image file.

1. Convert ELF to RPRC format

RPRC Image Format is as shown below:



As shown in above diagram, RPRC file has one file header and multiple sections. File Header contains Magic string for sanity check of the image, Entry Point, Number of Sections and One reserved word.

File header is followed by multiple Sections. Each Section has section header and section data. The Section Header has five words: Load address, one reserved word, size and two reserved words. Address specifies the destination address of that section and size specifies section size in bytes of the Section Data that follows. The whole section has to be copied to the load address. RPRC sections are generated for each elf section as part of the elf executable.

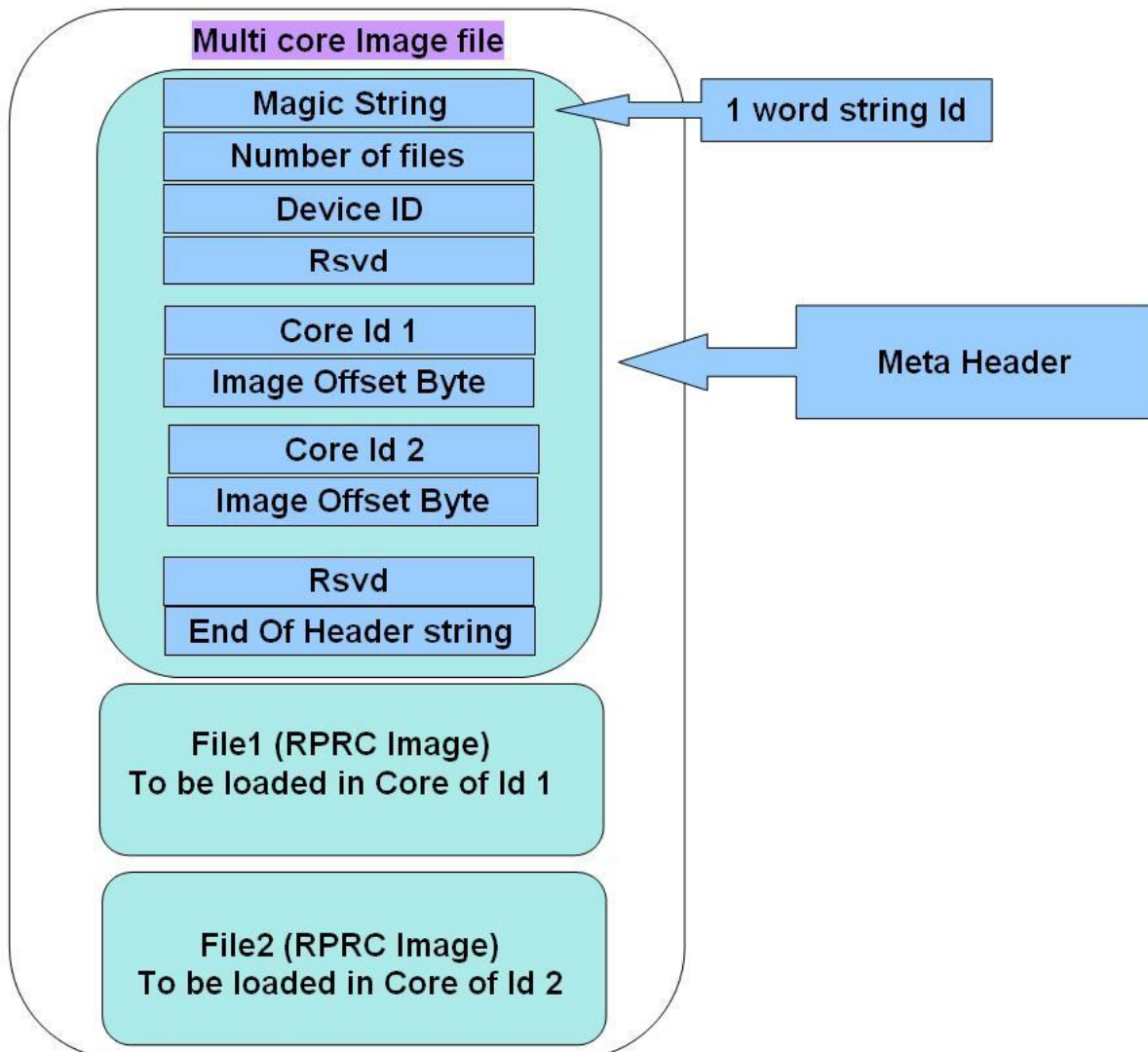
Convert the different application image files (ELF/COFF) for various cores to RPRC image using the following command:

```
$out2rprc.exe <App_In_name(elf or coff)> <App_out_name>
```

out2rprc.exe is provided in tools as part of the starterware package(bootloader/Tools/out2rprc).

2. Multi core Image generation from the application RPRC images.

Multi core image format is as shown below:



Meta Header is of variable length and depends on the number of files included. First word is a magic string for sanity check of the image. MSTR in ASCII will be used as the magic string. Second word is the number of RPRC image files included in the generated multi core image file. Third word is the Device Id followed by a reserved word. After this, follows the two word structure of Core Id and the offset for RPRC Image i.e. the starting byte of RPRC image. For a given structure, RPRC Image starting at the given offset will be loaded on the Core mentioned as Core Id.

Core IDs for TDA2xx device:

MPU_CPU0_ID (Master)	0
MPU_CPU1_ID	1
IPU1_CPU0_ID	2
IPU1_CPU1_ID	3
IPU1_CPU_SMP_ID	4
IPU2_CPU0_ID	5
IPU2_CPU1_ID	6
IPU2_CPU_SMP_ID	7
DSP1_ID	8
DSP2_ID	9
EVE1_ID	10
EVE2_ID	11
EVE3_ID	12
EVE4_ID	13

Core IDs for TDA2Ex device:

MPU_CPU0_ID (Master)	0
IPU1_CPU0_ID	2
IPU1_CPU1_ID	3
IPU1_CPU_SMP_ID	4
DSP1_ID	8

Core Id for TDA3xx device:

CORE_ID_IPU1_CPU0	2
CORE_ID_IPU1_CPU1	3
CORE_ID_DSP1	6
CORE_ID_DSP2	7
CORE_ID_EVE1	8
CORE_ID_IPU1	14

There is a reserved word after the Core Id and the offset values, it should be used for calculating Meta header CRC to validate header before parsing the RPRC images. String MEND in ASCII is used as last word at the end of header string.

