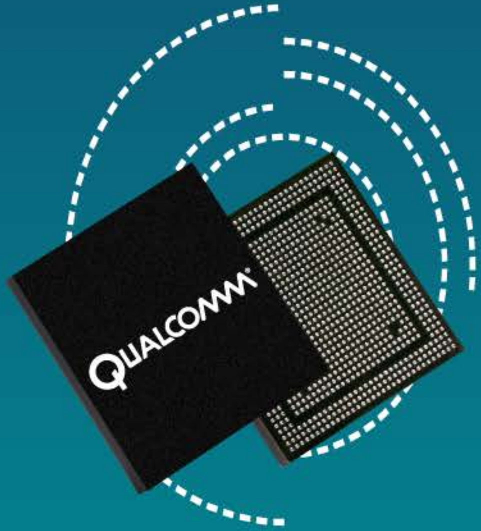


MSM8909 Boot Architecture Overview

80-NR964-3 B

Confidential and Proprietary – Qualcomm Technologies, Inc.

NO PUBLIC DISCLOSURE PERMITTED: Please report postings of this document on public servers or websites to: DocCtrlAgent@qualcomm.com.

Restricted Distribution: Not to be distributed to anyone who is not an employee of either Qualcomm or its subsidiaries without the express approval of Qualcomm's Configuration Management.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

Qualcomm reserves the right to make changes to the product(s) or information contained herein without notice. No liability is assumed for any damages arising directly or indirectly by their use or application. The information provided in this document is provided on an "as is" basis.

This document contains confidential and proprietary information and must be shredded when discarded.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

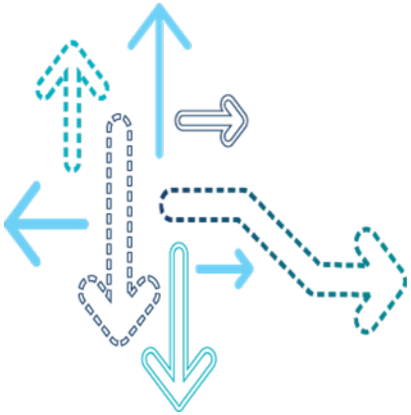
Revision History

Revision	Date	Description
A	Sep 2014	Initial release
B	Oct 2014	The following changes were made: ▪ Added slide 7 ▪ Updated slides 31, 32, and 34

Contents

- MSM8909 Boot Architecture
- MSM8909 PBLs
- Boot Loader Customization
- Watchdog Reset Debug
- References
- Questions?

MSM8909 Boot Architecture



PBL Boot Option – Major Boot Difference Between MSM8909, MSM8916, and MSM8x10/MSM8x12

- The Boot_config [0..4] GPIOs or BOOT_CONFIG fuses can be used to select the following boot options. Once the fuse is blown, the GPIOs used for the boot option are free to be used as common GPIOs.

BOOT_CONFIG	MDM8909	Comments
0x00	BOOT_DEFAULT_OPTION	eMMC @ SDC1 → HS USB 2.0
0x01	BOOT_SDC_PORT2_THEN_SDC_PORT1_OPTION	SD @ SDC2 → eMMC@SDC1
0x02	BOOT_eMMC_OPTION	eMMC@SDC1
0x03	BOOT_USB_OPTION	HS USB 2.0
0x04	BOOT_NAND_OPTION	NAND boot

- OEMs must select default boot device as eMMC boot or NAND boot based on FAST BOOT fuses blown. The eMMC GPIO interface is MUXed with NAND GPIOs for bootable device (both are mutual exclusive).
- 0x00 is always the default boot option for all earlier MSM™ chipsets but in MSM8909 chipset eMMC and NAND boot are mutually exclusive.

Boot Config GPIO information for eMMC and NAND

eMMC pin	GPIO number
SDC1_CLK	100
SDC1_DAT[7]	101
SDC1_DAT[6]	102
SDC1_DAT[5]	103
SDC1_DAT[4]	104
SDC1_DAT[3]	105
SDC1_DAT[2]	106
SDC1_DAT[1]	107
SDC1_DAT[0]	108
SDC_CMD	109

NAND pin	GPIO number
EBI2_OE_N	100
EBI2_ALE_N	101
EBI2_CLE_N	102
EBI2_A_D[4]	103
EBI2_A_D[5]	104
EBI2_A_D[7]	105
EBI2_A_D[6]	106
EBI2_A_D[3]	107
EBI2_WE_N	108
EBI2_BUSY_N	109
EBI2_A_D[2]	8
EBI2_A_D[1]	9
EBI2_A_D[0]	10
EBI2_NAND_C_S_N	11

Note: Do not pull up GPIO by default. Drive strength, pull values, and directions are controlled by the following registers:

- HWIO_TLMM_EBI2_EMMC_GPIO_CFG
- HWIO_TLMM_SDC1_HDRV_PULL_CTL

Other Key Boot Differences Between MSM8909, MSM8916, and MSM8x10/MSM8x12

	MSM8909	MSM8916	MSM8x10/MSM8x12
Boot processor	APPS CPU processor – Cortex-A7 (ARMv7-based CPU)	APPS CPU processor – Cortex-A53 (A53 ARMv8-based CPU)	RPM Processor (cortex-M3)
Boot ROM	PBL supports ELF load	PBL supports ELF load	Supports MBN format
Boot devices/interfaces	eMMC, HSUSB, SD and NAND	eMMC, HSUSB, SD	eMMC, HSUSB, SD
Default boot option	eMMC @ SDC1 →USB SBL image is loaded from either NAND or eMMC based on fast boot fuses blown.	eMMC @ SDC1→SD @ SDC2→USB	eMMC @ SDC1→SD @ SDC2→USB
Aarch mode	AArch32 mode only	AArch32 mode and AArch64 mode	AArch32 mode only.
SBL internal memory usage	L2 as TCM and RPM code-RAM	L2 as TCM and RPM code-RAM	L2 as TCM and IMEM
SBL loads	SBL loads: • QSEE, RPM_FW, and APPSBL • SDI functionality is merged with SBL and TZ, which is a different image loaded by SBL in MSM8x10/MSM8x12 • SBL1 supports 32-bit ELF load (RPM_FW is not applicable)	SBL loads: • QSEE, QHEE, RPM_FW, and APPSBL • SDI functionality is merged with SBL and TZ, which is a different image loaded by SBL in MSM8x10/MSM8x12. • SBL1 supports 32-bit and 64-bit ELF load (RPM_FW is not applicable)	QSEE, SDI, RPM_FW, and APPSBL

Other Key Boot Differences Between MSM8909, MSM8916, and MSM8x10/MSM8x12 (cont.)

