User's Guide

TI mmWave xWRLx432 Bootloader Flow and Warm Reset Recommendations

Tim Henderson and Santosh Krishnan

Table of Contents

List of Figures

List of Tables

Trademarks

1 Introduction

This document details how to utilize the ROM Bootloader(RBL) in a variety of different use-cases and resolve issues that may arise from using the RBL in these scenarios. In addition, due to the rigidity of the RBL (as it is programmed into the device ROM), this document also introduces the secondary bootloader(SBL), which provides greater flexibility in what can be loaded onto the device (primary and secondary image or multiple unique user applications) and how it can be loaded (via any serial streaming interface). Lastly this document covers reset behavior, with a focus on warm reset behavior on the xWRLx432 and considerations to be made in user application design and hardware design.

This document provides a better understanding of how the application software interacts with the RBL/SBL and how it can leverage the RBL and SBL for varying levels of host-control during booting and reset procedures. Complete implementation details on the RBL and the SBL can be found in the xWRLx432 TRM.

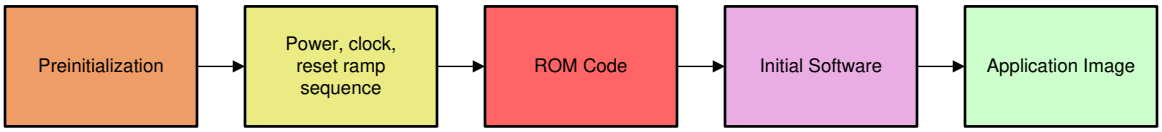

Figure 1-1. Device Initialization

Above is an overview of the initialization process and is detailed below:

- Preinitialization: Power, clock, and control connections must be present, and the boot configuration pins must be held at the desired logical levels
- Power, clock, reset ramp sequence: Specific sequence that is applied by the power-management chip
- ROM bootloader (RBL): Responsible for finding, downloading, and executing the initial software
- Initial Software: Secondary bootloader/other boot-emulation software
- Application Image: The application that runs on the main core/processor

Once the device is in functional mode, the RBL ensures that the application image is properly transferred over to RAM and hands control of the device to the user application. If this flow is interrupted by any common failure, the device goes into a safe state and will not boot the user application.

2 Basic Bootloader Flow

This section primarily focuses on the bootloader flow in functional mode. Figure 2-1 shows the bootloader flow from the start of RBL execution in functional mode.

