Application Note

AWR294x/AWR2544 Primary and Secondary Bootloader

ABSTRACT

This document describes the basic detail of AWR294x/AWR2544 Primary and Secondary Bootloader (RBL and SBL). The document also describes design and implementation considerations for secondary bootloader software. This document focuses only on non-secure AWR294x/AWR2544 device variant.

Table of Contents

Trademarks

1 Definitions, Abbreviations, Acronyms

Term	Definition
RBL	ROM Bootloader
SBL	Secondary Bootloader
TCM	Tightly Coupled Memory
CAN	Controller Area Network
MPU	Memory Protection Unit
AWR294x	AWR2943 and AWR2944
MSS	Main Subsystem
DSS	DSP Subsystem
RSS/BSS	Radar Subsystem/BIST Subsystem
sFLASH/SDF	Serial Flash
CCS	Code Composer Studio

2 Introduction

The AWR294x/AWR2544 device can be broadly split into three subsystems (see Figure 2-1,Figure 2-2), as follows:

- Main subsystem (MSS): ARM® Cortex®-R5F and associated peripherals, hosts the user application
- DSP subsystem (DSS): TI C66x and associated peripherals, hosts the user application. The DSP core TI C66x is not applicable for AWR2544 as the core is not there in the design.
- Radar/BIST SubSystem: Programmed using predefined message transactions specified by TI (reference driver : mmWaveLink provided by TI). This subsystem is black box and not available for user application.