	MSM8909	MSM8916	MSM8x10/MSM8x12
RPM_FW execution	RPM_FW starts execution when QSEE brings RPM CPU (Cortex-M3) out of reset	RPM_FW starts execution when QHEE running at AP CPU (Cortex-A53) in EL2 brings RPM CPU (Cortex-M3) out of reset	RPM_FW starts execution by a handshake with SBL running at AP CPU
Hash checking in boot ROM for ELF segments	Hash checking is made by default in boot ROM to check the data integrity of the loaded SBL	Hash checking is made by default in boot ROM to check the data integrity of the loaded SBL	Not supported
Battery charging	Battery charging @ SBL	Battery charging @ SBL	Battery charging @ SBL
APPS-CPU HYP Mode	HYP mode is not used	HYP mode is used	HYP mode is not used

Note: Refer [Q2] for MSM8916 boot architecture details.

Boot Address for MSM8909 Processors

- There are different processors in the MSM8909 chipset. The following table lists the processor types and boot addresses:

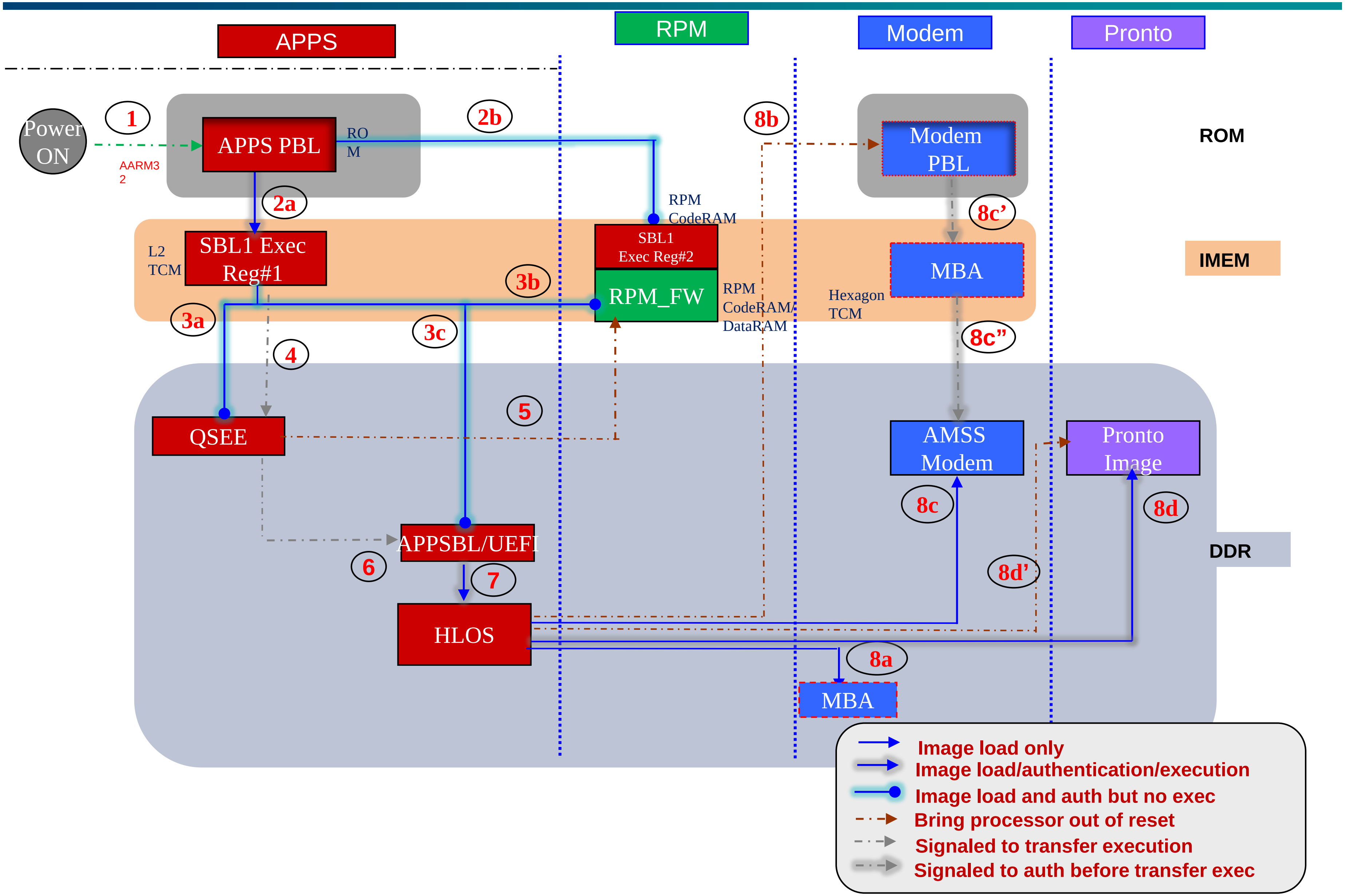
Subsystem	Processor	Boot address	
APPS	Cortex-A7	0xFC010000*	
RPM	Cortex-M3	$0x00200000_{\text{(Subsystem view)}}$	$0x0_{\text{(System view)}}$
Modem	MSS_QDSP6	Configurable*	
Pronto	ARM9™	0x0 or 0xFFFF0000 or hardware remap*	

*No change in boot address of System and Subsystem views.

Boot Call Stack

Component	Based on processor	Loaded from	ZI/RW allocated in	Executes in	Function
APPS PBL (Application Primary Boot Loader)	Cortex-A7	Not Applicable	L2 TCM, RPM CodeRAM	ROM	Boot device and interface detection, Emergency Download mode support, and load and authenticate SBL1 ELF segments across L2 TCM and RPM CodeRAM
SBL1 (Secondary Boot Loader Stage 1)	Cortex-A7	NAND/eMMC	L2 TCM, DDR	L2 TCM	Init memory subsystem (buses, DDR, clocks, and CDT), load/authorize TZ, RPM_FW, APPSBL images, memory dump via USB2.0 and Sahara, Watchdog debug retention (e.g., L2 flush), RAM dump to eMMC/SD support, Mass Storage mode support, USB driver support, USB charging, thermal check, PMIC driver support, configure DDR, and flush L1/L2/ETB to crash debug support related configuration
			OCIMEM		
			RPM CodeRAM	RPM CodeRAM	
QSEE (Secure Exec Environment)	Cortex-A7	NAND/eMMC	LPDDR2/3	LPDDR2/3	Equivalent to TZBSP. Setup secure runtime execution environment, configure xPU, support fuse driver, setup SMMU configurations, SDI-logic-after-wdog-reset to flush L1/L2/ETB for crash debug support-related configuration for non-production OR debug-enabled device.
RPM_FW	Cortex-M3	NAND/eMMC	RPM DataRAM/ MessageRAM	RPM CodeRAM	Resource Power Manager
APPSBL image	Cortex-A7	NAND/eMMC	LPDDR2/3	LPDDR2/3	HLOS-specific feature-rich bootloader (LK), load/auth Kernel, recovery mode, etc.
Modem PBL (Modem Primary Boot Loader)	MSS Q6	Not applicable	MSS Q6 TCM	ROM	Setup Q6 TCM, load MBA from LPDDR2 into Q6 TCM, authenticate MBA in Q6 TCM, and lock SMMU CB
MBA (Modem Boot Authenticator)	MSS Q6	NAND/eMMC	MSS Q6 TCM	LPDDR2/3 and Q6 TCM	Authenticates the modem image, xPU protects the DDR regions for Modem