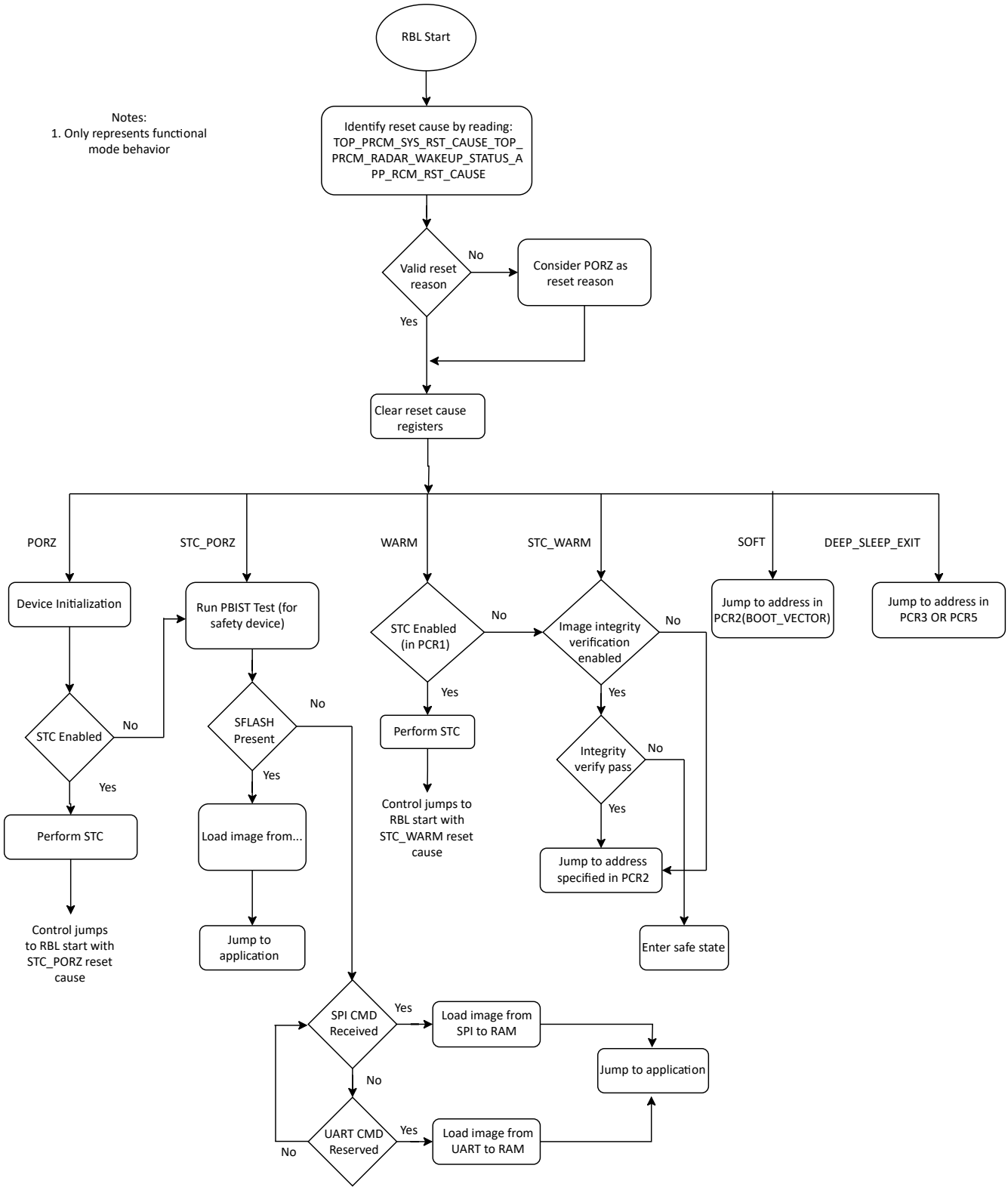

Figure 2-1. Basic Bootloader Flow Chart

Note

Load from SFLASH or download over SPI/UART is only performed in the STC_PORZ path of Figure 2-1.

2.1 Programming Serial Data Flash Over UART (Bootloader Service)

The xWRLx432 device from TI can be configured to operate as an autonomous radar sensor. In this configuration, the user application and TI firmware patches are hosted in an SDF interfaced to the xWRLx432 over the QSPI port.

SDF programming supports downloading meta images that are a combination of the following components:

- User application image for M4F (application subsystem)
- Firmware patches

The flash programmer connects to the device over UART-B. Specifics are as follows:

- UART of the xWRLx432 device:
  - RX: Ball F11/J11
  - TX: Ball E10/L12
- Baud rate: 115200
- Data length: 8bit
- Stop bit: 1bit
- Parity: None
- Maximum Data Chunk Size: 240bytes

Note

Commands ‘Write File to SFLASH’ and ‘Write File to SRAM’ support a maximum data chunk size of 240 bytes only.

The file is split into N commands where

N = (file size/240) + ((file size%240) ? 1 : 0)

2.2 Binary File Format

The target binary file is composed of the following sections:

- Header
- M4F application
- FECSS patch

The MMWAVE-L-SDK package for the xWRLx432 device from TI includes the *MulticoreImageGen* utility, which constructs the complete image with the previously listed components. See Boot Image Format under Device Initialization in the TRM for more information on the structure of the application image

2.3 Flash Programming Sequence

1. Boot the device in device management mode mode.
2. Open the *UniFlash* tool or SDK Visualizer (as listed in the MMWAVE-L-SDK for xWRLx432).
3. Connect to the device over the RS232(UART-B) application/UART port (the device expects a UART break signal – please see the xWRLx432 TRM for more details).
4. Flash the desired images <META_IMAGE1/ META_IMAGE2/ META_IMAGE3/ META_IMAGE4>
   a. For more information on Meta Images, see Secondary Bootloader

2.4 Supported UART Commands/Response and Format

2.5 Flashing Sequence

The xWRLx432 TRM contains detailed information on Flashing an application image over UART with the xWRLx432 device.

2.6 ROM-Assisted Image Download Sequence

The ROM-assisted image download sequence is entered by placing the device in flashing mode. See Programming Serial Data Flash Over UART (Bootloader Service), for further details on the handshake with an external host to receive the image. Figure 2-2 shows the communication with the serial data flash (SDF).

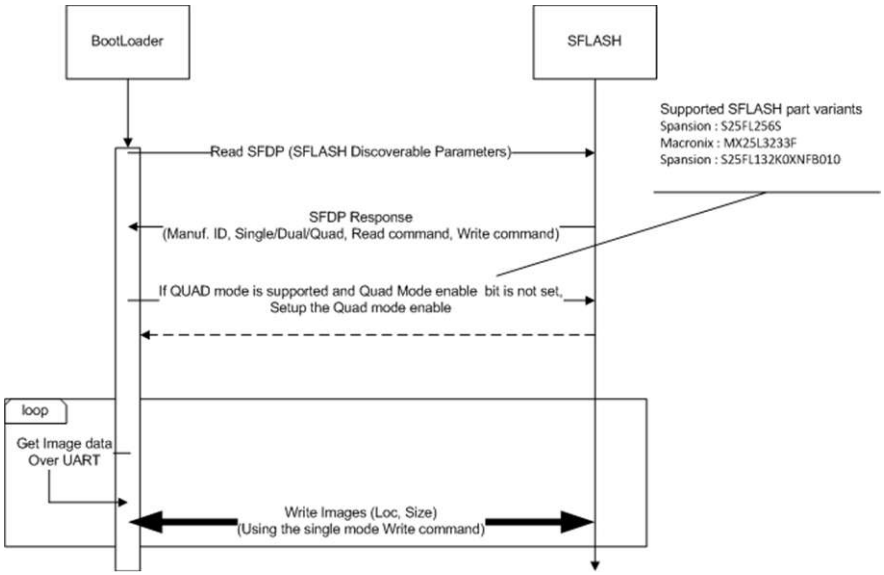

Figure 2-2. ROM-Assisted Image Download Sequence

Key points:

- The ROM-assisted download should work with all flash variants that allow for *memory-mapped mode* and *Page program command (0x2)*, with one dummy byte and 24-bit addressing.
- Setting the QE bit varies from one SDF vendor to another. The ROM bootloader supports setting the QE bit for Spansion® and Macronix® variants (certain specific part variants only) in this flow.
- In addition to a checksum-based integrity check for every packet received over the UART, a CRC32-based integrity check is performed over the complete image. The CRC32 is computed incrementally as the packets are received and written to the SDF.

2.7 Booting Application Image

2.7.1 Booting From Serial Flash

In functional mode, the bootloading of an image from the SDF is the first bootmode attempted by the bootloader (see Figure 2-3). If for some reason quad or dual read mode fails in this process, the RBL tries single mode.

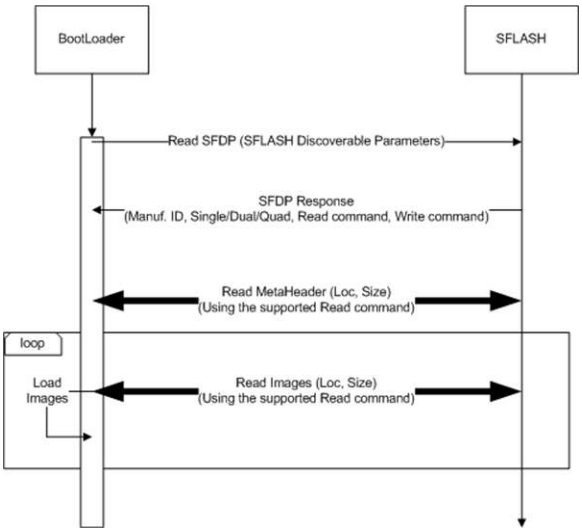

Figure 2-3. Image Load Sequence

This bootmode involves the following steps (taken care of by the ROM Bootloader):

1. Pinmux the QSPI pins of the xWRLx432 device.
2. QSPI is set up to operate at (system clock / 2) = (160/2) = 80MHz.
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.

Key points:

- The ROM bootloader 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.
- Fallback images: the bootloader supports loading of images from the following locations 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 – 0x80000)
  - META IMG3(SDF offset – 0x100000)
  - META IMG4(SDF offset – 0x180000)

For image format details, see the *AWRL6432, IWRL6432, AWRL1432, IWRL1432 Technical Reference Manual* .

2.7.2 Bootmode – SPI

When the Flash device is not present or discoverable from the JEDEC response, the bootloader looks for an image to be loaded via SPI or UART interface, if the first ping/command is initiated through SPI interface then RBL switch to SPI mode of image download. This image can be loaded in from a host PC or external processor. For more details on booting over SPI, see the *AWRL6432, IWRL6432, AWRL1432, IWRL1432 Technical Reference Manual* .

2.7.3 Bootmode - UART

When the Flash device is not present or discoverable from the JEDEC response, the bootloader looks for an image to be loaded via SPI or UARTB interface, if the first ping/command is initiated through UART interface then RBL switch to UART mode of image download.

The UART communication protocol involves a simple command-response flow. The host first sends the command packet to the xWRLx432 device and then waits for a response from the device. Multi-byte fields are transmitted in big endian format with MSB bytes transmitted followed by the LSB bytes. Each command packet is immediately processed by the bootloader without any scheduling or processing in the background and then a valid response packet is sent. The response can either be an ACK, a NACK or it may contain some specific response information. For more details on booting over UART, see the *AWRL6432, IWRL6432, AWRL1432, IWRL1432 Technical Reference Manual* .

3 Secondary Bootloader

Booting an application from SFLASH is usually done by the ROM Bootloader (RBL). The RBL looks through SFLASH, and upon reading a valid image header, load that image into RAM and start the application. However, a limitation of the RBL is that it is only able to boot one application, and provides no flexibility in changing that application as it is not configurable without modifying SOP settings on the board. Any changes that need to be made to the application require the device to be set to flashing mode to load a new application to SFLASH, and then back to functional mode to restart the boot process. The secondary bootloader, which is discussed in this section, provides the added flexibility that makes binary updates much simpler and provides the user more control and flexibility in what gets loaded and how it is loaded.