Create the Multicore Image from all the RPRC images using the following command.

```
$ MulticoreImageGen.exe <ENDIAN> <Dev Id> <App out file> <Core Id 1> <RPRC in file for Core Id 1>
[<Core Id n> <RPRC in file for Core Id n> ...]
```

ENDIAN (BE/LE): specifies whether the out file is in Big Endian/Little Endian format. For QSPI/SPI boot mode application image should be in BE, for other boot modes application image should be in LE.

Dev Id: This is the device Id. Still the Device Ids of the supported devices are not populated & random values are chosen. In the present implementation of the SBL, only warning is printed on UART console if the device id doesn't match and the boot is not aborted.

App out file: This is the name of the multicore image file to be created.

Core Id: This is the Id of the core as given in the table above to be followed by the *RPRC image file name* to be loaded in that core. The order of the core Id and RPRC file pair given while generating the file determines the order in which the slave cores are booted. If the image is SMP, please select the SMP core-id.

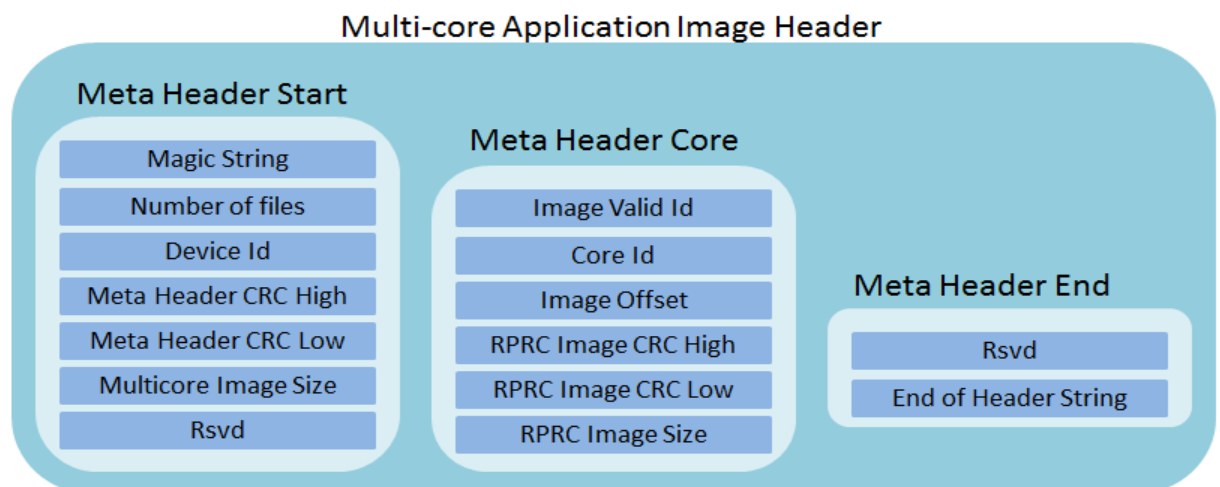
multicoreImageGen.exe is provided in tools as part of the starterware package (bootloader/Tools/multicore_image_generator).

There is a **batch file** provided under tools to generate application multicore Image from elf files. Refer to the section "Multi core Image generation script" of this user guide for batch file usage.

3.3.1 TDA3xx SBL

TDA3xx bootloader supports CRC checks on application image and hence the multicore application image header has the golden CRC embedded in it. This golden CRC is calculated using offline tool and SBL computes CRC using the TDA3xx CRC hardware and checks for CRC match. In case the CRC does not match the boot is aborted.

Sample multi-core header for TDA3xx application image with one core's application image is given below:



The multicore image can be generated using the tool present in tda3xx folder in multicore tool directory. CRC is computed using the tool crc_multicore_image present in the tools folder.

Command to run CRC tool is given below:

```
"crc_multicore_image\crc_multicore_image.exe" %Out_Path%\AppImage_LE %Out_Path%\AppImage_BE
```

Note: User should make sure that he uses corresponding tools with TDA3xx SBL, otherwise SBL boot will fail.

4 Building the SBL

SBL is part of starterware package located under bootloader sub-directory. Please refer the starterware user-guide to step-up the initial build environment. Command to build the SBL in release mode is as follows:

```
$gmake -s sbl PLATFORM=<device> BOOTMODE=<bootmode>
```

<device>: This parameter can be 'tda2xx', 'tda2ex' or 'tda3xx'

<bootmode>: To build sbl for SD boot mode the parameter is 'sd' (valid only for tda2xx/tda2ex)

To build sbl for QSPI boot mode the parameter is 'qspi'

To build sbl for NOR boot mode the parameter is 'nor'

To build sbl for QSPI_SD boot mode the parameter is 'qspi_sd' (valid only for tda3xx)

Note: Command to build the SBL for all boot modes in release mode:

```
$gmake -s sbl_all PLATFORM=<device>
```

The SBL sections should be mapped into OCMC Region. This is specified in the Linker command file. The RBL loads the SBL into OCMC RAM for non-XIP memory devices.

For NOR boot mode, the entry point has to be mapped to 0x08000000. This is specified in the NOR Linker command file.

Locate the SBL elf under directory binary/sbl_<bootmode>/bin/<PLATFORM>/sbl.out.

To convert the elf image into bin & tiimage file refer to the section 3.1 "SBL Bootloader Image" of this userguide.