Boot Code Flow



- Image load only
- Image load/authentication/execution
- Image load and auth but no exec
- Bring processor out of reset
- Signaled to transfer execution
- Signaled to auth before transfer exec

MSM8909 Boot Flowchart

1. The system powers on and takes MSM8909 AP CPU out of reset.
2. In Cortex-A7 APPS PBL executes

   (a) Loads, Execute and authenticates the SBL1 segment #1 from the boot device to L2 (as TCM).

   (b) Loads, Execute and authenticates SBL1 segment #2* (DDR/SDI equivalent) to RPM code RAM, then jumps to SBL1.
3. SBL1#1

   (a) Loads and authenticates the QSEE/TZ image from the boot device to DDR.

   (b) Loads and authenticates the RPM firmware image from the boot device to RPM code RAM.

   (c) Loads and authenticates the HLOS APPSBL image from the boot device to DDR.
4. SBL1 #1 transfers execution to QSEE/TZ.
5. QSEE/TZ set up secure environment and bring RPM out of RESET to start execution of RPM firmware.
6. QSEE/TZ jumps to HLOS APPSBL to start execution.

- *SBL1 segment#2 is equal to DDR driver + SDI equivalent copied to RPM code RAM.
- DDR is initialized by SBL1 segment#2 and part of the SDI functionality included in SBL1 segment#2. For more details, refer to watchdog reset debug slides (slide 30 to 33).

MSM8909 Boot Flowchart

7. HLOS APPSBL loads and authenticates the HLOS kernel.
8. HLOS kernel:

(a) Loads the Modem Boot Authenticator (MBA) to DDR via PIL.
(b) Brings modem DSP Q6 out of reset.
(c) Loads the AMSS modem image to DDR via PIL.
(c’) Modem PBL copies the MBA from DDR to modem TCM and authenticates MBA and jump to MBA image.
(c”) MBA authenticates modem image and then jumps to modem.
(d) HLOS loads the Pronto image to DDR via PIL.
(d’) HLOS brings Pronto out of reset and Pronto image starts execution.

SBL-QSEE Interface

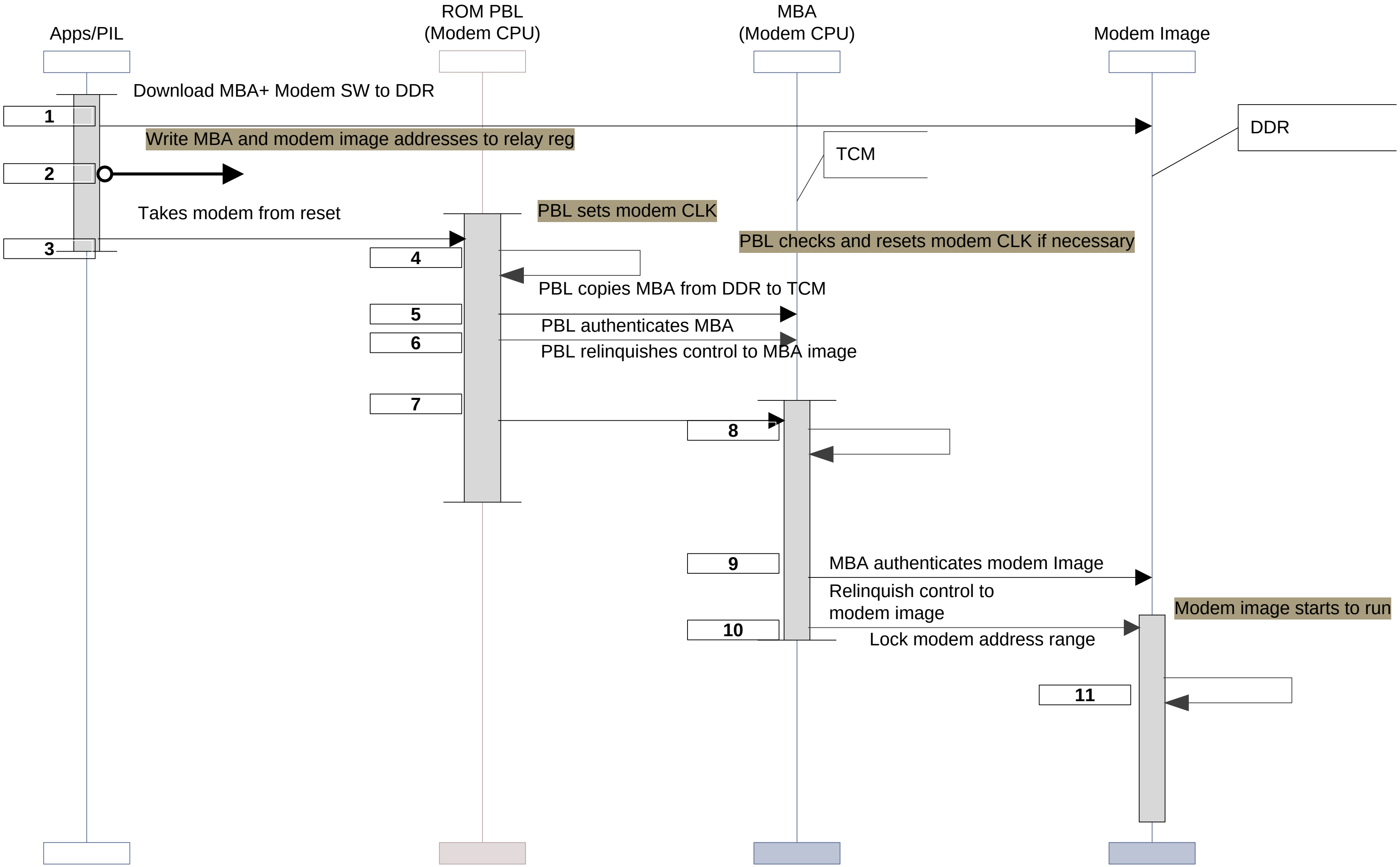
SBL loads and authenticates different boot images and hands over the control to QSEE. SBL populates image information as captured in the below structure for all images loaded by SBL, and then passes to QSEE.

```
typedef struct boot_sbl_qsee_interface
{
  uint32 magic_1;
  uint32 magic_2;
  uint32 version;
  uint32 number_images;
  uint32 reserved_1;
  boot_images_entry
  boot_image_entry[BOOT_IMAGES_NUM_ENTRIES];

} boot_sbl_qsee_interface;
```

```
boot_images_entry
{
  secboot_sw_type image_id;
  uint32 e_ident;
  uint64 entry_point;
  secboot_verified_info_type  image_verified_info;
  uint32 reserved_1;
  uint32 reserved_2;
  uint32 reserved_3;
  uint32 reserved_4;

} boot_images_entry;
```

```
typedef struct
secboot_verified_info_type
{
  uint32      version_id;
  uint64      sw_id;
  uint64      msm_hw_id;
  uint32      enable_debug;
  secboot_image_hash_info_type
  image_hash_info;
  uint32 enable_crash_dump;
} secboot_verified_info_type
```