The secondary bootloader(SBL) is a tool that allows "over-the-air" binary updates to SFLASH via serial interface and subsequently execute the new application. This allows for modifications to the binaries in SFLASH without having to switch SOP modes. It also allows the user to flash multiple images: 1 main application image and 1 back-up image in case the main image fails to load. The main image is directly loaded into SFLASH via UART by the SBL application, and then boots the application. However, the SBL can be modified to use other interfaces like SPI, CAN, and LIN.

The SBL differs from the RBL in that it only focuses on booting the device with the application code rather than also initializing the device. It also can load images in multiple partitions of SFLASH, while the RBL loads the image in the first partition with a valid image. The SBL is also configurable since it is an application loaded by the RBL from FLASH as opposed to being ROM'ed on the device like the RBL. Some limitations of the SBL example in the SDK include no support for image authentication and adds complexity for systems with no host control. However, the SBL source code is available in the SDK and can be modified by the users to address these limitations and add features such as encryption to satisfy the use-case.

3.1 SBL Execution Flow

The SBL execution flow is as follows:

- The SBL application is loaded from partition 1 of SFLASH by the TI RBL

  During the execution of the SBL, it provides the user a configurable amount of time to interrupt the auto-boot process

  - The user can send the main application over a serial interface, and the SBL stores the application into the correct section of FLASH (see Flash Memory Partitioning for SBL Execution)
    - The SBL example can be found in the MMWAVE-L-SDK (for UART) and in the Radar Toolbox (for CAN/LIN)
  - Once it is flashed, the device attempts to boot the application code from FLASH and loads it into RAM, performing validity checks on the image during this process
- If the configurable timer countdown expires, the SBL auto-boots the image in partition, and if it is invalid, it auto-boots the back-up image stored in partition 4 of SFLASH (which is flashed along with the SBL)
  - The device performs validity checks on the image during the process
- If this boot is successful, the SBL updates the program counter and jump to the application that was loaded into RAM

3.1.1 Flash Memory Partitioning for SBL Execution

Typical Flash partitions for Secondary Bootloader Use Case – From SDK SBL Example

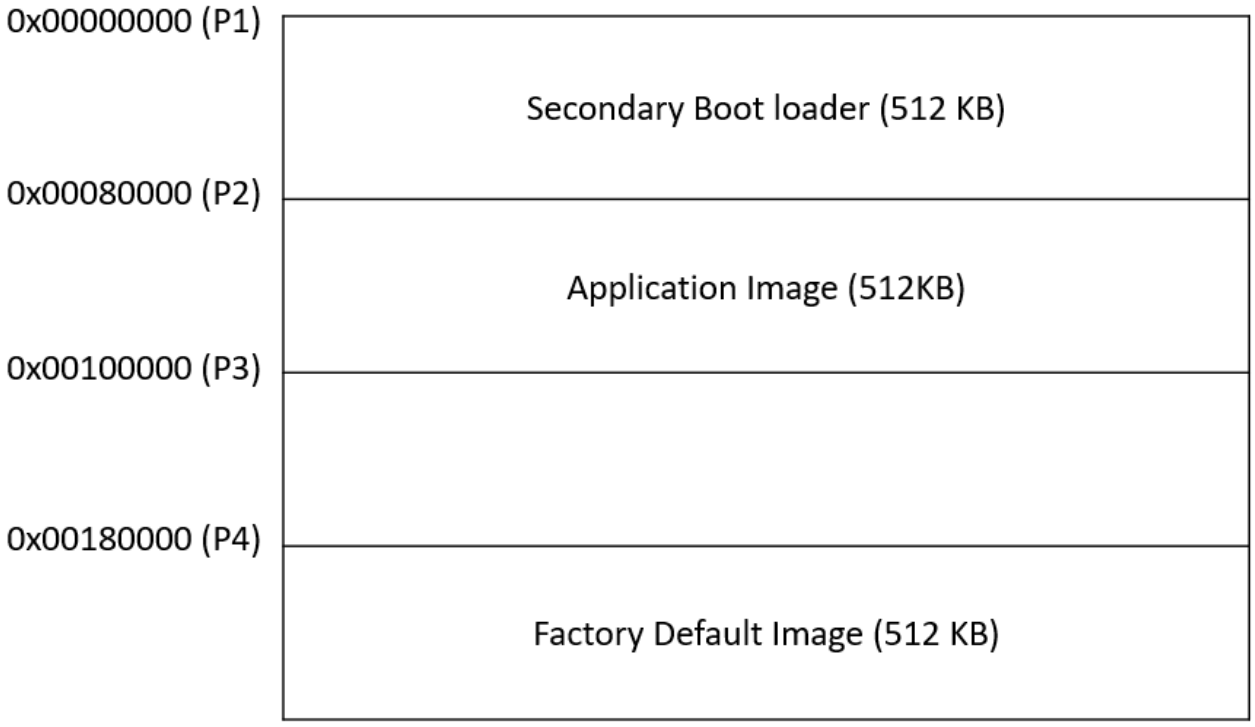

Figure 3-1. Flash Partitions for Secondary Bootloader

- The SBL application is stored in partition 1 of SFLASH so that the RBL can load this application into RAM
- The factory back-up image MUST be stored in partition 4. The main image cannot be directly loaded into partition 2 via UNIFLASH or any other flashing tool due to calibration information being saved at the start of the partition
  - The SBL application loads the main application into the correct region of partition 2 + 3 to avoid overwriting the calibration information stored in SFLASH