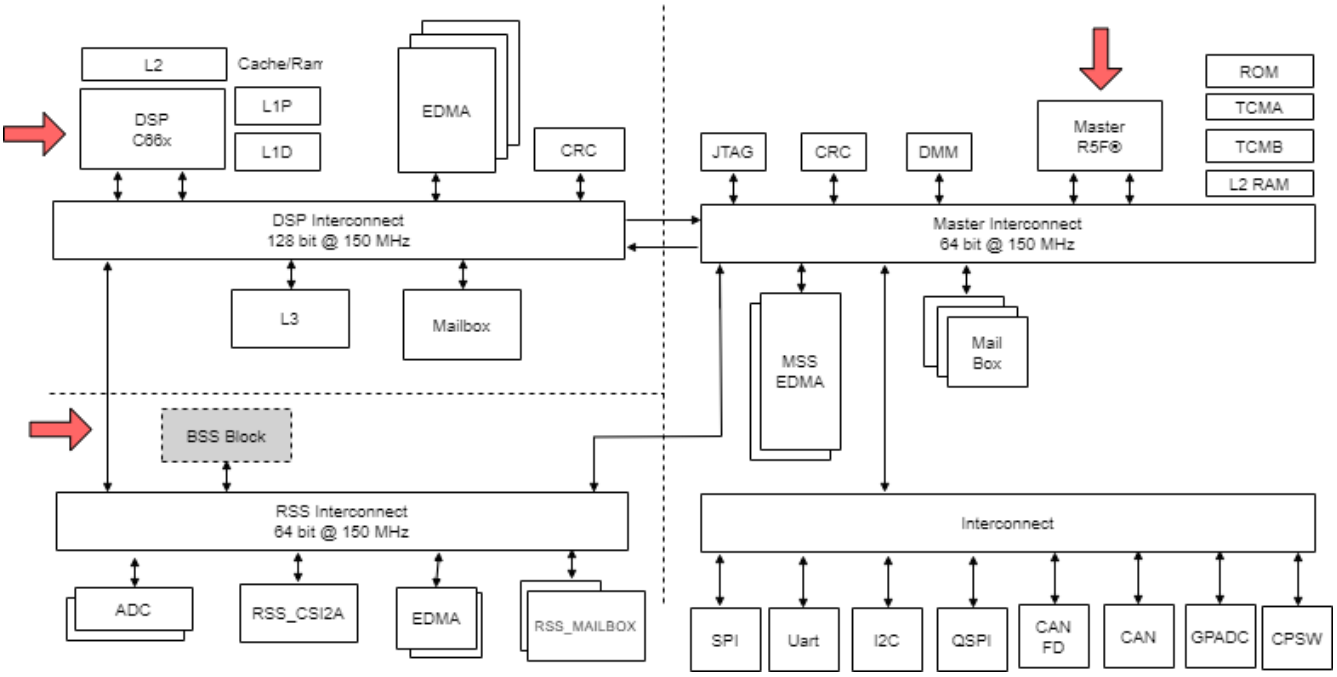

Figure 2-1. AWR294x Subsystems

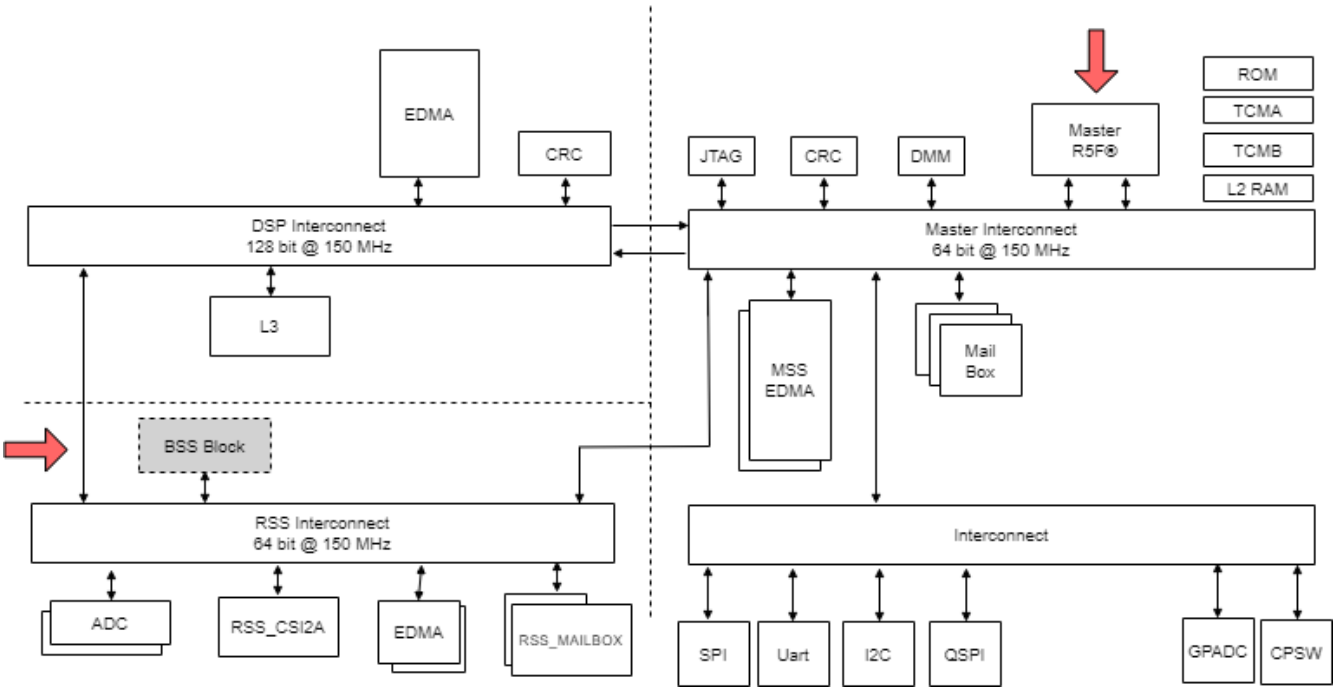

Figure 2-2. AWR2544 Subsystems

User application components (R5F and DSP) are expected to be stored in the serial data flash (SDF) interfaced to the AWR294x device over the quad serial peripheral interface (QSPI) interface. For AWR2544 user application component is R5F appimage only as the DSP is not applicable.

Main subsystem is the first programmable block to get activated after the AWR294x/AWR2544 device reset is de-asserted. The bootloader of the AWR294x/AWR2544 device is hosted in the read-only memory (ROM) of the main subsystem, and takes control immediately.

From this point onward, the AWR294x/AWR2544 bootloader can operate in two modes: flashing and functional. The bootloader checks the state of the sense on power (SOP) I/Os – SOP lines driven externally for choosing the specific mode (see Table 2-1).

Table 2-1. SOP Lines and Boot Modes

SOP2 (T17)	SOP1 (R14)	SOP0 (R14)	Bootloader Mode and Operation
0	0	1	Functional or QSPI Boot Mode The RBI loads the SBL from the QSPI serial flash to the internal RAM (MSS L2) and switches the over control.
1	0	1	Flashing or UART Boot mode The RBL spins in loop to allow user to load SBL or the User application over UART(XMODEM) to RAM directly. This mode can be used to load the flash programmer application to RAM which further is responsible to load SBL and user application to SFLASH.

UART boot mode of the RBL allows an external entity to load the customer application (Flash programmer) image to the RAM (MSS L2 only) which further downloads Secondary Bootloader (SBL) and/or user application to SDF (see Figure 2-3).