5 Boot Modes of SBL

Supported boot modes on TDA2xx ES1.1 device:

Boot Mode	EVM Switch Setting SYSBOOT(SW2)[1:16]	EVM Switch Setting SW5[1:10]	Execution Mode	Entry location
QSPI_1	01101100 10000001	1110100000	Non-XIP	0x40300000
QSPI_4	11101100 10000001	1110100000	Non-XIP	0x40300000
NOR	10101100 10000101	0100100000	XIP	0x08000000
SD	00001100 10000001	0001100000	Non-XIP	0x40300000
Debug	00000000 10000001	XXXXXXXX		

Supported boot modes on TDA2xx ES1.0 device:

Boot Mode	EVM Switch Setting SYSBOOT(SW2)[1:16]	EVM Switch Setting SW5[1:10]	Execution Mode	Entry location
QSPI	01101100 10000001	1110100000	Non-XIP	0x40300000
NOR	10101100 10000101	0100100000	XIP	0x08000000
SD	11100000 10000001	0001100000	Non-XIP	0x40300000
Debug	00000000 10000001	XXXXXXXX		

Supported boot modes on TDA2Ex ES1.0 device:

Boot Mode	EVM Switch Setting SYSBOOT(SW2)[1:16]	EVM Switch Setting SW5[1:10]	Execution Mode	Entry location
QSPI_1	01101100 10000001	0001100000	Non-XIP	0x40300000
QSPI_4	11101100 10000001	0001100000	Non-XIP	0x40300000
NOR	10101100 10000101	0100100000	XIP	0x08000000
SD	00001100 10000001	0001100000	Non-XIP	0x40300000
Debug	00000000 10000001	XXXXXXXX		

Supported boot modes on TDA3xx 15X15 ES1.0 device:

Boot Mode	EVM Switch Setting SYSBOOT(SW2)[1:16]	EVM Switch Setting SW8001[1:8]	Execution Mode	Entry location
QSPI_1	00011000 10000001	0100 0001	Non-XIP	0x00300000
QSPI_4	10011000 10000001	0100 0001	Non-XIP	0x00300000
NOR	01011000 10000101	1100 0001	XIP	0x08000000
Debug	00111000 10000001	XXXXXXXX		

Supported boot modes on TDA3xx 12X12 ES1.0 device:

BootMode	EVM Switch Setting SYSBOOT(SW2)[04789]	EVM Switch Setting SW8001[1:8]	Execution Mode	Entry location
QSPI_1	01110	0100 0001	Non-XIP	0x00300000
QSPI_4	11110	0100 0001	Non-XIP	0x00300000

Note: Debug mode is not available on 12X12 EVM

5.1 Flash Tools

5.1.1 QSPI Flash Writer

In order to flash the SBL tiimage and multicore AppImage in QSPI flash, QSPI flash writer tool is required.

This tool is present in directory tools\flashtools\qspi_flash_writer.

Use the following command to build this tool in release mode:

\$make -s qspiFlashWriter PLATFORM=<platform>

The binary qspiFlashWriter_m4_release.xem4 will be created in folder
binary\qspiFlashWriter\bin\<PLATFORM>.

5.1.2 NOR Flash Writer

In order to flash the SBL bin file and multicore AppImage in NOR flash, NOR flash writer tool is required.

This tool is present in directory tools\flashtools\nor_flash_writer.

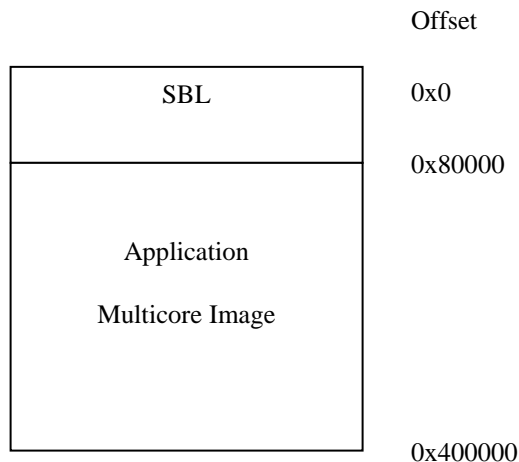
Use the following command to build this tool in release mode:

\$make -s nor_flash_writer PLATFORM=<platform>

The binary nor_flash_writer_m4_release.xem4 will be created in folder
binary\nor_flash_writer\bin\<PLATFORM>.

5.2 QSPI boot mode

QSPI Memory Map:



In QSPI flash, flash SBL tiimage at offset 0x0 and the multicore Application Image file at offset 0x80000.

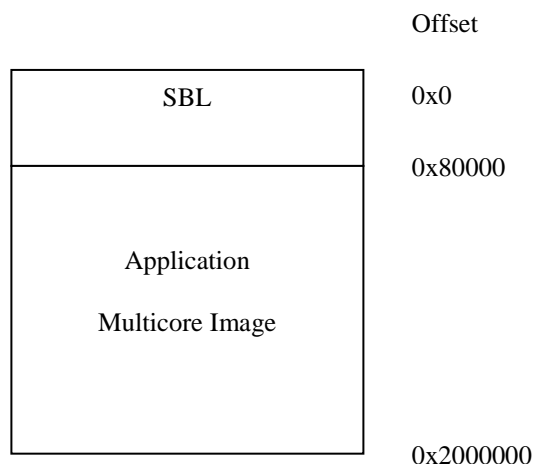
Flashing the image into QSPI Flash:

Please follow the steps below to flash the image at a particular offset address in QSPI-Flash memory:

- i) Open CCS & launch the target configuration.
 - ii) Change the SYSBOOT Switch (SW2) to debug mode.
 - iii) Set the relevant switches as given above to access QSPI FLASH.
 - iv) Connect the UART terminal & launch Uart console.
 - v) Make sure both SBL & AppImage are in Big Endian format.
 - vi) Connect Master CPU target. Load & run SBL ELF image (SD for tda2xx/tda2ex and QSPI_SD for tda3xx).
- Note: Refer to section 4 of this userguide to build SBL ELF image file.
- vii) Load the qspiFlashWriter_m4_release.xem4 on Cortex-M4 present at location binary\qspiFlashWriter\bin\<PLATFORM> & execute it from CCS.
 - viii) Select the relevant flash type (Spansion 4 bit for most of TI EVMs).
 - ix) Enter the image name to be flashed into QSPI.
 - x) Enter the offset value in hex.
 - xi) Select the erase mode. In full erase mode, tool will erase the whole flash & terminates. Reset the target & start from step VI.
 - xii) On selecting the Load option there will be two options: fread using RTS library or CCS scripting console. Fread using RTS could be slow for large images, so use the loadRaw command using the CCS scripting console.
 - xiii) Tool will first erase the flash & start writing the image.
 - xiv) “Verifying...Success” on the console signifies successful completion of the flashing.