Independent Modem Authentication Flow



Apps/PIL

ROM PBL (Modem CPU)

MBA (Modem CPU)

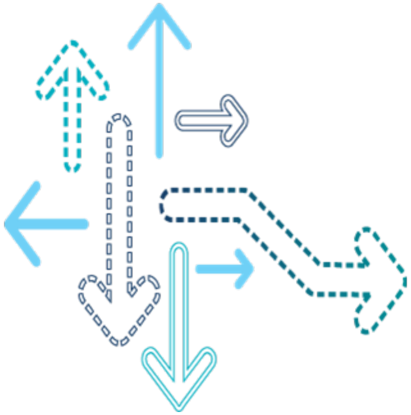
Modem Image

1. Download MBA+ Modem SW to DDR
2. Write MBA and modem image addresses to relay reg
3. Takes modem from reset
4. PBL sets modem CLK; PBL checks and resets modem CLK if necessary
5. PBL copies MBA from DDR to TCM
6. PBL authenticates MBA
7. PBL relinquishes control to MBA image
8.
9. MBA authenticates modem Image
10. Relinquish control to modem image
11. Lock modem address range; Modem image starts to run

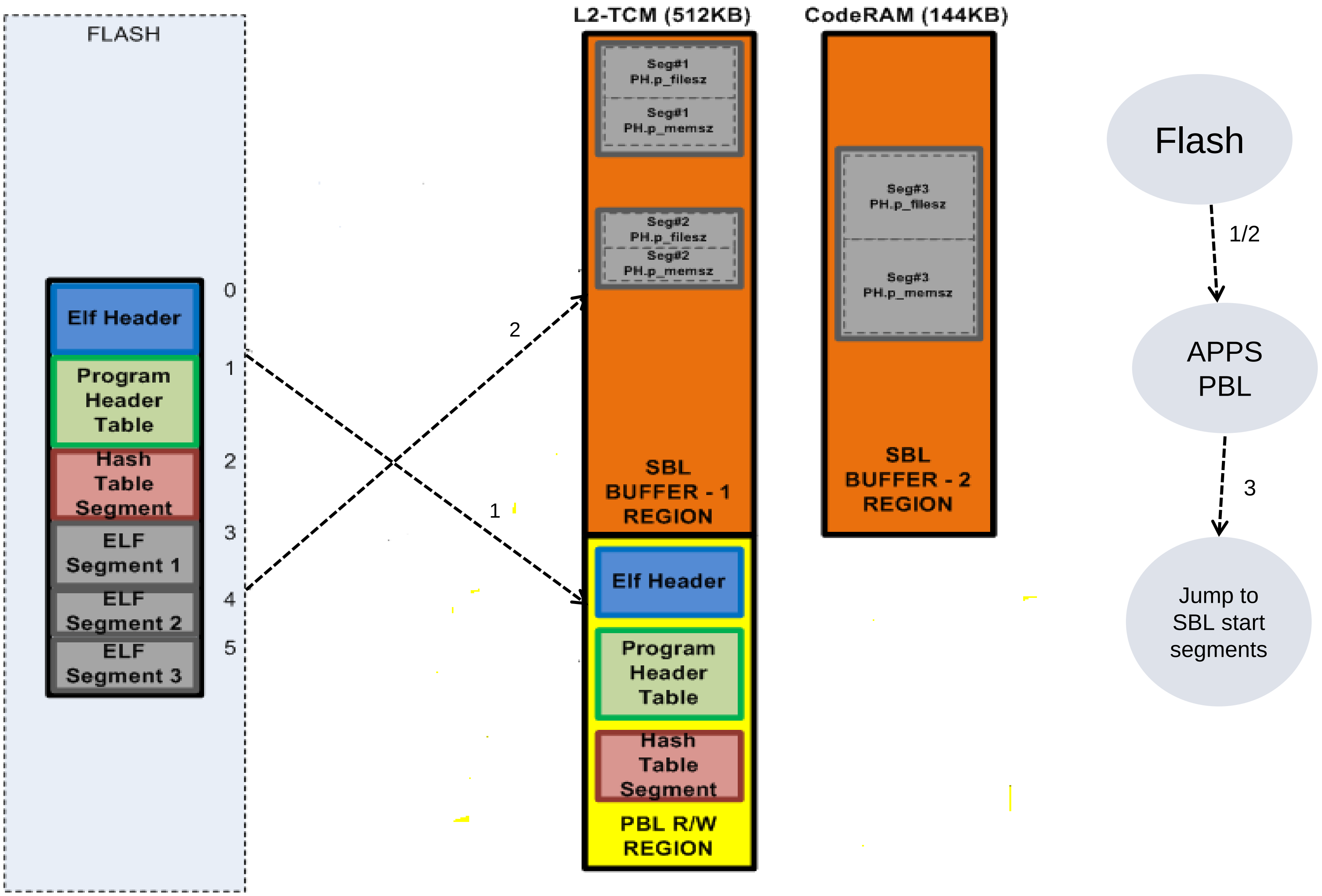
Boot Flowchart – Independent Modem Authentication