3.1.2 SBL Feature Modifications

Image Select:

- The SBL can be modified to provide options for which image can be loaded into RAM
  - The current flow only allows for one image and one back-up image that is automatically loaded if any issue occurs during the loading of the main image
  - This can be instead interrupted by a UART_read that waits for a user input on which section in SFLASH to pull image data from
    - Based on this selection, one of 2-3 images can be loaded as opposed to a single image
- This modification provides further flexibility in what can be loaded onto the device and provides a variety of options for the end-user to explore

3.1.3 SBL Development Considerations

Shared Memory Considerations

- The SBL example runs from shared memory to avoid conflicting with any application code in the M4 RAM.
- When loading an application with the SBL, the application cannot store any text, data, or read-only information in shared memory. For more information, see the DIG#14 in the errata. However, examples using L3 memory in shared memory can still utilize shared memory.

General Development Considerations

- IWR devices require RAM1 memory bank to be explicitly initialized due to EFUSE differences between IWR and AWR devices
  - IWR devices have ECC disabled by default, therefore requiring explicit memory initialization
  - It is good practice to initialize all the memory banks, but results in a delay in boot time
    - Can also be resolved by disabling the SYSTICK timer which was causing unnecessary access to memory banks upon execution of the ISR
- If executing a warm reset on an application that was loaded by the SBL, please make sure the warm reset considerations are executed by the SBL and the SBL triggers the warm reset
  - Specifically required for watchdog resets in the application that trigger a warm reset as it is an uncontrolled reset that may not handle all the necessary steps for ensuring a successful warm reset upon exit of the program

4 Warm Reset

On a STC_WARM or WARM reset reason, as seen by the RBL flow in Figure 2-1, the appropriate STC tests and image verification is performed and then the execution jumps to the address stored in the boot vector (TOP_PRCM:PC_REGISTER2). This address is typically the start of the user application. In this flow, there is necessary HW infrastructure provisioned to ensure that the memory contents are retained on a warm reset cycle with the power to the device maintained to be intact. The RBL will not by default reload an image from SDF in this flow.

A Warm reset is internally generated by the device, or triggered by device pin WARM_RESET. A write to the TOP_PRCM:RST_SOFT_RESET register generates the reset. The WARM_RESET can be used to reset the device from the external world or to report the reset to the external world if it is generated by an internal source such as watchdog timer.

4.1 Integrity Verification

The RBL performs an integrity verification step to ensure that an application has valid data. The RBL makes the following assumptions when performing the integrity check in the STC_WARM reset flow:

- RBL reads the boot vector from TOP_PRCM:PC_REGISTER2[24:0] to determine the boot vector to jump to
- RBL performs the integrity check for the APPSS image only
- RBL computes the CRC of the APPSS image based on image length (starting from boot vector) field in TOP_PRCM:PC_REGISTER3[31:0] and compares it with the data integrity check field TOP_PRCM:PC_REGISTER4[31:0]. It is expected that the application populates this register
- The parity of TOP_PRCM:PC_REGISTER3 and TOP_PRCM:PC_REGISTER4 is expected to be correctly populated in TOP_PRCM:PC_REGISTER1[23:20] and TOP_PRCM:PC_REGISTER2[19:16] by the application. For more information, see Section 5.2.

4.2 LSTC/PBIST

Programmable built-in self test (PBIST) is a feature that is used for self-test the memory regions in the SoC without any external test equipment. In an embedded system, these tests are typically used during boot time or shutdown of the system to check the health of an SoC.

Logic built-in self test controller (LSTC) IP addresses the logic self-test requirements for functional safety applications as per ISO-26262 ASIL compliance. The LBIST is triggered through the RBL during power-up and/or during a warm reset if enabled, and if a functional safety rated device is being used.

For more details, see the *Safety Modules* section of the *AWRL6432, IWRL6432, AWRL1432, IWRL1432 Technical Reference Manual* .

4.3 Watchdog Timer

A watchdog timer module is available on the xWRLx432 device. A watchdog timer can act as a last method of defense against runaway code or infinitely looping code. This can prevent the device from entering an unknown state and can result in unknown power draw or loss of communication within a larger system.