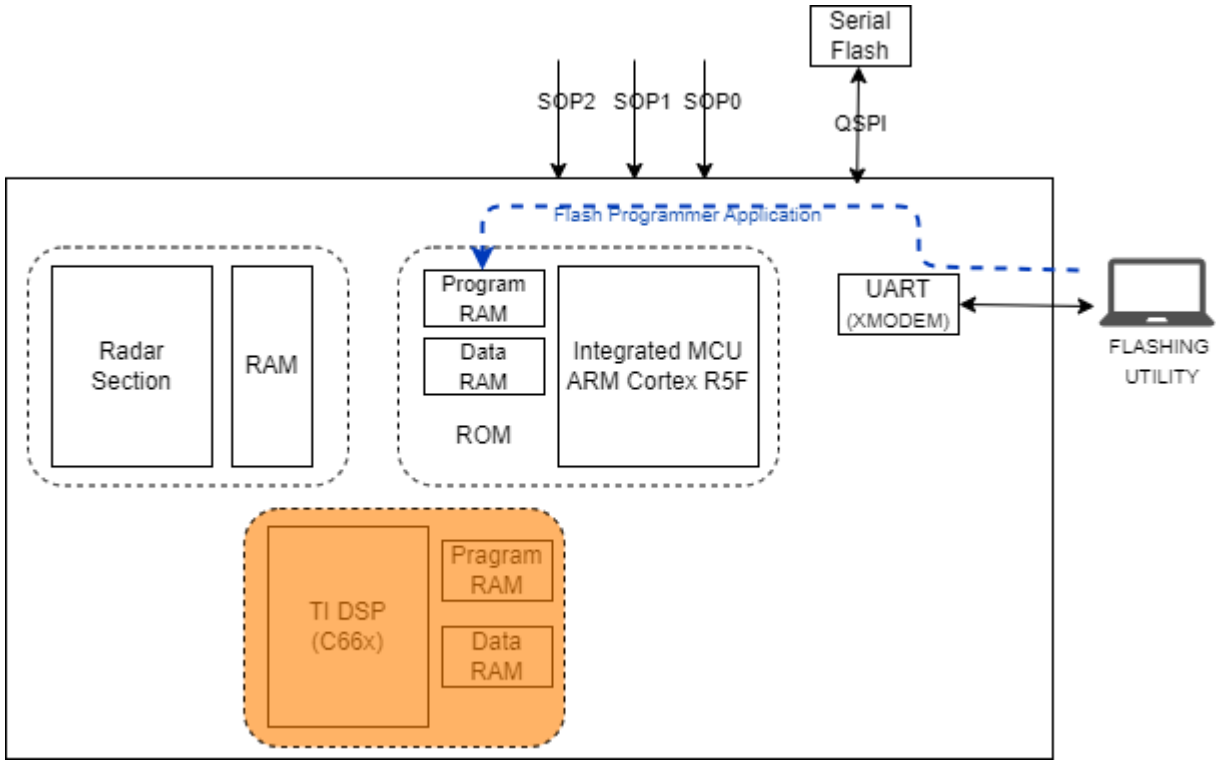

Figure 2-3. Flashing Mode of RBL

Functional mode of the RBL relocates the image stored in the SDF to the R5F memory subsystems, this R5F image is SBL. Towards the end of this process, the bootloader passes control to R5F user defined SBL. Loading

and unhaulting (start execution) of the DSP and R4F (BSS) image/core is the responsibility of the SBL (see Figure 2-4).

Note

Please note TI C66x is not applicable for AWR2544.

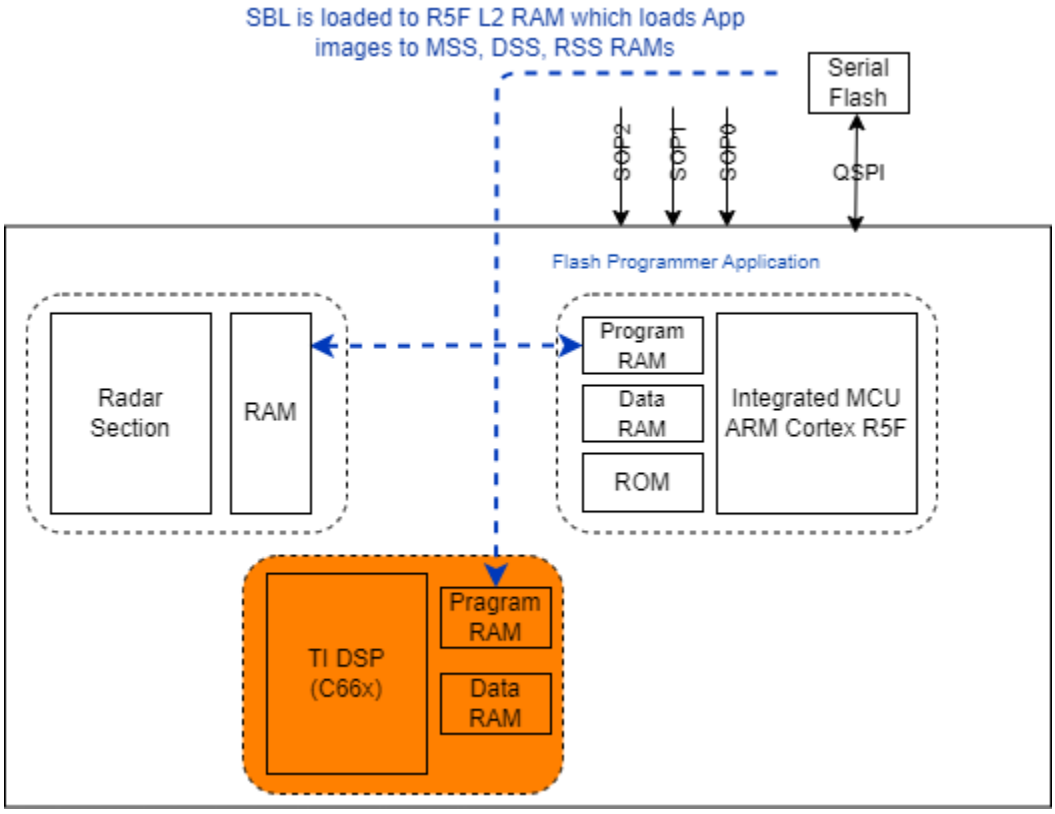

Figure 2-4. Functional Mode of RBL

Note

Please note TI C66x is not applicable for AWR2544.

Key points

- AWR294x's RBL can load only one primary user image (can have content for R5F L2 only).
- This primary user image is SBL which takes responsibility to load & download MSS, DSS and BSS image/ patch to/from SDF. Customer must invest in SBL to handle multiple images (factory programmed, back-up, and so on).

3 Basic Bootloader Flow

3.1 Boot Flow Introduction

Booting user defined applications on a SOC involves multiples steps as listed below,

- Firstly, there are multiple steps involved to convert a user application, created using a compiler+linker toolchain, into a binary format that is designed to be booted by the SOC
- Next, we need to flash this binary file to the on-board serial flash.
- Finally, when the SOC is powered on, the previously flashed binary is executed.
- After powering on the device in functional mode, the boot flow takes place mainly in two steps
  - ROM boot: In which the RBL boots an SBL reading from the sFlash.
  - SBL boot: In which the secondary bootloader boots the application reading from the sFlash.
- Note, that a system application (i.e. metaimage) itself can consist of multiple CPU specific application binaries that all collaborate together to realize the overall system goal.