5.3 NOR Boot Mode

NOR Memory Map:



For NOR boot mode, make sure that the SBL is in bin format. Just convert the SBL.out file to binary and **don't** convert to tiimage file.

Flash the SBL binary image in the NOR Flash at 0x0 offset and the Application multicore Image file at offset 0x80000.

Flashing the image into NOR Flash:

Please follow the steps below to flash the image at a particular offset address in NOR-Flash memory:

- i) Open CCS & launch the target configuration.
- ii) Change the SYSBOOT (SW2) switch to debug mode.
- iii) Set the relevant switches as given above to access NOR FLASH.
- iv) Connect the UART terminal & launch Uart console.
- v) Make sure the AppImage is in Little Endian format.
- vi) Connect Master CPU target. Load & run SBL image (QSPI/SD ELF images) on OCMC RAM.
Note: Refer to section 4 of this userguide to build SBL ELF image file.
- vii) Load the `nor_flash_writer_m4_release.xem4` into Cortex-M4 present at location `binary\nor_flash_writer\bin\<PLATFORM>` & execute it from CCS.
- viii) Enter the image name to be flashed into NOR.
- ix) Enter the offset value in Hex.
- x) Select the erase mode. In full erase mode, tool will erase the whole flash & terminates. Reset the target & start from step VI.
- xi) Select the Load option. Fread using RTS library or CCS scripting console. Fread using RTS could be slow for large images, so use the `loadRaw` command using the CCS scripting console.
- xii) Tool will first erase the flash & start writing the image.
- xiii) “Verifying...Success” on the console signifies successful completion of the flashing.

5.4 MMCSDBoot Mode

Create a primary FAT partition on MMC/SD card (FAT32 format with sector size 512).

Rename the SBL tiimage as MLO and copy to the SD card.

Rename the Application multicore image file as “AppImage” and copy to SD card.

Steps to create bootable SD card are given below

5.4.1 Using PC tool

- Ensure Empty SD card (at least 256MB, preferably 4GB SDHC) is available.
- Ensure SD memory card reader is available.
- Create a primary FAT partition on MMC/SD card (FAT32 format with sector size 512) and mark it as Active. A partition manager utility has to be used for the same.
- Format SD card from DOS command line as below.
“format <drive> /A:512 /FS:FAT32”

Make SD card partition as active using below tool:

<http://www.pcdisk.com/download.html>

IMPORTANT NOTE: Create a primary FAT partition on MMC/SD card (FAT32 format with sector size 512 bytes mark the partition as active.

5.4.2 Option 2: Steps to prepare a bootable SD card using DISKPART

- Open windows 7 Command prompt and Run as Administrator mode
- Enter command "diskpart.exe"

C:\Windows\system32>diskpart.exe will take you DISKPART prompt

Warning: Enter below command carefully w.r.t your computer/laptop SD card disk number. Choosing wrong disk number may delete data present in other drive

To list all disk drive present on computer:

DISKPART> list disk

Select the SD card disk number, in my case it is disk 1:

DISKPART> select disk 1

Now all next command applicable only to disk 1(SD card)

Delete entire partition:

DISKPART> clean

To create Primary partition:

DISKPART> create partition primary

To list partition:

DISKPART> list partition

Select partition:

DISKPART> select partition 1

To list volume:

DISKPART> list volume

Select volume associated with SD card, in our case it is 3:

DISKPART> select volume 3

Format SD card, please wait this may take few seconds:

DISKPART>format quick fs=fat32 unit=512 label=SD_BOOT

Make disk active:

DISKPART> active

To exit utility:

DISKPART> exit

5.5 QSPI_SD Boot Mode

In this boot mode, SBL is flashed on QSPI flash and AppImage is present on SD card.

Steps for how to flash the SBL on QSPI memory are present in section 5.2 of this userguide.

Create a primary FAT partition on MMC/SD card (FAT32 format with sector size 512).

Rename the Application multicore image file as “AppImage” and copy to SD card.

Note: Windows format disk not makes the SD card as bootable. Please format the card as bootable on Linux based PC.

NOTE: Only SBL image should be flashed on QSPI.

6 Build Mode

SBL supports two types of build modes: *production* and *development*.

In production build mode, if a valid app image for a given CPU core is not found, then the SBL puts the corresponding core to power down mode which helps save power in the device.

In development build mode, the CPU is not put to power down mode when valid app image is not there.

Production build mode is selected by default.

SBL can be built for different build modes using below command:

gmake -s sbl PLATFORM=<platform> SBL_BUILD_MODE=<mode>

mode: To build sbl for production boot mode the parameter is ‘prod’

To build sbl for development boot mode the parameter is ‘dev’

7 EMIFMODE

SBL supports three different LISA configurations for tda2xx device: DUAL_EMIF_2X512MB, DUAL_EMIF_1GB_512MB & SINGLE_EMIF_256MB. The different modes can be set with a build option EMIFMODE=<option>. By default, DUAL_EMIF_2X512MB is selected. The build command is given below:

```
$gmake -s sbl PLATFORM=<platform> BOOTMODE=<boot_option> EMIFMODE=<option>
```

Build SBL using below command for running starterware binaries and Vision SDK:

```
$gmake -s sbl PLATFORM=tda2xx BOOTMODE=<boot_option>  
EMIFMODE=DUAL_EMIF_2X512MB
```

8 Multi OPP Support

SBL supports three different OPPs for TDA2xx/TDA2Ex and TDA3xx devices: opp_nom, opp_high and opp_od. For TDA2xx device, SBL only one more OPP: opp_low. The different OPPs can be set using the build option OPPMODE=<option>. By default, opp_nom is selected. The build command is given below:

```
$gmake -s sbl PLATFORM=<platform> BOOTMODE=<boot_option> OPPMODE=<option>
```

Build SBL using below command for running starterware binaries:

```
$gmake -s sbl PLATFORM=<platform> BOOTMODE=<boot_option> OPPMODE=opp_nom
```

You can build SBL for all OPPs in all supported bootmodes using the target sbl_all_opps:

```
$gmake -s sbl_all_opps PLATFORM=<tda2xx/tda2ex/tda3xx>
```

Multiple OPPs are supported across different voltage rails where every rail might not support all three OPPs. If any voltage rail does not support a particular OPP and SBL is built for that particular OPP then that voltage rail is configured for OPP NOM.