1. APPS HLOS downloads the MBA and modem image to DDR
2. APPS HLOS writes MBA and modem image address to RMB registers
3. APPS HLOS takes the modem out of reset
4. Modem PBL executes and sets its own clock
5. Modem PBL copies the MBA from DDR to modem TCM
6. Modem PBL authenticates the MBA
7. Modem PBL relinquishes control to the MBA
8. MBA closes the public domain and locks the modem address range
9. MBA authenticates the modem image
10. MBA relinquishes control to the modem image
11. Modem image starts to run

MSM8909 PBLs



Apps PBL Boot – SBL ELF Loading



FLASH

Elf Header

Program Header Table

Hash Table Segment

ELF Segment 1

ELF Segment 2

ELF Segment 3

0

1

2

3

4

5

L2-TCM (512KB)

Seg#1 PH.p_filesz

Seg#1 PH.p_memsz

Seg#2 PH.p_filesz

Seg#2 PH.p_memsz

SBL BUFFER - 1 REGION

Elf Header

Program Header Table

Hash Table Segment

PBL R/W REGION

CodeRAM (144KB)

Seg#3 PH.p_filesz

Seg#3 PH.p_memsz

SBL BUFFER - 2 REGION

Flash

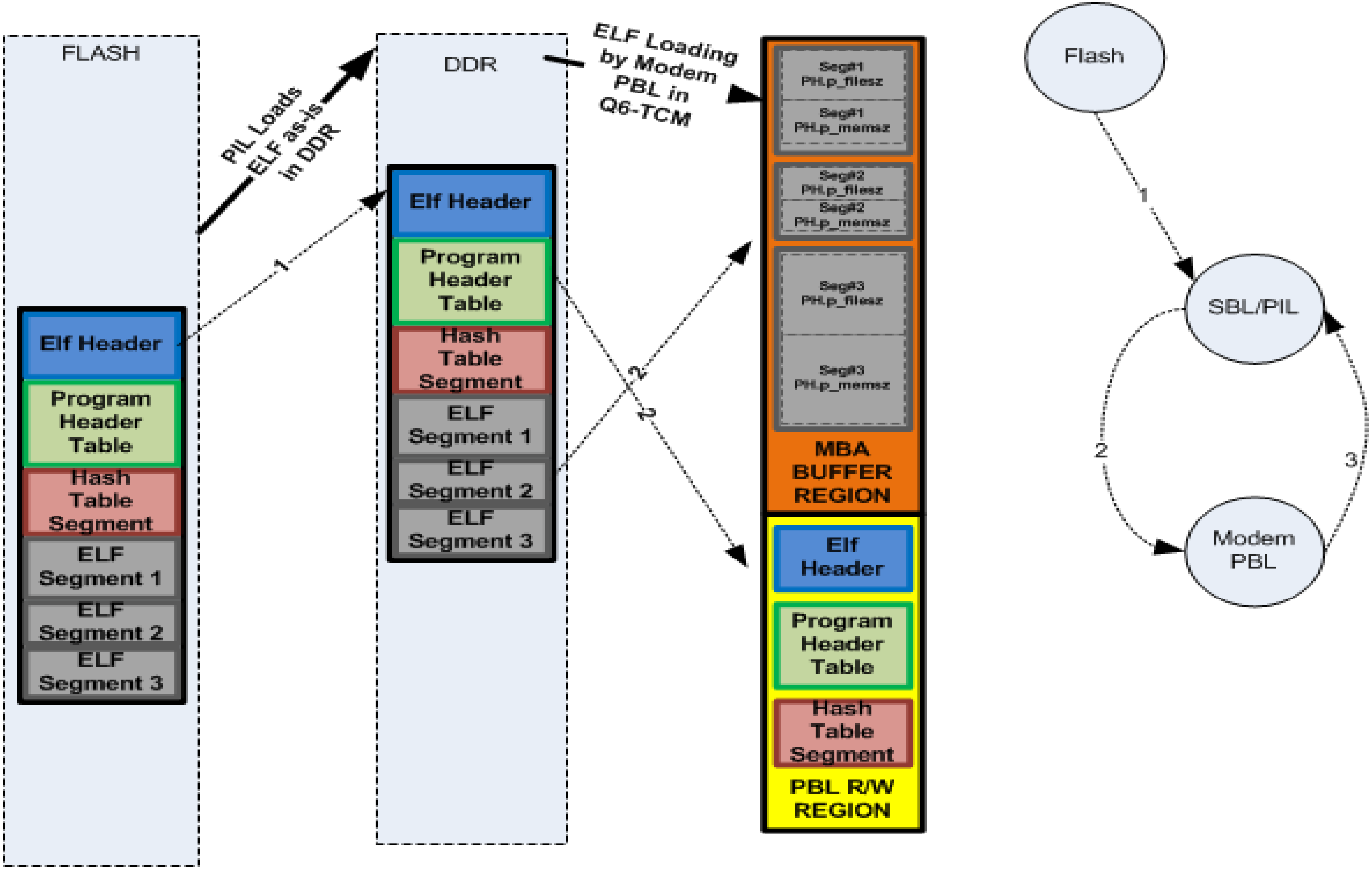
1/2

APPS PBL

3

Jump to SBL start segments

Modem PBL Boot – MBA ELF Loading



PBL Boot – ELF Loading

- Apps PBL ELF loading of SBL image
  - APPS PBL starts ELF loading by verifying the ELF headers, loads program headers, and then authenticates hash segments
  - APPS PBL loads the loadable segments and verifies hashes for each of those segments
- Modem PBL ELF loading of MBA image
  - Before bringing Hexagon™ processor out of RESET, PIL loads MBA ELF as is in DDR and writes start address to RMB0
  - Modem PBL starts loading MBA ELF from DDR to Hexagon L2 TCM
  - Modem PBL first loads and verifies ELF headers, loads program headers and hash segments, and then authenticates hash segments
  - Modem PBL loads loadable segments and verifies hashes for those segments