3.2 Preparing the Application for Boot

Shown below are the different steps that are done to convert the compiler+linker generated application .out into a format designed for flashing and booting.

- For each CPU, the compiler+linker toolchain is used to create the application .out "ELF" file which can be loaded and run via CCS/ JTAG IDE.
- The below "post build" steps are then used to convert the application .out into a "flash" friendly format
  - For each CPU, out2rpc converts the application executable (.out) into custom TI RPRC (.rprc) image. This tool strips out the initialized sections from the executable file (*.out) and places them in a compact format that the SBL can understand. The output RPRC file is typically much smaller than the original executable (*.out) file.
  - multiCoreGen is then used to combine all the RPRC files per CPU into a single .appimage file which is a concatenation of the individual CPU specific RPRC files.
- This .appimage can then be flashed to the device.

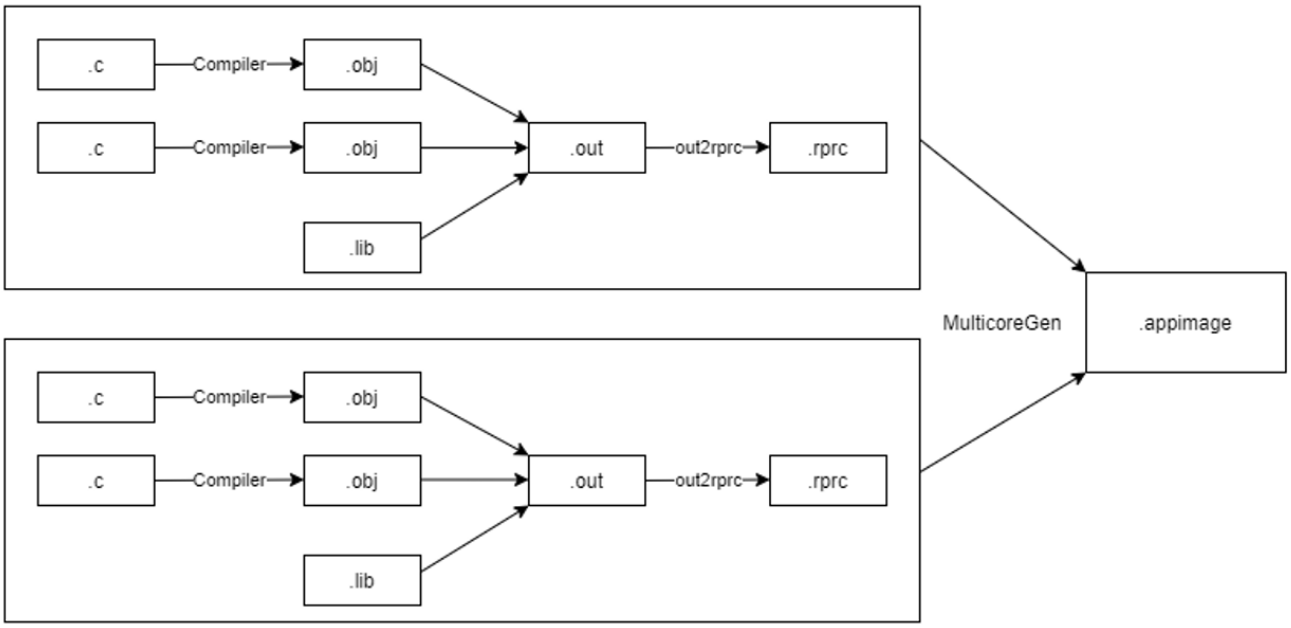

Figure 3-1. Post Build Steps

3.2.1 SBL Image Format

TI provides example R5F SBL in the MCU plus SDK package. The format the SBL is provided is custom TI image (.TIIMAGE). The output file (.out) from the compiler+linker toolchain is converted to binary format (.bin) and a certificate is attached to this image to make the final custom TI Image.

3.2.2 Signing Scripts

The RBL requires the boot image (mostly SBL) to always be signed. If not signed, the RBL is not be able to boot the SBL. Any image being loaded by the RBL in flashing or functional mode needs to be signed. The process of certificate creation and attaching the certificate to the binary file is referred to as signing the script.

Key Points

- The RBL always expects a signed image in any mode of operation. Thus, any image being loaded by the RBL has to signed. In functional mode this means the SBL in the sFLASH must be signed. In flashing mode this means the flash programmer image being loaded by the RBL onto the RAM must also be signed.
- The above formats are custom reference format provided by TI. As the SBL is user implemented and booting the .appimage, the user can define any format for the .appimage as per their preference and implement SBL in the required manner to parse this image.

See the Booting Tools section in the readme file available in the mmWave MCU Plus SDK release for more details.

After an SBL and application image is flashed, shown below is the high-level boot flow, after the SOC is powered on.