The WDT does not provide coverage when the device is in low power/deep sleep mode

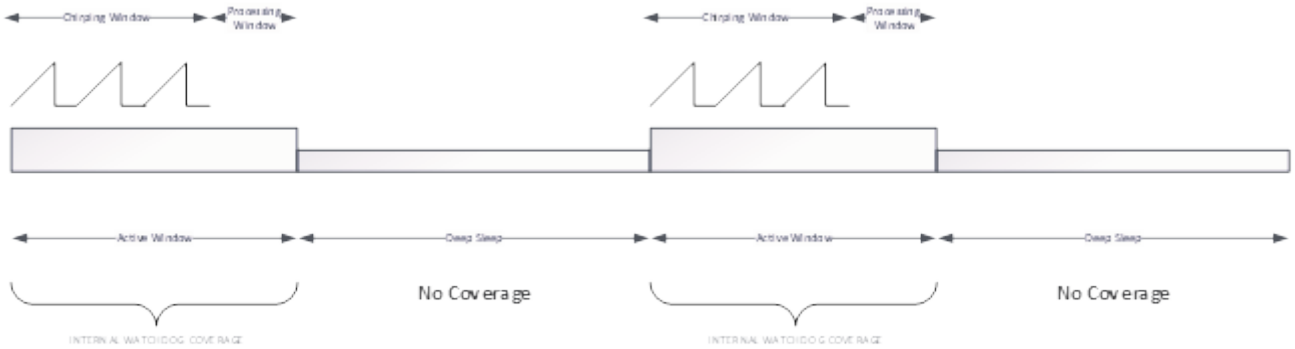

Figure 4-1. xWRLx432 WDT Coverage

The internal watchdog is not meant to cover all failure modes on the device, which includes but is not limited to random bit flips or transient faults. An appropriate functional safety assessment must be taken by the system integrator to determine if the internal watchdog is sufficient for the use-case.

4.4 Reset-Triggered Flash Reload of Application

Depending on an application use-case, a full reload of user application may be desired upon a warm reset of the device. This ensures a "clean" start for the application image and would recover the device from any memory corruption issues that may have caused runaway code and a watchdog warm reset in the first place.

The RBL, by design, does not support loading from flash on a warm reset. It is thus recommended that the customer take appropriate care in the design of their hardware PCB if this behavior is desired.

4.4.1 Hardware Solutions

Multiple Hardware solutions exist to create a reload from Flash on a reset of the device. Two of which are listed below

4.4.1.1 PMIC I2C Messaging

Depending on the PMIC chosen in the system, an I2C message can be sent to the PMIC to trigger a NRESET via the power supply. This would provide a clean restart of the device and reload the flash contents from FLASH memory. For example: PMIC TPS650360 can be used for this purpose, presuming reset out pin of PMIC is tied to reset pin of mmWave radar device. The mmWave radar device can request warm reset of PMIC through I2C message and then PMIC will reset the mmWave radar device.

4.4.1.2 External Watchdog Timer

External watchdog timer can be used to monitor the mmWave radar device for detecting the software codes (Application code, driver code, and so forth) from entering the lockup state or increase in implementation time. The external watchdog timer can be serviced from the GPIO's of the mmWave radar device. If the external watchdog timer is not serviced regularly as expected then the timer can trigger the NRESET signal to the device. On reset from external watchdog timer, mmWave radar device would provide a clean restart of the device and reload the flash contents from FLASH memory. One of the examples for the external watchdog timer is TPS3430 - Automotive Window Watchdog Timer with Programmable Reset Delay. The watchdog timer can also be used to detect and recover from a hardware fault.

4.4.1.3 External Voltage Monitoring or Voltage Supervisors

These are also known as the reset ICs and the voltage monitors. These devices act as external voltage monitors which supervises or monitors the voltage rails and assert a signal to reset, enable or disable another device on detecting the voltage rail deviated from its permissible tolerance (under-voltage and over-voltage thresholds). This reset to mmWave radar device can provide a clean restart of the device and reload the flash contents from FLASH memory. One of the voltage supervisor examples is TPS3850, which can be used for detecting under-voltage and over-voltage of 1.2V, 1.8V and 3.3V power rails.

4.4.2 Software Solutions

There are software-based workarounds that can be considered should a customer want to reload an image from flash on a warm reset. Both are detailed below.