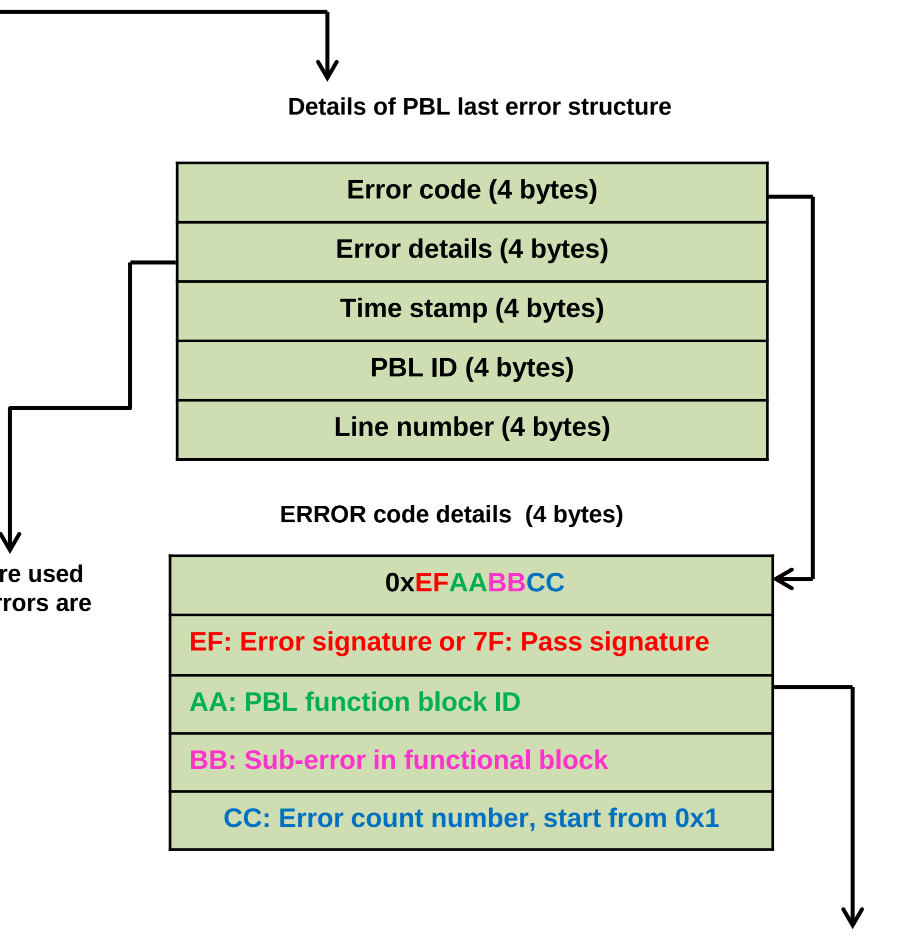
PBL Error Logging

- APPS PBL − Saves the error information in RPM code RAM
  1. Once a PBL error occurs, it goes to the error handler.
  2. PBL logs the necessary log information into the RPM code RAM.
  3. PBL goes to Emergency Download mode if it is not disabled by the fuse.
  4. In Emergency Download mode, PBL enters the Sahara protocol to receive and authenticate the flash programmer from the host.
  5. Once loaded and authentications are passed, the system jumps to the flash programmer start address.
  6. The flash programmer executes and downloads the necessary boot images from the host side.
- Modem PBL – Updates PBL-logs' error status and error details on RMB-status registers

APPS PBL Error Log Format

140 Bytes	APPS PBL last error info x 5 (The APPS PBL will have 4 last error entries to be able to debug multiple error re-entrant case

```
/* Error log structure to store data describing error  */

typedef struct boot_pbl_err_type
{
   uint32                    pbl_err_code_start;
   uint32                    pbl_err_details;
   uint32                    timestamp;
   uint32                    pbl_id;
   uint32                    patch_id;
   const char*               filename;
   uint32                    line_num;
   uint32                    pbl_err_code_end;
} boot_pbl_err_type;
```



Details of PBL last error structure

Error code (4 bytes)
Error details (4 bytes)
Time stamp (4 bytes)
PBL ID (4 bytes)
Line number (4 bytes)

ERROR code details (4 bytes)

0xEFAABBCC
EF: Error signature or 7F: Pass signature
AA: PBL function block ID
BB: Sub-error in functional block
CC: Error count number, start from 0x1

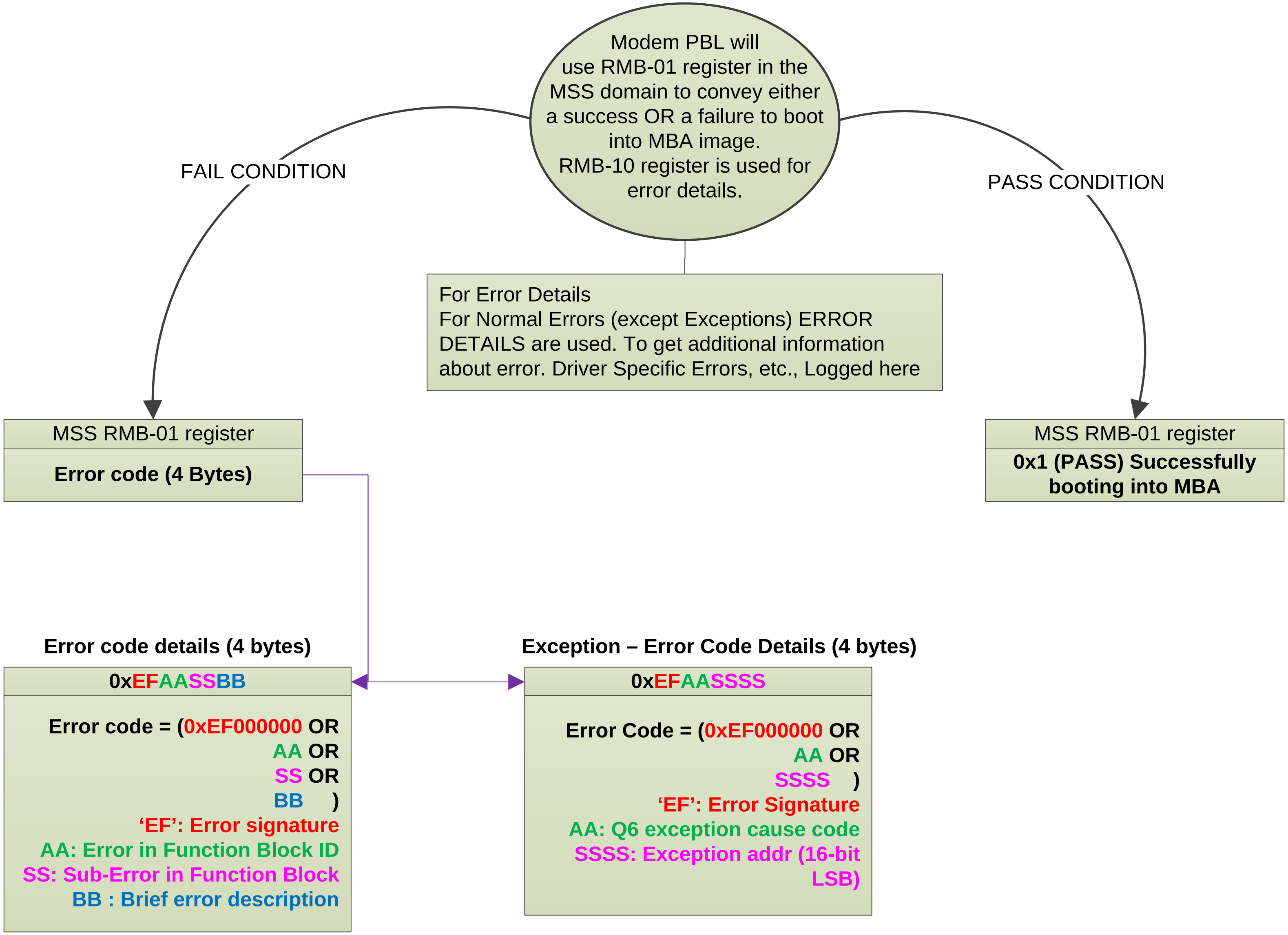
For Normal Errors (except Exceptions) ERROR DETAILS are used to get additional information about error. Driver specific errors are logged here