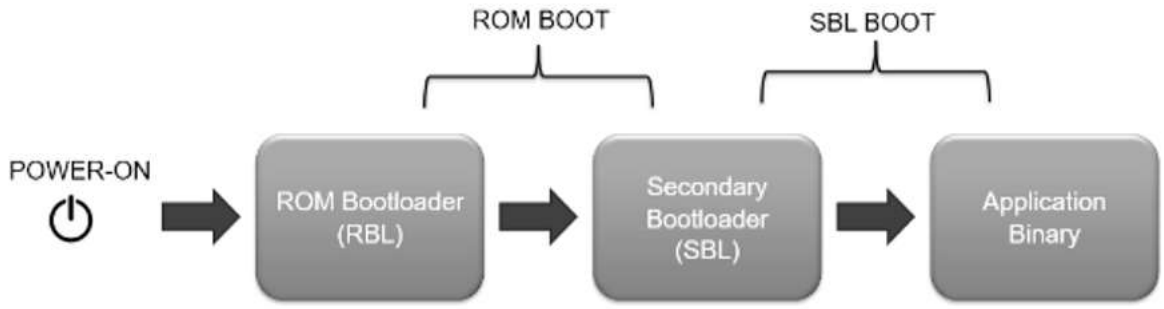

Figure 3-2. High Level Boot Flow

3.3 ROM Boot

The boot process consists of two consecutive steps: primary or ROM boot loader (RBL) process, followed by the secondary boot loader (SBL) process. As soon as the EVM is powered ON, the ROM bootloader or RBL starts running. The RBL is the primary bootloader. The goal of RBL is to load, verify, optionally decrypt, and launch authentic the R5F software image that accomplishes secure boot goals (in a secure variant). The RBL process is implemented jointly by the R5F and HSM ROM as illustrated in the figure (figure 4) below. The RBL expects the image (SBL in this case) to always be signed.

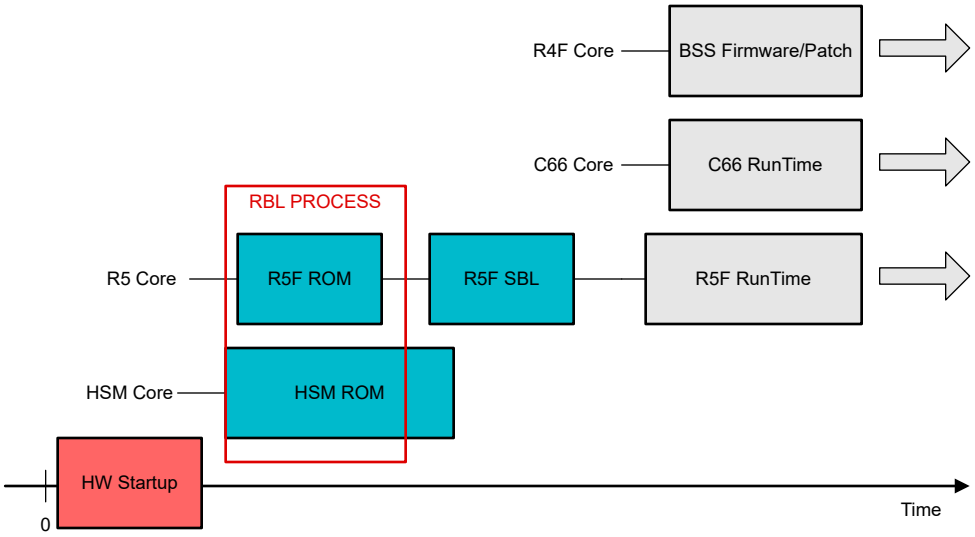

Figure 3-3. Boot Process

Note

Please note that the C66 Run time and C66 Core is not applicable for AWR2544.

At high level the boot flow can be explained as below.

- The HSM ROM is the first code to be executed at system reset. HSM ROM executes a set of self-tests for data SRAM, program SRAM, ROM code integrity, as part of device initialization and configures the APLL. HSM ROM also releases the R5F from reset**.**
- In the HSM ROM PBIST is performed on secure RAM, public RAM and ROM code integrity check is performed for the HSM sub system.
- In the R5F RBL, PBIST is performed on MSS TCMA,MSS TCMB and MSS_L2 memories.
- The R5F checks for the SOP settings and based on continued execution.
- In the UART boot mode/Flashing mode the RBL is expected to get flash programmer (or any other relevant image) from UART. The flash programmer is generally used to download and flash the SBL to the QSPI flash.
- In the QSPI boot mode/Functional mode the RBL loads the SBL from the flash memory on to the internal RAM and begins execution.

Note

Please note for AWR2544 device the RBL does not perform the PBIST on MSS TCM memories and MSS L2. The user is required to perform the operation in the SBL. Please check the AWR2544 SDK for this implementation.