4.4.2.1 Setting Boot Vector to 0x0

A software-based workaround to load the image from Flash is possible by modifying the jump address used in the warm reset flow. Proper care must also be taken to ensure that the FECSS (Front End Controller Subsystem) is in the correct state. A user must perform any necessary safety assessment of their own system to determine if this software-based workaround meets their needs. The flow of the workaround sequence is as follows:

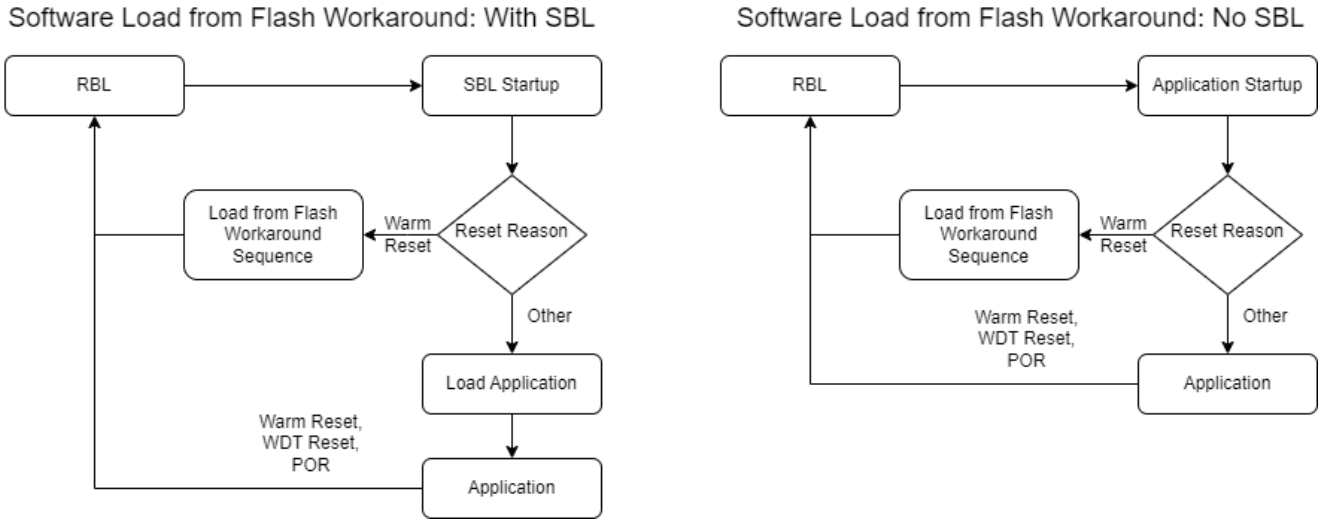

Figure 4-2. Software Load From Flash on a Warm Reset

1. Gracefully power down FECSS and HWASS
2. Power on FECSS
3. Write Boot vector to TOP_PRCM:PC_REGISTER2[24:0] = 0
4. Write boot vector parity to TOP_PRCM:PC_REGISTER1[15:12] = 0
5. Disable image integrity check - TOP_PRCM:PC_REGISTER1[5:3] = 0
6. Disable STC test - TOP_PRCM:PC_REGISTER1[8:6] = 0
7. Trigger warm reset - TOP_PRCM:RST_SOFT_RESET[0] = 1

Execution re-enters the RBL after the warm reset in step 7 occurs, which starts execution as shown in Figure 2-1. The RBL determines the reset reason as invalid because the reset registers have been cleared. It assumes a POR because of this, and the application image is loaded from Flash once again. The application or SBL can continue as designed, until another reset occurs.

View the watchdog_reset example in the MMWAVE-L-SDK under: `<SDK-Install-Directory>/examples/drivers/watchdog/watchdog_reset/` for the fully-coded sequence to perform the reload from flash.

5 Relevant Registers

After initialization, RBL clears all reset registers, and stores the reset register values in this register.

5.1 Reset Registers

The reset registers can be used to identify if the type of reset that was triggered matches the intended reset the user wanted to trigger. Please see the tables below for descriptions of these registers and what the values in those register represent.

Table 5-1. SYS_RST_CAUSE Register

wWRLx432:TOP_PRCM:SYS_RST_CAUSE	Filed Name	Description
Bit#16	SYS_RST_CAUSE_SYS_RST_CAUSE_CLR	Clear's the sys_rst_cause register 0x0 -> sys_rst_cause capture enable 0x1 -> sys_rst_cause reg clear and disable
Bit#<2:0>	SYS_RST_CAUSE_SYS_RST_CAUSE	System Reset Cause register 3'b001 - POR reset 3'b010 - Warm reset due to soft register 3'b100 - Warm reset due to wdog

Note

On a STC_POR reset, the SYS_RST_CAUSE register will be 0x0. This field will only be set to 3'b001 on the true POR reset.