Current list of PBL function block IDs (128 max):
Refer next page
pbl_log__block_type

APPS PBL Error Code Definitions

```
typedef enum
{
  PBL_LOG_GENR           = 0x010000,
  PBL_LOG_PROC           = 0x020000,
  PBL_LOG_LOADER         = 0x030000,
  PBL_LOG_FUSE           = 0x040000,
  PBL_LOG_AUTH           = 0x050000,
  PBL_LOG_TIMER          = 0x060000,
  PBL_LOG_CLOCK          = 0x070000,
  PBL_LOG_SEC_HW         = 0x080000,
  PBL_LOG_SECBOOT        = 0x090000,
  PBL_LOG_SEC_IMG_AUTH   = 0x0A0000,
  PBL_LOG_SDCC           = 0x0B0000,
  PBL_LOG_SAHARA         = 0x0C0000,
  PBL_LOG_NAND           = 0x0D0000,
  PBL_LOG_PCIe           = 0x0E0000,
  PBL_LOG_UFS            = 0x0F0000,
  PBL_LOG_USB            = 0x100000,
  PBL_LOG_EXCEPTION      = 0x110000,
  PBL_LOG_ELF            = 0x120000,
  PBL_LOG_FORCE32BITS    = 0x7FFFFFFF /* To ensure it's 32 bits wide */
}pbl_log_block_type;
```

Modem PBL Error Log Format



Modem PBL will use RMB-01 register in the MSS domain to convey either a success OR a failure to boot into MBA image. RMB-10 register is used for error details.

FAIL CONDITION

PASS CONDITION

For Error Details
For Normal Errors (except Exceptions) ERROR DETAILS are used. To get additional information about error. Driver Specific Errors, etc., Logged here

MSS RMB-01 register

Error code (4 Bytes)

MSS RMB-01 register

0x1 (PASS) Successfully booting into MBA

Error code details (4 bytes)

0xEFAASSBB

Error code = (0xEF000000 OR
AA OR
SS OR
BB)
'EF': Error signature
AA: Error in Function Block ID
SS: Sub-Error in Function Block
BB : Brief error description

Exception – Error Code Details (4 bytes)

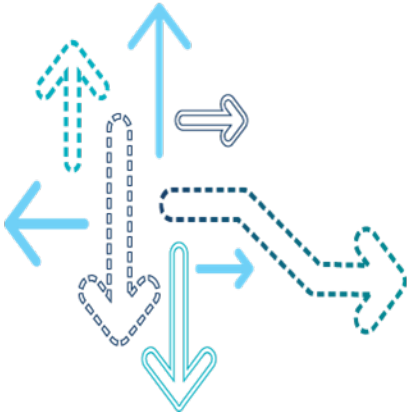
0xEFAASSSS

Error Code = (0xEF000000 OR
AA OR
SSSS)
'EF': Error Signature
AA: Q6 exception cause code
SSSS: Exception addr (16-bit LSB)

Fuses Used in PBL

Fuse bits eFuse	Domain	GPIO	Description
FAST_BOOT (4 bits)	OEM		Used by boot code to specify which device has priority to be booted from; this helps speed up the boot flow. However, in MSM8909 chipset, eMMC and NAND boot are mutually exclusive.
VID (16 bits)	OEM		USB vendor ID
PID (16 bits)	OEM		USB product ID
E_DLOAD_DISABLE	OEM		Disables emergency downloader
USB_ENUM_TIMEOUT	OEM		BOOT ROM must support USB enumeration timeout. Timeout applies to USB Download mode. If USB is not enumerated within time (90 sec), quit USB enumeration. If USB suspends or is disconnected after enumeration, start timer again.
OEM_PK_HASH	OEM		Blows the hash of OEM public key
FORCE_USB_BOOT	No fuse	1 pin, GPIO37	Forces boot from USB

Boot Loader Customization



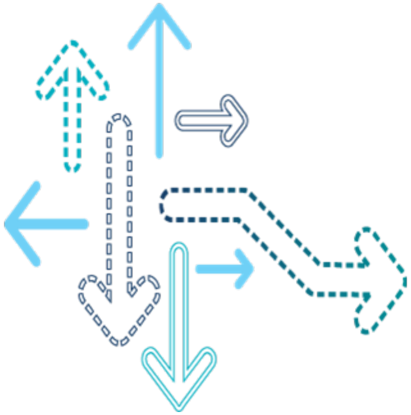
Boot Loader Customization

- If the OEM uses different memory parts than those used in the reference design, the boot loader needs to be updated
- For DDR memory
  - CDT in the .xml file needs to be reviewed and updated
  - DDR device type, mode (interleaved or not), density (rank/bank/row/col), and JEDEC specification default timing data
  - Retrieved by SBL1 and stored in shared IMEM (OCIMEM), and accessed by RPM
- Hardware platform ID
  - Defines PLATFORM_SUBTYPE based on the same chipset (MTP, CDP, etc.)
  - Controls the alternate functions on GPIOs
  - Stored in SMEM (DDR) and accessed by Little Kernel boot loader and the Linux kernel

Single Instance DDR Driver

- Eliminate 3 instances of DDR driver (SBL1, RPM_FW, and SDI) to 1 instance
- RPM CodeRAM size increased by 16 KB (144 KB) to host a SBL1 ELF segment which includes DDR driver.
- Since RPM (M3) and APPS (Cortex-A7) are part of ARMv7 family, both SBL1 and RPM_FW plug into the same driver instance loaded in RPM CodeRAM by PBL. SDI watchdog reset path also uses the same instance to bring DDR out of self refresh.
- Rationale
  - Scale DDR driver sequence and settings maintenance
  - Image size optimization