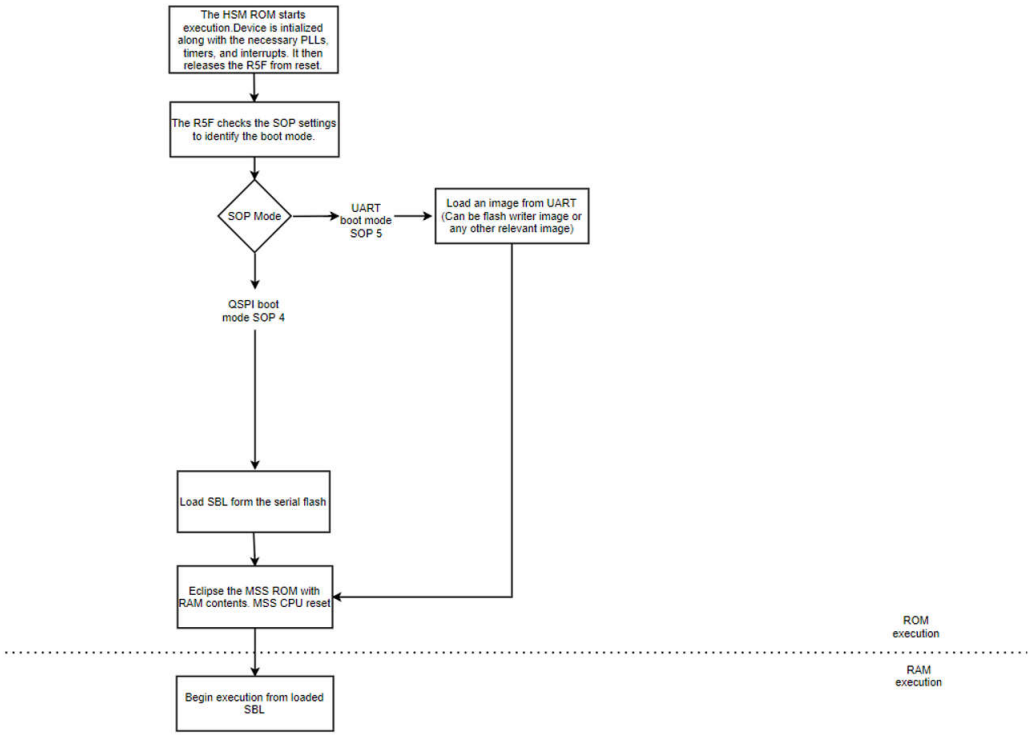

Figure 3-4. ROM Boot Flow

Key points

- The ROM bootloader loads only one image and to R5F L2 RAM only.
- The ROM bootloader sets up the root clock by starting the APLL. The root clock is at 200MHz

3.3.1 Boot mode - SFLASH

3.3.1.1 Image Load Sequence

In functional mode, the bootloading of an image from the SDF is the first bootmode attempted by the bootloader. This bootmode involves the following steps:

1. Pinmux the QSPI pins of the AWR294x device:
   - QSPI[0]: Ball U11
   - QSPI[1]: Ball V11
   - QSPI[2]: Ball T11
   - QSPI_CLK: Ball R10
   - QSPI_CS: Ball U12
2. QSPI is set up to operate at (system clock / 5) = (200/5) = 40 MHz.
3. The sFLASH discoverable parameters (SFDP) command is issued to retrieve the JEDEC compliant response, which includes information regarding the sFLASH capabilities and command set. When the SFDP response is received, the information is used to communicate with the SDF and further interpret the contents and load the images. For more information on the flash variants supported with AWR294x/AWR2544 devices please refer to the application note.

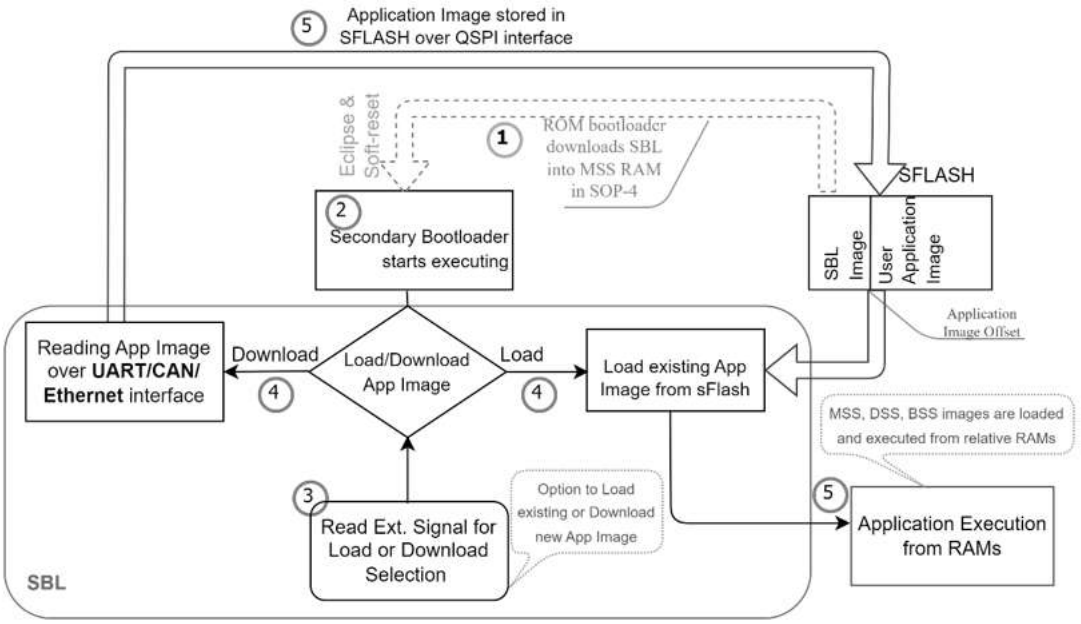

Figure 3-5. Loading of SBL and App Meta Image from SFLASH

Note

Please note that the CAN interface is not applicable for AWR2544 device.