Voltage Rail	TDA2xx OPP SUPPORT			
	OPP LOW	OPP NOM	OPP OD	OPP HIGH
VD_MPU	Supported	Supported	Supported	Supported
VD_DSPEVE	Not Supported	Supported	Supported	Supported
VD_IVA	Not Supported	Supported	Supported	Supported
VD_GPU	Not Supported	Supported	Supported	Supported
VD_CORE	Not Supported	Supported	Not Supported	Not Supported

Voltage Rail	TDA2Ex OPP SUPPORT		
	OPP NOM	OPP OD	OPP HIGH
VD_MPU	Supported	Supported	Supported
VD_DSPEVE	Supported	Supported	Supported
VD_IVA	Supported	Supported	Supported
VD_GPU	Supported	Supported	Supported
VD_CORE	Supported	Not Supported	Not Supported

Voltage Rail	TDA3xx OPP SUPPORT		
	OPP NOM	OPP OD	OPP HIGH
VD_DSPEVE	Supported	Supported	Supported
VD_CORE	Supported	Not Supported	Not Supported

9 SBL Optimization Level

SBL supports different levels of optimization in order to achieve better performance. This feature needs platform specific modifications and is available only on TDA3xx platform. There are three levels of optimization: 'high', 'medium' and 'low'. The amount of prints given by SBL reduces as the optimization level increases. By default the optimization level is set as 'low'. You can specify optimization level by using below build command:

gmake -s sbl PLATFORM=tda3xx SBL_OPT_MODE=<mode>

mode: To build sbl for high optimization level, the parameter is 'high'
 To build sbl for medium optimization level, the parameter is 'medium'
 To build sbl for low optimization level, the parameter is 'low'

10 WatchDog Timer2 (WD_TIMER2)

For tda2xx/tda2ex device, ROM Bootloader enables the WD_TIMER2 & sets the value to three minutes. Second stage Bootloader and application need to handle the WD_TIMER2. SBL refreshes the WD_Timer2 but does not disable it.

For tda3xx device, RTI DWWD is disabled by SBL.

11 DSP Boot address alignment

Boot address for the DSP Core should be aligned to '0x400' boundary. When the SBL brings DSP core out of reset it can load only the higher order 22 bits of the boot address and the lower order 10 bits are expected to be 0.

In the linker command file you can specify the following to align the boot address.

In case of non rtsc project the Entry point symbol `_c_int00` for will be defined in `rts*.lib` file.

SECTIONS

```
{  
    boot :  
    {  
        rts*.lib<boot.obj>(.text)  
    }load > MemorySectionName ALIGN(0x400)  
}
```

In case of the RTSC projects xdtools specify the Entry point symbol.

```
Program.sectMap[".text:_c_int00"] = new Program.SectionSpec();  
Program.sectMap[".text:_c_int00"].loadSegment = "DSP1_PROG";  
Program.sectMap[".text:_c_int00"].loadAlign = 0x400;
```

12 AD PLL Clock Frequency

Device ADPLLJM & ADPLL M are clocked at respective frequencies depending on the OPP. Refer to the device datasheet for OPP NOM frequency values.

13 Boot-Up time

13.1 TDA2xx Device

This section summarizes the SBL boot-up cycles for TDA2xx device in all three boot modes & techniques used to improve the boot-up cycles. It provides the boot-up time taken by SBL to initialize the SOC & to launch the application from external storage memories. Report does not include the ROM code boot-up time & application boot-up time. ARM Cortex PMU cycle counter has been used to measure the elapsed CPU cycles
VISION SDK App Image has been used for the SBL boot-up time profiling.

13.1.1 QSPI Boot mode:

QSPI driver throughput has improved with following settings:

1. Configure QSPI SCLK at 64MHz for ES1.1 device and 48 MHz for ES1.0 device.
2. Change the QSPI into memory mapped mode & quad read mode.
3. Use EDMA to copy the sections from serial flash to device memory.

13.1.2 NOR Boot mode:

NOR Flash data through put has improved by making following changes:

1. Configured the GPMC timing parameter for 266 MHZ GPMC FCLK. Previously it was configured for 66.5MHZ GPMC FCLK.
2. Changed to A-sync Page read mode.
3. Replaced the memcpy with EDMA channel copy.

13.1.3 SD Boot mode:

SD card data throughput has improved by making following changes in the SD driver:

1. EDMA Mode: Added the EDMA mode to read data from receive buffer.
2. BUSWIDTH: SD protocol supports 1-bit & 4-bit bus width. Changed the bus-width from 1-bit to 4-bit
3. Multi sector: Implemented multi-sector support in the MMC/SD driver

Summary of three boot modes

MPU runs at 750MHz and this measurement was done on ES1.0 device

Sl. No	Break Up	QSPI Boot mode	NOR Boot mode	SD Boot mode
		Time in ms	Time in ms	Time in ms
1	VM Initialization Clock Cycles	12.55	6.61	15.54
2	SOC PRCM Initialization Clock Cycles	19.65	20.00	19.64
3	DDR3 Initialization Clock Cycles	5.84	5.96	5.87
4	App Image Load Clock Cycles (App Image Size 24,746 KB)	1516.46	568.64	20367.33
5	Slave Core Bring-up Clock Cycles	5.70	5.76	5.69
6	SBL Total Boot-up Clock Cycles	1566.40	613.29	20420.26

13.2 TDA2Ex Device

This section summarizes the SBL boot-up cycles for tda2ex device in all three boot modes. It provides the boot-up time taken by SBL to initialize the SOC & to launch the application from external storage memories. Report does not include the ROM code boot-up time & application boot-up time. ARM Cortex PMU cycle counter has been used to measure the elapsed CPU cycles. MPU runs at 800 MHz.

VISION SDK App Image has been used for the SBL boot-up time profiling.