Table 5-2. RADAR_WAKEUP_STATUS Register Field Descriptions

Bit	Field	Type	Reset	Description
31-21	RESERVED	R/W	X	
20	radar_state_is_deep_sleep	R	0h	RADAR power FSM is in DEEP_SLEEP state
19	radar_state_is_go_to_deep_sleep	R	0h	RADAR power FSM is in GO_TO_DEEP_SLEEP state
18	radar_state_is_sleep	R	0h	RADAR power FSM is in SLEEP state
17	radar_state_is_idle	R	0h	RADAR power FSM is in IDLE state
16	radar_state_is_wake_up	R	0h	RADAR power FSM is in WAKEUP state
15-9	RESERVED	R/W	X	
8	wakeup_status_clear	R/W	0h	Clear's the wakeup status and source register 0x 0 -> Wakeup status and source capture enable 0x 1 -> Wakeup status and source reg clear and disable
7-2	wakeup_source	R	0h	It indicate wakeup source from SLEEP/DEEP SLEEP state Bit 0 -> Sleep counter as Wakeup source Bit 1 -> UART as Wakeup source Bit 2 -> SPI as Wakeup source Bit 3 -> GPIO as Wakeup source Bit 4 -> RTC counter as Wakeup source Bit 5 -> FRC frame start intr as Wakeup source
1-0	wakeup_status	R	0h	It indicates the wakeup status of the device, whether it is Wakeup due to POR or from SLEEP/DEEP SLEEP state 0x 1 -> Wakeup from SLEEP 0x 2 -> Wakeup from DEEP SLEEP

Table 5-3. RST_CAUSE Register Field Descriptions

Bit	Field	Type	Reset	Description
31-8	RESERVED	R	X	
7-0	COMMON	R	3h	Reset cause register for APP CPU 0x00 - All cleared 0x01 - Power On Reset (PoR) 0x02 - Subsystem Reset (Combination of Warm Reset initiated from PRCM using xWRLx432:TOP_PRCM:RST_APP_PD_SOFT_RESET and PoR reset) 0x04 - STC RESET 0x08 - Reserved 0x10 - CPU Only Reset triggered by writing to xWRLx432:APP_RCM:RST_FSM_TRIG<RST_FSM_TRIG_CPU> 0x20 - Core Reset initiated from PRCM using xWRLx432:TOP_PRCM:RST_SOFT_APP_CORE_SYSRESET_REQ (reset CPU unconditionally - by debugger) or xWRLx432:TOP_PRCM:APP_CORE_SYSRESET_PARAM_WAKEUP_OUT_STATE 0x40 - Reserved

Bit	Source
[3:0]	**Reset Reason Identification by bootloader** M_BOOT_RESET_REASON_PORZ: (0x1U) M_BOOT_RESET_REASON_PORZ: (0x1U) M_BOOT_RESET_REASON_WARM: (0x2U) M_BOOT_RESET_REASON_DEEPSLEEP: (0x3U) M_BOOT_RESET_REASON_SOFT: (0x4U) M_BOOT_RESET_REASON_STC_WARM: (0x5U) M_BOOT_RESET_REASON_STC_PORZ: (0x6U)
[7:4]	SYS_RST_CAUSE[2:0]
[15:7]	RADAR_WAKEUP_STATUS[7:0]
[23:16]	RST_CAUSE[7:0]

5.2 PC Registers

Figure 5-1 shows the organization of relevant PC registers. These registers contain important information that is used by the RBL.

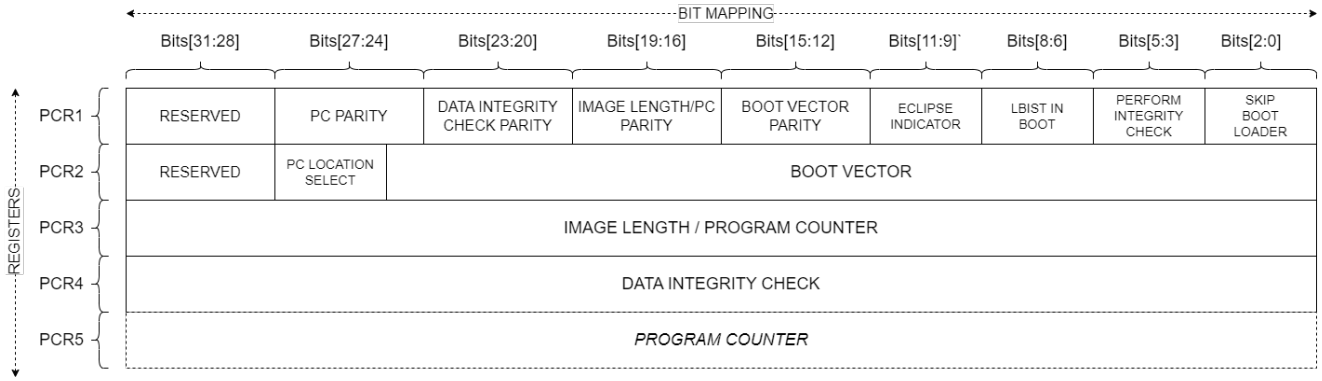

Figure 5-1. Relevant PC Register Formats

5.2.1 Addresses

PC_REGISTER1 Register (Physical Address = 5A040054h) [Reset = 00000000h]

PC_REGISTER2 Register (Physical Address = 5A040058h) [Reset = 00000000h]

PC_REGISTER3 Register (Physical Address = 5A04005Ch) [Reset = 00000000h]

PC_REGISTER4 Register (Physical Address = 5A040060h) [Reset = 00000000h]

PC_REGISTER5 Register (Physical Address = 5A040064h) [Reset = 00000000h]

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