Key points

- The RBL performs the read from the SDF, based on the highest capability mode (quad, dual, or single) as published by the SDF in response to the SFDP command.
- For SDF variants that support quad mode, the quad mode commands are issued; if the quad enable (QE) bit is not set, the communication fails. In such cases, the load flow assumes that the QE bit in the SDF is already set.
- SBL is user implemented entity which has logic to achieve above mentioned flow. SBL can use any interface out of UART, CANFD or Ethernet if required.
- Fallback images: the RBL supports loading of images from the following locations only as a fallback Mechanism, if one of the images is corrupted in the SDF. The locations of the images are:
  - – META IMG1(SDF offset – 0x0)
  - – META IMG2(SDF offset – 0x40000)

3.3.1.2 Boot Mode UART

UART boot mode is based on the XMODEM protocol. UART port can be configured with following parameters:

Parameter	Value
Physical port	0
Baud rate	115200
Data bits	8
Stop bits	1
Parity	None
Flow Control	None

R5 starts Xmodem receive protocol with sending out PING character (“C”) every 3 seconds (Ping timeout).

3.3.1.2.1 Image Download Sequence

This image download sequence is entered by placing the device in flashing mode

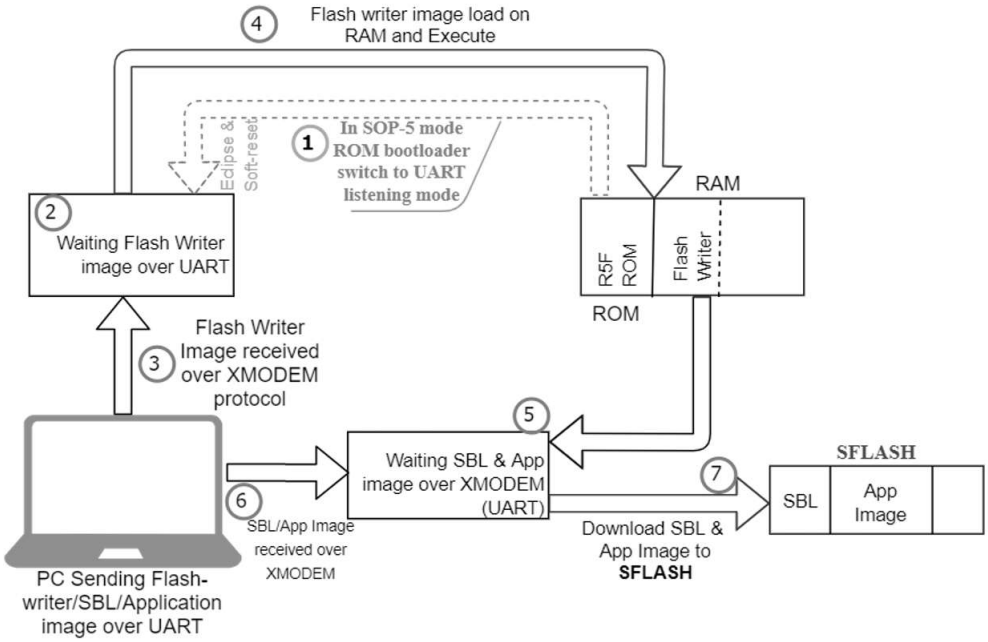

Figure 3-6. Loading flash writer image

Key points

- The RBL expects a valid image over the UART peripheral. This image can be flash writer image which can then download the SBL and APP images over the UART and store them in the sFlash device.
- If the user wishes to download the SBL and APP images over CAN or Ethernet peripheral then the peripheral can load another relevant image (instead of the flash writer image which downloads images over UART) to achieve that.

3.4 SBL Boot

The SBL is essentially an example application of the bootloader library. SBL is called a secondary bootloader because SBL is booted by the RBL, which is the primary bootloader. An SBL typically does a bunch of SOC specific initializations and proceeds to the application loading.