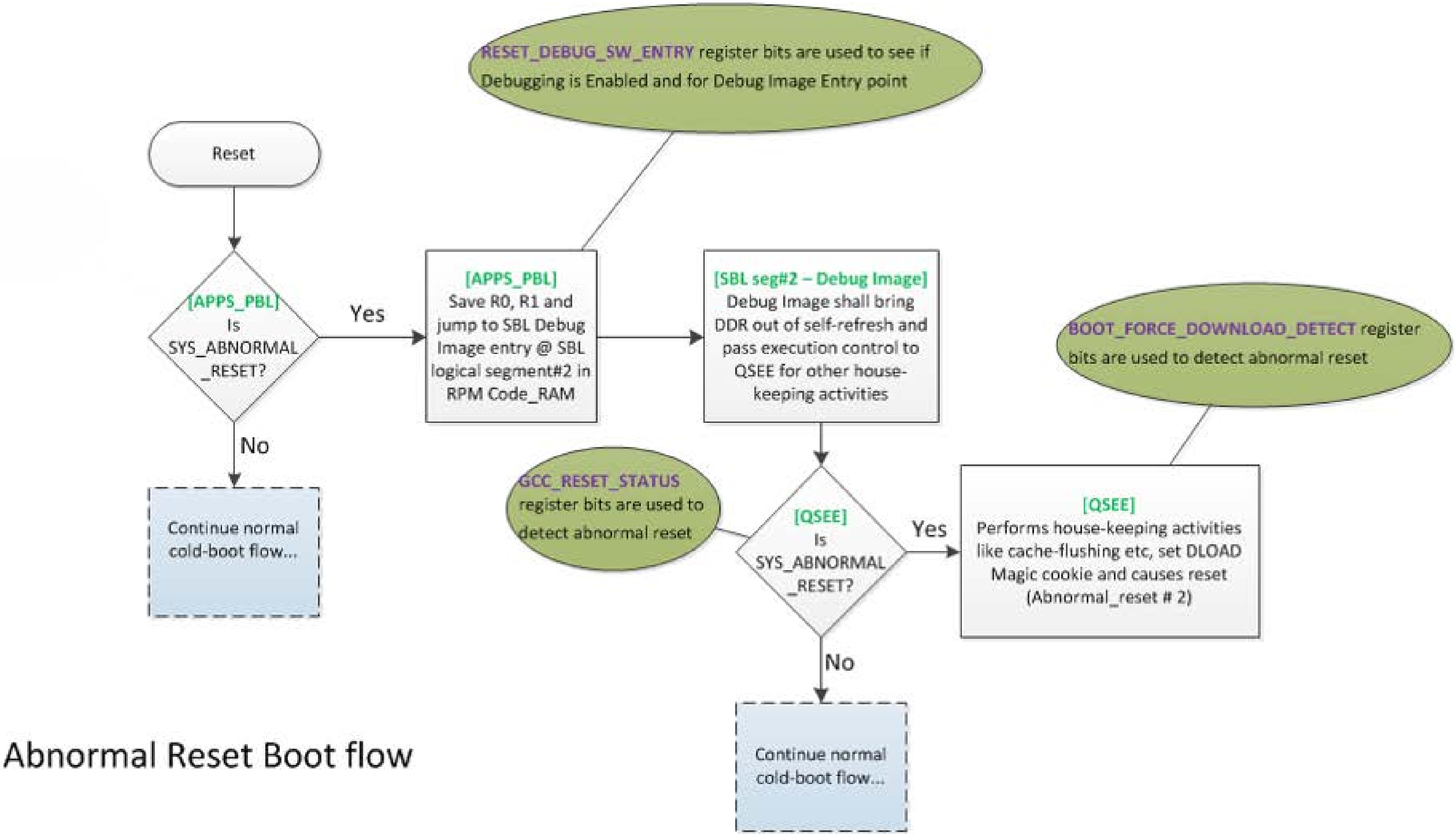
Watchdog Reset Debug



SDI Handling Changes Compared to MSM8x26, MSM8x10, and MSM8x12

- MSM8926
  - During cold boot, a separate SDI image is loaded by SBL1 into OCIMEM for non-production device. SDI brings DDR out of self refresh and flushes the L2/secure-cache/etc and other contents for watchdog/crash dump debug
  - PBL on watchdog reset jumps to OCIMEM fixed offset
- MSM8909 and MSM8916
  - No separate SDI image. Part (~20%) of SDI logic is in SBL1 RPM-code-ram segment and the remaining is in TZ image.
  - During cold boot, SDI logic part of TZ enables wdog-reset/SDI-path only for non-production device.
  - DDR handling logic, during SDI, reuses SBL1 DDR driver segment in RPM CodeRAM. All the L2/secure cache flush logic, during SDI, is reused from the TZ image.
  - PBL on watchdog reset jumps to SBl1 segment#2 of RPM CodeRAM fixed offset.

Watchdog Debug Flow Diagram


RESET_DEBUG_SW_ENTRY register bits are used to see if Debugging is Enabled and for Debug Image Entry point
Reset
[APPS_PBL] Is SYS_ABNORMAL _RESET?
Yes
No
[APPS_PBL] Save R0, R1 and jump to SBL Debug Image entry @ SBL logical segment#2 in RPM Code_RAM
[SBL seg#2 – Debug Image] Debug Image shall bring DDR out of self-refresh and pass execution control to QSEE for other house-keeping activities
BOOT_FORCE_DOWNLOAD_DETECT register bits are used to detect abnormal reset
Continue normal cold-boot flow...
GCC_RESET_STATUS register bits are used to detect abnormal reset
[QSEE] Is SYS_ABNORMAL _RESET?
Yes
[QSEE] Performs house-keeping activities like cache-flushing etc, set DLOAD Magic cookie and causes reset (Abnormal_reset # 2)
No
Continue normal cold-boot flow...
Abnormal Reset Boot flow

Watchdog Debug Flow

- Cold boot
  - APPS PBL → SBL1 → QSEE → RPM_FW* → APPSBL.

    *Logically QSEE transfers the control to APPSBL (not RPM_FW).
- Watchdog reset (first reset)
  - APPS PBL → SBL1 segment #2, bring DDR out of self-refresh → QSEE
  - QSEE flushes the caches, saves on-chip debug buffer, before forcing second reset via PS_HOLD drop
- Second reset
  - APPS PBL → SBL1 → Sahara download mode
  - SBL1 supports default memory dump feature to dump the DDR via USB for non-production device.

References

Ref.	Document	
Qualcomm Technologies		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*MSM8916/MSM8936/MSM8939 Boot Architecture Overview*	80-NL239-1

Questions?

https://support.cdmatech.com