Sl. No	Break Up	QSPI Boot mode	NOR Boot mode	SD Boot mode
		Time in ms	Time in ms	Time in ms
1	VM Initialization Clock Cycles	8.23	6.31	8.23

2	SOC PRCM Initialization Clock Cycles	19.37	19.75	19.36
3	DDR3 Initialization Clock Cycles	4.35	4.39	4.37
4	App Image Load Clock Cycles (App Image Size 15,274 KB)	719.91	347.16	12500.43
5	Slave Core Bring-up Clock Cycles	1.13	1.14	1.12
6	SBL Total Boot-up Clock Cycles	759.04	384.87	12539.55

13.3 TDA3xx Device

This section summarizes the SBL boot-up cycles for tda3xx device in all three boot modes. It provides the boot-up time taken by SBL to initialize the SOC & to launch the application from external storage memories. Report does not include the application boot-up time. 32K Timer has been used to measured elapsed time. By default, SBL instrumentation is enabled to measure the boot-up cycles.

VISION SDK App Image has been used for the SBL boot-up time profiling.

Summary of all boot modes on 15X15 EVM (SBL QSPI images in tiimage format):

		QSPI Boot mode	QSPI_SD Boot mode	NOR Boot mode
Sl. No	Break Up	Time in ms	Time in ms	Time in ms
1	Reset to SBL Init Cycles	17.36	23.25	3.87
2	SBL Initial Config Cycles	2.50	5.52	3.32
3	SOC Init Cycles Including TESOC Tests other than IPU Cycles	45.28	48.09	45.53
4	TESOC Tests other than IPU Cycles	44.98	47.21	45.10
5	DDR Config Clock Cycles	4.05	4.11	4.08
6	App Image Load Cycles (App Image size 16,043 KB)	550.96	17648.59	396.88
7	Slave Core Boot-up Cycles	0.03	60.45	0.03
8	SBL Boot-up Cycles	602.84	17766.84	469.84

Summary of two boot modes on 15X15 EVM (SBL QSPI images in chimage format):

		QSPI Boot mode	QSPI_SD Boot mode
Sl. No	Break Up	Time in ms	Time in ms
1	Reset to SBL Init Cycles	16.17	21.39
2	SBL Initial Config Cycles	2.47	5.43
3	SOC Init Cycles Including TESOC Tests other than IPU Cycles	45.28	48.09
4	TESOC Tests other than IPU Cycles	45.01	47.21
5	DDR Config Clock Cycles	4.05	4.11
6	App Image Load Cycles (App Image size 16,043 KB)	550.93	17650.93
7	Slave Core Boot-up Cycles	0.03	60.42
8	SBL Boot-up Cycles	602.84	17769.10

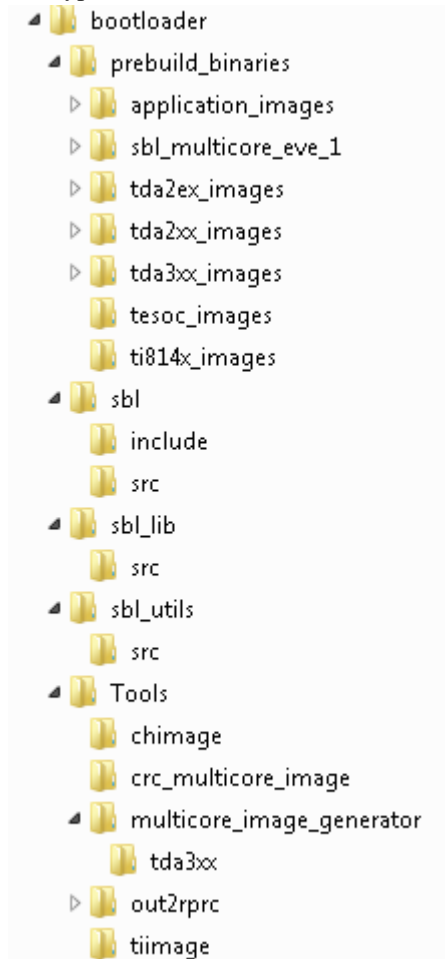
Summary of two boot modes on 12X12 EVM (SBL QSPI images in tiimage format):

		QSPI Boot mode	QSPI_SD Boot mode
Sl. No	Break Up	Time in ms	Time in ms
1	Reset to SBL Init Cycles	17.36	23.25
2	SBL Initial Config Cycles	3.23	6.40
3	SOC Init Cycles Including TESOC Tests other than IPU Cycles	45.28	48.09
4	TESOC Tests other than IPU Cycles	44.98	47.21
5	DDR Config Clock Cycles	2.83	2.89

6	App Image Load Cycles (App Image size 7,216 KB)	251.55	7995.20
7	Slave Core Boot-up Cycles	0.03	58.77
8	SBL Boot-up Cycles	302.97	8071.44

14 Component Folder

The typical folder Structure of the bootloader after installing the starterware package:



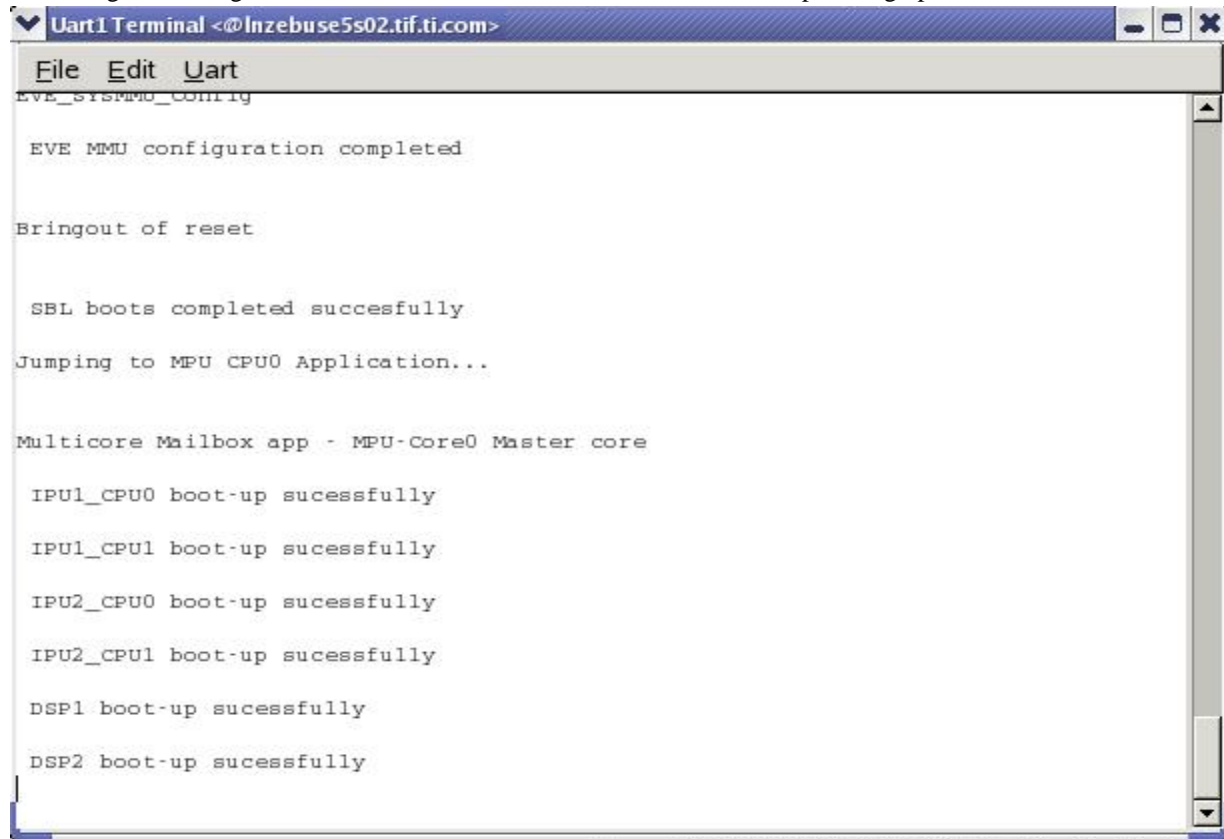
15 UART Console

Splash screen is displayed on the UART console on boot-up. SBL sends the log messages to UART console.

Messages display clock ticks are also sent to UART Console.

Connect to UART-1 terminal on the EVM for tda2xx device and UART-3 for tda3xx device & configure the Serial Terminal at 115200 baud rate, 8 bit data, 1 stop bit, no parity and no flow control.

Following is the image of the UART console for mailbox multi core example bring up from SBL for tda2xx device.



```

Uart1 Terminal <@Inzebuse5s02.tif.ti.com>
File Edit Uart
EVE_MMU_Config

EVE MMU configuration completed

Bringout of reset

SBL boots completed successfully

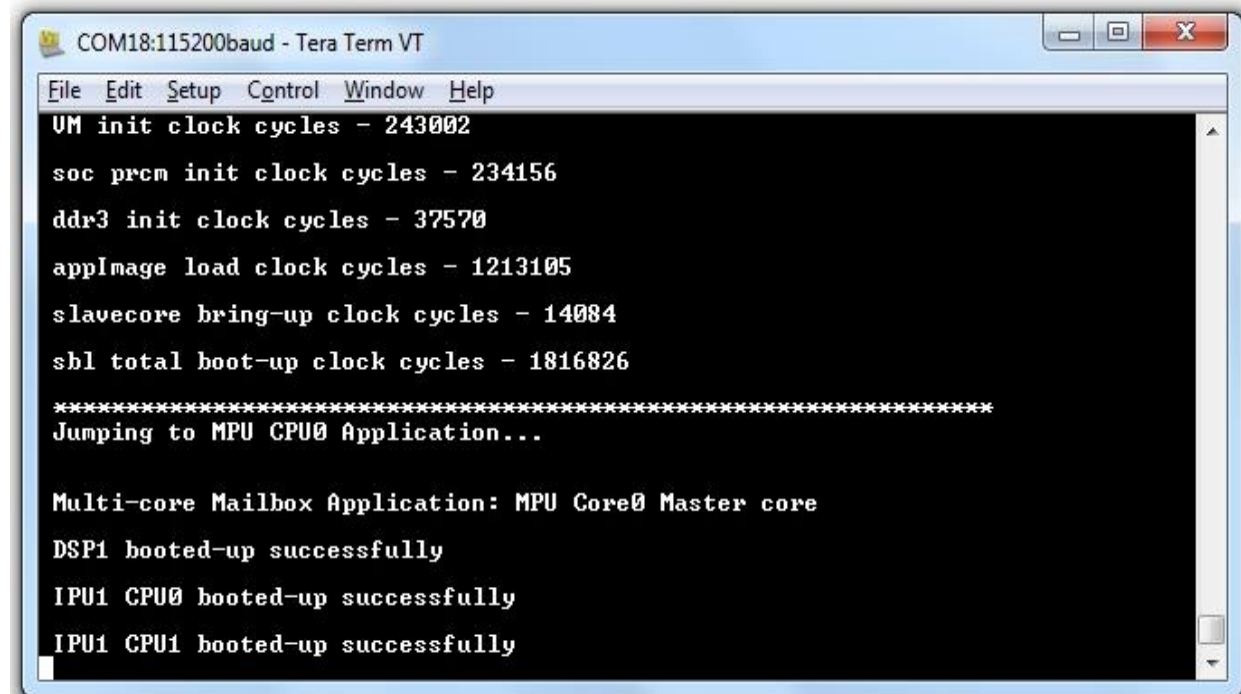
Jumping to MPU CPU0 Application...

Multicore Mailbox app - MPU-Core0 Master core

IPU1_CPU0 boot-up successfully
IPU1_CPU1 boot-up successfully
IPU2_CPU0 boot-up successfully
IPU2_CPU1 boot-up successfully
DSP1 boot-up successfully
DSP2 boot-up successfully

```

Following is the image of the UART console for mailbox multi core example bring up from SBL for tda2ex device