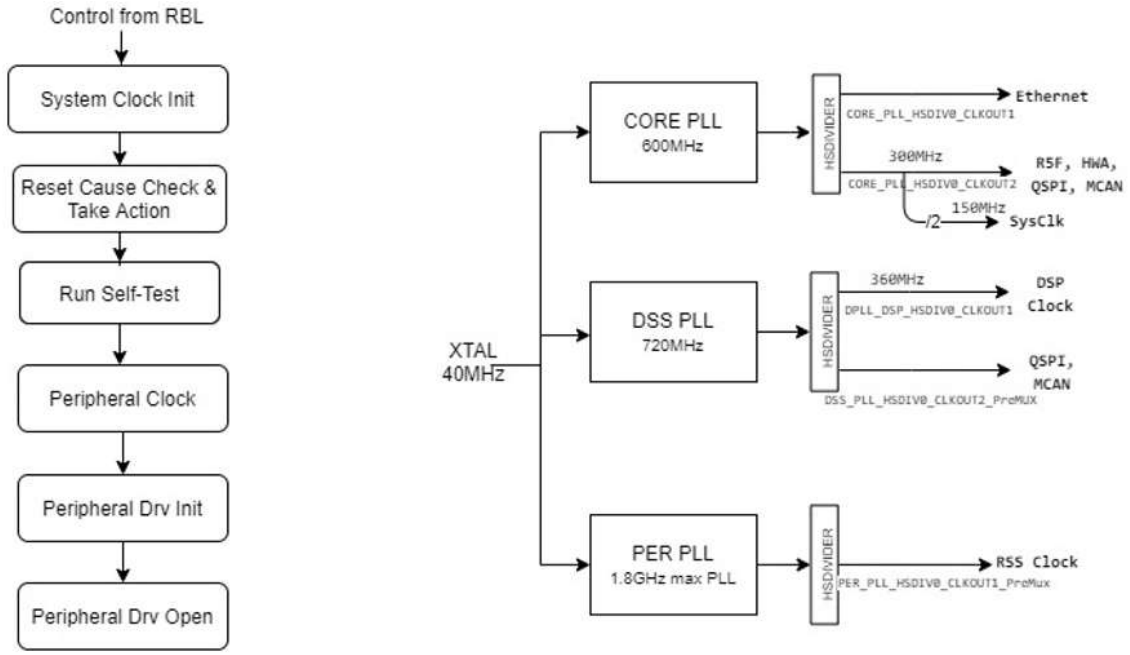

Figure 3-7. SBL Design Flow

The secondary bootloader can update the application meta image in the sFLASH by receiving the image over a serial interface. The bootloader then loads and runs the updated application meta image. The ROM (primary) bootloader always loads the SBL. User defined SBL can choose to either update or load and run the existing application meta image.

The SBL can make sure that the factory default backup image is never erased or updated by the image updater. The upgrade is done for the entire meta image. The user defined SBL can perform the checksum verification to verify the validity of the meta image trying to load from sFLASH. In the case where the image is corrupted or download is interrupted, the SBL provides a failsafe mechanism to reload the image. This can be achieved resetting the board. If the primary meta image fails to load, the factory default backup image is loaded by the SBL. If both fail, SBL can reset the board so the user can re-attempt the download of the meta image.

mmWave MCU Plus SDK provides a reference implementation of SBL, which user can refer and write their own SBL . Here are basic features of this SBL

- SBL looks for the multicore appimage of the application binary at a specified location in a boot media[GJ3] .
- If the appimage is found, the multicore appimage is parsed into multiple RPRCs[GJ4] . These are optimized binaries which are then loaded into individual CPUs.
- Each RPRC image has information regarding the core on which the image is to be loaded, entry points and multiple sections of that application binary
- The SBL uses this information to initialize each core which has a valid RPRC. The SBL then loads the RPRC according to the sections specified, sets the entry points and releases the core from reset. Now the core starts running

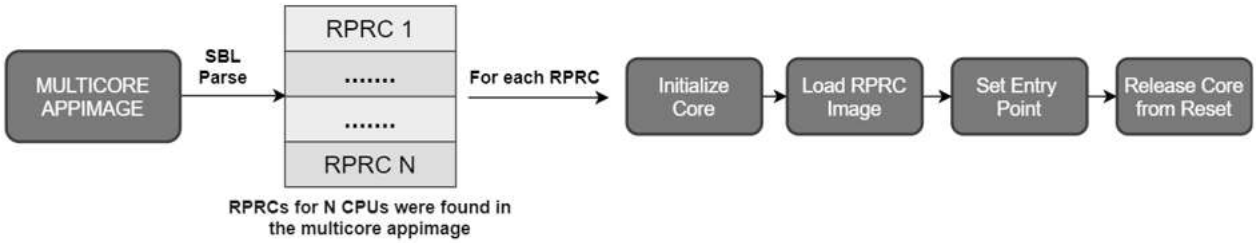

Figure 3-8. SBL Boot

3.4.1 R5 SBL Flash Offset

RBL loads primary and secondary SBL image from the following locations:

SBL Image	Flash offset
Primary	0x00000000
Secondary	0x00040000

In QSPI Flash mode, the Primary Flash offset address is first checked for a valid certificate and image. If R5F SBL is not present, then the Secondary Flash offset is checked for a valid certificate and image. If none is found, the RBL waits in WFI mode. The Watchdog on the HSM Boot ROM expires (180 seconds) and the system is reset.

If a valid R5F SBL is found, RBL loads that to MSS L2 memory, eclipse ROM to RAM and boot.

3.4.2 R5 SBL Image Size

The Maximum SBL image (without certificate) size is 952KB. The RBL for AWR294x device can only load contents for R5 L2 memory which is 960 KB in size and hence the maximum SBL is also 960 KB. The maximum SBL size is 898 KB for AWR2544 device with the certificate and 890KB without the certificate.

4 Conclusion

This application note documents the RBL boot flow and the typical SBL boot flow. The user can implement their SBL as per the suggestions and recommendations mentioned in the document. The RBL always require a signed image to work with. In context of the RBL, every image received is referred to as the SBL. In functional mode the RBL reads from the SDF at zero offset and expects a SBL to already be present at that offset. In flashing mode, RBL loads the flash writer image onto the RAM over UART and begins execution. For RBL this image can also be signed and the RBL treats this image as the SBL as well. Thus, in both the modes the RBL loads a valid image (for the SBL) onto the RAM memory of the R5F or the main sub system. The max size of the SBL or the flash writer or any other image that is loaded on the RAM memory can be 960 KB along with the attached certificate. TI also handles these various images as SBL. Hence the example flash writer image is referred as “sbl_uart_uniflash” in the MCU plus SDK package which simply downloads images onto the serial flash. These downloaded images can be another SBL and application images. Thus, in functional mode where the RBL loads a valid image form the sFLASH, RBL loads the SBL downloaded earlier. This SBL can then perform all the necessary functions as described in section 4.4 and then load the application images onto the respective cores.

5 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (April 2023) to Revision A (April 2024)	Page
• Global: Changed AWR294x to AWR294x/AWR2544	0
• Changed block text from Data Handshake Memory to DSS_MAILBOX	3

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265