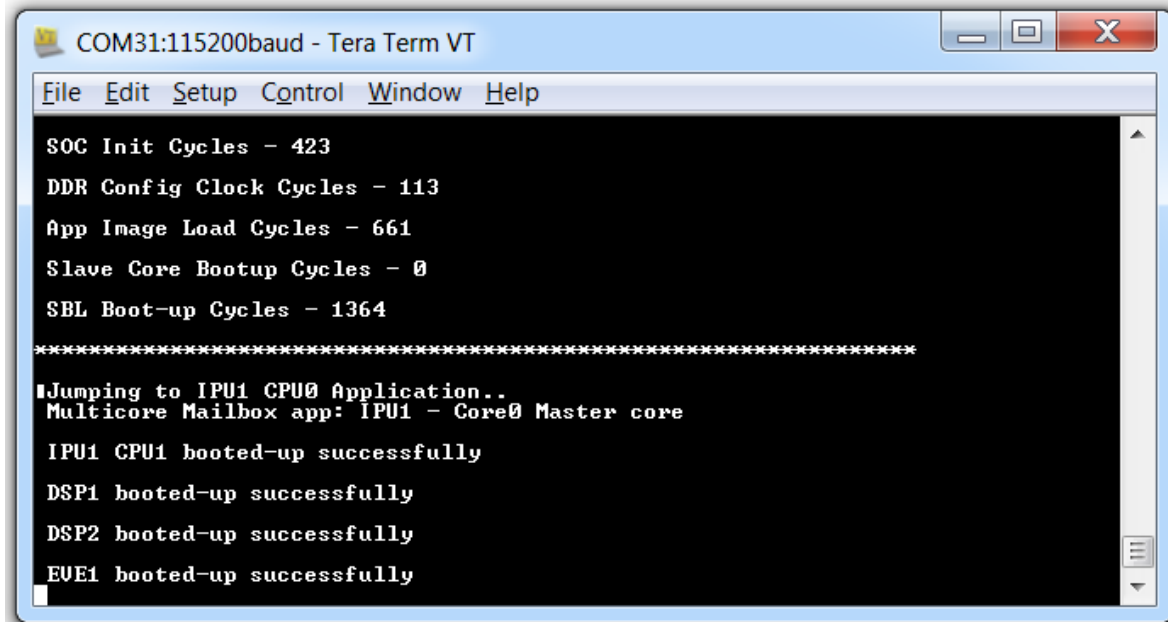
```

COM18:115200baud - Tera Term VT
File Edit Setup Control Window Help
UM init clock cycles - 243002
soc prcm init clock cycles - 234156
ddr3 init clock cycles - 37570
applImage load clock cycles - 1213105
slavecore bring-up clock cycles - 14084
sbl total boot-up clock cycles - 1816826
*****
Jumping to MPU CPU0 Application...

Multi-core Mailbox Application: MPU Core0 Master core
DSP1 booted-up successfully
IPU1 CPU0 booted-up successfully
IPU1 CPU1 booted-up successfully

```

Following is the image of the UART console for mailbox multi core example bring up from SBL for tda3xx device.



16 Memory Usage

Details of Static memory taken by SBL image for tda2xx created from release mode binary are given below:

		QSPI Boot Mode	SD Boot Mode	NOR Boot Mode
Section	flag	size(Bytes)	size(Bytes)	size(Bytes)
.text	AX	54672	66036	46728
.data	WA	21600	21600	13956
.bss	WA	1984	4088	1664
.stack	WA	4096	4096	4096
.rodata	A	21716	22068	21700
Total		104068	117888	88144

Details of Static memory taken by SBL image for tda2ex created from release mode binary are given below:

		QSPI Boot Mode	SD Boot Mode	NOR Boot Mode
section	Flag	size(Bytes)	size(Bytes)	size(Bytes)
.text	AX	41824	53140	36356
.data	WA	13388	13388	13388
.bss	WA	1952	4052	1632
.stack	WA	4096	4096	4096
.rodata	A	18516	18868	18500
Total		79776	93544	73972

Details of Static memory taken by SBL image for tda3xx created from release mode binary are given below:

		QSPI Boot Mode	QSPI_SD Boot Mode	NOR Boot Mode
section	Flag	size(Bytes)	size(Bytes)	size(Bytes)
.text	AX	35386	52936	33750
.data	WA	8611	15195	8291
.bss	WA	236	1372	236
.cinit	A	7096	8632	7096
.stack	WA	4096	4096	4096
.rodata	A	9804	10672	9792
Total		65229	92903	63261

17 Prebuilt Binaries

prebuild_binaries folder present in directory: \bootloader\prebuild_binaries contains the SBL images for different boot-modes & AppImages for multicore mailbox application. The EVE binary required to create multicore mailbox app is present in folder \bootloader\prebuild_binaries\sb1_multicore_eve_1.

SBL images for all the boot modes are created using release mode binaries/ELF files.

For more information on release and debug mode refer to section 5.2 of Starterware Userguide.

18 Multi core Image generation script

Under Tools, a batch file “MulticoreImageGen_<Device>.bat” is provided for generating the multi core Image from the elf files. User needs to edit the batch file with the path to the elf files.

Edit the CPU batch variables to provide the elf files along with path

If it is not required to load a particular core then leave the corresponding variable blank.

Edit Out_Path with the location where the generated multi core image should be stored.

Paths to tools used out2rprc and MulticoreImageGen are the relative paths in the SBL.

Script create 2 files AppImage_LE and AppImage_BE

AppImage_LE is a little endian file used in SD and NOR boot mode.

AppImage_BE is a big endian target used for QSPI and SPI boot mode.

19 SBL Multicore Mailbox app

It is multicore sample application and uses mailbox for inter-processor communication. It is used to validate the multi-core boot-up use case. Master application sends wake-up message to all the slave cores & waits for acknowledgement message from the slave cores in an infinite loop. Each slave core waits for wake-up message from the master core & responds back with an acknowledgement message.

Multicore mailbox app place under /examples/sbl_multicore_mbx

AppImage of this example placed under prebuilt binaries.

20 Regression Support

Regression feature is supported in SBL for tda2xx device. SBL supports to run the binaries in regression mode & logs the result in UART. Regression feature is tested with DV & Sival test cases on 23X23 & 17X17 devices. To run SBL in regression mode, build the image with regression option. Convert the out file to MLO. Copy all the test cases and MLO file in SD card & boot the device in SD boot mode.

\$gmake -s sbl PLATFORM=<device> BOOTMODE=<regression>

21 Known Issues

Please refer to the package release notes for the list of open issues & issues closed in the current release.